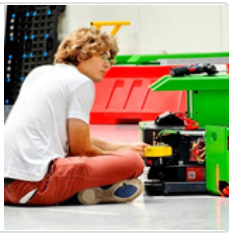
Kubernetes Autoscaling: HPA & VPA for optimal pod performance

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



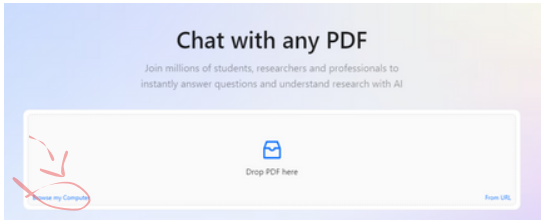
Ask Personal AI Document assistant to learn interactively (FASTER)!

1. Download PDF
2. Go to website
3. Browse file
4. Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesAutoscaleP1.pdf

Click there to go to ChatPdf website 2

3



Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes Autoscaling: Horizontal autoscaler HPA

1) What is Horizontal Pod Autoscaler (HPA)

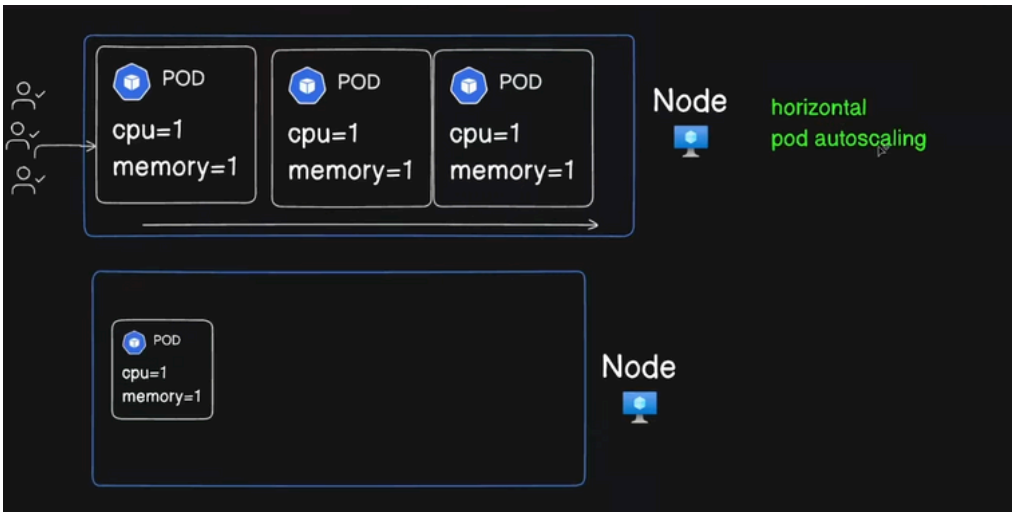
HPA adjusts the number of pods in a depncing performance and cost-efficiency as demand changes.

2) Key Benefits of HPA

- **Resource Optimization:** Scales pods based on actual needs, minimizing waste and costs.
- **Improved Reliability:** Provides resources during traffic spikes to avoid downtime.
- **Customizable Metrics:** Supports tailored scaling policies with custom metrics.loyment or replica set based on observed CPU, memory, or custom metrics

3) How Does HPA Work?

HPA monitors metrics and compares them to set thresholds. When CPU usage exceeds the target, more pods are added; when it drops below, the pod count is reduced to save resources.



4) Create deployment

Deploy metric server, so hpa could read cluster metrics:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/HPA/metrics-server.yaml
```

apply

```
kubectl apply -f metrics-server.yaml
```

Download manifest from git repo:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/HPA/Deploy.yaml
```

deployment creates php-apache containers:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
[...]
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

apply manifest:

```
kubectl apply -f Deploy.yaml
```

5) Create HPA configuration

Download manifest from git repo:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/HPA/hpa.yaml
```

hpa resource:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

Configuration scales pods to keep CPU utilization at 50% and memory utilization at 80%, between 1 and 10 replicas.

apply manifest

