OCI & Terraform

Mimanshu Maheshwari

March 27, 2025

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 What is OCI?

Oracle Cloud Infrastructure (OCI) is a cloud computing service offered by Oracle that provides Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and Data as a Service (DaaS) solutions. It is designed to support high-performance computing, enterprise workloads, and cloud-native applications. OCI provides services such as compute, storage, networking, databases, security, and Artificial Intelligence (AI)/Machine Learning (ML) tools.

Chapter 2

Terraform

2.1 What is IaC?

Infrastructure as Code (IaC) tools allow you to manage infrastructure with configuration files rather than through a graphical user interface. IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share. Terraform is HashiCorp's infrastructure as code tool. It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle. Using Terraform has several advantages over manually managing your infrastructure:

- Terraform can manage infrastructure on multiple cloud platforms.
- The human-readable configuration language helps you write infrastructure code quickly.
- Terraform's state allows you to track resource changes throughout your deployments.
- You can commit your configurations to version control to safely collaborate on infrastructure.

2.2 Manage any infrastructure

Terraform plugins called providers let Terraform interact with cloud platforms and other services via their application programming interfaces (APIs). HashiCorp and the Terraform community have written over 1,000 providers to manage resources on Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, and DataDog, just to name a few. Find providers for many of the platforms and services you already use in the Terraform Registry. If you don't find the provider you're looking for, you can write your own.

2.3 Standardize your deployment workflow

Providers define individual units of infrastructure, for example compute instances or private networks, as resources. You can compose resources from different providers into reusable Terraform configurations called modules, and manage them with a consistent language and workflow. Terraform's configuration language is declarative, meaning that it describes the desired end-state for your infrastructure, in contrast to procedural programming languages that require step-by-step instructions to perform tasks. Terraform providers automatically calculate dependencies between resources to create or destroy them in the correct order.

To deploy infrastructure with Terraform(2.1):

- **Scope**: Identify the infrastructure for your project.
- **Author**: Write the configuration for your infrastructure.
- **Initialize**: Install the plugins Terraform needs to manage the infrastructure.

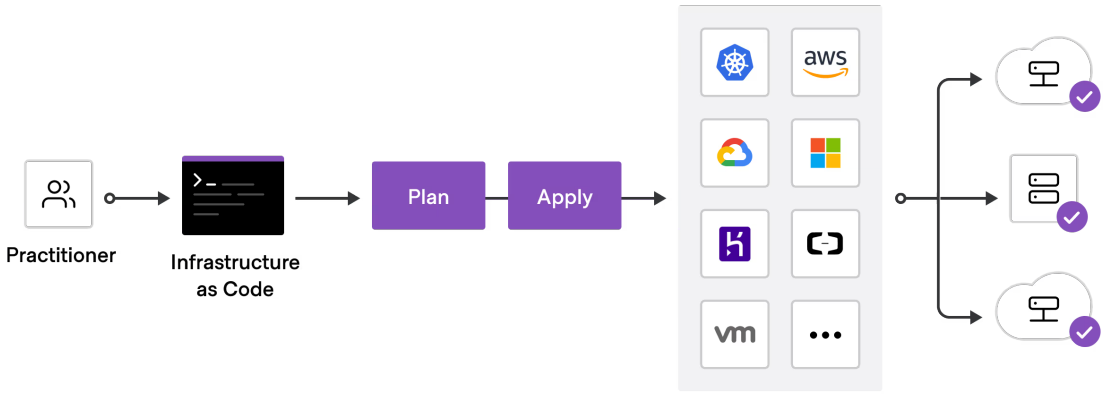

Figure 2.1: Terraform process flow

- **Plan**: Preview the changes Terraform will make to match your configuration.
- **Apply**: Make the planned changes.

2.4 Track your infrastructure

Terraform keeps track of your real infrastructure in a state file, which acts as a source of truth for your environment. Terraform uses the state file to determine the changes to make to your infrastructure so that it will match your configuration.

```
terraform {
  required_providers {
    docker ={
      source ="kreuzwerker/docker"
      version ="~> 3.0.2"
    }
  }
  required_version ="~> 1.7"
}
```

Listing 2.1: terraform.tf

This file includes the terraform block, which defines the provider and Terraform versions you will use with this project.

```
provider "docker" {}

resource "docker_image" "nginx" {
  name ="nginx:latest"
  keep_locally =false
}

resource "docker_container" "nginx" {
  image =docker_image.nginx.image_id
  name ="tutorial"
  ports {
    internal =80
    external =8000
  }
}
```

Listing 2.2: main.tf

```
terraform init
```

Provision the NGINX server container with apply. When Terraform asks you to confirm, type yes and press ENTER.

```
terraform apply
```

Run `docker ps` to view the NGINX container running in Docker via Terraform.

```
docker ps
```

```
terraform destroy
```

To stop the container and destroy the resources created in this tutorial, run `terraform destroy`. When Terraform asks you to confirm, type yes and press ENTER.

