**1. Watch the Terraform-03 video.**

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

**3. Integrate Terraform in Jenkins using the Terraform plugin.**

**4. Create one Jenkins job using Maven Project for the code below with two stages:**

**Stage 1: Git clone**

**Stage 2: Maven Compilation Code:** [**https://github.com/betawins/java-Working-app.git**](https://github.com/betawins/java-Working-app.git)

[**https://github.com/betawins/hiring-app**](https://github.com/betawins/hiring-app)

**5. Use the same code and create a parameterized job in Jenkins with:**

**Stage 1: Git clone**

**Stage 2: Maven Compilation Code:** [**https://github.com/betawins/java-Working-app.git**](https://github.com/betawins/java-Working-app.git)

[**https://github.com/betawins/hiring-app**](https://github.com/betawins/hiring-app)

**6. What are the global variables in Jenkins?**

**1. Watch the Terraform-03 video.**

**Completed.**

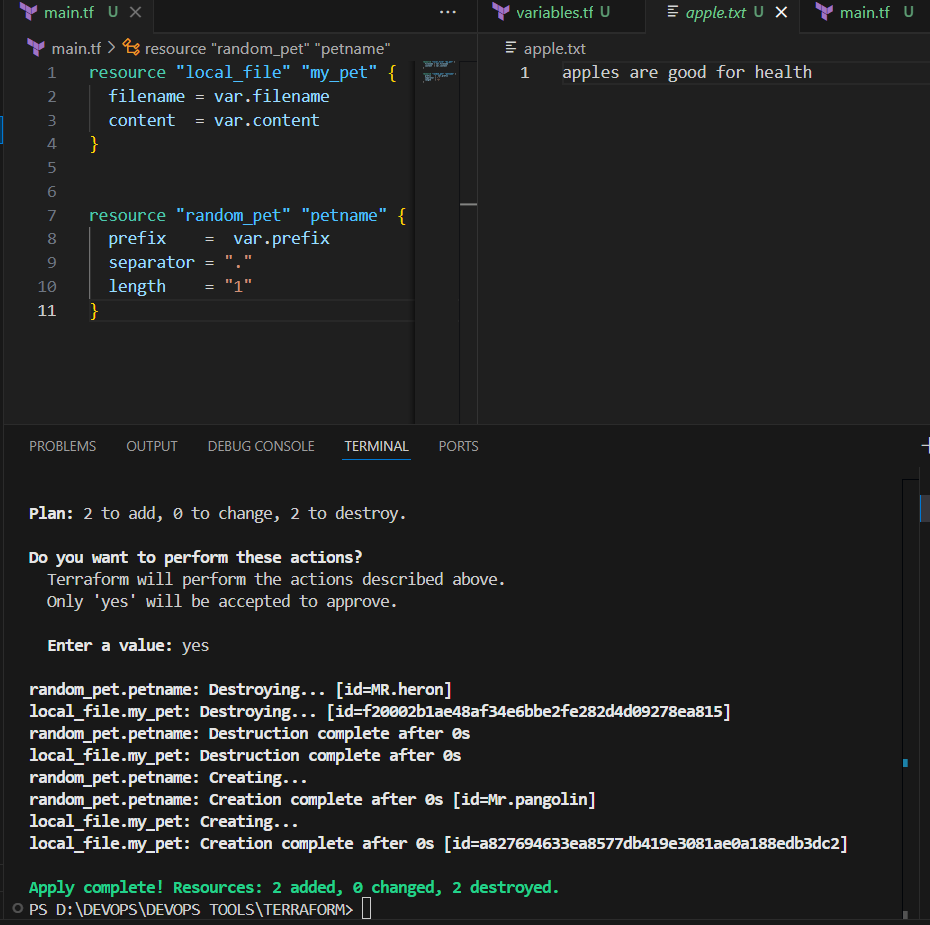
**2. Execute the script shown in the video.**

**Using Variables**

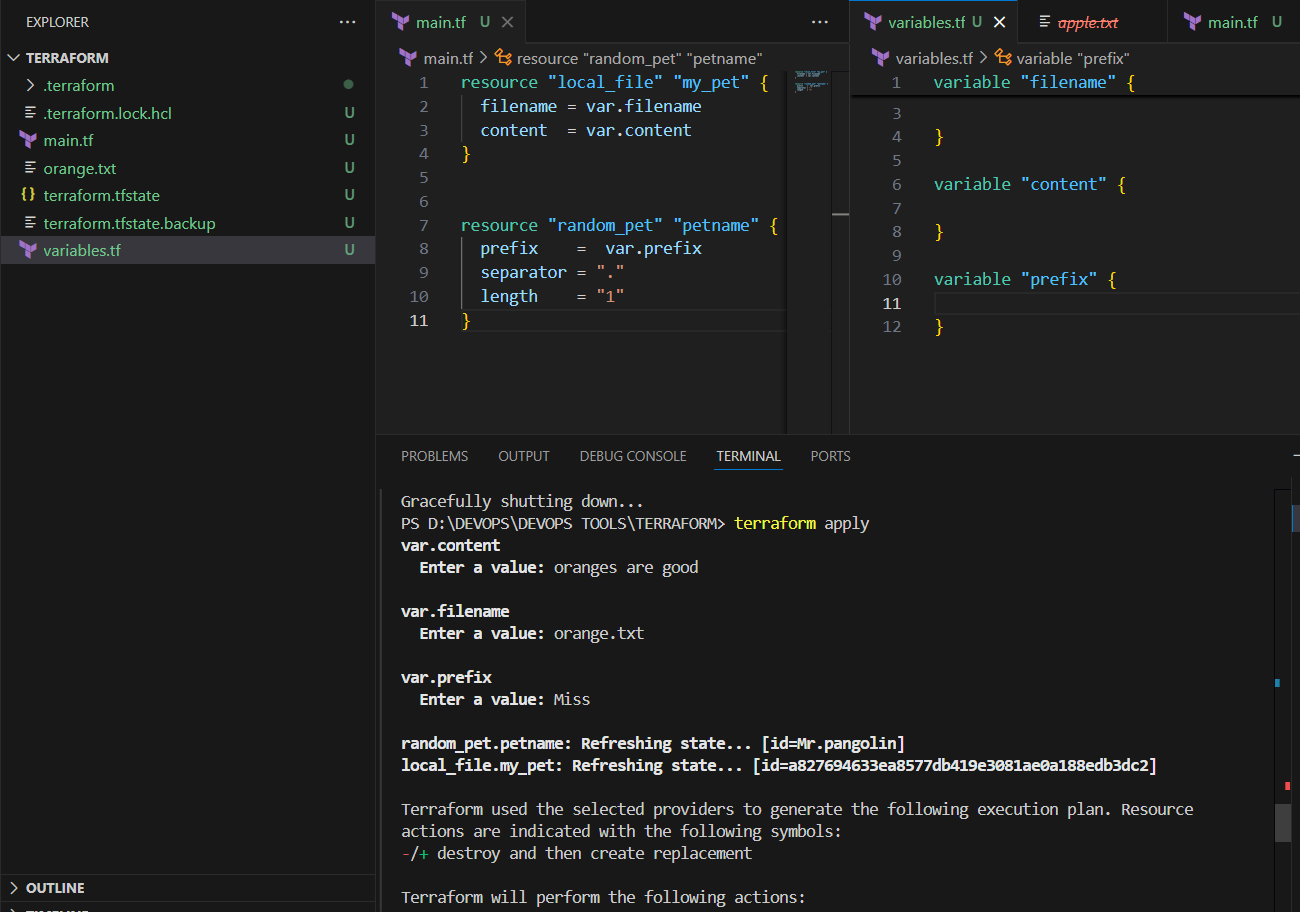


Once we do terraform apply

The variables send parameters and required [apple.tx](http://apple.tx)t is created

