

Basic Concept : Creating First EC2 Instance Terraform

```
→ iac-in-action git:(main) ✘ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/tls versions matching "~> 2.1"...
- Finding hashicorp/http versions matching "~> 2.0"...
- Finding hashicorp/aws versions matching "~> 2.70"...
- Finding hashicorp/local versions matching "~> 1.4.0"...
- Installing hashicorp/tls v2.2.0...
- Installed hashicorp/tls v2.2.0 (signed by HashiCorp)
- Installing hashicorp/http v1.2.0...
- ...

→ iac-in-action git:(main) ✘ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
not be
persisted to local or remote state storage.

data.http.icanhazip: Refreshing state...
data.aws_vpc.iac_in_action: Refreshing state...

→ iac-in-action git:(main) ✘ terraform apply
data.http.icanhazip: Refreshing state...
data.aws_vpc.iac_in_action: Refreshing state...
```

About Terraform

Write, Plan, Apply

Terraform is an open-source infrastructure as code software tool that provides a consistent CLI workflow to manage hundreds of cloud services. Terraform codifies cloud APIs into declarative configuration files.

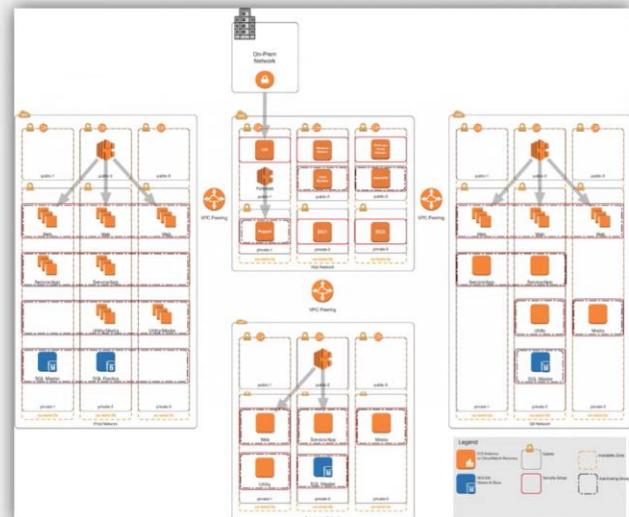
First EC2 Instance using Terraform

AWS Account, Terraform Installation

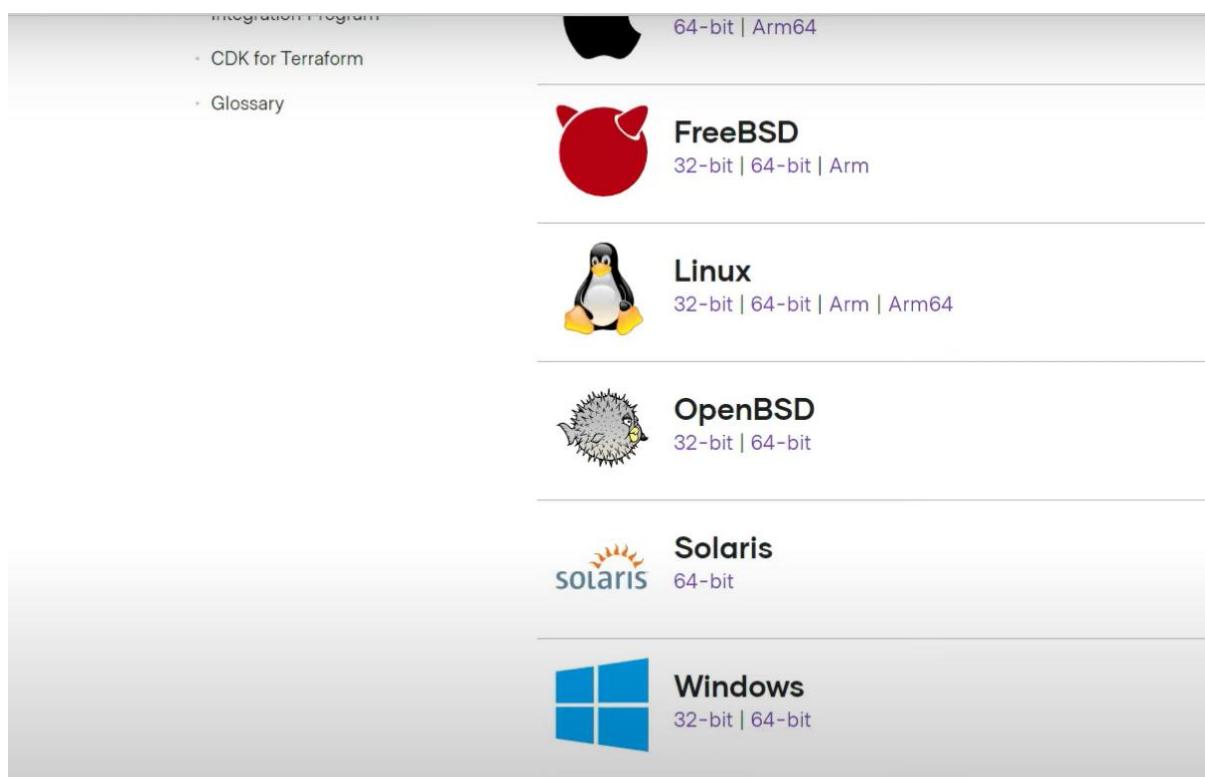
- We required 3 important things
 - How to Authenticate to AWS.
 - Which region?
 - Resource Type to launch?



Comparison

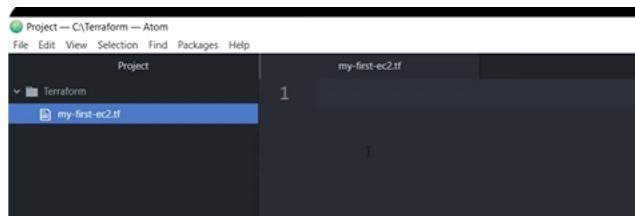


Terraform installation on windows



```
PS C:\Terraform>
```

Using Editor:



Terraform Registry:

The Terraform Registry homepage. At the top, there's a search bar with placeholder text "Search Providers and Modules", and navigation links for "Browse", "Publish", and "Sign-in". Below the header, the title "Terraform Registry" is displayed, followed by a subtitle: "Discover Terraform providers that power all of Terraform's resource types, or find modules for quickly deploying common infrastructure configurations." Two buttons are present: "Browse Providers" and "Browse Modules". A footer note states "1618 providers, 7428 modules & counting".

