The Bombay Salesian Society's DON BOSCO INSTITUTE OF TECHNOLOGY

(An Autonomous Institute affiliated to University of Mumbai)

Department of Information Technology

Experiment 1 : Introduction to Devops on Cloud

Aim:- To understand the basic concepts of DevOps and its implementation on cloud platforms.

Introduction

DevOps is a software development methodology that combines development (Dev) and operations (Ops) to improve collaboration, automate processes, and ensure continuous delivery of applications. Implementing DevOps on the cloud provides benefits like scalability, high availability, faster deployment, and reduced infrastructure costs.

Cloud platforms such as AWS, Azure, and Google Cloud offer integrated services for DevOps, enabling developers to automate building, testing, and deployment workflows. Tools like AWS CodeBuild, CodeDeploy, and CodePipeline facilitate CI/CD (Continuous Integration/Continuous Deployment) pipelines on the cloud.

Theory:

1. DevOps Overview
   - Combines development and operations to shorten development cycles, increase deployment frequency, and deliver stable software releases.
   - CI/CD Pipelines automate the process from code commit to production deployment.
2. Cloud Computing and Service Providers
   - Cloud computing provides on-demand infrastructure and services over the internet.
   - Service Providers: AWS, Microsoft Azure, Google Cloud Platform (GCP).
   - Services Used in DevOps on Cloud:
     - Compute Services: EC2 instances, Lambda (serverless computing)
     - Storage Services: S3 buckets for artifacts and application storage
     - Database Services: RDS, DynamoDB
3. AWS DevOps Services
   - CodeBuild: Automates building and testing code.
   - CodeDeploy: Automates deployment of applications on EC2 instances or on-premises servers.
   - CodePipeline: Automates the CI/CD workflow from code commit to deployment.

4. High Availability and Auto-Scaling

- High Availability: Using multiple instances across availability zones ensures no single point of failure.
- Auto-Scaling: Adjusts computing resources dynamically according to the traffic load.

Procedure :

Step 1: Log in to AWS

1. Open the AWS Management Console.
2. Navigate to the EC2 service under “Compute”.

Step 2: Launch a New Instance

1. Click Launch Instance.
2. Provide a Name for your instance (e.g., DevOps-Lab-Instance).
3. Choose an Amazon Machine Image (AMI):
    - Select Amazon Linux 2 (or Ubuntu) for a simple Linux server.
4. Choose an Instance Type:
    - For lab purposes, select t3.micro (eligible for free tier).

Step 3: Configure Instance

1. Number of Instances: Enter 1 (or more if needed for high availability).
2. Network and Subnet: Choose your default VPC and select an availability zone.
3. Auto-assign Public IP: Enable (so you can access it via SSH or web).
4. Leave other options default for a simple lab setup.

Step 4: Add Storage

1. By default, an 8 GB EBS volume is attached.
2. You can increase size if needed for your application.

Step 5: Configure Security Group

1. Either create a new security group or select an existing one.
2. Add rules for:
    - SSH (port 22) – to connect via terminal.
    - HTTP (port 80) – if hosting a web application.
    - HTTPS (port 443) – if needed.

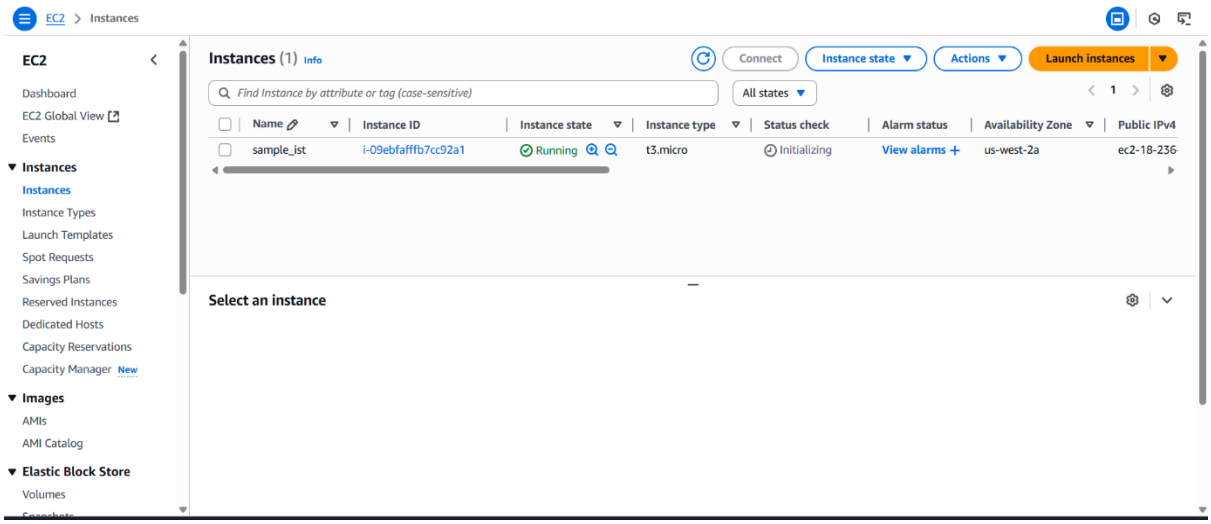
Step 6: Key Pair

1. Create a new key pair (choose RSA or Ed25519).
2. Download the .pem file securely – this is needed to access your instance.

Step 7: Launch Instance

1. Click Launch Instance.
2. Wait for the instance status to show running.

Output



Conclusion:

- DevOps on cloud platforms simplifies software development and deployment.
- High availability and auto-scaling ensure fault tolerance and scalability.
- This lab demonstrates real-time deployment, scalability, and automation using cloud services, preparing for real-world DevOps implementations.

The Bombay Salesian Society's Don Bosco Institute of Technology
(An Autonomous Institute affiliated to University of Mumbai)

Department of Information Technology

EXP 2: Container orchestration using Kubernetes

Title:
Setting up and Deploying Applications on Kubernetes using Minikube

Objective:
To understand the setup of a Kubernetes environment using Minikube and perform basic operations such as deploying a pod and an Nginx deployment using kubectl commands.

Procedure:
- **1. **Download and Install kubectl****

  Open PowerShell and run the following command to download the Kubernetes CLI tool (kubectl):
  ```
  curl.exe -LO "https://dl.k8s.io/release/v1.33.0/bin/windows/amd64/kubectl.exe"
  ```
  Verify the installation:
  ```
  kubectl version --client
  ```
- **2. **Download kubectl-convert Plugin****

  This plugin allows conversion between different Kubernetes API versions.
  ```
  curl.exe -LO "https://dl.k8s.io/release/v1.33.0/bin/windows/amd64/kubectl-convert.exe"
  ```
  Check if the plugin is working:
  ```
  kubectl convert --help
  ```
- **3. **Create Minikube Directory****

  ```
  New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
  ```
- **4. **Download Minikube Executable****

```
$ProgressPreference = 'SilentlyContinue'
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri
'https://github.com/kubernetes/minikube/releases/latest/download/minikube-
windowsamd64.exe' -UseBasicParsing
```

- **5. **Add Minikube to System PATH****

  Run the following command as Administrator:
  ```
  $oldPath = [Environment]::GetEnvironmentVariable('Path',
[EnvironmentVariableTarget]::Machine)
  if ($oldPath.Split(';') -inotcontains 'C:\minikube'){
    [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath),
[EnvironmentVariableTarget]::Machine)
  }
  ```

- **6. **Start Minikube Cluster****

  If Docker driver is used, run as Administrator:
  ```
  minikube delete
  minikube start --driver=docker
  ```

- **7. **Verify Minikube Cluster****

  Check the running nodes:
  ```
  kubectl get nodes
  ```

- **8. **Deploy Example Pod****

  ```
  kubectl apply -f https://k8s.io/examples/pods/simple-pod.yaml
kubectl get pods
  ```

- **9. **Deploy Nginx Deployment****

  ```
  kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml
  kubectl get pods
  ```

- **10. **Delete a Pod (Optional)****

```
kubectl delete pod <pod-name>
kubectl get pods
```

Output:

1) Download kubectl

PS C:\Users\Yashraj Patil> curl.exe -LO
"https://dl.k8s.io/release/v1.33.0/bin/windows/amd64/kubectl.exe"
100 58.8M 100 58.8M   0   0   529k   0 0:01:53 0:01:53 --:--:-- 429k

2) Verify kubectl Version

PS C:\Users\Yashraj Patil> kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0

3) Download kubectl-convert Plugin

PS C:\Users\Yashraj Patil> curl.exe -LO
"https://dl.k8s.io/release/v1.33.0/bin/windows/amd64/kubectl-convert.exe"
100 57.7M 100 57.7M   0   0 1262k   0 0:00:46 0:00:46 --:--:-- 1609k
PS C:\Users\Yashraj Patil> kubectl convert --help
Convert config files between different API versions... (help message displayed successfully)

4) Create Minikube Directory

PS C:\Users\Yashraj Patil> New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
Directory created successfully at C:\minikube

5) Download Minikube Executable

PS C:\Users\Yashraj Patil> Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri
'https://github.com/kubernetes/minikube/releases/latest/download/minikube-windows-
amd64.exe' -UseBasicParsing

6) Add Minikube to PATH

Executed administrator PowerShell command to permanently add C:\minikube to the system PATH.

