Program No-5

Basic Infrastructure Deployment

- **Tool**: Terraform
- **Program**:
    - Write a Terraform configuration file to provision a single EC2 instance on AWS.

Use Terraform commands (terraform init, terraform plan, terraform apply, terraform destroy) to manage the infrastructure.

Infrastructure as Code (IaC)

Before the advent of IaC, infrastructure management was typically a manual and time-consuming process. System administrators and operations teams had to:

1. Manually Configure Servers: Servers and other infrastructure components were often set up and configured manually, which could lead to inconsistencies and errors.

2. Lack of Version Control: Infrastructure configurations were not typically version-controlled, making it difficult to track changes or revert to previous states.

3. Documentation Heavy: Organizations relied heavily on documentation to record the steps and configurations required for different infrastructure setups. This documentation could become outdated quickly.

4. Limited Automation: Automation was limited to basic scripting, often lacking the robustness and flexibility offered by modern IaC tools.

5. Slow Provisioning: Provisioning new resources or environments was a time-consuming process that involved multiple manual steps, leading to delays in project delivery.

IaC addresses these challenges by providing a systematic, automated, and code-driven approach to infrastructure management. Popular IaC tools include Terraform, AWS CloudFormation, Azure Resource Manager templates others.

These tools enable organizations to define, deploy, and manage their infrastructure efficiently and consistently, making it easier to adapt to the dynamic needs of modern applications and services.

Why Terraform ?

There are multiple reasons why Terraform is used over the other IaC tools but below are the main reasons.

1. **Multi-Cloud Support**: Terraform is known for its multi-cloud support. It allows you to define infrastructure in a cloud-agnostic way, meaning you can use the same configuration code to provision resources on various cloud providers (AWS, Azure, Google Cloud, etc.) and even on-premises infrastructure. This flexibility can be beneficial if your organization uses multiple cloud providers or plans to migrate between them.

2. **Large Ecosystem**: Terraform has a vast ecosystem of providers and modules contributed by both HashiCorp (the company behind Terraform) and the community. This means you can find pre-built modules and configurations for a wide range of services and infrastructure components, saving you time and effort in writing custom configurations.

3. **Declarative Syntax**: Terraform uses a declarative syntax, allowing you to specify the desired end-state of your infrastructure. This makes it easier to understand and maintain your code compared to imperative scripting languages.

4. **State Management**: Terraform maintains a state file that tracks the current state of your infrastructure. This state file helps Terraform understand the differences between the desired and actual states of your infrastructure, enabling it to make informed decisions when you apply changes.

5. **Plan and Apply**: Terraform's "plan" and "apply" workflow allows you to preview changes before applying them. This helps prevent unexpected modifications to your infrastructure and provides an opportunity to review and approve changes before they are implemented.

6. **Community Support**: Terraform has a large and active user community, which means you can find answers to common questions, troubleshooting tips, and a wealth of documentation and tutorials online.

7. **Integration with Other Tools**: Terraform can be integrated with other DevOps and automation tools, such as Docker, Kubernetes, Ansible, and Jenkins, allowing you to create comprehensive automation pipelines.

8. **HCL Language**: Terraform uses HashiCorp Configuration Language (HCL), which is designed specifically for defining infrastructure. It's human-readable and expressive, making it easier for both developers and operators to work with.

To get started with Terraform, it's important to understand some key terminology and concepts. Here are some fundamental terms and explanations.

1. **Provider**: A provider is a plugin for Terraform that defines and manages resources for a specific cloud or infrastructure platform. Examples of providers include AWS, Azure, Google Cloud, and many others. You configure providers in your Terraform code to interact with the desired infrastructure platform.

2. **Resource**: A resource is a specific infrastructure component that you want to create and manage using Terraform. Resources can include virtual machines, databases, storage

buckets, network components, and more. Each resource has a type and configuration parameters that you define in your Terraform code.

3. **Module**: A module is a reusable and encapsulated unit of Terraform code. Modules allow you to package infrastructure configurations, making it easier to maintain, share, and reuse them across different parts of your infrastructure. Modules can be your own creations or come from the Terraform Registry, which hosts community-contributed modules.

