بسم الله الرحمن الرحيم

Assignment on,

Software Quality in Practice

Department of Software Engineering (SE)

Assignment no. 1

Assignment To

Md. Mahmodul Hasan

Lecturer at SE ,

Bangladesh Digital University

Assignment from

Irshad Hossain (2303030)

Department of SE,

Bangladesh Digital University

Key Benefits of Continuous Integration (CI) and Continuous Deployment (CD) in Software Development

1. Faster Feedback and Reduced Risks

CI helps developers quickly find out if their code changes work properly by automatically testing them. This makes it easier to fix bugs early, saving time and money. CD ensures that only the code that has passed tests gets deployed, which reduces the chances of introducing issues into production.

2. Better Teamwork

CI encourages developers to share their code frequently. By merging their code into one common place, everyone stays updated on what’s being worked on, which improves communication and teamwork.

3. Improved Software Quality

Automated tests in CI/CD check that the code meets certain standards, catching bugs early before they become bigger problems. This leads to higher-quality software that's more reliable when it’s released.

4. Faster Release Times

With CD, the process of delivering new features, updates, or bug fixes is quicker. Code can be automatically deployed once it’s ready, allowing businesses to get new versions of software to users faster, keeping them competitive in fast-moving markets.

5. More Efficiency and Scalability

CI/CD pipelines help automate repetitive tasks, making the development process more efficient. As teams grow, CI/CD makes it easier to manage large projects without increasing manual effort.

The CI/CD Pipeline for an Application

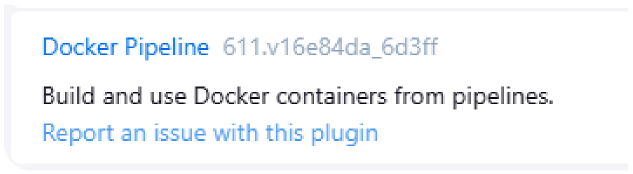
A CI/CD pipeline is a series of automated steps that handle code changes from development to deployment. Here’s how it typically works:

1. **Code Integration (CI):** Developers push their code to a shared repository. Automated tools check the code, build the application, and run tests to make sure it works with the existing code.
2. **Automated Testing:** The pipeline runs various tests (like unit tests or integration tests) to ensure that the new code doesn't break anything and functions as expected.
3. **Continuous Delivery (CD):** If the code passes all tests, it is packaged and prepared for deployment. At this point, manual approval may be needed before deploying to production.
4. **Continuous Deployment (CD):** If the team is fully automated, the code is automatically deployed to production without manual approval once it passes the tests.

CI/CD Example with Jenkins

Prerequisite :

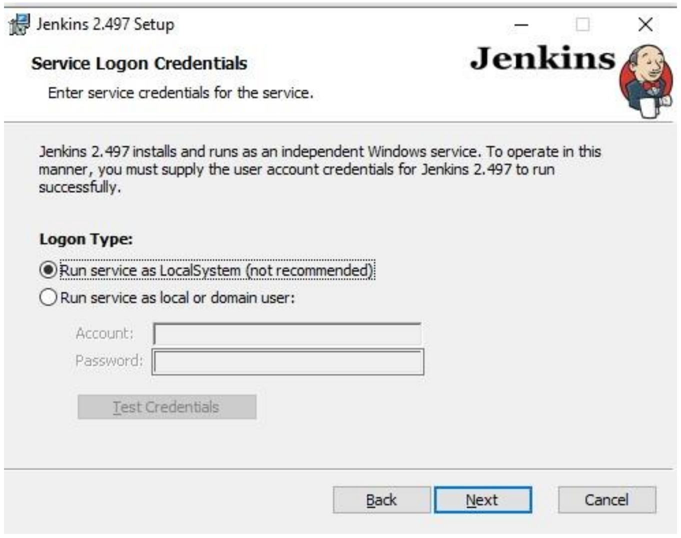
1. GCC install and System Environment Path
2. Jenkins & Java install and verify
3. Jenkins plugin ‘Docker Pipeline’ install
4. GitHub profile


Docker Pipeline 611.v16e84da_6d3ff

Build and use Docker containers from pipelines.

Report an issue with this plugin

Step 1: Download and Install Jenkins and Java



Jenkins 2.497 Setup

Service Logon Credentials

Enter service credentials for the service.

Jenkins 2.497 installs and runs as an independent Windows service. To operate in this manner, you must supply the user account credentials for Jenkins 2.497 to run successfully.

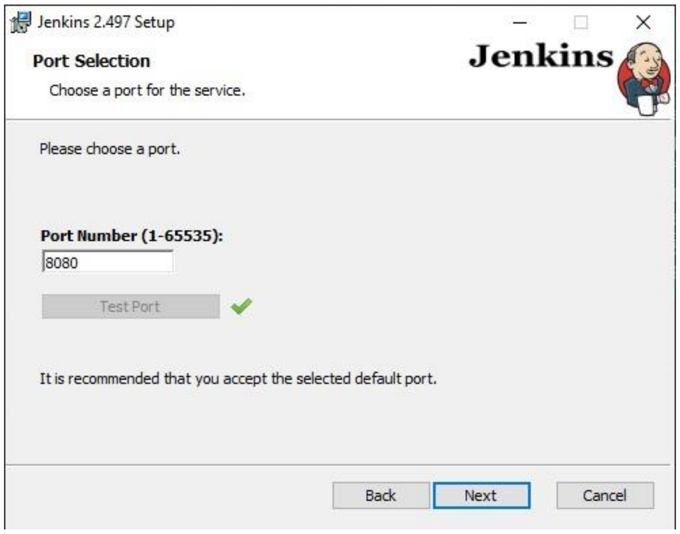
Logon Type:

[✓ Run service as LocalSystem (not recommended)]

Run service as local or domain user:

Account:

Password:



Jenkins 2.497 Setup

Port Selection

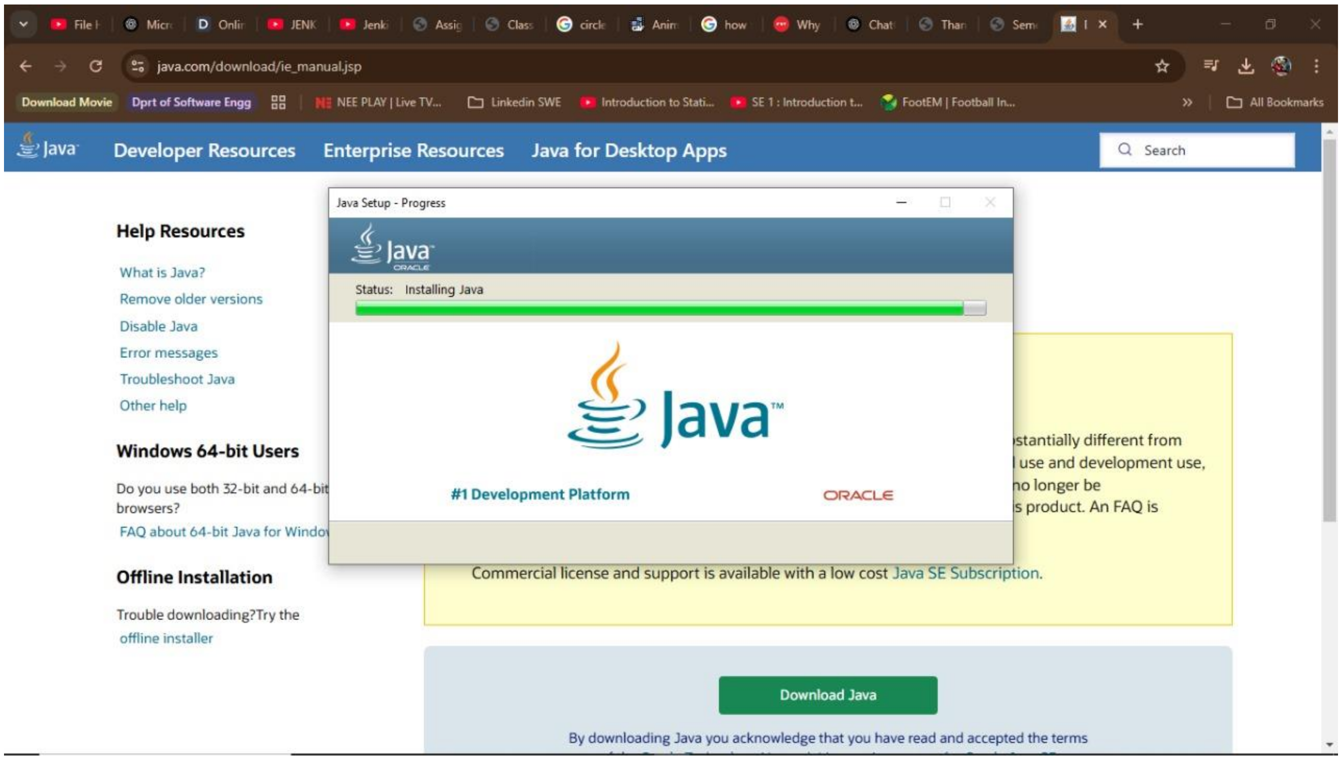
Choose a port for the service.