7) Start Minikube

PS C:\Windows\system32> minikube start *
Automatically selected the hyperv driver.
X Exiting due to PR_HYPERV_MODULE_NOT_INSTALLED: Hyper-V PowerShell Module not available.

Suggestion: Run: 'Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-VTools-All -All'

8) Restart Minikube with Docker Driver

PS C:\Windows\system32> minikube start --driver=docker
* Using Docker Desktop driver with root privileges
* Kubernetes v1.33.1 configured successfully
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use 'minikube' cluster

9) Verify Node Status

PS C:\Users\Yashraj Patil> kubectl get nodes NAME
STATUS ROLES AGE VERSION minikube
Ready control-plane 2m33s v1.33.1

10) Deploy Example Pod

PS C:\Users\Yashraj Patil> kubectl apply -f https://k8s.io/examples/pods/simple-pod.yaml
pod/nginx created
PS C:\Users\Yashraj Patil> kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx 0/1 ContainerCreating 0 21s

11) Deploy Nginx Deployment

PS C:\Users\Yashraj Patil> kubectl apply -f
https://k8s.io/examples/controllers/nginxdeployment.yaml deployment.apps/nginx-deployment created PS C:\Users\Yashraj Patil> kubectl get pods
nginx 0/1 ContainerCreating 0 93s nginx-deployment-647677fc66-p7rwx 0/1 ContainerCreating 0 7s nginx-deployment-647677fc66-pcfxb 0/1 ContainerCreating 0 7s nginx-deployment-647677fc66-pcw5r 0/1 ContainerCreating 0 7s

12) Delete a Pod and Verify

PS C:\Users\Yashraj Patil> kubectl delete pod nginx-deployment-647677fc66-p7rwx Pod deleted successfully.
PS C:\Users\Yashraj Patil> kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx 1/1 Running 0 5m53s
nginx-deployment-647677fc66-pcfxb 1/1 Running 0 4m27s nginx-deployment-647677fc66-pcw5r 1/1 Running 0 4m27s nginx-deployment-647677fc66-wwrwb 1/1 Running 0 45s

Conclusion:

Through this practical, we learned how to set up a local Kubernetes cluster using Minikube, manage it with kubectl commands, and deploy sample pods and deployments. This forms the foundation for deploying and managing containerized applications in a Kubernetes environment.

EXP 3: Infrastructure Automation using Terraform on AWS

Name: Parth SHikhare
Roll No: 53

Aim: - To understand and implement Infrastructure as Code (IaC) using Terraform for provisioning and destroying cloud infrastructure resources (EC2 instance) on AWS.

Steps:

1. **Install Terraform** ○ Download and install Terraform on the local system. ○ Verify installation using:
   `terraform -version`

2. **Configure AWS CLI** ○ Install AWS CLI and configure it using IAM user credentials: `aws configure` ○ Provide the Access Key, Secret Key, region, and output format.

3. **Create a Terraform Configuration File** ○ Create a file named `main.tf` containing the resource definition for an AWS EC2 instance:

```
provider "aws" {
region = "ap-south-1"
}  resource "aws_instance" "example" {
ami           = "ami-0c02fb55956c7d316"
instance_type = "t2.micro"
}
```

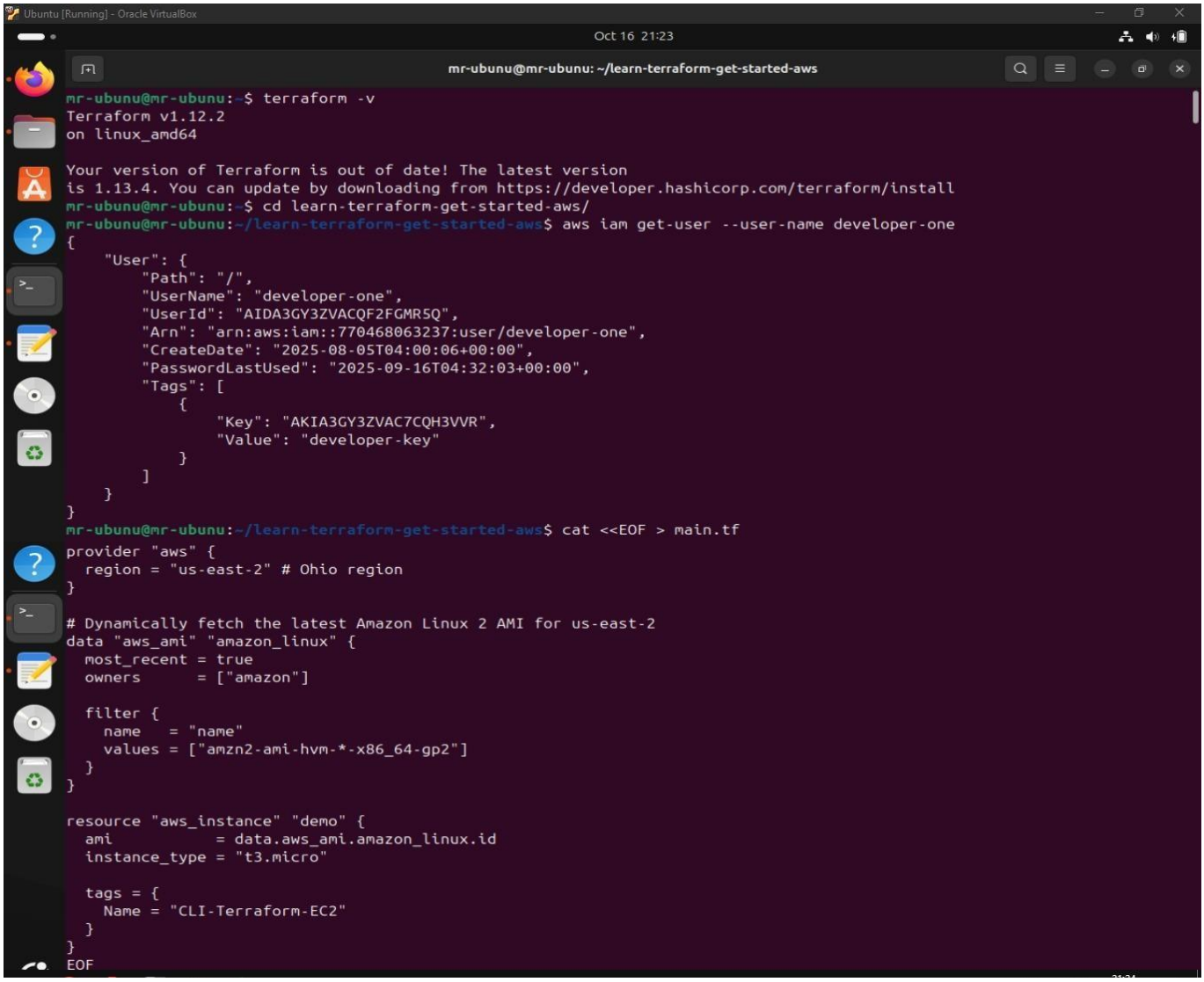
4. **Initialize Terraform** ○ Initialize the working directory containing Terraform configuration:
   `terraform init`
5. **Validate Configuration** ○ Validate the Terraform configuration to check for syntax or logical errors:
   `terraform validate`
6. **Create an Execution Plan** ○ Review what Terraform will create or modify:
   `terraform plan`

