**Hand-on Lab CI Pipeline setup**

Contents

[Logon to Jenkins 2](#_Toc22052264)

[Create a new pipeline job 2](#_Toc22052265)

[Adding ad-hoc parameters to pipeline 3](#_Toc22052266)

[Adding username as parameter 3](#_Toc22052267)

[Generating sonarqube for build token 4](#_Toc22052268)

[Adding Sonarqube CLI as parameter in pipeline 6](#_Toc22052269)

[Configuring Pipeline 7](#_Toc22052270)

[Explaining the Jenkinsfile 8](#_Toc22052271)

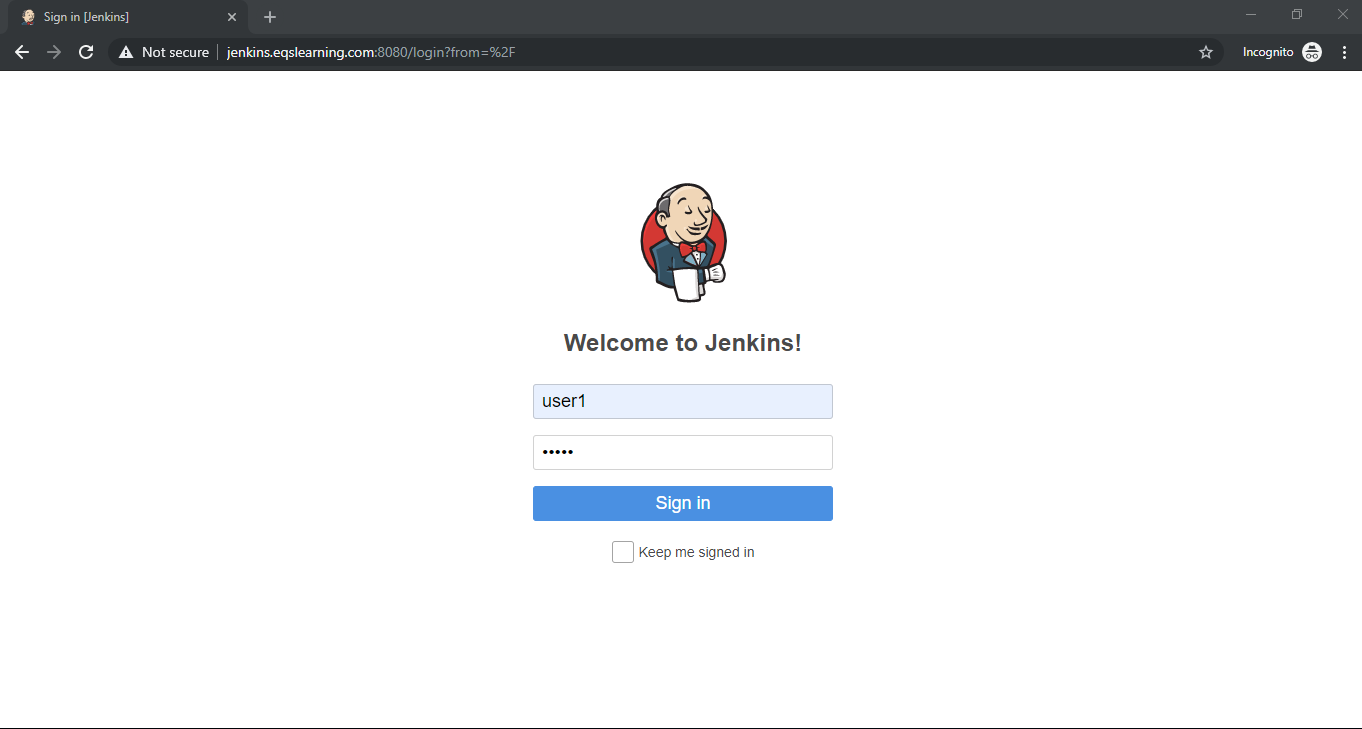
[Triggering the Jenkins pipeline 9](#_Toc22052272)

[Verify the Docker image in registry 11](#_Toc22052273)

# Logon to Jenkins

Logon to Jenkins with below URL address with user id shared to you. E.g : user1. Password would same as username.