A detailed view of the Terraform Registry's "Providers" section. On the left, a sidebar contains "FILTERS" and "Clear Filters" buttons, along with filters for "Tier" (Official, Verified, Community) and "Category" (HashiCorp Platform, Public Cloud, Asset Management, Cloud Automation, Communication & Messaging, Container Orchestration, Continuous Integration/Deployment (CI/CD), Data Management, Database, Infrastructure (IaaS)). The main content area is titled "Providers" and describes them as logical abstractions of upstream APIs. It features six cards, each representing a provider: AWS (orange background, white icon), Azure (blue background, white icon), Google Cloud Platform (blue background, white icon), Kubernetes (blue background, white icon), Oracle Cloud Infrastructure (red background, white icon), and Alibaba Cloud (dark grey background, orange icon). Each card also has its name below the icon.

```

my-first-ec2.tf                                         Press [Esc] to exit full screen
1 provider "aws" {
2     region = "ap-south-1"
3     access_key = "AKIAKX6W4IEODQT2PP06N"
4     secret_key = "caxCZwX28X9yQGrarvTLYXt2lstd8uJU7p45Tff"
5 }
6
7 resource "aws_instance" "my-first-ec2" {
8     ami           = "ami-0108d6a82a783b352"
9     instance_type = "t2.micro"
10
11    tags = {
12        Name = "My-First-EC2-Using-TF"
13    }
14 }
15

```

```

PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.66.0...
- Installed hashicorp/aws v3.66.0 (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.

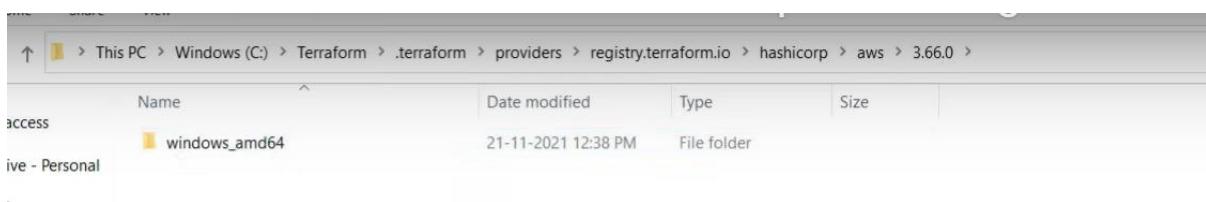
Terraform has been successfully initialized!

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.

```

As soon as terraform init. terraform file will get generated.



Creating one more provider and need to reinitialize

```
my-first-ec2.tf

1 provider "aws" {
2   region = "ap-south-1"
3   access_key = "AKIAJX6W4IEODQT2PP06N"
4   secret_key = "caxCZWwX28X9yQGraryvTLYXt2lsl8uJU7p45Tff"
5 }
6
7 provider "azurerm" {}
8
9 resource "aws_instance" "my-fisrt-ec2" {
10   ami           = "ami-0108d6a82a783b352"
11   instance_type = "t2.micro"
12
13   tags = {
14     Name = "My-First-EC2-Using-TF"
15   }
16 }
17
```

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Finding latest version of hashicorp/azurerm...
- Using previously-installed hashicorp/aws v3.66.0
- Installing hashicorp/azurerm v2.86.0...
- Installed hashicorp/azurerm v2.86.0 (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.

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.
```

```
PS C:\Terraform> terraform 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_instance.my-fisrt-ec2 will be created
+ resource "aws_instance" "my-fisrt-ec2" {
  + ami = "ami-0108d6a82a783b352"
  + arn = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone = (known after apply)
  + cpu_core_count = (known after apply)
  + cpu_threads_per_core = (known after apply)
  + disable_api_termination = (known after apply)
  + ebs_optimized = (known after apply)
  + get_password_data = false
  + host_id = (known after apply)
  + id = (known after apply)
  + instance_initiated_shutdown_behavior = (known after apply)
```

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
PS C:\Terraform> terraform apply
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_instance.my-fisrt-ec2 will be created
+ resource "aws_instance" "my-fisrt-ec2" {
  + ami = "ami-0108d6a82a783b352"
  + arn = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability zone = (known after apply)
```

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

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

```
aws_instance.my-fisrt-ec2: Creating...
aws_instance.my-fisrt-ec2: Still creating... [10s elapsed]
```

Instances (1) <small>Info</small>								
<input type="button" value="Filter instances"/>		<input type="button" value="Launch instances"/>						
Instance state: running <small>X</small>		<input type="button" value="Clear filters"/>						
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public	
My-First-EC2-Using-TF	i-0303130557a1910f7	Running	t2.micro	Initializing	No alarms	ap-south-1a	ec2-3-	

Instance: i-0303130557a1910f7 (My-First-EC2-Using-TF)

Details	Security	Networking	Storage	Status checks	Monitoring	Tags
Tags						
<input type="text"/>						
Key	Value					
Name	My-First-EC2-Using-TF					

1 - Basic Concept - tfstate backup and destroy, target flag.

Tfstate file : Initial state of a file



```
Project my-first-ec2.tf terraform.tfstate .terraform.lock.hcl
43     "get_password_data": false,
44     "hibernation": false,
45     "host_id": null,
46     "iam_instance_profile": "",
47     "id": "i-03d3750506acc80a4",
48     "instance_initiated_shutdown_behavior": "stop",
49     "instance_state": "running",
50     "instance_type": "t2.micro",
51     "ipv6_address_count": 0,
52     "ipv6_addresses": [],
53     "key_name": "",
54     "launch_template": [],
55     "metadata_options": [
56         {
57             "http_endpoint": "enabled",
58             "http_put_response_hop_limit": 1,
59             "http_tokens": "optional"
60         }
61     ]
62 }
```

Tfsate backup: if we make any changes/modifications it will create a backup file. It is a step back configuration

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS C:\terraform> terraform plan
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]
No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so nothing else needs to be done.
```

From AWS console we are making changes not from terraform code. (terraform compare changes Actual state and desired state) Terraform state file /configuration file is consisting of **default** security group.



```
80     "tags": {},
81     "throughput": 0,
82     "volume_id": "vol-0e2a59a8868b50d5a",
83     "volume_size": 8,
84     "volume_type": "gp2"
85   }
86 ],
87   "secondary_private_ips": [],
88   "security_groups": [
89     "default"
90   ],
91   "source_dest_check": true,
92   "subnet_id": "subnet-083ac25158a74398f",
93   "tags": {
94     "Name": "My-First-EC2-Using-TF"
95   },
96 }
```

From console **making change** in security group.

The screenshot shows the 'Change security groups' dialog for an EC2 instance. It displays the instance details (Instance ID: i-03d3750506acc80a4, Network interface ID: eni-02e30430f4c028f7f) and the associated security groups (Security group name: default, Security group ID: sg-05b9d5b829faba080; Security group name: Devops_SG, Security group ID: sg-0328abd51be15469e). A 'Remove' button is visible next to the 'default' group. At the bottom, there are 'Cancel' and 'Save' buttons.

It's showing SG is got changed.