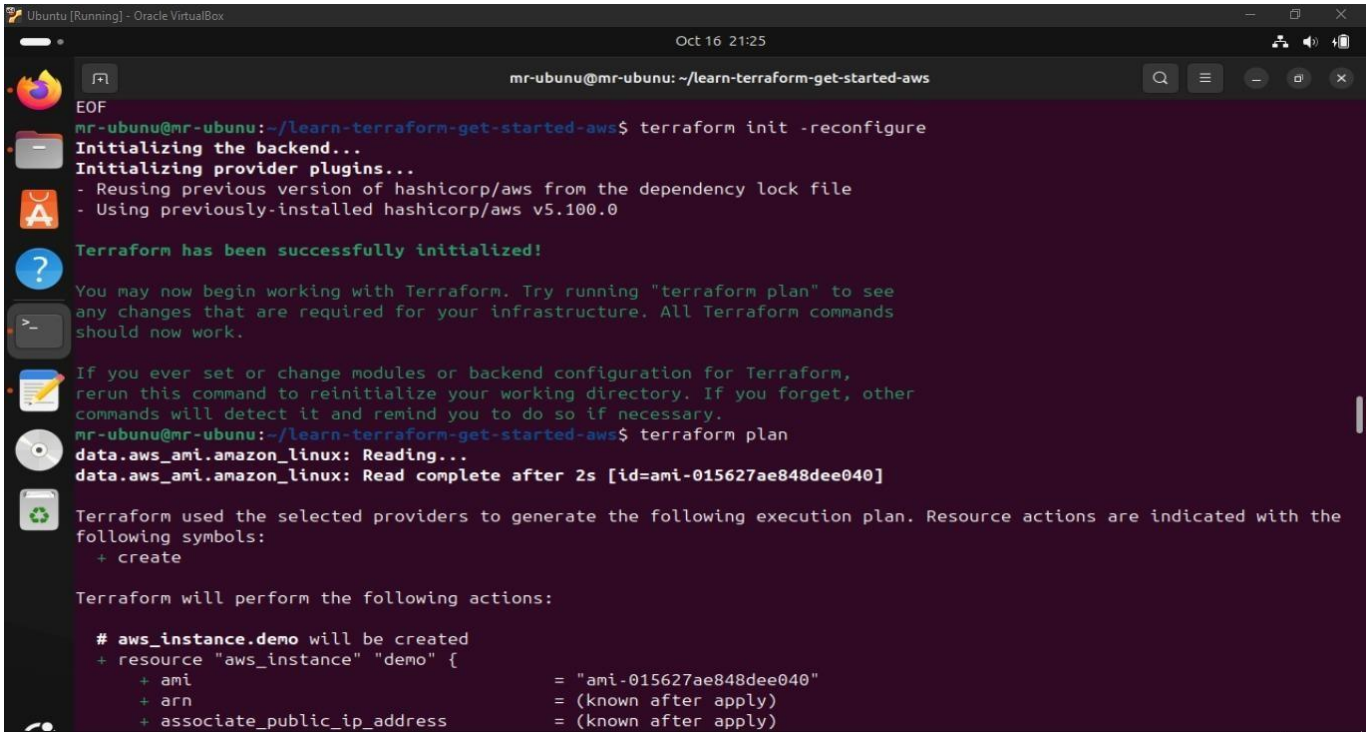
7. **Apply Configuration to Create EC2 Instance** ○ Deploy the infrastructure: `terraform apply` ○ Confirm by typing "yes" when prompted.

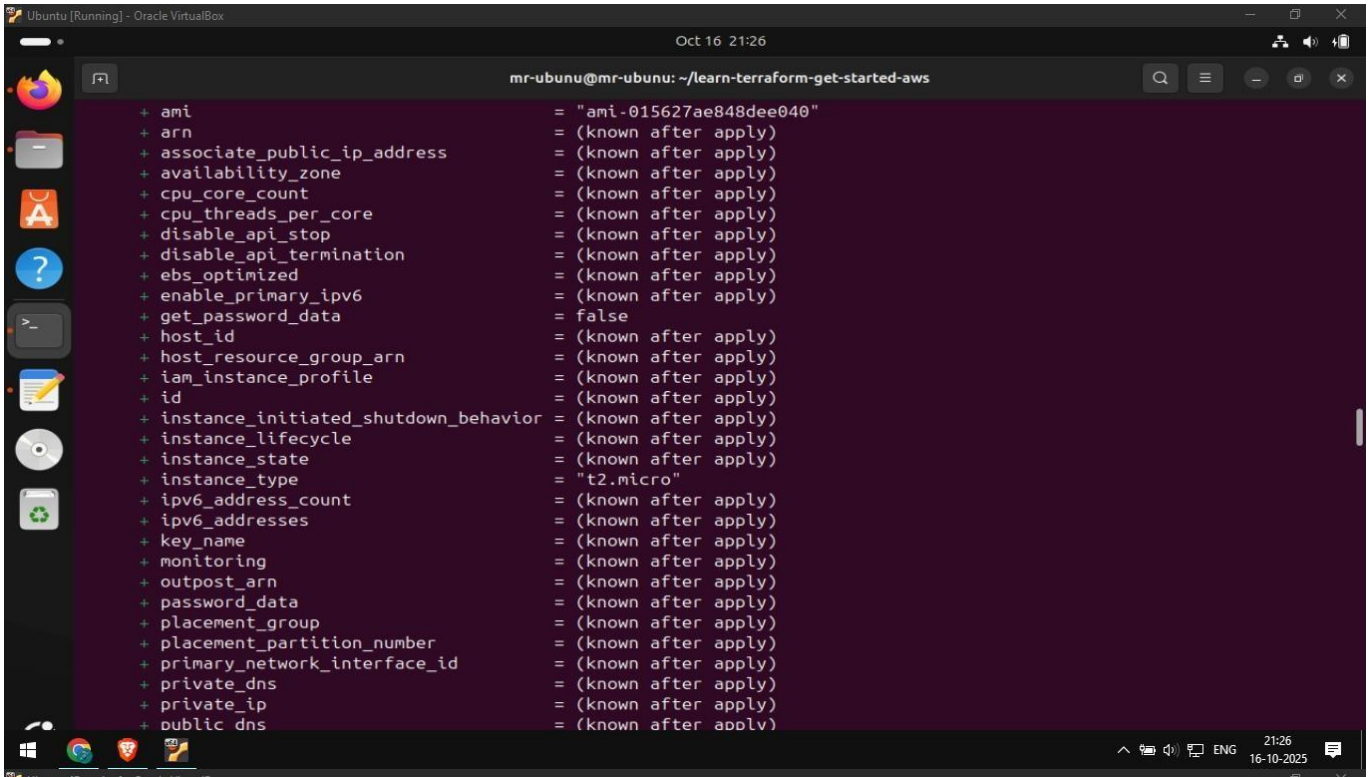
8. **Verify the Instance** ○ Check AWS Management Console → EC2 → Instances to confirm successful creation.

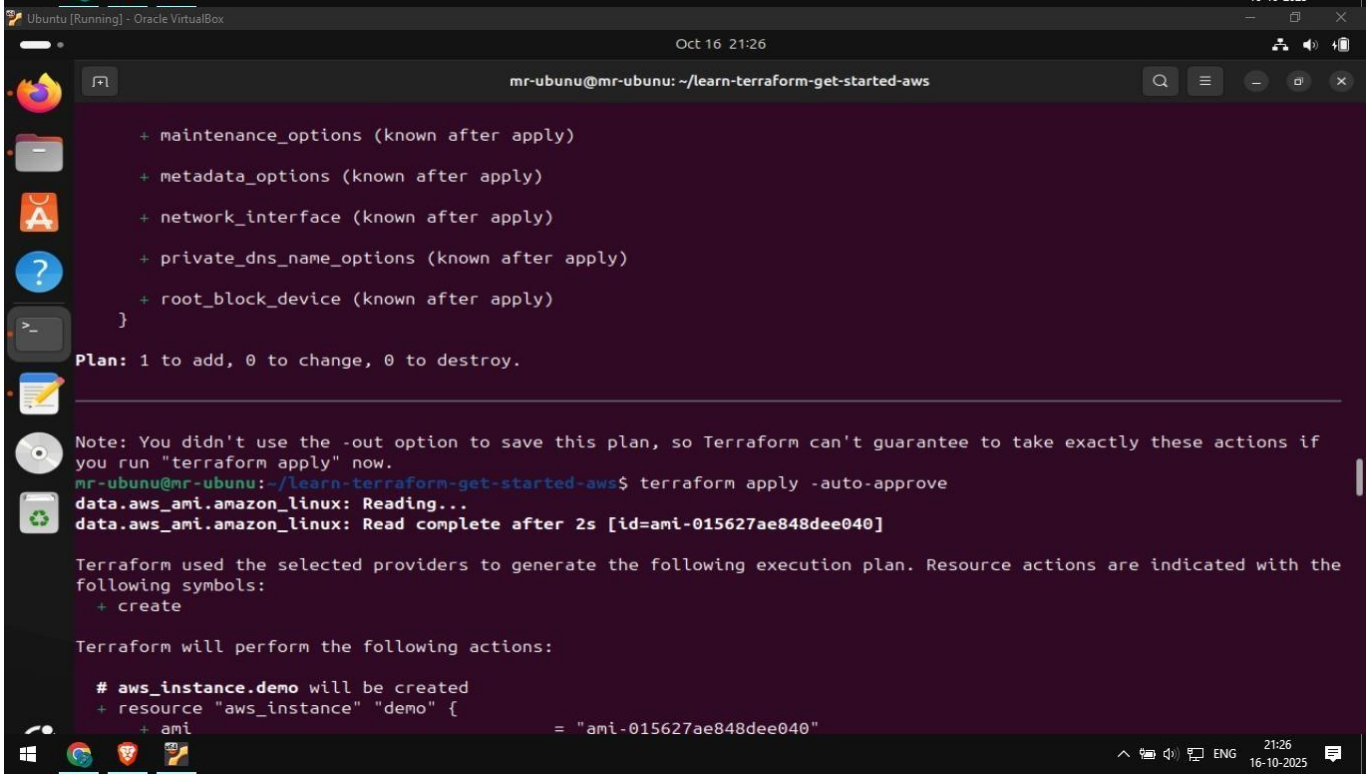
9. **Destroy the Infrastructure** ○ Remove all created resources:
   `terraform destroy`
   ○ Confirm with "yes" when prompted.

