**Autoscaling**

**Horizontal Pod Autoscaler (HPA) vs. Vertical Pod Autoscaler (VPA)**

Both HPA and VPA are used to scale workloads in Kubernetes, but they serve different purposes.

A screenshot of a computer

AI-generated content may be incorrect.

**📌 Real-Time Use Case for HPA**

**Scenario: Auto-scaling an Nginx Deployment Based on CPU Usage**

Imagine you have an **Nginx-based web application** running in Kubernetes.

* When traffic increases, CPU usage spikes.
* To handle the load, **HPA increases the number of pods** dynamically.
* When the load decreases, it **removes pods** to optimize resource usage.

**🔹 Step-by-Step Implementation of HPA**

**1️.Create a Nginx Deployment**

apiVersion: "apps/v1"

kind: "Deployment"

metadata:

name: "buggycpu"

namespace: "default"

labels:

app: "buggycpu"

spec:

replicas: 1

selector:

matchLabels:

app: "buggycpu"

template:

metadata:

labels:

app: "buggycpu"

spec:

containers:

- name: buggycpu

image: sumanth17121988/buggyapp:v1

resources:

limits:

cpu: "1"

memory: "1024Mi"

requests:

cpu: "500m"

memory: "512Mi"

kubectl apply -f nginx-deployment.yaml

kubectl expose deployment nginx-deployment --type=ClusterIP --port=80

kubectl apply -f <https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml>

kubectl get deployment -n kube-system | grep metrics-server

**Create an HPA Resources**

apiVersion: "autoscaling/v2"

kind: "HorizontalPodAutoscaler"

metadata:

name: "buggycpu-hpa"

namespace: "default"

labels:

app: "buggycpu"

spec:

scaleTargetRef:

kind: "Deployment"

name: "buggycpu"

apiVersion: "apps/v1"

minReplicas: 1

maxReplicas: 3

metrics:

- type: "Resource"

resource:

name: "cpu"

target:

type: "Utilization"

averageUtilization: 80

**kubectl apply -f hpa.yml**

**📌 Real-Time Use Case for VPA**

**Scenario: Auto-Adjusting Resource Limits for an Application**

* **Your app runs a Python web service that is consuming high memory.**
* **Instead of scaling out, VPA increases pod memory limits to optimize performance.**

**kubectl apply -f** [**https://github.com/kubernetes/autoscaler/releases/latest/download/vertical-pod-autoscaler.yaml**](https://github.com/kubernetes/autoscaler/releases/latest/download/vertical-pod-autoscaler.yaml)

apiVersion: autoscaling.k8s.io/v1

kind: VerticalPodAutoscaler

metadata:

name: python-vpa

spec:

targetRef:

apiVersion: "apps/v1"

kind: Deployment

name: python-app

updatePolicy:

updateMode: "Auto"

**kubectl apply -f python-vpa.yaml**

**kubectl describe vpa python-vpa**

**How updateMode: "Auto" Works?**

1. VPA continuously monitors CPU and memory usage of the pods.
2. If the usage exceeds the current limits, VPA calculates new optimal values.
3. VPA evicts the pod (i.e., deletes and restarts it with new limits).
4. The pod restarts with updated resource requests/limits.

**Example Use Case:**

* **You have a Java application running with 100m CPU and 200Mi memory.**
* **VPA detects that it needs more memory (e.g., 400Mi).**
* **If updateMode: "Auto", VPA terminates and restarts the pod with the updated memory limit.**

**Impact of Auto Mode:**

**✅ Ensures the pod always runs with optimal resource allocation.  
✅ Reduces wasted resources by continuously adjusting limits.  
⚠️ Can cause disruptions because pods will be restarted when resource recommendations change.**