JENKINS

JENKINS

- Self-contained, open source automation server
- Automate all sorts of tasks related to building, testing, and delivering or deploying software
- Continuous integration and continuous delivery tool
- Written in Java
- Forked from Hudson
- Platform independent
- Rich set of plugins(Over 1000 plugins)
- Easily configurable
- Easy to create new Jenkins plugin if one is not available

BENEFITS

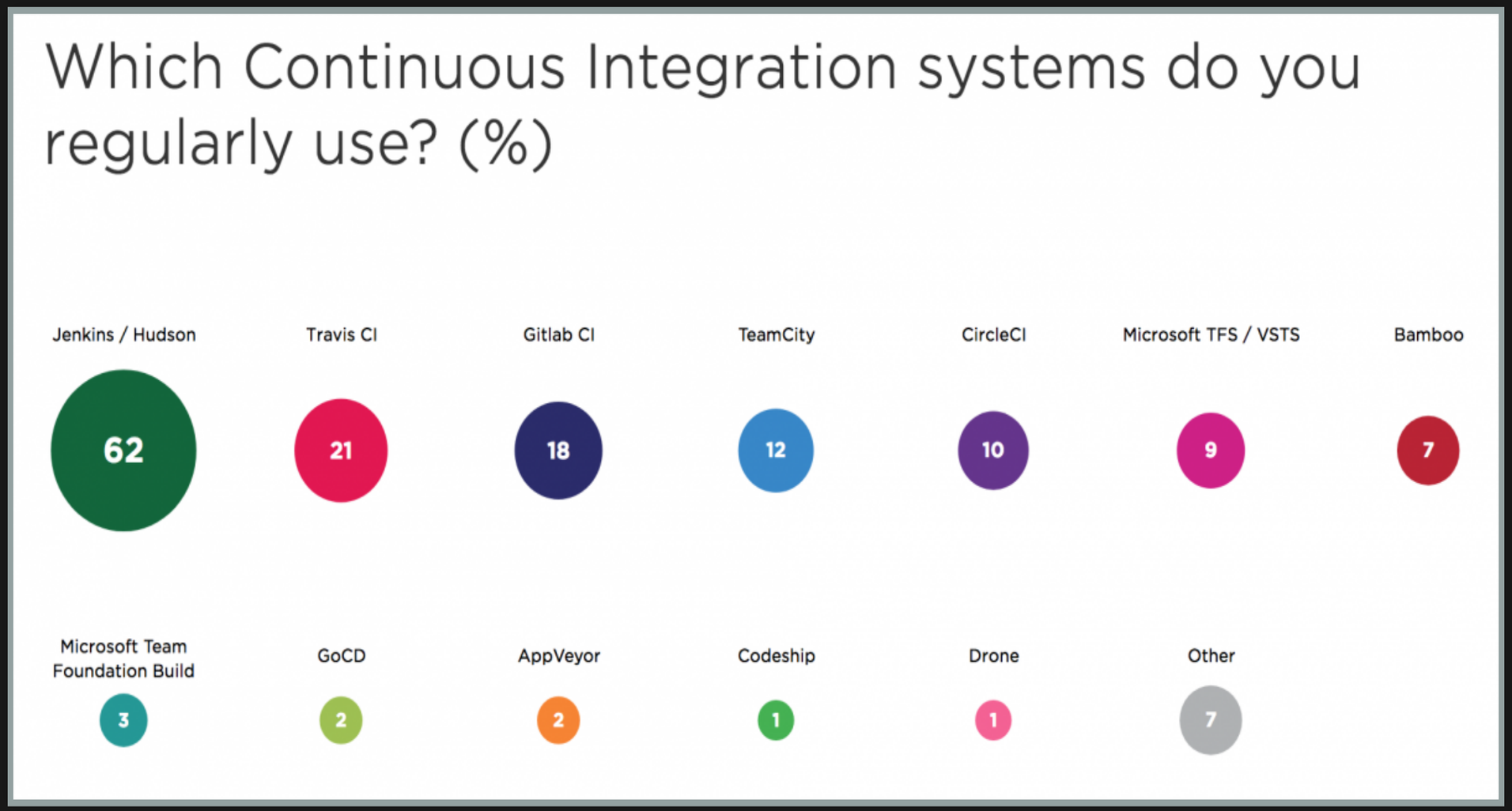
- Developer time is concentrated on work that matters making development faster
- Issues are detected and resolved almost right away which keeps the software in a state - where it can be released at any time safely.
- Complete history maintained
- Write Pipeline once, use many times
- Deployment made easy
- Improves Software development process
- Almost all sorts of plugins(Github, Gitlab, Bitbucket, etc)

JENKINS TYPICAL WORKFLOW

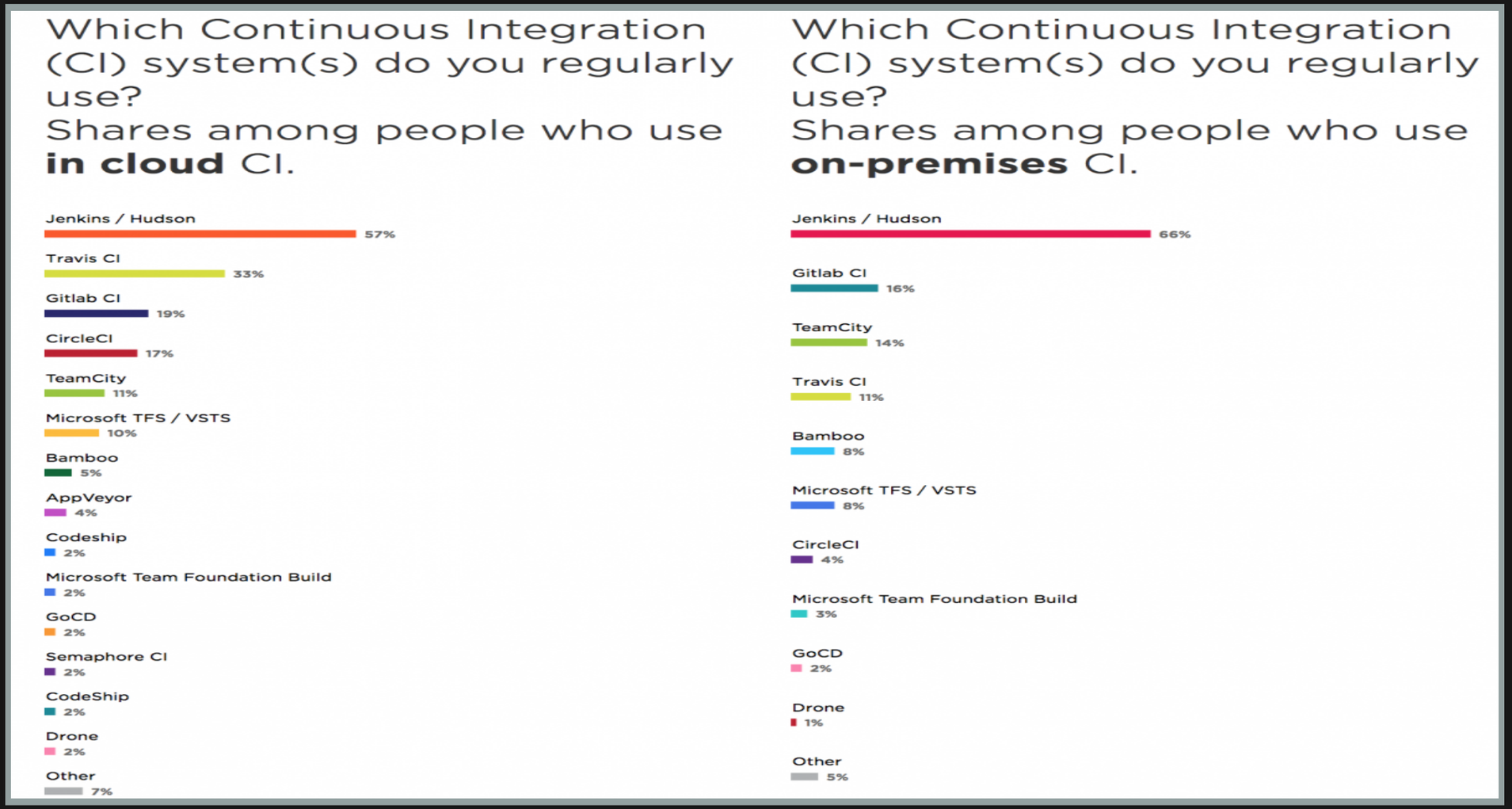
- Code committed in PR
- Build triggered through Webhook
- Pipeline Steps:
  - Get source code from repo
  - Build & Test the code
  - Build & Push Docker Image
  - Generate Report
  - Deploy in Dev/Stage environment(Optional)
  - Continuous Deployment (Optional)
  - Notify teams especially in Failure

JENKINS ADOPTION

From https://dzone.com/articles/jenkins-is-showing-the-cicd-way

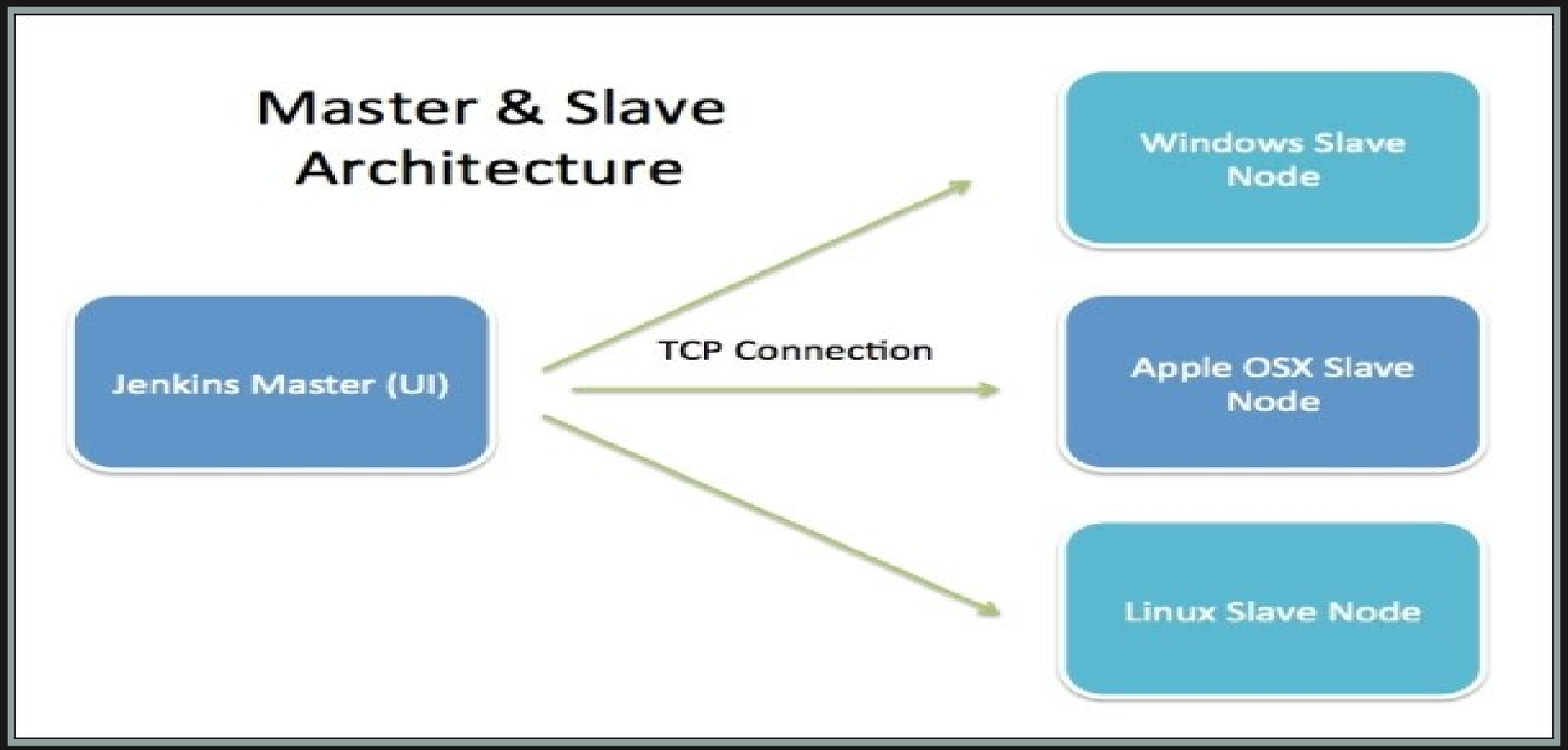


ON-PREMISE VS CLOUD ADOPTION



JENKINS ARCHITECTURE

- Master and slave architecture
- Can have single master and multiple slaves with different environment to test app



JENKINS TERMINONOLOGIES

- **Job**: A unit of work
- **Master**: central & main unit, Does job scheduling
- **Slave**: Execute one or more jobs
- **Plugin**: Common workflows merged into one
- **View**: Different Jobs can be combined in a single view

LAB - INSTALL JENKINS

- Stand alone application
- Docker
- Package

STAND ALONE APPLICATION

Jenkins comes with an embedded Winstone server; therefore enabling it to run as a standalone application.

Create directory for jenkins stand alone application

```
mkdir jenkins-standalone
cd jenkins-standalone
```

Download Jenkins WAR file to jenkins-standalone directory on your machine

To run Jenkins as stand alone application Java is required. Open terminal and check if Java is installed by run the following command

```
java -version
```

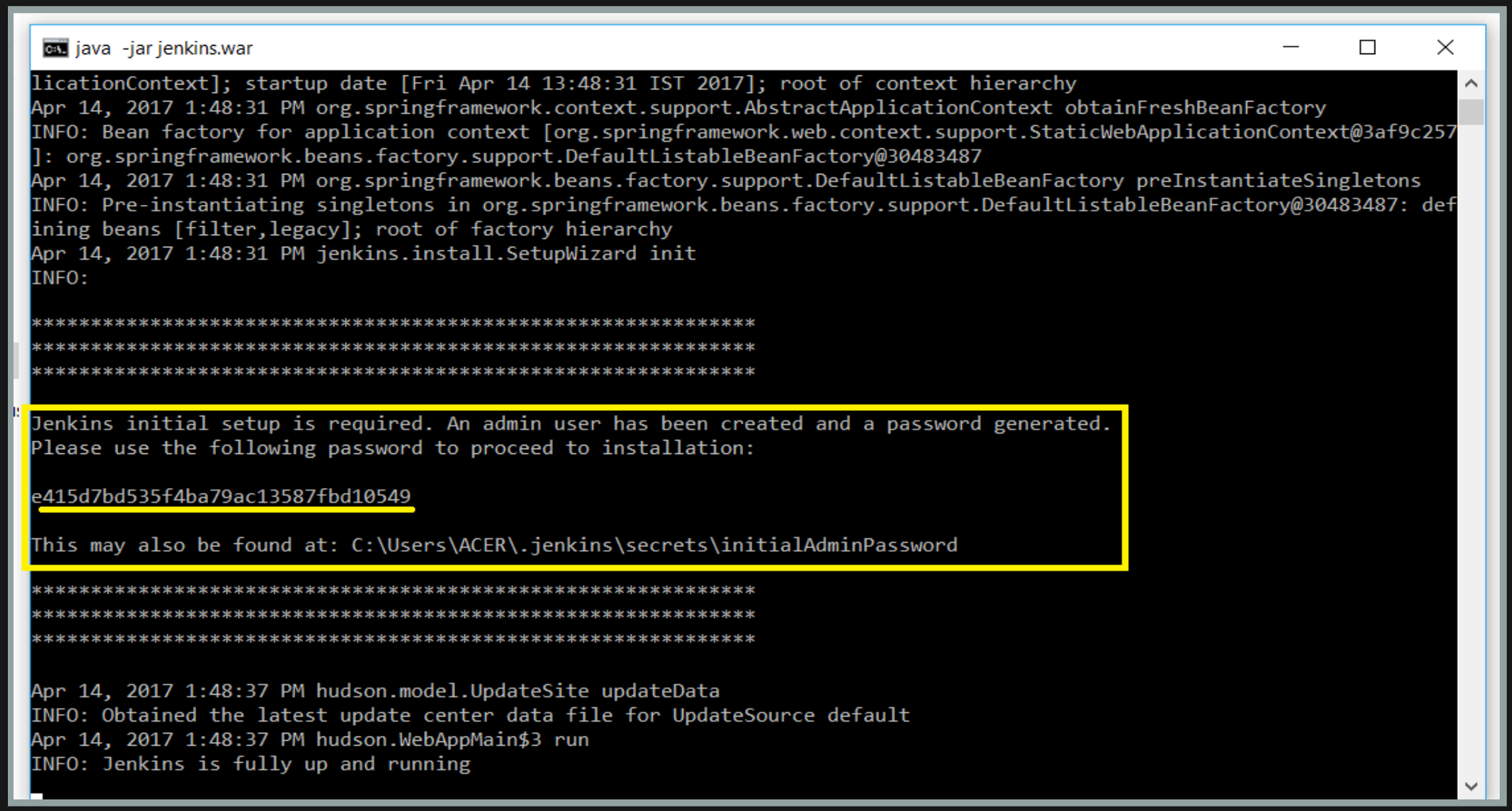
If Java is not installed, you can install Java by running the following command

```
sudo apt-get install openjdk-8-jdk -y
```

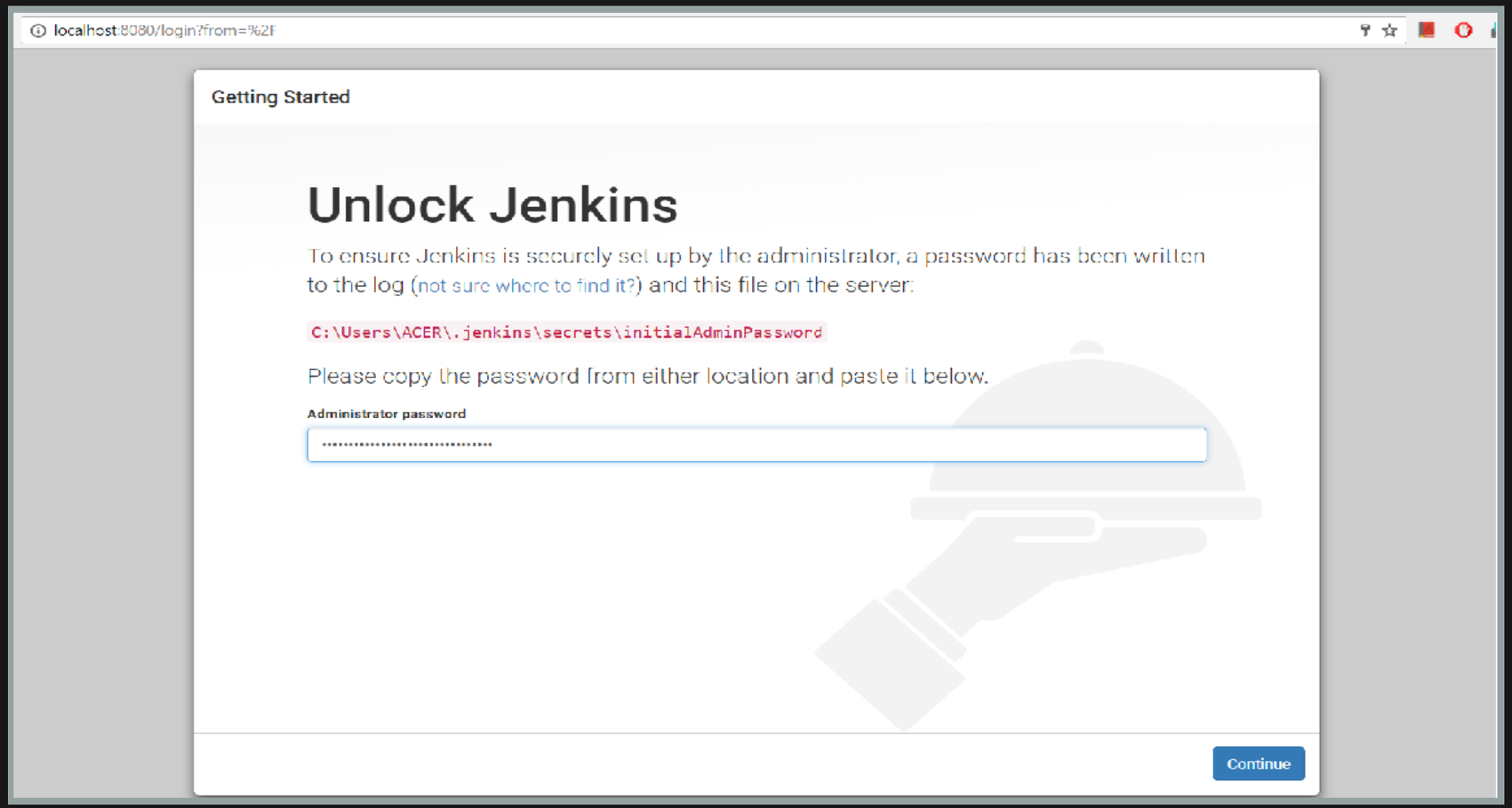
Run Jenkins

```
java -jar jenkins.war
```

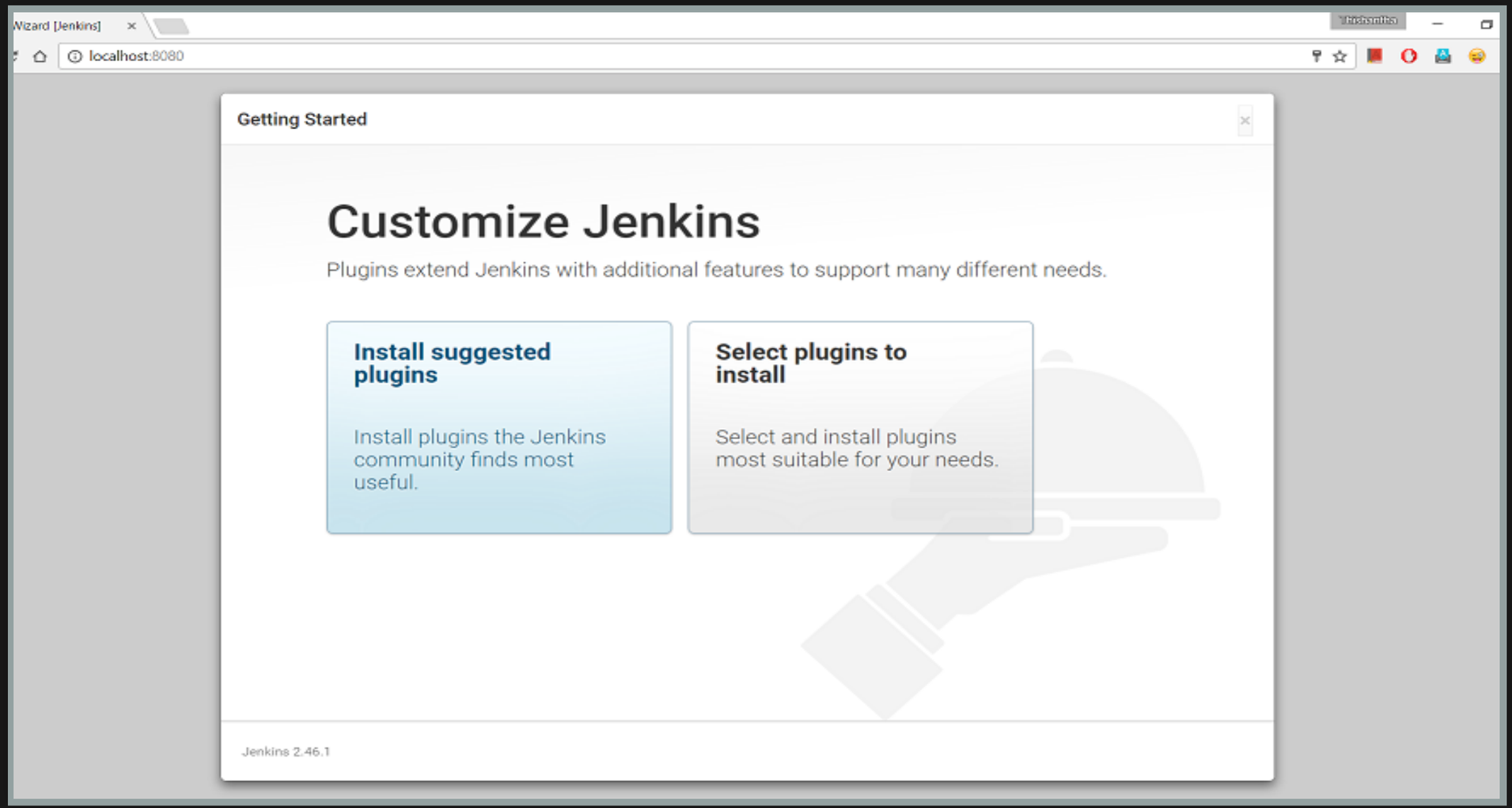
Now you should see the initial setups being run in the command line. The setup will generate a random password for you to use on the initial login. Note this one down.



