OCI & Terraform

Mimanshu Maheshwari

March 27, 2025

Contents

List of Figures

Listings

Chapter 1

OCI

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 AI/ML tools.

1.2 What is realm in OCI?

A region in OCI is a geographically distinct area where OCI resources are deployed. Each region consists of multiple Availability Domains (ADs), ensuring high availability and disaster recovery. Examples of OCI regions include "us-ashburn-1" (US East), "uk-london-1" (UK), "ap-mumbai-1" (India), etc.

1.3 What is realm in OCI?

A realm in OCI is a collection of OCI regions that share the same identity namespace and authentication policies. Each OCI realm is isolated from others.

- Example: The commercial realm contains most OCI public regions, while the government realm includes regions meant for government customers.
- Common realms include:
  - oc1 → Commercial cloud
  - oc2 → Government cloud
  - oc3 → Dedicated Oracle Cloud

1.4 What is a physical domain and fault domain in OCI?

- Availability Domain (AD): A physically independent data center within an OCI region. It has its own power, cooling, and network. Some OCI regions have multiple ADs.
- Fault Domain (FD): A logical grouping of resources within an AD to protect against hardware failures. If a fault domain fails, only resources in that domain are affected.

Example: If a region has 3 Availability Domain, each AD has multiple Fault Domain to distribute workloads and avoid single points of failure.

1.5 What is tenancy in OCI?

A tenancy in OCI is a root-level account that represents an organization's cloud environment. It contains compartments, users, groups, and policies that control access to resources. Example: If a company has multiple departments, each department can have its own compartment within a single tenancy.

1.6 What is compartment in OCI?

A compartment is a logical container for OCI resources such as compute instances, storage, and databases. It allows for fine-grained access control using OCI Identity and Access Management (IAM) policies. Example: A company might create separate compartments for Development, Testing, and Production environments.

1.7 What is a group and dynamic group in OCI?

- Group: A collection of IAM users that share the same access policies.
- Dynamic Group: A special type of group that includes compute instances and other cloud resources instead of users.

Example: If you want all compute instances in a compartment to access Object Storage, you create a dynamic group and grant permissions via IAM policies.

1.8 How is authentication done in OCI?

OCI supports multiple authentication methods:

- Username & Password: Standard login via OCI Console
- Application Platform Interface (API) Key Authentication: Uses public-private key pairs for programmatic access
- Instance Principal Authentication: Allows compute instances to access resources without credentials
- Identity Federation: Integrates with Okta, Microsoft Azure AD (MAAD), and other Identity Provider (IdP)s for Single Sign On (SSO)
- Resource Principal Authentication: Allows OCI services to authenticate with other OCI services

1.9 Understanding IAM in OCI?

OCI Identity and Access Management (IAM) controls who can access what within OCI. Key IAM concepts include:

- Users → Individual accounts
- Groups → Collections of users with shared permissions
- Policies → Define permissions using a human-readable language
- Compartments → Logical containers for resources
- Dynamic Groups → Allow compute instances to have permissions

1.10 Using Java SDK to interact with OCI?

OCI provides a Java SDK to programmatically interact with its services. To use it:

1. Step 1: Add OCI SDK Dependency Maven

```
<dependency>
<groupId>com.oracle.oci.sdk</groupId>
<artifactId>oci-java-sdk-common</artifactId>
<version>3.0.0</version>
</dependency>
```

2. Step 2: Configure Authentication Use an API key or Instance Principals for authentication.

3. Step 3: Initialize the SDK & Make a Request

```
import com.oracle.bmc.auth.ConfigFileAuthenticationDetailsProvider;
import com.oracle.bmc.identity.IdentityClient;
import com.oracle.bmc.identity.requests.ListRegionsRequest;
import com.oracle.bmc.identity.responses.ListRegionsResponse;

public class OCIExample {
  public static void main(String[] args) throws Exception {
    ConfigFileAuthenticationDetailsProvider provider = new
        ↪ ConfigFileAuthenticationDetailsProvider("~/.oci/config");
    IdentityClient identityClient = new IdentityClient(provider);
    ListRegionsResponse response = identityClient.listRegions(ListRegionsRequest.
        ↪ builder().build());
    response.getItems().forEach(region -> System.out.println(region.getName()));
    identityClient.close();
  }
}
```