```
kubectl apply -f hpa.yaml
```

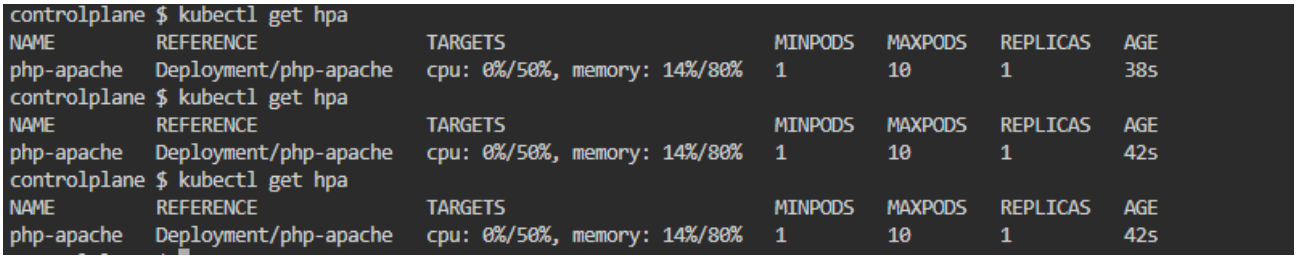
you can create HPA resource imparative way:

```
kubectl autoscale deployment php-apache \
 --cpu-percent=50 \
 --memory-percent=80 \
 --min=1 \
 --max=10
```

6) Test autoscaler

Test current memory and cpu usage on deployment:

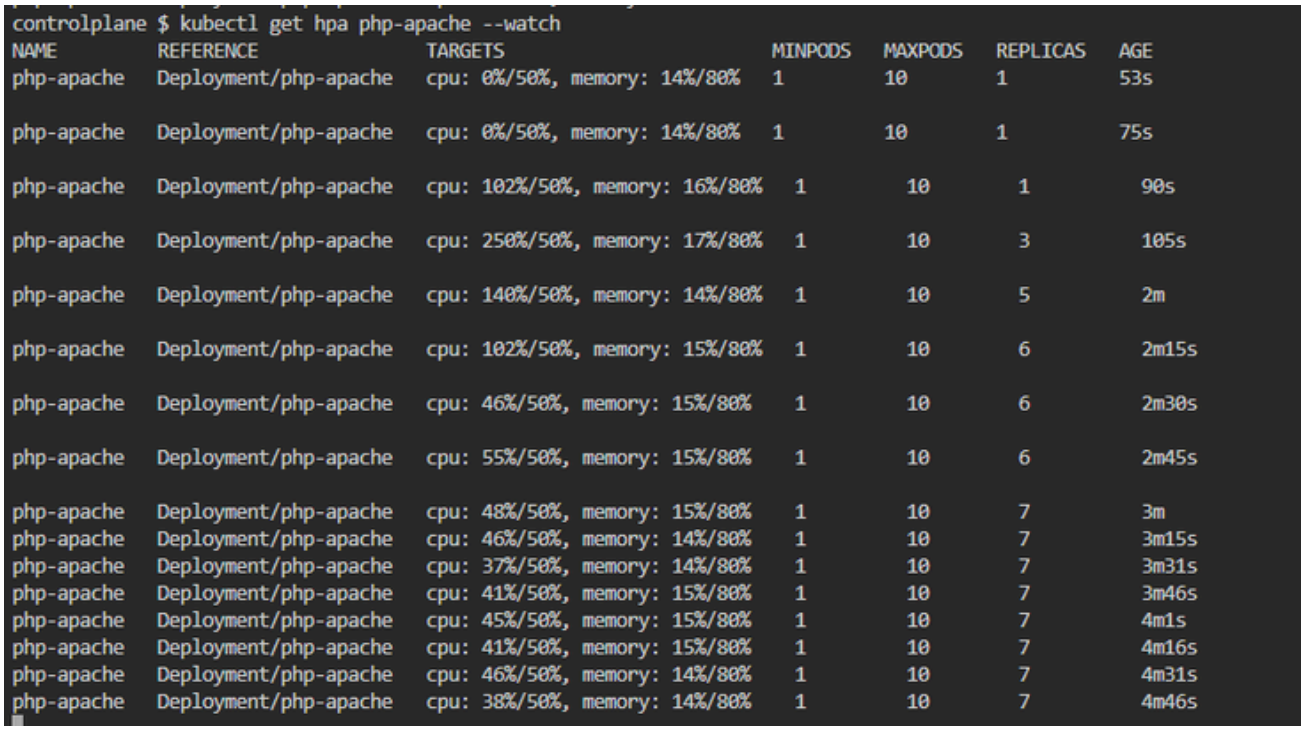
```
kubectl get hpa php-apache --watch
```



Now simulate workload:

```
kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

Output shoud look like this:



```
controlplane $ kubectl get hpa php-apache --watch
NAME         REFERENCE               TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   cpu: 0%/50%, memory: 14%/80%   1         10        1          53s

php-apache   Deployment/php-apache   cpu: 0%/50%, memory: 14%/80%   1         10        1          75s

php-apache   Deployment/php-apache   cpu: 102%/50%, memory: 16%/80%   1         10        1          90s

php-apache   Deployment/php-apache   cpu: 250%/50%, memory: 17%/80%   1         10        3          105s

php-apache   Deployment/php-apache   cpu: 140%/50%, memory: 14%/80%   1         10        5          2m

php-apache   Deployment/php-apache   cpu: 102%/50%, memory: 15%/80%   1         10        6          2m15s

php-apache   Deployment/php-apache   cpu: 46%/50%, memory: 15%/80%    1         10        6          2m30s

php-apache   Deployment/php-apache   cpu: 55%/50%, memory: 15%/80%    1         10        6          2m45s

php-apache   Deployment/php-apache   cpu: 48%/50%, memory: 15%/80%    1         10        7          3m
php-apache   Deployment/php-apache   cpu: 46%/50%, memory: 14%/80%    1         10        7          3m15s
php-apache   Deployment/php-apache   cpu: 37%/50%, memory: 14%/80%    1         10        7          3m31s
php-apache   Deployment/php-apache   cpu: 41%/50%, memory: 15%/80%    1         10        7          3m46s
php-apache   Deployment/php-apache   cpu: 45%/50%, memory: 15%/80%    1         10        7          4m1s
php-apache   Deployment/php-apache   cpu: 41%/50%, memory: 15%/80%    1         10        7          4m16s
php-apache   Deployment/php-apache   cpu: 46%/50%, memory: 14%/80%    1         10        7          4m31s
php-apache   Deployment/php-apache   cpu: 38%/50%, memory: 14%/80%    1         10        7          4m46s
```

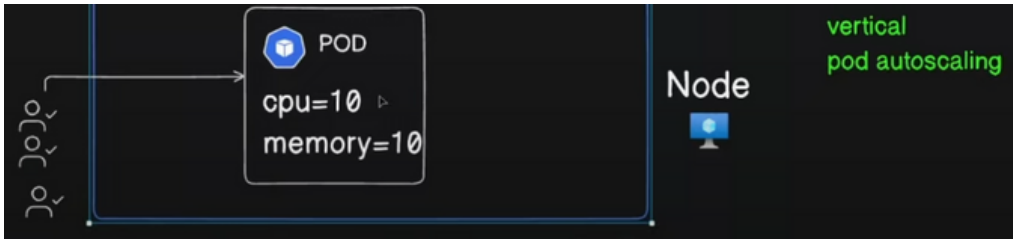
Autoscaler keept creating new pods until cpu usage stabilized around 50%, as was indicated in hpa.yml

Kubernetes Autoscaling: Vertical autoscaler VPA

1) What is VPA (Vertical Pod Autoscaler)

Kubernetes tool designed to optimize the resource allocation of pods by adjusting their CPU and memory requests and limits.

Before VPA reaction to workload



After VPA reaction to workload

2) How it works?

VPA analyzes historical and real-time resource usage and makes recommendations or directly modifies the pod specifications. It has four modes that control its behavior:

1) Deploy VPA

Deploy metric server as shown in HPA example. Then get modified deployment (deleted resource limits and requests definition, hence VPA will do it automatically.

