



Integrated Cloud Applications & Platform Services



# Deploying to Oracle Cloud Infrastructure with Terraform

Student Guide

D106286GC10 | D106477

Learn more from Oracle University at [education.oracle.com](https://education.oracle.com)



**Copyright © 2019, Oracle and/or its affiliates. All rights reserved.**

#### **Disclaimer**

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

#### **Restricted Rights Notice**

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

##### **U.S. GOVERNMENT RIGHTS**

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

#### **Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

**1007312019**

# Contents

## 1 Terraform: Overview

- Course Objectives 1-2
- Lesson Objectives 1-3
- Target Audience 1-4
- Prerequisites 1-5
- Why Infrastructure as Code (IAC)? 1-6
- Infrastructure as Code: Benefits 1-7
- Terraform 1-8
- Terraform: Introduction 1-10
- Differences Between Automation Tools 1-14
- Configuration Management and Orchestration 1-15
- Mutable and Immutable Infrastructure 1-16
- Procedural and Declarative Approach 1-17
- Core Terraform Workflow 1-18
- Terraform Commands 1-19
- Summary 1-20

## 2 Terraform Installation

- Lesson Objectives 2-2
- Installing Terraform on Linux 2-3
- Terraform on Linux 2-4
- Verifying Terraform 2-5
- Installing Terraform on Windows 2-6
- Terraform on Windows 2-7
- Verifying Terraform 2-8
- Writing Terraform Configurations 2-9
- Terraform Configuration Files 2-10
- Terraform Configuration Files: Providers 2-11
- Terraform Configuration Files: OCI Provider 2-12
- Terraform Configuration Files: Resources 2-13
- Building Your First Resource 2-14
- Terraform Plan 2-15
- Terraform Plan: Example 2-16
- Terraform Plan: Indicators 2-17
- terraform apply 2-18
- terraform apply: Example 2-19
- Terraform State File 2-20

- Terraform Local State File 2-21
- Adding a Second Resource 2-22
- Terraform: Targeting Resources 2-23
- Terraform: Outputting Plans 2-24
- Terraform Show 2-25
- Terraform Destroy 2-26
- Terraform Variables 2-28
- Terraform Outputs 2-31
- Terraform Output: Example 2-32
- Terraform Modules 2-33
- Summary 2-34

### **3 Using Terraform on OCI**

- Lesson Objectives 3-2
- Identity and Access Management 3-3
- IAM: Components 3-4
- Dynamic Group 3-5
- Tenancy and Policy 3-6
- Policy Basics 3-7
- Example Case 3-8
- Terraform Resource Syntax 3-9
- What Is a Virtual Cloud Network? 3-10
- Subnet 3-11
- Internet Gateway 3-12
- Dynamic Routing Gateway 3-13
- Security Lists 3-14
- Default VCN Components 3-15
- Public Versus Private Subnets 3-16
- Private IP Versus Public IP 3-17
- Terraform Resource Syntax: VCN 3-18
- Object Storage 3-20
- Backup/Archive 3-22
- Object Storage Resources 3-23
- Compartment 3-24
- Ways to Access Object Storage 3-25
- Terraform Resource Syntax: Object Storage 3-26
- Compute Service: Overview 3-27
- Virtual Machine 3-28
- Components for Launching Instances 3-29
- Sample Diagram: Icons, Groupings, and Connectors 3-30
- Block Volume: Overview 3-31

Volume Types 3-32  
Terraform Resource Syntax: Compute 3-33  
Terraform Resource Syntax 3-34  
Load Balancing: Overview 3-35  
How Load Balancing Works 3-36  
Public Load Balancer 3-37  
Private Load Balancer 3-38  
All Load Balancers 3-39  
Simple Architecture Diagram 3-40  
Load Balancing Concepts 3-41  
Terraform Resource Syntax: Load Balancing 3-42  
File Storage: Overview 3-43  
File Systems Concepts 3-44  
How File Storage Permissions Work 3-46  
Terraform Resource Syntax: EFS 3-47  
Summary 3-48

#### **4 Jenkins**

Objectives 4-2  
Prerequisites 4-3  
What Is Jenkins? 4-4  
Advantages of Jenkins 4-5  
Disadvantages of the Waterfall Model 4-6  
What Is Continuous Integration (CI)? 4-7  
Continuous Integration with Jenkins 4-8  
Impact of Jenkins 4-9  
Jenkins Distributed Architecture 4-10  
Summary 4-11



1



# Terraform: Overview

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Course Objectives



After completing this course, you should be able to describe:

- Terraform
- Its features and providers
- How it is different from other DevOps tools
- Core Terraform workflow
- Execution plans
- Resource graph
- Resource configuration
- Terraform commands (CLI)

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



## Lesson Objectives

After completing this lesson, you should be able to:

- Describe Infrastructure as Code
- Describe Core Terraform Workflow



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Target Audience

This course is mainly intended for:

- System Administrators
- Cloud Architects
- DevOps Engineers



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Prerequisites

You should have:

- A general understanding of cloud technology
- Hands-on experience of Linux/Windows operating systems
- Completed the Oracle Cloud Infrastructure Administration Essentials course



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Why Infrastructure as Code (IAC)?

- Back in the good old days, when you needed to deploy the infrastructure, a group of powerful beings known as sysadmins used to deploy infrastructure manually. Every server, routers, load balancers, and database configuration was created and managed manually.
- There always used to be the fear of accidental misconfiguration, fear of down time, and fear of slow and fragile deployments.
- What if sys admins are not available?
- To overcome these type of scenarios or issues, DevOps has come into play where we write code to do things Infrastructure as Code.
- The idea behind IAC is to write code to define, provision, and manage your infrastructure.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Infrastructure as Code: Benefits

- The entire provisioning and deployment process can be automated, which makes it much faster and more reliable than any manual process.
- You can represent the state of the infrastructure in source files that anyone can read.
- You can store those source files using any version control software. This will allow you to have multiple versions of the same file with most recent updates and commit log messages as well, easily roll back the changes if something is not working.
- You can validate all infrastructure changes through code reviews and automated tests.
- Tools for IAC are available in the market.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Terraform

- Terraform allows you to write infrastructure as code, that is, whatever tasks you do using a console graphically, you can do using code.
- It's automation of the infrastructure.
- Terraform can manage existing and popular service providers as well as custom in-house solutions.
- You can keep your infrastructure in a certain state.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. Terraform is part of the cloud orchestration tools, which handles infrastructure and application lifecycle.

# Terraform

- Terraform enables you to make your infrastructure auditable.
  - That is, you can upload your source files to version control systems, such as Git.
- The main use of Terraform is that it will automate the provisioning of the Infrastructure itself.
  - Example: By using Oracle Cloud (API), you can use Terraform with other providers as well, where you have API functionality.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Infrastructure as Code's goal is to create and manage cloud infrastructure and deployments predictably and repeatedly. It makes use of templates and automation for just about everything.

As part of your automation tools, Terraform can be integrated with configuration management tools, such as Chef, Puppet, and Ansible.

# Terraform: Introduction

- Why not a configuration management tool?
- What can you use Terraform for?
  - Multi-tier applications
  - Self-service infrastructure
  - Production, development, and testing environments
  - Continuous delivery



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Why Not a Configuration Management Tool?

Configuration management tools such as Ansible, Chef, and Puppet focus on automating the installation and configuration of software.

Puppet, Chef, and Ansible are excellent at managing applications and services. However, they're often focused on software configuration rather than building the infrastructure components that underpin your applications. Terraform is instead focused on building and deploying infrastructure.

## Multi-Tier Applications

Terraform is focused on infrastructure deployment and supports a diverse collection of components including compute, storage, and networking assets. This makes it ideal to build N-tier applications.

## Self-Service Infrastructure

Often operations teams aspire to being infrastructure coordinators and controllers rather than bottlenecks in the infrastructure provisioning and management process. In this world, operations provide scalable and cost effective self-service infrastructure to their customers.



## **Production, development, and testing environments**

A common problem in complex environments is that development and testing environments do not always reflect the state of the production environments. With Terraform, because your infrastructure is now codified, it's easy to ensure that your production configuration can be shared with your development and testing environments to ensure consistency and compatibility.

## **Continuous Delivery**

If you're operating in an environment where applications are packaged in artifacts and your application configuration is done with configuration management tools, then adding Terraform allows you to also codify your infrastructure. This means all of the pieces of your environment are now automatically deployable, scalable, and manageable.

## Terraform: Introduction

- Written in Go
- Fast development - releases monthly+
- HCL – Hashi Configuration Language and simple markup format
  - JSON interoperable

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Terraform is written in Go, which is easily extensible, as the OCI Provider for Terraform shows.

Terraform development moves quickly. As soon as the Cloud providers change, Terraform quickly adapts. This is important – users don't want to have to write the abstractions for a given cloud service.

HCL is a declarative language. Therefore, your goal is to describe the infrastructure you want, and Terraform will figure out how to create it.

HCL configuration language supports comments, has auto-formatting built-in (terraform fmt), and can have comments. Terraform itself can read JSON in place of HCL if needed.

# Terraform: Introduction

- Traditional procedural build process:
  - Log in to a dashboard.
  - Go to the Compute tab.
  - Click Launch New Instance.
  - .....
  - Configure VM, and so on.
- Terraform uses a declarative build process:
  - A VM with two OCPUs and 7 GB RAM
  - 100 GB of block storage
  - CentOS 7
  - Two network interfaces with IP 10.0.0.1/24 and 10.0.1.1/24

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Terraform provides declarative execution plans for building and running applications and infrastructure. What does declarative mean? Declarative plans are ones in which you specify outcomes. So, you let the tool take care of the steps required and focus on the outcome. This means you no longer need to worry about what specific commands to run, buttons to push, or settings to tweak. The tool underneath takes care of those details.

