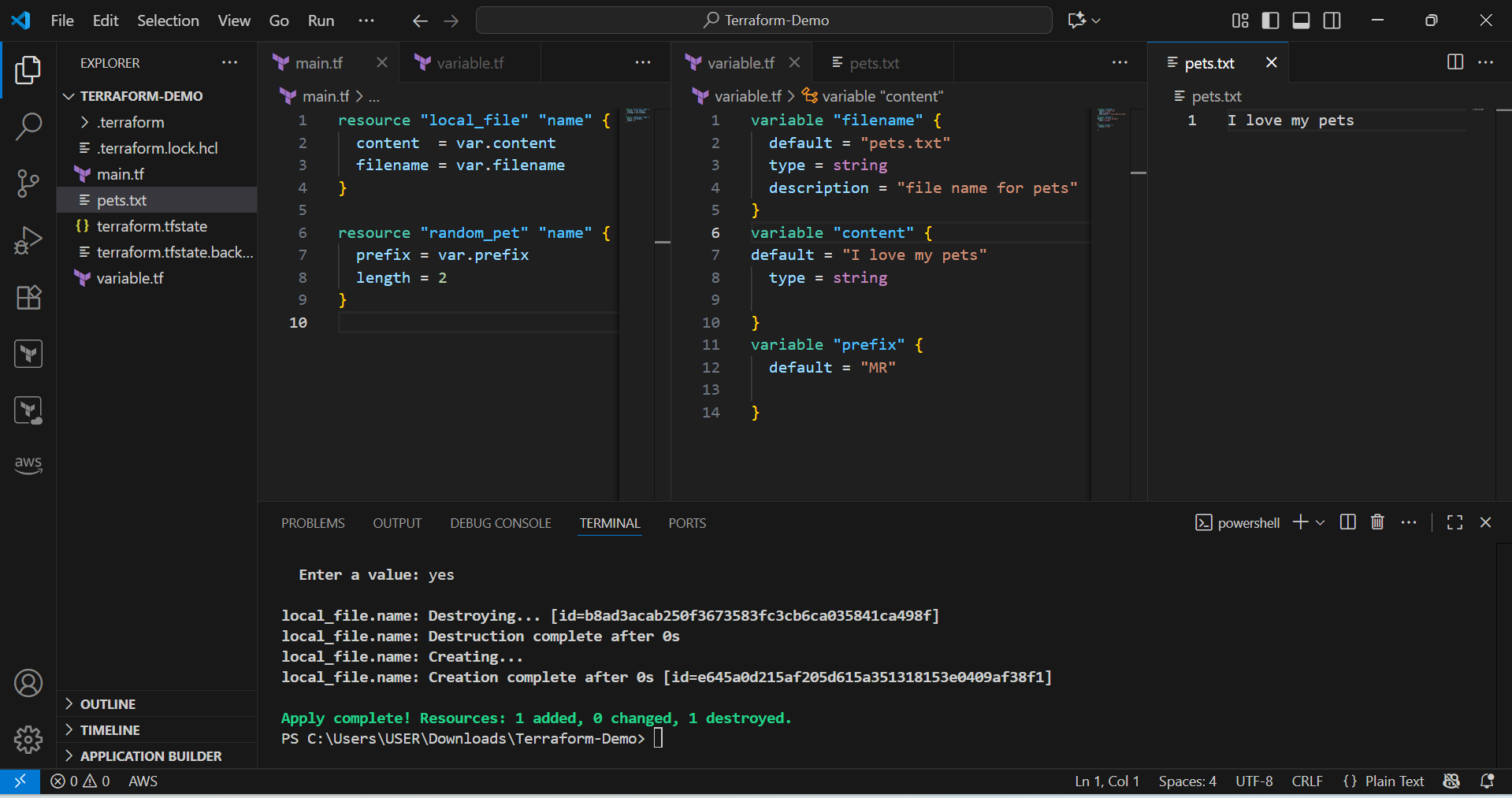
**Terraform -03**

1. **Watch the Terraform-03 video.**
2. **Execute the script shown in the video.**

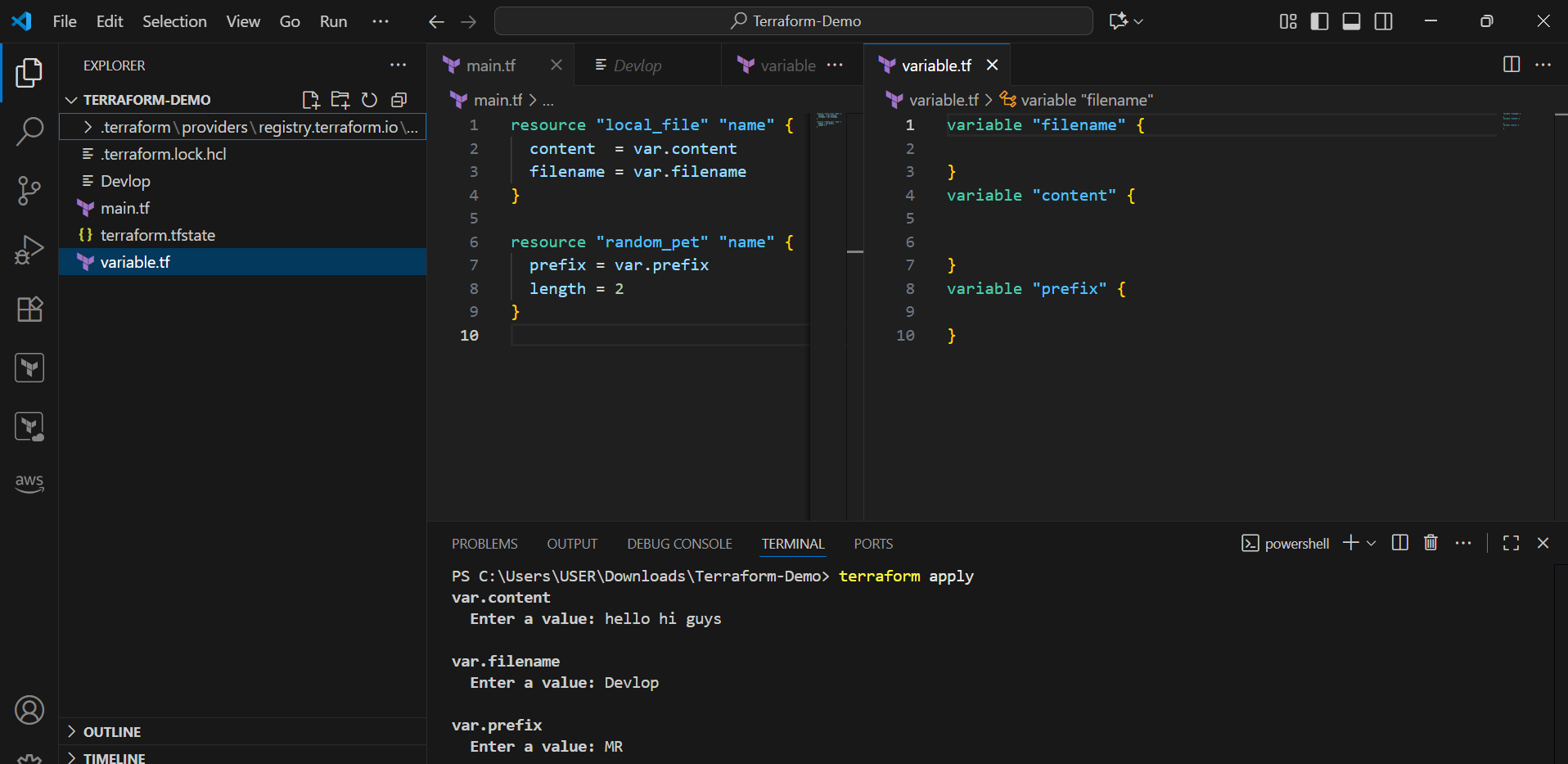
Using of variables:

==================

1) By using varibles.tf file



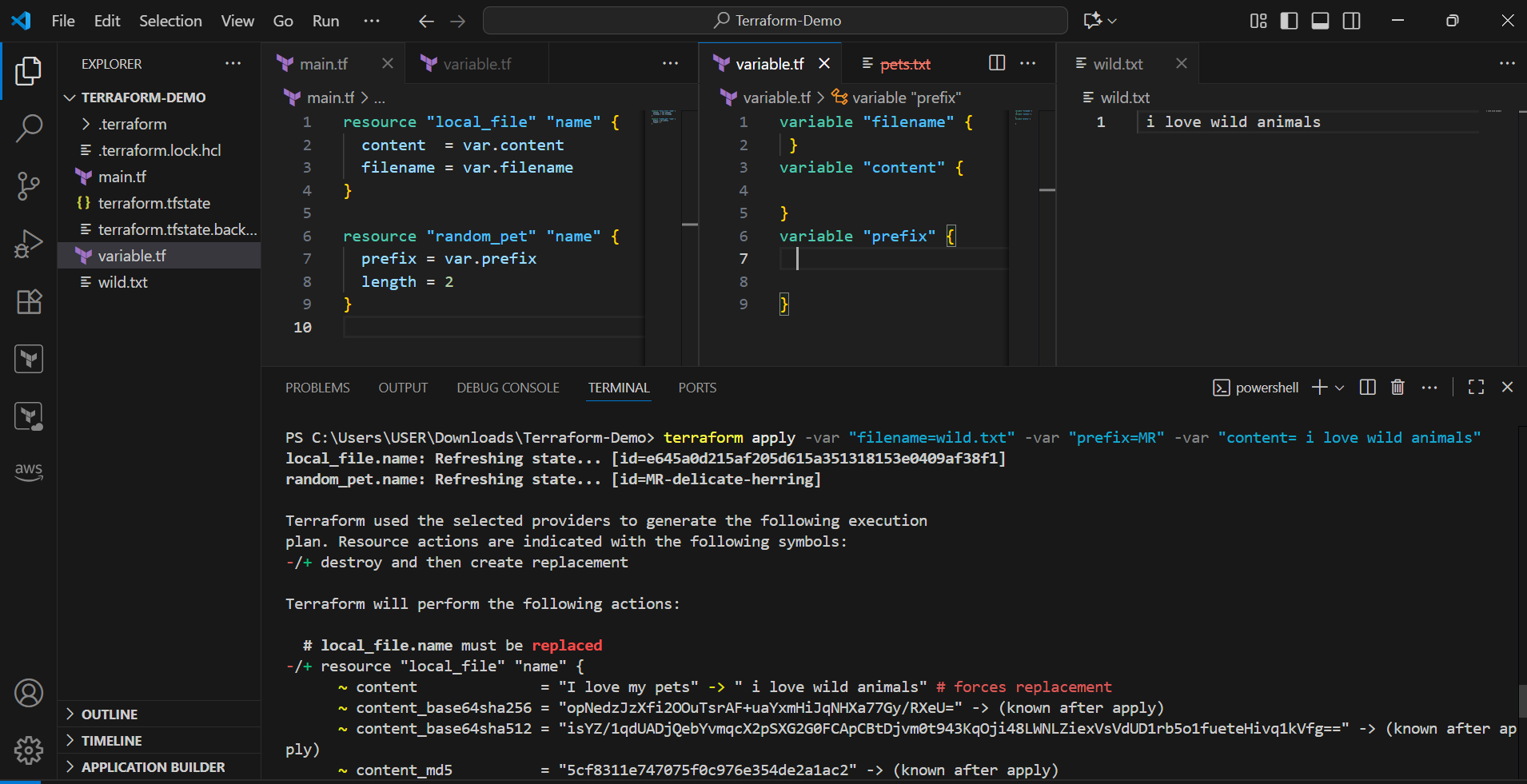
2) By using interactive mode (This will get activated if we dont pass default value in variable.tf file)

****

3) Command line flags

-->

terraform apply -var "filename=wild.txt" -var "prefix=MR" -var "content= i love wild animals"