```
PS C:\terraform> terraform plan
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply":

# aws_instance.my-fisrt-ec2 has been changed
~ resource "aws_instance" "my-fisrt-ec2" {
  id                               = "i-03d3750506acc80a4"
  ~ security_groups                 =
    + "Devops_SG",
    + "default",
  ]
  tags                            = {
    "Name" = "My-First-EC2-Using-TF"
  }
  ~ vpc_security_group_ids         =
    + "sg-0328abd51be15469e",
    - "sg-05b9d5b829faba080",
  ]
  # (26 unchanged attributes hidden)
```

```
our configuration already matches the changes detected above. If you'd like
and apply a refresh-only plan:
  terraform apply -refresh-only
$ C:\terraform> terraform apply -refresh-only
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]
```

In state file we are having sg group: devOps_sg

```

  "throughput": 0,
  "volume_id": "vol-0e2a59a8868b50d5a",
  "volume_size": 8,
  "volume_type": "gp2"
}
],
"secondary_private_ips": [],
"security_groups": [
  "Devops_SG"
]

```

In backup file we are having sg group : default

```

  "throughput": 0,
  "volume_id": "vol-0e2a59a8868b50d5a",
  "volume_size": 8,
  "volume_type": "gp2"
}
],
"secondary_private_ips": [],
"security_groups": [
  "default"
]

```

Create the GIT repo using terraform:

Settings / Developer settings

New personal access token

GitHub Apps OAuth Apps Personal access tokens

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note:

Expiration *: 30 days (The token will expire on Mon, Jan 3 2022)

Select scopes: Scopes define the access for personal tokens. Read more about OAuth scopes.

<input type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status

Settings / Developer settings

GitHub Apps OAuth Apps Personal access tokens

Personal access tokens

Generate new token Revoke all

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

Copied! ✓ ghp_A7i6E2bLHMkQ7AB3FogypN1aEl8gs2t07Si ✓

Delete

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

```
terraform
└── terraform
    ├── terraform.lock.hcl
    └── github.tf
        ├── my-first-ec2.tf
        ├── terraform.state
        └── terraform.state.backup
1   terraform {
2     required_providers {
3       github = {
4         source  = "integrations/github"
5         version = "~> 4.0"
6       }
7     }
8   }
9
10 # Configure the GitHub Provider
11 provider "github" {
12   token = "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"
13 }
```

github_repository

This resource allows you to create and manage repositories within your GitHub organization or personal account.

Example Usage

```
resource "github_repository" "example" {
  name      = "example"
  description = "My awesome codebase"

  visibility = "public"

  template {
    owner      = "github"
    repository = "terraform-module-template"
  }
}
```

```
my-first-ec2.tf          github.tf          terraform.lock.hcl
1  terraform {
2      required_providers {
3          github = {
4              source  = "integrations/github"
5              version = "~> 4.0"
6          }
7      }
8  }
9
10 # Configure the GitHub Provider
11 provider "github" {
12     token = "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"
13 }
14
15 resource "github_repository" "example" {
16     name      = "My_first_repo_using_TF"
17     visibility = "private"
18 }
```

```
PS C:\terraform>
PS C:\terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Finding integrations/github versions matching "~> 4.0"...
- Using previously-installed hashicorp/aws v3.68.0
```

```
PS C:\terraform> terraform plan
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]

Terraform used the selected providers to generate the following execution plan.
following symbols:
+ create
Terraform will perform the following actions:
```

```
PS C:\terraform> terraform apply
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]
```

The screenshot shows a GitHub interface with a search bar and a list of repositories. The repository 'jhawithu/My_first_repo_using_TF' is highlighted with a yellow box.

Repository	Description
jhawithu/hello-world	
jhawithu/My_first_repo_using_TF	
jhawithu/vprofile-project	

Now tf state file consist of info of both ec2+GIT repo

Terraform target:

If I want to destroy on GITHUB repo provider not ec2 provider in that case we will use terraform target:

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.  
PS C:\terraform> terraform destroy -target github_repository.example
```

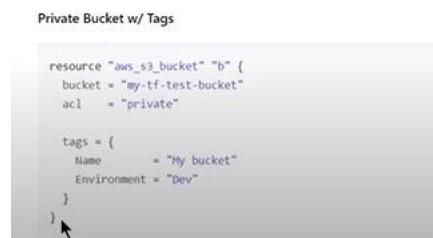
Terraform destroy -target resourcename. Localname

```
github_repository.example: Refreshing state... [id=My_first_repo_using_TF]  
  
Terraform used the selected providers to generate the following execution plan.  
following symbols:  
- destroy  
  
Terraform will perform the following actions:  
  
# github_repository.example will be destroyed  
- resource "github_repository" "example" {
```

After removing git hub provider state file consist of information about only for ec2 provider and backup file consist of both ec2 provider + git hub provider info

3 - Terraform Resource Attribute and Output Values

Create s3 and Elastic IP and we will fetch it and display in the output.



Declaring an Output Value

Each output value exported by a module must be declared using an `output` block:

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

Attributes Reference

In addition to all arguments above, the following attributes are exported:

- `allocation_id` - ID that AWS assigns to represent the allocation of the Elastic IP address for use with instances in a VPC.
- `association_id` - ID representing the association of the address with an instance in a VPC.
- `carrier_ip` - Carrier IP address.
- `customer_owned_ip` - Customer owned IP.
- `domain` - Indicates if this EIP is for use in VPC (`vpc`) or EC2 Classic (`standard`).
- `id` - Contains the EIP allocation ID.
- `private_dns` - The Private DNS associated with the Elastic IP address (if in VPC).
- `private_ip` - Contains the private IP address (if in VPC).
- `public_dns` - Public DNS associated with the Elastic IP address.
- `public_ip` - Contains the public IP address.
- `tags_all` - A map of tags assigned to the resource, including those inherited from the provider `default_tags` configuration block.

```
1 provider "aws" {
2   region = "ap-south-1"
3   access_key = "AKIAK6W4IEODYTWVDQ55"
4   secret_key = "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"
5 }
6
7 resource "aws_eip" "lb" {
8   vpc      = true
9 }
10
11 output "eip_output" {
12   value = aws_eip.lb.public_ip
13 }
14
15 resource "aws_s3_bucket" "b" {
16   bucket = "my-tf-test-bucket-4567"
17 }
18
19 output "s3_arn" {
20   value = aws_s3_bucket.b.arn
21 }
```

Value = resource name. local name. attribute reference name

```
Plan: 2 to add, 0 to change, 0 to destroy.
```

```
Changes to Outputs:
```

```
+ eip_output = (known after apply)
+ s3_arn    = (known after apply)
```

```
PS C:\Terraform> terraform apply
```

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

```
Outputs:
```

```
eip_output = "3.111.106.127"
s3_arn = "arn:aws:s3:::my-tf-test-bucket-4567"
```

Amazon S3 > my-tf-test-bucket-4567

my-tf-test-bucket-4567 [Info](#)

Objects Properties Permissions Metrics Management Access Points

Bucket overview

AWS Region	Amazon Resource Name (ARN)
Asia Pacific (Mumbai) ap-south-1	arn:aws:s3:::my-tf-test-bucket-4567

Name	Type	Allocation ID
3.111.106.127	Public IP	eipalloc-05cb58c403811d74c

If we have not given attribute reference name ,it will fetch all the values

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

eip_output = {
  "address" = tostring(null)
  "allocation_id" = "eipalloc-0e3965ce6413b8979"
  "associate_with_private_ip" = tostring(null)
  "association_id" = ""
  "carrier_ip" = ""
  "customer_owned_ip" = ""
  "customer_owned_ipv4_pool" = ""
  "domain" = "vpc"
  "id" = "eipalloc-0e3965ce6413b8979"
  "instance" = ""
  "network_border_group" = "ap-south-1"
  "network_interface" = ""
  "private_dns" = tostring(null)
  "private_ip" = ""
  "public_dns" = "ec2-3-111-104-11.ap-south-1.compute.amazonaws.com"
  "public_ip" = "3.111.104.11"
  "public_ipv4_pool" = "amazon"
  "tags" = tomap(null)/* of string */
  "tags_all" = tomap({})
  "timeouts" = null /* object */
  "vpc" = true
}
```

