Terraform

WHAT IS INFRASTRUCTURE AS CODE (IAC):

- **Infrastructure as Code (IaC)** is the managing and provisioning of infrastructure through code instead of through manual processes.
- With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations. It also ensures that you provision the same environment every time.
- Version control is an important part of IaC, and your configuration files should be under source control just like any other software source code file.
- Automating infrastructure provisioning with IaC means that developers don't need to manually provision and manage servers, operating systems, storage, and other infrastructure components each time they develop or deploy an application.
- **There are 2 ways to approach IaC:**
  - Declarative
  - imperative.

DECLARATIVE:

- A declarative approach defines the desired state of the system, including what resources you need and any properties they should have, and an IaC tool will configure it for you.
- A declarative approach also keeps a list of the current state of your system objects, which makes taking down the infrastructure simpler to manage.

IMPERATIVE:

- An imperative approach instead defines the specific commands needed to achieve the desired configuration, and those commands then need to be executed in the correct order.

NOTE:

- Many IaC tools use a declarative approach and will automatically provision the desired infrastructure. If you make changes to the desired state, a declarative IaC tool will apply those changes for you. An imperative tool will require you to figure out how those changes should be applied.

BENEFITS OF IAC:

- Provisioning infrastructure has historically been a time-consuming and costly manual process. As virtualization, containers, and cloud computing have become the norm, infrastructure management has moved away from physical hardware in data centers—providing many benefits, but also creating some new challenges.
- With cloud computing, the number of infrastructure components has grown, more applications are being released to production on a daily basis, and infrastructure needs to be able to be spun up, scaled, and taken down frequently. Without an IaC practice in place, it becomes increasingly difficult to manage the scale of today's infrastructure.

BENEFITS:

- Cost reduction
- Increase in speed of deployments
- Reduce errors
- Improve infrastructure consistency
- Eliminate configuration drift

WHY DOES IAC MATTER FOR DEVOPS:

- IaC is an important part of implementing DevOps practices and continuous integration/continuous delivery (CI/CD).
- IaC takes away the majority of provisioning work from developers, who can execute a script to have their infrastructure ready to go. That way, application deployments aren't held up waiting for the infrastructure, and sysadmins aren't managing time-consuming manual processes.
- Aligning development and operations teams through a DevOps approach leads to fewer errors, manual deployments, and inconsistencies.
- IaC helps you to align development and operations because both teams can use the same description of the application deployment, supporting a DevOps approach.
- DevOps best practices are also applied to infrastructure in IaC. Infrastructure can go through the same CI/CD pipeline as an application does during software development, applying the same testing and version control to the infrastructure code.

IAC TOOLS:

- Server automation and configuration management tools can often be used to achieve IaC. There are also solutions specifically for IaC.
- These are some popular choices:

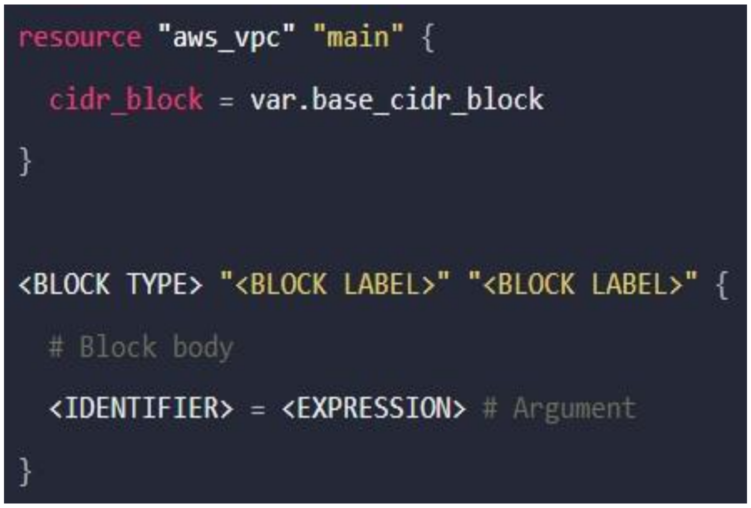
    - Chef
    - Puppet
    - Red Hat Ansible Automation Platform
    - Saltstack
    - Terraform
    - AWS CloudFormation
    - Microsoft ARM
    - Vagrant

TERRAFORM:

- Terraform is an open source **“Infrastructure as Code” tool,** created by **HashiCorp**.
- It is used to **build, change,** and **version** cloud and on-prem resources safely and efficiently.
- A declarative coding tool, uses a high-level configuration language called **HCL** (**HashiCorp Configuration Language**).

