**EXPERIMENT NO. – 5**

**AIM:** To explain terraform lifecycle, core concepts / terminologies and install it on a Linux Machine.

**LO:** LO1:- To explain the fundamentals of Cloud Computing and be fully proficient with Cloud based DevOps solution deployment options to meet your business requirements.

LO3:- To apply best practices for managing infrastructure as code environment and use terraform to define and deploy cloud infrastructure.

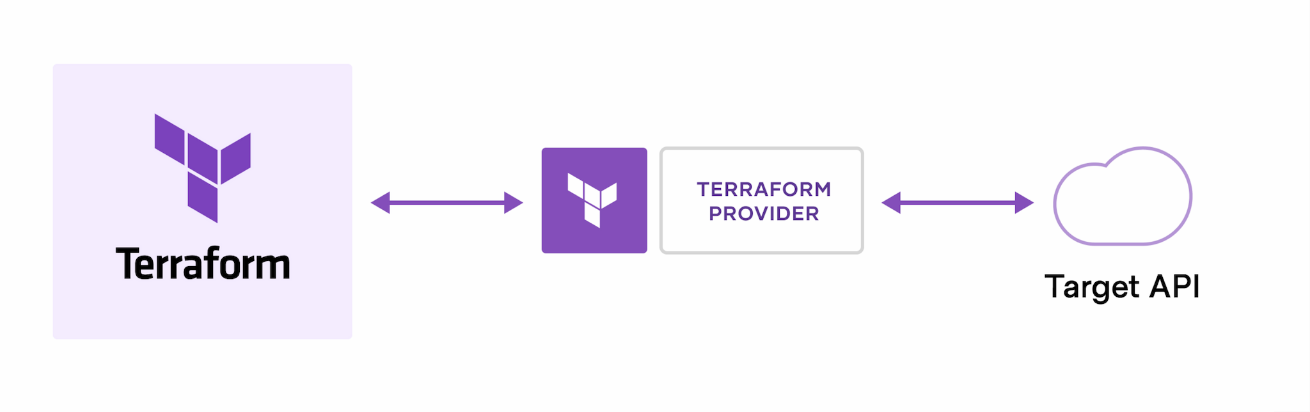
**THEORY:**

* **What is 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. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

* **How Terraform works?**

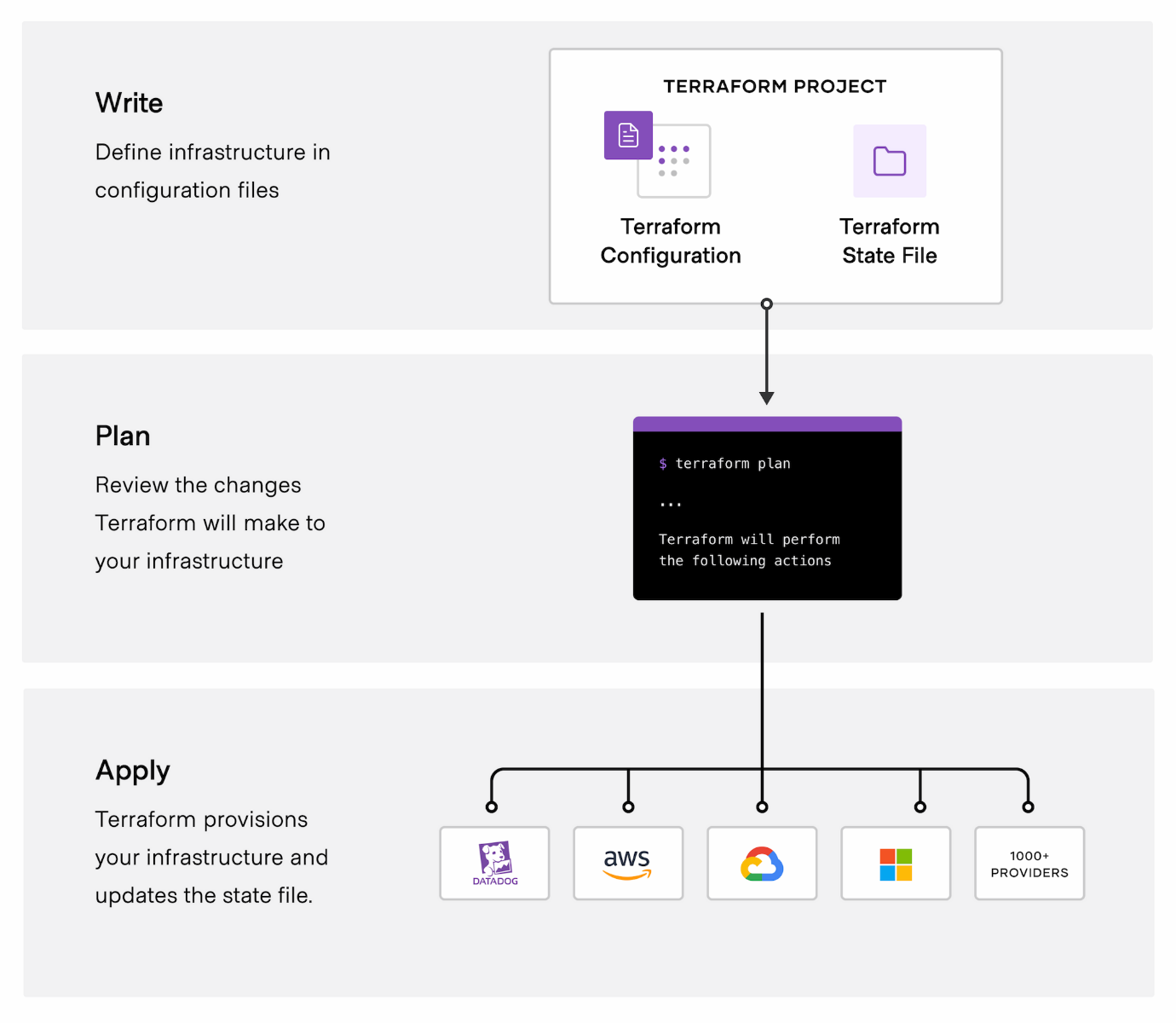
Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.



