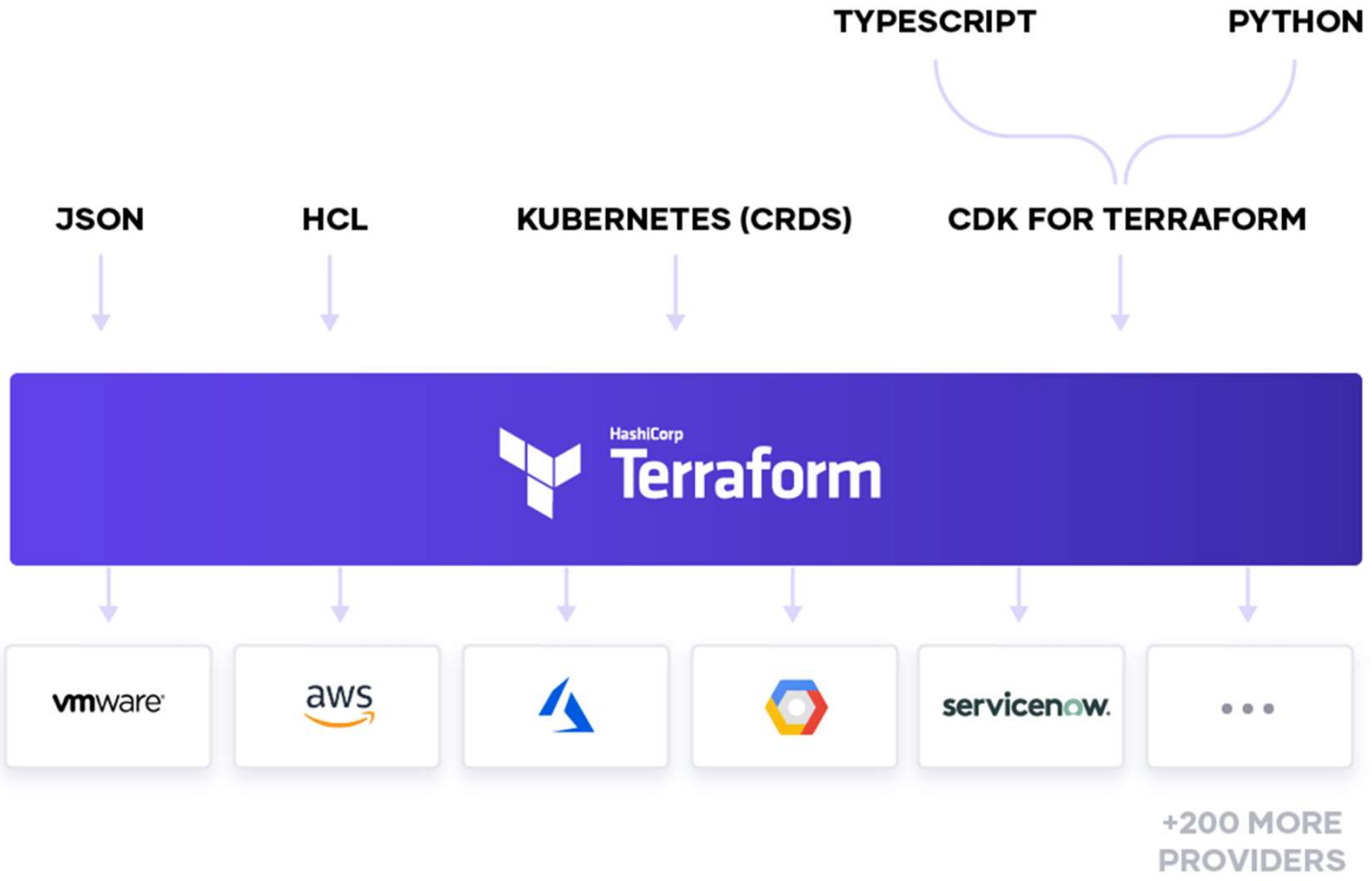
