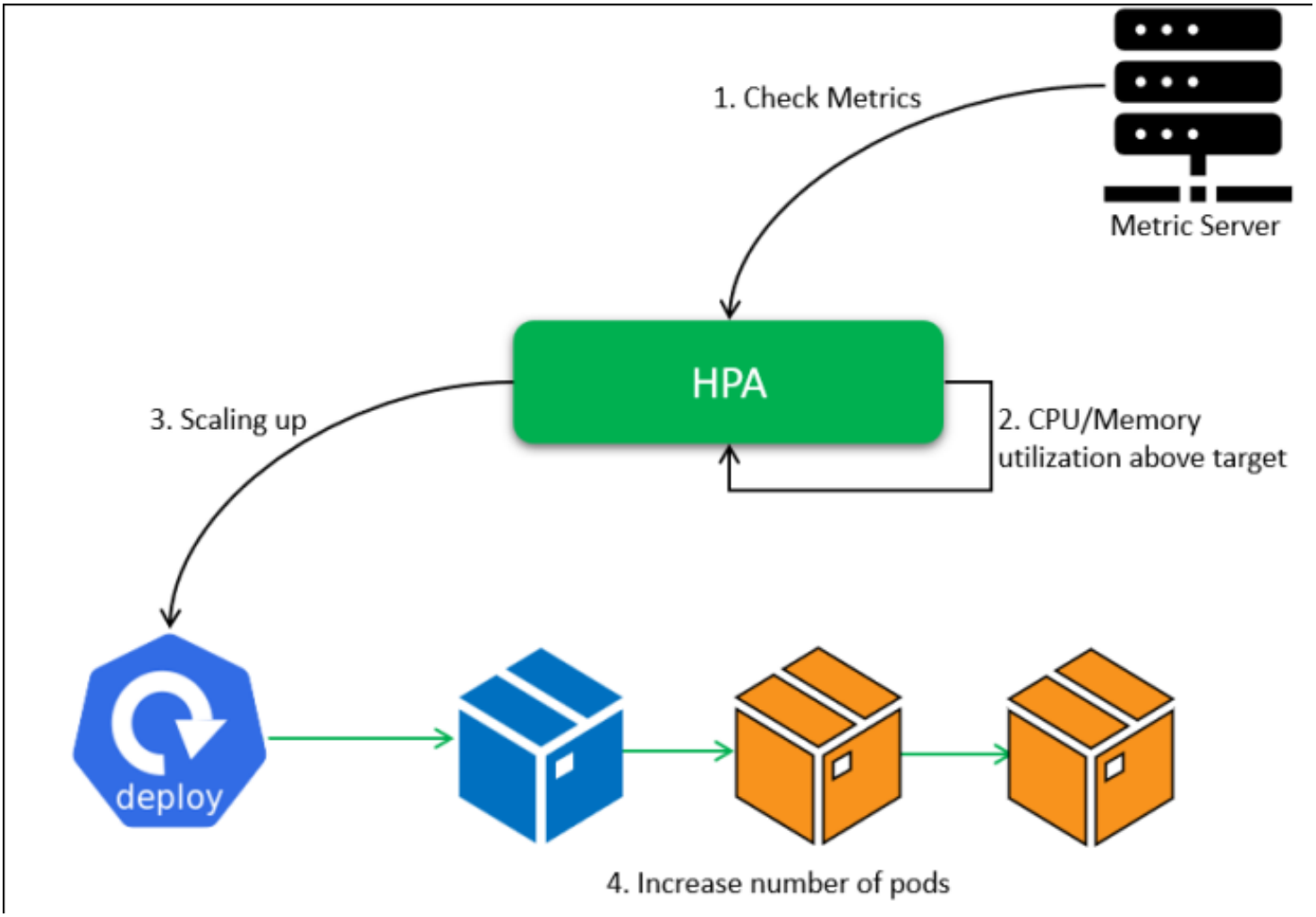
Horizontal Pod Autoscaling



In kubernetes the metric server sends metrics of resource consumption to HPA and based on the rules you have defined in HPA manifest file, this object decides to scale up or down the pods. For example, if the CPU usage was more than 80 percentage, the HPA order replica Set and deployment to scale up pods and if the usage came below 10 percentage, the additional pods will be removed.This is how Kubernetes HPA work

