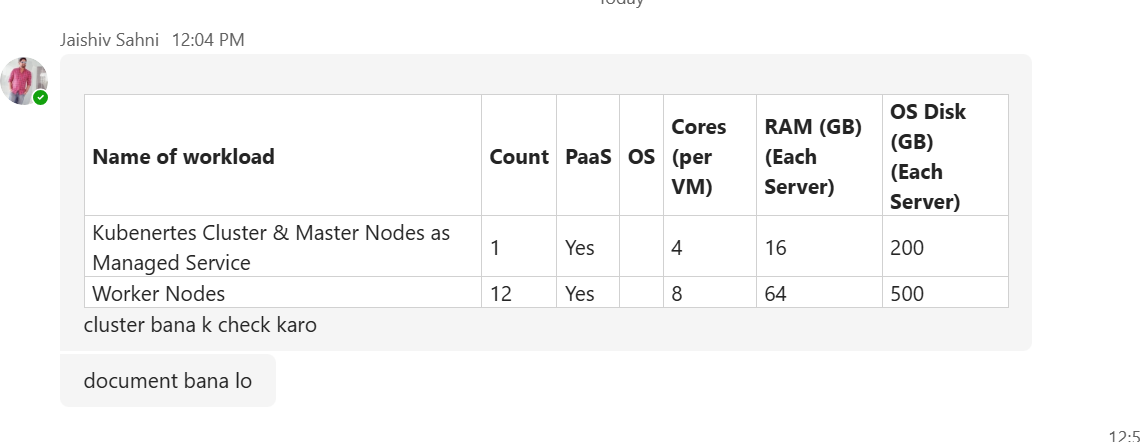
Date – 10 nov.2025

Topic deploy EKs and test-



Already created a user and attached admin privileges permission.

Objective:-

* Control plane: Managed (Amazon EKS)
* Worker nodes: 12 VMs, 8 vCPU / 64 GB RAM / 500 GB disk each (I choose instance type r5.2xlarge which is 8 vCPU & 64 GB RAM)
* Download kubectl in our local machine powershell

**Step-by-step to install kubectl on Windows**

Step 1 — Open PowerShell as Administrator

* Click on Start
* Type PowerShell
* Right-click - Run as Administrator

**Step 2 — Make a folder for kubectl**

Copy-paste this in PowerShell and press Enter:

New-Item -ItemType Directory -Force -Path "C:\kubectl"

**Step 3 — Download kubectl**

Now copy this whole command (make sure it’s exactly like this) and press Enter:

Invoke-WebRequest -Uri "https://dl.k8s.io/release/v1.30.0/bin/windows/amd64/kubectl.exe" -OutFile "C:\kubectl\kubectl.exe"

This command downloads the kubectl file from the official Kubernetes website and saves it in C:\kubectl.

**Step 4 — Add kubectl to your PATH**

So you can run it from anywhere:

setx PATH "$($Env:PATH);C:\kubectl"

Then close PowerShell and open it again (important).

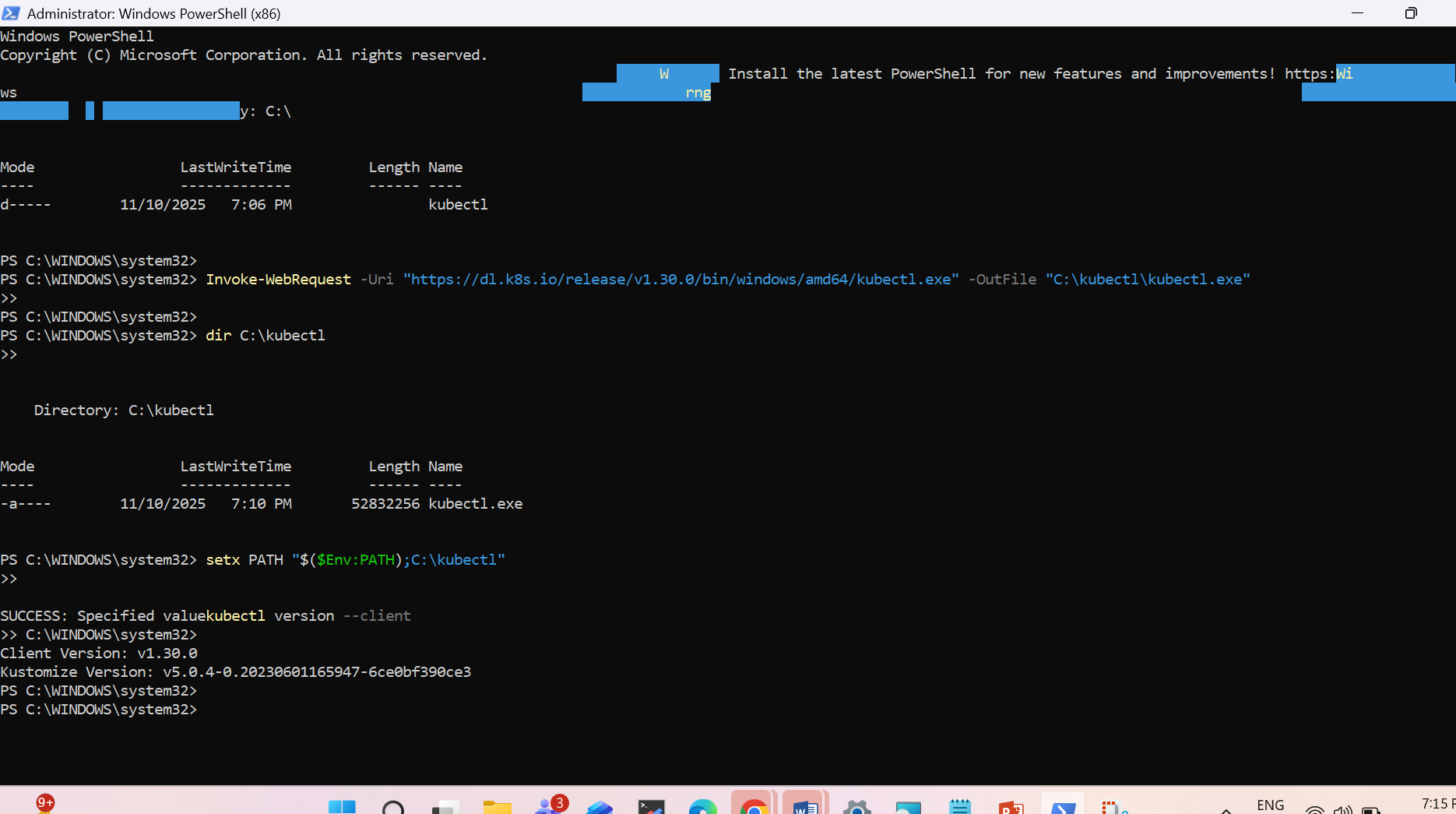
**Step 5 — Check if it’s working**

Type this and press Enter:

kubectl version --client

If everything worked, you’ll see something like:

Client Version: v1.30.0



You now have kubectl installed on your system

**For Windows (PowerShell)**

Run this command:

choco install eksctl -y

If Chocolatey isn’t installed, run this first:

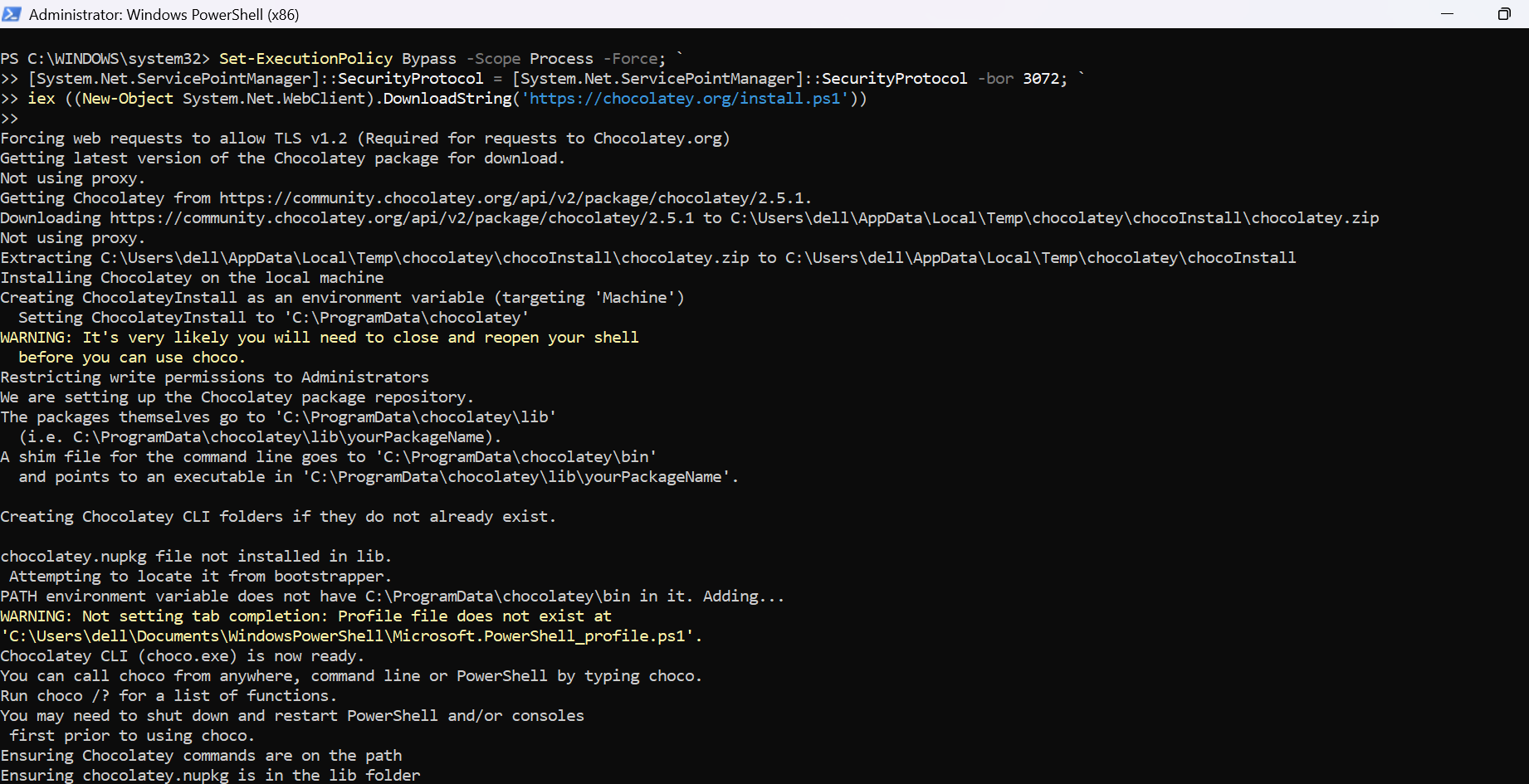
Set-ExecutionPolicy Bypass -Scope Process -Force; `

