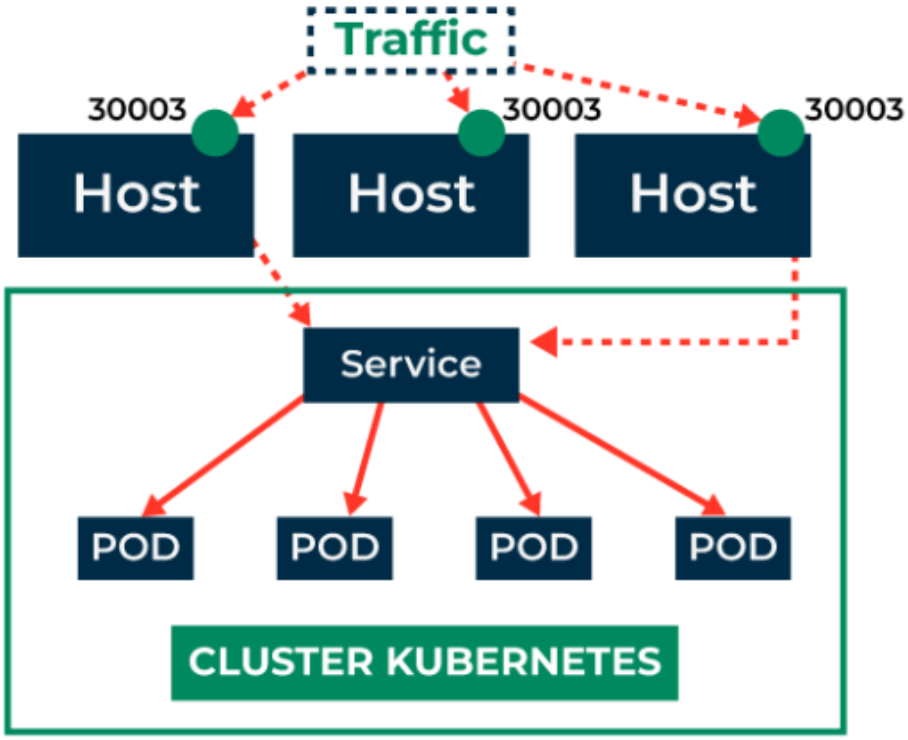
Kubernetes Services

Service

Mainly we 3 types of services.

1. **NodePort**
2. **ClusterIP**
3. **LoadBalancer**

1. NodePort



Let's create a service

vim service.yaml

```
apiVersion: v1
kind: Service
metadata:
 name: mysvc
spec:
 selector:
  app: webserver
 type: NodePort
 ports:
  - targetPort: 80
    port: 80
    nodePort: 30080
```

kubectl create -f service.yaml

```
root@master-node:~# vim service.yaml
root@master-node:~# kubectl create -f service.yaml
service/mysvc created
root@master-node:~#
root@master-node:~# kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        10d
mysvc        NodePort    10.100.161.39   <none>        80:30080/TCP   17s
root@master-node:~#
```

kubectl describe svc mysvc

```
root@master-node:~# kubectl describe svc mysvc
Name:                     mysvc
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=webserver
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.100.161.39
IPs:                      10.100.161.39
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30080/TCP
Endpoints:
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

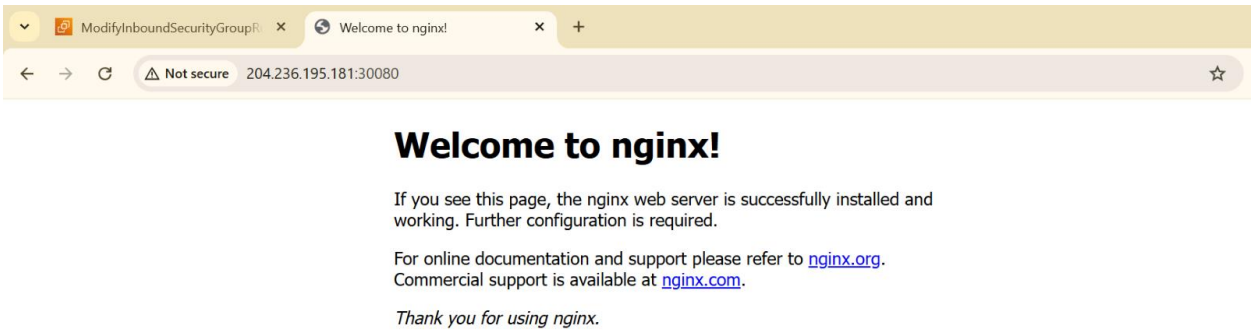
kubectl create -f deployment.yaml

```
root@master-node:~# kubectl create -f deployment.yaml
deployment.apps/mywebsite created
root@master-node:~#
root@master-node:~#
root@master-node:~# kubectl get deployments.apps
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
mywebsite   50/50   50           50          29s
root@master-node:~#
```

Now access our application using worker node public ip and 30080 port

Go browser and access

http://worker-node-ip/30080



Ok now let's update image in deployment file and replace deployment with our custom image.