```

53. def = `

  "acceleration_status" = ""
  "acl" = "private"
  "arn" = "arn:aws:s3:::my-tf-test-bucket-4567"
  "bucket" = "my-tf-test-bucket-4567"
  "bucket_domain_name" = "my-tf-test-bucket-4567.s3.amazonaws.com"
  "bucket_prefix" = tostring(null)
  "bucketRegionalDomainName" = "my-tf-test-bucket-4567.s3.ap-south-1.amazonaws.com"
  "cors_rule" = tolist([])
  "force_destroy" = false
  "grant" = toset([])
  "hostedZoneId" = "Z11RGJOFQNVJUP"
  "id" = "my-tf-test-bucket-4567"
  "lifecycle_rule" = tolist([])
  "logging" = toset([])
  "objectLockConfiguration" = tolist([])
  "policy" = tostring(null)
  "region" = "ap-south-1"
  "replicationConfiguration" = tolist([])
  "requestPayer" = "BucketOwner"
  "serverSideEncryptionConfiguration" = tolist([])
  "tags" = tomap(null) /* of string */
  "tagsAll" = tomap({})
  "versioning" = tolist([
    {
      "enabled" = false
      "mfaDelete" = false
    },
  ])
  "website" = tolist([])
  "websiteDomain" = tostring(null)
  "websiteEndpoint" = tostring(null)
}

```

4- Terraform Provider Version handling.

```

PS C:\Terraform> terraform version
Terraform v1.1.4
on windows_386

```

```

Provider.tf          .terraform.lock.hcl

1  terraform {
2    required_providers {
3      aws = {
4        source = "hashicorp/aws"
5        version = "3.74.0"
6      }
7    }
8  }
9
10 provider "aws" {
11   # Configuration options
12 }

```

- If we will have not mentioned the version it will fetch the latest version.
- If we give different version in provider.tf file (configuration file) it will throw an error because it will check for lock file as well for the -upgrade without removing the lock file.
- If any version is mentioned, First will check the tf file if not will check the lock file

```

PS C:\Terraform> terraform init

Initializing the backend...

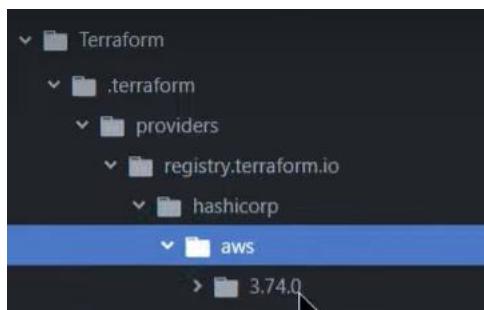
Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
- Installed hashicorp/aws v3.74.0 (signed by HashiCorp)

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.

```



```

# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version      = "3.74.0"
  constraints = "3.74.0"
  hashes = [
    "h1:Nl0E3bUh3oLF/Cn7JGGUumwg297uUCj3x+vswoEyKk=",
    "zh:00767509c13c0d1c7ad6af702c6942e6572aa6d529b40a00baacc0e73faafea2",
    "zh:03aafdc903ad49c2eda03889f927f44212674c50e475a9c6298850381319eec2",
    "zh:2de8a6a97b180f909d652f215125aa4683e99db15fcf3b28d62e3d542f875ed6",
    "zh:3ac29ebc3af99028f4230a79f56606a0c2954b68767bd749b921a76eb4f3bd30",
    "zh:50add2e2d118a15a644360eabc5a34cec59f2560b491f8fabf9c52ab83ca7b09",
    "zh:85dd8e81910ab79f841a4a595fd8ac358fbe460956144afb0be3d81f91fe10",
    "zh:895de83d0f0941fde31bfc53fa6b1ea276901f006bec221bbdee4771a04f3693",
    "zh:a15c9724aac52d1ba5001d2d83e42843099b52b1638ea29d84e20be0f45fa4f1",
    "zh:c982a64463bd73e9bff2589de214b1de0a571438d9015001f9eae45cf3a2559",
    "zh:e9ef973c18078324e43213ea1252c12b9441e566bf054ddfdbff5dd62f3035d9",
    "zh:f297e705b0f339c8baa27ae70db5df9aa6578adfe1ea3d2ba8edc186512464eb",
  ]
}

```

If want to downgrade the version we can do it but we will get an error in lock file

```

PS C:\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: locked provider registry.terraform.io/
not match configured version constraint 3.73.0; must use terraform init -upgrade to allow selection of new versions

```

Solution1→You can delete the lock file

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.73.0"...
- Installing hashicorp/aws v3.73.0...
- Installed hashicorp/aws v3.73.0 (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!
```

Solution2→Instead deleting lock file mention below mention command

>terraform init upgrade→It will unlock the lock file and mark the required information

```
| Provider.tf          .terraform.lock.hcl
1  terraform {
2    required_providers {
3      aws = {
4        source = "hashicorp/aws"
5        version = "3.68.0"
6      }
7    }
8  }
9
10 provider "aws" {
11   # Configuration options
12 }
```

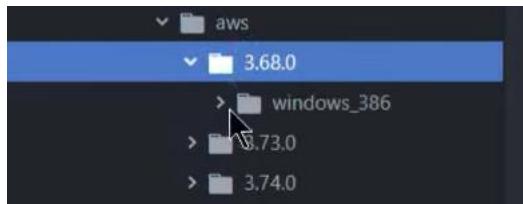
```
PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.68.0"...
- Installing hashicorp/aws v3.68.0...
- Installed hashicorp/aws v3.68.0 (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.

Terraform has been successfully initialized!
```



Solution 3: If we remove particular version line from provider.tf

Terraform will check from provider.tf and then check it in .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      = "3.68.0"
    constraints = "3.68.0"
    hashes = [
        "h1:TiCcr...286vmz5wB9zPx/B4GGD9zhhrfUZ2pmLJKqIig="
        "zh:05a43a7dbd409451c08a958610234619d7e0d102e601"
        "zh:0d195fa738a348e511550de39caec3f10cfb9afe8d69"
        "zh:3d88a19b2a810559bc6953fe92b7a7c6e3251c550186"
    ]
}
```

It will update the version mentioned in constraints.

There are multiple ways for specifying the version of a provider

Version Argument	Description
<code>>=3.0</code>	Greater than equal to the version
<code><=3.0</code>	Less than equal to the version
<code>~>2.0</code>	Any version in 2.X range
<code>>=3.10,<=3.50</code>	Any version between 3.10 and 3.50

```

Provider.tf          .terraform.lock.hcl

1  terraform {
2      required_providers {
3          aws = {
4              source = "hashicorp/aws"
5              version = "~>2.0"
6          }
7      }
8  }
9
10 provider "aws" {
11     # Configuration options
12 }

```

It will download maximum into 2 series i.e one is 2 version and another one is 3version

```

PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 2.0"...
- Installing hashicorp/aws v2.70.1...
- Installed hashicorp/aws v2.70.1 (signed by HashiCorp)

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

Terraform has been successfully initialized!

```

```

        hashicorp
          aws
            > 2.70.1
            > 3.68.0
.terraform.lock.hcl
Provider.tf
> Kubernetes-netpol

4  provider "registry.terraform.io/hashicorp/aws" {
5      version      = "2.70.1"
6      constraints = "~> 2.0"
7      hashes = [
8          "h1:Kyqfh56bEYm0YbCfWwVF6PLEvNpcFJxPVC5LevgBm8A=",
9          "zh:04137cdf128cf21dc190bbba4d4bba43c7868c52ad646",
10         "zh:30c9f956133a102b4a426d76dd3ef1a42332d9875261a0",
11         "zh:3107a43647454a3d6d847fba6aa593650af0f6a353272c",
12         "zh:3f17285478313af822447b453fa4e37f30ef221f0b0e8f",
13         "zh:5a626f7a3c4a9fe23bdfde63aedbf6ea73760f3b228f7"

```

```

PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching ">= 3.10.0, <= 3.50.0"...
- Installing hashicorp/aws v3.50.0...
- Installed hashicorp/aws v3.50.0 (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.

Terraform has been successfully initialized!

```

If we give not mentioned version it will throw an error.

```

PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "2.70.2"...

Error: Failed to query available provider packages

Could not retrieve the list of available versions for provider hashicorp/aws: no available releases match the given constraints 2.70.2

```

5 - Terraform Format, Validate, EIP

Terraform Format:



```

  ec2-eip-sg.tf          Terraform state

  terraform {
    required_providers {
      aws = {
        source  = "hashicorp/aws"
        version = "3.74.0"
      }
    }
  }

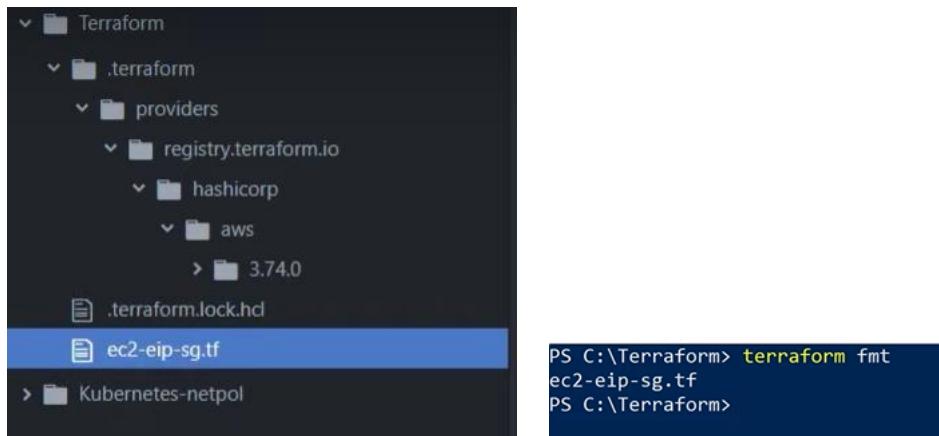
  provider "aws" {
    region     = "ap-south-1"
    access_key = "AKIAJ6W4IEODXH3GQ04I"
    secret_key = "9SzSNdbrJ4VJfpcILKvcBPcRrPk9W3fcTh4972h/"
  }

  resource "aws_instance" "myec2" {
    ami           = "ami-0f1fb91a596abf28d"
    instance_type = "t2.micro"
  }

  resource "aws_eip" "lb" {

```

```
PS C:\Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
```



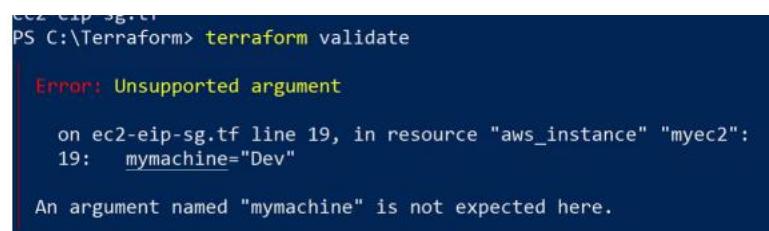
All the spaces, syntax get aligned in terraform file

Terraform Validate:

If we add any attributes which are not exit. Terraform validate and throw an error like “**Unsupported arguments**”

```
provider "aws" {
  region      = "ap-south-1"
  access_key  = "AKIAK6W4IEODXH3GQ04I"
  secret_key  = "9SzSNdbrJ4VJfpcILKVcBPcRrPk9W3fcTh4972h/"
}

resource "aws_instance" "myec2" {
  ami           = "ami-0f1fb91a596abf28d"
  instance_type = "t2.micro"
  mymachine="Dev"
}
```



```
PS C:\Terraform> terraform plan

Error: Unsupported argument      ↵

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
19:   mymachine"=Dev"

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

```
PS C:\Terraform> terraform validate

Error: Unsupported argument

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
19:   mymachine"=Dev"

  An argument named "mymachine" is not expected here.

PS C:\Terraform> terraform plan

Error: Unsupported argument      ↵

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
19:   mymachine"=Dev"

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

TERRAFORM APPLY will always ask for the confirmation instead we can use auto approve

```
an: 2 to add, 0 to change, 0 to destroy.      ↵

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: no
```

```
PS C:\Terraform> terraform apply --auto-approve
```

```
Plan: 2 to add, 0 to change, 0 to destroy.
aws_eip.lb: Creating...
aws_instance.myec2: Creating...
aws_eip.lb: Creation complete after 0s [id=eipalloc-0844c78c64e6afbdd]
```

Terraform EIP: Association (Linking) EIP with EC2 instance.

```
resource "aws_eip_association" "eip_assoc" {
  instance_id    = aws_instance.web.id
  allocation_id = aws_eip.example.id
}

resource "aws_instance" "web" {
  ami           = "ami-21f78e11"
  availability_zone = "us-west-2a"
  instance_type   = "t2.micro"

  tags = {
    Name = "HelloWorld"
  }
}

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

Instance_id =resource name. local name. attribute name

Terraform will perform the following actions:

```
# aws_eip_association.eip_assoc will be created
+ resource "aws_eip_association" "eip_assoc" {
  + allocation_id      = "eipalloc-0844c78c64e6afbdd"
  + id                 = (known after apply)
  + instance_id        = "i-07253684726792b3f"
  + network_interface_id = (known after apply)
  + private_ip_address = (known after apply)
  + public_ip          = (known after apply)
}
```

PS C:\Terraform> **terraform apply --auto-approve**

The screenshot shows a CloudWatch Metrics dashboard with a single metric named 'EIPAllocated'. The metric has a value of 1 and is associated with the instance ID 'i-07253684726792b3f'. Below the dashboard, the AWS Lambda function 'EIPAllocated' is shown with its code and configuration.

EIPAllocated

```
function handler(event, context) {
  const { instanceId } = event;
  const { associationId } = event;

  const eipAlloc = require('aws-sdk').Service('ElasticIp');
  const ec2 = require('aws-sdk').Service('EC2');

  const params = {
    instanceId,
    allocationId: associationId
  };

  eipAlloc.associate(params, (err, data) => {
    if (err) {
      console.error(`Error associating EIP: ${err}`);
      return;
    }

    console.log(`EIP successfully associated with instance ${instanceId}`);
  });
}
```

Metrics

Metric	Value
EIPAllocated	1

Logs

```
2023-09-12T10:30:00Z -> {"instanceId": "i-07253684726792b3f", "associationId": "eipassoc-08bf47e0ccf7644f6"} 2023-09-12T10:30:00Z <- {"instanceId": "i-07253684726792b3f", "allocationId": "eipalloc-0844c78c64e6afbdd"} EIP successfully associated with instance i-07253684726792b3f
```

CloudWatch Metrics

Series	Value
EIPAllocated	1

EC2 Instances

Name	Allocated IPv4 add...	Type	Allocation ID	Reverse DNS record
-	3.110.2.78	Public IP	eipalloc-0844c78c64e6afbdd	-

Associated Instances

Public IP	Association ID	Scope	Associated instance ID	Private IP address
3.110.2.78	eipassoc-08bf47e0ccf7644f6	VPC	i-07253684726792b3f	172.31.7.41

Instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
-	i-07253684726792b3f	Running	t2.micro	2/2 checks passed	No alarms	ap-south-1b

6 - Terraform Shared Credential, AWS CLI, Comments



The screenshot shows a code editor with a file named `static_credential-best-practise.tf`:

```
EXPLORER ... static_credential-best-practise.tf
TERRAFORM static_credential-best-practise.tf > terraform > required_providers
1 terraform {
2   required_version = "~> 1.1.4"
3   required_providers {
4     aws = {
5       source  = "hashicorp/aws"
6       version = "3.74.0"
7     }
8   }
9 }
10
11 provider "aws" {
12   region      = "ap-south-1"
13   access_key  = "AKIAK6W4IEODXSVKO4FQ"
14   secret_key  = "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"
15 }
16
17 resource "aws_instance" "myec2" {
18   ami          = "ami-0f1fb91a596abf28d"
19   instance_type = "t2.micro"
20 }
21
22
```

Comments :

```
# this is single line comment
// this is also single line comment
/*
This is multi line comment
2
3 */
```

```

Terraform output - Part 3 - Basic Concepts - Terraform 0.12.20

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
- Installed hashicorp/aws v3.74.0 (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.

PS C:\Terraform> terraform fmt
PS C:\Terraform> terraform validate
Success! The configuration is valid.

```

```

EXPLORER
...
TERRAFORM
static_credential-best-practise.tf
  static_credential-best-practise.tf > provider "aws"
    terraform {
      required_version = "~> 1.1.4"
      required_providers {
        aws = {
          source  = "hashicorp/aws"
          version = "3.74.0"
        }
      }
    }
    provider "aws" {
      region     = "ap-south-1"
      access_key = "AKIAKGWMIIEODXSVK04FQ"
      secret_key = "A9PGt1svf+Ja+RQOjf+1Y8ThzktYWlhXaZirEuV+"
    }
    resource "aws_instance" "myec2" {
      ami           = "ami-0f1fb91a596abf28d"
      instance_type = "t2.micro"
    }
  }

```

Static Credentials

⚠ Warning:

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

Static credentials can be provided by adding an `access_key` and `secret_key` in-line in the AWS provider block:

Usage:

```

provider "aws" {
  region     = "us-west-2"
  access_key = "my-access-key"
  secret_key = "my-secret-key"
}

```

Instead of using Static Credentials we need to use Environment Variable, Shared credentials files.

Environment Variables

You can provide your credentials via the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, environment variables, representing your AWS Access Key and AWS Secret Key, respectively. Note that setting your AWS credentials using either these (or legacy) environment variables will override the use of `AWS_SHARED_CREDENTIALS_FILE` and `AWS_PROFILE`. The `AWS_REGION` or `AWS_DEFAULT_REGION` and `AWS_SESSION_TOKEN` environment variables are also used, if applicable:

```
provider "aws" {}
```

Usage:

Shared Credentials File

You can use AWS credentials or configuration files to specify your credentials and configuration. The default locations are `$HOME/.aws/credentials` and `$HOME/.aws/config` on Linux and macOS, or `"%USERPROFILE%\aws\credentials"` and `"%USERPROFILE%\aws\config"` on Windows. You can optionally specify a different location in the Terraform configuration by providing the `shared_credentials_files` and `shared_config_files` arguments or using the `AWS_SHARED_CREDENTIALS_FILE` and `AWS_CONFIG_FILE` environment variables. This method also supports the `profile` configuration or corresponding `AWS_PROFILE` environment variable:

Usage:

```
provider "aws" {
  region      = "us-west-2"
  shared_config_files = ["~/users/tf_user/.aws/conf"]
  shared_credentials_files = ["~/users/tf_user/.aws/creds"]
  profile     = "customprofile"
```

AWS CLI (Command Line Interface):

Here we are store our credentials we are not using in terraform file.

Installation requirements

- We support the AWS CLI on Microsoft-supported versions of 64-bit Windows.
- Admin rights to install software

Installation instructions

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the [AWS CLI changelog](#) on GitHub.

1. Download and run the AWS CLI MSI installer for Windows (64-bit):

<https://awscli.amazonaws.com/AWSCLIV2.msi>

Alternatively, you can run the `msiexec` command to run the MSI installer.

```
C:\> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```

For various parameters that can be used with `msiexec`, see [msiexec](#) on the Microsoft Docs

To install AWS CLI

```
PS C:\Terraform> aws configure
AWS Access Key ID [*****04FQ]:
AWS Secret Access Key [*****EuV+]:
Default region name [ap-south-1]:
Default output format [None]:
PS C:\Terraform>
```

Location where the credentials and config files got installed in windows.

C:/Users/Username/.aws

```

PS C:\Users\Alok> cd .\.aws\
PS C:\Users\Alok\.aws> ls

    Directory: C:\Users\Alok\.aws

Mode                LastWriteTime         Length Name
----                -----          ---- - 
-a----       12-02-2022 10:51 AM            32 config
-a----       12-02-2022 10:51 AM        119 credentials

PS C:\Users\Alok\.aws> cat config
[default]
region = ap-south-1
PS C:\Users\Alok\.aws> cat .\credentials
[default]
aws_access_key_id = AKIAJX6W4IEODXSVK04FQ
aws_secret_access_key = A9PGtLsvf+Ja+RQOjf+1Y0ThzktYWlhXaZirEuV+
PS C:\Users\Alok\.aws>

```

```

* static_credential-best-practise.tf
  static_credential-best-practise.tf > ...
  1  terraform {
  2      required_version = "~> 1.1.4"
  3      required_providers {
  4          aws = {
  5              source  = "hashicorp/aws"
  6              version = "3.74.0"
  7          }
  8      }
  9  }
 10
 11 provider "aws" {
 12     region      = "ap-south-1"
 13 }
 14
 15 resource "aws_instance" "myec2" {
 16     ami           = "ami-0f1fb91a596abf28d"
 17     instance_type = "t2.micro"
 18 }
```

7 Running Nginx from Docker Container using Terraform:

Create EC2 instance, create docker env on ec2 and pull docker image nginx.



```
#!/bin/sh
yum update -y
yum install docker -y
service docker start
docker pull nginx:alpine
docker run -it -d --name my-first-container -p 80:80 nginx:alpine
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#). Commercial support is available at [nginx.com](#).

Thank you for using nginx.

Project

- Terraform
 - docker-container-user-data.tf
- Kubernetes-netpol

```
3   required_providers {
4     aws = {
5       source  = "hashicorp/aws"
6       version = "3.74.0"
7     }
8   }
9 }
10
11 provider "aws" {
12   region = "ap-south-1"
13 }
14
15 resource "aws_instance" "myec2" {
16   ami           = "ami-0f1fb91a596abf28d"
17   instance_type = "t2.micro"
18   vpc_security_group_ids = [
19     "sg-0328abd51be15469e"
20   ]
21
22 }
```

Add user data

```

user_data = <<EOF
#!/bin/sh
yum update -y
yum install docker -y
service docker start
docker pull nginx:alpine
docker run -it -d --name my-first-container -p 80:80 nginx:alpine
EOF
}

PS C:\Terraform> aws configure
AWS Access Key ID [*****NOAI]:
AWS Secret Access Key [*****GI/3]:
Default region name [ap-south-1]:
Default output format [None]:
PS C:\Terraform> ls

Directory: C:\Terraform

Mode                LastWriteTime         Length Name
----                -----          318 docker-container-user-data.tf

PS C:\Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...

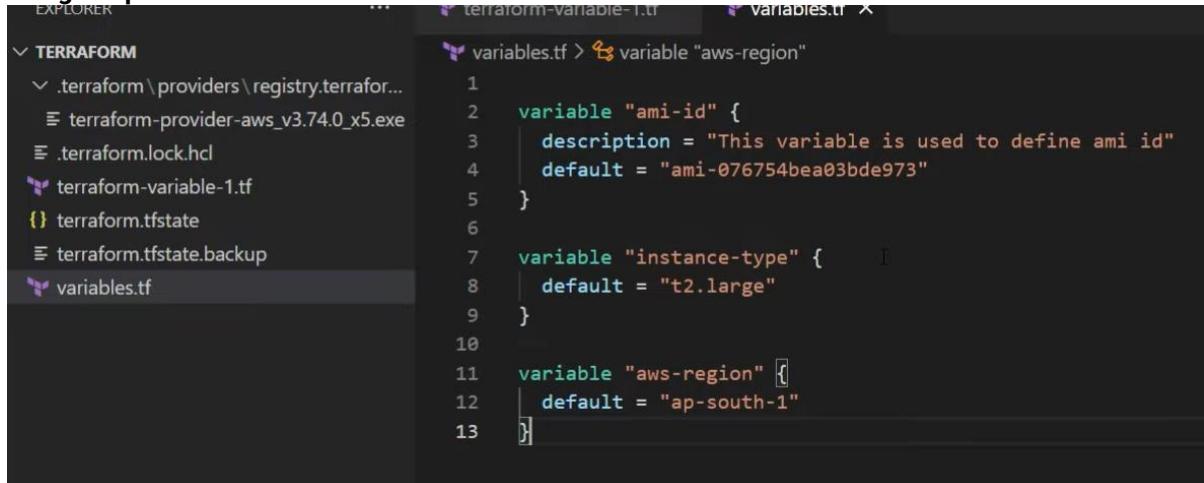
PS C:\Terraform> terraform validate
Success! The configuration is valid.

PS C:\Terraform> terraform fmt
docker-container-user-data.tf
PS C:\Terraform> terraform plan

```

8 - Terraform Input Variable Part – 1

It is good practice to have different files.



The screenshot shows a code editor with a dark theme. On the left, the Explorer sidebar shows a folder structure under TERRAFORM:

- .terraform\providers\registry.terraform
- terraform-provider-aws_v3.74.0_x5.exe
- .terraform.lock.hcl
- terraform-variable-1.tf
- terraform.tfstate
- terraform.tfstate.backup
- variables.tf

The main pane displays the content of the variables.tf file:

```

variable "aws-region"
1
2   variable "ami-id" {
3     description = "This variable is used to define ami id"
4     default = "ami-076754bea03bde973"
5   }
6
7   variable "instance-type" {
8     default = "t2.large"
9   }
10
11  variable "aws-region" {
12    default = "ap-south-1"
13  }

```

```

terraform-variable-1.tf > variable "aws-region" > default
  terraform {
    required_version = "~> 1.1.4"
    required_providers {
      aws = {
        source  = "hashicorp/aws"
        version = "3.74.0"
      }
    }
  }

  variable "ami-id" {
    description = "This variable is used to define ami id"
    default = "ami-076754bea03bde973"
  }

  variable "instance-type" {
    default = "t2.micro"
  }

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

  resource "aws_instance" "myec2-1" {
    ami           = "ami-076754bea03bde973"
    instance_type = "t2.micro"
  }

  provider "aws" {
    region = "ap-south-1"
  }

  resource "aws_instance" "myec2-1" {
    ami           = var.ami-id
    instance_type = var.instance-type
  }

  resource "aws_instance" "myec2-2" {
    ami           = var.ami-id
    instance_type = var.instance-type
  }

```

Scenario1 → If we define the variable but we have not given any default value then terraform plan will not give any error but it will ask for the entry

```

TERRAFORM
└── .terraform\providers\registry.terraform...
    └── terraform-provider-aws_v3.74.0_x5.exe
    └── .terraform.lock.hcl
    └── terraform-variable-1.tf
    └── terraform.tfstate
    └── terraform.tfstate.backup
    └── variables.tf

```

```

variable "ami-id" {
  description = "This variable is used to define ami id"
  default     = "ami-076754bea03bde973"
}

variable "instance-type" {}

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

```

```

PS C:\Terraform> terraform plan
var.instance-type
Enter a value: 

```

Scenario2 → You do not have access to the file to edit but all of sudden you have got requirement to make change default to t2.large than time you can override it.

Even if default value is given already if you want to edit /modify any of the files you can use CLI and you can do **override**

```

variables.tf > variable "instance-type"
variable "ami-id" {
  description = "This variable is used to define ami id"
  default     = "ami-076754bea03bde973"
}

variable "instance-type" {
  default = "t2.micro"
}

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

```

```

PS C:\Terraform> terraform plan -var instance-type="t2.large"

```

9 -Variable, Count and Generating and Applying TF Plan Part-2

» The `count` Meta-Argument

JUMP TO SECTION ▾

Version note: Module support for `count` was added in Terraform 0.13, and previous versions can only use it with resources.

Note: A given resource or module block cannot use both `count` and `for_each`.

`count` is a meta-argument defined by the Terraform language. It can be used with modules and with every resource type.

The `count` meta-argument accepts a whole number, and creates that many instances of the resource or module. Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"

  tags = {
    Name = "Server ${count.index}"
  }
}

variable "ami-id" {
  description = "This variable is used to define ami id"
  default = "ami-076754bea03bde973"
}

variable "instance-type" {
  default = "t2.large"
}

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

variable "aws-instance-count" {
  default = 3
}
```

Override the values.

