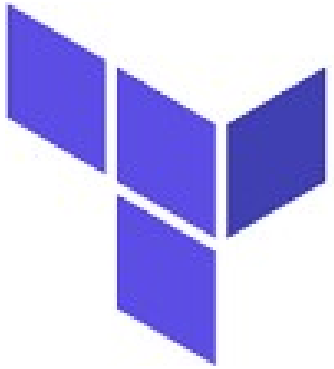
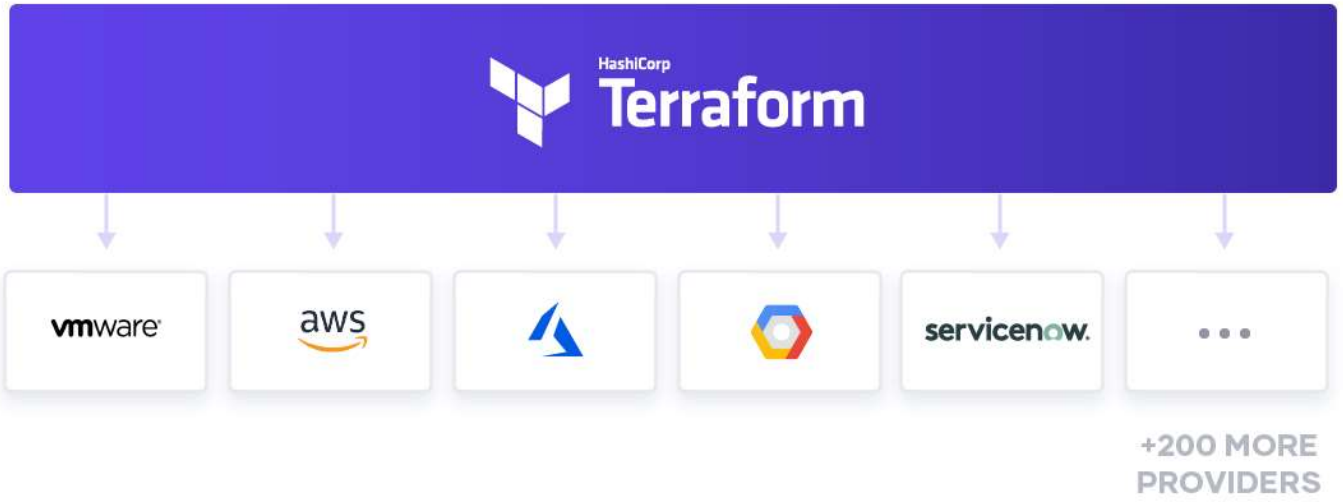


