Table of Contents

- Table of Contents
  - Kubernetes Terminology
  - Kubernetes Architecture
  - Kubernetes Interactive Labs
  - Kubernetes Objects
    - Pods
    - ReplicaSets
      - Scaling Replicaset
    - Deployments
    - Namespaces
      - Kubernetes Namespaces Concepts
    - Kubernetes Service
      - NodePort
      - ClusterIP
      - LoadBalancer
  - Kubernetes Service vs Deployment
  - Reference
    - Kubernetes Cluster Setup

Kubernetes Terminology

- **Nodes**: Hosts that run Kubernetes applications/containers
- **Containers**: Units of packaging application
- **Pods**: Units of deployment which is collection of containers
- **Replication Controller**: Ensures availability and scalability
- **Labels**: Key-value pairs for identification (similar to tags)
- **Services**: Collection of pods exposed as an endpoint

--

Kubernetes Architecture

- **Master** controls the cluster, and the nodes in it. It ensures the execution only happens in nodes and co-ordinates the container execution on nodes.
- **Nodes** host the containers; in-fact these Containers are grouped logically to form **Pods**.
- Each node can run multiple such **Pods**, which are a group of containers, that interact with each other, for a **deployment**.
- **Replication Controller** is Master's resource to ensure that the requested no. of pods are always running on nodes.
- **Replica Set**: replica sets are created by deployment, these deployments contains declaration of containers which you want to run in cluster like image/tag, env variable, data volumes Kubernetes has several components in its architecture.
- **Service** is an object on Master that provides load balancing across a replicated group of Pods.

- **POD**: POD is the smallest unit of deployment in K8S object. Each POD then contains the Docker containers. Each POD can host a different set of Docker containers. The proxy is then used to control the exposing of these services to the outside world. PODs are created by replicaset.

--

- **etcd** – This component is a highly available `key-value` store that is used for storing shared configuration and service discovery.
- **kube-apiserver** – This is an API which can be used to orchestrate the Docker containers.
- **kube-controller-manager** – This is used to control the Kubernetes services.
- **kube-scheduler** – This is used to schedule the containers on hosts.
- **Kubelet** – This is used to control the launching of containers via manifest files from worker host. (which talks with K8S cluster).
- **kube-proxy** – This is used to provide network proxy services to the outside world.

Kubernetes Interactive Labs

- Create an account with DockerHub and Sign-In
- Navigate to Play with Kubernetes
- Click on **Add New Instance** on the left side of the screen to bring up an instance on the right side.
- Perform the steps mentioned in the terminal.
    - Copy the commands using **Ctrl+Insert** and Paste the commands using **Shift+Insert**
    - To maximize the window use **Alt+Enter**

```
# Initializes cluster master node:
kubeadm init --apiserver-advertise-address $(hostname -i) --pod-network-cidr
10.5.0.0/16

# Initialize cluster networking:
kubectl apply -f https://raw.githubusercontent.com/cloudnativelabs/kube-
router/master/daemonset/kubeadm-kuberouter.yaml

# To get node information
kubectl get nodes

# (Optional) Create an nginx deployment:
# kubectl apply -f
https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/ap
plication/nginx-app.yaml
```

--

- To add Worker Node: Click on **Add New Instance**
- Copy the output of 1st command from 1st instance into the New Instance

```
kubeadm join 192.168.0.13:6443 --token n8lvoe.2h1ubmjyfmcxr00s \
    --discovery-token-ca-cert-hash
```

```
sha256:089afdc26c7908fd5d0ec54d9e812a85ac6dc194457968961ccaa19d8f7cb7a3
```

