KPLABS Course

HashiCorp Certified: Terraform Associate

Base Domain

ISSUED BY

Zeal

REPRESENTATIVE

instructors@kplabs.in

Base Domain - Getting Started & Setting Up Labs

Module 1: Choosing a right Infrastructure as Code tool

1.1 Exploring Toolsets

There are various types of tools that can allow you to deploy infrastructure as code :

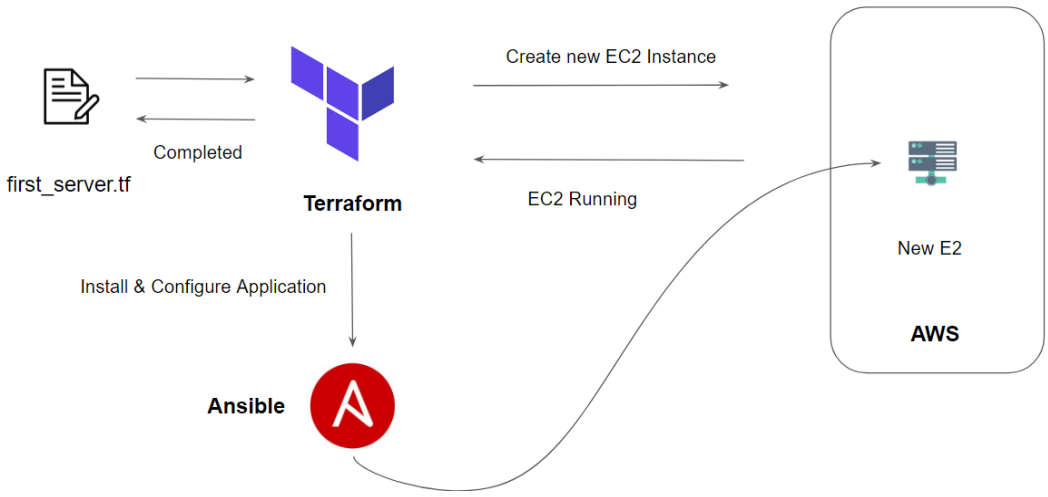
- Terraform
- CloudFormation
- Heat
- Ansible
- SaltStack
- Chef, Puppet and others

1.2 Configuration Management vs Infrastructure Orchestration

Ansible, Chef, Puppet are configuration management tools which means that they are primarily designed to install and manage software on existing servers.

Terraform, CloudFormation are the infrastructure orchestration tools which basically means they can provision the servers and infrastructure by themselves.

Configuration Management tools can do some degree of infrastructure provisioning, but the focus here is that some tools are going to be better fit for certain types of tasks.



1.3 Which tool to choose?

Question remains on how to choose right IAC tool for the organization

i) Is your infrastructure going to be vendor specific in longer term ? Example AWS.

ii) Are you planning to have multi-cloud / hybrid cloud based infrastructure ?

iii) How well does it integrate with configuration management tools ?

iv) Price and Support

1.4 Terraform

i) Supports multiple platforms, has hundreds of providers.

ii) Simple configuration language and faster learning curve.

iii) Easy integration with configuration management tools like Ansible.

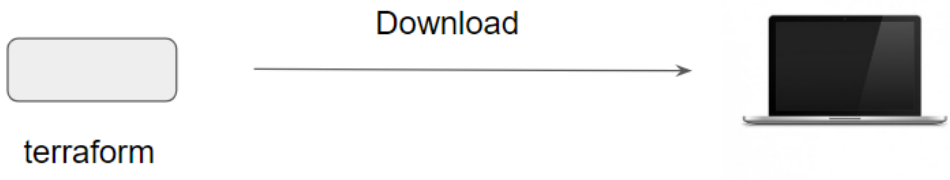
iv) Easily extensible with the help of plugins.

v) Free !!!

Module 2: Installation Process of Terraform

Terraform installation is very simple.

You have a single binary file, download and use it.



Terraform works on multiple platforms, these includes:

- Windows
- macOS
- Linux
- FreeBSD
- OpenBSD
- Solaris

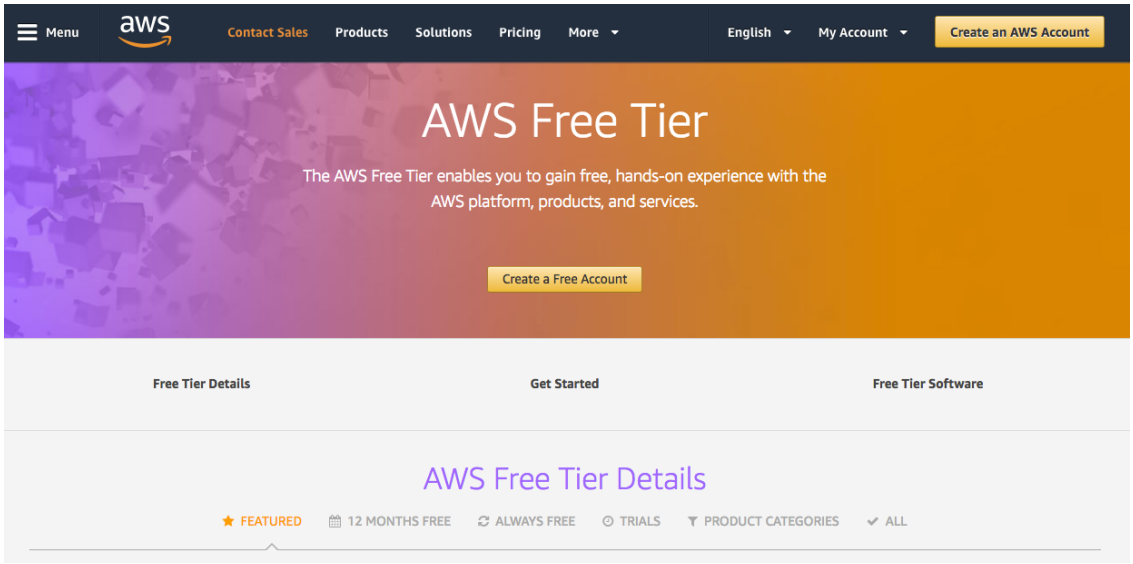
You can download Terraform binary through the following link:

https://www.terraform.io/downloads

Module 3: Setting up the Lab

i) Create a new AWS Account.

ii) Begin the course

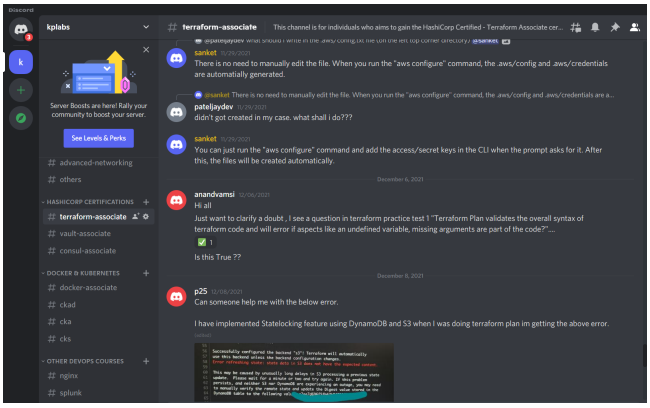


Join Our Discord Community

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



KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 1

ISSUED BY
Zeal

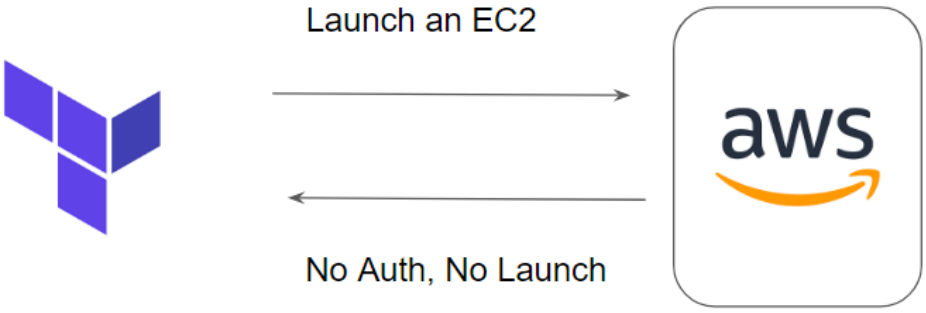
REPRESENTATIVE
instructors@kplabs.in

Domain 1 - Deploying Infrastructure with Terraform

Module 1: Creating first EC2 instance with Terraform

When you want to launch a resource in AWS, there are three important considerations.

- How will you authenticate to AWS?
- Which region the resource needs to be launched.
- Which resource you want to launch.



Each Provider has resources divided into multiple categories.

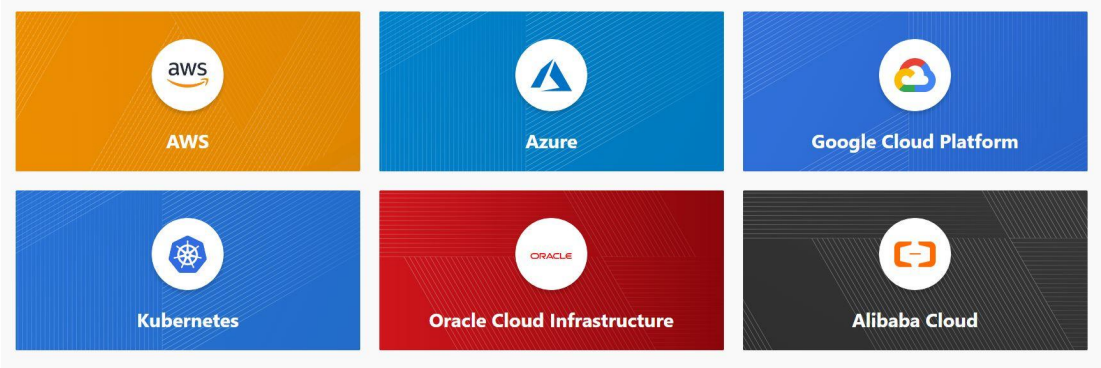
Resources require a certain set of arguments based on the use-case.

Module 2: Provider and Resources

2.1 Overview of Providers:

Terraform supports multiple providers.

Depending on what type of infrastructure we want to launch, we have to use appropriate providers accordingly.



2.2 Initialization Phase

Upon adding a provider, it is important to run terraform init which in-turn will download plugins associated with the provider.

```
C:\Users\Zeal Vora\Desktop\kplabs-terraform>terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Finding latest version of hashicorp/azurerm...
- Installing hashicorp/aws v3.26.0...
- Installed hashicorp/aws v3.26.0 (signed by HashiCorp)
- Installing hashicorp/azurerm v2.45.1...
- Installed hashicorp/azurerm v2.45.1 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!
```

2.3 Resources

Resources are the reference to the individual services which the provider has to offer

Example:

- resource aws_instance
- resource aws_alb
- resource iam_user
- resource digitalocean_droplet

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

2.4 Important Update - Newer Version

From 0.13 onwards, Terraform requires explicit source information for any providers that are not HashiCorp-maintained, using a new syntax in the required_providers nested block inside the Terraform configuration block

```hcl
provider "aws" {
  region     = "us-west-2"
  access_key = "PUT-YOUR-ACCESS-KEY-HERE"
  secret_key = "PUT-YOUR-SECRET-KEY-HERE"
}
```

HashiCorp Maintained

```hcl
terraform {
  required_providers {
    digitalocean = {
      source = "digitalocean/digitalocean"
    }
  }
}

provider "digitalocean" {
  token = "PUT-YOUR-TOKEN-HERE"
}
```

Non-HashiCorp Maintained

Terraform 0.13 onwards

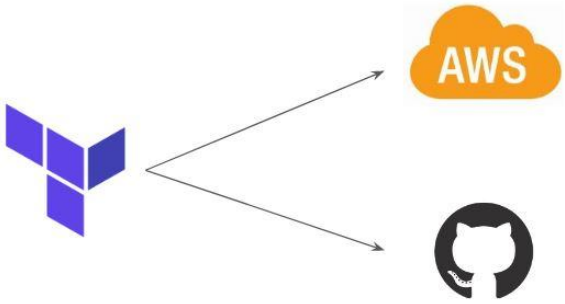
```hcl
provider "aws" {
  region     = "us-west-2"
  access_key = "PUT-YOUR-ACCESS-KEY-HERE"
  secret_key = "PUT-YOUR-SECRET-KEY-HERE"
}
```

```hcl
terraform {
  required_providers {
    digitalocean = {
      source = "digitalocean/digitalocean"
    }
  }
}

provider "digitalocean" {
  token = "PUT-YOUR-TOKEN-HERE"
}
```

.

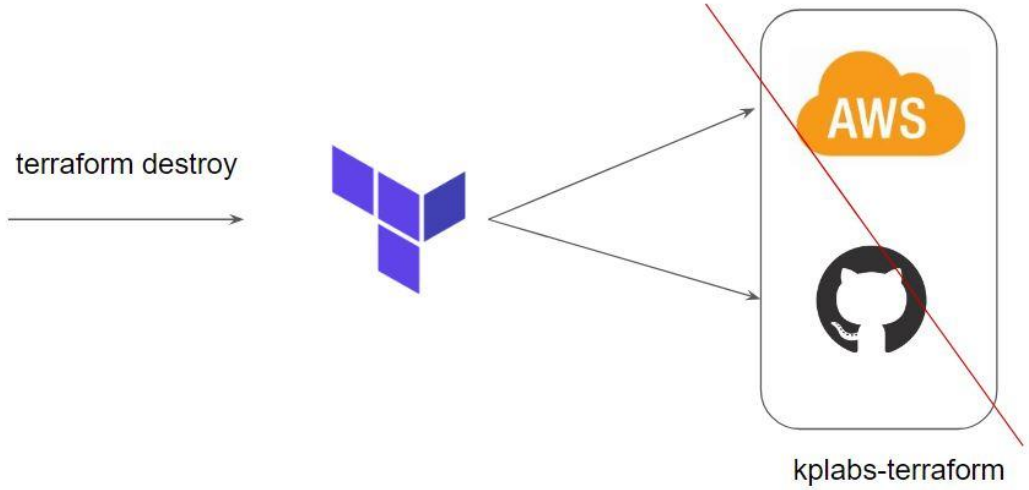
Module 3: Destroying Infrastructure with Terraform (NEW)

If you keep the infrastructure running, you will get charged for it. Hence it is important for us to also know how we can delete the infrastructure resources created via terraform.



