130 Horizontal Pod Autoscaler (HPA)

Scaling a workload the manual way [Horizontally].

I'm a k8s administrator & i'm sitting on my machine looking at a cluster & i'm tasked to make sure that there is always sufficient workload to support demand for this application

nginx.yaml

```
apiVersions: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        resources:
          requests:
            cpu: "250m"
          limits:
            cpu: "500m"
```

From a deployment configuration perspective, I see this pod requests 250 mille CPU and has a limit of 500 millicores of CPU. This means that 500 milli cores is the max the CPU it gets after which it doesn't get any more.

And the capacity that a single pod can handle is 500 millicores of CPU.

So, I would run the `kubectl top pod` command & monitor the resource consumption of the pod

```
kubectl top pod my-app-pod
```

| NAME | CPU (cores) | MEMORY (bytes) |
|---|---|---|
| my-app-pod | 450m | 350Mi |

Remember:-

We must have the "metrics server" running on the cluster to be able to monitor the resource usage like this

⇒ Now, when it reaches the threshold of 450 millicores or whatever it is that I've defined as the threshold, or close to that, I would run the "kubectl scale" command to scale the deployment to add additional pods

```
kubectl scale deployment my-app --replicas=3
```

The problem with this approach is we've to sit infront of the machine & continuously monitor resource usage.

I need to manually run the commands to scale up & down

And if there is sudden traffic spike & app breaks or something, I may not be able to reach fast enough to support the spike in the application as in the traffic.

To solve this problem, we use the Horizontal Pod Autoscaler

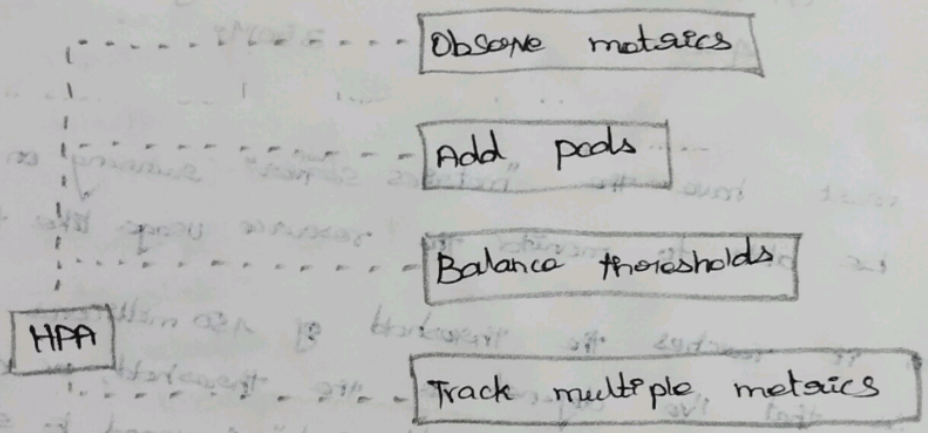
So,

The Horizontal Pod Autoscaler * Continuously monitors the metrics as we did manually using the top command

* It then automatically increases or decreases the no of pods in a deployment statefulset or replicaset based on the CPU memory or custom metrics

* And if the CPU memory or memory usage goes too high, HPA creates more pods to handle that, and if that drops it removes the extra pods to save resources & this balances the threshold & note that it can also track multiple different types of metrics

Horizontal Pod Autoscaler (HPA)




lets see this in action

⇒ For the given NGINX deployment, we can configure a Horizontal Pod Autoscaler by running the kubectl autoscale command targetting the deployment my-app, and specifying a CPU threshold of 50% with a minimum of 1 & maximum of 10 pods

Imagine that the last page "nginx.yaml" deployment file here.