- Execute the commands on the Control Plane Node.

```
kubectl get nodes
```

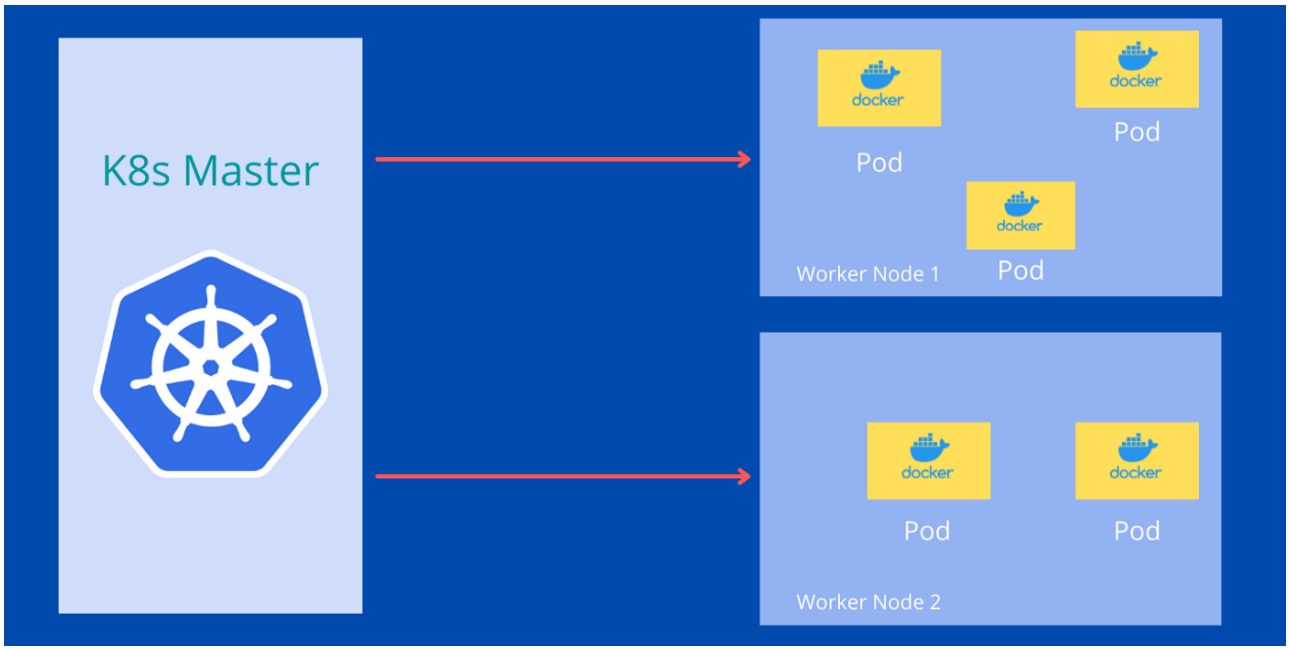
Kubernetes Objects

- Pods
- ReplicaSets
- Deployments
- Namespaces
- Service
- Configmap
- Secrets

Pods

- Kubernetes doesn't deploy containers directly on the worker node.
- Kubernetes is designed to handle single or multiple containers. Containers are encapsulated into a Kubernetes object.That Kubernetes object is called **Pod**.
    - If we have to run a Docker container in the Kubernetes cluster then we need a Pod object.
- A Pod is the smallest and simplest Kubernetes object.
- The Pod is one of the Kubernetes components and the smallest unit in the Kubernetes cluster.
- A group of one or more containers is called a Pod.
- Containers in a Pod are deployed together, and are started, stopped, and replicated as a group.
- Each Pod has only 1 IP, irrespective of number of containers.
- All container in a Pod shares IP, cgroups, namespaces, localhost adapter, volumes every pod can interact directly with other pod via Pod N/W (Inter-Pod communication)
- It is the unit of deployment in Kubernetes, which represents a single instance of the application.

--

- 

--

- Create a Pod with **Imperative Command**.

```
# Imperative Command is simply a kubectl command line with options to create
objects.
# Syntax
kubectl run pod-name --image image_name:tag

kubectl run nginx-pod --image nginx:1.16.2
# kubectl run => Standard command to run a Pod.
# nginx-pod => Name of the Pod. You can give any name.
# --image nginx:1.16.2 => This is the Docker image that will be used to launch the
Container in the Pod.
kubectl get pods

kubectl get pods -o wide

kubectl describe pod nginx-pod

kubectl run new-nginx-pod --image nginx:1.16.1

# View Pods in the cluster
kubectl get pods

# Check for details of the pod
kubectl describe pod new-nginx-pod
```

--

- Create a Pod with **Declarative Command**, where Kubernetes objects can be created, updated, and deleted by storing multiple object configuration files in a directory and using kubectl apply to recursively create and update those objects as needed.

pod-definition.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: testapp-pod
  labels:
     app: testapp-pod
     type: front-end-pod
spec:
  containers:
  - name: nginx-container
    image: nginx:1.16.1
```

- Execute below commands to applly and validate the YAML manifest file.

```
kubectl get pods -o wide

kubectl create -f pod-definition.yaml
OR
kubectl apply -f pod-definition.yaml


# Get the list of running Pods using the below command
kubectl get pods
# OUTPUT
NAME        READY   STATUS    RESTARTS   AGE
testapp-pod   1/1     Running   0          10s

# Check for details of the pod
kubectl describe pod testapp-pod

# List pods with wide option which also provide Node information on which Pod is
running.
kubectl get pods -o wide

# Login inside the pod using below command
kubectl exec -it testapp-pod /bin/bash

# In your shell, list the root directory:
root@testapp-pod:/# ls /
bin   dev                 docker-entrypoint.sh  home  lib64  mnt  proc  run   srv
tmp  var
boot  docker-entrypoint.d  etc                   lib   media  opt  root  sbin  sys
usr

# You can run these example commands inside the container

ls /
apt-get update
```

```
apt-get install -y procps curl
ps aux
ps aux | grep nginx

# Modifying the root page for nginx Webserver
curl http://localhost/
cat /usr/share/nginx/html/index.html
echo 'Content inside NGINX File' > /usr/share/nginx/html/index.html

# To quit the shell in the container and go back to host machine
exit

$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
testapp-pod   1/1     Running   0          6m10s

# Running individual commands in a container
kubectl exec testapp-pod -- printenv
kubectl exec testapp-pod -- ls /

# Print the logs for a container in a pod
kubectl logs -f testapp-pod

# Delete all Evicted Pods
kubectl get pod  | grep Evicted | awk '{print $1}' | xargs kubectl delete pod
```

ReplicaSets

- **ReplicaSet** is a way to setup replication of pods.
- ReplicaSet Definition File

`replicaset-definition.yaml`

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: testapp-replicaset
  labels:
    app: testapp
    type: front-end
spec:
  replicas: 3
  selector:
    matchLabels:
      type: front-end
  template:
    metadata:
      name: testapp-pod
      labels:
```

```
        app: testapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

--

- Execute below commands:

