Terraform Crash Course

For Absolute Beginners

Course Overview

What you will learn!

1. Introduction to **Infrastructure as Code (IaC)**
2. IaC with **Terraform**
3. Terraform **Core Concepts**
4. Build a complete **Practice project**

Section 1

Introduction to Infrastructure as Code (IaC)

HashiCorp Terraform is an **infrastructure as code tool** that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.

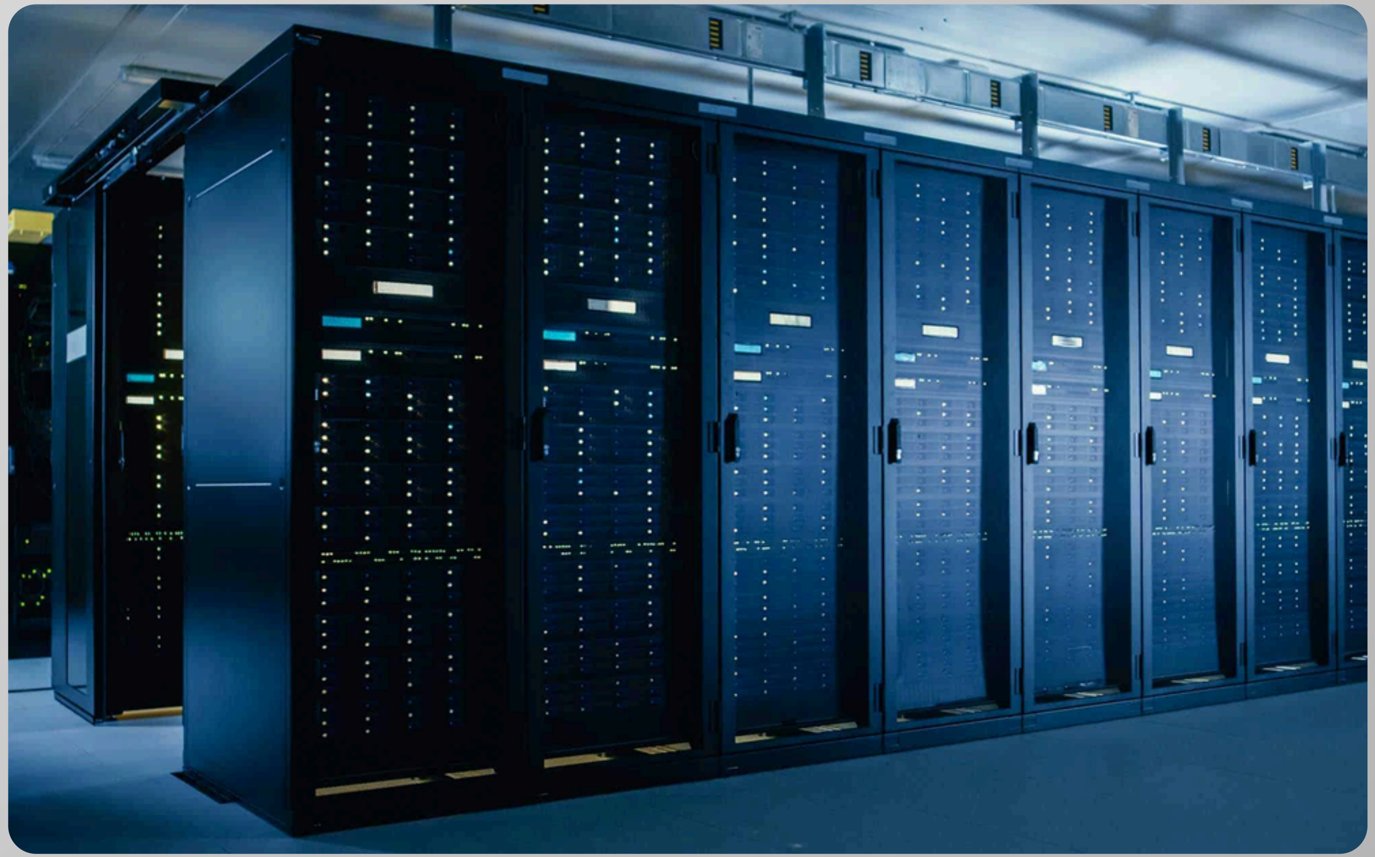
What do we mean by "Infrastructure"?

- Everything that **supports** the application/service to run
- **Includes:**
  - servers
  - network configurations
  - storage
  - monitoring
- Types - **Physical**, **On Cloud**, Hybrid, Edge Infrastructure

Managing your Infrastructure – "The Traditional way"

How did that happen?

- Physical or On-prem
- Any changes needed – **were made "manually"**
- Working fine – because **changes were rare!**
- **Problems:**
  - **Not able to scale** based on demand
  - **High** maintenance cost
  - **Low flexibility** – challenging to re-configure again



Managing your Infrastructure – Shift to Cloud Environments

What changes we observed?

- Infra. components are API-driven – **low manual effort**
- Able to scale up or down – **On demand**
- Takes **less time to maintain** – handled by the Provider
- **Benefits:**
  - **Easily customizable** based on needs
  - Cost efficient – **“Pay-as-you-go”**
  - Easily **deploy + manage** resources across multiple regions

Managing your Infrastructure – Shift to Cloud Environments

But, there are Problems!

- Lot of **“manual” effort** – How?
- Inconsistencies due to **Configuration Drift**
- **Scaling becomes complex** – Again!
- Solution that makes sense? **Defining the Infrastructure “as Code”**

Infrastructure as Code (IaC)

Infrastructure as Code (IaC) allows you to **manage infrastructure with configuration files** rather than through a graphical user interface.

- **Core Idea:** Write code to define the **"desired state"** of your infrastructure
- **Benefits:**
  - **Consistent** across different environments
  - We are **"automating" the process** entirely - Saves time & efforts
  - **Version Control**

Infrastructure as Code (IaC)

Tools to provision your infrastructure as code!

