PW SKILLS | DevOps and Cloud Computing

Introduction to Terraform: Cloud-Agnostic Infrastructure Automation

Objective

- Understand Terraform’s purpose and benefits in cloud automation.
- Learn about providers, resources, state, modules, workspaces, and backends.
- Install and set up Terraform on Linux, macOS, and Window
- Explore the Terraform workflow for infrastructure provisioning.
- Understand the basic structure of Terraform configuration files.
- Learn the role of Terraform providers in managing cloud resources.

PW SKILLS

Explaining what Terraform is and why it is cloud-agnostic.

Let’s see

- **Terraform** is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp that enables engineers to define, provision, and manage infrastructure using declarative, human-readable configuration files.
- Instead of manually configuring infrastructure through cloud consoles or command lines, Terraform lets you write code that describes the desired state of your infrastructure, which it then automatically creates or updates by interacting with cloud provider APIs and other services.

Let’s see

- Terraform is **cloud-agnostic** because it supports multiple cloud platforms and services through a plugin system called "providers." Each provider acts as an interface to a specific cloud or service API (such as AWS, Azure, Google Cloud, Kubernetes, and many others).
- This design allows you to manage resources across different clouds and environments using the same workflow and configuration language (HashiCorp Configuration Language, HCL).

PW SKILLS

Benefits of declarative syntax (HCL - HashiCorp Configuration Language) over imperative scripting.

Benefits

The benefits of declarative syntax (such as HashiCorp Configuration Language, HCL) over imperative scripting in Infrastructure as Code (IaC) include:

- Focus on Desired State, Not Steps
- Reduced Errors and Configuration Drift
- Idempotence
- Simplified Maintenance and Version Control
- Improved Productivity and Automation
- Lower Technical Debt

Pop Quiz

Q. Why is declarative syntax preferred for infrastructure as code (IaC)?

A

It automates state tracking and change management

B

It allows detailed scripting for manual changes

Pop Quiz

Q. Why is declarative syntax preferred for infrastructure as code (IaC)?

A

It automates state tracking and change management

B

It allows detailed scripting for manual changes

PW SKILLS

Compare Terraform with other IaC tools like CloudFormation and Ansible.

Let’s compare

Category	Terraform	CloudFormation	Ansible
Primary Use	Infrastructure provisioning (multi-cloud)	AWS-only infrastructure provisioning	Configuration management & app deployment
Language	Declarative (HCL)	Declarative (JSON/YAML)	Procedural (YAML playbooks)
State Management	Explicit state file (`terraform.tfstate`)	Automatic state tracking via AWS	No native state tracking

Let’s compare

Strengths	Multi-cloud orchestration, modular design	Deep AWS integration, automatic drift detection	Agentless, OS/software configuration, idempotent playbooks
Weaknesses	Manual state file management, limited Day 1+ config	AWS vendor lock-in, slower feature adoption	Limited provisioning, no native state tracking
Best For	Multi-cloud setups, complex dependencies, immutable infrastructure	AWS-centric environments, serverless architectures	Configuring servers, app deployment, hybrid cloud management

PW SKILLS

Discussing real-world use cases where Terraform automates infrastructure management.

Use cases

- Multi-Cloud Deployment: Provision Kubernetes clusters, databases, and VPCs across AWS, Azure, and GCP using a unified workflow, avoiding vendor lock-in.
- Application Infrastructure Orchestration: Automate deployment of N-tier apps (web servers, databases, APIs) with dependency management (e.g., deploying databases before app servers).
- Disaster Recovery Automation: Rebuild entire environments in alternate regions using stored Terraform state and modular code, ensuring rapid recovery.
- High-Availability Systems: Implement auto-scaling groups, load balancers, and redundant architectures for fault-tolerant web apps.
- Cost Optimization: Automate resource tagging, shutdown schedules, and right-sizing with cloud provider integrations (e.g., AWS Cost Explorer).

PW SKILLS

The role of Providers (AWS, Azure, GCP, Kubernetes, etc.) in defining infrastructure.

