Kubernetes Horizontal Pod Autoscaler (HPA) – Detailed Tutorial

This tutorial explains **Horizontal Pod Autoscaling (HPA)** in Kubernetes using **CPU-based autoscaling**, step by step, with **hands-on commands**, **why each step is required**, and **what to observe**.

What is HPA?

Horizontal Pod Autoscaler (HPA) automatically increases or decreases the number of pod replicas in a deployment based on resource usage (CPU/memory) or custom metrics.

In this tutorial:

- Metric: CPU utilization
- Scaling type: Horizontal (pods)
- Trigger: CPU > 50%

Architecture Overview

```
User Load
   ↓
Service
   ↓
Pods (Deployment)
   ↓
Metrics Server → HPA Controller
   ↓
Replica Scale Up / Down
```

Step 1: Deploy Metrics Server (Mandatory)

Why Metrics Server?

HPA depends on **live resource metrics** (CPU/memory). Without Metrics Server:

- HPA shows `0%` CPU
- Scaling does NOT happen

Install Metrics Server

```
kubectl apply -f https://github.com/vilasvarghese/docker-k8s/blob/master/yaml/metricServer/metric-server.ya
```

Verify installation:

```
kubectl get pods -n kube-system | grep metrics-server
kubectl top nodes
```

- If metrics are visible, Metrics Server is working.

Step 2: Create a Deployment

Purpose

This deployment runs a CPU-intensive sample application used for HPA testing.

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpa-demo-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      run: hpa-demo-deployment
  template:
    metadata:
      labels:
        run: hpa-demo-deployment
    spec:
      containers:
```

```
    - name: hpa-demo-deployment
      image: k8s.gcr.io/hpa-example
      ports:
      - containerPort: 80
      resources:
        requests:
          cpu: 200m
        limits:
          cpu: 500m
```

Apply deployment:

```
kubectl apply -f deployment.yml
kubectl get deploy
```

Why CPU Requests Are Important

HPA calculates utilization as:

```
CPU Usage / CPU Request
```

⚠️ If `requests.cpu` is missing → HPA will NOT work.

Step 3: Create a Service

Why a Service?

The Service exposes the pods internally and provides a stable endpoint for load generation.

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hpa-demo-deployment
  labels:
    run: hpa-demo-deployment
spec:
  ports:
  - port: 80
  selector:
    run: hpa-demo-deployment
```

Apply service:

```
kubectl apply -f service.yaml
kubectl get svc
```

Step 4: Create the Horizontal Pod Autoscaler

What HPA Does Here

- Monitors CPU usage
- Scales pods between **1 and 10**
- Targets **50% CPU utilization**

hpa.yaml

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-demo-deployment
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hpa-demo-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Apply HPA:

```
kubectl apply -f hpa.yaml
watch -n 1 kubectl get hpa
```

Expected Output (Initially)

```
TARGETS: 0%/50%
```

⚠️ CPU shows `0%` until Metrics Server starts reporting data.

Step 5: Generate Load

Purpose

Simulate user traffic to increase CPU utilization.

Load Generator Command

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

Observe Scaling

```
kubectl get hpa
kubectl get hpa -w
watch -n 1 kubectl get deployment hpa-demo-deployment
```

What Happens Internally

1. CPU usage crosses 50%
2. Metrics Server reports usage
3. HPA controller increases replicas
4. New pods are created

Step 6: Monitor Metrics and Events

View Pod CPU Usage

```
kubectl top pods --all-namespaces
```

Describe Deployment

```
kubectl describe deploy hpa-demo-deployment
```

Look for:

- Replica count changes
- Scaling events

Step 7: Decrease the Load

Stop the load generator (`Ctrl+C`).

Observe Scale Down

```
kubectl get hpa
kubectl get deployment hpa-demo-deployment
kubectl get events
```

Scale Down Behavior

- HPA waits before scaling down (cooldown period)
- Gradual reduction of replicas

Common Issues & Fixes

Issue	Cause	Fix
CPU shows 0%	Metrics Server missing	Install metrics-server
No scaling	CPU request missing	Add requests.cpu
HPA not working	Wrong selector	Match labels

Key Takeaways

- Metrics Server is mandatory for HPA

- CPU requests are required
- HPA reacts to **average pod CPU usage**
- Scaling is automatic and gradual

Interview-Ready Explanation

> "HPA monitors pod-level metrics via Metrics Server and automatically adjusts replicas based on CPU utilization thresholds defined in the HPA resource."

Next Steps (Advanced Topics)

- Memory-based HPA
- Custom metrics HPA
- HPA with Prometheus Adapter
- VPA vs HPA
- HPA with KEDA