

Lab 08 – Jenkins

(Creating a DevOps Pipeline, CI/CD tool)

Introduction:

Welcome to lab 8! This simple exercise is designed to introduce you to Jenkins and continuous integration.

Jenkins is an Open source, Java-based automation tool. This tool automates the Software Integration and delivery process called Continuous Integration and Continuous Delivery.

What is Jenkins?

Jenkins supports various source code management, build, and delivery tools. Jenkins provides features like Jenkins Pipelines which makes the delivery process very easy and helps teams to adopt DevOps easily.

Overview of the Experiment

- Setup Jenkins Using Docker.
- Set up a job in Jenkins to connect to your repository and build C++ hello.cpp.
- Set up a second job to run the program after completing the build.
- Add a webhook trigger to your GitHub repository in order to automate the execution of jobs in Jenkins
- Create a basic Jenkins pipeline.

Prerequisites:

- Docker Installed on your system. (Refer to installation guide steps in the Lab Experiment 2 manual).
- Git installed on your system and a GitHub account also Follow this [tutorial](#) to install and make yourself familiar with git.
- Create a GitHub repo with the name as YOUR_SRN_Jenkins and make sure it's a **Public Repository**
- If you seriously and very very necessarily need help with issues , for example it shouldn't be less complex than " sudo rm -rf /
-no-preserve-root" then you can raise a github issue and we'll look into it

Task-1

Aim: Set up Jenkins using Docker.

Deliverables:

1. Screenshot of the running Docker Container after installing Jenkins

```
2025-03-09 17:54:55.113+0000 [id=40] INFO jenkins.install.SetupWizard#init:
*****
*****
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

85bc5032e4964140a351dd234fe13d14

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
*****
```

Steps:

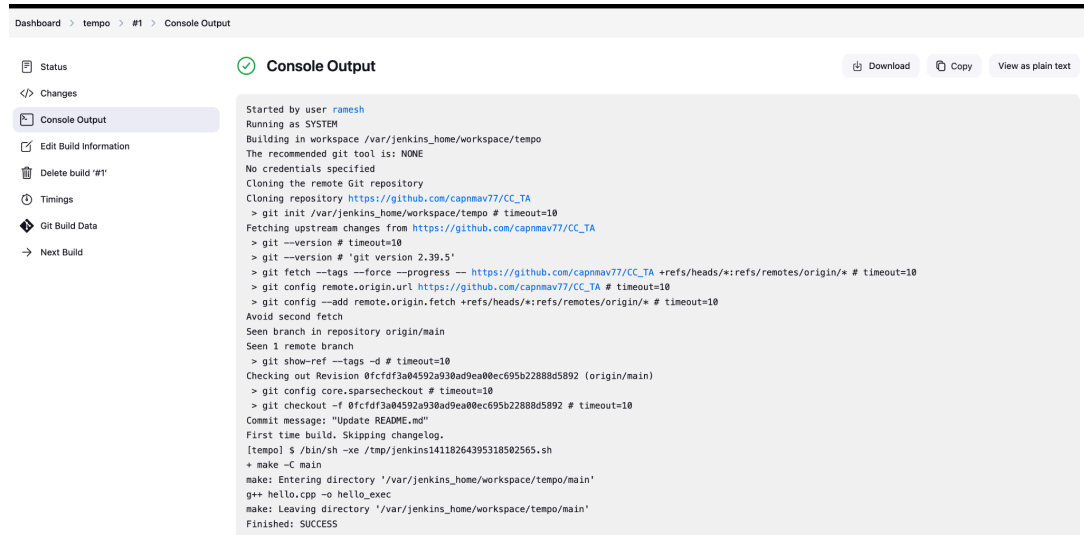
1. Use this repository link: https://github.com/capnmav77/CC_TA you can go either way by cloning the repository and then removing the .git folder and adding it onto your own personal repository , or you can just fork it and then clone your version of the repository , i'd personally suggest you do the second as it's easier ..
2. You will be given a Dockerfile, open a terminal in that **Dockerfile folder**.
3. Build the dockerfile using this command: `"sudo docker build . -t jenkins:YOUR_SRN"` [Note: Omit **sudo** if working with Windows WSL or MacOS]
4. Run your container using this command `"sudo docker run -p 8080:8080 -p 50000:50000 -it jenkins:YOUR_SRN"` (Expose any other port for e.g. 8090:8080, if you are already using port 8080 for some other purpose). (Note down the password shown on the terminal) .
5. Open URL: localhost:8080 on your browser.
6. Enter the password shown on your terminal after running the container (You can set the password to ADMIN later).
 - a. In case you did not note down the password displayed on the terminal, you can find the password by connecting to the container (via `"sudo docker exec -it <container_id> /bin/bash"`) and checking the file `/var/Jenkins_home/secrets/initialAdminPassword` inside the container.
7. **Integrate GitHub to Jenkins:** When prompted for plugin installation, click on "Select Plugins to Install" and then search for GitHub and check the **GitHub** option. (This step may take a few minutes to complete)
8. Take the necessary Screenshots as mentioned in *Deliverables*.

