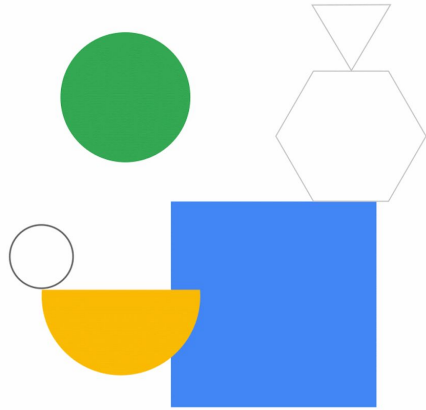


Introduction to Terraform for Google Cloud



Objectives

Upon completion of this module, you will be able to:

- | | |
|----|-------------------------------------------------------|
| 01 | Define infrastructure as code. |
| 02 | Explain the features and benefits of using Terraform. |
| 03 | Explain use cases of Terraform for Google Cloud. |
| 04 | Describe how to use Terraform for Google Cloud. |



Welcome to the “Introduction to Terraform for Google Cloud” module. This is an introductory module that covers the business need for Terraform. We’ll start with the basics by providing an overview of infrastructure as code (IaC), which is the basic concept for Terraform. We will explore how Terraform can be used as an IaC tool on Google Cloud and also cover its features and benefits. We’ll then look at how Terraform transforms lines of code into real infrastructure on Google Cloud.

Let’s jump in!

Topics

01 [Infrastructure as code](#)

02 Terraform overview

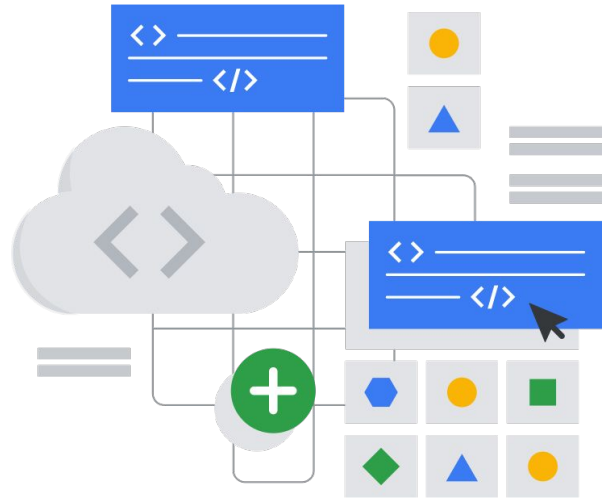
03 Using Terraform



Let us start with the fundamental concept required to understand Terraform, Infrastructure as Code (IaC).

What is infrastructure as code (IaC)?

Instead of clicking around a web UI or using SSH to connect to a server and manually executing commands, **with IaC you can write code in files to define, provision, and manage your infrastructure.**

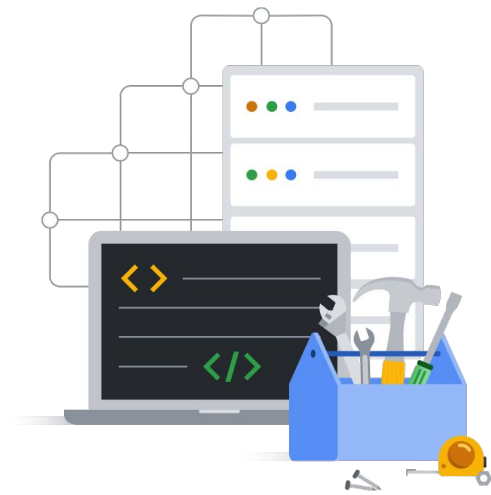


Infrastructure as code is a foundation concept for understanding Terraform. Let's start by defining what Infrastructure as code means and what problems it solves.

Infrastructure as code, as the name indicates, involves managing and provisioning the cloud infrastructure as code through a configuration language. In other words, instead of clicking around a web UI or using SSH to connect to a server and manually executing commands, **with IaC you can write code in files to define, provision, and manage your infrastructure.**

Gone are the days where a system administrator has to manually configure hundreds of servers, networks, and firewall rules by interacting with a UI. With IaC, you only need to declare the desired end state of the infrastructure, and IaC tools manage provisioning the infrastructure.