Ajinkya Kale

★ Practical implementation :-

Step 1: Deploy the Metrics Server

- kubectl apply -f https://github.com/kubernetessigs/metricsserver/releases/latest/download/components.yaml

```
controlplane $ kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
controlplane $
```

Step 2: Download the patch for Metrics Server

- wget -c https://gist.githubusercontent.com/initcron/1a2bd25353e1faa22a0ad41ad1c01b62/raw/008e23f9fbf4d7e2cf79df1dd008de2f1 db62a10/k8s-metrics-server.patch.yaml

```
controlplane $ wget -c https://gist.githubusercontent.com/initcron/1a2bd25353e1faa22a0ad41ad1c01b62/raw/008e23f9fbf4d7e2cf79
df1dd008de2f1db62a10/k8s-metrics-server.patch.yaml
--2025-01-27 05:40:42--  https://gist.githubusercontent.com/initcron/1a2bd25353e1faa22a0ad41ad1c01b62/raw/008e23f9fbf4d7e2cf
79df1dd008de2f1db62a10/k8s-metrics-server.patch.yaml
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.108.133, 185.199.110.133, 185.199.111.133, ...
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 205 [text/plain]
Saving to: 'k8s-metrics-server.patch.yaml'

k8s-metrics-server.patch.yaml  100%[=================================================>]     205  --.-KB/s    in 0s

2025-01-27 05:40:42 (6.42 MB/s) - 'k8s-metrics-server.patch.yaml' saved [205/205]

controlplane $
```

Step 3: Apply the Patch for deployed Metrics Server

- kubectl patch deploy metrics-server -p "$(cat k8s- metrics server.patch.yaml)" -n kube-system

```
controlplane $ ls
filesystem  k8s-metrics-server.patch.yaml  snap
controlplane $ kubectl patch deploy metrics-server -p "$(cat k8s-metrics-server.patch.yaml)" -n kube-system
deployment.apps/metrics-server patched
controlplane $
```

Step 4: Verify the Metrics Server Pod status

- kubectl get pods -n kube-system

```
controlplane $ kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS      AGE
calico-kube-controllers-94fb6bc47-4wx95    1/1     Running   2 (22m ago)   24d
canal-mfc56                                2/2     Running   2 (22m ago)   24d
canal-zstf2                                2/2     Running   2 (22m ago)   24d
coredns-57888bfdc7-6sqfr                   1/1     Running   1 (22m ago)   24d
coredns-57888bfdc7-jnrx9                   1/1     Running   1 (22m ago)   24d
etcd-controlplane                          1/1     Running   2 (22m ago)   24d
kube-apiserver-controlplane                1/1     Running   2 (22m ago)   24d
kube-controller-manager-controlplane       1/1     Running   2 (22m ago)   24d
kube-proxy-sqc72                           1/1     Running   2 (22m ago)   24d
kube-proxy-xknck                           1/1     Running   1 (22m ago)   24d
kube-scheduler-controlplane                1/1     Running   2 (22m ago)   24d
metrics-server-8dfb9988d-898rt             1/1     Running   0             98s
controlplane $
```

Step 5: Create a two deployments httpd and nginx for testing

```
controlplane $ kubectl create deployment httpd-app --image=docker.io/httpd --replicas=2
deployment.apps/httpd-app created
controlplane $ kubectl create deployment nginx-app --image=docker.io/nginx --replicas=2
deployment.apps/nginx-app created
controlplane $
```

Step 6: Check Pod Status

```
controlplane $ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
httpd-app-6c8fb74ccd-mhp9z   1/1     Running   0          110s
httpd-app-6c8fb74ccd-nkbb7   1/1     Running   0          111s
nginx-app-67bcb76dfd-rhmpk   1/1     Running   0          75s
nginx-app-67bcb76dfd-sjdb8   1/1     Running   0          75s
controlplane $
```