```
vim deployment.yaml
```

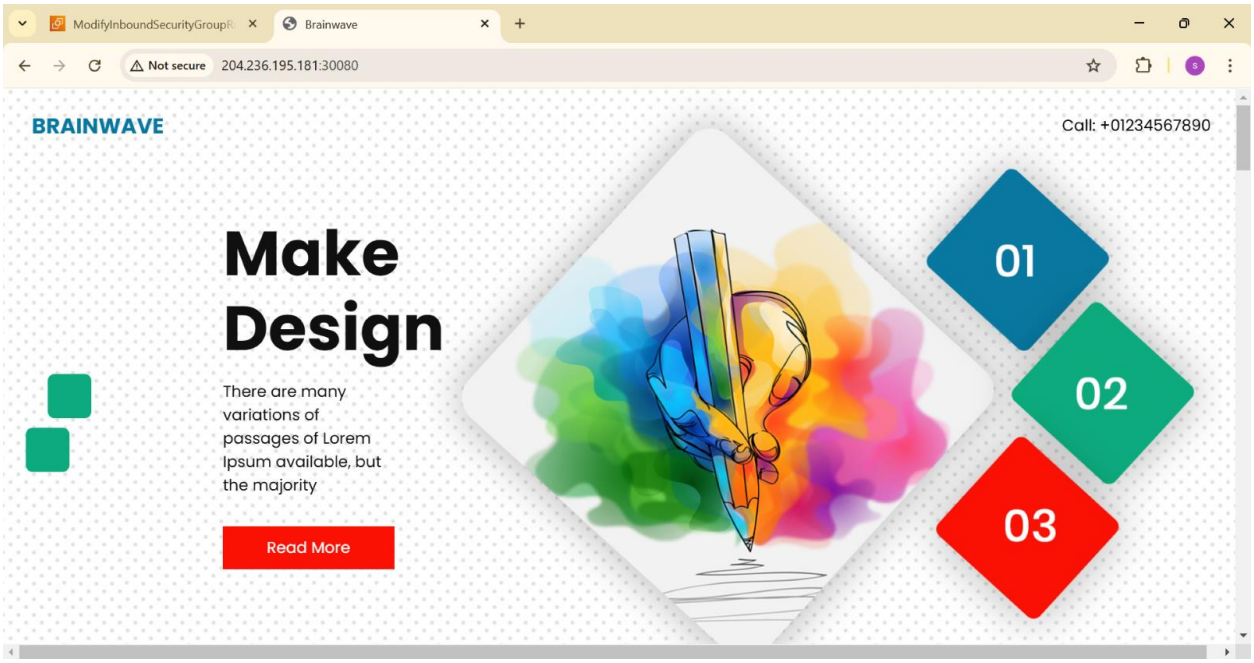
```yaml
kind: Deployment
metadata:
  name: mywebsite
spec:
  selector:
    matchLabels:
      app: webserver
  replicas: 50
  template:
    metadata:
      name: xyz
      labels:
        app: webserver
    spec:
      containers:
        - name: sdfdf
          image: jacksneel/centos-web
```

```
kubectl replace -f deployment.yaml
```

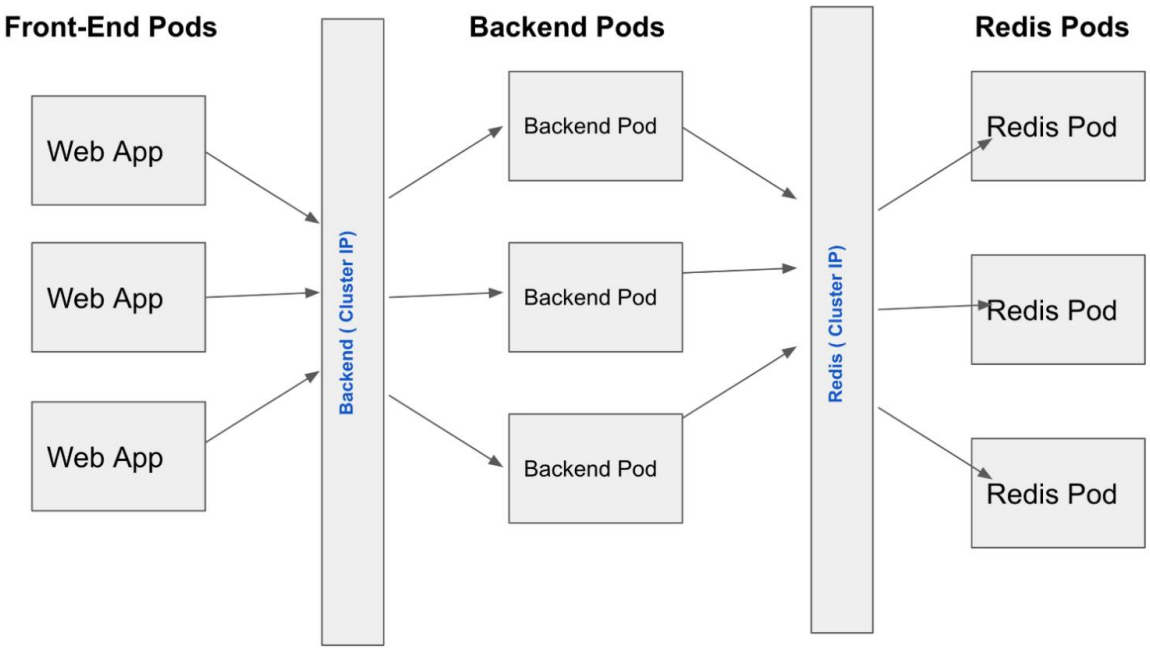
```
root@master-node:~# vim deployment.yaml
root@master-node:~# kubectl replace -f deployment.yaml
deployment.apps/mywebsite replaced
root@master-node:~#
root@master-node:~# kubectl get deployments.apps
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
mywebsite   38/50   37           38          11m
root@master-node:~#
```

Let's access application

http://worker-node-ip/30080



2. ClusterIP



Lets create a ClusterIP service for exiting pods

