Contents

[1. Introduction 2](#_Toc144992632)

[1.1. Advantages of IaC 2](#_Toc144992633)

[2. Comparison of IaC Tools 3](#_Toc144992634)

[2.1. Terraform and its features. 3](#_Toc144992635)

[2.2. Differences between Configuration and Provisioning Tools 4](#_Toc144992636)

[3. Installing Terraform 7](#_Toc144992637)

[4. Terraform State Files 7](#_Toc144992638)

[5. Manage Infrastructure 8](#_Toc144992639)

[5.1. How Terraform Works 8](#_Toc144992640)

[5.2. Basic Flow 9](#_Toc144992641)

[5.2.2. AWS EC2 walk through. 10](#_Toc144992642)

[5.2.3. Providers 14](#_Toc144992643)

[5.2.4. Deploying Resources to AWS – EC2 & S3 16](#_Toc144992644)

[5. Variables and Values 18](#_Toc144992645)

[5.1. Input Variables 18](#_Toc144992646)

[5.2. Walkthrough on AWS - Launch EC2 instances in the VPC with Internet Gateway. 20](#_Toc144992647)

# Introduction

A computer screen shot of a computer

Description automatically generated

A diagram of a company

Description automatically generated with medium confidence

## Advantages of IaC

* Consistency: Deployment, Staging, Production, and all environments are controlled and the same. They are replicable.
* Version Controlled: Configuration files can be tracked through git and therefore their versions can be controlled.
* Infrastructure validation before deployment: we can get to know what the infrastructure would look like before deployment. This is enabled by static code analysis and preview how the infrastructure would look like.
* Speed: You can deploy things at greater speed with complete script with quick set-up.
* Accountability: Version control give full transparency over all changes.
* High efficiency: reduces error.
* Reusability: configuration files can be used across different environments and modules can be used to bring in variables from different scripts.
* Minimization of costs & effort: less people used for managing and setting up infrastructure.

# Comparison of IaC Tools

We have:

* AWS CloudFormation
* Azure Resource Manager
* Google Cloud Deployment Manager

A screenshot of a computer

Description automatically generated

## Terraform and its features.

* Define and provision infrastructure using Hashicorp Configuration Language or JSON.
* Written in Go programming language.
* Released under Free and Open-source Mozilla Public License.
* Codifies APIs into declarative configuration files.
* Files are treated as code; they can be reviewed, edited and versioned.

Terraform has:

**Execution Plans**

**Resource Graphs** (guilds a graph of all the resources and their dependencies)

**Change Automation**

A group of logos

Description automatically generated

A white background with black text

Description automatically generated

## Differences between Configuration and Provisioning Tools

Mutable vs. Immutable architecture

* Mutable paradigm means that the servers are continually modified in place.
* Terraform provides support for immutable environments.
* In immutable paradigm, the servers are never modified.
* Data is carried over to the new server and once verified, the old one is decommissioned.

Procedural vs. Declarative language

* In procedural style, you write code that specifies, step by step, how to achieve some desired end state.
* Terraform encourages declarative style. You specify your desired end state and the IaC tool itself is responsible for figuring out how to achieve that state.
* Procedural code does not fully capture the state of infrastructure.
* Procedural code limits reusability.

Master and Agents

* Chef, Puppet, and SaltStack require a master server for storing the infrastructure state.
* Terraform is primarily master less.
* Terraform communicates with cloud providers using the cloud provider’s APIs.
* With Terraform, you don’t need to install any agent software as well.

State Files

* Terraform follows a stateful management approach.
* The state of the infrastructure contains thorough information about the provisioned infrastructure and its various configurations.
* When we create an infrastructure, terraform will store the config files in a state file.
* This state is stored locally by default, as “terraform.tfstate”, but it can also be stored remotely.
* State helps provides idempotence to terraform as it already knows if one resource is present and prevent it to be created again when the same configuration executes. A screen shot of a computer program

  Description automatically generated

# Installing Terraform

For Windows:

1. Install from: <https://developer.hashicorp.com/terraform/downloads>
2. Place the terraform.exe executable file in a folder of your choice.
3. Add the path to that folder in your environment variable for path.
4. Use the command prompt/cmd to check terraform version, with ```terraform version```.
5. Alternatively, use a **package manager** such as **chocolatey**  to install terraform.
   1. ```choco install terraform```
   2. ```choco install terraform –pre```

# Terraform State Files

State Files:

* Terraform stores the infrastructure objects and its values in the state file.
* This state is used by terraform to map real world resources.
* Prior to any operation, terraform does a refresh to update the state with the real infrastructure.
* State file named terraform.tfstate will get generated.
* Terraform state can contain sensitive data, depending on the resources to use.
* By default, the state file will be stored on local machine running Terraform.
* But it can be configured to use Remote Storage.
* Quite a few backends can be used to support Remote Storage.
* State helps provides idempotence to terraform.
* It allows the users to make incremental changes in the future by changing the state file to the desired state.