Please choose a port.

Port Number (1-65535): 8080

It is recommended that you accept the selected default port.

Correction: Port 8090 (due to reinstallation)



Java Setup - Progress

Status: Installing Java

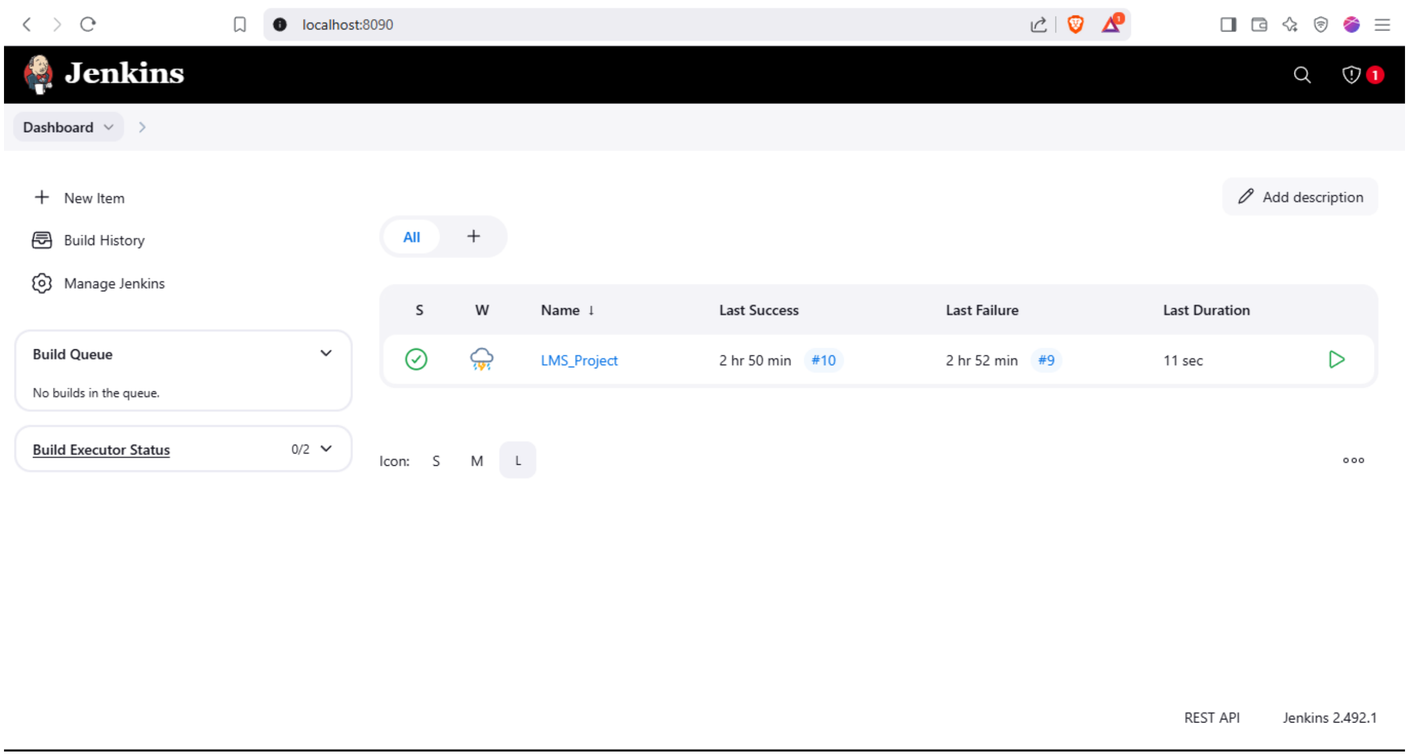
#1 Development Platform

```
C:\Users\irsha>java -version
java version "1.8.0_441"
Java(TM) SE Runtime Environment (build 1.8.0_441-b07)
Java HotSpot(TM) Client VM (build 25.441-b07, mixed mode, sharing)

C:\Users\irsha>
```

Verify Java with CMD

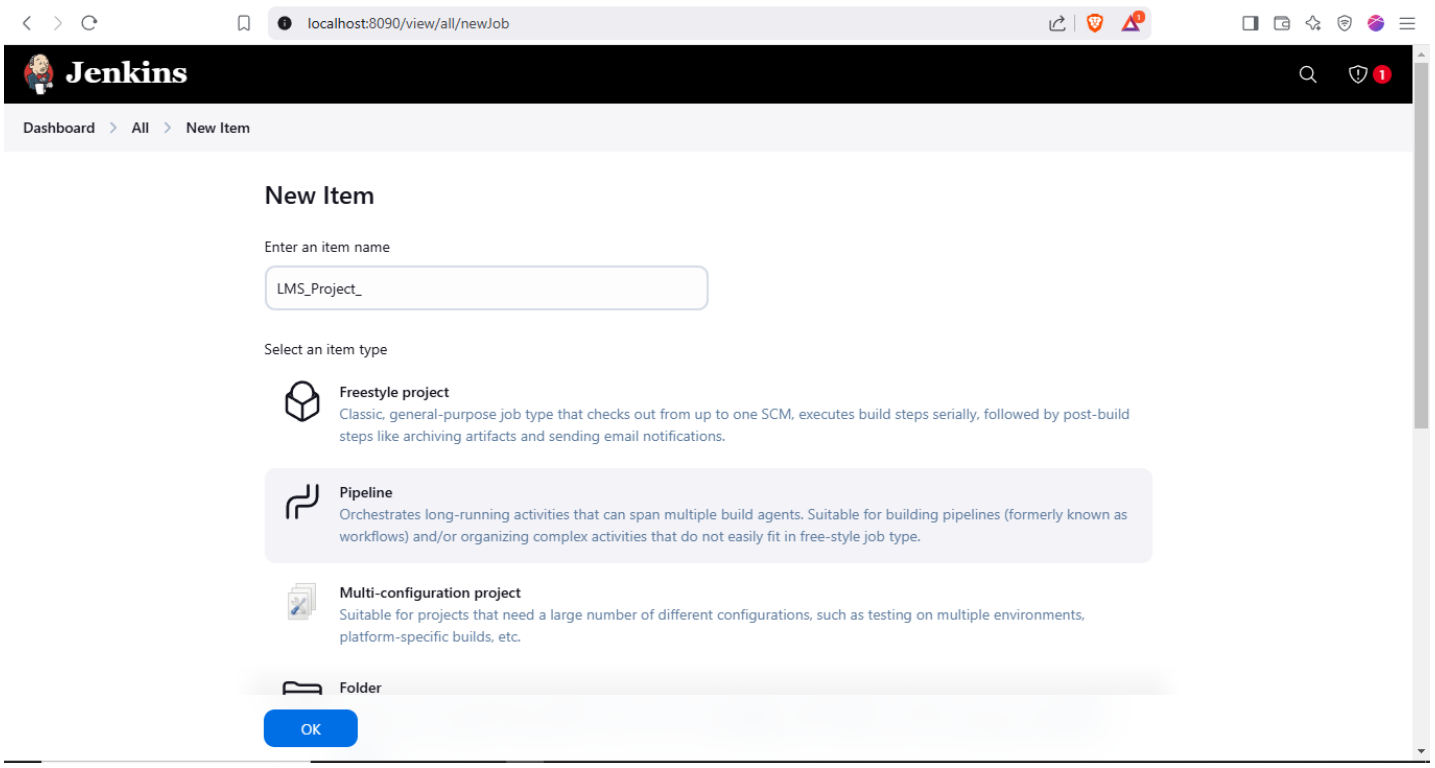
Step 2: Create our first Pipeline for LMS Project



No builds in the queue.

