WHAT IS TERRAFORM? TERRAFORM & ITS IAC ROLE EXPLAINED

Managing infrastructure is a core requirement for most modern applications. Even in PaaS or serverless environments, there will still be components that require user intervention for customization and management. With the ever-increasing complexity of software applications, more and more infrastructure modifications are required to facilitate the functionality of the software.

It is unable to keep up with the rapid development cycles with manual infrastructure management. It will create bottlenecks leading to delays in the delivery process.

Infrastructure as Code (IaC) has become the solution to this issue—allowing users to align infrastructure changes with development. It also facilitates faster automated repeatable changes by codifying all the infrastructure and configuration and managing them through the delivery pipeline.

Terraform is one of the leading platform agnostic IaC tools that allow users to define and manage infrastructure as code. In this article, let's dig into what Terraform is and how we can utilize it to manage infrastructure at scale.

What is Infrastructure as Code?

Before moving into Terraform, we need to understand Infrastructure as Code. To put it simply, IaC enables users to codify their infrastructure. It allows users to:

- Create repeatable version-controlled configurations
- Integrate them as a part of the CI/CD pipeline
- Automate the infrastructure management

If an infrastructure change is needed in a more traditional delivery pipeline, the infrastructure team

will have to be informed. The delivery pipeline cannot proceed until the change is made to the environment. Having an inflexible manual process will hinder the overall efficiency of the SDLC with practices like DevOps leading to fast yet flexible delivery pipelines.

IaC allows infrastructure changes to be managed through a source control mechanism like Git and integrated as an automated part of the CI/CD pipeline. It not only automates infrastructure changes but also facilitates auditable changes and easy rollbacks of changes if needed.

What is Terraform?

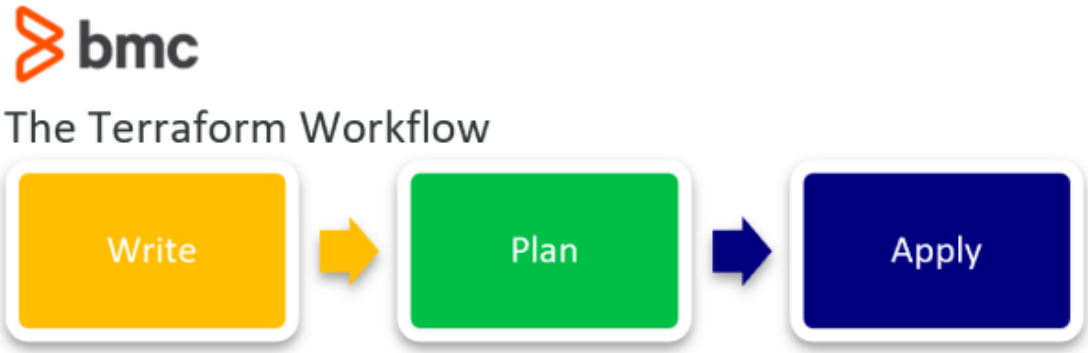
Terraform is an open-source infrastructure as a code tool from HashiCorp. It allows users to define both on-premises and cloud resources in human-readable configuration files that can be easily versioned, reused, and shared. Terraform can be used to manage both low-level components (like compute, storage, and networking resources) as well as high-level resources (DNS, PaaS, and SaaS components).

Terraform is a declarative tool further simplifying the user experience by allowing users to specify the expected state of resources without the need to specify the exact steps to achieve the desired state of resources. Terraform manages how the infrastructure needs to be modified to achieve the desired result.

Terraform is a platform-agnostic tool, meaning that it can be used across any supported provider. Terraform accomplishes this by interacting with the APIs of cloud providers. When a configuration is done through Terraform, it will communicate with the necessary platform via the API and ensure the defined changes are carried out in the targeted platform. With more than 1,700 providers from HasiCorp and the Terraform community available with the Terraform Registry, users can configure resources from leading cloud providers like Azure, AWS, GCP, and Oracle Cloud to more domain-specific platforms like Cloudflare, Dynatrace, elastic stack, datadog, and Kubernetes.

