INFRASTRUCTURE AS CODE (IAC) USING TERRAFORM IN AWS

Deploying AWS Cloud Resources via Terraform Automation

Prepared by: Gayathri Nandakumar

Role: AWS Trainer/ AWS solution Architect Associate Enthusiastic

Date: 12 January 2026

TABLE OF CONTENTS

ABSTRACT

Infrastructure as Code (IaC) is an approach to automate cloud resource provisioning using machine-readable configuration files instead of manual processes. This project demonstrates the deployment of foundational AWS resources using Terraform, including a Virtual Private Cloud (VPC), Internet Gateway, Route Table, Subnet, and an EC2 instance. Through declarative configuration, Terraform ensures consistent, repeatable, and scalable deployments without manual console operations.

The objective of this project is to showcase how IaC can simplify cloud operations by enabling automated resource creation, dependency management, and version-controlled infrastructure. The implementation follows a step-by-step procedure including initialization, validation, planning, and applying Terraform configurations. Outputs are generated to provide visibility into important AWS resource identifiers.

This project highlights key benefits of IaC such as improved reliability, faster provisioning, auditability, and reusability in DevOps and Cloud environments. The solution is suitable for beginners seeking hands-on exposure to Terraform automation while aligning with industry standards for cloud infrastructure management.

INTRODUCTION

Cloud computing has transformed the way modern infrastructure is deployed and managed by providing scalable, flexible, and on-demand resources. Traditionally, system administrators provisioned infrastructure manually using graphical dashboards or command-line tools, which is error-prone, time-consuming, and difficult to replicate across environments. Infrastructure as Code (IaC) addresses these limitations by allowing infrastructure to be defined, provisioned, and managed through machine-readable configuration files.

Terraform, developed by HashiCorp, is one of the most widely adopted IaC tools used in DevOps and cloud automation. It follows a declarative approach, which means the user defines the desired end-state of the infrastructure, and Terraform automatically computes and executes the required steps to reach that state. With support for multiple cloud providers and third-party services, Terraform enables consistent, repeatable, and version-controlled infrastructure deployments.

This project demonstrates how Terraform can be used to provision foundational AWS infrastructure components such as networking resources and compute instances. The implementation highlights core IaC concepts including resource dependency management, automated deployment, and standardized configuration. The outcome of this project is a reproducible environment that can be deployed, destroyed, and redeployed without manual intervention, aligning with modern DevOps practices.

OBJECTIVES

The primary objectives of this project are as follows:

- To demonstrate the concept of Infrastructure as Code (IaC) using Terraform.
- To automate the provisioning of AWS infrastructure components through configuration files.
- To deploy networking resources including VPC, Subnet, Route Table, and Internet Gateway.
- To launch an EC2 instance within a custom VPC architecture using Terraform.
- To produce reproducible, scalable, and version-controlled infrastructure deployments.
- To enable efficient resource creation, deletion, and re-creation without manual intervention.
- To introduce beginners to Terraform workflows such as initialization, validation, planning, and application.

TECHNOLOGIES USED

This project makes use of the following technologies and tools:

- **Terraform:** Infrastructure as Code (IaC) tool used for automated cloud provisioning.
- **AWS (Amazon Web Services):** Cloud platform used for resource hosting and deployment.
- **AWS VPC:** Virtual Private Cloud service used for creating isolated network environments.
- **EC2 (Elastic Compute Cloud):** AWS compute service used for launching virtual machine instances.
- **Internet Gateway:** Networking component used to enable public access from VPC to the internet.
- **Route Table:** Component responsible for defining network traffic routes.
- **Subnet:** Logical segmentation of VPC used for resource placement.
- **CLI Tools:** Includes AWS CLI or terminal for Terraform command execution.
- **Git & GitHub:** Version control and repository storage for source code and documentation.

SYSTEM REQUIREMENTS

The following hardware and software requirements are necessary to implement this project:

Hardware Requirements

- 64-bit Laptop or Desktop System
- Minimum 4 GB RAM (8 GB Recommended)
- Minimum 10 GB Free Disk Space
- Stable Internet Connectivity

Software Requirements

- Windows, Linux, or macOS Operating System
- Terraform Installed (Version 0.14+ Recommended)
- AWS Account with Access Credentials (Access/Secret Key)
- AWS Management Console Access
- AWS CLI (Optional but Recommended)
- Web Browser (Chrome, Firefox, Edge, etc.)
- Git & GitHub Account (For Version Control and Repository)

SYSTEM DESIGN

The system design focuses on automating AWS infrastructure provisioning using Infrastructure as Code (IaC) principles with Terraform. The primary objective is to eliminate manual configuration steps by defining reusable infrastructure templates that can be applied consistently across environments.

The designed IaC system provisions foundational cloud components such as a Virtual Private Cloud (VPC), Subnet, Internet Gateway, Route Table, Security Group, and EC2 instance. All resources are declared as code, enabling version control, repeatability, and automated deployment.

Architectural Overview

The AWS infrastructure architecture consists of the following components:

- **VPC:** A logically isolated virtual network for hosting cloud resources.
- **Subnet:** A public subnet configured within the VPC.
- **Internet Gateway (IGW):** Allows outbound/inbound internet traffic.
- **Route Table:** Routes traffic from the subnet to the Internet through the IGW.
- **Security Group:** Acts as a virtual firewall controlling inbound/outbound rules.
- **EC2 Instance:** A compute resource deployed within the public subnet.

The infrastructure flow is designed such that the EC2 instance interacts with the internet through the subnet’s routing configuration while maintaining controlled network access.

Terraform Configuration Approach

Terraform configuration follows a modular and declarative structure using:

- **Provider Block:** Specifies AWS region and authentication.
- **Resource Blocks:** Define cloud components programmatically.
- **Variables & Outputs (Optional):** Improve flexibility and manage display values.

The code execution lifecycle is driven by Terraform commands such as `init`, `plan`, and `apply` to provision infrastructure.

Advantages of IaC Design

The IaC-based design offers the following benefits:

- Eliminates configuration drift
- Enables reproducible infrastructure deployment
- Enhances scalability and automation
- Improves maintainability via version control (Git)
- Reduces human error in cloud configuration

This design ensures reliable cloud deployments suitable for modern DevOps-driven environments.