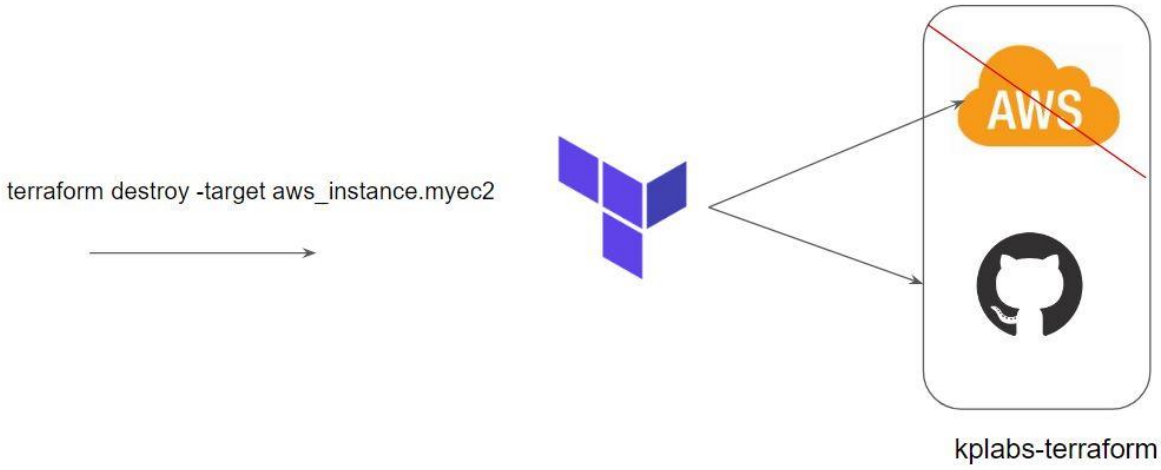
3.1 Approach 1

terraform destroy allows us to destroy all the resources that are created within the folder.



3.2 Approach 2

terraform destroy with -target flag allows us to destroy the specific resource.



3.3 Terraform Destroy with Target

The -target option can be used to focus Terraform's attention on only a subset of resources.

Combination of: Resource Type + Local Resource Name

Resource Type	Local Resource Name
aws_instance	myec2
github_repository	example

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

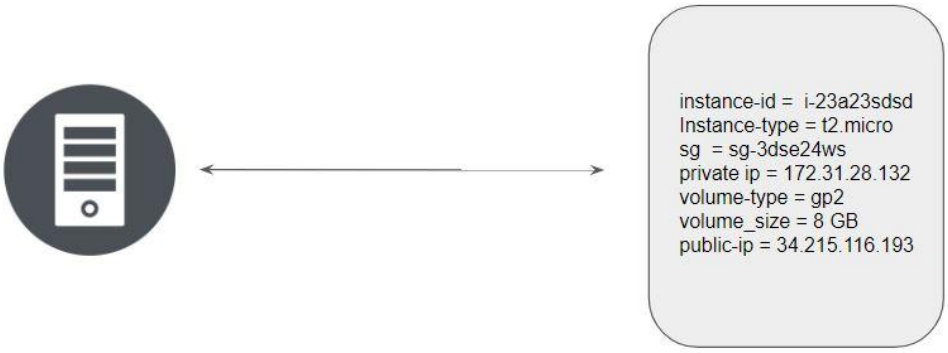
```
resource "github_repository" "example" {
  name        = "terraform-repo"

  visibility  = "private"

}
```

Module 4: Terraform State File

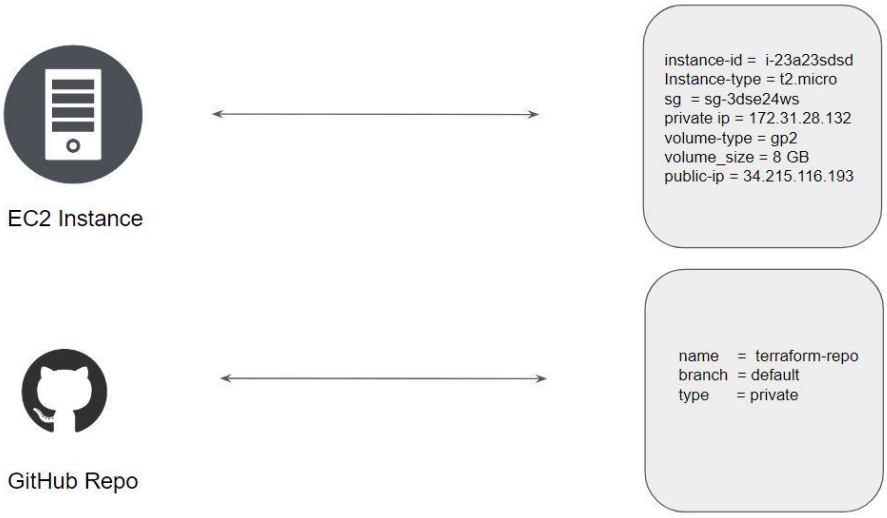
4.1 Overview of State Files:

Terraform stores the state of the infrastructure that is being created from the TF files.

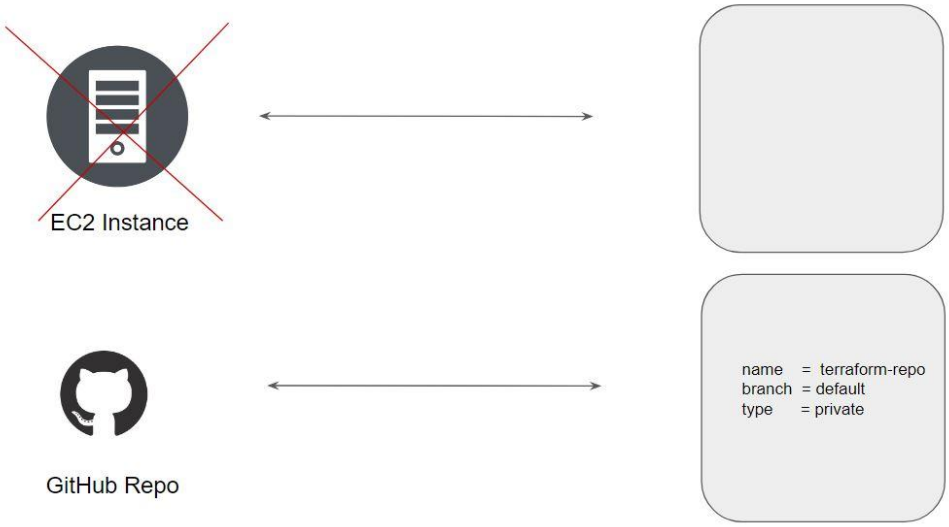
This state allows terraform to map real-world resources to your existing configuration.



Multiple resources in Terraform will have a separate block with the state file.



Whenever a resource is removed, its corresponding entry under the state file is also removed.



Module 5: Desired & Current State

5.1 Desired State

Terraform's primary function is to create, modify, and destroy infrastructure resources to match the desired state described in a Terraform configuration

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

5.2 Current State

The current state is the actual state of a resource that is currently deployed.

```
resource "aws_instance" "myec2" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
}
```

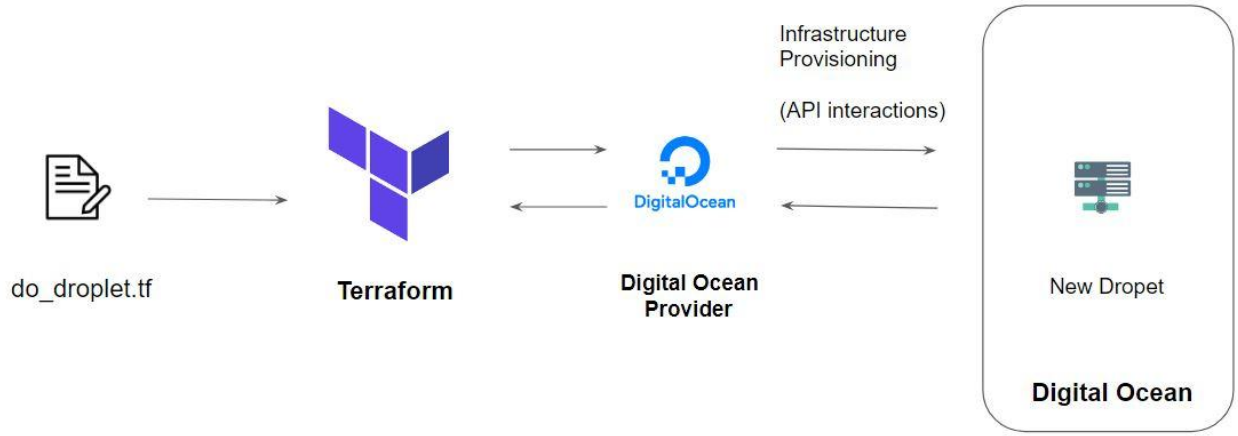
5.3 Important Pointer

Terraform tries to ensure that the deployed infrastructure is based on the desired state.

If there is a difference between the two, terraform plan presents a description of the changes necessary to achieve the desired state.

Module 6: Provider Versioning

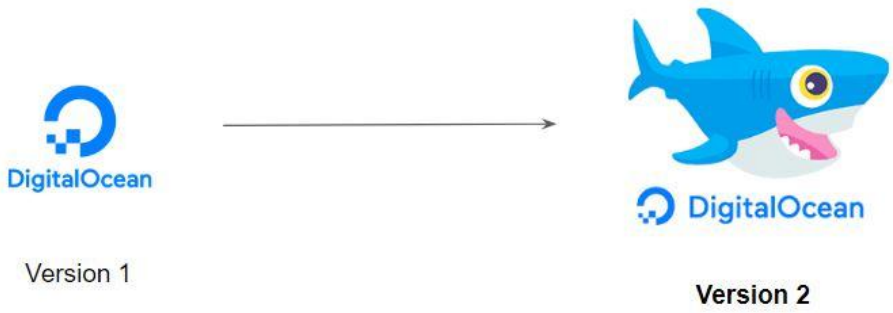
6.1 Overview of Provider Architecture



6.2 Provider Versioning

Provider plugins are released separately from Terraform itself.

They have a different set of version numbers.



6.3 Explicitly Setting Provider Version

During terraform init, if the version argument is not specified, the most recent provider will be downloaded during initialization.

For production use, you should constrain the acceptable provider versions via configuration, to ensure that new versions with breaking changes will not be automatically installed

```
provider "aws" {
  region     = "us-west-2"
  version    = "2.7"
}
```

6.4 Arguments for Specifying the provider

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

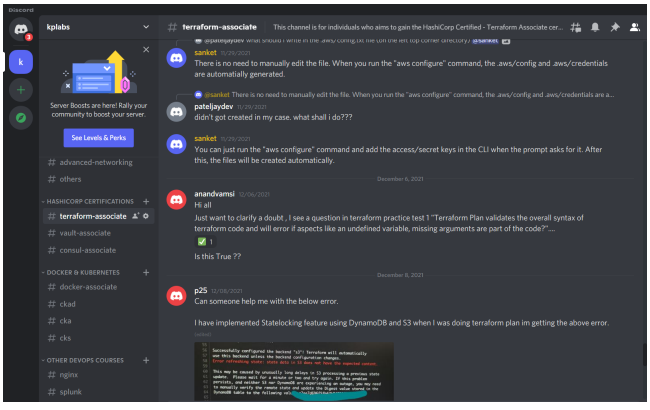
Version Number Arguments	Description
>=1.0	Greater than equal to the version
<=1.0	Less than equal to the version
~>2.0	Any version in the 2.X range.
>=2.10,<=2.30	Any version between 2.10 and 2.30

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



KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 2

ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

Domain 2 - Read, Generate, Modify Configurations

Module 1: Attributes and Output Values

Terraform has the capability to output the attribute of a resource with the output values.

Example:

- ec2_public_ip = 35.161.21.197
- bucket_identifier = terraform-test-kplabs.s3.amazonaws.com

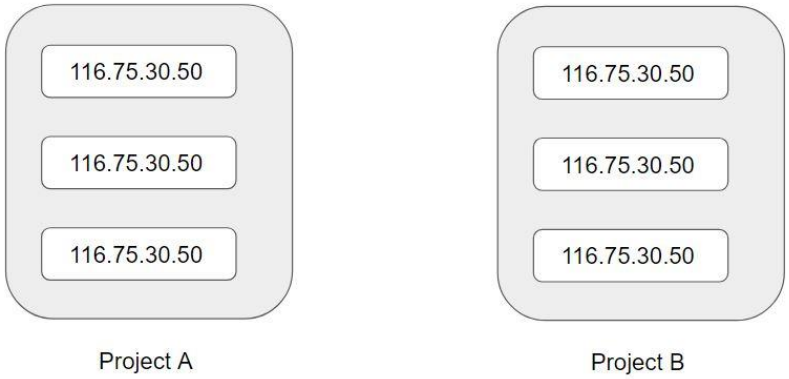
An outputed attributes can not only be used for the user reference but it can also act as an input to other resources being created via terraform

Let's understand this with an example:

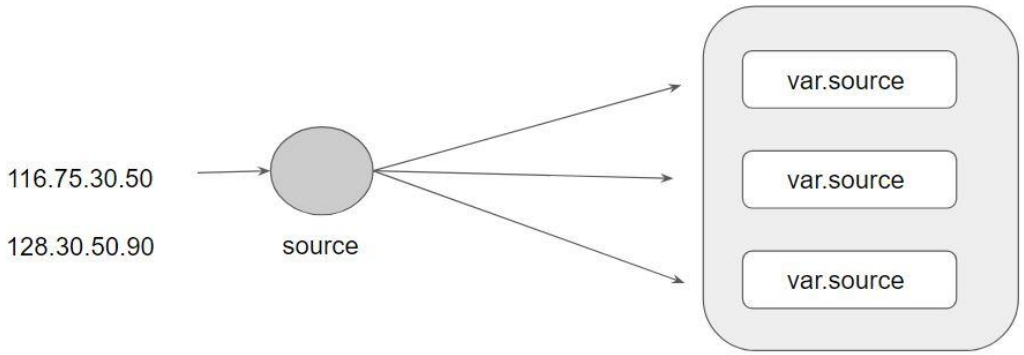
After EIP gets created, it's IP address should automatically get whitelisted in the security group.

Module 2: Terraform variables

Repeated static values can create more work in the future.



Terraform Variables allows us to centrally define the values that can be used in multiple terraform configuration blocks.



Module 3: Approach for Variable Assignment

Variables in Terraform can be assigned values in multiple ways.

Some of these include:

- Environment variables
- Command Line Flags
- From a File
- Variable Defaults

Sample Commands for the following:

i) Environment Variables:

```
export TF_VAR_instancetype="t2.nano"
echo $TF_VAR
```

ii) Command Line Flags:

```
terraform plan -var="instancetype=t2.small"
terraform plan -var-file="custom.tfvars"
```

iii) From a File (terraform.tfvars):

```
instancetype="t2.large"
```

iv) Variable Defaults:

```
variable "instancetype" {
  default = "t2.micro"
}
```

Module 4: Data Types for Variables

4.1 Overview of Type Constraints

The type argument in a variable block allows you to restrict the type of value that will be accepted as the value for a variable

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

If no type constraint is set then a value of any type is accepted.

