**Study Note: Launch an EC2 Instance using Terraform**

In the realm of infrastructure as code (IaC), Terraform has emerged as a powerful tool for automating the provisioning and management of cloud resources.

This is my study note; which describes Terraform basic terms, and steps for launch an EC2 Instance with a ssh keypair using terraform code.

**1. Create a working directory**

Create a folder for writing the code, and open it in Visual studio code IDE. I have configured the AWS CLI, HashiCorp terraform and HCL (HashiCorp Configuration Language) extensions in VS code.

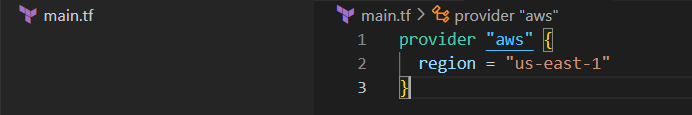
**2. Define provider in main.tf file**

**A provider in Terraform is a plugin that enables interaction with an API.**

This block of **provider** configuration in Terraform that specifies which cloud provider to use. The AWS provider allows Terraform to create, modify, and manage AWS services.

We can also create a separate **provider.tf** file for the provider configurations.

***Terraform Doc. link for AWS provider template:*** [***https://registry.terraform.io/providers/hashicorp/aws/latest/docs***](https://registry.terraform.io/providers/hashicorp/aws/latest/docs)

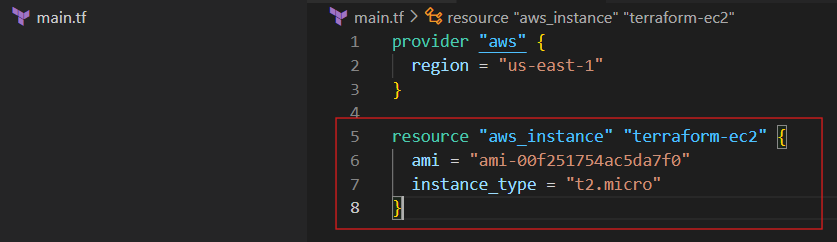


* provider "aws”: This declares that you are using the **AWS** provider for your Terraform configuration.
* region = "us-east-1”: This sets the **region** for the AWS services. All resources managed by this provider will be created in the us-east-1.

**3. Define an AWS EC2 instance Terraform block by hardcoding the values in main.tf**

A **main.tf** file in Terraform typically contains the core configuration for defining resources, variables, providers, and any other infrastructure-related settings.

***Terraform Doc. link for AWS EC2-Instance template:*** [***https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance***](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance)



* resource "aws\_instance" "terraform-ec2”:
* Defines the resource of type **aws\_instance** (i.e., an EC2 instance).
* **terraform-ec2** is the name of this resource within the Terraform configuration
* ami = "ami-00f251754ac5da7f0":
* Specifies the **Amazon Machine Image (AMI)** ID that the EC2 instance will use.
* The AMI ID is region-specific, so make sure it matches the region where you want to launch the instance.

**4. Parameterize the hardcoded values in Resource block using Terraform Variables**

**Variables**

Input and output variables in Terraform are essential for parameterizing and sharing values within your Terraform configurations and modules.

They allow you to make your configurations more dynamic, reusable, and flexible.

Variables let you input different values without modifying the actual configuration file, making it easier to manage configurations across different environments (e.g., development, production).

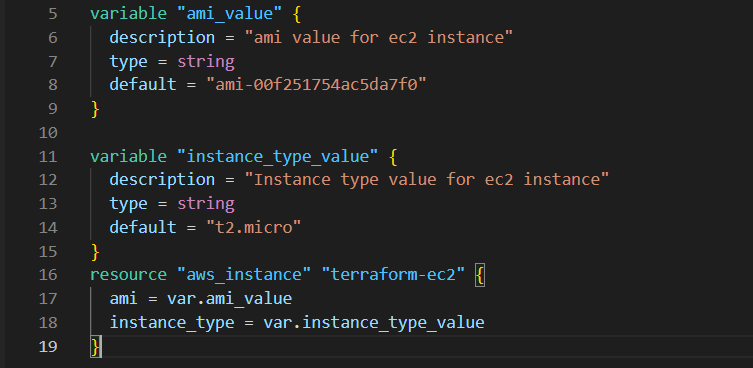
**Input Variables**

Input variables are used to parameterize your Terraform configurations. They allow you to pass values into your modules or configurations from the outside.

Input variables can be defined within a module or at the root level of your configuration.

