

**Lab Manual- Cluster autoscaling Horizontal POD Scaling (HPA)-(Lab1)**

**Prepared for**:

**Date:** 18th Dec 2023

**Prepared by:**

Document Name: Lab Manual **Document Number** AZLabn916

**Contributor:**

Contents

[1. Objective 2](#_Toc159325091)

[2. AKS Cluster autoscaler with custom Metric 3](#_Toc159325092)

[3. Create AKS Cluster with autoscaler and Connect 4](#_Toc159325093)

[4. Deploy Ngnix Application 5](#_Toc159325094)

[5. Enable HPA the Ngnix Deployment 5](#_Toc159325095)

[6. Scale Ngnix Deployment by Increasing CPU 6](#_Toc159325096)

# Objective

The cluster autoscaler component watches for pods in your cluster that can't be scheduled because of resource constraints. When the cluster autoscaler detects issues, it scales up the number of nodes in the node pool to meet the application demand. It also regularly checks nodes for a lack of running pods and scales down the number of nodes as needed.

# AKS Cluster autoscaler with custom Metric

To configure the AKS (Azure Kubernetes Service) cluster autoscaler to scale based on custom metrics specific to your application or workload, you can leverage custom metrics and horizontal pod autoscaler (HPA) configurations in Kubernetes. Here's how you can achieve this:

1. **Set Up Custom Metrics**:

First, you need to expose custom metrics from your application or monitoring system. This could involve instrumenting your application code to emit custom metrics or using monitoring solutions like Prometheus or Azure Monitor to collect custom metrics.

1. **Configure Horizontal Pod Autoscaler (HPA)**:

Once you have custom metrics available, you can configure an HPA in Kubernetes to scale your application based on these metrics. You define a custom HPA resource with your custom metrics as the scaling target.

For example, let's say you have a custom metric named **custom\_metric**, and you want to scale your deployment based on this metric. You can create an HPA like this:

apiVersion: autoscaling/v2beta2

kind: HorizontalPodAutoscaler

metadata:

  name: myapp-hpa

spec:

  scaleTargetRef:

    apiVersion: apps/v1

    kind: Deployment

    name: myapp-deployment

  minReplicas: 1

  maxReplicas: 10

  metrics:

  - type: External

    external:

      metricName: custom\_metric

      targetAverageValue: 1000m

1. In this example, the HPA scales the deployment named **myapp-deployment** based on the custom metric **custom\_metric**. The **targetAverageValue** specifies the desired value for the custom metric at which the deployment should be scaled.
2. **Enable Custom Metrics in AKS Cluster Autoscaler**:

By default, the AKS cluster autoscaler supports scaling based on CPU and memory metrics. To enable scaling based on custom metrics, you need to deploy and configure a custom metrics adapter that translates custom metrics into a format understandable by the HPA and the cluster autoscaler.

You can use the Kubernetes Metrics Server with a custom metrics adapter, such as the Prometheus Adapter or Azure Monitor Metrics Adapter, to expose custom metrics to the HPA and the cluster autoscaler.

1. **Deploy Custom Metrics Adapter**:

Deploy the custom metrics adapter in your AKS cluster. You can find instructions for deploying specific custom metrics adapters in the respective documentation.

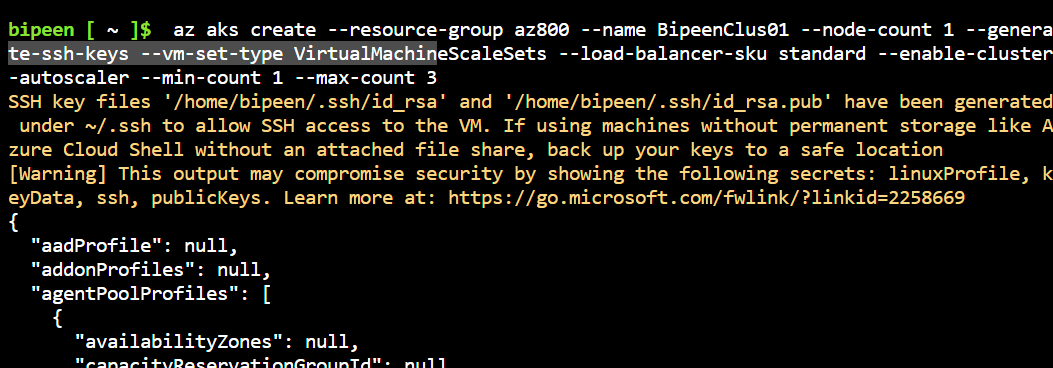
1. **Update AKS Node Pool Configuration**:

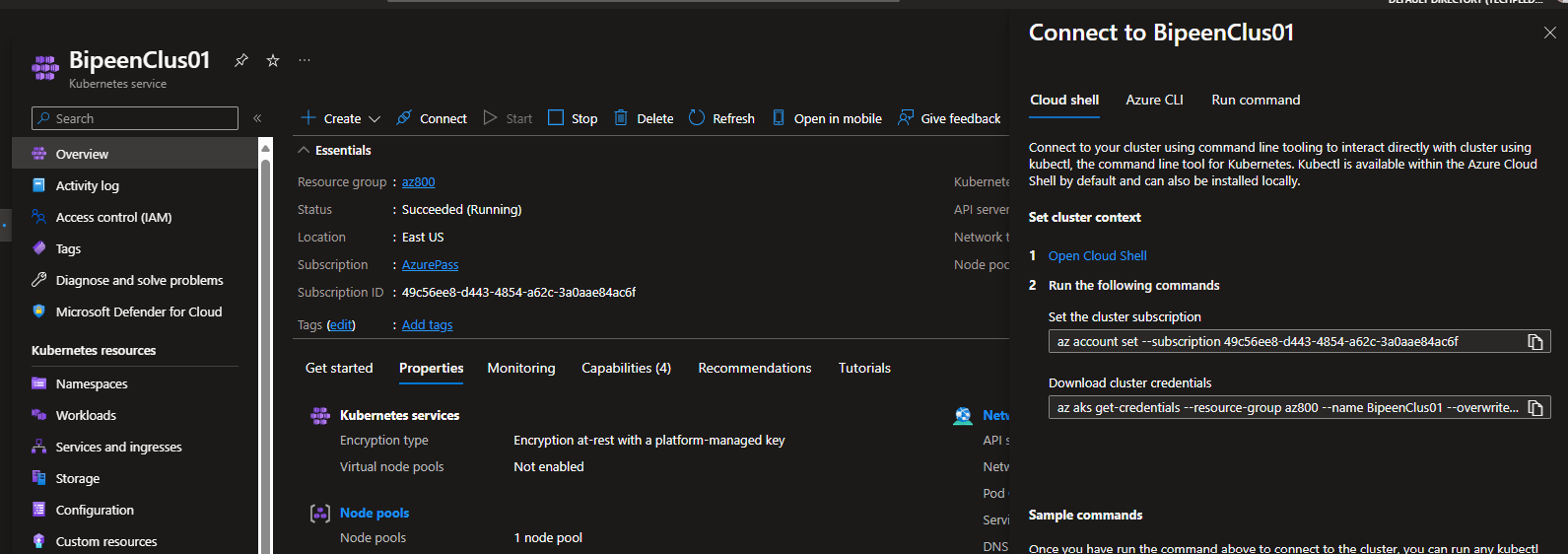
Ensure that the AKS node pool associated with your application has autoscaling enabled. You can configure the minimum and maximum number of nodes in the node pool to scale dynamically based on the workload demands.

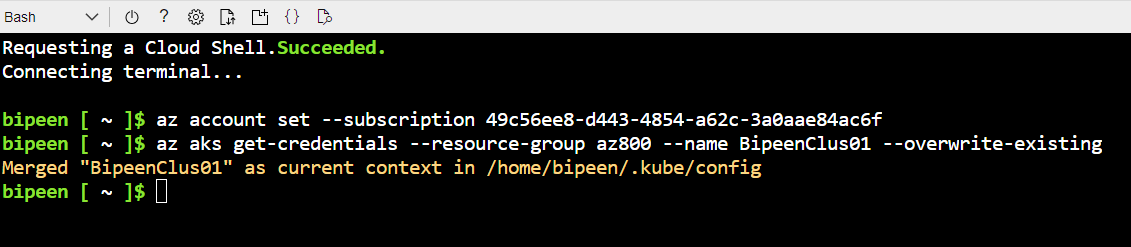
By following these steps, you can configure the AKS cluster autoscaler to scale based on custom metrics specific to your application or workload. Adjust the HPA configuration and custom metrics adapter based on your requirements and monitoring setup.