2.5 CLI Usage

Create a directory named `learn-terraform-docker-container`.

```
mkdir learn-terraform-docker-container
```

This working directory houses the configuration files that you write to describe the infrastructure you want Terraform to create and manage. When you initialize and apply the configuration here, Terraform uses this directory to store required plugins, modules (pre-written configurations), and information about the real infrastructure it created. Navigate into the working directory.

```
cd learn-terraform-docker-container
```

In the working directory, create a file called main.tf and paste the following Terraform configuration into it.

```

terraform {
  required_providers {
    docker ={
      source ="kreuzwerker/docker"
      version ="~> 3.0.1"
    }
  }
}

provider "docker" {
  # remove host for mac or linux
  host ="npipe:////.//pipe//docker_engine"
}

resource "docker_image" "nginx" {
  name ="nginx"
  keep_locally =false
}

resource "docker_container" "nginx" {
  image =docker_image.nginx.image_id
  name ="tutorial"
```

```
24
25    ports {
26      internal =80
27      external =8000
28    }
29  }
```

Listing 2.3: main.tf

Initialize the project, which downloads a plugin called a provider that lets Terraform interact with Docker.

```
1   terraform init
```

Provision the NGINX server container with apply. When Terraform asks you to confirm type yes and press ENTER.

```
1   terraform apply
```

Verify the existence of the NGINX container by visiting localhost:8000 in your web browser or running `docker ps` to see the container. To stop the container, run terraform destroy.

```
1   terraform destroy
```

You've now provisioned and destroyed an NGINX webserver with Terraform.

2.6 Build infrastructure

Deploy a Virtual Cloud Network (VCN) on OCI using Terraform. Other OCI resources need to deploy into a VCN.

2.6.1 Prerequisites

1. OCI Tenancy
2. The Terraform Command Line Interface (CLI) installed.
3. The OCI CLI installed.

```
1   oci session authenticate
2   # Enter a region by index or name(e.g.
3   # 1: ap-chiyoda-1, 2: ap-chuncheon-1, 3: ap-hyderabad-1, 4: ap-melbourne-1, 5: ap-mumbai-1,
4   # 6: ap-osaka-1, 7: ap-seoul-1, 8: ap-sydney-1, 9: ap-tokyo-1, 10: ca-montreal-1,
5   # 11: ca-toronto-1, 12: eu-amsterdam-1, 13: eu-frankfurt-1, 14: eu-zurich-1, 15: me-dubai-1,
6   # 16: me-jeddah-1, 17: sa-santiago-1, 18: sa-saopaulo-1, 19: uk-cardiff-1, 20: uk-gov-cardiff-1,
7   # 21: uk-gov-london-1, 22: uk-london-1, 23: us-ashburn-1, 24: us-gov-ashburn-1, 25: us-gov-
        ↪ chicago-1,
8   # 26: us-gov-phoenix-1, 27: us-langley-1, 28: us-luke-1, 29: us-phoenix-1, 30: us-sanjose-1):
```

Listing 2.4: Configure the OCI CLI from your terminal

Follow the prompts to enter the region where you have OCI tenancy. A browser window automatically opens and prompts you for your OCI user name and password. Enter them and click the "Sign In" button. Then return to your terminal. It displays a success message, which means that you have configured the OCI CLI with a default profile.

```
1   oci session authenticate
2   # Enter a region by index or name(e.g.
3   # 1: ap-chiyoda-1, 2: ap-chuncheon-1, 3: ap-hyderabad-1, 4: ap-melbourne-1, 5: ap-mumbai-1,
4   # 6: ap-osaka-1, 7: ap-seoul-1, 8: ap-sydney-1, 9: ap-tokyo-1, 10: ca-montreal-1,
5   # 11: ca-toronto-1, 12: eu-amsterdam-1, 13: eu-frankfurt-1, 14: eu-zurich-1, 15: me-dubai-1,
```

```
6   # 16: me-jeddah-1, 17: sa-santiago-1, 18: sa-saopaulo-1, 19: uk-cardiff-1, 20: uk-gov-cardiff-1,
7   # 21: uk-gov-london-1, 22: uk-london-1, 23: us-ashburn-1, 24: us-gov-ashburn-1, 25: us-gov-
        ↪ chicago-1,
8   # 26: us-gov-phoenix-1, 27: us-langley-1, 28: us-luke-1, 29: us-phoenix-1, 30: us-sanjose-1):
```

Follow the prompts to enter the region where you have OCI tenancy. A browser window automatically opens and prompts you for your OCI user name and password. Enter them and click the "Sign In" button. Then return to your terminal. It displays a success message and prompts you for a profile name. Enter learn-terraform for your profile name.

