**1) Watch terraform-04 video.**

**2) Execute the script shown in video.**

\* Adding resource as localfile provider

resource "local\_file" "my-pet" {

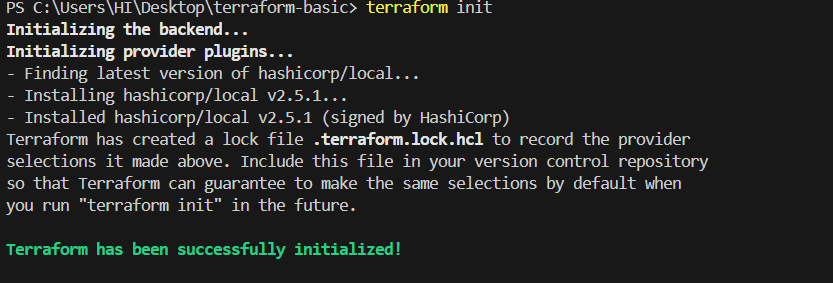
  locafile = "pets.txt"

  content = "my pet name is BB"

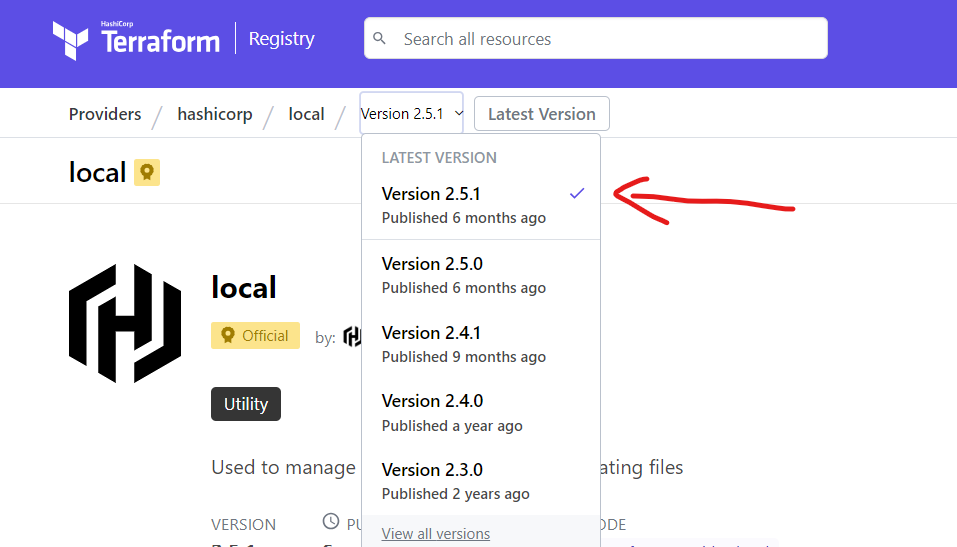
}

terraform init

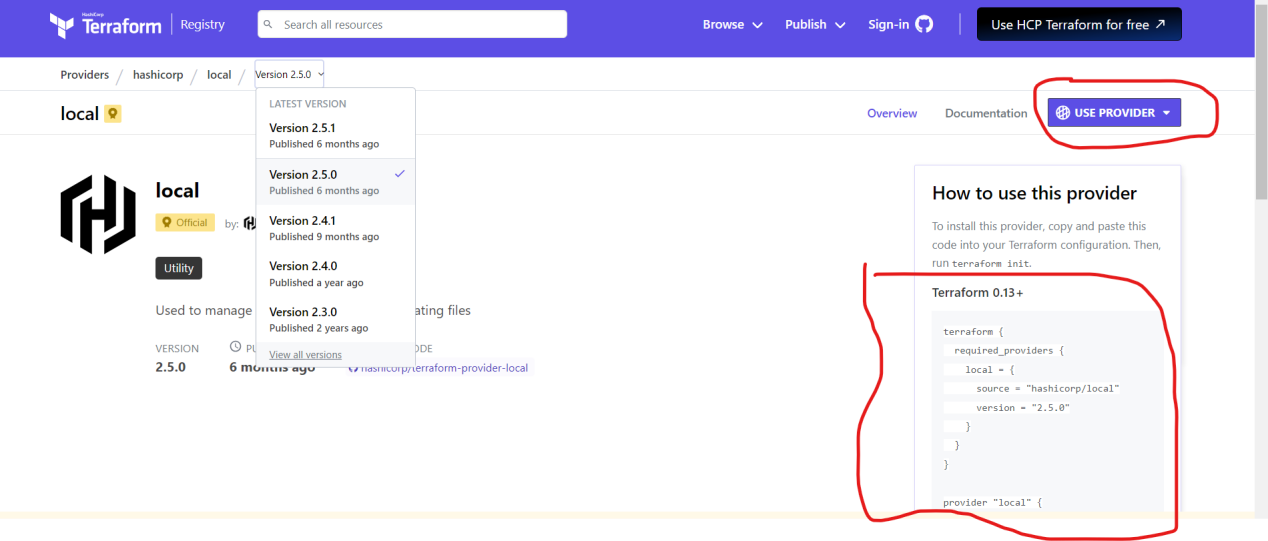
\* the terraform will install the lastest version of provider



\* By default it will checks for the latest version of the provider from **terraform registry**



\* Now I want to use the 2.5.0 version of the plugin , then use this required provider to paste it in main.tf file



\* Before execute this delete previous version

\* Then do terraform init

terraform {

  required\_providers {

    local = {

      source = "hashicorp/local"

      version = "2.5.0"

    }

  }

}

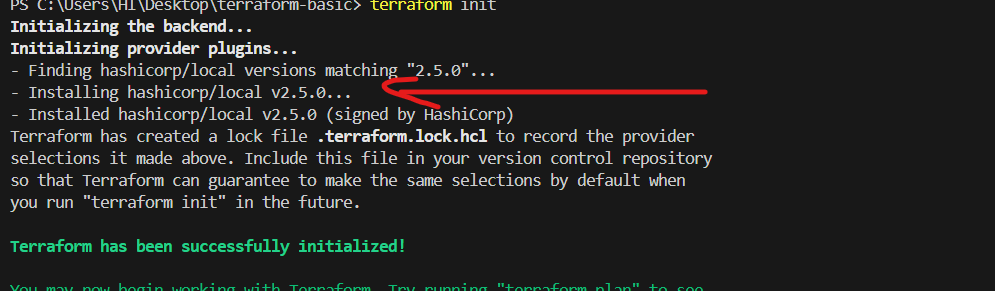
resource "local\_file" "my-pet" {

  locafile = "pets.txt"

  content = "my pet name is BB"

}

\* it will installed 2.5.0 version of plugin



\* Now I want to install 2.4.1 version of plugin without deleting previous version

terraform {

  required\_providers {

    local = {

      source = "hashicorp/local"

      version = "2.4.1"

    }

  }

}

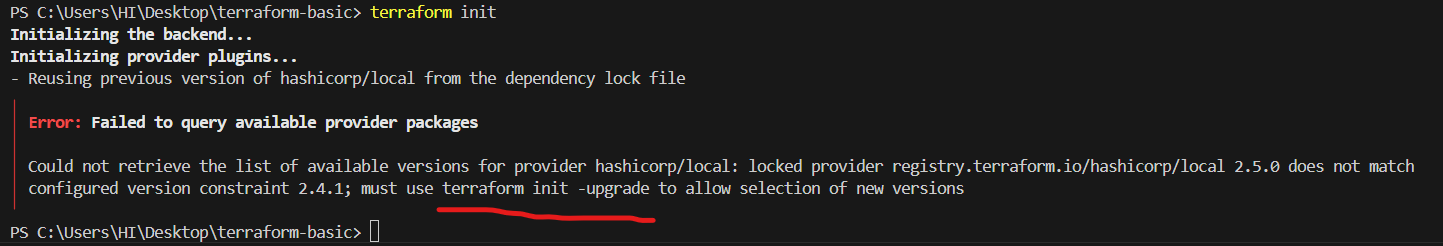
resource "local\_file" "my-pet" {

  locafile = "pets.txt"

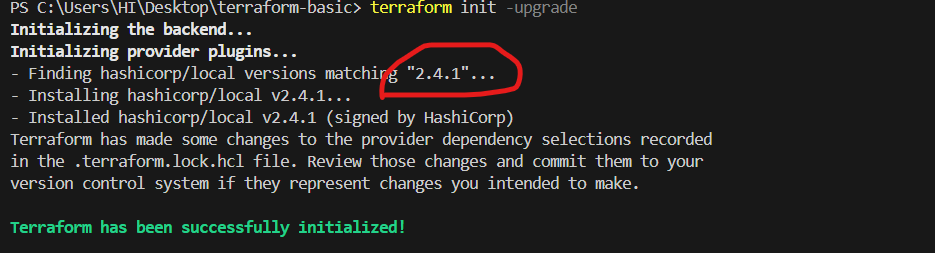
  content = "my pet name is BB"