Wait for the console output to print 'Jenkins is fully up and running'. Now go to your browser and type the url: http://localhost:8080/ and you should see a screen as shown below. Enter the initial administrator password that was generated in the previous step. Input the password and click 'Continue'.



In the next screen you are asked to install necessary plugins for Jenkins. This enables the integration of various platforms with Jenkins. Select 'Install selected plugins'. Wait for the plugin installation to complete



localhost:8080

Getting Started

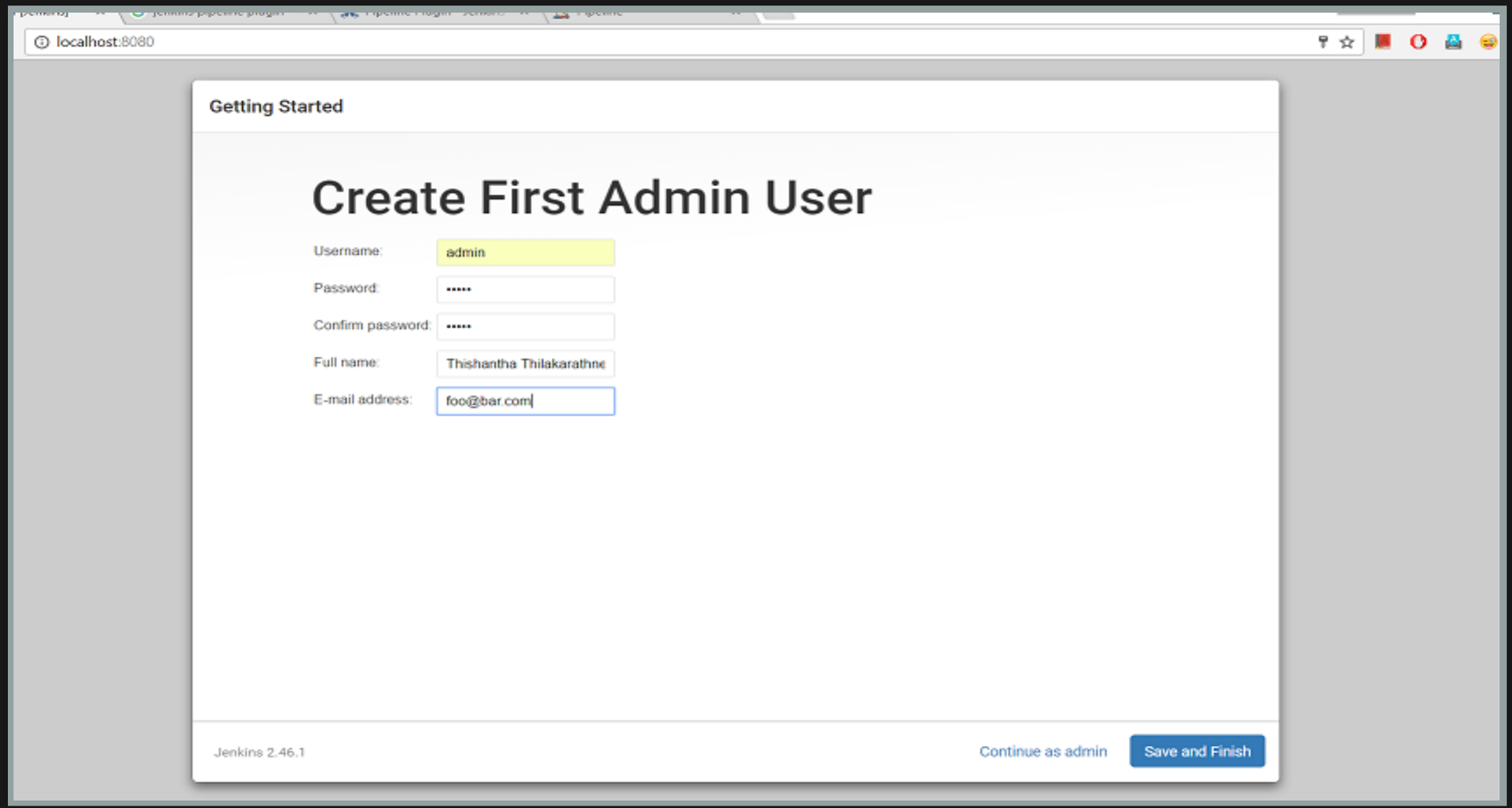
Getting Started

Folders Plugin	OWASP Markup Formatter Plugin	build timeout plugin	Credentials Binding Plugin	** bouncycastle API Plugin
Timestamper	Workspace Cleanup Plugin	Ant Plugin	Gradle Plugin	
Pipeline	GitHub Organization Folder Plugin	Pipeline: Stage View Plugin	Git plugin	
Subversion Plug-in	SSH Slaves plugin	Matrix Authorization Strategy Plugin	PAM Authentication plugin	
LDAP Plugin	Email Extension Plugin	Mailer Plugin		

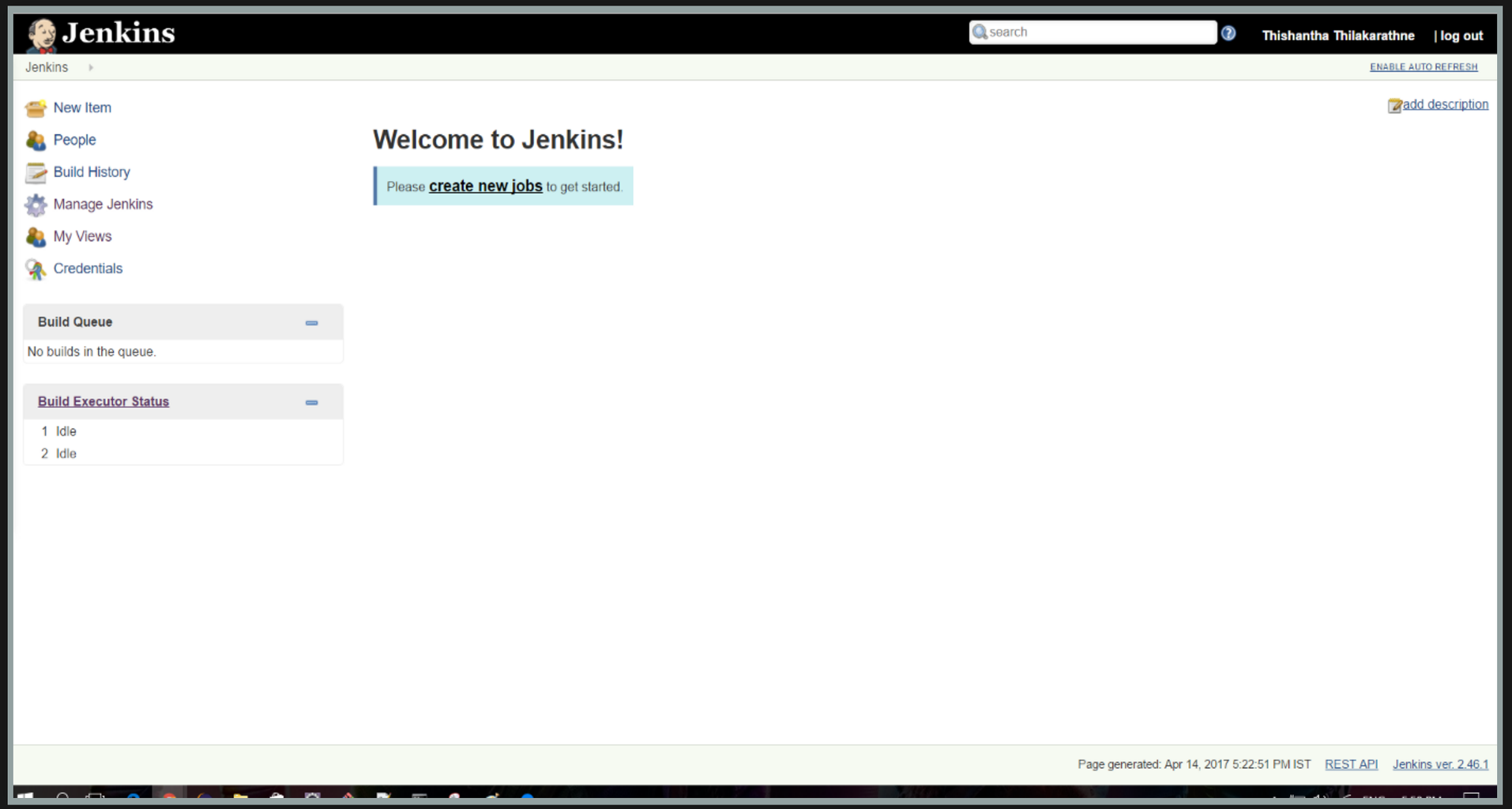
** - required dependency

Jenkins 2.46.1

Create an Admin user in the next screen once the plugin installation is complete.



Once finished, click Save and Finish and now proceed to the Jenkins dashboard where you can start creating jobs.



DOCKER

You can create a Jenkins container by running the following command. This will create a container named jenkins.

```
docker run -d -p 8081:8080 --name jenkins jenkins/jenkins
```

You can access Jenkins at http://localhost:8081. You can follow the same steps specified in Stand alone application section

PACKAGE

You will install Jenkins through apt by running the following command

```
wget -q -O - https://pkg.jenkins.io/debian/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.lis
sudo apt-get update
sudo apt-get install jenkins -y
```

Check the status of Jenkins by running the command

```
systemctl status jenkins
```

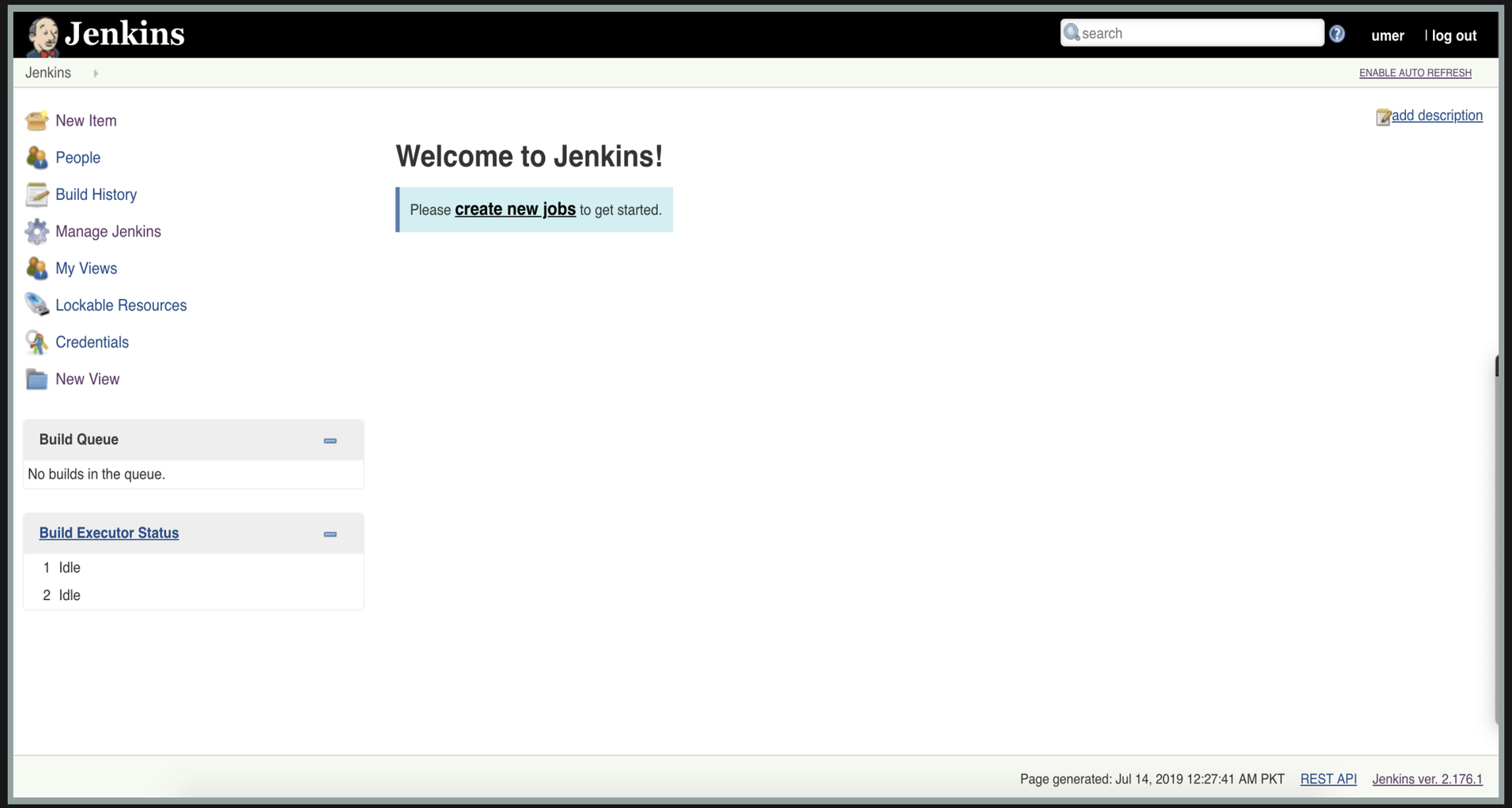
If jenkins is not running you can start Jenkins service by running the command

```
systemctl start jenkins
```

You can access Jenkins at http://localhost:8080. You can follow the same steps specified in Stand alone application section.

JENKINS UI

JENKINS UI


Jenkins
search
umer
| log out
Jenkins
ENABLE AUTO REFRESH
New Item
People
Build History
Manage Jenkins
My Views
Lockable Resources
Credentials
New View
add description
Welcome to Jenkins!
Please create new jobs to get started.
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Page generated: Jul 14, 2019 12:27:41 AM PKT
REST API
Jenkins ver. 2.176.1

JENKINS UI

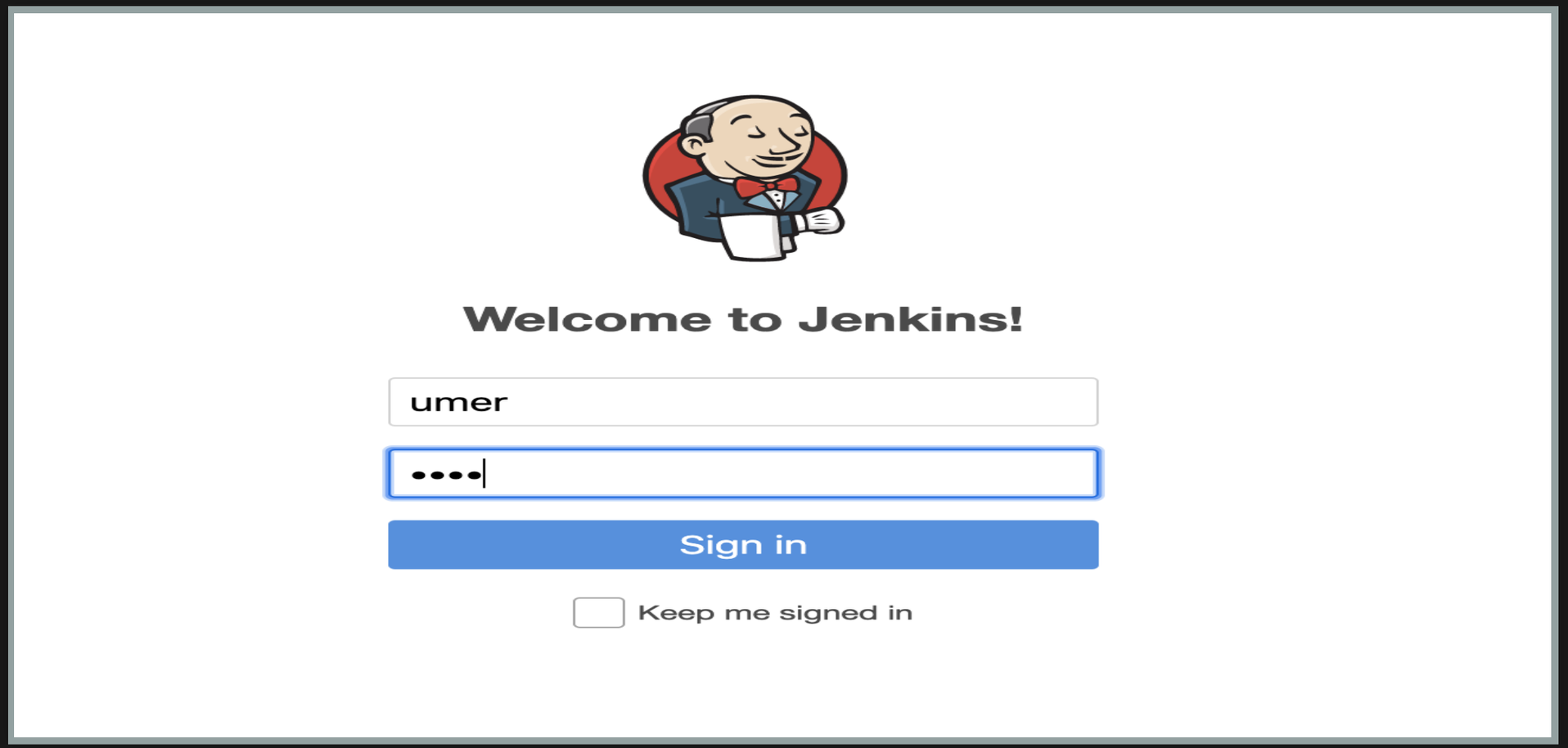
- **New Item**: To create a new job
- **People**: Manages users within Jenkins
- **Build History**: Shows history of builds
- **Manage Jenkins**: Used to configure Jenkins
- **My Views**: Private view for filtering Jenkins jobs
- **Credentials**: Manage global credentials
- **Build Queue**: Builds that have been triggered
- **Build Executor Status**: Status of running builds

LAB - FIRST JENKINS JOB

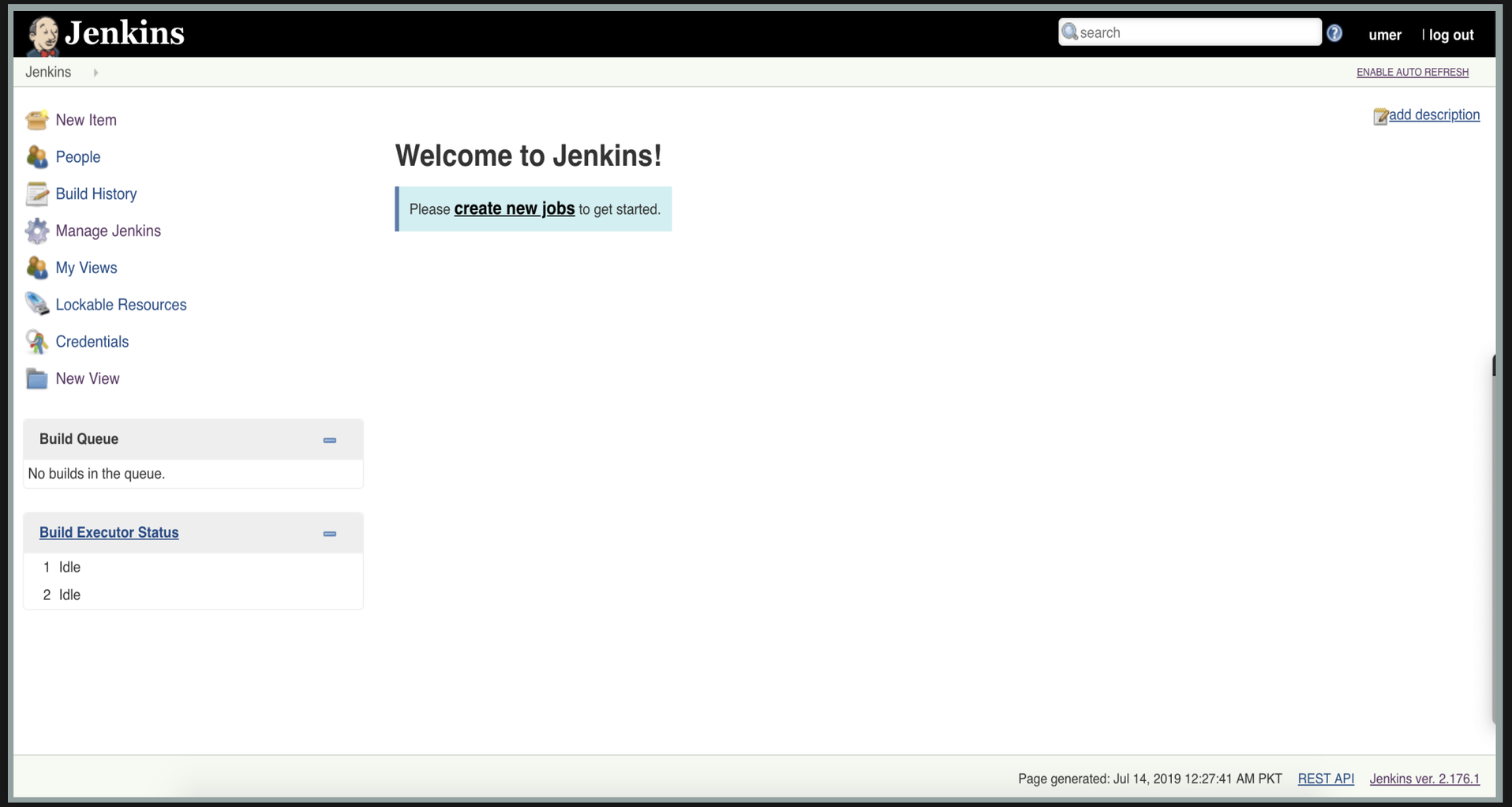
- Create First Job
- Edit Existing Job
- Run Shell Script
- Parameterized Job

