Oracle Cloud Infrastructure

Study Guide

Infrastructure as Code

Release 04/2018

ORACLE Certified Associate

ORACLE CLOUD INFRASTRUCTURE

Walter Goerner

Version: 04/18

Walter Goerner
Director Cloud Business Development Oracle EMEA

ORACLE

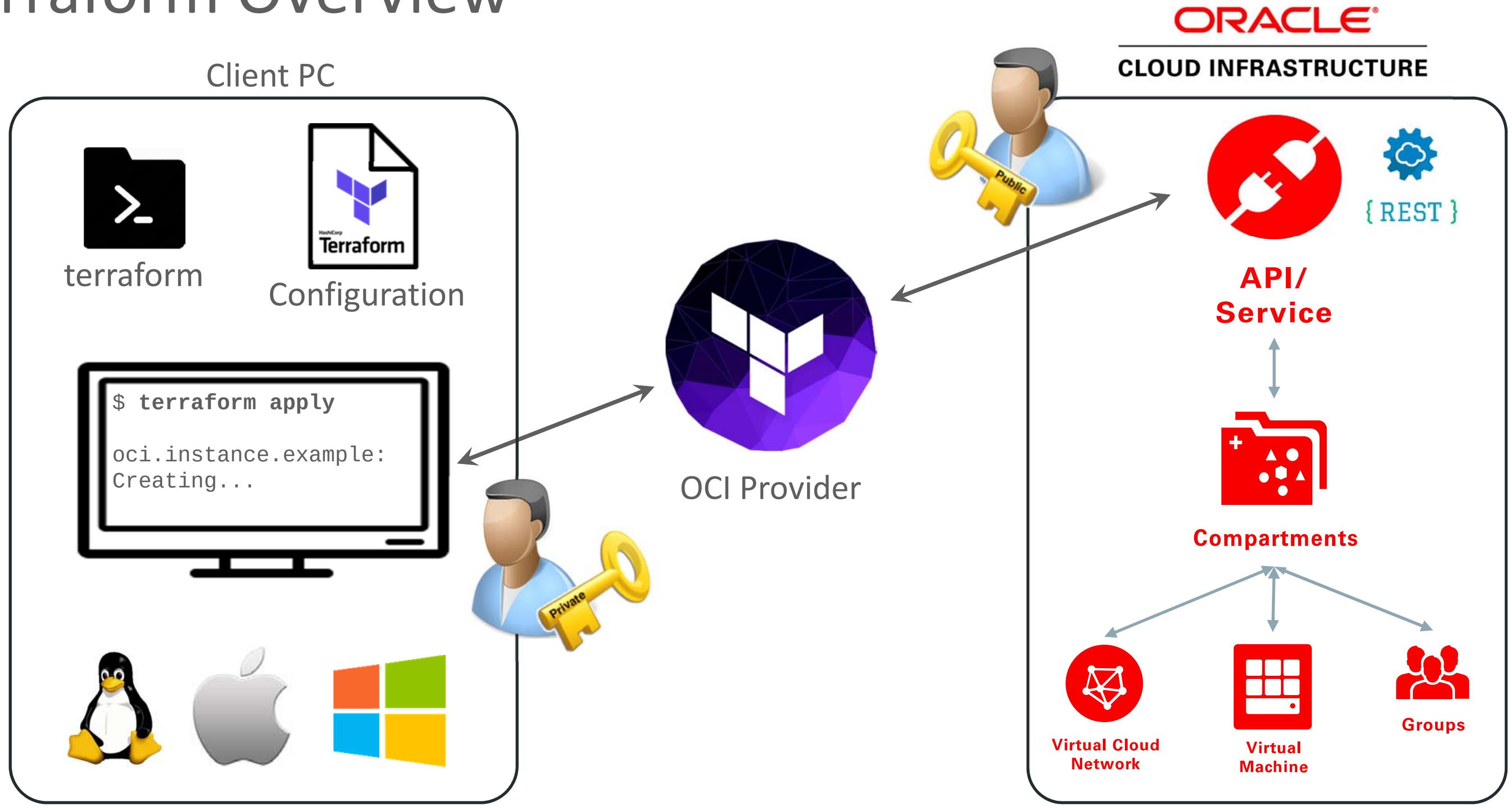
Terraform

Terraform

In this chapter we will cover the following topics:

- Terraform overview
- Terraform on OCI installation and configuration
- Terraform building blocks and examples
- Additional Terraform Examples
- Advanced Terraform Building Blocks

Terraform Overview

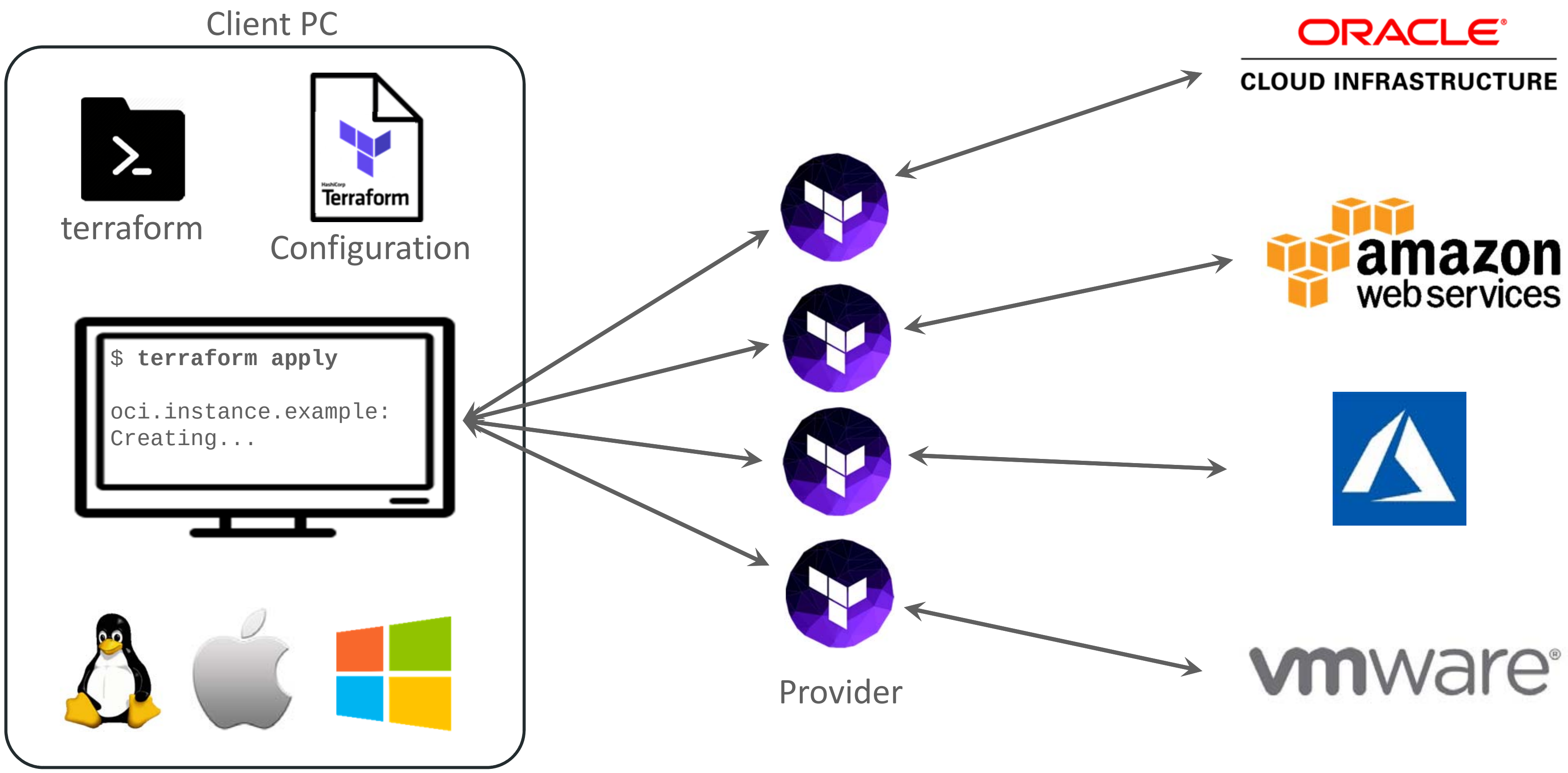


```
$ terraform apply

oci.instance.example:
Creating...
```

Terraform configuration file for creating an OCI user account (`1_user.tf`):

```
variable "tenancy_ocid" {}
variable "user_ocid" {}
variable "fingerprint" {}
variable "private_key_path" {}
variable "compartment_ocid" {}
variable "region" {}

provider "oci" {
tenancy_ocid     = "${var.tenancy_ocid}"
user_ocid        = "${var.user_ocid}"
fingerprint      = "${var.fingerprint}"
private_key_path = "${var.private_key_path}"
region           = "${var.region}"
}

resource "oci_identity_user" "user1" {
name        = "TFExampleUser"
description = "A user managed with Terraform"
}
```

Terraform Working with Multiple Providers

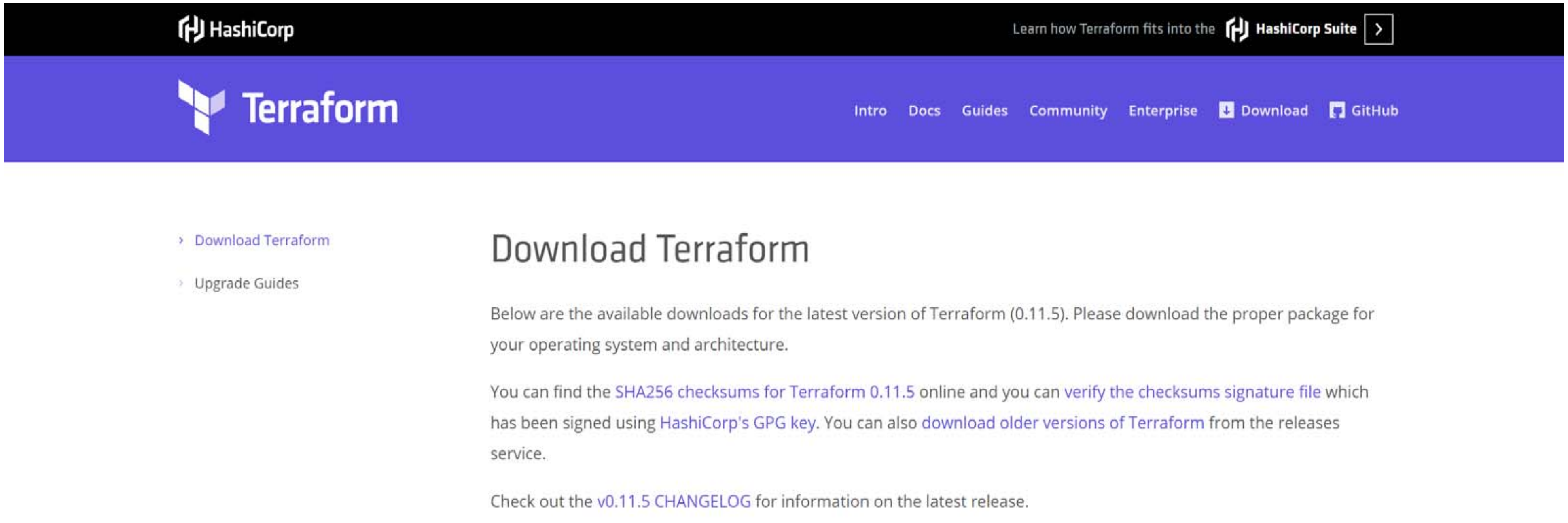


```
$ terraform apply

oci.instance.example:
Creating...
```

Terraform Installation Requirements

Requirement	Description
	An **Oracle Cloud Infrastructure account**, i.e. a tenancy together with a compartment.
	An **OCI user** created in that account, in a group with a policy that grants the desired permissions for the OCI resources to be managed. This account user can be or yourself, another person, or a system that calls the API.
	A **keypair** used for signing API requests, with the public key uploaded to Oracle. Only the user calling the API should possess the private key.
	Download the latest version of **Terraform** from its homepage https://www.terraform.io/ as well as the required providers either from GitHub or the Terraform page as well.