## Differences Between Automation Tools

Tool	Released by	Code	Method	Approach	Written in	Installation	Type	Architecture	Infrastructure
Ansible	Red Hat	Open Source	Push	Procedural	Python/YAML	Easy	Config Mgt	Client only	Mutable
Terraform	HashiCorp	Open Source	Push	Declarative	GO	Easy	Orchestration	Client only	Immutable
Chef	Chef	Open Source	Pull	Procedural	Ruby	Hard	Config Mgt	Server/Client	Mutable
Puppet	Puppet	Open Source	Pull	Declarative	Ruby	Hard	Config Mgt	Server/Client	Mutable
SaltStack	SaltStack	Open Source	Pull & Push	Declarative	Python	Hard	Config Mgt	Server/Client	Mutable
CloudFormation	AWS	Closed Source	Pull	Declarative	JSON	Hard	Orchestration	Client only	Immutable

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Configuration Management and Orchestration

- **Configuration management tools** – After a server is provisioned, these tools are designed to install and manage software on the server.
  - Examples: Chef, Puppet, Ansible, and SaltStack
- **Orchestration tools** – These are designed to provision the servers themselves, leaving the job of configuring those servers to the other tools.
  - Examples: CloudFormation (doesn't support OCI) and Terraform



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Mutable and Immutable Infrastructure

- **Mutable** – If you use Ansible to install a new version of OpenSSL, it'll run the software update on your existing servers and the changes will happen in-place.
  - Chef, Puppet, Ansible, and SaltStack are mutable infrastructure.
- **Immutable** – If you're using a provisioning tool, such as Terraform, to deploy machine images created by Docker or Packer, then every "change" is actually a deployment of a new server (just like every "change" to a variable in functional programming actually returns a new variable).
  - Terraform and CloudFormation are immutable infrastructure.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

### Immutable Infrastructure: Example

To deploy a new version of OpenSSL, you would create a new image using Packer or Docker with the new version of OpenSSL already installed, deploy that image across a set of totally new servers, and then undeploy the old servers.

## Procedural and Declarative Approach

- The **Procedural** approach defines specific commands that need to be executed in the appropriate order to end with the desired conclusion (step-by-step order).
  - Chef and Ansible
- The **Declarative** approach needs a more declarative style where you write code that specifies your desired end state. The IAC tool itself is responsible for figuring out how to achieve that state.
  - Terraform, CloudFormation, SaltStack, and Puppet
  - Terraform uses state information to monitor all entities. When an additional change is pushed, it compares it with existing metadata and applies only the delta changes to infrastructure, thus reducing the time and efforts put by developer to maintain the inventory information.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Core Terraform Workflow

- The core Terraform workflow has three steps:
  1. **Write** – Author infrastructure as code.
  2. **Plan** – Preview changes before applying.
  3. **Apply** – Provision reproducible infrastructure.
- Terraform has a "planning" step where it generates an *execution plan*. The execution plan shows what Terraform will do when you call apply. This lets you avoid any surprises when Terraform manipulates infrastructure.



## Resource Graph

- Terraform builds a graph of all your resources, and parallelizes the creation and modification of any nondependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Terraform Commands

- **Apply:** The `terraform apply` command is used to apply the changes required to reach the desired state of the configuration, or the predetermined set of actions generated by a terraform execution plan.
- **Destroy:** To destroy the infrastructure, use this command.
- **Init:** The `terraform init` command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.
- **Validate:** To validate the syntax of Terraform files, use this command.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Summary

In this lesson, you should have learned to describe:

- Infrastructure as Code
- Core Terraform Workflow



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Terraform Installation

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Lesson Objectives



After completing this lesson, you should be able to:

- Install Terraform
- Describe Terraform configuration files
- Create a resource using Terraform
- Explain basic Terraform commands and flags

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Installing Terraform on Linux

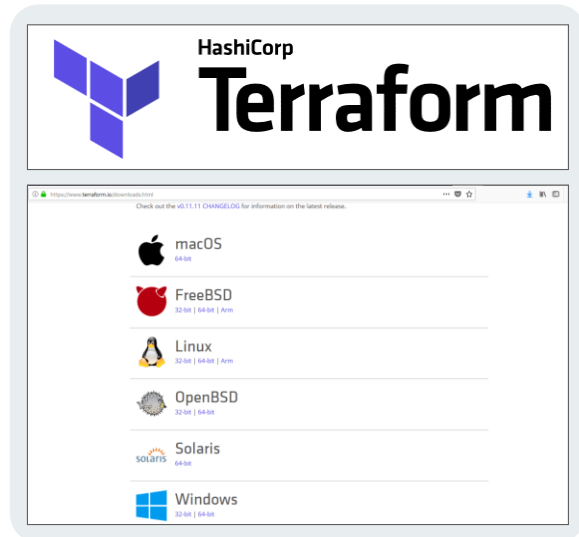
Installing Terraform is similar to all OS but with a slight difference.

You can download the Terraform software (depending on the OS version) from this link:

<https://www.terraform.io/downloads.html>

We're going to install it on Linux and Windows.

Download and unzip the package on the VM.



```
dr-xr-xr-x. 18 root root    4096 Jan  6 11:01 ..
-rw-r--r--.  1 root root 20971661 Jan 11 01:34 terraform_0.11.11_linux_amd64.zip
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform on Linux

```
[root@server3 opt]# unzip terraform_0.11.11_linux_amd64.zip
Archive:  terraform_0.11.11_linux_amd64.zip
  inflating: terraform
[root@server3 opt]#
```

After you have unzipped the Terraform package, update the `PATH` environment variable pointing to Terraform. The `/usr/local/bin` folder is already set to the `PATH` environment variable; you don't need to set it again. If you are using any other location, specify it in the `PATH` environment variable, either in `.bash_profile` or in `/etc/profile`.

```
[root@server3 opt]# mv terraform /usr/local/bin
```

```
[root@server3 opt]# cd /usr/local/bin
[root@server3 bin]# ls -l terraform
-rwxrwxr-x. 1 root root 89483552 Dec 14 16:20 terraform
[root@server3 bin]#
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

**Note:** If you get an error that Terraform could not be found, your `PATH` environment variable was not set up properly. Go back and ensure that your `PATH` variable contains the directory where Terraform was installed.