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		LMS_Project	2 hr 50 min #10	2 hr 52 min #9	11 sec

This is the Jenkins Dashboard, running on port 8090. Generally, it will automatically run on port 8080



New Item

Enter an item name

LMS_Project_

Select an item type

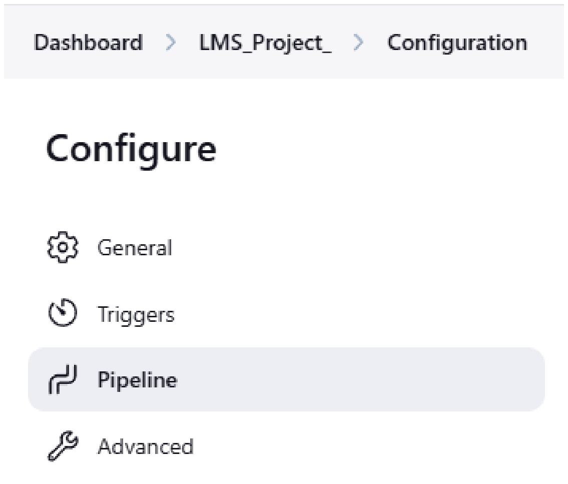
Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder

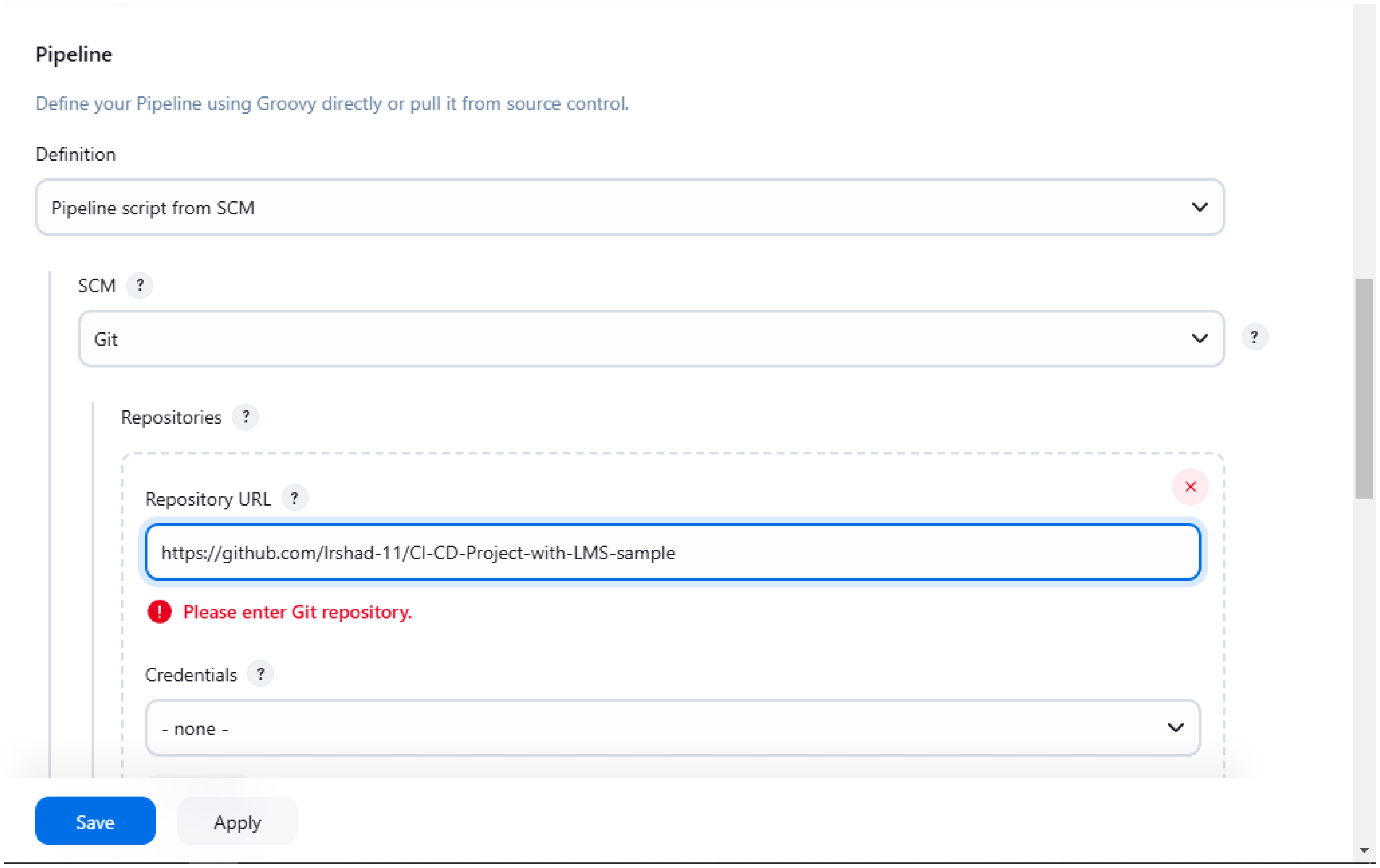
To create our first pipeline, we should follow these steps: Click **New Item**, enter a name for the pipeline, select the **Pipeline** option, and click **Save**.



Dashboard > LMS_Project_ > Configuration

Configure

Leave these fields blank for now; we'll only fill in the basic and essential information. Navigate to the **Pipeline** option from the sidebar



Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition: Pipeline script from SCM

SCM: Git

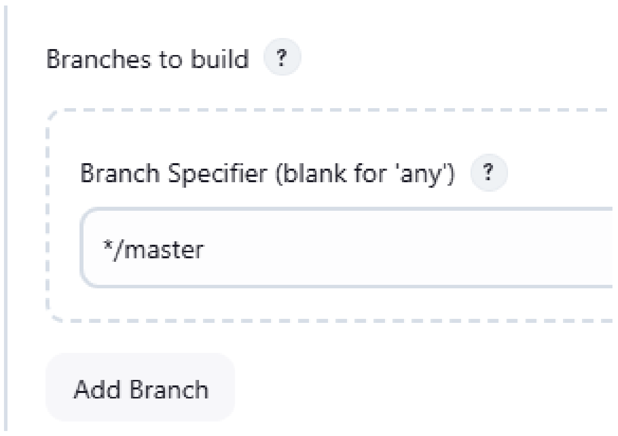
Repositories

Repository URL: https://github.com/Irshad-11/CI-CD-Project-with-LMS-sample

Please enter Git repository.