```
# To apply the replicaset
kubectl apply -f replicaset-definition.yaml
$ kubectl get pods
# OUTPUT
NAME                        READY   STATUS    RESTARTS   AGE
testapp-replicaset-5bfml    1/1     Running   0          14s
testapp-replicaset-9m58f    1/1     Running   0          14s
testapp-replicaset-g962z    1/1     Running   0          14s
$ kubectl get replicaset
$ kubectl get rs
# OUTPUT
NAME                 DESIRED   CURRENT   READY   AGE
testapp-replicaset   3         3         3       17s

$ kubectl delete pod testapp-replicaset-5bfml
pod "testapp-replicaset-5bfml" deleted

$ kubectl get rs
NAME                 DESIRED   CURRENT   READY   AGE
testapp-replicaset   3         3         3       116s

$ kubectl describe rs testapp-replicaset

$ kubectl get pods
# OUTPUT
NAME                        READY   STATUS    RESTARTS   AGE
testapp-replicaset-57rmw    1/1     Running   0          26s
testapp-replicaset-9m58f    1/1     Running   0          118s
testapp-replicaset-g962z    1/1     Running   0          118s

$ kubectl describe pod testapp-replicaset-57rmw
```

--

Scaling Replicaset

- There are multiple ways to scale replicaset:
    - 1. Update the number of replicas in the **replicaset-definition.yaml** definition file as **replicas: 5** and Re-apply the same file.

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

- 2. Use **kubectl scale** command with manifest yaml file.

```
kubectl scale --replicas=2 -f replicaset-definition.yaml
```

- 3. Use **kubectl scale** command with type and name.

```
kubectl scale --replicas=3 replicaset testapp-replicaset
```

Deployments

- Create Deployment with Imperative Command.

```
$ kubectl get deployments

$ kubectl create deployment hello-node --image=k8s.gcr.io/echoserver:1.4

$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
hello-node   1/1     1            1           48s

$ kubectl describe deployment hello-node


$ kubectl get rs
NAME                    DESIRED   CURRENT   READY   AGE
hello-node-7567d9fdc9   1         1         1       50s
$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-node-7567d9fdc9-cw8n5   1/1     Running   0          55s
$ kubectl describe pod hello-node-7567d9fdc9-cw8n5
Name:         hello-node-7567d9fdc9-cw8n5
Namespace:    default
Priority:     0
Node:         minikube/10.0.0.7
Start Time:   Tue, 26 Jul 2022 02:35:26 +0000
Labels:       app=hello-node
              pod-template-hash=7567d9fdc9
Annotations:  <none>
Status:       Running
IP:           172.18.0.6
IPs:
  IP:           172.18.0.6
Controlled By:  ReplicaSet/hello-node-7567d9fdc9
```

```
Containers:
  echoserver:
    Container ID:
docker://837d38e387b4d44eaa04b87ca00344a07e3170e4aa0f2f9df97edc22a0a5cfed
    Image:          k8s.gcr.io/echoserver:1.4
    Image ID:       docker-
pullable://gcr.io/google_containers/echoserver@sha256:5d99aa1120524c801bc8c1a7077e
8f5ec122ba16b6dda1a5d3826057f67b9bcb
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Tue, 26 Jul 2022 02:35:28 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-4vfrr (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-4vfrr:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-4vfrr
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  2m3s  default-scheduler  Successfully assigned default/hello-
node-7567d9fdc9-cw8n5 to minikube
  Normal  Pulling    2m2s  kubelet            Pulling image
"k8s.gcr.io/echoserver:1.4"
  Normal  Pulled     2m1s  kubelet            Successfully pulled image
"k8s.gcr.io/echoserver:1.4" in 487.451376ms
  Normal  Created    2m1s  kubelet            Created container echoserver
  Normal  Started    2m1s  kubelet            Started container echoserver

kubectl explain deployment
```

--

- Create deployment with **Declarative** Command, Download or use below command to apply the Deployment