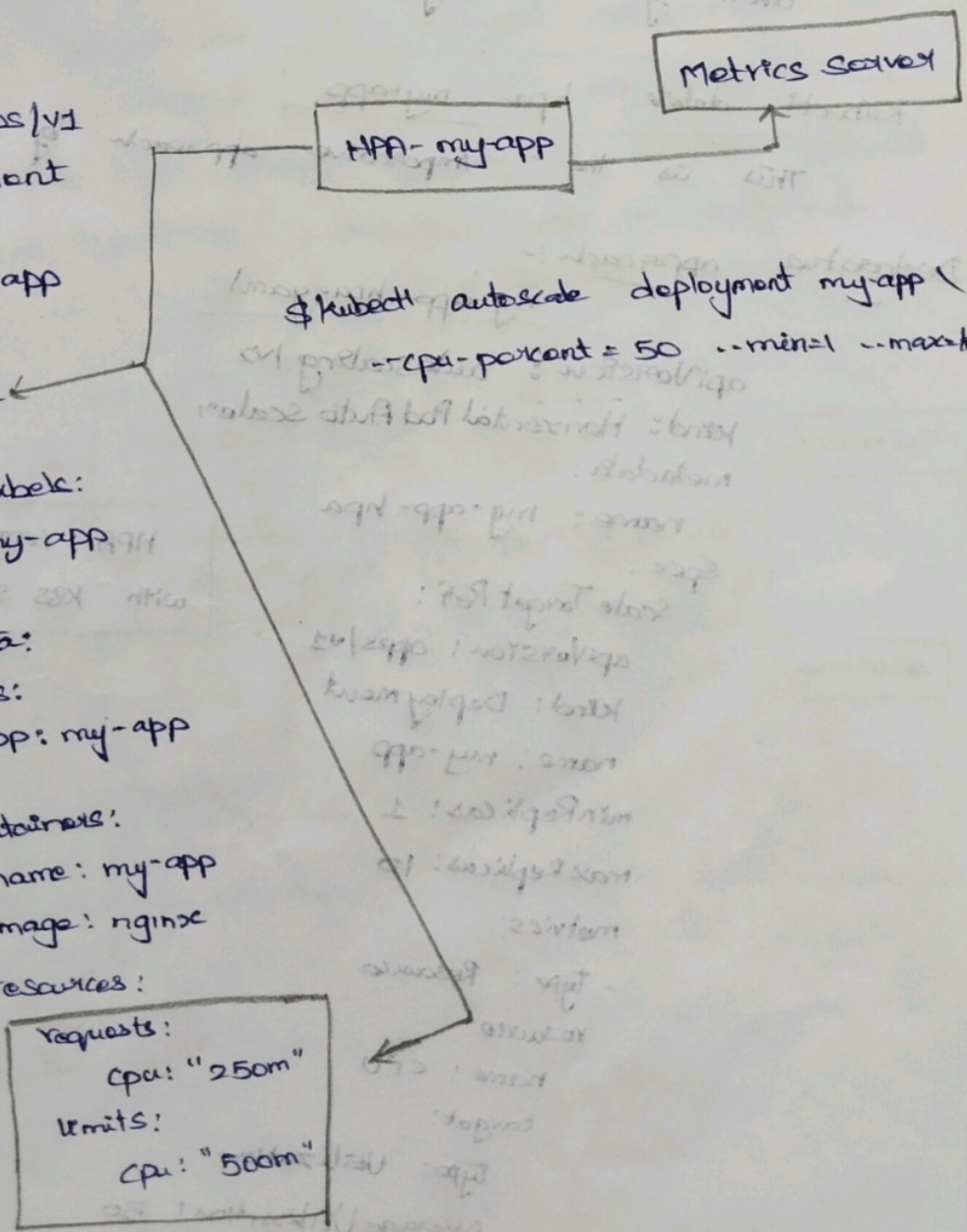
$ kubectl autoscale deployment my-app \
--cpu-percent=50 --min=1 --max=10

So when this command is run, K8s creates a Horizontal Pod Autoscaler for this deployment that first reads the limits configured on the pod, & then learns that its set to 500 millicore.

