linode

Declarative Cloud Infrastructure Management with Terraform

How to Provision Cloud Infrastructure Using a **Single Workflow**

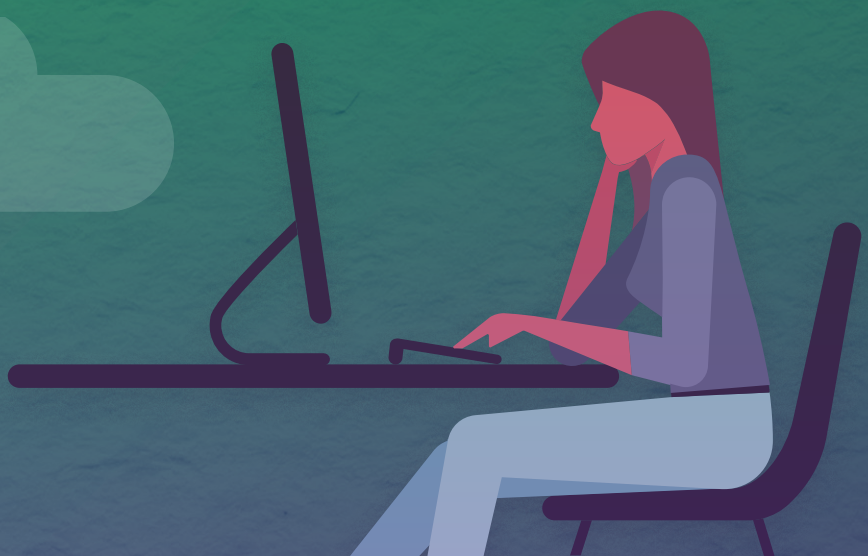


Table of Contents

Terraform is one of the most popular cloud infrastructure provisioning tools that supports Infrastructure as Code principles while working with constantly scaling infrastructure and/or multiple cloud providers. After reading this guide, you'll understand the underlying concepts of Terraform, declarative cloud infrastructure management, and key basics of Terraform's language, the Hashicorp Configuration Language.

What is Infrastructure as Code?

Infrastructure as Code (IaC) is a development and operations methodology that allows server deployments and software configuration to be represented as code. This methodology reduces chances for human error, makes complex systems more manageable, eases collaboration on systems engineering projects, and offers a number of other benefits.

- Create reproducible environments that can scale up or down rapidly to help diagnose problems and improve performance, without affecting a production environment
- Manage thousands of servers and services in a clean interface to run updates and implement changes
- Collaborate on infrastructure configuration via a Git repository or other tool you're already using, instead of everyone needing access to a cloud infrastructure console
- Automate checks and balances to help you test and track changes to your infrastructure

Terraform is a well-known and widely used tool to manage infrastructure as code. As a tool that's extremely simple to use and requires minimal code experience, Terraform supports developers and organizations in streamlining cloud infrastructure management without changing their cloud providers or resources – only slight shifts in their infrastructure management workflow.

INFRASTRUCTURE AS CODE VERSUS CONFIGURATION MANAGEMENT

Though these two concepts can often simultaneously apply to your workloads, it's important to understand the differences between configuration management and infrastructure as code, and in particular, where Terraform sits in that relationship.

Infrastructure as Code

Using source code to treat cloud and IT infrastructure like software in order to configure and deploy, and rapidly replicate and destroy, infrastructure resources.

Configuration Management

Providing consistency of systems and software over time. Configuration management happens repeatedly to keep server software configurations up to date and operational for a product or service to maintain its performance and reliability.

Operations performed through an IaC tool like Terraform fundamentally affect the existence of resources or infrastructure, not perform ongoing maintenance or optimization, hence Terraform is not considered a configuration management tool.

IAC AND LINODE

With more organizations, individuals, and services relying on cloud infrastructure than ever before, Linode aims to make scaling and managing resources as painless as possible. However, we understand there are limitations to what we can do that is native to the Linode platform, or our users' preferences for existing third party or open source tools. That's why we offer integrations with IaC tools, to make the alternative cloud more accessible for workloads of all sizes.

Why Terraform?

Terraform is an IaC tool that focuses on creating, modifying, and destroying servers and cloud resources. Not to be confused with configuration management software, Terraform does not manage the software on those servers. Any public cloud with an API can be a provider within the Terraform lifecycle.

