Terraform – Student Handout

# 1. Introduction

Terraform is an open-source tool developed by HashiCorp in 2014. It enables Infrastructure as Code (IaC), allowing infrastructure to be provisioned, managed, and automated across multiple cloud providers such as AWS, Azure, GCP, and more.  
  
- Written in: Go  
- License: Mozilla Public License 2.0  
- Type: Cloud Provisioner & IaC tool  
- Key Feature: Uses declarative configuration to define the desired state of infrastructure.

# 2. The Problem – Why Terraform?

Challenges in Infrastructure Automation:  
- Each cloud provider has its own console/portal for creating resources.  
- Difficult to maintain multi-cloud environments consistently.  
- Traditional Configuration Management tools (Puppet, Chef, Ansible) focus on applications & services running inside VMs, not on infrastructure itself.  
  
Infrastructure includes:  
- Networks & Subnets  
- Firewalls  
- Load Balancers  
- Storage  
- Public IPs  
- DNS entries, etc.

# 3. Terraform – The Solution

Terraform allows you to:  
- Treat infrastructure as code.  
- Automate creation, modification, and deletion of infrastructure.  
- Manage resources across multiple providers (AWS, Azure, GCP, VMware, SaaS APIs, etc.).  
  
Unlike Ansible or Puppet, Terraform manages the infrastructure lifecycle itself, not just the configuration inside servers.

# 4. Terraform Goals

- Unified view of resources using IaC.  
- Support for IaaS, PaaS, and SaaS.  
- Predictable & repeatable changes to infrastructure.  
- Provider-agnostic workflow (works with any API-based service).  
- Manages low-level (compute, storage, networking) and high-level (DNS, SaaS, cloud features) components.

# 5. How Terraform Works

1. Define the desired end state of infrastructure in configuration files (.tf).  
2. Terraform automatically calculates dependencies between resources.  
3. Executes actions in the correct order (e.g., create VPC before EC2, allocate Elastic IP before DNS).  
4. Maintains state files to remember the current infrastructure state.  
  
This makes Terraform a Cloud Provisioner, capable of safely building and updating environments.

# 6. Advantages of Terraform over Config Mgmt Tools

- Maintains infrastructure state (stored in a state file).  
- Reusable templates across cloud providers.  
- Easily integrates into CI/CD pipelines.  
- Supports immutable infrastructure (fresh deployment vs in-place updates).  
- Simpler change management – remembers the last applied state and applies only the difference.

# 7. Example – AWS Terraform Script

provider "aws" {  
 region = "us-west-2"  
 access\_key = "accesskey"  
 secret\_key = "secretkey"  
}  
  
resource "aws\_instance" "example" {  
 ami = "ami-8803e0f0"  
 instance\_type = "t2.micro"  
}  
  
Breakdown:  
- provider: Defines which cloud provider to use (AWS, Azure, GCP, etc.).  
- resource: Defines the actual infrastructure object (here, an EC2 instance).  
- ami: The Amazon Machine Image ID.  
- instance\_type: The VM size (t2.micro).

# 8. Terraform Workflow & Commands

File Format:  
Terraform configurations use .tf files. Example: example.tf  
  
Commands:  
1. Initialize a working directory  
 terraform init  
 - Downloads provider plugins.  
 - Prepares the environment.  
  
2. Apply changes (create/update/delete resources)  
 terraform apply  
 - Shows an execution plan.  
 - Creates/modifies/deletes infrastructure to match .tf configuration.  
  
3. Destroy infrastructure  
 terraform destroy  
 - Removes all resources defined in .tf files.

# 9. Key Takeaways

- Terraform is not a replacement for Ansible/Puppet/Chef – it complements them.  
- It handles infrastructure provisioning while config tools handle software setup inside VMs.  
- Ideal for multi-cloud automation.  
- Works with both small-scale deployments and enterprise-grade infrastructures.  
- A must-know tool in DevOps pipelines for cloud-native infrastructure.