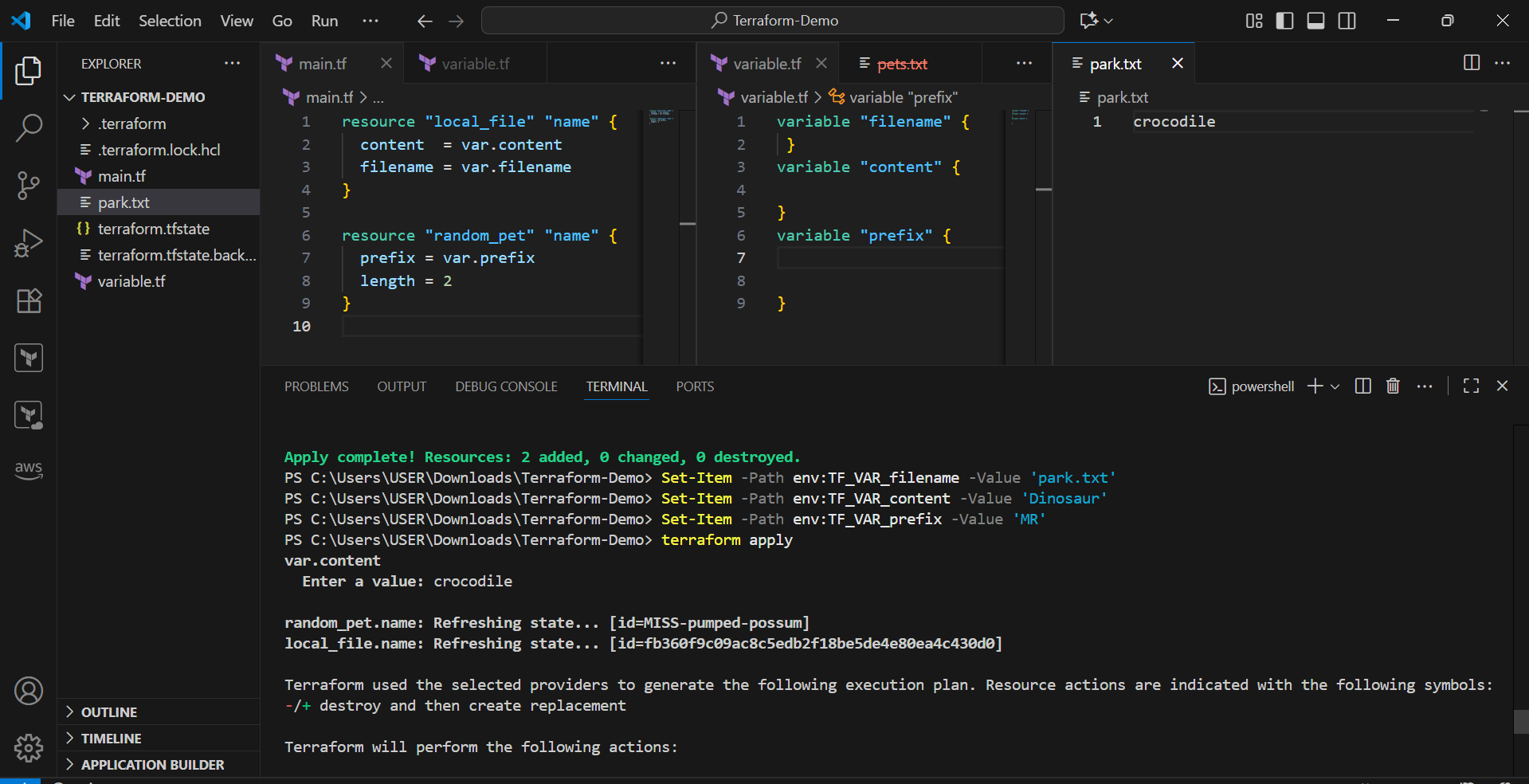
4) Environment variables

--> Set-Item -Path env:TF TF\_VAR\_filename="/root.pets.txt"

--> Set-Item -Path env:TF TF\_VAR\_prefix= "MR"

--> Set-Item -Path env:TF\_VAR\_filename -Value 'wild.txt'

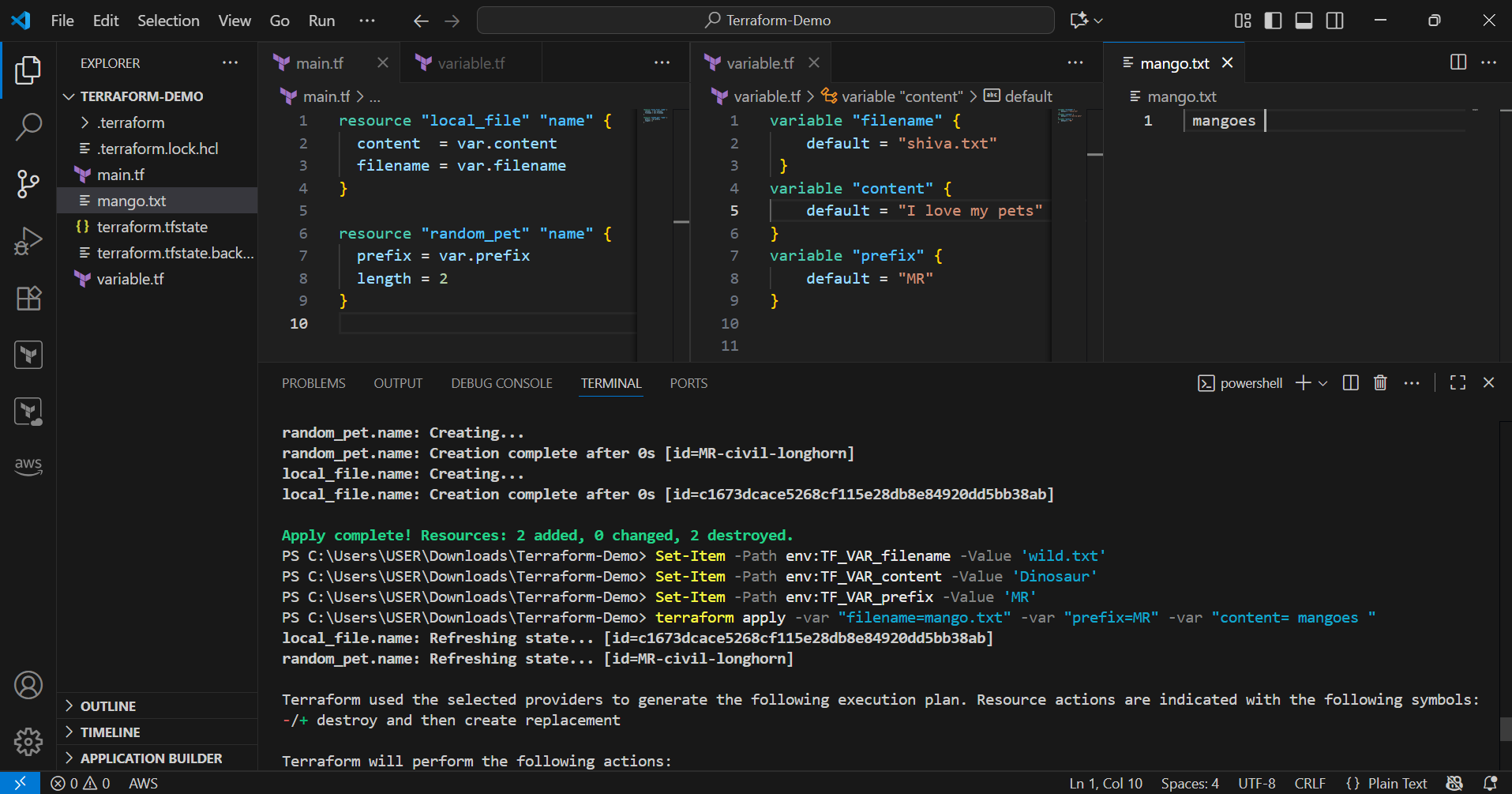
terraform apply



Varible definition precedence:

=============================

If we use multiple ways to define varibles for the same file then terraform uses varible definition precedence

****

Precedence order:

================

in the above example we have passed all the possible varibles,which will terraform laod first and which will override ?

Order Option

1 Environment variables

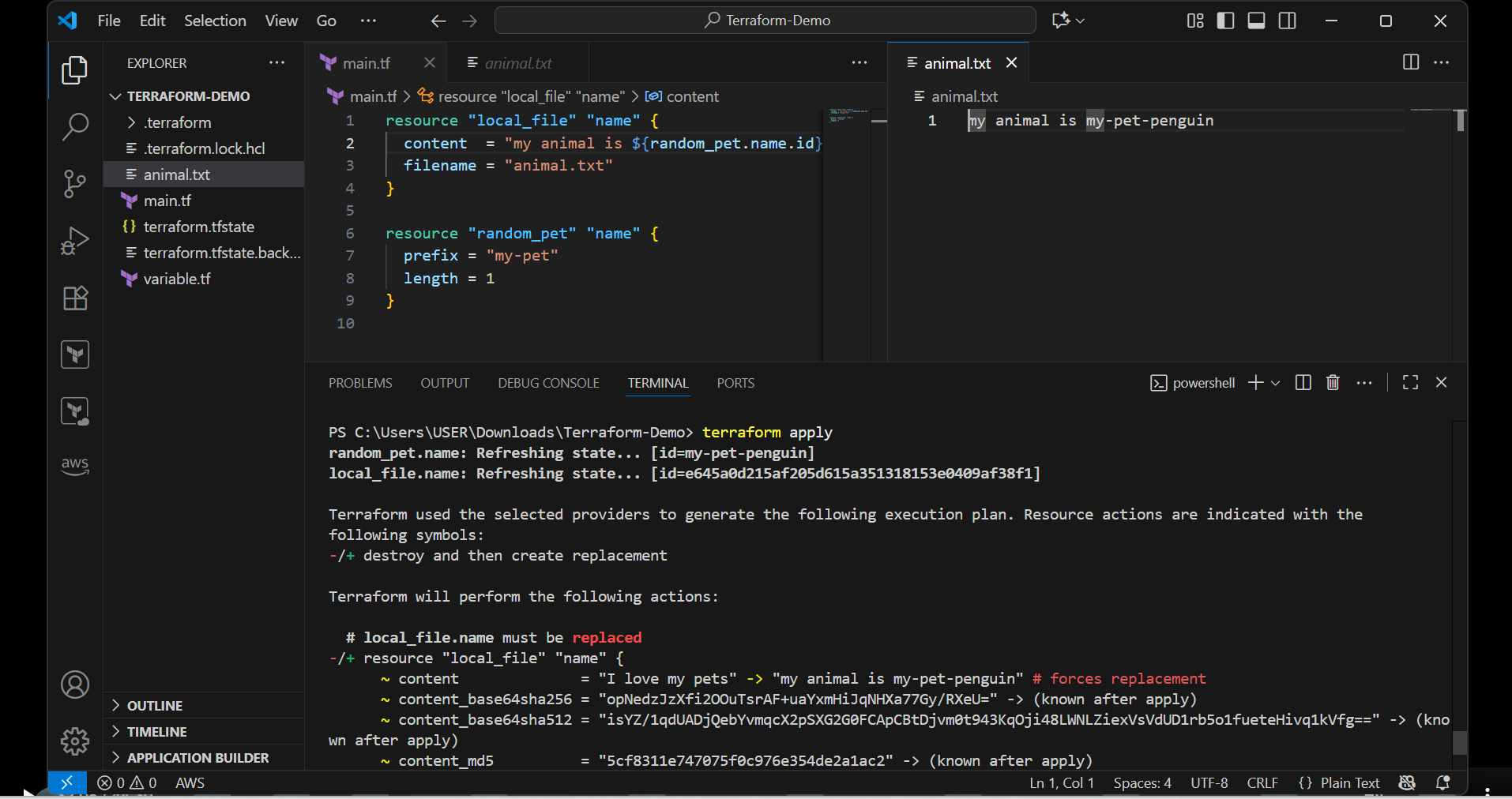
2 Terraform.tfvars

3 \*.auto.tfvars(alphabetical order)

4 -var or -var-file (Command line flags)

When we execute terraform apply it will create random id with pet name,

now i want to add this pet name in my content file (using output of one resource as input for another resource).

****

Resource Dependencies:

=====================

From the above main.tf we have local\_file which is dependent on random\_pet resource.

We are not mentioning that anywhere but still terraform will figure that out, we call them as implicit dependecy.

Still we can define the dependency explicit by ourself using "depends\_on" module

Output variables:

=================

These are used to display the output of the resources.

Ex:

resource "random\_pet" "mypet" {

prefix = "MR"

separator = "."

length = "1"

}

output my-pet {

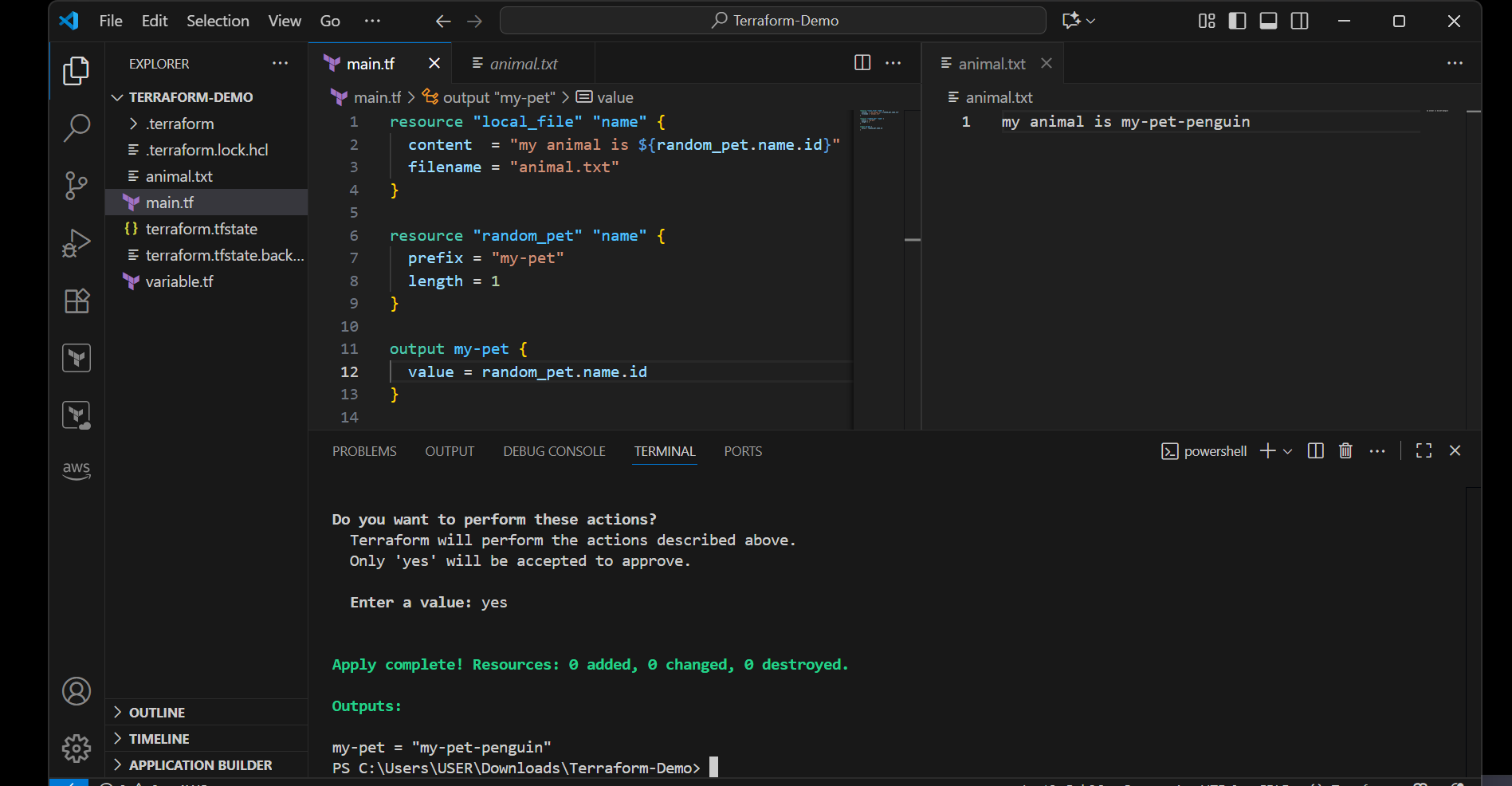
value = random\_pet.my-pet.id

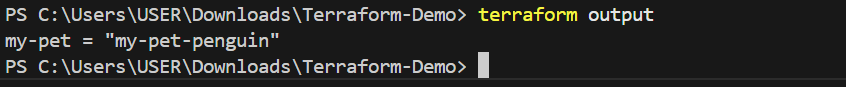
description = optional name

}

when we use terraform apply we can see the id as output.

we can use terraform output command to see the output of the resource.

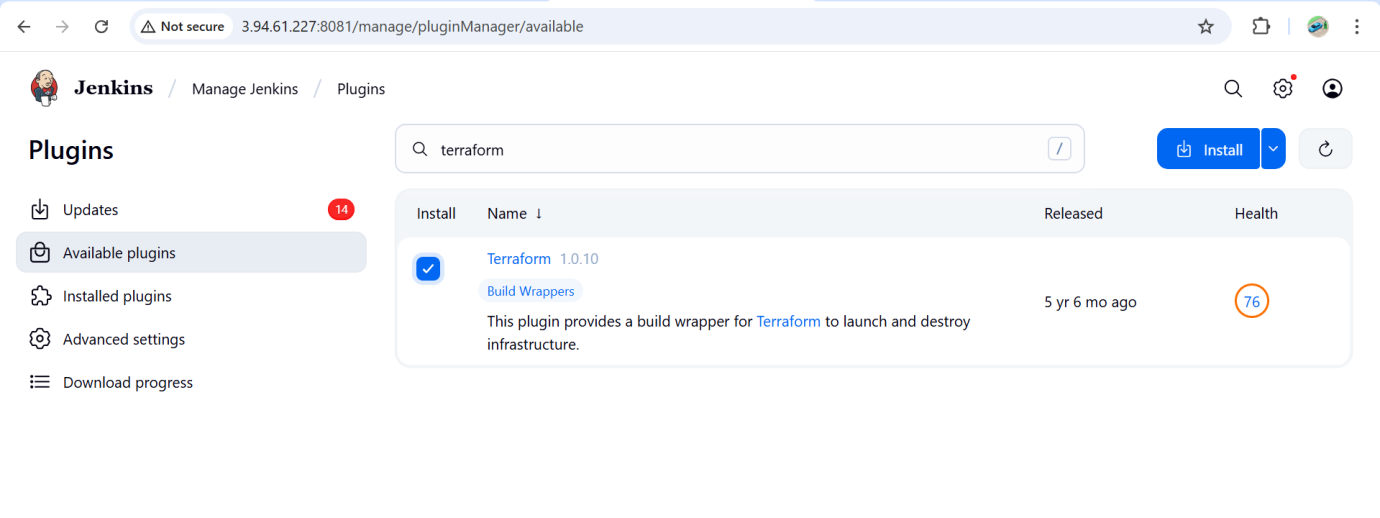


****

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

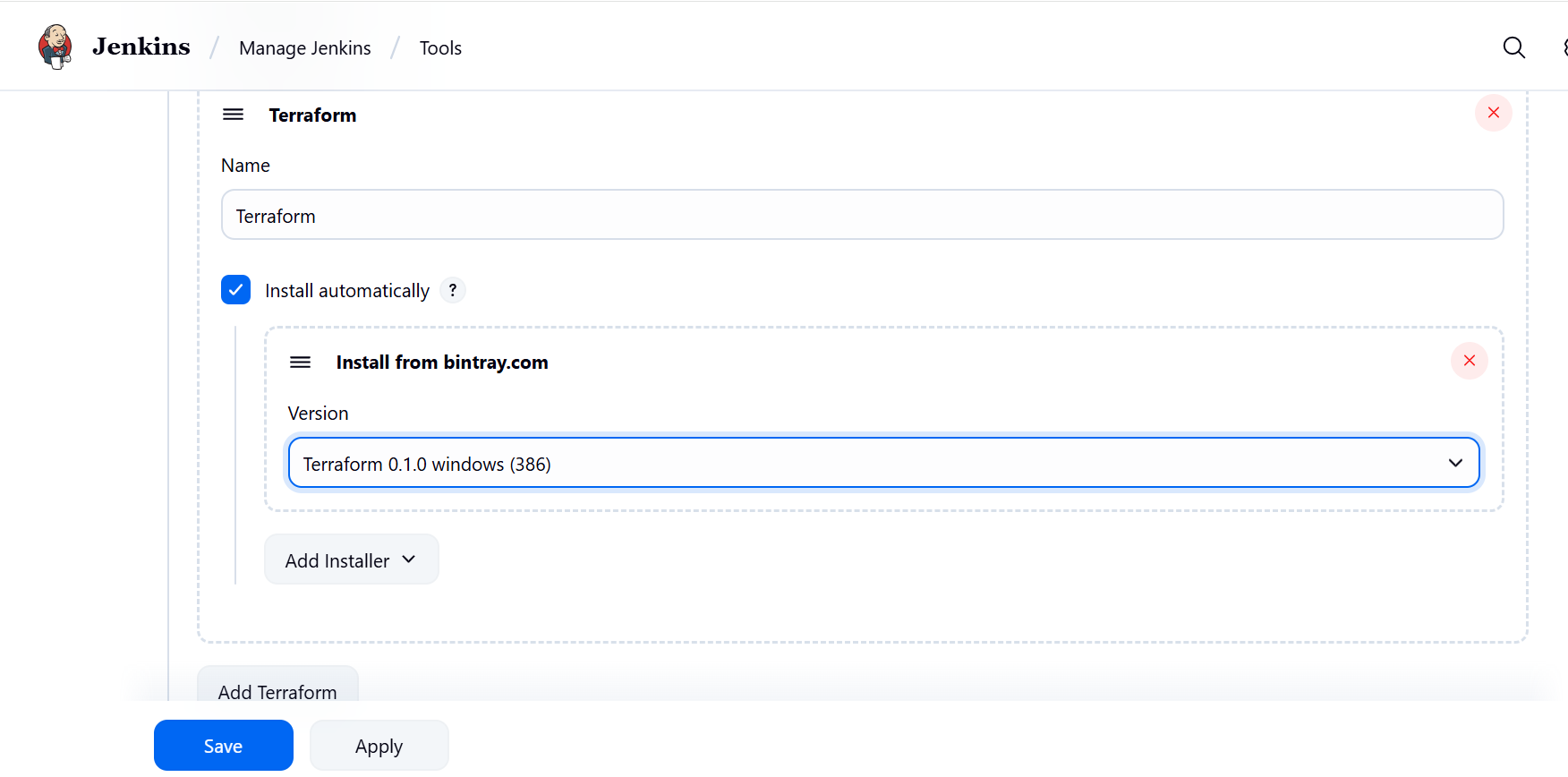
**Step 1: Install the Terraform Plugin in Jenkins**

1. Go to **Jenkins Dashboard → Manage Jenkins → Plugins → Available Plugins**.
2. Search for **Terraform**.
3. Install **Terraform Plugin**
4. Restart Jenkins if required.



## ****Step 2: Configure Terraform in Jenkins****

1. Go to **Manage Jenkins → Tools**.
2. Find **Terraform installations**.
3. Click **Add Terraform**:
   * Name: terraform
   * Install automatically: ✅ (Jenkins will download it, or you can specify the path if already installed).
4. Save.

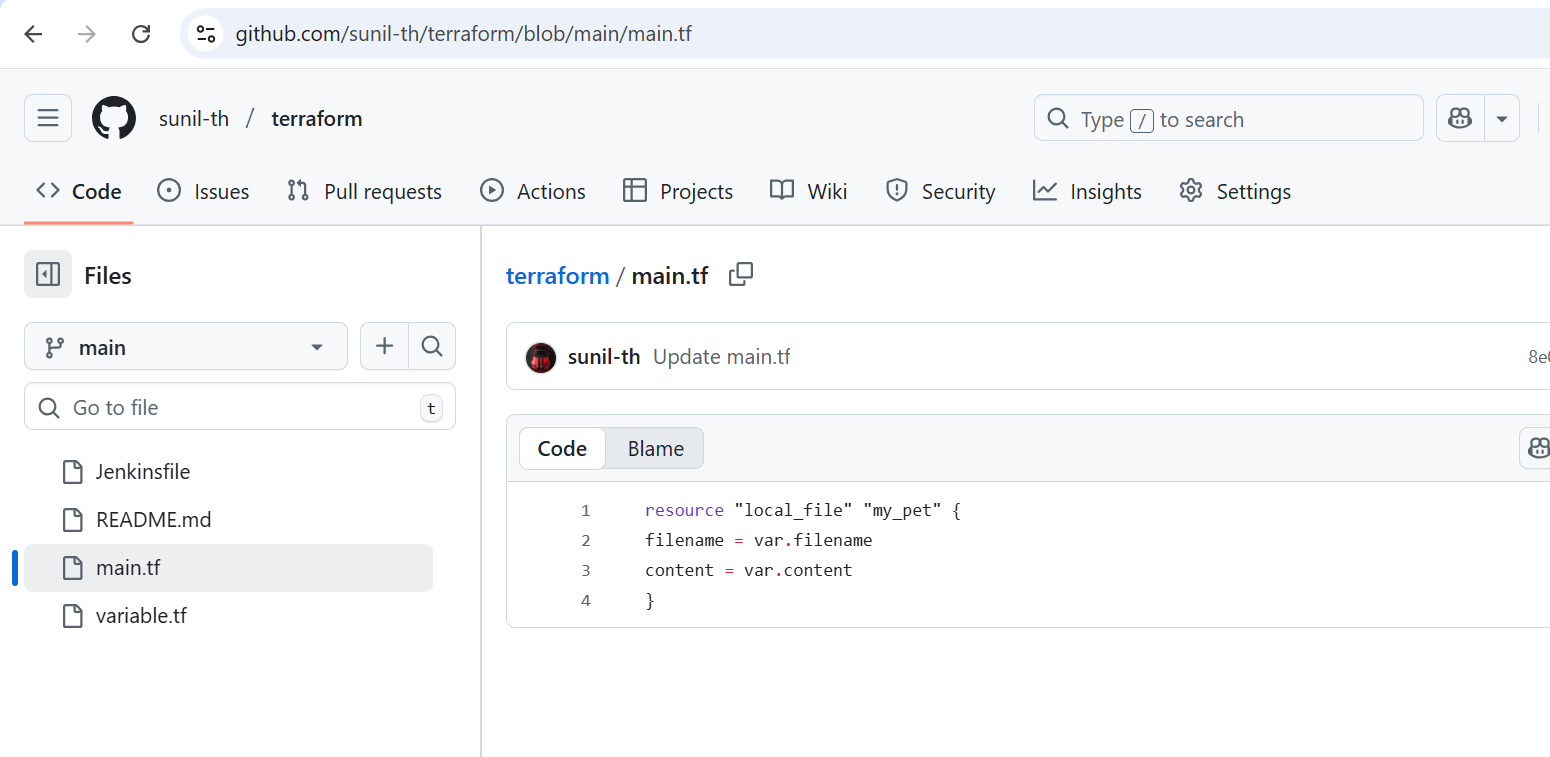


## ****Step 3: Create a Jenkins Job (Pipeline or Freestyle)****

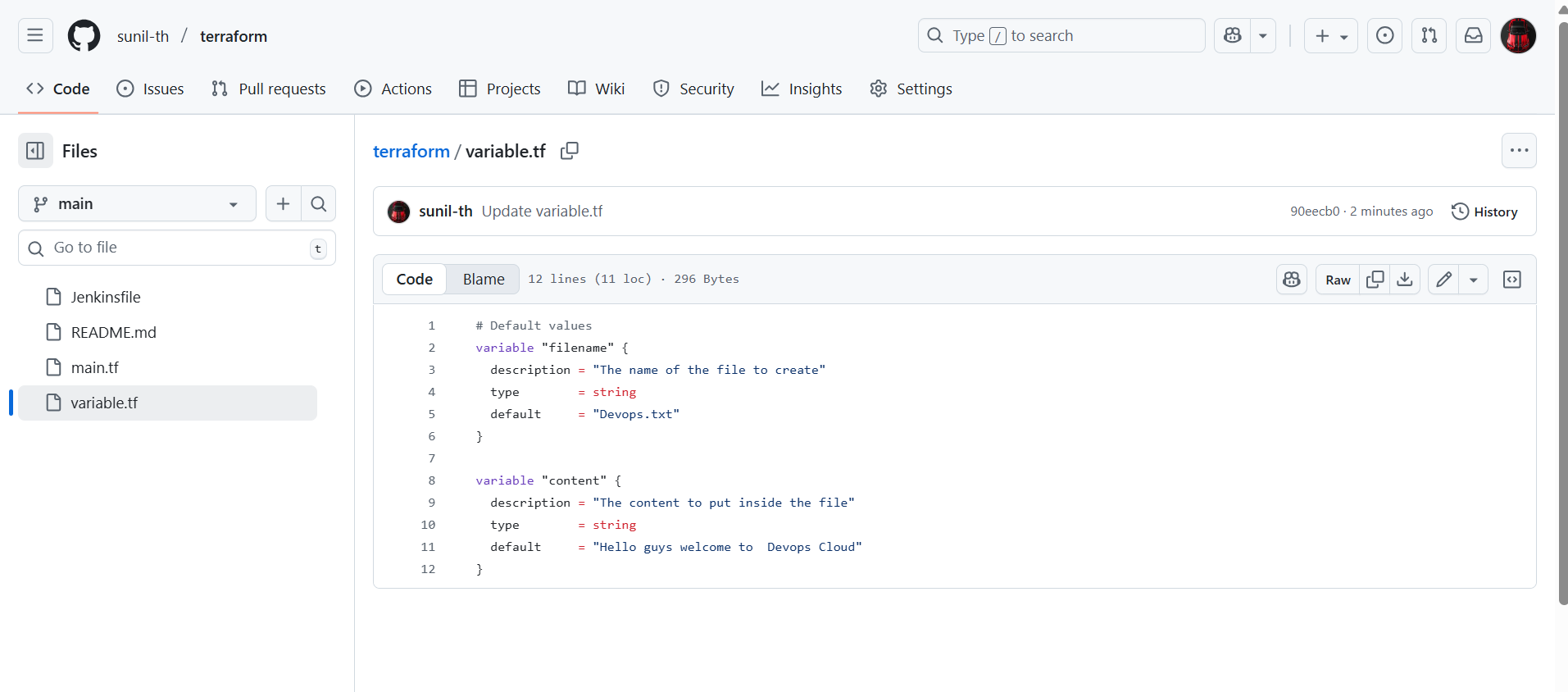
You can use either **Freestyle** or **Pipeline**.

**In Git hub add these files**

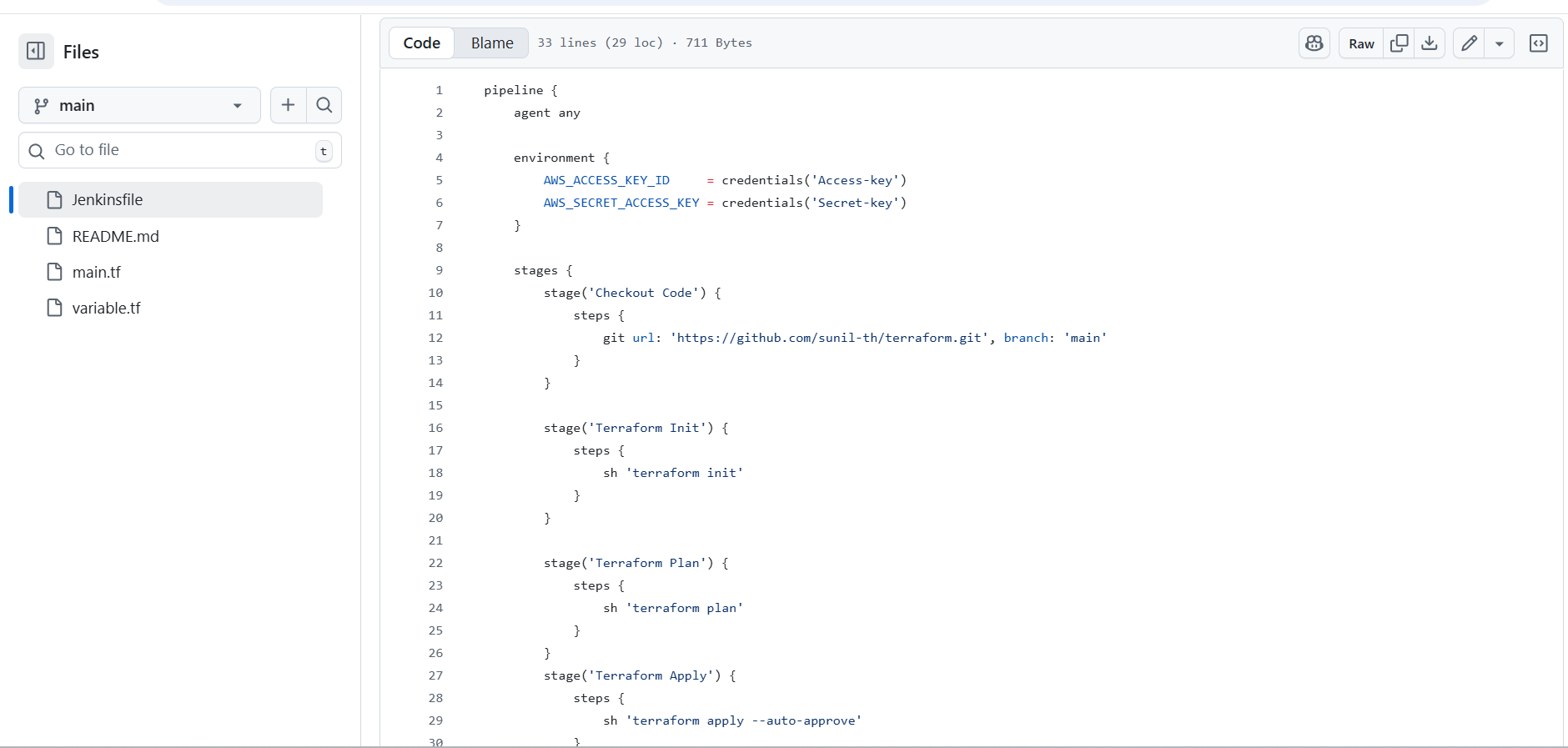
**Main.tf**

****

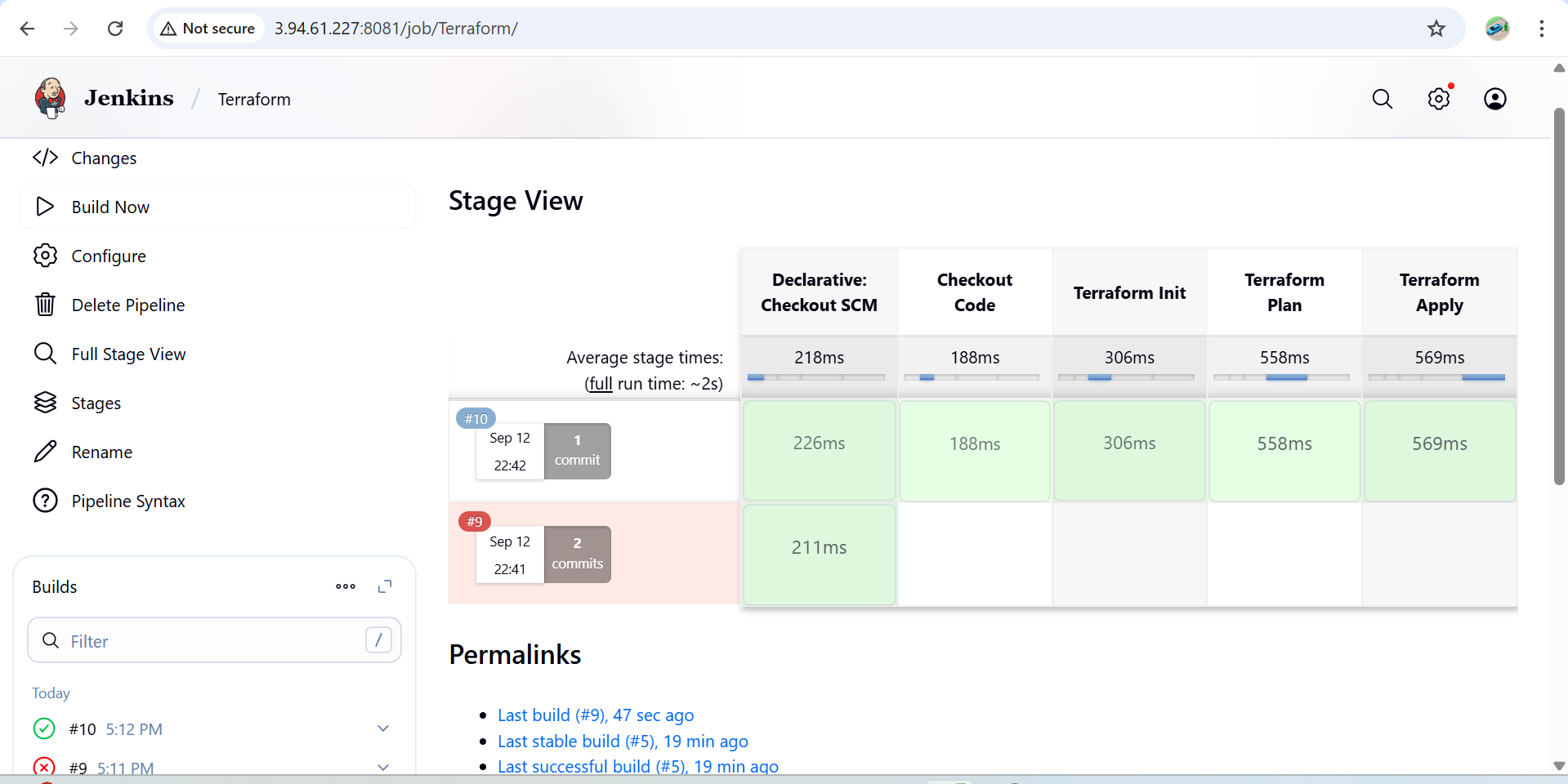
**Variable.tf**

****

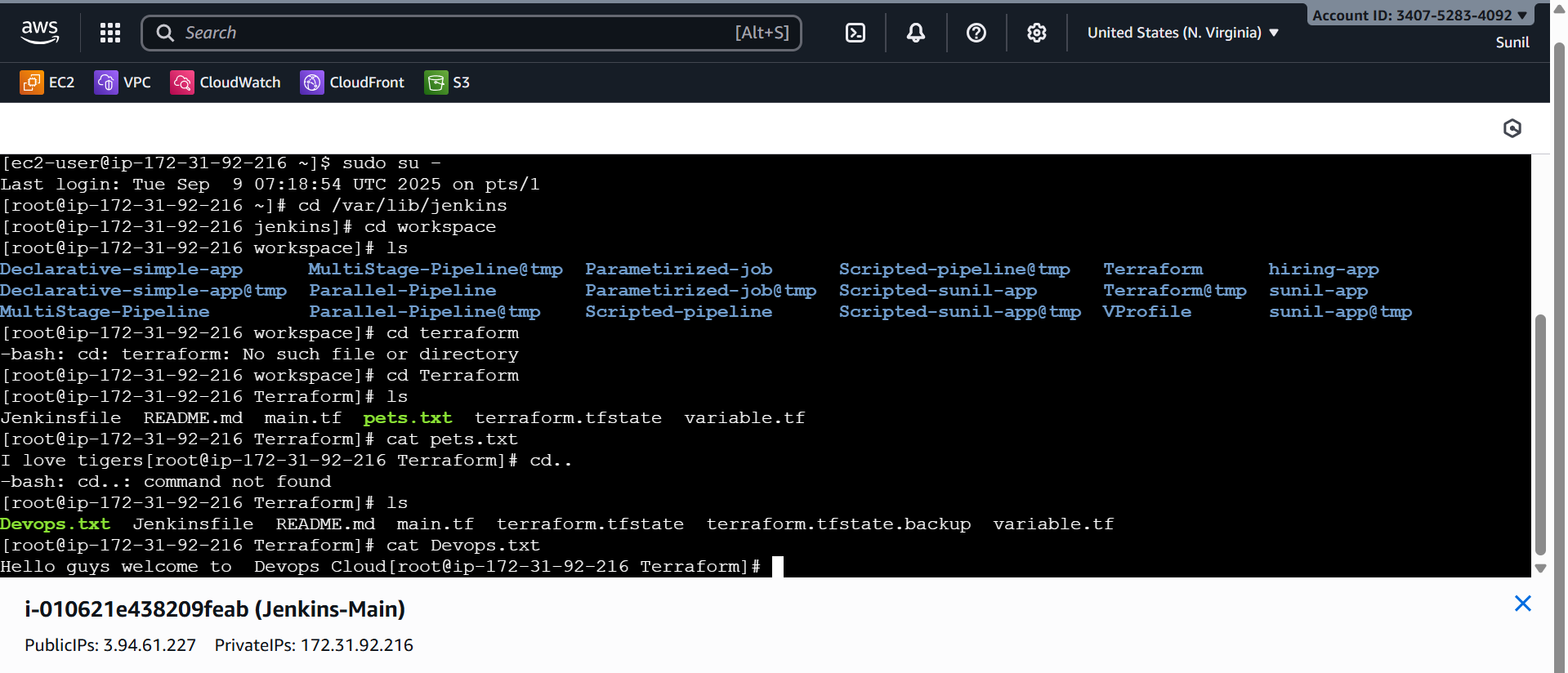
**Jenkinfile**

****

**Create a jenkinJob and build**

****

Check the Jenkins main workspace and cat the file inside the job you can view the content in the file



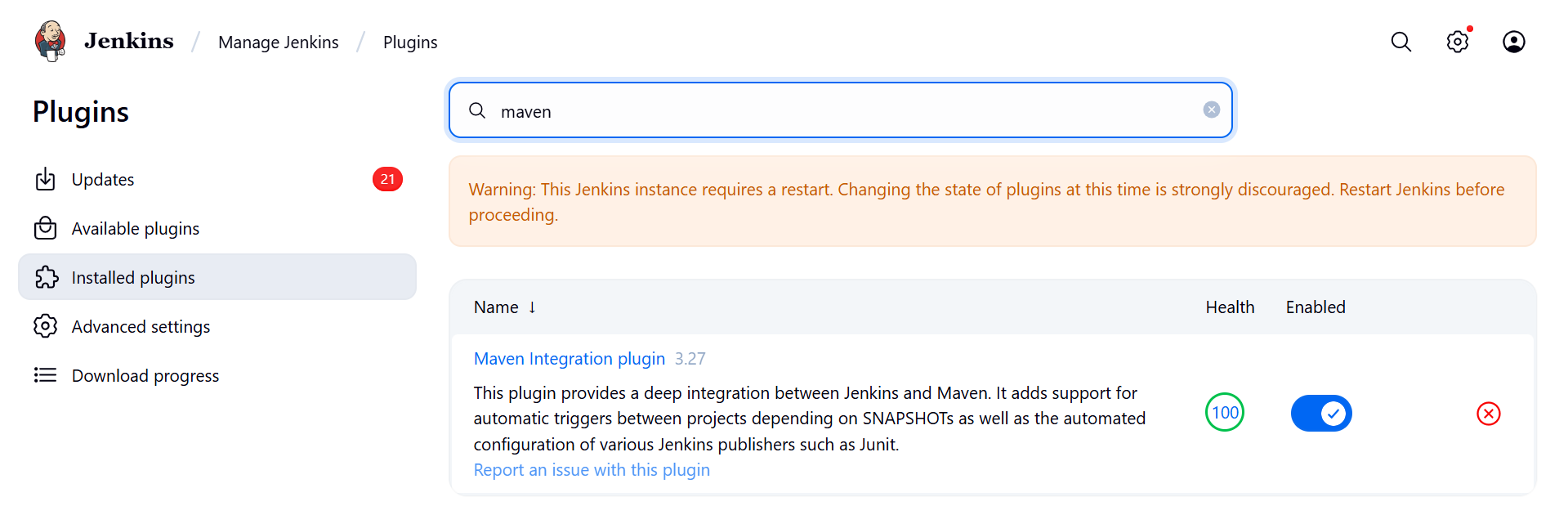
1. **Create one Jenkins job using Maven Project for the code below with two stages:**

**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)

**Step-1**

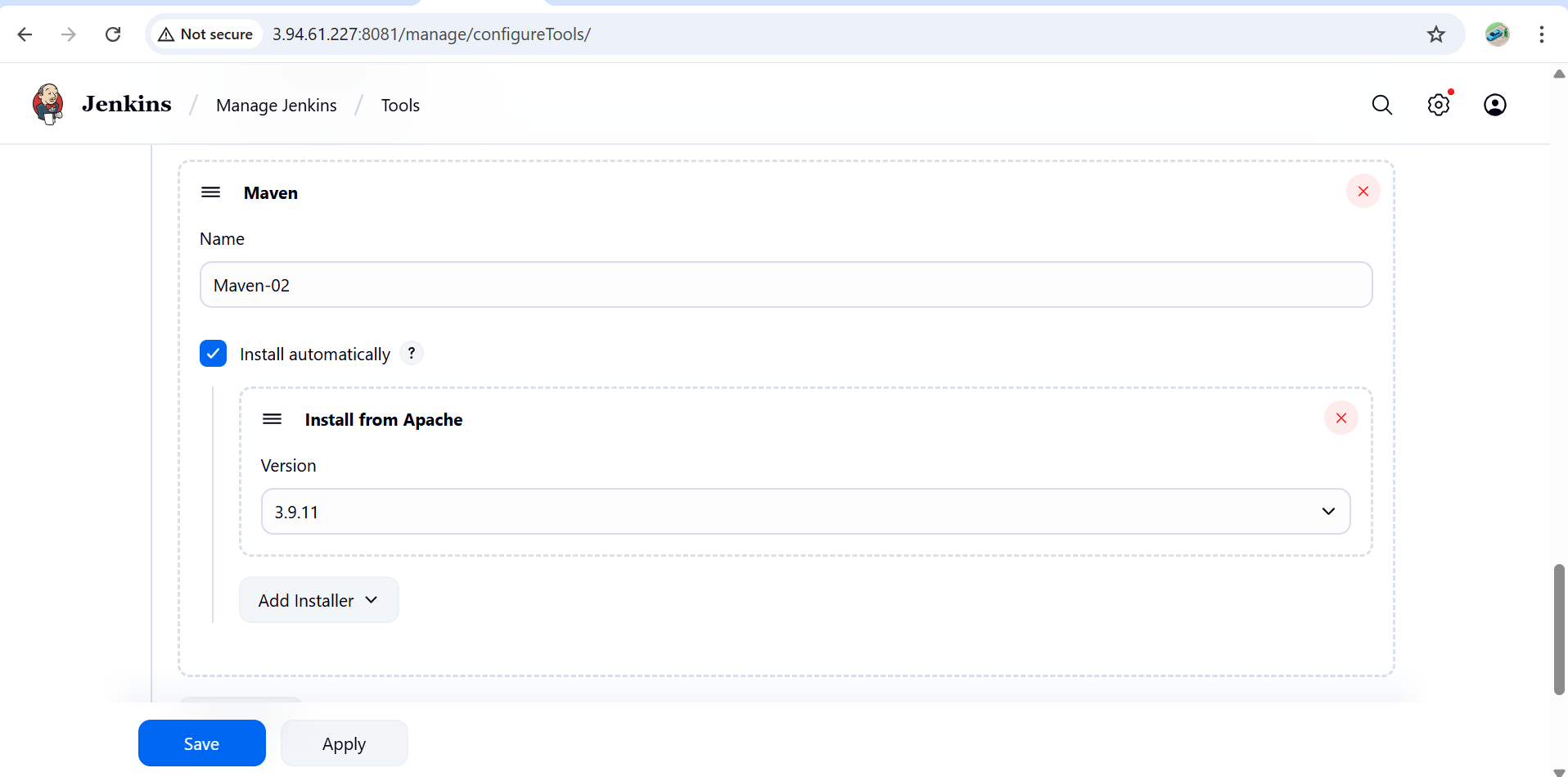
Install the **Maven Integration plugin** from **Manage Jenkins → Plugins**.

****

**Step-2**

Go to **Manage Jenkins → Tool Configuration → Maven**.

Add a Maven installation → check **"Install automatically"** → Jenkins downloads Maven when needed.

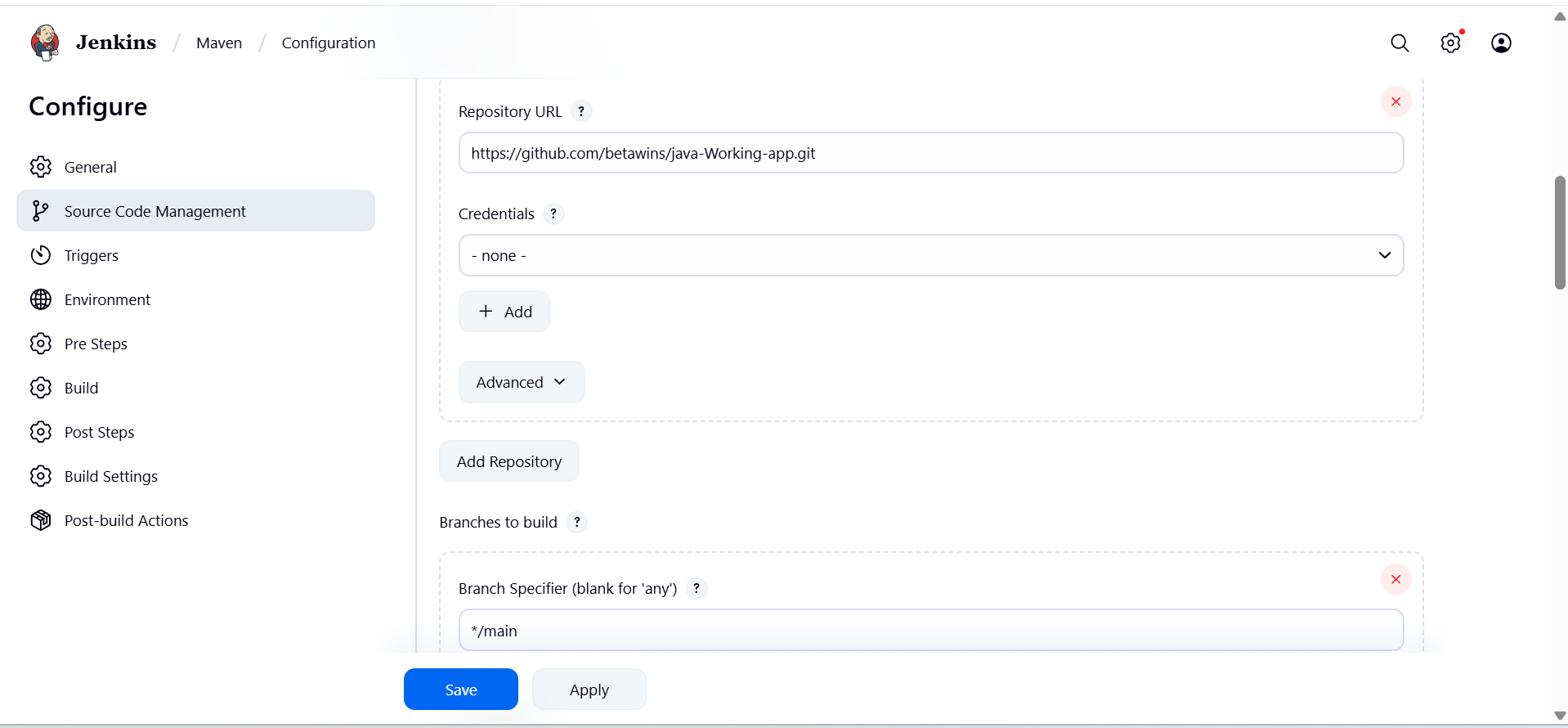
****

**Step-3**

Go to **Jenkins Dashboard → New Item → Maven Project**.

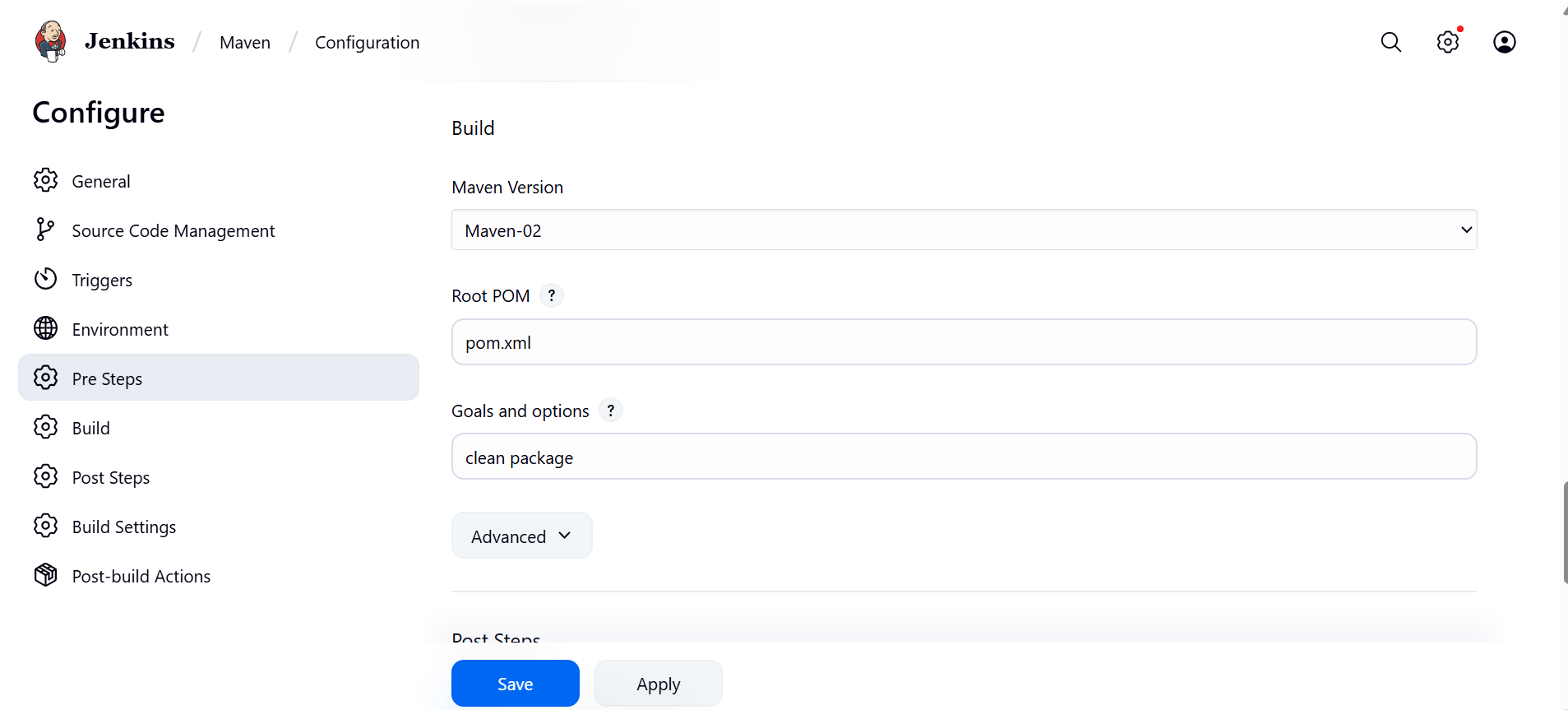
Enter job name → select **Maven project**.

In **Source Code Management**, select **Git** and add your repo URL ([**https://github.com/betawins/java-Working-app.git**](https://github.com/betawins/java-Working-app.git)).

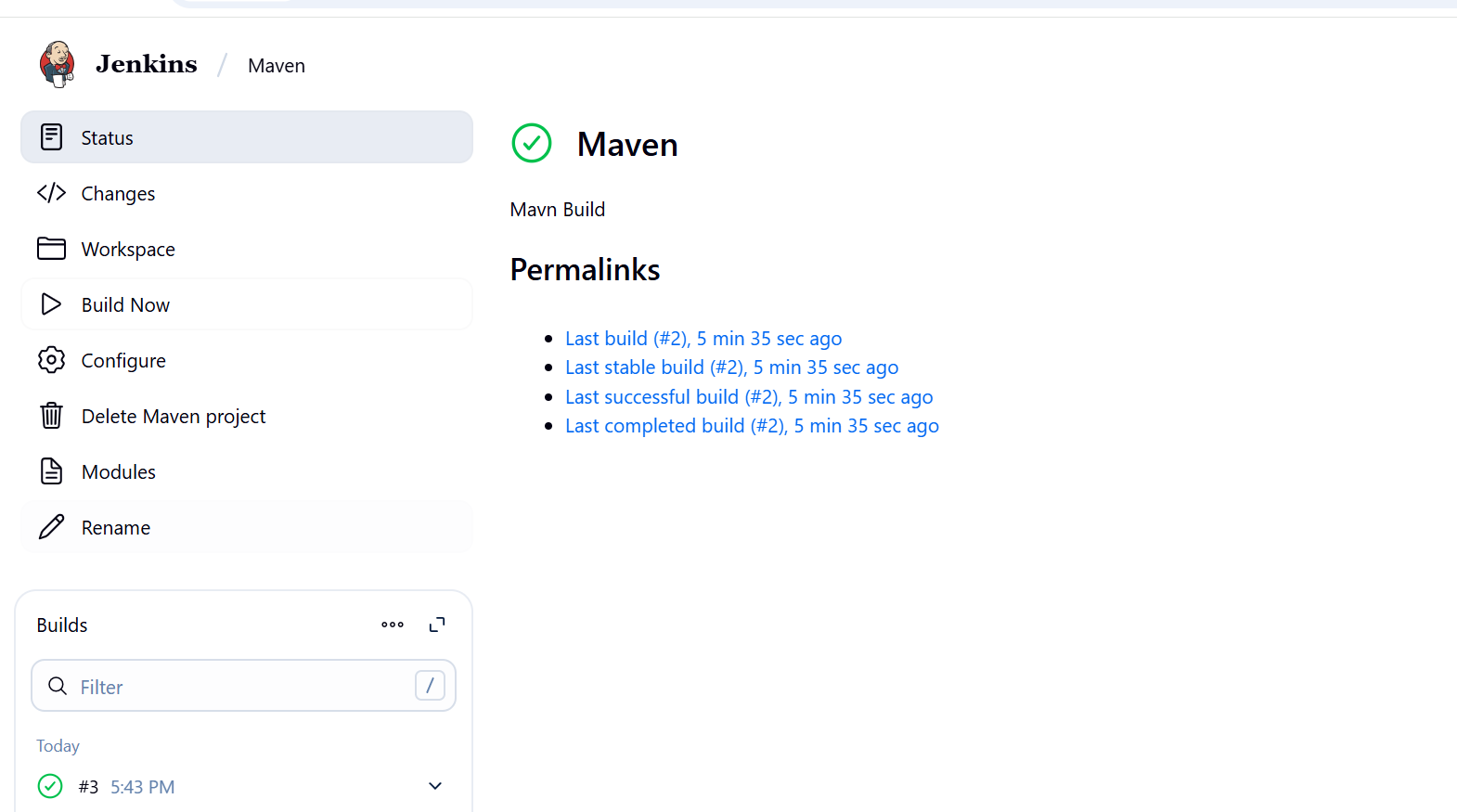
****

In **Build**, set:

* **Goals and options:** clean package
* Jenkins will run Maven using the installed Maven tool.

****

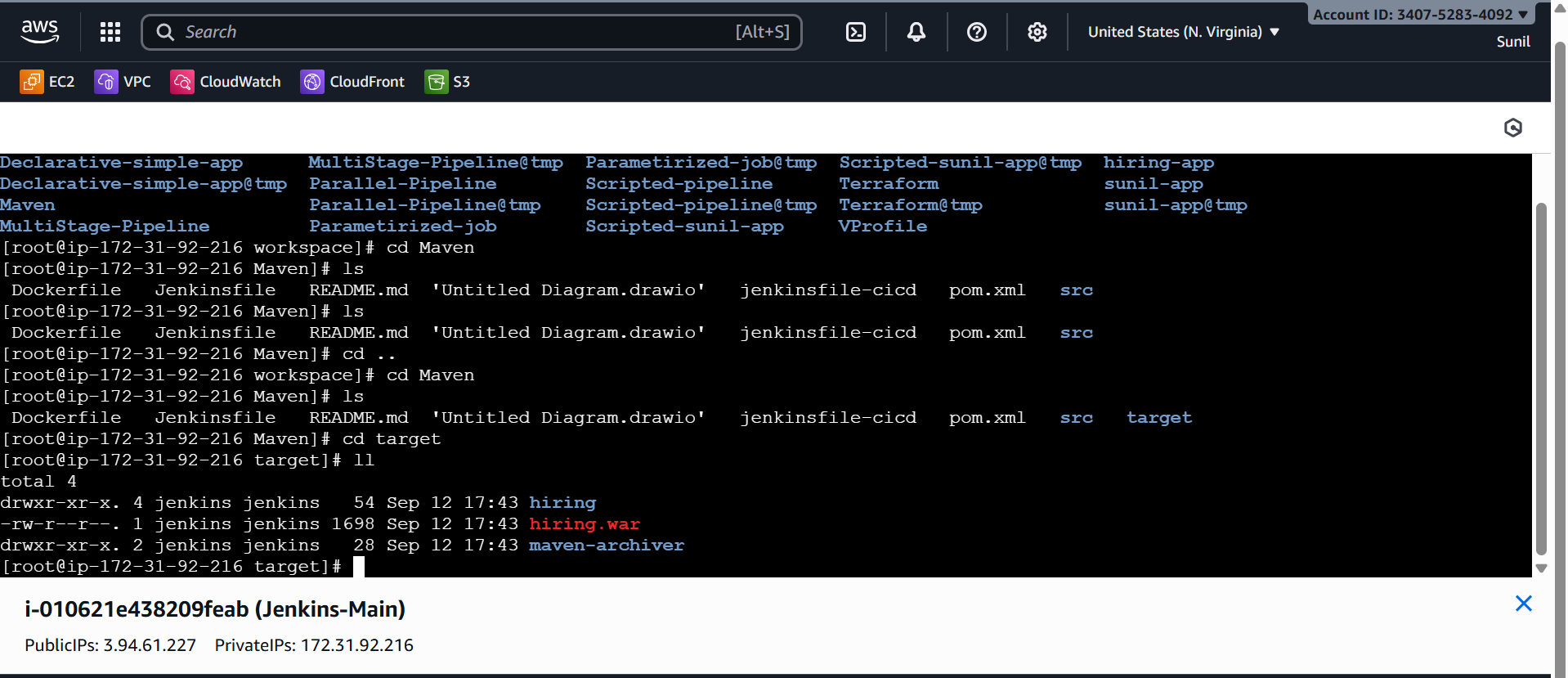
**Save and Build**

****

Maven by default puts the output in the project’s **target/ directory**.

Jenkins checks out your Git repo into a **workspace**:

**/var/lib/jenkins/workspace/Maven/target/**

****

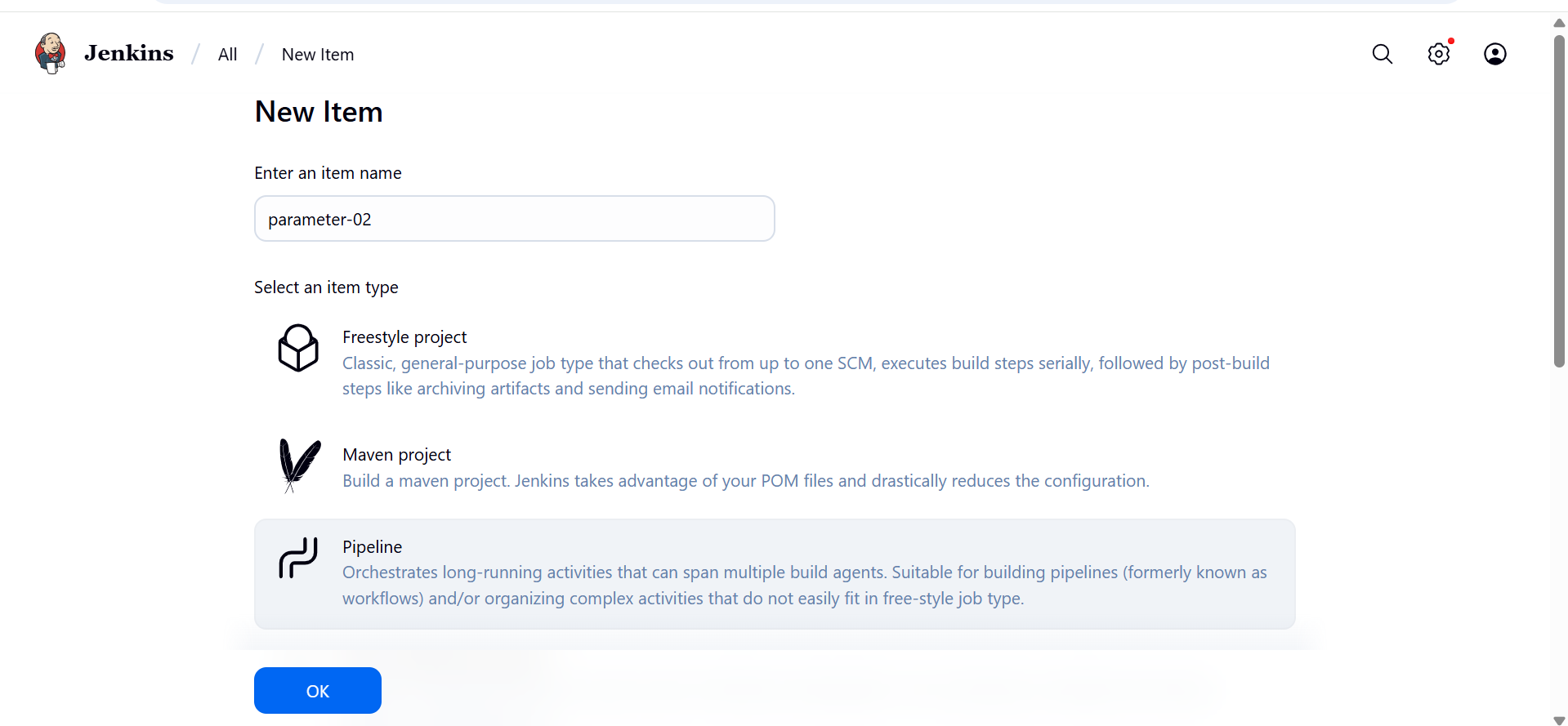
1. **Use the same code and create a parameterized job in Jenkins with:**

**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)

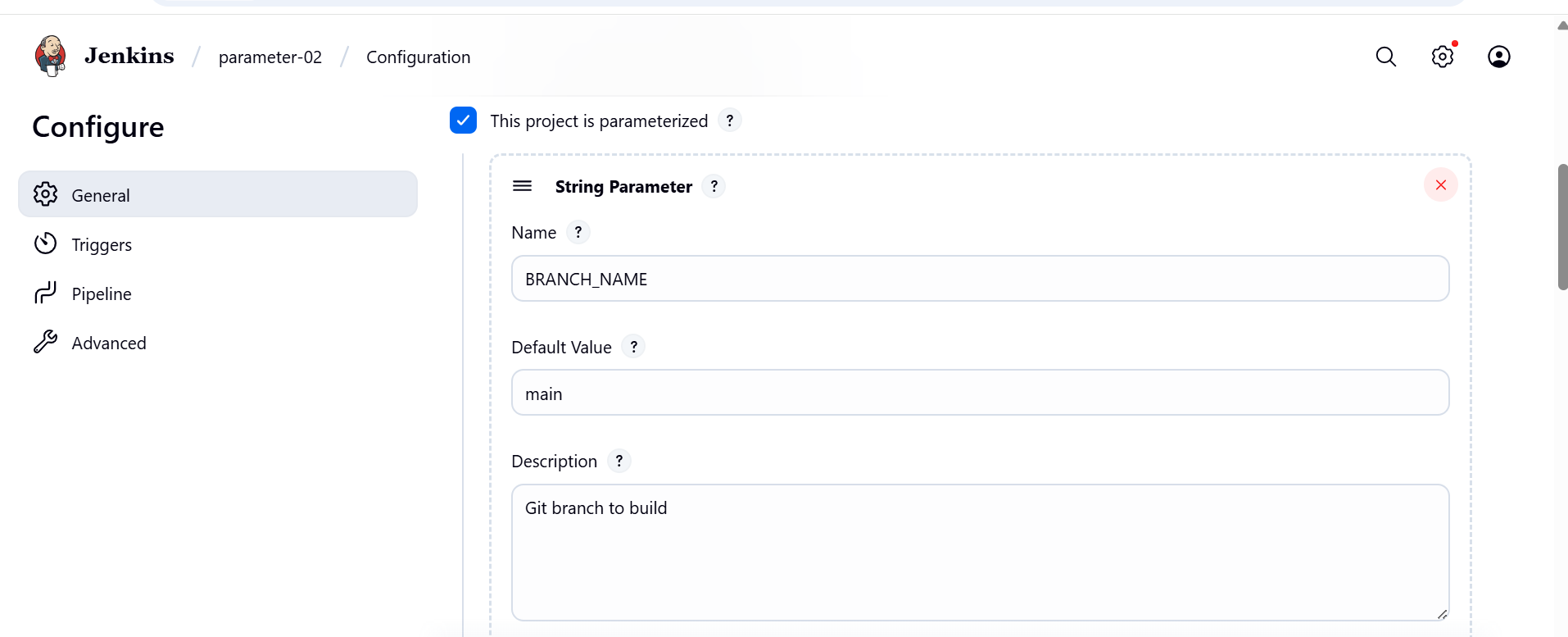
## Step 1: Create a Parameterized Job

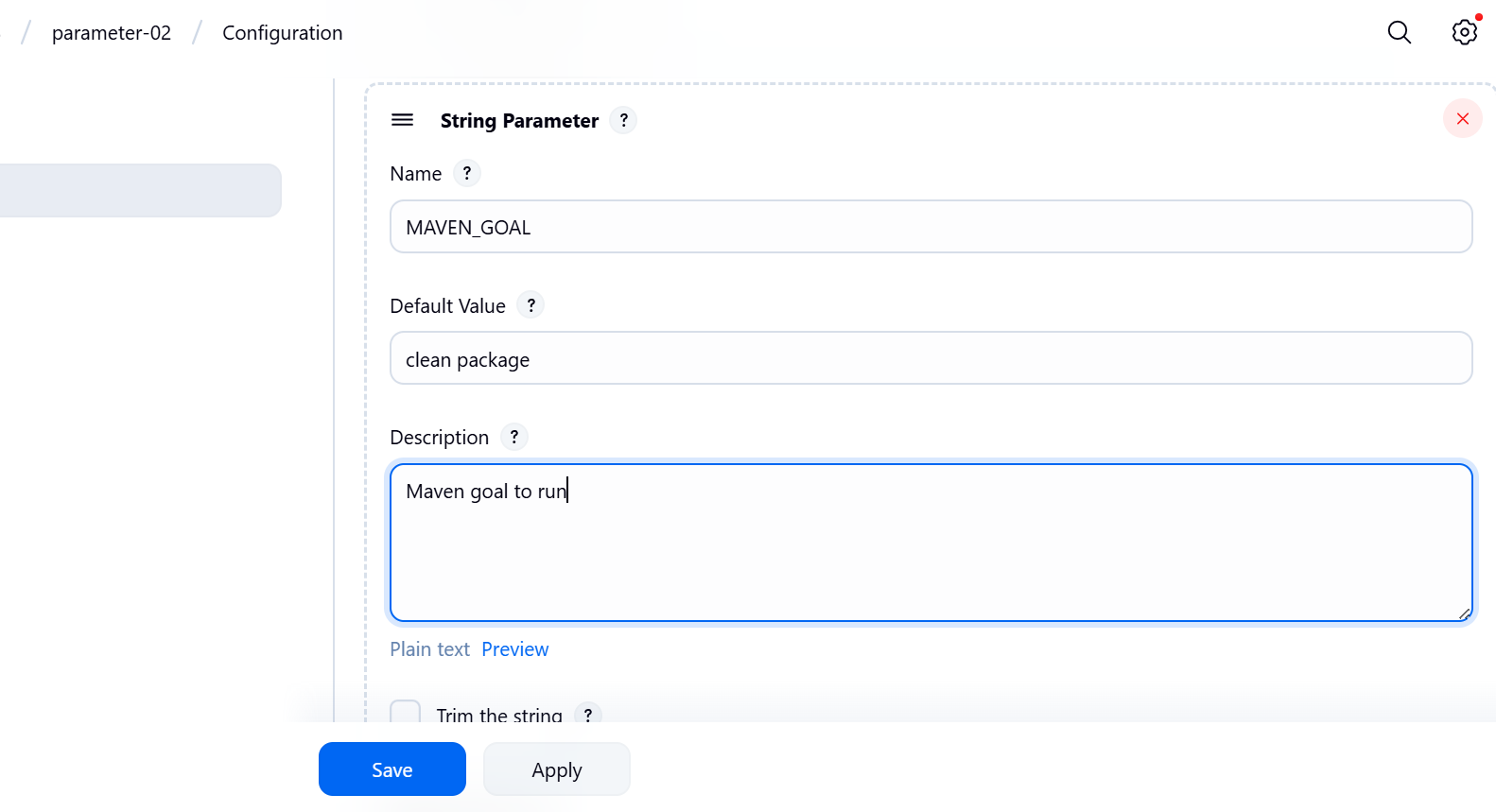
1. Go to **Jenkins Dashboard → New Item**
2. Choose **Pipeline** → give a name → click OK

****

In the job config:

* Check **“This project is parameterized”**
* Add parameters:
  + **String Parameter** → Name: BRANCH\_NAME, Default: main, Description: Git branch to build
  + **String Parameter** → Name: MAVEN\_GOAL, Default: clean package, Description: Maven goal to run

****

****

**Step 2: Add Scripted Pipeline Code**

node {

def mvnHome = tool name: 'Maven-02', type: 'maven'

env.PATH = "${mvnHome}/bin:${env.PATH}"

stage('Git Clone') {

git branch: "${BRANCH\_NAME}", url: 'https://github.com/betawins/java-Working-app.git'

}

stage('Maven Compilation') {

sh "mvn ${MAVEN\_GOAL}"

