KPLABS Course

HashiCorp Certified: Terraform Associate

Important Pointers for Exams

ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

Pointer 1: Terraform Providers

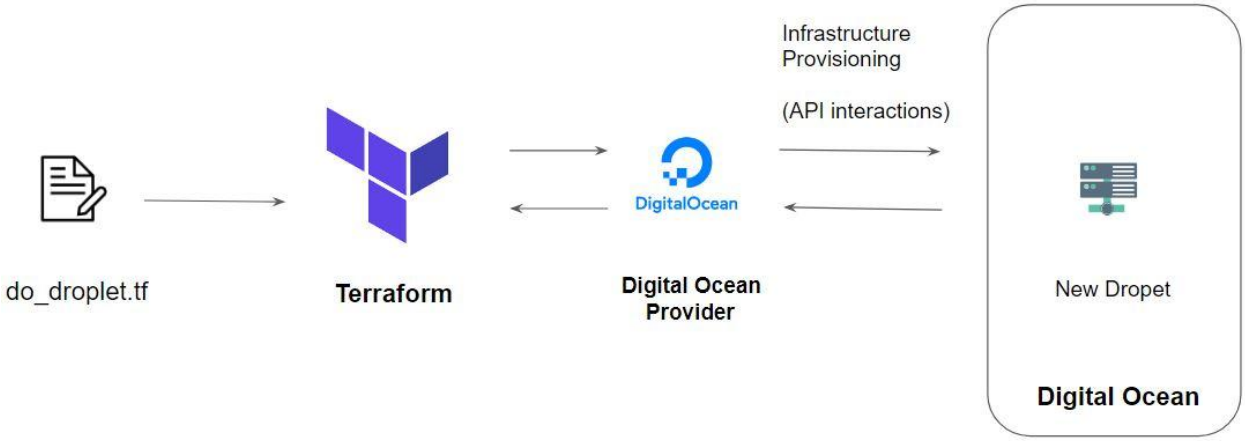
A provider is responsible for understanding API interactions and exposing resources.

Most of the available providers correspond to one cloud or on-premises infrastructure platform and offer resource types that correspond to each of the features of that platform.

You can explicitly set a specific version of the provider within the provider block.

To upgrade to the latest acceptable version of each provider, run terraform init -upgrade

Following is the high-level architecture of the provider.



Pointer 2 - alias: Multiple Provider Instances

You can have multiple provider instances with the help of an alias

```
provider "aws" {
  region = "us-east-1"
}
provider "aws" {
  alias  = "west"
  region = "us-west-2"
}
```

The provider block without alias set is known as the default provider configuration. When an alias is set, it creates an additional provider configuration.

Pointer 3 - Terraform Init

The terraform init command is used to initialize a working directory containing Terraform configuration files.

During init, the configuration is searched for module blocks, and the source code for referenced modules is retrieved from the locations given in their source arguments.
Terraform must initialize the provider before it can be used.

Initialization downloads and installs the provider's plugin so that it can later be executed.

It will not create any sample files like example.tf

Pointer 4 - Terraform Plan

The terraform plan command is used to create an execution plan.

It will not modify things in infrastructure.

Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files.

This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state.

Pointer 5 - Terraform Apply

The terraform apply command is used to apply the changes required to reach the desired state of the configuration.

Terraform apply will also write data to the terraform.tfstate file.

Once apply is completed, resources are immediately available.

Pointer 6 - Terraform Refresh

The terraform refresh command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure.

This does not modify infrastructure but does modify the state file.

Pointer 7 - Terraform Destroy

The terraform destroy command is used to destroy the Terraform-managed infrastructure.

terraform destroy command is not the only command through which infrastructure can be destroyed.

Pointer 8 - Terraform Format

The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style.

For use-case, where the all configuration written by team members needs to have a proper style of code, terraform fmt can be used.

Pointer 9 - Terraform Validate

The terraform validate command validates the configuration files in a directory.

Validate runs checks that verify whether a configuration is syntactically valid and thus primarily useful for general verification of reusable modules, including the correctness of attribute names and value types.

It is safe to run this command automatically, for example, as a post-save check in a text editor or as a test step for a reusable module in a CI system. It can run before terraform plan.