HashiCorp and the Terraform community have already written **more than 1700 providers** to manage thousands of different types of resources and services, and this number continues to grow. You can find all publicly available providers on the [Terraform Registry](https://registry.terraform.io/), including Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog, and many more.

The core Terraform workflow consists of three stages:

* **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 machines 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 example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.



* **Terraform Cloud**

Terraform Cloud is a SaaS application that runs Terraform in a stable, remote environment and securely stores state and secrets. It includes a rich user interface that helps you better understand your Terraform operations and resources, allows you to define role-based access controls, and offers a private registry for sharing modules and providers. Terraform Cloud also integrates with the Terraform CLI and connects to common version control systems (VCS) like GitHub, GitLab, and Bitbucket. When you connect a Terraform Cloud workspace to a VCS repository, new commits and changes can automatically trigger Terraform plans. Terraform Cloud also offers an API, allowing you to integrate it into existing workflows.

* **Why Terraform Cloud?**

Terraform Cloud lets you:

* Run Terraform from the local CLI or in a remote environment, trigger operations through your version control system, or use an API to integrate Terraform Cloud into your existing workflows.
* Ensure that only approved teams can access, edit, and provision infrastructure with Terraform Cloud workspaces, single sign-on, and role-based access controls.
* Securely store and version Terraform state remotely, with encryption at rest. Versioned state files allow you to access state file history.
* Publish configuration modules in the Terraform Cloud private registry that define approved infrastructure patterns. For example, a module may allow users to choose the cloud provider on which to deploy their Java application. This allows consumers to implement your organization’s best practices without becoming infrastructure or cloud experts.
* Enforce best practices and security rules with the Sentinel embedded policy as code framework. For example, policies may restrict regions for production deployments.
* **Terraform Resource Lifecycle**

Resources have a strict lifecycle, and can be thought of as basic state machines. Understanding this lifecycle can help better understand how Terraform generates an execution plan, how it safely executes that plan, and what the resource provider is doing throughout all of this.