State Locking

* While we are adding something to the configuration file, a lock gets enabled.
* Once the update gets completed, the lock is released.
* It makes sure that no two members can do the configuration updates and changes at the same time.
* If state locking fails, Terraform will not continue.

Local State Storage

* It stores the state file on the local file system.
* Using system APIs, it will lock the state file and performs all the operations locally.
* Use of local file makes terraform usage hard among team members.
* State file stores date (including the sensitive content) in the plain-text json format.

Remote State Storage

* Terraform writes the state data to a remote data store, which can then be shared between all members of a team.
* Terraform supports storing state in Terraform cloud, HashiCorp Consul, Amazon S3, Alibaba Cloud OSS and more.
* Remote work allows you to delegate the outputs to other teams.
* Storing the state file remotely helps prevent sensitive information.
* Storing state remotely can provide better security as well.

# Manage Infrastructure

## How Terraform Works

Main Parts:

* Open-source tool which is written in Go.
* Need the binary to deploy the infrastructure.
  + In the back-end Terraform binary makes API calls to the cloud providers and leverage their services.

A diagram of a computer process

Description automatically generated

Terraform Core Responsibilities:

* IaC
* Resource state management
* Construction of the Resource Graph (builds on modern Graph Theory).
  + Graphs model the relationship between resources so operators can manage infrastructure resources IaaS, PaaS and SaaS.
* Plan execution
* Communication with the Plugin over RPC

Plugins:

* External single static binaries
* Each plugin exposes an implementation for a service provider or provisioner.
* Several Provisoners are built-in, while Providers are discovered dynamically.

A close-up of a white background

Description automatically generated

**Working**

A diagram of a diagram

Description automatically generated

* Terraform relies on CSP APIs to make the communication.
* It can invoke upstream CRUD operations.

Providers:

* Communicate with clouds and other servers; expose resources.
* Below are some providers:
  + GitLab
  + GCP
  + AWS
  + Azure
  + Rancher