SYSTEM ARCHITECTURE DIAGRAM

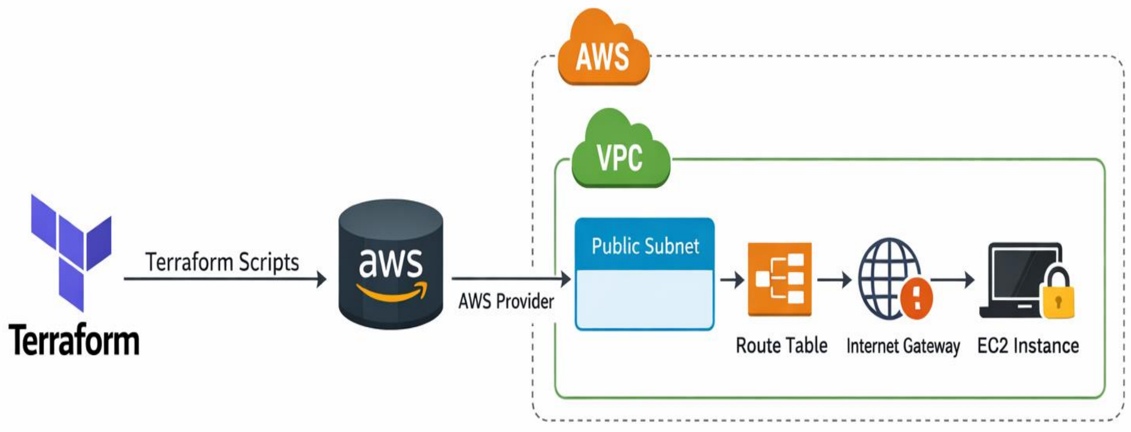


Figure 1: *Terraform-based AWS Infrastructure Architecture*

IMPLEMENTATION

The implementation phase involves writing Terraform configuration files to provision AWS infrastructure programmatically. Each Terraform block represents a specific AWS resource, ensuring consistency and full automation during deployment.

Provider and Credential Setup

Terraform requires a provider configuration to interact with AWS APIs. The AWS provider block specifies the desired region. AWS credentials are configured locally using either the AWS CLI or environment variables. This allows Terraform to authenticate securely and perform infrastructure operations.

Defining AWS Infrastructure Resources

Terraform resource blocks were written to define and deploy core networking components:

- **VPC** for hosting cloud resources
- **Subnet** associated with the VPC
- **Internet Gateway** for internet access
- **Route Table** configured with routing rules
- **Security Group** enabling SSH/HTTP access
- **EC2 Instance** deployed in the public subnet

Each resource is declared in the `main.tf` file, ensuring consistent provisioning.

Terraform Execution Workflow

The Infrastructure as Code execution lifecycle includes:

1. **Initialization:**
   `terraform init` downloads providers and modules.
2. **Planning:**
   `terraform plan` previews infrastructure changes before applying.
3. **Execution:**
   `terraform apply` provisions resources via AWS APIs.
4. **Output Extraction:**
   `terraform output` retrieves resource details such as instance public IP.
5. **Destruction (Optional):**
   `terraform destroy` removes all created AWS resources to avoid cost.

This workflow allows safe infrastructure provisioning, updates, and teardown.

Validation and Testing

After deployment, infrastructure validation was performed via:

- AWS Console inspection to verify networking and instance configuration
- Public IP connectivity testing via browser/SSH
- Security group rule verification for traffic flow

Successful validation confirmed that Terraform provisioned the infrastructure as expected.

Version Control Integration

Git was used to track Terraform files (main.tf, outputs.tf, screenshots, documentation). Version control ensures:

- Collaboration support
- Traceability of configuration changes
- Reusability of IaC templates

The repository also contains README documentation for reproducibility.

Final Outcome

The final implementation produced a fully automated AWS infrastructure deployment using Terraform. The configuration enabled repeatable, scalable, and error-free provisioning aligned with DevOps best practices.

Terraform State Management

Terraform maintains a state file named terraform.tfstate which records the current status of all provisioned infrastructure. This state allows Terraform to understand which resources already exist, detect configuration drift, and determine the required changes during updates. In production environments, remote backends such as Amazon S3 (for storing state) with DynamoDB (for state locking) are commonly used to ensure security, collaboration, and consistency among DevOps teams.

METHODOLOGY

The methodology adopted in this project follows a structured sequence of steps designed to automate AWS resource provisioning using Infrastructure as Code (IaC) principles. The approach ensures reliability, consistency, and repeatability across cloud environments.

Requirement Analysis

The first phase involved understanding project needs and defining scope. Requirements included the automated creation of a VPC-based infrastructure with networking components and an EC2 instance. Non-functional requirements such as scalability, automation, reproducibility, and minimal human intervention were also considered.

Environment Setup

A suitable development environment was prepared to support Terraform operations. This involved:

- Installing Terraform
- Configuring AWS credentials
- Preparing a code editor (VS Code)
- Setting up a GitHub repository for version control

This ensured a smooth provisioning workflow.

Credential Configuration

AWS credentials (Access Key & Secret Key) were securely configured through the local environment to enable Terraform to interact with the AWS API. Optional AWS CLI integration was used to validate connectivity.

Infrastructure Definition using Terraform

Terraform configuration files (`main.tf` and `outputs.tf`) were authored to define the required AWS resources including:

- VPC
- Public Subnet
- Internet Gateway & Route Table
- Security Group
- EC2 Instance

Terraform's declarative configuration model ensured that resources were consistently defined and version-controlled.

Initialization and Validation

Terraform was initialized using `terraform init` to download provider plugins. The configuration syntax was validated using `terraform validate` to ensure correctness. A dry-run was performed using `terraform plan` to preview the changes before execution.

Execution and Provisioning

The infrastructure was provisioned using `terraform apply`, which created AWS resources through backend API calls. Terraform state management ensured resources were tracked and could be updated or destroyed systematically.