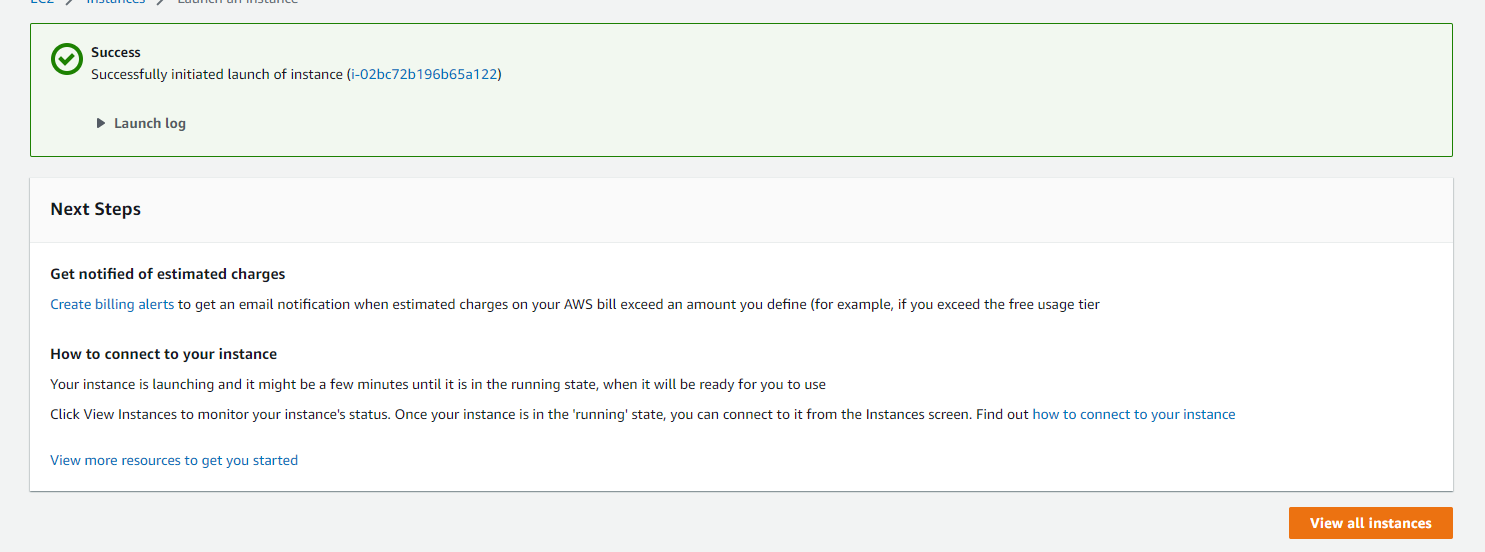
* **Lifecycle**

A resource roughly follows the steps below:

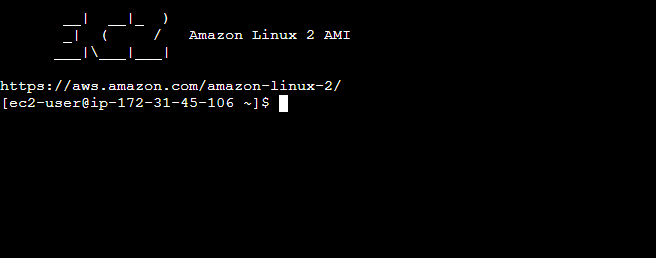
1. [ValidateResource](https://www.terraform.io/internals/lifecycle#validateresource) is called to do a high-level structural validation of a resource's configuration. The configuration at this point is raw and the interpolations have not been processed. The value of any key is not guaranteed and is just meant to be a quick structural check.
2. [Diff](https://www.terraform.io/internals/lifecycle#diff) is called with the current state and the configuration. The resource provider inspects this and returns a diff, outlining all the changes that need to occur to the resource. The diff includes details such as whether or not the resource is being destroyed, what attribute necessitates the destroy, old values and new values, whether a value is computed, etc. It is up to the resource provider to have this knowledge.
3. [Apply](https://www.terraform.io/internals/lifecycle#apply) is called with the current state and the diff. Apply does not have access to the configuration. This is a safety mechanism that limits the possibility that a provider changes a diff on the fly. Apply must apply a diff as prescribed and do nothing else to remain true to the Terraform execution plan. Apply returns the new state of the resource (or nil if the resource was destroyed).
4. If a resource was just created and did not exist before, and the apply succeeded without error, then the provisioners are executed in sequence. If any provisioner errors, the resource is marked as tainted, so that it will be destroyed on the next apply.

Steps:

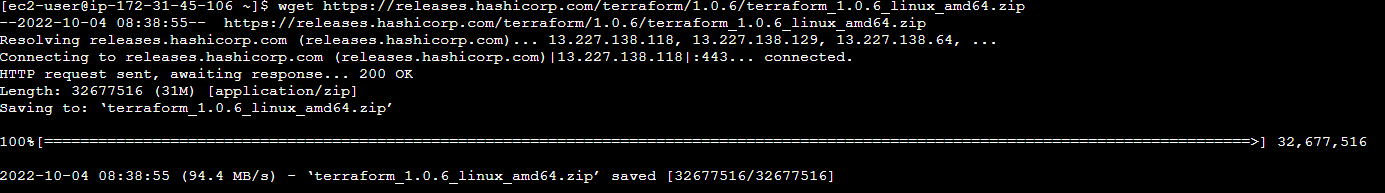
1. Create an instance



1. Connect to terminal



1. Install Terraform



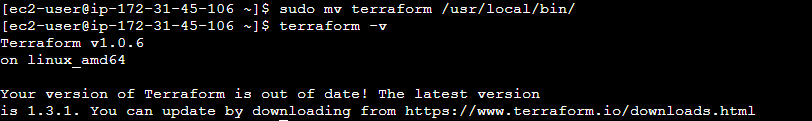
1. Check for the zip folder



1. Unzip the zip folder



1. Check for installation and version



**CONCLUSION:**

From this experiment we have studied the terraform lifecycle and core concepts / terminologies of terraform and installed terraform on a Linux Machine.

**LOs and POs achieved:**

With this experiment we have achieved the Lab Outcomes One and Three (LO1 and LO3). The Program Outcomes (PO’s) achieved from this experiment are PO1, PO2, PO3, PO4, PO5, PO9, PO10 and PO12.