Download Terraform



Download Terraform

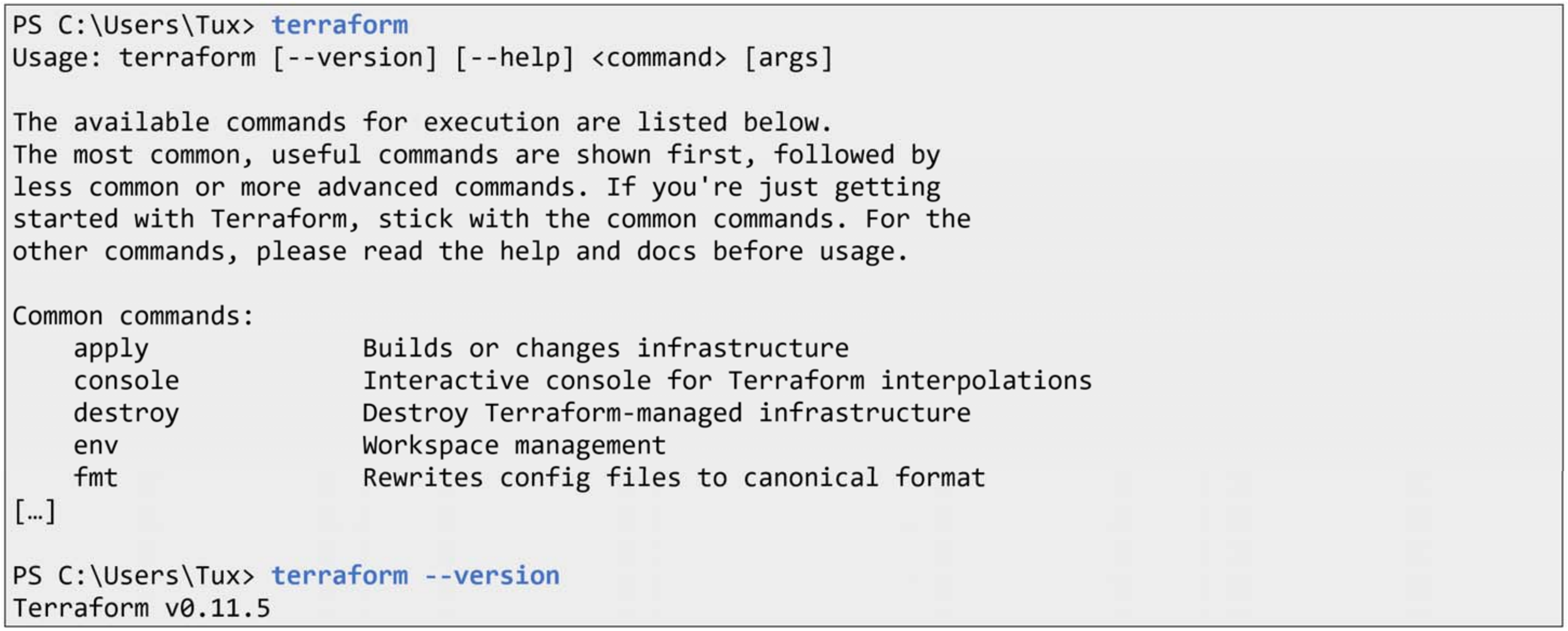
Below are the available downloads for the latest version of Terraform (0.11.5). Please download the proper package for your operating system and architecture.

You can find the SHA256 checksums for Terraform 0.11.5 online and you can verify the checksums signature file which has been signed using HashiCorp's GPG key. You can also download older versions of Terraform from the releases service.

Check out the v0.11.5 CHANGELOG for information on the latest release.

Download ZIP file, expand and place terraform.exe in PATH folder, e.g. `~/bin`.

Running Terraform for the First Time

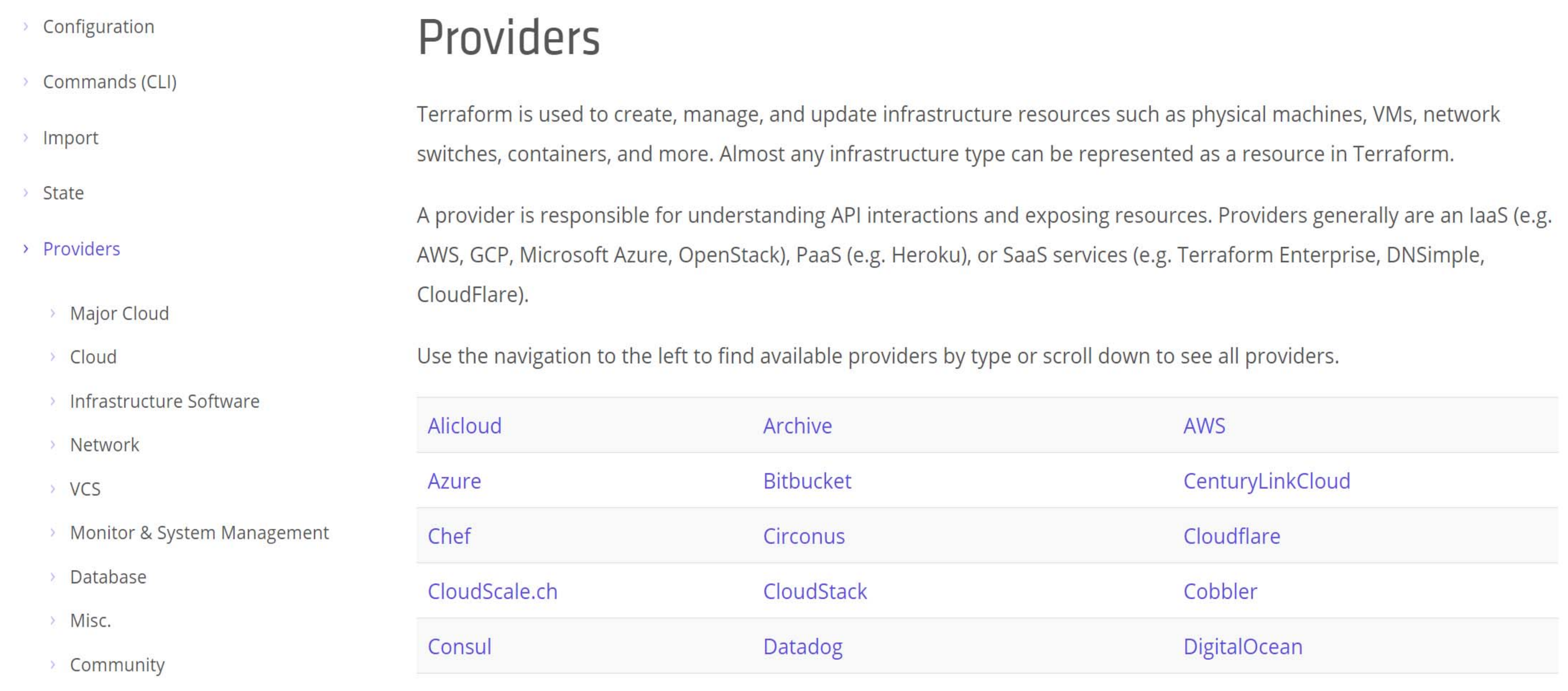


```
PS C:\Users\Tux> terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
    apply              Builds or changes infrastructure
    console            Interactive console for Terraform interpolations
    destroy            Destroy Terraform-managed infrastructure
    env                Workspace management
    fmt                Rewrites config files to canonical format
[…]

PS C:\Users\Tux> terraform --version
Terraform v0.11.5
```

Terraform Providers for Oracle Cloud

Provider	Homepage, Description
Oracle Cloud Platform Provider **“oraclepaas”**	https://www.terraform.io/docs/providers/oraclepaas/ Supports the Oracle Database Cloud Service as well as the Oracle Java Cloud Service.
Oracle Cloud Infrastructure Provider **“oci”**	https://github.com/oracle/terraform-provider-oci Supports the Oracle Cloud Infrastructure Services like compute, network, storage, etc.
Oracle Cloud Infrastructure Classic Provider **“opc”**	https://www.terraform.io/docs/providers/opc/ Supports Oracle Cloud Infrastructure Classic as well as Cloud at Customer Services.

Terraform Providers



- Configuration
- Commands (CLI)
- Import
- State
- Providers
  - Major Cloud
  - Cloud
  - Infrastructure Software
  - Network
  - VCS
  - Monitor & System Management
  - Database
  - Misc.
  - Community

Providers

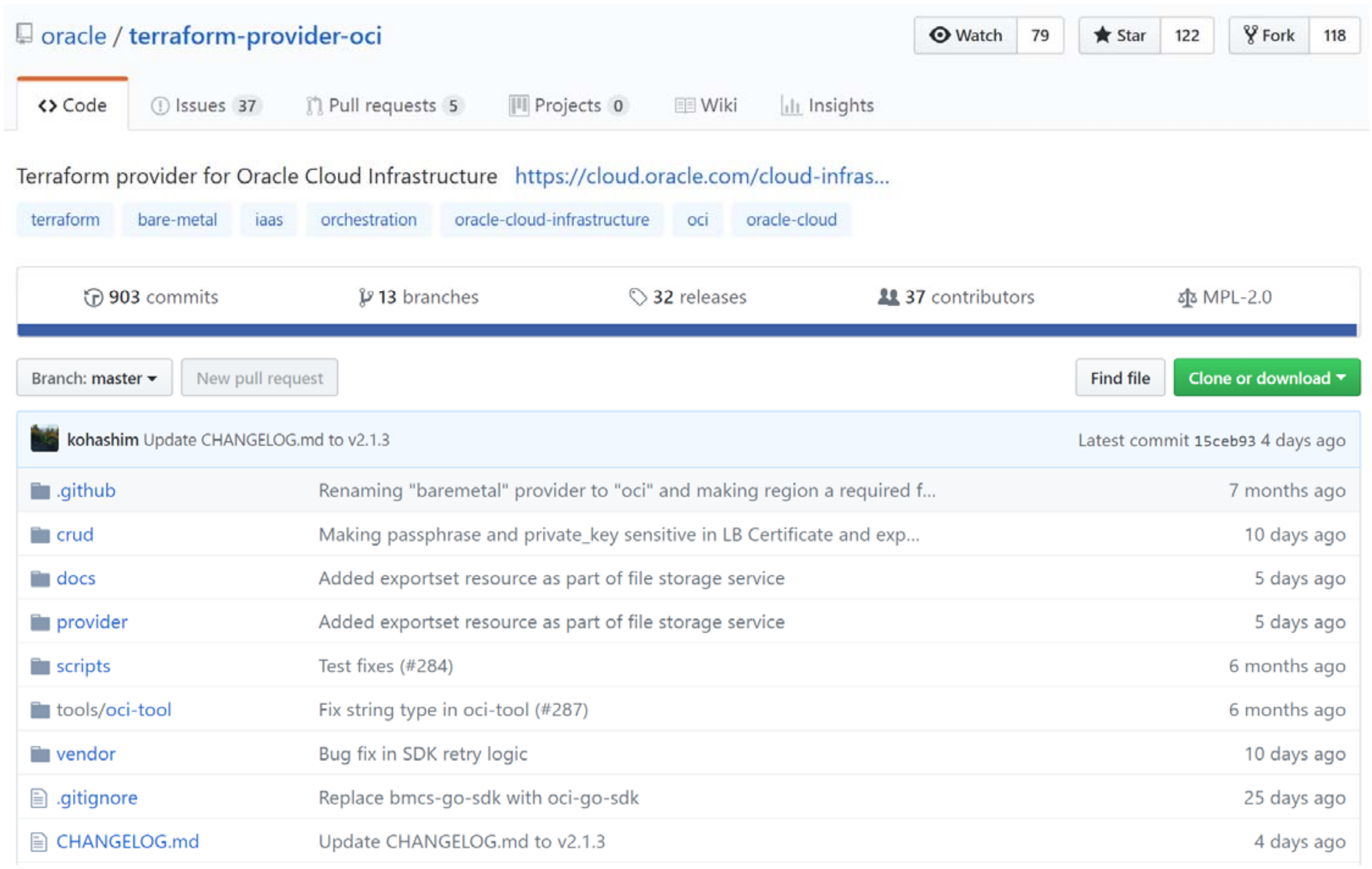
Terraform is used to create, manage, and update infrastructure resources such as physical machines, VMs, network switches, containers, and more. Almost any infrastructure type can be represented as a resource in Terraform.

