
Terraform
MODULES

❖ MODULES:

- Modules are containers for multiple resources that are used together.
- A module consists of a collection of **.tf** and/or **.tf.json** files kept together in a directory.
- Modules are the main way to package and **reuse resource configurations** with Terraform.
- Most commonly, modules use:
  - **Input variables** to accept values from the calling module.
  - **Output values** to return results to the calling module, which it can then use to populate arguments elsewhere.
  - **Resources** to define one or more infrastructure objects that the module will manage.

MODULES TYPES:

ROOT MODULE:

- Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the **.tf files** in the main working directory.

CHILD MODULES:

- A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration.
- A module that has been called by another module is often referred to as a child module.
- Child modules can be called multiple times within the same configuration, and multiple configurations can use the same child module.

PUBLISHED MODULES:

- In addition to modules from the local filesystem, Terraform can load modules from a **public** or **private registry.** This makes it possible to publish modules for others to use, and to use modules that others have published.
- The Terraform Registry hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure.
- These modules are free to use.

➢ WHAT PROBLEMS DO TERRAFORM MODULES SOLVE:

- Code Repetition
- Lack of Code Clarity
- Lack of Compliance
- Human Error

➢ MODULE BLOCKS:

CALLING A CHILD MODULE:

- To call a module means to include the contents of that module into the configuration with specific values for its input variables.
- Modules are called from within other modules using module blocks:

```
module "servers" {
  source = "./web-servers"
  version = "0.0.5"
  servers = 3
}
```

  - The **source** argument is mandatory for all modules.
  - The **version** argument is recommended for modules from a registry.
  - Most other arguments correspond to **input variables** defined by the module.
  - Terraform defines a few other meta-arguments that can be used with all modules, including **for_each** and **depends_on.**

META-ARGUMENTS:

count: Creates multiple instances of a module from a single module block.

for_each: Creates multiple instances of a module from a single module block.

providers: Passes provider configurations to a child module. If not specified, the child module inherits all of the default (un-aliased) provider configurations from the calling module.

depends_on: Creates explicit dependencies between the entire module and the listed targets.

➢ ACCESSING MODULE OUTPUT VALUES:

- The resources defined in a module are encapsulated, so the calling module cannot access their attributes directly. However, the child module can declare output values to selectively export certain values to be accessed by the calling module.

```
resource "aws_ec2" "example" {
  # ...

  instances = module.servers.instance_ids
}
```

REPLACING RESOURCES WITHIN A MODULE:

- You may have an object that needs to be replaced with a new object for a reason that isn't automatically visible to Terraform, such as if a particular virtual machine is running on degraded underlying hardware.

  #terraform plan -replace=module.example.aws_instance.example

➢ MODULE SOURSE:

LOCAL MODULE:

- A local module is a module that wasn’t published in any registry and when it is sourced, it is using the path to that particular module.

PUBLISHED MODULE:

- A published module refers to a module that has been pushed to a Terraform Registry, or even simply on a VCS and has a tag associated with it. When a published module is sourced, the URL of that module is used either from the registry or from the VCS itself.

➢ REMOVING MODULES:

- By default, after you remove the module block, Terraform will plan to destroy any resources it is managing that were declared in that module. This is because when you remove the module call, that module's configuration is no longer included in your Terraform configuration.

```
removed {
  from = module.example


  lifecycle {
    destroy = false
  }
}
```

  - The **from argument** is the address of the module you want to remove, without any instance keys (such as "module.example[1]").
  - The **lifecycle block** is required. The **destroy argument** determines whether Terraform will attempt to destroy the objects managed by the module or not. A value of false means that Terraform will remove the resources from state without destroying them.