4. **Configuration File**: Terraform uses configuration files (often with a .tf extension) to define the desired infrastructure state. These files specify providers, resources, variables, and other settings. The primary configuration file is usually named main.tf, but you can use multiple configuration files as well.

5. **Variable**: Variables in Terraform are placeholders for values that can be passed into your configurations. They make your code more flexible and reusable by allowing you to define values outside of your code and pass them in when you apply the Terraform configuration.

6. **Output**: Outputs are values generated by Terraform after the infrastructure has been created or updated. Outputs are typically used to display information or provide values to other parts of your infrastructure stack.

7. **State File**: Terraform maintains a state file (often named terraform.tfstate) that keeps track of the current state of your infrastructure. This file is crucial for Terraform to understand what resources have been created and what changes need to be made during updates.
8. **Plan**: A Terraform plan is a preview of changes that Terraform will make to your infrastructure. When you run terraform plan, Terraform analyzes your configuration and current state, then generates a plan detailing what actions it will take during the apply step.
9. **Apply**: The terraform apply command is used to execute the changes specified in the plan. It creates, updates, or destroys resources based on the Terraform configuration.

10. **Workspace**: Workspaces in Terraform are a way to manage multiple environments (e.g., development, staging, production) with separate configurations and state files. Workspaces help keep infrastructure configurations isolated and organized.

11. **Remote Backend**: A remote backend is a storage location for your Terraform state files that is not stored locally. Popular choices for remote backends include Amazon S3, Azure Blob Storage, or HashiCorp Terraform Cloud. Remote backends enhance collaboration and provide better security and reliability for your state files.

Install Terraform:

Windows

Installing Terraform on Windows

Here are the steps to install Terraform on Windows:

1. Download Terraform:

- Open your web browser and go to the official Terraform download page: https://www.terraform.io/downloads.html
- Find the appropriate Windows version (usually windows_amd64 for 64-bit systems).
- Click the download link to download the ZIP archive.

2. Extract the ZIP Archive:

- Once the download is complete, locate the ZIP file (e.g., terraform_1.7.5_windows_amd64.zip).
- Right-click on the ZIP file and select "Extract All..."
- Choose a destination folder to extract the contents (e.g., C:\Program Files\Terraform).
- Click "Extract".

3. Add Terraform to your PATH:

- To run Terraform from any command prompt window, you need to add the Terraform executable to your system's PATH environment variable.
- **Open Environment Variables Settings:**
  - Right-click on the Start button and select "System".
  - Click on "Advanced system settings".
  - In the "System Properties" dialog, click on the "Environment Variables..." button.
- **Edit the PATH Variable:**
  - In the "Environment Variables" dialog, under "System variables", find the "Path" variable and click "Edit...".
  - Click "New".
  - Enter the path to the directory where you extracted the Terraform executable (e.g., C:\Program Files\Terraform).
  - Click "OK" to close all the dialog boxes.

4. Verify the Installation:

- Open a new Command Prompt window (or PowerShell).
- Type terraform -version and press Enter.
- If Terraform is installed correctly, you will see the Terraform version number displayed.

Example:

terraform -version
Terraform v1.7.5
on windows_amd64

awscli Installation on windows:

For Windows:

1. Download the latest AWS CLI MSI installer from the official AWS documentation:
   - https://aws.amazon.com/cli/
2. Execute the installer and follow the installation wizard.
3. To verify installation, open Command Prompt and run:

   aws --version

awscli Configuration

After successful installation, configure the AWS CLI by supplying the necessary credentials:

aws configure

You will be prompted to enter:

- **AWS Access Key ID**
- **AWS Secret Access Key**
- **Default region name** (e.g., us-east-1)
- **Default output format** (json, yaml, table, etc.)

The credentials are stored in the ~/.aws/credentials and ~/.aws/config files by default.

- **Steps to Create AWS Access and Secret Key**

1. Sign in to AWS Console

- Go to: https://console.aws.amazon.com/
- Log in as a user with **IAM privileges** (or as the root user, though this is not recommended for day-to-day use).

2. Navigate to IAM (Identity and Access Management)

- In the AWS Management Console, search for **"IAM"**.
- Click **Users** from the left-hand menu.

3. Select or Create a User

Option A: Use an existing user

- Click the username.
- Go to the **"Security credentials"** tab.

Option B: Create a new user

1. Click **Add users**.
2. Enter a username (e.g., terraform-user).
3. Select **Programmatic access** (this is what gives you Access and Secret keys).
4. Click **Next** to attach permissions:
    - Attach existing policies like AmazonEC2FullAccess, AmazonS3FullAccess, etc., or use AdministratorAccess (use cautiously).
5. Finish user creation.

4. Generate Access Keys

- Under the **"Security credentials"** tab, click **Create access key**.
- Choose the use case (e.g., **CLI**).
- Click **Next**, and then **Create access key**.
- You'll be shown:
    - **Access Key ID**
    - **Secret Access Key** (only shown once)

Important: Save the secret key securely. You won't be able to view it again. You can download a .csv file at this step.

1. Simple Terraform Project to Launch an EC2 Instance

Step 1: Create a Project Folder with name as terraform on your desktop.

Step 2: Create the main.tf File

Create a file named main.tf and paste the following code:

```
provider "aws" {
   region = "us-west-2"
}
resource "aws_instance" "ec2_machine" {
  ami = "ami-07b0c09aab6e66ee9"
  instance_type = "t2.micro"
                                                    //  count=4
  tags ={
    Name = "Terra EC2"
  }
}
```

Step 3: Initialize Terraform: execute following command on VS code terminal :

terraform init

This sets up Terraform in the folder and downloads necessary provider plugins.

Step 4: Preview the Plan

terraform plan

This shows what Terraform will create.

Step 5: Apply the Configuration

terraform apply

When prompted, type yes to confirm. Terraform will create the EC2 instance.

Step 6: Verify

Go to your **AWS Console → EC2 → Instances**, and you'll see the instance running.

Step 7: Destroy When Done (to avoid charges)

terraform destroy

Confirm with yes.

b. **Terraform configuration** to create an AWS S3 bucket:

1. Make sure you have a text file ready:

Create a local file named sample.txt in the current directory

Sample1.txt

Hellow..welcome to terraform.

2. Create a main.tf file:

```
provider "aws" {
   region = "us-west-2"
}
resource "aws_instance" "ec2_machine" {
  ami = "ami-07b0c09aab6e66ee9"
  instance_type = "t2.micro"
  count=4

  tags ={
    Name = "Terra EC2"
  }
}

resource "aws_s3_bucket" "demo_bucket" {
   bucket = "my-unique-s3-bucket-2025-upload-demo" (This name is globally unique, so change the name while creating s3-bucket)
  tags = {
    Name = "upload-demo"
  }
}

resource "aws_s3_bucket_object" "text_file" {
  bucket = aws_s3_bucket.demo_bucket.bucket
  key    = "sample1.txt"
  source = "./sample1.txt"
}
```

How to Run

1. Initialize:

   - terraform init

2. Apply:

   - terraform apply

3. Confirm yes when prompted.

Result

After applying, sample.txt will be uploaded to the specified S3 bucket. You can verify in the AWS Console > S3.

3. Deploy a Scalable Web Server on AWS Using Terraform

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_security_group" "web_sg" {
  name        = "web_sg"
  description = "Allow HTTP inbound traffic"

  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"]
  }
}

resource "aws_instance" "web_server" {
  ami           = "ami-07b0c09aab6e66ee9"
  instance_type = "t2.micro"
  security_groups = [aws_security_group.web_sg.name]

  user_data = <<-EOF
              #!/bin/bash
              yum update -y
              yum install -y httpd
              systemctl start httpd
              systemctl enable httpd
              echo "<h1>Deployed via Terraform</h1>" > /var/www/html/index.html
              EOF

  tags = {
    Name = "TerraformWebServer"
  }
}
```

Save the above code to a file named `main.tf`.

In the same directory, run:

```
terraform init
terraform apply
```

Confirm the prompt to provision resources.

Once complete, go to your AWS EC2 dashboard to find the instance's public IP and test it in a browser (`http://<public-ip>`).