# Verifying Terraform

Verify the installation of Terraform with the following command:

```
[raghu@server3 ~]$ 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
  get            Download and install modules for the configuration
  graph          Create a visual graph of Terraform resources
  import         Import existing infrastructure into Terraform
  init           Initialize a Terraform working directory
  output         Read an output from a state file
  plan           Generate and show an execution plan
  providers      Prints a tree of the providers used in the configuration
  push           Upload this Terraform module to Atlas to run
  refresh        Update local state file against real resources
  show           Inspect Terraform state or plan
  taint          Manually mark a resource for recreation
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

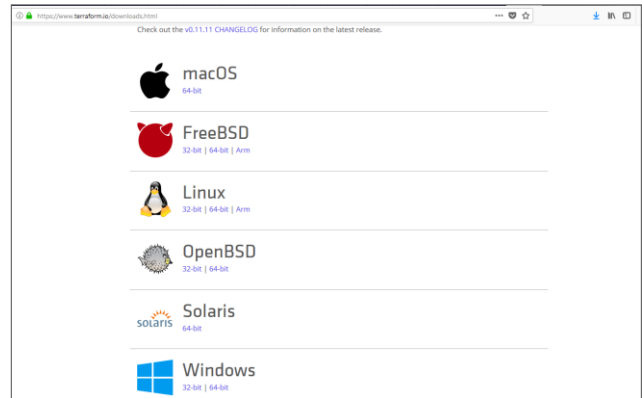
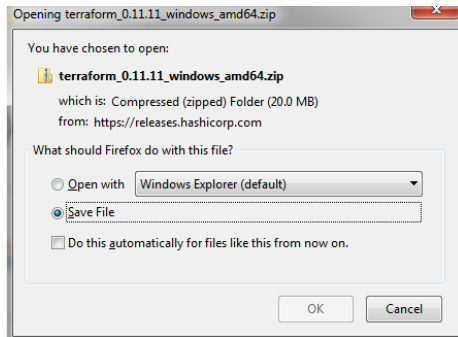
Oracle Internal & Oracle Academy Use Only

# Installing Terraform on Windows

Installing Terraform is similar to all OS but with a slight difference.

You can download the Terraform software (depending on the OS version) from this link:

<https://www.terraform.io/downloads.html>



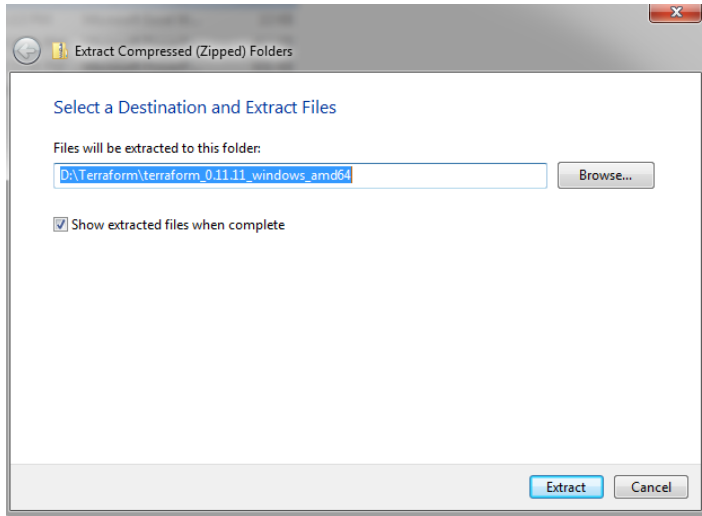
ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Terraform on Windows



Unzip the folder by right-clicking the folder and selecting Extract all.

You can use either cmd or PowerShell to execute the executable depending on the version of the OS.

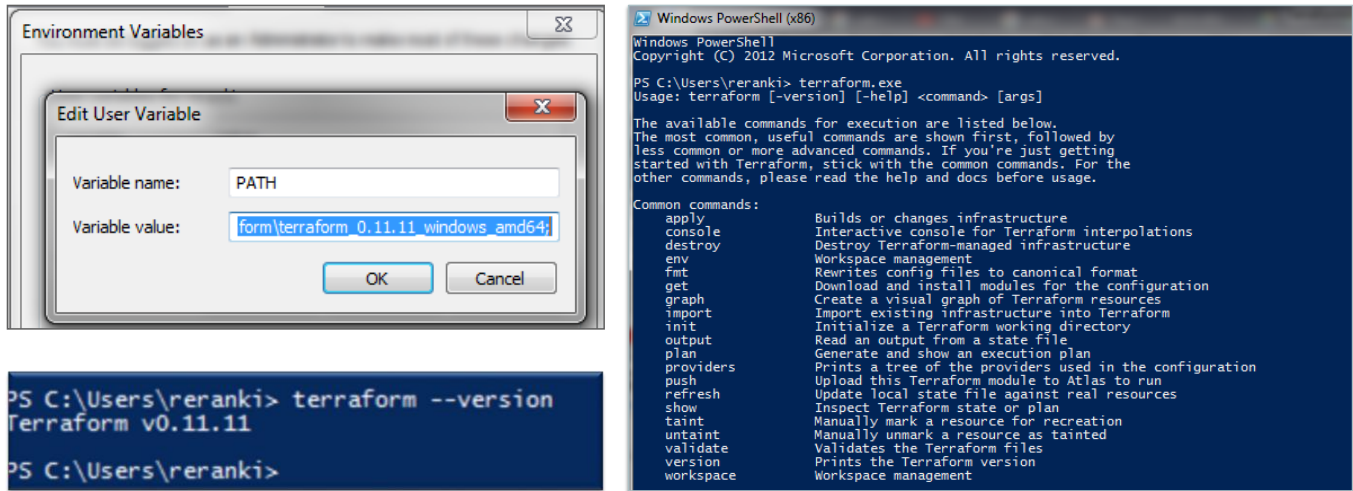
If you need to have Terraform executable everywhere, then you must set it in `PATH`.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Verifying Terraform



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

**Note:** Depending on the OS version, steps to add path will vary slightly.

**Example:** Windows 7 OS

Computer > Right-click > Properties > Advance system Settings > Environment Variable > PATH

Then edit the PATH field and update the new path of Terraform.

## Writing Terraform Configurations

- Describe your Oracle Cloud Infrastructure using the HashiCorp Configuration Language format (HCL) in Terraform configuration files.
- Configuration files that use the HCL format end with the `.tf`.
- Use Terraform configurations to define your Oracle Cloud Infrastructure resources, variable definitions, data sources, and a great deal more.
- Terraform then converts your Oracle Cloud Infrastructure configurations into a set of API calls.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform Configuration Files

- Configuration can be in a single file or split across multiple files.
- Terraform will merge all files in the current working directory, which ends in `.tf` or `.tf.json`.
- Subfolders are not included (nonrecursive).
- Files are merged in alphabetical order.
- Any files with a different extension are ignored (for example, `.jpg`).

**Tip:** Set up a directory to host all your Terraform files.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When Terraform runs inside a directory, it will load any Terraform configuration files. Any non-configuration files are ignored and Terraform will not recurse into any sub-directories. Each file is loaded in alphabetical order, and the contents of each configuration file are appended into one configuration.

## Terraform Configuration Files: Providers

- Providers connect Terraform to the infrastructure you want to manage.
- They provide configuration like connection details and authentication credentials.
- You can think about them as a wrapper around the services whose infrastructure you want to manage.
- To download the providers you're using in your environment, you need to run the `terraform init` command to install any required providers.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Providers connect Terraform to the infrastructure you want to manage, for example, OCI, or a variety of other Cloud, network, storage, and SaaS services. They provide configuration like connection details and authentication credentials. You can think about them as a wrapper around the services whose infrastructure you wish to manage.

Providers are not shipped with Terraform since Terraform 0.10. To download the providers you use in your environment, you need to run the `terraform init` command to install any required providers.

## Terraform Configuration Files: OCI Provider

- First, set up a provider.
- Providers abstract the APIs from any given third party to create infrastructure. Here is the OCI example:

```
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}"  
}
```

- The OCI provider enables Terraform to create, manage, and destroy resources in your tenancy on OCI.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You need to specify connections details, such as to which OCI region to connect. Also specify authentication credentials, such as the access and private key that will allow you to use your OCI account from Terraform.

## Terraform Configuration Files: Resources

- After a provider is configured, you can start using that provider's resources.
- Using the OCI provider, you can start creating instances, block and object storage, networks, and so on.

The following example starts an instance:

Component	Provider	Type	Name
-----------	----------	------	------

```
resource "oci_core_instance" "web01" {  
  availability_domain = "${lookup(data.oci_identity_availability_domains.ADs.availability_domains[0], "name")}"  
  compartment_id     = "${var.compartment_ocid}"  
  display_name       = "web-server-01"  
  image              = "${lookup(data.oci_core_images.OS_image_ocid.images[0], "id")}"  
  shape              = "${var.WebShape}"  
}
```

**Note:** The combination of type and name must be unique in your configuration.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Resources are the bread and butter of Terraform. They represent the infrastructure components you want to manage: hosts, networks, firewalls, DNS entries, and so on. The resource object is constructed of a type, name, and a block containing the configuration of the resource.

The type of resource here is `core_instance`, which manages OCI host instances. Each type of resource is linked to a provider; you can tell which by the leftmost value in the type, here `oci`. This indicates that this type of resource is provided by the OCI provider.

The name of the resource is specified next. This name is defined by you. Here, you have named this resource `base`. The name of the resource should generally describe what the resource is or does.

The combination of type and name must be unique in your configuration. Therefore, there can be only one `oci_core_instance` named `base` in your configuration. If you specify more than one resource type with the same name, you'll see an error like the following:

```
* oci_core_instance.web01: resource repeated multiple times
```

## Building Your First Resource

Q: How do we get Terraform to create that resource?

A: Terraform calls the process of creating and managing infrastructure an execution plan. You will likely run execution plans in one of two modes:

- Plan - Display the proposed changes to make.
- Apply - Apply the changes.

**Example:**

```
resource "oci_identity_user" "user01" {  
  name = "user01-TF"  
  description = "A user managed with Terraform"  
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



## Terraform Plan

- The plan shows you what will happen.
- You can save plans to guarantee what will happen.
- Plans show reasons for certain actions (such as re-create).
- Prior to Terraform, users had to guess change ordering, parallelization, and rollout effect.

```
# terraform plan
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The `plan` command enables you to see what Terraform is proposing to do; a check that you're going to make the right changes. This is useful for ensuring you're doing the right thing and not about to break your infrastructure. You should always run a plan before you apply any changes.

When looking at a plan, remember that all the `.tf` files in the directory are considered at the same time. There is no recursion into sub-directories.

## Terraform Plan: Example

```
[opc@terraform-server tftest]$ 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.

data.oci_identity_availability_domains.ADs: Refreshing state...

-----

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_identity_user.user01
  id:          <computed>
  compartment_id: <computed>
  description:   "A user managed with Terraform"
  inactive_state: <computed>
  name:          "user01-TF"
  state:         <computed>
  time_created:  <computed>
  time_modified: <computed>
```





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

-----  
Note: You didn't specify an "-out" parameter to save this plan, so Terraform  
can't guarantee that exactly these actions will be performed if  
"terraform apply" is subsequently run.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform Plan: Indicators

-  indicates that a resource will be created.
-  indicates that a resource will be destroyed.
-  indicates that a resource will be updated in-place.
-  indicates that a resource will be destroyed and re-created.

**ORACLE**

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can see that the resource identifier is prefixed with a +. This indicates that this resource is a planned change that has not yet been made. Think about the + like you would in a version control system. It indicates a new, pending change. There are a series of similar indicators:

- +: A resource will be added.
- -: A resource will be destroyed.
- -/+: A resource will be destroyed and then added again.
- ~: A resource will be changed.

## terraform apply

- Executes changes in order based on the resource graph
- Parallelizes changes when possible
- Handles and recovers transient errors
- Updates existing resources when updates are allowed
- Re-creates existing resources when updates are not allowed

```
# terraform apply
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

After reviewing the plan output, everything looks good. The next stage of the process is to actually create the resource. This is done using the `apply` command.