[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `

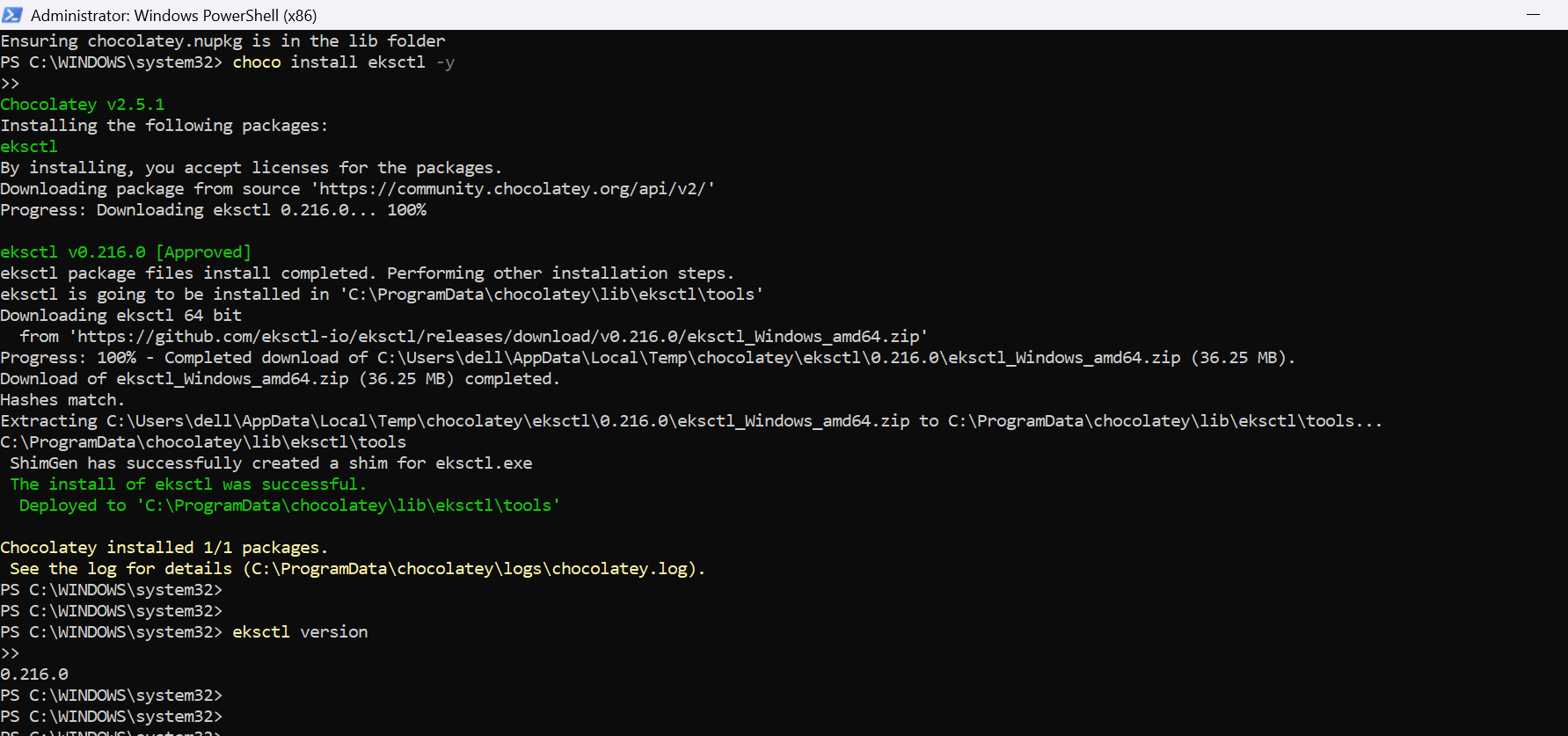
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

Then re-run: # choco install eksctl -y

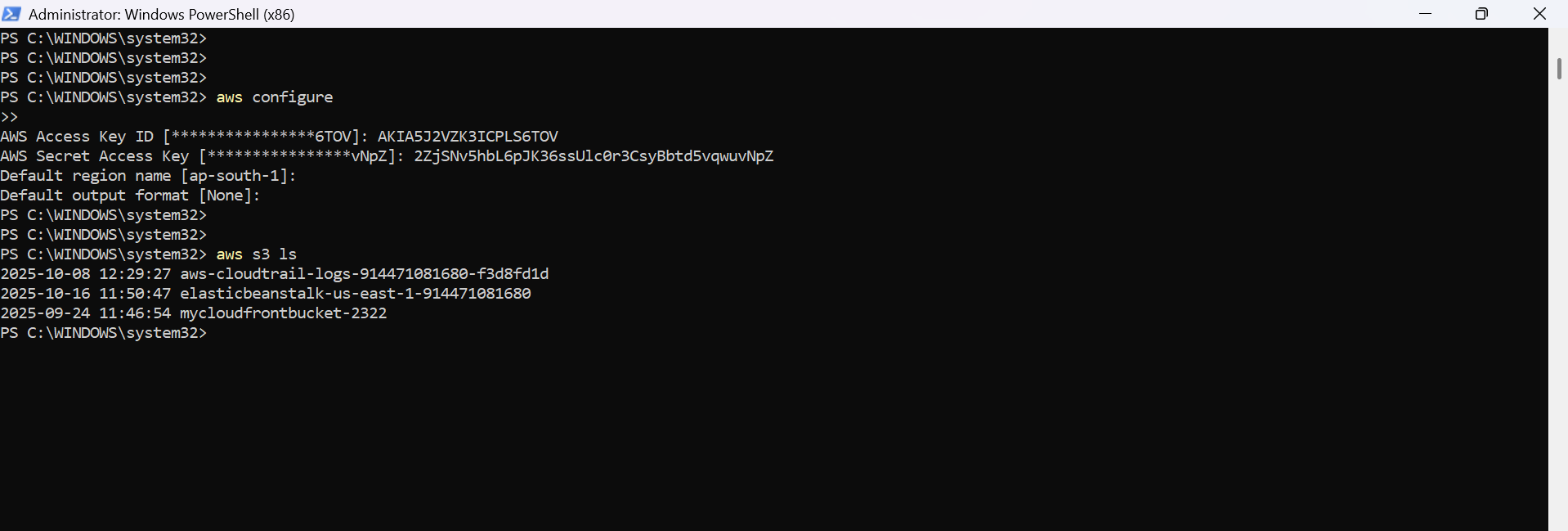
Verify it worked: eksctl version



choco install eksctl -y



AWS CLI configure



Step 4: Create Your Kubernetes Cluster (EKS)

Now we’ll create:

1 managed control plane (EKS Cluster)

12 worker nodes (EC2 instances)

Run this in PowerShell:

eksctl create cluster `

--name my-cluster `

--region ap-south-1 `

--version 1.30 `

--nodegroup-name worker-nodes `

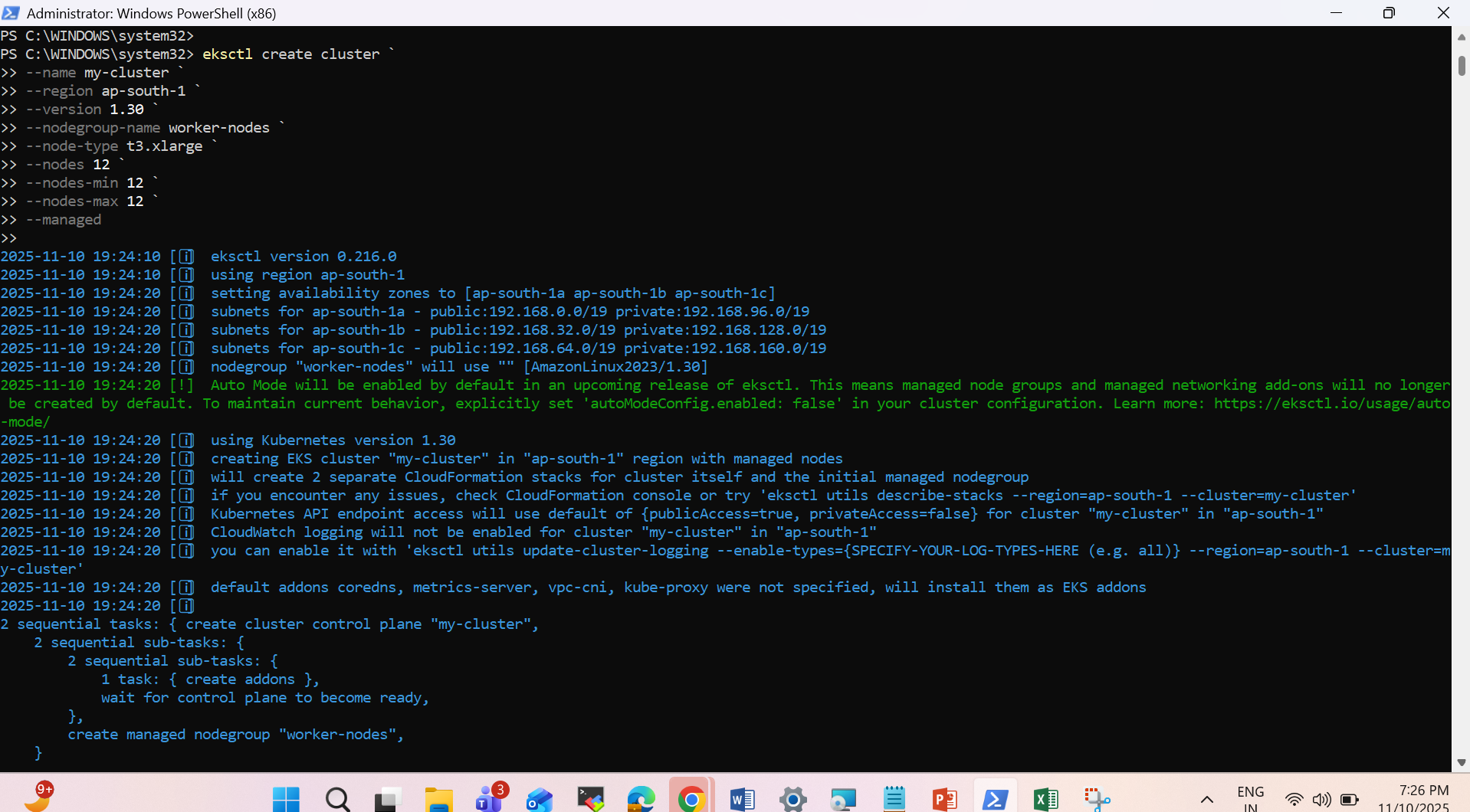
--node-type t3.xlarge `

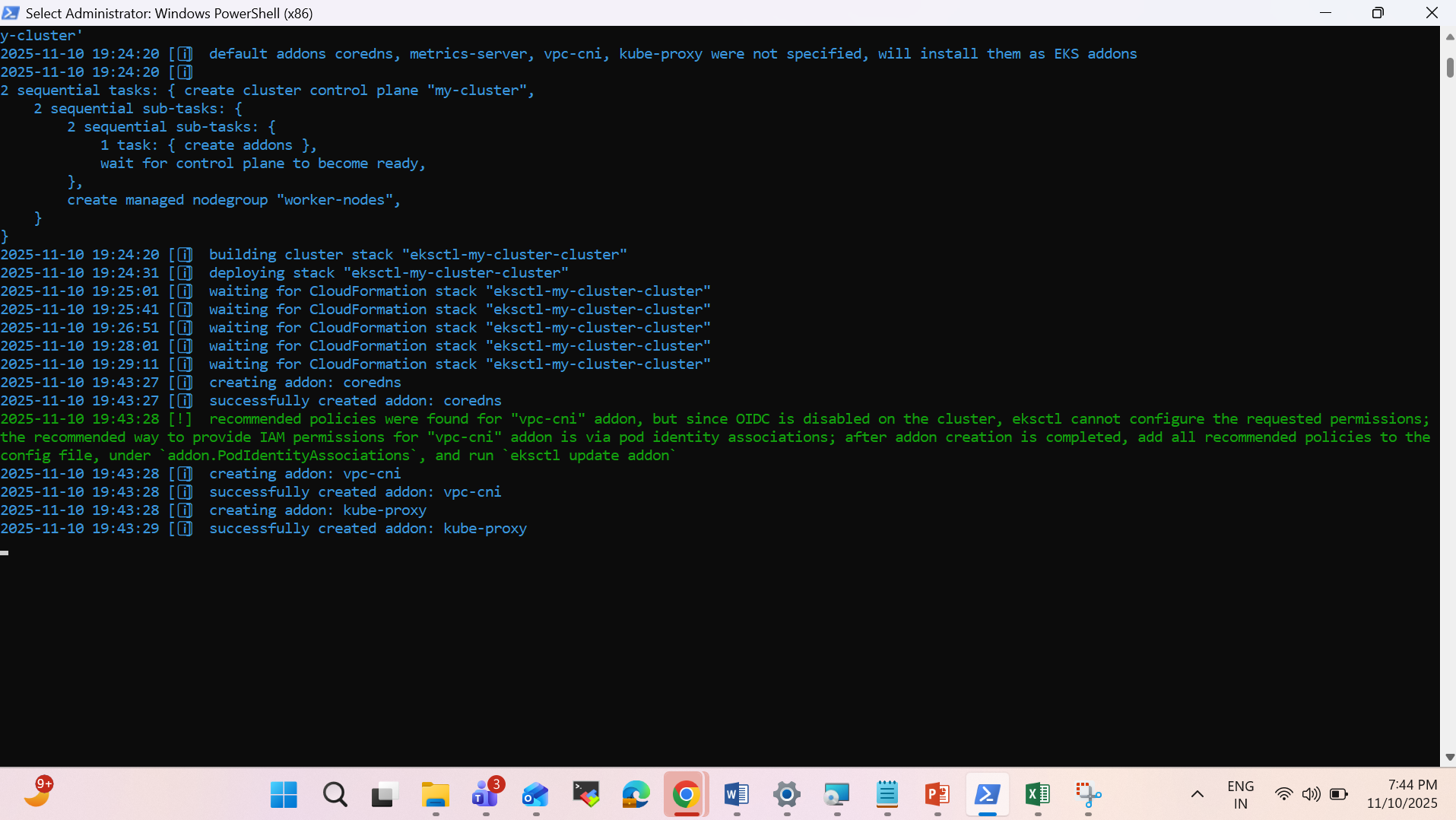
--nodes 12 `

--nodes-min 12 `

--nodes-max 12 `

--managed

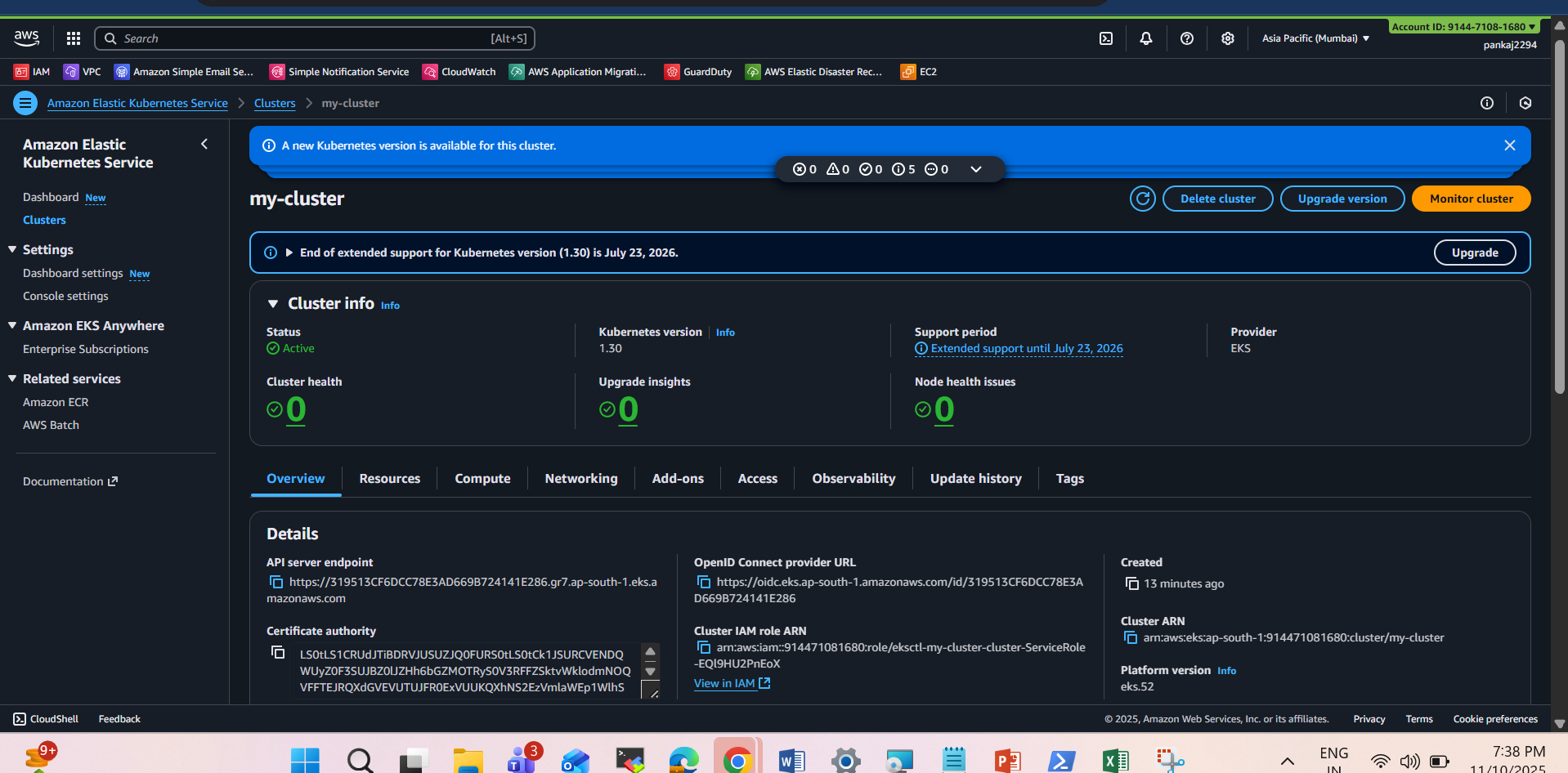


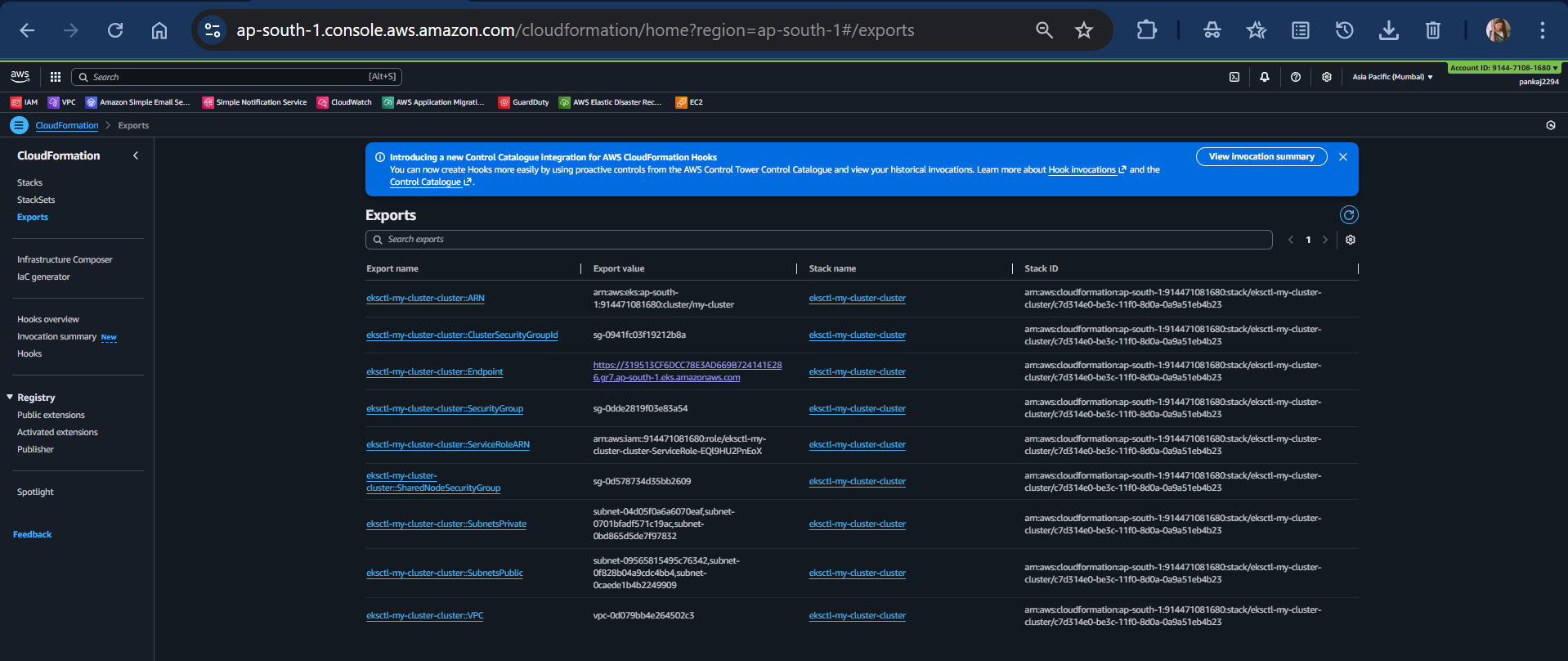


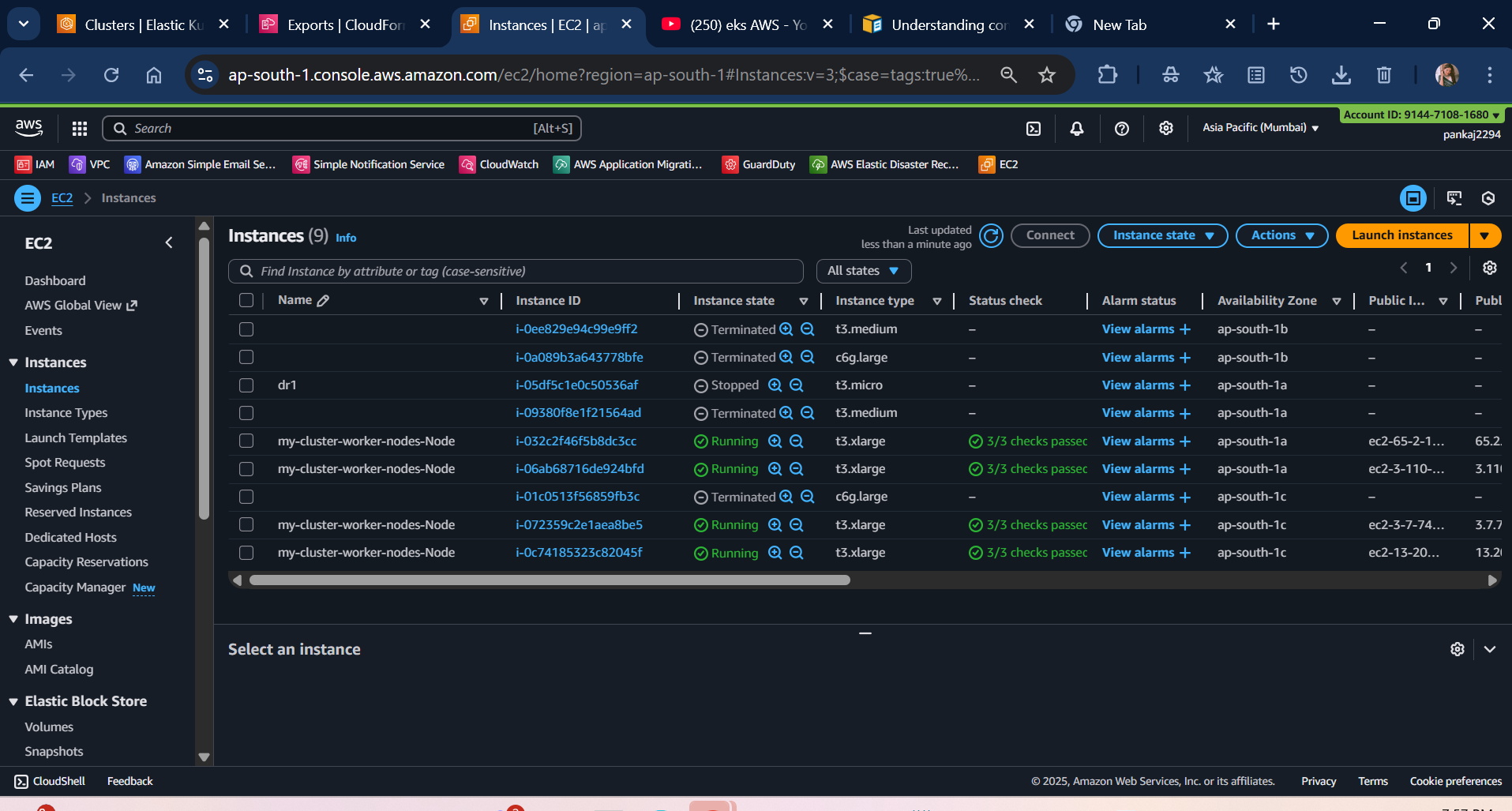
It still takes more than 25 minutes

What this does:

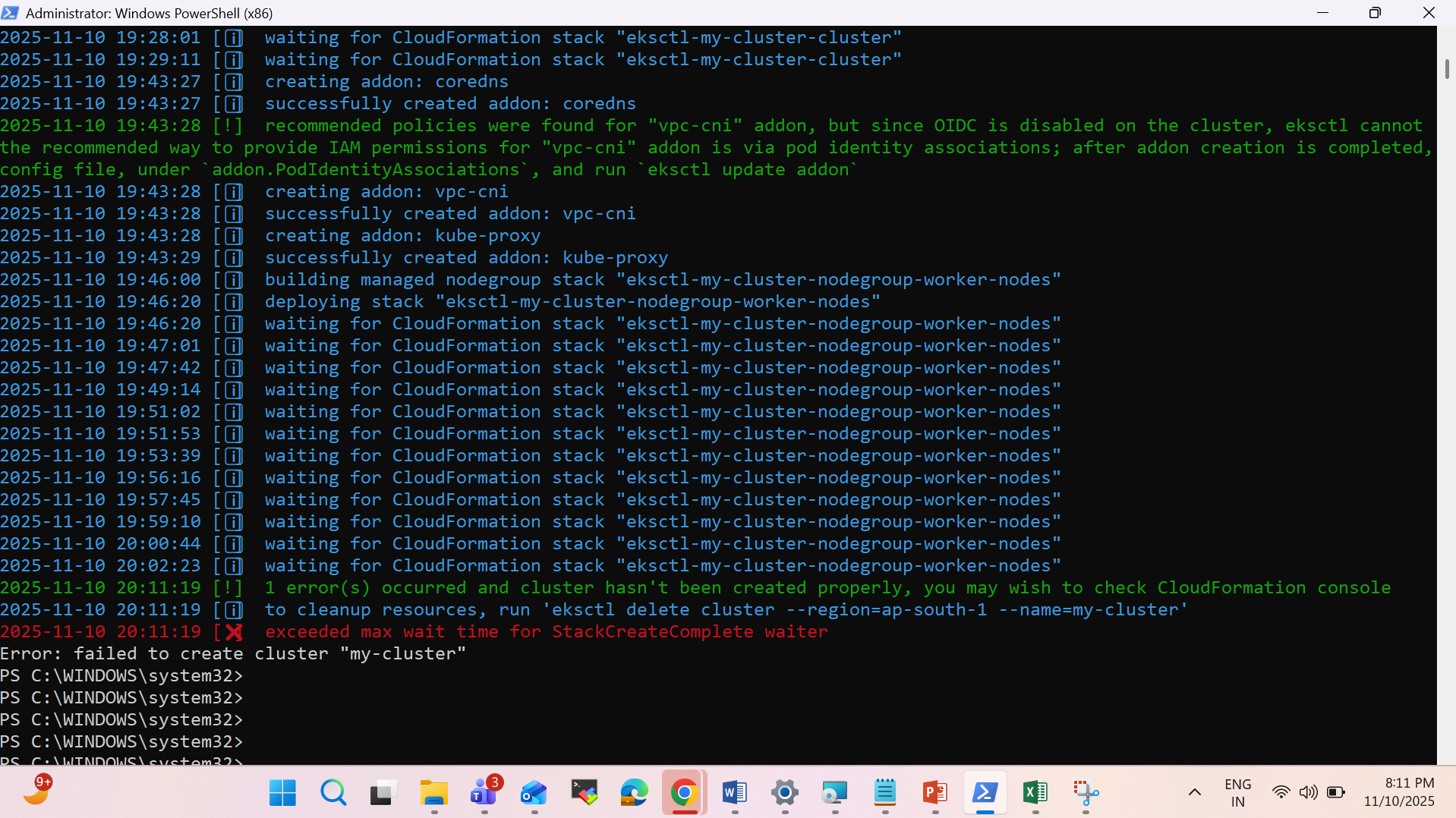
* Creates an EKS control plane (Kubernetes Master nodes managed by AWS)
* Launches 12 worker nodes (EC2 instances) with 8 vCPU & 64 GB RAM each (as per your plan)
* Automatically connects them to your cluster
* Sets up kubectl so you can manage your cluster right away

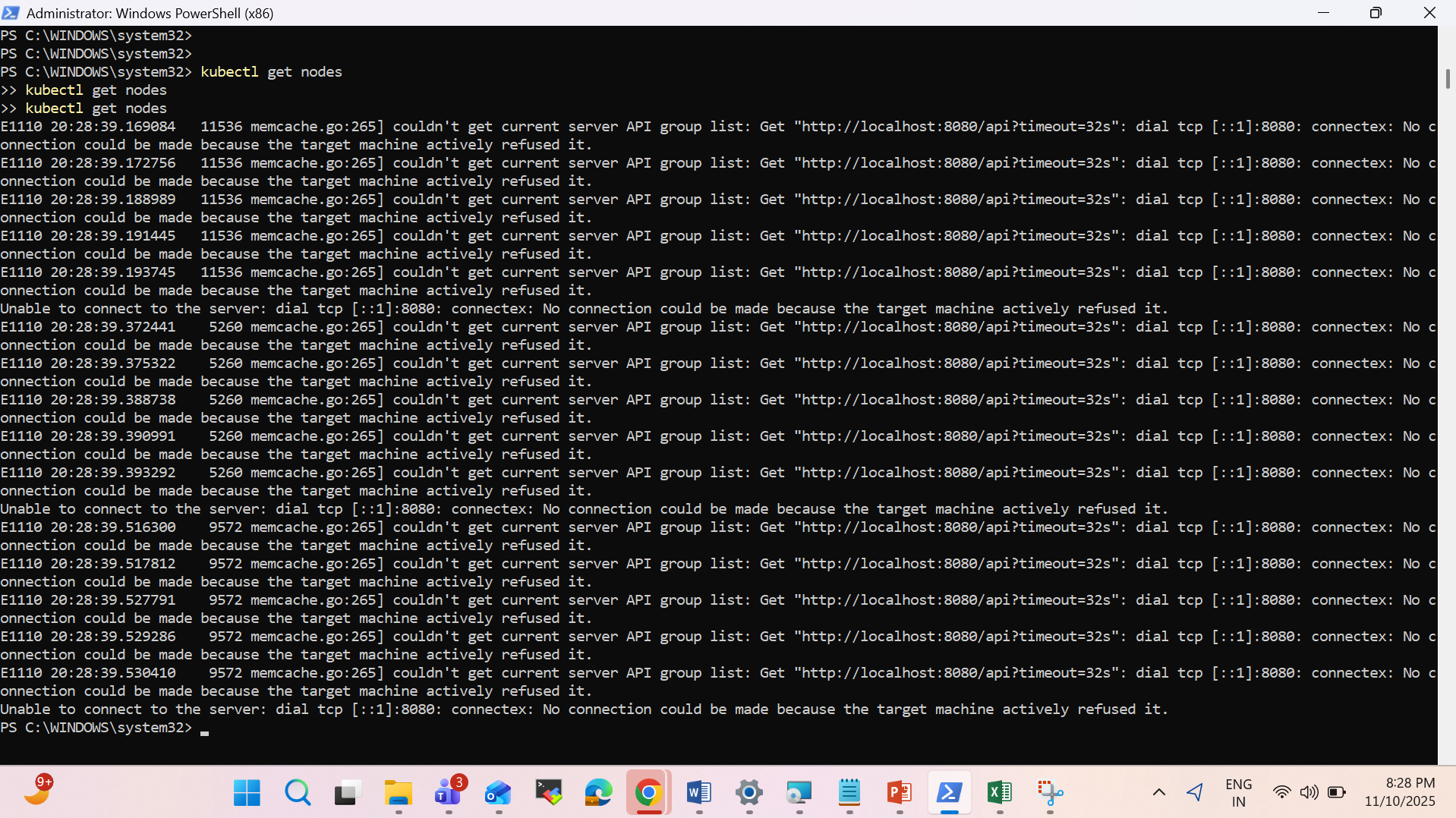












Meaning:

kubectl is trying to talk to a Kubernetes cluster on your local machine (localhost:8080) — but there’s no cluster running yet, or it’s not configured.

In simple words:

kubectl is ready, but it doesn’t know which cluster to connect to.

We must now connect it to your real cluster (EKS) using AWS CLI.

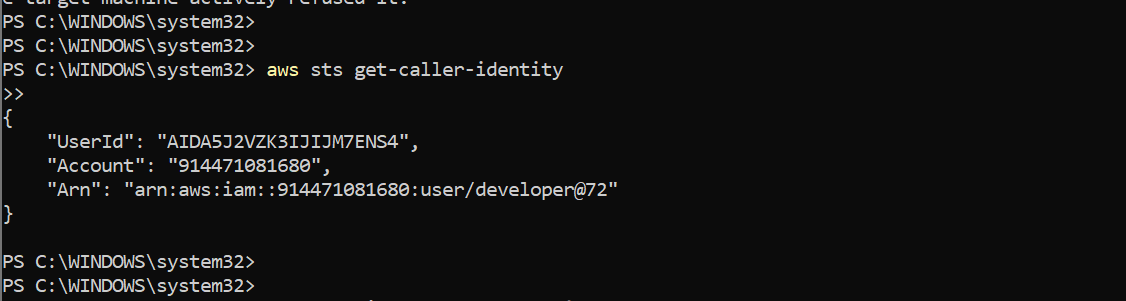
Fix Step-by-Step

Step 1 — Confirm AWS CLI is working

Run:

aws sts get-caller-identity

You should see something like:

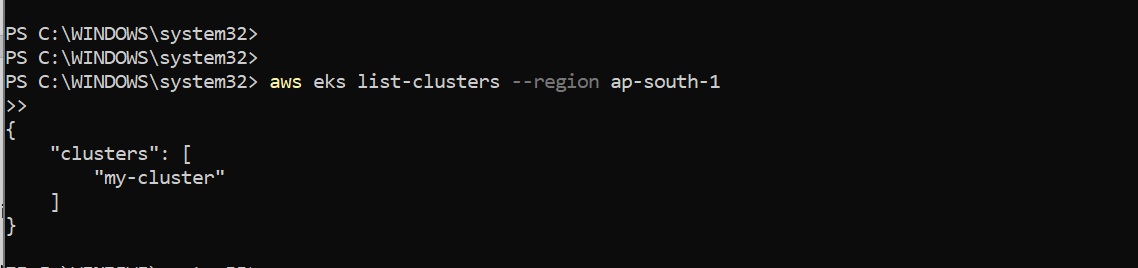


If this works, your AWS credentials are fine.