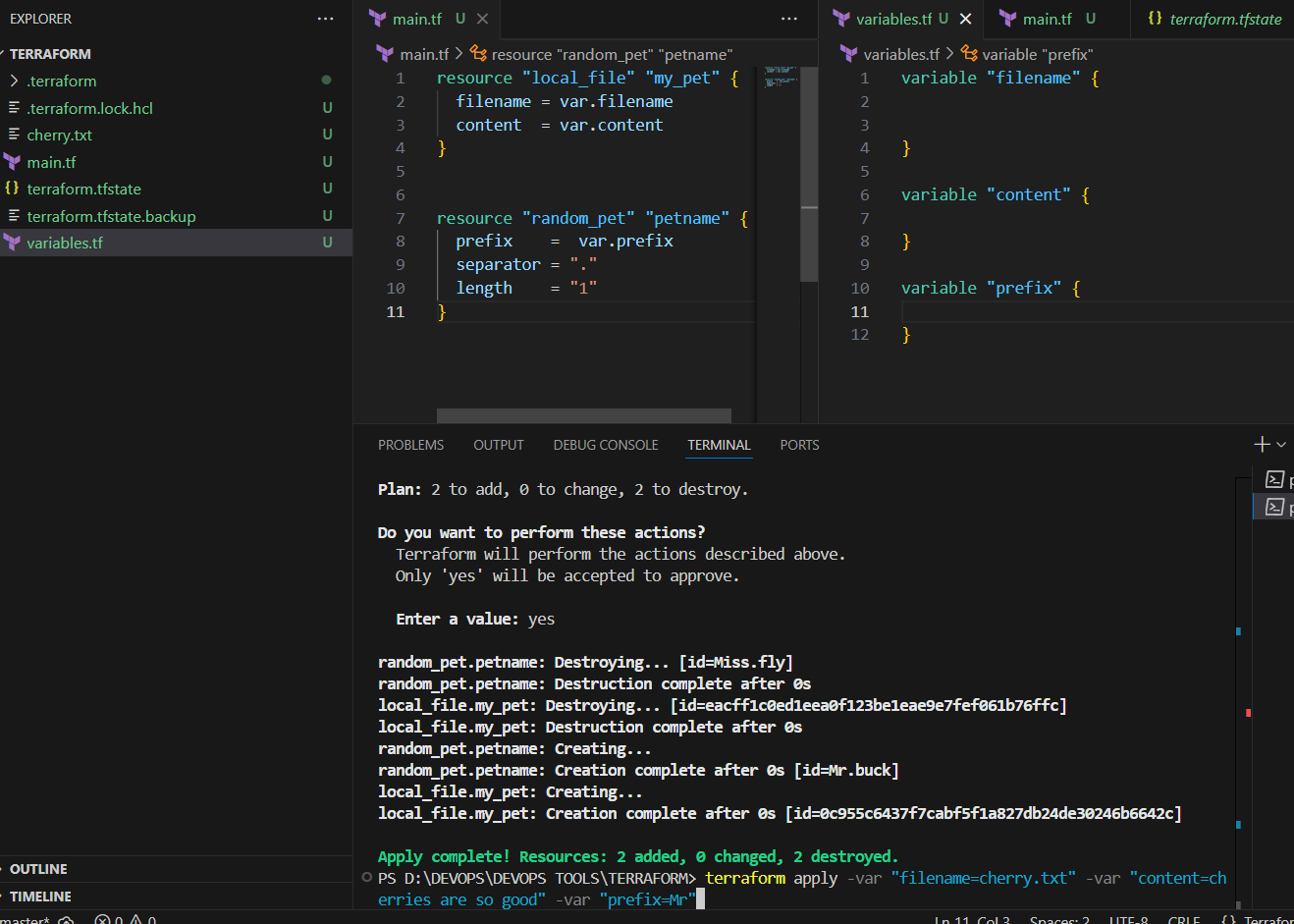


Interactive way for passing variables

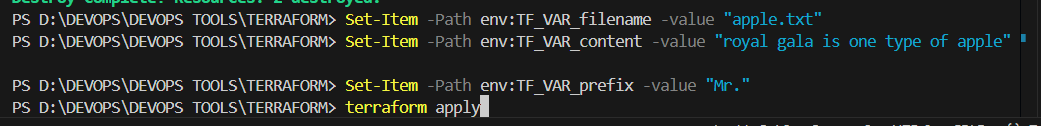


**Command line Flags**

**terraform apply -var "filename=cherry.txt" -var "content=cherries are so good" -var "prefix=Mr"**

****

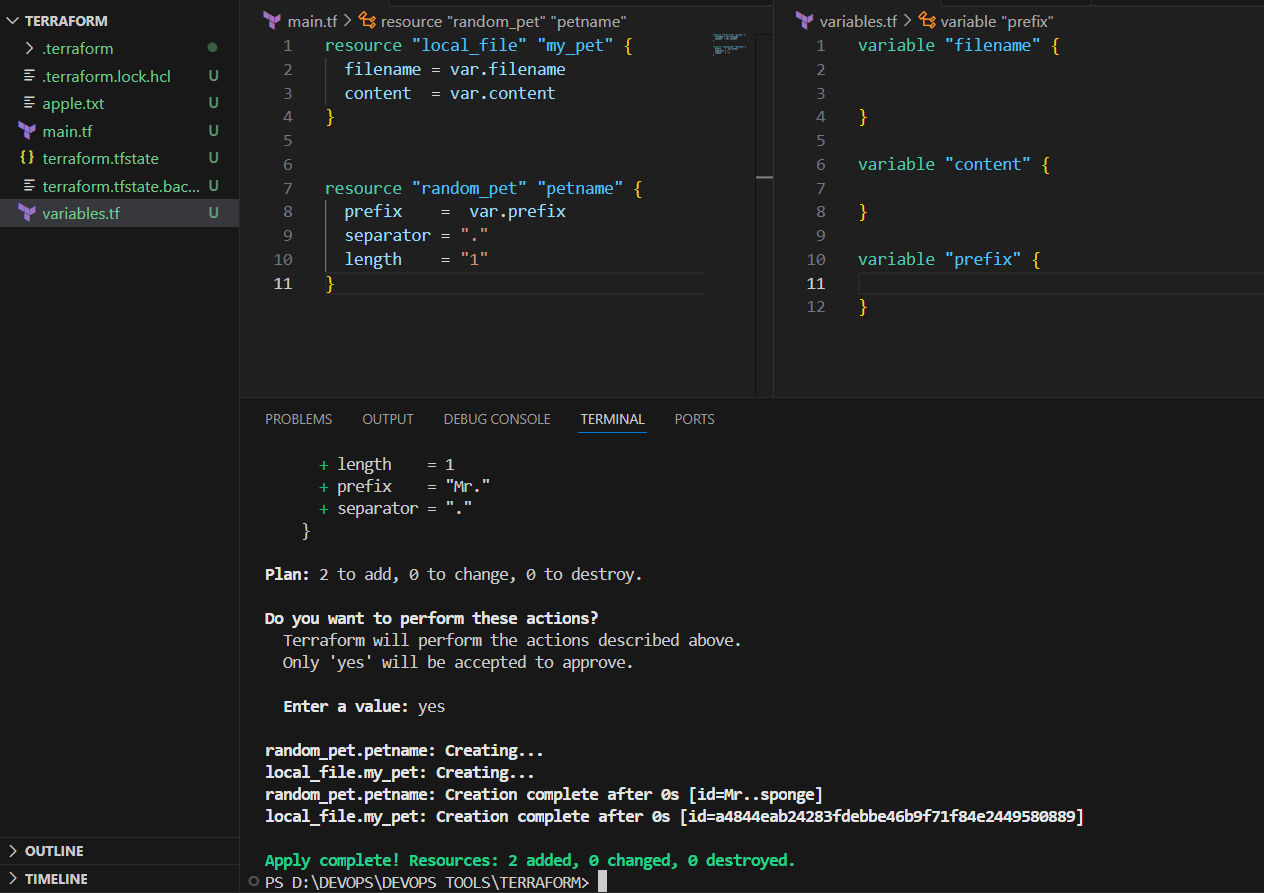
**Environment Variables**

****

**Set-Item -Path env:TF\_VAR\_filename -value "apple.txt"**

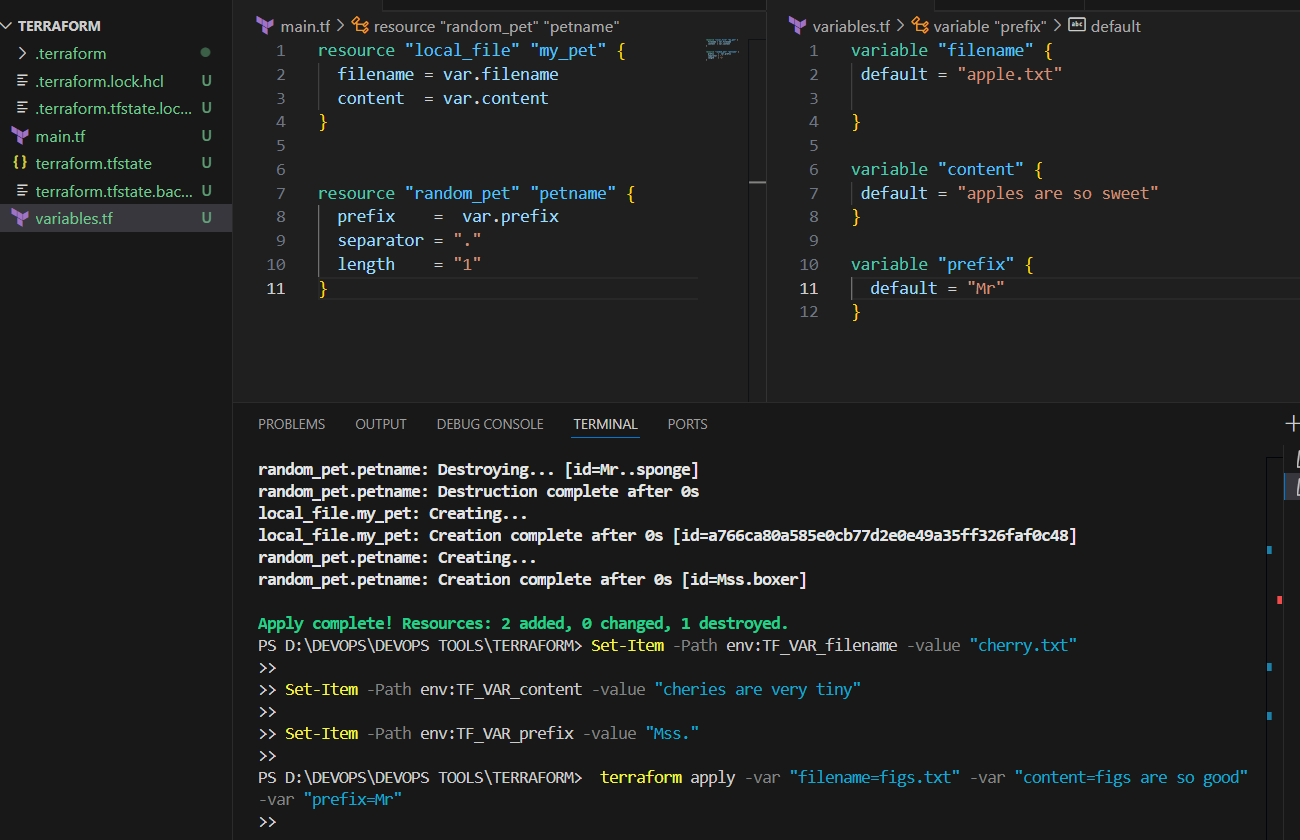
**Set-Item -Path env:TF\_VAR\_content -value "royal gala is one type of apple"**