Testing and Verification

After deployment, the environment was validated through:

- Terraform output values (EC2 Public IP, VPC ID, etc.)
- AWS Management Console checks
- Public accessibility tests (based on allowed security rules)

Successful validation confirmed proper resource creation and network routing.

Documentation and Versioning

All files were documented and uploaded to a GitHub repository for version control and public reference. Screenshots, demo video links, and PDF documentation were also included to support future reuse and evaluation.

This methodology ensured that the entire infrastructure lifecycle—from setup to teardown—was fully automated, reproducible, and aligned with modern DevOps practices.

RESULTS

The implementation of Infrastructure as Code (IaC) using Terraform successfully automated the provisioning of a complete AWS environment without manual intervention through the AWS Console. The deployment achieved the desired objectives of consistency, repeatability, and infrastructure versioning.

The key results obtained from the project are summarized below:

Infrastructure Provisioning Output

Terraform executed resource creation using the `terraform apply` command. Upon successful deployment, the following AWS resources were provisioned:

- Virtual Private Cloud (VPC)
- Public Subnet
- Internet Gateway (IGW)
- Route Table with associations
- Security Group with configured ingress/egress rules
- EC2 Instance accessible via public IP

Terraform's command-line output confirmed the creation of these resources and displayed resource identifiers and status details.

Public Connectivity Verification

The EC2 instance received a valid public IPv4 address, which was extracted using Terraform output values. The instance was accessible externally based on security group rules, confirming that:

- Routing was correctly configured through the IGW
- Public subnet networking operated as expected
- Security group rules permitted SSH/ICMP/HTTP traffic (based on configuration)

AWS Console Validation

Validation via AWS Management Console confirmed the provisioning of:

- VPC and Subnet under VPC Dashboard
- Internet Gateway and Route Table associations
- Security Group attached to the EC2 instance
- EC2 instance in "running" state with valid networking attributes

This verified that the Terraform configuration accurately represented the intended infrastructure.

Reproducibility and Consistency

Due to the IaC approach, the entire infrastructure could be:

- Recreated
- Modified
- Destroyed

with consistent outcomes using Terraform commands.

The terraform destroy command was executed (optional) to validate resource cleanup functionality, confirming successful deletion of all provisioned components.

Performance and Efficiency

The provisioning process demonstrated:

- **Reduced deployment time** compared to manual provisioning
- **Elimination of manual configuration errors**
- **Infrastructure repeatability** across development environments

This validates Terraform as an efficient tool for cloud resource automation.

Repository and Documentation Output

The final project output includes:

- Terraform configuration files (main.tf, outputs.tf)
- Project documentation (PDF)
- Screenshots from AWS Console
- Demo video showing deployment execution
- GitHub repository for version control and public access

Summary of Results

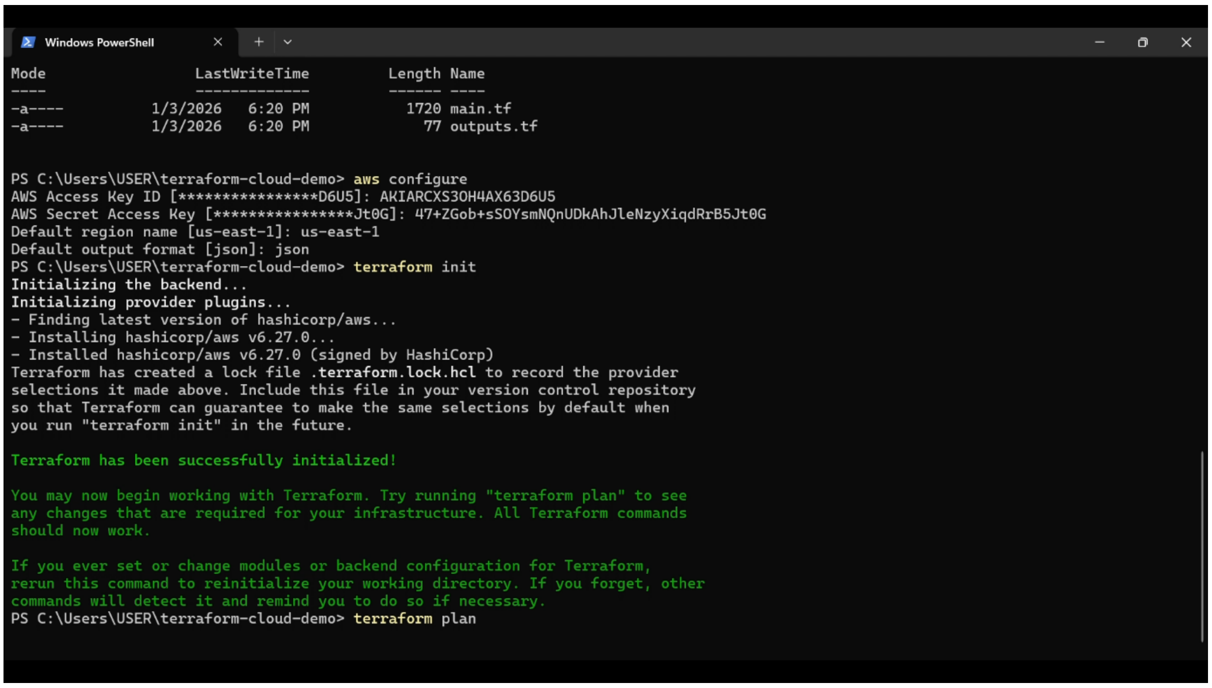
The project conclusively demonstrated that IaC using Terraform enables:

Automated provisioning
Reliable infrastructure replication
Programmatic infrastructure management
Integration with modern DevOps workflows

The successful deployment validates Terraform's role as a core DevOps and cloud automation technology.