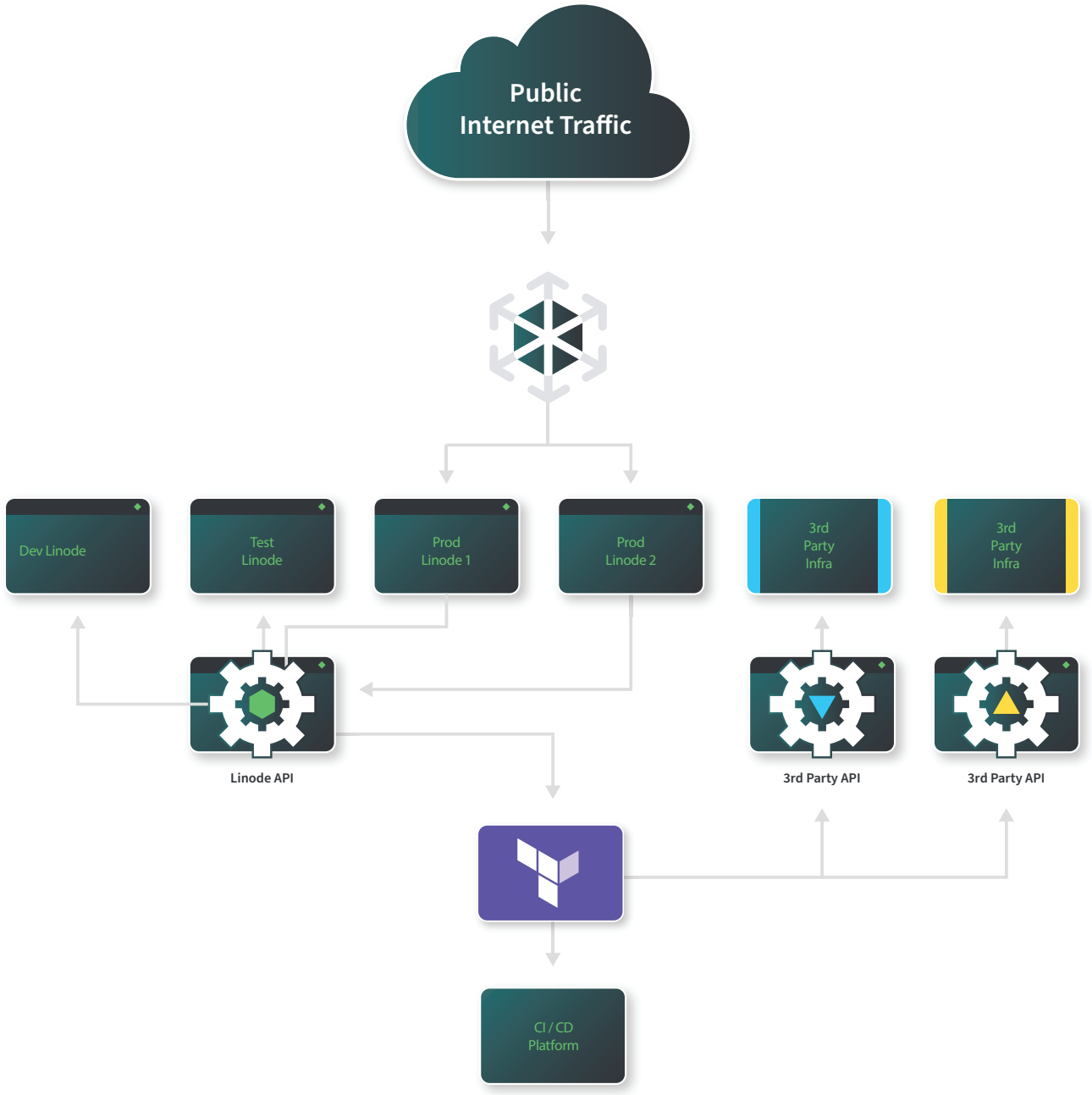
Terraform's representation of your resources in configuration files is referred to as Infrastructure as Code (IaC). The benefits of this methodology and of using Terraform include:

- **Version control of your infrastructure**
  Because your resources are declared in code, you can track changes to that code over time in version control systems like Git.
- **Minimization of human error and provisioning inconsistencies**
  Terraform's analysis of your configuration files will produce the same results every time it creates your declared resources. In addition, instructing Terraform to repeatedly apply the same configuration will not result in extra resource creation, as it tracks changes made over time.
- **Better collaboration among team members**
  Terraform's backend allows multiple team members to safely work on the same configuration simultaneously and securely.
- **Allows cloud infrastructure to be more self-service across providers**
  The need to log in to various management dashboards and understand how to provision resources is eliminated when you can rely on performing these basic operations through Terraform.
- **Consistency for management and compliance needs**
  With Terraform, you provide a blueprint of your cloud infrastructure's desired state in an accurate and concise way that describes what the end result should look like.

