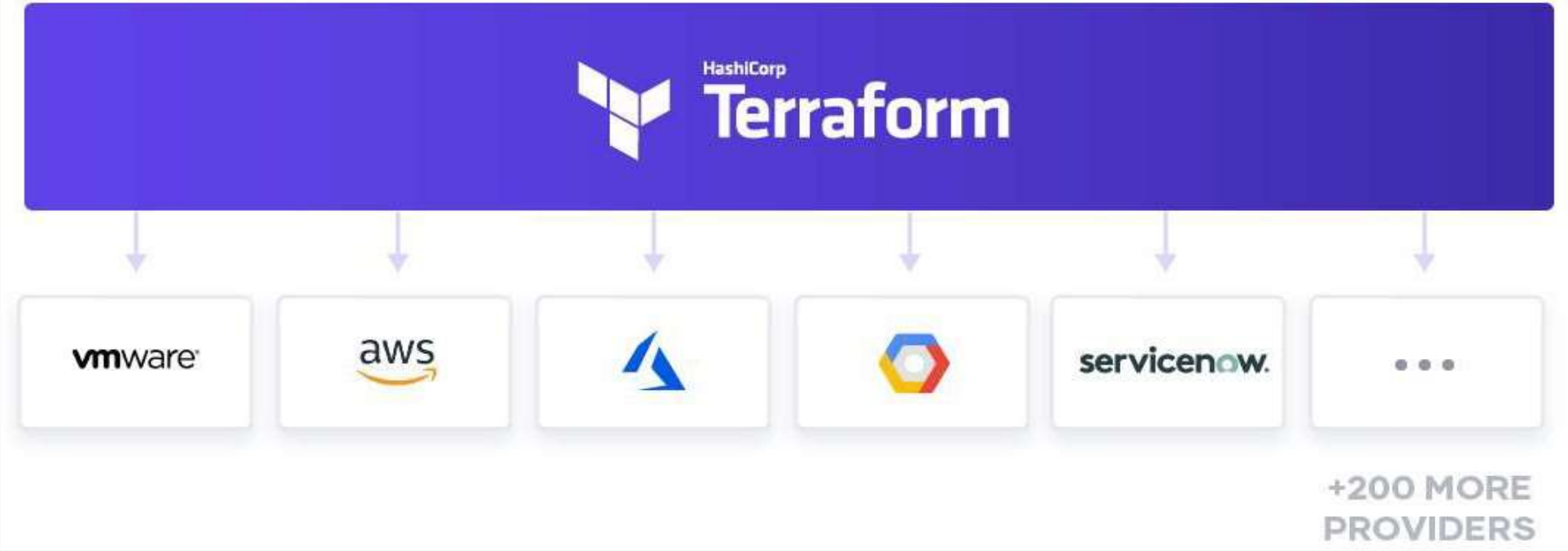
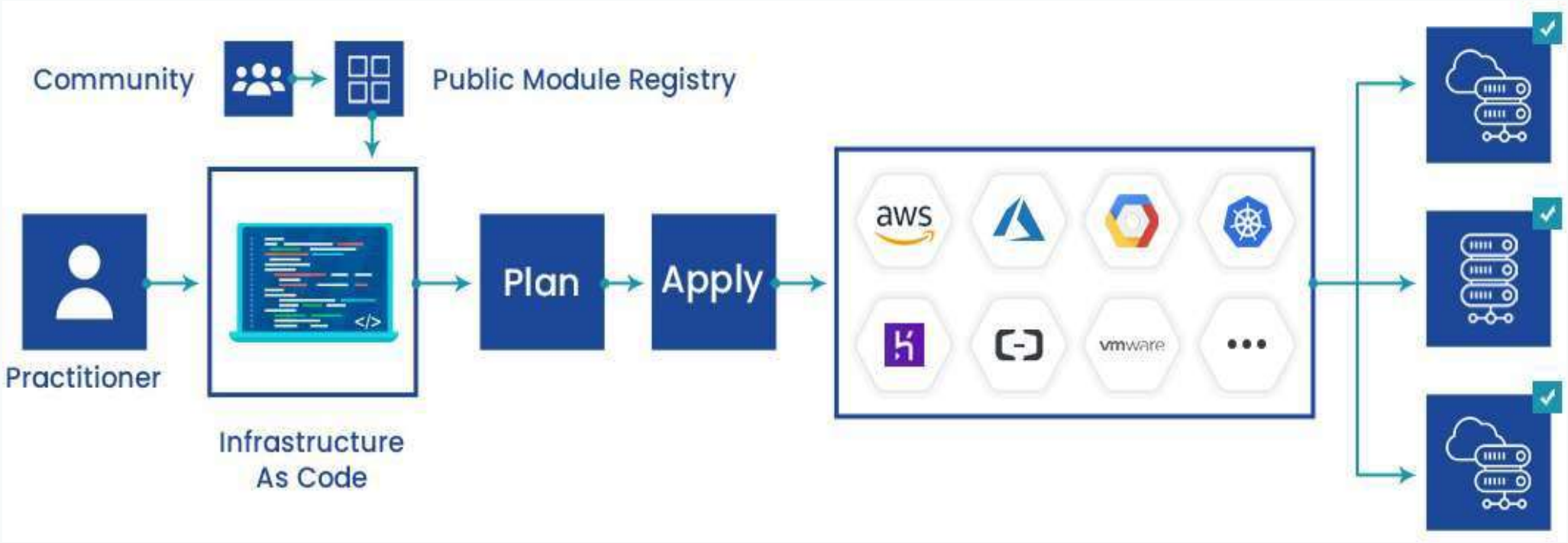
Terraform