TERRAFORM LANUAGE:

- The main purpose of the Terraform language is **declaring resources,** which represent **infrastructure objects.**
- A Terraform configuration is a complete document in the Terraform language that tells Terraform how to manage a given collection of infrastructure.
- A configuration can consist of **multiple files** and **directories.**
- The syntax of the Terraform language consists of only a few basic elements:



```
resource "aws_vpc" "main" {
  cidr_block = var.base_cidr_block
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

BLOCKS:

- **Blocks** are containers for other content and usually represent the configuration of some kind of object, like a resource.
- Blocks have a block type, can have zero or more labels, and have a body that contains any number of arguments and nested blocks.
- Most of Terraform's features are controlled by top-level blocks in a configuration file.

ARGUMENTS:

- Arguments assign a value to a name. They appear within blocks.

EXPRESSIONS:

- Expressions represent a value, either literally or by referencing and combining other values.
- They appear as values for arguments, or within other expressions.

IDENTIFIERS:

- Argument names, block type names, and the names of most Terraform-specific constructs like resources, input variables, etc. are all identifiers.
- Identifiers can contain letters, digits, underscores (_), and hyphens (-).
- The first character of an identifier must not be a digit, to avoid ambiguity with literal numbers.

WHY TERRAFORM:

- Opensource
- Platform agnostic