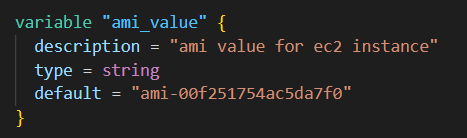
***Terraform Doc. link for AWS Input variable template:*** [***https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-variables***](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-variables)

Here, we have to parameterize the **ami value** and **instance\_type value** as separate variables in above EC2 Instance resource block.

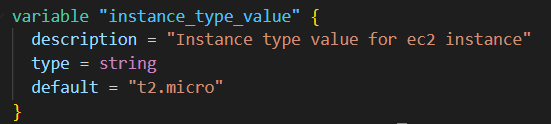


**Variable Definition:**

* **ami\_value**

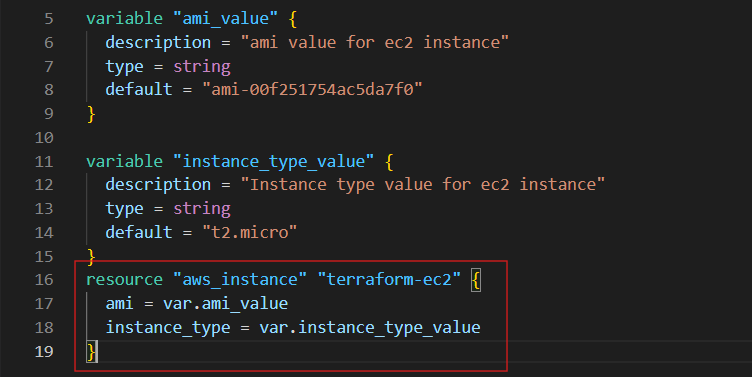


* **variable** is used to declare an input variable named **ami\_value**.
* **description** provides a human-readable description of the variable.
* **type** specifies the data type of the variable, string. (e.g., string, number, list, map, etc.).
* **default** provides a default value for the variable.
* **instance\_type\_value**



* This defines a variable named **instance\_type\_value** for specifying the **instance type** of the EC2 instance.
* The default value is "t2.micro".

**Using the Variables in the Resource Block**



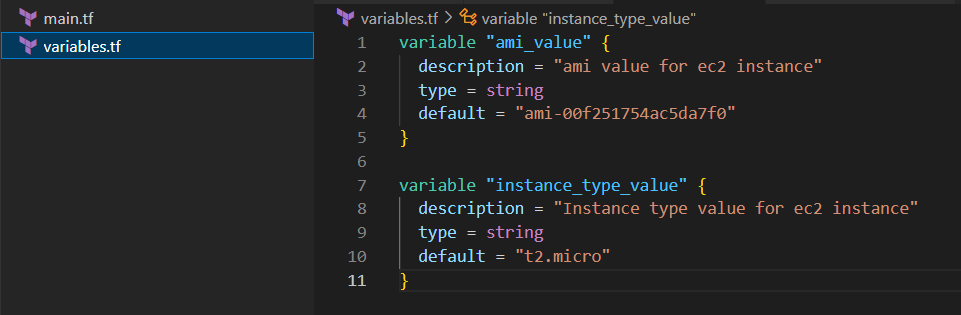
* **The aws\_instance resource is configured to use these variables**
* **ami = var.ami\_value**

This sets the **ami** attribute of the aws\_instance to the value of **var.ami\_value**.

* **instance\_type = var.instance\_type\_value**

This sets the instance\_type attribute of the aws\_instance to the value of var.instance\_type\_value.

Then, we have created a separate **variables.tf** file and move the input variables block from main.tf



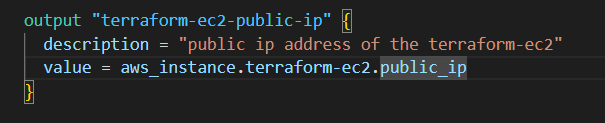
**Output Variables**

Output variables in Terraform are used to extract and display information about resources created in your infrastructure.

They allow you to retrieve values after the execution of a Terraform plan, making it easy to access important data, such as resource IDs or IP addresses.

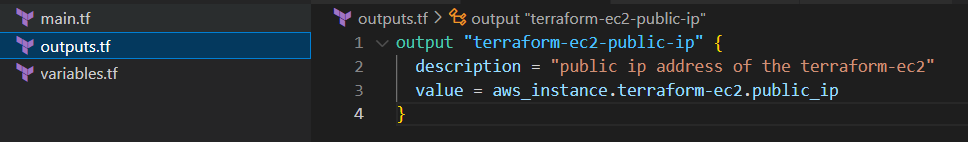
***Terraform Doc. link for AWS Output variable template:*** [***https://developer.hashicorp.com/terraform/language/values/outputs***](https://developer.hashicorp.com/terraform/language/values/outputs)

Here, we have to add an Output variable block for display the public IP address of the instance (terraform-ec2) after terraform apply.

