

# **HashiCorp Certified: Terraform Authoring & Operations Professional**



# Understanding the Need

Terraform Associate certification is primarily an MCQ-based exam and is relatively easier to pass even by memorizing the concepts.

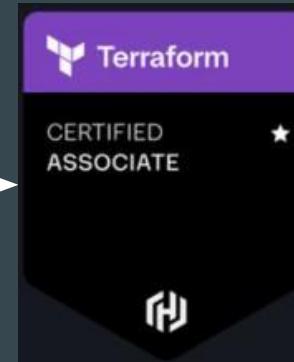


What are Providers in Terraform?

Option A

Option B

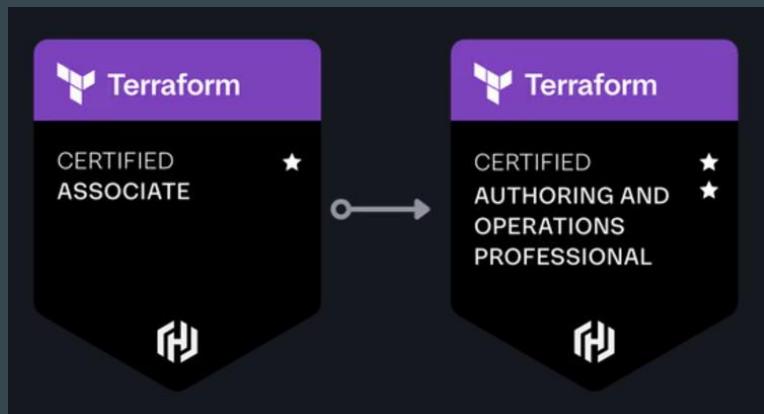
Option C



Sample Question

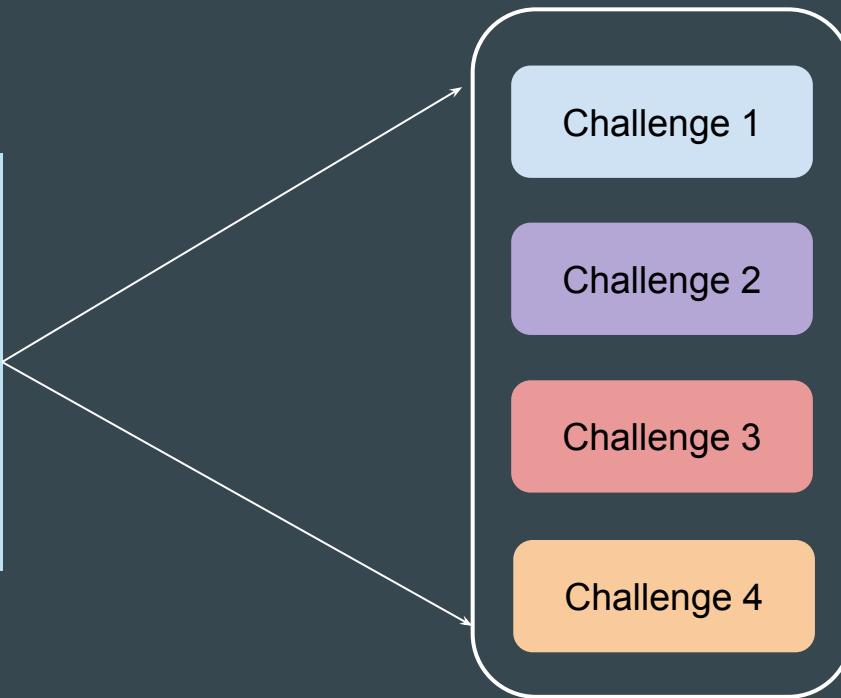
# Introducing Professional Certification

Terraform Authoring and Operations Professional exam assesses both **advanced configuration** authoring and a **deep understanding** of Terraform workflows.



# Lab Based Exams

Terraform Professional exam is a **4-hour intensive lab-based exam** where you will have to solve multiple scenarios presented to you.



# What Does this Course Cover?

We cover ALL the topics of the official exams.

Base Terraform knowledge is a **pre-requisite** for this Professional-level course.



# Base Course Structure

- Domain 1 - Manage resource lifecycle
- Domain 2 - Develop and troubleshoot dynamic configuration
- Domain 3 - Develop collaborative Terraform workflows
- Domain 4 - Create, maintain, and use Terraform modules
- Domain 5 - Configure and use Terraform providers
- Domain 6 - Collaborate on infrastructure as code using HCP Terraform
- Domain 7 - **AWS Integration Practicals**
- Domain 8 - Exam Preparation Section

# The Exciting Part

We have an exam preparation section to help you get prepared for the exam.

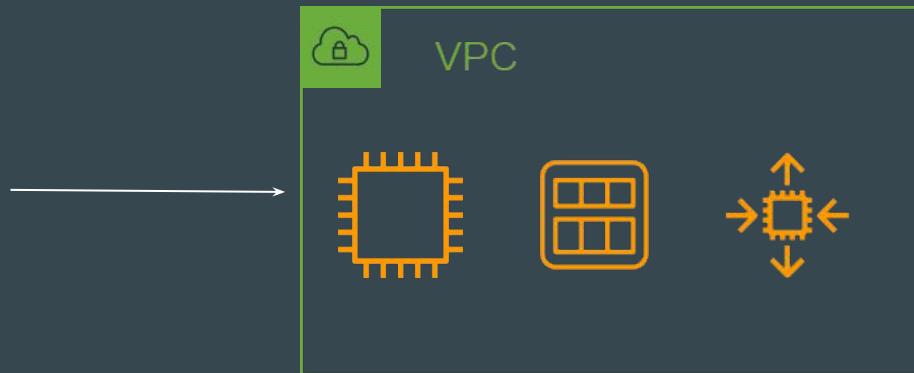
We also have practice tests available as part of the course.



# Important Point to Note

The lab-based challenges in Terraform Professional exams are based on specific Cloud provider.

The exam has been launched with the initial support of the **AWS** cloud provider.



# What About Users from Different Provider?

We have intentionally separated deep AWS-related topics in a separate section called AWS Integrations.

Other sections use very basic AWS services to demonstrate Terraform concepts. It is very similar to our Terraform Associate level course.



# About Me

- DevSecOps Engineer - Defensive Security.
- Teaching is one of my passions.
- I have total of 16 courses, and around 400,000+ students now.

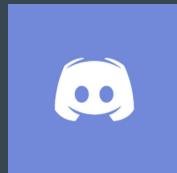
Something about me :-



- HashiCorp Certified [Terraform Professional] [Vault and Consul Associate]
- AWS Certified [Advanced Networking, Security Specialty, DevOps Pro, SA Pro, ...]
- RedHat Certified Architect (RHCA) + 13 more Certifications
- Part time Security Consultant

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **About the Course and Resources**

# 1 - Aim of This Course

The **primary aim** of this course is to learn.

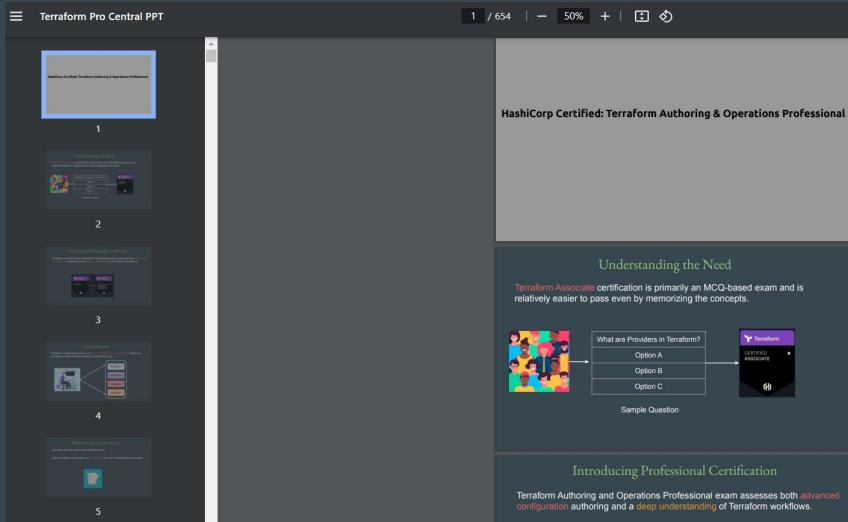
Certification is the byproduct of learning.



## 2 - PPT Slides PDF

ALL the slides that we use in this course is available to download as PDF.

We have around 700+ slides that are available to download.



## 3 - PPT Version

The course is updated regularly and so are the PPTs.

We add the PPT release date in the PPT itself and the lecture from which you download the PPTs.

PPT Version

PPT Release Date = 6th December 2024



### Central PPT Notes

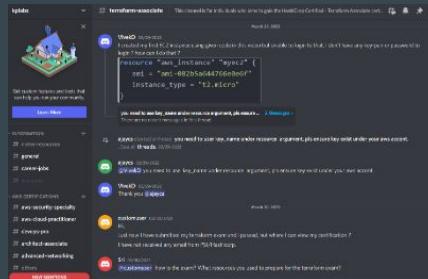
PPT Release Date - 1st January 2025

## 4 - Our Community (Optional)

You can **join our Discord community** for any queries / discussions. You can also connect with other students going through the same course in Discord (Optional)

Discord Link: <https://kplabs.in/chat>

Category: #terraform-professional



# 5 - Course Resource - GitHub

All the code that we use during practicals have been added to our GitHub page.

Section Name in the Course and GitHub are same for easy finding of code.

The screenshot shows a GitHub repository named "terraform-professional". The repository is private, has 1 branch, and 0 tags. The main branch contains the following files:

File	Description	Last Commit
Domain 1 - Manage resource lifecycle	Base Code for TF Pro Certification	now
Domain 2 - Develop and troubleshoot dynam...	Base Code for TF Pro Certification	now
Domain 3 - Develop collaborative Terraform w...	Base Code for TF Pro Certification	now
Domain 4 - Create, maintain, and use Terrafor...	Base Code for TF Pro Certification	now
Domain 5 - Configure and use Terraform provi...	Base Code for TF Pro Certification	now
Domain 6 - Collaborate on infrastructure as co...	Base Code for TF Pro Certification	now
Domain 7 - AWS Integration	Base Code for TF Pro Certification	now
Readme.md	Base Code for TF Pro Certification	now

## 6 - Lab Environment

We use folder of **kplabs-terraform** as a base for creating and managing TF files.

**Visual Studio Code** is the default code editor used for this course and exams.



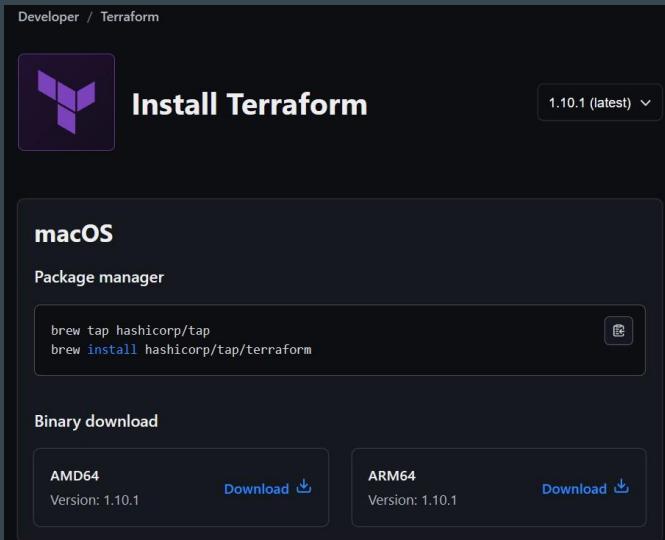
A screenshot of the Visual Studio Code interface. On the left, there's a tree view showing the project structure: C: > kplabs-terraform > main.tf > ... . The main editor area shows the following Terraform code:

```
resource "aws_instance" "a" {
    ami      = "ami-0453ec754f44f9a4a"
    instance_type = "t2.micro"
}
```

## 7 - Terraform Version

We use the **latest version of Terraform** - whichever is available at the time.

In the Exam Preparation section challenges, you can switch to the exact Terraform version of exam for practice.



## 8 - Some External Topics

We have included few external topics that are not part of exam blueprint but are important from real world environment and interviews.

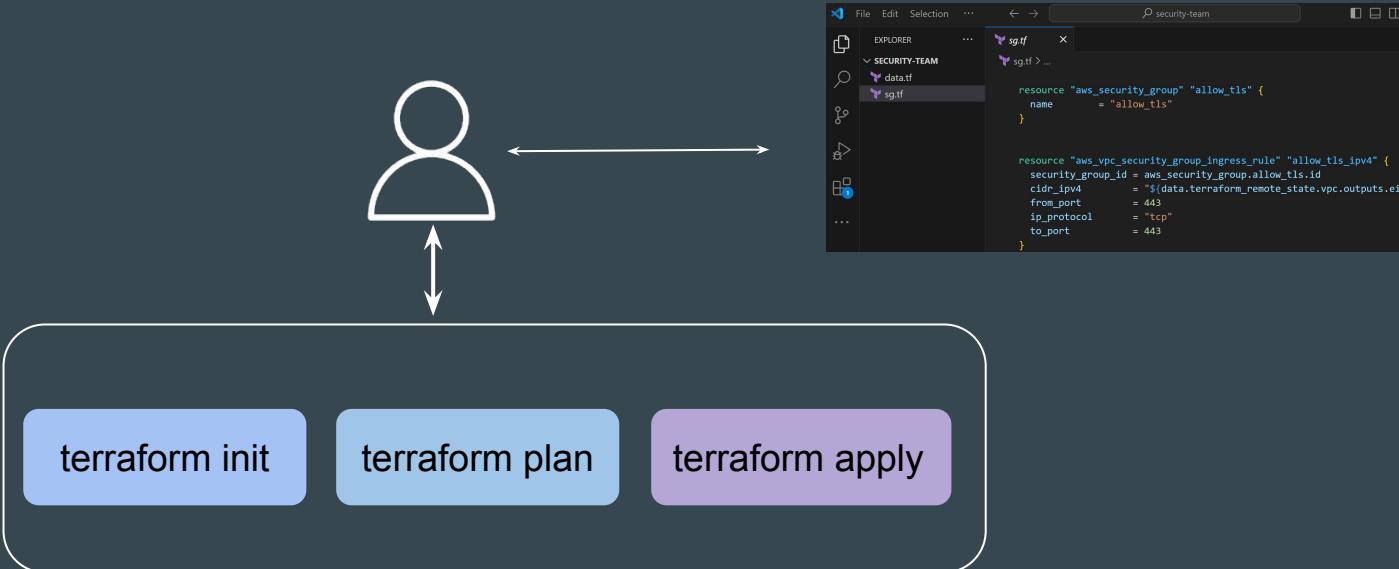
We have clearly marked them as **External**

✓ Lecture 14:  Static Analysis Tool for Terraform - External	▼
✓ Lecture 15:  Installing Checkov in Linux - External	▼
✓ Lecture 16:  Analyzing Terraform Code with Checkov - External	▼

# **Terraform in Automation Environments**

# Standard Workflow

In many organizations, a **standard operator based workflow** is used to create and manage environments using Terraform.



# Advantages of Standard Operator-Based Approach

1. Direct control over each step of the process
2. Easier debugging and troubleshooting
3. Flexibility to make ad-hoc changes
4. Simpler setup, no need for additional automation infrastructure



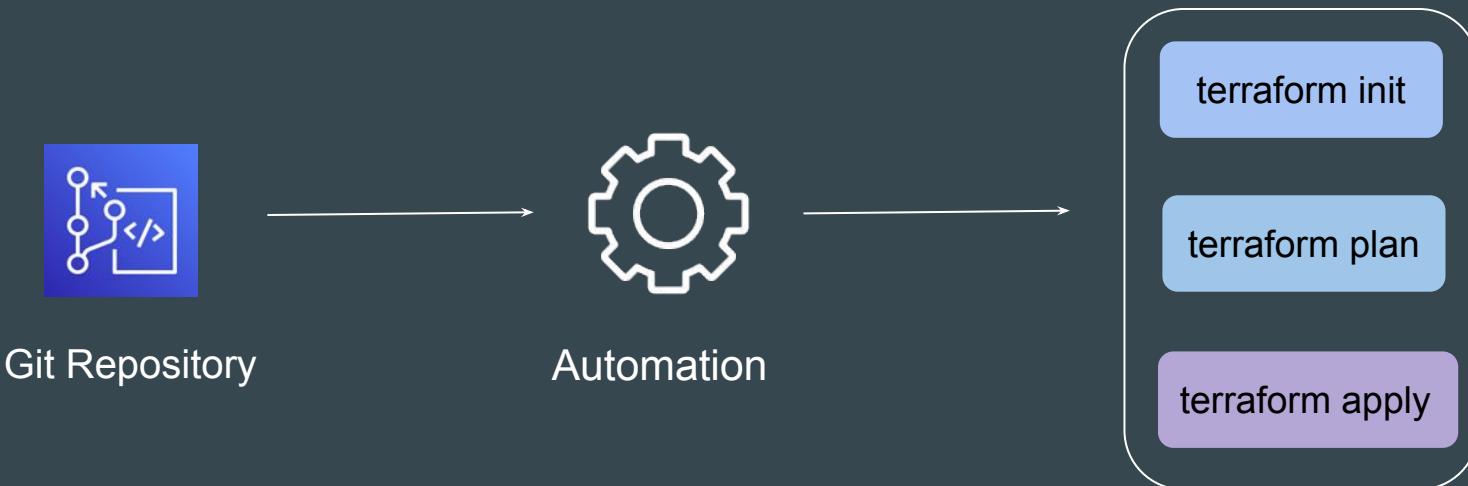
# Disadvantages of Standard Operator-Based Approach

1. Manual process prone to human error
2. Lack of standardization across team members
3. Not ideal for large-scale environments where repetitive manual execution can be time-consuming and error-prone.
4. Relies heavily on the knowledge and expertise of the operators, which can be a bottleneck if not well-documented.
5. Availability of Human Operator.

# Automation Based Workflows

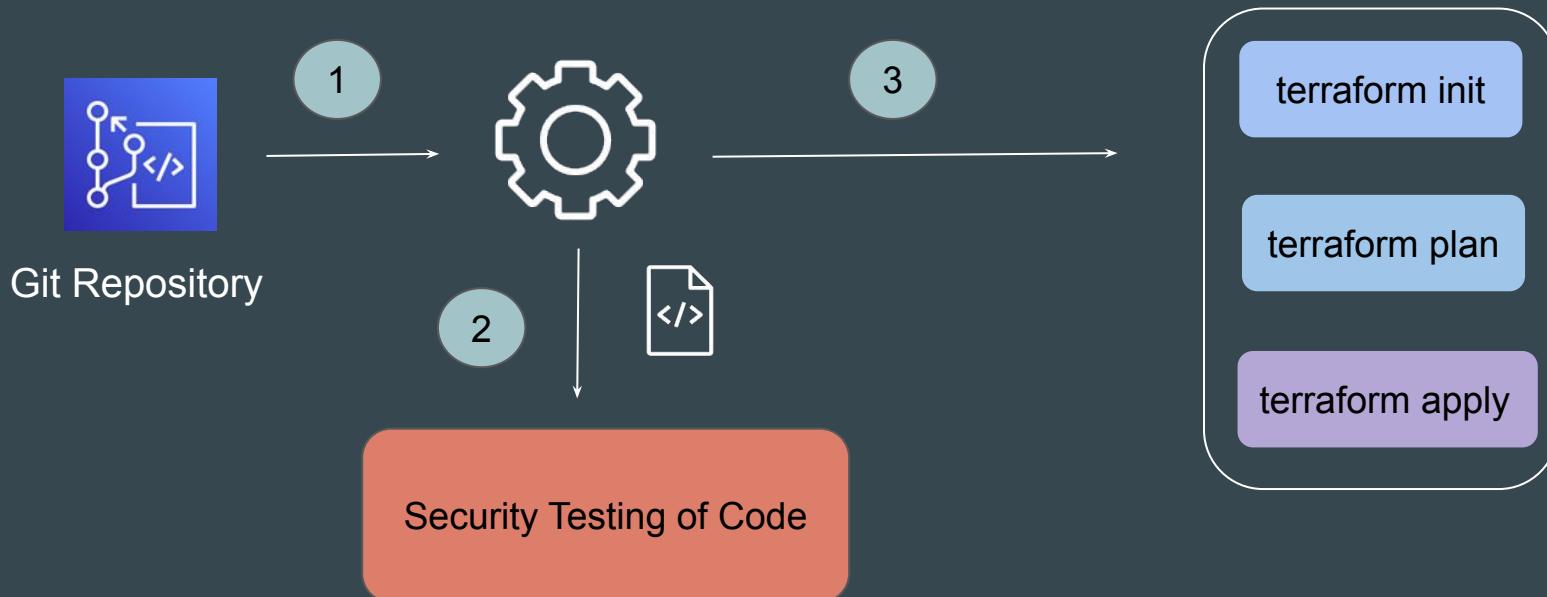
In larger enterprises, the standard operator based workflows are not always used.

Infrastructure is created using Terraform with some form of Automation.



# Automation Environments can be Complex

Automation environments can involve complex set of workflow steps to ensure consistency, security of the deployed infrastructure resource.



# What is this Automation Component

1. This can be a simple to complex shell scripts or Python program that completes the entire workflow.
2. This can be an entire CI/CD pipeline based workflow using Jenkins or other tools.

# Base for Automation

Although the major commands of Terraform remains to be same in manual and automated approaches, the sub-options of CLI command changes.

## Manual Env

```
terraform init
```

```
terraform plan
```

```
terraform apply
```

## Automated Env

```
terraform init -input=false
```

```
terraform plan -input=false -no-color -out=tf.plan
```

```
terraform apply "tf.plan" -auto-approve
```

# **CLI Commands and Options**

# Setting the Base

In Terraform, the CLI commands also provide various sets of options

These options enhances functionality of the command and allow for more precise control over the infrastructure management process.

terraform plan

- `-compact-warnings` - Shows any warning messages in a compact form which includes only the summary messages, unless the warnings are accompanied by at least one error and thus the warning text might be useful context for the errors.
- `-detailed-exitcode` - Returns a detailed exit code when the command exits. When provided, this argument changes the exit codes and their meanings to provide more granular information about what the resulting plan contains:
  - 0 = Succeeded with empty diff (no changes)
  - 1 = Error
  - 2 = Succeeded with non-empty diff (changes present)
- `-generate-config-out=PATH` - (Experimental) If `import` blocks are present in configuration, instructs Terraform to generate HCL for any imported resources not already present. The configuration is written to a new file at PATH, which must not already exist, or Terraform will error. If the plan fails for another reason, Terraform may still attempt to write configuration.
- `-input=false` - Disables Terraform's default behavior of prompting for input for root module input variables that have not otherwise been assigned a value. This option is

# Point to Note

Depending on the Terraform version, the command options can change.

New options can get introduced in newer versions however the standard sub-options remains to be the same.

The screenshot shows a dark-themed web page from the Terraform documentation. At the top, the URL is visible: Developer / Terraform / Terraform CLI / commands / plan. The main title is "Command: plan". Below the title, there is a brief description of what the `terraform plan` command does. To the right of the text, a dropdown menu is open, showing a list of Terraform versions. The top item in the list is "v1.9.x (latest)". Other items in the list include "v1.10.0 (alpha)", "v1.8.x", "v1.7.x", "v1.6.x", "v1.5.x", and "v1.4.x". At the bottom of the page, there is a section titled "Hands-on: Try the [Terraform: Get Started](#) tutorials. For more in-depth details on the `plan` command, check out the [Create a Terraform Plan tutorial](#)".

# Format to Define Sub-Options

Options of CLI commands are appended in the same command itself.

```
C:\Users\zealv\kplabs-terraform>terraform plan -no-color -detailed-exitcode
Terraform used the selected providers to generate the following execution plan.
following symbols:
+ create

Terraform will perform the following actions:

# aws_security_group.allow_tls will be created
+ resource "aws_security_group" "allow_tls" {
    + arn              = (known after apply)
    + description      = "Managed by Terraform"
    + egress           = (known after apply)
    + id               = (known after apply)
    + ingress          = (known after apply)
    + name             = "allow_tls"
    + name_prefix      = (known after apply)
    + owner_id         = (known after apply)
    + revoke_rules_on_delete = false
    + tags_all         = (known after apply)
    + vpc_id           = (known after apply)
}
```

# List of Commands for Terraform Version

To view a list of the commands available in your current Terraform version, run `terraform` with no additional arguments:

```
C:\Users\zealv>terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init      Prepare your working directory for other commands
  validate   Check whether the configuration is valid
  plan       Show changes required by the current configuration
  apply      Create or update infrastructure
  destroy    Destroy previously-created infrastructure

All other commands:
  console    Try Terraform expressions at an interactive command prompt
  fmt        Reformat your configuration in the standard style
  force-unlock Release a stuck lock on the current workspace
  get        Install or upgrade remote Terraform modules
  graph      Generate a Graphviz graph of the steps in an operation
  import     Associate existing infrastructure with a Terraform resource
  login      Obtain and save credentials for a remote host
  logout     Remove locally-stored credentials for a remote host
  metadata   Metadata related commands
```

# Easy Way to See Options of CLI Command

To get specific help for any specific command, use the **-help** option with the relevant subcommand.

```
C:\Users\zealv>terraform plan -h
Usage: terraform [global options] plan [options]

Generates a speculative execution plan, showing what actions Terraform
would take to apply the current configuration. This command will not
actually perform the planned actions.

You can optionally save the plan to a file, which you can then pass to
the "apply" command to perform exactly the actions described in the plan.

Plan Customization Options:

The following options customize how Terraform will produce its plan. You
can also use these options when you run "terraform apply" without passing
it a saved plan, in order to plan and apply in a single command.

-destroy      Select the "destroy" planning mode, which creates a plan
              to destroy all objects currently managed by this
              Terraform configuration instead of the usual behavior.

-refresh-only  Select the "refresh only" planning mode, which checks
                whether remote objects still match the outcome of the
                most recent Terraform apply but does not propose any
                actions to undo any changes made outside of Terraform.

-refresh=false Skip checking for external changes to remote objects
                 while creating the plan. This can potentially make
                 planning faster, but at the expense of possibly planning
                 against a stale record of the remote system state.
```

## Point to Note

In the context of Terraform CLI commands, "flags" and "options" generally refer to similar thing: they are additional parameters or switches you can use to modify the behavior of the commands.

These terms are often used interchangeably, although "flags" is the more common terminology in command-line interfaces.

# **Importance of Input=False**

# Setting the Base

For many use-cases, if values are not defined, Terraform will prompt for the input in the CLI.

Providing input through CLI is possible in manual environment but not feasible in automated environments.

```
C:\kplabs-terraform>terraform plan  
var.file_name  
Enter a value: |
```

# Input of False

With `-input=false`, Terraform will error if input was required.

Providing input through CLI is possible in manual environment but not feasible in automated environments.

```
C:\kplabs-terraform>terraform plan -input=false
Error: No value for required variable

  on main.tf line 12:
  12: variable "file_name" {}

The root module input variable "file_name" is not set, and has no default value.
argument to provide a value for this variable.
```

## Point to Note

The `-input=false` option indicates that Terraform should not attempt to prompt for input, and instead expect all necessary values to be provided by the configuration file or through CLI options.

The `-input=false` is supported in wide variety of Terraform CLI commands.

## Example 2 - Input=False

This example shows that a remote backend of S3 is used.

The backend configuration has a “key” missing that specifies path to state file.

```
terraform {  
  backend "s3" {  
    bucket = "demo-user-sample-bucket"  
    region = "ap-southeast-1"  
  }  
}
```



```
C:\kplabs-terraform>terraform init  
Initializing the backend...  
key  
  The path to the state file inside the bucket  
  
  Enter a value:
```

# **Importance of No-Color Option**

# Setting the Base

Various Terraform CLI commands have `-no-color` flag available.

```
C:\Users\zealv\kplabs-terraform>terraform plan
```

```
Terraform used the selected providers to generate the following execution plan.  
following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_security_group.allow_tls will be created  
+ resource "aws_security_group" "allow_tls" {  
    + arn                = (known after apply)  
    + description        = "Managed by Terraform"  
    + egress             = (known after apply)  
    + id                = (known after apply)  
    + ingress            = (known after apply)  
    + name               = "allow_tls"  
    + name_prefix        = (known after apply)  
    + owner_id           = (known after apply)  
    + revoke_rules_on_delete = false  
    + tags_all           = (known after apply)  
    + vpc_id              = (known after apply)  
}
```

Color Enabled

```
C:\Users\zealv\kplabs-terraform>terraform plan -no-color
```

```
Terraform used the selected providers to generate the following execution plan.  
following symbols:
```

```
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_security_group.allow_tls will be created  
+ resource "aws_security_group" "allow_tls" {  
    + arn                = (known after apply)  
    + description        = "Managed by Terraform"  
    + egress             = (known after apply)  
    + id                = (known after apply)  
    + ingress            = (known after apply)  
    + name               = "allow_tls"  
    + name_prefix        = (known after apply)  
    + owner_id           = (known after apply)  
    + revoke_rules_on_delete = false  
    + tags_all           = (known after apply)  
    + vpc_id              = (known after apply)  
}
```

No Color Flag Set

# Basics of No Color Flag

Disables terminal formatting sequences in the output.

Use this if you are running Terraform in a context where its output will be rendered by a system that cannot interpret terminal formatting.

# Use-Case 1: Color is Enabled

terraform plan > color.plan

CLI Command



```
color.plan ✘
```

```
color.plan
```

```
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
```

```
ESC[32m+ESC[0m createESC[0m
```

```
Terraform will perform the following actions:
```

```
ESC[1m # aws_security_group.allow_tlsESC[0m will be created
ESC[0m   ESC[32m+ESC[0mESC[0m resource "aws_security_group" "allow_tls" {
    ESC[32m+ESC[0mESC[0m arn                               = (known after apply)
    ESC[32m+ESC[0mESC[0m description                = "Managed by Terraform"
    ESC[32m+ESC[0mESC[0m egress                  = (known after apply)
    ESC[32m+ESC[0mESC[0m id                     = (known after apply)
    ESC[32m+ESC[0mESC[0m ingress                 = (known after apply)
    ESC[32m+ESC[0mESC[0m name                   = "allow_tls"
    ESC[32m+ESC[0mESC[0m name_prefix             = (known after apply)
```

# Use-Case 2: Color is Disabled

```
terraformer plan -no-color >  
nocolor.plan
```

CLI Command



```
nocolor.plan X  
nocolor.plan  
  
Terraform used the selected providers to generate the following execution  
plan. Resource actions are indicated with the following symbols:  
+ create  
  
Terraform will perform the following actions:  
  
# aws_security_group.allow_tls will be created  
+ resource "aws_security_group" "allow_tls" {  
    + arn                      = (known after apply)  
    + description              = "Managed by Terraform"  
    + egress                   = (known after apply)  
    + id                      = (known after apply)  
    + ingress                  = (known after apply)  
    + name                     = "allow_tls"  
    + name_prefix              = (known after apply)  
    + owner_id                 = (known after apply)
```

# **Plugin Caching in Terraform**

# Setting the Base

By default, Terraform downloads these provider plugins every time you run `terraform init`.

This can lead to **time consumption** and increased bandwidth usage.

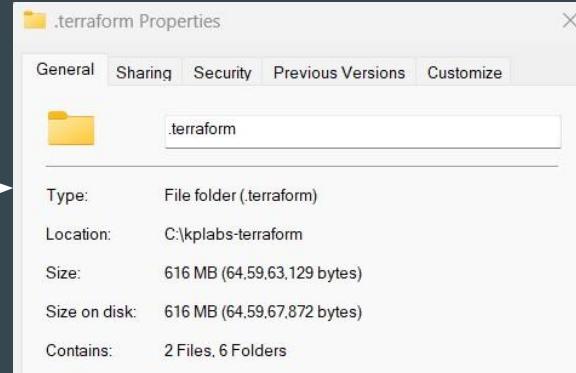
```
C:\kplabs-terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.76.0...
- Installed hashicorp/aws v5.76.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

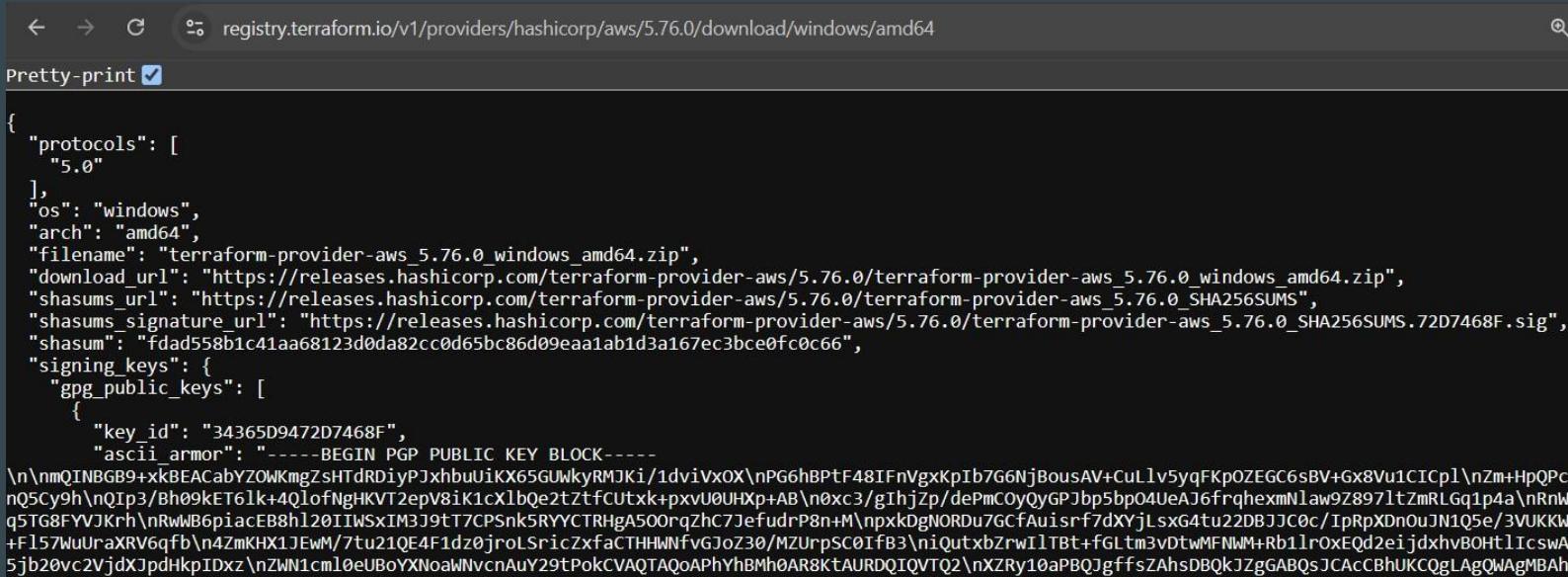
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```



# Role of Compression in Plugin Downloads

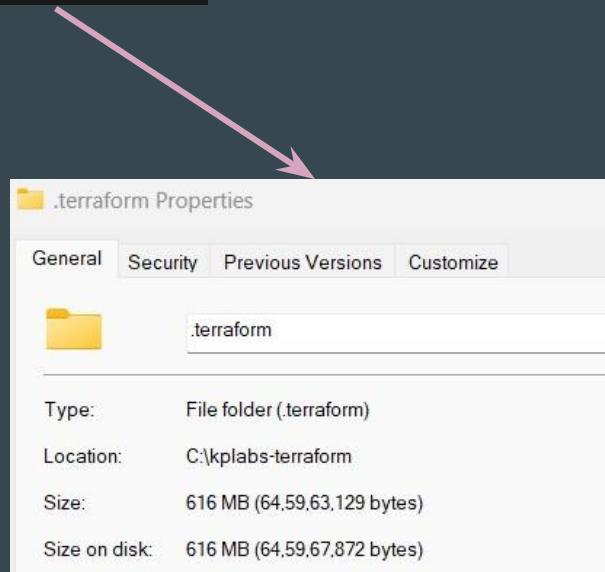
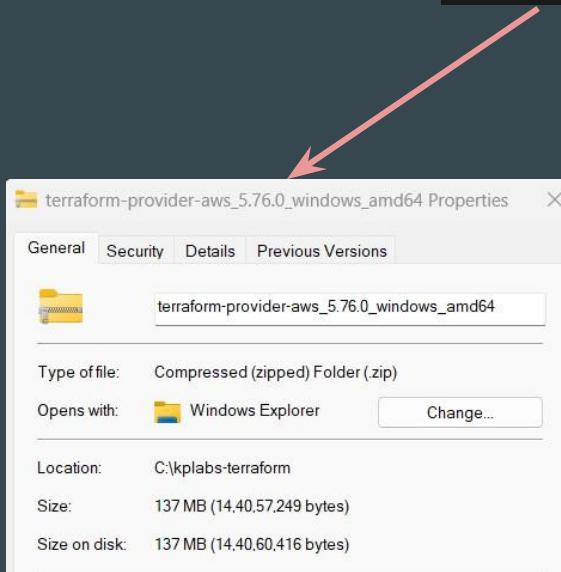
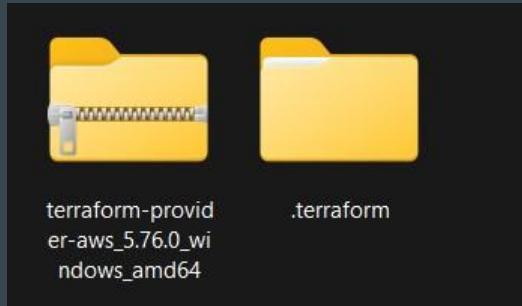
The provider plugins are downloaded in a compressed format to reduce the size.



A screenshot of a web browser window displaying the URL `registry.terraform.io/v1/providers/hashicorp/aws/5.76.0/download/windows/amd64`. The browser interface includes back and forward buttons, a search bar, and a pretty-print checkbox. The main content area shows a JSON object representing the plugin's metadata. The JSON object includes fields for protocols (5.0), operating system (windows), architecture (amd64), filename ('terraform-provider-aws\_5.76.0\_windows\_amd64.zip'), download URL ('https://releases.hashicorp.com/terraform-provider-aws/5.76.0/terraform-provider-aws\_5.76.0\_windows\_amd64.zip'), shasums URL ('https://releases.hashicorp.com/terraform-provider-aws/5.76.0/terraform-provider-aws\_5.76.0\_SHA256SUMS'), shasums signature URL ('https://releases.hashicorp.com/terraform-provider-aws/5.76.0/terraform-provider-aws\_5.76.0\_SHA256SUMS.72D7468F.sig'), shasum ('fdad558b1c41aa68123d0da82cc0d65bc86d09eaa1ab1d3a167ec3bce0fc0c66'), and signing keys. The signing keys section contains a GPG public key block, which is partially visible at the bottom of the JSON object.

```
{  
  "protocols": [  
    "5.0"  
  ],  
  "os": "windows",  
  "arch": "amd64",  
  "filename": "terraform-provider-aws_5.76.0_windows_amd64.zip",  
  "download_url": "https://releases.hashicorp.com/terraform-provider-aws/5.76.0/terraform-provider-aws_5.76.0_windows_amd64.zip",  
  "shasums_url": "https://releases.hashicorp.com/terraform-provider-aws/5.76.0/terraform-provider-aws_5.76.0_SHA256SUMS",  
  "shasums_signature_url": "https://releases.hashicorp.com/terraform-provider-aws/5.76.0/terraform-provider-aws_5.76.0_SHA256SUMS.72D7468F.sig",  
  "shasum": "fdad558b1c41aa68123d0da82cc0d65bc86d09eaa1ab1d3a167ec3bce0fc0c66",  
  "signing_keys": {  
    "gpg_public_keys": [  
      {  
        "key_id": "34365D9472D7468F",  
        "ascii_armor": "-----BEGIN PGP PUBLIC KEY BLOCK-----\n\nQmQINBGB9+xkBECAbYZOWKmgZsHTdRDiYPJxbuUiKX65GUWkyRMJKi/1dvivxOX\\nPG6hBPtF48IFnVgxKpIb7G6NjBousAV+CuL1v5yqFKpOZEGC6sBV+Gx8Vu1CICpl\\nZm+HpQPCInQ5Cyh\\nQ1p3/Bh09kET6lk+4QlofFnghKVT2epV8iK1cxlbqe2tzTfcUtxk+pxvU0UhXp+AB\\n0xc3/gIhjZp/dePmCoyGPJbp5bp04UeAJ6frqhexmNlaw9Z897ltZmRLGq1p4a\\nRnwLq5TG8FYVJKrh\\nRwB6piacEB8h120IIWsxIM3J9t7CPsnk5RYYCTRhgA500rqzhC7jerudrP8n+M\\npxkDgnORDu7GcfAuisrf7dXYjLsxG4tu22DBJJc0c/IpRpxDnouJN1Q5e/3VUKK\\+fL57wuUraxRV6fb\\n4ZmKHX1jEwM/7tu21QE4F1dz0jroLsricZxfafCThHWNfvGJoZ30/MZUrpsc0IfB3\\niqutxbZrwIlTBt+fGLtm3vDtWMFNWM+Rb1lrOxEd2eijsdxhvB0HtlIcsAR5jb20vc2VjdXJpdHkpIDxz\\nZWn1cmlo\\nUBoYXNoa\\nvncauY29tPokCVAQTAQoAPhYhBMh0AR8KTAURDQIQVTO2\\nXZRy10aPBQJgffsZAhsDBQkJZgGABQsJCaccBhUKCQgLAGQWAgnMBAh4-----\n-----END PGP PUBLIC KEY BLOCK-----"}  
    ]  
  }  
}
```

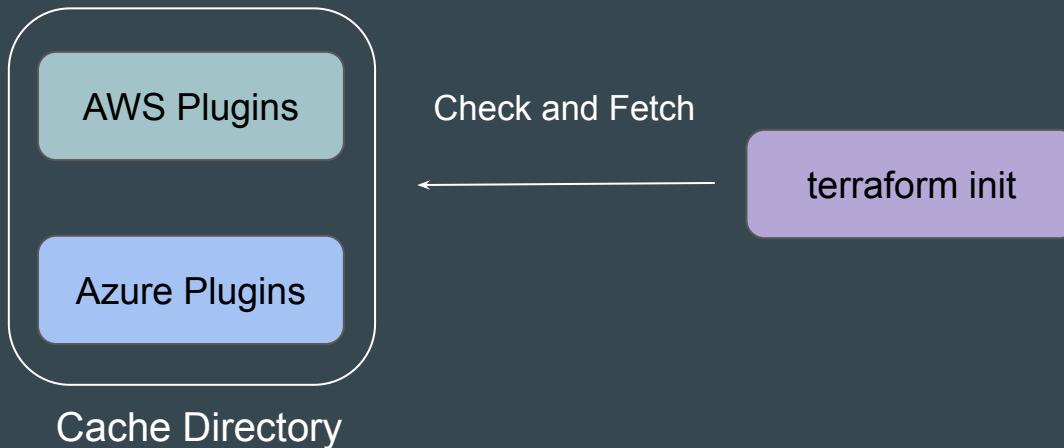
# Size Comparison - Compressed vs Final Install



# Basics of Plugin Caching

To overcome the issues, Terraform offers **plugin caching**, which allows Terraform to store provider plugin binaries in a local cache.

You can store provider plugins in the cache directory and Terraform will first check the cache before attempting to download it again.

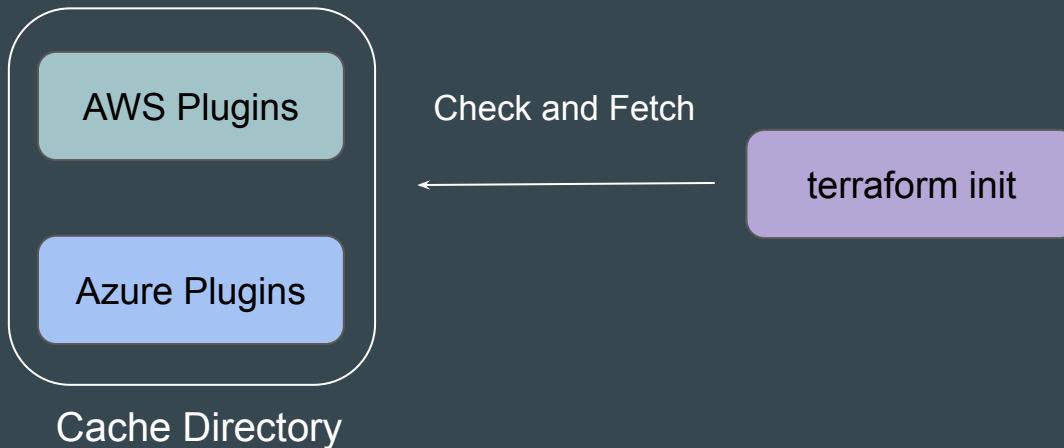


# **Plugin Caching in Terraform - Practical**

# Basics of Plugin Caching

To overcome the issues, Terraform offers **plugin caching**, which allows Terraform to store provider binaries in a local cache.

You can store provider plugins in the cache directory and Terraform will first check the cache before attempting to download it again.



# Setting Provider Plugin Cache Option

To enable the plugin cache, use the `plugin_cache_dir` setting within the Terraform RC file or set `TF_PLUGIN_CACHE_DIR` environment variable.

```
root@demo:~# cat ~/.terraformrc
plugin_cache_dir    = "$HOME/tf-plugin-cache/"
```

## Point to Note - File Name and Location

On Windows, the file must be named `terraform.rc` and placed in the relevant user's `%APPDATA%` directory.

On all other systems, the file must be named `.terraformrc` and placed directly in the home directory of the relevant user.

## Important Point to Note - 1

Currently Terraform will use an entry from the global plugin cache only if it matches a checksum already recorded in the dependency lock file.

You can make that permanent by putting the lock file in your version control system as terraform init mentions once it has created the file for the first time.

## Important Point to Note - 2

For those teams that don't preserve the dependency lock file, Terraform allows an additional CLI setting which tells Terraform to always treat a package in the cache directory as valid even if there isn't already an entry in the dependency lock file to confirm it:

Alternatively, you can set the environment variable

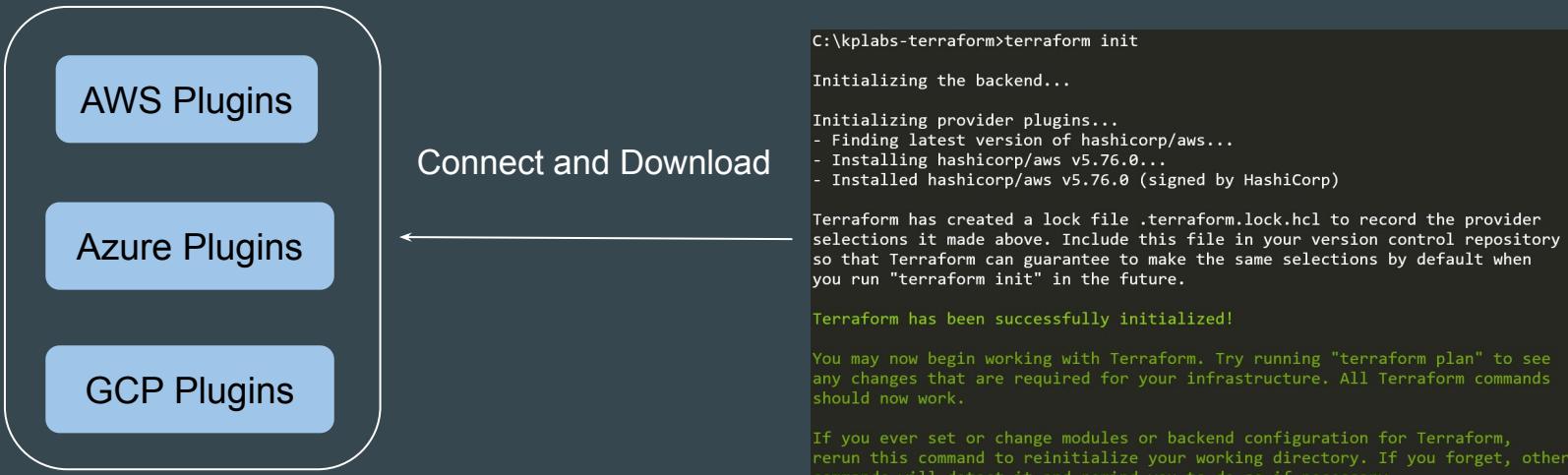
`TF_PLUGIN_CACHE_MAY_BREAK_DEPENDENCY_LOCK_FILE` to any value

```
plugin_cache_may_break_dependency_lock_file = true
```

# **File System Mirror in Terraform**

# Setting the Base

By default, when we run `terraform init`, Terraform downloads the provider from the HashiCorp Registry



[registry.terraform.io](https://registry.terraform.io)

# What if No Internet?

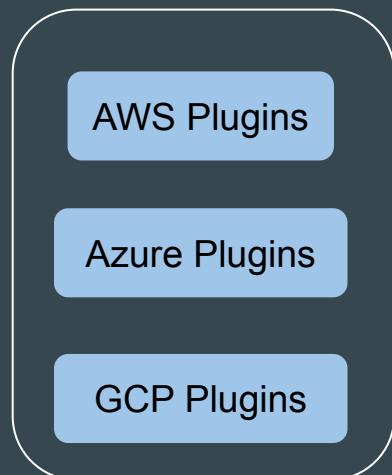
If internet connectivity is not present, Terraform will give error during init phase as it is not able to connect to `registry.terraform.io` to download provider plugins.

```
C:\kplabs-terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/random...
Error: Failed to query available provider packages

Could not retrieve the list of available versions for provider hashicorp/random: could not connect to
registry.terraform.io: failed to request discovery document: Get
"https://registry.terraform.io/.well-known/terraform.json": dial tcp: lookup registry.terraform.io: no such host
```

# Downloading the Plugins Locally

In environments where there are connectivity restrictions, you can decide to download the provider plugins to local system.



[registry.terraform.io](https://registry.terraform.io)

# Use Local Provider Plugins for Project

You can instruct Terraform to fetch the plugins from specific path of local file system to your project folder.



Local File System

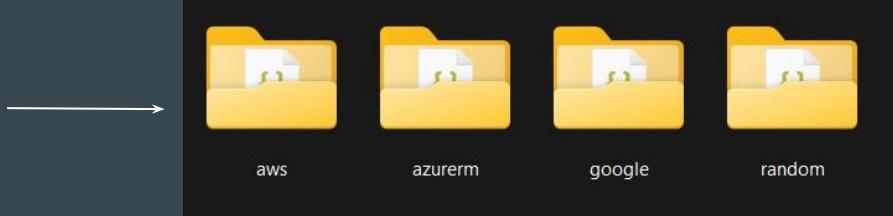


```
C:\kplabs-terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "3.0.0"...
- Finding latest version of hashicorp/aws...
- Finding latest version of hashicorp/google...
- Installing hashicorp/azurerm v3.0.0...
- Installed hashicorp/azurerm v3.0.0 (unauthenticated)
- Installing hashicorp/aws v5.76.0...
- Installed hashicorp/aws v5.76.0 (unauthenticated)
- Installing hashicorp/google v6.11.1...
- Installed hashicorp/google v6.11.1 (unauthenticated)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
```

# Step 1 - Download Required Providers

The `terraform providers mirror` command downloads the providers required for the current configuration and copies them into a directory in the local filesystem.

```
C:\kplabs-terraform>terraform providers mirror C:/plugin_cache
- Mirroring hashicorp/azurerm...
  - Selected v3.0.0 to match dependency lock file
  - Downloading package for windows_amd64...
  - Package authenticated: signed by HashiCorp
- Mirroring hashicorp/aws...
  - Selected v5.76.0 to match dependency lock file
  - Downloading package for windows_amd64...
  - Package authenticated: signed by HashiCorp
- Mirroring hashicorp/google...
  - Selected v6.11.1 to match dependency lock file
  - Downloading package for windows_amd64...
  - Package authenticated: signed by HashiCorp
```



## Step 2 - Add Provider Installation Block

A `provider_installation` block in the CLI configuration allows overriding Terraform's default installation behaviors, so you can force Terraform to use a local mirror for some or all of the providers you intend to use.

