Kubernetes Container-Level CPU Autoscaling with Grafana & HPA

Objective

Demonstrate CPU-based autoscaling in Kubernetes at the **container level**, using:

- Horizontal Pod Autoscaler (HPA)
- Prometheus + Grafana stack for monitoring
- Custom deployment with sidecar container

Project Components

1. **Kubernetes Deployment**
2. **HPA Configuration**
3. **Grafana Dashboard for Monitoring**
4. **Sidecar Container for Demo Purposes**

✅ What This Script Will Do:

1. Install Docker, Kind, Kubectl, and Helm
2. Increase system limits (`ulimit`)
3. Create a Kind cluster with necessary configs
4. Deploy the Prometheus + Metrics server + HPA setup

Bash Script: `setup-k8s-cpu-autoscaling.sh`

```
#vim setup-k8s-cpu-autoscaling.sh
```

```
#!/bin/bash

set -e
```

```
echo "[1/8] Updating system and installing dependencies..."
sudo apt-get update && sudo apt-get install -y apt-transport-https
ca-certificates curl gnupg lsb-release docker.io jq

echo "[2/8] Starting Docker..."
sudo systemctl enable docker
sudo systemctl start docker

echo "[3/8] Installing kubectl..."
curl -LO
"https://storage.googleapis.com/kubernetes-release/release/v1.29.0/bin
/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/

echo "[4/8] Installing kind..."
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind

echo "[5/8] Increasing ulimit for open files..."
echo "* soft nofile 65535" | sudo tee -a /etc/security/limits.conf
echo "* hard nofile 65535" | sudo tee -a /etc/security/limits.conf
ulimit -n 65535

echo "[6/8] Creating Kind cluster with extra port and config..."
cat <<EOF | kind create cluster --name cpu-demo --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    extraPortMappings:
      - containerPort: 30000
        hostPort: 30000
EOF

echo "[7/8] Installing Helm..."
```

```
curl
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
bash

echo "[8/8] Deploying Prometheus + metrics server..."
kubectl create namespace monitoring
helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
helm repo update
helm install prometheus prometheus-community/kube-prometheus-stack
--namespace monitoring --set prometheus.service.type=NodePort --set
prometheus.service.nodePort=30000

echo "Setup complete!"
echo "Access Prometheus on: http://<EC2-PUBLIC-IP>:30000"
echo "Now deploy your CPU demo and HPA config."
```

```
chmod +x setup-k8s-cpu-autoscaling.sh
./setup-k8s-cpu-autoscaling.sh
```

Check Cluster:

```
kubectl get nodes
kubectl get all -n monitoring
```

Deploy CPU Demo App

Create a YAML file called cpu-demo-deployment.yaml:

Vim cpu-demo-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: cpu-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cpu-demo
  template:
    metadata:
      labels:
        app: cpu-demo
    spec:
      containers:
      - name: cpu-demo
        image: vish/stress
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
        args:
        - -cpus
        - "2"
      - name: sidecar-logger
        image: busybox
        command: ["sh", "-c", "while true; do echo sidecar running;
sleep 10; done"]
        resources:
          limits:
            cpu: 100m
            memory: 64Mi
          requests:
            cpu: 50m
            memory: 32Mi
---
apiVersion: v1
kind: Service
metadata:
  name: cpu-demo-service
```

```
spec:
  selector:
    app: cpu-demo
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: NodePort

#kubectl apply -f cpu-demo-deployment.yaml
#kubectl get pods
```

```
^Cubuntu@ip-172-31-45-181:~$ kubectl get pods
NAME                        READY   STATUS    RESTARTS        AGE
cpu-demo-7bf8d99447-22mjb   2/2     Running   4 (5m37s ago)   4d3h
cpu-demo-7bf8d99447-5vqr6   2/2     Running   4 (5m37s ago)   4d3h
```

```
#kubectl get svc
```

2. Create HPA for CPU Autoscaling

Now create an HPA YAML file called cpu-hpa.yaml:
Vim cpu-hpa.yaml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-demo-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cpu-demo
  minReplicas: 1
  maxReplicas: 2
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
```

```
        type: Utilization
        averageUtilization: 50
```

