* **Multi-Cloud DR Project Using (AWS & AZURE Cloud )**

Google Drive Link :

<https://drive.google.com/drive/folders/1UG6wh9vMlngzJObVn-CCHhmcVBnrm0SW?usp=sharing>

GitHub Link :

<https://github.com/sagarpatilbox/Multi-Cloud-DR-UpGrad-Project>

* On your **local machine** Install
* Install:
  + Terraform
  + AWS CLI
  + Azure CLI

**1. Install Terraform on Windows**

**Step 1 — Download Terraform**

1. Go to: <https://developer.hashicorp.com/terraform/downloads>
2. Under **Windows**, download:  
   64-bit Terraform ZIP

**Step 2 — Extract Terraform**

1. Extract the ZIP file.
2. Move **terraform.exe** to a folder such as:

**C:\terraform\**

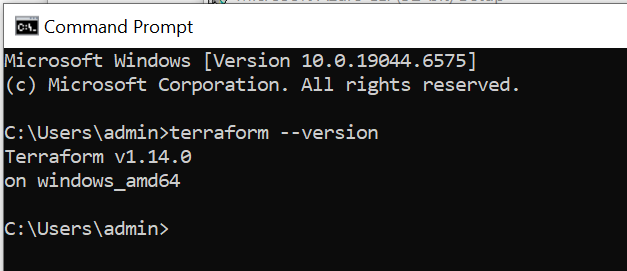
**Step 3 — Test installation**

* Open PowerShell and run:

**terraform -version**

* You should see output like:

**Terraform v1.x.x**



**2. Install AWS CLI on Windows**

**Step 1 — Download AWS CLI**

Download 64-bit Windows installer: <https://awscli.amazonaws.com/AWSCLIV2.msi>

**Step 2 — Run Installer**

1. Double-click **AWSCLIV2.msi**
2. Follow prompts → Finish

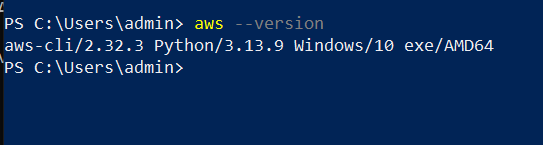
**Step 3 — Verify installation**

* Open PowerShell:

**aws --version**

* Expected output:

**aws-cli/2.x.x Python/3.x**



**3.** **Install Azure CLI on Windows**

### ****Step 1 — Download Azure CLI****

Go to: <https://aka.ms/installazurecliwindows>

This downloads **AzureCLI.msi**.

### ****Step 2 — Run installer****

Follow wizard → Install

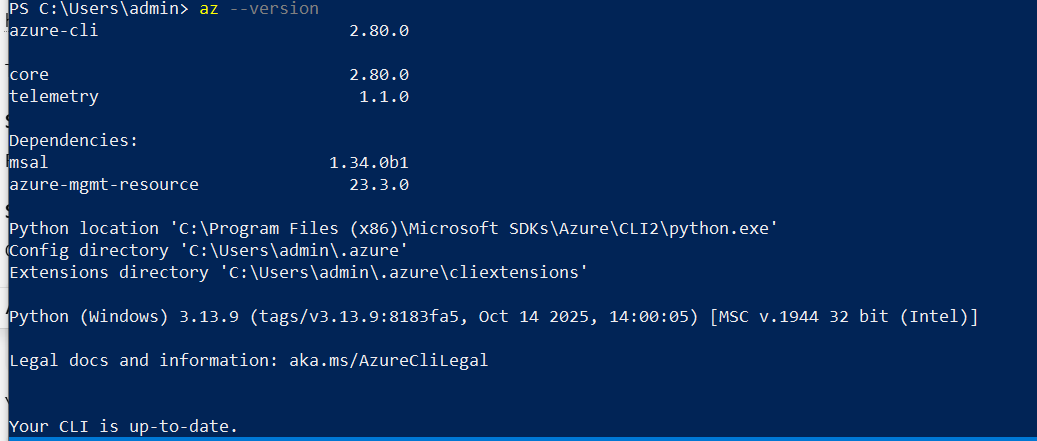
### ****Step 3 — Verify installation****

* Open PowerShell:

**az --version**

* You should see:

**azure-cli 2.x.x**



* On your **local machine** Install

**Using Git Bash (Recommended for Terraform & SSH)**

**Step 1 — Open Git Bash**

* Install Git for Windows: <https://git-scm.com/download/win>
* Then open **Git Bash**.

**Step 2 — Run the same SSH keygen command**

**ssh-keygen -t rsa -b 4096 -C "multi-cloud-dr" -f ~/.ssh/mcdr\_key**

**Your key files will be created at:**

**C:\Users\<YourUsername>\.ssh\mcdr\_key**

**C:\Users\<YourUsername>\.ssh\mcdr\_key.pub**

* mcdr\_key → private key
* mcdr\_key.pub → public key

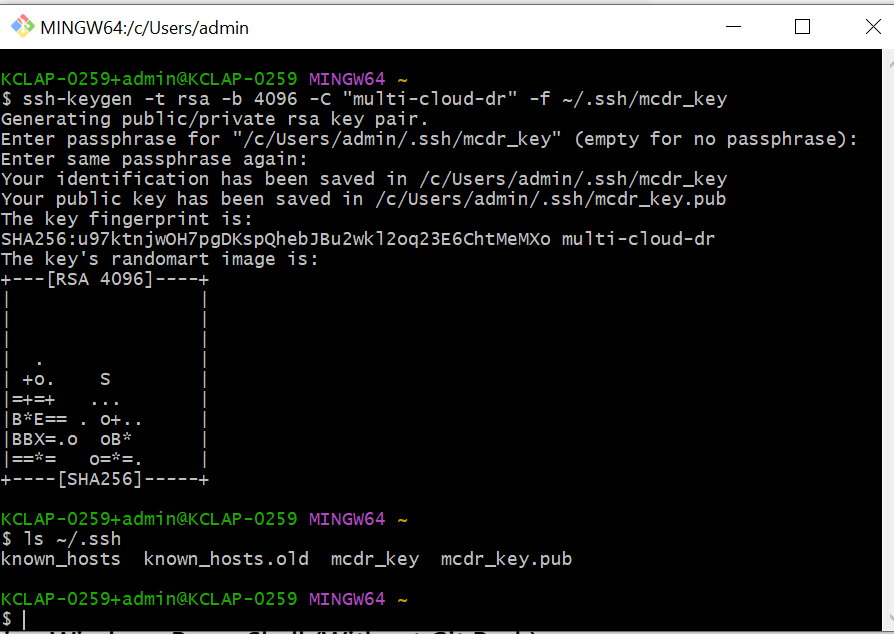
**Step 3 — Verify keys**

* In Git Bash:

ls ~/.ssh

* You should see:

**mcdr\_key mcdr\_key.pub**



**Task 1 – Infrastructure Provisioning (AWS & Azure)**

**1.1 AWS – VPC, Subnets, EC2 (App + Tools)**

**1.1.1 Create folder & files**

mkdir -p project/aws

cd project/aws

Create **variables.tf:**

variable "aws\_region" {

description = "AWS region"

type = string

default = "us-east-1"

}

variable "project\_name" {

description = "Project name prefix"

type = string

default = "multi-cloud-dr"

}

variable "public\_key\_path" {

description = "Path to your SSH public key"

type = string

}

Create **main.tf**:

terraform {

required\_version = ">= 1.3.0"

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

provider "aws" {

region = var.aws\_region

}

# VPC

resource "aws\_vpc" "main" {

cidr\_block = "10.0.0.0/16"

enable\_dns\_support = true

enable\_dns\_hostnames = true

tags = {

Name = "${var.project\_name}-vpc"

}

}

# Internet Gateway

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.main.id

tags = {

Name = "${var.project\_name}-igw"

}

}

# Public subnets

resource "aws\_subnet" "public\_1" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "${var.aws\_region}a"

map\_public\_ip\_on\_launch = true

tags = {

Name = "${var.project\_name}-public-1"

}

}

resource "aws\_subnet" "public\_2" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.2.0/24"

availability\_zone = "${var.aws\_region}b"

map\_public\_ip\_on\_launch = true

tags = {

Name = "${var.project\_name}-public-2"

}

}

# Private subnets

resource "aws\_subnet" "private\_1" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.11.0/24"

availability\_zone = "${var.aws\_region}a"

tags = {

Name = "${var.project\_name}-private-1"

}

}

resource "aws\_subnet" "private\_2" {

vpc\_id = aws\_vpc.main.id

cidr\_block = "10.0.12.0/24"

availability\_zone = "${var.aws\_region}b"

tags = {

Name = "${var.project\_name}-private-2"

}

}

# NAT

resource "aws\_eip" "nat\_eip" {

domain = "vpc"

tags = { Name = "${var.project\_name}-nat-eip" }

}

resource "aws\_nat\_gateway" "nat" {

allocation\_id = aws\_eip.nat\_eip.id

subnet\_id = aws\_subnet.public\_1.id

depends\_on = [aws\_internet\_gateway.igw]

tags = {

Name = "${var.project\_name}-nat"

}

}

# Public route table

resource "aws\_route\_table" "public" {

vpc\_id = aws\_vpc.main.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

tags = {

Name = "${var.project\_name}-public-rt"

}

}

resource "aws\_route\_table\_association" "public\_1" {

subnet\_id = aws\_subnet.public\_1.id

route\_table\_id = aws\_route\_table.public.id

}

resource "aws\_route\_table\_association" "public\_2" {

subnet\_id = aws\_subnet.public\_2.id

route\_table\_id = aws\_route\_table.public.id

}

# Private route table

resource "aws\_route\_table" "private" {

vpc\_id = aws\_vpc.main.id

route {

cidr\_block = "0.0.0.0/0"

nat\_gateway\_id = aws\_nat\_gateway.nat.id

}

tags = {

Name = "${var.project\_name}-private-rt"

}

}

resource "aws\_route\_table\_association" "private\_1" {

subnet\_id = aws\_subnet.private\_1.id

route\_table\_id = aws\_route\_table.private.id

}

resource "aws\_route\_table\_association" "private\_2" {

subnet\_id = aws\_subnet.private\_2.id

route\_table\_id = aws\_route\_table.private.id

}

# Security group

resource "aws\_security\_group" "web\_sg" {

name = "${var.project\_name}-web-sg"

description = "Allow SSH and HTTP"

vpc\_id = aws\_vpc.main.id

ingress {

description = "SSH"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

description = "HTTP"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "${var.project\_name}-web-sg"

}

}

# Key pair using your public key

resource "aws\_key\_pair" "default" {

key\_name = "${var.project\_name}-key"

public\_key = file(var.public\_key\_path)

}

# App instance

resource "aws\_instance" "app" {

ami = "ami-0ecb62995f68bb549"

instance\_type = "t3.micro"

subnet\_id = aws\_subnet.public\_1.id

vpc\_security\_group\_ids = [aws\_security\_group.web\_sg.id]

key\_name = aws\_key\_pair.default.key\_name

associate\_public\_ip\_address = true

tags = {

Name = "${var.project\_name}-app"

Role = "app"

}

}

# Tools instance

resource "aws\_instance" "tools" {

ami = "ami-0ecb62995f68bb549"

instance\_type = "t3.micro"

subnet\_id = aws\_subnet.public\_2.id

vpc\_security\_group\_ids = [aws\_security\_group.web\_sg.id]

key\_name = aws\_key\_pair.default.key\_name

associate\_public\_ip\_address = true

tags = {

Name = "${var.project\_name}-tools"

Role = "tools"

}

}

Create **outputs.tf**:

output "app\_public\_ip" {

value = aws\_instance.app.public\_ip

description = "Public IP of the App Machine"

}

output "tools\_public\_ip" {

value = aws\_instance.tools.public\_ip

description = "Public IP of the Tools Machine"

}

**1.1.2 Configure & Run Terraform**

**First Configure aws Login :**

aws configure

**Give Below Information :**

AWS Access Key ID: <from CSV>

AWS Secret Access Key: <from CSV>

Default region name: us-east-1

Default output format: json

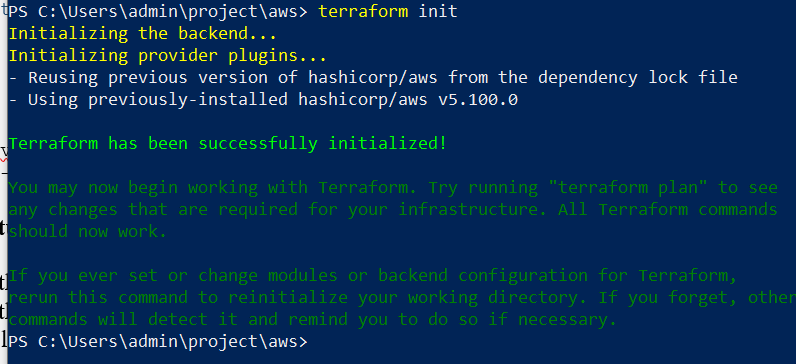
**Run Terraform**

cd project/aws

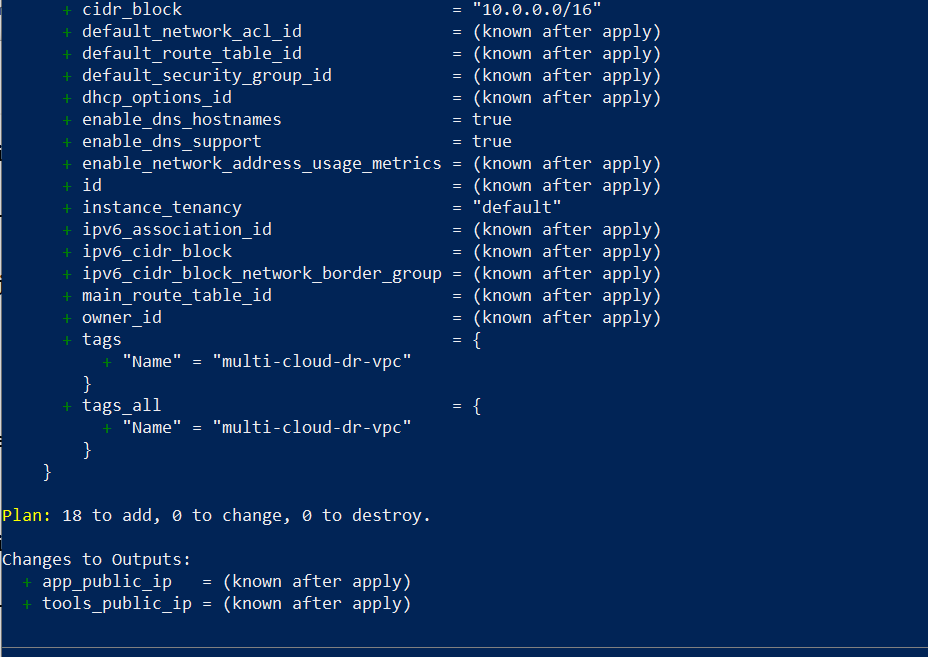
terraform init

terraform plan -var="public\_key\_path=$HOME/.ssh/mcdr\_key.pub"

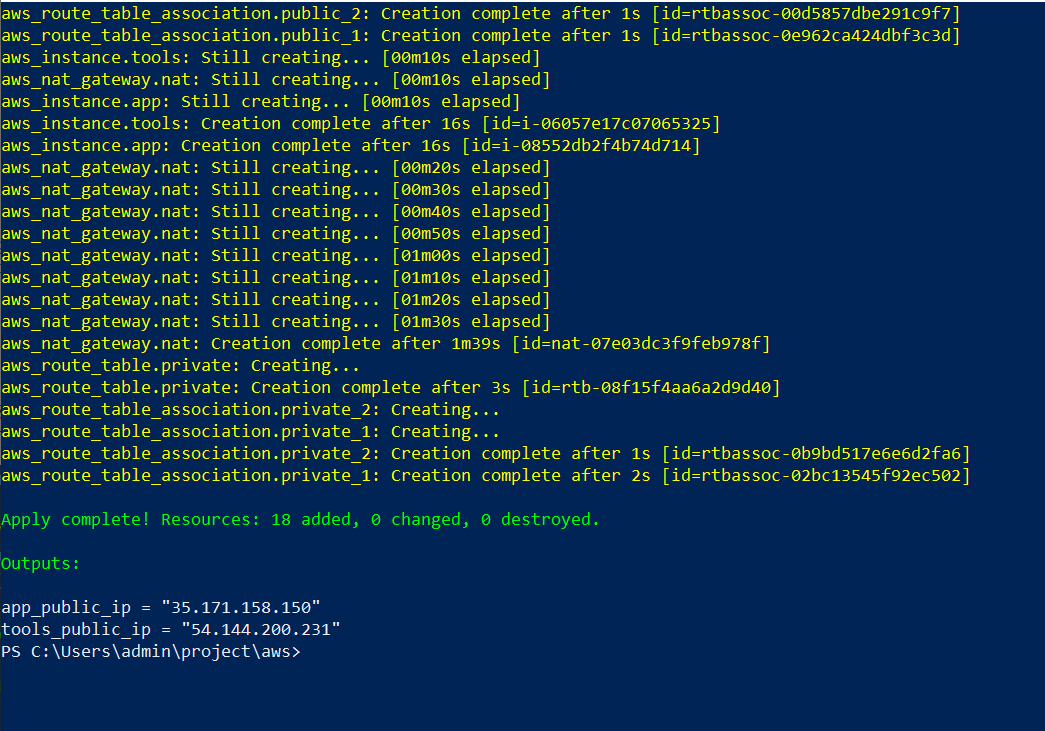
terraform apply -var="public\_key\_path=$HOME/.ssh/mcdr\_key.pub"



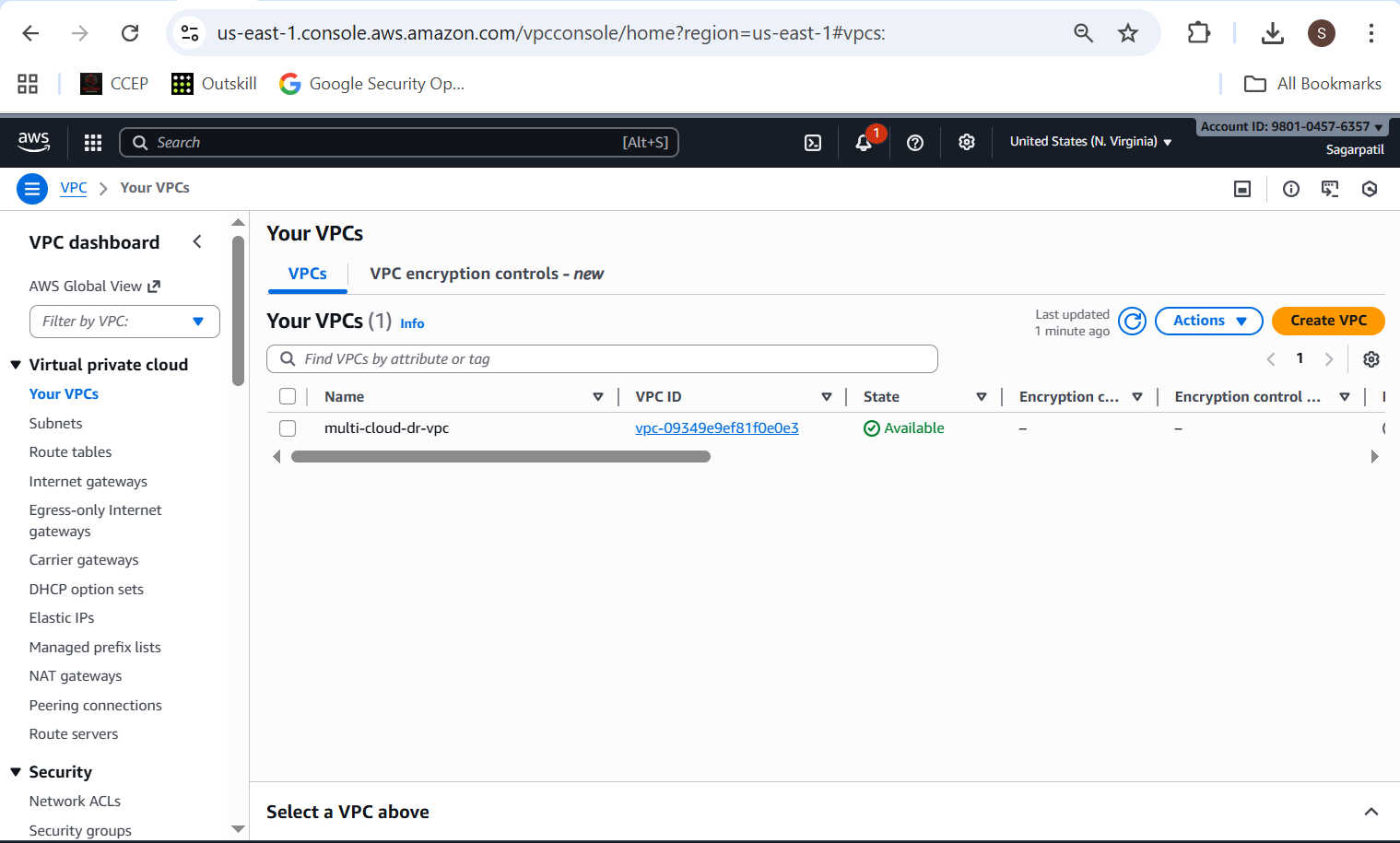
* **Screenshots to capture:**
* Terminal with **terraform plan** summary (AWS).



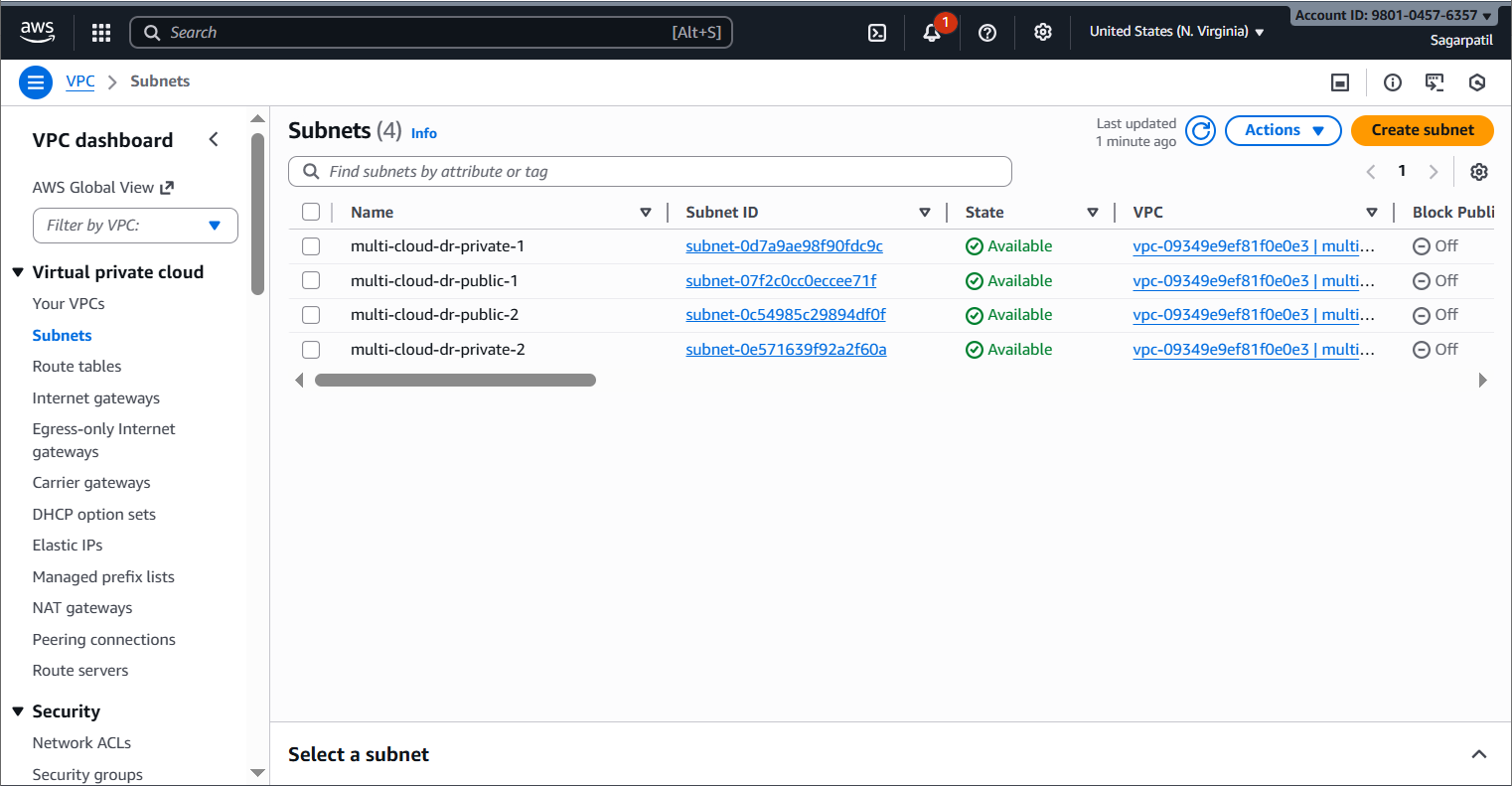
* Terminal with **terraform apply** success.