Cloud Specific

Infra. on a single cloud provider

aws

- CloudFormation
- Elastic Beanstalk

- Cloud Deployment Manager

- Resource Manager Templates
- Blueprints

Cloud Agnostic

Infra. on a multiple cloud providers

Pulumi

OpenTofu

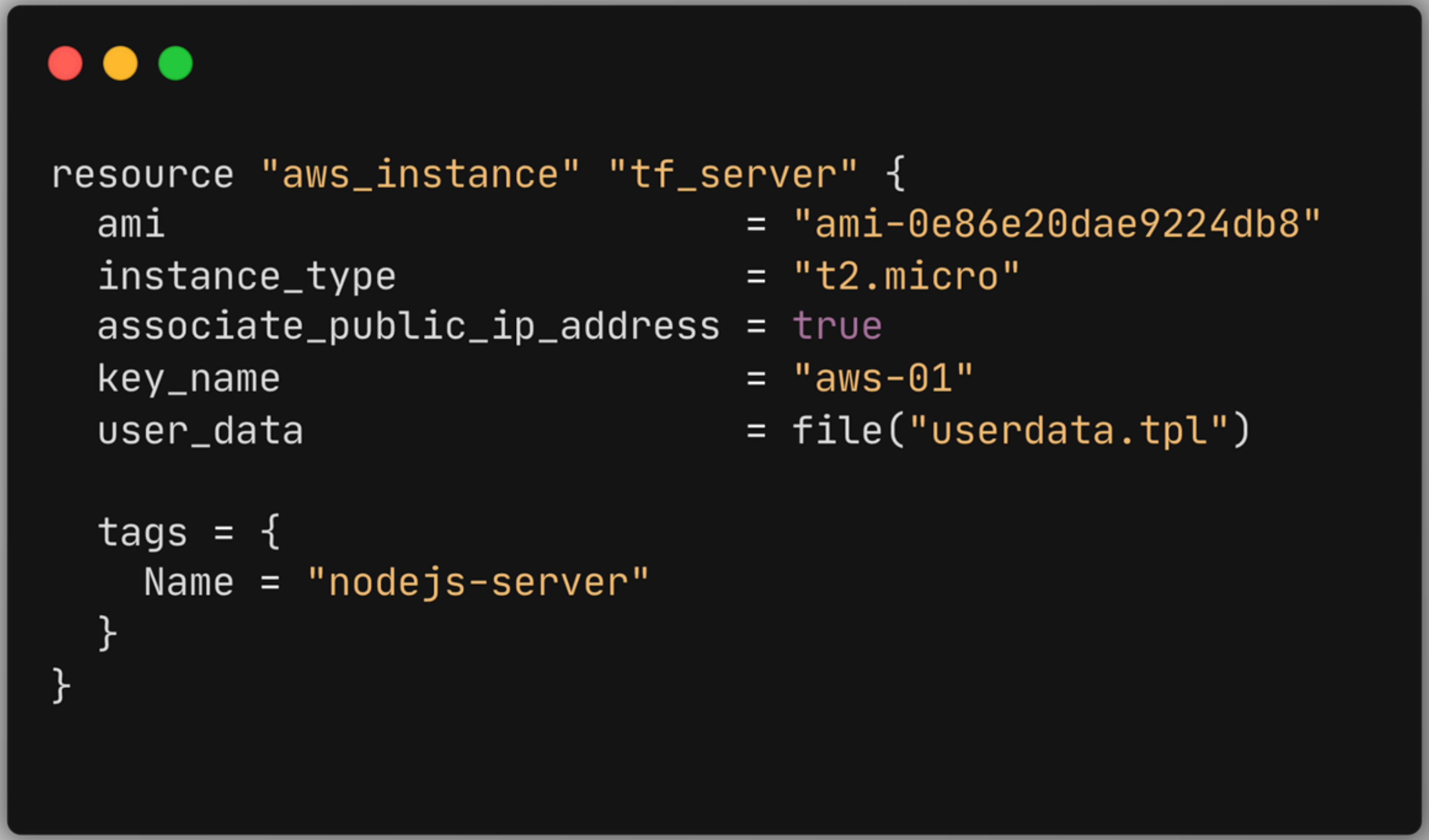
Section 2

Infrastructure as Code with Terraform

Infrastructure as Code Using Terraform

HashiCorp Terraform is an **infrastructure as code tool** that lets you define both **cloud and on-prem resources** in **human-readable configuration files** that you can **version**, **reuse**, and **share**.

- Uses **HCL (HashiCorp Configuration Language)**
- Follows a **Declarative approach**
- **Automates** infra. lifecycle management
- **Version Control** and Reusable