Listing 1.1: lists all OCI regions using SDK

```
import com.oracle.bmc.auth.ConfigFileAuthenticationDetailsProvider;
import com.oracle.bmc.core.ComputeClient;
import com.oracle.bmc.core.requests.ListInstancesRequest;
import com.oracle.bmc.core.responses.ListInstancesResponse;

public class OCITest {
  public static void main(String[] args) throws Exception {
    ConfigFileAuthenticationDetailsProvider provider = new
        ↪ ConfigFileAuthenticationDetailsProvider("~/.oci/config", "DEFAULT");
    ComputeClient computeClient = new ComputeClient(provider);
    ListInstancesRequest request = ListInstancesRequest.builder()
    .compartmentId("ocid1.compartment.oc1..example")
    .build();
    ListInstancesResponse response = computeClient.listInstances(request);
    response.getItems().forEach(instance -> System.out.println(instance.getDisplayName
        ↪ ()));
    computeClient.close();
  }
}
```

Listing 1.2: lists all compute instances in a specified compartment

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.

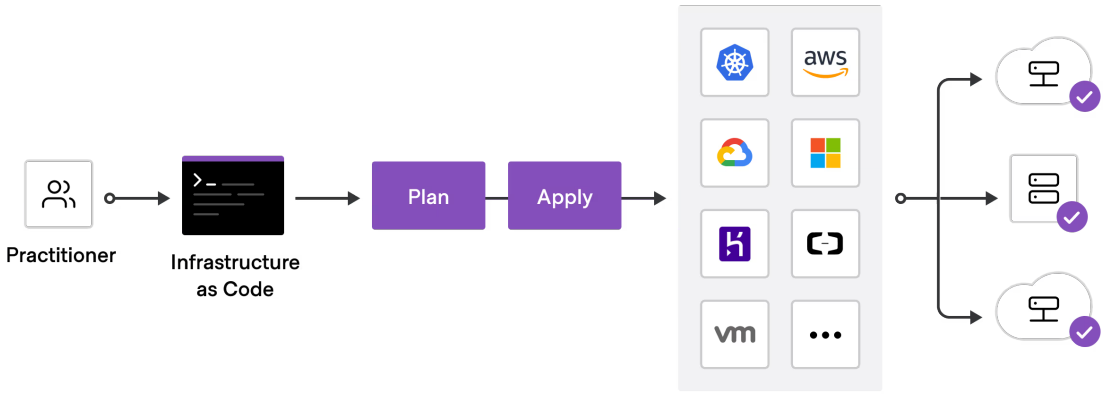

Figure 2.1: Terraform process flow

- **Initialize**: Install the plugins Terraform needs to manage the infrastructure.
- **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
  }
```

```
15  }
```

Listing 2.2: main.tf

```
1   terraform init
```

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

```
1   terraform apply
```

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

```
1   docker ps
```

```
1   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`.

```
1   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.

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

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

```
1
2   terraform {
3     required_providers {
4       docker ={
5         source ="kreuzwerker/docker"
6         version ="~> 3.0.1"
7       }
8     }
9   }
10
11  provider "docker" {
12    # remove host for mac or linux
13    host ="npipe:////.//pipe//docker_engine"
14  }
15
16  resource "docker_image" "nginx" {
17    name ="nginx"
18    keep_locally =false
19  }
20
21  resource "docker_container" "nginx" {
22    image =docker_image.nginx.image_id
```

```
23      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 Oracle Cloud Infrastructure (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.

2.11 Format and validate the configuration

We recommend using consistent formatting in all of your configuration files. The terraform fmt command automatically updates configurations in the current directory for readability and consistency. Format your configuration. Terraform will print out the names of the files it modified, if any. In this case, your configuration file was already formatted correctly, so Terraform won't return any file names.

```
terraform fmt
```

You can also make sure your configuration is syntactically valid and internally consistent by using the terraform validate command. Validate your configuration. The example configuration provided above is valid, so Terraform will return a success message.

```
terraform validate
```

2.12 Create infrastructure

Apply the configuration now with the terraform apply command. Terraform will print output similar to what is shown below. We have truncated some of the output to save space.

```
terraform apply

# Terraform used the selected providers to generate the following execution plan. Resource
    ↪ actions are indicated with the following symbols:
# + create
#
# Terraform will perform the following actions:
#
# # oci_core_vcn.internal will be created
# + resource "oci_core_vcn" "internal" {
  # + byoipv6cidr_blocks = (known after apply)
```

```
    # + cidr_block = "172.16.0.0/20"
    # + cidr_blocks = (known after apply)
    # + compartment_id = "ocid1.tenancy.oc1...."
    # + default_dhcp_options_id = (known after apply)
    # + default_route_table_id = (known after apply)
    # + default_security_list_id = (known after apply)
    # + defined_tags = (known after apply)
    # + display_name = "My internal VCN"
    # + dns_label = "internal"
    # + freeform_tags = (known after apply)
    # + id = (known after apply)
    # + ipv6cidr_blocks = (known after apply)
    # + ipv6private_cidr_blocks = (known after apply)
    # + is_ipv6enabled = (known after apply)
    # + is_oracle_gua_allocation_enabled = (known after apply)
    # + state = (known after apply)
    # + time_created = (known after apply)
    # + vcn_domain_name = (known after apply)
  # }
  #
  # Plan: 1 to add, 0 to change, 0 to destroy.
  #
  # Do you want to perform these actions?
  # Terraform will perform the actions described above.
  # Only 'yes' will be accepted to approve.
  #
  # Enter a value:
```