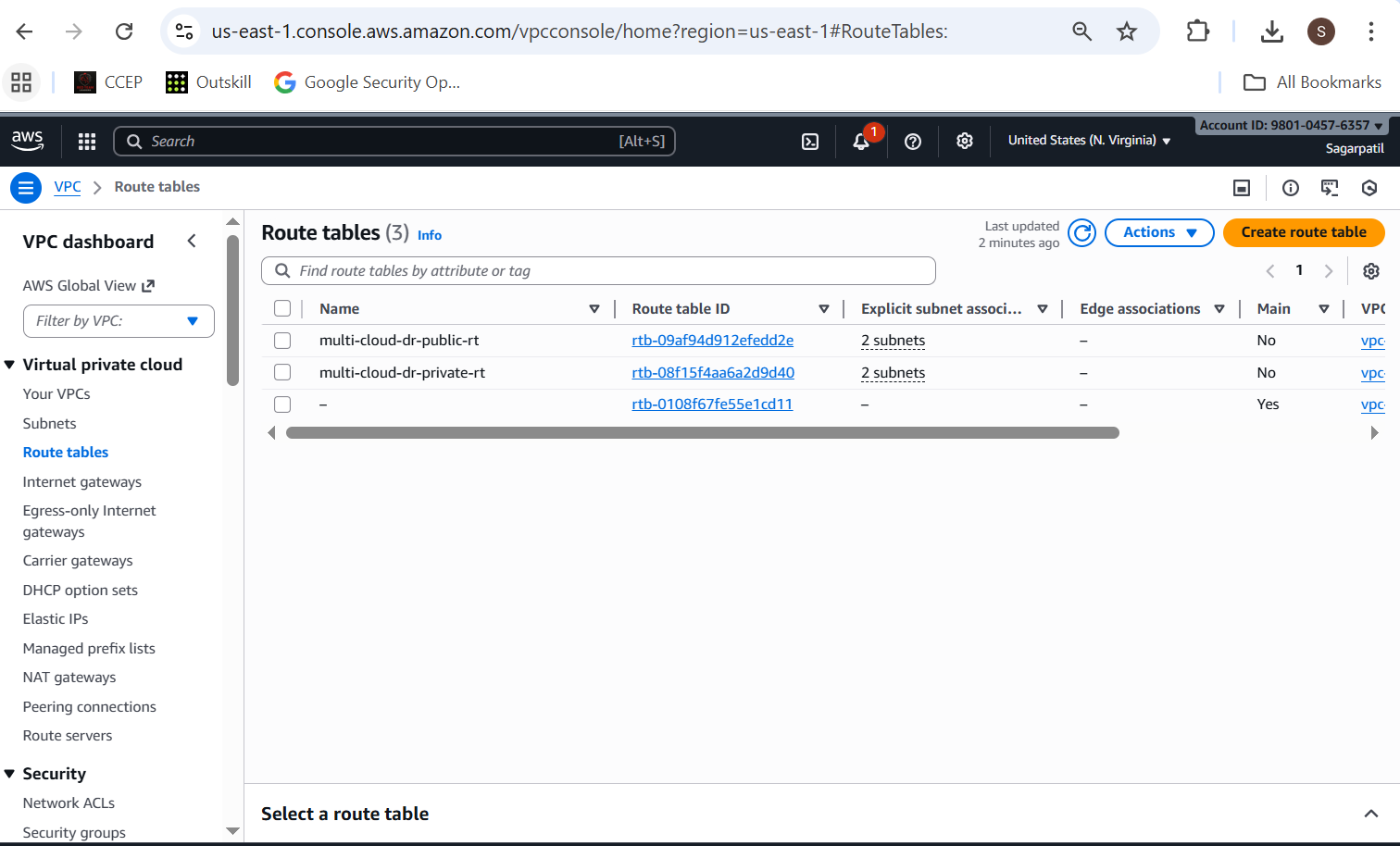
* AWS Console:
  + VPC details



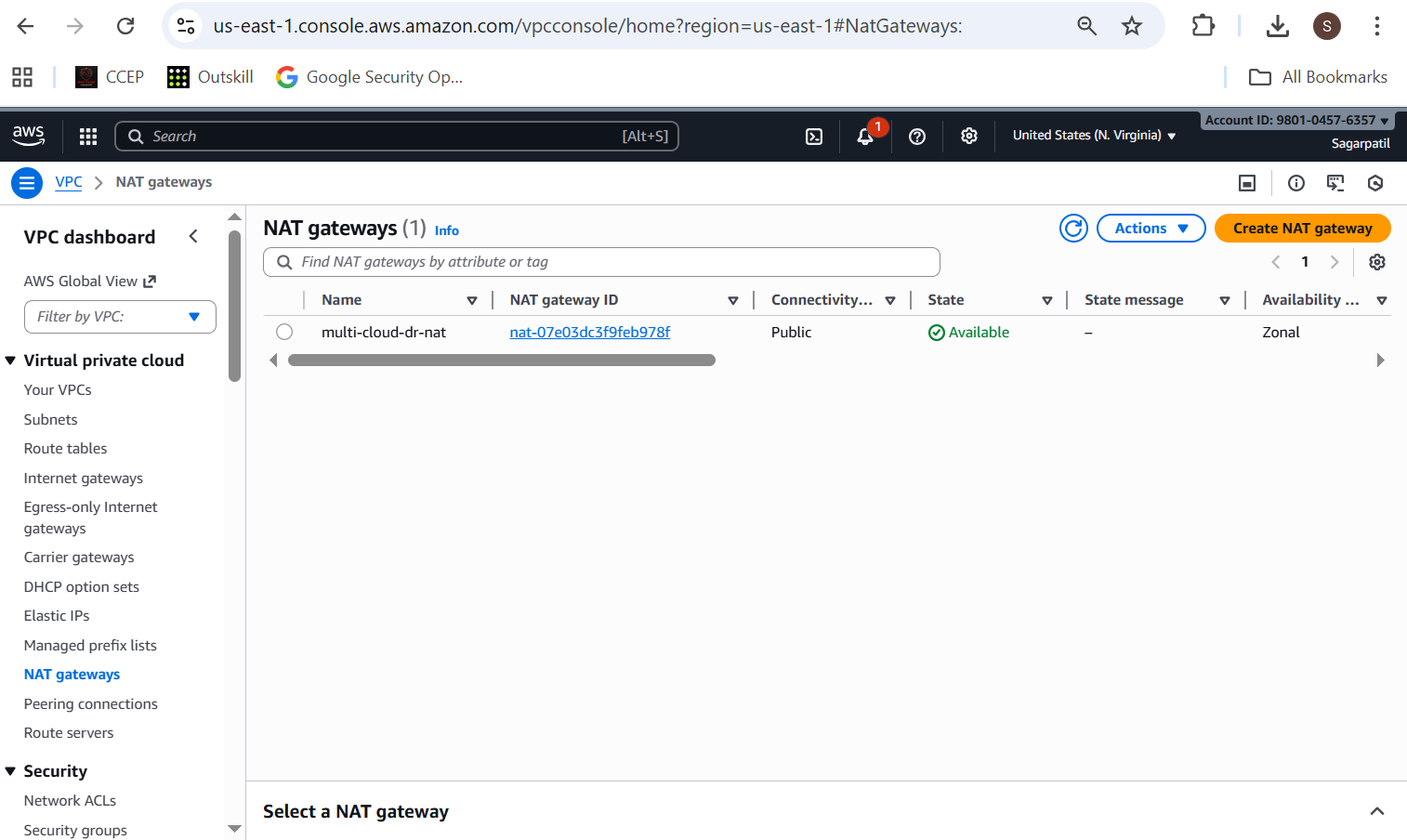
* + Subnets listing (2 public, 2 private)



* + Route tables (public + private)



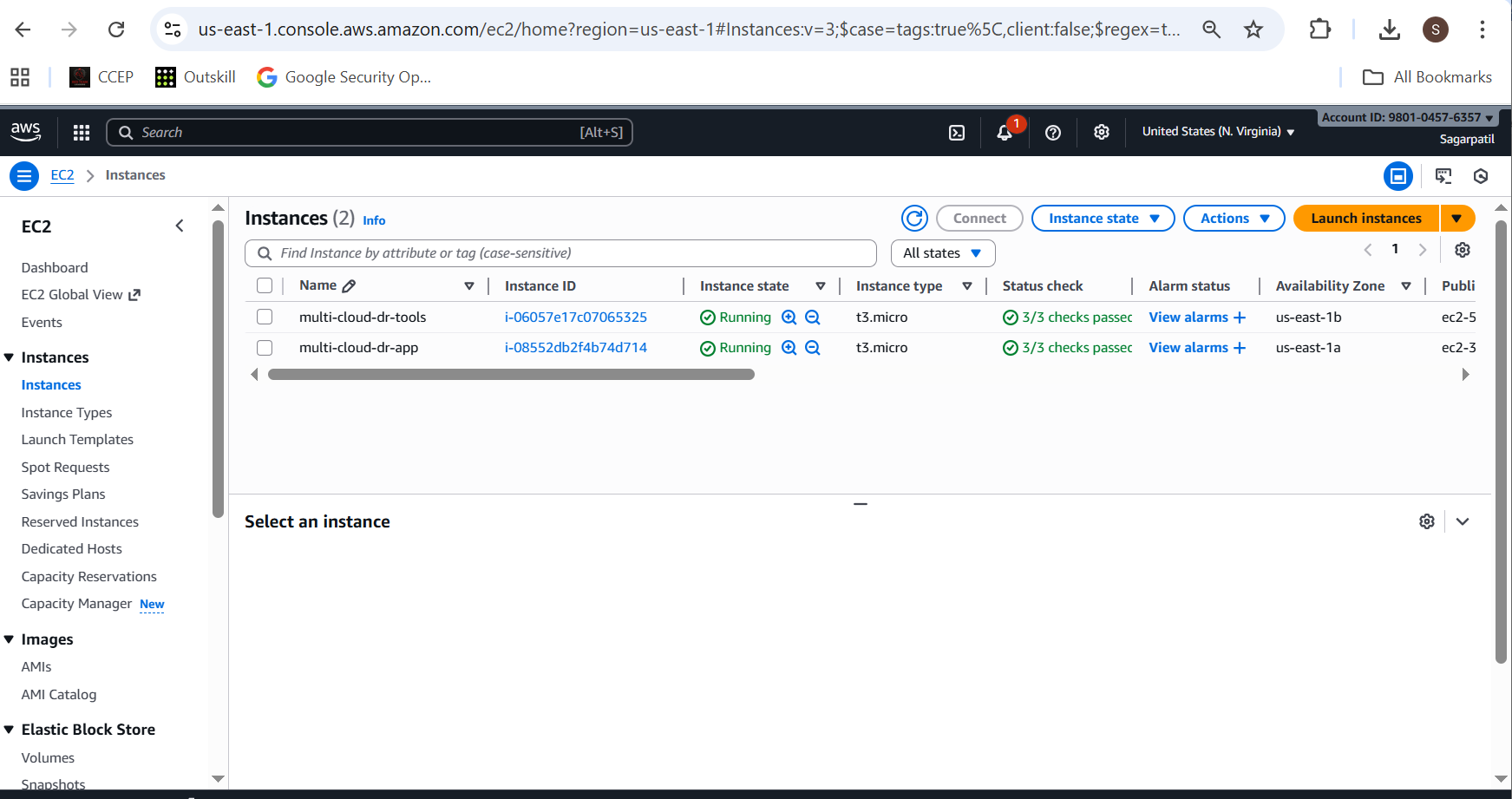
* + NAT gateway



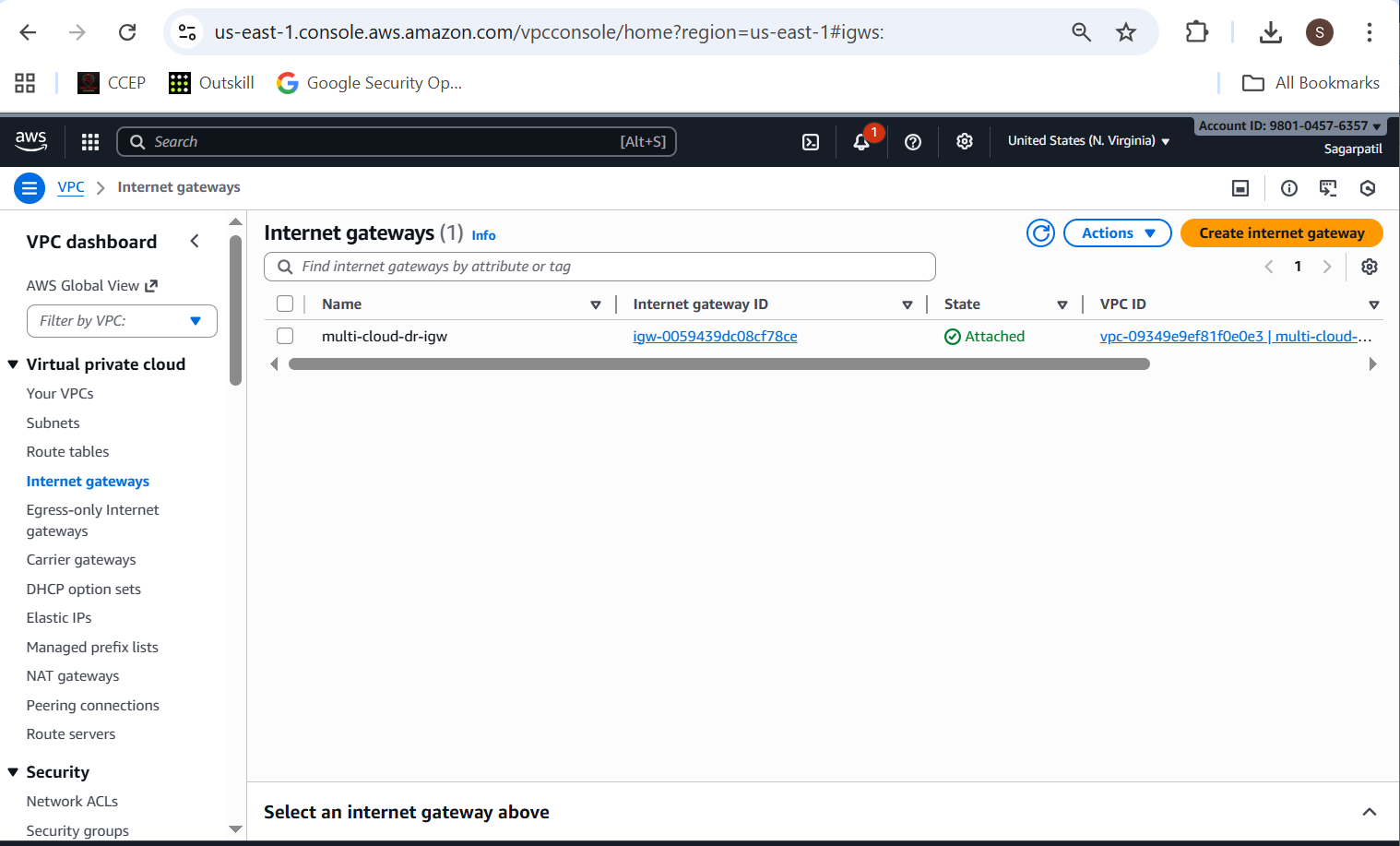
* + Security group inbound rules (22, 80)



* + EC2 instances list showing App + Tools machines



* + Internet Gateway



**1.1.3 SSH test to EC2s**

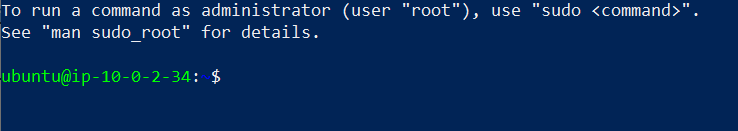
From local:

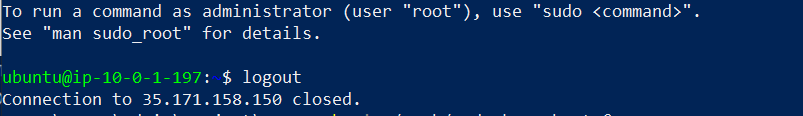
**ssh -i ~/.ssh/mcdr\_key ubuntu@<AWS\_APP\_PUBLIC\_IP>**

**ssh -i ~/.ssh/mcdr\_key ubuntu@<AWS\_TOOLS\_PUBLIC\_IP>**

**Screenshot:**

* Terminal showing successful SSH into both EC2s.





**1.2 Azure – VNet, Subnet, NSG, VM**

**1.2.1 Create folder & files**

cd project

mkdir -p azure

cd azure

**variables.tf**:

variable "azure\_location" {

description = "Azure location"

type = string

default = "eastus"

}

variable "project\_name" {

description = "Project name"

type = string

default = "multi-cloud-dr"

}

variable "admin\_username" {

description = "Admin username for VM"

type = string

default = "azureuser"

}

variable "public\_key\_path" {

description = "Path to SSH public key"

type = string

}

**main.tf**:

terraform {

required\_version = ">= 1.3.0"

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "~> 4.0"

}

}

}

provider "azurerm" {

features {}

}

resource "azurerm\_resource\_group" "rg" {

name = "${var.project\_name}-rg"

location = var.azure\_location

}

resource "azurerm\_virtual\_network" "vnet" {

name = "${var.project\_name}-vnet"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

address\_space = ["10.10.0.0/16"]

}

resource "azurerm\_subnet" "subnet\_app" {

name = "${var.project\_name}-subnet-app"

resource\_group\_name = azurerm\_resource\_group.rg.name

virtual\_network\_name = azurerm\_virtual\_network.vnet.name

address\_prefixes = ["10.10.1.0/24"]

}

resource "azurerm\_network\_security\_group" "nsg" {

name = "${var.project\_name}-nsg"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

security\_rule {

name = "SSH"

priority = 100

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "22"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

security\_rule {

name = "HTTP"

priority = 110

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "80"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

}

resource "azurerm\_subnet\_network\_security\_group\_association" "subnet\_nsg\_assoc" {

subnet\_id = azurerm\_subnet.subnet\_app.id

network\_security\_group\_id = azurerm\_network\_security\_group.nsg.id

}

resource "azurerm\_public\_ip" "app\_pip" {

name = "${var.project\_name}-app-pip"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

allocation\_method = "Static"

sku = "Basic"

}

resource "azurerm\_network\_interface" "app\_nic" {

name = "${var.project\_name}-app-nic"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.subnet\_app.id

private\_ip\_address\_allocation = "Dynamic"

public\_ip\_address\_id = azurerm\_public\_ip.app\_pip.id

}

}

resource "azurerm\_linux\_virtual\_machine" "app\_vm" {

name = "${var.project\_name}-app-vm"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

size = "Standard\_B1s"

admin\_username = var.admin\_username

disable\_password\_authentication = true

network\_interface\_ids = [azurerm\_network\_interface.app\_nic.id]

admin\_ssh\_key {

username = var.admin\_username

public\_key = file(var.public\_key\_path)

}

os\_disk {

name = "${var.project\_name}-app-osdisk"

caching = "ReadWrite"

storage\_account\_type = "Standard\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "0001-com-ubuntu-server-jammy"

sku = "22\_04-lts"

version = "latest"

}

computer\_name = "appvm"

tags = {

Role = "app"

}

}

**outputs.tf**:

output "app\_vm\_public\_ip" {

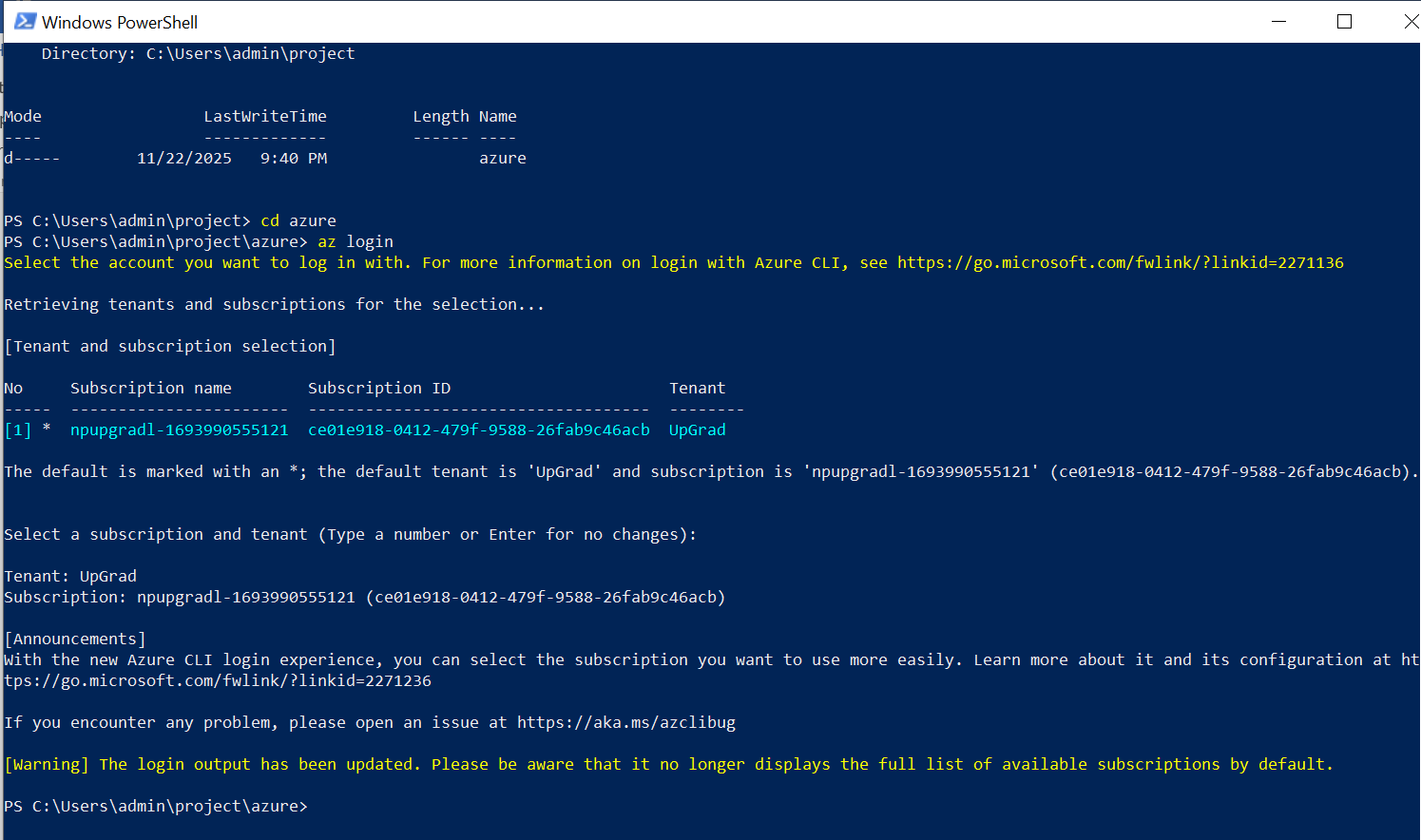
value = azurerm\_public\_ip.app\_pip.ip\_address

description = "Public IP of Azure App VM"