Validation requires an initialized working directory with any referenced plugins and modules installed

Pointer 10 - Terraform Provisioners

Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

Provisioners should only be used as a last resort. For most common situations, there are better alternatives.

Provisioners are inside the resource block.

Have an overview of local and remote provisioner

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command = "echo The server's IP address is ${self.private_ip}"
  }
}
```

Pointer 11 - Debugging In Terraform

Terraform has detailed logs that can be enabled by setting the **TF_LOG** environment variable to any value.

You can set TF_LOG to one of the log levels TRACE, DEBUG, INFO, WARN or ERROR to change the verbosity of the logs.

Example:

TF_LOG=TRACE

To persist logged output, you can set **TF_LOG_PATH**

Pointer 12 - Terraform Import

Terraform is able to import existing infrastructure.

This allows you to take resources that you've created by some other means and bring it under Terraform management.

The current implementation of Terraform import can only import resources into the state. It does not generate configuration.

Because of this, prior to running terraform import, it is necessary to write a resource configuration block manually for the resource, to which the imported object will be mapped.

terraform import aws_instance.myec2 instance-id

Pointer 13 - Local Values

A local value assigns a name to an expression, allowing it to be used multiple times within a module without repeating it.

The expression of a local value can refer to other locals, but as usual reference cycles are not allowed. That is, a local cannot refer to itself or to a variable that refers (directly or indirectly) back to it.

It's recommended to group together logically-related local values into a single block, particularly if they depend on each other.

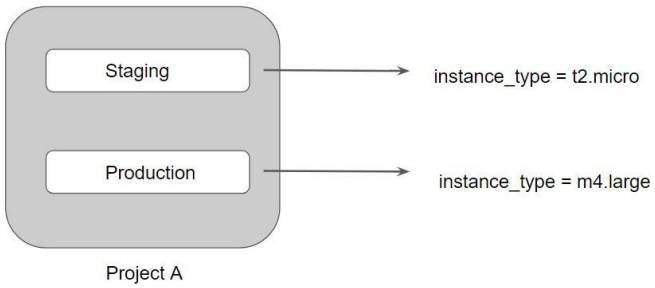
Pointer 14 - Overview of Data Types

Type Keywords	Description
string	Sequence of Unicode characters representing some text, like "hello".
list	Sequential list of values identified by their position. Starts with 0 [“mumbai” ,”singapore”, ”usa”]
map	a group of values identified by named labels, like {name = "Mabel", age = 52}.
number	Example: 200

Pointer 15 - Terraform Workspace

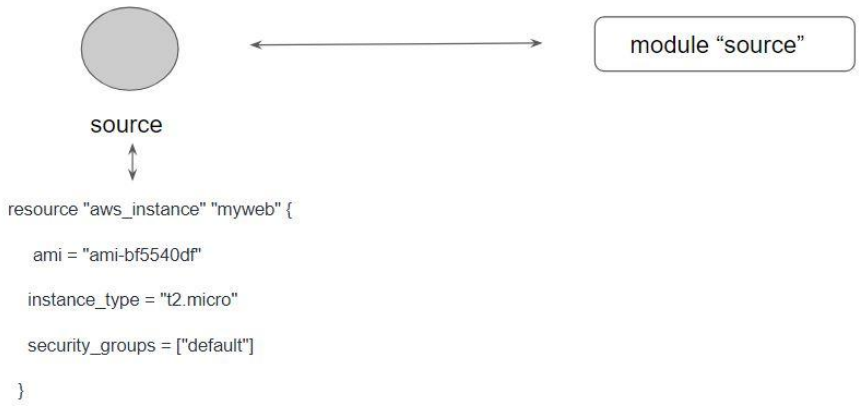
Terraform allows us to have multiple workspaces; with each of the workspaces, we can have a different set of environment variables associated.

Workspaces allow multiple state files of a single configuration.



Pointer 16 - Terraform Modules

We can centralize the terraform resources and can call out from TF files whenever required.



Pointer 17 - ROOT and Child Modules

Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory.

A module can call other modules, which lets you include the child module's resources into the configuration in a concise way.
A module that includes a module block like this is the calling module of the child module.

