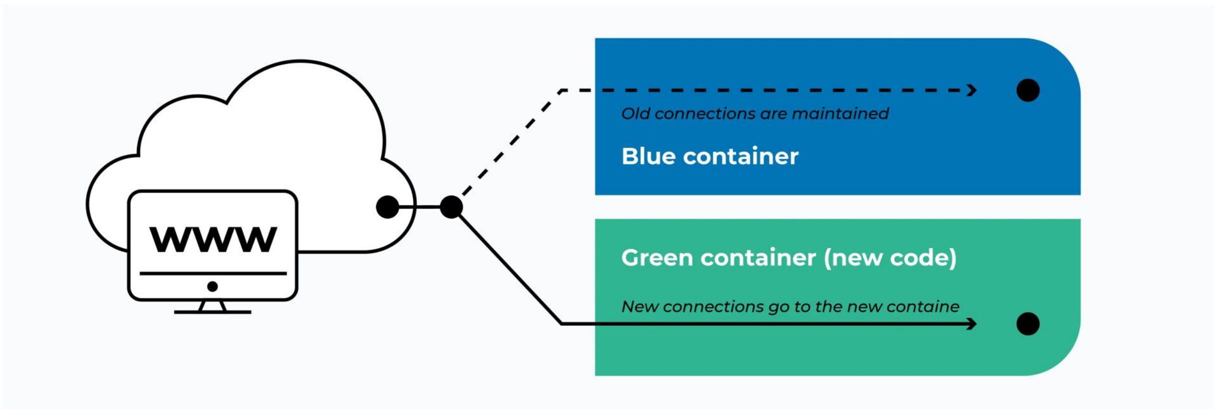
●● Blue-Green Deployment in Kubernetes


www
Old connections are maintained
Blue container
Green container (new code)
New connections go to the new containe

What is Blue-Green Deployment in Kubernetes?

In Kubernetes, **Blue-Green Deployment** means running two versions of your application (Blue = Current Live, Green = New Version) **simultaneously** in the cluster, and using a **Kubernetes Service** to switch traffic between them.

- You deploy **two separate Deployments** (blue and green) with different labels.
- A **Kubernetes Service** acts as a fixed endpoint (like a load balancer) that routes traffic to **only one** of them at a time.
- To switch, you **change the selector** of the Service to point to the new version.

How It Works

1. **Create the Blue Deployment**
   - Current production app.
   - Service points to it using a label like version: blue.
2. **Deploy the Green Deployment**
   - New version of the app.
   - It runs alongside the blue one.
   - But the Service still points to blue.
3. **Test Green**
   - Expose it via a temporary service or port-forward for testing.

4. **Switch Traffic**
   - Once green is ready and tested, routing (via load balancer, DNS, etc.) is changed from blue to green.
   - Green becomes the new live environment.

5. **Rollback?**
   - Just switch the selector back to version: blue.

6. **Blue Becomes Idle (or Backup)**
   - Blue is kept idle as a backup for quick rollback.
   - If something fails in green, revert traffic back to blue.

🟩 Why Use Blue-Green in K8s?

- 🟩 Zero downtime
- 🟩 Fast rollback
- 🟩 Parallel running versions
- 🟩 Easy integration with GitOps or CI/CD pipelines

Advantages of Blue-Green Deployment

Advantage	Explanation
Zero Downtime	Traffic switches instantly without affecting users.
Instant Rollback	Easily revert to blue if green fails.
Safer Testing	Can test green in a real environment before it goes live.
Reduced Risk	Live and staging are identical, minimizing surprises.

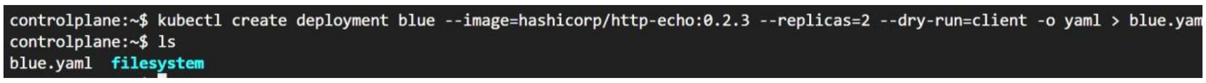
Disadvantages

Disadvantage	Explanation
Expensive	Requires two complete environments (infra/resources).
Data Sync Issues	Need strategies to handle DB migrations between environments.
Configuration Complexity	Requires load balancer or routing strategy.