```
provider_installation {  
  filesystem_mirror {  
    path      = "C:/plugin_cache"  
    include   = ["registry.terraform.io/hashicorp/*"]  
  }  
}
```

## Optional Step - Use Direct Installation Method Type

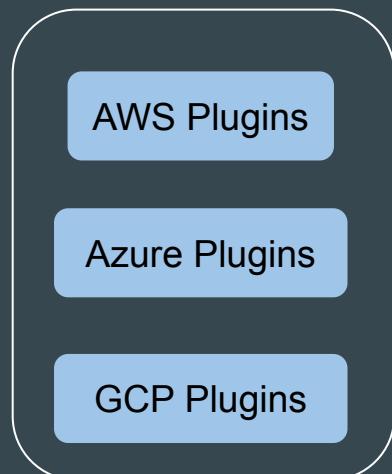
Specifying direct installation type request information about the provider directly from its origin registry and download over the network from the location that registry indicates.

```
provider_installation {
  filesystem_mirror {
    path      = "C:/plugin_cache"
    include   = ["registry.terraform.io/hashicorp/*"]
  }
  direct {
    include   = ["registry.terraform.io/hashicorp/local"]
  }
}
```

# **File System Mirror in Terraform - Practical**

# Downloading the Plugins Locally

In environments where there are connectivity restrictions, you can decide to download the provider plugins to local system.



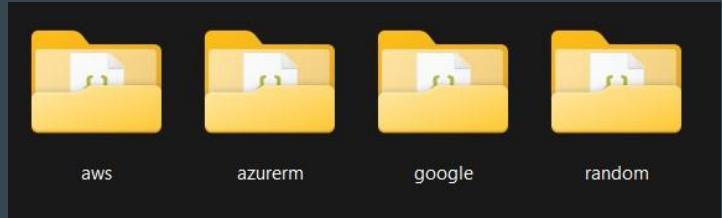
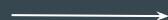
[registry.terraform.io](https://registry.terraform.io)

# Step 1 - Download Required Providers

The `terraform providers mirror` command downloads the providers required for the current configuration and copies them into a directory in the local filesystem.

```
C:\kplabs-terraform>terraform providers mirror C:/plugin_cache
```

```
- Mirroring hashicorp/azurerm...
  - Selected v3.0.0 to match dependency lock file
  - Downloading package for windows_amd64...
  - Package authenticated: signed by HashiCorp
- Mirroring hashicorp/aws...
  - Selected v5.76.0 to match dependency lock file
  - Downloading package for windows_amd64...
  - Package authenticated: signed by HashiCorp
- Mirroring hashicorp/google...
  - Selected v6.11.1 to match dependency lock file
  - Downloading package for windows_amd64...
  - Package authenticated: signed by HashiCorp
```



## Step 2 - Add Provider Installation Block

A `provider_installation` block in the CLI configuration allows overriding Terraform's default installation behaviors, so you can force Terraform to use a local mirror for some or all of the providers you intend to use.

```
provider_installation {
  filesystem_mirror {
    path      = "C:/plugin_cache"
    include   = ["registry.terraform.io/hashicorp/*"]
  }
}
```

## Optional Step - Use Direct Installation Method Type

Specifying direct installation type request information about the provider directly from its origin registry and download over the network from the location that registry indicates.

```
provider_installation {
  filesystem_mirror {
    path      = "C:/plugin_cache"
    include   = ["registry.terraform.io/hashicorp/*"]
  }
  direct {
    include = ["registry.terraform.io/hashicorp/local"]
  }
}
```

# Relax and Have a Meme Before Proceeding

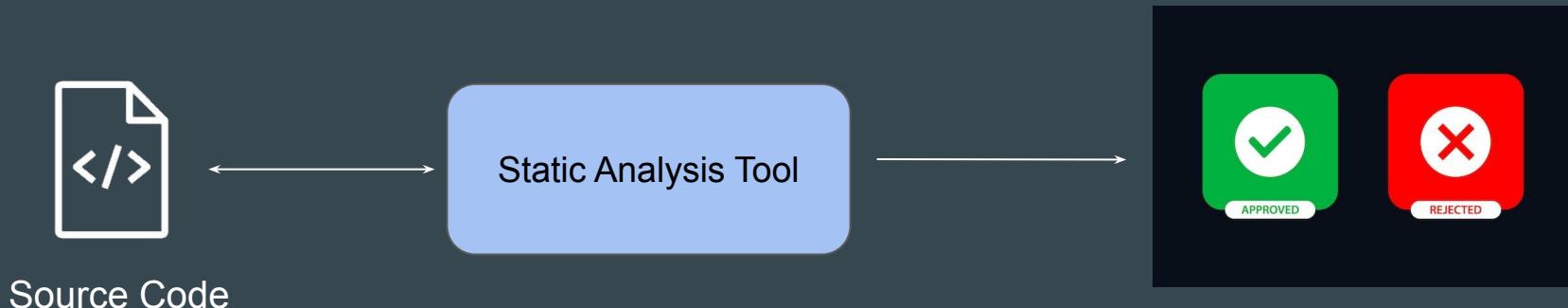
When you write 100 lines  
of code and it compiles on  
your first attempt



# **Static Analysis Tools for Terraform**

# Basics of Static Analysis

Static Code analysis is a software verification activity that analyzes source code for quality, reliability, and security without executing the code.



# Introducing Checkov

Checkov is a **static code analysis tool for scanning infrastructure as code (IaC) files** for misconfigurations that may lead to security or compliance problems.

```
secrets scan results:

Passed checks: 0, Failed checks: 2, Skipped checks: 0

Check: CKV_SECRET_2: "AWS Access Key"
    FAILED for resource: 25910f981e85ca04baf359199dd0bd4a3ae738b6
    File: /test.tf:4-5
    Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/secrets-2

        4 |     access_key = "AKIAI*****"

Check: CKV_SECRET_6: "Base64 High Entropy String"
    FAILED for resource: d70eab08607a4d05faa2d0d6647206599e9abc65
    File: /test.tf:5-6
    Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/secrets-6

        5 |     secret_key = "wJalrX*****"
```

# Benefits of Static Analysis Tools

1. Identifying issues with the code in early stages of development.
2. Enforces coding standards.
3. Provides instant feedback for your code.

# Benefits of Checkov as Static Analysis Tool for Terraform

Checkov scans your Terraform code for misconfigurations that could lead to security breaches. It has over **750 policies to check for common issues** like open security groups, IAM users with excessive permissions, and hardcoded secrets.

By catching errors early in the development process, Checkov helps prevent security vulnerabilities from being deployed to production.

# **Saving Terraform Plan to File**

# Setting the Base

Terraform allows saving a plan to a file.

```
terraform plan -out ec2.plan
```

```
Saved the plan to: ec2.plan
```

```
To perform exactly these actions, run the following command to apply:  
terraform apply "ec2.plan"
```

# Apply from Plan File

You can run the terraform apply by referencing the plan file.

This ensures the infrastructure state remains exactly as shown in the plan to ensure consistency.

```
C:\Users\zealv\kplabs-terraform>terraform apply infra.plan
aws_security_group.allow_tls: Creating...
aws_security_group.allow_tls: Creation complete after 4s [id=sg-02ed88b1d80a484a6]
aws_vpc_security_group_ingress_rule.app_port: Creating...
aws_vpc_security_group_ingress_rule.ssh_port: Creating...
aws_vpc_security_group_ingress_rule.ftp_port: Creating...
aws_vpc_security_group_ingress_rule.app_port: Creation complete after 1s [id=sgr-0b8f10164824c9027]
aws_vpc_security_group_ingress_rule.ssh_port: Creation complete after 1s [id=sgr-042688b669ba3d824]
aws_vpc_security_group_ingress_rule.ftp_port: Creation complete after 2s [id=sgr-0962b3406710b00ba]

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

# Exploring Terraform Plan File

The saved Terraform plan file will be a binary file.

You can use the `terraform show` command to read the contents in detail.

```
{  
    "format_version": "1.2",  
    "terraform_version": "1.9.1",  
    "variables": {  
        "app_port": {  
            "value": "8080"  
        },  
        "ftp_port": {  
            "value": "21"  
        },  
        "ssh_port": {  
            "value": "22"  
        },  
        "vpn_ip": {  
            "value": "200.20.30.50/32"  
        }  
    },  
    "planned_values": {  
        "root_module": {  
            "resources": [  
                {  
                    "address": "aws_security_group.allow_tls",  
                    "mode": "managed",  
                    "type": "aws_security_group",  
                    "name": "allow_tls",  
                    "provider_name": "registry.terraform.io/hashicorp/aws",  
                    "schema_version": 1,  
                    "values": {  
                        "description": "Managed from Terraform",  
                        "name": "terraform-firewall",  
                        "values": {}  
                    }  
                }  
            ]  
        }  
    }  
}
```

# Use-Cases of Saving Plan to a File

Many organizations require documented proof of planned changes before implementation.

These changes will further be reviewed and approved.

Running apply from plan ensures consistent desired outcome.

# **Code Scanning vs Plan Scanning**

# Revising Basics of Code Scans

Code Scan feature will analyze your Terraform code and provide with you the scan results.

```
provider "aws" {
  region      = "us-east-1"
  access_key  = "AKIAIOSFODNN7EXAMPLE"
  secret_key  = "wJalrXUtnFEMI/K7MDENG/bPxRfICYEXAMPLEKEY"
}

resource "aws_security_group" "this" {
  name        = "demo-firewall"
}
```



```
secrets scan results:

Passed checks: 0, Failed checks: 2, Skipped checks: 0

Check: CKV_SECRET_2: "AWS Access Key"
      FAILED for resource: 25910f981e85ca04baf359199dd0bd4a3ae738b6
      File: /test.tf:4-5
      Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/secrets-2
          4 |   access_key = "AKIAI*****"

Check: CKV_SECRET_6: "Base64 High Entropy String"
      FAILED for resource: d70eab08607a4d05faa2d0d6647206599e9abc65
      File: /test.tf:5-6
      Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/secrets-6
          5 |   secret_key = "wJalrX*****"
```

# Setting the Base

Tools like checkov supports both code-level scans and plan-level scans.



# Example Use-Case - Firewall Rule

Checkov can detect if important ports like 22 have 0.0.0.0/0 allowed.

```
resource "aws_security_group" "this" {
  name      = "demo-firewall"
}

resource "aws_vpc_security_group_ingress_rule" "example" {
  security_group_id = aws_security_group.this.id

  cidr_ipv4    = "0.0.0.0/0"
  from_port    = 22
  ip_protocol = "tcp"
  to_port      = 22
}
```



```
Check: CKV_AWS_24: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 22"
      FAILED for resource: aws_vpc_security_group_ingress_rule.example
      File: /mysg.tf:18-25
      Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies,king-1-port-security
```

```
18 | resource "aws_vpc_security_group_ingress_rule" "example" {
19 |   security_group_id = aws_security_group.this.id
20 |
21 |   cidr_ipv4    = "0.0.0.0/0"
22 |   from_port    = 22
23 |   ip_protocol = "tcp"
24 |   to_port      = 22
25 | }
```

# Bypassing the Check

There are multiple small tricks that can be used to avoid detection of code level scans by static analysis tools.

```
variable "vpn_ip" {}

resource "aws_vpc_security_group_ingress_rule" "example" {
    security_group_id = aws_security_group.this.id

    cidr_ipv4      = var.vpn_ip
    from_port       = 22
    ip_protocol     = "tcp"
    to_port         = 22
}
```

→

```
Check: CKV_AWS_24: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 22"
    PASSED for resource: aws_security_group.this
    File: /mysg.tf:1-3
    Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-king-1-port-security
Check: CKV_AWS_24: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 22"
    PASSED for resource: aws_vpc_security_group_ingress_rule.example
    File: /mysg.tf:8-15
    Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-king-1-port-security
```

# Plan Level Scans

Tools like checkov can also evaluate rules against terraform plan expressed in a json file.

The plan file will probably have final value related to variable.

```
Check: CKV_AWS_24: "Ensure no security groups allow ingress from 0.0.0.0:0 to port 22"
      FAILED for resource: aws_vpc_security_group_ingress_rule.example
      File: /tfplan.json:38-49
      Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-p
king-1-port-security

39 |         "values": {
40 |             "cidr_ipv4": "0.0.0.0/0",
41 |             "cidr_ipv6": null,
42 |             "description": null,
43 |             "from_port": 22,
44 |             "ip_protocol": "tcp",
45 |             "prefix_list_id": null,
46 |             "referenced_security_group_id": null,
47 |             "tags": null,
48 |             "tags_all": {},
49 |             "to_port": 22
```

# Custom Policies in Checkov

Plan can provide us with wide variety of information like, if resource will be created, updated or destroyed.

You can write custom set of rules based on this information.

```
"resource_changes": [
  {
    "address": "aws_security_group.this",
    "mode": "managed",
    "type": "aws_security_group",
    "name": "this",
    "provider_name": "registry.terraform.io/hashicorp/aws",
    "change": {
      "actions": [
        "create"
      ],
    }
  }
]
```

Action of Create

```
"resource_drift": [
  {
    "address": "aws_security_group.this",
    "mode": "managed",
    "type": "aws_security_group",
    "name": "this",
    "provider_name": "registry.terraform.io/hashicorp/aws",
    "change": {
      "actions": [
        "update"
      ],
    }
  }
]
```

Action of Update

**TF\_IN\_AUTOMATION**

# Setting the Base

By default, some Terraform commands conclude by presenting a description of a possible next step to the user, often including a specific command to run next.

```
root@checkov:~# terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.66.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

# Understanding the Importance

In environments like CI/CD, the entire process is automated and hence these suggestions are less helpful.

Removing these suggestions will make output less lengthy.

Set the TF\_IN\_AUTOMATION to non-empty value to achieve this.

```
root@checkov:~# terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.66.0

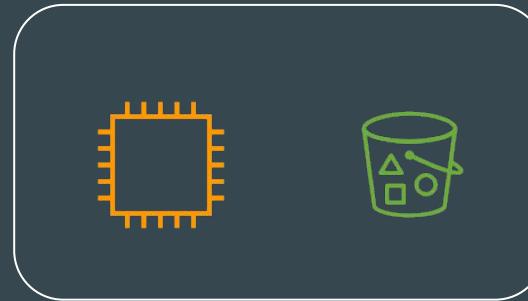
Terraform has been successfully initialized!
```

# **Terraform Import**

# Typical Challenge

It can happen that all the resources in an organization are created manually.

Organization now wants to start using Terraform and manage these resources via Terraform.

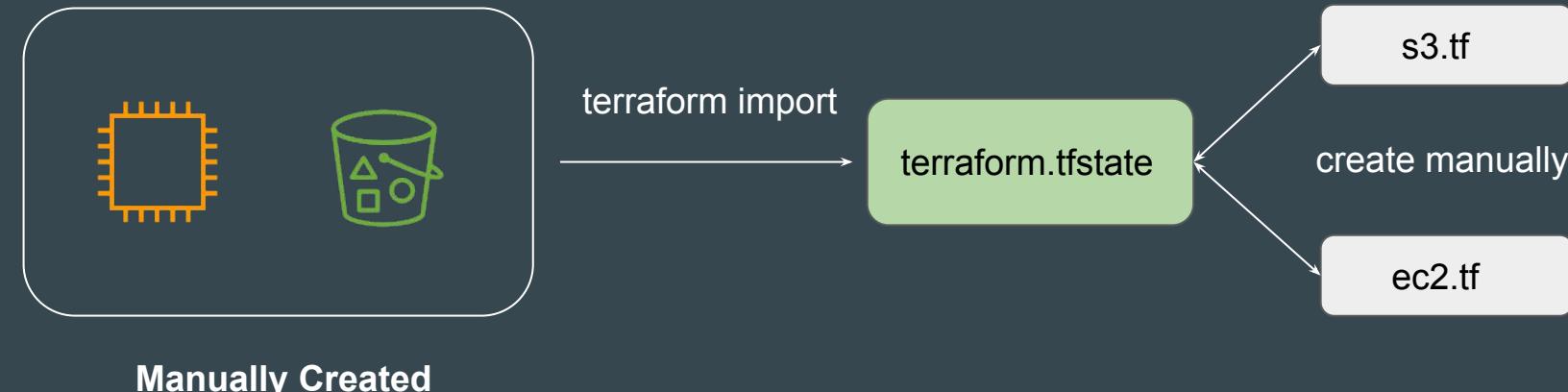


Manually Created

# Earlier Approach

In the older approach, Terraform import would create the state file associated with the resources running in your environment.

Users still had to write the tf files from scratch.



# Newer Approach

In the newer approach, **terraform import** can automatically create the terraform configuration files for the resources you want to import.



## Point to Note

Terraform 1.5 introduces automatic code generation for imported resources.

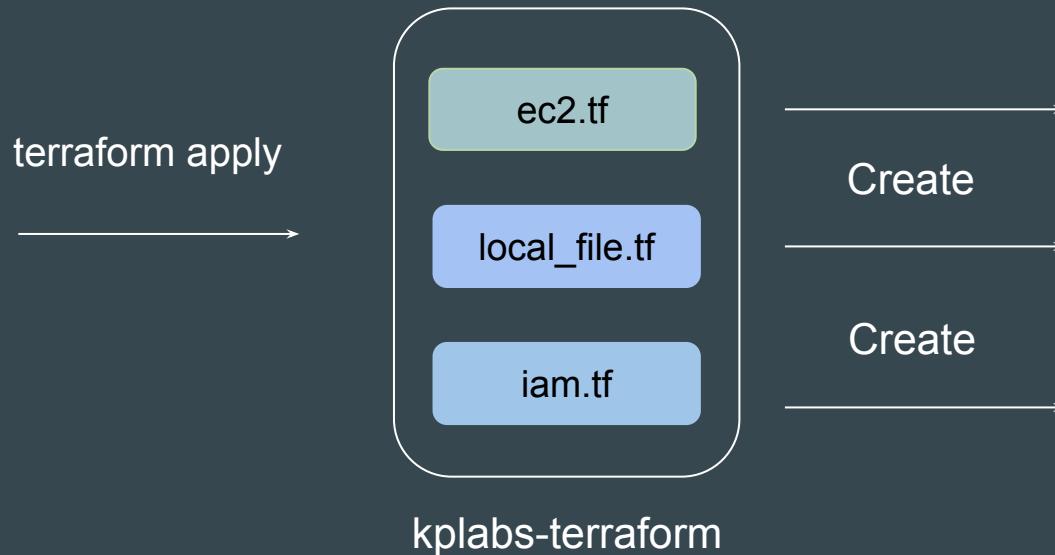
This dramatically reduces the amount of time you need to spend writing code to match the imported

This feature is not available in the older version of Terraform.

# **Resource Targeting**

# Setting the Base

In a typical Terraform workflow, you apply the entire plan at once. This is also the default behaviour.



# Understanding Resource Targeting

Resource targeting in Terraform allows you to **apply changes to a specific subset of resources** rather than applying changes to your entire infrastructure.

terrafrom plan <local\_file>



```
resource "aws_iam_user" "dev" {
    name = "kplabs-user-01"
}

resource "aws_security_group" "prod" {
    name      = "terraform-firewalls"
}

resource "local_file" "foo" {
    content  = "foo!"
    filename = "${path.module}/foo.txt"
}
```

# Using the -target flag

You can use Terraform's **-target** option to target specific resources, modules as part of operation.

```
resource "aws_iam_user" "dev" {
    name = "kplabs-user-01"
}

resource "aws_security_group" "prod" {
    name          = "terraform-firewalls"
}

resource "local_file" "foo" {
    content  = "foo!"
    filename = "${path.module}/foo.txt"
}
```

terraform plan -target local\_file.foo



## Use-Cases

A project has 10 resources. Multiple team members are working on each resource's Terraform code to release some updates.

When you run terraform plan, you see Terraform trying to make changes to all 10 resources.

There is a critical issue and you need to add a port 80 in security group that is managed via terraform without having to update other 9 resources.

## Points to Note

Occasionally you may want to apply only part of a plan, such as when Terraform's state has become out of sync with your resources due to a network failure, a problem with the upstream cloud platform, or a bug in Terraform or its providers.

To support this, Terraform lets you target specific resources when you plan, apply, or destroy your infrastructure.

Targeting individual resources can be useful for troubleshooting errors, **but should not be part of your normal workflow.**

# **Resource - Random Integer**

# Setting the Base

The resource `random_integer` generates random values from a given range, described by the `min` and `max` attributes of a given resource.

```
resource "random_integer" "this" {  
    min = 1  
    max = 50000  
}
```



Outputs:

```
random_integer = 48489
```

# Example Use-Case - S3

In many cloud environments, resource names must be globally unique.

Using `random_integer` can help to append a random number to resource names to avoid conflicts especially in automated environments.

```
resource "random_integer" "this" {
  min = 1024
  max = 50000
}

resource "aws_s3_bucket" "bucket" {
  bucket = "kplabs-bucket-${random_integer.this.result}"
}
```



# **Resource Targeting - Handling Dependency**

# Dealing with Dependencies

Resource targeting updates resources that the target depends on, but not resources that depend on it.

Resource 1

```
resource "random_integer" "this" {
    min = 1000
    max = 50000
}
```

```
resource "aws_s3_bucket" "example" {
    bucket = "kplabs-testing-bucket-${random_integer.this.result}"
```

Resource 2  
dependent on  
Resource 1

# Example 1 - Updating Resource 1

Random Integer resource depends on NO other resource.

```
resource "random_integer" "this" {  
    min = 1000  
    max = 50000  
}
```

Modification Made

```
resource "random_integer" "this" {  
    min = 1005  
    max = 50000  
}
```



```
C:\kplabs-terraform>terraform plan -target random_integer.this  
random_integer.this: Refreshing state... [id=42086]  
  
Terraform used the selected providers to generate the following  
following symbols:  
-/+ destroy and then create replacement  
  
Terraform will perform the following actions:  
  
    # random_integer.this must be replaced  
-/+ resource "random_integer" "this" {  
        ~ id      = "42086" -> (known after apply)  
        ~ min    = 1000 -> 1005 # forces replacement  
        ~ result = 42086 -> (known after apply)  
        # (1 unchanged attribute hidden)  
    }  
  
Plan: 1 to add, 0 to change, 1 to destroy.
```

# Example 2 - Updating Resource 2

S3 bucket depends on Random Integer.

If random integer resource is modified and you target S3, it will take into account random\_integer as well.

```
C:\kplabs-terraform>terraform plan -target aws_s3_bucket.example
random_integer.this: Refreshing state... [id=42086]
aws_s3_bucket.example: Refreshing state... [id=kplabs-testing-bucket-42086]

Terraform used the selected providers to generate the following execution plan.
The plan is shown below.

+/- destroy and then create replacement

Terraform will perform the following actions:

  # aws_s3_bucket.example must be replaced
-/+ resource "aws_s3_bucket" "example" {
    + acceleration_status      = (known after apply)
    + acl                      = (known after apply)
    ~ arn                      = "arn:aws:s3:::kplabs-testing-bucket-42086"
    ~ bucket                   = "kplabs-testing-bucket-42086" -> (known after apply)
    ~ bucket_domain_name       = "kplabs-testing-bucket-42086.s3.amazonaws.com" -> (known after apply)
    + bucket_prefix             = (known after apply)
    ~ bucketRegionalDomainName = "kplabs-testing-bucket-42086.s3.us-east-2" -> (known after apply)
    ~ hostedZoneId              = "Z3AQBSTGFYJSTF" -> (known after apply)
```

# **Escape Sequence in Quoted String**

# Understanding the Basic

In Terraform, a quoted string is a sequence of characters enclosed within double quotes (").

```
"hello"
```

# Understanding the Challenge

Sometimes, you need to include special characters or symbols in these strings that would otherwise be interpreted differently.

Escape sequences are used to handle such cases.

≡ friends.txt

The name of best friends are "Alice", and "Bob".

# Testing with Local File Resource

You might want to include quotes as shown in the friends.txt file

Without escaping, Terraform would think the string ends at the first quote inside the word

```
resource "local_file" "foo" {
    content  = "The name of best friends are "Alice", and "Bob""
    filename = "friends.txt"
}
```

```
C:\kplabs-terraform>terraform validate
Error: Missing newline after argument

on escape-seq.tf line 3, in resource "local_file" "foo":
  3:   content  = "The name of best friends are "Alice", and "Bob""

An argument definition must end with a newline.
```

# Escape Sequence in Terraform

Here's a list of common escape sequences available in Terraform:

Sequence	Replacement
\n	Newline
\r	Carriage Return
\t	Tab
\"	Literal quote (without terminating the string)
\\\	Literal backslash
\uNNNN	Unicode character from the basic multilingual plane (NNNN is four hex digits)
\UNNNNNNNN	Unicode character from supplementary planes (NNNNNNNN is eight hex digits)

# Solution - Dealing with Multiple Quotes

\\" allows us to enter a literal quote without terminating the string

```
resource "local_file" "foo" {
  content  = "The name of best friends are \"Alice\\\", and \"Bob\\\""
  filename = "friends.txt"
}
```

# Literal Backslash

The backslash (\) is used as an escape character, so to include an actual backslash in the string, you need to escape it with another backslash (\\).

```
resource "local_file" "foo" {  
    content  = "C:\\Users\\zealv\\.aws"  
    filename = "filename.txt"  
}
```

```
resource "local_file" "foo" {  
    content  = "C:\\\\Users\\\\zealv\\\\.aws"  
    filename = "filename.txt"  
}
```

# New Line

The `\n` sequence inserts a newline, making the string appear on two lines.

```
resource "local_file" "foo" {  
  content  = "Hello,\nWelcome to Terraform!"  
  filename = "newline.txt"  
}
```

≡ newline.txt  
Hello,  
Welcome to Terraform!

# Tab Character \t

The \t sequence inserts a tab character, which is useful for aligning text.

```
resource "local_file" "tab" {  
    content  = "Item 1:\tValue 1\nItem 2:\tValue 2"  
    filename = "tab.txt"  
}
```

≡ tab.txt

Item 1: Value 1  
Item 2: Value 2

# **Heredoc Strings in Terraform**

# Heredoc Strings

In Terraform, Heredoc (short for "Here Document") is a **way of using multi-line strings** in your configuration files.

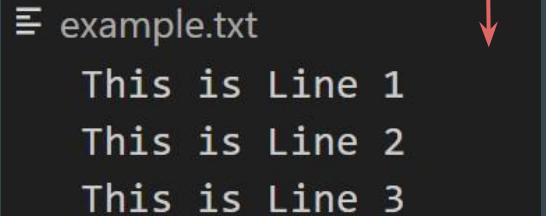
It's particularly useful when working with long blocks of text, such as scripts, configuration files, or JSON data that you need to pass into resources, outputs, or variables.

```
<<EOT
hello
world
EOT
```

## Example - Multi-Line Strings

For multi-line strings, many users uses the escape sequence of \n within the quoted string.

```
resource "local_file" "example" {  
    filename = "example.txt"  
    content = "This is Line 1\nThis is Line 2\nThis is Line 3"  
}
```



The image shows a terminal window with a dark background and light-colored text. It displays the file 'example.txt' which contains three lines of text: 'This is Line 1', 'This is Line 2', and 'This is Line 3'. A red arrow points from the word 'content' in the code above to the file icon in the terminal window.

```
≡ example.txt  
This is Line 1  
This is Line 2  
This is Line 3
```

# Better Alternative for Readability

You can use a heredoc to define **multiline content** which makes it easier to read.

```
resource "local_file" "example" {
  filename = "example.txt"
  content = "This is Line 1\nThis is Line 2\nThis is Line 3"
}
```

Older Approach

```
resource "local_file" "example" {
  filename = "example.txt"

  content = <<-EOT
  This is Line 1
  This is Line 2
  This is Line 3
  EOT
}
```

Newer Approach

# Use-Case - Writing a Script

Heredoc strings are extensively used while adding script / set of commands within the resource.

```
resource "local_file" "example" {
    filename = "myscript.sh"

    content = <<EOF
#!/bin/bash
echo "This is a script generated and managed by Terraform"
echo "Installing Nginx"
sudo yum -y install nginx
sudo systemctl start nginx
sudo systemctl enable nginx
EOF
}
```

```
$ myscript.sh
#!/bin/bash
echo "This is a script generated and managed by Terraform"
echo "Installing Nginx"
sudo yum -y install nginx
sudo systemctl start nginx
sudo systemctl enable nginx
```

# Real World Example - User data in EC2

Heredoc is often used when passing user data to AWS EC2 instances. User data is typically a shell script or cloud-init configuration that runs when the instance starts.

```
resource "aws_instance" "test" {
    ami           = "ami-06b21ccaeff8cd686"
    instance_type = "t2.micro"

    user_data = <<-EOF
        #!/bin/bash
        echo "Installing Nano Editor"
        sudo yum -y install nano
    EOF
}
```

# Real World Example 2 - IAM Policies

Heredoc strings are also used while writing IAM policies.

```
resource "aws_iam_policy" "policy" {
    name      = "demo-policy"
    description = "My test policy"

    policy = <<EOT
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
EOT
}
```

# **Heredoc Strings - Part 2**

# Components of Heredoc String

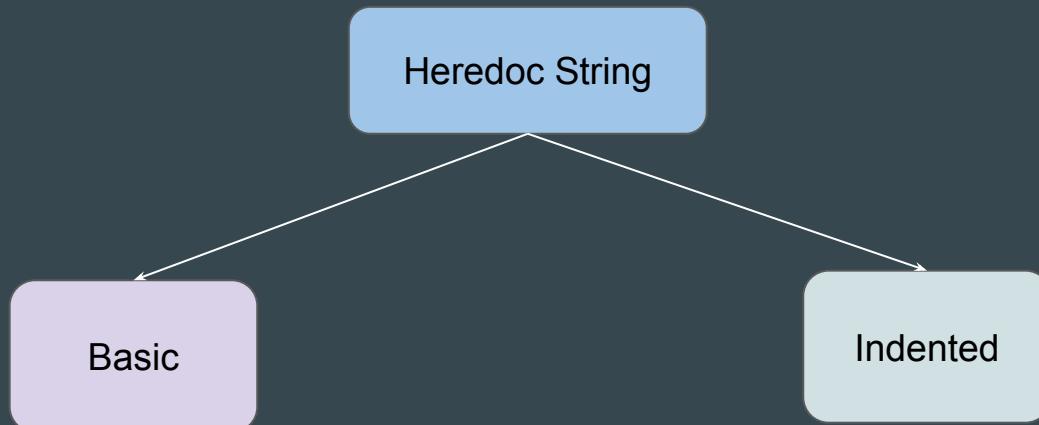
A heredoc string consists of an opening sequence consisting of:

1. A heredoc marker (<< or <<- — two less-than signs, with an optional hyphen for indented heredocs)
2. A delimiter word of your own choosing
3. A line break

```
<<EOT  
hello  
world  
EOT
```

# Type of Heredoc String

There are two type of Heredoc strings: Basic and Indented



# Basic Heredoc Strings

Heredoc marker of << is used to signify basic type.

Basic type preserves all whitespace, including leading space or indentation within the heredoc block.

```
output "basic_example" {
  value = <<EOF
    This is a basic heredoc example.
    The indentation here is preserved exactly.
    Line 1
      Line 2 (with additional space)
        Line 3 (even more indentation)
  EOF
}
```



Outputs:

```
basic_example = <<EOT
  This is a basic heredoc example.
  The indentation here is preserved exactly.
  Line 1
    Line 2 (with additional space)
      Line 3 (even more indentation)
```

# Indented Heredoc Strings

Heredoc marker of `<<-` is used to signify indented type.

Indented type **strips** leading whitespace from each line, which allows for more readable indentation in your Terraform configuration without affecting the actual content.

```
output "indented_example" {
  value = <<-EOF
    This is an indented heredoc example.
    The leading spaces at the start of each line
    are removed in the final output.
    Line 1
      Line 2 (with additional space)
        Line 3 (even more indentation)
    EOF
}
```



```
indented_example = <<EOT
  This is an indented heredoc example.
  The leading spaces at the start of each line
  are removed in the final output.
  Line 1
    Line 2 (with additional space)
      Line 3 (even more indentation)

  EOT
```

# When is Indented Heredoc Useful

You want to keep your Terraform code properly indented to align with the surrounding blocks, but you don't want that indentation to appear in the actual content of the string.

```
resource "aws_instance" "web" {
    ami           = "ami-12345678"
    instance_type = "t2.micro"

    user_data = <<-EOF
        #!/bin/bash
        echo "Starting setup..."
        sudo apt-get update -y
        sudo apt-get install -y nginx
        echo "Hello from Terraform" > /var/www/html/index.html
    EOF
}
```

# **Clarity Video - Leading Whitespace**

# Basics of Leading Whitespace

The term leading whitespace refers to any whitespace characters (spaces or tabs) that appear at the beginning of each line within the heredoc block.

```
output "indented_example" {
    value = <<-EOF
        ... This is an indented heredoc example.
        The leading spaces at the start of each line
        are removed in the final output.
        Line 1
            Line 2 (with additional space)
            Line 3 (even more indentation)
        EOF
}
```

# How Indented Heredoc Works

When using an indented heredoc (<<-EOF), Terraform looks for the shortest series of leading whitespace across all lines in the heredoc, and removes exactly that amount of whitespace from every line.

# Sample Example Code for Testing

- Line 1 starts with 4 spaces.
- Line 2 starts with 6 spaces (2 more than Line 1).
- Line 3 starts with 4 spaces (same as Line 1).

```
output "example" {
    value = <<-EOF
    |...|Line 1
        Line 2 (indented)
    Line 3
EOF
}
```

# The Final Output

Terraform determines that the minimum amount of leading whitespace is 4 spaces (from Line 1 and Line 3), so it strips exactly 4 spaces from the beginning of each line:

```
output "example" {
  value = <<-EOF
  Line 1
    Line 2 (indented)
  Line 3
EOF
}
```



Outputs:

```
example = <<EOT
Line 1
  Line 2 (indented)
Line 3
```

# Relax and Have a Meme Before Proceeding

When you're the only one who  
can pass the helicopter mission  
in GTA Vice City and your friend  
call you to pass it for him



# Terraform Functions - jsonencode and jsondecode

# Basics of jsonencode

`jsonencode` function takes a Terraform data structure (like a list, map, or object) and converts it into a JSON-encoded string.

This is essential when you need to output data in a valid JSON format for external consumption.

```
resource "local_file" "with_jsonencode" {
  filename = "./json-encode.json"
  content  = jsonencode({
    name  = "Alice"
    age   = 30
    skills = ["Terraform", "AWS", "Azure"]
  })
}
```



```
{ } json-encode.json > ...
{"age":30,"name":"Alice","skills":["Terraform","AWS","Azure"]}
```

# Converting a List to JSON

`jsonencode` function can also take lists and convert it to JSON.

```
resource "local_file" "with_jsonencode" {
  filename = "./json-encode.json"
  content  = jsonencode({
    name  = "Alice"
    age   = 30
    skills = ["Terraform", "AWS", "Azure"]
  })
}
```



```
{ } json-list.json > ...
["Terraform", "AWS", "Azure"]
```

# Basics of jsondecode

This function takes a JSON string and converts it into a native Terraform data structure (like a list, map, or object).

```
{} sample.json > ...
{
  "name": "Alice Katherine",
  "age": 30,
  "skills": ["Terraform", "AWS", "Ansible"],
  "active": true,
  "projects": [
    {
      "name": "Project X",
      "status": false
    },
    {
      "name": "Project Y",
      "completed": true
    }
  ]
}
```



```
test = {
  "active" = true
  "age" = 30
  "name" = "Alice Katherine"
  "projects" = [
    {
      "name" = "Project X"
      "status" = false
    },
    {
      "completed" = true
      "name" = "Project Y"
    }
  ]
  "skills" = [
    "Terraform",
    "AWS",
    "Ansible",
  ]
}
```

## **More Examples - jsondecode**

# Simplistic JSON Decode

This function takes a JSON string and converts it into a native Terraform data structure (like a list, map, or object).

```
{} sample.json > ...
{
  "name": "Alice Katherine",
  "age": 30,
  "skills": ["Terraform", "AWS", "Ansible"],
  "active": true,
  "projects": [
    {
      "name": "Project X",
      "status": false
    },
    {
      "name": "Project Y",
      "completed": true
    }
  ]
}
```



```
test = {
  "active" = true
  "age" = 30
  "name" = "Alice Katherine"
  "projects" = [
    {
      "name" = "Project X"
      "status" = false
    },
    {
      "completed" = true
      "name" = "Project Y"
    }
  ]
  "skills" = [
    "Terraform",
    "AWS",
    "Ansible",
  ]
}
```

# Example 1 - Filtering Values Based on Keys

You can fetch values associated with specific keys from JSON.

```
locals {
    json_string = <<EOT
    {
        "name": "Mr A",
        "age": 30,
        "city": "Mumbai"
    }
    EOT

    decoded_json = jsondecode(local.json_string)
}

output "test" {
    value = local.decoded_json["name"]
}
```



Outputs:

**test = "Mr A"**

## Example 2 - Array of Strings

You can decode array of strings from JSON and then refer to it through the index.

```
locals {
    json_string = <<EOT
    [
        "apple",
        "banana",
        "cherry"
    ]
EOT

    decoded_json = jsondecode(local.json_string)
}

output "test" {
    value = local.decoded_json[0]
}
```



Outputs:

test = "apple"

## Example 3 - Array of Objects

Following example outputs the first element from the jsondecode data.

```
locals {
    json_string = <<EOT
    [
        { "name": "Alice", "age": 25 },
        { "name": "Bob", "age": 28 },
        { "name": "Charlie", "age": 35 }
    ]
    EOT

    decoded_json = jsondecode(local.json_string)
}

output "test" {
    value = local.decoded_json[0]
}
```



Outputs:

```
test = {
    "age" = 25
    "name" = "Alice"
}
```

## Example 4 - Array of Objects

This example outputs the “name” key within the first element.

```
locals {
    json_string = <<EOT
    [
        { "name": "Alice", "age": 25 },
        { "name": "Bob", "age": 28 },
        { "name": "Charlie", "age": 35 }
    ]
    EOT

    decoded_json = jsondecode(local.json_string)
}

output "test" {
    value = local.decoded_json[0]["name"]
}
```



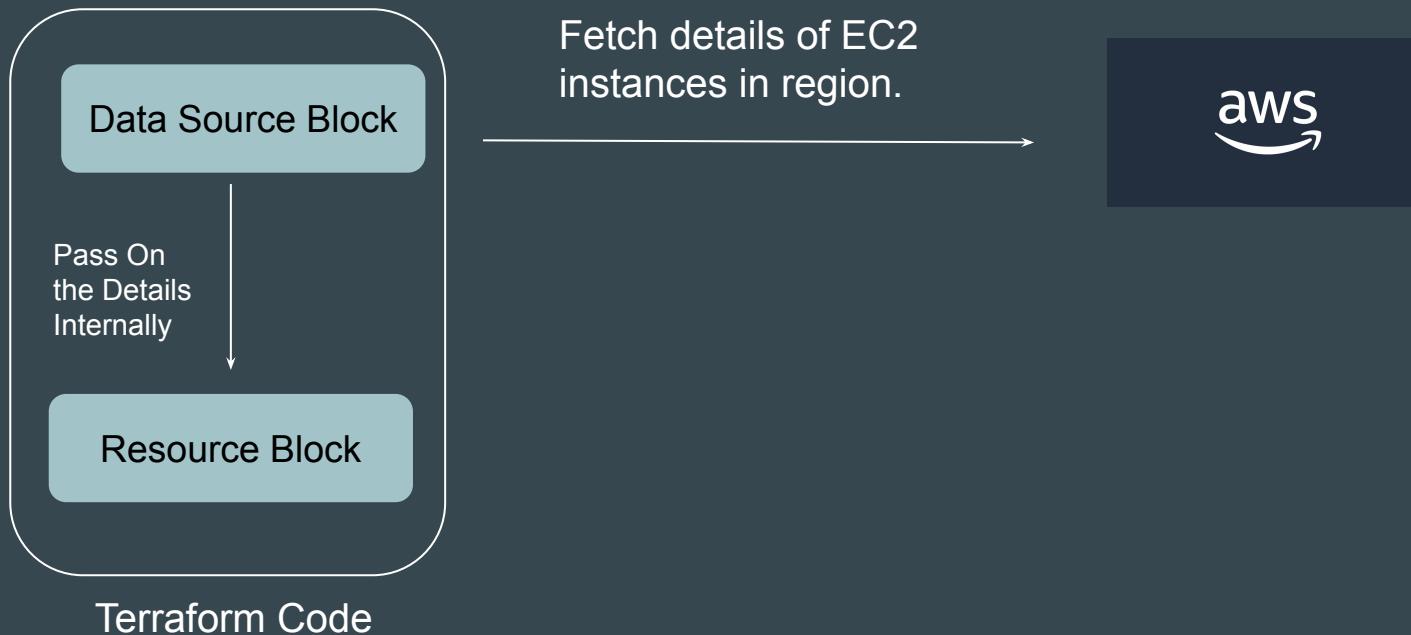
Outputs:

test = "Alice"

# **Data Sources**

# Introducing Data Sources

Data sources allow Terraform to **use / fetch** information defined outside of Terraform



## Example 1 - Reading Info of DO Account

Following data source code is used to get information on your DigitalOcean account.

```
data "digitalocean_account" "example" {}
```

## Example 2 - Reading a File

Following data source allows you to read contents of a file in your local filesystem.

```
data "local_file" "foo" {
    filename = "${path.module}/demo.txt"
}
```

# Clarity regarding path.module

`${path.module}` returns the current file system path where your code is located.

```
data "local_file" "foo" {
    filename = "${path.module}/demo.txt"
}
```

## Example 3 - Fetch EC2 Instance Details

Following data source code is used to fetch details about the EC2 instance in your AWS region.

```
data "aws_instances" "example" {}
```

# **Data Sources Documentation Reference**

# Finding Available Data Sources

List of available data source are associated with each resource of a provider.

The screenshot shows a documentation page for the DigitalOcean Provider. The left sidebar has a 'DIGITALOCEAN DOCUMENTATION' header and a 'Filter' search bar. Below the filter is a list of resources under 'digitalocean provider': 'Resources' (indicated by a right-pointing arrow) and 'Data Sources' (indicated by a downward-pointing arrow). Under 'Data Sources', there is a list of provider names: 'digitalocean\_account', 'digitalocean\_app', 'digitalocean\_certificate', 'digitalocean\_container\_registry', 'digitalocean\_database\_ca', 'digitalocean\_database\_cluster', 'digitalocean\_database\_connection\_pool', and 'digitalocean\_database\_replica'. The main content area is titled 'DigitalOcean Provider' and contains text explaining that the provider interacts with DigitalOcean resources and needs configuration. It also includes a 'Example Usage' section with a code snippet:

```
terraform {  
    required_providers {  
        digitalocean = {  
            source  = "digitalocean/digitalocean"  
            version = "~> 2.0"  
        }  
    }  
}
```

A 'Copy' button is located next to the code snippet.

# **Data Sources Format**

# Understanding the Basic Structure

A data source is accessed via a special kind of resource known as a **data resource**, declared using a **data** block:

Following data block requests that Terraform read from a given data source ("aws\_instance") and export the result under the given local name ("foo").

```
data "aws_instance" "foo" {}
```

# Filter Structure

Within the block body (between { and }) are query constraints defined by the data source.

```
data "aws_instance" "foo" {
  filter {
    name    = "tag:Team"
    values  = ["Production"]
  }
}
```

# **Fetching Latest OS Image Using Data Sources**

# Understanding the Requirement

You have been given a requirement to write a Terraform code that creates EC2 instance using latest OS Image of Amazon Linux.

# Approach that New User will Take

We want to use the latest OS image for creating server in AWS.

Steps that we typically follow:

1. Go to EC2 Console.
2. Fetch the latest AMI ID
3. Add that AMI ID in Terraform code.



# Sample Reference Code

```
resource "aws_instance" "web" {  
    ami              = "ami-0440d3b780d96b29d" ← Hard Coded static value  
    instance_type   = "t2.micro"  
}
```

# Static Information is Boring

Hardcoding static details in your Terraform code will lead you to repeatedly modify your code to meet changing requirements.



# Another Challenge with Static Values

In many of the cases, the static value changes depending on the region.

Example: AMI IDs are specific to region.

Hardcoded AMI in code will only work for single region.

Mumbai Region



ami-1234

Singapore Region



ami-5678

Tokyo Region

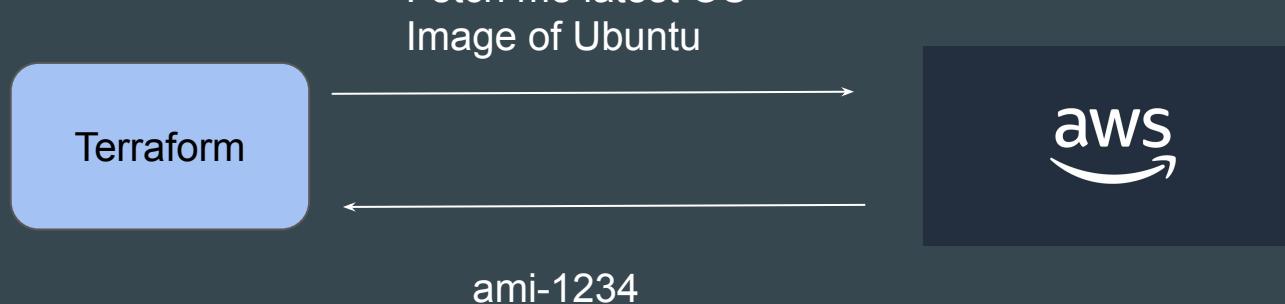


ami-9012

# Time to be Pros - Dynamic Configuration

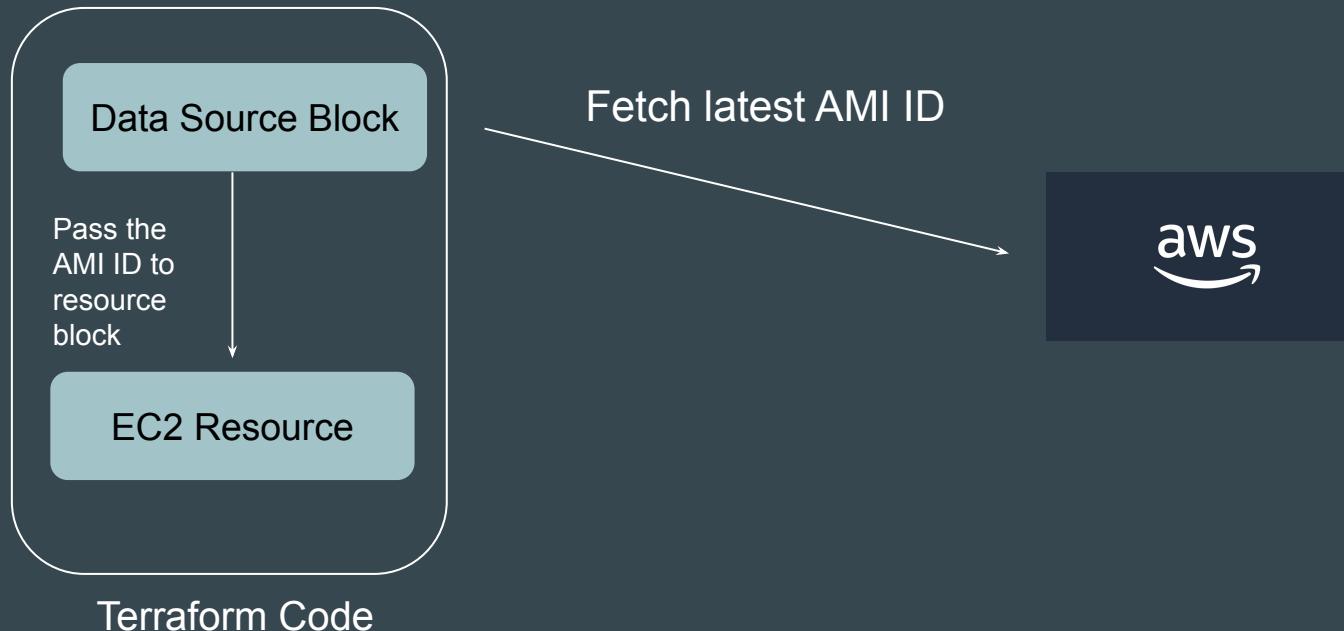
We want Terraform to automatically query the latest OS image in AWS or any other provider and use that for creating server.

We need code which works for all region without modification.



# Introducing Data Sources

Data sources allow Terraform to **use information defined outside of Terraform** and we can use that information to provision resources.



# **Input Variable Validation**

# Understanding the Challenge

Many times, the terraform plan would run properly without errors; however, as soon as you run “terraform apply”, you get an error.

```
C:\kplabs-terraform>terraform plan
```

```
Terraform used the selected providers to generate the following
following symbols:
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_s3_bucket.examplebucket will be created
+ resource "aws_s3_bucket" "examplebucket" {
    + acceleration_status      = (known after apply)
    + acl                      = (known after apply)
    + arn                      = (known after apply)
    + bucket                   = "hi"
    + bucket_domain_name       = (known after apply)
    + bucket_prefix             = (known after apply)
    + bucketRegionalDomainName = (known after apply)
    + force_destroy             = false
```

Plan Stage

```
aws_s3_bucket.this: Creating...
```

```
| Error: creating S3 Bucket (hi): operation error S3: CreateBucket, https response error
T87GNG0E, HostID: W/rGOHVr4EyBDQhKsGWWrY9HLx/NteAN0+e0Un14g9G2r2HgyDJ3djNPaBk9bX8zYlvirI
The specified bucket is not valid.
```

```
with aws_s3_bucket.this,
on tf-logs.tf line 4, in resource "aws_s3_bucket" "this":
  4: resource "aws_s3_bucket" "this" {
```

Apply Stage

# Reason is Due to Lack of Validation

Every service has some restrictions and limits on aspects like naming convention, capacity, etc.

## IAM name requirements

IAM names have the following requirements and restrictions:

- Policy documents can contain only the following Unicode characters: horizontal tab (U+0009), linefeed (U+000A), carriage return (U+000D), and characters in the range U+0020 to U+00FF.
- Names of users, groups, roles, policies, instance profiles, server certificates, and paths must be alphanumeric, including the following common characters: plus (+), equals (=), comma (,), period (.), at (@), underscore (\_), and hyphen (-). Path names must begin and end with a forward slash (/).
- Names of users, groups, roles, and instance profiles must be unique within the account. They aren't distinguished by case, for example, you can't create groups named both **ADMINS** and **admins**.



# Terraform and Provider Validation

For many of the resources, Terraform and it's associated providers will validate the input so that any potential errors can be detected quickly.