Role of Providers

Providers act as plugins that enable Terraform to interact with specific platforms (AWS, Azure, GCP, Kubernetes, etc.) by translating infrastructure code into API calls.

They define resources (e.g., virtual machines, databases) and data sources (e.g., existing resources) for their respective platforms, allowing Terraform to:

- Authenticate with cloud APIs using credentials.
- Manage resources (create/update/delete) via platform-specific APIs.
- Handle dependencies between resources (e.g., creating a VPC before subnets).
- Track state to map real-world infrastructure to configuration files.

PW SKILLS

Discussing Resources, the basic building blocks in Terraform (VMs, storage, networking, etc.).

Let’s discuss

Terraform Resources are the basic building blocks used to define and manage infrastructure components like virtual machines (VMs), storage, networking, and databases. They act as "recipes" that describe what you want to create, not how to create it.

- Syntax: Defined in code using a simple structure:

```
 resource "provider_type" "name" {  # e.g., aws_instance,
google_storage_bucket
   attribute1 = value1              # e.g., instance_type = "t2.micro"
   attribute2 = value2              # e.g., bucket_name = "my-app-data"
 }
```

Let’s discuss

Examples:

- VMs: aws instance, azurerm virtual machine.
- Storage: aws s3 bucket, google storage bucket.
- Networking: aws vpc, azurerm network security group.

PW SKILLS

Introducing State files and why Terraform tracks infrastructure changes.

Let’s see

Terraform uses a state file (terraform.tfstate) to track the real-world infrastructure it manages. This JSON file acts as a source of truth, storing details like resource IDs (e.g., AWS EC2 instance IDs), attributes (IP addresses, configurations), and dependencies between resources.

Why Terraform Tracks Changes:

- Map Real Infrastructure: Links resources in code to actual cloud objects (e.g., AWS S3 buckets).
- Detect Drift: Compares current infrastructure with code to identify changes (e.g., manual edits).
- Plan Accurately: Determines what to create, update, or delete during terraform apply.
- Idempotence: Ensures repeated apply commands don’t recreate existing resources.
- Performance: Avoids querying cloud APIs for every operation, speeding up workflows.

PW SKILLS

Explaining Modules for reusability and organizing Terraform code.

Let’s see

Terraform Modules are reusable, self-contained packages of Terraform configurations that group resources (e.g., VMs, networks) into logical units. They simplify code organization and enable sharing across projects.

Key Benefits:

- Reusability
- Abstraction
- Organization
- Consistency

Let’s see

Structure Example:

```
modules/
├── network/
│   ├── main.tf        # VPC, subnets
│   ├── variables.tf   # Inputs (e.g., CIDR block)
│   └── outputs.tf     # Exposed values (e.g., VPC ID)
└── compute/
    └── main.tf        # EC2 instances, security groups
```

Usage:

```
module "network" {
  source = "./modules/network"
  vpc_cidr = "10.0.0.0/16"
}
```

PW SKILLS

Discussing Workspaces for managing multiple environments (dev, staging, production).

Let’s discuss

Terraform Workspaces manage multiple environments (dev, staging, production) using isolated state files within the same codebase.

Each workspace maintains its own .tfstate, enabling environment-specific configurations (e.g., resource counts, names) via ${terraform.workspace} interpolation.

Key Features:

- Isolation: Separate state per environment to prevent conflicts.
- Code Reuse: Single configuration for all environments.

Let’s discuss

Commands:

- terraform workspace new dev
- terraform workspace select staging.

Example:

```
resource "aws_s3_bucket" "env_bucket" {
  bucket = "${terraform.workspace}-bucket" # "dev-bucket", "prod-bucket"
}
```

PW SKILLS

Explaining Backends (local vs. remote) for storing Terraform state.

Let’s see

Terraform Backends tell Terraform where to store its state file (terraform.tfstate), which keeps track of your infrastructure.

Two Main Types:

- Local: State file lives on your computer. Easy for testing, but bad for teams because it can cause conflicts.
- Remote: State file lives in a shared, secure place (like AWS S3, Azure Blob Storage, or Terraform Cloud). Good for teams because it allows collaboration, locking (prevents multiple people from changing things at once), and version history.