```
module "servers" {
  source = "./app-cluster"

  servers = 5
}
```

Pointer 18 - Accessing Output Values in Modules

The resources defined in a module are encapsulated, so the calling module cannot access their attributes directly.

However, the child module can declare output values to selectively export certain values to be accessed by the calling module.

A module includes a module block like this is the calling module of the child module.

```
output "instance_ip_addr" {
  value = aws_instance.server.private_ip
}
```

Pointer 19 -Suppressing Values in CLI Output

An output can be marked as containing sensitive material using the optional sensitive argument:

```
output "db_password" {
  value       = aws_db_instance.db.password
  description = "The password for logging in to the database."
  sensitive   = true
}
```

Setting an output value in the root module as sensitive prevents Terraform from showing its value in the list of outputs at the end of terraform apply

Sensitive output values are still recorded in the state, and so will be visible to anyone who is able to access the state data.

Pointer 20 - Module Versions

It is recommended to explicitly constraining the acceptable version numbers for each external module to avoid unexpected or unwanted changes.

Version constraints are supported only for modules installed from a module registry, such as the Terraform Registry or Terraform Cloud's private module registry.

```
module "consul" {
  source  = "hashicorp/consul/aws"
  version = "0.0.5"

  servers = 3
}
```

Pointer 21 - Terraform Registry

The Terraform Registry is integrated directly into Terraform.

The syntax for referencing a registry module is

<NAMESPACE>/<NAME>/<PROVIDER>.

For example hashicorp/consul/aws

```
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0"
}
```

Pointer 22 - Private Registry for Module Sources

You can also use modules from a private registry, like the one provided by Terraform Cloud.

Private registry modules have source strings of the following form: <HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>.

This is the same format as the public registry, but with an added hostname prefix.

While fetching a module, having a version is required.

```
module "vpc" {
  source = "app.terraform.io/example_corp/vpc/aws"
  version = "0.9.3"
}
```

Pointer 23 - Terraform Functions

The Terraform language includes a number of built-in functions that you can use to transform and combine values.

> max(5, 12, 9)
12

The Terraform language does not support user-defined functions, and so only the functions built into the language are available for use

Be aware of basic functions like element, lookup.

Pointer 24 - Count and Count Index

The count parameter on resources can simplify configurations and let you scale resources by simply incrementing a number.

In resource blocks where the count is set, an additional count object (count.index) is available in expressions, so that you can modify the configuration of each instance.

```
resource "aws_iam_user" "lb" {
  name = "loadbalancer.${count.index}"
  count = 5
  path = "/system/"
}
```

Pointer 25 - Find the Issue Use-Case

You can expect use-case with terraform code, and you have to find what should be removed as part of Terraform best practice.

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
    access_key = 1234
    aecret_key = 1234567890
  }
  }
```

Pointer 26 - Terraform Lock

If supported by your backend, Terraform will lock your state for all operations that could write state.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

Pointer 27 - Use-Case - Resources Deleted Out of Terraform

You have created an EC2 instance. Someone has modified the EC2 instance manually. What will happen if you do terraform plan yet again?

- Someone has changed EC2 instance type from t2.micro to t2.large?
- Someone has terminated the EC2 instance.

Answer 1. Terraform's current state will have t2.large, and the desired state is t2.micro. It will try to change back instance type to t2.micro.

Answer 2. Terraform will create a new EC2 instance.

Pointer 28 - Resource Block

Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records.

A resource block declares a resource of a given type ("aws_instance") with a given local name ("web").

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

Pointer 29 - Sentinel

Sentinel is an embedded policy-as-code framework integrated with the HashiCorp Enterprise products.

Can be used for various use-cases like:

- Verify if EC2 instance has tags.
- Verify if the S3 bucket has encryption enabled.

Pointer 30 - Sensitive Data in State File

If you manage any sensitive data with Terraform (like database passwords, user passwords, or private keys), treat the state itself as sensitive data.

Approaches in such a scenario:

Terraform Cloud always encrypts the state at rest and protects it with TLS in transit. Terraform Cloud also knows the identity of the user requesting state and maintains a history of state changes.

The S3 backend supports encryption at rest when the encrypt option is enabled.

Pointer 31 - Dealing with Credentials in Config

Hard-coding credentials into any Terraform configuration are not recommended, and risks the secret leakage should this file ever be committed to a public version control system.

You can store the credentials outside of terraform configuration.