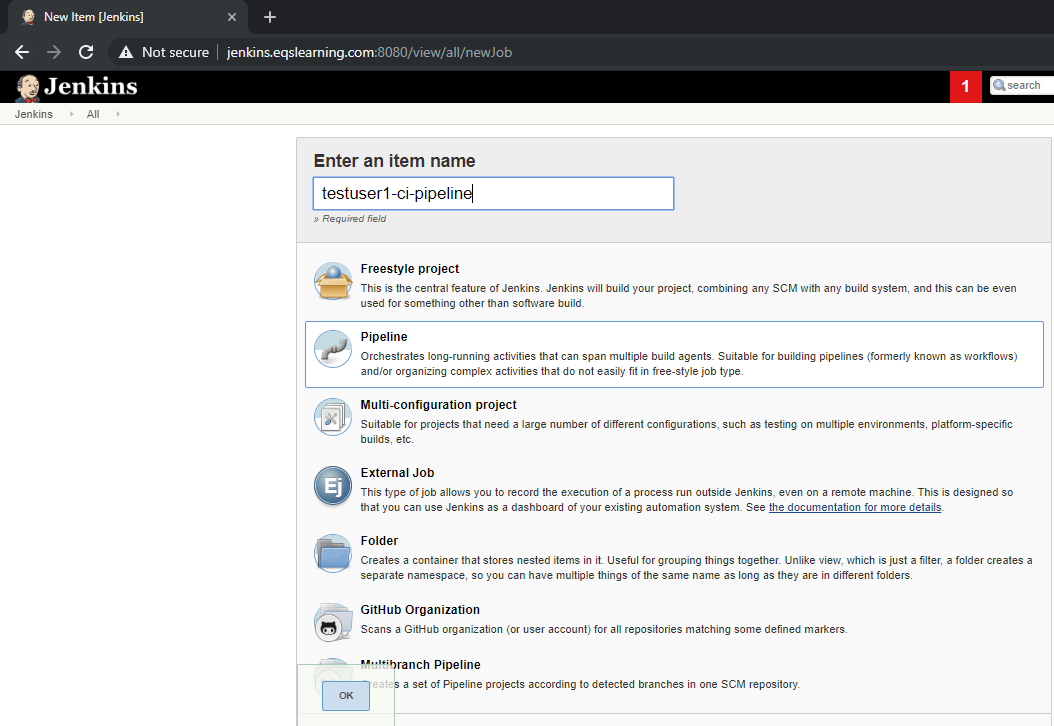
[http://jenkins.eqslearning.com:8080](http://jenkins.eqslearning.com:8080/)



# Create a new pipeline job

Click on **New Item** option on the left menu bar to create new Jenkins Job. Select job type as **Pipeline** andEnter the job name and click **OK** button on the bottom of the page.

Note: For convenience of others, kindly use the naming convention as **<username>-ci-handson**



# Adding ad-hoc parameters to pipeline

Since we are using common git repo to create multiple pipeline, we need to separate the sonarqube setup for each user. Hence we are passing the username and sonarqube cli as parameters.

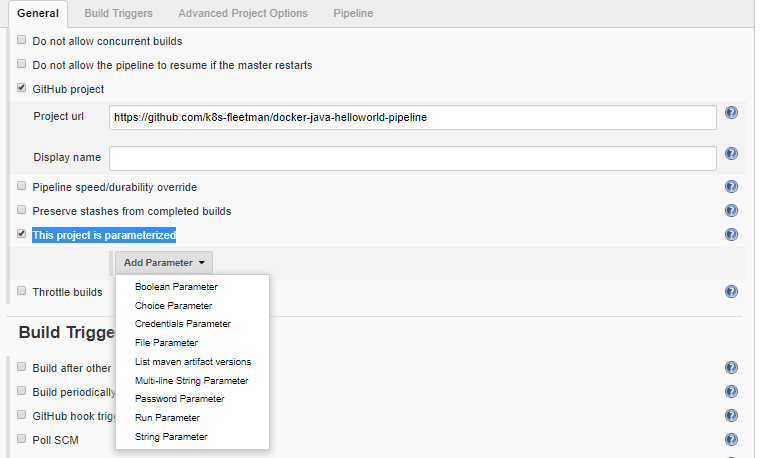
On newly created pipeline, update the below GitHub project URL.

https://github.com/k8s-fleetman/docker-java-helloworld-pipeline

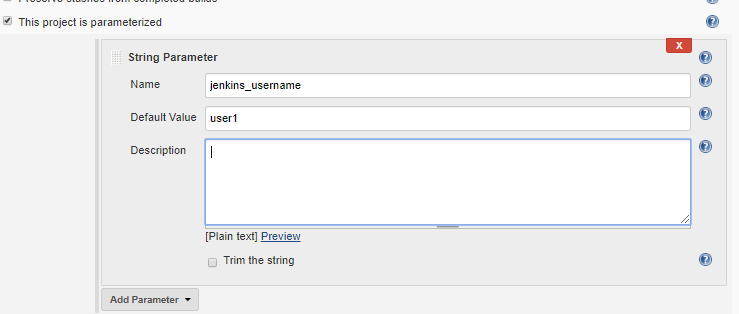
## Adding username as parameter

To build separate Docker image for each user, we will be creating a parameter which will be used in pipeline for building the Docker container.