****

* **output** is used to declare an output variable named **terraform-ec2-public-ip**.
* **description** provides a description of the output variable.
* **value** specifies the value that you want to expose as an output variable.
* This expression accesses the **public\_ip** attribute of the aws\_instance resource named **terraform-ec2**. It retrieves the public IP address assigned to that EC2 instance.

Then, we have created a separate **outputs.tf** file and move the output variable block from main.tf



**5. Define SSH Key pair**

Defining SSH key pairs in Terraform can be done in several ways, depending on your requirements.

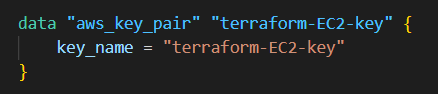
**With existing Key pair**

To use an existing SSH key pair that you have created in the AWS Management Console in your Terraform configuration, you will reference the key pair by its name.

The public key is automatically associated with the EC2 instances that you launch with that key pair.

Here, I have created a key pair “**terraform-EC2-key**” from aws console and download the private key to my desired folder and give the necessary permissions.

Then, edited the main.tf by defining the aws-key-pair with the existing key pair.



* **data**:

This data block is used to retrieve information about an existing AWS key pair named "terraform-EC2-key".

**In Terraform, a data block lets you access resources that already exist outside of Terraform’s management, allowing you to retrieve their information without creating or modifying them.**

* **"aws\_key\_pair"**:

This specifies the type of data source you are querying. Here, it indicates that you want to retrieve information about an AWS EC2 key pair.

* **“terraform-EC2-key”:**

The local name you assign to this specific data source.

* **key\_name**:

This argument specifies the name of the existing key pair that you want to reference.

* **terraform-EC2-key:**

Actual name of the key pair that you created earlier in the AWS Management Console.

**By Generating Keypair**

**Generate using tls\_private\_key**

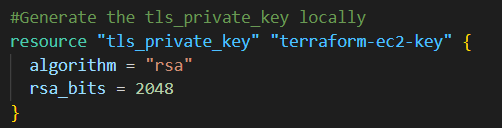
Firstly, we are defining an AWS key pair using Terraform after generating the SSH keypair with tls\_provider.

The TLS provider provides utilities for working with **Transport Layer Security/*****Secure Sockets Layer*, or SSL. TLS** keys and certificates. It provides resources that allow private keys, certificates and certificate requests to be created as part of a Terraform deployment.

This allows you to dynamically create or reference a private key within your Terraform configuration, making it useful for automation, especially when creating secure SSH keys without managing key files manually on disk.

Steps;

**1. Generates an RSA private key locally within Terraform.**



***Terraform Doc. link for AWS tls\_private\_key:*** [***https://developer.hashicorp.com/terraform/language/values/outputs***](https://developer.hashicorp.com/terraform/language/values/outputs)

* **resource "tls\_private\_key" "terraform-ec2-key"**:

Declares a Terraform resource using the tls\_private\_key resource type.

The identifier "terraform-ec2-key" is the name of this resource instance within your Terraform configuration. This name can be used to reference the key later in your configuration.

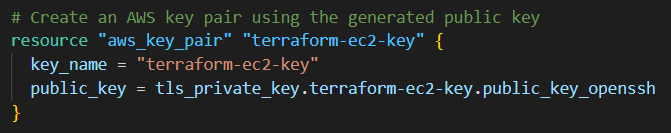
* **algorithm = "rsa"**:

Specifies that the private key generated should use the RSA algorithm.

* **rsa\_bits = 2048**:

Specifies the bit length of the RSA key to be generated,

**2. Create an AWS key pair using the generated public key**

****

* **resource "aws\_key\_pair" "terraform-ec2-key":**

This declares an aws\_key\_pair resource, which creates an SSH key pair in AWS.

"terraform-ec2-key" is the resource name in your Terraform configuration.

* **key\_name = "terraform-ec2-key":**

This is the name of the key pair as it will appear in AWS.

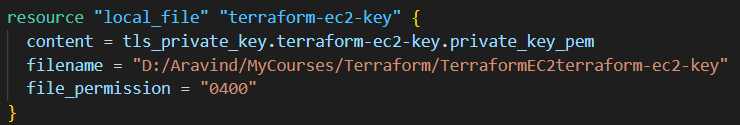
* **public\_key = tls\_private\_key.terraform-ec2-key.public\_key\_openssh**:

This references the public\_key\_openssh attribute of the tls\_private\_key resource defined earlier.

tls\_private\_key.terraform-ec2-key refers to the tls\_private\_key resource named terraform-ec2-key in your configuration.

.public\_key\_openssh retrieves the generated public key in OpenSSH format, which is required for AWS key pairs.