4.2 Example Use-Case 1

Every employee in Medium Corp is assigned an Identification Number.

Any resource that an employee creates should be created with the name of the identification number only.

variables.tf	terraform.tfvars
variable “instance_name” {}	instance_name=”john-123”

4.3 Example Use-Case 2

Every employee in Medium Corp is assigned an Identification Number.

Any EC2 instance that employee creates should be created using the identification number only.

variables.tf	terraform.tfvars

variable “instance_name” {
type=number
}
	instance_name=”john-123”

4.4 Overview of Data Types

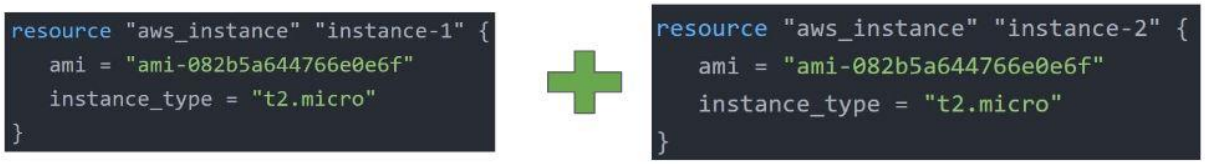
Type Keywords	Description
string	Sequence of Unicode characters representing some text, like "hello".
list	Sequential list of values identified by their position. Starts with 0 [“mumbai” ,”singapore”, ”usa”]
map	a group of values identified by named labels, like {name = "Mabel", age = 52}.
number	Example: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

Module 5: Count Parameter

5.1 Overview of Count:

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

Let’s assume, you need to create two EC2 instances. One of the common approaches is to define two separate resource blocks for aws_instance.



With the count parameter, we can simply specify the count value and the resource can be scaled accordingly.

```
resource "aws_instance" "instance-1" {
   ami = "ami-082b5a644766e0e6f"
   instance_type = "t2.micro"
   count = 5
}
```

5.2 Count Index

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

This object has one attribute:

count.index — The distinct index number (starting with 0) corresponding to this instance.

5.3 Challenges with Count Parameter

With the below code, terraform will create 5 IAM users. But the problem is that all will have the same name.

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

count.index allows us to fetch the index of each iteration in the loop.

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

Understanding Challenge with Default Count Index

Having a username like loadbalancer0, loadbalancer1 might not always be suitable.

Better names like dev-loadbalancer, stage-loadbalancer, prod-loadbalancer is better.

count.index can help in such a scenario as well.

```
variable "elb_names" {
  type        = list
  default     = ["dev-loadbalancer", "stage-loadbalancer", "prod-loadbalancer"]
}
```

Module 6: Conditional Expression

A conditional expression uses the value of a bool expression to select one of two values.

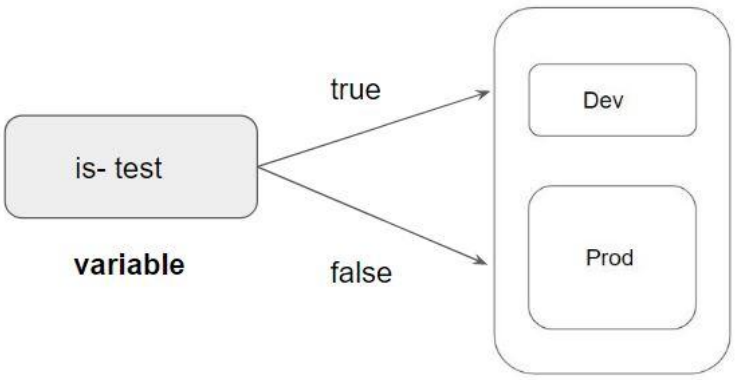
Syntax of Conditional expression:

condition ? true_val : false_val

If the condition is true then the result is true_val. If the condition is false then the result is false_val.

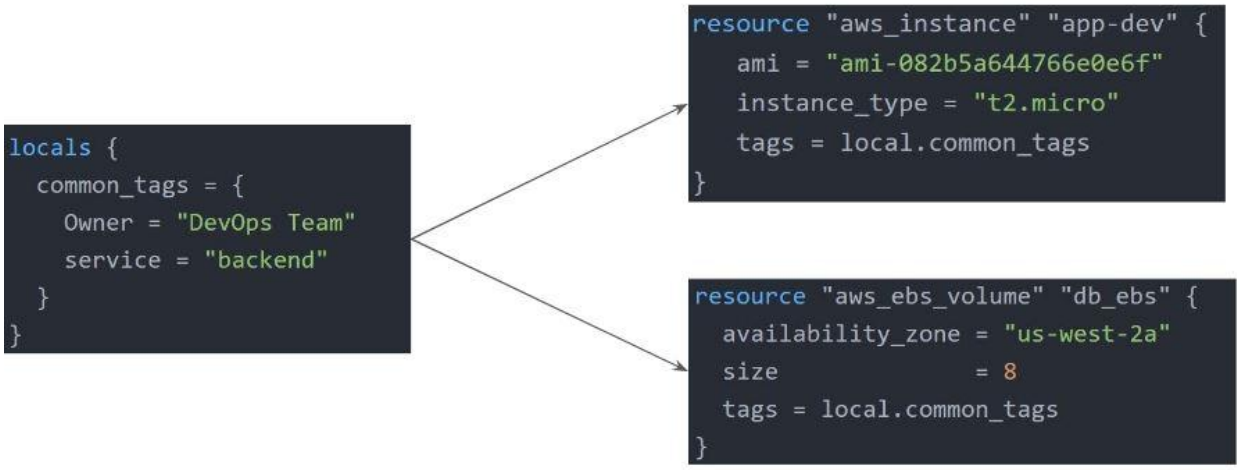
Let's assume that there are two resource blocks as part of terraform configuration.

Depending on the variable value, one of the resource blocks will run.



Module 7: Local Values

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



Local Values Support for Expression

Local Values can be used for multiple different use-cases like having a conditional expression.

```
locals {
  name_prefix = "${var.name != "" ? var.name : var.default}"
}
```

Important Pointers for Local Values:

Local values can be helpful to avoid repeating the same values or expressions multiple times in a configuration.

If overused they can also make a configuration hard to read by future maintainers by hiding the actual values used

Use local values only in moderation, in situations where a single value or result is used in many places and that value is likely to be changed in the future.

Module 8: Terraform Functions

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

The general syntax for function calls is a function name followed by comma-separated arguments in parentheses:

function (argument1, argument2)

Example:

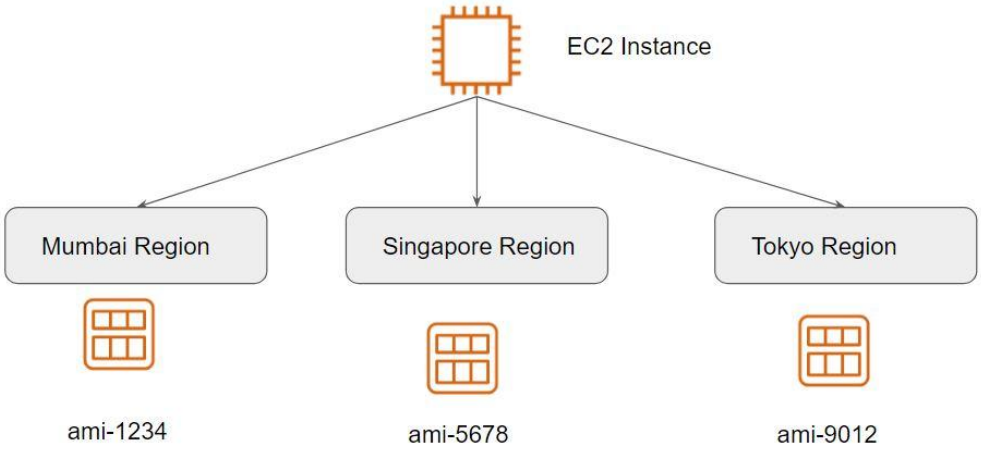
> max(5, 12, 9)
12

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

- Numeric
- String
- Collection
- Encoding
- Filesystem
- Date and Time
- Hash and Crypto
- IP Network
- Type Conversion

Module 9: Data Sources

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



A data source is defined under the data block.

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

```
data "aws_ami" "app_ami" {
  most_recent = true
  owners = ["amazon"]


  filter {
    name   = "name"
    values = ["amzn2-ami-hvm*"]
  }
}
```

```
resource "aws_instance" "instance-1" {
    ami = data.aws_ami.app_ami.id
   instance_type = "t2.micro"
}
```

Module 10: 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

```
bash-4.2# terraform plan
2020/04/22 13:45:31 [INFO] Terraform version: 0.12.24
2020/04/22 13:45:31 [INFO] Go runtime version: go1.12.13
2020/04/22 13:45:31 [INFO] CLI args: []string{"/usr/bin/terraform", "plan"}
2020/04/22 13:45:31 [DEBUG] Attempting to open CLI config file: /root/.terraformrc
2020/04/22 13:45:31 [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2020/04/22 13:45:31 [DEBUG] checking for credentials in "/root/.terraform.d/plugins"
2020/04/22 13:45:31 [INFO] CLI command args: []string{"plan"}
2020/04/22 13:45:31 [TRACE] Meta.Backend: built configuration for "s3" backend with hash value 789489680
2020/04/22 13:45:31 [TRACE] Meta.Backend: backend has not previously been initialized in this working directory
2020/04/22 13:45:31 [DEBUG] New state was assigned lineage "a10f92bf-686d-e6cf-3e9d-755be5c8a6a3"
2020/04/22 13:45:31 [TRACE] Meta.Backend: moving from default local state only to "s3" backend
```

Important Pointers for Debugging:

TRACE is the most verbose and it is the default if TF_LOG is set to something other than a log level name.

To persist logged output you can set TF_LOG_PATH in order to force the log to always be appended to a specific file when logging is enabled.

Module 11: Terraform Format

Anyone who is into programming knows the importance of formatting the code for readability.

```
provider "aws" {
      region     = "us-west-2"
  access_key = "AKIAQIW66DN2W7WOYRGY"
  secret_key    = "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"
  version    = ">=2.10,<=2.30"
}
```

The terraform fmt command is used to rewrite Terraform configuration files to take care of the overall formatting

Before fmt

```
provider "aws" {
      region     = "us-west-2"
  access_key = "AKIAQIW66DN2W7WOYRGY"
  secret_key    = "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"
  version    = ">=2.10,<=2.30"
}
```

After fmt

```
provider "aws" {
  region     = "us-west-2"
  access_key = "AKIAQIW66DN2W7WOYRGY"
  secret_key = "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"
  version    = ">=2.10,<=2.30"
}
```

Module 12: Terraform Validate

Terraform Validate primarily checks whether a configuration is syntactically valid.

It can check various aspects including unsupported arguments, undeclared variables, and others.

```
resource "aws_instance" "myec2" {
  ami           = "ami-082b5a644766e0e6f"
  instance_type = "t2.micro"
  sky = "blue"
}
```

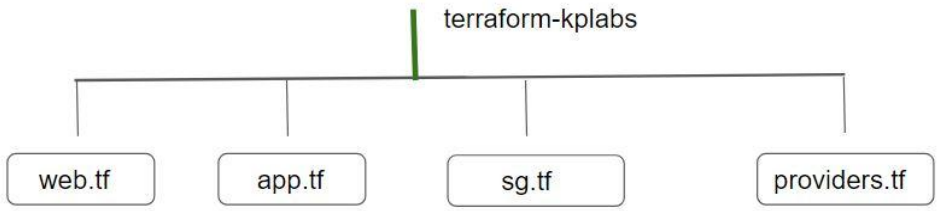
→

```
bash-4.2# terraform validate

Error: Unsupported argument

  on validate.tf line 10, in resource "aws_instance" "myec2":
  10:   sky = "blue"

An argument named "sky" is not expected here.
```

Module 13: Load Order & Semantics

Terraform generally loads all the configuration files within the directory specified in alphabetical order.

The files loaded must end in either .tf or .tf.json to specify the format that is in use.



Module 14: Dynamic Blocks

14.1 Understanding the Challenge:

In many of the use-cases, there are repeatable nested blocks that need to be defined.

This can lead to a long code and it can be difficult to manage in a long time.

```
ingress {
  from_port   = 9200
  to_port     = 9200
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

```
ingress {
  from_port   = 8300
  to_port     = 8300
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