Credentials: - none -

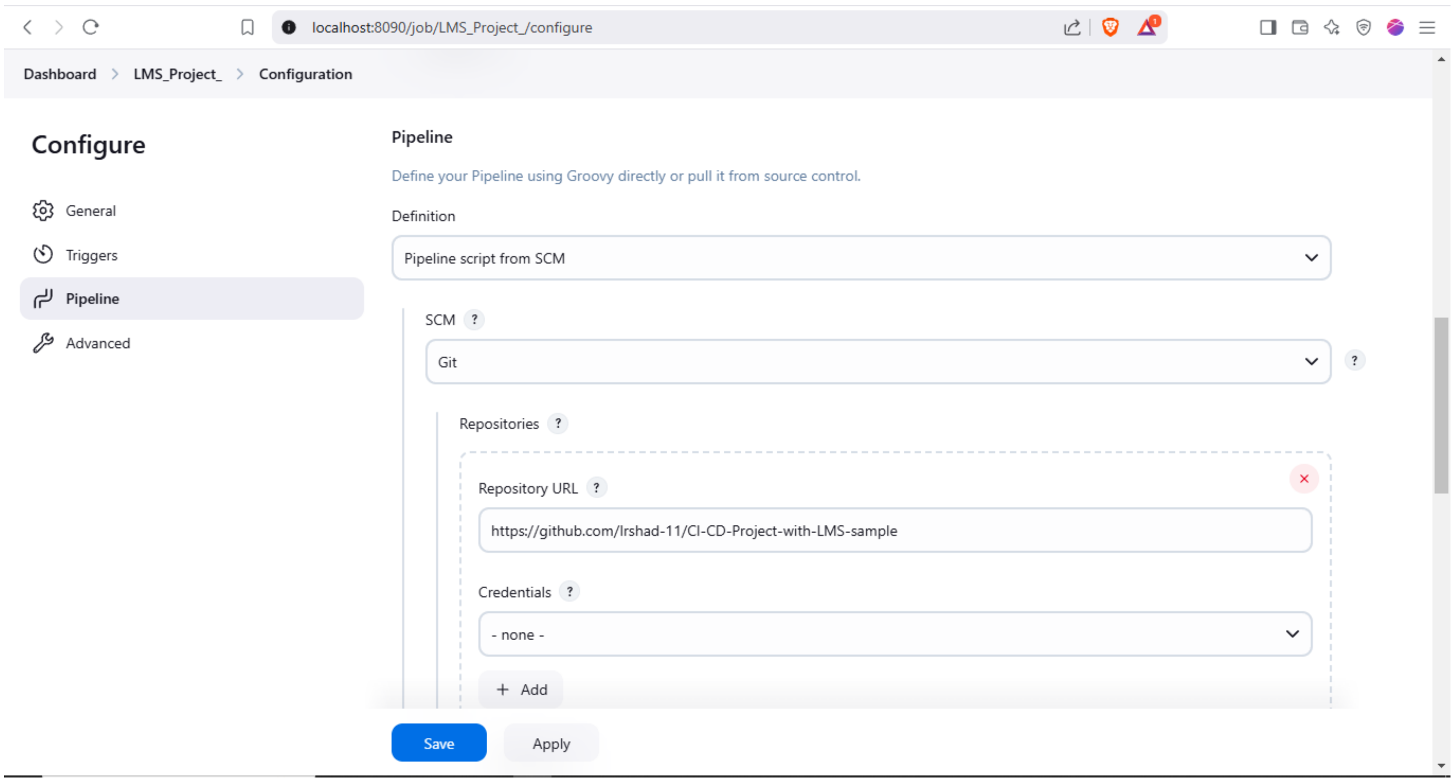
Here, from the *Definition* dropdown menu, select **Pipeline script from SCM** . Then, from the *SCM* dropdown menu, choose **Git**. Next, paste the GitHub repository URL that contains the Jenkinsfile and project



Branches to build

Branch Specifier (blank for 'any'): */master

Rename the branch from **/master* to ***/main** to avoid errors during the build phase



Configure

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition: Pipeline script from SCM

