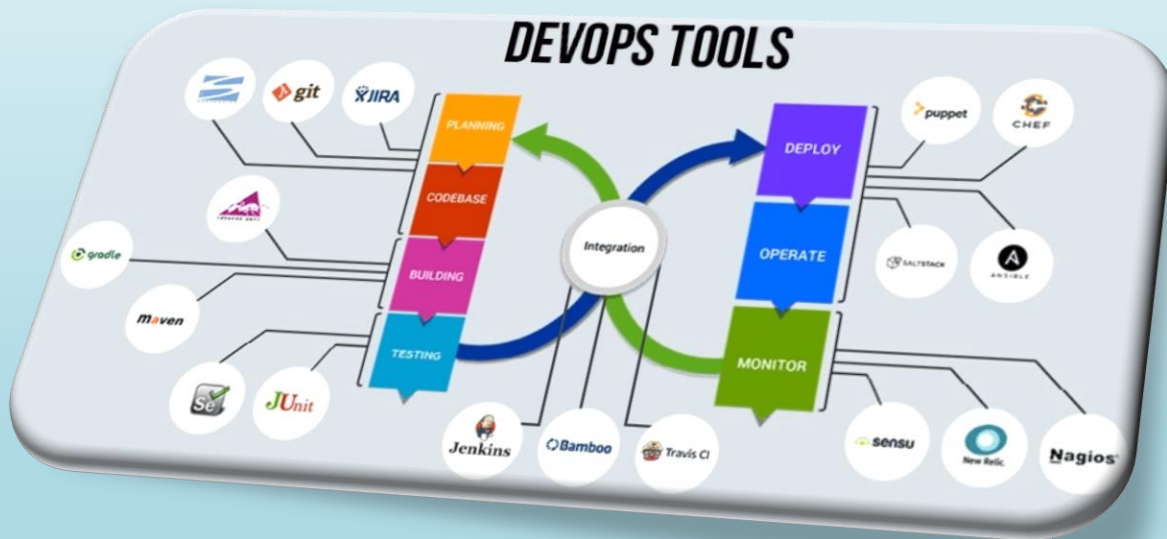





CLOUD TRAIN

ACCELERATE YOUR GROWTH

Infrastructure as a Code [Terraform]



Agenda

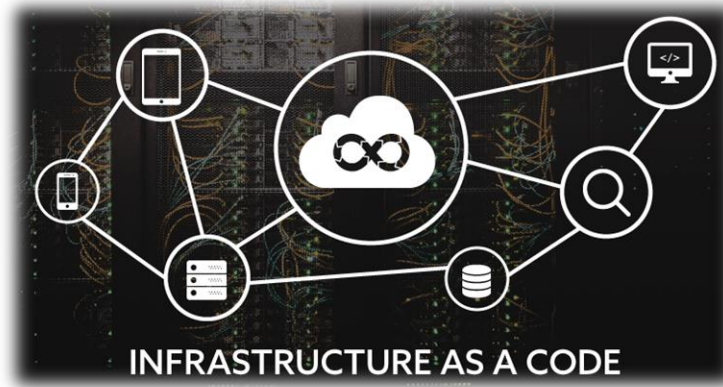
- 
- WHAT IS IAAC?
 - INTRODUCTION TO TERRAFORM
 - TERRAFORM ARCHITECTURE
 - TERRAFORM LIFECYCLE
 - INSTALLING TERRAFORM
 - TERRAFORM STATE
 - TERRAFORM VARIABLES
 - TERRAFORM MODULES

What is IaaS?

What is IaasC?

Infrastructure as a code (IaaSC) is the process of managing and provisioning computer data centres through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

The IT infrastructure managed by this process comprises both physical equipment, such as bare-metal servers, as well as virtual machines, and associated configuration resources. The definitions may be in a version control system. It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.



Why IaaS?