14.2 Overview of Dynamic Blocks

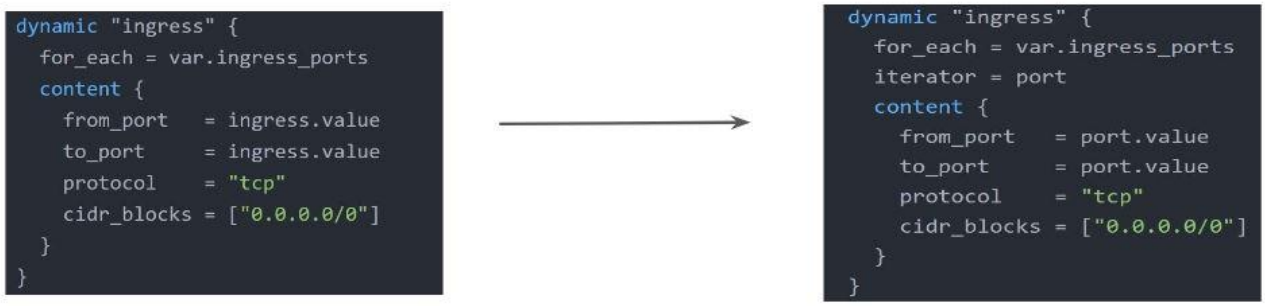
Dynamic Block allows us to dynamically construct repeatable nested blocks which is supported inside resource, data, provider, and provisioner blocks:

```
dynamic "ingress" {
  for_each = var.ingress_ports
  content {
    from_port   = ingress.value
    to_port     = ingress.value
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

14.3 Overview of Iterators

The iterator argument (optional) sets the name of a temporary variable that represents the current element of the complex value

If omitted, the name of the variable defaults to the label of the dynamic block ("ingress" in the example above).



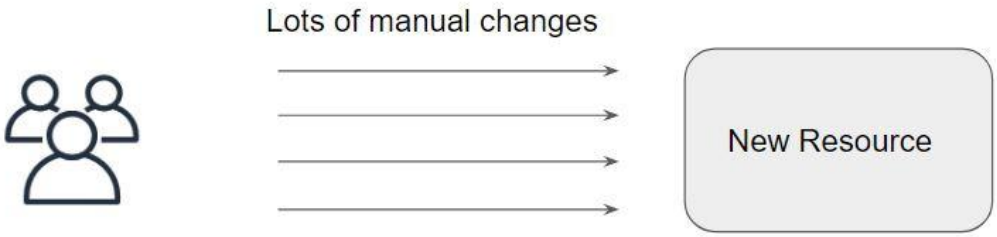
Module 15: Terraform Taint

15.1 Understanding the Challenge:

You have created a new resource via Terraform.

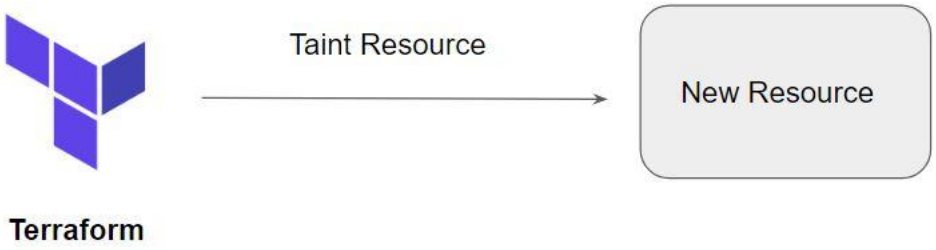
Users have made a lot of manual changes (both infrastructure and inside the server)

Two ways to deal with this: Import The Changes to Terraform / Delete & Recreate the resource



15.2 Overview of 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.



15.3 Important Pointers for Terraform Taint

This command will not modify infrastructure but does modify the state file in order to mark a resource as tainted.

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.

Note that tainting a resource for recreation may affect resources that depend on the newly tainted resource.

Module 16: Splat Expression

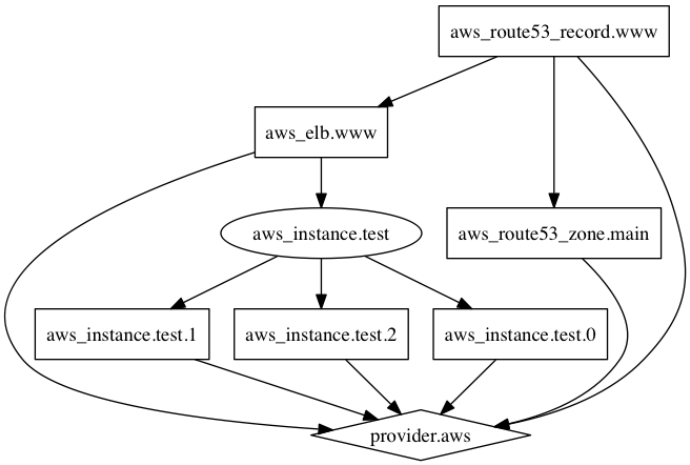
Splat Expression allows us to get a list of all the attributes.

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

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

Module 17: 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.



Module 18: Saving Terraform Plan to a File

The generated terraform plan can be saved to a specific path.

This plan can then be used with terraform apply to be certain that only the changes shown in this plan are applied.

Example:

terraform plan -out=path

Module 19: 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",
]
```

Module 20: Terraform Settings

The special terraform configuration block type is used to configure some behaviors of Terraform itself, such as requiring a minimum Terraform version to apply your configuration.

Terraform settings are gathered together into terraform blocks:

```
terraform {
  # ...
}
```

20.1 Setting 1 - Terraform 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"
}
```

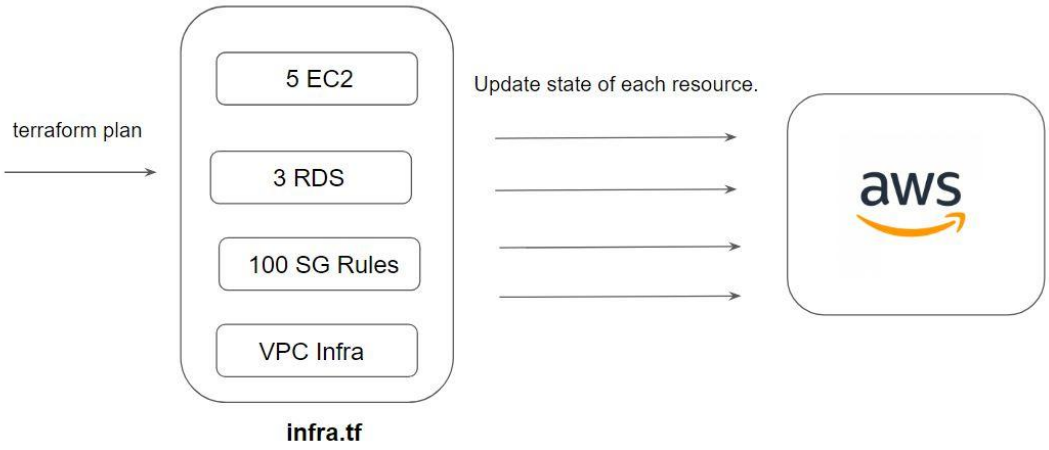
20.1 Setting 2 - Provider Version

The required_providers block specifies all of the providers required by the current module, mapping each local provider name to a source address and a version constraint.

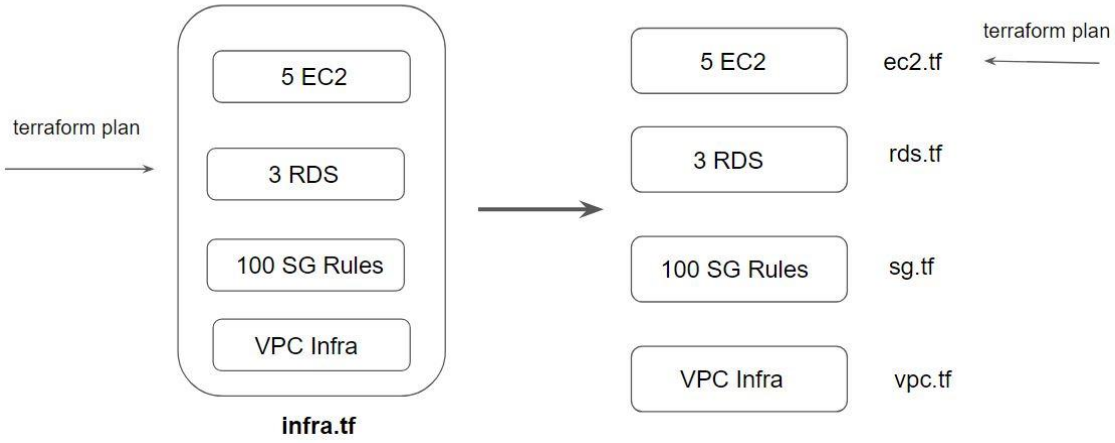
```
terraform {
  required_providers {
    mycloud = {
      source  = "mycorp/mycloud"
      version = "~> 1.0"
    }
  }
}
```

Module 21: Dealing with Large Infrastructure

When you have a larger infrastructure, you will face issues related to API limits for a provider.



It is important to switch to a smaller configurations were each can be applied independently.



21.1 Setting Refresh to False

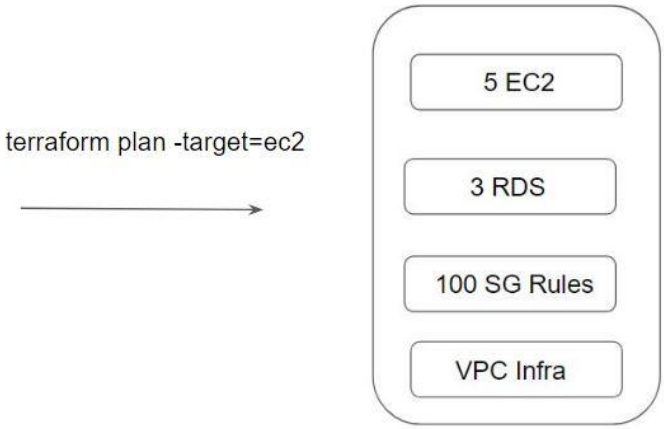
We can prevent terraform from querying the current state during operations like terraform plan.

This can be achieved with the -refresh=false flag

21.2 Specify the Target

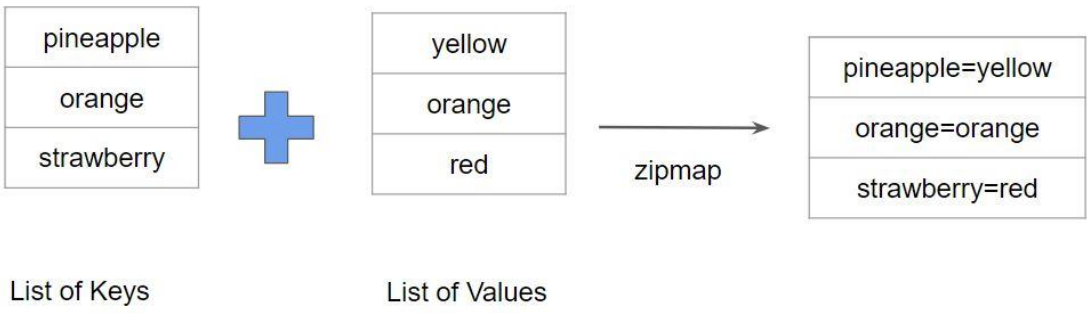
The -target=resource flag can be used to target a specific resource.

Generally used as a means to operate on isolated portions of very large configurations



Module 22: Zipmap Function

The zipmap function constructs a map from a list of keys and a corresponding list of values.



Following screenshot shows a sample output of Zipmap

```
←[J> zipmap(["pineapple","oranges","strawberry"], ["yellow","orange","red"])
{
  "oranges" = "orange"
  "pineapple" = "yellow"
  "strawberry" = "red"
}
```

Let us understand Zipmap with a sample use-case

You are creating multiple IAM users.

You need output which contains direct mapping of IAM names and ARNs

```
zipmap = {
  "demo-user.0" = "arn:aws:iam::018721151861:user/system/demo-user.0"
  "demo-user.1" = "arn:aws:iam::018721151861:user/system/demo-user.1"
  "demo-user.2" = "arn:aws:iam::018721151861:user/system/demo-user.2"
}
```

Module 23: Comments in Terraform Code

A comment is a text note added to source code to provide explanatory information, usually about the function of the code

```
'''In this program, we check if the number is positive or
negative or zero and
display an appropriate message'''

num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

The Terraform language supports three different syntaxes for comments:

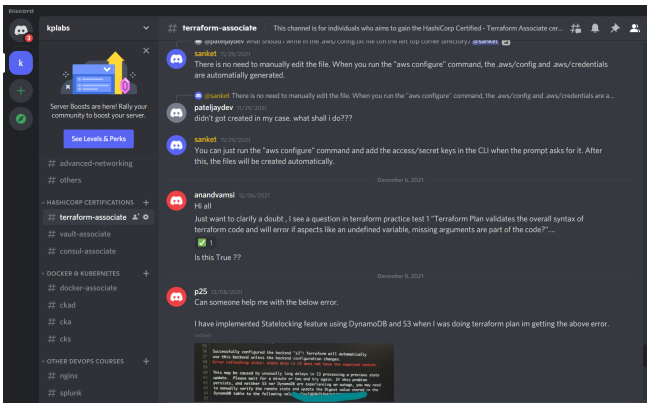
Type	Description
#	begins a single-line comment, ending at the end of the line.
//	also begins a single-line comment, as an alternative to #.
/* and */	are start and end delimiters for a comment that might span over multiple lines.

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



KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 3 - Terraform Provisioners

ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

Module 1: Understanding Provisioners in Terraform

1.1 Understanding the Challenge

Till now we have been working only on the creation and destruction of infrastructure scenarios.

Let's take an example:

We created a web-server EC2 instance with Terraform.

Problem: It is only an EC2 instance, it does not have any software installed.

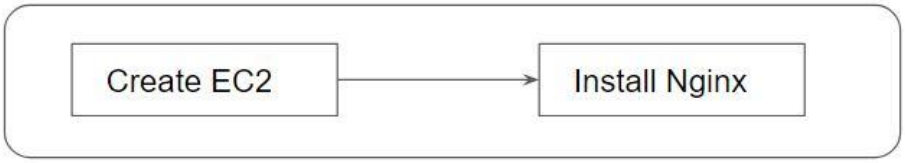
What if we want a complete end to end solution?

1.2 Introducing Terraform Provisioners

Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction.

Let's take an example:

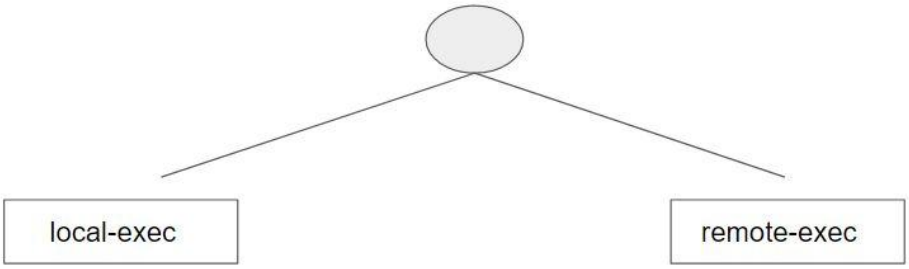
On creation of Web-Server, execute a script which installs Nginx web-server.



Module 2: Types of Provisioners

Terraform has the capability to turn provisioners both at the time of resource creation as well as destruction.

There are two main types of provisioners:



2.1 Local Exec Provisioners

local-exec provisioners allow us to invoke a local executable after the resource is created.

One of the most used approaches of local-exec is to run ansible-playbooks on the created server after the resource is created.

Let's take an example:

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

2.2 Remote Exec Provisioners

Remote-exec provisioners allow invoking scripts directly on the remote server.

Let's take an example:

```
resource "aws_instance" "web" {
  # …

  provisioner "remote-exec" {
                        ……...
  }
}
```

Module 3: 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.

If when = destroy is specified, the provisioner will run when the resource it is defined within is destroyed.

