Lab 13 – Terraform Data Sources (A New Core Concept Explained Simply)

Creator: Sandeep Kumar Sharma

Learning Objectives

- Understand what **data sources** are in Terraform.
- Learn why data sources are used in real-world Terraform projects.
- Learn how to fetch existing AWS resources dynamically.
- Use data sources to retrieve AMIs, VPC IDs, subnets, and security groups.
- Deploy EC2 using values retrieved from Terraform data sources.

Learning Outcome

By the end of this lab, the learner will be able to: - Explain Terraform data sources with practical real-world examples. - Use AWS AMI data sources to fetch the latest AMI. - Use VPC and subnet data sources to reference existing AWS networks. - Combine data sources with resources to deploy EC2 without hardcoding values.

Concept Explanation (Natural Style)

Let's understand Terraform **data sources** with a simple example.

Imagine you are traveling and booking a hotel. Instead of remembering the exact address of every hotel, you simply search online: - "Hotels near me" - "Available rooms today"

The website fetches **existing data** and shows it to you.

Terraform data sources work exactly like this.

👉What is a Terraform Data Source?

A data source allows Terraform to **fetch existing information** from AWS.

This information can be anything like: - Latest AWS AMI - Default VPC ID - List of subnets - Existing security groups - Existing S3 buckets - Existing IAM roles

👉Why do we use Data Sources?

Because in real-world projects, you rarely create everything from scratch.

Often, you need to use: - Existing VPC - Existing subnets - Existing AMIs - Existing key pairs

Instead of hardcoding IDs (which is risky and causes breakage), Terraform dynamically fetches them.

👉Resource vs Data

Terraform Resource	Terraform Data
Creates something	Reads something
Example: create EC2	Example: fetch an AMI
Requires apply	Only needed in plan/apply

😠Part 1: Create Project Structure

```
mkdir terraform-lab13-data-sources
cd terraform-lab13-data-sources
```

Create file:

```
touch main.tf
```

😠Part 2: Write Terraform Code

We will: 1. Fetch **latest Amazon Linux 2 AMI** using a data source. 2. Fetch **default VPC**. 3. Fetch **default subnets**. 4. Deploy EC2 using only fetched information.

---

main.tf

```
provider "aws" {
  region = "ap-south-1"
}
```

```
# -------------------------------------------
# Data Source 1 – Fetch Latest Amazon Linux 2 AMI
# -------------------------------------------
data "aws_ami" "amazon_linux" {
  most_recent = true

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }

  owners = ["amazon"]
}

# -------------------------------------------
# Data Source 2 – Fetch Default VPC
# -------------------------------------------
data "aws_vpc" "default" {
  default = true
}

# -------------------------------------------
# Data Source 3 – Fetch Default Subnets
# -------------------------------------------
data "aws_subnets" "default_subnets" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default.id]
  }
}

# -------------------------------------------
# Create EC2 using data sources
# -------------------------------------------
resource "aws_instance" "lab13_ec2" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"
  subnet_id     = data.aws_subnets.default_subnets.ids[0]

  tags = {
    Name        = "Terraform-Lab13-EC2"
    AMI_Type    = "Amazon Linux 2"
    VPC_Used    = data.aws_vpc.default.id
  }
}

# Outputs
output "ami_id" {
  value = data.aws_ami.amazon_linux.id
}

output "vpc_id" {
  value = data.aws_vpc.default.id
}

output "subnet_used" {
  value = data.aws_subnets.default_subnets.ids[0]
}
```

😠Explanation of What We Just Did

👉🏽 `data "aws_ami"` → fetched latest AMI

No hardcoded AMI IDs.

👉🏽 `data "aws_vpc"` → fetched default VPC

No need to manually look for VPC ID.

👉🏽 `data "aws_subnets"` → fetched available subnets

Terraform dynamically retrieves subnet IDs.

👉🏽EC2 instance was created using:

- AMI from data source
- Subnet from data source
- VPC from data source

This is **real corporate-level practice**.

😠Part 3: Initialize Terraform

```
terraform init
```

😠Part 4: Plan

```
terraform plan
```

Terraform will show: - Fetched AMI - Fetched VPC - Fetched subnets - Plan to create EC2

😠Part 5: Apply

```
terraform apply
```

Type **yes**.

😠Part 6: Validate in AWS Console

Go to: - **EC2 → Instances** → Terraform-Lab13-EC2

Check: - AMI ID - Subnet - VPC

These should match the outputs.

😠Part 7: Destroy (Optional)

```
terraform destroy
```

Type **yes**.

Summary

In this lab, you learned: - What Terraform data sources are. - Why data sources are essential in real-world projects. - How to fetch latest AMIs, VPC IDs, and subnet IDs. - How to create EC2 without hardcoding values. - How Terraform reads existing infrastructure safely.

In the next lab, we will explore **Terraform Conditionals and Expressions** to make configurations more dynamic.

End of Lab 13