}

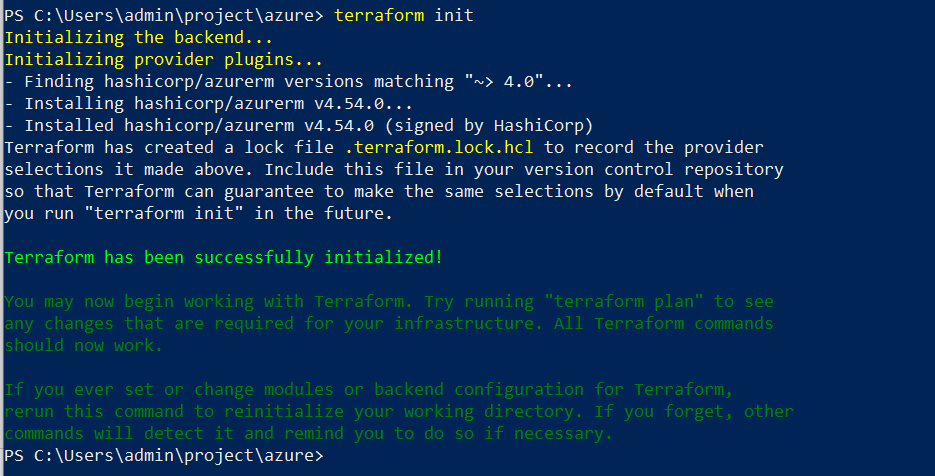
**1.2.2 Run Terraform**

**az login** # if not already logged in



cd project/azure

terraform init

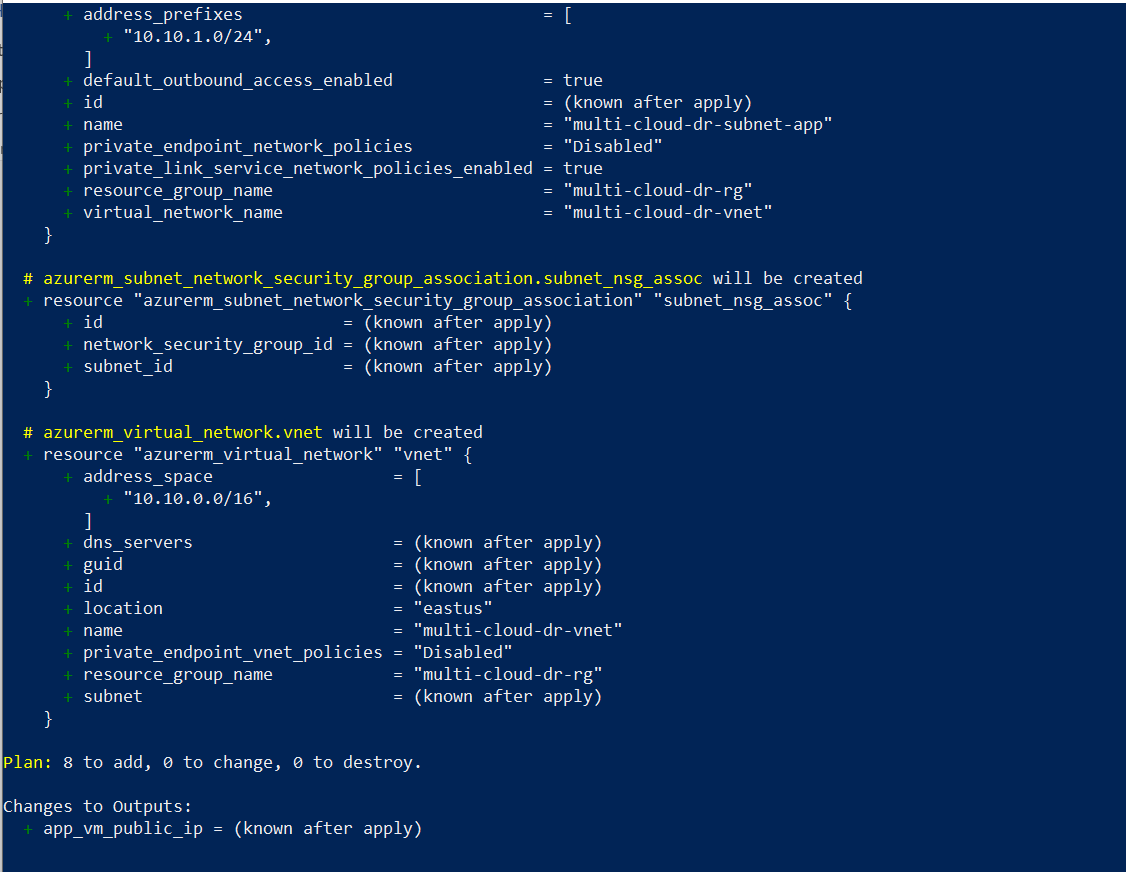


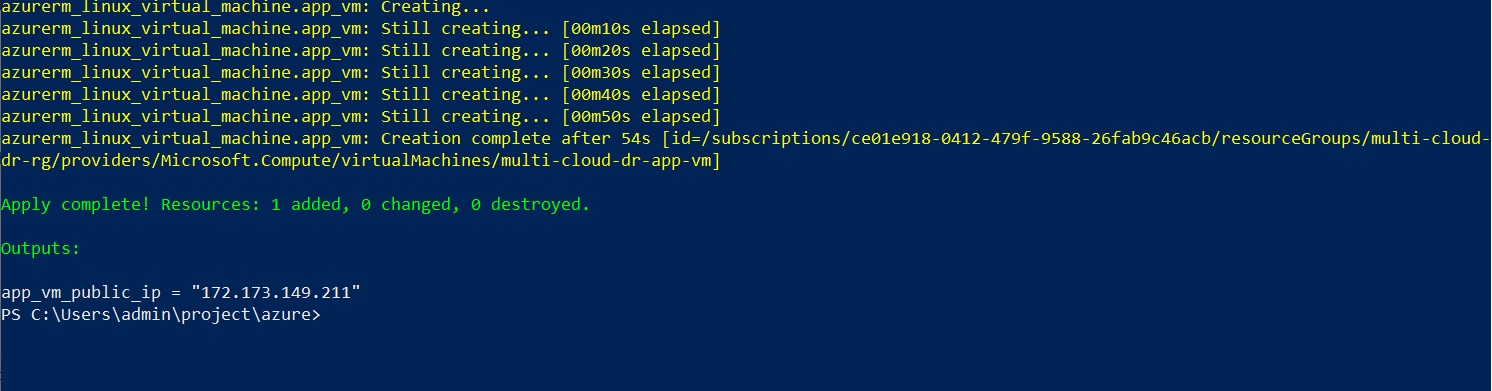
terraform plan -var="public\_key\_path=$HOME/.ssh/mcdr\_key.pub"

terraform apply -var="public\_key\_path=$HOME/.ssh/mcdr\_key.pub"

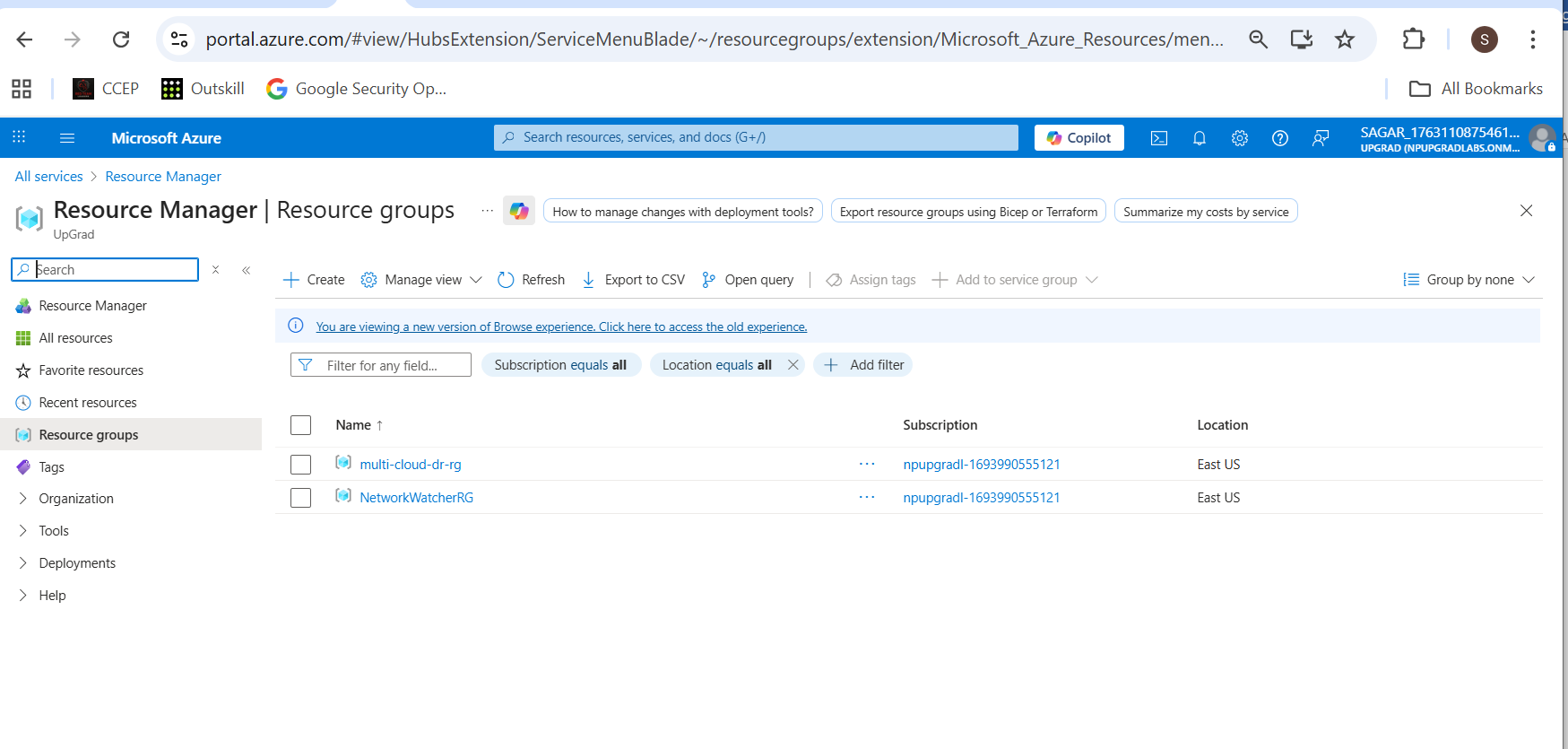
**Screenshots:**

* Terminal with plan and apply (Azure).

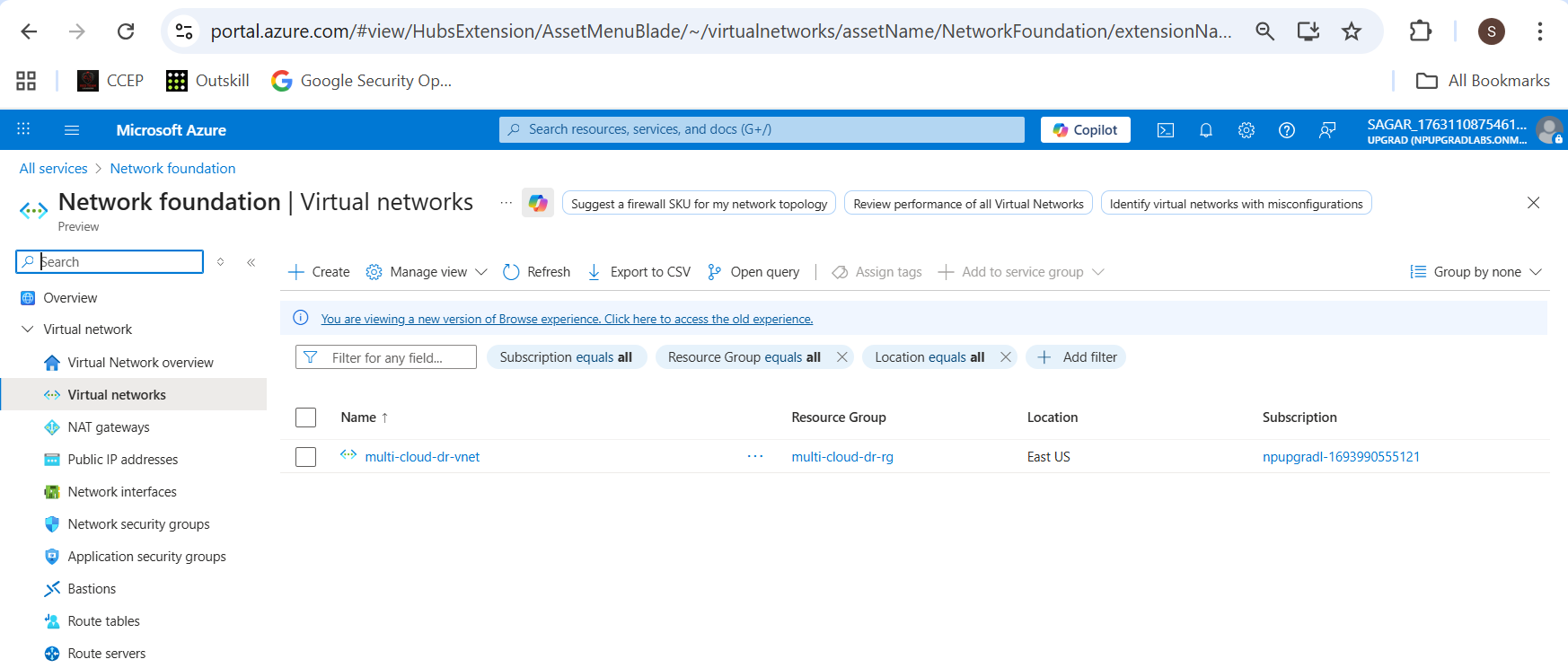




* Azure portal:
  + Resource group



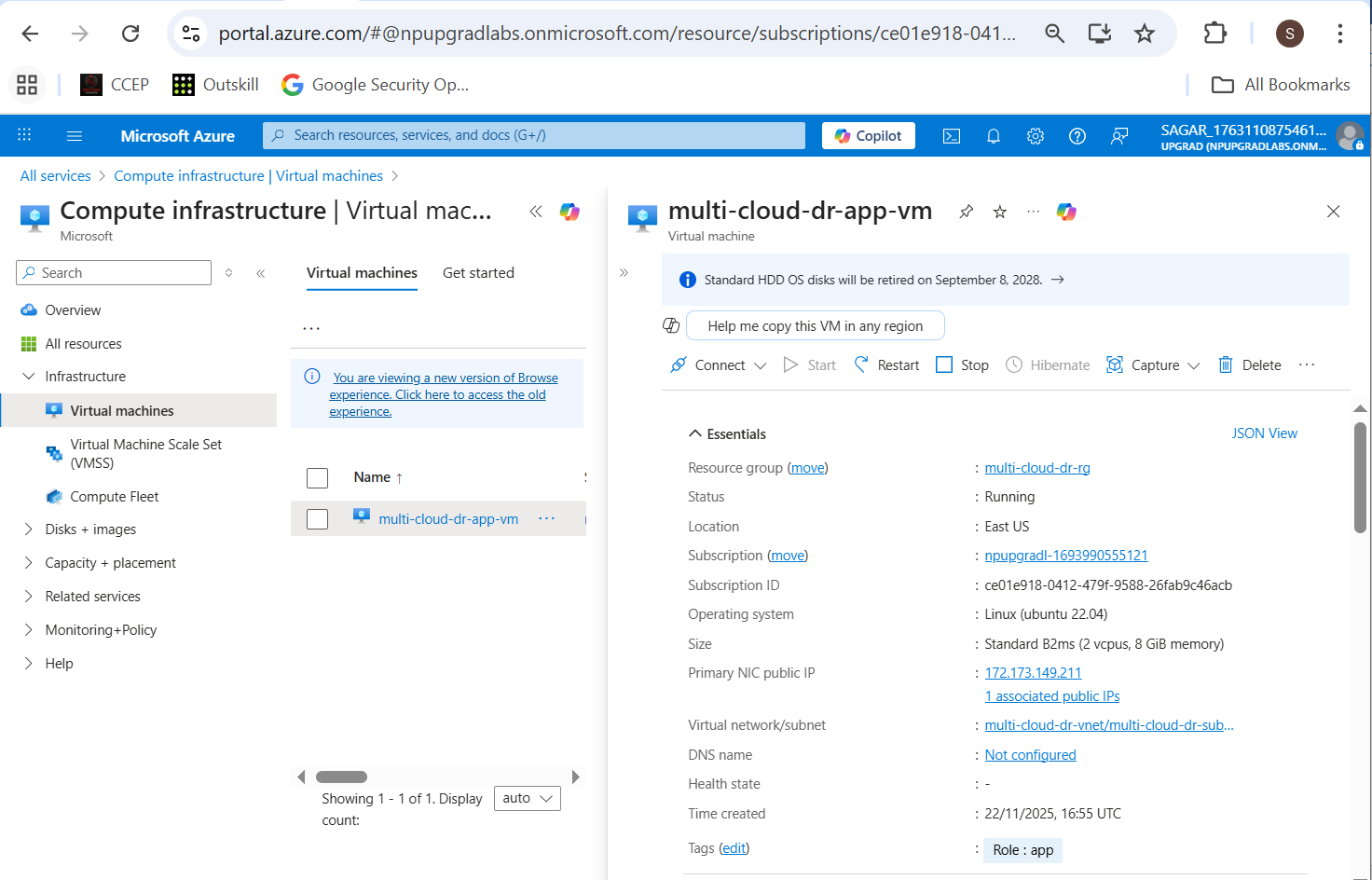
* + VNet + subnet



* + NSG inbound rules (22, 80)



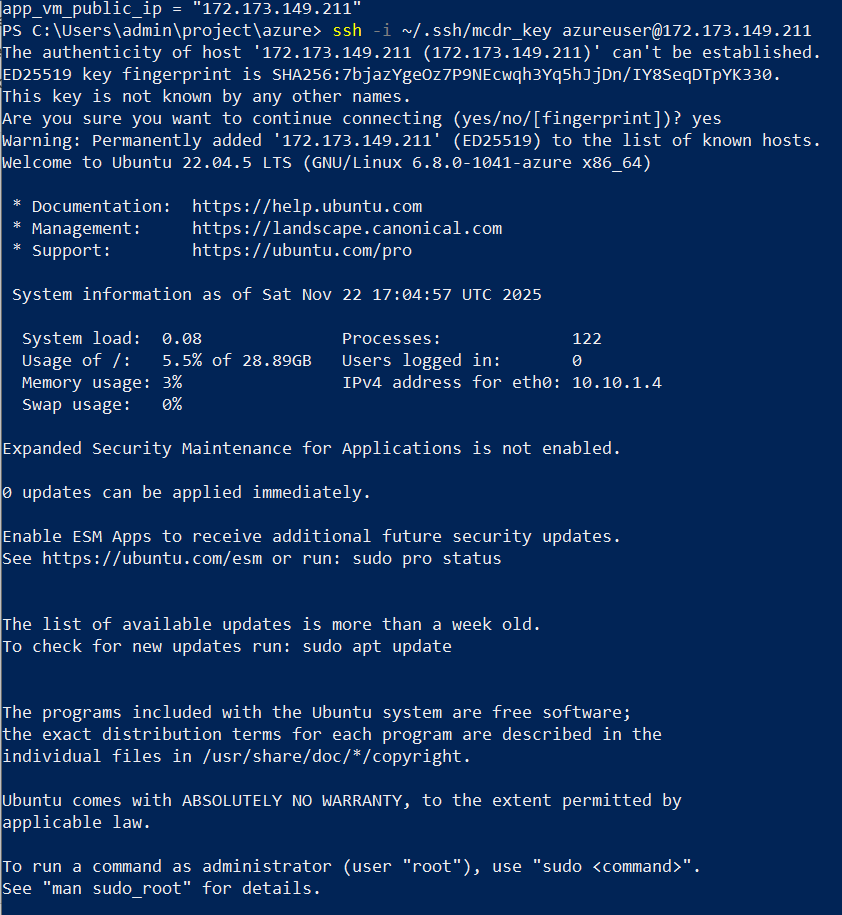
* + VM overview (running, public IP).



**1.2.3 SSH test to Azure VM**

**ssh -i ~/.ssh/mcdr\_key azureuser@<AZURE\_APP\_PUBLIC\_IP>**

**Screenshot:**  
 Terminal showing successful SSH login.



**Task 2 – Configuration Management (Ansible)**

All Ansible work happens on the **AWS Tools Machine**.

**2.1 Install Ansible on Tools Machine**

* SSH to Tools Machine:

**ssh -i ~/.ssh/mcdr\_key ubuntu@<AWS\_TOOLS\_PUBLIC\_IP>**

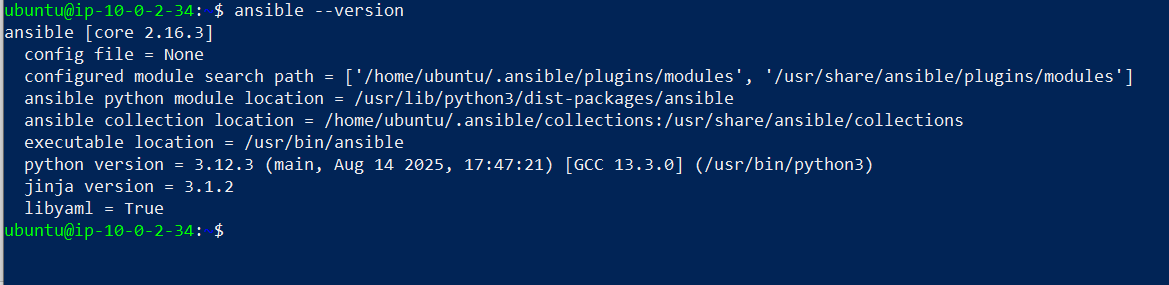
* Install Ansible:

**sudo apt update**

**sudo apt install -y ansible git**

**ansible --version**

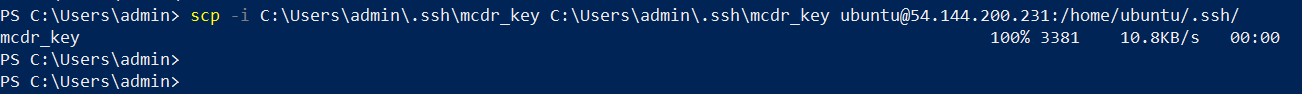
**Screenshot:**  
**ansible --version** output.



**2.2 Copy SSH key to Tools Machine**

On **local machine**:

scp -i C:\Users\admin\.ssh\mcdr\_key C:\Users\admin\.ssh\mcdr\_key [ubuntu@<AWS\_TOOLS\_PUBLIC\_IP>:/home/ubuntu/.ssh/](mailto:ubuntu@%3cAWS_TOOLS_PUBLIC_IP%3e:/home/ubuntu/.ssh/)



On **Tools Machine**:

chmod 600 ~/.ssh/mcdr\_key

**2.3 Create Ansible inventory**

On Tools Machine:

mkdir -p ~/multi-cloud-ansible

cd ~/multi-cloud-ansible

vim inventory.ini

**inventory.ini**:

[aws\_app]

aws\_app ansible\_host=<AWS\_APP\_PUBLIC\_IP> ansible\_user=ubuntu ansible\_ssh\_private\_key\_file=/home/ubuntu/.ssh/mcdr\_key

[azure\_app]

azure\_app ansible\_host=<AZURE\_APP\_PUBLIC\_IP> ansible\_user=azureuser ansible\_ssh\_private\_key\_file=/home/ubuntu/.ssh/mcdr\_key

[all\_app\_servers:children]

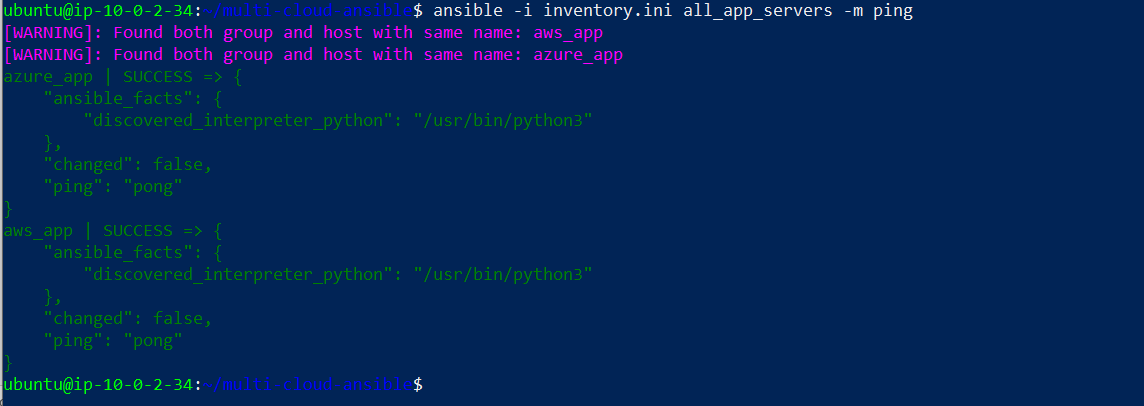
aws\_app

azure\_app

Test connectivity:

ansible -i inventory.ini all\_app\_servers -m ping

**Screenshot:**  
Ping output showing pong from both hosts.



**2.4 Ansible playbook to install & start Nginx**

Create **nginx\_setup.yml**:

---

- name: Install and configure Nginx on AWS and Azure app servers

hosts: all\_app\_servers

become: yes

vars:

nginx\_pkg\_name: nginx

tasks:

- name: Update APT cache

ansible.builtin.apt:

update\_cache: yes

cache\_valid\_time: 3600

- name: Install Nginx

ansible.builtin.apt:

name: "{{ nginx\_pkg\_name }}"

state: present

- name: Ensure Nginx is enabled and started

ansible.builtin.service:

name: nginx

state: started

enabled: yes

- name: Check Nginx default page

ansible.builtin.command: curl -s http://localhost

register: nginx\_page

changed\_when: false

- name: Show first line of Nginx page

ansible.builtin.debug:

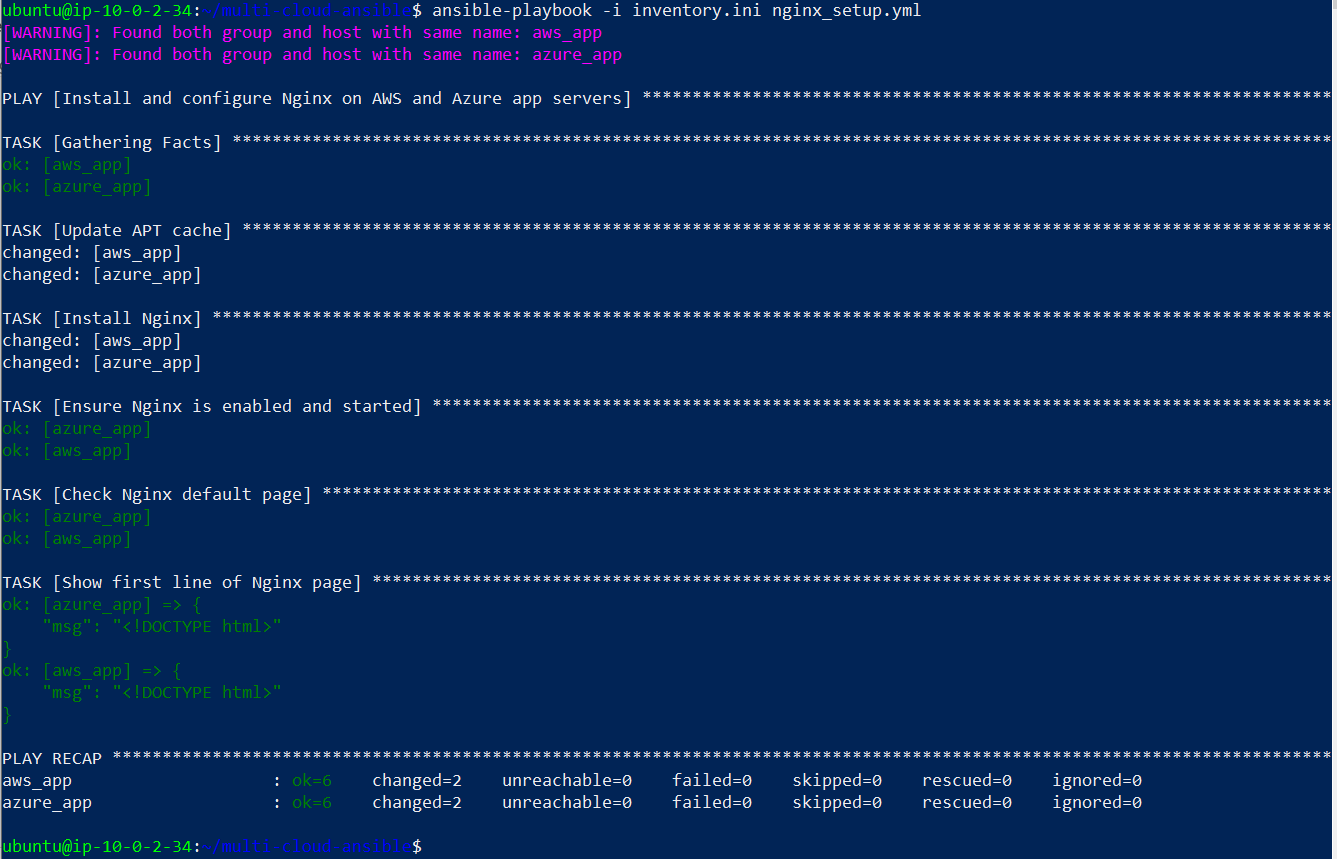
msg: "{{ nginx\_page.stdout\_lines | default([]) | first | default('no output') }}"

Run it:

ansible-playbook -i inventory.ini nginx\_setup.yml

**Screenshots:**

* Playbook run showing success.