**Set-Item -Path env:TF\_VAR\_prefix -value "Mr."**

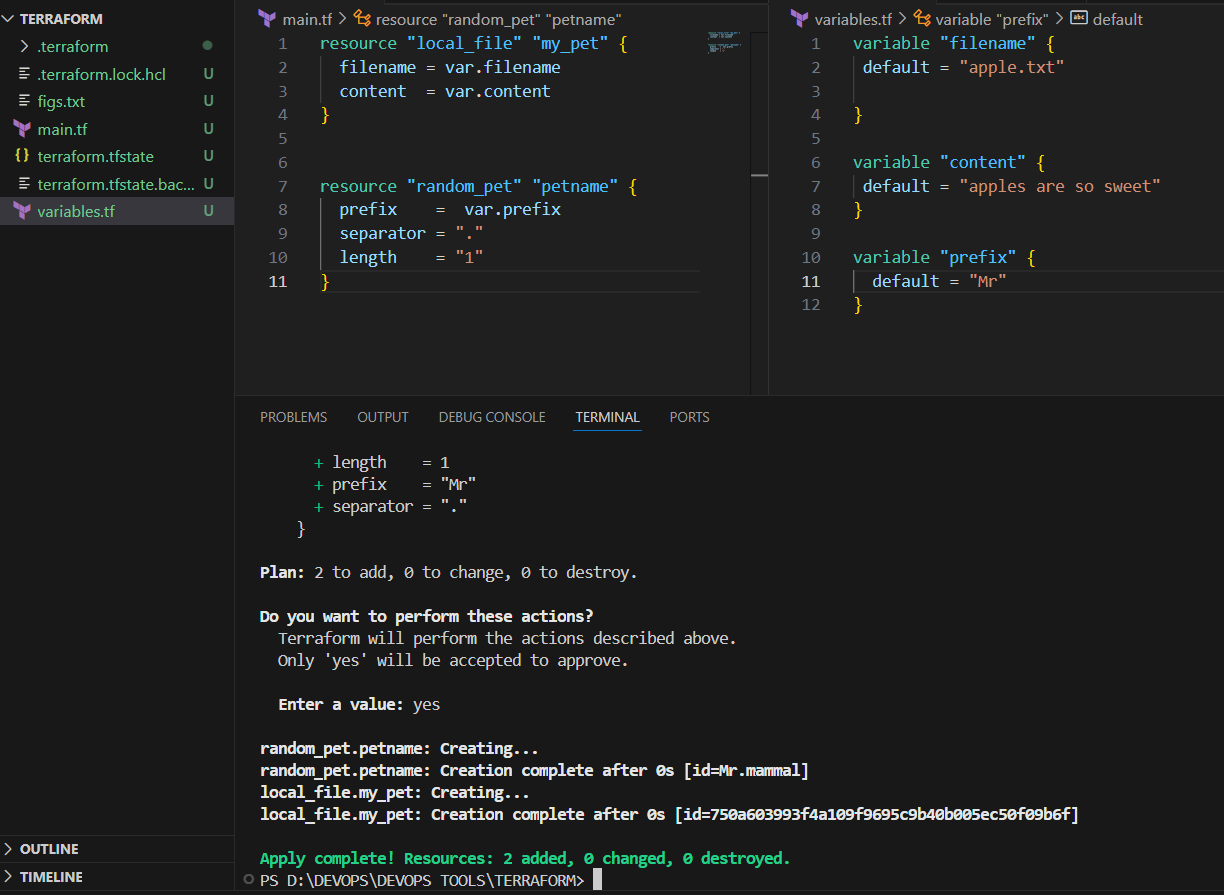
****

**Varibale Precedence Order**

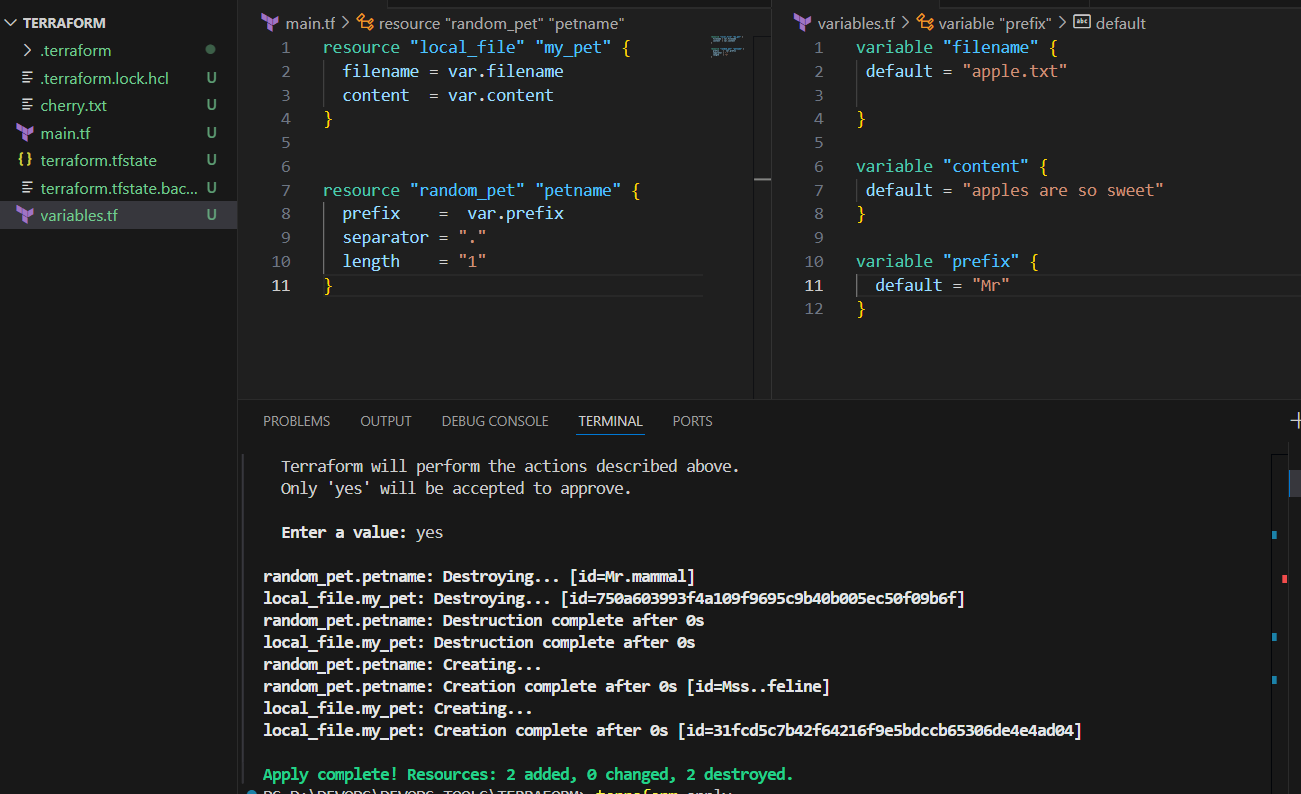
1. **Environment variables**
2. **Terraform.tfvars**
3. **\*.auto.tfvars (alphabetical order)**
4. **-var or -var-file (Command Line Flags)**