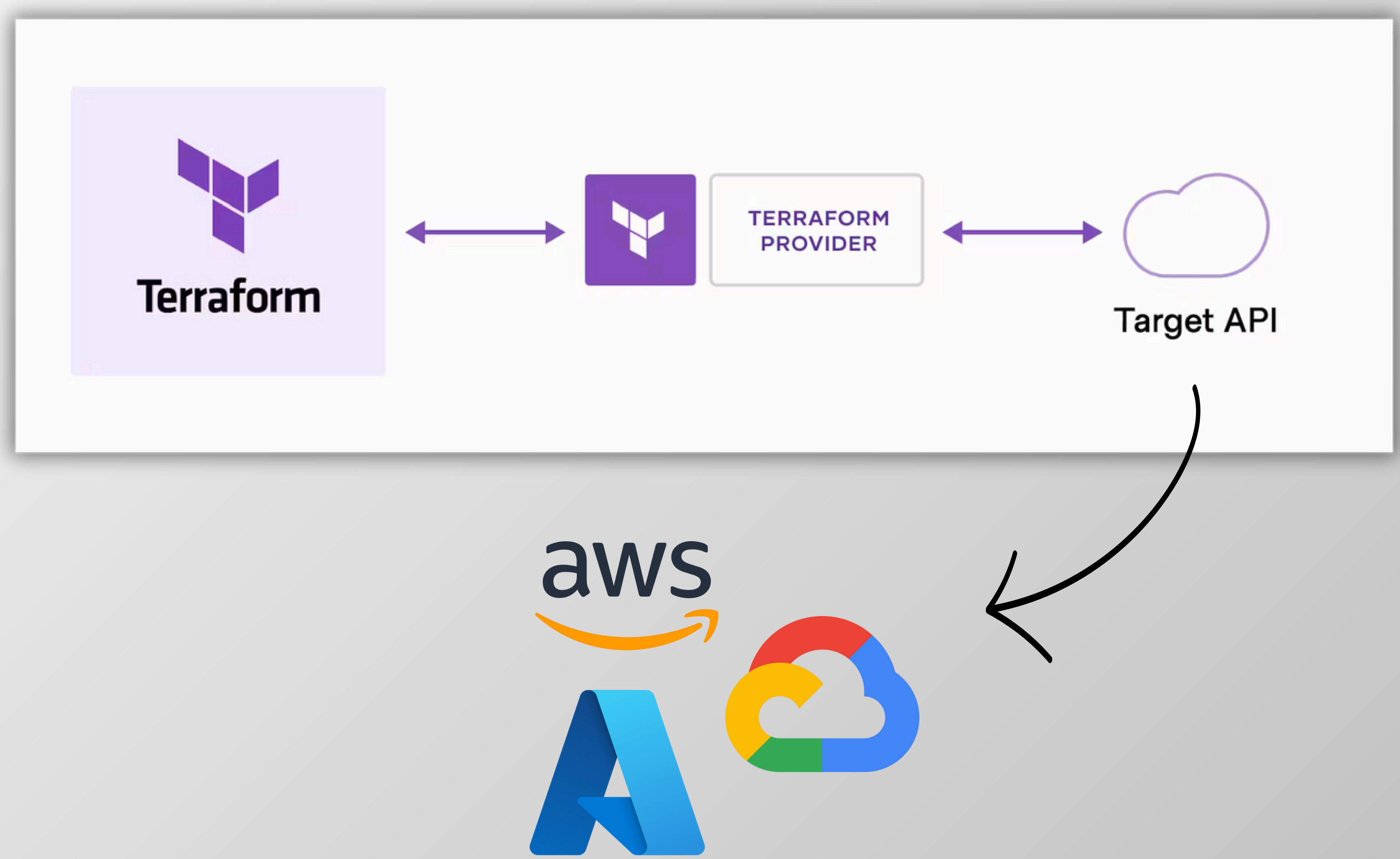


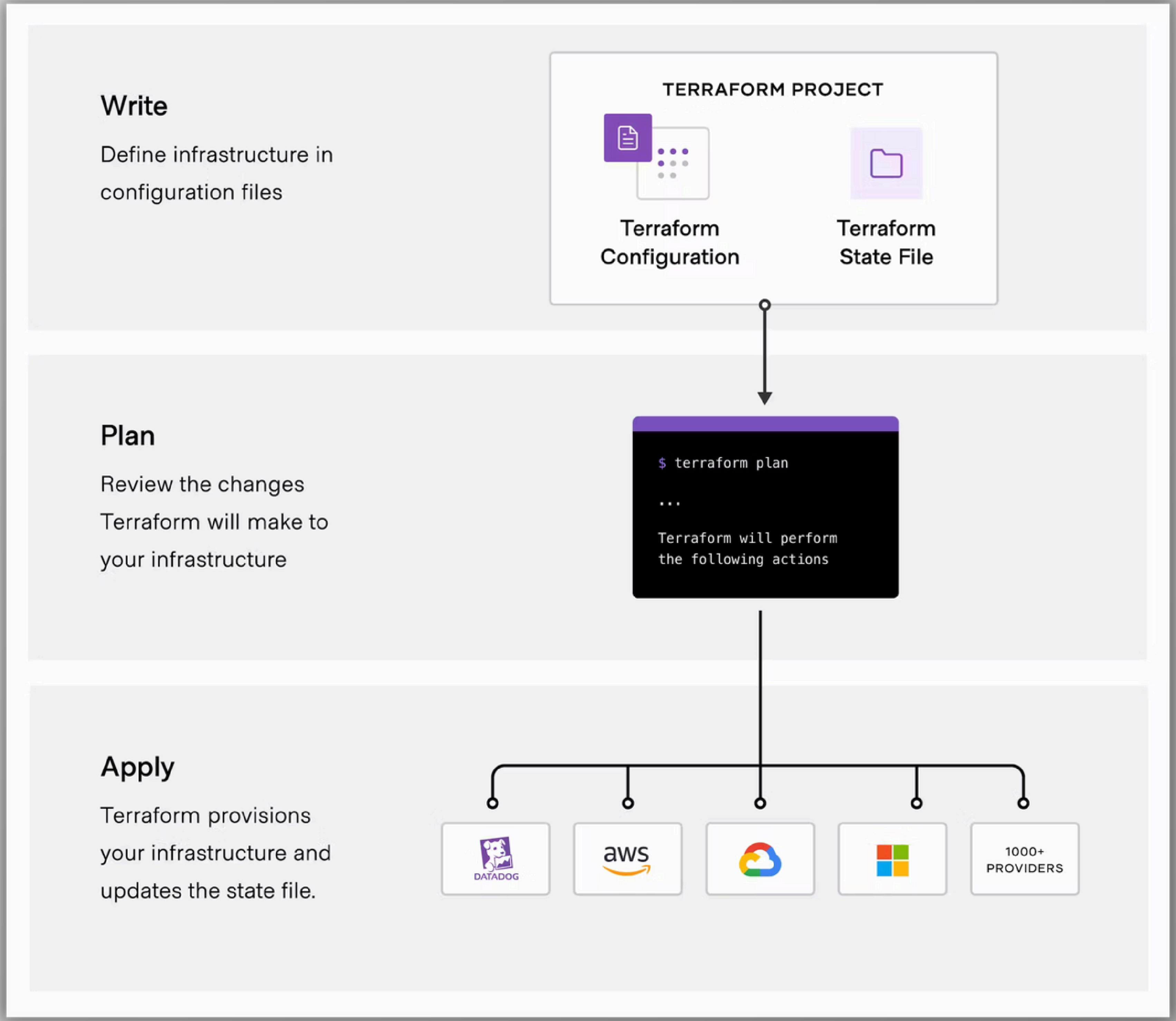
```
resource "aws_instance" "tf_server" {
  ami                         = "ami-0e86e20dae9224db8"
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = "aws-01"
  user_data                   = file("userdata.tpl")

  tags = {
    Name = "nodejs-server"
  }
}
```

How does Terraform work?

Whats happening in the backend!