Storing credentials as part of environment variables is also a much better approach than hard coding it in the system.

Pointer 32 - Remote Backend for Terraform Cloud

The remote backend stores Terraform state and may be used to run operations in Terraform Cloud.

When using full remote operations, operations like terraform plan or terraform apply can be executed in Terraform Cloud's run environment, with log output streaming to the local terminal.

Pointer 33 - Miscellaneous Pointers

Terraform does not require go as a prerequisite.

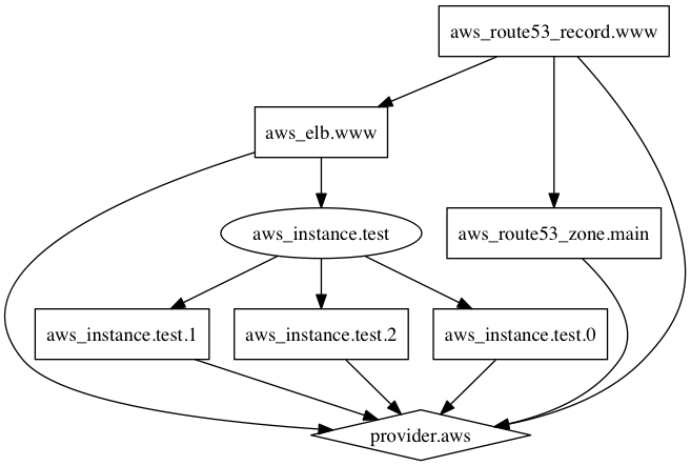
It works well in Windows, Linux, MAC.

Windows Server is not mandatory.

Pointer 34 - Terraform Graph

The terraform graph command is used to generate a visual representation of either a configuration or execution plan

The output of terraform graph is in the DOT format, which can easily be converted to an image



Pointer 35 - Terraform Graph

Splat Expression allows us to get a list of all the attributes.

```
resource "aws_iam_user" "lb" {
  name = "iamuser.${count.index}"
  count = 3
  path = "/system/"
}

output "arns" {
  value = aws_iam_user.lb[*].arn
}
```

The documentation referred to during the video:

https://www.terraform.io/docs/configuration/expressions.html

Pointer 36 - Provider Configuration

Provider Configuration block is not mandatory for all the terraform configuration.

```
provider "aws" {
  region     = "us-east-1"
  access_key = "YOUR-KEY"
  secret_key = "YOUR-KEY"
}

resource "aws_iam_user" "iam" {
  name = "iamuser"
  path = "/system/"
}
```

```
locals {
  arr = ["value1", "value2"]
}

output "test" {
  value = local.arr
}
```

Pointer 37 - Terraform Output

The terraform output command is used to extract the value of an output variable from the state file.

```
C:\Users\Zeal Vora\Desktop\terraform\terraform output>terraform output iam_names
[
  "iamuser.0",
  "iamuser.1",
  "iamuser.2",
]
```

Pointer 38 - Terraform Unlock

If supported by your backend, Terraform will lock your state for all operations that could write state.

Not all backends support locking functionality.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

terraform force-unlock LOCK_ID [DIR]

Pointer 39 - Miscellaneous Pointers - 1

There are three primary benefits of Infrastructure as Code tools:

Automation, Versioning, and Reusability.

Various IAC Tools Available in the market:

- Terraform
- CloudFormation
- Azure Resource Manager
- Google Cloud Deployment Manager

Pointer 40 - Miscellaneous Pointers - 2

Sentinel is a proactive service.

Terraform Refresh does not modify the infrastructure but it modifies the state file.

Slice Function is not part of the string function. Others like join, split, chomp are part of it.

It is not mandatory to include the module version argument while pulling the code from terraform registry.

Pointer 41 - Miscellaneous Pointers - 3

The overuse of dynamic blocks can make configuration hard to read and maintain.

Terraform Apply can change, destroy, and provision resources but cannot import any resource.

Pointer 42 - Terraform Enterprise & Terraform Cloud

Terraform Enterprise provides several added advantages compared to Terraform Cloud.

Some of these include:

- Single Sign-On
- Auditing
- Private Data Center Networking
- Clustering

Team & Governance features are not available for Terraform Cloud Free (Paid)

Pointer 43 - Variables with undefined values

If you have variables with undefined values, it will not directly result in an error.