```
resource "aws_iam_user" "dev" {
    name = "kplabs-user-01#"
}
```



```
C:\kplabs-terraform>terraform plan
| Error: invalid value for name (must only contain alphanumeric characters, hyphens, underscores, commas, periods, @ symbols, plus and equals signs)
| with aws_iam_user.dev,
| on tf-logs.tf line 5, in resource "aws_iam_user" "dev":
|   5:   name = "kplabs-user-01#"
```

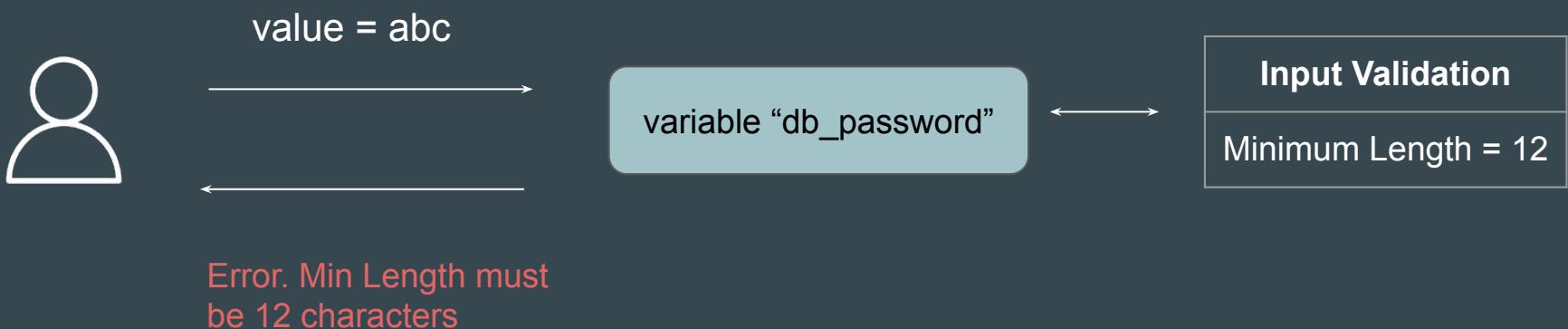
# Terraform and Provider Validation

The provider-side validation works for many of the resources but NOT all of the resources.

Hence, it is essential to have user-side validation to ensure consistency and conformance to the standards.

# Input Variable Validation

The **input variable validation** feature allows you to enforce certain rules or constraints on the values that can be assigned to input variables.



# Reference Screenshot

```
C:\kplabs-terraform>terraform plan
```

```
var.db_password
```

```
  Password for the database
```

```
Enter a value: abc
```

```
Planning failed. Terraform encountered an error while generating this plan.
```

```
Error: Invalid value for variable
```

```
on tf-logs.tf line 3:
```

```
 3: variable "db_password" {
```

```
    |-----|
    | var.db_password is "abc"
```

```
Database password must be at least 12 characters long.
```

```
This was checked by the validation rule at tf-logs.tf:7,3-13.
```

# Importance of Validation

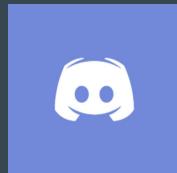
It allows organizations to ensure consistency and adhere to organizational best practices.

Allows organizations to make your Terraform code more predictable.

Allows organizations to catch misconfigurations at an early stage, saving time and potential infrastructure issues.

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



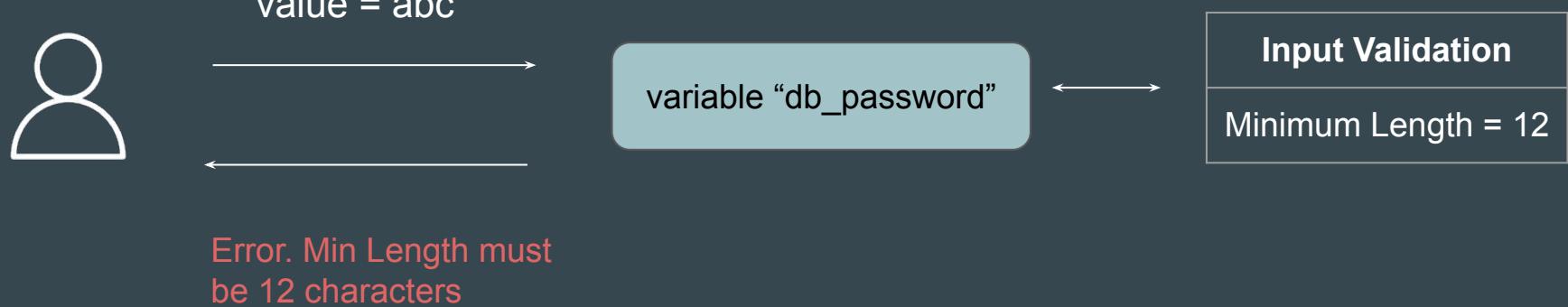
[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Input Variable Validation - Practical**

# Scenario To Consider for Practical

The value for “db\_password” variable must be of minimum 12 characters length.

If user enters anything less than 12, Terraform should throw an error.



# Understanding the Structure

There are three important components of input variable validation feature.

Validation Block, Condition, and Error Message

```
variable "db_password" {
  type = string
  description = "Password for the database"

  → validation {
    →   condition      = length(var.db_password) >= 12
    →   error_message = "Database password must be at least 12 characters long."
    → }
  }
```

# Understanding Each Components

**Validation block** is used within a variable declaration to define one or more conditions that the variable's value must satisfy.

Condition is a boolean expression that must evaluate to true for the validation to pass.

The **error message** is a string that is displayed when the condition fails.

```
variable "db_password" {
  type = string
  description = "Password for the database"

  validation {
    condition      = length(var.db_password) >= 12
    error_message = "Database password must be at least 12 characters long."
  }
}
```

# **More Examples - Input Variable Validation**

# Example 1 - Choose Between Certain Instance Types

This code block allows users to launch EC2 instance based on only 3 instance types.

```
resource "aws_instance" "myec2" {
    ami = "ami-123"
    instance_type = var.instance_type
}

variable "instance_type" {
    type = string

    validation {
        condition      = contains(["t3.micro", "t3.medium", "m5.large"], var.instance_type)
        error_message = "Invalid instance type. Choose from: t3.micro, t3.medium, m5.large"
    }
}
```

## Example 2 - String Matching Condition

This code block ensures that a string matches a specific pattern using regex.

```
variable "email_id" {
  type = string

  validation {
    condition      = can(regex("^\S+@\S+\.\S+$", var.email_id))
    error_message = "The email must be a valid email address"
  }
}
```

## Example 3 - Valid AMI ID for EC2

The following example checks whether the AMI ID has valid syntax.

```
resource "aws_instance" "myec2" {
    ami = var.ami_id
    instance_type = "t2.micro"
}

variable "ami_id" {
    type      = string

    validation {
        condition      = length(var.image_id) > 4 && substr(var.ami_id, 0, 4) == "ami-"
        error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
    }
}
```

# **Preconditions and Postconditions**

# Introducing Pre-Conditions

Terraform preconditions are custom conditions that are checked BEFORE evaluating the object they are associated with.

Example: Launch EC2 instance only if it is part of free tier



Launch Big Server



Terraform

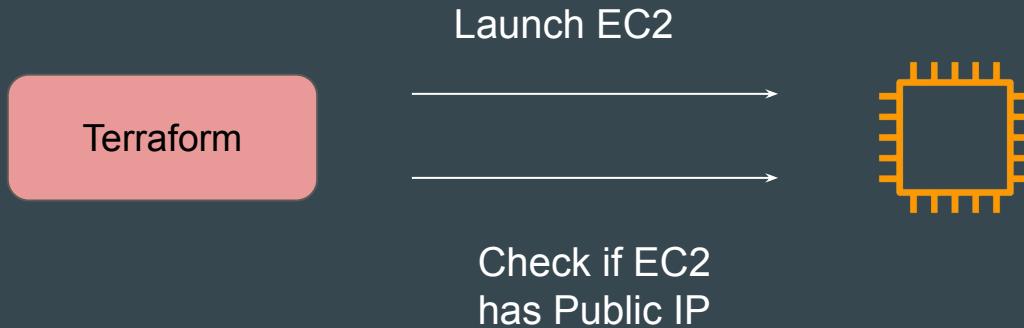
Budget Constraints.

Only Free Tier EC2  
allowed.

# Introducing Post-Conditions

Terraform postconditions are custom conditions that are checked **AFTER** evaluating the object they are associated with.

Example: Verify if EC2 has Public IP address after it has been created.



# Overall Syntax

The precondition and postcondition are defined inside the lifecycle block.

```
resource "aws_instance" "example" {
    instance_type = "t3.micro"
    ami           = "ami-123"

    lifecycle {
        precondition {
            condition      = logic-here
            error_message = "Error Message Here"
        }

        postcondition {
            condition      = logic-here
            error_message = "Error Message Here"
        }
    }
}
```

# Reference Code

```
variable "instance_type" {}

data "aws_ec2_instance_type" "example" {
    instance_type = var.instance_type
}

resource "aws_instance" "example" {
    instance_type = var.instance_type
    ami           = "ami-066784287e358dad1"

    lifecycle {
        precondition {
            condition = data.aws_ec2_instance_type.example.free_tier_eligible
            error_message = "Instance Type is not part of free tier"
        }
    }

    postcondition {
        condition = self.public_dns == ""
        error_message = "Public IPV4 or DNS is mandatory for this server"
    }
}
```

## Point to Note

You can use the `self` object in postcondition blocks to refer to attributes of the instance under evaluation.

Preconditions and postconditions are available in Terraform v1.2.0 and later.

Preconditions and Postconditions are supported for resources, data sources, and outputs

# **More Examples - Preconditions and Postconditions**

## Example 1 - File Exists

The following code ensures that a file named db.txt exists before a local file named app.txt gets created.

```
resource "local_file" "example" {
    filename = "app.txt"
    content  = "Sample App Content"

    lifecycle {
        precondition {
            condition      = fileexists("${path.module}/db.txt")
            error_message = "The db.txt file must exist."
        }
    }
}
```

## Example 2 - Multiple Functions in Condition

The following code ensures that a file named db.txt exists and must not be empty before a local file named app.txt gets created.

```
resource "local_file" "example" {
    filename = "app.txt"
    content  = "Sample App Content"

    lifecycle {
        precondition {
            condition      = fileexists("${path.module}/db.txt") && file("${path.module}/db.txt") != ""
            error_message = "The db.txt file must exist."
        }
    }
}
```

## Example 3 - Referencing to Variable Value

The following code ensures that sensitive.txt file does NOT get created if the environment is not production.

```
variable "environment" {}

resource "local_file" "sensitive_data" {
  filename = "sensitive.txt"
  content  = "This is sensitive data"

  lifecycle {
    precondition {
      condition      = var.environment == "production"
      error_message = "Sensitive File can only be created in Production."
    }
  }
}
```

## Example 4 - Multiple Pre-Conditions

The following code has multiple precondition, both of them should be fulfilled before resource can be created.

```
variable "environment" {}

resource "local_file" "sensitive_data" {
    filename = "sensitive.txt"
    content  = "This is sensitive data"

    lifecycle {
        precondition {
            condition      = var.environment == "production"
            error_message = "Sensitive File can only be created in Production."
        }

        precondition {
            condition      = fileexists("${path.module}/app.txt")
            error_message = "app.txt must exist before sensitive file is created ."
        }
    }
}
```

# **Check Blocks**

# Sample Use-Case

The check block can validate your infrastructure outside the usual resource lifecycle.

The server should only launch if a specific external website is up and running.

A screenshot of the HashiCorp Terraform Continuous Validation interface. The top navigation bar shows the workspace: hashicorp-training / Projects & workspaces / learn-terraform-checks / Health / Continuous Validation. On the left is a dark sidebar with a logo, a search icon, and a menu icon. The main content area has a title 'Continuous validation' with a status indicator 'Ready'. Below it says 'Last successfully checked a few seconds ago. Next assessment in 8 hours.' A message states 'Your infrastructure does not satisfy the assertions defined in your configuration. Learn more about continuous validation.' Below this is a summary table with counts: Failed 1, Passed 0, Unknown 0, All 1. A search bar is present. The table has three columns: ASSERTION, STATUS, and MESSAGE. One row is shown: 'check.certificate' with a status of 'Failed' (indicated by a red circle with a white minus sign) and a count of 1, and the message 'Certificate status is ISSUED'.

ASSERTION	STATUS	MESSAGE
check.certificate	✗ Failed 1	Certificate status is ISSUED

# Introducing Pre-Conditions

Terraform preconditions are custom conditions that are checked **BEFORE** evaluating the object they are associated with.

Example: If the Publicly Accessible Website is UP and responding, launch the server.



Launch only if google.com is accessible



Terraform

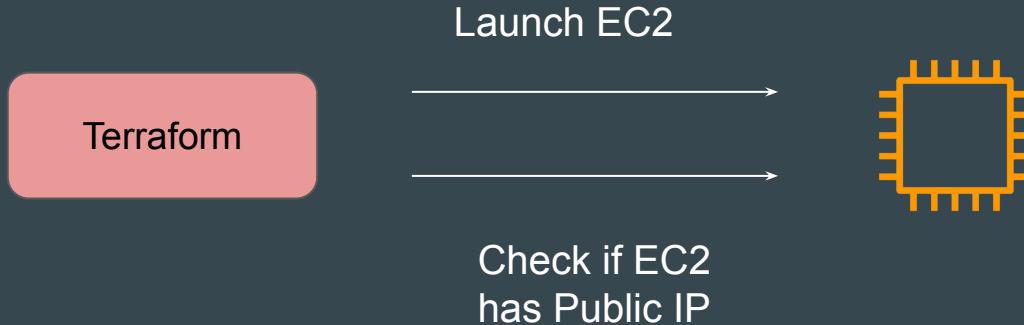


```
resource "aws_instance" "myec2" {  
    ami = "ami-00c39f71452c08778"  
    instance_type = "m5.xlarge"  
}
```

# Introducing Post-Conditions

Terraform preconditions are custom conditions that are checked AFTER evaluating the object they are associated with.

Example: Verify if EC2 has Public IP address after it has been created.



# Overall Syntax

The precondition and postcondition are defined inside the lifecycle block.

```
resource "aws_instance" "example" {
    instance_type = "t3.micro"
    ami           = "ami-123"

    lifecycle {
        precondition {
            condition      = logic-here
            error_message = "Error Message Here"
        }

        postcondition {
            condition      = logic-here
            error_message = "Error Message Here"
        }
    }
}
```

# Reference Code

```
resource "aws_instance" "example" {
  instance_type = "t3.micro"
  ami           = "ami-066784287e358dad1"

  lifecycle {
    precondition {
      condition      = data.aws_ami.example.architecture == "x86_64"
      error_message = "The selected AMI must be for the x86_64 architecture."
    }

    postcondition {
      condition      = self.public_dns != ""
      error_message = "EC2 instance must be in a VPC that has public DNS hostnames enabled."
    }
  }
}
```

## Point to Note

You can use the `self` object in postcondition blocks to refer to attributes of the instance under evaluation.

Preconditions and postconditions are available in Terraform v1.2.0 and later.

# **Sensitive Parameter**

# Setting the Base

By default, Terraform will show the values associated with defined attributes in the CLI output during plan, apply operations for most of the resources.

```
C:\kplabs-terraform>terraform plan

Terraform used the selected providers to generate the following execution plan.
following symbols:
+ create

Terraform will perform the following actions:

# local_file.foo will be created
+ resource "local_file" "foo" {
    + content              = "supersecretpassw0rd!"
    + content_base64sha256 = (known after apply)
    + content_base64sha512 = (known after apply)
    + content_md5          = (known after apply)
    + content_sha1          = (known after apply)
    + content_sha256         = (known after apply)
    + content_sha512         = (known after apply)
    + directory_permission   = "0777"
    + file_permission        = "0777"
    + filename               = "password.txt"
    + id                     = (known after apply)
}
```

# What to Expect

We should design our Terraform code in such way that no sensitive information is available and shown out of the box in CLI Output, Logs, etc.



# Basics of Sensitive Parameter

Adding sensitive parameter ensures that you do not accidentally expose this data in CLI output, log output

```
variable "sensitive_content" {
  sensitive = true
  default   = "supersecretpassw0rd"
}

resource "local_file" "foo" {
  content  = var.sensitive_content
  filename = "password.txt"
}
```



```
Terraform will perform the following actions:

# local_file.foo will be created
+ resource "local_file" "foo" {
  + content          = (sensitive value)
  + content_base64sha256 = (known after apply)
  + content_base64sha512 = (known after apply)
  + content_md5      = (known after apply)
  + content_sha1     = (known after apply)
  + content_sha256    = (known after apply)
  + content_sha512    = (known after apply)
  + directory_permission = "0777"
  + file_permission    = "0777"
  + filename           = "password.txt"
  + id                 = (known after apply)
}
```

# Sensitive Values AND Output Values

If you try to reference sensitive value in output values, Terraform will immediately give you an error.

```
variable "sensitive_content" {  
    sensitive = true  
    default = "supersecretpassw0rd"  
}  
  
resource "local_file" "foo" {  
    content = var.sensitive_content  
    filename = "password.txt"  
}  
  
output "content" {  
    value = local_file.foo.content  
}
```



Error: Output refers to sensitive values

on local\_file.tf line 12:  
12: output "content" {

To reduce the risk of accidentally exporting sensitive data that any root module output containing sensitive data be explicitly m

If you do intend to export this data, annotate the output value :  
sensitive = true

# Sensitive Values AND Output Values

If you still want sensitive content to be available in “output” of state file but should not be visible in CLI Output, Logs, following approach can be used.

```
variable "sensitive_content" {
  sensitive = true
  default   = "supersecretpassw0rd"
}

resource "local_file" "foo" {
  content  = var.sensitive_content
  filename = "password.txt"
}

output "content" {
  value = local_file.foo.content
  sensitive = "true"
}
```



```
Changes to Outputs:
+ content = (sensitive value)

You can apply this plan to save these new output values to the Terraform state

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

Outputs:

content = <sensitive>
```

# Important Point to Note

Sensitive parameter will NOT protect / redact information from State file.

```
variable "sensitive_content" {  
    sensitive = true  
    default = "supersecretpassw0rd"  
}  
  
resource "local_file" "foo" {  
    content  = var.sensitive_content  
    filename = "password.txt"  
}
```

Configuration File

```
...  
"resources": [  
    {  
        "mode": "managed",  
        "type": "local_file",  
        "name": "foo",  
        "provider": "provider[\\"registry.terraform.io/hashicorp/azurerm\"\"]",  
        "instances": [  
            {  
                "schema_version": 0,  
                "attributes": {  
                    "content": "supersecretpassw0rd",  
                    "content_base64": null,  
                    "content_base64sha256": "7Bj9buJUR+BnQAGN/nDad3...  
                }  
            }  
        ]  
    }  
]
```

State File

# Benefits of Mature Providers

Various providers like AWS will automatically consider the password argument for any database instance as sensitive and will redact it as a sensitive value

```
resource "aws_db_instance" "default" {  
    allocated_storage      = 10  
    db_name                = "mydb"  
    engine                 = "mysql"  
    engine_version         = "8.0"  
    instance_class          = "db.t3.micro"  
    username               = "foo"  
    password               = "foobarbaz"  
    parameter_group_name   = "default.mysql8.0"  
    skip_final_snapshot     = true  
}
```

```
C:\kplabs-terraform>terraform apply -auto-approve

Terraform will perform the following actions:

# aws_db_instance.default will be created
+ resource "aws_db_instance" "default" {
    + address                      = (known after apply)
    + allocated_storage              = 10
    + password                      = (sensitive value)
    + performance_insights_enabled = false
    + performance_insights_kms_key_id = (known after apply)
    + performance_insights_retention_period = (known after apply)
    + port                           = (known after apply)
```

---

# Overview of Vault

HashiCorp Certified: Vault Associate

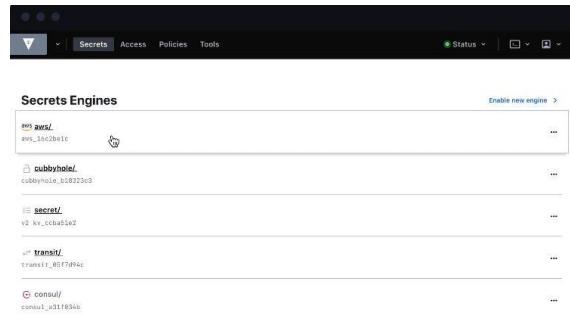
---

# Let's get started

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”

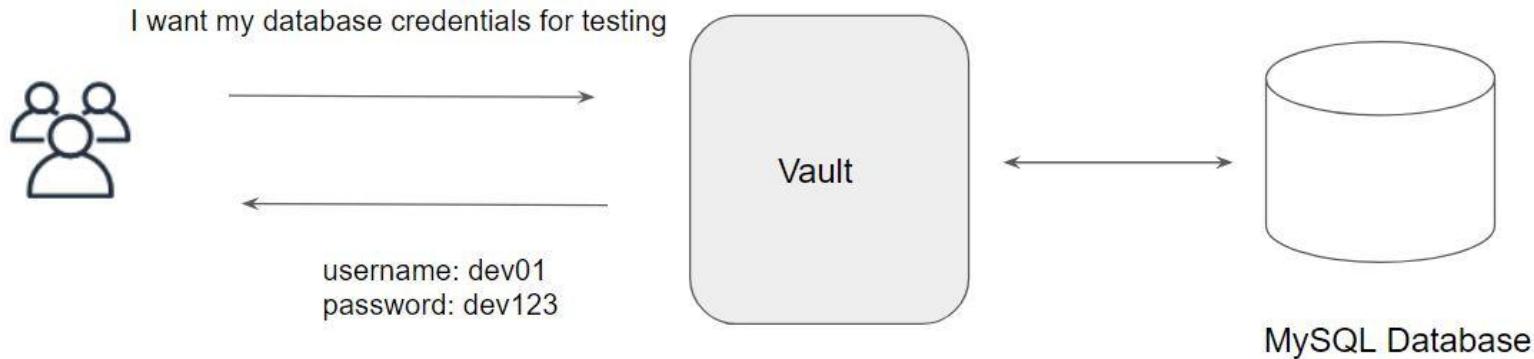
Secrets can include, database passwords, AWS access/secret keys, API Tokens, encryption keys and others.



The screenshot shows the HashiCorp Vault web interface. At the top, there is a navigation bar with tabs for "Secrets", "Access", "Policies", and "Tools". Below the navigation bar, the title "Secrets Engines" is displayed. A button "Enable new engine >" is located in the top right corner of this section. The main area lists several secret engines:

- aws/**  
aws\_16c2b0fc
- cubbyhole/**  
cubbyhole\_b1B31203
- secret/**  
v2\_kv\_cchab1e2
- transit/**  
transit\_85f7d94c
- consul/**  
consul\_a31fb34b

# Dynamic Secrets



# Life Becomes Easier

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

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



---

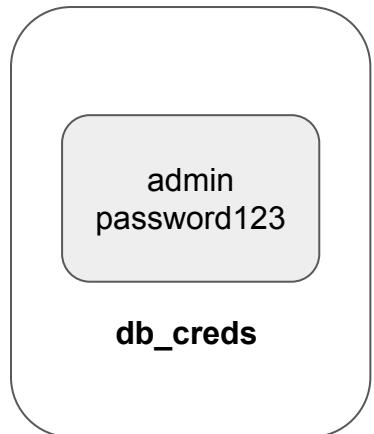
# Vault Provider

[Back to Providers](#)

---

# Vault Provider

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



Inject in Terraform

```
provider "vault" {
  address = "http://127.0.0.1:8200"
}

data "vault_generic_secret" "demo" {
  path = "secret/db-creds"
}
```

Vault

# Important Note

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

# **Resource Behavior and Meta-Argument**

# Understanding the Basics

A **resource block** declares that you want a particular infrastructure object to exist with the given settings

```
resource "aws_instance" "myec2" {
    ami = "ami-00c39f71452c08778"
    instance_type = "t2.micro"
}
```

# How Terraform Applies a Configuration

Create resources that exist in the configuration but are not associated with a real infrastructure object in the state.

Destroy resources that exist in the state but no longer exist in the configuration.

Update in-place resources whose arguments have changed.

Destroy and re-create resources whose arguments have changed but which cannot be updated in-place due to remote API limitations.

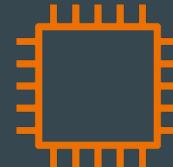
# Understanding the Limitations

What happens if we want to change the default behavior?

Example: Some modification happened in Real Infrastructure object that is not part of Terraform but you want to ignore those changes during terraform apply.

```
resource "aws_instance" "web" {
    ami           = "ami-00c39f71452c08778"
    instance_type = "t3.micro"

    tags = {
        Name = "HelloWorld"
    }
}
```



Name	HelloWorld
------	------------

Env	Production
-----	------------

# Solution - Using Meta Arguments

Terraform allows us to include **meta-argument** within the resource block which allows some details of this standard resource behavior to be customized on a per-resource basis.

Inside resource block

```
resource "aws_instance" "myec2" {
    ami = "ami-00c39f71452c08778"
    instance_type = "t2.micro"

    lifecycle {
        ignore_changes = [tags]
    }
}
```

# Different Meta-Arguments

Meta-Argument	Description
depends_on	Handle hidden resource or module dependencies that Terraform cannot automatically infer.
count	Accepts a whole number, and creates that many instances of the resource
for_each	Accepts a map or a set of strings, and creates an instance for each item in that map or set.
lifecycle	Allows modification of the resource lifecycle.
provider	Specifies which provider configuration to use for a resource, overriding Terraform's default behavior of selecting one based on the resource type name

# **Meta Argument - LifeCycle**

# Basics of Lifecycle Meta-Argument

Some details of the default resource behavior can be customized using the special nested lifecycle block within a resource block body:

```
resource "aws_instance" "myec2" {
    ami = "ami-0f34c5ae932e6f0e4"
    instance_type = "t2.micro"

    tags = {
        Name = "HelloEarth"
    }

    lifecycle {
        ignore_changes = [tags]
    }
}
```

# Arguments Available

There are four argument available within lifecycle block.

Arguments	Description
create_before_destroy	New replacement object is created first, and the prior object is destroyed after the replacement is created.
prevent_destroy	Terraform to reject with an error any plan that would destroy the infrastructure object associated with the resource
ignore_changes	Ignore certain changes to the live resource that does not match the configuration.
replace_triggered_by	Replaces the resource when any of the referenced items change

# Replace Triggered By

Replaces the resource when any of the referenced items change.

```
resource "aws_appautoscaling_target" "ecs_target" {
    # ...

    lifecycle {
        replace_triggered_by = [
            # Replace `aws_appautoscaling_target` each time this instance of
            # the `aws_ecs_service` is replaced.
            aws_ecs_service.svc.id
        ]
    }
}
```

# **LifeCycle - Ignore Changes Argument**

# Ignore Changes

In cases where settings of a remote object is modified by processes outside of Terraform, the Terraform would attempt to "fix" on the next run.

In order to change this behavior and ignore the manually applied change, we can make use of `ignore_changes` argument under `lifecycle`.

```
resource "aws_instance" "myec2" {
    ami = "ami-00c39f71452c08778"
    instance_type = "t2.micro"

    lifecycle {
        ignore_changes = [tags]
    }
}
```

# Points to Note

Instead of a list, the special keyword `all` may be used to instruct Terraform to ignore all attributes, which means that Terraform can create and destroy the remote object but will never propose updates to it.

```
resource "aws_instance" "myec2" {
    ami = "ami-0f34c5ae932e6f0e4"
    instance_type = "t2.micro"

    tags = {
        Name = "HelloEarth"
    }

    lifecycle {
        ignore_changes = all
    }
}
```

# **Data Type - List**

# List Data Type

Allows us to store **collection of values** for a single variable / argument.

Represented by a pair of square brackets containing a comma-separated sequence of values, like ["a", 15, true].

Useful when multiple values needs to be added for a specific argument

```
variable "my-list" {  
  type = list  
  default = ["mumbai","bangalore","delhi"]  
}
```

# Data Type and Documentation

Arguments for a resource requires specific data types.

Some argument requires list, some requires map and so on.

The details of data type expected for an argument is mentioned in documentation.

- `volume_tags` - (Optional) Map of tags to assign, at instance-creation time, to root and EBS volumes.

**⚠ NOTE:**

Do not use `volume_tags` if you plan to manage block device tags outside the `aws_instance` configuration, such as using `tags` in an `aws_ebs_volume` resource attached via `aws_volume_attachment`. Doing so will result in resource cycling and inconsistent behavior.

- `vpc_security_group_ids` - (Optional, VPC only) List of security group IDs to associate with.

# Use-Case 1: List Data Type

EC2 instance can have one or more security groups.

**Requirement:**

Create EC2 instance with 2 security groups attached.

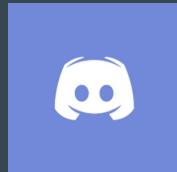
# Specify the Type of Values in List

We can also specify the type of values expected in a list.

```
variable "my-list" {  
    type = list(number)  
    default  = ["1","2","3"]  
}
```

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Map - Data Type**

# Map Data Type

A map data type represents a **collection of key-value pair elements**

```
variable "instance_tags" {  
  type = map  
  default = {  
    Name = "app-server"  
    Environment = "development"  
    Team = "payments"  
  }  
}
```

# Use-Case of Map

We can add multiple tags to AWS resources.

These tags are key-value pairs.

The screenshot shows the AWS EC2 Instances page with one instance listed:

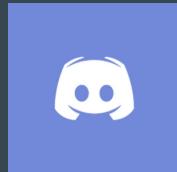
- Name:** kplabs-ec2
- Instance ID:** i-0b8abce20ade4fe35
- Instance state:** Running
- Instance type:** t2.micro
- Status check:** Initializing

The instance details are shown in the main pane:

- Tags:** Environment: Production, Team: Security, Name: kplabs-ec2

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Data Type - Object**

# Revising Map Data Type

In a basic setup, a map is a collection of key-value pairs where all values must be of the same type whereas keys are string.

Strict structure is not required explicitly.

```
variable "my-map" {
  type = map(number)
}

output "variable_value" {
  value = var.my-map
}
```



```
C:\kplabs-terraform>terraform plan
var.my-map
  Enter a value: {"Name"="123","Location"="456"}

Changes to Outputs:
  + variable_value = {
      + Location = 456
      + Name     = 123
    }
```

# Introducing Object Data Type

An **object** is also a collection of key-value pairs, but each value can be of a different type.

A proper structure is generally required while defining object data type.

```
variable "my-object" {
  type = object({Name = string, userID = number})
}

output "variable_value" {
  value = var.my-object
}
```



```
C:\kplabs-terraform>terraform plan
var.my-object
  Enter a value: {"Name"="Zeal", "userID=123}

Changes to Outputs:
+ variable_value = {
    + Name    = "Zeal"
    + userID = 123
  }
```

# Proper Structure is Required

It is important to have a structure of attributes allowed as part of the object.

```
variable "my-map" {  
    type = map  
}
```

This will work

```
variable "my-object" {  
    type = object  
}
```

This will NOT work

# Keep in Mind the Syntax

`object(...)`: a collection of named attributes that each have their own type.

The schema for object types is `{ <KEY> = <TYPE>, <KEY> = <TYPE>, ... }` — a pair of curly braces containing a comma-separated series of `<KEY> = <TYPE>` pairs.

Extra attributes are discarded during type conversion.

# Example - Extra Attributes Provided

In the following example, an additional key=value pair is provided that is not part of the object structure. We see on how it gets discarded in the output.

```
variable "my-object" {
  type = object({Name = string, userID = number})
}

output "variable_value" {
  value = var.my-object
}
```



```
C:\kplabs-terraform>terraform plan
var.my-object
Enter a value: {"Name"="Zeal","userID"=123,"email"="instructors@kplabs.in"}

Changes to Outputs:
+ variable_value = {
    + Name      = "Zeal"
    + userID   = 123
}
```

# **Nesting in Variable Types**

# Setting the Base

We have been using common data types that Terraform supports like list, maps

The variable types also support **nesting**, which allows you to build complex and highly structured configurations

```
variable "my-map" {  
  type = map  
  default = {  
    Name = "Alice"  
    Team = "Payments"  
  }  
}
```

Data Type of Map

```
variable "my-list" {  
  type = list  
  default = ["us-east-1a","us-east-1b"]  
}
```

Data Type of List

## Example 1 - List of Lists

A list of lists is a list that contains lists as its elements

Each inner list can have a different length, allowing for different structures.

```
variable "list_of_list" {  
    type = list(list(string))  
}
```



```
variable "list_of_list" {  
    type = list(list(string))  
    default = [[{"alice"}, {"bob"}, {"john"}]]  
}
```

## Example 2 - List of Maps

List of maps is essentially a list where each element is a map, allowing you to organize and structure complex data effectively.

```
variable "list_of_maps" {  
  type = list(map(string)) →  
}  
}
```

```
variable "list_of_maps" {  
  type = list(map(string))  
  default = [  
    {  
      user  = "alice"  
      email = "alice@kplabs.in"  
    },  
    {  
      user  = "john"  
      email = "john@kplabs.in"  
    }  
  ]  
}
```

## Example 3 - Map of Maps

Map of maps is a structure where the values are themselves maps. This enables you to create nested structures, making your configuration more organized and readable.

```
variable "map_of_maps" {  
    type = map(map(string))  
}
```



```
variable "map_of_maps" {  
    type = map(map(string))  
    default = {  
        dev = {  
            Name     = "nginx-server"  
            Team     = "Security"  
        },  
        prod = {  
            Name     = "web-server"  
            Team     = "SRE"  
        }  
    }  
}
```

## Example 4 - Map of Lists

It is a map where the values are lists.

```
variable "lists" {  
    type = map(list(string))  
}
```

```
variable "map_of_lists" {  
    type = map(list(string))  
    default = {  
        value1=["abc"],  
        value2=["def"]  
    }  
}
```

## Example 5 - List of Objects

List of Objects allows us to define a list where each element is an object.

```
variable "list_of_object" {  
    type = list(object({  
        name = string  
        age  = number  
        email = string  
        tags  = map(string)  
    }))  
}
```



```
variable "list_of_object" {  
    type = list(object({  
        name = string  
        age  = number  
        email = string  
        tags  = map(string)  
    }))  
  
    default = [  
        {  
            name = "alice"  
            age  = 32  
            email = "alice@kplabs.internal"  
            tags  = {"env"="dev", "provider"="aws"}  
        }  
    ]  
}
```

# Relax and Have a Meme Before Proceeding

How I feel after a few people upvoted  
my meme, knowing I made some people smile:



# **Fetching Values from Nested Variable Types**

# Basics of Length Function

length function determines the length of a given list, map, or string.

```
variable "demo_lists" {
  type = list
  default = ["apple","banana","mango","grapes"]
}

output "list_values" {
  value = length(var.demo_lists)
}
```



Outputs:

list\_values = 4

# Example 1 - Length Function with Nested List

```
variable "matrix_config" {  
  type = list(list(string))  
  default = [  
    ["a1", "a2", "a3"],  
    ["b1", "b2", "b3"],  
    ["c1", "c2"]  
  ]  
}  
  
output "matrix_value" {  
  value = length(var.matrix_config)  
}
```



Outputs:

**matrix\_value = 3**

## Example 2 - Length Function with Nested List

```
variable "list_2" {  
    type = list(list(number))  
    default = [[1],[2,3],[4,5,6]]  
}  
  
output "this" {  
    value = length(var.list_2)  
}
```



Outputs:  
this = 3

# Referencing to Specific Index

Each inner list is associated with a specific index.

```
variable "matrix_config" {  
  type = list(list(string))  
  default = [  
    ["a1", "a2", "a3"],  
    ["b1", "b2", "b3"],  
    ["c1", "c2", "c3"]  
  ]  
}  
  
output "matrix_value" {  
  value = var.matrix_config  
}
```



Index	Items
0	"a1", "a2", "a3"
1	"b1", "b2", "b3"
2	"c1", "c2", "c3"

# Output Screenshot - Referencing to Index

Within the list, you can reference to a specific item using its index.

```
variable "matrix_config" {  
  type = list(list(string))  
  default = [  
    ["a1", "a2", "a3"],  
    ["b1", "b2", "b3"],  
    ["c1", "c2"]  
  ]  
  
  output "list_values" {  
    value = var.matrix_config[0]  
  }  
}
```

Outputs:

```
list_values = tolist([  
  "a1",  
  "a2",  
  "a3",  
])
```

# Referencing to Specific Value

To access elements in a nested list, you use multiple indices – one for each level of the list.

The **first index** specifies which of the inner lists to access, and the **subsequent indices** specify the element within that inner list.

```
variable "matrix_config" {  
  type = list(list(string))  
  default = [  
    ["a1", "a2", "a3"],  
    ["b1", "b2", "b3"],  
    ["c1", "c2"]  
  ]  
}  
  
output "list_values" {  
  value = var.matrix_config[0][1]  
}
```



Outputs:  
list\_values = "a2"

# Example 1 - Reference Values from List of Maps

```
variable "users" {
  type = list(map(string))
  default = [
    {
      name  = "john"
      role  = "admin"
      email = "john@example.com"
    },
    {
      name  = "jane"
      role  = "developer"
      email = "jane@example.com"
    }
  ]
}

output "map" {
  value = var.users[0]
}
```



Outputs:

```
list_values = 2
map = tomap({
  "email" = "john@example.com"
  "name" = "john"
  "role" = "admin"
})
```

## Example 2 - Reference Values from List of Maps

```
variable "users" {
  type = list(map(string))
  default = [
    {
      name  = "john"
      role  = "admin"
      email = "john@example.com"
    },
    {
      name  = "jane"
      role  = "developer"
      email = "jane@example.com"
    }
  ]
}

output "map" {
  value = var.users[0]["email"]
}
```



Outputs:

```
map = "john@example.com"
```

# Example 3 - Map of Maps

```
variable "environment_config" {  
  type = map(map(string))  
  default = {  
    dev = {  
      instance_type = "t2.micro"  
      region       = "us-east-1"  
      replicas     = "1"  
    }  
    prod = {  
      instance_type = "t2.large"  
      region       = "us-west-2"  
      replicas     = "3"  
    }  
  }  
  
  output "list_values" {  
    value = var.environment_config[ "dev" ]  
  }  
}
```

Outputs:

```
list_values = tomap({  
  "instance_type" = "t2.micro"  
  "region"       = "us-east-1"  
  "replicas"     = "1"  
})
```

# Example 4 - List of Objects

```
variable "instances" {
  type = list(object({
    name      = string
    instance_type = string
    tags      = map(string)
  }))
  default = [
    {
      name      = "web-server"
      instance_type = "t2.micro"
      tags      = { "env" = "production", "role" = "web" }
    },
    {
      name      = "db-server"
      instance_type = "t2.medium"
      tags      = { "env" = "production", "role" = "db" }
    }
  ]
}

output "instances" {
  value = var.instances[0]
}
```



## Outputs:

```
instances = {
  "instance_type" = "t2.micro"
  "name" = "web-server"
  "tags" = tomap({
    "env" = "production"
    "role" = "web"
  })
}
```

---

# Challenges with Count

Meta-Argument

# Revising the Basics

Resources are identified by the index value from the list.

```
variable "iam_names" {
  type = list
  default = ["user-01","user-02","user-03"]
}

resource "aws_iam_user" "iam" {
  name = var.iam_names[count.index]
  count = 3
  path = "/system/"
```



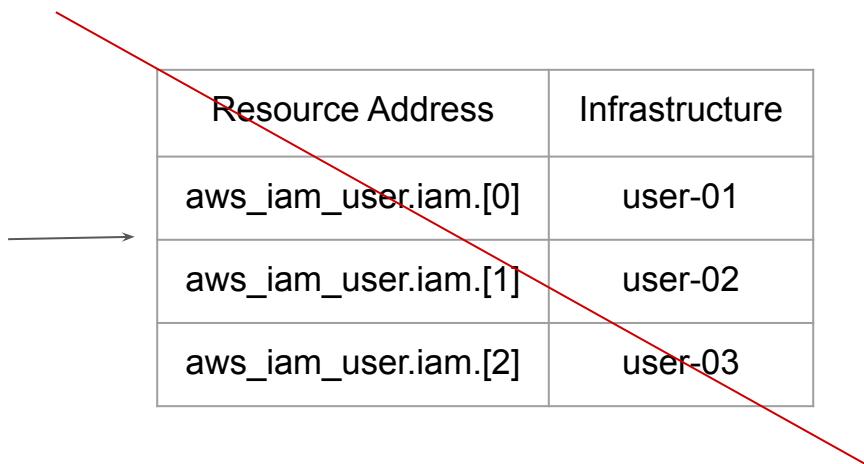
Resource Address	Infrastructure
aws_iam_user.iam[0]	user-01
aws_iam_user.iam[1]	user-02
aws_iam_user.iam[2]	user-03

# Challenge - 1

If the order of elements of index is changed, this can impact all of the other resources.

```
variable "iam_names" {
  type = list
  default = ["user-0","user-01","user-02","user-03"]
}

resource "aws_iam_user" "iam" {
  name = var.iam_names[count.index]
  count = 4
  path = "/system/"
}
```



# Important Note

If your resources are almost identical, count is appropriate.

If distinctive values are needed in the arguments, usage of `for_each` is recommended.

```
resource "aws_instance" "server" {
    count = 4 # create four similar EC2 instances
    ami      = "ami-a1b2c3d4"
    instance_type = "t2.micro"
}
```

# **SET - Data Type**

# Revising List Data Type

Lists are used to store multiple items in a single variable.

These items can be duplicates as well.

```
C:\Users\zealv\kplabs-terraform>terraform apply -auto-approve
var.my-list
  Enter a value: ["hello","world","hello"]

Changes to Outputs:
+ variable_output = [
    + "hello",
    + "world",
    + "hello",
  ]

You can apply this plan to save these new output values to the Terraform
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

variable_output = tolist([
  "hello",
  "world",
  "hello",
])
```

# List and Index

List items are indexed, the first item has index [0], the second item has index [1] etc.

```
variable "user_names" {  
    type = list  
    default = ["alice", "bob", "john"]  
}
```

0      1      2

The diagram illustrates the indexing of a list. Three white arrows point upwards from the numbers 0, 1, and 2 positioned below the list to the corresponding elements "alice", "bob", and "john" in the "default" array defined in the code. This visualizes how each index corresponds to a specific item in the list.

# SET Data Type

Sets can only store unique elements.

Any duplicates are automatically removed.

```
C:\Users\zealv\kplabs-terraform>terraform apply -auto-approve
var.my-set
  Enter a value: ["hello","world","hello"]

Changes to Outputs:
  + variable_output = [
      + "hello",
      + "world",
    ]
You can apply this plan to save these new output values to the Terraform
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

variable_output = toset([
  "hello",
  "world",
])
```

# Point to Note

While defining a SET, you need to also define the type of value that is expected.

```
variable "my-set" {
    type = set(string)
}
```

# SET is Unordered

A set does not store the order of the elements.

Terraform only tracks the presence of elements, not their order.

If the elements in a set change order, Terraform won't detect that as a change. However, if an element is added or removed, Terraform will apply updates accordingly.

```
variable "user_names" {  
    type = list  
    default = ["alice","bob","john"]  
}
```

```
variable "user_names" {  
    type = set(string)  
    default = ["john","bob","alice"]  
}
```

# **The for\_each Meta-Argument**

# Setting the Base

By default, a resource block configures one real infrastructure object.

However, sometimes you want to manage several similar objects (like a fixed pool of compute instances) without writing a separate block for each one.

Terraform has two ways to do this: `count` and `for_each`.

```
resource "aws_iam_user" "lb" {
    name = "alice"
}
```

# Creating 5 IAM Users

If we want to create multiple resources with different configuration, we have to add multiple different resource blocks.



```
iam.tf
●
iam.tf > ...

resource "aws_iam_user" "lb" {
    name = "alice"
}

resource "aws_iam_user" "lb" {
    name = "bob"
}

resource "aws_iam_user" "lb" {
    name = "johh"
}

resource "aws_iam_user" "lb" {
    name = "james"
}

resource "aws_iam_user" "lb" {
    name = "will"
}
```

# Introducing for\_each

If a resource block includes a `for_each` meta argument whose value is a map or a set of strings, Terraform creates one instance for each member of that map or set.

```
variable "user_names" {  
    type = set(string)  
    default = ["alice", "bob"]  
}  
  
resource "aws_iam_user" "lb" {  
    for_each = var.user_names  
    name = each.value  
}
```



```
C:\kplabs-terraform>terraform plan  
  
Terraform used the selected providers to generate the following  
following symbols:  
+ create  
  
Terraform will perform the following actions:  
  
# aws_iam_user.lb["alice"] will be created  
+ resource "aws_iam_user" "lb" {  
    + arn          = (known after apply)  
    + force_destroy = false  
    + id           = (known after apply)  
    + name         = "alice"  
    + path          = "/"  
    + tags_all     = (known after apply)  
    + unique_id    = (known after apply)  
}  
  
# aws_iam_user.lb["bob"] will be created  
+ resource "aws_iam_user" "lb" {  
    + arn          = (known after apply)  
    + force_destroy = false  
    + id           = (known after apply)  
    + name         = "bob"  
    + path          = "/"  
    + tags_all     = (known after apply)  
    + unique_id    = (known after apply)  
}
```

# Point to Note

In blocks where `for_each` is set, an additional `each` object is available.

These object has two attributes:

Each Object	Description
<code>each.key</code>	The map key (or set member) corresponding to this instance.
<code>each.value</code>	The map value corresponding to this instance

# Example - for\_each with Map

When `for_each` is used with map, we can make use of each object to extract both key and value from the given map.

```
variable "mymap" {
  default = {
    dev = "ami-123"
    prod = "ami-456"
  }
}

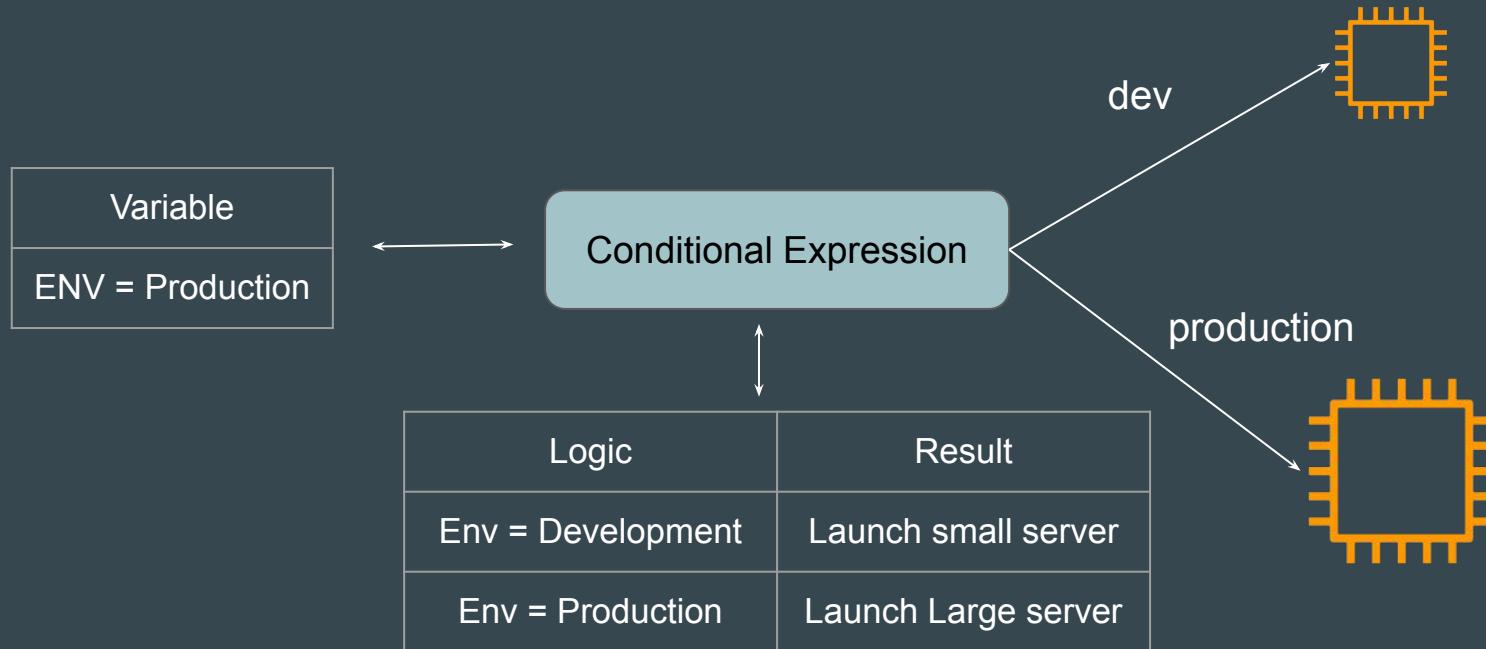
resource "aws_instance" "web" {
  for_each      = var.mymap
  ami           = each.value
  instance_type = "t3.micro"

  tags = {
    Name = each.key
  }
}
```

# Conditional Expressions

# Setting the Base

Conditional expressions in Terraform allow you to choose between two values based on a condition



# Syntax of Conditional Expression

The syntax of a conditional expression is as follows:

```
condition ? true_val : false_val
```

If condition is true then the result is true\_val. If condition is false then the result is false\_val.

# Conditional Expression Based on Use-Case

If Environment is Development, t2.micro instance type should be used.

If Environment is NOT development, m5.large instance type should be used.

```
variable "environment" {
  default = "development"
}

resource "aws_instance" "example" {
  instance_type = var.environment == "development" ? "t2.micro" : "m5.large"
  ami           = "ami-12345678"
}
```

# Conditional Expression with Multiple Variables

In the following example, **only if** env=production and region=us-east-1, the larger instance type of m5.large can be used.

```
variable "environment" {
    default = "production"
}

variable "region" {
    default = "ap-south-1"
}

resource "aws_instance" "example" {
    instance_type = var.environment == "production" && var.region == "us-east-1" ? "m5.large" : "t2.micro"
    ami           = "ami-12345678"
}
```

# for Expressions

# Setting the Base

for Expressions are used to transform certain values to perform specific operations.

Use-Case: Transform values in a list to all upper case.

```
variable "user_names" {  
    type = list  
    default = ["alice","bob","john"]  
}
```



```
output "test" {  
    value = [for data in var.user_names : upper(data)]  
}
```

```
C:\kplabs-terraform>terraform plan  
  
Changes to Outputs:  
+ test = [  
  + "ALICE",  
  + "BOB",  
  + "JOHN",  
]
```

# How it Works

This for expression **iterates over each element** of var.list and the “data” represents element as part of an iteration.

It then passes the value of “data” to upper() for transformation to happen.

```
variable "user_names" {  
    type = list  
    default = ["alice","bob","john"]  
}  
  
output "test" {  
    value = [for data in var.user_names : upper(data)]  
}
```

```
C:\kplabs-terraform>terraform plan  
  
Changes to Outputs:  
+ test = [  
    + "ALICE",  
    + "BOB",  
    + "JOHN",  
]
```

# **Terraform Function - csvdecode**

# Revising Basics of CSV

Comma-separated values (**CSV**) is a text file format that uses commas to separate values, and newlines to separate records

Each line of file typically represents one data record

demo.csv

```
instance_id,instance_type,ami
instance1,t2.micro,ami-1234
instance2,t2.small,ami-4576
instance3,instance_type,ami-7891
instance4,m5.large,ami-2345
```

	A	B	C
1	instance_id	instance_type	ami
2	instance1	t2.micro	ami-1234
3	instance2	t2.small	ami-4576
4	instance3	instance_type	ami-7891
5	instance4	m5.large	ami-2345

# Handling CSV Files in Terraform

**csvdecode** function decodes a string containing CSV-formatted data and produces a list of maps representing that data.

	A	B	C
1	instance_id	instance_type	ami
2	instance1	t2.micro	ami-1234
3	instance2	t2.small	ami-4576
4	instance3	instance_type	ami-7891
5	instance4	m5.large	ami-2345



```
> csvdecode(file("./demo.csv"))
tolist([
{
  "ami" = "ami-1234"
  "instance_id" = "instance1"
  "instance_type" = "t2.micro"
},
{
  "ami" = "ami-4576"
  "instance_id" = "instance2"
  "instance_type" = "t2.small"
},
{
  "ami" = "ami-7891"
  "instance_id" = "instance3"
  "instance_type" = "instance_type"
},
{
  "ami" = "ami-2345"
  "instance_id" = "instance4"
  "instance_type" = "m5.large"
},
])
```

# **for Expression - Part 2**

# How it Works

This for expression **iterates over each element** of var.list and the “data” represents element as part of an iteration.

It then passes the value of “data” to upper() for transformation to happen.

```
variable "user_names" {  
    type = list  
    default = ["alice","bob","john"]  
}  
  
output "test" {  
    value = [for data in var.user_names : upper(data)]  
}
```

```
C:\kplabs-terraform>terraform plan  
  
Changes to Outputs:  
+ test = [  
    + "ALICE",  
    + "BOB",  
    + "JOHN",  
]
```

# Supported Input Types

A for expression's input (given after the in keyword) can be a list, a set, a tuple, a map, or an object.

# Dealing with Map

Map can contain multiple set of key,value pairs.

Hence, we can make use of two symbols to reference to key and value separately.

```
variable "mymap" {  
    default = {  
        dev = "ami-123"  
        prod = "ami-456"  
    }  
}
```



```
output "test" {  
    value = [for k,v in var.mymap : upper(k)]  
}
```

```
C:\kplabs-terraform>terraform plan  
Changes to Outputs:  
+ test = [  
    + "DEV",  
    + "PROD",  
]
```

# Point to Note

You can also use the two-symbol form with lists and tuples, in which case the additional symbol is the index of each element starting from zero, which conventionally has the symbol name `i` or `idx`.

