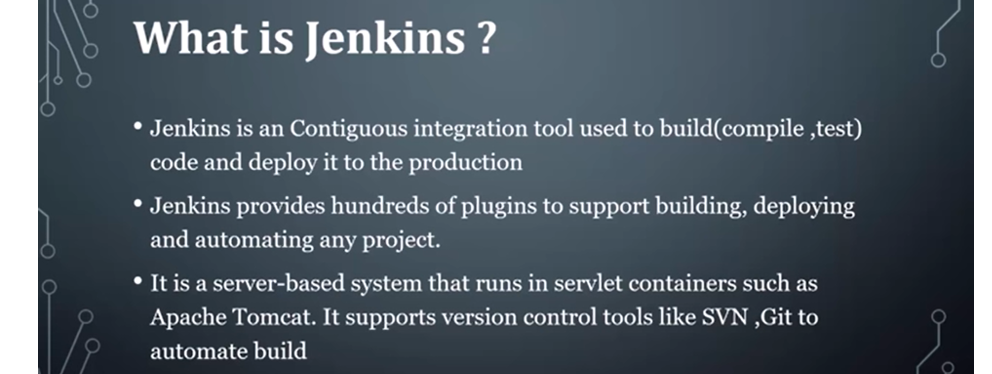
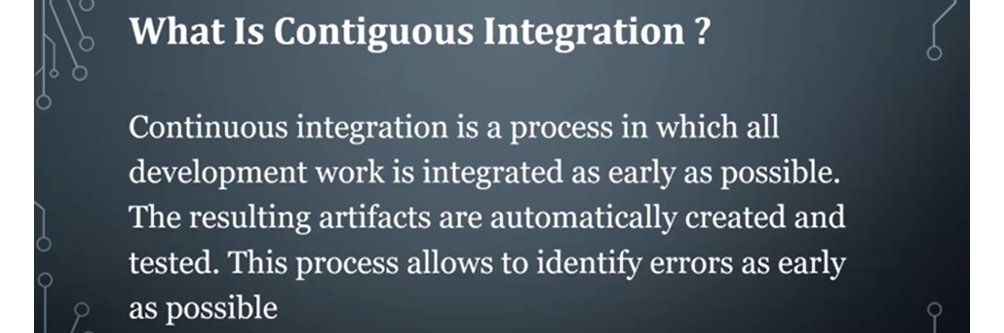
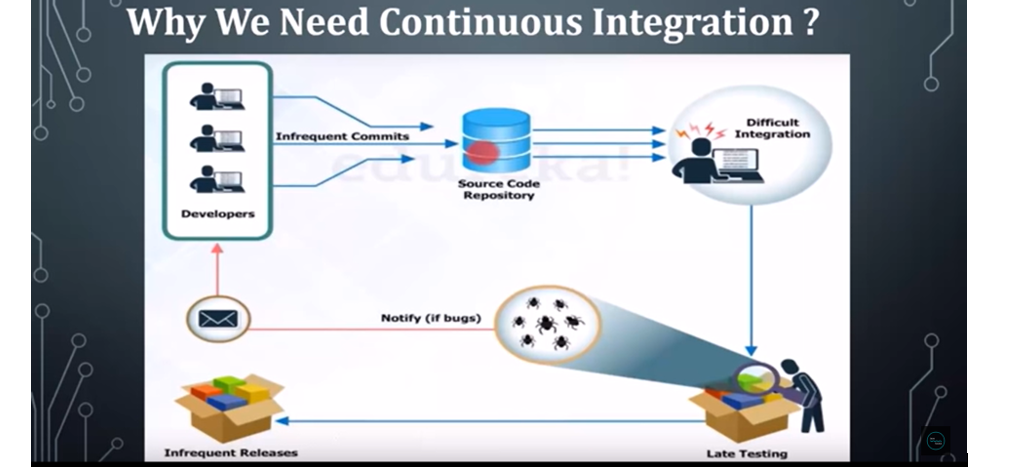
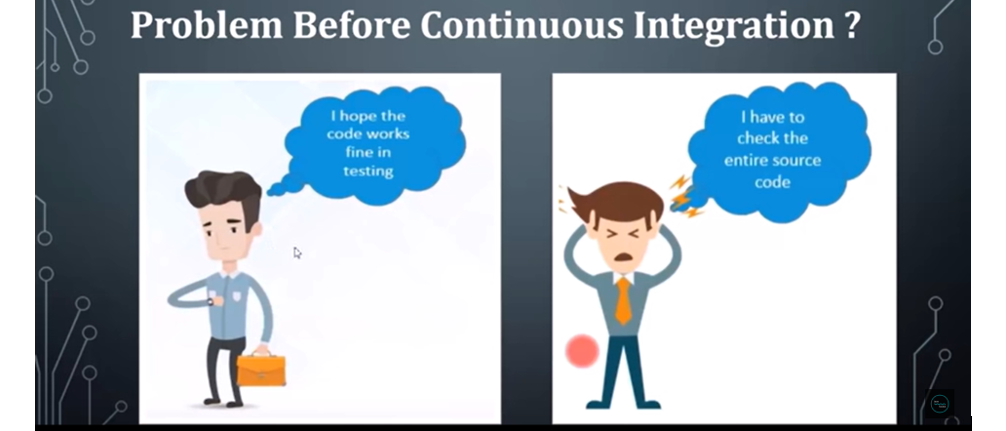
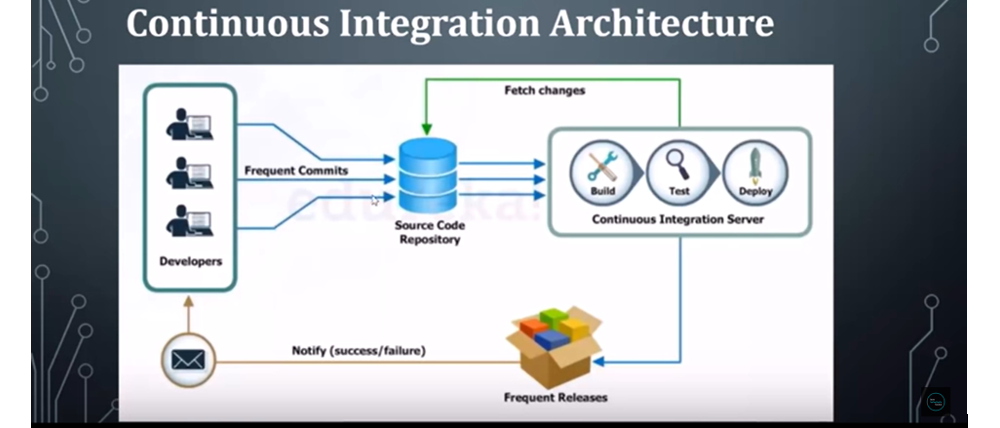
* **1. What is Jenkins | Continuous Integration & Installation process in Windows**



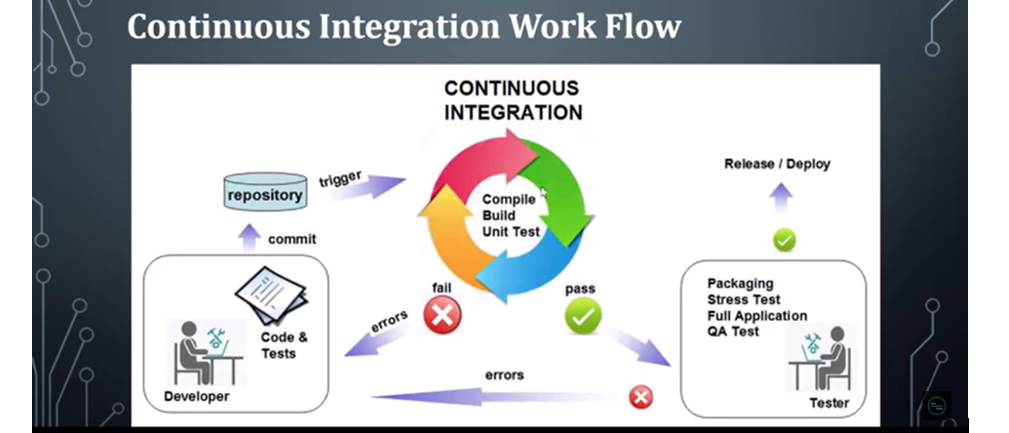


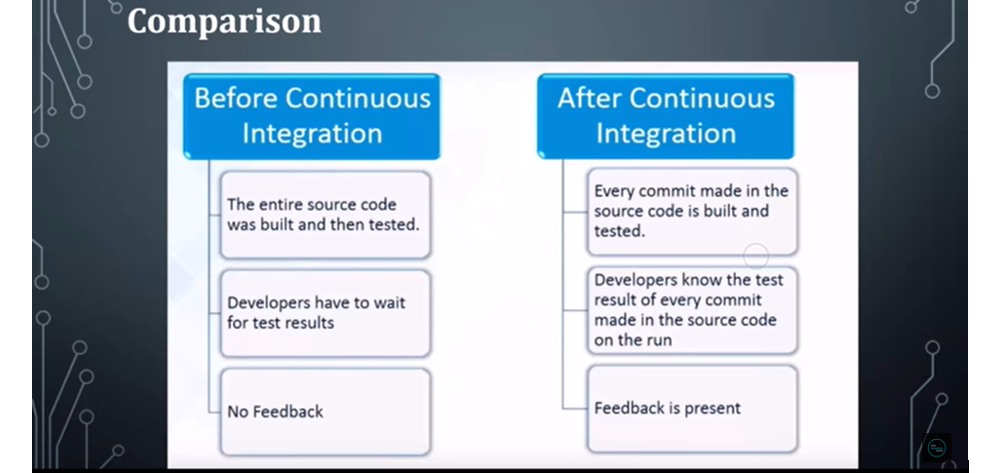






So here you can see for the developer no need to wait for the test result, so at the time of build only if we get any error developer can get the error details.