Write
Define infrastructure in configuration files
TERRAFORM PROJECT
Terraform Configuration
Terraform State File
Plan
Review the changes Terraform will make to your infrastructure
Apply
Terraform provisions your infrastructure and updates the state file.
DATADOG
aws
1000+ PROVIDERS

```
$ terraform plan
...
Terraform will perform
the following actions
```

Source: https://developer.hashicorp.com/terraform/intro

Section 3

Terraform Core Concepts

Core Concepts

- Providers
- Resources
- HCL Language features
- State management
- Variables and Outputs
- Modules

Installation

macOS

```
brew tap hashicorp/tap

brew install hashicorp/tap/terraform
```

Windows

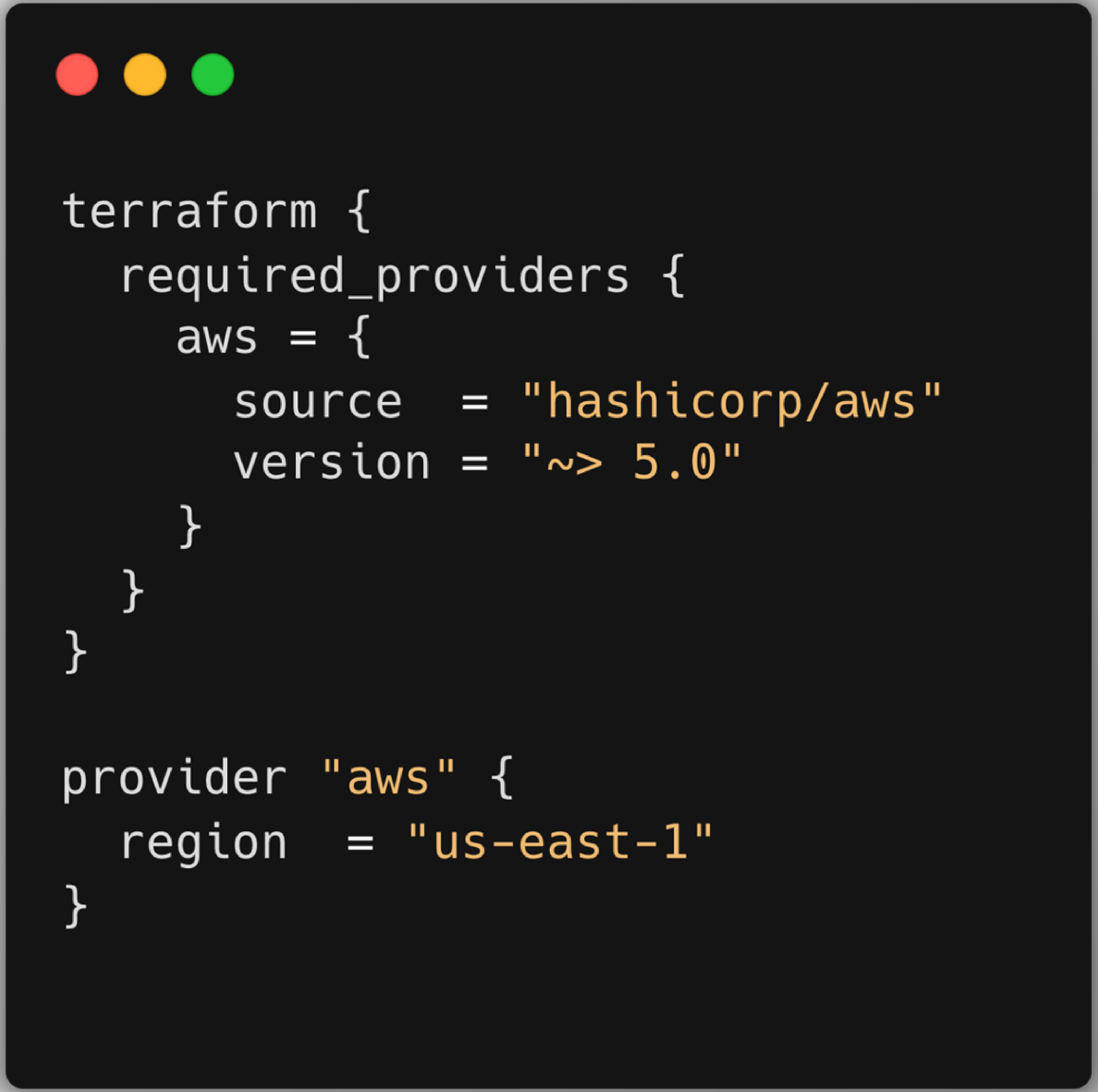
```
choco install terraform
```

Providers

- A **plugin** used to interact with APIs.

- Where do they come from?

  - Terraform Registry



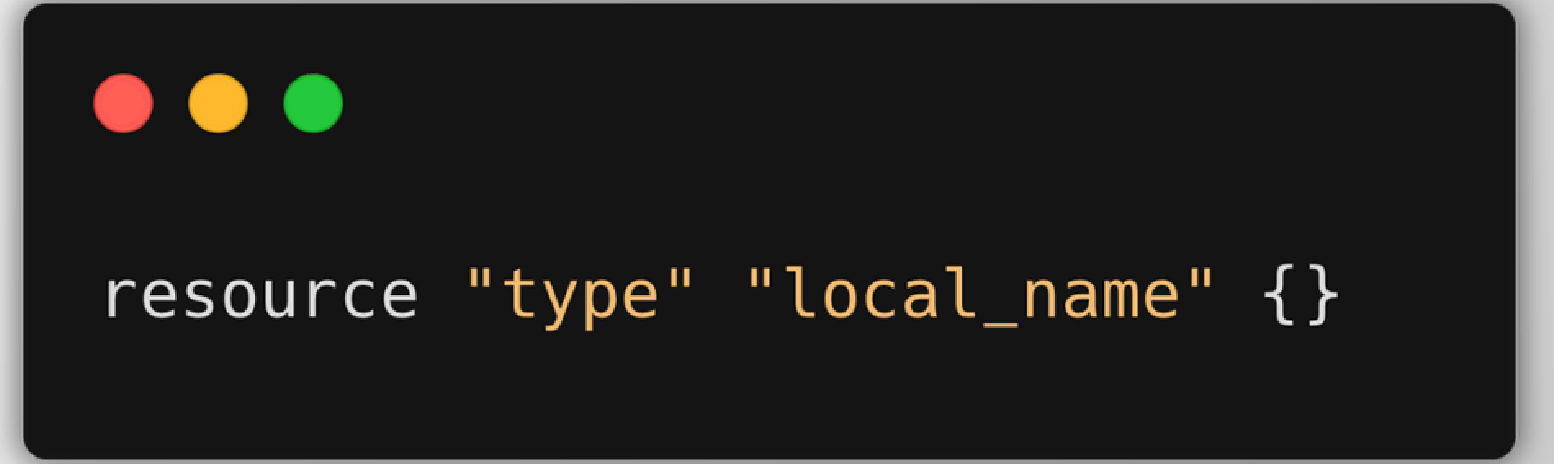
```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region  = "us-east-1"
}
```

