Kubernetes\_3 notes

This one manifest file will create both service and pod

We already demonstrated LoadBalancer service

Kubernetes NodePort service

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

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: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

ubuntu@ip-172-31-9-165:~$ vi k8s-pod-svc-manifest-NodePort.yml

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-svc-manifest-NodePort.yml

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

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: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

ubuntu@ip-172-31-9-165:~$ ls -l

total 34160

drwxr-xr-x 3 ubuntu ubuntu 4096 May 16 18:46 aws

-rw-rw-r-- 1 ubuntu ubuntu 34958926 May 17 23:42 eksctl.tar.gz

-rw-rw-r-- 1 ubuntu ubuntu 458 May 18 22:39 k8s-pod-manifest-new.yml

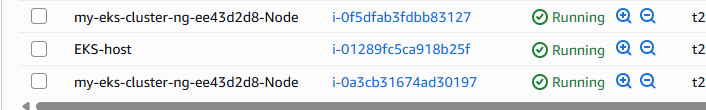
-rw-rw-r-- 1 ubuntu ubuntu 229 May 18 21:29 k8s-pod-manifest.yml

-rw-rw-r-- 1 ubuntu ubuntu 457 May 24 14:50 k8s-pod-svc-manifest-NodePort.yml

-rw-rw-r-- 1 ubuntu ubuntu 195 May 18 22:05 k8s-service-manifest.yml

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-manifest-NodePort.yml

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 k8s-pod-svc-manifest-NodePort.yml

pod/javawebapp created

service/javawebappsvc created

ubuntu@ip-172-31-9-165:~$ kubectl get pod

NAME READY STATUS RESTARTS AGE

javawebapp 1/1 Running 0 75s

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-manifest-NodePort.yml

pod/javawebapp created

service/javawebappsvc created

ubuntu@ip-172-31-9-165:~$ kubectl get pod

NAME READY STATUS RESTARTS AGE

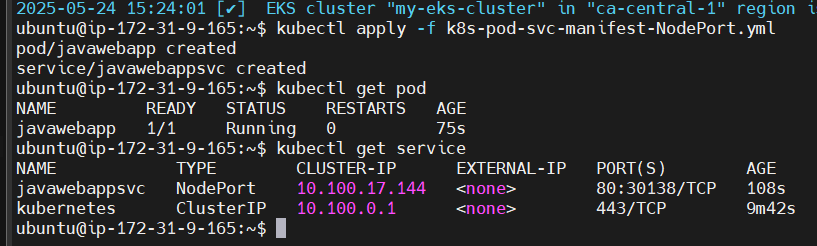
javawebapp 1/1 Running 0 75s

ubuntu@ip-172-31-9-165:~$ kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

javawebappsvc NodePort 10.100.17.144 <none> 80:30138/TCP 108s

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 9m42s



ubuntu@ip-172-31-9-165:~$ kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

javawebapp 1/1 Running 0 36m 192.168.40.10 ip-192-168-35-160.ca-central-1.compute.internal <none> <none>

ubuntu@ip-172-31-9-165:~$ kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

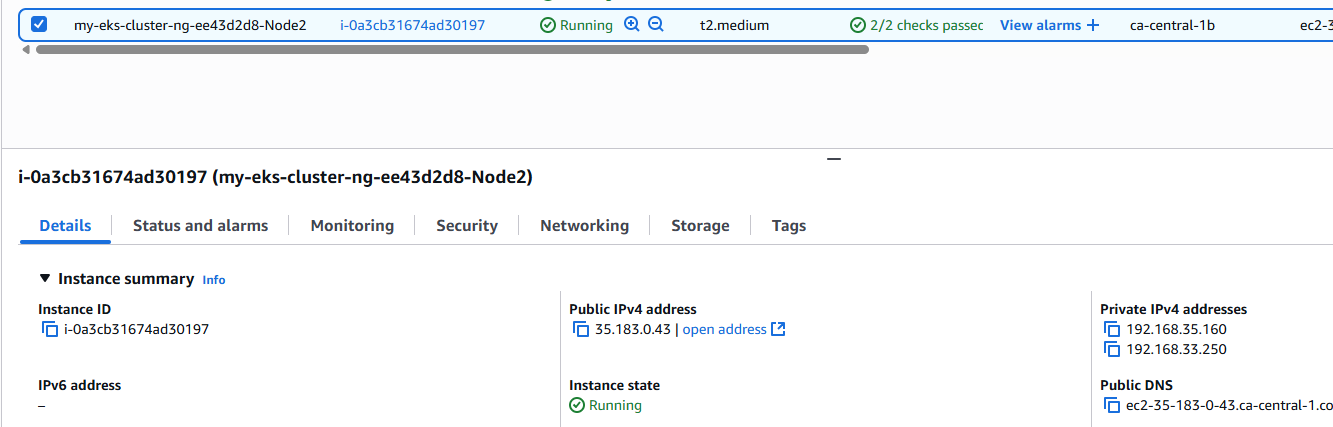
javawebappsvc NodePort 10.100.17.144 <none> 80:30138/TCP 108s