**3. Save the private key locally for SSH access**

****

* **resource "local\_file" "terraform-ec2-key"**:

This defines a local\_file resource in Terraform, which allows you to create a file on your local filesystem as part of the provisioning process.

"terraform-ec2-key" is the name of this resource

* **content = tls\_private\_key.terraform-ec2-key.private\_key\_pem:**

This sets the file content to the private key in PEM format, generated by the tls\_private\_key resource.

tls\_private\_key.terraform-ec2-key.private\_key\_pem extracts the

privatekey from the tls\_private\_key resource named terraform-ec2-key.

* **filename ="D:/Aravind/MyCourses/Terraform/TerraformEC2/terraform-ec2-key"**:

Specifies the location and name of the file to be created on your local machine.

* **file\_permission = "0400"**:

Sets file permissions to **0400**, meaning only the file owner has read access.

**Summary of the Workflow**

1. **Generate a private key with tls\_private\_key.**
2. **Create the AWS Key Pair with aws\_key\_pair, using the public key from tls\_private\_key.**
3. **Store the private key on the local filesystem using the local\_file resource**.

**Generate using ssh-keygen**

**1. Generating the SSH keypair with ssh-keygen.**

We have using the **ssh-keygen -t rsa -b 2048** command to generate the ssh-keypair.

* **ssh-keygen**: This is the command used to generate, manage, and convert authentication keys for SSH (Secure Shell).
* **-t rsa**: This option specifies the type of key to create.
* **-b 2048**: This option specifies the number of bits in the key. -b 2048 means that the key will be 2048 bits long.
* **-t D:\Aravind\MyCourses\Terraform\TerraformEC2\terraform-ec2-key:**

This flag specifies the output file location and name for the SSH key pair,

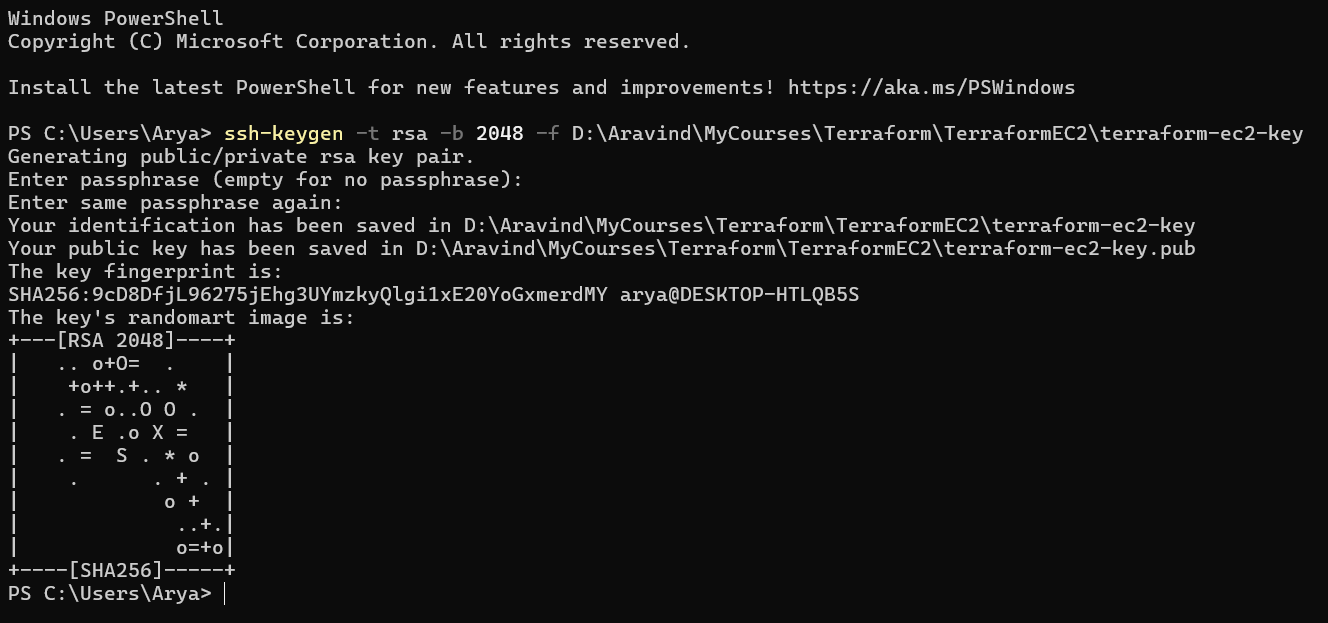
Without it, ssh-keygen will use the default location (~/.ssh/id\_rsa).

It’s generated, public and private key.

