# Terraform Architecture:

# Key Components and Real-Time AWS Deployment

Terraform is an Infrastructure as Code (IaC) tool that allows you to define, provision, and manage infrastructure through declarative configuration files. Here's an architectural flow of how Terraform works along with the components involved and a real-time example to illustrate its use.

## Terraform Architectural Flow

1. **Write Configuration:** Define your infrastructure in configuration files using the HashiCorp Configuration Language (HCL) or JSON. These files describe the resources you want to create and manage.

2. **Initialize:** Run `terraform init` to initialize your configuration. This command downloads the necessary provider plugins (e.g., AWS, Azure, Google Cloud).

3. **Plan:** Run `terraform plan` to create an execution plan. This plan shows what actions Terraform will take to achieve the desired state described in your configuration files.

4. **Apply:** Run `terraform apply` to execute the plan and create or modify your infrastructure to match the desired state.

5. **Destroy:** Run `terraform destroy` to remove all infrastructure managed by your Terraform configurations.

# Key Components:

1. **Configuration Files:**
    - These are the `.tf` files where you define your desired infrastructure state.
2. **Providers:**
    - Providers are plugins that interact with APIs of different cloud platforms and services (e.g., AWS, Azure, GCP).
3. **Resources:**
    - Resources represent the components of your infrastructure (e.g., virtual machines, storage buckets).
4. **State File:**
    - Terraform maintains a state file (`terraform.tfstate`) that maps your configuration to the real-world resources. This file is critical for tracking resource changes.
5. **Modules:**
    - Modules are reusable configurations that can be shared and composed together to create more complex infrastructure.

# Real-Time Example

Let's walk through a real-time example of deploying a web server on AWS using Terraform.

*Step 1: Write Configuration*
Create a directory for your Terraform configuration files and add the following files:

'main.tf'

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"

  tags = {
    Name = "web-server"
  }
}
```

**‘variables.tf’**

```
variable "aws_region" {
  description = "The AWS region to create resources in"
  default     = "us-west-2"
}
```

**‘outputs.tf’**

```
output "instance_id" {
  description = "The ID of the web server instance"
  value       = aws_instance.web.id
}

output "instance_public_ip" {
  description = "The public IP of the web server instance"
  value       = aws_instance.web.public_ip
}
```

## *Step 2: Initialize*

Run ‘**`terraform init`**’ to initialize your configuration. This will download the AWS provider plugin.

```
terraform init
```

## *Step 3: Plan*

Run ‘**`terraform plan`**’ to see what actions Terraform will take to create your web server.

```
terraform plan
```

### *Step 4: Apply*

Run **`terraform apply`** to create the web server instance.

```
terraform apply
```

You will be prompted to confirm the action. Type '**`yes`**' to proceed.

### *Step 5: Verify*

After the apply command completes, Terraform will output the instance ID and public IP of your new web server.

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

instance_id = "i-0123456789abcdef0"
instance_public_ip = "54.123.45.67"
```

You can now access your web server using the public IP address.

### *Step 6: Destroy*

When you no longer need the resources, you can run '**`terraform destroy`**' to remove all the infrastructure created by your configuration.

```
terraform destroy
```

Again, you will be prompted to confirm the action. Type **`yes`** to proceed

This example demonstrated how to use Terraform to deploy a web server on AWS. The key components involved were:

1. **Configuration Files (`main.tf`, `variables.tf`, `outputs.tf`)**: Define the desired state of your infrastructure.

2. **Provider (AWS)**: Interacts with AWS APIs to provision resources.
3. **Resource (AWS Instance)**: Represents the EC2 instance to be created.
4. **State File**: Tracks the current state of your infrastructure.

By following these steps, you can use Terraform to manage infrastructure as code, making your infrastructure deployments more consistent, repeatable, and version-controlled.