

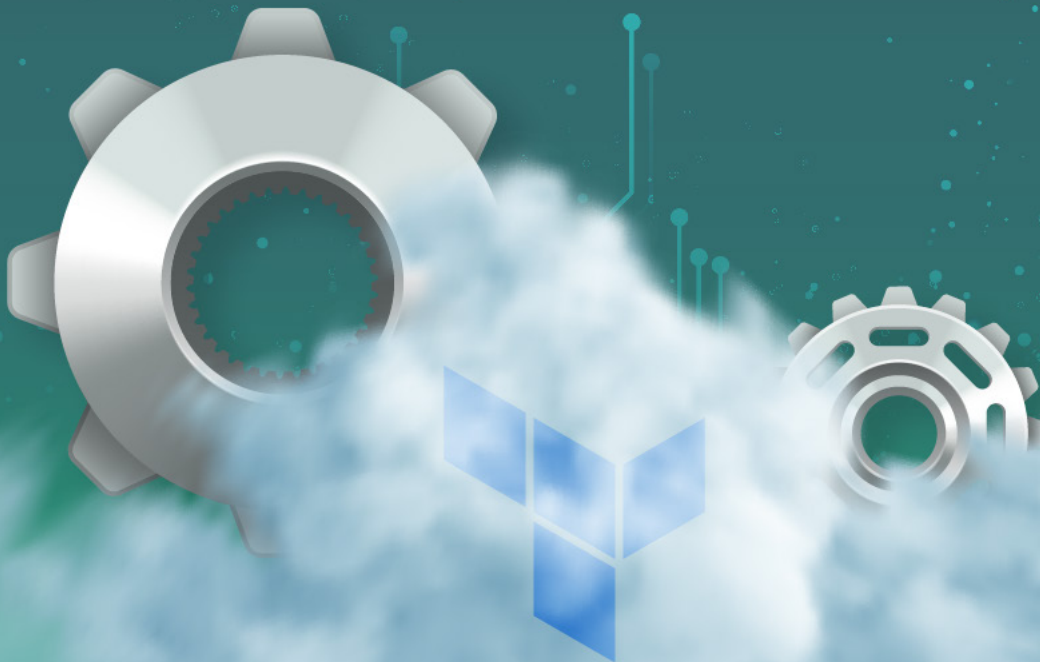


linode

Akamai Cloud Computing

Declarative Cloud Infrastructure Management with Terraform

How to Provision Cloud Infrastructure
with a Single Workflow



UPDATED FOR 2022



linode

Akamai Cloud Computing

EBOOK

TERRAFORM

Table of Contents

What is Infrastructure as Code?	04
Developing In a Cloud Native & Multicloud Landscape	06
What is Terraform?	09
Working with Terraform.	11
Getting Started with the HashiCorp Configuration Language (HCL).	15
Using Terraform to Provision Linode Environments	17
Linode Kubernetes Engine and Terraform	18
Importing Existing Infrastructure to Terraform	19
Terraform Automation Options.	20
Comparing Terraform and Ansible	22
Other Infrastructure as Code Tools and Integrations	25
About	27

Introduction

Terraform is one of the most popular cloud infrastructure provisioning tools that supports Infrastructure as Code (IaC) principles while working with constantly scaling infrastructure and/or multiple cloud providers. With a growing ecosystem of providers and plugins to integrate Terraform with DevOps workflows, Terraform is continually increasing in market share and usability.

In this ebook, you will learn:

- what Terraform is and how it works with your cloud provider(s);
- benefits of using IaC tools, especially Terraform;
- initial steps to install Terraform and use the Linode Terraform Provider;
- basics of the HashiCorp Configuration Language (HCL);
- when to use Terraform instead of (and with) Ansible;
- and more.

Linode cloud computing services from Akamai supports development workflows by strongly focusing on integrating our cloud platform with IaC tools you already use, and providing accessible cloud infrastructure with straightforward pricing.

Using Linode and Terraform together is the best way to follow along. Sign up for a new Linode account today and you'll get [\\$100 in free credit](#).

What is Infrastructure as Code?

Infrastructure as Code (IaC) is a development and operations methodology that allows resource deployment and configuration to be represented as code. This methodology is conducive to automation, makes complex environments more manageable, and offers a number of additional benefits

- 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.
- Automated 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 and Configuration Management

Although these two concepts can often simultaneously apply to your workloads, it's important to understand the differences between configuration management, provisioning 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 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.

Want to learn more IaC tools in addition to Terraform?

Check out our Try IaC educational series, taught by Justin Mitchel of Coding for Entrepreneurs. Watch on-demand video tutorials and download an in-depth ebook.

