A Terraform module is a container for multiple resources that are used together. Modules are the primary way to package and reuse resource configurations with Terraform. They can be thought of as building blocks that abstract away the complexity of the resource definitions and can be reused and shared across different environments and projects.

Here’s a guide on how to create, use, and manage Terraform modules:

**Creating a Terraform Module**

1. **Module Directory Structure**:
   * Create a directory for your module. Inside this directory, you typically have the following files:
     + main.tf: The main file where the resource definitions go.
     + variables.tf: Define input variables for the module.
     + outputs.tf: Define output values from the module.
     + README.md: Documentation for the module.

**Example Structure**:

my-module/

├── main.tf

├── variables.tf

├── outputs.tf

└── README.md

**Define Resources in main.tf**:

resource "aws\_instance" "example" {

ami = var.ami\_id

instance\_type = var.instance\_type

}

**Define Variables in variables.tf**:

variable "ami\_id" {

description = "The AMI ID to use for the instance"

type = string

}

variable "instance\_type" {

description = "The type of instance to start"

type = string

default = "t2.micro"

}

**Define Outputs in outputs.tf**:

output "instance\_id" {

description = "The ID of the instance"

value = aws\_instance.example.id

}

**Document the Module in README.md**:

markdown

# My Module

This module creates an AWS EC2 instance.

## Inputs

- `ami\_id`: The AMI ID to use for the instance.

- `instance\_type`: The type of instance to start (default: `t2.micro`).

## Outputs

- `instance\_id`: The ID of the instance.

**Using a Terraform Module**

1. **Create a Root Configuration**: In your main Terraform configuration, you can call the module like this:

hcl

1. module "my\_module" {
2. source = "./path/to/my-module"
3. ami\_id = "ami-12345678"
4. instance\_type = "t2.large"
5. }
6. **Initialize and Apply**: Run terraform init to initialize the configuration and download the module, then terraform apply to create the resources.

**Best Practices**

1. **Version Control**: Use version control for your modules, especially if you are publishing them for others to use. Tag versions and use the version attribute when specifying the module source.
2. **Module Registry**: Use the Terraform Registry to share your modules with others. This makes it easy for others to find and use your modules.
3. **Input Validation**: Use variable validation rules to enforce constraints on the inputs.
4. **Outputs for Interoperability**: Clearly define outputs to allow other modules and configurations to use the information your module provides.
5. **Modular Design**: Break down complex configurations into smaller, reusable modules to keep things manageable and modular.

**Example of Calling a Public Module**

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

version = "2.0.0"

name = "my-vpc"

cidr = "10.0.0.0/16"

azs = ["us-west-1a", "us-west-1b", "us-west-1c"]

private\_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]

public\_subnets = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

tags = {

Terraform = "true"

Environment = "dev"

}

}

By following these guidelines and best practices, you can create, use, and manage Terraform modules effectively, enabling you to build scalable and maintainable infrastructure configurations.