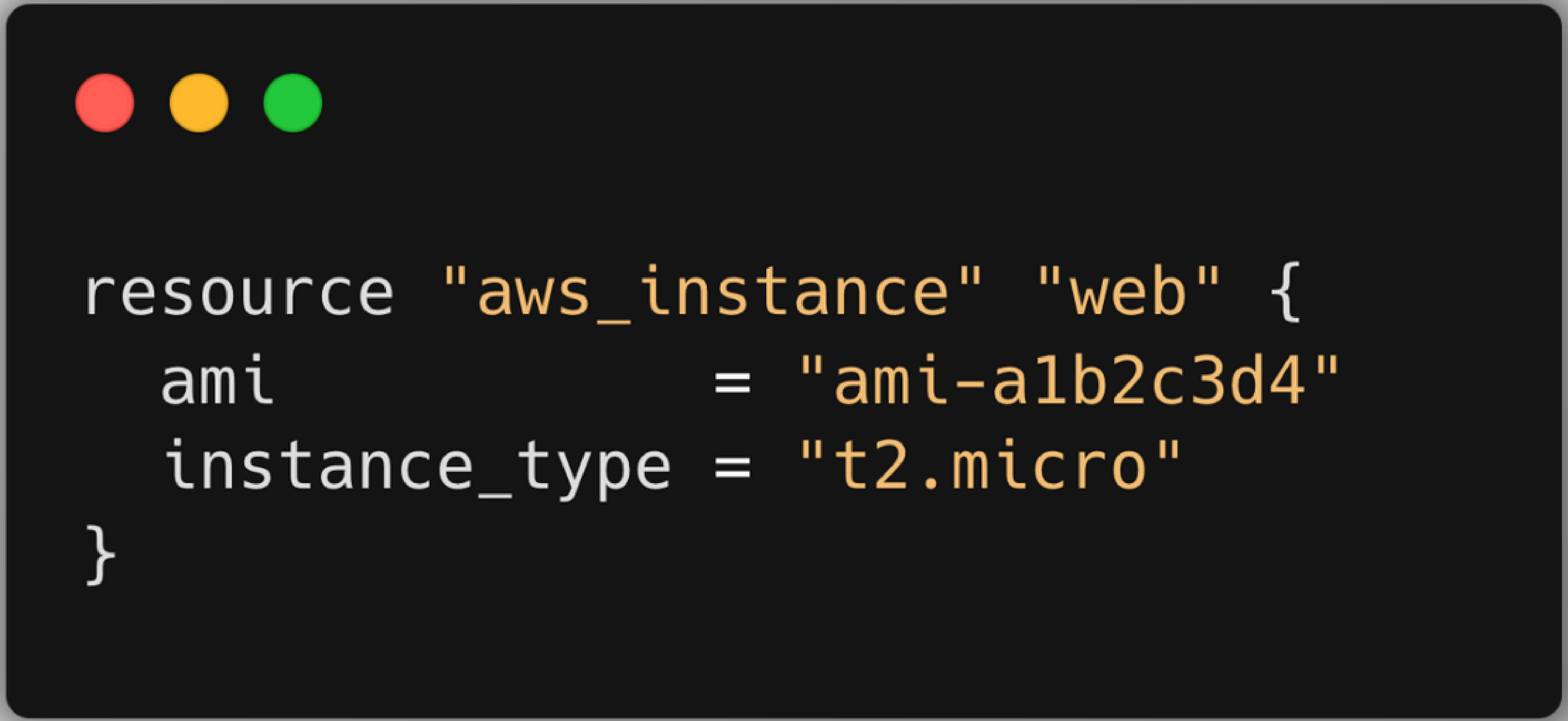
Resources

- Defines the **ACTUAL components** of the infrastructure.

- **Syntax:**



```
resource "type" "local_name" {}
```



```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

Resources

Meta-Arguments

- **Special arguments** that can be used with every resource.
- **List:**
  - depends_on
  - count
  - for_each
  - provider
  - lifecycle

Terraform State

State file (terraform.tfstate)

- A mapping of **terraform config <> real world**
- **Declarative nature** of Terraform
- Stored in **- terraform.tfstate**