```
variable "user_names" {
  type = list
  default = ["alice", "bob", "john"]
}

output "test" {
  value = [for i, v in var.user_names : "${i} is ${v}"]
}
```



```
C:\kplabs-terraform>terraform plan

Changes to Outputs:
+ test = [
    + "0 is alice",
    + "1 is bob",
    + "2 is john",
]
```

# Filtering Elements

A for expression can also include an optional **if** clause to filter elements from the source collection, producing a value with fewer elements than the source value

```
variable "user_names" {
  type = list
  default = ["alice","bob","john"]
}

output "test" {
  value = [for data in var.user_names : upper(data) if data != "alice"]
}
```



```
C:\kplabs-terraform>terraform plan

Changes to Outputs:
+ test = [
    + "BOB",
    + "JOHN",
]
```

# Result Types - Tuple

The type of brackets around the for expression decide what type of result it produces.

The [ ] produces tuple

# Result Types - Object

If you use { and } instead, the result is an object and you must provide two result expressions that are separated by the => symbol:

```
variable "user_names" {
  type = list
  default = ["alice", "bob", "john"]
}

output "test" {
  value = {for data in var.user_names : data => upper(data)}
}
```



```
C:\kplabs-terrafarm>terraform plan

Changes to Outputs:
+ test = {
    + alice = "ALICE"
    + bob   = "BOB"
    + john  = "JOHN"
}
```

# **Create Multiple Resource based on CSV Values**

# Sample Use-Case

We want to create 4 EC2 instances based on the details specified in the CSV file.

To avoid multiple repeated `aws_instance` blocks, make use of `for_each`

	A	B	C
1	instance_id	instance_type	ami
2	instance1	t2.micro	ami-1234
3	instance2	t2.small	ami-4576
4	instance3	instance_type	ami-7891
5	instance4	m5.large	ami-2345

# Step 1 - Use the CSV Decode Function

Make use of csvdecode () to produces a list of maps representing that data.

```
locals {  
    data = csvdecode(file("./demo.csv"))  
}
```



```
+ demo = [  
+ {  
+   ami           = "ami-1234"  
+   instance_id  = "instance1"  
+   instance_type = "t2.micro"  
},  
+ {  
+   ami           = "ami-4576"  
+   instance_id  = "instance2"  
+   instance_type = "t2.small"  
},  
+ {  
+   ami           = "ami-7891"  
+   instance_id  = "instance3"  
+   instance_type = "t3.micro"  
},  
+ {  
+   ami           = "ami-2345"  
+   instance_id  = "instance4"  
+   instance_type = "m5.large"  
},  
]
```

# List of Maps

When you use `csvdecode()`, Terraform will convert this into a list of maps.

However, Terraform's `for_each` can't work with a list directly because a list doesn't have unique keys. Terraform needs a map where each instance has a unique key

```
+ demo = [
+ {
+   ami          = "ami-1234"
+   instance_id  = "instance1"
+   instance_type = "t2.micro"
},
{
+   ami          = "ami-4576"
+   instance_id  = "instance2"
+   instance_type = "t2.small"
},
{
+   ami          = "ami-7891"
+   instance_id  = "instance3"
+   instance_type = "t3.micro"
},
{
+   ami          = "ami-2345"
+   instance_id  = "instance4"
+   instance_type = "m5.large"
},
]
```

# Need unique key

We want to transfer list of maps to map which has a unique key so that `for_each` can work.

```
+ demo = [
+ {
+   + ami           = "ami-1234"
+   + instance_id  = "instance1"
+   + instance_type = "t2.micro"
},
+ {
+   + ami           = "ami-4576"
+   + instance_id  = "instance2"
+   + instance_type = "t2.small"
},
+ {
+   + ami           = "ami-7891"
+   + instance_id  = "instance3"
+   + instance_type = "t3.micro"
},
+ {
+   + ami           = "ami-2345"
+   + instance_id  = "instance4"
+   + instance_type = "m5.large"
},
]
```



```
demo = {
  "instance1" = {
    instance_id  = "instance1"
    instance_type = "t2.micro"
    ami           = "ami-1234"
  },
  "instance2" = {
    instance_id  = "instance2"
    instance_type = "t2.small"
    ami           = "ami-4576"
  },
  "instance3" = {
    instance_id  = "instance3"
    instance_type = "t3.micro"
    ami           = "ami-7891"
  }
}
```

# for expression to rescue

Expression used:

```
{for data in local.data : data.instance_id => data}
```

```
demo = {
    "instance1" = {
        instance_id    = "instance1"
        instance_type = "t2.micro"
        ami           = "ami-1234"
    },
    "instance2" = {
        instance_id    = "instance2"
        instance_type = "t2.small"
        ami           = "ami-4576"
    },
    "instance3" = {
        instance_id    = "instance3"
        instance_type = "t2.medium"
        ami           = "ami-7891"
    }
}
```

# Final Code using for\_each

Expression used:

```
demo.tf > ...

locals {
    data = csvdecode(file("./demo.csv"))
}

resource "aws_instance" "web" {
    for_each = {for data in local.data : data.instance_id => data}
    ami        = each.value.ami
    instance_type = each.value.instance_type
}
```

# **Nested Loops in For Expression**

# Basics of Nested Loops

Nested loop is a loop specified inside another loop

```
variable "list_01" {
  default = [1,2,3]
}

variable "list_02" {
  default = [4,5,6]
}

output "test" {
  value = [for data in var.list_01 : [for value in var.list_02 : "${data} ${value}"]]
}
```

# Example 1 - Nested Loop

Identify what this code is trying to do without running.

```
locals {
    x = range(3)
    y = range(1,10)
}

output "nested_for_loop_output" {
    value = [for x in local.x : [for y in local.y : y]]
}
```

# Reference Output Screenshot

```
locals {
    x = range(3)
    y = range(1,10)
}

output "nested_for_loop_output" {
    value = [for x in local.x : [for y in local.y : y]]
}
```



```
nested_for_loop_output = [
    [
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
    ],
    [
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
    ],
    [
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
    ],
]
```

## Point to Note

For each iteration of the outer loop, all of the iterations of the inner loop are executed before moving on to the next iteration of the outer loop

You can reference to data associated with the outer loop from the inner loop.

## Example 2 - Output Data Based on Screenshot

Based on the locals code given to you, output data based on the following screenshot.

```
locals {  
    environments = ["dev", "prod"]  
    regions      = {  
        dev  = ["us-west-1", "us-east-1"]  
        prod = ["us-west-2", "us-east-2"]  
    }  
}
```



```
nested_loop = [  
    [  
        "dev-us-west-1",  
        "dev-us-east-1",  
    ],  
    [  
        "prod-us-west-2",  
        "prod-us-east-2",  
    ],  
]
```

# Final Solution - Exercise 3

```
locals {
    environments = ["dev", "prod"]
    regions      = {
        dev  = ["us-west-1", "us-east-1"]
        prod = ["us-west-2", "us-east-2"]
    }
}

nested_loop = [for env in local.environments : [for regions in local.regions[env] : "${env}-${regions}"]]
}

output "nested_loop" {
    value = local.nested_loop
}
```

## Example 3 - Using Conditional Expressions

You can also make use of conditional expressions within for expressions.

```
locals {  
    environments = ["dev", "prod"]  
    regions      = {  
        dev  = ["us-west-1", "us-east-1"]  
        prod = ["us-west-2", "us-east-2"]  
    }  
}
```

Base Code



```
prod_services = [  
    "prod-us-west-2",  
    "prod-us-east-2",  
]
```

Required Output

# **Terraform Function - Flatten and Distinct**

# Basics of Flatten Function

The **flatten** function in Terraform transforms a list of lists or a complex nested structure into a single flattened list.

```
> flatten([[1],[2,3],[4,5,6]])
[
  1,
  2,
  3,
  4,
  5,
  6,
]
```

## Example 1 - List with mixed data types

```
> flatten([[1, 2], ["a", "b"], [true, false]])  
[  
  1,  
  2,  
  "a",  
  "b",  
  true,  
  false,  
]
```

## Example 2 - List with mixed data types

```
locals {  
    instance_configs = [  
        {  
            name = "web"  
            tags = ["http", "https", "web"]  
        },  
        {  
            name = "app"  
            tags = ["application", "backend"]  
        }  
    ]  
  
    all_tags = flatten([for config in local.instance_configs : config.tags])  
}  
  
output "flatten" {  
    value = local.all_tags  
}
```



Outputs:

```
flatten = [  
    "http",  
    "https",  
    "web",  
    "application",  
    "backend",  
]
```

# Basics of Distinct Function

`distinct` takes a list and returns a new list with any duplicate elements removed.

```
> distinct(["a", "b", "a", "c", "d", "b"])
tolist([
  "a",
  "b",
  "c",
  "d",
])
```

# **Templates in Terraform**

# Setting the Base

We use ready-made base templates for various use cases and modify them based on our requirements.



# Understanding Challenge in Terraform

In Terraform, you might have a base configuration file that you might want to modify depending on the environment.

```
⚙ nginx.conf
server {
    listen      8080;
    server_name kplabs.internal;

    location / {
        proxy_pass http://10.0.10.56:8085;
    }
}
```

Base Nginx Configuration

**Production**

```
listen 443;
server_name kplabs.in;
```

**Development**

```
listen 80;
server_name kplabs.internal;
```

# Introducing Templating in Terraform

A template is a file that **contains placeholders** (variables) that you want to replace with dynamic values.

These placeholders are generally enclosed within \${}, which allows Terraform to insert the values during runtime.

```
nginx.conf
server {
    listen      8080;
    server_name kplabs.internal;

    location / {
        proxy_pass http://10.0.10.56:8085;
    }
}
```



```
nginx.tftpl
server {
    listen      ${nginx_port};
    server_name ${server_name};

    location / {
        proxy_pass http://${backend_ip}:${backend_port};
    }
}
```

Base Nginx Configuration

Template File

# Templatefile Function

The `templatefile` function in Terraform loads a template from a file and then replaces its placeholders with values from a map of variables.

```
templatefile(path, vars)
```



```
resource "local_file" "this" {
    content = templatefile("./nginx.tftpl", { key1="value1",key2="value2"})
    filename = "./nginx.conf"
}
```

# Naming Convention

\*`.tftpl` is the recommended naming pattern to use for your template files.

Terraform will not prevent you from using other names, but following this convention will help your editor understand the content and likely provide better editing experience as a result.

# **Templatefile - Dealing with List and Maps**

# Setting the Base

For many use-cases, the configurations defined in the base file is repetitive.

Example: /etc/resolv.conf

resolv.conf

```
nameserver 8.8.8.8  
nameserver 4.4.4.4  
nameserver 10.77.0.2
```

resolv.tftpl

```
nameserver ${dns_server_01}  
nameserver ${dns_server_02}  
nameserver ${dns_server_03}
```

# Using for expression

The **for loop** inside the `templatefile` function in Terraform can be a powerful tool for dynamically generating text or configurations based on iterating over a list or map of variables

resolv.tftpl

```
nameserver ${dns_server_01}  
nameserver ${dns_server_02}  
nameserver ${dns_server_03}
```

resolv.tftpl

```
%{ for dns in dns_ips ~}  
  nameserver ${dns}  
%{ endfor ~}
```

# Example Solution - Nameserver

The **for loop** inside the `templatefile` function in Terraform can be a powerful tool for dynamically generating text or configurations based on iterating over a list or map of variables

resolv.tftpl

```
%{ for dns in dns_ips ~}  
nameserver ${dns}  
%{ endfor ~}
```



```
resource "local_file" "dns" {  
content = templatefile("./resolv.tftpl", {dns_ips=[ "8.8.8.8", "4.4.4.4", "10.77.0.2" ]})  
filename = "./resolv.conf"  
}
```

# Final Output

⚙️ resolv.tftpl

```
%{ for dns in dns_ips ~}  
nameserver ${dns}  
%{ endfor ~}
```



```
resource "local_file" "dns" {  
    content = templatefile("./resolv.tftpl", {dns_ips=[ "8.8.8.8", "4.4.4.4", "10.77.0.2" ]})  
    filename = "./resolv.conf"  
}
```



⚙️ resolv.conf

```
nameserver 8.8.8.8  
nameserver 4.4.4.4  
nameserver 10.77.0.2
```

# Importance of Tilde

The `~` at the end suppresses the newline that would normally be inserted after this line.

Without the tilde, Terraform would add a newline after the `for` statement, which could result in unwanted extra space in the output.

resolv.conf

```
nameserver 8.8.8.8  
nameserver 4.4.4.4  
nameserver 10.77.0.2
```

With Tilde

resolv.conf

```
nameserver 8.8.8.8  
nameserver 4.4.4.4  
nameserver 10.77.0.2
```

Without Tilde

# Dealing with Maps

```
≡ map.tftpl
```

```
%{ for config_key, config_value in config }  
  set ${config_key} = ${config_value}  
%{ endfor ~}
```



```
resource "local_file" "map" {  
  content = templatefile("./map.tftpl", { config = {"x"="y", "foo"="bar", "key"="value" } } )  
  filename = "./map.conf"  
}
```



```
⚙ map.conf  
      set foo = bar  
  
      set key = value  
  
      set x = y
```

# **Exercise - Retrieve Values using for expression**

# Setting the Base

Sample Data has been provided to you in a JSON file. You need to work on various use-case to extract certain information from the file using for expression.

```
{} config.json > ...
{
  "servers": [
    {
      "name": "web01",
      "ip": "192.168.1.1",
      "roles": ["web", "app"],
      "metadata": {
        "owner": "teamA",
        "environment": "production",
        "dns": ["1.1.1.1"]
      }
    },
    {
      "name": "web02",
      "ip": "192.168.1.2",
      "roles": ["web"],
      "metadata": {
        "owner": "teamB",
        "environment": "development"
      }
    }
  ]
}
```

# Use-Case 1 - Extract All Information from File

Output the data that is part of the config.json file using for expression

```
C:\kplabs-terraform>terraform apply -auto-approve

Changes to Outputs:
+ all_data = [
    +
    + {
        + alerting    = true
        + logging     = false
        + monitoring  = true
        + security    = true
    },
    +
    + {
        + dns         = [
            + "8.8.8.8",
            + "8.8.4.4",
        ]
        + subnets     = [
            + "192.168.1.0/24",
            + "192.168.2.0/24",
        ]
    },
    +
    + [
        + {
            + ip          = "192.168.1.1"
            + metadata   = {
                + dns      = [
                    + "1.1.1.1",
                ]
            }
        }
    ]
]
```

# Use-Case 1 - Solution

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "all_data" {
    value = [for s in local.config : s]
}
```

## Use-Case 2 - Extract Name of Servers

Extract the name of all servers that are part of the JSON file.

```
{  
    "name": "web02",  
    "ip": "192.168.1.2",  
    "roles": ["web"],  
    "metadata": {  
        "owner": "teamB",  
        "environment": "development"  
    }  
,  
{  
    "name": "db01",  
    "ip": "192.168.1.10",  
    "roles": ["db"],  
    "metadata": {  
        "owner": "teamA",  
        "environment": "production"  
    }  
}
```



Outputs:

```
all_data = [  
    "web01",  
    "web02",  
    "db01",  
]
```

## Use-Case 2 - Solution

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "all_data" {
    value = [for s in local.config.servers : s.name]
}
```

# Use-Case 3 - Extract Name and IP information

Associate IP address associated with each server name from the file.

```
{  
    "name": "web02",  
    "ip": "192.168.1.2",  
    "roles": ["web"],  
    "metadata": {  
        "owner": "teamB",  
        "environment": "development"  
    }  
},  
{  
    "name": "db01",  
    "ip": "192.168.1.10",  
    "roles": ["db"],  
    "metadata": {  
        "owner": "teamA",  
        "environment": "production"  
    }  
}
```



Outputs:

```
server_ips = {  
    "db01" = "192.168.1.10"  
    "web01" = "192.168.1.1"  
    "web02" = "192.168.1.2"  
}
```

## Use-Case 3 - Solution

If you use { and }, the result is an object and you must provide two result expressions that are separated by the => symbol:

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "server_ips" {
    value = { for s in local.config.servers : s.name => s.ip}
}
```

# Use-Case 4 - Find Production IP addresses

Output all the IP addresses of production environment servers.

```
{  
  "name": "web02",  
  "ip": "192.168.1.2",  
  "roles": ["web"],  
  "metadata": {  
    "owner": "teamB",  
    "environment": "development"  
  }  
,  
{  
  "name": "db01",  
  "ip": "192.168.1.10",  
  "roles": ["db"],  
  "metadata": {  
    "owner": "teamA",  
    "environment": "production"  
  }  
}
```



Outputs:

```
production_server_ips = [  
  "192.168.1.1",  
  "192.168.1.10",  
]
```

# Use-Case 4 - Solution

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "production_server_ips" {
    value = [for s in local.config.servers : s.ip if s.metadata.environment == "production"]
}
```

# Use-Case 5 - Extract Roles Data

Extract unique values across all the roles into a single flattened list.

```
{  
    "name": "web02",  
    "ip": "192.168.1.2",  
    "roles": ["web"],  
    "metadata": {  
        "owner": "teamB",  
        "environment": "development"  
    }  
,  
{  
    "name": "db01",  
    "ip": "192.168.1.10",  
    "roles": ["db"],  
    "metadata": {  
        "owner": "teamA",  
        "environment": "production"  
    }  
}
```



```
all_roles = tolist([  
    "web",  
    "app",  
    "db",  
])
```

# Use-Case 5 - Solution

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "all_roles" {
    value = distinct(flatten([for s in local.config.servers : s.roles]))
}
```

# Use-Case 6 - Extract Server Name and Environment

Extract server name with the server environment in ()

```
{  
  "name": "web02",  
  "ip": "192.168.1.2",  
  "roles": ["web"],  
  "metadata": {  
    "owner": "teamB",  
    "environment": "development"  
  }  
},  
{  
  "name": "db01",  
  "ip": "192.168.1.10",  
  "roles": ["db"],  
  "metadata": {  
    "owner": "teamA",  
    "environment": "production"  
  }  
}
```



Outputs:

```
server_env_list = [  
  "web01 (production)",  
  "web02 (development)",  
  "db01 (production)",  
]
```

# Use-Case 6 - Solution

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "server_env_list" {
    value = [for s in local.config.servers : "${s.name} (${s.metadata.environment})"]
}
```

# Use-Case 7 - Fetch IP along with Roles

Extract details of IP address of server along with its roles.

```
{  
  "name": "web02",  
  "ip": "192.168.1.2",  
  "roles": ["web"],  
  "metadata": {  
    "owner": "teamB",  
    "environment": "development"  
  }  
},  
{  
  "name": "db01",  
  "ip": "192.168.1.10",  
  "roles": ["db"],  
  "metadata": {  
    "owner": "teamA",  
    "environment": "production"  
  }  
}
```

Outputs:

```
server_ip_roles = [  
  "192.168.1.1 roles: web,app",  
  "192.168.1.2 roles: web",  
  "192.168.1.10 roles: db",  
]
```

# Use-Case 7 - Solution

```
locals {
    config = jsondecode(file("${path.module}/config.json"))
}

output "server_ip_roles" {
    value = [for s in local.config.servers : "${s.ip} roles: ${join(", ", s.roles)}"]
}
```

# Relax and Have a Meme Before Proceeding



# **Exercise 1 - Create Resource Based on CSV File Data**

# Use-Case 1 - Create SG Rules Using CSV Data

Create Inbound and Outbound Security group rules dynamically based on the data of the CSV files using the following resource types:

`aws_vpc_security_group_ingress_rule` and `aws_vpc_security_group_egress_rule`

<b>name</b>	<b>direction</b>	<b>protocol</b>	<b>cidr_block</b>	<b>port</b>
rule-01	in	tcp	172.31.0.0/16	443
rule-02	in	tcp	172.31.0.0/16	80
rule-03	out	tcp	10.66.0.0/16	8443
rule-04	out	tcp	10.77.0.0/16	3306

## **Exercise 2 - Create Resource Based on CSV File Data**

## Use-Case 2 - Create SG Rules Using CSV Data

Create Inbound and Outbound Security group rules dynamically based on the data of the CSV files using the following resource types:

`aws_vpc_security_group_ingress_rule` and `aws_vpc_security_group_egress_rule`

<b>name</b>	<b>direction</b>	<b>protocol</b>	<b>cidr_block</b>	<b>port</b>
rule-01	in	tcp	172.31.0.0/16	443
rule-01	in	tcp	172.31.0.0/16	80
rule-02	out	tcp	10.66.0.0/16	8443
rule-02	out	tcp	10.77.0.0/16	3306

# Dealing with Situation with No Unique Value

If there are **no unique values** to iterate through as part of `for_each`, you can use `index` for iteration.

```
locals {
    csv_data = csvdecode(file("./sg-02.csv"))
    inbound_rules = [ for rule in local.csv_data : rule if rule.direction == "in"]
    outbound_rules = [ for rule in local.csv_data : rule if rule.direction == "out"]

    for_each = { for idx, rule in local.inbound_rules : idx => rule }
}
```

```
Changes to Outputs:
+ inbound_rules = [
+   "0" = {
+     + cidr_block = "172.31.0.0/16"
+     + direction  = "in"
+     + name       = "rule-01"
+     + port       = "443"
+     + protocol   = "tcp"
+   }
+   "1" = {
+     + cidr_block = "172.31.0.0/16"
+     + direction  = "in"
+     + name       = "rule-01"
+     + port       = "80"
+     + protocol   = "tcp"
+   }
]
```

## **Exercise 3 - Create Resource Based on CSV File Data**

# Use-Case - Create SG Rules Using CSV Data

Create Inbound Security group rule dynamically based on the data of the CSV files using the following resource types:

`aws_vpc_security_group_ingress_rule`

A	B	C	D	E
<b>name</b>	<b>direction</b>	<b>protocol</b>	<b>cidr_block</b>	<b>port</b>
rule-01	in	tcp	172.31.0.0/16	443-445
rule-01	in	tcp	196.18.10.50/32	80-100
rule-01	in	tcp	10.77.0.35/32	8443

## **Exercise 4 - Create Resource Based on CSV and JSON File Data**

# Use-Case - Create SG Rules Using CSV and Json Data

Create Inbound Security group rule dynamically based on the data of the CSV and JSON files using the following resource types:

aws\_vpc\_security\_group\_ingress\_rule

<b>name</b>	<b>direction</b>	<b>protocol</b>	<b>cidr_block</b>	<b>port</b>
rule-01	in	tcp	app-1	80
rule-01	in	tcp	app-2	443

```
app.json > ...
{
    "app-1": "10.77.0.0/16",
    "app-2": "172.16.0.0/24"
}
```

## Part 2 - Little Different Change

<b>name</b>	<b>direction</b>	<b>protocol</b>	<b>cidr_block</b>	<b>port</b>
rule-01	in	tcp	app-1	80
rule-01	in	tcp	app-2	443

```
app.json > ...
{
    "app": "10.77.0.0/16",
    "database": "172.16.0.0/24"
}
```

# Terraform Settings

# Setting the Base

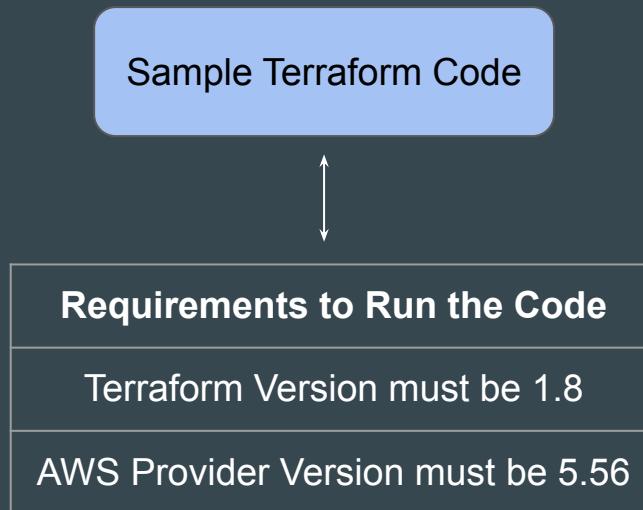
We can use the provider block to define various aspects of the provider, like region, credentials and so on.

```
provider "aws" {
    region      = "us-east-1"
    access_key  = "AKIAIOSFODNN7EXAMPLE"
    secret_key  = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
}

resource "aws_security_group" "sg_01" {
    name = "app_firewall"
}
```

# Specific Version to Run Your Code

In a Terraform project, your code might require a very specific set of versions to run.



# Introducing Terraform Settings

Terraform Settings are used to configure project-specific Terraform behaviors, such as requiring a minimum Terraform version to apply to your configuration.

Terraform settings are gathered together into **terraform blocks**:

```
terraform {  
    # <setting-1>  
    # <setting-2>  
}
```

# 1 - Specifying a Required Terraform Version

If your code is compatible with specific versions of Terraform, you can use the **required\_version** block to add your constraints.

```
terraform {  
    required_version = "1.8"  
}
```

## 2 - Specifying Provider Requirements

The `required_providers` block can be used to specify all of the providers required by your Terraform code.

You can further fine-tune to include a specific version of the provider plugins.

```
terraform {
  required_providers {
    aws = {
      version = "5.56"
      source = "hashicorp/aws"
    }
  }
}
```

# Flexibility in Settings Block

There are a wide variety of options that can be specified in the Terraform block.

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



Options That Can be Defined
Required Terraform Version
Required Provider and Version
BackEnd Configuration
Experimental Features

## Point to Note

It is a good practice to include the `terraform { }` block to include details like `required_providers` as part of your project.

The `provider { }` block is still important to specify various other aspects like regions, credentials, alias and others.

---

# Terraform Backend

Terraform in detail

---

# 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.

```
provider "vault" {
  address = "http://127.0.0.1:8200"
}

data "vault_generic_secret" "demo" {
  path = "secret/db_creds"
}

output "vault_secrets" {
  value = data.vault_generic_secret.demo.data_json
  sensitive = "true"
}
```

demo.tf



```
terraform.tfstate
1  {
2    "version": 4,
3    "terraform_version": "1.1.9",
4    "serial": 1,
5    "lineage": "f7ba581a-ab47-b03e-2e54-e683a2dc4ba2",
6    "outputs": {
7      "vault_secrets": {
8        "value": "{\"admin\":\"password123\"}",
9        "type": "string",
10       "sensitive": true
11     }
12   },
13   "resources": [
14     {
15       "mode": "data",
16       "type": "vault_generic_secret",
17       "name": "demo",
18       "provider": "provider[\"registry.terraform.io/hashicorp/vault\"]",
19       "instances": [

```

terrafrom.tfstate

# Challenge with Local Backend

Nowadays Terraform project is handled and collaborated by an entire team.

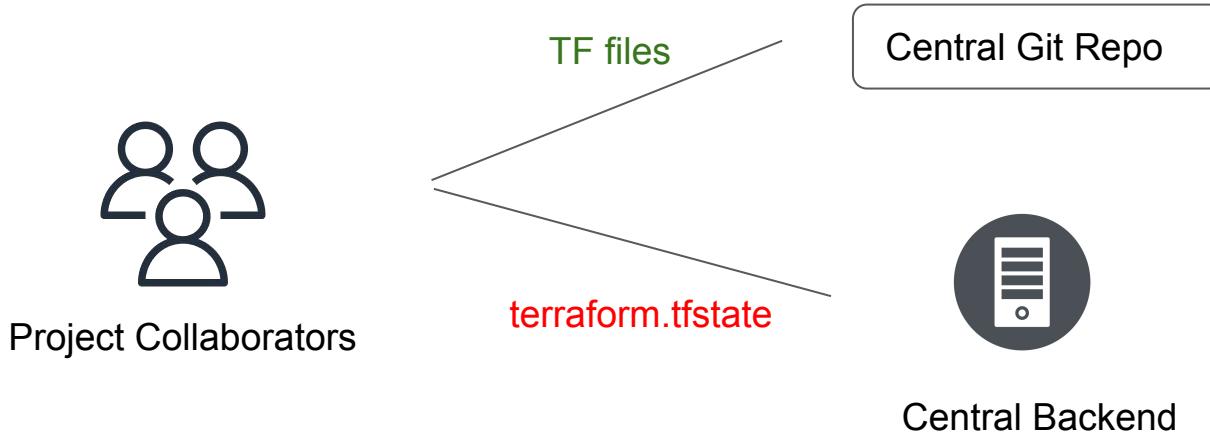
Storing the state file in the local laptop will not allow collaboration.



# Ideal Architecture

Following describes one of the recommended architectures:

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



# Backends Supported in Terraform

Terraform supports multiple backends that allows remote service related operations.

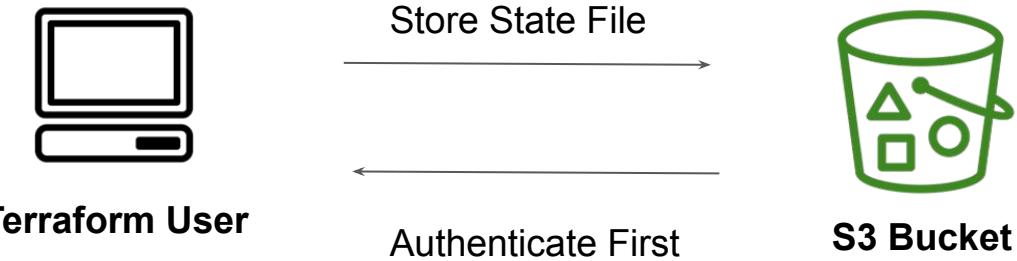
Some of the popular backends include:

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

# 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.



---

# State Locking

Let's Lock the State

---

# 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.

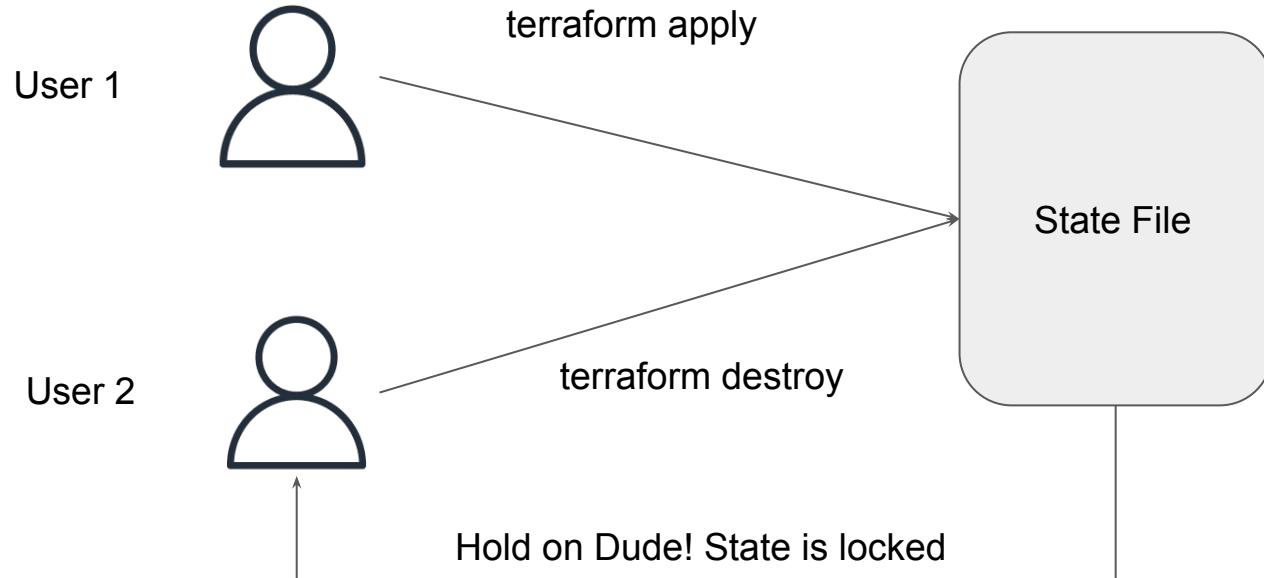
```
C:\Users\Zeal Vora\Desktop\tf-demo\remote-backend>terraform plan
```

```
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.
```

# Basic Working



# 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.

# 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.

---

# State Locking in S3 Backend

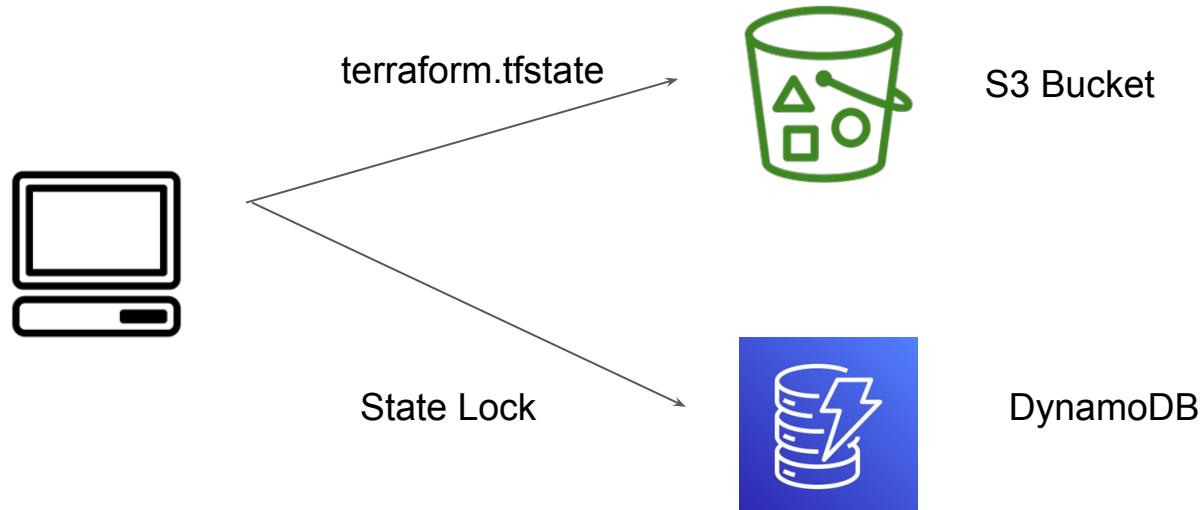
Back to Providers

---

# State Locking in S3

By default, S3 does not support State Locking functionality.

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



# **Terraform State Management**

# Setting the Base

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

It is **NOT** recommended to modify the state file manually.



# State Management

The **terraform state** command is used for advanced state management

Sub-Commands	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.
replace-provider	Used to replace the provider for resources in a Terraform state.

## Sub-Command 1 - List

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

Useful if you want to quickly view all resources managed by Terraform.

```
C:\kplabs-terraform>terraform state list  
aws_iam_user.dev  
aws_security_group.dev
```

## Sub-Command 2 - Show

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

Useful for debugging and understanding the current attributes of a resource.

```
C:\kplabs-terraform>terraform state show aws_iam_user.dev
# aws_iam_user.dev:
resource "aws_iam_user" "dev" {
    arn                  = "arn:aws:iam::042025557788:user/kplabs-user-01"
    force_destroy        = false
    id                  = "kplabs-user-01"
    name                = "kplabs-user-01"
    path                = "/"
    permissions_boundary = null
    tags                = {}
    tags_all            = {}
    unique_id           = "AIDAQTSFLD4OALWEWB2AW"
}
```

# Sub-Command 3 - pull

The **terraform state pull** command is used to pull the state from a remote backend and output it to stdout.

Useful to view or backup the current state stored in a remote backend.

```
C:\kplabs-terraform>terraform state pull
{
    "version": 4,
    "terraform_version": "1.9.1",
    "serial": 6,
    "lineage": "78f3ac79-1cb9-e724-964b-37bb3dc649a8",
    "outputs": {},
    "resources": [
        {
            "mode": "managed",
            "type": "aws_iam_user",
            "name": "dev",
            "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
            "instances": [
                {
                    "schema_version": 0,
                    "attributes": {
                        "arn": "arn:aws:iam::042025557788:user/kplabs-user-01",
                        "force_destroy": false,
                        "id": "kplabs-user-01",
                        "name": "kplabs-user-01",
                        "path": "/",
                        "permissions_boundary": ""
                    }
                }
            ]
        }
    ]
}
```

## Sub-Command 4 - rm

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

Use this when you need to remove a resource from Terraform's state management without destroying it.

```
C:\kplabs-terraform>terraform state rm aws_security_group.dev
Removed aws_security_group.dev
Successfully removed 1 resource instance(s).
```

## Sub-Command 5 - mv

The **terraform state mv** command is used to move an item in the state to a different address.

```
C:\kplabs-terraform>terraform state mv aws_security_group.dev aws_security_group.prod
Move "aws_security_group.dev" to "aws_security_group.prod"
Successfully moved 1 object(s).
```

# Sub-Command 6 - replace-provider

The `terraform state replace-provider` command is used to replace the provider for resources in a Terraform state.

```
C:\kplabs-terraform>terraform state replace-provider hashicorp/local kplabs.in/internal/local
Terraform will perform the following actions:

  ~ Updating provider:
    - registry.terraform.io/hashicorp/local
    + kplabs.in/internal/local

Changing 1 resources:

  local_file.foo

Do you want to make these changes?
Only 'yes' will be accepted to continue.

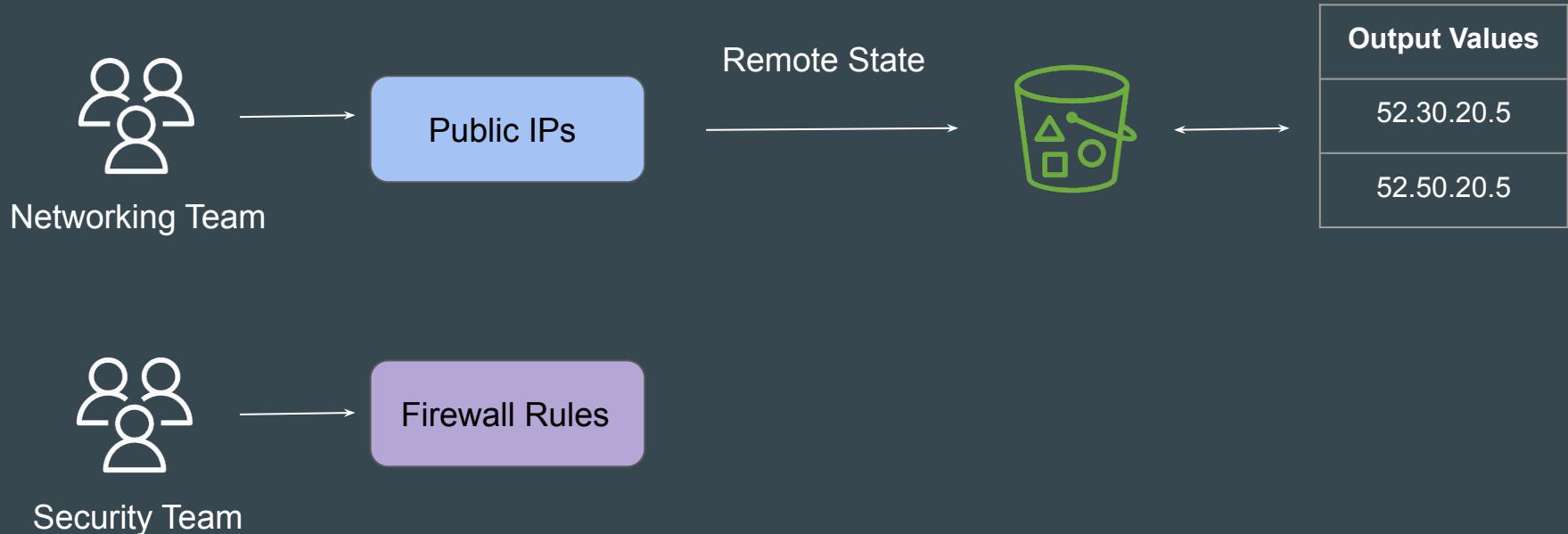
Enter a value: yes

Successfully replaced provider for 1 resources.
```

# **Remote State Data Source**

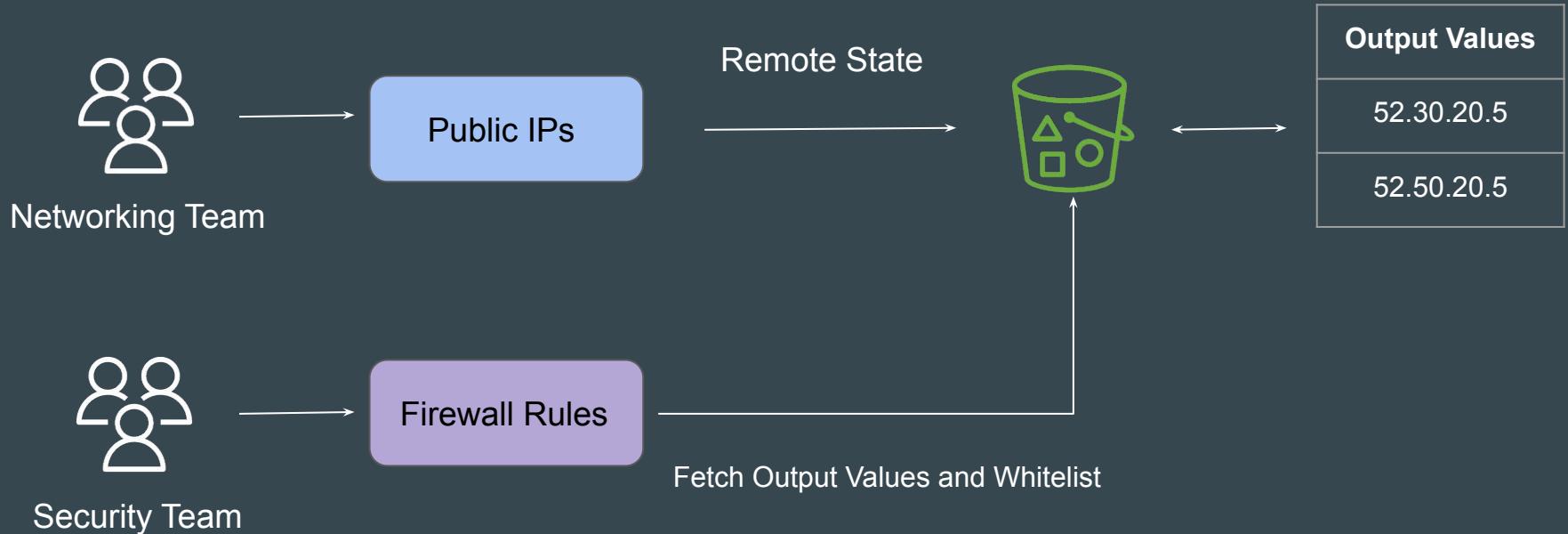
# Setting up the Base

In larger enterprises, there can be multiple different teams working on different aspects of a infrastructure resource



# Understanding the Challenge

Security Team wants that all the IP addresses added as part of Output Values in tfstate file of Networking Team project should be whitelisted in Firewall.



# What Needs to be Achieved

1. The code from Security Team project should connect to the `terraform.tfstate` file managed by the Networking team.
2. The code should fetch all the IP addresses mentioned in the output values in the state file.
3. The code should whitelist these IP addresses in Firewall rules.

# Practical Workflow Steps

1. Create two folders for networking-team and security-team
2. Create Elastic IP resource in Networking Team and Store the State file in S3 bucket. Output values should have information of EIP.
3. In Security Team, use Terraform Remote State data source to connect to the tfstate file of Networking Team.
4. Use the Remote State to fetch EIP and whitelist it in Security Group rule.

# Introducing Remote State Data Source

The `terraform_remote_state` data source allows us to fetch output values from a specific state backend

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

```
resource "aws_vpc_security_group_ingress_rule" "allow_tls_ipv4" {  
  security_group_id = aws_security_group.allow_tls.id  
  cidr_ipv4        = "${data.terraform_remote_state.eip.outputs.eip_addr}/32"  
  from_port         = 443  
  ip_protocol       = "tcp"  
  to_port           = 443  
}
```

Step 1 - Define Remote State Source

Step 2 - Define Data to Fetch

# **Terraform Testing Framework**

# Relatable Scenario

A few years back, I went on a jungle safari.

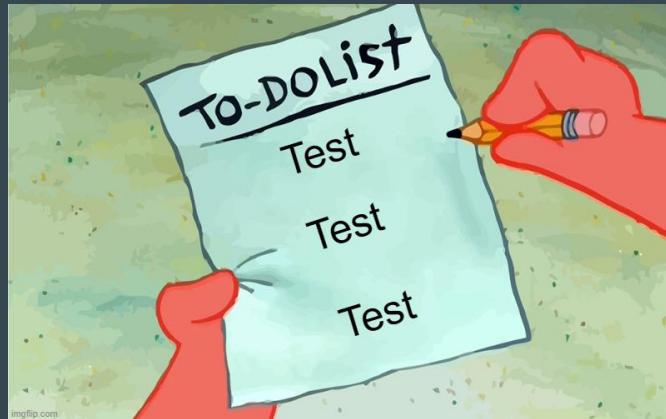
Before the journey started, I was testing whether window grills are loose if we shake, whether doors are opening even if locked, and other possible scenarios.



# Testing in Terraform

Whenever you are writing Terraform code, it is important to test the code thoroughly before it is committed to the Git repository.

Tests can range from smallest unit like function to entire unit using `terraform apply`.



# Use-Case: S3 Bucket Terraform Code

A user in your organization has written a simple Terraform code that creates a S3 bucket.

Question: Is this code valid and if so, will it work all the time?

```
variable "s3_bucket_name" {}

resource "aws_s3_bucket" "example" {
    bucket = var.s3_bucket_name
}
```

# Lot of IFs factor

There are many restrictions, like Bucket Naming rules, that variable values must adhere to before a bucket can be created.

## General purpose buckets naming rules

The following naming rules apply for general purpose buckets.

- Bucket names must be between 3 (min) and 63 (max) characters long.
- Bucket names can consist only of lowercase letters, numbers, dots (.), and hyphens (-).
- Bucket names must begin and end with a letter or number.
- Bucket names must not contain two adjacent periods.
- Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
- Bucket names must not start with the prefix `xn--`.
- Bucket names must not start with the prefix `sthree-`.
- Bucket names must not start with the prefix `sthree-configurator`.
- Bucket names must not start with the prefix `amzn-s3-demo-`.
- Bucket names must not end with the suffix `-s3alias`. This suffix is reserved for access point alias names. For more information, see [Using a bucket-style alias for your S3 bucket access point](#).
- Bucket names must not end with the suffix `--ol-s3`. This suffix is reserved for Object Lambda Access Point alias names. For more information, see [How to use a bucket-style alias for your S3 bucket Object Lambda Access Point](#).
- Bucket names must not end with the suffix `.mrap`. This suffix is reserved for Multi-Region Access Point names. For more information, see [Rules for naming Amazon S3 Multi-Region Access Points](#).
- Bucket names must not end with the suffix `--x-s3`. This suffix is reserved for directory buckets. For more information, see [Directory bucket naming rules](#).
- Bucket names must be unique across all AWS accounts in all the AWS Regions within a partition. A partition is a grouping of Regions. AWS currently has three partitions: `aws` (Standard Regions), `aws-cn` (China Regions), and `aws-us-gov` (AWS GovCloud (US)).

# Introducing Terraform Test

Terraform tests let authors validate that configurations do not introduce breaking changes.

Tests
Verify if Length > 3
Verify if Length < 63
Verify if name does not start with xn--
Verify if name does not start with sthree-

Testing Code



```
variable "s3_bucket_name" {  
  default = "sample kplabs-bucket"  
}  
  
resource "aws_s3_bucket" "example" {  
  bucket = var.s3_bucket_name  
}
```

# Results Terraform Tests

The **terraform test** command can test the code based on the tests defined and can provide you with pass/failed results.

```
variable "s3_bucket_name" {  
    default = "sample kplabs-bucket"  
}  
  
resource "aws_s3_bucket" "example" {  
    bucket = var.s3_bucket_name  
}
```



```
C:\kplabs-terraform>terraform test  
tests\s3-names.tftest.hcl... in progress  
  run "verify_character_limit"... pass  
  run "verify_invalid_name_xn"... pass  
  run "verify_invalid_name_sthree"... pass  
tests\s3-names.tftest.hcl... tearing down  
tests\s3-names.tftest.hcl... pass  
  
Success! 3 passed, 0 failed.
```

# Failed Result of Terraform Test

Following screenshot shows where 1 test has failed and 2 have passed.

```
C:\kplabs-terraform>terraform test
tests\s3-names.tftest.hcl... in progress
  run "verify_character_limit"... fail

  Error: Test assertion failed

  on tests\s3-names.tftest.hcl line 7, in run "verify_character_limit":
  7:   condition      = length(var.s3_bucket_name) > 3 && length(var.s3_bucket_name) < 63
    |
    | var.s3_bucket_name is "hi"

  Bucket names must be between 3 (min) and 63 (max) characters long

  run "verify_invalid_name_xn"... pass
  run "verify_invalid_name_sthree"... pass
tests\s3-names.tftest.hcl... tearing down
tests\s3-names.tftest.hcl... fail

Failure! 2 passed, 1 failed.
```

# Testing Scope

Many a times, your resource might pass all the test, but when you try to deploy it in production, you might get an error.

**terraform test**, can validate test at a **plan stage** and **apply stage**.

```
resource "aws_instance" "myec2" {  
    ami = "ami-00c39f71452c08778"  
    instance_type = "t2.micro"  
  
    tags = {  
        Name = "HelloWorld"  
    }  
}
```



# Apply Stage Workflow

1. Run “terraform apply” to launch the resource defined
2. If everything works, destroy the resource that were created.

Note: Terraform Test uses and manages state file within memory.

```
C:\kplabs-terraform>terraform test
tests\ec2.tftest.hcl... in progress
  run "verify_ec2_creation"... pass
tests\ec2.tftest.hcl... tearing down
tests\ec2.tftest.hcl... pass

Success! 1 passed, 0 failed.
```



Instances (1) <a href="#">Info</a>				
Last updated <span style="border: 1px solid #ccc; padding: 2px;">3 minutes ago</span> <a href="#">G</a> <a href="#">Connect</a>				
<input type="text"/> Find Instance by attribute or tag (case-sensitive)				
<input type="button" value="Clear filters"/>				
	Name ↴	Instance ID	Instance state	Instance type
<input type="checkbox"/>	HelloWorld	i-09fed888645421168	Terminated	t2.micro

# Point to Note

Terraform testing framework is available in Terraform v1.6.0 and later.

# **Terraform Tests - Practical**

# Revising Terraform Tests

Terraform tests let authors validate that configuration do not introduce breaking changes.

Tests
Verify if Length > 3
Verify if Length < 63
Verify if name does not start with xn--
Verify if name does not start with sthree-

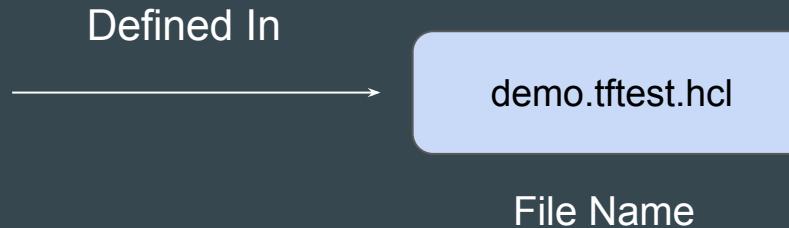
Testing Code

```
variable "s3_bucket_name" {  
  default = "sample kplabs-bucket"  
}  
  
resource "aws_s3_bucket" "example" {  
  bucket = var.s3_bucket_name  
}
```

# Creating Terraform Tests File

Each Terraform test lives in a test file. Terraform discovers test files are based on their file extension: `.tftest.hcl` or `.tftest.json`

Tests
Verify if Length > 3
Verify if Length < 63
Verify if name does not start with xn--
Verify if name does not start with sthree-



# Inside a Test File

Each test file contains one to many `run` blocks

Whenever you have defined a simple run block, it will deploy resource defined in tf file to verify the success or failure.

