Assignment 4(a) – Jenkins (Creating a DevOps Pipeline, CI/CD tool)

Introduction:

Welcome to Assignment 4! 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 the build completes.
- Add a webhook trigger to your GitHub repository in order to automate execution of jobs in Jenkins
- Create a basic Jenkins pipeline.

Prerequisites:

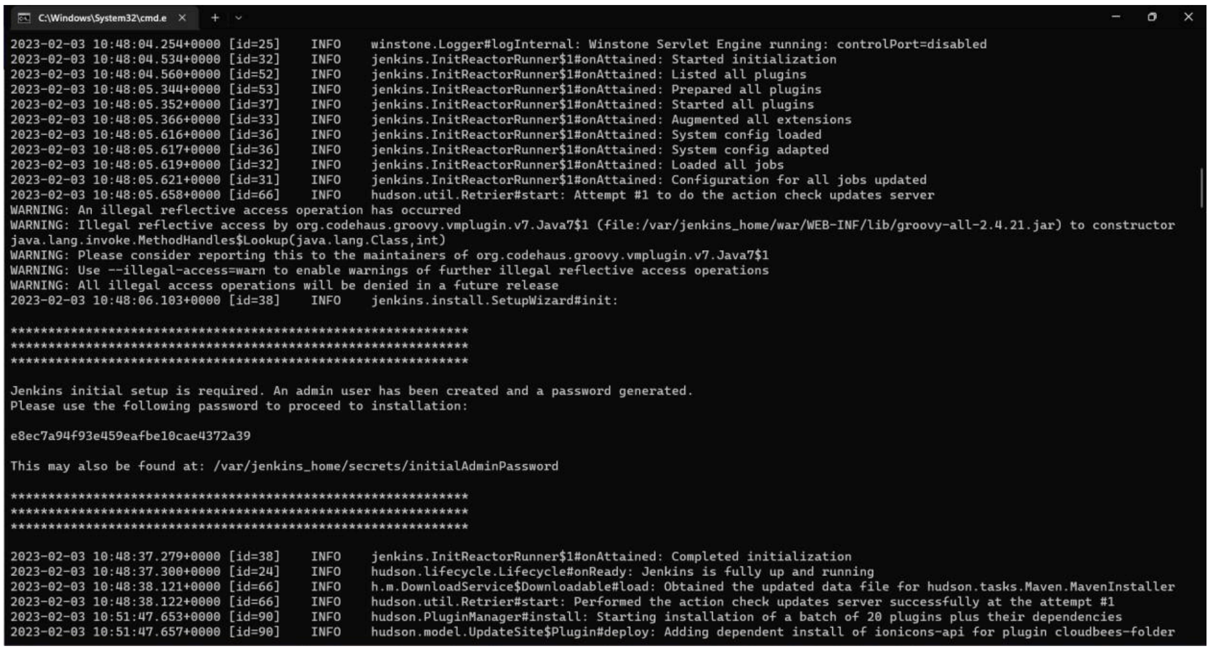
- Docker Installed on your system. (Refer to installation guide steps in the Lab Experiment 2 manual).
- Git installed on your system and a GitHub account

  Follow this tutorial to install and make yourself familiar with git
  https://www.youtube.com/watch?v=2j7fD92g-gE
- Create a GitHub repo with the name as YOUR_SRN_Jenkins

Task-1

Aim: Set up Jenkins using Docker.

Deliverables:



```
2023-02-03 10:48:04.254+0000 [id=25]    INFO    winstone.Logger#logInternal: Winstone Servlet Engine running: controlPort=disabled
2023-02-03 10:48:04.534+0000 [id=32]    INFO    jenkins.InitReactorRunner$1#onAttained: Started initialization
2023-02-03 10:48:04.560+0000 [id=52]    INFO    jenkins.InitReactorRunner$1#onAttained: Listed all plugins
2023-02-03 10:48:05.344+0000 [id=53]    INFO    jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
2023-02-03 10:48:05.352+0000 [id=37]    INFO    jenkins.InitReactorRunner$1#onAttained: Started all plugins
2023-02-03 10:48:05.366+0000 [id=33]    INFO    jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
2023-02-03 10:48:05.616+0000 [id=36]    INFO    jenkins.InitReactorRunner$1#onAttained: System config loaded
2023-02-03 10:48:05.617+0000 [id=36]    INFO    jenkins.InitReactorRunner$1#onAttained: System config adapted
2023-02-03 10:48:05.619+0000 [id=32]    INFO    jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
2023-02-03 10:48:05.621+0000 [id=31]    INFO    jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs updated
2023-02-03 10:48:05.658+0000 [id=66]    INFO    hudson.util.Retrier#start: Attempt #1 to do the action check updates server
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1 (file:/var/jenkins_home/war/WEB-INF/lib/groovy-all-2.4.21.jar) to constructor
java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)
WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.vmplugin.v7.Java7$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2023-02-03 10:48:06.103+0000 [id=38]    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:

e8ec7a94f93e459eafbe10cae4372a39

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

*************************************************************
*************************************************************
*************************************************************

2023-02-03 10:48:37.279+0000 [id=38]    INFO    jenkins.InitReactorRunner$1#onAttained: Completed initialization
2023-02-03 10:48:37.300+0000 [id=24]    INFO    hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
2023-02-03 10:48:38.121+0000 [id=66]    INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
2023-02-03 10:48:38.122+0000 [id=66]    INFO    hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt #1
2023-02-03 10:51:47.653+0000 [id=90]    INFO    hudson.PluginManager#install: Starting installation of a batch of 20 plugins plus their dependencies
2023-02-03 10:51:47.657+0000 [id=90]    INFO    hudson.model.UpdateSite$Plugin#deploy: Adding dependent install of ionicons-api for plugin cloudbees-folder
```

1. Screenshot of the running Docker Container after installing Jenkins

Steps:

1. Use this repository link: https://github.com/ectagithub/Jenkins_lab and download the zip file, extract the Jenkins_lab-main folder.
2. You will be given a Dockerfile, open a terminal in that 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