}

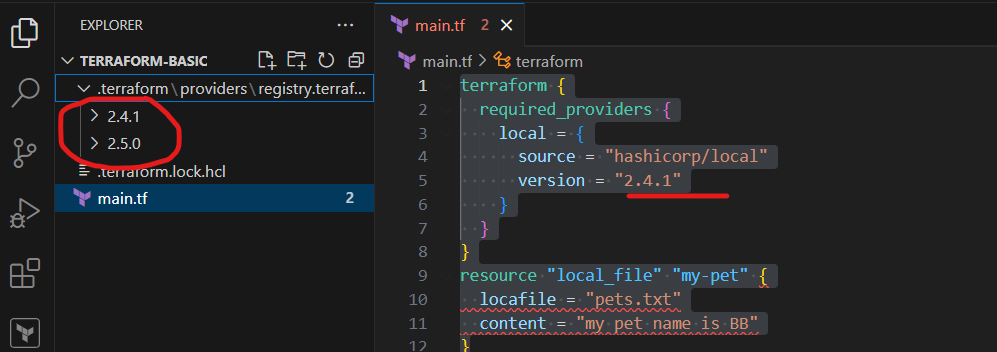
\* While executing **terraform init ,** it will tries to tell upgrade the version using terraform init -upgrade



\* It downloaded the given version



\* here we can see two versions of the plugin



\* Now I want to download the without using this version of 2.4.1

terraform {

  required\_providers {

    local = {

      source = "hashicorp/local"

      version = "!=2.4.1"

    }

  }

}

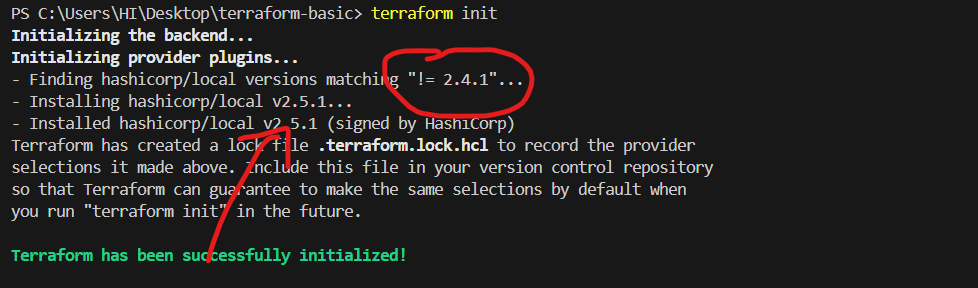
resource "local\_file" "my-pet" {

  locafile = "pets.txt"

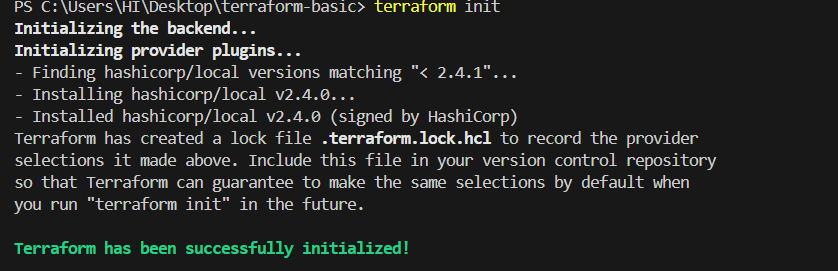
  content = "my pet name is BB"

}

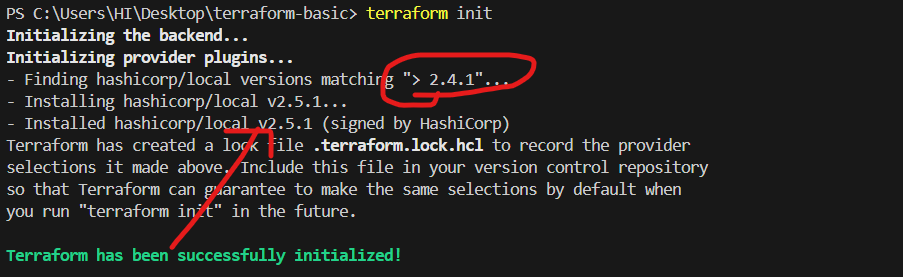
\* Then it will checks and install latest version of plugin



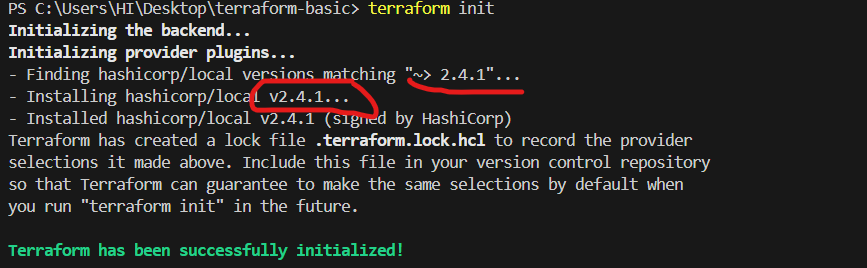
\* Now I want to install lessthan the present version of plugin



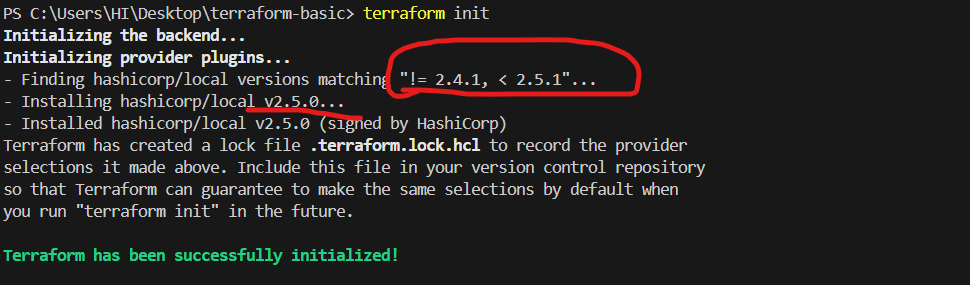
\* Here I want to install above the version to present version of plugin



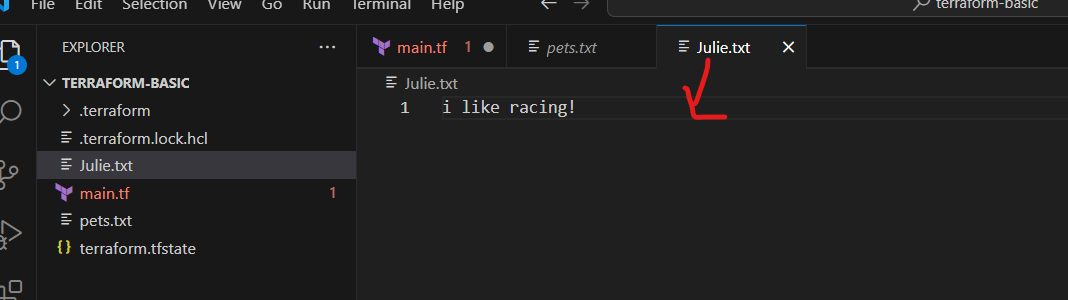
\*Here I want to install specific or higher version of the plugin



\* Now am tries to install the provider plugin between these two versions = "!=2.4.1,<2.5.1"



\* Manually I created one txt file



\* Now am adding data block

\* which is used to read the content of this file only by using resource block (the pets.txt content shuld be the content of julie.txt)

resource "local\_file" "my-pet" {

  filename = "pets.txt"

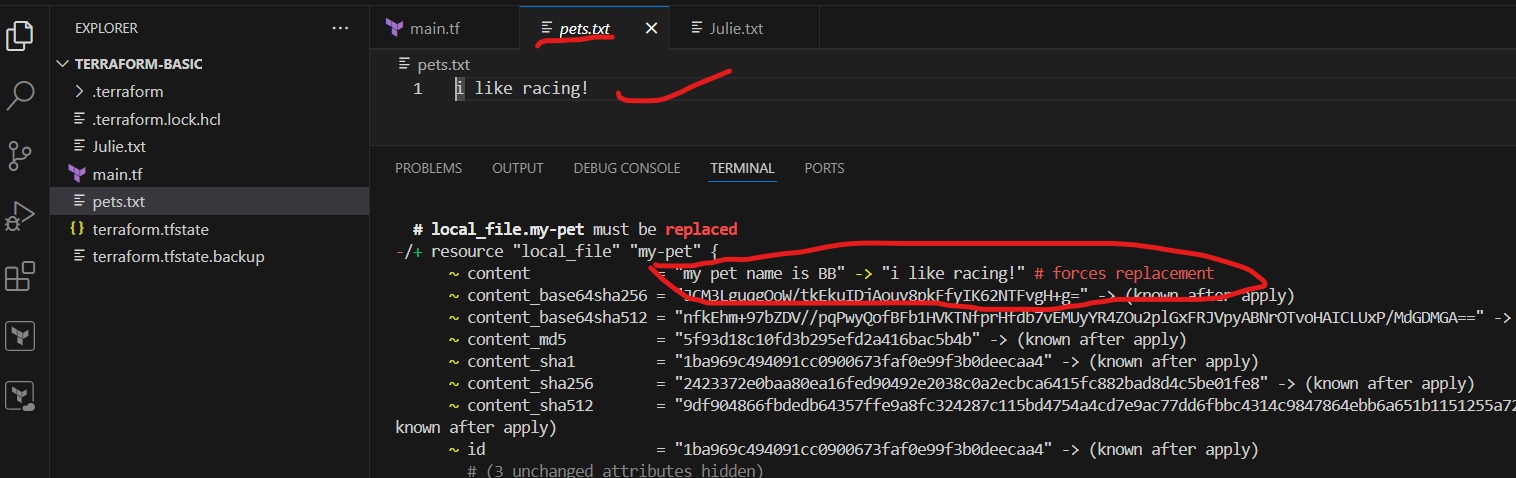
  content = data.local\_file.Julie.content