```
resource "aws_security_group" "allow_tls" {  
    name          = "demo-firewall"  
}
```

sg.tf

≡ demo.tftest.hcl

```
run "test_run" {}
```

demo.tftest.hcl

# Running Terraform tests

You can use the `terraform test` command to run the tests.

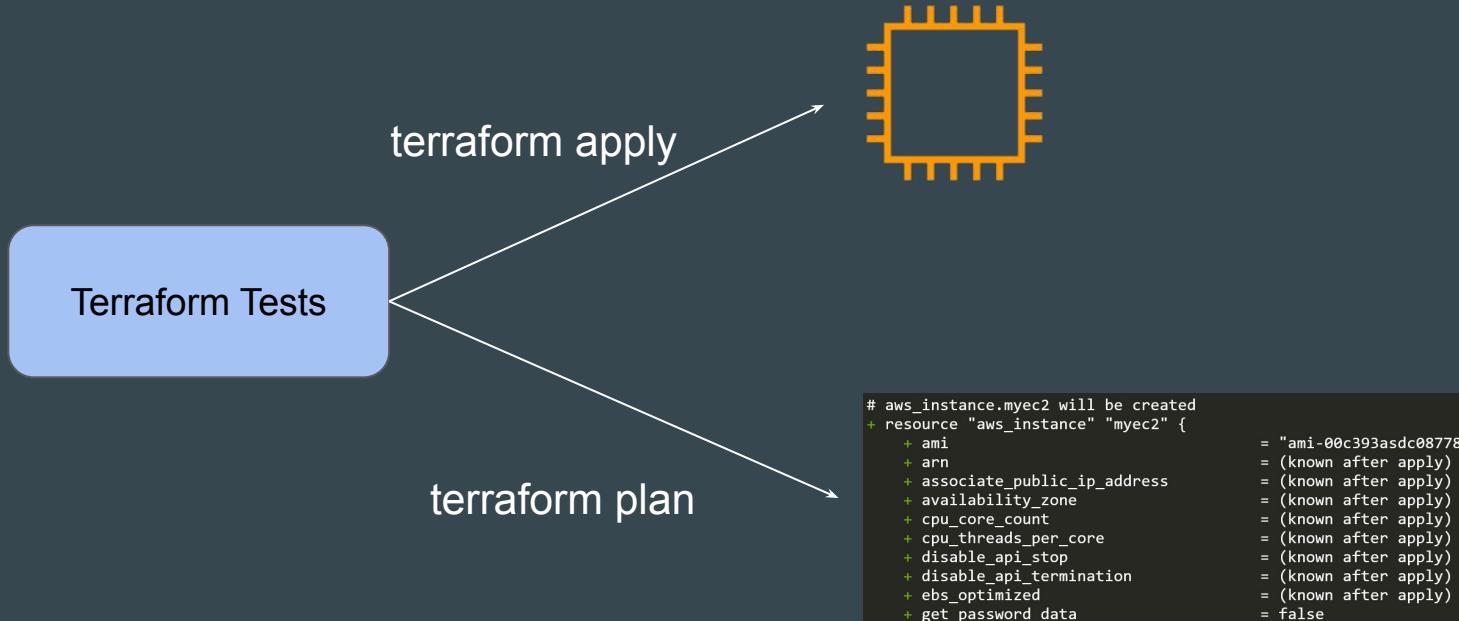
The output of test will be a pass / fail result.

```
C:\kplabs-terraform>terraform test
demo.tftest.hcl... in progress
  run "test_run"... pass
demo.tftest.hcl... tearing down
demo.tftest.hcl... pass

Success! 1 passed, 0 failed.
```

# Test Command

Terraform Test can either run based on plan stage or apply stage.



# Plan Command in RUN

By default, if you do not specify any command, Terraform Test will deploy the resource (terraform apply)

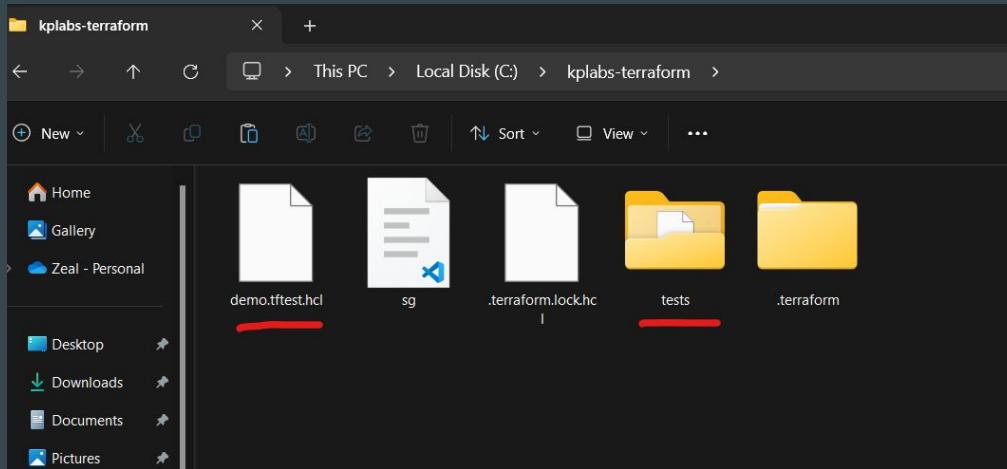
By specifying command = plan, Tests will simply run Plan to verify the workflow.

```
run "test_run" {
    command = plan
}
```

# Location for Tests File

Test files can be defined within the root folder.

Terraform can also automatically pickup tests file defined in the `tests/` directory.



# Relax and Have a Meme Before Proceeding



# **Terraform Tests - Assertions**

# Setting the Base

A basic set of run block allow users to execute the apply operation or the plan operation on resources defined in the configuration.

≡ demo.tfstate.hcl

```
run "test_run" {}
```

≡ demo.tfstate.hcl

```
run "test_run" {
    command = plan
}
```

# More Flexibility Needed

We would need better flexibility in terms of what is tested as part of the code to achieve a wide variety of use-cases.

Tests
Verify if Length > 3
Verify if Length < 63
Verify if name does not start with xn--
Verify if name does not start with sthree-

Testing Code

```
variable "s3_bucket_name" {
  default = "sample-kplabs-bucket"
}

resource "aws_s3_bucket" "example" {
  bucket = var.s3_bucket_name
}
```

# Assertions in Run Block

Terraform run block assertions are Custom Conditions, consisting of a **condition** and an **error message**.

```
run "test_run" {
    command = plan

    assert {
        condition      = length(var.s3_bucket_name) > 3
        error_message = "S3 bucket name must be min 3 characters"
    }
}
```

# **Terraform Tests - Root Level Attributes**

# Setting the Base

Each Terraform test lives in a test file based on their file extension: `.tfest.hcl` or `.tfest.json`.

Each test file contains the following root-level attributes and blocks:

- One to many run blocks.
- Zero to one variables block.
- Zero to many provider blocks.

```
≡ demo.tfest.hcl

provider "aws" {
    region = "ap-south-1"
}

variables {
    firewall_name = "test-firewall"
}

run "firewall_test" {}
```

# 1 - One to Many Run Blocks

Single or multiple set of run blocks can be specified in a single file or multiple sets of files.

Terraform test will run each block for verification.

```
run "bucket_min_char" {
    command = plan

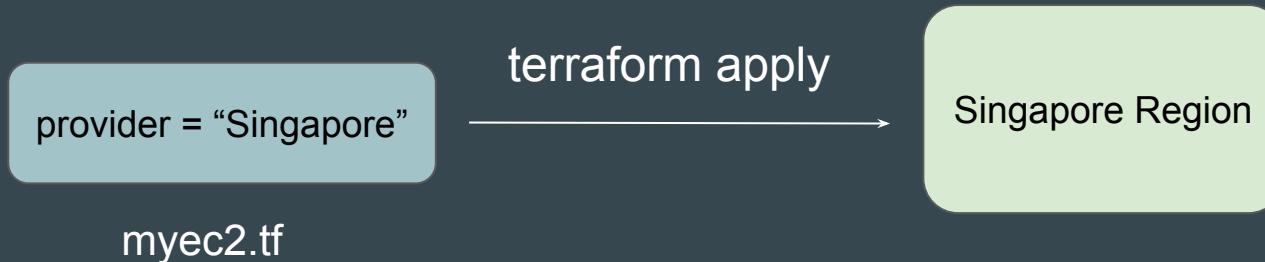
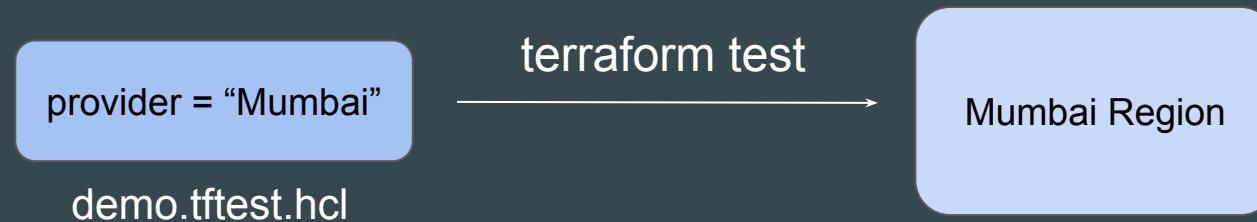
    assert {
        condition      = length(var.s3_bucket_name) > 3
        error_message = "S3 bucket name must be min 3 characters"
    }
}

run "bucket_max_char" {
    command = plan

    assert {
        condition      = length(var.s3_bucket_name) < 63
        error_message = "S3 bucket name must be max of 63 characters"
    }
}
```

## 2 - Provider Block

You can set or override the required providers within the main configuration from your testing files by using provider block.



## 3 - Variables Block

You can provide values for Input Variables within your configuration directly from your test files.

Terraform passes all variable values from the test file into all run blocks within the file.

```
variable "firewall_name" {  
  default = "demo-firewall"  
}  
  
resource "aws_security_group" "allow_tls" {  
  name        = var.firewall_name  
}
```

Main Config File

```
≡ demo.tfstate.hcl  
  
variables {  
  firewall_name = "test-firewall"  
}  
  
run "firewall_test" {}
```

Test File

## Point to Note

If you do not provide provider configuration within a testing file, Terraform attempts to initialize any providers within its configuration using the provider's default settings.

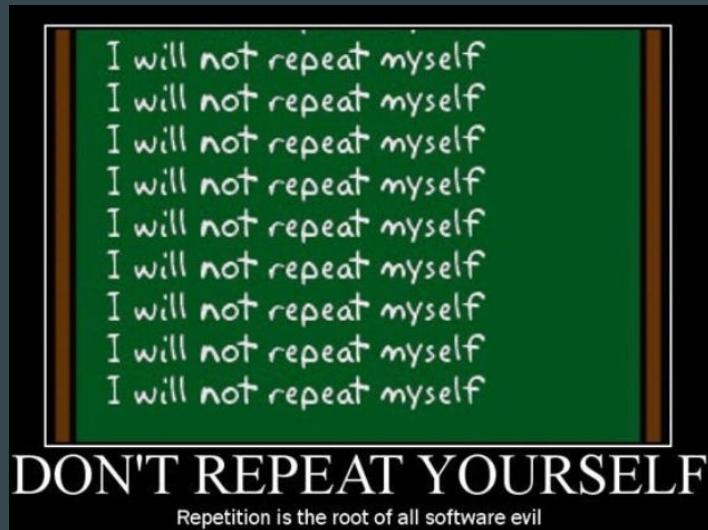
Input variables are referred as `variables {}` inside test file and not `variable {}`

The `variables` defined in test files take the highest precedence, overriding environment variables, variables files, or command-line input.

# Terraform Modules

# Understanding the Basic

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



# Understanding the Challenge

Let's assume there are 10 teams in your organization using Terraform to create and manage EC2 instances.

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
}
```

Team1

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
}
```

Team 2

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
}
```

Team 3

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
}
```

Team 4

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
}
```

Team 5

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
}
```

Team 6

# Challenge with the Previous Example

1. Repetition of Code.
2. Change in AWS Provider specific option will require change in EC2 code blocks of all the teams.
3. Lack of standardization.
4. Difficult to manage.
5. Difficult for developers to use.

# Better Approach

In this approach, the DevOps Team has defined standard EC2 template in a central location that all can use.

```
resource "aws_instance" "web" {  
    ami           = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring    = true  
    vpc_security_group_ids = ["sg-12345678"]  
    associate_public_ip_address = true  
    instance_initiated_shutdown_behavior = "stop"  
    ebs_optimized = true  
    source_dest_check = false  
    hibernation = true  
    disable_api_termination = true  
}
```

Standard Template

Team 1 Code

Team 2 Code

Team 3 Code

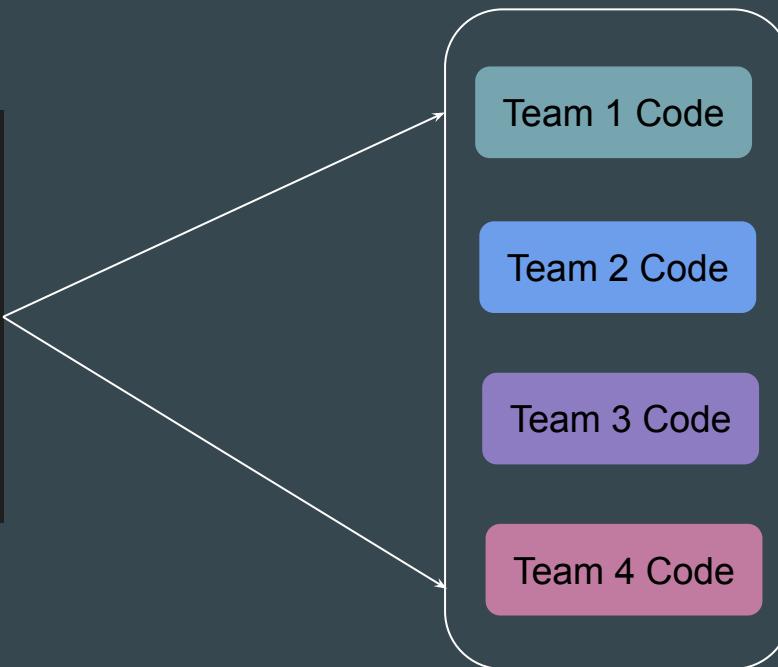
Team 4 Code

# Introducing Terraform Modules

Terraform Modules allows us to centralize the resource configuration and it makes it easier for multiple projects to re-use the Terraform code for projects.

```
resource "aws_instance" "web" {  
    ami          = "ami-1234"  
    instance_type = "t3.micro"  
    key_name      = "user1"  
    monitoring     = true  
    vpc_security_group_ids = ["sg-12345678"]  
    associate_public_ip_address = true  
    instance_initiated_shutdown_behavior = "stop"  
    ebs_optimized = true  
    source_dest_check = false  
    hibernation = true  
    disable_api_termination = true  
}
```

Terraform Module



# Multiple Modules for a Single Project

Instead of writing code from scratch, we can use multiple ready-made modules available.

```
resource "aws_instance" "web" {
    ami           = "ami-1234"
    instance_type = "t3.micro"
    key_name      = "user1"
    monitoring    = true
    vpc_security_group_ids = ["sg-12345678"]
    associate_public_ip_address = true
    instance_initiated_shutdown_behavior = "stop"
    ebs_optimized = true
    source_dest_check = false
    hibernation = true
    disable_api_termination = true
}
```

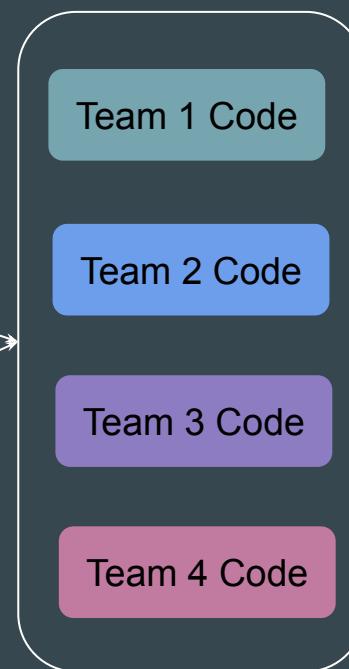
EC2 Module

```
resource "aws_vpc" "this" {
    count = local.create_vpc ? 1 : 0

    cidr_block      = var.use_ipam_pool ? null : var.cidr
    ipv4_ipam_pool_id = var.ipv4_ipam_pool_id
    ipv4_netmask_length = var.ipv4_netmask_length

    assign_generated_ipv6_cidr_block = var.enable_ipv6 && !var.use_ipam_pool ? true : null
    ipv6_cidr_block                = var.ipv6_cidr
    ipv6_ipam_pool_id              = var.ipv6_ipam_pool_id
    ipv6_netmask_length            = var.ipv6_netmask_length
    ipv6_cidr_block_network_border_group = var.ipv6_cidr_block_network_border_group
}
```

VPC Module



Infrastructure Created

# **Points to Note - Referencing Terraform Modules**

# Understanding the Base

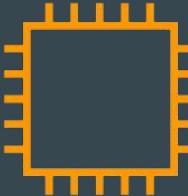
For some infrastructure resources, you can directly use the module calling code, and the entire infrastructure will be created for you.

```
VSCode ec2-module.tf > ...
```

```
module "ec2-instance" {  
  source  = "terraform-aws-modules/ec2-instance/aws"  
  version = "5.6.1"  
}
```

terra

form apply



# Avoiding Confusion

Just by referencing any module, it is not always the case that the infrastructure resource will be created for you directly.

Some modules require specific inputs and values from the user side to be filled in before a resource gets created.

# Example Module - AWS EKS

If you try to use an AWS EKS Module directly and run “terraform apply”, it will throw an **error**.

```
 eks-module.tf > ...
  module "eks" {
    source  = "terraform-aws-modules/eks/aws"
    version = "20.11.1"
  }
```

↓  
terraform apply

```
Plan: 21 to add, 0 to change, 0 to destroy.

Error: Error in function call

on .terraform\modules\eks\main.tf line 48, in resource "aws_eks_cluster" "this":
48:   subnet_ids          = coalescelist(var.control_plane_subnet_ids, var.subnet_ids)

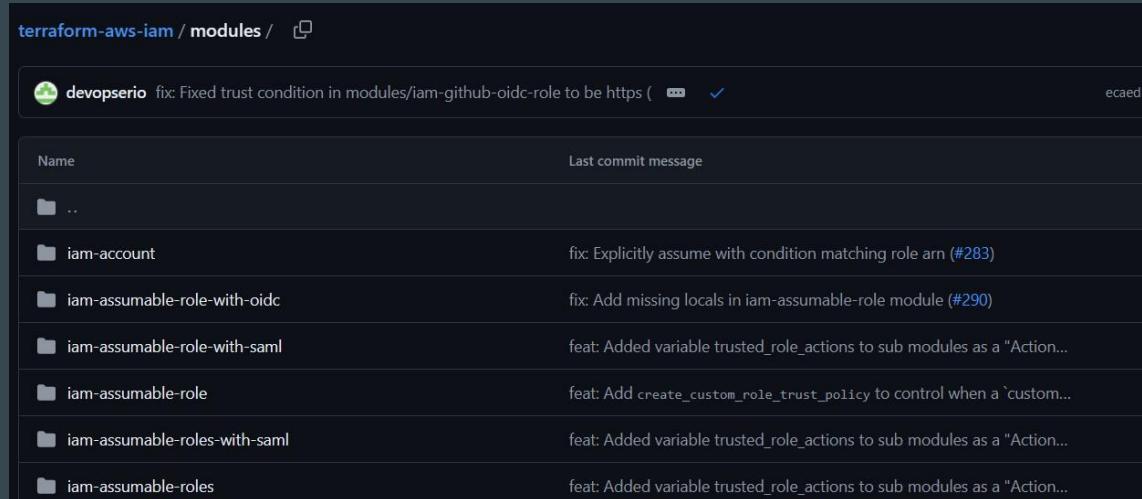
      while calling coalescelist(vals...)
      var.control_plane_subnet_ids is empty list of string
      var.subnet_ids is empty list of string

Call to function "coalescelist" failed: no non-null arguments.
```

# Module Structure Can be Different

Some module pages in GitHub can contain multiple sets of modules together for different features.

In such cases, you have to reference the exact sub-module required.



A screenshot of a GitHub repository page titled "terraform-aws-iam / modules". The page displays a list of several sub-modules under the "modules" directory. Each entry shows the folder name, the last commit message, and the commit author and date. The commits are:

Name	Last commit message
..	
iam-account	fix: Explicitly assume with condition matching role arn (#283)
iam-assumable-role-with-oidc	fix: Add missing locals in iam-assumable-role module (#290)
iam-assumable-role-with-saml	feat: Added variable trusted_role_actions to sub modules as a "Action..."
iam-assumable-role	feat: Add <code>create_custom_role_trust_policy</code> to control when a `custom...
iam-assumable-roles-with-saml	feat: Added variable trusted_role_actions to sub modules as a "Action..."
iam-assumable-roles	feat: Added variable trusted_role_actions to sub modules as a "Action..."

# Learnings for Today's Video

Always read the Module Documentation to understand the overall structure, important information, and what is expected from the user side when creating a resource.

# **Choosing the Right Terraform Module**

# Understanding the Base

Terraform Registry can contain multiple modules for a specific infrastructure resource maintained by different users

The screenshot shows the Terraform Registry interface. At the top, there is a search bar with the query 'ec2'. Below the search bar, there are navigation links: 'Browse', 'Publish', 'Sign-in', and a button to 'Use Terraform Cloud for free'. On the left side, there is a sidebar with filters for 'Providers' and 'Modules'. Under 'Providers', there is a checkbox for 'Partner' which is unchecked. Under 'Modules', there is a list of providers: Alibaba, Aws, Azure, Boundary, Consul, Google, Helm, Nomad, Oracle, and Vault. The main content area is titled 'Modules' and contains three results:

- boldlink / ec2**: Terraform registry repository for AWS EC2 module. 124.8K downloads. AWS provider.
- clouddrove / ec2**: Terraform module to create an EC2 resource on AWS with Elastic IP Addresses and Elastic Block Store. 13.8K downloads. AWS provider.
- kurron / ec2**: A Terraform plan that populates an existing VPC with subnets and EC2 instances. 5.3K downloads. AWS provider.

# 1 - Check Total Downloads

Module Downloads can provide early indication about level of acceptance by users in the Terraform community

The screenshot shows the Terraform Registry page for the `ec2-instance` module. The module is categorized under the `aws provider`. It is described as a Terraform module to create AWS EC2 instance(s) resources. The module was published on March 7, 2024, by `terraform-aws-modules`, managed by `antonbabenko`, and has a GitHub source code link.

A dropdown menu shows the current version is `5.6.1 (latest)`. To the right, a summary of module downloads is displayed:

Downloads this week	153,193
Downloads this month	476,217
Downloads this year	2.8M
Downloads over all time	14.4M

Below the download stats, there is a section titled **Provision Instructions** containing Terraform configuration code:

```
module "ec2-instance" {  
  source = "terraform-aws-modules/ec2-instance"  
  version = "5.6.1"  
}
```

## 2 - Check GitHub Page of Module

GitHub page can provide important information related to the Contributors, Reported Issues and other data.

The screenshot shows the GitHub repository page for the "aws-ec2-instance" Terraform module. The repository has 177 commits across 2 branches and 83 tags. The README file is the active tab. The page includes sections for About, Releases, Sponsor this project, Packages, and Contributors. A prominent yellow banner at the bottom encourages users to help Ukraine.

**About**  
Terraform module to create AWS EC2 instance(s) resources. [registry.terraform.io/modules/terraform-aws-ec2-instance/](#)

**Releases** 32  
[v5.6.1 \[Latest\]](#) on Mar 7  
+ 31 releases

**Sponsor this project**  
[antonbabenko](#) /anton Babenko <https://www.paypal.me/antonbabenko>

**Packages**  
No packages published

**Contributors** 42  
+ 28 contributors

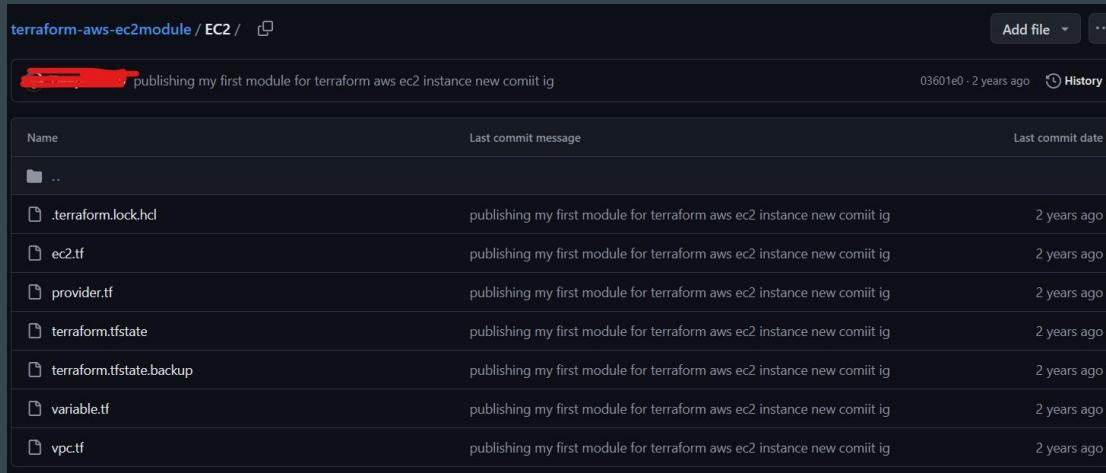
**AWS EC2 Instance Terraform module**  
Terraform module which creates an EC2 instance on AWS.

Russia invaded Ukraine, killing tens of thousands of civilians and displacing millions more. It's a genocide. Please help us defend freedom, democracy and Ukraine's right to exist.

Help Ukraine Now →

## 3 - Avoid Modules Written by Individual Participant

Avoid module that are maintained by a single contributor as regular updates, issues and other areas might not always be maintained.

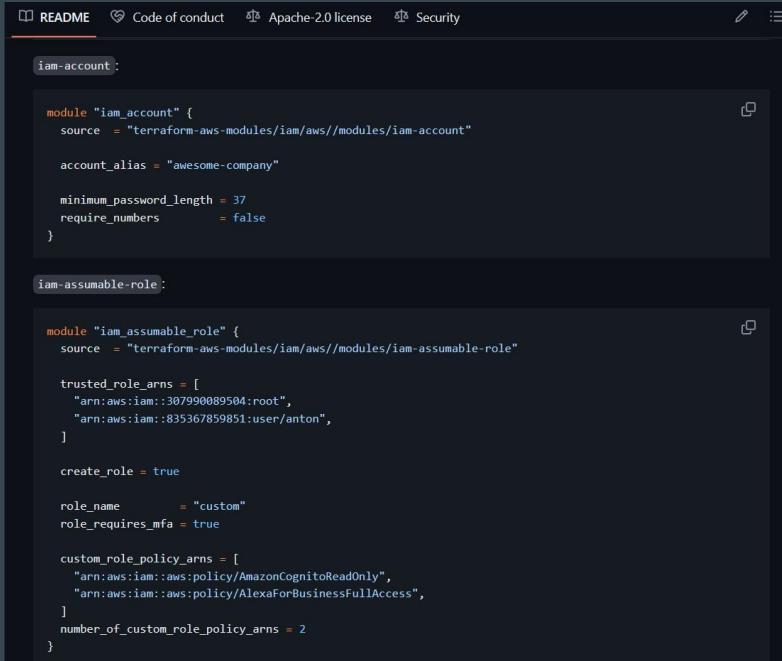


The screenshot shows a GitHub repository named "terraform-aws-ec2module / EC2". A single commit was made two years ago, publishing the first module for Terraform AWS EC2 instances. The commit message is "publishing my first module for terraform aws ec2 instance new comitt ig". The commit details show it was made by a user with the GitHub handle "03601e0" two years ago. The table lists the files affected by this commit:

Name	Last commit message	Last commit date
..		
.terraform.lock.hcl	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago
ec2.tf	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago
provider.tf	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago
terraform.tfstate	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago
terraform.tfstate.backup	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago
variable.tf	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago
vpc.tf	publishing my first module for terraform aws ec2 instance new comitt ig	2 years ago

# 4 - Analyze Module Documentation

Good documentation should include an overview, usage instructions, input and output variables, and examples.



The screenshot shows a GitHub README page with several tabs at the top: README (which is active), Code of conduct, Apache-2.0 license, and Security. Below the tabs, there are two code snippets. The first snippet is for the `iam-account` module:

```
module "iam_account" {
  source  = "terraform-aws-modules/iam/aws//modules/iam-account"
  account_alias = "awesome-company"
  minimum_password_length = 37
  require_numbers      = false
}
```

The second snippet is for the `iam-assumable-role` module:

```
module "iam_assumable_role" {
  source  = "terraform-aws-modules/iam/aws//modules/iam-assumable-role"
  trusted_role_arns = [
    "arn:aws:iam::307990089504:root",
    "arn:aws:iam::8355367859851:user/anton",
  ]
  create_role = true
  role_name      = "custom"
  role_requires_mfa = true
  custom_role_policy_arns = [
    "arn:aws:iam::aws:policy/AmazonCognitoReadOnly",
    "arn:aws:iam::aws:policy/AlexaForBusinessFullAccess",
  ]
  number_of_custom_role_policy_arns = 2
}
```

# 5 - Check Version History of Module

Look at the version history. Frequent updates and a clear versioning strategy suggest active maintenance.

The screenshot shows the Terraform Registry interface for the "iam" module. The top navigation bar includes links for "Browse", "Publish", "Sign-in", and "Use Terraform Cloud for free". The main content area displays the "iam" module details, which is an "aws provider" used to create AWS IAM resources. It was published on May 15, 2024, by "terraform-aws-modules" and managed by "antonbabenko". The source code is available on GitHub at [github.com/terraform-aws-modules/terraform-aws-iam](https://github.com/terraform-aws-modules/terraform-aws-iam). Below this, there are buttons for "Submodules" and "Examples". A dropdown menu titled "Version 5.39.1 (latest)" lists previous versions from 5.32.1 to 5.39.1. To the right, a "Module Downloads" section shows statistics for the current week, month, year, and all time. At the bottom, "Provision Instructions" provide a Terraform configuration snippet:

```
module "iam" {  
    source = "terraform-aws-modules/iam"  
    version = "5.39.1"  
}
```

# 6 - Analyze the Code

Inspect the module's source code on GitHub or another platform. Clean, well-structured code is a good sign.

```
resource "aws_instance" "this" {
    count = local.create && !var.ignore_ami_changes && !var.create_spot_instance ? 1 : 0

    ami              = local.ami
    instance_type    = var.instance_type
    cpu_core_count   = var.cpu_core_count
    cpu_threads_per_core = var.cpu_threads_per_core
    hibernation      = var.hibernation

    user_data         = var.user_data
    user_data_base64 = var.user_data_base64
    user_data_replace_on_change = var.user_data_replace_on_change

    availability_zone = var.availability_zone
    subnet_id         = var.subnet_id
    vpc_security_group_ids = var.vpc_security_group_ids

    key_name          = var.key_name
    monitoring        = var.monitoring
    get_password_data = var.get_password_data
    iam_instance_profile = var.create_iam_instance_profile ? aws_iam_instance_profile.this[0].name : var.iam_instance_profile
```

# 7 - Check the Community Feedback

The number of stars and forks on GitHub can indicate popularity and community interest.

The screenshot shows the GitHub repository page for 'terraform-aws-ec2-instance'. The repository is public and has 735 stars and 1.8k forks. The 'Code' tab is selected, showing the master branch with 2 branches and 83 tags. The commit history lists several recent changes, including updates to CI workflows, examples, and wrappers, along with a chore release. The repository is described as a Terraform module to create AWS EC2 instances. It includes links to the registry, AWS, AWS-EC2, EC2-Instance, Terraform-Module, Readme, Apache-2.0 license, Code of conduct, Security policy, Activity, and Custom properties.

terraformer-aws-modules / **terraform-aws-ec2-instance** Public

Type  to search

Code Issues 3 Pull requests 1 Actions Security Insights

**terraform-aws-ec2-instance** Public

Sponsor Watch 24 Fork 1.8k Star 735

master 2 Branches 83 Tags Go to file Add file Code About

semantic-release-bot chore(release): version 5.6.1 [skip ci] 4f8387d · 2 months ago 177 Commits

.github/workflows fix: Update CI workflow versions to remove deprecated runti... 2 months ago

examples feat: Add example for connecting via Session Manager witho... 9 months ago

wrappers feat: Support Private DNS name options (#370) 5 months ago

.editorconfig [ci skip] Create ".editorconfig". 4 years ago

.gitignore chore: update documentation based on latest terraform-doc... 3 years ago

.pre-commit-config.yaml fix: Update CI workflow versions to remove deprecated runti... 2 months ago

.releaserc.json chore: Update release configuration files to correctly use con... 3 years ago

Terraform module to create AWS EC2 instances(s) resources UA

registry.terraform.io/modules/terraform-aws-ec2-instance

aws aws-ec2 ec2-instance

terraform-module

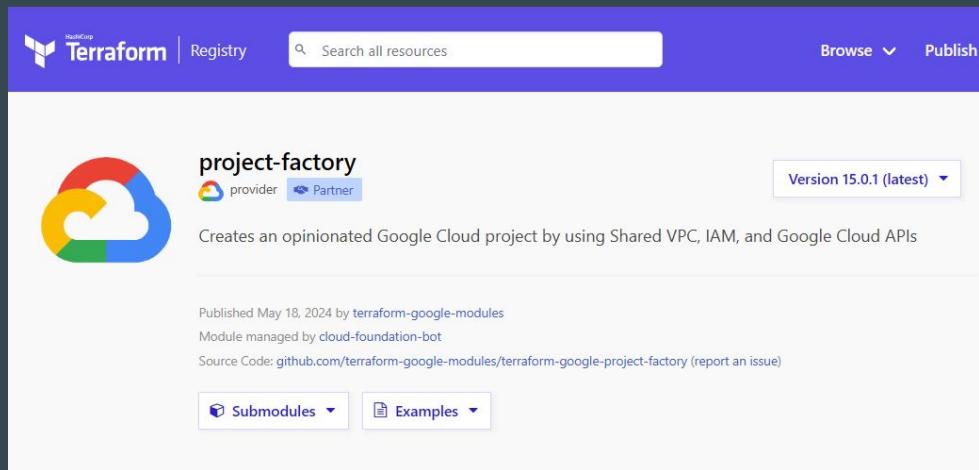
Readme Apache-2.0 license

Code of conduct Security policy

Activity Custom properties

# 8 - Modules Maintained by HashiCorp Partner

Search for modules that are maintained by HashiCorp Partners



## Important Point to Note

Avoid directly trying any random Terraform module that is not actively maintained and looks shady (primarily by sole individual contributors)

An attacker can include malicious code in a module that sends information about your environment to the attacker.

# Which Modules do Organizations Use?

In most of the scenarios, organizations maintain their own set of modules.

They might initially fork a module from the Terraform registry and modify it based on their use case.

# **Creating Base Module Structure**

# Understanding the Base Structure

A base “modules” folder.

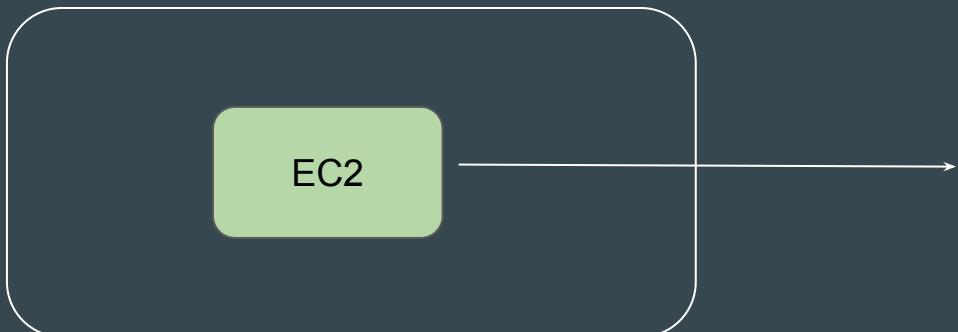
A sub-folder containing name for each modules that are available.



modules folder

# What is Inside the Sub-Folders

Each module's sub-folder contains the actual module Terraform code that other projects can reference from.



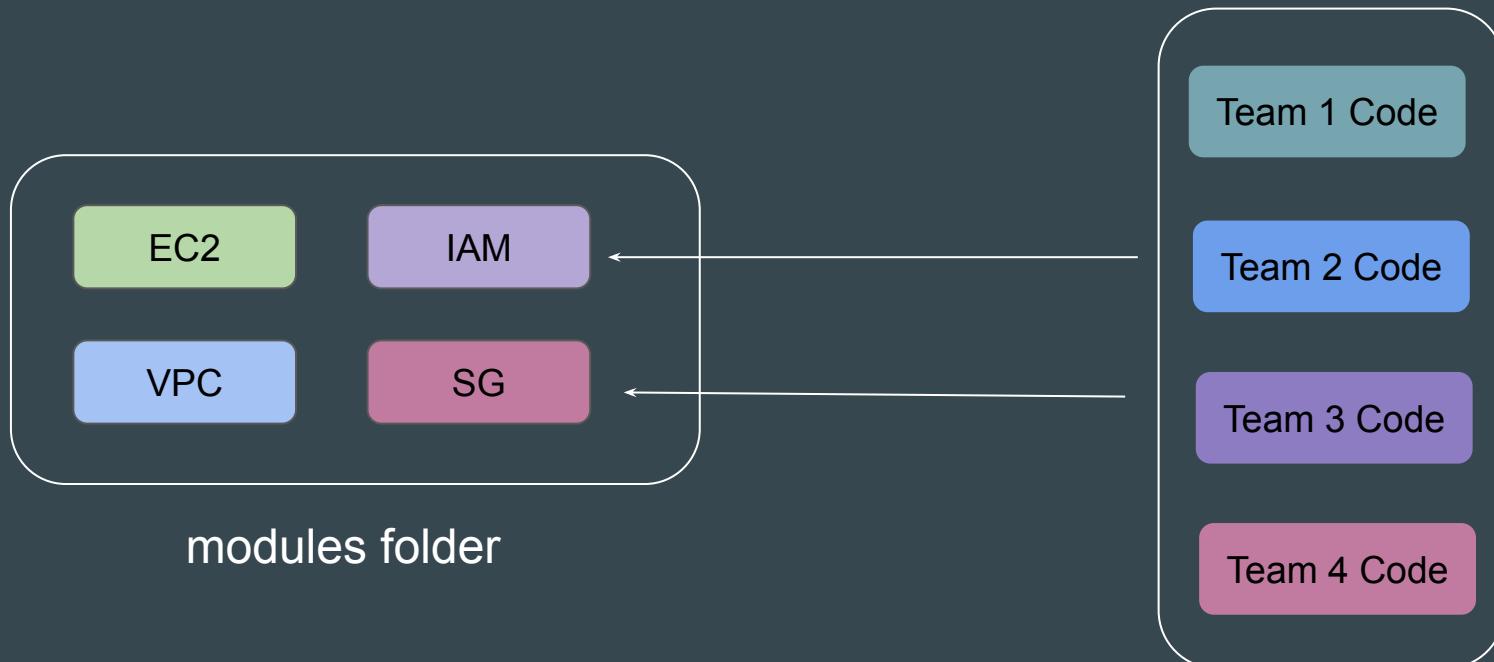
modules folder

```
resource "aws_instance" "web" {
    ami          = "ami-1234"
    instance_type = "t3.micro"
    key_name     = "user1"
    monitoring    = true
    vpc_security_group_ids = ["sg-12345678"]
    associate_public_ip_address = true
    instance_initiated_shutdown_behavior = "stop"
    ebs_optimized   = true
    source_dest_check = false
    hibernation = true
    disable_api_termination = true
}
```

main.tf

# Calling the Module

Each Team can call various set of modules that are available in the modules folder based on their requirements.

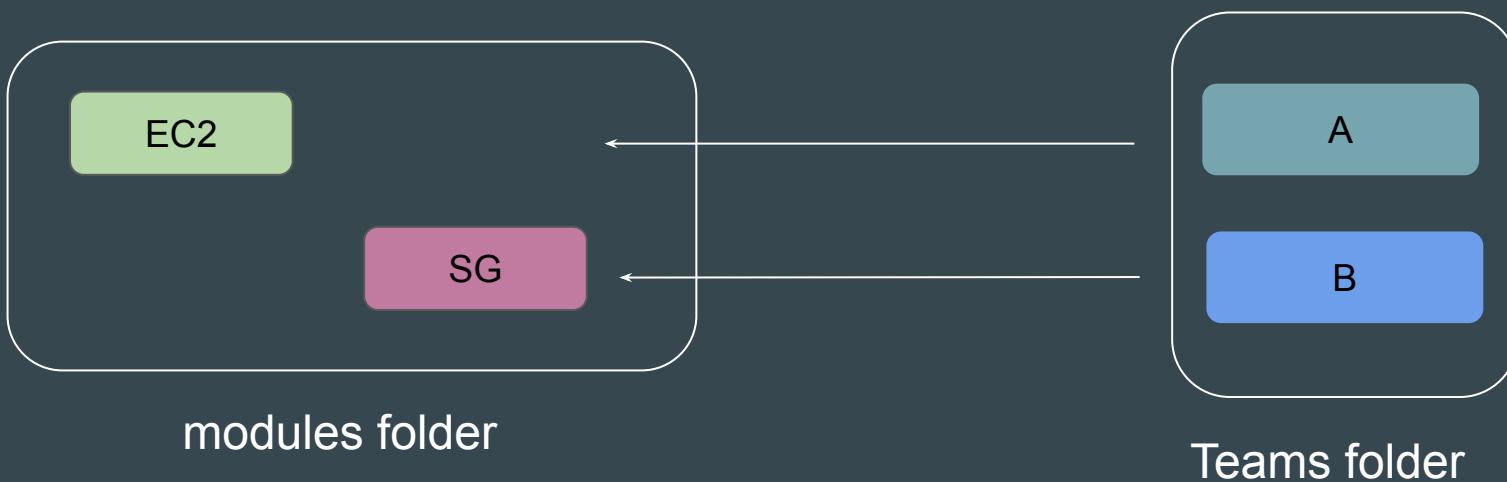


# Our Practical Structure

Our practical structure will include two main folders (modules and teams).

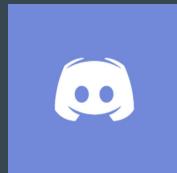
Modules sub-folder will contain sub-folders of modules that are available.

Teams sub-folder will contain list of teams that we want to be made available.



# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Module Sources - Calling a Module**

# Understanding the Base

Module source code can be present in wide variety of locations.

These includes:

1. GitHub
2. HTTP URLs
3. S3 Buckets
4. Terraform Registry
5. Local paths

# Base - Calling the Module

In order to reference to a module, you need to make use of **module** block

The module block must contain source argument that contains location to the referenced module.

```
module "ec2" {  
    source = "https://example.com/vpc-module.zip"  
}
```

## Example 1 - Local Paths

Local paths are used to reference to module that is available in local filesystem.

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

```
module "ec2" {
    source = "../modules/ec2"
}
```

## Example 2 - Generic Git Repository

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

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

# Module Version

A specific module can have multiple versions.

You can reference to specific version of module with the **version** block

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "20.11.1"
}
```

# **Improvements in Custom Module Code**

# Our Simple Module

We had created a very simple module that allows developers to launch an EC2 instance when calling the module.

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

resource "aws_instance" "myec2" {
    ami = "ami-0bb84b8ffd87024d8"
    instance_type = "t2.micro"
}
```

# Need to Analyze Shortcomings

Being a simplistic and a basic module code, there is a good room of improvements.

In today's video, we will be discussing about some of the important shortcomings with the code.



# Challenge 1 - Hardcoded Values

The values are hardcoded as part of the module.

If developer is calling the module, he will have to stick with same values.

Developer will not be able to override the hardcoded values of the module.

Hard-Coded Values

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

resource "aws_instance" "myec2" {
    ami = "ami-0bb84b8ffd87024d8"
    instance_type = "t2.micro"
}
```

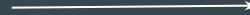
# Challenge 2 - Provider Improvements

Avoid hard-coding region in the Module code as much as possible.

A **required\_provider** block with version constraints for module to work is important.

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

resource "aws_instance" "myec2" {
  ami = "ami-0bb84b8ffd87024d8"
  instance_type = "t2.micro"
}
```



```
terraform {

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 5.5"
    }
  }

  resource "aws_instance" "myec2" {
    ami = "ami-0bb84b8ffd87024d8"
    instance_type = "t2.micro"
  }
}
```

# **Variables in Terraform Modules**

## Point to Note

As much as possible, avoid hardcoded values as part of the Modules.

This will make the module less flexible.



# Convert Hard Coded Values to Variables

For modules, it is especially recommended to convert hard-coded values to **variables** so that it can be overridden based on user requirements.

```
resource "aws_instance" "myec2" {  
    ami = "ami-0bb84b8ffd87024d8"  
    instance_type = "t2.micro"  
}
```



Bad Approach

Good Approach

# Advantages of Variables in Module Code

Variable based approach will allow the teams to override the values.

```
resource "aws_instance" "myec2" {
    ami = var.ami
    instance_type = var.instance_type
}
```

Main Module

Team 1

instance\_type = t2.micro

Team 2

instance\_type = m5.large

# Reviewing Professional EC2 Module Code

Reviewing an EC2 Module code that is professionally written, we see that the values associated with arguments are not hardcoded and variables are used extensively.

```
resource "aws_instance" "this" {
    count = local.create && !var.ignore_ami_changes && !var.create_spot_instance ? 1 : 0

    ami           = local.ami
    instance_type = var.instance_type
    cpu_core_count = var.cpu_core_count
    cpu_threads_per_core = var.cpu_threads_per_core
    hibernation   = var.hibernation

    user_data      = var.user_data
    user_data_base64 = var.user_data_base64
    user_data_replace_on_change = var.user_data_replace_on_change

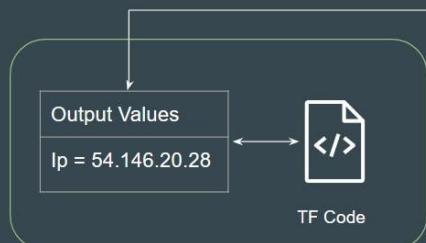
    availability_zone = var.availability_zone
    subnet_id        = var.subnet_id
    vpc_security_group_ids = var.vpc_security_group_ids

    key_name        = var.key_name
    monitoring      = var.monitoring
    get_password_data = var.get_password_data
    iam_instance_profile = var.create_iam_instance_profile ? aws_iam_instance_profile.this[0].name : var.iam_instance_profile
```

# **Module Outputs**

# 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.



Fetch



```
resource "aws_eip" "lb" {  
    domain    = "vpc"  
}  
  
output "public-ip" {  
    value = aws_eip.lb.public_ip  
}
```

Project A

# Understanding the Challenge

If you want to create a resource that has a dependency on an infrastructure created through a module, you won't be able to implicitly call it without output values.

```
resource "aws_instance" "myec2" {  
    ami = "ami-123"  
    instance_type = "t2.micro"  
}
```



```
module "ec2" {  
    source = "../../modules/ec2"  
}  
  
resource "aws_eip" "lb" {  
    instance = module.ec2.instance_id  
    domain  = "vpc"  
}
```

# Accessing Child Module Outputs

Ensure to include output values in the module code for better flexibility and integration with other resources and projects.

**Format:** module.<MODULE NAME>.<OUTPUT NAME>

```
resource "aws_instance" "myec2" {
    ami = var.ami
    instance_type = var.instance_type
}

output "instance_id" {
    value = aws_instance.myec2.id
}
```

```
module "ec2" {
    source = "../../modules/ec2"
}

resource "aws_eip" "lb" {
    instance = module.ec2.instance_id
    domain  = "vpc"
}
```

# Relax and Have a Meme Before Proceeding

No matter how hard you work, your boss will always arrive when you're sitting



# **Root and Child Modules**

# Root Module

Root Module resides in the **main working directory of your Terraform configuration**. This is the entry point for your infrastructure definition

```
module "ec2" {  
    source = "../../modules/ec2"  
}
```

Root Module

# Child Module

A **module that has been called by another module** is often referred to as a child module.



# **Standard Module Structure**

# Setting the Base

At this stage, we have been keeping the overall module structure very simple to understand the concepts.

In production environments, it is important to follow recommendations and best-practices set by HashiCorp.

# Basic of Standard Module Structure

The **standard module structure** is a file and directory layout HashiCorp recommends for reusable modules.

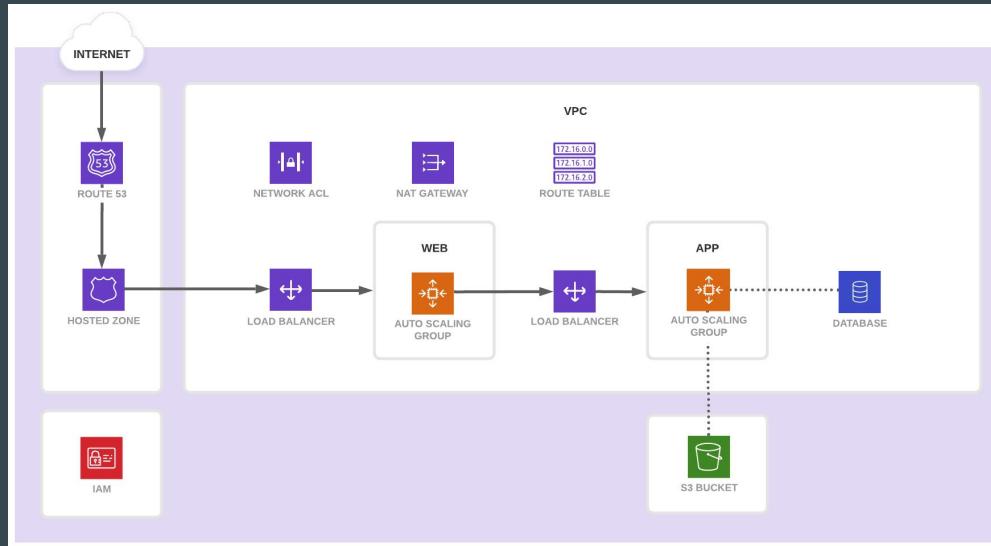
A minimal recommended module following the standard structure is shown below

```
$ tree minimal-module/
.
├── README.md
└── main.tf
    ├── variables.tf
    └── outputs.tf
```

# Scope the Requirements for Module Creation

A team wants to provision their infrastructure using Terraform.

The following architecture diagram depicts the desired outcome.



# Planning a Module Structure

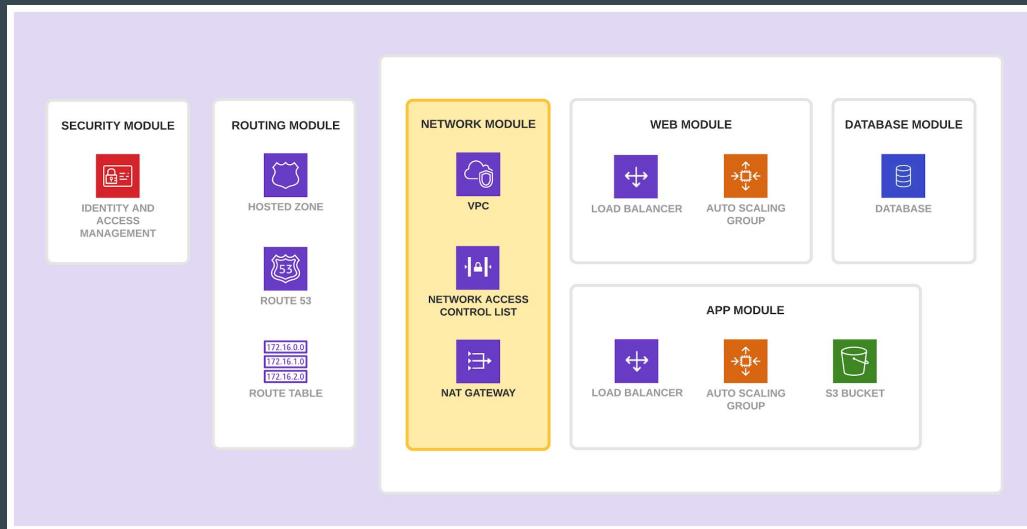
In this scenario, a team of Terraform producers, who write Terraform code from scratch, will build a collection of modules to provision the infrastructure and applications.

The members of the team in charge of the application will consume these modules to provision the infrastructure they need.

# Final Module Output

After reviewing the consumer team's requirements, the producer team has broken up the application infrastructure into the following modules:

Network, Web, App, Database, Routing, and Security.



# **Moved Blocks - Modules**

# Setting the Base

The aim is to move the reference of a created S3 bucket from an individual resource configuration to an S3 module for better flexibility.

```
resource "aws_s3_bucket" "a" {  
    bucket = "kplabs-sample-bucket"  
}
```

```
module "s3-bucket" {  
    source  = "terraform-aws-modules/s3-bucket/aws"  
    version = "4.1.2"  
    bucket  = "kplabs-sample-bucket"  
}
```



Managed through Module

# Final Solution - Reference Screenshot

```
module "s3-bucket" {
  source  = "terraform-aws-modules/s3-bucket/aws"
  version = "4.1.2"
  bucket  = "kplabs-moved-practical"
}

/*
resource "aws_s3_bucket" "moved" {
  bucket = "kplabs-moved-practical"
}
*/
moved {
  from = aws_s3_bucket.moved
  to   = module.s3-bucket.aws_s3_bucket.this[0]
}
```

# **Modules - Using Multiple Provider Configurations**

# Standard Single Provider Configuration

In a simple configurations, a child module automatically inherits default provider configurations from its parent.

This means that explicit provider blocks appear only in the root module, and child modules can simply declare resources for that provider.