PRACTICAL (Step-by-Step)

🟩 STEP 1: Create Blue Deployment YAML Using Dry-Run

- **kubectl create deployment blue --image=hashicorp/http-echo:0.2.3 --replicas=2 --dry-run=client -o yaml > blue.yaml**

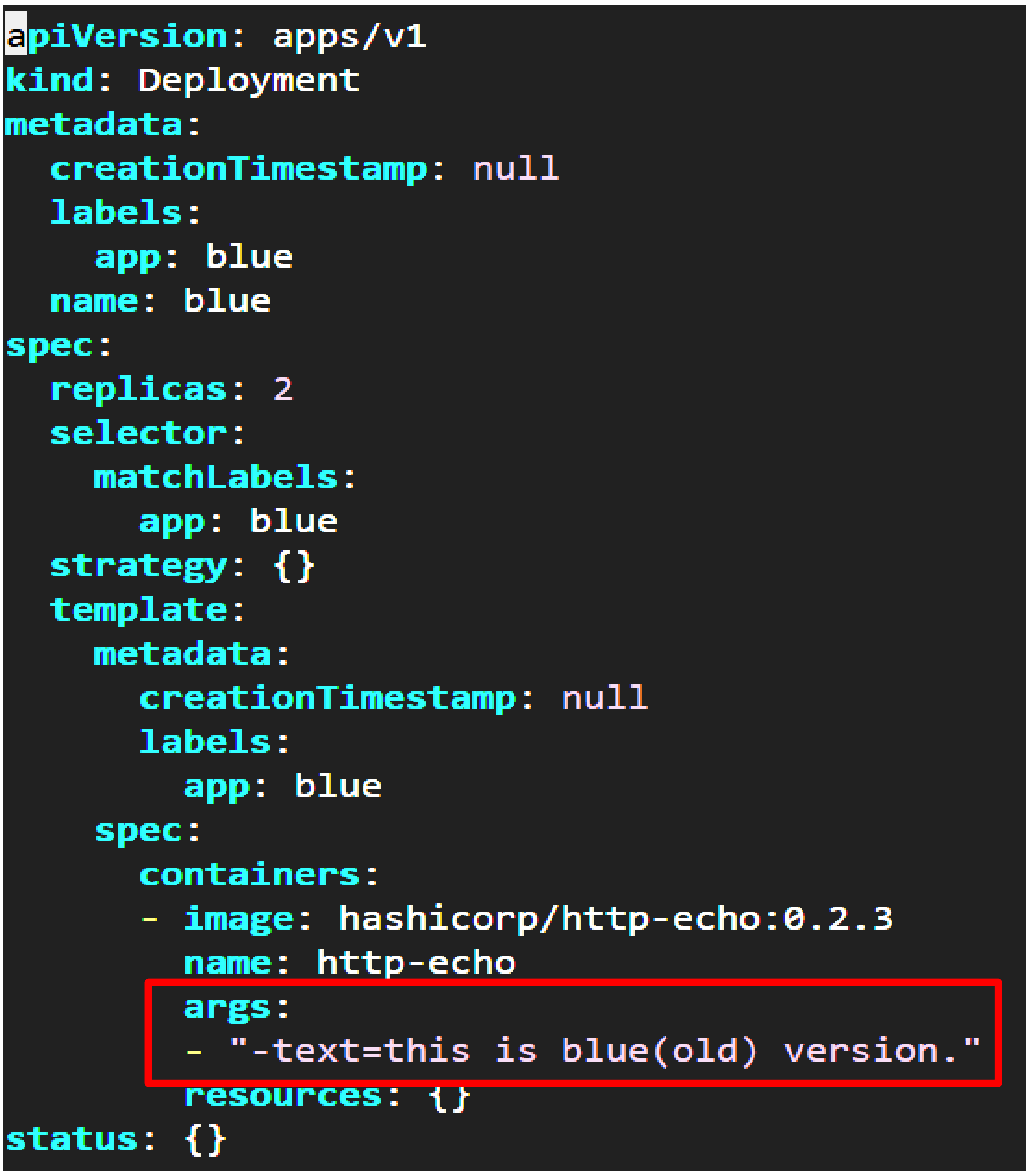


```
controlplane:~$ kubectl create deployment blue --image=hashicorp/http-echo:0.2.3 --replicas=2 --dry-run=client -o yaml > blue.yam
controlplane:~$ ls
blue.yaml  filesystem
```

Edit blue.yaml

Add:

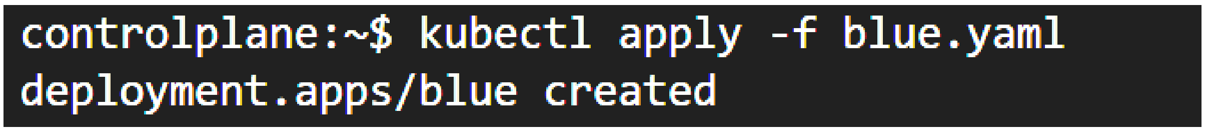
- Argument section to get specific text from specific version of deployment.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: blue
  name: blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: blue
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: blue
    spec:
      containers:
      - image: hashicorp/http-echo:0.2.3
        name: http-echo
        args:
        - "-text=this is blue(old) version."
        resources: {}
status: {}
```

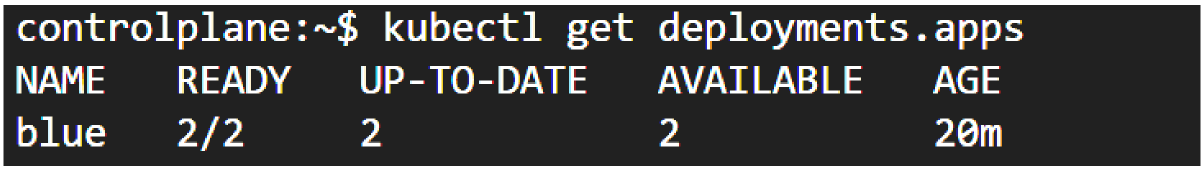
Apply Blue Deployment

- kubectl apply -f blue.yaml



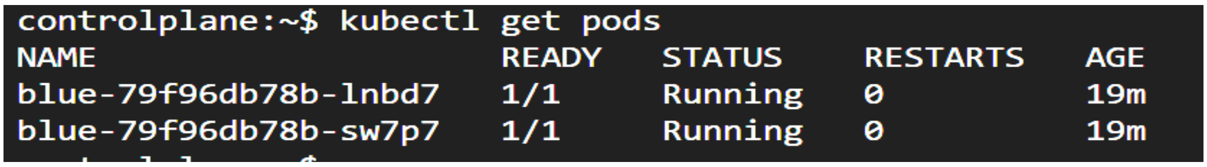
```
controlplane:~$ kubectl apply -f blue.yaml
deployment.apps/blue created
```

- Check deployment



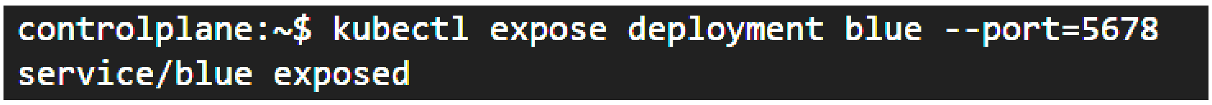
```
controlplane:~$ kubectl get deployments.apps
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
blue    2/2     2            2           20m
```

- Check the status of pods



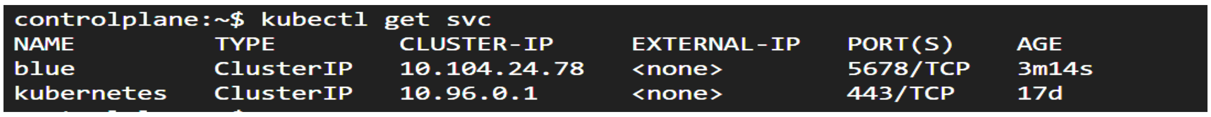
```
controlplane:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
blue-79f96db78b-lnbd7     1/1     Running   0          19m
blue-79f96db78b-sw7p7     1/1     Running   0          19m
```

🟩 STEP 2: Create Service



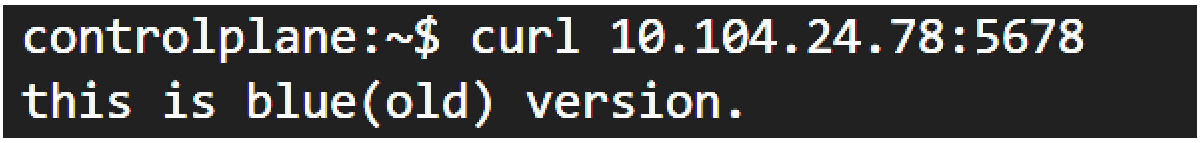
```
controlplane:~$ kubectl expose deployment blue --port=5678
service/blue exposed
```

- Check service and access service



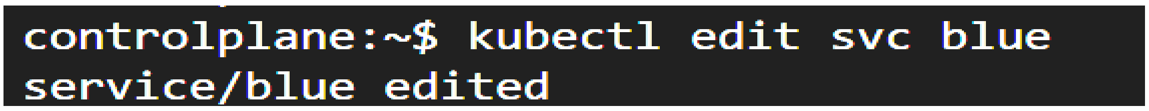
```
controlplane:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
blue         ClusterIP   10.104.24.78    <none>        5678/TCP   3m14s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    17d
```

- Access the service

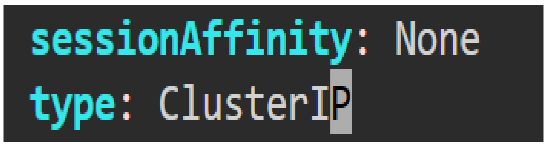