}

data "local\_file" "Julie" {

  filename = "julie.txt"

}



* **I need to create 3 files using META ARGUMENTS**

**\*** Now am adding the count value in resource block to create 3 files

resource "local\_file" "my-pet" {

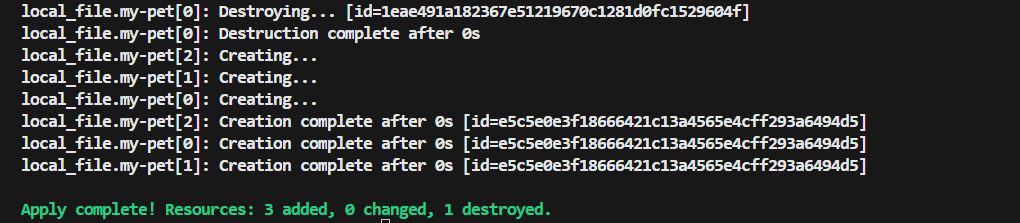
  filename = "pets.txt"

  content = "i like driving"

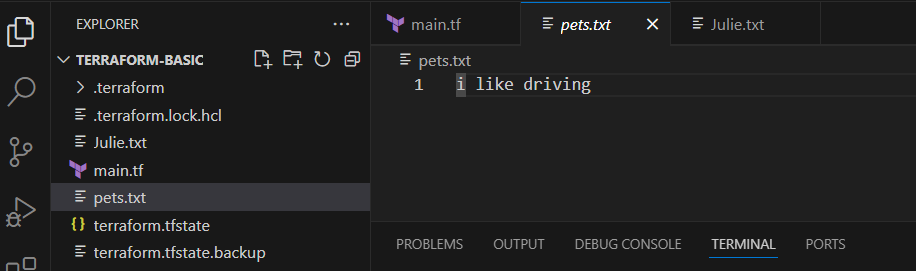
  count = 3

}

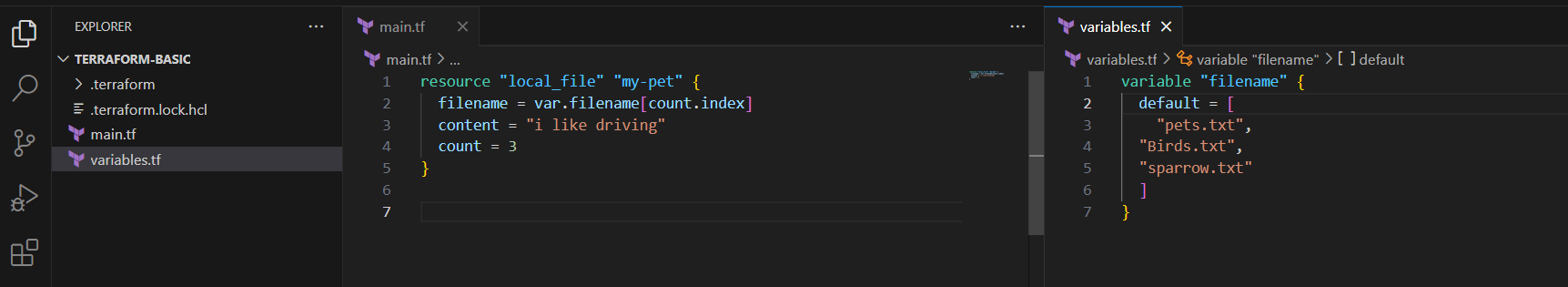
\* while am executing , it will create 3 files but it will override of these files



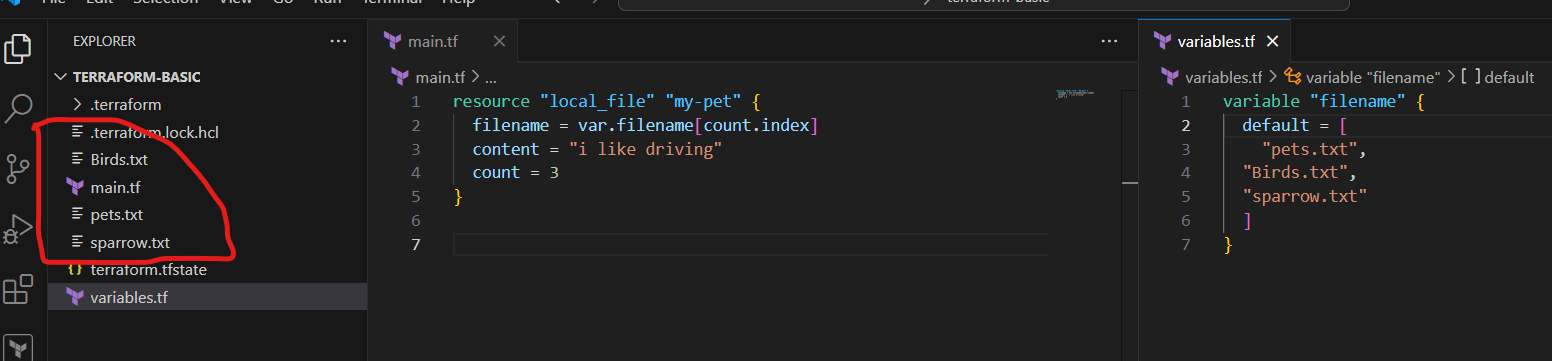
\* only single file should be created and added content in this file



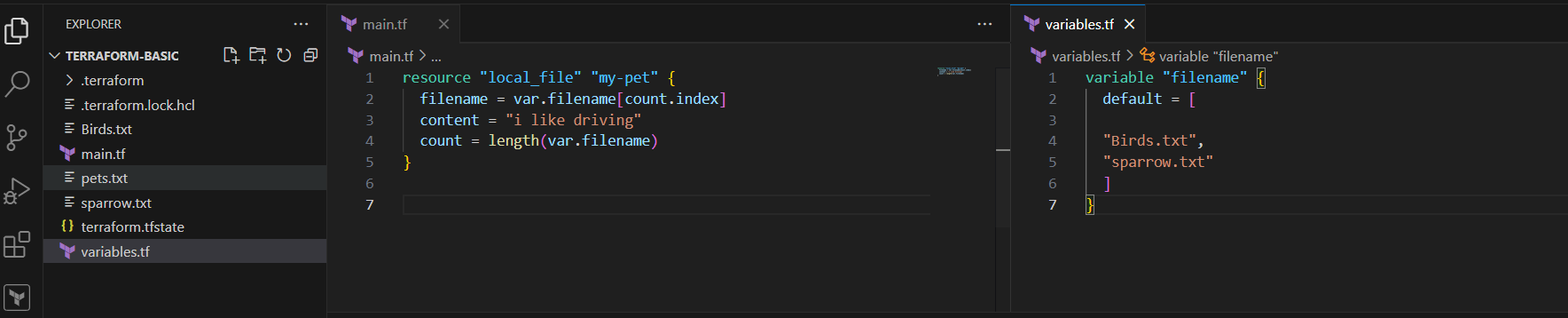
\* here am using variable to and index to creating the files,



