**Build a Docker Jenkins Pipeline to Implement CI/CD Workflow**

This document has been written to explain how to create “the continuous integration and delivery by building a Docker image and pushed it in docker hub using Jenkins Pipeline”. A CI CD Pipeline serves as a way of automating software application build, tests, and deployments, which is backbone of any organization with a DevOps culture. CI CD Pipelines with Docker are best for your organization to improve code quality and deliver software releases quickly without any human errors and boosts your business greatly.

* **Objective**:  
  Explain step-by-step walkthrough on how CI/CD pipeline can be build using Jenkins pipeline.
* **Solution build will provide below capabilities**:
  + Availability of the application and its versions in the GitHub.
    - Track their version every time a code is committed to the repository.
  + Create a Docker Jenkins Pipeline that will create a Docker image from the Dockerfile and host it on Docker Hub.
  + It should also pull the Docker image and run it as a Docker container.
  + Build the Docker Jenkins Pipeline to demonstrate the continuous integration and continuous delivery workflow.
* **Project goal**:
  + is to deliver the software product will be frequently to the production with high-end quality.
  + Have track on the build status of Jenkins for every commit of the project
* **Tools which have been used:**
  + Docker: To build the application from a Dockerfile and push it to Docker Hub.
  + Docker Hub: To store the Docker image.
  + Git: To connect and push files from the local system to GitHub
  + Docker, Docker Hub, GitHub, Git, Linux (Ubuntu), Jenkins the GitHub.
  + Linux (Ubuntu): As a base operating system to start and execute the project.
  + Jenkins: To automate the deployment process during continuous integration.
  + Eclipse or any other IDE for write and debug app.

* **Project Expected Result:**
  + Jenkins pipeline with CICD pipeline as shown below, demonstrating the java simple application build and deployment process automated with git poll scm, Docker and Jenkins with 'Pipeline as a Code' approach.
* **Project Documentation**:  
  In this document, we are going to see the following topics in detail:  
   1. Prepare Your System (Installation of pre-requisites/tools).  
   2. Application and their docker build creation and testing on local system  
   3. Configurate Continuous Integration using git as Version Control System or Source Code Management System   
   4. Configure Continuous Deployment and execute a Jenkins pipeline job and execute dockerfile respectively through a shall script  
   5. Use a Jenkins plugin "CloudBees Docker Build and Publish" for Continuous Deployment in docker hub.  
   6. Automate the process using Jenkins Build Trigger  
  + **Prepare Your System (Installation of pre-requisites/tools)** 
    - Create your Virtual Machine with any cloud services or pick the operating system that suits you best. Generally, Ubuntu is preferred more, and we will be using the same. Operating system must be updated and have installed with below pre-requisites/tools and check they have installed successfully or not.   
      * Install & check java (1.8)
        + For create and run your java application on machine, you must have Java i.e JDK installed on your machine. For check java is installed or not run below command

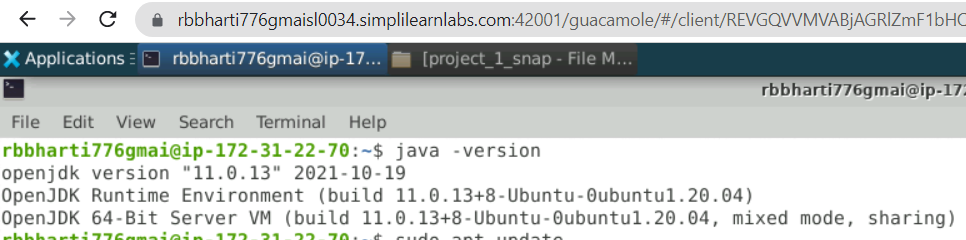
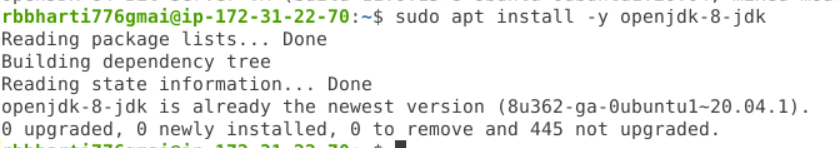
java - version   


Figure 1.check\_java\_installed.

* + - * + For install java in you can run below command



$ *sudo apt update*

*$ sudo apt install -y openjdk-8-jdk*

$ *java -version [Verify java]*

Figure 2. java\_8\_installations  
  
 A black screen with white text

Description automatically generated with low confidence

Figure 3.java\_8\_installed.

* + - * Install & check Jenkins (whether Jenkins running on port 8080)
        + Jenkins is important tool which is required to create pipeline. First, you can check Jenkins is running or not on your machine, if not use below command.

Add the Jenkins key:  
$ *wget -q -O – https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add –*

Add the jenkins pkg repo to package manager:  
Edit the file:

$ *sudo vi /etc/apt/sources.list*

[OR]

$ *sudo nano /etc/apt/sources.list*

Add this line at the end of the file and save it:

*deb* [*https://pkg.jenkins.io/debian-stable binary/*](https://pkg.jenkins.io/debian-stable%20binary/)

Execute:

$ *sudo apt-get update*

Install Jenkins:

$ *sudo apt-get install -y jenkins*

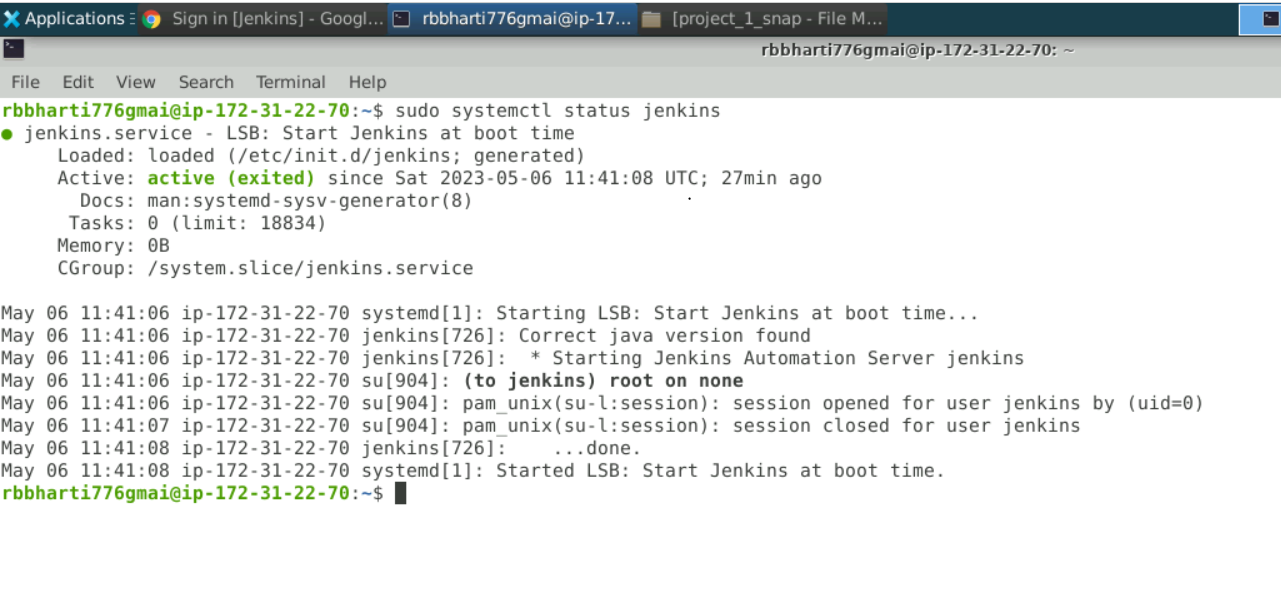
check Jenkins installed on machine through below command.  
 sudo systemctl restart Jenkins   
 

Figure 4.check\_jenkins\_installed\_or\_not

To get the initial admin password to unlock Jenkins:

  
  
 Figure 5.Gennerate\_jenkins\_unlock\_code

To get the initial admin password to unlock Jenkins:

$ *sudo cat /var/lib/jenkins/secrets/initialAdminPassword*  
  
Install suggested plugins.

Create your first admin user.  
 This credential you need to login your Jenkins console.

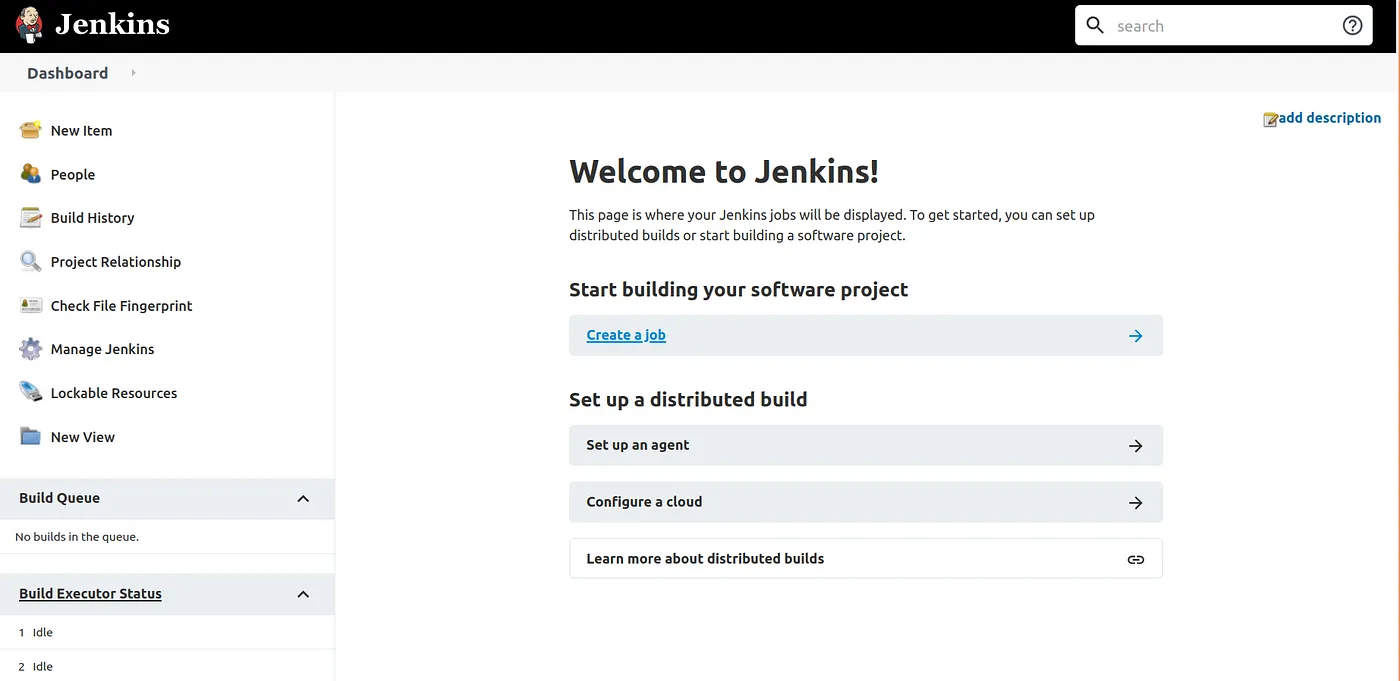
On go to browser and crawl to localhost:8080, Jenkins will ask you to unlock himself. You have paste generated code in given filed. 

Figure 6. Unlock\_jenkins\_console

Now, on click on submit button, it was to create new user and for active this user you need to take restart of jenkins setup. A screenshot of a login page

Description automatically generated with low confidence

Figure 7. Create\_new\_jenkins\_userforLogin

For restart Jenkins, follow below command.  
   