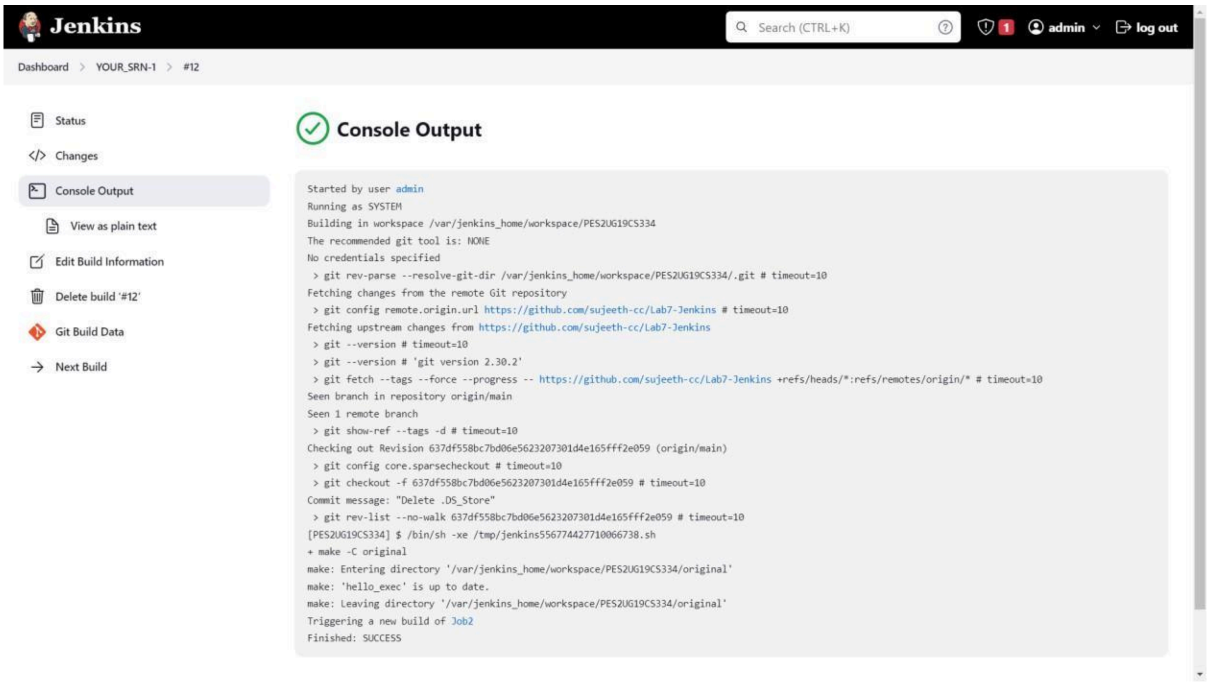
–it <container_id> /bin/bash") and checking the file /var/Jenkins_home /secrets/initialAdminPassword inside the container.

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

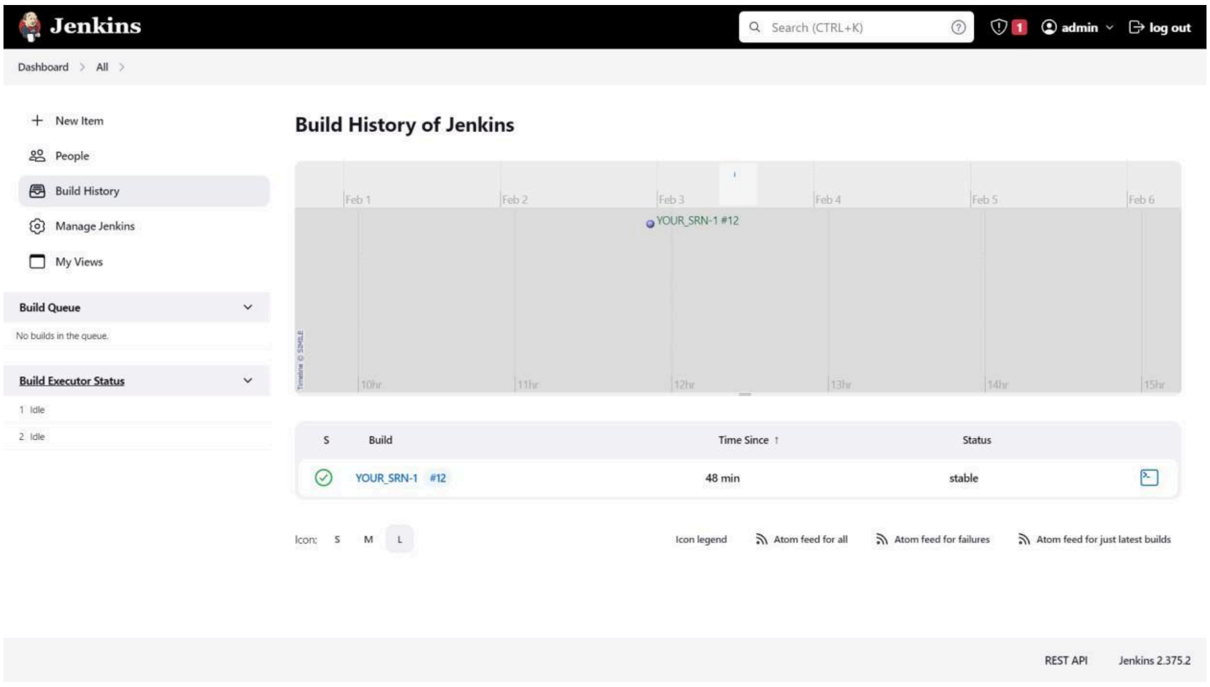
Aim: Set up a job in Jenkins to connect to your repository and build C++ hello.cpp.

Deliverables:



```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/PES2UG19CS334
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/jenkins_home/workspace/PES2UG19CS334/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/sujeeth-cc/Lab7-Jenkins # timeout=10
Fetching upstream changes from https://github.com/sujeeth-cc/Lab7-Jenkins
 > git --version # timeout=10
 > git --version # 'git version 2.30.2'
 > git fetch --tags --force --progress -- https://github.com/sujeeth-cc/Lab7-Jenkins +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 637df558bc7bd06e5623207301d4e165fff2e059 (origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 637df558bc7bd06e5623207301d4e165fff2e059 # timeout=10
Commit message: "Delete .DS_Store"
 > git rev-list --no-walk 637df558bc7bd06e5623207301d4e165fff2e059 # timeout=10
[PES2UG19CS334] $ /bin/sh -xe /tmp/jenkins556774427710066738.sh
+ make -C original
make: Entering directory '/var/jenkins_home/workspace/PES2UG19CS334/original'
make: 'hello_exec' is up to date.
make: Leaving directory '/var/jenkins_home/workspace/PES2UG19CS334/original'
Triggering a new build of Job2
Finished: SUCCESS
```

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


Jenkins | Search (CTRL+K) | admin | log out

Dashboard > All >

Build History of Jenkins

S	Build	Time Since ↑	Status
✓	YOUR_SRN-1 #12	48 min	stable

2. Picture showing the Stable state of the task in Build History of Jenkins

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 .
   - git commit -m "Describe this commit"
   - git log
   - git push -f origin main

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)