A provider is responsible for understanding API interactions and exposing resources. Providers generally are an IaaS (e.g. AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Enterprise, DNSimple, CloudFlare).

Use the navigation to the left to find available providers by type or scroll down to see all providers.

Alicloud	Archive	AWS
Azure	Bitbucket	CenturyLinkCloud
Chef	Circonus	Cloudflare
CloudScale.ch	CloudStack	Cobbler
Consul	Datadog	DigitalOcean

OCI Provider Homepage



oracle / terraform-provider-oci

Watch 79 | Star 122 | Fork 118

Code | Issues 37 | Pull requests 5 | Projects 0 | Wiki | Insights

Terraform provider for Oracle Cloud Infrastructure https://cloud.oracle.com/cloud-infras...

terraform | bare-metal | iaas | orchestration | oracle-cloud-infrastructure | oci | oracle-cloud

903 commits | 13 branches | 32 releases | 37 contributors | MPL-2.0

Branch: master | New pull request | Find file | Clone or download

kohashim Update CHANGELOG.md to v2.1.3 — Latest commit 15ceb93 4 days ago

.github	Renaming "baremetal" provider to "oci" and making region a required f...	7 months ago
crud	Making passphrase and private_key sensitive in LB Certificate and exp...	10 days ago
docs	Added exportset resource as part of file storage service	5 days ago
provider	Added exportset resource as part of file storage service	5 days ago
scripts	Test fixes (#284)	6 months ago
tools/oci-tool	Fix string type in oci-tool (#287)	6 months ago
vendor	Bug fix in SDK retry logic	10 days ago
.gitignore	Replace bmcs-go-sdk with oci-go-sdk	25 days ago
CHANGELOG.md	Update CHANGELOG.md to v2.1.3	4 days ago

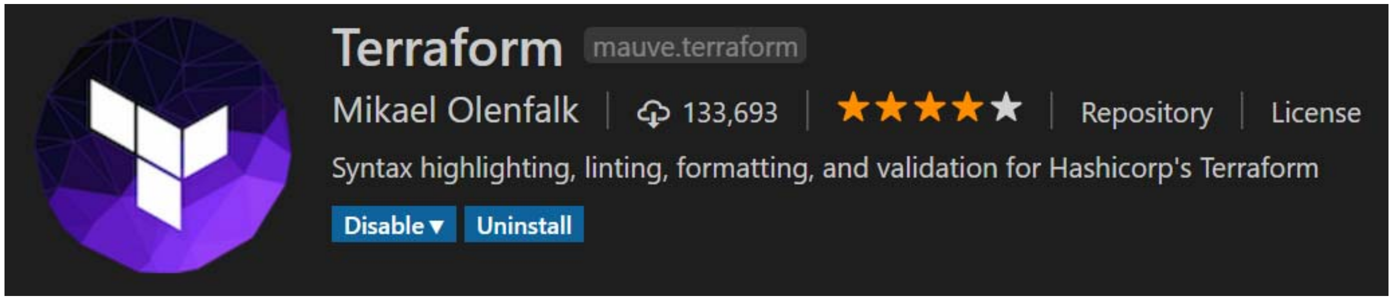
Terraform Provider Installation and Configuration

`terraform-provider-oci_v2.1.3`
copy to
`~/.terraform.d/plugins/`
 	
Environment Variables in `~/.bash_profile`:
export TF_VAR_tenancy_ocid=
export TF_VAR_user_ocid=
export TF_VAR_compartment_ocid=<root compartment)
export TF_VAR_fingerprint=
export TF_VAR_private_key_path=<fully qualified path>
export TF_VAR_region=
 	
Close and reopen your terminal session.
	`terraform-provider-oci_v2.1.3.exe`
	copy to
	`%APPDATA%/terraform.d/plugins/`
	
	Environment Variables in shell:
	setx TF_VAR_tenancy_ocid <value>
	setx TF_VAR_user_ocid <value>
	setx TF_VAR_compartment_ocid <value>
	setx TF_VAR_fingerprint <value>
	setx TF_VAR_private_key_path <value>
	setx TF_VAR_region <value>
	
	Close and reopen your terminal session.

Environment Variables on a Windows PC

```
setx TF_VAR_tenancy_ocid “ocid1.tenancy.oc1..aaaaaaaabzwjukrknofmqfti5mepdses4p7iwaubdck[…]”
setx TF_VAR_user_ocid “ocid1.user.oc1..aaaaaaaatcq7x4jerietj5q5vibs4ampx56wtgy2nacvcdkzmnjulqk6ejya”
setx TF_VAR_compartment_ocid “ocid1.compartment.oc1..aaaaaaaal7yma7iwruue2altr2cmgns4svg[…]”
setx TF_VAR_fingerprint “ba:4e:76:06:76:fd:28:7f:09:47:be:3f:50:a1:d8:00”
setx TF_VAR_private_key_path “C:\Users\Tux\.oci\oci_api_key_public.pem”
setx TF_VAR_region “eu-frankfurt-1”
```

VS Code Terraform Extension


Terraform mauve.terraform

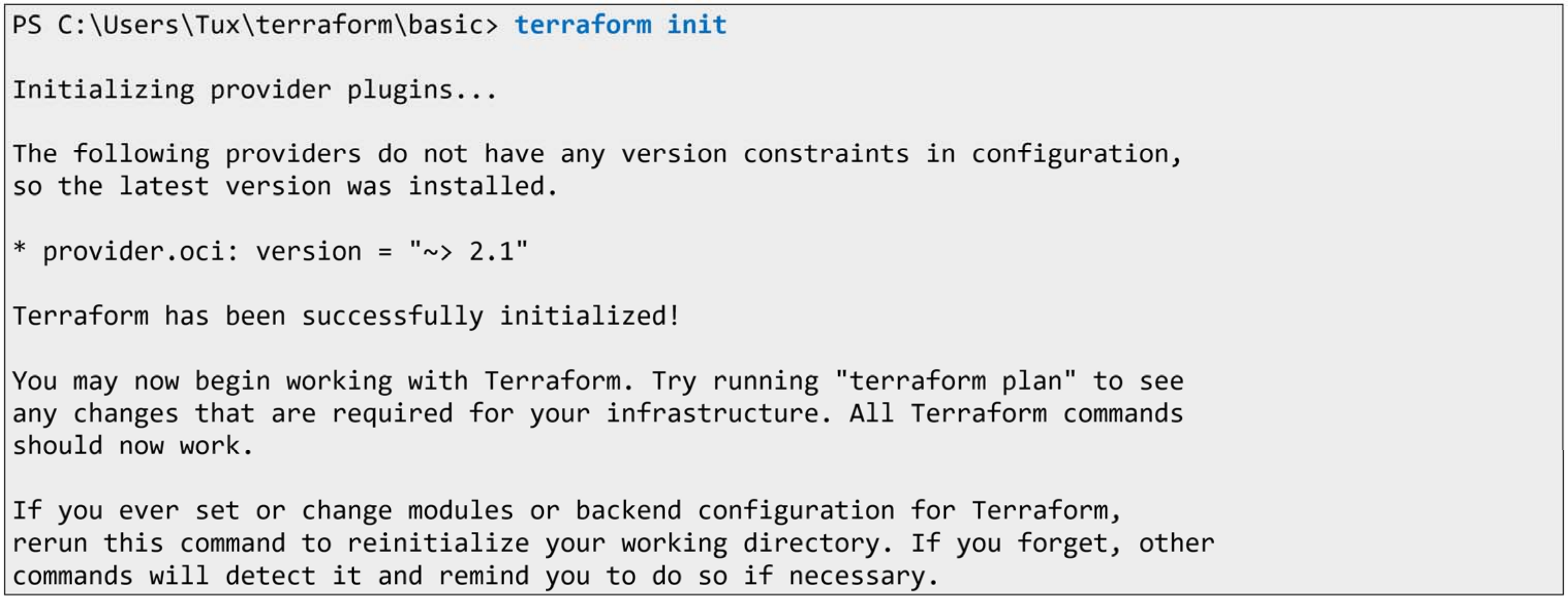
Mikael Olenfalk | 133,693 | ★★★★★ | Repository | License

Syntax highlighting, linting, formatting, and validation for Hashicorp's Terraform

