

USE MODULE FROM TERRAFORM

Search the module based on your requirements from the link:

<https://registry.terraform.io/browse/modules>

I am going to launch One Instance on AWS Cloud and for that I am going to import the Module which supports the service of EC2.

<https://registry.terraform.io/modules/terraform-aws-modules/ec2-instance/aws/latest>

Provision Instructions

Copy and paste into your Terraform configuration, insert the variables, and run terraform init:

```
module "ec2-instance" {  
  source = "terraform-aws-modules/ec2-i  
  version = "5.2.1"  
}
```

Terraform give the facility to import this module and This module require some variables. After inserting those variables. Then I am going to apply the command

terraform init

When we run the terraform init command, then automatically Terraform downloads the module file in local system. In this module, you can also get help from readme File. In the readme file you can get the overview of the module. Where they talk about:

- **Single EC2 Instance**
- **Multiple EC2 Instance**
- **Spot EC2 Instance**

Main.tf

```
module "ec2-instance" {  
  source = "terraform-aws-modules/ec2-instance/aws"  
  version = "5.2.1"  
  
# Mentioning The Instance Type as T2 Micro, Key Name and AMI ID  
  instance_type = "t2.micro"  
  key_name = "HimanshuTF"  
  ami = "ami-0da59f1af71ea4ad2"  
}
```

Here I am importing the Module from Terraform Where My requirement is to launch the EC2 instance. And In this module, the developer written the code in such a way, based

on the requirement we can pass the values to the variable and they will use these values inside the code.

I am running this code first time. For which I am going to run terraform init.

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 4> .\terraform.exe init

Initializing the backend...
Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/ec2-instance/aws 5.2.1 for ec2-instance...
- ec2-instance in .terraform\modules\ec2-instance

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.11.0

Terraform has been successfully initialized!
```

Terraform will download the module for me.

We can see the code inside the .terraform\modules\ec2-instance directory

```
Directory: C:\Users\Himanshu Kumar\Documents\Terraform\Day 4\.terraform\modules\ec2-instance

Mode                LastWriteTime         Length Name
----                -
d-----          10-08-2023      11:06          .github
d-----          10-08-2023      11:06        examples
d-----          10-08-2023      11:06        wrappers
-a-----          10-08-2023      11:06           589 .editorconfig
-a-----          10-08-2023      11:06           709 .gitignore
-a-----          10-08-2023      11:06          1245 .pre-commit-config.yaml
-a-----          10-08-2023      11:06          1061 .releaserc.json
-a-----          10-08-2023      11:06         29511 CHANGELOG.md
-a-----          10-08-2023      11:06         10349 LICENSE
-a-----          10-08-2023      11:06         23192 main.tf
-a-----          10-08-2023      11:06          7053 outputs.tf
-a-----          10-08-2023      11:06         25768 README.md
-a-----          10-08-2023      11:06          4198 UPGRADE-3.0.md
-a-----          10-08-2023      11:06         13172 variables.tf
-a-----          10-08-2023      11:06           158 versions.tf
```

Now I am going to launch one instance from this module.

After applying the terraform plan command. It will show the output of my desire state.

Terraform plan

```
module.ec2-instance.data.aws_ssm_parameter.this[0]: Reading...
module.ec2-instance.data.aws_partition.current: Reading...
module.ec2-instance.data.aws_partition.current: Read complete after 0s [id=aws]
module.ec2-instance.data.aws_ssm_parameter.this[0]: Read complete after 0s [id=/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2]

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:

```
# module.ec2-instance.aws_instance.this[0] will be created
+ resource "aws_instance" "this" {
  + ami                        = "ami-0da59f1af71ea4ad2"
  + instance_type             = "t2.micro"
  + key_name                   = "HimanshuTF"
  + tags = {
    + Name = "OS 1"
  }
}
```

