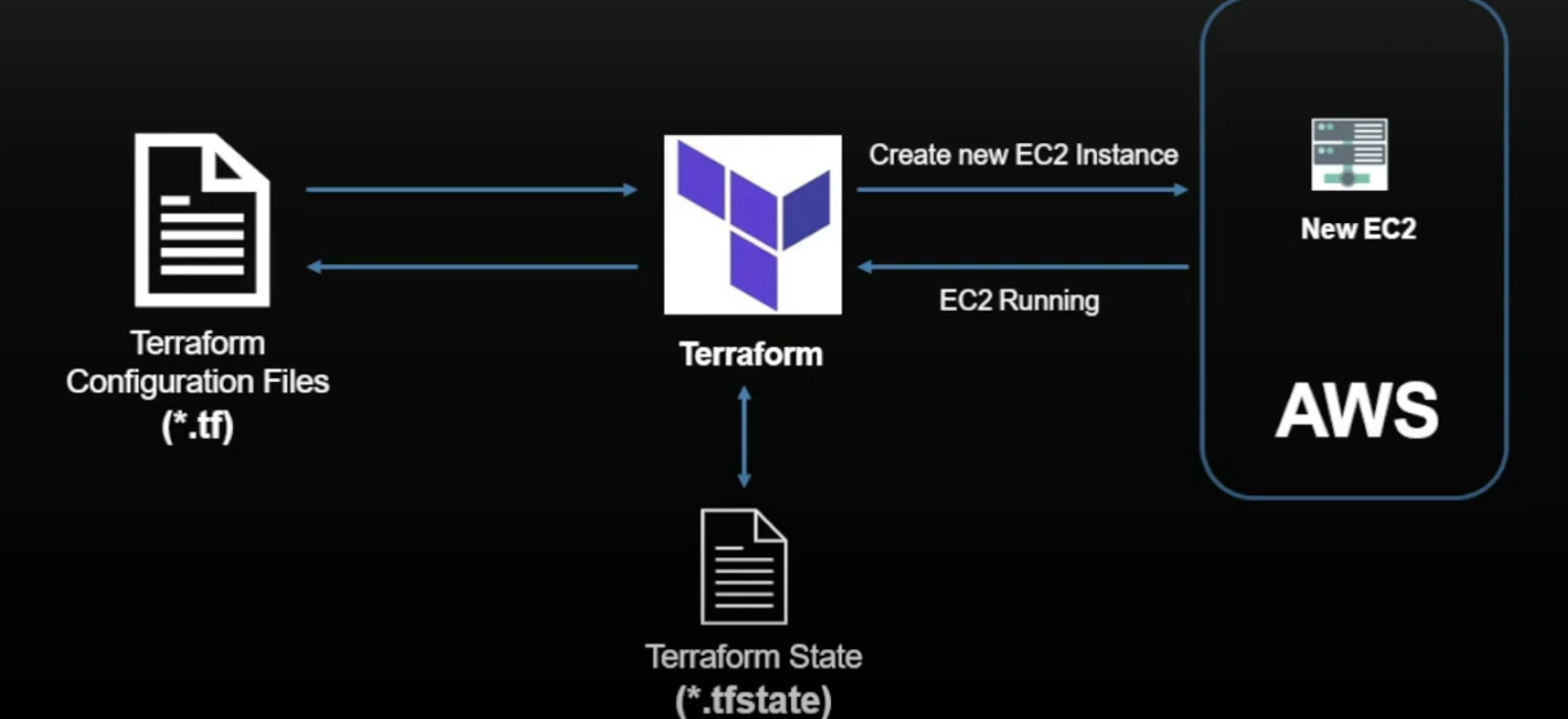
**Terraform Architecture:**



**Terraform Workflow :**

terraform  commands

write configurations files

create .tf files (main.tf variable.tf )

terraform init --> initialize first it will install or download the prviders aws ,gcp and azure with the latest terraform and provider versions and plugins and modules .terraform folder

terraform fmt -- >when you run terraform fmt it is arrange proper format of the code or it will set format of terraform code

terraform init –upgrade 🡪 >= 4.0 now your using ,Tommorow 5.1 is came that time you can use upgrade command or change the version >=5.0

terraform validate --> it will check the syntax and consistency

terraform plan --> show the changes before applying

terraform apply --> apply changes it will ask yes

terraform apply --auto-approve --> it will not ask yes directly apply changes

terraform destroy 🡪destroy the changes

**How to declare the variables**

.tfvars and auto.tfvars\

Variable definition (.tfvars) files:

Terraform also automatically load a number of variables definition of file if they are present:

File named exactly terraform.tfvars or terraform.tfvars.json

Any files with names ending in .auto.tfvars or auto.tfvars.json

Specifty the file in command line

Terraform apply -var-file=”dev.tfvars” 🡪filename is different execute these –var-file=”dev.tfvars” or “prod.tfvars”

Dev.auto.tfvars

Sit.auto.tfvars

Prod.auto.tfvars

No need to execute -var-file in command line below like these

**Terraform State file:**

Every time you run terraform plan or terraform apply, Terraform finds the resources it created previously and updates them accordingly.

Terraform knows which resources to manage by using the Terraform state file.

By default, Terraform creates a file called terraform.tfstate.This file is in JSON format and records a mapping between the resources in your configuration files and the actual infrastructure in the real world.

When you modify your code and apply it, Terraform checks the state file, compares the changes, and updates the infrastructure as needed.

**Individual project for single person to handle :**

For personal projects, storing state in a single local terraform.tfstate file works fine.

**For team projects, this approach causes problems such as:**

Shared storage – all team members need access to the same state file, which requires a shared location.

Locking – without state locking, if two people run Terraform at the same time, it may cause race conditions, conflicts, or corrupted state files.

Isolation – separating environments (like dev, test, staging, and prod) is difficult if all resources share the same state file.

terraform {

  required\_version = ">= 1.3.0"

  required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 5.0"

    }

  }

#Shared storage for terraform state files

  backend "s3" {

    bucket         = "terraform-state-bucketsai-fresh"  #Get the state file from these here and update the state file from these here

    key            = "dev/terraform.tfstate"

    region         = "ap-south-1"

    dynamodb\_table = "terraform-state-lock"  #this for state file locking

    profile        = "terraformprofile"

  }

}

# AWS Provider Configuration

provider "aws" {

  region  = "ap-south-1"

  profile = "terraformprofile"

}

#Terraform block Configuration like terraform version and

provider version

terraform {

#  required\_version = ">=1.3.0" #terraform Version

  required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 5.0" #provider Version like aws/azure/Gcp providers or if you don’t specify the version it will download latest version FYI

    }

  }

}

# Configure the AWS Provider

provider "aws" {

  region = "ap-south-1"

#  profile = "terraformprofile" #pass the profile name

# it you want pass the aws IAM role you can pass like these

/\*

   assume\_role {

    role\_arn     = "arn:aws:iam::123456789012:role/ROLE\_NAME"

}

\*/

#this one way to pass to the credentials aws iam user credentials  access\_key/secret\_key

/\*

  access\_key = "AKIA3XVUMBQA3F4KSXOP"

  secret\_key = "bhXioNO27Q2ufPTEq7q9XyGze5LnOfGvNe8Vn1U0"

\*/

#Another way of aws IAM user credentials using Environment Varibale

/\* In the linux terminal you canuser export below commands

export AWS\_ACCESS\_KEY\_ID="AKIA3XVUMBQA3F4KSXOP"

export AWS\_SECRET\_ACCESS\_KEY="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"

export AWS\_REGION="us-west-2"

#if you aws credentials in different path use below syntax

provider "aws" {

shared\_config\_files = ["/Users/tf\_user/.aws/conf"]

shared\_credentials\_files = ["/Users/tf\_user/.aws/creds"]

profile = "customprofile"

}

Dev :

Folder : Dev

Main.tf

Resource.tf

Varibales.tf

Dev.auto.tfvars

Bucket main tf.state

QA

Bucket tf.state

Terraform module

Provider

Res