\* Now 3 files has been created

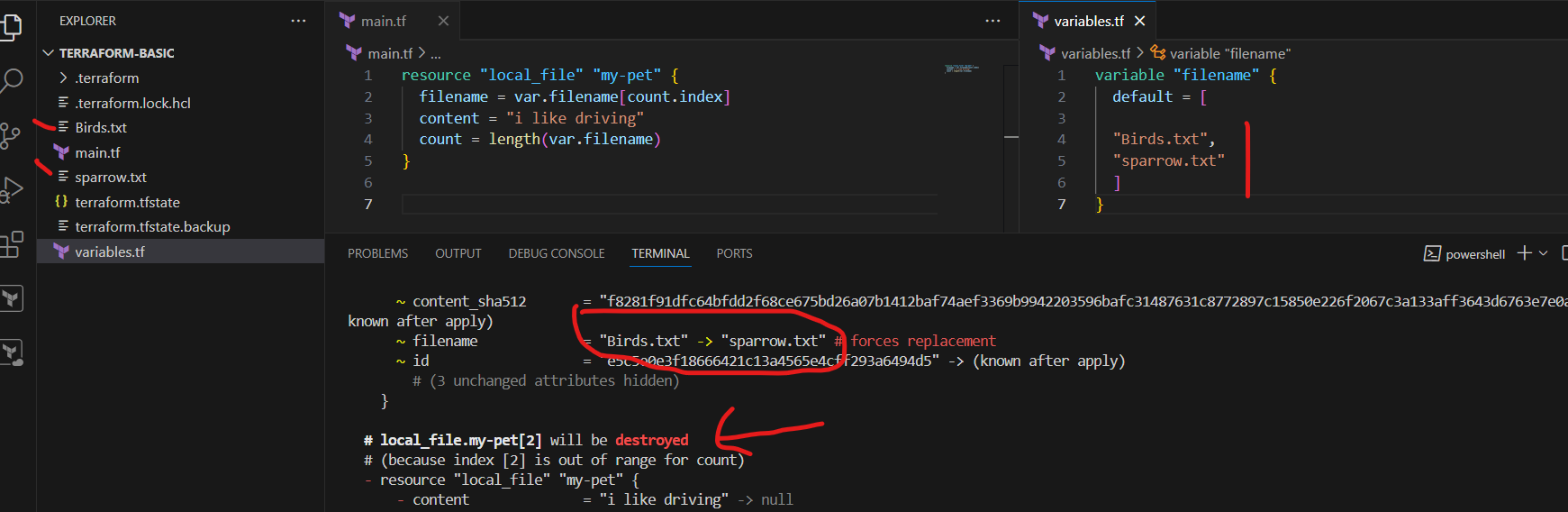


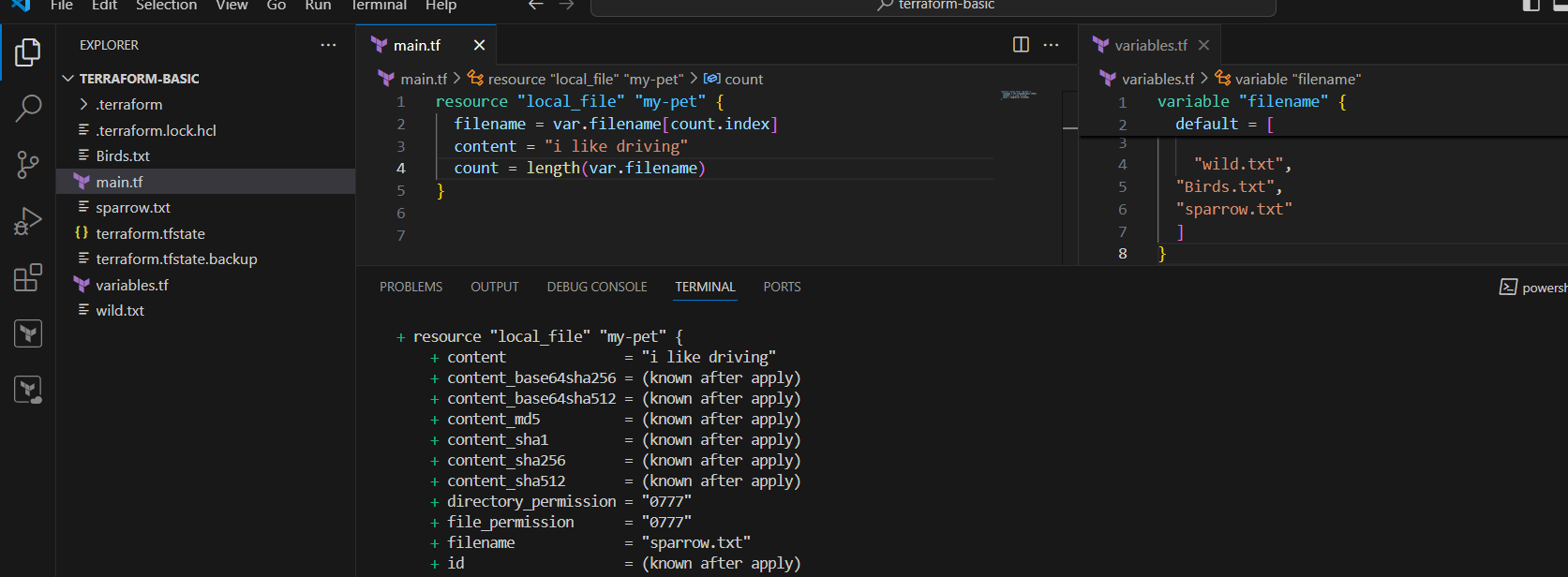
\* But any files removed in the given variable then manually we can edit the count



\* By using **length** of filename , we can create what ever files we have added in **variables**

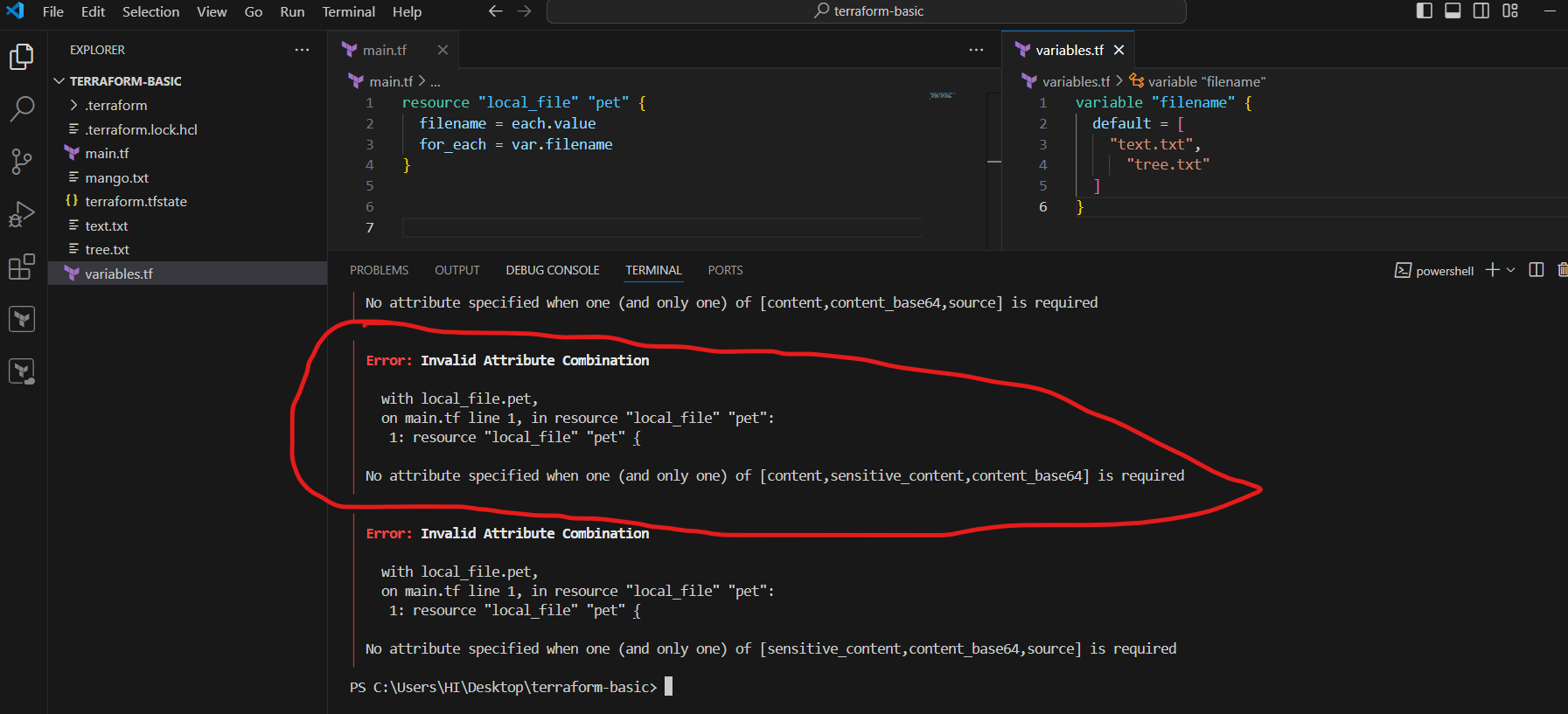
\* and it will delete previous files which are not available in variable filename