Terraform will ask you to supply the value associated with them.

Example Code:

variable custom_var { }

```
C:\Users\Zeal Vora\Desktop\terraform\tmp\1>terraform plan
var.custom_var
  Enter a value:
```

Pointer 44 - Environment Variables

Environment variables can be used to set variables.

The environment variables must be in the format TF_VAR_name

```
export TF_VAR_region=us-west-1

export TF_VAR_ami=ami-049d8641

export TF_VAR_alist='[1,2,3]'
```

Pointer 45 - Structural Data Types

A structural type allows multiple values of several distinct types to be grouped together as a single value.

List contains multiple values of the same type while objects can contain multiple values of different types.

Structural Type	Description

object	A collection of named attributes that each have their own type.
	object({<ATTR NAME> = <TYPE>, ... })
	object({ name=string, age=number })
	
	{
	name = "John"
	age = 52
	}

tuple	tuple([<TYPE>, ...])

Pointer 46 - BackEnd Configuration

Backends are configured directly in Terraform files in the terraform section.

After configuring a backend, it has to be initialized.

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

46.1 First-Time Configuration

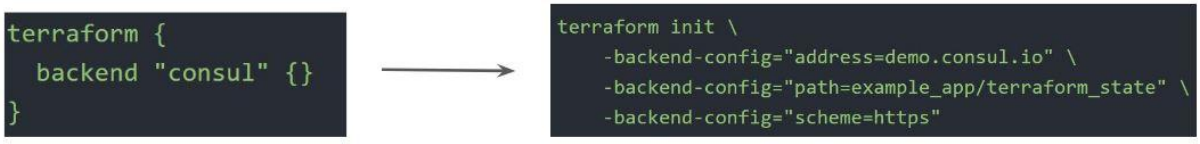
When configuring a backend for the first time (moving from no defined backend to explicitly configuring one), Terraform will give you the option to migrate your state to the new backend.

This lets you adopt backends without losing any existing state.

46.1 Partial Time Configuration

You do not need to specify every required argument in the backend configuration. Omitting certain arguments may be desirable to avoid storing secrets, such as access keys, within the main configuration.

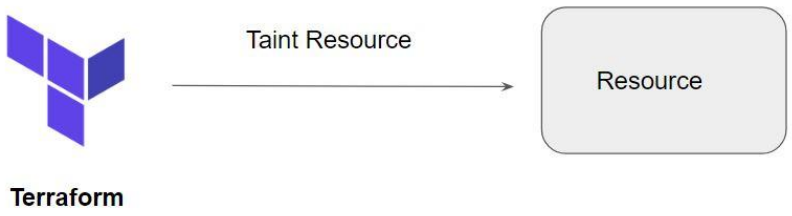
With a partial configuration, the remaining configuration arguments must be provided as part of the initialization process.



Pointer 47 - Terraform Taint

The terraform taint command manually marks a Terraform-managed resource as tainted, forcing it to be destroyed and recreated on the next apply.

Once a resource is marked as tainted, the next plan will show that the resource will be destroyed and recreated and the next apply will implement this change.



Terraform Taint can also be used to taint resources within a module. terraform taint [options] address

```
$ terraform taint "module.couchbase.aws_instance.cb_node[9]"
Resource instance module.couchbase.aws_instance.cb_node[9] has been marked as tainted.
```

For multiple sub-modules, the following syntax-based example can be used

module.foo.module.bar.aws_instance.aex

Pointer 48 - Terraform Provisioners

48.1 local-exec provisioner

The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource

```
resource "aws_instance" "web" {

  provisioner "local-exec" {
    command = "echo ${aws_instance.web.private_ip} >> private_ips.txt"
  }
}
```

48.2 remote-exec provisioner

The remote-exec provisioner invokes a script on a remote resource after it is created.

The remote-exec provisioner supports both ssh and winrm type connections.

```
resource "aws_instance" "web" {

  provisioner "remote-exec" {
    inline = [
      "yum -y install nginx"
      "yum -y install nano"
    ]
  }
}
```

48.3 Failure Behavior in Provisioners

By default, provisioners that fail will also cause the terraform apply itself to fail.