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 9m42s

A NodePort# will be assigned automatically

Using NodePort we can access our app with WorkerNode public IP address

Pod is running in Node2

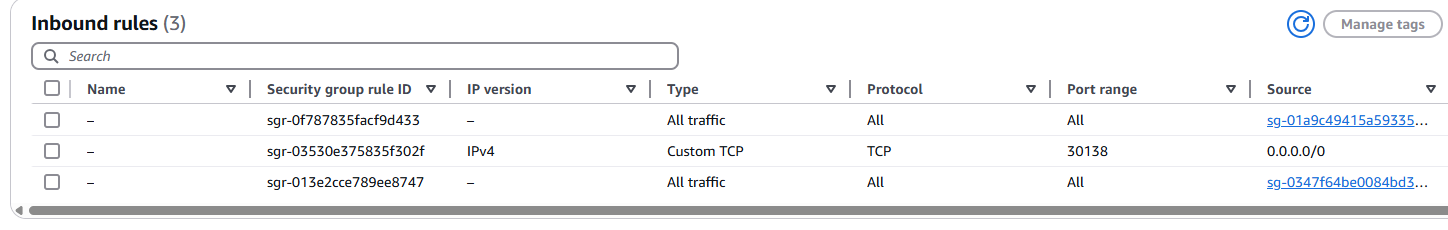


ubuntu@ip-172-31-9-165:~$ kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

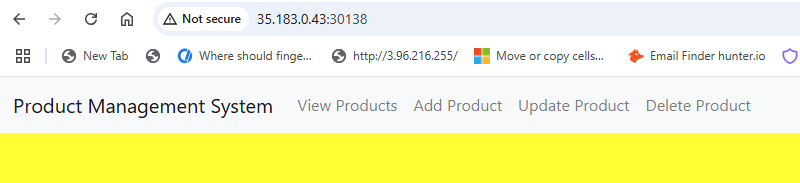
javawebapp 1/1 Running 0 36m 192.168.40.10 ip-192-168-35-160.ca-central-1.compute.internal <none> <none>

So we Add 30138 to Inbound Rules of Node2



We pick the Public IP then add Port#

<http://35.183.0.43:30138/>



NodePort can also be used to expose Application Pods

ubuntu@ip-172-31-9-165:~$ kubectl delete all --all

pod "javawebapp" deleted

service "javawebappsvc" deleted

service "kubernetes" deleted

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-manifest-NodePort.yml

pod/javawebapp created

service/javawebappsvc created

ubuntu@ip-172-31-9-165:~$ kubectl get svc

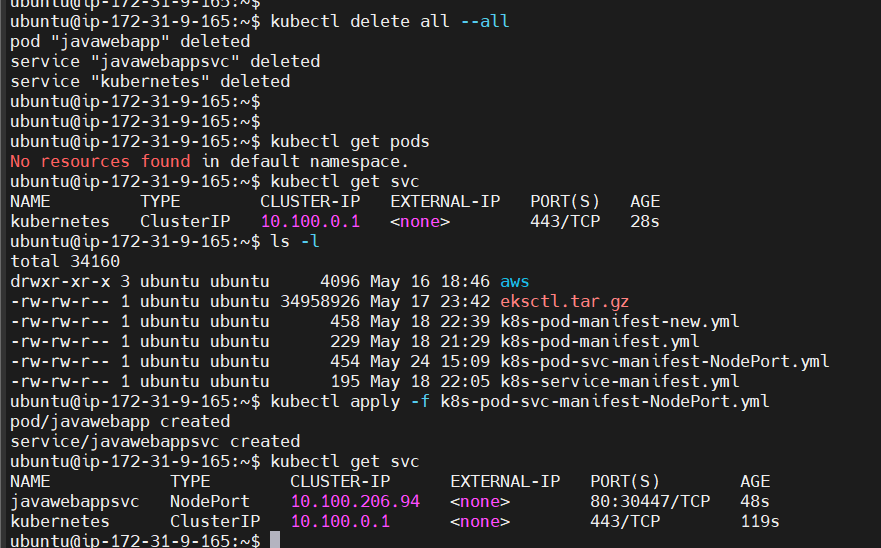
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

javawebappsvc NodePort 10.100.206.94 <none> 80:30447/TCP 48s

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 119s

This time it is a different port number

javawebappsvc NodePort 10.100.206.94 <none> 80:30447/TCP 48s



NodePort#:

If we don’t specify NodePort number in service-manifest yaml file, then Kubernetes will assign a random NodePort number for the service within the range 30000 to 32767. However, we can specify NodePort number in service-manifest yaml

kubectl apply -f <yaml> ---> Access the application using Public IP of workerNode alongside NodePort number. Also add NodePort number in Security group

Add one extra line

nodePort: 30044

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

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: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30044

...

ubuntu@ip-172-31-9-165:~$ vi k8s-pod-svc-manifest-NodePort.yml

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-svc-manifest-NodePort.yml

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

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: NodePort

selector:

app: javawebapp

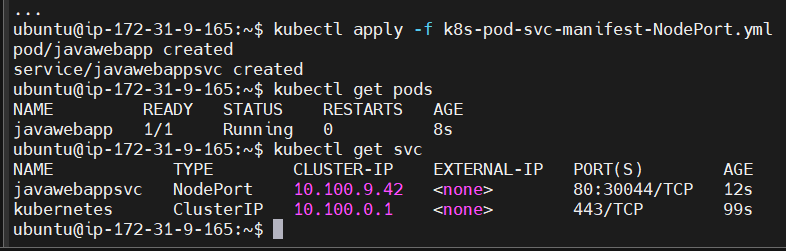
ports:

- port: 80

targetPort: 8080

nodePort: 30044

...

