TF AWS Module use document

Terraform AWS Modules Directory Path> {SCM}::/terraform/AWS/AWS modules/modules/*

AWS Modules (TF):

- Terraform Module supported TF version: terraform (0.13) && providers aws = " version = "~> 3.40"
- Generally, Root module has main file for the service, variable file and dependent files or nested module and output file
- ♣ Most of common and frequently used services are covered in this Module directory path.
- Many modules are category wise folder structured into the respected directory.

Below list of AWS services along with tf templates (Eg: provider_version, provider-aws etc...)

```
$ Is -dR */*
                                        0.1.4.compute/eks/
0.0.provider version/
                                                                         0.1.7.monitoring/aws-budget-alarms-slack/
0.0.provider-aws/
                                        0.1.4.compute/workspace/
                                                                         0.1.7.monitoring/cloudwatch alarm/
O.O. provider-aws-accesskey/(Detailed template)
                                        0.1.5.devops/codebuild/
                                                                         0.1.7.monitoring/metric-alarms-by-multiple-
dimensions/
0.1.2.0.tf-remote-state/{S3/DynamoDb}
0.1.2.random/
                                           0.1.5.devops/codecommit/
                                                                                0.1.7.monitoring/ses/
0.1.1.aws_network/{VPC/FlowLog}
                                           0.1.5.devops/codedeploy/
                                                                                0.1.7.monitoring/sns-topic/
0.1.3.security/ec2-iam-role/
                                           0.1.5.devops/codepipeline/
                                                                                0.1.7.monitoring/sqs/
0.1.3.security/iam-enhanced/
                                           0.1.5.devops/lambda/
                                                                                0.1.8.storage/dynamo-db-table/
0.1.3.security/iam-instance-admin-role/
                                           0.1.6.network/acm/
                                                                                0.1.8.storage/ebs/
0.1.3.security/secrets-manager/
                                           0.1.6.network/cloudfront/
                                                                                0.1.8.storage/efs/
                                                                                0.1.8.storage/fsx/
0.1.3.security/sq-count/
                                           0.1.6.network/directory-service/
0.1.3.security/sq-count-adv/
                                           0.1.6.network/eip/
                                                                                0.1.8.storage/rds/
                                                                                0.1.8.storage/rds-aurora-serverless-db/
0.1.3.security/sq-dynamic-ingress/
                                          0.1.6.network/elb/
                                          0.1.6.network/route-53-hz-internal/
                                                                                0.1.8.storage/s3/
0.1.3.security/sgrule/
                                           0.1.6.network/route-53-hz-public/
0.1.3.security/ssm-patching/*
                                                                               0.1.9.tags/
0.1.4.compute/asq/
                                           0.1.6.network/transit-gateway/
<mark>0.1.4.compute</mark>/ec2-count-auto-recovery/
                                           0.1.6.network/vpn-gateway/{Site / client}*
```

- Setup a project_folder to create Terraform configuration or call Terraform modules.
- Some Terraform template structure like: 0.0.provider_version, 0.0.provider-aws, 0.0.provider-aws-accesskey (Detailed template) in place in module section.
 - 1. Note:
 - Use those provider template / configuration as per need directly inside your project (calling as module not recommended)
 - Use those version template / configuration as per need directly or you call as module (if you are using only resources which present in this "{SCM}.../.. /modules/*" or any other TF configuration which supports similar required version)
- Here Most of the modules are tested and applied at "project_demo". That can be referred as Module EXAMPLES.

- 1. Path for "project_demo":" {SCM}../../modules/projects/project_demo "
- 2. Note: Most of the components has count = 0 or /* */ or Disabled. Enable it while using those modules.
- 3. Some of modules has some example directory with different use cases which is placed in AWS module directory. That also can take as reference while calling modules from the project.

```
$ Is
0.0.0.provider-aws.tf
                                          3.1.0.0.directory-service.tf
0.0.1.0.aws\_network.tf
                                          3.1.0.1.workspace.tf
0.0.1.2.aws_vpc_network_variable.tf
                                         4.0.0.0.devops.tf
0.0.1.3.aws_vpc_flow_log_var.tf
0.0.1.4.common_tag_variable.tf
                                         4.0.0.1.devops.auto.tfvars 4.0.0.1.lambda.tf
0.0.2.1.random_generator.tf
                                          4.0.0.2.rds.tf
0.1.0.0.iam.tf
                                         4.0.0.3.rds-aurora-serverless.tf
0.1.0.0.security-group.tf
1.0.0.0.ec2-vm.tf
                                          5.0.0.0.ecs.tf
                                          5.0.0.1.eks-asg.tf
1.0.0.1.s3.tf
                                          5.0.0.2.eks-ng.tf
1.0.0.2.elb.tf
                                          5.0.0.3.eks-basic.tf
1.0.0.3.asg.tf
                                         cli.terraformrc
1.0.1.0.route53i.tf
                                         cred-var.tf
1.1.0.0.sns.tf
                                         dev.auto.tfvars
                                          info-common-value-for-all-module.txt
1.1.0.1.sqs.tf
1.1.0.2.ses.tf
                                          lambda
                                         terraform.auto.tfvars
1.1.1.0.cw.tf
1.1.1.1.budget-alerm.tf 2.0.0.1.vpngateway.tf
                                         z_backend-remote-tfcloud-state.tf
                                         z_backend-remote-tfcloud-state.hcl
2.0.0.2.client-gateway.tf
                                         tf-file.terraformignore
2.0.0.3.transit-gateway.tf
                                          version.tf
2.0.0.4.wafv2.tf
                                          z.backend-remote-s3-state.tf
3.0.0.0.cdn-acm.tf
```

