Erlang Solutions

Infrastructure as Code with Terraform

9/2023

Agenda

- What is Infrastructure as Code (IaC)
- What is Terraform? Why use Terraform?
- How does Terraform work?
- Terraform VS similar technologies
- Getting started with Terraform - Demo
- Q&A
- What's next?

LET'S TALK ABOUT:

INFRASTRUCTURE AS CODE (IAC)

- *Infrastructure as code* is the process of managing and provisioning infrastructure through machine readable definition files.

- We use IaC to automate processes that used to be done manually.

Key Benefits of IaC

- *Provisioning tool*—Deploys infrastructure, not just applications.
- *Easy to use*—For all of us non-geniuses.
- *Declarative*—Say what you want, not how to do it.
- *Free and open source*—Who doesn't like free?
- *Cloud-agnostic*—Deploy to any cloud using the same tool.
- *Expressive and extendable*—You aren't limited by the language.

What’s Terraform?

HashiCorp
Terraform

Terraform is an IaC tool

The provisioning of cloud resources, for instance, is one of the main use cases of Terraform.

It’s a cloud-agnostic, open-source provisioning tool written in the Go language and created by HashiCorp.


WHY USE
TERRAFORM?

- open source

- Terraform can manage infrastructure on multiple cloud platforms

- This simplifies management and orchestration for large-scale, multi-cloud infrastructures.

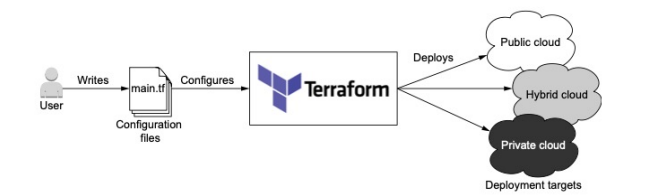
- The human-readable configuration language helps you write infrastructure code quickly.

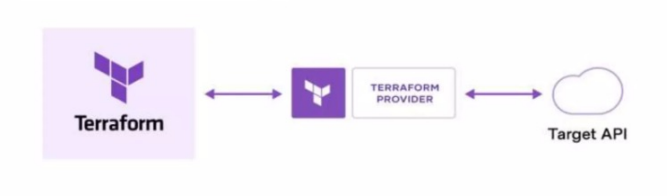
- Terraform's state allows you to track resource changes throughout your deployments.

How does Terraform work?

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs).

Providers enable Terraform to work with virtually any platform or service with an accessible API





How does Terraform work?

Write: You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machine in a Virtual Private Cloud (VPC) network with security groups and a load balancer.

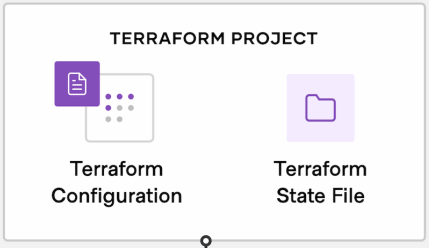
Plan: Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration

Apply: On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For Exam

Terraform Workflow

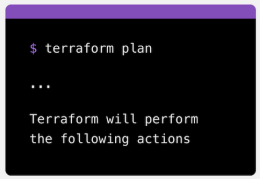
Write

Define infrastructure in configuration files



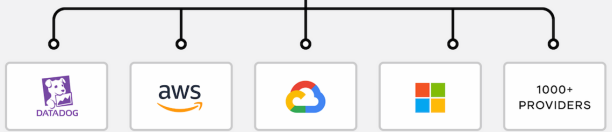
Plan

Review the changes Terraform will make to your infrastructure

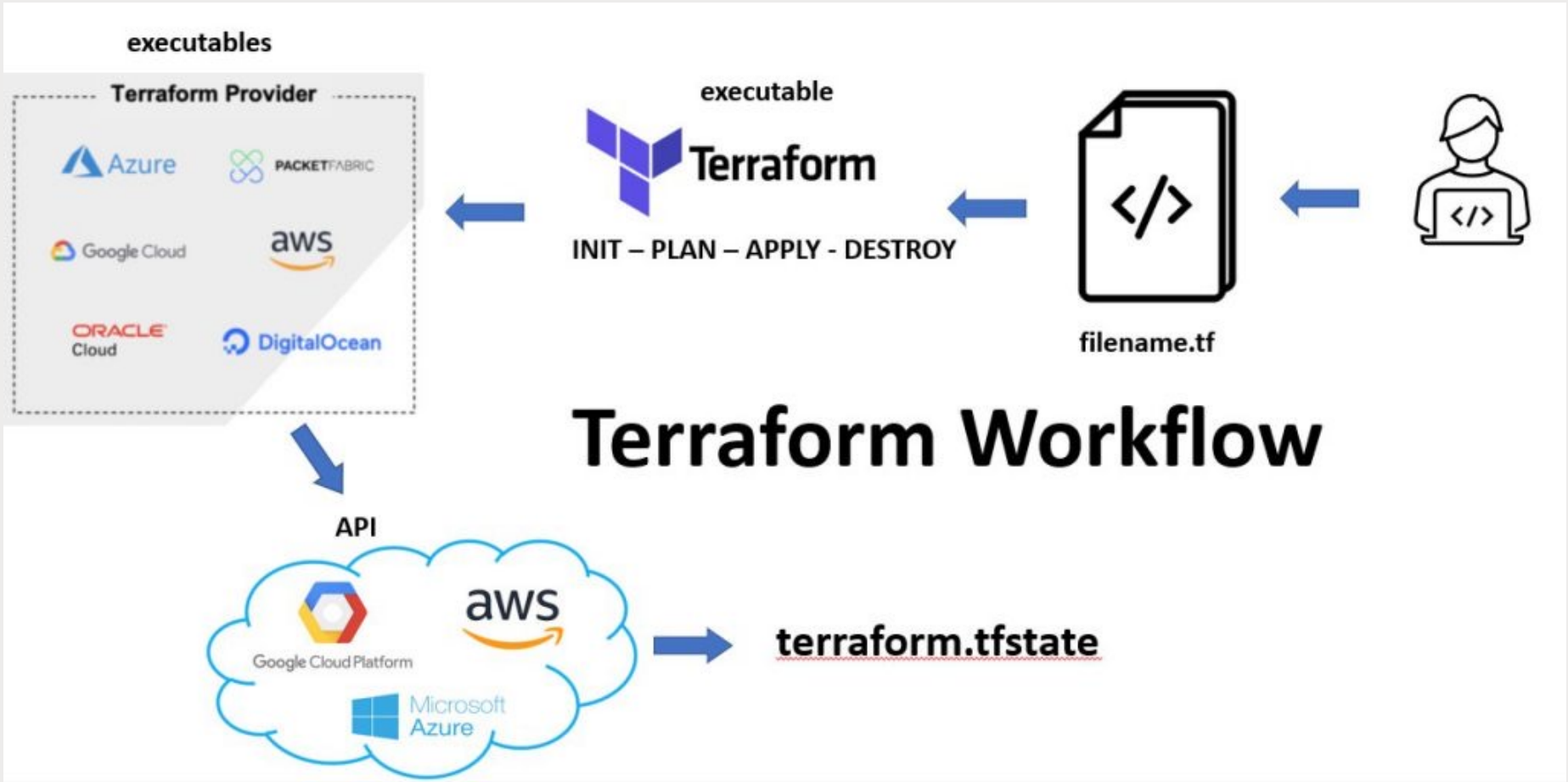


Apply

Terraform provisions your infrastructure and updates the state file.



Terraform Workflow



Terraform vs Competitors?

Name	Key features					
	Provisioning tool	Easy to use	Free and open source	Declarative	Cloud-agnostic	Expressive and extendable
Ansible (www.ansible.com)		X	X		X	X
Chef (www.chef.io)			X	X	X	X
Puppet (www.puppet.com)			X	X	X	X
SaltStack (www.saltstack.com)		X	X	X	X	X
Terraform (www.terraform.io)	X	X	X	X	X	X
Pulumi (www.pulumi.com)	X		X		X	X
AWS CloudFormation (https://aws.amazon.com/cloudformation)	X	X		X		
GCP Deployment Manager (https://cloud.google.com/deployment-manager)	X	X		X		
Azure Resource Manager (https://azure.microsoft.com/features/resource-manager)	X			X		

Demo

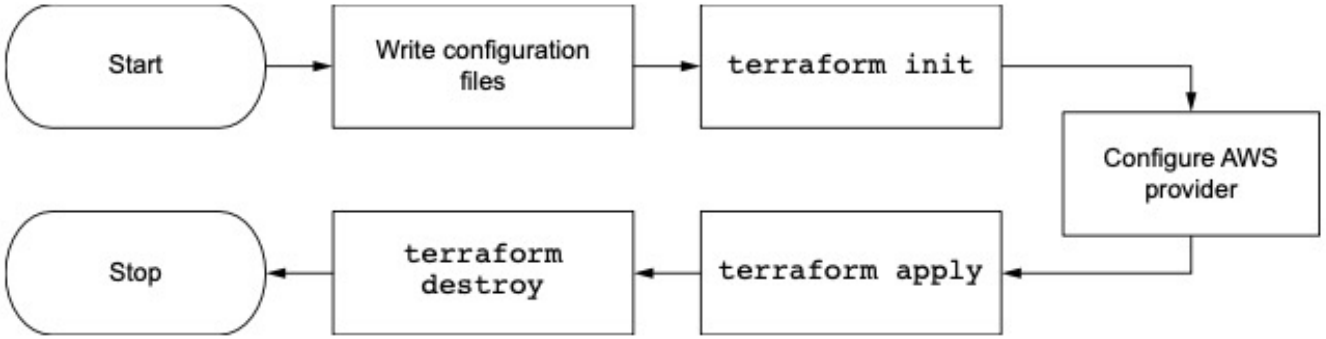
Hello World!