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

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 4> .\terraform.exe apply -auto-approve
module.ec2-instance.data.aws_ssm_parameter.this[0]: Reading...
module.ec2-instance.data.aws_partition.current: Reading...
module.ec2-instance.data.aws_partition.current: Read complete after 0s [id=aws]
module.ec2-instance.data.aws_ssm_parameter.this[0]: Read complete after 0s [id=/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2]
```

After running the **terraform apply -auto-approve** command.

Terraform will launch the Instance by using the Module.

Instances (1) Info

Refresh

Connect

Instance state

Actions

Launch instances

Find instance by attribute or tag (case-sensitive)

< 1 >

Settings

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input type="checkbox"/>	OS 1	i-0009ca905bd5d7fb0	<div>Running</div>	t2.micro	<div>2/2 checks passed</div>	No alarms +	ap-south-1a

Terraform destroy -auto-approve

Destroying the Environment: Deleting the OS 1 Operating System.

LOOPS In Terraform

In Terraform, loops are used to iterate over a list, map, or set of values and perform the same set of actions for each element. Loops can be quite useful when you want to create multiple similar resources or perform a certain operation on a collection of items.

Terraform primarily supports loops through the use of expressions and functions. Here are some common ways to work with loops in Terraform:

Creating the Variable where I have stored some values.

```
variable "DB" {  
  type = list  
  default = ["a", "b", "c"]  
}
```

Now If I try to print the values for the variable DB, then it will print those values as a list. But if I don't want to print the values as list then for this, I need to use the concept of Loop. And in Terraform we have one loop available that is for.

Outputs:

```
DBOuptut = tolist([  
  "a",  
  "b",  
  "c",  
])
```

```
variable "DB" {  
  type = list  
  default = ["a", "b", "c"]  
}  
  
output "DBOuptutUsingFor" {  
  description = "The Output is Print by For Loop"  
  value = [for x in var.DB : x]  
}
```

Store Every Value of var.DB in X and then Print X. Instead of X we can also call the terraform functions.

```
DBOuptutUsingFor = [  
  "a",  
  "b",  
  "c",  
]
```

```
variable "DB" {  
  type = list  
  default = ["a", "b", "c"]  
}  
  
output "DBOuptutUsingFor-Upper" {  
  description = "The Output is Print by For Loop"  
  value = [for x in var.DB : upper(x)]  
}
```

Outputs:

```
DBOuptutUsingFor-Upper = [  
  "A",  
  "B",  
  "C",  
]
```

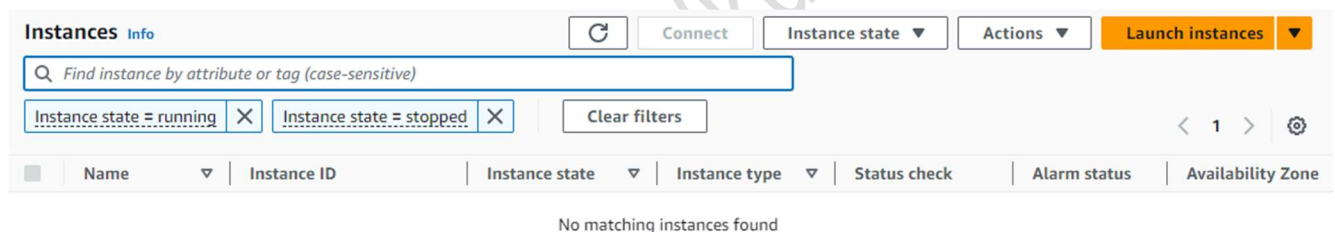
How to use For loop in better Way?

Imagine that I've launched an EC2 Instance dedicated to serving as a web server. In order to ensure optimal accessibility, I'm planning to make adjustments to the Inbound Permissions of the associated Security Group. This alteration will enable anyone interested in accessing my instance to effortlessly connect to my web server. To facilitate this, I intend to authorize permissions for the following protocols: HTTP, HTTPS, and SSH.

- **1. SSH:** One of the protocols I'm employing is SSH, which allows me to establish a secure connection to my instance. This is crucial for remote management and maintenance tasks.
- **2. HTTP:** By configuring the Security Group to permit inbound traffic over the HTTP protocol, visitors will be able to access the web content hosted on my instance. This opens up the possibility of sharing information and services through a web browser.
- **3. HTTPS:** Additionally, I plan to grant access through the HTTPS protocol. This ensures encrypted and secure communication between clients and my web server. Visitors can confidently interact with my web services while safeguarding their data.

In essence, these adjustments to the Inbound Permissions of the Security Group will create a seamless experience for anyone interested in accessing my web server, while maintaining the necessary security measures for controlled and protected interactions.

Currently I don't have any instance running in My AWS Account.