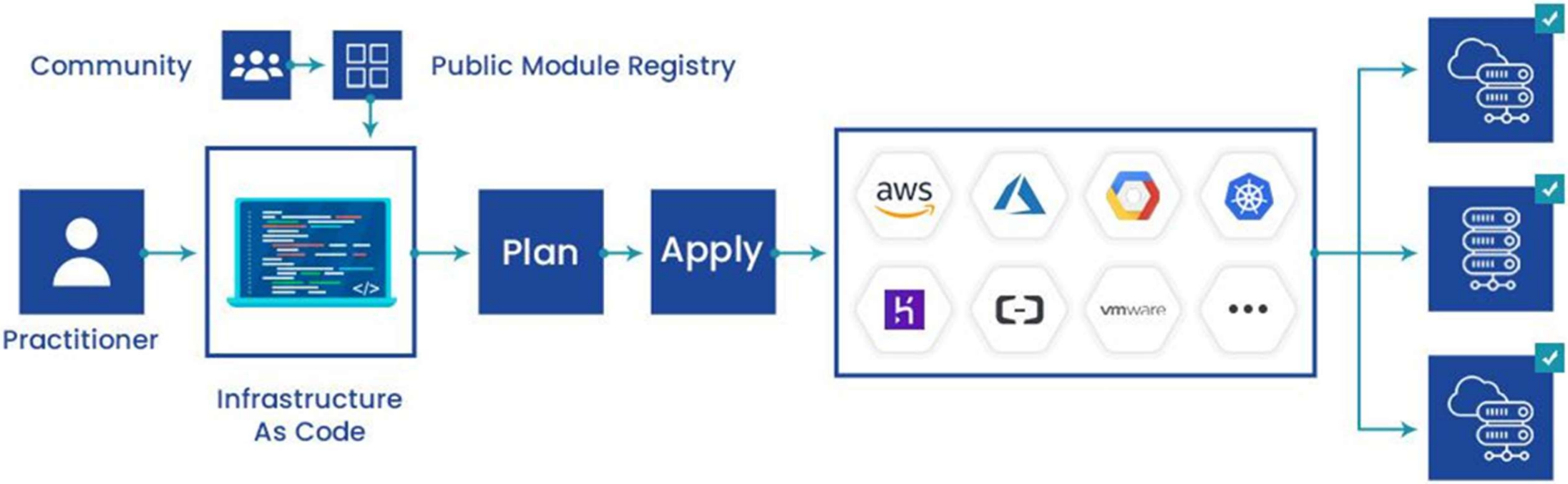
HashiCorp

