Steps to create a Kubernetes solution in Microsoft Azure

Step	Description
1	Prepare tools and Azure account
2	Log in to Azure
3	Create resource group
4	Create AKS cluster
5	Get credentials
6	Verify cluster status
7	Deploy app
8	Monitor and scale

Designed and deployed a Kubernetes solution on Microsoft Azure using AKS (Azure Kubernetes Service). Logged into the Azure Portal and utilized Azure Cloud Shell (Bash) to create a resource group in the East US region. Provisioned an AKS cluster named *myAKSCluster* with 3 nodes and auto-generated SSH keys. Connected to the cluster using kubectl, deployed a containerized NGINX application exposed on port 80 with a LoadBalancer service type. Retrieved the external IP for public access and verified deployment health. Monitored cluster metrics and logs via Azure Insights to ensure performance and availability.

Kubernetes Solution Deployment on Microsoft Azure

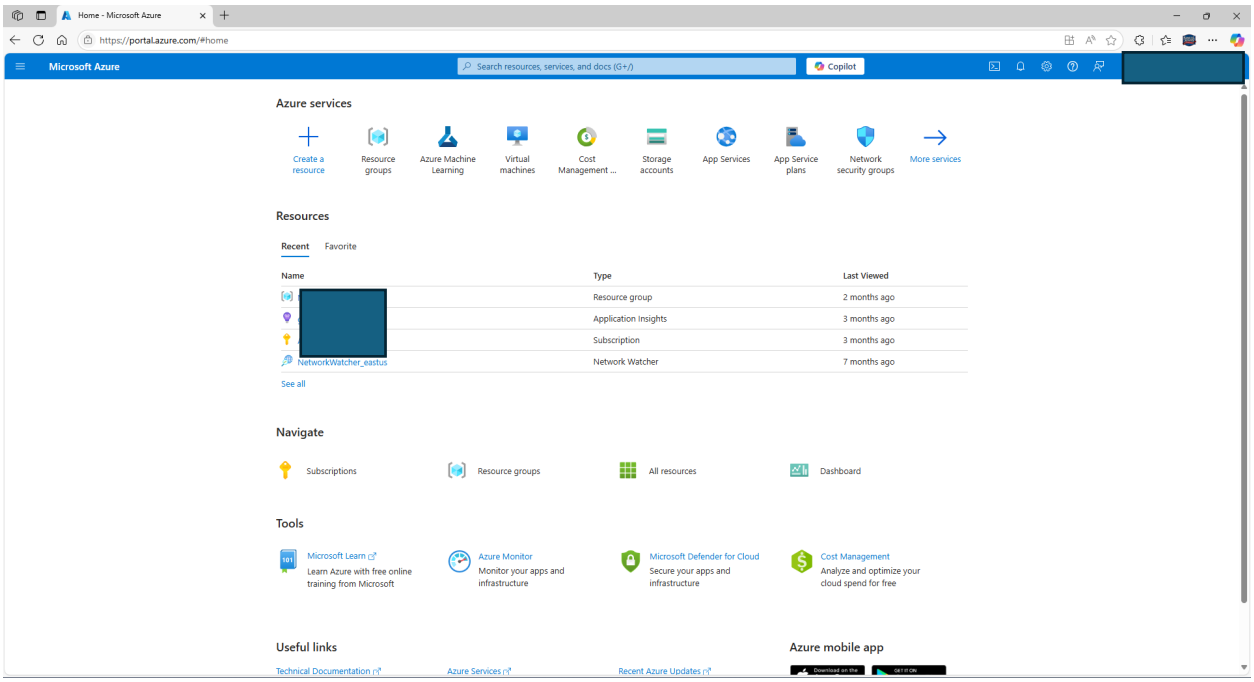
- **Problem Statement:** Deployed a scalable, containerized web application using Azure-managed Kubernetes infrastructure.
- **Challenges:** Set up secure access, managed cluster scaling, enabled public exposure of services, and configured monitoring for performance visibility.
- **Tools/Technologies:** Azure Portal, AKS, Azure Cloud Shell (Bash), kubectl, NGINX, Azure Monitor/Insights.
- **Outcomes:** Successfully provisioned a 3-node AKS cluster (*myAKSCluster*) with SSH access, deployed a LoadBalancer-exposed NGINX application, retrieved external IP for access, and validated performance through Azure Insights.