## terraform apply: Example

```
[opc@terraform-server tftest]$ terraform apply
data.oci_identity_availability_domains.ADs: Refreshing state...
```

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_identity_user.user01
  id:          <computed>
  compartment_id: <computed>
  description:  "A user managed with Terraform"
  inactive_state: <computed>
  name:        "user01-TF"
  state:       <computed>
  time_created: <computed>
  time_modified: <computed>
```

```
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: yes

```
oci_identity_user.user01: Creating...
```

```
compartment_id: "" => "<computed>"
description:    "" => "A user managed with Terraform"
inactive_state: "" => "<computed>"
name:          "" => "user01-TF"
state:         "" => "<computed>"
time_created:  "" => "<computed>"
time_modified: "" => "<computed>"
```

```
oci_identity_user.user01: Creation complete after 1s (ID: oci1.user.oc1..)
```

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

```
[opc@terraform-server tftest]$
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

`terraform apply` takes the planning phase into action. `apply` goes out and instructs the provider to actually create all the indicated resources.

Terraform, by default, creates a backup of the state file, which is built during `apply`, `refresh`, and `destroy` operations.

If errors are encountered during an `apply`, there is no automatic rollback. However, whatever issues that come up can be remedied and then the configuration can be applied again. For example, if there is an API timeout with the provider that exceeds a threshold then a `reapply` can be used to get past that type of error.

After `apply` complete, you have an updated state file, which matches the deployed infrastructure.

`terraform apply` now has a prompt, which requires the input "YES" to proceed. This was added because `apply` can end up using `destroy-recreate` on certain resources without a lifecycle directive.

## Terraform State File

- Terraform stores the state of your managed infrastructure from the last time Terraform was run.
- Terraform uses this state to create plans and make changes to your infrastructure.
- It is critical that this state is maintained appropriately so future runs operate as expected.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

After creating your resource, Terraform has saved the current state of your infrastructure into a file called `terraform.tfstate` in the `base` directory. This is called a state file. The state file contains a map of resources and their data to resource IDs.

## Terraform Local State File

- State is stored locally on a local machine in JSON format.
- Because it must exist, it is a frequent source of merge conflicts.
- It is generally acceptable for individuals and small teams.
  - Tends not to scale for large teams
  - Requires a more "mono repo" pattern



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The state is the canonical record of what Terraform is managing for you. This file is important because it is canonical. If you delete the file, Terraform will not know what resources are being managed, and it will attempt to apply all configurations from scratch.

Ensure you preserve the state file. Terraform also creates a backup file of the state from the most recent previous execution in a file called `terraform.tfstate.backup`.

You can see that the `terraform.tfstate` file is a JSON representation of all the resources you're managing. The JSON is topped with some metadata, including the version of Terraform that created the state file.

Because this file is the source of truth for the infrastructure being managed, it's critical to use only Terraform to manage that infrastructure. If you make a change to your infrastructure manually, or if you use another tool, it can be easy for this state to get out of sync with reality. You can then lose track of the state of your infrastructure and its configuration, or have Terraform reset your infrastructure back to a potentially non-functioning configuration when it runs.

We strongly recommend that you manage specific infrastructure with Terraform, so that it becomes the sole way of managing that infrastructure.

## Adding a Second Resource

```
resource "oci_identity_user" "user01" {
  name = "user01-TF"
  description = "A user managed with Terraform"
}

resource "oci_identity_ui_password" "user01_password" {
  user_id = "${oci_identity_user.user01.id}"
}
```

```
[opc@terraform-server tfest]$ terraform show
oci_identity_ui_password.user01_password:
  id = ocid1.user.oc1..aaaaaaafath5fw2pkq5brbgee3otugp2qupqo2eifrli7m3fbh4az2yua7q
  password = URz8ziw0rwl&v!$ikJ(0
  state = ACTIVE
  time_created = 2018-03-23 16:07:43.809 +0000 UTC
  user_id = ocid1.user.oc1..aaaaaaafqth5fw2pkq5brbgee3otugp2qupqo2eifrli7m3fbh4az2yua7q
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

This will set up the password for your `oci` user created by Terraform.

Every Terraform plan or apply follows the same process:

1. You query the state of the resources, if they exist now.
2. Compare that state against any proposed changes to be made, building the graph of resources and their relationships. As a result of the graph, Terraform will only propose the set of required changes.
3. If they are not the same, either show the proposed change, if in the plan phase, or make the change, if in the apply phase.



## Terraform: Targeting Resources

To enable systematic and incremental rollout of resources, Terraform has another useful flag: `-target`.

```
[opc@terraform-server tfest]$ terraform plan -target=oci_identity_user.user02
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_identity_user.user02
  id:          <computed>
  compartment_id: <computed>
  description:   "A user managed with Terraform"
  inactive_state: <computed>
  name:          "user02-TF"
  state:         <computed>
  time_created:  <computed>
  time_modified: <computed>

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

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can use the `-target` flag on both the `terraform plan` and `terraform apply` commands. It allows you to target a resource or more if you specify multiple `-target` flags to be managed in an execution plan.

## Terraform: Outputting Plans

- Terraform has an approach for trying to limit the risk of large-scale destructive changes to the environment.
- The plan output enables incremental changes.

```
[opc@terraform-server tftest]$ terraform plan -out user02.plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
persisted to local or remote state storage.

oci_identity_user.user01: Refreshing state... (ID: ocid1.user.oc1
oci_identity_ui_password.user01_password: Refreshing state... (ID

-----

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

This plan was saved to: user02.plan

To perform exactly these actions, run the following command to apply:
  terraform apply "user02.plan"
```

**ORACLE**

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Terraform has an approach to limit the risk of large-scale destructive changes to your environment and allows you to make incremental changes. To do this, Terraform captures the proposed changes by outputting the plan it intends to run to a file.

Terraform calls this a plan output. You capture the plan by specifying the `-out` flag on a `terraform plan` command. This will capture the proposed changes in a file you specify. The plan output means you can make small, incremental changes to your infrastructure.

## Terraform Show

- `terraform show`:
  - Displays human-friendly state
  - Can be used to get on-the-fly information
- Run `terraform show` and get the user information you just created.

```
[opc@terraform-server tftest]$ terraform show
oci_identity_user.user01:
  id = ocid1.user.oc1..aaaaaaafqth5fw2pkq5brbgee3otugp2qupqp2eifrli7m3fbh4az2yua7q
  compartment_id = ocid1.tenancy.oc1..aaaaaaa3epk7xxnme4cs6r7umreuulyou5poomg2kpd5xd7xykvdcypbzqa
  description = A user managed with Terraform
  name = user01-TF
  state = ACTIVE
  time_created = 2018-03-23 15:45:53.913 +0000 UTC
[opc@terraform-server tftest]$
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can see a full list of the resource's attributes and their values. This includes the values of the attributes you configured earlier and some new values, computed when you created the resource. This means all the "`<user>`" attributes have now been replaced with actual values from OCI. For example, now that you've created your `oci` user resource, you know its password and other useful information.

# Terraform Destroy

- `terraform destroy`:
  - Destroys the running infrastructure
  - Does not touch infrastructure that is not managed by Terraform
  - Will ask for permission, requiring an explicit `yes` as input
  - Without any options, destroys everything
- If you want to destroy only a specific resource, then you can use the `-target` flag.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Completely destroying an infrastructure using Terraform is almost easier than creating it.

All that needs to be done is to run `terraform destroy`. By default, without the `-force` option, "yes" is required to proceed with a destroy.

Another way of destroying things, which is more likely to be useful over time is to remove resources from the template file and then run apply again. This will handle destroying the resources and their dependencies without having to destroy everything.

Another way of prompting a `destroy-recreate` of a specific resource would be to taint the resource. Tainted resources will be re-created on the next apply.

Another way of dealing with destruction of resources with Terraform is to use a lifecycle directive in the configuration. This can prevent destruction, direct terraform to re-create a resource before destroy, and also ignore changes. Using this, it would be possible to prevent destruction of a resource for a change in something like a `user_data` bootstrap script.

# Terraform Destroy

```
[opc@terraform-server tftest]$ terraform destroy
oci_identity_user.user02: Refreshing state... (ID: oci_identity_user.user02)
oci_identity_user.user01: Refreshing state... (ID: oci_identity_user.user01)
oci_identity_ui_password.tf_password: Refreshing state... (ID: oci_identity_ui_password.tf_password)
oci_identity_ui_password.user01_password: Refreshing state... (ID: oci_identity_ui_password.user01_password)

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_identity_ui_password.tf_password
- oci_identity_ui_password.user01_password
- oci_identity_user.user01
- oci_identity_user.user02

Plan: 0 to add, 0 to change, 4 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_identity_ui_password.user01_password: Destroying... (ID: ocid1.user.oc1..aaaaaaaafqth5f)
oci_identity_ui_password.tf_password: Destroying... (ID: ocid1.user.oc1..aaaaaaaad6hb5v)
oci_identity_ui_password.user01_password: Destruction complete after 0s
oci_identity_ui_password.tf_password: Destruction complete after 0s
oci_identity_user.user01: Destroying... (ID: ocid1.user.oc1..aaaaaaaafqth5f)
oci_identity_user.user02: Destroying... (ID: ocid1.user.oc1..aaaaaaaad6hb5v)
oci_identity_user.user02: Destruction complete after 1s
oci_identity_user.user01: Destruction complete after 1s
```

Destroy complete! Resources: 4 destroyed.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Terraform Variables

- Define the parameterization of Terraform configurations
- Can have defaults, be provided with a variables file, be asked for at execution, or overridden via the CLI

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Variables enable you to centralize and manage values in your configuration.

## Terraform Variables

- In these examples, you saw some variables, such as `compartment_id`, `image`, and `shape_id`. Ideally, variables are defined in a `variables.tf` file, where defaults can be supplied.
- Variables can be string, list, boolean, and map.

### String Variables

```
# Choose an Availability Domain
variable "AD" {
  default = "1"
}
variable "InstanceShape" {
  default = "VM.Standard1.2"
}
variable "InstanceImageDisplayName" {
  default = "Oracle-Linux-7.4-2017.10.25-0"
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Variables can be defined anywhere, but best practices suggest creating a `variables.tf` file and defining them as well as providing descriptions setting default values where needed.

There are a few basic variable types. The default variable type is string. There are also maps (associative arrays) and lists (arrays).

Descriptions are useful especially when the variables can have impact, such as the region variable for the provider.

# Terraform Variables

## Map Variable Example

```
variable "environment" { default = "dev" }
variable "shape" {
  type = "map"
  default = {
    dev = "VM.Standard1.2"
    test = "VM.Standard1.4"
    prod = "BM.Standard1.36"
  }
}

resource "oci_core_instance" "app-server" {
  image = "${var.image}"
  shape = "${lookup(var.instance_type, var.environment)}"
  subnet_id = "${var.subnet_id}"
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Maps can be used to do things, such as setting an image size based on an environment variable.