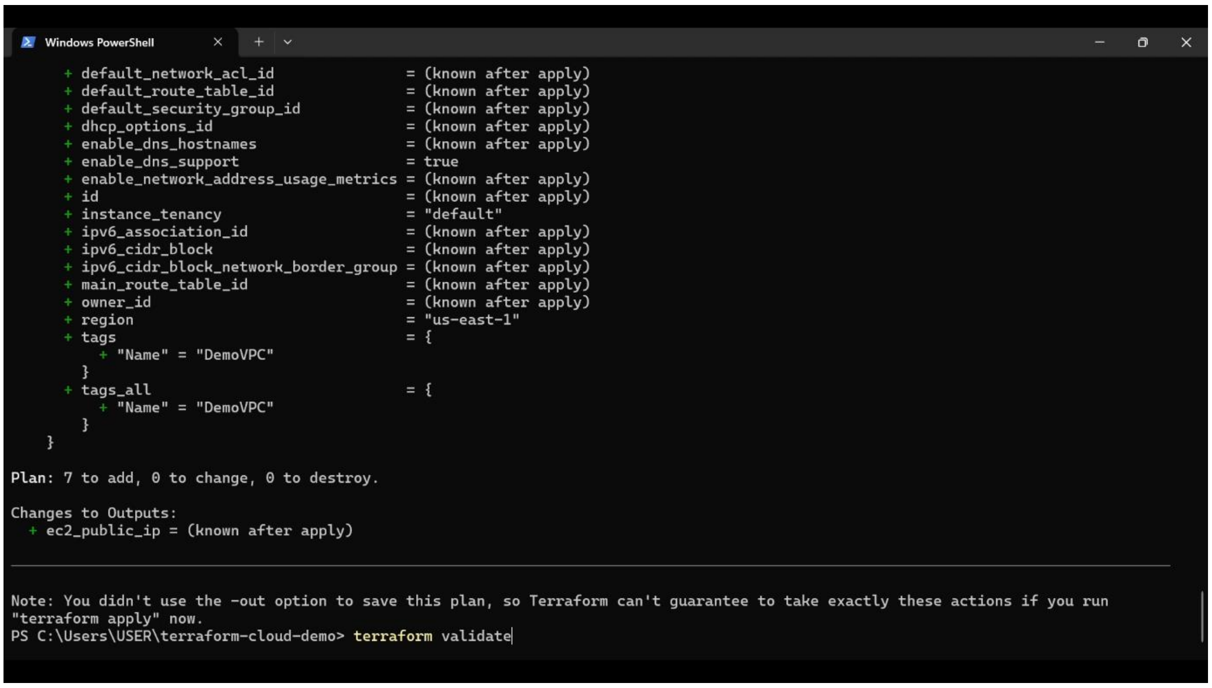
Screenshots

1. Terraform init



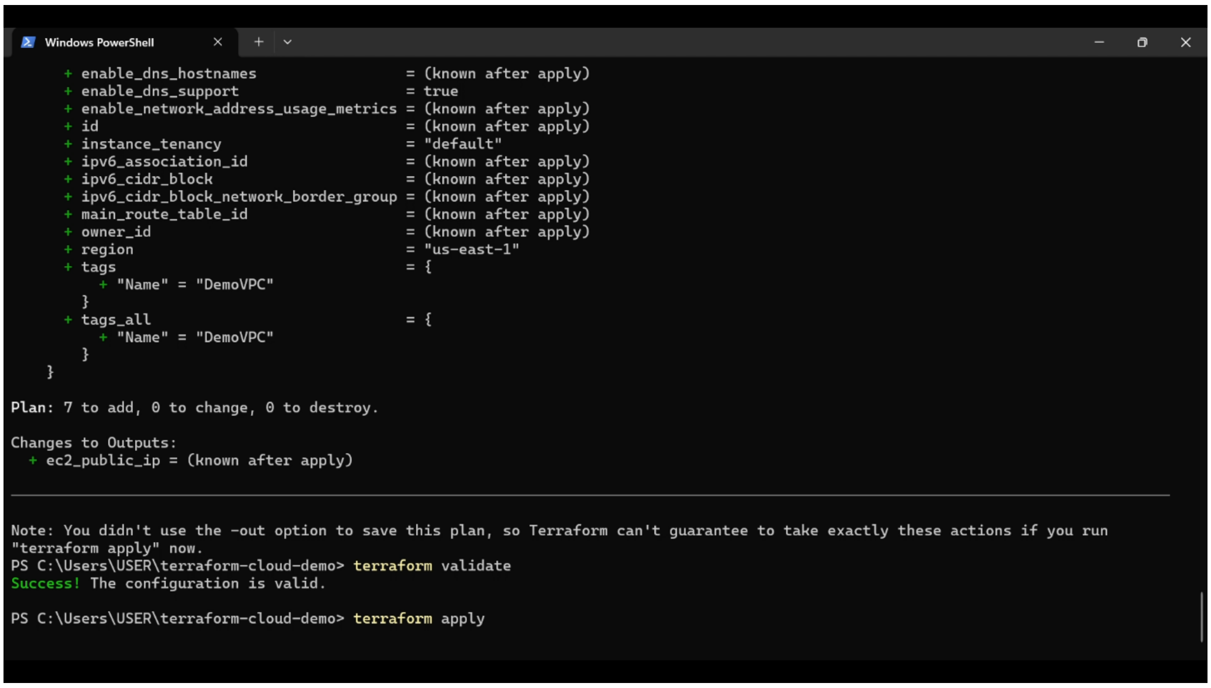
```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          1/3/2026   6:20 PM           1720 main.tf
-a----          1/3/2026   6:20 PM             77 outputs.tf


PS C:\Users\USER\terraform-cloud-demo> aws configure
AWS Access Key ID [****************D6U5]: AKIARCXS3OH4AX63D6U5
AWS Secret Access Key [****************Jt0G]: 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
Default region name [us-east-1]: us-east-1
Default output format [json]: json
PS C:\Users\USER\terraform-cloud-demo> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.27.0...
- Installed hashicorp/aws v6.27.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
PS C:\Users\USER\terraform-cloud-demo> terraform plan
```