On General section of the newly created pipeline, check the option “**This project is parameterized**” and click on option **Add parameter** and select **String parameter**.

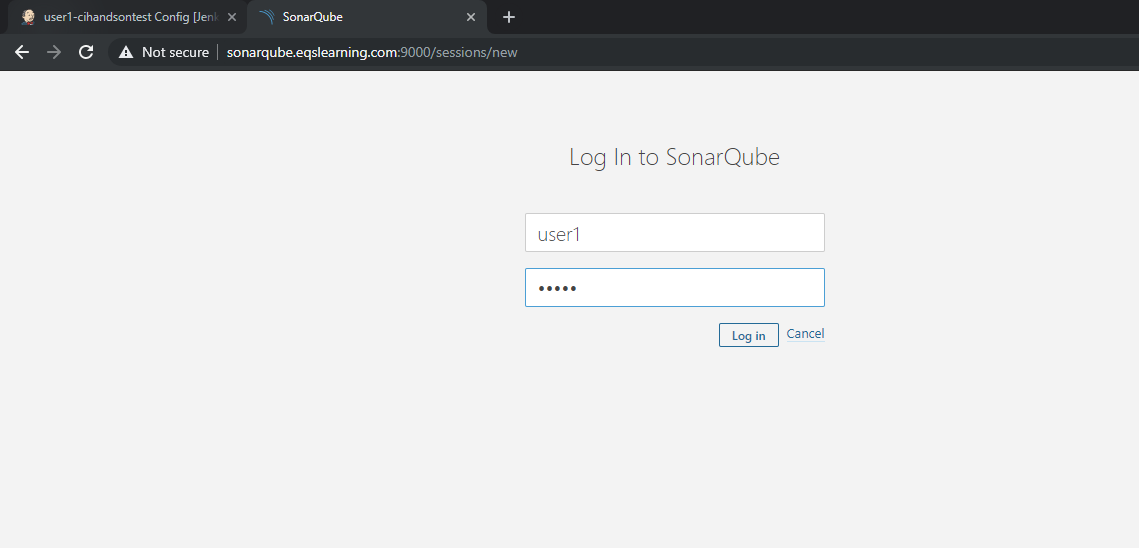


Enter the parameter name as **jenkins\_username** and enter the default value as your username.