How to call a module?

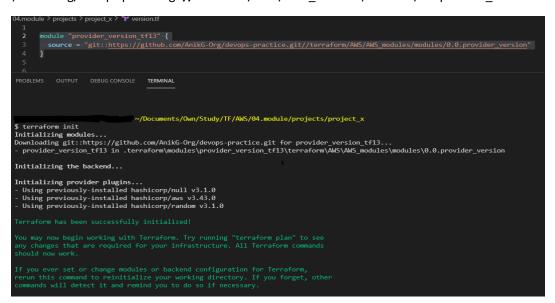
Here many ways to call a module form source(3 way example given):

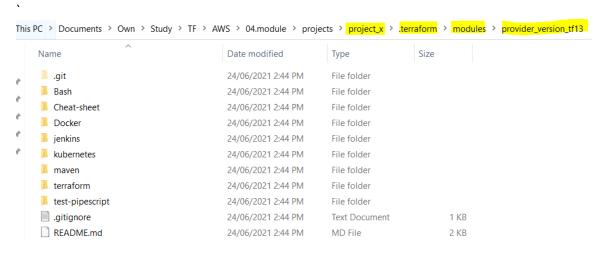
Eg: Project directory (Module) here called: project_x. Note: If you are calling Private SCM, make sure machine has preconfigured connectivity with the target SCM.

1. Directly call module from Git:

 $\textbf{source} = \text{"git::https://github.com/AnikG-Org/devops-practice.git//terraform/AWS/AWS_modules/modules/0.0.provider_version"}$

Note: use // after git repo path

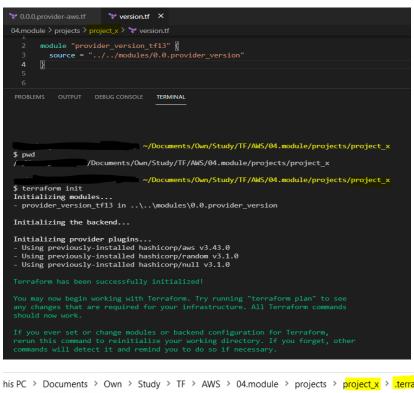


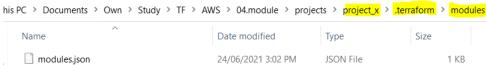


2. Clone modules locally and call those from local:

When you have local source: Eg:

source = "../../modules/0.0.provider version"





modules.json (mapped this module with local directory path(TF will take care this config)) >>>

 $\label{lem:control_$

3. Directly call module from AWS Codecommit: source = "git::https://git-codecommit.us-east-1.amazonaws.com/v1/repos/terraform-module-aws//modules//0.0.provider_version"

Note: use // after repolink to reach module main file directory.

```
module "provider_version_tf13" {
    source = "git::https://git-codecommit.us-east-1.amazonaws.com/v1/repos/terraform-module-aws//modules//0.0.provider_version"
}

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

"/Documents/Own/Study/TF/AWS/04.module/projects/project_x
$ terraform init
Initializing modules...
Downloading git::https://git-codecommit.us-east-1.amazonaws.com/v1/repos/terraform-module-aws for provider_version_tf13 in .terraform\modules\provider_version_tf13\modules\0.0.provider_version

Initializing the backend...

Initializing provider plugins...
- Using previously-installed hashicorp/aws v3.42.0
- Using previously-installed hashicorp/random v3.1.0
- Using previously-installed hashicorp/random v3.1.0

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.
```

✓ Other sources: Terraform Link for better understanding: https://www.terraform.io/docs/language/modules/sources.html

✓ Use command: **terraform init** to initialize a working directory containing Terraform configuration files (necessary module/plugins).

Starting a new project?

