Kubernetes_4

```
eksctl create cluster --name my-eks-cluster --region ca-central-1 --node-type t2.medium --zones ca-central-1a,ca-central-1b
```

```
ubuntu@ip-172-31-9-165:~$ kubectl apply -f deployment.yml
deployment.apps/javawebdeployment created
ubuntu@ip-172-31-9-165:~$
```

```
ubuntu@ip-172-31-9-165:~$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/javawebdeployment-57988f5cd7-9g8tr   1/1     Running   0          21s
pod/javawebdeployment-57988f5cd7-p7cb7   1/1     Running   0          22s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   8m11s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/javawebdeployment   2/2     2            2           22s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/javawebdeployment-57988f5cd7   2         2         2       22s
```

```
ubuntu@ip-172-31-9-165:~$ kubectl scale deployment javawebdeployment --replicas 4
deployment.apps/javawebdeployment scaled
ubuntu@ip-172-31-9-165:~$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
javawebdeployment-57988f5cd7-9g8tr   1/1     Running   0          2m
javawebdeployment-57988f5cd7-g2th6   1/1     Running   0          5s
javawebdeployment-57988f5cd7-p7cb7   1/1     Running   0          2m1s
javawebdeployment-57988f5cd7-t6xpg   1/1     Running   0          5s
ubuntu@ip-172-31-9-165:~$
```

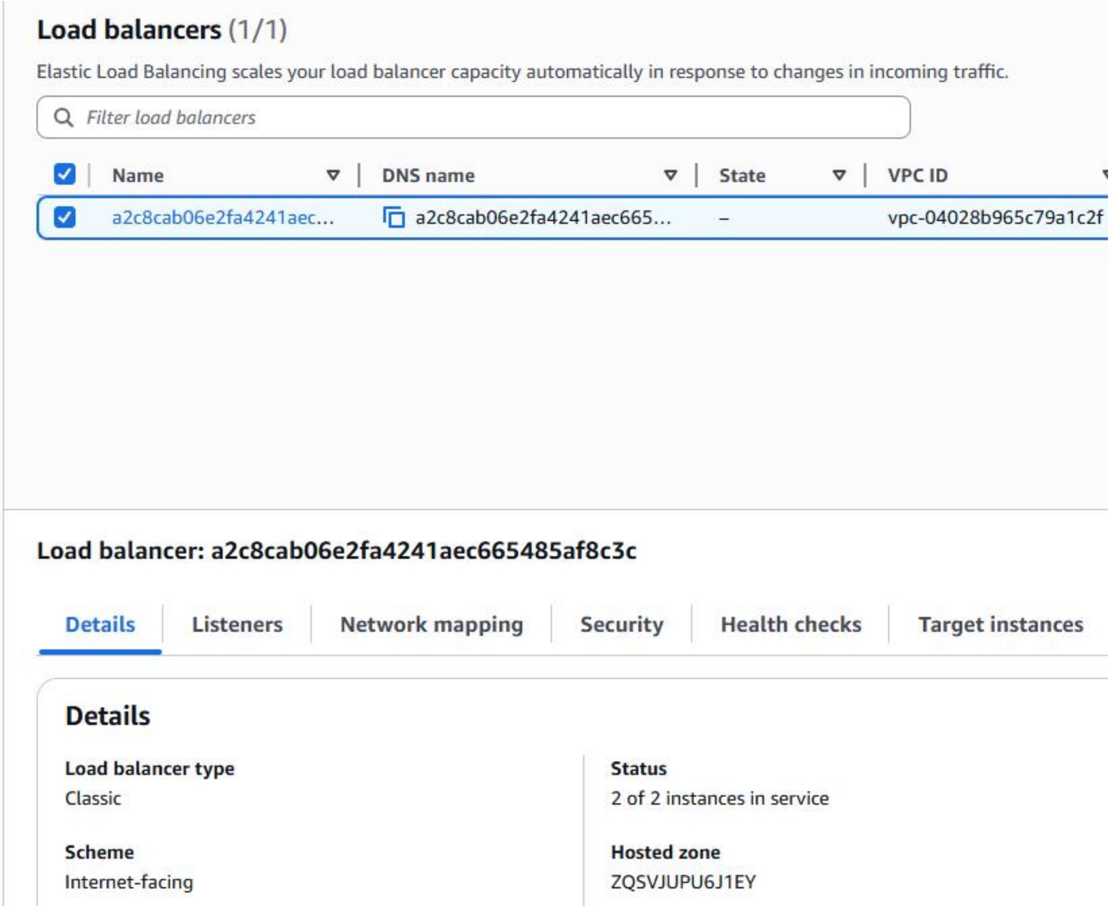
If you don’t explicitly specify service, the default service is ClusterIP

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: javawebdeploy
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: javawebapp
  template:
    metadata:
      name: javawebpod
      labels:
        app: javawebapp
    spec:
      containers:
        - name: javawebappcontainer
          image: hacker123shiva/springbt-in-docker:latest
          ports:
           - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
    name: javawebappsvc
spec:
    type: LoadBalancer
    selector:
        app: javawebapp
    ports:
        - port: 80
          targetPort: 8080
…
```

```
ubuntu@ip-172-31-9-165:~$ cat dep-svc.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: javawebdeployment
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: javawebapp
  template:
    metadata:
      name: javawebpod
      labels:
        app: javawebapp
    spec:
      containers:
        - name: javawebappcontainer
          image: hacker123shiva/springbt-in-docker:latest
          ports:
           - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
    name: javawebappsvc
spec:
    type: LoadBalancer
    selector:
        app: javawebapp
    ports:
        - port: 80
          targetPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~$ kubectl apply -f dep-svc.yml
deployment.apps/javawebdeployment created
service/javawebappsvc created


ubuntu@ip-172-31-9-165:~$ kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/javawebdeployment-57988f5cd7-4mhbd   1/1     Running   0          70s
pod/javawebdeployment-57988f5cd7-wm8gz   1/1     Running   0          70s