```
modules > network > 🏫 sg.tf > ...
```

```
resource "aws_security_group" "dev" {
  name      = "dev-sg"
}

resource "aws_security_group" "prod" {
  name      = "prod-sg"
}
```

Child Module

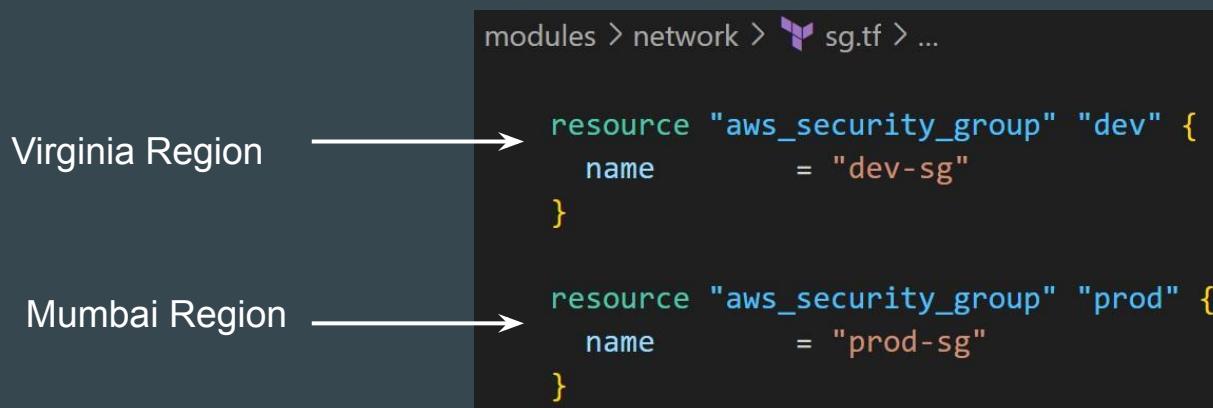
```
provider "aws" {
  region = "us-east-1"
}

module "sg" {
  source = "./modules/network"
}
```

Root Module

# Understanding the Use-Case

There can be many requirements where a specific module might want to access multiple regions for end to end resource creation.



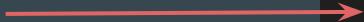
# Step 1 - Passing Providers Explicitly

You can use provider argument within a module block to explicitly define which provider configurations are available to the child module

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

provider "aws" {
    alias = "mumbai"
    region = "ap-south-1"
}

module "sg" {
    source = "./modules/network"
    providers = {
        aws.prod    = aws.mumbai
    }
}
```



## Step 2 - Declare Configuration Aliases

In the child module, you need to also declare the configuration aliases for the provider.

modules > network >  sg.tf > ...

```
terraformer {
    required_providers {
        aws = {
            source  = "hashicorp/aws"
            configuration_aliases = [ aws.prod]
        }
    }
}
```

## Step 3 - Use Provider Meta-Argument for Resources

You can use provider meta-argument within resource to choose appropriate provider configuration.

```
resource "aws_security_group" "dev" {
    name      = "dev-sg"
→ provider  = aws.prod
}

resource "aws_security_group" "prod" {
    name      = "prod-sg"
}
```

## Points to Note

The providers argument within a module block is similar to the provider argument within a resource, but is **a map** rather than a single string because a module may contain resources from many different providers.

Provider configurations (those with the alias argument set) are never inherited automatically by child modules, and so must always be passed explicitly using the providers map

# **Refactoring Use-Case 1 - Renaming Resource with Count**

# Setting the Base

If you change the name of a resource, Terraform will assume that the previous resource (a) needs to be removed and a new resource (b) needs to be created.

```
resource "aws_instance" "a" {  
  count = 2  
  ami   = "ami-123"  
  instance_type = "t2.micro"  
}
```



```
resource "aws_instance" "b" {  
  count = 2  
  ami   = "ami-123"  
  instance_type = "t2.micro"  
}
```

↑  
terraform apply

Resource Address
aws_instance.a[0]
aws_instance.a[1]

# Solution for Use-Case 1

If you later choose a different name for this resource, then you can change the name label in the resource block and record the old name inside a **moved** block.

Both the address in `aws_instance.a` are referred to as a whole so you don't have to mention [0] and [1] separately.

```
resource "aws_instance" "a" {  
  count = 2  
  ami    = "ami-123"  
  instance_type = "t2.micro"  
}
```



```
resource "aws_instance" "b" {  
  count = 2  
  ami    = "ami-123"  
  instance_type = "t2.micro"  
}  
  
moved {  
  from = aws_instance.a  
  to   = aws_instance.b  
}
```

# **Refactoring Use-Case 2 - Enabling Count for a Resource**

# Setting the Base

We can also refactor multiple set of same resource types into a single resource block that uses the count parameter.

```
resource "aws_instance" "a" {  
    ami      = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}  
  
resource "aws_instance" "b" {  
    ami      = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}
```



```
resource "aws_instance" "c" {  
    count      = 2  
    ami        = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}
```

## Solution - Use-Case 2

For migrating individual resource type to a new resource that uses count, you need to reference the index position in the moved block.

```
resource "aws_instance" "a" {  
    ami      = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}  
  
resource "aws_instance" "b" {  
    ami      = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}
```



```
resource "aws_instance" "c" {  
    count = 2  
    ami   = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}  
  
moved {  
    from = aws_instance.a  
    to   = aws_instance.c[0]  
}  
  
moved {  
    from = aws_instance.b  
    to   = aws_instance.c[1]  
}
```

# **Refactoring Use-Case 3 - Enabling `for_each` for a Resource**

# Refactoring Based on Count

Refactoring based on count is generally used when resources have same argument values.

```
resource "aws_instance" "a" {  
    ami      = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}  
  
resource "aws_instance" "b" {  
    ami      = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}
```



```
resource "aws_instance" "c" {  
    count = 2  
    ami   = "ami-0453ec754f44f9a4a"  
    instance_type = "t2.micro"  
}  
  
moved {  
    from = aws_instance.a  
    to   = aws_instance.c[0]  
}  
  
moved {  
    from = aws_instance.b  
    to   = aws_instance.c[1]  
}
```

# Setting the Base

For resource types that have different set of configurational values, you can make use of `for_each` meta-argument to combine into single resource block.

Since `for_each` is used, you have to ensure data is fetched from map or set.

```
resource "aws_instance" "a" {
  ami      = "ami-0453ec754f44f9a4a"
  instance_type = "t2.micro"
}

resource "aws_instance" "b" {
  ami      = "ami-0866a3c8686eaeeba"
  instance_type = "t3.micro"
}
```



```
resource "aws_instance" "b" {
  for_each      = local.instance_data
  ami           = each.value.ami_id
  instance_type = each.value.instance_type
}
```

# Solution - Use-Case 3

```
resource "aws_instance" "a" {
    ami      = "ami-0453ec754f44f9a4a"
    instance_type = "t2.micro"
}

resource "aws_instance" "b" {
    ami      = "ami-0866a3c8686eaeeba"
    instance_type = "t3.micro"
}
```



```
locals {
    instance_data = {
        first = {
            instance_type = "t2.micro"
            ami_id       = "ami-0453ec754f44f9a4a"
        }
        second = {
            instance_type = "t3.micro"
            ami_id       = "ami-0866a3c8686eaeeba"
        }
    }
}

resource "aws_instance" "c" {
    for_each      = local.instance_data
    ami           = each.value.ami_id
    instance_type = each.value.instance_type
}

moved {
    from = aws_instance.a
    to   = aws_instance.c["first"]
}

moved {
    from = aws_instance.b
    to   = aws_instance.c["second"]
}
```

## Points to Note

If you only have one argument whose value is different, you can also choose to make use of set.

# **Refactoring Use-Case 4 - Enabling count for Module Call**

# Setting the Base

Applying the following configuration would cause Terraform to create objects whose addresses begin with module.iam\_user

```
resource "aws_iam_user" "lb" {  
    name = "kplabs-user-1"  
}
```

```
module "iam_user" {  
    source = "./modules/iam"  
}
```

Child Module



Resource Address

module.iam\_user.aws\_iam\_user.lb

# Enabling Count in Module

At a later stage, if you need multiple set of users with different names, you might want to enable count for that resource.

Due to this, the first object that was initially created will be moved to index 0

```
resource "aws_iam_user" "lb" {
    name = "kplabs-user-${count.index}"
    count = 5
}
```

```
# module.iam_user.aws_iam_user.lb[0] will be updated in-place
# (moved from module.iam_user.aws_iam_user.lb)
~ resource "aws_iam_user" "lb" {
    id
        = "kplabs-user-1"
    ~ name
        = "kplabs-user-1" -> "kplabs-user-0"
    tags
        = {}
    # (6 unchanged attributes hidden)
}
```

## Solution - Use-Case 4

To preserve an object that was previously associated to be modified, you can add a moved block to specify which instance key that object will take in the new configuration

```
module "iam_user" {
    source = "./modules/iam"
}

moved {
    from = module.iam_user.aws_iam_user.lb
    to   = module.iam_user.aws_iam_user.lb[1]
}
```

# **Refactoring Use-Case 5 - Splitting One Module into Multiple**

# Setting the Base

As your configuration grows to support new requirements, it might eventually grow big enough to warrant splitting into two separate modules.

```
resource "aws_iam_user" "lb" {  
  name = "alice-user"  
}  
  
resource "aws_security_group" "sg" {  
  name      = "demo-sg"  
}
```



IAM Child Module



SG Child Module

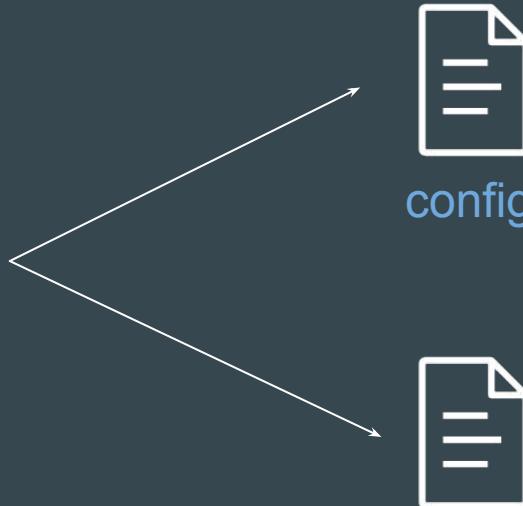
Root Configuration

# AWS CLI - Config and Credential Files

# Setting the Base

Whenever Access/Secret keys and region are added using the `aws configure` command, behind the scenes, your data gets stored to a file named config and credentials.

```
root@demo:~# aws configure
AWS Access Key ID [None]: ASIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/
Default region name [None]: us-east-1
Default output format [None]:
```



credentials

# Location of the Files

The credentials and config files are stored in .aws folder in your home directory.

Home directory location varies based on the operating system, but is referred to using the environment variables %UserProfile% in Windows and \$HOME or ~ (tilde) in Unix-based systems.

```
root@demo:~# ls -l ~/.aws
total 8
-rw----- 1 root root 29 Oct 18 02:46 config
-rw----- 1 root root 116 Oct 18 02:46 credentials
```

OS = Linux

```
C:\>dir %USERPROFILE%\.aws
Volume in drive C has no label.
Volume Serial Number is D835-695B

Directory of C:\Users\zealv\.aws

30-06-2023  13:18    <DIR>      .
16-10-2024  14:01    <DIR>      ..
16-04-2024  12:19          71 config
01-04-2024  21:38          939 credentials
```

OS = Windows

# What Data Gets Stored Where

The actual secrets (access/secret keys) gets stored in the credentials file.

The other configurations like region details, output format gets stored in config file.

```
root@demo:~# cat ~/.aws/credentials
[default]
aws_access_key_id = ASIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Credentials File

```
root@demo:~# cat ~/.aws/config
[default]
region = us-east-1
```

Config File

# **Shared Configuration and Credentials Files**

# Understanding the Basic

The AWS Provider can source credentials and other settings from the shared configuration and credentials files.

By default, these files are located in the .aws folder of the home directory and names are config and credentials.

```
root@demo:~# ls -l ~/.aws
total 8
-rw----- 1 root root 29 Oct 18 02:46 config
-rw----- 1 root root 116 Oct 18 02:46 credentials
```

OS = Linux

```
C:\>dir %USERPROFILE%\.aws
Volume in drive C has no label.
Volume Serial Number is D835-695B

Directory of C:\Users\zealv\.aws

30-06-2023  13:18    <DIR>      .
16-10-2024  14:01    <DIR>      ..
16-04-2024  12:19          71 config
01-04-2024  21:38        939 credentials
```

OS = Windows

# Setting the Base

Since Terraform automatically fetches the values from the shared configuration and settings file, you can skip mentioning the credentials in the provider {} block.

```
provider "aws" {}

resource "aws_security_group" "allow_tls" {
    name      = "demo-firewall"
}
```

# Different Path for Config and Credentials

If you want to fetch the credentials from different set of files in a different location, you can explicitly specify the parameters of `shared_config_files` and `shared_credentials_files`

```
provider "aws" {
    shared_config_files      = ["C:/kplabs-aws/config"]
    shared_credentials_files = ["C:/kplabs-aws/credentials"]
}

resource "aws_security_group" "allow_tls" {
    name          = "demo-firewall"
}
```

## Alternative - Environment Variable

If you do not want to hardcode the `shared_config_files` and `shared_credentials_files` in the provider block, you can also make use of environment variable.

ENV Variable: `AWS_CONFIG_FILE` and `AWS_SHARED_CREDENTIALS_FILE`

# **Named Profiles in AWS CLI**

# Setting the Base

Whenever we configure the credentials using aws configure command, the details gets stored as part of the [default] profile.

```
root@demo:~# aws configure
AWS Access Key ID [None]: ASIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/
Default region name [None]: us-east-1
Default output format [None]:
```

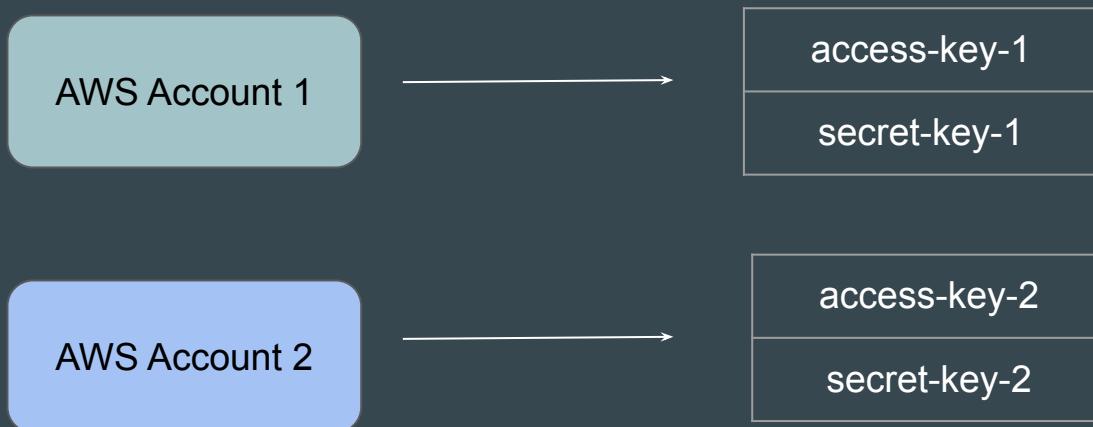
```
root@demo:~# cat ~/.aws/credentials
[default]
aws_access_key_id = ASIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfCiCYEXAMPLEKEY
```

```
root@demo:~# cat ~/.aws/config
[default]
region = us-east-1
```

# Understanding the Challenge

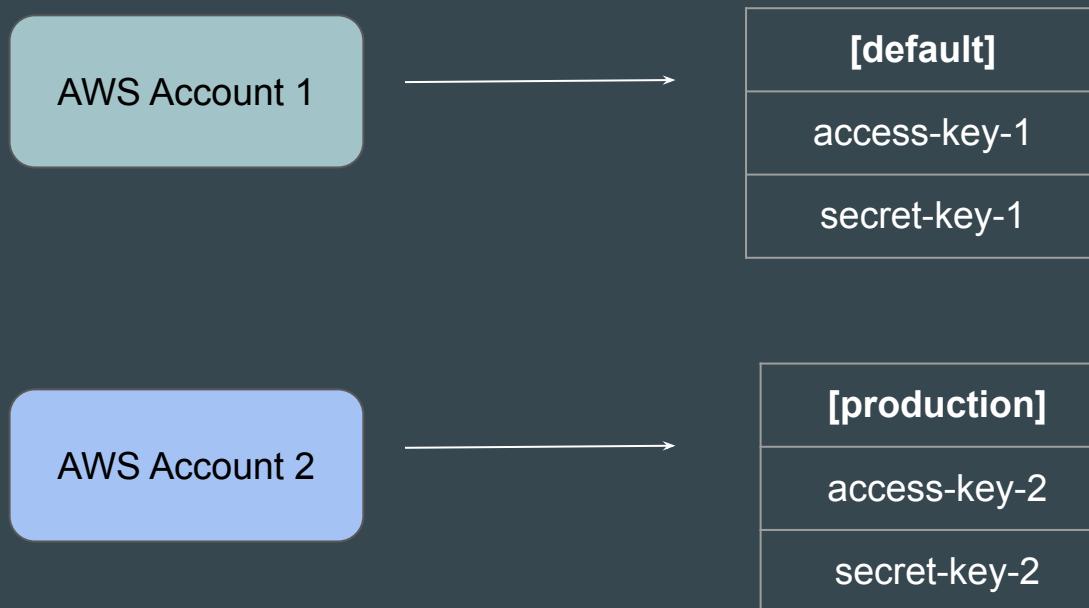
Each Access/Secret key belongs to one AWS account.

Organizations can have multiple AWS Accounts.



# Introducing Named Profiles

AWS CLI allows us to add **additional profiles** apart from [default] to store different set of access/secret keys and configuration.



# Defining Additional Profiles

Additional profiles can be added by using the `--profile` flag with `aws configure` command

```
root@demo:~# aws configure --profile production
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfICYEXAMPLEKEY
Default region name [None]: ap-south-1
Default output format [None]: json
```

# Exploring Config and Cred File for Multi-Profile

AWS CLI allows us to add additional profiles apart from [default] to store different set of access/secret keys and configuration.

```
root@demo:~# cat ~/.aws/credentials
[default]
aws_access_key_id = ASIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
[production]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws secret access key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

```
root@demo:~# cat ~/.aws/config
[default]
region = us-east-1
[profile production]
region = ap-south-1
output = json
```

## See Available List of Profiles

Use the **aws configure list-profiles** command to see list of available profiles.

```
root@demo:~# aws configure list-profiles
default
production
```

# Use a Specific Profile

You can add the --profile flag after a AWS CLI command to use a specific set of credentials associated with that profile.

```
root@demo:~# aws s3 ls --profile production
2023-02-17 06:00:59 aws-cloudtrail-logs-042025557788-4ed304e6
2024-09-02 06:36:09 becoming-shepherd-bucket
2023-05-29 04:12:19 cf-templates-187ru7orvw7ns-ap-south-1
2023-12-31 14:34:30 cf-templates-187ru7orvw7ns-us-east-2
2023-05-20 05:09:11 codepipeline-ap-south-1-169622477739
2023-05-20 04:24:51 codepipeline-ap-southeast-1-853474315569
2023-06-02 08:04:25 codepipeline-us-east-1-748395562256
2023-02-18 07:07:51 config-bucket-042025557788
```

# Relax and Have a Meme Before Proceeding

Me: Sorry I'm late, I broke down on the way to work.

Boss: Is your car working fine now?

Me: Car?

Boss:

Me:



# **AWS Provider - Setting Named Profile**

# Setting the Base

If no named profile is specified, the default profile is used.



sg.tf > ...

```
provider "aws" {}

resource "aws_security_group" "allow_tls" {
    name          = "demo-firewall"
}
```

# Specifying a Profile

You can explicitly specify a profile for the provider.

Based on the profile, Terraform will fetch the AWS access/secret keys and region values accordingly.

```
└── sg.tf > ...
  provider "aws" {
    profile = "production"
  }

  resource "aws_security_group" "allow_tls" {
    name          = "demo-firewall"
  }
```

# Point to Note

To set a profile, you can use the `profile` parameter or `AWS_PROFILE` environment variable to specify a named profile.

# **Multiple Provider Configuration**

# Understanding the Requirement

There can be a requirement that multiple resource types in the same TF file need to be deployed in separate regions.

```
resource "aws_instance" "myec2" {  
    ami          = "ami-08a0d1e16fc3f61ea"  
    instance_type = "t2.micro"  
}
```

```
resource "aws_security_group" "allow_tls" {  
    name          = "staging_firewall"  
}
```



Singapore Region



Mumbai Region

# Setting the Base

At this stage, we have been dealing with single-provider configuration.

In the below code, both resources will be created in Singapore region.

```
provider "aws" {
    region = "ap-southeast-1"
}

resource "aws_instance" "myec2" {
    ami          = "ami-08a0d1e16fc3f61ea"
    instance_type = "t2.micro"
}

resource "aws_security_group" "allow_tls" {
    name        = "staging_firewall"
}
```

# Alias Meta-Argument

Each provider can have one default configuration, and **any number of alternate configurations** that include an extra name segment (or "alias").

```
resource "aws_instance" "myec2" {
    ami           = "ami-08a0d1e16fc3f61ea"
    instance_type = "t2.micro"
}
```



```
provider "aws" {
    region = "ap-southeast-1"
}

provider "aws" {
    alias  = "mumbai"
    region = "ap-south-1"
}

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



```
resource "aws_security_group" "allow_tls" {
    name        = "staging_firewall"
}
```

# Final Output Using Alias

By using the provider meta-argument, you can select an alternate provider configuration for a resource.

```
provider "aws" {
    region = "ap-southeast-1"
}

provider "aws" {
    alias  = "mumbai"
    region = "ap-south-1"
}

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

```
resource "aws_instance" "myec2" {
    provider = aws.mumbai
    ami       = "ami-08a0d1e1e16fc3f61ea"
    instance_type = "t2.micro"
}
```

```
resource "aws_security_group" "allow_tls" {
    provider     = aws.usa
    name         = "staging_firewall"
}
```

# **AWS Provider - Default Tags**

# Setting the Base

AWS recommends that you define a robust and consistent tagging strategy to enable better auditing, cost, and access control for your AWS resources.

We manually add the tags associated with each resource that we create.

```
resource "aws_security_group" "allow_tls" {
    name          = "demo-firewall"

    tags = {
        Managed = "Terraform"
        Team    = "Security"
        Env     = "Production"
    }
}

resource "aws_iam_user" "lb" {
    name = "kplabs-user"

    tags = {
        Managed = "Terraform"
        Team    = "Security"
        Env     = "Production"
    }
}
```

# Issues with Manual Tagging

Many a times, the tagging standards are not followed in the organizations.

Repeatedly adding same tags for each resource increases the code length.

```
resource "aws_security_group" "allow_tls" {
    name          = "demo-firewall"

    tags = {
        Team = "Security"
    }
}

resource "aws_iam_user" "this" {
    name = "kplabs-user"

    tags = {
        Env  = "Production"
    }
}
```

# Default Tags at Provider Level

The AWS Terraform provider allows you to add default tags to all resources that the provider creates.

This is designed to replace redundant per-resource tags configurations.

```
provider "aws" {
  default_tags {
    tags = {
      Managed = "Terraform"
      Team    = "Security"
      Env     = "Production"
    }
  }
}
```

# Overriding Default Tags

Provider tags can be overridden with new values, but not excluded from specific resources.

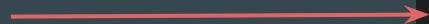
To override provider tag values, use the tags argument within a resource to configure new tag values for matching keys

```
provider "aws" {
  default_tags {
    tags = {
      Managed = "Terraform"
      Team    = "Security"
      Env     = "Production"
    }
  }
}

resource "aws_security_group" "allow_tls" {
  name          = "demo-firewall"

  tags = {
    Team    = "Networking"
  }
}
```

Higher Precedence



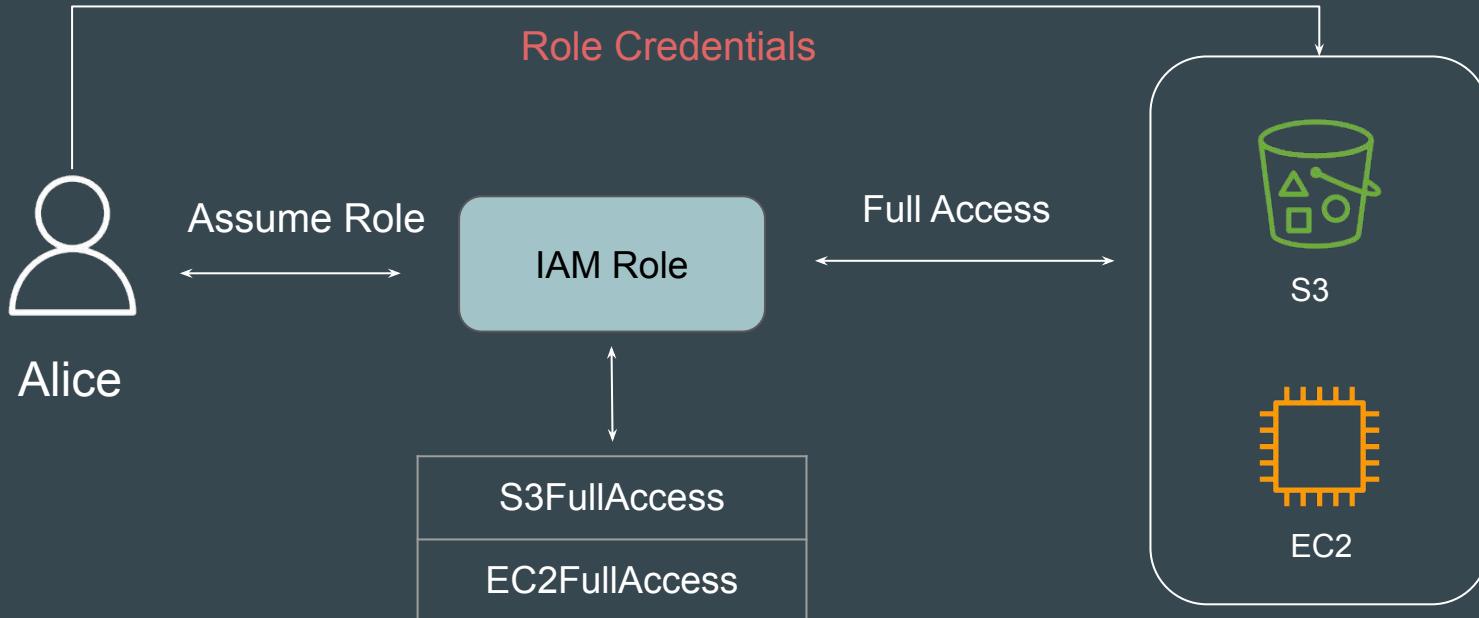
## Point to Note

This functionality is supported in all resources that implement tags, with the exception of the aws\_autoscaling\_group resource.

# **AWS Provider - Assuming an IAM Role**

# Understanding the Basic

In many scenarios, users will have to assume a specific IAM role to obtain the appropriate permissions to make changes to the infrastructure.



# Assume IAM Role in Terraform

If provided with a role ARN, the AWS Provider will attempt to assume this role using the supplied credentials.

```
provider "aws" {
  assume_role {
    role_arn      = "arn:aws:iam::123456789012:role/ROLE_NAME"
    session_name = "kplabs-demo"
  }
}

resource "aws_security_group" "allow_tls" {
  name          = "demo-firewall"
}
```

# **Dependency Lock File**

# Revising the Basics

Provider Plugins and Terraform are managed independently and have different release cycle.

AWS Plugin v1

AWS Plugin v2

AWS Plugin v3

Terraform v1.2

# Understanding the Challenge

The AWS code written in Terraform is working perfectly well with AWS Plugin v1

It can happen that same code might have some issues with newer AWS plugins.



# Version Dependencies

Version constraints within the configuration itself determine which versions of dependencies are potentially compatible.

After selecting a specific version of each dependency Terraform remembers the decisions it made in a dependency lock file so that it can (by default) make the same decisions again in future.

```
≡ .terraform.lock.hcl
1 # This file is maintained automatically by "terraform init".
2 # Manual edits may be lost in future updates.
3
4 provider "registry.terraform.io/hashicorp/aws" {
5     version      = "4.62.0"
6     constraints = "~> 4.0"
7     hashes = [
8         "h1:TBu xeLRMrChr zgDhAyil3HITJhuNrL+X58QRC+FTmFQ=",
9         "zh:12059dc2b639797b9facb6397ac6aec563891634be8e5aadf3a4
10        "zh:1b3515d70b6998359d0a6d3b3c287940ab2e5c59cd02f95c7d9d
```



```
demo.tf
●
demo.tf > ...
1  terraform {
2      required_providers {
3          aws = {
4              source  = "hashicorp/aws"
5              version = "~> 4.0"
6          }
7      }
8  }
9
10 # Configure the AWS Provider
11 provider "aws" {
12     region = "us-east-1"
13 }
```

# Default Behaviour

What happens if you update the TF file with version that does not match the `terraform.lock.hcl`?

```
➜ .terraform.lock.hcl
1 # This file is maintained automatically by "terraform init".
2 # Manual edits may be lost in future updates.
3
4 provider "registry.terraform.io/hashicorp/aws" {
5   version    = "4.62.0"
6   constraints = "~> 4.0"
7   hashes = [
8     "h1:TBu xeL RMr Chr zgDhAy il3HITJhuN rL+X58QRC+FTmFQ=",
9     "zh:12059dc2b639797b9facb6397ac6aec563891634be8e5aadf3a4
10    "zh:1b3515d70b6998359d0a6d3b3c287940ab2e5c59cd02f95c7d9d
```

```
demo.tf > ...
  terraform {
    required_providers {
      aws = {
        source  = "hashicorp/aws"
        version = "4.60"
      }
    }
  }

  # Configure the AWS Provider
  provider "aws" {
    region = "us-east-1"
  }
```

```
C:\Users\zealv\Desktop\kplabs-terraform>terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file

Error: Failed to query available provider packages

Could not retrieve the list of available versions for provider hashicorp/aws: local configured version constraint 4.60.0; must use terraform init -upgrade to allow
```

# Upgrading Option

If there is a requirement to use newer or downgrade a provider, can override that behavior by adding the `-upgrade` option when you run `terraform init`, in which case Terraform will disregard the existing selections

```
C:\Users\zealv\Desktop\kplabs-terraform>terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "4.60.0"...
- Installing hashicorp/aws v4.60.0...
- Installed hashicorp/aws v4.60.0 (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.
```

# Points to Note

When installing a particular provider for the first time, Terraform will pre-populate the hashes value with any checksums that are covered by the provider developer's cryptographic signature, which usually covers all of the available packages for that provider version across all supported platforms.

```
≡ .terraform.lock.hcl
  # This file is maintained automatically by "terraform init".
  # Manual edits may be lost in future updates.

  provider "registry.terraform.io/hashicorp/aws" {
    version      = "4.60.0"
    constraints = "4.60.0"
    hashes = [
      "h1:M90xusbiz/HW7zF+jLddXqdpzsFZ38Fa2S+Yaquad2g=",
      "zh:1853d6bc89e289ac36c13485e8ff877c1be8485e22f545bb32c7a30f1d1856e8",
      "zh:4321d145969e3b7ede62fe51bee248a15fe398643f21df9541eef85526bf3641",
      "zh:4c01189cc6963abfe724e6b289a7c06d2de9c395011d8d54efa8fe1aac444e2e",
      "zh:5934db7baa2eec0f9acb9c7f1c3dd3b3fe1e67e23dd4a49e9fe327832967b32b",
```

## Points to Note

At present, the dependency lock file tracks only provider dependencies.

Terraform does not remember version selections for remote modules, and so Terraform will always select the newest available module version that meets the specified version constraints.

# Debugging Terraform

# Basics of Debugging

Debugging is the **process of finding the root cause** of a specific issue.

30-40% of the time of a System Administrator goes into Debugging.



# Example - SSH Verbosity

One of the important requirement in Debugging is getting detailed Log

Depending on the application, the approach to get detailed logs will differ.

```
C:\Users\zealv\OneDrive\Desktop>ssh -v -i dp_rsa root@64.227.154.149
OpenSSH_for_Windows_8.6p1, LibreSSL 3.4.3
debug1: Authenticator provider $SSH_SK_PROVIDER did not resolve; disabling
debug1: Connecting to 64.227.154.149 [64.227.154.149] port 22.
debug1: Connection established.
debug1: identity file dp_rsa type -1
debug1: identity file dp_rsa-cert type -1
debug1: Local version string SSH-2.0-OpenSSH_for_Windows_8.6
debug1: Remote protocol version 2.0, remote software version OpenSSH_9.6p1 Ubuntu-3ubuntu13.4
debug1: compat_banner: match: OpenSSH_9.6p1 Ubuntu-3ubuntu13.4 pat OpenSSH* compat 0x04000000
debug1: Authenticating to 64.227.154.149:22 as 'root'
debug1: load_hostkeys: fopen C:\\\\Users\\\\zealv/.ssh/known_hosts2: No such file or directory
debug1: load_hostkeys: fopen __PROGRAMDATA__\\\\ssh\\\\ssh_known_hosts: No such file or directory
debug1: load_hostkeys: fopen __PROGRAMDATA__\\\\ssh\\\\ssh_known_hosts2: No such file or directory
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256
debug1: kex: host key algorithm: ssh-ed25519
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
```

# Debugging in Terraform

Similar to SSH Verbosity, even Terraform allows us to set wide variety of log levels for getting detailed logs for debugging purpose.

```
root@test-kplabs:~# terraform plan
2024-08-24T05:01:48.844Z [INFO]  Terraform version: 1.9.5
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/go-tfe v1.58.0
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/hcl/v2 v2.20.0
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/terraform-svchost v0.1.1
2024-08-24T05:01:48.845Z [DEBUG] using github.com/zclconf/go-cty v1.14.4
2024-08-24T05:01:48.845Z [INFO] Go runtime version: go1.22.5
2024-08-24T05:01:48.845Z [INFO] CLI args: []string{"terraform", "plan"}
2024-08-24T05:01:48.845Z [TRACE] Stdout is a terminal of width 125
2024-08-24T05:01:48.845Z [TRACE] Stderr is a terminal of width 125
2024-08-24T05:01:48.845Z [TRACE] Stdin is a terminal
2024-08-24T05:01:48.845Z [DEBUG] Attempting to open CLI config file: /root/.terraformrc
2024-08-24T05:01:48.845Z [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2024-08-24T05:01:48.845Z [DEBUG] ignoring non-existing provider search directory terraform.d/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /root/.terraform.d/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /root/.local/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /usr/local/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /usr/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /var/lib/snapd/desktop/terraform/plugins
2024-08-24T05:01:48.848Z [INFO] CLI command args: []string{"plan"}
```

# Storing the Logs to File

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

```
root@test-kplabs:~# terraform plan
2024-08-24T05:01:48.844Z [INFO] Terraform version: 1.9.5
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/go-tfe v1.58.0
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/hcl/v2 2.20.0
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/terraform-svchost v0.1.1
2024-08-24T05:01:48.845Z [DEBUG] using github.com/cilicon/ci-cty v1.14.4
2024-08-24T05:01:48.845Z [INFO] Go runtime version: go1.22.5
2024-08-24T05:01:48.845Z [INFO] CLI args: []string{"terraform", "plan"}
2024-08-24T05:01:48.845Z [TRACE] Stdout is a terminal of width 125
2024-08-24T05:01:48.845Z [TRACE] Stderr is a terminal of width 125
2024-08-24T05:01:48.845Z [TRACE] Stdin is a terminal
2024-08-24T05:01:48.845Z [DEBUG] Attempting to open CLI config file: /root/.terraformrc
2024-08-24T05:01:48.845Z [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2024-08-24T05:01:48.845Z [DEBUG] ignoring non-existing provider search directory /root/.terraform.d/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /root/.terraform.d/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /root/.local/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /usr/local/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /usr/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /var/lib/snapd/desktop/terraform/plugins
2024-08-24T05:01:48.848Z [INFO] CLI command args: []string{"plan"}
```



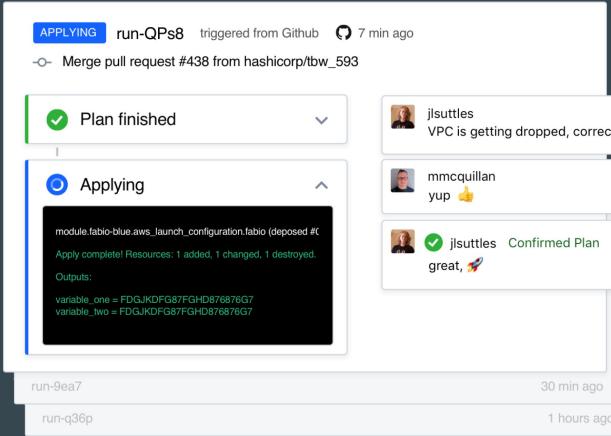
# **HashiCorp Cloud Platform - Terraform**

# Setting the Base

Till now, we have been managing Terraform through CLI

Although this approach is decent, we should also be aware that there is a GUI based offering that is available.

```
root@test-kplabs:~# terraform plan
2024-08-24T05:01:48.844Z [INFO]  Terraform version: 1.9.5
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/go-tfe v1.58.0
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/hcl/v2 v2.20.0
2024-08-24T05:01:48.845Z [DEBUG] using github.com/hashicorp/terraform-svchost v0.1.1
2024-08-24T05:01:48.845Z [DEBUG] using github.com/zclconf/go-cty v1.14.4
2024-08-24T05:01:48.845Z [INFO] Go runtime version: go1.22.5
2024-08-24T05:01:48.845Z [INFO] CLI args: []string{"terraform", "plan"}
2024-08-24T05:01:48.845Z [TRACE] Stdout is a terminal of width 125
2024-08-24T05:01:48.845Z [TRACE] Stderr is a terminal of width 125
2024-08-24T05:01:48.845Z [TRACE] Stdin is a terminal
2024-08-24T05:01:48.845Z [DEBUG] Attempting to open CLI config file: /root/.terraformrc
2024-08-24T05:01:48.845Z [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2024-08-24T05:01:48.845Z [DEBUG] ignoring non-existing provider search directory /root/.terraform.d/plugins
2024-08-24T05:01:48.845Z [DEBUG] ignoring non-existing provider search directory /root/.local/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /usr/local/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /usr/share/terraform/plugins
2024-08-24T05:01:48.846Z [DEBUG] ignoring non-existing provider search directory /var/lib/snapd/desktop/terraform/plugins
2024-08-24T05:01:48.848Z [INFO] CLI command args: []string{"plan"}
```



# HCP Terraform

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

The screenshot shows the HCP Terraform interface. At the top, it displays a message from 'mykplabs' that triggered a run from the UI a few seconds ago. Below this, the 'Run Details' section shows a green checkmark indicating the 'Plan finished' status a few seconds ago. It also shows 'Resources: 1 to add, 0 to change, 0 to destroy'. A prominent green bar indicates '+ 1 to create'. The interface includes filters for resources by address and action, and links to download raw logs and sentinel mocks. In the bottom section, a green checkmark indicates the 'Cost estimation finished' status a few seconds ago. It shows 'Resources: 1 of 1 estimated · \$8.35/mo · +\$8.35'. A table provides detailed cost information for an 'aws\_instance' named 'myec2':

RESOURCE	NAME	HOURLY COST	MONTHLY COST	MONTHLY DELTA
✓ aws_instance	myec2	\$0.012	\$8.352	+\$8.352

# Not Everything is Free

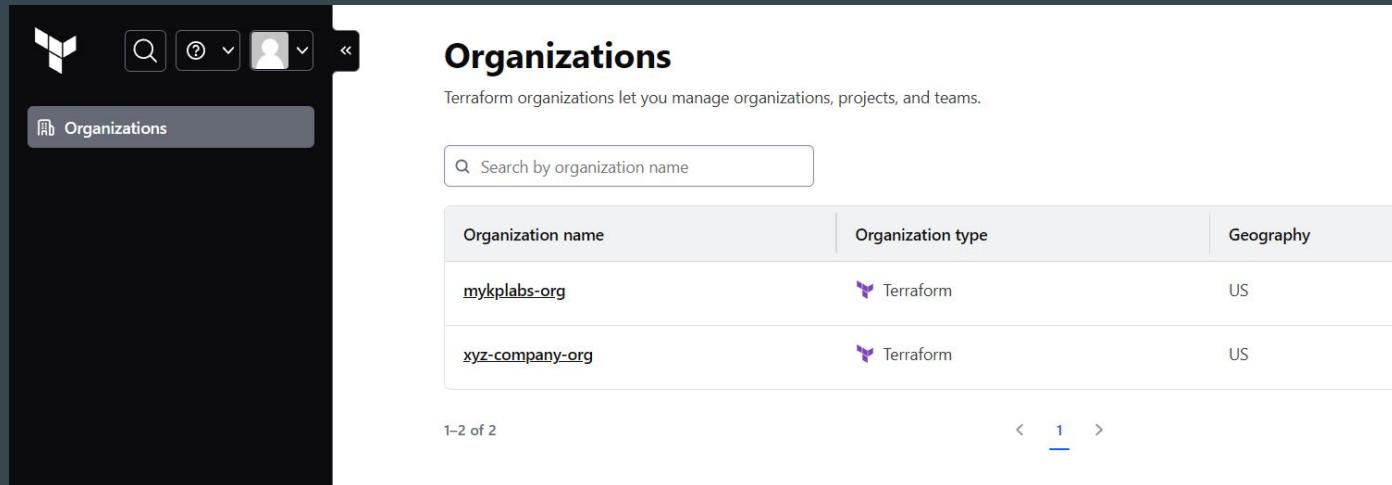
HCP Terraform is not entirely free. Depending on the usage and features needed, there are multiple pricing plans that are available.

Terraform pricing			
Free	Standard	Plus	Enterprise
UP TO <b>500 resources</b> per month	STARTING AT <b>\$0.00014</b> per hour per resource	Custom	Custom
Cloud Get started with all capabilities needed for infrastructure as code provisioning. No credit card required	Cloud For professional individuals or teams adopting infrastructure as code provisioning. Enterprise support included	Cloud For enterprises standardizing and managing infrastructure automation and lifecycle, with scalable runs. Enterprise support included	Self-managed For enterprises with special security, compliance, and additional operational requirements. Enterprise support included
<a href="#">Start for free</a>	<a href="#">Start for free</a>	<a href="#">Contact sales</a>	<a href="#">Contact sales</a>

# **HCP Terraform - Basic Structure**

# 1 - Organizations

Organizations are a shared space for one or more teams to collaborate on workspaces.



The screenshot shows the Terraform Cloud interface with the 'Organizations' page selected. The top navigation bar includes a logo, search, help, and user icons. A sidebar on the left has a 'Organizations' button. The main content area is titled 'Organizations' and contains a sub-header: 'Terraform organizations let you manage organizations, projects, and teams.' Below this is a search bar labeled 'Search by organization name'. A table lists two organizations:

Organization name	Organization type	Geography
<a href="#">mykplabs-org</a>	Terraform	US
<a href="#">xyz-company-org</a>	Terraform	US

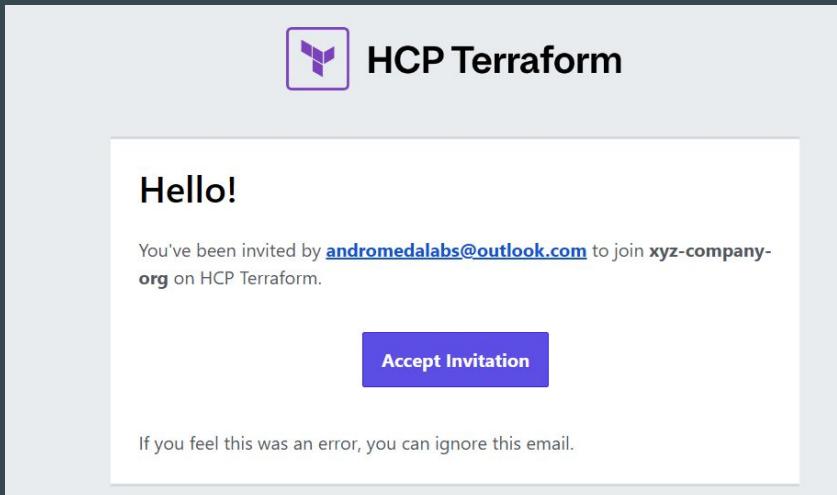
At the bottom, it shows '1-2 of 2' and a page number '1'.

Organization name	Organization type	Geography
<a href="#">mykplabs-org</a>	Terraform	US
<a href="#">xyz-company-org</a>	Terraform	US

# Points to Note - Organizations

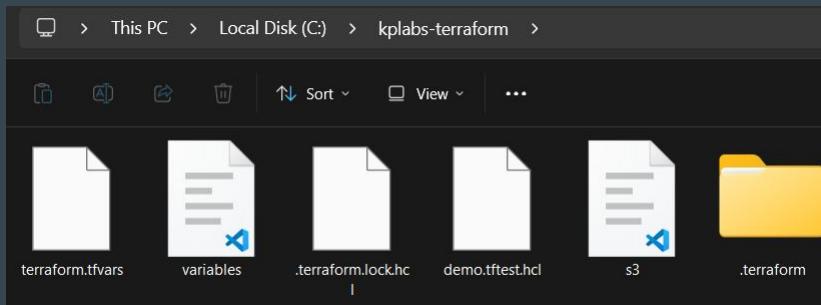
HCP Terraform manages plans and billing at the organization level.

Each HCP Terraform user can belong to multiple organizations, which might subscribe to different billing plans.



# 2 - Workspace

HCP Terraform manages infrastructure collections with workspaces instead of directories

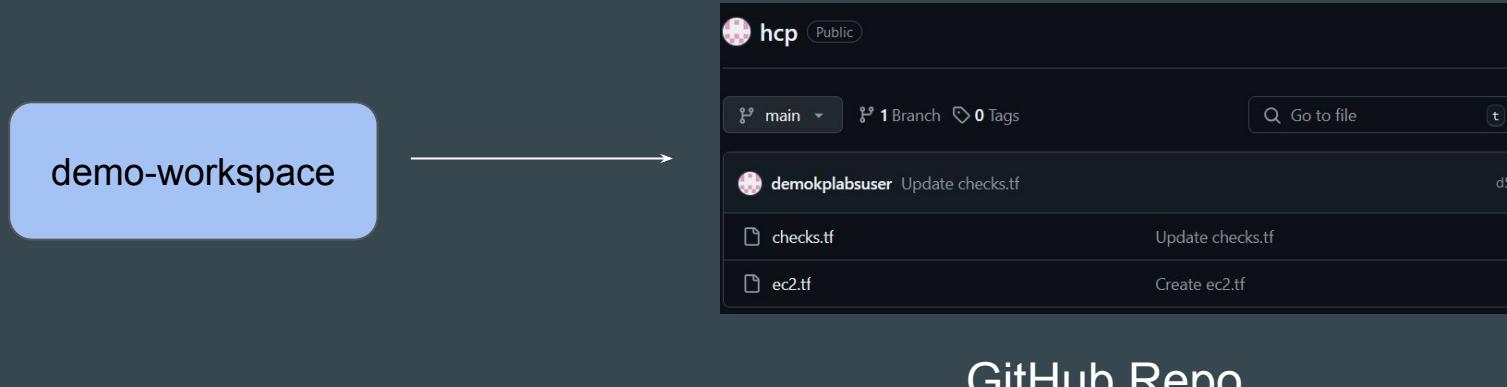


A screenshot of the HashiCorp Cloud Platform (HCP) workspace interface. The workspace is named 'hcp' with ID 'ws-SfMu8aw7NnmPzx2F'. It shows 'Running', 'Resources 1', 'Terraform v1.9.5', and 'Updated an hour ago'. The 'Latest Run' section shows a run triggered via UI by 'mykplabs' an hour ago. It details policy checks ('Add'), estimated cost increase ('\$0.35'), plan duration ('Less than a minute'), and resources to be changed ('+1 ~0 -0'). A 'See details' button is available. On the right, there's a sidebar with project settings: 'Execution mode: Remote', 'Auto-apply API, CLI, & VCS runs: Off', 'Auto-apply run triggers: Off', 'Auto-destroy: Off', and 'Project: kplabs-workspace'. At the bottom, a 'Metrics' section shows the last 2 runs.

# Workspace & Configuration Files

The Terraform configuration file (sample.tf) is not directly uploaded to a workspace.

Instead, workspace is connected to GitHub repository where it can fetch code from.



# Workspace vs Directories

Component	Local Terraform	HCP Terraform
Terraform configuration	On disk	In linked version control repository, or periodically uploaded via API/CLI
Variable values	As <code>.tfvars</code> files, as CLI arguments, or in shell environment	In workspace
State	On disk or in remote backend	In workspace
Credentials and secrets	In shell environment or entered at prompts	In workspace, stored as sensitive variables

# 3 - Projects

HCP Terraform projects let you organize your workspaces into groups.

 **security-team-project** New ▾

ID: prj-QN8iVccxZ7ML2eE1 

[Add project description](#)

 Teams 0  Workspaces 3

Workspace name	Repository	Health	Latest change
 <a href="#">azure-hardening</a> No status reported	None	None	a few seconds ago
 <a href="#">gcp-hardening</a> No status reported	None	None	a few seconds ago
 <a href="#">aws-hardening</a> No status reported	None	None	a minute ago

## Point to Note - Projects

You can structure your projects based on your organization's resource usage and ownership patterns, such as teams, business units, or services.

With HCP Terraform Standard Edition, you can give teams access to groups of workspaces using projects.

# **The CLI-driven Run Workflow**

# Setting the Base

Whenever we create a new Workspace in HCP, following are the 3 types of workflow modes that are available.

The screenshot shows a web-based interface for creating a new workspace. At the top, the URL is "mykplabs-org / Workspaces / New Workspace". The main title is "Create a new Workspace". Below it, a descriptive text states: "HCP Terraform organizes your infrastructure resources by workspaces. A workspace contains infrastructure resources, variables, state data, and run history. Learn more about workspaces in HCP Terraform." A section titled "Choose your workflow" presents three options:

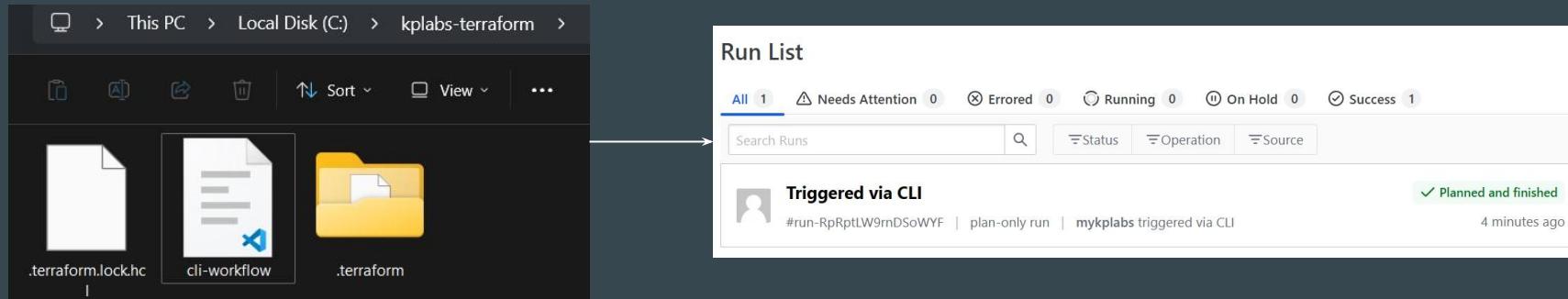
- Version Control Workflow**: Trigger runs based on changes to configuration in repositories. Best for those who need traceability and transparency.
- CLI-Driven Workflow**: Trigger runs in a workspace using the Terraform CLI. Best for those comfortable with Terraform CLI.
- API-Driven Workflow**: Trigger runs using the HCP Terraform API. Best for those with custom integrations and pipelines.

A "Cancel" button is located at the bottom right of the form.

# Basics of CLI Driven Workflow

In this approach, the working directory on your workstation is linked with HCP Workspace.

The code file can be present in your laptop, and plan/apply operations can also be initiated from local workstation.



# **The CLI-driven Run Workflow - Practical**

# Setting the Base

Whenever we create a new Workspace in HCP, following are the 3 types of workflow modes that are available.