```
```

```
kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml
```

nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- **kind** - What kind of object you want to create.
- **metadata** - Data that helps uniquely identify the object, including a **name** string, **label**.
- **spec.replicas** — Tells Kubernetes how many pods to create during a deployment. Modifying this field is an easy way to scale a containerized application.
- **spec.selector** — An optional object that tells the Kubernetes deployment controller to only target pods that match the specified labels.
- **spec.template.metadata.labels** — Adds labels to a deployment specification.

--

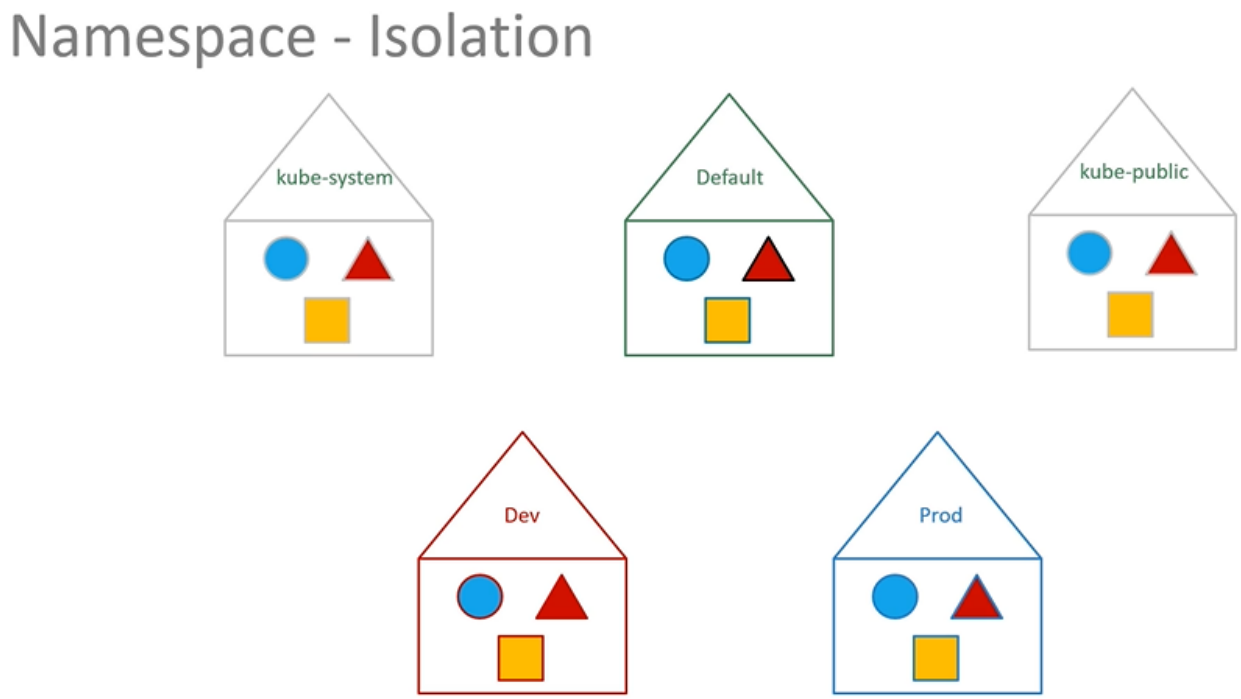
Execute Commands : https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#scaling-a-deployment

---

Namespaces

- A Kubernetes namespace helps separate a cluster into logical units.
- In a new cluster, Kubernetes automatically creates the following namespaces: `default` (for user workloads) and for the Kubernetes control plane: `kube-system`.

- Kubernetes also allows admins to manually create custom namespaces.



-

Kubernetes Namespaces Concepts

- There are two types of Kubernetes namespaces: Kubernetes system namespaces and custom namespaces.
- **default** - a default space for objects that do not have a specified namespace.
- **kube-system** - a default space for Kubernetes system objects, such as kube-dns and kube-proxy, and add-ons providing cluster-level features, such as web UI dashboards, ingresses, and cluster-level logging.

--

- Use this command to list all the available namespaces in your environment.

```
kubectl get namespaces

kubectl get pods -n default
kubectl get pods -n kube-system

kubectl create namespace development

kubectl get namespaces

kubectl get pods -n development

# Launch a new pod in this newly created namespace
kubectl run new-nginx-pod --image nginx:1.16.1 -n development
```

```
kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml -n
development

kubectl get deployments -n development
kubectl get pods -n development
kubectl get pods -n default
kubectl get pods

