**Azure Kubernetes Service (AKS)**

Azure Kubernetes Service (AKS) is a managed Kubernetes service offered by Microsoft Azure.

**Kubernetes:**

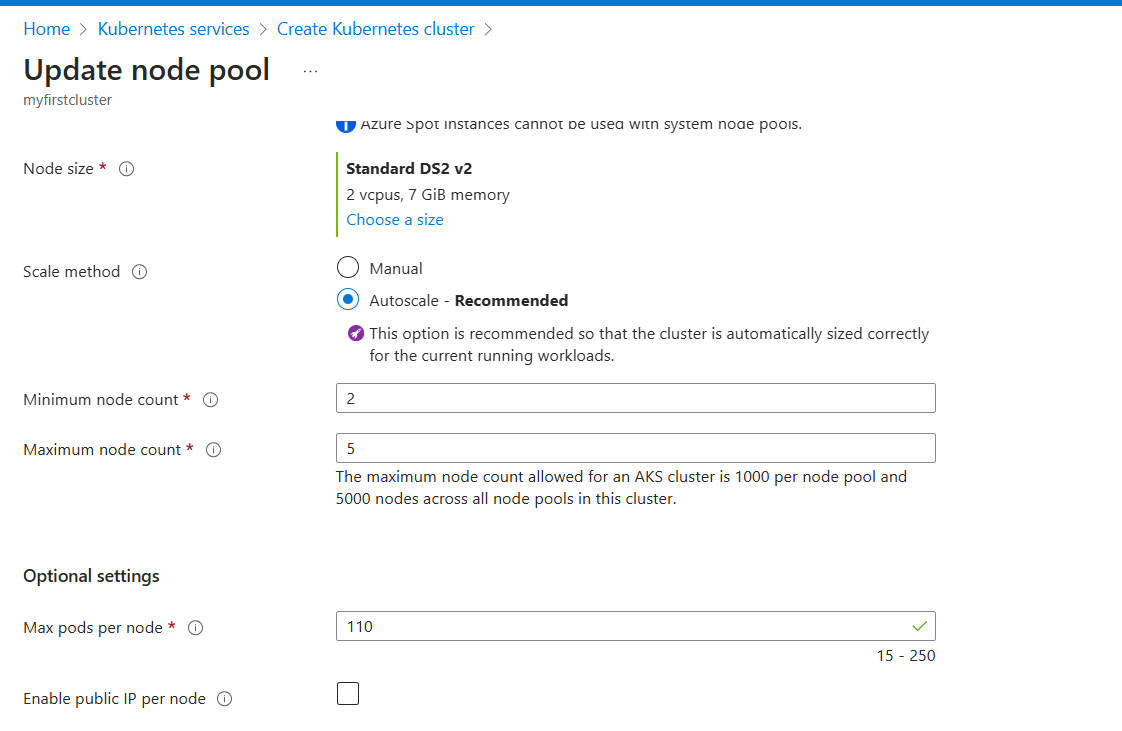
* Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. Containers allow applications to run consistently across different environments.

In order to work with the AKS we have to follow the below steps:

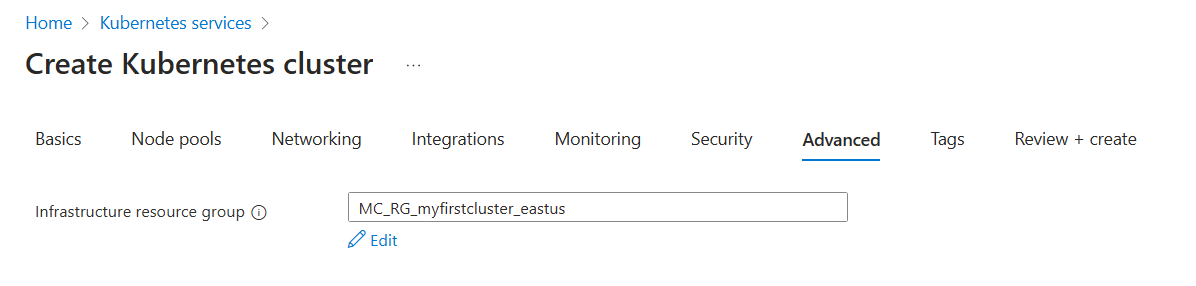
1. Create Ubuntu VM. (Acts as a client/user).
2. Install the Azure CLI.
3. Install the Kubctl
4. Login to the azure portal using Azure CLI
5. Connect to the AKS.

Let’s work with AKS practically.

**Step1:** Create the AKS in azure portal.



**Note1:** Default number of pods can be created on a worker node are 110.

Note2: The maximum node count allowed for an AKS cluster is 1000 per node pool and 5000 nodes across all node pools in this cluster.