DOWNLOAD TRY IAC EBOOK

START VIDEO SERIES



Developing In a Cloud Native and Multicloud Landscape

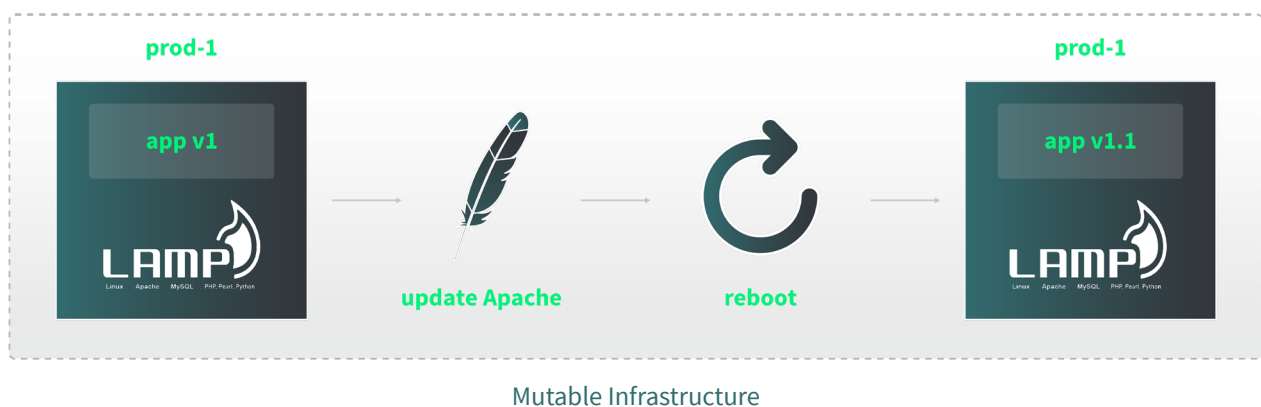
A key component and requirement of cloud native or multicloud applications is portability. Infrastructure as Code, at its core, is portable and universally compatible with all major cloud providers. Cloud native development, even with a single provider, is built on granular separation of each component and requires flexible and fast provisioning to keep up with changing resource utilization. IaC tools are a natural fit to build and sustain the environments your application needs while allowing DevOps teams to do more with less.

Cloud native: Cloud native refers to application architecture that is designed to run on one or more clouds to build and scale applications from the initial development phase, as opposed to migrating from on-prem or local infrastructure to the cloud. Cloud native developments utilize modern application development techniques including containerization, microservices, and declarative APIs.

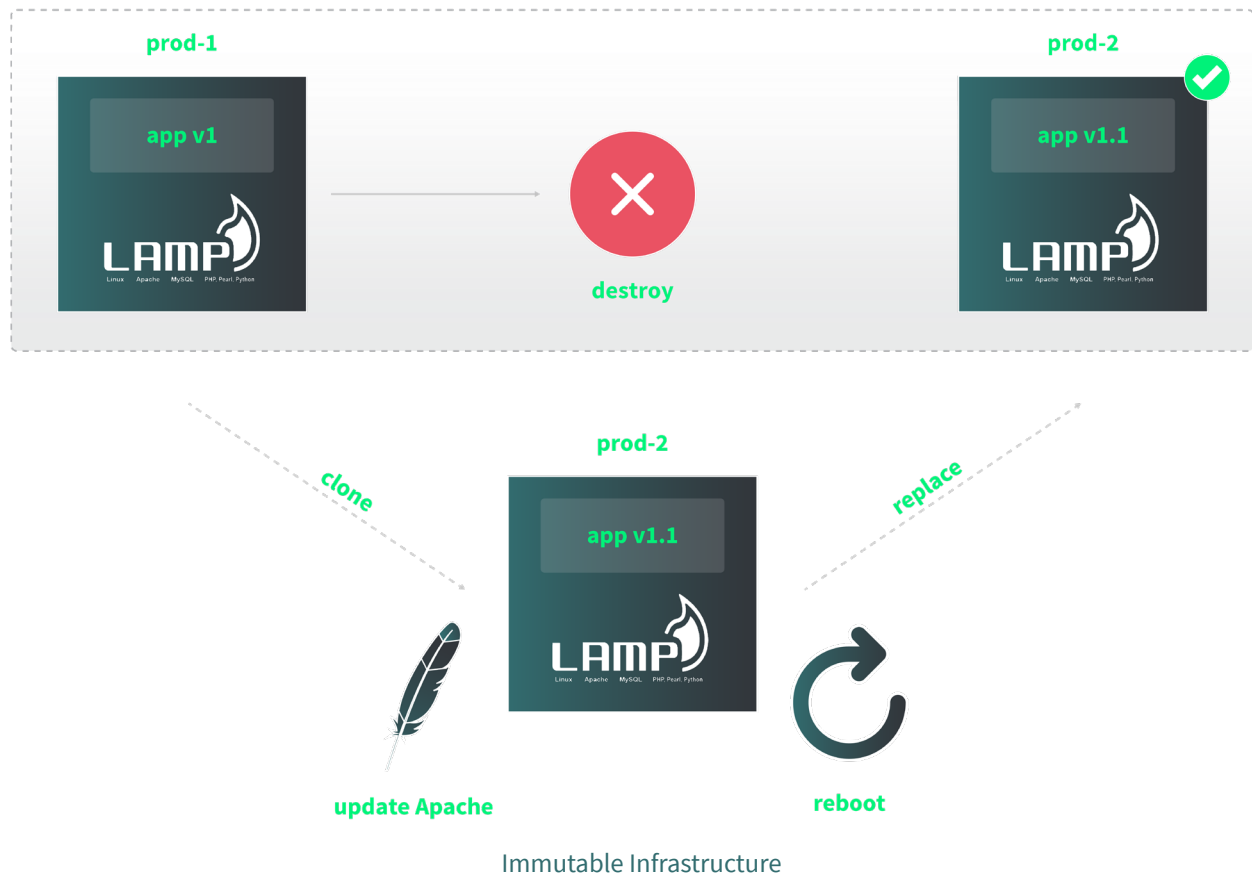
Another key factor of cloud native deployments is immutable infrastructure, or replacing pieces of infrastructure as part of significant changes or upgrades instead of making changes on your existing infrastructure. Immutable infrastructure allows you to deploy and test new versions of each component with your application on fresh images instead of updating an existing instance. This significantly reduces the chance of an update causing problems in a production environment. The following example quickly illustrates the differences between mutable and immutable infrastructure while updating a version of Apache on a web server.