# Delete all pods in namespace
kubectl delete pods --all -n development

# To delete a namespace.
kubectl delete namespace development
```

> Note: Kubernetes will always list the pods from the **default** namespace. you need to specify the namespace name to display the objects in it.

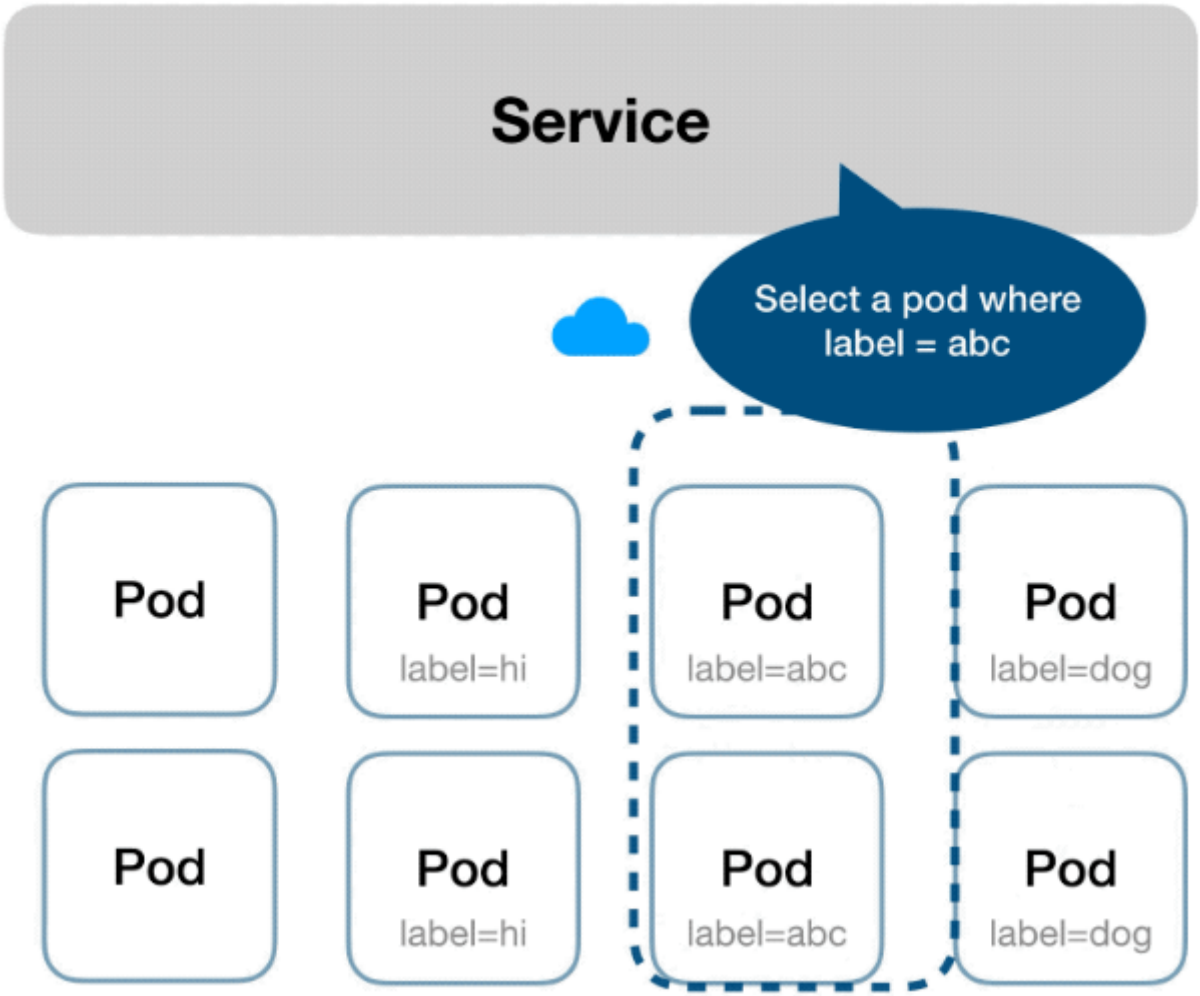
Kubernetes Service

- In the Kubernetes world, the pods, where the application lives, are temporary and get a new IP address every time they are launched.
- The pods are usually dynamically destroyed and recreated with each deployment.
- In the absence of the Kubernetes service, we would have to track the IP addresses of all active pods.
- The **Kubernetes service** creates an abstraction that maps to one or more pods.
- This abstraction allows other applications to reach the service by simply referring to the service name.
- It means that other applications no longer need to know the IP addresses assigned to the pods.
- External applications and end-users can also access the services assuming that they are exposed publicly to the internet.
- Kubernetes Services enables communication between various components within and outside of the application Container Pods
- A Service enables network access to a set of Pods in Kubernetes.

--

- **Services** match a set of Pods using **labels** and **selectors**.
- Labels are key/value pairs attached to objects and can be used in any number of ways:
    - Designate objects for **development, test, and production**
    - Classify an object using **tags**
- When a network request is made to the **service**, it selects all **Pods** in the cluster matching the **service's selector**, chooses one of them, and forwards the network request to it.