```
terraform state -h
```

```
{
  "version": 4,
  "terraform_version": "1.0.11",
  "serial": 5,
  "lineage": "d4d7e3f5-88b5-4f8b-9f47-48bf72a8e3b1",
  "outputs": {
    "instance_ip": {
      "value": "3.121.32.15",
      "type": "string",
      "sensitive": false
    }
  },
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "example",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "ami": "ami-123456",
            "arn": "arn:aws:ec2:us-west-2:123456789012:instance/i-0b5a1c2d3e4f56789",
            "instance_type": "t2.micro",
            "public_ip": "3.121.32.15",
            "tags": {
              "Name": "example-instance"
            }
          }
        }
      ]
    }
  ]
}
```

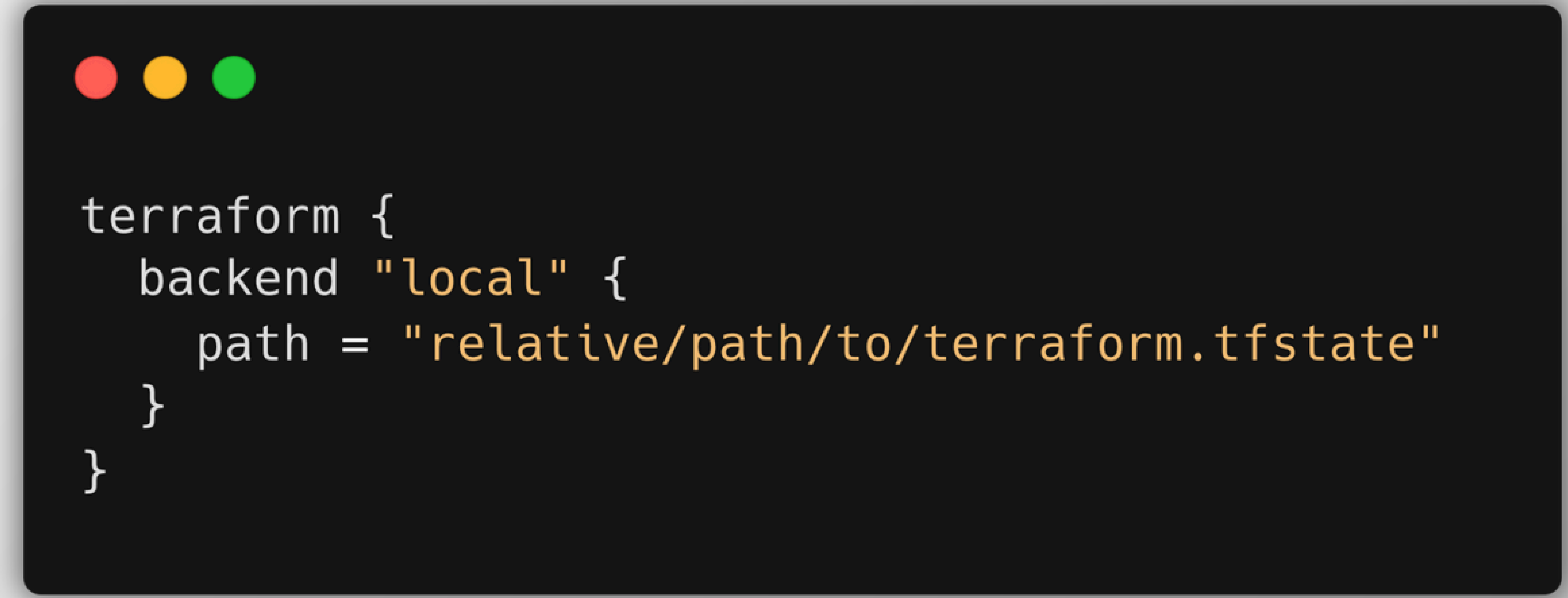
Terraform State

Backend Configuration

- Sensitive data is **VISIBLE** in state file
- Different ways to store the state file - **Backends**

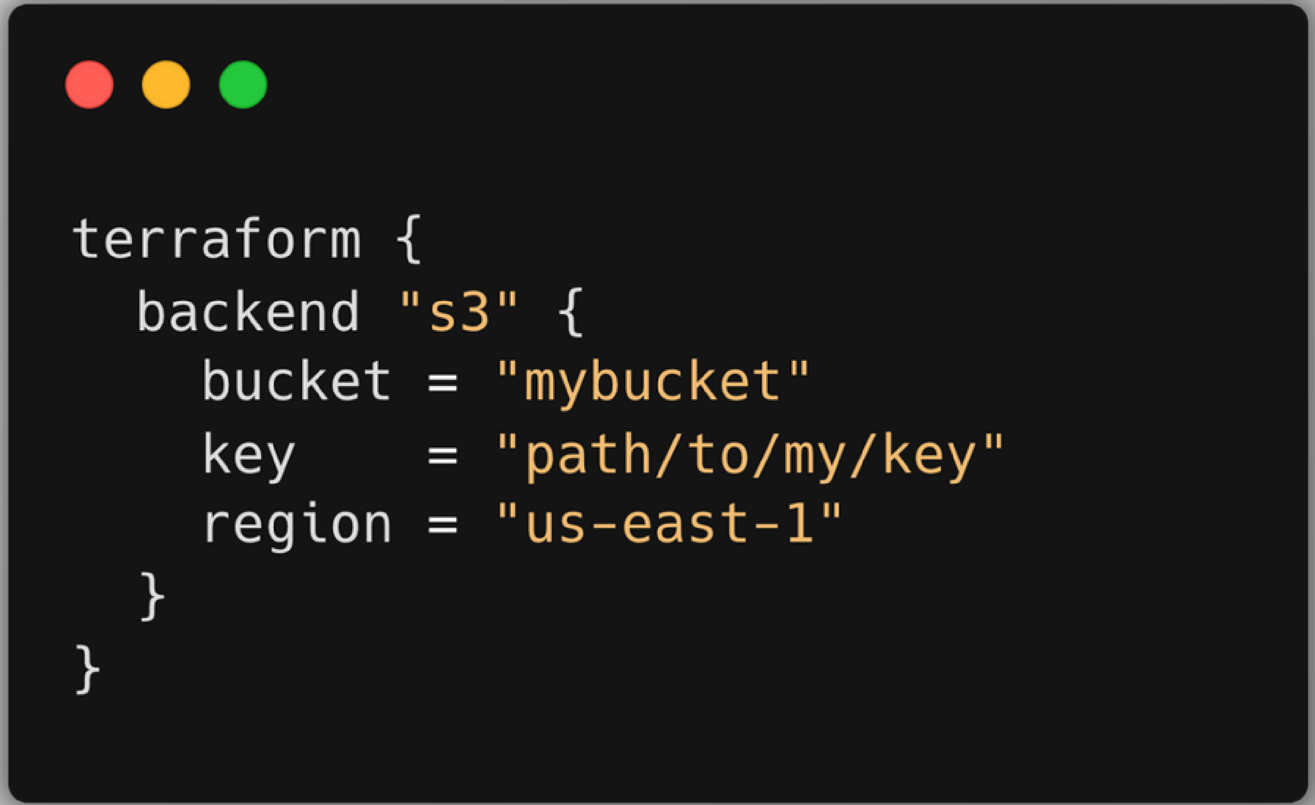
BEST PRACTICE

Local Backend (default)