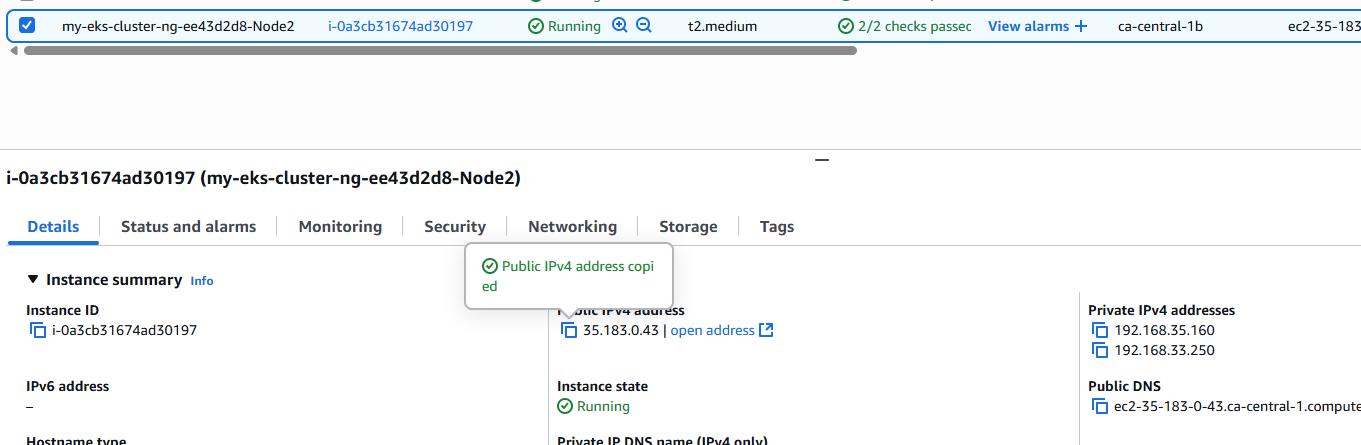


ubuntu@ip-172-31-9-165:~$ kubectl get pods -o wide

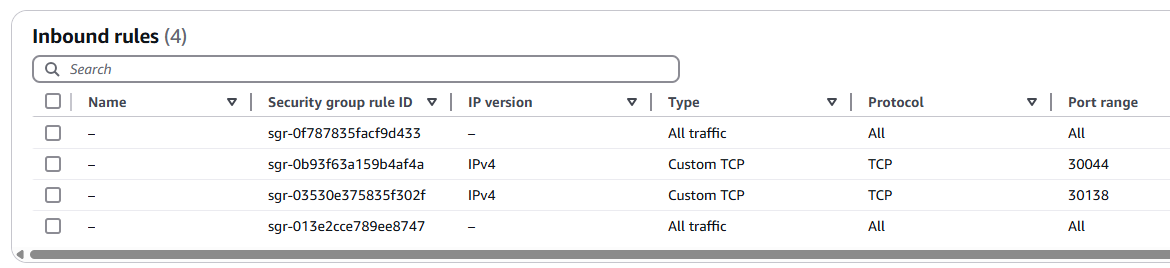
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

javawebapp 1/1 Running 0 77s 192.168.43.212 ip-192-168-35-160.ca-central-1.compute.internal <none> <none>

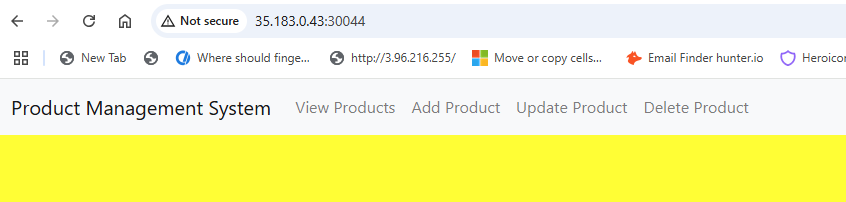
192-168-35-160 is in Node2



Security groups



<http://35.183.0.43:30044/>



ubuntu@ip-172-31-9-165:~$ kubectl delete all --all

pod "javawebapp" deleted

service "javawebappsvc" deleted

service "kubernetes" deleted

Cluster IP service

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

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: ClusterIP

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

ubuntu@ip-172-31-9-165:~$ vi k8s-pod-svc-manifest-clusterIP.yml

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-svc-manifest-clusterIP.yml

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

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: ClusterIP

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-manifest-clusterIP.yml

pod/javawebapp created

service/javawebappsvc created

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-manifest-clusterIP.yml

pod/javawebapp created

service/javawebappsvc created

ubuntu@ip-172-31-9-165:~$

ubuntu@ip-172-31-9-165:~$

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

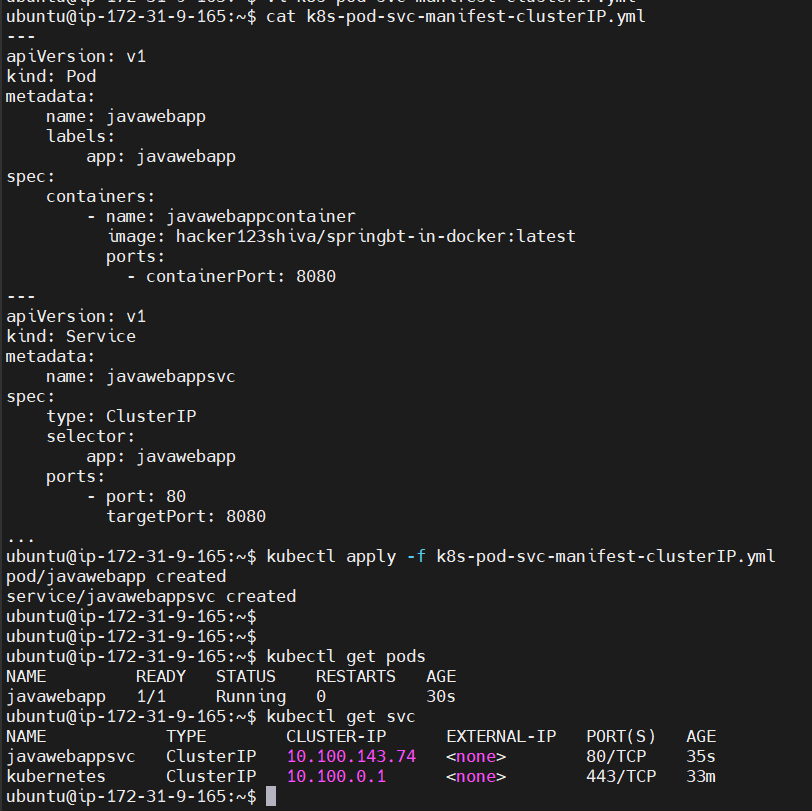
javawebapp 1/1 Running 0 30s

ubuntu@ip-172-31-9-165:~$ kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

javawebappsvc ClusterIP 10.100.143.74 <none> 80/TCP 35s

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 33m



ClusterIP will be used to access our pods within Cluster and one Static IP will be created to access the pods

ClusterIP cannot be access from outside network. So NodePort is required

ubuntu@ip-172-31-9-165:~$ kubectl delete all --all

pod "javawebapp" deleted

service "javawebappsvc" deleted

service "kubernetes" deleted

Kubetnetes Namespaces:

Why we need Namespace? To group the resources

frontend-app-pods --> frontend-app-ns

backend-app-pods --> backend-app-ns

database-pods --> database-ns

ubuntu@ip-172-31-9-165:~$ kubectl get ns

NAME STATUS AGE

default Active 4h19m

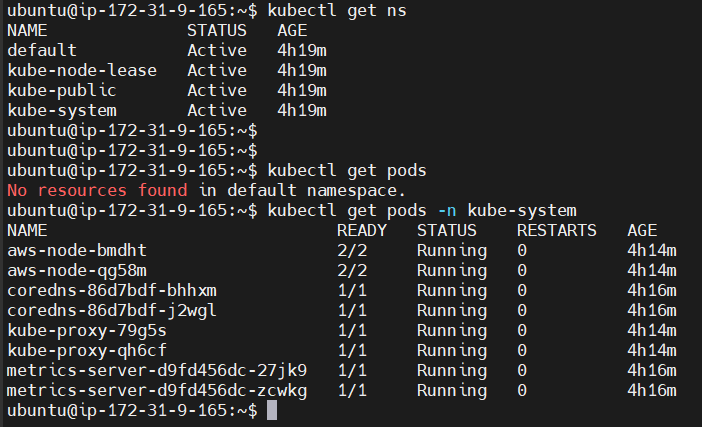
kube-node-lease Active 4h19m

kube-public Active 4h19m

kube-system Active 4h19m

ubuntu@ip-172-31-9-165:~$ kubectl get pods

No resources found in default namespace.



ubuntu@ip-172-31-9-165:~$ kubectl get pods -n kube-public

No resources found in kube-public namespace.

ubuntu@ip-172-31-9-165:~$ kubectl create ns my-namespace

namespace/my-namespace created

ubuntu@ip-172-31-9-165:~$ kubectl get ns

NAME STATUS AGE

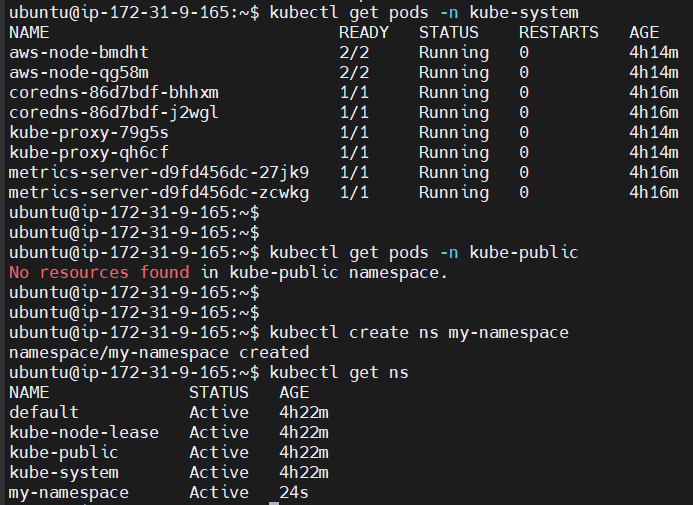
default Active 4h22m

kube-node-lease Active 4h22m

kube-public Active 4h22m

kube-system Active 4h22m

my-namespace Active 24s



We can create a namespace in Kubernetes cluster: kubectl create ns my-namespace

If needed, we can create multiple namespaces in Kubernetes cluster

To display all the available namespaces --> $ kubectl get ns

To get the pods available in specific namespace

$ kubectl get pods -n <namespace-name>

Ex: to get pods available in kube-system namespace

$ kubectl get pods -n kube-system

We can create a namespace in K8s cluster in 2 ways:

1. Using kubectl create ns command: kubectl create ns my-namespace
2. Using yml file:

-rw-rw-r-- 1 ubuntu ubuntu 195 May 18 22:05 k8s-service-manifest.yml

ubuntu@ip-172-31-9-165:~$ vi k8s-namespace.yml

ubuntu@ip-172-31-9-165:~$ cat k8s-namespace.yml

---

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace

...

ubuntu@ip-172-31-9-165:~$ cat k8s-namespace.yml

---

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace

...

ubuntu@ip-172-31-9-165:~$

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-namespace.yml

Warning: resource namespaces/my-namespace 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.

namespace/my-namespace configured

ubuntu@ip-172-31-9-165:~$ kubectl get ns

NAME STATUS AGE

default Active 5h1m

kube-node-lease Active 5h1m

kube-public Active 5h1m

kube-system Active 5h1m

my-namespace Active 38m

ubuntu@ip-172-31-9-165:~$ vi k8s-namespace.yml

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-namespace.yml

namespace/my-namespace-2 created

ubuntu@ip-172-31-9-165:~$ cat k8s-namespace.yml

---

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace-2

...

ubuntu@ip-172-31-9-165:~$ kubectl get ns

NAME STATUS AGE

default Active 5h3m

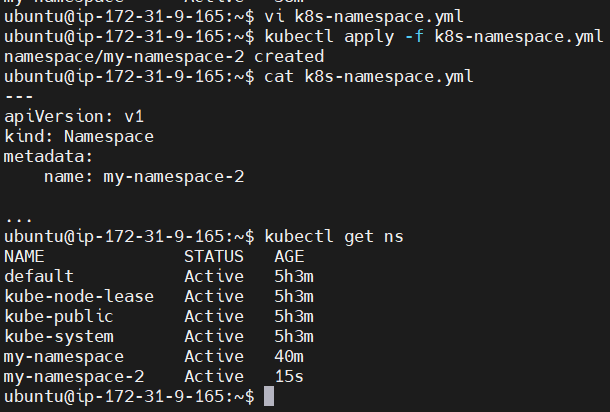
kube-node-lease Active 5h3m

kube-public Active 5h3m

kube-system Active 5h3m

my-namespace Active 40m

my-namespace-2 Active 15s



ubuntu@ip-172-31-9-165:~$ kubectl delete ns my-namespace

namespace "my-namespace" deleted

ubuntu@ip-172-31-9-165:~$ kubectl delete ns my-namespace

namespace "my-namespace" deleted

ubuntu@ip-172-31-9-165:~$ kubectl delete ns my-namespace-2

namespace "my-namespace-2" deleted

ubuntu@ip-172-31-9-165:~$ kubectl get ns

NAME STATUS AGE

default Active 5h19m

kube-node-lease Active 5h19m

kube-public Active 5h19m

kube-system Active 5h19m

---

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace-1

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

namespace: my-namespace-1

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:~$ vi k8s-pod-svc-ns.yml

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-svc-ns.yml

---

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace-1

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

namespace: my-namespace-1

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

...