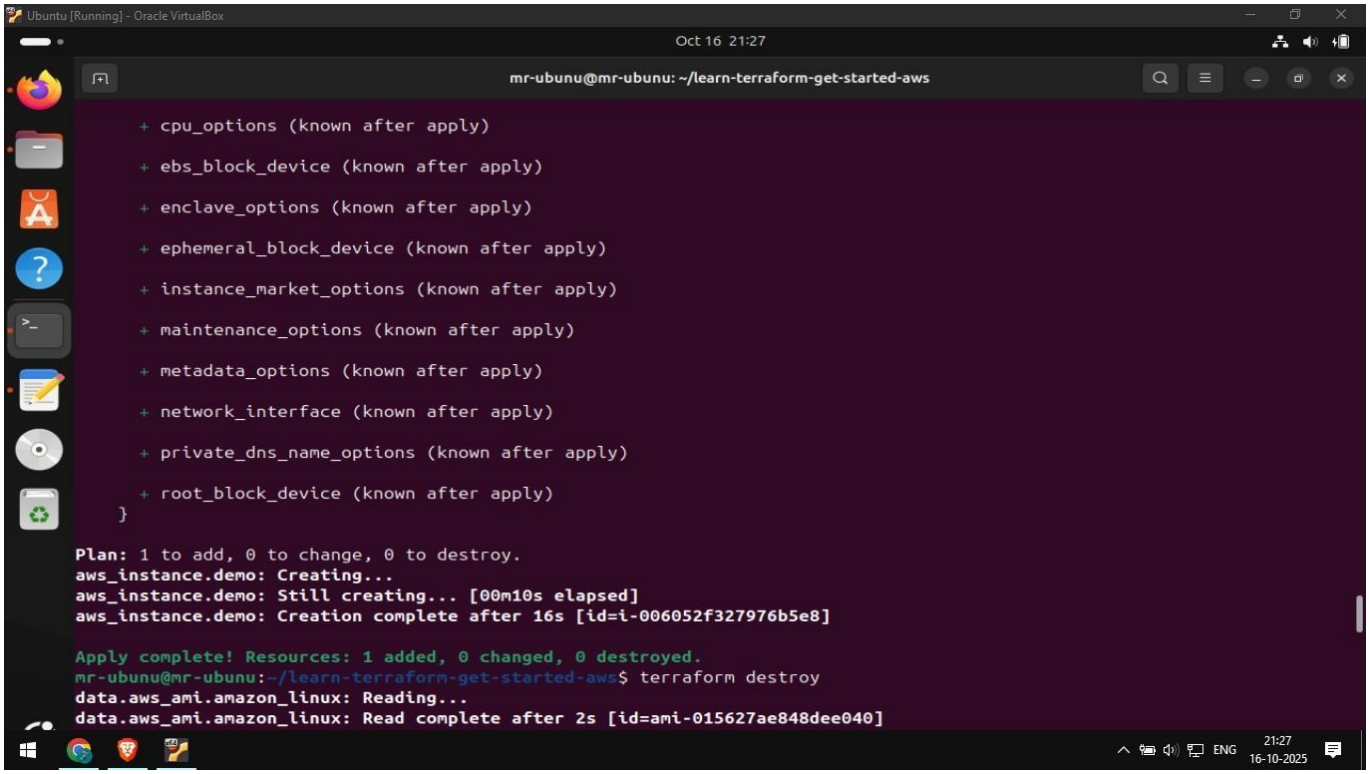
Output:


Ubuntu [Running] - Oracle VirtualBox
Oct 16 21:23
mr-ubunu@mr-ubunu: ~/learn-terraform-get-started-aws
mr-ubunu@mr-ubunu:~$ terraform -v
Terraform v1.12.2
on linux_amd64
Your version of Terraform is out of date! The latest version
is 1.13.4. You can update by downloading from https://developer.hashicorp.com/terraform/install
mr-ubunu@mr-ubunu:~$ cd learn-terraform-get-started-aws/
mr-ubunu@mr-ubunu:~/learn-terraform-get-started-aws$ aws iam get-user --user-name developer-one
{
"User": {
"Path": "/",
"UserName": "developer-one",
"UserId": "AIDA3GY3ZVACQF2FGMR5Q",
"Arn": "arn:aws:iam::770468063237:user/developer-one",
"CreateDate": "2025-08-05T04:00:06+00:00",
"PasswordLastUsed": "2025-09-16T04:32:03+00:00",
"Tags": [
{
"Key": "AKIA3GY3ZVAC7CQH3VVR",
"Value": "developer-key"
}
]
}
}
mr-ubunu@mr-ubunu:~/learn-terraform-get-started-aws$ cat <<EOF > main.tf
provider "aws" {
region = "us-east-2" # Ohio region
}
# Dynamically fetch the latest Amazon Linux 2 AMI for us-east-2
data "aws_ami" "amazon_linux" {
most_recent = true
owners = ["amazon"]
filter {
name = "name"
values = ["amzn2-ami-hvm-*-x86_64-gp2"]
}
}
resource "aws_instance" "demo" {
ami = data.aws_ami.amazon_linux.id
instance_type = "t3.micro"
tags = {
Name = "CLI-Terraform-EC2"
}
}
EOF

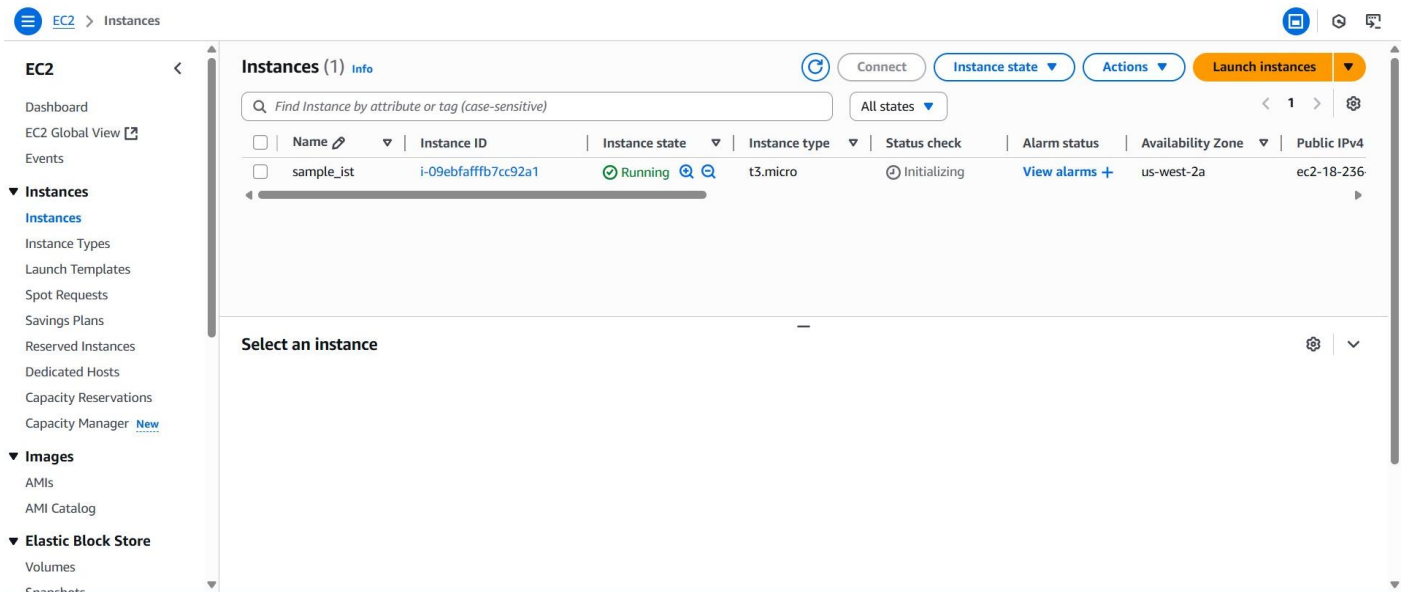

Ubuntu [Running] - Oracle VirtualBox
Oct 16 21:25
mr-ubunu@mr-ubunu: ~/learn-terraform-get-started-aws
EOF
mr-ubunu@mr-ubunu:~/learn-terraform-get-started-aws$ terraform init -reconfigure
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.100.0
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
mr-ubunu@mr-ubunu:~/learn-terraform-get-started-aws$ terraform plan
data.aws_ami.amazon_linux: Reading...
data.aws_ami.amazon_linux: Read complete after 2s [id=ami-015627ae848dee040]
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
+ create
Terraform will perform the following actions:
# aws_instance.demo will be created
+ resource "aws_instance" "demo" {
+ ami = "ami-015627ae848dee040"
+ arn = (known after apply)
+ associate_public_ip_address = (known after apply)