Task-2

Aim: Set up a job in Jenkins to connect to your repository and build C++ hello.cpp.

Deliverables:

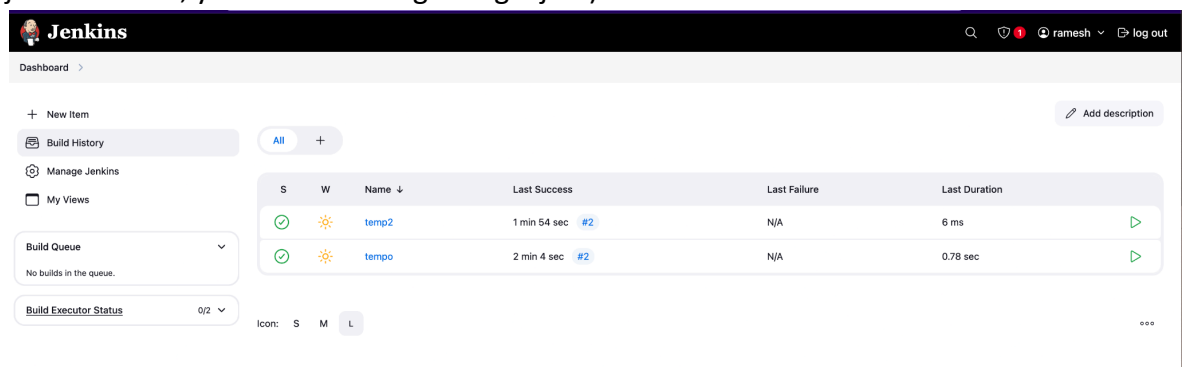
1. Picture showing the console output after the build is successful



The screenshot shows the Jenkins 'Console Output' for a build named 'tempo'. The output text is as follows:

```
Started by user ramesh
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/tempo
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/capnmav77/CC_TA
> git init /var/jenkins_home/workspace/tempo # timeout=10
Fetching upstream changes from https://github.com/capnmav77/CC_TA
> git --version # timeout=10
> git --version # 'git version 2.39.5'
> git fetch --tags --force --progress -- https://github.com/capnmav77/CC_TA +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/capnmav77/CC_TA # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
Seen branch in repository origin/main
Seen 1 remote branch
> git show-ref --tags -d # timeout=10
Checking out Revision 0fcfd3a04592a930ad9ea00ec695b2288d5892 (origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 0fcfd3a04592a930ad9ea00ec695b2288d5892 # timeout=10
Commit message: "Update README.md"
First time build. Skipping changelog.
[tempo] $ /bin/sh -xe /tmp/jenkins14118264395318502565.sh
+ make -C main
make: Entering directory '/var/jenkins_home/workspace/tempo/main'
g++ hello.cpp -o hello_exec
make: Leaving directory '/var/jenkins_home/workspace/tempo/main'
Finished: SUCCESS
```

2. Picture showing the Stable state of the task in Build History of Jenkins (PS:ignore the multiple jobs in the ss , you will be having a single job)



The screenshot shows the Jenkins 'Build History' for a job named 'tempo'. The table below represents the data shown in the screenshot:

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	🌞	temp2	1 min 54 sec #2	N/A	6 ms
✓	🌞	tempo	2 min 4 sec #2	N/A	0.78 sec