Your production application is running on a virtual machine, currently running Apache that is a few versions behind, and you want to upgrade to the latest version.

With mutable infrastructure, you perform the upgrades on a separate dev environment that's running the same stack, and the upgrade goes smoothly. So you perform the same configuration update in place on your primary production instance. The update may run according to plan, or you could encounter an issue that affects your production users.



With immutable infrastructure, you set up an entirely new instance that's running the new version of Apache, and have time to perform tests and make sure everything is working as it should. When it's time to go live, simply update your DNS or follow your usual procedure to get a new server into production. When that's complete, destroy your old production server.



There are other considerations to make while designing immutable infrastructure, such as storing all of your data via a block storage volume that can be easily attached/detached, but this outlines the process of **replacing** infrastructure elements instead of upgrading in place. This protects your application from potential failure, and allows you to frequently make iterative changes and upgrades, instead of needing to roll out major releases that require more time to build, test, and deploy without disrupting the application's services.

As the Cloud Native Computing Foundation (CNCF) succinctly states in [their charter](#):

“These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”

With the focus on containerization and distributing application layers for increased performance and resiliency, here are quick definitions on the most common types of infrastructure deployments:

- **Multicloud:** Using two or more public clouds to host and scale your application. This can be as simple as using one cloud provider for compute instances and another for scalable storage like Object Storage.
- **Hybrid Cloud:** Using a combination of on-prem infrastructure and public cloud providers. For example, an on-prem infrastructure resource can access and interact with cloud-based infrastructure.
- **On-Premise:** Often shortened to “on-prem”, this refers to IT infrastructure that is located within an organization’s controlled IT boundary that is not the cloud.

With more cloud providers and services to choose from, multicloud is rapidly becoming the standard. Knowing tools like Terraform become more important to provide increased control and less overhead when deploying infrastructure. With Terraform, there is no need to build new environments from scratch. Everything can be templated while still providing version control.

Using cloud-agnostic tools like Terraform simplifies infrastructure management across providers, and allows developers to provision infrastructure from multiple providers without needing to write different scripts or maintain a separate set of images.

What is 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 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. Traditionally, anything more than a handful of servers requires an investment in a system management tool and can be difficult to maintain for smaller teams or organizations, especially when working with multiple cloud providers.

