Google Cloud

Guide to automating deployment with Terraform

So far, you've learned that a cloud cybersecurity team can use automation to streamline tasks like building and modifying infrastructure. **Infrastructure as code** (**IaC**) is the practice of automating and managing infrastructure using reusable scripts. Terraform is a popular IaC tool that lets you automate infrastructure deployment. In this reading, you'll find out how to use Terraform to configure and deploy Google Cloud resources using Cloud Shell.

Note: In the corresponding lab, you'll clone a Terraform example repository. Cloning the example repository creates a local copy of the Terraform directory and its contents in Cloud Shell, allowing you to access the files you need to deploy infrastructure.

Terraform configuration file

Terraform configurations are stored in a Terraform configuration file which are plain text files with a `.tf` file extension. A Terraform configuration tells Terraform how to manage a given collection of infrastructure resources. Terraform configurations are written in the Terraform language known as HashiCorp Configuration Language (HCL) which is a *declarative* language used to define infrastructure resources that you want to provision. As a cloud cybersecurity analyst, you can use HCL to define the desired state and configuration of the infrastructure, rather than outlining the steps required to reach the desired state.

Configuration syntax

The HCL configuration syntax was designed to be easy to read and write. There are many components to the syntax but you should be aware of two key components: *blocks* and *arguments*. Here's an example of a Terraform configuration file named `main.tf`. with code that you can refer to throughout this reading:

```
resource "google_compute_firewall" "default" {
  name    = "test-firewall"
  network = google_compute_network.default.name

  allow {
    protocol = "icmp"
  }

  allow {
```

```
    protocol = "tcp"
    ports    = ["80", "8080", "1000-2000"]
  }

  source_tags = ["web"]
}

resource "google_compute_network" "default" {
  name = "test-network"
}
```

Blocks

A block acts as a container for other content. Blocks represent the configuration of an object such as a firewall. Blocks are enclosed in curly brackets `{ }`, which contain arguments in key value pairs. A block has a type and each block type defines an amount of required labels such as a name. In the `main.tf` example, there are two `resource` blocks.

Resource blocks define the infrastructure resources to provision and sets the values of various arguments for that resource, such as a unique name. In the `main.tf` example, the objects are Google Cloud resources: a firewall `google_compute_firewall` with the name `default` and a virtual network `google_compute_network` with the name `default`.

Each resource is associated with a single resource type, which determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.

Note: Resource types differ and can require different arguments and attributes.

Arguments

Within the resource block itself is the configuration needed for the resource in the form of arguments. An argument assigns a value to a name. In the `main.tf` example, `name = "test-firewall"` and `network=google_compute_network.default.name` are examples of arguments. Within the block body—between `{` and `}`—are the configuration arguments for the resource itself. The arguments often depend on the resource type.

Terraform commands

Once you've declared your desired infrastructure using the configuration file, you can use the command-line interface to deploy the infrastructure using the `terraform` command. There are

a variety of subcommands that you can use, and you should be familiar with the following:

- `terraform init`: This command initializes a directory containing Terraform configuration files. This is the first command to use after creating a new Terraform configuration or cloning an existing configuration from version control. A directory must be initialized before Terraform can perform any operations in it.
- `terraform apply`: After initializing the directory, you can run other commands. This command allows you to preview and apply the actions in the execution plan. The execution plan outlines the actions that Terraform will make to the infrastructure according to what's outlined in the configuration file. When this command is run, you will be prompted to perform the actions. To perform the actions, you'll enter `yes`.

Key takeaways

Terraform is a powerful IaC tool that you can use to help safely and predictably create, change, and improve cloud infrastructure. As a cloud cybersecurity analyst, it's essential to understand how IaC tools such as Terraform can be used to deploy infrastructure within the cloud.

Resources for more information

If you'd like to learn more about Terraform, visit the following resources:

- To learn more about Terraform language, review Terraform language documentation
- To learn more about configuration syntax, investigate HCL native syntax specification
- For more Google Cloud configuration examples, explore Terraform Google examples
- For more information on configuring Google Cloud infrastructure with Terraform, visit Google Cloud Terraform provider