HORIZONTAL POD AUTOSCALER

REFERENCE LINK:-
https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

TABLE OF CONTENTS

PREREQUISITE

For setting up HPA you must have a working kubernetes cluster with proper master and nodes server in proper up and ready state, you may need to edit the cluster to fetch metrics and for that you need to deploy metric-server deployment on to your cluster. To apply HPA follow the steps that are mentioned in this document below.

UPDATING CLUSTER CONFIGURATION

First edit the cluster configuration by using the below command:-

```
kops edit cluster
```

It will show you the cluster configuration, replace the kubelet part in the configuration with the part mentioned below:-

```
kubelet:
    anonymousAuth: false
    authenticationTokenWebhook: true
    authorizationMode: Webhook
```

Save this configuration and update your cluster with the command below:-

```
kops update cluster --name=(cluster-name) --yes
```

If asks for rolling update then update cluster with rolling upadate:-

```
kops rolling-update cluster --yes
```

NOTE:- It may take some time to update because you have have edited the cluster configuration as it may terminate the old instances and bring up the new instances on its own, so have to do nothing just wait for 5-10 minutes.

DEPLOYING METRICS SERVER ON THE CLUSTER FOR HPA

For deploying the metrics server clone the github repository on your server

```
git clone https://github.com/kubernetes-incubator/metrics-server.git
```

Now change to directory metrics-server/deploy/kubernetes

```
 cd metrics-server/deploy/kubernetes
```

You will find some yaml files in this location, edit the **metrics-server-deployment.yaml** file and replace the contents of file with contents mentioned below:-

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
 name: metrics-server
 namespace: kube-system
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: metrics-server
 namespace: kube-system
 labels:
   k8s-app: metrics-server
spec:
 selector:
   matchLabels:
     k8s-app: metrics-server
 template:
   metadata:
     name: metrics-server
     labels:
       k8s-app: metrics-server
```

```
   spec:
     serviceAccountName: metrics-server
     volumes:
     # mount in tmp so we can safely use from-scratch images and/or
read-only containers
     - name: tmp-dir
       emptyDir: {}
     containers:
     - name: metrics-server
       image: k8s.gcr.io/metrics-server-amd64:v0.3.6
       command:
         - /metrics-server
         - --v=2
         - --kubelet-insecure-tls
         - --kubelet-preferred-address-types=InternalIP

       args:
         - --cert-dir=/tmp
         - --secure-port=4443
       ports:
       - name: main-port
         containerPort: 4443
         protocol: TCP
       securityContext:
         readOnlyRootFilesystem: true
         runAsNonRoot: true
         runAsUser: 1000
       imagePullPolicy: Always
       volumeMounts:
       - name: tmp-dir
         mountPath: /tmp
     nodeSelector:
       beta.kubernetes.io/os: linux
       kubernetes.io/arch: "amd64"
```

Now execute the below command to deploy metric-server on your cluster:-

```
kubectl create -f .
```

Now wait for few seconds and execute the below commands to check your pods for metric-server

```
kubectl get pods -n kube-system
```

To check logs of metric-server

```
kubectl -n kube-system logs (mertic-server-pod-name)
```

It should start fetching metrics from nodes and pods

TESTING HORIZONTAL POD SCALER

For testing HPA we will create a deployment and service and **do not forget to add limits and resources in your deployment.yaml, because if you will skip them in your yaml then you wont be able to apply hpa.**

For example create a deployment and service.yaml as mentioned in below example:-

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example
        ports:
        - containerPort: 80
```

```
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m

---

apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

Save the above yaml file with the name that you want, in my case i have named it as **php-deployment.yaml**

Now create deployment with command:-

```
kubectl create -f php-deployment.yaml
```

CREATE HORIZONTAL POD AUTOSCALER

Now that the server is running, we will create the autoscaler using kubectl autoscale. The following command will create a Horizontal Pod Autoscaler that maintains between 1 and 10 replicas of the Pods controlled by the php-apache deployment we created in the first step of these instructions. Roughly speaking, HPA will increase and decrease the number of replicas (via the deployment) to maintain an average CPU utilization across all Pods of 50% (since each pod requests 200 milli-cores by kubectl run), this means average CPU usage of 100 milli-cores).

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

We may check the current status of autoscaler by running:

```
kubectl get hpa
```

Now wait for 1-2 minutes as it will start fetching metrics

Increase load

Now, we will see how the autoscaler reacts to increased load. We will start a container, and send an infinite loop of queries to the php-apache service (please run it in a different terminal):

```
kubectl run --generator=run-pod/v1 -it --rm load-generator --image=busybox
/bin/sh
# Hit enter for command prompt
while true; do wget -q -O- http://php-apache.default.svc.cluster.local;
done
```

Within a minute or so, we should see the higher CPU load by executing:

```
kubectl get hpa
```

```
NAME         REFERENCE                     TARGET      MINPODS   MAXPODS
REPLICAS   AGE
php-apache   Deployment/php-apache/scale   305% / 50%  1         10
1          3m
```

Here, CPU consumption has increased to 305% of the request. As a result, the deployment was resized to 7 replicas:

```
kubectl get deployment php-apache
```

```
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   7/7      7           7           19m
```