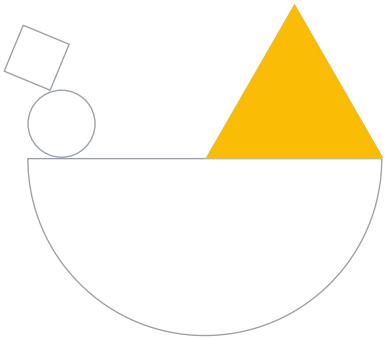
Google Cloud

Infrastructure Automation



Now that we've covered several of Google Cloud's services and features, it makes sense to talk about how to automate the deployment of Google Cloud infrastructure.

Calling the cloud API from code is a powerful way to generate infrastructure. But writing code to create infrastructure also has some challenges. One issue is that the maintainability of the infrastructure depends directly on the quality of the software. For example, a program could have a dozen locations that call the cloud API to create VMs. Fixing a problem with the definition of one VM would require first identifying which of the dozen calls actually created it. Standard software development best practices will apply, and it's important to note that applications undergo changes rapidly, requiring maintenance on your code.

Clearly another level of organization is needed. That's the purpose of Terraform. Terraform uses a system of highly structured templates and configuration files to document the infrastructure in an easily readable and understandable format. Terraform conceals the actual Cloud API calls, so you don't need to write code and can focus on the definition of the infrastructure.

Agenda

01 Terraform

Lab: Automating the Deployment of Infrastructure Using Terraform

02 Google Cloud Marketplace

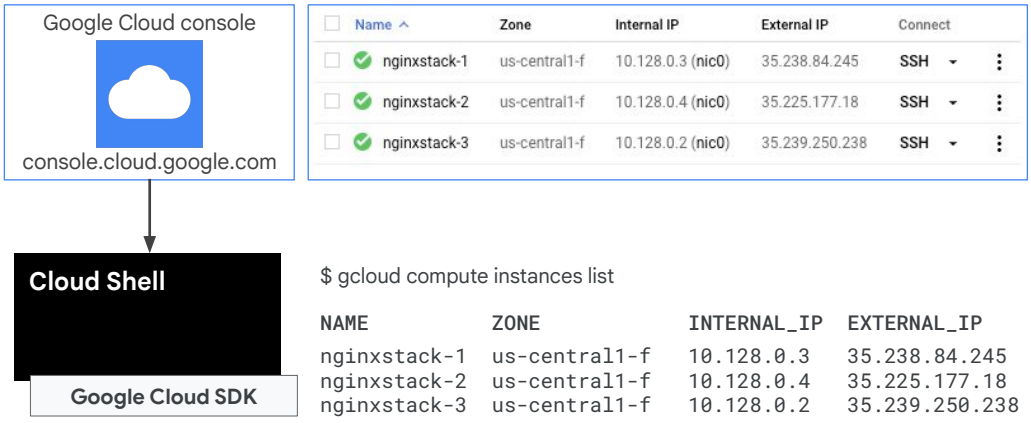
In this module, we cover how to use Terraform to automate the deployment of infrastructure and how to use Google Cloud Marketplace to launch infrastructure solutions.

You will use Terraform to deploy a VPC network, a firewall rule, and VM instances in the lab of this module.

Terraform

Let’s start by talking about Terraform.

Google Cloud console, Cloud SDK, and Cloud Shell



So far, you have been creating Google Cloud resources using the Google Cloud console and Cloud Shell. We recommend the console when you are new to using a service or if you prefer a UI. Cloud Shell works best when you are comfortable using a specific service and you want to quickly create resources using the command line. Terraform takes this one step further.

Infrastructure as code (IaC) allows quick provisioning and removing of infrastructures

- Build an infrastructure when needed.
- Destroy the infrastructure when not in use.
- Create identical infrastructures for dev, test, and prod.
- Can be part of a CI/CD pipeline.
- Templates are the building blocks for disaster recovery procedures.
- Manage resource dependencies and complexity.
- Google Cloud supports many IaC tools.