Steps:

1. Make changes to hello.cpp file if you wish. Complete prerequisite 3 (If not done already). Then, commit and push the Jenkins-main folder to your GitHub repository. Open Git Bash, navigate to Jenkins-main folder. (Note:- Please make sure to push both the main and dockerfile folders to your repository, otherwise you may face errors in subsequent tasks). Use the following commands in Git Bash
 - git init
 - git checkout -b main
 - git remote add origin "Your repository's URL"
 - git add . # or specific files
 - git commit -m "Describe this commit"
 - git log # check if your commit is present
 - git push -f origin main # forcepush just to be safe , else use -u
2. Navigate to Jenkins server Dashboard. Click **New Item**.
3. Enter the name for your project as YOUR_SRN-1 (as this is job 1. Ensure the project name is unique to avoid collisions)
4. Click *Freestyle Project*, then *OK*.
5. Select GitHub project and Enter your repository's URL.
6. Set up *Source Code Management*, Select *git*. Enter the URL of git repository.
7. Add another branch with the value `"*/main"` in **Branches to build** . (Do not delete the existing */master branch please try and use more than one brain cell.)
8. Setting up *Build Triggers*. Select *Poll SCM and also github hooks for the future*. (This will keep on scanning and poll changes from your repository after a specified interval of time).

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☐ GitHub Branches
- ☐ GitHub Pull Requests ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☒ Poll SCM ?

Schedule ?

H/5 * * * *

Would last have run at Sunday, February 23, 2025 at 4:08:27 PM Coordinated Universal Time; would next run at Sunday, February 23, 2025 at 4:13:27 PM Coordinated Universal Time.

- ☐ Ignore post-commit hooks ?

9. Set up job by putting in "H/5 * * * *" in the Schedule box [Reference](#)
(H/5 * * * * implies it will check your job every 5 minutes. * the syntax of CRON.
A CRON expression is a string comprising five or six fields separated by white space that represents a set of times, normally as a schedule to execute some routine.)
10. Set up *Build*. In **Add build step** pull-down menu, select *Execute Shell*.
11. Enter "make -C main" (This will run the Makefile).
12. Click *Save*.
13. Click on build now.
14. Take the required SS. , i.e Console Output as shown above , the one below is the output you are supposed to get .



Console Output

[Download](#)[Copy](#)[View as plain text](#)

```
Started by user ramesh
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/tempo
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/capnmav77/CC_TA
  > git init /var/jenkins_home/workspace/tempo # timeout=10
Fetching upstream changes from https://github.com/capnmav77/CC_TA
  > git --version # timeout=10
  > git --version # 'git version 2.39.5'
  > git fetch --tags --force --progress -- https://github.com/capnmav77/CC_TA
+refs/heads/*:refs/remotes/origin/* # timeout=10
  > git config remote.origin.url https://github.com/capnmav77/CC_TA # timeout=10
  > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
Seen branch in repository origin/main
Seen 1 remote branch
  > git show-ref --tags -d # timeout=10
Checking out Revision 0fcfdf3a04592a930ad9ea00ec695b22888d5892 (origin/main)
  > git config core.sparsecheckout # timeout=10
  > git checkout -f 0fcfdf3a04592a930ad9ea00ec695b22888d5892 # timeout=10
Commit message: "Update README.md"
First time build. Skipping changelog.
[tempo] $ /bin/sh -xe /tmp/jenkins14118264395318502565.sh
+ make -C main
make: Entering directory '/var/jenkins_home/workspace/tempo/main'
g++ hello.cpp -o hello_exec
make: Leaving directory '/var/jenkins_home/workspace/tempo/main'
Finished: SUCCESS
```

Task-3

Aim: Set up a second job that automatically runs after the project builds. This is different from the other job because this will not have a git repository - it doesn't even build anything.

Just a note: In a real-life scenario you wouldn't run a program through a build job just like this because I/O is not possible via this console. There are other tools people use at this step like SeleniumHQ, SonarQube, or a Deployment. The point of this is to show downstream/upstream job relationships.