Apply this:

```
#kubectl apply -f cpu-hpa.yaml
#kubectl get hpa -w
```

3. Generate CPU Load to Trigger Autoscaling

🔁 Step-by-Step: Generate CPU Load with BusyBox

Run this temporary pod:

#kubectl run cpu-loader --image=busybox --restart=Never -it -- /bin/sh

Inside the pod, run:

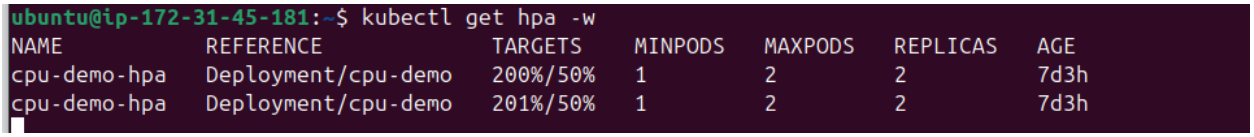
#while true; do :; done

This will create high CPU usage on the node, triggering the autoscaler.

⏱️ Then, Monitor the HPA:

In a new terminal, run:

#kubectl get hpa -w



You should soon see the number of replicas increase as the average CPU utilization rises.

🧼 Clean Up When Done:

#kubectl delete pod cpu-loader

—————————————————————

Easiest Fix – Install Metrics Server

This is the quickest and simplest way to get HPA working.

🔧 Step 1: Install Metrics Server

Run this command to install the official Metrics Server:

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

🔧 Step 2: Patch the Deployment to Accept Insecure TLS (if needed)

Some clusters (especially on AWS, kind or kubeadm) need extra args for Metrics Server:

```
kubectl patch deployment metrics-server -n kube-system \
  --type=json \

-p='[{"op":"add","path":"/spec/template/spec/containers/0/args/-","value":"--kubelet-insecure-tls"}]'
```

Wait ~30 seconds and test:

```
kubectl top pods
```

If this returns CPU/Memory info – the HPA will now work.

✅ What Happened

- Your CPU load hit **~230%**, well over your target threshold of 50%.
- The HPA **scaled up the `cpu-demo` deployment from 1 to 5 replicas** (your max limit).
- All `cpu-demo` pods are now consuming ~450m CPU individually – validating real load.
- The `cpu-loader` pod is also doing its job, keeping CPU pressure active.

Access Grafana Dashboard
#kubectl port-forward -n monitoring svc/prometheus-grafana 32000:80 --address 0.0.0.0

Access Promotheus Dashboard
#kubectl port-forward svc/prometheus-kube-prometheus-prometheus -n monitoring 9090:9090 --address 0.0.0.0

🔐 Login credentials:
#kubectl get secret -n monitoring prometheus-grafana -o jsonpath="{.data.admin-password}" | base64 --decode