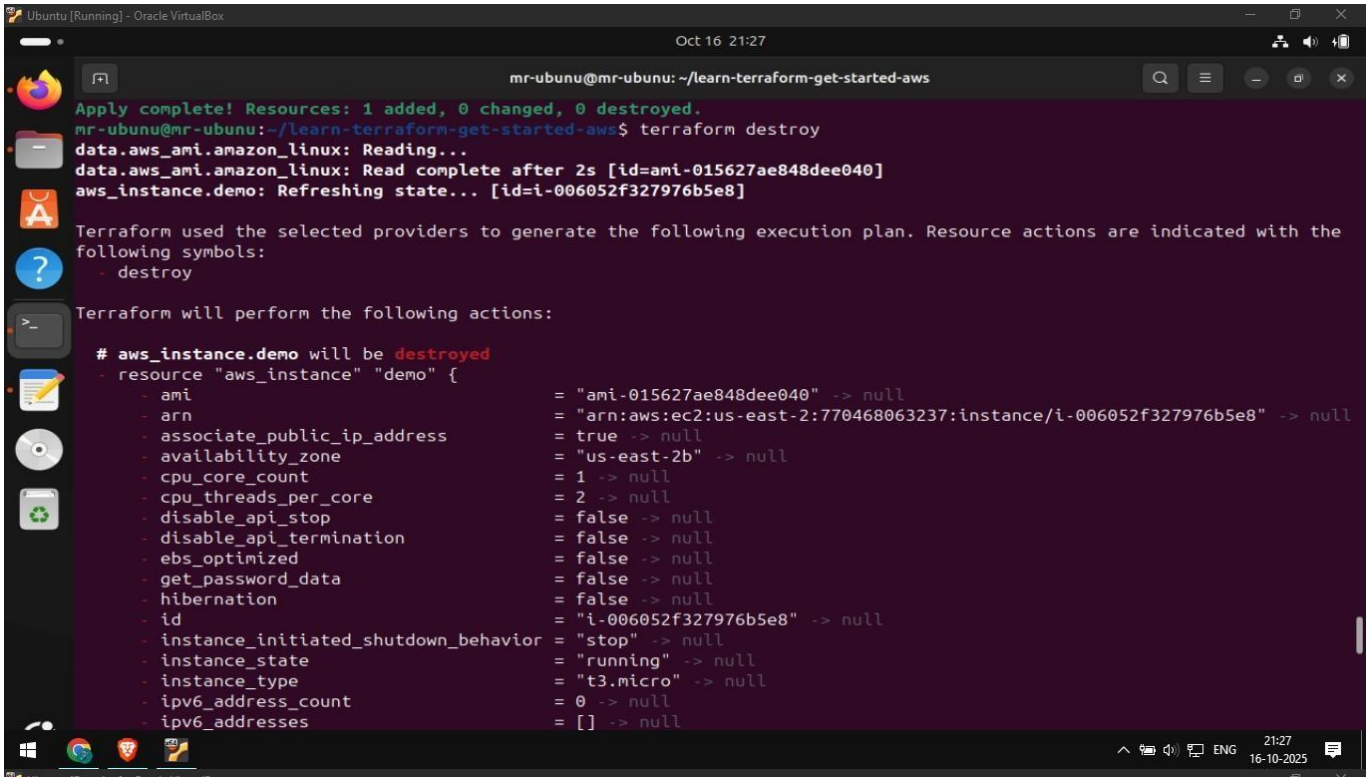

Ubuntu [Running] - Oracle VirtualBox
Oct 16 21:26
mr-ubunu@mr-ubunu: ~/learn-terraform-get-started-aws
+ ami = "ami-015627ae848dee040"
+ arn = (known after apply)
+ associate_public_ip_address = (known after apply)
+ availability_zone = (known after apply)
+ cpu_core_count = (known after apply)
+ cpu_threads_per_core = (known after apply)
+ disable_api_stop = (known after apply)
+ disable_api_termination = (known after apply)
+ ebs_optimized = (known after apply)
+ enable_primary_ipv6 = (known after apply)
+ get_password_data = false
+ host_id = (known after apply)
+ host_resource_group_arn = (known after apply)
+ iam_instance_profile = (known after apply)
+ id = (known after apply)
+ instance_initiated_shutdown_behavior = (known after apply)
+ instance_lifecycle = (known after apply)
+ instance_state = (known after apply)
+ instance_type = "t2.micro"
+ ipv6_address_count = (known after apply)
+ ipv6_addresses = (known after apply)
+ key_name = (known after apply)
+ monitoring = (known after apply)
+ outpost_arn = (known after apply)
+ password_data = (known after apply)
+ placement_group = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns = (known after apply)
+ private_ip = (known after apply)
+ public_dns = (known after apply)
ENG 21:26 16-10-2025

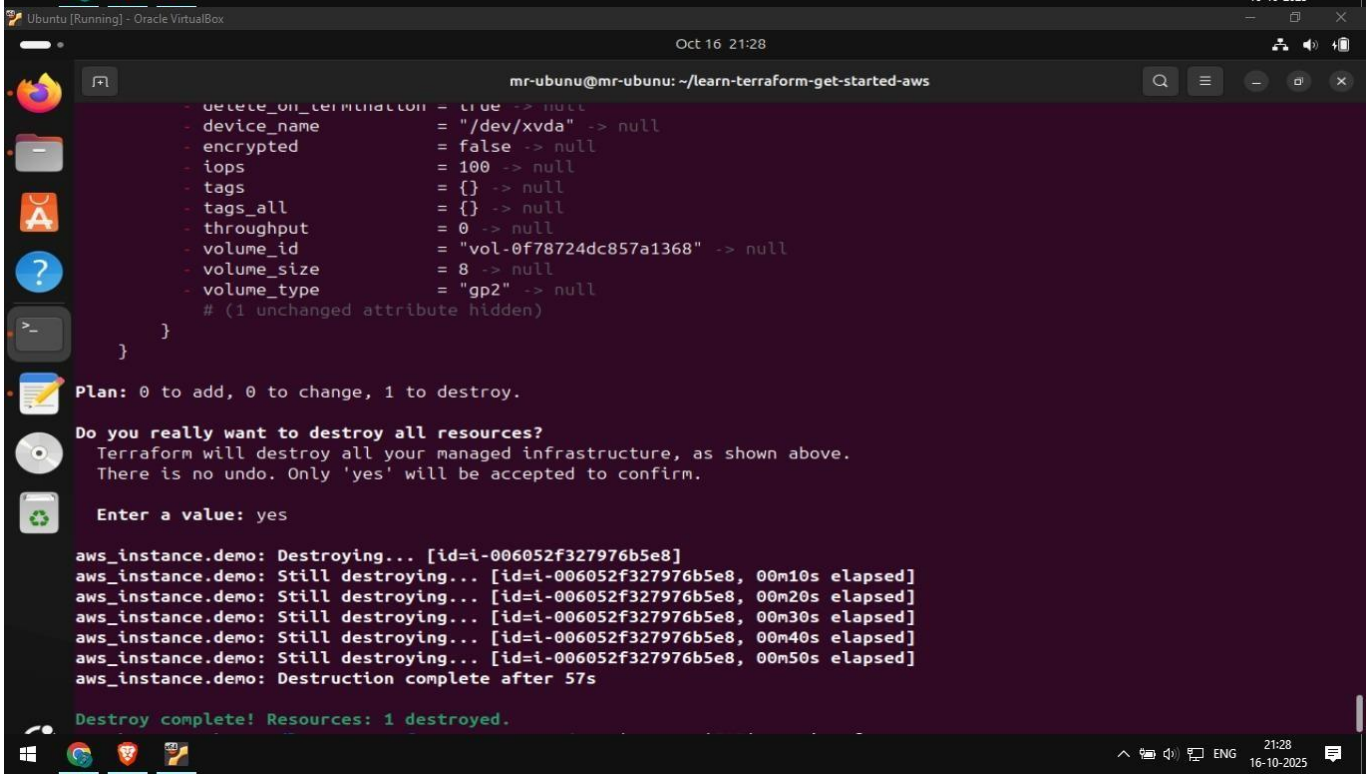

Ubuntu [Running] - Oracle VirtualBox
Oct 16 21:26
mr-ubunu@mr-ubunu: ~/learn-terraform-get-started-aws
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}
Plan: 1 to add, 0 to change, 0 to destroy.
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
mr-ubunu@mr-ubunu:~/learn-terraform-get-started-aws$ terraform apply -auto-approve
data.aws_ami.amazon_linux: Reading...
data.aws_ami.amazon_linux: Read complete after 2s [id=ami-015627ae848dee040]
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create
Terraform will perform the following actions:
# aws_instance.demo will be created
+ resource "aws_instance" "demo" {
+ ami = "ami-015627ae848dee040"
ENG 21:26 16-10-2025