For example, if `environment` is `dev`, use `vm1.2` shape; if `environment` is `test`, use `1.4` shape; and if `environment` is `production`, use the bare metal shape.

If a variable has no default and the value has not been set, Terraform will prompt the user interactively to set the variable.

Variables can be overridden by environment variables (prefixed with `TF_VAR_`), on the command line, `tfvars` files, or overridden in the template or when using a module.

Using environment variables for sensitive information is handy to prevent it from being checked in.

If a `terraform run` is attempted and the variables that are used in the Terraform file have no definition, Terraform will prompt the user to fill in the variables value interactively to proceed.

An example of overriding a variable from the command line is as follows:

```
$ terraform apply -var 'InstanceShape=VM.Standard2.1'
```



## Terraform Outputs

- Terraform can be directed to display the variables that are generated dynamically as part of the process of creating the infrastructure.
- For example, after a run, you might want to see the public IP of the host:

```
$ cat outputs.tf
output "InstancePrivateIP" {value =
["${data.oci_core_vnic.InstanceVnic.private_ip_address}"]}
output "InstancePublicIP" {value =
["${data.oci_core_vnic.InstanceVnic.public_ip_address}"]}
```

*After a terraform apply:*

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

State path:

Outputs:

InstancePrivateIP = [ 10.0.0.10 ]

InstancePublicIP = [ 129.146.3.173]



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The output construct can be used in any Terraform configuration. It is a way to highlight specific information from the attributes of resources that are being created. This allows the operator to selectively return critical information to the user or to another application rather than returning all the possible attributes of all resources and having to filter the information down.

Outputs are often used to facilitate interaction with other infrastructure tools. Terraform show (human readable) and `terraform.tfstate` file also store these outputs.

Outputs can also be flagged as SENSITIVE and their output can be redacted when displayed or logged. Typically, private keys, passwords, and other items in the provider are by default marked sensitive and show up redacted by default.

## Terraform Output: Example

```
output "User01-Password" {  
  sensitive = false  
  value = ["${oci_identity_ui_password.user01_password.password}"]  
}
```

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

```
User01-Password = [  
  AP]N9AMLN6]BC&GmA]Zv  
]
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Terraform Modules

## Modules:

- Are portable Terraform configurations (packages)
- Allow separation of concerns and responsibilities among teams
- Are just Terraform configurations inside a folder. There's nothing special about them.

```
module "users" {  
  source = "modules/users"  
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Modules are defined with the module block. Modules are a way of constructing reusable bundles of resources. They allow you to organize collections of Terraform code that you can share across configurations.

Often you have a configuration construct, such as infrastructure like an OCI VCN, an application stack, or other collection of resources that you need to repeat multiple times in your configurations. Rather than cutting and pasting and repeating all the resources required to configure that infrastructure, you can bundle them into a module. You can then reuse that module in your configurations.

# Summary

In this lesson, you should have learned to:

- Download and install Terraform
- Describe Terraform configuration files
- Create a resource using Terraform
- Explain basic Terraform commands and flags



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



## Using Terraform on OCI

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Lesson Objectives



After completing this lesson, you should be able to describe and create various resources on OCI using Terraform.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Identity and Access Management

- Oracle Cloud Infrastructure Identity and Access Management (IAM) lets you control who has access to your cloud resources.
- You can control what type of access a group of users have and to which specific resources.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## IAM: Components

- **Resources:** The cloud objects that your company's employees create and use when interacting with Oracle Cloud Infrastructure
  - **Example:** Compute instances, block storage volumes, virtual cloud networks (VCNs), subnets, route tables, and so on
- **User:** An individual employee or system that needs to manage or use your company's Oracle Cloud Infrastructure resources. Users might need to launch instances, manage remote disks, work with your virtual cloud network, and so on.
- **Group:** A collection of users who need the same type of access to a particular set of resources or compartment



ORACLE

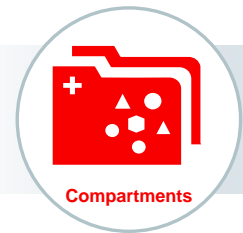
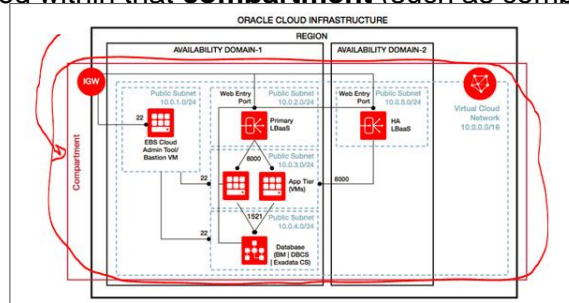
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



## Dynamic Group

- It is a special type of group that contains resources (such as compute instances) that match the rules that you define. (Thus the membership can change dynamically as matching resources are created or deleted.)
- These instances act as "principal" actors and can make API calls to services according to policies that you write for the dynamic group.

**Compartment:** It is a logical container in your account used to store Oracle Cloud Infrastructure (OCI) Resources created within that **compartment** (such as compute, storage, and network).



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Tenancy and Policy

Tenancy is an Oracle Cloud Account given to you when you register for Oracle Cloud Infrastructure (OCI).

## Policy

- A **policy** is a document that specifies who can access which Oracle Cloud Infrastructure resources that your company has, and how. A **policy** simply allows a group.
- Define users, groups, and one or more compartments to hold the cloud resources for your organization.
- Create one or more policies, each written in the policy language.
- Place users into the appropriate groups depending on the compartments and resources they need to work with.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Policy Basics

Allow group `<group_name>` to `<verb>` `<resource-type>` in compartment `<compartment_name>`

In some cases, you'll want the policy to apply to the tenancy and not a compartment inside the tenancy. In that case, change the end of the policy statement as:

Allow group `<group_name>` to `<verb>` `<resource-type>` in tenancy

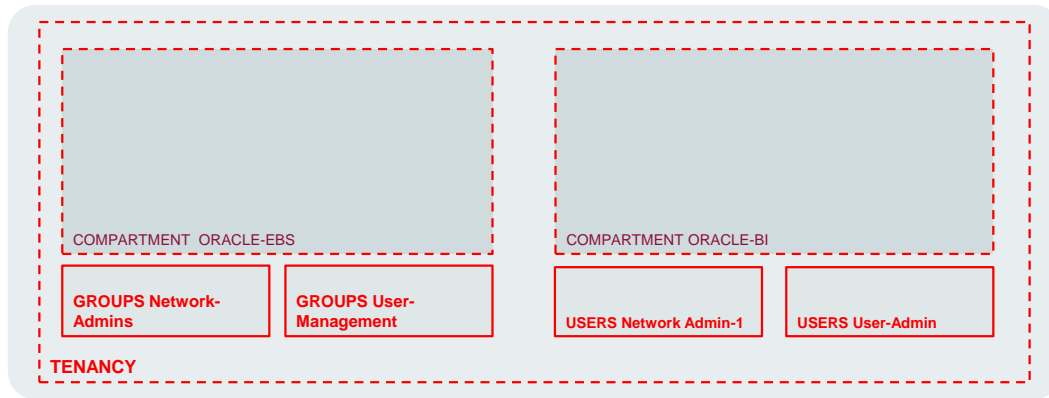
ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Example Case

A company has two or more different applications and you don't want users to access other resources apart from what they should. In this example, you will have Oracle EBS and Oracle BI as example applications. You want to have the network admin group, which manages networks in all compartments and admin groups for each different compartments with specific policies. In addition, you will have a group for user management. Each group will have one user assigned.

This is what compartment, group, and user layout will look like:



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Terraform Resource Syntax

## ## Compartment Creation

```
resource "oci_identity_compartment" "compartment" {  
  name          = "Terraform-compartment"  
  description = "Terraform-compartment"  
}
```