1. Install

https://learn.hashicorp.com/terraform/getting-started/install.html

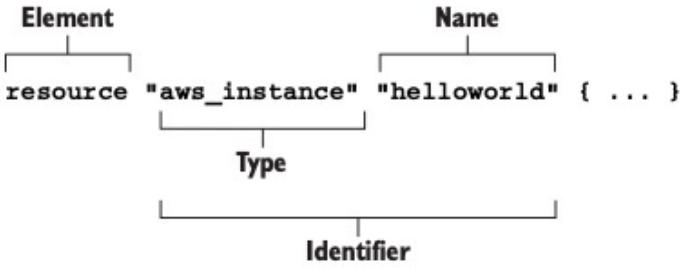
2. Using Terraform to deploy EC2 instances to AWS

1.1 Write Terraform configuration files.
2.2 Configure the AWS provider.
3.3 Initialize Terraform with `terraform init`.
4.4 Deploy the EC2 instance with `terraform apply`.
5.5 Clean up with `terraform destroy`.



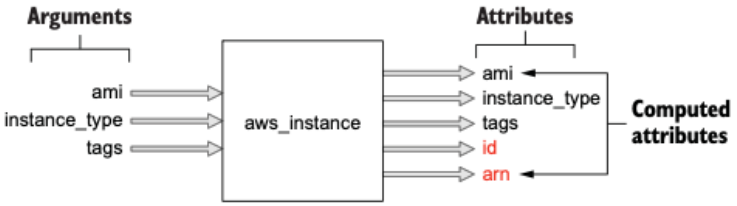
1. Prepare main.tf file

```
resource "aws_instance” "helloworld" {
    ami = "ami-09dd2e08d601bff67"
    instance_type = "t2.micro"
    tags = { Name = "HelloWorld" }}
```



Each resource has inputs and outputs. Inputs are called *arguments*, and outputs are called *attributes*. Arguments are passed through the resource and are also available as resource attributes.

There are also *computed attributes* that are only available after the resource has been created.
Computed attributes contain calculated information about the managed resource.



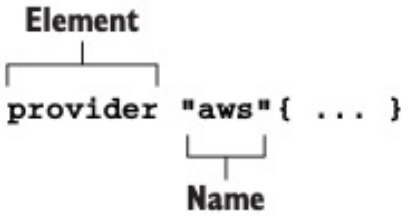
2. Configuring AWS provider

```
provider "aws" {
    region = "us-west-2"}

resource "aws_instance" "helloworld" {
    ami = "ami-09dd2e08d601bff67"
    instance_type = "t2.micro"
    tags = { Name = "HelloWorld" }}
```

Configures a deployment region

Declares the AWS provider

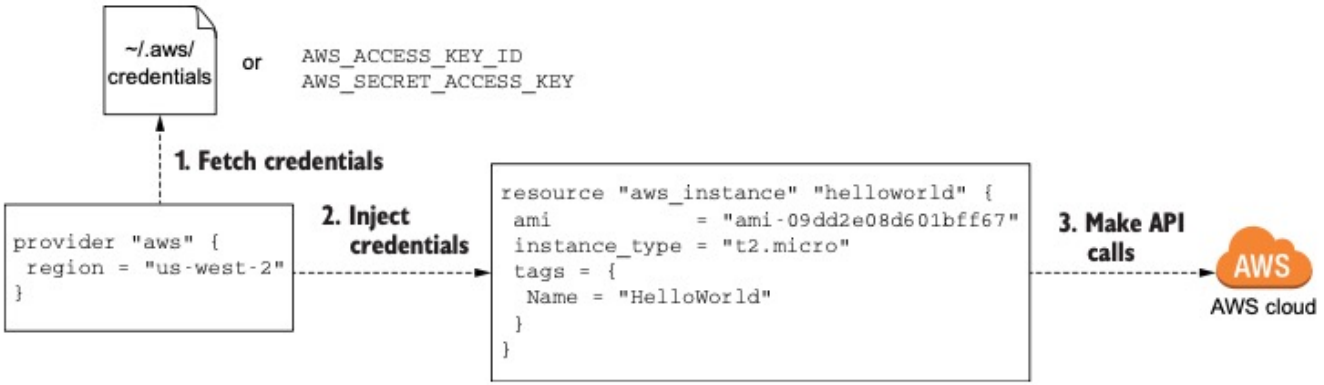
Unlike resources, providers have only one label: `Name`. This is the official name of the provider as published in the Terraform Registry (e.g. “aws” for AWS, “google” for GCP, and “azurerm” for Azure)