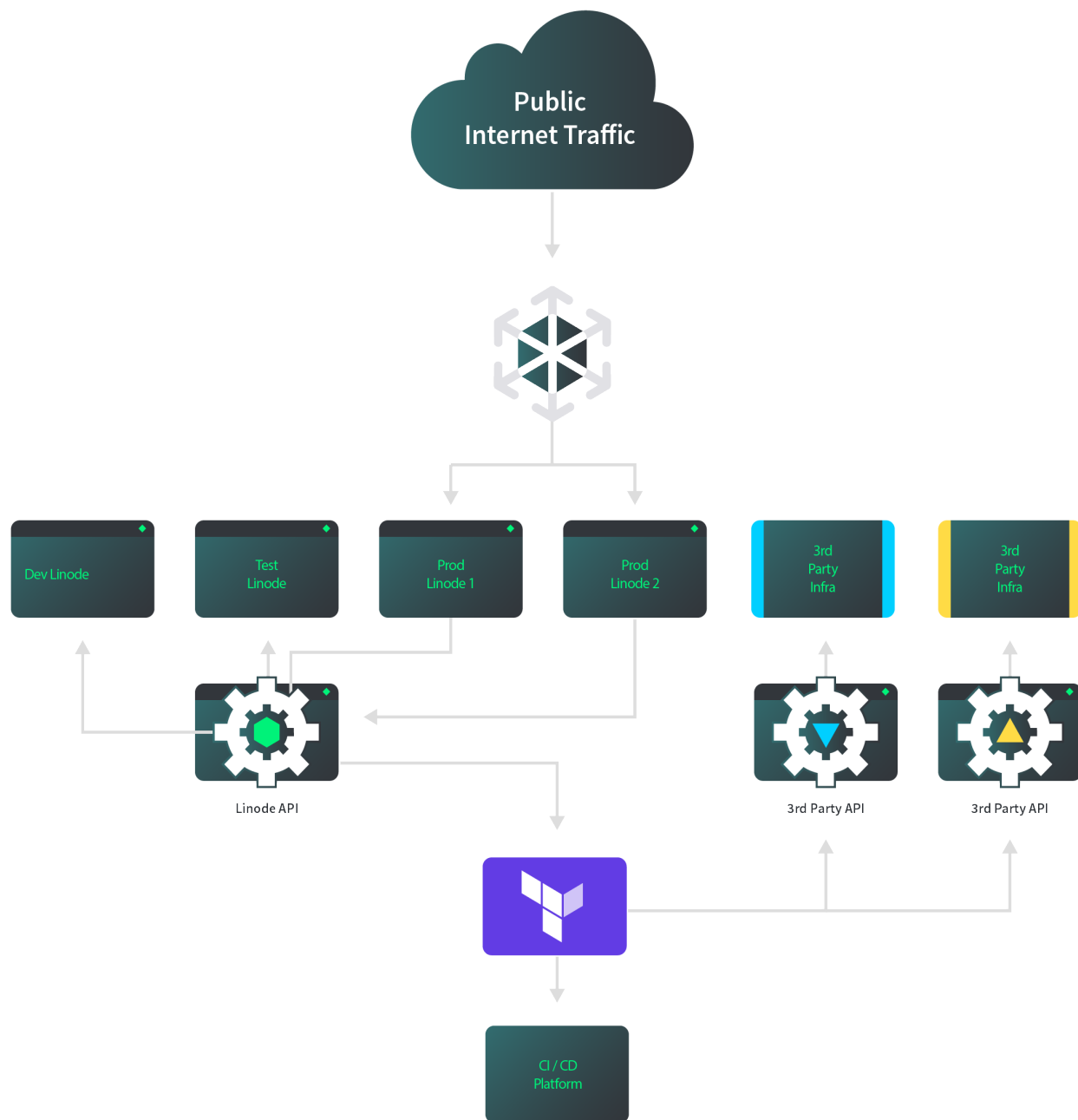
Built by HashiCorp, Terraform is a declarative tool written in Go that employs configuration files in a custom language for infrastructure management that is both human and machine-readable. Being declarative means you write code in Terraform to describe what your infrastructure should look like at the end state, or provide a blueprint of what Terraform should execute for you. Unlike an imperative tool, which would run each line of script or code to reach an outcome, a declarative tool is only concerned with the end state. Think of imperative as caring about “how” something happens while declarative only cares about “what” happens.

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.
- **Increase in 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.

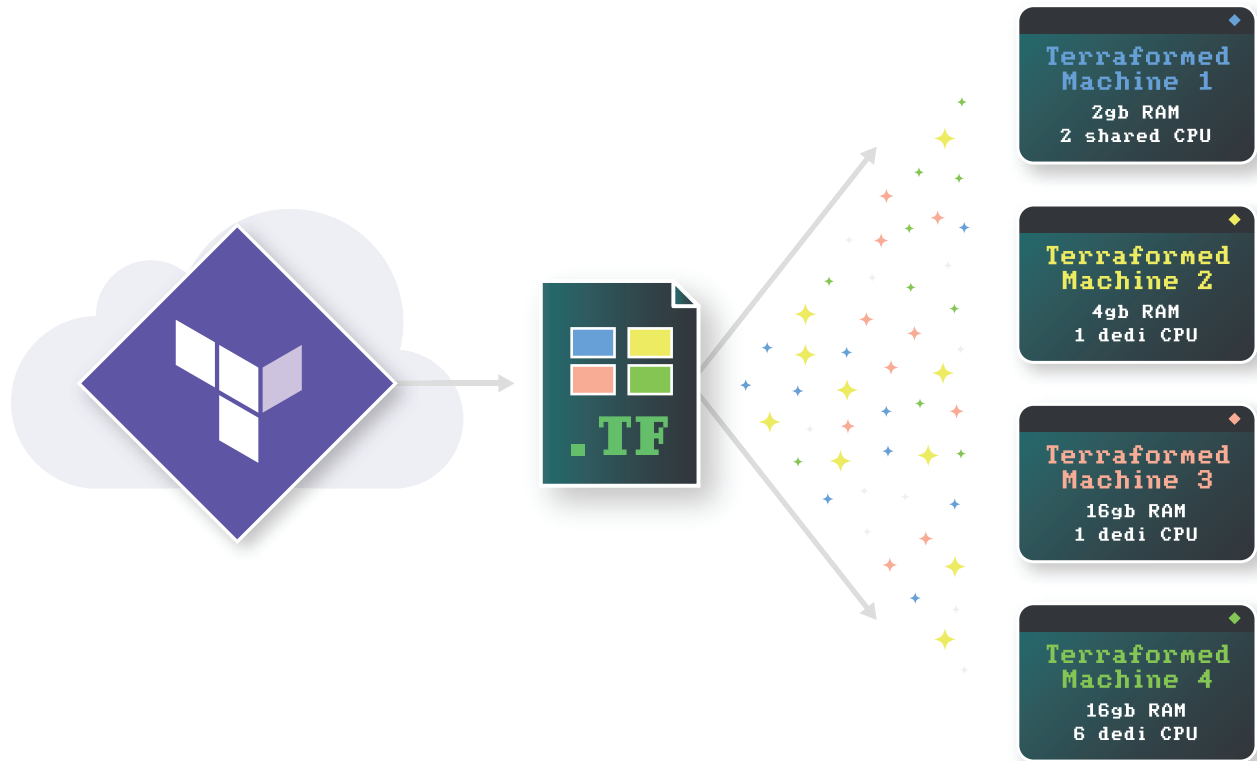
Since Terraform uses the same language and framework regardless of cloud provider, the need for developers and administrators with provider-specific management skills or certifications decreases. When this essential management can be streamlined through Terraform, it reduces complexity and the need for more specific skills that can take additional time to achieve proficiency in.