## ## Group Creation

```
resource "oci_identity_group" "test_group" {  
  #Required  
  compartment_id = "xxx"  
  description = "terraform-users-group"  
  name = "terraform-users4"  
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## What Is a Virtual Cloud Network?

- A VCN is a virtual private network that you set up in datacenters. A VCN covers single, continuous IPv4 block of your choice.
- The range that you can use in OCI is /16 through /30. VCN resides within a single region but can cross multiple Availability Domains.
- OCI reserves the first two IP addresses and the last one in each subnet's CIDR. Before proceeding with the creation of VCN, be sure about the size. You can't change the size after you create them.
- Oracle recommends using one of the private IP address ranges in RFC 1918 (10.0.0.0/8, 172.16/12 and 192.168/16) for VCN address space.
- Terraform Module: **oci\_core\_vcn**

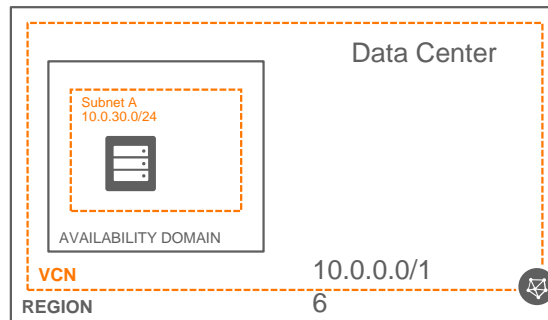


ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Subnet

- Subnet is a logical subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting.
- Each VCN is subdivided into subnets and subnets are contained within a single Availability Domain (AD).
- Subnets can be designed as Public or Private.
- You can have more than one subnet in an AD for a given VCN.



ORACLE

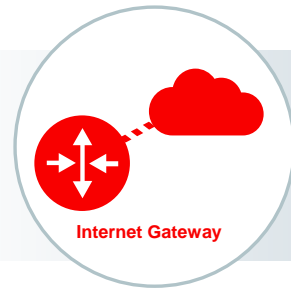
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Internet Gateway

Internet Gateway provides a path for network traffic between your VCN and the Internet.

## Route Table

It is a set of rules, often viewed in table format, that is used to determine where data packets travelling over an Internet protocol network will be directed.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Dynamic Routing Gateway

- To connect an on-premises network to your Virtual Cloud Network (VCN), you use Dynamic Routing Gateway (DRG).
- You can think of a DRG as a virtual router that provides a path for private traffic (that is, traffic that uses private IPv4 addresses) between your VCN and networks outside the VCN's region.
- After attaching a DRG, you must add a route for the DRG in the VCN's route table to enable traffic.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Security Lists

- It is a common set of firewall rules associated with a subnet and applied to all instances launched inside the subnet.
- Security lists provide ingress and egress rules that specify the type of traffic allowed in and out of the instances.
- You can even define whether a given rule is stateful or stateless.



ORACLE

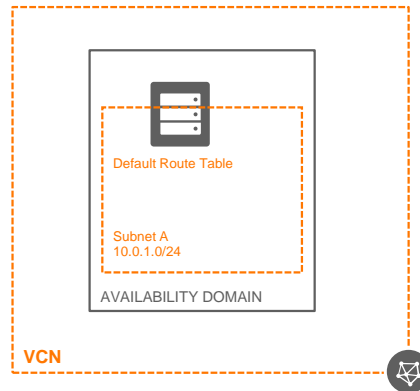
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Default VCN Components

VCN automatically comes with some default components:

- Default Route Table
- Default Security List
- Default set of DHCP options



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Public Versus Private Subnets

- When you create a subnet, by default, it's considered public, which means instances in that subnet are allowed to have public IP addresses. Whoever launches the instance chooses whether it will have a public IP address.
- You can override that behavior when creating the subnet and request that it be private, which means instances launched in the subnet are prohibited from having public IP addresses.
- Network administrators can, therefore, ensure that instances in the subnet have no Internet access, even if the VCN has a working Internet Gateway (IGW), and security lists and firewall rules allow the traffic.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Private IP Versus Public IP

- **Private IP:** Each instance has at least one Primary Private IP address.
  - Example: Database server that can't be accessed over the Internet
- **Public IP:** It is an IPv4 address that is reachable from the Internet; assigned to private IP object on the source (instance, load balancer).
  - Example: Web server that needs to be accessed over the Internet

**Note:** For a public IP to be reachable over the Internet, its VCN must have an IGW, along with a route table and security list configured accordingly.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform Resource Syntax: VCN

```
# Creation of VCN
resource "oci_core_vcn" "test_vcn" {
  #Required
  cidr_block = "10.16.0.0/16"
  compartment_id = "ocidl.compartment.oc1.aaa"
}
```

```
# Creation of a new NAT Gateway
resource "oci_core_nat_gateway" "terraform-NAT-gateway" {
  compartment_id = "ocidl.compartment.oc1"
  display_name = "terraform-NAT-gateway"
  vcn_id = "${oci_core_virtual_network.terraform-vcn.id}"
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform Resource Syntax: VCN

```
# Creation of a subnet
resource "oci_core_subnet" "test_subnet" {
  #Required
  availability_domain = "${var.subnet_availability_domain}"
  cidr_block = "${var.subnet_cidr_block}"
  compartment_id = "${var.compartment_id}"
  security_list_ids = "${var.subnet_security_list_ids}"
  vcn_id = "${oci_core_vcn.test_vcn.id}"
}
```

The reason why you're seeing `var.xxx` is because of using variables instead of adding the values directly

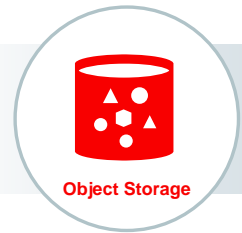
ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Object Storage

Oracle Cloud Infrastructure offers two distinct storage class tiers to address the need:

- **Object Storage:** Use Object Storage for data to which you need fast, immediate, and frequent access. Data accessibility and performance justifies a higher price point to store data in the Object Storage tier.
- **Archive Storage:** Use Archive Storage for data to which you seldom or rarely need access, but that must be retained and preserved for long periods of time. The cost efficiency of the Archive Storage tier offsets the long lead time required to access the data.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Object Storage

- The Object Storage service can store an unlimited amount of unstructured data of any content type, including analytic data and rich content, such as images and videos.
- Object Storage is a regional service and is not tied to any specific compute instance. You can access data from anywhere inside or outside the context of Oracle Cloud Infrastructure, as long as you have Internet connectivity.
- Object Storage also supports private access from Oracle Cloud Infrastructure resources in a VCN through a service gateway.
- A service gateway allows connectivity to the Object Storage public endpoints from private IP addresses in private subnets.
- Object Storage offers a scalable storage platform that lets you store large datasets and operate seamlessly on those datasets.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Backup/Archive

You can use Object Storage to preserve backup and archive data that must be stored for an extended duration to adhere to various compliance mandates.

- **Log Data:** Preserve application log data to retroactively analyze the data
- **Very Large Datasets:** Store generated application data that needs to be preserved for future use



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

### Log Data

You can use Object Storage to preserve application log data so that you can retroactively analyze this data to determine usage pattern and/or debug issues.

### Very Large Datasets

You can use Object Storage to store generated application data that needs to be preserved for future use. Pharmaceutical trials data, genome data, and Internet of Things (IoT) data are examples of generated application data that you can preserve using Object Storage.

## Object Storage Resources

- Object
- Bucket
- Namespace

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

### Object

Any type of data, regardless of content type, is stored as an object. The object is composed of the object itself and metadata about the object. Each object is stored in a bucket.

### Bucket

It is a logical container for storing objects. Users or systems create buckets as needed within a region.

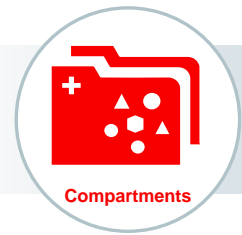
A bucket is associated with a single compartment. The compartment has policies that indicate what actions a user can perform on a bucket and all the objects in the bucket.

### Namespace

It is a logical entity that serves as a top-level container for all buckets and objects, allowing you to control bucket naming within your tenancy. Each tenancy is provided one unique and uneditable Object Storage namespace that is global, spanning all compartments and regions.

# Compartment

- Compartments help you organize resources and make it easier to control access to those resources.
- Object Storage automatically creates a root compartment when a compartment is provisioned.
- An administrator can then create additional compartments within the root compartment.
- A bucket can exist only in one compartment.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

It is a collection of related resources that can be accessed only by users or groups who are explicitly granted access permission by an administrator.

Compartments help you organize resources and make it easier to control access to those resources.

Object Storage automatically creates a root compartment when a compartment is provisioned.

An administrator can then create additional compartments within the root compartment and add access rules for those compartments.

A bucket can exist only in one compartment.

## Ways to Access Object Storage

- Console
- OCI command-line interface (CLI)
- REST API
- Oracle Cloud Infrastructure SDKs

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- The Console is an easy-to-use, browser-based interface. When you sign up to use Oracle Cloud Infrastructure, you receive a customized URL for your organization. For example, <https://console.us-ashburn-1.oraclecloud.com?tenant=CompanyABC>
- The command-line interface (CLI) provides both quick access and full functionality without the need for programming.
- The REST API provides the most functionality, but requires programming expertise. [API Reference and Endpoints](#) provides endpoint details and links to the available API reference documents.
- The Oracle Cloud Infrastructure SDKs offer tools to interact with Object Storage without having to create a framework.

## Terraform Resource Syntax: Object Storage

### # Creation of a new bucket

```
resource "oci_objectstorage_bucket" "terraform-bucket" {  
  compartment_id = "${var.compartment_ocid}"  
  namespace      = "ocuocictrng23"  
  name           = "tf-example-bucket"  
  access_type    = "NoPublicAccess"  
}
```