CREATE FIRST JOB

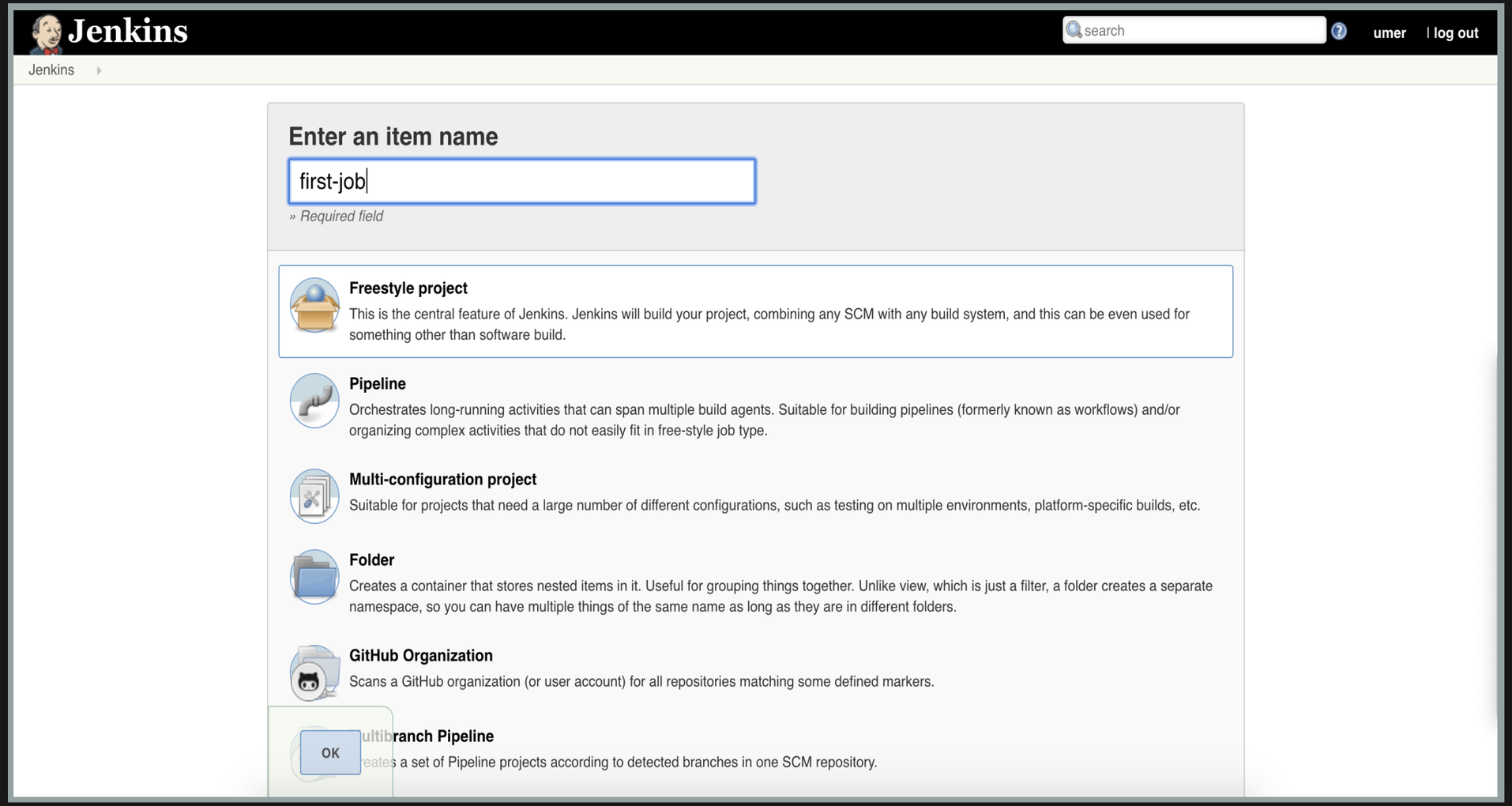
You are going to create your first Jenkins job. Go to your browser and type: http://localhost:8080/ and enter the admin credentials you created during configuration.



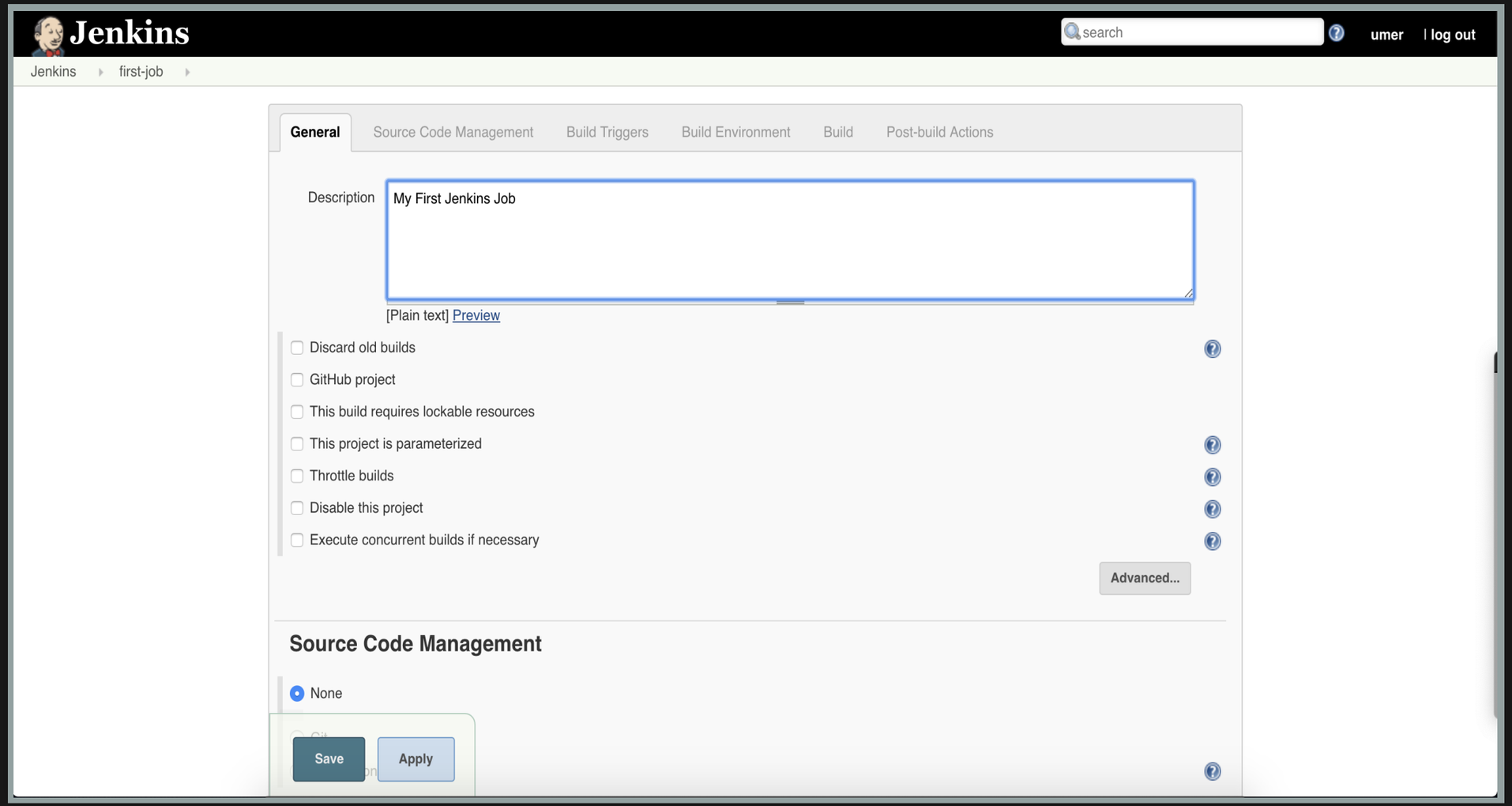
Once logged in, you will see the Jenkins dashboard shown below. Click New Item



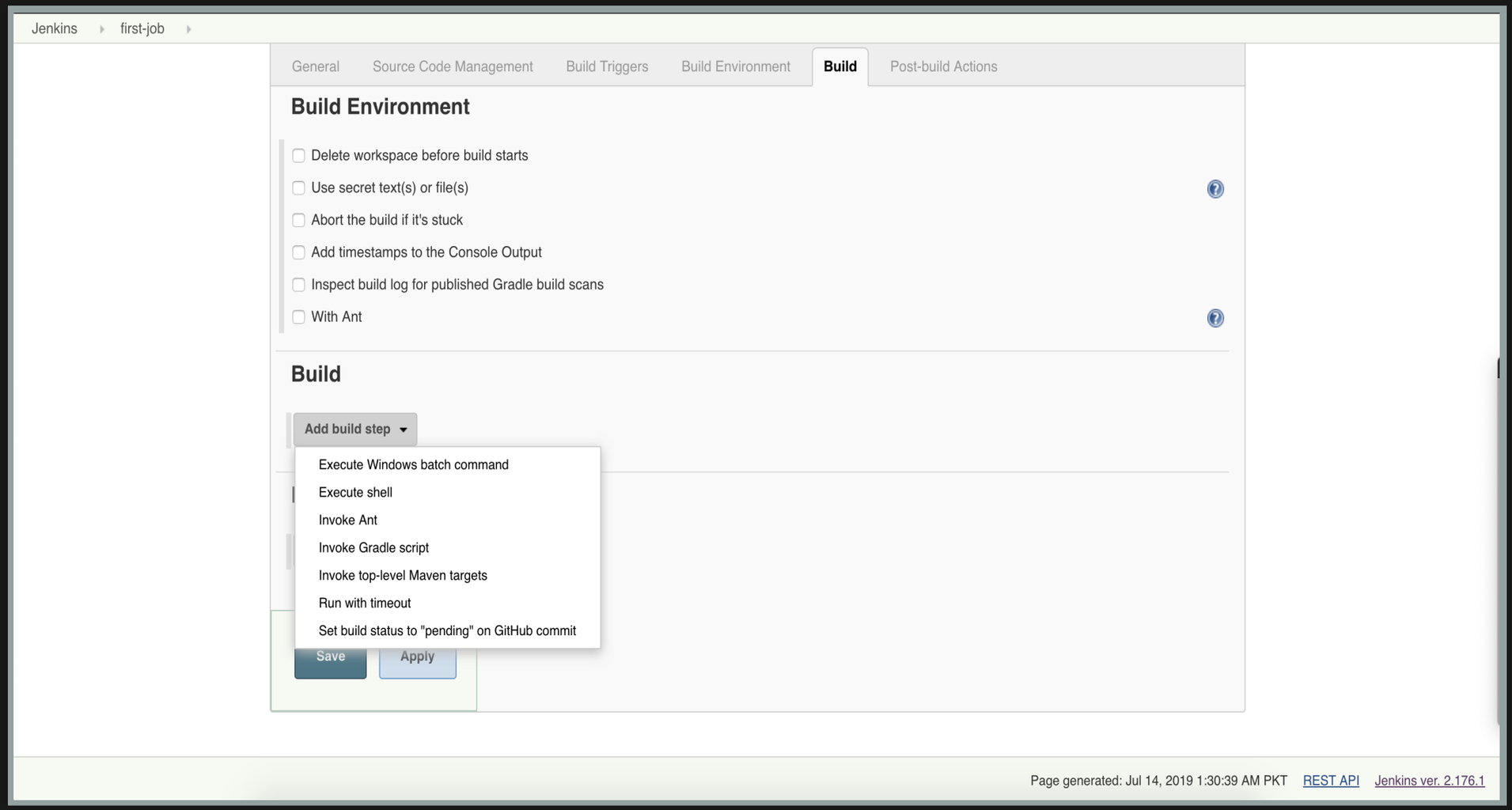
Enter name of your first job i.e. first-job as shown below, select Freestyle Project and click OK



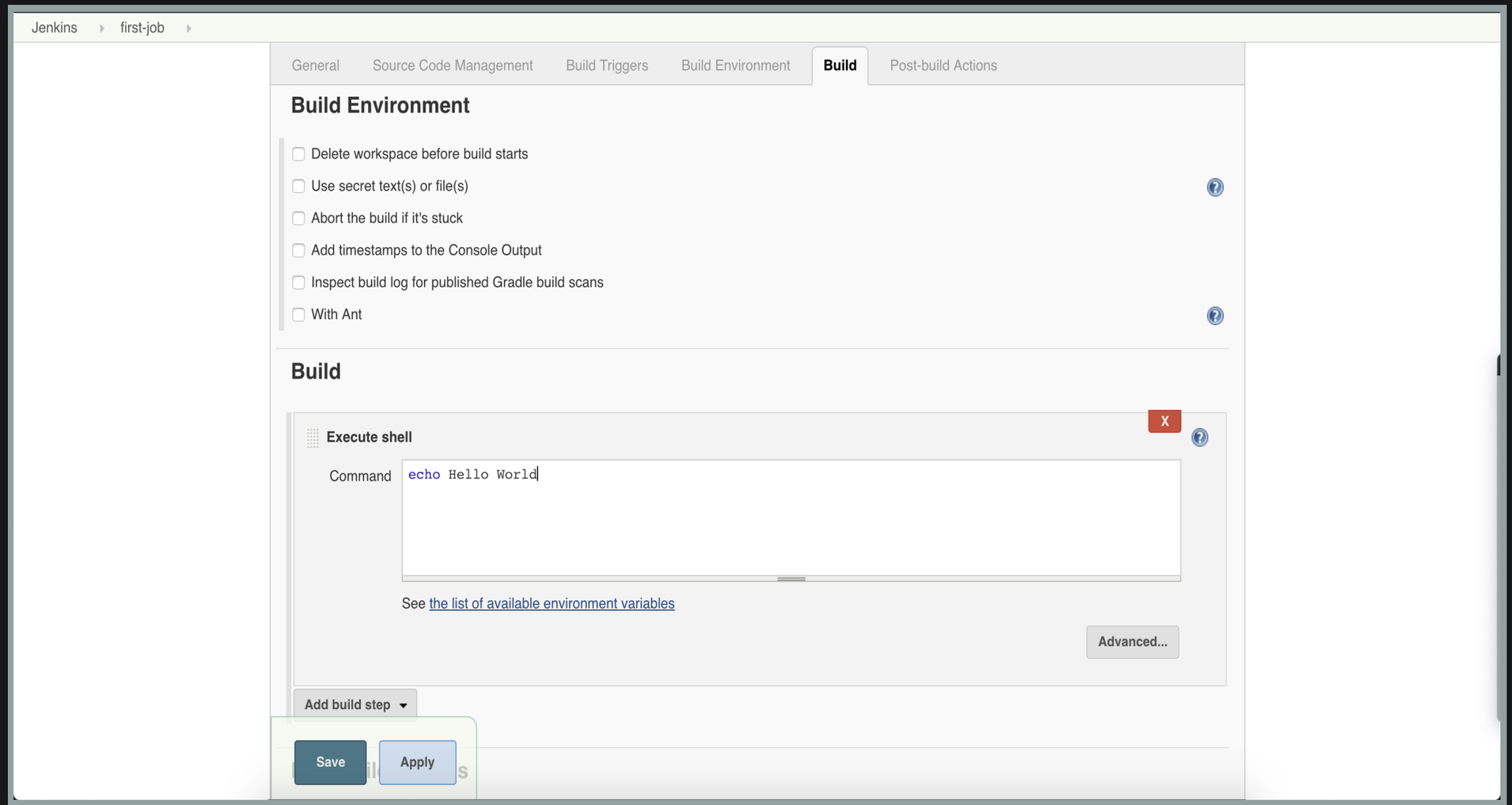
Next specify description as show below and click Build. This will take you to the Build Section.



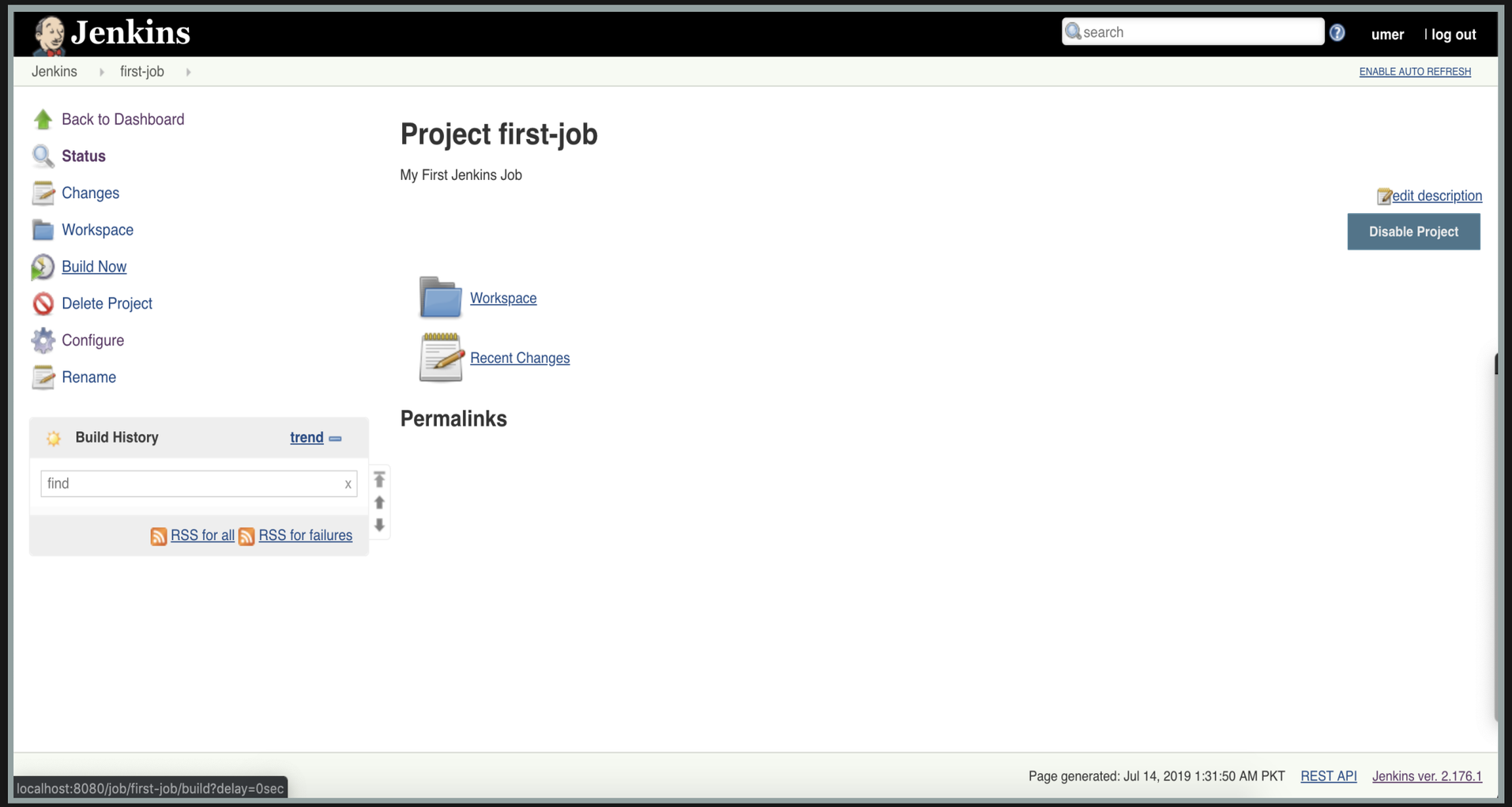
Select Execute shell as shown below.



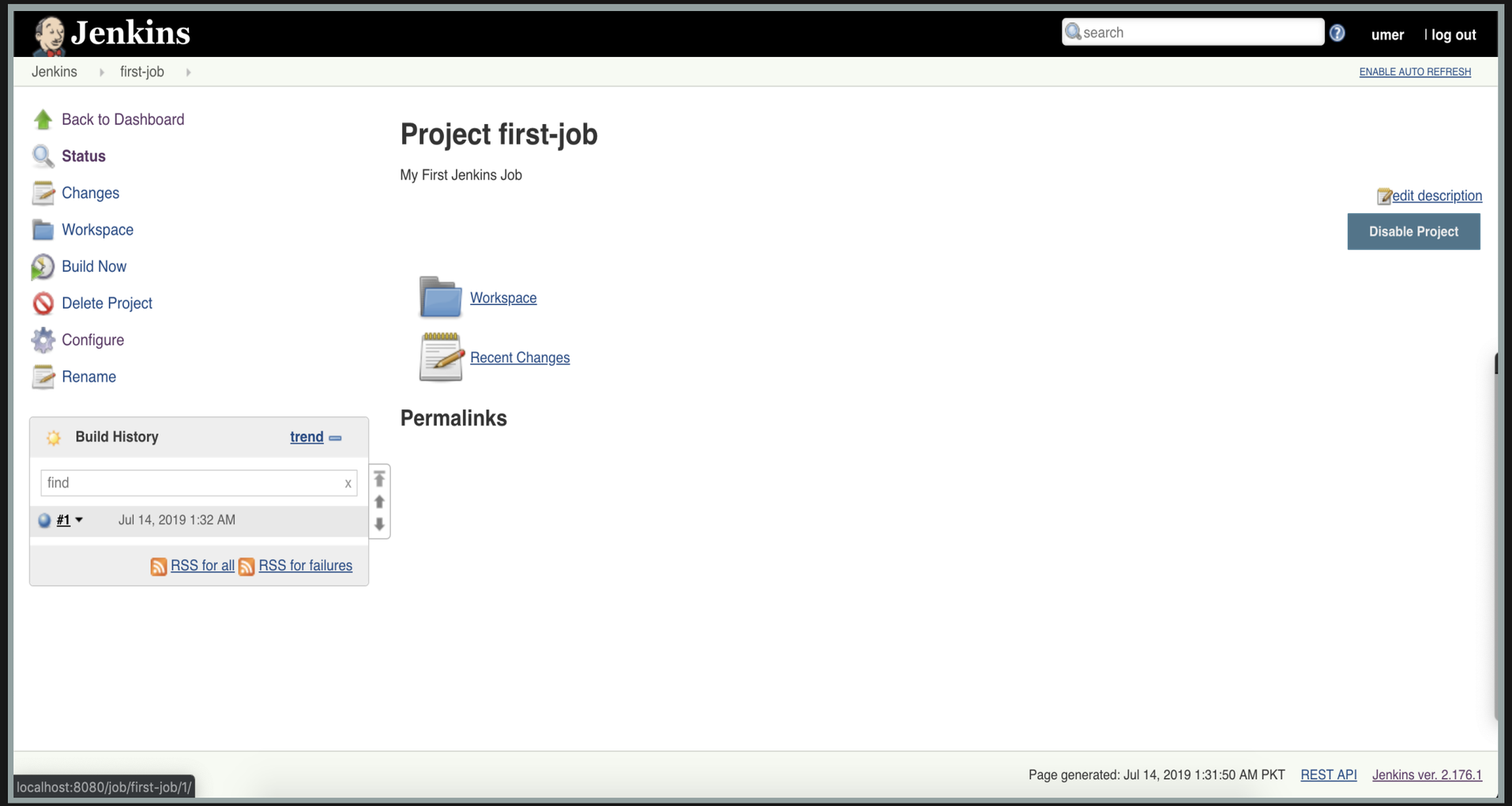
Type 'echo Hello World' in the Command field and click Save. This job will run this command.