```
resource "aws_security_group" "allow_tls" {
  name           = "allow_tls"
  description    = "Allow TLS inbound traffic"
  vpc_id        = "vpc-00e29075de4bd00ab"

  ingress {
    from_port     = 443
    to_port       = 443
    protocol      = "tcp"
    cidr_blocks   = ["0.0.0.0/0"]
  }

  ingress {
    from_port     = 8080
    to_port       = 8080
    protocol      = "tcp"
    cidr_blocks   = ["0.0.0.0/0"]
  }
}
```

```

ingress {
  from_port      = 22
  to_port        = 22
  protocol       = "tcp"
  cidr_blocks    = ["0.0.0.0/0"]
}

tags = {
  Name = "allow_tls"
}
}

```

After running the **terraform apply -auto-approve**. My Instance will launch with the Inbound rules that I have set.

Instances (1/1) Info

Find instance by attribute or tag (case-sensitive)

Instance state = running X Instance state = stopped X Clear filters

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
OS 1	i-09d754178ff93d750	Running	t2.micro	Initializing	No alarms	ap-south-1a

Instance: i-09d754178ff93d750 (OS 1)

Filter rules

Name	Security group rule ID	Port range	Protocol	Source
-	sgr-0c0694c1edde99093	443	TCP	0.0.0.0/0
-	sgr-093ddfb1f2d97f980	80	TCP	0.0.0.0/0
-	sgr-059118fbfc7be3a01	22	TCP	0.0.0.0/0

Here you can see that I need to mention that Ingress blocks multiple times for achieving my target. But we can also achieve this setup by writing lesser number of code by using dynamic method.

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 4> .\terraform.exe destroy -auto-approve
```

Before that I am destroying my Environment.

Here I need to make the Ingress block dynamic.

And also want to use multiple rules for my OS. For this, Terraform have concept of dynamic.

```

ingress {
  from_port      = 443
  to_port        = 443
  protocol       = "tcp"
  cidr_blocks    = ["0.0.0.0/0"]
}

```

Dynamic Blocks

Dynamic blocks in Terraform provide a way to create multiple blocks of configuration dynamically within a resource or module. This is especially useful when you want to generate repetitive configurations based on input data, like maps or lists. Dynamic blocks allow you to generate configurations for sub-resources, like security group rules, ingress/egress rules, and more, based on a variable.

I have created a variable and mentioned the port Values.

```
variable "SG-Ports" {
  type = list
  default = [22, 8080, 443]
  description = "Apply Mentioned Port Rules in EC2 Instance"
}
```

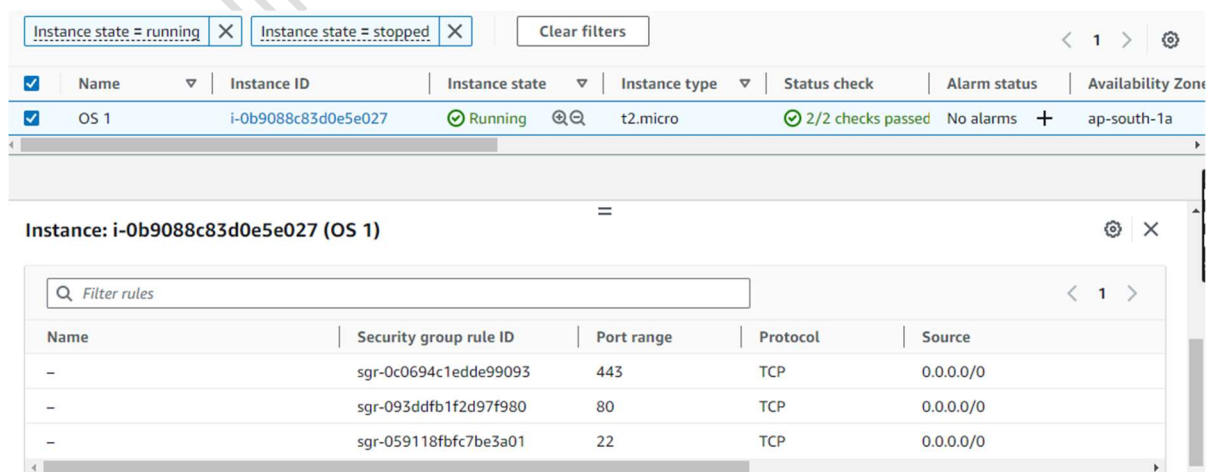
```
resource "aws_security_group" "allow_tls" {

  name          = "allow_tls"
  description   = "Allow TLS inbound traffic"
  vpc_id        = "vpc-00e29075de4bd00ab"

  dynamic "ingress" {
    for_each = var.SG-Ports
    iterator = port
    content {
      from_port   = port.value
      to_port     = port.value
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}
```