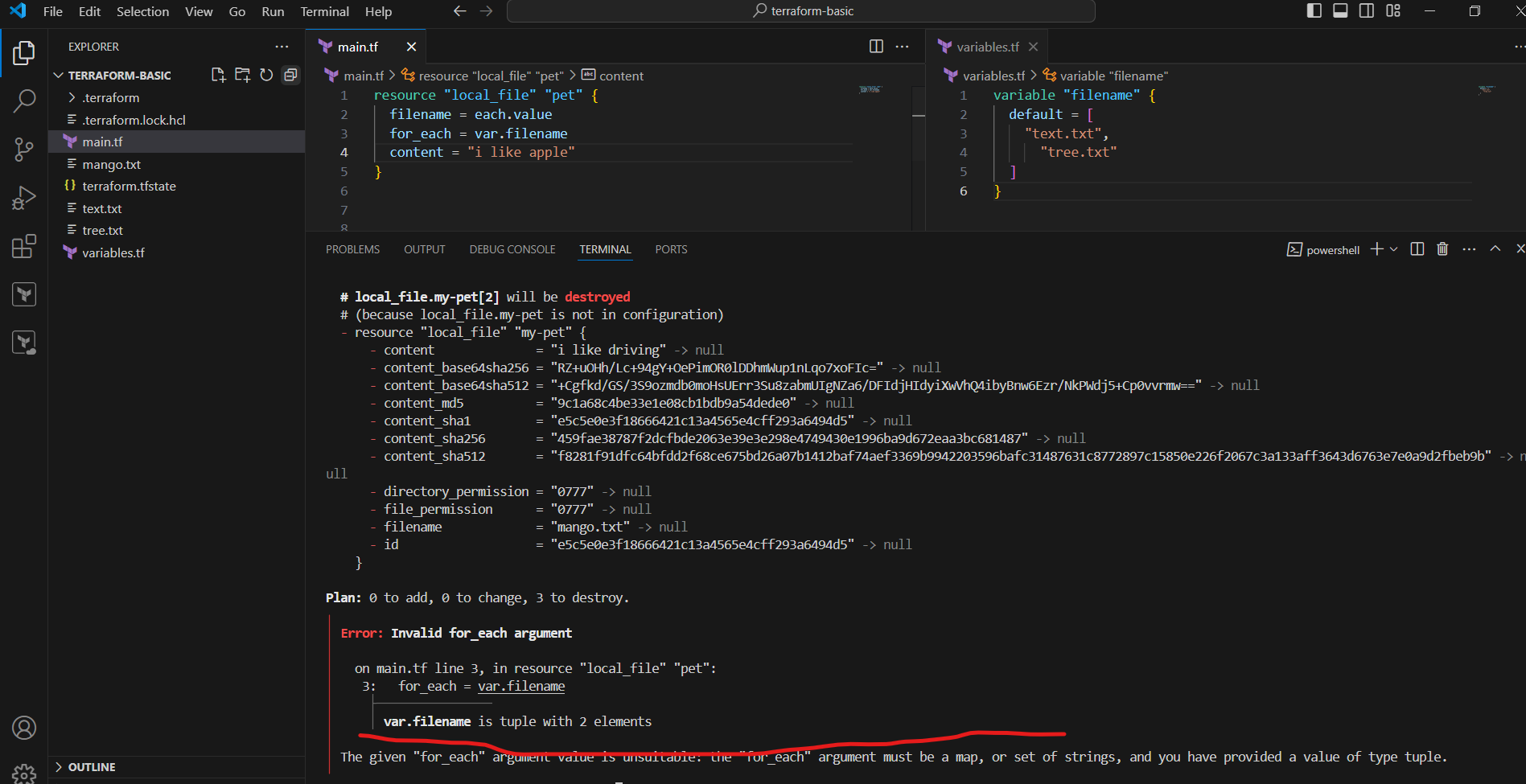


* Using **For-Each META Arguments**

\* here am not giving content, then it will throws the error , in for-each it asks to add a content

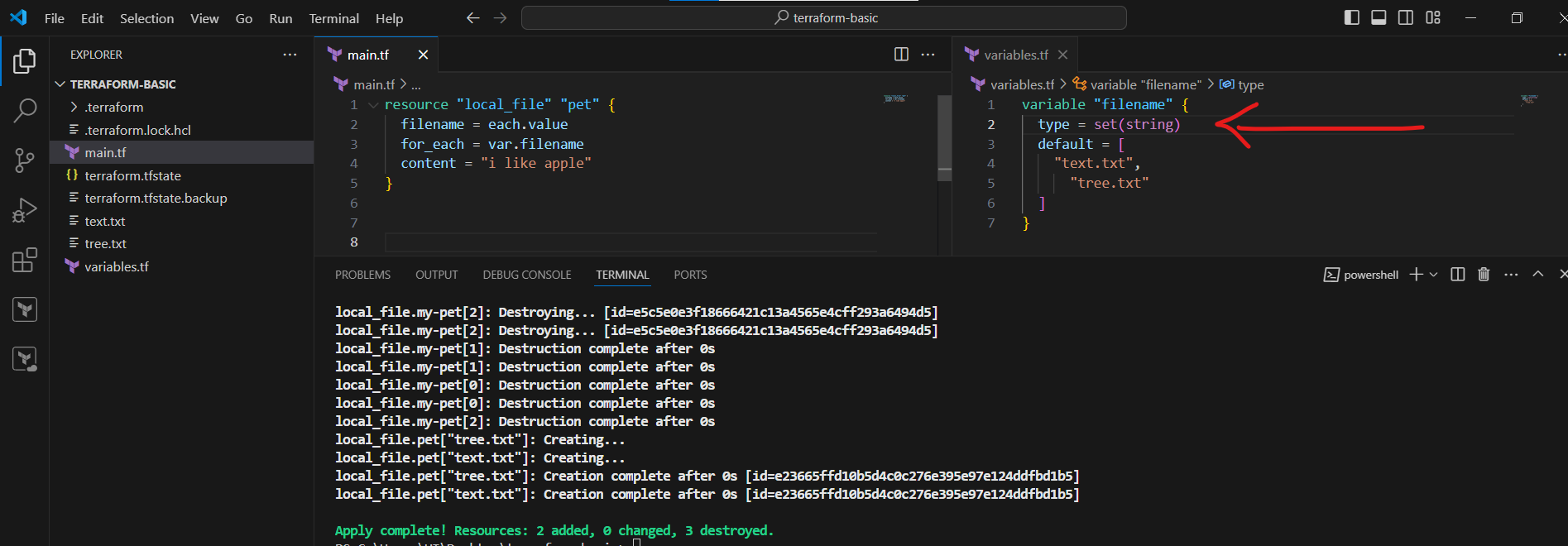


\*After I added content then also it shows error

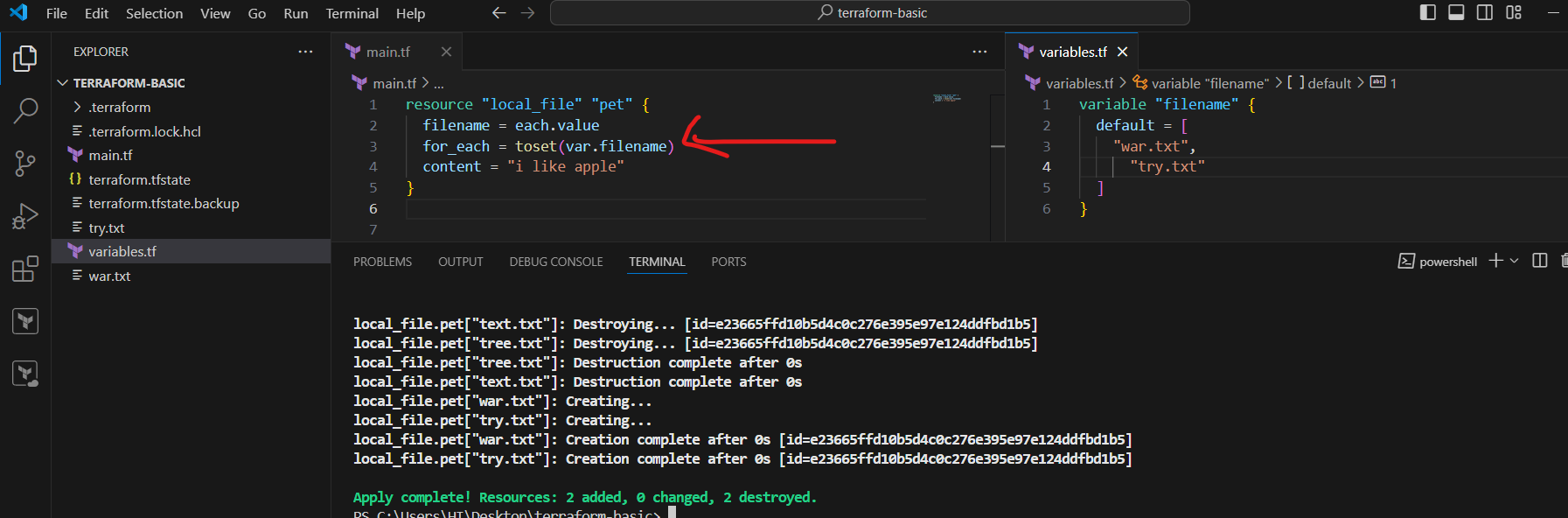


\* Now I have added type as String in variable , then it will executed

\* The list type will throw the error



\* By using inbuilt function called ”**toset**” to use the list varibale



\* Now am using **AWS\_IAM\_USER** to create a user

resource "aws\_iam\_user" "Admin-user" {

  name = "techie"