On **each app** server:

**# On AWS app**

systemctl status nginx

ls -l /var/www/html/

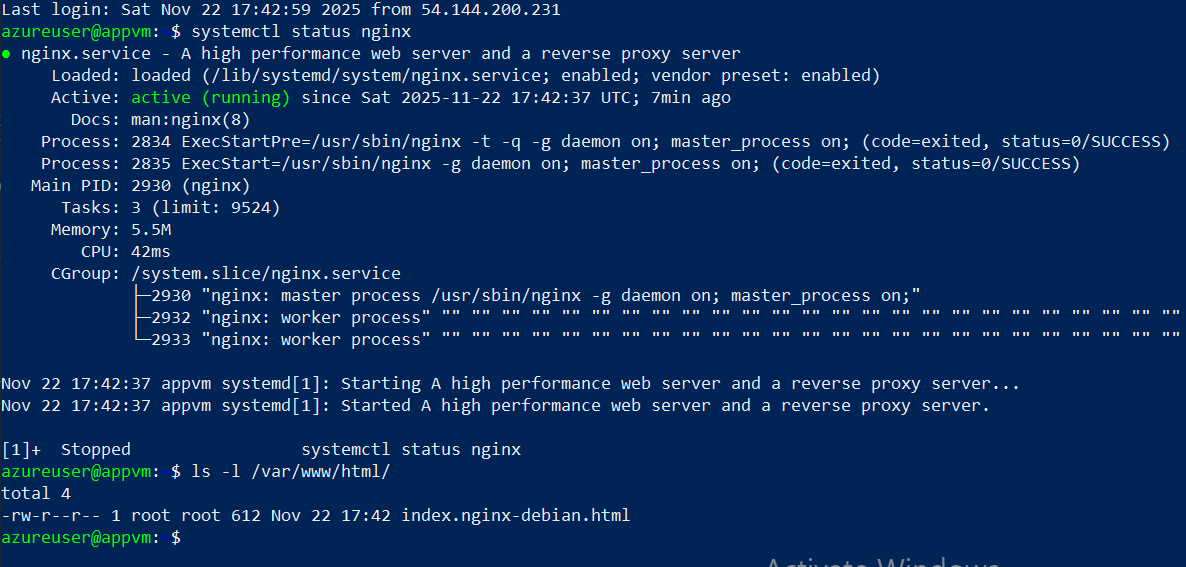
**# On Azure app**

systemctl status nginx

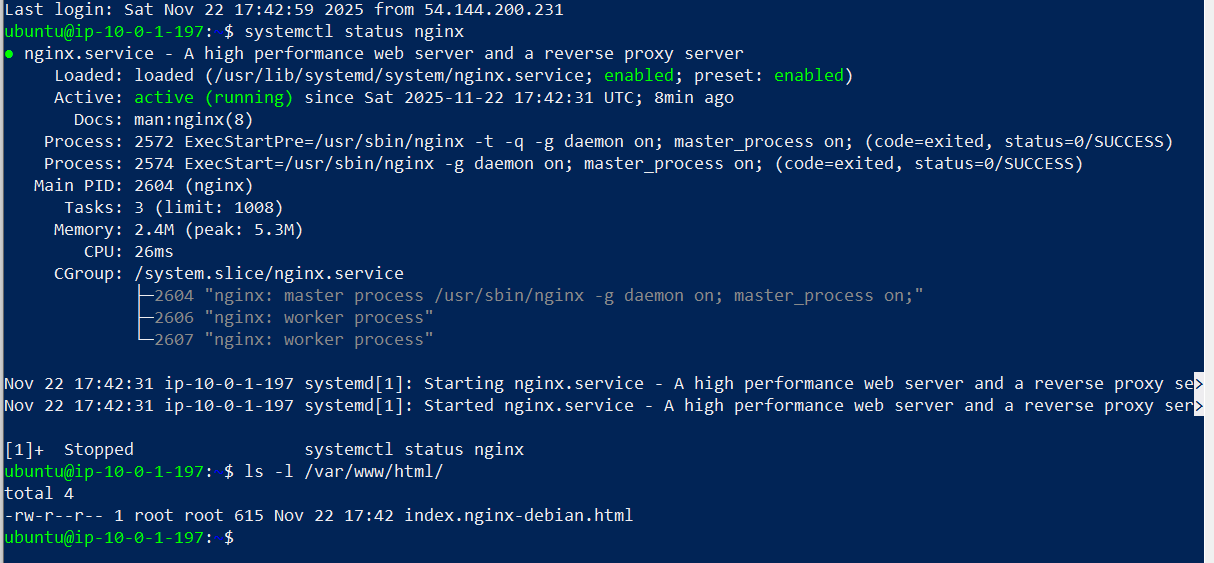
ls -l /var/www/html/

**Screenshots:**

* systemctl status nginx on AWS & Azure = active.



* /var/www/html listing.



**Task 3 – Application Deployment (Custom Nginx Pages)**

**3.1 Create custom html files on Tools Machine**

From ~/multi-cloud-ansible:

cat > index-aws.html << 'EOF'

<!doctype html>

<html>

<head><title>Welcome to AWS</title></head>

<body>

<h1>Welcome to AWS</h1>

<p>This page is served from the AWS App Machine.</p>

</body>

</html>

EOF

cat > index-azure.html << 'EOF'

<!doctype html>

<html>

<head><title>Welcome to Azure</title></head>

<body>

<h1>Welcome to Azure</h1>

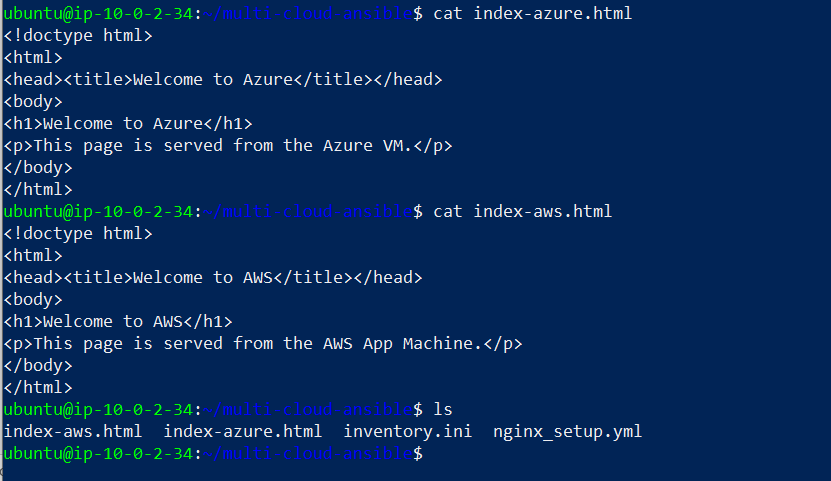
<p>This page is served from the Azure VM.</p>

</body>

</html>

EOF

**Screenshot:**  
ls showing **index-aws.html** and **index-azure.html**.



**3.2 Add cloud flag to inventory**

Edit **inventory.ini**:

[aws\_app]

aws\_app ansible\_host=<AWS\_APP\_PUBLIC\_IP> ansible\_user=ubuntu ansible\_ssh\_private\_key\_file=/home/ubuntu/.ssh/mcdr\_key cloud=aws

[azure\_app]

azure\_app ansible\_host=<AZURE\_APP\_PUBLIC\_IP> ansible\_user=azureuser ansible\_ssh\_private\_key\_file=/home/ubuntu/.ssh/mcdr\_key cloud=azure

[all\_app\_servers:children]

aws\_app

azure\_app

**3.3 Deployment playbook**

Create **nginx\_deploy.yml**:

---

- name: Deploy custom index pages to AWS and Azure

hosts: all\_app\_servers

become: yes

tasks:

- name: Set index source based on cloud

ansible.builtin.set\_fact:

custom\_index\_src: "index-{{ cloud }}.html"

- name: Copy custom index.html to web root

ansible.builtin.copy:

src: "{{ custom\_index\_src }}"

dest: /var/www/html/index.html

owner: www-data

group: www-data

mode: '0644'

notify: Restart nginx

- name: Ensure Nginx default site uses index.html

ansible.builtin.lineinfile:

path: /etc/nginx/sites-available/default

regexp: '^\s\*index '

line: "\tindex index.html;"

backup: yes

notify: Restart nginx

- name: Check Nginx page from localhost

ansible.builtin.command: curl -s http://localhost

register: nginx\_page

changed\_when: false

- name: Assert correct welcome text

ansible.builtin.assert:

that:

- >

(cloud == 'aws' and 'Welcome to AWS' in nginx\_page.stdout)

or

(cloud == 'azure' and 'Welcome to Azure' in nginx\_page.stdout)

fail\_msg: "Custom welcome text not correct on {{ inventory\_hostname }}"

success\_msg: "Custom welcome text correct on {{ inventory\_hostname }}"

handlers:

- name: Restart nginx

ansible.builtin.service:

name: nginx

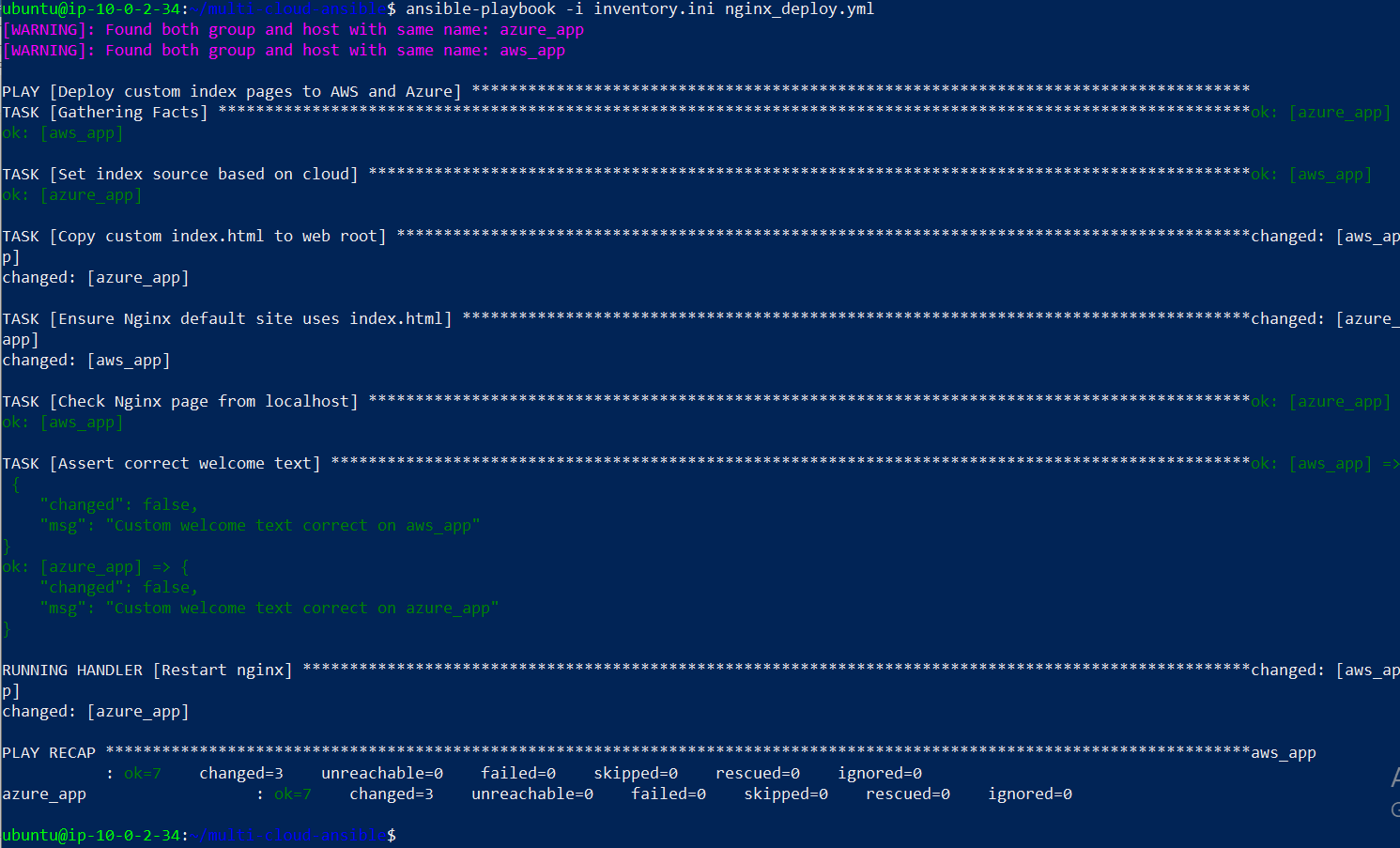
state: restarted

Run:

ansible-playbook -i inventory.ini nginx\_deploy.yml

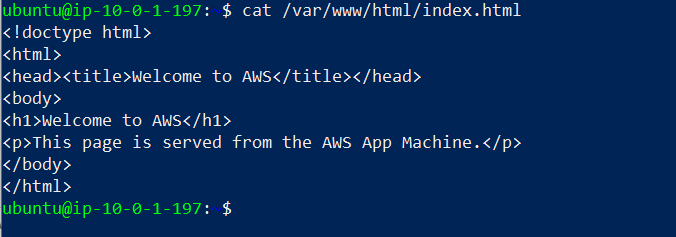
**Screenshots:**

* Playbook run showing copy + assert tasks success.

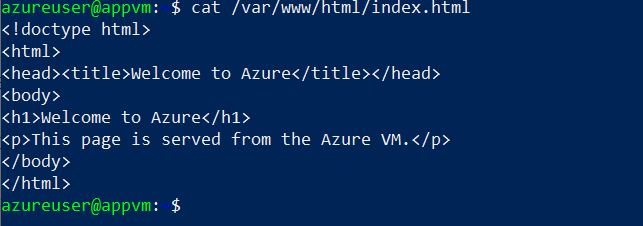


On **AWS app**:

cat /var/www/html/index.html



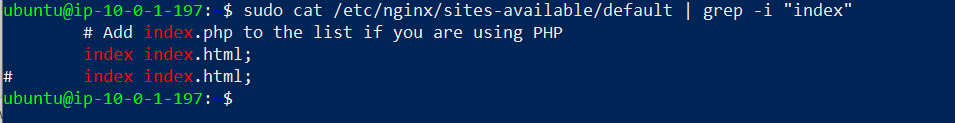
On **Azure app**: cat /var/www/html/index.html

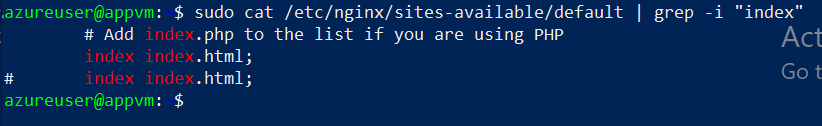


Nginx config:

sudo cat /etc/nginx/sites-available/default | grep -i "index"

**Screenshot:**  
Line showing **index index.html**;.

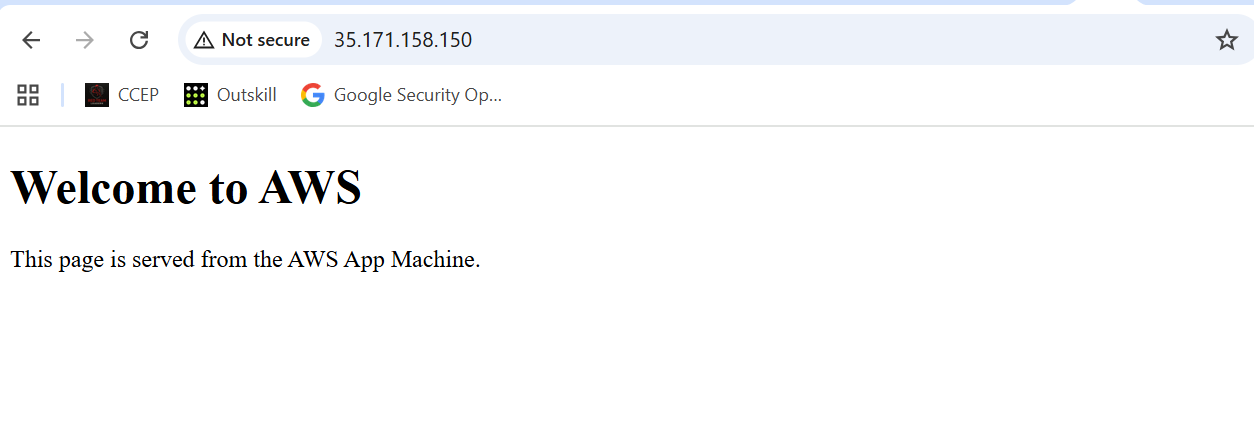




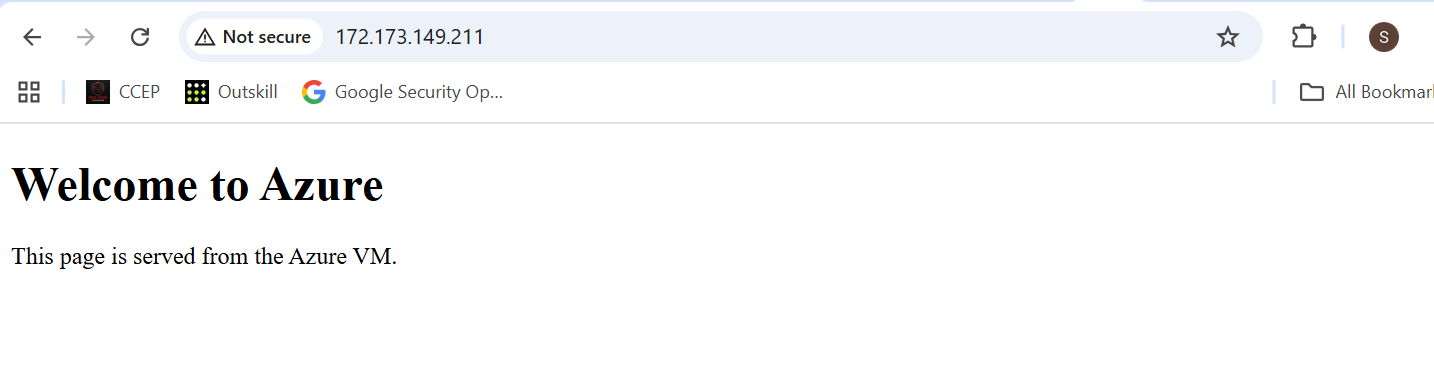
**3.4 Browser verification**

From your local machine:

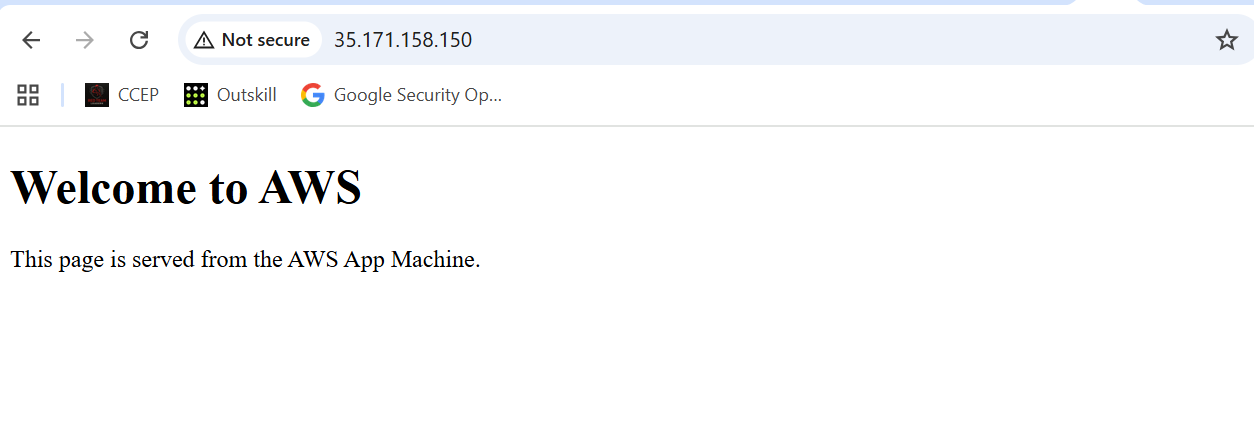
* Open http://<AWS\_APP\_PUBLIC\_IP> → **Welcome to AWS**



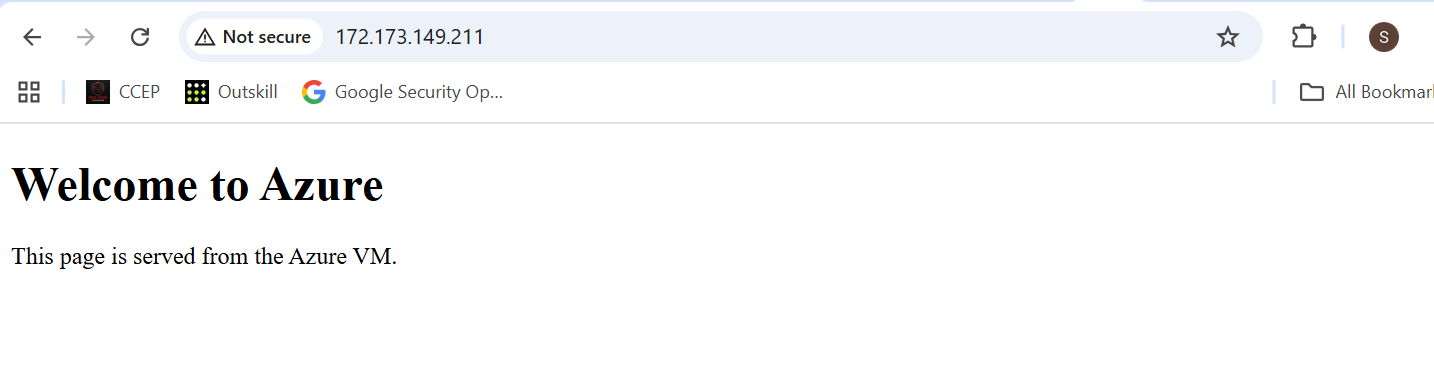
* Open http://<AZURE\_APP\_PUBLIC\_IP> → **Welcome to Azure**



**Screenshots:** Browser for AWS page.



* Browser for Azure page.