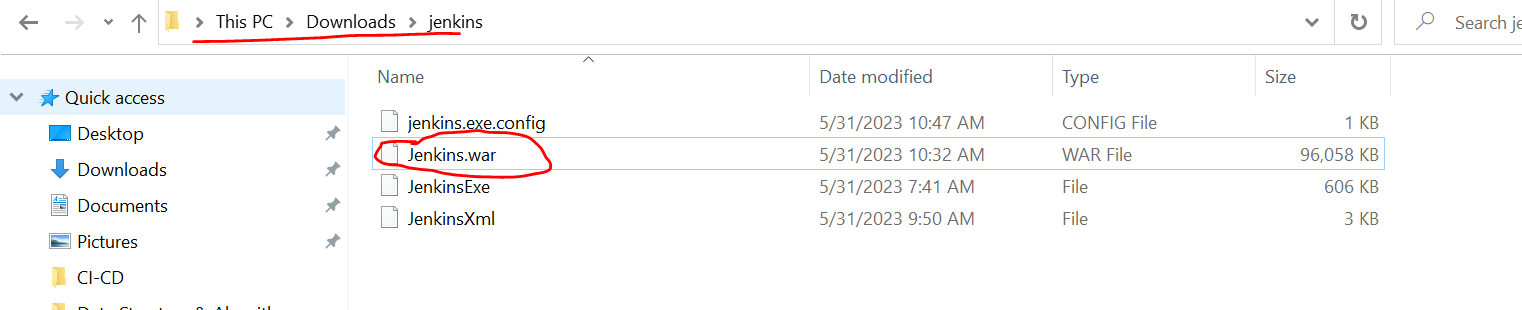


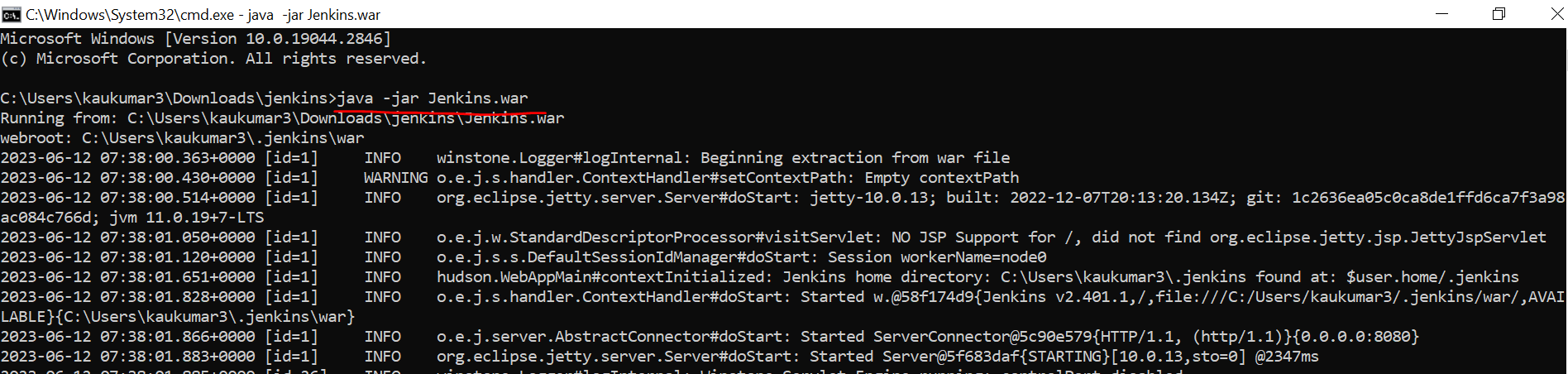
**Let’s Install Jenkins in our Machines:-**

[**https://www.jenkins.io/doc/book/installing/windows/**](https://www.jenkins.io/doc/book/installing/windows/)

**or another way download war file and run…otw first one is run installer…**

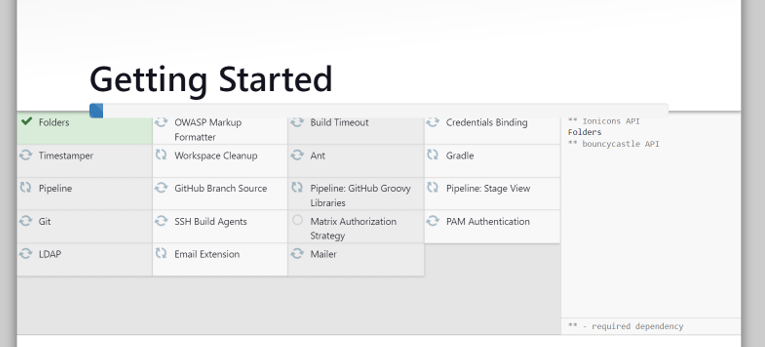
**Download the Jenkins war file and start**