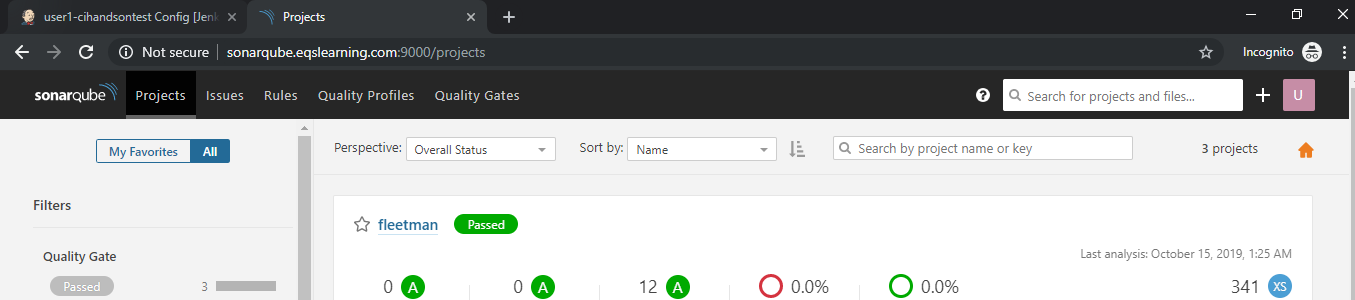


# Generating sonarqube for build token

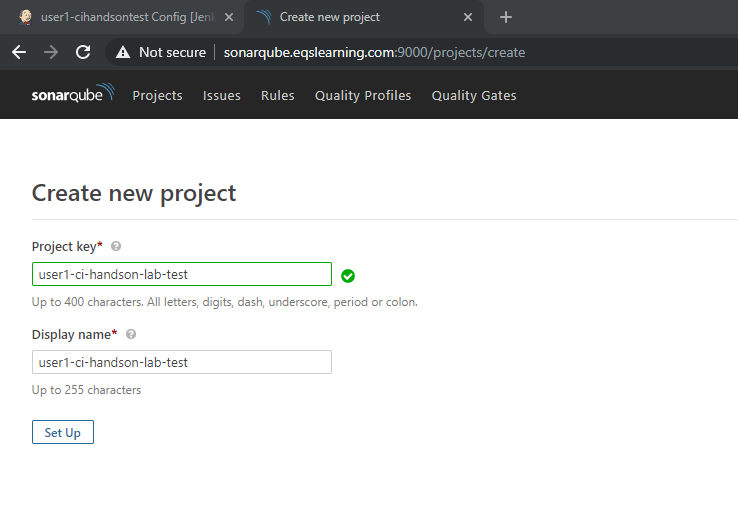
Logon to sonarqube in a seprate tab/page using URL [http://sonarqube.eqslearning.com:9000](http://sonarqube.eqslearning.com:9000/) with your user name and credential which was provided for Jenkins.



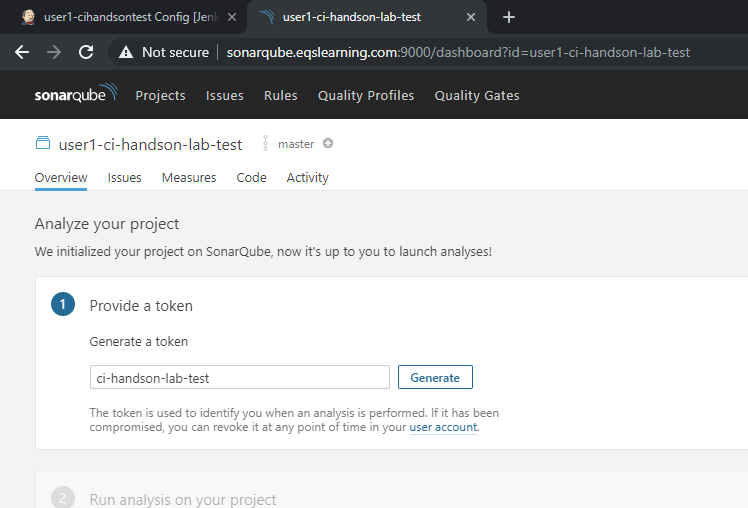
After logon, click on + symbol on the right top corner and select the option “**create new project**”.



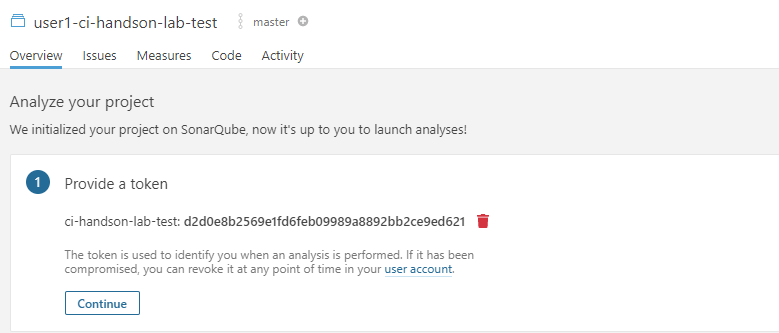
Enter the appropriate name for the project with naming convention and add a description for the project. Click on Setup to get the access token.



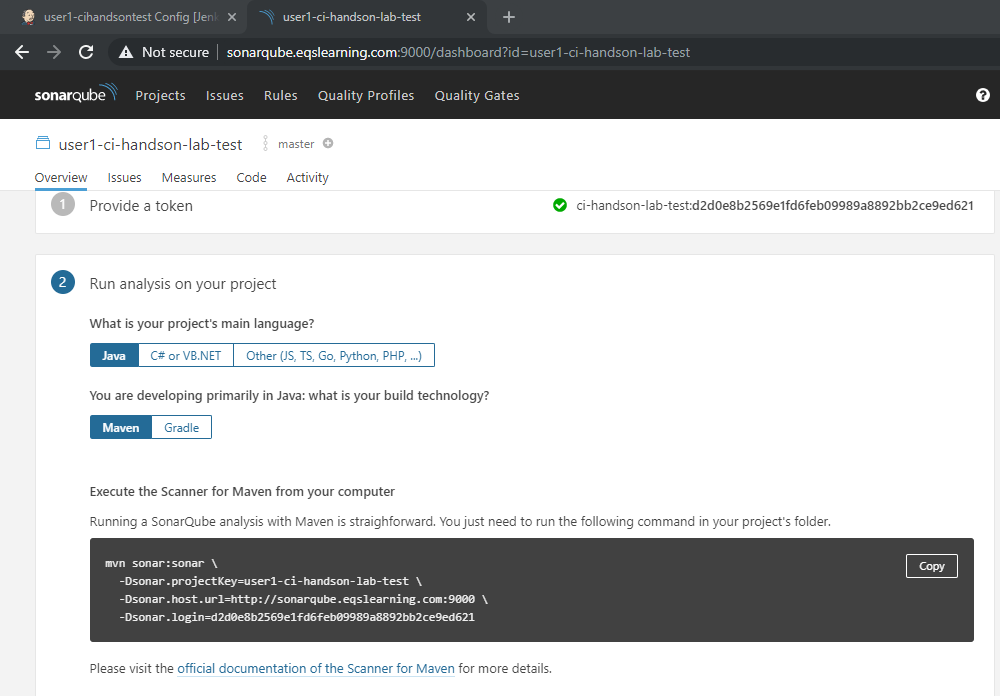
Enter a description to get a new token for the project and click on generate button.