# Create AKS Cluster with autoscaler and Connect

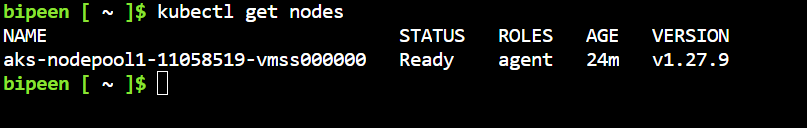
az aks create --resource-group az800 --name BipeenClus01 --node-count 1 --generate-ssh-keys --vm-set-type VirtualMachineScaleSets --load-balancer-sku standard --enable-cluster-autoscaler --min-count 1 --max-count 3







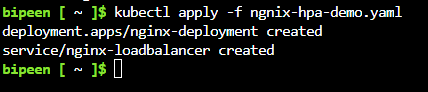
kubectl get nodes



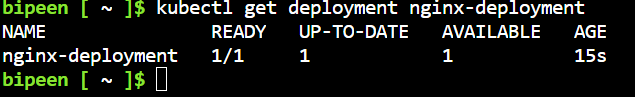
# Deploy Ngnix Application

Clone the Code from : <https://github.com/bipeensinha/AKS-Scaleing>

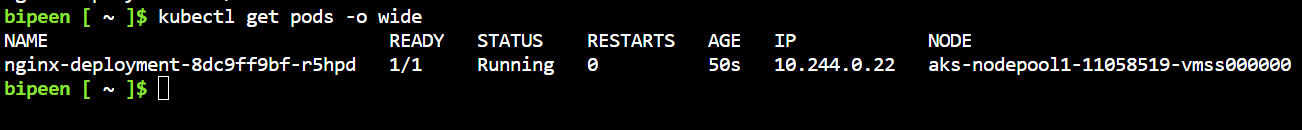
kubectl apply -f ngnix-hpa-demo.yaml



kubectl get deployment nginx-deployment

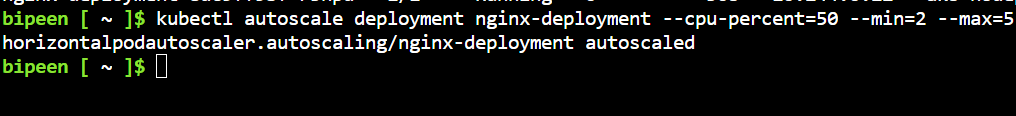


kubectl get pods -o wide

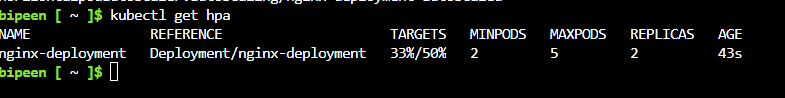