Since Terraform uses the same language and framework for a variety of cloud providers, the need for developers and managers with provider-specific management skills or certifications decreases. When this essential management can be streamlined through Terraform, it reduces complexity and the need for more specific cloud computing skills that can be difficult to learn as a developer, and hire for as a manager.

To demonstrate Terraform's simplicity, here's a typical Terraform setup for a web application running on Linode.

FIGURE 1



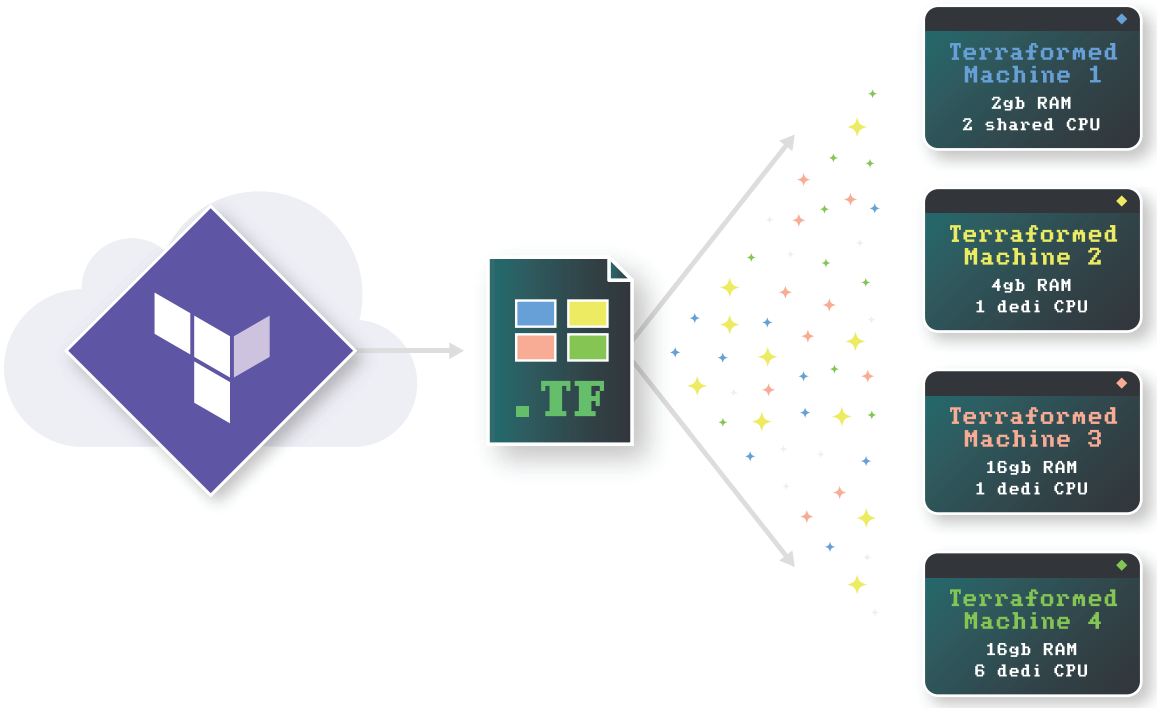
What is Terraform?

Terraform simplifies cloud resource and service management with a consistent CLI and transforms cloud API calls into declarative configuration files. Terraform is a significant resource when your cloud infrastructure hits critical mass. Often, anything more than a handful of servers is difficult to manage for many teams or organizations, especially when working with multiple cloud providers.

Built by HashiCorp, Terraform is a declarative tool written in Go that employs configuration files in a custom language for infrastructure management that is both human and machine-readable. Being declarative means you write code in Terraform to describe what your infrastructure should look like at the end state, or provide a blueprint of what Terraform should execute for you.

Instead of running commands line by line to create infinite resources based on code you write, you're creating your architecture within Terraform and telling it to execute the actions needed to match what you mapped out in your Terraform file, or .tf. The .tf is accessible to your cloud providers via API endpoints.

FIGURE 2



Terraform configuration files are written in the HashiCorp Configuration Language (HCL), which breaks down critical commands into three elements.

1) **Block:** Container for content, usually used to define the configuration for a resource or service, like an instance from a cloud provider.

2) **Argument:** Syntax construct that assigns a value to a name, appearing in a block.

3) **Expression:** Syntax that represents value, and used as values within arguments.