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



Working with Terraform

Instead of running commands line by line to create infinite resources based on code you write, you create your architecture within Terraform and tell 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.



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

- **Block:** Container for content, usually used to define the configuration for a resource or service, like an instance from a cloud provider.
- **Argument:** Syntax construct that assigns a value to a name, appearing in a block.
- **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 Twilio. 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 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](#).

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

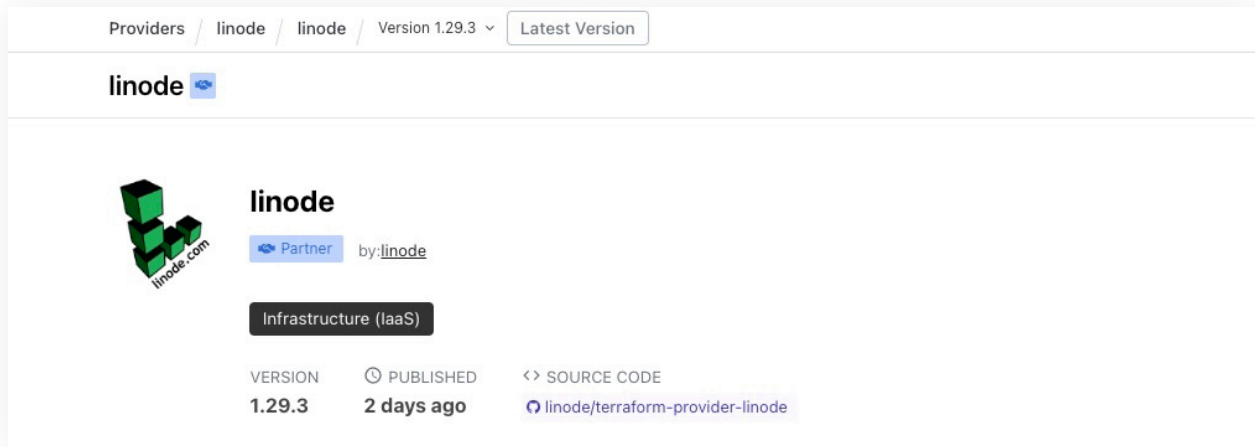
```
terraform -v
```

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

```
Terraform v0.01.01
```

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.



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.

```
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 the 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.

Add Personal Access Token

Label

Expiry

Access	None	Read Only	Read/Write
Select All	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Account	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Databases	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Domains	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Events	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Firewalls	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Kubernetes	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Images	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
IPs	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Linodes	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Longview	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
NodeBalancers	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Object Storage	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
StackScripts	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Volumes	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Cancel
Create Token

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

Getting Started with the 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

```
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 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.



Pro-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. [Read Terraform's official documentation for more.](#)

Linode Kubernetes Engine and Terraform

Linode Kubernetes Engine (LKE) is a fully-managed container orchestration engine for deploying and managing containerized applications and workloads. LKE combines our user-friendly interface and simple pricing with the infrastructure efficiency of Kubernetes. When you deploy a LKE cluster, you receive a Kubernetes control plane 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 primary node runs the Kubernetes control plane processes, including the API, scheduler, and resource controllers.