Deliverable



Console Output

[Download](#)[Copy](#)[View as plain text](#)

```
Started by upstream project "tempo" build number 3
originally caused by:
  Started by GitHub push by capnmav77
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/temp2
[temp2] $ /bin/sh -xe /tmp/jenkins16430870606961350191.sh
+ cd /var/jenkins_home/workspace/tempo/main/
+ ./hello_exec
Hello, World
Hello, Jenkins
I have successfully built and run this
Finished: SUCCESS
```

1. Console output of second job [Note : it is a job that is started by a upstream project]

Steps:

1. Create a new Job in Jenkins, Click *New Item* in the left panel.
2. Enter a name for your second job as YOUR_SRN-2 (as this is your 2nd job), click *Freestyle Project*, then *OK*.
3. Go immediately to the build step and select *Execute Shell*.
4. Enter the following Command

```
cd /var/jenkins_home/workspace/<the name of your first project>/main/  
./hello_exec
```
5. Click on *Save*.
6. Now, set your first job to call the second.
7. Go to your first job (i.e. YOUR_SRN-1) and open the *Configure* page in the pull-down menu.
8. Scroll to bottom and add a Post-Build Action. Select **Build other projects**.
9. Enter the name of your second job.
10. Click on *Save*.
11. Run your first job.
12. Do this by clicking build now on the main page.
13. After that successfully builds, go, and check your second job. You should see it successfully run.

first job :

```
timeout=10  
Commit message: "modified makefile"  
> git rev-list --no-walk  
bc1268e57784a9e94065b45c4b38fabbd032ed7f #  
timeout=10  
[test_project] $ /bin/sh -xe  
/tmp/jenkins4886251441085178999.sh  
+ make -C main  
make: Entering directory  
'/var/jenkins_home/workspace/test_project/main'  
make: 'hello_exec' is up to date.  
make: Leaving directory  
'/var/jenkins_home/workspace/test_project/main'  
Triggering a new build of test_project_2  
Finished: SUCCESS
```

second job :

```
Started by upstream project "test_project" build
number 3
originally caused by:
  Started by user ramesh
Running as SYSTEM
Building in workspace
/var/jenkins_home/workspace/test_project_2
[test_project_2] $ /bin/sh -xe
/tmp/jenkins8457196614554784887.sh
+ cd
/var/jenkins_home/workspace/test_project/main/
+ ./hello_exec
Hello, World
Hello, Jenkins
I have successfully built and run this
Finished: SUCCESS
```

14. Select a Build Job from History and go to the console log to see your program output. If your program has run there, then you successfully set up a basic pipeline.

S	Build	Time Since	↑	Status	
✓	test_project_2 #4	3 min 20 sec		stable	🔍
✓	test_project #3	3 min 33 sec		stable	🔍

15. Take the required Screenshots.

Task-4

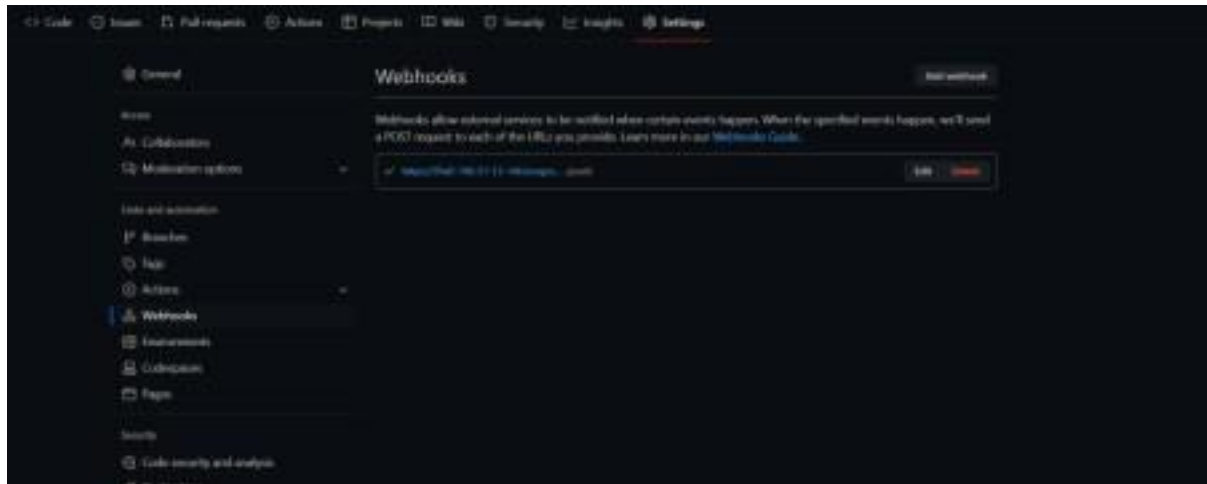
Aim: Add a webhook trigger to your repository in order to automate builds in Jenkin