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

  provisioner "local-exec" {
    when    = destroy
    command = "echo 'Destroy-time provisioner'"
  }
}
```

Module 4: Failure Behavior

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:

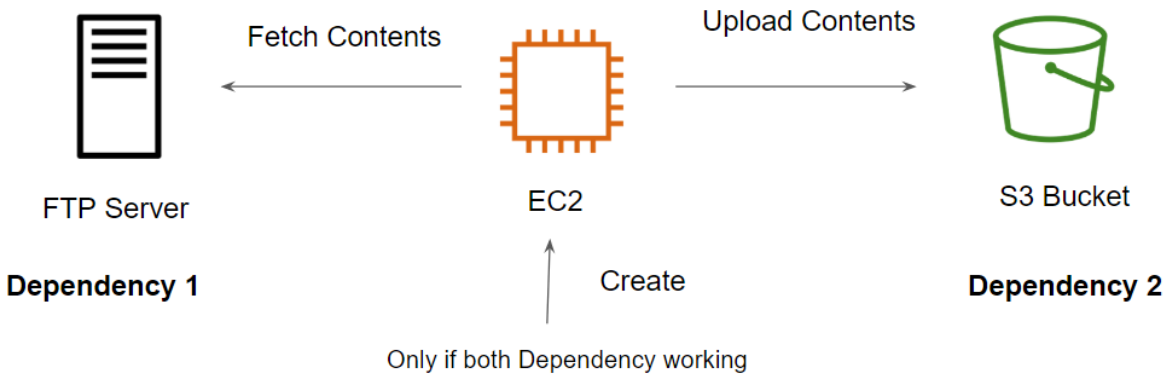
Allowed Values	Description
continue	Ignore the error and continue with creation or destruction.
fail	Raise an error and stop applying (the default behavior). If this is a creation provisioner, taint the resource.

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

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

.

Module 5: Null Resource

The null_resource implements the standard resource lifecycle but takes no further action.

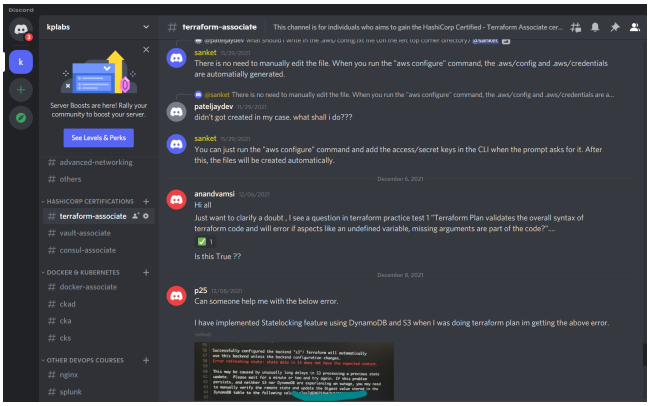


Join Our Discord Community

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


Discord
kplabs
terraform-associate
This channel is for individuals who aims to gain the HashiCorp Certified - Terraform Associate cer...
Server Boosts are here! Rally your community to boost your server.
See Levels & Perks
advanced-networking
others
HASHICORP CERTIFICATIONS
terraform-associate
vault-associate
consul-associate
DOCKER & KUBERNETES
docker-associate
ckad
cka
cks
OTHER DEVOPS COURSES
nginx
splunk
sanket 11/29/2021
There is no need to manually edit the file. When you run the "aws configure" command, the .aws/config and .aws/credentials are automatically generated.
@sanket There is no need to manually edit the file. When you run the "aws configure" command, the .aws/config and .aws/credentials are a...
pateljaydev 11/29/2021
didn't got created in my case. what shall i do???
sanket 11/29/2021
You can just run the "aws configure" command and add the access/secret keys in the CLI when the prompt asks for it. After this, the files will be created automatically.
December 6, 2021
anandvamsi 12/06/2021
Hi all
Just want to clarify a doubt , I see a question in terraform practice test 1 "Terraform Plan validates the overall syntax of terraform code and will error if aspects like an undefined variable, missing arguments are part of the code?' ....
1
Is this True ??
December 6, 2021
p25 12/08/2021
Can someone help me with the below error.
I have implemented Statelocking feature using DynamoDB and S3 when I was doing terraform plan im getting the above error.

KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 4

ISSUED BY
Zeal

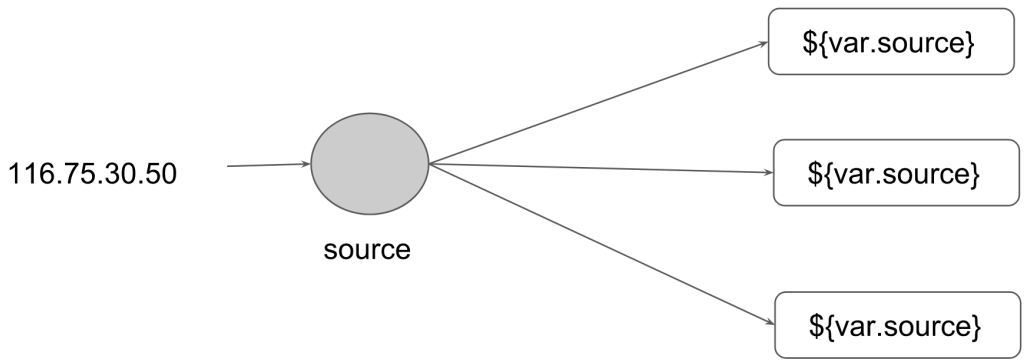
REPRESENTATIVE
instructors@kplabs.in

Domain 4 - Terraform Modules & Workspaces

Module 1: Understanding the DRY principle

In software engineering, don't repeat yourself (DRY) is a principle of software development aimed at reducing repetition of software patterns.

In the earlier lecture, we were making static content into variables so that there can be a single source of information.



1.1 Generic Scenario:

We do repeat multiple times various terraform resources for multiple projects.

Sample EC2 Resource

```
resource "aws_instance" "myweb" {
    ami = "ami-bf5540df"
   instance_type = "t2.micro"
   security_groups = ["default"]
 }
```

Instead of repeating a resource block multiple times, we can make use of a centralized structure.

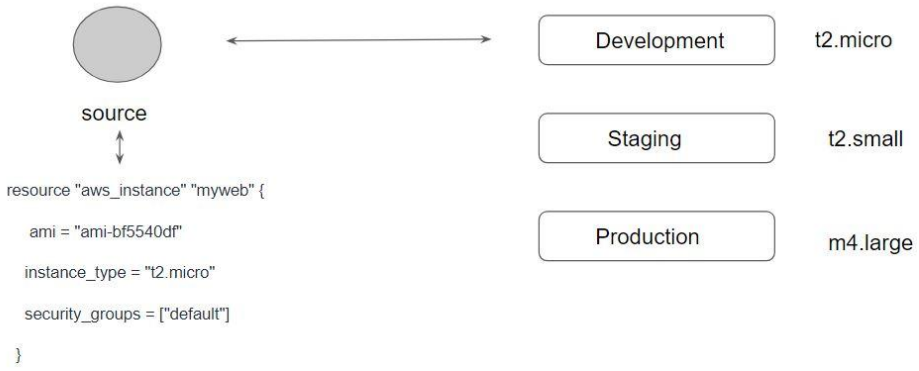
1.2 Centralized Structure:

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

Module 2: Challenges with Terraform Modules

One common need for infrastructure management is to build environments like staging, production with a similar setup but keeping environment variables different.

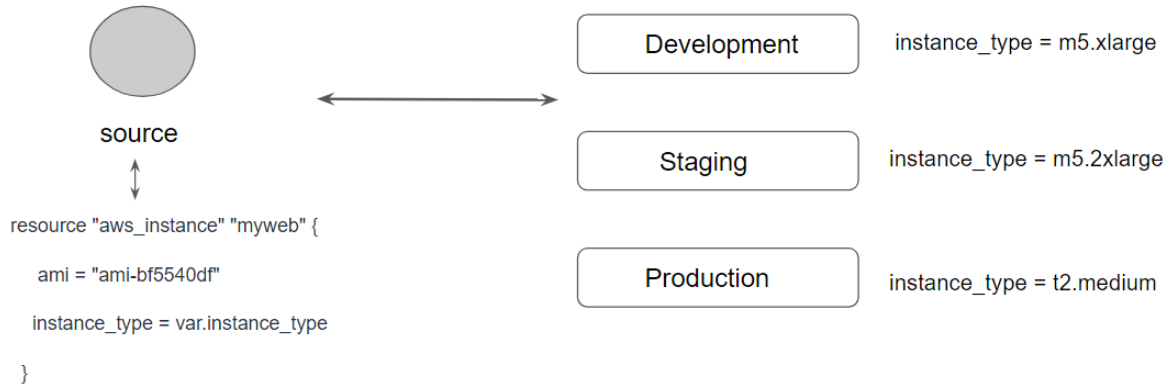
When we use modules directly, the resources will be a replica of code in the module.



Module 3: Using Locals with Modules

3.1 Understanding the Challenge

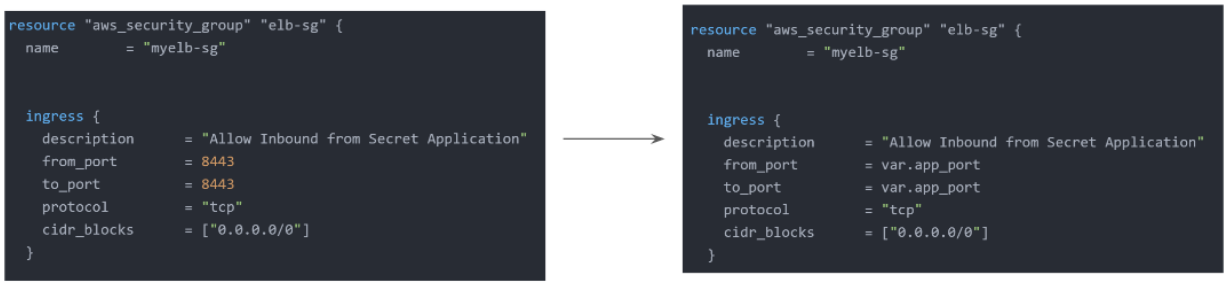
Using variables in Modules can also allow users to override the values which you might not want.



3.2 Setting the Context

There can be many repetitive values in modules and this can make your code difficult to maintain.

You can centralize these using variables but users will be able to override it.



Hardcoded Port

Variable Port

3.3 Using Locals

Instead of variables, you can make use of locals to assign the values.

You can centralize these using variables but users will be able to override it.

```
resource "aws_security_group" "ec2-sg" {
  name        = "myec2-sg"

  ingress {
    description      = "Allow Inbound from Secret Application"
    from_port        = local.app_port
    to_port          = local.app_port
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

