Terraform Variables, Count, and Conditional Expressions — Beginner to Pro Guide

Author: Pratham Patel

1. Understanding Terraform Variables

Variables in Terraform allow you to parameterize configurations — making the code reusable, dynamic, and environment-independent.

Example variable files:

```
variables.tf → stores variable definitions
dev.tfvars → values for development
prod.tfvars → values for production
```

2. Restricting Type of Data

Terraform lets you define what kind of data each variable can accept. This helps catch input errors early.

Type	Description	Example
string	A simple text value	"hello"
number	Numeric value	5
bool	True or False	true
list	Ordered collection	["a", "b", "c"]
map	Key-value pairs	{name = "Alice", age = 24}

3. Using 'count' to Create Multiple Resources

The count meta-argument allows you to create multiple instances of a resource with minimal code.

```
resource "aws_iam_user" "that" { name = var.users[count.index] count = 3 }
```

This code creates 3 IAM users automatically, using var.users list values with count.index.

4. Conditional Expressions (Ternary Operator)

Conditional expressions in Terraform allow logic-based resource configuration using a simple syntax:

```
Syntax: condition ? true_val : false_val
```

Example:

```
resource "aws_instance" "example" { instance_type = var.environment == "production" &&
var.region == "us-east-1" ? "m5.large" : "t2.micro" ami = "ami-1245678" }
```

5. Best Practices

■ Keep variables in a dedicated variables.tf file

■ Maintain separate .tfvars files for each environment

■ Use type constraints to prevent input mistakes

■ Use count for identical resources and for_each for named ones

■ Keep conditional logic simple and readable

Prepared by: Pratham Patel