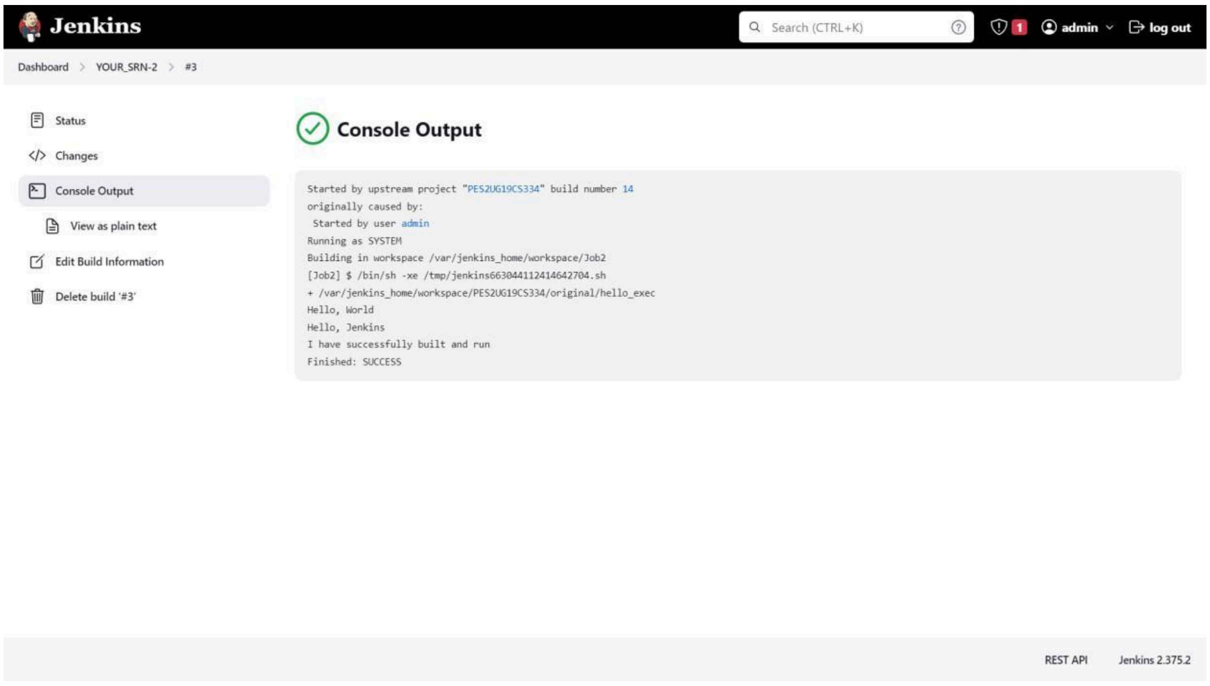
8. Setting up *Build Triggers.* Select *Poll SCM*. (This will keep on scanning and poll changes from your repository after a specified interval of time).
9. Set up job by putting in "`H/5 * * * *`" in the Schedule box
(H/5 * * * * implies it will check your job every 5 minutes. * the syntax of CRON. 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.

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.

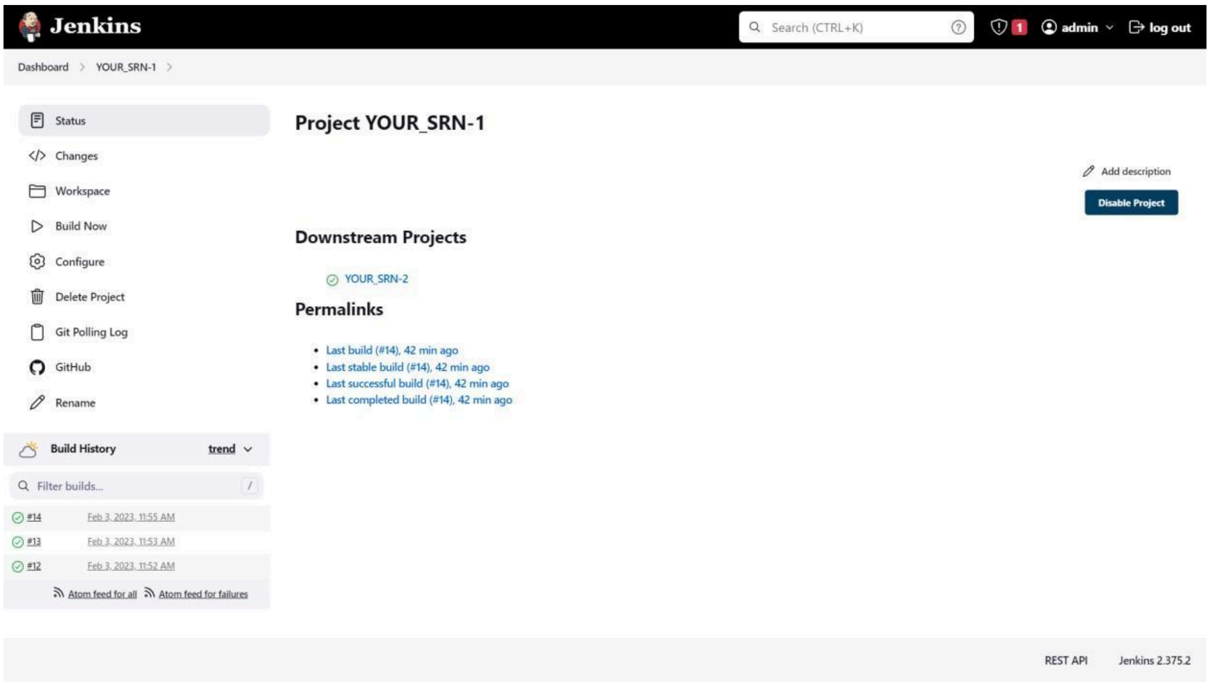
Deliverables



Console Output

```
Started by upstream project "PES2UG19CS334" build number 14
originally caused by:
 Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/Job2
[Job2] $ /bin/sh -xe /tmp/jenkins6630441124146427O4.sh
+ /var/jenkins_home/workspace/PES2UG19CS334/original/hello_exec
Hello, World
Hello, Jenkins
I have successfully built and run
Finished: SUCCESS
```

1. *Console output of second job*



Project YOUR_SRN-1

Downstream Projects

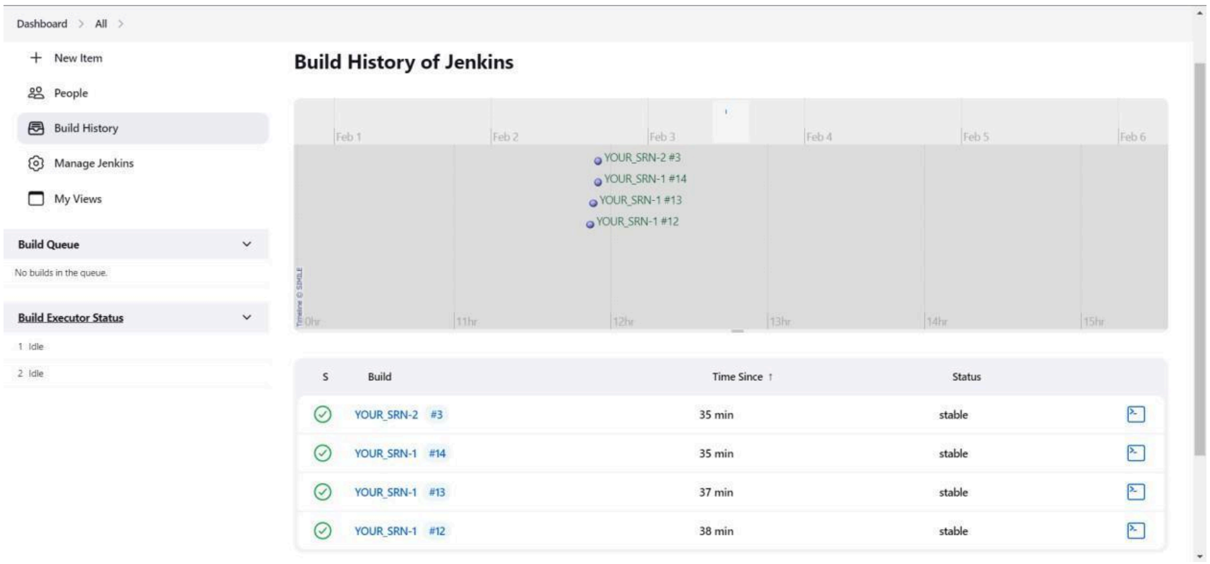
YOUR_SRN-2

Permalinks

- Last build (#14), 42 min ago
- Last stable build (#14), 42 min ago
- Last successful build (#14), 42 min ago
- Last completed build (#14), 42 min ago