2. Terraform plan



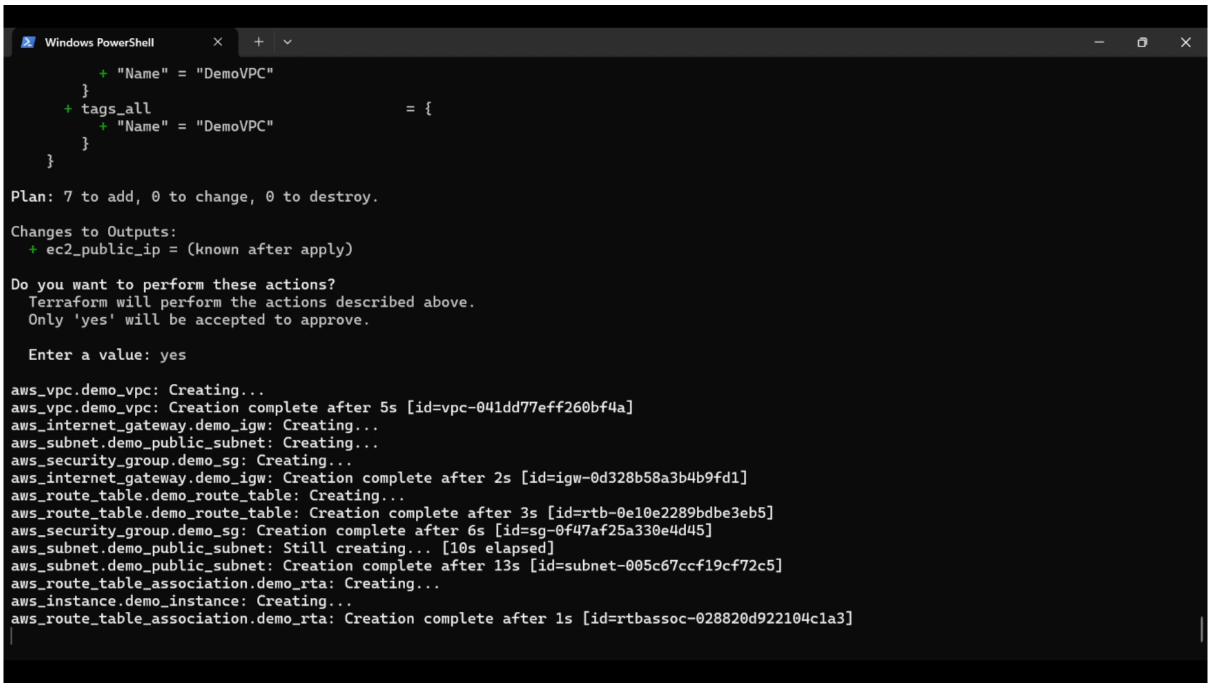
```
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + region                               = "us-east-1"
      + tags                                 = {
          + "Name" = "DemoVPC"
        }
      + tags_all                             = {
          + "Name" = "DemoVPC"
        }
    }

Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + ec2_public_ip = (known after apply)

─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

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.
PS C:\Users\USER\terraform-cloud-demo> terraform validate
```

3. Terraform validate



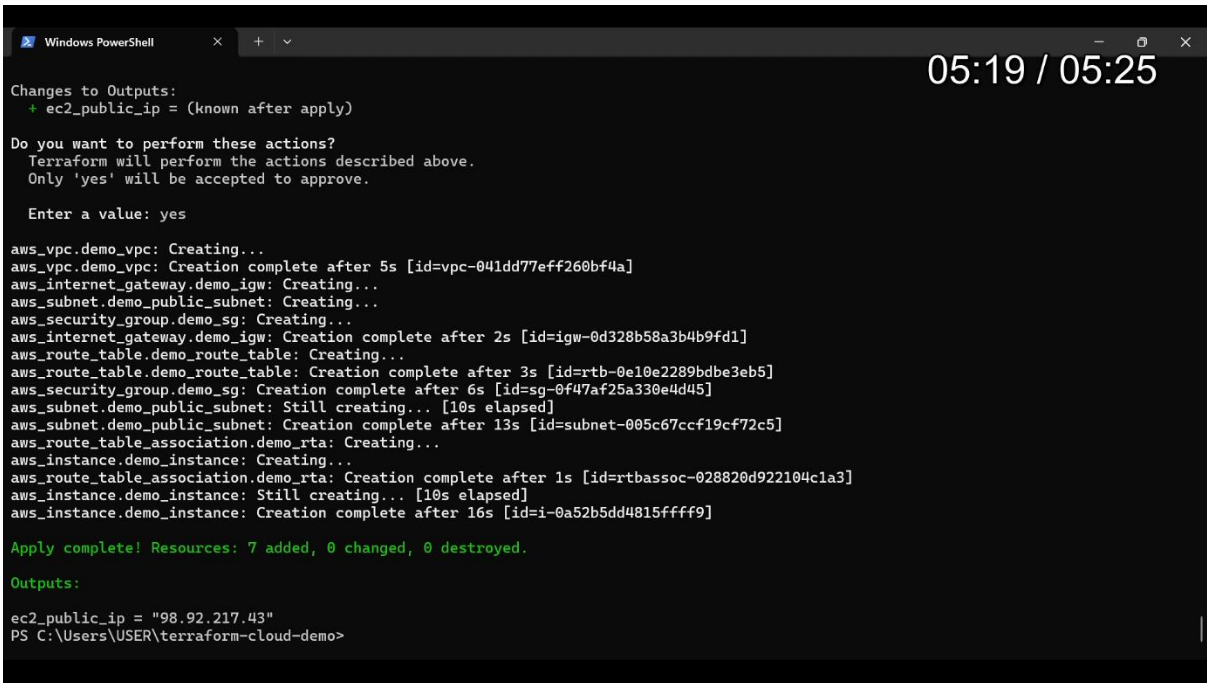
```
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + region                               = "us-east-1"
      + tags                                 = {
          + "Name" = "DemoVPC"
        }
      + tags_all                             = {
          + "Name" = "DemoVPC"
        }
    }

Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + ec2_public_ip = (known after apply)

─────────────────────────────────────────────────────────────────────────────────────────────────────────

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.
PS C:\Users\USER\terraform-cloud-demo> terraform validate
Success! The configuration is valid.

PS C:\Users\USER\terraform-cloud-demo> terraform apply
```