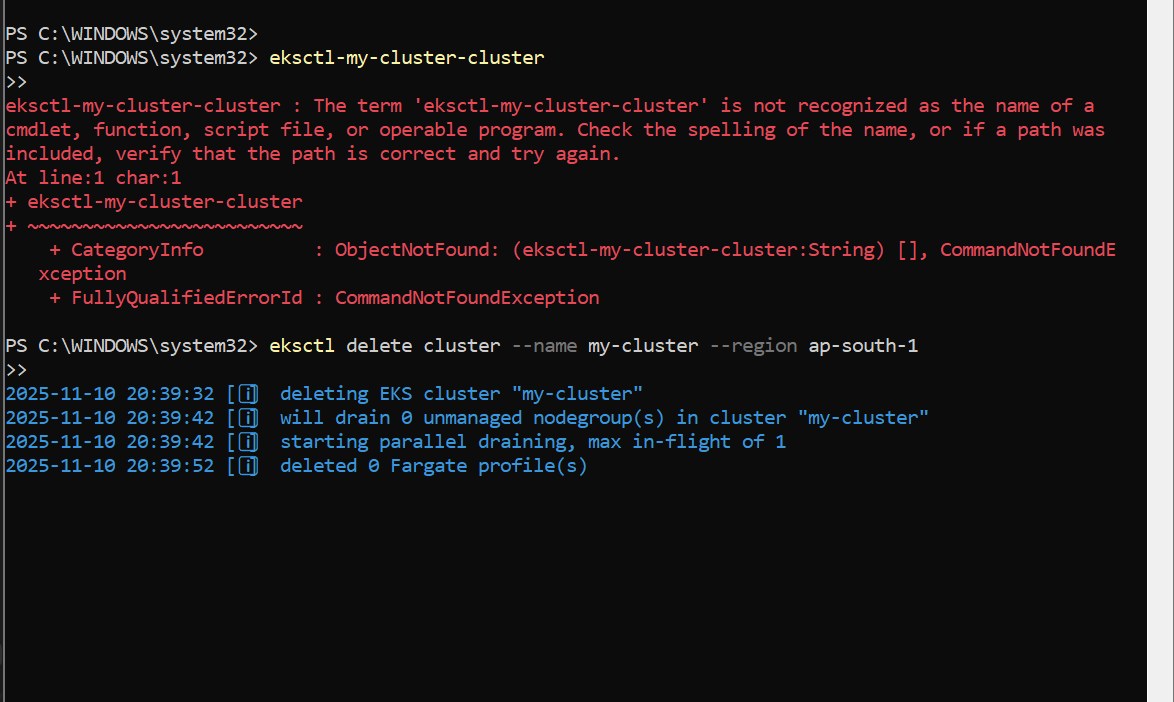
Make sure your EKS cluster exists

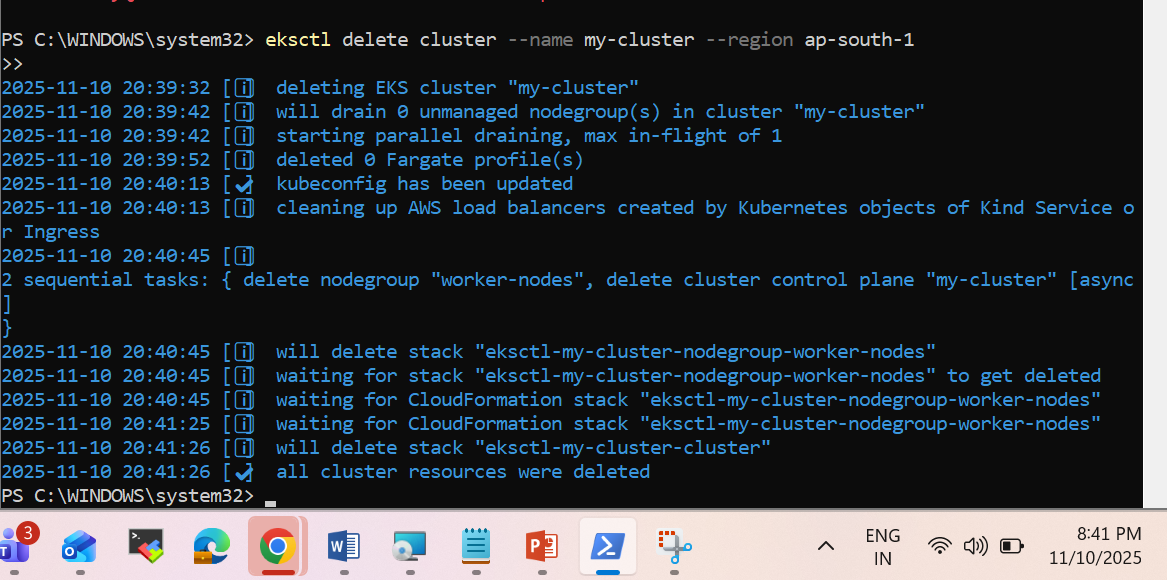
Run:

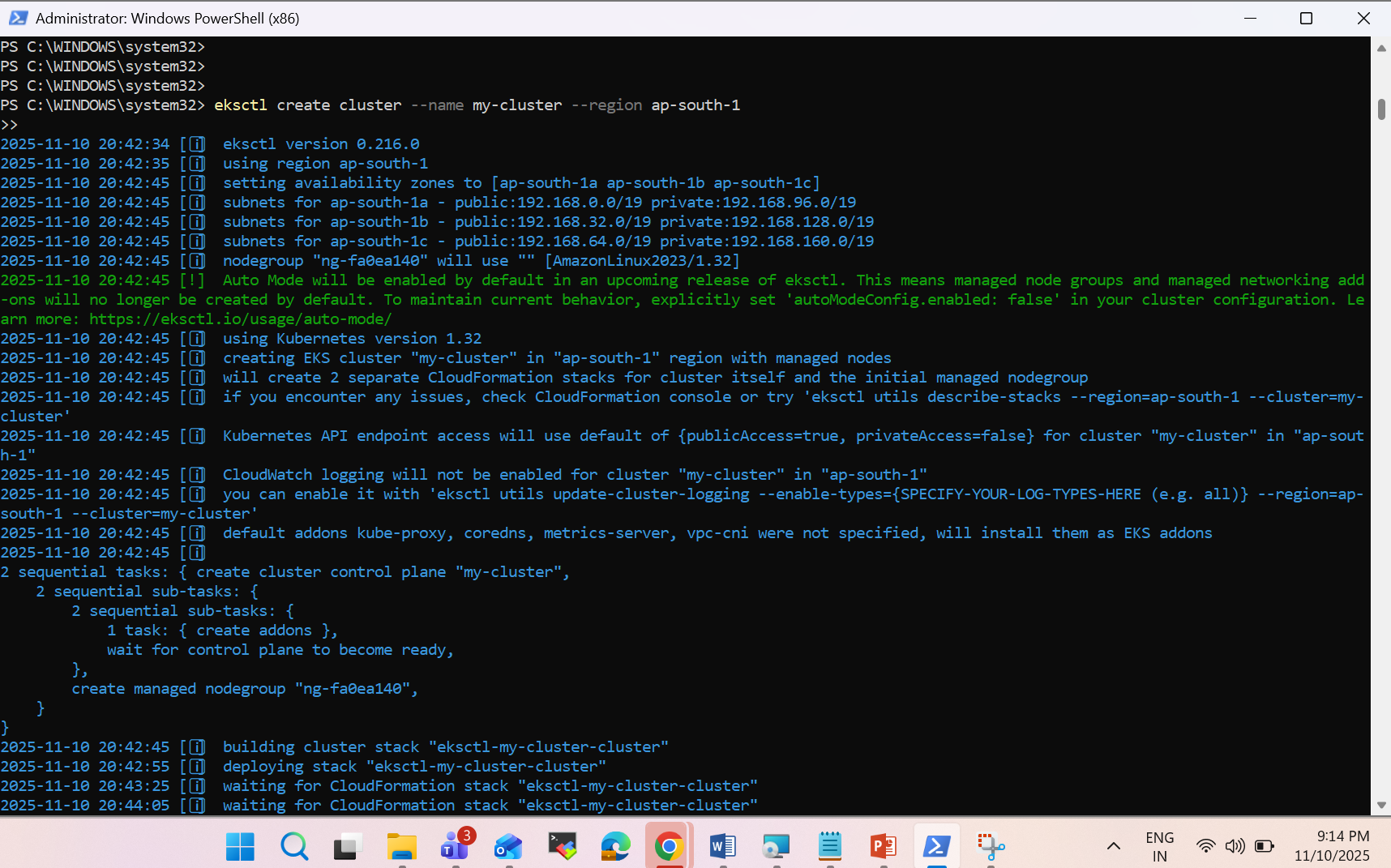
aws eks list-clusters --region ap-south-1



Deleting first cluster and will another to finish it soon.



