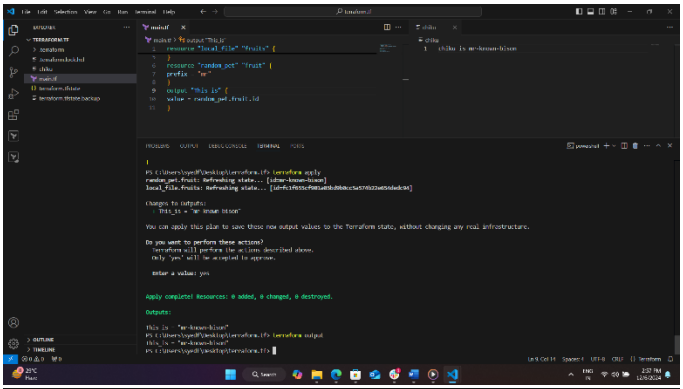
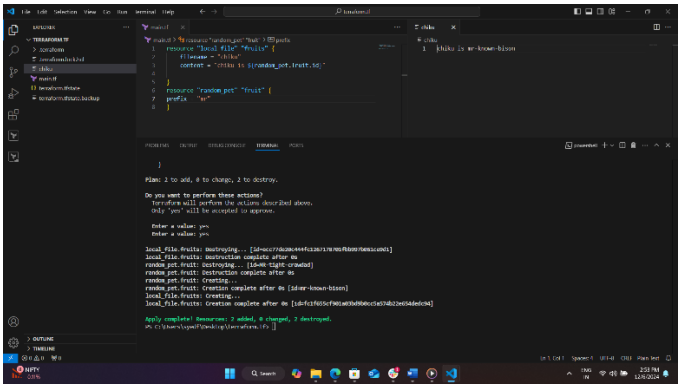
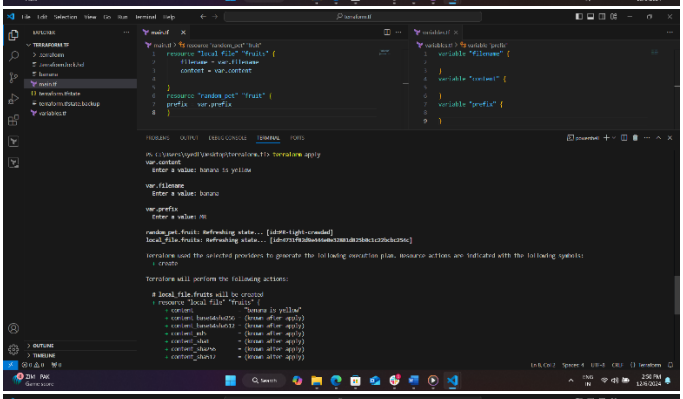
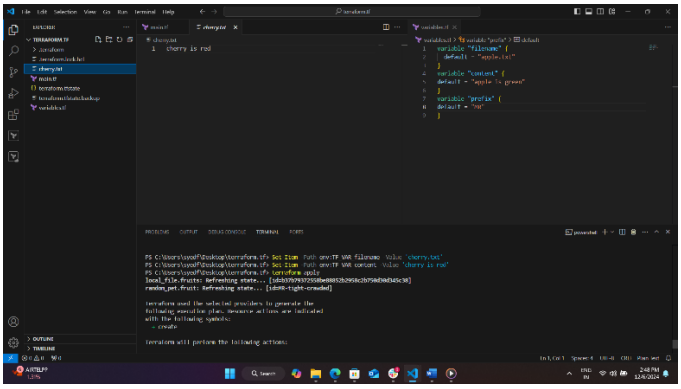
TERRAFORM-03_04

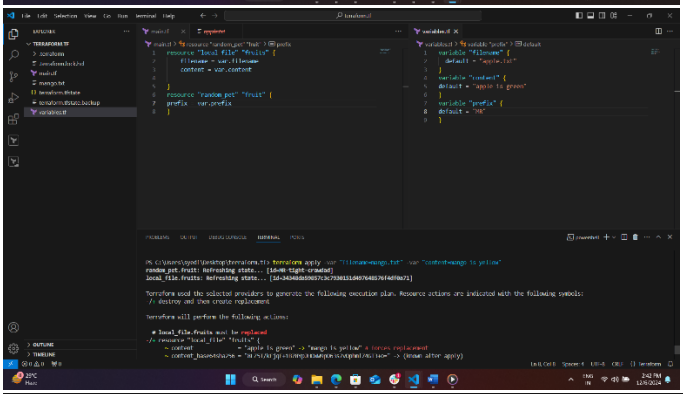
1) Watch terraform-03 video. and 2)Execute the script shown in video.

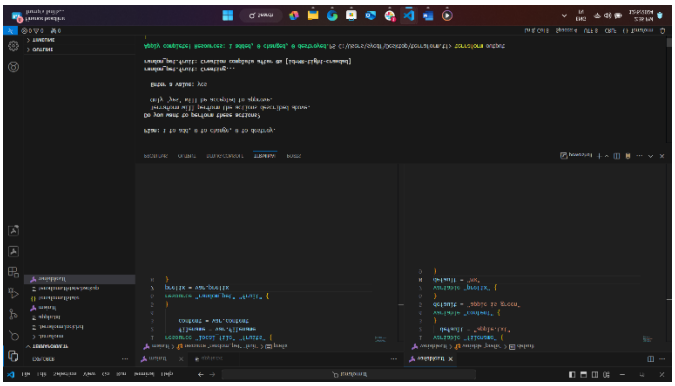


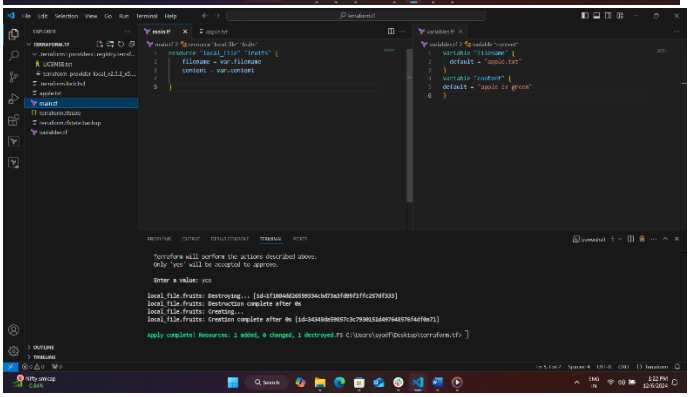




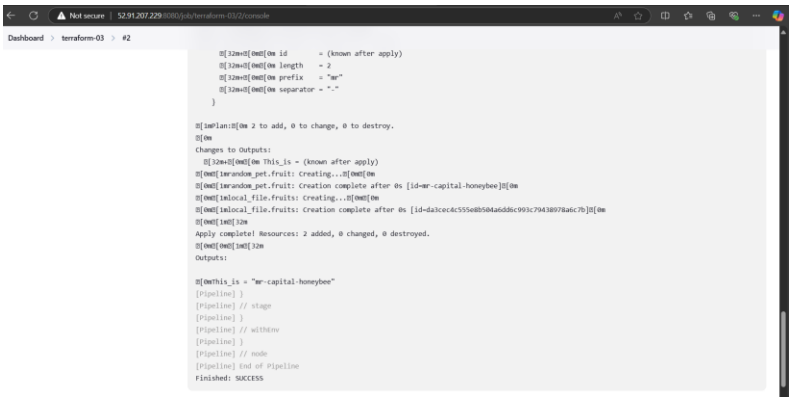








3) Intergrate terrafrom in jenkins using Terraform plugin.



4) Create one jenkins job using MAVEN PROJECT for the below code with two stages.

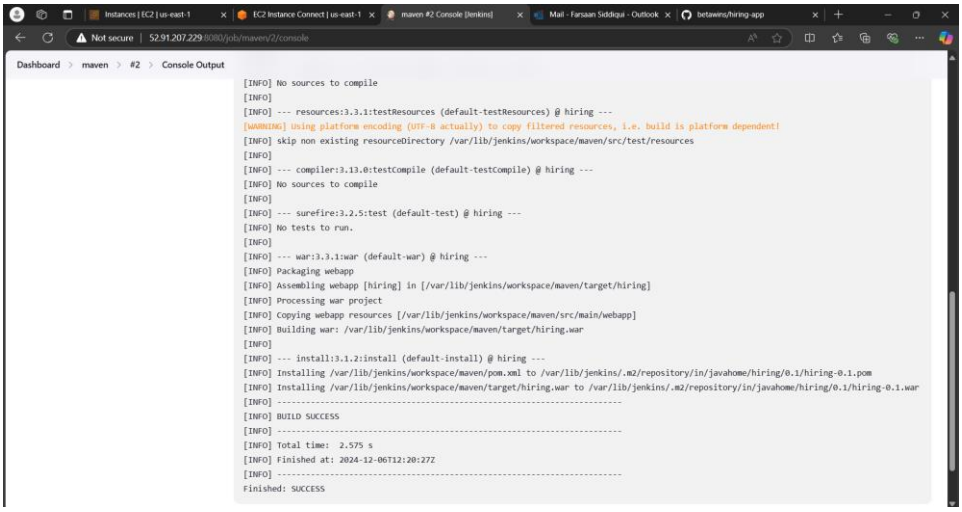
stage 1: Git clone

```
Last login: Fri Dec  6 09:32:52 2024 from 18.206.107.29
[ec2-user@ip-10-0-0-19 ~]$ cd /var/lib/jenkins/workspace/maven/
[ec2-user@ip-10-0-0-19 maven]$ ll
total 24
-rw-r--r--. 1 jenkins jenkins  476 Dec  6 12:20  Dockerfile
-rw-r--r--. 1 jenkins jenkins 1757 Dec  6 12:20  Jenkinsfile
-rw-r--r--. 1 jenkins jenkins   88 Dec  6 12:20  README.md
-rw-r--r--. 1 jenkins jenkins  408 Dec  6 12:20 'Untitled Diagram.drawio'
-rw-r--r--. 1 jenkins jenkins 1801 Dec  6 12:20  jenkinsfile-cicd
-rw-r--r--. 1 jenkins jenkins  890 Dec  6 12:20  pom.xml
drwxr-xr-x. 3 jenkins jenkins   18 Dec  6 12:20  src
drwxr-xr-x. 4 jenkins jenkins   60 Dec  6 12:20  target
```

stage 2: Maven Compilation

Code: https://github.com/betawins/java-Working-app.git

```
[ec2-user@ip-10-0-0-19 target]$ cd /var/lib/jenkins/workspace/maven/target/hiring/
[ec2-user@ip-10-0-0-19 hiring]$ ll
total 4
drwxr-xr-x. 2 jenkins jenkins  6 Dec  6 12:20 META-INF
drwxr-xr-x. 3 jenkins jenkins 36 Dec  6 12:20 WEB-INF
-rw-r--r--. 1 jenkins jenkins 86 Dec  6 12:20 index.jsp
[ec2-user@ip-10-0-0-19 hiring]$ cd /var/lib/jenkins/.m2/repository/in/javahome/hiring/0.1/
[ec2-user@ip-10-0-0-19 0.1]$ ll
total 12
-rw-r--r--. 1 jenkins jenkins  174 Dec  6 12:20 _remote.repositories
-rw-r--r--. 1 jenkins jenkins  890 Dec  6 12:20 hiring-0.1.pom
-rw-r--r--. 1 jenkins jenkins 1698 Dec  6 12:20 hiring-0.1.war
```



5) Use the below code and create a parameterized job in jenkins

stage 1: Git clone

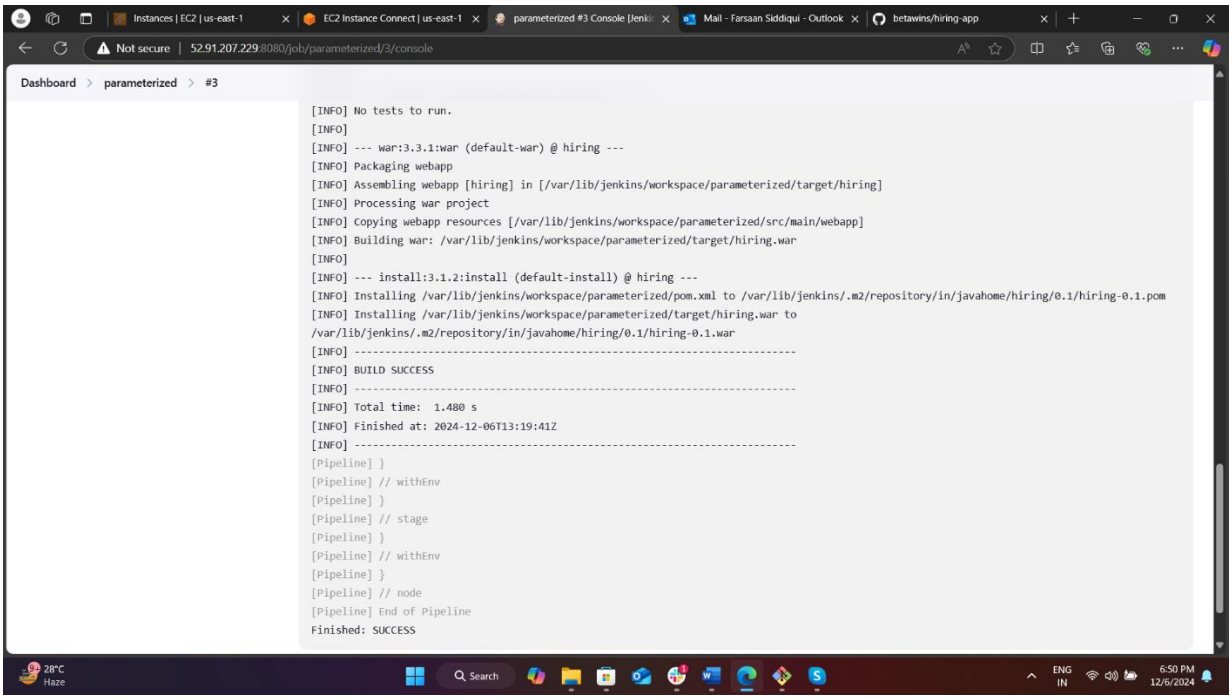
stage 2: Maven Compilation

Code: https://github.com/betawins/java-Working-app.git

>> "NEW ITEM" > SELECT "PIPELINE" > CHECK THE "THIS PROJECT IS PARAMETERIZED" > "ADD PARAMETER" > STRING PARAMETER > NAME= BRANCH > PIPELINE SCRIPT

```
pipeline {
  agent any
  tools {
    maven 'MAVEN' // Name of the Maven installation in Jenkins
  }
  parameters {
    string(name: 'branch', defaultValue: 'main', description: 'Branch to build')
  }
  stages {
    stage('Git Clone') {
      steps {
        git branch: "${params.branch}", url: 'https://github.com/betawins/java-Working-app.git'
      }
    }
    stage('Maven Compilation') {
      steps {
        sh 'mvn clean install'
      }
    }
  }
}
```

>>SAVE > BUILD WITH PARAMETERS >BRANCH = **main** > BUILD



6) What are the global varaiables in jenkins?

- `env`: This variable contains all the environment variables available to the Jenkins job. You can access any environment variable using `env.VARIABLE_NAME`.

- `currentBuild`: This variable provides information about the current build, such as the build number, result, and duration.

- `params`: This variable contains all the parameters passed to the Jenkins job. You can access any parameter using `params.PARAMETER_NAME`.

- `docker`: This variable provides access to Docker-related functions, allowing you to interact with Docker containers and images.

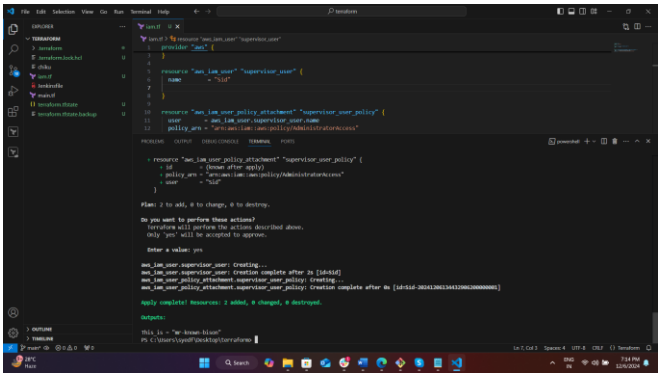
- `scm`: This variable contains information about the source code management (SCM) configuration for the job, such as the Git repository URL and branch.

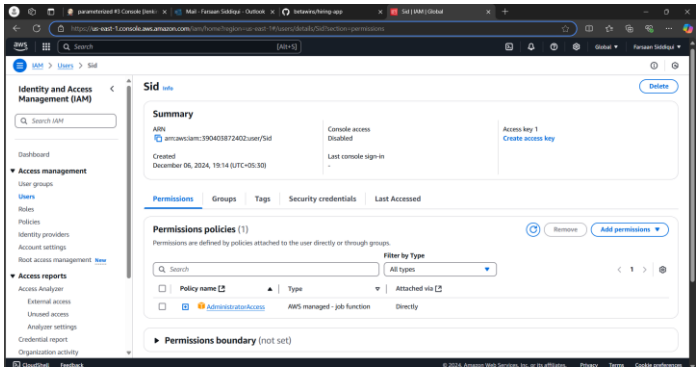
- `pipeline`: This variable provides access to the pipeline script itself, allowing you to control the flow of the pipeline.

- `node`: This variable represents the Jenkins node (agent) on which the job is running.

- `steps`: This variable provides access to the steps available in the pipeline, such as `sh`, `bat`, `echo`, and more

7) Watch terraform-04 video 8) Execute the script shown in video.





9) Integrate terrafrom in jenkins using Terraform plugin.

10) Create CICD pipeline for Nodejs Application.

https://github.com/betawins/Trading-UI.git

11) Explain 10 Maven commands.

1. **mvn clean:**This command eliminates the target directory, holding compiled classes and other built artefacts. This promotes an essentially clean build environment.

2. **mvn compile:**Compiles project's source code. Source files are processed and it puts the compiled.class file in the target folder

3. **mvn test:**These instructions run tests by an already applied framework, such as JUnit, because it compiles a piece of test code and subsequently invokes the test cases for its execution.

4. **mvn package:**This one comes to package the compiled code in a distributable manner such as JAR and WAR files, and deposits this into the target directory

5. **Mvn Install:**This command installs that JAR or WAR to the local Maven repository on which other projects can avail that same machine.

6. **Mvn deploy:**This command moves over all packaged code to a given deployment kind of remote repository where these people could access it by multiple developers and projects. There are also other goals where

7. **Mvn site:**This command makes a site for the project, containing reports and documentation; this is viewable in any web browser.

8. **mvn validate:**This command validates the structure of the project, so it makes sure all of the needed information is present.

9. **mvn dependency:tree**:This command will list the dependencies of the project as they are associated with one another in terms of the relationship.

10. **mvn exec:java:**This command runs a Java program within the Maven project. You need to specify the main class to execute.