4. Terraform apply



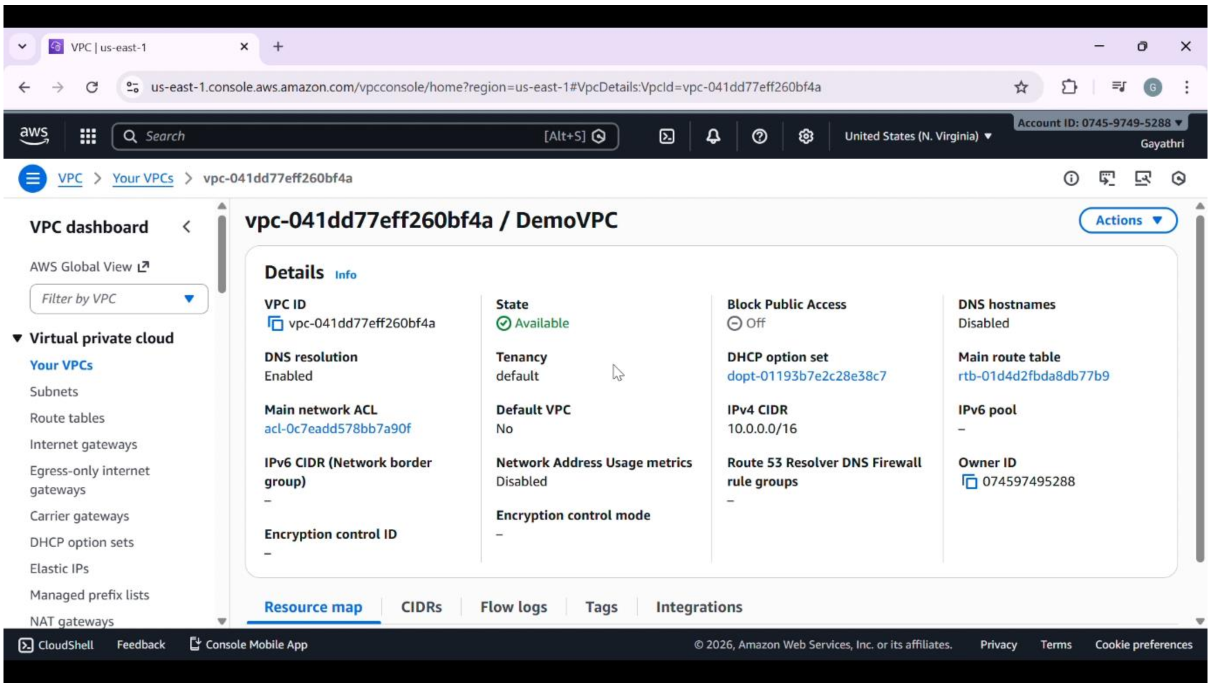
```
          + "Name" = "DemoVPC"
        }
      + tags_all                             = {
          + "Name" = "DemoVPC"
        }
    }

Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + ec2_public_ip = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.demo_vpc: Creating...
aws_vpc.demo_vpc: Creation complete after 5s [id=vpc-041dd77eff260bf4a]
aws_internet_gateway.demo_igw: Creating...
aws_subnet.demo_public_subnet: Creating...
aws_security_group.demo_sg: Creating...
aws_internet_gateway.demo_igw: Creation complete after 2s [id=igw-0d328b58a3b4b9fd1]
aws_route_table.demo_route_table: Creating...
aws_route_table.demo_route_table: Creation complete after 3s [id=rtb-0e10e2289bdbe3eb5]
aws_security_group.demo_sg: Creation complete after 6s [id=sg-0f47af25a330e4d45]
aws_subnet.demo_public_subnet: Still creating... [10s elapsed]
aws_subnet.demo_public_subnet: Creation complete after 13s [id=subnet-005c67ccf19cf72c5]
aws_route_table_association.demo_rta: Creating...
aws_instance.demo_instance: Creating...
aws_route_table_association.demo_rta: Creation complete after 1s [id=rtbassoc-028820d922104c1a3]
```

5. Output ip



```
Changes to Outputs:
  + ec2_public_ip = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.demo_vpc: Creating...
aws_vpc.demo_vpc: Creation complete after 5s [id=vpc-041dd77eff260bf4a]
aws_internet_gateway.demo_igw: Creating...
aws_subnet.demo_public_subnet: Creating...
aws_security_group.demo_sg: Creating...
aws_internet_gateway.demo_igw: Creation complete after 2s [id=igw-0d328b58a3b4b9fd1]
aws_route_table.demo_route_table: Creating...
aws_route_table.demo_route_table: Creation complete after 3s [id=rtb-0e10e2289bdbe3eb5]
aws_security_group.demo_sg: Creation complete after 6s [id=sg-0f47af25a330e4d45]
aws_subnet.demo_public_subnet: Still creating... [10s elapsed]
aws_subnet.demo_public_subnet: Creation complete after 13s [id=subnet-005c67ccf19cf72c5]
aws_route_table_association.demo_rta: Creating...
aws_instance.demo_instance: Creating...
aws_route_table_association.demo_rta: Creation complete after 1s [id=rtbassoc-028820d922104c1a3]
aws_instance.demo_instance: Still creating... [10s elapsed]
aws_instance.demo_instance: Creation complete after 16s [id=i-0a52b5dd4815ffff9]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

ec2_public_ip = "98.92.217.43"
PS C:\Users\USER\terraform-cloud-demo>
```