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.

Why Terraform?

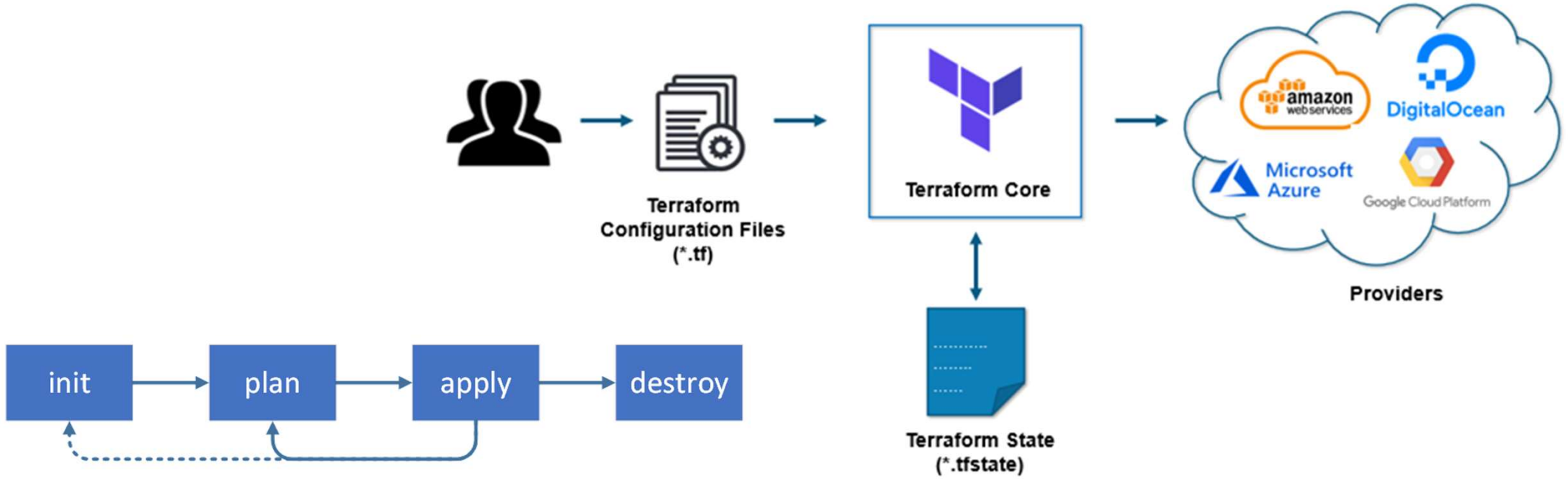
- IAAC
- Platform agnostic
- Enables you to implement all kinds of coding principles
  - Having code in source control
  - Ability to write automated tests
- Community and is open source
- Speed

What is infrastructure as code?

- The process of managing and provisioning computer data centers through machine-readable definition files
- The definitions may be in a version control system
- It can use either scripts or declarative definitions
- More often used to promote declarative approaches.

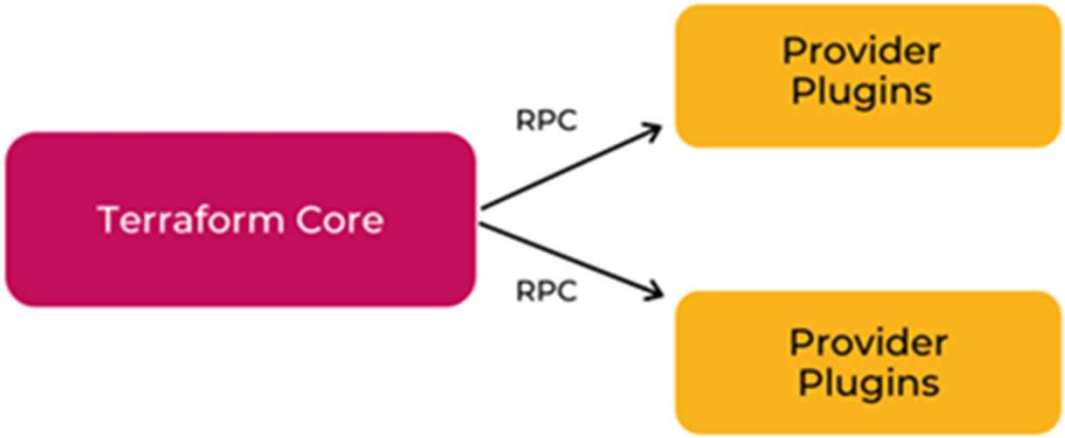
How Terraform works?

- Split into two main parts
  - Terraform Core and
    - Uses remote procedure calls (RPC) to communicate with Terraform Plugins
  - Terraform Plugins
    - Expose an implementation for a specific service, such as AWS