Terraform offers plugins, called "Providers," to interface with different cloud hosts. Terraform requires a supported API token from your cloud provider(s), that allows you to use HCL to modify your infrastructure architecture. Terraform can also integrate with SaaS platforms like GitHub and Fastly. Terraform takes elements of critical infrastructure that are connected and makes it possible to provision and manage these different tools in one ecosystem, using one language.

TERRAFORM INSTALLATION OVERVIEW

To start using Terraform to manage new or existing cloud infrastructure, the bulk of the work is in initial setup and making sure your resource declarations and variables are set up correctly. When you become familiar with your Terraform provider's plugin and how to both define and run commands on your resources, it becomes significantly easier to scale your cloud infrastructure and to evaluate and manage many resources with just a few lines of code.

The initial Terraform installation does not contain plugins for cloud providers out of the box. Like most other cloud providers, Linode is available as a provider plugin within Terraform. Providers are added and configured after completing the initial installation.

Get started by downloading Terraform for your operating system from the Terraform website.

Once you have Terraform installed on your development machine, run the following command to make sure it's working correctly:

```
terraform –v
```

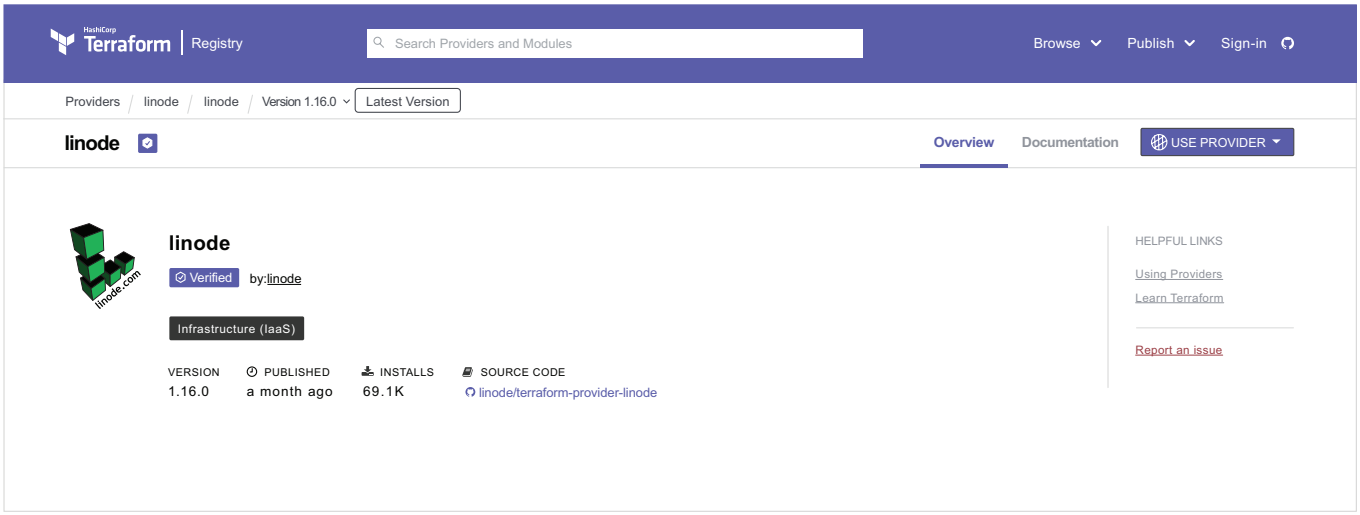
The output for a successful Terraform installation will look something like this, which represents the Terraform version number you downloaded and installed successfully.

```
Terraform v0.15.3
```

From there, you can finish following Linode's Beginner's Guide to Terraform to complete your setup and get ready to start working with cloud infrastructure resources on your preferred provider(s).

To download the plugin for your preferred provider, head to the Terraform Provider Registry site and search for your provider. Or, for example, head directly to Linode's provider page.

FIGURE 3



Each provider has a code snippet that you can paste into a Terraform file. With the code snippet added, you can run the `terraform init` command from a terminal to install the plugin.

