KPLABS Course

HashiCorp Certified: Terraform Associate

Important Pointers for Exams

ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

Pointer 1: Terraform Providers

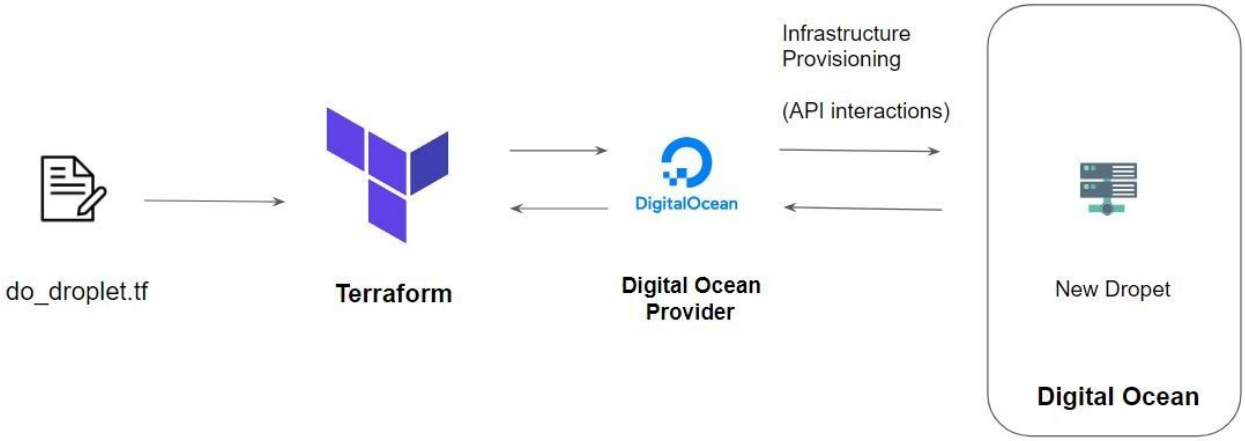
A provider is responsible for understanding API interactions and exposing resources.

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.

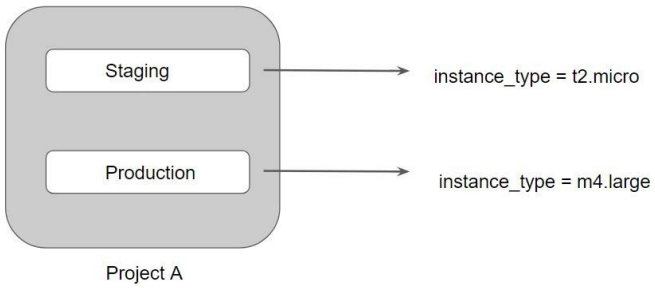
Pointer 14 - Overview of Data Types

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

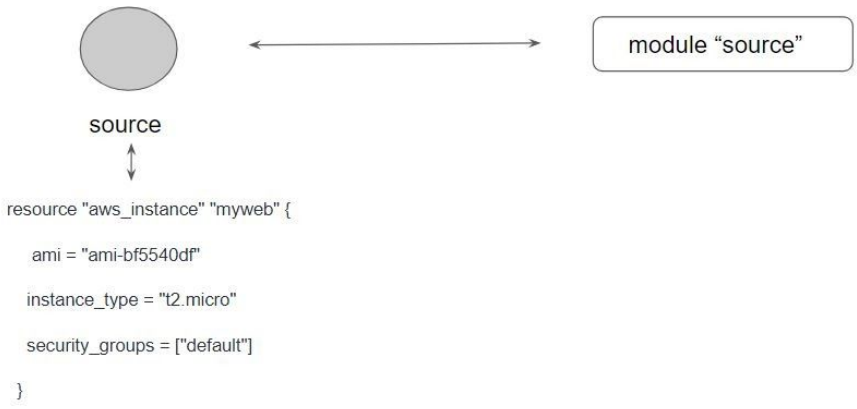
Terraform allows us to have multiple workspaces; with each of the workspaces, we can have a different set of environment variables associated.

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.

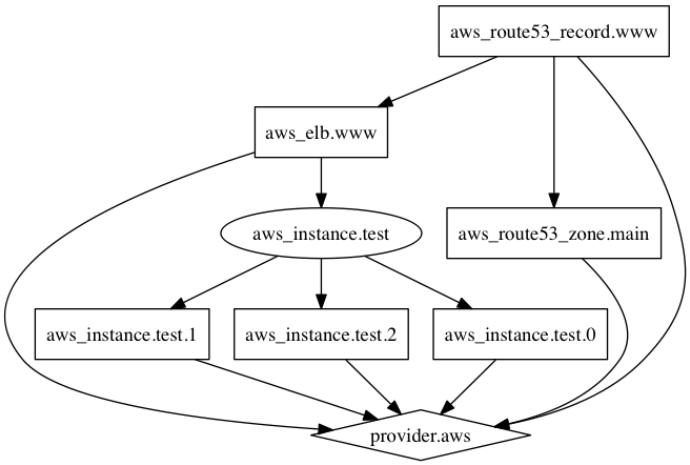
It works well in Windows, Linux, MAC.

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.

Best of Luck for Exams, Rockstar!