NAME                TYPE           CLUSTER-IP       EXTERNAL-IP                                                  PORT(S)
AGE
service/javawebappsvc   LoadBalancer   10.100.177.181   a2c8cab06e2fa4241aec665485af8c3c-
58365812.ca-central-1.elb.amazonaws.com   80:31546/TCP   70s
service/kubernetes      ClusterIP      10.100.0.1       <none>
443/TCP        7m48s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/javawebdeployment   2/2     2            2           71s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/javawebdeployment-57988f5cd7   2         2         2       71s
```

Go to EC2 Loadbalancer



Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

☑	Name	DNS name	State	VPC ID
☑	a2c8cab06e2fa4241aec...	a2c8cab06e2fa4241aec665...	–	vpc-04028b965c79a1c2f

Load balancer: a2c8cab06e2fa4241aec665485af8c3c

Details

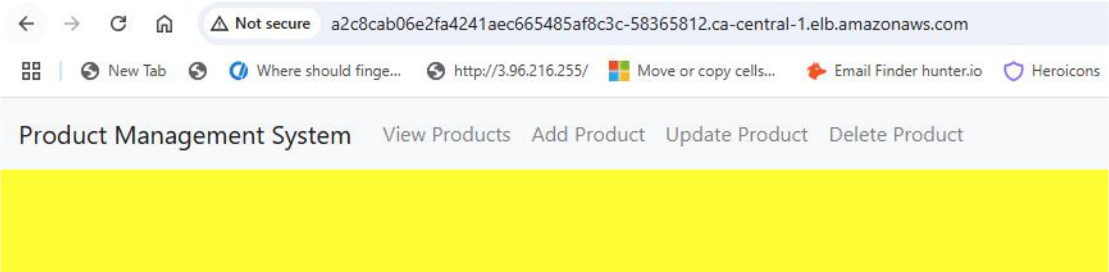
Load balancer type
Classic

Status
2 of 2 instances in service

Scheme
Internet-facing

Hosted zone
ZQSVJUPU6J1EY

a2c8cab06e2fa4241aec665485af8c3c-58365812.ca-central-1.elb.amazonaws.com



```
ubuntu@ip-172-31-9-165:~$ kubectl delete all --all
pod "javawebdeployment-57988f5cd7-4mhbd" deleted
pod "javawebdeployment-57988f5cd7-wm8gz" deleted
service "javawebappsvc" deleted
service "kubernetes" deleted
deployment.apps "javawebdeployment" deleted
replicaset.apps "javawebdeployment-57988f5cd7" deleted
```

Autoscaling
Process of increasing or decreasing the infrastructure resources based on the demand

Autoscaling: Horizontal (increasing / decreasing infrastructure resources based on the demand) and Vertical scaling (increasing the capacity of the same single system / machine / pod)

HPA (Horizontal Pod Autoscaling) and VPA (Vertical Pod Autoscaling)

Why Autoscale in K8s?
High availability of application or better availability of application
Elasticity
Efficient resource utilization
Seamless load management

HPA (Horizontal Pod Autoscaling) --> used to scale up or scale down number of pod replicas based on observed metrics (CPU or memory utilization). we cannot simply add or remove resources, we got to check certain metrics before we hire someone

HPA needs metrics to adjust the pods
HPA observes all metrics --> based on the observation, it will add/delete pods, tracks multiple metrics

HPA will interact with Metric server to identify CPU/memory utilization of POD

Metric server is an application that collects metrics from objects, pods, nodes according to state of CPU and memory

```
ubuntu@ip-172-31-9-165:~$ kubectl top nodes
NAME                                          CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
ip-192-168-13-3.ca-central-1.compute.internal     39m         2%       629Mi           18%
ip-192-168-34-135.ca-central-1.compute.internal   24m         1%       550Mi           16%
```

Metrics server will not be present by default in Kubernetes server

Install metrics API

```
$ mkdir k8s-metrics-server
$ cd k8s-metrics-server
$ vi metrics-api-server.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: metrics-server
  name: metrics-server
  namespace: metrics
…
```

```
ubuntu@ip-172-31-9-165:~$ cat metrics-api-server.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: metrics-server
  name: metrics-server
  namespace: metrics
...
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ cat metrics-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: metrics-server
  namespace: kube-system
  labels:
    k8s-app: metrics-server
spec:
  selector:
    matchLabels:
      k8s-app: metrics-server
      app.kubernetes.io/instance: metrics-server
      app.kubernetes.io/name: metrics-server
  strategy:
    rollingUpdate:
      maxUnavailable: 0
  template:
    metadata:
      labels:
        k8s-app: metrics-server
        app.kubernetes.io/instance: metrics-server
        app.kubernetes.io/name: metrics-server
    spec:
      serviceAccountName: metrics-server
      nodeSelector:
        kubernetes.io/os: linux
      priorityClassName: system-cluster-critical
      containers:
        - name: metrics-server
          image: k8s.gcr.io/metrics-server/metrics-server:v0.5.0
          imagePullPolicy: IfNotPresent
          args:
            - --cert-dir=/tmp
            - --secure-port=443
            - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
            - --kubelet-use-node-status-port
            - --metric-resolution=15s
            - --kubelet-insecure-tls
          ports:
            - containerPort: 443
              name: https
              protocol: TCP
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /livez
              port: https
              scheme: HTTPS
            periodSeconds: 10
          readinessProbe:
            failureThreshold: 3
            httpGet:
              path: /readyz
              port: https
              scheme: HTTPS
            initialDelaySeconds: 20
            periodSeconds: 10
          resources:
            requests:
              cpu: 100m
              memory: 200Mi
          securityContext:
            readOnlyRootFilesystem: true
            runAsNonRoot: true
            runAsUser: 1000
          volumeMounts:
            - mountPath: /tmp
              name: tmp-dir
      volumes:
        - name: tmp-dir
          emptyDir: {}
...
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ cat metrics-rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metrics-server
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:aggregated-metrics-reader
rules:
  - apiGroups: ["metrics.k8s.io"]
    resources: ["pods", "nodes"]
    verbs: ["get", "list", "watch"]
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:metrics-server
rules:
  - apiGroups: [""]
    resources:
      - pods
      - nodes
      - nodes/stats
      - namespaces
      - configmaps
      - services
    verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metrics-server:system:auth-delegator
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
  - kind: ServiceAccount
    name: metrics-server
    namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:metrics-server
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:metrics-server
subjects:
  - kind: ServiceAccount
    name: metrics-server
    namespace: kube-system
...
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ cat metrics-server-service.yaml