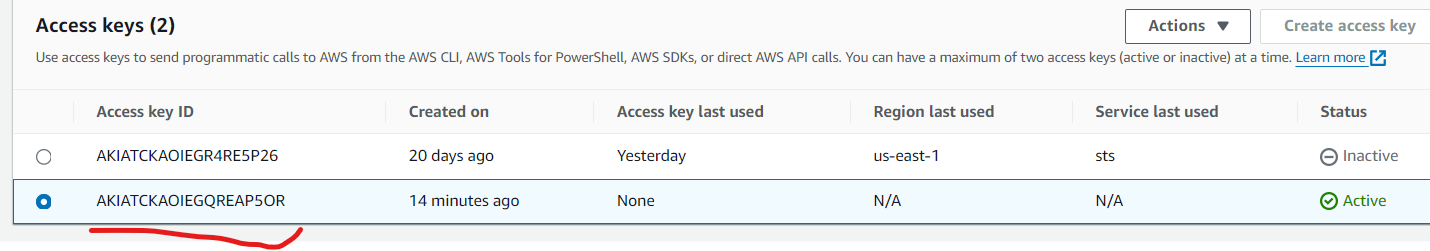
tags = {

  description = "Technical Team lead"

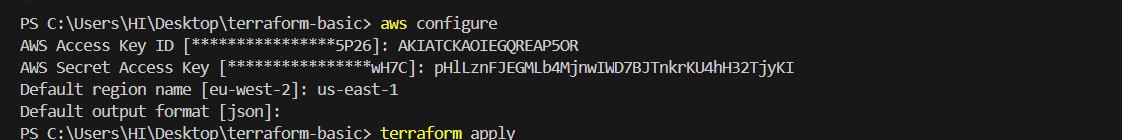
}

}

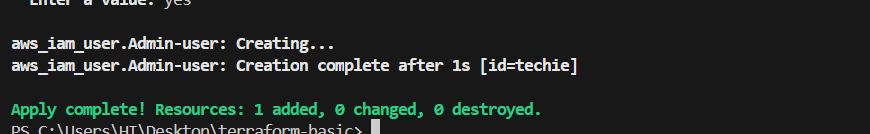
\*creating Access credentials

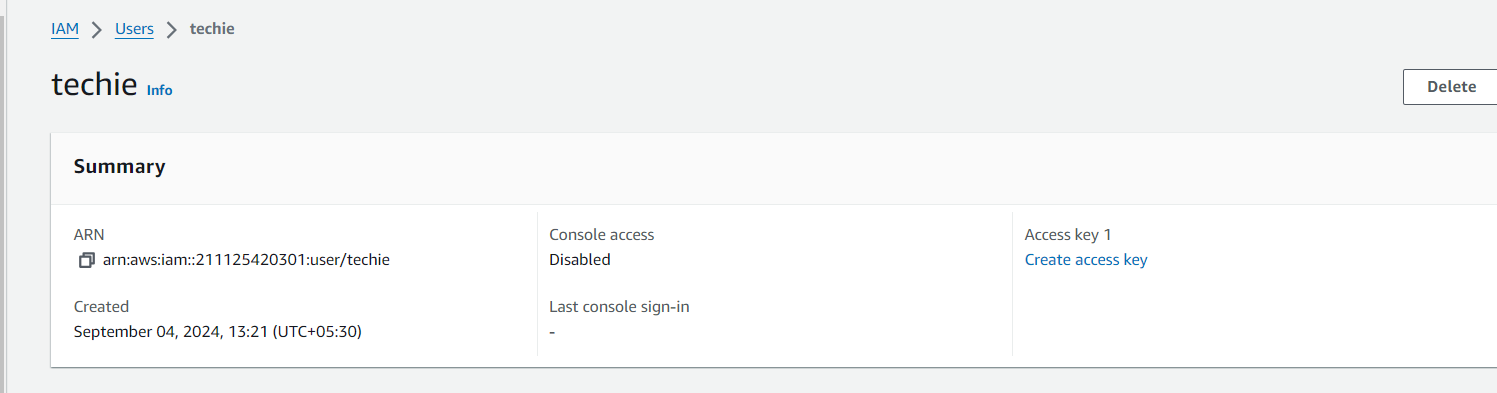


\* Configure the credentials in terraform for access to the terraform



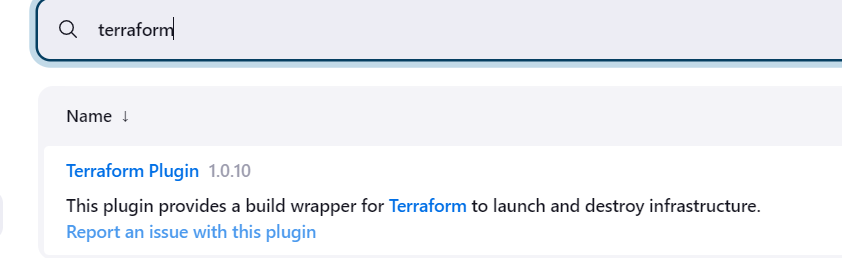
\* created a **IAM user** called **“techie”**





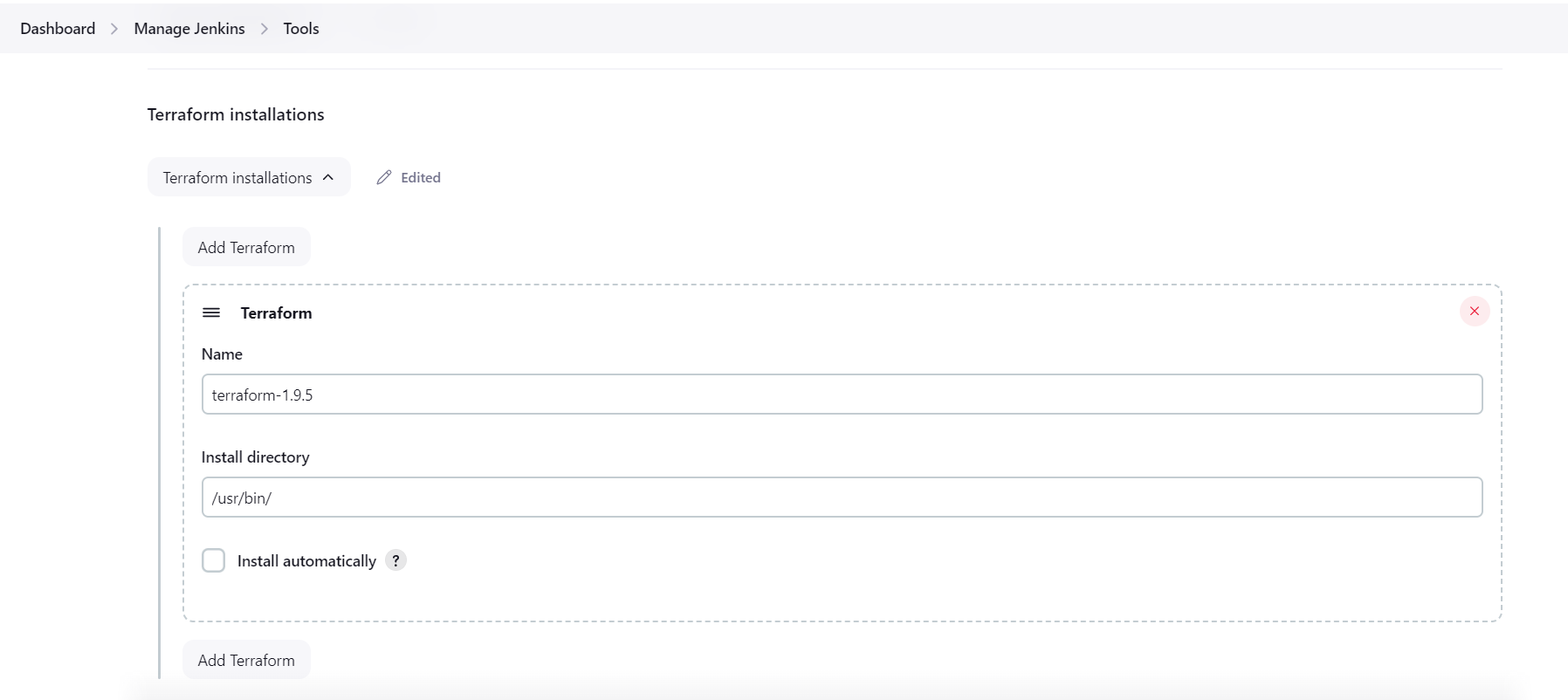
1. **Integrate terrafrom in jenkins using Terraform plugin.**