  
  
  
  
  
  
  
  
  
  


Manually restart the jenkins server:

$ *sudo systemctl restart jenkins*

[OR]

$ *sudo service jenkins restart*

Now go and run below url in broswer

Browse: ***http://localhost:8080/restart***

Figure 8.Jenkins\_landing\_page

* + - * Git installation and configuration
        + **Git installation**

*$ sudo apt-get update*  
*$ sudo apt-get install -y git*  
  
[Verify git installation]  
*git -- version*

A screenshot of a computer

Description automatically generated

Figure 9.check\_git\_version

* + - * + Git configuration

$ git config --global user.email "YOUR\_EMAIL\_ID"

$ git config --global user.name "YOUR\_USER\_NAME"

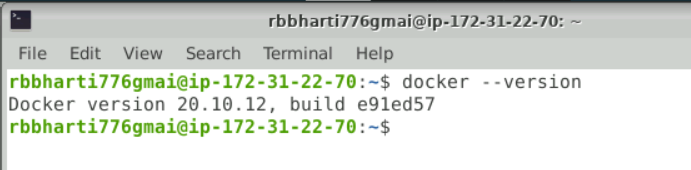
$ git config --list

* + - * + Git repo creation

Create repo from git console by login it with your credentials  
 **https://github.com/**  
  
Now, Clone your repository in your Linux machine  
$ git clone https://github.com/robert776/Simplilearn\_robharti\_A1Project1.git

$ cd Simplilearn\_robharti\_A1Project1

* + - * Docker installation and configuration
        + Docker installation



$ sudo apt-get update

$ sudo apt-get install -y docker.io

$ docker --version

Figure 10.check\_docker\_installed\_or\_not

* + - * + Docker permission required for perform.

ONLY If you get Permission Denied error:

$ sudo docker --version

$ sudo usermod -aG docker <your\_linux\_user\_name>

[Relogin]

$ docker --version

And relog in

* + **Application and his docker build creation and testing on local System.**
    - Sample Application code in java  
      * For write and execute java application you have java installed in your machine which you can check from command “*java-version”*
      * Now, you can create the app for your requirement inside this local system and check output by execute your code on local system.

$ nano hello\_lmsapp.java  
  
class hello\_lmsapp {

public static void main(String[] args) {

System.out.println("Hi Simplilearn lms - PG devops program");

System.out.println("This is a java app created using Dockerfile");

System.out.println("Assessment project ");

System.out.println("Author :- Robert Bharti,rbbharti776@gmail.com");

} }

* + - * For execute, your java app i.e hello\_lmsapp.java locally, you have to run below command   
         *javac hello\_lmsapp.java* 🡪 generate complied class file “*hello\_lmsapp.class* “  
         java hello\_lmsapp 🡪 for run class file   
          
         A screenshot of a computer

        Description automatically generated with medium confidence

Figure 11. create\_folder\_for\_project.

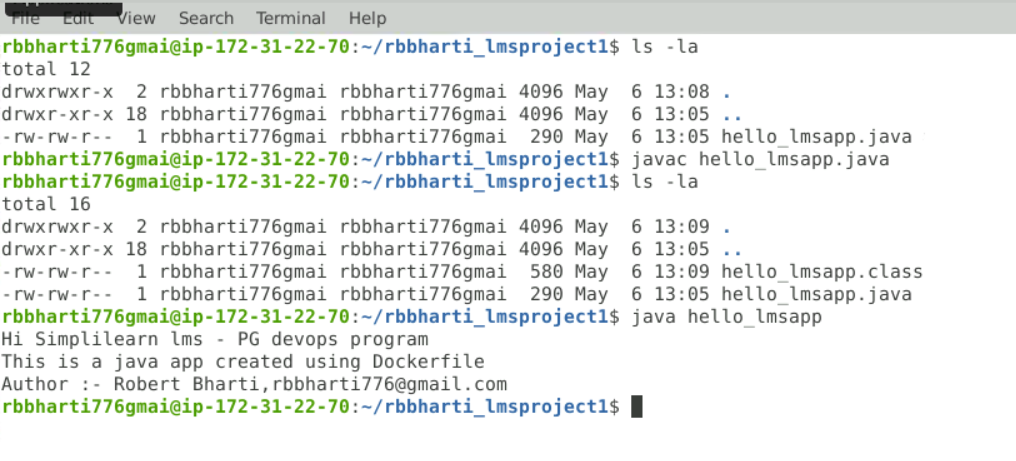


Figure 12. run\_javaCode\_onlocal\_system

* + - Dockerfile for sample application
      * Dockerfile is used to create image for your application. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Create a file called Docker File and edit it using vim or nano. Please note that the name of the file has to be "Dockerfile" with "D" as capital.
      * Dockerfile:

## ***use base image to run java app***

FROM openjdk:8

***## define workdir***

WORKDIR /var/www/java

***## copy all files & folders***

COPY . /var/www/java

***## Run for generate java compile class***

RUN javac hello\_lmsapp.java

***## Excute java Class***

CMD ["java", "hello\_lmsapp"]

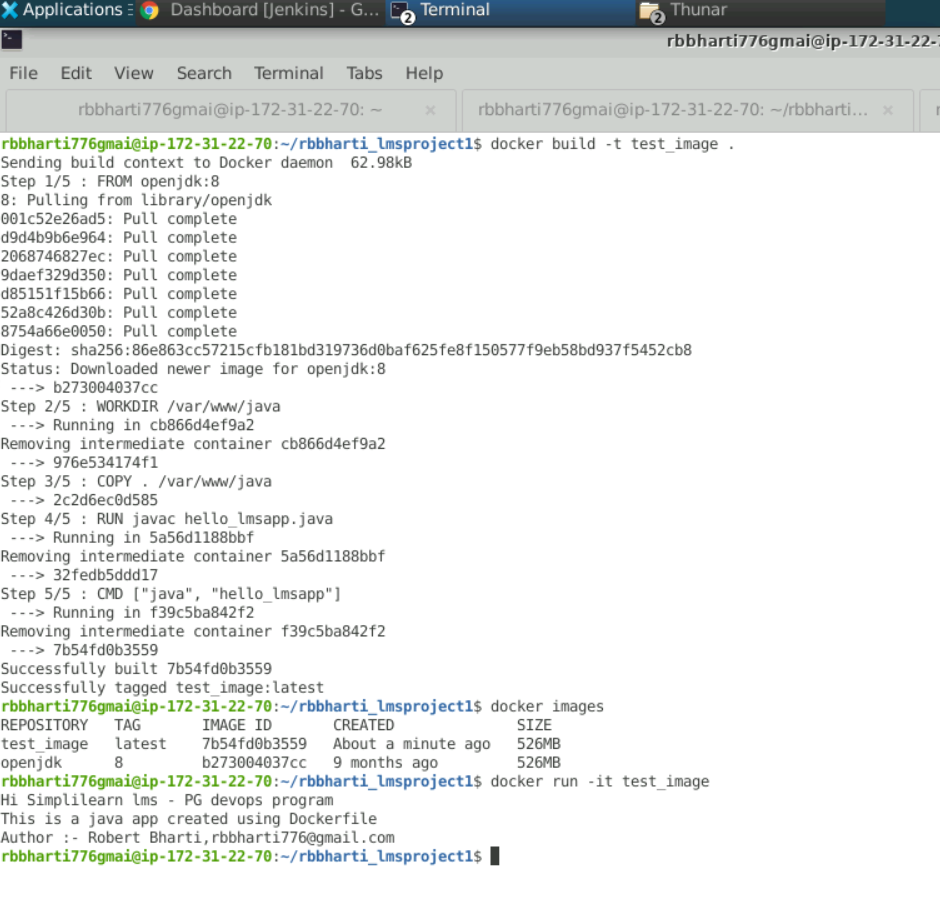
* + - * For run Dockerfile and create image of your app on local system, you have check first docker is installed or not.  
         docker -- version 🡪 check docker version  
          