- 1. Create a Project directory
- 2. Create an AWS Provider configuration based on needs. (Can take reference of provider module).
- 3. Create a version configuration by calling the existing module or independent based on use cases.
- 4. Almost all of services I made mandate String tags. >>>>

Environment = var.environment

Project = var.project SCM = var.git_repo

ServiceProvider = var. ServiceProvider Created_Via = "Terraform IAAC"

All of those default values is "", means empty.

You can use those variable tags at child module level to define those tag key values hardcoded.

Or you can variable there as well (create a comm-tag-var.tf to keep those variable files)

And add actual value at terraform.tfvars/terraform.auto.tfvars file. For central input management for all variables.

- 5. Not recommended to store user credential files at configuration/SCM.
- 6. ".gitignore" file is very important while using SCM with TF codes. Security reason: Need to avoid pushing system generated file/logs or cred files or config to SCM. Template is already available in this repo.
- 7. Some of the modules are mixture of multiple Services, understand the code and make use of them on needs.

```
tags = merge(
  var.additional_tags,
  {
   Name = "${var.ec2tagname}${count.index + 001}"
   #timestamp = format("Created or Modified Da
   instance_sequence = count.index + 001
   Environment = var.environment
   Created_Via = "Terraform IAAC"
   Project = var.project
   SCM = var.git_repo
   ServiceProvider = var.ServiceProvider
  },
   var.tags
```

How to use modules / write terraform code by calling existing modules?

Eg: Some screenshot added here: S3>>>

ec2_count_autorecovery>>>

- So here basically how to call module and create your code already defined above screenshot.
 Point:
 - Name Your local module as per requirement. Call module from exact source
 - Use code variables exactly defined at root level. Example of ec2_count_autorecovery>>

"instance_count" we got this name/variable from master module. Understand what is the use case & identify the need to refer master module.

Once we reached the source code main file path [= "../../modules/0.1.4.compute/ec2-count-auto-recovery"], You can see below.

```
resource "aws_instance" "ec2_count"
                                   = var.instance count
       ami
                                   = var.ami
       instance_type
                                   = var.instance type
       iam instance profile
                                   = var.iam instance profile
       subnet id
                                   = var.subnet id
15
       key name
                                   = var.key name
       monitoring
                                   = var.monitoring
       associate_public_ip_address = var.associate_public_ip_address
       user data
                                   = var.user data
       user data base64
                                   = var.user data base64 #filebase64
                                   = var.security_groups
       security_groups
       vpc_security_group_ids = var.vpc_security_group_ids #If y
```

This is instance has "count" virialized as "var.instance_count".

And in the variable file/section you can see >>>

```
instance_count variable type is Number & default value is 1.
```

```
#EC2 variable
variable "instance_count" {

type = number

default = 1
```

- So, here in the Project/Child module you can define instance_count with your expected value or you can ignore as well when you are good with existing module default value, which is already defined as 1.
- In example files defined with some required example configuration, if you need beyond than this, and if that feature already in place in the root module, please go through the master module, use the feature variable name at child module with required parameter.

EG: in ec2_count_autorecovery example chile module:

I've not defined ebs_optimized because I left that with default value from root variable.

```
variable "ebs_optimized" {{
    description = "If true, the launched EC2 instance will be EBS-optimized"
    type = bool
    default = false
```

If you need to use that feature in can define at child project/ module with variable value (eg: bool type = true). Keep variable defined type in mind while using at module.

Point Note: Write your code while calling root module according to root module definition.

- You can use same modules as many as possible with different naming based on case to case.
- Generally Main module has main file for the service, variable file and dependent files and output file.
 - If you need to use other modules output value as different/dependent modules input, you can make use of output feature of the service from main module.
 - For better understanding visit Root module output file.
 - Output Values TF document: https://www.terraform.io/docs/language/values/outputs.html

```
1 output "vpc_id" {
2    value = aws_vpc.default.id
3    }
4 output "private_subnet_ids" {
5    value = aws_subnet.private[*].id
6    }
```

- ♣ Terraform Backend with AWS native services:
 - 1. Terraform Backend used to keep TF state file at remote location and safe.
 - You can use module "0.1.2.0.tf-remote-state/remote-state-req-components-enable-if-required" (OPTIONAL)
 Which contains S3 and DynamoDB.
 Enable and configure services based on requirement.
 - create a S3 backend configuration with target S3 and DynamoDB table (OPTIONAL- used for "terraformstate-lock")

```
terraform {
  backend "s3" {|
    bucket = module.s3_remote_state.bucket_id # "remote-backends"
  key = "tfstate/terraform.tfstate"
  region = var.provider_region
  # access_key = "YOUR-ACCESS-KEY"
  # secret_key = "YOUR-SECRET-KEY"
  dynamodb_table = module.s3_remote_state.dynamodb_table_id #"terraform-lock"
}
```