locals {
  app_port = 8443
}
```

.

Module 4: Module Outputs

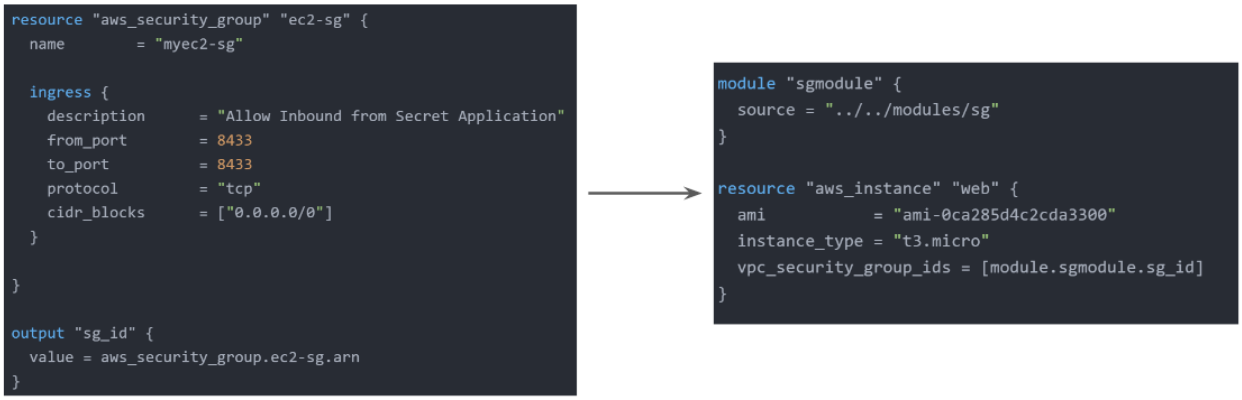
4.1 Revising Output Values

Output values make information about your infrastructure available on the command line, and can expose information for other Terraform configurations to use.

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

4.2 Accessing Child Module Outputs

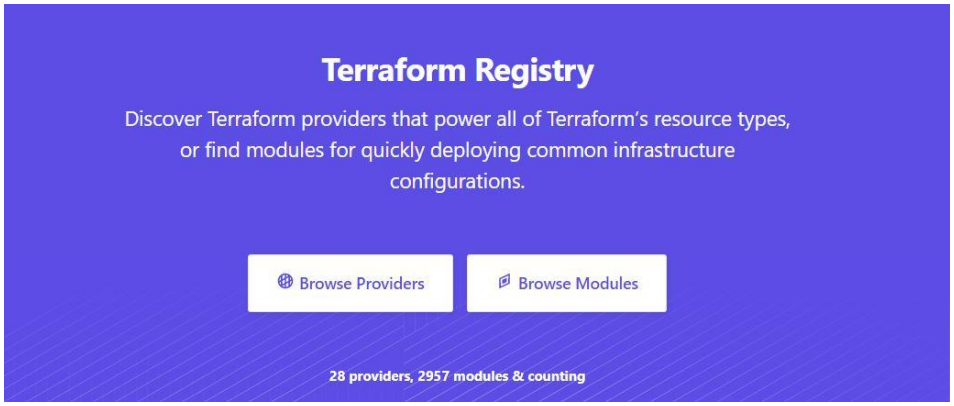
In a parent module, outputs of child modules are available in expressions as module.<MODULE NAME>.<OUTPUT NAME>



Module 5: Terraform Registry

The Terraform Registry is a repository of modules written by the Terraform community.

The registry can help you get started with Terraform more quickly



5.1 Module Location

If we intend to use a module, we need to define the path where the module files are present.

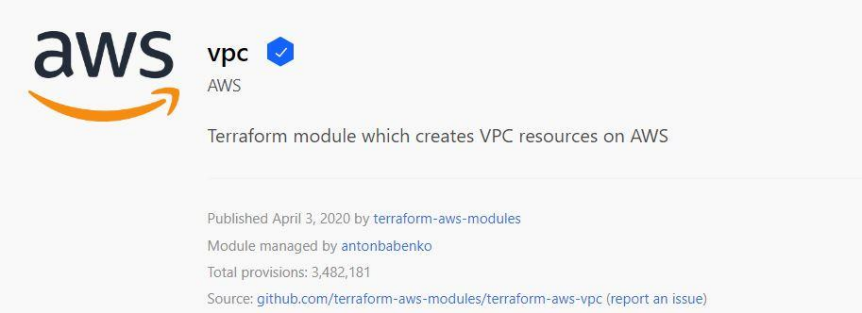
The module files can be stored in multiple locations, some of these include:

- Local Path
- GitHub
- Terraform Registry
- S3 Bucket
- HTTP URLs

5.2 Verified Modules in Terraform Registry

Within Terraform Registry, you can find verified modules that are maintained by various third-party vendors.

These modules are available for various resources like AWS VPC, RDS, ELB, and others



Verified modules are reviewed by HashiCorp and actively maintained by contributors to stay up-to-date and compatible with both Terraform and their respective providers.

The blue verification badge appears next to modules that are verified.

Module verification is currently a manual process restricted to a small group of trusted HashiCorp partners.

5.3 Using Registry Modules in Terraform

To use the Terraform Registry module within the code, we can make use of the source argument that contains the module path.

Below code references the EC2 Instance module within terraform registry.

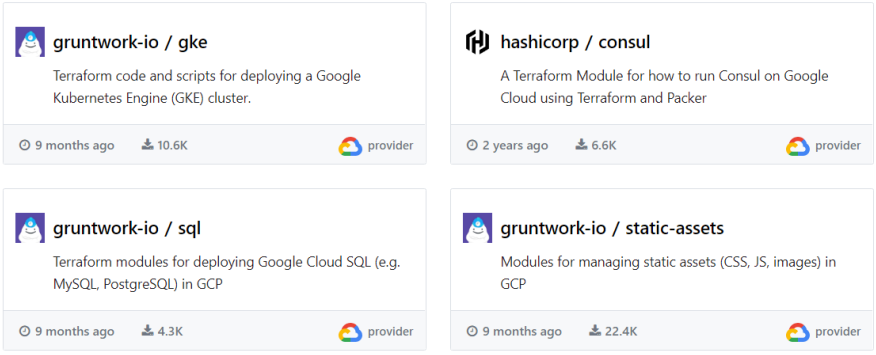
```
module "ec2-instance" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "2.13.0"
  # insert the 10 required variables here
}
```

Module 6: Requirement for Publishing Modules in Terraform Registry

6.1 Overview of Publishing Modules

Anyone can publish and share modules on the Terraform Registry.

Published modules support versioning, automatically generate documentation, allow browsing version histories, show examples and READMEs, and more.

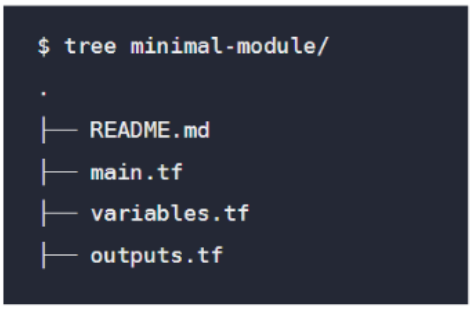


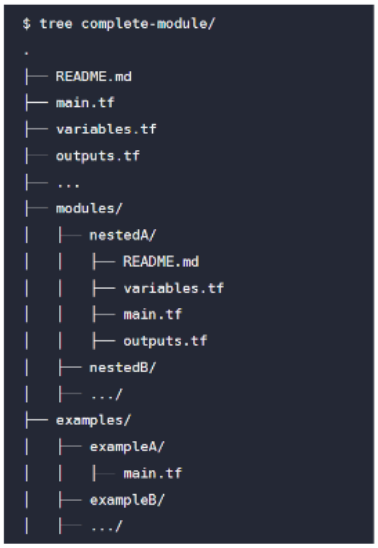
6.2 Requirements for Publishing Module

Requirement	Description
GitHub	The module must be on GitHub and must be a public repo. This is only a requirement for the public registry.
Named	Module repositories must use this three-part name format terraform-<PROVIDER>-<NAME>
Repository description	The GitHub repository description is used to populate the short description of the module.
Standard module structure	The module must adhere to the standard module structure.
x.y.z tags for releases	The registry uses tags to identify module versions. Release tag names must be a semantic version, which can optionally be prefixed with a v. For example, v1.0.4 and 0.9.2

6. 3 Standard Module Structure

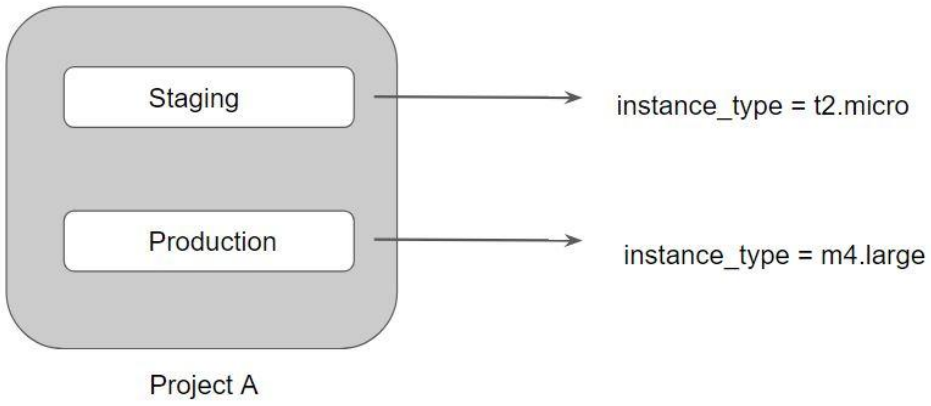
The standard module structure is a file and directory layout that is recommend for reusable modules distributed in separate repositories





Module 7: Terraform Workspace

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



Terraform starts with a single workspace named "default".

This workspace is special both because it is the default and also because it cannot ever be deleted.

If you've never explicitly used workspaces, then you've only ever worked on the "default" workspace.

Workspaces are managed with the terraform workspace set of commands.

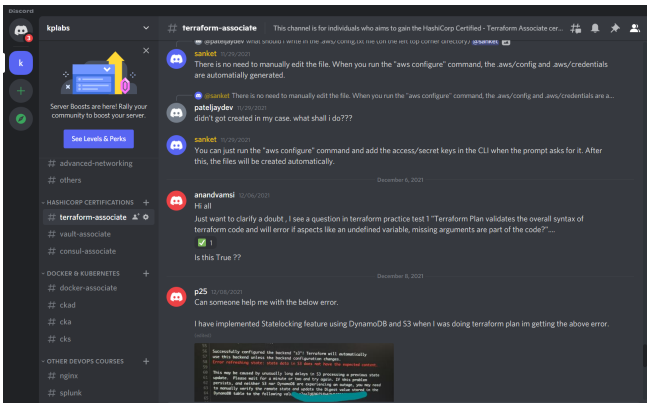
To create a new workspace and switch to it, you can use terraform workspace new; to switch workspaces you can use terraform workspace select; etc.

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



KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 5 - Remote State Management

ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

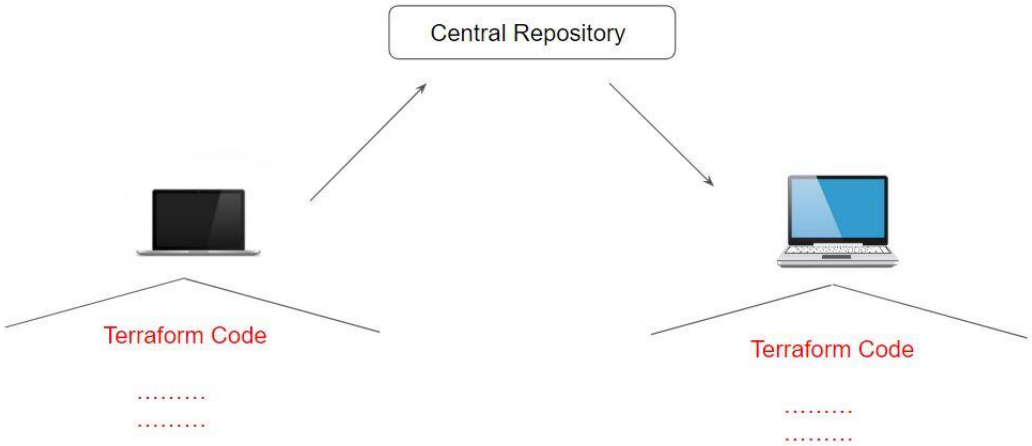
Domain 5 - Remote State Management

Module 1: Integrating with GIT for team management

Till now, we have been working with terraform code locally.

However, storing your configuration files locally is not always an idea specifically in the scenario were other members of the team are also working on Terraform.

For such cases, it is important to store your Terraorm code to a centralized repository like in Git.



Module 2: Module Sources in Terraform

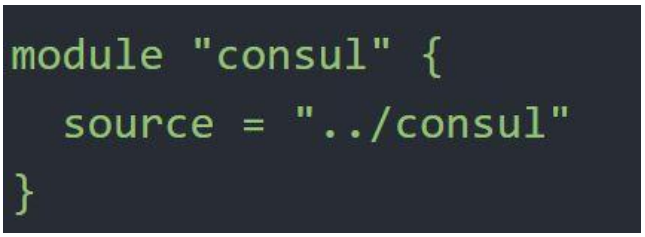
The source argument in a module block tells Terraform where to find the source code for the desired child module.

- Local paths
- Terraform Registry
- GitHub
- Bitbucket
- Generic Git, Mercurial repositories
- HTTP URLs
- S3 buckets
- GCS buckets

Let us explore some of the supported module sources.

2.1 Local Path

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



2.2 Git Module Source

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"
}
```

2.3 Referencing to a Branch

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.

Module 3: Terraform & GitIgnore

The .gitignore file is a text file that tells Git which files or folders to ignore in a project.

.gitignore
conf/
*.artifacts
credentials

Depending on the environment, it is recommended to avoid committing certain files to GIT.

Files to Ignore	Description
.terraform	This file will be recreated when terraform init is run.
terraform.tfvars	Likely to contain sensitive data like usernames/passwords and secrets.
terraform.tfstate	Should be stored in the remote side.
crash.log	If terraform crashes, the logs are stored to a file named crash.log

Module 4: Terraform Backend

4.1 Basics of Backends

Backends primarily determine where Terraform stores its state.

By default, Terraform implicitly uses a backend called local to store state as a local file on disk.

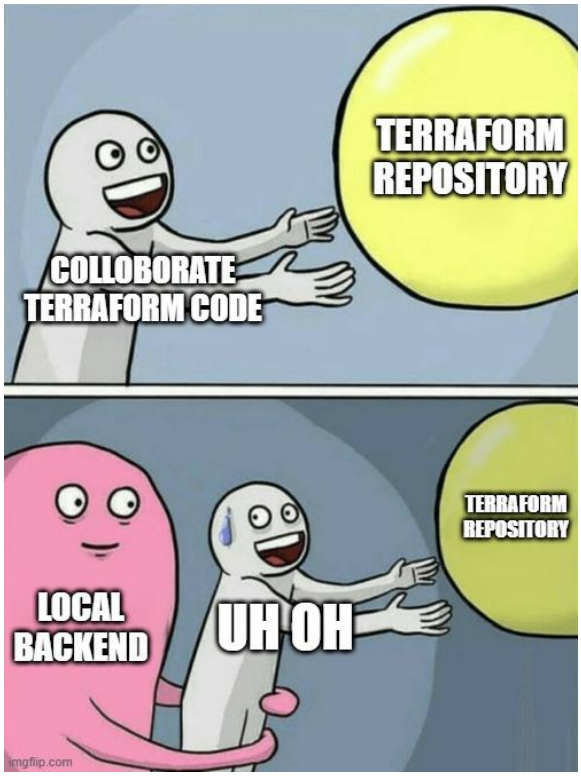
demo.tf

terraform.tfstate

4.2 Challenge with Local Backend

Nowadays Terraform projects are handled and collaborated by an entire team.

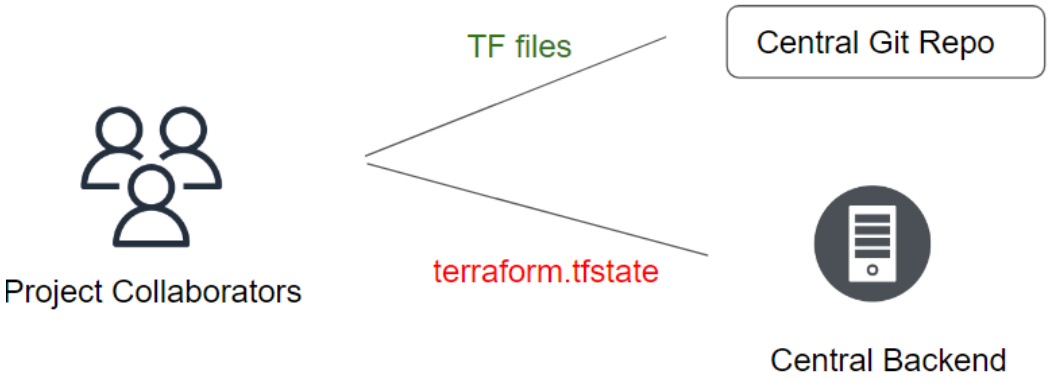
Storing the state file in the local laptop will not allow collaboration.



4.3 Ideal Architecture

Following describes one of the recommended architectures:

The Terraform Code is stored in Git Repository.
The State file is stored in a Central backend.



4.4 Backends Supported in Terraform

Terraform supports multiple backends that allow remote service related operations.

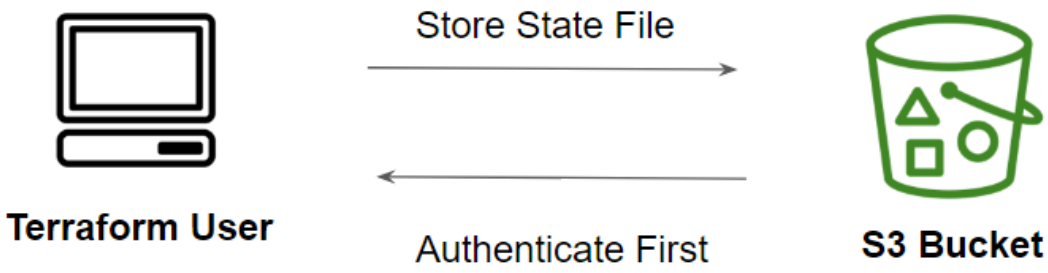
Some of the popular backends include:

- S3
- Consul
- Azurerm
- Kubernetes
- HTTP
- ETCD

4.5 Important Note

Accessing state in a remote service generally requires some kind of access credentials

Some backends act like plain "remote disks" for state files; others support locking the state while operations are being performed, which helps prevent conflicts and inconsistencies.



.