**Note3:** While creating the cluster by default a resource group is created as shown in above figure.

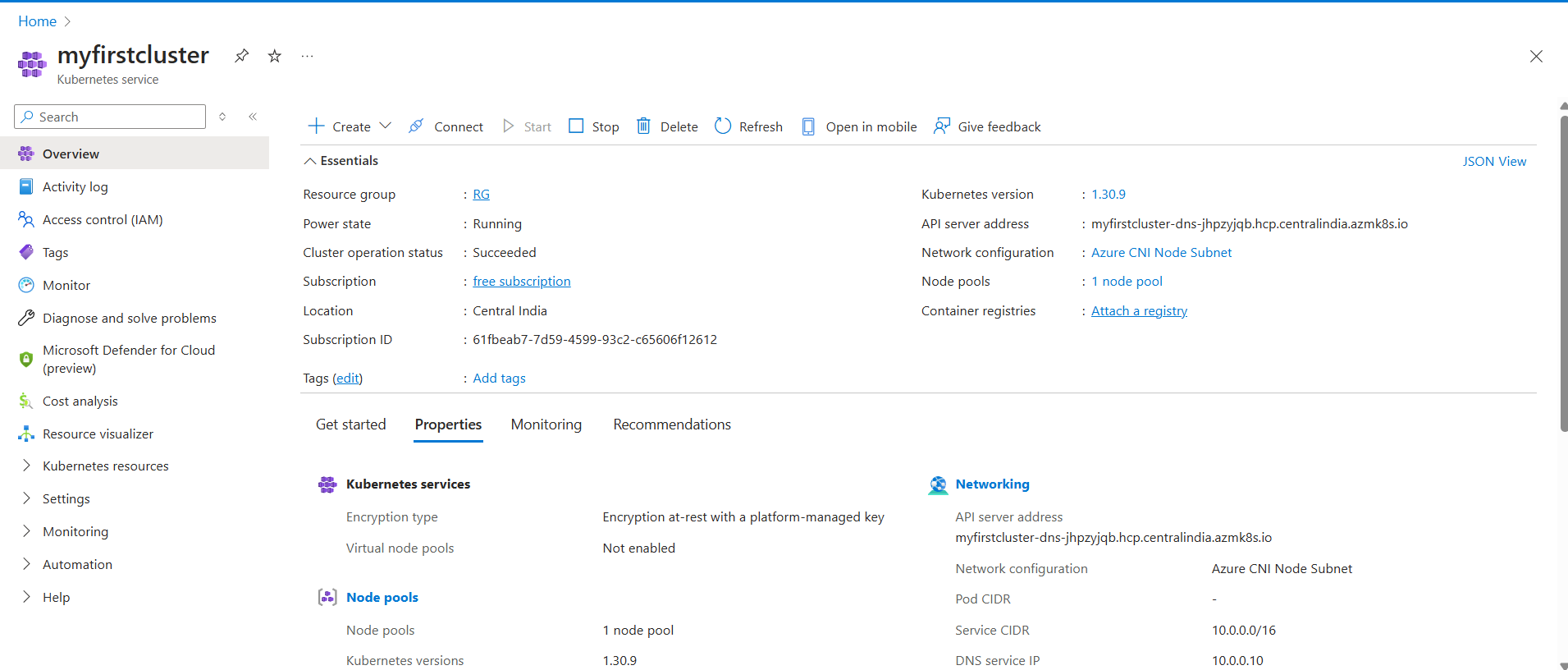


Fig: AKS is created successfully.

**Note4:** While creating cluster a Load balancer with frontend IP and backend pool with nodes are created as shown below figure.