When I run terraform apply -auto-approve.

Then it will create 1 Instance and apply the Ingress rule to the particular instance.



The screenshot shows the AWS Management Console interface. At the top, there are filters for 'Instance state = running' and 'Instance state = stopped'. Below the filters, a table lists EC2 instances. The first instance, 'OS 1' with ID 'i-0b9088c83d0e5e027', is in a 'Running' state and has a 't2.micro' instance type. Below the instance list, the details for 'Instance: i-0b9088c83d0e5e027 (OS 1)' are shown. A table lists the security group rules applied to this instance:

Name	Security group rule ID	Port range	Protocol	Source
-	sgr-0c0694c1edde99093	443	TCP	0.0.0.0/0
-	sgr-093ddfb1f2d97f980	80	TCP	0.0.0.0/0
-	sgr-059118fbfc7be3a01	22	TCP	0.0.0.0/0

Locking state

Locking state in the context of Terraform refers to the practice of applying locks or concurrency control mechanisms to prevent multiple simultaneous operations from modifying the same Terraform state file. Terraform state is a crucial component that keeps track of the current state of deployed resources and helps Terraform understand what changes need to be made to your infrastructure when you apply configuration changes.

Without proper locking mechanisms, concurrent operations from different users or automation systems could potentially lead to conflicts, inconsistent states, or data corruption in the state file. Locking ensures that only one operation can modify the state at a time, preventing these issues.

Terraform provides several options for state locking:

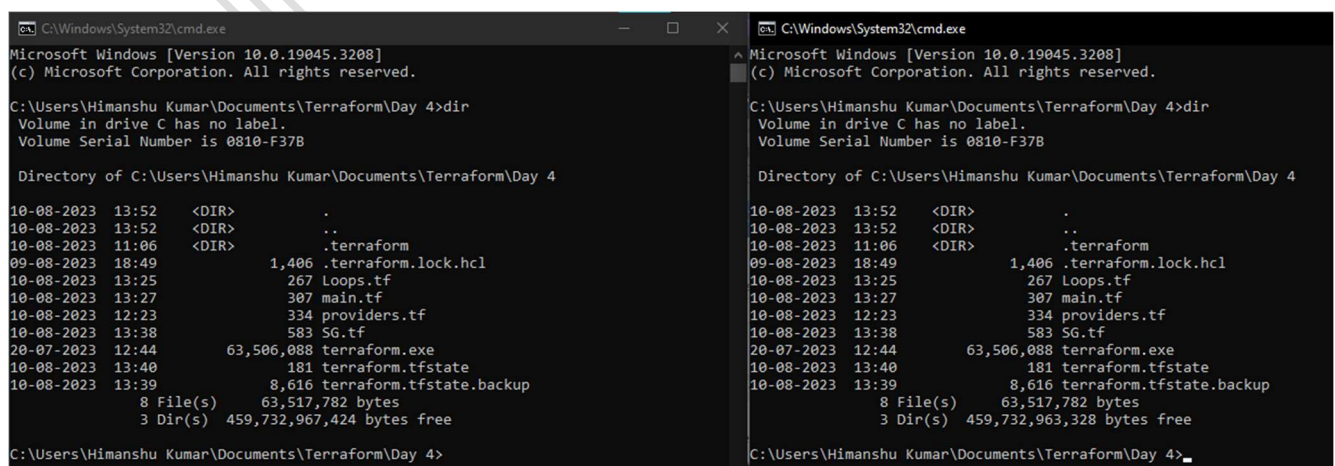
- **Local Locking:** This is the default locking mechanism provided by Terraform when using the built-in local backend. It relies on the filesystem to create a lock file, ensuring only one Terraform command can operate on the state file at a time within a specific directory.
- **Remote Backends with Locking:** When using remote backends (like AWS S3, Azure Storage, or HashiCorp Consul) to store your state, you can enable built-in locking features provided by some of these backends. For example, the AWS S3 backend can be configured to use S3's native object locking to manage concurrent access.
- **External Locking Systems:** Some organizations might use external systems like HashiCorp Consul or HashiCorp Vault for state locking. These systems offer advanced locking capabilities and can integrate with Terraform to provide robust concurrency control.