Module 5: State File Locking

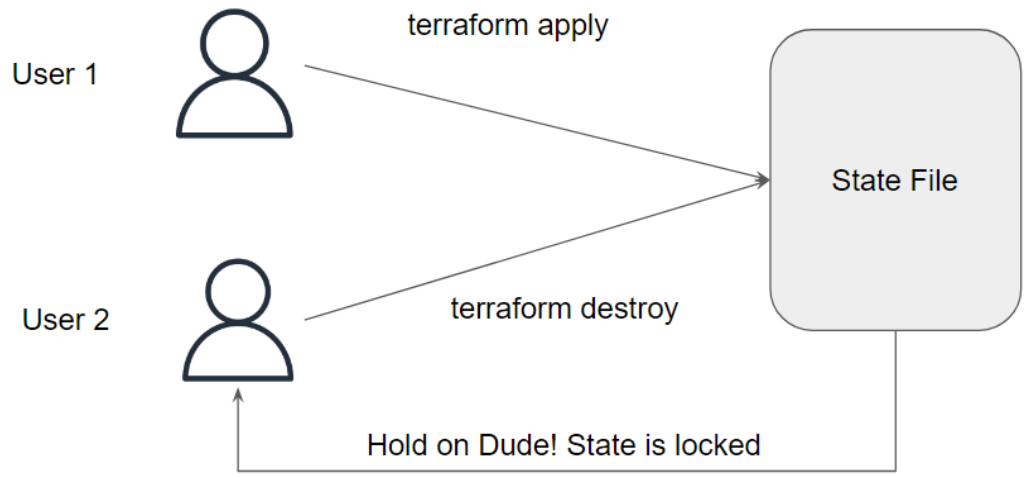
5.1 Understanding State Lock

Whenever you are performing write operation, terraform would lock the state file.

This is very important as otherwise during your ongoing terraform apply operations, if others also try for the same, it can corrupt your state file.

```
C:\Users\Zeal Vora\Desktop\tf-demo\remote-backend>terraform plan

Error: Error acquiring the state lock

Error message: Failed to read state file: The state file could not be read: read terraform.tfstate: The process
cannot access the file because another process has locked a portion of the file.

Terraform acquires a state lock to protect the state from being written
by multiple users at the same time. Please resolve the issue above and try
again. For most commands, you can disable locking with the "-lock=false"
flag, but this is not recommended.
```

Following diagram illustrates the basic working.



5.2 Important Note

State locking happens automatically on all operations that could write state. You won't see any message that it is happening

If state locking fails, Terraform will not continue

Not all backends support locking. The documentation for each backend includes details on whether it supports locking or not.

5.3 Force Unlocking State

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

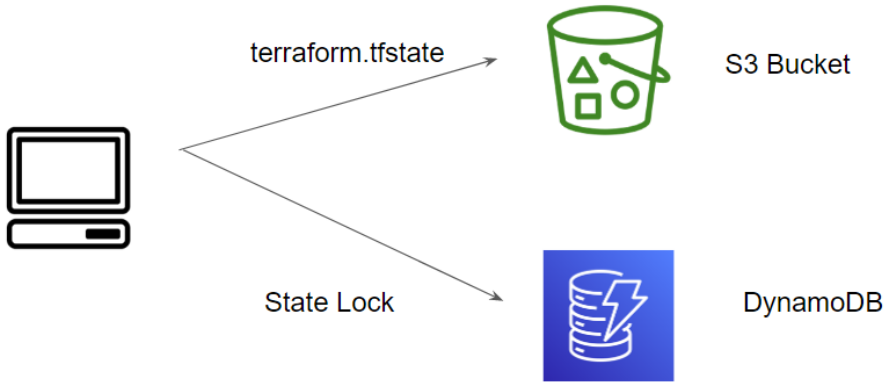
If you unlock the state when someone else is holding the lock it could cause multiple writers.

Force unlock should only be used to unlock your own lock in the situation where automatic unlocking failed.

Module 6: State Locking in S3 Backend

By default, S3 does not support State Locking functionality.

You need to make use of the DynamoDB table to achieve state locking functionality.



Module 7: Terraform State Management

As your Terraform usage becomes more advanced, there are some cases where you may need to modify the Terraform state.

It is important to never modify the state file directly. Instead, make use of terraform state command.

There are multiple sub-commands that can be used with terraform state, these include:

State Sub Command	Description
list	List resources within terraform state file.
mv	Moves item with terraform state.
pull	Manually download and output the state from remote state.
push	Manually upload a local state file to remote state.
rm	Remove items from the Terraform state
show	Show the attributes of a single resource in the state.

7.1 Sub Command - List

The terraform state list command is used to list resources within a Terraform state.

```
bash-4.2# terraform state list
aws_iam_user.lb
aws_instance.webapp
```

7.2 Sub Command - Move

The terraform state mv command is used to move items in a Terraform state.

This command is used in many cases in which you want to rename an existing resource without destroying and recreating it.

Due to the destructive nature of this command, this command will output a backup copy of the state prior to saving any changes

Overall Syntax:

terraform state mv [options] SOURCE DESTINATION

7.3 Sub Command - Pull

The terraform state pull command is used to manually download and output the state from a remote state.

This is useful for reading values out of state (potentially pairing this command with something like jq).

7.4 Sub Command - Push

The terraform state push command is used to manually upload a local state file to remote state.

This command should rarely be used.

7.5 Sub Command - Remove

The terraform state rm command is used to remove items from the Terraform state.

Items removed from the Terraform state are not physically destroyed.

Items removed from the Terraform state are only no longer managed by Terraform

For example, if you remove an AWS instance from the state, the AWS instance will continue running, but terraform plan will no longer see that instance.

7.6 Sub Command - Show

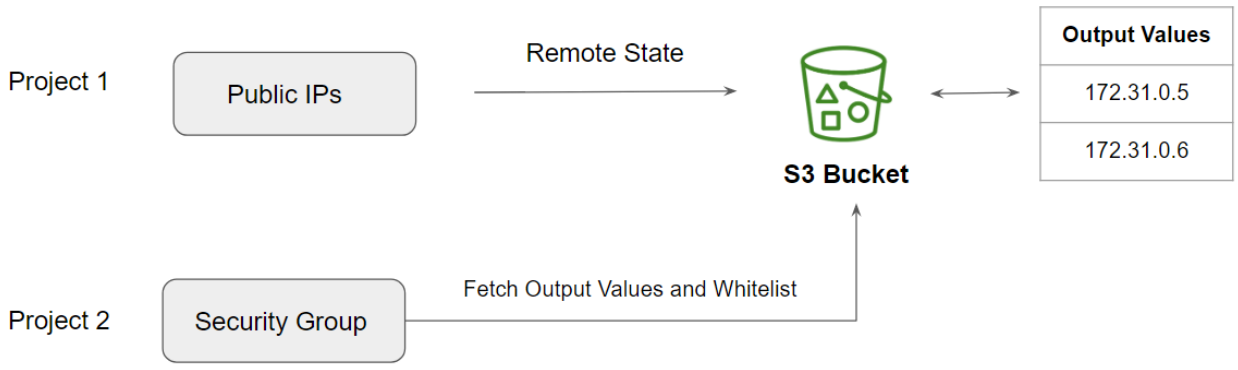
The terraform state show command is used to show the attributes of a single resource in the Terraform state.

```
bash-4.2# terraform state show aws_instance.webapp
# aws_instance.webapp:
resource "aws_instance" "webapp" {
    ami                          = "ami-082b5a644766e0e6f"
    arn                          = "arn:aws:ec2:us-west-2:018721151861:instance/i-0107ea9ed06c467e0"
    associate_public_ip_address  = true
    availability_zone            = "us-west-2b"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-0107ea9ed06c467e0"
    instance_state               = "running"
    instance_type                = "t2.micro"
```

Module 8: Connecting Remote States

8.1 Basics of Terraform Remote State

The **terraform_remote_state** data source retrieves the root module output values from some other Terraform configuration, using the latest state snapshot from the remote backend.



8.2 Step 1 - Create a Project with Output Values & S3 Backend

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

output "eip_addr" {
  value = aws_eip.lb.public_ip
}
```



```
terraform {
  backend "s3" {
    bucket = "kplabs-terraform-backend"
    key    = "network/eip.tfstate"
    region = "us-east-1"
  }
}
```

8.3 Step 2 - Reference Output Values from Different Project

```
data "terraform_remote_state" "eip" {
  backend = "s3"
  config = {
    bucket = "kplabs-terraform-backend"
    key    = "network/eip.tfstate"
    region = "us-east-1"
  }
}
```

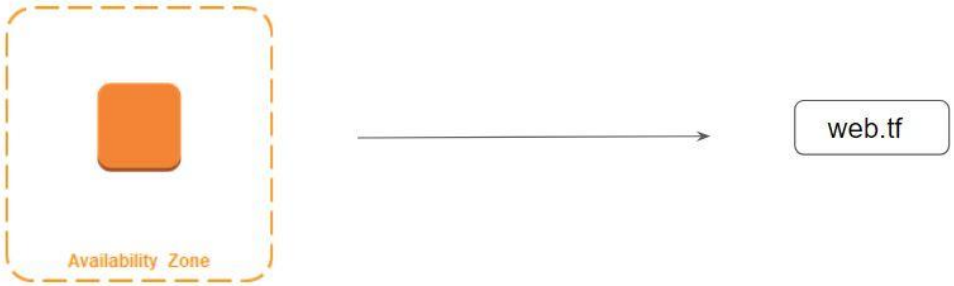


```
resource "aws_security_group" "allow_tls" {
  name        = "allow_tls"
  description = "Allow TLS inbound traffic"

  ingress {
    description      = "TLS from VPC"
    from_port        = 443
    to_port          = 443
    protocol         = "tcp"
    cidr_blocks      = ["${data.terraform_remote_state.eip.outputs.eip_addr}/32"]
  }

}
```

Module 9: Terraform Import

It might happen that there is a resource that is already created manually.

In such a case, any change you want to make to that resource must be done manually.



Terraform is able to import existing infrastructure. This allows you to take resources 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. A future version of Terraform will also generate configuration.

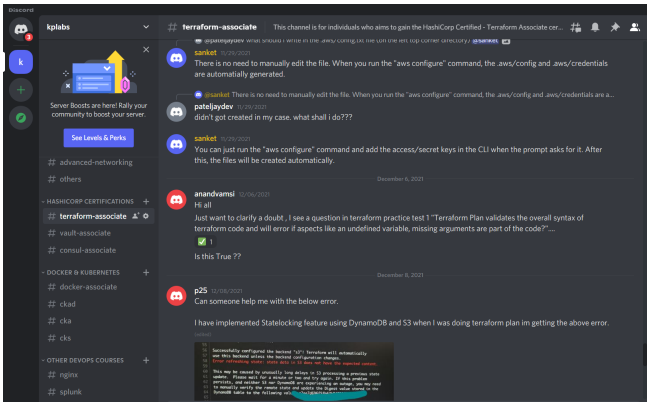
Because of this, prior to running terraform import it is necessary to write manually a resource configuration block for the resource, to which the imported object will be mapped.

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



KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 6

ISSUED BY
Zeal

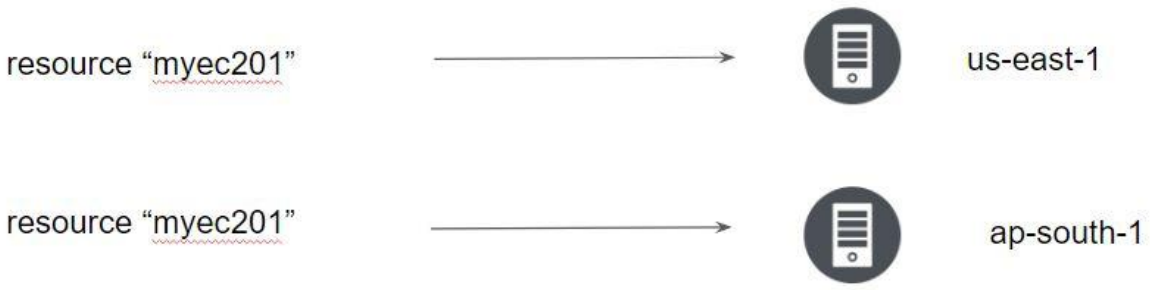
REPRESENTATIVE
instructors@kplabs.in

Domain 6 - Security Primer

Module 1: Provider Configuration

Till now, we have been hardcoding the aws-region parameter within the providers.tf

This means that resources would be created in the region specified in the providers.tf file.



By default, resources use a default provider configuration inferred from the first word of the resource type name.

For example, a resource of type aws_instance uses the default (un-aliased) aws provider configuration unless otherwise stated.

To select an aliased provider for a resource or data source, set its provider meta-argument to a <PROVIDER NAME>.<ALIAS> reference:

```
resource "aws_instance" "foo" {
  provider = aws.west

  # ...
}
```

Module 2: Handling Multiple AWS Profiles in Terraform

You can optionally define multiple configurations for the same provider, and select which one to use on a per-resource or per-module basis.

The primary reason for this is to support multiple regions for a cloud platform.

To include multiple configurations for a given provider, include multiple provider blocks with the same provider name, but set the alias meta-argument to an alias name to use for each additional configuration. For example:

```
# The default provider configuration
provider "aws" {
  region = "us-east-1"
}

# Additional provider configuration for west coast region
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.

For providers that have no required configuration arguments, the implied empty configuration is considered to be the default provider configuration.

Module 3: Sensitive Parameter

With the organization managing its entire infrastructure in terraform, it is likely that you will see some sensitive information embedded in the code.

When working with a field that contains information likely to be considered sensitive, it is best to set the Sensitive property on its schema to true

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

Setting the sensitive to “true” will prevent the field's values from showing up in CLI output and in Terraform Cloud

It will not encrypt or obscure the value in the state, however.

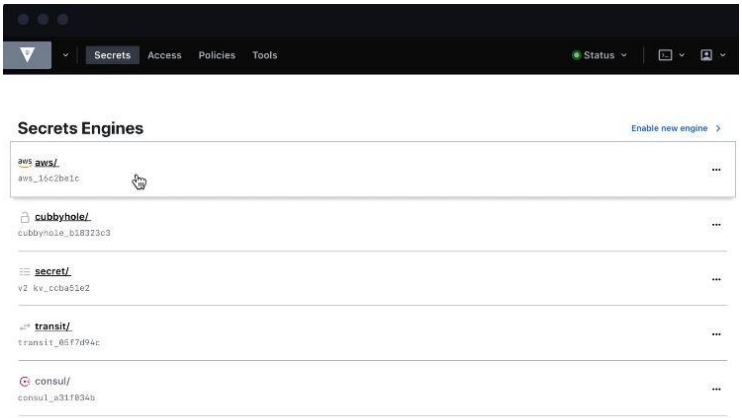
```
C:\Users\Zeal Vora\Desktop\terraform\sensitive data>terraform apply

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

Outputs:

db_password = <sensitive>
```

Module 4: Overview of HashiCorp Vault

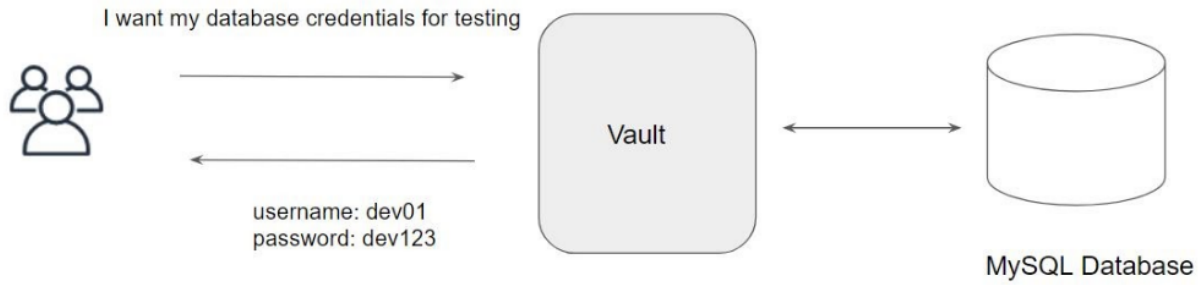
HashiCorp Vault allows organizations to securely store secrets like tokens, passwords, certificates along with access management for protecting secrets.