****

**Command line flag executed figs.txt created**

****

**When again i do terraform apply envirnonment vaiables taken cherry.txt created**

****

**Resource attribute (Implicit Behavior)**

**resource "local\_file" "my\_pet" {**

**filename = "pets.txt"**

**content = "my cat name is ${random\_pet.petname.id}"**

**}**

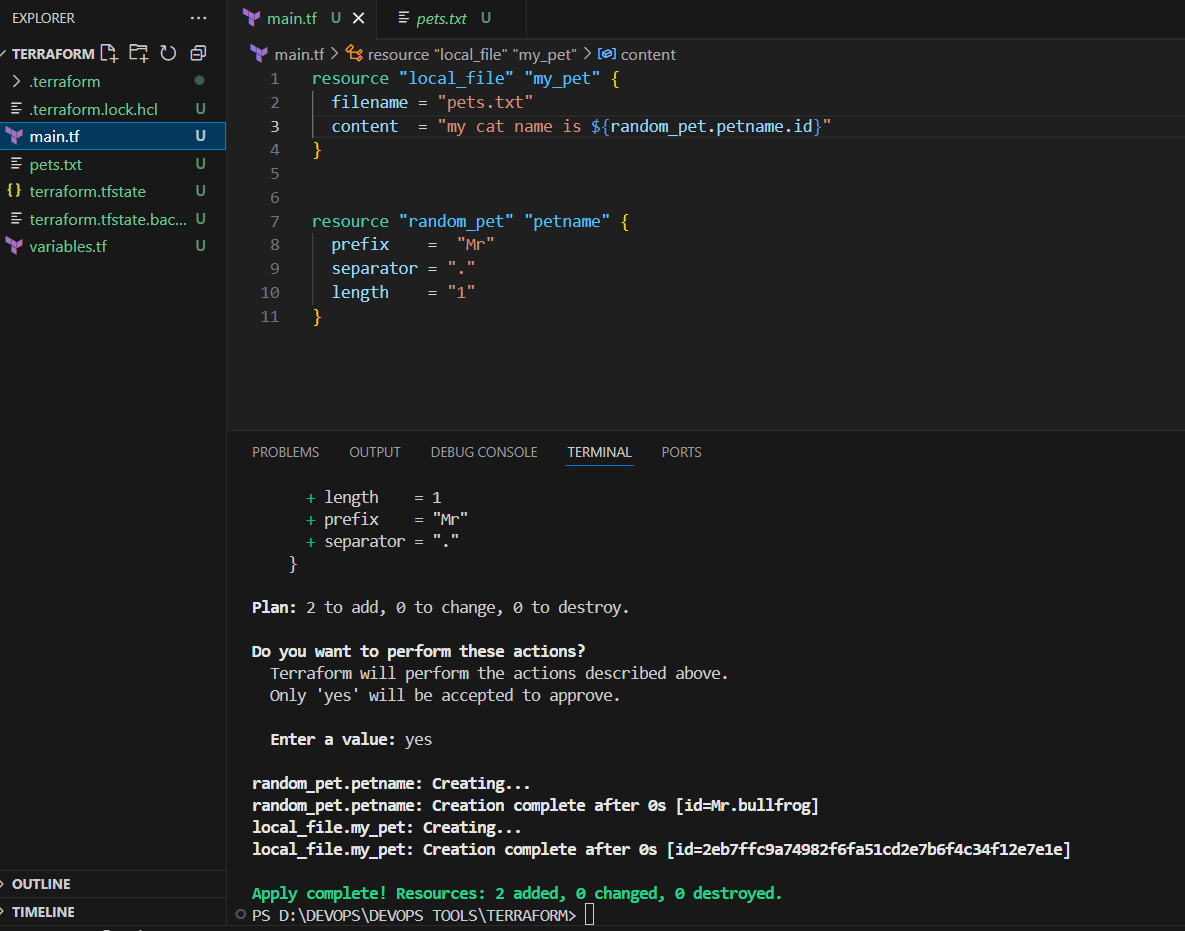
**resource "random\_pet" "petname" {**

**prefix = "Mr"**

**separator = "."**

**length = "1"**

**}**

****

**Output variables**

**Displays the output the resources.**

**In** [**main.tf**](http://main.tf) **we can write output**

**output "petname" {**

**value=random\_pet.petname**

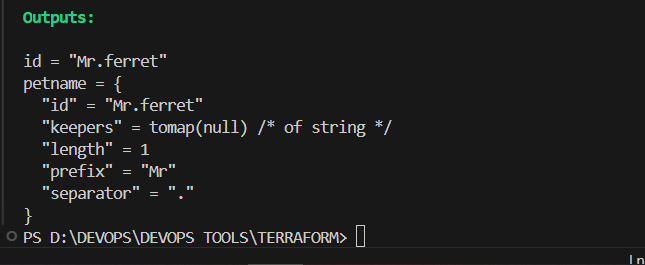
**}**

**output "id" {**

**value=random\_pet.petname.id**

**}**

**Terraform apply will display the output.**

****

**State file locking**

**No multiple users can execute terraform same template at the same time.**

**3. Integrate Terraform in Jenkins using the Terraform plugin.**

1. **Launched Jenkins ec2, installed java, git, jenkins, terraform.**

**# Download latest version (Linux 64-bit)**

**wget** [**https://releases.hashicorp.com/terraform/1.13.1/terraform\_1.13.1\_linux\_amd64.zip**](https://releases.hashicorp.com/terraform/1.13.1/terraform_1.13.1_linux_amd64.zip)

**# Unzip and move**

**unzip terraform\_1.13.1\_linux\_amd64.zip**

**sudo mv terraform /usr/local/bin/**

**# Verify**

**terraform -v**

1. **In jenkins master installed terraform plugin.**
2. **In my git repo** [**https://github.com/imrankhanmohammad257/hiring-app.git**](https://github.com/imrankhanmohammad257/hiring-app.git)
3. **Added** [**main.tf**](http://main.tf) **and** [**variables.tf**](http://variables.tf) **, Terraform.tfvars, jenkinsfile**

[**main.tf**](http://main.tf)

**resource "local\_file" "my\_pet" {**

**filename = var.filename**

**content = var.content**

**}**

**resource "random\_pet" "petname" {**

**prefix = var.prefix**

**separator = "."**

**length = 1**

**}**

[**variables.tf**](http://variables.tf)