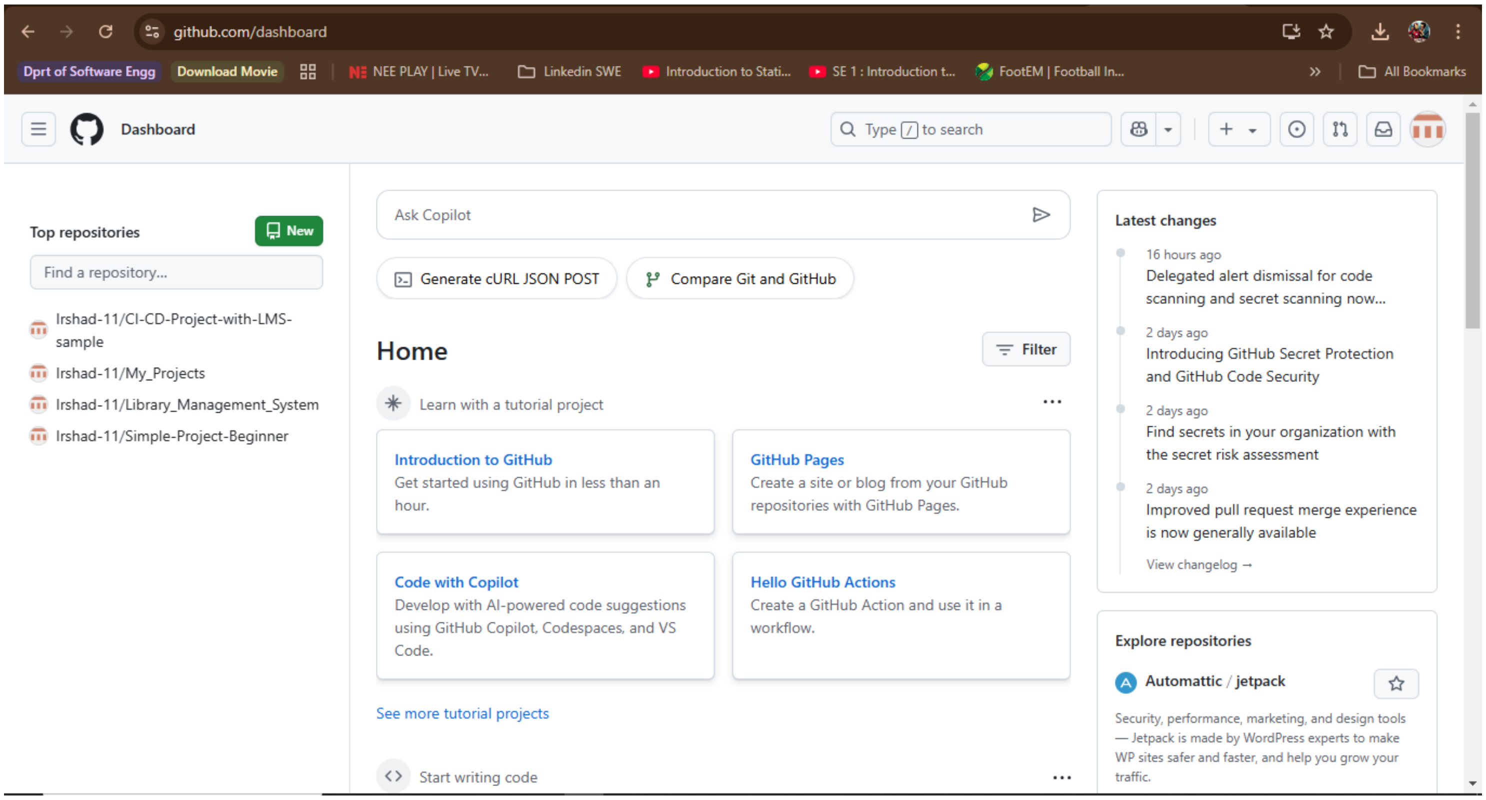
SCM: Git

Repositories

Repository URL: https://github.com/Irshad-11/CI-CD-Project-with-LMS-sample

Credentials: - none -

Save the Changes



Top repositories

Irshad-11/CI-CD-Project-with-LMS-sample

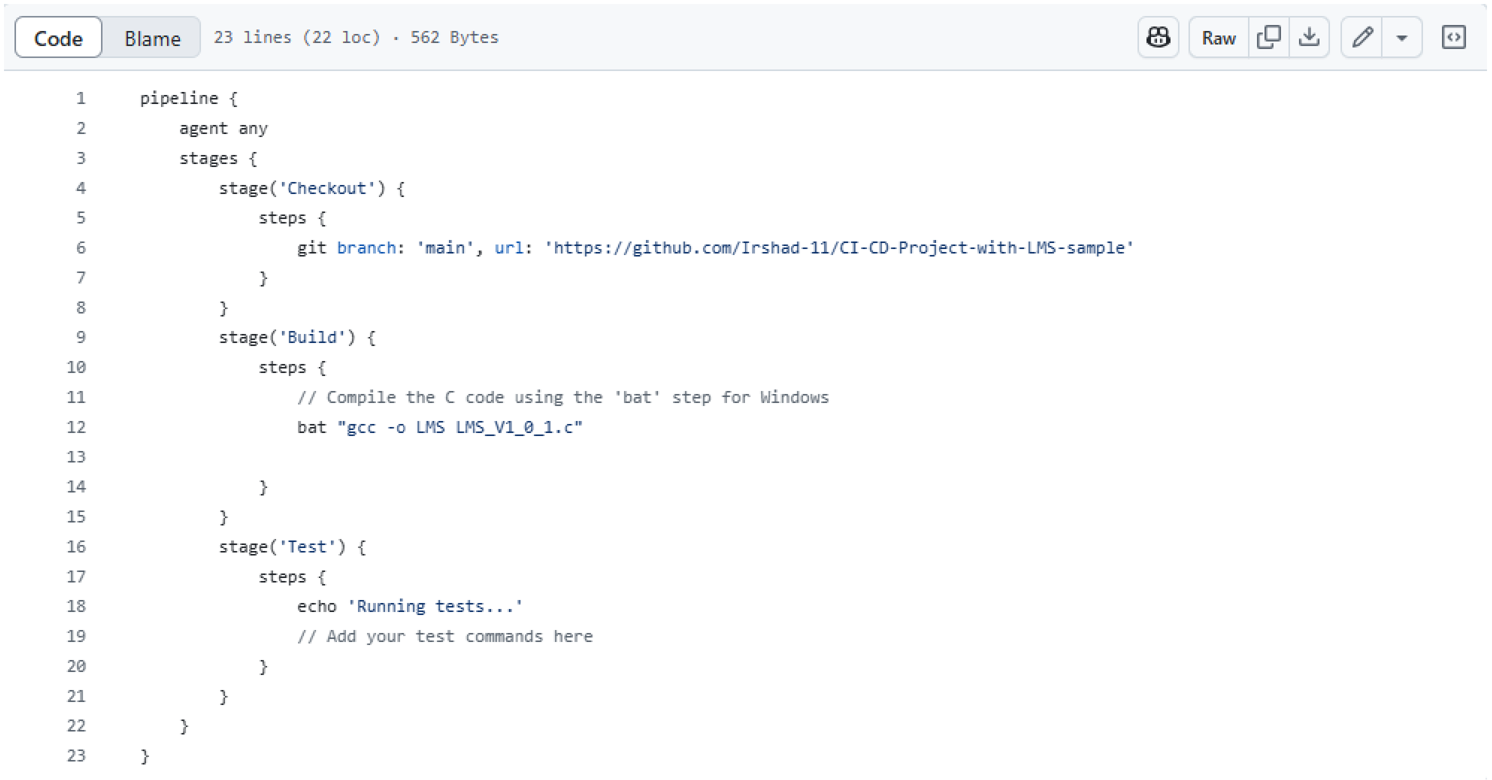
Irshad-11/My_Projects

Irshad-11/Library_Management_System

Irshad-11/Simple-Project-Beginner

Move to GitHub Dashboard and create a new repository by clicking on the New . Name our repository by entering a title Owner * Irshad-11 / Repository name * Library_Management_Syst in the designated field. Then, scroll down and click on the Create repository .Our repository is now created. Next, we need to create a **Jenkinsfile** where we will edit our pipeline code. To do this, scroll down and click on the creating a new file located here. Name this file as **Jenkins** Jenkins . This is important because Jenkins will look for a file named Jenkinsfile in our GitHub repository to define the pipeline. Now, we can proceed to define the pipeline by editing this file.

Pipeline Code



Code Blame 23 lines (22 loc) · 562 Bytes Raw

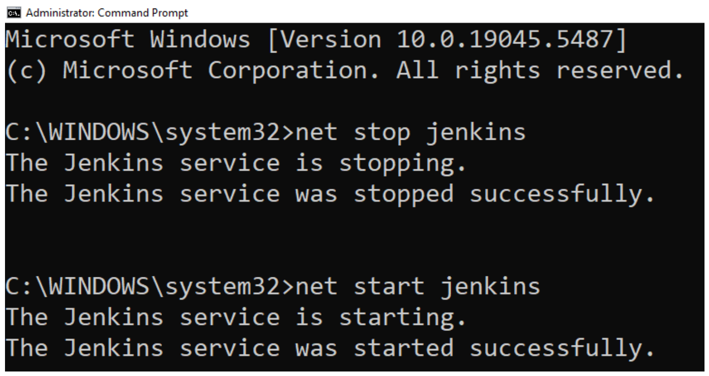
```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/Irshad-11/CI-CD-Project-with-LMS-sample'
            }
        }
        stage('Build') {
            steps {
                // Compile the C code using the 'bat' step for Windows
                bat "gcc -o LMS LMS_V1_0_1.c"

            }
        }
        stage('Test') {
            steps {
                echo 'Running tests...'
                // Add your test commands here
            }
        }
    }
}
```

Explanation of this code:

1. `pipeline {agent any}` : This defines that the pipeline will run on any available Jenkins agent. In this case the agent is Windows
2. `stages {}` : This block contains all the stages of the Jenkins pipeline. Each stage will be executed in sequence.
3. `Stage Checkout` :
   - The *git* command fetches the code from the specified GitHub repository.
   - The branch: *main* specifies that it will use the main branch.
   - The `url: https://github.com/Irshad-11/CI-CD-Project-with-LMS-sample` provides the repository URL from which the pipeline will pull the latest code.
4. `Stage Build` :
   - `bat gcc -o LMS LMS.c`: This command compiles the C code (located in LMS.c) using the gcc compiler, which will output an executable named LMS.exe.
5. `Stage Test` :
   - `echo Running tests...`: This prints a message to the Jenkins log indicating that the test stage has started.
   - `bat LMS.exe`: This command runs the compiled executable (LMS.exe) to verify that the program works as expected.