         then, execute below command for generate docker image  
         docker build -t test image .   
          
         For check your image create or not run below command   
         docker images   
         For run your app docker image on system  
         docker run -it test\_images  
           
           
        

Figure 13. run\_docker\_file\_locally

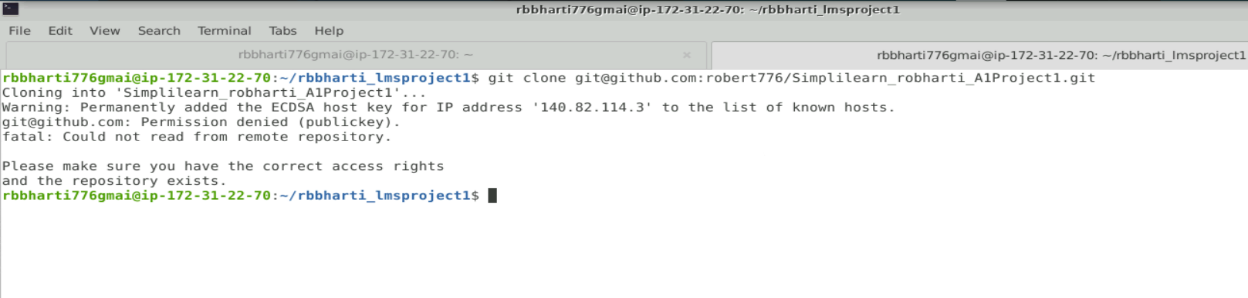
* + **Configurate Continuous Integration using git as Source Code Management System**
    - Once, java application and docker build run successfully on local system, commit these files in your git repo, so that can use for create CICD pipeline. <https://github.com/robert776/Simplilearn_robharti_A1Project1.git>
    - You will get error, if your machine ssh key is not added in your repo  
      

Figure 14. ssh\_key\_missing\_ingit\_repo

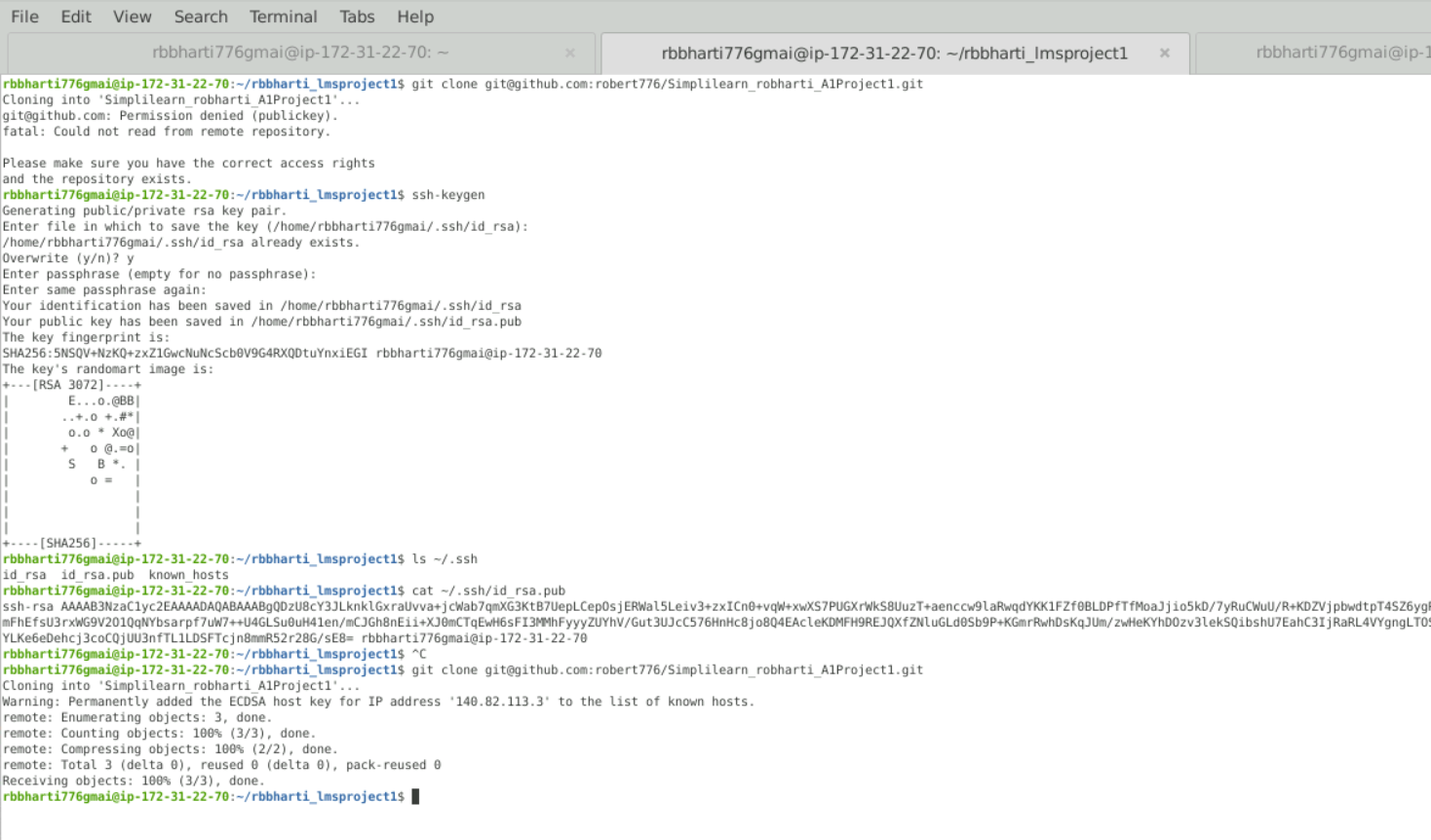
* + - For fix below error, you need to generate ssh key, copy it and add to git repo. Below command and steps can help you.
      * Follow below step in your local machine 

Figure 15.clone\_repo\_inlocalsystem



Figure 16.Add\_ssh\_key\_inGitRepo.

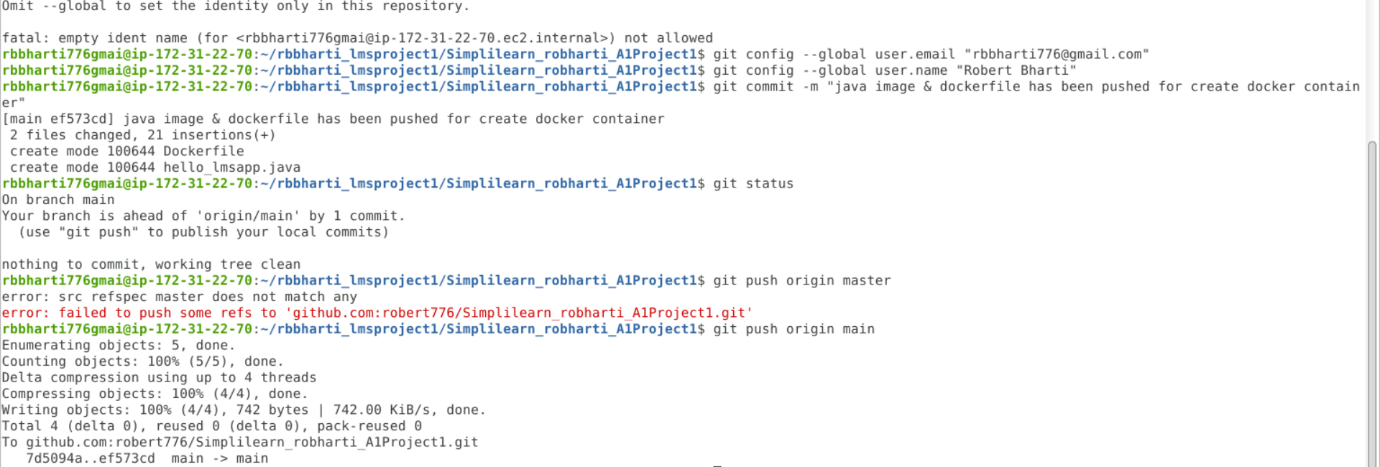
* + - Post after clone repo in local system, add your application and Dockerfile inside project folder and push it repo  
        
      

Figure 17.commit\_code\_repo

* + - After successful commit change in repo, we need to configure our Jenkins pipeline for start CICD.
  + **Configure Continuous Deployment and execute a Jenkins pipeline job and execute dockerfile respectively through a shall script.**
    - For create Jenkins pipeline, Open Jenkins, create a freestyle project. Use the GitHub url as the SCM and In the build section execute docker build command to test whether Jenkins is able to execute it

Jenkins > New Item > Freestyle project (BuildJob) - SCM > Git > Add your repo url

- Build > Add build step > Execute Shell

Command: **docker build -t my\_test\_image .**

- Save

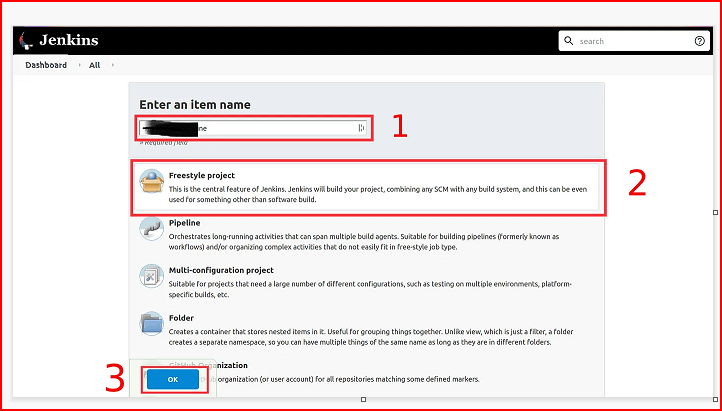
* + - 

Figure 18.Create\_freestyle\_pipeline

A screenshot of a computer

Description automatically generated with medium confidence

Figure 19.inserting\_git\_repo\_details\_inPipeline

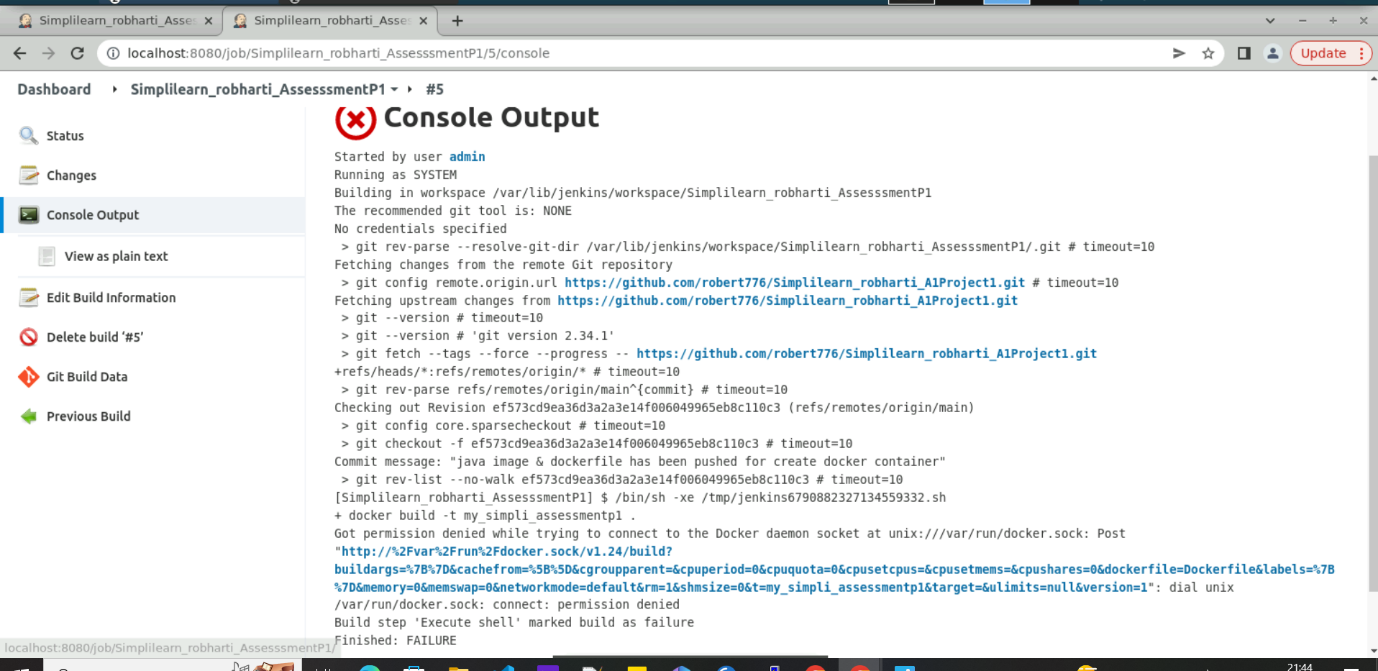
* + - Once freestyle pipeline creation completed, click on Build bottom on left side panel and run the Job and Check the build console. If you get permission error, execute the below steps. 

Figure 20.build\_failed\_dueToDockerpermission.

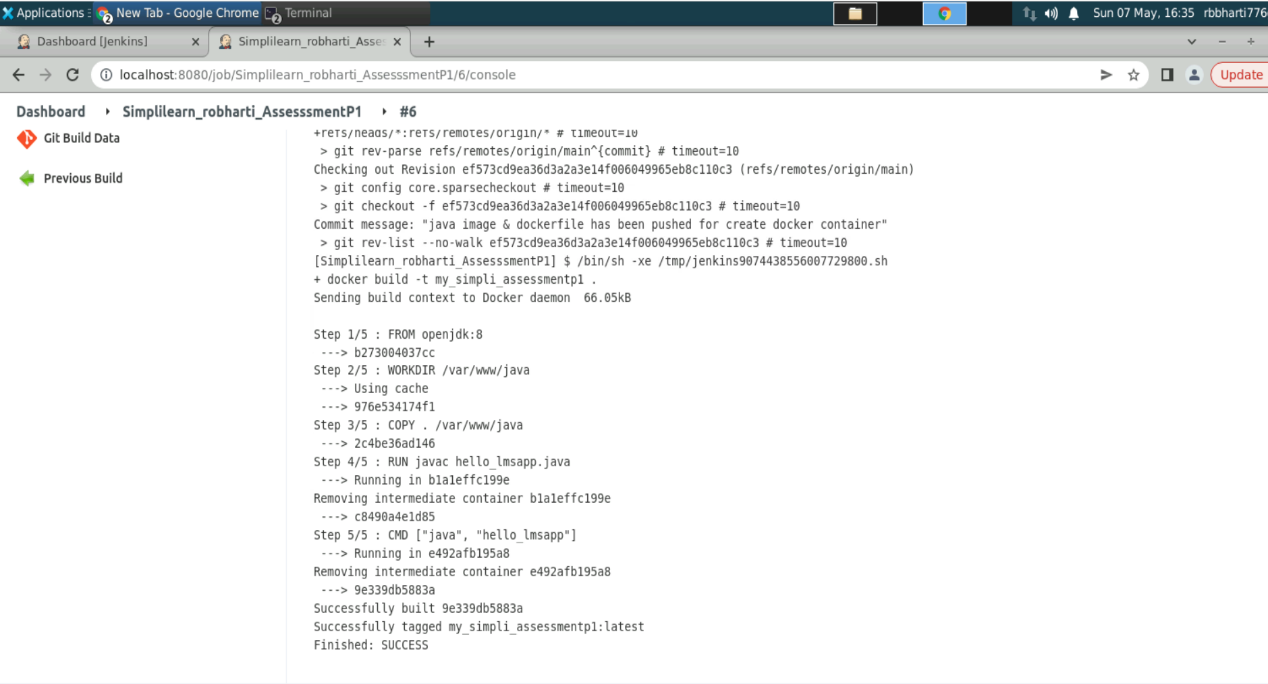
* + - For fix above error, run below command and restart Jenkins again from terminal.  
       sudo usermod -aG docker jenkins **sudo service jenkins restart  
       sudo chmod 777 /var/run/docker.sock**
    - Now, Re-login & build the job once again and see jenkins build logs.  
      

Figure 21Build\_get\_success\_after\_permission.

* + **Use a Jenkins plugin "CloudBees Docker Build and Publish" for Continuous Deployment in docker hub.**
    - Above steps can we also complete by plugins “CloudBees Docker Build and Publish” provide by jenkins. For that we have to add plugins in jenkins

> Jenkins > Manage Jenkins > Manage Plugins > Available(tab) > search for "CloudBees"

> Select CloudBees Docker Build and Publish

> Download now and Install after restart

- Restart Jenkins

- Relogin

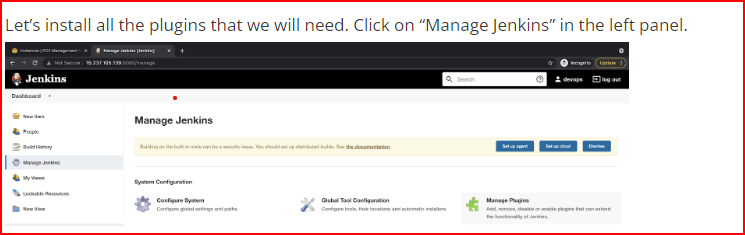
* + - Follow below steps to add plugins  
      

Figure 22.Add\_Plugun\_in\_jenkins

* + - A screenshot of a computer

      Description automatically generated

Figure 23.Add\_CloudBees\_plugin\_jenkins

* + - A screenshot of a computer

      Description automatically generated

Figure 24.CloudBees\_plugin\_completed

* + - Now go and restart you jenkins build and you will find your plugin in installed list.
    - After plugin setup, you have to configure pipeline as per CloudBees Docker Build and Publish so that can provide expected output. Follow below steps.

- BuildJob > Configure > Build section

- remove Execute shell

- Add Build step > Docker Build and Publish

- Repository Name: <Your\_docker\_id>/myassessmentimage

- Add credentials > Add > Jenkins - give username & password > Add

- Use this credential as Registry credential

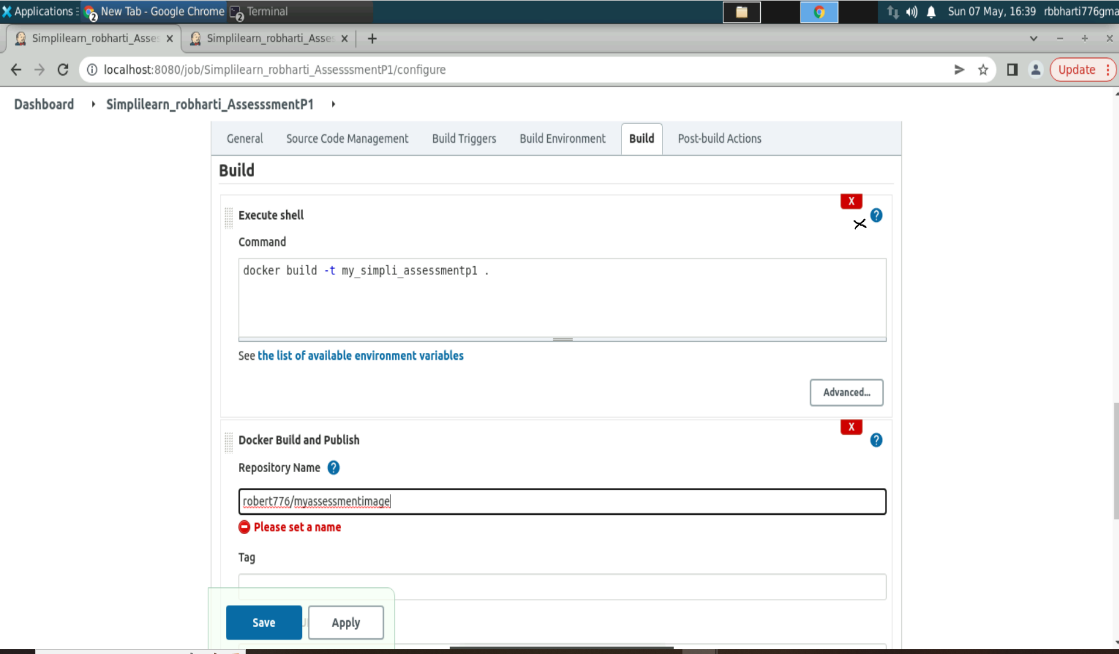
* + - Here, you have removed command form execute shell command and your dockerhub details  
      

Figure 25.Edit\_pipeline\_remove\_shall\_script

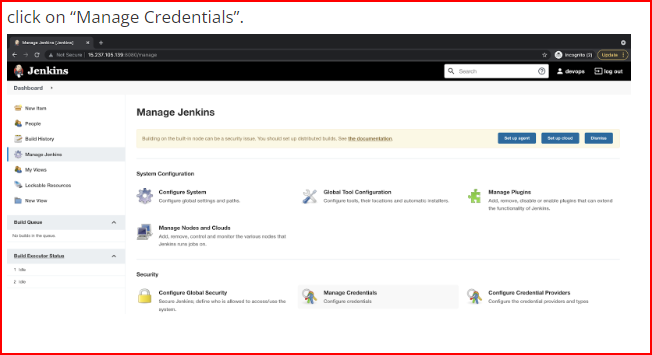
* + - Add your docker credentials by follow below steps  
      

Figure 26.Add\_credential\_in\_jenkins\_1

* + - A screenshot of a computer

      Description automatically generated with medium confidence

Figure 27..Add\_credential\_in\_jenkins\_2

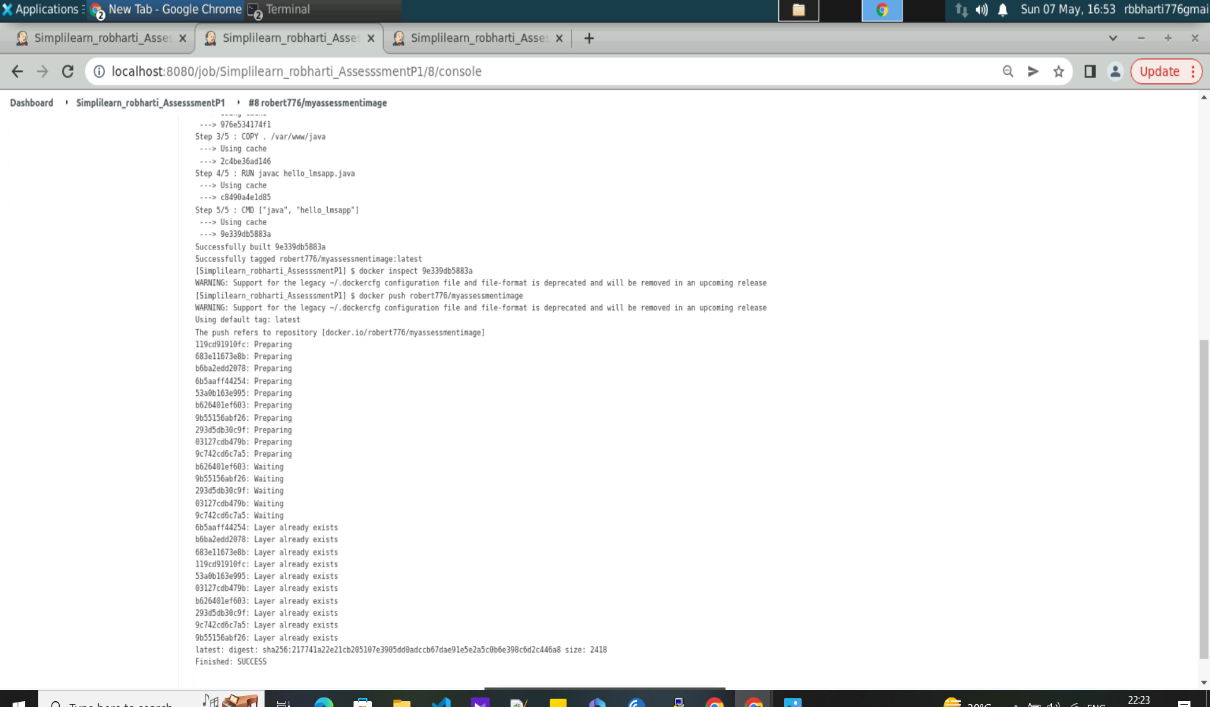
* + - Post after configuration, click on save button. Now run build manually by click on build button on left side panel and see output in console. 