- Immutable Infrastructure
- Easy Code readability
- Manage any infrastructure
- Track your infrastructure
- Automate changes
- Standardize configurations
- Collaborate

USE CASES:

- Multi-Cloud Deployment
- Application Infrastructure Deployment, Scaling, and Monitoring Tools
- Self-Service Clusters
- Policy Compliance and Management
- PaaS Application Setup
- Software Defined Networking
- Kubernetes
- Parallel Environments

TERRAFORM COMMUNITY:

- We welcome questions, suggestions, and contributions from the community.

Ask questions: https://discuss.hashicorp.com/c/terraform-core/27

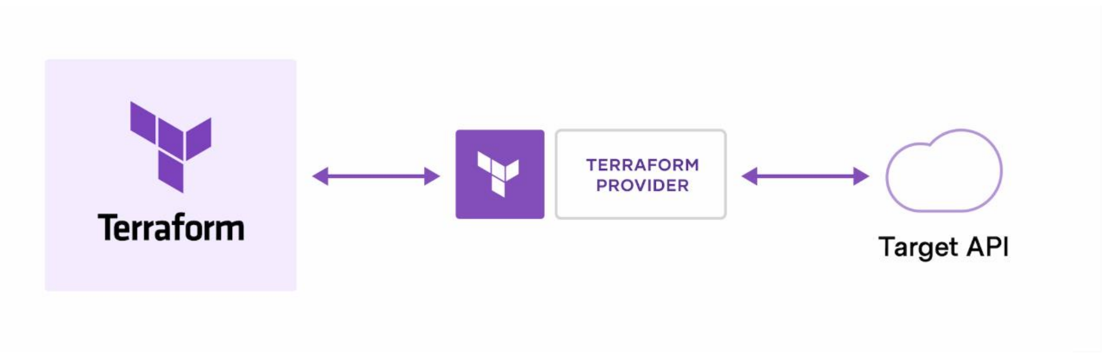
Read our contributing guide:
https://github.com/hashicorp/terraform/blob/main/.github/CONTRIBUTING.md

Submit an issue for bugs and feature requests:

https://github.com/login?return_to=https%3A%2F%2Fgithub.com%2Fhashicorp%2Fterraform%2Fissues%2Fnew%2Fchoose

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.



- HashiCorp and the Terraform community have already written thousands of providers to manage many different types of resources and services.
- Providers are including, Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog, and many more.

TERRAFORM WORKFLOW CORE STAGES:

WRITE:

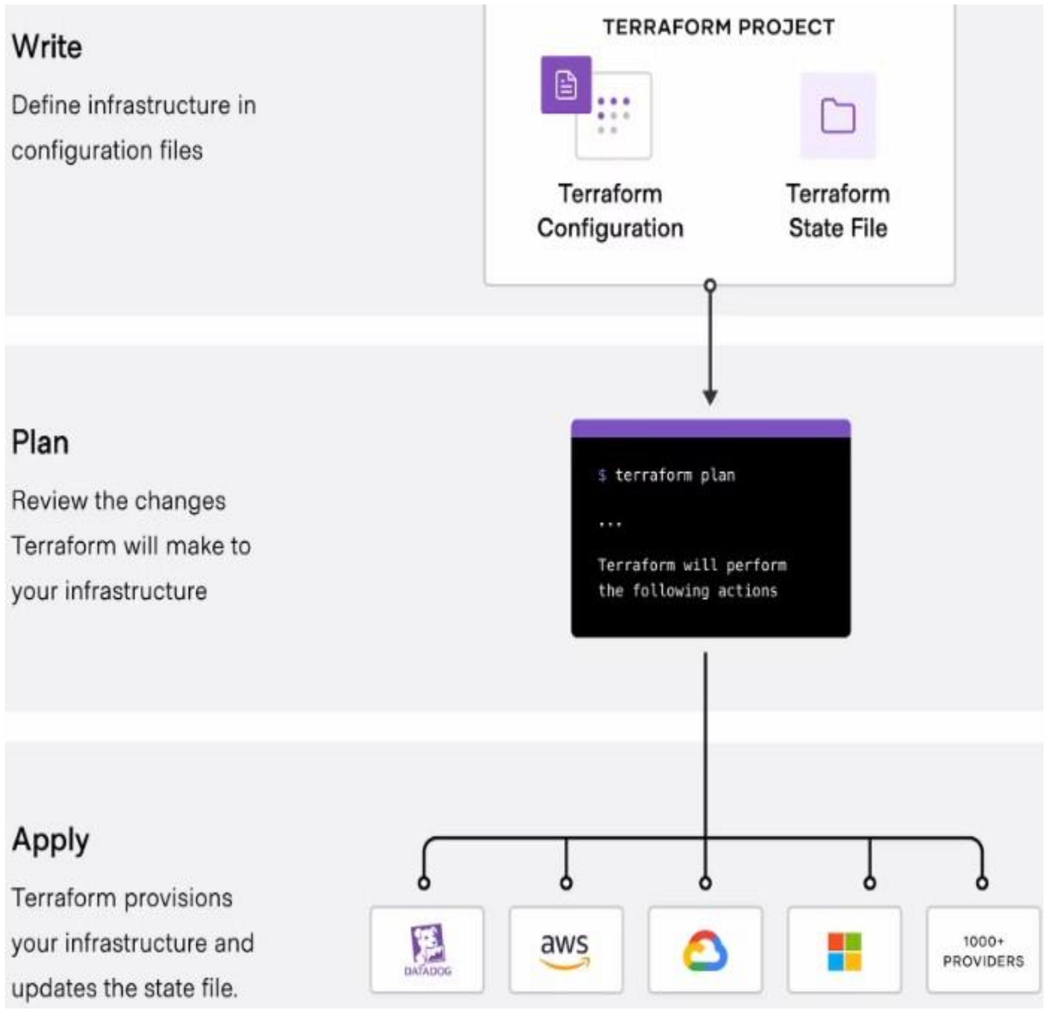
- You define resources, which may be across multiple cloud providers and services. For example, create a configuration to deploy an application on virtual machines.

PLAN:

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

APPLY:

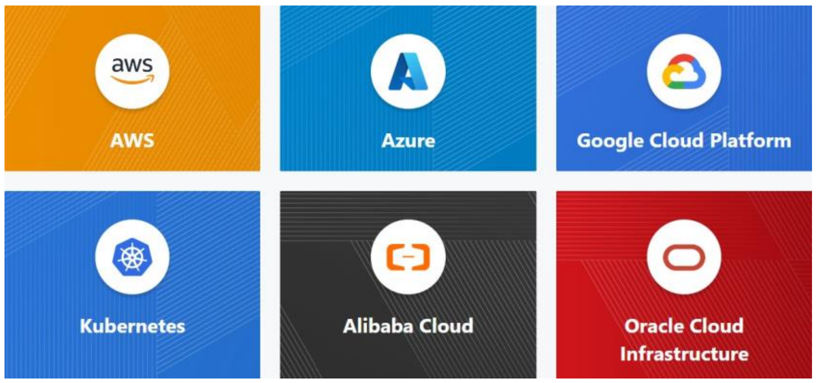