We create another file due to indentation issues

ubuntu@ip-172-31-9-165:~$ cat k8s-pod-svc-ns1.yml

---

apiVersion: v1

kind: Namespace

metadata:

name: my-namespace-1

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapp

namespace: my-namespace-1

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: hacker123shiva/springbt-in-docker:latest

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

namespace: my-namespace-1

spec:

type: LoadBalancer

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-ns1.yml

namespace/my-namespace-1 unchanged

pod/javawebapp created

service/javawebappsvc created

ubuntu@ip-172-31-9-165:~$ kubectl get pods

No resources found in default namespace.

This time pods are getting created in a specific namespace and not default namespace

ubuntu@ip-172-31-9-165:~$ kubectl get ns

NAME STATUS AGE

default Active 5h46m

kube-node-lease Active 5h46m

kube-public Active 5h46m

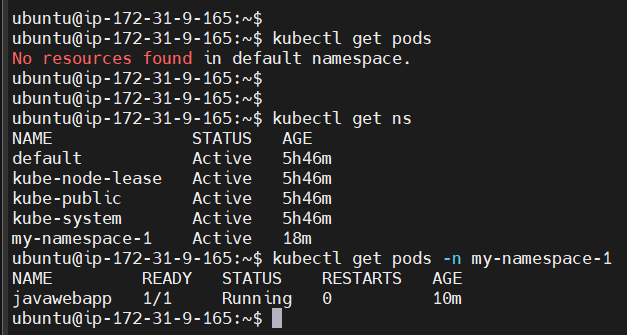
kube-system Active 5h46m

my-namespace-1 Active 18m

ubuntu@ip-172-31-9-165:~$ kubectl get pods -n my-namespace-1

NAME READY STATUS RESTARTS AGE

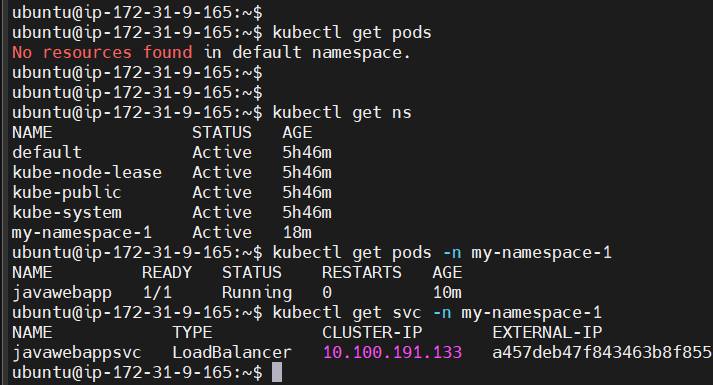
javawebapp 1/1 Running 0 10m



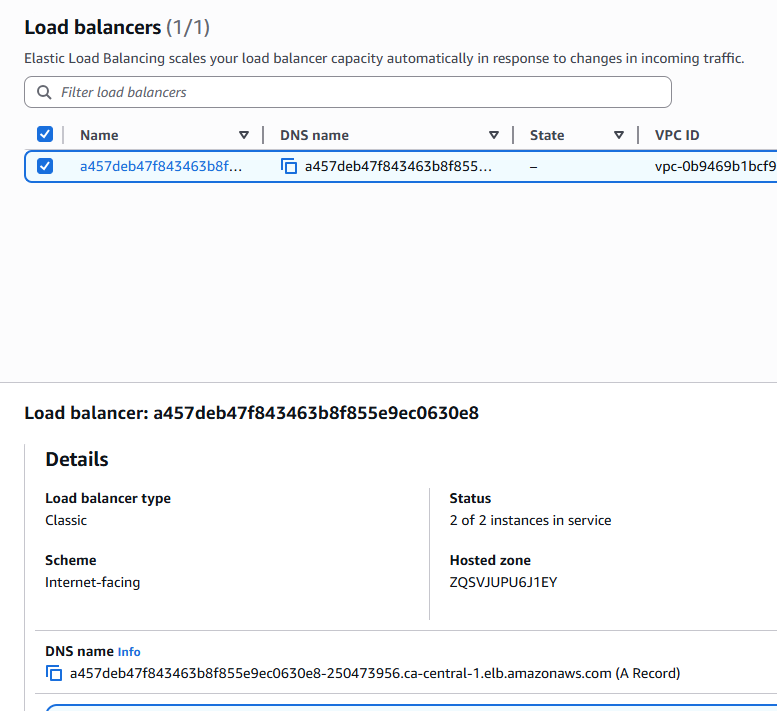
ubuntu@ip-172-31-9-165:~$ kubectl get svc -n my-namespace-1

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

javawebappsvc LoadBalancer 10.100.191.133 a457deb47f843463b8f855e9ec0630e8-250473956.ca-central-1.elb.amazonaws.com 80:30478/TCP 11m



A loadbalancer is created in EC2



ubuntu@ip-172-31-9-165:~$ kubectl get all -n my-namespace-1

NAME READY STATUS RESTARTS AGE

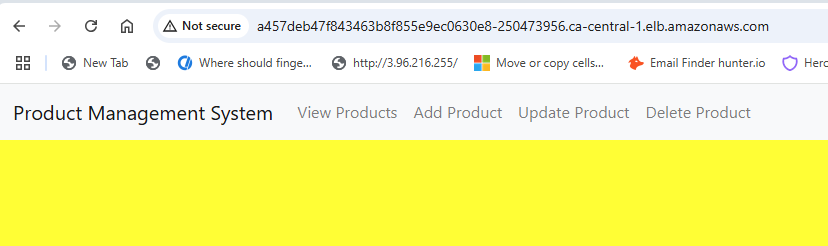
pod/javawebapp 1/1 Running 0 14m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/javawebappsvc LoadBalancer 10.100.191.133 a457deb47f843463b8f855e9ec0630e8-250473956.ca-central-1.elb.amazonaws.com 80:30478/TCP 14m