All is set. Now, commit the changes and proceed back to Jenkins



Administrator: Command Prompt

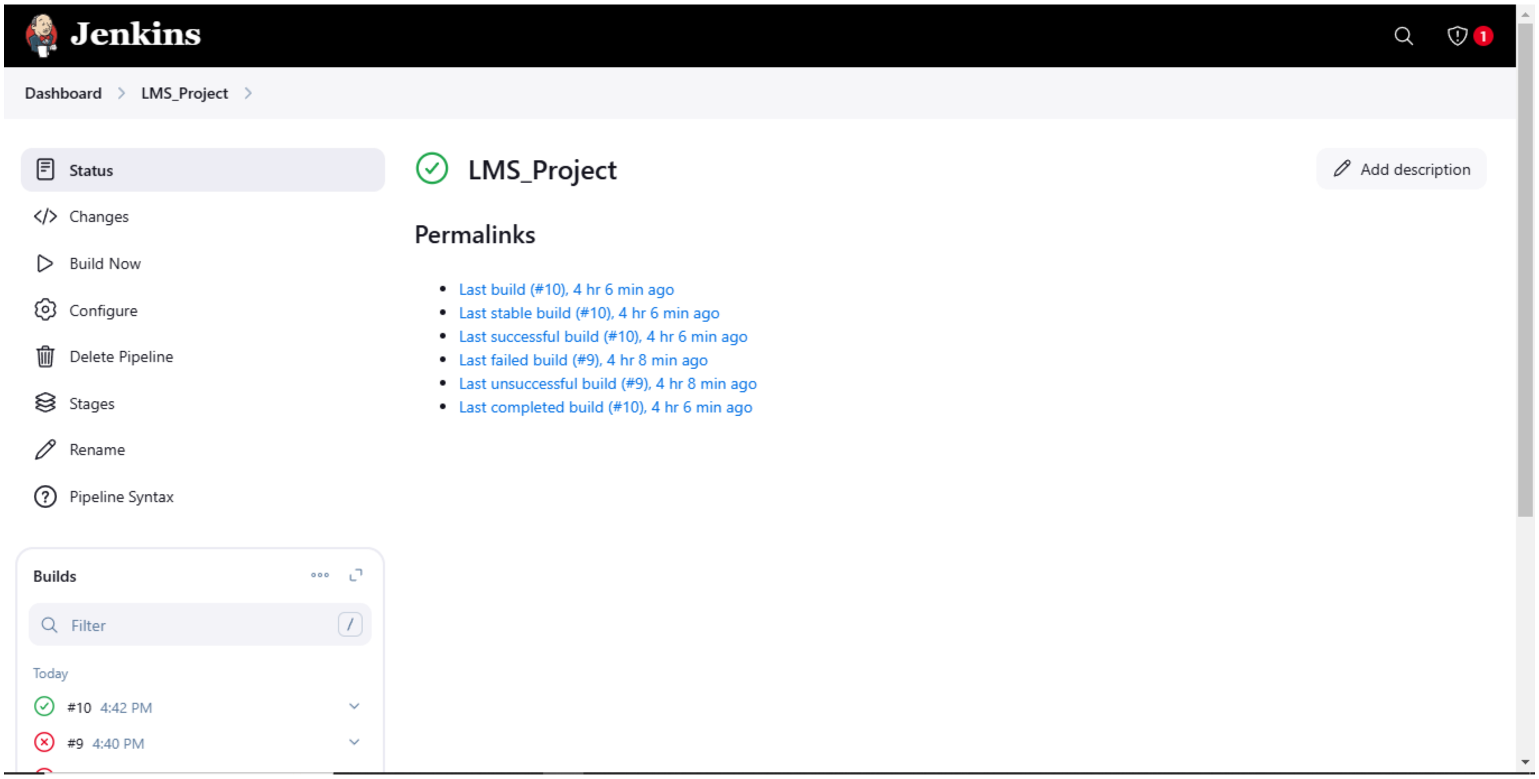
```
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>net stop jenkins
The Jenkins service is stopping.
The Jenkins service was stopped successfully.


C:\WINDOWS\system32>net start jenkins
The Jenkins service is starting.
The Jenkins service was started successfully.
```

Ensure that Jenkins Restart Successfully

Step 3: ▷ Build Now


Jenkins

Dashboard > LMS_Project >

LMS_Project

Permalinks

- Last build (#10), 4 hr 6 min ago
- Last stable build (#10), 4 hr 6 min ago
- Last successful build (#10), 4 hr 6 min ago
- Last failed build (#9), 4 hr 8 min ago
- Last unsuccessful build (#9), 4 hr 8 min ago
- Last completed build (#10), 4 hr 6 min ago

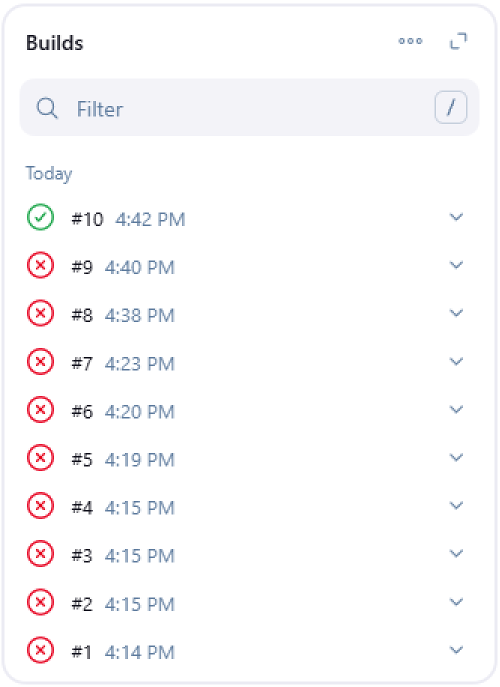
Builds

Today

#10 4:42 PM

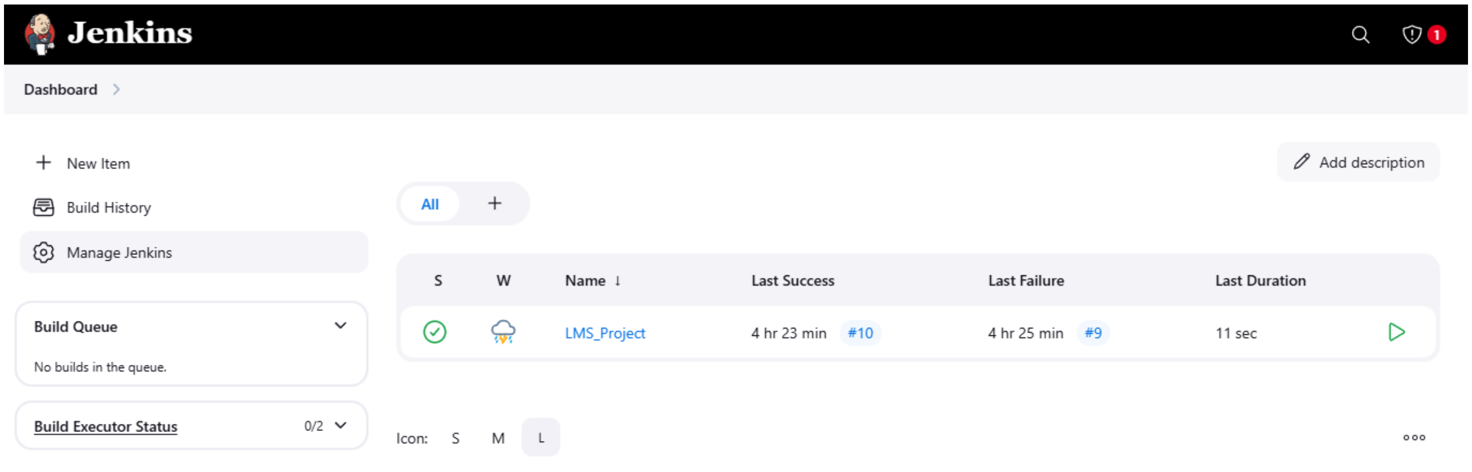
#9 4:40 PM

Since I have tested this previously, so interface might appear slightly different from mine. However, we need to click on the ▷ Build Now button, and we will see the progress and results in the Builds section. It's highly possible that the build will initially fail, marked with a cross icon ⊗ . To identify the cause of the problem, click on the dropdown icon ⌄ and select Console Output Note that it took me ten builds to achieve success, so be prepared to troubleshoot. Finally the Dashboard will like this :


Builds

Today

✓ #10	4:42 PM
⊗ #9	4:40 PM
⊗ #8	4:38 PM
⊗ #7	4:23 PM
⊗ #6	4:20 PM
⊗ #5	4:19 PM
⊗ #4	4:15 PM
⊗ #3	4:15 PM
⊗ #2	4:15 PM
⊗ #1	4:14 PM