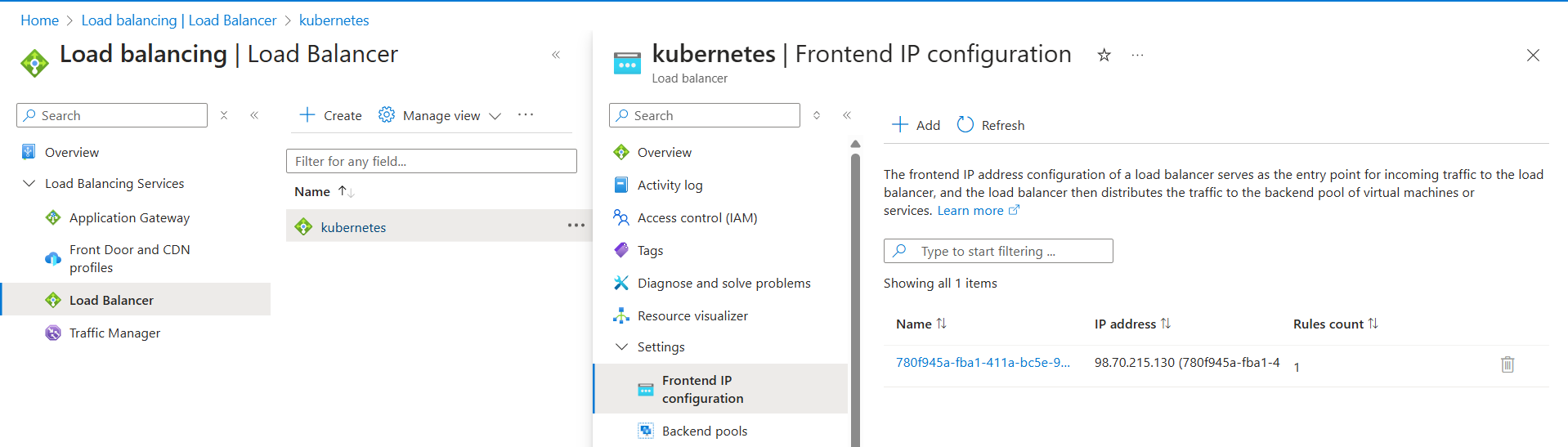


Fig: Default Load balancer.

**Note:** This default created Load balancer is responsible for the administrative level tasks/operations like create pod, delete pod and so on… requests go through this load balancer.

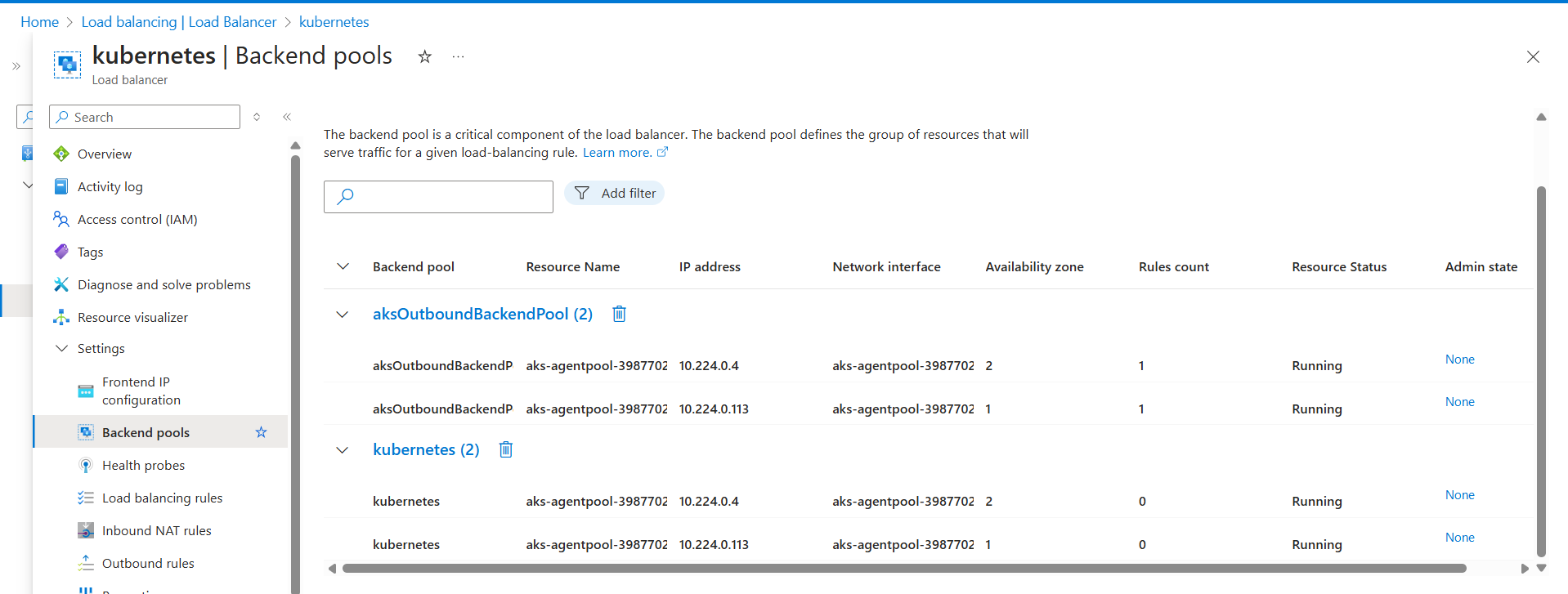


Fig: Backend pool of Load balancer with two nodes.