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/VPA/Deploy.yaml
```

apply

```
kubectl apply -f Deploy.yaml
```

Install VPA on you k8s cluster:

```
git clone https://github.com/kubernetes/autoscaler.git
```

Then go into vertical-pod-autoscaler dir and execute

```
./hack/vpa-up.sh
```

Get VPA autoscaler from my git

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/VPA/vpa.yaml
```

it looks like:

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: php-apache-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: php-apache
  updatePolicy:
    updateMode: "Auto" # Change to "Off", "Initial", or "Recreate" as needed
```

2) VPA modes

VPA has four modes that control its behavior:

Off Mode

- Purpose: Collects data, recommends resources, but doesn't change pod configurations.
- Use Case: Evaluate resource needs without modifications.

Initial Mode

- Purpose: Sets resource values only at pod creation.
- Use Case: Optimize resources for new pods without affecting running ones.

Auto Mode

- Purpose: Dynamically adjusts resources by evicting and recreating pods.
- Use Case: For workloads with variable resource demands.

Recreate Mode

- Purpose: Updates resources when pods are recreated, not via forced evictions.
- Use Case: For controlled updates without disrupting running pods.

3) Auto Mode demo

apply vpa

```
kubectl apply -f vpa.yaml
```

After minute, check if vpa calculated optimised resources for pods in deployment:

```
kubectl get vpa
```

```
controlplane $ kubectl get vpa
NAME             MODE   CPU   MEM       PROVIDED   AGE
php-apache-vpa   Auto   25m   262144k   True       11m
controlplane $
```

As seen on image, vpa provided CPU and RAM to pods. Check what exact configuration was calculated:

```
kubectl describe verticalpodautoscaler php-apache-vpa
```

```
  Recommendation:
    Container Recommendations:
      Container Name:  php-apache
      Lower Bound:
        Cpu:     25m
        Memory:  262144k
      Target:
        Cpu:     25m
        Memory:  262144k
      Uncapped Target:
        Cpu:     25m
        Memory:  262144k
      Upper Bound:
        Cpu:     25m
        Memory:  262144k
```

Lower Bound: The minimum necessary resources (25m CPU and 262144k memory) to ensure the container runs without performance issues.

Target: The optimal resource values for efficient operation, which in this case are the same as the lower bound. (This is set on pods)

Uncapped Target: The ideal resource allocation, ignoring constraints like resource limits, but still set to the same values.

Upper Bound: The maximum resource allocation, also set to the same values, to prevent over-provisioning.

4) Important about HPA and VPA

The Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) both adjust resources in Kubernetes, but they are not designed to work together. HPA scales the number of pod replicas based on metrics like CPU and memory usage, while VPA adjusts the resource requests and limits for individual pods.

Using both autoscalers simultaneously can cause conflicts, as HPA changes the number of replicas, and VPA alters pod resources, leading to potential issues. Kubernetes may address this integration in future releases.

common troubleshooting

1) HPA Not Scaling Pods

Cause: Incorrect resource requests/limits or Metrics Server issues.
Solution: Ensure correct resource settings and verify the Metrics Server is running with kubectl get deployment metrics-server -n kube-system.

2) VPA Not Adjusting Resources

Cause: Conflicting resource settings in the deployment.
Solution: Check VPA configuration and avoid manual resource overrides. Use kubectl describe vpa <vpa-name> for troubleshooting.

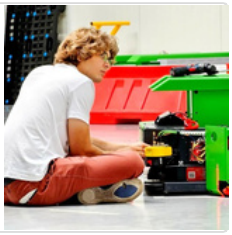
3) HPA and VPA Conflicting

Cause: HPA scales pods, VPA adjusts resources, leading to conflicts.
Solution: Avoid using both together or disable one with --vpa-exempt on HPA.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

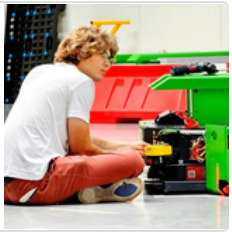
Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!