Terraform

1. What is Terraform?

- **Tool type**: Open-source IaC (Infrastructure as Code) tool by HashiCorp.
- **What it does**: Provisions and manages infrastructure on cloud providers (AWS, Azure, GCP, etc.) or even on-prem.
- **Why it's popular**:
  - *Declarative*: you describe **what you want**, not **how to get it**.
  - *Idempotent*: run the same code multiple times, it won't recreate resources if they already exist.
  - *Provider-based*: supports almost every service via *providers* (AWS, Kubernetes, Docker, GitHub, etc.).

---

2. Terraform Architecture

Terraform has a simple flow:

1. **Write Code** → Define infrastructure in .tf files (using HCL, HashiCorp Configuration Language).
2. **Initialize** → Terraform downloads the required provider plugins.
3. **Plan** → Shows what changes will happen.
4. **Apply** → Executes and creates/updates infrastructure.
5. **Destroy** → Tears down infrastructure.

🔑 Core components:

- **Providers**: Plugins for cloud/services (AWS, Azure, etc.).
- **Resources**: Actual things you create (EC2, S3 bucket, VPC).
- **State File**: Terraform's "memory" (terraform.tfstate) — tracks real infrastructure vs. your code.
- **Modules**: Reusable sets of Terraform code.

---

3. Installation

Linux

```
# Install Terraform
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update

sudo apt install terraform
```

Verify

```
terraform -v
```

4. Basic File Structure

A simple project looks like this:

my-terraform-project/

├── main.tf # main configuration file (infrastructure definition)

├── variables.tf # variables (inputs)

├── outputs.tf # outputs (things Terraform shows after apply)

├── provider.tf # provider configuration (AWS, etc.)

├── terraform.tfstate # state file (auto-generated)

---

5. Terraform Commands

You'll use these constantly:

- terraform init → Initialize directory, download provider plugins.
- terraform fmt → Format code.
- terraform validate → Validate syntax.
- terraform plan → Preview changes.
- terraform apply → Apply changes (creates infra).
- terraform destroy → Destroy infra.
- terraform show → Shows state.
- terraform state list → Shows tracked resources.

---

6. Writing Your First Infrastructure (AWS Example)

Let's create a simple **EC2 instance** on AWS.

👉 Prerequisites:

- AWS CLI installed (aws configure to set up credentials).
- An IAM user with programmatic access.

Step 1: provider.tf

```
provider "aws" {
  region = "us-east-1"
}
```

Step 2: aws-ami.tf

```
data "aws_ami" "amiID" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  owners = ["099720109477"]
}
output "instance_id" {
  description = "AMI ID of ubuntu server"
  value       = data.aws_ami.amiID.id
}
```

Step 3: Generate Public and Private keys in local machine

Command:

```
$ ssh-keygen
```

After that provide name to create in current folder. Copy the public key content in the following code.

Step 3: key-pair.tf

```
resource "aws_key_pair" "shah-key" {
  key_name   = "shah-key"
  public_key = "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCocmqrCRbrIU4pP6ufrw7/h6/iCSqd5w4yfHps/0Wad5
UJwOsfzCFtJj0GqpVUkKlhOwGBAU3C8vaXMJLM0HnJT1tmCq2CCJUh/osynzZloPMuPjBTC/2o
ykS5GD78hlfKzVqyrAXDcdGzQA/ACx6rp9lGm3eAnxz9sFAaMn/t0uDr70JZaGmqPMX+m/9nRI
A0n3lfQAQGuhor5G6JP0ybUctoreyyjyTkmyhh/CmNIDoCLF+DOYCBbJWMMInLeCbgDPMQn/7Y
9EeBUSoPYGFLrr+EydSSgCSBBm+mtJzFtbMU3Py29Xj0N0glOs77hwSXtEoGbilhB2ax4LW8qa
3nBdiX3OId19LcgJxEHhSRNoY1kDZpnpgBz/lGH73ClLklapPcEceH/g1k9jUStCK8SGiMUEMW
yrAhbhOz9gMBcFAMyDU2T8ReSz2E+tBmdqPdOvQPuoCIo/L0S1qQq2rKq/cf3LDajKRFXU8Bxo
HKYuabdFXByS9xe4uQgTZjo1U= shah@DESKTOP-R8S8HST"
}
```