Before it applies any changes, Terraform prints out the execution plan which describes the actions Terraform will take in order to change your infrastructure to match the configuration. The output format is similar to the diff format generated by tools such as Git. The output has a + next to "oci_core_vcn.internal", meaning that Terraform will create this resource. Beneath that, it shows the attributes that will be set. When the value displayd is *knownafterapply*, it means that the value won't be known until the resource is created. For instance, OCI assigns its own ID when it creats the VCN, so Terraform can't know the value of the ID attribute until you apply the change and the OCI provider returns that vlaue from the OCI Application Platform Interface (API),

Terraform will n2wo pause and wait for you approval before proceeding. If anything int the plan seems incorrect or dangerous, it is safe to abort here with no changes made to you infrastructure.

In this case the plan is acceptable, so type yes at the confirmation prompt to proceed. Executing the plan will take a few minutes since Terraform waits for the VCN to become available. You've now created infrastructure using Terraform! Visit the OCI console VCN page, select your compartment, and find your new VCN, listed as "My internal VCN".

2.13 Inspect State

When you applied your configuration, Terraform wrote data into a file called terraform.tfstate. Terraform stores the IDs and properties of the resources it manages in this file, so that it can update or destroy those resources going forward.

The Terraform state file is the only way Terraform can track which resources it manages, and often contains sensitive information, so you must store your state file securely and distribute it only to trusted team members who need to manage your infrastructure. In production, we recommend storing your state remotely with HCP Terraform or Terraform Enterprise. Terraform also support servral other remote backends you can use to store and manage your state.

Inspect state using `terraform show`

```
terraform show
# # oci_core_vcn.internal:
# resource "oci_core_vcn" "internal" {
  # cidr_block = "172.16.0.0/20"
  # cidr_blocks = [
```

```
6       # "172.16.0.0/20",
7       # ]
8       # compartment_id = "ocid1.tenancy.oc1...."
9       # default_dhcp_options_id = "ocid1.dhcpoptions.oc1.us-sanjose-1.
            ↪ aaaaaaaa6odqyurw4mf7jmf3jy6ehtw6n32ohyogy4w5c43qoubgewyxr2va"
10      # default_route_table_id = "ocid1.routetable.oc1.us-sanjose-1.
            ↪ aaaaaaaan3n6iazjfubarvwwtszl3v6gdzqvfoccdj555p2ujehbo4tlu7ma"
11      # default_security_list_id = "ocid1.securitylist.oc1.us-sanjose-1.
            ↪ aaaaaaaant6vlu2y77pwwzjubmzg6czzvo2laii4h3p5d7w2nqcr4fey5gaa"
12      # defined_tags = {
13        # "Oracle-Tags.CreatedBy" = "oracleidentitycloudservice/redacted"
14        # "Oracle-Tags.CreatedOn" = "2021-04-07T18:25:06.555Z"
15      # }
16      # display_name = "My internal VCN"
17      # dns_label = "internal"
18      # freeform_tags = {}
19      # id = "ocid1.vcn.oc1.us-sanjose-1.
            ↪ amaaaaaapqqlmeyaklull6tpfms534aoijpjwpkzjo25rxqiqhadgdzodnua"
20      # ipv6cidr_blocks = []
21      # ipv6private_cidr_blocks = []
22      # is_ipv6enabled = false
23      # state = "AVAILABLE"
24      # time_created = "2021-04-07 18:25:06.558 +0000 UTC"
25      # vcn_domain_name = "internal.oraclevcn.com"
26    # }
```

When Terraform created this Virtual Cloud Network (VCN), it also gathered a lot of information about it from the OCI provider. These values can be referenced to configure other resources or outputs.

2.14 Manually managing state

Terraform has a built-in command called terraform state for advanced state management. For example, you may want a list of the resources in your project's state, which you can get by using the list subcommand.

```
1   terraform state list
2   # oci_core_vcn.internal
```