The output prints the location where the CLI has stored your token. Terraform will automatically detect the token later in the tutorial, and use the credentials there to create infrastructure. The token has a 1-hour Time To Live (TTL). If it expires, refresh it by providing the profile name.

```
1   oci session refresh --profile learn-terraform
2   # Attempting to refresh token from https://auth.us-sanjose-1.oraclecloud.com/v1/authentication/
        ↪ refresh
3   # Successfully refreshed token
```

2.6.2 Write configuration

- The set of files used to describe infrastructure in Terraform is known as a Terraform configuration.
- You will write your first configuration to define a single OCI VCN.
- Each Terraform configuration must be in its own working directory.
- Create a directory for your configuration.

```
1   mkdir learn-terraform-oci
```

```
1   cd learn-terraform-oci
```

```
1   touch main.tf
```

Open main.tf in your text editor and paste in the configuration below.

```
1   terraform {
2     required_providers {
3       oci ={
4         source ="oracle/oci"
5       }
6     }
7   }
8
9   provider "oci" {
10    region ="us-sanjose-1"
11    auth ="SecurityToken"
12    config_file_profile ="learn-terraform"
13  }
14
15  resource "oci_core_vcn" "internal" {
16    dns_label ="internal"
17    cidr_block ="172.16.0.0/20"
18    compartment_id ="<your_compartment_OCID_here>"
19    display_name ="My internal VCN"
20  }
```

Customize the following values:

- **region** - value should match your OCI region.
- **compartment_id** - value should match your "Oracle cloud ID (OCID)" which you can get by clicking on the profile icon in the far top right of the OCI console and selecting "Tenancy: YourUsername" from the dropdown menu.

Save the customized file. This is a complete configuration that you can deploy with Terraform. In the following sections, we will review each block of this configuration in more detail.

2.7 Terraform block

The "terraform {}" block contains Terraform settings, the required providers Terraform will use to provision your infrastructure. For each provider, the "source" attribute defines an optional "hostname", a "namespace", and the provider "type". Terraform installs providers from the Terraform Registry by default. In this example configuration, the oci provider's source is defined as oracle/oci, which is shorthand for **registry.terraform.io/oracle/oci**.

You can also set a version constraint for each provider defined in the `required_providers` block. The `version` attribute is optional, but we recommend using it to constrain the provider version so that Terraform does not install a version of the provider that does not work with your configuration. If you do not specify a provider version, Terraform will automatically download the most recent version during initialization.

More information at provider source documentation

2.8 Providers

The provider block configures the specified provider, in our case oci. A provider is a plugin that Terraform uses to create and mange your resources.

The `config\_file\_profile` attribute in the OCI provider block refers Terraform to the token credentials stored in the file that the OCI CLI created when you configured it. Never hard-code credentials or other secrets in your configuration files. Like other types of code, you may share and manage your Terraform configuration files using source control, so hard-coding secret values can expose them to attackers.

You can use multiple provider blocks in your Terraform configuration to manage resources from different providers. You can even use different providers together. For example, you could pass the Internet Protocol (IP) address of an OCI instance to a monitoring resource from DataDog.

2.9 Resources

Use resource blocks to define components of your infrastructure. A resource might be a physical or virtual component such as a VCN, or it can be a logical resource such as a Heroku application.

Resource blocks have two strings before the block: the **resource type** and the **resource name**. In this example, the resource type is "oci_core_vcn" and the name is "internal". The prefix of the type maps to the name of the provider, in this case, oci. Together, the resource type and name form Terraform's unique ID for the resource. For example, the ID of your VCN is "oci_core_vcn.internal"

Resource blocks contain arguments which you use to configure the resource. The example configuration contains arguments that set the Domain Name Server (DNS) label, the Classless Inter-Domain Routing (CIDR) block for the VCN, your OCID, and the display name. The OCI provider documentation documents the requried and optional arguments for each resource in the OCI provider.

2.10 Initialize the directory

```
terraform init

# Initializing the backend...

# Initializing provider plugins...
# - Finding latest version of oracle/oci...
# - Installing oracle/oci v5.7.0...
# - Installed oracle/oci v5.7.0 (signed by a HashiCorp partner, key ID 1533A49284137CEB)

# Terraform has created a lock file .terraform.lock.hcl to record the provider
# selections it made above. Include this file in your version control repository
# so that Terraform can guarantee to make the same selections by default when
# you run "terraform init" in the future.

# Terraform has been successfully initialized!

# You may now begin working with Terraform. Try running "terraform plan" to see
# any changes that are required for your infrastructure. All Terraform commands
# should now work.

# If you ever set or change modules or backend configuration for Terraform,
# rerun this command to reinitialize your working directory. If you forget, other
# commands will detect it and remind you to do so if necessary.
```

Terraform downloads the oci provider and installs it in a hidden subdirectory of your current working directory, named .terraform. The terraform init command prints out which version of the provider was installed. Terraform also creates a lock file named .terraform.lock.hcl which specifies the exact provider versions used, so that you can control when you want to update the providers used for your project.