Confidence

Repeatability

Troubleshooting

Disaster Recovery

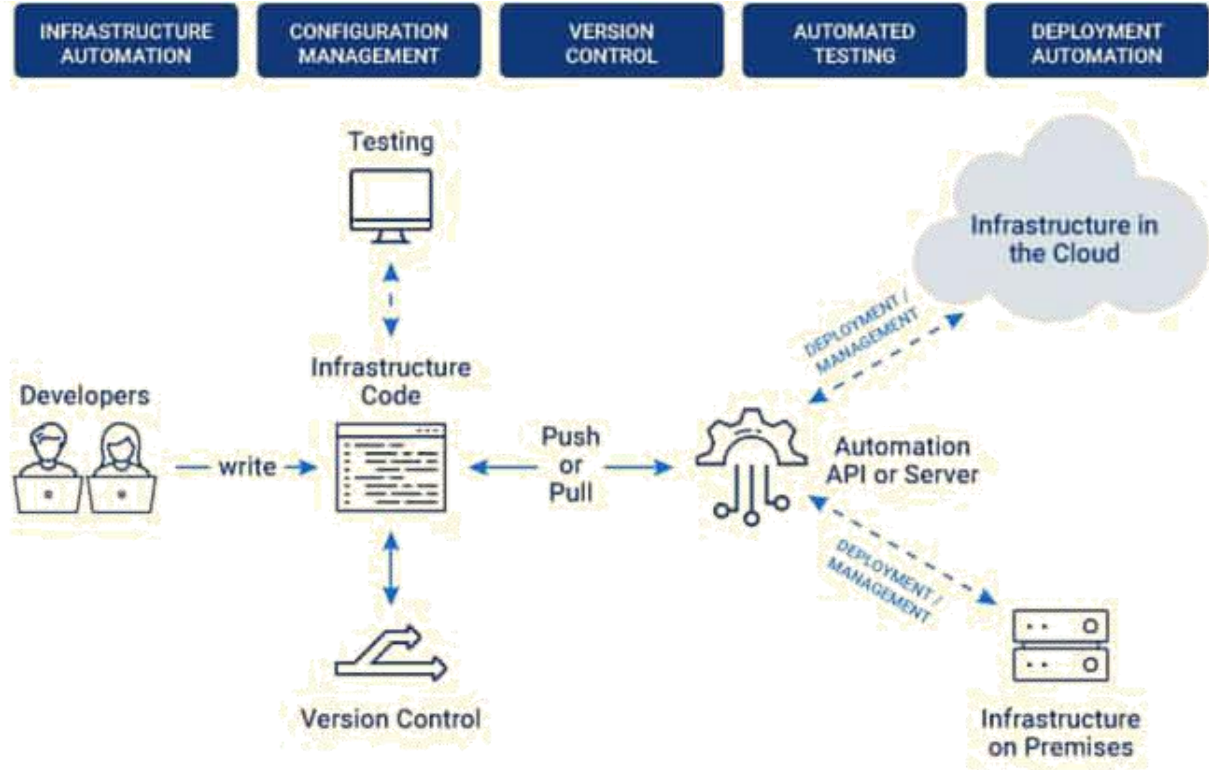
Auditability

Visibility

Portability

Security

What is IaaS?



Introduction to Terraform

Introduction to Terraform

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions.

- Configuration files describe to Terraform the components needed to run a single application or your entire data center.
- Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure.
- As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.
- The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.