user:admin
passwd:prom-operator

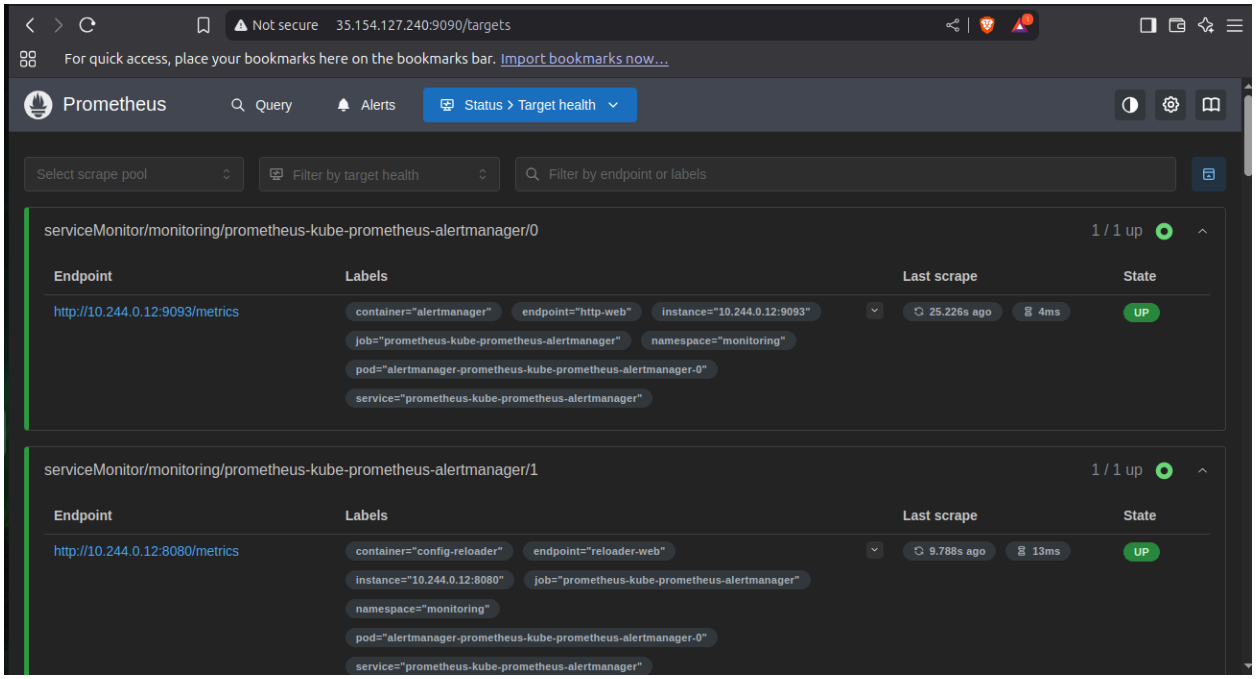
Monitoring with Grafana

- **Dashboard**: Kubernetes > Compute Resources > Pod
- **Data Source**: Prometheus
- **Observed Metrics**:
    - CPU usage and throttling visible **per container** (cpu-demo, sidecar-logger)
    - Requests & limits shown with dashed lines

Key Insight:

- These metrics come from **kubelet + cAdvisor** and represent **system-level resource metrics**.

Prometheus Scraping the data



GRAFANA pod level Monitoring

GRAFANA CPU level Monitoring

Miscellaneous:

To visualize CPU and auto scaling metrics, you can now do one of the following:

🔹 Option 1: Import ready-made dashboards
Go to Grafana → Dashboards → Import

Use an official Prometheus dashboard ID like:

ID 6417 (Kubernetes cluster monitoring with Prometheus)

ID 8588 (Node Exporter Full for CPU, memory, etc.)

Click "Load", select your Prometheus data source, and import.

#kubectl get svc -n monitoring

#kubectl get pods -n monitoring -l app.kubernetes.io/name=prometheus

To delete the pod forcefully

#kubectl delete pod prometheus-prometheus-kube-prometheus-prometheus-0 -n monitoring --grace-period=0 --force

#kubectl rollout restart deployment -n monitoring prometheus-grafana

To reach Inside the pod
#kubectl exec -n monitoring -it prometheus-prometheus-kube-prometheus-prometheus-0 -c prometheus -- /bin/sh
Inside pod:
rm -rf /prometheus/wal
Exit

Troubleshooting & Lessons Learned

Issue	Solution
CPU metrics not appearing per container	Add proper labels and annotations; ensure Prometheus scraping is enabled
Sidecar container not visible in Grafana	Metrics appear only if container consumes resources or has limits/requests set
CPU usage flatlined at pod level	Switch to per-container view to understand true container behavior
High throttling observed	Adjust CPU limits to reduce throttling on stressed containers
Application metrics missing	Expose custom metrics endpoint and update Prometheus scrape config

Conclusion

We successfully demonstrated container-level autoscaling using HPA and visualized detailed CPU usage and throttling metrics using Grafana. We also differentiated between system-level metrics (cAdvisor/kubelet) and application-level metrics (custom Prometheus exporters).