```
vim cluster.yml
```

```
apiVersion: v1
kind: Service
metadata:
 name: myip
spec:
 selector:
   app: webserver
 type: ClusterIP
 ports:
  - targetPort: 80
    port: 80
```

```
kubectl create -f cluster.yml
```

```
root@master-node:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        10d
myip         ClusterIP   10.97.33.109    <none>        80/TCP         7s
mysvc        NodePort    10.100.161.39   <none>        80:30080/TCP   30m
root@master-node:~#
```

```
root@master-node:~#
root@master-node:~# curl 10.97.33.109
<!DOCTYPE html>
<html lang="en">

<head>
  <!-- Basic -->
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <!-- Mobile Metas -->
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-f
it=no" />
  <!-- Site Metas -->
```

3. LoadBalancer

Exposes the Service externally using an external load balancer. Kubernetes does not directly offer a load balancing component; you must provide one, or you can integrate your Kubernetes cluster with a cloud provider.

type: LoadBalancer

On cloud providers which support external load balancers, setting the `type` field to `LoadBalancer` provisions a load balancer for your Service. The actual creation of the load balancer happens asynchronously, and information about the provisioned balancer is published in the Service's `.status.loadBalancer` field. For example:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
```

```
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  clusterIP: 10.0.171.239
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.0.2.127
```

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

To implement a Service of type: LoadBalancer, Kubernetes typically starts off by making the changes that are equivalent to you requesting a Service of type: NodePort. The cloud-controller-manager component then configures the external load balancer to forward traffic to that assigned node port.

You can configure a load balanced Service to omit assigning a node port, provided that the cloud provider implementation supports this.

Some cloud providers allow you to specify the loadBalancerIP. In those cases, the load-balancer is created with the user-specified loadBalancerIP. If the loadBalancerIP field is not specified, the load balancer is set up with an ephemeral IP address. If you specify a loadBalancerIP but your cloud provider does not support the feature, the loadbalancerIP field that you set is ignored.

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

To implement a Service of type: LoadBalancer, Kubernetes typically starts off by making the changes that are equivalent to you requesting a Service of type: NodePort. The cloud-controller-manager component then configures the external load balancer to forward traffic to that assigned node port.

You can configure a load balanced Service to omit assigning a node port, provided that the cloud provider implementation supports this.

Some cloud providers allow you to specify the loadBalancerIP. In those cases, the load-balancer is created with the user-specified loadBalancerIP. If the loadBalancerIP field is not specified, the load balancer is set up with an ephemeral IP address. If you specify a loadBalancerIP but your cloud provider does not support the feature, the loadbalancerIP field that you set is ignored.

Imperative and Declarative Approach in Kubernetes

Difference Between Imperative and Declarative Approach in Kubernetes

Imperative Approach (Command-Based)

Definition:
In the **imperative approach**, we **give direct commands** to Kubernetes to perform actions like creating or modifying resources using kubectl run, kubectl create, or kubectl expose.

Example:
If we want to create an **Nginx pod** using the imperative approach, we run:

```
kubectl run nginx --image=nginx --port=80
```

Or to expose a service:

```
kubectl expose pod nginx --port=80 --type=NodePort
```

- **Features of Imperative Approach:**

  - **Direct Commands:** Resources are created and modified using direct commands.
  - **Instant Execution:** The commands execute immediately.
  - **Less Maintainable:** Changes must be manually applied each time.

Declarative Approach (YAML-based)

Definition:
In the **declarative approach**, we **define the desired state** of Kubernetes resources in **YAML files**, and then apply them using kubectl apply. Kubernetes ensures that the system reaches and maintains this desired state.

Example:
If we want to deploy **Nginx using a Deployment**, we define the following **YAML file**:

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

To apply this YAML file:

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

- **Features of Declarative Approach:**

- **State-Based Management:** We define the **desired state**, and Kubernetes automatically maintains it.
- **Version Control Friendly:** YAML files can be stored in **Git** for tracking changes.
- **Easier Maintenance:** It allows for automation and better management of the cluster.