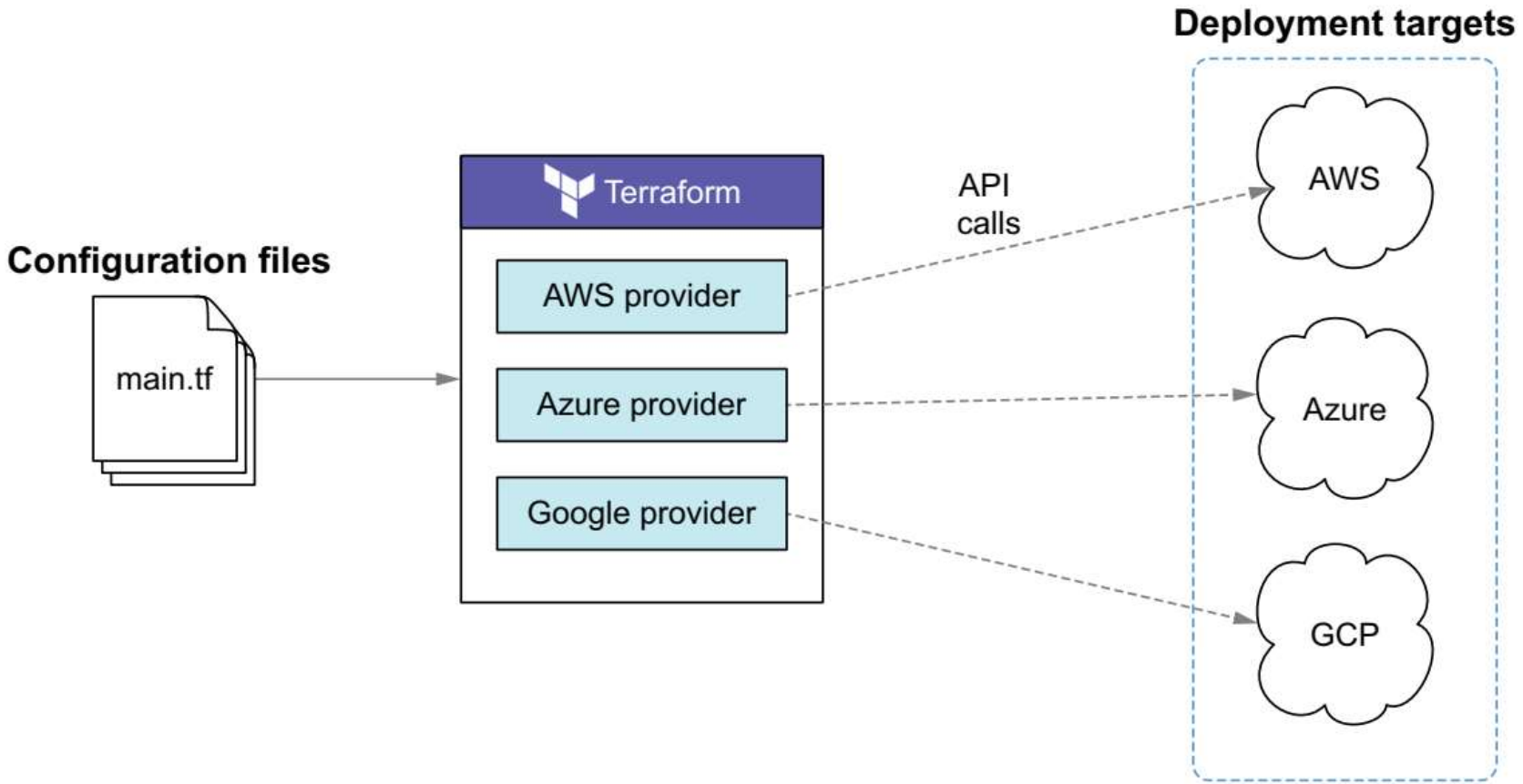
Introduction



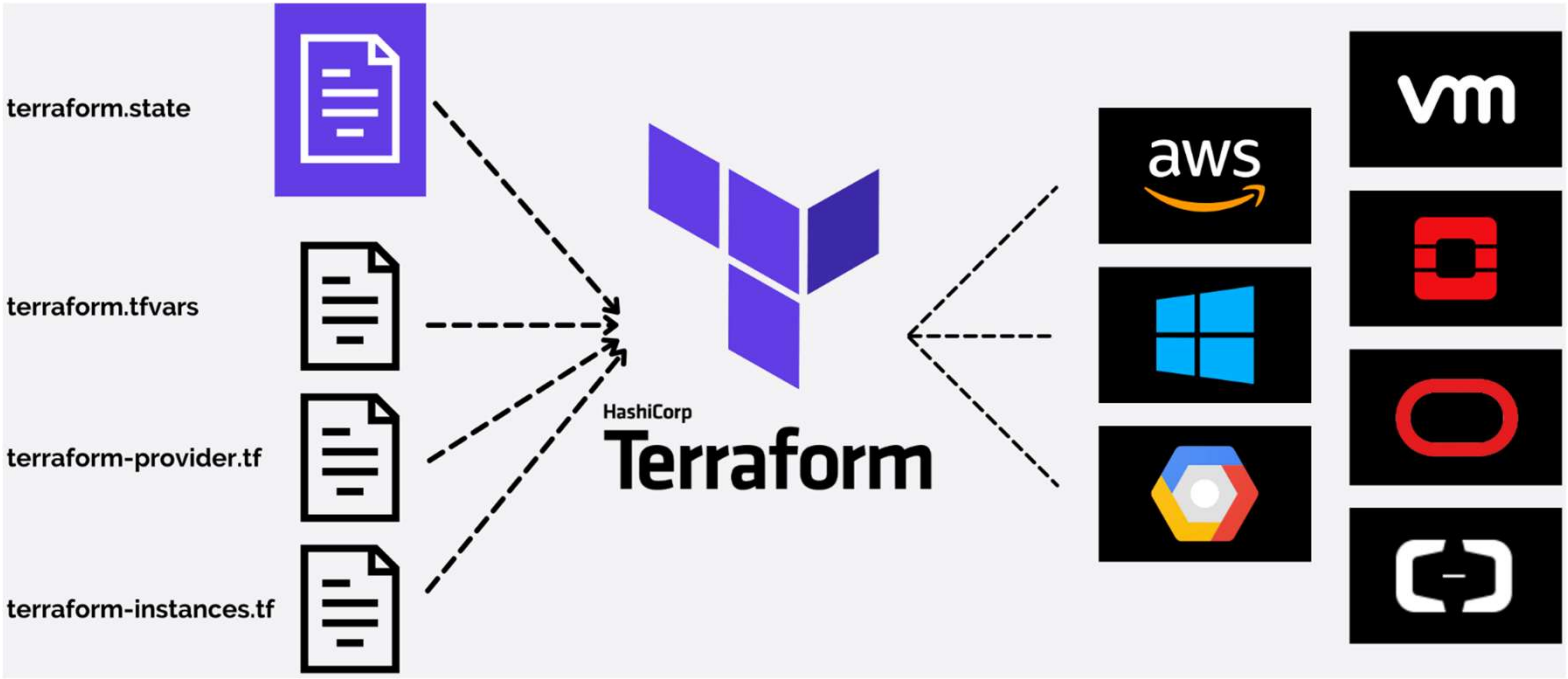
Introduction