--


Service
Select a pod where
label = abc
Pod
Pod
label=hi
Pod
label=abc
Pod
label=dog
Pod
Pod
label=hi
Pod
label=abc
Pod
label=dog

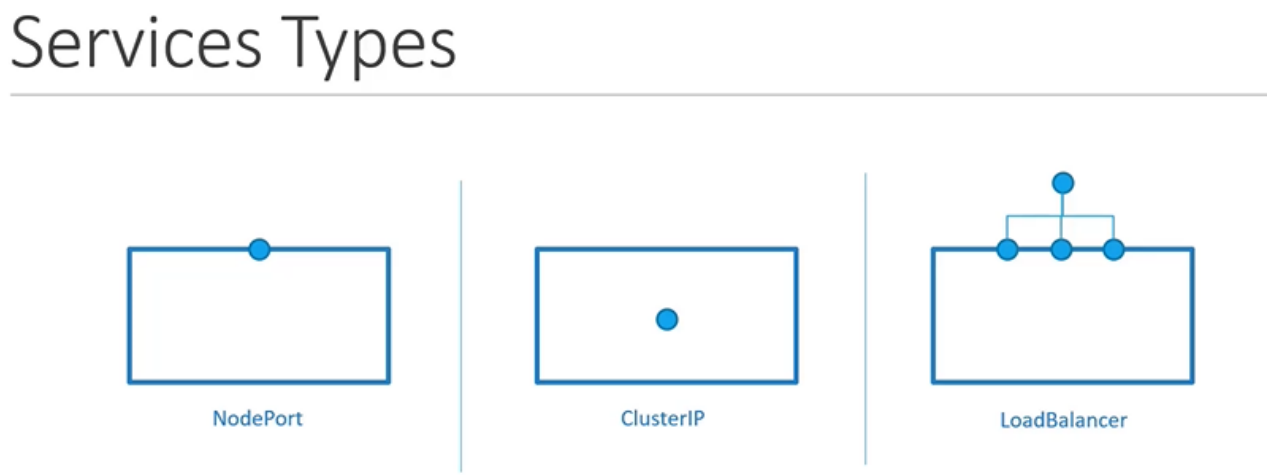
- --


Network
Request

-

--

- There are 3 types of service types in kubernetes.



--

NodePort

- Exposes the Service on each Node's IP at a static port (the NodePort).
- If is possible to contact the NodePort Service, from outside the cluster, by requesting **NodeIP:NodePort**
- A **ClusterIP** Service, to which the NodePort Service routes, is automatically created.
- In NodePort, the service makes an internal POD accessible on a PORT on the NODE.

--

`nginx-deployment-definition.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  # Create 3 replicas of the Pods.
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
```

```
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

service-definition.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  # Expose the service on a static port on each node
  # so that we can access the service from outside the cluster
  type: NodePort

  # When the node receives a request on the static port (30163)
  # "select pods with the label 'app' set to 'echo-hostname'"
  # and forward the request to one of them
  selector:
    app: nginx

  ports:
    # Three types of ports for a service
    # nodePort - a static port assigned on each the node
    # port - port exposed internally in the cluster
    # targetPort - the container port to send requests to
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

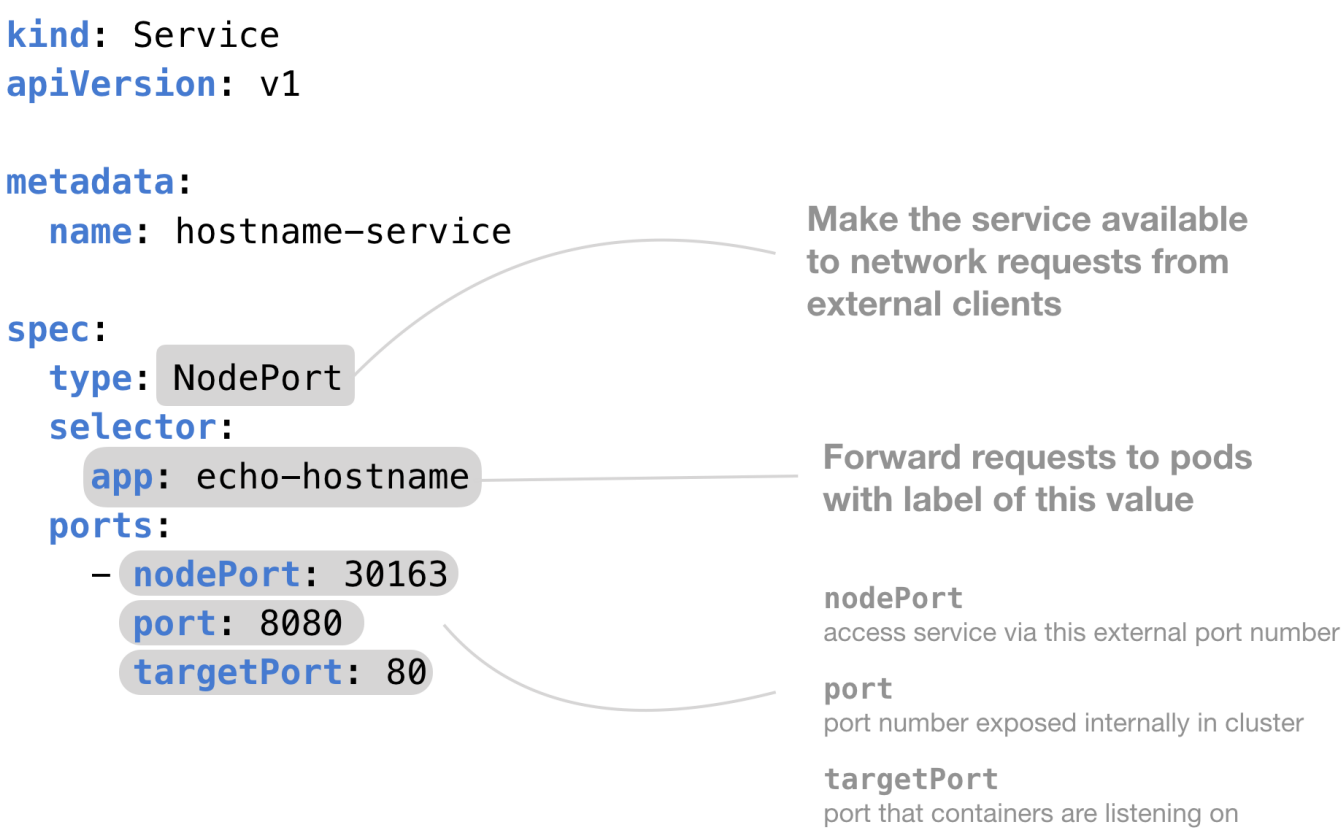
--

- Use **kubectl apply** to apply both above YAML Manifest Files