One of the common challenges nowadays in an organization is “Secrets Management”

Secrets can include database passwords, AWS access/secret keys, API Tokens, encryption keys and others.



Vault can also generate dynamic secrets like AWS Access/Secret keys that has validity for a limited time.

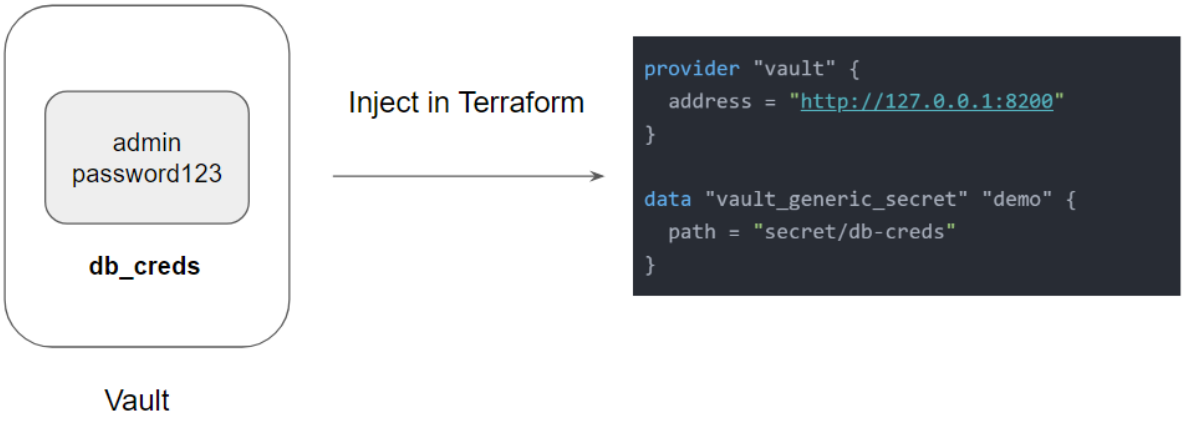


Once Vault is integrated with multiple backends, your life will become much easier and you can focus more on the right work.

Major aspects related to Access Management can be taken over by vault.

Module 5: Terraform and Vault Integration

The Vault provider allows Terraform to read from, write to, and configure HashiCorp Vault.



Important Note:

Interacting with Vault from Terraform causes any secrets that you read and write to be persisted in Terraform's state file.

KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 7

ISSUED BY

Zeal

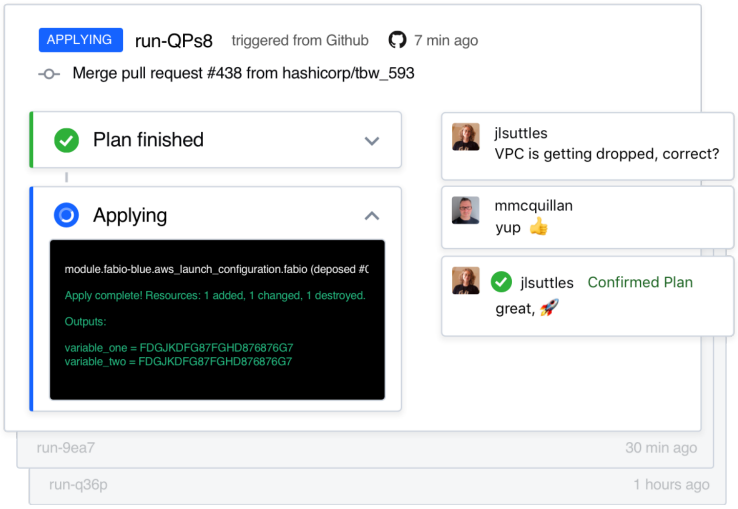
REPRESENTATIVE

instructors@kplabs.in

Domain 7 - Terraform Cloud & Enterprise Capabilities

Module 1: Overview of Terraform Cloud

Terraform Cloud manages Terraform runs in a consistent and reliable environment with various features like access controls, private registry for sharing modules, policy controls, and others.



Terraform Cloud is available as a hosted service at https://app.terraform.io.

Module 2: Overview of Sentinel

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

It enables fine-grained, logic-based policy decisions, and can be extended to use information from external sources.

Note: Sentinel policies are paid feature

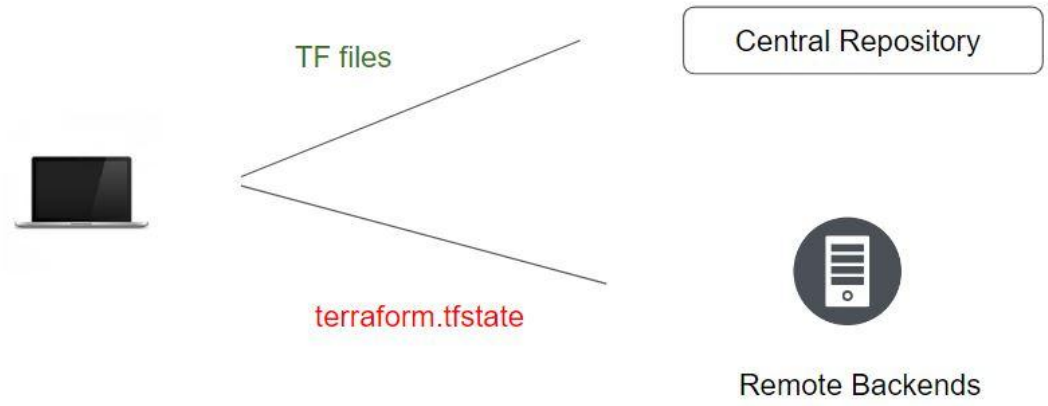
Module 3: Remote Backends

Terraform supports various types of remote backends which can be used to store state data.

As of now, we were storing state data in local and GIT repository.

Depending on remote backends that are being used, there can be various features.

- Standard BackEnd Type: State Storage and Locking
- Enhanced BackEnd Type: All features of Standard + Remote Management



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.

Remote plans and applies use variable values from the associated Terraform Cloud workspace.

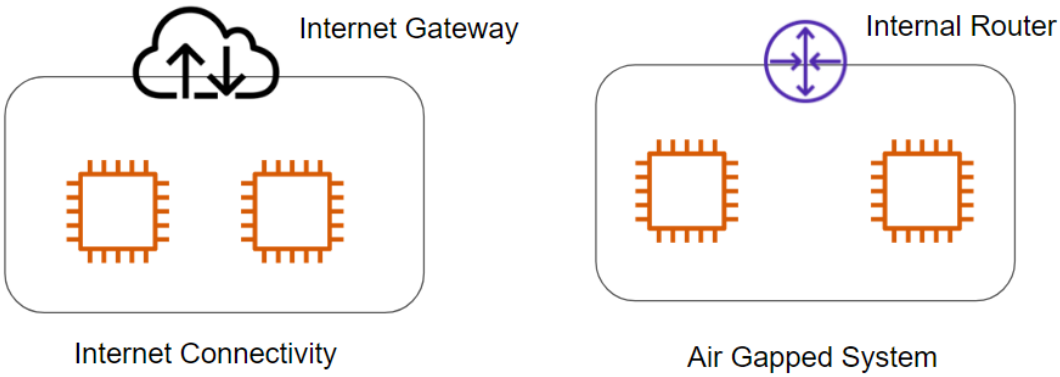
Terraform Cloud can also be used with local operations, in which case only state is stored in the Terraform Cloud backend.

.

Module 4: Air Gapped Environments

4.1 Understanding Concept of Air Gap

An air gap is a network security measure employed to ensure that a secure computer network is physically isolated from unsecured networks, such as the public Internet.



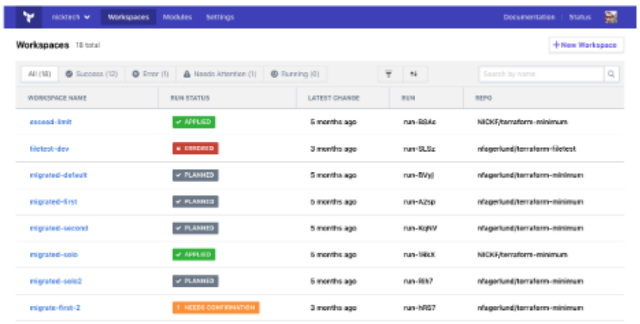
4.2 Usage of Air Gapped Systems

Air Gapped Environments are used in various areas. Some of these include:

- Military/governmental computer networks/systems
- Financial computer systems, such as stock exchanges
- Industrial control systems, such as SCADA in Oil & Gas fields

4.3 Terraform Enterprise Installation Methods

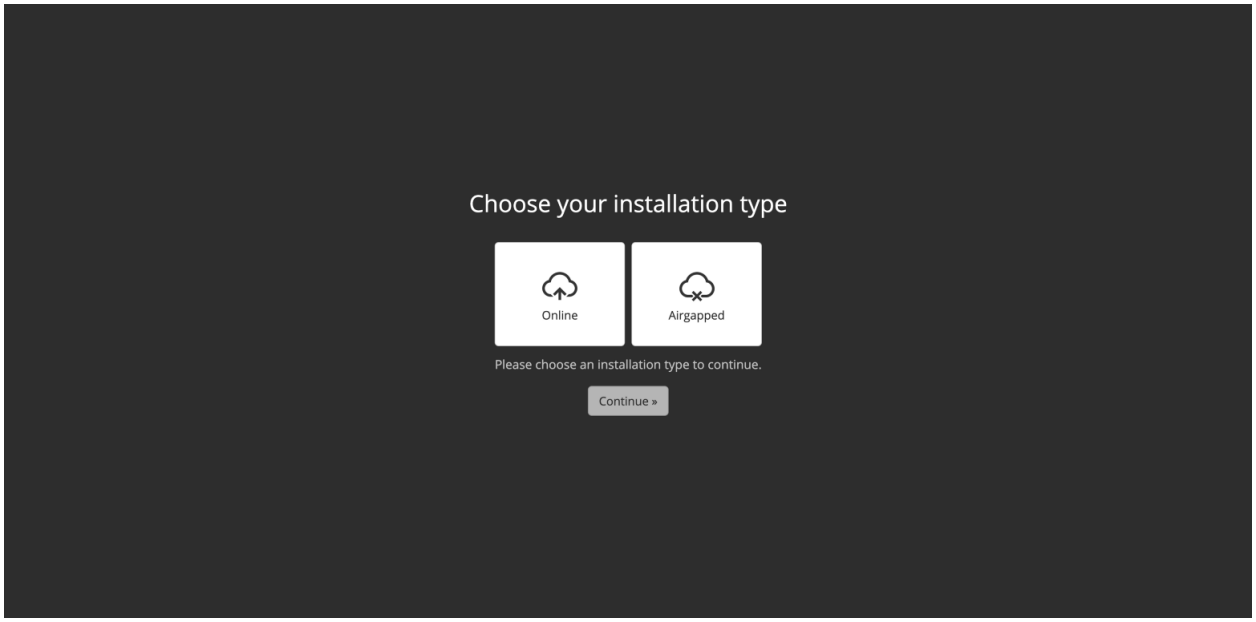
Terraform Enterprise installs using either an online or air gapped method and as the names infer, one requires internet connectivity, the other does not

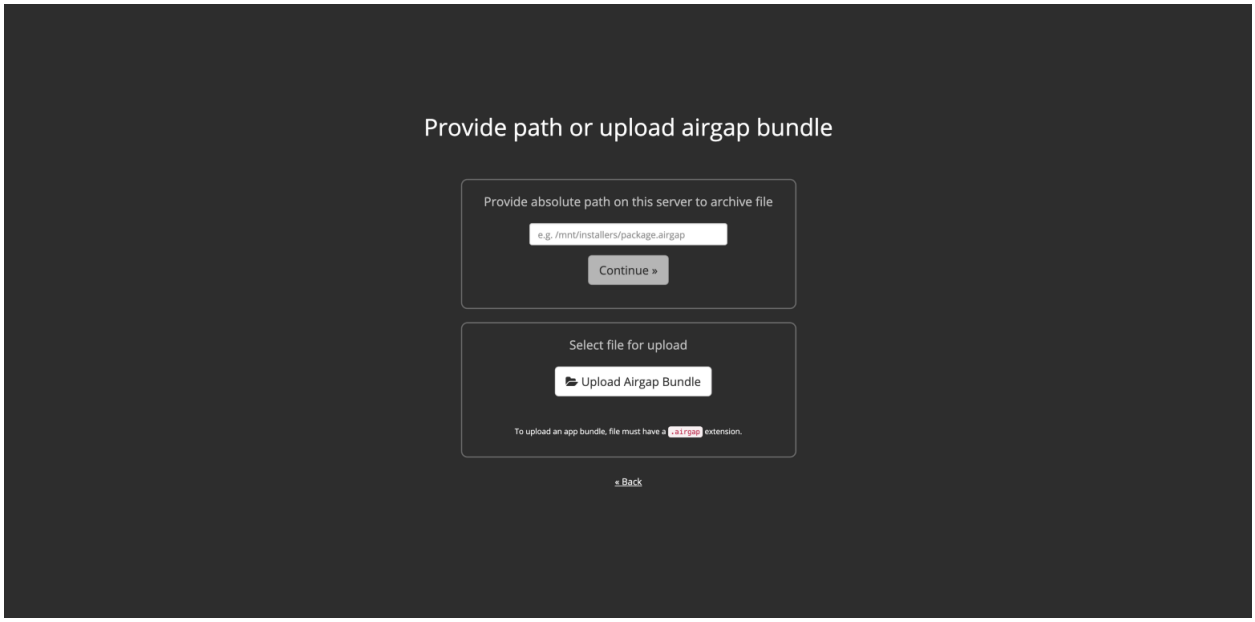


Terraform Enterprise

Air Gap Install

Isolated Server




Provide path or upload airgap bundle
Provide absolute path on this server to archive file
e.g. /mnt/installers/package.airgap
Continue »
Select file for upload
Upload Airgap Bundle
To upload an app bundle, file must have a .airgap extension.
« Back