6. VPC Dashboard

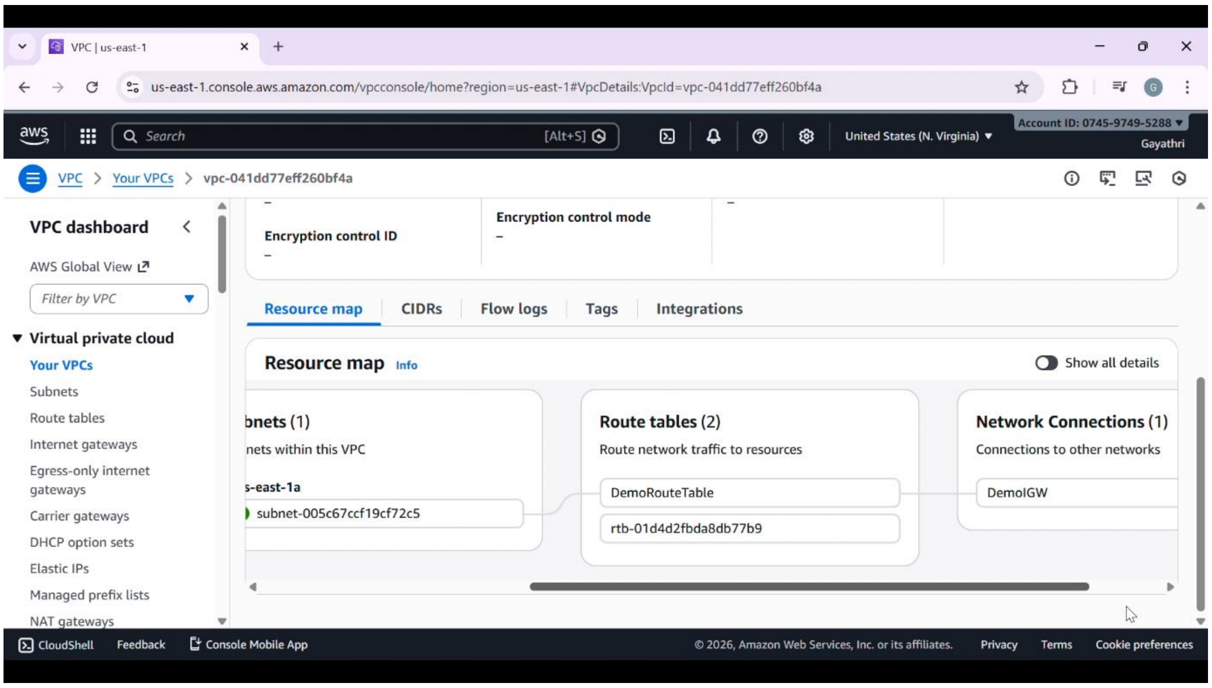


vpc-041dd77eff260bf4a / DemoVPC

Details Info

VPC ID vpc-041dd77eff260bf4a	**State** Available	**Block Public Access** Off	**DNS hostnames** Disabled
DNS resolution Enabled	**Tenancy** default	**DHCP option set** dopt-01193b7e2c28e38c7	**Main route table** rtb-01d4d2fbda8db77b9
Main network ACL acl-0c7eadd578bb7a90f	**Default VPC** No	**IPv4 CIDR** 10.0.0.0/16	**IPv6 pool** –
IPv6 CIDR (Network border group) –	**Network Address Usage metrics** Disabled	**Route 53 Resolver DNS Firewall rule groups** –	**Owner ID** 074597495288
Encryption control ID –	**Encryption control mode** –		

7. Subnet, RT, IGW


VPC | us-east-1
us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#VpcDetails:VpcId=vpc-041dd77eff260bf4a
Account ID: 0745-9749-5288
Gayathri
United States (N. Virginia)
VPC > Your VPCs > vpc-041dd77eff260bf4a
VPC dashboard
AWS Global View
Filter by VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
NAT gateways
CloudShell
Feedback
Console Mobile App
© 2026, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Encryption control ID
–

Encryption control mode
–

Resource map | CIDRs | Flow logs | Tags | Integrations

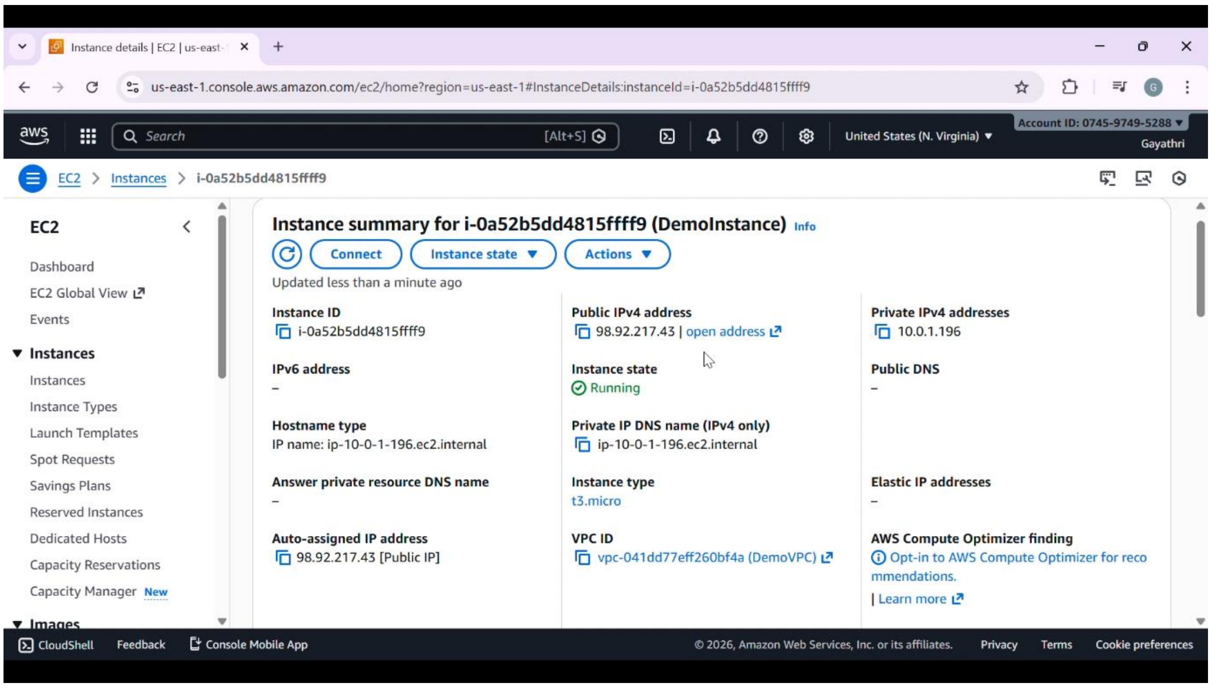
Resource map Info Show all details

bnets (1)
nets within this VPC
s-east-1a
subnet-005c67ccf19cf72c5

Route tables (2)
Route network traffic to resources
DemoRouteTable
rtb-01d4d2fbda8db77b9