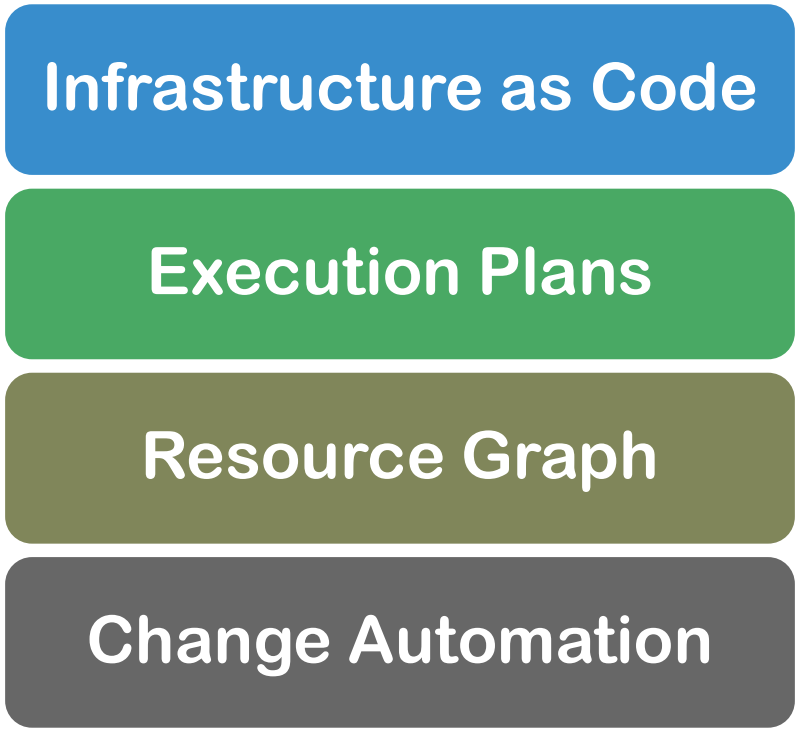
Deploy to multiple cloud using Terraform



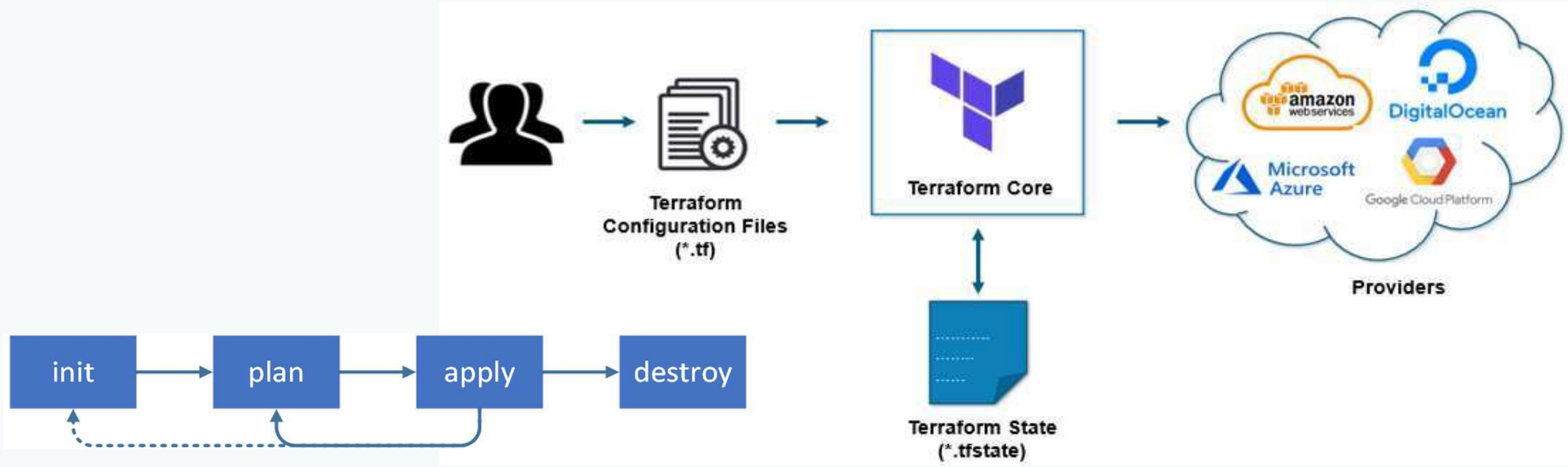
What is Terraform?