#14	Feb 3, 2023, 11:55 AM
#13	Feb 3, 2023, 11:53 AM
#12	Feb 3, 2023, 11:52 AM

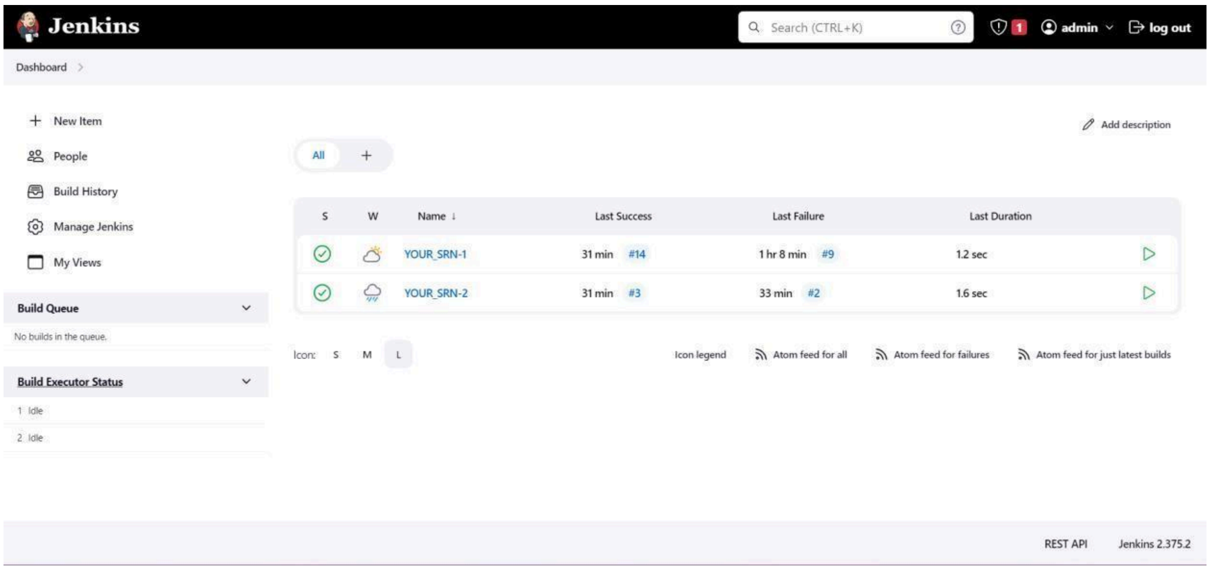
2. *Status page of first job*



Build History of Jenkins

S	Build	Time Since ↑	Status
✓	YOUR_SRN-2 #3	35 min	stable
✓	YOUR_SRN-1 #14	35 min	stable
✓	YOUR_SRN-1 #13	37 min	stable
✓	YOUR_SRN-1 #12	38 min	stable

3. *Build History of Jenkins*



No builds in the queue.

1 Idle
2 Idle

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		YOUR_SRN-1	31 min #14	1 hr 8 min #9	1.2 sec
✓		YOUR_SRN-2	31 min #3	33 min #2	1.6 sec

4. Jenkins Dashboard

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 `/var/jenkins_home/workspace/<the name of your first project>/main/hello_exec`
5. Click on Save.

Now, set your first job to call the second.

6. Go to your first job (i.e. YOUR_SRN-1) and open the *Configure* page in the pull-down menu.
7. Scroll to bottom and add a Post-Build Action. Select ***Build other projects***.
8. Enter the name of your second job.
9. Click on Save.
10. Run your first job.
11. Do this by clicking build now on the main page.
12. After that successfully builds, go, and check your second job. You should see it successfully run.
13. Select a Build Job from History and go to the console log to see your program output. If you program has run there, then you successfully set up a basic pipeline.
14. Take the required Screenshots.