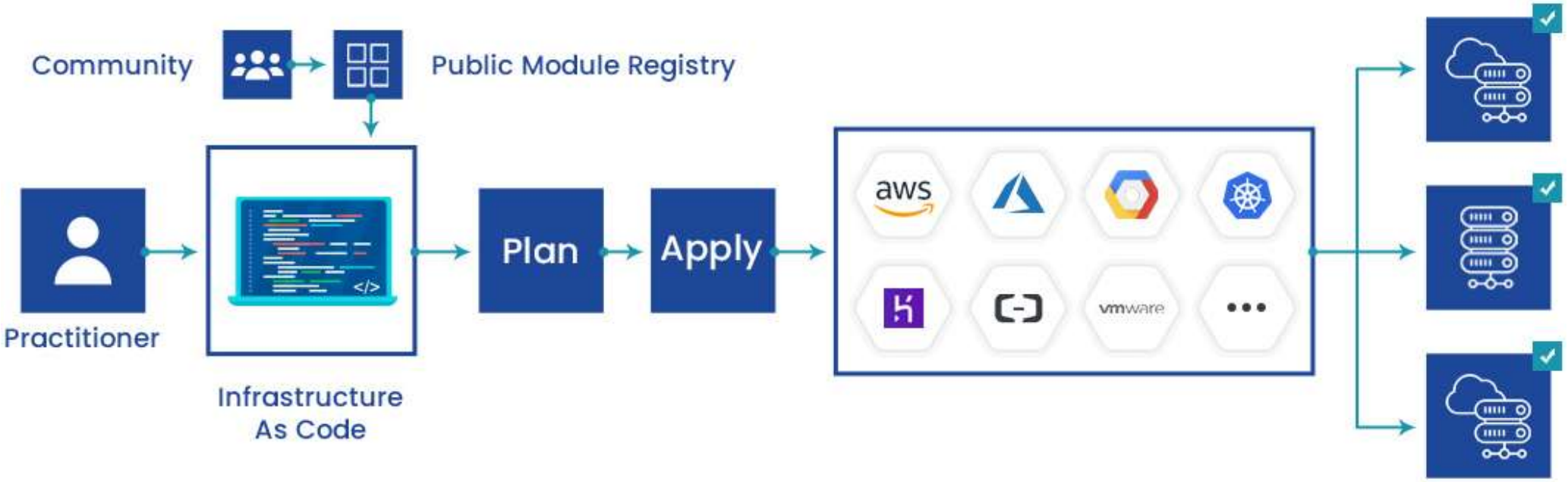
HashiCorp

Terraform

Introduction



Introduction



What is Terraform?

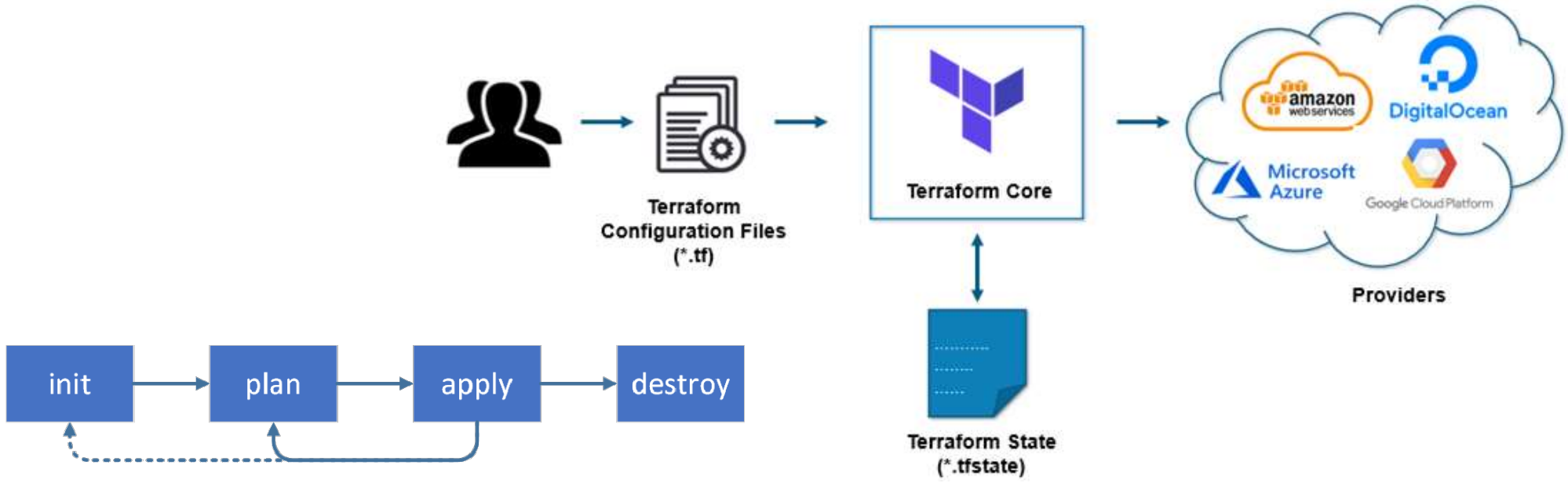
- Tool for
  - Building,
  - Changing, and
  - Versioning infrastructure safely and efficiently

- Can manage
  - Existing and popular service providers as well as
  - Custom in-house solutions.

The key features of Terraform

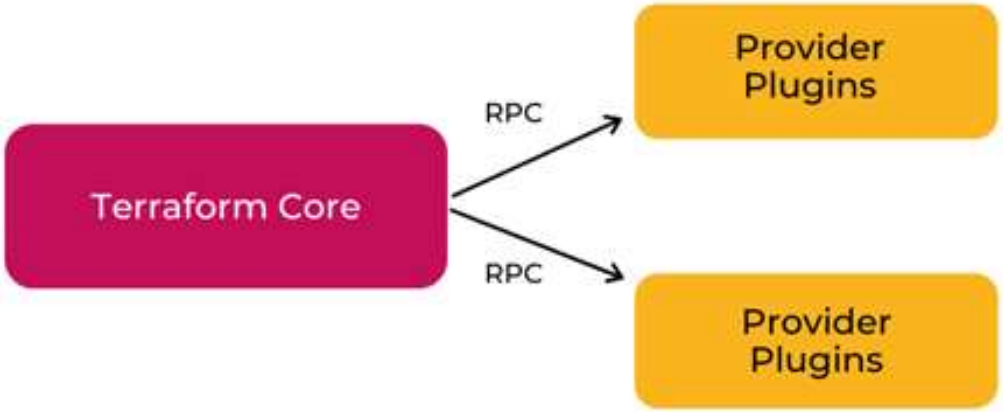
- Infrastructure as Code
- Execution Plans
- Resource Graph
- Change Automation

How Terraform works?