```
terraform {
  backend "local" {
    path = "relative/path/to/terraform.tfstate"
  }
}
```

Remote Backend (s3 bucket)

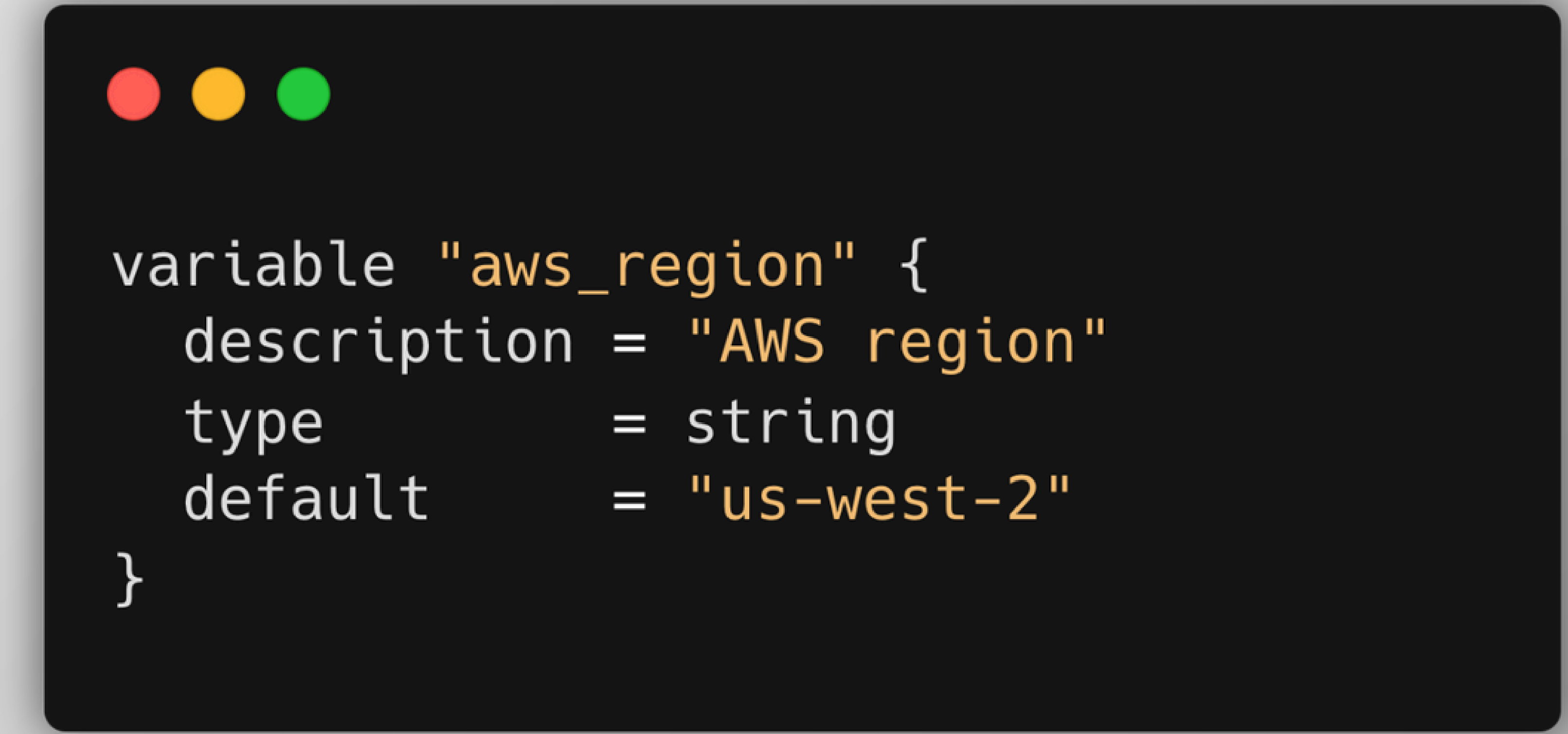