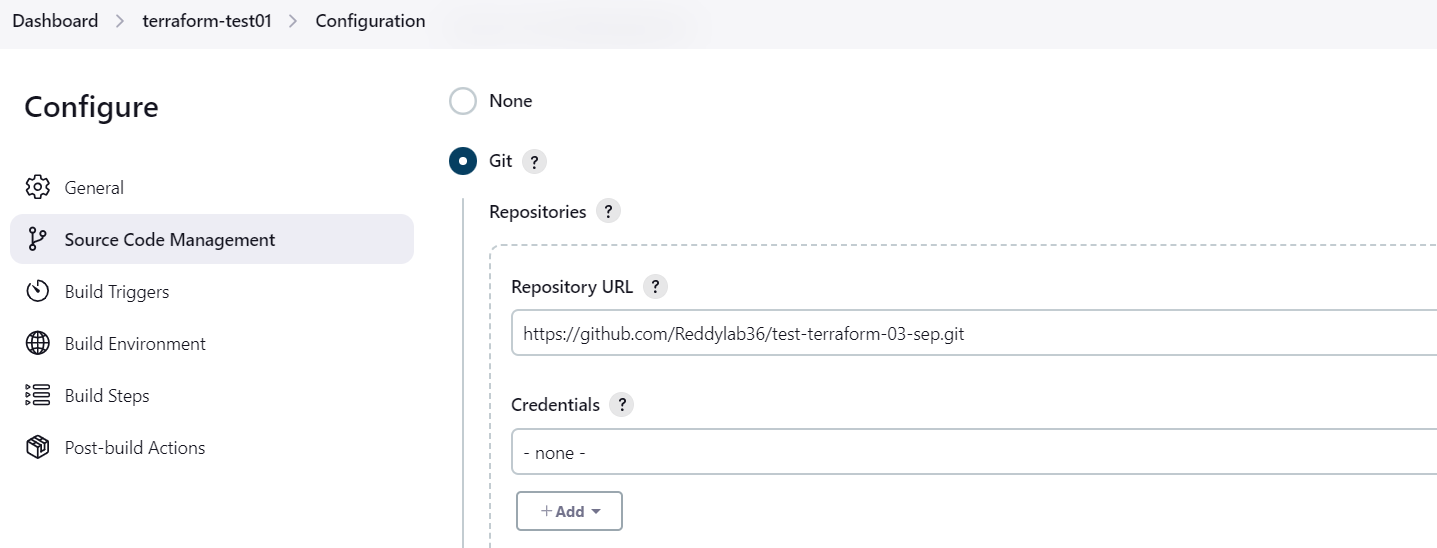
Login to Jenkins server→go to Dashboard→Go to manage jenkins→click on plugin→go to available plugin→search terraform plugin→Install

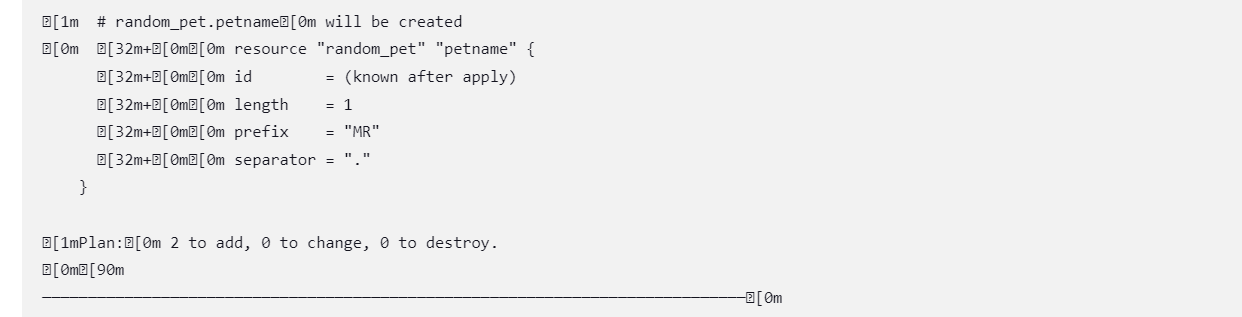


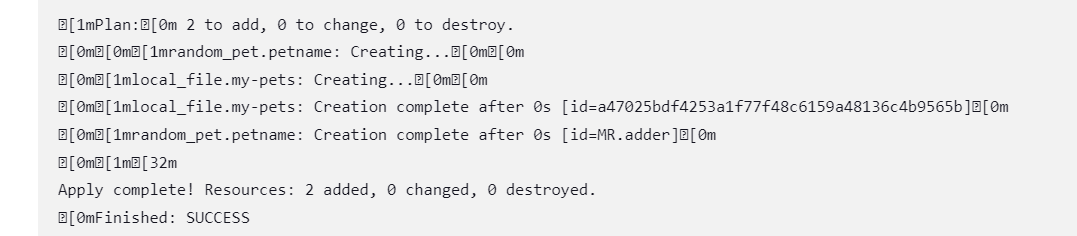
\* Go to Jenkins Dashboard→Manage Jenkins→Tools configuration → click on terraform Installation

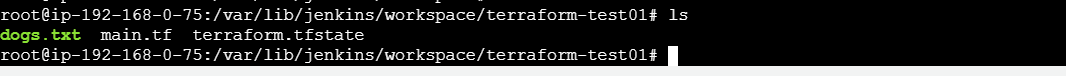
* Then select the terraform version
* Give the path for installed directory







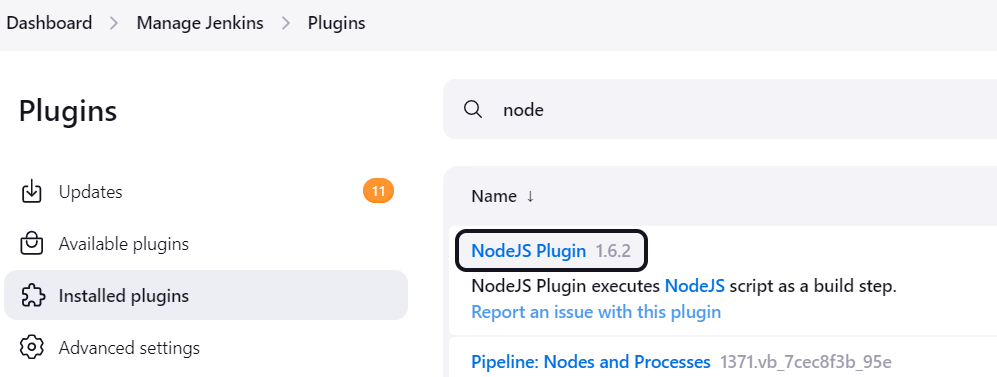




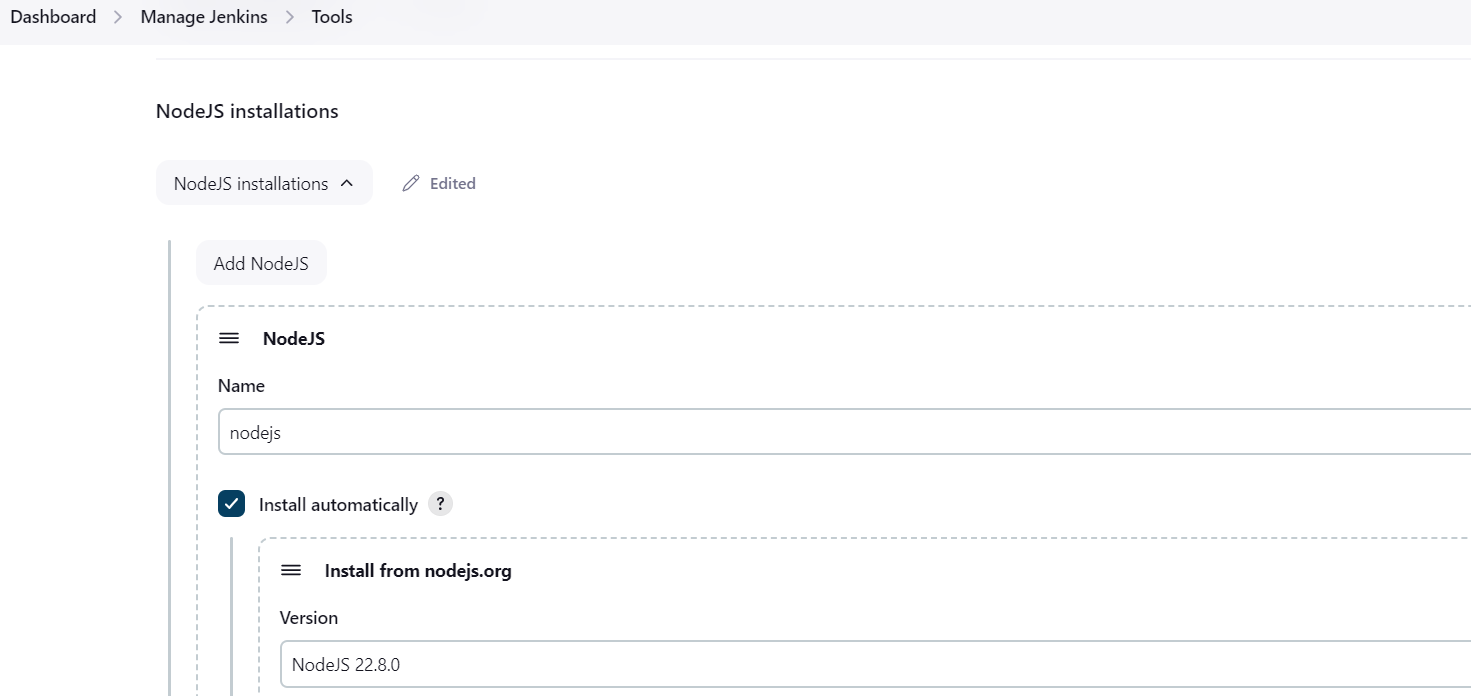
**4) Create CICD pipeline for Nodejs Application.**