Open the DNS url

http://a457deb47f843463b8f855e9ec0630e8-250473956.ca-central-1.elb.amazonaws.com/

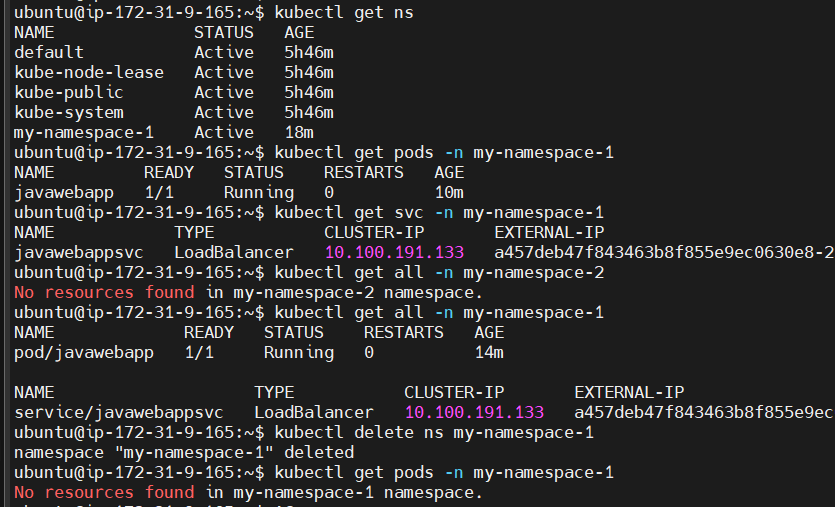


ubuntu@ip-172-31-9-165:~$ kubectl delete ns my-namespace-1

namespace "my-namespace-1" deleted

ubuntu@ip-172-31-9-165:~$ kubectl get pods -n my-namespace-1

No resources found in my-namespace-1 namespace.



$ kubectl apply -f <yml>

$ kubectl get ns

$ kubectl get pods -n <namepace-name>

$ kubectl get service -n <namepace-name>

$ kubectl get all -n <namepace-name>

$ kubectl delete ns <namepace-name>

Container orchestration --> K8s introduction --> Advantages of K8s --> Architecture --> Components of architecture --> K8s cluster setup --> K8s resources --> POD, Services (ClusterIP, NodePort, LoadBalancer, Namespace)

Advantages of K8s:

Self-healing (if any container/pod gets crashed, it will be replaced by new container)

Load-balancing

Auto-scaling

ubuntu@ip-172-31-9-165:~$ kubectl apply -f k8s-pod-svc-ns1.yml

namespace/my-namespace-1 created

pod/javawebapp created

service/javawebappsvc created

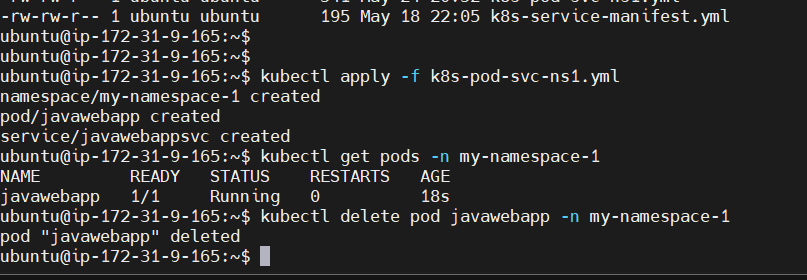
ubuntu@ip-172-31-9-165:~$ kubectl get pods -n my-namespace-1

NAME READY STATUS RESTARTS AGE

javawebapp 1/1 Running 0 18s

ubuntu@ip-172-31-9-165:~$ kubectl delete pod javawebapp -n my-namespace-1

pod "javawebapp" deleted



Now I have deleted the pod.

ubuntu@ip-172-31-9-165:~$ kubectl get pods -n my-namespace-1

No resources found in my-namespace-1 namespace.

Is it self-healing? No pods are there. No

So far we have created POD directly using POD manifest YML (kind: Pod)

If we create POD directly then we don’t get self-healing capability

If POD is damaged/crashed/deleted, then K8s will not create a new POD

In this case, if POD is damaged then our application will be down

Hence we should not create POD directly to deploy our application in K8s and we need to use the concept of K8s resources to create the PODs

If we create POD using K8s resources then Pod lifecycle will be managed by K8s

We have following resources to create PODs ->

1. ReplicationController (outdated)
2. ReplicaSet
3. Deployment
4. DaemonSet
5. StatefulSet

We have to use these methods to obtain the advantages or benefits of K8s

ReplicaSet:

It is one of the K8s resource used to create and manage Pods and ReplicaSet will take care of Pod lifecycle

When Pod is damaged/crashed/deleted, then ReplicaSet will create a new Pod

It will always maintain given number of Pod count for our application

For example, in the manifest file, if we say replicas: 2, it will maintain 2

It will help us to achieve high availability for our application. We can also scale-up and scale-down our pod count

replicaSet is a part of apps group

replicas: 2 I want 2 pods running at all the time

We are giving only 2 spacings

matchLabels: replicas for which pod

replicas : 2

$ vi replicaSet.yml

---

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: javawebrs

spec:

replicas: 2

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebpod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: hacker123shiva/springbt-in-docker:latest

ports:

- containerPort: 8080

...

ubuntu@ip-172-31-9-165:~$ cat replicaSet.yml

---

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: javawebrs

spec:

replicas: 2

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebpod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: hacker123shiva/springbt-in-docker:latest

ports:

- containerPort: 8080

...

ubuntu@ip-172-31-9-165:~$ kubectl apply -f replicaSet.yml

replicaset.apps/javawebrs created

ubuntu@ip-172-31-9-165:~$ kubectl get all

NAME READY STATUS RESTARTS AGE

pod/javawebrs-6cfpx 1/1 Running 0 11m

pod/javawebrs-76wx9 1/1 Running 0 11m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 5h22m