```
controlplane:~$ curl 10.104.24.78:5678
this is blue(old) version.
```

- To Access outside we have to edit service type=NodePort

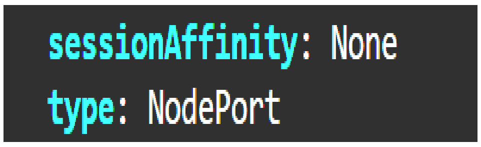


```
controlplane:~$ kubectl edit svc blue
service/blue edited
```



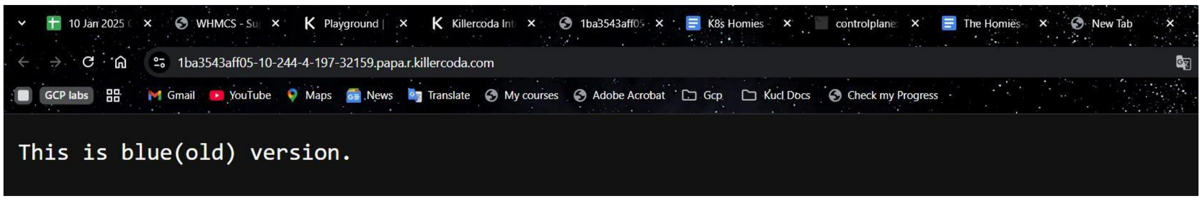
```
sessionAffinity: None
type: ClusterIP
```

to



```
sessionAffinity: None
type: NodePort
```

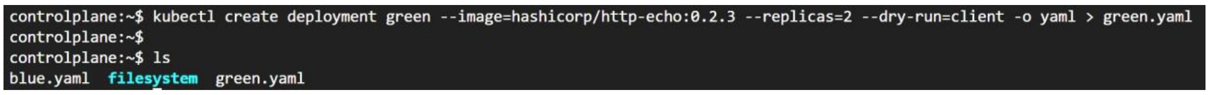
- **Now you can access it outside using node ip & port.**



```
This is blue(old) version.
```

STEP 3: Create Green Deployment Using Dry-Run

- **Kubectl create deployment green --image=hashicorp/http-echo:0.2.3 --replicas=2 --dry-run=client -o yaml > green.yaml**

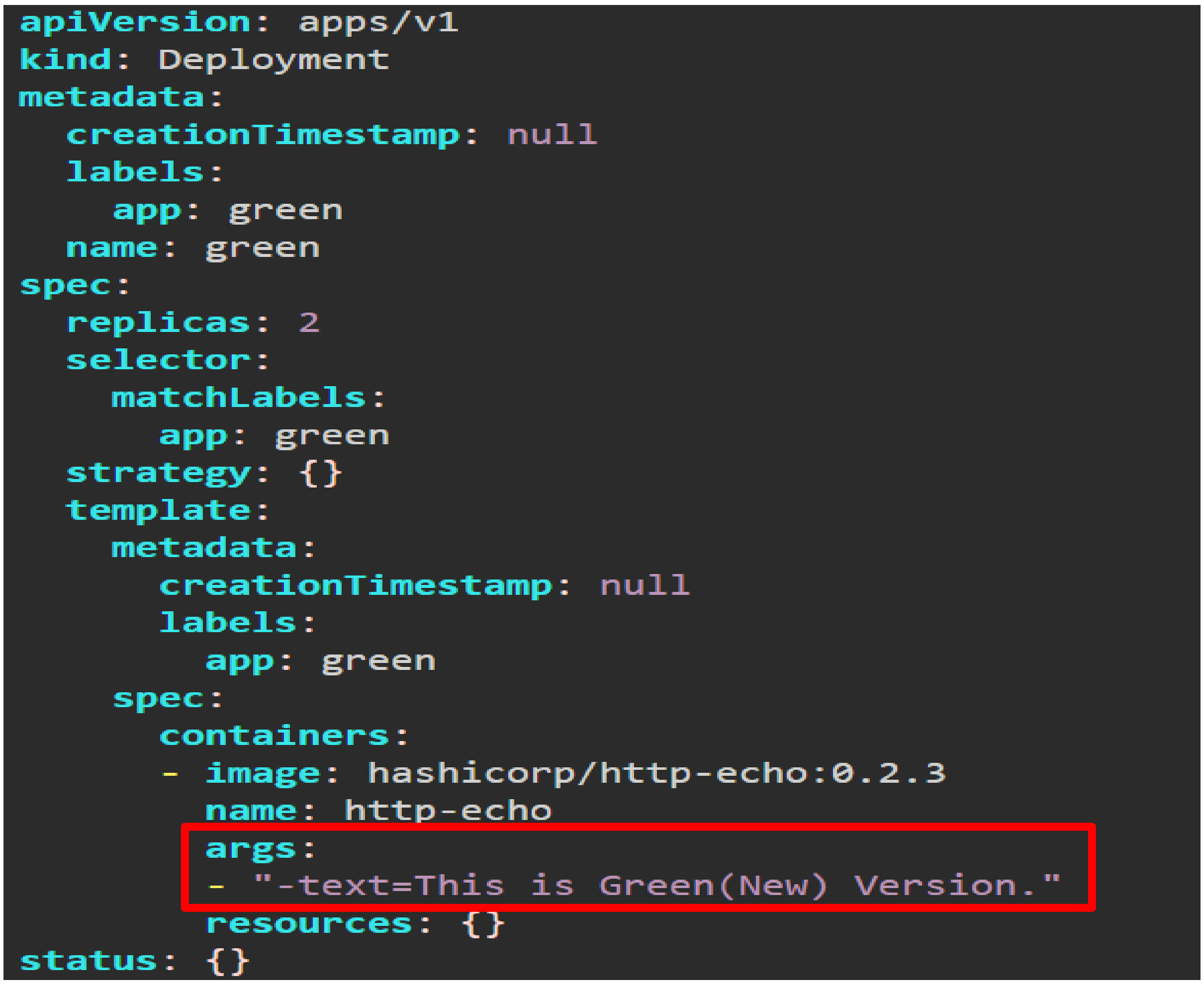


```
controlplane:~$ kubectl create deployment green --image=hashicorp/http-echo:0.2.3 --replicas=2 --dry-run=client -o yaml > green.yaml
controlplane:~$
controlplane:~$ ls
blue.yaml  filesystem  green.yaml
```

Edit green.yaml

Add:

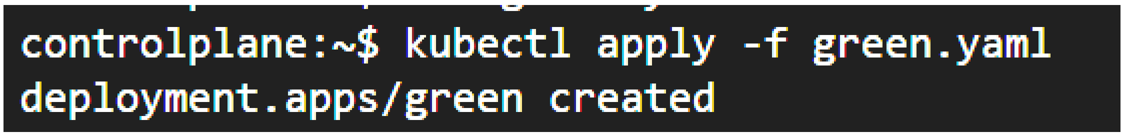
- **Argument section to get specific text from specific version of deployment.**



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: green
  name: green
spec:
  replicas: 2
  selector:
    matchLabels:
      app: green
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: green
    spec:
      containers:
      - image: hashicorp/http-echo:0.2.3
        name: http-echo
        args:
        - "-text=This is Green(New) Version."
        resources: {}
status: {}
```

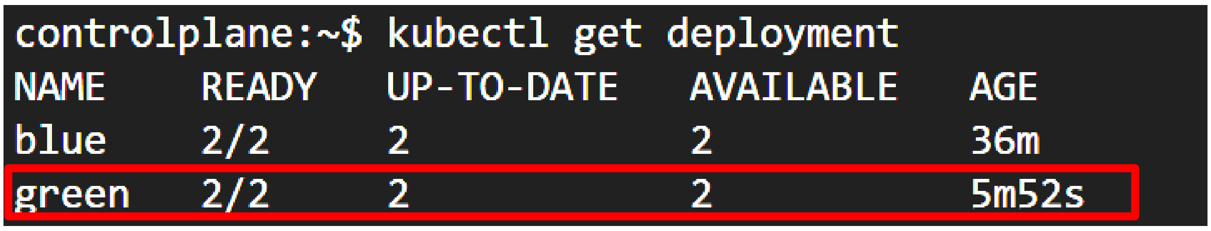
) **Apply Green Deployment**

- **kubectl apply -f green.yaml**



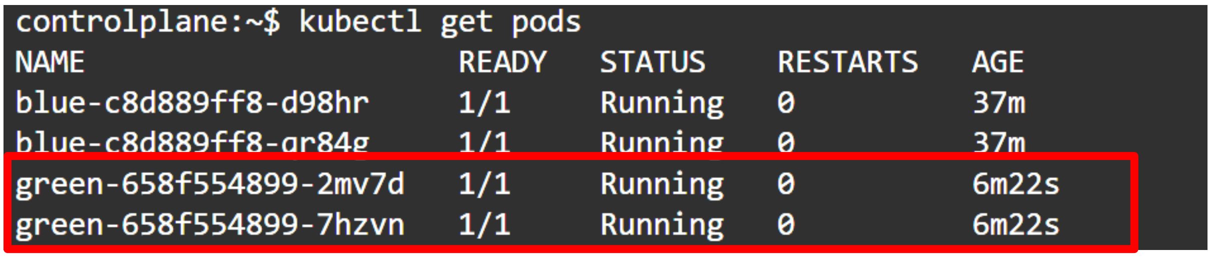
```
controlplane:~$ kubectl apply -f green.yaml
deployment.apps/green created
```

- **Check deployment**



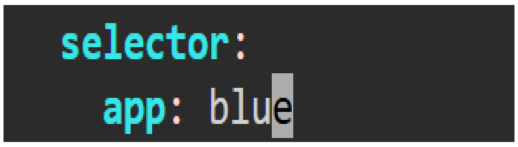
```
controlplane:~$ kubectl get deployment
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
blue    2/2     2            2           36m
green   2/2     2            2           5m52s
```

- **Check the status of pods**



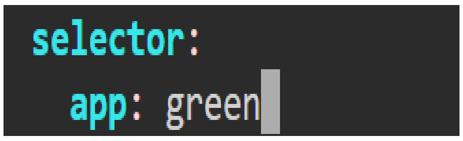
```
controlplane:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
blue-c8d889ff8-d98hr      1/1     Running   0          37m
blue-c8d889ff8-gr84g      1/1     Running   0          37m
green-658f554899-2mv7d    1/1     Running   0          6m22s
green-658f554899-7hzvn    1/1     Running   0          6m22s
```

🟩 STEP 4: Now we have to edit the service to access the Green (new Version) of the deployment so change Selector

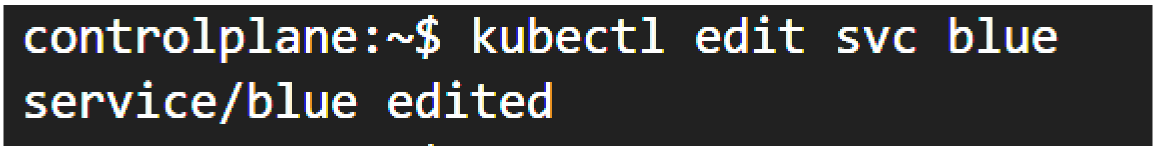


```
  selector:
    app: blue
```

To

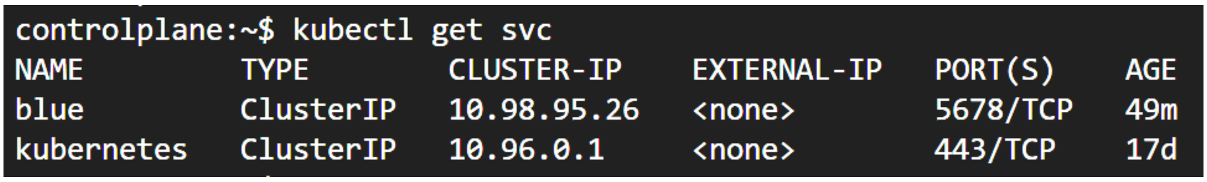


```
  selector:
    app: green
```



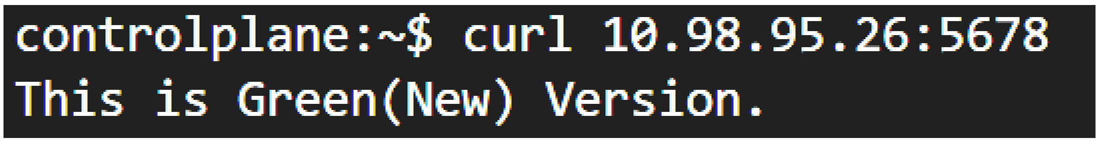
```
controlplane:~$ kubectl edit svc blue
service/blue edited
```

- **Check service and access service**



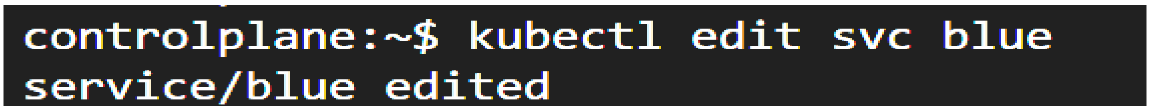
```
controlplane:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
blue         ClusterIP   10.98.95.26   <none>        5678/TCP   49m
kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP    17d
```

- **Access the service**

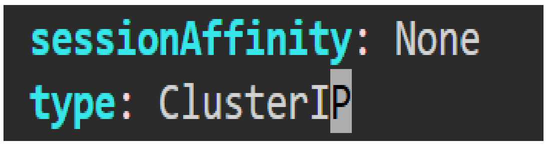


```
controlplane:~$ curl 10.98.95.26:5678
This is Green(New) Version.
```

- To Access outside we have to edit service type=NodePort

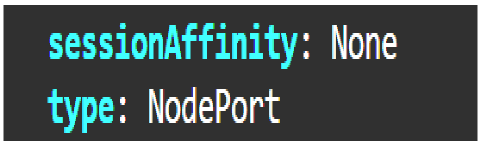


```
controlplane:~$ kubectl edit svc blue
service/blue edited
```



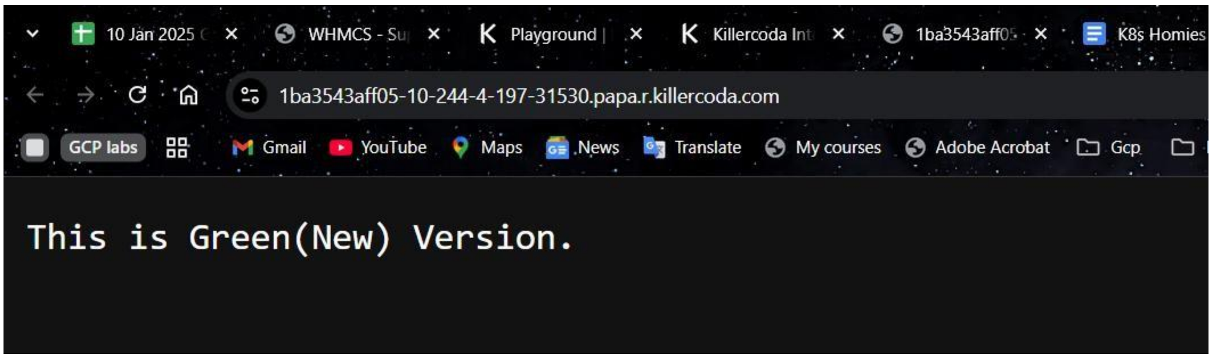
```
sessionAffinity: None
type: ClusterIP
```

to



```
sessionAffinity: None
type: NodePort
```

- Now you can access it outside using node ip & port. & here you can Green version is successfully deployed.