- On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies.


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

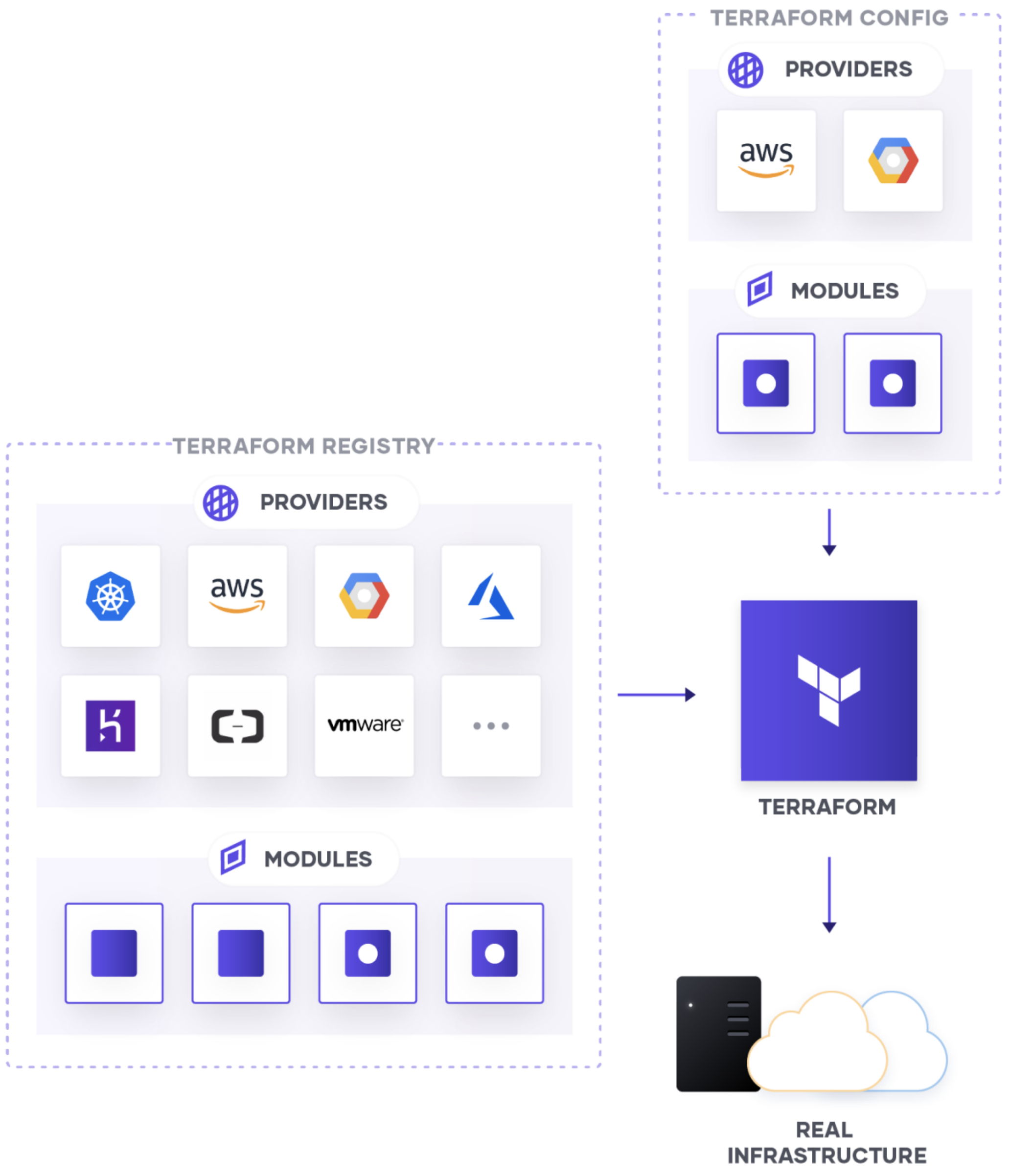
```
$ terraform plan

...

Terraform will perform
the following actions
```

TERRAFORM PROVIDERS:

- Terraform providers are plugins that implement resource types.
- Providers are a logical abstraction of an upstream API. They are responsible for understanding API interactions and exposing resources.



HOW TERRAFORM, PROVIDERS AND MODULES WORK:



TERRAFORM:

- Terraform provisions, updates, and destroys infrastructure resources such as physical machines, VMs, network switches, containers, and more.

CONFIGURATIONS:

- Configurations are code written for Terraform, using the human-readable HashiCorp Configuration Language (HCL) to describe the desired state of infrastructure resources.

PROVIDERS:

- Providers are the plugins that Terraform uses to manage those resources. Every supported service or infrastructure platform has a provider that defines which resources are available and performs API calls to manage those resources.

MODULES:

- Modules are reusable Terraform configurations that can be called and configured by other configurations. Most modules manage a few closely related resources from a single provider.

TERRAFORM REGISTRY:

- The Terraform Registry makes it easy to use any provider or module. To use a provider or module from this registry, just add it to your configuration; when you run **`terraform init`,** Terraform will automatically download everything it needs.