The screenshot shows a web-based interface for creating a new workspace. At the top, the URL is "mykplabs-org / Workspaces / New Workspace". The main title is "Create a new Workspace". Below it, a descriptive text states: "HCP Terraform organizes your infrastructure resources by workspaces. A workspace contains infrastructure resources, variables, state data, and run history. Learn more about workspaces in HCP Terraform." A section titled "Choose your workflow" presents three options:

- Version Control Workflow**: Trigger runs based on changes to configuration in repositories. Best for those who need traceability and transparency.
- CLI-Driven Workflow**: Trigger runs in a workspace using the Terraform CLI. Best for those comfortable with Terraform CLI.
- API-Driven Workflow**: Trigger runs using the HCP Terraform API. Best for those with custom integrations and pipelines.

A "Cancel" button is located at the bottom right of the form.

# Step 1 - Setup Cloud Integration

You have to add code block within your .tf file to setup cloud integration.

This code will contain details about your HCP organization and workspace name.

```
terraform {
  cloud {

    organization = "mykplabs-org"

    workspaces {
      name = "remote-operation-workspace"
    }
  }
}
```

## Step 2 - Terraform Login

Once your cloud integration code block has been added, next step is to run the **terraform login** command.

```
C:\kplabs-terraform>terraform login
Terraform will request an API token for app.terraform.io using your browser.
```

```
If login is successful, Terraform will store the token in plain text in
the following file for use by subsequent commands:
```

```
C:\Users\zealv\AppData\Roaming\terraform.d\credentials.tfrc.json
```

```
Do you want to proceed?
```

```
Only 'yes' will be accepted to confirm.
```

```
Enter a value: yes
```

---

```
Terraform must now open a web browser to the tokens page for app.terraform.io.
```

```
If a browser does not open this automatically, open the following URL to proceed:
```

```
https://app.terraform.io/app/settings/tokens?source=terraform-login
```

# Step 3 - Initialize

Run the `terraform init` command to initialize

```
C:\kplabs-terraform>terraform init
Initializing HCP Terraform...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.70.0...
- Installed hashicorp/aws v5.70.0 (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.
```

HCP Terraform has been successfully initialized!

You may now begin working with HCP Terraform. Try running "terraform plan" to
see any changes that are required for your infrastructure.

If you ever set or change modules or Terraform Settings, run "terraform init"
again to reinitialize your working directory.

# Step 4 - Run the Plan / Apply Operations

Once initialized, the terraform “plan”, and “apply” commands when entered through CLI will run in HCP Terraform with output streamlined in terminal.

```
C:\kplabs-terraform>terraform plan
Running plan in HCP Terraform. Output will stream here. Pressing Ctrl-C
will stop streaming the logs, but will not stop the plan running remotely.

Preparing the remote plan...

To view this run in a browser, visit:
https://app.terraform.io/app/mykplabs-org/remote-operation-workspace/runs/run-RpRptLW9rnDSoWYF

Waiting for the plan to start...

Terraform v1.9.7
on linux_amd64
Initializing plugins and modules...

Terraform used the selected providers to generate the following execution plan. Resource actions
following symbols:
+ create
```

# Relax and Have a Meme Before Proceeding

if you ever feel useless, remember  
someone made a protective cover  
for Nokia 3310



**Ock Capital**  
@KSE\_Sports

It was to protect the floor

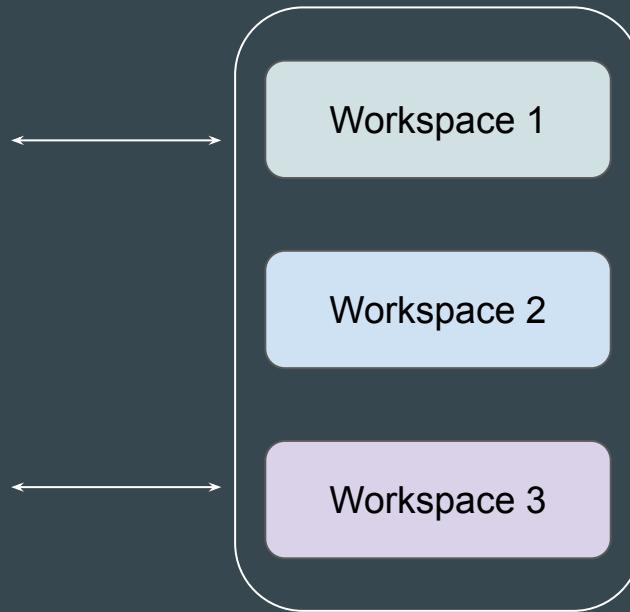
# **Variable Sets**

# Understanding the Challenge

It is common for multiple workspaces to save same set of variables.

If any variable value changes, it needs to be replaced at all the workspace level.

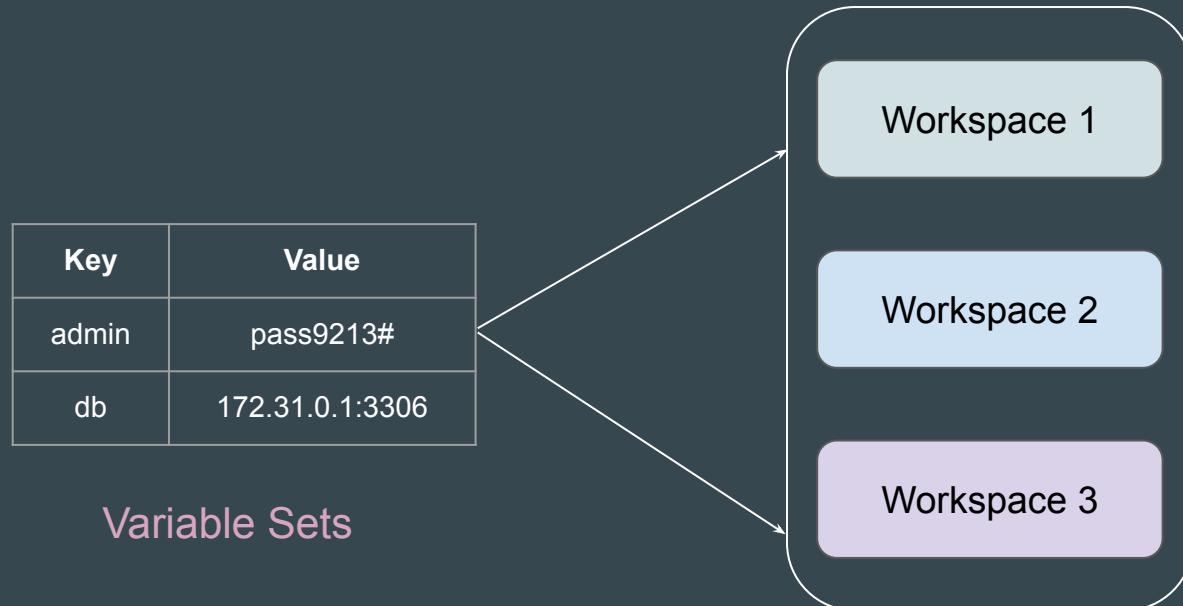
Key	Value
admin	pass9213#
db	172.31.0.1:3306



Key	Value
admin	pass9213#
db	172.31.0.1:3306

# Setting the Base

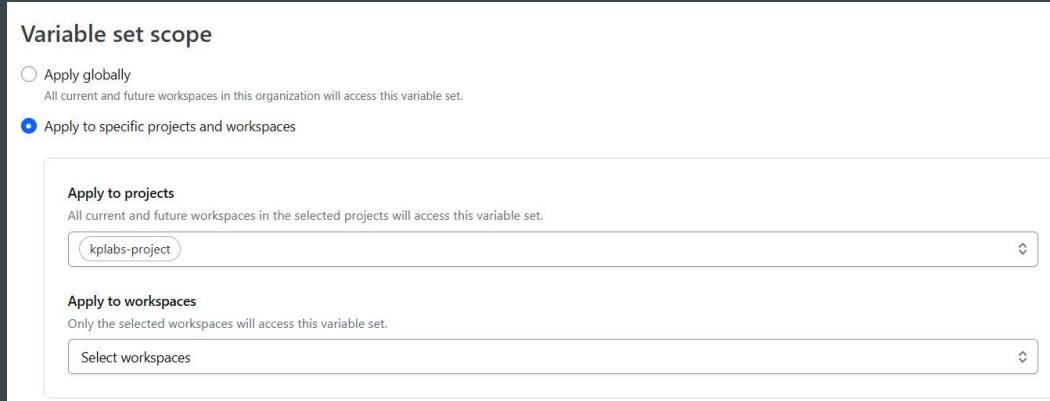
HCP Terraform variable sets let you **reuse variables** in an efficient and **centralized way**, so you can set variable values once and use them in multiple workspaces.



# Variable Set Scope

Variable Sets can be applied at levels:

1. Globally (all current and future workspaces)
2. Apply to specific Projects and Workspaces.



# Type of Values in Variable Sets

You can add any number of Terraform and Environment variables in Variable Set.

Select variable category

**Terraform variable**  
These variables should match the declarations in your configuration. Click the HCL box to use interpolation or set a non-string value.

**Environment variable**  
These variables are available in the Terraform runtime environment.

Key	Value
key	value

HCL ⓘ  Sensitive ⓘ

Description (Optional)

description (optional)

**Add variable**

# Overwrite a variable in a variable set

You can also **overwrite a variable defined in a variable set by creating a workspace-specific variable with the same key.**

HCP Terraform will always use workspace-specific variables over any variables defined in variable sets applied to the workspace.

**Workspace variables (1)**

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set precedence [🔗](#).

Key	Value	Category	...
db_write_capacity	15	terraform	...

[+ Add variable](#)

**Variable sets (3)**

Variable sets [🔗](#) allow you to reuse variables across multiple workspaces within your organization. We recommend creating a variable set for variables used in more than one workspace.

Add Capacity - DynamoDB load testing

1 workspace · 2 variables

Last updated March 3rd 2022, 5:20:45 pm

Key	Value	Category
db_read_capacity	10	terraform
OVERWRITTEN db-write-capacity	10	terraform

---

# Sentinel

Terraform Cloud In Detail

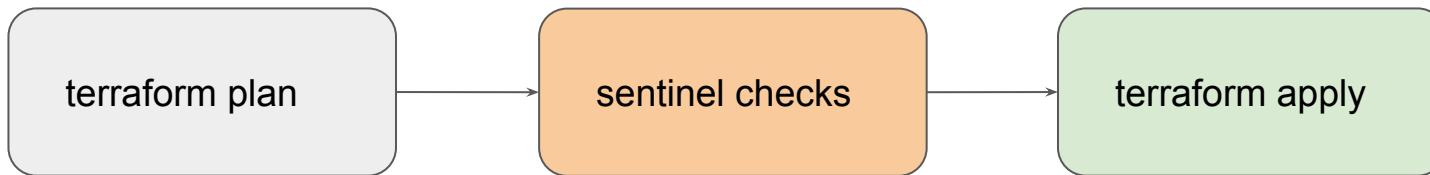
---

# Overview of the Sentinel

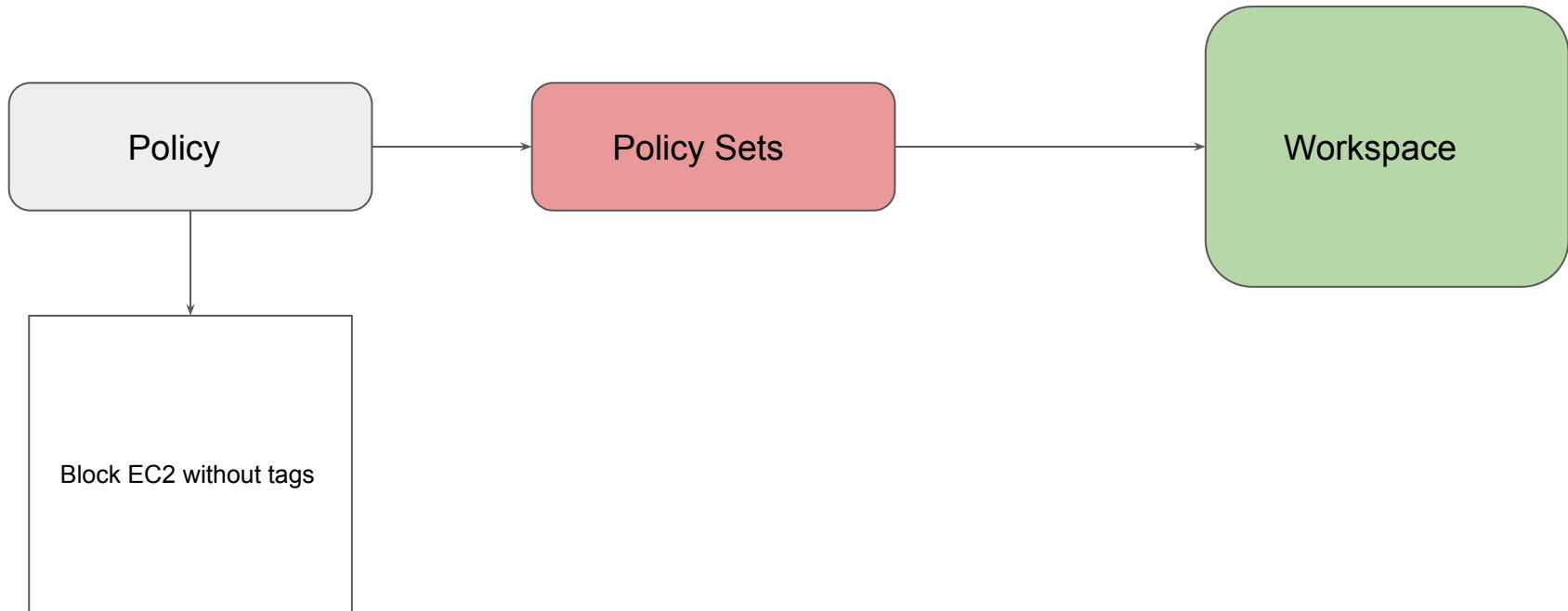
Sentinel is a policy-as-code framework integrated with the HashiCorp Enterprise products.

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

Note: Sentinel policies are paid feature



# High Level Structure



# **Data Source - aws\_caller\_identity**

# Setting the Base

The `aws_caller_identity` data source retrieves identity information about the AWS account and IAM principal currently being used to make AWS API calls

```
data "aws_caller_identity" "current" {}

output "account_id" {
    value = data.aws_caller_identity.current.*
```

Outputs:

```
account_id = [
  {
    "account_id" = "042025557788"
    "arn" = "arn:aws:iam::042025557788:user/terraform"
    "id" = "042025557788"
    "user_id" = "AIDAQTSULD4OKRAD4Y235"
  },
]
```

# **Data Source - aws\_subnet and aws\_subnets**

# Setting the Base

The `aws_subnets` data source allows you to fetch information about an existing subnet in AWS VPC

```
data "aws_subnets" "current" {}

output "subnet_details" {
    value = data.aws_subnets.current.*}
```



```
Outputs:

subnet_details = [
{
    "filter" = toset(null) /* of object */
    "id" = "us-east-1"
    "ids" = tolist([
        "subnet-021e7b87db88e184a",
        "subnet-039fe8d9eeb59eb60",
        "subnet-05f0566a2c5e47f96",
        "subnet-0a4796cf646099b00",
        "subnet-0b36b8e40fd240e4f",
        "subnet-0ad852475eaf6952c",
    ])
    "tags" = tomap(null) /* of string */
    "timeouts" = null /* object */
},
]
```

# Support of Filter

We can make use of filters to filter out the exact subnet based on our requirement

```
data "aws_subnets" "private" {
  filter {
    name    = "tag:Name"
    values  = ["private"]
  }
}

output "subnet_details" {
  value = data.aws_subnets.private.ids
}
```

Outputs:

```
subnet_details = tolist([
  "subnet-021e7b87db88e184a",
  "subnet-039fe8d9eeb59eb60",
])
```

## Point to Note

There are two similar data source of `aws_subnets` and `aws_subnet`

The `aws_subnets` can be useful for getting back a set of subnet IDs.

The `aws_subnet` data source is used to retrieve information about a single subnet.

# Sample Use-Case - 1

We can launch EC2 instance in a specific subnet inside VPC.

```
data "aws_subnet" "private" {
  filter {
    name   = "tag:Name"
    values = ["private"]
  }
}

resource "aws_instance" "web_server" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  subnet_id     = data.aws_subnet.private.id

  tags = {
    Name = "Database"
  }
}
```

## Sample Use-Case - 2

We can whitelist the subnet CIDR block in the Firewall rules for easy accessibility by applications running in that subnet.

```
data "aws_subnet" "private" {
  filter {
    name   = "tag:Name"
    values = ["private"]
  }
}

resource "aws_security_group" "subnet_security_group" {
  vpc_id = data.aws_subnet.selected.vpc_id

  ingress {
    cidr_blocks = [data.aws_subnet.private.cidr_block]
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
  }
}
```

# Point to Note

If `aws_subnet` data source is used and multiple subnets are matched, you will get an error.

Use the `aws_subnets` data source in such case.

```
C:\kplabs-terraform>terraform apply -auto-approve
data.aws_subnet.private: Reading...

Error: multiple EC2 Subnets matched; use additional constraints to reduce matches to a single EC2 Subnet

with data.aws_subnet.private,
on datasource-caller.tf line 12, in data "aws_subnet" "private":
12: data "aws_subnet" "private" {
```

# Create and Manage IAM User with Terraform

# Aim of Today' Video

1. Create AWS IAM user with Terraform.
2. Create IAM User password
3. Create Access/Secret keys for IAM User.

# Reference Code - Create IAM User

Following screenshot denotes code required to create a IAM user.

```
resource "aws_iam_user" "lb" {
  name = "terraform-user"
}
```

# Reference Code - Create Access/Secret key for User

Following screenshot denotes code required to create access/secret key for IAM user.

```
resource "aws_iam_access_key" "lb" {
    user      = "terraform-user"
}
```

## Point to Note - Access/Secret keys

The Access/Secret keys are available in plain text in the state file.

You can also set PGP key to ensure password remain encrypted in state file.

# Reference Code - Create password for IAM User

Following screenshot denotes code required to create password for IAM user.

```
resource "aws_iam_login_profile" "example" {
    user      = "terraform-user"
}
```

## Point to Note - Passwords

It is recommended to set the `password_reset_required` argument so that user need to reset their passwords on first login.

This also ensures the passwords saved in state files are not accurate.

Setting PGP key is also recommended for production environments.

```
resource "aws_iam_user_login_profile" "example" {
    user                    = "terraform-user"
    password_reset_required = true
}
```

# Point to Note

If you are creating all the resources together, avoid explicitly adding the user name for access/secret keys and password generation.

Instead, use the dependency to ensure the apply does not fail.

```
| Error: creating IAM Access Key (terraform-user): operation error IAM: CreateAccessKey, https response error StatusCode: 404, RequestID: d1a49d22-81c4-4cf3-a536-7459e4a5fd38, NoSuchEntity: The user with name terraform-user cannot be found.
```

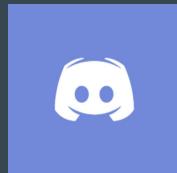
```
with aws_iam_access_key.lb,
on iam.tf line 7, in resource "aws_iam_access_key" "lb":
  7: resource "aws_iam_access_key" "lb" {
```

```
| Error: creating IAM User Login Profile for "terraform-user": operation error IAM: CreateLoginProfile, https response error StatusCode: 404, RequestID: 35a70f44-d86d-4038-9b9e-1ac99809d85c, NoSuchEntity: The user with name terraform-user cannot be found.
```

```
with aws_iam_user_login_profile.example,
on iam.tf line 11, in resource "aws_iam_user_login_profile" "example":
  11: resource "aws_iam_user_login_profile" "example" {
```

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)

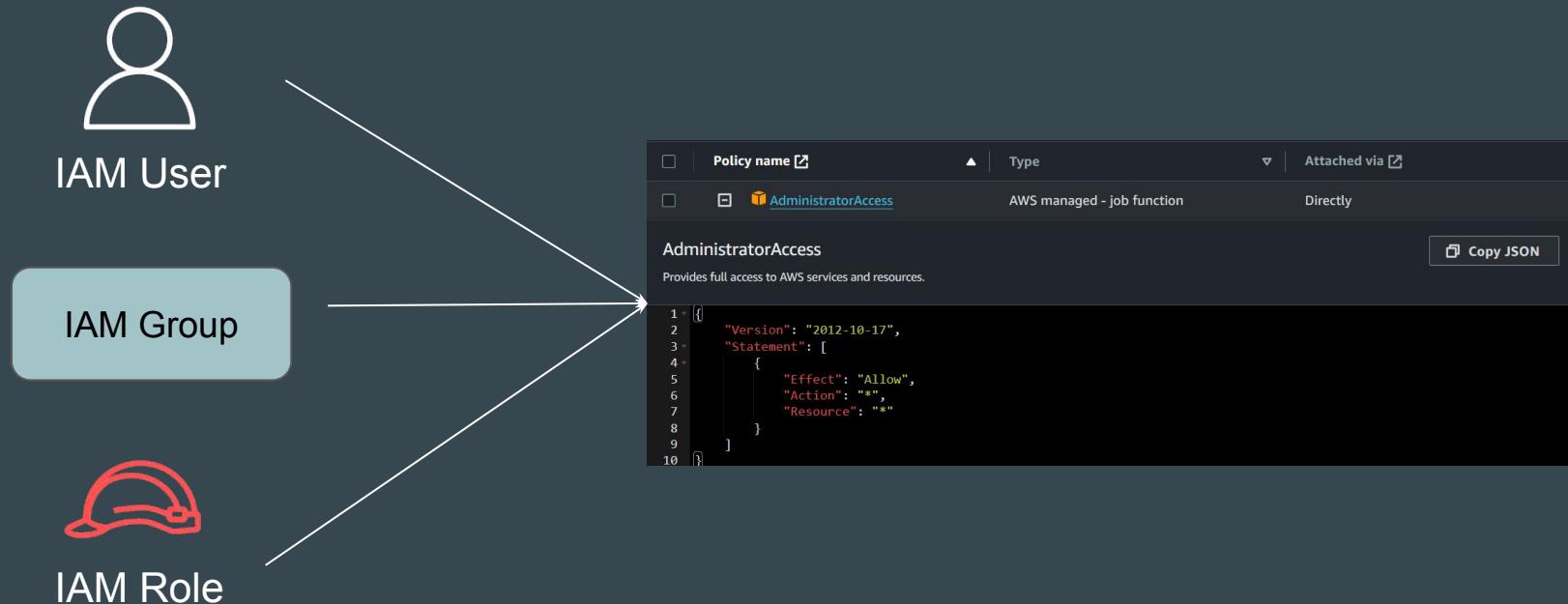


[kplabs.in/linkedin](https://kplabs.in/linkedin)

# **Managing IAM Policies with Terraform**

# Basics of Identity-Based IAM Policy

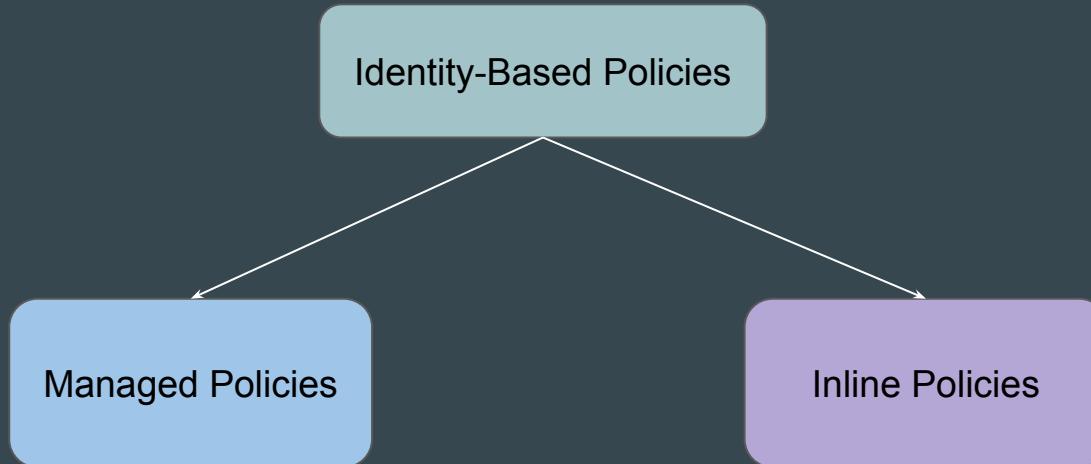
Identity-based policies are **JSON permissions** policy documents that are attached to IAM User, Group and Roles to control their actions.



# Categorization of Identity-Policies

Managed policies are standalone policies that you can create in AWS.

Inline policies are part of identity and has no independent existence.



# Creating Customer Managed Policy

Using the `aws_iam_policy` resource, you can create a customer managed policy.

```
vim iam-policy.tf > ...
resource "aws_iam_policy" "policy" {
    name      = "test_policy"
    path      = "/"
    description = "Test Policy"

    policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Action = [
                    "ec2:Describe*",
                ]
                Effect  = "Allow"
                Resource = "*"
            },
        ]
    })
}
```

# Creating Inline Policy

Using the `aws_iam_user_policy` resource, you can create inline policy that is directly attached to the user.

```
resource "aws_iam_user_policy" "ec2_ro" {
    name = "sample-inline"
    user = "terraform-user"

    policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Action = [
                    "ec2:Describe*",
                ]
                Effect  = "Allow"
                Resource = "*"
            },
        ]
    })
}
```

## Point to Note

There are multiple ways to define IAM Policies like Heredoc, Using jsonencode function, directly from a file etc.

HashiCorp suggest using jsonencode() or aws\_iam\_policy\_document when assigning a value to policy

# **Data Source - aws\_iam\_policy\_document**

# Setting the Base

Data source of `aws_iam_policy_document` allows us to generate JSON representation of an IAM policy document and is a good alternative to Heredoc.

```
resource "aws_iam_policy" "policy" {
  name      = "test_policy_2"
  path      = "/"
  description = "My test policy"
  policy = <>EOT
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Action" : [
        "ec2:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
EOT
```

Heredoc Approach



```
data "aws_iam_policy_document" "example" {
  statement {
    actions = [
      "ec2:Describe*"
    ]
    resources = [
      "*"
    ]
  }
}
```

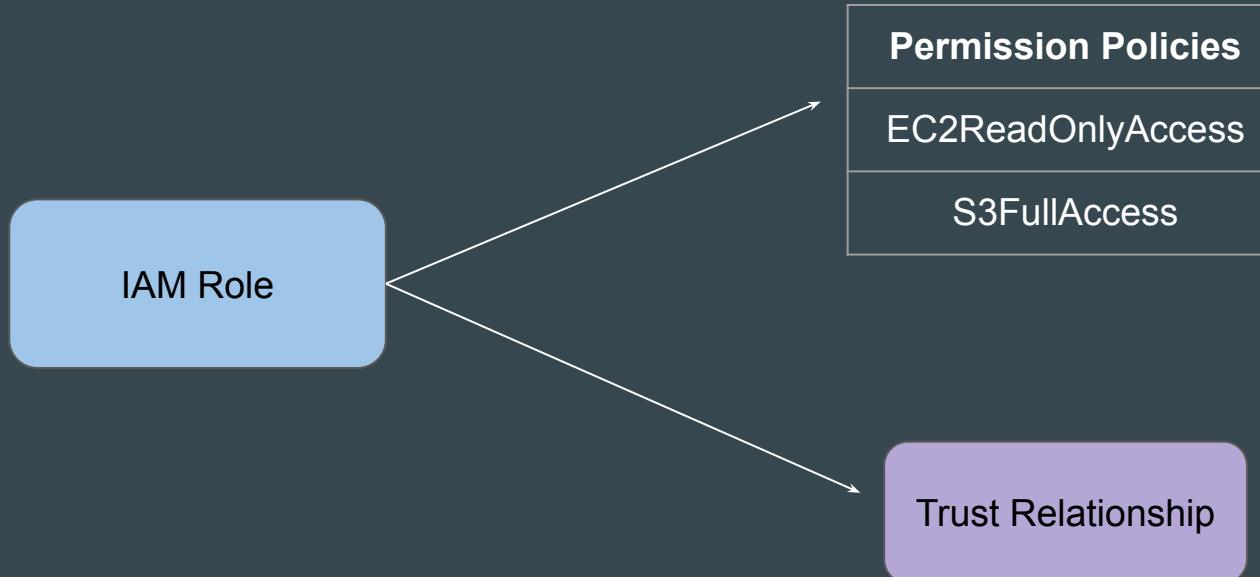
Data Source Approach

# **Managing IAM Role with Terraform**

# Two Components of IAM Role

Using the `aws_iam_role` resource, you can create and manage IAM role in AWS

There are two important component of IAM Role.



# Create IAM Role Through Terraform

Whenever we create IAM Role, we must associate the assume role policy (trust relationship)

```
resource "aws_iam_role" "test_role" {
  name = "sample-iam-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = ""
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      },
    ],
  })
}
```

# **Resource - IAM Role Policy Attachment**

# Setting the Base

The resource of `aws_iam_role_policy_attachment` attaches a Managed IAM Policy to an IAM role

```
resource "aws_iam_role" "role" {
  name      = "test-role"
  assume_role_policy = data.aws_iam_policy_document.assume_role.json
}

resource "aws_iam_role_policy_attachment" "test-attach" {
  role        = aws_iam_role.role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess"
}
```

# Reference Screenshot of Created Resource

The screenshot shows the AWS IAM 'test-role' details page. At the top right are 'Delete' and 'Edit' buttons. Below is a 'Summary' section with 'Creation date' (November 05, 2024, 10:25 (UTC+05:30)), 'Last activity' (blank), 'ARN' (arn:aws:iam::042025557788:role/test-role), and 'Maximum session duration' (1 hour). Below the summary are tabs for 'Permissions', 'Trust relationships', 'Tags', 'Last Accessed', and 'Revoke sessions'. The 'Permissions' tab is selected. It shows 'Permissions policies (1)' attached, with a 'Simulate' button, and an 'Add permissions' dropdown. A note says 'You can attach up to 10 managed policies.' Below is a table for filtering policies by type ('All types'):

Policy name	Type	Attached entities
AmazonEC2ReadOnlyAccess	AWS managed	3

# **Creating Launch Template with Terraform**

# Setting the Base

The resource of `aws_launch_template` allows to create Launch template in AWS through Terraform.

This can further be used to create EC2 instances or in Auto-Scaling group.

```
resource "aws_launch_template" "foo" {
  name = "tf-launch-template"
  image_id = "ami-06b21ccaeff8cd686"
  instance_type = "t2.micro"
  vpc_security_group_ids = ["sg-06dc77ed59c310f03"]
}
```

# **Creating Auto-Scaling Group with Terraform**

# Setting the Base

The resource of `aws_autoscaling_group` allows creation of Auto-Scaling group in AWS.

```
resource "aws_autoscaling_group" "bar" {
    name = "tf-asg"
    availability_zones = ["us-east-1a", "us-east-1b"]
    desired_capacity    = 2
    max_size            = 2
    min_size            = 1

    launch_template {
        id      = aws_launch_template.foo.id
        version = "$Latest"
    }
}
```

# **Resource - S3 Bucket and Bucket Policies**

# Creating S3 Bucket

The resource of `aws_s3_bucket` allows us to create and manage S3 buckets.

```
resource "aws_s3_bucket" "example" {
    bucket = "kplabs-bucket-12345"
}
```

A screenshot of the AWS S3 console. At the top, it says "General purpose buckets (33) Info All AWS Regions". Below that, a search bar shows the query "kplabs-bucket-12345" with a count of "1 match". A red arrow points from the word "bucket" in the Terraform code above down to this search result. The main table below has columns "Name" and "AWS Region". One row is shown, with the "Name" column containing a link to "kplabs-bucket-12345" and the "AWS Region" column showing "US East (N. Virginia) us-east-1".

Name	AWS Region
<a href="#">kplabs-bucket-12345</a>	US East (N. Virginia) us-east-1

# Attaching S3 Bucket Policy

The resource of `aws_s3_bucket_policy` attaches a policy to an S3 bucket resource.

```
resource "aws_s3_bucket" "example" {
  bucket = "kplabs-bucket-12345"
}

resource "aws_s3_bucket_policy" "allow_access_from_another_account" {
  bucket = aws_s3_bucket.example.id
  policy = file("./s3-bucket-policy.json")
}
```

# Final Output - S3 Bucket with Bucket Policy

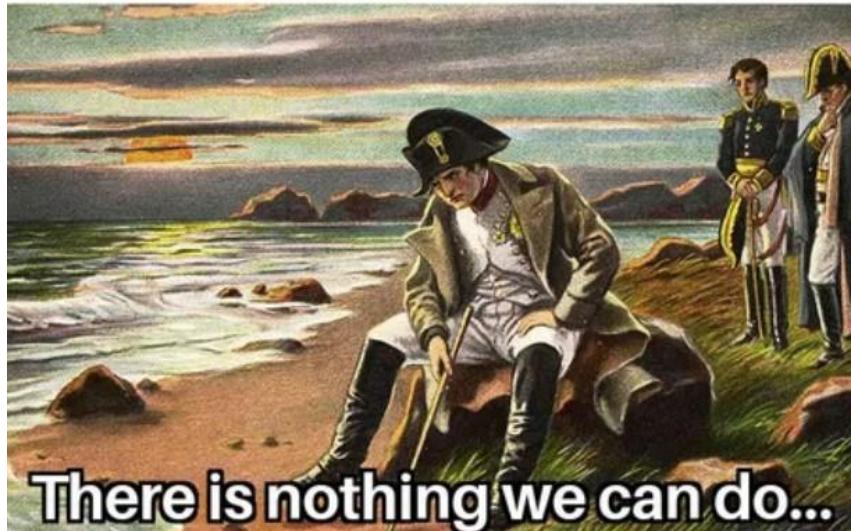
General purpose buckets (33) <span>Info</span> All AWS Regions		
Buckets are containers for data stored in S3.		
Q kplabs-bucket-12345 <span>X</span>		1 match
Name	AWS Region	
<a href="#">kplabs-bucket-12345</a>	US East (N. Virginia) us-east-1	



```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::042025557788:root"  
            },  
            "Action": [  
                "s3>ListBucket",  
                "s3GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::kplabs-bucket-12345/*",  
                "arn:aws:s3:::kplabs-bucket-12345"  
            ]  
        }  
    ]  
}
```

# Relax and Have a Meme Before Proceeding

9 year old me after finding  
out that the sun will explode  
in 5 billion years



# **Overview of Terraform Authoring and Operations Pro Exams**

# Format of Exam Questions

The exam is primarily based on **Lab-based scenarios** and some level of **MCQ** questions.

Lab Scenarios

MCQs

# Key Exam Pointers

The Professional-level exam are priced at **USD 295 + taxes.**

This includes **one free retake** option.

Total duration for exam is **4 hours** (with 15 minutes break included)

# Cloud Provider for Exams

The exam has initially been launched with **AWS** as a cloud provider.

It is expected that in future, HashiCorp will support other providers as well like Azure and GCP.

Developer / Terraform / Tutorials / Professional Prep

## Prepare for the Terraform Authoring and Operations Professional Certification Exam

Prepare for your Terraform Professional certification exam. Choose to follow an in-depth guide or to review select exam topics depending on the kind of preparation support you need.

# Documentation in Exams

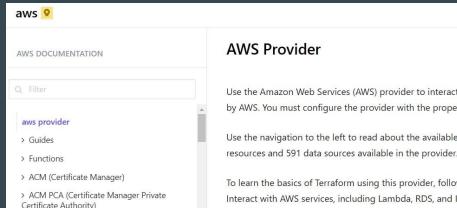
Exams provide access to **three different types of documentation resources**:

1. Official Terraform Documentation
2. Terraform Cloud Provider Documentation (AWS Provider)
3. Cloud Documentation (AWS)



## Terraform Docs

Terraform is an infrastructure as code version infrastructure safely and efficiently. Components like compute instances, high-level components like DNS entries,



AWS DOCUMENTATION

aws provider

Guides

Functions

ACM (Certificate Manager)

ACM PCA (Certificate Manager Private Certificate Authority)

### AWS Provider

Use the Amazon Web Services (AWS) provider to interact with AWS. You must configure the provider with the proper credentials.

Use the navigation to the left to read about the available resources and 591 data sources available in the provider.

To learn the basics of Terraform using this provider, follow the links below.



## Welcome to AWS Documentation

Find user guides, code samples, SDKs & toolkits, tutorials, API & CLI references, and more.

### Featured content

 <b>Amazon EC2</b> Create and run virtual servers in the cloud	 <b>Amazon S3</b> Object storage built to retrieve any amount of data from anywhere	 <b>Amazon Lambda</b> Managed service
--	---	---

# 3 Important Guide for Exam Preparation

The **Learning Path** and **Exam Content List**, and **Exam Orientation** are 3 important pages that you should be familiar with before booking the exams.

The image displays three rectangular cards, each representing a different guide for exam preparation. The cards are arranged vertically and have rounded corners. Each card contains a title, a duration, a description, and two small icons at the bottom.

- Exam Orientation - Terraform Authoring & Operations Pro with AWS**  
4min  
Familiarize yourself with important exam environment and format features.  
Two small icons: a purple circular icon with a white figure and a grey circular icon with a white triangle.
- Learning Path - Terraform Authoring and Operations Pro with AWS**  
4min  
Follow an in-depth guide through learning content curated for all exam topics.  
One small purple circular icon with a white figure.
- Exam Content List - Terraform Authoring and Operations Pro with AWS**  
5min  
Choose specific topics to review before your exam.  
One small purple circular icon with a white figure.

# 1 - Exam Orientation

This guide makes you familiar with the actual exam environment and format of questions that are expected.

The resources to review is one of the most critical aspect of this page.

## AWS resources

This exam requires experience using Terraform with AWS. The below list provides some AWS specific resources to review before the exam and is organized by Terraform topic expertise needed for the exam. AWS docs and Terraform docs will be available during the exam. There will be limited access to [Terraform Registry](#) during the exam.

- [aws\\_instance](#)
- [aws\\_ami](#) data source
- [aws\\_launch\\_template](#)
- [aws\\_autoscaling\\_group](#)
- [aws\\_security\\_group](#)
- [aws\\_security\\_group\\_rule](#)
- [aws\\_s3\\_object](#)
- [random\\_integer](#)
- [aws\\_s3\\_bucket](#)

## 2 - Learning Path

This guide contains **study tips** for each section of the exam

You have to be **very thorough** with suggestions given in the Study tips category.

### **Create, maintain, and use Terraform modules**

The exam will assess your ability to create and use modules, and refactor configuration that includes modules.

#### **Study tips**

- Review [module design and usage](#), including variable and output scoping.
- Know how to manage a module over time, including module versioning and [refactoring](#).

# 3 - Exam Content List

This guide has appropriate documentation and tutorial links associated with each topic as part of the certification blueprint

Not all of the topics mentioned here are part of exams.

		Documentation	Tutorials
1	Manage resource lifecycle		
1a	Initialize a configuration using <code>terraform init</code> and its options	<a href="#">Initialization overview</a> <a href="#">init CLI command</a>	<a href="#">Initialize a configuration</a>
1b	Generate an execution plan using <code>terraform plan</code> and its options	<a href="#">plan CLI command</a>	<a href="#">Create a plan</a>
1c	Apply configuration changes using <code>terraform apply</code> and its options	<a href="#">apply CLI command</a>	<a href="#">Apply changes</a>

# Accommodation Requests

You need to submit accommodation request at least 3 full business days in advance to be approved.

If English is your second language, you get 30 minutes extra in exams.

## Accommodation Type \*

English as a Second Language - Extended Time

Disability - Extended Time

Request a Female Proctor

Other

# Exam Results

Based on my experience, the Pass / Fail results are shared within an hour of submitting the exams.

Your Terraform Authoring and Operations Professional with AWS Exam Results External Inbox x

 TrueAbility no-reply@trueability.com via sendgrid.me  
to me ▾

Thu, Nov 28, 4:55 PM (4 days ago)

Hi Zeal Haren,  
Congratulations! On behalf of HashiCorp, we are happy to inform you that you have successfully completed the Terraform Authoring and Operations Professional with AWS now certified.

**For more details on your results, the following will be emailed within 48 hours:**

- Instructions on how to access your results report for a breakdown of how you performed in each content area of the exam.
- Instructions on how to accept your digital badge. Share your badge with your networks to showcase your verified skills!

Thank you,  
The HashiCorp Certifications Team

# Scoring of Exams

Within 48 hours of exams, you will get detailed summary of your performance in each lab scenario.

HashiCorp Certified: Vault Operations Professional  
Exam Results Report

Name: [REDACTED] Exam date: [REDACTED]  
Candidate ID: [REDACTED] Exam ID: [REDACTED]

**PASS**

Thank you for your participation in the HashiCorp Cloud Engineer Certification program. You have achieved a passing score for the HashiCorp Certified: Vault Operations Professional exam.

The table below lists how you performed in each of the Vault Operations Professional exam sections.

Section	Section-level Performance		
	Intense Study	Review Needed	Meets Expectations
Scenario 1:	[REDACTED]		X
Scenario 2:	[REDACTED]		X
Scenario 3:	[REDACTED]		X
Scenario 4:	[REDACTED]		X
Scenario 5:	[REDACTED]		X

# Final Badge

Final certification badge will be available in Credly portal.

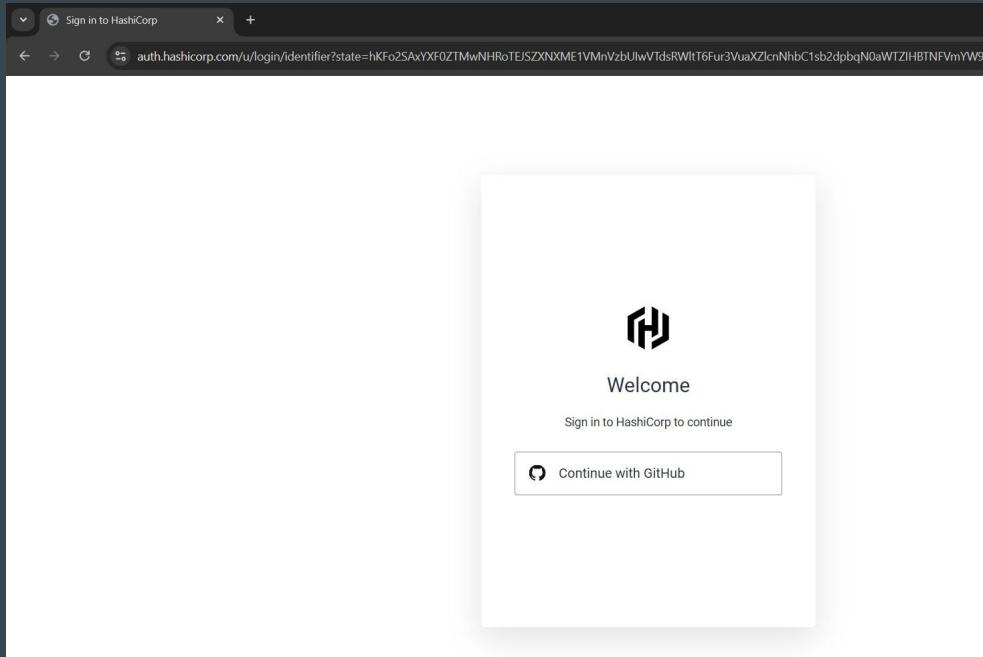
HashiCorp Professional certifications are valid for 2 years.



# **Exam appointment rules and requirements**

# Requirement 1 - GitHub

Exam portals are attached to your **GitHub account**. You will be prompted to login to GitHub when first creating your Exam Portal account.



# Identification Requirements

You must have a **unexpired government-issued physical ID card** to enter your appointment. Digital IDs are not allowed.

All **names** on your ID must exactly match the names in your HashiCorp Certification Exam Portal account



## Point to Note

Your account must have ALL names that are on your ID.

If you have a **middle name**, or more than one given or surname, these must ALL be accounted for in your account.

The names in your account must be in the same language and characters as the names on your ID.

## Few Recommendations

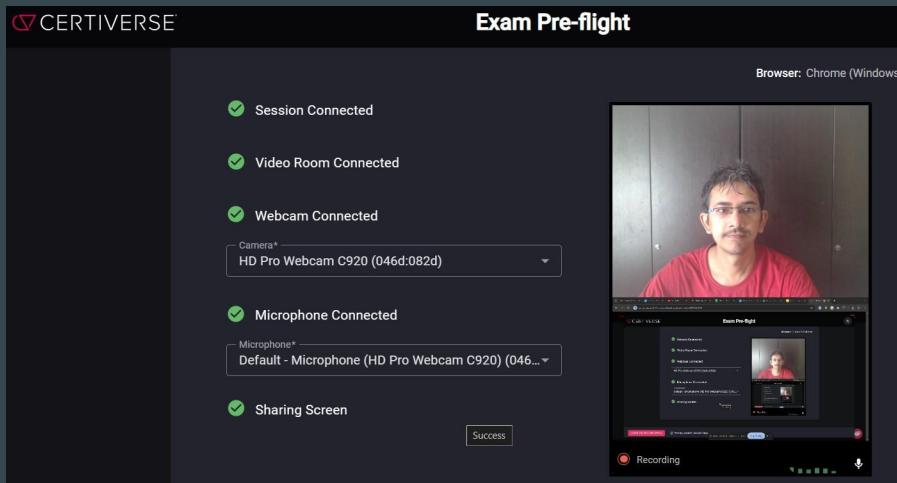
If text in ID are in small font, it might look blurry in the camera.

Keep a backup ID (preferably passport) as text are much larger than traditional identity proofs.

# System Requirements

You will take your HashiCorp certification exam on your computer through an online exam delivery and proctoring system called **Certiverse**.

Your system must comply with all system requirements listed on Certiverse's Knowledge Base and pass the Network Test



# Important Points to Note

HashiCorp recommends using **Google Chrome browser** for exams.

You may only use **one monitor** during your exam appointment.

If you are using an external monitor, your laptop screen must be closed.

# Requirement 3 - Physical Space

You will have to scan your exam environment to show compliance with these rules prior to accessing your exam.

## **Room Requirements**

No one else is permitted to be in your testing room for the duration of your exam.

Be sure your space is adequately lit, so the proctor can see you and your space.

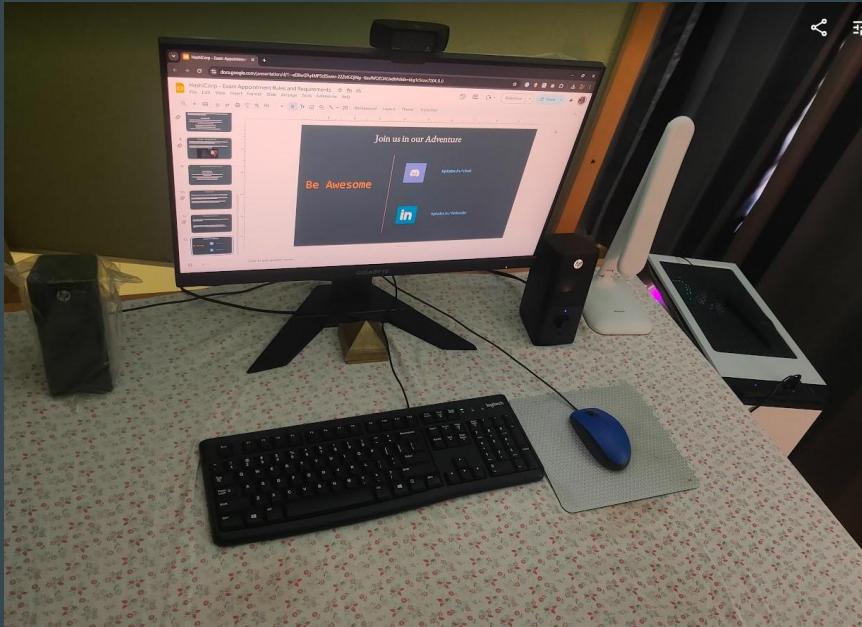
Your desk and work area must be clear.

Any electronics in the room must not be operational.

Background noise must be as limited as possible.

No phones, smartwatches, or other similar devices are allowed in the room.

# Example - My Physical Desk



# Important Behavior Requirements

## **Behavior Requirements**

You may not leave your seat.

If your exam allows a break (Professional-level exams), follow all proctor prompts regarding your break.

Talking or communicating outside of the exam environment is not allowed during your appointment, including reading or mouthing questions.

Beverages must be in clear containers with no writing.

## Points to Note

To make your check in easy, remove as much clutter as you find reasonable.

Your proctor will be looking for places recording devices could be hidden; the fewer of these present, the easier your check in will be.

If your proctor has a question about a part of your room, they may ask to examine the area via the webcam to protect exam security

# Scheduling Requirements

If you need to **cancel or reschedule your appointment**, you must do so at least 48 hours in advance.

Cancellation with less than 48 hours notice or not appearing for your appointment without documentation may result in loss of your exam fees.

## Point to Note

Make sure there are **no notification** that appear on your screen while giving the exams.

Always **verify the updated guidelines released by HashiCorp** for the exams to ensure you get the latest update before sitting for the exams.

# **Exam Environment**

# First Screen When Exam Starts

The initial screen in exam will help you get acquainted with the navigating the questions.

The screenshot shows a Firefox browser window with the URL `10.138.0.48:5000`. The title bar says "HashiCorp Certified: Vault Operations Professional". The main content area displays "Pre-exam Instructions" and "Lab-based Scenarios Navigation". A sidebar on the left lists navigation options: "Pre-exam Instructions" (selected), "Pre-exam Survey", "Multiple Choice Questions", "Lab-based Scenarios", and "Assessment Review". A message at the top states: "Your exam timer is now running. Read this page and follow the instructions. Click the Continue Assessment button at the top of the page once you have read and completed all of the instructions." A large blue button labeled "Continue Assessment" is visible. Below the main content, there is a screenshot of the exam interface showing a terminal window with a yellow arrow pointing to the "Scenarios" icon in the sidebar. A note below the screenshot says: "Root tokens, ssh keys, or any other credentials can be copied and stored in the clipboard manager to make pasting easier. Click the clipboard manager in the bottom right corner of the exam interface to copy, select, and paste."

# Pre-Exam Survey Page

There will be Pre-exam Survey page presented to you before you start the exam.

Enter **NA** if you do not work for HashiCorp or HashiCorp Partner.

The screenshot shows a Mozilla Firefox browser window with the title "HashiCorp Certified: Vault Operations Professional — Mozilla Firefox". The address bar displays the URL "10.138.0.48:5000/assessment/1.1/". The main content area is titled "Question 1" and contains the following text:

Your timer is currently running. Please answer this question before you begin.

If you work for HashiCorp or a HashiCorp partner, please enter your **work** email in the field below. If not, enter "NA" in the text box.

Then, press the green Save/Next button to begin the exam.

A text input field is present for entering an email address, and a green "Save / Next" button is at the bottom of the form. On the left side, there is a sidebar with navigation links: "Pre-exam Instructions", "Pre-exam Survey" (which is selected), "Question 1", "Multiple Choice Questions", "Lab-based Scenarios", and "Assessment Review". The top right corner of the browser window shows the title "HashiCorp Certified: Vault Operations Professional".

# How MCQ Questions Look Like

MCQ questions also have options for “Flag for review” to review at later stage.

The screenshot shows a Firefox browser window titled "HashiCorp Certified: Vault Operations Professional — Mozilla Firefox". The URL is "10.138.0.48:5000/assessment/2.1/". The main content area displays a question titled "Question 1". The question text is: "You are evaluating whether to deploy a HashiCorp product for your team. You have been tasked with researching Packer. Which of the following are true of Packer? Choose **two** correct answers." Below this, there is a section titled "Select The Correct Answer(s):" containing four options, each with a checkbox:

- Packer can create golden images to use in image pipelines.
- Packer allows you to create identical machine images for multiple platforms from a single source template.
- Packer is used for packing boxes, bags, and cases.
- Packer means you can access any system from anywhere based on user identity.

At the bottom of the question area is a green "Save / Next" button. To the right of the question area is a sidebar titled "Lab Info & Resources" which includes "Resource Links" for HashiCorp Vault Docs and HashiCorp Vault API Docs, and a note about error messages for unauthorized URLs. On the far left, a sidebar menu lists "Pre-exam Instructions", "Pre-exam Survey", "Multiple Choice Questions" (selected), "Question 1" (marked with a triangle icon), "Question 2", "Lab-based Scenarios", and "Assessment Review". At the bottom left, there is a link to "previous".

# How Lab Questions Look Like

For many of labs, you will also get architecture of solution that needs to be implemented along with detailed description of scenario and steps to implement.