**Step2:** Create the Ubuntu virtual machine.

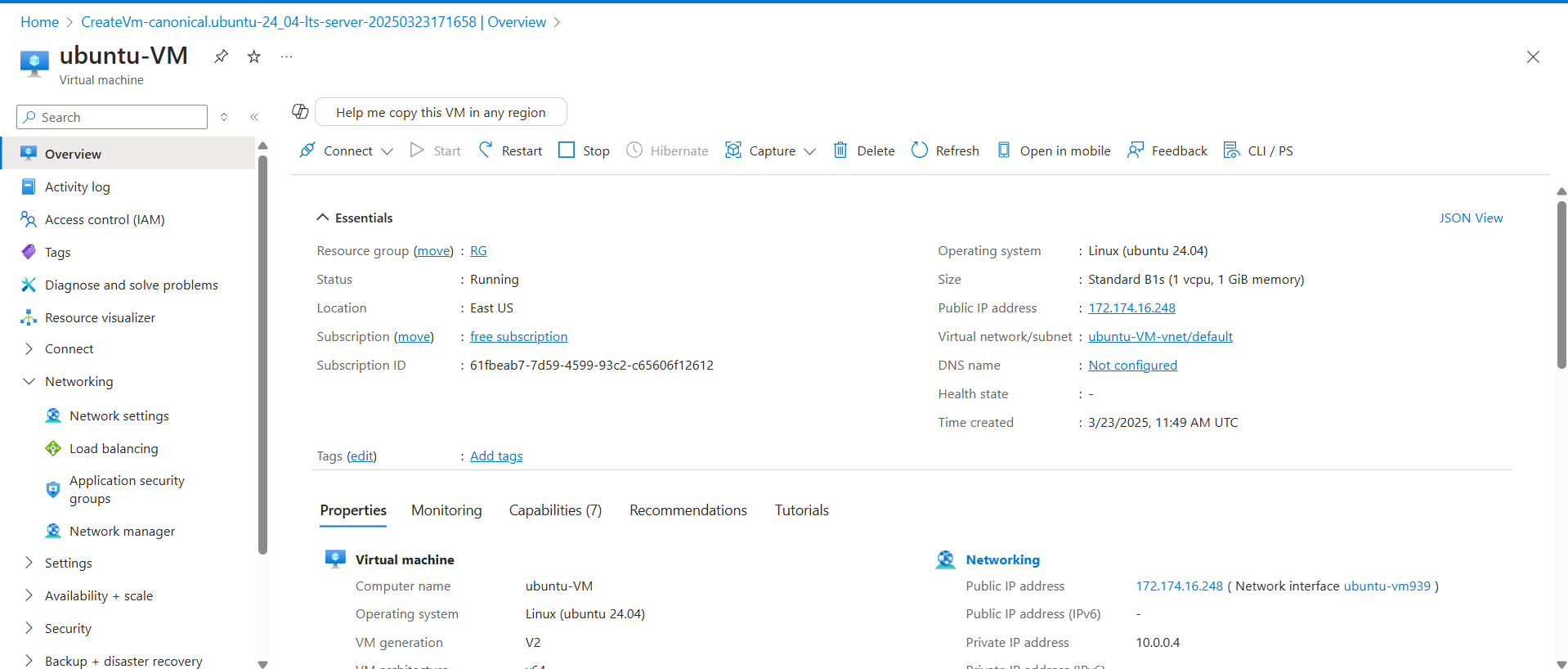
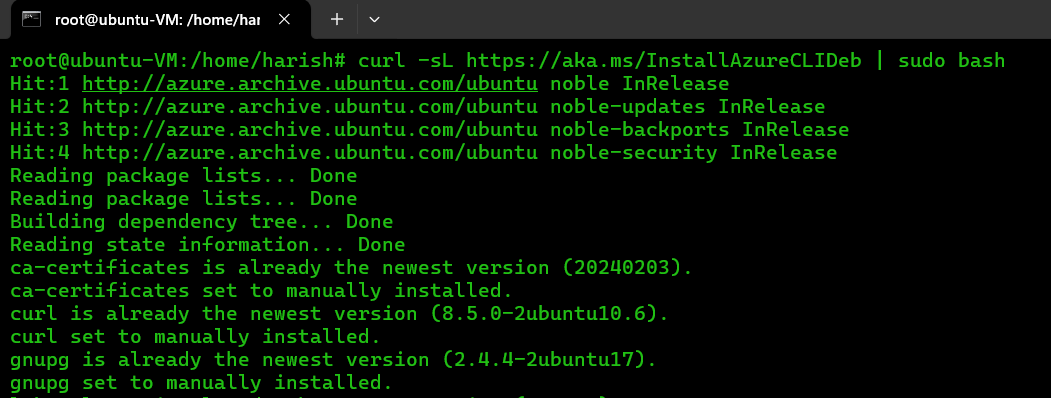


Fig: Ubuntu virtual machine.

Step3: Login to the Ubuntu VM and install Azure CLI and Kubectl in it.