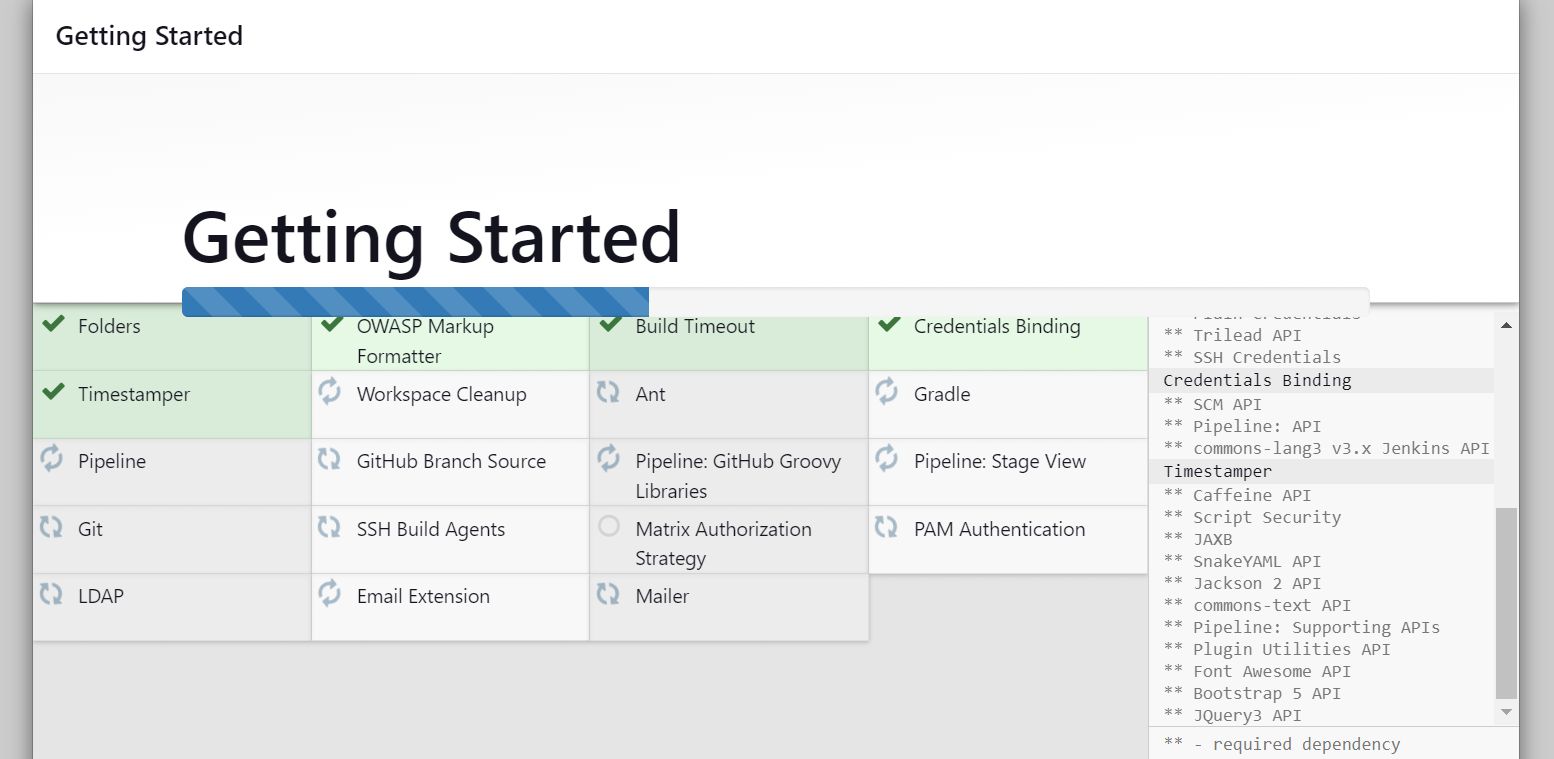




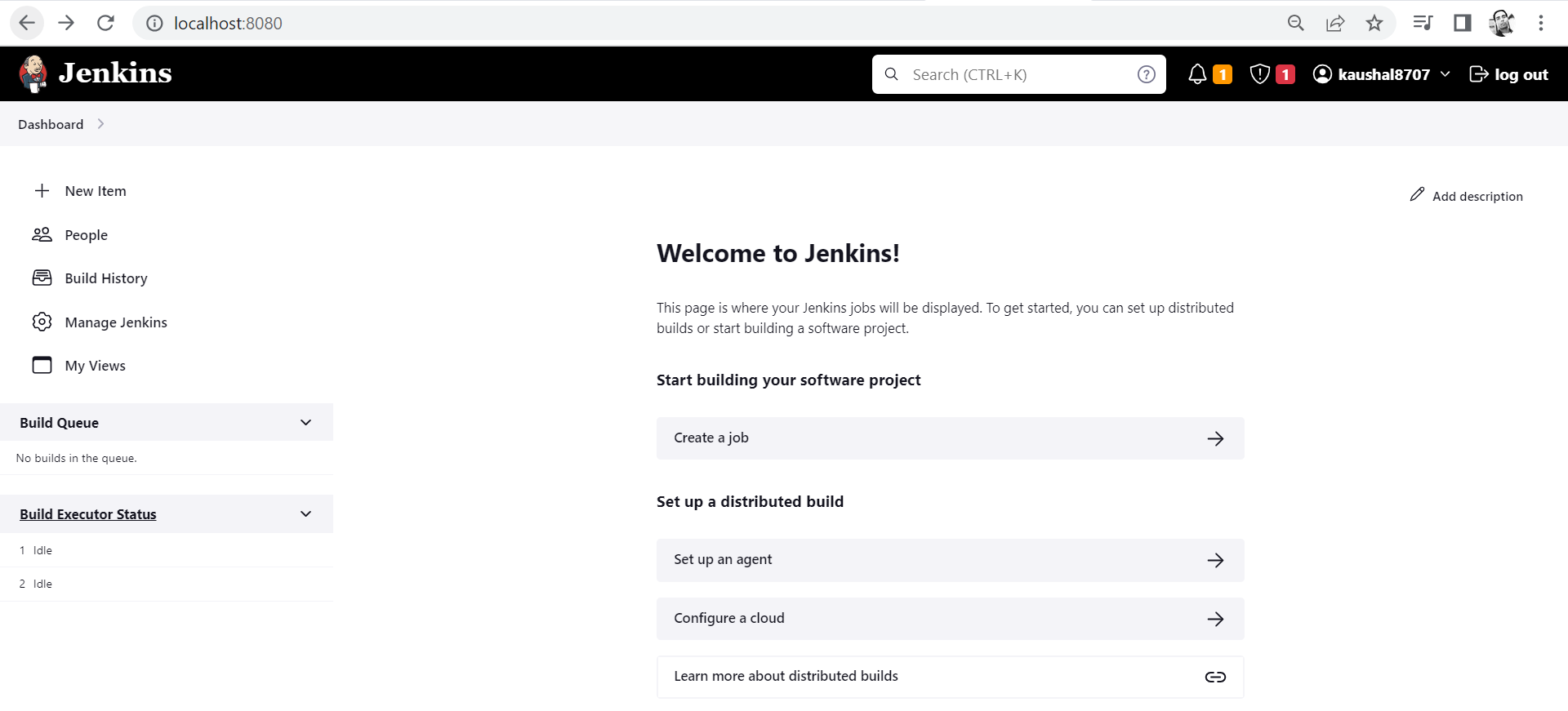
**>> java -jar Jenkins.war**

**Jenkins by default runs on port 8080**





[**http://localhost:8080/**](http://localhost:8080/)



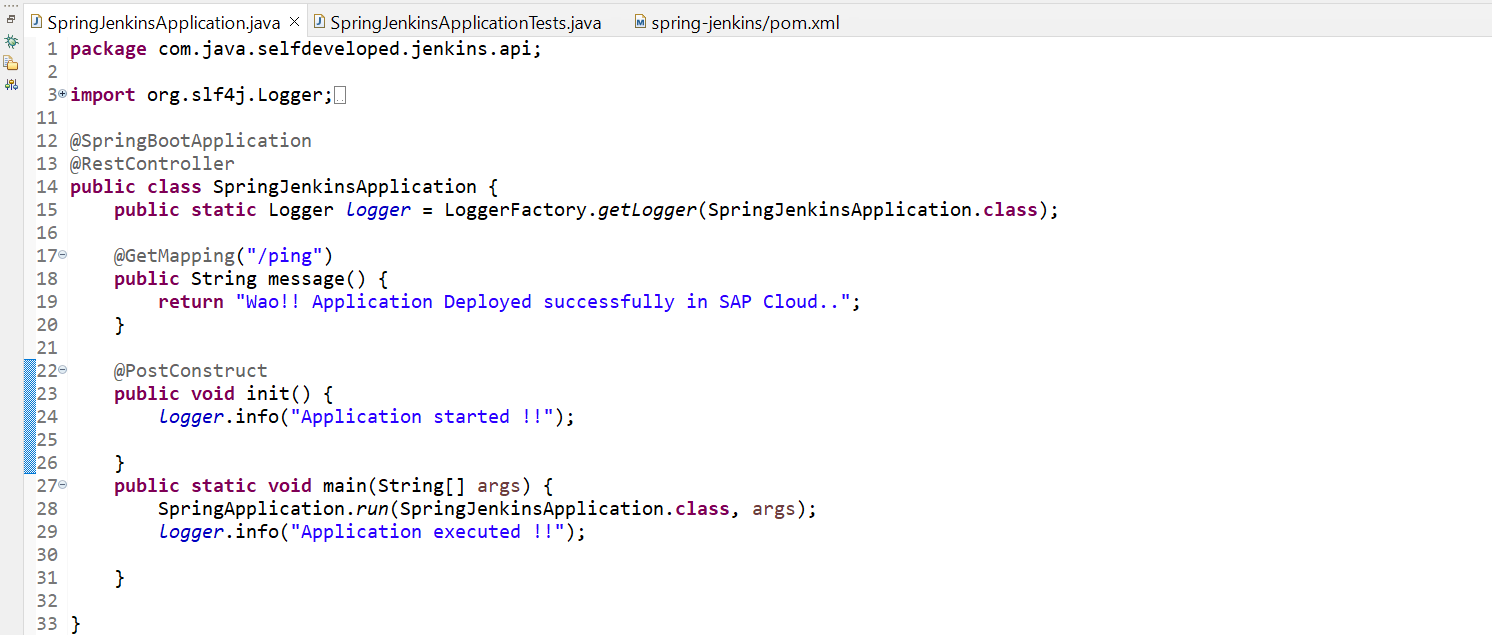
* **2. Jenkins | continuous integration with GitHub & Enable Notification**

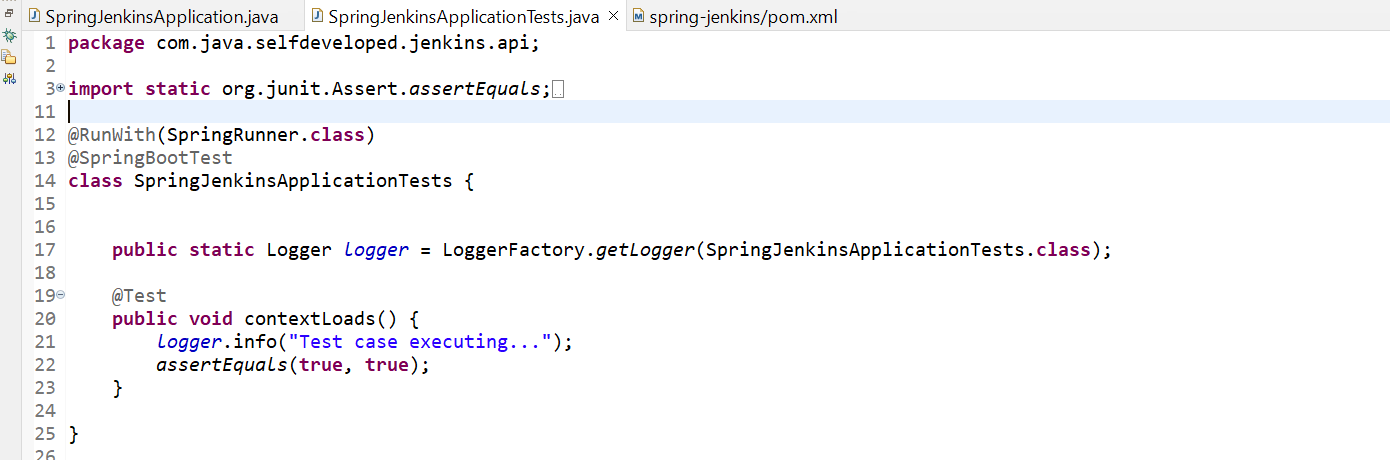
Let’s see how we can integrate our Continuous Integration tool i.e Jenkins with our GitHub.

Let’s first create a project in Eclipse/STS.

* Let’s first create a project with name spring-Jenkins
* Create a Git repository with name spring-Jenkins
* Commit your code to this Git repository.