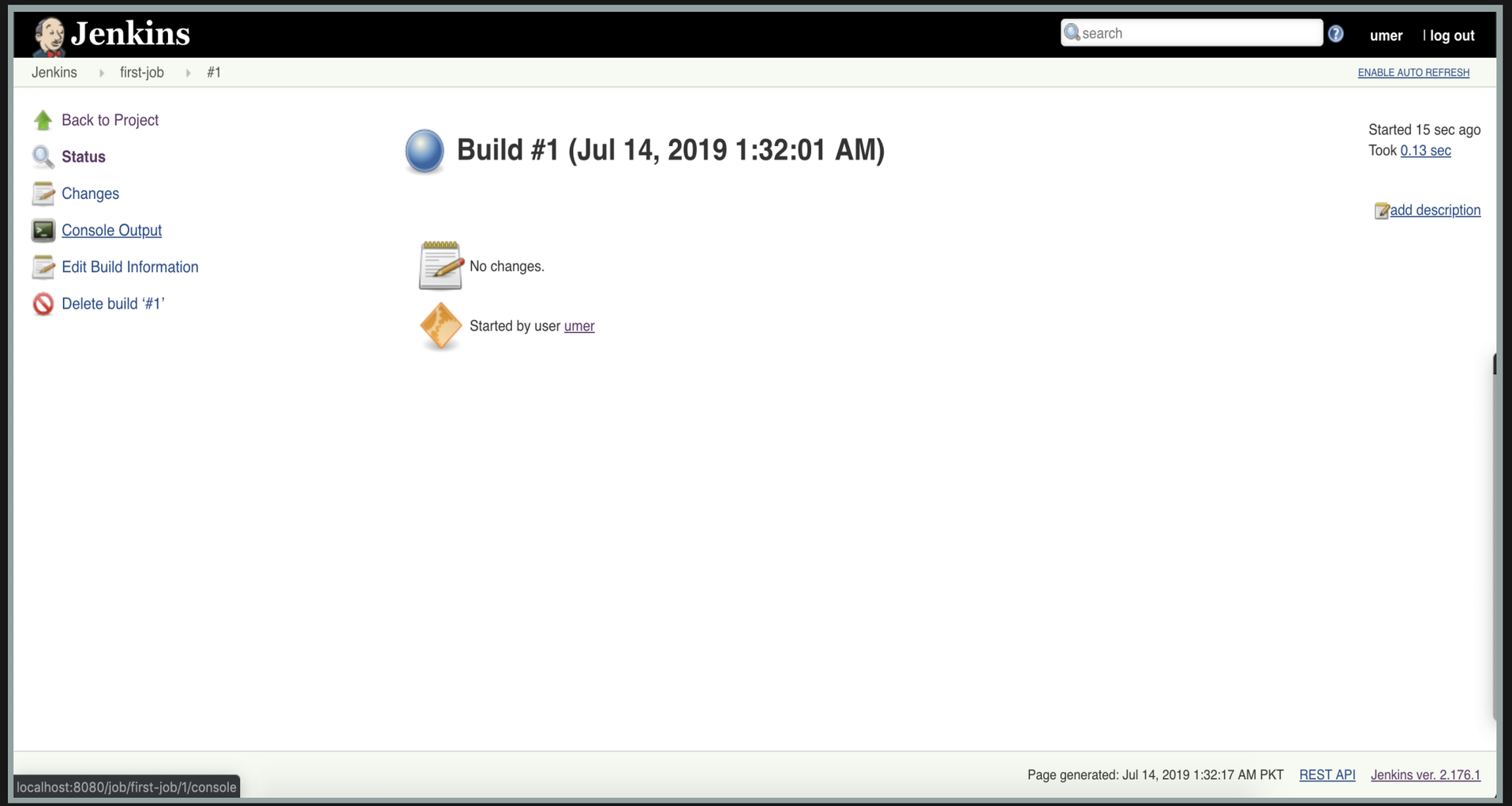
Next you will run your first job. Click Build Now



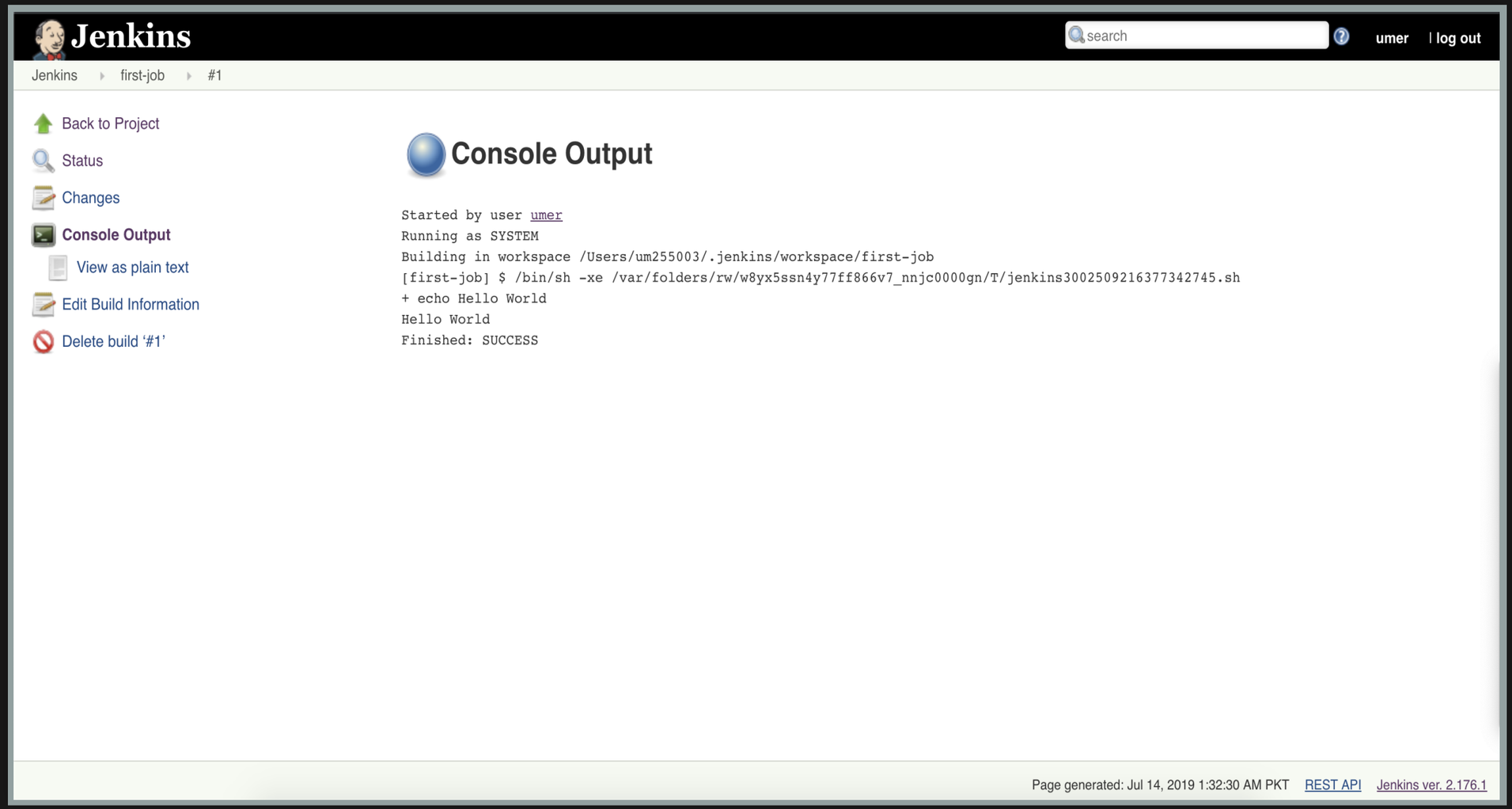
Once you click Build Now, you will notice a new entry in Build History



Click on the new build to see details as shown below. Next click on Console Output

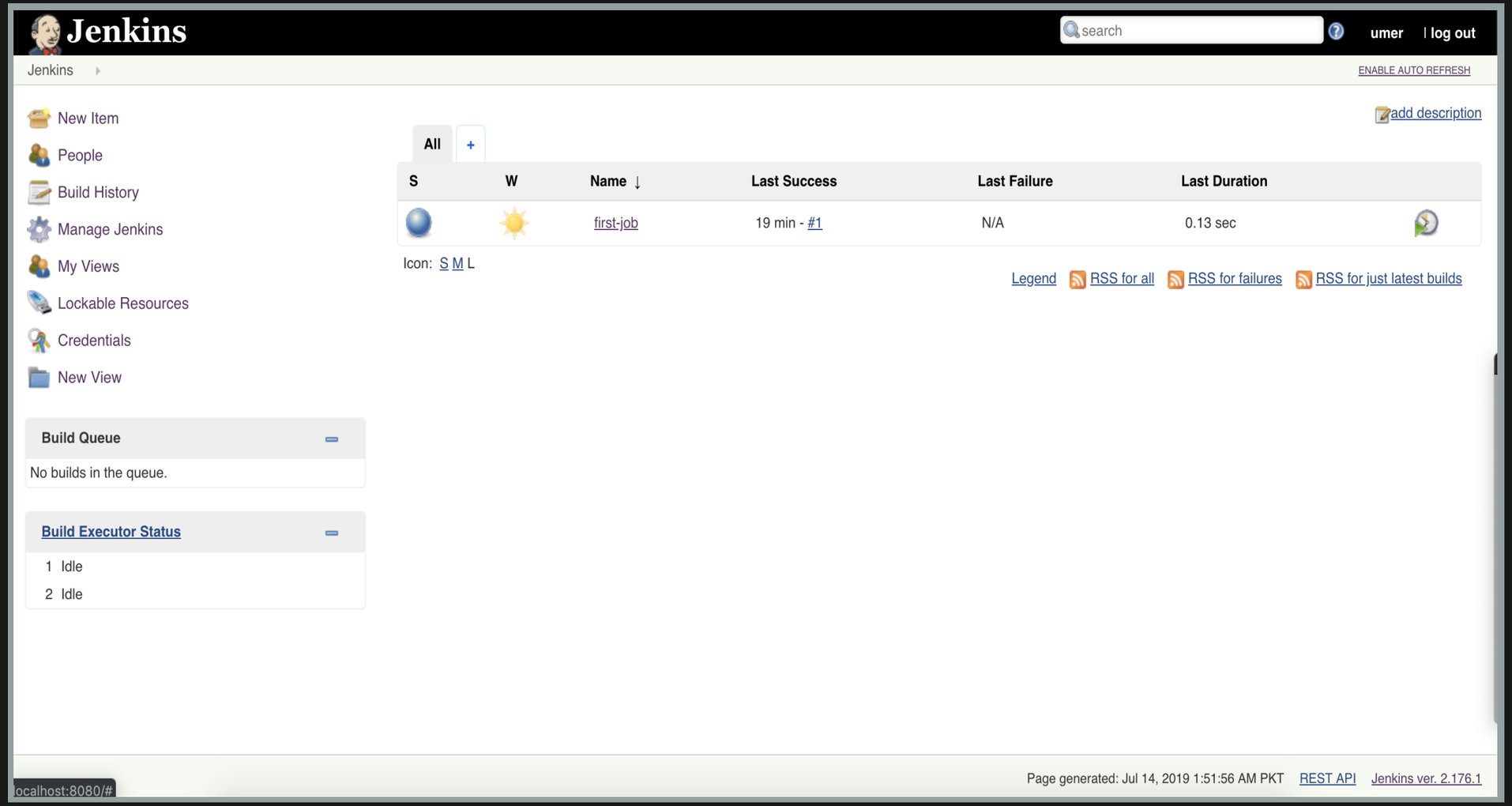


You should see Hello World displayed

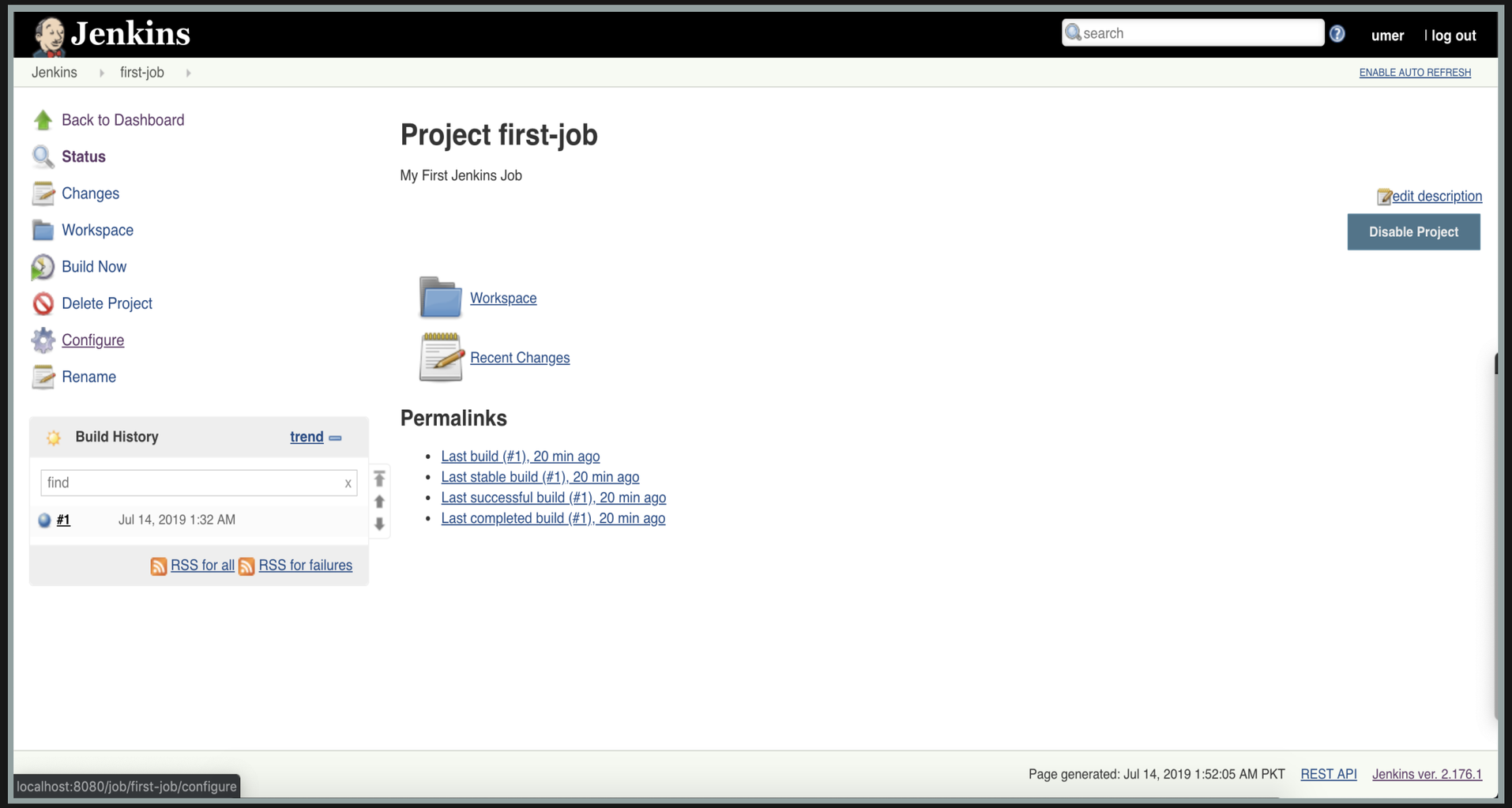


EDIT EXISTING JOB

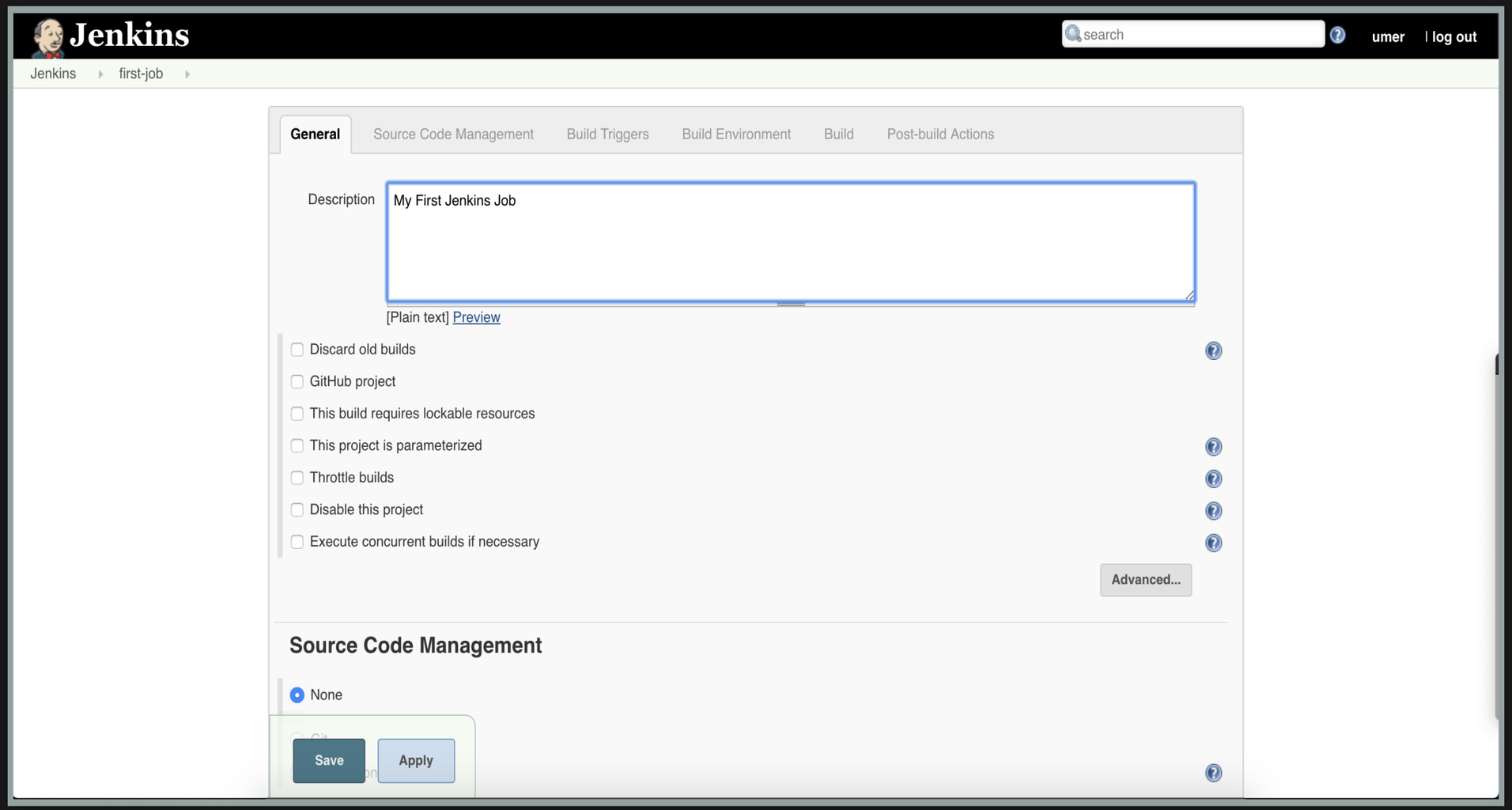
You are going to edit your first Jenkins job. Click on the previous job you created