NAME DESIRED CURRENT READY AGE

replicaset.apps/javawebrs 2 2 2 11m



ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebrs-6cfpx 1/1 Running 0 12m

javawebrs-76wx9 1/1 Running 0 12m

2 pods are there, we specified replicas as 2

ubuntu@ip-172-31-9-165:~$ kubectl get rs

NAME DESIRED CURRENT READY AGE

javawebrs 2 2 2 13m

ubuntu@ip-172-31-9-165:~$ kubectl get all

NAME READY STATUS RESTARTS AGE

pod/javawebrs-6cfpx 1/1 Running 0 11m

pod/javawebrs-76wx9 1/1 Running 0 11m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 5h22m

NAME DESIRED CURRENT READY AGE

replicaset.apps/javawebrs 2 2 2 11m

ubuntu@ip-172-31-9-165:~$

ubuntu@ip-172-31-9-165:~$

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebrs-6cfpx 1/1 Running 0 12m

javawebrs-76wx9 1/1 Running 0 12m

ubuntu@ip-172-31-9-165:~$ kubectl get rs

NAME DESIRED CURRENT READY AGE

javawebrs 2 2 2 13m

ubuntu@ip-172-31-9-165:~$ kubectl delete pod javawebrs-6cfpx

pod "javawebrs-6cfpx" deleted

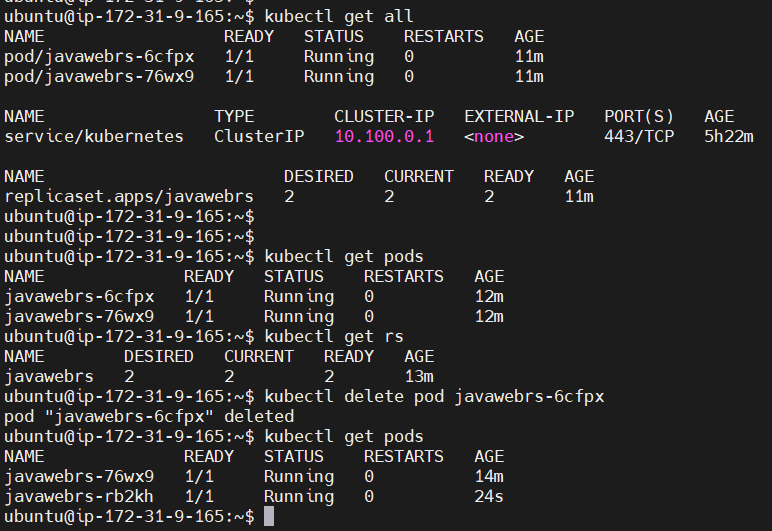
ubuntu@ip-172-31-9-165:~$ kubectl get pods

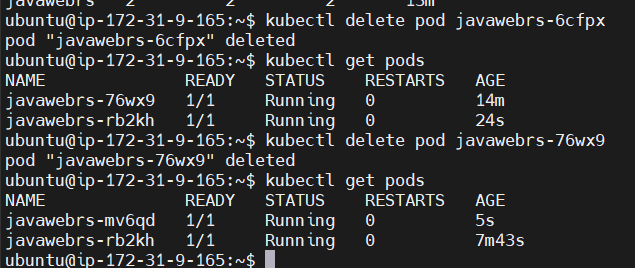
NAME READY STATUS RESTARTS AGE

javawebrs-76wx9 1/1 Running 0 14m

javawebrs-rb2kh 1/1 Running 0 24s

We deleted pod "javawebrs-6cfpx” but pod “javawebrs-76wx9” is created. So Self-healing





On run I am changing replicaSet.yml: replicas to 4

---

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: javawebrs

spec:

replicas: 4

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebpod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: hacker123shiva/springbt-in-docker:latest

ports:

- containerPort: 8080

...

Not newly created it is only configured

ubuntu@ip-172-31-9-165:~$ vi replicaSet.yml

ubuntu@ip-172-31-9-165:~$ kubectl apply -f replicaSet.yml

replicaset.apps/javawebrs configured

Now we have 4 pods

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebrs-k7tgg 1/1 Running 0 29s

javawebrs-mv6qd 1/1 Running 0 12m

javawebrs-rb2kh 1/1 Running 0 20m

javawebrs-rg9g8 1/1 Running 0 29s

ubuntu@ip-172-31-9-165:~$ kubectl scale rs javawebrs --replicas 6

ubuntu@ip-172-31-9-165:~$ kubectl scale rs javawebrs --replicas 6

replicaset.apps/javawebrs scaled

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebrs-b6ffd 1/1 Running 0 6s

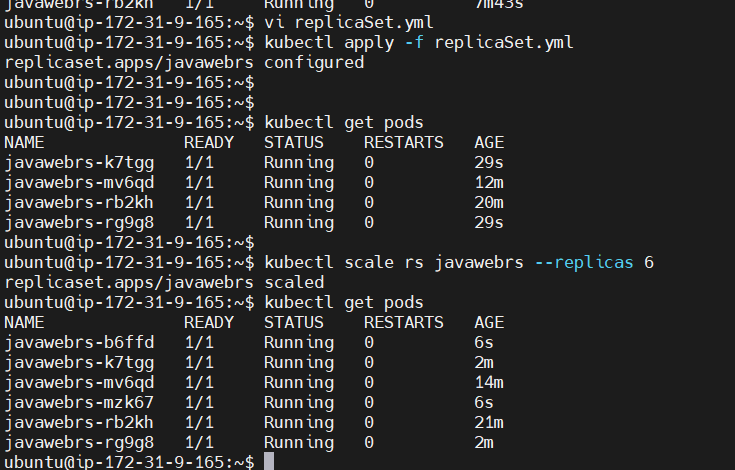
javawebrs-k7tgg 1/1 Running 0 2m

javawebrs-mv6qd 1/1 Running 0 14m

javawebrs-mzk67 1/1 Running 0 6s

javawebrs-rb2kh 1/1 Running 0 21m

javawebrs-rg9g8 1/1 Running 0 2m



Now scaled down to 2

ubuntu@ip-172-31-9-165:~$ kubectl scale rs javawebrs --replicas 2

replicaset.apps/javawebrs scaled

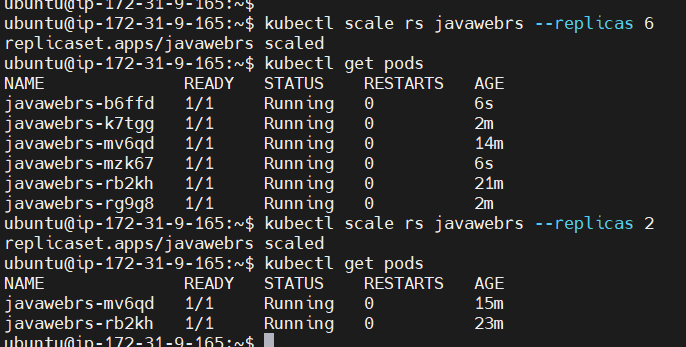
ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebrs-mv6qd 1/1 Running 0 15m