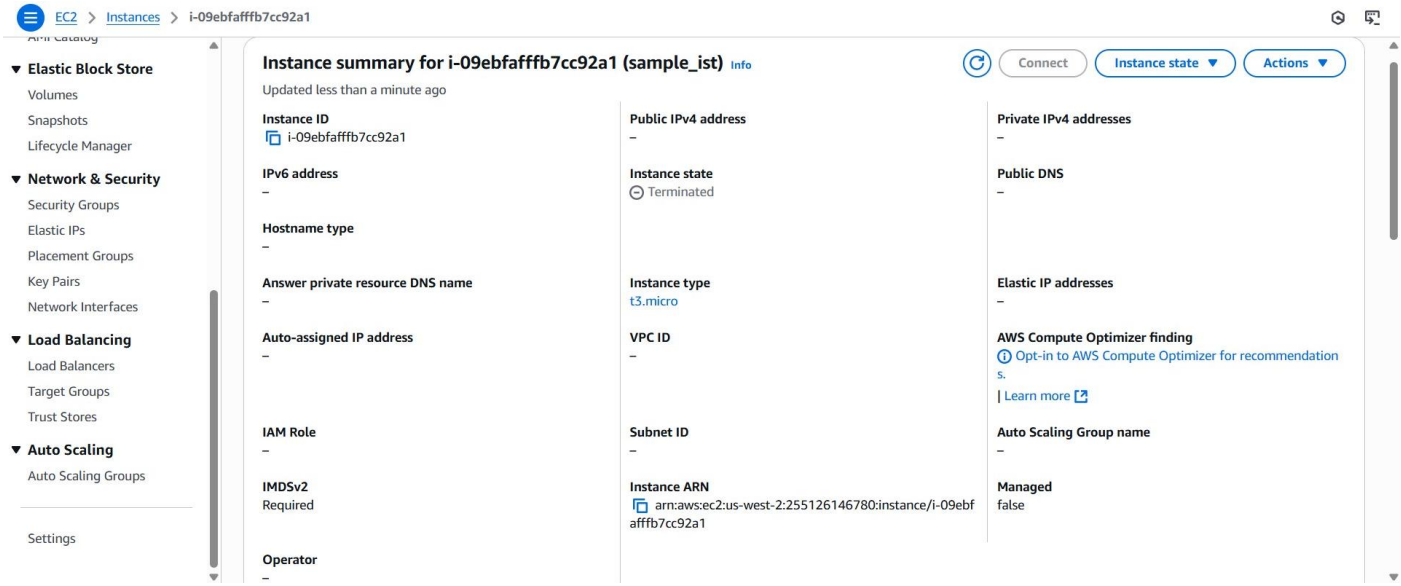


Instance is created and is visible on AWS Web Interface




Ubuntu [Running] - Oracle VirtualBox
Oct 16 21:27
mr-ubunu@mr-ubunu: ~/learn-terraform-get-started-aws
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
mr-ubunu@mr-ubunu:~/learn-terraform-get-started-aws$ terraform destroy
data.aws_ami.amazon_linux: Reading...
data.aws_ami.amazon_linux: Read complete after 2s [id=ami-015627ae848dee040]
aws_instance.demo: Refreshing state... [id=i-006052f327976b5e8]
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- destroy
Terraform will perform the following actions:
# aws_instance.demo will be destroyed
- resource "aws_instance" "demo" {
- ami = "ami-015627ae848dee040" -> null
- arn = "arn:aws:ec2:us-east-2:770468063237:instance/i-006052f327976b5e8" -> null
- associate_public_ip_address = true -> null
- availability_zone = "us-east-2b" -> null
- cpu_core_count = 1 -> null
- cpu_threads_per_core = 2 -> null
- disable_api_stop = false -> null
- disable_api_termination = false -> null
- ebs_optimized = false -> null
- get_password_data = false -> null
- hibernation = false -> null
- id = "i-006052f327976b5e8" -> null
- instance_initiated_shutdown_behavior = "stop" -> null
- instance_state = "running" -> null
- instance_type = "t3.micro" -> null
- ipv6_address_count = 0 -> null
- ipv6_addresses = [] -> null
ENG 21:27 16-10-2025


Ubuntu [Running] - Oracle VirtualBox
Oct 16 21:28
mr-ubunu@mr-ubunu: ~/learn-terraform-get-started-aws
- device_name = "/dev/xvda" -> null
- encrypted = false -> null
- iops = 100 -> null
- tags = {} -> null
- tags_all = {} -> null
- throughput = 0 -> null
- volume_id = "vol-0f78724dc857a1368" -> null
- volume_size = 8 -> null
- volume_type = "gp2" -> null
# (1 unchanged attribute hidden)
}
}
Plan: 0 to add, 0 to change, 1 to destroy.
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value: yes
aws_instance.demo: Destroying... [id=i-006052f327976b5e8]
aws_instance.demo: Still destroying... [id=i-006052f327976b5e8, 00m10s elapsed]
aws_instance.demo: Still destroying... [id=i-006052f327976b5e8, 00m20s elapsed]
aws_instance.demo: Still destroying... [id=i-006052f327976b5e8, 00m30s elapsed]
aws_instance.demo: Still destroying... [id=i-006052f327976b5e8, 00m40s elapsed]
aws_instance.demo: Still destroying... [id=i-006052f327976b5e8, 00m50s elapsed]
aws_instance.demo: Destruction complete after 57s
Destroy complete! Resources: 1 destroyed.
ENG 21:28 16-10-2025



Conclusion:

In this practical, Terraform was successfully used to automate the creation and destruction of an AWS EC2 instance using Infrastructure as Code (IaC). The process demonstrated how Terraform simplifies cloud resource management, ensures consistency, and enhances automation by defining infrastructure declaratively.

The Bombay Salesian Society's
DON BOSCO INSTITUTE OF TECHNOLOGY
(An Autonomous Institute affiliated to University of Mumbai)
Department of Information Technology

Experiment 4 : Application Vulnerability Assessment using SonarQube and Jenkins

Aim:-
To identify and remediate application vulnerabilities early in the development process by performing static code analysis using SonarQube and integrating it with Jenkins for automated security checks.

Introduction
In the modern software development lifecycle, ensuring application security from the early stages is crucial. Static Application Security Testing (SAST) tools such as SonarQube help developers identify vulnerabilities, bugs, and code smells directly from source code before deployment. By integrating SonarQube with Jenkins, we can automate continuous inspection of code quality and security in CI/CD pipelines, thereby embedding security into the DevOps process. This experiment focuses on analyzing Java or web-based projects for potential security flaws and improving code quality metrics through automated analysis workflows.

Theory
1. Static Code Analysis
- Examines source code without executing it.
- Detects potential security issues like SQL injection, XSS, hardcoded credentials, and insecure configurations.
- Improves maintainability and reliability.

2. SonarQube Overview
- An open-source platform for continuous inspection of code quality.
- Supports multiple languages including Java, Python, JavaScript, and C++.
- Provides metrics like code coverage, duplications, vulnerabilities, and technical debt.

3. Jenkins Integration
- Jenkins is a CI/CD automation tool that integrates with SonarQube to perform automatic code scans during build pipelines.
- Each time new code is pushed, Jenkins triggers a SonarQube scan and reports results

on the dashboard.

4. Benefits of Integrating SonarQube with Jenkins
- Automated vulnerability scanning before deployment.
- Continuous feedback to developers.
- Early detection of defects reduces remediation costs.

Procedure

Step 1: Install and Configure SonarQube
- Download and install SonarQube Community Edition.

- Start the SonarQube server and log in to the dashboard (default port: 9000).

- Create a new project and generate an authentication token.

Code Snippet:
sonar-scanner -Dsonar.projectKey=myapp -Dsonar.sources=. -
Dsonar.host.url=http://localhost:9000 -Dsonar.login=<TOKEN>

Output:
SonarQube server started successfully on port 9000.
Project 'myapp' created.
Authentication token generated.

Step 2: Configure Jenkins
- Install Jenkins and required plugins (SonarQube Scanner for Jenkins, Git, Maven).

- Navigate to Manage Jenkins → Configure System → SonarQube Servers.

- Add the SonarQube server details and authentication token.

Code Snippet:
Manage Jenkins → Configure System → Add SonarQube server
Name: SonarQube
Server URL: http://localhost:9000
Authentication Token: <TOKEN>

Output:
SonarQube integration configured successfully in Jenkins.

Step 3: Create a Jenkins Job

- Create a new pipeline or freestyle project.

- Link the source code repository (GitHub, GitLab, or local repo).

- Add a build step: "Execute SonarQube Scanner".

- Configure project key and paths to source code.