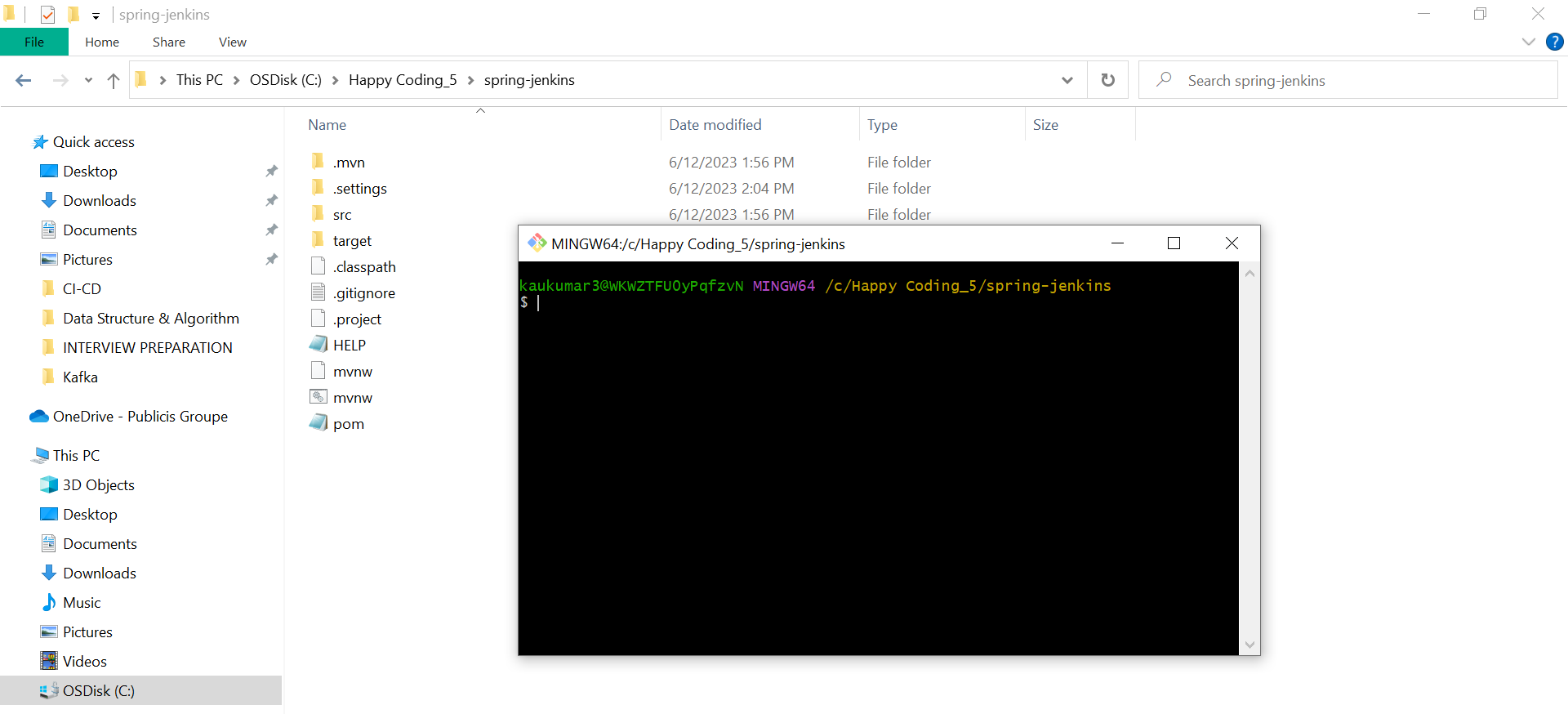
\*\*\* Now we have written few lines of codes and added logger as well …and now we have written a test class to identifies either our test process is automated or not with the help of Jenkins.





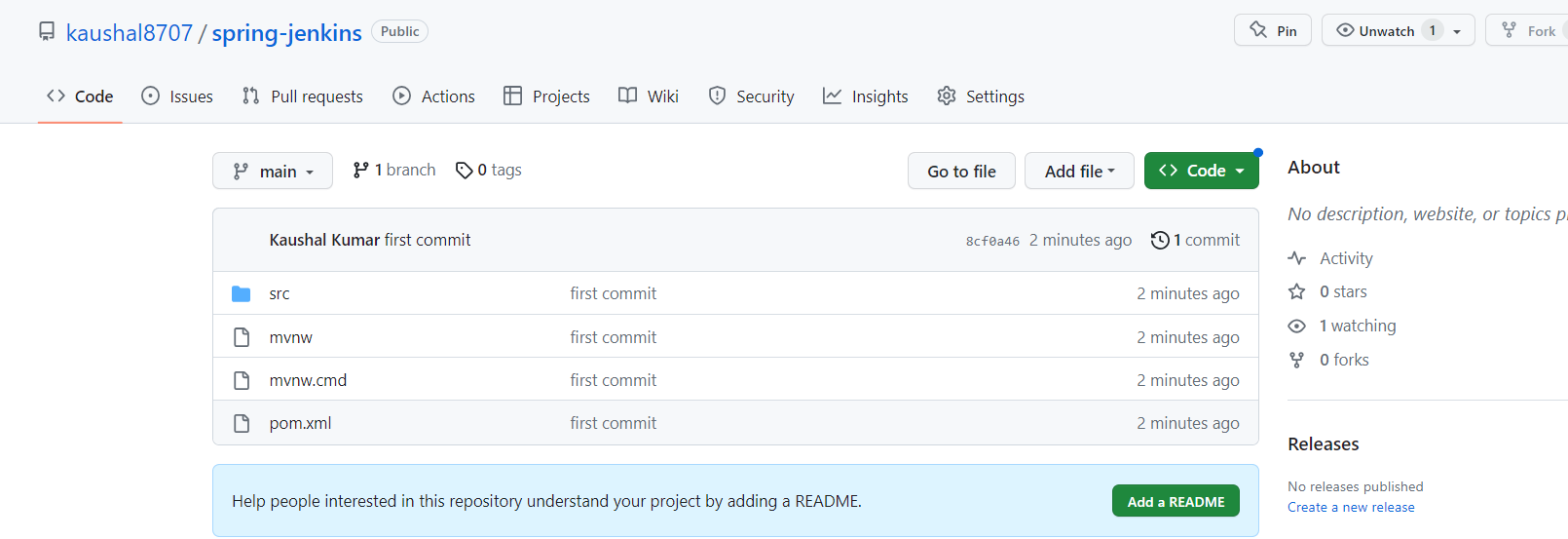
\*\*\* We have created one repository and now let’s sink up our code with that repository.

\*\*\* Now go to project workspace and open Git bash here and run below commands…



* git init
* git status
* git add \*
* git commit -m "first commit"
* git branch -M main
* git remote add origin https://github.com/kaushal8707/spring-jenkins.git
* git push -u origin main

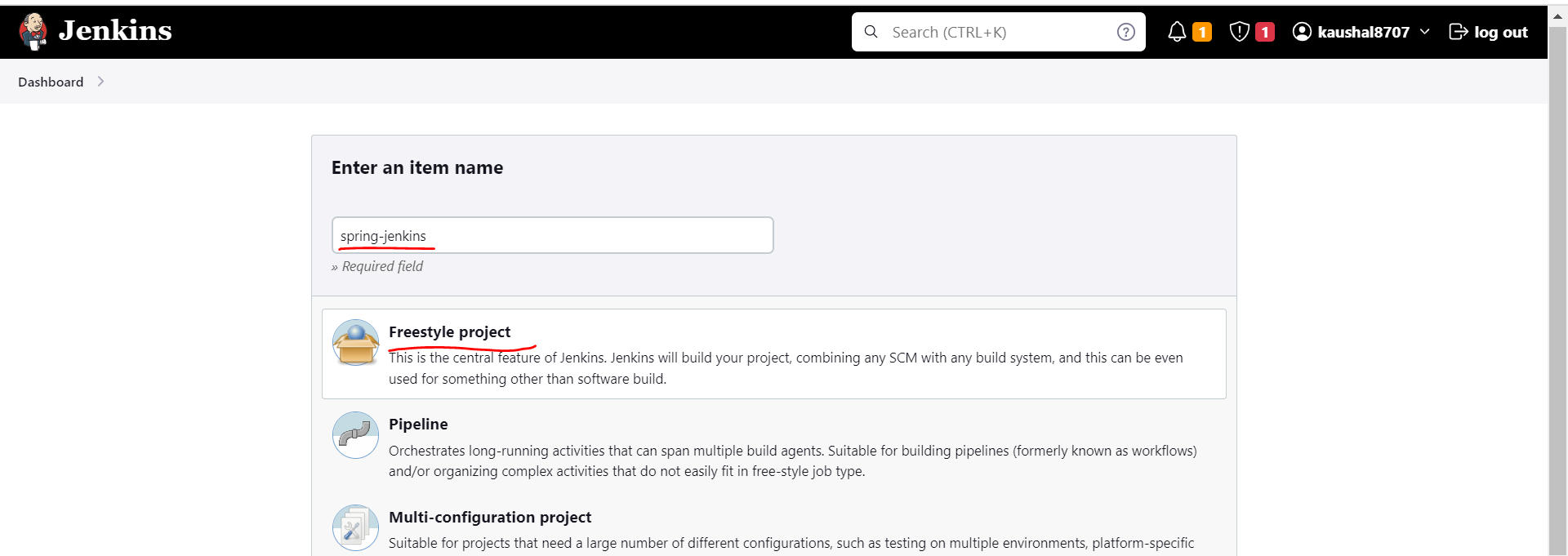
now we can see our project in Git hub.