**Task 4 – Jenkins Setup for Continuous Deployment**

**4.1 Install Jenkins on Tools Machine**

On Tools Machine:

**sudo apt update**

**sudo apt install -y openjdk-17-jdk**

**java -version**

Add Jenkins repo & install:

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

**sudo apt update**

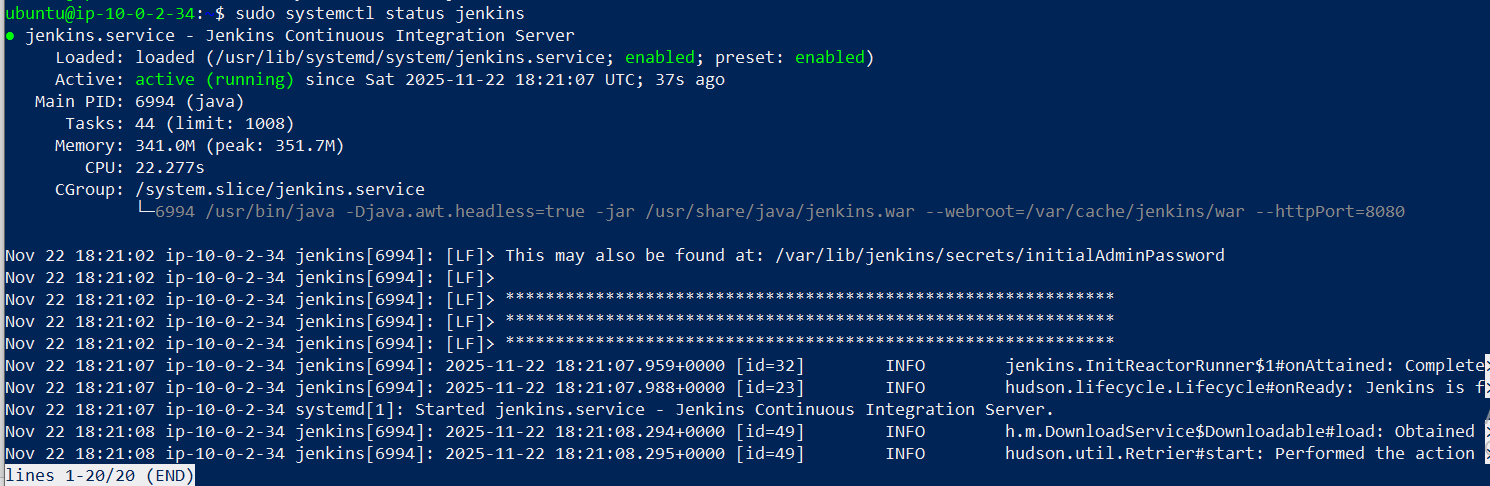
**sudo apt install -y jenkins**

**sudo systemctl enable jenkins**

**sudo systemctl start jenkins**

**sudo systemctl status jenkins**

**Screenshot:**  
systemctl status jenkins = active.



Update security group for Tools Machine to allow **TCP 8080** from your IP.

Open from browser:

http://<AWS\_TOOLS\_PUBLIC\_IP>:8080

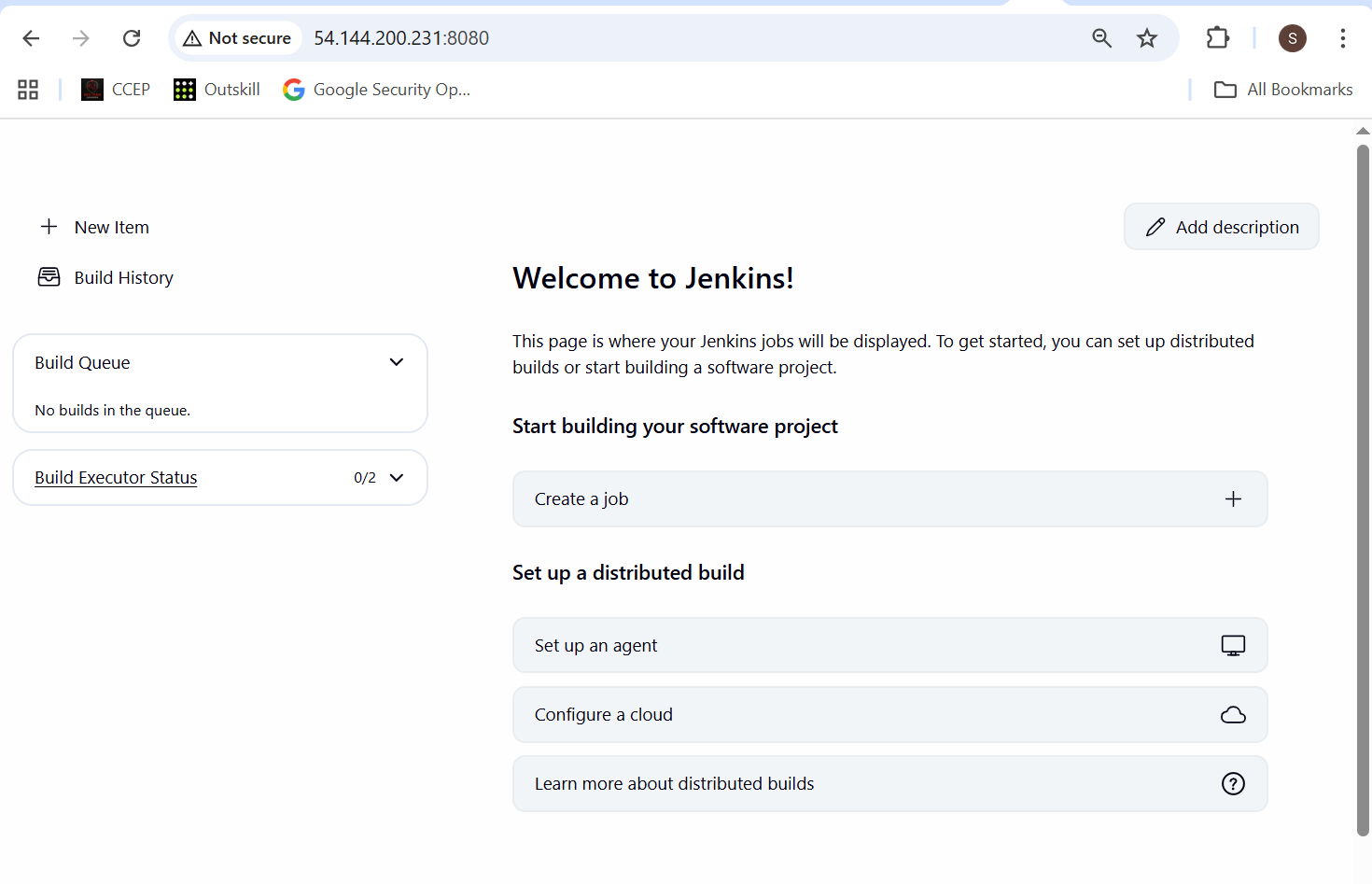
Get initial password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Finish Jenkins setup and create admin user.

**Screenshots:**

* Jenkins first dashboard after setup.



**4.2 Prepare GitHub repo**

Create a GitHub repo, e.g. **multi-cloud-dr-cd** containing:

* inventory.ini (same content but key path **/var/lib/jenkins/.ssh/mcdr\_key**)
* nginx\_deploy.yml
* index-aws.html
* index-azure.html
* Jenkinsfile

On Tools Machine, copy key for Jenkins:

sudo mkdir -p ~jenkins/.ssh

sudo cp /home/ubuntu/.ssh/mcdr\_key ~jenkins/.ssh/

sudo chown -R jenkins:jenkins ~jenkins/.ssh

sudo chmod 700 ~jenkins/.ssh

sudo chmod 600 ~jenkins/.ssh/mcdr\_key

In repo’s **inventory.ini**:

[aws\_app]

aws\_app ansible\_host=<AWS\_APP\_PUBLIC\_IP> ansible\_user=ubuntu ansible\_ssh\_private\_key\_file=/var/lib/jenkins/.ssh/mcdr\_key cloud=aws

[azure\_app]

azure\_app ansible\_host=<AZURE\_APP\_PUBLIC\_IP> ansible\_user=azureuser ansible\_ssh\_private\_key\_file=/var/lib/jenkins/.ssh/mcdr\_key cloud=azure

[all\_app\_servers:children]

aws\_app

azure\_app

**4.3 Jenkinsfile**

Create **Jenkinsfile** in repo:

pipeline {

agent any

environment {

ANSIBLE\_FORCE\_COLOR = 'true'

}

stages {

stage('Checkout') {

steps {

checkout scm

}

}

stage('Deploy via Ansible') {

steps {

sh '''

ansible-playbook -i inventory.ini nginx\_deploy.yml

'''

}

}

}

post {

success {

echo 'Deployment successful.'

}

failure {

echo 'Deployment failed. Check logs.'

}

}

}

Push everything to GitHub.

**4.4 Configure Jenkins job**

In Jenkins UI:

1. **New Item** → choose **Pipeline**.
2. Name: multi-cloud-dr-cd.
3. Under **Pipeline**:
   * Definition: **Pipeline script from SCM**
   * SCM: **Git**
   * Repo URL: https://github.com/sagarpatilbox/multi-cloud-dr-cd.git
   * Credentials: add if private
   * Script Path: **Jenkinsfile**

Save.

**Screenshot:**  
Jenkins job config page.

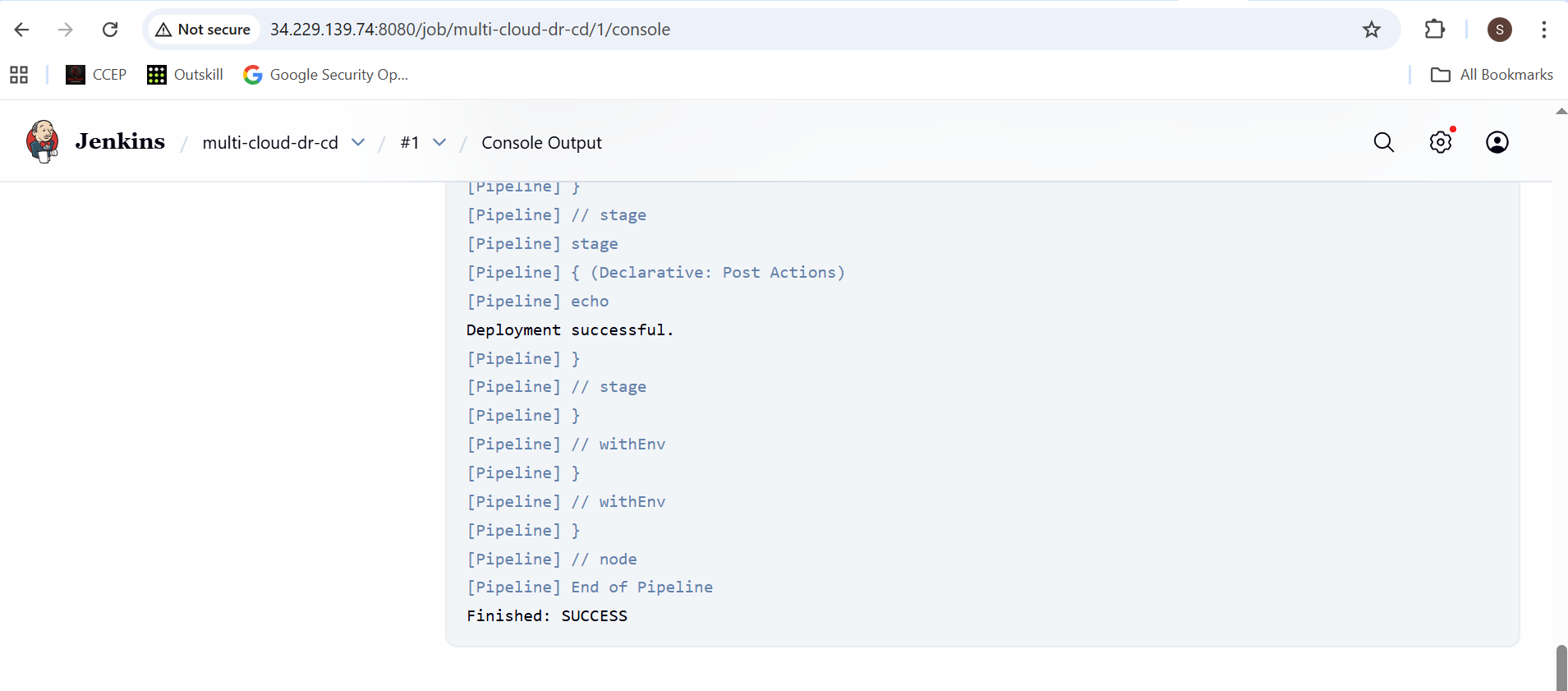


**4.5 Run pipeline**

Click **Build Now**.

**Screenshots:**

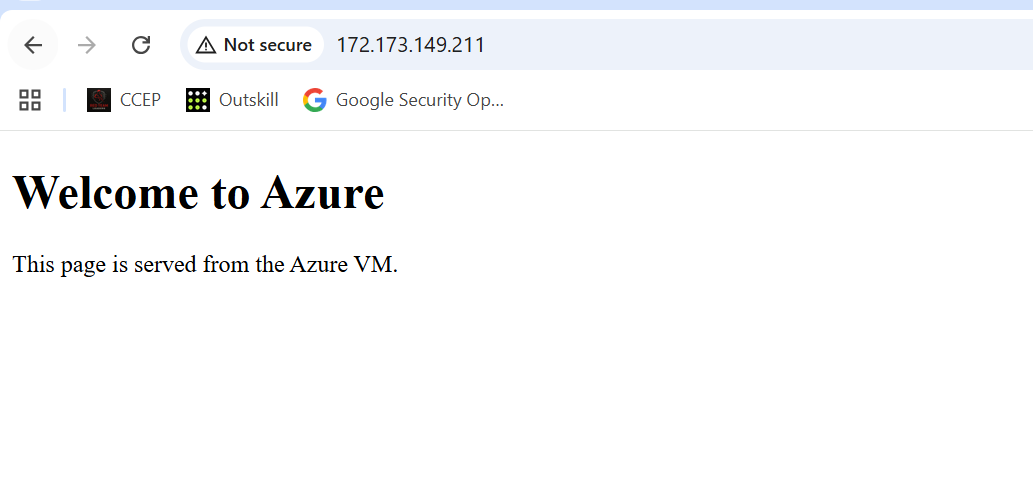
* Console output showing:

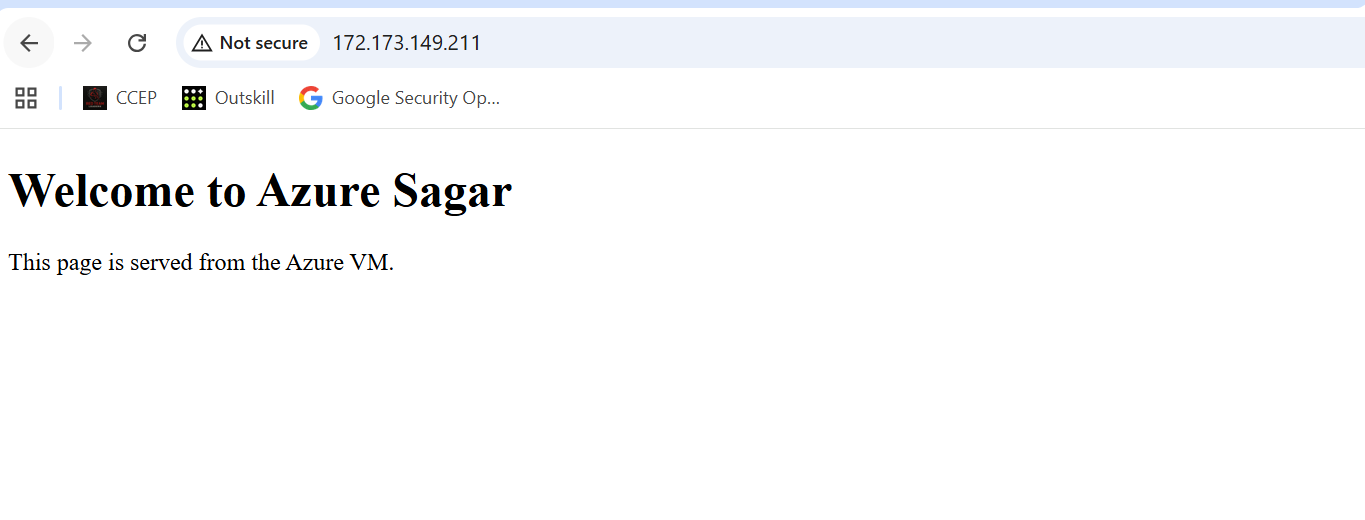


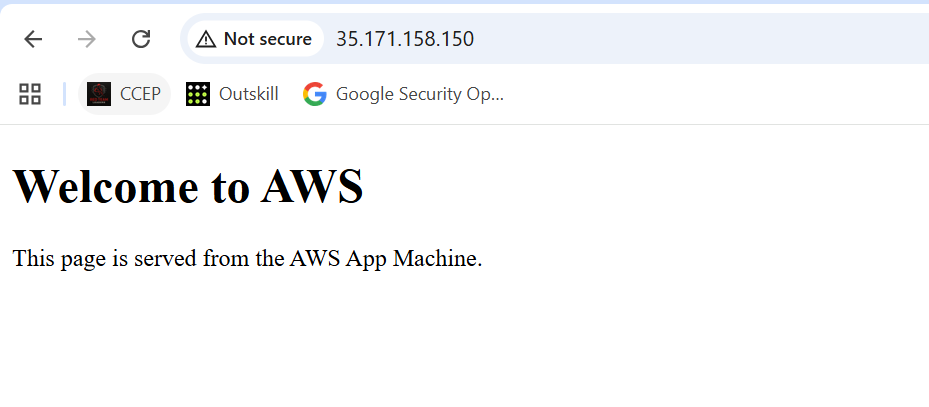
Now edit HTML in GitHub (e.g. add v2) → run/build again → check pages updated.

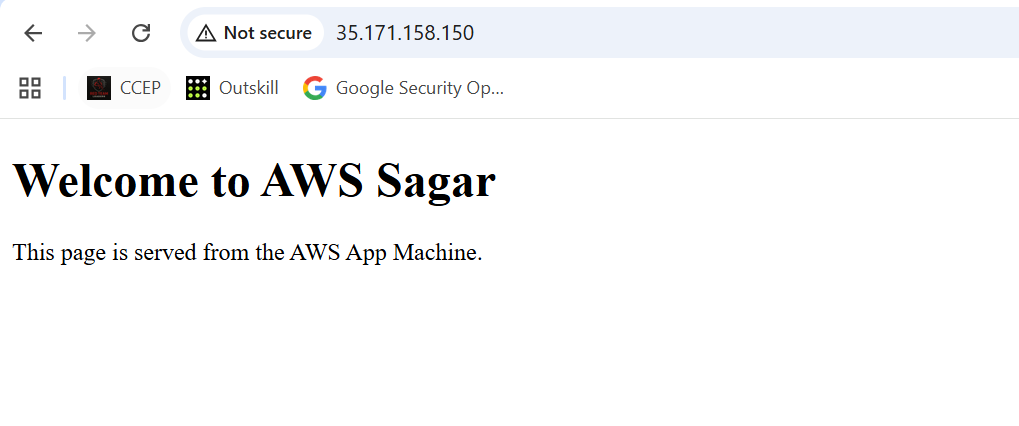
**Screenshots:**

* Before/after browser pages.









**Task 5 – Traffic Management Using AWS Route 53**

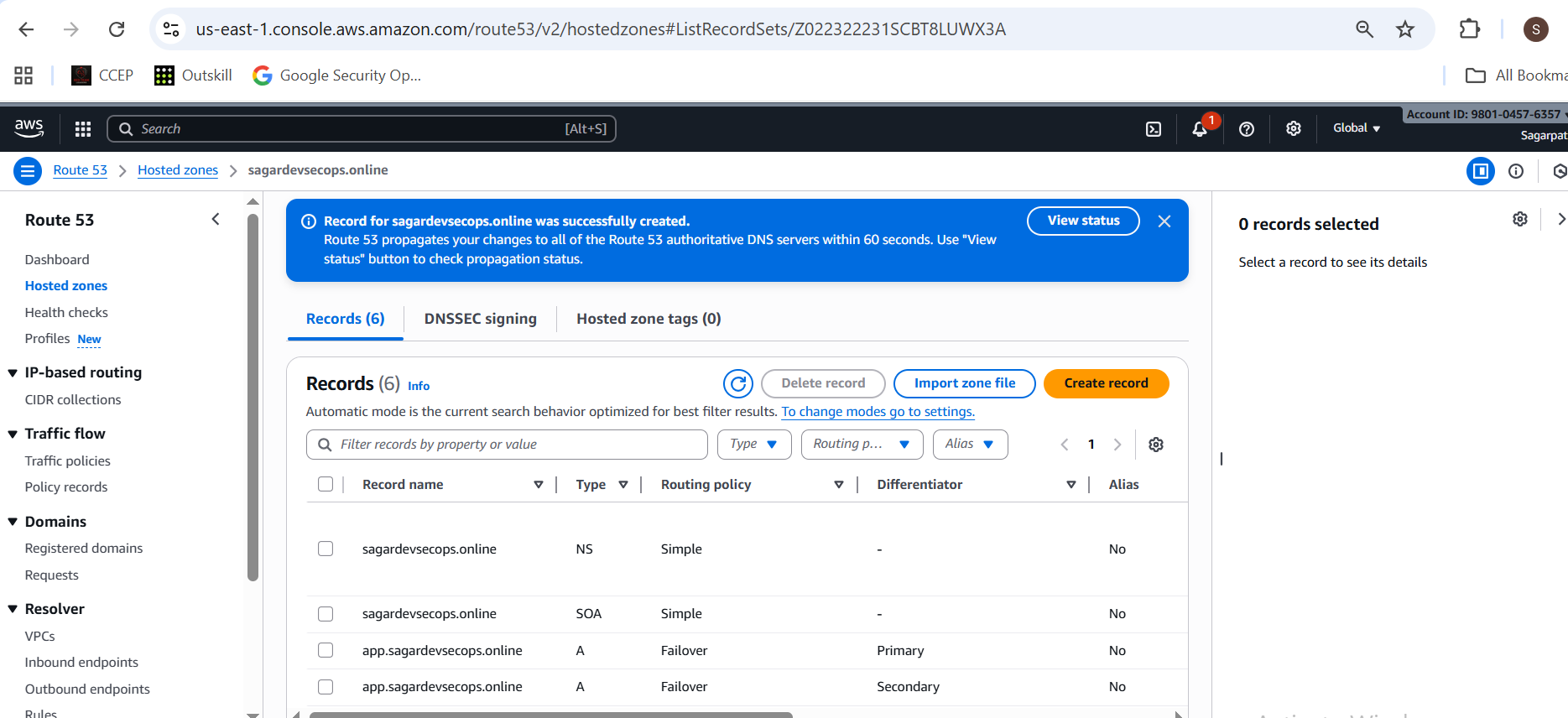
*I am Using My domain* ***app.sagardevsecops.online***

**5.1 Create hosted zone**

In AWS Console → **Route 53 → Hosted zones → Create hosted zone**:

* Domain name: sagardevsecops.online
* Type: Public hosted zone

**Screenshot:**  
Hosted zone overview.

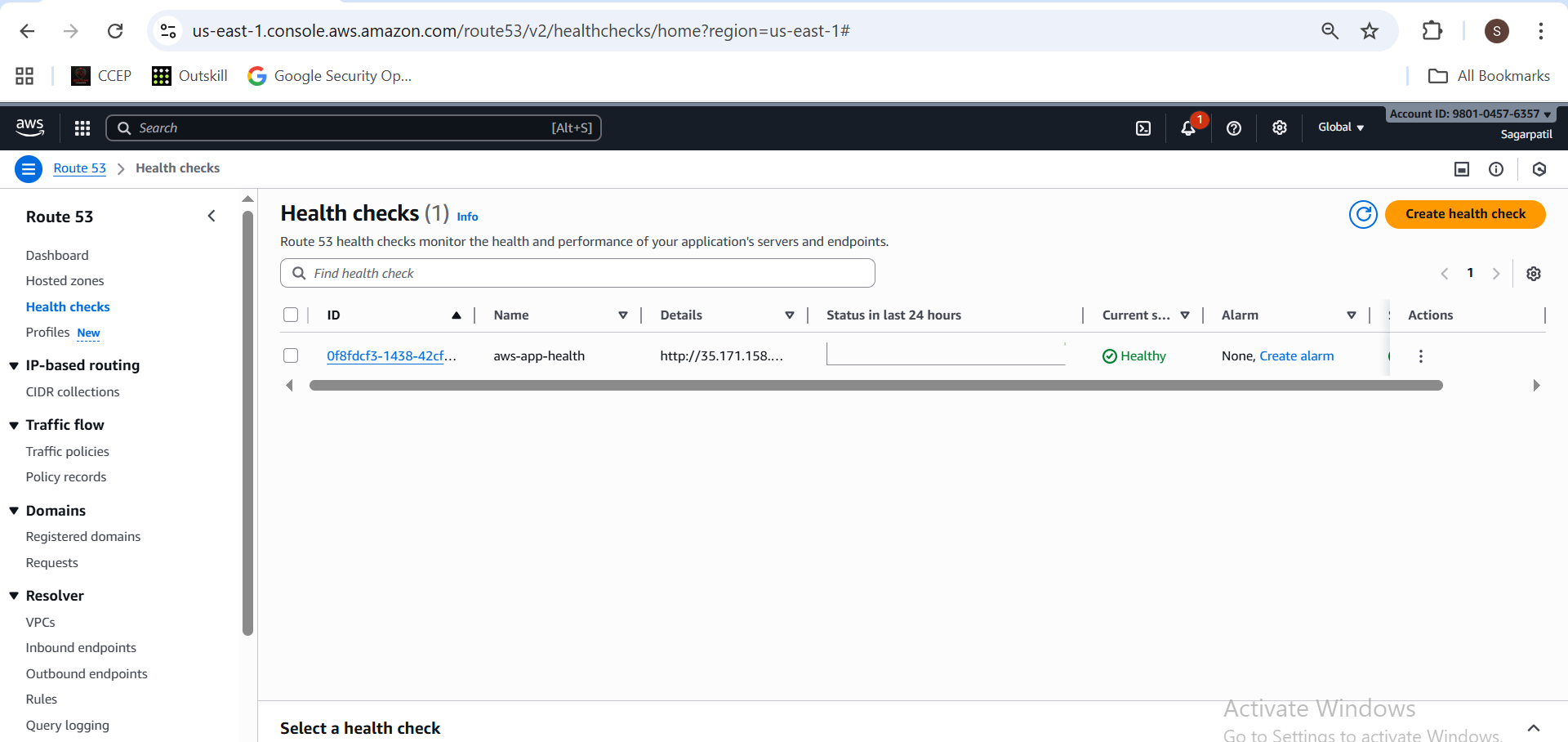


**5.2 Create health check for AWS app**

Route 53 → **Health checks → Create health check**:

* Name: **aws-app-health**
* Endpoint → IP address = <AWS\_APP\_PUBLIC\_IP>
* Protocol: HTTP
* Port: 80
* Path: /

**Screenshot:**  
Health check details (status OK when app running).



**5.3 Create failover records**

In hosted zone → **Create record**:

**Primary record (AWS)**

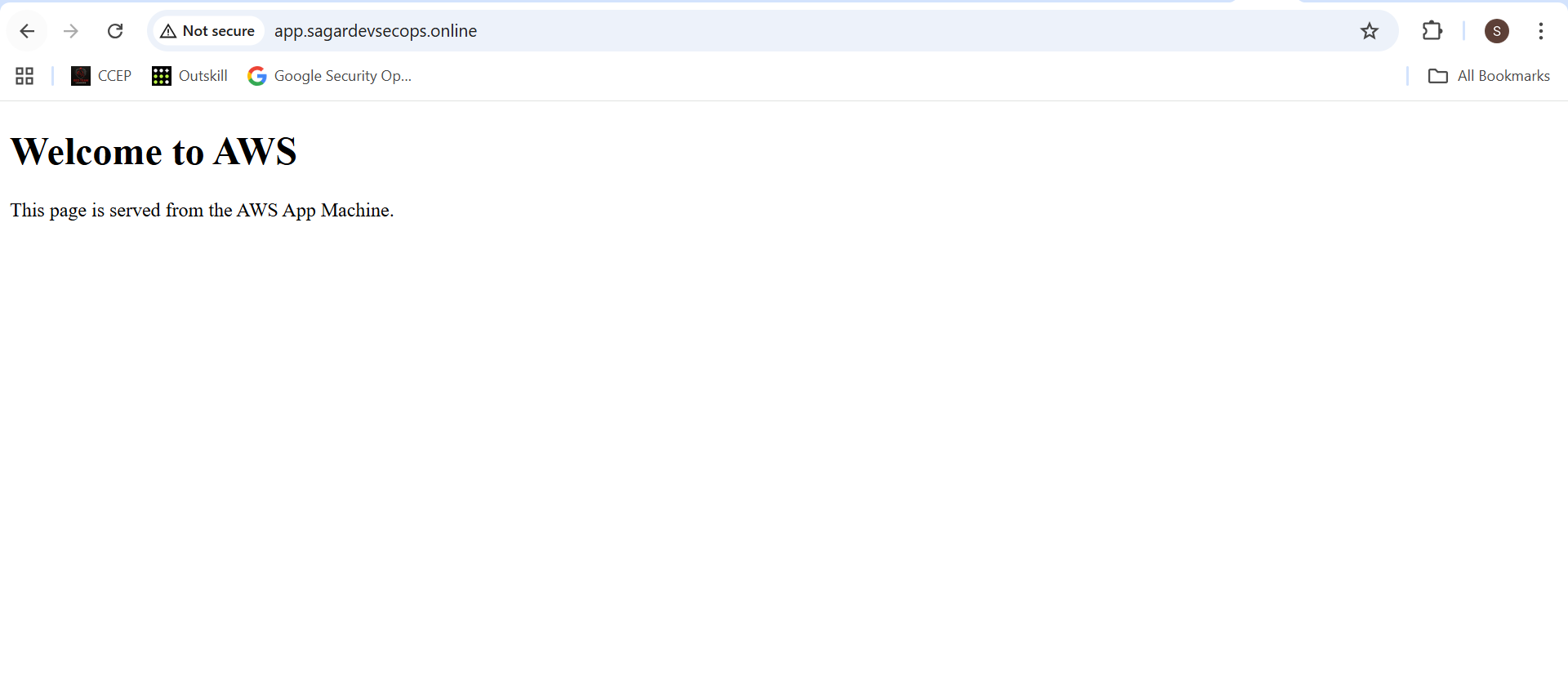
* Record name: app.sagardevsecops.online
* Type: A
* Routing policy: **Failover**
* Failover record type: **Primary**
* Value: <AWS\_APP\_PUBLIC\_IP>
* Health check: aws-app-health
* Record ID: aws-primary

**Secondary record (Azure)**

* Record name: app.sagardevsecops.online
* Type: A
* Routing policy: **Failover**
* Failover record type: **Secondary**
* Value: <AZURE\_APP\_PUBLIC\_IP>
* No health check
* Record ID: azure-secondary

**Screenshots:**

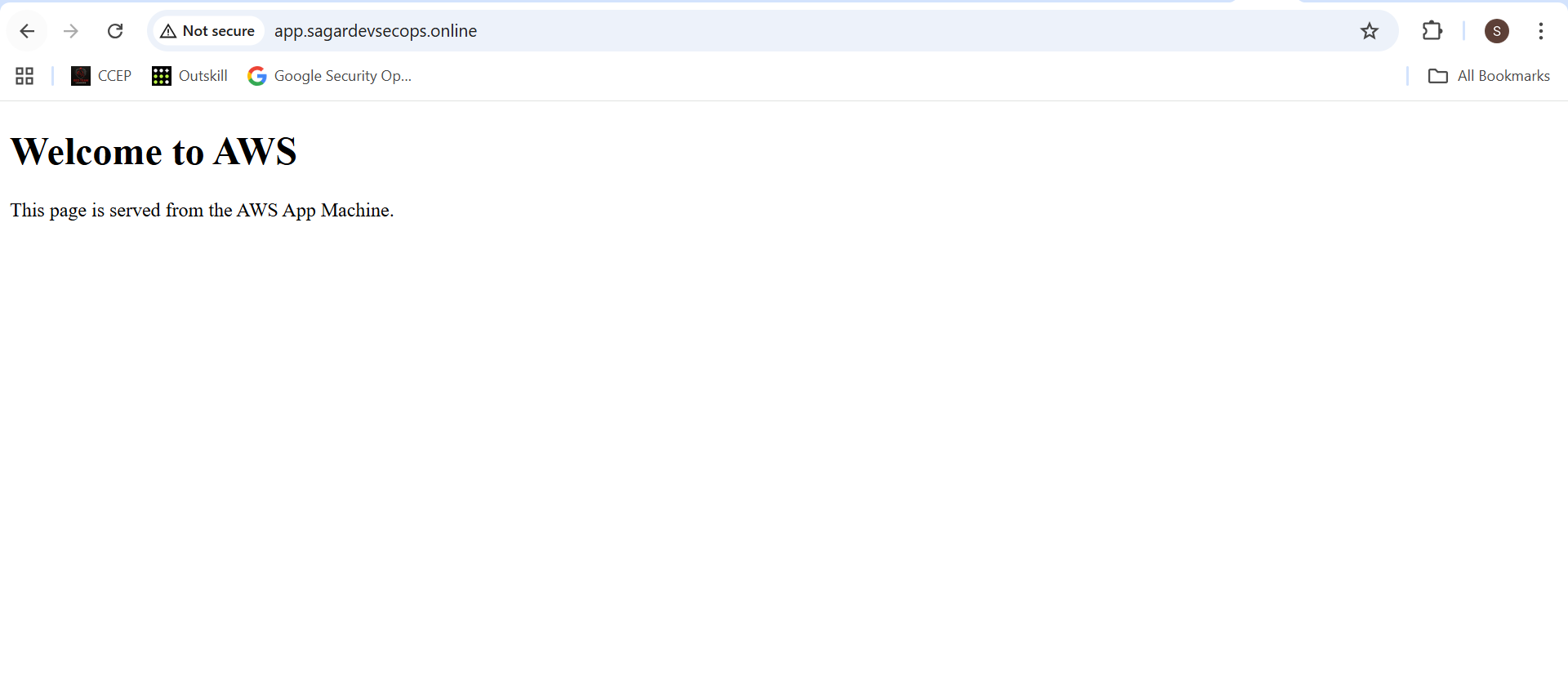
app.sagardevsecops.online



**5.4 Test failover**

1. With AWS app **running**:
2. curl <http://app.sagardevsecops.online>

**Screenshot:** Browser output.

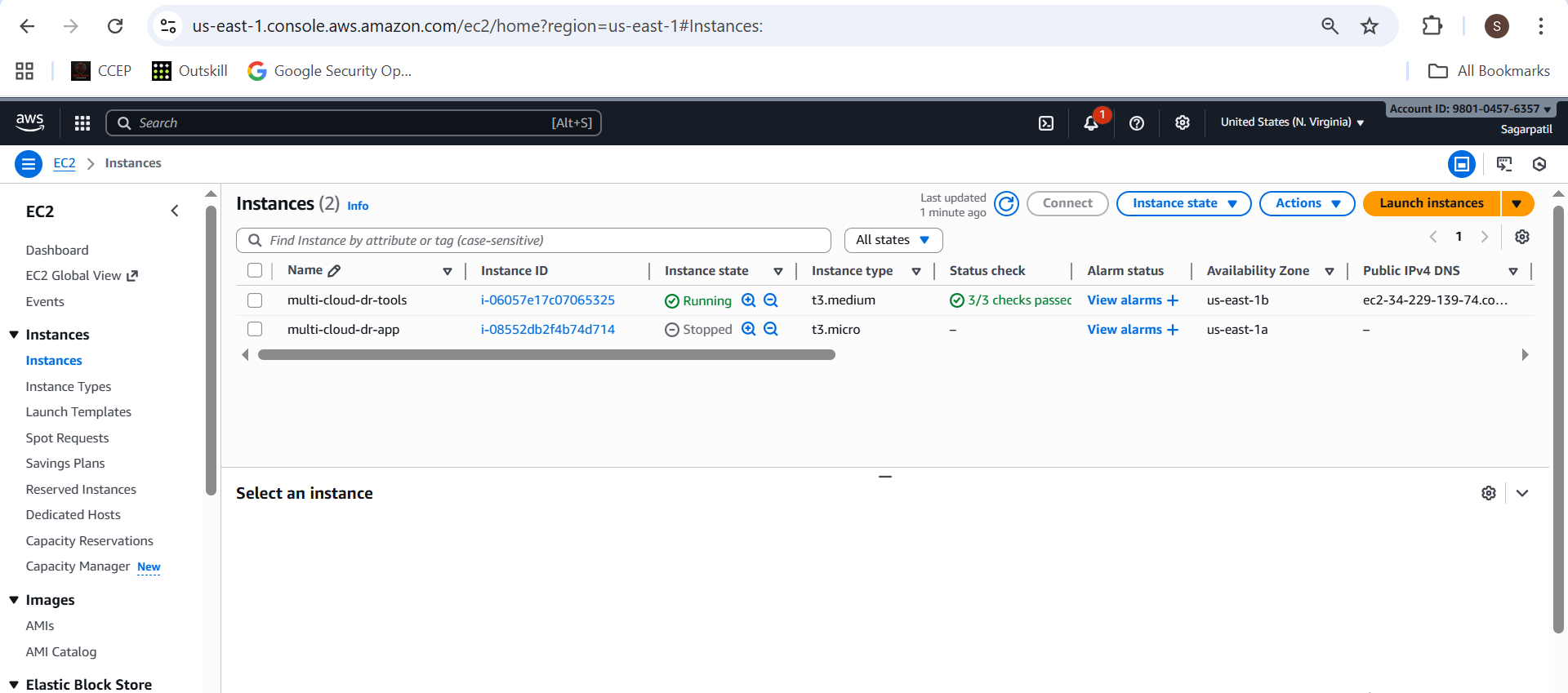


1. Stop AWS App instance in EC2 console.
   * Wait for health check to show **Unhealthy**.

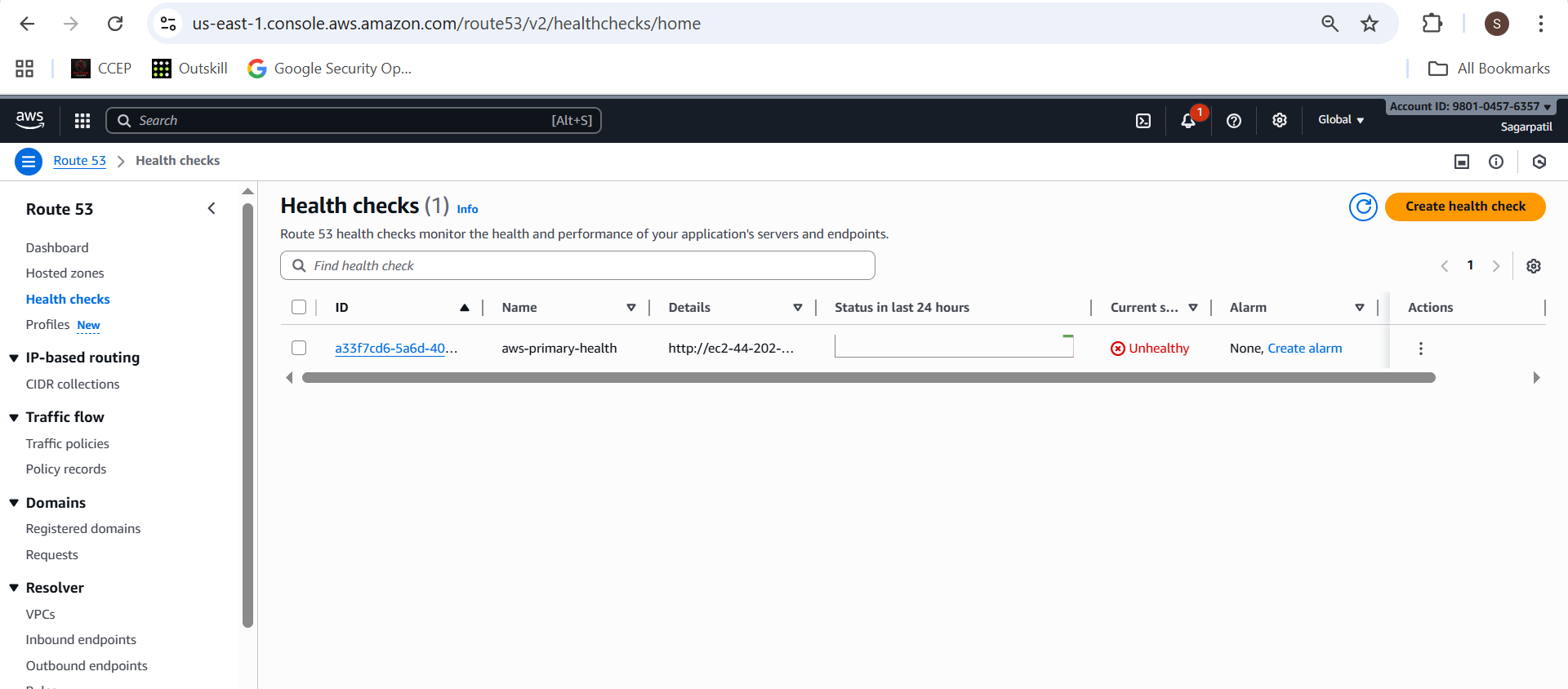
Now you should see **Welcome to Azure**.

**Screenshots:**

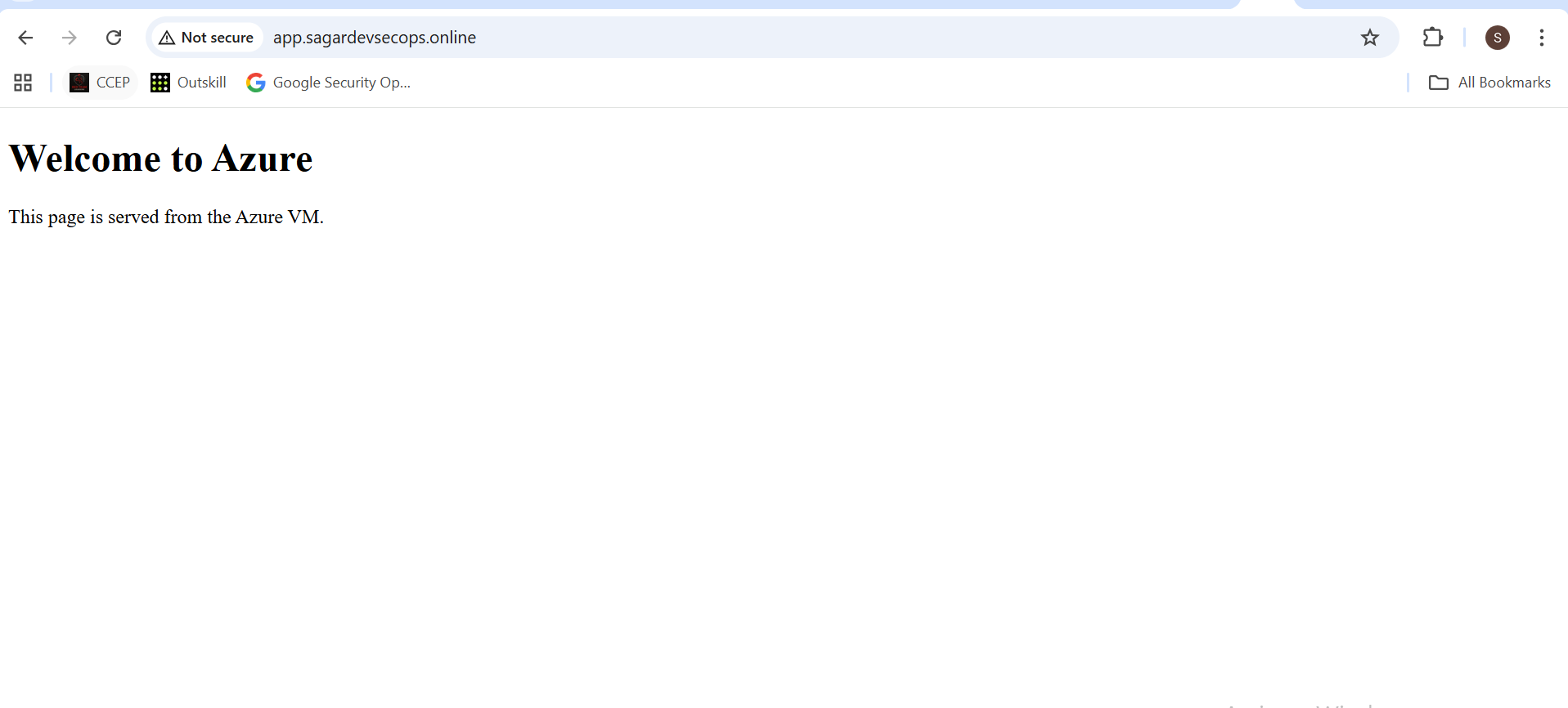
* + EC2 instance state = stopped.



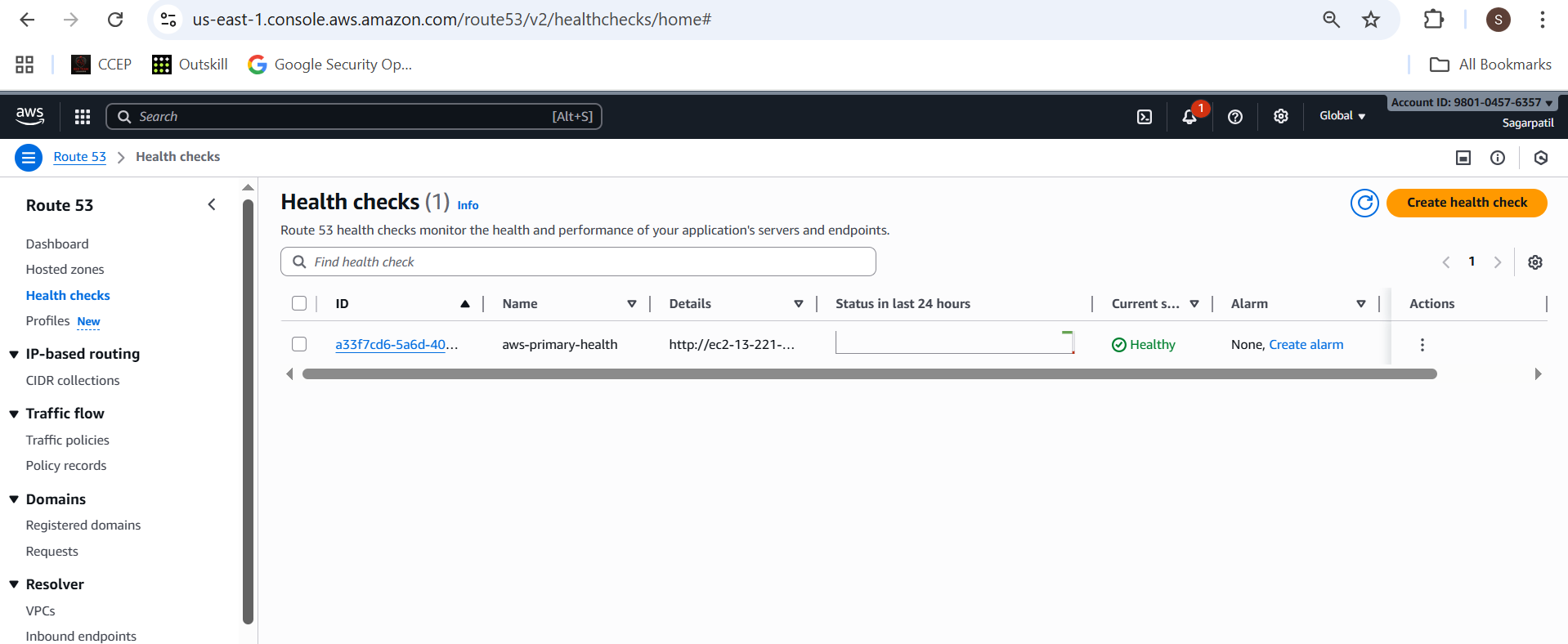
* + Health check status = Unhealthy.