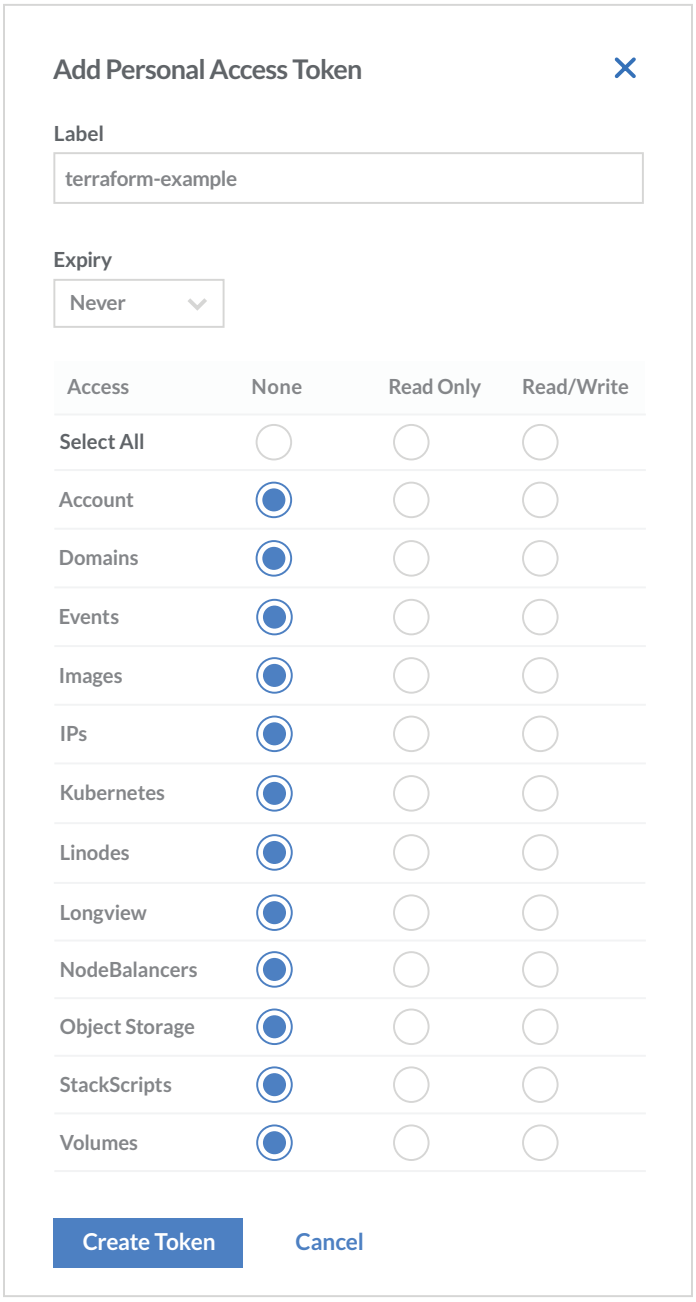
FIGURE 4

```
terraform {
  required_providers {
    linode = {
      source = "linode/linode"
      version = "version number"
    }
  }
}

provider "linode" {
  # Configuration options
}
```

Create a personal access token from the Linode Cloud Manager and select which Linode products and services the token should be able to read, or both read and write, and you're ready to start creating Linode resources using Terraform.

FIGURE 5



Using Terraform does require a certain level of familiarity with the Linode API, in addition to learning HCL. Luckily, the most critical functionality of Terraform works with just a handful of intuitive commands.

Getting Started with HashiCorp Configuration Language (HCL)

HashiCorp Configuration Language is the custom language created by HashiCorp to use within all of their products. HCL is designed to be readable and machine friendly, which also makes it much easier to take full advantage of Terraform's declarative benefits in terms of defining and modifying your cloud resources. It is possible to use JSON for Terraform deployments, but HCL is strongly recommended, and all examples provided in this ebook use HCL.

Here are the essential components of HCL and what will make up your declarative cloud resource configuration.

- **Resources:** Infrastructure resources that can be managed by Terraform including compute instances, DNS records, and network resources.
- **Dependencies:** Terraform resources that depend on another resource to exist. When one resource depends on another, it will be created after the resource it depends on, even if it is listed before the other resource in your configuration file.
- **Input Variables:** A symbolic name associated with a value (also referred to as just Variables).
- **Resource Declarations:** Correspond with the components of your Linode infrastructure: Linode instances, Block Storage volumes, etc.
- **Resource Provisioners:** Scripts and commands in your local development environment or on Terraform-managed servers that are performed when you apply your Terraform configuration.

Refer to Terraform's Glossary to get to know the other various terms you will encounter while using Terraform, in addition to some cloud computing terms you may not be familiar with.