Let's evaluate the gaps that the tech industry faces and the role IaC plays in filling those gaps.



DevOps

Emphasizes the collaboration and communication of both software developers and IT operations teams

Automates the process of software delivery and infrastructure changes

The rapid increase of business demand today has manifested a **cultural change** called **DevOps**, which emphasizes the **collaboration** and communication of both **software developers** and **IT operations** teams while **automating** the process of **software delivery** and **infrastructure changes**.

Problems that IaC can solve



Inability to scale rapidly

Requires rapid scaling of IT infrastructure



Operational bottlenecks

Challenge of managing infrastructure consistently in scale



Disconnected feedback loops

Communication gap between software and IT teams



High manual errors

Increased scale leads to increased human errors

With the growing competition among industries to benefit from the cloud, **software and IT operations** teams are **struggling** to efficiently and effectively **manage growing environments** and **rapidly expanding businesses**. This struggle involves four major challenges:

- **Inability to scale rapidly:** High business demand requires the rapid scaling of IT infrastructure across industries.
- **Operational bottlenecks:** Due to the rapid scaling of IT infrastructure, Ops teams need to overcome new organizational and technical bottlenecks, such as managing infrastructure **consistently** in scale.
- **Disconnected feedback loops:** Whenever the infrastructure is changed, the dev and Ops teams struggle to collaborate and audit changes. Both closing the communication gap and auditing changes between the software and IT teams are imperative for successful deployments.
- **Manual errors:** Increased quantity and scale has led to greater human error with the potential for significant impacts.

Benefits of IaC

Declarative	Specify the desired state of infrastructure, not updates.
Code management	Commit, version, trace, and collaborate, just like source code.
Auditable	Compare infrastructure between desired state and current state.
Portable	Build reusable modules across an organization.

Some of the benefits of using IaC are:

- The first major benefit of infrastructure as code is that it's **declarative**. The declarative nature of IaC makes it simple to implement infrastructure at scale. Instead of having to specify the exact sequence of steps to make a particular change, you can focus on the desired state of the infrastructure. For example, you might specify that you want three subnets across two regions with a production label. Terraform can then determine the necessary updates to make the live state match your desired state by adding a label to one existing subnet while creating two new subnets in additional regions. Through declarative abstractions, IaC tools free you to focus on the exact changes to your desired state while the tools manage the implementation details. The declarative approach allows anyone to read the state of the infrastructure instead of depending on what the system admin is thinking. We'll cover the differences between imperative and declarative in more detail shortly.
- Another benefit is simply that infrastructure as code can be checked into **source control and managed the same way as application source code**. This feature allows you to have a full history of all the changes to your infrastructure available in version history. If something breaks, you can roll back to a known good state. In contrast, if you made changes directly in the Google Cloud Console that caused an outage, you'd have to manually identify the breaking change and how to fix it. This is a much more error-prone process than immediately reverting to the last stable version of your

- infrastructure source code. With version control, the entire history of your infrastructure is now captured in the commit log. Additionally, you can empower developers to collaborate on changes. For example, if a developer needs a port opened on a firewall to support their application, they can simply submit a pull request to suggest the change instead of waiting in a ticketing queue for an administrator to manually open the port. Changes can be discussed, reviewed, and audited without causing information silos—which can lead to more collaborative ownership as infrastructure evolves.
- Another major benefit of IaC is that it adds an auditable history of your infrastructure. Normally, your infrastructure doesn't contain comments: you might see that a change was made through audit logs, but it's often difficult to understand why. By modeling infrastructure as code, you can include explanations of exactly why a specific change was made. For example, you might comment that a specific subnet allocation is required for a certain application. This commentable, auditable history gives you a clear view of how infrastructure has evolved over time. When proposing new changes, you can also preview them and inspect exactly how they will affect live infrastructure before they're applied. This allows you to detect drift and ensure that your infrastructure remains robust.
- Portability is another major benefit of infrastructure as code. By packaging infrastructure as code, you can build reusable modules that encapsulate conventions and allow you to consistently build infrastructure from a shared template. Instead of having to rebuild infrastructure manually each time, you can have many instances of the same template deployed for different applications or regions. In fact, this feature is exactly what enables CFT (Cloud Foundation Toolkit). We've developed reusable modules that can be used across many organizations to instantly adopt best practices without having to rebuild them from scratch. Using the library of reusable, documented, tested infrastructure code makes it easier to scale and evolve your infrastructure.

