Master Infrastructure as a Code with Terraform

From Fundamentals to Advanced including Certification

Module 1 : Introduction to Infrastructure as Code (IaC) and Terraform

- Challenges with Traditional IT Infrastructure
- Types of IaC: Declarative vs. Imperative
- Why and What is Terraform?
- Overview of Terraform and its advantages over traditional methods
- Key use cases in GCP

Module 2: Getting Started with Terraform, gcloud, and IDE Setup

- Installing Terraform on Windows, macOS, and Linux
- Installing Google Cloud SDK (gcloud CLI)
- Authenticating gcloud to GCP:
  i. Using Principal Account Authentication
  ii. Using Service Account Authentication
  iii. Application-Based Authentication
- Setting up IDEs for Terraform:
  i. Visual Studio Code with Terraform plugins
  ii. JetBrains IntelliJ with Terraform support

Module 3 : Terraform Architecture

- Understanding Terraform Workflow: Initialization, Planning, Applying, Destroying
- Terraform Core Commands: `terraform init, terraform plan, terraform apply, terraform destroy`
- Terraform Configuration Syntax: Overview of how configurations are structured in .tf files
- Terraform Settings Block: Specifying provider versions, Terraform version, etc.
- Terraform Provider Block: Defining and configuring the Google Cloud provider
- Dependency Lock File: Understanding the .terraform.lock.hcl file and version locking

- Terraform Resource Block: Defining resources such as GCP VM instances, networks
- Terraform Data Block: Using data sources to reference external resources
- Handling Multiple Providers: Managing and configuring multiple cloud providers in the same Terraform configuration
- Terraform State File

Module 4 : Terraform Providers and the Provider Registry

- What are Terraform Providers?
- Understanding the Terraform Provider Registry: Exploring available providers for GCP and other platforms
- Installing and Versioning Terraform Providers: Managing provider versions and ensuring compatibility
- Explain Multi-Cloud and Provider-Agnostic Benefits: Advantages of using Terraform across multiple cloud platforms
- Writing Terraform Configuration Using Multiple Providers: Examples of using multiple providers within a single configuration
- Describe How Terraform Finds and Fetches Providers: How Terraform discovers and downloads providers from the registry

Module 5 : Terraform Settings Block

- Defining the Terraform Settings Block: The purpose and components of the terraform block
- Specifying Required Terraform Version: Ensuring compatibility by specifying the minimum required Terraform version
- Defining Required Providers: Specifying which providers are required for the configuration
- Adding Version Constraints: Setting constraints on provider versions to ensure compatibility and avoid breaking changes
- Configuring Backend Settings: Setting up remote state backends such as GCS or S3

Module 6 : Terraform Resource Block

- What is the Terraform Resource Block?: Understanding the structure and purpose of resource blocks
- Terraform Resource Block Syntax: Proper syntax and structure for defining resources
- Creating Resources: How to define and create resources like VM instances, networks, and more in GCP
- Understanding Resource Arguments and Attributes: Accessing and using resource arguments (e.g., machine types) and attributes (e.g., IP addresses)
- Resource Behavior:
  i. Create resources
  ii. Destroy resources
  iii. Update in-place resources
  iv. Destroy and re-create resources

Module 7 : Terraform Data Sources

- What are Data Sources in Terraform?: Understanding the purpose of data sources
- Using Data Sources to Fetch Existing Infrastructure: Querying existing resources in GCP
- Data Source Syntax: Proper structure and usage of data sources in Terraform
- Common Data Sources in GCP: Examples like google_compute_image, google_project
- Combining Data Sources and Resources: Using data sources to provide input for resource creation

Module 8: Terraform Meta Arguments

- What are Meta Arguments?
- Using count for Creating Multiple Instances: Dynamically creating multiple instances of a resource
- Using for_each for Iterating Over Complex Collections: Handling multiple items with more control over the resource creation process
- Using depends_on to Manage Resource Dependencies: Explicitly defining dependencies between resources
- Using lifecycle to Control Resource Lifecycle: Managing creation, update, and deletion behavior for resources
- Using provider to Override the Default Provider: Assigning a specific provider to a resource block