Terraform is one of the tools used for Infrastructure as Code or IaC. Before we dive into understanding Terraform, let's look at what Infrastructure as Code is. In essence, infrastructure as code allows for the quick provisioning and removing of infrastructures.

The on-demand provisioning of a deployment is extremely powerful. This can be integrated into a continuous integration pipeline that smoothens the path to continuous deployment.

Automated infrastructure provisioning means that the infrastructure can be provisioned on demand, and the deployment complexity is managed in code. This provides the flexibility to change infrastructure as requirements change. And all the changes are in one place. Infrastructure for environments such as development and test can now easily replicate production and can be deleted immediately when not in use. All because of infrastructure as code.

Several tools can be used for IaC. Google Cloud supports Terraform, where deployments are described in a file known as a configuration. This details all the resources that should be provisioned. Configurations can be modularized using templates which allow the abstraction of resources into reusable components across deployments.

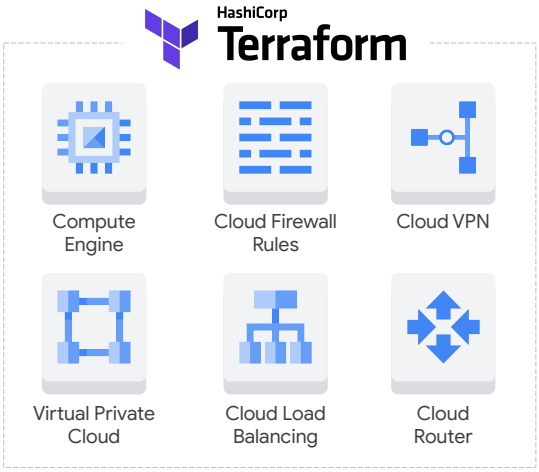
In addition to Terraform, Google Cloud also provides support for other IaC tools, including:

- Chef
- Puppet
- Ansible
- Packer

In this course we will focus on Terraform.

Terraform is an infrastructure automation tool

- Repeatable deployment process
- Declarative language
- Focus on the application
- Parallel deployment
- Template-driven



Terraform lets you provision Google Cloud resources—such as virtual machines, containers, storage, and networking—with declarative configuration files. You just specify all the resources needed for your application in a declarative format and deploy your configuration. HashiCorp Configuration Language (HCL) allows for concise descriptions of resources using blocks, arguments, and expressions.

This deployment can be repeated over and over with consistent results, and you can delete a whole deployment with one command or click. The benefit of a declarative approach is that it allows you to specify what the configuration should be and let the system figure out the steps to take.

Instead of deploying each resource separately, you specify the set of resources which compose the application or service, allowing you to focus on the application. Unlike Cloud Shell, Terraform will deploy resources in parallel.

Terraform uses the underlying APIs of each Google Cloud service to deploy your resources. This enables you to deploy almost everything we have seen so far, from instances, instance templates, and groups, to VPC networks, firewall rules, VPN tunnels, Cloud Routers, and load balancers. For a full list of supported resource types, a link to the Using Terraform with Google Cloud documentation page is included in the Course Resources.

[https://cloud.google.com/docs/terraform?hl=en]

Terraform language

- Terraform language is the interface to declare resources.
- Resources are infrastructure objects.
- The configuration file guides the management of the resource.

```
resource "google_compute_network" "default" {
  name = "${var.network_name}"
  auto_create_subnetworks = false
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```


The Terraform language is the user interface to declare resources. Resources are infrastructure objects such as Compute Engine virtual machines, storage buckets, containers, or networks. A Terraform configuration is a complete document in the Terraform language that tells Terraform how to manage a given collection of infrastructure. A configuration can consist of multiple files and directories.

The syntax of the Terraform language includes:

- Blocks that represent objects and can have zero or more labels. A block has a body that enables you to declare arguments and nested blocks.
- Arguments are used to assign a value to a name.
- An expression represents a value that can be assigned to an identifier.

Terraform can be used on multiple public and private clouds

- Considered a first-class tool in Google Cloud
- Already installed in Cloud Shell