---
apiVersion: v1
kind: Service
metadata:
  name: metrics-server
  namespace: kube-system
  labels:
    k8s-app: metrics-server
spec:
  selector:
    k8s-app: metrics-server
  ports:
    - port: 443
      targetPort: https
      protocol: TCP
      name: https
  type: ClusterIP
...
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ cat metrics-serviceaccount.yaml
---
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: v1beta1.metrics.k8s.io
  labels:
    k8s-app: metrics-server
spec:
  group: metrics.k8s.io
  version: v1beta1
  insecureSkipTLSVerify: true
  groupPriorityMinimum: 100
  versionPriority: 100
  service:
    name: metrics-server
    namespace: kube-system
...
```

Create a metric system inside the namespace: kube-system

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ ls -l
total 20
-rw-rw-r-- 1 ubuntu ubuntu  138 May 25 18:42 metrics-api-server.yaml
-rw-rw-r-- 1 ubuntu ubuntu  923 May 25 18:44 metrics-deployment.yaml
-rw-rw-r-- 1 ubuntu ubuntu 1212 May 25 18:48 metrics-rbac.yaml
-rw-rw-r-- 1 ubuntu ubuntu  288 May 25 18:52 metrics-server-service.yaml
-rw-rw-r-- 1 ubuntu ubuntu  102 May 25 18:55 metrics-serviceaccount.yaml
```

Run or Execute the Yaml file

```
$ kubectl apply -f k8s-metrics-server (run directory only if all yml files inside the dir are correct)
```

Recommended approach: Run all yaml files individually

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ kubectl apply -f metrics-api-server.yaml
Warning: resource serviceaccounts/metrics-server is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
serviceaccount/metrics-server configured
```

```
kubectl apply -f metrics-deployment.yaml
deployment.apps/metrics-server created
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ kubectl apply -f metrics-rbac.yaml
serviceaccount/metrics-server configured
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
```

```
Warning: resource clusterroles/system:metrics-server is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
clusterrole.rbac.authorization.k8s.io/system:metrics-server configured
Warning: resource clusterrolebindings/metrics-server:system:auth-delegator is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator configured
Warning: resource clusterrolebindings/system:metrics-server is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server configured
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ kubectl apply -f metrics-server-service.yaml
Warning: resource services/metrics-server is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
service/metrics-server configured
```

```
ubuntu@ip-172-31-9-165:~/k8s-metrics-server$ kubectl apply -f metrics-serviceaccount.yaml
Warning: resource apiservices/v1beta1.metrics.k8s.io is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io configured
```

It will create metrics-server, service-account, role, role binding and all the stuff required

Check top nodes using metrics-server

```
ubuntu@ip-172-31-9-165:~$ eksctl create addon --name metrics-server --cluster my-eks-cluster --region ca-central-1  --force
```

```
ubuntu@ip-172-31-9-165:~$ kubectl get pods -n kube-system -l k8s-app=metrics-server
NAME                              READY   STATUS    RESTARTS   AGE
metrics-server-79bb88c6fc-gv6mv   0/1     Running   0          11m
```

```
kubectl -n kube-system exec -it metrics-server-79bb88c6fc-gv6mv -- /bin/sh
curl -k https://<worker-node-ip>:10250/stats/summary
```

I will debug later

```
ubuntu@ip-172-31-9-165:~$ kubectl top nodes
error: Metrics API not available
```

https://docs.aws.amazon.com/eks/latest/userguide/metrics-server.html

This is working fine

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

```
kubectl get deployment metrics-server -n kube-system
```

kubectl top nodes

```
ubuntu@ip-172-31-9-165:~$ kubectl get deployment metrics-server -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           84s
ubuntu@ip-172-31-9-165:~$ kubectl top nodes
NAME                                             CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
ip-192-168-13-3.ca-central-1.compute.internal     28m          1%       622Mi           18%
ip-192-168-34-135.ca-central-1.compute.internal   28m          1%       617Mi           18%

ubuntu@ip-172-31-9-165:~$ kubectl top pods
No resources found in default namespace.
```

Note: metrics-server will be installed under kube-system namespace

Deploy same application

```
ubuntu@ip-172-31-9-165:~$ cat hpa-demo-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpa-demo-deployment
  labels:
    app: hpa-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hpa-demo
  template:
    metadata:
      labels:
        app: hpa-demo
    spec:
      containers:
      - name: hpa-demo-container
        image: k8s.gcr.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 100m
          limits:
            cpu: 500m

ubuntu@ip-172-31-9-165:~$ kubectl apply -f  hpa-demo-deployment.yaml
deployment.apps/hpa-demo-deployment created
```

Create HPA service

```
ubuntu@ip-172-31-9-165:~$ cat hpa-demo-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: hpa-demo-service
  labels:
    run: hpa-demo
```

```
spec:
  selector:
    run: hpa-demo
  ports:
  - port: 80
    targetPort: 80

ubuntu@ip-172-31-9-165:~$ kubectl apply -f hpa-demo-service.yaml
service/hpa-demo-service created
```

Create HPA

```
ubuntu@ip-172-31-9-165:~$ cat hpa-demo.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-demo-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hpa-demo-deployment
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50

ubuntu@ip-172-31-9-165:~$ kubectl apply -f hpa-demo.yaml
horizontalpodautoscaler.autoscaling/hpa-demo-hpa created

ubuntu@ip-172-31-9-165:~$ kubectl get deploy
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
hpa-demo-deployment   1/1     1            1           25m


ubuntu@ip-172-31-9-165:~$ kubectl get svc
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
hpa-demo-service   ClusterIP   10.100.42.151   <none>        80/TCP    18m
kubernetes         ClusterIP   10.100.0.1      <none>        443/TCP   5h57m


ubuntu@ip-172-31-9-165:~$ kubectl get hpa
NAME           REFERENCE                        TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 1%/50%   1         5         1          8m39s
```

To demonstrate auto-scaling, I will increase load on this machine

```
kubectl run -i --tty load-generator --image=busybox --restart=Never -- /bin/sh -c “while sleep 0.01; do
wget -q -o- http://hpa-demo; done”
```

kubectl run -i --tty load-generator --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://hpa-demo-service; done"

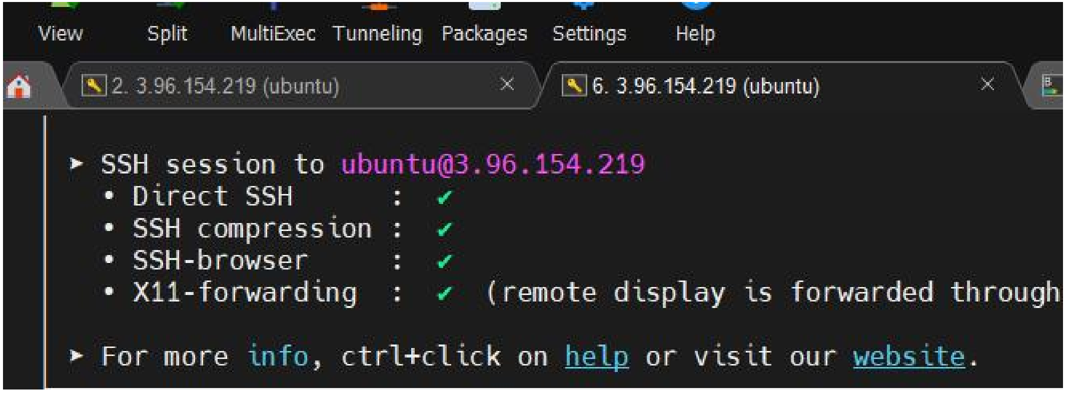
```
ubuntu@ip-172-31-9-165:~$ kubectl get pods -l app=hpa-demo
NAME                              READY  STATUS    RESTARTS  AGE
hpa-demo-deployment-7577d65cb7-ckp7c  1/1    Running  0        48m
```

Make sure