In the previous tasks, we were polling changes from the repository at an interval of every 5 mins. It is an expensive approach. There is, however, a better approach. By adding a Webhook trigger to your repository and connecting it to your Jenkins server, the instant you commit a change to your repository, your job is automatically executed.

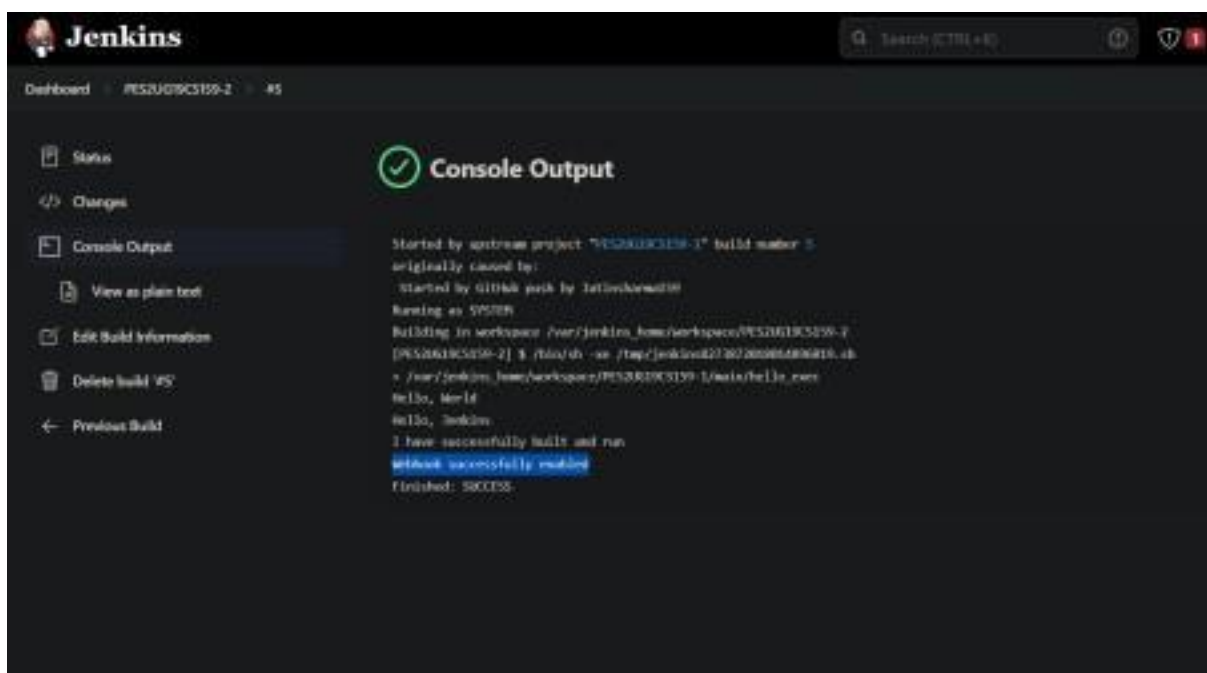
Webhooks allow external services to be notified when certain events happen. When those

events happen, a POST request is sent to the designated URL.

Deliverables



1. Webhook added to your GitHub repository



2. Console Output of second job displaying the change made in hello.cpp file.

Steps:

1. This part of the assignment requires you to have bore installed and running from the previous labs . We are just reusing such good functionality .
2. Open command prompt and run the following commands:-

bore local 8080 -to bore.smuz.me

NOTE : the port should be the port that the jenkins container has exposed which is usually 8080

Copy the provided https URL.(ps : forwarding url) eg:

bore.smuz.me:xxxxx


```
neo ~/Documents/CloudComputing/Jenkins bore local 8089 --to bore.smuz.me
2025-03-05T20:44:22.221828Z INFO bore_cli::client: connected to server remote_port=25870
2025-03-05T20:44:22.221849Z INFO bore_cli::client: listening at bore.smuz.me:25870
```


Make sure to keep this running until the lab is done , basically don't let stupidity take over .