Upon new key generated, click on Continue to get command which we need to execute in our Jenkins build.



Copy the code generated in sonarqube for maven based build on Java language as shown below.

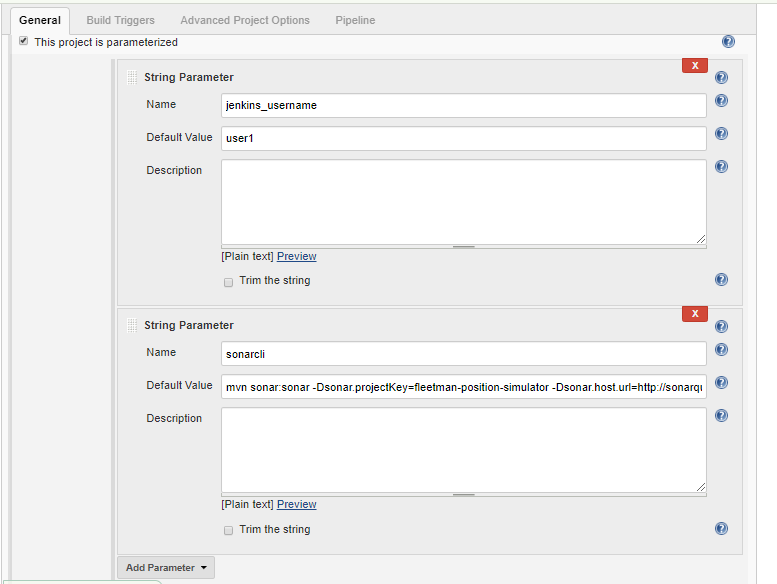


Copy the sonarqube command with toke details and save it for updating in Jenkins job.

## Adding Sonarqube CLI as parameter in pipeline

In Jenkins, Click on Add parameters once again and enter the newly generated CLI which was created as mentioned in previous section. Enter the name of the parameter as ‘**sonarcli’** and copy the cli which was copied from sonarqube to default value field.

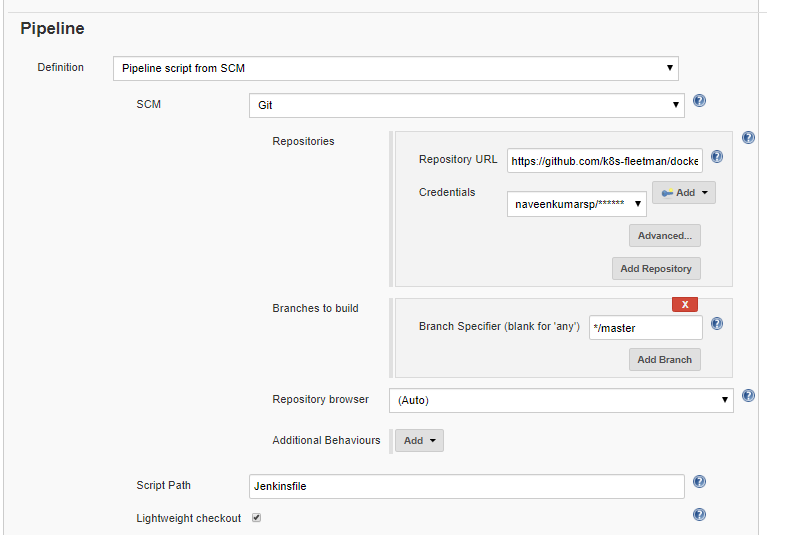
Note: make sure you remove the multiline separator i.e ‘\’ from cli before updating to default value field.



# Configuring Pipeline

On newly created pipeline, Go to Pipeline section and update the Definition field to “**Pipeline Scripte from SCM**”. Select the SCM as ‘**Git**’ and update the Repository URL as listed below as well as credentials. Up on all updates listed above, click on **Apply/save** button.