The on_failure setting can be used to change this. The allowed values are:

Allowed Values	Description
continue	Ignore the error and continue with creation or destruction.
fail	Raise an error and stop applying (the default behavior). If this is a creation provisioner, taint the resource.

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command    = "echo The server's IP address is ${self.private_ip}"
    on_failure = continue
  }
}
```

48.4 Provisioner Types

There are two primary types of provisioners:

Types of Provisioners	Description
Creation-Time Provisioner	Creation-time provisioners are only run during creation, not during updating or any other lifecycle If a creation-time provisioner fails, the resource is marked as tainted.
Destroy-Time Provisioner	Destroy provisioners are run before the resource is destroyed.

Pointer 49 - Input Variables

The value associated with a variable can be assigned via multiple approaches.

```
variable "image_id" {
  type = string
}
```

The value associated with the variables can be defined via CLI as well as in tfvars file.

Following is syntax to load custom tfvars file:

terraform apply -var-file="testing.tfvars"

Pointer 50 - Variable Definition Precedence

Terraform loads variables in the following order, with later sources taking precedence over earlier ones:

- Environment variables
- The terraform.tfvars file, if present.
- The terraform.tfvars.json file, if present.
- Any *.auto.tfvars or *.auto.tfvars.json files, processed in lexical order of their filenames.
- Any -var and -var-file options on the command line, in the order they are provided.

If the same variable is assigned multiple values, Terraform uses the last value it finds.

Pointer 51 - Terraform Local Backend

The local backend stores state on the local filesystem locks that state using system APIs and perform operations locally.

By default, Terraform uses the "local" backend, which is the normal behavior of Terraform you're used to

```
terraform {
  backend "local" {
    path = "relative/path/to/terraform.tfstate"
  }
}
```

Pointer 51 - Required Providers

Each Terraform module must declare which providers it requires so that Terraform can install and use them.

Provider requirements are declared in a required_providers block.

```
terraform {
  required_providers {
    mycloud = {
      source  = "mycorp/mycloud"
      version = "~> 1.0"
    }
  }
}
```

Pointer 52 - Required Version

The required_version setting accepts a version constraint string, which specifies which versions of Terraform can be used with your configuration.

If the running version of Terraform doesn't match the constraints specified, Terraform will produce an error and exit without taking any further actions.

```
terraform {
  required_version = "> 0.12.0"
}
```

Pointer 53 - Versioning Arguments

There are multiple ways of specifying the version of a provider.

Version Number Arguments	Description
>=1.0	Greater than equal to the version
<=1.0	Less than equal to the version
~>2.0	Any version in the 2.X range.
>=2.10,<=2.30	Any version between 2.10 and 2.30

,

Pointer 54 - Fetching Values from Map

To reference to image-abc from the below map, the following approaches needs to be used:

var.ami_ids[“mumbai”]

```
variable "ami_ids" {
   type = "map"
   default = {
       "mumbai" = "image-abc"
       "germany" = "image-def"
       "states"  = "image-xyz"
    }
}
```

Pointer 55 - Terraform and GIT

55.1 - Terraform and .gitignore

If you are making use of the GIT repository for committing terraform code, the .gitignore should be configured to ignore certain terraform files that might contain sensitive data.

Some of these can include:

terraform.tfstate file (this can include sensitive information)

*.tfvars (may contain sensitive data like passwords)

.

55.2 Integrating Terraform with GIT

Arbitrary Git repositories can be used by prefixing the address with the special git:: prefix.

After this prefix, any valid Git URL can be specified to select one of the protocols supported by Git.

```
module "vpc" {
  source = "git::https://example.com/vpc.git"
}

module "storage" {
  source = "git::ssh://username@example.com/storage.git"
}
```

By default, Terraform will clone and use the default branch (referenced by HEAD) in the selected repository.

You can override this using the ref argument:

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

.
The value of the ref argument can be any reference that would be accepted by the git checkout command, including branch and tag names.

,

Pointer 56 - Terraform Workspace

Workspaces are managed with the terraform workspace set of commands.

State File Directory = terraform.tfstate.d

Not suitable for isolation for strong separation between workspace (stage/prod)

Use-Case	Command
Create New Workspace	terraform workspace new kplabs
Switch to a specific Workspace	terraform workspace select prod

```
$ terraform workspace new bar
Created and switched to workspace "bar"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
```

Pointer 57 - Dependency Types

57.1 Implicit Dependency

With implicit dependency, Terraform can automatically find references of the object and create an implicit ordering requirement between the two resources.