Terraform Example Configuration VCN.TF

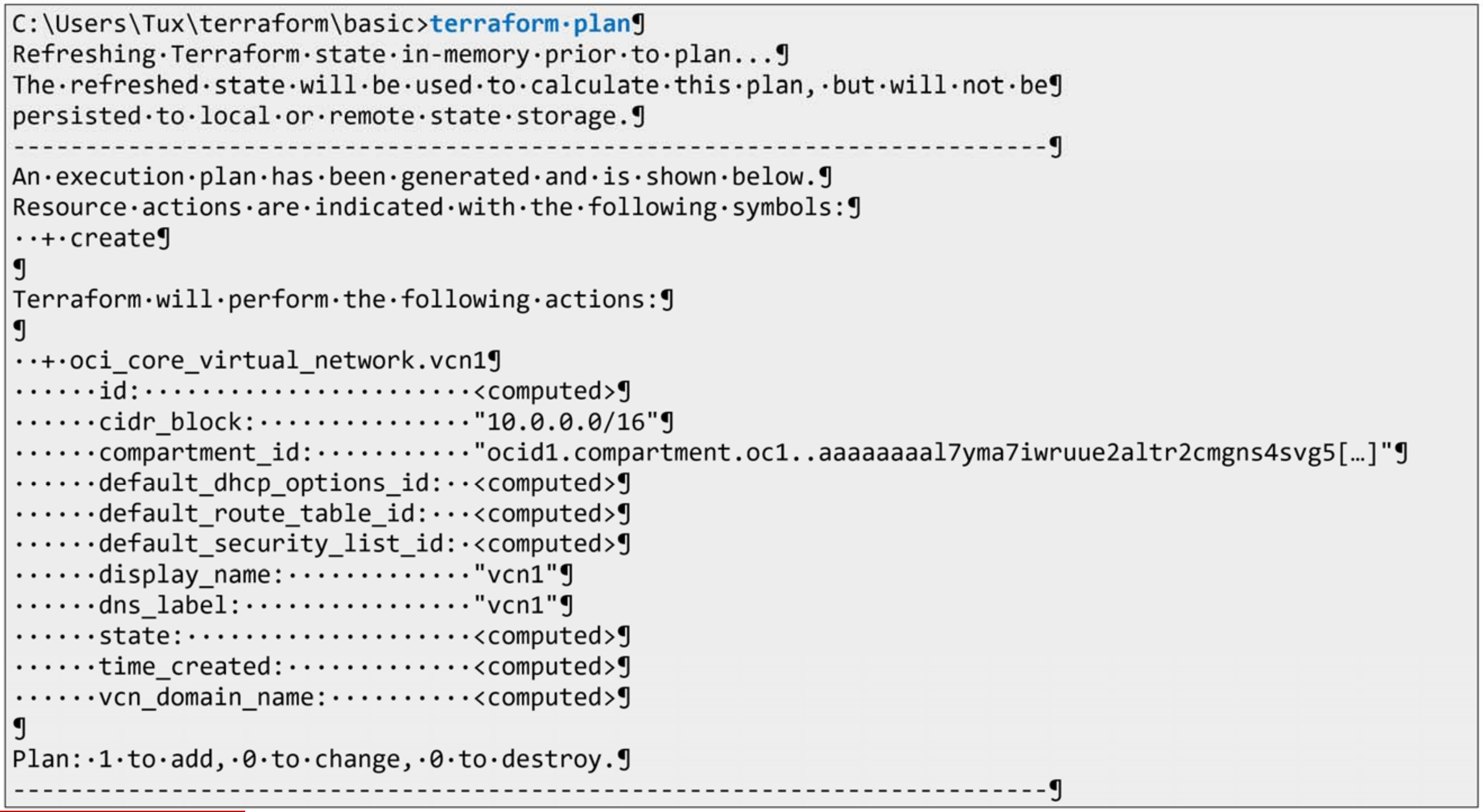
```
variable "tenancy_ocid" {}
variable "user_ocid" {}
variable "fingerprint" {}
variable "private_key_path" {}
variable "compartment_ocid" {}
variable "region" {}

provider "oci" {
tenancy_ocid     = "${var.tenancy_ocid}"
user_ocid        = "${var.user_ocid}"
fingerprint      = "${var.fingerprint}"
private_key_path = "${var.private_key_path}"
region           = "${var.region}"
}

resource "oci_core_virtual_network" "vcn1" {
cidr_block     = "10.0.0.0/16"
dns_label      = "vcn1"
compartment_id = "${var.compartment_ocid}"
display_name   = "vcn1"
}
```

terraform init



```
PS C:\Users\Tux\terraform\basic> terraform init

Initializing provider plugins...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

* provider.oci: version = "~> 2.1"

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 plan



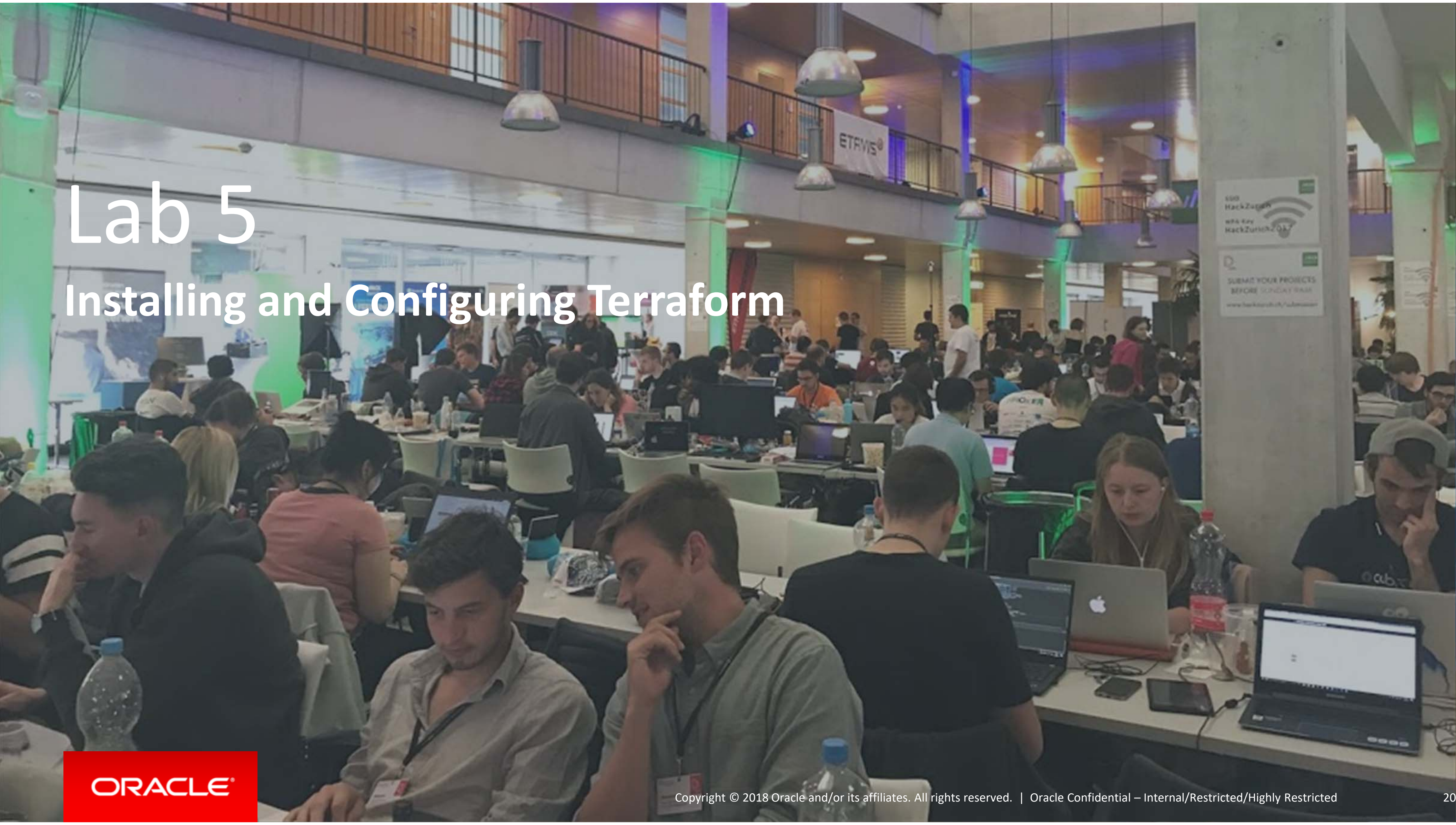
```
C:\Users\Tux\terraform\basic>terraform·plan¶
Refreshing·Terraform·state·in-memory·prior·to·plan...¶
The·refreshed·state·will·be·used·to·calculate·this·plan,·but·will·not·be¶
persisted·to·local·or·remote·state·storage.¶
------------------------------------------------------------------------¶
An·execution·plan·has·been·generated·and·is·shown·below.¶
Resource·actions·are·indicated·with·the·following·symbols:¶
··+·create¶
¶
Terraform·will·perform·the·following·actions:¶
¶
··+·oci_core_virtual_network.vcn1¶
······id:·······················<computed>¶
······cidr_block:···············"10.0.0.0/16"¶
······compartment_id:···········"ocid1.compartment.oc1..aaaaaaaal7yma7iwruue2altr2cmgns4svg5[…]"¶
······default_dhcp_options_id:··<computed>¶
······default_route_table_id:···<computed>¶
······default_security_list_id:·<computed>¶
······display_name:·············"vcn1"¶
······dns_label:················"vcn1"¶
······state:····················<computed>¶
······time_created:·············<computed>¶
······vcn_domain_name:··········<computed>¶
¶
Plan:·1·to·add,·0·to·change,·0·to·destroy.¶
------------------------------------------------------------------------¶
```

terraform apply

```
PS C:\Users\Tux\terraform\basic> terraform apply
[…]

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

oci_core_virtual_network.vcn1: Creating...
  cidr_block:               "" => "10.0.0.0/16"
  compartment_id:           "" => "ocid1.compartment.oc1..aaaaaaaal7yma7iwruue2altr2cmgns4svg5[…]"
  default_dhcp_options_id:  "" => "<computed>"
  default_route_table_id:   "" => "<computed>"
  default_security_list_id: "" => "<computed>"
  display_name:             "" => "vcn1"
  dns_label:                "" => "vcn1"
  state:                    "" => "<computed>"
  time_created:             "" => "<computed>"
  vcn_domain_name:          "" => "<computed>"
oci_core_virtual_network.vcn1: Creation complete after 0s (ID: ocid1.vcn.oc1.eu-frankfurt-1.aaa[…])

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

terraform destroy

```
PS C:\Users\Tux\terraform\basic> terraform destroy
oci_core_virtual_network.vcn1: Refreshing state... (ID: ocid1.vcn.oc1.eu-frankfurt-1.aaaaaa[…])

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - oci_core_virtual_network.vcn1

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

oci_core_virtual_network.vcn1: Destroying... (ID: ocid1.vcn.oc1.eu-frankfurt-
1.aaaaaaaadj...rcnyrd2c5ma6vqjpnt63rxbyt4pwxuondjvv5q)
oci_core_virtual_network.vcn1: Destruction complete after 4s

Destroy complete! Resources: 1 destroyed.
```



Lab 5

Installing and Configuring Terraform

ORACLE

Lab 5: Installing and Configuring Terraform

Objectives

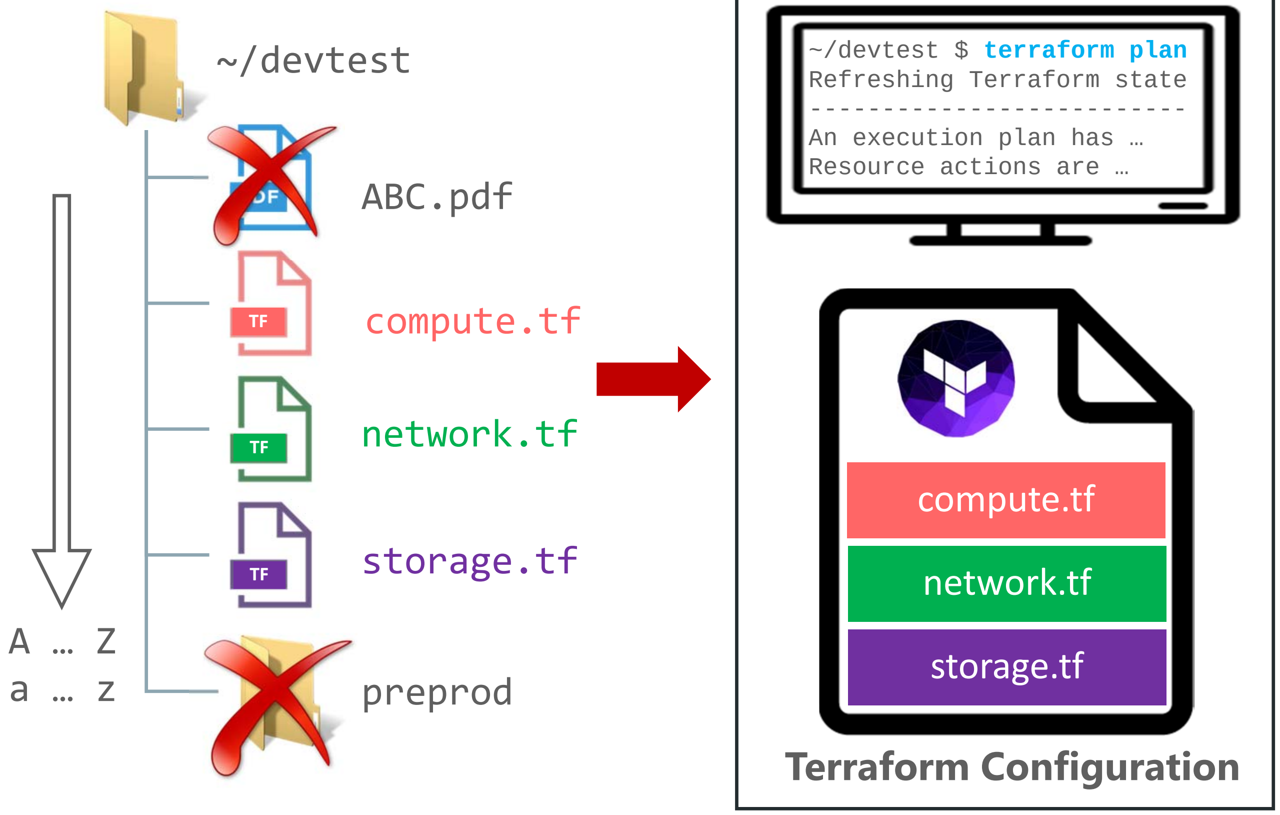
The objectives of this lab are to install and configure Terraform on your Computer also verifying basic Terraform functionality. The second main objective is to configure VS Code for Terraform usage.