Provisioning versus configuration

Infrastructure as code

- Used for provisioning and managing cloud resources.
- Example: Creating and provisioning a VM instance.
- Referring to frameworks that manipulate Google Cloud APIs to deploy the infrastructure.

Configuration Management

- Used for virtual machine OS-level configuration.
- Example: Configuring the internals of the VMs.
- Referring to package configurations and software maintenance.

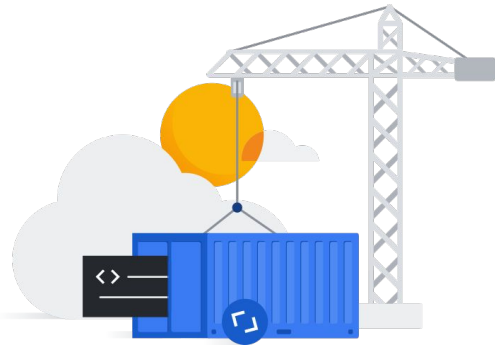
Provisioning and configuring are two terms that are sometimes misinterpreted. Fundamentally, IaC is used for provisioning and managing cloud resources, and configuration management is used for VM OS-level configuration.

For example, **infrastructure as code** is used for creating and **provisioning** a VM instance, and **configuration management** is used for **configuring** the internals of the VMs.

Configuration management is a **broad topic** that covers more **than we'll be able to discuss** in this section. To keep it simple, activities like **configuring a VM for application dependencies** are considered configuration management. Configuration could consist of starting services, installing dependencies, installing applications, running updates, and much more, so it was a lot of manual effort.

To elaborate further, the term **Infrastructure as code** generally refers to **frameworks** that manipulate Google Cloud APIs to **deploy the infrastructure** required to run application code, but configuration management refers to package configurations and software maintenance.

Provisioning versus configuration



Infrastructure as a code

- ✓ Launch a GKE cluster

Configuration management

- ✓ Deploy containers into the GKE cluster

Let's look at another common example to differentiate between provisioning and configuration.

laC automates tasks involved in **launching** a **GKE cluster** into Google Cloud, and **configuration management** automates tasks involved in **deploying containers** into the GKE cluster.

laC takes the declarative approach to infrastructure

Imperative (command)

Command line

"Give me five servers"

How to create?

VS.

Declarative (statement)

YAML

"I should have five servers"

What to create?

One of the important principles to understand about infrastructure as code is that it is declarative.

This contrasts with the **imperative** model that most programming languages use, where you might specify the exact action you want to take, like creating five servers. This model is challenging for infrastructure because, if you need to run the script again—potentially to apply some updates—it might create five new servers even if some already exist. With an imperative workflow, you don't have an easy way of determining the difference between live infrastructure and your desired state; therefore, the same resources are created repeatedly.

You declare the **desired** state of your infrastructure and let the tool determine the details.

For example, you might **declare** that you should have five servers. The first time Terraform runs, it might create all five servers. Then someone accidentally deletes one of the servers. The next time Terraform runs, it would recognize that the four servers already exist and restore only the missing server.

This is why we encourage adopting declarative infrastructure tools like Terraform. Focus on how the infrastructure *should* be, while automation determines the details of updating the infrastructure to match your desired state. Declarative management enables focusing on the WHAT rather than the HOW.

Topics

- | | |
|----|------------------------------------|
| 01 | Infrastructure as code |
| 02 | Terraform overview |
| 03 | Using Terraform |



Now that we've covered what IaC is, let's explore more about Terraform.