Step 3: Create SecurityGroup.tf

```
resource "aws_security_group" "shah-sg" {
  name        = "shah-sg"
  description = "Allow TLS inbound traffic and all outbound traffic"
  tags = {     Name = "shah-sg"           } }
resource "aws_vpc_security_group_ingress_rule" "sshFromAnywhere" {
  security_group_id = aws_security_group.shah-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
```

```
  to_port           = 22      }
resource "aws_vpc_security_group_ingress_rule" "allowFrom_http" {
  security_group_id = aws_security_group.shah-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 80
  ip_protocol       = "tcp"
  to_port           = 80      }
resource "aws_vpc_security_group_ingress_rule" "allowFrom_https" {
  security_group_id = aws_security_group.shah-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 443
  ip_protocol       = "tcp"
  to_port           = 443          }
resource "aws_vpc_security_group_ingress_rule" "allowFromRangeOfPorts" {
  security_group_id = aws_security_group.shah-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 3000
  ip_protocol       = "tcp"
  to_port           = 9000    }
resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv4" {
  security_group_id = aws_security_group.shah-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1" # semantically equivalent to all ports     }
resource "aws_vpc_security_group_egress_rule" "allow_all_traffic_ipv6" {
  security_group_id = aws_security_group.shah-sg.id
  cidr_ipv6         = "::/0"
  ip_protocol       = "-1" # semantically equivalent to all ports     }
```

Step 3: Instance.tf

```
resource "aws_instance" "muhsin-instance" {
  ami                    = data.aws_ami.amiID.id
  instance_type          = "t3.micro"
  key_name               = aws_key_pair.shah-key.key_name
  vpc_security_group_ids = [aws_security_group.shah-sg.id]
  availability_zone      = "us-east-1a"
  tags = {
    Name    = "MuhsinShahInstance"
    project = "Shah_project"
  }
}
```

Step 3: output.tf

```
output "instance_ip" {
  value = aws_instance.muhsin-instance.public_ip
}
```

Step 3: Initialize & Apply

Run these commands one by one

```
terraform init
terraform plan
terraform apply
```

Terraform will ask Do you want to perform these actions?, type **yes**.

Now check AWS Console → EC2 → Your instance is running 🎉.

Step 4: Destroy (cleanup)

```
terraform destroy
```

---

7. Terraform Variables (reusability)

Define in variables.tf:

```
variable "instance_type" {
  default = "t2.micro"
}
Use in main.tf:
resource "aws_instance" "my_ec2" {
  ami           = "ami-08c40ec9ead489470"
  instance_type = var.instance_type
}
```

---

9. Terraform State

What is terraform.tfstate?

- It's a JSON file that Terraform creates automatically.
- Purpose: keeps track of the real-world infrastructure Terraform manages.
- Think of it like Terraform's memory or "source of truth."

When you run terraform apply, Terraform:

1. Reads your .tf configuration (what you want).
2. Reads the .tfstate file (what exists).
3. Talks to the cloud provider (AWS, etc.) to confirm actual state.
4. Figures out the difference and applies changes.

---

2. Why do we need it?

- Without it, Terraform wouldn't know if a resource already exists.
- Example: You define 1 EC2 instance.

    - First run → creates EC2 and saves info in terraform.tfstate.
    - Second run → checks state file, sees EC2 already exists → does nothing.
- Without the state file, Terraform might try to recreate everything every time.

---

4. Key Points about State

- Sensitive: It may store secrets (like DB passwords). Don't share it casually.
- Local by default: terraform.tfstate is created in your project folder.
- Remote state (best practice for teams): store in S3 + DynamoDB for locking.
- Locking: prevents two people from running terraform apply at the same time (avoids conflicts).

---

5. Useful Commands

- terraform show → Pretty prints the state file.
- terraform state list → Lists all resources in state.
- terraform state show <resource> → Shows details of one resource.

👉 **Quick analogy:**
Imagine you're the manager of a hotel.

- Your .tf files are the blueprints (what you want).
- The AWS cloud is the actual hotel.
- The .tfstate file is your guestbook — it tells you who's in which room.

Without the guestbook, you'd have to run around the whole hotel every time to figure out what's already there.

---

10. Terraform Modules (advanced, later)

Modules = reusable blocks of infra. Example: you could create a "VPC module" and reuse it across projects.