**Creating a Dynamic Terraform Script**

When managing infrastructure as code with Terraform, one of the best practices is to make your scripts dynamic and reusable. This tutorial walks you through creating a Terraform script for dynamically provisioning EC2 instances in AWS, pulling required values such as subnet and security group IDs from external modules. This approach improves flexibility, reduces hardcoding, and aligns with DevOps best practices.

**Why Dynamic Scripts?**

Dynamic scripts allow you to:

* Scale configurations easily.
* Reuse code across different environments.
* Integrate seamlessly with existing modules.
* Minimize maintenance effort.

**Overview of the Code Structure**

This Terraform script provisions multiple EC2 instances dynamically. Subnet and security group IDs are passed as variables from other modules, enhancing modularity and scalability.

**Provider Configuration**

The script begins with defining the AWS provider:

Copy

provider "aws" {

region = "us-east-1"

}

This ensures Terraform interacts with AWS services in the specified region (us-east-1 in this case).

**EC2 Resource Configuration (EC2.tf)**

The aws\_instance resource dynamically provisions EC2 instances using a for\_each loop:

resource "aws\_instance" "one" {

for\_each = var.ec2\_variables

ami = each.value.ami

instance\_type = each.value.instance\_type

vpc\_security\_group\_ids = [var.sg\_ids]

subnet\_id = var.subnet\_ids[each.value.subnet\_key]

tags = {

Name = each.value.name

}

}

**Key Highlights:**

**Dynamic Resource Creation:**  
The for\_each statement iterates over var.ec2\_variables, creating one EC2 instance for each entry in the map.

**AMI and Instance Type:**  
These values are dynamically fetched from the variable map to ensure instances are provisioned with the correct configuration.

**Security Group and Subnet IDs:**  
Values for vpc\_security\_group\_ids and subnet\_id are passed as variables (var.sg\_ids and var.subnet\_ids). These are derived from other modules, promoting modularity.

**Tagging:**  
Each instance is tagged with a unique name for easy identification.

**Input Variables (variable.tf)**

The input variables provide flexibility by allowing different configurations for subnets, security groups, and EC2 instances:

variable "subnet\_ids" {

type = map(string)

}

variable "sg\_ids" {

type = string

}

variable "ec2\_variables" {

type = map(object({

name = string

ami = string

instance\_type = string

subnet\_key = string

}))

default = {

"vm1" = {

name = "vm1"

ami = "ami-06b21ccaeff8cd686"

instance\_type = "t2.micro"

subnet\_key = "subnet\_1"

}

"vm2" = {

name = "vm2"

ami = "ami-0ddc798b3f1a5117e"

instance\_type = "t2.micro"

subnet\_key = "subnet\_2"

}

}

}

* **subnet\_ids**: A map of strings where keys represent logical subnet names and values are the actual subnet IDs.
* **sg\_ids**: A single string for the security group ID. While this is simplified here, it could be extended to support multiple security groups.
* **ec2\_variables**: This map allows configuring multiple EC2 instances dynamically, with instance-specific details like name, AMI, instance type, and associated subnet key.
* **Customization**: This structure can easily be expanded to include additional properties (e.g., user data, key pairs, or additional tags).

**Outputs (output.tf)**

The output variables help retrieve and display useful information about the created resources:

output "instance\_id" {

value = { for k, v in aws\_instance.one : k => v.id }

}

output "public\_ip" {

value = { for k, v in aws\_instance.one : k => v.public\_ip }

}

* **instance\_id**: This output creates a map where each key corresponds to an instance name, and the value is the instance ID.
* **public\_ip**: Similarly, this output maps instance names to their respective public IPs.

These outputs are especially useful for debugging or feeding the data into other systems or modules.

**Calling Subnet and Security Group IDs from Modules**

The subnet\_ids and sg\_ids are called from VPC module and Security group modules respectively into the main.tf.

provider "aws" {

region = "us-east-1"

}

module "vpc" {

source = "./Modules/VPC"

}

module "sg" {

source = "./Modules/Security group"

}

module "ec2" {

source = "./Modules/EC2"

subnet\_ids = module.vpc.subnet\_ids

sg\_ids = module.sg.Sg\_id

}

The subnet\_ids and sg\_ids are called dynamically from the modules of VPC and Security group this eliminates the manual entry of the values.

**Execution Steps**

1. terraform init :- to initialize the terraform in the folder
2. terraform validate :- to validate the whole script
3. terraform plan :- to make the plan of resource deployment
4. terraform apply --auto-approve :- to deploy the resources into the AWS cloud

**Final Thoughts**

By leveraging dynamic variables and external module outputs, this Terraform script simplifies the process of provisioning EC2 instances while maintaining flexibility and modularity. This approach aligns with modern infrastructure-as-code practices, making it a robust choice for managing AWS resources.