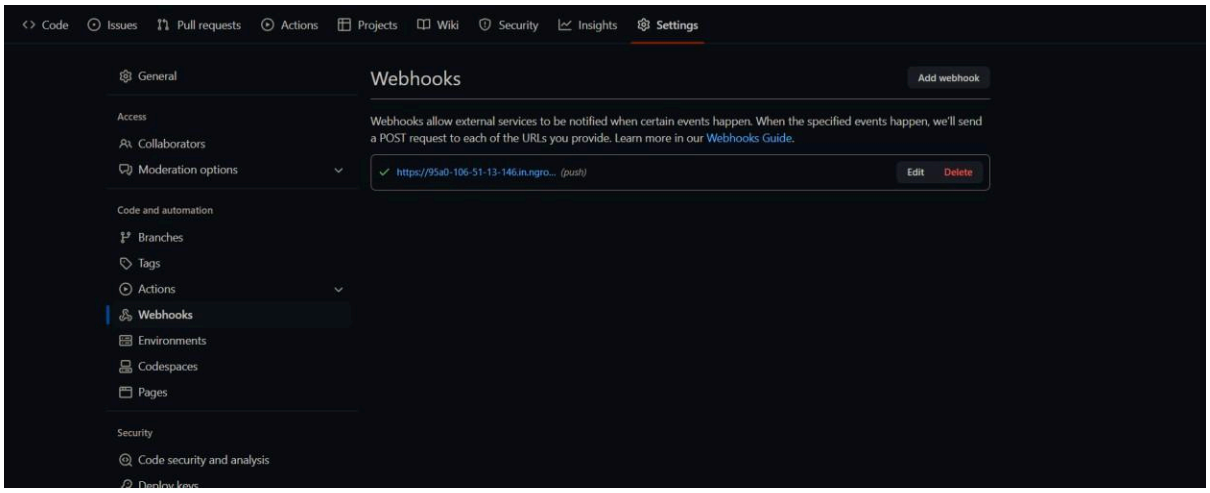
Task-4

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

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

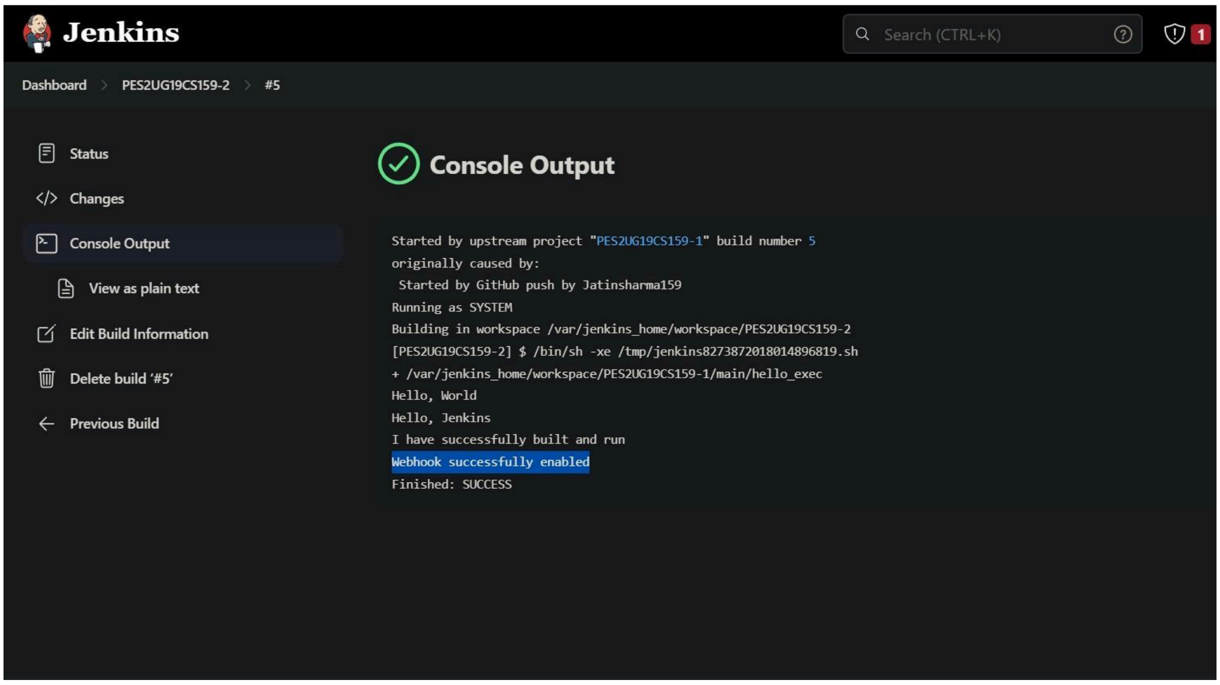


Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

https://95a0-106-51-13-146.in.ngro... (push)

1. Webhook added to your GitHub repository



Console Output

```
Started by upstream project "PES2UG19CS159-1" build number 5
originally caused by:
 Started by GitHub push by Jatinsharma159
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/PES2UG19CS159-2
[PES2UG19CS159-2] $ /bin/sh -xe /tmp/jenkins8273872018014896819.sh
+ /var/jenkins_home/workspace/PES2UG19CS159-1/main/hello_exec
Hello, World
Hello, Jenkins
I have successfully built and run
Webhook successfully enabled
Finished: SUCCESS
```

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

Steps:

1. Download ngrok from https://ngrok.com/download

   What is ngrok?

   ngrok is a cross-platform application that enables developers to expose a local development server to the Internet with minimal effort.

2. Open command prompt, navigate to the path where ngrok is downloaded and run the following commands:-

   ngrok –version

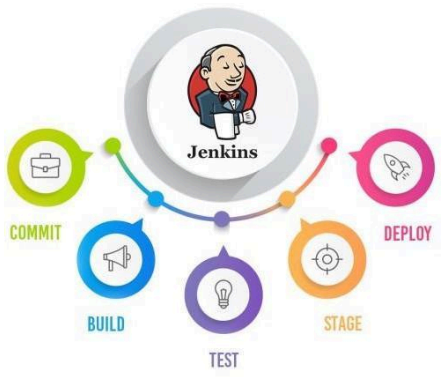
   ngrok http 8080 (Since, Jenkins runs on port 8080)

   Copy the provided https URL.

3. 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 → Add webhook.
4. 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?

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.

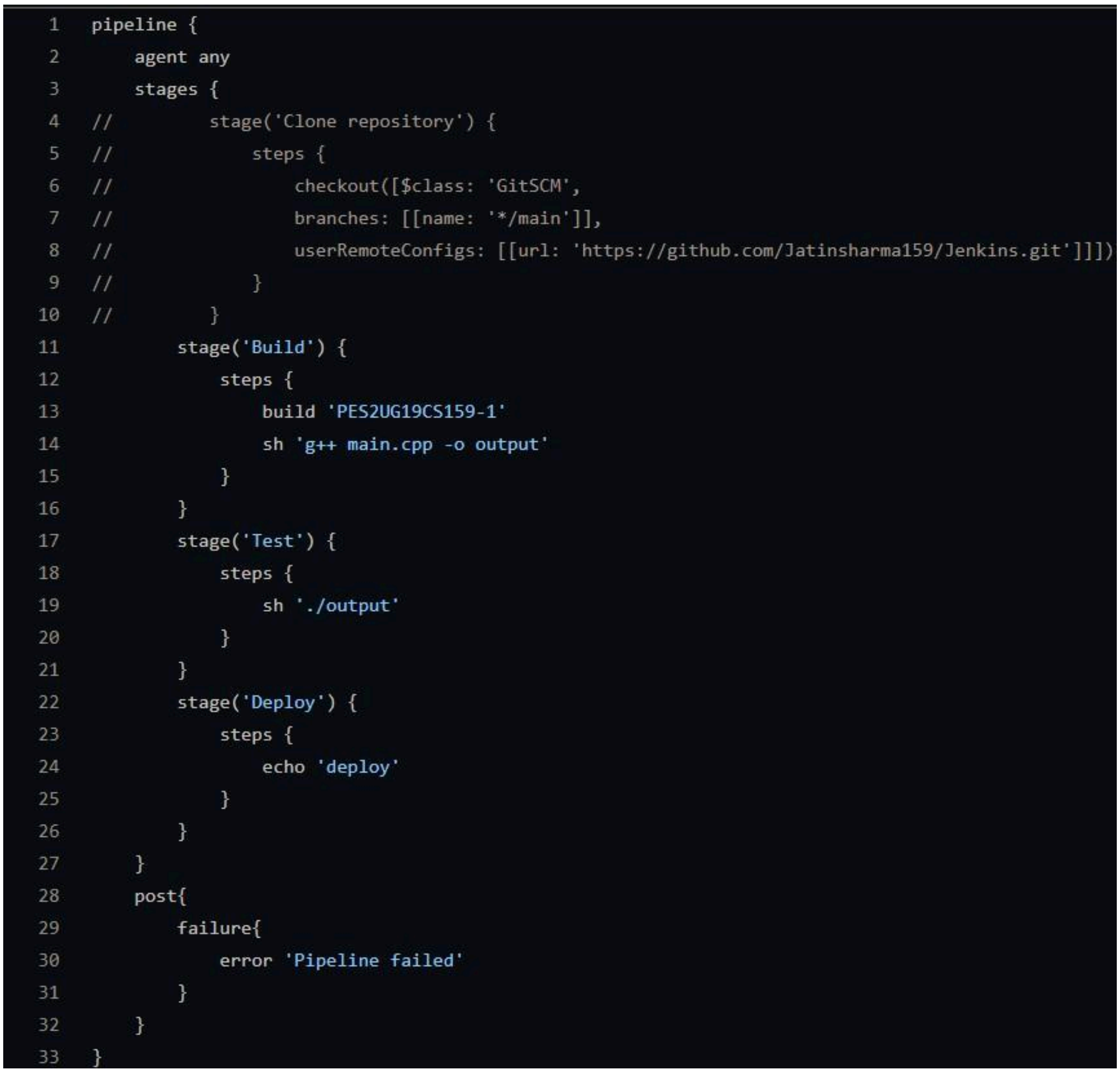
- Pipeline defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

