Experiment 7: Build and Manage Cloud Infrastructure Using Terraform

Learning Objective: Students should be able to understand how to use Terraform to build, deploy, and manage cloud infrastructure as code (IaC).

Tools: Terraform, AWS/GCP/Azure, VS Code, CLI

Theory:

Terraform is an open-source Infrastructure as Code (IaC) tool that allows developers to define and provision infrastructure using declarative configuration files. It supports multiple cloud providers such as AWS, GCP, and Azure, enabling automation of resource deployment, scaling, and management.

In this experiment, we will use Terraform to provision cloud resources, such as virtual machines (EC2 instances), storage, and networking components. The process involves writing Terraform configuration files, initializing Terraform, applying the configuration, and managing infrastructure updates and destruction.

- **Automation**: Automated infrastructure provisioning.
- **Version Control**: Infrastructure configurations can be versioned and tracked.
- **Consistency**: Ensures that the infrastructure is consistently deployed in any environment.
- **Collaboration**: Team members can collaborate on infrastructure design and configuration just like on any software codebase.

Key Features of Terraform:

1. **Declarative Configuration**:
   - Terraform uses a declarative configuration language known as **HashiCorp Configuration Language (HCL)**. This allows users to define their infrastructure requirements (such as compute resources, storage, and networking components) in simple configuration files.
   - The focus is on defining the "what" rather than the "how" – for example, specifying the number of virtual machines, type of storage, or the networking rules without worrying about the individual commands needed to create these resources.
2. **Multi-Cloud Support**:
   - Terraform is designed to work with a wide variety of cloud providers. It supports **AWS**, **Google Cloud Platform (GCP)**, **Microsoft Azure**, and many others. This makes it ideal for managing multi-cloud environments with a single set of configurations.
   - By defining resources using Terraform, you can ensure that the same infrastructure code can be applied across different cloud environments, enabling seamless hybrid-cloud or multi-cloud architectures.
3. **Automation of Infrastructure**:
   - Terraform automates the entire lifecycle of infrastructure management. This includes provisioning new resources, scaling infrastructure, updating configurations, and decommissioning resources.
   - With Terraform, you can automate the provisioning of infrastructure, enabling teams to deploy cloud resources consistently, reliably, and at scale, with minimal human intervention.
   - For example, Terraform can automatically deploy multiple EC2 instances in AWS, set up networking, storage, and security rules, without having to manually execute a series of commands in the cloud provider's console.
4. **Version Control**:
   - As with software code, Terraform configurations are text-based files that can be stored in a version control system (VCS) like **Git**. This allows you to track changes, manage different versions of your infrastructure, and collaborate with team members.

- This versioning also allows for rollback to a previous state if something goes wrong, making it easier to manage infrastructure in a controlled and auditable way.

5. **Consistency**:
   - One of the major benefits of using Terraform is the ability to ensure consistency across different environments. Since infrastructure is defined as code, it can be replicated in different regions or even across different cloud providers.
   - By using the same Terraform configurations, you can ensure that the infrastructure is deployed exactly the same in development, staging, and production environments, reducing the risk of discrepancies and "works on my machine" issues.

Command & Output:

1. **Install Terraform and verify installation**

```
terraform -version
```

2. **Initialize a new Terraform project**

```
mkdir terraform_project && cd terraform_project
terraform init
```

3. **Create a Terraform configuration file (`main.tf`)**

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "my_vm" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}
```

4. **Initialize Terraform and apply the configuration**

```
terraform init
terraform apply -auto-approve
```

5. **Verify the created resources**

```
terraform show
```

6. **Destroy the infrastructure**

terraform destroy -auto-approve

Conclusion:

Viva Questions:

1. What is Terraform, and how does it help in infrastructure management?
2. What is Infrastructure as Code (IaC), and why is it important?
3. What are the key components of a Terraform configuration file?
4. How does Terraform differ from other IaC tools like Ansible or CloudFormation?
5. What is the purpose of the `terraform init` and `terraform apply` commands?

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance/ Learning Attitude [20%]**	
Marks Obtained				