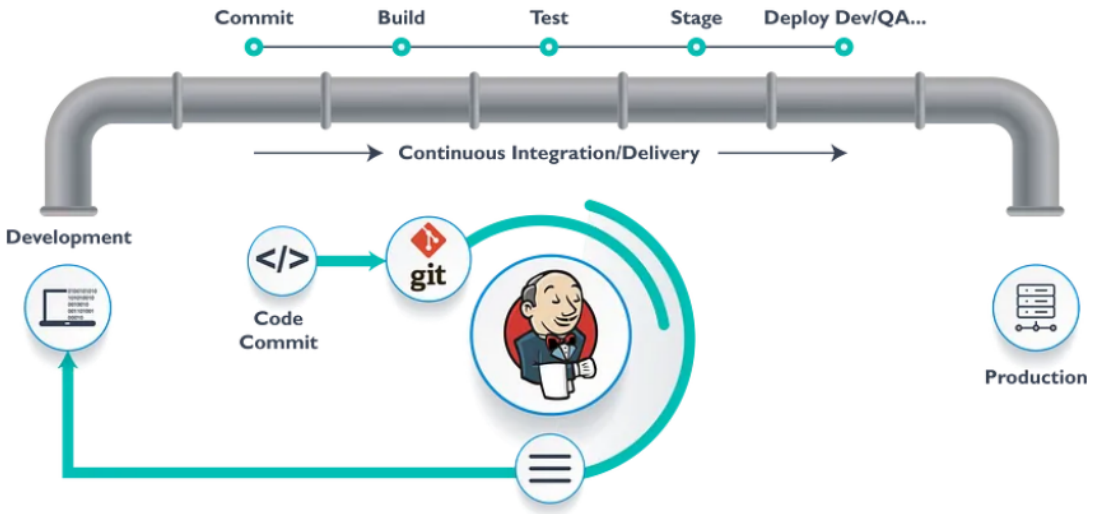
PROJECT 9: TOOLING WEBSITE DEPLOYMENT AUTOMATION WITH CONTINUOUS INTEGRATION

In Project 8, I deployed a Tooling website with 2 Webservers and a Load Balancer to allow for even distribution of traffic between them. The deployment of multiple servers will take a lot of resources and manpower if we use the same approach hence the need for Automation. The process of automation will allow the system to carry out a repeated set of processes to replace manual work done such as provisioning of servers. This project covers the automation of tasks with the help of Jenkins. Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, delivering or deploying software, facilitating continuous integration and continuous delivery.

Continuous integration (CI) is a software development strategy that increases the speed of development while ensuring the quality of the code that teams deploy. Developers continually commit code in small increments (at least daily, or even several times a day), which is then automatically built and tested before it is merged with the shared repository. The development cycle is based on the DevOps model with the help of Jenkins a CI/CD tool illustrated below. This ensures that every change made to the source code in GitHub https://github.com/<yourname>/tooling will be automatically updated to the Tooling Website



Step 1 - Launch an EC2 Instance

Create an AWS EC2 server based on Ubuntu Server 20.04 LTS. By default, the Jenkins server uses TCP port 8080 so ensure this Inbound Rule is set in the Security Group.

EC2 Dashboard
EC2 Global View

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4
Ansible	i-04bb48e760a699d72	Running	t2.micro	2/2 checks passed	No alarms	eu-west-2b	ec2-35-177-41-62.eu-w...	35.177.41.6

Step 2 – Install the Jenkins server

Install JDK as Jenkins is a Java-based application and Jenkins by running the following commands.

sudo apt update

sudo apt install default-jdk-headless

```
ubuntu@ip-172-31-40-224:~$ sudo apt install default-jdk-headless
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jre-headless fontconfig-config fonts-dejavu-core java-common
  libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libfontconfig1
  libgraphite2-3 libharfbuzz0b libjpeg-turbo8 libjpeg8 liblcms2-2 libpcsclite1
  openjdk-11-jdk-headless openjdk-11-jre-headless
```

Install Jenkins

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \
  /etc/apt/sources.list.d/jenkins.list'

sudo apt update

sudo apt-get install jenkins

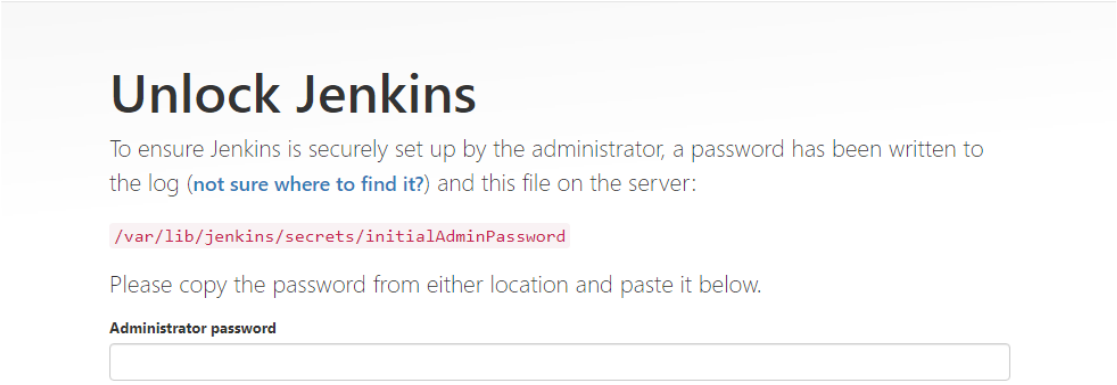
Make sure Jenkins is up and running: sudo systemctl status Jenkins

```
ubuntu@ip-172-31-40-224:~$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-40-224:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-40-224:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2023-07-04 21:07:14 UTC; 1min 7s ago
   Main PID: 4901 (java)
      Tasks: 36 (limit: 1141)
     Memory: 294.5M
     CGroup: /system.slice/jenkins.service
             └─4901 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war -
Jul 04 21:06:42 ip-172-31-40-224 jenkins[4901]: c5f04c8bcf1d4455b08c2e786cf4f032
Jul 04 21:06:42 ip-172-31-40-224 jenkins[4901]: This may also be found at: /var/lib/jenkins/s
Jul 04 21:06:42 ip-172-31-40-224 jenkins[4901]:
```

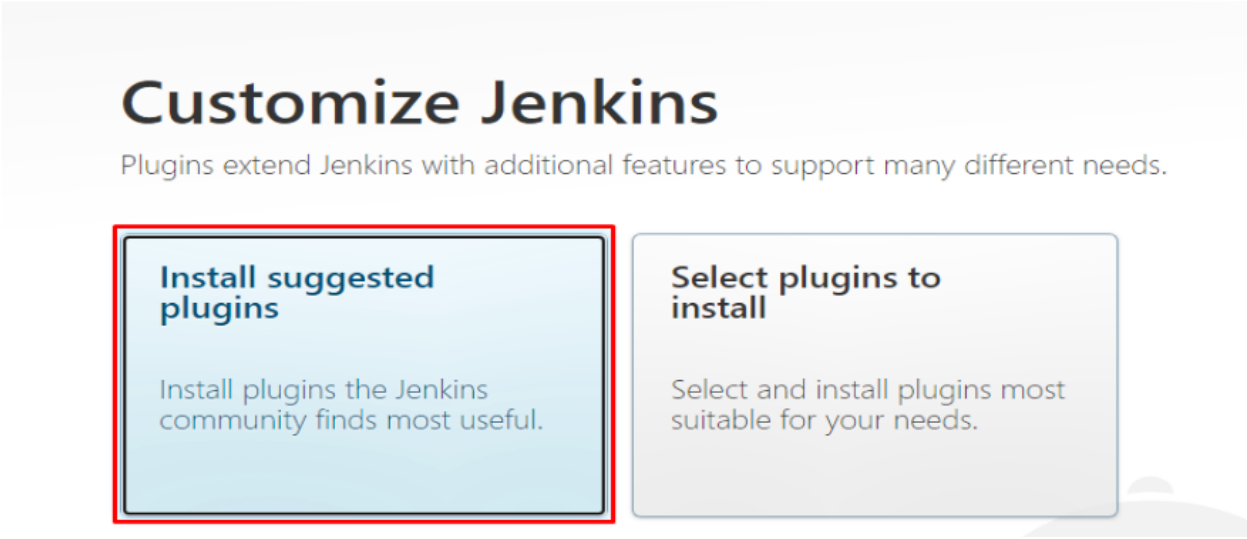
Perform initial Jenkins setup

From your browser access http://<Jenkins-Server-Public-IP-Address>:8080. You will be prompted to provide a default admin password. This password can be retrieved from the Jenkins server on the path stated on the browser page.

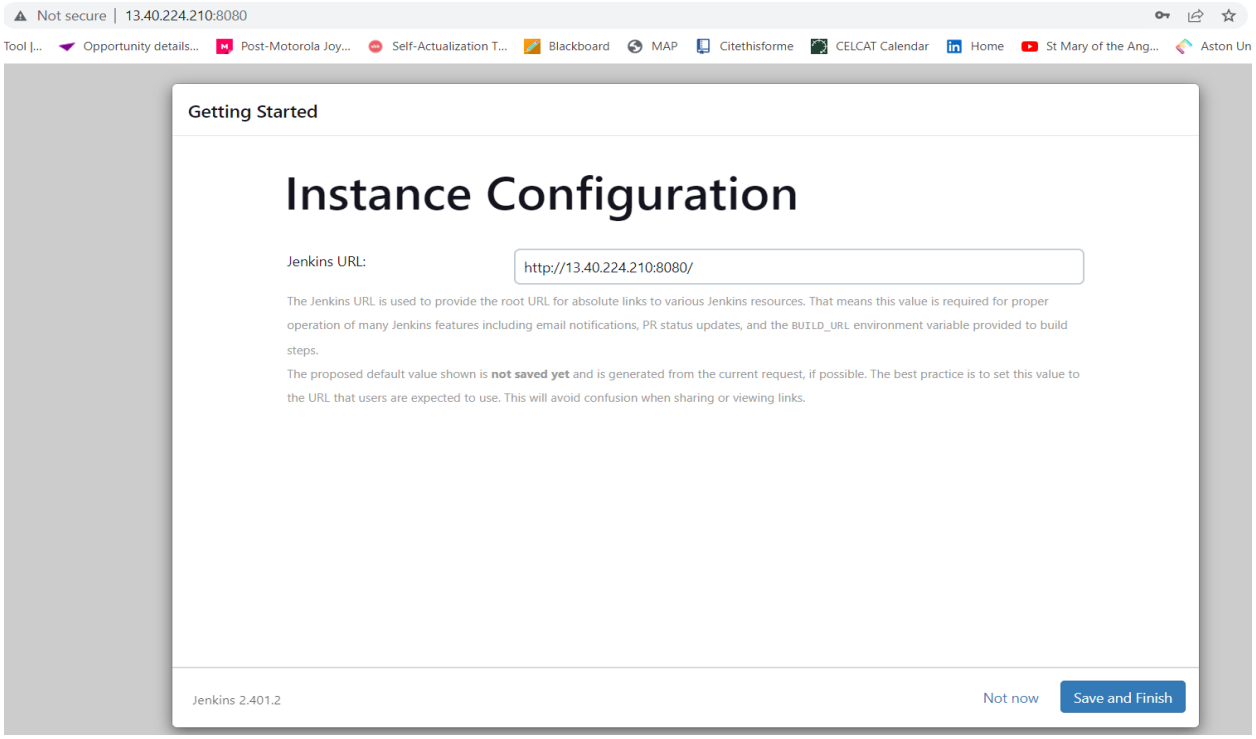
sudo cat /var/lib/jenkins/secrets/initialAdminPassword



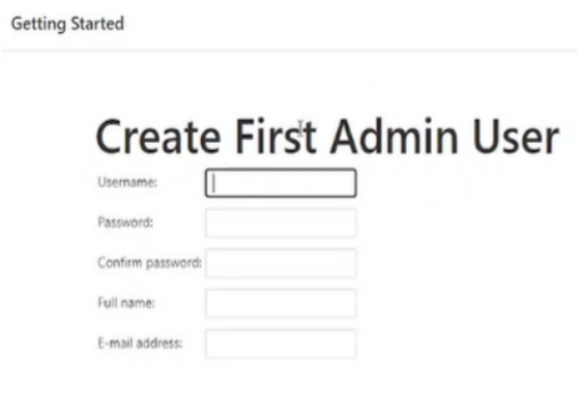
Then you will be asked which plugins you wish to use to customize Jenkins and select install suggested plugins.

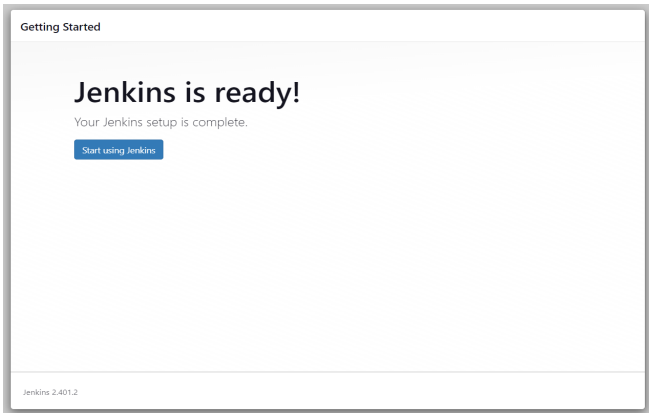


Set up the Jenkins URL in the configuration which is the same as the Jenkins server address.



Create an admin user and you will get the notification that the set-up is complete.

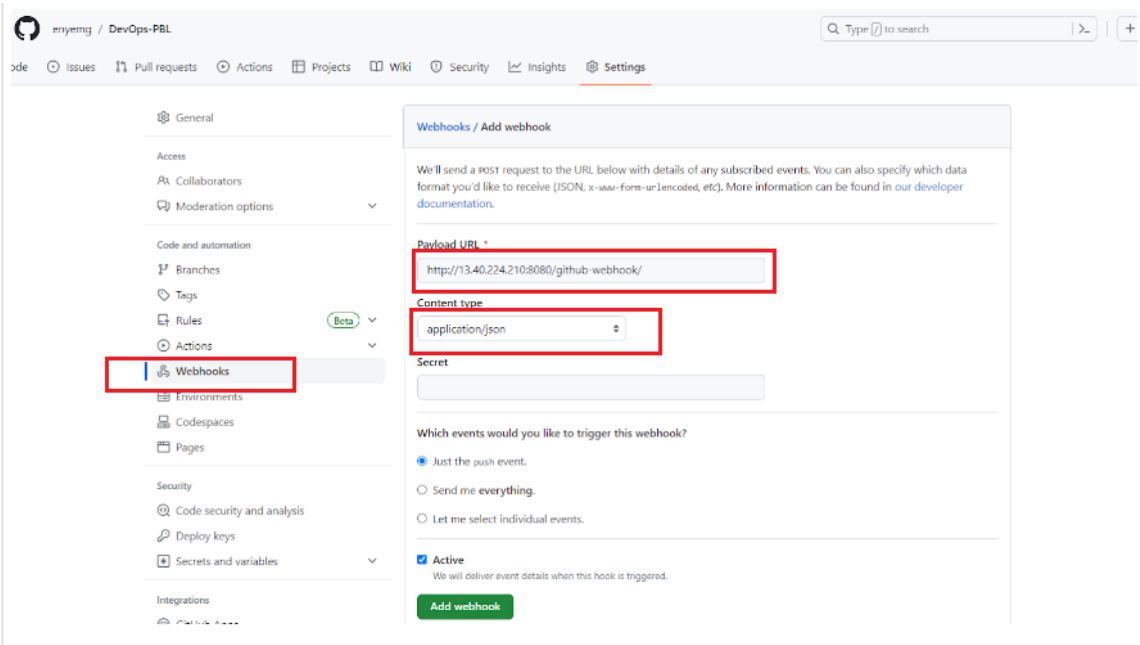




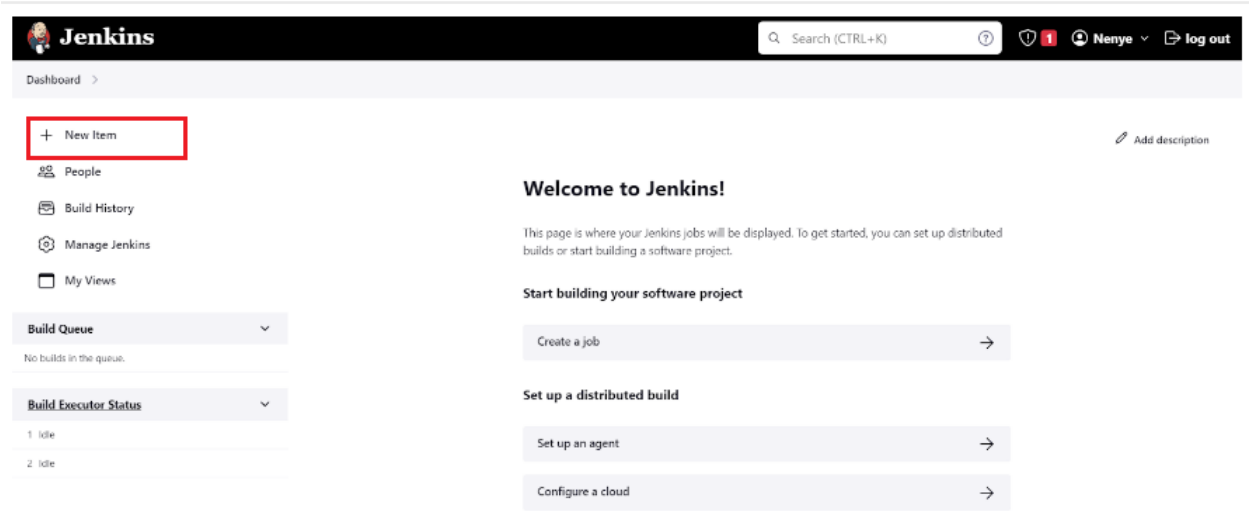
Step 3 – Configure Jenkins to retrieve source codes from GitHub using Webhooks

Set up a simple Jenkins job, and configure it to retrieve source codes from GitHub. This will be triggered by GitHub webhooks and will execute a ‘build’ task to retrieve codes from GitHub and store it locally on the Jenkins server.

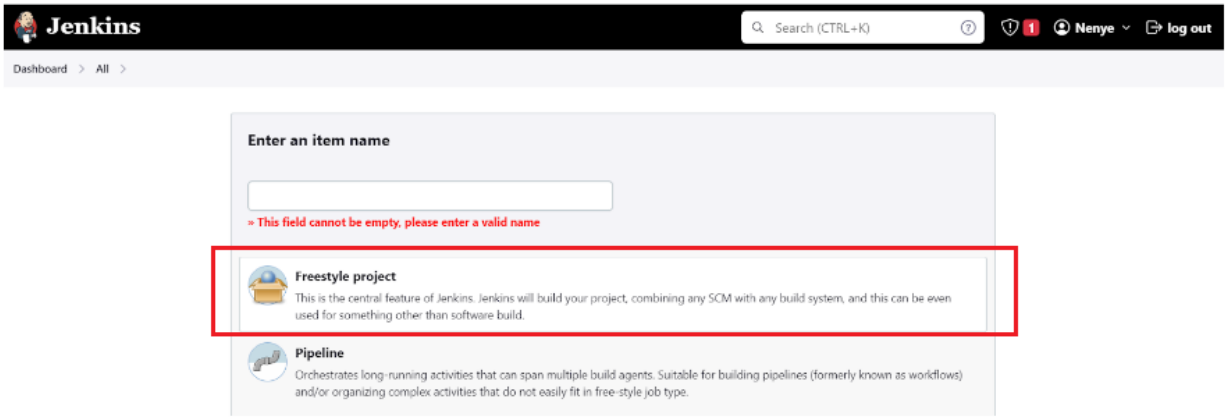
1. Enable webhooks in your GitHub repository settings. Go to settings > Add webhook > Webhooks > Type the payload URL (http://<Jenkins-Server-Public-IP-Address>:8080/github-webhook/) > Select Content type



2. Go to Jenkins web console, click “New Item”



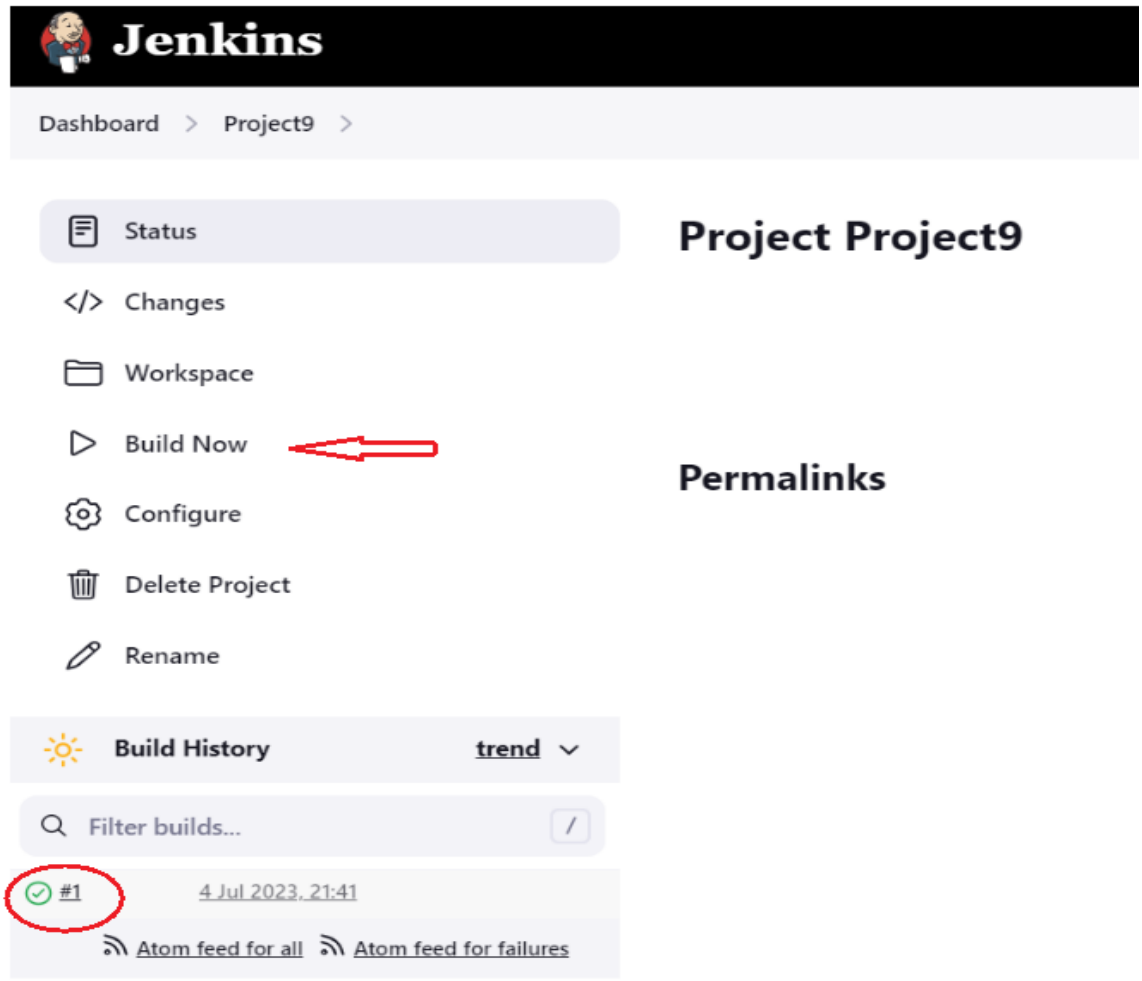
3. Select “Freestyle project” and enter an item name - I used Project 9 in this case.



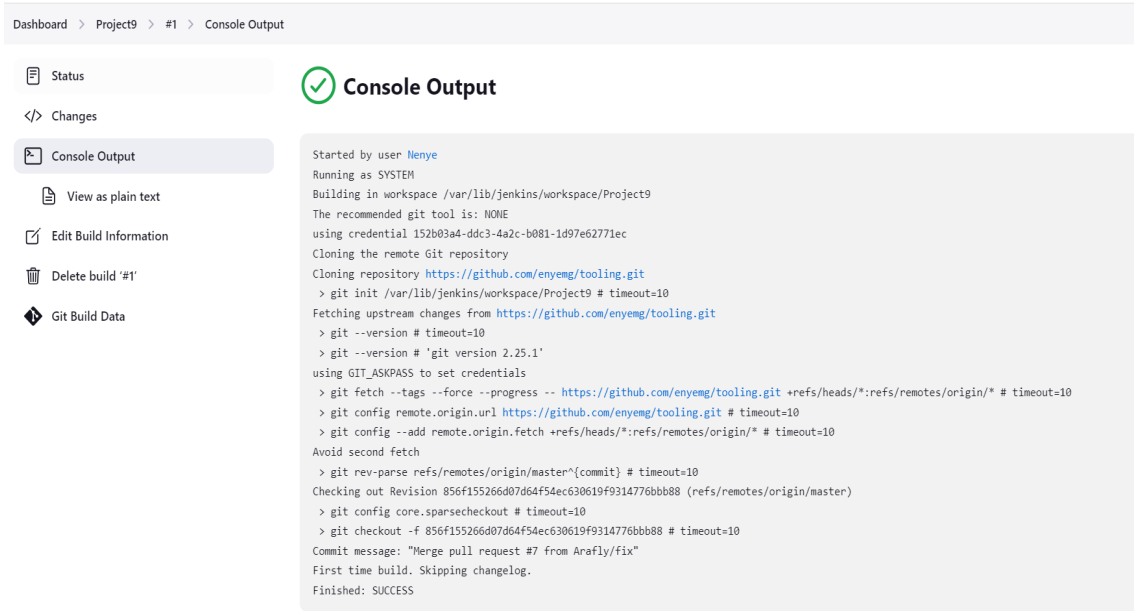
4. Provide the link to your Tooling GitHub repository and credentials (user/password) in Source Code Management so Jenkins could access files in the repository.

5. Select Apply and save changes

6. Click the “Build Now” button, if you have configured everything correctly, the build will be successful and you will see it under #1



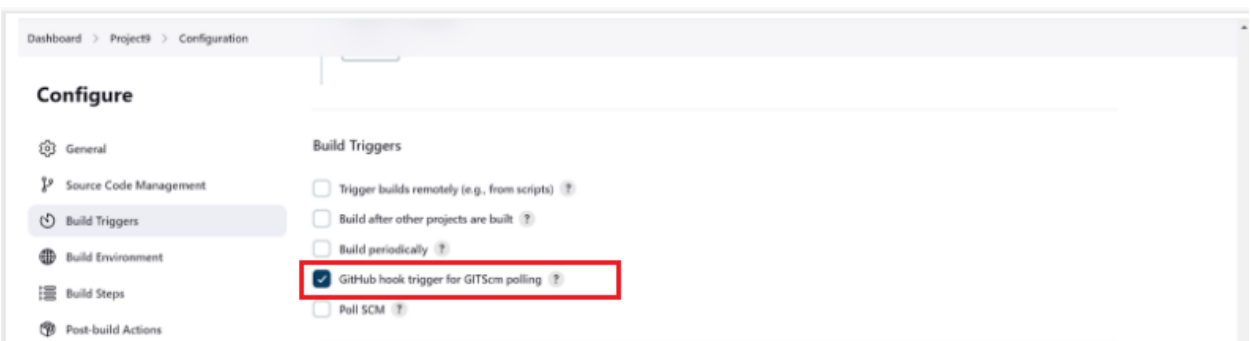
7. Click on the “Console Output” to confirm if the build has run successfully. At the moment, this build does not produce anything and it runs only when we trigger it manually.



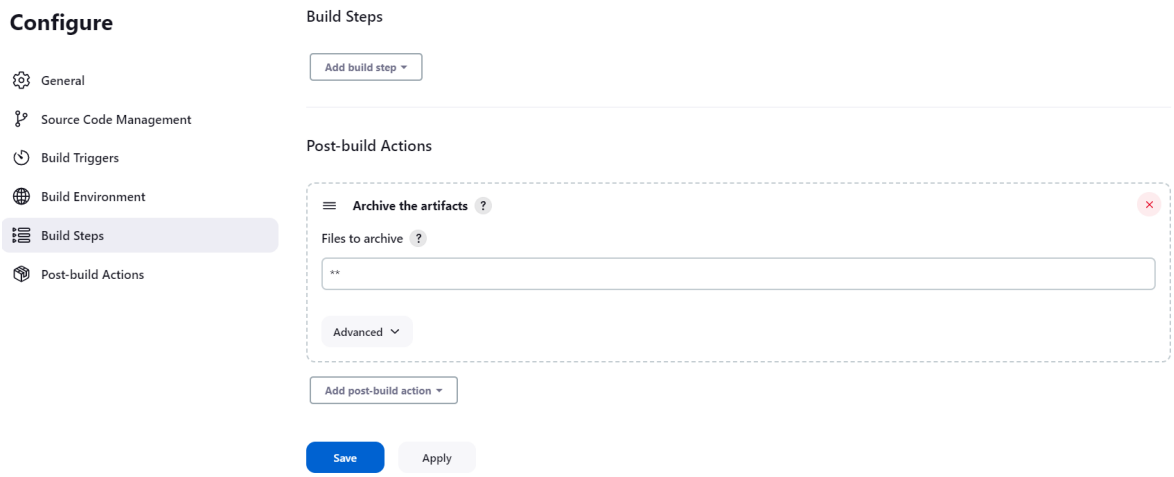
8. The build number should also appear on the server.

```
ubuntu@ip-172-31-40-224:~$ cd /var/lib/jenkins/jobs/
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs$ ls
Project9
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs$ cd Project9
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9$ ls
builds  config.xml  nextBuildNumber
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9$ cd builds
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9/builds$ ls
1  legacyIds  permalinks
```

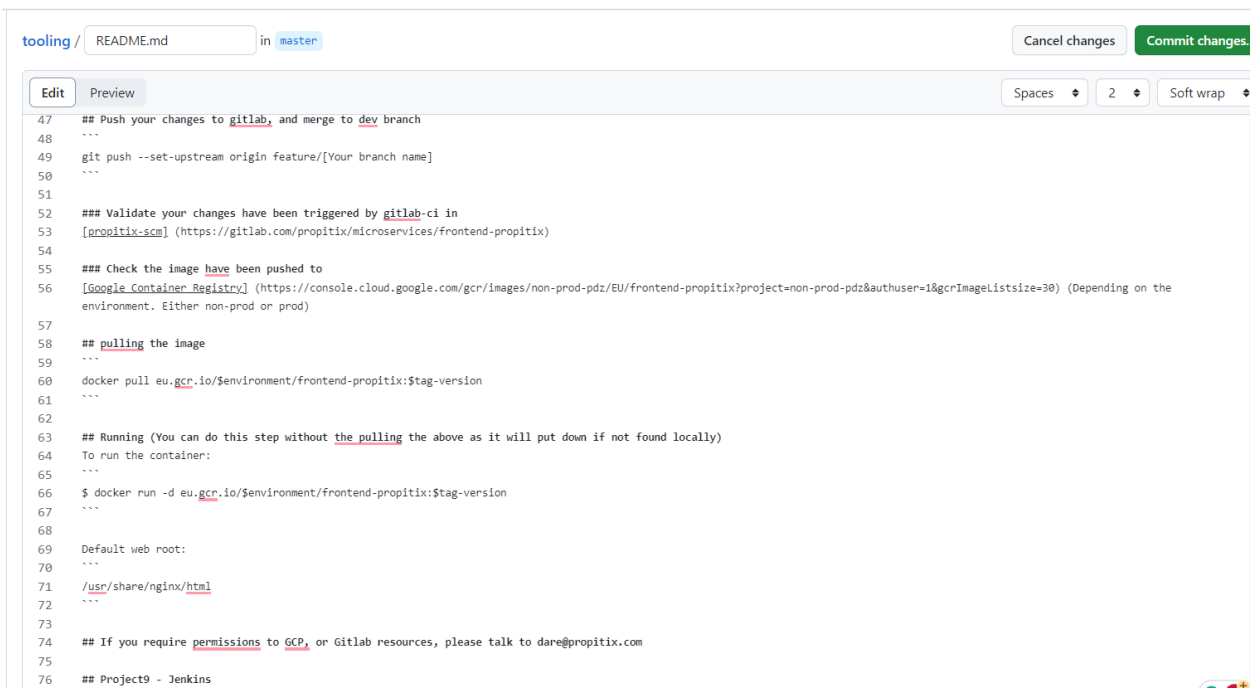
9. To automate this build, click on "Configure", and scroll down to Build triggers to add this configuration.



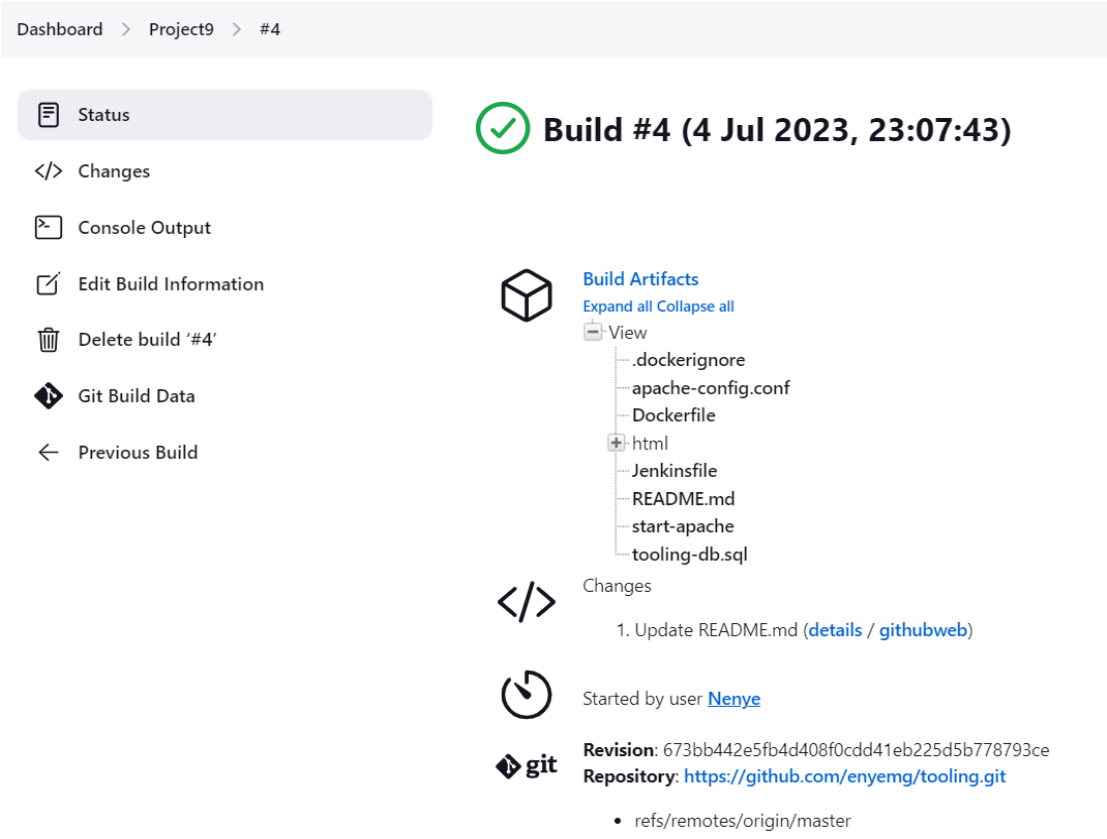
10. Add post-build action to archive all files to archive, type ** in the box, select apply and then save.



11. Now, in your GitHub repository, make some changes in any file (e.g. README.md file) and commit the changes to the master branch.



The new build has been launched automatically (by webhook) and you can see its results – artifacts, saved on Jenkins server. I have now configured an automated Jenkins job that receives files from GitHub by webhook trigger.



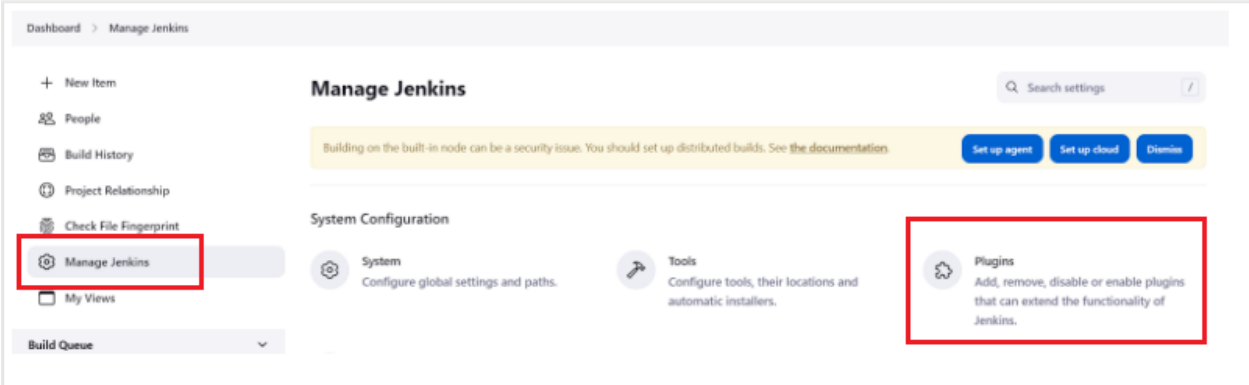
By default, the artifacts are stored on Jenkins server locally on this path /var/lib/jenkins/jobs/tooling_github/builds/<build_number>/archive/

```
ubuntu@ip-172-31-40-224:~$ cd /var/lib/jenkins/jobs/
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs$ ls
Project9
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs$ cd Project9
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9$ ls
builds  config.xml  nextBuildNumber
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9$ cd builds
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9/builds$ ls
1  legacyIds  permalinks
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9/builds$ cd 1
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9/builds/1$ ls
build.xml  changelog.xml  log
ubuntu@ip-172-31-40-224:/var/lib/jenkins/jobs/Project9/builds/1$
```

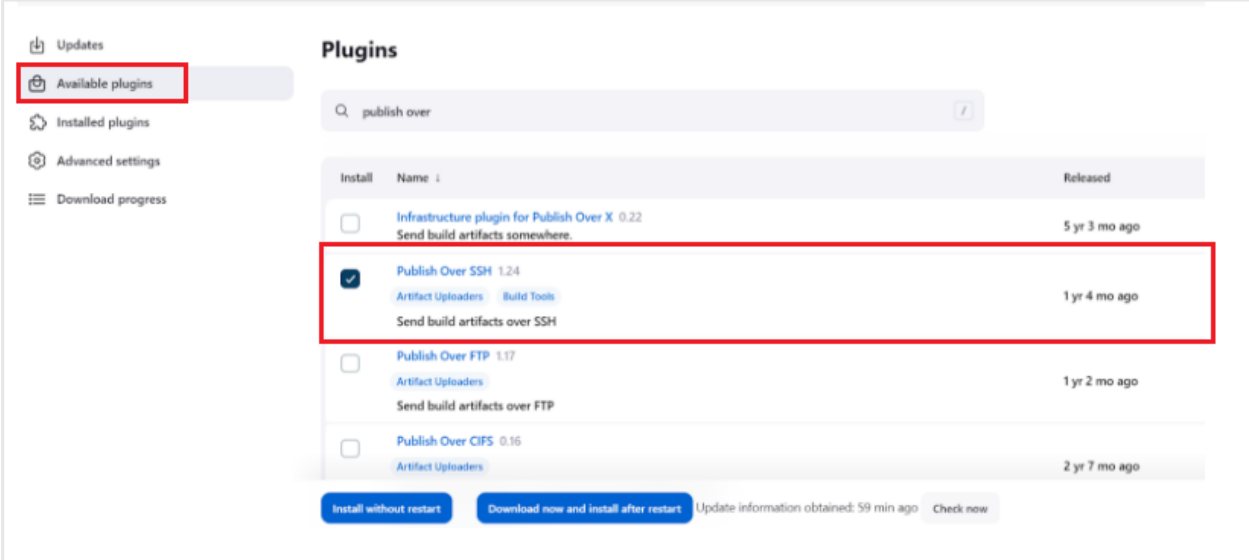
Step 4 – Configure Jenkins to copy files to the NFS server via SSH

Artifacts are saved locally on the Jenkins server, the next step is to copy them to our NFS server to the/mnt/apps directory. Jenkins is a highly extendable application and has 1400+ plugins available. We will need a plugin that is called “Publish Over SSH”.

1. Install the “Publish Over SSH” plugin. Go to the dashboard select “Manage Jenkins” and choose the “Plugins” menu item**.**



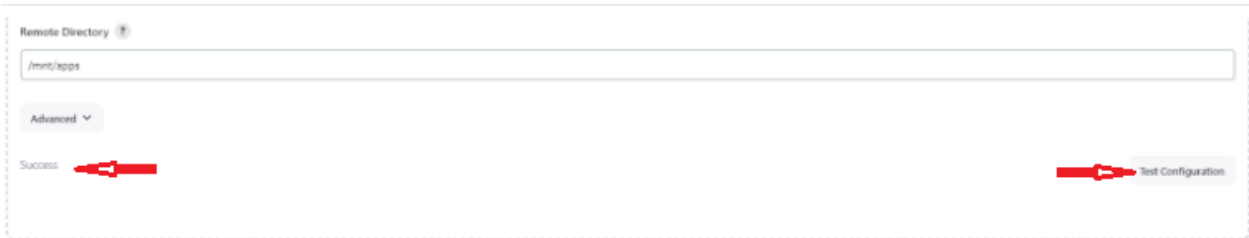
2. Select the “Available Plugins” tab search for the “Publish Over SSH” plugin and select Install without restart



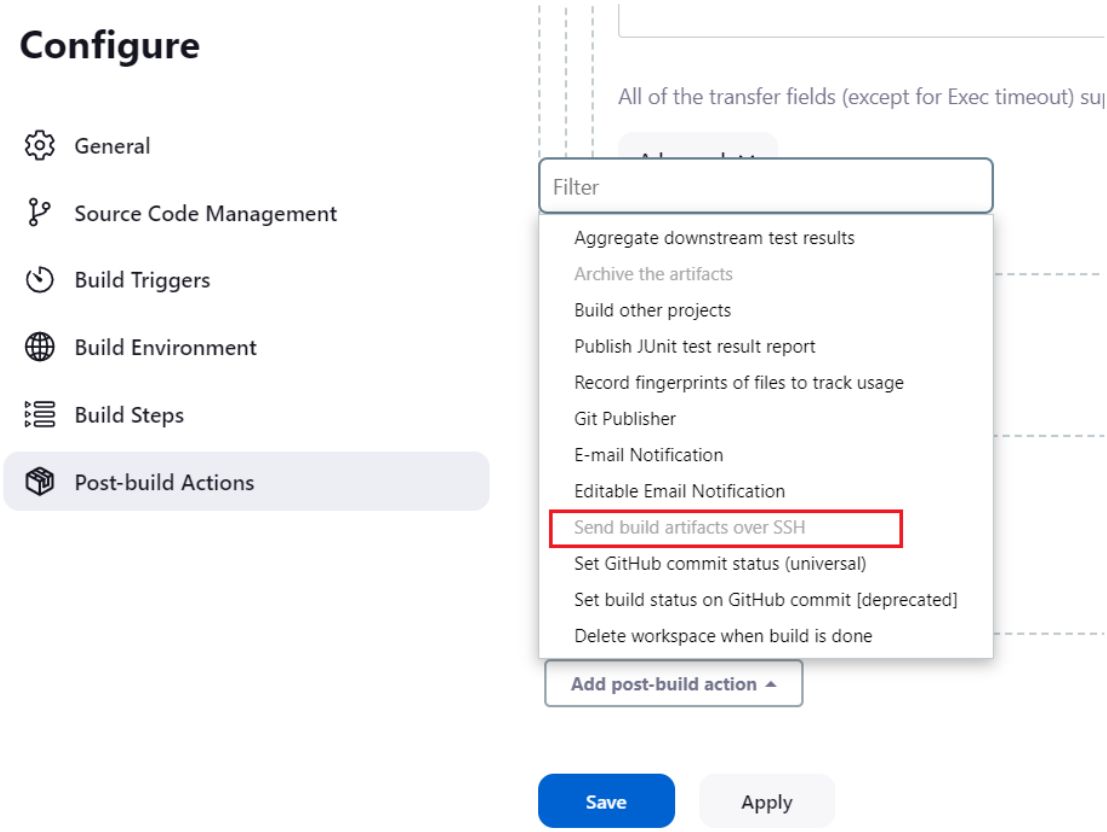
3. Configure the job/project to copy artifacts over to the NFS server. To do this, go to the Dashboard select “Manage Jenkins” and choose the “Configure System” menu item. Scroll down to Publish over the SSH plugin configuration section and configure it to be able to connect to your NFS server:

a. Provide a private key (the content of the .pem file used to connect to the NFS server via SSH)
b. Name
c. Hostname – Private IP address of the NFS server
d. Username – ec2-user (since the NFS server is based on EC2 with RHEL 8)
e. Remote directory – /mnt/apps since our Web Servers use it as a mounting point to retrieve files from the NFS server

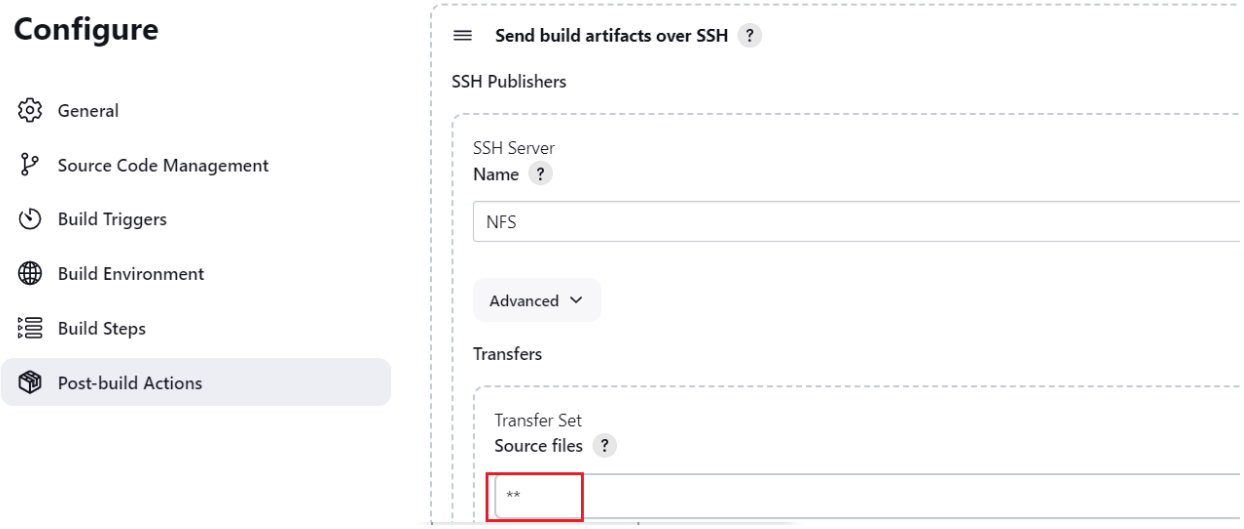
4. Test the configuration and make sure the connection returns Success. TCP port 22 on NFS server must be open to receive SSH connections.



5. Go back to Dashboard and open the Jenkins project configuration page and add another Post-build Action called “Send build artifacts over SSH”



6. Configure it to send all files produced by the build into our previously define remote directory. In this case, we want to copy all files and directories – so use ** and save changes



Edit the README.md file in your GitHub Tooling repository again and the webhook should trigger a new job in the “Console Output” of the job you will find something like the below.

Console Output

```
Started by user Nenye
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Project9
The recommended git tool is: NONE
using credential 152b03a4-ddc3-4a2c-b081-1d97e62771ec
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Project9/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/enyemg/tooling.git # timeout=10
Fetching upstream changes from https://github.com/enyemg/tooling.git
 > git --version # timeout=10
 > git --version # 'git version 2.25.1'
using GIT_ASKPASS to set credentials
 > git fetch --tags --force --progress -- https://github.com/enyemg/tooling.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 673bb442e5fb4d408f0cdd41eb225d5b778793ce (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 673bb442e5fb4d408f0cdd41eb225d5b778793ce # timeout=10
Commit message: "Update README.md"
 > git rev-list --no-walk 5f81660e4e733e4f072ef3007c047477c023fca0 # timeout=10
Archiving artifacts
SSH: Connecting from host [ip-172-31-40-224]
SSH: Connecting with configuration [NFS] ...
SSH: Disconnecting configuration [NFS] ...
SSH: Transferred 25 file(s)
Finished: SUCCESS
```

To make sure that the files in /mnt/apps have been updated – connect via SSH to your NFS server and check the README.md file. If you see the changes you had previously made in your GitHub – the job works as expected.

```
[ec2-user@ip-172-31-43-93 ~]$ cat /mnt/apps/README.md
[![nginx 1.17.2](https://img.shields.io/badge/nginx-1.17.2-brightgreen.svg?&logo=nginx&logoColo
r=white&style=for-the-badge)](https://nginx.org/en/CHANGES) [![php 7.3.8](https://img.shields.i
o/badge/php--fpm-7.3.8-blue.svg?&logo=php&logoColor=white&style=for-the-badge)](https://secure.
php.net/releases/7_3_8.php)


## Introduction
This is a Dockerfile to build a debian based container image running nginx and php-fpm 7.3.x /
7.2.x / 7.1.x / 7.0.x & Composer.

### Versioning
| Docker Tag | GitHub Release | Nginx Version | PHP Version | Debian Version |
|-----|-------|-----|--------|--------|
| latest | master Branch |1.17.2 | 7.3.8 | buster |


## How to use this repository
The build is automatically triggered by a git push to your feature/[branch]

## First clone the repository to your workstation
```
$ git clone https://gitlab.com/propitix/microservices/php-frontend.git
$ cd frontend-propitix
```

Create a feature branch. # Always start with feature/[name of your branch]
```
git branch -b feature/add-css-style-to-about-us-page
```
```
```
```

I have now implemented my first Continuous Integration solution using Jenkins and the architecture should now look like this.