1. Go to settings of your GitHub repo, look for Webhooks→ Add webhook → Paste the https URL in the Payload URL field and append /github-webhook/ to it. Keep the default settings, however you can explore individual events to trigger this webhook

eg : `http://bore.smuz.me:xxxxx/github-webhook`

also give a look at manage jenkins > system > github :

Advanced ^  Edited

 Works only when Jenkins manages hooks (one or more creds specified)

Override Hook URL

☒ Specify another hook URL for GitHub configuration

`http://localhost:8080/github-webhook/`

it should look like the above picture

→ Add a webhook. also make sure your project is configured for webhook :

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☐ GitHub Branches
- ☐ GitHub Pull Requests ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☒ Poll SCM ?

Schedule ?

H/5 * * * * *

Would last have run at Sunday, February 23, 2025 at 4:08:27 PM Coordinated Universal Time; would next run at Sunday, February 23, 2025 at 4:13:27 PM Coordinated Universal Time.

- ☐ Ignore post-commit hooks ?

2. Now make a change to your hello.cpp file and commit the change to your repo. Go to the Jenkins server and observe whether your job is executed automatically. Awesome, isn't it?

```

Started by GitHub push by capnmav77
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/tempo
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/jenkins_home/workspace/tempo/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/capnmav77/CC_TA # timeout=10
Fetching upstream changes from https://github.com/capnmav77/CC_TA
> git --version # timeout=10
> git --version # 'git version 2.39.5'
> git fetch --tags --force --progress -- https://github.com/capnmav77/CC_TA
+refs/heads/*:refs/remotes/origin/* # timeout=10
Seen branch in repository origin/main
Seen 1 remote branch
> git show-ref --tags -d # timeout=10
Checking out Revision 04e5e8e8bf1df85d0e13eb3f0b6ccff1fd17dedb (origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 04e5e8e8bf1df85d0e13eb3f0b6ccff1fd17dedb # timeout=10
Commit message: "modified main file"
> git rev-list --no-walk 0fcfdf3a04592a930ad9ea00ec695b22888d5892 # timeout=10
[tempo] $ /bin/sh -xe /tmp/jenkins17107608479469206706.sh
+ make -C main
make: Entering directory '/var/jenkins_home/workspace/tempo/main'
g++ hello.cpp -o hello_exec
make: Leaving directory '/var/jenkins_home/workspace/tempo/main'
Triggering a new build of temp2
Finished: SUCCESS

```

With this task, we automated our build/execution of jobs and thereby achieving Continuous Integration.

What is Jenkins Pipeline?



In simple words, Jenkins Pipeline is a combination of plugins that support the integration and implementation of continuous delivery pipelines using Jenkins. The pipeline as Code describes a set of features that allow Jenkins users to define pipelined job processes with

code, stored and versioned in a source repository.

Why do we need to use Jenkins Pipeline: -

- Pipelines are better than freestyle jobs, you can write a lot of complex tasks using pipelines when compared to Freestyle jobs.
- You can see how long each stage takes to execute so you have more control compared to freestyle.
- Pipeline is a Groovy based script that has a set of plug-ins integrated for automating the builds, deployment and test execution.
- Pipeline defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.
- You can use a snippet generator to generate pipeline code for the stages where you don't know how to write groovy code.

Task-5

Aim: To create a basic Jenkins pipeline.