<https://github.com/k8s-fleetman/docker-java-helloworld-pipeline.git>



# Jenkinsfile - Explained

For versioning purpose, Typically Jenkins file will be kept in code repository. We have written a simple Jenkinsfile which will define the multiple build stages/workflows which get executed as part of build setup. Here is the pipeline which is listed in our GitHub repo which will trigger the jobs in sequential order.

pipeline {

agent any

environment {

// You must set the following environment variables

// ORGANIZATION\_NAME

// YOUR\_DOCKERHUB\_USERNAME (it doesn't matter if you don't have one)

SERVICE\_NAME = "docker-java-helloworld-pipeline"

IMAGE\_NAME = "ci-pipeline-demo-${jenkins\_username}"

REPOSITORY\_TAG="${DOCKERHUB\_URL}/${IMAGE\_NAME}:${BUILD\_ID}"

}

stages {

stage('Preparation') {

steps {

cleanWs()

git credentialsId: 'GitHub', url: "https://github.com/${ORGANIZATION\_NAME}/${SERVICE\_NAME}"

}

}

stage('Build') {

steps {

sh '''mvn clean install package'''

}

}

stage('SonarQube') {

steps {

sh '${sonarcli}'

}

}

stage('Build Image') {

steps {

sh 'scp -r ${WORKSPACE} jenkins@${DOCKER\_HOST\_IP}:/home/jenkins/docker/${BUILD\_ID}'

sh 'ssh jenkins@${DOCKER\_HOST\_IP} docker image build -t ${REPOSITORY\_TAG} /home/jenkins/docker/${BUILD\_ID}'

sh 'ssh jenkins@${DOCKER\_HOST\_IP} docker image ls'

sh 'ssh jenkins@${DOCKER\_HOST\_IP} rm -rf /home/jenkins/docker/${BUILD\_ID}'

}

}

stage('Push Image to repo') {

steps {

sh 'ssh jenkins@${DOCKER\_HOST\_IP} docker push ${REPOSITORY\_TAG}'

}

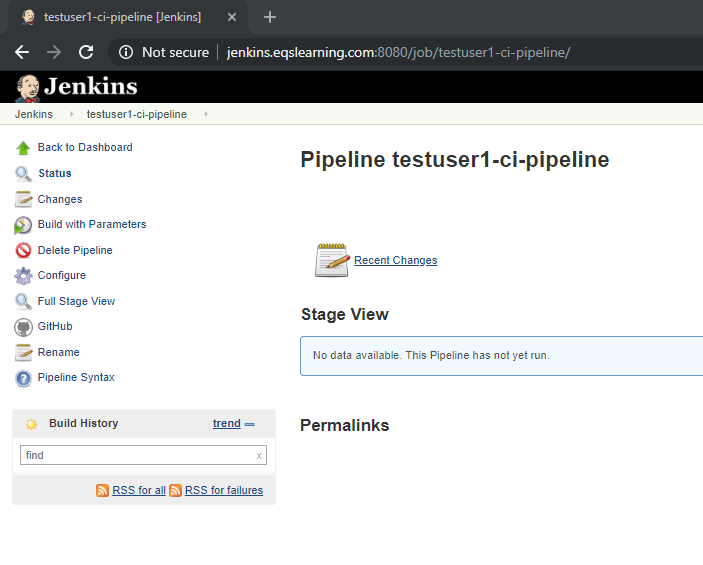
}

}

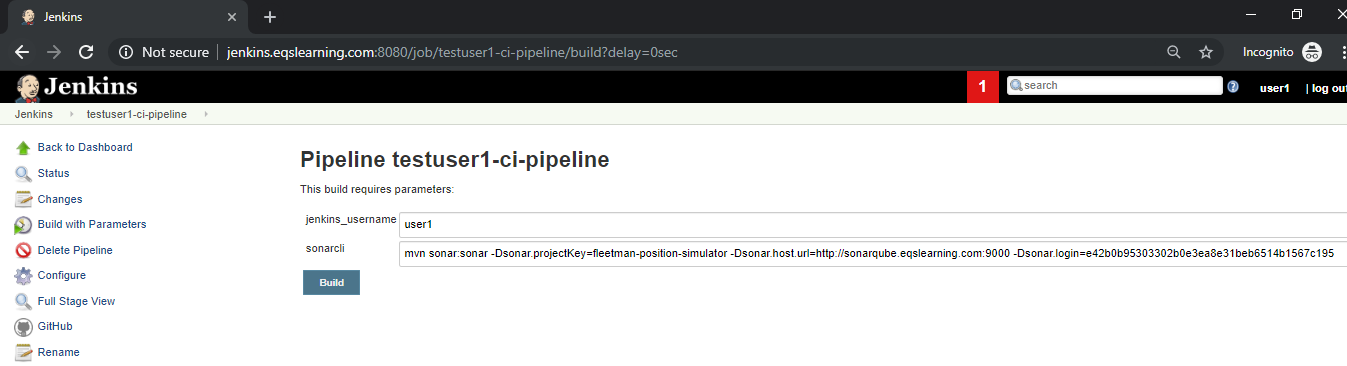
}

# Triggering the Jenkins pipeline

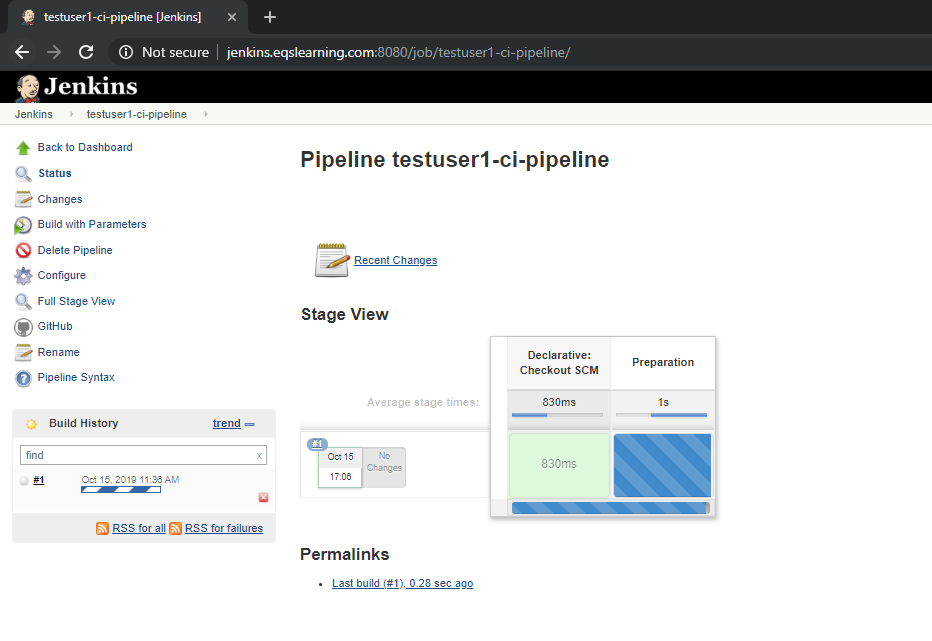
In order to trigger the pipeline, navigate to your pipeline and click on “**Build with parameters**” listed on left side menu of the page.



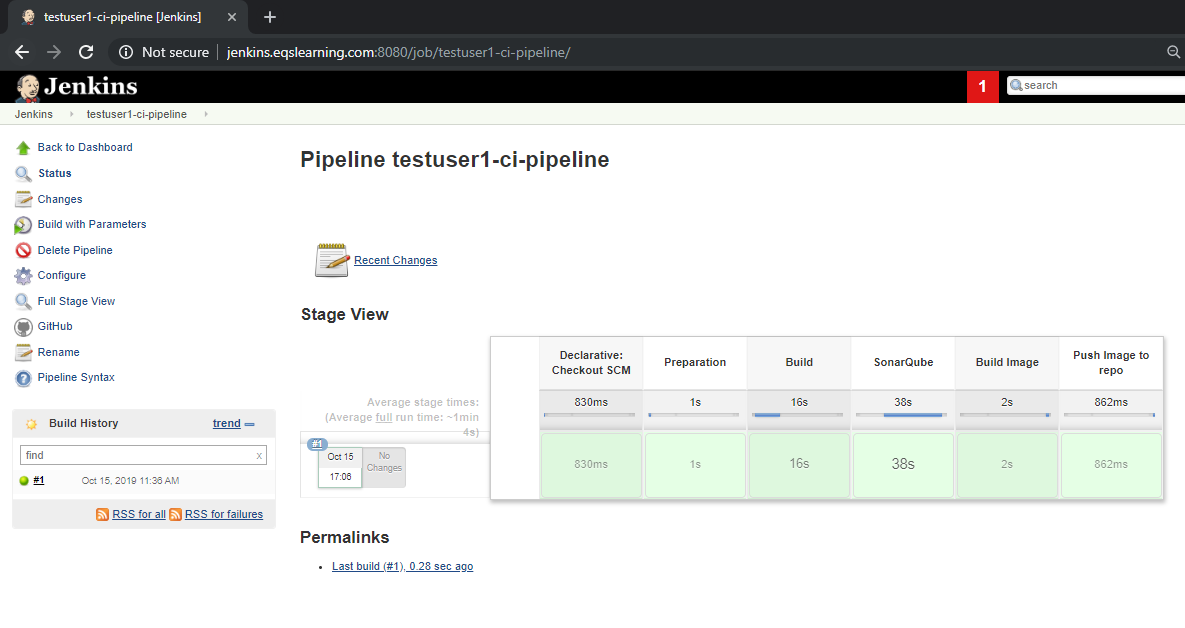
Parameter page lists all the default value which you have updated. Validate the value and click on build to trigger the pipeline.