- 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:



```
pipeline {
    agent any
    stages {
//         stage('Clone repository') {
//             steps {
//                 checkout([$class: 'GitSCM',
//                 branches: [[name: '*/main']],
//                 userRemoteConfigs: [[url: 'https://github.com/Jatinsharma159/Jenkins.git']]])
//             }
//         }
        stage('Build') {
            steps {
                build 'PES2UG19CS159-1'
                sh 'g++ main.cpp -o output'
            }
        }
        stage('Test') {
            steps {
                sh './output'
            }
        }
        stage('Deploy') {
            steps {
                echo 'deploy'
            }
        }
    }
    post{
        failure{
            error 'Pipeline failed'
        }
    }
}
```

1. Code/script written to create basic pipeline using GitHub repository

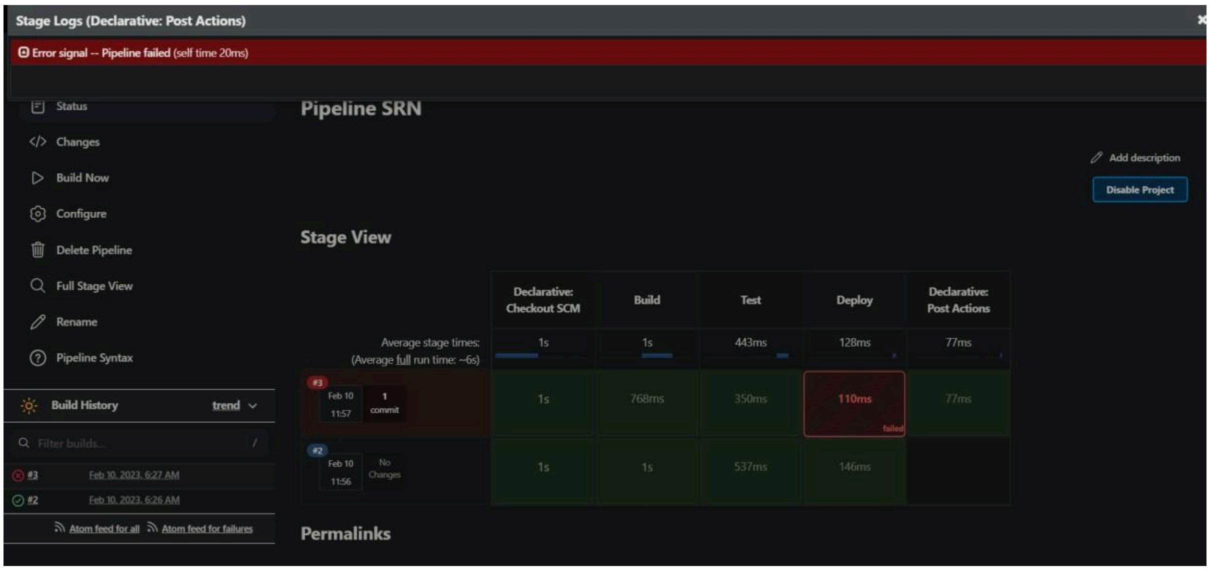
   Sample 1 (reference template)

```
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.



Stage Logs (Declarative: Post Actions)

Error signal -- Pipeline failed (self time 20ms)

Pipeline SRN

Add description

Disable Project

#3 Feb 10, 2023, 6:27 AM
#2 Feb 10, 2023, 6:26 AM

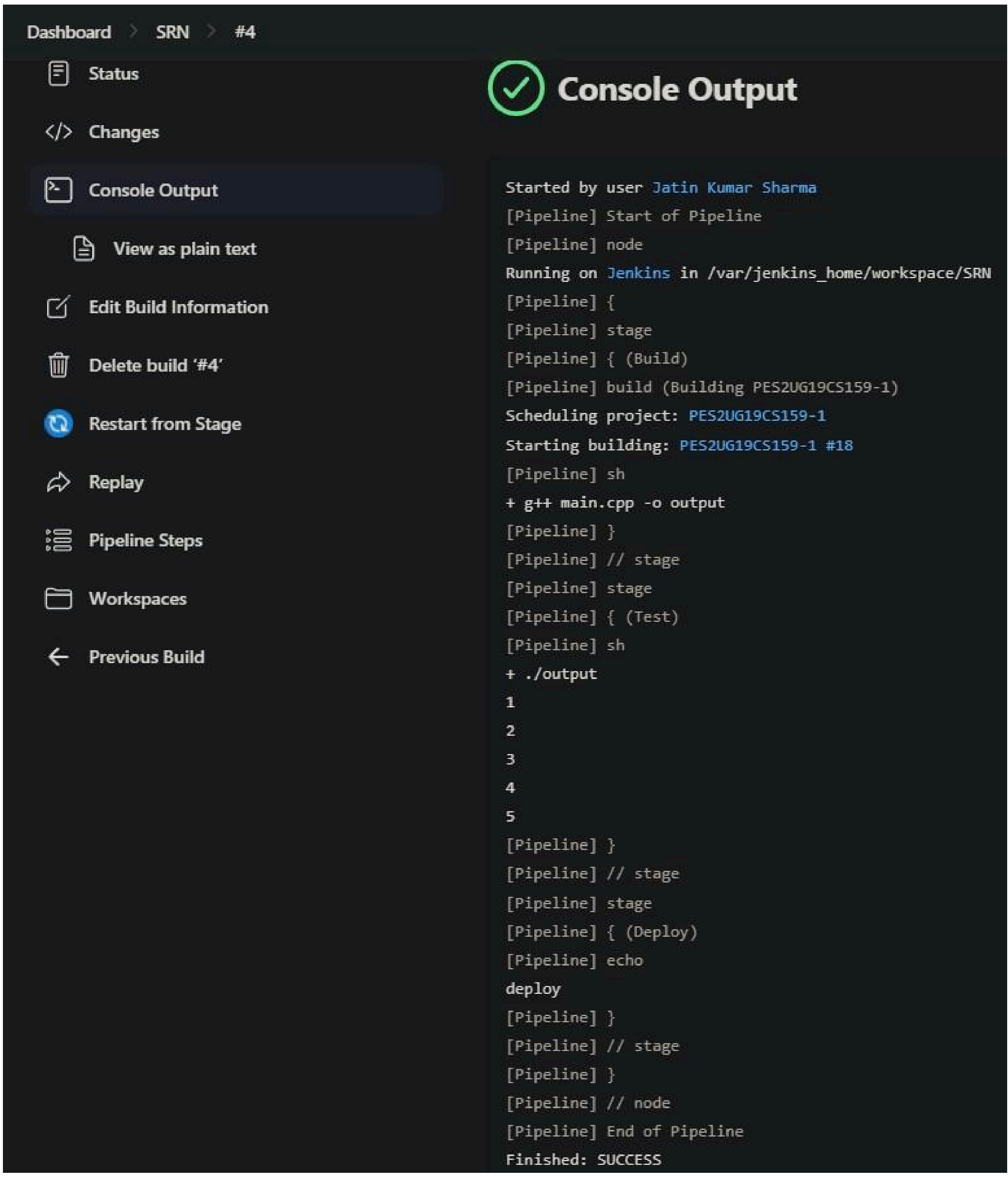
Atom feed for all Atom feed for failures

Stage View

	Declarative: Checkout SCM	Build	Test	Deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~6s)	1s	1s	443ms	128ms	77ms
#3 Feb 10 11:57 1 commit	1s	768ms	350ms	110ms failed	77ms
#2 Feb 10 11:56 No Changes	1s	1s	537ms	146ms	

Permalinks

3. Console Output of the Pipeline



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:

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 repositoryAdd file create new fileName 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 required screenshot of the Stage View.

Assignment - 4(b) – GitHub Actions