The Terraform workflow

The Terraform workflow is one of the simplest workflows only consisting of three steps to manage any type of infrastructure. It provides users the flexibility to change the workflow to support their exact implementation needs.



1. Write

The first stage of the workflow is where users create the configurations to define or modify the underlying resources. It can be as simple as provisioning a simple compute instance in a cloud provider to deploy a multi-cloud Kubernetes cluster. This writing part can be facilitated either

through HasiCorp Configuration Language (HCL), the default language to define resources or using the Cloud Development Kit for Terraform (CDKTF) which allows users to define resources using any supported common programming languages like Python, C#, Go, and Typescript.

2. Plan

This is the second stage of the workflow where Terraform will look at the configuration files and create an execution plan. It enables users to see the exact charges that will happen to the underlying infrastructure from what new resources are getting created, resourced, modified, and deleted.

3. Apply

This is the final stage of the workflow which takes place if the plan is satisfactory once the user has confirmed the changes. Terraform will carry out the changes to achieve the desired state in a specific order respecting all the resource dependencies. It will happen regardless of whether you have defined dependencies in the configuration. Terraform will automatically identify the resource dependencies of the platform and execute the changes without causing issues.

Terraform uses the state to keep track of all the changes to the infrastructure and detect config drifts. It will create a state file at the initial execution and subsequently update the state file with new changes. This state file can be stored locally or in a remote-backed system like an s3 bucket. Terraform always references this state file to identify the resources it manages and keep track of the changes to the infrastructure.

Benefits of Terraform

Let's look at why so many people appreciate Terraform

- **Declarative nature.** A declarative tool allows users to specify the end state and the IaC tools will automatically carry out the necessary steps to achieve the user configuration. It is in contrast to other imperative IaC tools where users need to define the exact steps required to achieve the desired state.
- **Platform agnostics.** Most IaC tools like AWS CloudFormation and Azure Resource templates are platform specific. Yet, Terraform allows users to use a single tool to manage infrastructure across platforms with applications using many tools, platforms, and multi-cloud architectures.
- **Reusable configurations.** Terraform encourages the creation of reusable configurations where users can use the same configuration to provision multiple environments. Additionally Terraform allows creating reusable components within the configuration files with modules.
- **Managed state.** With state files keeping track of all the changes in the environment, all modifications are recorded and any unnecessary changes will not occur unless explicitly specified by the user. It can be further automated to detect any config drifts and automatically fix the drift to ensure the desired state is met at all times.
- **Easy rollsbacks.** As all configurations are version controlled and the state is managed, users can easily and safely roll back most infrastructure configurations without complicated reconfigurations.
- **Integration to CI/CD.** While IaC can be integrated into any pipeline, Terraform provides a simple three-step workflow that can be easily integrated into any CI/CD pipeline. It helps to

completely automate the infrastructure management.

(Learn how to set up a CI/CD pipeline.)

How to use Terraform

You can start using Terraform by simply installing it in your local environment. Terraform supports Windows, Linux, and macOS environments. It provides users the option to install manually using a pre-compiled binary, or use a package manager like Homebrew on Mac, Chocolatey on Windows, Apt/Yum on Linux. It offers users the flexibility to install Terraform in their environments and integrate it into their workflows.

HashiCorp also provides a managed solution called Terraform Cloud. It provides users with a platform to manage infrastructure on all supported providers without the hassle of installing or managing Terraform itself. Terraform Cloud consists of features like;

- Remote encrypted state storage
- Direct CI/CD integrations
- Fully remote and SOC2 compliant collaborative environment
- Version Controls
- Private Registry to store module and Policy as Code support to configure security and compliance policies
- Complete auditable environment.
- Cost estimations before applying infrastructure changes in supported providers.