Terraform is an infrastructure as code tool created by HashiCorp that lets you provision Google Cloud resources with **declarative** configuration files

Terraform is an infrastructure as code tool created by HashiCorp, and is currently licensed under the HashiCorp Business Source License v1.1. It lets you provision Google Cloud resources, such as virtual machines, containers, storage, and networking, with declarative configuration files. Terraform allows infrastructure to be expressed as code in a simple, human-readable language called HashiCorp Configuration Language (HCL). It reads configuration files and provides an execution plan of changes, which can be reviewed for safety and then applied and provisioned. At a high level, Terraform allows operators to author files that contain definitions of their desired resources on the Google Cloud provider and automates the creation of those resources at the time of the apply step.

Terraform features



Multi-cloud and multi-API



Enterprise support



Large community



Infrastructure provisioning

Terraform features include:

- **Multi-cloud and multi-API:** Support for all major cloud providers including Google Cloud and many other services exposed through an API (like GitHub and Kubernetes).
- **Enterprise support:** Three different editions that range from self-hosted to fully managed with enterprise-level support.
- **Large community:** Thousands of reusable modules are publicly available from the Terraform Registry for Google Cloud deployments.
- **Infrastructure provisioning:** Terraform focuses on provisioning infrastructure, not configuring it.

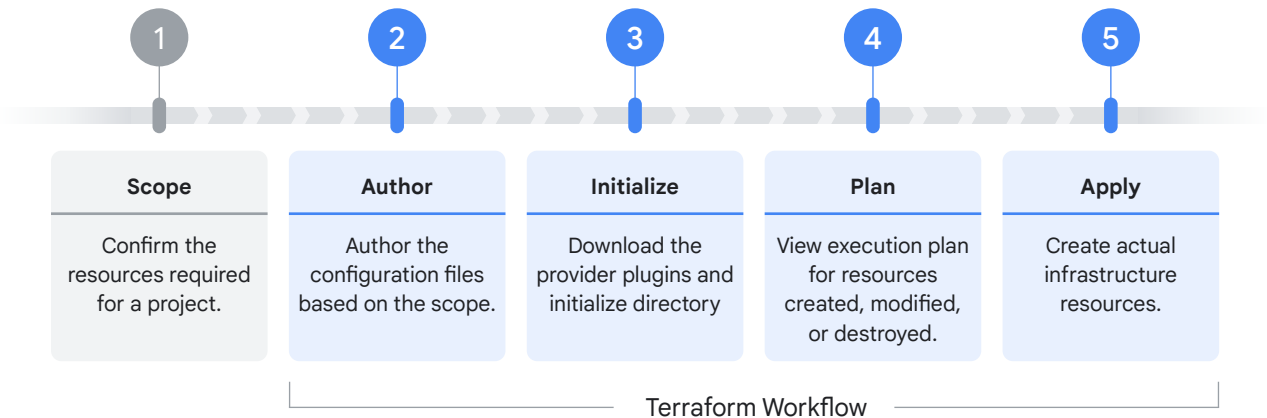
Terraform for Google Cloud

- ✓ Provision resources
- ✓ Create resource dependencies
- ✓ Standardize configurations
- ✓ Validate inputs to resource arguments

You can use Terraform for Google Cloud to:

- **Provision resources**, which means you can use resource blocks to define infrastructure elements such as VMs, network, firewall resources.
- **Create resource dependencies**: You can create explicit dependencies between resources so that a given resource can only be created after the creation on another resource.
- **Standardize configurations**: You can standardize how a given resource is created by creating reusable modules.
- **Validate inputs to resource arguments**: You can limit the values that a user can provide for a given resource argument using validation rules within Terraform.

