Demo Project: Create a CI Pipeline with Jenkinsfile (Freestyle, Pipeline, Multibranch Pipeline)

Jenkins Job Type: Freestyle

Step 1: Create a Freestyle Job in Jenkins

1. **Create a Pipeline Job**: In Jenkins, navigate to "New Item" and create a job named, for example, `java-maven-build`. Choose “Freestyle project”.
2. **Configure the Pipeline**: Connect this pipeline build to your GitLab repository.
    - Add the Repository URL.
    - Select the credentials.
    - Branch Specifier: `*/jenkins-jobs`

Push Image to Docker Hub

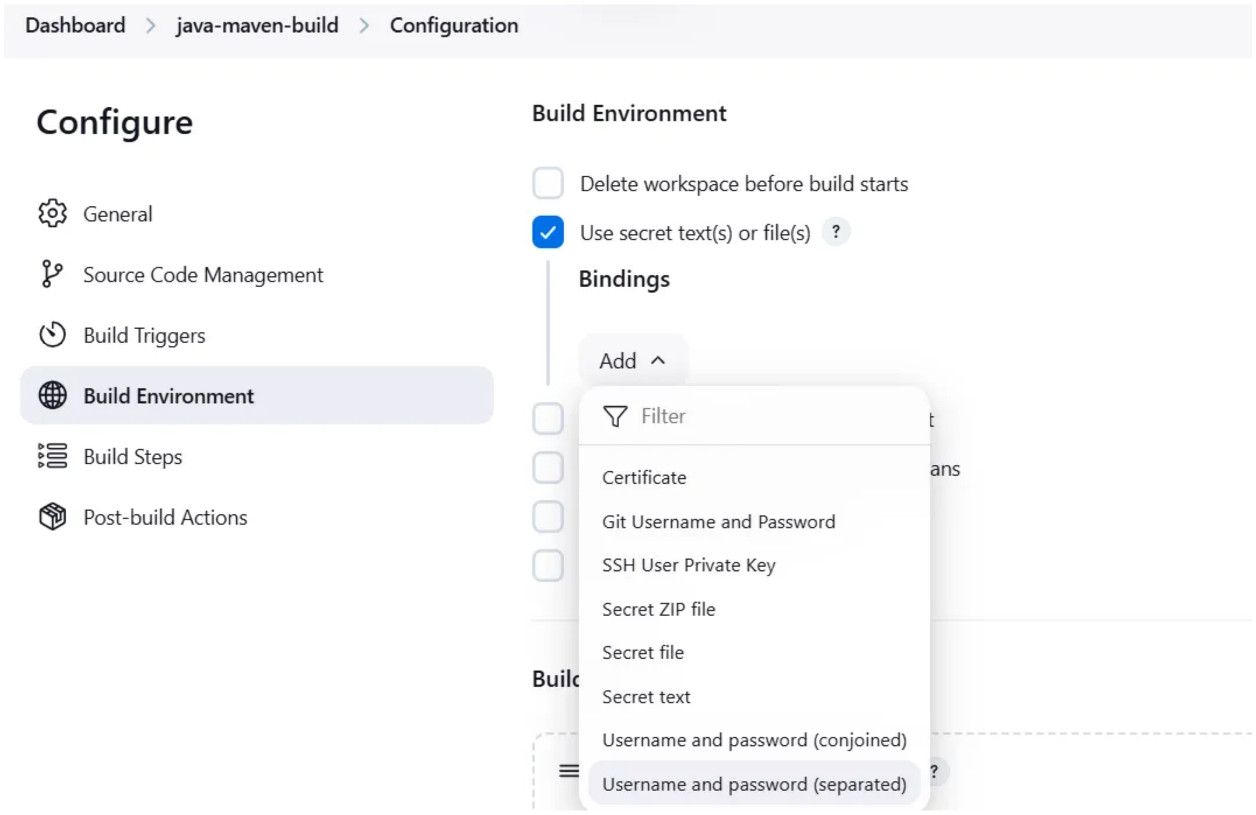
Follow the steps below to configure and push the Docker image to DockerHub:

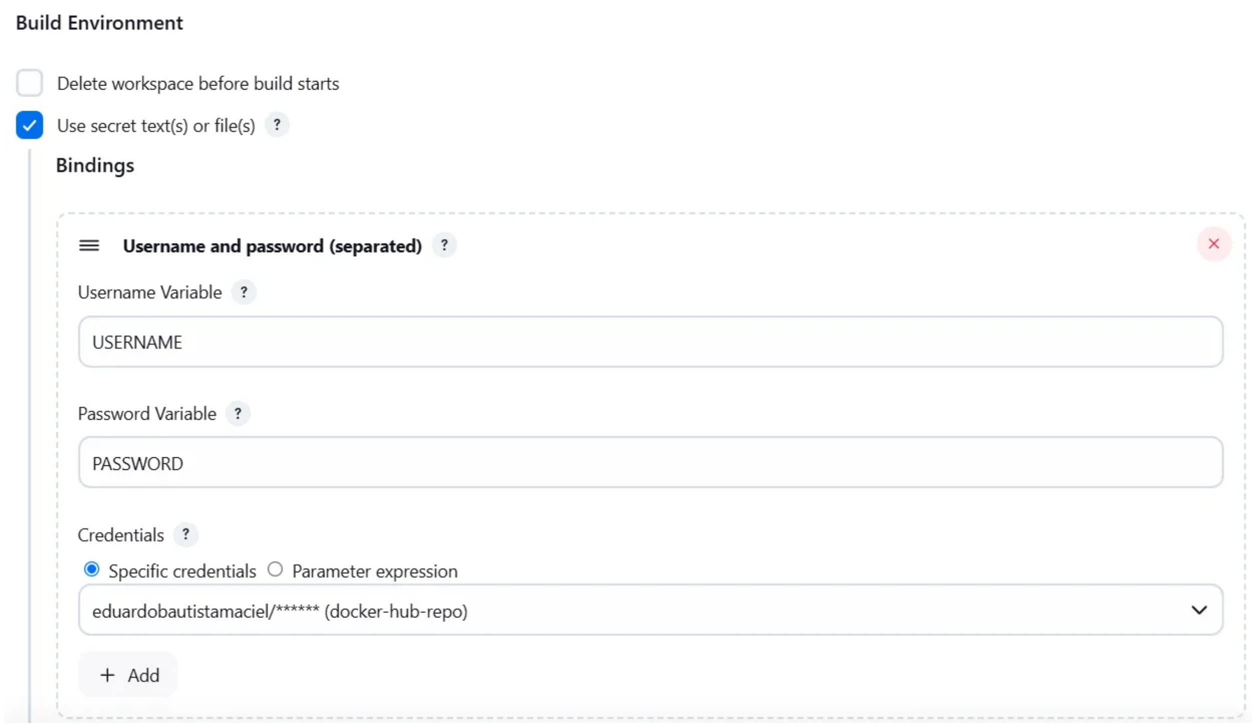
Step 1: Create a Private Repository on DockerHub

1. In DockerHub, go to "Create a Repository" and name it **demo-app** as a private repository.

Step 2: Configure Jenkins Environment Variables

1. In Jenkins, go to your job and click on **"Configure"**.
2. Under **"Build Environment"**, select the following to make the `USERNAME` and `PASSWORD` variables available throughout the build:


Dashboard > java-maven-build > Configuration
Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Build Environment
Delete workspace before build starts
Use secret text(s) or file(s)
Bindings
Add
Filter
Certificate
Git Username and Password
SSH User Private Key
Secret ZIP file
Secret file
Secret text
Username and password (conjoined)
Username and password (separated)


Build Environment

☐ Delete workspace before build starts

☑ Use secret text(s) or file(s)

Bindings

Username and password (separated)