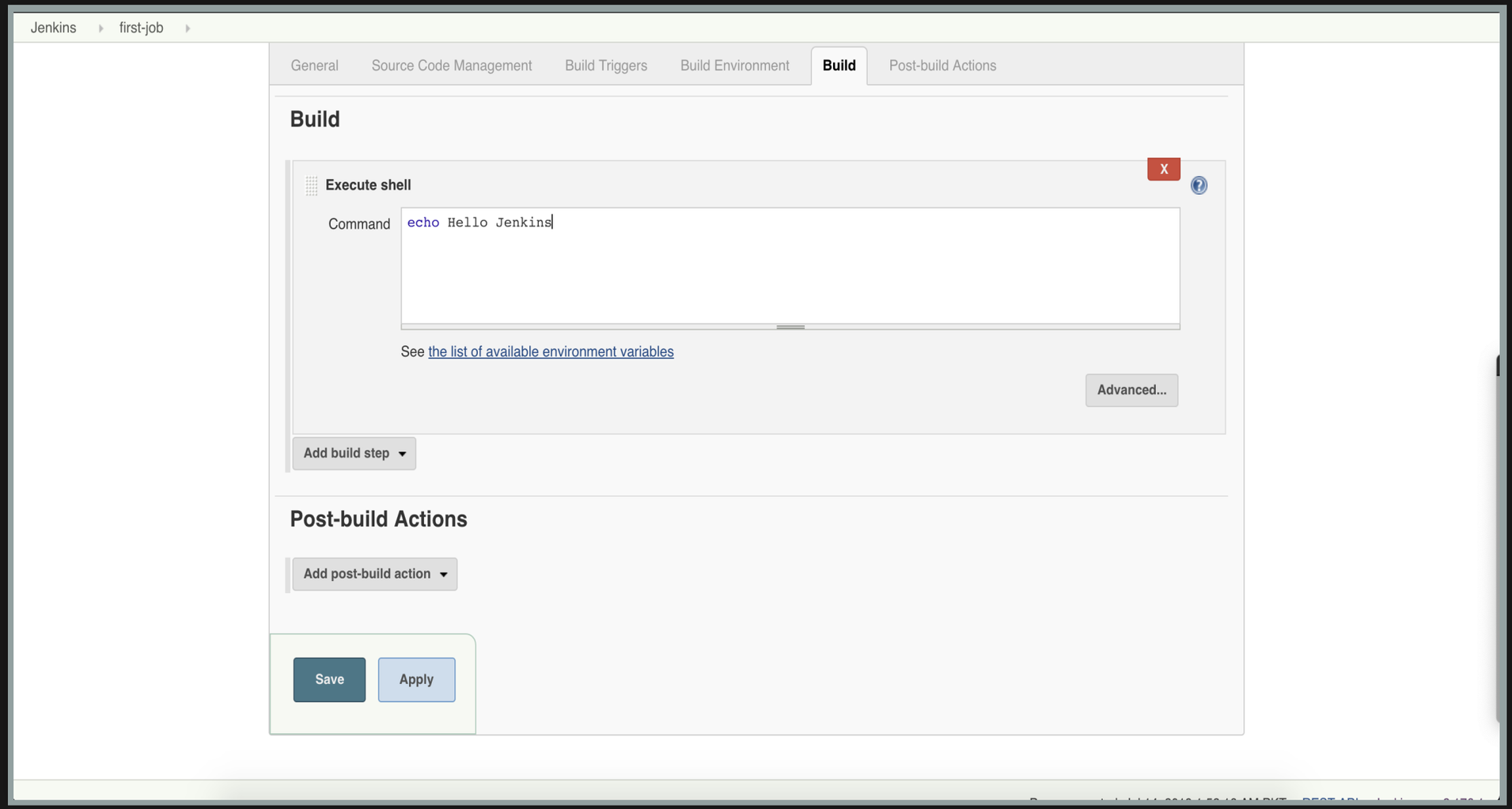
Next click Configure to edit your job



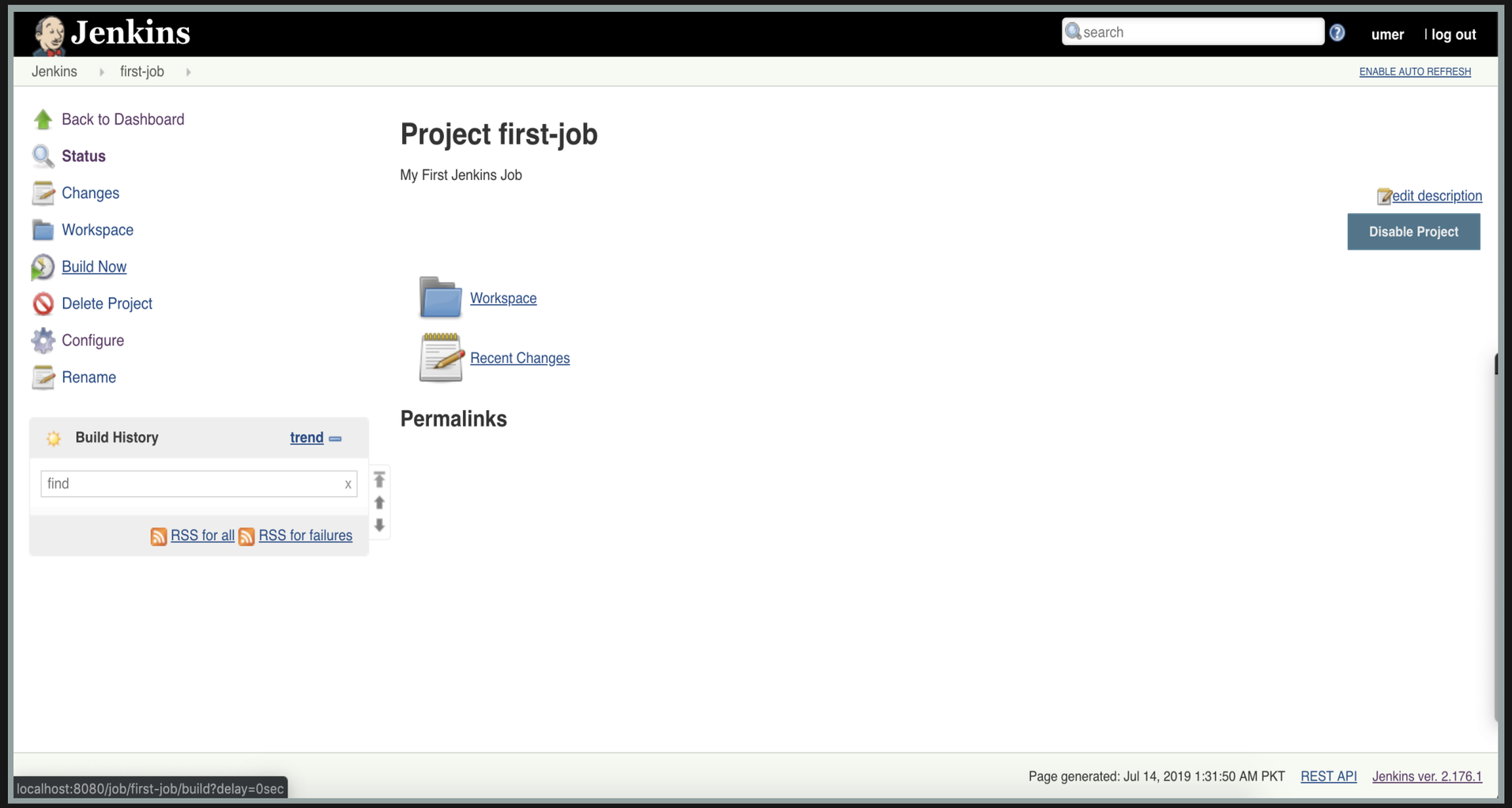
Click Build


Jenkins
search
umer
log out
Jenkins
first-job
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Description
My First Jenkins Job
[Plain text] Preview
Discard old builds
GitHub project
This build requires lockable resources
This project is parameterized
Throttle builds
Disable this project
Execute concurrent builds if necessary
Advanced...
Source Code Management
None
Save
Apply

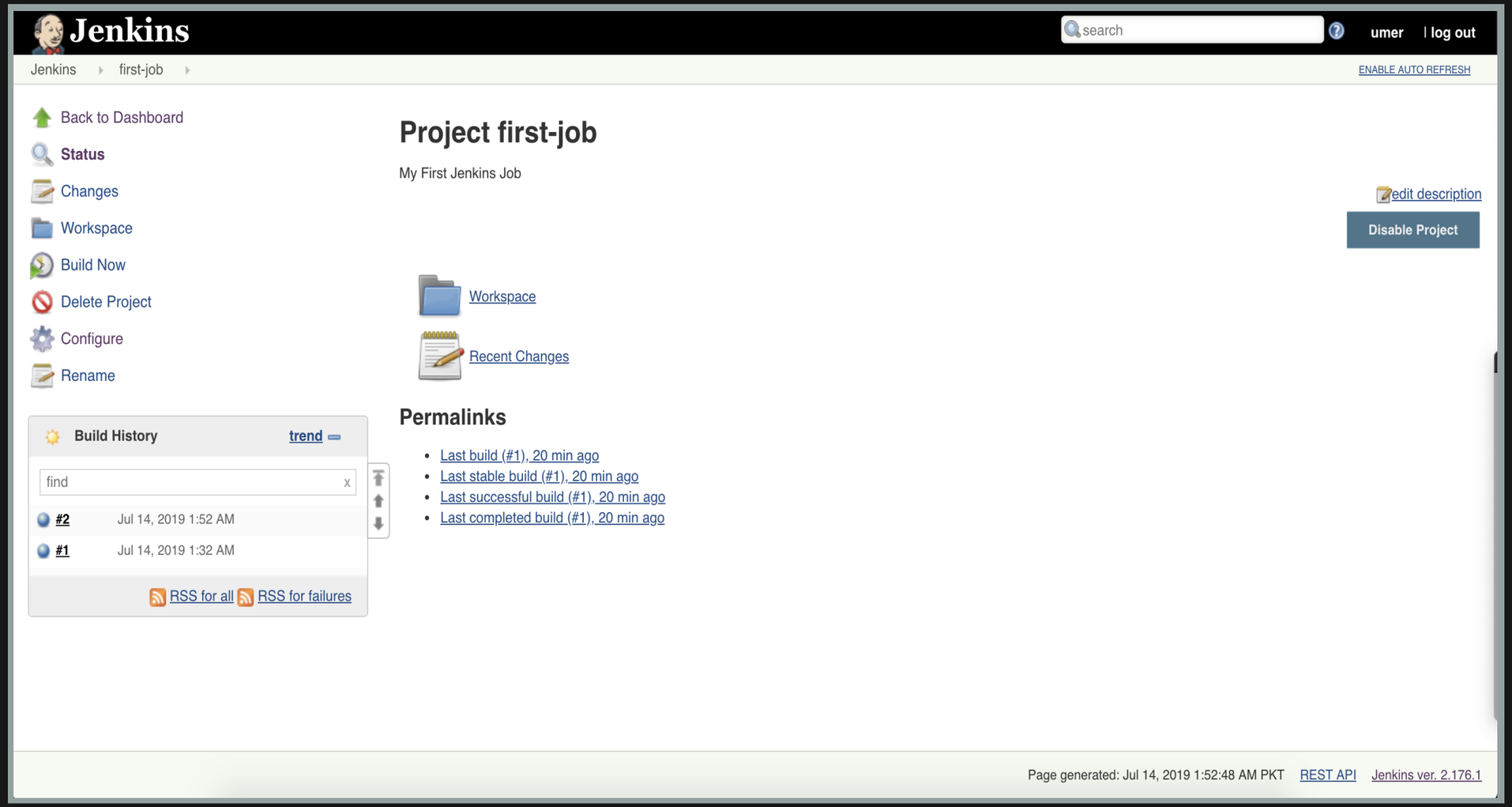
In the command field replace World with Jenkins and Click Save



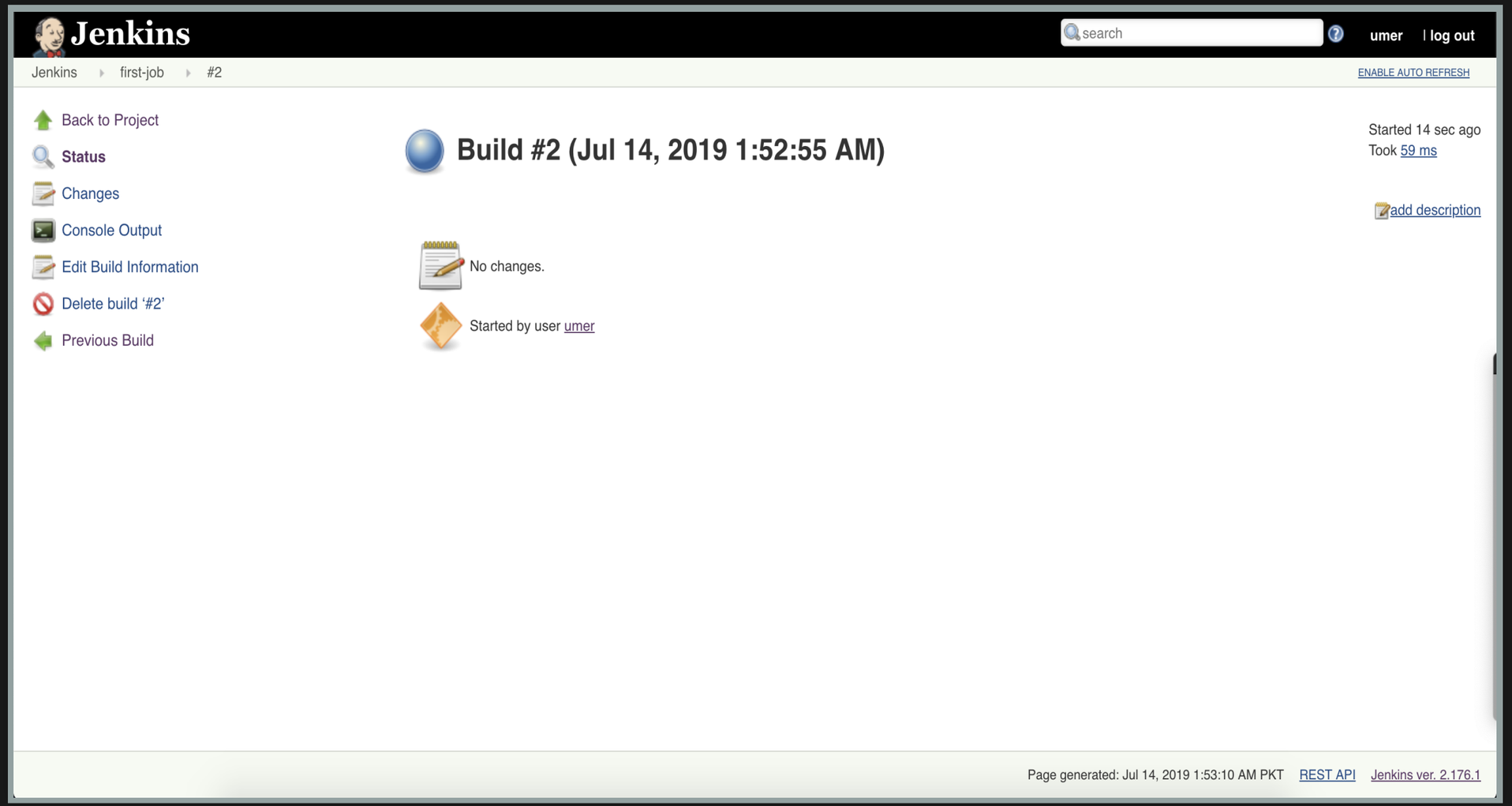
Next we are going to run our edited job for that click Build Now



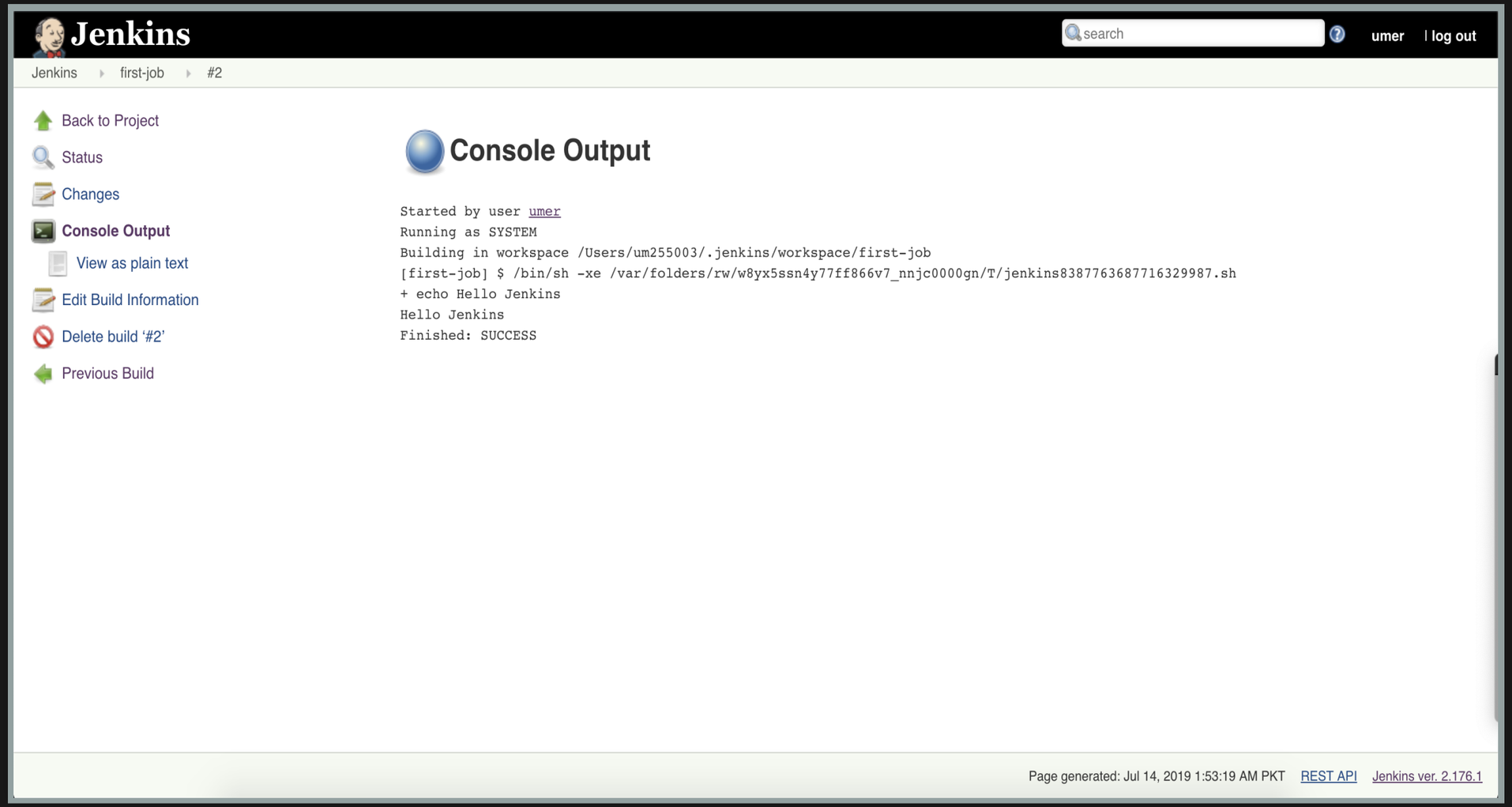
Once you click Build Now, you will notice a new entry in Build History



Click on the new build to see details as shown below. Next click on Console Output



You should see Hello Jenkins displayed



RUN SHELL SCRIPT

You are going to create a job to run a shell script. First created a shell script by running the command

```
# create shell script
vi /tmp/jenkins_script.sh
```

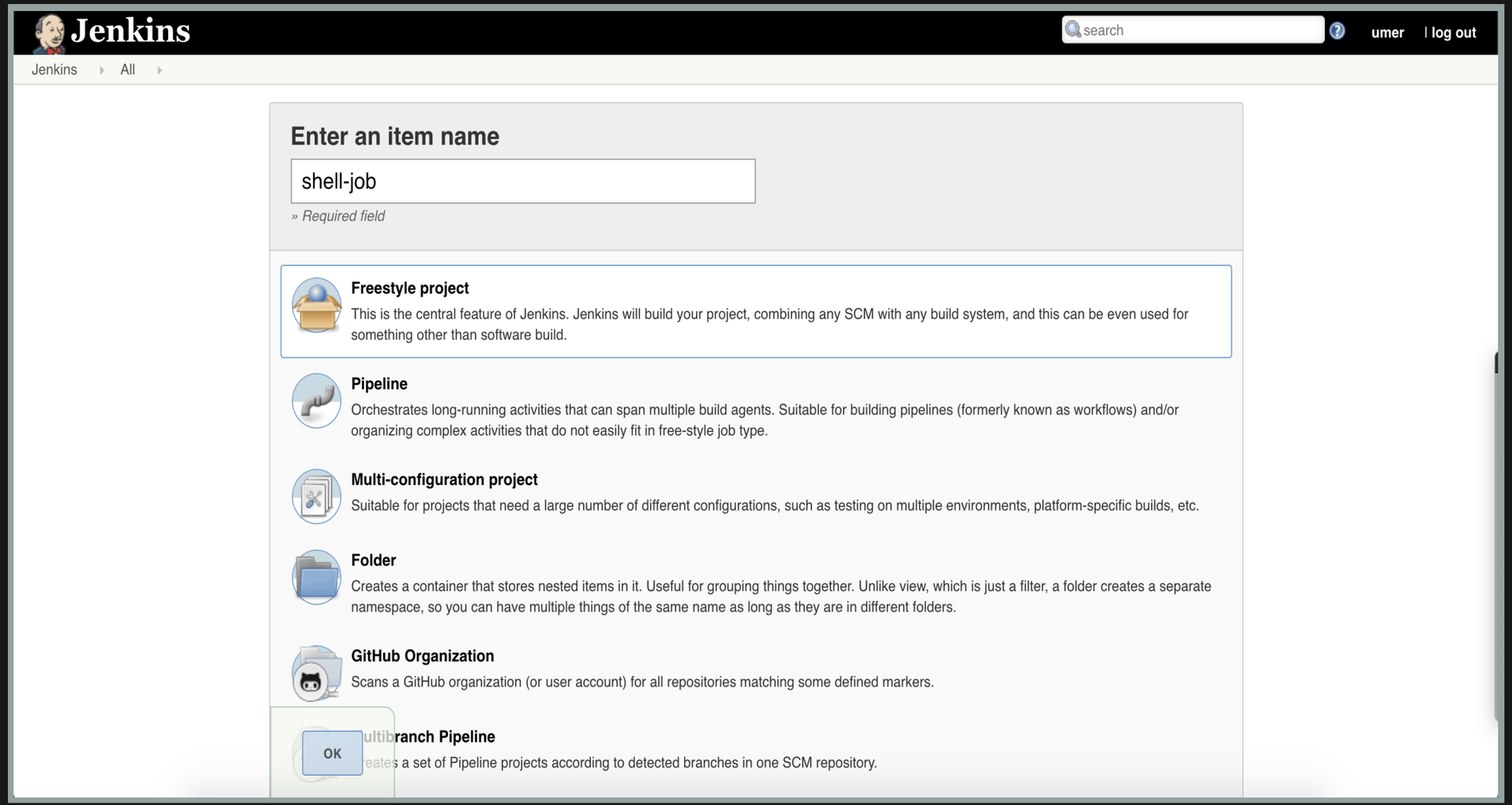
Copy following lines

```
#!/bin/sh
echo Hello $1 $2
```

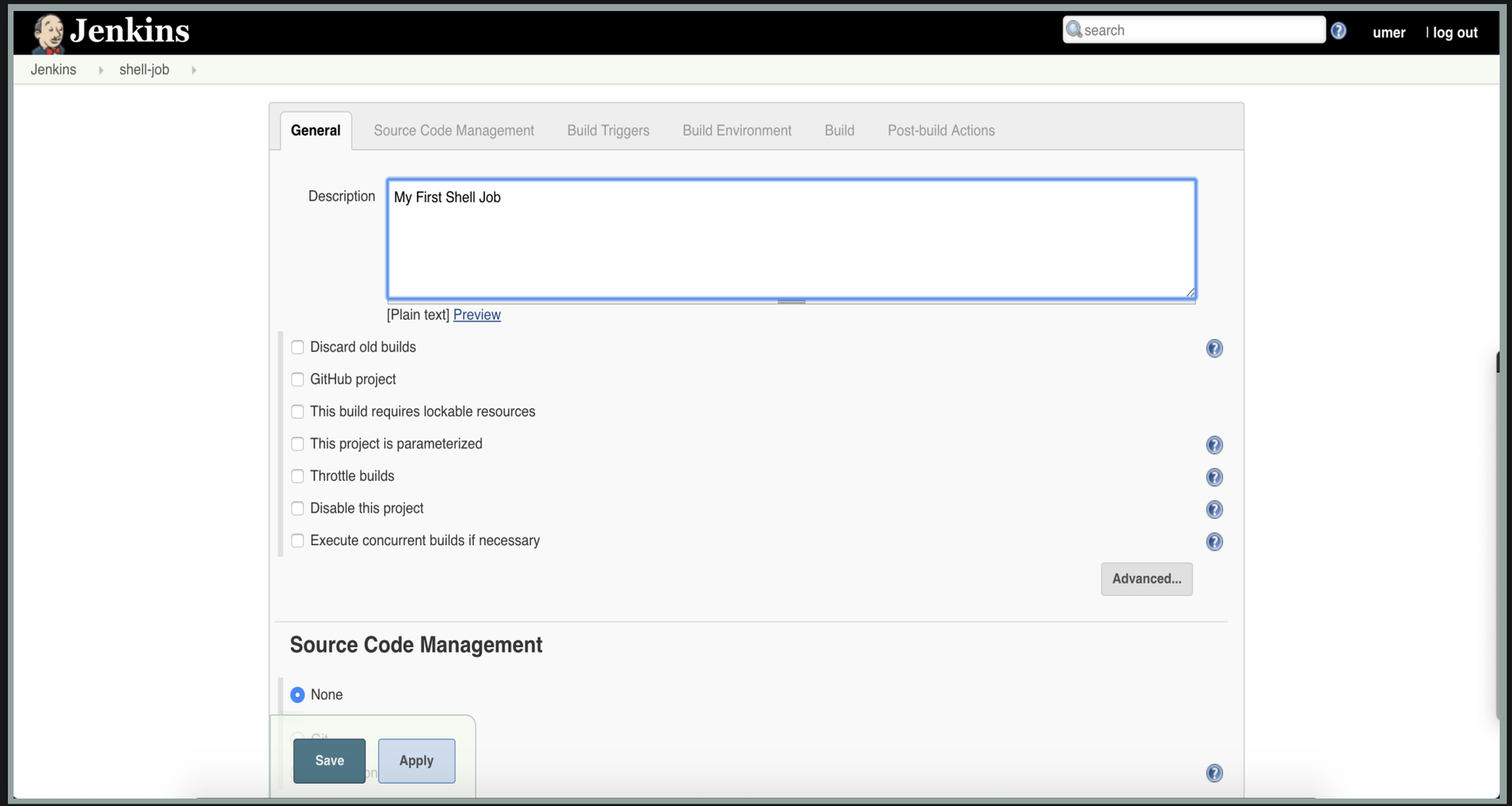
Now make the script executable

```
chmod u+x /tmp/jenkins_script.sh
```