The key features of Terraform



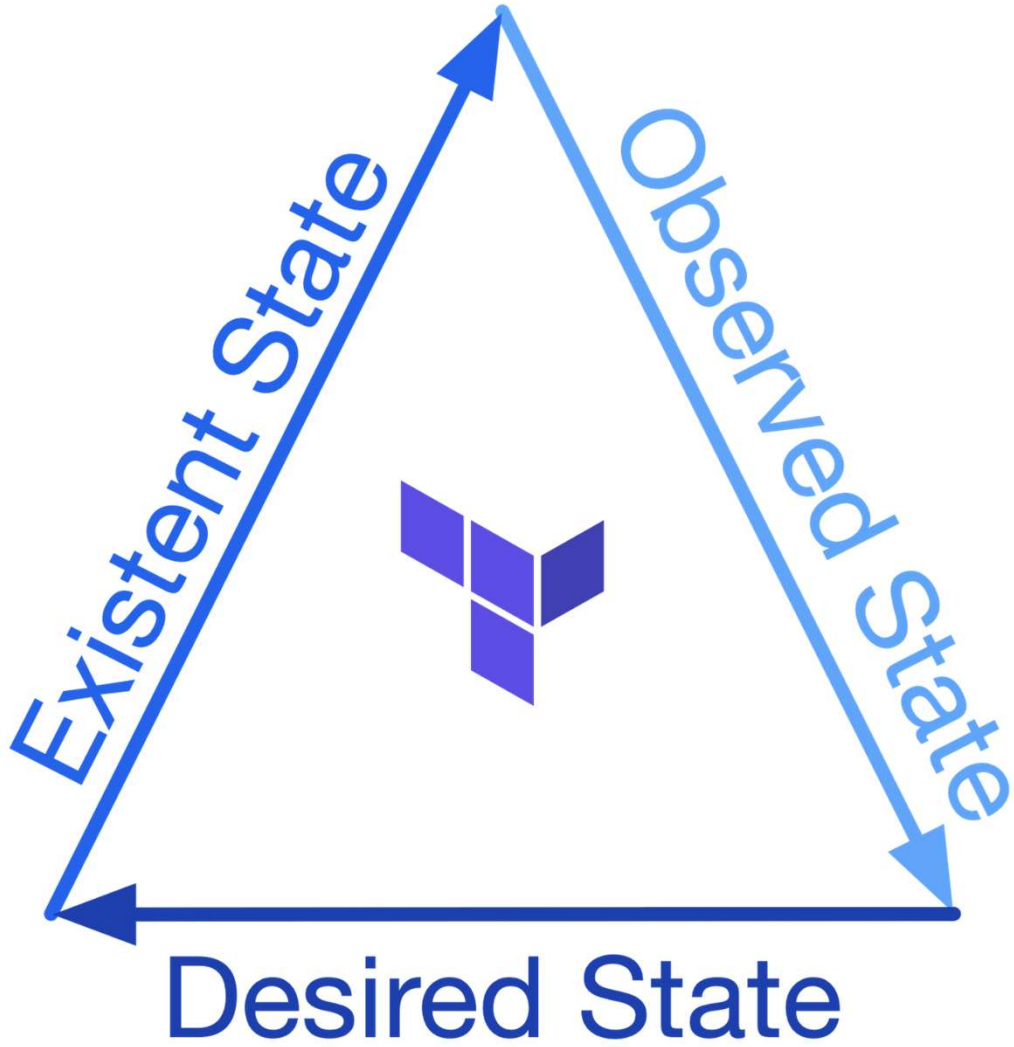
How Terraform works?



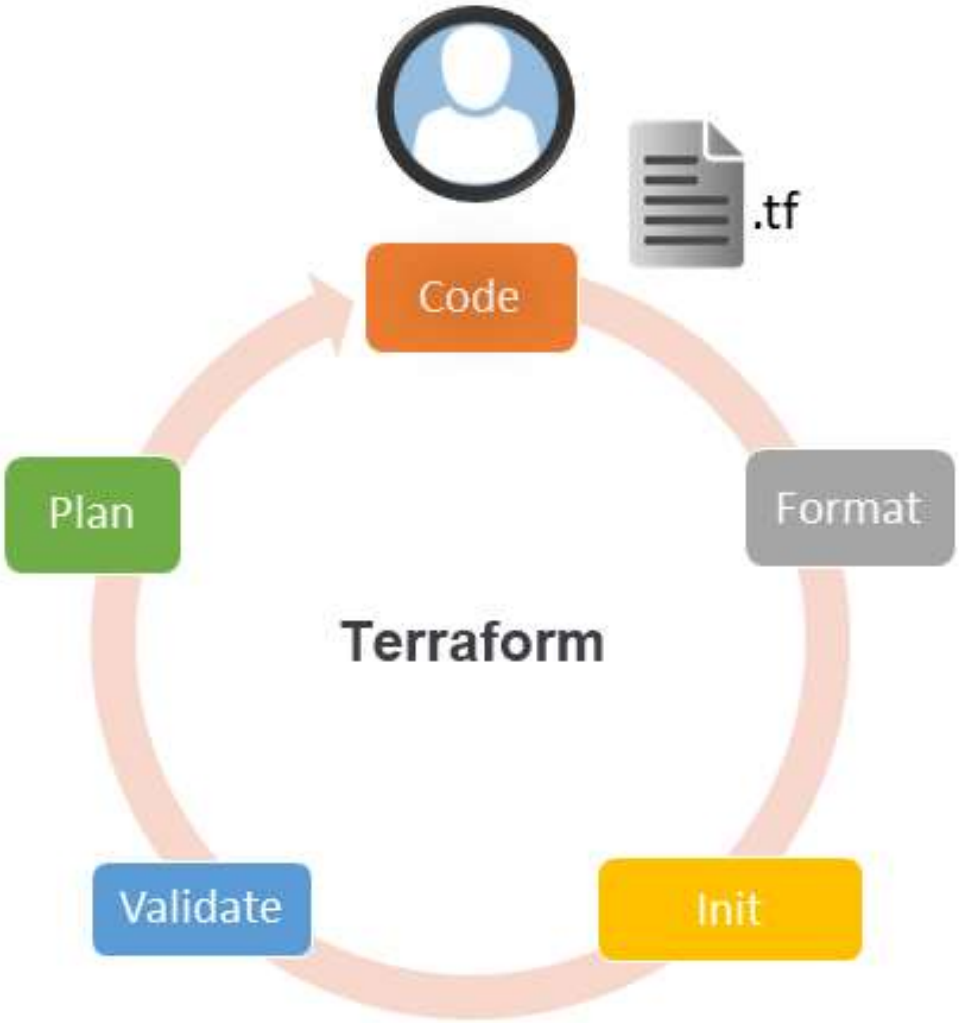
Install Azure CLI and Terraform

- Azure CLI:
  - curl -L https://aka.ms/InstallAzureCli | bash
- For Terraform, refer:
  - https://learn.hashicorp.com/tutorials/terraform/install-cli