1. Log in to Azure

a. Username: joshphillis@hotmail.com

b. Password:

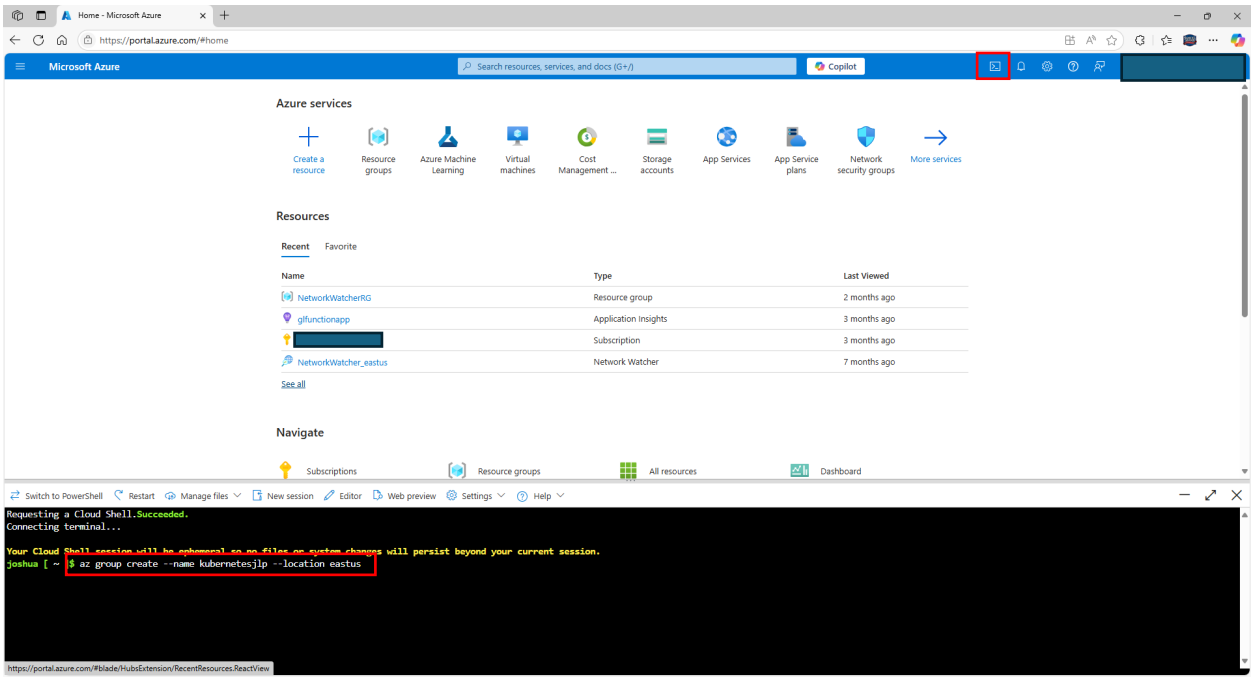
c. Microsoft Authenticator: 6 digit code