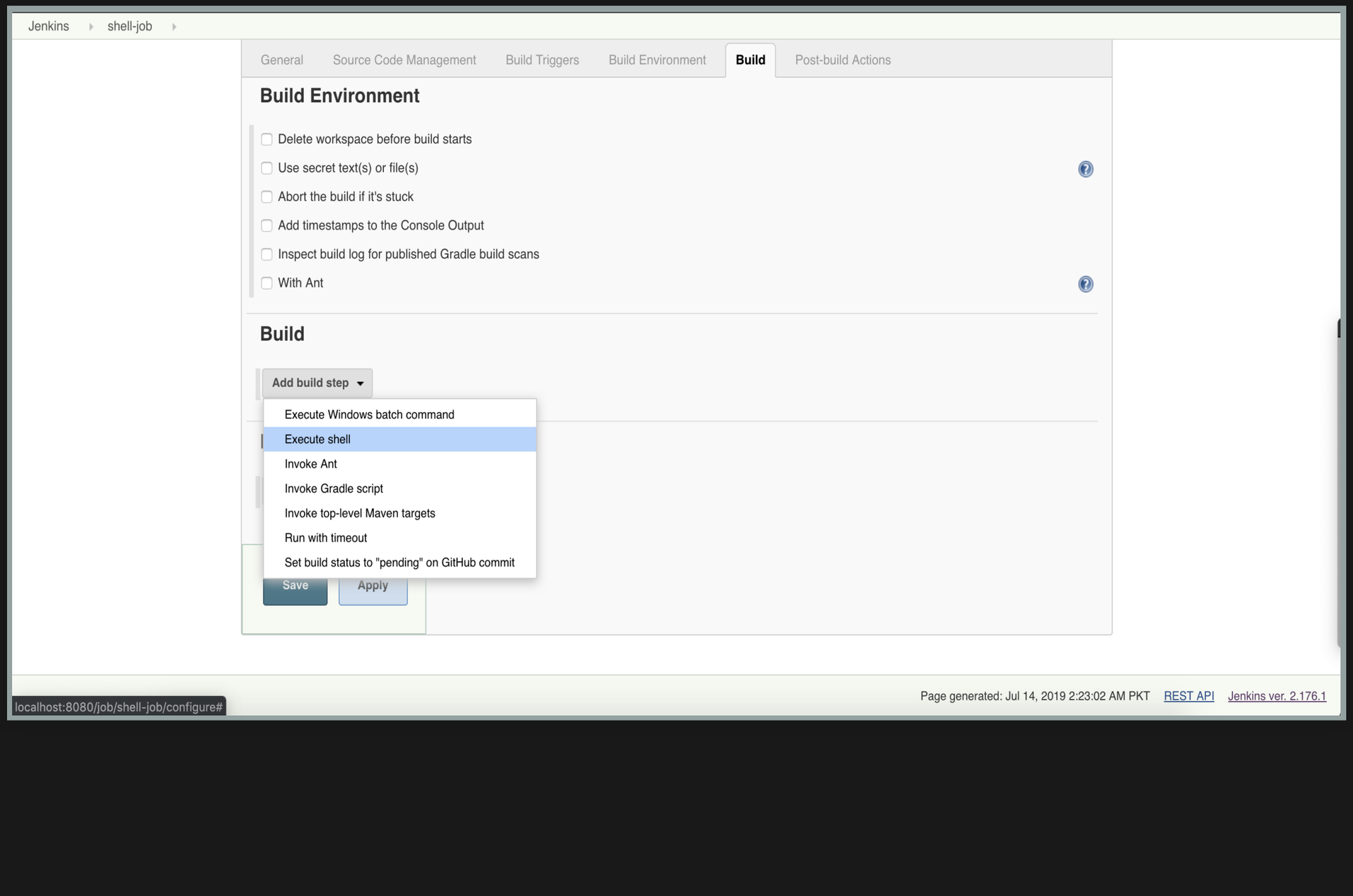
Enter the job name, select Freestyle Project and Click OK



Enter you the description of your job and click Build to move to that section



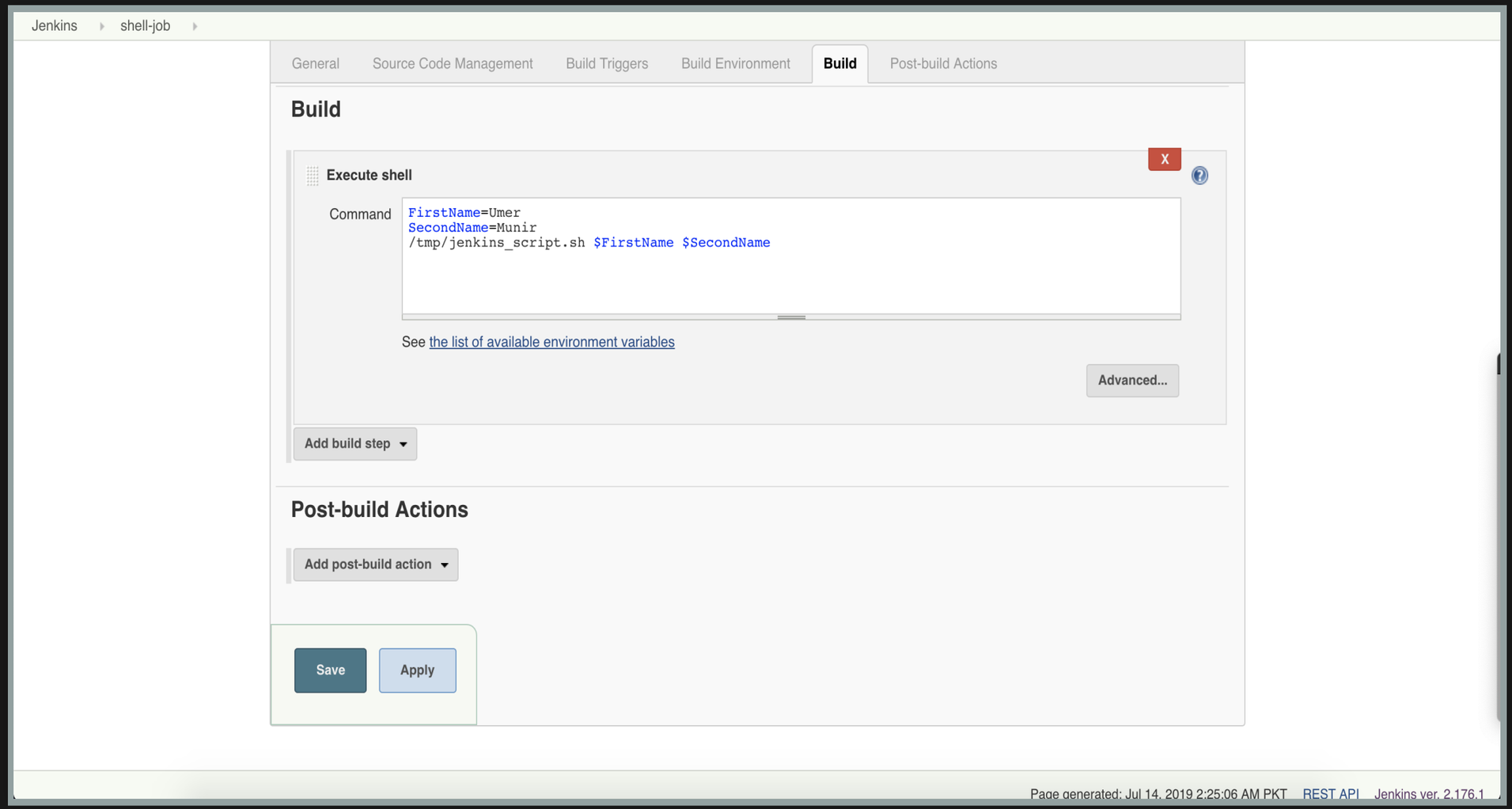
Select Execute shell in the Add build step drop down as shown below



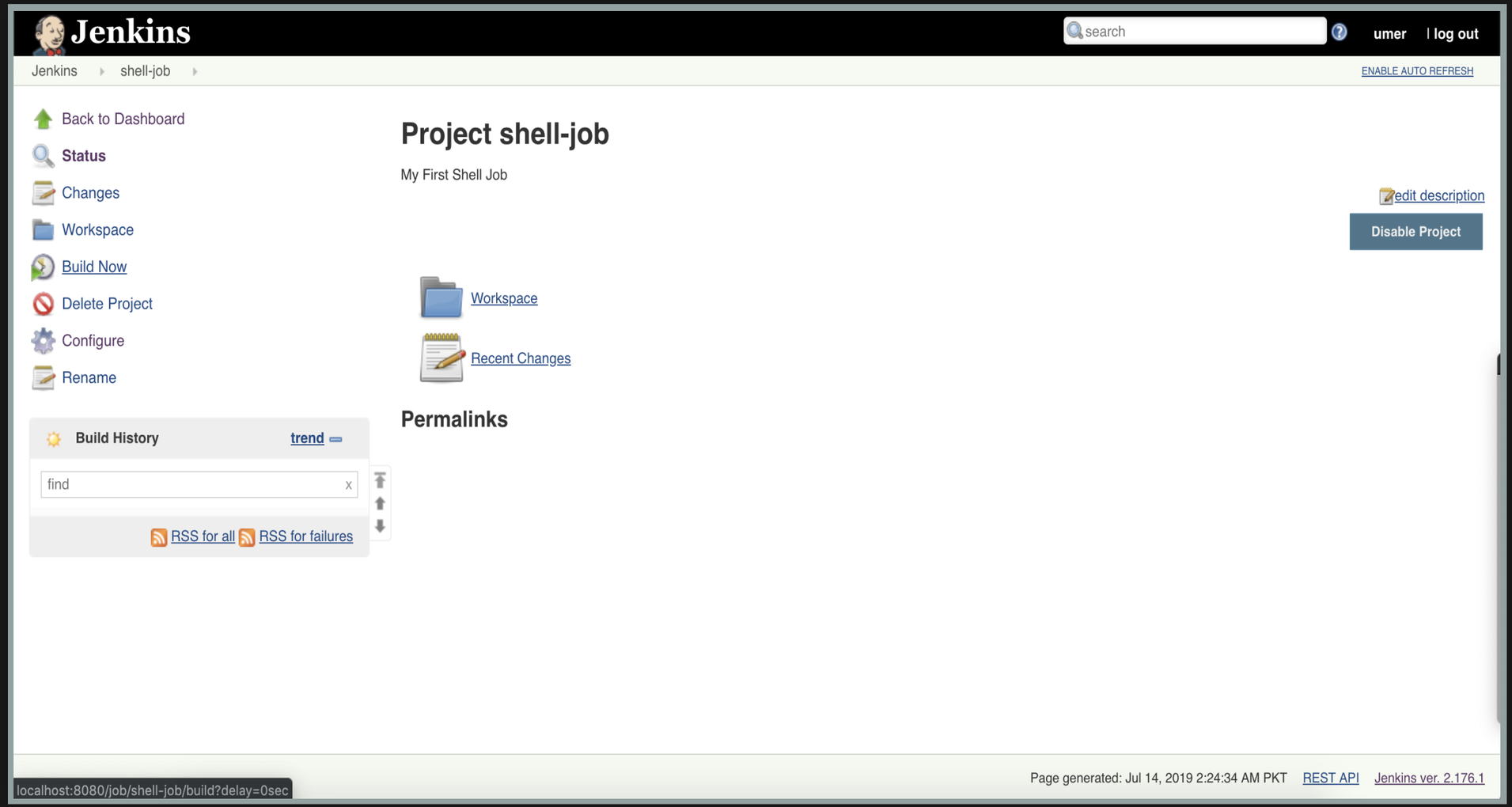
In the command field type the following commands and Click Save

```
FirstName=FIRSTNAME
LastName=LAST-NAME
/tmp/jenkins_script $FirstName $LastName
```

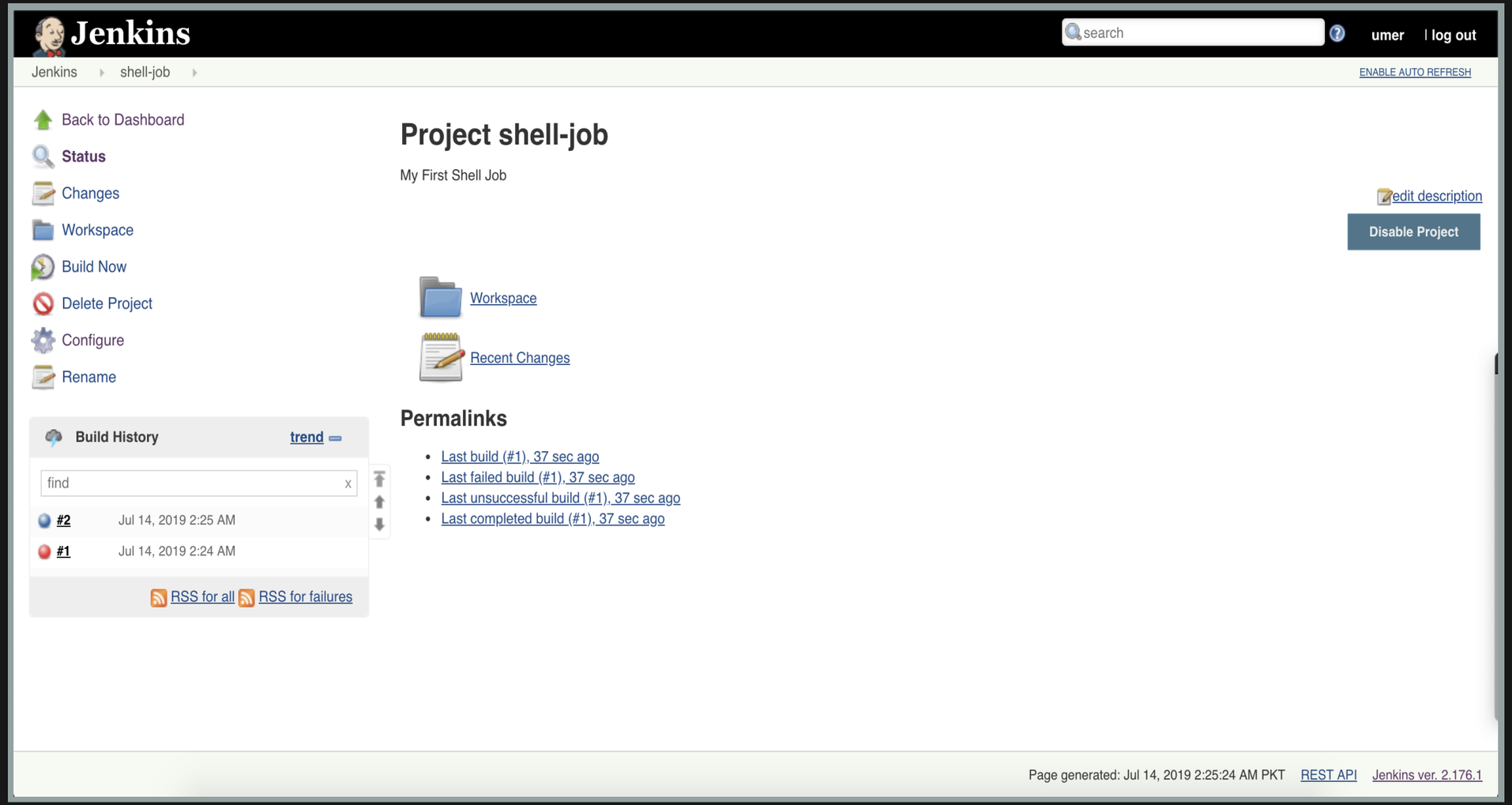
Replace FirstName and LastName with your First name and last name



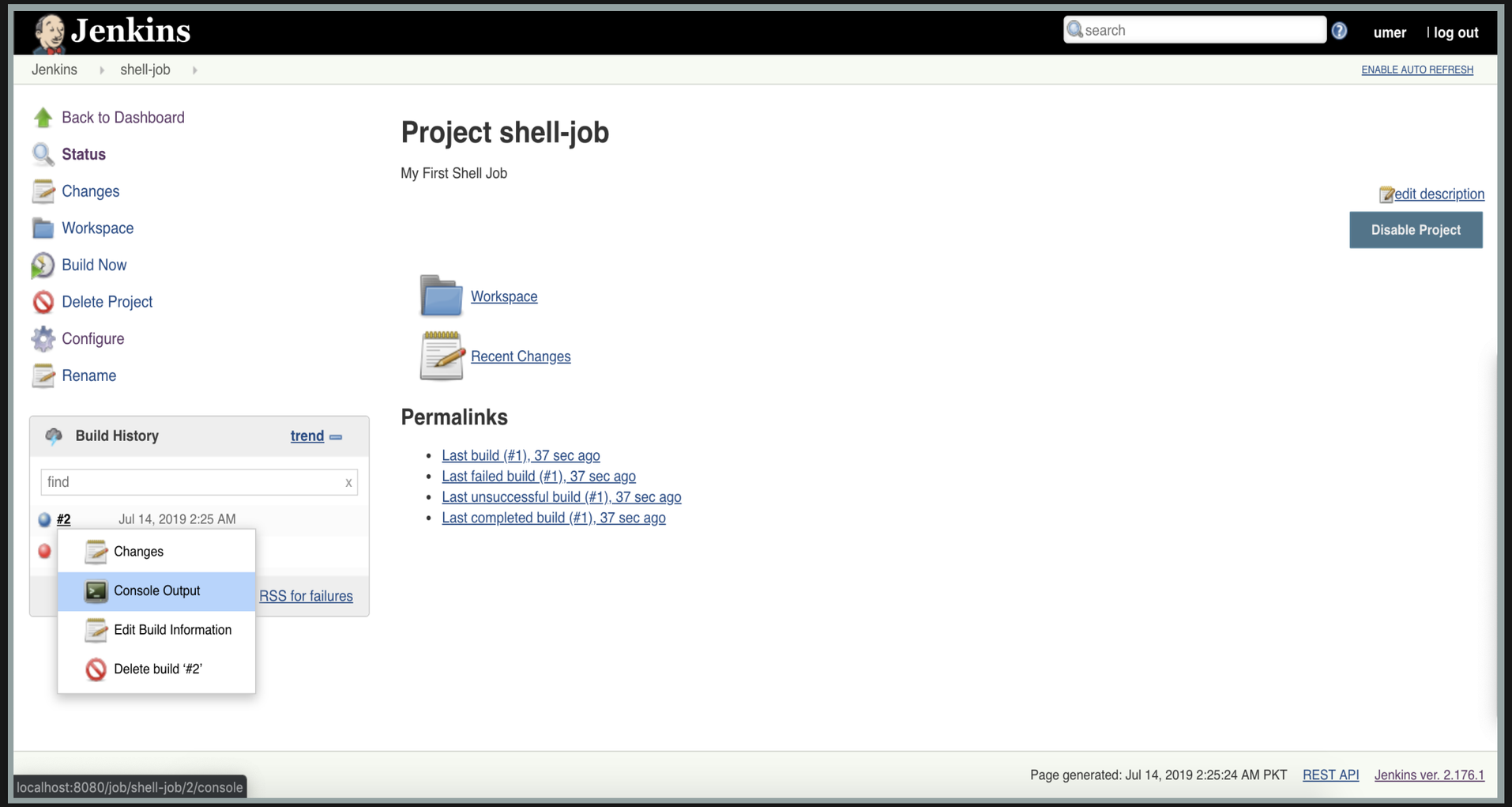
Click Build Now to start manual build



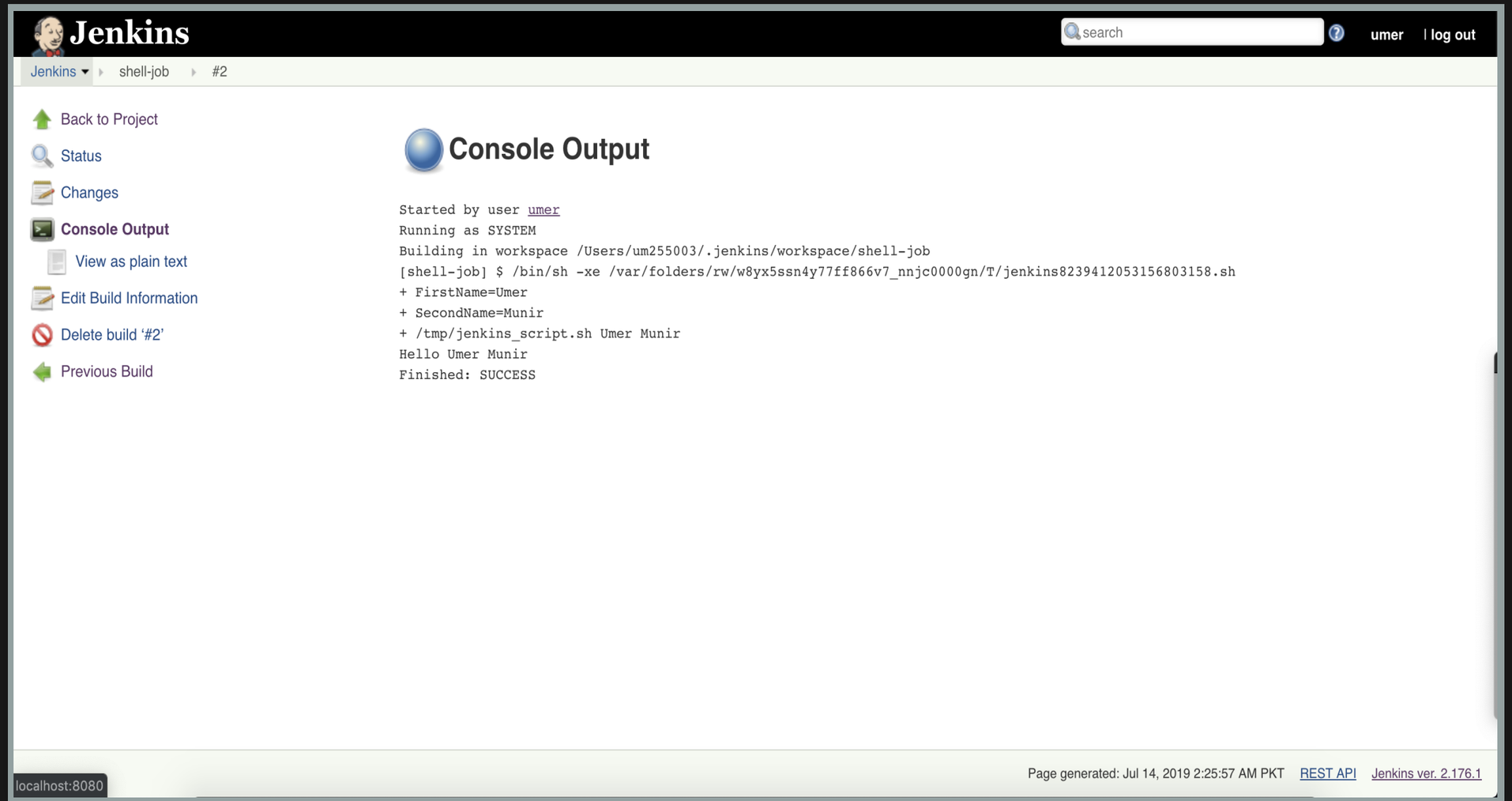
Successfull builds are color coded as blue and failed builds as red