Code Snippet:

```
pipeline {
  agent any
  stages {
    stage('SonarQube Analysis') {
      steps {
        script {
          withSonarQubeEnv('SonarQube') {
            bat 'mvn clean verify sonar:sonar'
          }
        }
      }
    }
  }
}
```

Output:
Jenkins pipeline configured successfully.
SonarQube Scanner step added to the build process.

Step 4: Run the Build

- Trigger the Jenkins build manually or via Git commit.

- Jenkins executes the SonarQube scan and uploads analysis results.

- View detailed reports on vulnerabilities, bugs, and code smells in the SonarQube dashboard.

Output:
Build started...
SonarQube analysis completed.
Results uploaded to SonarQube dashboard.
Quality Gate: PASSED.

Step 5: Review and Remediate

- Review identified issues and fix critical vulnerabilities.

- Re-run the analysis to verify remediation.

Output:
Critical issues fixed and verified.
Final scan shows no remaining vulnerabilities.

Output

The SonarQube dashboard displayed the following details after analysis:
- Security vulnerabilities and their severity levels.
- Code smells and duplicated blocks.
- Quality gate status (Pass/Fail).
- Trends in project maintainability and reliability.

Conclusion

This experiment demonstrated how integrating SonarQube with Jenkins enables early detection and remediation of application vulnerabilities through automated static analysis. The combined use of these tools promotes secure coding practices, ensures continuous code quality monitoring, and embeds security within the DevOps lifecycle.

The Bombay Salesian Society's DON BOSCO INSTITUTE OF TECHNOLOGY

(An Autonomous Institute affiliated to University of Mumbai) **Department of Information Technology**

EXP 5 : DevSecOps: Continuous Monitoring using Nagios

Aim:- To understand the concept and importance of **continuous monitoring** in DevSecOps.

1. Introduction

In a modern DevSecOps pipeline, **continuous monitoring** plays a vital role in ensuring that systems remain **secure**, **available**, and **resilient**. Continuous monitoring involves the **realtime detection**, **reporting**, and **response** to incidents or issues within an organization's infrastructure.

One of the most popular open-source tools used for this purpose is **Nagios**.

- It allows administrators to **monitor servers, network devices, services**, and applications.
- Nagios provides **alerts and notifications** in case of failures or security issues.
- It supports **plugins (NRPE)** for monitoring remote hosts.
- This ensures that **DevSecOps teams** can react quickly and maintain system reliability.

2. Procedure Step 1: Prerequisites

- A Linux machine (e.g., Ubuntu)
- Root or sudo access
- Internet connection
- Two or more servers (one Nagios server, one or more monitored hosts)

Step 2: System Update and Required Packages sudo apt update && sudo apt upgrade -y sudo apt install apache2 php php-cgi libapache2-mod-php php-mbstring unzip wget buildessential libgd-dev -y

Apache and PHP are required for the Nagios web interface.

Step 3: Create Nagios User and Group

sudo useradd nagios sudo groupadd

nagcmd sudo usermod -a -G nagcmd
nagios sudo usermod -a -G nagcmd
www-data
This creates the user and group required for running Nagios services.

Step 4: Download and Install Nagios Core cd /tmp wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.4.6.tar.gz tar xzf nagios-4.4.6.tar.gz cd nagios-4.4.6
./configure --with-command-group=nagcmd
make all sudo make install sudo make
install-init sudo make install-
commandmode sudo make install-config
sudo make install-webconf

Step 5: Set Up Web Interface Authentication sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
Enter and confirm password sudo
systemctl restart apache2
You can now access Nagios at: http://<your-server-ip>/nagios

Step 6: Install and Configure Nagios Plugins cd /tmp wget https://nagios-plugins.org/download/nagios-plugins-2.3.3.tar.gz tar xzf nagios-plugins-2.3.3.tar.gz cd nagios-plugins-2.3.3
./configure --with-nagios-user=nagios --with-nagios-group=nagios
make sudo make
install

Step 7: Install and Configure NRPE on Client Host
On each monitored host: sudo apt install nagios-nrpe-
server nagios-plugins -y sudo nano
/etc/nagios/nrpe.cfg

Add the Nagios server IP in the allowed_hosts line

allowed_hosts=127.0.0.1,<Nagios_server_IP>

sudo systemctl restart nagios-nrpe-server On Nagios server:

sudo apt install nagios-nrpe-plugin -y

Step 8: Add Hosts and Services to Nagios Edit host configuration file:

sudo nano /usr/local/nagios/etc/servers/client.cfg

Example: define host{

use linux-server

host_name client1 alias

Ubuntu Client address

192.168.1.10

max_check_attempts 5

check_period 24x7

notification_interval 30

notification_period 24x7

}

define service{

use generic-service

host_name client1

service_description PING

check_command

check_ping!100.0,20%!500.0,60% }

Include new config: sudo nano /usr/local/nagios/etc/nagios.cfg # Add line:

cfg_file=/usr/local/nagios/etc/servers/client.cf

g

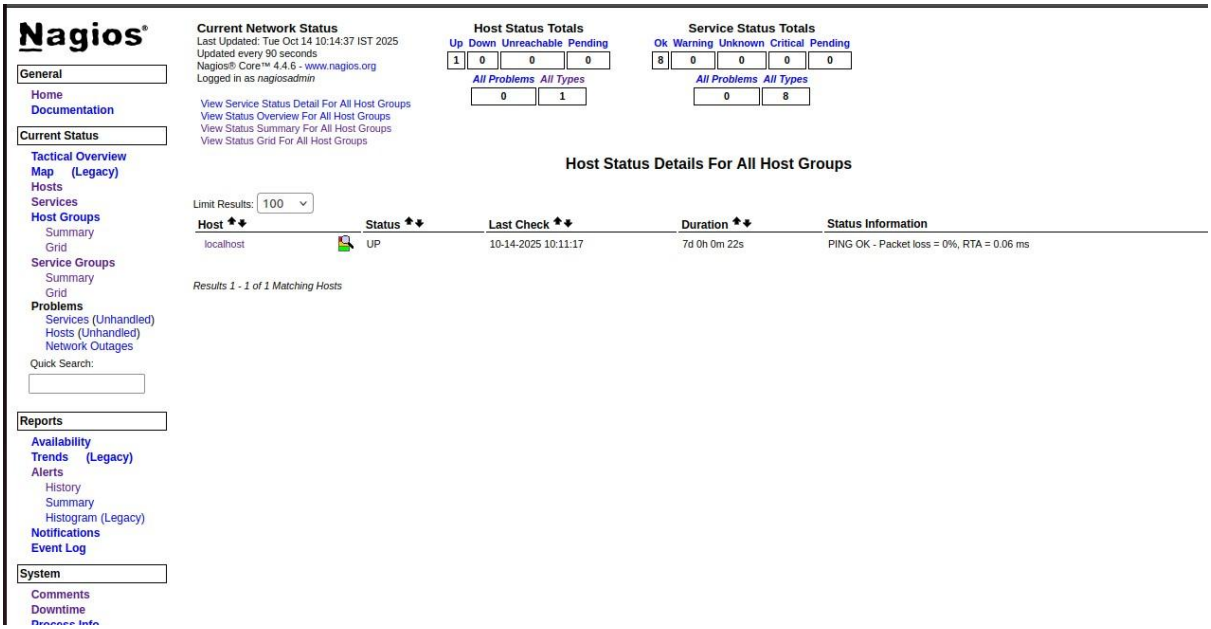
Step 9: Verify and Restart Nagios sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg sudo systemctl restart nagios

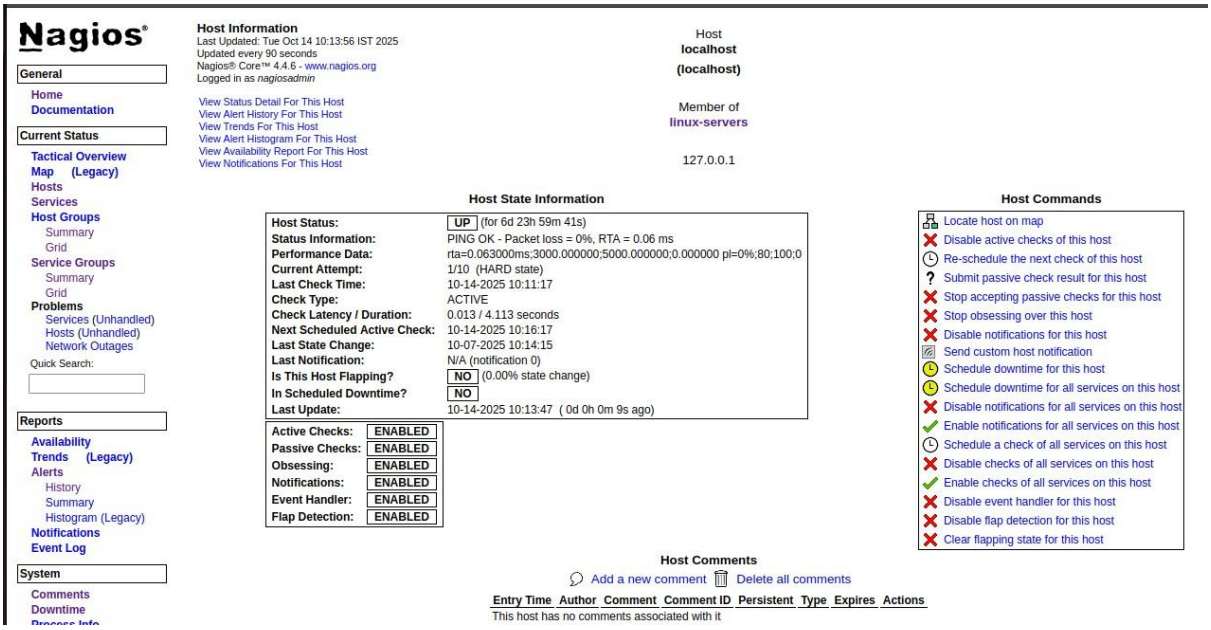
Step 10: Access Dashboard

Open browser:
http://<your-server-ip>/nagios Login with nagiosadmin credentials.