IaC configuration workflow

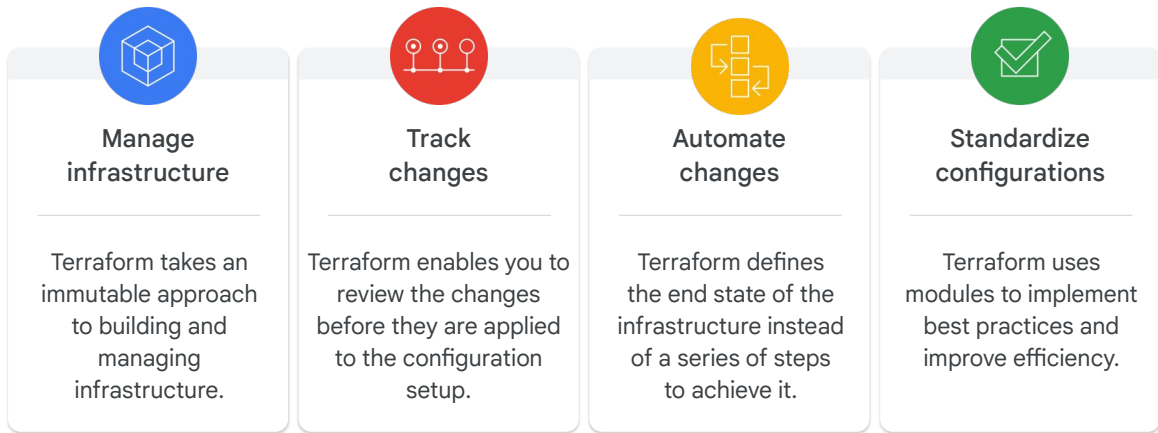


This is a typical IaC configuration workflow.

1. You first determine what resources should be created for a project. For example, for a common 2-tier architecture, you need a pool of web servers that use a database tier. Thus, at this phase, you scope the requirements for the Google Cloud resources, such as the type of compute and database instances that you need, and plan how these instances should be connected.
2. The terraform workflow begins from the author phase. You **author** the configuration code for the infrastructure you want to create. Referring back to our example, at this phase, you will code the configuration of those instances and the VPC network in the HashiCorp language and organize them in configuration files, such as variables, main, and tfvars, which we will explore later in the course.
3. You then **initialize** Terraform for the project, which installs any provider plugins or modules you'll use. Initializing Terraform involves executing the `terraform init` command, which initializes the Terraform configuration directory.
4. In the **plan** step, you execute the `terraform plan` command that enables you to review the changes Terraform makes to the Google Cloud infrastructure.
5. After you review the configuration described in the generated plan, **apply** it and create both real infrastructure and a state file for your infrastructure.

Later in the course we'll explore the Terraform workflow in more detail.

Terraform use cases



You can use Terraform for various use cases.

- **Manage infrastructure:** You can use Terraform to take an immutable approach to building Google Cloud infrastructure. An immutable approach means you write code to reduce the complexity involved in upgrading or modifying the services and infrastructure. You can reuse code blocks from the Terraform registry, which is a collection of configuration files that exist for Google Cloud services.
- **Track infrastructure changes:** Whenever a new change is planned or applied, Terraform prompts you to approve the change before Terraform modifies the state of the infrastructure. When an infrastructure is applied (created) by using Terraform, a state file is automatically generated. State file reflects the current state of your infrastructure. This state file is used to determine the changes you made to the existing infrastructure by providing an overview of the number and type of Google Cloud resources modified in your configuration.
- **Automate changes:** Because Terraform configuration files are declarative in nature, which means that you describe the end state of the Google Cloud infrastructure instead of a series of steps to achieve that end state. Thus, you do not have to write detailed, step-by-step instructions to build the infrastructure. You only need to define the end state of the infrastructure, and Terraform manages the dependency and provisions the resources.
- **Standardize configurations:** Terraform supports a concept called *modules*, which enable you to define infrastructure configuration that can be reused to

- save time and implement best practices. You can also leverage the publicly available modules from the registry.

Terraform can also be used to automate the enforcement of policies on the types of resources teams can provision and use.

Topics

01 Infrastructure as code

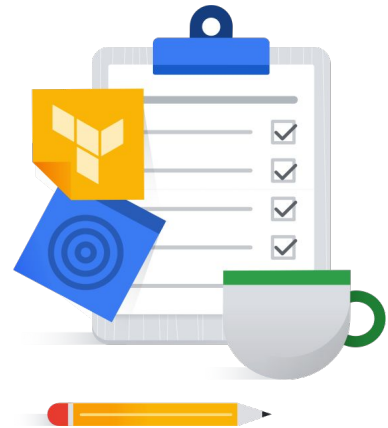
02 Terraform overview

03 [Using Terraform](#)



Using Terraform




- 1 Terraform recognizes configuration files written in .tf file.
- 2 Terraform generates an execution plan.
- 3 Terraform uses this plan to create infrastructure.
- 4 Terraform determines the changes and creates incremental execution plans.