**variable "filename" {}**

**variable "content" {}**

**variable "prefix" {}**

**Terraform.tfvars**

**filename = "apple.txt"**

**content = "royal gala is one type of apple"**

**prefix = "Mr."**

**Jenkinsfile-terraform**

**Created a declarative script.**

**pipeline {**

**agent any**

**stages {**

**stage('Checkout') {**

**steps {**

**git url: 'https://github.com/imrankhanmohammad257/hiring-app.git', branch: 'main'**

**}**

**}**

**stage('Terraform Init') {**

**steps {**

**sh 'terraform init -input=false'**

**}**

**}**

**stage('Terraform Plan') {**

**steps {**

**sh 'terraform plan -no-color -out=tfplan'**

**}**

**}**

**stage('Approve') {**

**steps {**

**input message: 'Do you want to apply Terraform changes?'**

**}**

**}**

**stage('Terraform Apply') {**

**steps {**

**sh 'terraform apply -auto-approve tfplan'**

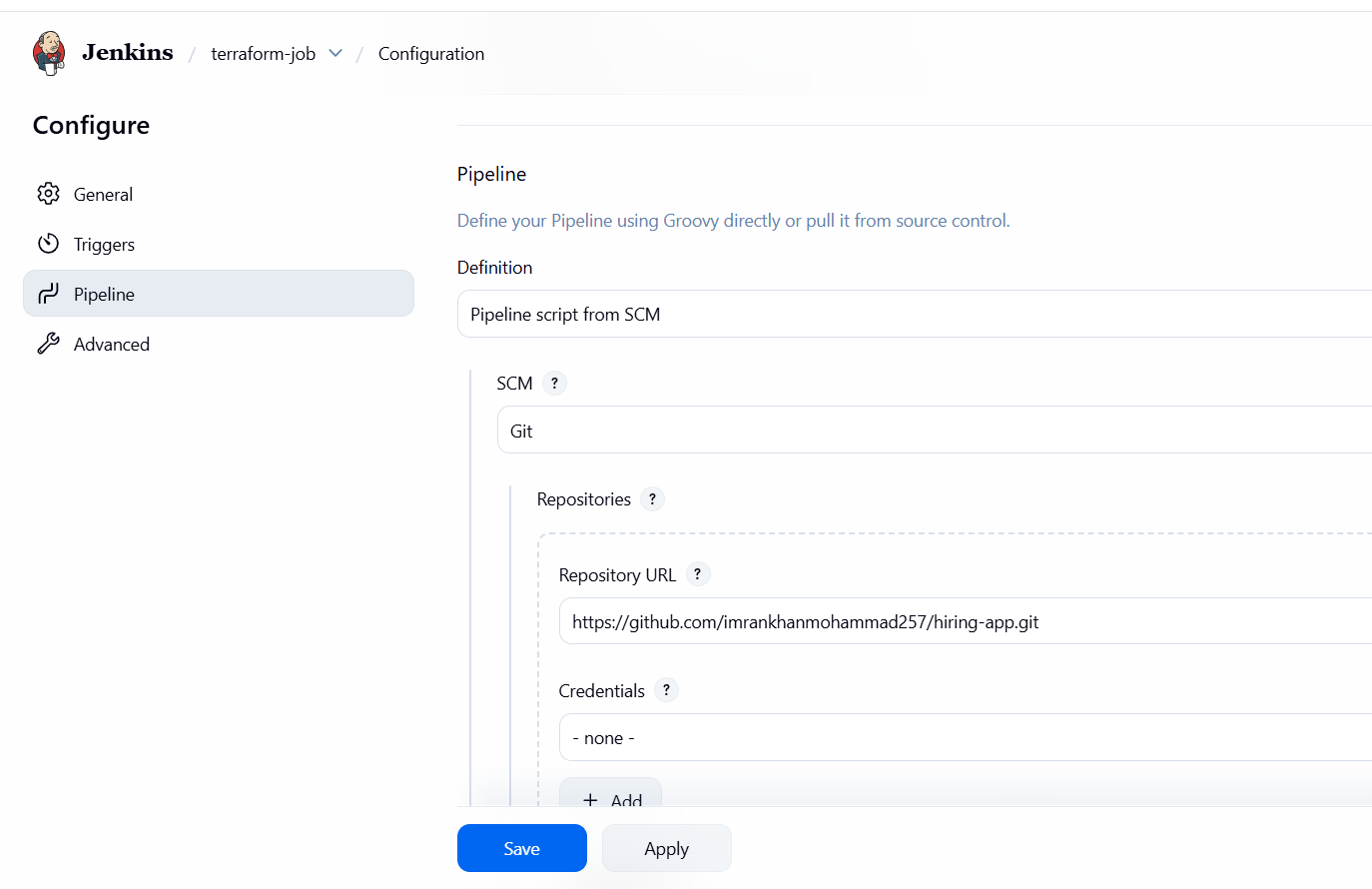
**}**

**}**

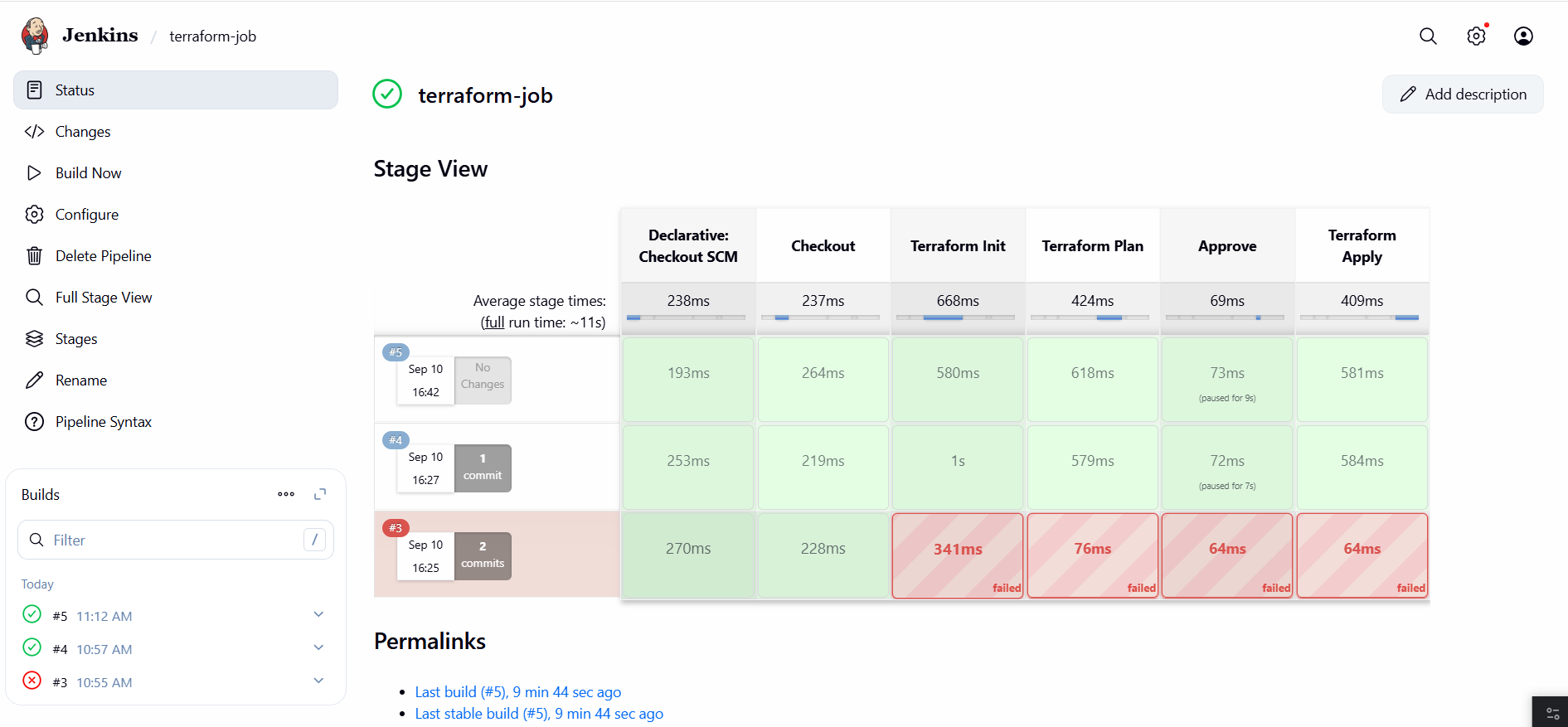
**}**

**}**

**In jenkins master create and build job**

****

**Job build successfully.**

****

**In workspace job created and files displayed.**

****

**👉 Here’s the communication chain:**

1. **Jenkins clones your GitHub repo (where main.tf, variables.tf, and terraform.tfvars are stored).**
2. **Jenkins runs Terraform commands (init, plan, apply).**
3. **Terraform reads:**
   * **variables.tf → what inputs are required.**
   * **terraform.tfvars → the default values for those inputs.**
   * **main.tf → the resources that use those values.**
4. **Jenkins shows the output of terraform plan and apply in the pipeline console.**

**🔗 How they talk to each other (Simplified Flow)**

**Jenkinsfile → runs terraform commands**

**↓**

**Terraform reads main.tf → needs var.filename, var.content, var.prefix**

**↓**

**Terraform checks variables.tf → sees those vars are declared**

**↓**

**Terraform loads terraform.tfvars → injects real values**

**↓**

**Resources get created (local\_file + random\_pet)**

**✅ So in short:**

* **main.tf = blueprint (what to build).**
* **variables.tf = input definitions (what’s needed).**
* **terraform.tfvars = values (what to fill in).**
* **Jenkinsfile = automation (how to run Terraform in CI/CD).**

**4. Create one Jenkins job using Maven Project for the code below with two stages:**

**Stage 1: Git clone**

**Stage 2: Maven Compilation Code:** [**https://github.com/betawins/java-Working-app.git**](https://github.com/betawins/java-Working-app.git)

[**https://github.com/betawins/hiring-app**](https://github.com/betawins/hiring-app)

## **Step 1: Jenkins Job Setup**

1. **Login to Jenkins.**
2. **Create a new job:**
   * **Click “New Item”.**
   * **Enter a name, e.g., terraform-maven.**
   * **Select Pipeline → Click OK.**
3. **Pipeline Configuration:**
   * **Under Pipeline section, select Pipeline script from SCM.**
   * **SCM: Git**
   * **Repository URL: https://github.com/betawins/hiring-app**
   * **Branch: main**
   * **Script Path: Jenkinsfile-maven (your pipeline file).**
4. **Configure Maven Tool:**
   * **Go to Manage Jenkins → Global Tool Configuration → Maven.**
   * **Add Maven3 with a name Maven3.**
   * **Jenkins will download it automatically if not installed.**
5. **Save the job.**

## **Step 2: Understand the Pipeline (Jenkinsfile-maven)**

**Your pipeline has 5 main stages:**

### **Stage 1: Checkout**

**stage('Checkout') {**

**steps {**

**git url: 'https://github.com/imrankhanmohammad257/hiring-app.git', branch: 'main'**

**}**

**}**

**Jenkins clones your repository into the workspace.**

**Workspace path (e.g.): /var/lib/jenkins/workspace/terraform-maven/**

**Stage 2: Terraform Init & Plan**

**stage('Terraform Init & Plan') {**

**steps {**

**sh 'terraform init -input=false'**

**sh 'terraform validate'**

**sh 'terraform plan -out=tfplan'**

**}**

**}**

**terraform init: initializes Terraform, downloads providers.**

**terraform validate: checks .tf files for syntax errors.**

**terraform plan -out=tfplan: generates a plan file for apply.**

**Important:**

* **Your main.tf points to the Jenkins workspace for the file output:**

**filename = "/var/lib/jenkins/workspace/terraform-maven/${var.filename}"**

**This avoids the permission denied error you saw earlier.**

**Stage 3: Approve Apply**

**stage('Approve Apply') {**

**steps {**

**input message: 'Proceed with Terraform Apply?'**

**}**

**}**

**This is a manual approval step.**

**Jenkins will pause and wait for a user to click “Proceed” in the web interface.**

**Helps prevent accidental apply in production.**

**Stage 4: Terraform Apply**

**stage('Terraform Apply') {**

**steps {**

**sh 'terraform apply -auto-approve tfplan || terraform apply -auto-approve'**

**}**

**}**

**Applies the Terraform plan.**

**If tfplan file is missing, it applies directly (fallback).**

**This will create resources defined in main.tf, e.g., apple.txt in Jenkins workspace.**

**Stage 5: Maven Build**

**stage('Maven Build') {**

**steps {**

**sh 'mvn -B -DskipTests clean package'**

**}**

**}**

* **Runs Maven to compile your Java project (pom.xml in repo).**
* **-B = batch mode (non-interactive), -DskipTests = skip tests to speed up.**