Additionally, Terraform Cloud is deeply integrated with other HasiCrop Cloud Platform services like Vault, Consul, and Packer to manage secrets, provide service mesh and create images. All these things allow users to manage their entire infrastructure using the HasiCorp platform.

Using Terraform to provision resources

Finally, let's look at a simple Terraform configuration. Assume you want to deploy a web server instance in your AWS environment. It can be done by creating an HCL configuration similar to the following.

```
terraform {

required_providers {

aws = {
source  = "hashicorp/aws"
version = "~> 3.74"
}
}
}
# Specifiy the Provider
provider "aws" {
region  = var.region
# AWS Credentials
```

```
access_key = "xxxxxxxxxxxx"
secret_key = "yyyyyyyyyyyyy"
default_tags {
tags = {
Env            = "web-server"
Resource_Group = "ec2-instances"
}
}
}

# Configure the Security Group

resource "aws_security_group" "web_server_access" {
name        = "server-access-control-sg"
description = "Allow Access to the Server"
vpc_id      = local.ftp_vpc_id
ingress {

from_port        = 22
to_port          = 22
protocol         = "tcp"
cidr_blocks      =
ipv6_cidr_blocks =
}
ingress {

from_port        = 443
to_port          = 443
protocol         = "tcp"
cidr_blocks      =
ipv6_cidr_blocks =
}
egress {
from_port        = 0
to_port          = 0
protocol         = "-1"
cidr_blocks      =
ipv6_cidr_blocks =

}

tags = {
Name     = "server-access-control-sg"
}
}

# Get the latest Ubuntu AMI
```

```
data "aws_ami" "ubuntu" {
most_recent = true
owners      =  # Canonical
filter {
name   = "name"
values =
}
filter {

name   = "virtualization-type"
values =
}
}

# Elastic IP

resource "aws_eip" "web_server_eip" {
instance = aws_instance.web_server.id
vpc      = true
tags = {
Name     = "web-server-eip"
Imported = false
}
}

# Web Server Instance

resource "aws_instance" "web_server" {

ami                         = data.aws_ami.ubuntu.id
instance_type               = "t3a.small"
availability_zone           = "eu-central-1a"
subnet_id                   = "subnet-yyyyyy"
associate_public_ip_address = false
vpc_security_group_ids      = "sg-xxxxxxx"
key_name                    = "frankfurt-test-servers-common"
disable_api_termination     = true
monitoring                  = true
credit_specification {
cpu_credits = "standard"
}

root_block_device {
volume_size = 30
}

tags = {
```

```
Name     = "web-server"
}
}
```

In the HCL file, we are pointing to the AWS provider and providing the AWS credentials (Access Key and Secret Key) which will be used to communicate with AWS and provision resources.

We have created a security group, elastic IP, and ec2 instance with the necessary configuration options to obtain the desired state in the configuration itself. Additionally, the AMI used for the ec2 instance is also queried by the configuration itself by looking for the latest Ubuntu image. Its easily understandable syntax pattern allows users to easily define their desired configurations using HCL and execute them via Terraform. You can have an in-depth look at all the available options for the AWS provider in the Terraform documentation.

Terraform summary

Terraform is a powerful IaC tool that aims to provide the best balance between user friendliness and features. Its declarative and platform-agnostic nature allows this tool to be used in any supported environment without being vendor-locked or having to learn new platform-specific tools. The flexible workflow and configuration options of Terraform allow it to be run in local environments.

Furthermore, users have the flexibility to select the exact implementation suited for their needs to manage Terraform Cloud solutions. All this has led Terraform to become one of the leading IaC tools.

Related reading

- BMC DevOps Blog
- IT Infrastructure Automation: A Beginner's Guide
- Serverless vs Function-as-a-Service (FaaS): What's The Difference?
- GitOps Explained: Concepts, Benefits & Getting Started
- DataOps Vs DevOps: What's The Difference?
- The Complete DevOps Certifications Guide