HashiCorp

Terraform

Write, Plan, and Create Infrastructure as Code

Terraform Architecture

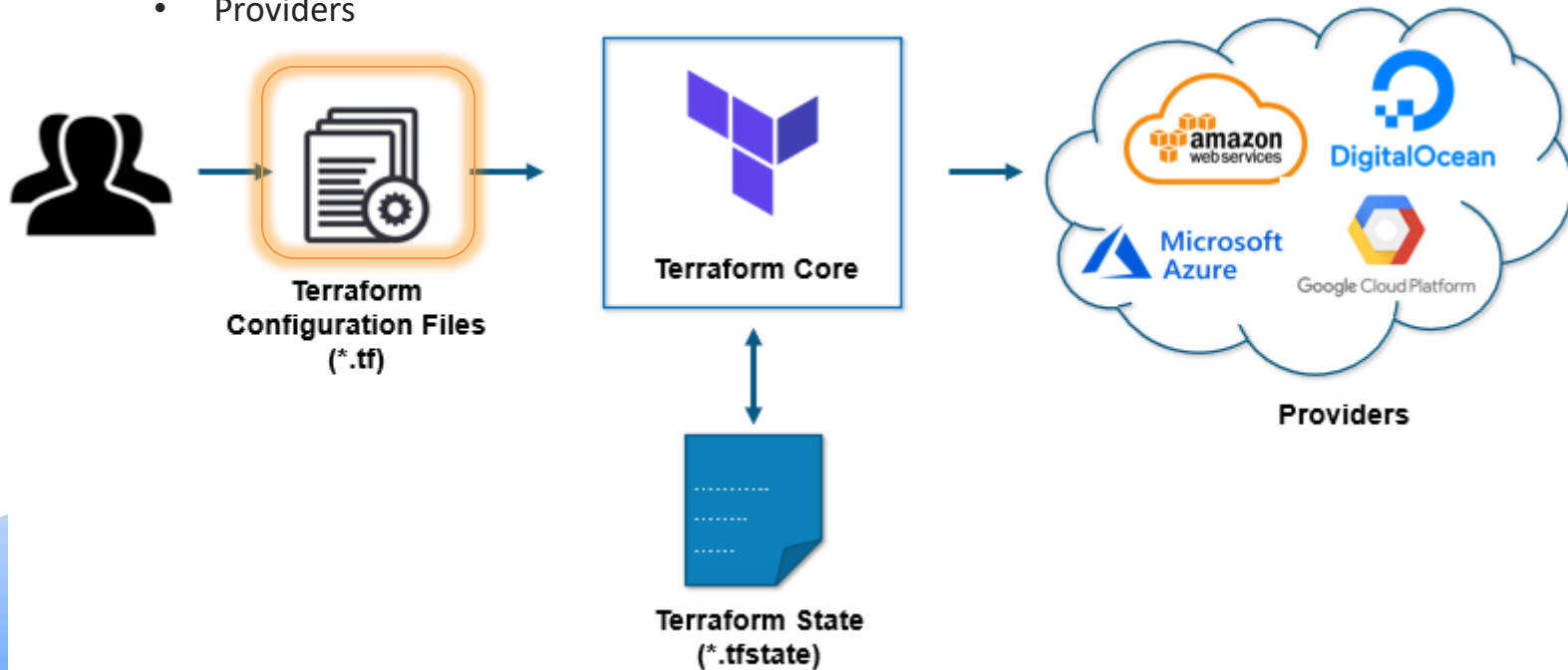
Terraform Core Concepts

- **Variables:** Also used as input-variables, it is key-value pair used by Terraform modules to allow customization.
- **Provider:** It is a plugin to interact with APIs of service and access its related resources.
- **Module:** It is a folder with Terraform templates where all the configurations are defined
- **State:** It consists of cached information about the infrastructure managed by Terraform and the related configurations.
- **Resources:** It refers to a block of one or more infrastructure objects (compute instances, virtual networks, etc.), which are used in configuring and managing the infrastructure.
- **Data Source:** It is implemented by providers to return information on external objects to terraform.
- **Output Values:** These are return values of a terraform module that can be used by other configurations.
- **Plan:** It is one of the stages where it determines what needs to be created, updated, or destroyed to move from real/current state of the infrastructure to the desired state.
- **Apply:** It is one of the stages where it applies the changes real/current state of the infrastructure in order to move to the desired state.