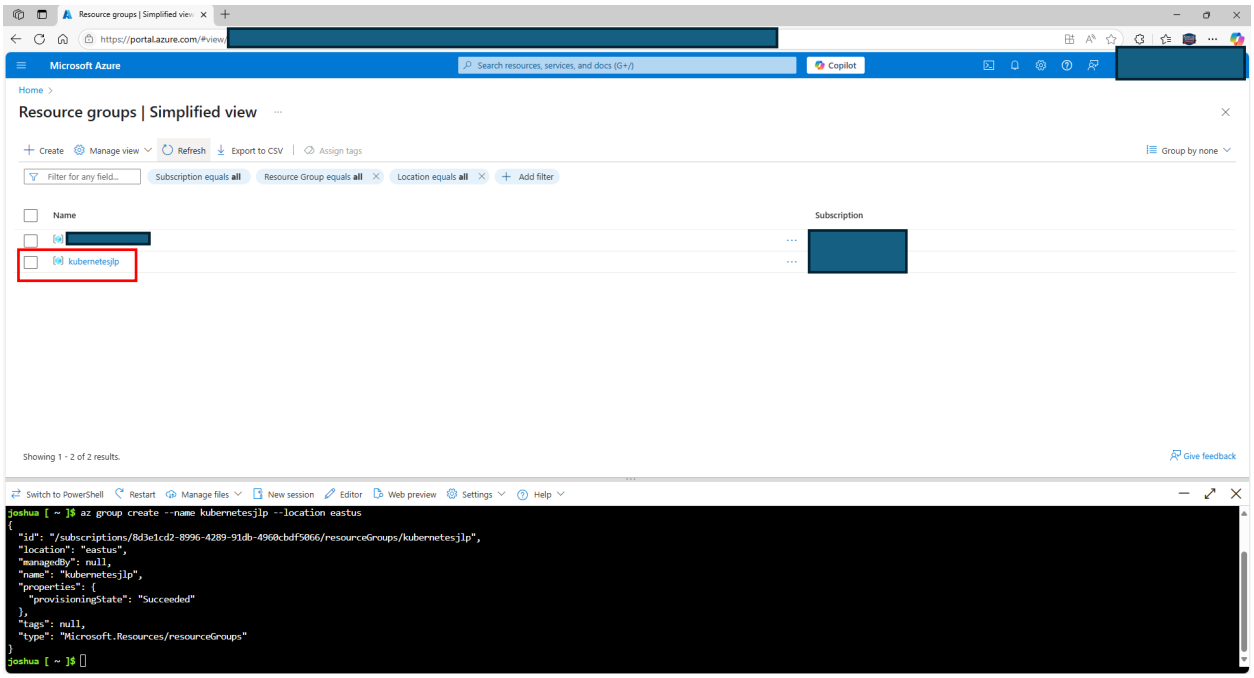


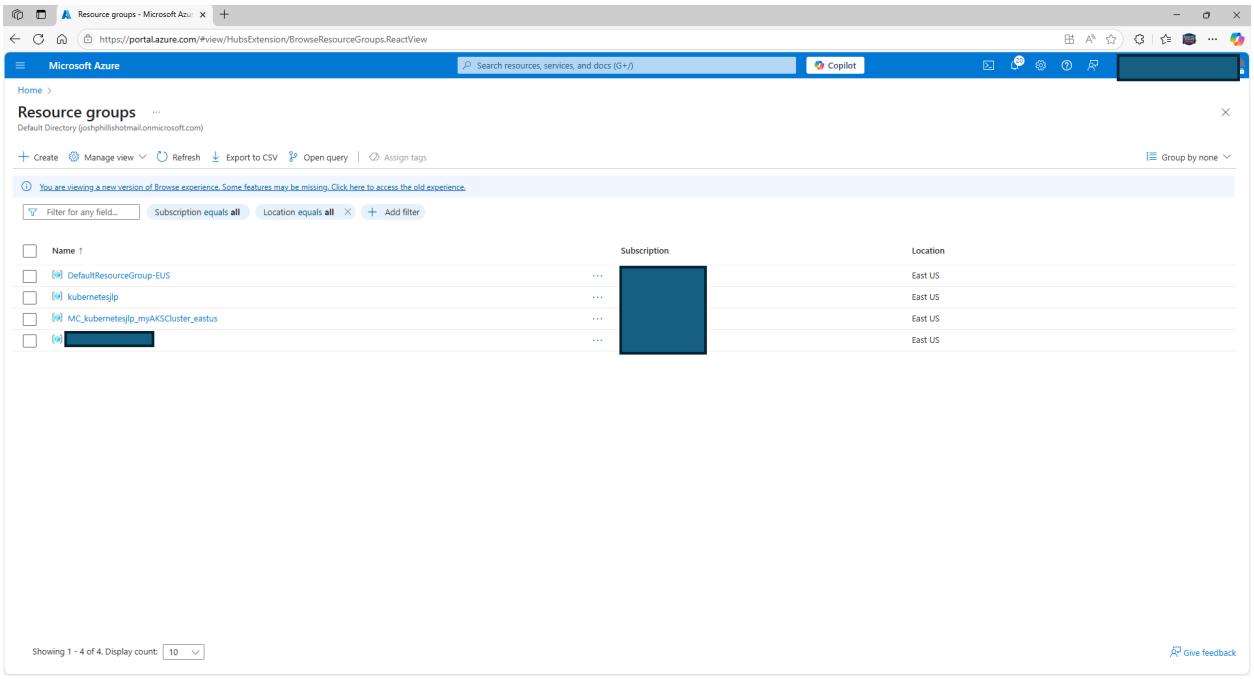
2. Create a Resource Group (Bash)

Select **Cloud Shell** icon

Enter: az group create --name kubernetesjlp --location eastus






Resource groups - Microsoft Azu
https://portal.azure.com/#view/HubsExtension/BrowseResourceGroups.ReactView
Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
Home
Resource groups
Default Directory (joshphillishotmail.onmicrosoft.com)
Create
Manage view
Refresh
Export to CSV
Open query
Assign tags
Group by none
You are viewing a new version of Browse experience. Some features may be missing. Click here to access the old experience.
Filter for any field...
Subscription equals all
Location equals all
Add filter
Name
Subscription
Location
DefaultResourceGroup-EUS
East US
kubernetesjlp
East US
MC_kubernetesjlp_myAKSCluster_eastus
East US
East US
Showing 1 - 4 of 4. Display count:
10
Give feedback

3. Create the AKS Cluster (Bash) - *At first, the script did not work. I had to adjust the naming.*

Enter: az aks create \

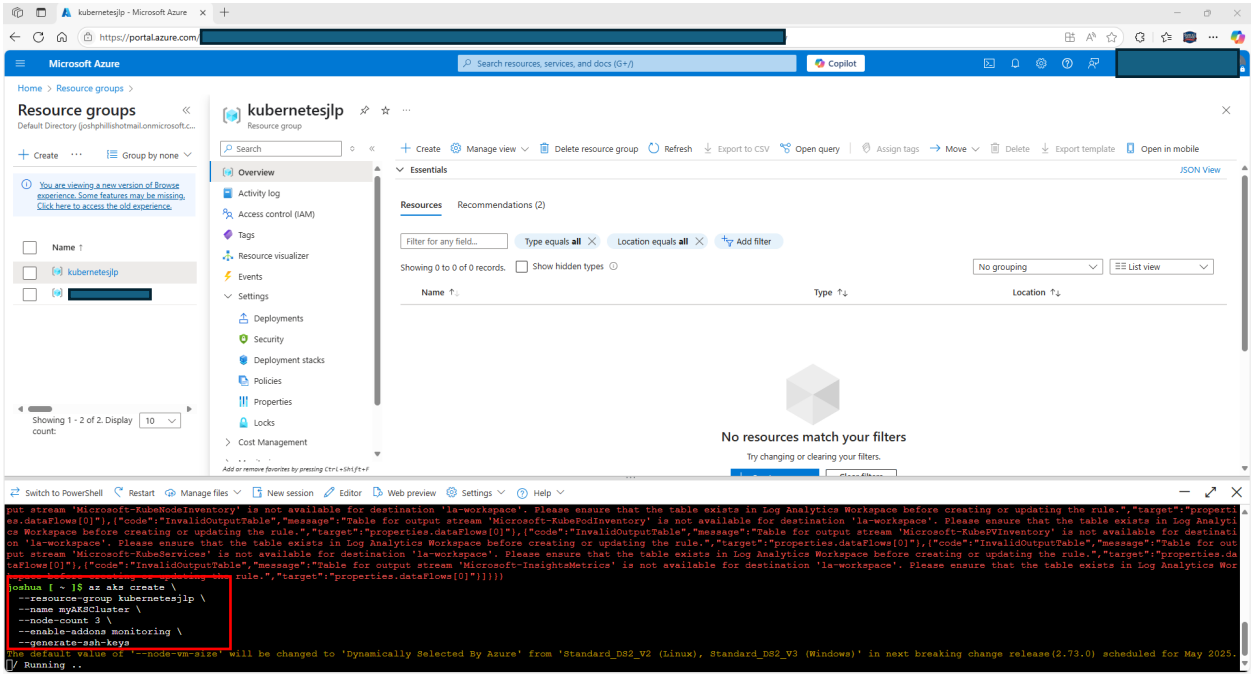
--resource-group myResourceGroup \

--name myAKSCluster \

--node-count 3 \

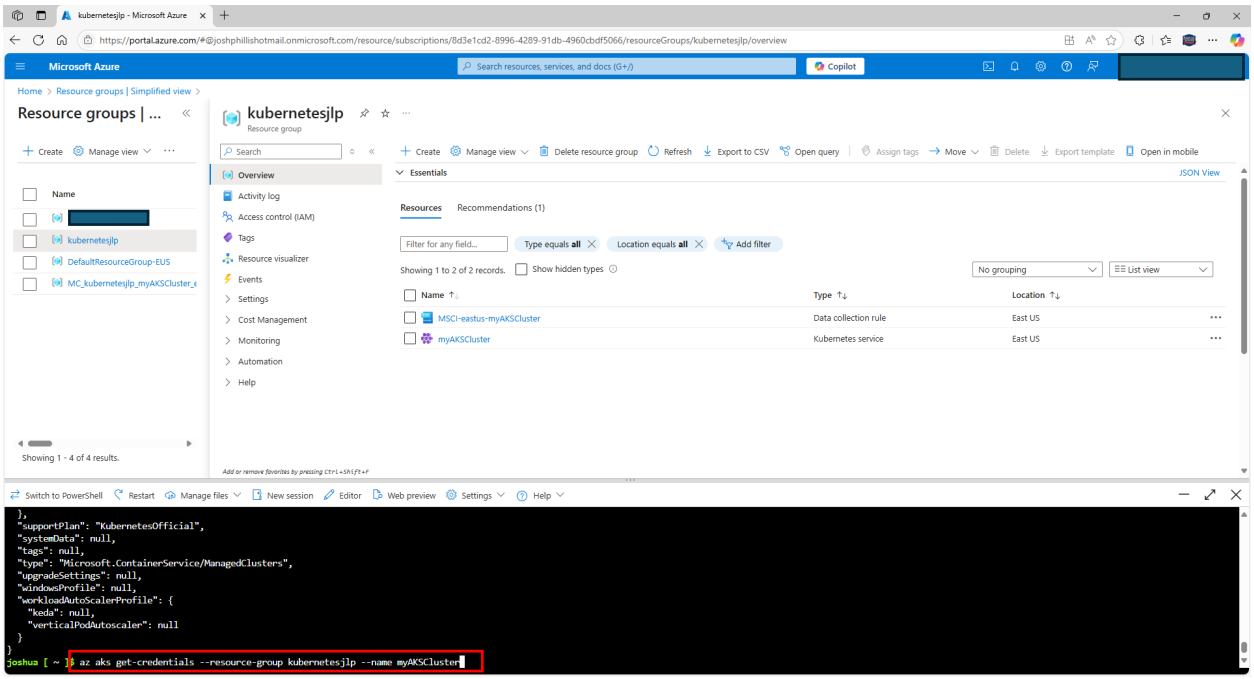
--enable-addons monitoring \

--generate-ssh-keys

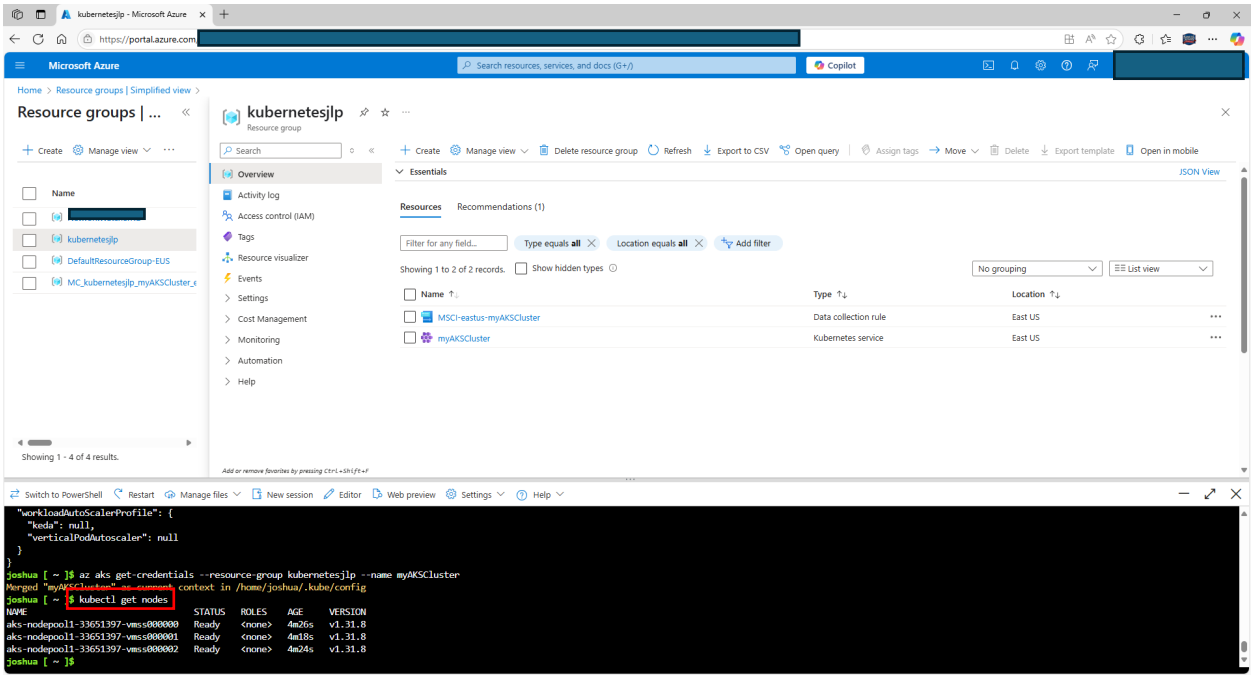


4. Connect to the Cluster

Enter: az aks get-credentials --resource-group kubernetesjlp --name myAKSCluster



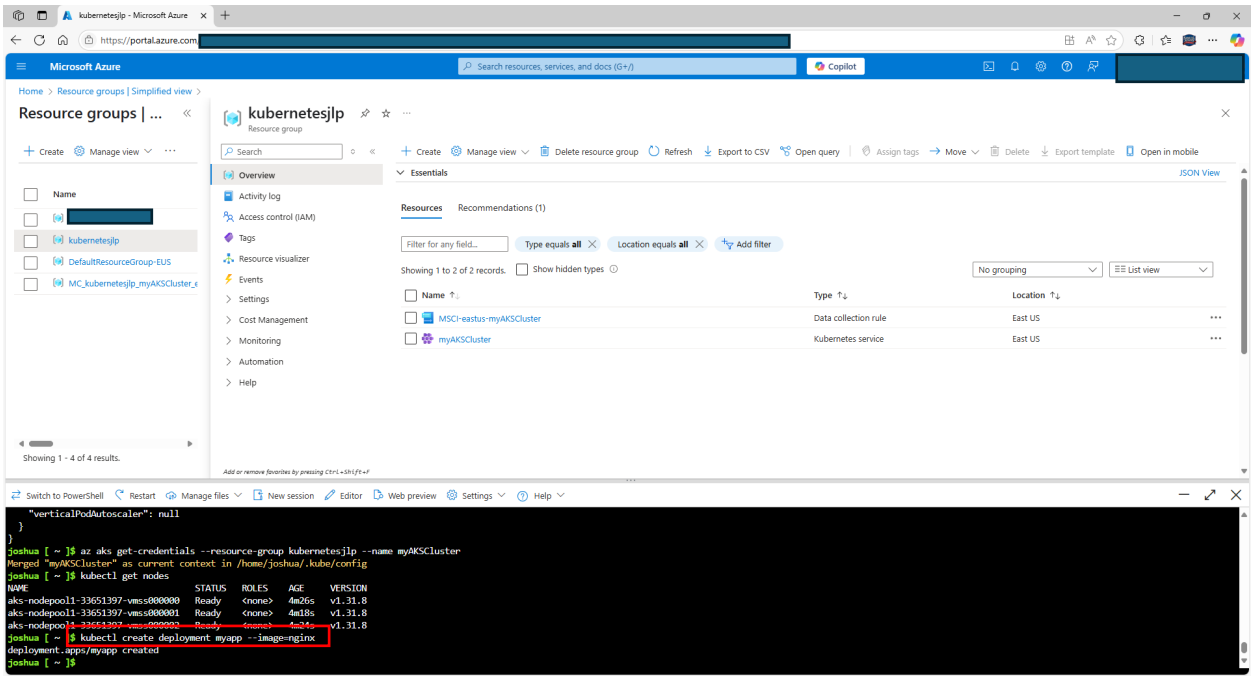
Enter: kubectl get nodes



5. Deploy Your Application

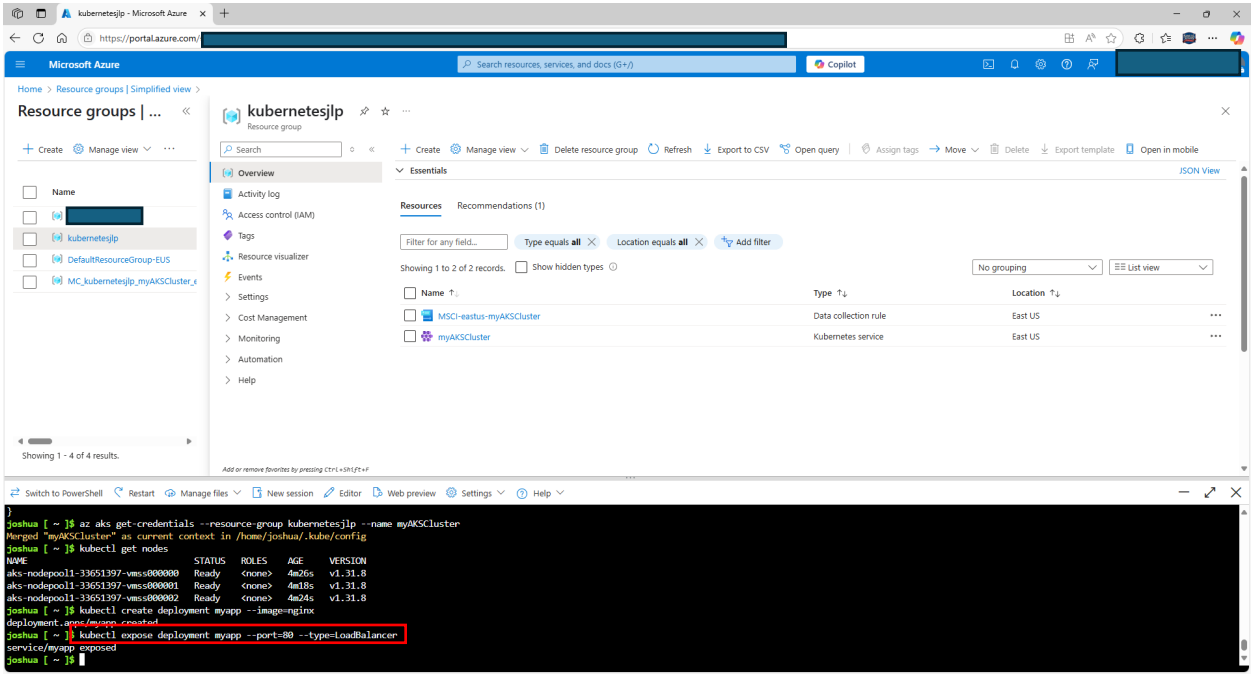
Enter: kubectl create deployment myapp --image=nginx