It's important to choose a locking mechanism that suits your infrastructure, team, and deployment needs. While locking adds a layer of safety, it can also introduce potential delays when multiple operations are waiting for locks to be released. Some organizations might prefer more permissive locking, especially when using automation and CI/CD pipelines, to balance safety and agility.

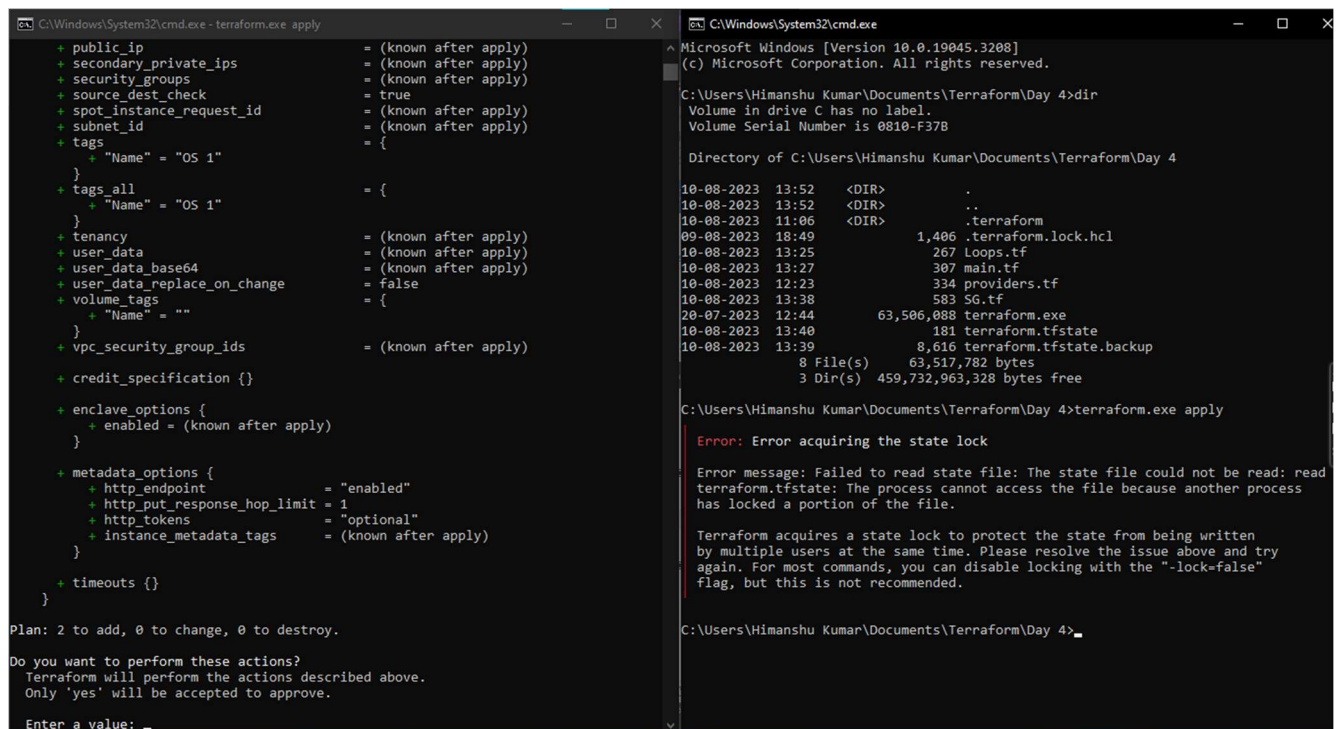
When configuring your Terraform project, make sure to consider your locking strategy as part of your infrastructure management process. Terraform's documentation provides detailed information about locking mechanisms for each backend, so you can choose the one that best fits your requirements.