```
PS C:\Terraform> terraform plan -var instance-type="t2.micro" -var aws-instance-count=1
```

Create customised file:

Scenario → To override multiple variables will use. **Generate a plan files** and put it in the myplan.plan file.

-out myplan.plan

```
Apply cancelled.  
PS C:\Terraform> terraform plan -var instance-type="t2.micro" -var aws-instance-count=1 -out myplan.plan
```

Saved the plan to: myplan.plan

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



```
PS C:\Terraform> terraform apply myplan.plan  
aws_instance.server[0]: Creating...
```

10 -. Input Var from *.tfvars, *.auto.tfvars & env var Part-3

Variable Definition Precedence

The above mechanisms for setting variables can be used together in any combination. If the same variable is assigned multiple values, Terraform uses the *last* value it finds, overriding any previous values. Note that the same variable cannot be assigned multiple values within a single source.

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

- Environment variables
- The `terraform.tfvars` file, if present.
- The `terraform.tfvars.json` file, if present.
- Any `*.auto.tfvars` or `*.auto.tfvars.json` files, processed in lexical order of their filenames.
- Any `-var` and `-var-file` options on the command line, in the order they are provided. (This includes variables set by a Terraform Cloud workspace.)

The variable precedence starts from the below

1)-var and -var-file

2) *. auto. tfvars/*.auto.tfvars.json

3) terraform. tfvars. json

4)terraform.tfvars

5) Environment variable

```
*Untitled - Notepad
File Edit Format View Help
Set environment variable
=====
export TF_VAR_instance-type="t2.medium"
export TF_VAR_aws-instance-count="10"

echo $TF_VAR_instance-type, $TF_VAR_aws-instance-count

unset TF_VAR_instance-type
unset TF_VAR_aws-instance-count
```

```
variables.tf
variable "instance-type"
variable "ami-id" {
  description = "This variable is used to define ami id"
  default = "ami-076754bea03bde973"
}
variable "instance-type" {
  default = "t2.micro"
}
variable "aws-region" {
  default = "ap-south-1"
}
```

```
PS C:\Terraform> terraform init
Initializing the backend... ↵

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v3.74.0

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.
PS C:\Terraform> terraform validate
Success! The configuration is valid.

PS C:\Terraform> terraform fmt
terraform-variable-2.tf
variables.tf
PS C:\Terraform>
```

variable.tf file will be override by **terraform. tfvars** since its higher precision

The screenshot shows a terminal window with a dark theme. On the left, there's a file tree for a 'TERRAFORM' workspace. It includes files like '.terraform\providers\registry.terraform', 'terraform-provider-aws_v3.74.0_x5.exe', '.terraform.lock.hcl', 'terraform-variable-2.tf', 'terraform.tfstate', 'terraform.tfstate.backup', 'terraform.tfvars' (which is highlighted with a mouse cursor), and 'variables.tf'. On the right, the content of 'terraform.tfvars' is displayed, showing two lines of code: 'instance-type = "t2.micro"' and 'aws-instance-count = "1"'. The code editor interface has syntax highlighting for Terraform variables.

Scenario1 → We have to explicitly mention custom files while terraform plan **terraform plan -var-file= “qa.tfvars” – higher precedence**

```
PS C:\Terraform> terraform plan -var-file="qa.tfvars"
```

Scenario2 → You don't want to give every time i.e var-file name during plan for that you want to make it **default file** you need to rename it with the name: **qa.auto.tfvars [filename.auto.tfvars]**

The screenshot shows a terminal window with a dark theme. On the left, there's a file tree for a 'TERRAFORM' workspace. It includes files like '.terraform\providers\registry.terraform', 'terraform-provider-aws_v3.74.0_x5.exe', '.terraform.lock.hcl', and 'qa.auto.tfvars' (which is highlighted with a blue selection bar). On the right, the content of 'qa.auto.tfvars' is displayed, showing two lines of code: 'instance-type = "t2.xlarge"' and 'aws-instance-count = "5"'. The code editor interface has syntax highlighting for Terraform variables.

11 - Implement Variable Type as Str, Num, List & Map. Part-4

Input Variables

JUMP TO SECTION ▾

Note: This page is about Terraform 0.11 and earlier. For Terraform 0.12 and later, see Configuration Language: [Input Variables](#).

Input variables serve as parameters for a Terraform module.

When used in the root module of a configuration, variables can be set from CLI arguments and environment variables. For *child* modules, they allow values to pass from parent to child.

This page assumes you're familiar with the configuration syntax already.

Input variables can be defined as follows:

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

variable "images" {
  type = "map"

  default = {
    us-east-1 = "image-1234"
    us-west-2 = "image-4567"
  }
}

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

Strings

String values are simple and represent a basic key to value mapping where the key is the variable name. An example is:

```
variable "key" {
  type = "string"
  default = "value"
}
```

[Copy](#)

A multi-line string value can be provided using heredoc syntax.

```
variable "long_key" {
  type = "string"
  default = <<EOF
This is a long key.
Running over several lines.
EOF
}
```

[Copy](#)

Maps

A map value is a lookup table from string keys to string values. This is useful for selecting a value based on some other provided value.

A common use of maps is to create a table of machine images per region, as follows:

```
variable "images" {
  type    = "map"
  default = {
    "us-east-1" = "image-1234"
    "us-west-2" = "image-4567"
  }
}
```

Copy ↗

LIST:

```
variable "ami-id" {
  description = "This variable is used to define ami id"
  default     = "ami-076754bea03bde973"
}

variable "instance-type" {
  type = list(string)
  default = ["t2.micro","t2.medium","t2.large"]
}

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

variable "aws-instance-count" {
  default = 1
}
```

```
 terraform-variable-4.tf > 📁 resource "aws_instance" "ec2" > ⌂ instance_type
1  terraform {
2    required_version = "~> 1.1.4"
3    required_providers {
4      aws = {
5        source  = "hashicorp/aws"
6        version = "3.74.0"
7      }
8    }
9  }
10
11
12 provider "aws" {
13   region = var.aws-region
14 }
15
16 resource "aws_instance" "ec2" {
17   count = var.aws-instance-count
18
19   ami           = var.ami-id
20   instance_type = var.instance-type[1]
21
22   tags = {
23     Name = "myec2-${count.index}"
24   }
25 }
26
```

If we mention out of the default values will get an error.

```
Alok@Alok-DevOps MINGW64 /c/Terraform
$ terraform plan

Error: Invalid index

  on terraform-variable-4.tf line 20, in resource "aws_instance" "ec2":
20:   instance_type = var.instance-type[3]

      var.instance-type is list of string with 3 elements

The given key does not identify an element in this collection value: the
given index is greater than or equal to the length of the collection.
```

MAP:

```
type = map(string)
default = {
  "dev" = "t2.micro"
  "qa" = "t2.medium"
  "uat" = "t2.large"
}

instance_type = var.instance-type["dev"] => t2.micro

*/
```

```
variable "instance-type" {
  type = map(string)
  default = {
    "dev" = "t2.micro"
    "qa" = "t2.medium"
    "uat" = "t2.large"
  }
}
```

```
resource "aws_instance" "ec2" {
  count = var.aws-instance-count

  ami           = var.ami-id
  instance_type = var.instance-type["qa"]

  tags = {
    Name = "myec2-${count.index}"
  }
}
```