So, how can you use Terraform to create, update, or destroy infrastructure resources?

1. In order to use Terraform, you must write your infrastructure as code in configuration files. These configuration files describe to Terraform the resources you want to provision.
2. Terraform generates an execution plan that describes what it will do to reach the desired state, and then executes the plan to build the described infrastructure.
3. Terraform creates this infrastructure and saves the state.
4. As the configuration changes, Terraform can determine what changed and create incremental execution plans that can be applied.

Running Terraform in production

	Managed	Pros	Cons
Terraform Community Edition		<ul style="list-style-type: none"> • Deployed on a local machine or compute resource in cloud • No license cost • Use public registry within your code 	<ul style="list-style-type: none"> • Does not support concurrent deployments • Only interfaced through CLI
Terraform Cloud		<ul style="list-style-type: none"> • SaaS based version • Small operational overhead • Comes in three plans • Supports concurrent deployments • Can be accessed through GUI and CLI 	<ul style="list-style-type: none"> • License cost for advanced features
Terraform Enterprise		<ul style="list-style-type: none"> • Private implementation • Supports concurrent deployments • Secure deployment • Can be accessed through GUI and CLI 	<ul style="list-style-type: none"> • Infrastructure and license costs • Large operational overhead

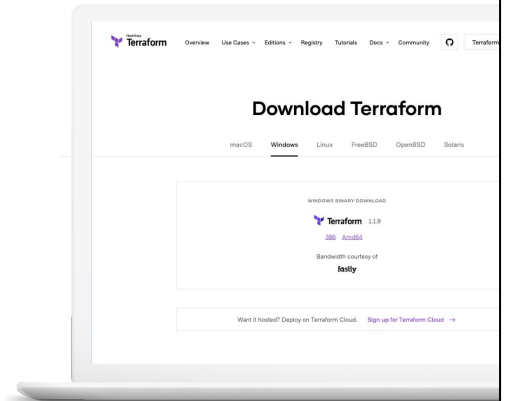
Google Cloud

Terraform is available in three editions: Community Edition, Cloud, and Enterprise.

- **Community Edition**, is a free version of the software available for download on a local machine or compute resource in the cloud. With this version, many features are available with no license cost. You can provision several different types of resources and codify your infrastructure. **Community Edition** grants access to publicly available templates so you can apply code writing best practices.
You can only use the Community Edition version on your local machine, and it does not support concurrent deployments. Terraform **Community Edition** does not have version control, so you can't track changes or ensure that your commits do not affect the infrastructure setup. With the **Community Edition** version, Terraform can only be interacted with through the CLI, whereas Terraform Cloud and Enterprise come with a GUI option.
- **Terraform Cloud** is useful for collaborative environments. Terraform Cloud has three plans: free, standard and plus. Unlike Terraform Community Edition, Terraform Cloud and Terraform Enterprise support concurrent deployment. With Terraform Cloud – like most other SaaS solutions – the operational overhead is low.
- **Terraform Enterprise** is hosted on-premises or on an infrastructure controlled by you. Due to infrastructure maintenance and its integration with Terraform, Terraform Enterprise has a high operational overhead.

Installing Terraform on local machine

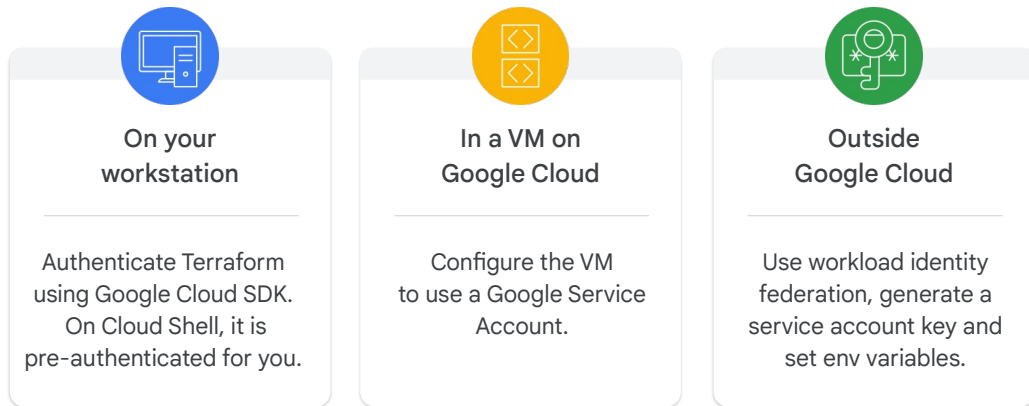
- 1 Download the appropriate package based on your system.
- 2 Unzip the package. Terraform includes a single binary called `terraform`.
- 3 Edit the `PATH` variable to include `terraform`.
- 4 Verify the installation, enter `terraform -help` in a new terminal.



