TOOLING WEBSITE DEPLOYMENT AUTOMATION WITH CONTINUOUS INTEGRATION. INTRODUCTION TO JENKINS

TASK

In this project, we will enhance the architecture prepared in Project 8 by adding a Jenkins server, configure a job to automatically deploy source codes changes from Git to NFS server.

1. 1nfrastructure: **AWS**
2. Webserver Linux: **Red hat Enterprise Linux 8**
3. Storage Server: **Red Hat Enterprise Linux + NFS Server**
4. Code Repository: **GitHub**
5. Automation Server: **Jenkins Server (Linux Ubuntu 20.04)**

STEP 1.

CREATE AND CONFIGURE JENKINS SERVER

We will create another AWS EC2 instance on ubuntu server 20.04 LTS, name it "Jenkins".

Connect to a terminal and update.

$ sudo apt update -y

We will install **JDK** (Java development kit) because Jenkins is a java-based application.

$ sudo apt install default-jdk-headless

The command above installs default OpenJDK package without any graphical user interface (headless version) on a Debian-based system.

Install Jenkins

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

This is used to download the Jenkins repository key and add it to the apt keyring on a Debian-based system.

**$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > **

/etc/apt/sources.list.d/jenkins.list'

**$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee **

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

**$ 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

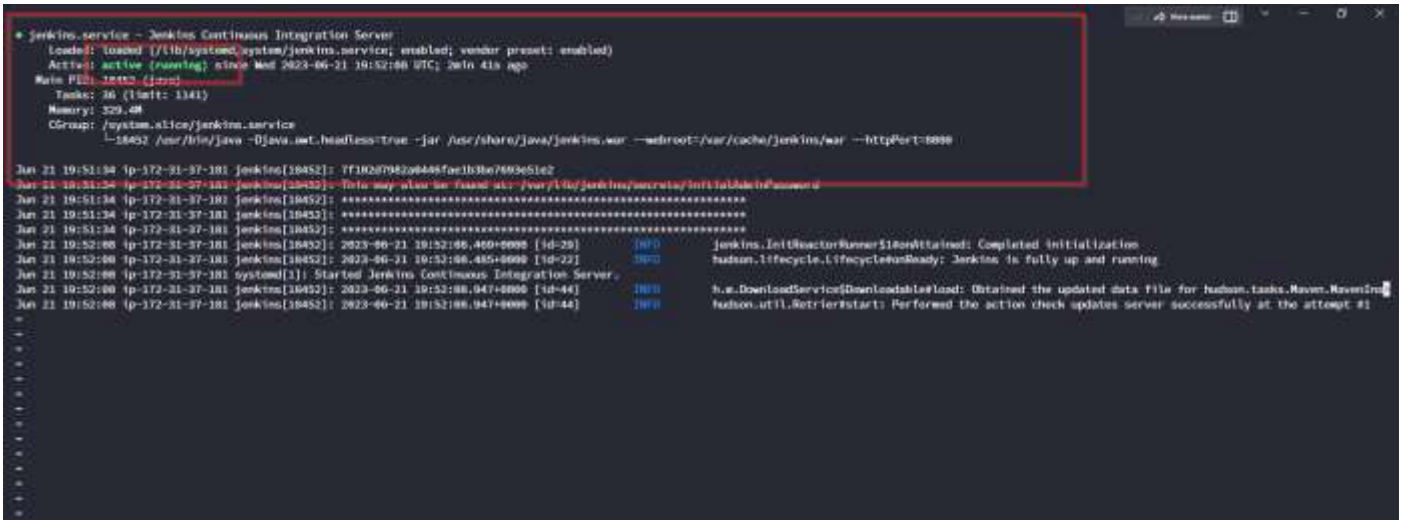
$ sudo apt-get update

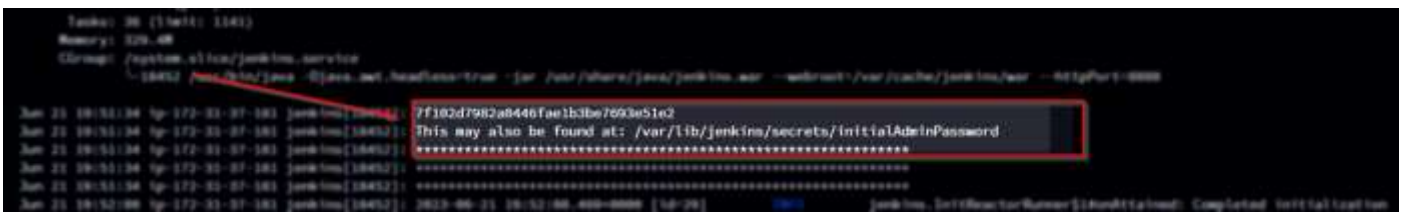
$ sudo apt-get install fontconfig openjdk-11-jre

$ sudo apt-get install Jenkins

check that Jenkins is installed correctly and running.

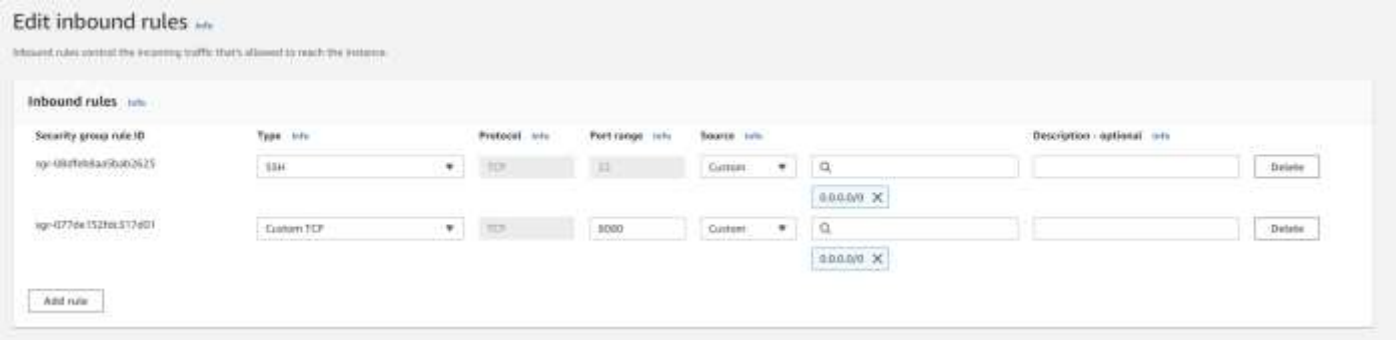
$ sudo systemctl status Jenkins





Notice the Jenkins password highlighted which can also be found in the /var/lib/jenkins/secrets/initialAdminPassword directory

By default, Jenkins server listens on port 808. Hence, we will need to open up port 8080 in our security group in-bound rules.

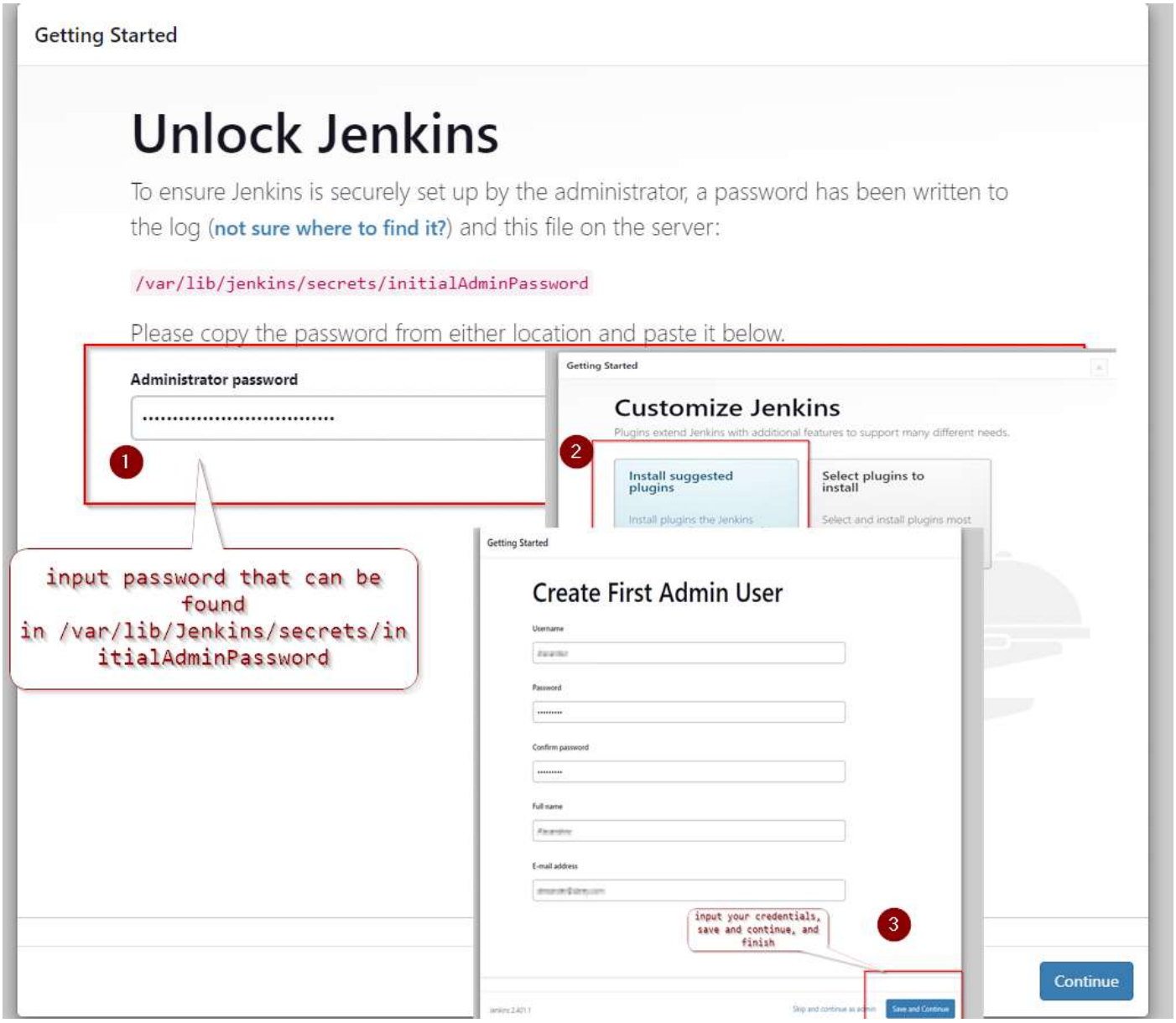


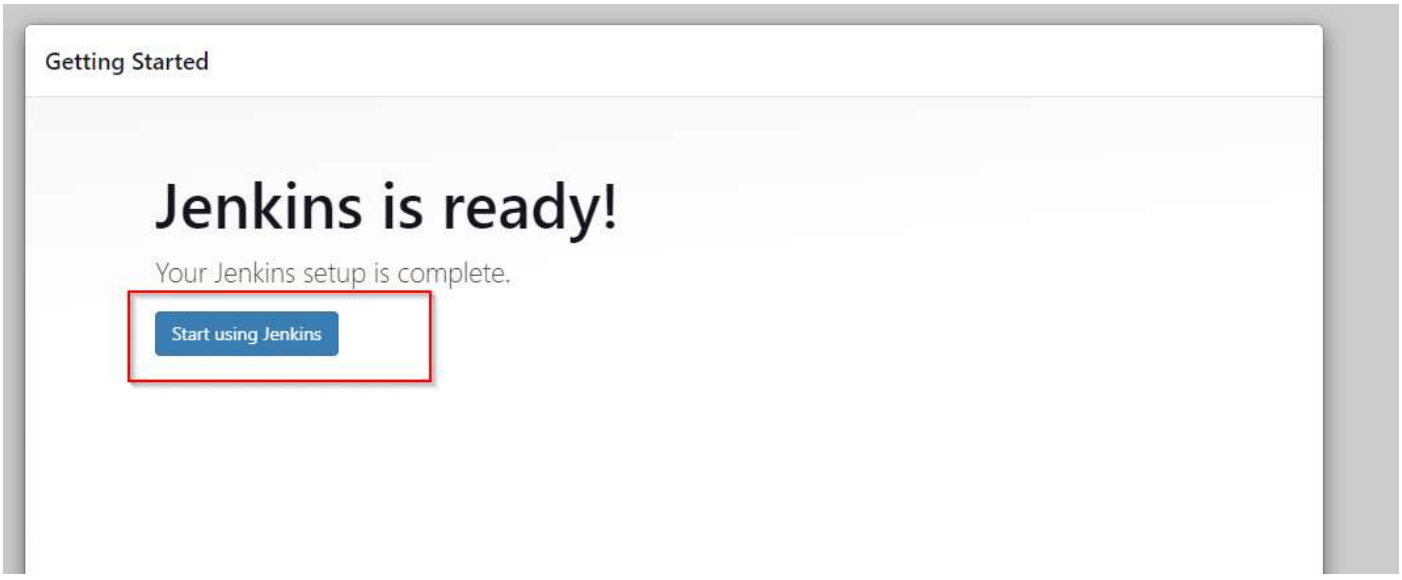
Perform initial Jenkins setup.

From your browser access http://<Jenkins-Server-Public-IP-Address-or-Public-DNS-Name>:8080

$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword to retrieve password.

Input default admin password, then you will be asked which plugins to install – choose suggested plugins.


Getting Started
Unlock Jenkins
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:
/var/lib/jenkins/secrets/initialAdminPassword
Please copy the password from either location and paste it below.
Administrator password
1
input password that can be found in /var/lib/Jenkins/secrets/initialAdminPassword
Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
2
Install suggested plugins
Select plugins to install
Getting Started
Create First Admin User
Username
Password
Confirm password
Full name
E-mail address
input your credentials, save and continue, and finish
3
Continue

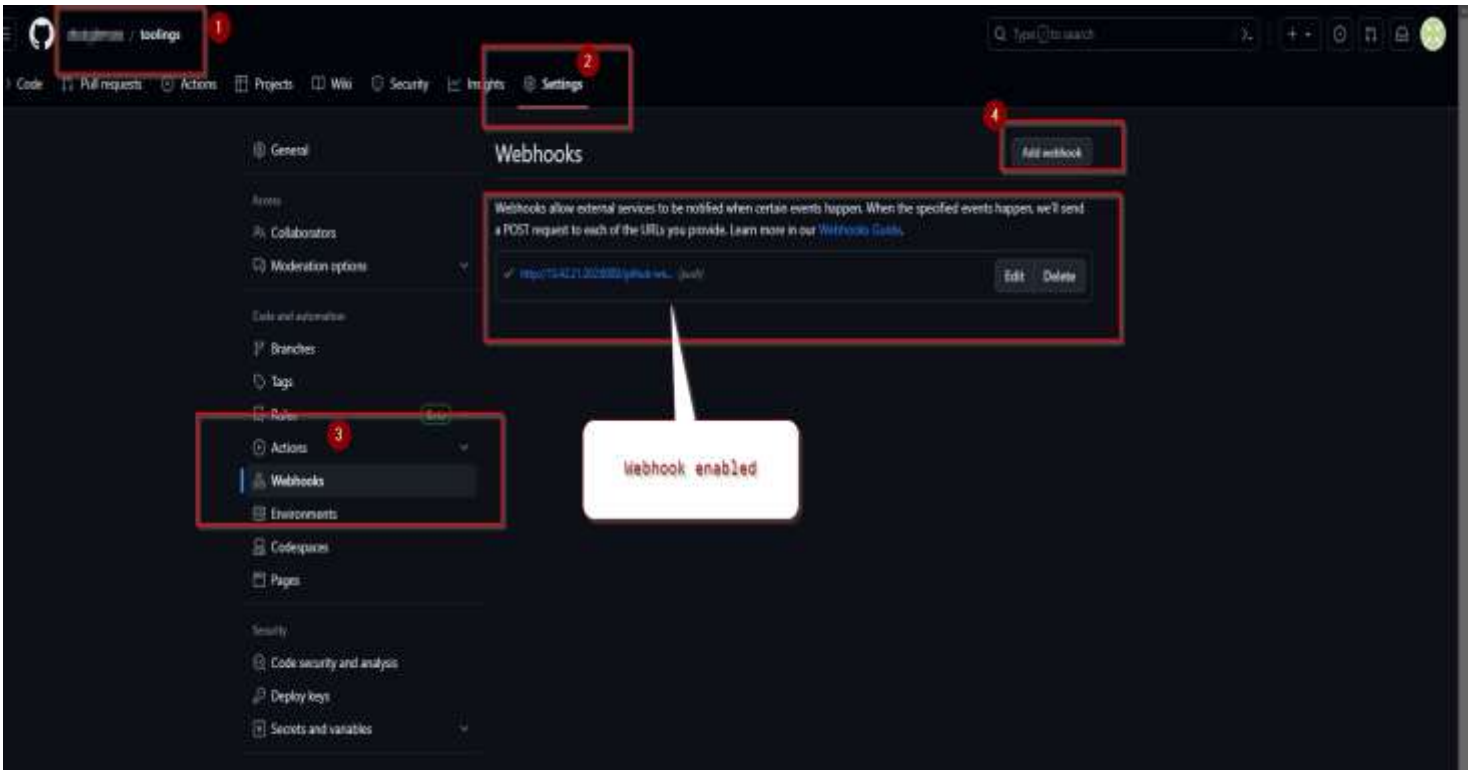

Getting Started
Jenkins is ready!
Your Jenkins setup is complete.
Start using Jenkins

STEP 2.

CONFIGURE JENKINS TO RETRIEVE SOURCE CODES FROM GITHUB USING WEBHOOKS

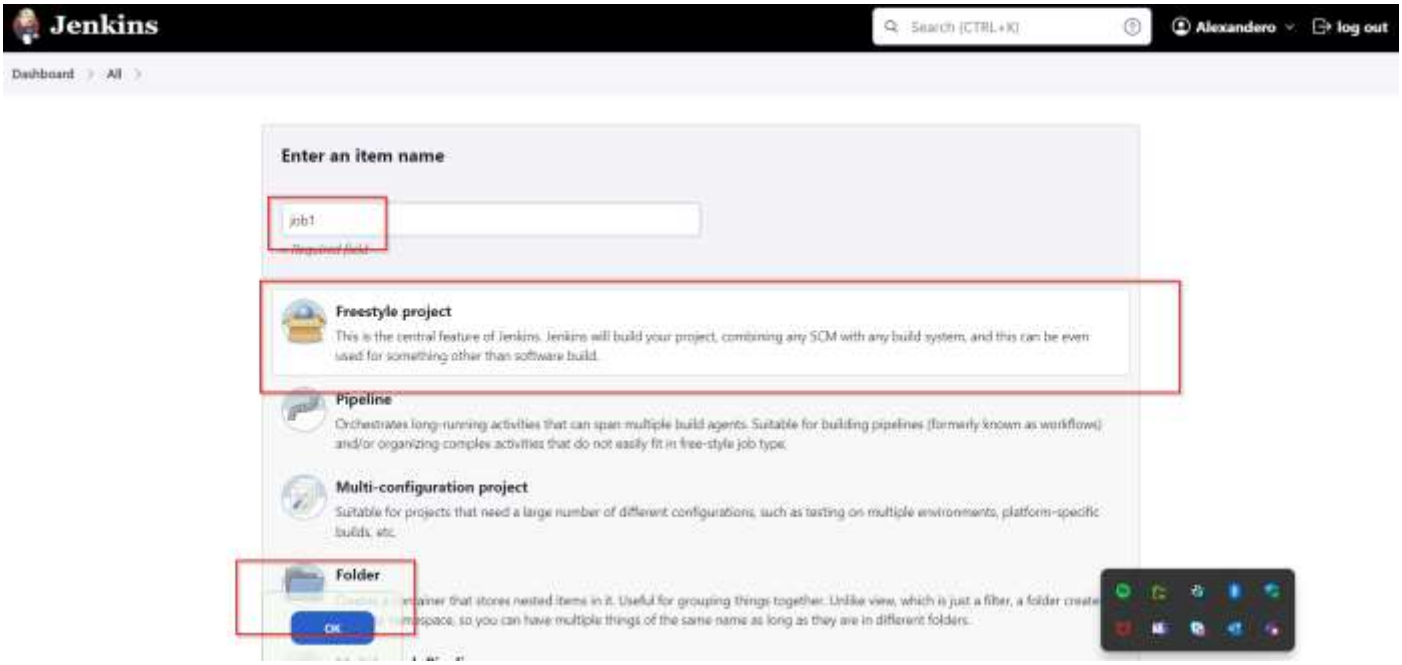
In this part, you will learn how to configure a simple Jenkins job/project. This job will be triggered by GitHub webhooks and will execute a ‘build’ task to retrieve codes from GitHub and store it locally on Jenkins server.

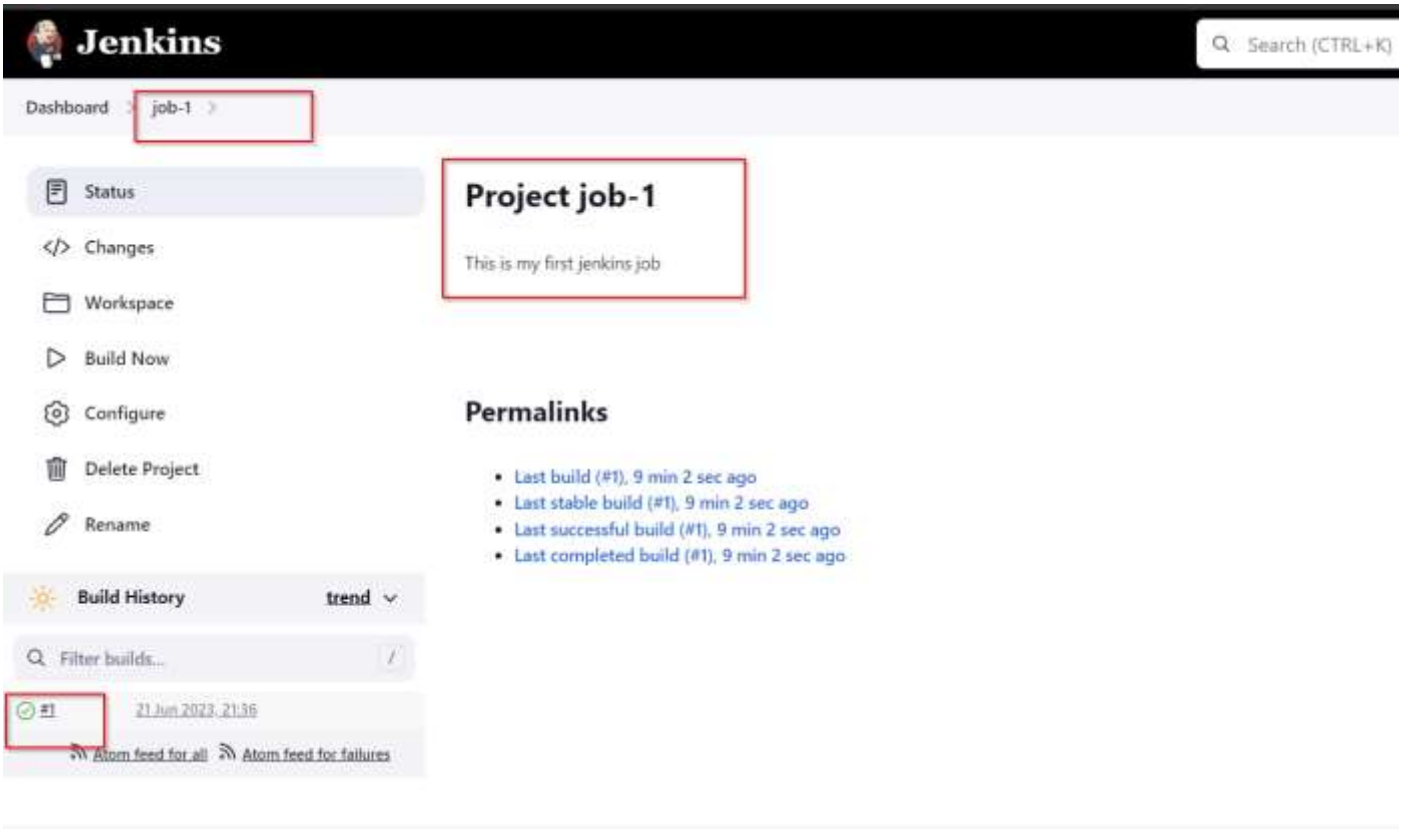
1. Enable webhooks in your GitHub repository settings.



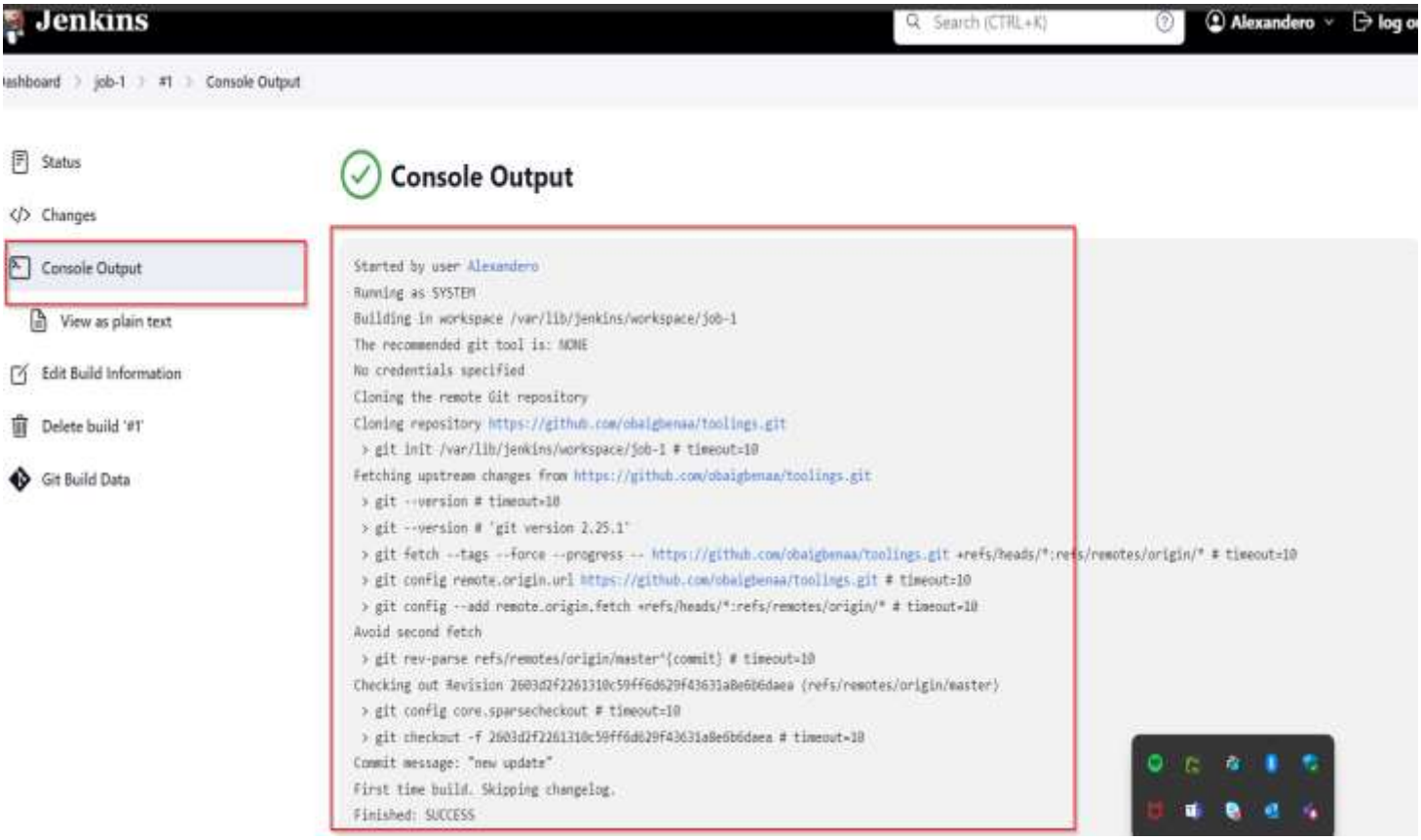
Go to Jenkins web console, click "New Item" and create a "Freestyle project".

Connect your GitHub repository, you will need to provide its URL, you can copy from the repository itself.





Open the build and check in "Console Output" if it has run successfully.



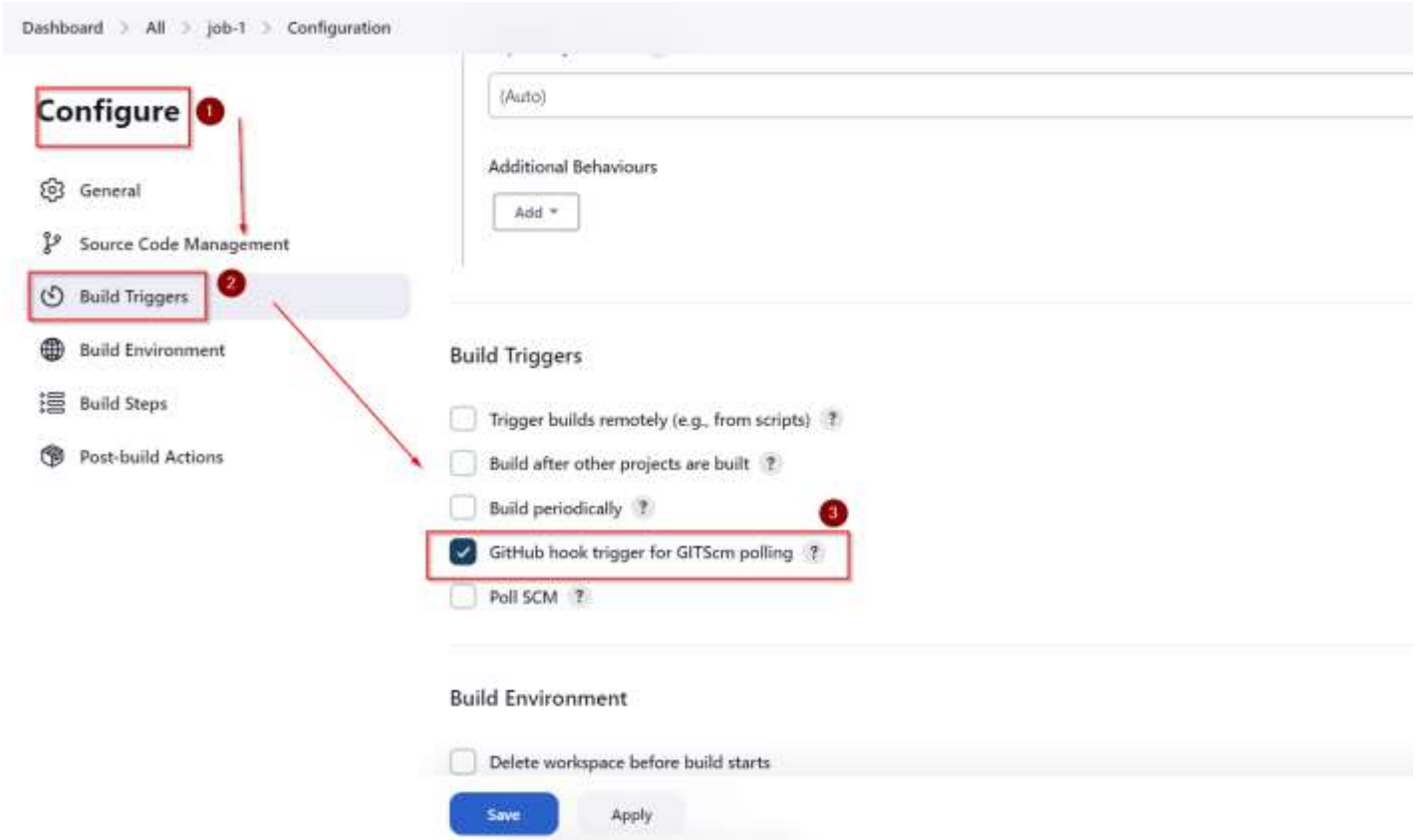
Check for job-1 on the terminal in the directory below

$ cd /var/lib/Jenkins/workspace

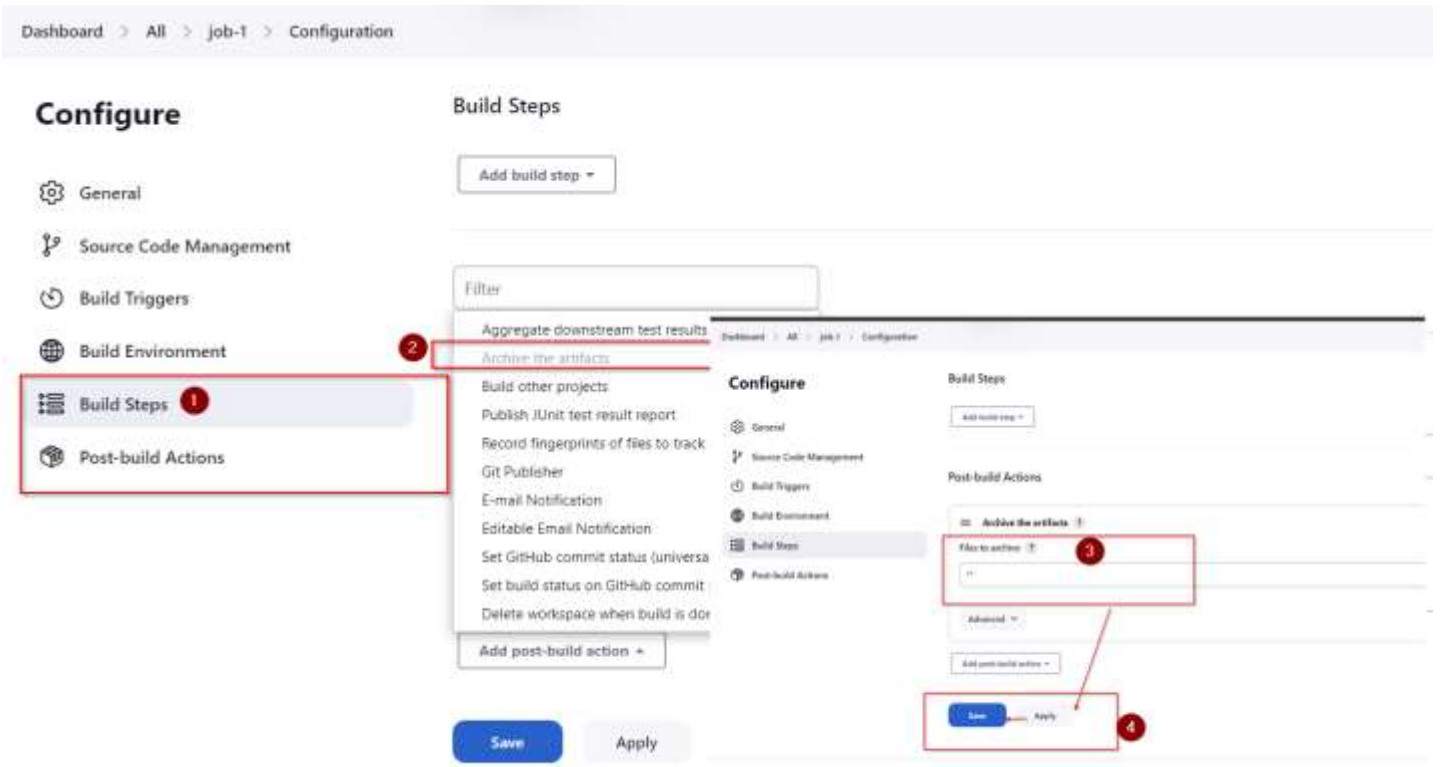
```
ubuntu@ip-172-31-37-181:~$ cd /var/lib/jenkins/workspace/job-1
ubuntu@ip-172-31-37-181:/var/lib/jenkins/workspace/job-1$ ls
Dockerfile  Jenkinsfile  README.md  apache-config.conf  html  start-apache  tooling-db.sql
ubuntu@ip-172-31-37-181:/var/lib/jenkins/workspace/job-1$ cat README.md
this is a readme file adjusted
ubuntu@ip-172-31-37-181:/var/lib/jenkins/workspace/job-1$ cd ..
ubuntu@ip-172-31-37-181:/var/lib/jenkins/workspace$ ls
job-1
ubuntu@ip-172-31-37-181:/var/lib/jenkins/workspace$
```

Click "Configure" your job/project and add these two configurations.

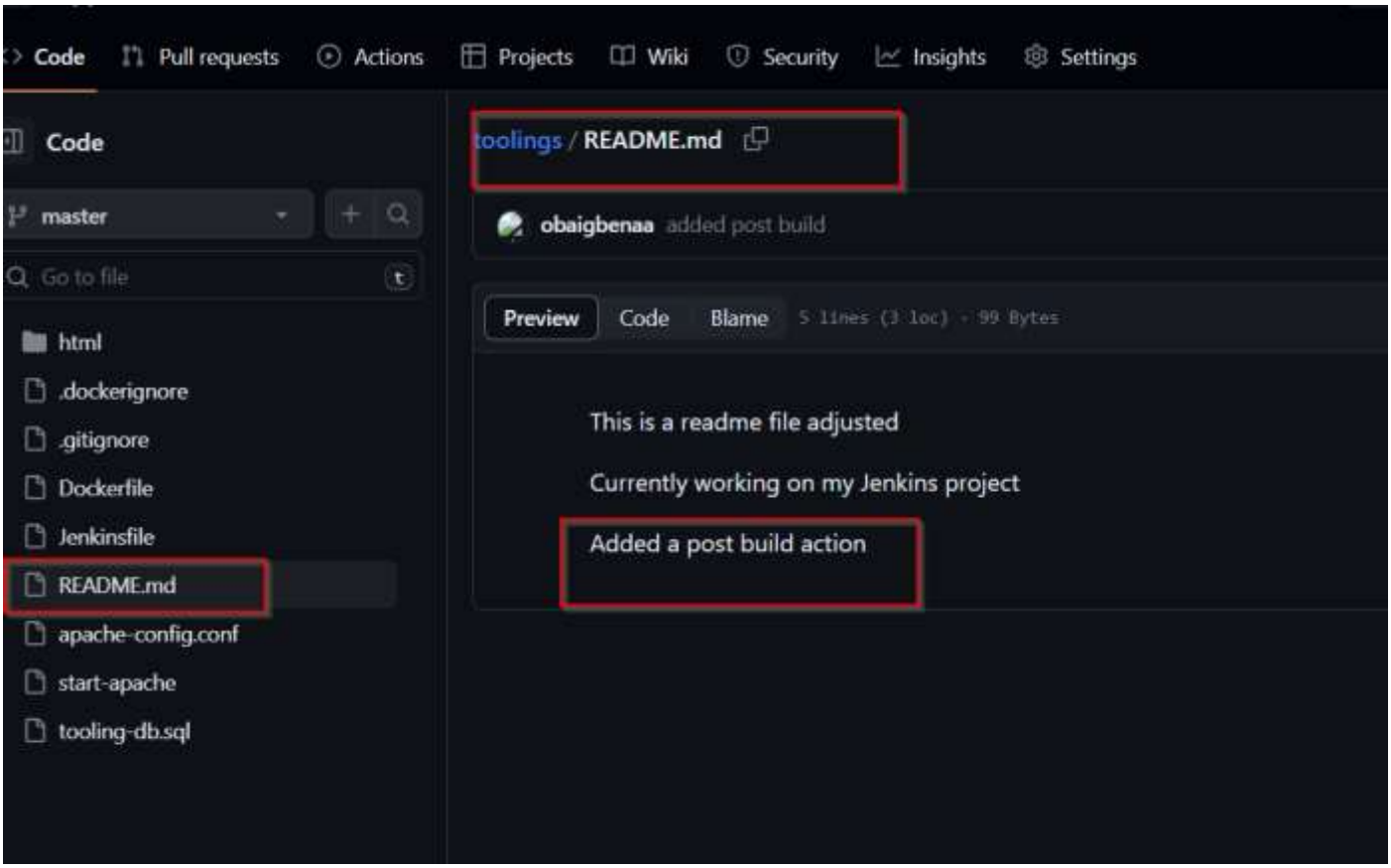
1. Configure triggering the job from GitHub webhook:



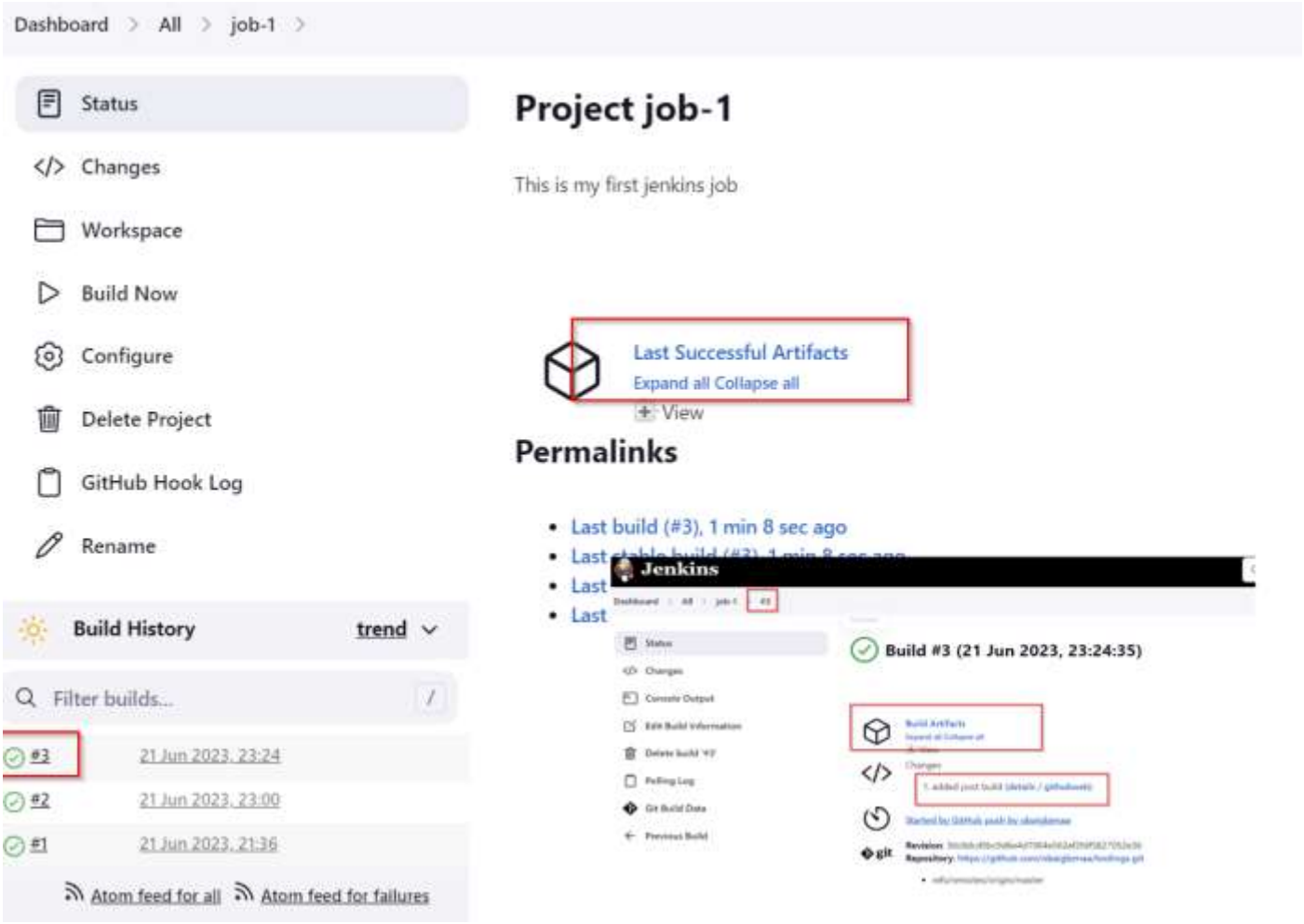
2. Configure "Post-build Actions" to archive all the files – files resulted from a build are called "artifacts.



Now, go ahead and make some changes in any file in your GitHub repository (e.g. README.MD file) and push the changes to the master branch.



You will see that a new build has been launched automatically (by webhook) and you can see its results – artifacts, saved on Jenkins server.



By default, the artifacts are stored on Jenkins server locally.

$ cd /var/lib/jenkins/jobs/job-1/builds/3

```
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/job-1/
builds  config.xml  github-polling.log  nextBuildNumber
ubuntu@ip-172-31-37-181:~$ cd builds
-bash: cd: builds: No such file or directory
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/job-1/builds/
1  2  3  legacyIds  permalinks
ubuntu@ip-172-31-37-181:~$ cd 3
-bash: cd: 3: No such file or directory
ubuntu@ip-172-31-37-181:~$ cat 3
cat: 3: No such file or directory
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/job-1/builds/3/
archive  build.xml  changelog.xml  log  polling.log
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/job-1/builds/3/archive/
Dockerfile  Jenkinsfile  README.md  apache-config.conf  html  start-apache  tooling-db.sql
ubuntu@ip-172-31-37-181:~$ ls /var/lib/jenkins/jobs/job-1/builds/3/archive/README.md
/var/lib/jenkins/jobs/job-1/builds/3/archive/README.md
ubuntu@ip-172-31-37-181:~$ cd /var/lib/jenkins/jobs/job-1/builds/3/
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/job-1/builds/3$ ls
archive  build.xml  changelog.xml  log  polling.log
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/job-1/builds/3$ pwd
/var/lib/jenkins/jobs/job-1/builds/3
ubuntu@ip-172-31-37-181:/var/lib/jenkins/jobs/job-1/builds/3$
```

Step 3.

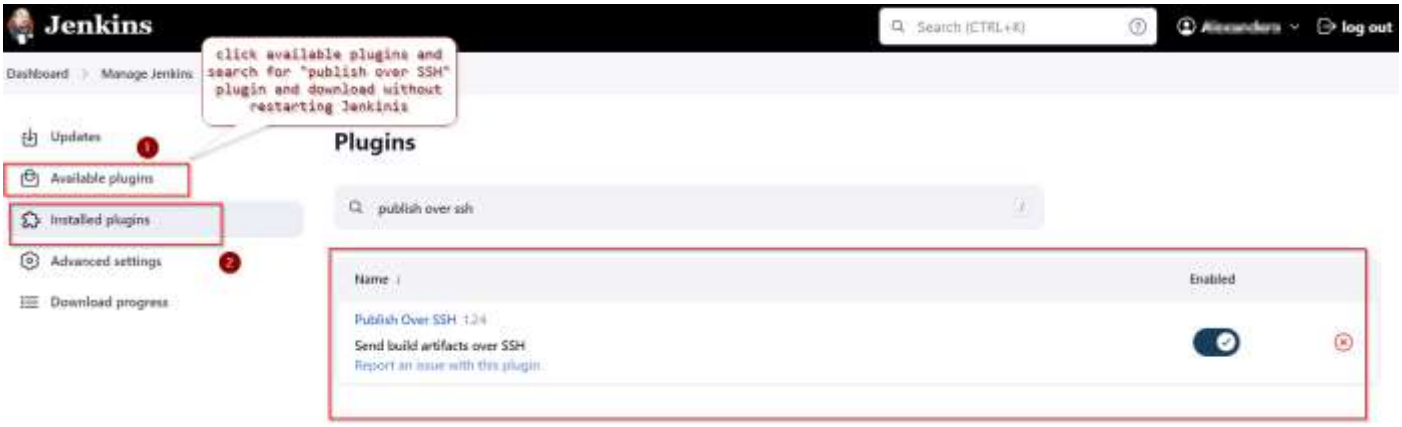
CONFIGURE JENKINS TO COPY FILES TO NFS SERVER VIA SSH

Now we have our artifacts saved locally on Jenkins server, the next step is to copy them to our NFS server to /mnt/apps directory

1. Install "Publish Over SSH" plugin.

On main dashboard select "Manage Jenkins" and choose "Manage Plugins" menu item.

On "Available" tab search for "Publish Over SSH" plugin and install it



2. Configure the job/project to copy artifacts over to NFS server.

On main dashboard select "Manage Jenkins" and choose "System" menu item.

Scroll down to Publish over SSH plugin configuration section and configure it to be able to connect to your NFS server:

Provide a private key (content of **.pem** file that you use to connect to NFS server via SSH)

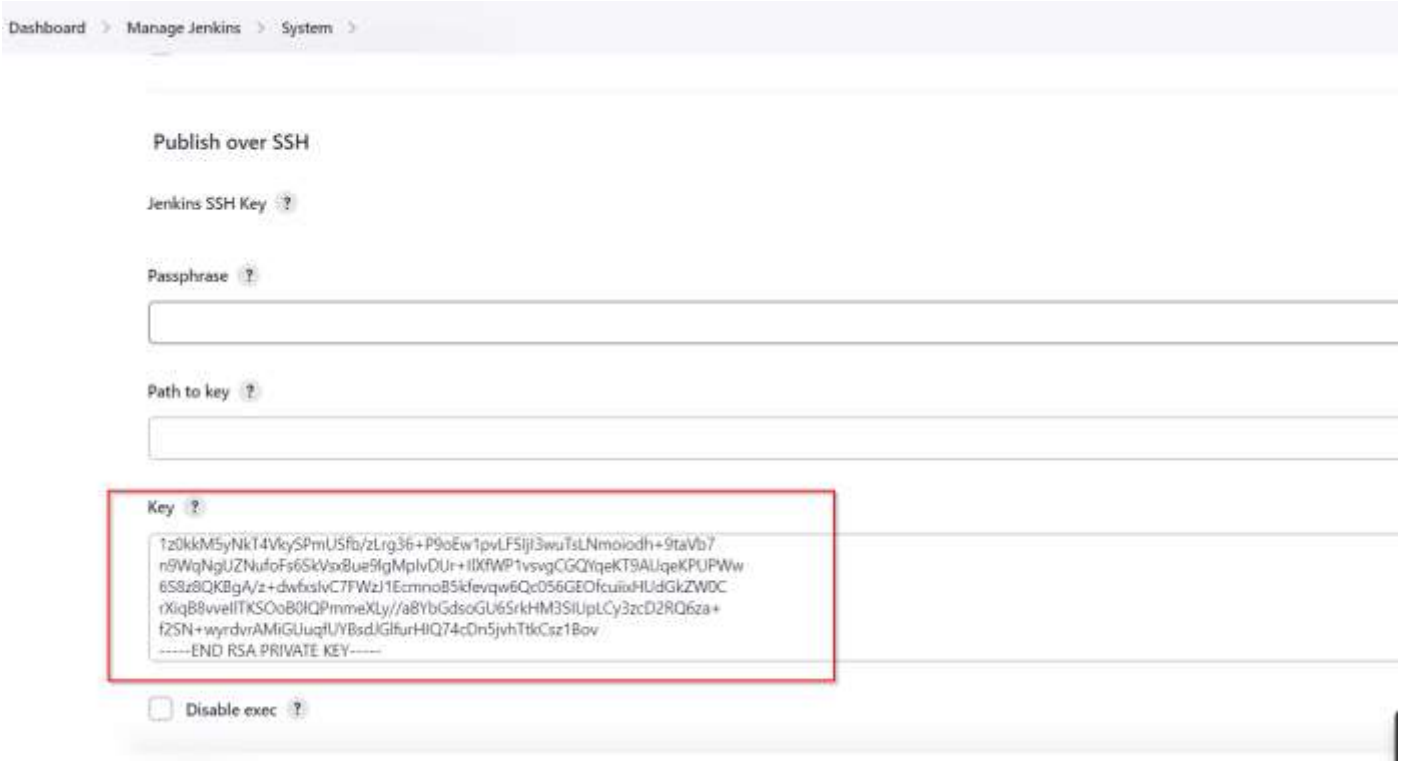
Arbitrary name

Hostname – Here, input the private IP address of your NFS server

Username – ec2-user (since NFS server is based on EC2 with RHEL 8)

Remote directory – /mnt/apps since our Web Servers use it as a mounting point to retrieve files from the NFS server.

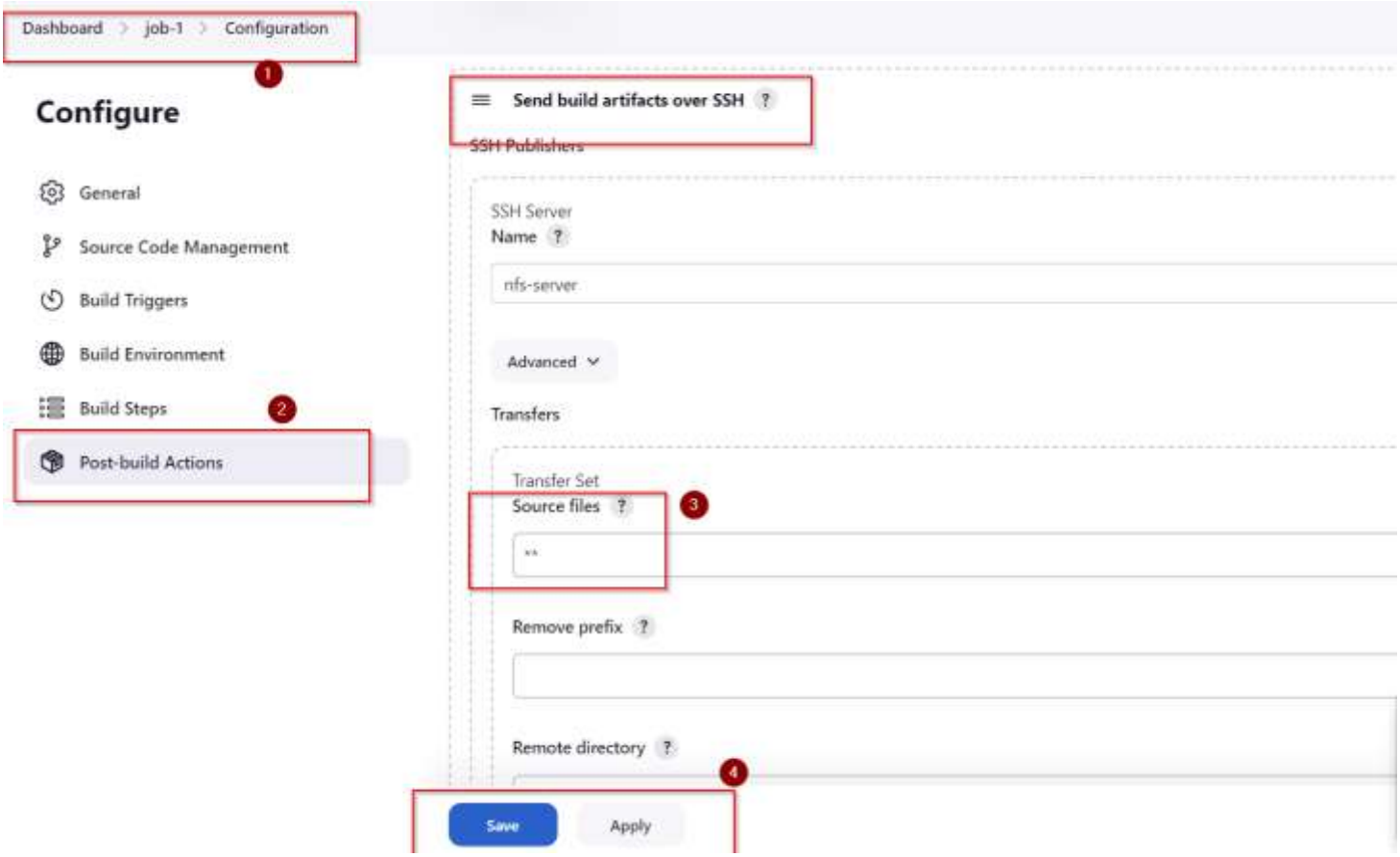
Test the configuration and make sure the connection returns Success. Remember, that TCP port 22 on NFS server must be open to receive SSH connections.



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

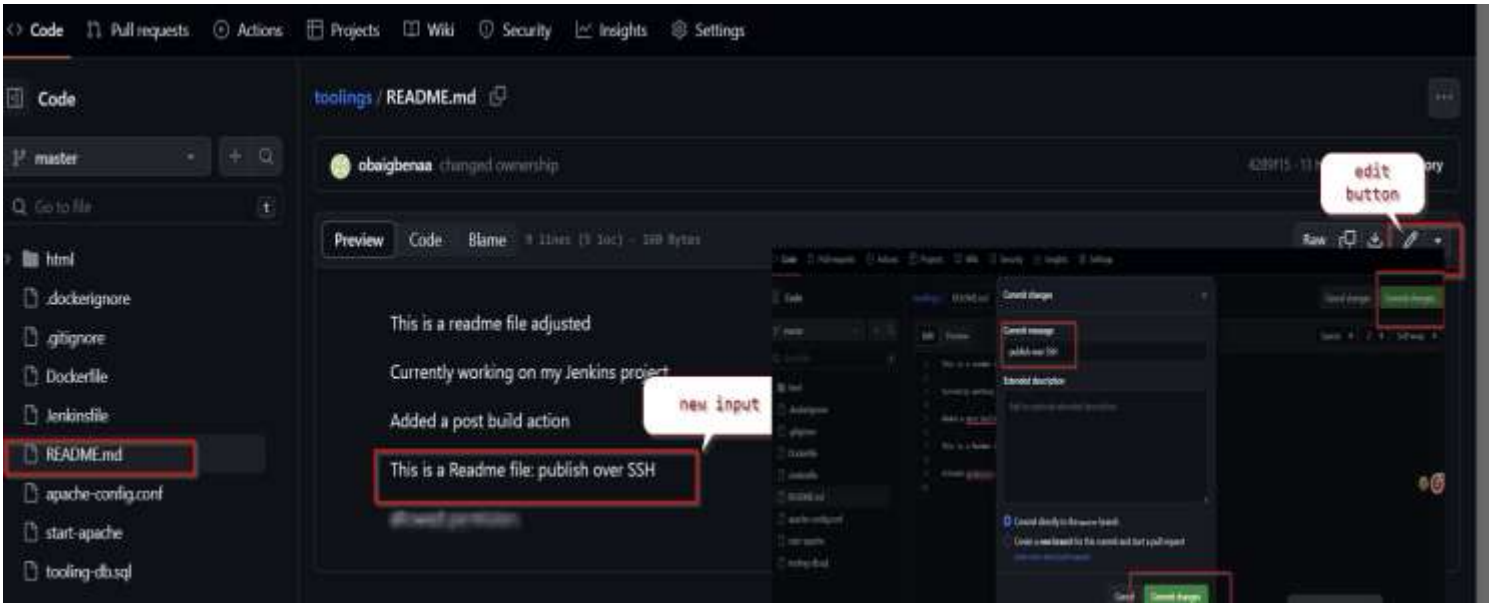
Save the configuration, open your Jenkins job/project configuration page and add another one "Post-build Action".

Configure it to send all files produced by the build into our previously defined remote directory. In our case we want to copy all files and directories – so we use ** to send all files. Save this configuration and go ahead.



Go to your github repository, edit the Readme.md file for job-1.

Webhook will trigger a new job and in the "Console Output”.



Notice the error message when webhook tries to trigger a build in the console output, this is because the username inputed during the SSH server configuration was “ec2-user”. But the /mnt/apps directory is owned by “root” by default.

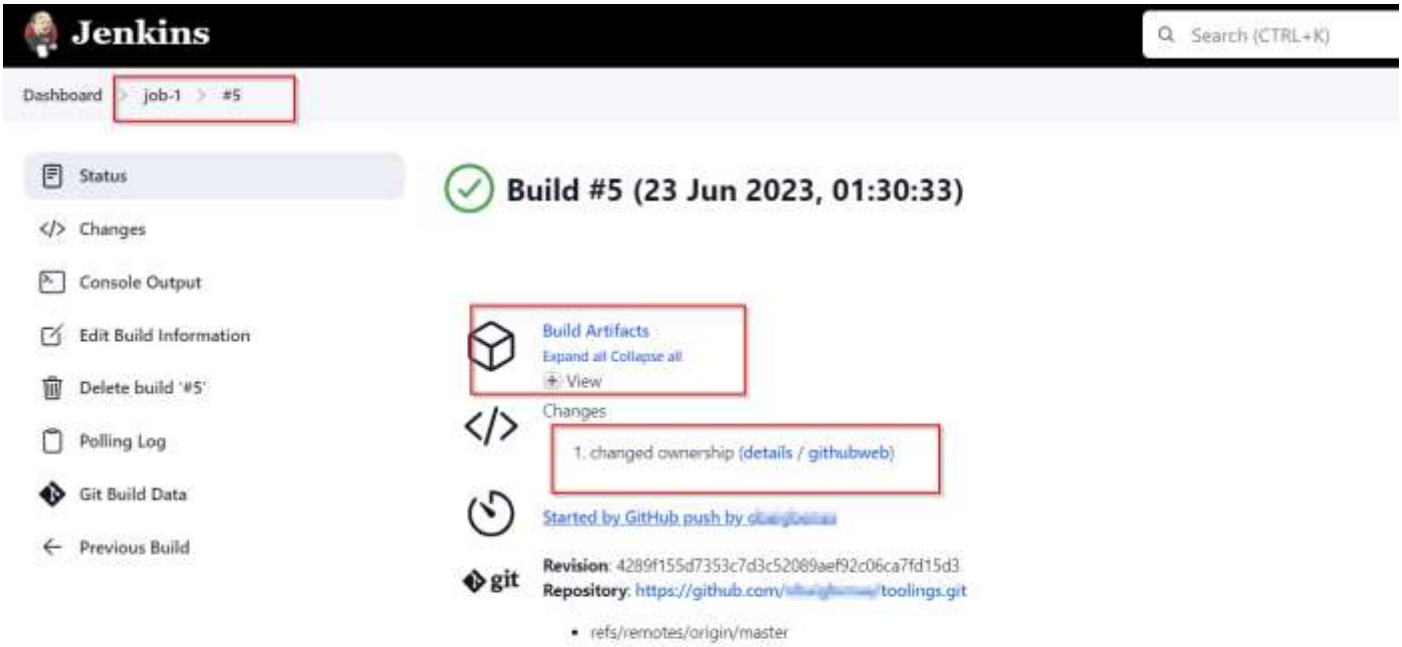
To change ownership to ec2-user for ‘user’ and ‘group’.

Use the command:

$ sudo chown -R ec2-user:ec2-user /mnt/apps/

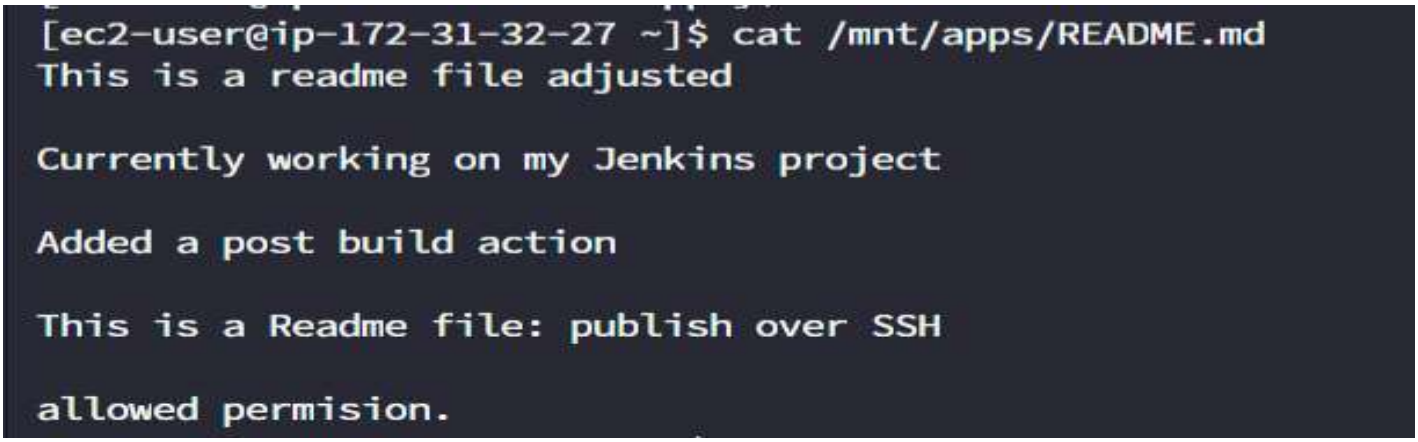
```
[ec2-user@nfs-server ~]$ ls -ls /mnt/apps/
total 4
4 drwxr-xr-x. 3 root root 4096 Jun 21 12:06 html
[ec2-user@nfs-server ~]$ sudo chown -R ec2-user:ec2-user /mnt/apps/
[ec2-user@nfs-server ~]$ ls
[ec2-user@nfs-server ~]$ ls
[ec2-user@nfs-server ~]$ ls -latr /mnt/apps/
total 36
drwxr-xr-x. 5 root     root       41 Jun 19 23:51 ..
drwxr-xr-x. 3 ec2-user ec2-user 4096 Jun 21 12:06 html
-rw-rw-r--. 1 ec2-user ec2-user   47 Jun 23 01:30 .dockerignore
-rw-rw-r--. 1 ec2-user ec2-user  313 Jun 23 01:30 Dockerfile
-rw-rw-r--. 1 ec2-user ec2-user 4202 Jun 23 01:30 Jenkinsfile
-rw-rw-r--. 1 ec2-user ec2-user  160 Jun 23 01:30 README.md
-rw-rw-r--. 1 ec2-user ec2-user  332 Jun 23 01:30 apache-config.conf
-rw-rw-r--. 1 ec2-user ec2-user  163 Jun 23 01:30 start-apache
drwxr-xr-x. 3 ec2-user ec2-user  161 Jun 23 01:30 .
-rw-rw-r--. 1 ec2-user ec2-user 1674 Jun 23 01:30 tooling-db.sql
[ec2-user@nfs-server ~]$ cd /mnt/apps/
[ec2-user@nfs-server apps]$ ls
apache-config.conf  Dockerfile  html  Jenkinsfile  README.md  start-apache  tooling-db.sql
```

Now we will trigger the last build on github and the result should look like below.



To make sure the files in /mnt/apps have been updated – connect via SSH to your NFS server and check README.MD file

$ cat /mnt/apps/README.md



If you see the changes you previously made in your GitHub – the job works as expected!