It then continuously pulls the metric server to monitor the usage. & when the usage goes beyond 50%. it modifies the no. replicas to scale up or down depending on the usage.

nginx.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: nginx
        resources:
          requests:
            cpu: "250m"
          limits:
            cpu: "500m"
```



To view the status of the created HPA

```
Kubectl get hpa
```

| NAME | REFERENCE | TARGETS | MINPODS | MAXPODS | REPLICAS |
|---|---|---|---|---|---|
| my-app | Deployment/my-app | 3% 50% | 1 | 10 | 1 |

AGE
1m

The targets column shows the current CPU usage vs the threshold value we set, & the minimum & maximum, & the current count of replicas. So it would never go beyond the maximum that we have specified when scaling up & it would never go beyond the minimum that we've specified when scaling down

when you no longer need HPA, you can delete it

```
Kubectl delete hpa my-app
```

This is the Imperative approach of creating HPA

Declarative approach :-

my-app-hpa.yaml

```
apiVersion : autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  ScaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: CPU
      target:
        type: Utilization
        averageUtilization: 50
```

HPA comes built-in with K8S since version 1.23

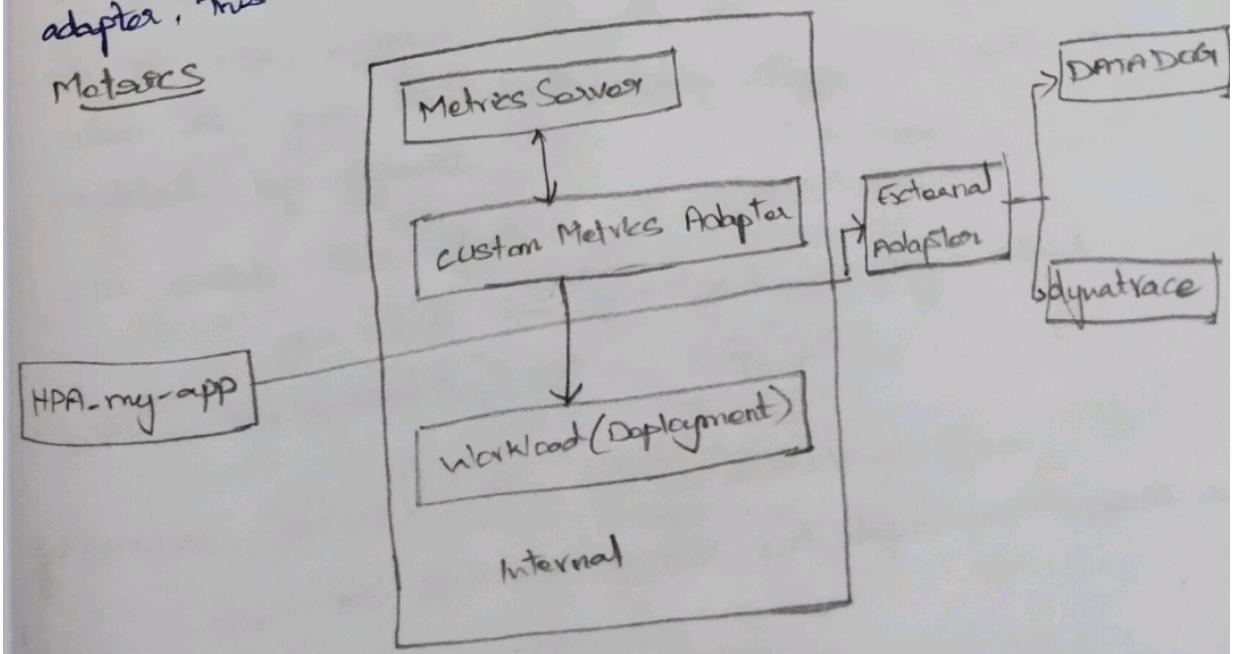
⇒ Since, HPA comes built in with K8s since version 1.23 hence there is no separate installation required

NOTE:- that it relies on metrics server, so that it is a prerequisite. So talking about metrics server, we spoke about the metrics server that HPA depends on to get cluster resource utilization numbers.

Now what we have been referring to as the internal metrics server, but there are also other resources that we can refer to such as a Custom Metrics Adapter that can retrieve information from other internal sources like a workload deployed in a cluster

However there are still internal resources we can also refer to external sources such as tools or other instances that are outside of the K8s cluster, such as a Datadog or Dynatrace instance using an external adapter. This is beyond the scope of this course.

Metrics




131 @ 132 [Do practice test]