Let's see the practical, How this thing will work:

I have opened terminal 2 times. Now in one window I am going to run command terraform apply, and in other window I am running the same command.



The image shows two side-by-side terminal windows, both titled "C:\Windows\System32\cmd.exe". Both windows show the same directory listing for "C:\Users\Himanshu Kumar\Documents\Terraform\Day 4". The listing shows a directory named ".terraform" and a file named ".terraform.lock.hcl". The file size is 1,406 bytes. The directory listing also shows other files and directories, including "main.tf", "providers.tf", "SG.tf", "terraform.exe", "terraform.tfstate", and "terraform.tfstate.backup". The file sizes for these files are 267, 307, 334, 583, 63,506,088, 181, and 8,616 bytes respectively. The directory listing also shows the total size of the files (8 File(s) 63,517,782 bytes) and the free space (3 Dir(s) 459,732,963,328 bytes free).



```
C:\Windows\System32\cmd.exe - terraform.exe apply
+ public_ip = (known after apply)
+ secondary_private_ips = (known after apply)
+ security_groups = (known after apply)
+ source_dest_check = true
+ spot_instance_request_id = (known after apply)
+ subnet_id = (known after apply)
+ tags = {
  + "Name" = "OS 1"
}
+ tags_all = {
  + "Name" = "OS 1"
}
+ tenancy = (known after apply)
+ user_data = (known after apply)
+ user_data_base64 = (known after apply)
+ user_data_replace_on_change = false
+ volume_tags = {
  + "Name" = ""
}
+ vpc_security_group_ids = (known after apply)
+ credit_specification {}
+ enclave_options {
  + enabled = (known after apply)
}
+ metadata_options {
  + http_endpoint = "enabled"
  + http_put_response_hop_limit = 1
  + http_tokens = "optional"
  + instance_metadata_tags = (known after apply)
}
+ timeouts {}
}

Plan: 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: _

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Himanshu Kumar\Documents\Terraform\Day 4>dir
Volume in drive C has no label.
Volume Serial Number is 0810-F378

Directory of C:\Users\Himanshu Kumar\Documents\Terraform\Day 4

10-08-2023 13:52 <DIR> .
10-08-2023 13:52 <DIR> ..
10-08-2023 11:06 <DIR> .terraform
09-08-2023 18:49 1,486 .terraform.lock.hcl
10-08-2023 13:25 267 Loops.tf
10-08-2023 13:27 307 main.tf
10-08-2023 12:23 334 providers.tf
10-08-2023 13:38 593 SG.tf
20-07-2023 12:44 63,506,088 terraform.exe
10-08-2023 13:40 181 terraform.tfstate
10-08-2023 13:30 8,616 terraform.tfstate.backup
8 File(s) 63,517,782 bytes
3 Dir(s) 459,732,963,328 bytes free

C:\Users\Himanshu Kumar\Documents\Terraform\Day 4>terraform.exe apply

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.

C:\Users\Himanshu Kumar\Documents\Terraform\Day 4>_
```

In one console window, I executed the `terraform apply` command. Subsequently, in another command prompt, I attempted to run the `terraform apply` command again. However, this resulted in an error due to state locking. This occurrence is attributed to multiple developers concurrently modifying a shared codebase and initiating changes in the cloud simultaneously.

The first developer to execute the command holds the state lock, which temporarily halts state modifications. Consequently, other developers are unable to apply their changes until the initial command's execution concludes and the lock is released. This ensures data consistency and prevents conflicting updates.