Terraform Architecture

Terraform has two main components that make up its architecture:

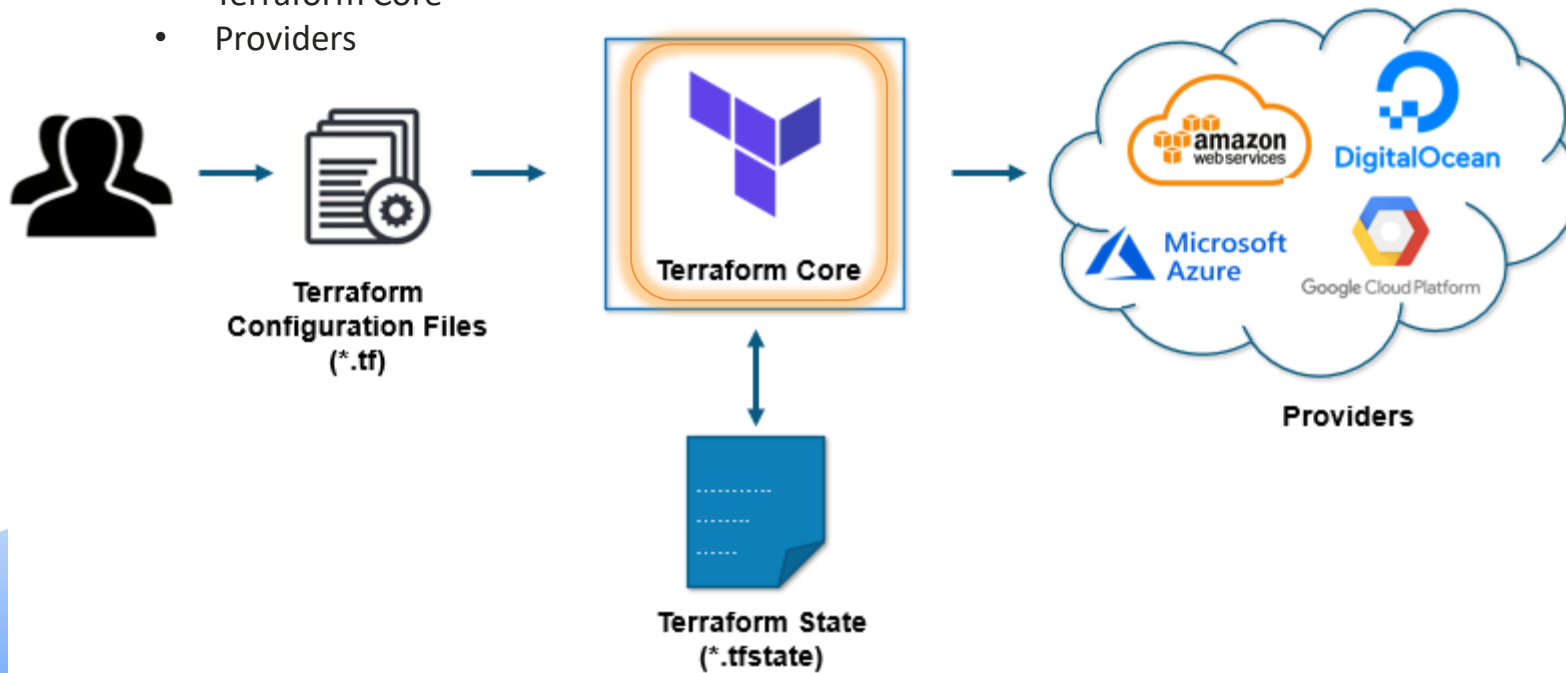
- Terraform Core
- Providers



Terraform Architecture

Terraform has two main components that make up its architecture:

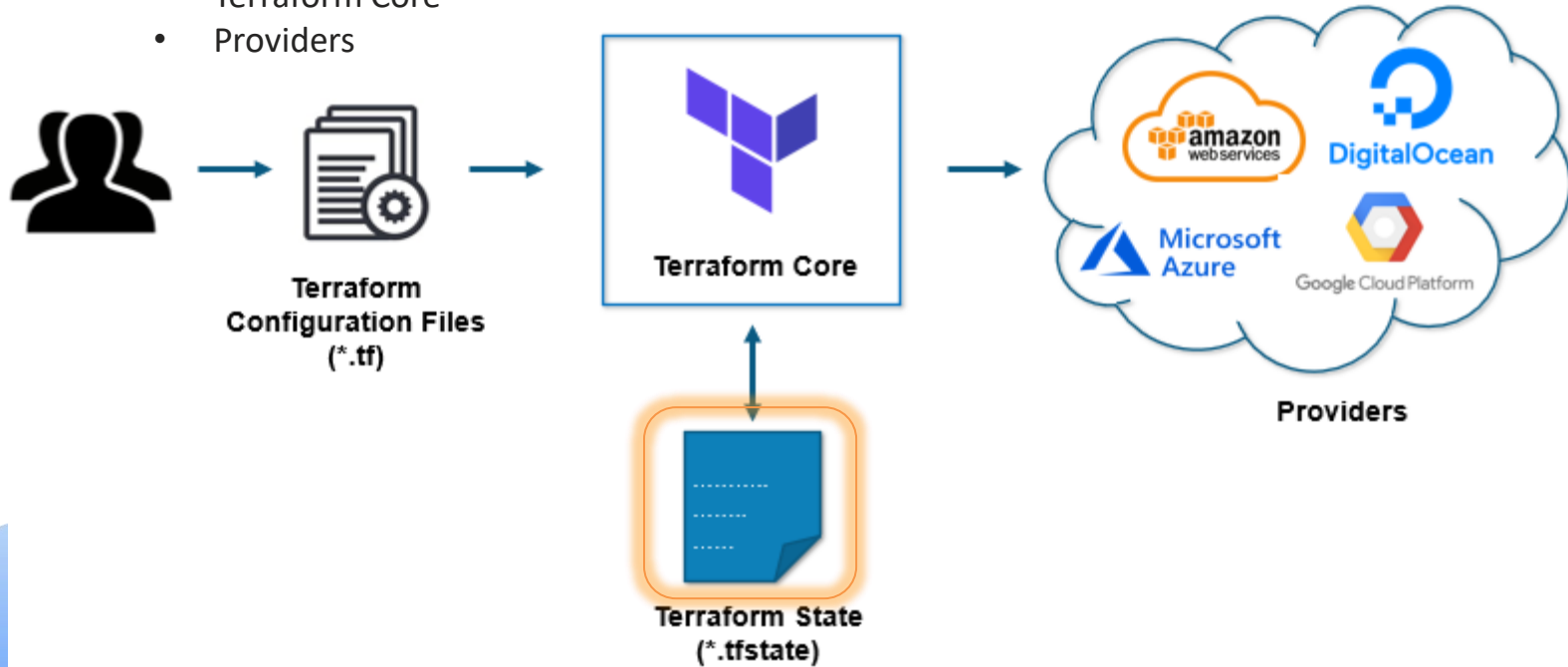
- Terraform Core
- Providers



Terraform Architecture

Terraform has two main components that make up its architecture:

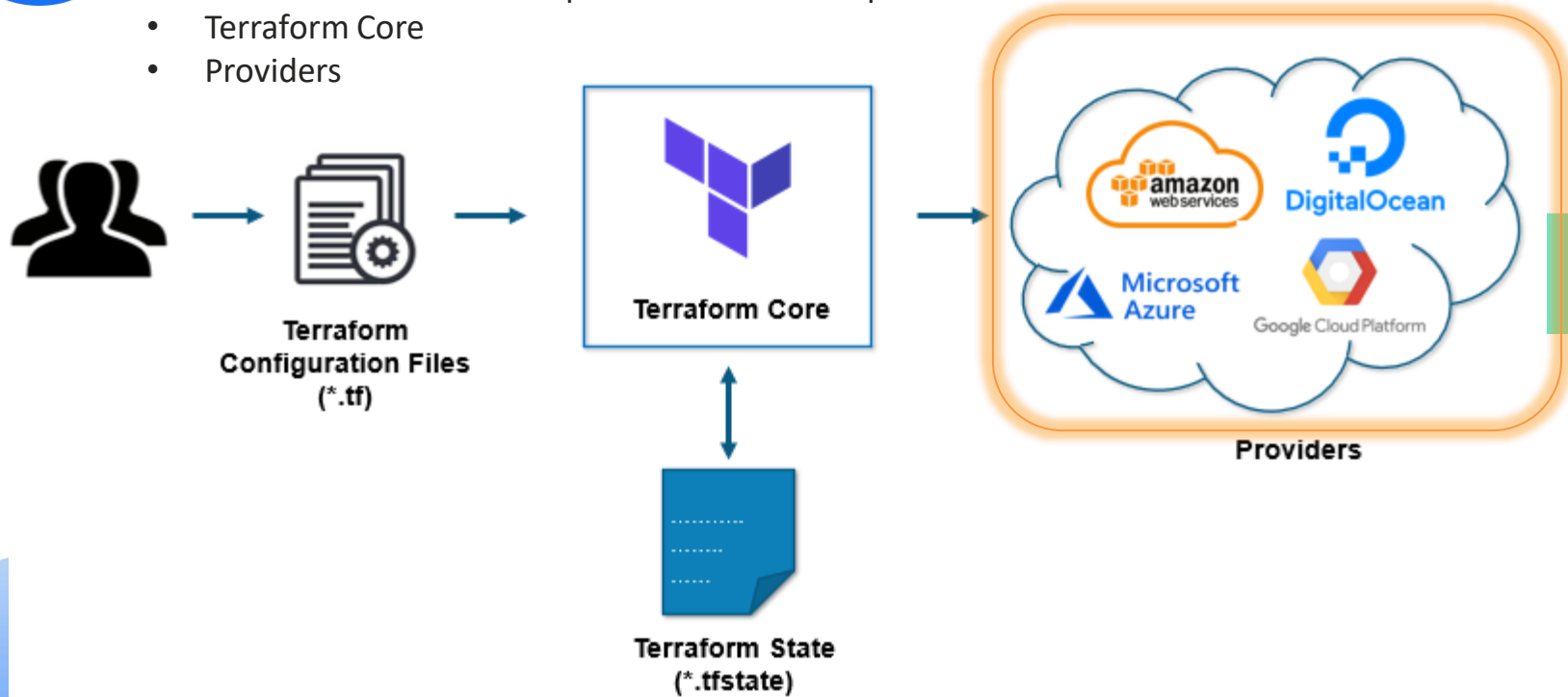
- Terraform Core
- Providers



Terraform Architecture

Terraform has two main components that make up its architecture:

- Terraform Core
- Providers



Terraform Lifecycle

Terraform Actions



Terraform Actions

Init

Plan

Apply

Destroy

Terraform init initializes the working directory which consists of all the configuration files.



Terraform Actions

Init

Plan

Apply

Destroy

Terraform plan is used to create an execution plan to reach a desired state of the infrastructure. Changes in the configuration files are done in order to achieve the desired state.



Terraform Actions

Init

Plan

Apply

Destroy

Terraform apply then makes the changes in the infrastructure as defined in the plan, and the infrastructure comes to the desired state.



Terraform Actions

Init

Plan

Apply

Destroy

Terraform destroy is used to delete all the old infrastructure resources, which are marked tainted after the apply phase



Installing Terraform

Terraform Installation

```
$ wget https://releases.hashicorp.com/terraform/0.13.0/terraform\_0.13.0\_linux\_amd64.zip
```

```
$ unzip terraform_0.13.0_linux_amd64.zip
```

```
$ sudo mv terraform /usr/local/bin/
```

```
$ terraform -v
```

Terraform State

Terraform State

The terraform state command is used for advanced state management. As your Terraform usage becomes more advanced, there are some cases where you may need to modify the Terraform state. Rather than modify the state directly, the terraform state commands can be used in many cases instead.