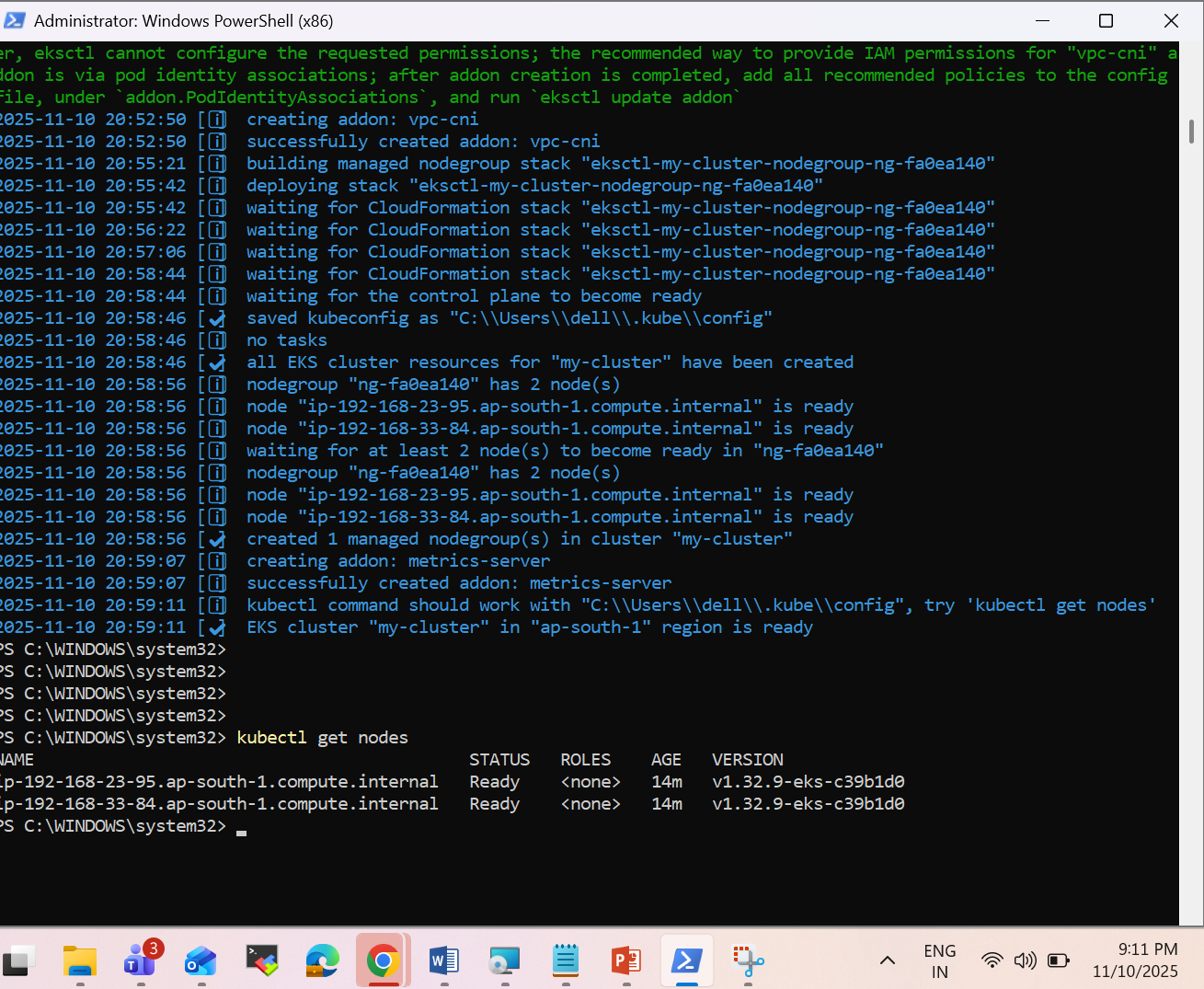


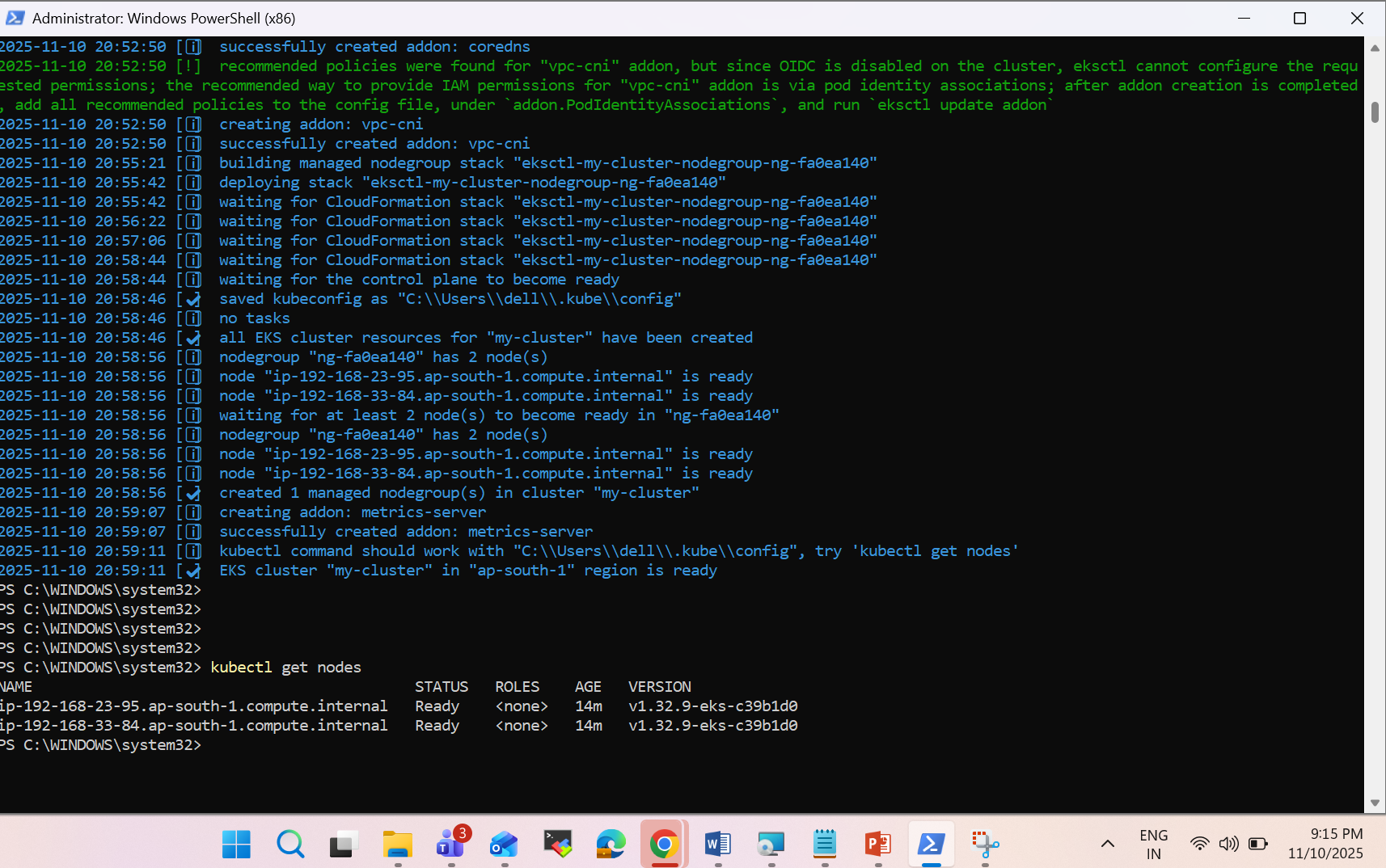


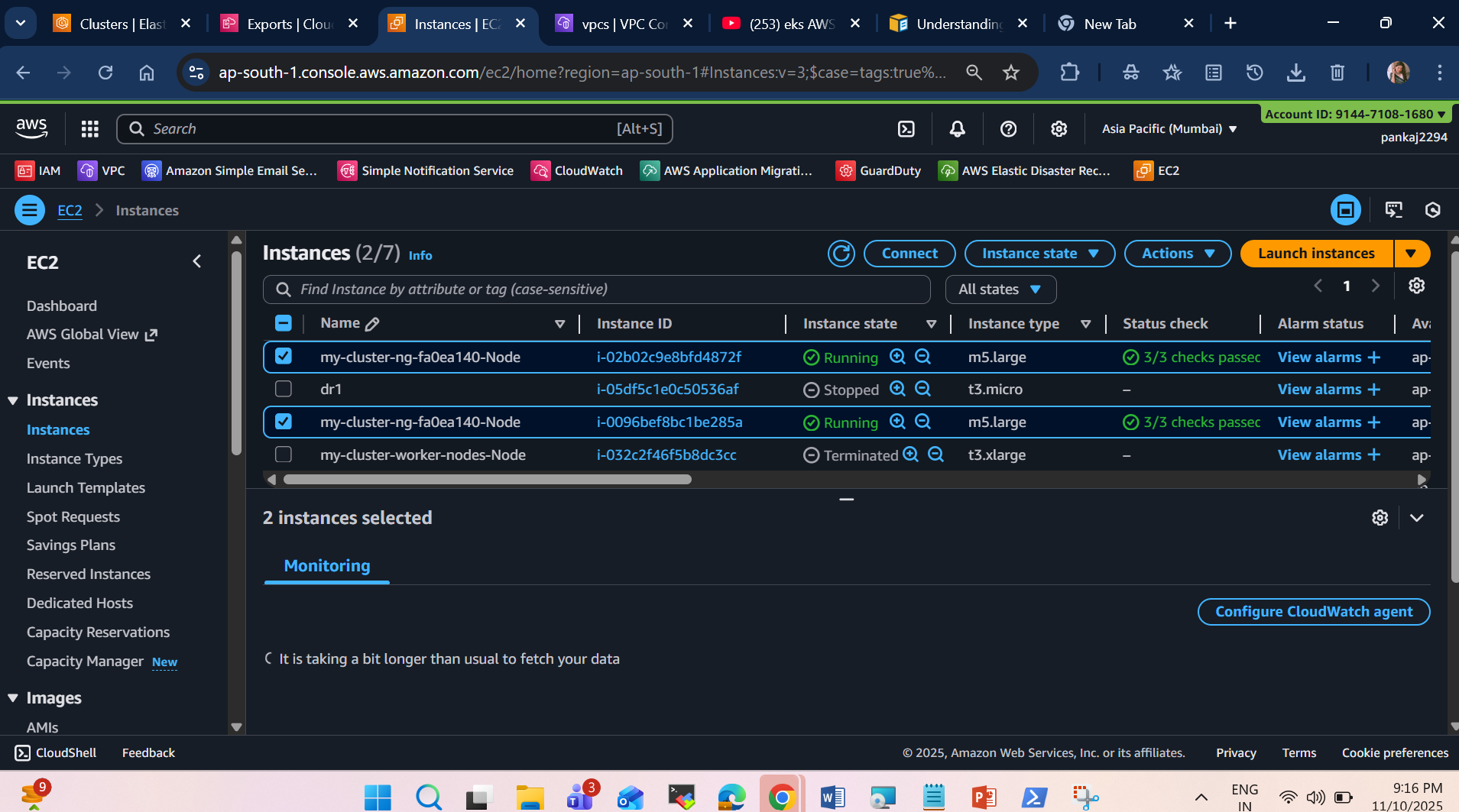
**Test the Connection**

Now try:

# kubectl get nodes







Verify cluster creation

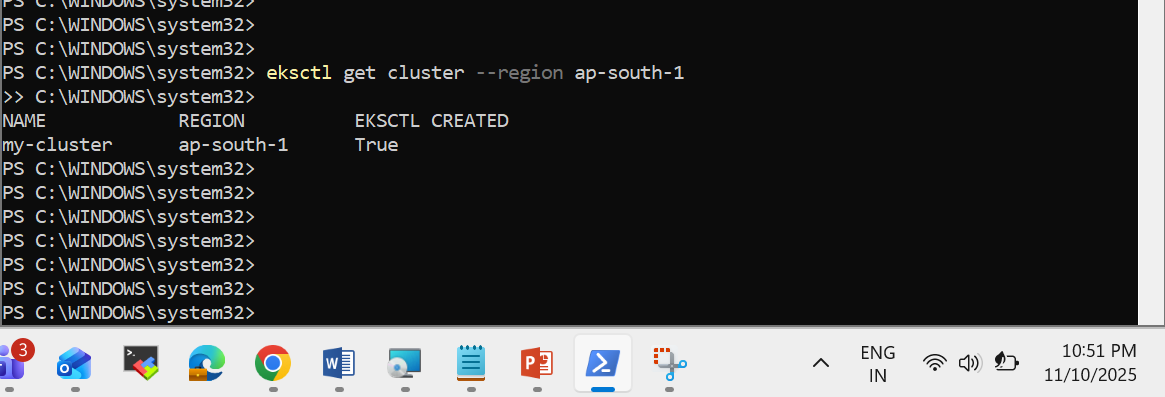
After it completes, check your clusters:

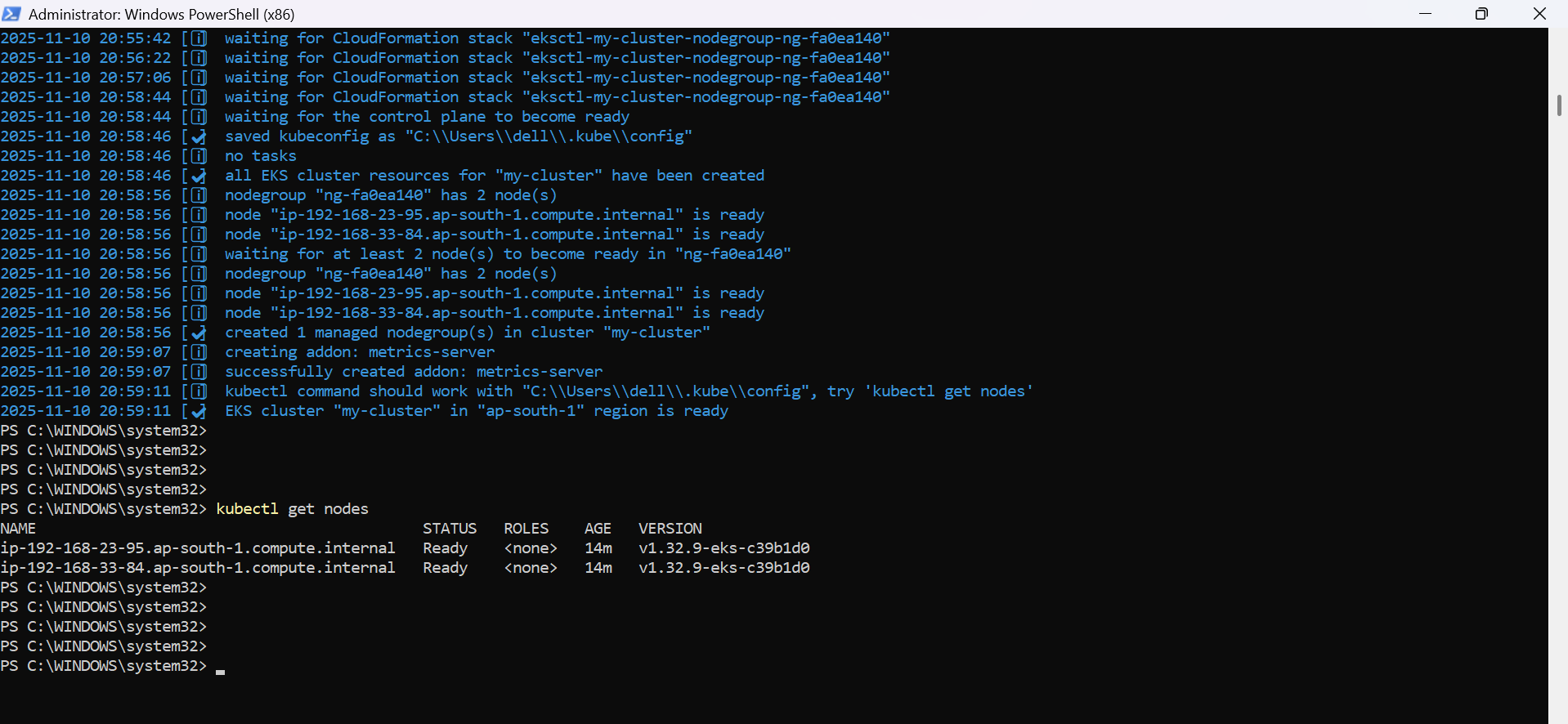
eksctl get cluster --region ap-south-1

You should see something like:

NAME REGION EKSCTL CREATED

my-cluster ap-south-1 True





**Test the Cluster**

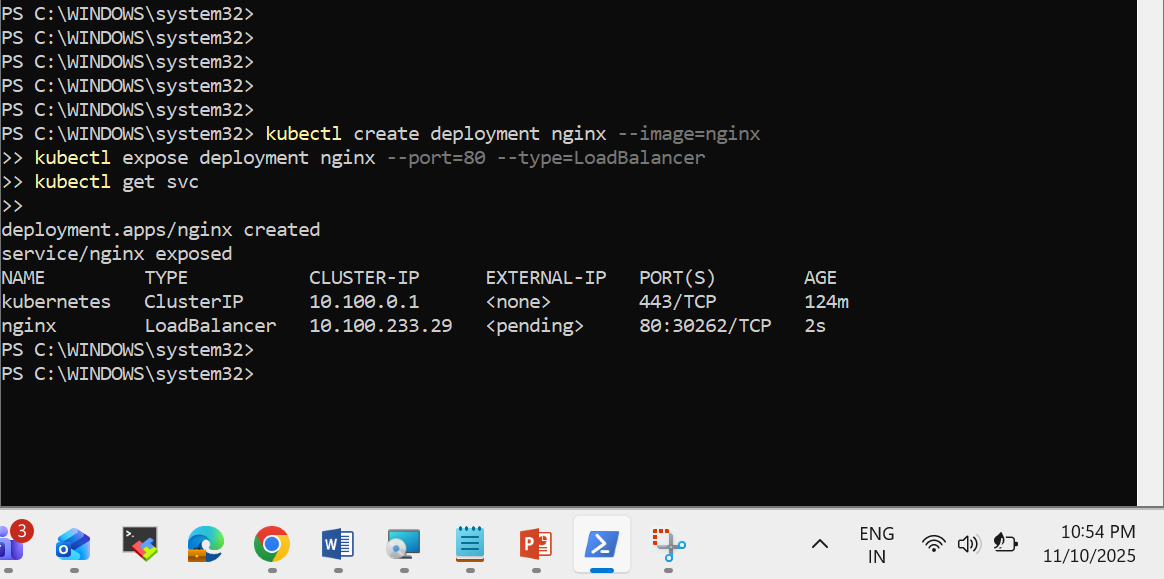
Let’s test by deploying a sample Nginx web app:

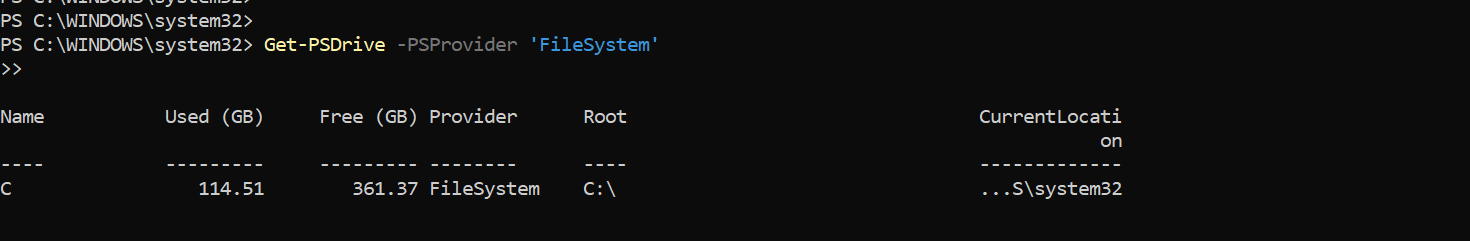
kubectl create deployment nginx --image=nginx

kubectl expose deployment nginx --port=80 --type=LoadBalancer

kubectl get svc

When you get the EXTERNAL-IP, open it in your browser — you’ll see the Nginx welcome page.







We will get the same details - PaaS, OS, CPU, RAM, Disk size by using the below code — shown neatly in your PowerShell terminal.

Write-Host "========================================"

Write-Host "🔹 SERVER SPECIFICATION DETAILS"

Write-Host "========================================"

# OS Information

$os = (Get-CimInstance Win32\_OperatingSystem).Caption

Write-Host "Operating System : $os"

# CPU Cores

$cores = (Get-WmiObject Win32\_Processor | Measure-Object -Property NumberOfCores -Sum).Sum

Write-Host "CPU Cores (per VM) : $cores"

# RAM in GB

$ram = [math]::Round((Get-CimInstance Win32\_ComputerSystem).TotalPhysicalMemory / 1GB)

Write-Host "RAM (GB) : $ram"

# OS Disk Size (C drive)

$disk = (Get-PSDrive C).Used + (Get-PSDrive C).Free

$disk\_gb = [math]::Round($disk / 1GB)

Write-Host "OS Disk Size (GB) : $disk\_gb"

# Hostname

$hostname = $env:COMPUTERNAME

Write-Host "Hostname : $hostname"

# Cloud Type Check (basic)

if (Test-Path "C:\ProgramData\Amazon") {

$paas = "AWS EC2 (Windows Instance)"

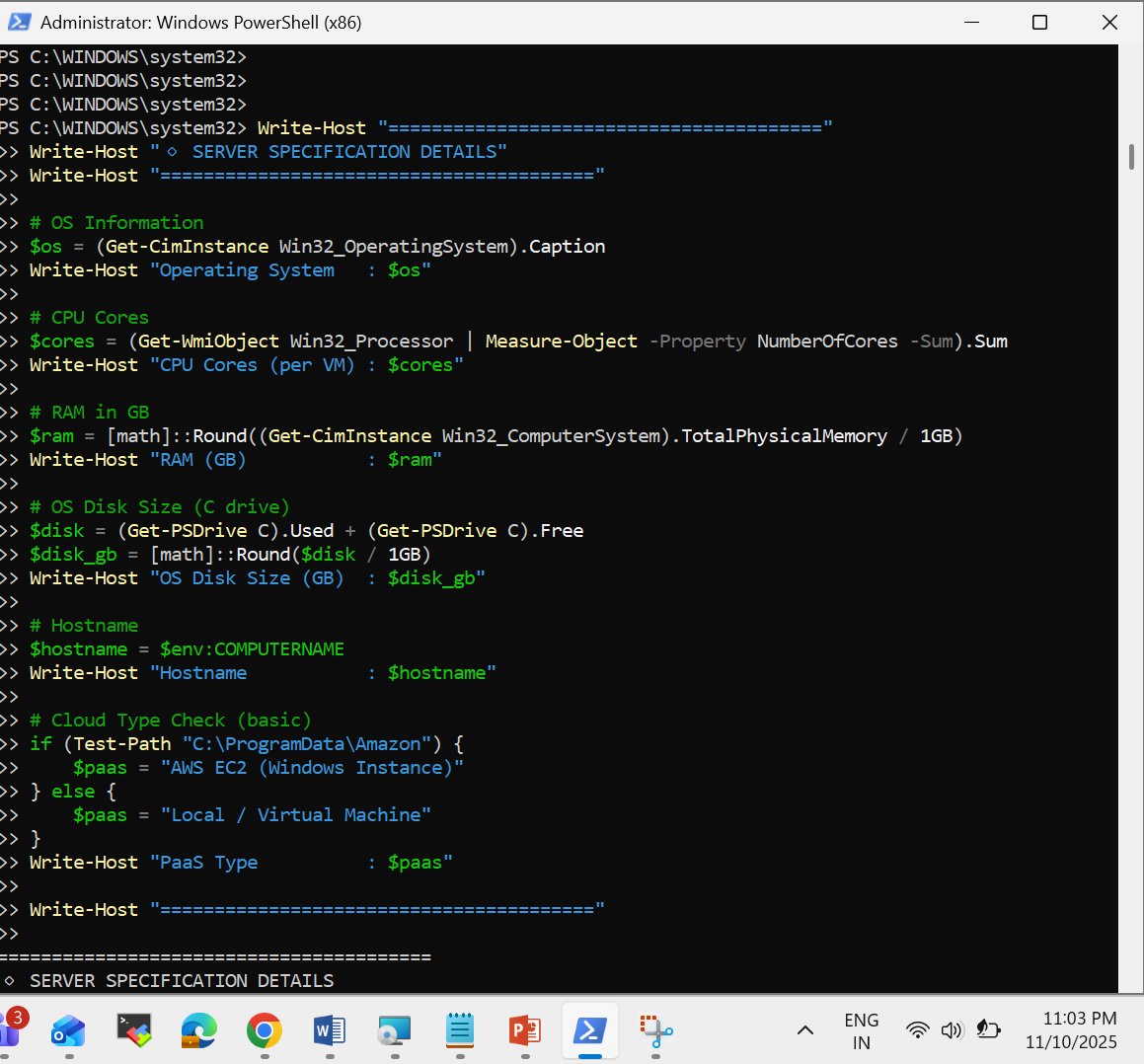
} else {

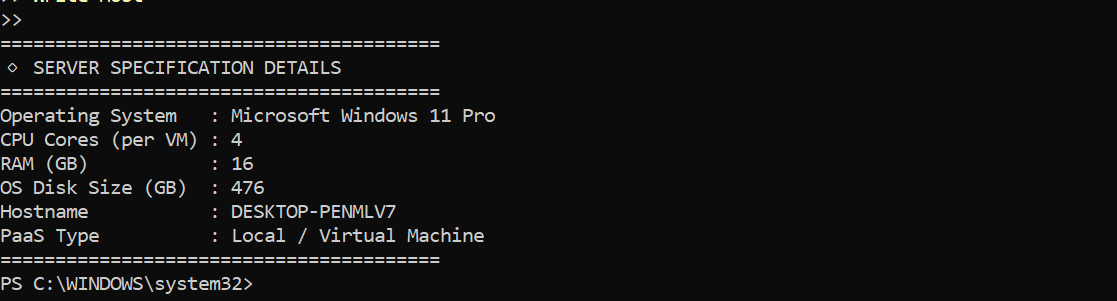
$paas = "Local / Virtual Machine"

}

Write-Host "PaaS Type : $paas"

Write-Host "========================================"





Yes done

our EXTERNAL-IP is showing as <pending>, which means your LoadBalancer service hasn’t been assigned a public IP yet.