```
ubuntu@ip-172-31-9-165:~$ kubectl get endpoints hpa-demo-service
NAME             ENDPOINTS          AGE
hpa-demo-service  192.168.32.129:80  7s
```

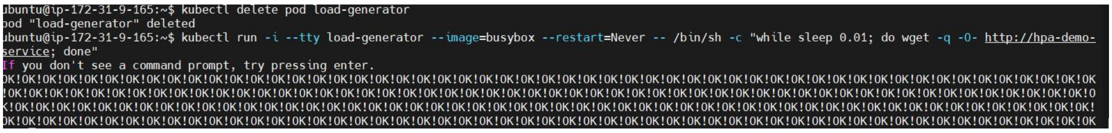
```
ubuntu@ip-172-31-9-165:~$ kubectl run -i --tty load-generator --image=busybox --restart=Never --
/bin/sh -c "while sleep 0.01; do wget -q -O- http://hpa-demo-service; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!^Cpod default/load-generator terminated (Error)
```

Duplicate tabs



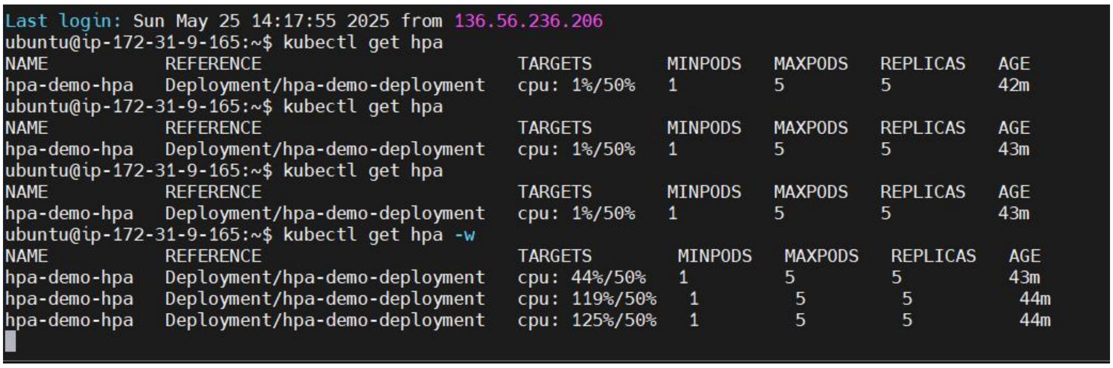
```
► SSH session to ubuntu@3.96.154.219
  • Direct SSH      :  ✓
  • SSH compression :  ✓
  • SSH-browser     :  ✓
  • X11-forwarding  :  ✓  (remote display is forwarded through

► For more info, ctrl+click on help or visit our website.
```

I run this in one window



```
ubuntu@ip-172-31-9-165:~$ kubectl delete pod load-generator
pod "load-generator" deleted
ubuntu@ip-172-31-9-165:~$ kubectl run -i --tty load-generator --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://hpa-demo-
service; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
```

Second window I observe



```
Last login: Sun May 25 14:17:55 2025 from 136.56.236.206
ubuntu@ip-172-31-9-165:~$ kubectl get hpa
NAME           REFERENCE                       TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment  cpu: 1%/50%  1         5         5          42m
ubuntu@ip-172-31-9-165:~$ kubectl get hpa
NAME           REFERENCE                       TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment  cpu: 1%/50%  1         5         5          43m
ubuntu@ip-172-31-9-165:~$ kubectl get hpa
NAME           REFERENCE                       TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment  cpu: 1%/50%  1         5         5          43m
ubuntu@ip-172-31-9-165:~$ kubectl get hpa -w
NAME           REFERENCE                       TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment  cpu: 44%/50%   1         5         5          43m
hpa-demo-hpa   Deployment/hpa-demo-deployment  cpu: 119%/50%   1         5         5          44m
hpa-demo-hpa   Deployment/hpa-demo-deployment  cpu: 125%/50%   1         5         5          44m
```

```
ubuntu@ip-172-31-9-165:~$ kubectl get hpa -w
NAME          REFERENCE                      TARGETS        MINPODS  MAXPODS  REPLICAS  AGE
hpa-demo-hpa  Deployment/hpa-demo-deployment  cpu: 44%/50%   1        5        5         43m
hpa-demo-hpa  Deployment/hpa-demo-deployment  cpu: 119%/50%   1        5        5         44m
hpa-demo-hpa  Deployment/hpa-demo-deployment  cpu: 125%/50%   1        5        5         44m
```

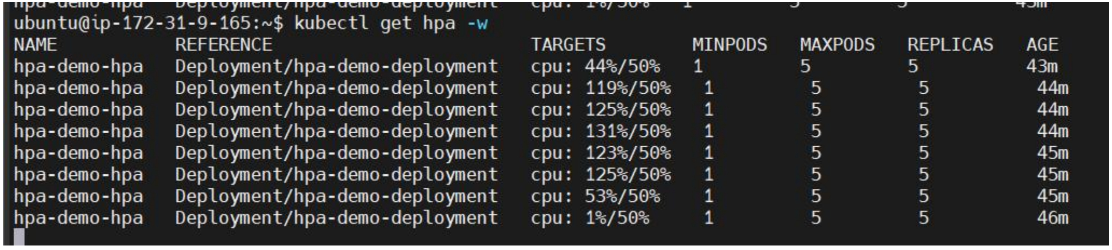
As load increases, POD increases, reached max 5

```
ubuntu@ip-172-31-9-165:~$ kubectl get hpa -w
NAME          REFERENCE                       TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 44%/50%   1        5        5        43m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 119%/50%   1        5        5        44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 125%/50%   1        5        5        44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 131%/50%   1        5        5        44m
```

I stopped the load balancer