Now come to Jenkins dashboard.

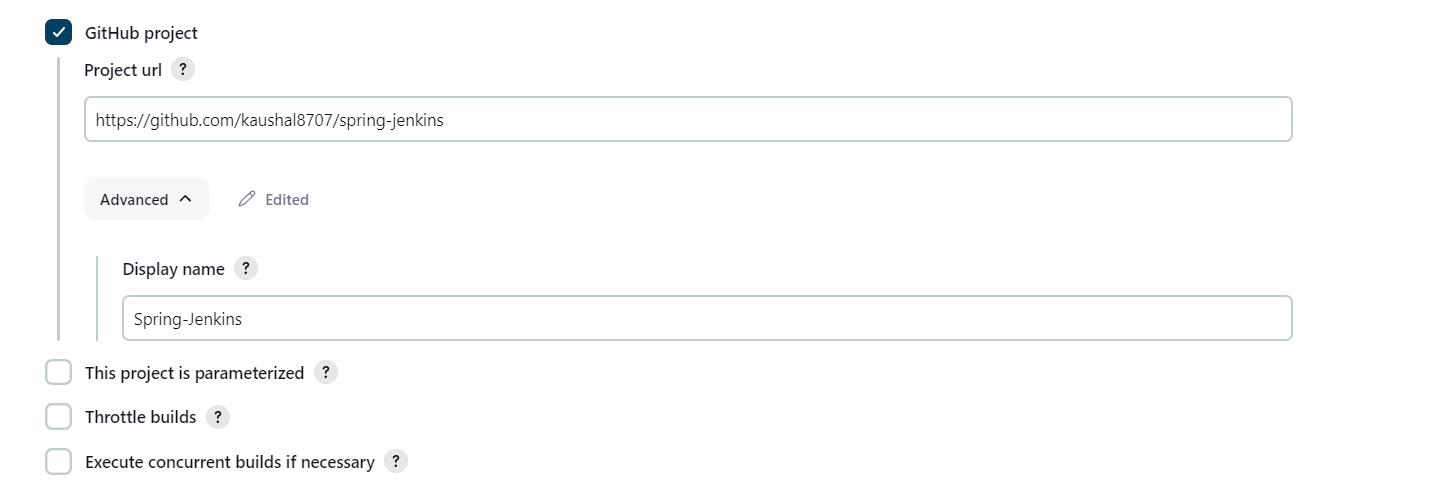
To come to Jenkins dashboard, you need to run Jenkins.war and then hit <http://localhost:8080/>

Let’s create a new **Job**.



We have selected freestyle project even we can select maven project also we have an option.

Since we are going to sink up Jenkins with our git base project so select GitHub project and provide GitHub url.



------------------------------------------------------------------------------------------------------------------------------------------



\*\*\*\* Then we can schedule our task, or we can schedule our build.

Suppose in every time change in our this **spring-jenkins** repository I want to be a new build. So, you can schedule your task. Now we can give cron expression like at every minute or at every hour whatever the changes will be done to your repository it will automatically pick and it will do the build with the help of Continuous Integration server. As a temporary basis we are giving \* \* \* \* \* but this is not the standard.

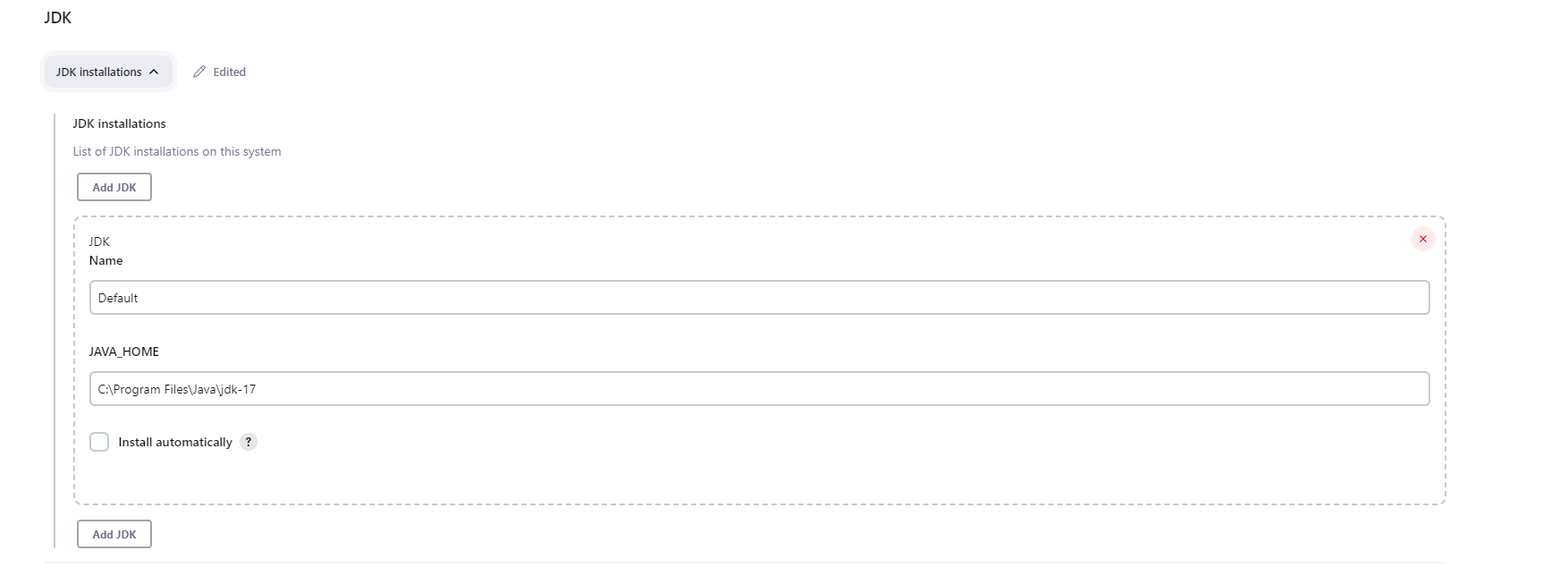


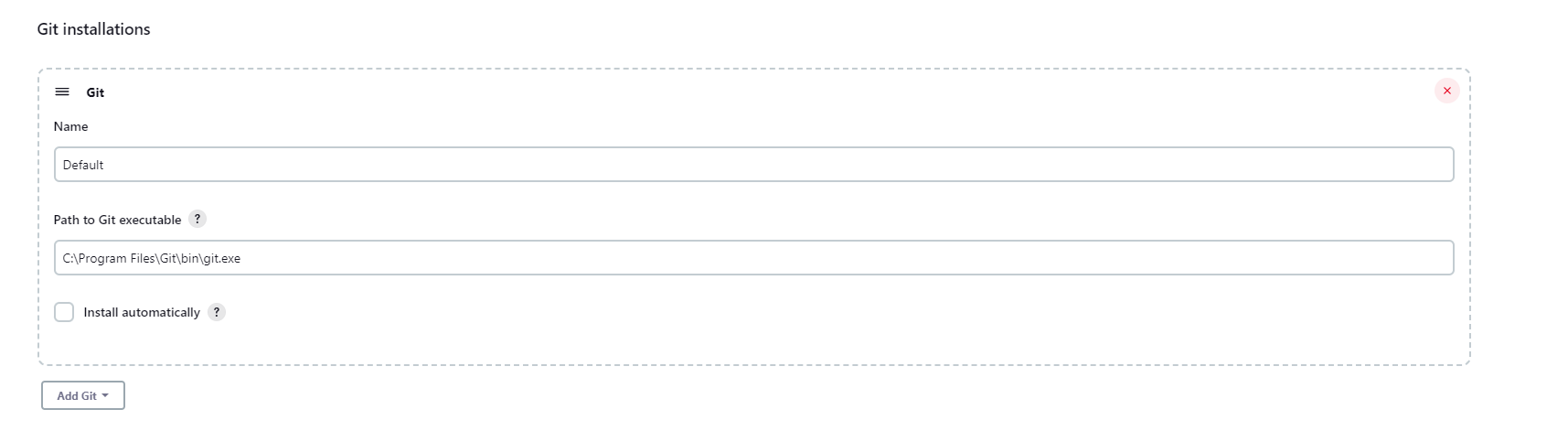
In Build step select **Invoke top-level Maven targets**

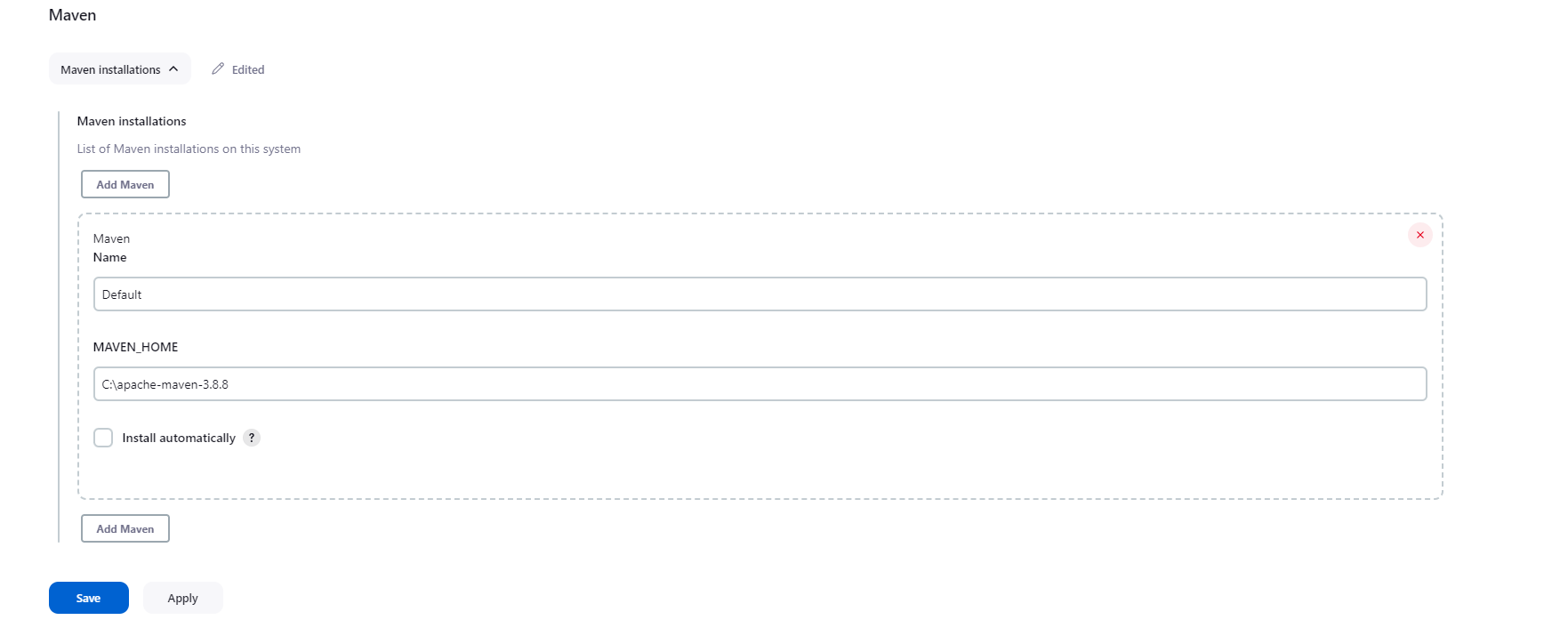


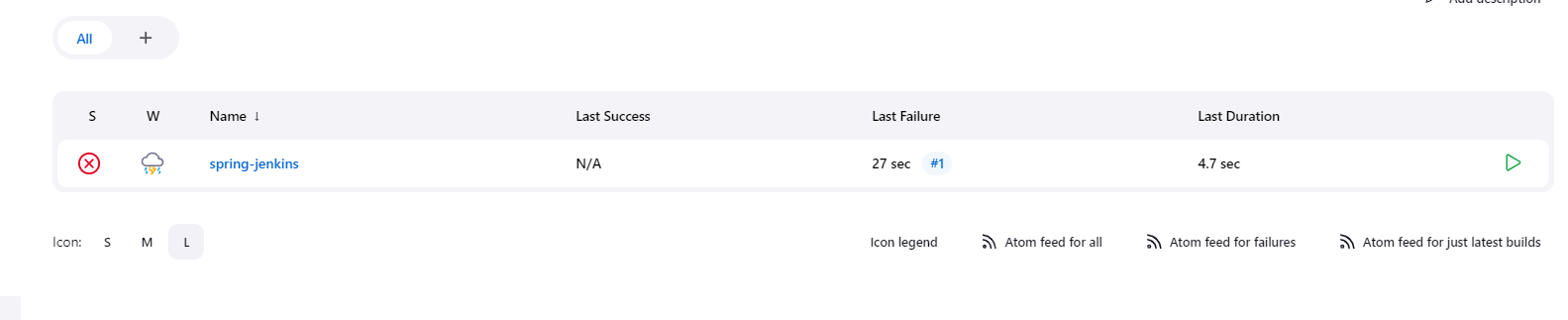
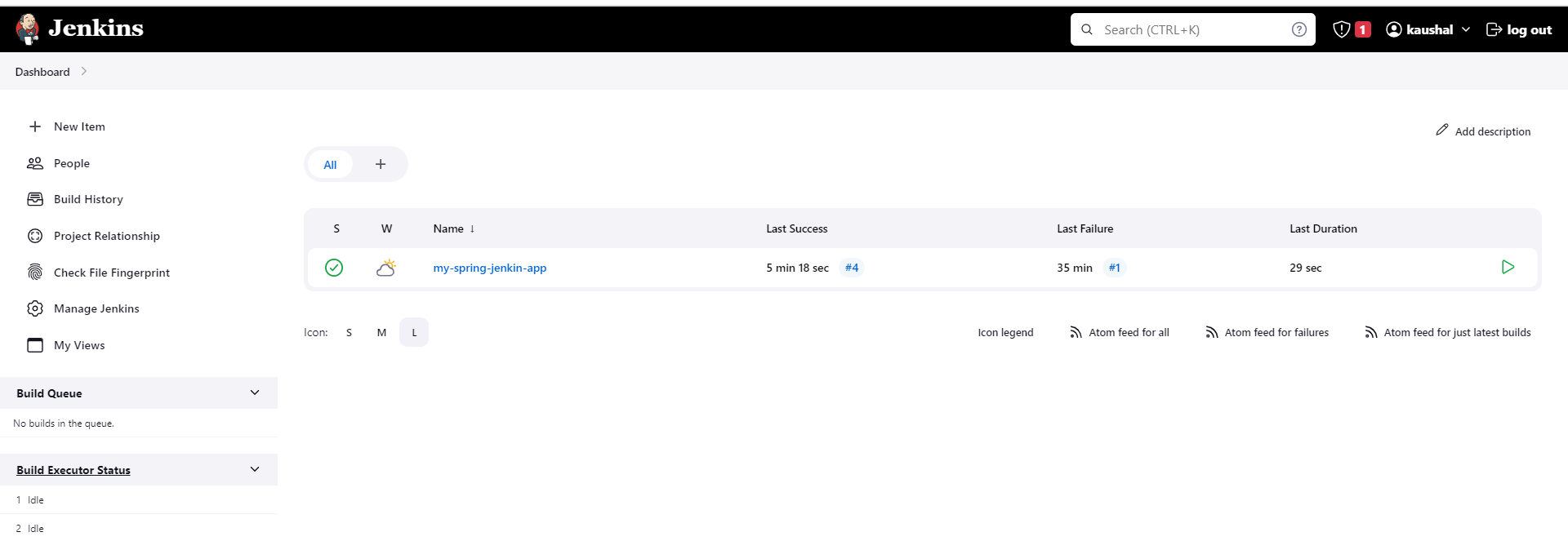
Saved it now go to our jenkins.