Here's an example of a Linode resource

FIGURE 6

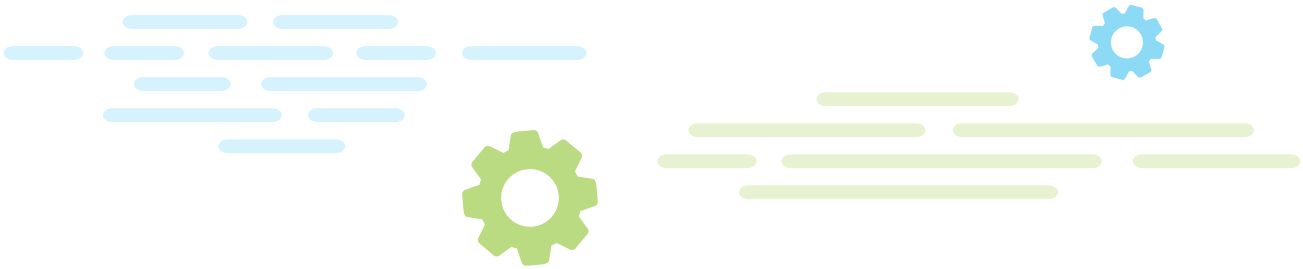
```
provider "linode" {
    token = var.token
}

resource "linode_instance" "example_instance" {
    label = "example_instance_label"
    image = "linode/ubuntu20.04"
    region = var.region
    type = "g6-standard-1"
    authorized_keys = [var.ssh_key]
    root_pass = var.root_pass
}

variable "token" {}
variable "root_pass" {}
variable "ssh_key" {}
variable "region" {
  default = "us-southeast"
}
```

An example of a dependency would be a DNS record for this example instance. Though it will be written as a separate resource, its existence in your overall infrastructure depends on the example Linode in order to work.

Some use cases will require the use of provisioners. However, they often add complexity and uncertainty to Terraform usage and are considered more advanced. Most provisioners are declared inside of a resource declaration. When multiple provisioners are declared inside a resource, they are executed in the order they are listed. Check out the full Terraform documentation for a full list of provisioners and how to use them.



Terraform Commands & Using Terraform to Provision Linode Environments

The Linode Provider can be used to create Linode instances, Images, domain records, Block Storage Volumes, Object Storage Buckets, StackScripts, and other resources. Terraform's official Linode provider documentation details each resource that can be managed.

Interacting with Terraform takes place via its command line interface. The Terraform CLI allows you to access and control your cloud infrastructure resources using Workspaces. Using multiple workspaces is recommended to manage separate testing and production infrastructures.

Key Commands:

- **Init -** Initializes Terraform, checks and downloads available provider plugins, and downloads the plugins for the providers you selected and provided API tokens for in your .tf file.
- **Plan -** Shows just a preview of the changes that will take place, including a list of resources that will be created or destroyed. (This step is optional, but strongly recommended, especially for production environments.)
- **Apply -** Applies the configuration changes and creates / modifies resources in your provider account(s).
- **State -** Initializes the ability to examine the current state of your .tf file. Use the state subcommand list to see all resources in your project. This is a useful way to get a wider snapshot of all the resources in your project versus scrolling through your .tf file where your resources are defined.
- **Show -** See a more detailed output of a resource's information after running the state command, including information that can only otherwise be found in the provider's console.

> **Tip:** This is the simplest way to view an individual resource's ID, region, and IP address, among other information, while staying in your Terraform environment.

- **Destroy -** By default, this command destroys all resources present in that .tf file, without removing the code from the file, in case you want to immediately recreate all those resources or be able to replicate that setup at another time.

> **Note:** To remove one specific resource from your provider, you can either delete or comment out that resource's information in the .tf file, and that individual resource will be destroyed the next time you run the apply command.

This is just the list of commands to help you get started with the basics of Terraform. Check out Terraform's official documentation for more.

Linode Kubernetes Engine (LKE) and Terraform

LKE is a fully-managed container orchestration engine for deploying and managing containerized applications and workloads. LKE combines Linode's ease of use interface and simple pricing with the infrastructure efficiency of Kubernetes. When you deploy a LKE cluster, you receive a Kubernetes Master at no additional cost; you only pay for the Linodes (worker nodes), NodeBalancers (load balancers), and Block Storage Volumes used to support your cluster. Your LKE cluster's master node runs the Kubernetes control plane processes – including the API, scheduler, and resource controllers.

Ultimately, the goal is to streamline Kubernetes management by creating reusable Terraform configuration files to define your Kubernetes cluster's resources.

Our friends at LearnK8s wrote a step-by-step breakdown on how to Provision Kubernetes Clusters on Linode with Terraform.