**Private key = terraform-ec2-key**

**Public key = terraform-ec2-key.pub**

We need to edit the permission for private key as required;



Now we have to assign the SSH key pair to the EC2 Instance resource block, for that the first step is;

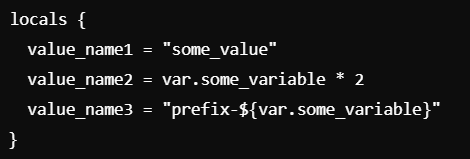
**2. Define the path to your locally generated key**

**Terraform Locals**

In Terraform, the locals block is used to define local values that can be referenced throughout a Terraform configuration.

Locals can be useful for calculations, storing reusable expressions, paths, or any values that might be needed multiple times within the configuration.

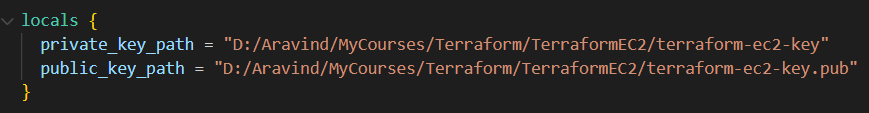
Syntax of locals in Terraform:



Key Points About locals;

* **Scope**: Local values are scoped to the module in which they are defined. This means they can be accessed anywhere within that module, but they are not available outside of it.
* **Flexibility**: They can store strings, numbers, lists, maps, or even more complex expressions.
* **Immutable**: Once a local value is defined, it cannot be changed elsewhere in the module. It is essentially a read-only value.
* **Convenience**: Locals allow you to avoid repeating complex expressions or long strings, making the code easier to read and modify.

The below command defines the path to your locally generated key;



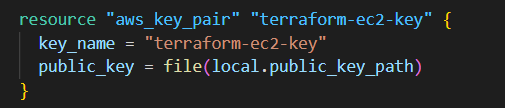
* **locals block**:

The locals block allows you to define variables that can be used only within the current Terraform configuration.

* **private\_key\_path** and **public\_key\_path**:

These local values define the paths to the private and public SSH key files.

**3. Create an AWS key pair using the generated public key**



* **resource "aws\_key\_pair" "terraform-ec2-key"**:

This defines a new aws\_key\_pair resource in Terraform, which will create or import an SSH key pair in AWS.

* **"terraform-ec2-key"** is the name of the resource in Terraform
* **key\_name = "terraform-ec2-key":**

This is the name of the key pair as it will appear in the AWS console.

* **public\_key = file(local.public\_key\_path):**

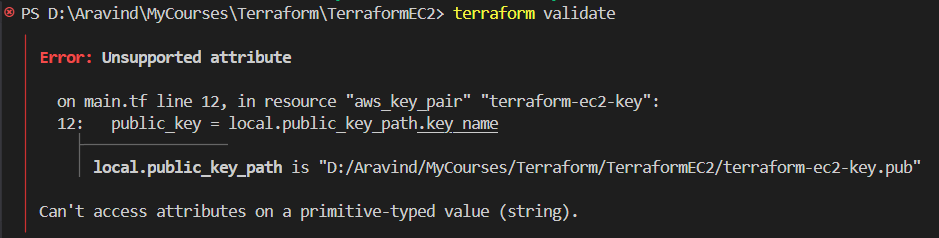
This assigns the public key to the public\_key attribute by reading it from the specified file path.

**file(local.public\_key\_path)** reads the content of the public key file defined in local.public\_key\_path, which was previously set in the local’s block.

**“Fixing errors in mistakes made”**

In my first try, I had done a mistake in defining the public key section. Mentioned the public key as **public\_key = local.public\_key\_path.**

But it gives me an error on terraform validate;



The error is due to how the public\_key is being assigned in the aws\_key\_pair resource. Currently, local.public\_key\_path is defined as a string path to the public key file, but in the code, it’s being accessed as if it were an object with a .key\_name attribute.

To fix this, use the file function to read the contents of the public key file, as the aws\_key\_pair resource expects the actual key content, not the file path.

Then edited it as above mentioned

* The file function reads the contents of the specified file and returns it as a string, which is needed for the public\_key attribute.
* local.public\_key\_path contains the path to the public key file, and file(local. public\_key\_path) retrieves the file’s contents as required by AWS.

Now we have coded the required blocks for define the AWS Instance creation. Including;

* Variables block for ami and instance-type values.
* An output variable for return the public IP of EC2 instance being created.
* Defined the ssh-key-pair (In different types.)

Then, before creates an AWS Instance we have to look into an important section; terraform.tfvars

**6. Replace the hardcoded default values in Input variables by terraform.tfvars**

**terraform.tfvars**

The terraform.tfvars file in Terraform is used to define Input variable values for your Terraform configuration.

