DevOps Practical

Thinks Check before go futher:

Check That git is install in system:

a) git --version

if not installed visit https://git-scm.com/ and download

b) java --version

Java version should be 8 or 11 or 21

if not there install and set path in environment variable

JAVA_HOME

paste path of bin folder of java 8 or 11 or 21

Step 1: Setup Git repo

1. Create folder in system and save java code into that folder

2. open git bash in that folder and init that folder

1. git init

3. stage java file

1. git add .

4. save and commit changes of file

1. git commit -m "<any message>"

5. Remote this repository on Github

Create Repository on www.github.com and copy link of repo.

1. git remote add origin <paste link here>

6. push folder on Github

1. git push -u origin master or git push -u origin main

Step 2: Download and Setup Jenkins

for mac:

brew install jenkins-lts

Start:

brew services start jenkins-lts

goto:

http://localhost:8080

copy password:

cat /usr/local/var/jenkins/home/secrets/initialAdminPassword

For windows:-

1. Download Jenkins (chose windows and download .msi installer file)
   https://www.jenkins.io/download

2. Install jenkins
   1. Follow the Installation Wizard:
      Select Install suggested plugins when prompted.
      Provide the path for your Java JDK (ensure Java is installed).

   2. Select Installation Folder:
      Choose the default folder or set a custom path for Jenkins.

   2. Set Jenkins Service Port:
      By default, Jenkins runs on port 8080.
   4. get initial password for initial signup in jenkins
      1. go to the secret folder
         C:\Program Files\Jenkins\secrets
      2. Open the file named initialAdminPassword
         copy the password

Step 3: Create a Jenkins Job

1. open Service port
   1. http://localhost:8080

(if any error occur open JENKINS PROBLEM readme file on my github (vanshbadjate07))

2. Create new JOB
   1. Click on new item in dashboard
   2. Enter Job name
   3. Select freestyle project and click ok

3. Configure job
   1. go to Source code management
   2. select git
   3. enter Github repository URL
   4. select branch main or keep master if git branch is master.

3. Build trigger
   1. check Poll SCM to let jenkins checks for changes in git repo
   2. in schedule field, enter:
      H/5 * * * *

4. Build Steps
   1. Scroll to build section
   2. Click Add build section and select Execute Windows batch command and Execute Shell for mac
   3. In command box, enter command to compile java code
      javac <program_name>.java

4. Add another build step to run program
   java <program_name>

5. Save and build the job
   1. Click the Save button at the bottom of the page
   2. You'll be redirected to the job's page.
   3. Click **Build Now** on the left-hand menu.

6. Verify the build output
   1. Go to the **Build History** section on the job's page (bottom-left corner).
   2. Click on the build number (e.g., #1) to open its details.
   3. Select **Console Output** to view the logs.
   4. Verify that:
      The code is pulled from the repository.
      The program is compiled successfully (javac).
      The program runs successfully (java).

Step 4: Automate the build

1. Configure Webhook in GitHub

A webhook is used to notify Jenkins whenever code changes are pushed to the repository.

1. **Open Your GitHub Repository**
   a) Log in to your GitHub account.
   b) Go to your repository where the code is stored.

2. Add a Webhook
   a) Click on the Settings tab in your repository.
   b) Select Webhooks from the left-hand menu.
   c) Click on Add webhook.
   d) In the Payload URL field, enter your Jenkins server's URL with /github-webhook/:
      http://localhost:8080/github-webhook/
   e)Set the Content type to application/json.
   f)Select Just the push event.
   g)Click Add webhook.

(if any error occur like **is not supported because it isn't reachable over the public Internet (localhost)** leave it as it is)

2. Install Github Integration plugin in jenkins
   1. Go to your Jenkins dashboard.
   2. Click Manage Jenkins > Manage Plugins.
   3. In the Available tab, search for GitHub Integration Plugin.
   4. Install it and restart Jenkins if required.

3. Update Jenkins Job Configuration
   Go to your Jenkins job (e.g., HelloWorldBuild).
   Click Configure.

Enable GitHub Trigger
Scroll to the Build Triggers section.
Select GitHub hook trigger for GITScm polling.

Verify Source Code Management
Ensure your repository URL is correctly set under Source Code Management (SCM).
Your credentials should already be configured.

Save the Changes
Click the Save button at the bottom.

4. Test the Automation
1. Go back to your GitHub repository.
2. Make a change to the code.
3. Commit and push the changes to the repository.

5. Verify the Automated Build
1. Go to the Jenkins dashboard.
2. Open the job .
3. Check the Build History section.
4. Verify that a new build is automatically triggered.
5. Open the Console Output of the latest build to confirm:
1. The code was pulled.
2. The build steps (compile, run) executed successfully.

Step 5 : Deploy Application locally

1. Create a Deployment Script
1. Open a Text Editor
Use any text editor.
2. Write the Script
For a Java application, the script might look like this:
On Windows (deploy.bat):

```
@echo off
echo Starting the HelloWorld application...
java <program name>
pause
```

on mac (deploy.sh): -

```
#!/bin/bash
echo "Starting the HelloWorld application..."
java <program name>
```

3. Save the Script
Save the file as deploy.bat (Windows) or deploy.sh (Linux/macOS) in the same directory as your code.

3. Test the Deployment Script Locally
1. Open a terminal or command prompt.
2. Run the script:
• On Windows:
deploy.bat3.
(Verify that the application runs as expected.)

3. Add the Script to Jenkins
1.Go to the Job Configuration
In Jenkins, navigate to the job.
Click Configure.

2. Add a Post-Build Action
Scroll down to the Post-build Actions section.
Click Add post-build action.
Select Execute Windows batch command (on Windows)

3. Enter the Command Windows:
deploy.bat

* Enter Command for mac
chmod +x deploy.sh
./deploy.sh

4. Save the Configuration:
Click Save at the bottom of the page.

4. Test the Deployment Pipeline
1. Push any changes to your Git repository to trigger the Jenkins job automatically.
2. Once the build completes, Jenkins will execute the deployment script as a post-build step.

5. Verify the Deployment
1. Check the Jenkins Console Output for the latest build.
Look for messages indicating the deployment script ran successfully.

2. Ensure the application runs locally as expected.

Step 6: Test the pipeline (CI/CD)
1. Make Changes to the Code
1. Open your project folder.
2. Edit the java file to make a simple change
3. Save the changes.

2. Push Changes to the Git Repository

Open git in your project folder.

Stage, commit, and push the changes:

```
git add HelloWorld.java
git commit -m "Updated message in HelloWorld"
git push origin master
```

3. Observe Jenkins Automation

Open your Jenkins dashboard.

Go to the job.

Verify that a new build is automatically triggered:

Check the Build History section for a new entry.

4. Verify the Build Output

Click on the latest build number in Build History.

Open the Console Output to confirm:

The code changes were pulled from the Git repository.

The application was built successfully (compilation).

The deployment script was executed successfully.

5. Test the Deployed Application

1. Check the output of the application in the terminal or command prompt where it was deployed.

   You should see the updated message

2. If the application is a web service, open the browser and test the local URL.

6. Troubleshoot Any Issues

If the build fails:

Check the Jenkins console output for errors.

Resolve any issues in the code, Git repository, or Jenkins configuration.

If the deployment fails:

Ensure the deployment script is correct and executable.

Verify that the environment (e.g., Java runtime) is properly set up.