Step 7: Verify if the Metrics Server is working or not using the kubectl top command

```
controlplane $ kubectl top pods
NAME                         CPU(cores)   MEMORY(bytes)
httpd-app-6c8fb74ccd-mhp9z   1m           6Mi
httpd-app-6c8fb74ccd-nkbb7   1m           6Mi
nginx-app-67bcb76dfd-rhmpk   0m           2Mi
nginx-app-67bcb76dfd-sjdb8   0m           3Mi
controlplane $
```

```
controlplane $ kubectl top nodes
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
controlplane   106m         10%    1182Mi          62%
node01         41m          4%     855Mi           45%
controlplane $
```

Ajinkya Kale

★ HPA Setup with Testing

Step 8: Create Deployment YAML file

This is our main application deployment that will scale based on the load.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: php-apache
  name: php-apache
spec:
  replicas: 1
  selector:
    matchLabels:
      run: php-apache
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: php-apache
    spec:
      containers:
      - image: k8s.gcr.io/hpa-example
        name: php-apache
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 200m
            memory: 256Mi
          limits:
            cpu: 500m
            memory: 512Mi
status: {}
```

Step 9: Apply and Check the deployment pod

```
controlplane $ kubectl apply -f php-apache-dep.yml
deployment.apps/php-apache created
controlplane $ kubectl get pods -w
NAME                          READY   STATUS              RESTARTS   AGE
php-apache-554b756989-pczj8   0/1     ContainerCreating   0          9s
php-apache-554b756989-pczj8   1/1     Running             0          21s
```

Step 10: Create a Service for PHP-Apache

```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  name: php-apache
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: php-apache
status:
  loadBalancer: {}
```

```
controlplane $ kubectl apply -f php-apache-svc.yml
service/php-apache created
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   39h
php-apache   ClusterIP   10.104.196.100   <none>        80/TCP    4s
controlplane $
```

Step 11: Create Horizontal Pod Autoscaler (HPA)

Now, we define the HPA that will scale our php-apache deployment **based on CPU usage**.

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: null
  name: php-apache
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  targetCPUUtilizationPercentage: 50
status:
  currentReplicas: 0
  desiredReplicas: 0
```

Step 12: Apply the HPA and Check that the HPA is correctly configured

```
controlplane $ kubectl apply -f php-apache-hpa.yml
horizontalpodautoscaler.autoscaling/php-apache created
controlplane $ kubectl get hpa
NAME         REFERENCE                TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    cpu: 0%/50%   1         10        1          13s
controlplane $
```

Step 13: Create a Load Generator Pod

This pod generates traffic to your application to trigger scaling.

```
controlplane $ kubectl run -i --tty load-generator --image=busybox -- /bin/sh
If you don't see a command prompt, try pressing enter.
/ #
/ #
```

Now Inside the Load Generator Pod execute this following command that will continuously send requests to your php-apache service, causing CPU utilization to increase, which should trigger the HPA to scale up the pods.

- **while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done**

```
/ # while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
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!
```

Step 14: Monitor HPA status, check how the HPA is scaling your deployment

- **kubectl get hpa**

```
controlplane $ kubectl get hpa -w
NAME         REFERENCE                  TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache      cpu: 124%/50%    1         10        1          2m26s
php-apache   Deployment/php-apache      cpu: 250%/50%    1         10        3          2m36s
php-apache   Deployment/php-apache      cpu: 160%/50%    1         10        5          2m52s
php-apache   Deployment/php-apache      cpu: 139%/50%    1         10        7          3m7s
php-apache   Deployment/php-apache      cpu: 62%/50%     1         10        7          3m22s
php-apache   Deployment/php-apache      cpu: 49%/50%     1         10        7          3m37s
php-apache   Deployment/php-apache      cpu: 48%/50%     1         10        7          3m52s
```