```
kubectl apply -f nginx-deployment-definition.yaml
kubectl get pods -o wide
kubectl apply -f service-definition.yaml
kubectl get svc
# OUTPUT
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes      ClusterIP   10.96.0.1      <none>        443/TCP          24m
nginx-service   NodePort    10.101.248.0   <none>        8080:30163/TCP   6m20s

# Access the Port 30163 in the View Port

$ kubectl describe service nginx-service
# OUTPUT
Name:                     nginx-service
```

```
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP Families:              <none>
IP:                       10.107.15.76
IPs:                      10.107.15.76
Port:                     <unset>  8080/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30163/TCP
Endpoints:                10.5.1.6:80,10.5.1.7:80,10.5.1.8:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>

# The Endpoints in the above Service Description are the POD IPs.
# Each time page is refreshed, the response of the WebPage is from any one of the
POD that is running with Label "app: nginx"

$ kubectl describe endpoints nginx-service
# OUTPUT
Name:         nginx-web-service
Namespace:    default
Labels:       <none>
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2023-04-
03T19:35:42Z
Subsets:
  Addresses:          10.5.1.6,10.5.1.7,10.5.1.8
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>

# Login inside one of the Pod to modify the /usr/share/nginx/html/index.html
kubectl get pods -o wide --show-labels
kubectl exec -it nginx-deployment-66b6c48dd5-kjlws /bin/bash

root@nginx-deployment-66b6c48dd5-2k7w8:/# cd /usr/share/nginx/html/
root@nginx-deployment-66b6c48dd5-2k7w8:/# ls
root@nginx-deployment-66b6c48dd5-2k7w8:/# echo "Modified File inside one POD with
Hostname as $HOSTNAME" > index.html
root@nginx-deployment-66b6c48dd5-2k7w8:/# exit

# Access the Port 30163 in the View Port/Refresh the page multiple times.
curl localhost:30163
curl localhost:30163
curl localhost:30163
# The Service will forward request to the Pod for which above Page is modified.
```

--



--

ClusterIP

- By default, Kubernetes creates a **ClusterIP** type of service. We can build different kinds of services by having a **spec.type** property in the service YAML file.
- Exposes the Service on a cluster-internal IP.
- Choosing this value makes the Service only reachable from within the cluster. This is the default **ServiceType**.

pod-service-definition.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-web-deployment
  labels:
    app: nginx-web
spec:
  # Create 3 replicas of the Pods.
  replicas: 3
  selector:
    matchLabels:
      app: nginx-web
  template:
```

```
    metadata:
      labels:
        app: nginx-web
    spec:
      containers:
      - name: nginx-web
        image: nginx:1.14.2
        ports:
        - containerPort: 80

# ---

apiVersion: v1
kind: Service
metadata:
  name: nginx-web-service
spec:
  # Expose the service on a static port on each node
  # so that we can access the service from outside the cluster
  type: ClusterIP

  # When the node receives a request on the static port (30163)
  # "select pods with the label 'app' set to 'echo-hostname'"
  # and forward the request to one of them
  selector:
    app: nginx-web

  ports:
    # Three types of ports for a service
    # port - port exposed internally in the cluster
    # targetPort - the container port to send requests to
    - port: 8080
      targetPort: 80
```

```
kubectl apply -f pod-service-definition.yaml
# OUTPUT
deployment.apps/nginx-web-deployment unchanged
service/nginx-web-service configured

kubectl get pods -o wide
$ kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE     IP
NODE       NOMINATED NODE   READINESS GATES
nginx-web-deployment-7949b98ff9-9hhzg   1/1     Running   0          9m13s
172.18.0.10   minikube   <none>           <none>
nginx-web-deployment-7949b98ff9-ht64b   1/1     Running   0          9m13s
172.18.0.9    minikube   <none>           <none>
nginx-web-deployment-7949b98ff9-s9qjh   1/1     Running   0          9m13s
172.18.0.11   minikube   <none>           <none>

$ kubectl get svc
```

```
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP
54m
nginx-web-service   ClusterIP   10.108.233.187   <none>        8080/TCP
8m58s

$ kubectl describe service nginx-web-service
Name:              nginx-web-service
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=nginx-web
Type:              ClusterIP
IP Families:       <none>
IP:                10.108.233.187
IPs:               10.108.233.187
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         172.18.0.10:80,172.18.0.11:80,172.18.0.9:80
Session Affinity:  None
Events:            <none>

curl 10.108.233.187:8080
curl 10.108.233.187:8080
```

--

LoadBalancer

- This service type creates load balancers in various Cloud providers like AWS, GCP, Azure, etc., to expose our application to the Internet.
- The Cloud provider will provide a mechanism for routing the traffic to the services. The most common example usage of this type is for a website or a web app.
- On cloud providers which support external load balancers, setting the type field to LoadBalancer provisions a load balancer for your Service.

pod-service-definition.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-web-service
spec:
  # Expose the service on a static port on each node
  # so that we can access the service from outside the cluster
  type: LoadBalancer