This happens because you’re likely running Kubernetes locally (for example, via minikube, kind, Docker Desktop, or a non-cloud cluster).

Only cloud-based clusters (like AWS EKS, Azure AKS, GCP GKE, etc.) automatically provide an external LoadBalancer IP.

To fix this (based on your setup):

If using Minikube

Run this:

# minikube service nginx

This will open NGINX in your default browser using a Minikube tunnel.

Or start a tunnel manually:

# minikube tunnel

Then check again:

# kubectl get svc

You should now see an EXTERNAL-IP assigned (like 127.0.0.1 or another local IP).

If using Docker Desktop (Windows)

Docker Desktop’s built-in Kubernetes doesn’t support LoadBalancer directly.

Use NodePort instead:

# kubectl expose deployment nginx --port=80 --type=NodePort

Then find your port:

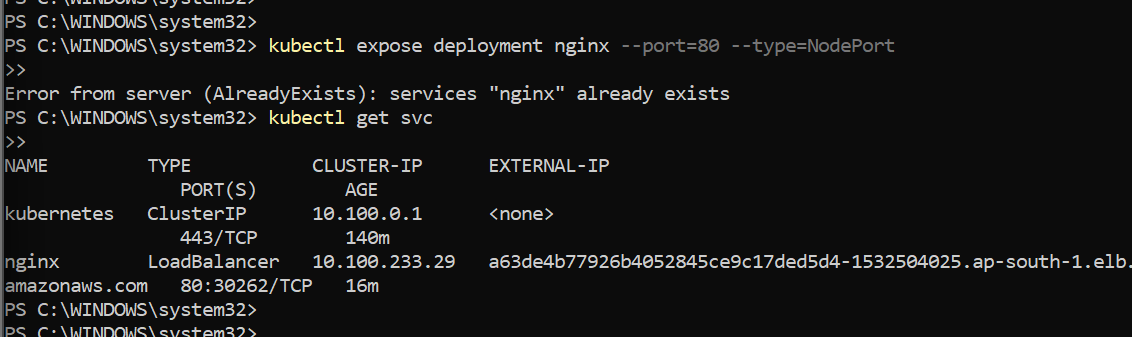
# kubectl get svc

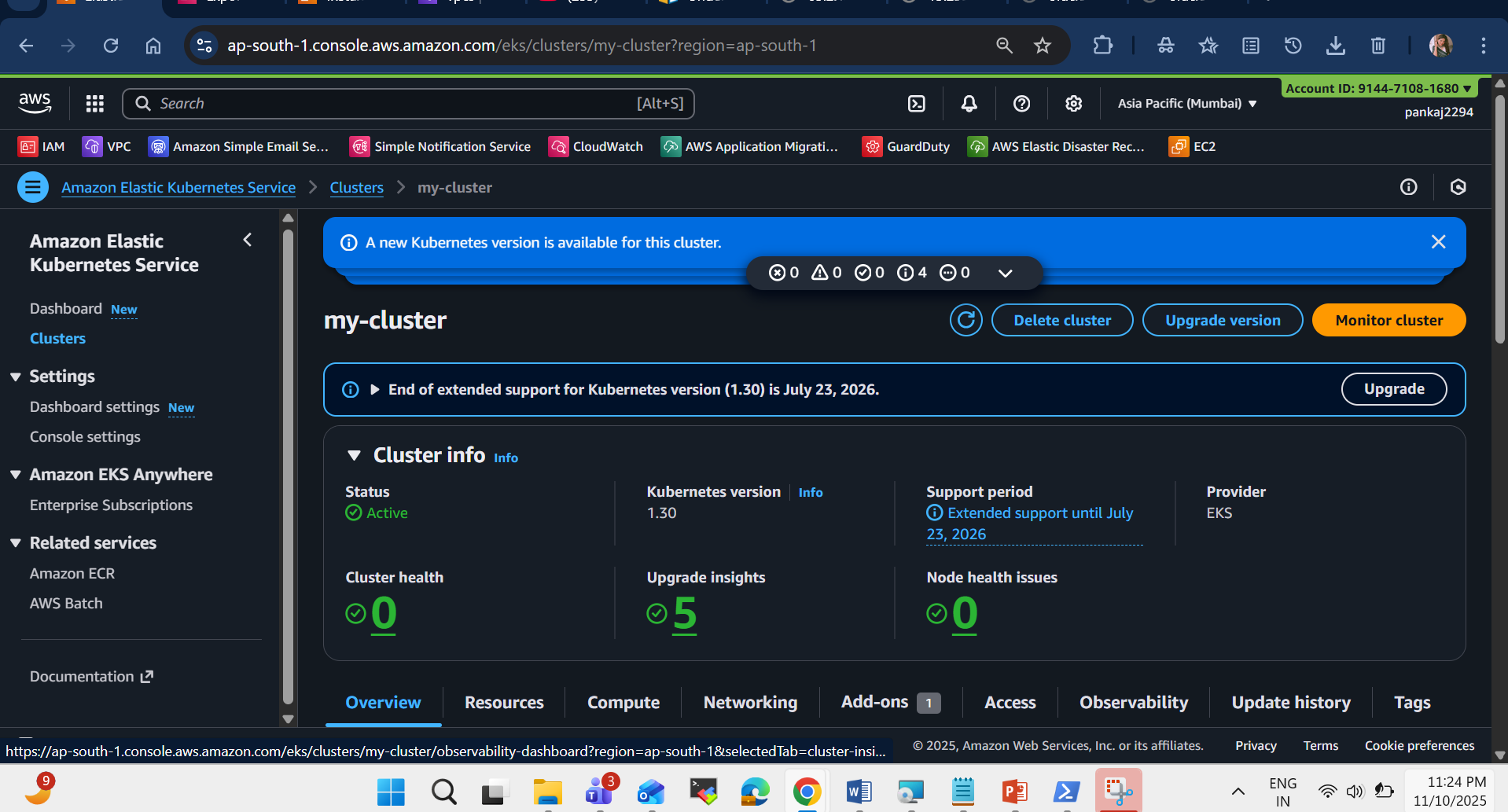
You’ll get something like:

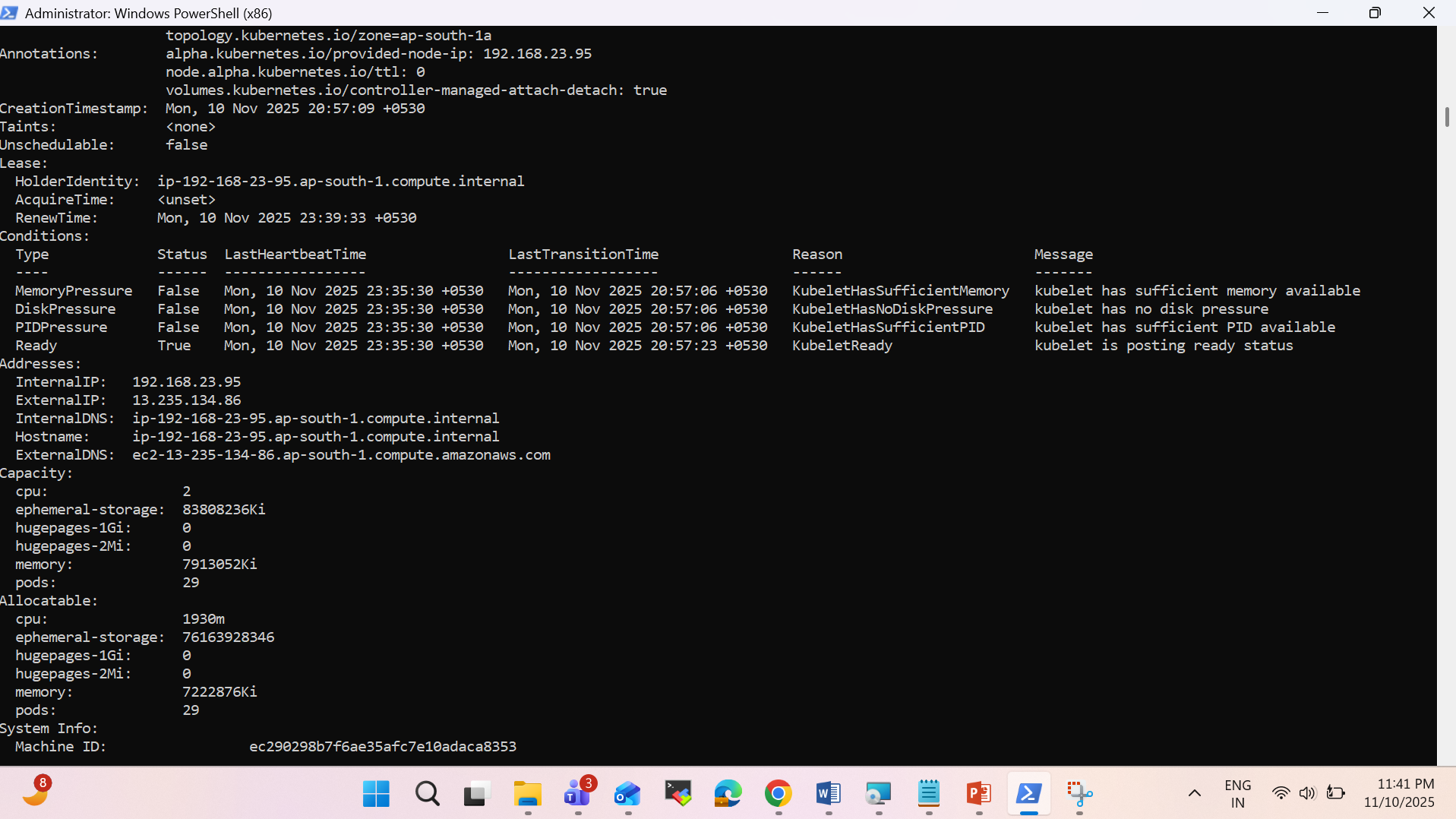
# nginx NodePort 10.100.x.x <none> 80:30262/TCP 3m

Now access it via:

# http://localhost:30262





**Step 6: Verify Resources** ****

**Check cluster and node details:**

* kubectl cluster-info
* kubectl top nodes
* kubectl describe nodes