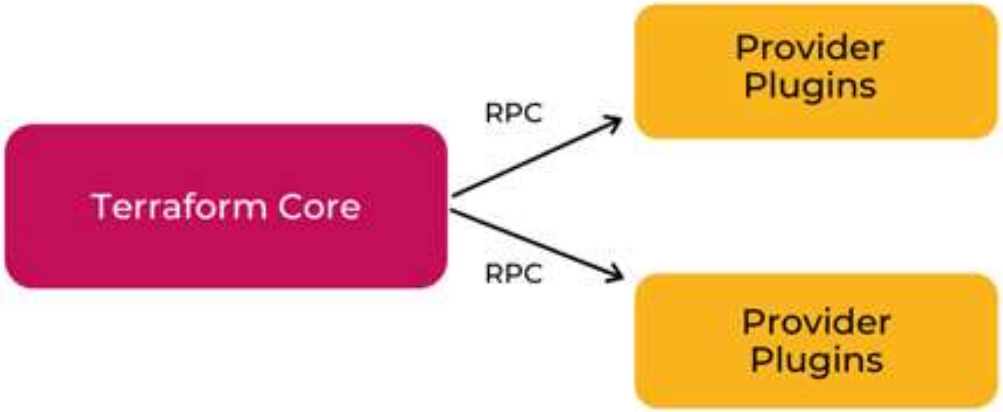
Terraform Core

- 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



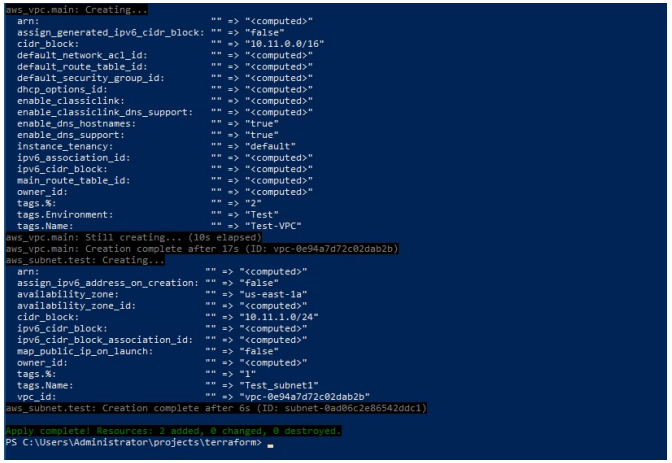
Terraform Plugins

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



Install Terraform

- curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add –
- sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main“
- sudo apt-get update && sudo apt-get install terraform
- terraform –help
- terraform -install-autocomplete
- Refer: 0-Install-Terraform.md