If you're new to containerization and Kubernetes, check out our Understanding Kubernetes eBook.

Importing Existing Infrastructure to Terraform

You don't need to start a brand new project to start working with Terraform. Existing Linode resources can be imported and brought under Terraform management using the terraform import command, which imports your existing resources into Terraform's state. Currently, Linode resources can only be imported to Terraform individually, and the import command does not generate a Terraform resource configuration.

State is Terraform's stored JSON mapping of your current Linode resources to their configurations. You can access and use the information provided by the state to manually create a corresponding resource configuration file and manage your existing Linode infrastructure with Terraform.

After following steps to download Terraform for your OS, install the Linode Terraform provider, and enter your Linode API access token. Here's a brief summary of the steps you need to take to successfully import and start managing your resource with Terraform:

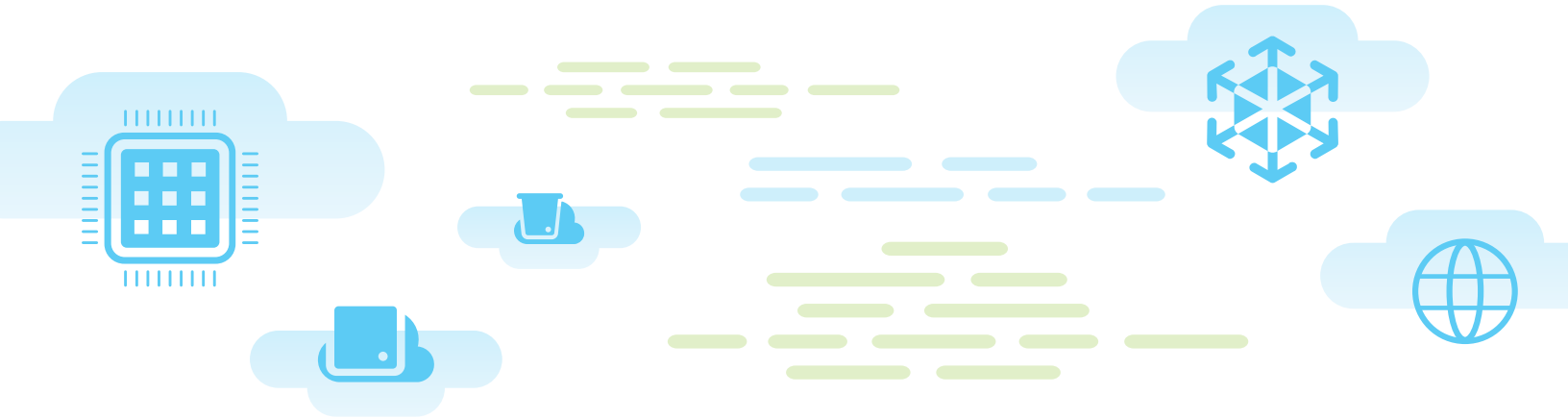
- Retrieve the resource's ID
- Create an empty resource configuration file
- Run the import command to import your resource to Terraform
- Fill in your resource's configuration data

> **Note:** When importing your infrastructure to Terraform, failure to accurately provide your Linode service's ID information can result in the unwanted alteration or destruction of the service. Utilizing multiple Terraform Workspaces is advisable to manage separate testing and production infrastructures.

You can import the following Linode infrastructure resources to Terraform:

- Linodes (Shared, Dedicated CPU, High Memory)
- Domains and Domain Records
- Block Storage Volumes
- Object Storage Buckets
- NodeBalancers

Step-by-step instructions to importing each type of Linode infrastructure is available on our Import Existing Infrastructure to Terraform guide.



Combining IaC with Configuration Management

Terraform is an IaC tool that focuses on creating, modifying, and destroying servers and cloud resources. Not to be confused with configuration management software, Terraform does not manage the software on those servers. Any public Cloud with an API can be a provider within the Terraform lifecycle.

Terraform accomplishes the essential tasks of creating, destroying, and modifying cloud infrastructure resources, and centralizing multicloud deployments by working with multiple cloud providers in one environment. But there is a critical difference between the functions performed by IaC and configuration management. Here's a brief overview of tools you could use to perform different tasks to automate or optimize your cloud infrastructure.

Ansible

Ansible is a configuration management tool. It excels in provisioning software and devices, and deploying applications that run on that infrastructure, including continuous integration (CI) pipelines. Ansible can integrate with cloud networks, runs on most Linux distributions, and provides support for Linode. Ansible operates on a particular instance in isolation from the network and ensures it is functioning normally. Ansible is designed to be minimal, consistent, secure, reliable, and easy to learn. Ansible is straightforward to install, and like Terraform, does not require any specific programming skills to start using.