**URL to install Azure CLI:** curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash



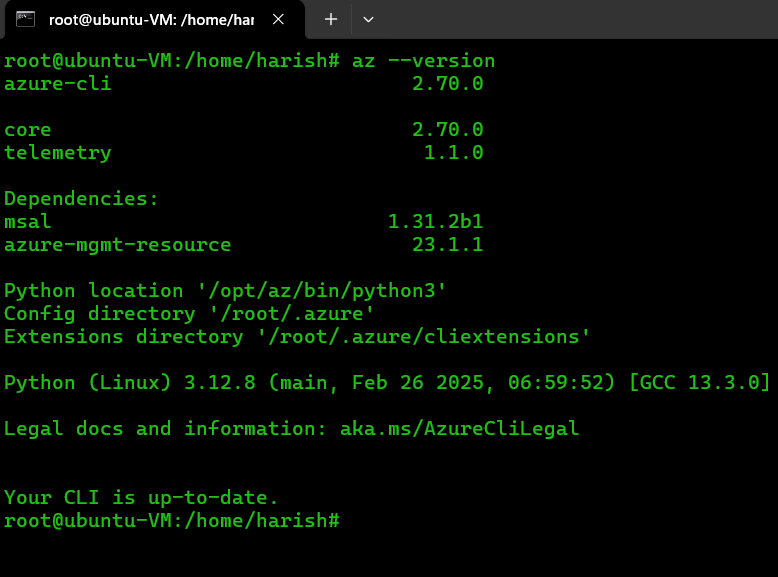


Fig: Azure CLI is installed successfully.

Installing of Kubectl in Ubuntu VM:

* 1. Download the latest release with the command:

**Command:** curl -LO [https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)

* 1. Install Kubectl:

**Command:** sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

#### Note: If you do not have root access on the target system, you can still install kubectl to the “~/.local/bin” directory:

chmod +x kubectl

mkdir -p ~/.local/bin

mv ./kubectl ~/.local/bin/kubectl

# and then append (or prepend) ~/.local/bin to $PATH

* 1. Test to ensure the version you installed is up-to-date:

**Command:** kubectl version --client

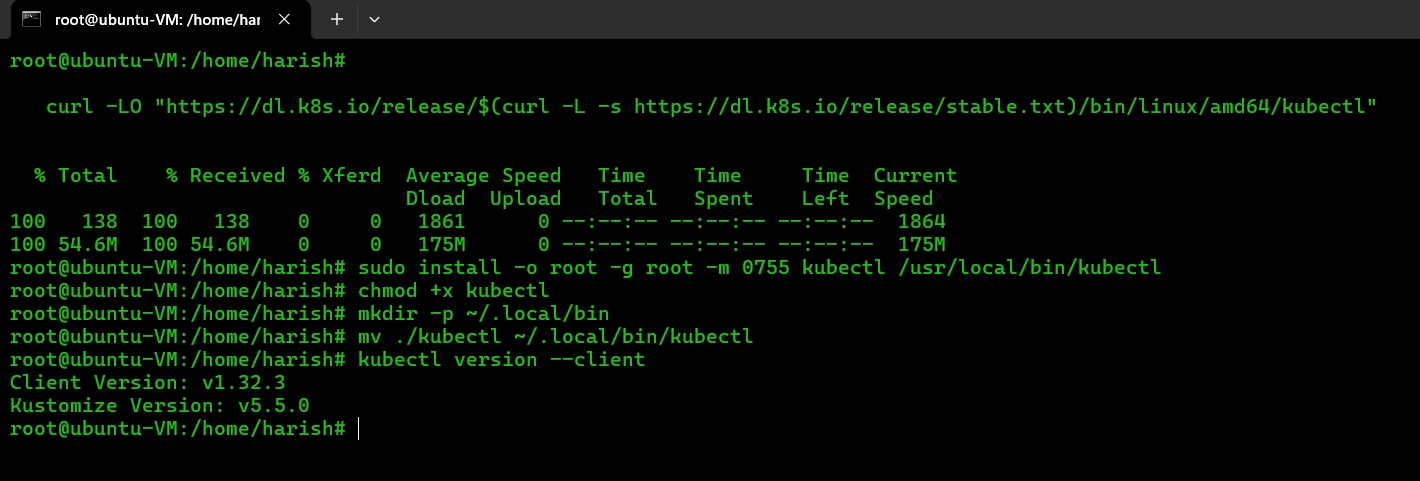


Fig: Kubectl is installed successfully.

**Step4:** Connect to the AKS (azure kubernetes service).

In order to connect to the azure Kubernetes services first we have to login to the azure portal.