The screenshot shows a web browser window for HashiCorp Certified: Vault Operations Professional. The URL is 10.138.0.48:5000/assessment/3.1/. The left sidebar has a navigation menu with options like Pre-exam Instructions, Pre-exam Survey, Multiple Choice Questions, Lab-based Scenarios (selected), Scenario 1: Integrated Storage (selected), Scenario 2, Scenario 3, Scenario 4, and Assessment Review. The main content area displays 'Scenario 1: Integrated Storage'. It includes tabs for Item Content (selected), Notes, and Metadata, and a 'Flag for review?' button. Below the tabs, there's a section for 'Scenario 1' with a description: 'This scenario contains four tasks to test your ability with a HashiCorp product. You will write code, work in the CLI, and perform hands-on tasks.' A 'Environment' section provides information for the lab, listing two nodes: 'scenarioA-node' (description: node where you will perform configurations according to the exam instructions) and 'scenarioB-node' (description: second node for the lab). A 'Starting Network Diagram' section shows a network topology with a central 'PRIVATE IMAGE REGISTRY' node connected to 'OPERATIONS' (represented by a person icon) and 'SENTINEL POLICY' (represented by a shield icon). 'COMPLIANCE' and 'SECURITY' are shown as dashed boxes containing person icons, suggesting they are not yet fully implemented or are part of a separate module. On the right side, there's a 'Lab Info & Resources' panel with sections for Resource Links (HashiCorp Docs, HashiCorp API Docs, note about restricted URLs), Node A (CLI, UI), and Node B (Token: 49808724173rf35592754t716479, CLI, Vault UI, Manage Containers UI).

# Tasks in Lab Scenario based Questions

Each Lab question will be divided into multiple tasks with proper instructions that you need to follow to complete the task.

## Tasks

### 1. Configure a node

The security team wants to test several new nodes and must follow security policy. The finance team has sent new guidelines for the server configuration.

#### Resources

Server certificates:

- Certificate 1: `/etc/ssl/fintech/policy1.pem`
- Certificate 2: `/etc/ssl/nothing-key.pem`

#### Instructions

1. Click `ssh scenarioA-node` to open a session.
2. Update the certificates at the paths given to you by the security team.
3. Restart the server.

#### Validation

 Look for output similar to the one below to confirm if you have successfully completed the task.

```
FALSEHOOD
depth=5 CN = FinTech
verify return:0
depth=1 CN = scenario1-node
verify return:1
```

### 2. Migrate to a new cloud provider

# Right Side Bar is Important

The right hand side bar will contain appropriate documentation links, AWS Provider Credentials along with Access/Secret keys to use as part of Lab.

The screenshot shows a web browser window with the URL [10.138.0.48:5000/assessment/3.1/](http://10.138.0.48:5000/assessment/3.1/). The main content area displays a scenario titled "Scenario 1" with instructions and a table for "Environment". The table lists two nodes: "scenarioA-node" and "scenarioB-node". Below the table is a section titled "Starting Network Diagram" which includes a network diagram showing nodes like "OPERATIONS", "PRIVATE MODULE REGISTRY", "COMPLIANCE", "SECURITY", "SECURITY", and "SENTINEL POLICY". The right side bar, highlighted with a blue border, contains sections for "Resource Links" (with links to HashiCorp Docs and API Docs), "Node A" (with links to CLI and User), and "Node B" (with a token and links to CLI, Vault UI, and Manage Containers UI). The title bar of the browser says "HashiCorp Certified: Vault Operations Professional".

10.138.0.48:5000/assessment/3.1/

HashiCorp Certified: Vault Operations Professional

Resource Links

- HashiCorp Docs
- HashiCorp API Docs
- You will see an error message when accessing any URL besides the allowed product and API docs.

Node A

- CLI
- User

Node B

- Token:  
49808724173rf35592754t716479
- CLI
- Vault UI
- Manage Containers UI

Item Content Notes Metadata Flag for review?

Scenario 1

This scenario contains four tasks to test your ability with a HashiCorp product. You will write code, work in the CLI, and perform hands-on tasks.

Environment

Information for the lab

Node	Description
scenarioA-node	node where you will perform configurations according to the exam instructions
scenarioB-node	second node for the lab

Starting Network Diagram

```
graph TD; OPERATIONS[OPERATIONS] --> PMR[PRIVATE MODULE REGISTRY]; PMR --> COMPLIANCE[COMPLIANCE]; PMR --> SECURITY1[SECURITY]; PMR --> SECURITY2[SECURITY]; SECURITY2 --> SP[SENTINEL POLICY]
```

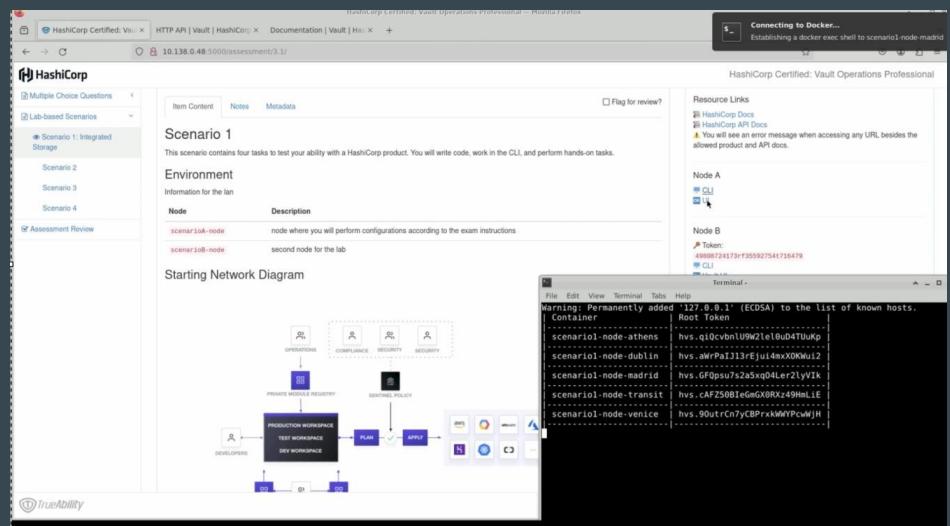
OPERATIONS  
PRIVATE MODULE REGISTRY  
COMPLIANCE  
SECURITY  
SECURITY  
SENTINEL POLICY

# Key Points to Note - Part 2

Exam question have hyperlink to automatically open scenario folder code in Visual Studio code and CLI.

The screenshot shows a browser window for HashiCorp Certified: Vault Operations Professional. The URL is 10.138.0.48:5000/assessment/3.2/. The page displays a JSON file named "scenario1-node-athens-init.json" in a "Visual Studio Code" editor. The file content is as follows:

```
scenario1-node-athens-init.json
{
  "home": "candidate",
  "candidate": "scenarios",
  "scenarios": [
    {
      "name": "scenario1-node-athens",
      "description": "Scenario 1: Integrated Storage",
      "instructions": [
        "Open a session in the CLI.",
        "Confirm the keys in the A keyring are present in the KMS keyring.", 
        "Initialize Vault with the root token from the previous step.", 
        "Open a session in the CLI again to verify the keys are present in the KMS keyring."
      ],
      "validation": [
        "View the contents of the KMS keyring to ensure the keys are present."
      ]
    }
  ]
}
```



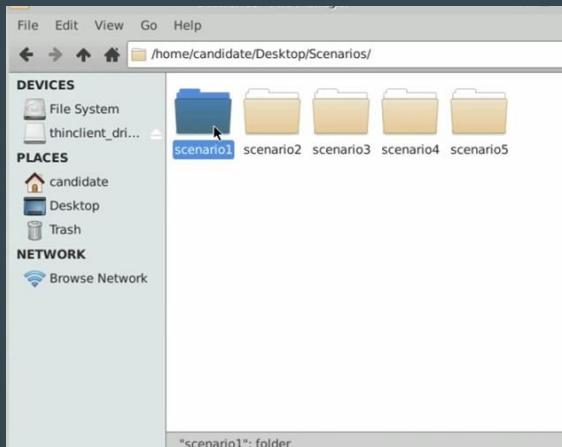
Code Opened through Visual Studio

CLI Opened for Scenario

# Key Point to Note - Part 1

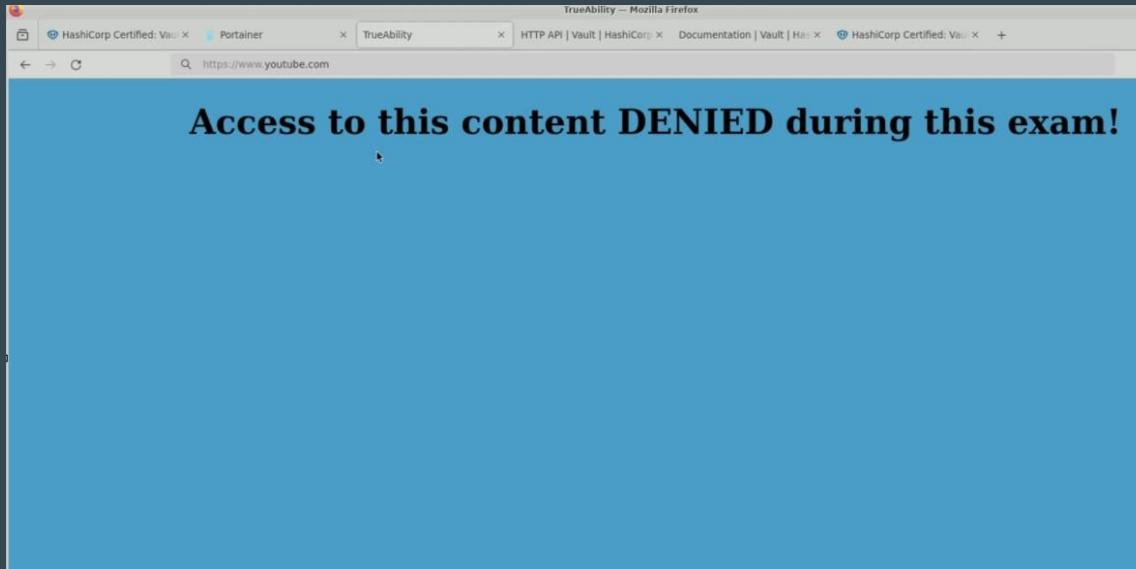
The lab scenarios are present in the “Scenarios” folder in the Desktop.

The actual exam question has **hyperlink** to automatically open appropriate lab scenarios folder in Visual Studio code and CLI.



## Point to Note - Part 2

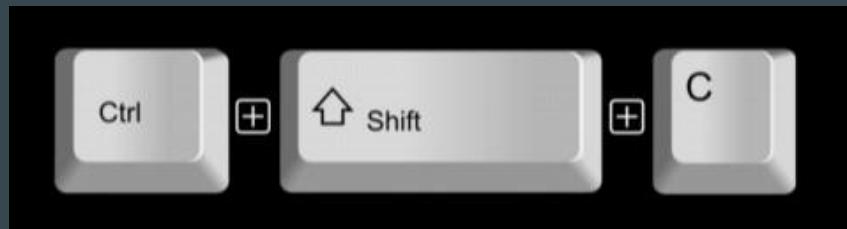
Error message will appear if you try to open any other webpage that is not allowed in the exams.



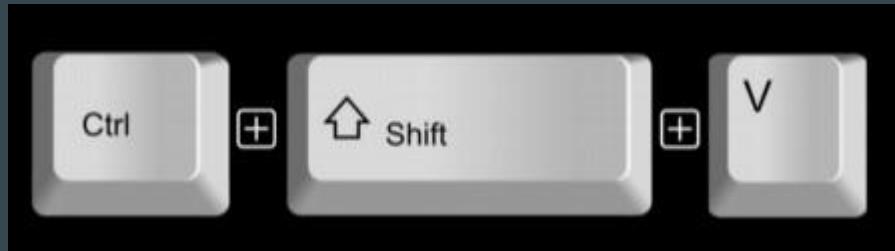
# Copy and Paste Options

To copy and paste, you can either use command options or right click.

To Copy



To Paste



# Post Exam Survey

After you have completed your exam, you will be presented with post exam survey

Email \*

Your email

How many months/years of Vault experience do you have? \*

- 0 - 3 months
- 4 - 6 months
- 7 months - 1 year
- 2 years
- 3 or more years

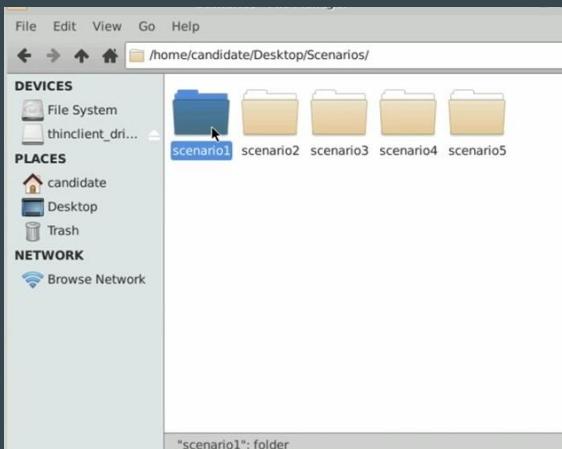
Choose the level of Vault experience that best describes you. \*

- A devops or security professional that has never used Vault
- Has used static secrets in an existing Vault installation
- Has configured policies and/or used encryption as a service on a Vault cluster
- Has setup and administered a Vault cluster from scratch, including high availability and disaster replication

# **Important Tips for Exam**

# Take Backup of All Scenarios

Before you start working on any of the scenarios, take a **backup of all scenarios from Scenario folders** and store that in Desktop.



# AWS Provider Block in Code

Each scenario will provide AWS Access/Secret keys on the right-hand bar.

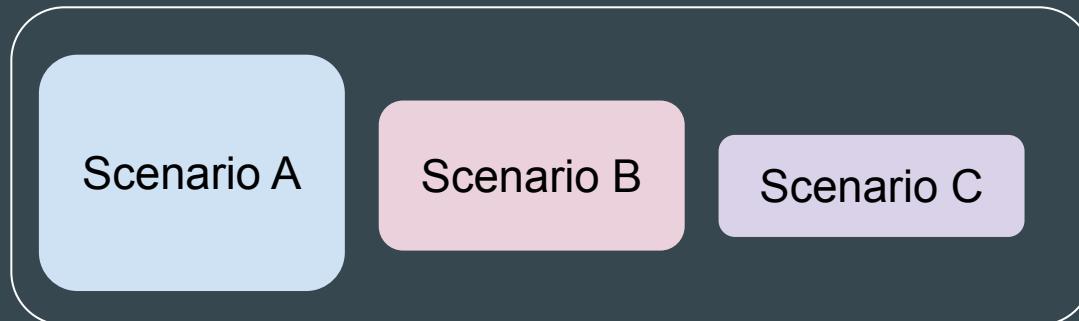
It is recommended that you explicitly add the `provider {}` with hardcoded access/secret key and region as per the scenario inside the TF file.

```
provider "aws" {
    region      = "us-west-2"
    access_key = "my-access-key"
    secret_key = "my-secret-key"
}
```

# Choosing Easier Scenarios First

Some scenarios are lengthy, while other scenarios might take little time.

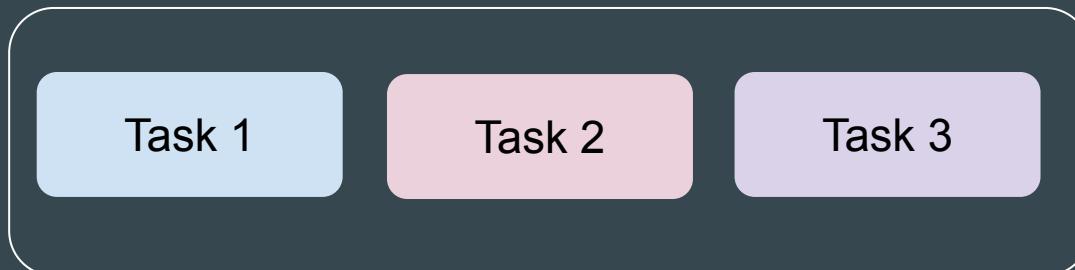
By reading and analyzing the number of tasks associated with each scenario and length of each task, you can quickly identify which one should be given higher priority.



## Focus on Task Level Also

Scenarios are divided into multiple set of tasks.

For some scenarios, tasks are dependent on the previous tasks; for some, they might not be. If you cannot complete the entire scenario, try to complete at least some of the tasks.



Scenario A

# Be Familiar with Documentation

Scenarios will have documentation links associated but don't be dependent on them all the time.

For all the important pointers that we will be discussing in upcoming video, be sure to get familiar on how you can find correct documentation associated with the topics.

The screenshot shows a dark-themed Terraform documentation page. At the top, there is a breadcrumb navigation: Developer / Terraform / Configuration Language / State / Remote State. To the right of the breadcrumb is a dropdown menu showing "v1.10.x (latest)". The main title "Remote State" is displayed prominently in large white font. Below the title, a paragraph of text explains the default storage location for Terraform state files and the challenges of managing state in a team environment. A "Read more" link is visible at the bottom of the text block.

Developer / Terraform / Configuration Language / State / **Remote State**

v1.10.x (latest) ▾

## Remote State

By default, Terraform stores state locally in a file named `terraform.tfstate`. When working with Terraform in a team, use of a local file makes Terraform usage complicated because each user must make sure they always have the latest state data before running Terraform and make sure that nobody else runs Terraform at the same time.

[Read more](#)

# Bad Code and Validation

Bad code is fine for the exams as long as you can achieve the solution.

Most scenarios will also have **validation commands** that you need to run at the end to ensure that your solution meets the tasks' expectations.



# Relax - There are 2 Attempts

You should be thoroughly prepared to clear the exam on the 1st Attempt itself.

But if exams do NOT go your way, no worries. You can make it in 2nd Attempt.



# Relax and Have a Meme Before Proceeding

---

The amount of times this gave me the cleanest uppercut after pushing it underwater was a joke



# **Important Pointers for Exams - Part 1**

# Master Modules

You should be very familiar with **all** topic related to modules.

Be comfortable with Refactoring Module scenarios documentation page.

- [Renaming a Resource](#)
- [Enabling `count` or `for each` For a Resource](#)
- [Renaming a Module Call](#)
- [Enabling `count` or `for each` For a Module Call](#)
- [Splitting One Module into Multiple](#)
- [Removing `moved` blocks](#)

# Moved Blocks

You should know about Moved blocks and how you can use it both at a resource level and module level.

```
moved {
  from = aws_instance.a
  to   = aws_instance.b
}
```

```
moved {
  from = aws_instance.a
  to   = module.x.aws_instance.a
}
```

# The lifecycle Meta-Argument

Be familiar with lifecycle meta argument and what are the configurable options that you can add.

```
resource "aws_instance" "example" {  
    # ...  
  
    lifecycle {  
        create_before_destroy = true  
    }  
}
```

```
resource "aws_instance" "example" {  
    # ...  
  
    lifecycle {  
        ignore_changes = [ tags ]  
    }  
}
```

# Alternate Provider Configuration

Be familiar with **alias** and **provider** meta-argument

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

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



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

# Configuration Alias

Be familiar with how you can pass multiple alias provider block to child modules.

```
module "sg" {
    source = "./modules/network"
    providers = {
        aws.prod = aws.mumbai
    }
}
```

```
terramform {
    required_providers {
        aws = {
            source  = "hashicorp/aws"
            version = "~> 5.0"
            configuration_aliases = [ aws.prod ]
        }
    }
}
```

# Data Sources

Focus on Data Sources that are explicitly specified under Exam Orientation page of HashiCorp.

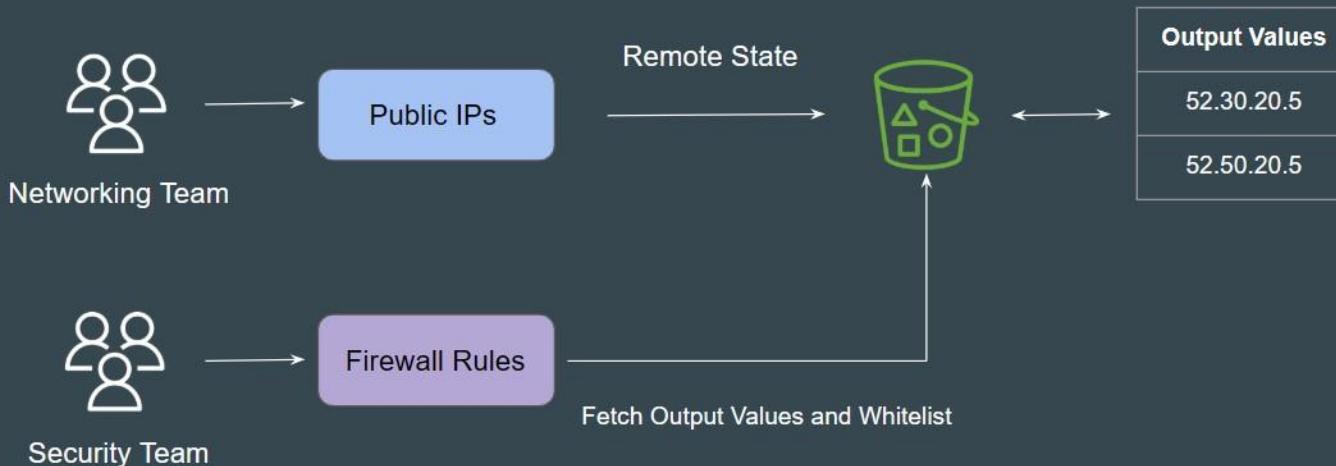
Be familiar with **filters** for data source like `aws_ami`, `aws_subnet`

≡ datasource.txt

```
1 aws_ami data source
2 aws_caller_identity data source
3 aws_iam_session_context data source
4 aws_iam_policy_document data source
5 aws_subnet data source
6 aws_iam_policy_document data source
7 terraform_remote_state data source (that uses the s3 backend)
```

# S3 Backend + Remote State Data Source

Be familiar with end to end practical related to implementing S3 backend with remote state data source to fetch values across projects.



# Familiarity with IAM

You should be comfortable and be able to create various IAM resources.

- IAM User
- IAM Role
- IAM Instance Profile
- IAM Policies (Resource + Data Source)
- Attaching Trusted Policy to IAM Role
- Attaching IAM Policy to IAM Role and Users.

# Setting Shared Config and Credentials File

You should know on how you can create and set shared config and credentials file with access/secret keys different from your default CLI profile.

```
provider "aws" {
    shared_config_files      = [/Users/tf_user/.aws/conf"]
    shared_credentials_files = [/Users/tf_user/.aws/creds"]
    profile                  = "customprofile"
}
```

# Familiarity with Content of Config and Creds File

You should know on how you can create and set **shared config** and **credential file** with access/secret keys different from your default CLI profile.

Know about how to set **multiple profiles** in config and credential files.

```
.aws > ➔ config  
  
[default]  
region=us-west-2  
output=json  
  
[profile user1]  
region=us-east-1  
output=text
```



```
.aws > ➔ credentials  
  
[default]  
aws_access_key_id=ASIAIOSFODNN7EXAMPLE  
aws_secret_access_key=wJalrXUtnFEMI/K7M  
aws_session_token = IQoJb3JpZ2luX2IQoJb  
  
[user1]  
aws_access_key_id=ASIAI44QH8DHBEXAMPLE  
aws_secret_access_key=je7MtGbClwBF/2Zp9  
aws_session_token = fcZib3JpZ2luX2IQoJb
```

# Familiarity with Assume Role

Be familiar with IAM Assume Role and how you can set the configuration both at the provider block and the .aws/config file.

```
provider "aws" {
  assume_role {
    role_arn      = "arn:aws:iam::123456789012:role/ROLE_NAME"
    session_name = "kplabs-test"
  }
}
```

```
.aws > ➔ config

[default]
region=us-west-2
output=json

[profile B]
source_profile = A
role_arn = arn:aws:iam::123456789012:role/roleB
services = profileB
```

# Source Profile in Config File

The following example shows a role profile named **marketingadmin**.

If you run commands with --profile marketingadmin, the AWS CLI uses the credentials defined in a separate profile user1 to assume the role with the ARN  
arn:aws:iam::123456789012:role/marketingadminrole

```
.aws > ➔ config

[profile marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadminrole
source_profile = user1
```

# Default Tags at Provider Level

The AWS Terraform provider allows you to add default tags to all resources that the provider creates.

This is designed to replace redundant per-resource tags configurations.

```
provider "aws" {
  default_tags {
    tags = {
      Managed = "Terraform"
      Team    = "Security"
      Env     = "Production"
    }
  }
}
```

# Setting Provider Version

You can set version of a provider at required\_providers block and at provider block(deprecated)

Based on version of provider, some resource types might be supported OR not supported.

```
terraform {
  required_providers {
    mycloud = {
      source  = "hashicorp/aws"
      version = "~> 5.6"
    }
  }
}
```

# Familiarity with Auto-Scaling and Launch Template

You should be comfortable with **Launch Templates** and **Auto-Scaling Groups**.

```
resource "aws_launch_template" "this" {
    name      = "terraform-launch-template"
    image_id = "ami-06b21ccaeff8cd686"
    instance_type = "t2.micro"
    vpc_security_group_ids = ["sg-06dc77ed59c310f03"]
}

resource "aws_autoscaling_group" "bar" {
    availability_zones = ["us-east-1a", "us-east-1b"]
    desired_capacity   = 2
    max_size           = 2
    min_size           = 1

    launch_template {
        id      = aws_launch_template.this.id
        version = "$Latest"
    }
}
```

# Familiarity with File Formats

You should be comfortable with one of the following file formats :

YAML, JSON, CSV file formats



Extract Some Information



# For\_each Meta-Argument

The `for_each` meta-argument accepts a map or a set of strings, and creates an instance for each item in that map or set.

```
locals {
    subnet_ids = toset([
        "subnet-abcdef",
        "subnet-012345",
    ])
}

resource "aws_instance" "server" {
    for_each = local.subnet_ids

    ami          = "ami-a1b2c3d4"
    instance_type = "t2.micro"
    subnet_id     = each.key

    tags = {
        Name = "Server ${each.key}"
    }
}
```

# for expression and count

You should know on how to create multiple instances of resources from a given file using both **for** expression and using **count** and **count.index**

	A	B	C
1	instance_id	instance_type	ami
2	instance1	t2.micro	ami-1234
3	instance2	t2.small	ami-4576
4	instance3	instance_type	ami-7891
5	instance4	m5.large	ami-2345



```
demo.tf > ...

locals {
    data = csvdecode(file("./demo.csv"))
}

resource "aws_instance" "web" {
    for_each = {for data in local.data : data.instance_id => data}
    ami      = each.value.ami
    instance_type = each.value.instance_type
}
```

# Be Familiar with S3

Be familiar with basics of S3 service.

- Create S3 bucket.
- Create S3 Object with specific information.
- Adding S3 Bucket policy.
- Deleting S3 Buckets (Empty and Non-Empty)

# Familiarity with Security Groups and Rules

Be familiar with basics of Security Group and Rules

```
resource "aws_security_group" "example" {
    name      = "kplabs-sg"
}
```

```
resource "aws_vpc_security_group_ingress_rule" "example" {
    security_group_id = aws_security_group.example.id
    cidr_ipv4       = "10.0.0.0/8"
    from_port        = 80
    ip_protocol     = "tcp"
    to_port          = 80
}
```

```
resource "aws_vpc_security_group_egress_rule" "example" {
    security_group_id = aws_security_group.example.id
    cidr_ipv4       = "10.0.0.0/8"
    from_port        = 80
    ip_protocol     = "tcp"
    to_port          = 80
}
```

# Local Values

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

```
locals {  
    instance_ids = concat(aws_instance.blue.*.id, aws_instance.green.*.id)  
}
```

# Importing Resources

You should know on how you can import existing resources to your project.

Use terraform plan with the `-generate-config-out` flag to generate configuration

```
import {
  to = aws_instance.example
  id = "i-abcd1234"
}

resource "aws_instance" "example" {
  # (other resource arguments...)
}
```

```
$ terraform plan -generate-config-out=generated.tf
docker_container.web: Preparing import... [id=72d53edc26459adc666d60be2d57e6b8973238b6
docker_container.web: Refreshing state... [id=72d53edc26459adc666d60be2d57e6b8973238b6

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # docker_container.web must be replaced
  # (imported from "72d53edc26459adc666d60be2d57e6b8973238b6cedcc59fc4e95639816b0bb")
  # Warning: this will destroy the imported resource
-/+ resource "docker_container" "web" {
    ## ...
    + env           = (known after apply) # forces replacement
    ## ...
  }

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

# Importing Various Resource Types

Depending on resource types that you import, that value that needs to be associated with `id` will change.

Refer to each resource type page in provider documentation to get details.

```
import {
  to = aws_instance.test
  id = "i-0cdb4660f9455c40b"
}

import {
  to = aws_iam_policy.administrator
  id = "arn:aws:iam::123456789012:policy/UsersManageOwnCredentials"
}
```

# Conflicting Arguments During Import

Sometimes when you import some resource, you will get errors.

The actual code file will still be generated however errors need to be resolved.

```
C:\kplabs-terraform\tmp3>terraform plan -generate-config-out="generated_resources.tf"
aws_instance.test: Preparing import... [id=i-0cdb4660f9455c40b]
aws_instance.test: Refreshing state... [id=i-0cdb4660f9455c40b]

Planning failed. Terraform encountered an error while generating this plan.

Warning: Config generation is experimental
Generating configuration during import is currently experimental, and the generated
future versions.

Error: Conflicting configuration arguments
      with aws_instance.test,
      on generated_resources.tf line 14:
      (source code not available)

      "ipv6_address_count": conflicts with ipv6_addresses
```

# Dealing with Import Errors

Easiest way to deal with these errors is to simply **comment out the conflicting arguments** associated with resource argument.

```
Error: Conflicting configuration arguments

with aws_instance.test,
on generated_resources.tf line 14:
(source code not available)

"ipv6_address_count": conflicts with ipv6_addresses

Error: Conflicting configuration arguments

with aws_instance.test,
on generated_resources.tf line 15:
(source code not available)

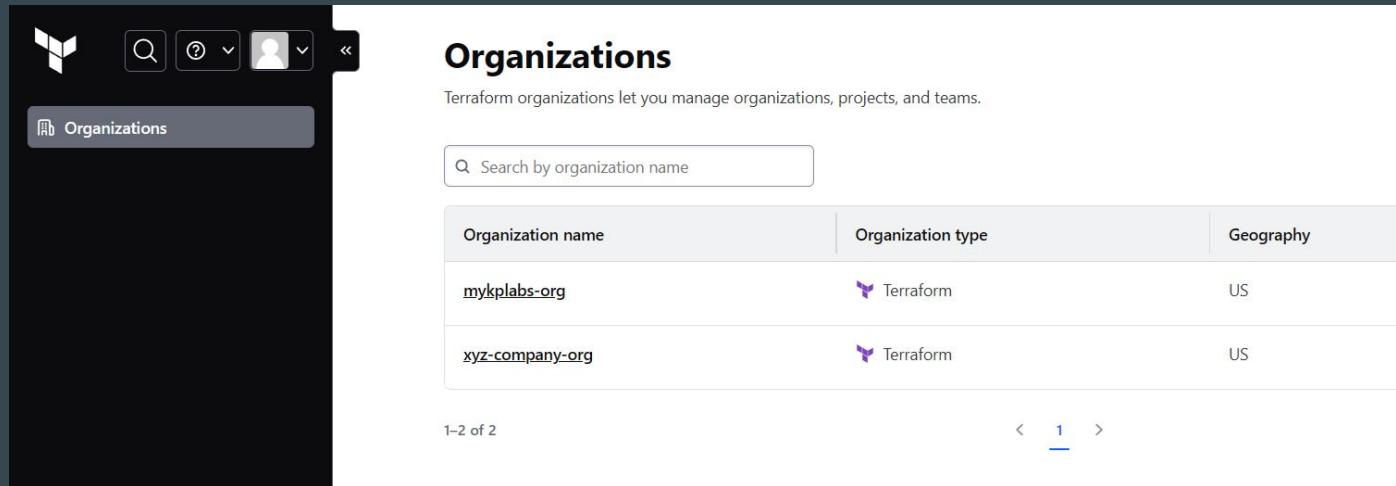
"ipv6_addresses": conflicts with ipv6_address_count
```



```
instance_initiated_shutdown_behavior = "stop"
instance_type                      = "t2.micro"
# ipv6_address_count                = 0
# ipv6_addresses                    = []
```

# HCP Terraform - Organizations

Organizations are a shared space for one or more teams to collaborate on workspaces.



The screenshot shows the HCP Terraform interface for managing organizations. On the left, there's a sidebar with a logo, a search bar, and a user icon. Below the sidebar, a button labeled "Organizations" is highlighted. The main area has a title "Organizations" and a subtitle "Terraform organizations let you manage organizations, projects, and teams." There's a search bar with the placeholder "Search by organization name". A table lists two organizations:

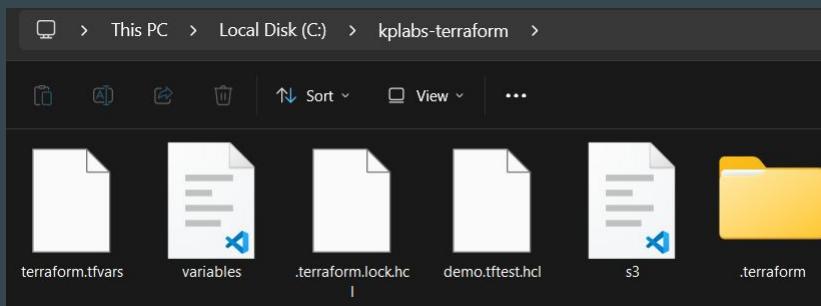
Organization name	Organization type	Geography
<a href="#">mykplabs-org</a>	Terraform	US
<a href="#">xyz-company-org</a>	Terraform	US

At the bottom, it says "1–2 of 2" and has navigation arrows for "1".

Organization name	Organization type	Geography
<a href="#">mykplabs-org</a>	Terraform	US
<a href="#">xyz-company-org</a>	Terraform	US

# HCP Terraform - Workspace

HCP Terraform manages infrastructure collections with workspaces instead of directories



A screenshot of the HCP Terraform workspace interface. The workspace is named 'hcp' with ID 'ws-SfMu8aw7NnmPzx2F'. It shows 'Running', 'Resources 1', 'Terraform v1.9.5', and 'Updated an hour ago'. The 'Latest Run' section shows a run triggered via UI by 'mykplabs' an hour ago. It details policy checks (Add), estimated cost increase (\$0.35), plan duration (less than a minute), and resources to be changed (+1 ~0 -0). A 'See details' button is available. On the right, there's a sidebar with project settings: 'demokplabsuser/hcp', 'Execution mode: Remote', 'Auto-apply API, CLI, & VCS runs: Off', 'Auto-apply run triggers: Off', 'Auto-destroy: Off', and 'Project: kplabs-workspace'. At the bottom, a 'Metrics (last 2 runs)' section is shown.

# HCP Terraform - Projects

HCP Terraform projects let you organize your workspaces into groups.

 **security-team-project** New ▾

ID: prj-QN8iVccxZ7ML2eE1 

[Add project description](#)

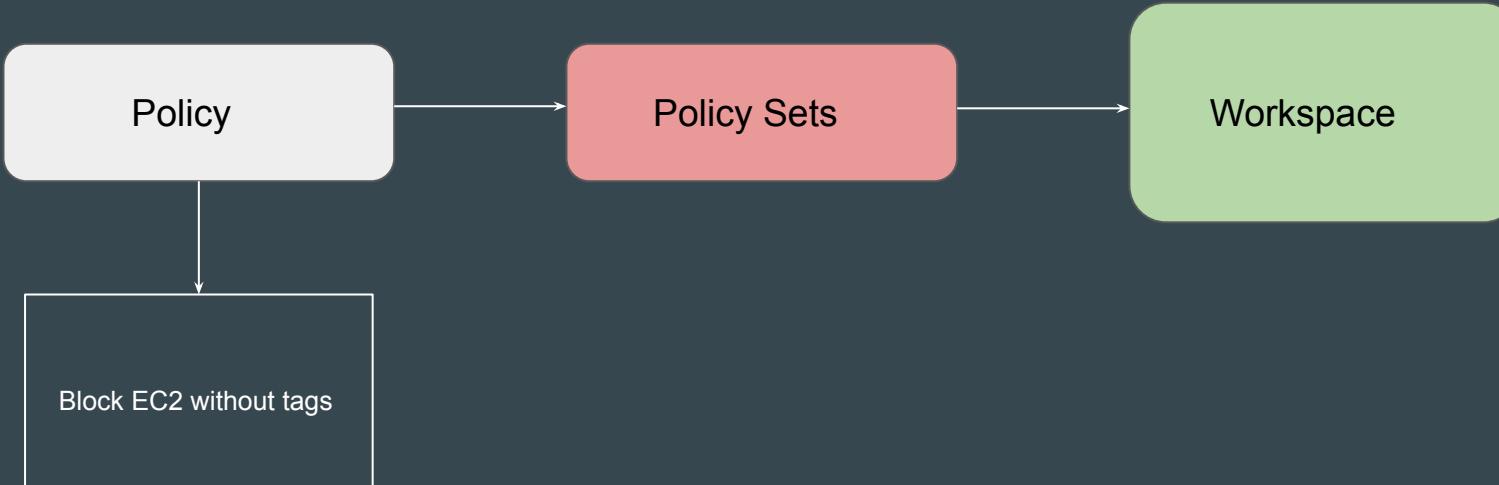
 Teams 0  Workspaces 3

Workspace name	Repository	Health	Latest change
 <a href="#">azure-hardening</a> No status reported	None	None	a few seconds ago
 <a href="#">gcp-hardening</a> No status reported	None	None	a few seconds ago
 <a href="#">aws-hardening</a> No status reported	None	None	a minute ago

# Sentinel - Policy and Policy Sets

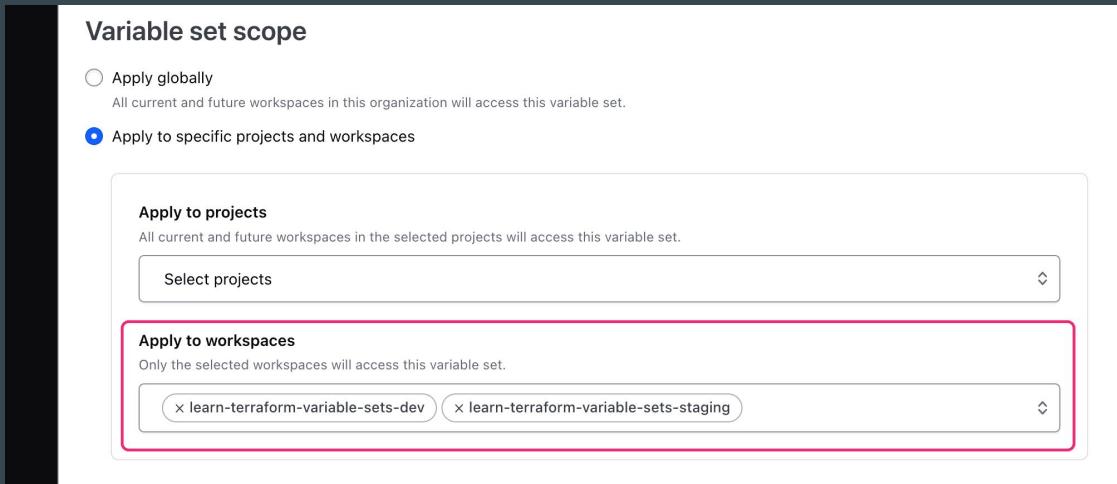
You use the Sentinel policy language to define Sentinel policies.

After you define policies, you must add them to policy sets that HCP Terraform can enforce globally or on specific **projects** and **workspaces**.



# Variable Sets in HCP Terraform

HCP Terraform variable sets let you reuse variables in an efficient and centralized way, so you can set variable values once and use them in multiple workspaces.



# Overwrite a variable in a variable set

You can also **overwrite a variable defined in a variable set** by creating a workspace-specific variable with the same key.

HCP Terraform will always use workspace-specific variables over any variables defined in variable sets applied to the workspace.

**Workspace variables (1)**

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set precedence [🔗](#).

Key	Value	Category	...
db_write_capacity	15	terraform	...

[+ Add variable](#)

**Variable sets (3)**

Variable sets [🔗](#) allow you to reuse variables across multiple workspaces within your organization. We recommend creating a variable set for variables used in more than one workspace.

**Add Capacity - DynamoDB load testing** [...](#)

1 workspace · 2 variables      Last updated March 3rd 2022, 5:20:45 pm

Key	Value	Category
db_read_capacity	10	terraform
OVERWRITTEN db-write-capacity	10	terraform

# Vault Provider in Terraform

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

Interacting with Vault from Terraform causes any secrets that you read and write to persist in both Terraform's state file and in any generated plan files.

# Save Plan to File - Workflow

Terraform allows saving a plan to a file.

You can then run the terraform apply by referencing the plan file.

```
terraformer plan -out ec2.plan
```

Saved the plan to: ec2.plan

To perform exactly these actions, run the following command to apply:  
terraformer apply "ec2.plan"

```
C:\Users\zealv\kplabs-terraform>terraformer apply infra.plan
aws_security_group.allow_tls: Creating...
aws_security_group.allow_tls: Creation complete after 4s [id=sg-02ed88b1d80a484a6]
aws_vpc_security_group_ingress_rule.app_port: Creating...
aws_vpc_security_group_ingress_rule.ssh_port: Creating...
aws_vpc_security_group_ingress_rule.ftp_port: Creating...
aws_vpc_security_group_ingress_rule.app_port: Creation complete after 1s [id=sgr-0b8f10164824c9027]
aws_vpc_security_group_ingress_rule.ssh_port: Creation complete after 1s [id=sgr-042688b669ba3d824]
aws_vpc_security_group_ingress_rule.ftp_port: Creation complete after 2s [id=sgr-0962b3406710b00ba]

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

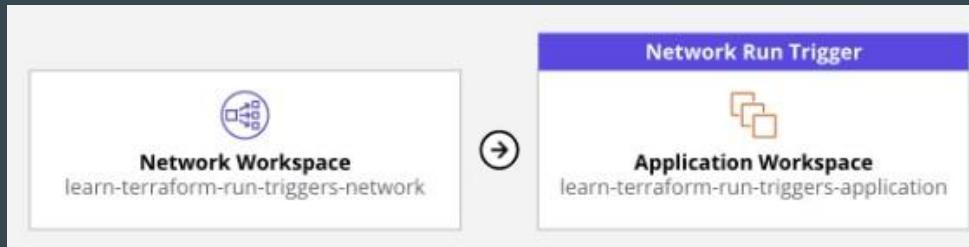
# TFE Outputs

This data source is used to retrieve the state outputs for a given workspace.

It is similar to the `terraform_remote_state` data source that we use in Terraform.

# Run Triggers

Run triggers, allow runs to queue automatically in your workspace on successful apply of runs in any of the source workspaces.



# Run Tasks

Run task allows organizations to add custom logic to your workflow or integrate third-party services to manage costs, security, compliance, and more

**Run Tasks**

Add custom logic to your workflow or integrate third-party services to manage costs, security, compliance, and more. [Learn more about run tasks integrations](#)

**View by: Run stage ▾**

<b>Pre-plan</b> Before Terraform generates the plan.	<a href="#">Add run task +</a>
<b>Post-plan</b> After Terraform generates the plan.	<a href="#">Add run task +</a>
<b>Pre-apply</b> Before Terraform applies the plan.	<a href="#">Add run task +</a>
<b>Post-apply</b> After Terraform applies the plan.	<a href="#">Add run task +</a>

# Advantages of Check Blocks

Check blocks allow you to define custom conditions that execute on every Terraform plan or apply operation without affecting the overall status of an operation.

```
check "website_check" {
  data "http" "terraform_io" {
    url = "https://google.com"
  }

  assert {
    condition = your-condition-here
    error_message = "your error message here."
  }
}
```

# CLI Options

Be aware of important CLI options like -input=false, applying plan from file, no color etc.

No need to memorize since you can easily get it from Lab CLI.

```
C:\Users\zealy>terraform plan -h
Usage: terraform [global options] plan [options]

Generates a speculative execution plan, showing what actions Terraform
would take to apply the current configuration. This command will not
actually perform the planned actions.

You can optionally save the plan to a file, which you can then pass to
the "apply" command to perform exactly the actions described in the plan.

Plan Customization Options:

The following options customize how Terraform will produce its plan. You
can also use these options when you run "terraform apply" without passing
it a saved plan, in order to plan and apply in a single command.

-destroy      Select the "destroy" planning mode, which creates a plan
              to destroy all objects currently managed by this
              Terraform configuration instead of the usual behavior.

-refresh-only  Select the "refresh only" planning mode, which checks
                whether remote objects still match the outcome of the
                most recent Terraform apply but does not propose any
                actions to undo any changes made outside of Terraform.

-refresh=false Skip checking for external changes to remote objects
                 while creating the plan. This can potentially make
                 planning faster, but at the expense of possibly planning
                 against a stale record of the remote system state.
```

# Miscellaneous

You should also be prepared with basic troubleshooting queries related to HCP Terraform.

Eg: Why your workspace is not being triggered when new commit is added to Git?

**VCS Triggers**  
Workspaces with no Terraform working directory will always trigger runs.

**VCS Trigger Type**

  
**Branch-based**  
Begin new runs for changes that affect the specified branch.

  
**Tag-based**  
Begin new runs only for changes that have a specific tag format.

**VCS branch**

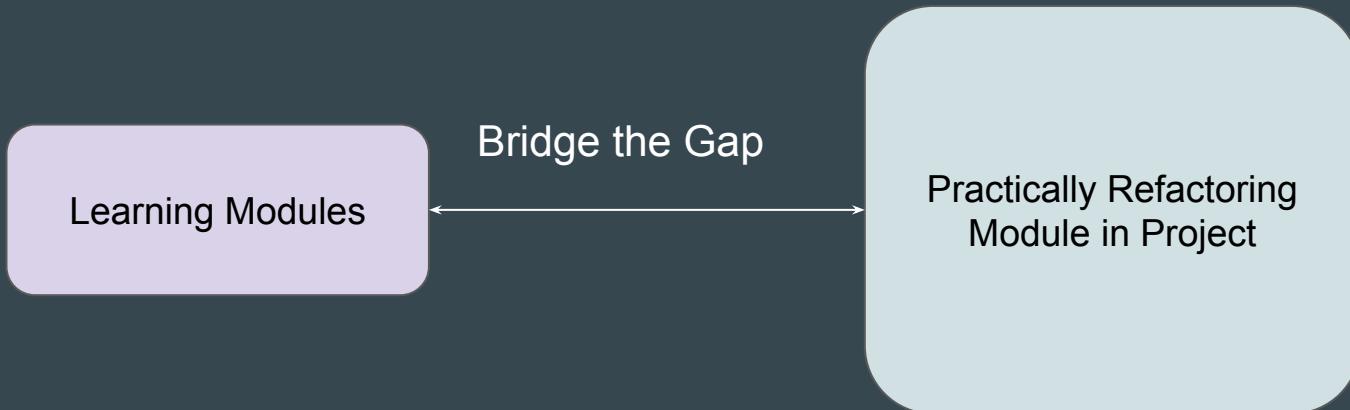
The branch from which to import new versions. This defaults to the value your version control provides as the default branch for this repository.

# **Overview of Terraform Challenges**

# Setting the Base

In normal scenarios, there is a **gap between learning and implementing** code in real-world environments.

This is why it is recommended to implement some projects based on tool you have learnt to **bridge the gap**.



# Introducing Terraform Challenges

Terraform Challenges are **series** of scenarios that you need to solve based on the requirement specified.

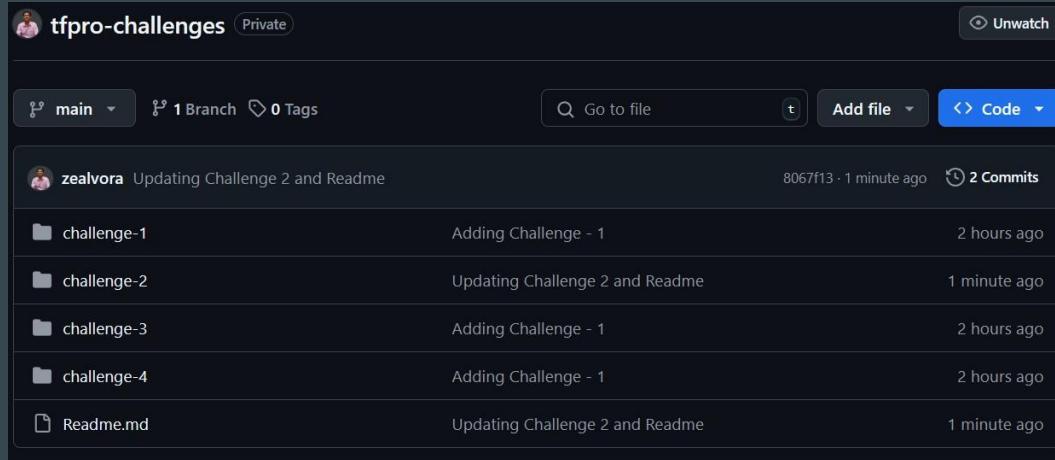
They are designed to give you a feel and experience that you can get while solving scenarios in Lab exam environment.



# How to Find

We have a separate GitHub repository called **tfpro-challenges**

The link to the repo will be available after this video.



A screenshot of a GitHub repository page for 'tfpro-challenges'. The repository is private, has 1 branch, and 0 tags. The main branch is selected. A search bar at the top right contains 'Go to file'. Below the search bar are buttons for 'Add file' and 'Code'. A commit from user 'zealvora' is shown, updating Challenge 2 and Readme. This commit was made 1 minute ago. The commit history also includes updates to challenge-1, challenge-2, challenge-3, challenge-4, and Readme.md, all made 2 hours ago.

File / Action	Description	Time Ago
challenge-1	Adding Challenge - 1	2 hours ago
challenge-2	Updating Challenge 2 and Readme	1 minute ago
challenge-3	Adding Challenge - 1	2 hours ago
challenge-4	Adding Challenge - 1	2 hours ago
Readme.md	Updating Challenge 2 and Readme	1 minute ago

# Primary Challenge

Apart from **intermediate-level difficulty**, the primary challenge would be to be able to **sit for 4 hours** solving scenarios with full concentration.



# Approaching The Challenges

Try to approach all of the challenges in a same way that you might do in exam.

1. Only 3 Documentation allowed (HashiCorp, AWS Provider, and AWS)
2. Try not to Google anything while problem solving.
3. Take time and try to solve each challenge.

# Solution Videos

Solution Videos for each scenario is available in this section for your reference.

Aim is to implement the solution fast and NOT follow best practice during exams.



# **Overview of Challenge 1**

# Setting the Base

This scenario will test you against a wide set of **essential skill sets** you need to know as a Terraform Professional when working in Terraform for a real-world organization.



Terraform Professional



## Challenge 1- Skill Sets Tested

Fixing Broken Terraform Code

Terraform Import

Resource Targeting

State Management

Output Values

Splat and For Expressions

## Estimated Time

Average **minimum time** to solve this challenge is 1 hour 30 minutes.

First time when you read and understand scenario carefully , it can take around 2-3 hours.

# **Overview of Challenge 2**

# Setting the Base

The primary aim of this scenario is to test your understanding about Modules and Module Refactoring related use-cases.

Other areas tested: Data Sources



Module Refactoring

Terraform Professional

## Estimated Time

Average **minimum time** to solve this challenge is 1 hour.

First time when you read and understand scenario carefully , it can take around 2 to 2 hour 30 minutes.

# **Overview of Challenge 3**

# Setting the Base

The primary aim of this scenario is to test your understanding about **AWS Provider** and it's integration with Terraform Code.



# **Overview of Challenge 4**

# Setting the Base

The primary aim of this scenario is to test your understanding about **count** and **for expression** along with **output values**



Terraform Professional



Test Areas
count and count.index
Transforming Values
for expressions
Output Values
Fetching Data from External File

# Aim of Such Challenges



# **Overview of Challenge 5**

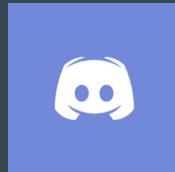
# **Overview of Challenge 6**

# **Overview of Challenge 7**

# **Overview of Challenge 8**

# Join us in our Adventure

Be Awesome



[kplabs.in/chat](https://kplabs.in/chat)



[kplabs.in/linkedin](https://kplabs.in/linkedin)