There's a reason why Kubernetes and Terraform work so well together: Kubernetes itself is a declarative system. When you deploy a Kubernetes cluster and modify your configuration files, you are describing the desired state of your application and how your cluster should respond, but not the sequence of commands or events to reach the desired state.

Typically, Kubernetes clusters are managed using `kubectl` or another CLI-based tool. This is great for the day-to-day management of your clusters, but as you scale using Kubernetes, adding Terraform to your workflow provides benefits that are not found in the other tools. HashiCorp outlines these benefits:

- **Unified Workflow:** If you are already provisioning Kubernetes clusters with Terraform, use the same configuration language to deploy your applications into your cluster.
- **Full Lifecycle Management:** Terraform doesn't only create resources, it updates, and deletes tracked resources without requiring you to inspect the API to identify those resources.
- **Graph of Relationships:** Terraform understands dependency relationships between resources. For example, if a Persistent Volume Claim claims space from a particular Persistent Volume, Terraform won't attempt to create the claim if it fails to create the volume.

Ultimately, the goal is to streamline Kubernetes management by creating reusable Terraform configuration files to define your Kubernetes cluster's resources. Get step by step instructions on how to deploy a cluster, access your cluster's kubeconfig, and build applications with our [guide to using LKE and Terraform](#).

[Kubernetes also has a Terraform provider](#) in the Terraform registry to deploy unmanaged Kubernetes clusters using Terraform. You can deploy managed Kubernetes clusters on Linode using the Linode Terraform Provider.

If you're new to containerization and Kubernetes, check out our [Understanding Kubernetes eBook](#) or take a [free beginner's course](#) on using LKE from our friends at [KodeKloud](#).

Importing Existing Linode Infrastructure to Terraform

You don't need to start a brand new project to begin 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)
- Kubernetes Clusters
- Managed Database Clusters
- Domains and Domain Records
- Block Storage Volumes
- Object Storage Buckets
- NodeBalancers

Step-by-step instructions to import each type of Linode infrastructure is available on our [Import Existing Infrastructure to Terraform guide](#).

Terraform Automation Options

Every company and organization has different needs in terms of workflows. Every application has its own needs for update scheduling, deployments, and version control. Terraform can automate your infrastructure provisioning, but what about adding other layers of automation and more advanced insights?

The needs for more features and support services are exactly what led HashiCorp to develop a SaaS Terraform offering, Terraform Cloud, and the self-managed Terraform Enterprise. Terraform Enterprise is offered as a private installation, so it can be hosted on your preferred cloud provider, but there are 3 extremely important factors to consider while weighing using Terraform Enterprise versus building your own automation:

- **Cost:** It's in the name. Terraform Enterprise is geared toward larger organizations. Though smaller organizations and individual developers can benefit from some features, the cost of ownership is likely unfeasible for smaller scales.
- **Ownership:** The features and workflows in Terraform Enterprise are owned by HashiCorp, instead of being owned by you as the application developer.
- **Customization:** A one size fits all approach to automation is unlikely to fit all of your application's requirements. Building your own automation workflows allows you to figure out exactly what your application needs.

Though a model like Terraform Enterprise might seem more appealing to organizations that prioritize saving time, it isn't a viable option for most developers. Here are overviews on free automation tools and workarounds to automate your Terraform workflows.

GitHub Actions

Coordinating changes executed by Terraform with corresponding documentation or pull requests in your GitHub repository can add extra steps. This process can be automated using GitHub actions. Save your configuration file that shows what Terraform changes were made, and use GitHub actions to create a pull request to add that documented output to a branch.

Read the Terraform guides on [GitHub Actions](#).