You can check the Console Output by clicking the down arrow next to latest build

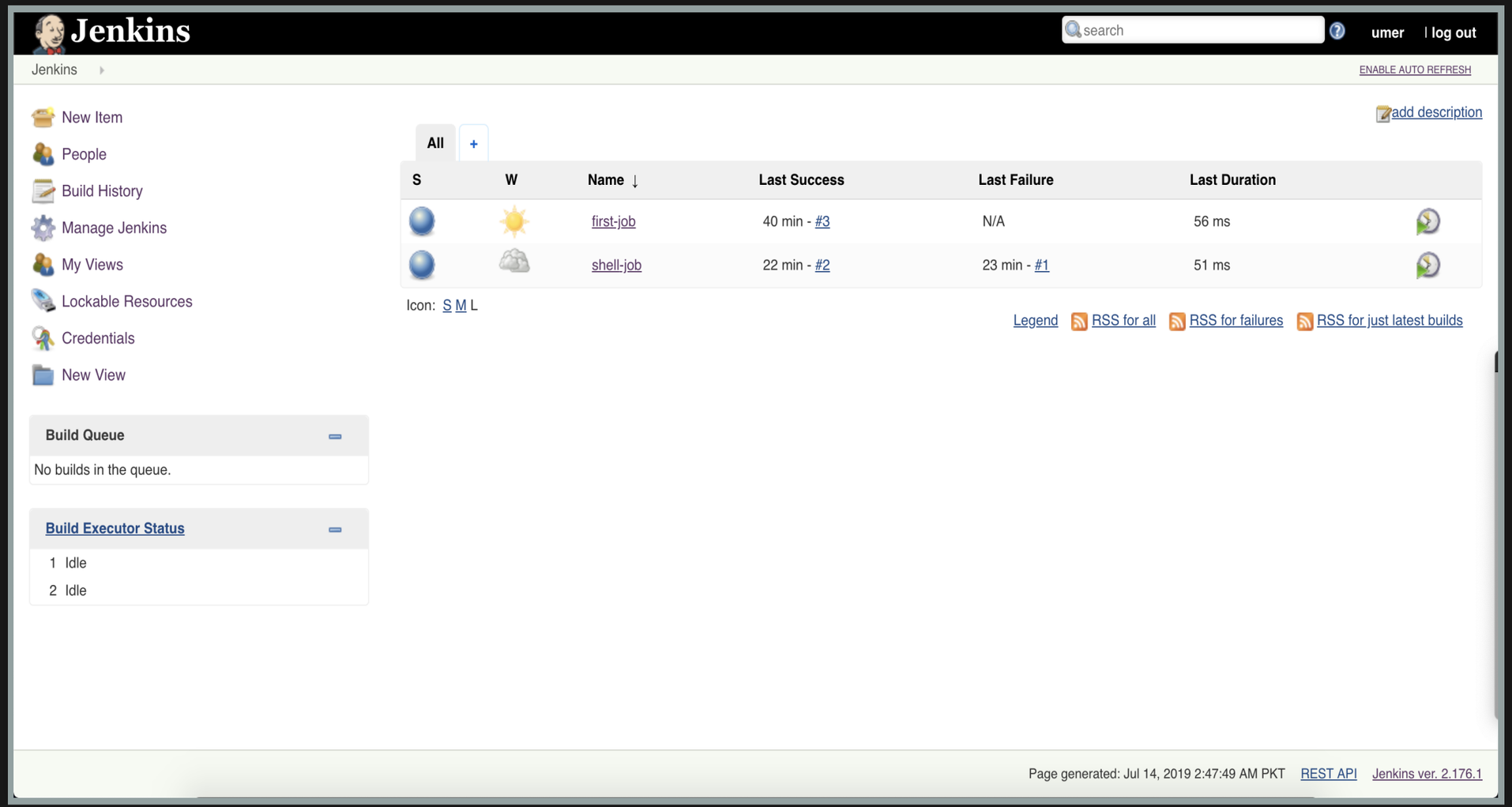


You should see your name in the Console Output section

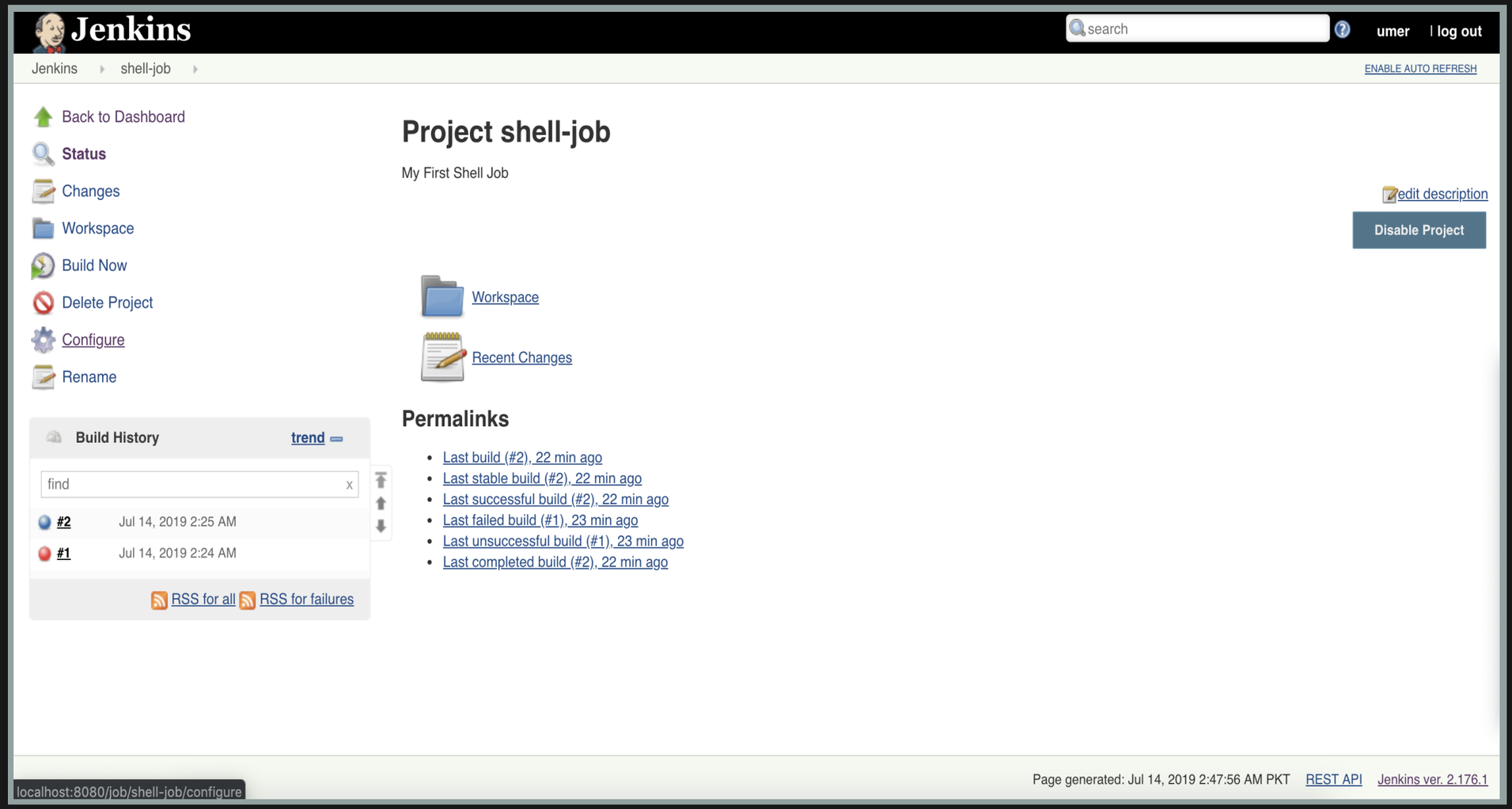


PARAMETERIZED JOB

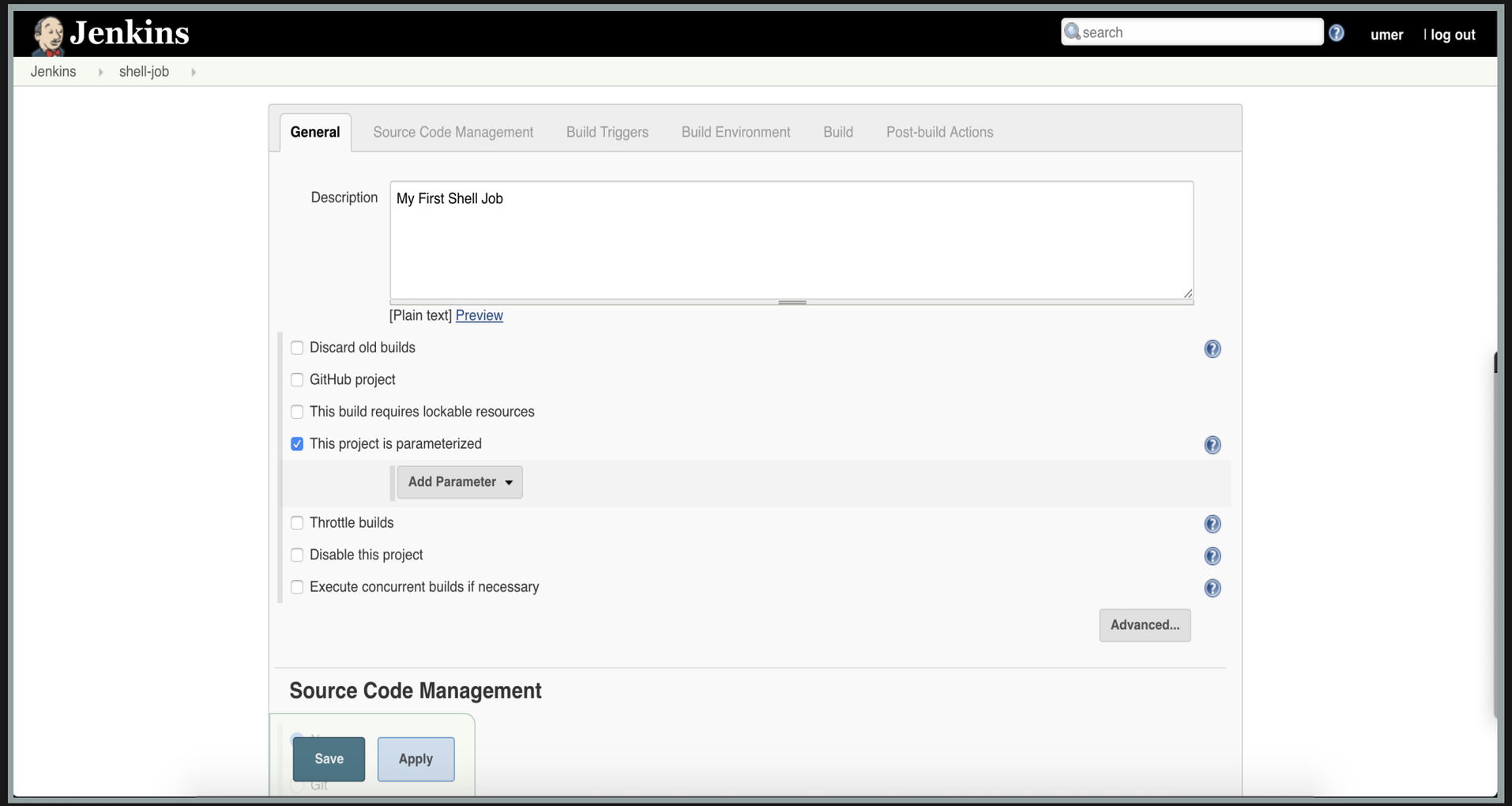
You are going to edit your shell job to parameterized job. Go to Jenkins landing page by clicking on Jenkins on the left top corner and click your shell job



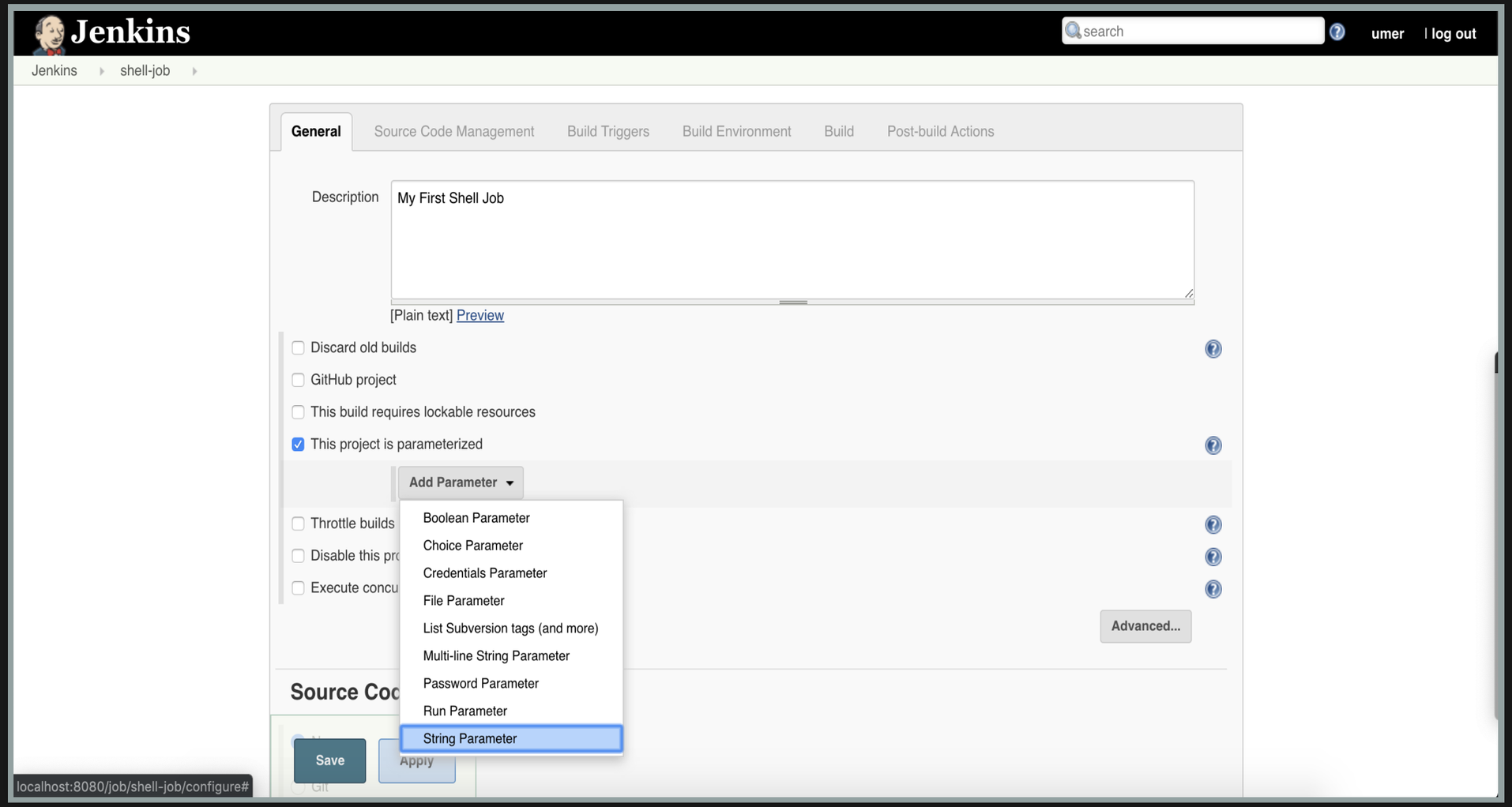
Click configure



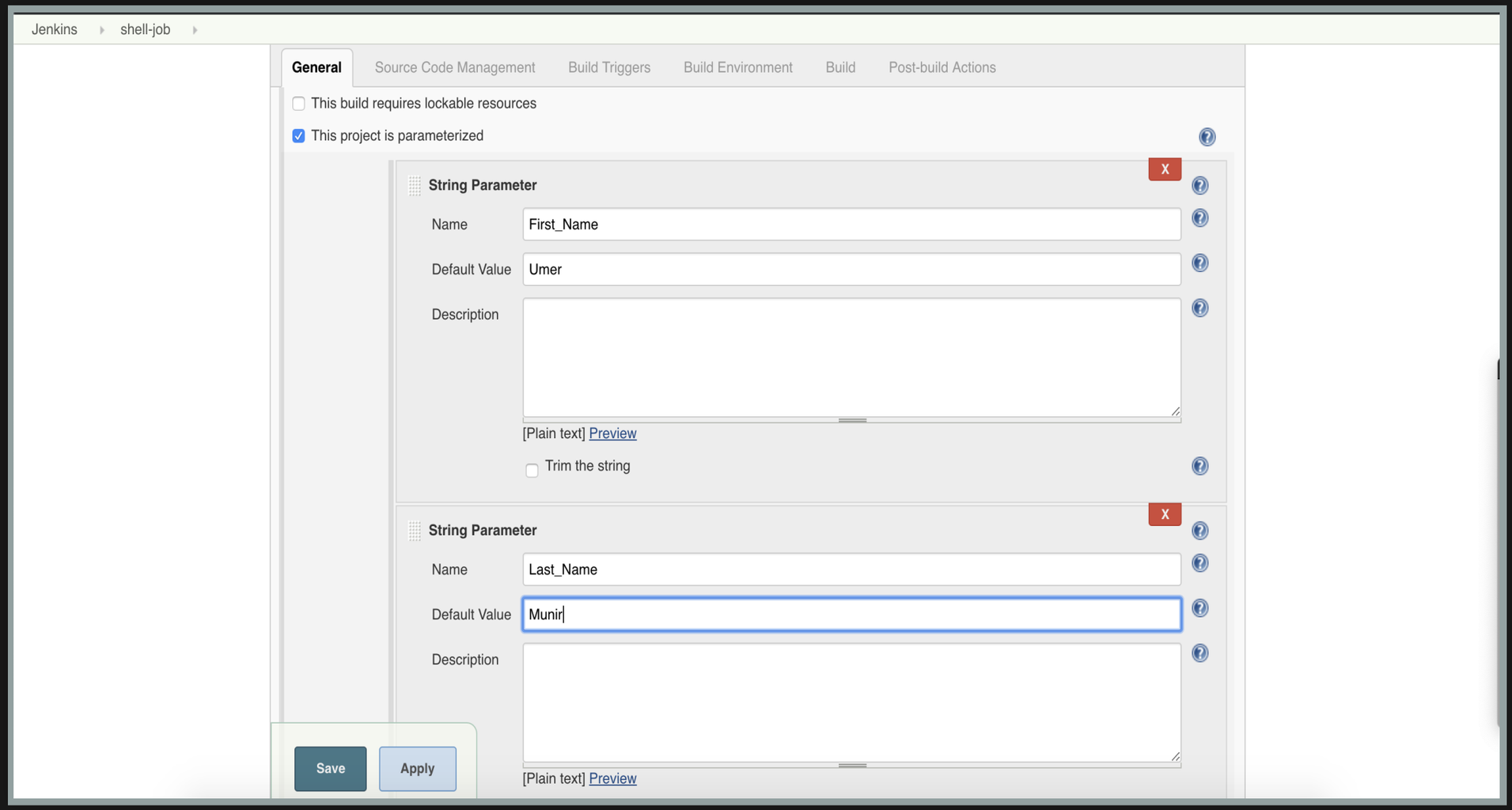
Check 'This project is parameterized box' as shown below and Click Add Parameter



Select String Parameter

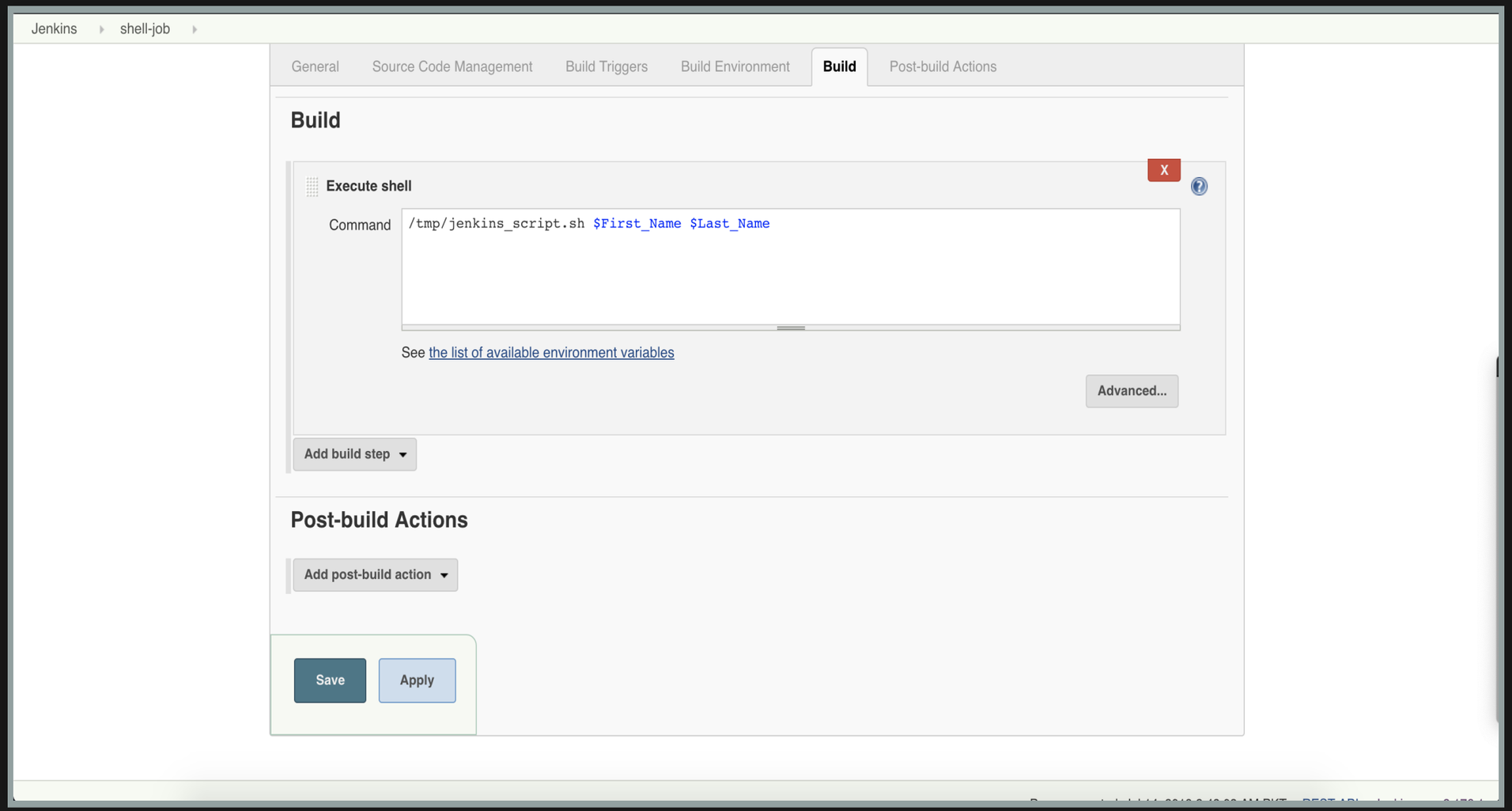


Specify Variable and Default Value as shown below. Replace First_Name and Last_Name with your First name and last name. Click Build to move to that section.