**Post Actions**

**post {**

**failure {**

**echo "❌ Pipeline failed. Please check logs."**

**}**

**success {**

**echo "✅ Pipeline succeeded!"**

**}**

**}**

**Prints success or failure message in console output.**

## **Step 3: Terraform Files**

* **variables.tf declares variables:**

**variable "filename" {}**

**variable "content" {}**

**variable "prefix" {}**

**terraform.tfvars provides values:**

**filename = "apple.txt"**

**content = "royal gala is one type of apple"**

**prefix = "Mr."**

**main.tf writes a file using these variables:**

**resource "local\_file" "my\_pet" {**

**content = var.content**

**filename = "/var/lib/jenkins/workspace/terraform-maven/${var.filename}"**

**file\_permission = "0777"**

**directory\_permission = "0777"**

**}**

**This ensures Terraform writes the file where Jenkins can access it**

**Jenkinsfile-maven**

**pipeline {**

**agent any**

**tools {**

**maven "Maven3" // Make sure Maven3 is configured in Jenkins global tools**

**}**

**environment {**

**TF\_IN\_AUTOMATION = '1'**

**}**

**stages {**

**stage('Checkout') {**

**steps {**

**git url: 'https://github.com/imrankhanmohammad257/hiring-app.git', branch: 'main'**

**}**

**}**

**stage('Terraform Init & Plan') {**

**steps {**

**sh 'terraform init -input=false'**

**sh 'terraform validate'**

**sh 'terraform plan -out=tfplan'**

**}**

**}**

**stage('Approve Apply') {**

**steps {**

**input message: 'Proceed with Terraform Apply?'**

**}**

**}**

**stage('Terraform Apply') {**

**steps {**

**// try with plan file, fallback to direct apply if not found**

**sh 'terraform apply -auto-approve tfplan || terraform apply -auto-approve'**

**}**

**}**

**stage('Maven Build') {**

**steps {**

**sh 'mvn -B -DskipTests clean package'**

**}**

**}**

**}**

**post {**

**failure {**

**echo "❌ Pipeline failed. Please check logs."**

**}**

**success {**

**echo "✅ Pipeline succeeded!"**

**}**

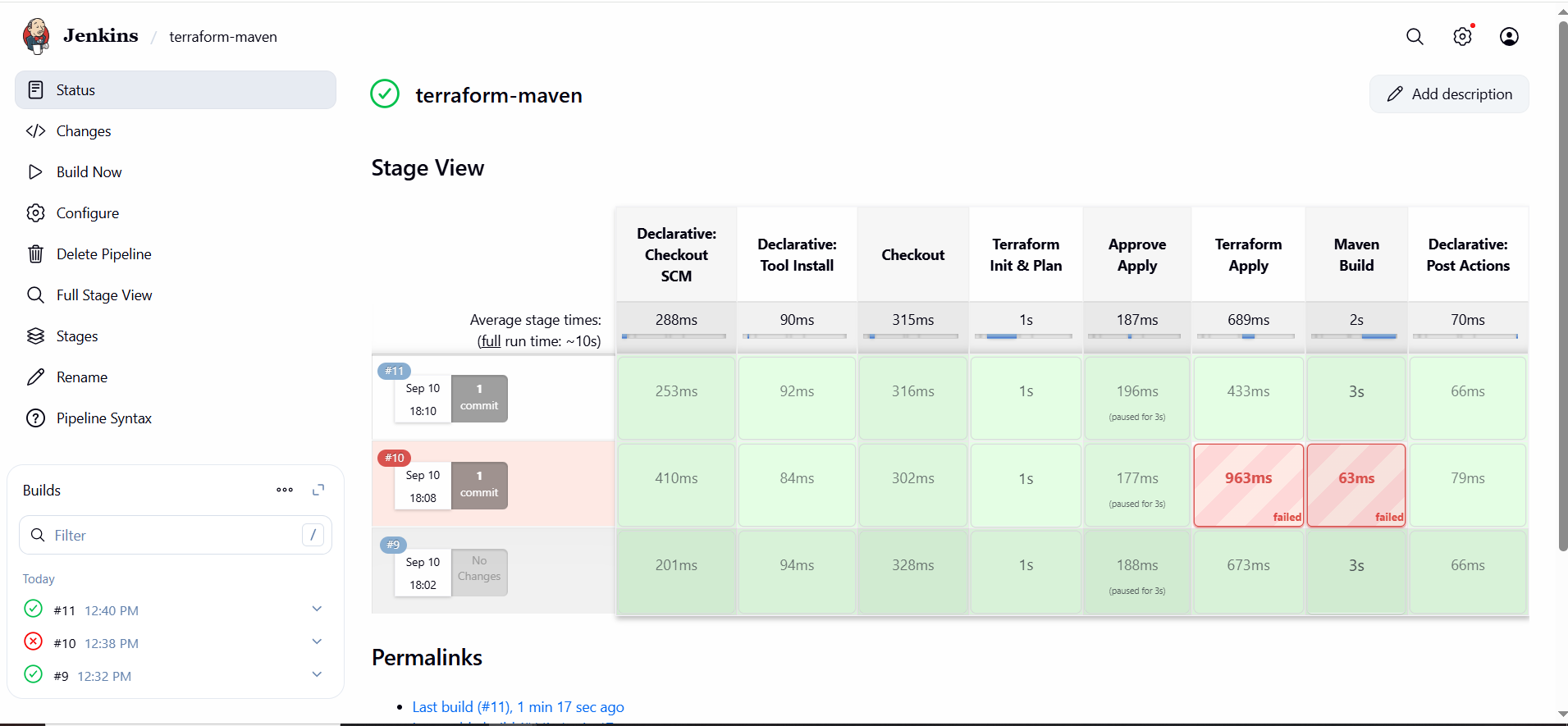
**}**

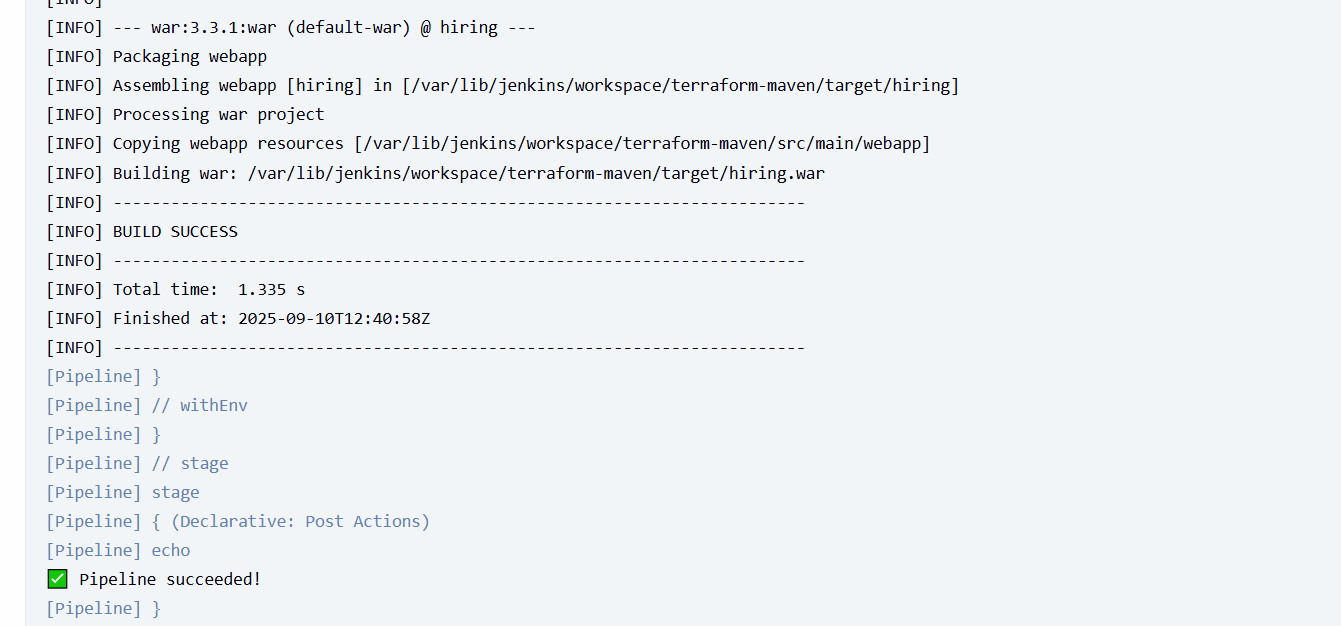
**}**

## **Step 4: Run the Job**

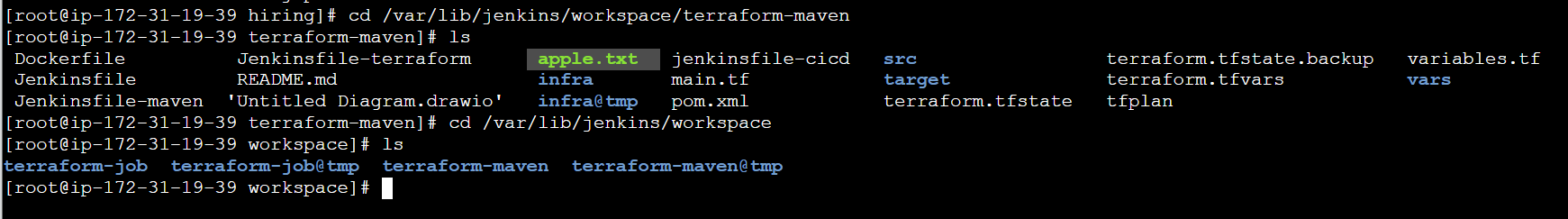
1. **Open the job in Jenkins.**
2. **Click Build Now.**
3. **Wait for:**
   * **Git clone → success**
   * **Terraform init & plan → success**
   * **Manual approval → click Proceed**
   * **Terraform apply → creates apple.txt in workspace**
   * **Maven build → creates .jar file in target/ folder.**
4. **Check Workspace:**
   * **Go to Workspace: /var/lib/jenkins/workspace/terraform-maven/**
   * **You should see apple.txt and Maven target/ artifacts.**