Terraform is pre-installed when it's used on Cloud Shell. But if you want to use Terraform on your local machine, you must install it. You can install Terraform as a binary package or even use popular package managers. To perform a manual installation for a Windows machine.

1. Download the appropriate package based on your system.
2. Unzip the package. Terraform includes a single binary called `terraform`.
3. Edit the `PATH` variable to include `terraform`
4. Verify the installation by entering `terraform -help` in a new terminal

Authentication for Google Cloud

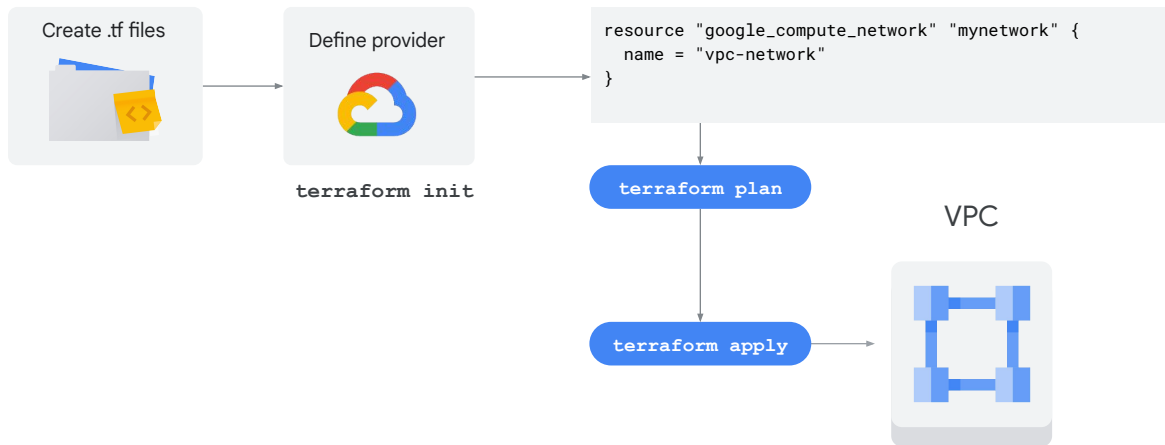


The authentication mechanism used for Google Cloud varies based on where Terraform is run. Note that it is the Google Cloud provider that will require authentication to communicate to the resource API for resource creation and not Terraform.

- If you're running Terraform **on your workstation**, authenticate by using the Google Cloud SDK. To do this, install gcloud. If you are not sure of how to install gcloud, refer to the [Google documentation](#) for detailed steps. Then run `gcloud auth application-default login` command to authenticate with Google Cloud. But if you're using Terraform on Cloud Shell, it's pre-authenticated for you. Cloud Shell is a Compute Engine virtual machine. The service credentials associated with this virtual machine are automatic, so there is no need to set up or download a service account key.
- If you're running Terraform in a **VM on Google Cloud**, configure that VM to use a Google Service Account. This allows Terraform to authenticate to Google Cloud without a separate credential/authentication file. Ensure that the VM has the Google Cloud API (<https://www.googleapis.com/auth/cloud-platform>) enabled.
- If you're running **Terraform outside Google Cloud**, you can also use a Google Cloud Service Account with Terraform. From the service account key page in the Cloud Console choose an existing account, or create a new one. Next, download the JSON key file. Name it something you can remember, and

- store it somewhere secure on your machine. You supply the key to Terraform using the environment variable `GOOGLE_APPLICATION_CREDENTIALS`, setting the value to the location of the file. Terraform can then use this key for authentication. But this method has a few limitations, and a recommended practice is to use workload identity federation.
 - **Using workload identity federation:** Service account keys in general are short-lived, do not allow key rotation, and also need to be protected. Use workload identity and workload identity federation as an alternative to mitigate short-lived identity tokens when running Terraform outside of Google Cloud. An example environment where might run outside Google Cloud is Gitlab CI environment. For detailed information about steps, refer to the [documentation](#).