**<https://github.com/betawins/Trading-UI.git>**

\*Go to jenkins Dashboard→Go to manage jenkins→Go to plugin→search nodejs plugin→install it



\* Go to jenkins Dashboard→Go to manage jenkins→Go to Tools configuration→select Nodejs Installation→give a name and select the version



\* Go to Dashboard→create the Job using pipeline→in configuration select the pipeleine script definition→create the script for CI/CD pipeline for nodejs Application

pipeline {

agent any

tools { nodejs "nodejs" }

stages {

stage('Build') {

steps {

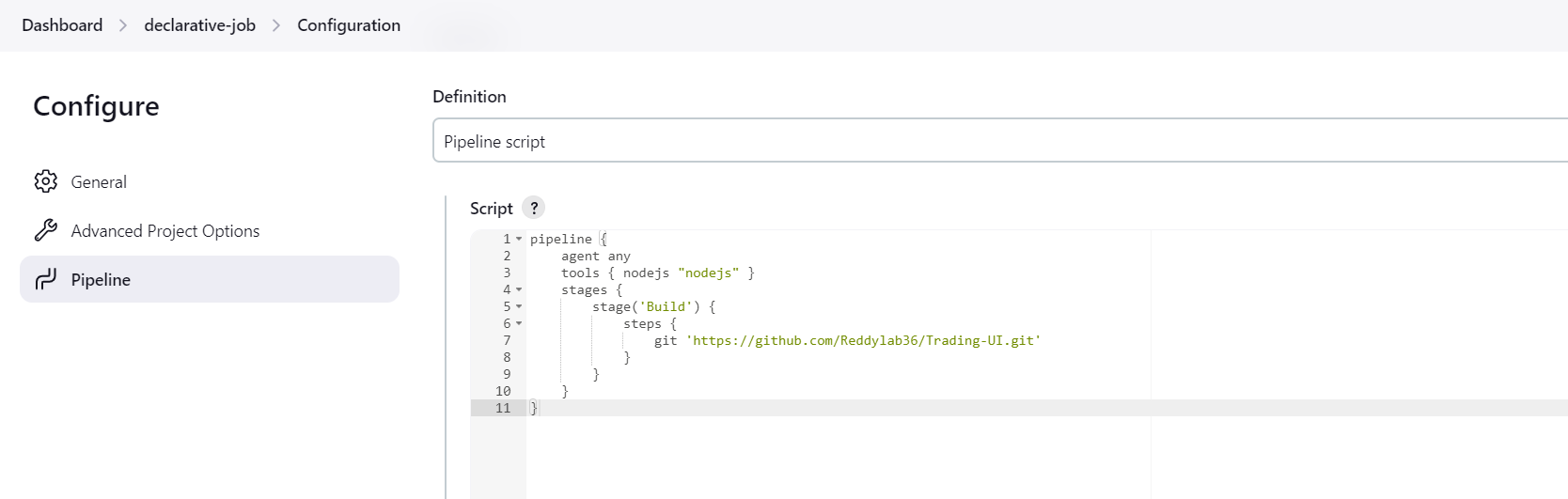
git 'https://github.com/Reddylab36/Trading-UI.git'

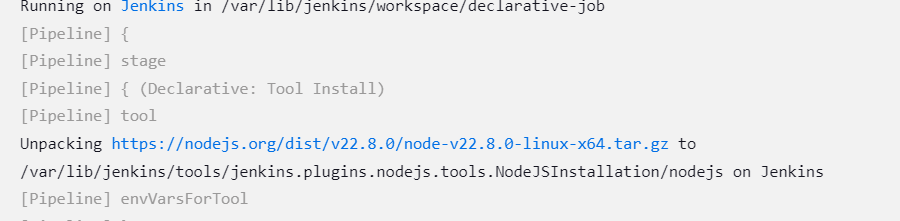
}

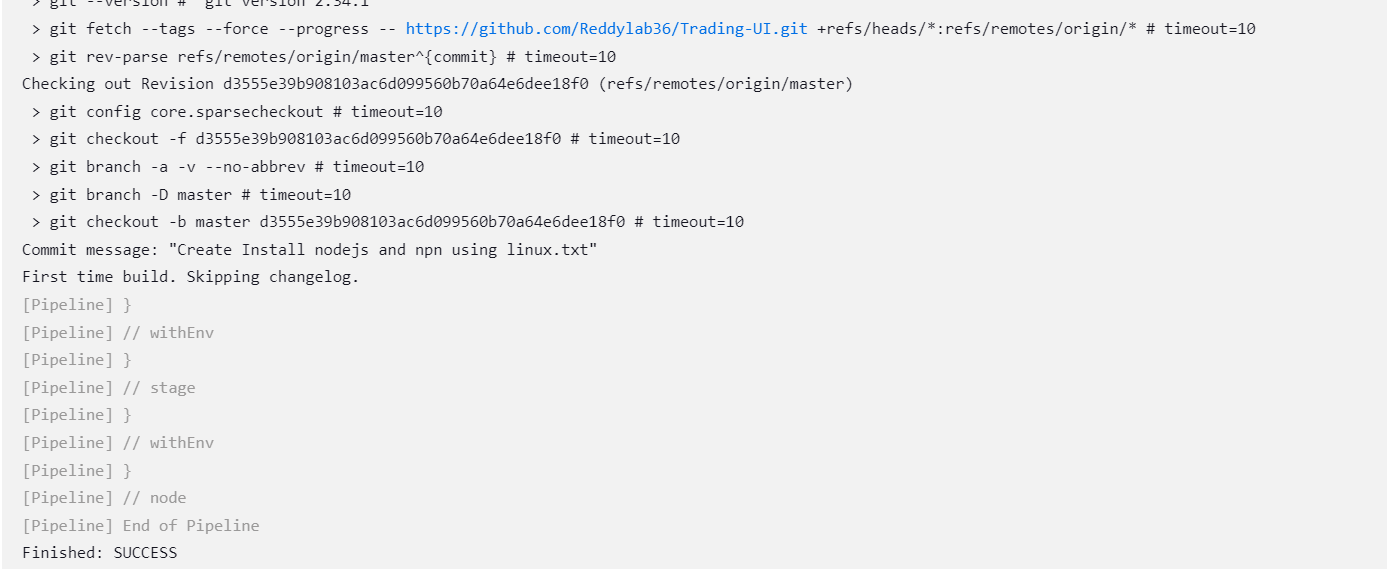
}

}

}







**5) Explain 10 Maven commands.**

* **Maven Commands**

With a simple command-line interface, Maven manages project dependencies, compiles source code, runs tests, and packages applications. Some basic Maven commands includes:

mvn clean: Cleans the project and removes all files generated by the previous build.

mvn compile: Compiles source code of the project.

mvn test-compile: Compiles the test source code.

mvn test: Runs tests for the project.

mvn package: Creates JAR or WAR file for the project to convert it into a distributable format.

mvn install: Deploys the packaged JAR/ WAR file to the local repository.

mvn site: generate the project documentation.

mvn validate: validate the project’s POM and configuration.

mvn idea:idea: generate project files for IntelliJ IDEA or Eclipse.

mvn release:perform: Performs a release build.

mvn deploy: Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.

mvn archetype:generate: This command is used to generate a new project from an archetype, which is a template for a project. This command is typically used to create new projects based on a specific pattern or structure.

mvn dependency:tree: This command is used to display the dependencies of the project in a tree format. This command is typically used to understand the dependencies of the project and troubleshoot any issues.