Jenkins

Dashboard >

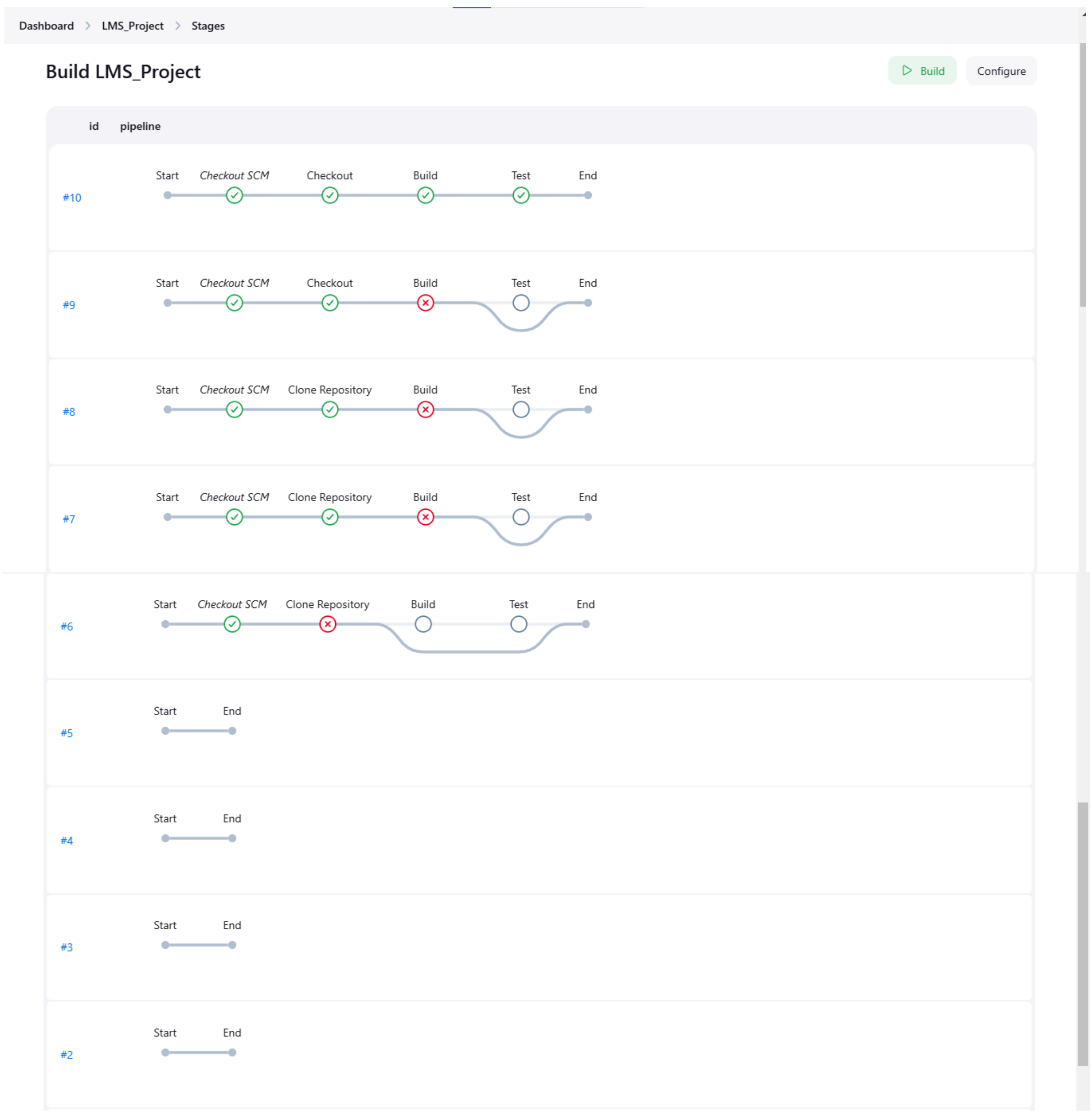
Build Queue

No builds in the queue.

Build Executor Status 0/2

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		LMS_Project	4 hr 23 min #10	4 hr 25 min #9	11 sec

Build Summary :



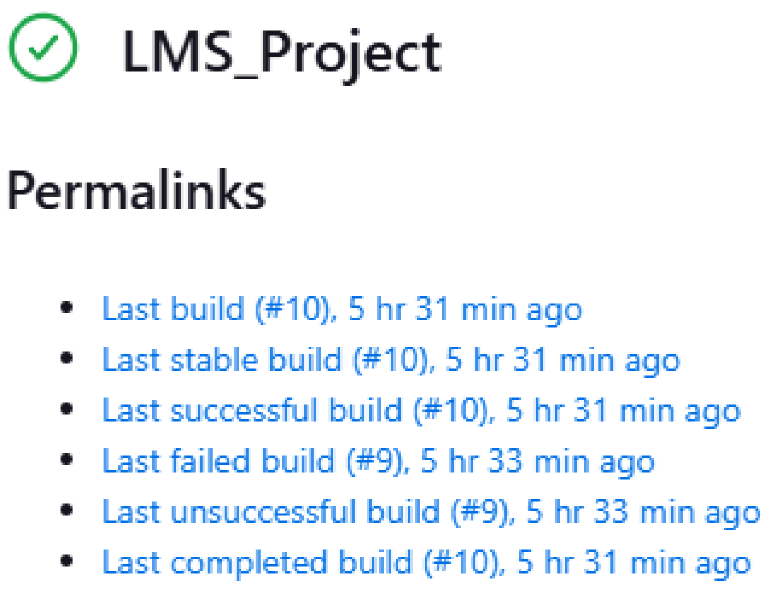
Dashboard > LMS_Project > Stages

Build LMS_Project

Build | Configure

id	pipeline
#10	Start — Checkout SCM ✓ — Checkout ✓ — Build ✓ — Test ✓ — End
#9	Start — Checkout SCM ✓ — Checkout ✓ — Build ✗ — Test ○ — End
#8	Start — Checkout SCM ✓ — Clone Repository ✓ — Build ✗ — Test ○ — End
#7	Start — Checkout SCM ✓ — Clone Repository ✓ — Build ✗ — Test ○ — End
#6	Start — Checkout SCM ✓ — Clone Repository ✗ — Build ○ — Test ○ — End
#5	Start — End
#4	Start — End
#3	Start — End
#2	Start — End

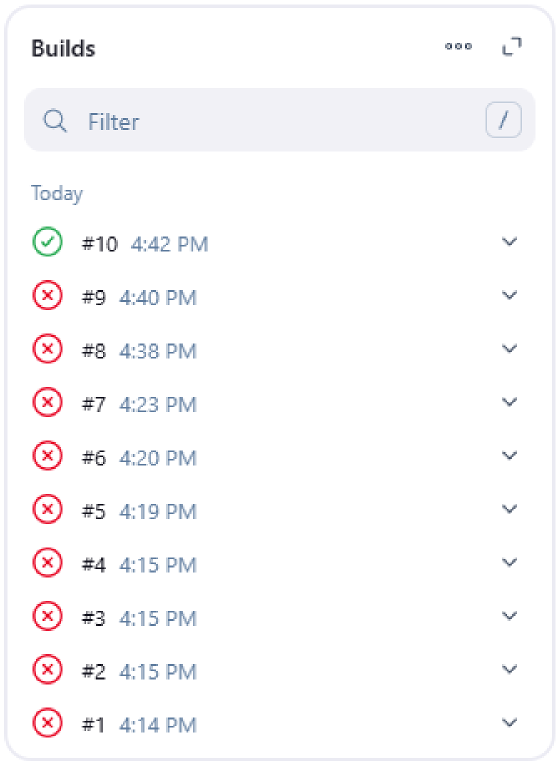
Summary :



LMS_Project

Permalinks

- Last build (#10), 5 hr 31 min ago
- Last stable build (#10), 5 hr 31 min ago
- Last successful build (#10), 5 hr 31 min ago
- Last failed build (#9), 5 hr 33 min ago
- Last unsuccessful build (#9), 5 hr 33 min ago
- Last completed build (#10), 5 hr 31 min ago