javawebrs-rb2kh 1/1 Running 0 23m

ubuntu@ip-172-31-9-165:~$



ubuntu@ip-172-31-9-165:~$ kubectl delete rs javawebrs

replicaset.apps "javawebrs" deleted

ubuntu@ip-172-31-9-165:~$ kubectl get pods

No resources found in default namespace.

If we want to delete the pods, then we have to delete the resource, which created the pods

$ kubectl delete rs javawebrs

In ReplicaSet, scale up and scale down is a manual process

K8s supports auto-scaling when we use “Deployment” resource to create Pods

K8s Deployment

It is one of the K8s resource/component recommended to deploy our application where Deployment will manage Pod lifecycle

Advantages with K8s deployment

1. Zero downtime
2. Auto scaling
3. Rolling update (it will delete and create new pods one by one) and Rollback

When I deploy application, existing pods (old pods will have old application) should be replaced by new pods with application (new pods will have new application). Rolling update means it will delete one of the old pods then create a new pod, same delete second old one replace with a new pod

We have deployment strategies:

1. Recreate (delete all existing pods and create new pods)
2. Rolling update (it will delete and create new pod one by one)

ubuntu@ip-172-31-9-165:~$ vi deployment.yml

ubuntu@ip-172-31-9-165:~$ cat deployment.yml

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: javawebrs

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

...

ubuntu@ip-172-31-9-165:~$ kubectl apply -f deployment.yml

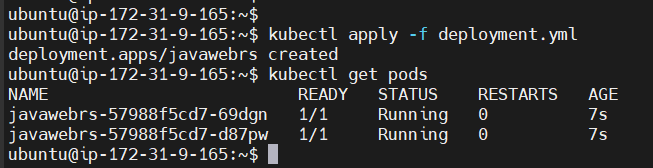
deployment.apps/javawebrs created

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebrs-57988f5cd7-69dgn 1/1 Running 0 7s

javawebrs-57988f5cd7-d87pw 1/1 Running 0 7s



ubuntu@ip-172-31-9-165:~$ kubectl get all

NAME READY STATUS RESTARTS AGE

pod/javawebrs-57988f5cd7-69dgn 1/1 Running 0 45s

pod/javawebrs-57988f5cd7-d87pw 1/1 Running 0 45s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

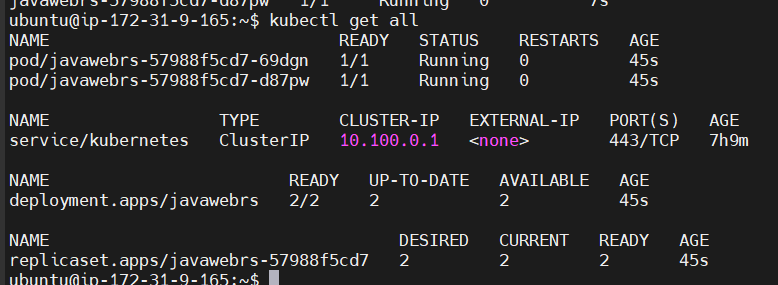
service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 7h9m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/javawebrs 2/2 2 2 45s

NAME DESIRED CURRENT READY AGE

replicaset.apps/javawebrs-57988f5cd7 2 2 2 45s



ubuntu@ip-172-31-9-165:~$ kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

javawebrs 2/2 2 2 92s

Updated yml file

ubuntu@ip-172-31-9-165:~$ cat deployment.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

...

ubuntu@ip-172-31-9-165:~$ kubectl apply -f deployment.yml

deployment.apps/javawebdeployment created

ubuntu@ip-172-31-9-165:~$ kubectl get pods

NAME READY STATUS RESTARTS AGE

javawebdeployment-57988f5cd7-6gnjl 1/1 Running 0 4s

javawebdeployment-57988f5cd7-c4dxc 1/1 Running 0 4s

ubuntu@ip-172-31-9-165:~$ kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

javawebdeployment 2/2 2 2 12s

ubuntu@ip-172-31-9-165:~$ kubectl scale deployment javawebdeployment --replicas 4

deployment.apps/javawebdeployment scaled

ubuntu@ip-172-31-9-165:~$ kubectl get all

NAME READY STATUS RESTARTS AGE

pod/javawebdeployment-57988f5cd7-2ktmv 1/1 Running 0 20s

pod/javawebdeployment-57988f5cd7-6gnjl 1/1 Running 0 65s

pod/javawebdeployment-57988f5cd7-98ggd 1/1 Running 0 20s

pod/javawebdeployment-57988f5cd7-c4dxc 1/1 Running 0 65s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 7h12m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/javawebdeployment 4/4 4 4 65s

NAME DESIRED CURRENT READY AGE

replicaset.apps/javawebdeployment-57988f5cd7 4 4 4 65s

Auto-scaling can be done in two ways: Horizontal scaling and Vertical scaling

To do more work, I have two options, give more pay to existing engineers (Vertical) or hire new engineers (Horizontal)

Vertical scaling, existing engineers should do more work and leads to less productivity so Horizontal scaling is better

Auto-scaling focuses on Horizontal scaling

Horizontal Pod Autoscaling (HPA): it is used to scale up and down number of pods/replicas based on observed metrics (CPU/memory utilization). To achieve autoscaling, we first have to create metric server. HPA will interact with this metric server to identify CPU/memory utilization, based on that Horizontal Autoscaling will be done. Creating new machines is Horizontal

kubectl delete all --all

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