```
resource "aws_eip" "my_eip"{
  vpc = "true"
}

resource "aws_instance" "my_ec2" {
   instance_type = "t2.micro"
   public_ip     = aws_eip.myeip.private_ip
}
```

57.2 - Explicit Dependency

Explicitly specifying a dependency is only necessary when a resource relies on some other resource's behavior but doesn't access any of that resource's data in its arguments.

```
resource "aws_s3_bucket" "example" {
  acl    = "private"
}

resource "aws_instance" "myec2" {
   instance_type = "t2.micro"
   depends_on = [aws_s3_bucket.example]
}
```

.

Pointer 58- Terraform State Commands

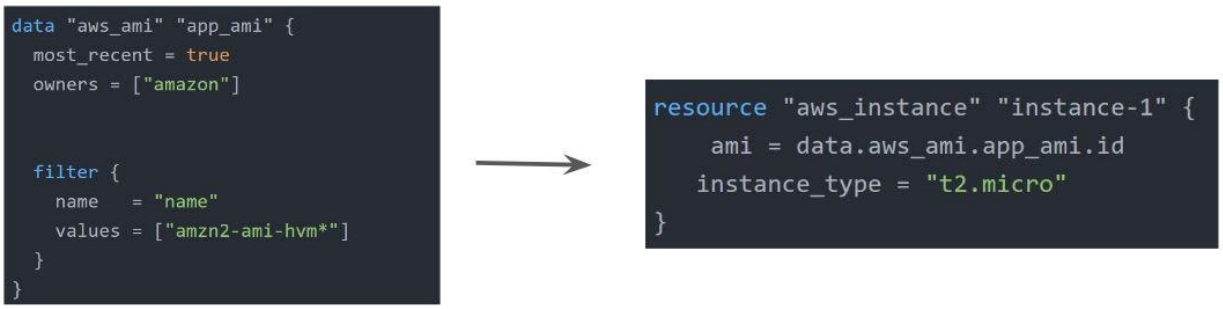
Rather than modify the state directly, the terraform state commands can be used in many cases instead.

State Command	Description
terraform state list	List resources within terraform state
terraform state mv	Move items within terraform state. Can be used to resource renaming.
terraform state pull	manually download and output the state from state file.
terraform state rm	Remove Items from terraform state file.
Terraform state show	Show the attributes of a single resource in the Terraform state.

Pointer 59 - Data Source

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.

Reads from a specific data source (aws_ami) and exports results under “app_ami”



Pointer 60 - Terraform Plan Destroy

The behavior of any terraform destroy command can be previewed at any time with an equivalent terraform plan -destroy command.

```
C:\Users\Zeal Vora\Desktop\terraform\tmp\1>terraform plan -destroy
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:

Terraform will perform the following actions:

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

Changes to Outputs:
  - test = [
      - "host1",
      - "host2",
      - "host3",
    ] -> null
```

Pointer 61 - Terraform Module Sources

The module installer supports installation from a number of different source types like Local paths, Terraform Registry, GitHub, S3 buckets, and others.

Local path references allow for factoring out portions of a configuration within a single source repository.

A local path must begin with either ./ or ../ to indicate that a local path is intended.