**BUILD**

****

****

**In JENKINS EC2**

****

**5. Use the same code and create a parameterized job in Jenkins with:**

**Stage 1: Git clone**

**Stage 2: Maven Compilation Code:** [**https://github.com/betawins/java-Working-app.git**](https://github.com/betawins/java-Working-app.git)

[**https://github.com/betawins/hiring-app**](https://github.com/betawins/hiring-app)

# **Jenkins Parameterized Pipeline Documentation**

## **Project Overview**

**This Jenkins job will:**

1. **Clone Git repository containing the Maven project.**
2. **Run Terraform to create a local file using parameters.**
3. **Compile Maven project.**
4. **Be parameterized, allowing user to enter FILENAME and CONTENT at build time.**

**Git Repo: https://github.com/betawins/hiring-app**

**Terraform Resources: local\_file to create a text file.**

## **1️⃣ Prerequisites**

1. **Jenkins installed and running.**
2. **Maven installed and configured in Jenkins Global Tool Configuration as Maven3.**
3. **Terraform installed on Jenkins agent.**
4. **Jenkins agent has permissions to write to workspace directories (avoid /root).**

## **2️⃣ Terraform Files**

### **variables.tf**

**variable "filename" {}**

**variable "content" {}**

**variable "prefix" {}**

[**main.tf**](http://main.tf)

**resource "local\_file" "my\_pet" {**

**content = var.content**

**filename = "/var/lib/jenkins/workspace/terraform-maven/${var.filename}"**

**file\_permission = "0777"**

**directory\_permission = "0777"**

**}**

**terraform.tfvars**

**filename = "fruits.txt"**

**content = "royal gala is one type of apple"**

**prefix = "Mr."**

**Note: The values in terraform.tfvars are default values. Jenkins parameters override these at build time.**

**3️⃣ Jenkinsfile-Maven (Parameterized)**

**pipeline {**

**agent any**

**parameters {**

**string(name: 'FILENAME', defaultValue: 'apple.txt', description: 'Terraform file name')**

**string(name: 'CONTENT', defaultValue: 'royal gala is one type of apple', description: 'Content of the file')**

**}**

**tools {**

**maven "Maven3"**

**}**

**environment {**

**TF\_IN\_AUTOMATION = '1'**

**}**

**stages {**

**stage('Checkout') {**

**steps {**

**git url: 'https://github.com/imrankhanmohammad257/hiring-app.git', branch: 'main'**

**sh 'ls -l' // Optional: list files**

**}**

**}**

**stage('Terraform Init & Plan') {**

**steps {**

**sh "terraform init -input=false"**

**sh "terraform validate"**

**sh "terraform plan -out=tfplan -var filename=${params.FILENAME} -var content='${params.CONTENT}'"**

**}**

**}**

**stage('Approve Apply') {**

**steps {**

**input message: 'Proceed with Terraform Apply?'**

**}**

**}**

**stage('Terraform Apply') {**

**steps {**

**sh "terraform apply -auto-approve tfplan || terraform apply -auto-approve -var filename=${params.FILENAME} -var content='${params.CONTENT}'"**

**}**

**}**

**stage('Maven Build') {**

**steps {**

**sh 'mvn -B -DskipTests clean package'**

**}**

**}**

**}**

**post {**

**success {**

**echo "✅ Pipeline succeeded!"**

**}**

**failure {**

**echo "❌ Pipeline failed. Check logs."**

**}**

**}**

**}**

## **4️⃣ Steps to Create Jenkins Parameterized Job**

### **Step 1: Create Job**

1. **Jenkins Dashboard → New Item**
2. **Enter Job Name: terraform-maven-parameterized**
3. **Select Pipeline → Click OK**

### **Step 2: Enable Parameters**

1. **In General → This project is parameterized → Check.**
2. **Add two String Parameters:**
   * **FILENAME → Default: apple.txt**
   * **CONTENT → Default: royal gala is one type of apple**

### **Step 3: Configure Pipeline**

1. **Pipeline → Definition: Pipeline script from SCM**
2. **SCM: Git**
3. **Repository URL: https://github.com/betawins/hiring-app.git**
4. **Branch: main**
5. **Script Path: Jenkinsfile-maven (or leave as default if in root)**

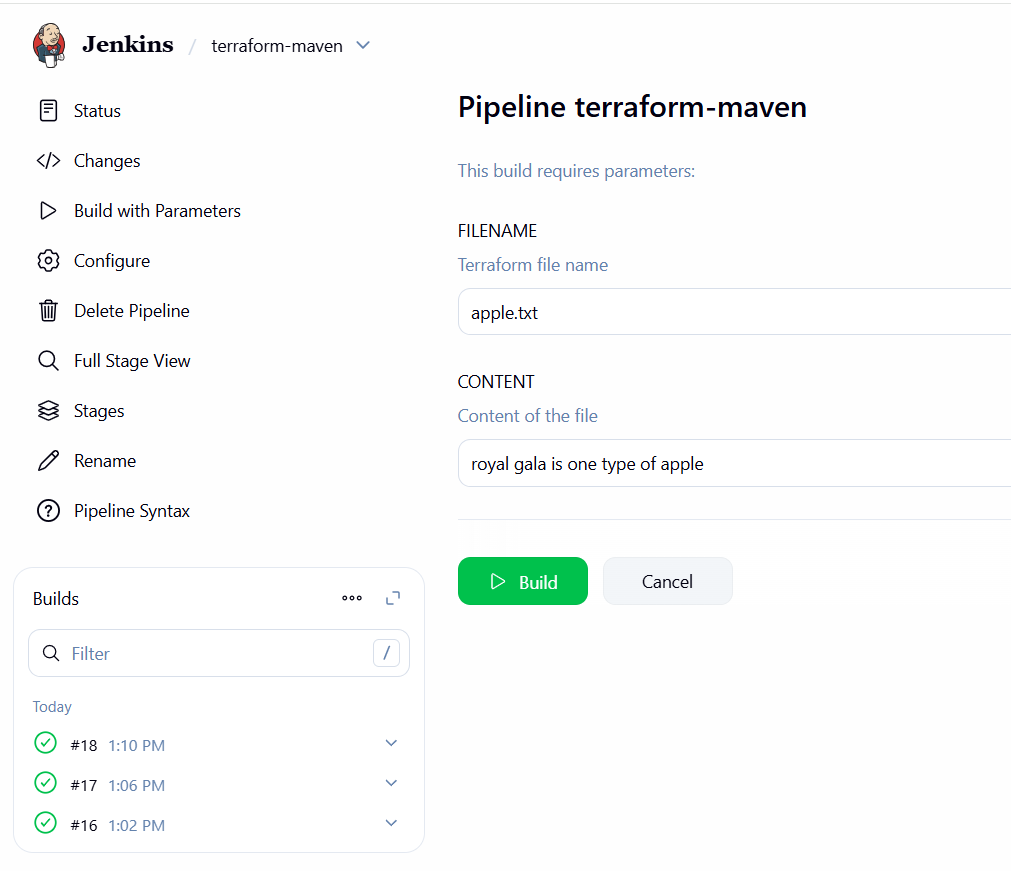
## **5️⃣ Build Flow**

1. **Click Build with Parameters.**
2. **Enter values for FILENAME and CONTENT (example: banana.txt, Banana is yellow).**
3. **Jenkins pipeline executes:  
     
    Stage 1: Checkout code from Git  
    Stage 2: Terraform Init & Plan → Creates tfplan file  
    Stage 3: Approve Apply → Manual confirmation  
    Stage 4: Terraform Apply → Creates file in workspace /var/lib/jenkins/workspace/terraform-maven/${FILENAME}  
    Stage 5: Maven Build → Packages the project**
4. **Output will show the generated file and Maven build status.**