  # When the node receives a request on the static port (30163)
  # "select pods with the label 'app' set to 'echo-hostname'"
```

```
    # and forward the request to one of them
    selector:
      app: nginx-web

    ports:
      # Three types of ports for a service
      # port - port exposed internally in the cluster
      # targetPort - the container port to send requests to
      - protocol: TCP
        port: 8080
        targetPort: 80
```

- Traffic from the external load balancer is directed at the backend Pods. The cloud provider decides how it is load balanced.

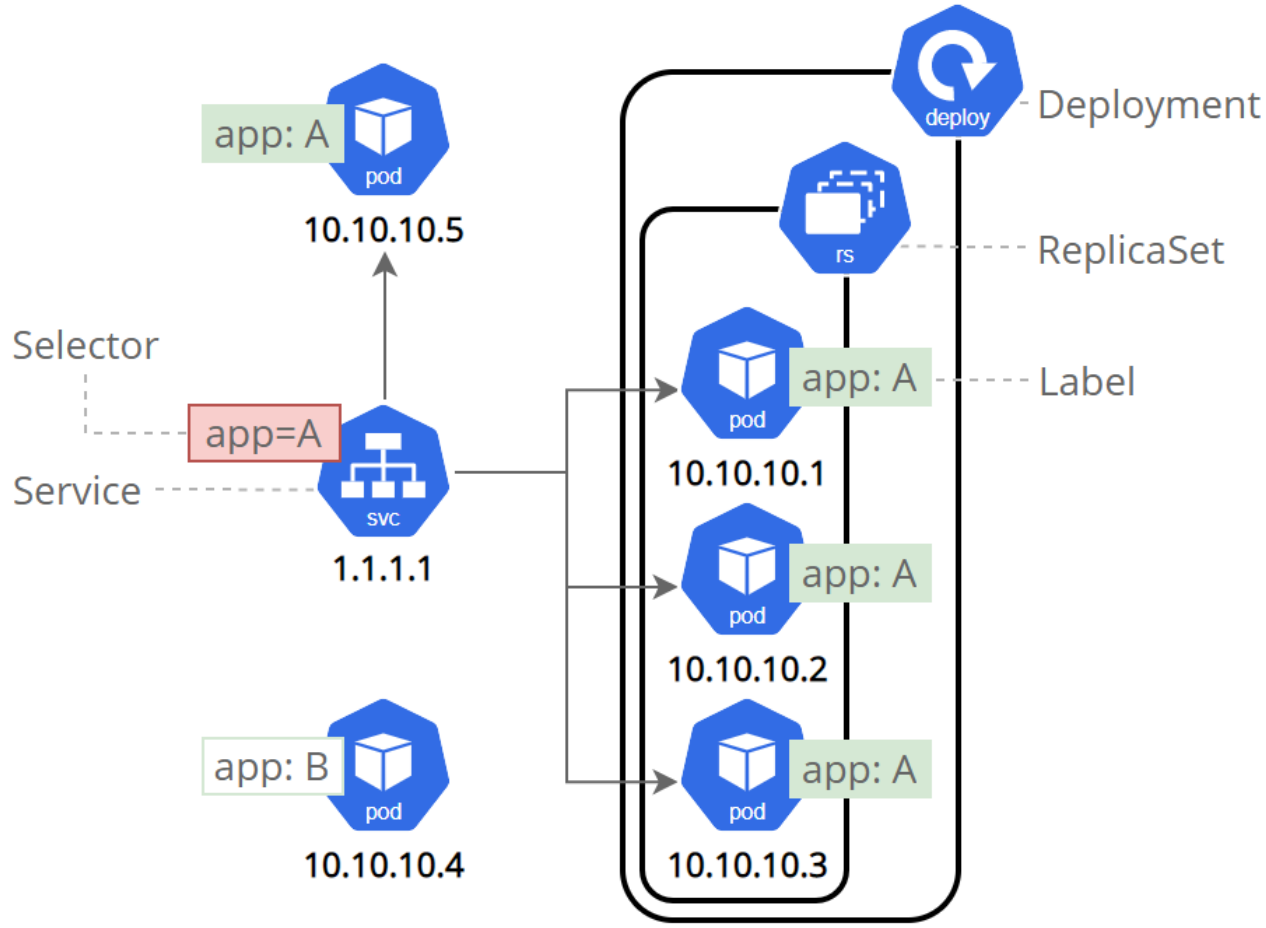
--

Kubernetes Service vs Deployment

- A deployment is responsible for keeping a set of pods running.
- A service is responsible for enabling network access to a set of pods.
- A deployment without a service can be created to keep a set of identical pods running in the Kubernetes cluster.
- The deployment could be scaled up and down and pods could be replicated.
- Each pod could be accessed individually via direct network requests (rather than abstracting them behind a service), but keeping track of this for a lot of pods is difficult.
- Services and Deployments are different, but they work together to route network requests across pods.

--

- Kubernetes Objects have labels and selectors as below:



Reference

Kubernetes Cluster Setup

- Minikube – Development and Learning
- Kops – Learning, Development, Production
- Kubeadm – Learning, Development, Production
- Docker for Mac - Learning, Development
- Kubernetes IN Docker - Learning, Development