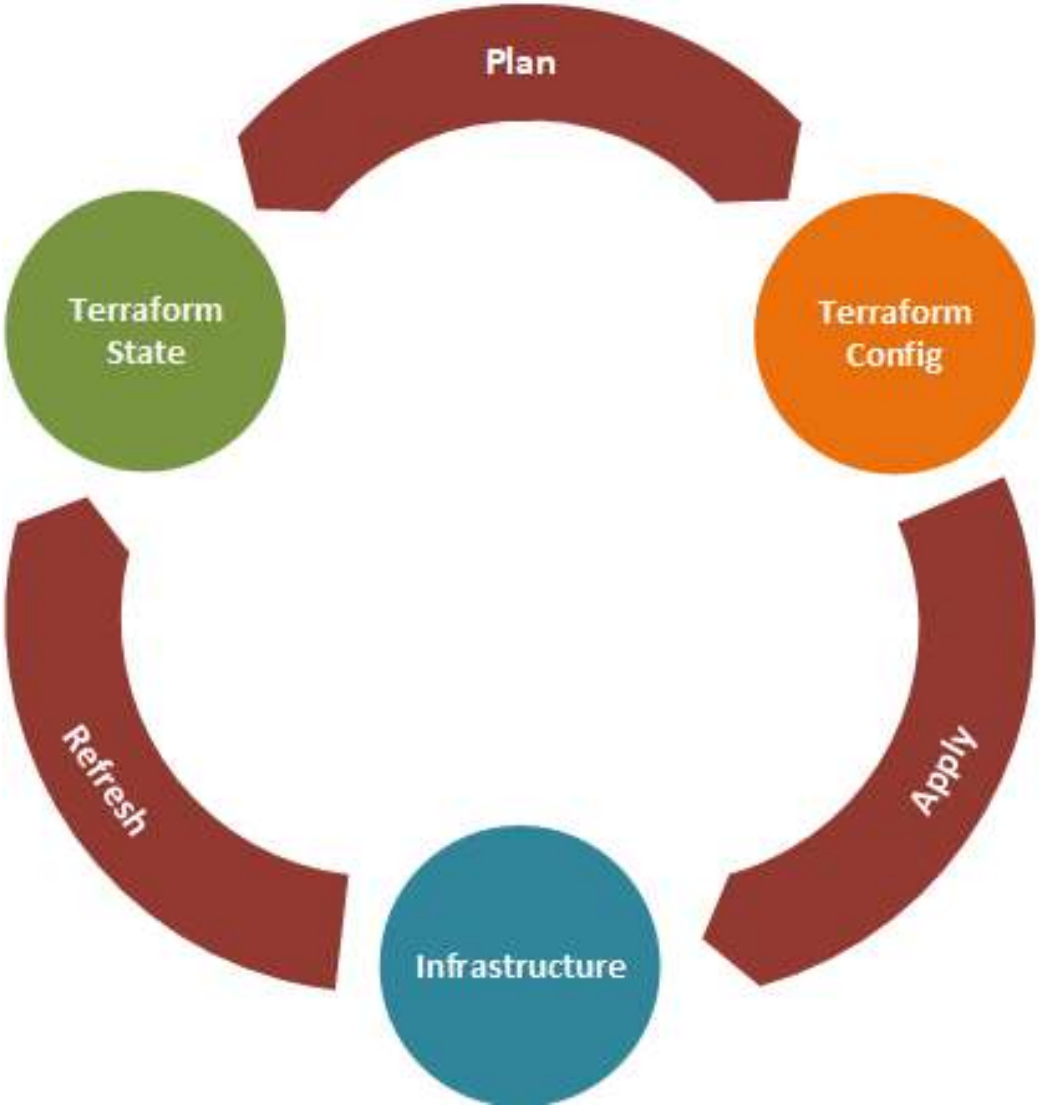
Terraform State



Steps



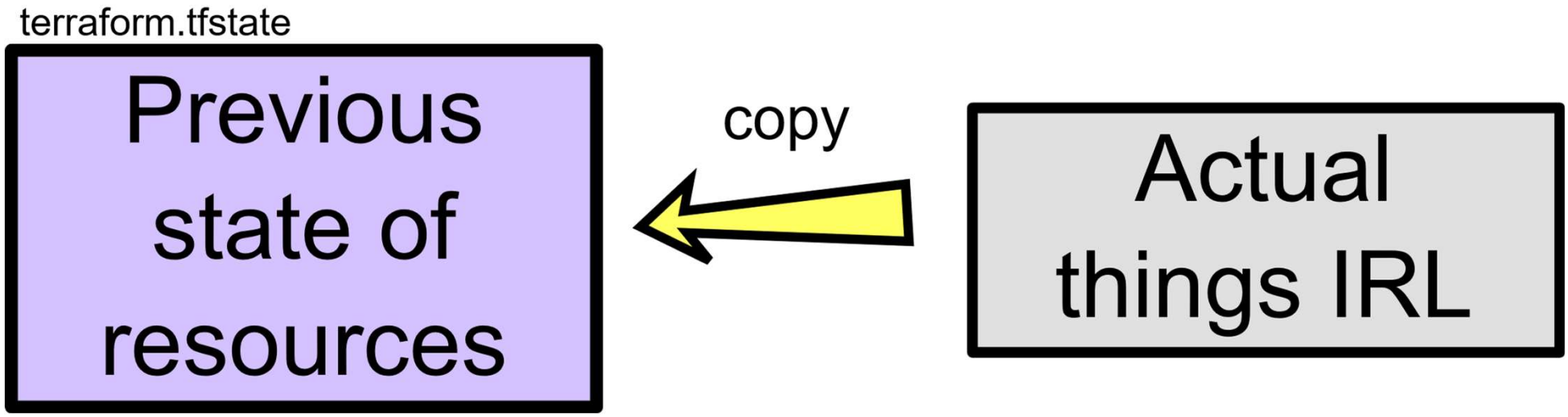
The Terraform Resource Lifecycle



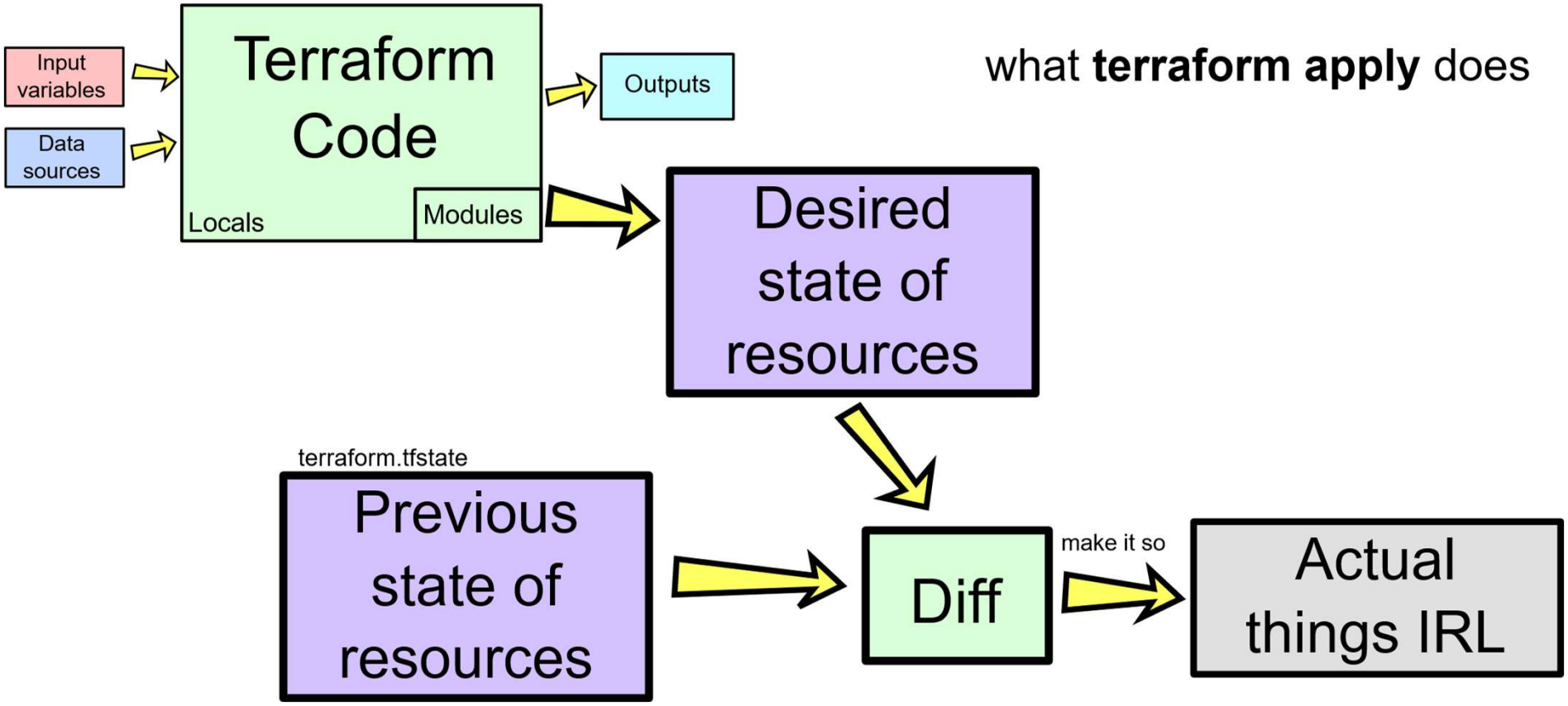