Figure 28.Build\_logs\_trigger\_with\_plugin

* + - A screenshot of a computer

      Description automatically generated with medium confidence

Figure 29.dockerHub\_details

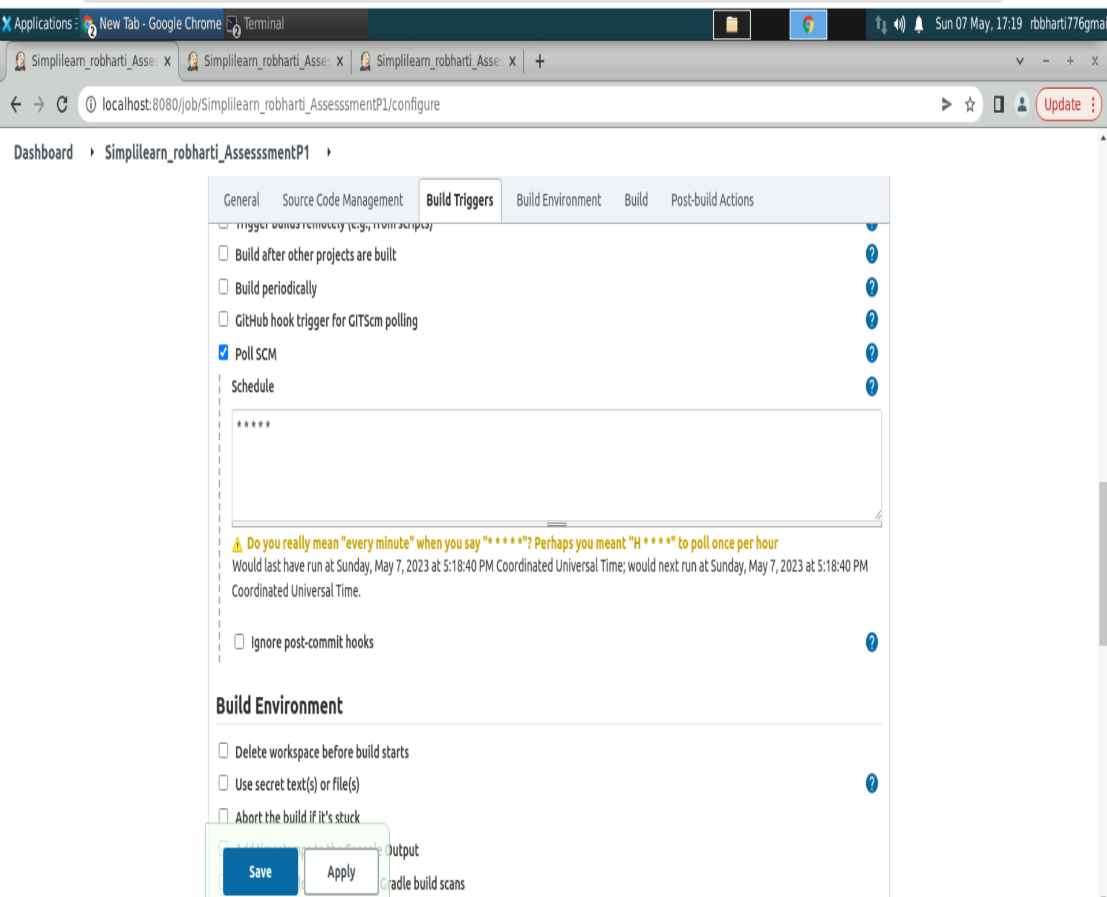
* + - Check image tags in docker hub repo and match it in jenkins build logs.
  + **Automate the process using Jenkins Build Trigger** 
    - Here for atomate our build trigger, we use Poll SCM option, what this option does is that it continuously queries the VCS, based on a predefined schedule, for new changes.
    - If new changes are encountered, then Jenkins will start the project build steps.
    - The predefined schedule used here is \* \* \* \* \* which is a time schedule for every minute, Jenkins uses a syntax similar to Crontab, explore the format on crontab.guru  
      .   
      

Figure 30.Poll\_SCM\_configuration\_for\_trigger

* + - After change and save, We are done setting up our project, the last step is to click Save at the bottom of screen and start the first build by clicking on the Build Now button.  
        
      A screenshot of a computer

      Description automatically generated

Figure 31.Build\_trigger\_pollSCM

* + - A screenshot of a computer

      Description automatically generated

Figure 32.Final\_build\_details

* **Solution overview**:
  + In this demonstrating, we have seen the continuous integration and delivery by building a Docker image and pushed it in docker hub using Jenkins Pipeline.
  + On Linux (ubuntu) machine, we are installing and configuring Java & JDK, git, Jenkins server and docker so that app can be tested and build locally and further can be deployed.
  + Excepted errors have been highlighted while configuration of CICD pipeline and provide solution how to fix it.
  + Git poll scm play important role in identify any change or commit in repo periodically and trigger Jenkins build.
  + Jenkins is a better choice to automate the Docker image build and pushing image to cloud repository provider.
  + Docker follows the Domain registry concept to push and the images and have all images tags history also with it.