Network Connections (1)
Connections to other networks
DemoIGW

8. EC2 Dashboard



Instance summary for i-0a52b5dd4815ffff9 (DemoInstance) Info

Updated less than a minute ago

Instance ID: i-0a52b5dd4815ffff9
Public IPv4 address: 98.92.217.43 | open address
Private IPv4 addresses: 10.0.1.196
IPv6 address: –
Instance state: Running
Public DNS: –
Hostname type: IP name: ip-10-0-1-196.ec2.internal
Private IP DNS name (IPv4 only): ip-10-0-1-196.ec2.internal
Answer private resource DNS name: –
Instance type: t3.micro
Elastic IP addresses: –
Auto-assigned IP address: 98.92.217.43 [Public IP]
VPC ID: vpc-041dd77eff260bf4a (DemoVPC)
AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more

CONCLUSION

This project successfully demonstrated the implementation of Infrastructure as Code (IaC) using Terraform to automate the provisioning of AWS cloud resources. By defining infrastructure through code rather than manual configuration, the deployment process became faster, more consistent, and highly repeatable.

The Terraform configuration files enabled the automated creation of a Virtual Private Cloud (VPC), public subnet, Internet gateway, route table, security group, and an EC2 instance. The workflow, which included initialization, planning, applying, and verification, showcased the efficiency and reliability of Terraform's declarative model and state management capabilities.

The deployment further highlighted several key benefits of IaC, including reduced human errors, improved infrastructure visibility, easier debugging, simplified rollback, and enhanced version control through Git. The use of AWS Management Console for verification ensured that the deployed resources aligned with the expected architecture.

Overall, the project validated Terraform as a powerful DevOps tool for managing cloud infrastructure in a scalable, structured, and automated manner. This approach is highly suitable for modern cloud environments, continuous deployment pipelines, and enterprise-level infrastructure management.

The workflow, which included initialization, planning, applying, and verification, demonstrated the full automation cycle of IaC. This approach produced reliable, repeatable, and scalable infrastructure deployments, validating Terraform as a powerful automation tool for modern cloud environments.

FUTURE ENHANCEMENTS

The current project can be extended further by incorporating the following improvements:

1. **Modular Terraform Architecture**
   Converting the configuration into reusable Terraform modules for better scalability and maintainability.
2. **Multi-Environment Support**
   Enabling infrastructure deployment across Dev, QA, Staging, and Production using workspaces or separate variable files.
3. **State Management with Remote Backend**
   Using Terraform Cloud or AWS S3 + DynamoDB for remote state and state locking to support team collaboration.
4. **CI/CD Pipeline Integration**
   Integrating Terraform with GitHub Actions, GitLab CI/CD, Jenkins, or AWS CodePipeline for automated deployments.
5. **Enhanced Security & Compliance**
   Implementing least privilege IAM roles, encryption, logging, and compliance checks using tools like Checkov or TFsec.
6. **Auto Scaling & Load Balancing**
   Extending infrastructure to include Application Load Balancer (ALB) and Auto Scaling Groups (ASG) for production readiness.
7. **Monitoring & Observability**
   Integrating CloudWatch dashboards, alarms, and log groups for better operational monitoring.

These enhancements would make the solution more enterprise-grade, scalable, and production-focused.

LIMITATIONS

Although the project achieved its intended objectives, certain limitations were observed:

1. **Manual Credential Configuration**
   AWS credentials were configured manually, which may not scale well for enterprise CI/CD environments.
2. **Single Region Deployment**
   The infrastructure was deployed in a single AWS region, without multi-region redundancy.
3. **Local State Management**
   Terraform state was stored locally, which limits collaboration and introduces risks during system failure.
4. **Limited Resource Scope**
   Only basic networking and compute resources were provisioned; advanced services (RDS, ALB, ASG) were out of scope.
5. **No Automated Testing**
   Infrastructure testing frameworks (Terratest, Inspec, or Kitchen-Terraform) were not integrated.

These limitations can be improved as part of future iterations.

DECLARATION

I hereby declare that the project entitled *“Infrastructure as Code (IaC) on AWS using Terraform”* submitted by me is a genuine work carried out under my supervision and guidance. This project has not been submitted, either in part or full, for any degree or diploma in any other institution or university.

All information and data used in this work are obtained from authentic sources and duly acknowledged in the Reference section of this document.

Place: Trichy
Date: 12 January 2026

Signature: Gayathri Nandakumar
Name: Gayathri Nandakumar

ACKNOWLEDGEMENT

I am grateful for the opportunity to work on this project titled **“Infrastructure as Code (IaC) on AWS using Terraform”**. This project allowed me to enhance my understanding of AWS cloud services, Infrastructure as Code practices, and Terraform automation, which are essential skills in modern DevOps and Cloud Architecture.

I would also like to acknowledge the availability of comprehensive documentation and learning resources provided by **AWS** and **HashiCorp Terraform**, which helped me in understanding cloud infrastructure provisioning and IaC methodologies efficiently.

Finally, I thank my family, peers, and well-wishers for their continuous support and encouragement throughout the development of this project.

Gayathri Nandakumar **Date:** 12 January 2026 **Place:** Trichy, India

REFERENCES

The following references were used to support this work:

1. Terraform Official Documentation — *HashiCorp*
   https://developer.hashicorp.com/terraform
2. AWS Documentation — *Amazon Web Services*
   https://docs.aws.amazon.com
3. Terraform AWS Provider Docs — *HashiCorp Registry*
   https://registry.terraform.io/providers/hashicorp/aws/latest/docs
4. DevOps & IaC Best Practices — *HashiCorp Blog and Case Studies*
5. Online DevOps Community Discussions, Labs, and Tutorials

These resources were used solely for understanding platform behaviors, best practices, and command specifications.

COPYRIGHT PAGE