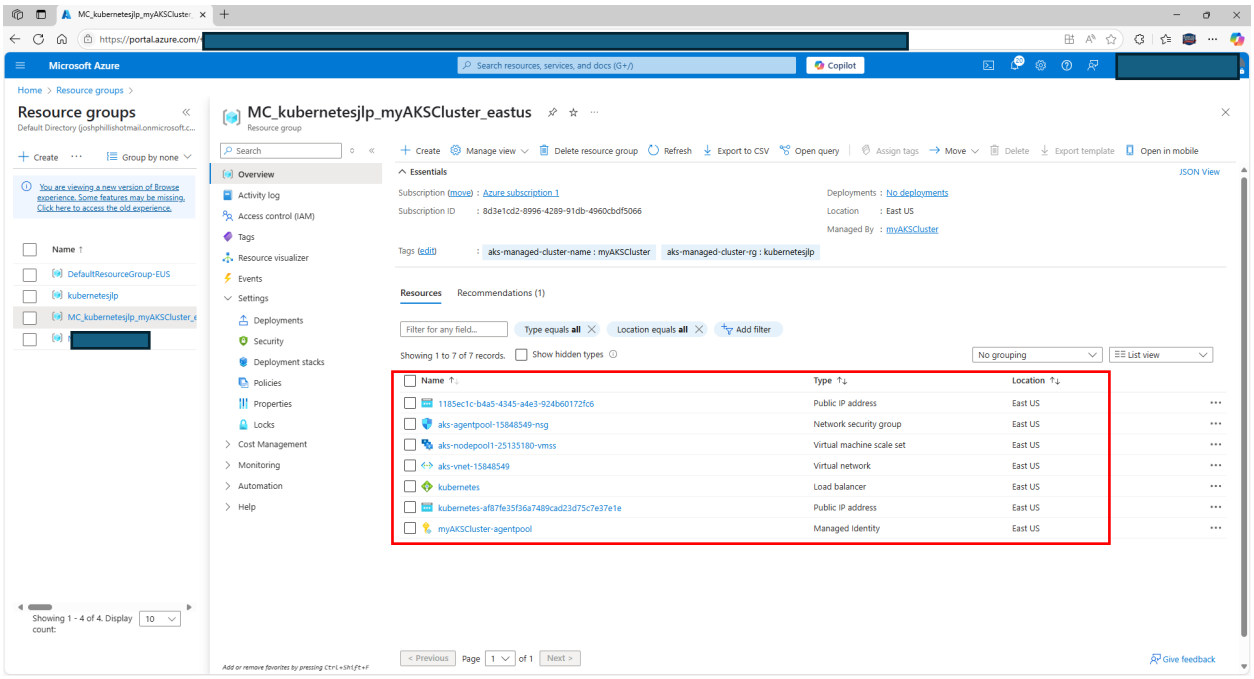
Output: **myapp** was created



Enter: kubectl expose deployment myapp --port=80 –type=LoadBalancer

Output: **myapp** exposed to **port 80** with a **Load Balancer**





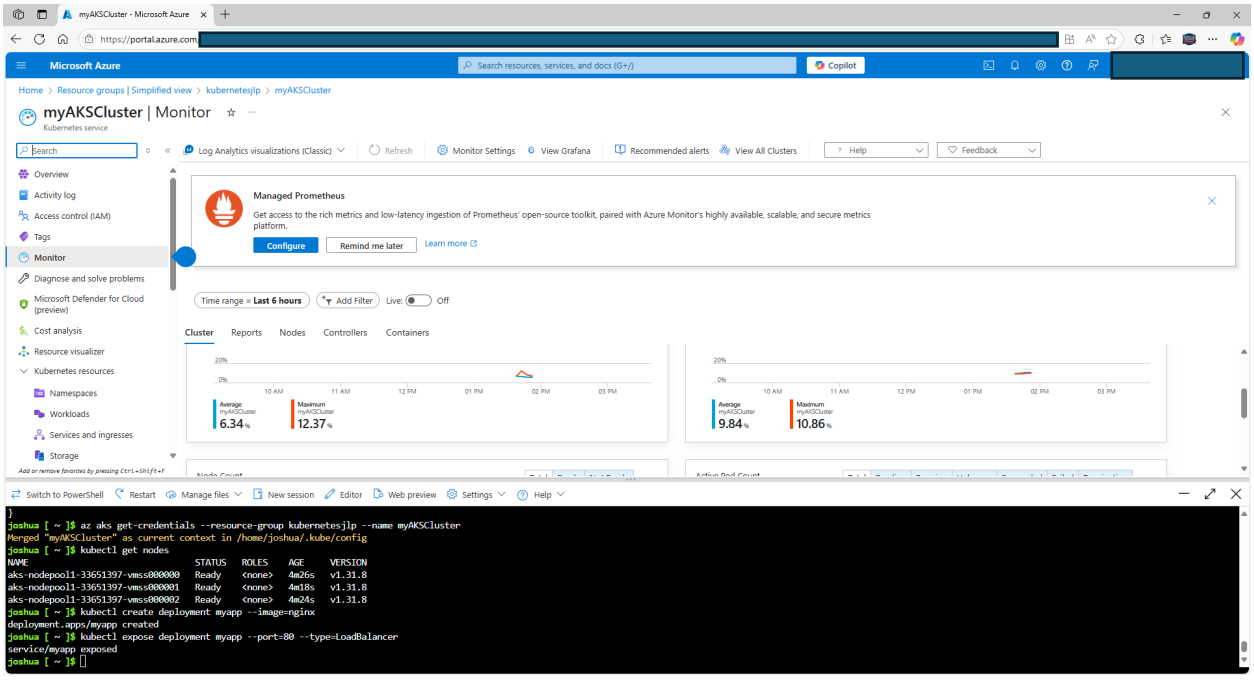
6. Monitor the Cluster

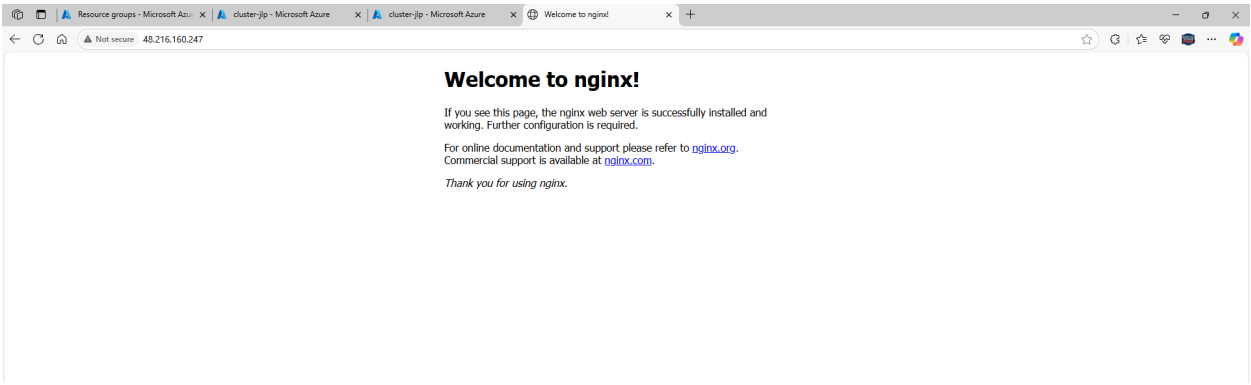
From portal, select **kubernetesjlp** Resource Group

Select **myAKSCluster**

Select **Monitor**

Select **Insights** to view Cluster metrics





Problem Statement:
The goal was to secure a Kubernetes application hosted on an Azure Kubernetes Service (AKS) cluster by enabling HTTPS to protect data in transit and enhance user trust.

Challenges:
While the AKS resource group and cluster were created via the Azure portal, automating TLS certificate issuance and renewal through Kubernetes required CLI interaction. Proper DNS setup and ingress configuration were necessary to route HTTPS traffic correctly. Certificate issuance could also fail if placeholder or invalid domains were used.

Tools & Techniques:
The NGINX Ingress Controller managed external access to the app. cert-manager integrated with Let's Encrypt automated certificate provisioning and renewal. DNS was configured to point the domain to the cluster's ingress IP. Most Kubernetes commands and resource management were performed through Bash using kubectl.

Outcomes:
The application became securely accessible over HTTPS with trusted certificates. Certificate lifecycle management was automated with minimal downtime. Traffic between clients and the ingress was encrypted, and a repeatable process for deploying TLS-secured applications in Kubernetes was established.