Getting Started & Setting Up Labs

- Install Terraform
- Choosing Right IDE for Terraform
  - - VSCode
- Use Azure and AWS account

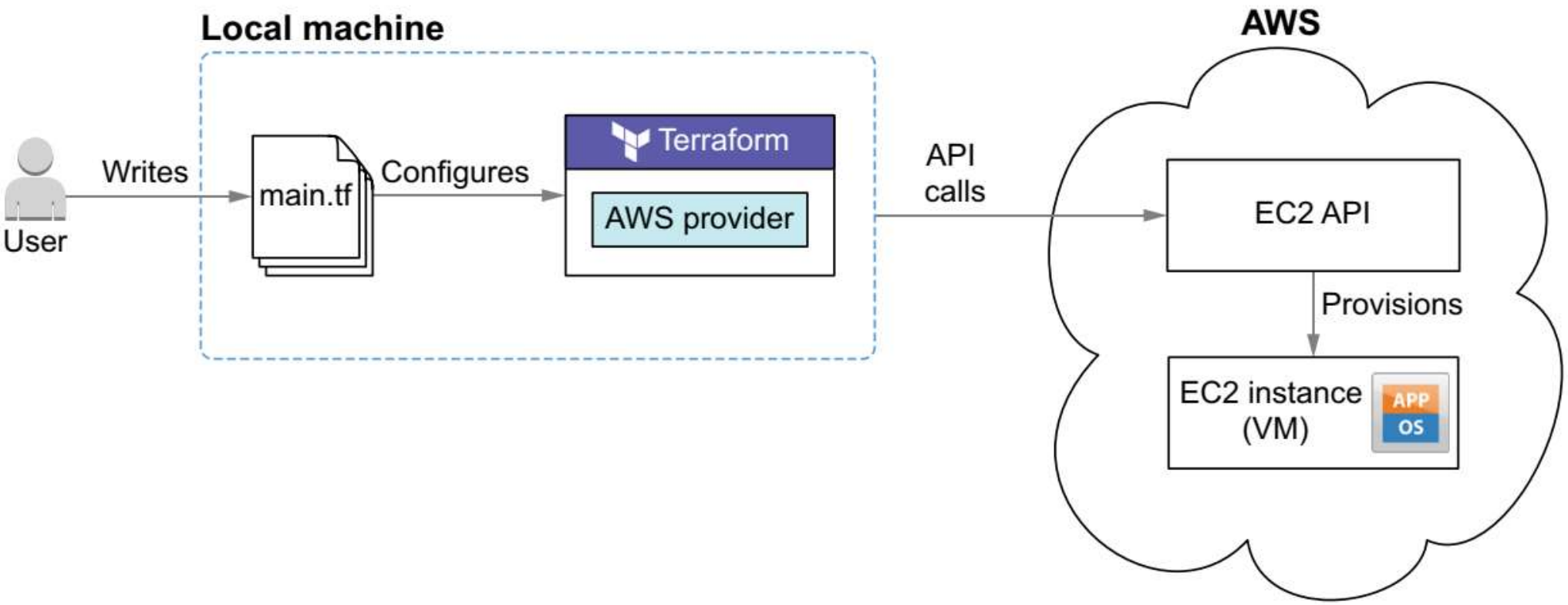
What terraform refresh does?