```
module "consul" {
  source = "../consul"
}
```

Pointer 62 - Dealing with Larger Infrastructure

Cloud Providers have a certain amount of rate-limiting set so Terraform can only request a certain amount of resources over a period of time.

It is important to break larger configurations into multiple smaller configurations that can be independently applied.

Alternatively, you can make use of -refresh=false and -target=resource flag for a workaround (not recommended)

Pointer 63 - Miscellaneous Pointers

1. lookup retrieves the value of a single element from a map - lookup(map, key, default)
2. Various commands run terraform refresh implicitly, some of these include:

   terraform [plan, apply, destroy]

Others like terraform [init, import] do not run refresh implicitly.

3. Array Datatype is not supported in Terraform.

4. Various variable definition files will be loaded automatically in terraform. These include:

- terraform.tfvars
- terraform.tfvars.json
- Any files with names ending in .auto.tfvars.json

5. Both implicit and explicit dependency information is stored in terraform.tfstate file.

6. terraform init -upgrade updates all previously installed plugins to the newest version.

7. The terraform console command provides an interactive console for evaluating expressions.

8. Difference 0.11 and 0.12

"${var.instance_type}" → 0.11
var.instance_type → 0.12

9 Requirements for publishing module in Terraform Registry

10. Ensure that you know the list of requirements for publishing module in Terraform registry.

https://www.terraform.io/docs/registry/modules/publish.html

11 . List:

list(...): a sequence of values identified by consecutive whole numbers starting with zero. The keyword list is a shorthand for list(any), which accepts any element type as long as every element is the same type

12 . Variable Name Constraints

We cannot use all words within variable names. Terraform reserves some additional names that can no longer be used as input variable names for modules. These reserved names are:

- count
- depends_on
- for_each
- lifecycle
- providers
- source

13. Air Gapped

If terraform needs to be installed in an environment without internet access, the installation is referred to as air-gapped

https://www.terraform.io/docs/enterprise/install/installer.html.

14. Index Function

index finds the element index for a given value in a list.

https://www.terraform.io/docs/configuration/functions/index.html

15. Terraform Enterprise

Before mid-2019, all distributions of Terraform Cloud used to be called Terraform Enterprise; the self-hosted distribution was called Private Terraform Enterprise (PTFE).

Terraform Enterprise supports the following data storage:

PostgresSQL

Any S3-compatible object storage service, GCP Cloud Storage or Azure blob storage meets Terraform Enterprise's object storage requirements.

If you already run your own Vault cluster in production, you can configure Terraform Enterprise to use that instead of running its own internal Vault instance.

https://www.terraform.io/docs/enterprise/before-installing/index.html

16. VCS Provider Support for Terraform Provider

- GitHub.com
- GitHub.com (OAuth)
- GitHub Enterprise
- GitLab.com
- GitLab EE and CE
- Bitbucket Cloud
- Bitbucket Server
- Azure DevOps Server
- Azure DevOps Services

https://www.terraform.io/docs/cloud/vcs/index.html

17. Zipmap function

https://www.terraform.io/docs/configuration/functions/zipmap.html

18 Supported Format for Comments

The Terraform language supports three different syntaxes for comments:

#
//
/* and */

https://www.terraform.io/docs/configuration/syntax.html

18.

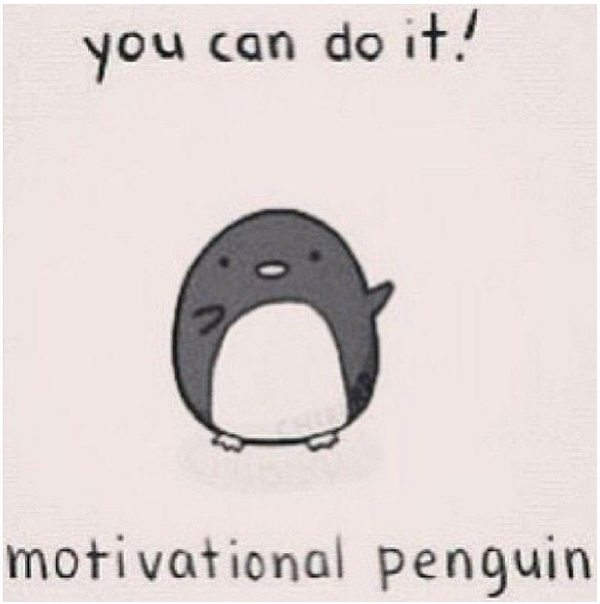
GitHub is not the supported backend type in Terraform.

https://www.terraform.io/docs/backends/types/index.html

When running terraform init, the plugins are downloaded in the sub-directory of the present working directory at the path of .terraform/plugins

API and CLI access for Terraform Cloud can be managed through API tokens that can be generated from Terraform Cloud UI.

Best of Luck for Exams, Rockstar!



Join Our Discord Community

We invite you to join our Discord community, where you can interact with our support team for any course-based technical queries and connect with other students who are doing the same course.

Joining URL:

http://kplabs.in/chat