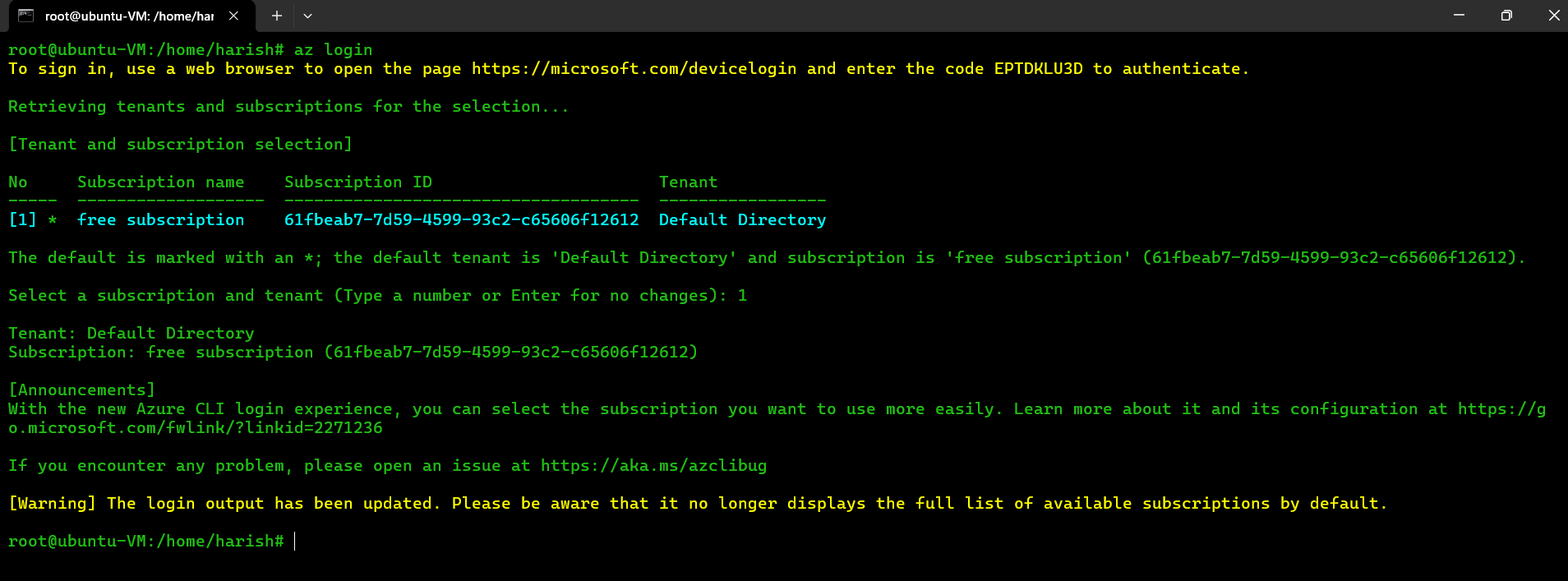
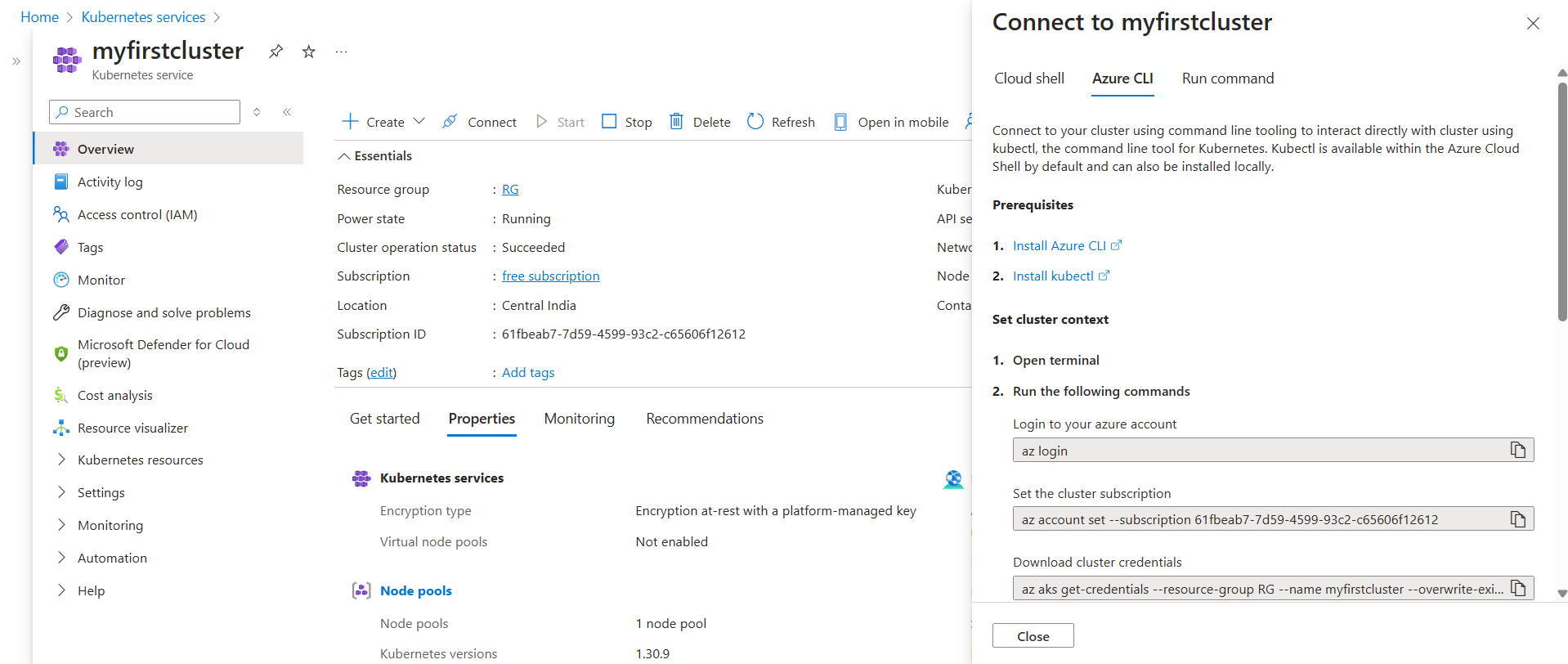
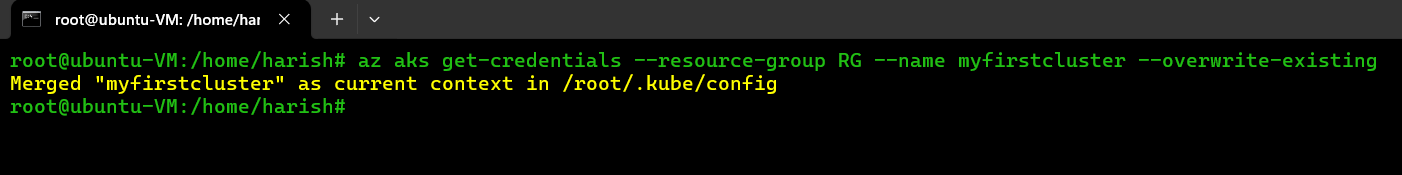