What Terraform apply does?

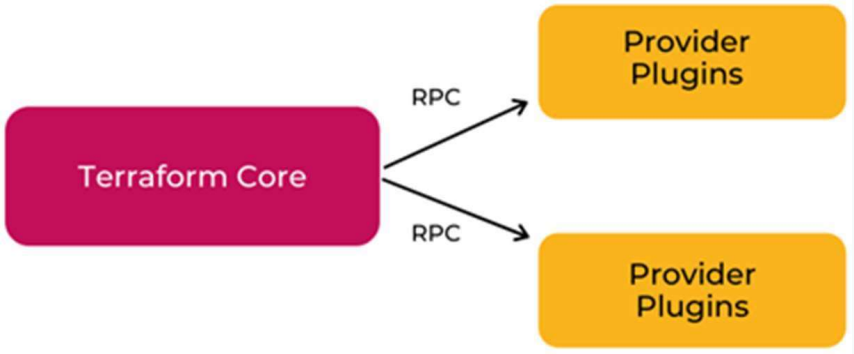


Using Terraform to Deploy EC2 instance to AWS



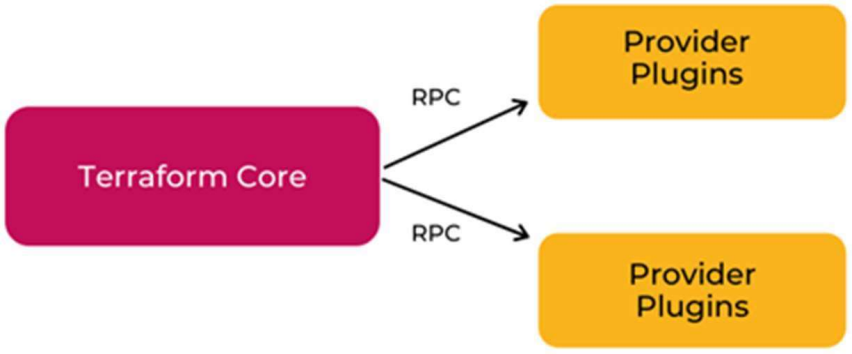
Terraform Core

- Written in the Go programming language
- The entry-point for anyone using Terraform
- Primary responsibilities
  - Reading configuration files
  - Resource state management
  - Construction of the Resource Graph
  - Plan execution
  - Communication with plugins



Terraform Plugins

- Each plugin exposes an implementation for a specific service
  - such as AWS, Azure
- Plugin Locations
  - ~/.terraform.d/plugins



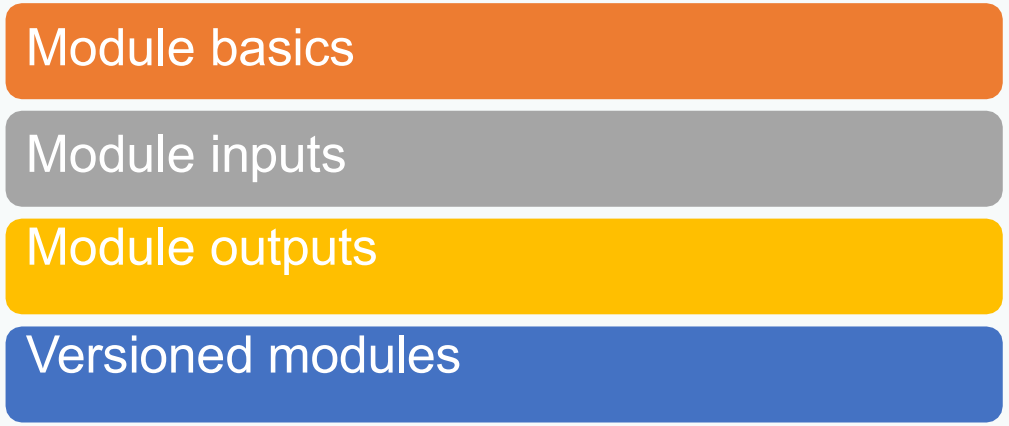
An Introduction to Terraform Syntax

- Called HashiCorp Configuration Language (HCL)
- Human readable as well as machine-friendly

```
# An AMI
variable "ami" {
description = "the AMI to use“
}
resource "aws_instance" "web" {
ami  = "${var.ami}"
count = 2
connection {
user = "root"
}
}
```