It allows you to set the values of variables that you've declared in your Terraform configuration files (usually in a variables.tf file).

They allow you to separate configuration values from your Terraform code, making it easier to manage different configurations for different environments (e.g., development, staging, production) or to store sensitive information without exposing it in your code.

Here's the purpose of `.tfvars` files:

**1. Separation of Configuration from Code:**

Input variables in Terraform are meant to be configurable so that you can use the same code with different sets of values. Instead of hardcoding these values directly into your `.tf` files, you use `.tfvars` files to keep the configuration separate. This makes it easier to maintain and manage configurations for different environments.

**2. Sensitive Information:**

`.tfvars` files are a common place to store sensitive information like API keys, access credentials, or secrets. These sensitive values can be kept outside the version control system, enhancing security and preventing accidental exposure of secrets in your codebase.

**3. Reusability:**

By keeping configuration values in separate `.tfvars` files, you can reuse the same Terraform code with different sets of variables. This is useful for creating infrastructure for different projects or environments using a single set of Terraform modules.

**4. Collaboration:**

When working in a team, each team member can have their own `. tfvars` file to set values specific to their environment or workflow. This avoids conflicts in the codebase when multiple people are working on the same Terraform project.

**Summary**

Here's how you typically use `.tfvars` files

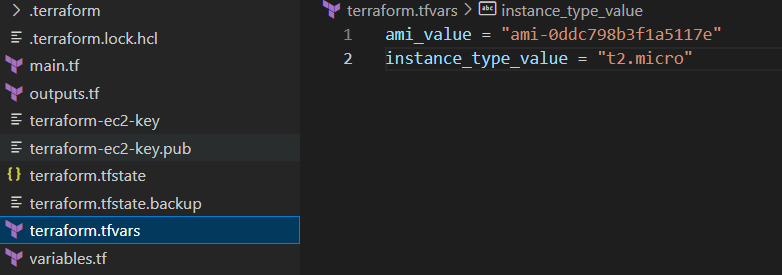
1. Define your input variables in your Terraform code (e.g., in a `variables.tf` file).

2. Create one or more `.tfvars` files, each containing specific values for those input variables.

3. When running Terraform commands (e.g., terraform apply, terraform plan), you can specify which. tfvars file(s) to use with the -var-file option:

terraform apply -var-file=dev.tfvars

Then we have to define the input variables value in terraform.tfvars in our code.



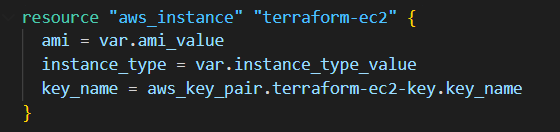
**Key-Value Pairs**: Each line defines a variable and assigns a value to it.

The format is **variable\_name = value.**

**7. Create the EC2 instance and associate it with the key pair**

***Terraform Doc. link for aws\_instance*** [***https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance***](https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance)

Now we can define the EC2 Instance creation;



* **resource "aws\_instance" "terraform-ec2"**

This line declares an aws\_instance resource, which tells Terraform to create an EC2 instance in AWS.

**"terraform-ec2"** is the name given to this specific instance resource in Terraform.

* **ami = var.ami\_value**

The ami parameter specifies the Amazon Machine Image (AMI) ID to use for the instance.

Here, it’s set to **var.ami\_value**, a variable that should be defined in your variables.tf file or provided in terraform.tfvars

* **instance\_type = var.instance\_type\_value**

The instance\_type parameter defines the type of instance to create, such as t2.micro.

This is set to **var.instance\_type\_value**, which should be defined as a variable in the configuration.

* **key\_name = aws\_key\_pair.terraform-ec2-key.key\_name**

The **key\_name** parameter specifies the SSH key pair to be used for accessing the instance.

The **key\_name** attribute specifies the name of an SSH key pair that will be used for accessing the EC2 instance.

**aws\_key\_pair.terraform-ec2-key** refers to a separate aws\_key\_pair resource defined elsewhere in your Terraform configuration. This resource represents the SSH key pair that is registered with AWS.

.**key\_name** is an attribute of the aws\_key\_pair resource that holds the name of the SSH key pair.