This command is a nested subcommand, meaning that it has further subcommands. These subcommands are listed to the left.

Usage:

terraform state <subcommand> [options] [args]

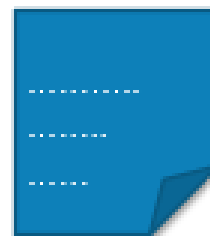
Terraform State

list

show

pull

shows the resource addresses for every resource Terraform knows about in a configuration, optionally filtered by partial resource address.



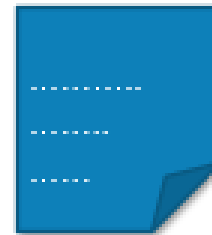
Terraform State

list

show

pull

displays detailed state data about one resource.



Terraform State

list

show

pull

Pull current state and output to stdout



Terraform Variables

Terraform Variables

How to Assign Values from the Root Module

1. Through commands on the CLI
2. The “.tfvars” file
3. Using Environment variables

Input Variables

String

List

Map

Boolean

Strings mark a single value per structure and are commonly used to simplify and make complicated values more user-friendly. Below is an example of a string variable definition.

A string variable can then be used in resource plans. Surrounded by double quotes, string variables are a simple substitution such as the example underneath.

```
variable "template" {  
  type = string  
  default = "Hello World"  
}
```

```
storage = var.template
```

Input Variables

String

List

Map

Boolean

List work much like a numbered catalogue of values. Each value can be called by their corresponding index in the list. Here is an example of a list variable definition.

Lists can be used in the resource plans similarly to strings, but you'll also need to denote the index of the value you are looking for.

```
variable "users" {  
  type    = list  
  default = ["root", "user1", "user2"]  
}
```

```
username = var.users[0]
```

Input Variables

String

List

Map

Boolean

Maps are a collection of string keys and string values. These can be useful for selecting values based on predefined parameters such as the server configuration by the monthly price.

You can access the right value by using the matching key. For example, the variable below would set the plan to "1xCPU-1GB"

```
variable "plans" {  
  type = map  
  default = {  
    "CONF1" = "1xCPU-1GB"  
    "CONF2" = "2xCPU-4GB"  
  }  
}
```

```
plan = var.plans["5USD"]
```


Input Variables

List

Map

Boolean

String

The last of the available variable type is boolean. They give the option to employ simple true or false values. For example, you might wish to have a variable that decides when to generate the root user password on a new deployment.

By default, the value is set to false in this example. However, you can overwrite the variable at deployment by assigning a different value in a command-line variable.

```
variable "set_password" {  
    default = false  
}
```

```
create_password = var.set_password
```

Output Variables

Output variables provide a convenient way to get useful information about your infrastructure. As you might have noticed, much of the server details are calculated at deployment and only become available afterwards. Using output variables you can extract any server-specific values including the calculated details.

Configuring output variables is really quite simple. All you need to do is define a name for the output and what value it should represent. For example, you could have Terraform show your server's IP address after deployment with the output variable below.

Alternatively, output variables can also be called on-demand using terraform output command.

```
output "public_ip" {  
    value = cloud_server.server_name.network_interface[0].ip_address  
}
```

Terraform Modules

Terraform Modules

A Terraform module is a set of Terraform configuration files in a single directory. Even a simple configuration consisting of a single directory with one or more `.tf` files is a module. When you run Terraform commands directly from such a directory, it is considered the **root module**. So in this sense, every Terraform configuration is part of a module. You may have a simple set of Terraform configuration files such as:

```
.  
├── LICENSE  
├── README.md  
├── main.tf  
├── variables.tf  
└── outputs.tf
```

In this case, when you run terraform commands from within the `minimal-module` directory, the contents of that directory are considered the root module.