EXAMPLE OF MODULES:

STEP1: Create a MODULE-1 in the root project:

```
#mkdir module-1
#cd module-1
#vim main.tf
terraform {
  required_version = ">=0.12"
}
```

```
resource "aws_instance" "webserver1" {
   ami = "ami-080e1f13689e07408"
   instance_type = "t2.micro"
   key_name= "ram"
   vpc_security_group_ids = [aws_security_group.main.id]
   tags = {
     Name = "WEB-SERVER1"
     Env = "DEV-SERVER"
   }

  user_data = <<-EOF
     #!/bin/sh
     sudo apt-get update
     sudo apt install -y apache2
     sudo systemctl status apache2
     sudo systemctl start apache2
     sudo chown -R $USER:$USER /var/www/html
     sudo echo "<html><body><h1> HELLO...! THIS IS MODULE-1
</h1></body></html>" > /var/www/html/index.html
     EOF
         }

resource "aws_security_group" "main" {
   name        = "WEBSERVER1-SG-1"
   description = "Webserver2 SG for EC2 Instance"
```

```
  ingress {
    from_port   = 80
    protocol    = "TCP"
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
    }
  ingress {
    from_port   = 22
    protocol    = "TCP"
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
  }
egress {
    from_port   = 0
    protocol    = "-1"
    to_port     = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}

output "PublicIPAddress" {
 value       = aws_instance.webserver1
 description = "AWS EC2 instancce Public IP"
}
```

```
output "InstanceState_Webserver1" {
 value       = aws_instance.webserver1
 description = "AWS EC2 instancce State"
}
```

CREATE A VARIABLE FILE:

```
$vim variables.tf
variable "web_instance_type" {default = "t2.micro"}
variable "ami_id" {default = "ami-0767046d1677be5a0"}
```

STEP 2: Create a MODULE-2 in the root project:

```
#mkdir module-2
#cd module-2
#vim main.tf
terraform {
  required_version = ">=0.12"
}

resource "aws_instance" "webserver2" {
    ami = "ami-080e1f13689e07408"
    instance_type = "t2.micro"
    key_name= "ram"
    vpc_security_group_ids = [aws_security_group.main.id]
    tags = {
```

```
    Name = "Web-SERVER2"
    Env = "PROD-SERVER"
  }

user_data = <<-EOF
    #!/bin/sh
    sudo apt-get update
    sudo apt install -y apache2
    sudo systemctl status apache2
    sudo systemctl start apache2
    sudo chown -R $USER:$USER /var/www/html
    sudo echo "<html><body><h1>HELLO...! THIS IS MODULE-2
</h1></body></html>" > /var/www/html/index.html
    EOF
}

resource "aws_security_group" "main" {
  name        = "WEBSERVER-SG-2"
  description = "Webserver2 SG for EC2 Instance"
 ingress {
  from_port   = 80
  protocol    = "TCP"
  to_port     = 80
  cidr_blocks = ["0.0.0.0/0"]
 }
```

```
 ingress {
  from_port   = 22
  protocol    = "TCP"
  to_port     = 22
  cidr_blocks = ["0.0.0.0/0"]
 }
 egress {
  from_port   = 0
  protocol    = "-1"
  to_port     = 0
  cidr_blocks = ["0.0.0.0/0"]
 }
}
output "PublicIPAddress" {
 value       = aws_instance.webserver2
 description = "AWS EC2 instancce Public IP"
}
output "InstanceState_Webserver2" {
 value       = aws_instance.webserver2
 description = "AWS EC2 instancce State"
}
```

CREATE A VARIABLE FILE:

```
$vim variables.tf
variable "web_instance_type" {default = "t2.micro"}
variable "ami_id" {default = "ami-0767046d1677be5a0"}
```

STEP3: Create a main.tf file in the main root project:

```
module "webserver-1" {
  source = "./module-1"
}
output "PublicIPAddress_Webserver1" {
  value       = module.webserver-1.PublicIPAddress.public_ip
  description = "AWS EC2 instancce Public IP"
}
output "InstanceState_Webserver1" {
value = module.webserver-1.InstanceState_Webserver1.instance_state
description = "AWS EC2 instancce State"
}
module "webserver-2" {
  source = "./module-2"
}
output "PublicIPAddress_Webserver2" {
  value       = module.webserver-2.PublicIPAddress.public_ip
  description = "AWS EC2 instancce Public IP"
}
output "InstanceState_Webserver2" {
value = module.webserver-2.InstanceState_Webserver2.instance_state
description = "AWS EC2 instancce State"
}
$terraform fmt     $terraform validate
$terraform plan    $terraform apply
```