```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

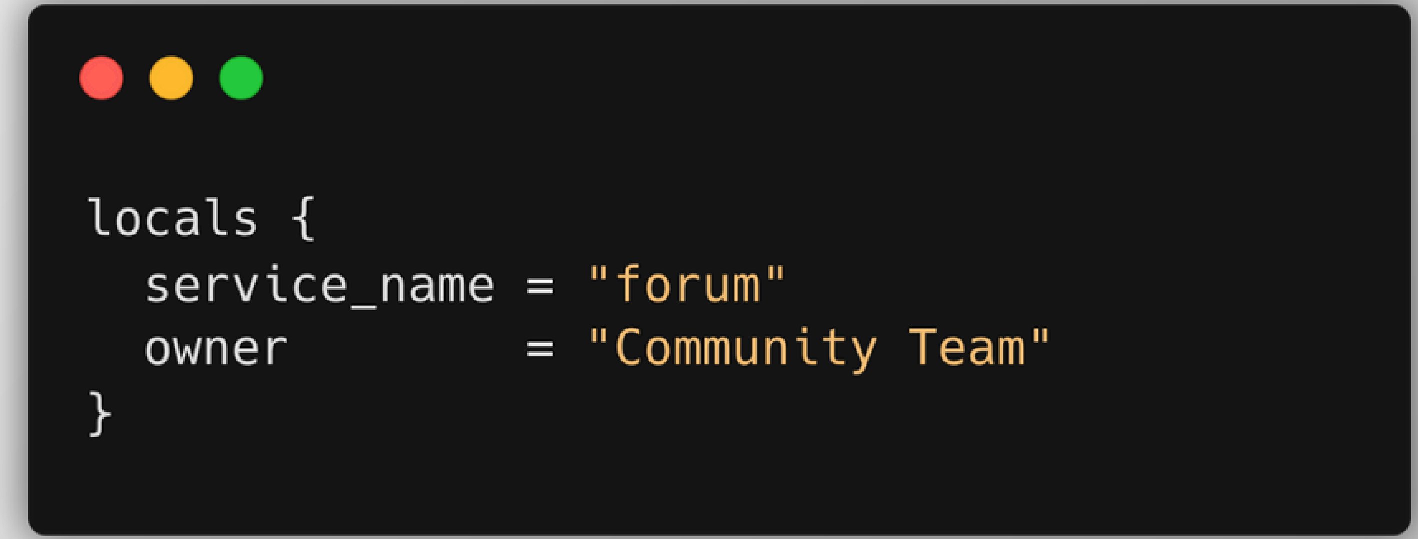
Customizing Terraform Configuration

Variables and Ouputs

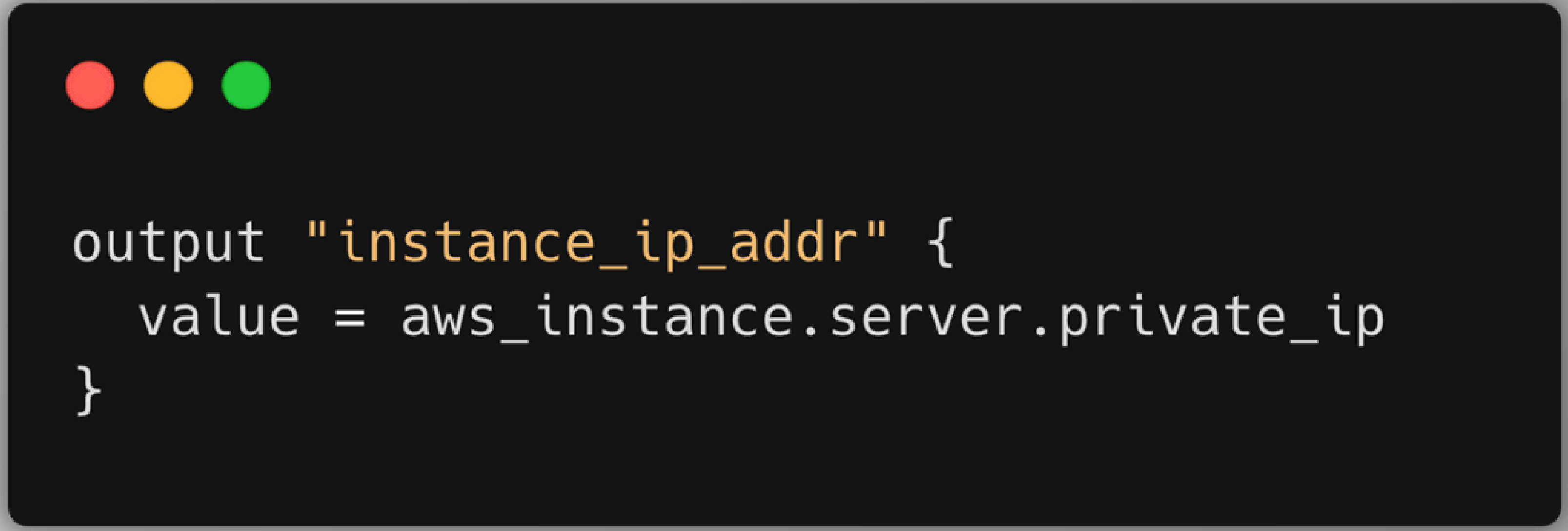
- **Types:**
  - **Input** variables
  - **Output** variables
  - **Local** variables



```
variable "aws_region" {
  description = "AWS region"
  type        = string
  default     = "us-west-2"
}
```



```
locals {
  service_name = "forum"
  owner        = "Community Team"
}
```



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

Customizing Terraform Configuration

Variables and Ouputs

- **Different ways to give value to Input Variables:**
    - **-var or -var-file** CLI flag
    - **terraform.tfvars** file
    - ***.auto.tfvars** file
    - **TF_VAR_<name>** - Environment variable
    - **Default value** in **variables.tf** file

Line of Precedence

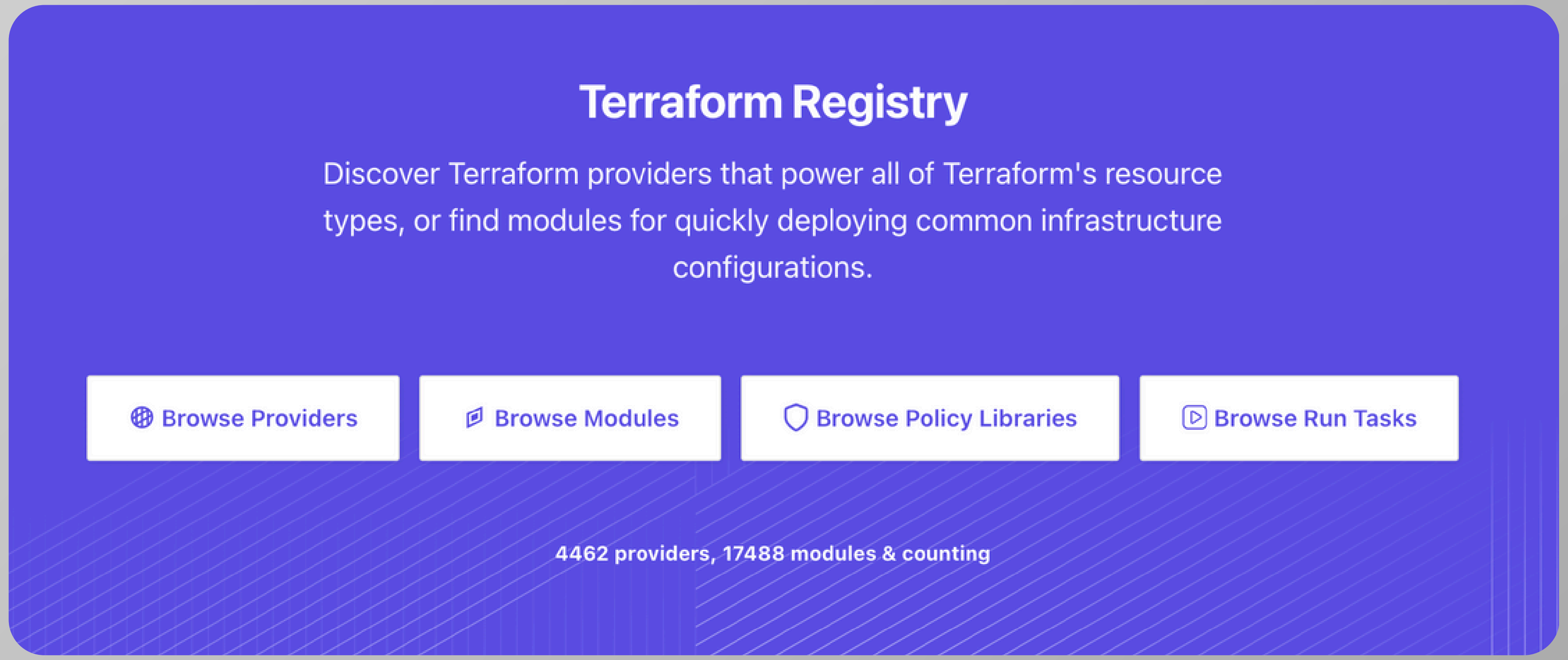
HIGHEST

LOWEST

Terraform Modules

What are they?

- Typically, all the **.tf files** in a working directory.
- A **self-contained package** for multiple resources that are used together.
- **Types:**
    - **Root** module **(default)**
    - **Child** module
    - **Published** module



Terraform Registry

Discover Terraform providers that power all of Terraform's resource types, or find modules for quickly deploying common infrastructure configurations.

4462 providers, 17488 modules & counting

Terraform Modules

Why to use them?

- **It is a Good Practice!**
- **Simplifies** complex infrastructure management
- Once created, it is **Reusable (saves time & effort)**
- Ensure **consistency** - across different environments
- Enables **collaboration**

Publish the modules

Section 4

Complete Practice Project

Bringing it all together!

Thank You

Happy Terraforming!