### # This data source provides the list of Buckets in Oracle Cloud Infrastructure Object Storage service.

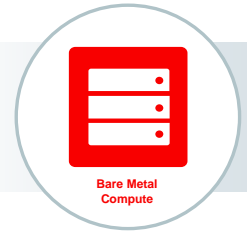
```
data "oci_objectstorage_bucket_summaries" "test_buckets" {  
  #Required  
  compartment_id = "${var.compartment_id}"  
  namespace      = "${var.bucket_namespace}"  
}
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Compute Service: Overview

- Oracle Cloud Infrastructure Compute lets you provision and manage compute hosts, known as instance.
- After you launch an instance, you can access it securely from your computer, restart it, attach and detach volumes, and terminate it when you're done with it.
- Oracle Cloud Infrastructure offers both Bare Metal and Virtual Machine instances.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

### Bare Metal

A bare metal compute instance gives you dedicated physical server access for highest performance and strong isolation.

# Virtual Machine

A Virtual Machine (VM) is an independent computing environment that runs on top of physical bare metal hardware. The virtualization makes it possible to run multiple VMs that are isolated from each other. VMs are ideal for running applications that do not require the performance and resources (CPU, memory, network bandwidth, storage) of an entire physical machine.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Components for Launching Instances

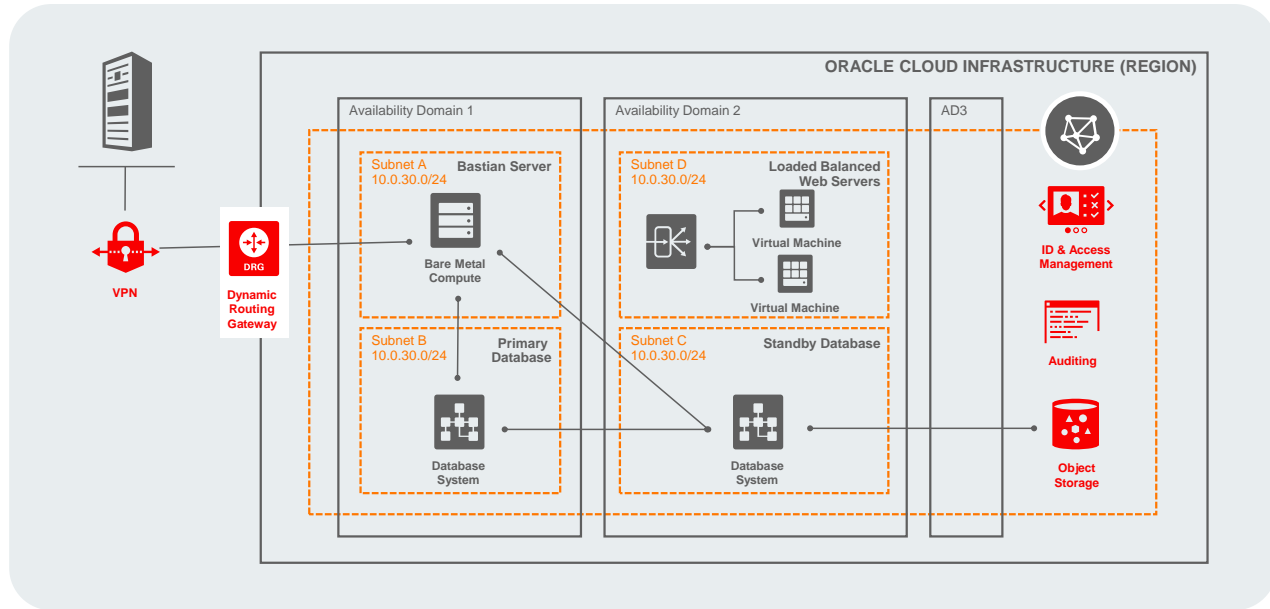
- **Availability domain:** The Oracle Cloud Infrastructure data center within your geographical region that hosts cloud resources, including your instances
- **Virtual Cloud Network:** A virtual version of a traditional network—including subnets, route tables, and gateways—on which your instance runs
- **Key pair (for Linux instances):** A security mechanism required for Secure Shell (SSH) access to an instance
- **Tags:** Can be applied to your resources to help you organize them according to your business needs
- **Image:** A template of a virtual hard drive that determines the operating system and other software for an instance
- **Shape:** A template that determines the number of CPUs, amount of memory, and other resources allocated to a newly created instance
- **Volumes:** The Oracle Cloud Infrastructure Block Volume service lets you dynamically provision and manage block storage volumes.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Sample Diagram: Icons, Groupings, and Connectors

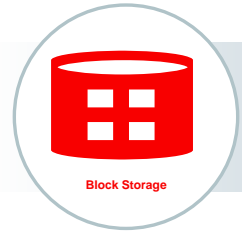


ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Block Volume: Overview

The Oracle Cloud Infrastructure Block Volume service lets you dynamically provision and manage block storage volumes. You can create, attach, connect, and move volumes as needed to meet your storage and application requirements.



**Instance:** A bare metal or virtual machine (VM) host running in the cloud

**Volume attachment:** There are two types of volume attachments:

- **iSCSI:** A TCP/IP-based standard used for communication between a volume and an attached instance
- **Paravirtualized:** A virtualized attachment available for VMs

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Volume Types

- **Block volume:** A detachable block storage device that allows you to dynamically expand the storage capacity of an instance
- **Boot volume:** A detachable boot volume device that contains the image used to boot a Compute instance

### Volume Groups

The Oracle Cloud Infrastructure Block Volume service provides you with the capability to group together multiple volumes in a volume group. A volume group can include both types of volumes, boot volumes, which are the system disks for your Compute instances, and block volumes for your data storage. You can use volume groups to create volume group backups and clones that are point-in-time and crash-consistent.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform Resource Syntax: Compute

### #Creation of a Instance

```
resource "oci_core_instance" "TFInstance" {
  count          = "1"
  availability_domain = "uwRT:EU-FRANKFURT-1-AD-1"
  compartment_id  = "${var.compartment_ocid}"
  display_name    = "TFInstance${count.index}"
  shape           = "VM.Standard2.1"
  image           = "ocidl.image.oc1.eu-frankfurt-
.aaaaaaaagbrvhganmn7awcr7plaa5vhabmzhx763z5afiitswjmzh7upna"
  subnet_id       = "ocidl.subnet.oc1.eu-frankfurt-
1.aaaaaaaayidhlv4zevgrfjz75z6uyypa5iwmvfnb23syqs7xhkykuyitidsoq"
}
```

### # Output the private and public IPs of the instance

```
output "InstancePrivateIPs"
```

```
{
  value = ["${oci_core_instance.TFInstance.*.private_ip}"]
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Terraform Resource Syntax

## # Creation of a new Block storage volume

```
resource "oci_core_volume" "TFBlock" {  
  count                = "1"  
  availability_domain = "uwRT:EU-FRANKFURT-1-AD-1"  
  compartment_id      = "${var.compartment_ocid}"  
  display_name        = "TFBlockusingscript"  
  size_in_gbs         = "50"  
}
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Load Balancing: Overview

- The Oracle Cloud Infrastructure Load Balancing service provides automated traffic distribution from one entry point to multiple servers reachable from your virtual cloud network (VCN).
- A load balancer improves resource utilization, facilitates scaling, and helps ensure high availability.
- Configure load balancing health checks using load balancing policies and application-specific health checks, so that traffic is routed only to healthy instances.
- The load balancer can reduce your maintenance window by draining traffic from an unhealthy application server before you remove it from service for maintenance.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## How Load Balancing Works

- The Load Balancing service enables you to create a public or private load balancer within your VCN.
- A public load balancer has a public IP address that is accessible from the Internet.
- A private load balancer has an IP address from the hosting subnet, which is visible only within your VCN.
- Both public and private load balancers can route data traffic to any back-end server that is reachable from the VCN.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



## Public Load Balancer

- To accept traffic from the Internet, you create a public load balancer.
- The service assigns it a public IP address that serves as the entry point for incoming traffic.
- A public load balancer is regional in scope. If your region includes multiple availability domains, a public load balancer requires two subnets; each in a separate availability domain.

**Note:** You cannot specify a [private subnet](#) for your public load balancer.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Private Load Balancer

- To isolate your load balancer from the Internet and simplify your security posture, you can create a private load balancer.
- The Load Balancing service assigns it a private IP address that serves as the entry point for incoming traffic.
- When you create a private load balancer, the service requires only one subnet to host both the primary and standby load balancers.
- The load balancer is accessible only from within the VCN that contains the associated subnet, or as further restricted by your security list rules.

**Note:** You cannot specify a [private subnet](#) for your public load balancer.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

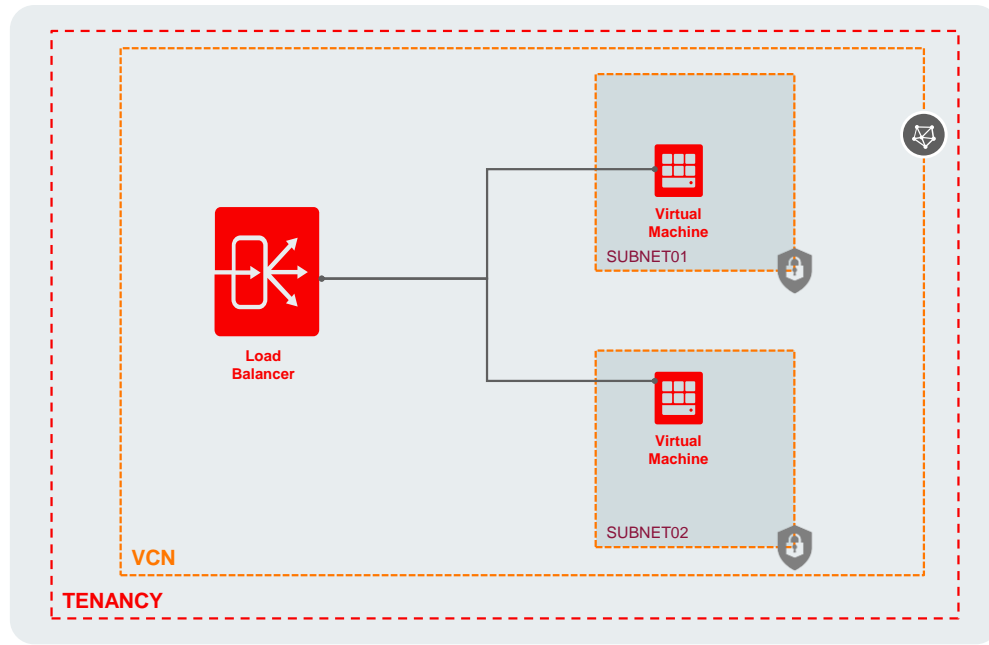
## All Load Balancers

- Your load balancer has a back-end set to route incoming traffic to your Compute instances. The back-end set is a logical entity that includes:
  - A list of back-end servers
  - A load balancing policy
  - A health check policy
  - Optional SSL handling
  - Optional session persistence configuration
- Oracle recommends that you distribute your back-end servers across all availability domains within the region.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Simple Architecture Diagram



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Load Balancing Concepts

- Back-end servers
- Back-end set
- Certificates
- Health check
- Health status
- Listener
- Load balancing policy
- Path route set
- Regions and Availability Domains
- Session Persistence
- Shape
- VCN
- Subnet
- SSL
- Visibility

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## Terraform Resource Syntax: Load Balancing

```
resource "oci_load_balancer_load_balancer" "test_load_balancer" {  
  #Required  
  compartment_id = "${var.compartment_id}"  
  display_name = "${var.load_balancer_display_name}"  
  shape = "${var.load_balancer_shape}"  
  subnet_ids = "${var.load_balancer_subnet_ids}"  
  #Optional  
  defined_tags = {"Operations.CostCenter"= "42"}  
  freeform_tags = {"Department"= "Finance"}  
  is_private = "${var.load_balancer_is_private}"  
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The reason why you're seeing `var.compartment_id` is because we're using variables instead of updating the values directly.