Get Started with the Linode Module on Ansible Galaxy.

Salt

Salt (also referred to as SaltStack) is a Python-based configuration management and orchestration system. Salt uses a master/client model in which a dedicated Salt master server manages one or more Salt minion servers. Two of Salt's primary jobs are remotely executing commands across a set of minions and applying Salt states to a set of minions (referred to generally as configuration management)

Learn more about Salt with Linode's Beginner's Guide to Salt guide.

Pulumi

Pulumi is designed to support organizations in managing their infrastructure using an existing skill set and/or tools. This allows better integration with legacy systems that interact with the network through the Pulumi Software Development Kit (SDK). This approach fits nicely into DevOps culture, because development teams can specify infrastructure using well-known imperative programming languages. It is not necessary to learn any new languages.

With Pulumi, teams can deploy to any cloud, integrate with a CI/CD system, and review changes before making them. Pulumi provides many advanced features such as audit capabilities, built-in encryption services, and integration with identity providers. It can take checkpoints or snapshots, and store sensitive configuration items, such as passwords, as secrets.

Learn more about how to use Terraform versus Pulumi.

Our Take

Terraform is an important tool for developers and IT professionals to maximize efficiencies of their cloud infrastructure and simplify multicloud strategies that are now essential for the vast majority of businesses operating in the cloud. Ensuring Linode is easy to use on Terraform aligns with our core mission to accelerate innovation by making cloud computing simple, affordable, and accessible to all. We encourage customers to stay cloud-agnostic while getting the products and services they need by encouraging the use of open source tools and third party resources to optimize performance and budget. Linode leads the industry in price-performance, and also believes that it's essential to provide customers with transparent billing to avoid unexpected costs.

Reducing complexity is key to our mission statement. But as businesses and workloads scale and become more sophisticated, cloud infrastructure becomes more complex, with some organizations requiring thousands of instances and orchestration across multicloud deployments. Terraform significantly reduces complexity by combining operations across providers into one platform, and allowing developers to easily and rapidly scale infrastructure in a system that is extremely readable and simple as compared to other code languages or infrastructure management tools.

Understanding the principles of IaC, and specifically Terraform, is becoming more industry-standard for DevOps engineers and developers who are the daily managers of cloud infrastructure. And for most companies already using the public cloud, that means working with multiple providers. Ninety-three percent of companies utilizing public cloud resources are using multiple cloud providers to run their business (Source: IDC). IaC and its connection to multicloud is an inevitable skill as technology continues to advance and organizations look to adapt cloud resources to maximize their infrastructure budgets.

The role of cloud infrastructure in global and local economies is constantly changing and expanding. Finding tools that help make sense of the chaos can only help developers and other infrastructure managers, in addition to helping organizations and clients make better decisions about their cloud resources.

NEXT STEPS

Using Terraform and IaC tools, even at a small scale, is a way to futureproof your workloads and cloud computing strategy.

While using Linode and Terraform, developers can simplify cloud infrastructure provisioning and billing for resources. All Linode plans include generous transfer, automated Advanced DDoS Protection, and

the ability to configure Cloud Firewalls and VLAN via Linode Cloud Manager, API, or CLI at no extra cost. Now that you understand the basics, try downloading Terraform and provisioning or importing Linode resources.

- Set up Terraform and create new resources while following our Using Terraform to Provision Linode Environments guide and start using Linode with a $100 account credit.
- Follow step-by-step instructions to Import Existing Linode Infrastructure to Terraform, or watch a video tutorial on the Linode YouTube channel.
- Become a certified Terraform Associate through Hashicorp's verified certification program.

In addition to cloud providers, Terraform Providers include other tools you might already be using, including GitHub, as well as community contributions. Explore the full Terraform Provider Registry.

For more detailed instructions and a walkthrough of all the steps needed to get started with Terraform, check out all of our Terraform resources.

About Linode

Our mission is to accelerate innovation by making cloud computing simple, affordable, and accessible to all.

Founded in 2003, Linode helped pioneer the cloud computing industry and is today the largest independent open cloud provider in the world. Headquartered in Philadelphia's Old City, the company empowers more than a million developers, startups, and businesses across its global network of 11 data centers.

The World's Largest Independent Open Cloud

linode.com | Support: 855-4-LINODE | Sales: 844-869-6072
249 Arch St., Philadelphia, PA 19106 Philadelphia, PA 19106