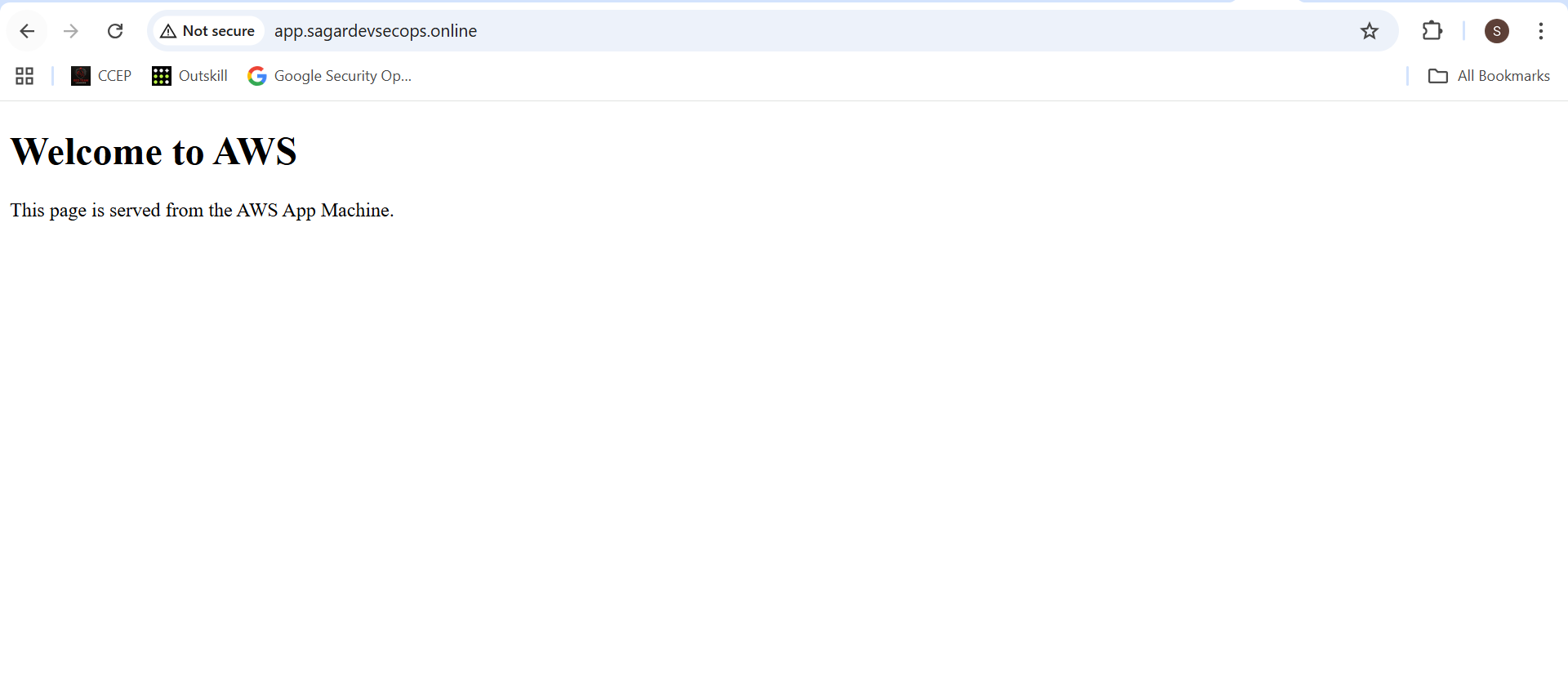
* + Browser showing Azure page via app.sagardevsecops.online.



1. Start AWS instance again, wait until health check is healthy, and re-test.



* Browser showing AWS page again.



**⭐ Bonus Task – Kubernetes (EKS + AKS + Nginx + Failover)**

This is optional but nice.

**B.1 EKS via Terraform (high-level)**

In project/k8s/aws-eks/main.tf you can use:

terraform {

required\_version = ">= 1.3.0"

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 5.0"

}

}

}

provider "aws" {

region = "us-east-1"

}

# 1) VPC module

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

version = "~> 5.0"

name = "eks-dr-vpc"

cidr = "10.20.0.0/16"

azs = ["us-east-1a", "us-east-1b"]

private\_subnets = ["10.20.1.0/24", "10.20.2.0/24"]

public\_subnets = ["10.20.11.0/24", "10.20.12.0/24"]

enable\_nat\_gateway = true

}

# 2) EKS module

module "eks" {

source = "terraform-aws-modules/eks/aws"

version = "~> 20.0"

cluster\_name = "eks-dr-cluster"

cluster\_version = "1.30"

vpc\_id = module.vpc.vpc\_id

subnet\_ids = module.vpc.private\_subnets

# EKS API endpoint - public everywhere (lab only!)

cluster\_endpoint\_private\_access = true

cluster\_endpoint\_public\_access = true

cluster\_endpoint\_public\_access\_cidrs = ["0.0.0.0/0"]

# Let the Terraform caller (your admin user) be cluster admin

authentication\_mode = "API\_AND\_CONFIG\_MAP"

enable\_cluster\_creator\_admin\_permissions = true

eks\_managed\_node\_groups = {

default = {

min\_size = 1

max\_size = 2

desired\_size = 1

instance\_types = ["t3.small"]

}

}

}

Run:

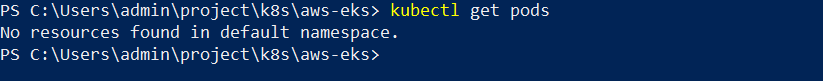
terraform init

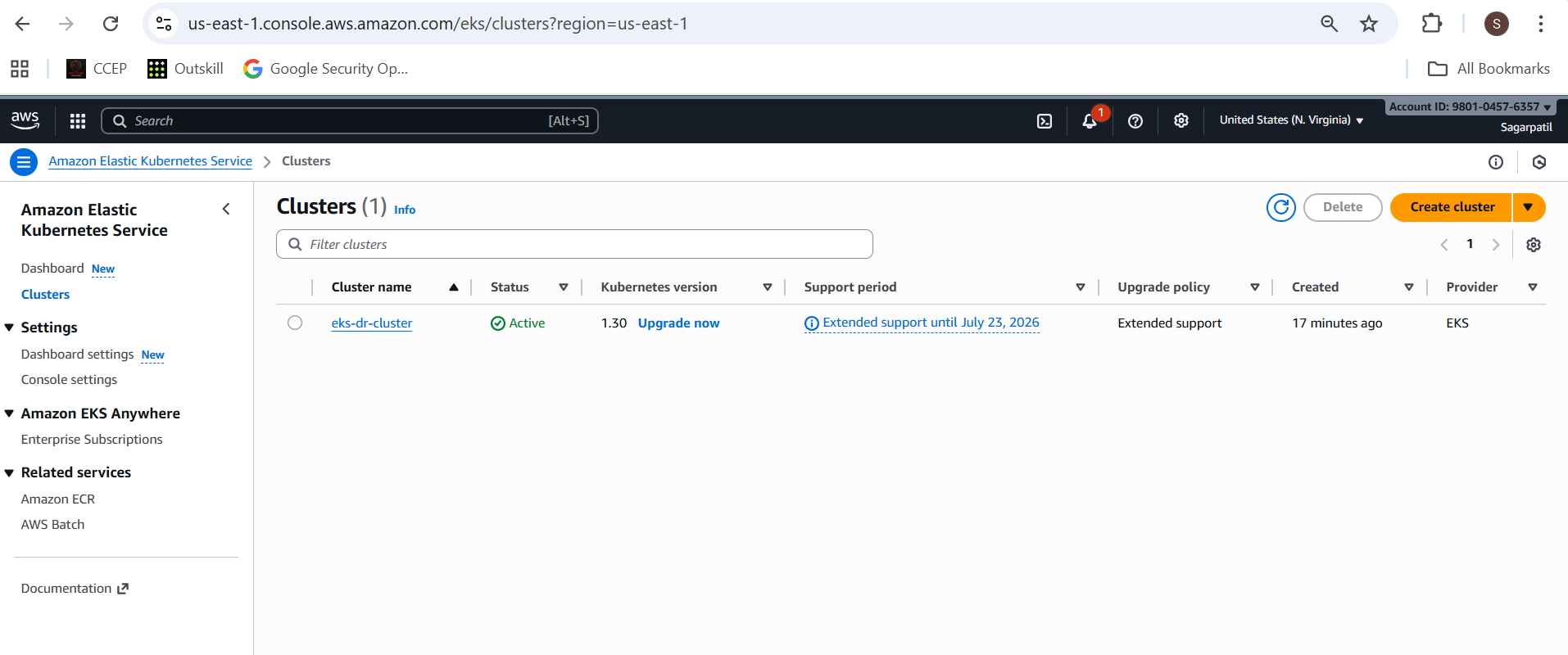
terraform apply

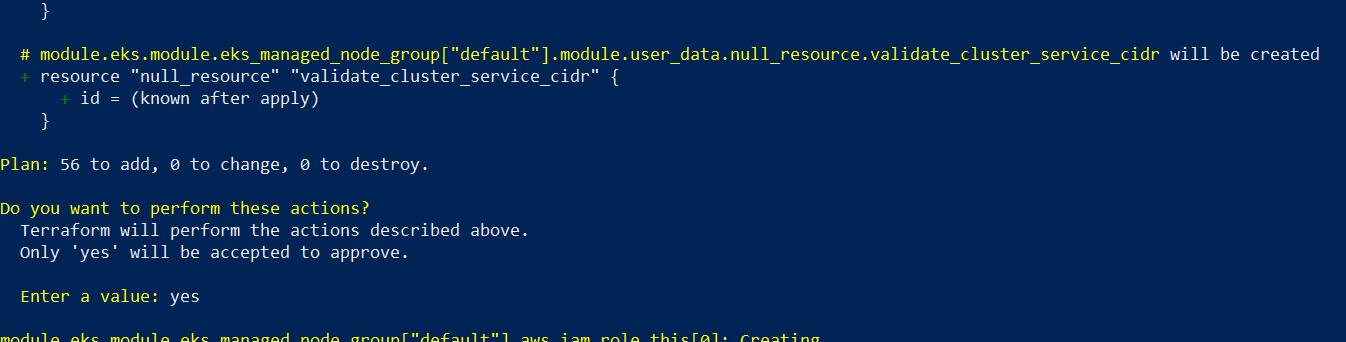
aws eks update-kubeconfig --name eks-dr-cluster --region us-east-1

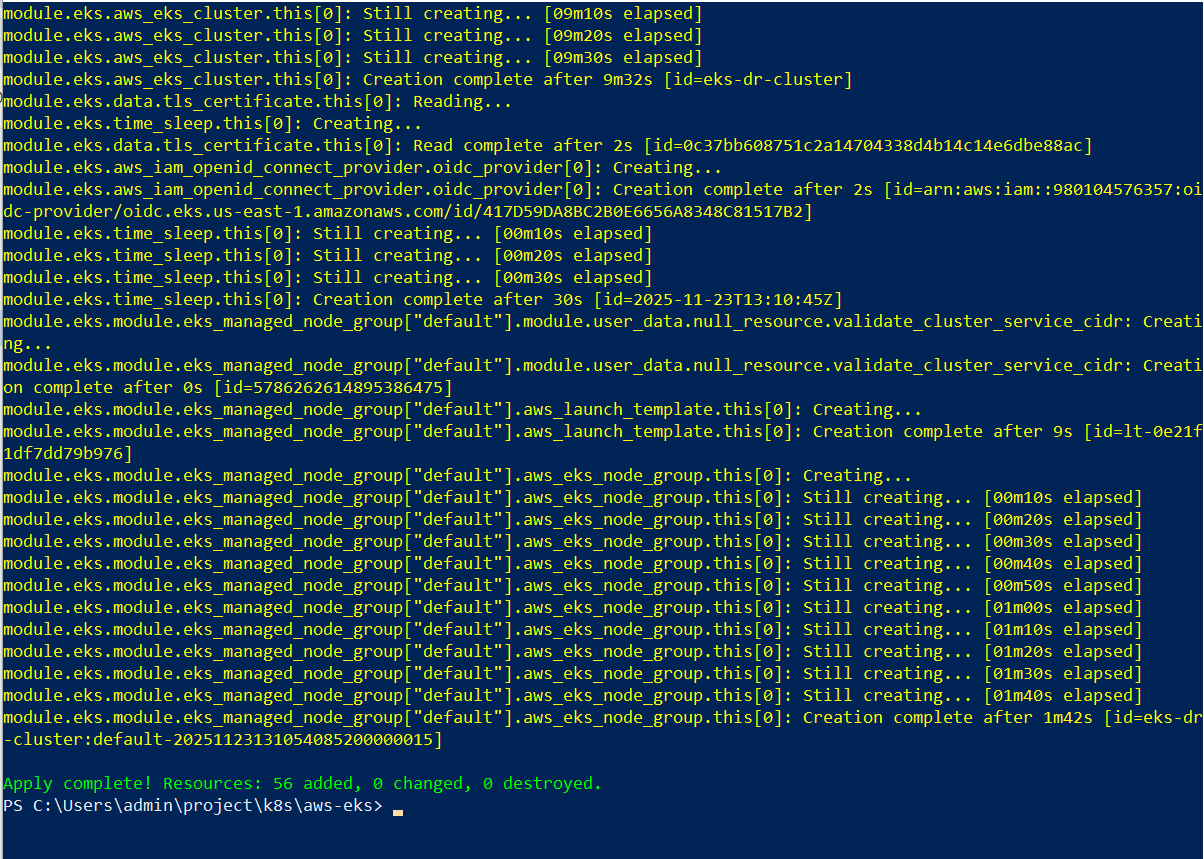
kubectl get nodes

**Screenshots:**  
kubectl get nodes , EKS console.









**B.2 AKS via Terraform**

project/k8s/azure-aks/main.tf similar to earlier AKS config:

provider "azurerm" {

features {}

subscription\_id = "4c419c18-22ee-4dfa-8014-71f05d545136"

}

resource "azurerm\_resource\_group" "rg" {

name = "aks-dr-rg"

location = "West US3"

}

resource "azurerm\_kubernetes\_cluster" "aks" {

name = "aks-dr-cluster"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

dns\_prefix = "aksdr"

default\_node\_pool {

name = "default"

node\_count = 1

vm\_size = "Standard\_B2s"

}

identity {

type = "SystemAssigned"

}

}

Then: **az login** # for login

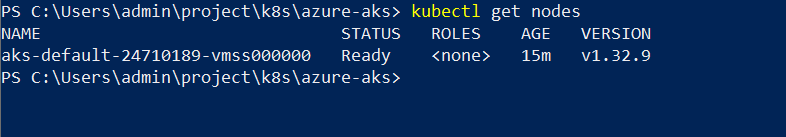
terraform init

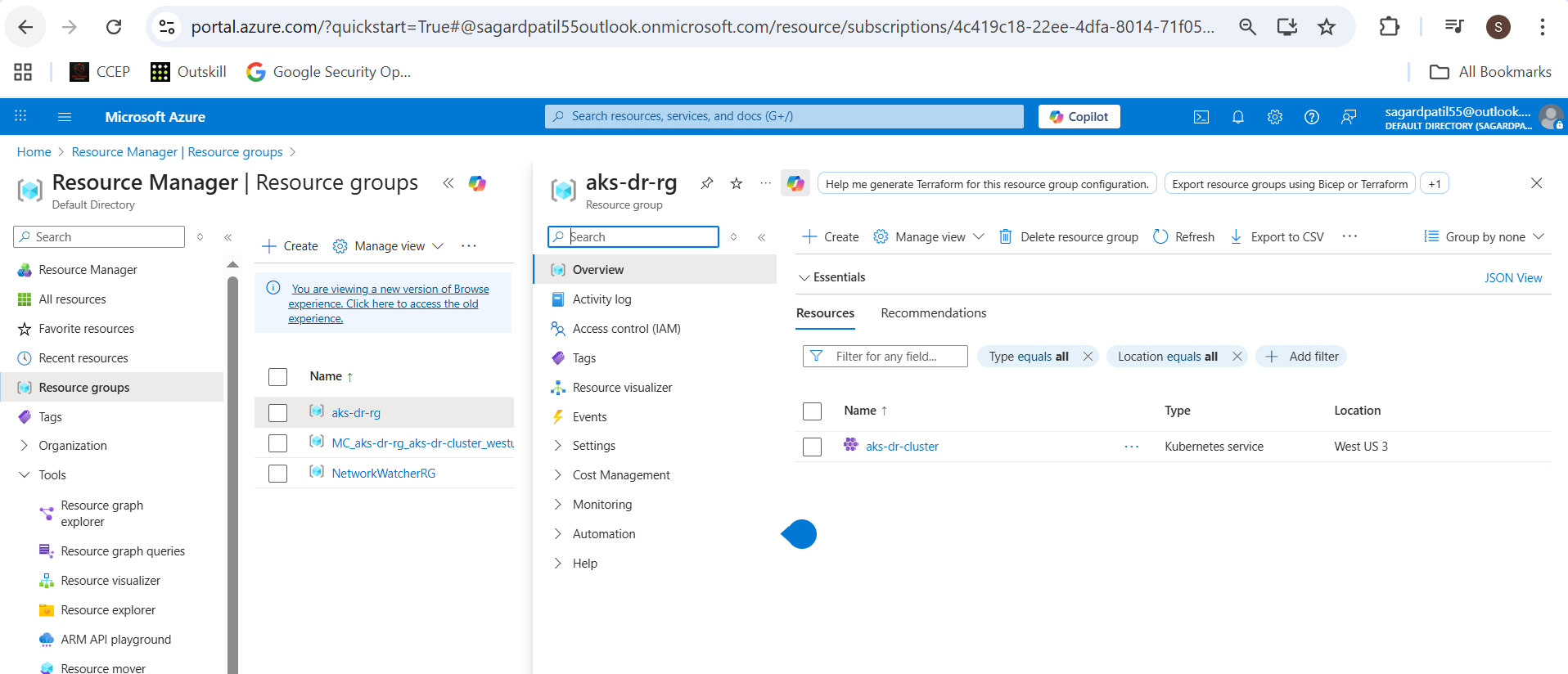
terraform apply

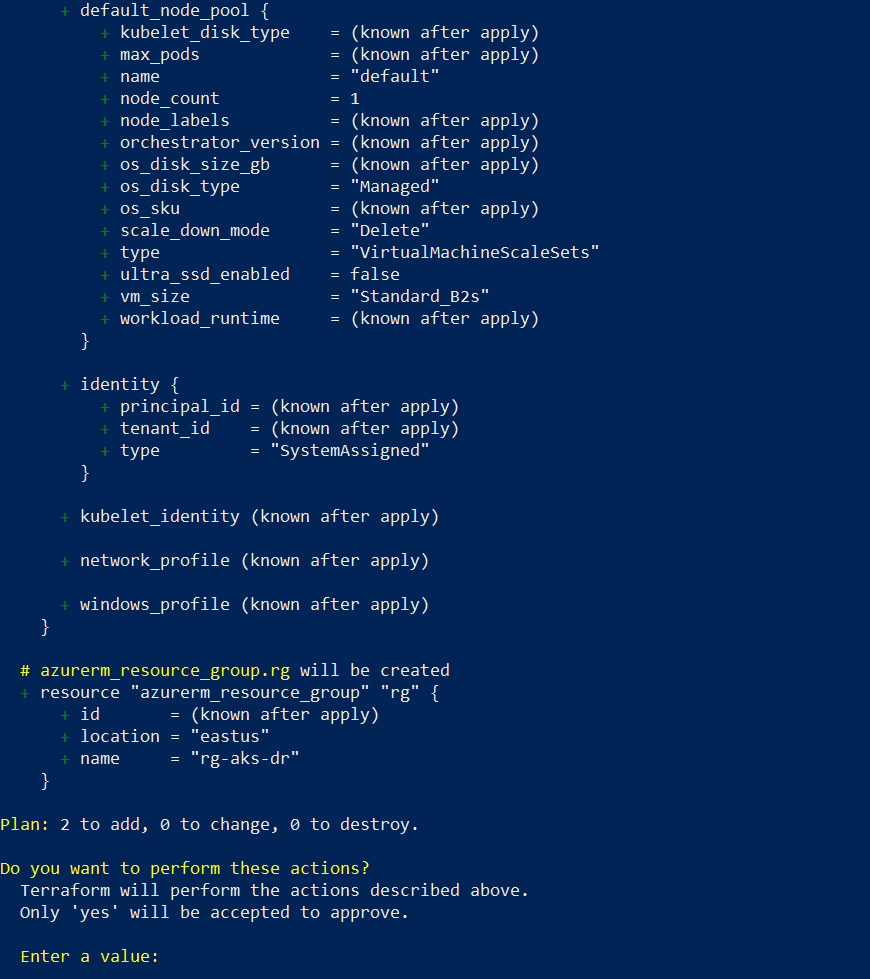
terraform output -raw kube\_config > aks.kubeconfig

KUBECONFIG=aks.kubeconfig kubectl get nodes

**Screenshots:**  
AKS cluster, kubectl get nodes.







**B.3 Nginx manifests (same for both clusters)**

**configmap-aws.yaml** (**apply to EKS**):

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-index

data:

index.html: |

<html><body><h1>Welcome to AWS - Kubernetes</h1></body></html>

**configmap-azure.yaml** **(apply to AKS):**

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-index

data:

index.html: |

<html><body><h1>Welcome to Azure - Kubernetes</h1></body></html>

**nginx-deployment.yaml**:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-k8s

spec:

replicas: 2

selector:

matchLabels:

app: nginx-k8s

template:

metadata:

labels:

app: nginx-k8s

spec:

containers:

- name: nginx

image: nginx:stable

ports:

- containerPort: 80

volumeMounts:

- name: index-html

mountPath: /usr/share/nginx/html/index.html

subPath: index.html

volumes:

- name: index-html

configMap:

name: nginx-index

items:

- key: index.html

path: index.html

nginx-service.yaml:

apiVersion: v1

kind: Service

metadata:

name: nginx-lb

spec:

type: LoadBalancer

selector:

app: nginx-k8s

ports:

- port: 80

targetPort: 80

Apply to **EKS**:

kubectl apply -f configmap-aws.yaml

kubectl apply -f nginx-deployment.yaml

kubectl apply -f nginx-service.yaml

kubectl get svc nginx-lb

Apply to **AKS**:

**For powershell** :

$Env:KUBECONFIG = "aks.kubeconfig"

az aks get-credentials `

--resource-group aks-dr-rg `

--name aks-dr-cluster

KUBECONFIG=aks.kubeconfig kubectl apply -f configmap-azure.yaml

KUBECONFIG=aks.kubeconfig kubectl apply -f nginx-deployment.yaml

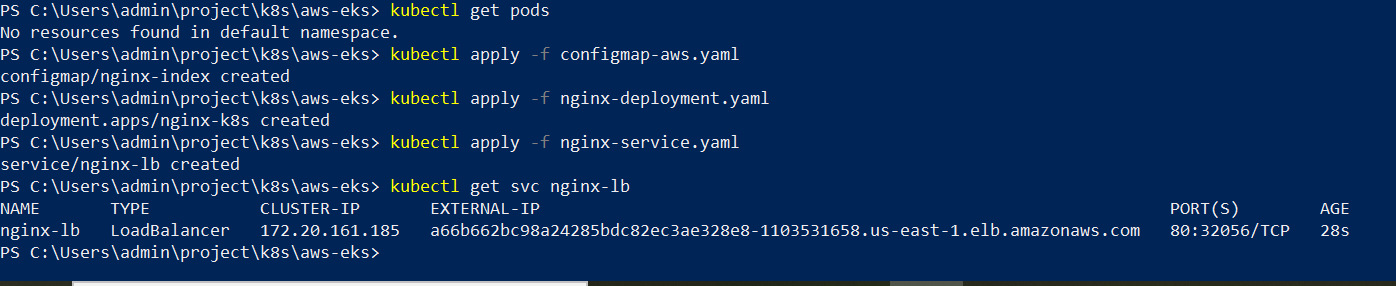
KUBECONFIG=aks.kubeconfig kubectl apply -f nginx-service.yaml

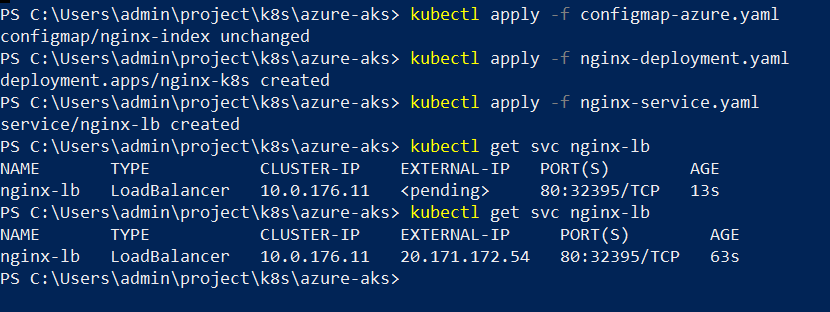
KUBECONFIG=aks.kubeconfig kubectl get svc nginx-lb

Use the **EXTERNAL-IP** values as endpoints.