Fig: Login succeeded.



Using cluster credential connect to the azure Kubernetes services.

Cluster credentials are downloaded on at **/root.kube/config** directory.

By performing the Kubectl commands let’s check whether cluster is connected or not.

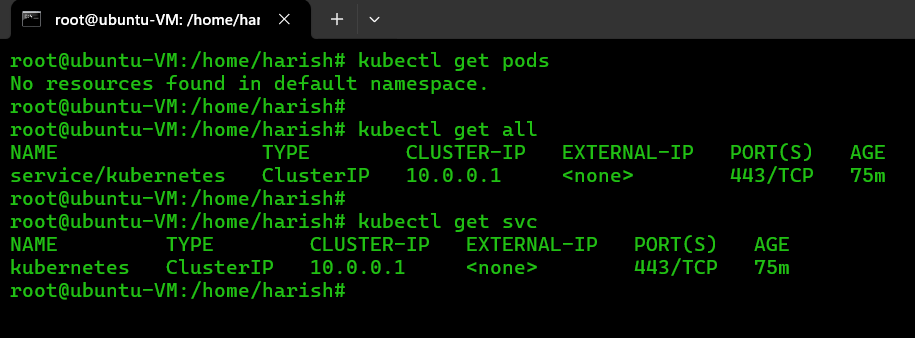


Fig: Successfully connected to the Azure Kubernetes cluster.

**Step5:** Create pod using a “.yaml” (deployment.yaml) file.

* A .yaml (YAML) files are human readable files with easy syntax.
* It is designed to represent data in a way that's easy for both humans to understand and for computers to process.
* YAML is widely used for configuration files in various software applications and systems such as Kubernetes.
* YAML files are the primary way you define and manage your applications and infrastructure. They are used to describe the desired state of your Kubernetes objects.
* YAML files allow you to treat your Kubernetes infrastructure as code.
* AML files are essential for Kubernetes because they provide a standardized, declarative, and human-readable way to define and manage your cluster's resources.

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-deployment

spec:

replicas: 2 # Number of pods to run

selector:

matchLabels:

app: my-app # Select pods with this label

template: # Pod template

metadata:

labels:

app: my-app # Labels for the pods

spec: # Pod specification

containers:

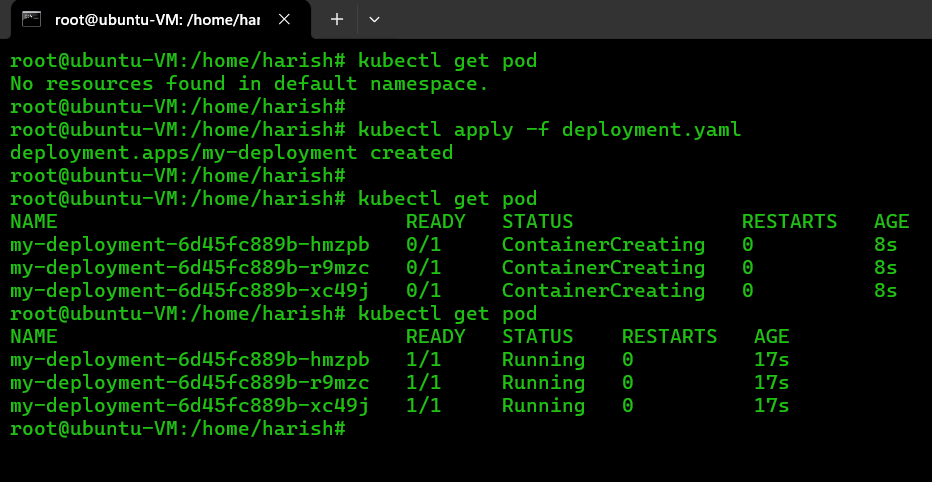
- name: my-container

image: nginx:latest # Container image

ports:

- containerPort: 80

Fig: deployment.yaml file.