Module 9: Terraform Input Variables

- Using What are Input Variables?: Understanding how to use input variables for dynamic configuration
- Defining Input Variables: Syntax and structure of variable blocks
- Variable Types: Understanding string, number, list, map, and complex object types
- Providing Input Values: Multiple ways to supply values, including:
- Using default values
    i. Interactive input prompts during execution
    ii. Via CLI using -var or -var-file flags
    iii. Environment variables using TF_VAR_name
    iv. Using var-file for structured input
    v. Using a terraform.tfvars file
    vi. Auto vars in automatically loaded .auto.tfvars files
    vii. Sensitive variables to protect confidential input
- Using Default Values: Setting default values for variables
- Validating Variables: Implementing validation rules for input variables
- Variable Precedence: Understanding the order of precedence for input variables

Module 10: Terraform Output Values

- What are Output Values?: Understanding how to use outputs to display important information
- Defining Output Values: Syntax and structure for defining outputs in a configuration

- Accessing Resource Attributes: Using output values to display attributes from created resources (e.g., IP addresses, VM names)
- Using Output Values Across Modules: Passing values between modules for modular and reusable infrastructure
- Sensitive Outputs: Marking outputs as sensitive to hide confidential information
- Real-Time Implementations for Outputs

Module 11: Terraform Local Values

- What are Local Values?: Understanding the role of local values in Terraform
- Defining Local Values: Syntax and structure for creating local values
- Using Local Values for Reusability: Reducing repetition and improving maintainability in configurations
- Local Values vs. Input Variables vs. Output Values: Understanding the difference between these three types of values
- Real-Time Implementations for Local Values

Module 12: Terraform State

- What is Terraform State?: Understanding how Terraform tracks infrastructure
- Managing Local vs. Remote State: The difference between storing state locally and in a remote backend
- Setting up a GCS Backend for State Management: Using Google Cloud Storage (GCS) as a remote backend for state files
- State Locking: Ensuring consistency with state locking
- State Commands: Common state commands (terraform state list, terraform state show, etc.)
- Implementing terraform import into state file
- Real-Time Implementations for Terraform State

Module 13: Terraform Workspaces

- What are Workspaces?: Understanding how Terraform uses workspaces to manage multiple environments
- Creating and Using Workspaces: Managing environments (dev, staging, prod) with workspaces
- Switching Between Workspaces: Commands for switching and managing workspaces
- Real-Time Implementations for Workspaces

Module 14: Terraform Provisioners

- What What are Provisioners?: Understanding the purpose of provisioners in Terraform
- Types of Provisioners:
  i. Local Provisioners: Running local scripts or commands
  ii. Remote Provisioners: Executing commands on remote resources.
  iii. File Provisioners: Uploading files to remote machines
- Provisioner Connections: Establishing connections to remote instances
- Handling Resource Creation Failures with Provisioners: Understanding the on_failure behavior
- When to Use Provisioners: Best practices and avoiding common pitfalls

- Real-Time Implementations for Provisioners

Module 15: Terraform Functions

- What are Functions in Terraform?: Overview of how functions are used in Terraform
- Below are the few functions we see use cases for.
- String Functions: join(), split(), format()
- Numeric Functions: min(), max()
- Collection Functions: length(), concat()
- Filesystem Functions: file(), filebase64()
- Date and Time Functions: timestamp()
- Real-Time Implementations for Functions

Module 16: Terraform Modules

- What are Terraform Modules?: Understanding how modules help with code reusability
- Creating and Using Modules: Structuring Terraform code using modules
- Module Sources: Using local, Git, and Terraform Registry modules
- Passing Variables Between Modules: Sharing input variables across modules
- Real-Time Implementations for Modules

Module 17: Integrating with CI Server

Module 18: Certification Preparation