## File Storage: Overview

- Oracle Cloud Infrastructure File Storage service provides a durable, scalable, secure, enterprise-grade network file system. You can connect to a File Storage service file system from any bare metal, virtual machine, or container instance in your VCN.
- If you're planning to access outside of VCN, then you need to go with VCN using Oracle Cloud Infrastructure FastConnect and Internet Protocol security (IPSec) virtual private network (VPN).
- The File Storage service supports the Network File System version 3.0 (NFSv3) protocol. The service supports the Network Lock Manager (NLM) protocol for file locking functionality.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The File Storage service is designed to meet the needs of applications and users that need an enterprise file system across a wide range of use cases, including the following:

- **General Purpose File Storage:** Access an unlimited pool of file systems to manage growth of structured and unstructured data
- **Big Data and Analytics:** Run analytic workloads and use shared file systems to store persistent data.
- **Lift and Shift of Enterprise Applications:** Migrate existing Oracle applications that need NFS storage, such as Oracle E-Business Suite and PeopleSoft.
- **Databases and Transactional Applications:** Run test and development workloads with Oracle, MySQL, or other databases.
- **Backups, Business Continuity, and Disaster Recovery:** Host a secondary copy of relevant file systems from on-premises to the cloud for backup and disaster recovery purposes.
- **MicroServices and Docker:** Deliver stateful persistence for containers. Easily scale as your container-based environments grow.

# File Systems Concepts

- **Mount Target:** An NFS endpoint that lives in a subnet of your choice and is highly available
- **Export:** Controls how NFS clients access file systems when they connect to a mount target
- **Export Set:** A collection of one or more exports that control what file systems the mount target exports using the NFSv3 protocol and how those file systems are found using the NFS mount protocol

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Mount Target

Mount target is an NFS endpoint that lives in a subnet of your choice and is highly available. The mount target provides the IP address or DNS name that is used in the mount command when connecting NFS clients to a file system. A single mount target can export many file systems.

## Export

Exports control how NFS clients access file systems when they connect to a mount target. File systems are exported (made available) through mount targets. Each mount target maintains an export set which contains one or many exports. A file system must have at least one export in one mount target in order for instances to mount the file system.

## Export Set

An export set is a collection of one or more exports that control what file systems the mount target exports using the NFSv3 protocol and how those file systems are found using the NFS mount protocol.



# File Systems Concepts

- **Export Options:** A set of parameters within the export that specify the level of access granted to NFS clients when they connect to a mount target
- **Snapshots:** Provide a consistent, point-in-time view of your file system, and you can take as many snapshots as you need
- **Encryption:** All files encrypted at rest by default

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Export Options

NFS export options are a set of parameters within the export that specify the level of access granted to NFS clients when they connect to a mount target. An NFS export options entry within an export defines access for a single IP address or CIDR block range.

## Snapshots

Snapshots provide a consistent, point-in-time view of your file system, and you can take as many snapshots as you need. You pay only for the storage used by your data and metadata, including storage capacity used by snapshots.

## How File Storage Permissions Work

- File Storage service resources include file systems, mount targets, and export sets. The AUTH\_UNIX style of authentication and permission checking is supported for remote NFS client requests.
- Use Oracle Cloud Infrastructure Identity and Access Management (IAM) policy language to define access to Oracle Cloud Infrastructure resources.
- Oracle Cloud Infrastructure users require resource permissions to create, delete, and manage resources. Without the appropriate IAM permissions, you cannot export a file system through a mount target.
- Until a file system has been exported, it cannot be mounted by Compute instances.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Terraform Resource Syntax: EFS

### ##Creation of filesystem

```
resource "oci_file_storage_file_system" "test_file_system" {  
  #Required  
  availability_domain = "3"  
  compartment_id = "ocidl.compartment.oc1.."  
}
```

### ##Creation of storage export and mount target

```
resource "oci_file_storage_export" "test_export" {  
  #Required  
  export_set_id =  
    "${oci_file_storage_mount_target.test_mount_target.export_set_id}"  
  file_system_id = "${oci_file_storage_file_system.test_file_system.id}"  
  path = "${var.export_path}"  
}
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Summary

In this lesson, you should have learned to describe and create various resources on OCI using Terraform.



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



# Jenkins

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Objectives



After completing this lesson, you should be able to describe:

- Jenkins
- The advantages of Jenkins
- Continuous integration

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Prerequisites

## Prerequisites

To successfully complete this course:

- General understanding of Software Development Life Cycle (SDLC)

## Suggested prerequisites

- General understanding of Continuous Integration and Continuous Deployment



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## What Is Jenkins?

- Jenkins is an open source automation tool written in Java with plug-ins built for Continuous Integration purpose.
- It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.
- Jenkins achieves Continuous Integration with the help of plug-ins. Plug-ins allow the integration of Various DevOps stages.
- If you want to integrate a particular tool, you need to install the plug-ins for that tool; for example, Git, Maven 2 project, OCI Plugin, HTML publisher, and so on.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



## Advantages of Jenkins

- An open source tool with great community support
- Easy to install
- Has 1000+ plug-ins to ease your work. If a plug-in does not exist, you can code it and share with the community.
- Free of cost
- Built with Java and, therefore, is portable to all the major platforms



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Disadvantages of the Waterfall Model

In the traditional Waterfall Model, developers used to develop the code and then deploy it on the test server for testing.

This approach has several disadvantages:

- Developers have to wait till the complete software is developed for the test results.
- There is a high possibility that the test results might show multiple bugs.
- It's tough for developers to locate bugs because they have to check the entire source code of the application. This slows the software delivery process.
- The whole process is manual, which increases the risk of frequent failure.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## What Is Continuous Integration (CI)?

- Continuous Integration is a development practice in which developers are required to commit changes to the source code in a shared repository several times a day or more frequently.
- Every commit made in the repository is then built.



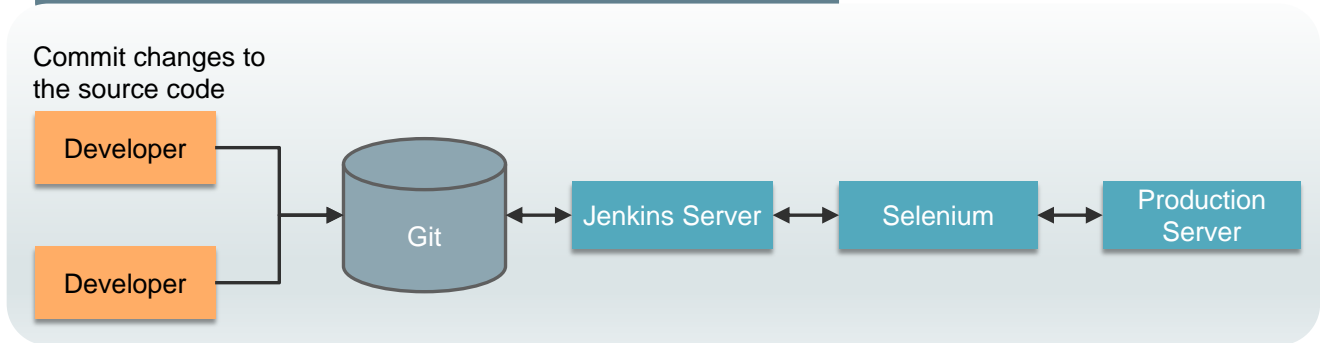
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Continuous Integration with Jenkins

With the traditional model, not only does the software delivery process becomes slow but the quality of software also goes down. To overcome these issues, you need a tool to continuously integrate everything.

This is what CI is all about. Jenkins is the most mature CI tool available. So let us see how Continuous Integration with Jenkins overcomes shortcomings of the traditional model.

Generic flow diagram of Continuous Integration with Jenkins



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide depicts the following functions:

- First, a developer commits the code to the source code repository.
- Meanwhile, the Jenkins server checks the repository at regular intervals for changes.
- Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull those changes and will start preparing a new build.
- If the build fails, then the concerned team will be notified.
- If the build is successful, then Jenkins deploys the build in the test server.
- After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.
- It will continue to check the source code repository for changes made in the source code and the whole process keeps on repeating.

# Impact of Jenkins

## Before Jenkins

The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time consuming, which in turn slows the software delivery process.

Developers have to wait for test results.

The whole process is manual.

## Post Jenkins

Every commit made in the source code is built and tested. So, instead of checking the entire source code developers need to focus only on a particular commit. This leads to frequent new software releases.

Developers know the test result of every commit made in the source code on the run.

You only need to commit changes to the source code and Jenkins will automate the rest of the process for you.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Jenkins Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through the TCP/IP protocol.

## Jenkins Master

Scheduling build jobs.

- Dispatching builds to the slaves for the actual execution
- Monitoring the slaves (possibly taking them online and offline as required)

## Jenkins Slave

- A slave is a Java executable that runs on a remote machine.
- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Summary

In this lesson, you should have learned how to describe:

- Jenkins
- The advantages of Jenkins
- Continuous integration



ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