1ba3543aff05-10-244-4-197-31530.papa.r.killercoda.com

```
This is Green(New) Version.
```

STEP 5: SWITCH TO GREEN (PATCH SERVICE)

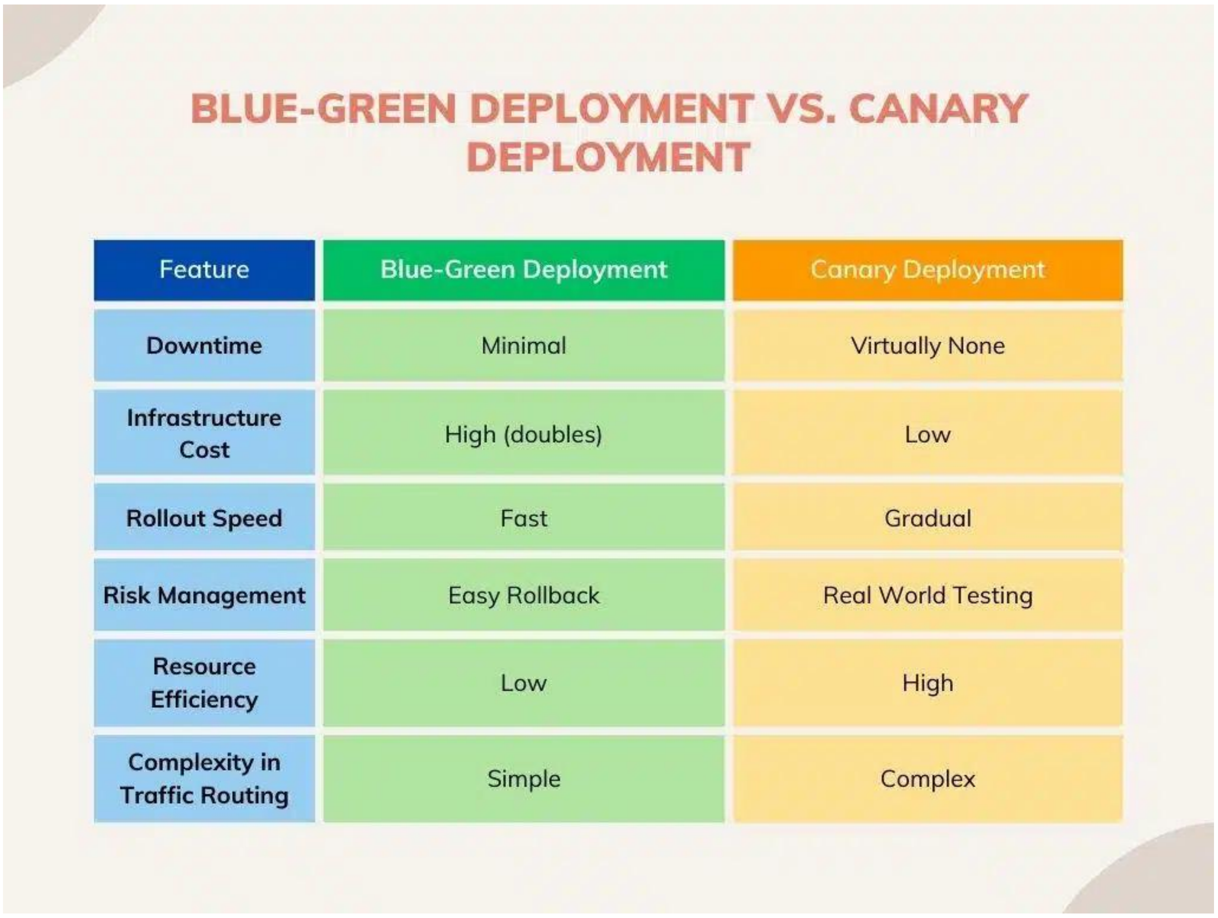
- kubectl patch service blue -p '{"spec":{"selector":{"app":"blue"}}}'

Check again in browser — now you'll see BLUE version!

ROLLBACK TO BLUE (If Needed)

- kubectl patch service blue -p '{"spec":{"selector":{"app":"green"}}}'

Differences from Rolling / Canary



BLUE-GREEN DEPLOYMENT VS. CANARY DEPLOYMENT

Feature	Blue-Green Deployment	Canary Deployment
Downtime	Minimal	Virtually None
Infrastructure Cost	High (doubles)	Low
Rollout Speed	Fast	Gradual
Risk Management	Easy Rollback	Real World Testing
Resource Efficiency	Low	High
Complexity in Traffic Routing	Simple	Complex

Thank your……