Procedural vs Declarative

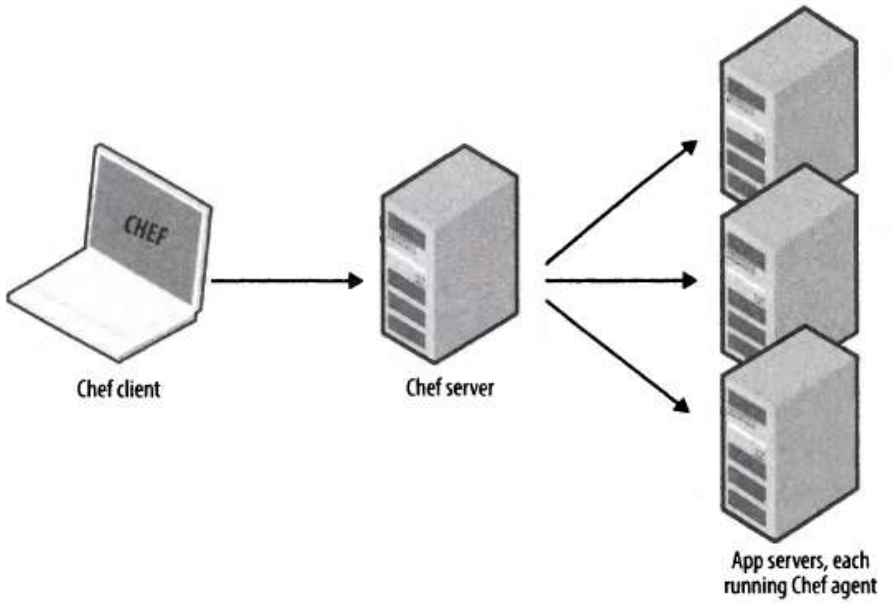
Ansible

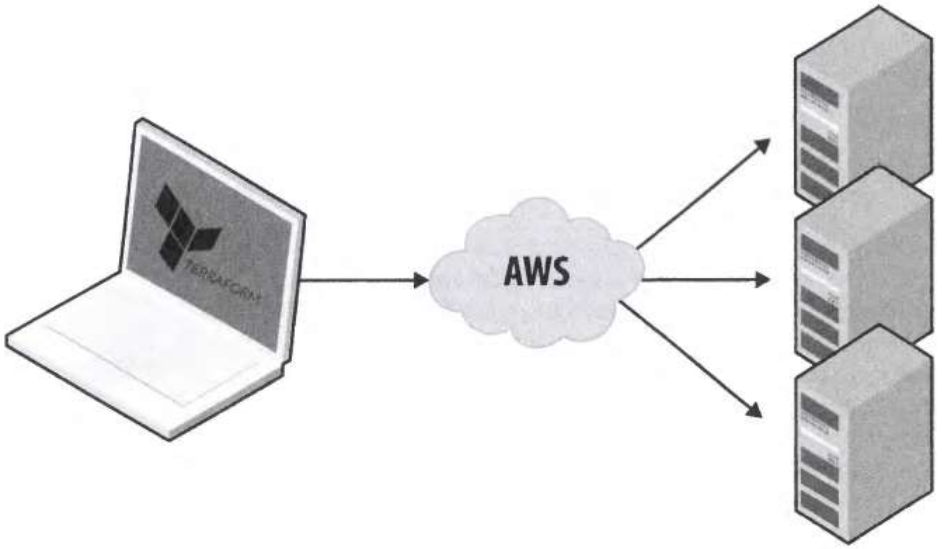
```
# Add 5 more

- ec2:
    count: 5
    image: aws-ami
    Instance_type: t2.micro
```

Terraform

```
# Make sure that we have 5

resource "aws_instance"
"ec2ins" {
 count = 5
 ami = "aws-ami"
 instance_type = "t2.micro"
}
```

Agent vs Agentless





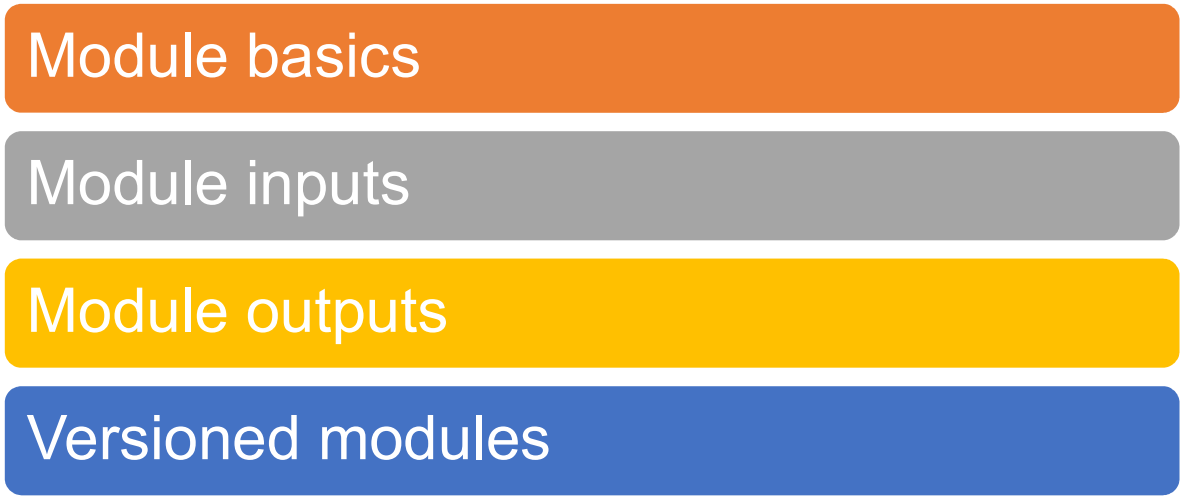
An Introduction to Terraform Syntax

- 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
  source_dest_check = false
  connection {
    user = "root"
  }
}
```


thanks

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}"
- }

Getting Started & Setting Up Labs

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


thanks