Types of Modules

Calling modules

Terraform commands will only directly use the configuration files in one directory, which is usually the current working directory. However, your configuration can use module blocks to call modules in other directories. When Terraform encounters a module block, it loads and processes that module's configuration files.

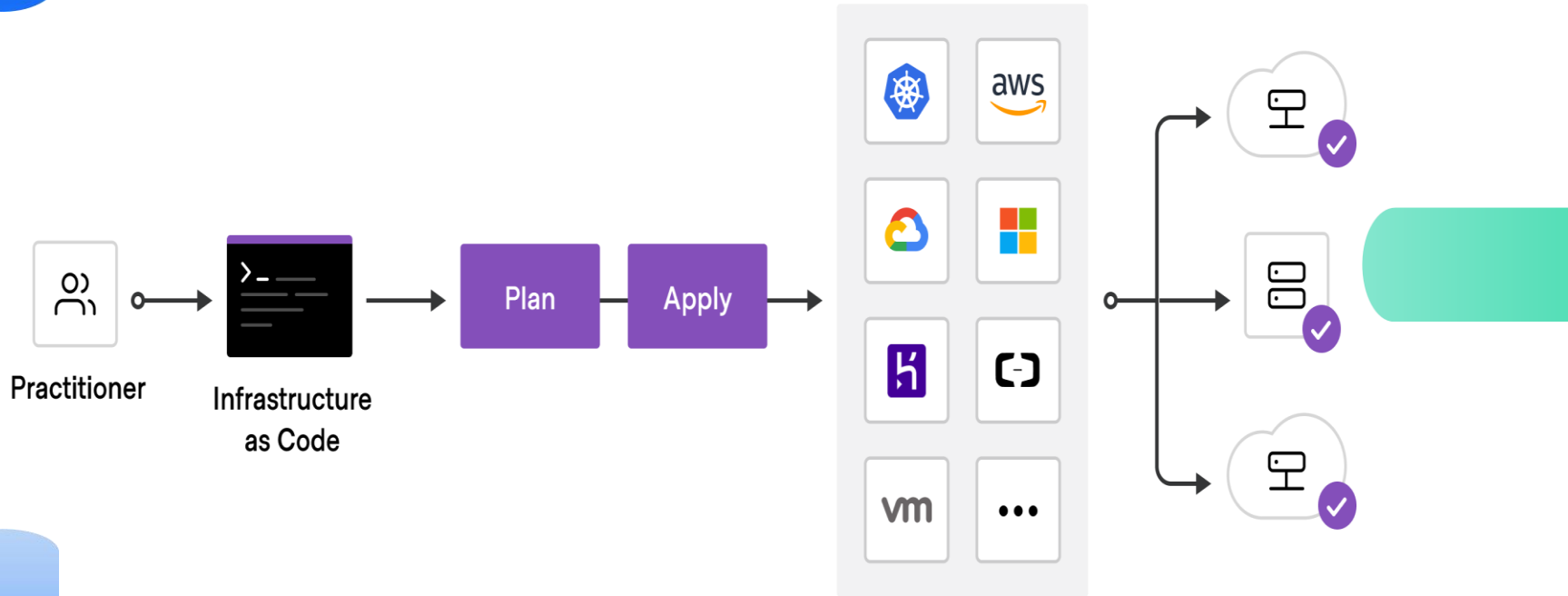
A module that is called by another configuration is sometimes referred to as a "child module" of that configuration.

Local and remote modules

Modules can either be loaded from the local filesystem, or a remote source. Terraform supports a variety of remote sources, including the Terraform Registry, most version control systems, HTTP URLs, and Terraform Cloud or Terraform Enterprise private module registries.

Hands On: Infra Deployment with Terraform

Terraform Demo



**Got
queries
or need
more
info?**

Contact us

TO ACCELERATE YOUR CAREER GROWTH

For questions and more details:

please call @ [+91 98712 72900](tel:+919871272900) or

visit <https://www.thecloudtrain.com/> or

email at join@thecloudtrain.com or

WhatsApp us >> 