```
ubuntu@ip-172-31-9-165:~$ kubectl get hpa -w
NAME          REFERENCE                       TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 44%/50%   1        5        5        43m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 119%/50%   1        5        5        44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 125%/50%   1        5        5        44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 131%/50%   1        5        5        44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 123%/50%   1        5        5        45m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 125%/50%   1        5        5        45m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 53%/50%   1        5        5        45m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 1%/50%    1        5        5        46m
```

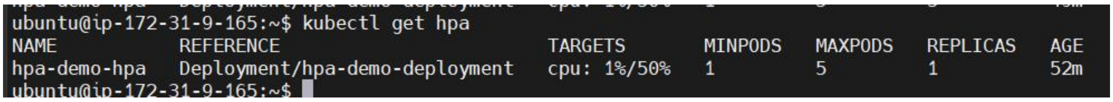
Now the load has dropped



```
ubuntu@ip-172-31-9-165:~$ kubectl get hpa -w
NAME           REFERENCE                        TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 44%/50%   1         5         5          43m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 119%/50%   1         5         5          44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 125%/50%   1         5         5          44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 131%/50%   1         5         5          44m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 123%/50%   1         5         5          45m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 125%/50%   1         5         5          45m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 53%/50%    1         5         5          45m
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 1%/50%     1         5         5          46m
```

After sometime, replicas is scaled-down as well

```
ubuntu@ip-172-31-9-165:~$ kubectl get hpa
NAME          REFERENCE                       TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 1%/50%   1        5        1        52m
```

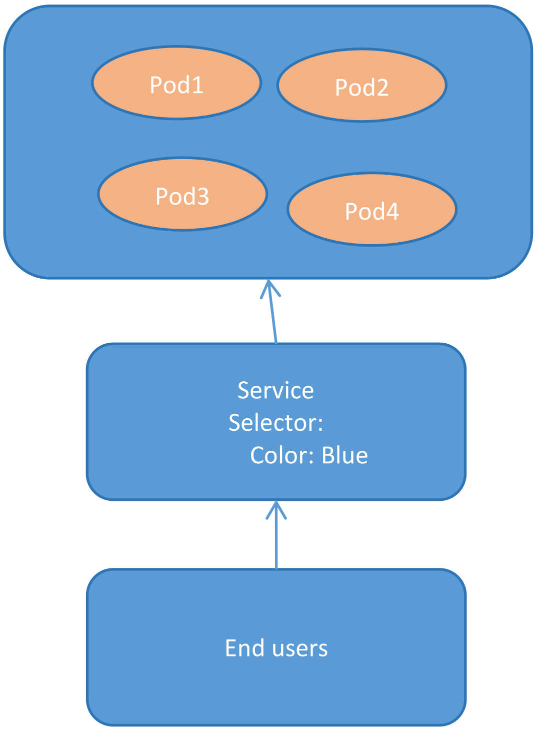


```
ubuntu@ip-172-31-9-165:~$ kubectl get hpa
NAME           REFERENCE                        TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
hpa-demo-hpa   Deployment/hpa-demo-deployment   cpu: 1%/50%   1         5         1          52m
ubuntu@ip-172-31-9-165:~$
```

Container orchestration --> K8s introduction --> Advantages --> Architecture --> Architecture components --> K8s cluster setup --> K8s resources --> POD, Service (ClusterIP, NodePort & LoadBalancer) --> Namespace --> ReplicaSet --> Deployment --> Metrics-server --> HPA

Blue-Green deployment model -> strategy of application deployment

Assume that application is running on different PODs. We cannot access the Pods directly so we create a Service to expose them. Endusers are accessing our application through Service. Application is inside the Pods, and after 10 days there are major updates that we want to do. If my Endusers are using the same Pods, inside Pods, application will be there. If we are exposing the same old Pods, then Endusers are using only the same Old applications. Every application changes over time. I want to have new Pods, if new Pods need to work, then it is a Deployment process. Deployment takes time and there will be some downtime. If I delete my old pods and create new pods, by default application downtime will be there. What if you have some errors or issues in your latest Pods. Don’t you think it is a big business loss. That’s where the concept of Blue-Green deployment comes into picture. It is like an application-release model, it will help you to decrease the downtime. It will follow a strategy in a way that downtime will also be reduced and if there is new problem or issue with the new Pods, Endusers can rollback / go back to the Old Pods with minimal time. It will show you how to release your application with minimal downtime. It is decreasing the risks we face during application release



If I say Selector: Color: Blue, people are able to access it. Then I will create another deployment and I will label as Green. I have created new Pods and this is deployed recently. Now before I give access to Endusers to access the new Pods, I will expose only to our testing team with the Pre-Production Service. Pre-Prod service is accessing the application only for the testing purpose. Inside the new Service, I will give the Selector as Green. Even if there is issues after new pods are deployed, our Endusers are not getting affected. Our testing team is now ok with the new pods. Then I change Selector color of Blue to Green. This is not re-deployment, it is not taking much time. New pods are already created, up and running, so no downtime. Just trying to change the Service from Blue to Green it wont take much time. We could add a Router/Live service also that would re-direct Users to Blue or Green environment. If Green environment is stable then we rename as Blue then new Green environment will start.

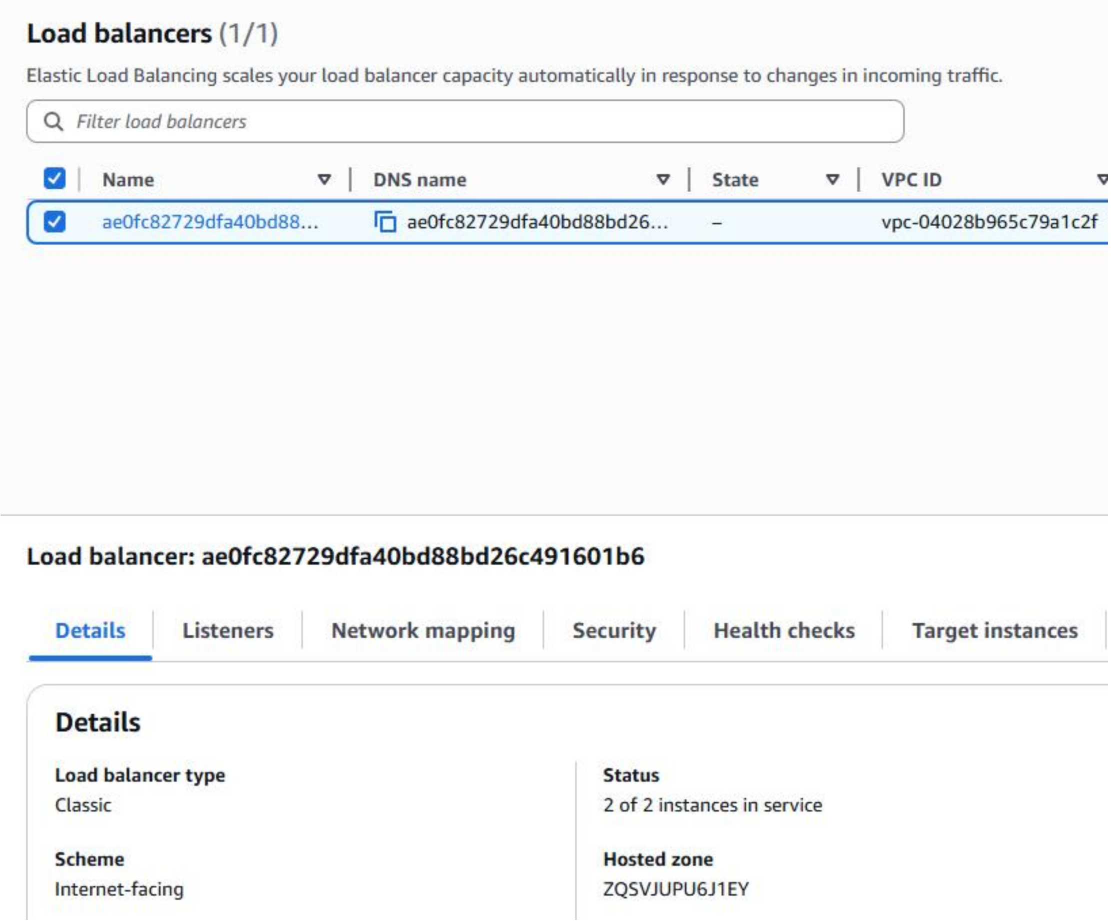
Blue-Green deployment is an application release model, which decreases risk and minimizes downtime. It uses two production environments known as Blue and Green. Old version of the application is called as the Blue environment and new version is known as Green environment. Four Yaml files are required: Blue deployment, Green deployment, Live service, Pre-prod service