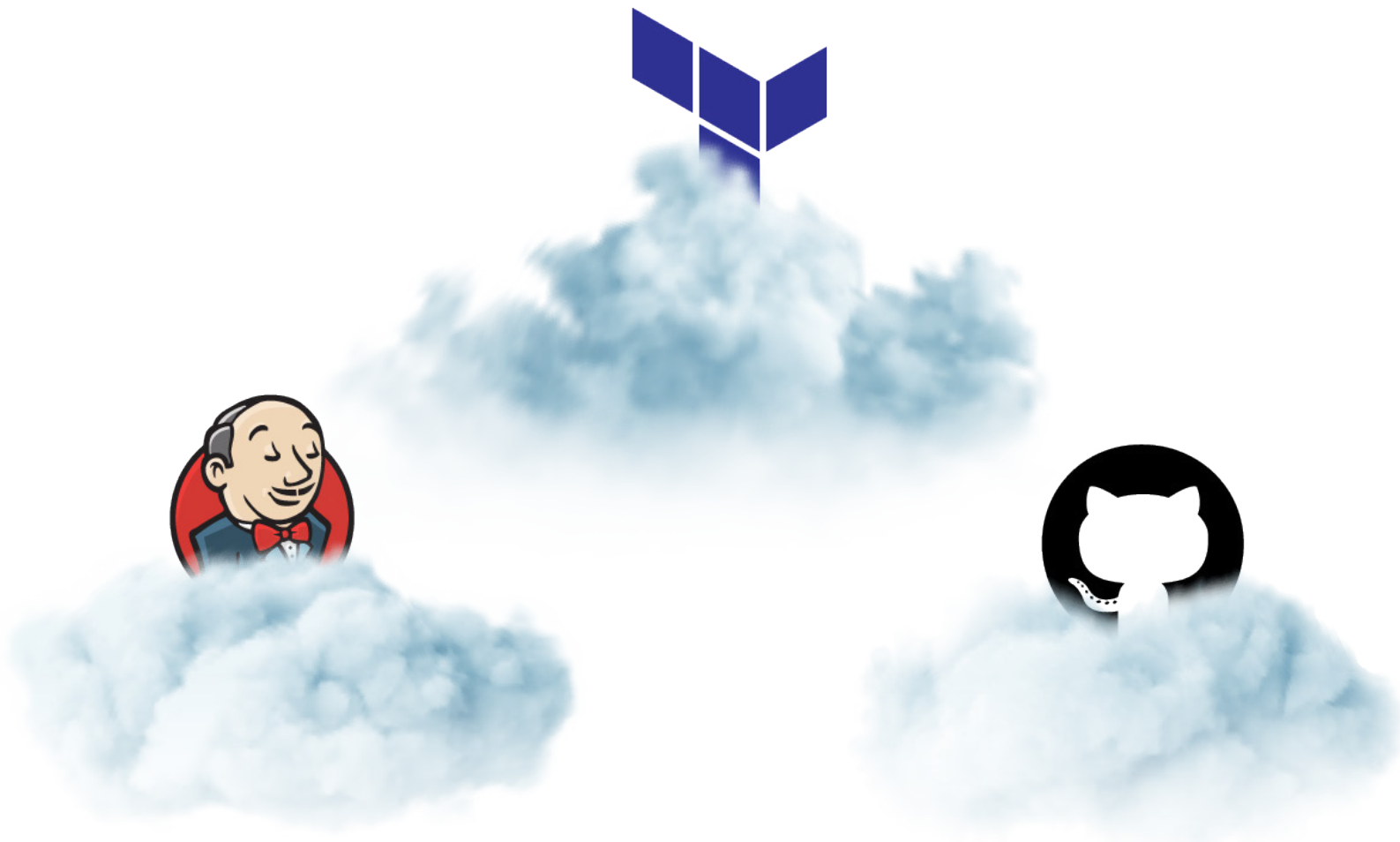
Jenkins

Jenkins is an open source CI/CD tool and automation server. You can use Jenkins to check out Terraform scripts based on specific parameters. Create a Terraform script that you would like to run based on certain conditions, such as a resource trigger or at a specific time. Deploy Jenkins to an instance. You can easily do that on Linode using our [Jenkins Marketplace app](#). Follow the steps to install Terraform to that instance. After you set up the options in Jenkins, it will check out .tf files from your repository using the pipeline you create. Jenkins will also execute the Terraform script, and you'll see the changes reflected in your cloud resources.

If you're a new Jenkins user, get started with a free Jenkins course from FreeCodeCamp.org via their [YouTube channel](#). Search and install Jenkins modules on the [Terraform Registry](#) for further integration.

Curate Your Automation

When a tool steadily increases in popularity, the number of “sub-tools” or services created by other developers also grows. View [Awesome Terraform](#), an open source list of Terraform resources and tools recommended by contributors.



Comparing Terraform and Ansible

As the most popular IaC tools, Terraform and Ansible are often compared. Each tool serves different purposes, and choosing one over the other ultimately depends on what you're looking to automate. If you're managing a significant amount of cloud infrastructure, the answer is probably Terraform AND Ansible.

What is Ansible?

Ansible is a configuration management tool that 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 operates on a designated instance in isolation from the network environment where the deployment is taking place. Ansible is written in YAML and doesn't require specific programming knowledge to get started.

What do Terraform and Ansible have in common?

- They are both **agentless**, meaning both are capable of SSH access to make changes to resources. Terraform and Ansible do not be installed on a specific instance.
- Both tools are used to reduce or eliminate problems associated with manual configuration.
- Both are easy to write and understand, and neither require experience with a particular coding language.
- Both require read/write access to your cloud infrastructure account via an API token.


What are the primary differences?

- Terraform is declarative whereas Ansible is procedural. Ansible allows conditional statements, including “when” to help specify sequences, i.e. if x service has started, do y.
- Ansible templates are stateless. You can run the same template on repeat. Terraform's built-in version control makes it a more intelligent state management tool.
- Terraform follows a mutable infrastructure model, whereas Ansible's default approach is immutable infrastructure.
- Terraform configures infrastructure based on your cloud provider(s), i.e. plan type and region. Ansible configures the specifics of the infrastructure, like the OS version.