Go to Manage jenkins -> Tools

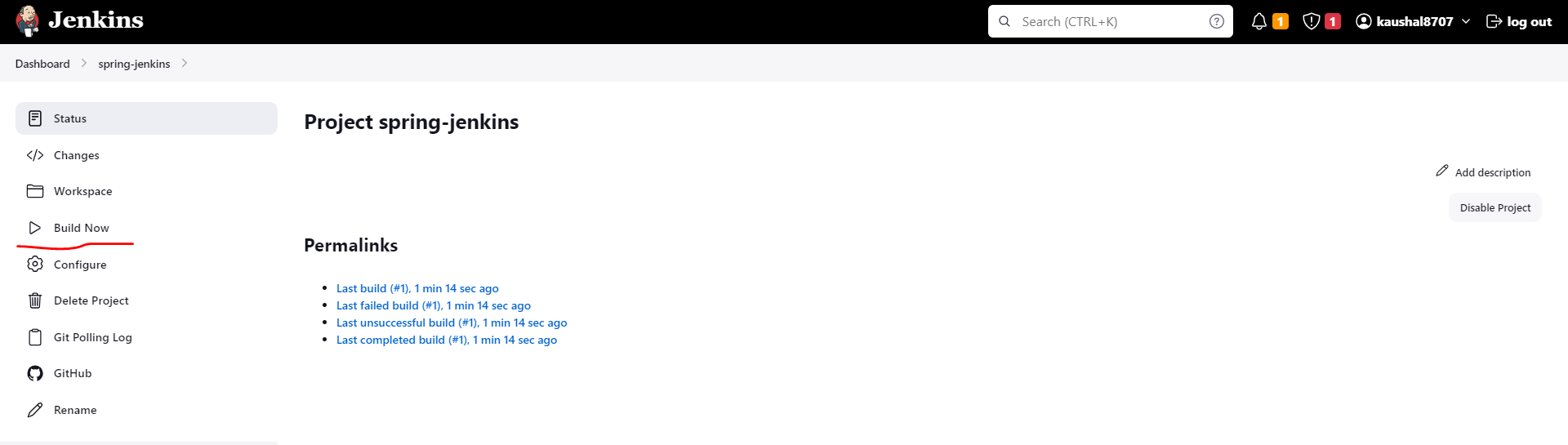




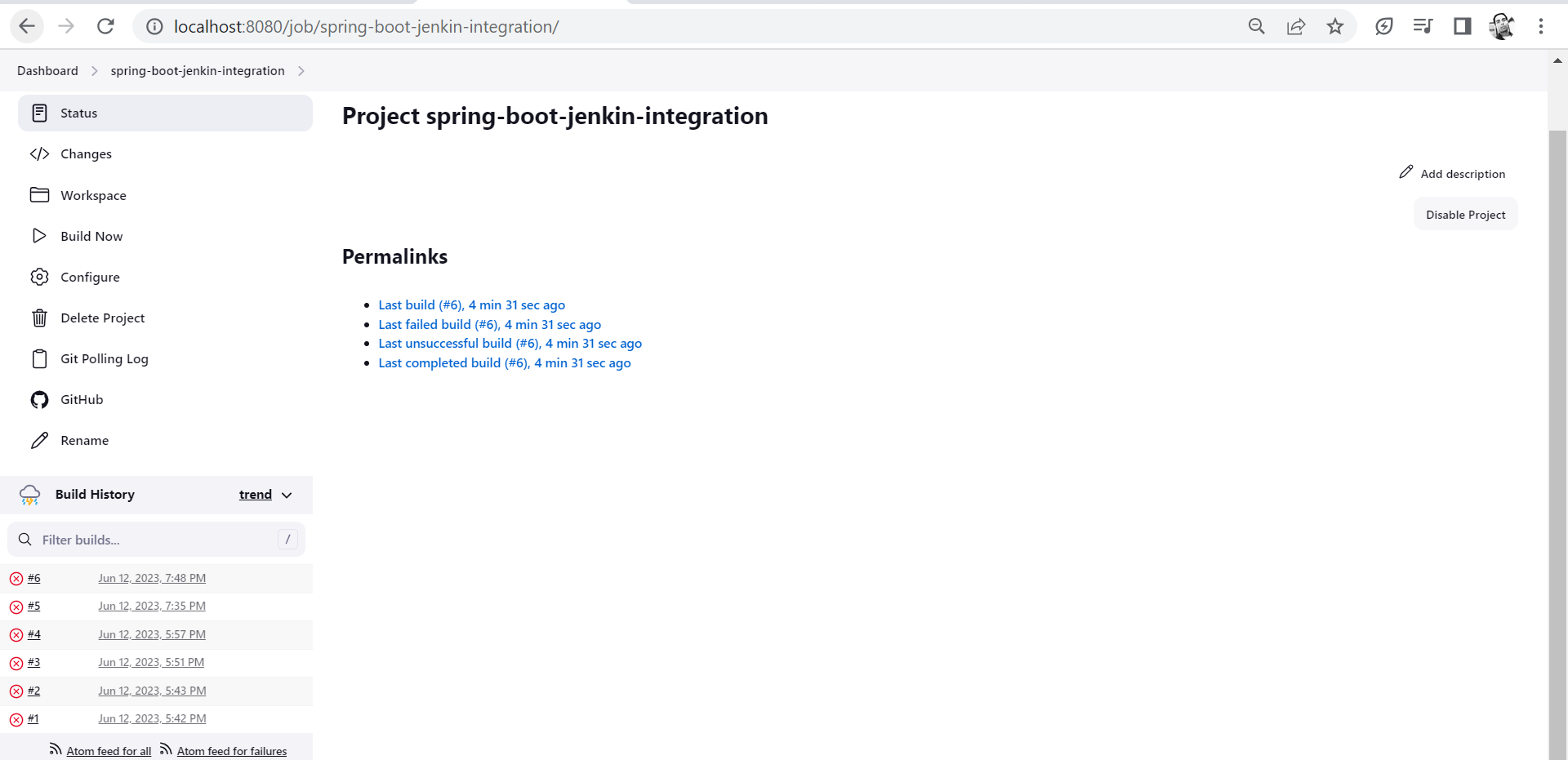
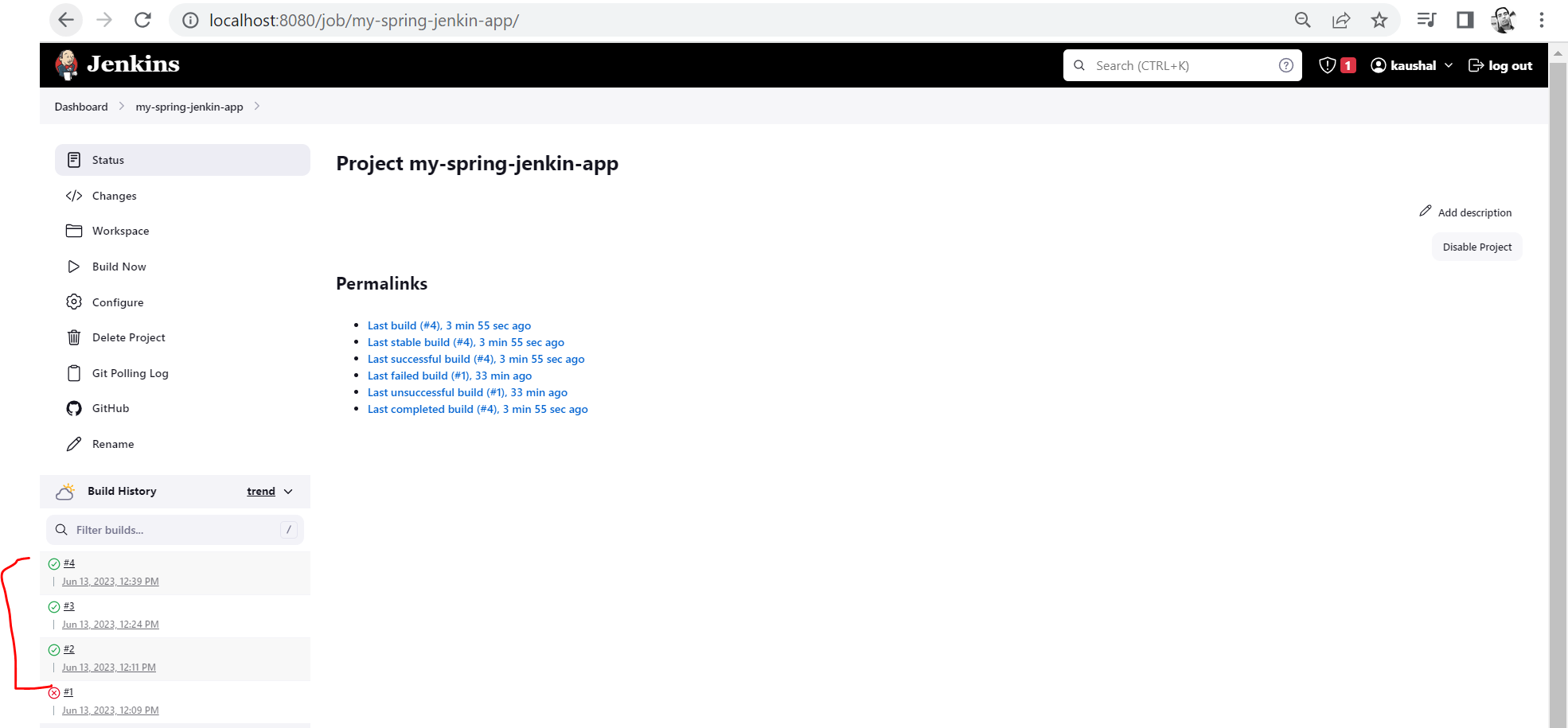