```
ubuntu@ip-172-31-9-165:~$ kubectl delete deployment metrics-server -n kube-system
deployment.apps "metrics-server" deleted
ubuntu@ip-172-31-9-165:~$ kubectl delete all --all
pod "hpa-demo-deployment-7577d65cb7-8tdh7" deleted
pod "load-generator" deleted
service "hpa-demo-service" deleted
service "kubernetes" deleted
deployment.apps "hpa-demo-deployment" deleted
horizontalpodautoscaler.autoscaling "hpa-demo-hpa" deleted
```

```
buntu@ip-172-31-9-165:~$ mkdir blue-green-model
ubuntu@ip-172-31-9-165:~$ cd blue-green-model/
ubuntu@ip-172-31-9-165:~/blue-green-model$ ls -l
total 0
ubuntu@ip-172-31-9-165:~/blue-green-model$ touch blue-deployment.yml
ubuntu@ip-172-31-9-165:~/blue-green-model$ touch live-service.yml
ubuntu@ip-172-31-9-165:~/blue-green-model$ touch green-deployment.yml
ubuntu@ip-172-31-9-165:~/blue-green-model$ touch pre-pod.yml

ubuntu@ip-172-31-9-165:~/blue-green-model$ cat blue-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: javawebbluedeploy
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: java-web-app
      version: v1
      color: blue
  template:
    metadata:
      labels:
        app: java-web-app
        version: v1
        color: blue
    spec:
      containers:
        - name: javawebappcontainer
          image: hacker123shiva/springbt-in-docker:latest
          imagePullPolicy: Always
          ports:
           - containerPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ cat live-service.yml
---
apiVersion: v1
kind: Service
metadata:
    name: javawebapplivesvc
spec:
    type: LoadBalancer
    selector:
        app: java-web-app # Matches the app
        color: blue # Sends traffic to the blue pods
    ports:
        - port: 80
          targetPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f blue-deployment.yml
deployment.apps/javawebbluedeploy created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f live-service.yml
service/javawebapplivesvc created

ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
javawebbluedeploy-68fc6554d6-fftdv   1/1     Running   0          52s
javawebbluedeploy-68fc6554d6-sxgqg   1/1     Running   0          52s


ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get svc
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)
AGE
javawebapplivesvc   LoadBalancer   10.100.87.253   ae0fc82729dfa40bd88bd26c491601b6-
408506887.ca-central-1.elb.amazonaws.com   80:31359/TCP   60s
kubernetes          ClusterIP      10.100.0.1      <none>
```



Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

☑	Name	DNS name	State	VPC ID
☑	ae0fc82729dfa40bd88...	ae0fc82729dfa40bd88bd26...	–	vpc-04028b965c79a1c2f

Load balancer: ae0fc82729dfa40bd88bd26c491601b6

Details

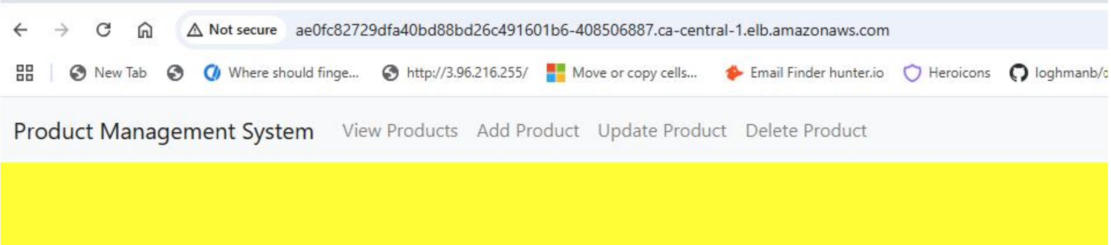
Load balancer type
Classic

Status
2 of 2 instances in service

Scheme
Internet-facing

Hosted zone
ZQSVJUPU6J1EY

http://ae0fc82729dfa40bd88bd26c491601b6-408506887.ca-central-1.elb.amazonaws.com/



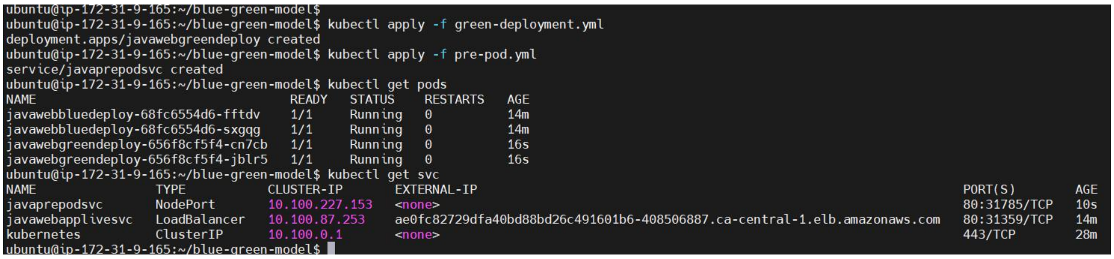
⚠ Not secure ae0fc82729dfa40bd88bd26c491601b6-408506887.ca-central-1.elb.amazonaws.com

Product Management System   View Products   Add Product   Update Product   Delete Product

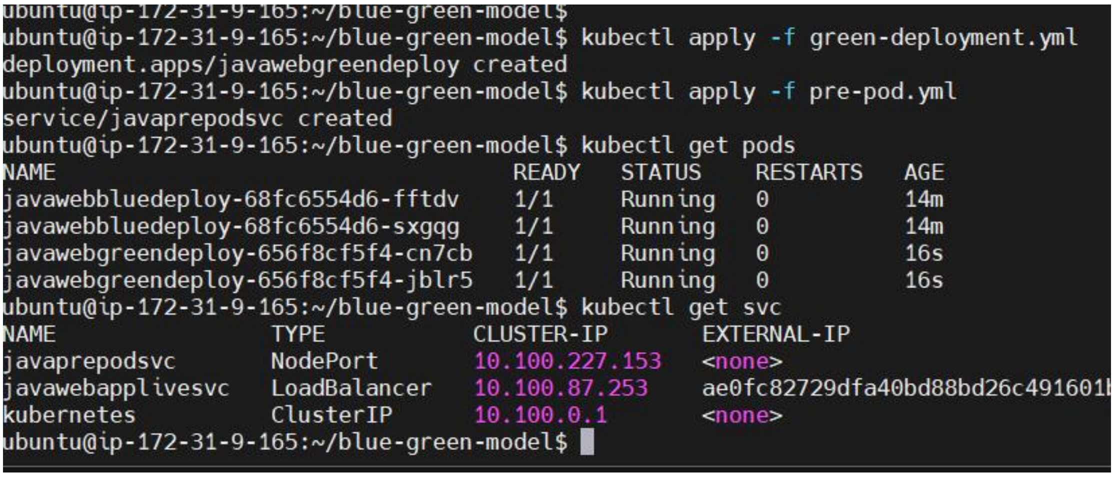
```
ubuntu@ip-172-31-9-165:~/blue-green-model$ cat green-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: javawebgreendeploy
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: java-web-app
      version: v2
      color: green
  template:
    metadata:
      labels:
        app: java-web-app
        version: v2
        color: green
    spec:
      containers:
        - name: javawebappcontainer
          image: hacker123shiva/springbt-in-docker:latest
          imagePullPolicy: Always
          ports:
           - containerPort: 8080
...

ubuntu@ip-172-31-9-165:~/blue-green-model$ cat pre-pod.yml
---
apiVersion: v1
kind: Service
metadata:
    name: javaprepodsvc
spec:
    type: NodePort
    selector:
        app: javawebapp
        color: green
    ports:
        - port: 80
          targetPort: 8080
          protocol: TCP
          nodePort: 31785 # Optional: remove this if you want auto-assign port
...

ubuntu@ip-172-31-9-165:~/blue-green-model$
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f green-deployment.yml
deployment.apps/javawebgreendeploy created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f pre-pod.yml
service/javaprepodsvc created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
javawebbluedeploy-68fc6554d6-fftdv   1/1     Running   0          14m
```

```
javawebbluedeploy-68fc6554d6-sxgqg    1/1     Running   0          14m
javawebgreendeploy-656f8cf5f4-cn7cb   1/1     Running   0          16s
javawebgreendeploy-656f8cf5f4-jblr5   1/1     Running   0          16s
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get svc
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)
AGE
javaprepodsvc      NodePort       10.100.227.153   <none>
80:31785/TCP   10s
javawebapplivesvc   LoadBalancer   10.100.87.253    ae0fc82729dfa40bd88bd26c491601b6-
408506887.ca-central-1.elb.amazonaws.com   80:31359/TCP   14m
kubernetes         ClusterIP      10.100.0.1       <none>                                                                    443/TCP
28m
```

Still Blue pods are available for Endusers. Green pods are working in the background



```
ubuntu@ip-172-31-9-165:~/blue-green-model$
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f green-deployment.yml
deployment.apps/javawebgreendeploy created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f pre-pod.yml
service/javaprepodsvc created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
javawebbluedeploy-68fc6554d6-fftdv    1/1     Running   0          14m
javawebbluedeploy-68fc6554d6-sxgqg    1/1     Running   0          14m
javawebgreendeploy-656f8cf5f4-cn7cb   1/1     Running   0          16s
javawebgreendeploy-656f8cf5f4-jblr5   1/1     Running   0          16s
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get svc
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)        AGE
javaprepodsvc       NodePort       10.100.227.153   <none>                                                                     80:31785/TCP   10s
javawebapplivesvc   LoadBalancer   10.100.87.253    ae0fc82729dfa40bd88bd26c491601b6-408506887.ca-central-1.elb.amazonaws.com   80:31359/TCP   14m
kubernetes          ClusterIP      10.100.0.1       <none>                                                                     443/TCP        28m
ubuntu@ip-172-31-9-165:~/blue-green-model$
```



```
ubuntu@ip-172-31-9-165:~/blue-green-model$
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f green-deployment.yml
deployment.apps/javawebgreendeploy created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f pre-pod.yml
service/javaprepodsvc created
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
javawebbluedeploy-68fc6554d6-fftdv    1/1     Running   0          14m
javawebbluedeploy-68fc6554d6-sxgqg    1/1     Running   0          14m
javawebgreendeploy-656f8cf5f4-cn7cb   1/1     Running   0          16s
javawebgreendeploy-656f8cf5f4-jblr5   1/1     Running   0          16s
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl get svc
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP
javaprepodsvc       NodePort       10.100.227.153   <none>
javawebapplivesvc   LoadBalancer   10.100.87.253    ae0fc82729dfa40bd88bd26c491601
kubernetes          ClusterIP      10.100.0.1       <none>
ubuntu@ip-172-31-9-165:~/blue-green-model$
```

How do we access NodePort?

Which service file I got to make changes now so Green deployment goes live instead of Blue?
live-service.yml

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ cat live-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: javawebapplivesvc
spec:
  type: LoadBalancer
  selector:
    app: java-web-app # Matches the app
     # color: blue # Sends traffic to the blue pods
    color: green
    ports:
        - port: 80
          targetPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f live-service.yml
service/javawebapplivesvc configured
```