# Enable HPA the Ngnix Deployment

kubectl autoscale deployment nginx-deployment --cpu-percent=50 --min=2 --max=5

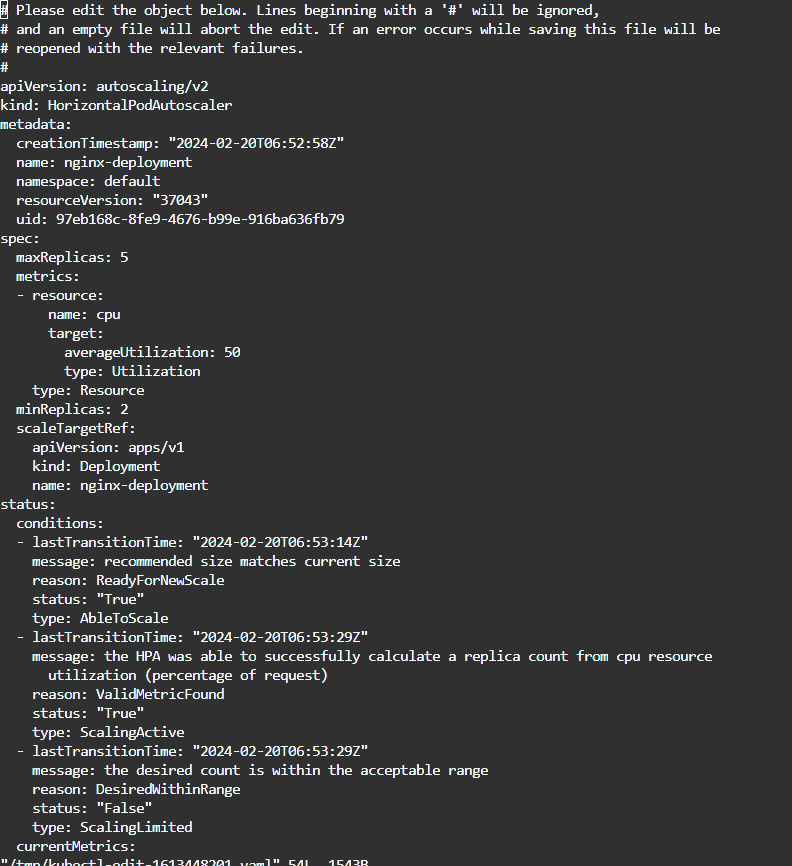


kubectl get hpa



kubectl edit hpa





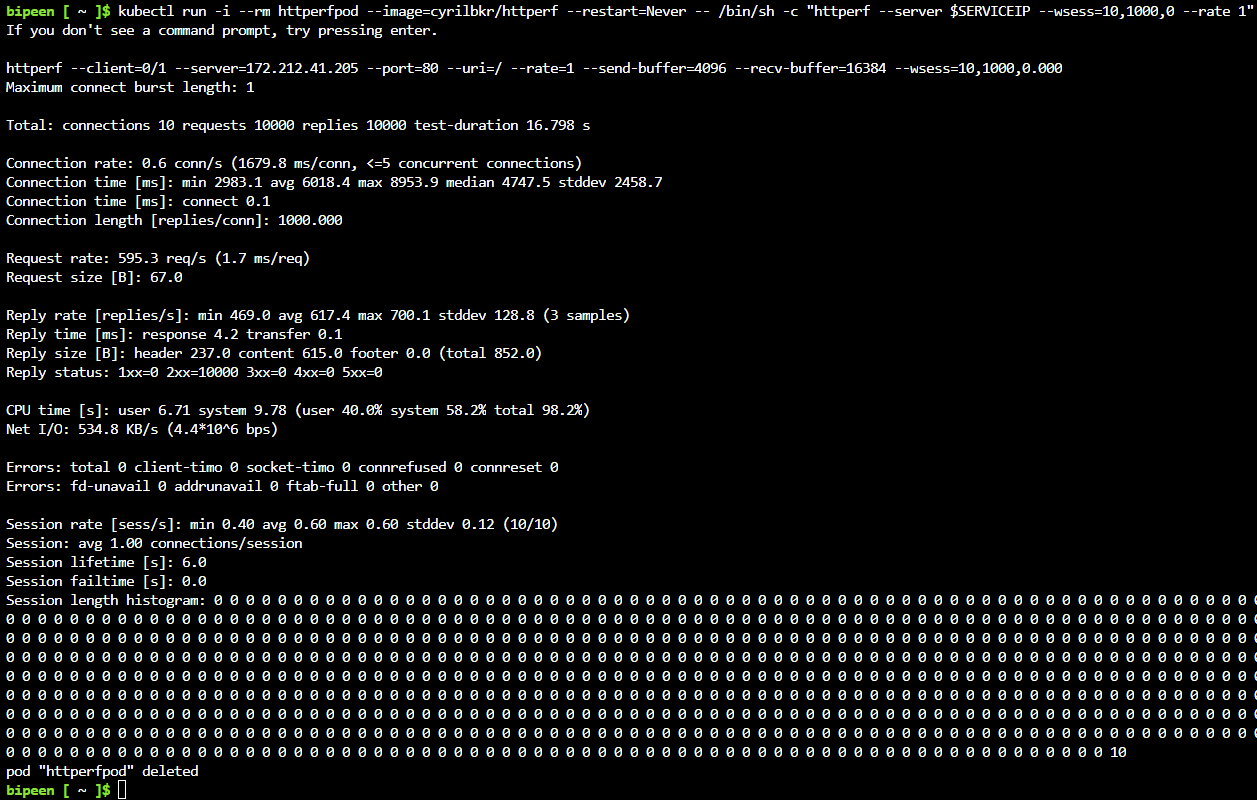
Exit without save

# Scale-up Ngnix Deployment by Increasing CPU

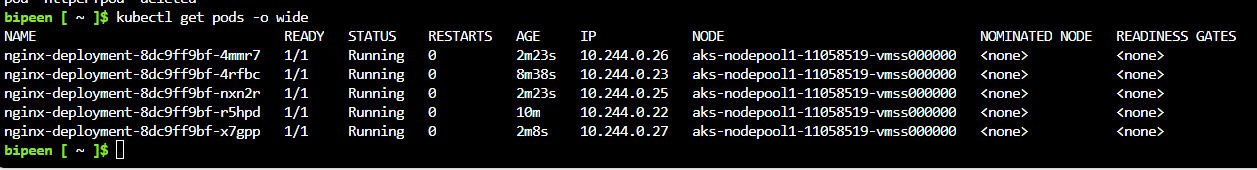
SERVICEIP=$(kubectl get service nginx-loadbalancer -o jsonpath='{.status.loadBalancer.ingress[0].ip}')



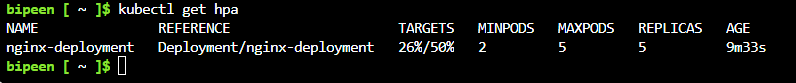
kubectl run -i --rm httperfpod --image=cyrilbkr/httperf --restart=Never -- /bin/sh -c "httperf --server $SERVICEIP --wsess=10,1000,0 --rate 1"



kubectl get pods -o wide



kubectl get hpa



# Scale-Down Ngnix Deployment Automatically

After 10 Min you can see the start scale down automatically