Terraform and Ansible can perform many of the same operations, but you should always select the best tool for the tasks you want to accomplish more effectively. The following is a high-level overview on what some of the most important aspects of cloud infrastructure management look like using Terraform and Ansible. There are also recommendations for which tool to use based on what you're looking to simplify or automate.

Note: The steps outlined are a brief summary of step-by-step instructions on deploying and managing Linode infrastructure that can be found in our documentation.

Provisioning Infrastructure

USING TERRAFORM	USING ANSIBLE
<ol style="list-style-type: none"> 1. Install Terraform 2. Download the Linode Terraform Provider 3. Configure the Terraform environment in your <code>.tf</code> file 4. Create and add a read/write Linode API token 5. Add resource specifications 6. Initialize Terraform configuration with the <code>terraform init</code> command <p>Learn more.</p>	<ol style="list-style-type: none"> 1. Provision a control node 2. Add a limited user to your account that can access the node 3. Install Ansible on your control node 4. Create a read/write Linode API token 5. Install Linode Ansible collection 6. Configure an Ansible Vault password file 7. Create your Ansible configuration file 8. Create an Ansible playbook using the <code>linode.cloud.instance</code> Fully Qualified Collection Name <p>Learn more.</p>
BEST TOOL	
<div>  <p>HashiCorp Terraform</p> </div> <p>Terraform! As you can see in the summary of steps to provision infrastructure using both tools, more steps and setup are required to deploy a new Linode using Ansible. In terms of just getting infrastructure provisioned, Terraform is the most efficient tool.</p>	

Configuration Management

USING TERRAFORM	USING ANSIBLE
<p>Terraform provisioners should only be used as a last result, and will add significant complexity to your workflow instead of simplifying it.</p> <p>Learn more via Terraform Docs.</p>	<ol style="list-style-type: none"> 1. Follow Ansible setup instructions 2. Add a limited user account 3. Create a password hash using Ansible Vault 4. Create an inventory file on your control node 5. Write a YAML playbook with the actions written in the order they should be executed 6. Create a file in your home directory 7. Add and run the playbook from your control machine <p>Learn more.</p>
BEST TOOL	
<div>  <p>Ansible! In its simplest form, an Ansible Playbook will define a group of target hosts, variables to use within the Playbook, a remote user to execute the tasks, and a set of named tasks to execute using relevant Ansible modules.</p> </div>	

Though Terraform is not a configuration management tool, it can be used with one like Ansible for a more comprehensive solution. Terraform can provide the higher-level abstraction of the network, while Ansible's configuration management can be used on the individual resources.

Terraform and Ansible aren't the only IaC tools available. In the next section, you'll learn about more tools and how each tool can automate or optimize different areas of your development workflow and application.

Combining IaC Tools

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.

Packer

Packer is a HashiCorp-maintained open source tool that is used to create machine images. A machine image provides the operating system, applications, application configurations, and data files that a virtual machine instance will run once it's deployed. Packer can be used in conjunction with common configuration management tools like Puppet and Ansible. Packer templates store the configuration parameters used for building an image. This standardizes the imaging building process and ensures that everyone using that template file will always create an identical image. This helps your team maintain an immutable infrastructure.

[Learn how to start using Linode and Packer to create custom images.](#)

Pulumi

Pulumi's CrossCode service works as a universal translation layer allowing organizations to manage 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.](#)

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](#).

Our Take

IT infrastructure will always be a significant spend for teams and organizations around the world. While operating on a cloud model shifts spending from capital expenditures (CapEx) to operating expenses (OpEx), the increased complexity of growing environments remains a major time sync for DevOps or traditional network and system teams. IaC tools like Terraform are essential tools that enable you to remove redundant tasks and decrease the potential for error in provisioning and changing your deployments. They also enable a multicloud strategy and help organizations 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, you 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](#).
- Want to see the full Terraform installation and deployment process? [Watch our popular Beginner's Guide to Terraform video](#) on YouTube.
- Follow step-by-step instructions to [Import Existing Linode Infrastructure to Terraform](#), or [watch a video tutorial](#) on YouTube.
- Become a [certified Terraform Associate](#) through Hashicorp's verified certification program.
- Learn how to use Terraform and other IaC tools with instructor Justin Mitchel of Coding for Entrepreneurs. [Download the comprehensive Try IaC ebook](#) or [register for the on-demand video series](#).

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](#).

Use other HashiCorp tools on the most scalable cloud provider

Linode makes it easy to use other HashiCorp tools. Deploy Nomad and Vault on the Linode Marketplace, in addition to other apps to streamline your development workflows.



BROWSE APPS

About Linode

Linode cloud computing services from Akamai is one of the easiest-to-use and most trusted infrastructure-as-a-service platforms. Built by developers for developers, Linode accelerates innovation by making cloud computing simple, affordable, and accessible to all. Learn more at <http://linode.com> or follow Linode on [Twitter](#) and [LinkedIn](#).



Cloud Computing Developers Trust

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