While there exists an option to disable the lock, it is generally not advisable to do so. Disabling the lock compromises the integrity of concurrent changes and introduces potential inconsistencies. If no alternative solutions are available, this option can be considered as a last resort.

In conclusion, managing state locking in Terraform is essential to maintain synchronization among developers and uphold the reliability of infrastructure deployments.

Another scenario arises when developers are utilizing multiple devices. In this context, they do not encounter the state lock file error. This discrepancy arises from the fact that in the prior case, the lock file is localized to the specific device. However, in this instance, developers are working across distinct systems. Consequently, if one developer aims to dismantle the environment while another endeavors to create it anew, they could simultaneously execute the `terraform apply` command.

To address this situation, Terraform offers the option of managing the lock file remotely. By doing so, the potential for conflicts due to concurrent operations is mitigated. This centralized approach to state locking ensures that developers working on different devices can seamlessly coordinate their actions, minimizing the likelihood of unintended consequences.

Here the centralized storage is known as Remote Backend. Let's see how to configure the Remote Backend.

OPEN AWS CLOUD > SEARCH : S3

Services

See all 7 results ▶



S3 ☆

Scalable Storage in the Cloud

Create a bucket

Every object in S3 is stored in a bucket. To upload files and folders to S3, you'll need to create a bucket where the objects will be stored.

Create bucket

CLICK ON CREATE BUCKET BUTTON

Give the unique name of your bucket in lower case and then click on create button without doing any changes.

Buckets (1) Info

Buckets are containers for data stored in S3. [Learn more](#)

Find buckets by name

Name	AWS Region	Access	Creation date
terraformstatemanagement	Asia Pacific (Mumbai) ap-south-1	Bucket and objects not public	August 11, 2023, 10:50:52 (UTC+05:30)

I have created my bucket on ap-south-1 with the name of terraformstatemanagement.

Inside this bucket I have created one Folder with the name of webdev. Where I am going to manage my terraform state file.

terraformstatemanagement Info

Objects

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
webdev/	Folder	-	-	-

In terraform we also need to maintain the state lock file. For this I am going to create a Table in Dynamo Db where my state-lock file is going to store.

NOTE: While creating table in Dynamo DB, we need to mention the partition key with the name of "LockID" and it's type is string.

Create table

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

Now we need to use this bucket in our code. Let's see. `# Centralized Module For EC2 Instance`

```
terraform {  
  backend "s3" {  
    bucket = "terraformstatemanagement"  
    key = "webdev/my.tfstate"  
    region = "ap-south-1a"  
  
    # Maintaing Lock File  
    dynamodb_table = "tf-state-file-locking"  
  }  
}
```

After that when I try to run the terraform apply command then I receive one error

Destroy complete! Resources: 2 destroyed.

PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 4> `.\terraform.exe apply`

Error: Backend initialization required, please run "terraform init"

I need to run the terraform init -reconfigure command.

Error: error configuring S3 Backend: no valid credential sources for S3 Backend found.

Please see <https://www.terraform.io/docs/language/settings/backends/s3.html> for more information about providing credentials.

Error: NoCredentialProviders: no valid providers in chain. Deprecated.
For verbose messaging see `aws.Config.CredentialsChainVerboseErrors`

But here again I am facing this issue, Here It says, they can't connect to AWS S3.

For this I have mentioned my access_key and Secret_key in the same block.

```
terraform {
  backend "s3" {
    bucket = "terraformstatemanagement"
    key    = "webdev/my.tfstate"
    region = "ap-south-1"

    # Maintaing Lock File
    dynamodb_table = "tf-state-file-locking"

    access_key = "You-Access-Key"
    secret_key = "You-Secret-Key"
  }

  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 5.0" # You can adjust the version constraint as needed
    }
  }
}
```

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 4> .\terraform.exe init -reconfigure
```

Initializing the backend...

Initializing modules...

Initializing provider plugins...

- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.11.0

Terraform has been successfully initialized!

After that, run the terraform apply command.

Amazon S3 > Buckets > terraformstatemanagement > webdev/

webdev/ Copy S3 URI

Objects | Properties

Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Refresh Copy S3 URI Copy URL Download Open Delete Actions Create folder

Upload

< 1 > ⚙

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	my.tfstate	tfstate	August 11, 2023, 11:28:50 (UTC+05:30)	180.0 B	Standard

Here My state file is now managed in AWS S3.

Now I am trying to run terraform apply command from two different terminals. Then again I am getting the error message which says. Terraform apply command is run by someone, so currently the state is locked.