}

stage('Archive Package') {

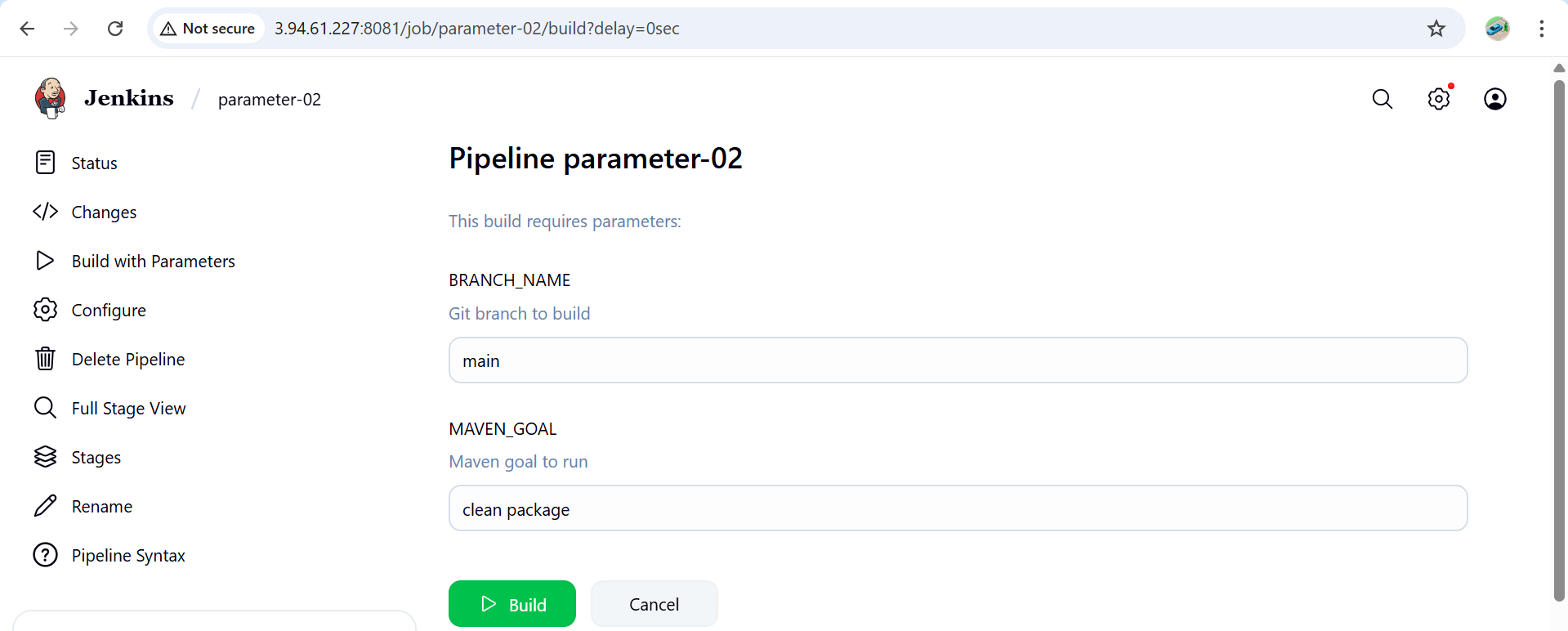
// Fix: WAR instead of JAR

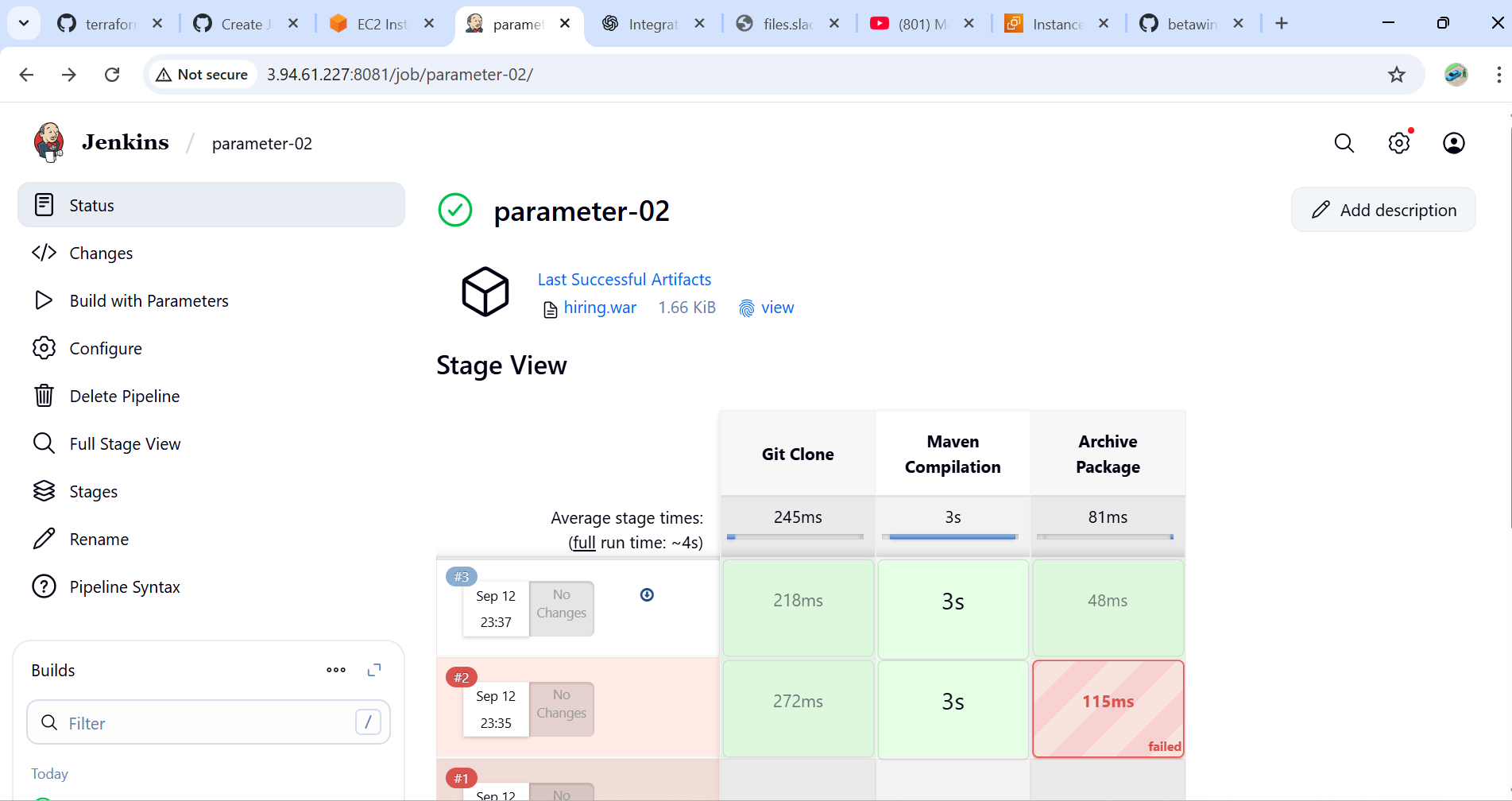
archiveArtifacts artifacts: 'target/\*.war', fingerprint: true

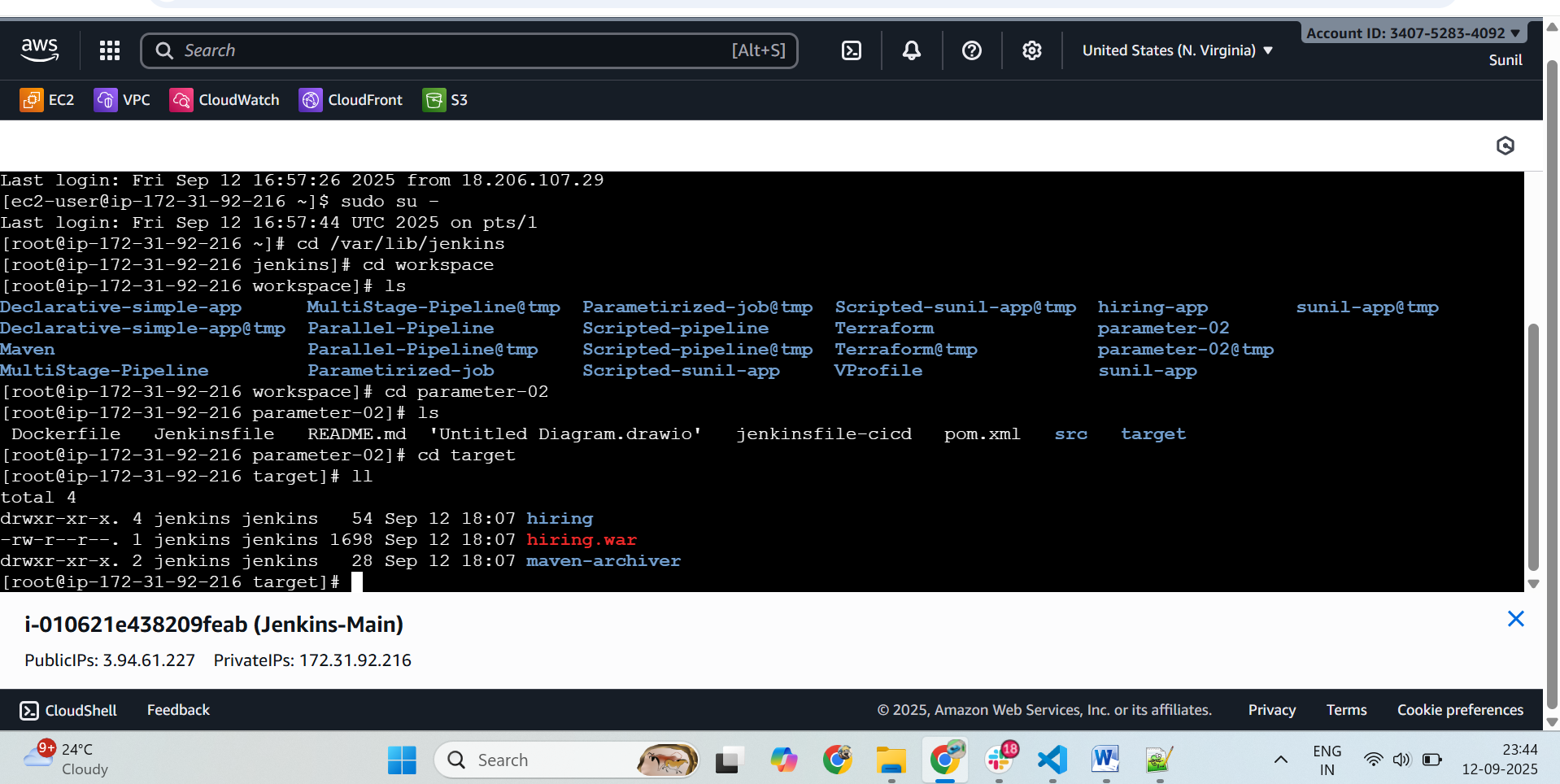
}

}

**Save and build with parameters**



****

****

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

In Jenkins Pipelines, **Global Variables** are special built-in environment variables and objects that Jenkins automatically provides. You don’t need to define them — they are available everywhere in your pipeline.

### ****Job & Build Info****

* BUILD\_ID → Unique build ID (timestamp)
* BUILD\_NUMBER → Build number (incremental)
* BUILD\_TAG → jenkins-${JOB\_NAME}-${BUILD\_NUMBER}
* JOB\_NAME → Name of the Jenkins job
* JOB\_BASE\_NAME → Job name without folder path
* BUILD\_URL → URL of the build in Jenkins
* JOB\_URL → URL of the job

### ****Node/Workspace Variables****

* NODE\_NAME → Name of the agent node running the build
* WORKSPACE → Path to the job’s workspace directory

### ****Git/SCM Variables**** (when using git step)

* GIT\_COMMIT → Commit hash being built
* GIT\_BRANCH → Branch being built
* GIT\_URL → Git repo URL

### ****Special Global Objects (Pipeline DSL)****

These are not strings, but powerful objects you can call methods on:

* steps → Provides pipeline steps (e.g. sh, git, echo)
* pipeline → DSL for declarative pipelines
* docker → Access Docker agents/containers if plugin is enabled
* input → Request user input
* credentials() → Inject stored credentials

## Summary

* **env** → Environment variables
* **params** → Build parameters
* **currentBuild** → Build metadata
* **Built-in vars** → BUILD\_NUMBER, JOB\_NAME, WORKSPACE, BUILD\_URL, etc.
* **SCM vars** → GIT\_COMMIT, GIT\_BRANCH, GIT\_URL