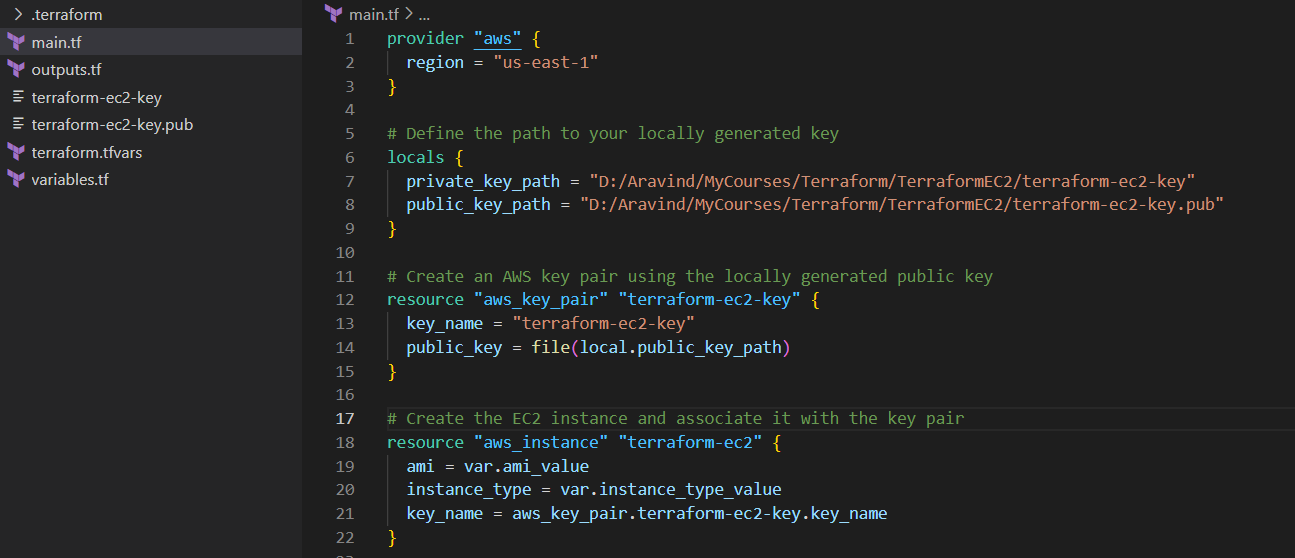
**“Why This is Important”**

* **Automatic Updates**: Using aws\_key\_pair.terraform-ec2-key.key\_name instead of hard-coding a name makes your configuration more flexible. If you later change the name of the key pair in the aws\_key\_pair resource, you don’t need to manually update every instance of that key name in other resources.
* **Consistency**: It ensures that the correct key pair is always used, reducing the risk of configuration mismatches.

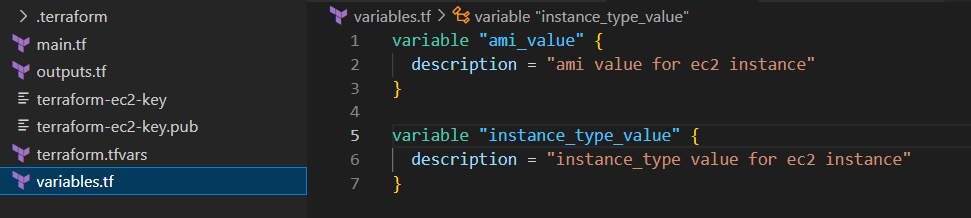
So, in short, .key\_name is a way to dynamically access the name of the SSH key pair from the aws\_key\_pair resource, ensuring that the EC2 instance is associated with the correct SSH key for secure access.

Now the terraform configuration part is completed, and the project directory looks likes below;

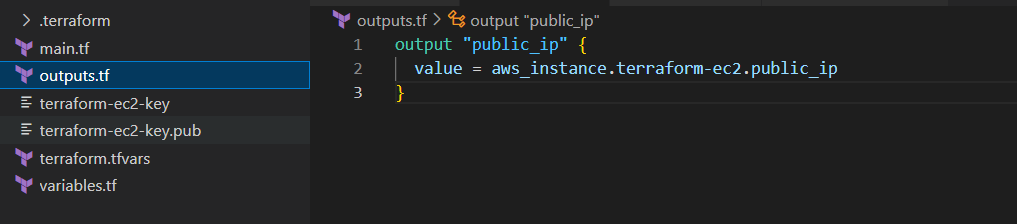
Main.tf



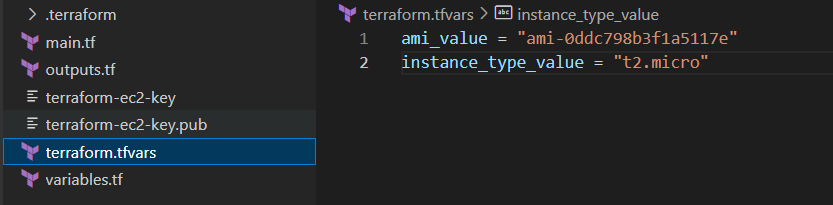
variables.tf



Outputs.tf



Terraform.tfvars



Also, we the public and private ssh keys are created in the same directory.