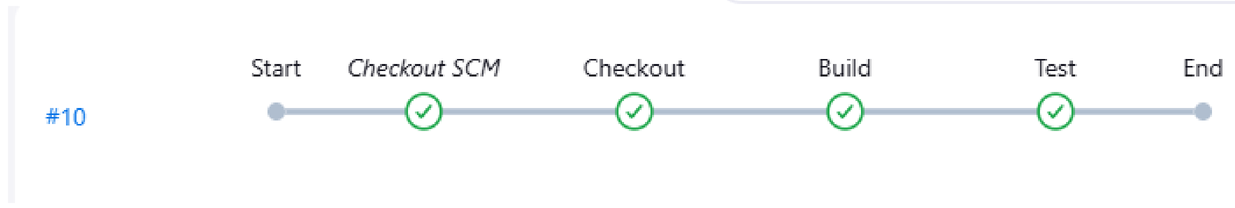


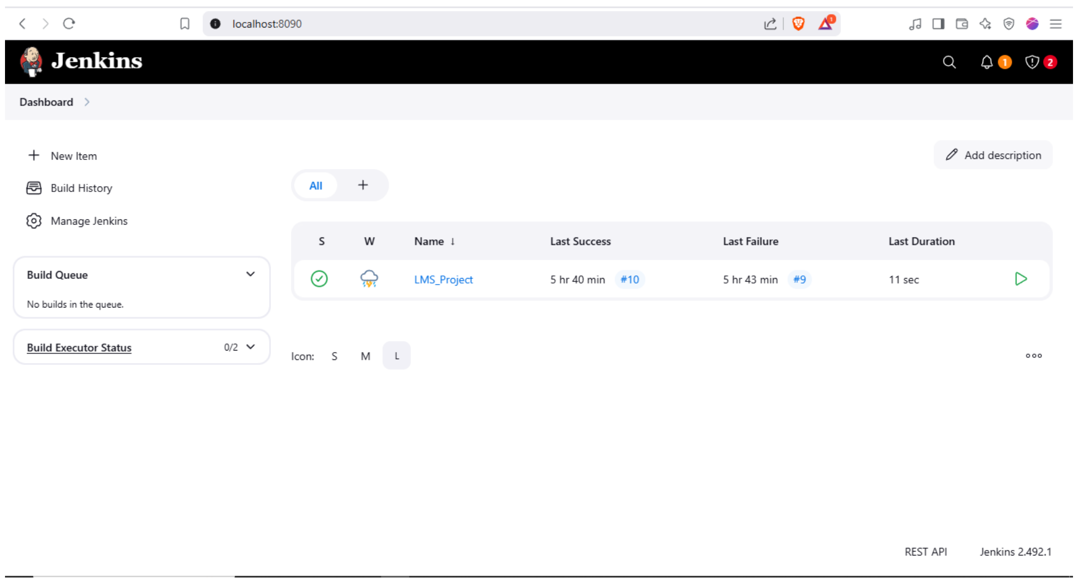
Builds

Filter

Today

✓	#10 4:42 PM
✗	#9 4:40 PM
✗	#8 4:38 PM
✗	#7 4:23 PM
✗	#6 4:20 PM
✗	#5 4:19 PM
✗	#4 4:15 PM
✗	#3 4:15 PM
✗	#2 4:15 PM
✗	#1 4:14 PM





S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		LMS_Project	5 hr 40 min #10	5 hr 43 min #9	11 sec

The CI/CD pipeline for the LMS project has been successfully set up

Challenges in Implementing CI/CD and How to Fix Them

1. **Challenge**: Lack of Automated Testing

If there aren't enough automated tests, it's hard to ensure the quality of code during fast deployments.

Solution: Invest in comprehensive tests, including unit, integration, and regression tests, to catch errors early.

2. **Challenge**: Tooling Complexity

Setting up a CI/CD pipeline involves integrating different tools for version control, testing, and deployment, which can be confusing.

Solution: Use all-in-one platforms like Jenkins CI/CD that simplify the process and reduce the number of tools we need to manage.

3. **Challenge**: Resistance to Change

Some team members may be hesitant to switch to CI/CD because they're used to older ways of working.

Solution: Provide training and showcase how CI/CD improves the workflow through pilot projects to help team members get on board.

4. **Challenge**: Infrastructure Limitations

If the infrastructure is not powerful enough, builds might be slow or unreliable.

Solution: Use cloud-based CI/CD solutions, which offer more scalable resources and faster build times.

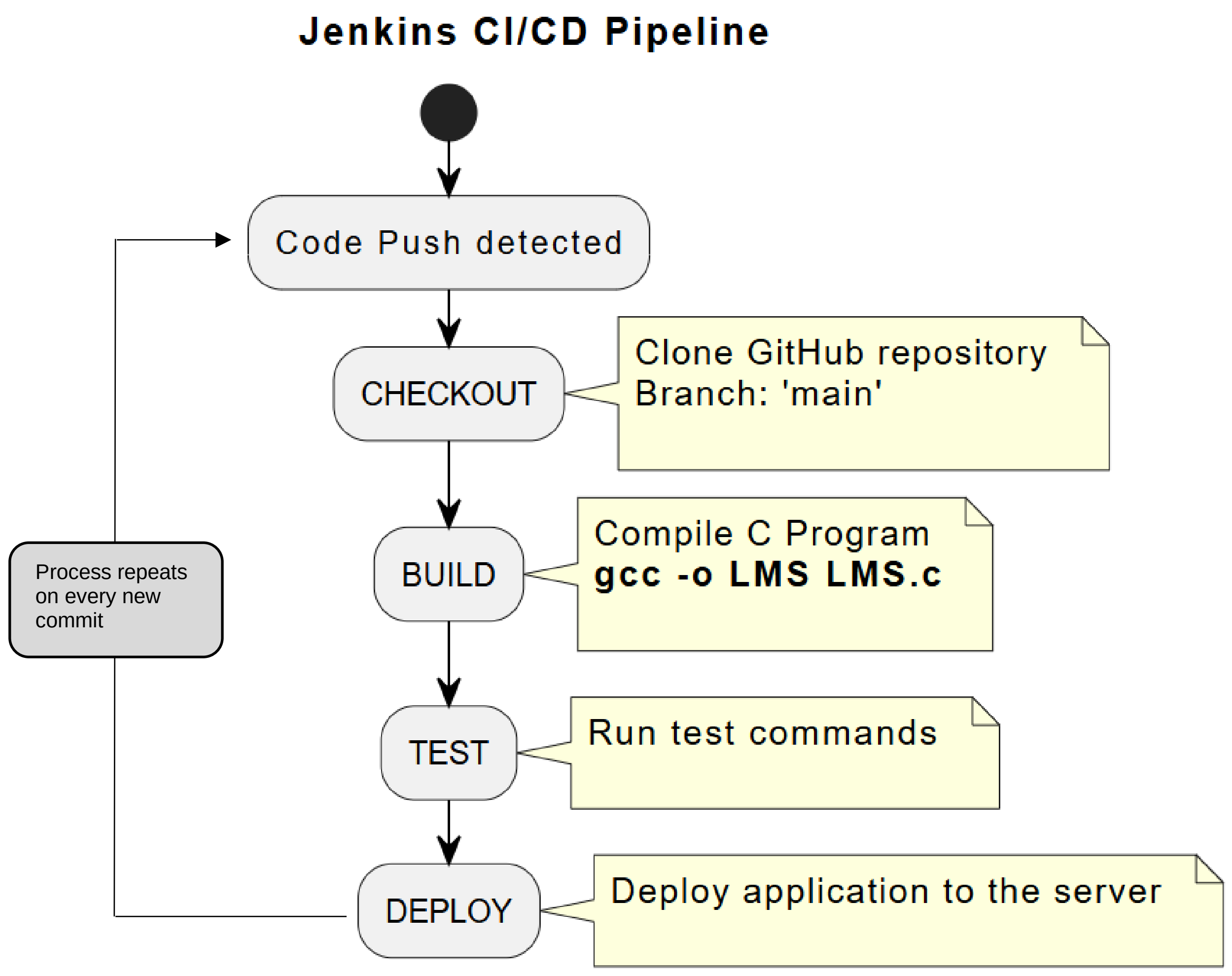


Diagram of CI/CD Pipeline

Assignment Publishing Date:	March 6, 2025
Tools Used:	Jenkins
Assigned By:	Irshad Hossain (ID: 2303030)