Step 15: Also check pod scalling up or down based on the load generated by the load-generator.

```
controlplane $ kubectl get pods -w
NAME                          READY   STATUS    RESTARTS      AGE
load-generator                1/1     Running   1 (11m ago)   14m
php-apache-554b756989-2fn65   1/1     Running   0             2m21s
php-apache-554b756989-2twl2   1/1     Running   0             2m6s
php-apache-554b756989-5k5vr   1/1     Running   0             111s
php-apache-554b756989-79qdt   1/1     Running   0             65s
php-apache-554b756989-pczj8   1/1     Running   0             23m
php-apache-554b756989-pzhzs   1/1     Running   0             2m6s
php-apache-554b756989-wq45w   1/1     Running   0             65s
```

========== ==========

★ Autoscaling with CPU and Memory Metrics

Now we configure Horizontal Pod Autoscaling (HPA) with both **CPU and memory** as metrics, like in your new YAML, the HPA will monitor both **CPU and memory resource utilization**. It will scale up or down based on the average utilization of these two resources according to the thresholds you've set.

This is our new HPA Configuration YAML file

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: null
  name: php-apache
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 40
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 20
status:
  currentReplicas: 0
  desiredReplicas: 0
```

Step 1: Apply the New HPA Configuration

This will update your HPA to use both CPU and memory metrics for scaling.

```
controlplane $ vi php-apache-hpa.yml
controlplane $ kubectl apply -f php-apache-hpa.yml
horizontalpodautoscaler.autoscaling/php-apache configured
controlplane $ kubectl get hpa
NAME         REFERENCE               TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 5%/30%   1         10        1          55s
controlplane $
```

Step 2: Now exec into the load-generator pod to create load and trigger scaling

```
controlplane $ kubectl exec -it load-generator -- /bin/sh
/ # while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
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!
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!
```

Inside the load-generator pod, run the following command to continuously hit the PHP-Apache service

- **while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done**

Step 3: Now monitor the scaling process by checking the status of the HPA

- **kubectl get hpa -w**

As you can see the number of replicas increasing as CPU or memory usage exceeds the threshold. For example, if CPU usage exceeds 40%, the HPA will scale up the deployment, and the number of replicas will increase.

```
controlplane $ kubectl get hpa -w
NAME         REFERENCE               TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 5%/30%    1         10        1          14m
php-apache   Deployment/php-apache   cpu: 97%/40%, memory: 6%/30%    1         10        1          15m
php-apache   Deployment/php-apache   cpu: 250%/40%, memory: 6%/30%   1         10        3          15m
php-apache   Deployment/php-apache   cpu: 215%/40%, memory: 6%/30%   1         10        6          16m
php-apache   Deployment/php-apache   cpu: 112%/40%, memory: 5%/30%   1         10        7          16m
php-apache   Deployment/php-apache   cpu: 69%/40%, memory: 5%/30%    1         10        7          16m
php-apache   Deployment/php-apache   cpu: 52%/40%, memory: 5%/30%    1         10        7          16m
php-apache   Deployment/php-apache   cpu: 52%/40%, memory: 5%/30%    1         10        10         17m
php-apache   Deployment/php-apache   cpu: 57%/40%, memory: 5%/30%    1         10        10         17m
php-apache   Deployment/php-apache   cpu: 65%/40%, memory: 6%/30%    1         10        10         17m
php-apache   Deployment/php-apache   cpu: 28%/40%, memory: 6%/30%    1         10        10         17m
php-apache   Deployment/php-apache   cpu: 2%/40%, memory: 6%/30%     1         10        10         18m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 6%/30%     1         10        10         18m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 6%/30%     1         10        10         18m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 6%/30%     1         10        10         19m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 6%/30%     1         10        10         22m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 6%/30%     1         10        5          22m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 8%/30%     1         10        2          23m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 8%/30%     1         10        2          27m
php-apache   Deployment/php-apache   cpu: 0%/40%, memory: 5%/30%     1         10        1          27m
```

For scale down :- Once you stop the load generation, the CPU and memory usage should fall below the target thresholds, and the HPA should scale down the deployment

Simply terminate the load command as you can see beloiw

```
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!^C
/ #
command terminated with exit code 130
controlplane $ 
```