Let’s see

Configuration Example (AWS S3):

```
terraform {
  backend "s3" {
    bucket = "my-terraform-state-bucket"
    key    = "terraform/state"
    region = "us-east-1"
  }
}
```

PW SKILLS

Explaining system requirements and installation methods for different OS platforms.

Let’s see

Windows

- System Requirements:
- Processor: 1 GHz or faster
- RAM: 4 GB (64-bit)
- Storage: 64 GB or more
- Graphics: DirectX 12 compatible

Installation Methods:

- Bootable USB/DVD
- Windows Update
- ISO file with Media Creation Tool

Let’s see

macOS

- System Requirements:
- Apple Silicon or Intel processor
- RAM: 4 GB minimum
- Storage: 35-45 GB free

Installation Methods:

- macOS Recovery
- USB installer
- App Store download

Let’s see

Linux (e.g., Ubuntu)

- System Requirements:
- Processor: 2 GHz dual-core
- RAM: 4 GB
- Storage: 25 GB

Installation Methods:

- Live USB/DVD
- Network installation
- Dual-boot setup

PW SKILLS

Provide installation steps for Linux, macOS, and Windows, including using package managers.

Installation steps

Linux (Ubuntu Example)

Install OS:

- Download Ubuntu ISO
- Create bootable USB
- Boot from USB and follow installer

Using Package Manager (APT):

```bash
sudo apt update
sudo apt install <package-name>
```

Installation steps

macOS

Install OS:

- Download from App Store or use macOS Recovery
- Follow installation prompts

Using Package Manager (Homebrew):

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
brew install <package-name>
```

Installation steps

Windows

Install OS:

- Download Media Creation Tool from Microsoft
- Create bootable USB
- Boot and follow installer

Using Package Manager (winget):

```
winget install <package-name>
```

PW SKILLS

Discuss how to verify the installation and troubleshoot common issues.

Let’s discuss

Verify Installation:

Check Version:

- Run command like --version or -v:

```
appname --version
```

Check Path:

- Ensure it's in system PATH:

```
which appname    # Linux/macOS
where appname    # Windows
```

Run the App:

- Launch the application to ensure it opens without errors.

Let’s discuss

Troubleshoot Common Issues

- Missing Dependencies
- Permission Errors
- PATH Issues
- Corrupted Installation
- Check Logs

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

Explaining the Terraform workflow and what each command does:

Let's see

Terraform Workflow:

1. Write

- Define infrastructure in .tf files using HCL (HashiCorp Configuration Language).

2. Initialize Command:

```
terraform init
```

- Initializes the project and downloads provider plugins.

3. Plan Command:

```
terraform plan
```

- Shows what actions Terraform will take (adds, changes, destroys).

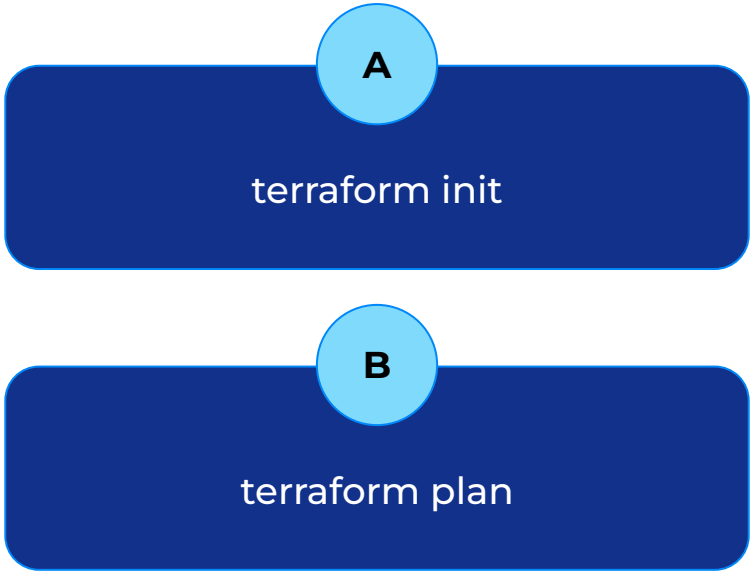
Let’s see

4. Apply Command:
```
terraform init
```
- Provisions the defined infrastructure.

5. Destroy Command:
```
terraform plan
```
- Tears down all resources defined in the config.

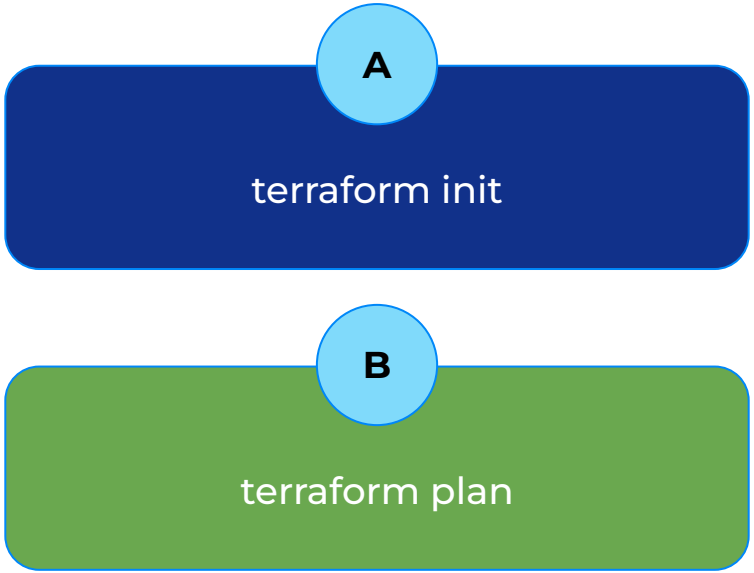
Pop Quiz

Q. Which command shows what Terraform will do before making any changes?



Pop Quiz

Q. Which command shows what Terraform will do before making any changes?



PW SKILLS

Discussing best practices for running Terraform workflows safely.

Let’s discuss

Terraform Best Practices:

1. Use Version Control
2. Use terraform plan Before Apply
3. Use Remote Backend for State
4. Enable State Locking
5. Use Variables and Workspaces
6. Limit IAM Permissions
7. Review and Test Changes in Non-Prod First

PW SKILLS

Explaining the structure of a Terraform project

Let’s see

main.tf

- Core config file
- Defines infrastructure resources (e.g., AWS EC2, S3)

variables.tf

- Declares input variables
- Allows reuse and customization

outputs.tf

- Defines output values
- Shares info like IP addresses or resource IDs

PW SKILLS

Discussing how to organize Terraform configurations efficiently.

Let’s discuss

Efficient Terraform Organization

1. Use Modules
2. Separate Environments
3. Group by Resource Type or Function
4. Use terraform.tfvars
5. Use Remote State

PW SKILLS

Explaining Terraform providers and their role in provisioning infrastructure.

Let's see

Definition:

- Plugins that let Terraform interact with APIs of cloud platforms and services (e.g., AWS, Azure, GitHub).

Role:

- They translate Terraform code into API calls to provision and manage infrastructure.

Example:

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

PW SKILLS

Discussing how Terraform interacts with different cloud services via providers.

Let’s discuss

How It Works:

- Terraform uses providers to connect to cloud APIs (like AWS, Azure, GCP).

Interaction Flow:

- You define resources in .tf files.
- The provider translates that into API calls.
- Terraform sends those API requests to create/update/delete resources

Examples:

- aws instance → AWS EC2 API
- google compute instance → GCP Compute Engine API
- azurerm resource group → Azure Resource Manager API

PW SKILLS

Explain how to use the Terraform Registry to discover official providers.

Let’s see

Visit:

- registry.terraform.io

Search:

- Use the search bar to find providers (e.g., AWS, Azure, GitHub).

Filter:

- Choose "Verified" or "Official" to ensure trusted providers.

Copy Usage:

- Each provider page shows how to declare and configure it in your main.tf.

PW SKILLS

Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this sessio
- Review your lecture notes
- Note down questions and queries regarding this session ar
  consult the teaching assistants

Thanks !