You will see:
Host status
Service status
Alerts and notifications
Performance graphs

3. Output





4. Observation

- Nagios continuously checks the health of both local and remote systems.
- When a service failed (e.g., stopped apache2 on client), Nagios dashboard reflected a **CRITICAL** alert.
- Alerts were displayed **within seconds**.
- Adding new hosts is simple and scalable through config files.
- Nagios provides a clear visualization of infrastructure health.

5. Conclusion

Continuous Monitoring is a **core pillar of DevSecOps**.

- It enables **proactive detection and response** to failures or security incidents.
- Nagios proved to be an effective, stable, and open-source monitoring solution.
- With NRPE, multiple servers can be monitored centrally.
- Real-time notifications help teams **reduce downtime**, **improve system reliability**, and **enhance security**.
- This setup is essential for maintaining healthy infrastructure in production environments.

The Bombay Salesian Society's
DON BOSCO INSTITUTE OF TECHNOLOGY
(An Autonomous Institute affiliated to University of Mumbai)
Department of Information Technology

Experiment 6: NoOps serverless computing

Serverless Computing (a)

1. Introduction

Serverless computing allows developers to build and run applications without managing the underlying infrastructure. AWS Lambda is a popular serverless compute service that automatically executes code in response to events. In this assignment, a simple "Hello World" Lambda function is implemented using the AWS Management Console.

2. Steps to Create AWS Lambda Function

- Step 1: Open AWS Lambda Service

Log in to the AWS Management Console and search for 'Lambda' in the services search bar. Click on the Lambda service to proceed.

- Step 2: Create a New Function

Click on 'Create Function' and choose 'Author from Scratch'. Enter a function name such as 'HelloLambda'. Select Python 3.x as the runtime. Under Permissions, choose 'Create a new role with basic Lambda permissions', then click 'Create Function'.

- Step 3: Add the Python Code

In the code editor section, replace the default code with the following Python function:

Python Code:

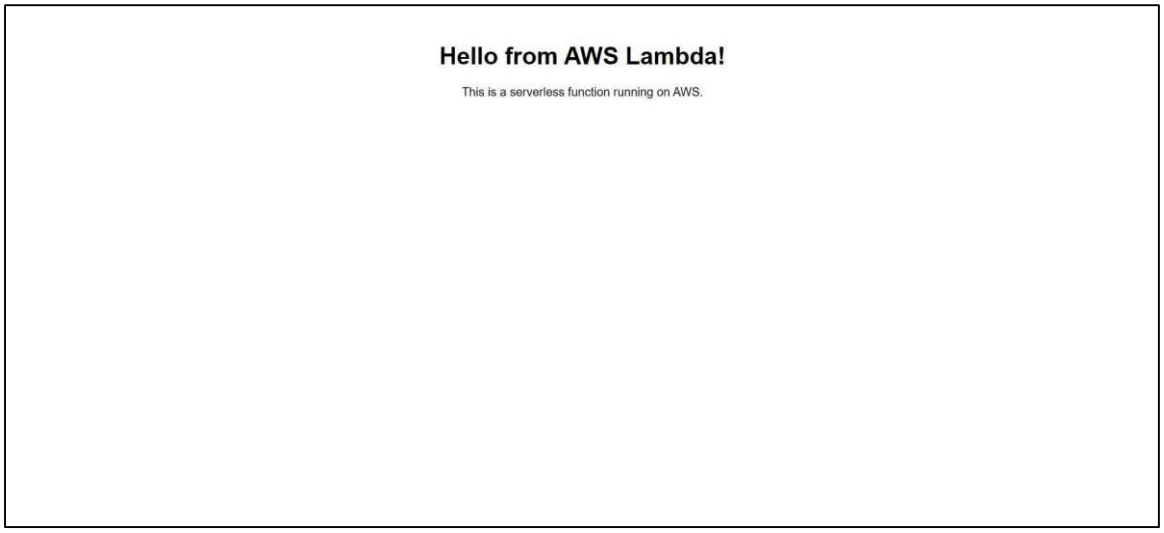
```
import json

def lambda_handler(event, context):
  """
  AWS Lambda handler function.
  This returns an HTML page that can be displayed in a web browser.  """
  html_content =
  """
  <html>
```

```
<head>
   <title>Hello from AWS Lambda</title>
   <style> body { font-family: Arial, sans-serif; text-align: center; margin-top:
      50px; } h1 { color: #4CAF50; }
   </style>
</head>
<body>
   <h1>Hello from AWS Lambda!</h1>
   <p>This is a serverless function running on AWS.</p>
</body>
</html>
""" return

{

'statusCo

de': 200,

   'headers': {
      'Content-Type': 'text/html'
   },
   'body': html_content
}
```

- Step 4: Test the Function

Click on the 'Test' button. If prompted, configure a new test event with any name and default input. Run the test and the output will display the 'Hello' message.

3. Output Screenshot



4. Conclusion

This simple example demonstrates how AWS Lambda enables serverless execution of code with minimal setup. By using the AWS Console, developers can quickly deploy and test functions without managing servers.

Serverless Computing (b)

1. Introduction

Serverless computing allows developers to build applications without managing infrastructure. AWS Lambda can be created and deployed using the AWS CLI, enabling automation and quick deployment. This assignment demonstrates creating a simple "Hello World" Lambda function using AWS CLI on Linux.

2. Steps to Create AWS Lambda Function via CLI

- Step 1: Install and Configure AWS CLI

Ensure AWS CLI v2 is installed. Configure it with your credentials using:
aws configure
Enter your Access Key, Secret Key, region (e.g., us-east-1), and output format (e.g., json).

- Step 2: Create Python Function File

Create a file named lambda_function.py with the following code:

- Step 3: Zip the Python File

zip function.zip lambda_function.py
This creates a zip file function.zip to upload to AWS Lambda.

- Step 4: Create IAM Role (if not already created)

Create a role with basic Lambda execution permissions. Save the Role ARN for the next step.

- Step 5: Create Lambda Function

aws lambda create-function --function-name HelloLambdaCLI --runtime python3.9 --role <Your-Role-ARN> --handler lambda_function.lambda_handler --zip-file fileb://function.zip
Replace <Your-Role-ARN> with your actual IAM Role ARN.

- Step 6: Test Lambda Function

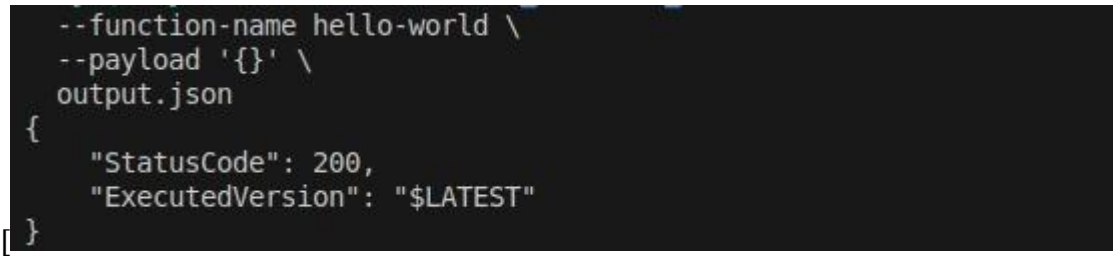
Invoke the Lambda function:
aws lambda invoke --function-name HelloLambdaCLI output.txt
Check the file output.txt to see the output: {"statusCode": 200, "body": "Hello from AWS Lambda via Console!"}

3. Python Code

```
def lambda_handler(event, context):
  return {
```

```
    'statusCode': 200,
    'body': 'Hello, World!
}
```

4. CLI Output Screenshot



5. Conclusion

This assignment demonstrates creating and deploying a Lambda function using AWS CLI on Linux. Using the zipped file method ensures that larger, more complex functions can be deployed easily and automated in scripts.