```
provider “google” {
   region      = “us-central1”
}

resource “google_compute_instance” {
   name        = “instance name”
   machine_type= “n1-standard-1”
   zone        = “us-central1-f”

   disk {
      image = “image to build instance”
   }
}

output “instance_ip” {
   value = “${google_compute.ip_address}”
}
```


Terraform can be used on multiple public and private clouds. Terraform is already installed in Cloud Shell.

The example Terraform configuration file shown starts with a provider block that indicates that Google Cloud is the provider. The region for the deployment is specified inside the provider block.

The resource block specifies a Google Cloud Compute Engine instance, or virtual machine. The details of the instance to be created are specified inside the resource block.

The output block specifies an output variable for the Terraform module. In this case, a value will be assigned to the output variable "instance_ip."

Example: Auto mode network with HTTP firewall rule

Let's look at a simple example in Terraform.

Before you get into the lab, let me walk you through how Terraform can be used to set up an auto mode network with an HTTP firewall rule.

For this example we are going to define our infrastructure in a single file, main.tf.

As our infrastructure becomes more complex we can build each element in a separate file to make the management easier.

Example: Auto mode network with HTTP firewall rule

main.tf

```
provider "google" {
}
```


Let’s start with the main.tf file. The main.tf file is where we specify the infrastructure we wish to create. It is like a blueprint for our desired state.

First we define the provider.

Example: Auto mode network with HTTP firewall rule

main.tf

```
provider "google" {
}
# [START vpc_auto_create]
resource "google_compute_network" "vpc_network" {
  project                 = var.project_id
  name                    = "my-auto-mode-network"
  auto_create_subnetworks = true
  mtu                     = 1460
}
```


Next we define our network, setting the auto create subnetworks flag to true which will automatically create a subnetwork in each region.

We also set the mtu to 1460.

Example: Auto mode network with HTTP firewall rule

main.tf

```
provider "google" {
}
.
.
.

# [START vpc_firewall_create]
resource "google_compute_firewall" "rules" {
  project     = var.project_id
  name        = "my-firewall-rule"
  network     = "vpc_network"
    allow {
    protocol = "tcp"
    ports    = ["80", "8080"]
  }
}
```


Next, we define our firewall. Here we are allowing TCP access to port 80 and 8080.

Terraform takes this main.tf file and uses it as the specification for what to create.

Deploying infrastructure with Terraform