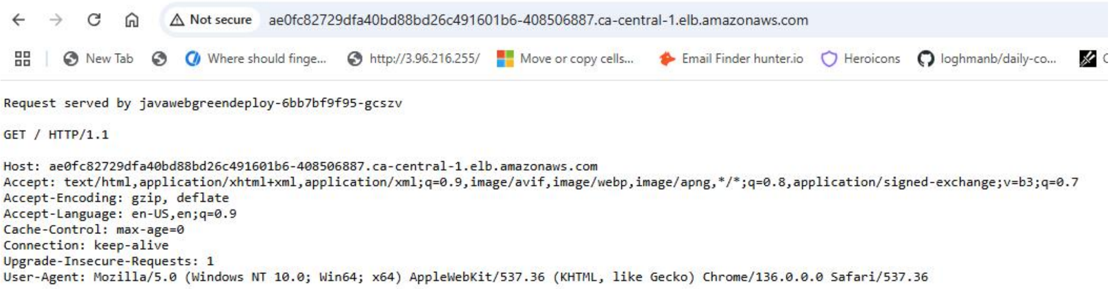
Whatever seconds it takes to re-apply that’s the only downtime we have here

Updated Docker image in green deployment so we can see which application goes live

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ cat green-deployment.yml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: javawebgreendeploy
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: java-web-app
      version: v2
      color: green
  template:
    metadata:
      labels:
        app: java-web-app
        version: v2
        color: green
    spec:
      containers:
        - name: javawebappcontainer
          image: jmalloc/echo-server
          imagePullPolicy: Always
          ports:
           - containerPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f green-deployment.yml
deployment.apps/javawebgreendeploy configured
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f live-service.yml
service/javawebapplivesvc unchanged
```

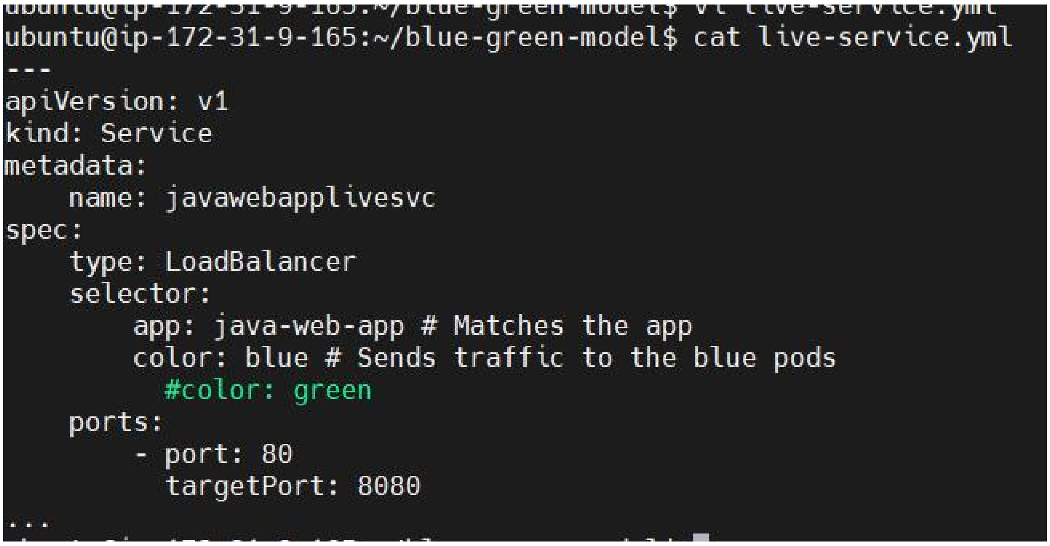
Again I go to the same LoadBalancer DNS
http://ae0fc82729dfa40bd88bd26c491601b6-408506887.ca-central-1.elb.amazonaws.com/



```
Request served by javawebgreendeploy-6bb7bf9f95-gcszv

GET / HTTP/1.1

Host: ae0fc82729dfa40bd88bd26c491601b6-408506887.ca-central-1.elb.amazonaws.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
```

Going back to Blue

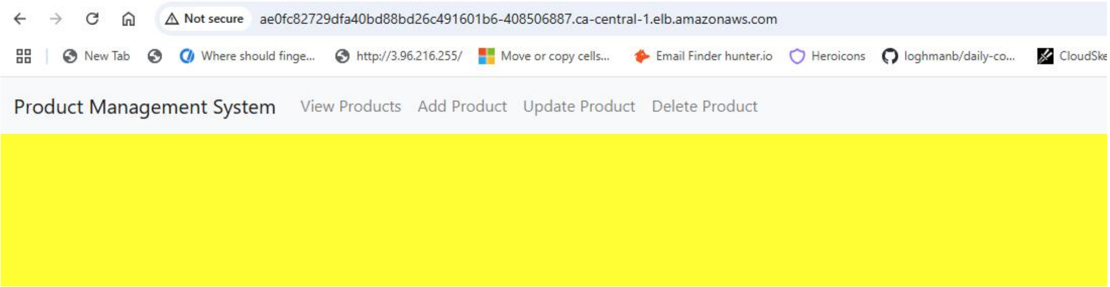
```
ubuntu@ip-172-31-9-165:~/blue-green-model$ cat live-service.yml
---
apiVersion: v1
kind: Service
metadata:
  name: javawebapplivesvc
spec:
  type: LoadBalancer
  selector:
    app: java-web-app # Matches the app
    color: blue # Sends traffic to the blue pods
      #color: green
  ports:
    - port: 80
      targetPort: 8080
...
```



```
ubuntu@ip-172-31-9-165:~/blue-green-model$ vi live-service.yml
ubuntu@ip-172-31-9-165:~/blue-green-model$ cat live-service.yml
---
apiVersion: v1
kind: Service
metadata:
    name: javawebapplivesvc
spec:
    type: LoadBalancer
    selector:
        app: java-web-app # Matches the app
        color: blue # Sends traffic to the blue pods
          #color: green
    ports:
        - port: 80
          targetPort: 8080
...
```

```
ubuntu@ip-172-31-9-165:~/blue-green-model$ kubectl apply -f live-service.yml
service/javawebapplivesvc configured
```

Now it switched back to the Old Blue environment
http://ae0fc82729dfa40bd88bd26c491601b6-408506887.ca-central-1.elb.amazonaws.com/



Delete cluster

```
eksctl delete cluster --name my-eks-cluster --region ca-central-1
```