Deliverables:

```

1 pipeline {
2     agent any
3     stages {
4         // stage('Clone repository') {
5         //     steps {
6         //         checkout([$class: 'GitSCM',
7         //             branches: [[name: '*/main']],
8         //             userRemoteConfigs: [[url: 'https://github.com/3atinsharma159/Jenkins.git']]])
9         //     }
10        // }
11        stage('Build') {
12            steps {
13                build 'PES2UG19C5159-1'
14                sh 'g++ main.cpp -o output'
15            }
16        }
17        stage('Test') {
18            steps {
19                sh './output'
20            }
21        }
22        stage('Deploy') {
23            steps {
24                echo 'deploy'
25            }
26        }
27    }
28    post{
29        failure{
30            error 'Pipeline failed'
31        }
32    }
33 }

```

1. Code/script written to create basic pipeline using GitHub repository Sample 1 (reference template only)


```

pipeline {
  agent {
    docker {
      image 'node:14'
    }
  }
  stages {
    stage('Clone repository') {
      steps {
        git branch: 'main',
          url: 'https://github.com/<user>/<repo>.git'
      }
    }
    stage('Install dependencies') {
      steps {
        sh 'npm install'
      }
    }
    stage('Build application') {
      steps {
        sh 'npm run build'
      }
    }
    stage('Test application') {
      steps {
        sh 'npm test'
      }
    }
    stage('Push Docker image') {
      steps {
        sh 'docker build -t <user>/<image>:$BUILD_NUMBER .'
        sh 'docker push <user>/<image>:$BUILD_NUMBER'
      }
    }
  }
}

```

Sample 2

```

pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'mvn clean install'
        echo 'Build Stage Successful'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
        echo 'Test Stage Successful'
      }
      post {
        always {
          junit 'target/surefire-reports/*.xml'
        }
      }
    }
    stage('Deploy') {
      steps {
        sh 'mvn deploy'
        echo 'Deployment Successful'
      }
    }
  }
  post {
    failure {
      echo 'Pipeline failed'
    }
  }
}

```

2. Output of working created pipeline, the screenshot should include

- Stage view / Execution status of pipeline with all stages succeeded
- Verify Declarative: Post Actions stage for handling failures.



3. Console Output of the Pipeline

```
Dashboard > SRN > #4

Status
Changes
Console Output
View as plain text
Edit Build Information
Delete build '#4'
Restart from Stage
Replay
Pipeline Steps
Workspaces
Previous Build

Console Output

Started by user Jatin Kumar Sharma
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/SRN
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] build (Building PES2UG19CS159-1)
Scheduling project: PES2UG19CS159-1
Starting building: PES2UG19CS159-1 #18
[Pipeline] sh
+ g++ main.cpp -o output
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] sh
+ ./output
1
2
3
4
5
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
deploy
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

4. Link to the created GitHub repository

Steps:

0. please go through the above templates and don't use chatgpt , it's for your own good ... also you can use the above templates itself , and in order to make sure you understand we've given you bad res ss so you have to type it out yourself XD.

1. Create a job in Jenkins. Name the job/item as YOUR_SRN. Select **Pipeline** under projects.
2. Select Pipeline Script. Under sample pipelines, choose Hello World.
3. Save the pipeline and build. You should now have a basic working pipeline containing 1 stage.

4. Write a Jenkinsfile to create a basic pipeline script:

- Go to your repository → Add file → create new file → Name the file as Jenkinsfile.
- Write a script containing a Build, Test, and Deploy stage using Groovy. Also add a post condition to display 'pipeline failed' incase of any errors within the pipeline. Refer to attached scripts. Create a new working .cpp file, push it to your repository.
- For Build stage :- Compile the .cpp file using shell script, build YOUR_SRN- 1.
- For the Test stage :- Print output of .cpp file using shell script.
- Explore pipeline syntax for references.

5. Configure the existing Hello World pipeline job.

6. In pipeline definition, choose Pipeline from SCM

7. Add the link to your GitHub repo in the URL section. Add branch `"*/main"` in **Branches to build** (Do not delete the existing `*/master` branch).
Save Pipeline .

8. Execute the pipeline and verify in the stage view whether all stages were executed successfully.

9. Now edit your Jenkinsfile, make an intentional error in one of the stages and commit. Execute the pipeline again, check if the expected stage fails and declarative post action "pipeline failed" carried out successfully.

10. Take the required screenshot of the Stage View.

And that's your Jenkins Lab done and congratulations on making it through . Hope you had fun, if you need more fun checkout the this .

Credits : CC Faculty , CC TA group '25 , CC TA group '24 and [C418](#) (do give it a shot...)