How to create reusable infrastructure

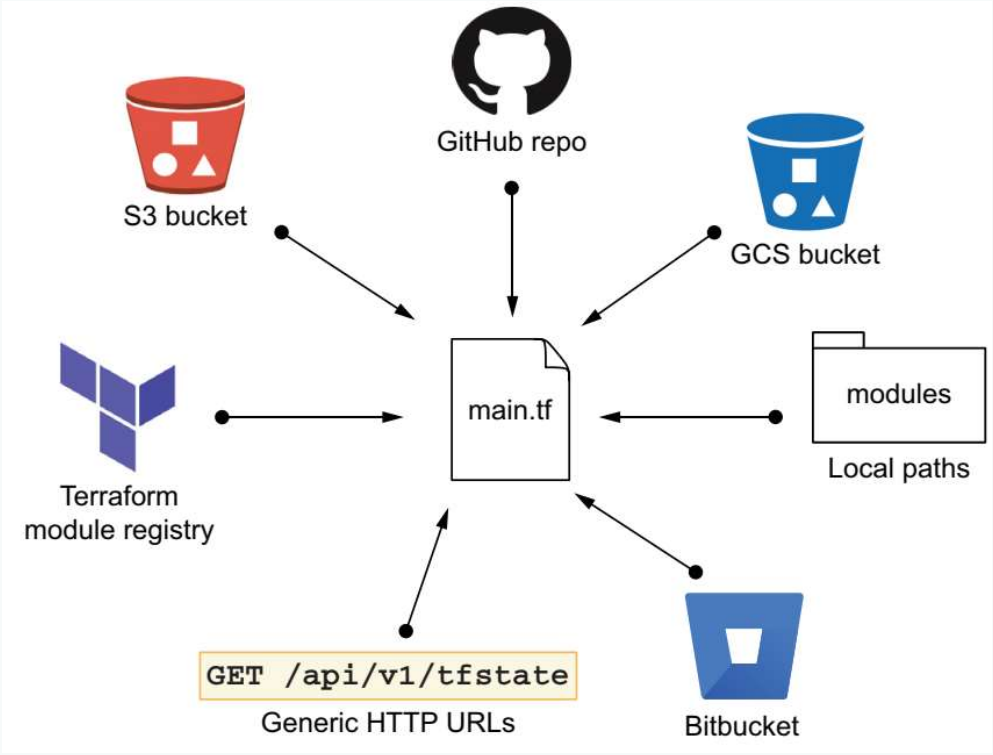
How to create reusable infrastructure



Module basics

- Any set of Terraform configuration files in a folder is a module.
  - $ tree minimal-module/
  - .
  - ├── README.md
  - ├── main.tf
  - ├── variables.tf
  - ├── outputs.tf

- A module can call other modules



Calling a Child Module

```
module "servers" {
source = "./app-cluster"
servers = 5
}
```

Module inputs

- Parameters for a Terraform module
- Like function arguments

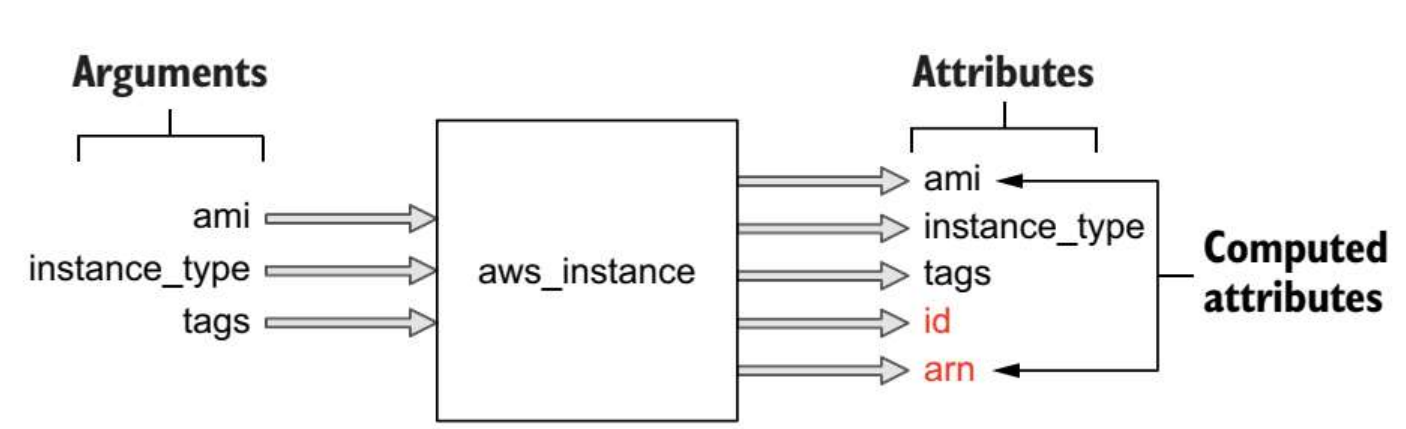
```
variable "image_id" {
type = string
}

variable "availability_zone_names" {
type          = list(string)
default = ["us-west-1a"]
}
```

Using Input Variable Values

- var.<NAME>

```
resource "aws_instance" "example" {
instance_type = "t2.micro"
ami = var.image_id
}
```



Variable Definitions (.tfvars) Files

- To set lots of variables, it is more convenient to specify their values in a variable definitions file
- And then specify that file on the command line with -var-file

```
terraform apply -var-file="testing.tfvars"
```

Module outputs

- Like the return values of a Terraform module

```
output "instance_ip_addr" {
value = aws_instance.server.private_ip
}
```

Loops

```
resource "aws_lam_user" "example" {
count = 3
name = "neo.${count.Index}"
}
```

Terraform vs Ansible

Terraform	Ansible
Terraform lays focus on infrastructure provisioning	Ansible fits really well in traditional automation
Terraform acts perfectly in configuring cloud infrastructures.	Ansible is employed for configuring servers.
Using Terraform, one can deploy load balancers, VPCs, etc	With Ansible, you can deploy apps on the top of infrastructure.
Terraform is declarative	Ansible is procedural
Upon giving an end instruction, Terraform can carry out all steps to present the final output	With Ansible, you need to dictate each step to achieve the end result.
Terraform is considered ideal for conserving the environment in a steady state	Ansible maintains all components in working condition and repairs an issue instead of replacing the entire infrastructure.

Thank You