In YAML file we configure such a way that to create three pods (replicas: 3) and pull the Nginx image from the Docker Hub as a results three pods are created successfully as show in above figure.

In order to check whether the Nginx image is pulled from Docker hub to the pod or not, first we have to create an external service (like LoadBalancer, Nodeport and ingress) to access Nginx from outside of the cluster.

By default a “**clusterIp service”** is created in our cluster which is used only for internal communication (between the pods), for external communication we have to create a service called Load balancer service.

To create a Load balancer service we have to write a yaml file for it.

apiVersion: v1

kind: Service

metadata:

name: my-loadbalancer-service

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 80

protocol: TCP

selector:

app: my-app # Replace with your app's label

Fig: service.yaml file.

Now apply this above service.yaml file in cluster to create the Load balancer service.

**Command:** kubectl apply –f service.yaml

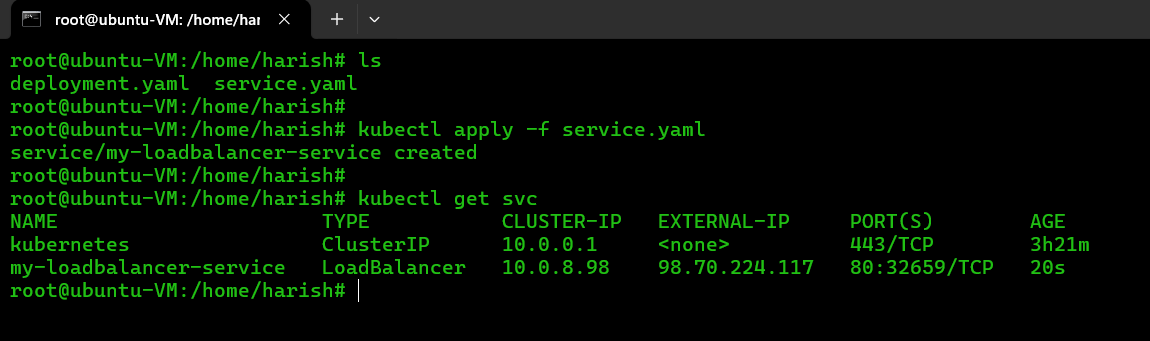
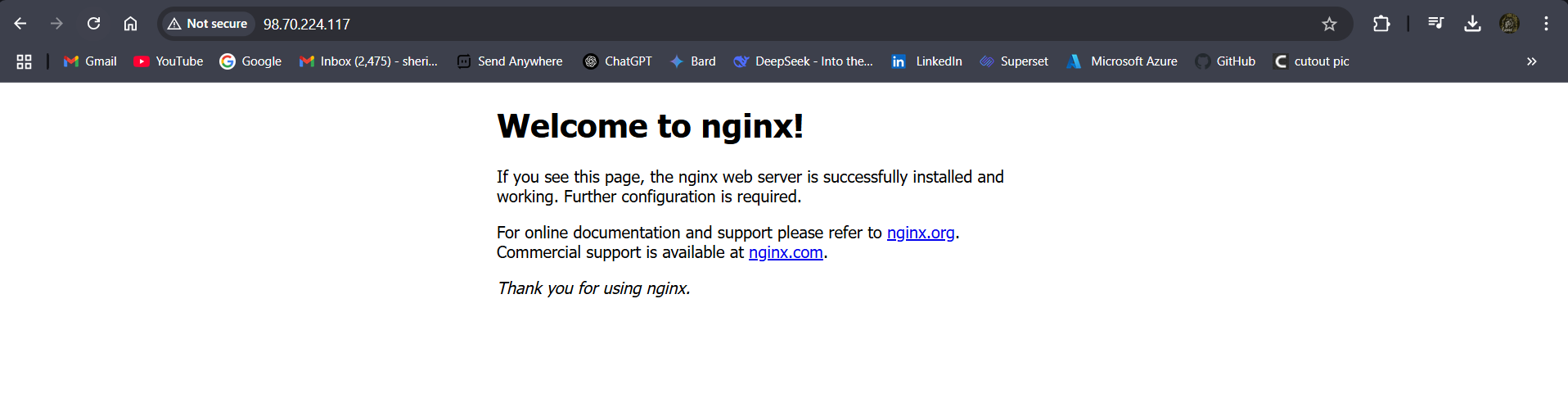


Fig: Load balancer service is created successfully.

In the above figure by using EXTERNAL-IP (98.70.224.117) of LoadBalancer we can access the Nginx Application from any browser.



**Note:** When we create a Load balancer service, an extra load balancer cannot created but it will created another public-IP (frontendIP).