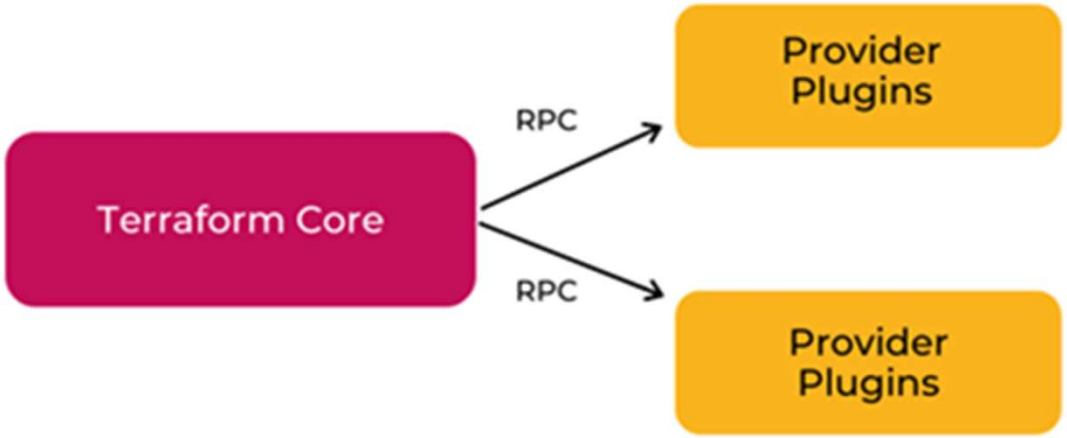
Terraform Core

- Written in the Go programming language
- Compiled binary
  - Is the command line tool (CLI) terraform
- The entrypoint for anyone using Terraform
- Primary responsibilities
  - Reading configuration files
  - Resource state management
  - Construction of the Resource Graph
  - Plan execution
  - Communication with plugins



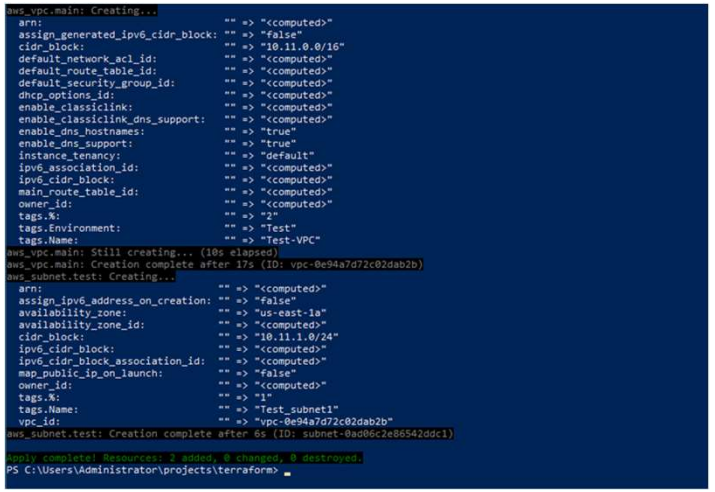
Terraform Plugins

- Written in Go
- Each plugin exposes an implementation for a specific service
  - such as AWS, Azure
- Primary responsibilities
  - Authentication with the Infrastructure Provider
  - Define Resources that map to specific Services
- 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



Compare to other IAAC tools

	Chef	Puppet	Ansible	SaltStack	Terraform
Cloud		All	All	All	All
Type	Config Mgmt	Config Mgmt	Config Mgmt	Config Mgmt	Orchestration
Infrastructure	Mutable	Mutable	Mutable	Mutable	Immutable
Language	Procedural	Declarative	Procedural	Declarative	Declarative
Architecture	Client/Server	Client/Server	Client only	Client only	Client only
Orchestration					
Lifecycle (state) management	No	No	No	No	Yes
VM provisioning	Partial	Partial	Partial	Partial	Yes
Networking	Partial	Partial	Partial	Partial	Yes
Storage Management	Partial	Partial	Partial	Partial	Yes

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"
  }
}
```

How to create reusable infrastructure

- Module basics
- Module inputs
- Module outputs
- Versioned modules

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

- At least one module, known as its root module
  - Consists of the resources defined in the .tf files in the main working directory.
- 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
}
```

Terraform loops, if-statements and deployment

- Loops
- If-statements
- If-else-statements
- Zero-downtime deployment

Loops

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

If-statements

- Terraform doesnt support if-statements, so this code won't work
- However, you can accomplish the same thing by using the count parameter
  - resource "aws_autoscallng_schedule" "scale_out_durlng_buslness_hours" {
  - count = "${var.enable_autoscaling}"
  - scheduled_action_name = "scale-out-durlng-buslness-hours"
  - niin_size = 2
  - nax_size = 10
  - desired_capacity = 10
  - recurrence ="09** *"
  - autoscaling_group_name = "${aws_autoscaling_group.example.name}"
  - }

If-else-statements

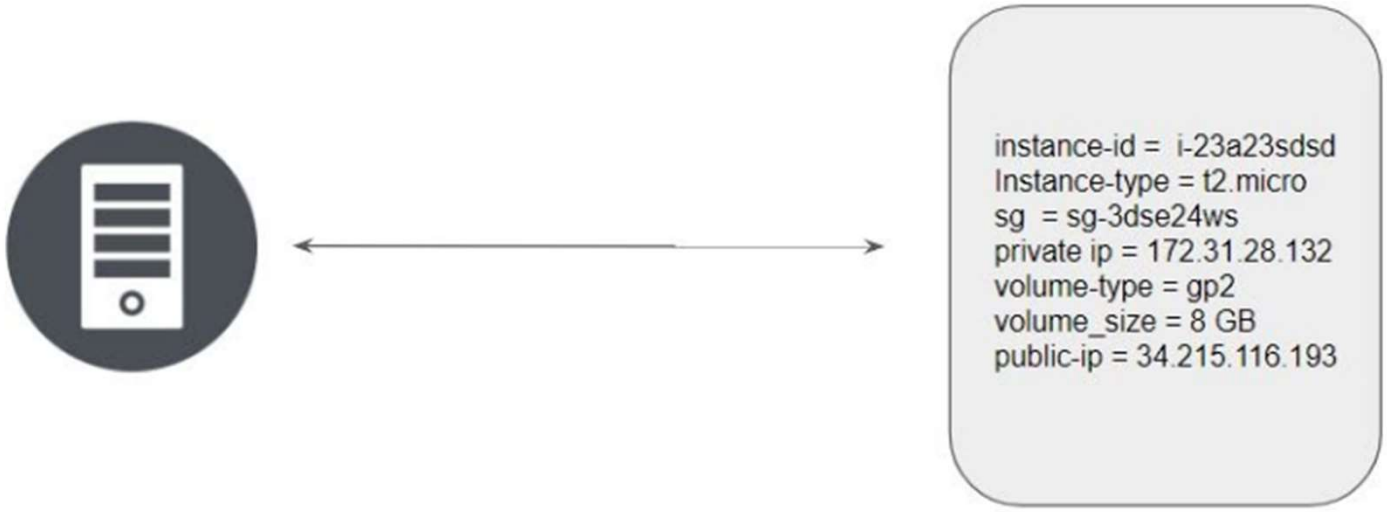
- resource "aws_lam_user_pollcy_attachment" "neo_cloudwatch_full_accessM {
- count = "${var.glve_neo_cloudwatch_full_access}"
- user = "${aws_lam_user.example.0.name}"
- pollcy_arn = "${aws_lam_pollcy.cloudwatch_full_access.arn}"
- }
- resource Maws_lam_user_pollcy_attachmentM Mneo_cloudwatch_read_only" {
- count = "${1 - var.glve_neo_cloudwatch_full_access}"
- user = "${aws_iam_user. example. 0. name}11
- pollcy_arn = M${aws_lam_pollcy.cloudwatch_read_only.arn}"
- }

Deploying Infrastructure with Terraform

Hands-On

TFState file

- Terraform stores the state of the infrastructure that is being created from the TF files.
- This state allows terraform to map real-world resource to your existing configuration.




thanks