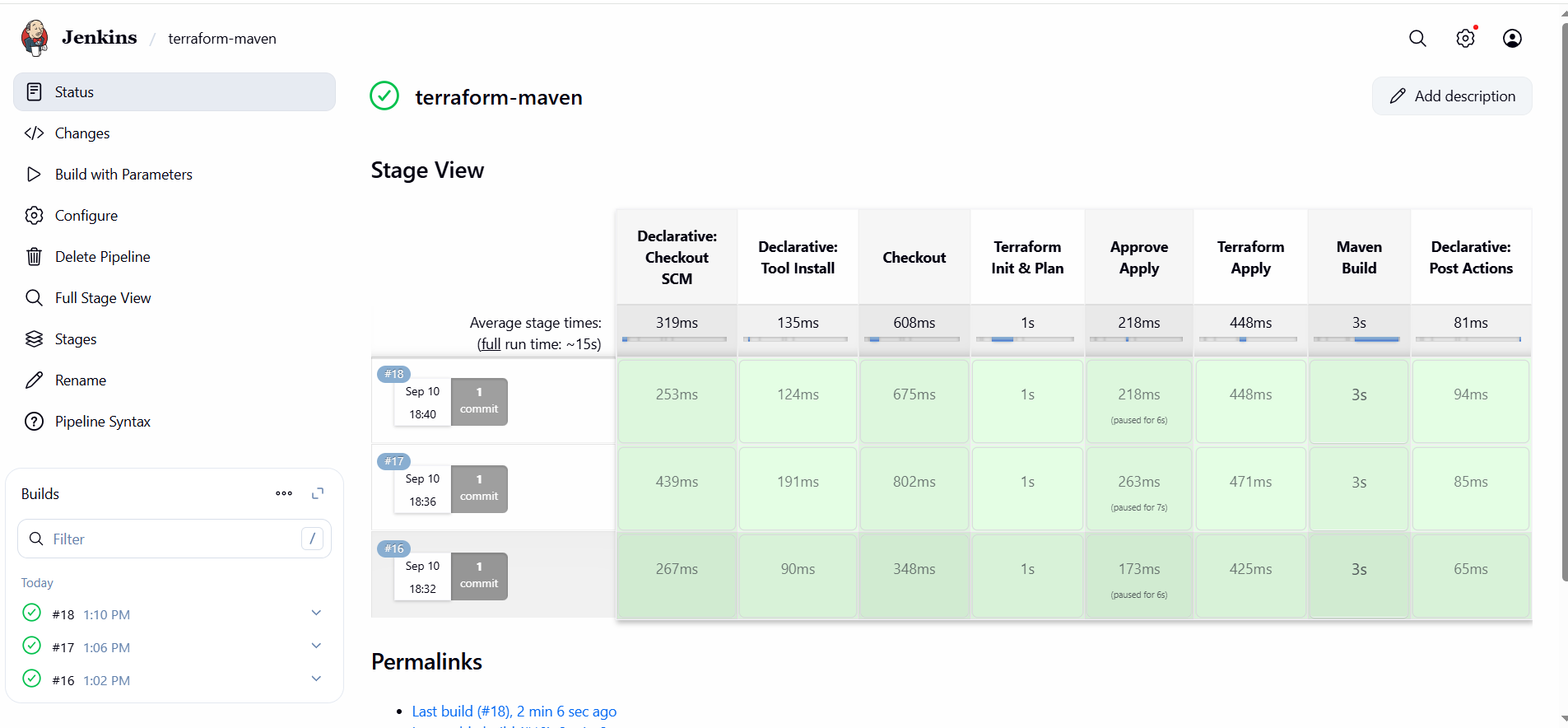
## **6️⃣ Notes / Best Practices**

* **Avoid hardcoding paths in Terraform; use ${path.module}/${var.filename} to stay inside Jenkins workspace.**
* **Ensure Jenkins agent user has write permission for Terraform workspace.**
* **Terraform plan and apply can fail if previous tfplan exists with different content; clean workspace if needed.**
* **For production pipelines, use separate workspaces per branch using terraform workspace or ${JOB\_NAME}.**

**BUILD**

****

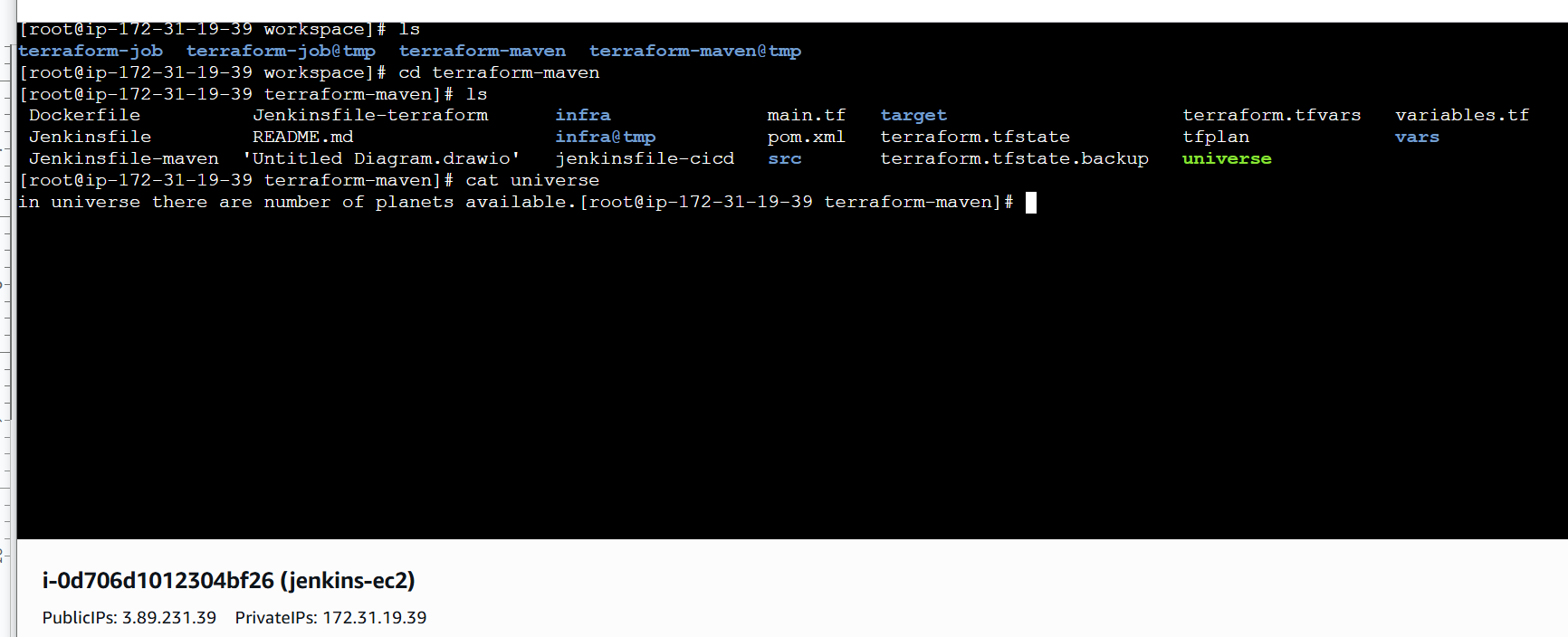
**STAGE VIEW**

****

**CONSOLE OUTPUT**

****

**JENKINS EC2**

****

**6. What are the global variables in Jenkins?**

**In Jenkins, global variables are pre-defined environment variables or objects that you can access in your pipeline scripts (Declarative or Scripted) without defining them yourself. They provide context about the build, job, agent, and environment, and are especially useful for making pipelines dynamic and portable.**

**Here’s a detailed breakdown:**

## **1️⃣ Built-in Global Environment Variables**

**These are automatically available in every pipeline. Examples:**

| **Variable** | **Description** |
| --- | --- |
| **BUILD\_NUMBER** | **Current build number of the job.** |
| **BUILD\_ID** | **The unique ID for the current build.** |
| **BUILD\_DISPLAY\_NAME** | **The display name of the build (often #<BUILD\_NUMBER>).** |
| **JOB\_NAME** | **Name of the job (folder/job).** |
| **JOB\_BASE\_NAME** | **Base name of the job (without folder path).** |
| **BUILD\_TAG** | **Tag for the build, like jenkins-${JOB\_NAME}-${BUILD\_NUMBER}.** |
| **EXECUTOR\_NUMBER** | **The executor number of the agent running the build.** |
| **NODE\_NAME** | **Name of the node/agent running the job (master if on master).** |
| **NODE\_LABELS** | **Labels assigned to the node/agent.** |
| **WORKSPACE** | **Absolute path of the workspace for the current job.** |
| **JENKINS\_URL** | **URL of the Jenkins server.** |
| **JOB\_URL** | **URL of the job.** |
| **BUILD\_URL** | **URL for the current build.** |
| **GIT\_COMMIT** | **Current Git commit SHA (if using Git plugin).** |
| **GIT\_BRANCH** | **Current Git branch name (if using Git plugin).** |

## **2️⃣ Pipeline Global Variables**

**Jenkins Pipeline Plugin defines some objects you can use globally in your pipeline script:**

| **Variable** | **Type / Usage** | **Description** |
| --- | --- | --- |
| **env** | **Map** | **Access environment variables: env.BUILD\_NUMBER, env.WORKSPACE.** |
| **params** | **Map** | **Access parameters for parameterized jobs: params.FILENAME.** |
| **currentBuild** | **Object** | **Provides info about current build: currentBuild.result, currentBuild.fullDisplayName.** |
| **scm** | **Object** | **Represents SCM configuration (used in checkout scm).** |
| **steps** | **Object** | **Provides access to Jenkins steps: steps.sh(), steps.echo().** |
| **tool** | **Function** | **Get tool installation path: tool 'Maven3'.** |
| **pwd()** | **Function** | **Returns the current directory of the build workspace.** |

**3️⃣ Example Usage**

**pipeline {**

**agent any**

**parameters {**

**string(name: 'FILENAME', defaultValue: 'apple.txt', description: 'Terraform file name')**

**}**

**stages {**

**stage('Print Info') {**

**steps {**

**echo "Job Name: ${env.JOB\_NAME}"**

**echo "Build Number: ${env.BUILD\_NUMBER}"**

**echo "Workspace Path: ${env.WORKSPACE}"**

**echo "Parameter FILENAME: ${params.FILENAME}"**

**}**

**}**

**stage('Use Maven') {**

**steps {**

**sh "${tool 'Maven3'}/bin/mvn clean package"**

**}**

**}**

**}**

**}**

### **4️⃣ Key Points**

1. **env is the most commonly used global variable map.**
2. **params is only available in parameterized jobs.**
3. **currentBuild gives access to status, duration, and other runtime info.**
4. **Global variables make your pipeline dynamic, reusable, and less hardcoded.**