Providers don’t have outputs—only inputs. You configure a provider by passing inputs, or *configuration arguments*, to the provider block. Configuration arguments are things like the service endpoint URL, region, and provider version and any credentials needed to authenticate against the API.

2. Configuring AWS provider (Cont.)

Providers don’t have outputs—only inputs. You configure a provider by passing inputs, or *configuration arguments*, to the provider block. Configuration arguments are things like the service endpoint URL, region, and provider version and any credentials needed to authenticate against the API.



2. Initializing

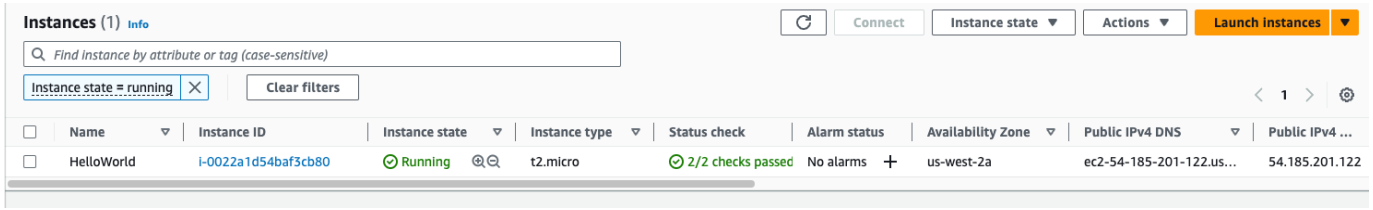
```
17:12:34 /Users/baoadinhcisco.com/esl/learn-terraform-docker-container/manning-code/chapter1/listing1.2$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.16.2...
- Installed hashicorp/aws v5.16.2 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

2. Deploying the EC2 instance

```
17:27:13 /Users/baoadinhcisco.com/esl/learn-terraform-docker-container/manning-code/chapter1/listing1.2$ terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.helloworld will be created
  + resource "aws_instance" "helloworld" {
      + ami                                  = "ami-09dd2e08d601bff67"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + spot_instance_request_id             = (known after apply)
      + subnet_id                            = (known after apply)
```

```
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "HelloWorld"
        }
      + tags_all                             = {
          + "Name" = "HelloWorld"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.helloworld: Creating...
aws_instance.helloworld: Still creating... [10s elapsed]
aws_instance.helloworld: Still creating... [20s elapsed]
aws_instance.helloworld: Still creating... [30s elapsed]
aws_instance.helloworld: Creation complete after 37s [id=i-0022a1d54baf3cb80]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
17:28:04 /Users/baoadinhcisco.com/esl/learn-terraform-docker-container/manning-code/chapter1/listing1.2$
```

Verify that your resource was created by locating it in the AWS console for EC2



3. Destroy the E2C instance

```
17:44:30 /Users/baoadinhcisco.com/esl/learn-terraform-docker-container/manning-code/chapter1/listing1.2$ terraform destroy
aws_instance.helloworld: Refreshing state... [id=i-0022a1d54baf3cb80]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  - destroy
```

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.helloworld: Destroying... [id=i-0022a1d54baf3cb80]
aws_instance.helloworld: Still destroying... [id=i-0022a1d54baf3cb80, 10s elapsed]
aws_instance.helloworld: Still destroying... [id=i-0022a1d54baf3cb80, 20s elapsed]
aws_instance.helloworld: Still destroying... [id=i-0022a1d54baf3cb80, 30s elapsed]
aws_instance.helloworld: Destruction complete after 32s

Destroy complete! Resources: 1 destroyed.
17:45:19 /Users/baoadinhcisco.com/esl/learn-terraform-docker-container/manning-code/chapter1/listing1.2$
```

Q&A


To be continued...

Chapter 2 is a deep dive into Terraform: resource lifecycle and state management.
Examine how Terraform generates and applies execution plans to per- form CRUD operations on managed resources and see how state plays a role in the process.