```
terraform init

terraform plan

terraform apply
```


Once we have completed the main.tf file, we can deploy the defined infrastructure in Cloud Shell.

We use the command `terraform init` to initialize the new Terraform configuration. We run this command in the same folder as the main.tf file.

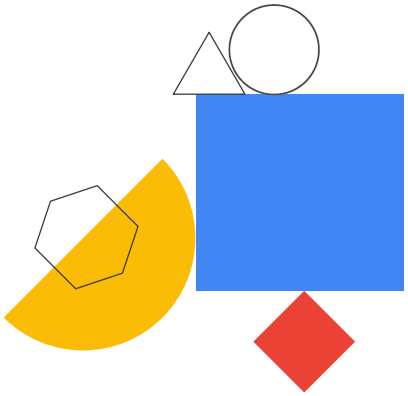
- The **`terraform init`** command makes sure that the Google provider plugin is downloaded and installed in a subdirectory of the current working directory, along with various other bookkeeping files. You will see an "Initializing provider plugins" message.
  Terraform knows that you're running from a Google project, and it is getting Google resources.

- The **`terraform plan`** command performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files.
  This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state.

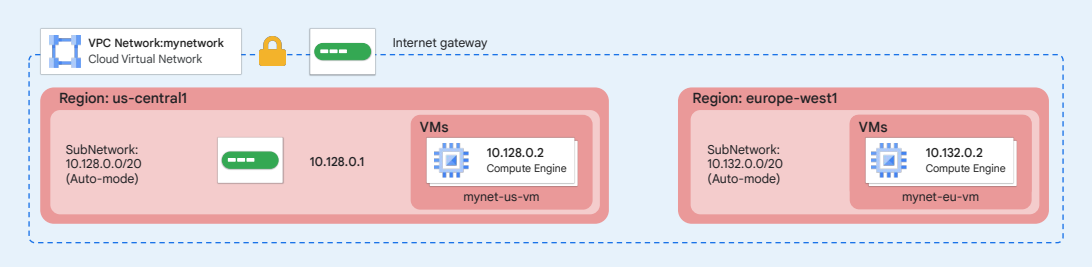
- The **`terraform apply`** command creates the infrastructure defined in the main.tf file. Once this command has completed you will be able to access the defined infrastructure.

Lab Intro

Automating the Deployment of Infrastructure Using Terraform



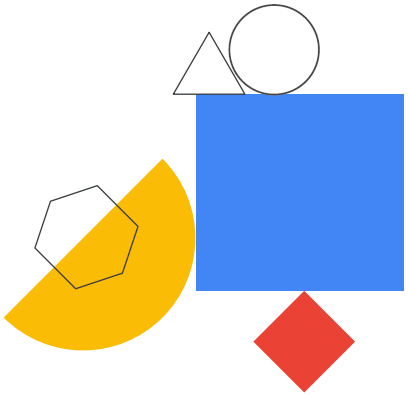
Let’s apply what you’ve just learned in a hands-on lab, where you will automate the deployment of VPC networks, firewall rules, and VM instances.



You deploy an auto mode network called mynetwork with a firewall rule to allow HTTP, SSH, RDP, and ICMP traffic. You also deploy the VM instances shown in this network diagram.

Review

Automating the Deployment
of Infrastructure Using Terraform



In this lab, you created a Terraform configuration with a module to automate the deployment of GCP infrastructure. As your configuration changes, Terraform can create incremental execution plans, which allows you to build your overall configuration step by step.

The instance module allowed you to re-use the same resource configuration for multiple resources while providing properties as input variables. You can leverage the configuration and module that you created as a starting point for future deployments.

You can stay for a lab walkthrough, but remember that GCP's user interface can change, so your environment might look slightly different.

Google Cloud Marketplace

Let’s learn a little more about Google Cloud Marketplace.

Google Cloud Marketplace

- Deploy production-grade solutions
- Single bill for Google Cloud and third-party services
- Manage solutions using Terraform
- Notifications when a security update is available
- Direct access to partner support

Google Cloud Marketplace lets you quickly deploy functional software packages that run on Google Cloud. Essentially, Cloud Marketplace offers production-grade solutions from third-party vendors who have already created their own deployment configurations based on Terraform. These solutions are billed together with all of your project's Google Cloud services. If you already have a license for a third-party service, you might be able to use a Bring Your Own License solution.

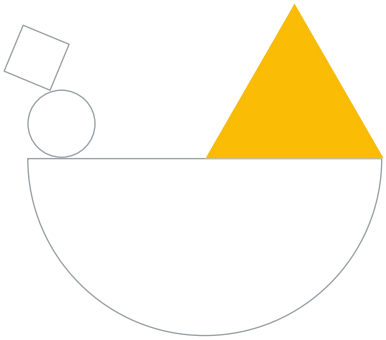
You can deploy a software package now and scale that deployment later when your applications require additional capacity. Google Cloud even updates the images of these software packages to fix critical issues and vulnerabilities, but doesn't update software that you have already deployed**.** You even get direct access to partner support.

Demo

Launch Infrastructure Solutions on
Google Cloud Marketplace

Philipp Maier

Review:
Infrastructure Automation



In this module, we automated the deployment of infrastructure using Terraform, and looked into infrastructure solutions in Cloud Marketplace.

Now, you might say that going through all the effort to deploy a network, a firewall rule, and two VM instances doesn't convince you to use Terraform. That's true, if you only need to create these resources once, and don't foresee ever creating them again. However, for those of us who manage several resources and need to deploy, update, and destroy them in a repeatable way, an infrastructure automation tool like Terraform becomes essential.