Username Variable: USERNAME

Password Variable: PASSWORD

Credentials: [✓ Specific credentials] Parameter expression

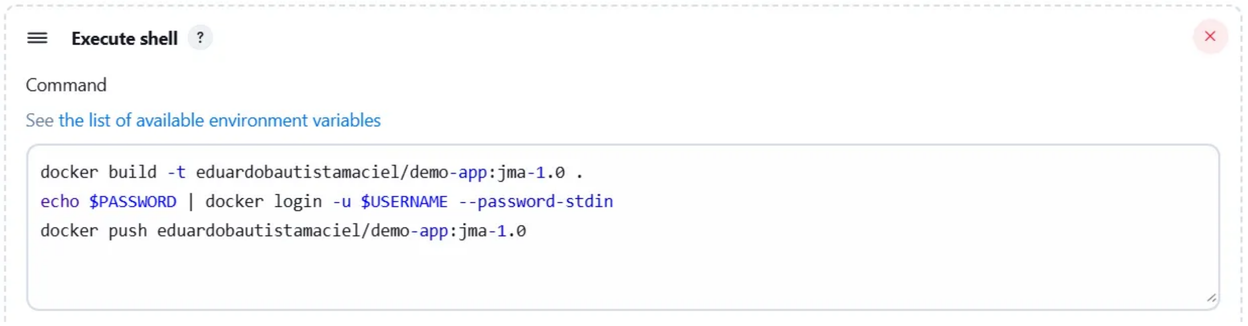
eduardobautistamaciel/****** (docker-hub-repo)

Step 3: Configure Build Steps

1. Scroll down to the build steps section and configure it to build the image by adding the following commands:

Note: Best practice in Docker is to provide the password as standard input.

```
docker build -t eduardobautistamaciel/demo-app:jma-1.0 .
echo $PASSWORD | docker login -u $USERNAME --password-stdin
docker push eduardobautistamaciel/demo-app:jma-1.0
```


Execute shell

Command

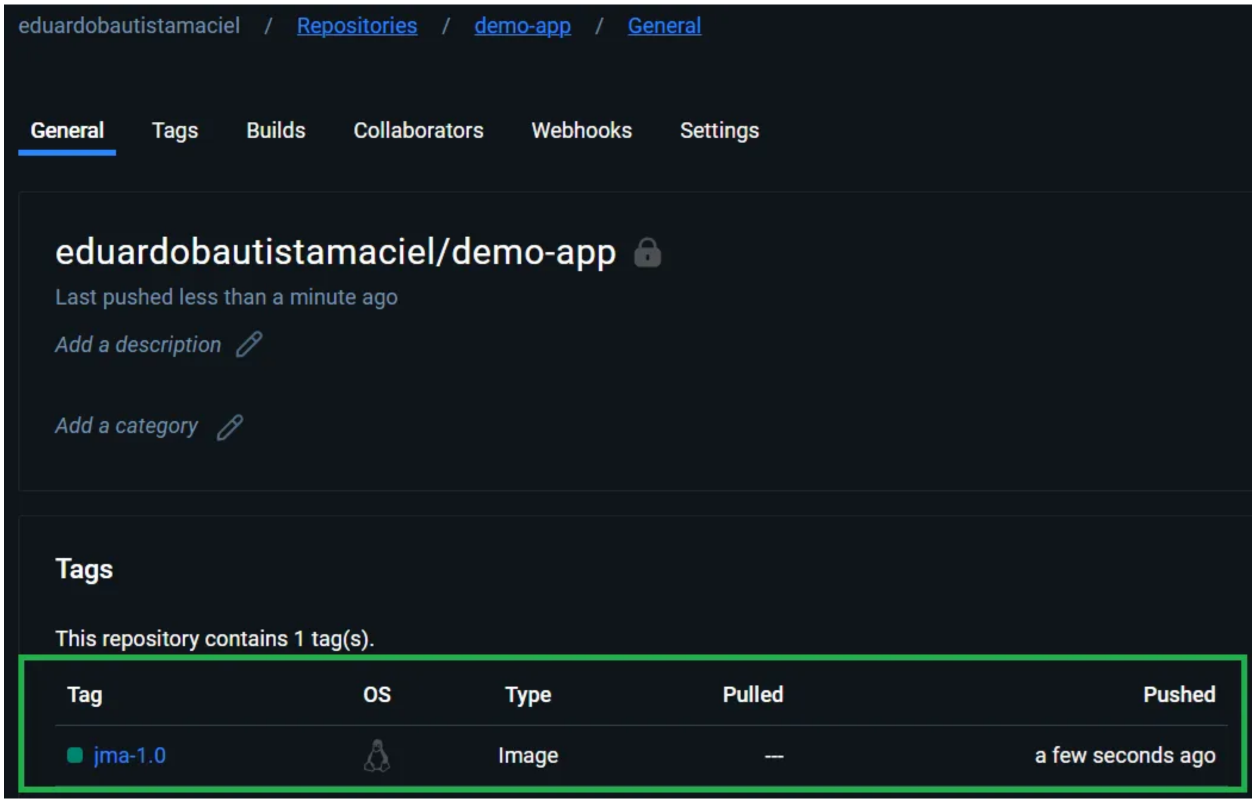
See the list of available environment variables

```
docker build -t eduardobautistamaciel/demo-app:jma-1.0 .
echo $PASSWORD | docker login -u $USERNAME --password-stdin
docker push eduardobautistamaciel/demo-app:jma-1.0
```

Step 4: Save and Build

1. Click "Save" to save your configuration.
2. Click "Build Now".

3. Once the build completes, check the "Console Output" to confirm the build and push were successful.
4. Verify the Docker image in DockerHub under the demo-app repository.



eduardobautistamaciel / Repositories / demo-app / General

eduardobautistamaciel/demo-app

Last pushed less than a minute ago

Add a description

Add a category

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
jma-1.0		Image	—	a few seconds ago

Jenkins Job Type: Pipeline

Note: before you start you need to have a Jenkinsfile in the branch

Step 1: Create a Pipeline Job in Jenkins

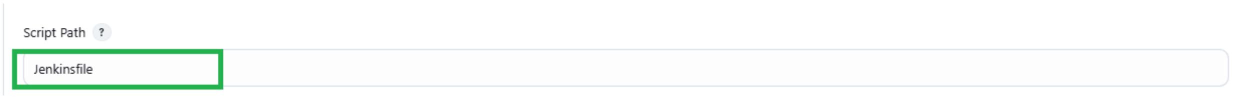
1. **Create a Pipeline Job**: In Jenkins, navigate to "New Item" and create a job named, for example, `my-pipeline`. Choose “pipeline”.
2. **Configure the Pipeline**: Connect this pipeline build to your GitLab repository.
    - Add the Repository URL.
    - Select the credentials.

- Branch Specifier: `*/jenkins-jobs`

2. **Configure the Pipeline**: Connect this pipeline build to your GitLab repository.

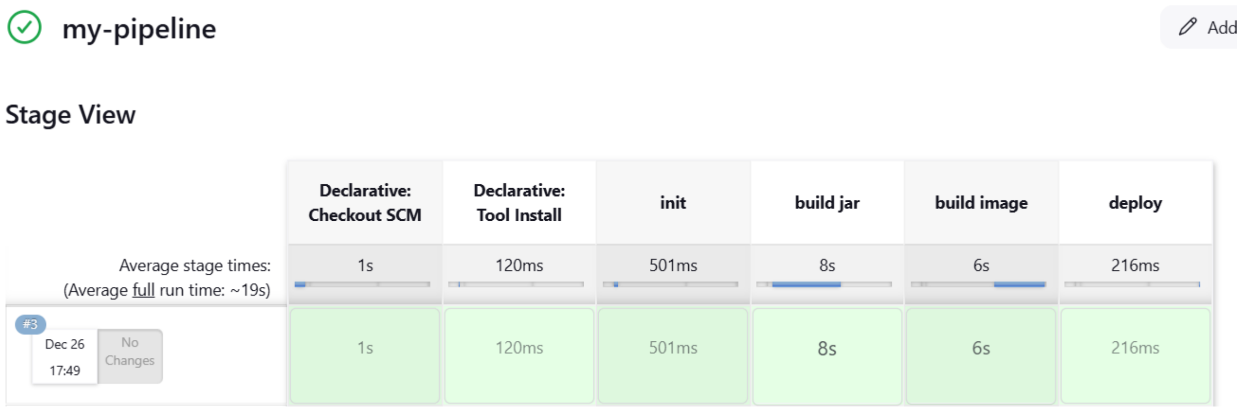
- Add the Repository URL.
- Select the credentials.
- Choose the appropriate branch.

3. **Check the Script Path**: Ensure the "Script Path" is set to `Jenkinsfile`. This means Jenkins will look for the `Jenkinsfile` in the root folder of the repository, which contains the pipeline configuration written in Groovy.


Script Path ?
Jenkinsfile

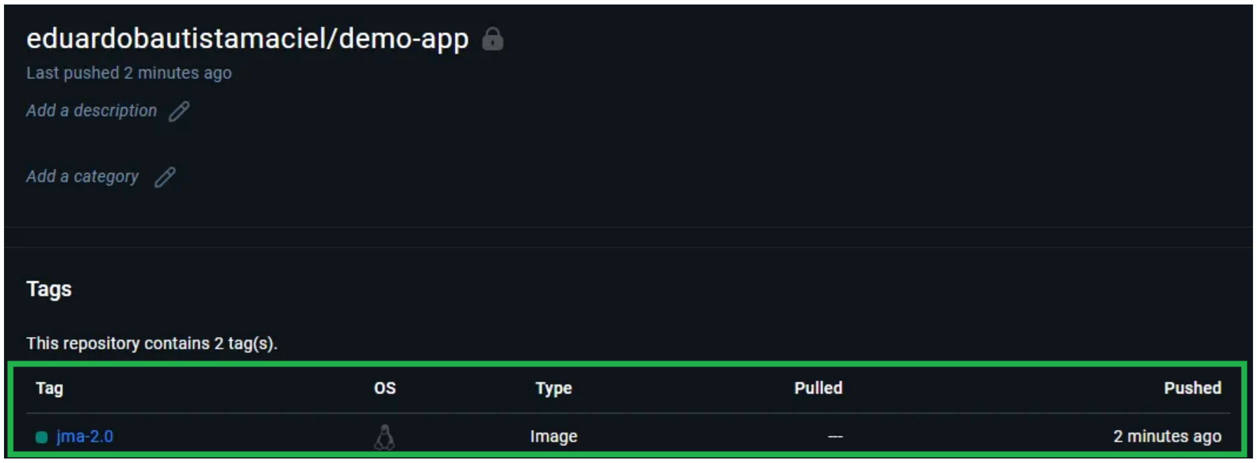
Step 2: Save and Build the Pipeline

1. Click **Save** to save your configuration.
2. Click **Build Now** to start the pipeline.
3. Once the build completes:
   - Check the "Console Output" to confirm the build and push were successful.



	Declarative: Checkout SCM	Declarative: Tool Install	init	build jar	build image	deploy
Average stage times: (Average full run time: ~19s)	1s	120ms	501ms	8s	6s	216ms
#3 Dec 26 17:49 No Changes	1s	120ms	501ms	8s	6s	216ms

- Verify the Docker image in DockerHub under the appropriate repository.


eduardobautistamaciel/demo-app
Last pushed 2 minutes ago
Add a description
Add a category

Tags
This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
jma-2.0		Image	---	2 minutes ago

Jenkins Job Type: Multibranch Pipeline

In this section, we extend the pipeline configuration by creating a **Multibranch Pipeline**. The **Multibranch Pipeline** automatically creates and manages branches in your pipeline job, allowing you to have different pipeline configurations for different branches (e.g., `develop`, `feature`, `main`).

To set up the Multibranch Pipeline:

1. **Add a Branch Conditional (Optional)**:

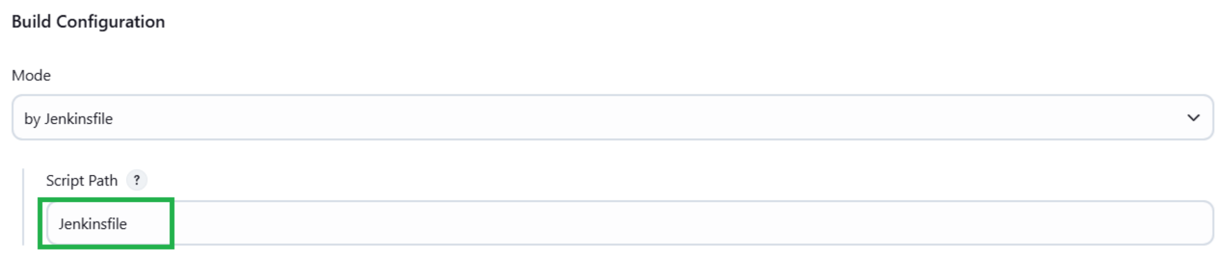
   If needed, add a condition in the Jenkinsfile to restrict builds and deployments to specific branches. For this project, only the `main` branch will build and deploy.

2. **Create a Pipeline Job**: In Jenkins, navigate to "New Item" and create a job named, for example, `my-multi-branch-pipeline`. Choose: Multibranch Pipeline

3. **Configure the Pipeline**: Connect this pipeline build to your GitLab repository.

   - Add the Repository URL.
   - Select the credentials.
   - Select: **Filters with a regular expression** and add the following `.*` to build all branches and dynamically discover new ones when created.

4. **Check the Script Path**: Ensure the "Script Path" is set to `Jenkinsfile`. This means Jenkins will look for the `Jenkinsfile` in the root folder of all repositories.



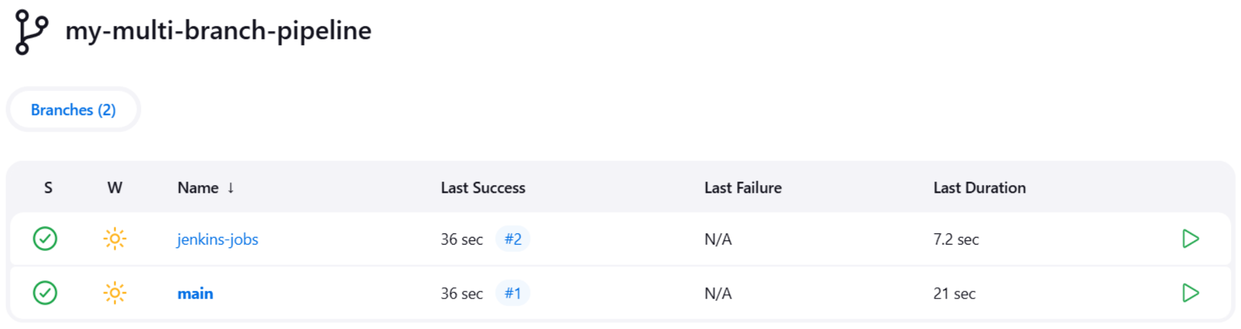
Build Configuration

Mode: by Jenkinsfile

Script Path: Jenkinsfile

Step 2: Save and Build the Pipeline

1. Click **Save** to save your configuration.
2. Click **Scan Multibranch Pipeline Now** to start scanning and building detected branches.

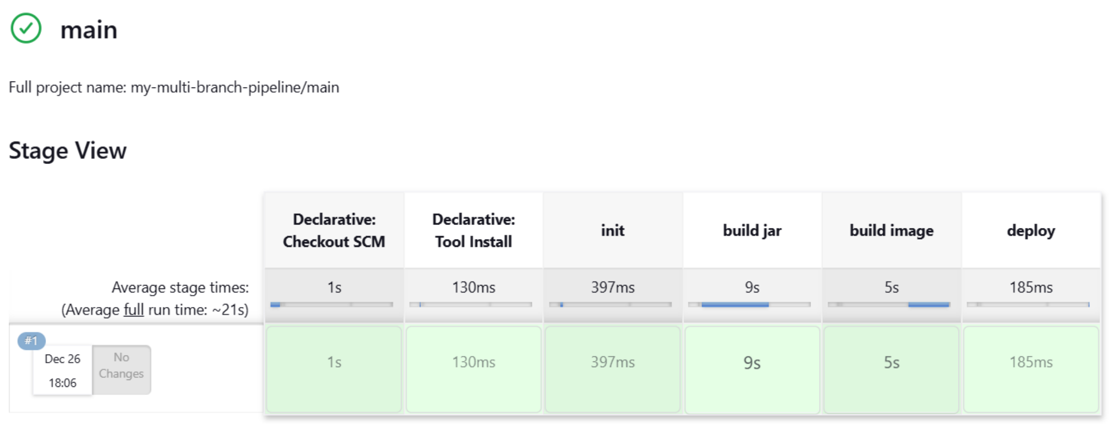


my-multi-branch-pipeline

Branches (2)

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀	jenkins-jobs	36 sec #2	N/A	7.2 sec
✓	☀	main	36 sec #1	N/A	21 sec

3. Once the build completes:
   - Check the **"Console Output"** of each job to confirm the build and, if applicable, the push was successful.
   - Note: In Step 1, a Branch Conditional was added for a specific stage to execute only if the branch is main. As a result:
     - The main branch will build and deploy successfully.



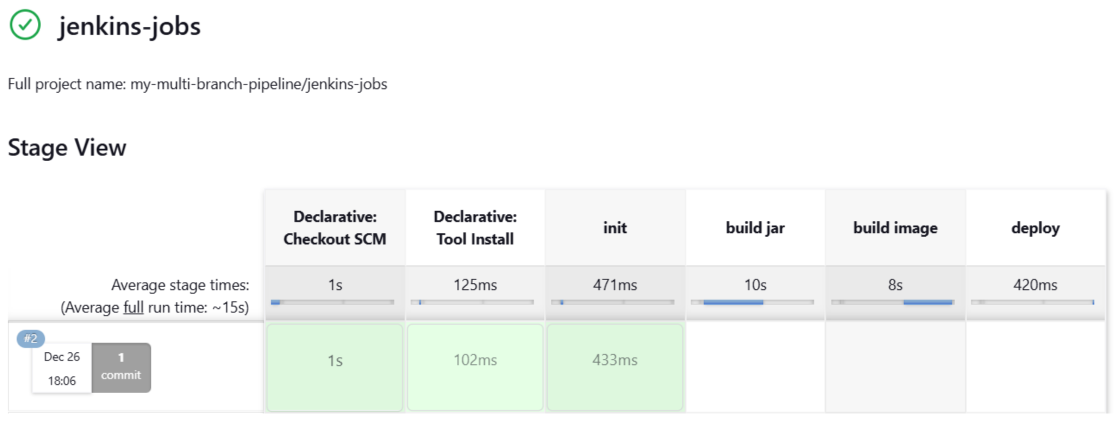
✓ main

Full project name: my-multi-branch-pipeline/main

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	init	build jar	build image	deploy
Average stage times: (Average full run time: ~21s)	1s	130ms	397ms	9s	5s	185ms
#1 Dec 26 18:06 No Changes	1s	130ms	397ms	9s	5s	185ms

- Other branches will skip the deployment stage, as designed.



✓ jenkins-jobs

Full project name: my-multi-branch-pipeline/jenkins-jobs

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	init	build jar	build image	deploy
Average stage times: (Average full run time: ~15s)	1s	125ms	471ms	10s	8s	420ms
#2 Dec 26 18:06 1 commit	1s	102ms	433ms			