Main tasks for this lab

- **Task 1:** Install and configure Terraform on your computer
- **Task 2:** Test basic Terraform functionality
- **Task 3:** Configure VS Code for Terraform language support

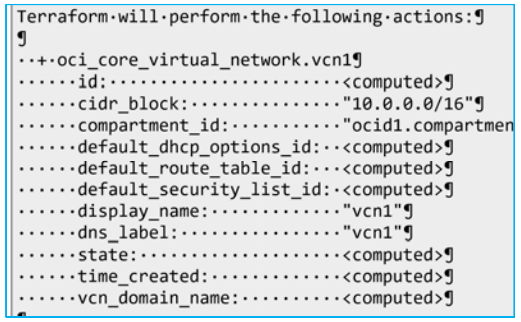
Estimated Time: 30 minutes

Terraform Configuration Build Process

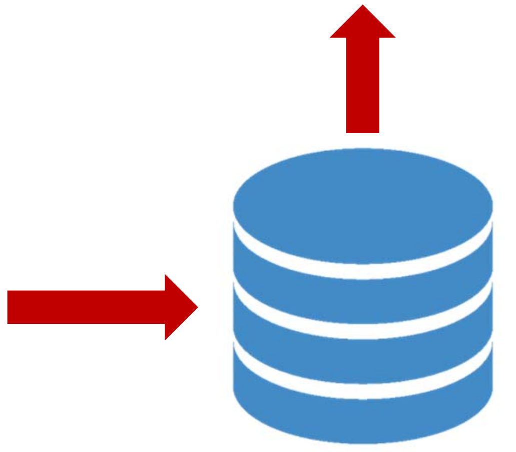


```
~/devtest $ terraform plan
Refreshing Terraform state
--------------------------
An execution plan has …
Resource actions are …
```

Execution Plan



```
Terraform·will·perform·the·following·actions:¶
¶
··+·oci_core_virtual_network.vcn1¶
······id:·······················<computed>¶
······cidr_block:···············"10.0.0.0/16"¶
······compartment_id:···········"ocid1.compartmen
······default_dhcp_options_id:··<computed>¶
······default_route_table_id:···<computed>¶
······default_security_list_id:·<computed>¶
······display_name:·············"vcn1"¶
······dns_label:················"vcn1"¶
······state:····················<computed>¶
······time_created:·············<computed>¶
······vcn_domain_name:··········<computed>¶
```

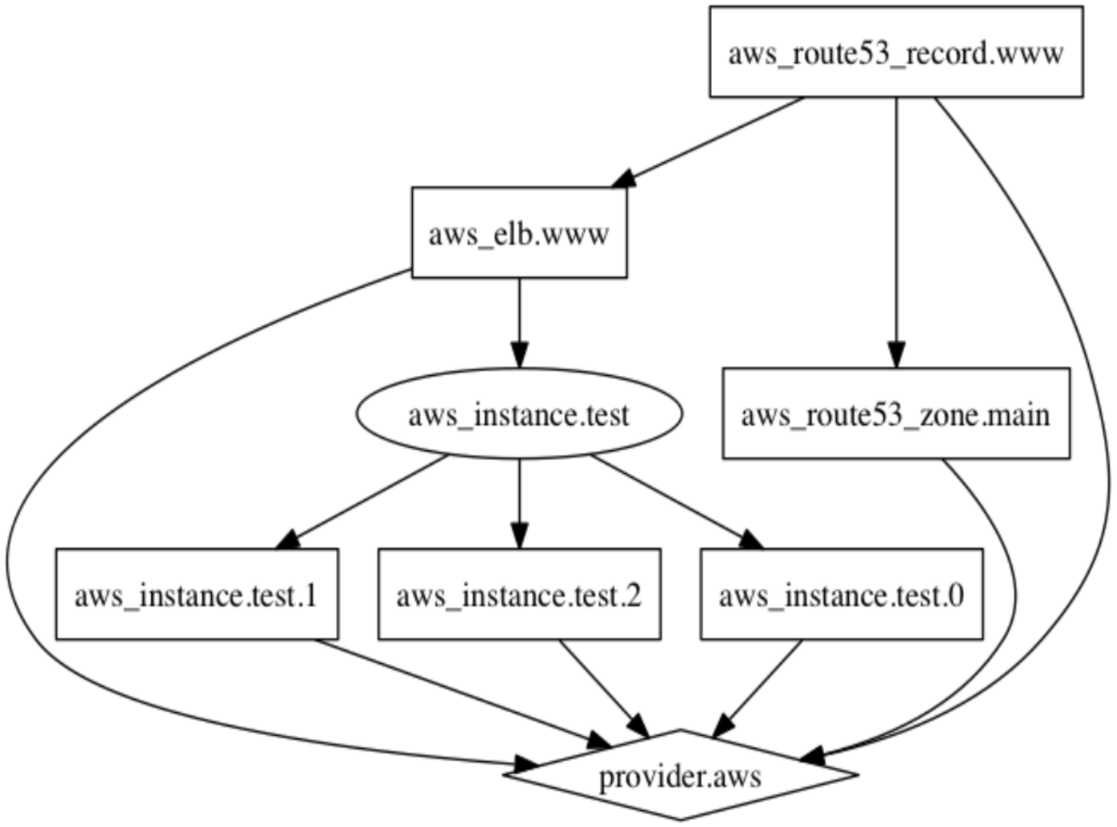


Terraform State

Procedural vs. Descriptive Configuration

Procedural	Descriptive
• The focus is on specifying the steps that produce the desired end-state (hopefully). • The code runs without context and state, i. e. it doesn’t care what is already installed or not. • Resource dependencies need to be coded by the programmer manually.	• The focus is on specifying the end state you want to achieve without specifying the individual steps. • There is a notion of context and state. • Resource dependencies need to be resolved by the provisioning system.
CHEF ANSIBLE	puppet labs SALTSTACK CloudFormation

Terraform Graph



Terraform State terraform.tfstate

```
{
    "version": 3,
    "terraform_version": "0.11.6",
    "serial": 6,
    "lineage": "4d255a76-5b98-6cf4-4f75-fa2248de2ea7",
    "modules": [
        {
            "path": [
                "root"
            ],
            "outputs": {},
            "resources": {
                "oci_core_virtual_network.vcn1": {
                    "type": "oci_core_virtual_network",
                    "depends_on": [],
                    "primary": {
                        "id": "ocid1.vcn.oc1.eu-frankfurt-1.aaaaaaaaw5qkx5gailsmnsbu5x[…]",
                        "attributes": {
                            "cidr_block": "10.0.0.0/16",
                            "compartment_id": "ocid1.compartment.oc1..aaaaaaaal7yma7iw[…]",
                            "default_dhcp_options_id": "ocid1.dhcpoptions.oc1.eu-frank[…]",
                            "default_route_table_id": "ocid1.routetable.oc1.eu-frankfu[…]",
                            "default_security_list_id": "ocid1.securitylist.oc1.eu-fra[…]",
                            "display_name": "vcn1",
                            "dns_label": "vcn1",
                            "id": "ocid1.vcn.oc1.eu-frankfurt-1.aaaaaaaaw5qkx5gailsmns[…]",
                            "state": "AVAILABLE",
                            "time_created": "2018-04-14 11:09:06.658 +0000 UTC",
                            "vcn_domain_name": "vcn1.oraclevcn.com"
                        },
[…]
```

Using Terraform

```
PS C:\Users\Tux\terraform\basic> terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

oci_core_virtual_network.vcn1: Refreshing state... (ID: ocid1.vcn.oc1.eu-frankfurt-
1.aaaaaaaaw5...bu5xqliere4yhoj4njulbxqrvxdksem5jtk6sa)

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

```
resource "oci_core_virtual_network" "vcn1" {
  cidr_block     = "10.0.0.0/16"
  dns_label      = "vcn1"
  compartment_id = "${var.compartment_ocid}"
  display_name   = "vcn1"

  count          = 2
}
```

Terraform Examples - Create an OCI User Account (1)

https://github.com/oracle/terraform-provider-oci/blob/master/docs/examples/iam/user/user.tf

Create a new sub-directory called “`examples`” in your `~/terraform` folder and download the above file from GitHub into the examples folder:

```
variable "tenancy_ocid" {}
variable "user_ocid" {}
variable "fingerprint" {}
variable "private_key_path" {}
variable "compartment_ocid" {}
variable "region" {}

provider "oci" {
    tenancy_ocid     = "${var.tenancy_ocid}"
    user_ocid        = "${var.user_ocid}"
    fingerprint      = "${var.fingerprint}"
    private_key_path = "${var.private_key_path}"
    region           = "${var.region}"
}
```

Terraform Examples - Create an OCI User Account (2)