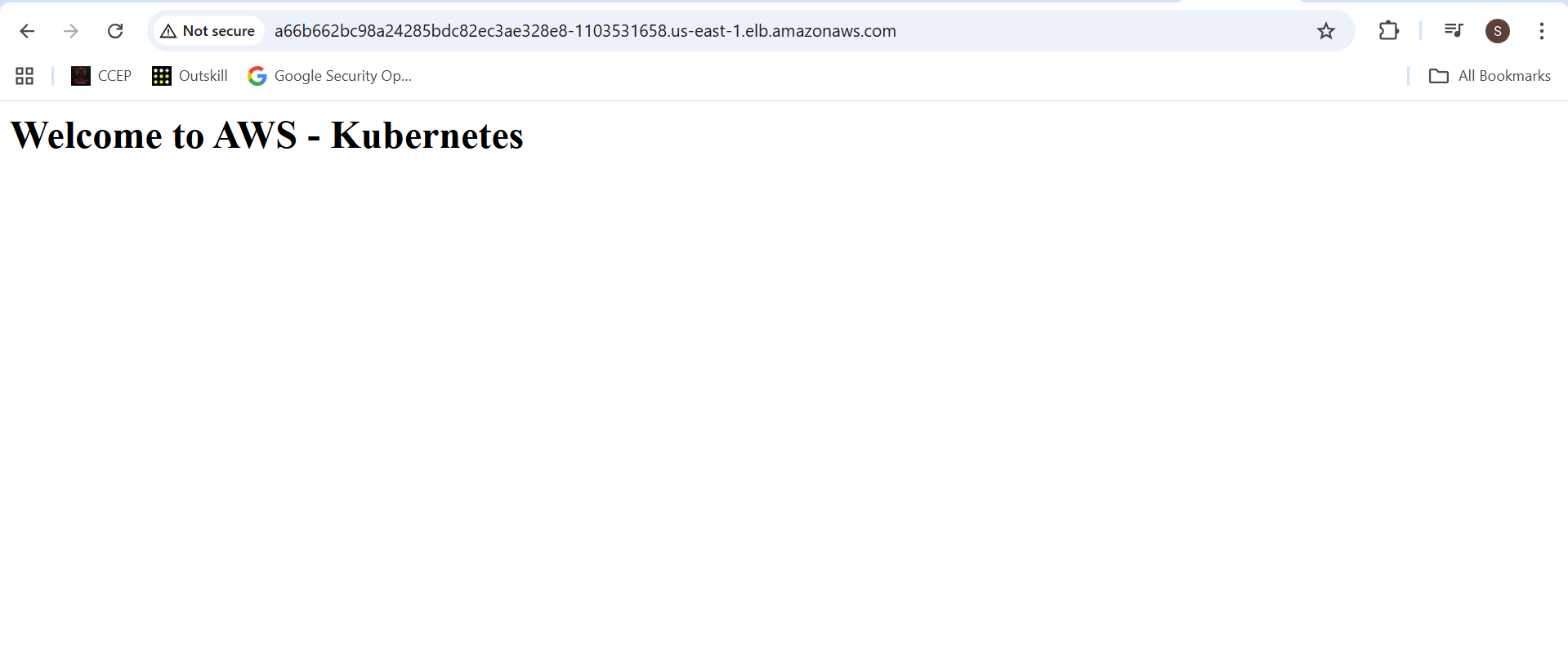
**Screenshots:**

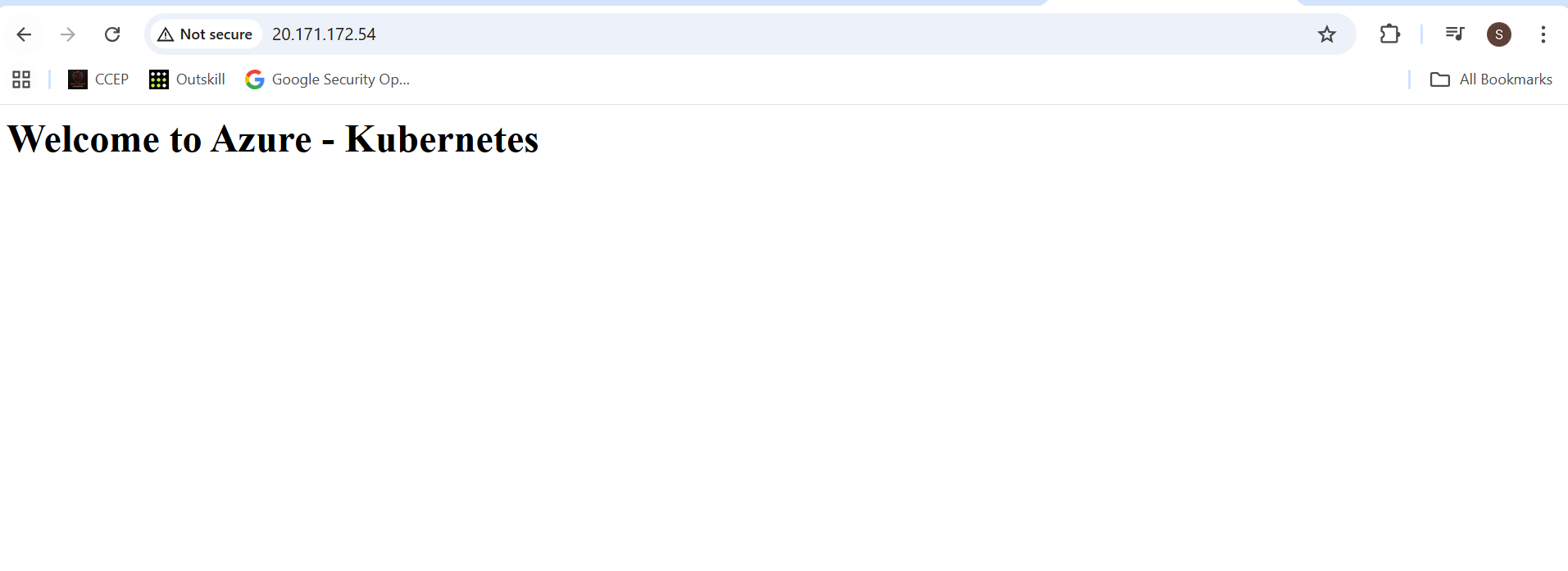
* kubectl get svc nginx-lb for both clusters.





* Browser showing AWS K8s page and Azure K8s page.







**B.4 Route 53 failover with LB IPs**

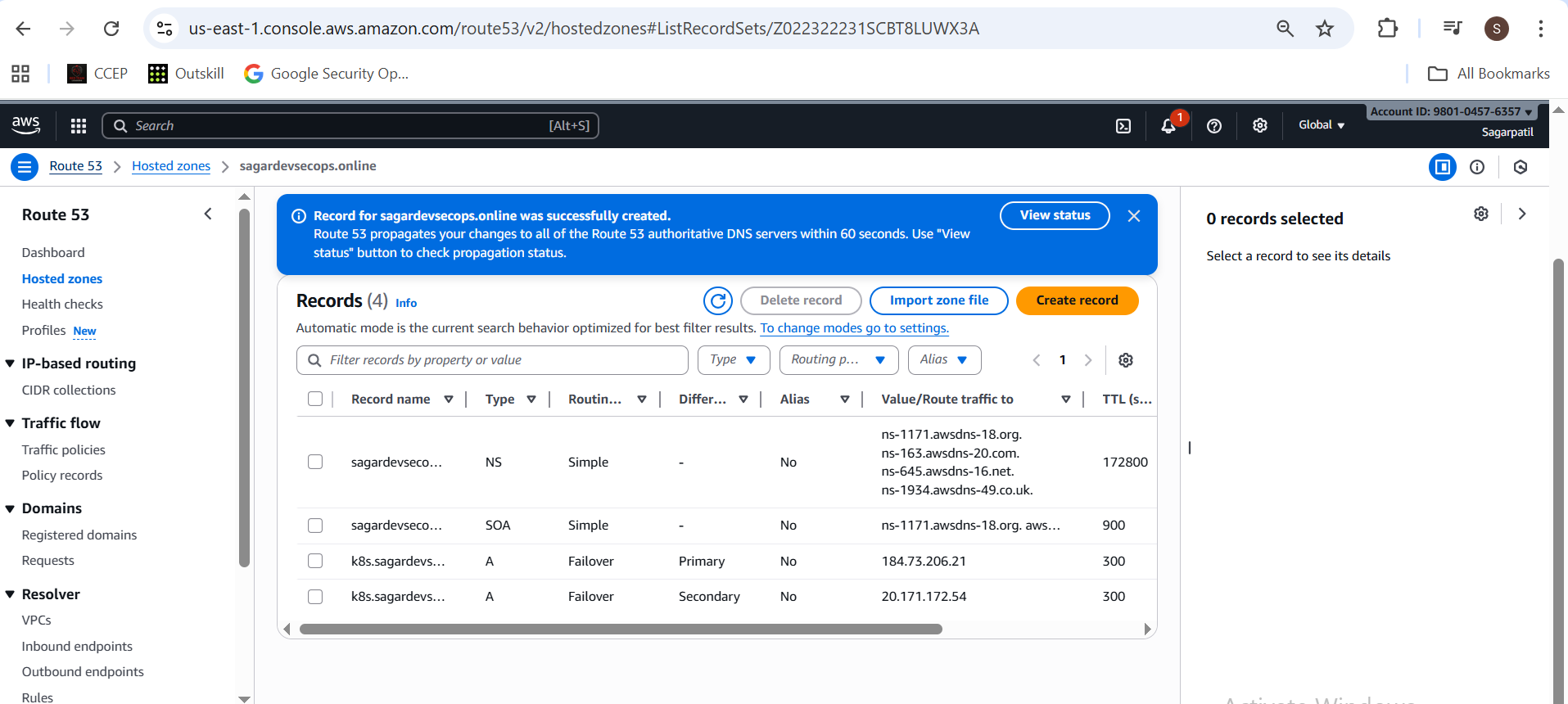
Update **k8s.sagardevsecops.online** failover records:

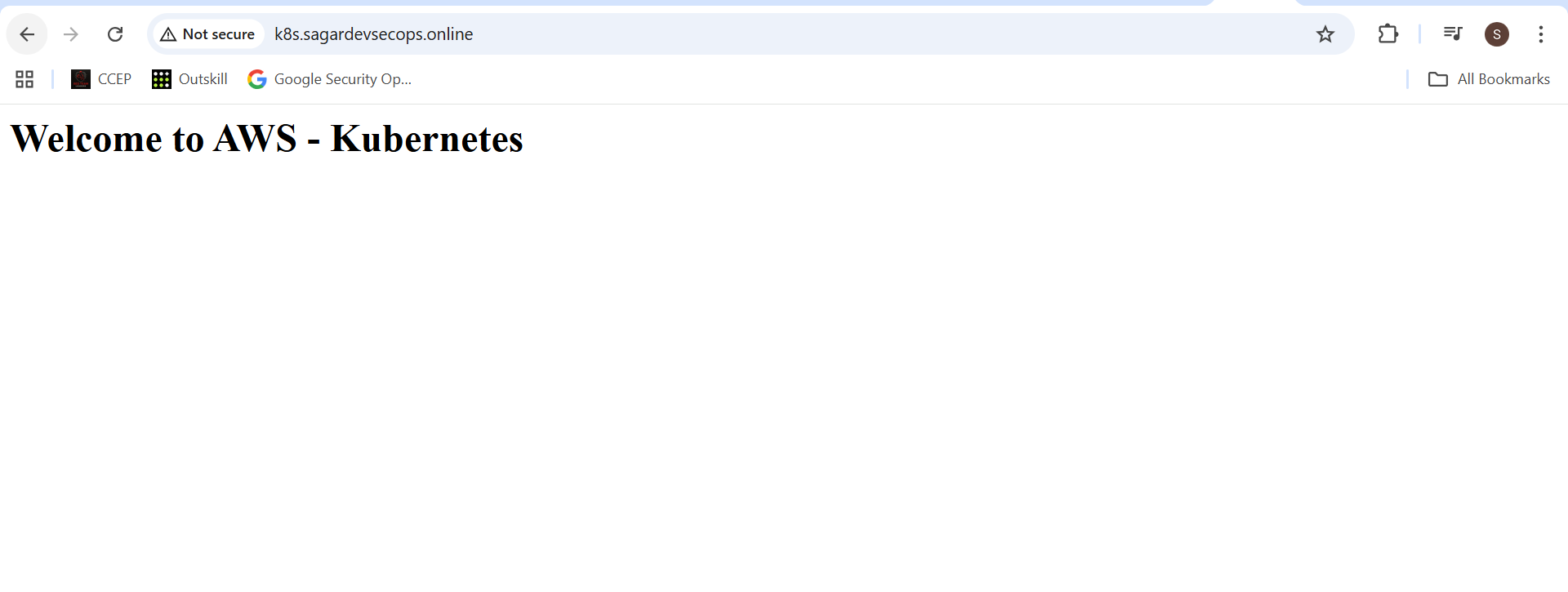
* Primary A → EKS LB IP
* Secondary A → AKS LB IP

(**Health check on EKS LB**.)

**Screenshots:**

* Route 53 records with LB IPs.





**B.5 Simulate EKS failure**

On **EKS:**

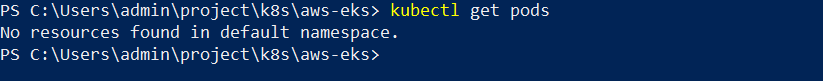
kubectl scale deploy nginx-k8s --replicas=0

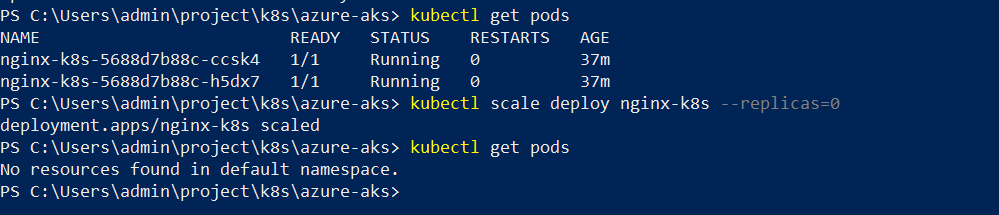
kubectl get pods

When health check fails, k8s.sagardevsecops.online should serve Azure K8s page.

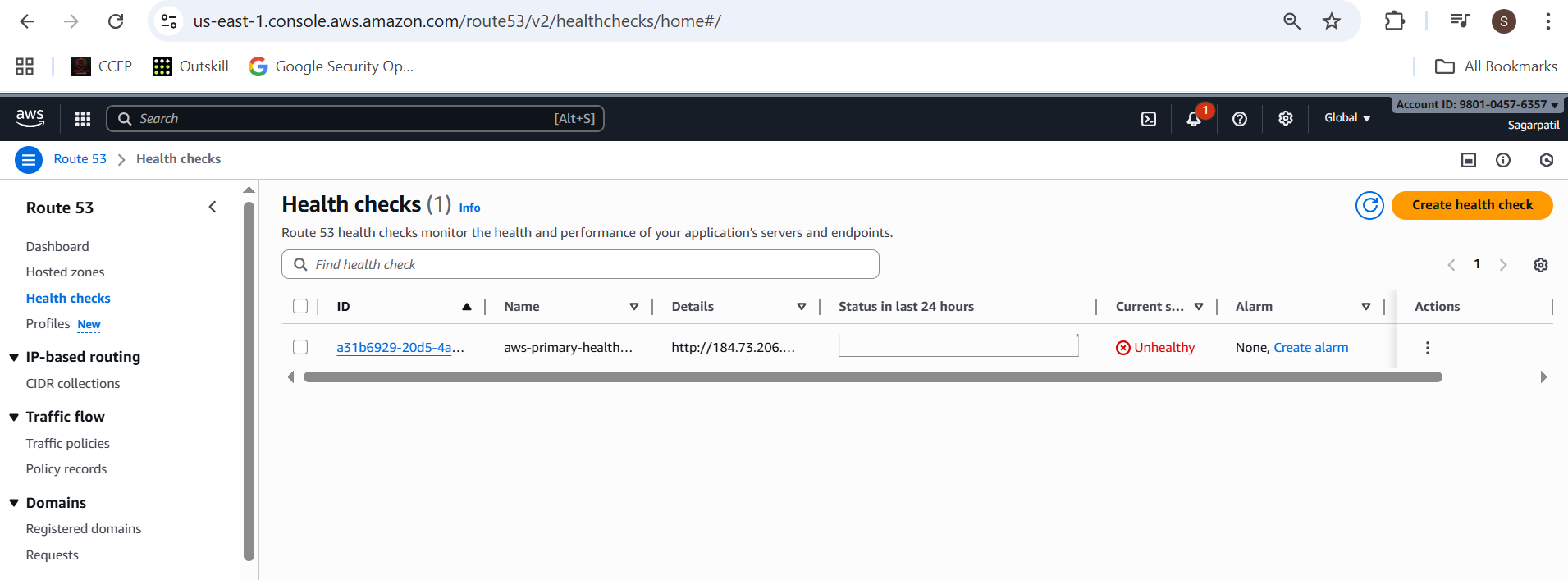
**Screenshots:**

* **kubectl get pods** showing 0 pods.

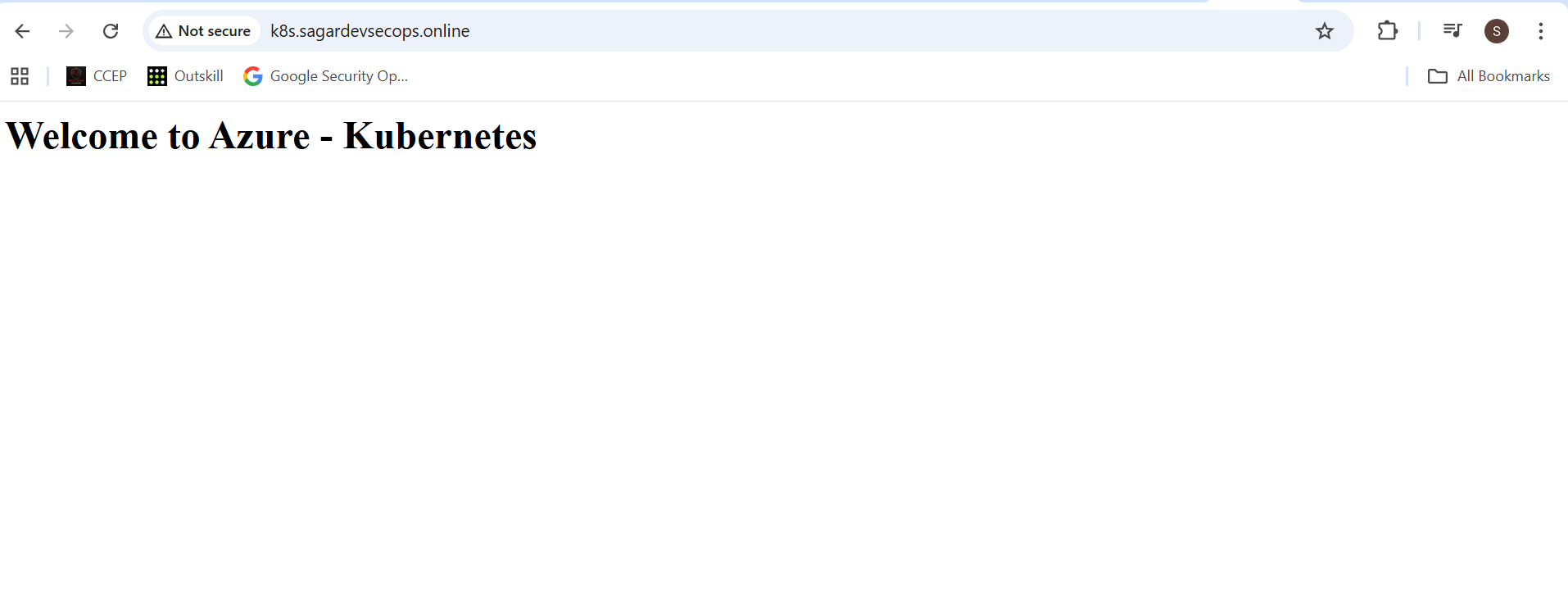




* Health check unhealthy.

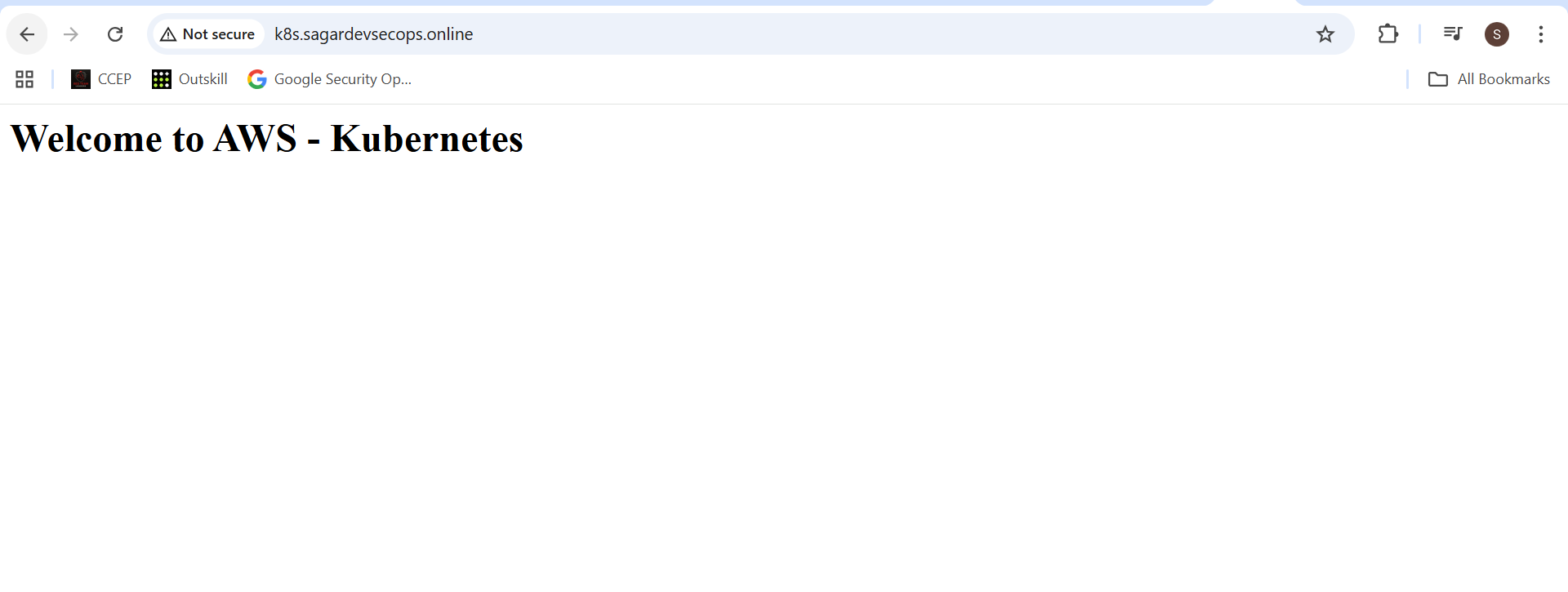


* Browser showing Azure K8s page via **app.sagardevsecops.online**.

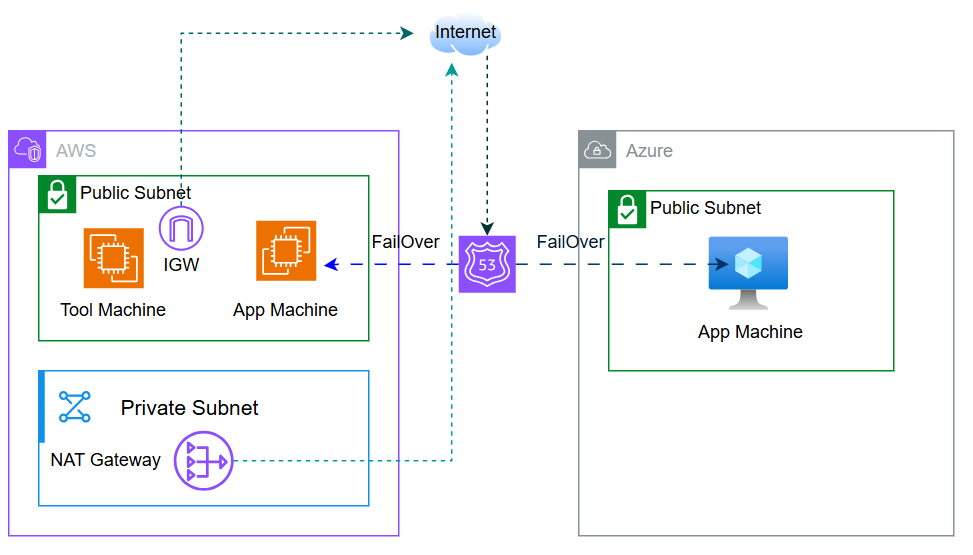


Scale back:

**kubectl scale deploy nginx-k8s --replicas=2**



**Multi-Cloud DR Project Architecture Diagram :**



* **Multi-Cloud DR Project – Total Cost to Company (TCO) Report :**

| **Cloud Provider** | **Resource** | **Qty** | **Estimate (US$/mo)** | **Estimate (INR/mo)** |
| --- | --- | --- | --- | --- |
| AWS | App EC2 t3.micro | 1 | 8.5 | 680 |
| AWS | Tools EC2 t3.micro | 1 | 8.5 | 680 |
| AWS | NAT Gateway | 1 | 32 | 2,560 |
| AWS | Route 53 hosted zone + health check | 1 | 1 | 80 |
| **AWS Total** |  |  | **≈ 50** | **≈ 3,998** |
| Azure | VM Standard\_B2ms | 1 | 70 | 5,600 |
| Azure | Public IP (Standard SKU) | 1 | 3 | 240 |
| Azure | OS disk + storage | 1 | 5 | 400 |
| **Azure Total** |  |  | **≈ 78** | **≈ 6,240** |
| **Grand Total** | Combined AWS + Azure |  | **≈ US$ 128** | **≈ ₹10,250** |