PRACTICAL NO. 4

Continuous Integration

LOB3 Understand the concept of CI and how Jenkins automates the process of integrating code changes from multiple contributors.

LO3 Configure Jenkins for automated build and deployment and assess its effectiveness in CI/CD workflows.

- Continuous Integration and Continuous Deployment (CI/CD) tools are essential in modern software development, automating the processes of code integration, testing, and deployment.
- Jenkins is an open-source automation server used for –
    a. continuous integration (CI) and
    b. continuous delivery (CD) in software development.
- It helps developers automate the process of building, testing, and deploying applications.
- In CI/CD, a workflow represents how code is built, tested, and deployed automatically.
- In Jenkins, workflows are automated build and deployment processes that consist of multiple stages (such as fetching code, compiling, testing, and deploying).
- Jenkins Pipeline uses Groovy-based syntax to define automated build, test, and deployment workflows.
- It provides a DSL (Domain-Specific Language) to define these workflows.
- Groovy is a dynamic, object-oriented scripting language for the Java platform. It is like Java but more concise and supports additional features like closures, dynamic typing, and simplified syntax.
- Key Concepts/Components in Jenkins Pipelines
    - Pipeline –
        - A series of steps that define an automation workflow.
        - The pipeline block is mandatory and serves as the root element of a Jenkins Pipeline script.
        - Defines the structure of the pipeline.
    - Agent –
        - The machine where Jenkins runs the pipeline.
    - Stages –
        - Logical divisions of a pipeline, like "Build", "Test", "Deploy".
        - Used to groups multiple stage blocks.
        - stage('Name') → Represents a phase in the pipeline.
    - Steps –
        - Commands executed inside a stage (e.g., echo, bat).
        - Contains commands or actions to execute.
    - Script –

        - Allows Groovy scripting inside Declarative Pipelines.
        - Used to run advanced Groovy logic (loops, conditions, etc.).
    - Post Actions -
        - Running after the pipeline finishes (success, failure).
        - Used to define cleanup actions (e.g., success, failure).
    - Workspace –
        - The directory where Jenkins stores pipeline files.
    - parameters –
        - Allows manual input when triggering the pipeline.
        - Used to define user inputs before execution.
    - triggers –
        - Schedules automatic pipeline execution.
        - Used to define automated execution rules.
    - tools –
        - Defines required software versions.
        - tools → Ensures necessary build tools are available.

- Example -

```
pipeline {
  agent any  // Defines where the pipeline runs

  stages {
    stage('Build') { // Defines a step in the pipeline
      steps {
        echo 'Building the project...'  // Print message to console
      }
    }

    stage('Test') {
      steps {
        echo 'Running tests...'
      }
    }

    stage('Deploy') {
      steps {
        echo 'Deploying the application...'
      }
    }
  }

  post {
```

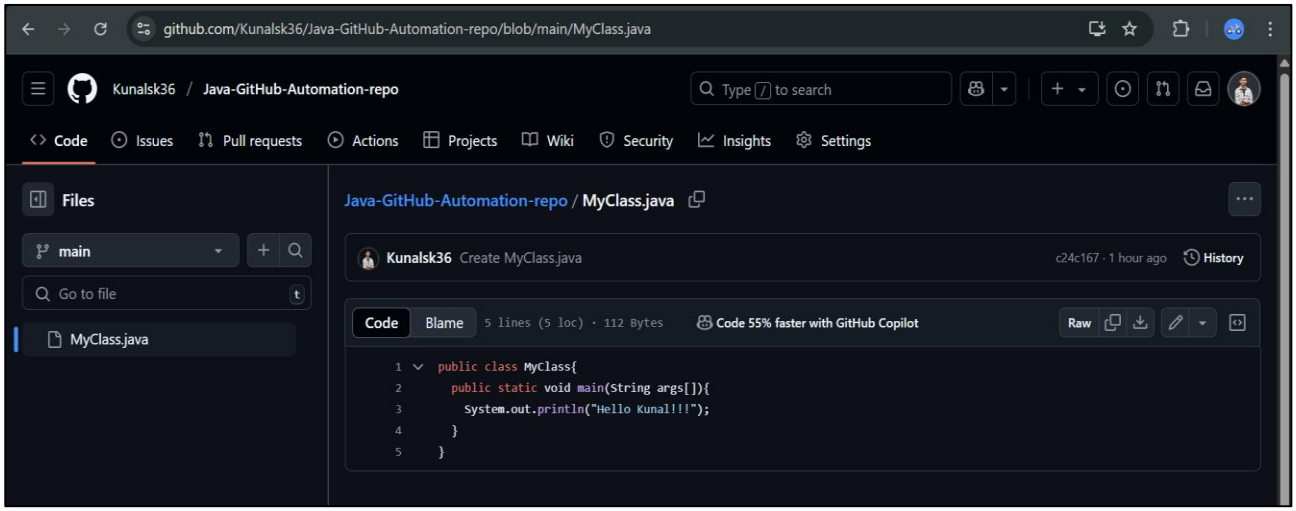
```
        success {
            echo 'Pipeline completed successfully!'  // Runs if the pipeline is successful
        }
        failure {
            echo 'Pipeline failed!'  // Runs if any stage fails
        }
    }
}
```

- Similarly, one can setup a pipeline for the NodeJS project using following steps -
    - Login to Jenkins and click on New Item on dashboard.
    - Assign a name to pipeline and select pipeline as an item's type.
    - Click on Configuration from your pipeline and add following to pipeline script -
    -

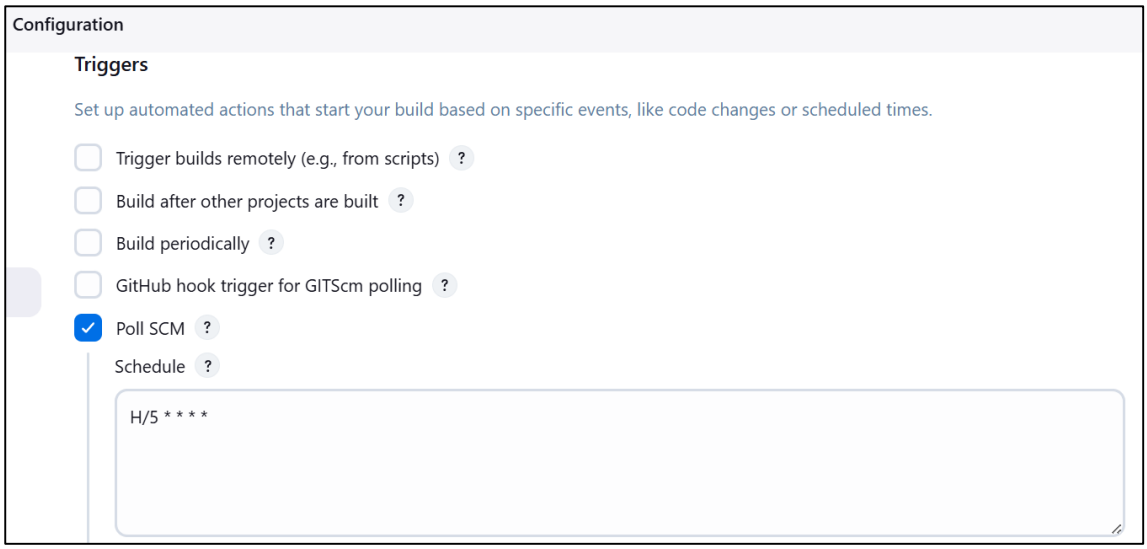
```
pipeline {
    agent any

    tools {nodejs "nodeJSDetls"}

    environment {
        NODE_HOME = "C:\\PROGRA~1\\nodejs"
        PATH = "${NODE_HOME};${env.PATH}"
        PROJECT_DIR = "E:\\NodeJSJenkins"
    }

    stages {
        stage('Install Dependencies') {
            steps {
                script {
                    dir(env.PROJECT_DIR) {
                        bat "npm init -y"
                        bat "npm install"
                        bat "npm install express"
                        bat "echo Starting Node.js application..."
                        bat "node index.js"
                    }
                }
            }
        }
    }
}
```

Exercise:

1. Create a Jenkins project that automates the execution of a simple Java program stored in a GitHub repository. The project should:
    a. Clone the GitHub repository containing the Java program.



    b. Poll the repository every 5 minutes for changes.



c. Compile and run the Java program using "Execute Windows batch command" in Jenkins.

Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Execute Windows batch command ?

Command

See the list of available environment variables

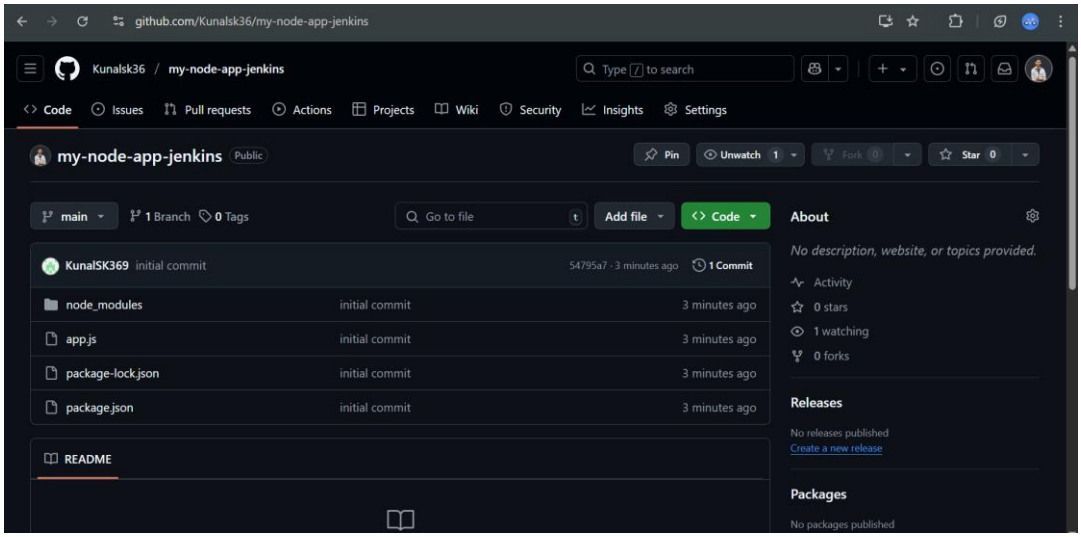
```
echo Cleaning workspace...
del /Q *.class 2>nul

echo Compiling Java program...
javac MyClass.java

echo Running Java program...
java MyClass
```

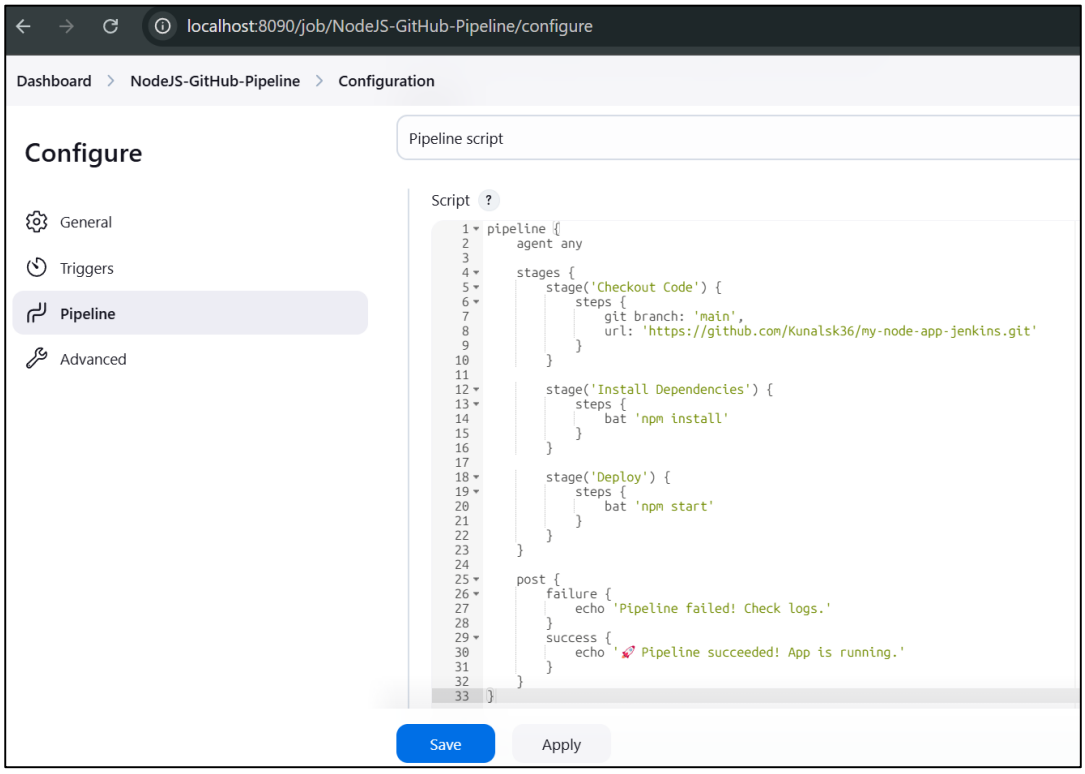
d. Display the program output in the Jenkins Console Output.

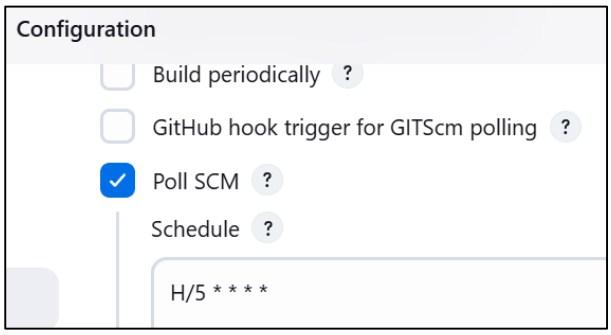
Console Output

```
C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>echo Cleaning workspace...
Cleaning workspace...

C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>del /Q *.class  2>nul

C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>echo Compiling Java program...
Compiling Java program...

C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>javac MyClass.java

C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>echo Running Java program...
Running Java program...

C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>java MyClass
Hello Kunal!!!

C:\ProgramData\Jenkins\.jenkins\workspace\Java-GitHub-Automation>exit 0
Finished: SUCCESS
```

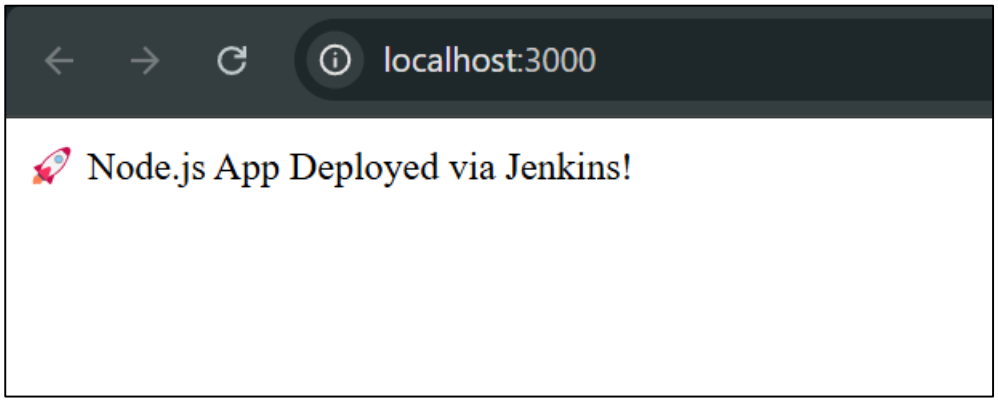
2. Create a Jenkins pipeline to automate the deployment of a simple Node.js application that is managed using GitHub. The pipeline should:
   a. Clone the Node.js application from a GitHub repository.
   b. Install dependencies using npm install.
   c. Deploy the application to a local server.
   d. Automatically trigger the pipeline whenever changes are pushed to the GitHub repository.


github.com/Kunalsk36/my-node-app-jenkins
Kunalsk36 / my-node-app-jenkins
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
my-node-app-jenkins
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
No description, website, or topics provided.
Activity
0 stars
1 watching
0 forks
KunalSK369 initial commit
54795a7 · 3 minutes ago
1 Commit
node_modules
initial commit
3 minutes ago
app.js
initial commit
3 minutes ago
package-lock.json
initial commit
3 minutes ago
package.json
initial commit
3 minutes ago
Releases
No releases published
Create a new release
Packages
No packages published
README

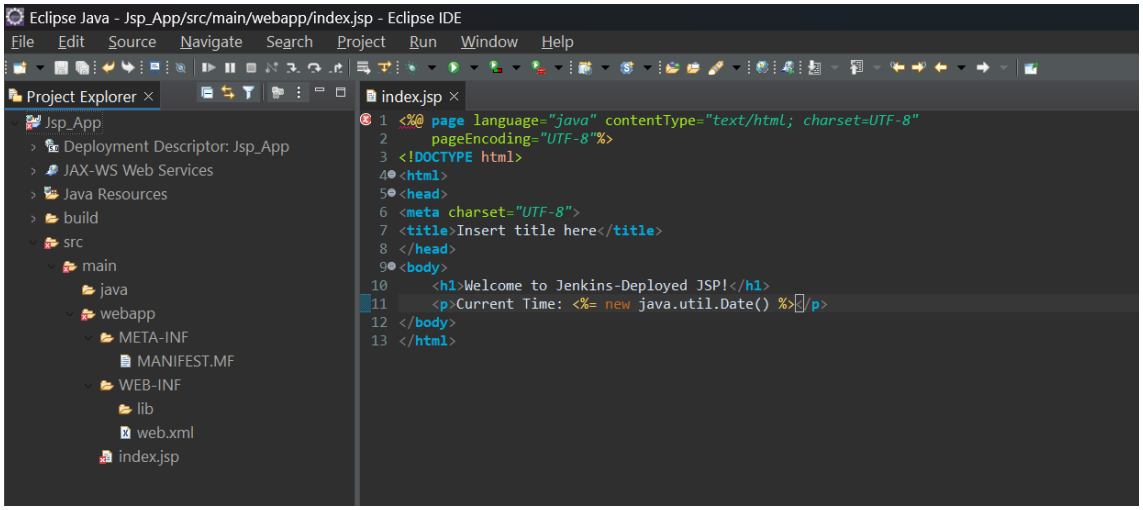

localhost:8090/job/NodeJS-GitHub-Pipeline/configure
Dashboard > NodeJS-GitHub-Pipeline > Configuration
Configure
General
Triggers
Pipeline
Advanced
Pipeline script
Script ?
pipeline {
agent any
stages {
stage('Checkout Code') {
steps {
git branch: 'main',
url: 'https://github.com/Kunalsk36/my-node-app-jenkins.git'
}
}
stage('Install Dependencies') {
steps {
bat 'npm install'
}
}
stage('Deploy') {
steps {
bat 'npm start'
}
}
}
post {
failure {
echo 'Pipeline failed! Check logs.'
}
success {
echo '🚀 Pipeline succeeded! App is running.'
}
}
}
Save
Apply

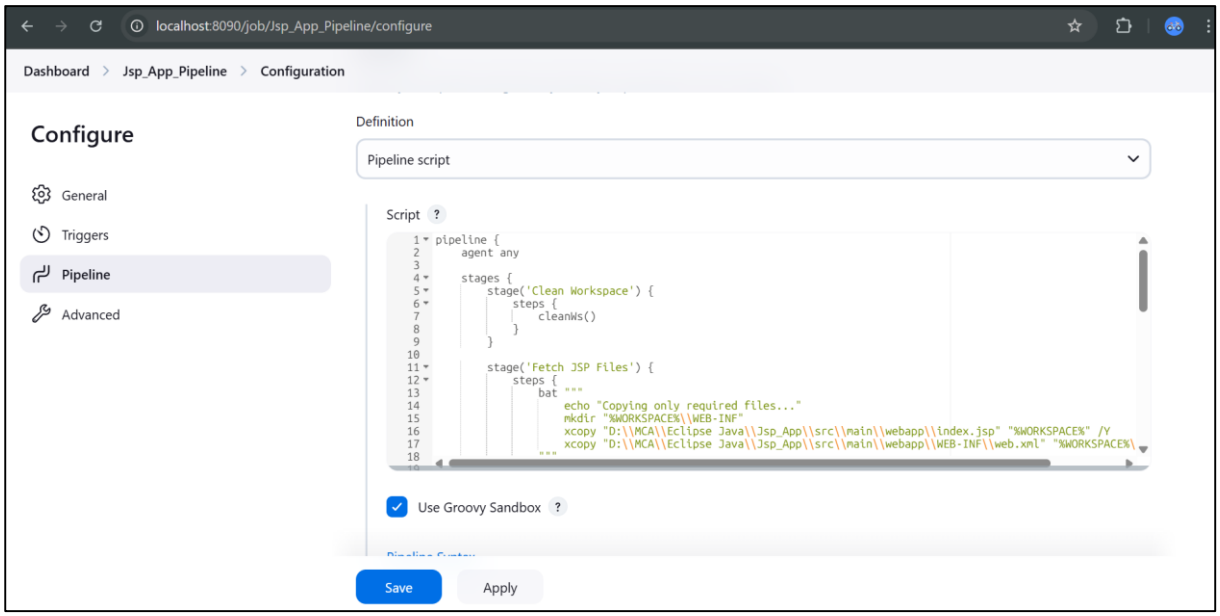

Configuration
Build periodically ?
GitHub hook trigger for GITScm polling ?
Poll SCM ?
Schedule ?
H/5 * * * *

```
C:\ProgramData\Jenkins\.jenkins\workspace\NodeJS-GitHub-Pipeline>npm install

up to date, audited 67 packages in 1s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
Did you forget the `def` keyword? WorkflowScript seems to be setting a field named delegate (to a value of type WorkflowScript)
which could lead to memory leaks or other issues.
Did you forget the `def` keyword? WorkflowScript seems to be setting a field named resolveStrategy (to a value of type Integer)
which could lead to memory leaks or other issues.
[Pipeline] bat

C:\ProgramData\Jenkins\.jenkins\workspace\NodeJS-GitHub-Pipeline>npm start

> my-node-app@1.0.0 start
> node app.js

Server running on http://localhost:3000
```



3. Create a Jenkins pipeline to automate the deployment of a simple JSP application that is stored on the developer's machine. The pipeline should:
    a. Fetch the JSP application from the developer's local machine.
    b. Deploy the application to an Apache Tomcat server on the developer's machine.
    c. Ensure the application is accessible via a web browser.

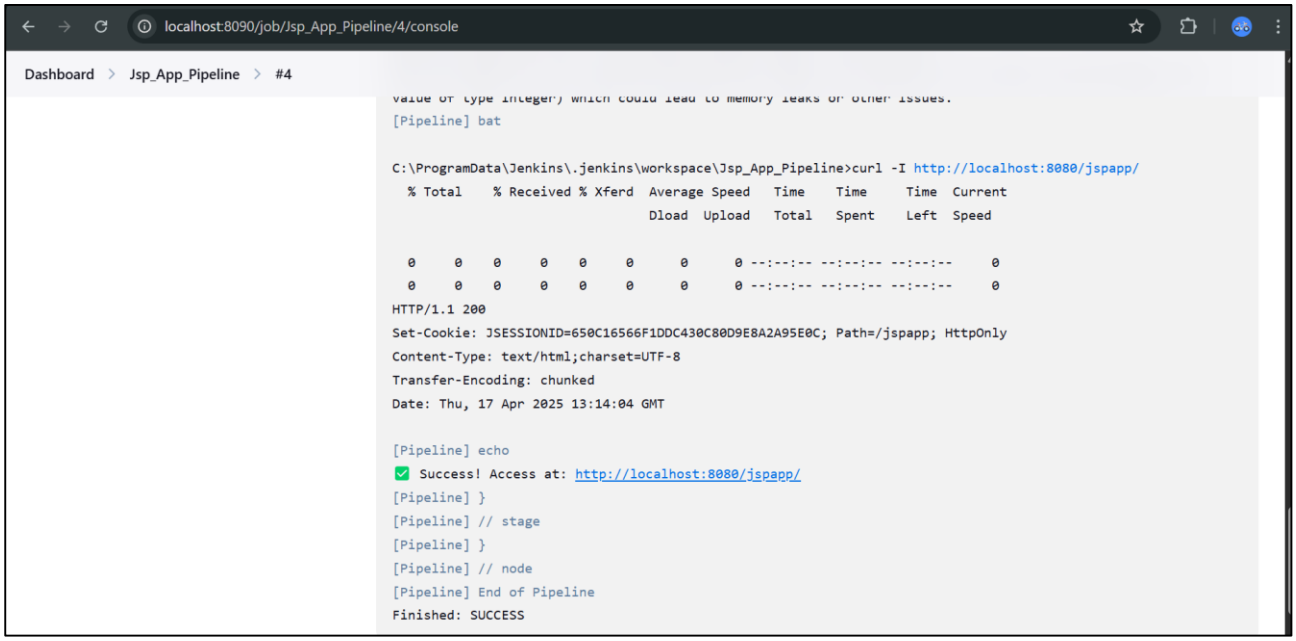




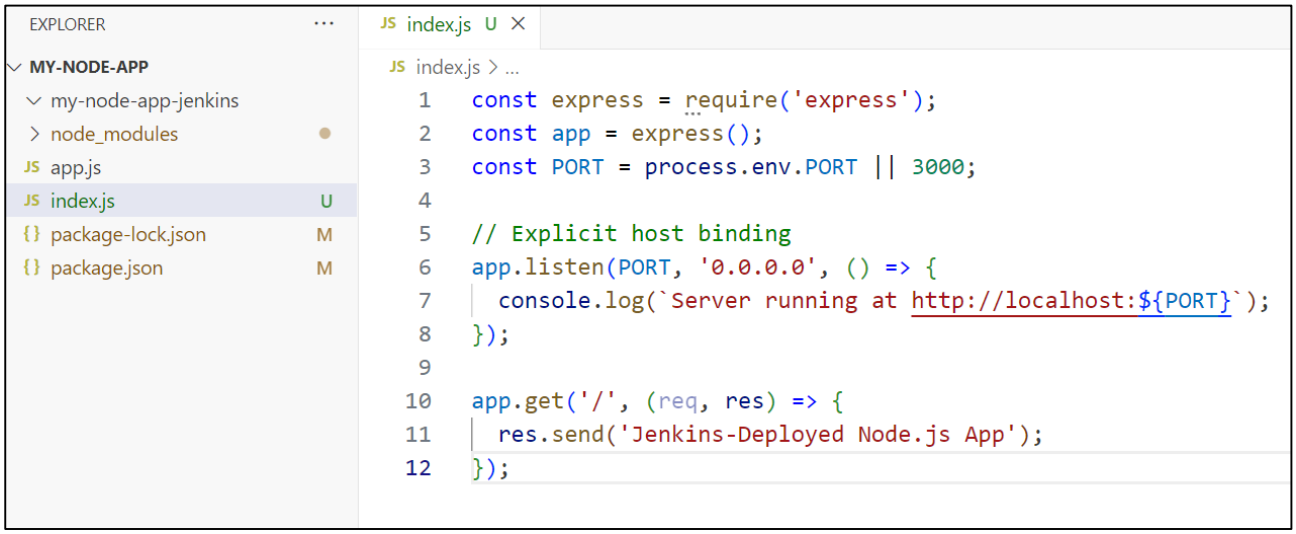
Pipeline Script:

```
pipeline {
   agent any
   stages {
      stage('Clean Workspace') {
         steps {
            cleanWs()
         }
      }
      stage('Fetch JSP Files') {
         steps {
            bat """
               echo "Copying only required files..."
               mkdir "%WORKSPACE%\\WEB-INF"
               xcopy "D:\\MCA\\Eclipse Java\\Jsp_App\\src\\main\\webapp\\index.jsp" "%WORKSPACE%" /Y
               xcopy "D:\\MCA\\Eclipse Java\\Jsp_App\\src\\main\\webapp\\WEB-INF\\web.xml" "%WORKSPACE%\\WEB-INF" /Y
            """
         }
      }
      stage('Build WAR') {
         steps {
            bat """
               echo "Building clean WAR..."
               cd "%WORKSPACE%"
               jar -cvf jsp_app.war *
            """
         }
      }
```

```
    stage('Deploy') {
      steps {
        withCredentials([usernamePassword(
          credentialsId: 'tomcat-admin',
          usernameVariable: 'TOMCAT_USER',
          passwordVariable: 'TOMCAT_PASS'
        )]) {
          bat """
            curl -X PUT --upload-file jsp_app.war ^

"http://%TOMCAT_USER%:%TOMCAT_PASS%@localhost:8080/manager/text/deploy?path=/jspapp"
          """
        }
      }
    }
    stage('Verify') {
      steps {
        bat 'curl -I http://localhost:8080/jspapp/'
        echo "✅ Success! Access at: http://localhost:8080/jspapp/"
      }
    }
  }
}
```



4. Create a Jenkins project that uses "Execute Windows batch command" to run a simple Node.js application stored on the developer's local machine. The project should:
    a. Fetch the Node.js application from the developer's machine.
    b. Install dependencies using npm install.
    c. Start the Node.js application using node index.js.
    d. Ensure the application is running and accessible.
    e. Enable automatic execution whenever changes are made to the application.



Execute Windows batch command ?

Command

See the list of available environment variables

```
echo Kill anything running on port 3000 to avoid conflict
for /f "tokens=5" %%a in ('netstat -a -n -o ^| findstr :3000') do taskkill /PID %%a /F

echo Change to the app directory
cd D:\MCA sem II\DevOps\my-node-app

echo Install dependencies
call npm install

echo Start the app in a new window (non-blocking)
node index.js

echo Optional: Wait a few seconds and check the app is up
timeout /t 5
curl http://localhost:3000
```

Advanced

Console Output

```
C:\ProgramData\Jenkins\.jenkins\workspace\NodeJsExecutableFile>cd D:\MCA sem II\DevOps\my-node-app

C:\ProgramData\Jenkins\.jenkins\workspace\NodeJsExecutableFile>echo Install dependencies
Install dependencies

C:\ProgramData\Jenkins\.jenkins\workspace\NodeJsExecutableFile>call npm install

up to date, audited 68 packages in 1s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Start the app in a new window (non-blocking)
Server running on http://localhost:3000
```

localhost:3000

✨ Node.js App Deployed via Jenkins Executable file!