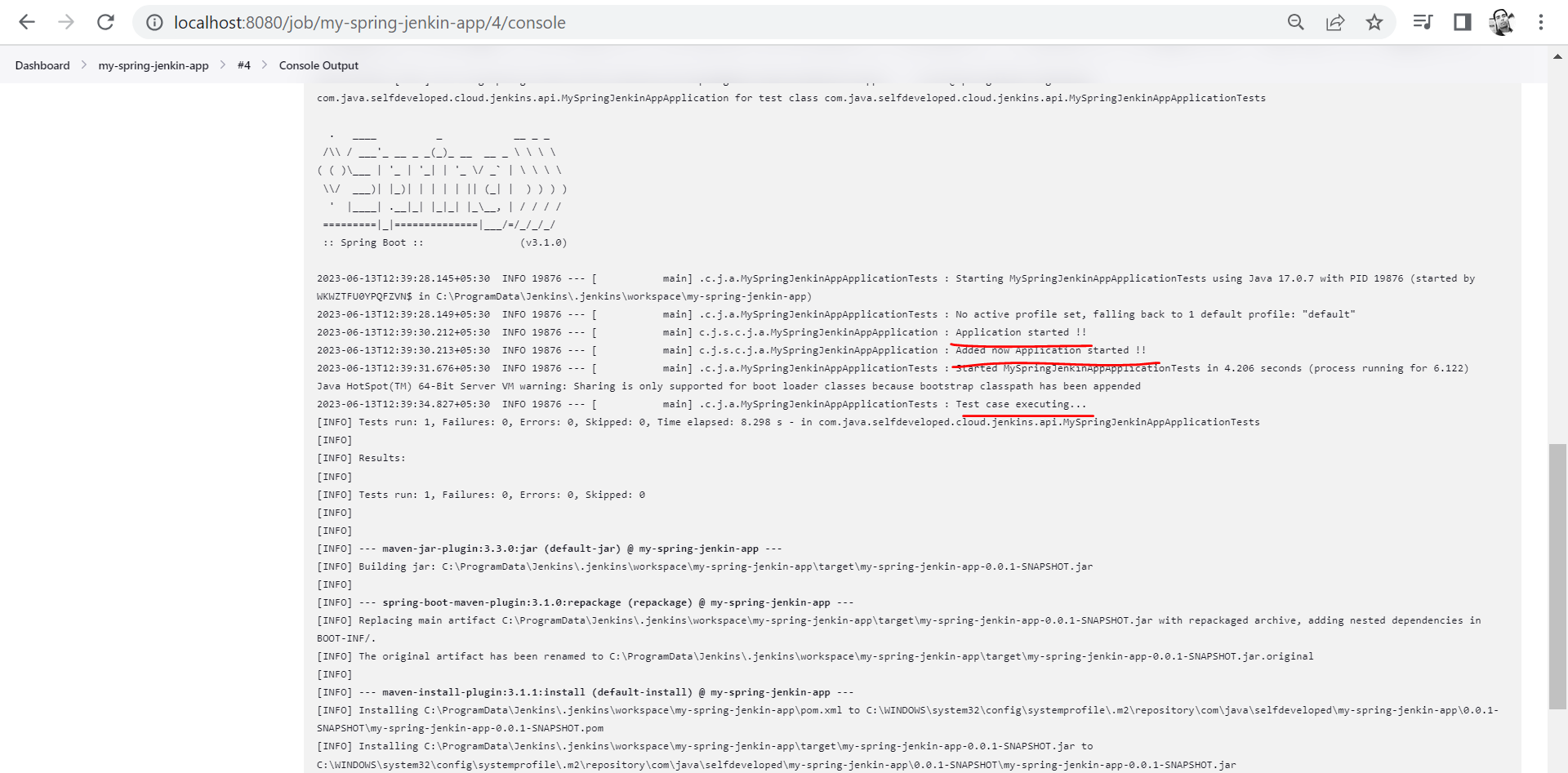


Click on our project

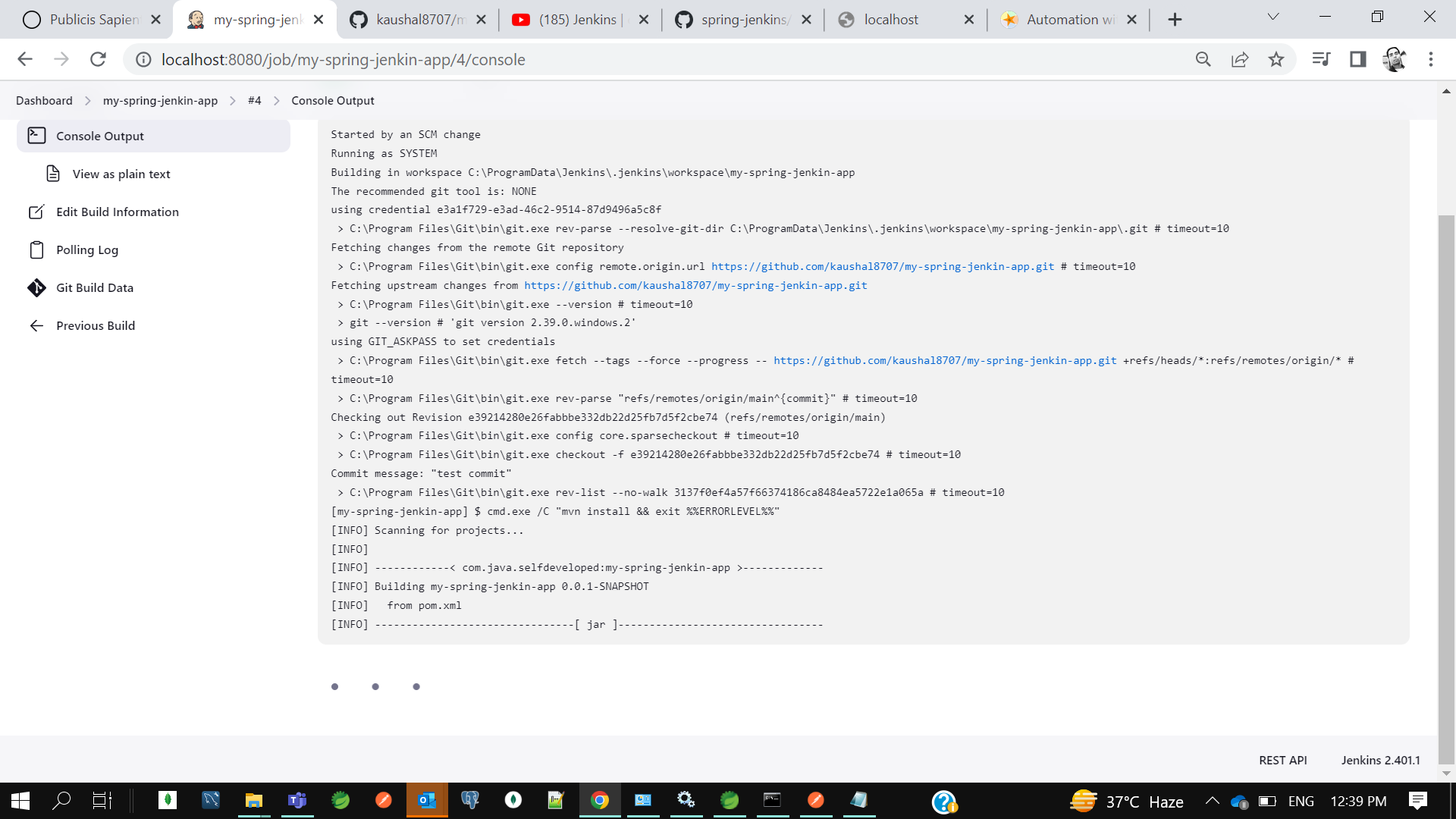


Click build now.





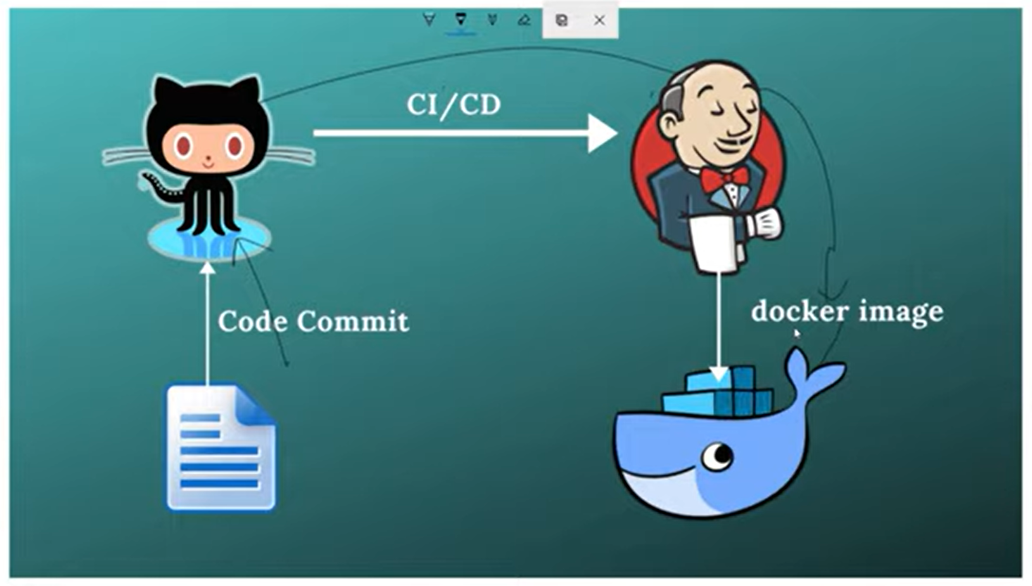
To check the build details message, click on output console.



So, this is how to integrate GitHub to Continuous Integration tool Jenkins and how we can automate the build.

So we can see our Build also created and it’s executed Testcase also.

**Building Docker Images using Jenkins step by step | Devops Integration Live Demo |**



for push and pull Docker Images In Docker Hub

---------------------------------------------

open cmd

>> docker

>> docker images

>> docker login (Enter uname(kaushal8707) & pwd for Docker Hub)

++ create a repository name in Docker Hub before pushing your images to docker hub ( Repo : selfdeveloped)

++ Tag your docker image to docker repository before pushing

>> docker tag spring-boot-dockerize.jar:latest kaushal8707/selfdeveloped:spring-boot-dockerize.jar

++ push docker image to Docker Hub

>>docker push kaushal8707/selfdeveloped:spring-boot-dockerize.jar

++ pull Docker Image from Docker Hub

>> docker pull kaushal8707/selfdeveloped:spring-boot-dockerize.jar

check the list of images

>> docker image ls

run docker image as a docker container

>> docker run -p 9090:8080 spring-docer-mayyapp.jar ( where myapp.jar is our application jar 9090 out of container to access app and 8080 we have mentioned to expose in docker file)

after deploy hit the rest endpoint url to access application rest API

create a docker images withdocker file commands

---------------------------------------------------

>> docker build -t spring-docer-mayyapp.jar .

>> docker image ls

>> docker run -p 9090:8080 spring-docer-mayyapp.jar

create a docker images without docker file commands

---------------------------------------------------

spring In built provides the cloud native buildpacks after 2.3.1 RELEASE version

++ to create a docker image

>> mvn spring-boot:build-image

>> docker images

>> docker run -p 8081:8081 spring-docer-mayyapp.jar