```

▼ TERMINAL
PS C:\Users\Himanshu Kumar\Documents\Terraform> cd .\Backend
PS C:\Users\Himanshu Kumar\Documents\Terraform\Backend> .\terraform
.exe apply

Error: Error acquiring the state lock

Error message: ConditionalCheckFailedException: The conditional
request failed
Lock Info:
  ID:          9094d039-b6e9-ffb3-23a5-750773b85ab2
  Path:        terraformstatemanagement/webdev/my.tfstate
  Operation:   OperationTypeApply
  Who:         AICPL-D010\Himanshu Kumar@AICPL-D010
  Version:     1.5.3
  Created:     2023-08-11 06:06:13.6604137 +0000 UTC
  Info:

Terraform acquires a state lock to protect the state from being
written

- encrypted          = false -> null
- iops                = 3000 -> null
- tags               = {} -> null
- throughput         = 125 -> null
- volume_id          = "vol-054e09c070091f59a" -> nul
1
- volume_size        = 8 -> null
- volume_type        = "gp3" -> null
}
- timeouts {}
}

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

And this is managed by Dynamo DB.

Terraform ALIAS

Some their is a requirement to launch same provider multiple times. But when we try to run the command. "Terraform apply" then it will fail.

```
# Authentication in AWS Cloud
provider "aws" {
  region      = "ap-south-1"
  access_key  = var.access_key
  secret_key  = var.secret_key
}

provider "aws" {
  region      = "us-east-1"
  access_key  = var.access_key
  secret_key  = var.secret_key
}
```

For this we need to configure the alias,.

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Multipro> .\terraform.exe apply
```

Error: Duplicate provider configuration

on EC2.tf line 19:
19: provider "aws" {

A default (non-aliased) provider configuration for "aws" was already given at EC2.tf:13,1-15. If multiple configurations are required, set the "alias" argument for alternative configurations.

```
# Authentication in AWS Cloud
provider "aws" {
  alias = "India"
  region      = "ap-south-1"
  access_key  = var.access_key
  secret_key  = var.secret_key
}

provider "aws" {
  alias = "Virginia"
  region      = "us-east-1"
  access_key  = var.access_key
  secret_key  = var.secret_key
}
```

Now If we try to run the **terraform apply** command then it won't give error.

```
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: 
```


HOW TO MANAGE SENSITIVE INFORMATION IN TERRAFORM

sensitive = true

By setting sensitive to true, you are instructing Terraform to treat this output value as sensitive information. When you apply the Terraform configuration, the value of this output will not be shown in the console output or stored in the state file in plain text. Instead, it will be masked or redacted to enhance security.

```
output "secretkey" {  
  value = var.secret_key  
  sensitive = true  
}
```

Changes to Outputs:

```
+ secretkey = (sensitive value)
```

TIME SLEEP () IN TERRAFORM

Let's understand this from the given scenario: When we configure our webserver in AWS cloud using Apache httpd. First we need to download the software then start the services and then we are storing our webpages in document root folder.

Here the webserver service take 2 second for starting their service but our code will store in that location. Here we can take the help from time sleep function and stop the code transfer for 5 second. And after 5 second when the service is start successfully then we can transfer the code to our document root.

I am not showing this example here but I have mentioned the syntax of the code:

```
resource "null_resource" "previous" {}  
  
resource "time_sleep" "wait_30_seconds" {  
  depends_on = [null_resource.previous]  
  create_duration = "30s"  
}
```

this code snippet creates a null_resource that doesn't do anything visible but is used to ensure dependencies, and a time_sleep resource that waits for 30 seconds after the null_resource completes its lifecycle. This kind of setup might be used in cases where you need to introduce a delay to allow for the completion of certain tasks before proceeding with subsequent actions.