```
resource "oci_identity_user" "user1" {
    name        = "TFExampleUser"
    description = "A user managed with Terraform"
}

resource "oci_identity_ui_password" "tf_password" {
    user_id = "${oci_identity_user.user1.id}"
}

output "UserUIPassword" {
    sensitive = false
    value     = ["${oci_identity_ui_password.tf_password.password}"]
}
```

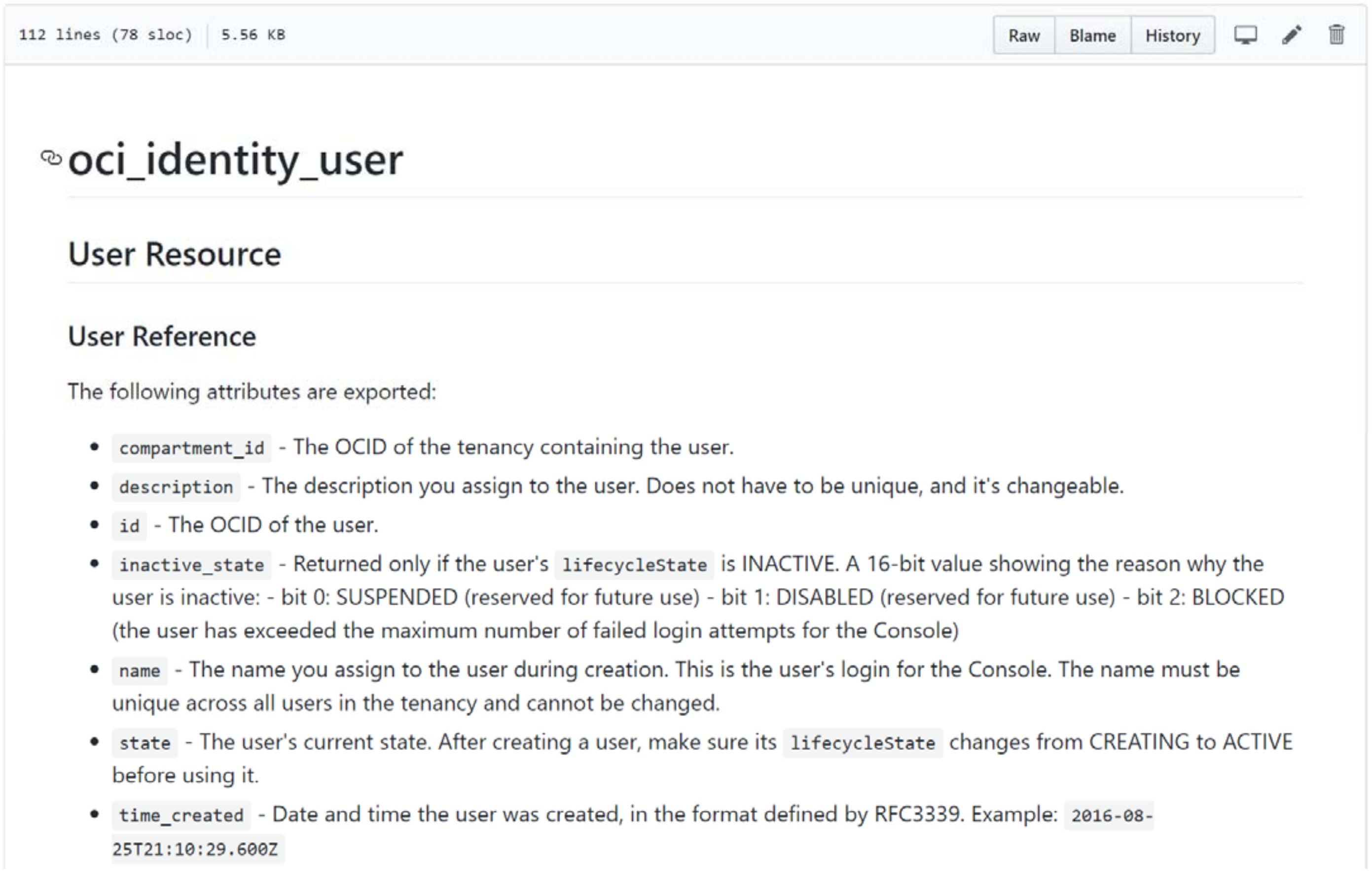
Terraform Environment Variables

```
setx TF_VAR_user_ocid “ocid1.user.oc1..aaaaaaaatcq7x4jerietj5q5vibs4amp[…]”
```

```
variable "tenancy_ocid" {}
```

```
    tenancy_ocid       = "${var.tenancy_ocid}"
```

OCI Terraform Resources (1)



oci_identity_user

User Resource

User Reference

The following attributes are exported:

- `compartment_id` - The OCID of the tenancy containing the user.
- `description` - The description you assign to the user. Does not have to be unique, and it's changeable.
- `id` - The OCID of the user.
- `inactive_state` - Returned only if the user's `lifecycleState` is INACTIVE. A 16-bit value showing the reason why the user is inactive: - bit 0: SUSPENDED (reserved for future use) - bit 1: DISABLED (reserved for future use) - bit 2: BLOCKED (the user has exceeded the maximum number of failed login attempts for the Console)
- `name` - The name you assign to the user during creation. This is the user's login for the Console. The name must be unique across all users in the tenancy and cannot be changed.
- `state` - The user's current state. After creating a user, make sure its `lifecycleState` changes from CREATING to ACTIVE before using it.
- `time_created` - Date and time the user was created, in the format defined by RFC3339. Example: `2016-08-25T21:10:29.600Z`

OCI Terraform Resources (2)

We need to pass the OCID of the newly created user resource as an argument, in other words, we need to interpolate the attribute of another resource, the OCID of our `user1` resource. The general syntax for doing so is as follows:

```
${TYPE.NAME.ATTRIBUTE}
```

In our example the type is `oci_identity_user`, the name of the resource is `user1`, and the attribute we are interested in is the `id`. With that, the user gets a random password assigned that needs to be changed at next logon. With that you also introduced a **dependency** into the configuration, the password resource is dependent on the user resource, without a user there is also no need for a password.

Create a Linux Server with Multiple VNICs (1)

The configuration file is called **`2_instance.tf`** and can be found on the book's GitHub page. Download the file and store it in a new folder called "`MultiNIC`":

```
1   # Adapted from:
2   # https://github.com/oracle/terraform-provider-oci/blob/master/docs/examples/compute/multi_vnic/multi_vnic.tf
3
4   # The usual variable declaration
5   variable "tenancy_ocid" {}
6   variable "user_ocid" {}
7   variable "fingerprint" {}
8   variable "private_key_path" {}
9   variable "compartment_ocid" {}
10  variable "region" {}
11
12  # A public key needed for ssh'ing into the VM
13  # This needs to be adapted to your environment!!! Go to line 95 and change the path there!
```

Create a Linux Server with Multiple VNICs (2)

In lines 31 to 41 a new variable type **“map”** is introduced for **“instance_image_ocid”**:

```
30  # OCIDs for Oracle-provided Linux 7.4 images per region
31  variable "instance_image_ocid" {
32  type = "map"
33
34  default = {
35  // See https://docs.us-phoenix-1.oraclecloud.com/Content/Resources/Assets/OracleProvidedImageOCIDs.pdf
36  // Oracle-provided image "Oracle-Linux-7.4-2018.02.21-1"
37  us-phoenix-1   = "ocid1.image.oc1.phx.aaaaaaaaupbfz5f5hdvejulmalhyb6goieolullgkpumorbvxlwkaowglslq"
38  us-ashburn-1   = "ocid1.image.oc1.iad.aaaaaaaajlw3xfie2t5t52uegyhiq2npx7bqyu4uvi2zyu3w3mqayc2bxmaa"
39  eu-frankfurt-1 = "ocid1.image.oc1.eu-frankfurt1.aaaaaaaa7d3fsb6272srnftyi4dphdgfjf6gurxqhmv6ileds7ba3m2gltxq"
40  uk-london-1    = "ocid1.image.oc1.uk-london1.aaaaaaaaa6h6gj6v4n56mqrbgnosskq63blyv2752g36zerymy63cfkojiiq"
41  }
42  }
```

```
image = "${var.instance_image_ocid[var.region]}"
```

First of all, Terraform will interpolate the square brackets `[var.region]` resulting in “`eu-frankfurt-1`”. It will then interpolate the curly brackets by looking up which key/value pair in our map variable has the value of “`eu-frankfurt-1`”. The resulting OCID will then be assigned to the image attribute as part of the VM definition. Terraform tends to be pretty elegant and efficient.

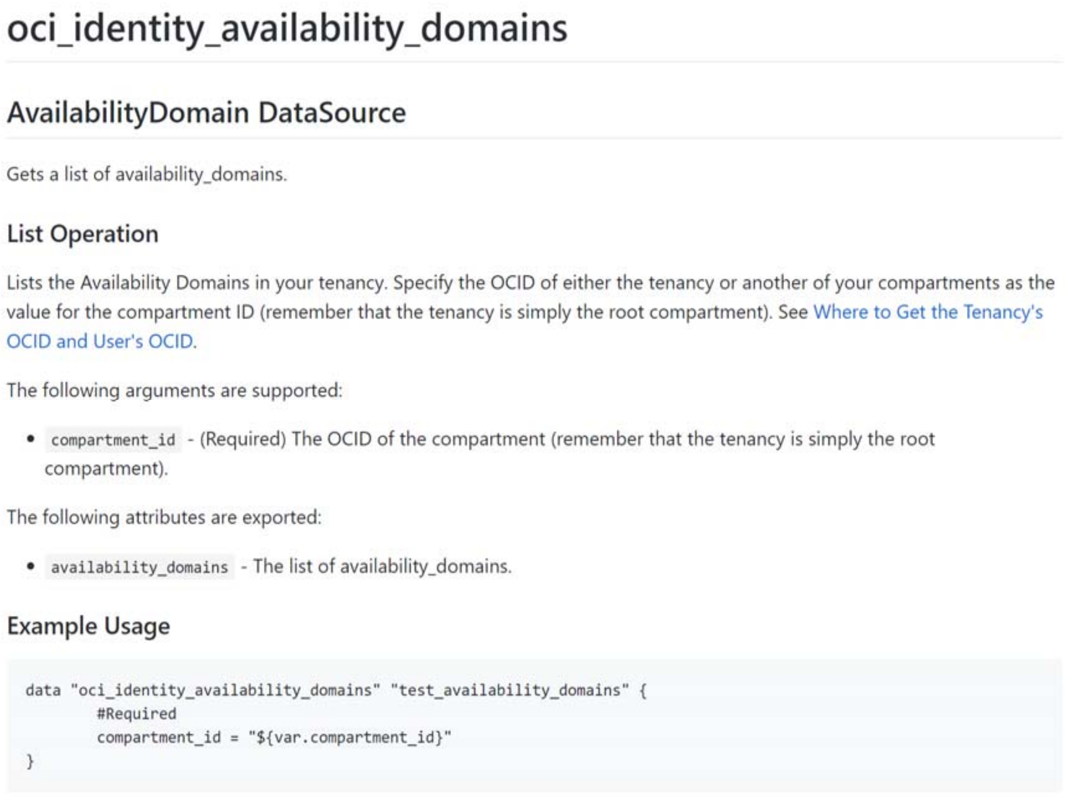
Create a Linux Server with Multiple VNICs (3)

Lines 44 to 51 are implementing the `oci` provider initialization which we discussed already, no news in here. Lines 53 to 56 implement a new Terraform concept called a **data source**:

```
53 # Populate oci availability domains for our tenancy
54 data "oci_identity_availability_domains" "ADs" {
55 compartment_id = "${var.tenancy_ocid}"
56 }
```

Data sources allow data to be fetched or computed for use elsewhere in a Terraform configuration. Use of data sources allows a Terraform configuration to build on information defined outside of Terraform, or defined by another separate Terraform configuration. *Providers* are responsible in Terraform for defining and implementing data sources. Whereas a *resource* causes Terraform to create and manage a new infrastructure component, *data sources* present **read-only views** into pre-existing data, or they compute new values on the fly within Terraform itself.

Create a Linux Server with Multiple VNICs (4)



oci_identity_availability_domains

AvailabilityDomain DataSource

Gets a list of availability_domains.

List Operation

Lists the Availability Domains in your tenancy. Specify the OCID of either the tenancy or another of your compartments as the value for the compartment ID (remember that the tenancy is simply the root compartment). See Where to Get the Tenancy's OCID and User's OCID.

The following arguments are supported:

- `compartment_id` - (Required) The OCID of the compartment (remember that the tenancy is simply the root compartment).

The following attributes are exported:

- `availability_domains` - The list of availability_domains.

Example Usage