## Basic Flow

```Init``` Command:

* Used to initialize a working directory containing Terraform config files.
* Terraform reads config files to determine which plugins are necessary.
* Searches and decides which plugin versions to use.
* Writes a lock file.
* Plugins behave in one of three ways:
  + Built-in provisoners
  + Providers distributed by HashiCorp
  + Third-party providers and provisioners

### 5.2.2. AWS EC2 walk through.

**Pre-requisites**:

1. AWS account
2. AWS user – make sure to keep track of or save the login details (username, password) and access key id + secret access key.
3. Download and install AWS CLI: <https://aws.amazon.com/cli/>
4. Once installed, open command prompt then run the following command:
   1. ```aws configure```
      1. Here you will need to provide the access key ID and Secret access key that should have been generated when creating the AWS user.
5. Download the Vim editor (optional) or Notepad++ (preferably keep both). You will use one as the default editor for git.

**Walk-through**

1. Create a terraform folder somewhere you want.
2. Create a file ‘aws-instance.tf’ inside that folder using an editor (e.g. notepad ++) and populate it as below:

A computer screen with white text

Description automatically generated

**Note**: that the ami image id and instance types can be found as options when launching a new EC2 instance on AWS. ***Make sure*** you are on the correct region (i.e., if you configured eu-west-2 then make sure the AMI ID you input is from the region ‘London’). Once done, save and close.

1. A screenshot of a computer program

   Description automatically generatedRun command ```terraform init``` on bash. It should look as follows:
2. We can review the infrastructure changes using the terraform plan command. On bash, run ```terraform plan```. The output should look as below – it is only a preview of the changes that is planned to be made and no actual changes have been implemented.

A screen shot of a computer

Description automatically generated

1. Next is to use the Terraform apply command. On the bash script, now run ```terraform apply```.
2. We can now use the Terraform destroy command to remove the resources. On the bash script, now run ```terraform destroy```. In the plan summary you should be able to see that one resource is set-up to be destroyed. If you are happy with the plan, type ‘yes’.

A screen shot of a computer program

Description automatically generated

Key commands:

* terraform init, terraform plan, terraform apply, terraform destroy.

### 5.2.3. Providers

Provider is a plugin used to communicate with the external services using APIs.

* Examples: AWS, GCP< Azure, OpenStack
* Examples – Heroku
* Examples – RabbitMQ, CloudFlare, PostgreSQL

Providers offers resource types and certain arguments for each types.

Provider and Resources:

* Provider Block:

A black rectangular object with blue and red text

Description automatically generated

* Resource Block:

A black background with orange text

Description automatically generated

Authentication (with AWS):

* Static credentials (best practice, do not hard code these into the config files):

A black background with orange text

Description automatically generated

* Environment Variables:

A black and white text

Description automatically generated

* Shared Credentials file:
  + A black background with white text

    Description automatically generatedAWS CLI creates the credentials file at $HOME/.aws/credentials on Linux, by default:
  + Alternatively, by configuring the AWS\_SHARED\_CREDENTIALS\_FILE environment variable.
* EC2 Role
  + Running Terraform from an EC2 instance with IAM Instance Profile using IAM Role (this is a preferred option).
  + Running Terraform on ECS or CodeBuild.

A yellow arrow pointing to the right

Description automatically generated

**Provider Meta-Arguments**

**Provider Versions**

* Providers are released on a separate rhythm from Terraform itself.
* A black background with green and white text

  Description automatically generatedWhen terraform init is run without provider version it prints a suggested version string.
* To upgrade the providers, use -upgrade flag.
* To constrain the provider version as needed, add a required\_providers block.

A computer screen with text and symbols

Description automatically generated

**Alias**

* A screen shot of a computer

  Description automatically generatedIt creates an additional provider configuration.
* List of arguments for AWS: <https://registry.terraform.io/providers/hashicorp/aws/latest/docs#argument-reference>

**Resources**

Each resource block describes one or more infrastructure objects, such as:

* Virtual networks
* Compute instances
* Higher-level components such as DNS records

Used in conjunction with providers:

A black background with text

Description automatically generated

* The resource type and name together serve as an identifier for a given resource.

**Resource type and Arguments**

A diagram of a resource type

Description automatically generated

* We can specify multiple resources using the resource meta-arguments. Example below:

A computer screen with text and numbers

Description automatically generated

Resource Behaviour:

* When Terraform creates a new infrastructure object represented by a resource block, the identifier for that real object is saved in terraform’s state.
* For already existing resources, terraform compares the actual configuration of the object with the arguments given in the configuration.

### 5.2.4. Deploying Resources to AWS – EC2 & S3

Use this documentation to support/guide you: <https://registry.terraform.io/providers/hashicorp/aws/latest/docs#argument-reference>

1. A screen shot of a computer

   Description automatically generatedSpecify a new resource aws elastic ip (aws\_eip) with load balancer (lb) in a new aws-instance-eip.tf file within the terraform folder:
2. A screenshot of a computer

   Description automatically generatedRun terraform plan and if you see that 2 to add, then run terraform apply. You should be able to see in the aws console an instance with an elastic IP has been created.
3. You can run the terraform destroy command to remove what you have created.
4. (OPTIONAL) If you wish, you can also create an AWS S3 bucket by adding the resource to the existing .tf file or creating a new one as below:

A screenshot of a computer program

Description automatically generated

# Variables and Values

## Input Variables

* To become truly shareable and useful in production, we need to parameterize the configurations.
* Source code in the shared repo can be used differently.
* Three aspects of using input variables:
  + Defining variables
  + Assigning values to variables
  + Using variables in a configuration
* Input variables are usually defined by using:
  + Name
  + Type (Optional)
  + Default Value (Optional)
* We can declare variables in a .tf file as below:

A screenshot of a computer program

Description automatically generated

* For default values specified within the variable, if no values are provided when using the variable, the default value will be used. If a separate value is provided for the variable, the default value will be overwritten. For “instance\_type” variable, no default value is provided and so one will need to be provided.
* Notice also that different variables can be provided different types.
* Type of the Variable:
  + String, Number or Boolean
  + Rich data types as list, set, map, object, or tuple.

Ways to assign Values to Variables

* Command-line flags:
  + ```terraform apply – var ‘region=eu-west-2’```
* In variables definition files:
  + We can create a file and assign variables within this file.
  + Files terraform.tfvars or any variation of \*.auto.tfvars are automatically loaded.
  + Can also specify file as terraform.tfvars.json or any file ending in .auto.tfvars.json.
  + Can use the -var-file flag to specify a file name.
    - ```terraform apply -var-file=” testing.tfvars”```
* Environment Variables:
  + Environment variables named TF\_VAR\_ followed by the name of a declared variable.
  + Can only populate string-type variables.
* UI input:
  + Terraform asks to input the values interactively, if unspecified.
* Priority of variables
  + A variable cannot be assigned values multiple times within the same source.
  + A variable can be assigned values multiple times across multiple sources.
  + Precedence order is as follows:
    - Environment variables
    - terraform.tfvars file
    - terraform.tfvars.json file
    - Any \*. auto.tfvars or \*.auto.tfvars.json files
    - Any -var and -var-file options on the command line.
* Using variables in Configuration
  + Value of the variable can be accessed from within expressions as var.<NAME>

A computer screen with white text

Description automatically generated

Custom Validation Rules

* + In addition to Type Constraints such as the type of input as string, Boolean etc.
  + Using Custom validation , we can specify arbitrary validation rules. As in the example below with a validation rule for checking ami\_id:

A screenshot of a computer program

Description automatically generated

## Walkthrough on AWS - Launch EC2 instances in the VPC with Internet Gateway.

Objectives:

1. In AWS, select region eu-west-2.
2. Create VPC & subnet
3. Spin up a t2.micro instance.
4. Provide two 10 GB EBS storage devices.
5. Add a tag with Name as Variables-1
6. Use the default security group and launch the instance.

To accomplish this objective, initialise and apply terraform on the following repository: