

Build a Docker Jenkins Pipeline to Implement CI/CD Workflow

By Lei Liu

Intro on the project

The goal of this project is to implement DevOps, facilitate CI/CD docker Jenkins pipeline to develop and manage the products. The tasks that need to be taken and managed include planning on the requirement, system config, and tracking the efficiency.

Steps of building CI/CD pipeline

Pre-requisite/tools:

1. working platform: Ubuntu Linux system

2. Install and check java on Ubuntu

```
$sudo apt-get update
```

```
$sudo apt-get install fontconfig openjdk-11-jre
```

```
$java --version
```

3. Install and check git on Ubuntu

```
$sudo apt-get update
```

```
$sudo apt install git-all
```

```
$git --version
```

4. Install and check Docker

Set up the Docker repository

```
$sudo apt-get update
```

```
$sudo apt-get install ca-certificates apt-transport-https curl lsb-release gnupg
```

Add official GPG key for docker

```
$curl -fsSL https://download.docker.com/linux/ubuntu/gpg
```

```
$sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Install and check docker

```
$sudo apt-get docker
```

```
$docker --version
```

```
$docker version
```

Add current user as docker super user to avoid using sudo command

List the available versions:

```
$apt-cache madison docker-ce | awk '{ print $3 }'
```

```
5:23.0.5-1~ubuntu.22.04~jammy
```

```
5:23.0.4-1~ubuntu.22.04~jammy
```

```
5:23.0.3-1~ubuntu.22.04~jammy
```

```
5:23.0.2-1~ubuntu.22.04~jammy
```

```
5:23.0.1-1~ubuntu.22.04~jammy
```

```
5:23.0.0-1~ubuntu.22.04~jammy
```

```
5:20.10.24~3-0~ubuntu-jammy
```

Select the desired version and install

create a variable

```
VERSION_STRING=5:23.0.5-1~ubuntu.22.04~jammy
```

```
$sudo apt-get install docker-ce=$VERSION_STRING docker-ce-  
cli=$VERSION_STRING containerd.io docker-buildx-plugin docker-compose-plugin
```

Verify

```
$sudo docker run hello-world
```

Follow steps from this web: <https://docs.docker.com/engine/install/linux-postinstall/>

Steps:

Add your user to the docker group

```
$sudo groupadd docker
```

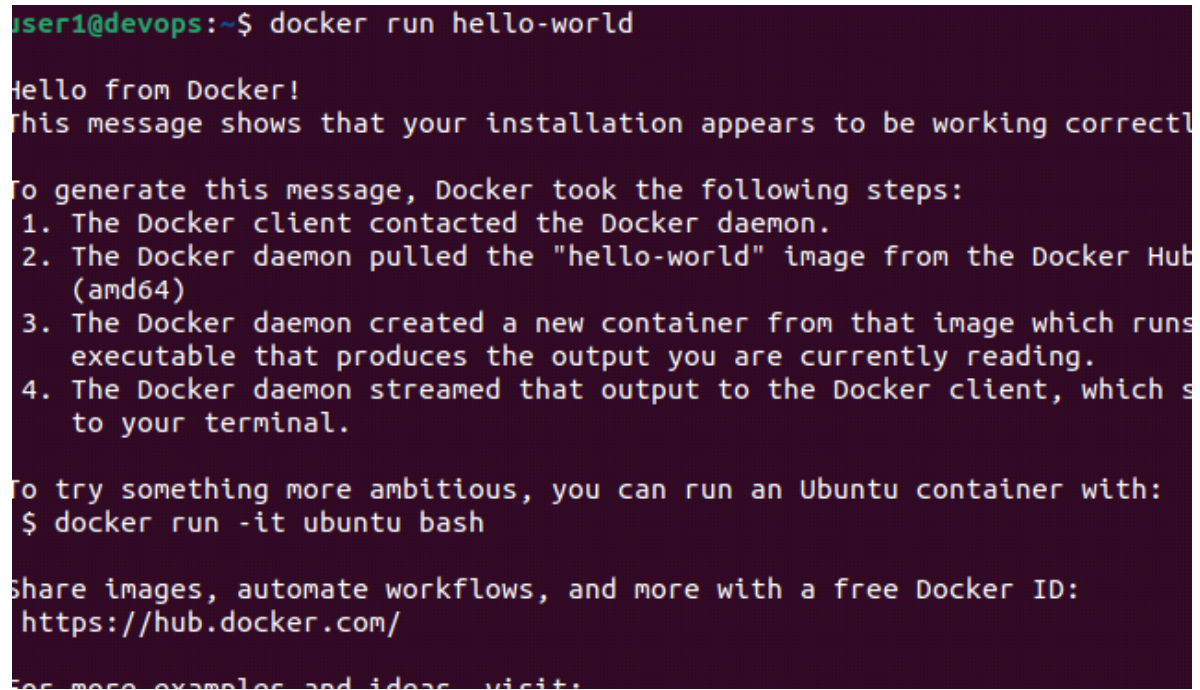
Activate the changes to groups:

```
$sudo usermod -aG docker $USER
```

```
$newgrp docker
```

Verify without using sudo command

```
$docker run hello-world
```

A terminal window with a dark purple background. The prompt is 'user1@devops:~\$'. The command 'docker run hello-world' has been executed. The output is: 'Hello from Docker! This message shows that your installation appears to be working correctly. To generate this message, Docker took the following steps: 1. The Docker client contacted the Docker daemon. 2. The Docker daemon pulled the "hello-world" image from the Docker Hub (amd64) 3. The Docker daemon created a new container from that image which runs executable that produces the output you are currently reading. 4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal. To try something more ambitious, you can run an Ubuntu container with: \$ docker run -it ubuntu bash Share images, automate workflows, and more with a free Docker ID: https://hub.docker.com/ For more examples and ideas, visit: https://docs.docker.com/get-started/next/'

```
user1@devops:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/next/
```

5. Set up docker hub account

Open browser: <https://www.hub.docker.com>

Create an account:

username: llszllsz22

pw: ***

6. Install Jenkins

Add key of debian package repo to your system:

```
$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Then add a Jenkins apt repo entry:

```
$ echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
```

Update local package index, then finally install Jenkins:

```
$sudo apt-get update
```

```
$sudo apt-get install fontconfig openjdk-11-jre
```

```
$sudo apt-get install jenkins
```

The last, verify installation of Jenkins

```
$ jenkins --version
```

----- alternative -----

6. Install Jenkins using Docker on Ubuntu

Search the official image of Jenkins from docker hub by typing Jenkins

Install Jenkins the latest version use below command

```
$docker run --name jenkins -d jenkins/jenkins:lts
```

Verify jenkins installation and status

```
$jenkins --version
```

```
$sudo systemctl status jenkins
```

If jenkins is not active, use below command to activate

```
$sudo systemctl start jenkins
```

7. Setup GitHub account

Open browser: <https://github.com>

Create an account:

username: llszllsz22

pw:***

Use Docker to build an application and push it to the Docker Hub.

Custom-build an image using Dockerfile

Create and access a directory

\$mkdir demo

\$cd demo

Create Dockerfile

\$nano Dockerfile

comments

FROM ubuntu

MAINTAINER llszlisz22@gmail.com

RUN apt-get update

RUN apt-get install -y nginx

CMD ["echo","Image created"]

Create an image from Dockerfile, name of the image: customimg

\$docker build -t customimg .

Verify image build, shown below

\$docker images

```
user1@devops:~/DemoDocker$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
customimg           latest      2cd46a4dc0b2  2 minutes ago 177MB
jenkins/jenkins     latest      bf31e7641f1e  13 hours ago 472MB
<none>              <none>     c2a84d92aa9b  6 days ago   77.8MB
```

Build a container from the custom image, name the container: customctn

```
$docker run -dP customctn
```

Check the container

```
$docker ps
```

Push an image to DockerHub:

Pre-requisit: DockerHub account id and password

account id: llszllsz22

password:***

Check and get the image_id

```
$docker images
```

Then tag your custom image with docker_id/image_name

```
$docker tag 2cd46a4dc0b2 llszllsz22/myimg:latest
```

Check the image, the name of the image should be like llszllsz22/myimg:latest

```
$docker images
```

```
user1@devops:~/DemoDocker$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
customimg	latest	2cd46a4dc0b2	11 minutes ago	177MB
llszllsz22/myimg	latest	2cd46a4dc0b2	11 minutes ago	177MB
jenkins/jenkins	latest	bf31e7641f1e	13 hours ago	472MB

Docker hub authenticate with username and password. After successful login, you should see Login succeeded.

```
$docker login
```

Type your username and password

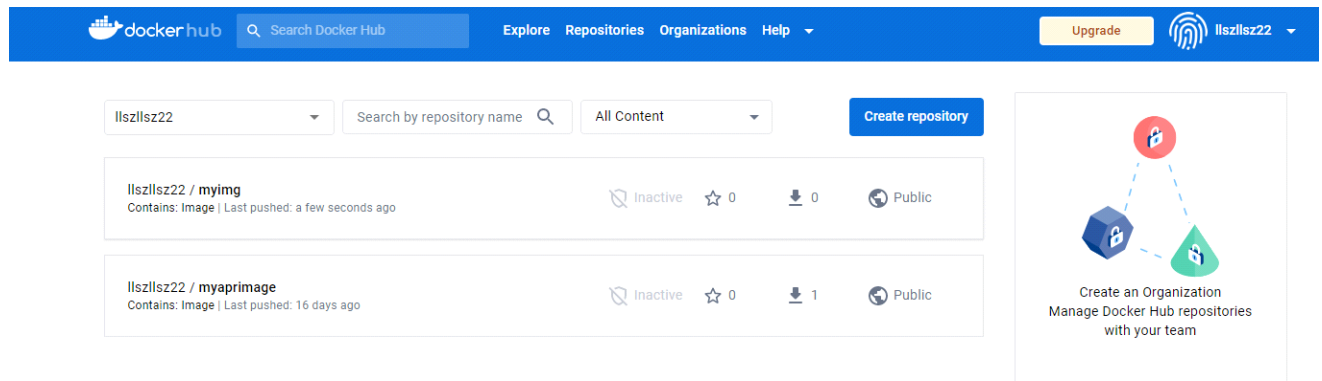
```
llszllsz22
```

```
***
```

Push the custom image to dockerhub

```
$docker push llszllsz22/myimg:latest
```

Verify the image from docker hub



Version control using git and GitHub

To connect and push file from local system to GitHub

Check git installation on Ubuntu

```
$git --version
```

Create, access a directory

```
$mkdir newrepo
```

```
$cd newrepo
```

Configure git

```
$git config --list
```

```
$git config --global user.name lksz22
```

```
$git config --global user.email lkszlsz22@gmail.com
```

Verify git config

```
$git config --list
```

Initialize the repo

```
$git init
```

Check the status of a git repo

```
$git status
```

Make changes by adding a file under the directory

```
$echo "this is a demo about git" > file1.txt
```

```
$git add .
```

```
$git status
```

Add a new version using git commit

```
$git commit -m "my first commit with file1"
```

```
$git status
```

```
$git log --oneline
```

Verify git remote before git push to remote repo on GitHub

```
$git remote -v
```

Add ssh credential to local repo

```
$git remote add ssh-origin git@github.com:llsz22/newrepo_fr_hub.git
```

Verify git remote

```
$git remote -v
```

Authentication for encryption and decryption purposes w/t using password

Create a ssh key pair (prt/pub) on the local machine

Press enter three times to generate the keys

The key pair shall be stored under ~/.ssh/

Steps:

```
~$ssh-keygen
```

Copy the pub key and past it into github

```
$cat ~/.ssh/id_rsa.pub
```

Log into github, paste the pub key content and generate authen key, go to

settings->SSH and GPG keys->New SSH key

Verify whether local machine authenticated with github

```
$ssh -T git@github.com
```

```
user1@devops:~$ ssh -T git@github.com
Hi llasz22! You've successfully authenticated, but GitHub does not provide shell access.
```

Move back to local repo directory, rename and verify branch name to match GitHub branch

```
$git branch -M main
```

Verify the change

```
$git log --oneline
```

Push local repo to remote repo on GitHub (sync local repo with remote repo on GitHub)

```
$git push -uf origin main
```

The screenshot shows the GitHub interface for a repository named 'newrepo_fr_hub' by user 'llasz22'. The repository is public and has 0 stars, 1 watcher, and 0 forks. The main branch is 'main' with 1 branch and 0 tags. The commit history shows a single commit 'my first commit w/file1' by 'llasz22' 14 minutes ago. The file 'file1.txt' is listed as part of this commit. A prompt encourages adding a README. The 'About' section mentions creating a repo at the hub for receiving from local. The 'Releases' section indicates no releases are published.

Using Jenkins to automate deployment

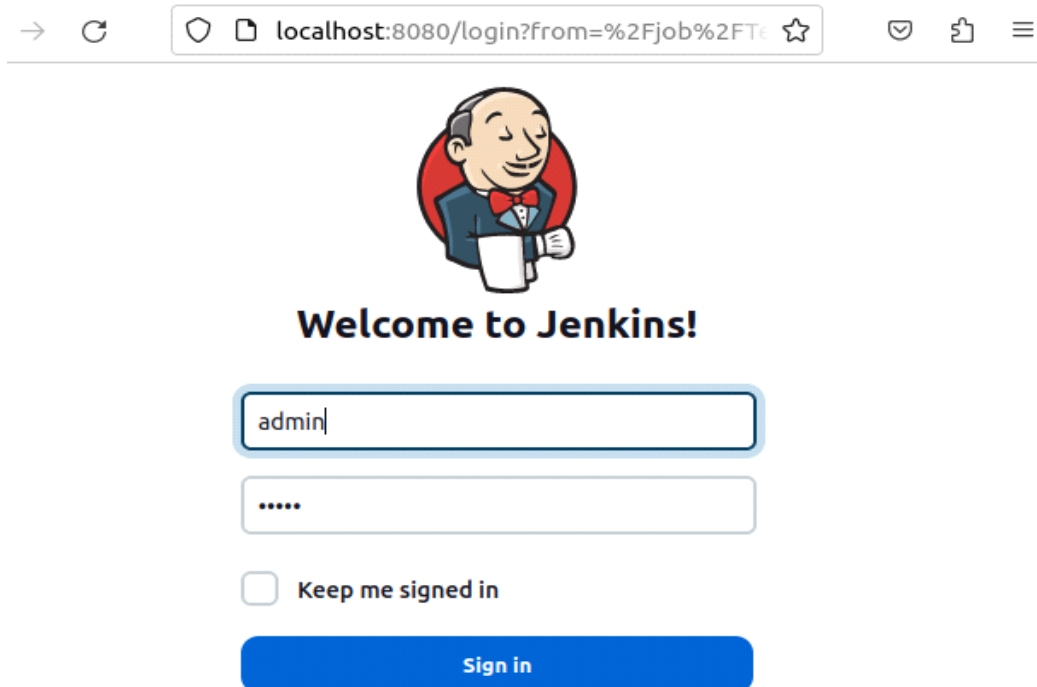
Verify Jenkins installation on Ubuntu

\$jenkins --version

Access jenkins server (on local machine in this case)

Browsing: localhost:8080

admin/admin (username and password)



The screenshot shows the Jenkins login interface in a web browser. The address bar displays 'localhost:8080/login?from=%2Fjob%2F...' with a star icon. Below the browser bar is the Jenkins logo, a cartoon character in a suit and bow tie. The text 'Welcome to Jenkins!' is centered. There are two input fields: the first contains 'admin' and the second contains masked characters '.....'. Below these fields is a checkbox labeled 'Keep me signed in' which is unchecked. At the bottom is a blue 'Sign in' button.

Jenkins job 1: create a Jenkins job on the Ubuntu server to use execute shell

On Jenkins dashboard, click New Item

Jenkins job: myjob1

Template: Freestyle

Build -> Add Build step -> Execute shell

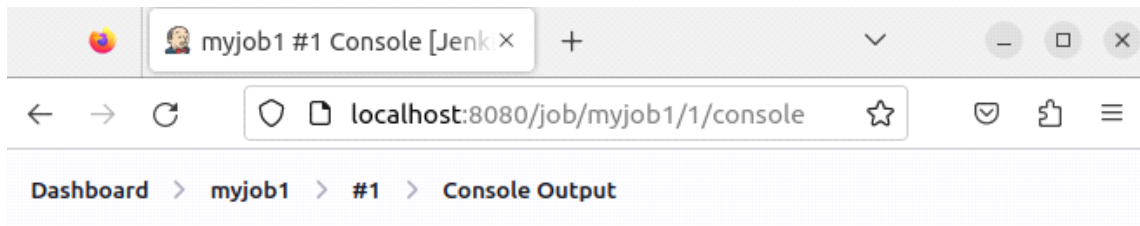
Command:

echo "Hello from Jenkins job --myjob"

>Build Now

myjob > console output

The job was successfully built shown as below



```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/myjob1
[myjob1] $ /bin/sh -xe /tmp/jenkins16170942977241017900.sh
+ echo Hello from Jenkins job-myjob
Hello from Jenkins job-myjob
Finished: SUCCESS
```

Back to the terminal, check the Job path:

```
$ ls -la /var/lib/jenkins/jobs/myjob1
```

```
user1@devops:~$ sudo ls -la /var/lib/jenkins/jobs/myjob1
total 20
drwxr-xr-x 3 jenkins jenkins 4096 May  2 20:59 .
drwxr-xr-x 5 jenkins jenkins 4096 May  2 20:56 ..
drwxr-xr-x 3 jenkins jenkins 4096 May  2 20:59 builds
-rw-r--r-- 1 jenkins jenkins  663 May  2 20:58 config.xml
-rw-r--r-- 1 jenkins jenkins    2 May  2 20:59 nextBuildNumber
```

+++++

Jenkins job 2: connect git repo to Jenkins, use Jenkins job to automate and deploy the process

On Jenkins dashboard, build a new job myjavajob

Steps:

Jenkins job: myjavajob

Template: Freestyle

SCM > Git

repo url: https://github.com/llsz22/newrepo_fr_hub.git

Modify branch : */main

Build > Add build step > Execute shell

Command:

javac HelloWorld.java

java HelloWorld

The job was successfully built as shown below



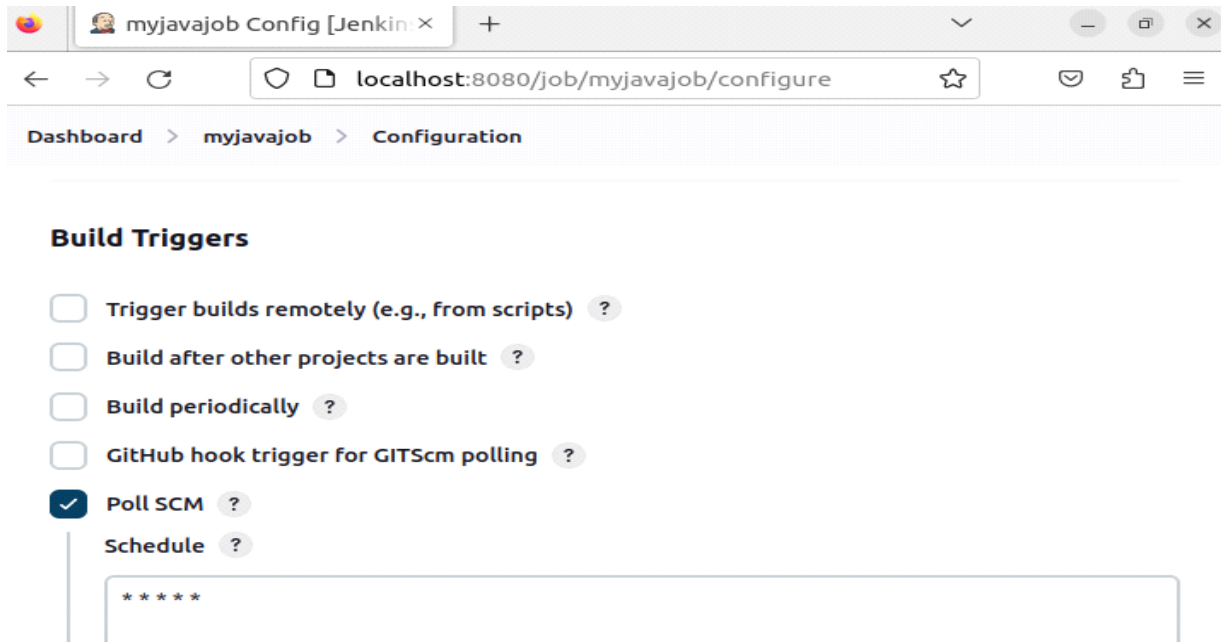
Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/myjavajob
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/llsz22/newrepo_fr_hub.git
> git init /var/lib/jenkins/workspace/myjavajob # timeout=10
Fetching upstream changes from https://github.com/llsz22/newrepo_fr_hub.git
> git --version # timeout=10
> git --version # 'git version 2.34.1'
> git fetch --tags --force --progress -- https://github.com/llsz22/newrepo_fr_hub.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/llsz22/newrepo_fr_hub.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision fb337efb908adff1937e6e71f46aaa59d60ec60f (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f fb337efb908adff1937e6e71f46aaa59d60ec60f # timeout=10
Commit message: "Add 2nd file"
First time build. Skipping changelog.
[myjavajob] $ /bin/sh -xe /tmp/jenkins3655302566325039566.sh
+ javac HelloWorld.java
+ java HelloWorld
Hello, world!
Finished: SUCCESS
```

Integrate Build Trigger using poll SCM to automatically update project

Steps:

Jenkins dashboard > myjavajob > Configuration > Build Trigger > "poll SCM" > schedule
* * * * * (meaning, it will trigger every minute)



Go to repo on Github, make some changes on the java file and save the changes.

Back to Jenkins, Jenkins dashboard > myjavajob

You should see the change on Github triggered an update of job auto build in a minute.

The screenshot shows the Jenkins web interface in a browser. The address bar indicates the URL is `localhost:8080/job/myjavajob/`. The breadcrumb navigation shows `Dashboard > myjavajob >`. Below the navigation, there are links for `Git Polling Log` and `Rename`. The main section is titled `Build History` with a `trend` dropdown. A search bar labeled `Filter builds...` is present. The build history table shows two builds: `#2` (green checkmark) at `May 3, 2023, 1:21 PM` and `#1` (green checkmark) at `May 3, 2023, 11:39 AM`. At the bottom, there are links for `Atom feed for all` and `Atom feed for failures`.

Click on #2 job, you should see console output as below, shows the trig succeeded.

Console Output

```
Started by an SCM change
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/myjavajob
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/myjavajob/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/llsz22/newrepo_fr_hub.git # timeout=10
Fetching upstream changes from https://github.com/llsz22/newrepo_fr_hub.git
> git --version # timeout=10
> git --version # 'git version 2.34.1'
> git fetch --tags --force --progress -- https://github.com/llsz22/newrepo_fr_hub.git +refs/heads,
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 1c20843e47ebd9fa64f235d6e01a645dbd84cabf (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 1c20843e47ebd9fa64f235d6e01a645dbd84cabf # timeout=10
Commit message: "Update HelloWorld.java"
> git rev-list --no-walk fb337efb908adff1937e6e71f46aaa59d60ec60f # timeout=10
[myjavajob] $ /bin/sh -xe /tmp/jenkins3531621821774904653.sh
+ javac HelloWorld.java
+ java HelloWorld
Hello, beautiful world!
Finished: SUCCESS
```

Conclusion

By now, the docker Jenkins CI/CD pipeline used to automate projects has been successfully implemented. The tasks include:

1. Install and configure Git, Jenkins, docker on Ubuntu server/local system.
2. set up docker hub account and GitHub account to manage version control of source code.
3. connect git with GitHub using ssh authentication pair key, push file from local system to GitHub repo, or clone repo on GitHub to local system, sync local repo with GitHub repo for sharing source code with other teams and accomplishing project integration.
4. create Dockerfile on local system, generate custom built image using Dockerfile, then use the image to create container, and push the container to docker hub as desired.
5. manage Jenkins by manage plugins, add plugins such as Ant, Maven, JDK; then configure global tools to your needs; build Jenkins jobs to deploy project automatically use Git SCM, also use poll trigger to pull changes from remote repo periodically based on desired time settings. all the changes and auto deployment kept in build history. As a integration platform, Jenkins has a lot more to offer and this project is just to demonstrate its rudimentary functionalities to automate simple tasks by integrating with other tools and plugins.

In general, this DevOps project demonstrates how to use git, GitHub, docker, docker hub, Jenkins pipeline to accomplish project source code version control and tracking, automate tasks, automate updating tasks based on desired settings on remote systems, demonstrates the core concept and practice of continuous integration and continuous delivery workflow.