Building A CI Pipeline With GitHub Actions

What is a CI Pipeline?

A CI (Continuous Integration) pipeline automates the process of building, testing, and potentially deploying software. In cloud computing, it helps developers integrate code changes frequently, catch errors early, and deliver software updates faster and more reliably.

What are GitHub Actions?

GitHub Actions is a tool that automates tasks within your software development process directly on GitHub. Think of it like building blocks (actions) you connect to create customized workflows – for example, testing your code, deploying updates, or managing project tasks.

Pre - Requisites

- A GitHub account.
- A basic understanding of Git, GitHub, CMake, and C++.

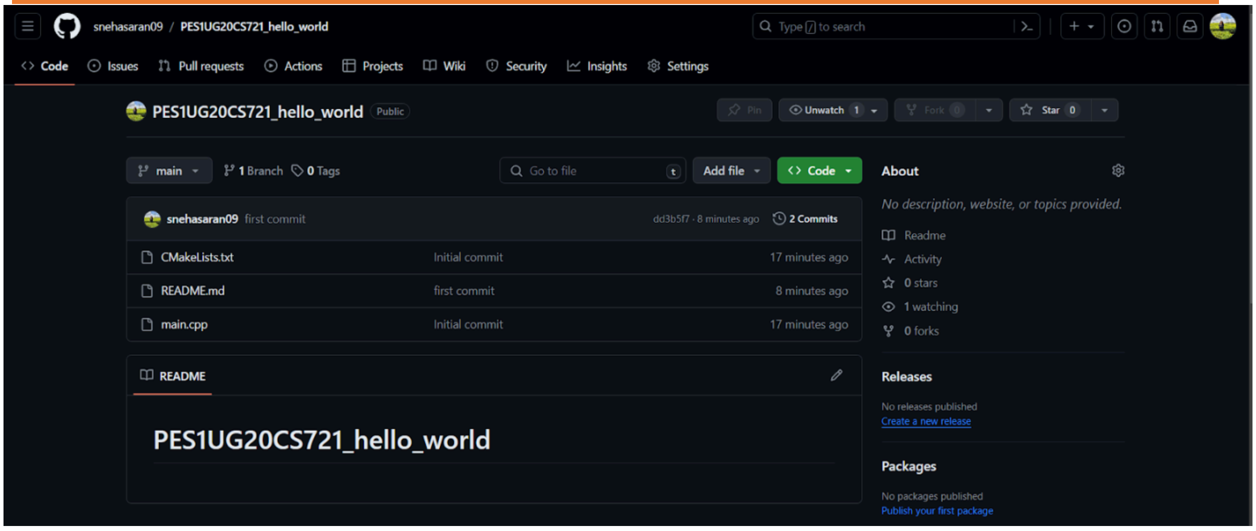
Project Setup:

1. Initialize Git Repository:

- Open a terminal or command prompt and navigate to the directory containing the two files shared - main.cpp and CMakeLists.txt. Change the SRN in the CMakeList to your SRN.
- Create a repository in GitHub as with the following Name: **SRN_hello_world**
- Push the two files shared in the repository created.

Note: Please use git commands to upload the files . Do not upload the files from the local directory.

(a) Take a screenshot of the repository in the manner given below after the two files have been uploaded onto the repository and name it (a)



snehasaran09 first commit dd3b5f7 · 8 minutes ago 2 Commits

CMakeLists.txt	Initial commit	17 minutes ago
README.md	first commit	8 minutes ago
main.cpp	Initial commit	17 minutes ago

PES1UG20CS721_hello_world

2. Create the GitHub Actions Workflow:

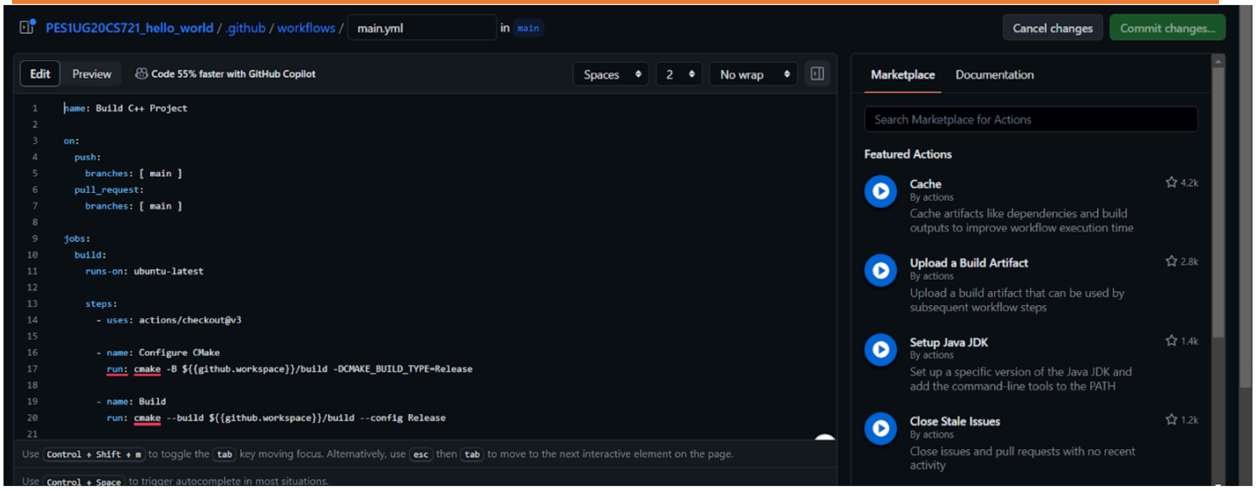
2.1 Create the Workflow File:

- In your GitHub repository, go to the Actions tab.
- Click on New workflow.
- Choose **set up a workflow yourself.**

2.2. Paste the Workflow Code:

```
name: Build C++ Project (Optional Testing Removed)
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Configure CMake
        run: cmake -B ${{github.workspace}}/build -DCMAKE_BUILD_TYPE=Release
      - name: Build
        run: cmake --build ${{github.workspace}}/build --config Release
```

(b) Take a screenshot of the code after it has been pasted onto the workflow and name it (b)



PES1UG20CS721_hello_world / .github / workflows / main.yml in main

```yaml
name: Build C++ Project

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Configure CMake
        run: cmake -B ${{github.workspace}}/build -DCMAKE_BUILD_TYPE=Release

      - name: Build
        run: cmake --build ${{github.workspace}}/build --config Release

```

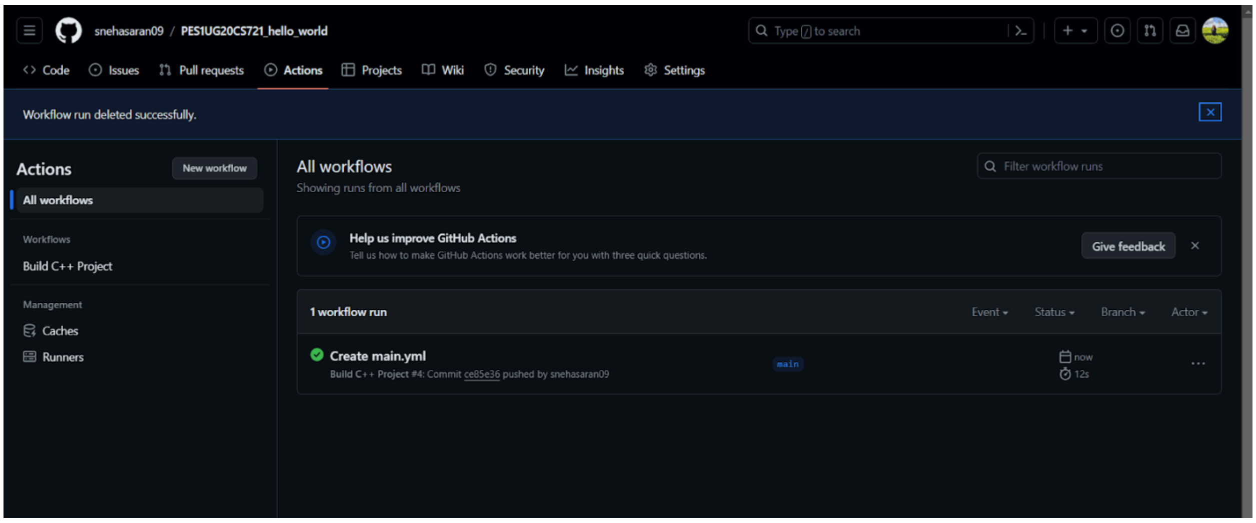
Use Control + Shift + m to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.

Use Control + Space to trigger autocomplete in most situations.

Cancel changes
Commit changes...
Edit
Preview
Code 55% faster with GitHub Copilot
Spaces
2
No wrap
Marketplace
Documentation
Search Marketplace for Actions
Featured Actions
Cache
By actions
Cache artifacts like dependencies and build outputs to improve workflow execution time
4.2k
Upload a Build Artifact
By actions
Upload a build artifact that can be used by subsequent workflow steps
2.8k
Setup Java JDK
By actions
Set up a specific version of the Java JDK and add the command-line tools to the PATH
1.4k
Close Stale Issues
By actions
Close issues and pull requests with no recent activity
1.2k

- Click on Commit changes

(c). Take a screenshot as given below indicating that the build is successful and name it (c)



Workflow run deleted successfully.

Create main.yml
Build C++ Project #4: Commit ce85e36 pushed by snehasaran09

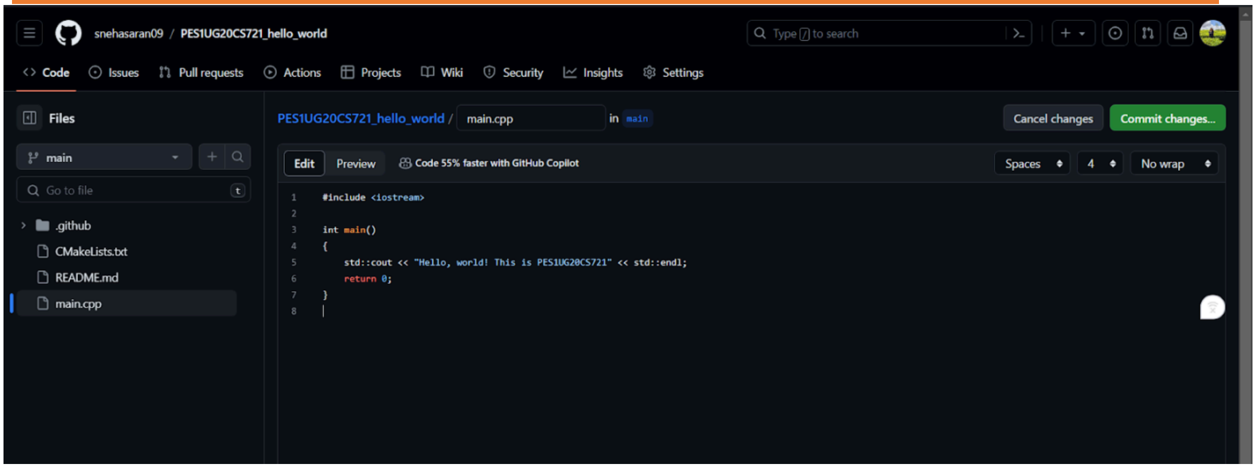
3. Monitor the Workflow:

The workflow will automatically trigger when you push changes to the main branch or create a pull request against it.

3.1 Making changes to the main branch:

- We will now make a small change in the main branch to check if its reflecting in the build
- Go to the main.cpp file in your repository
- Next to Hello World , add an additional sentence - “**This is <SRN>**”

(d) Take a screenshot as given below after making the change in the main.cpp file and name it (d)

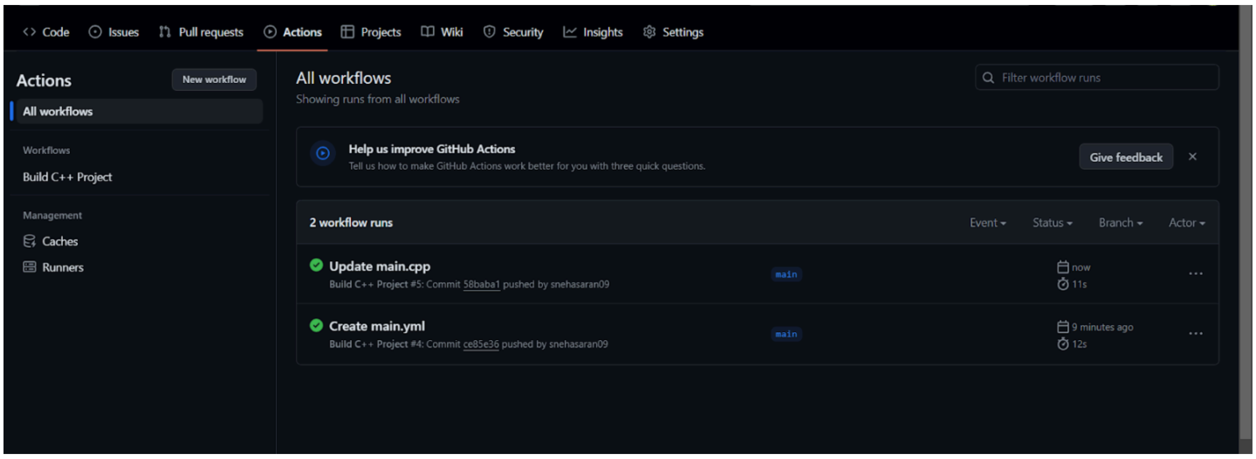


```
#include <iostream>

int main()
{
    std::cout << "Hello, world! This is PES1UG20CS721" << std::endl;
    return 0;
}
```

- Click on commit changes and in the added description enter - "**updated the .cpp file to check build** "
- Now go to Actions and check the status of the build

(e) Take a screenshot of the status of the build after it has passed and name it (e)



All workflows
Showing runs from all workflows

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

2 workflow runs

Update main.cpp
Build C++ Project #5: Commit 58baba1 pushed by snehasaran09 — main — now — 11s

Create main.yml
Build C++ Project #4: Commit ce85e36 pushed by snehasaran09 — main — 9 minutes ago — 12s