Example: Creating a VPC network



Once you've downloaded Terraform and authenticated successfully, you have everything ready to author and create Terraform configuration. Let's go over an example of creating a simple VPC network named `mynetwork` by applying the Terraform workflow we just covered.

Create the configuration file with a `.tf` extension and define Google Cloud as the provider and run the `terraform init` command to initialize the provider. Then add the HCL code involved in creating a Google Compute instance. We won't cover the details of the code for now but will discuss it in more detail in the next module. For now, let's focus on the workflow. After you have the final code saved, switch to the directory where you've saved the file and execute the `terraform plan` and `terraform apply`. When the apply command is successfully executed, you'll have a new VPC network named `mynetwork` created on Google Cloud. We will explore these commands in more detail in the next module.

Quiz



Quiz | Question 1

Question

Select the three Terraform editions available in production.

- A. Terraform Cloud
- B. Terraform Analytics
- C. Terraform Community Edition
- D. Terraform Cyber
- E. Terraform Enterprise

Quiz | Question 1

Answer

Select the three Terraform editions available in production.

- A. Terraform Cloud
- B. Terraform Analytics
- C. Terraform Community Edition
- D. Terraform Cyber
- E. Terraform Enterprise

Quiz | Question 2

Question

Select the two use-cases for Terraform.

- A. Automate changes
- B. Provision an application
- C. Standardize configurations
- D. Provide financial analytics
- E. Run OS level customization

Quiz | Question 2

Answer

Select the two use cases for Terraform.

- A. Automate changes
- B. Provision an application
- C. Standardize configurations
- D. Provide financial analytics
- E. Run OS level customization

Quiz | Question 3

Question

Which one of the following statements is true regarding Terraform?

- A. Terraform can be used for multi-cloud deployments.
- B. Terraform can only be used for on-premises deployments.
- C. Terraform is used to configure applications on Google Cloud.
- D. Terraform uses the imperative approach to define infrastructure components.

Quiz | Question 3

Answer

Which one of the following statements is true regarding Terraform?

- A. Terraform can be used for multi-cloud deployments.
- B. Terraform can only be used for on-premises deployments.
- C. Terraform is used to configure applications on Google Cloud.
- D. Terraform uses the imperative approach to define infrastructure components.

Quiz | Question 4

Question

What is infrastructure as code (IaC)?

- A. IaC is a cloud computing model that offers resources on demand to businesses and individuals by using the cloud.
- B. IaC is a tool to maintain consistency in an application deployment environment.
- C. IaC is a process to define, provision, and manage cloud infrastructure by writing code in files.
- D. IaC is a data warehouse running on serverless infrastructure.

Quiz | Question 4

Answer

What is infrastructure as code (IaC)?

- A. IaC is a cloud computing model that offers resources on demand to businesses and individuals by using the cloud.
- B. IaC is a tool to maintain consistency in an application deployment environment.
- C. IaC is a process to define, provision, and manage cloud infrastructure by writing code in files.
- D. IaC is a data warehouse running on serverless infrastructure.

Module review

- | | |
|----|-------------------------------------------------------|
| 01 | Define infrastructure as code. |
| 02 | Explain the features and benefits of using Terraform. |
| 03 | Explain use cases of Terraform for Google Cloud. |
| 04 | Describe how to use Terraform for Google Cloud. |



This brings you to the end of the first module of the Getting Started with Terraform for Google Cloud course. This module covered the basics of Terraform and Infrastructure as Code.

We defined Infrastructure as code and covered the business case for IaC. We explained the features and benefits of using Terraform. We also discussed common use cases, how to use Terraform, and the consumption model for Terraform.