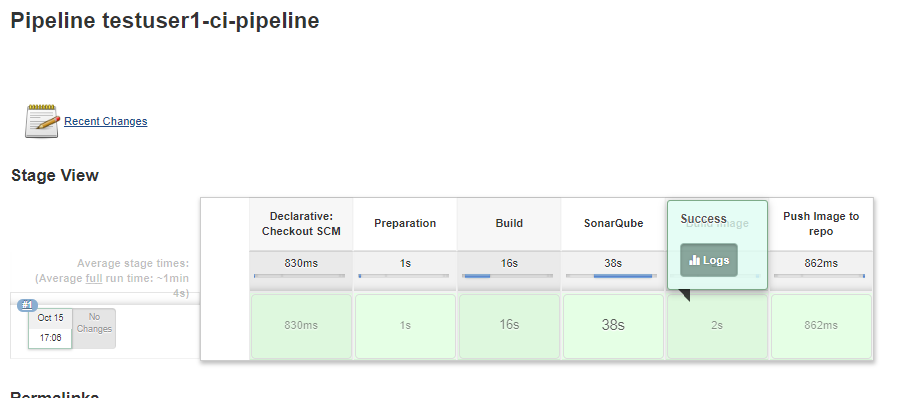
After couple of seconds, you will be able to see the pipeline stage view get updated and you will be able to see the progress. For the first time you may notice the build will create the stages one after the another.

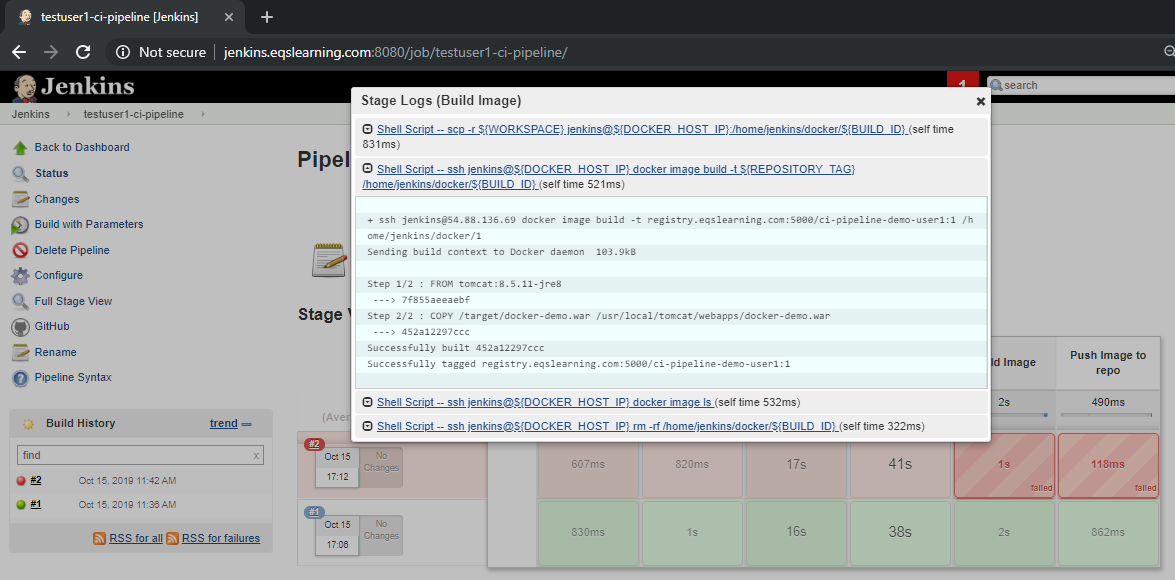


Up on completion of the Jobs, you may see the end to end pipeline setup.



You may review the logs on each stage and verify the entire process. It will be really helpful to review failures and validate the build workflow.





# Verify the Docker image in registry

As part of our pipeline, we will be building the application and validating the code quality. If the build completes these two stages successfully, it builds a Docker image and pushes it to private registry for deployment. You may verify the docker image listed in registry by accessing the URL [http://docker.eqslearning.com](http://docker.eqslearning.com/)