**8. Run the terraform code**

The Terraform workflow generally follows three main stages: **init**, **plan**, **apply** and **destroy**,

**terraform init:**

It initializes your working directory by preparing the necessary plugins, setting up the backend, and configuring other dependencies required for Terraform to manage your infrastructure.

**Downloads Provider Plugins:** Terraform requires provider plugins to communicate with different cloud providers (like AWS, Azure, GCP) and other services.

**terraform plan:**

**Preview Changes**: It shows you which resources will be created, modified, or destroyed to bring the infrastructure in line with the desired configuration.

**Identify Errors**: It helps you spot potential errors in your configuration without making actual changes.

**Plan Review**: It allows you to review the changes with team members or stakeholders before making any irreversible updates.

**terraform apply:**

**Deploy Changes**: It applies the changes outlined in the configuration files, provisioning new resources, modifying existing ones, or destroying outdated resources as necessary.

**Execute Planned Actions**: After confirming the plan, Terraform will execute the actions in the correct order to meet the desired configuration.

**Display Outputs**: If you’ve defined output values in your configuration, Terraform will display them after applying the changes, providing information such as resource IDs, IP addresses, etc.

**State File Updates**: After applying, the state file (terraform.tfstate) is updated to reflect the new actual state of the infrastructure.

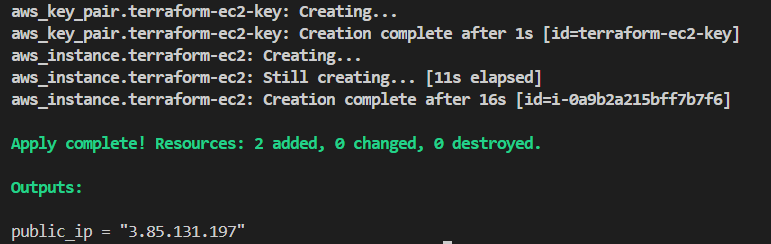
**terraform destroy:**

**Clean Up Resources:** It removes all resources defined in the configuration file.

**Avoid Cost:** It’s particularly useful in cloud environments where resources may incur costs, allowing you to delete infrastructure you no longer need.

**Reset Infrastructure:** It’s helpful for testing purposes, where you might need to tear down and redeploy infrastructure frequently.

After the first three steps; our configuration is deployed in AWS cloud on our configured IAM user profile (aws configure command is used to quickly set up the AWS Command Line Interface (CLI) with your credentials and default settings.)



The resources are created and display the public IP as we are configured the output variable.

**9. Verify the state file**

The .tfstate file (commonly referred to as the "state file") is a critical component of Terraform’s workflow. It’s a JSON file where Terraform stores information about the real-world state of your managed infrastructure.

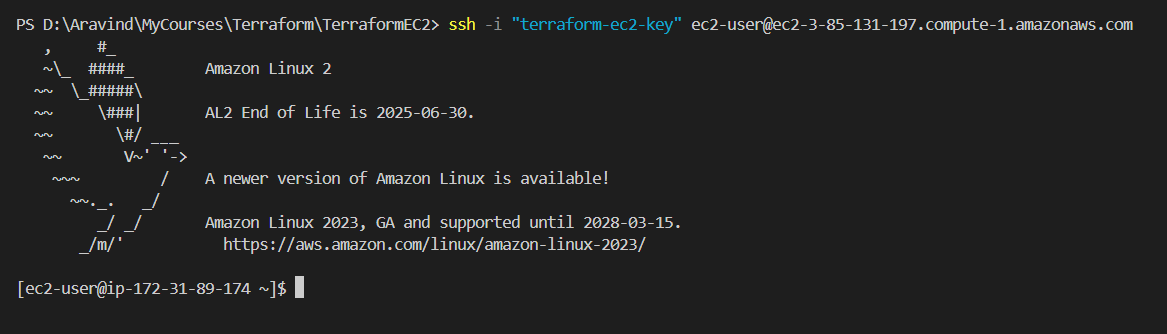
**Track Resource State**: The .tfstate file records the current state of resources, including properties, configurations, and relationships. This helps Terraform understand what has already been created and configured.

**Determine Changes**: By comparing the current .tfstate with the desired state defined in your .tf files, Terraform can determine the specific actions required to bring the actual infrastructure in line with the configuration.

**Store Metadata**: In addition to tracking resources, the .tfstate file includes metadata such as dependencies between resources, which helps Terraform create or update resources in the correct order.

**10. Access the EC2 Instance with SSH command**

We can verify our entire configuration by accessing the instance with ssh command.



**11. Delete the resources**

Run the terraform destroy command for delete the resources which we are created.

--End--