**1. Watch Terraform-04 video.**

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

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

**4. Create a CI/CD pipeline for a Nodejs Application:** [**https://github.com/betawins/Trading-UI.git**](https://github.com/betawins/Trading-UI.git)

**5. Explain 10 Maven commands.**

**1. Watch Terraform-04 video.**

**Completed**

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

Downloading terraform version

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

terraform {

required\_providers {

local = {

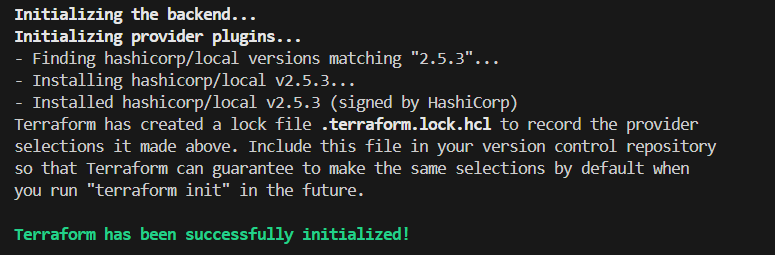
source = "hashicorp/local"

version = "2.5.3"

}

}

}

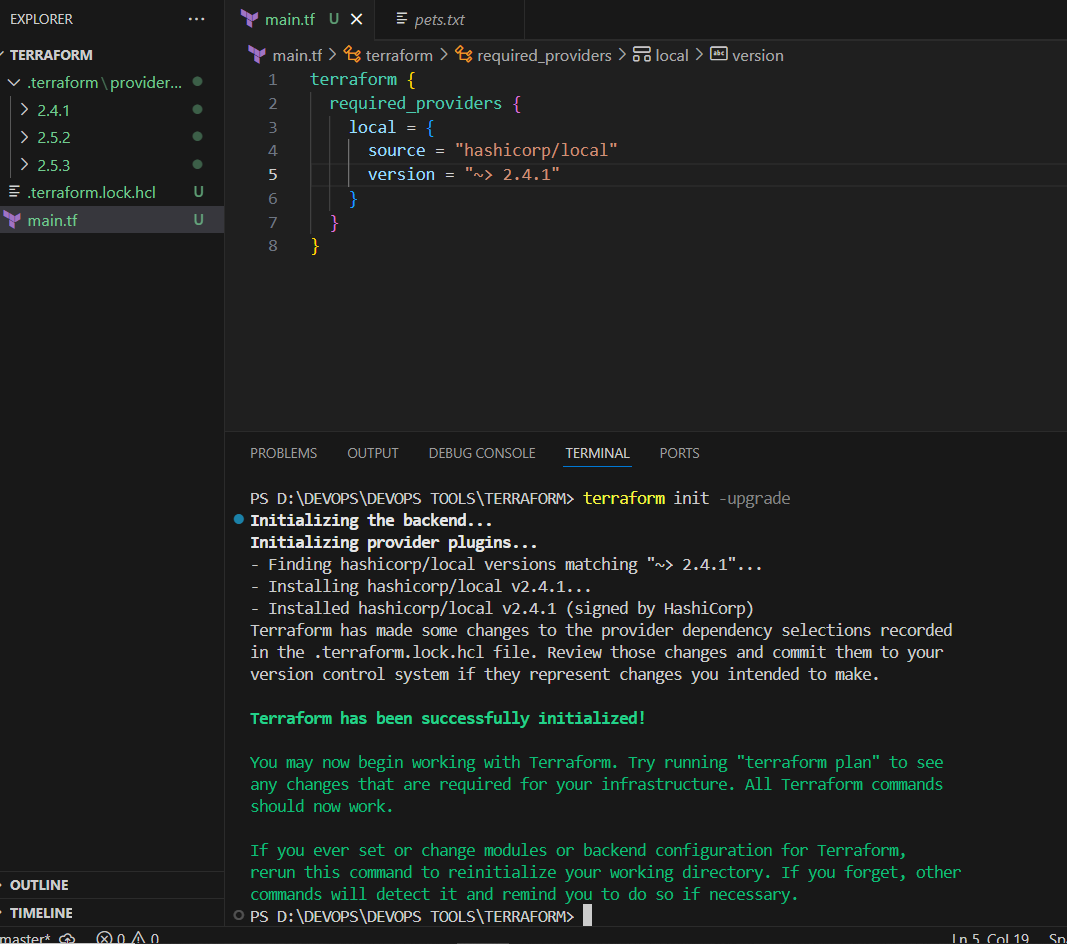


version = "!=2.5.3"

version = "< 2.5.3"

version = "> 2.4.1"

version = "~> 2.4.1"



DATA SOURCES

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

filename = "pets.txt"

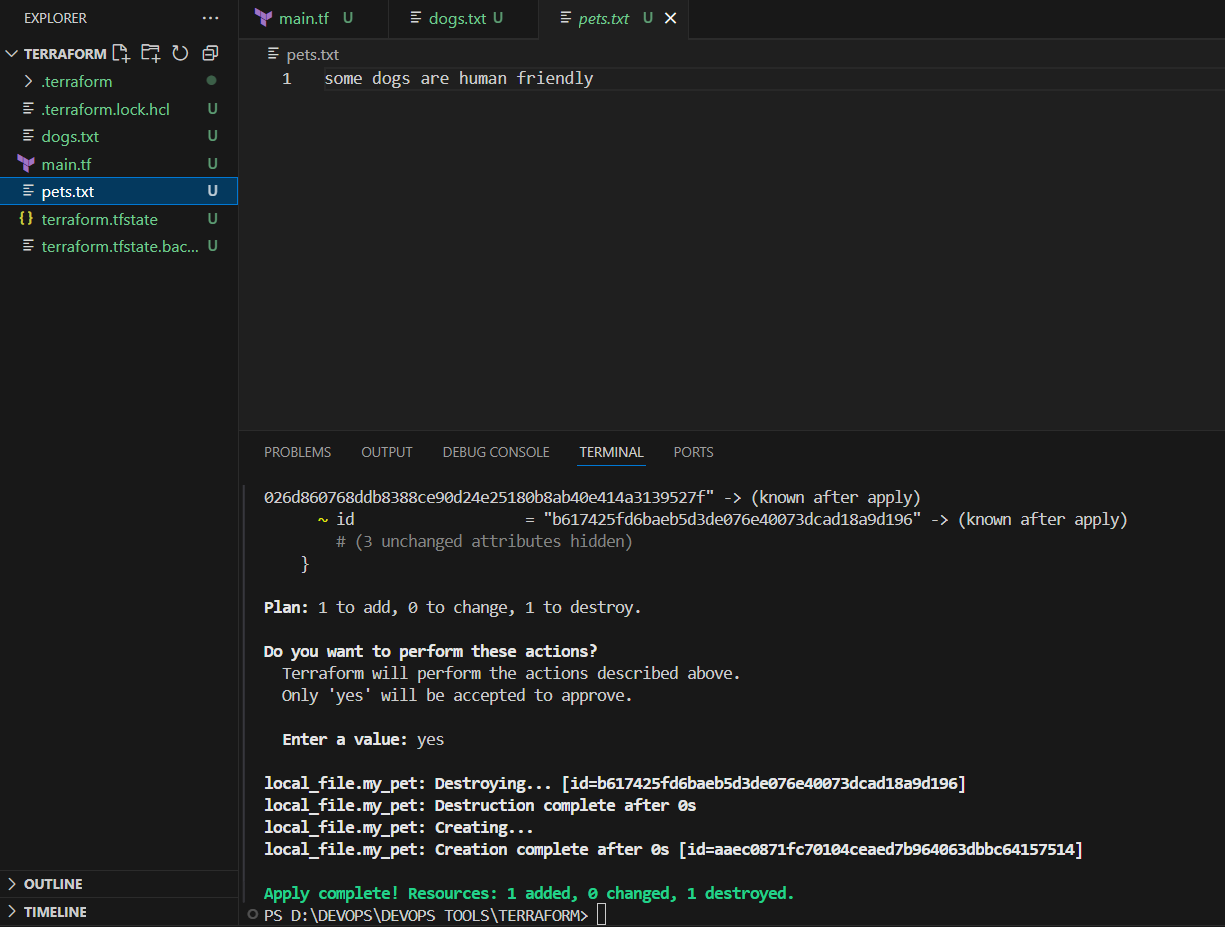
content = data.local\_file.dog.content

}

data "local\_file" "dog"{

filename = "dogs.txt"

}



META ARGUMENTS

1. Depends\_on
2. Life cycle rules
3. Count
4. for\_each

**COUNT**

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

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

filename = var.filename[count.index]

content = var.content[count.index]

count=length(var.filename)

}

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

variable "filename" {

default = [

"pets.txt",

"cats.txt",

"dogs.txt"

]

}

variable "content" {

default = [

"pets are so good",

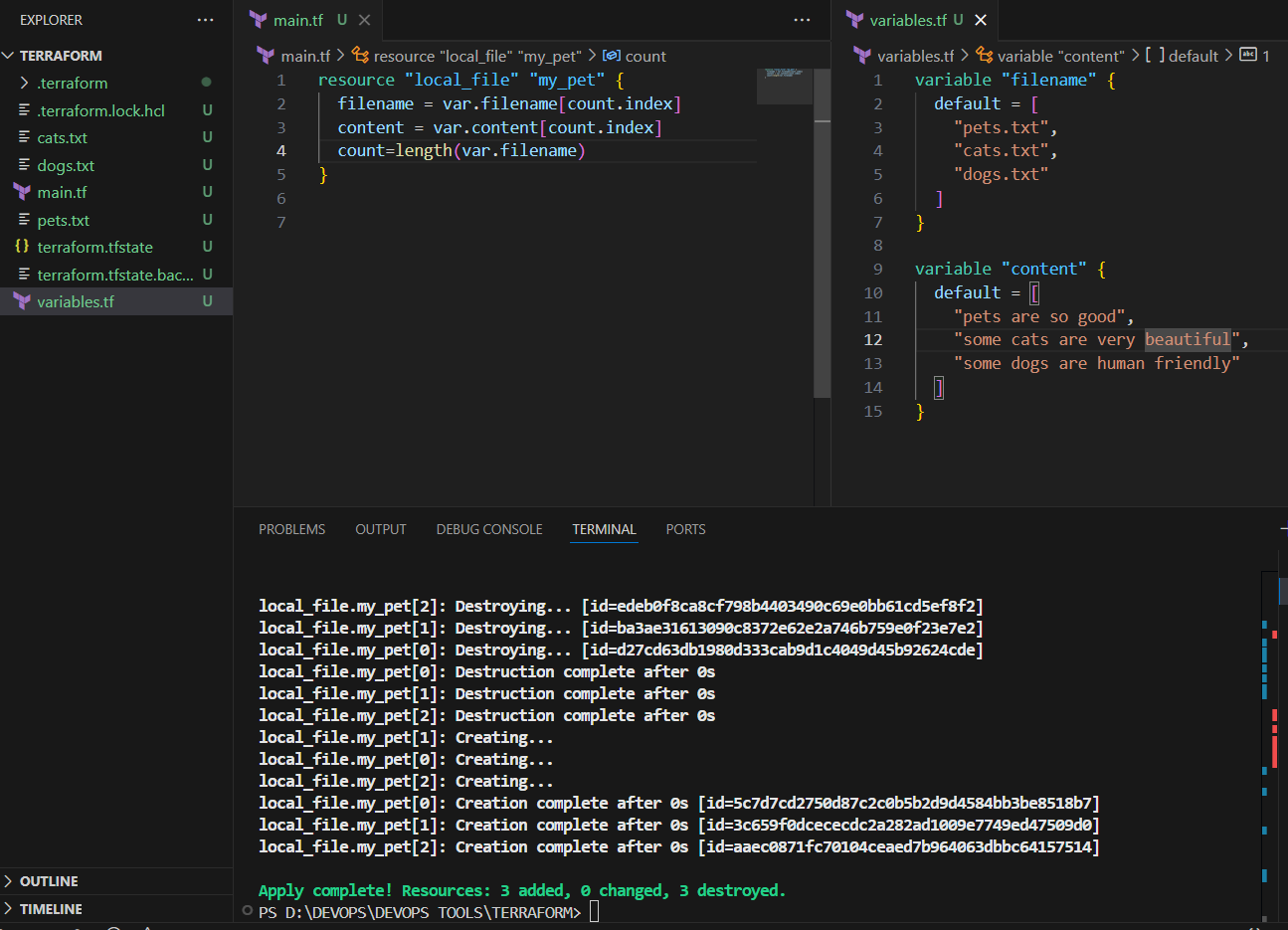
"some cats are very beautiful",

"some dogs are human friendly"

]

}

Terraform apply



For each

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

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

for\_each = toset(var.filename) # convert list → set for for\_each

filename = each.value # each.value = current file name

content = "i like cats"

}

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

variable "filename" {

default = [

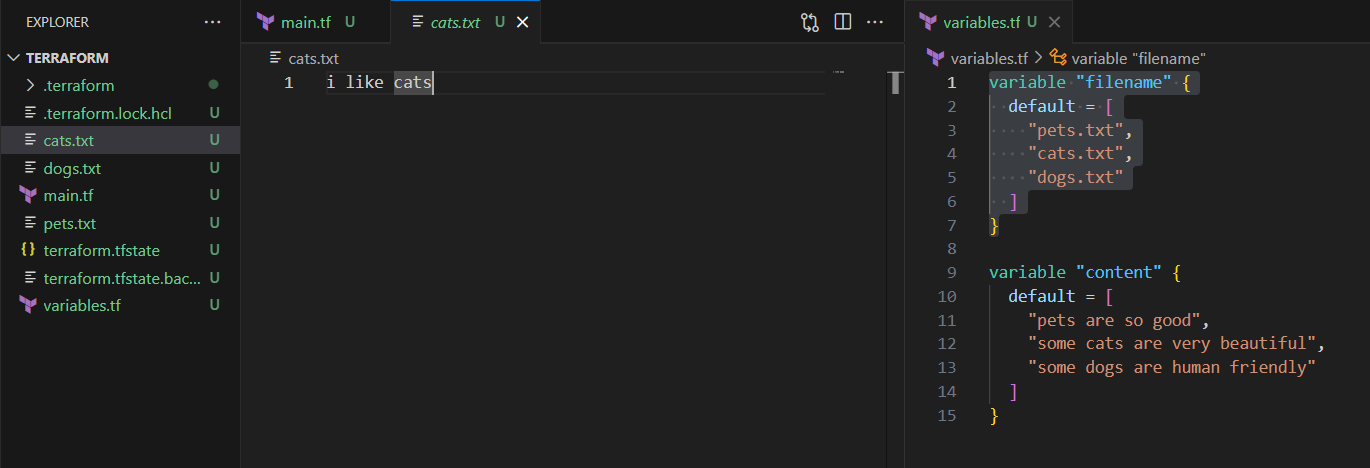
"pets.txt",

"cats.txt",

"dogs.txt"

]

}



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

for\_each = var.filename

filename = each.value

content = "i like cats some of them so cute"

}

variable "filename" {

type = set(string)

default = [

"pets.txt",

"cats.txt",

"dogs.txt"

]

}

FOR EACH

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

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

for\_each = var.files

filename = each.key # file name

content = each.value # file content

}

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

variable "files" {

default = {

"pets.txt" = "pets are so good"

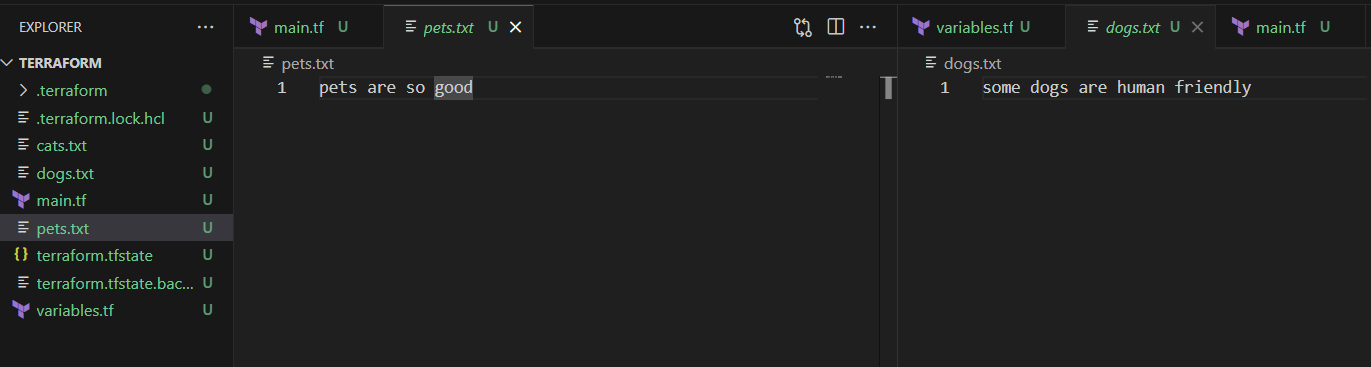
"cats.txt" = "some cats are very beautiful"

"dogs.txt" = "some dogs are human friendly"

}

}

Terraform apply



SAMPLE SCRIPT to Create IAM User.

1. Aws configure in bash tool / in VS Code terminal
2. Terraform template for creating user and attaching policy.

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

name="imran"

tags={

"description" = "Technical team lead"

}

}

resource "aws\_iam\_policy" "adminuser" {

name = "AdminUsers"

policy = <<EoF

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "1234567890",

"Effect": "Allow",

"Action": "\*",

"Resource": "\*"

}

]

}

EoF

}

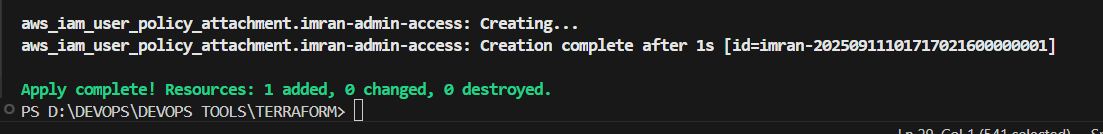
resource "aws\_iam\_user\_policy\_attachment" "imran-admin-access" {

user = aws\_iam\_user.Admin-user.name

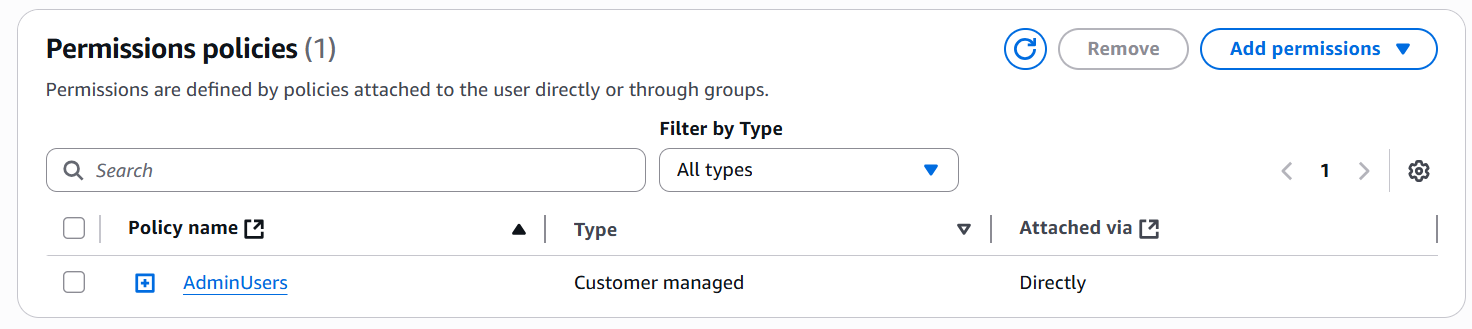
policy\_arn = aws\_iam\_policy.adminuser.arn

}

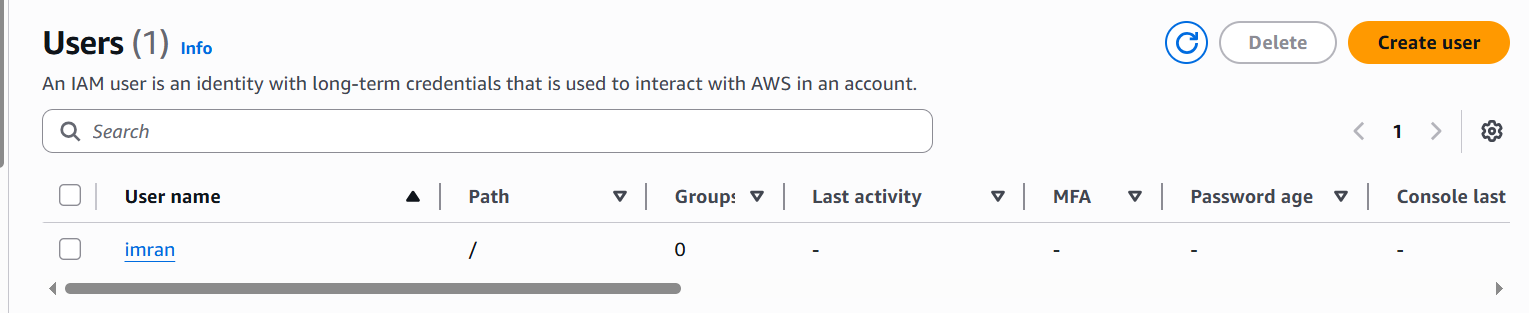
1. Terraform apply



In AWS account imran users created and policy attached.



<< EOF … its called Heredoc syntax

1. 

We can create admin-policy.json file also

**admin-policy.json**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Sid": "1234567890",**

**"Effect": "Allow",**

**"Action": "\*",**

**"Resource": "\*"**

**}**

**]**

**}**

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

**# Create IAM User**

**resource "aws\_iam\_user" "admin\_user" {**

**name = "lucy"**

**tags = {**

**description = "Technical Team Lead"**

**}**

**}**

**# Create IAM Policy using external JSON file**

**resource "aws\_iam\_policy" "admin\_policy" {**

**name = "AdminUsers"**

**policy = file("admin-policy.json")**

**}**

**# Attach IAM Policy to the User**

**resource "aws\_iam\_user\_policy\_attachment" "lucy\_admin\_access" {**

**user = aws\_iam\_user.admin\_user.name**

**policy\_arn = aws\_iam\_policy.admin\_policy.arn**

**}**

**terraform init**

**terraform validate**

**terraform plan**

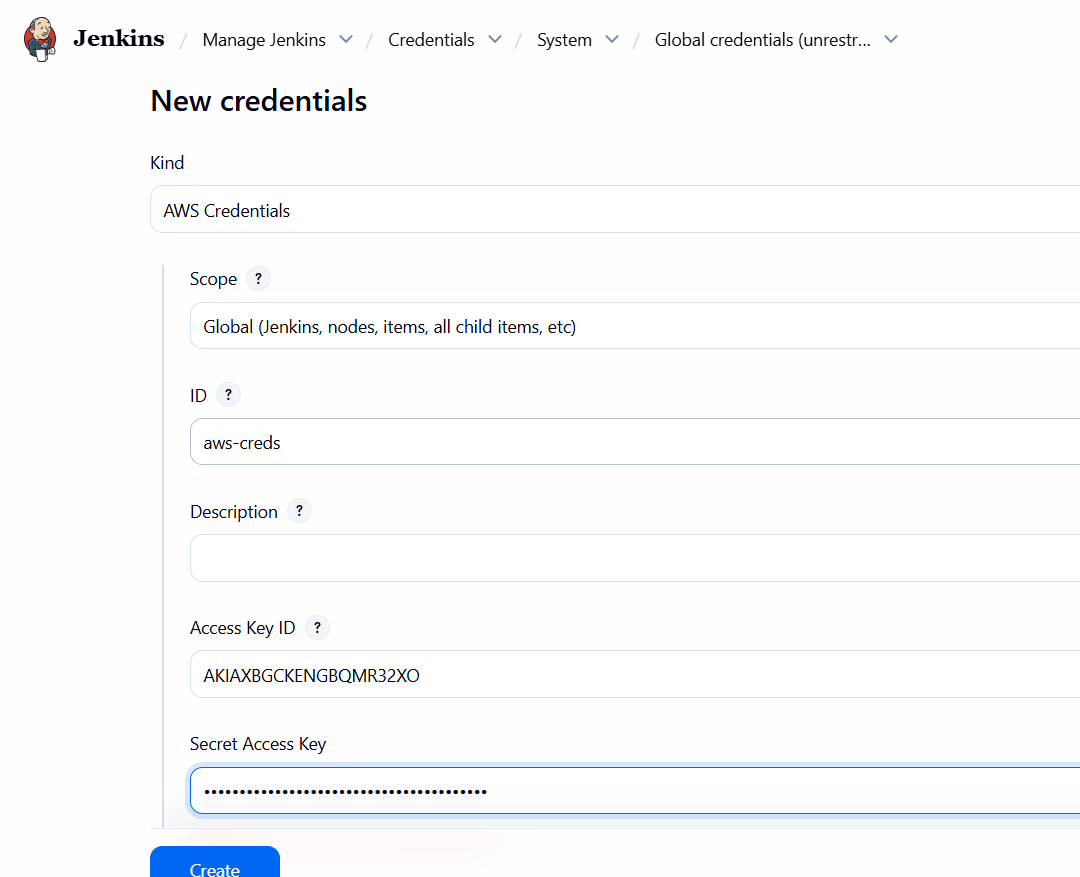
**terraform apply -auto-approve**

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

## **A — Jenkins setup (common steps)**

1. **Install plugins**
   * **Jenkins → Manage Jenkins → Manage Plugins → Available → install:**
     + **Terraform (Terraform plugin)**
     + **AWS Credentials (if you plan to store AWS keys natively)**
     + **Credentials Binding (should be installed by default)**
2. **Configure Terraform in Jenkins (Global Tool Configuration)**
   * **Jenkins → Manage Jenkins → Global Tool Configuration → find Terraform → Add Terraform installation (give a name like Terraform) and use the auto-installer (or point to an installed binary).**
   * **You can also manually install Terraform on the Jenkins agent and ensure it's on PATH.**
3. **Add AWS credentials to Jenkins**
   * **Jenkins → Credentials → System → Global domain → Add Credentials**
     + **Kind: AWS Credentials (recommended plugin). Give it ID aws-creds (or name you prefer).**
     + **Or: Kind Username with password (access key in username, secret in password) — but prefer AWS Credentials plugin.**

**Security tip: Do NOT hardcode AWS keys in .tf files. Use Jenkins credentials and inject them at runtime.**

****

## **B — Terraform files (place in repo root)**

**admin-policy.json**

**{**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Sid": "1234567890",**

**"Effect": "Allow",**

**"Action": "\*",**

**"Resource": "\*"**

**}**

**]**

**}**

**Important: For production, avoid Action: "\*", Resource: "\*". Replace with least-privilege statements.**

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

**provider "aws" {**

**region = var.region**

**# Credentials are read from environment variables injected by Jenkins (AWS\_ACCESS\_KEY\_ID / AWS\_SECRET\_ACCESS\_KEY)**

**}**

**variable "region" {**

**default = "us-east-1"**

**}**

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

**variable "username" {**

**description = "IAM username to create"**

**type = string**

**default = "lucy"**

**}**

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

**resource "aws\_iam\_user" "user" {**

**name = var.username**

**tags = {**

**Description = "Created by Jenkins/Terraform"**

**}**

**}**

**resource "aws\_iam\_policy" "admin\_policy" {**

**name = "${var.username}-policy"**

**policy = file("${path.module}/admin-policy.json")**

**}**

**resource "aws\_iam\_user\_policy\_attachment" "user\_attach" {**

**user = aws\_iam\_user.user.name**

**policy\_arn = aws\_iam\_policy.admin\_policy.arn**

**}**

**output "iam\_username" {**

**value = aws\_iam\_user.user.name**

**}**

## **C – Pipeline approach (recommended) — Jenkinsfile example**

**Put this Jenkinsfile in the same repo (root). It uses Jenkins AWS credentials binding, runs Terraform CLI (installed via Global Tool Config or on agent), and creates the IAM user/policy**

**pipeline {**

**agent any**

**parameters {**

**string(name: 'IAM\_USERNAME', defaultValue: 'lucy', description: 'IAM user name to create')**

**string(name: 'AWS\_REGION', defaultValue: 'us-east-1', description: 'AWS region')**

**}**

**environment {**

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

**POLICY\_NAME = 'AdminUsers' // adjust based on your terraform config**

**}**

**stages {**

**stage('Checkout') {**

**steps {**

**checkout scm**

**sh 'ls -la'**

**}**

**}**

**stage('Prepare Terraform') {**

**steps {**

**sh 'terraform -version || true'**

**}**

**}**

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

**steps {**

**withCredentials([[$class: 'AmazonWebServicesCredentialsBinding', credentialsId: 'aws-creds']]) {**

**sh '''**

**export AWS\_REGION=${AWS\_REGION}**

**export AWS\_DEFAULT\_REGION=${AWS\_REGION}**

**terraform init -input=false**

**terraform validate**

**terraform plan -out=tfplan -var "username=${IAM\_USERNAME}"**

**'''**

**}**

**}**

**}**

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

**steps {**

**input message: "Apply terraform changes to create IAM user ${params.IAM\_USERNAME}?"**

**}**

**}**

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

**steps {**

**withCredentials([[$class: 'AmazonWebServicesCredentialsBinding', credentialsId: 'aws-creds']]) {**

**sh '''**

**export AWS\_REGION=${AWS\_REGION}**

**export AWS\_DEFAULT\_REGION=${AWS\_REGION}**

**terraform apply -auto-approve tfplan || terraform apply -auto-approve -var "username=${IAM\_USERNAME}"**

**'''**

**}**

**}**

**}**

**}**

**post {**

**success {**

**slackSend (**

**channel: '#jenkins-integration',**

**message: "✅ AWS IAM User \*${params.IAM\_USERNAME}\* created with policy \*${env.POLICY\_NAME}\* in region \*${params.AWS\_REGION}\* by Imran Khan"**

**)**

**}**

**failure {**

**slackSend (**

**channel: '#jenkins-integration',**

**message: "❌ Terraform failed for IAM User \*${params.IAM\_USERNAME}\* in region \*${params.AWS\_REGION}\*. Check Jenkins logs."**

**)**

**}**

**}**

**}**

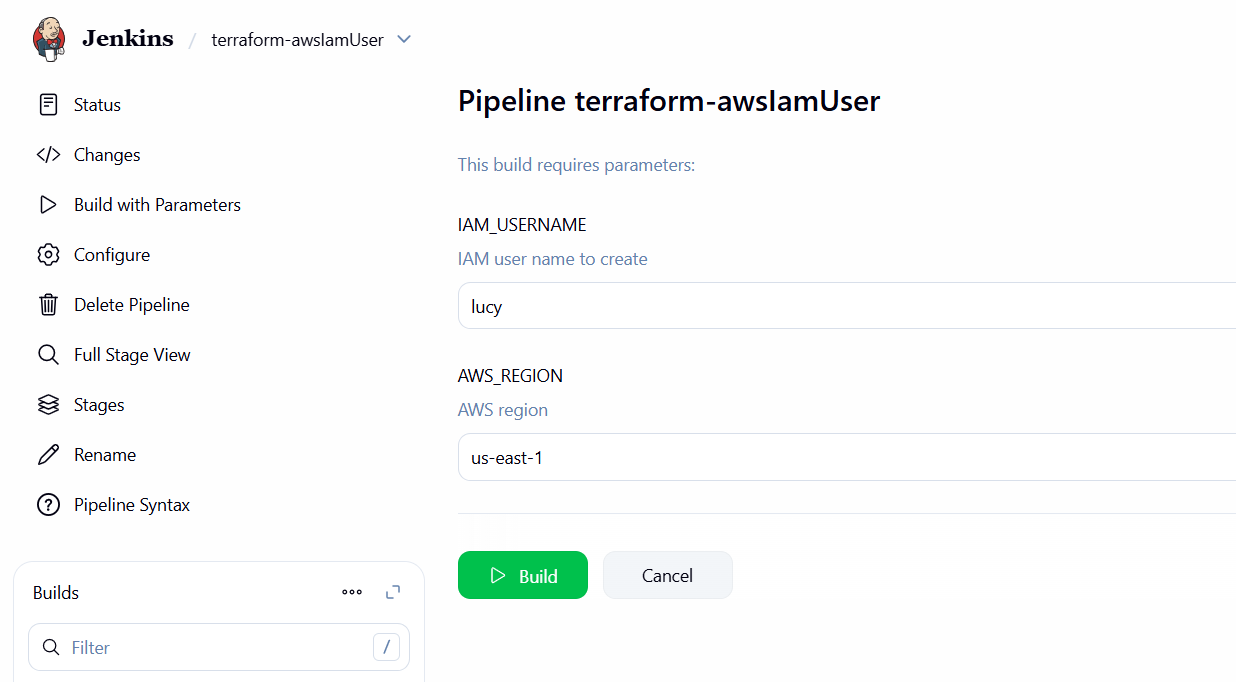
**Notes:**

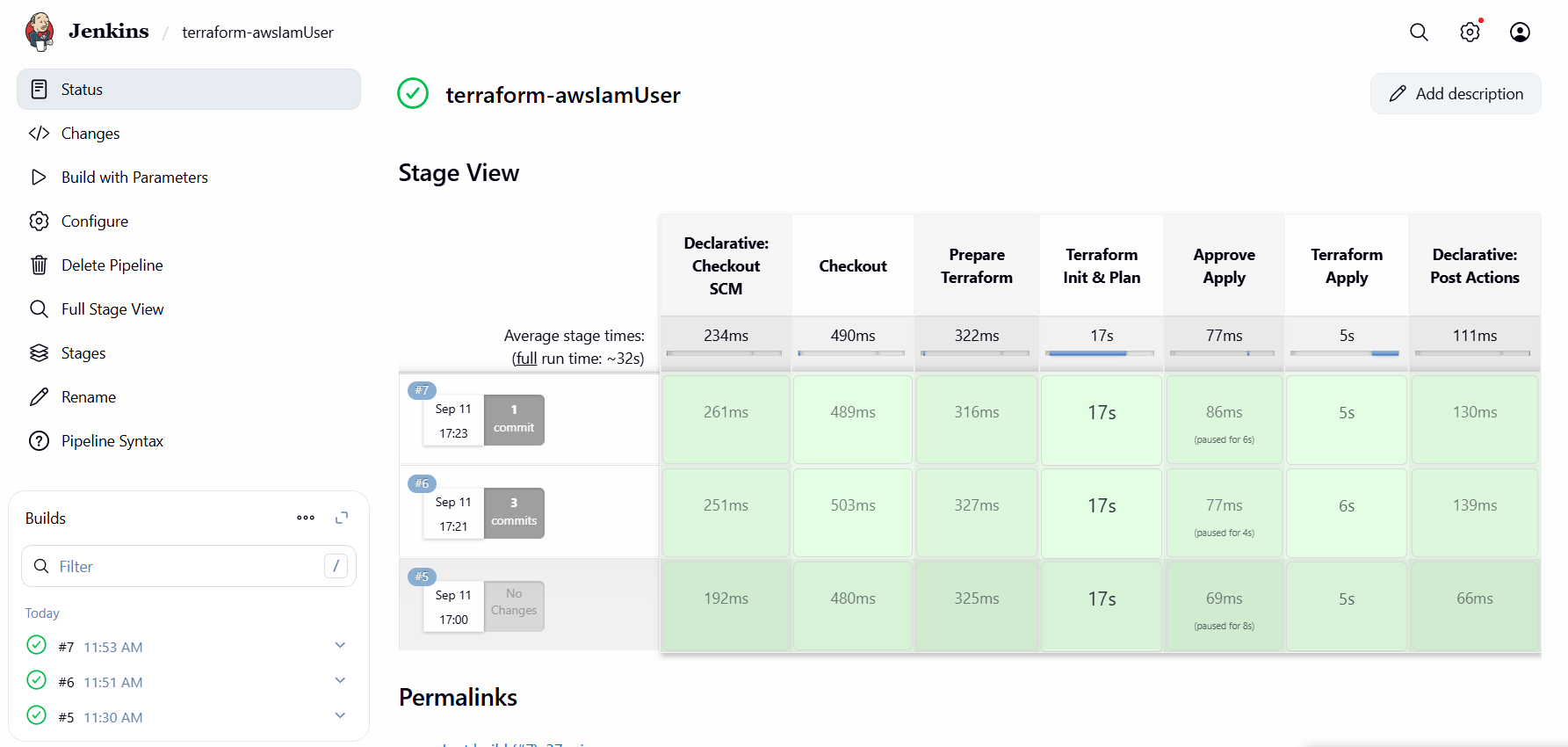
* **credentialsId: 'aws-creds' must match the ID you added in Jenkins credentials.**
* **Terraform reads AWS credentials from AWS\_ACCESS\_KEY\_ID / AWS\_SECRET\_ACCESS\_KEY injected by the AWS Credentials binding plugin.**
* **Always run apply behind a manual input approval stage in production.**
* **Prefer remote backend (S3 + DynamoDB lock) for state — I can add S3 backend config if you want.**

## **D — IAM permissions required for Jenkins credentials**

* **The AWS user whose keys you add to Jenkins must be allowed to:**
  + **iam:CreateUser, iam:CreatePolicy, iam:AttachUserPolicy, iam:PutUserPolicy, and iam:TagUser (or broader if needed).**
* **For admins you used in example, you may need iam:\* (but avoid that in production).**

**BUILD**

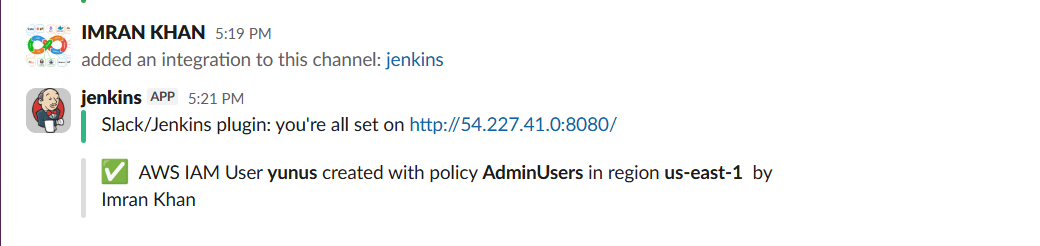
****

****

**CONSOLE OUTPUT**

****

**SLACK NOTIFICATION**

****

**4. Create a CI/CD pipeline for a Nodejs Application:**

[**https://github.com/imrankhanmohammad257/Trading-UI.git**](https://github.com/imrankhanmohammad257/Trading-UI.git)

**Install in jenkins ec2**

**sudo yum install -y nodejs npm**

**Plugins installed**

**Node js**

**Pipeline npm integration**

# **Documentation for Node.js Jenkins Pipeline**

## **1️⃣ Overview**

**This pipeline is a CI/CD workflow for a Node.js React project hosted on GitHub. It performs:**

1. **Node.js version validation**
2. **Source code checkout**
3. **Dependency installation**
4. **Running tests**
5. **Building the application**
6. **Slack notifications on success or failure**

**The pipeline is written using Jenkins Declarative Pipeline syntax.**

**2️⃣ Jenkinsfile Structure**

**pipeline {**

**agent any**

**tools {**

**nodejs "NodeJS14"**

**}**

**environment {**

**NODE\_VERSION = ''**

**}**

**stages {**

**stage('Validate Node.js') { ... }**

**stage('Checkout') { ... }**

**stage('Install') { ... }**

**stage('Test') { ... }**

**stage('Build') { ... }**

**}**

**post { ... }**

**}**

### **Explanation of blocks:**

* **pipeline → Main block defining the pipeline.**
* **agent any → Runs the pipeline on any available Jenkins agent.**
* **tools → Defines required tools (Node.js in this case). Must match your Jenkins NodeJS installation name.**
* **environment → Defines global environment variables available across stages (NODE\_VERSION).**

## **3️⃣ Stages**

### **Stage 1: Validate Node.js**

**stage('Validate Node.js') {**

**steps {**

**script {**

**NODE\_VERSION = sh(script: 'node -v', returnStdout: true).trim()**

**echo "Node.js version: ${NODE\_VERSION}"**

**}**

**}**

**}**

**Purpose:**

* **Checks which Node.js version is installed on the Jenkins agent.**
* **Stores the version in NODE\_VERSION environment variable.**
* **Ensures the pipeline is using the correct Node.js version.**

**Explanation:**

* **sh(script: 'node -v', returnStdout: true) → Runs node -v and captures output.**
* **.trim() → Removes newline characters.**
* **echo → Prints Node.js version in the Jenkins console log.**

**Stage 2: Checkout**

**stage('Checkout') {**

**steps {**

**git url: 'https://github.com/imrankhanmohammad257/Trading-UI.git', branch: 'master'**

**}**

**}**

**Purpose:**

* **Pulls source code from GitHub repository.**
* **Checks out the master branch.**

**Explanation:**

* **git url: '...', branch: '...' → Built-in Jenkins git step.**
* **Ensures the latest code is available for build and test.**

**Stage 3: Install Dependencies**

**stage('Install') {**

**steps {**

**sh '''**

**npm cache clean --force**

**rm -rf node\_modules package-lock.json**

**npm install --omit=optional**

**'''**

**}**

**}**

**Purpose:**

* **Cleans any existing node\_modules and lock files.**
* **Installs project dependencies from package.json.**

**Explanation:**

1. **npm cache clean --force → Clears npm cache to avoid conflicts.**
2. **rm -rf node\_modules package-lock.json → Removes old dependencies and lock file.**
3. **npm install --omit=optional → Installs only required dependencies (skips optional ones).**

**Note: Removing package-lock.json every build is optional; it can lead to slightly different dependency versions.**

**Stage 4: Test**

**stage('Test') {**

**steps {**

**sh 'npm test || echo "⚠️ No tests found"'**

**}**

**}**

**Purpose:**

* **Runs project unit tests using npm test.**
* **If no tests are found, prints a warning instead of failing the build.**

**Explanation:**

* **|| echo "⚠️ No tests found" → Prevents build from failing if there are no tests.**

**Stage 5: Build**

**stage('Build') {**

**steps {**

**withEnv(["CI=false"]) {**

**sh 'npm run build'**

**}**

**}**

**}**

**Purpose:**

* **Builds the React application into the build/ folder.**
* **CI=false avoids errors from React scripts in CI environments.**

**4️⃣ Post-Build Actions (Slack Notifications)**

**post {**

**success {**

**slackSend(**

**channel: '#jenkins-integration',**

**message: "✅ Pipeline succeeded. Node.js version installed: \*${NODE\_VERSION}\* By Imran Khan"**

**)**

**}**

**failure {**

**slackSend(**

**channel: '#jenkins-integration',**

**message: "❌ Pipeline failed. Node.js version installed: \*${NODE\_VERSION}\*. Check Jenkins logs."**

**)**

**}**

**}**

**Purpose:**

* **Sends notifications to Slack on pipeline success or failure.**
* **Includes the Node.js version used for the build.**

**Explanation:**

* **slackSend(channel: '...', message: '...') → Jenkins Slack plugin step.**
* **${NODE\_VERSION} → Displays the Node.js version captured earlier.**
* **Helps developers quickly see the environment and build result.**

## **5️⃣ Notes / Best Practices**

1. **Node.js version management**
   * **Ensure NodeJS version in Jenkins matches your project’s requirements.**
   * **Validate in the first stage (Validate Node.js) to prevent version mismatch issues.**
2. **Slack Notifications**
   * **Slack plugin must be installed in Jenkins.**
   * **Configure Slack integration under Manage Jenkins → Configure System → Slack.**
3. **Environment Variables**
   * **Use environment block for variables needed across multiple stages.**
4. **Build Caching**
   * **Optional: You can cache node\_modules between builds to save time.**

**JENKINSFILE**

**pipeline {**

**agent any**

**tools {**

**nodejs "NodeJS14" // or NodeJS20 if you upgrade**

**}**

**environment {**

**NODE\_VERSION = ''**

**}**

**stages {**

**stage('Validate Node.js') {**

**steps {**

**script {**

**NODE\_VERSION = sh(script: 'node -v', returnStdout: true).trim()**

**echo "Node.js version: ${NODE\_VERSION}"**

**}**

**}**

**}**

**stage('Checkout') {**

**steps {**

**git url: 'https://github.com/imrankhanmohammad257/Trading-UI.git', branch: 'master'**

**}**

**}**

**stage('Install') {**

**steps {**

**sh '''**

**npm cache clean --force**

**rm -rf node\_modules package-lock.json**

**npm install --omit=optional**

**'''**

**}**

**}**

**stage('Test') {**

**steps {**

**sh 'npm test || echo "⚠️ No tests found"'**

**}**

**}**

**stage('Build') {**

**steps {**

**withEnv(["CI=false"]) {**

**sh 'npm run build'**

**}**

**}**

**}**

**} // ✅ closes stages block**

**post {**

**success {**

**slackSend(**

**channel: '#jenkins-integration',**

**message: "✅ Pipeline succeeded. Node.js version installed: \*${NODE\_VERSION}\* By Imran Khan"**

**)**

**}**

**failure {**

**slackSend(**

**channel: '#jenkins-integration',**

**message: "❌ Pipeline failed. Node.js version installed: \*${NODE\_VERSION}\*. Check Jenkins logs."**

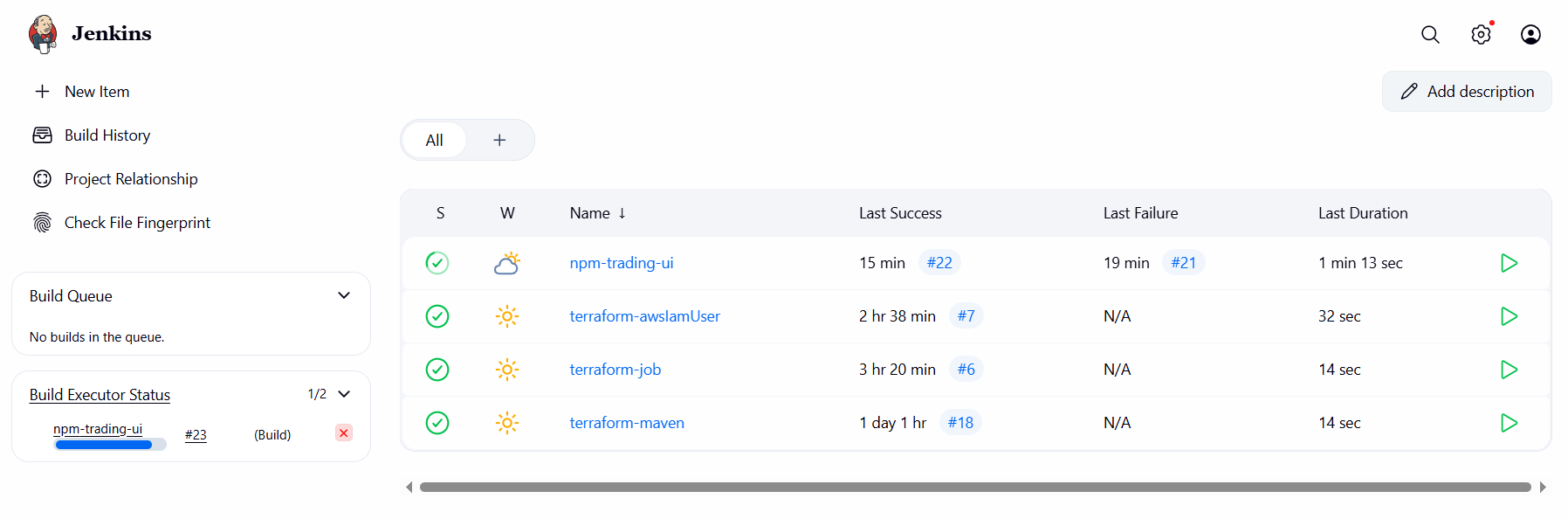
**)**

**}**

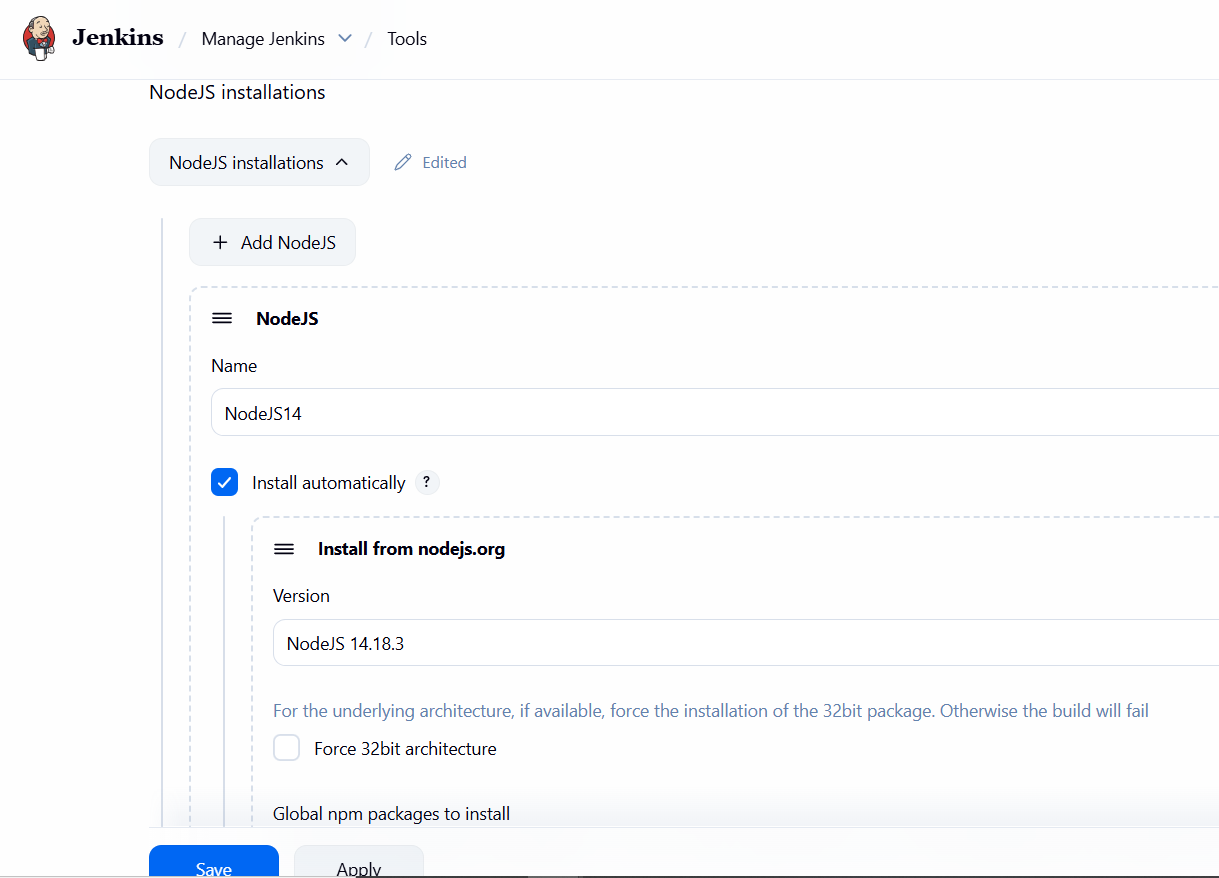
**}**

**} // ✅ closes pipeline**

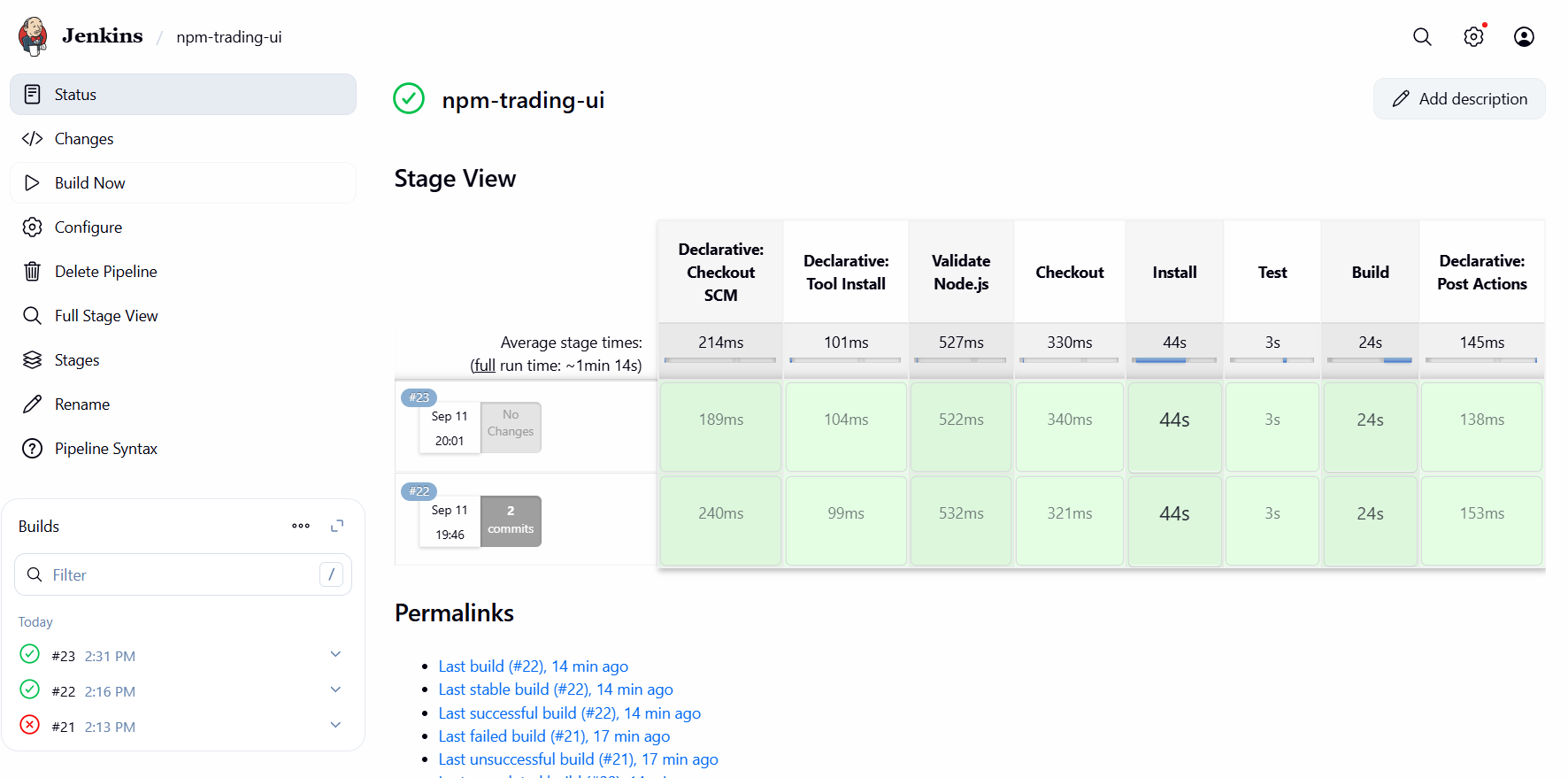
**BUILD**

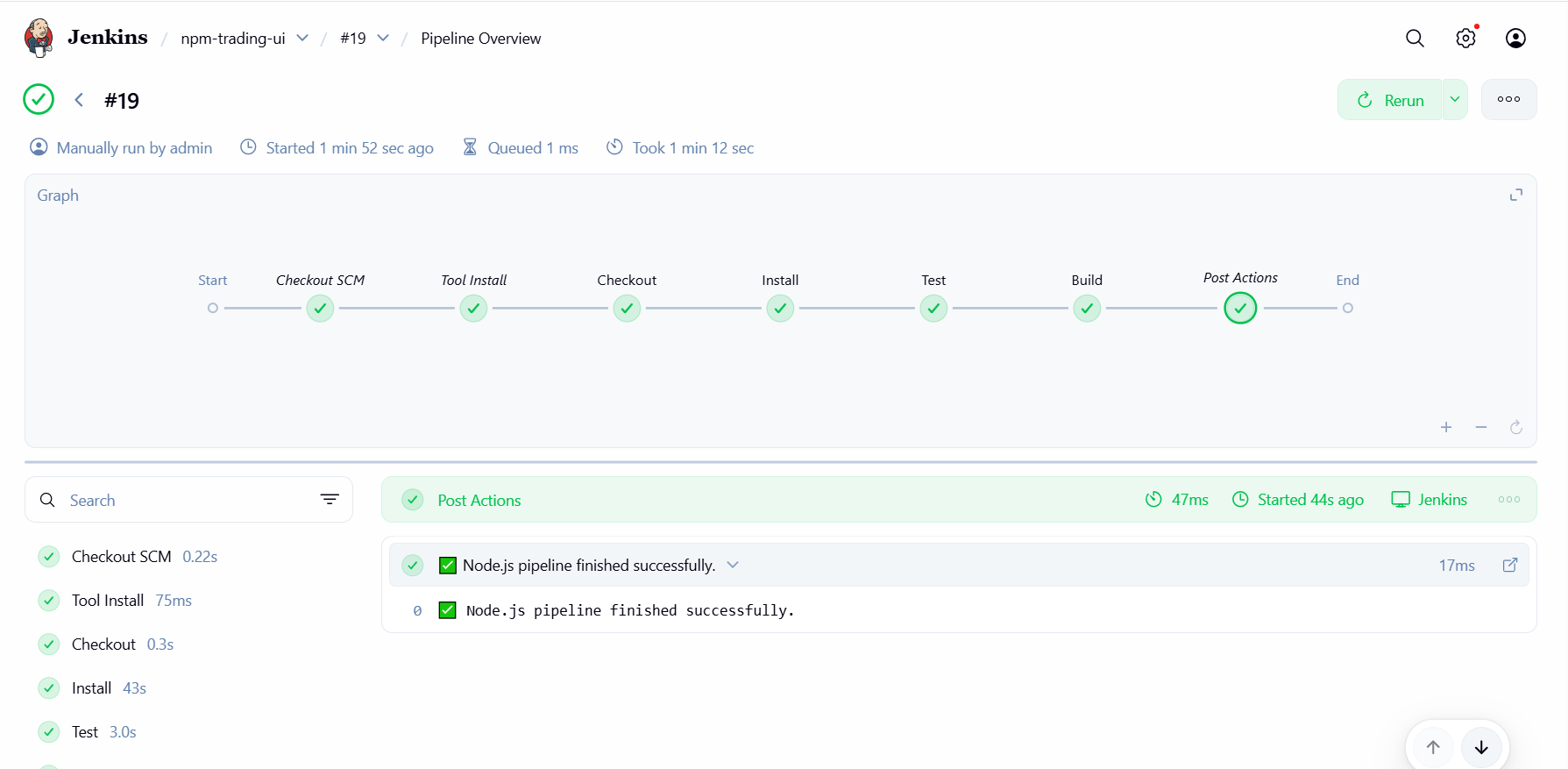
****

**TOOLS**

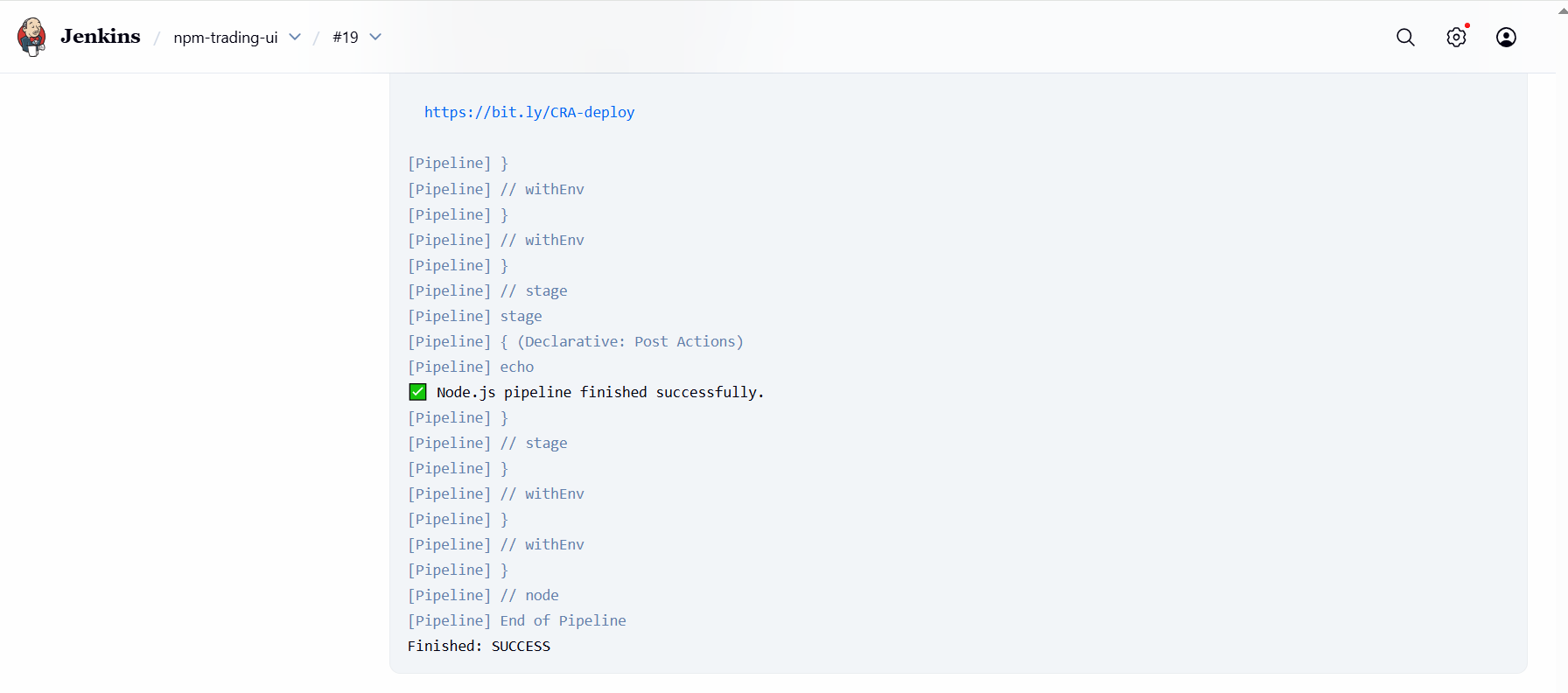
****

**JOB STATUS**

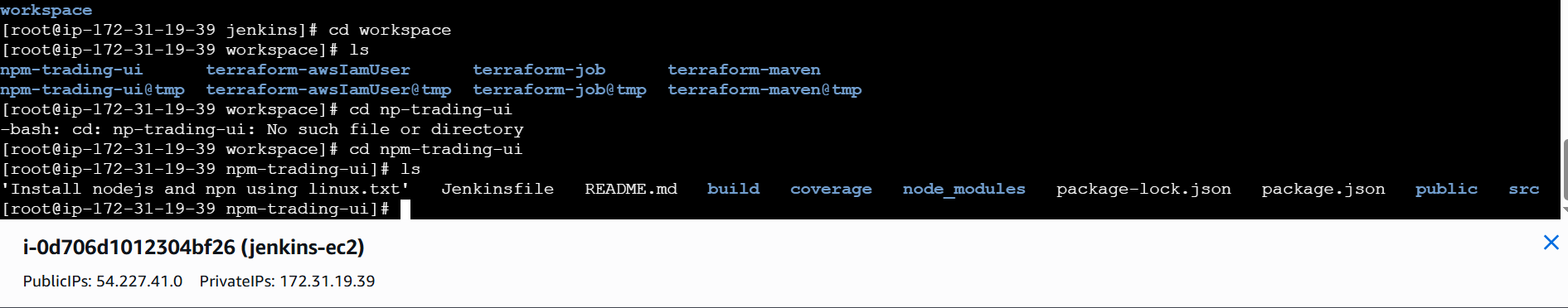
****

****

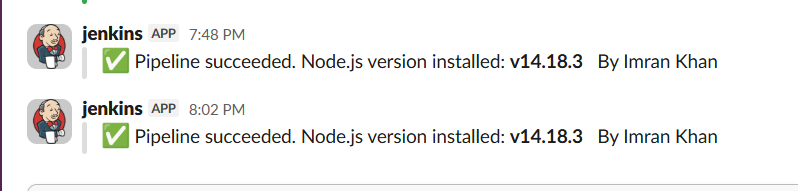
**CONSOLE OUTPUT**

****

**JENKINS EC2**

****

**SLACK NOTIFICATION**

****

**5. Explain 10 Maven commands.**

**I’ll list 10 commands you’ll use regularly, with short explanation and example usage.**

1. **mvn clean**
   * **Removes the target/ directory (build output). Good to start fresh.**
   * **Example: mvn clean**
2. **mvn compile**
   * **Compiles the source code of the project. Produces target/classes.**
   * **Example: mvn compile**
3. **mvn test**
   * **Runs unit tests (uses Surefire plugin by default).**
   * **Example: mvn -Dtest=MyTest test (run single test)**
4. **mvn package**
   * **Compiles, runs tests, and packages the code into its distributable format (JAR/WAR).**
   * **Example: mvn package**
5. **mvn install**
   * **Builds the project and installs the artifact into the local Maven repository (~/.m2/repository). Useful for multi-module builds.**
   * **Example: mvn install**
6. **mvn deploy**
   * **After successful package/install, uploads the artifact to a remote repository (like Nexus or Artifactory). Needs distributionManagement configured.**
   * **Example: mvn deploy -DskipTests**
7. **mvn verify**
   * **Runs integration tests and any checks that require the package to be built (verifies project is valid and meets quality gates).**
   * **Example: mvn verify**
8. **mvn -DskipTests=true package**
   * **Skip running tests while building the package. Useful for fast builds when tests are run separately.**
   * **Example: mvn -B -DskipTests clean package**
9. **mvn dependency:tree**
   * **Shows the dependency tree — great to debug conflicts and transitive dependencies.**
   * **Example: mvn dependency:tree -Dverbose**
10. **mvn help:effective-pom**

* **Prints the *effective* POM after inheritance & interpolation — useful to understand final config (plugins, versions).**
* **Example: mvn help:effective-pom**

**Bonus extras**

* **mvn -U forces update of SNAPSHOT dependencies.**
* **mvn versions:use-latest-releases to upgrade dependencies (with caution).**