```
data "oci_identity_availability_domains" "test_availability_domains" {
	#Required
	compartment_id = "${var.compartment_id}"
}
```

Looking at GitHub below gives us more details around the AvailabilityDomain data source (https://github.com/oracle/terraform-provider-oci/blob/master/docs/identity/availability_domains.md):

Create a Linux Server with Multiple VNICs (5)

```
PS C:\Users\Tux> oci iam availability-domain list -c $TEN
{
  "data": [
    {
      "compartment-id": "ocid1.tenancy.oc1..aaaaaaaabzwjukrknofmqfti5mepdses4p7i[…]",
      "name": "qSIW:EU-FRANKFURT-1-AD-1"
    },
    {
      "compartment-id": "ocid1.tenancy.oc1..aaaaaaaabzwjukrknofmqfti5mepdses4p7i[…]",
      "name": "qSIW:EU-FRANKFURT-1-AD-2"
    },
    {
      "compartment-id": "ocid1.tenancy.oc1..aaaaaaaabzwjukrknofmqfti5mepdses4p7i[…]",
      "name": "qSIW:EU-FRANKFURT-1-AD-3"
    }
  ]
}
```

ORACLE

Create a Linux Server with Multiple VNICs (6)

```
"${lookup(data.oci_identity_availability_domains.ADs.availability_domains[var.AD - 1],"name")}"
```

The idea again is pretty straight forward: We want to provision a subnet in a region and need to know the name of the AD in that region where we want to provision the subnet in (in our case it's the first AD). That's why we specified the data source `"instance_image_ocid"` to contain all AD names for our region. However, we already looked at the contents of the data source and for actually resolving the AD name we need some help; and that's where **`lookup`** comes into the play.

`lookup(map, key, [default])` - Performs a dynamic lookup into a map variable. The `map` parameter should be another variable, such as `var.amis`. If `key` does not exist in map, the interpolation will fail unless you specify a third argument, `default`, which should be a string value to return if no key is found in map. This function only works on flat maps and will return an error for maps that include nested lists or maps.

With that knowledge let's look at line 68 again: The interpolation syntax to access a data source is

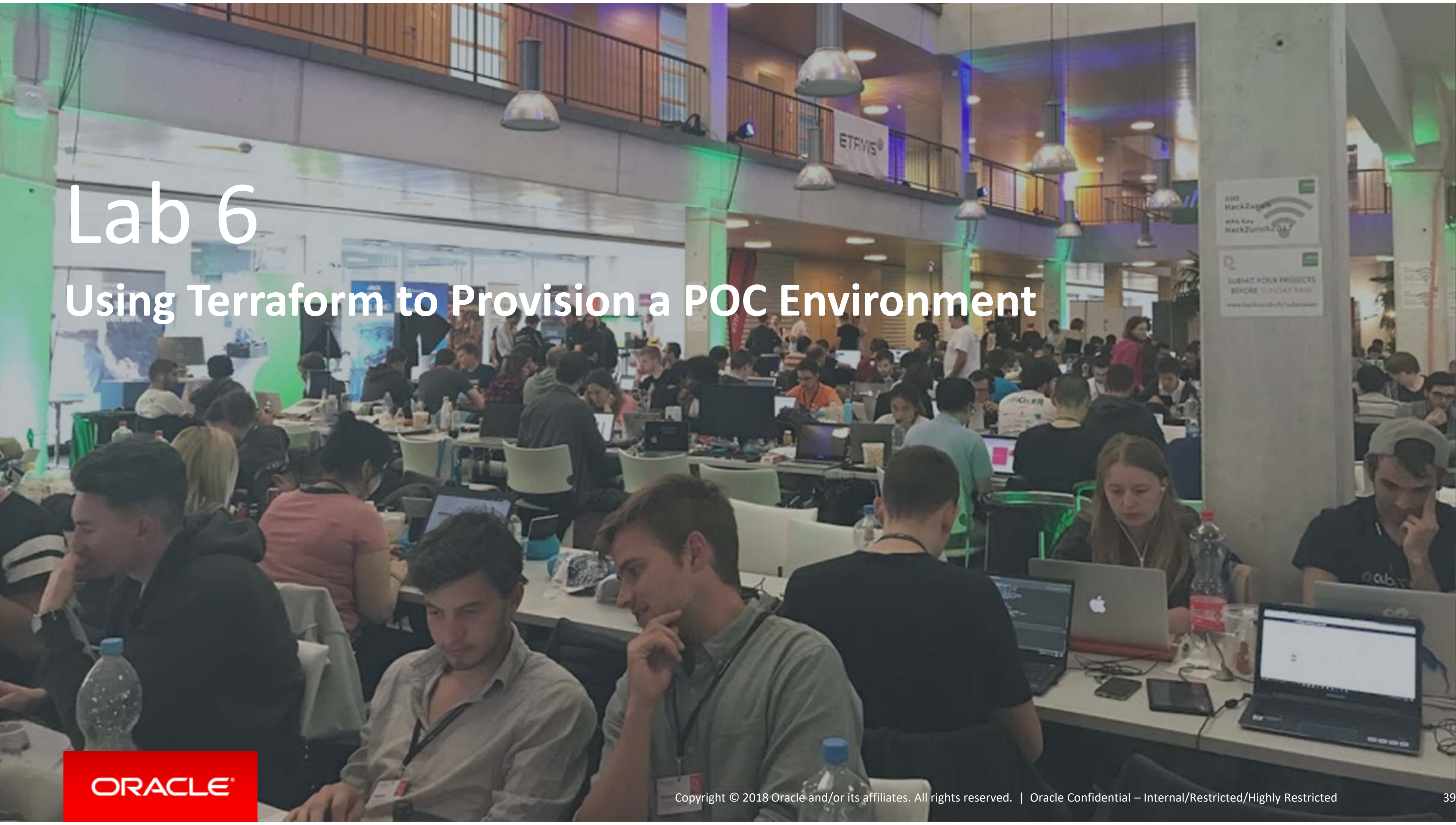
```
${data.TYPE.NAME.ATTRIBUTE}
```

Create a Linux Server with Multiple VNICs (7)

To do so we use another form of the interpolation syntax, the so called **count information syntax**: The syntax is `count.FIELD`. For example, `${count.index}` will interpolate the current index in a multi-count resource. And that's exactly what we are using for our naming scheme, look e.g. at line 110 that defines the display name for a secondary VNIC:

```
110 display_name              = "SecondaryVnic_${count.index}"
```

For the first secondary VNIC created the name will be "`SecondaryVnic_0`" as the index starts at 0, the second one will be called "`SecondaryVnic_1`" etc.



Lab 6

Using Terraform to Provision a POC Environment

Lab 6: Using Terraform to Provision a POC Environment

Objectives

The objective of this lab is to leverage Terraform to provision, setup, and configure another Proof of Concept (POC) environment running in OCI. The lab is considered complete when you are able to connect to the Linux server successfully and all POC requirements are met. You are made aware that there are a lot of sample Terraform configuration files on the Terraform oci Provider GitHub site that will help you get started. At the end of this lab all the resources provisioned should be destroyed.

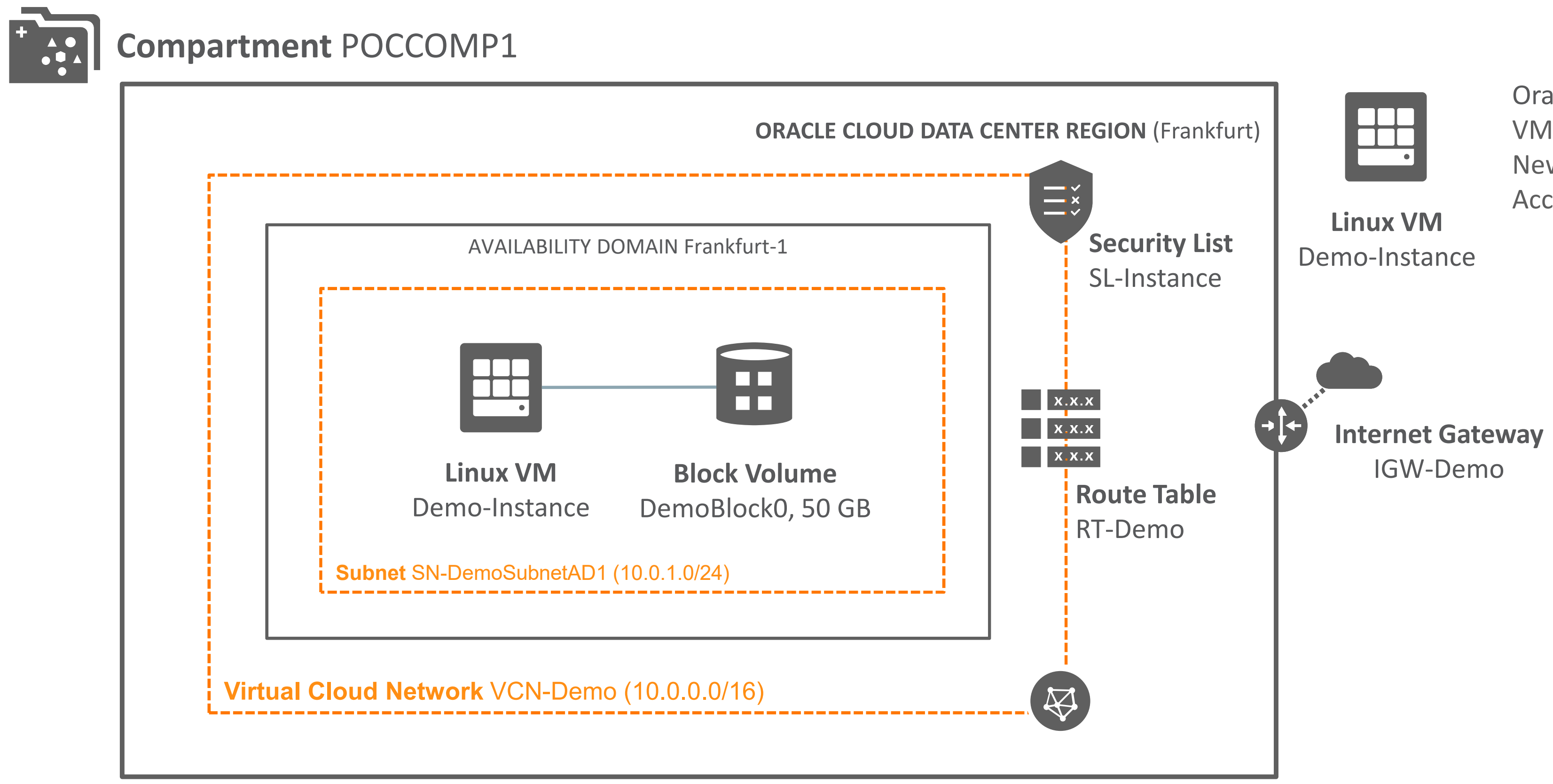
Main tasks for this lab

- **Task 1:** Review and analyze the desired OCI POC architecture
- **Task 2:** Analyze and understand the sample Terraform configuration on GitHub
- **Task 3:** Adapt the sample Terraform configuration to generate the desired POC architecture
- **Task 4:** Debug and test your Terraform configuration

Estimated Time: 90 minutes

ORACLE

Lab 6: POC Environment

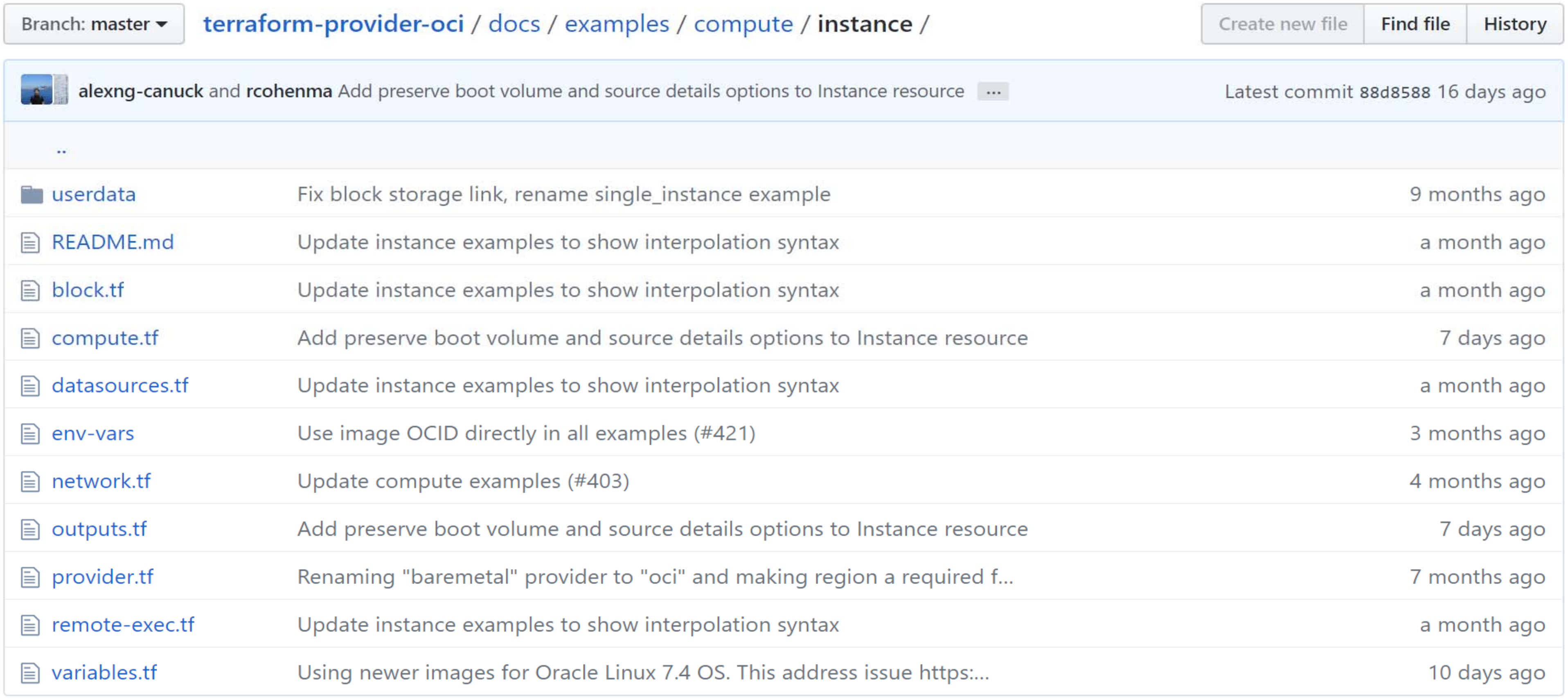


Compartment POCCOMP1
ORACLE CLOUD DATA CENTER REGION (Frankfurt)
AVAILABILITY DOMAIN Frankfurt-1
Linux VM Demo-Instance
Block Volume DemoBlock0, 50 GB
Subnet SN-DemoSubnetAD1 (10.0.1.0/24)
Virtual Cloud Network VCN-Demo (10.0.0.0/16)
Security List SL-Instance
x.x.x
x.x.x
x.x.x
Route Table RT-Demo
Internet Gateway IGW-Demo

Linux VM
Demo-Instance

Oracle Linux 7.4
VM.Standard1.2
New Public/Private Key Pair
Access via ssh

Lab 6 Guidance (1)



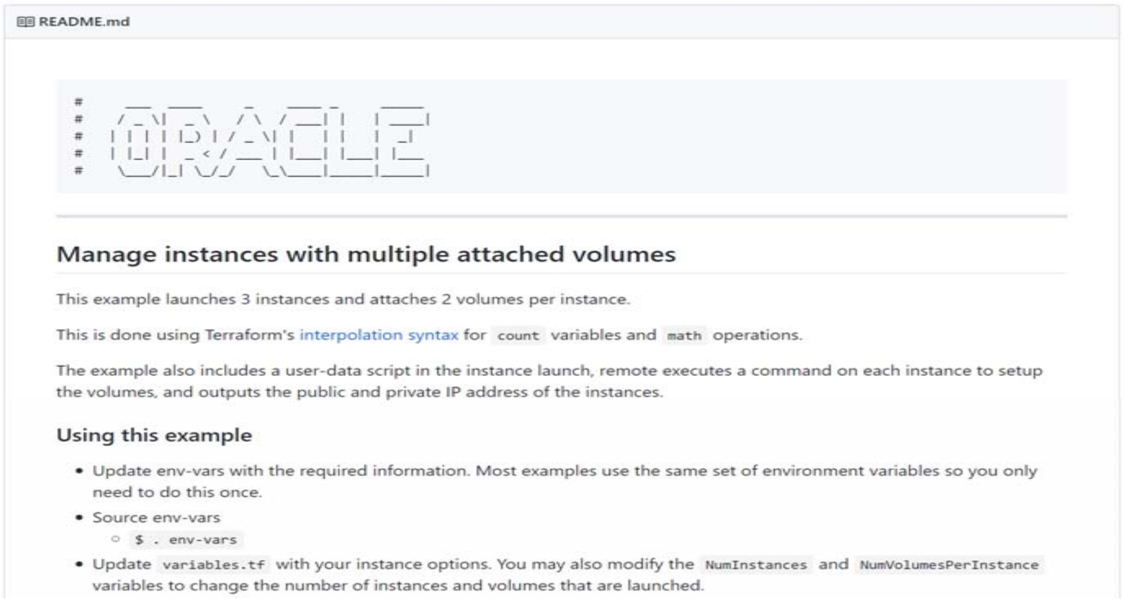
Branch: master ▾ terraform-provider-oci / docs / examples / compute / instance /

alexng-canuck and rcohenma Add preserve boot volume and source details options to Instance resource … Latest commit 88d8588 16 days ago

..

userdata	Fix block storage link, rename single_instance example	9 months ago
README.md	Update instance examples to show interpolation syntax	a month ago
block.tf	Update instance examples to show interpolation syntax	a month ago
compute.tf	Add preserve boot volume and source details options to Instance resource	7 days ago
datasources.tf	Update instance examples to show interpolation syntax	a month ago
env-vars	Use image OCID directly in all examples (#421)	3 months ago
network.tf	Update compute examples (#403)	4 months ago
outputs.tf	Add preserve boot volume and source details options to Instance resource	7 days ago
provider.tf	Renaming "baremetal" provider to "oci" and making region a required f…	7 months ago
remote-exec.tf	Update instance examples to show interpolation syntax	a month ago
variables.tf	Using newer images for Oracle Linux 7.4 OS. This address issue https:…	10 days ago

Lab 6 Guidance (2)



README.md

```
#    ___  ____      _    ____ _     _____
#   / _ \|  _ \    / \  / ___| |   | ____|
#  | | | | |_) |  / _ \| |   | |   |  _|
#  | |_| |  _ <  / ___ \ |___| |___| |___
#   \___/|_| \_\/_/   \_\____|_____|_____|
```

Manage instances with multiple attached volumes

This example launches 3 instances and attaches 2 volumes per instance.

This is done using Terraform's interpolation syntax for `count` variables and `math` operations.

The example also includes a user-data script in the instance launch, remote executes a command on each instance to setup the volumes, and outputs the public and private IP address of the instances.

Using this example

- Update env-vars with the required information. Most examples use the same set of environment variables so you only need to do this once.
- Source env-vars
  - `$ . env-vars`
- Update `variables.tf` with your instance options. You may also modify the `NumInstances` and `NumVolumesPerInstance` variables to change the number of instances and volumes that are launched.

It might be a good idea to try to get this sample configuration to actually work in your environment. Hint: Set the number of instances and the volumes per instance to 1 as this is what's requested in the POC environment (look at `variables.tf`). Take note of the names, IP-ranges, etc. you need to change based on the POC environment specifications.

Review Questions

Terraform

Review Questions for Terraform (1)

1. Terraform supports which public cloud providers?
   A. Oracle Cloud Infrastructure Classic
   B. Oracle Cloud Infrastructure
   C. Oracle Cloud Platform
   D. All of the above

2. Which of the following programming languages or formats can you use for Terraform configurations?
   A. Python
   B. HCL
   C. JSON
   D. Java

3. Which of the following executables do you need on a computer for Terraform to work?
   A. Terraform binary
   B. OCI SDK
   C. OCI CLI Tool
   D. Terraform provider binaries

Review Questions for Terraform (2)

4. Which of the following is a valid Terraform configuration environment variable name?
   - **A.** `OCI_tenancy`
   - **B.** `TENANCY_OCID`
   - **C.** `TF_VAR_tenancy_ocid`
   - **D.** `TERRAFORM.OCI.TENANCY.OCID`

5. What is the usual Terraform workflow of commands to get resources deployed?
   - **A.** Config / Plan / Deploy
   - **B.** Test / Provision / Execute
   - **C.** Init / Plan / Apply
   - **D.** None of the above

Review Questions for Terraform (3)

6. Which of the following statements regarding Terraform are true?
   - **A.** Configuration files in a folder are per default appended to build a configuration.
   - **B.** Terraform always searches the whole directory tree for configuration files.
   - **C.** Terraform uses the procedural way for building infrastructure.
   - **D.** Terraform uses the descriptive way for building infrastructure.

7. How does Terraform process dependencies between resources?
   - **A.** Terraform does not handle dependencies, this needs to be defined manually.
   - **B.** Terraform identifies dependencies, but the programmer needs to resolve them manually.
   - **C.** Terraform builds a resource dependency graph to resolve dependencies.
   - **D.** Terraform leverages the OCI dependency resolution service for this.

8. How are environment variables passed into Terraform configurations?
   - **A.** Define a Terraform variable with the same name as the environment variable.
   - **B.** Environment variables can be directly accessed with the env prefix.
   - **C.** Environment variable names need to start with TF_VAR_.
   - **D.** Terraform cannot use environment variables.

ORACLE

Review Questions for Terraform (4)

9. Which of the following expressions represent a valid Terraform interpolation syntax?

A. `$(var.tenancy_ocid}`

B. `${oci_identity_user.user1.id}`

C. `${count.index}`

D. All of the above.

10. What is a Terraform data source? Choose the best answer.

A. A data source is a connection between Terraform and an external database.

B. A data source presents a read-only view into pre-existing data.

C. Terraform uses data sources as the basis for defining resources.

D. Terraform uses data sources as inputs for built-in providers.

thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
merci
obrigado
dziękuję
bedankt
спасибо
рахмат
Баярлалаа
go raibh maith agat
mochchakkeram
sukriya
kop khun krap
terima kasih
grazie
arigatô
dakujem
tapadh leat
dank je
takk
감사합니다
ありがとう
মেরসি
мерси
хвала
ευχαριστώ

ORACLE

Integrated Cloud

Applications & Platform Services