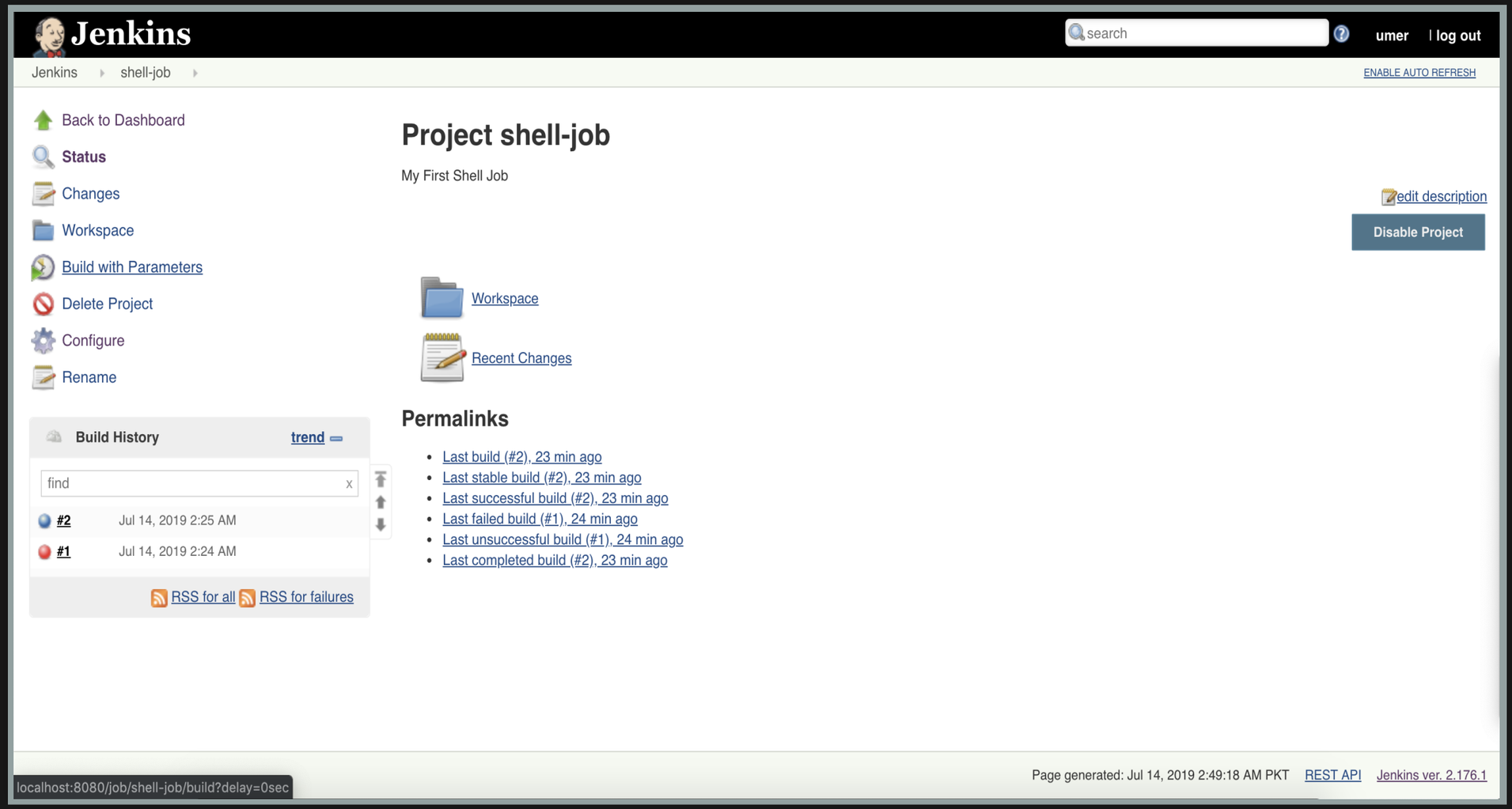


Replace command field with the following command and Click Save

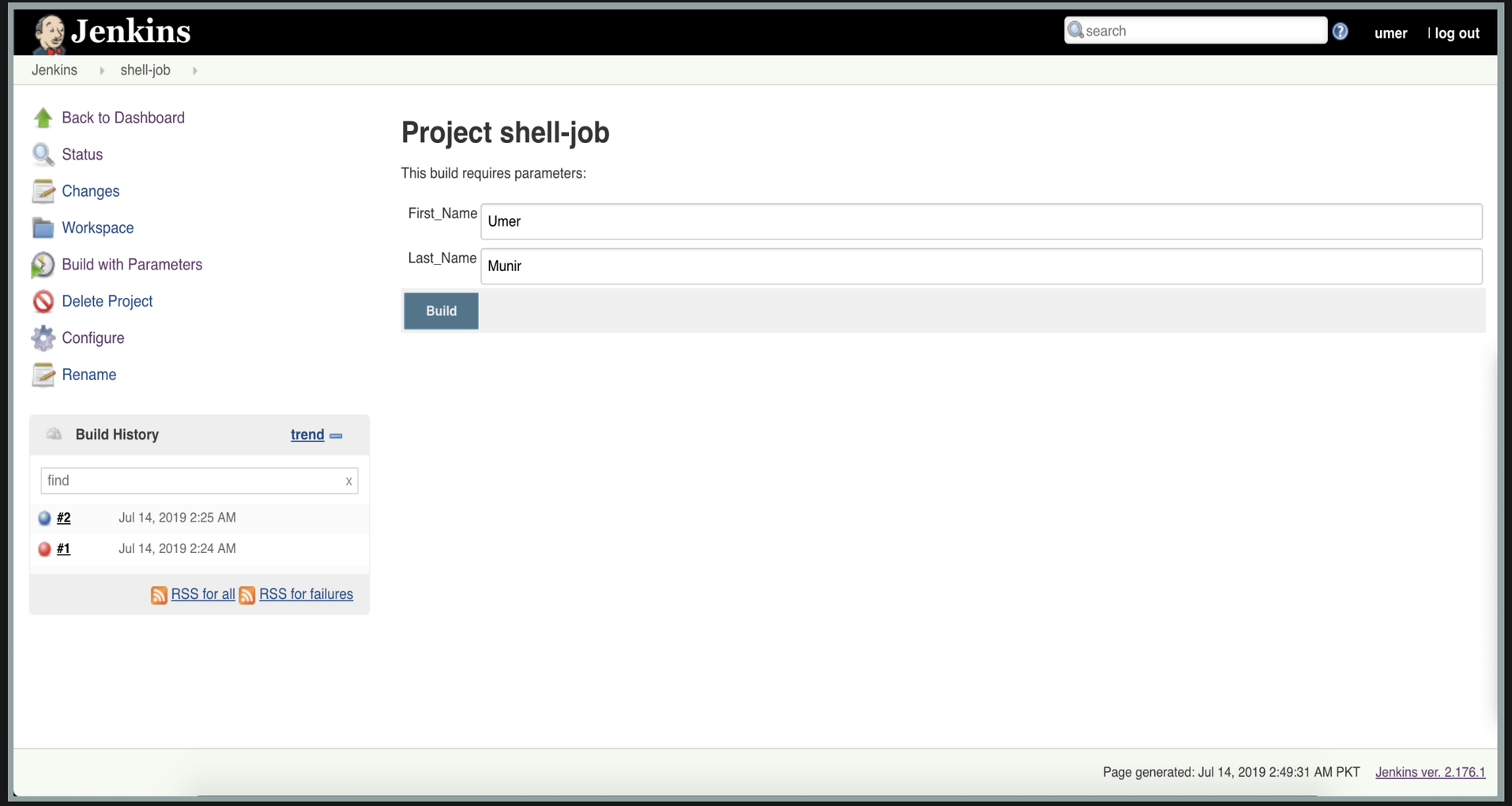
```
/tmp/jenkins_script.sh $First_Name $Last_Name
```



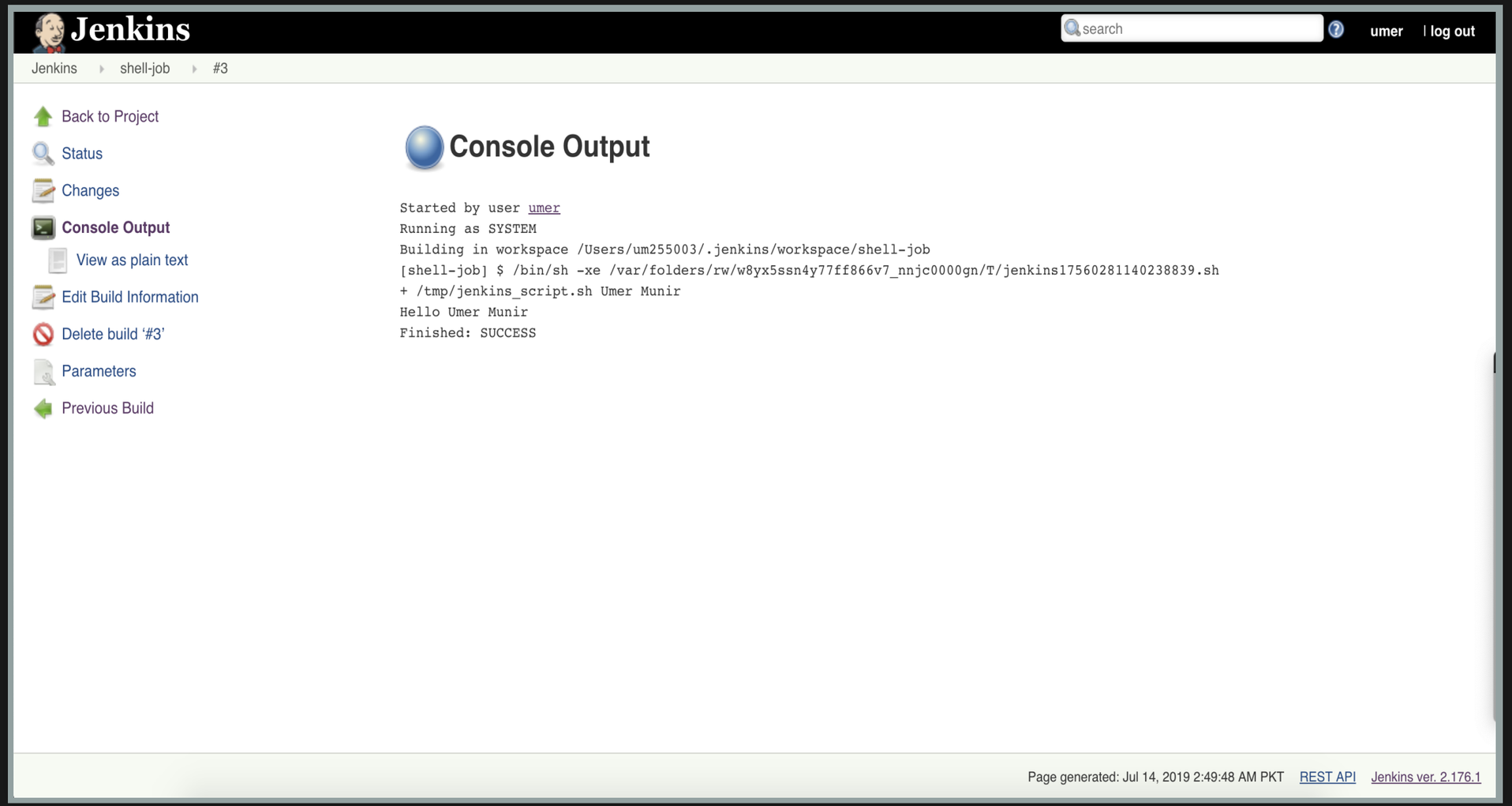
Click Build with Parameters to run the job.



It will prompt for parameters, enter the values and Click Build



You should see your name in the Console Output section

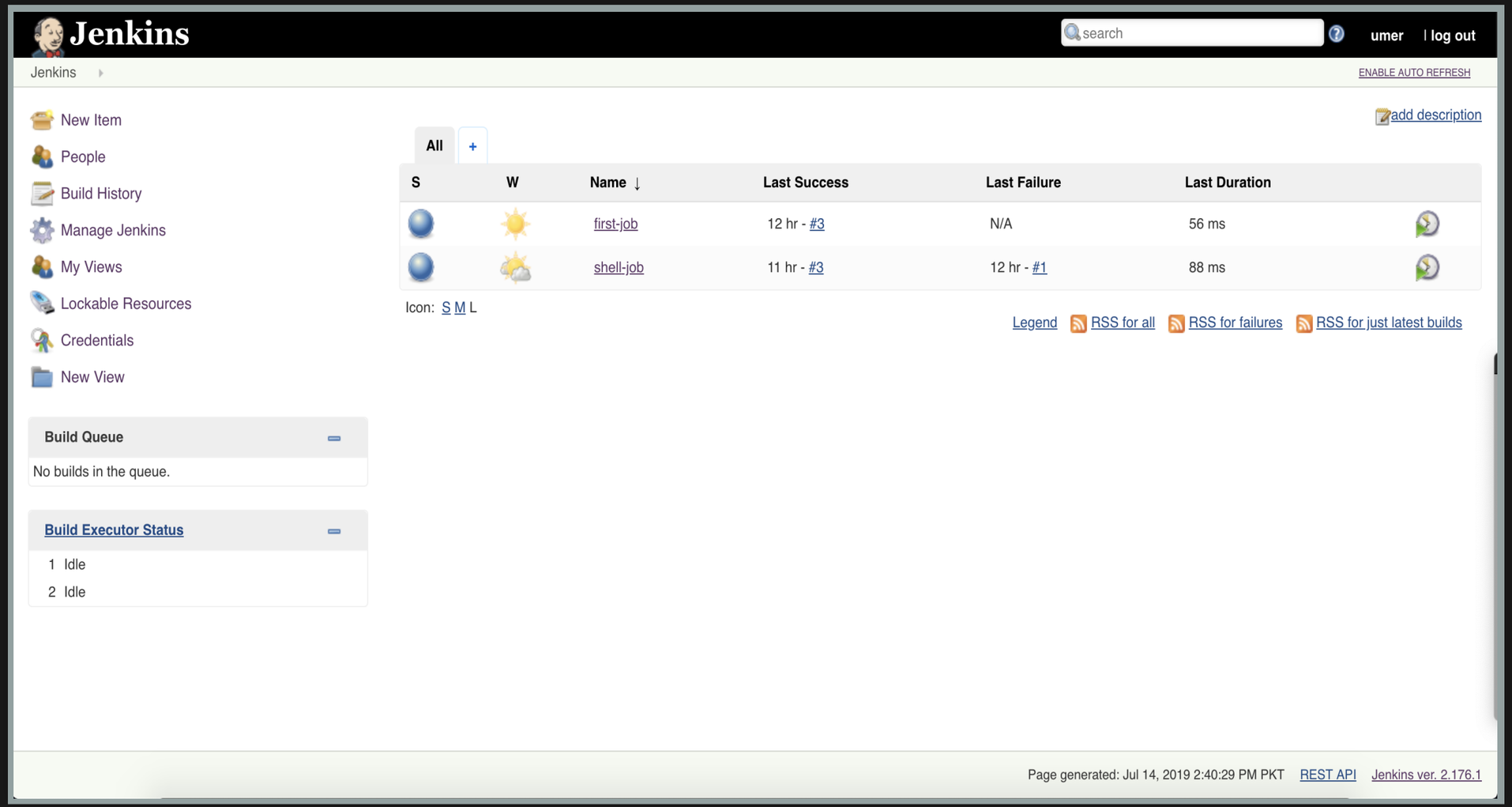


GITHUB INTEGRATION

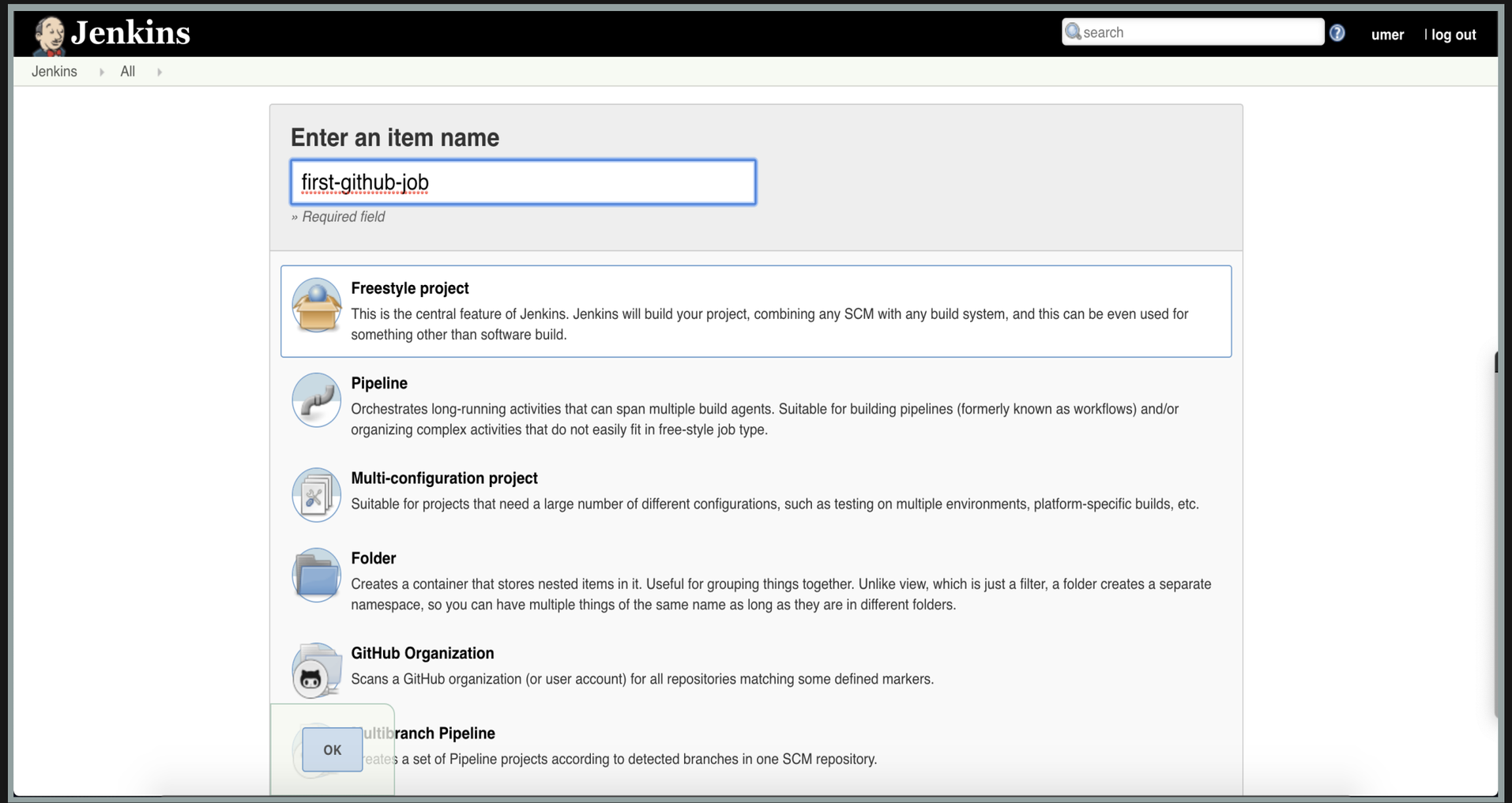
- Integrate Jenkins with GitHub
- Integration built into the default installation
- Number of plugins for integrating with GitHub
- Trigger jobs
  - Polling
  - Service Hook
- Status can be recorded back to GitHub
- Itself located on GitHub Jenkins
- GitHub Branch Source Plugin

LAB - GITHUB INTEGRATION

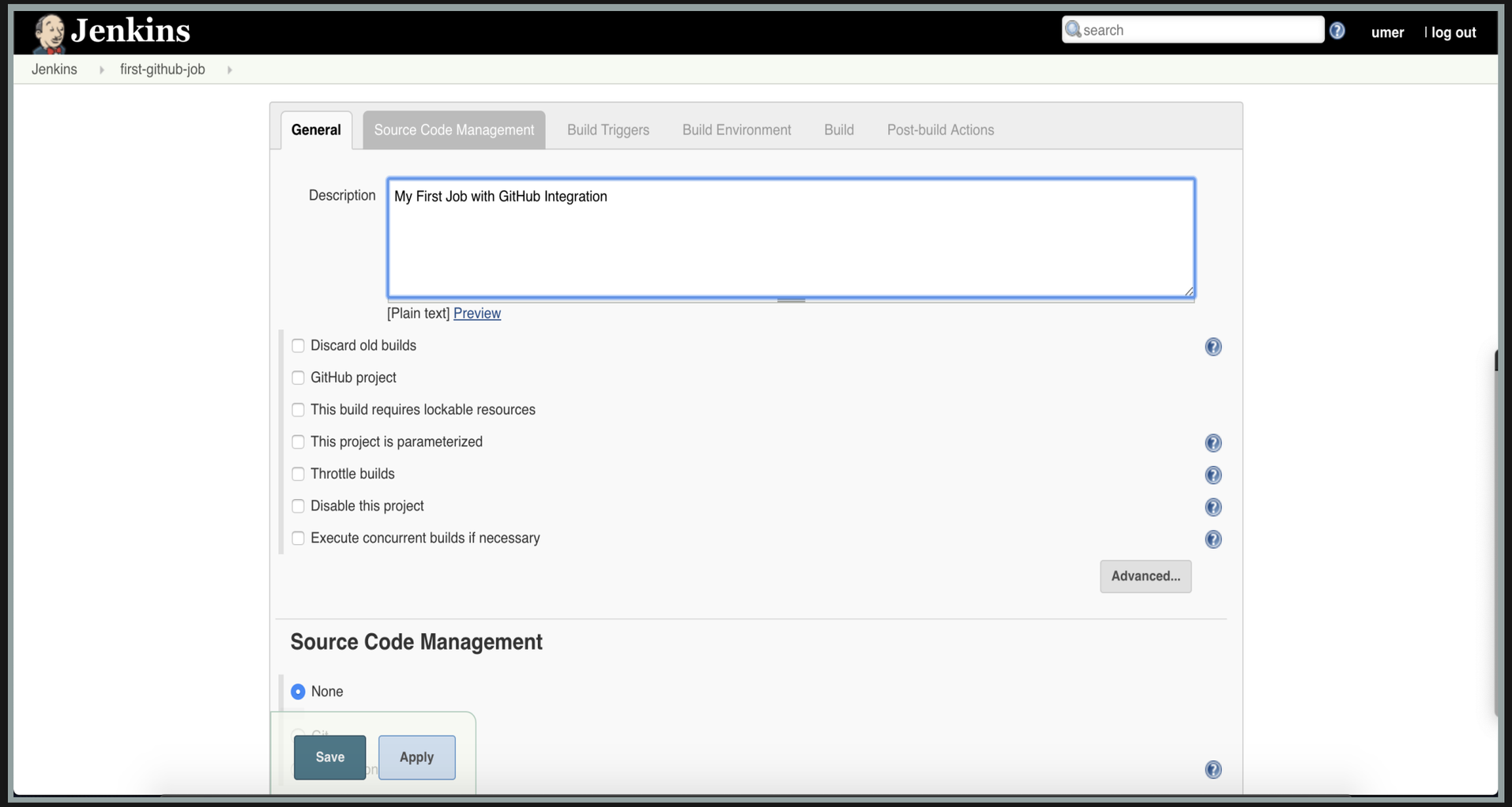
Go to Jenkins landing page by clicking on Jenkins on the left top corner and click New Item.



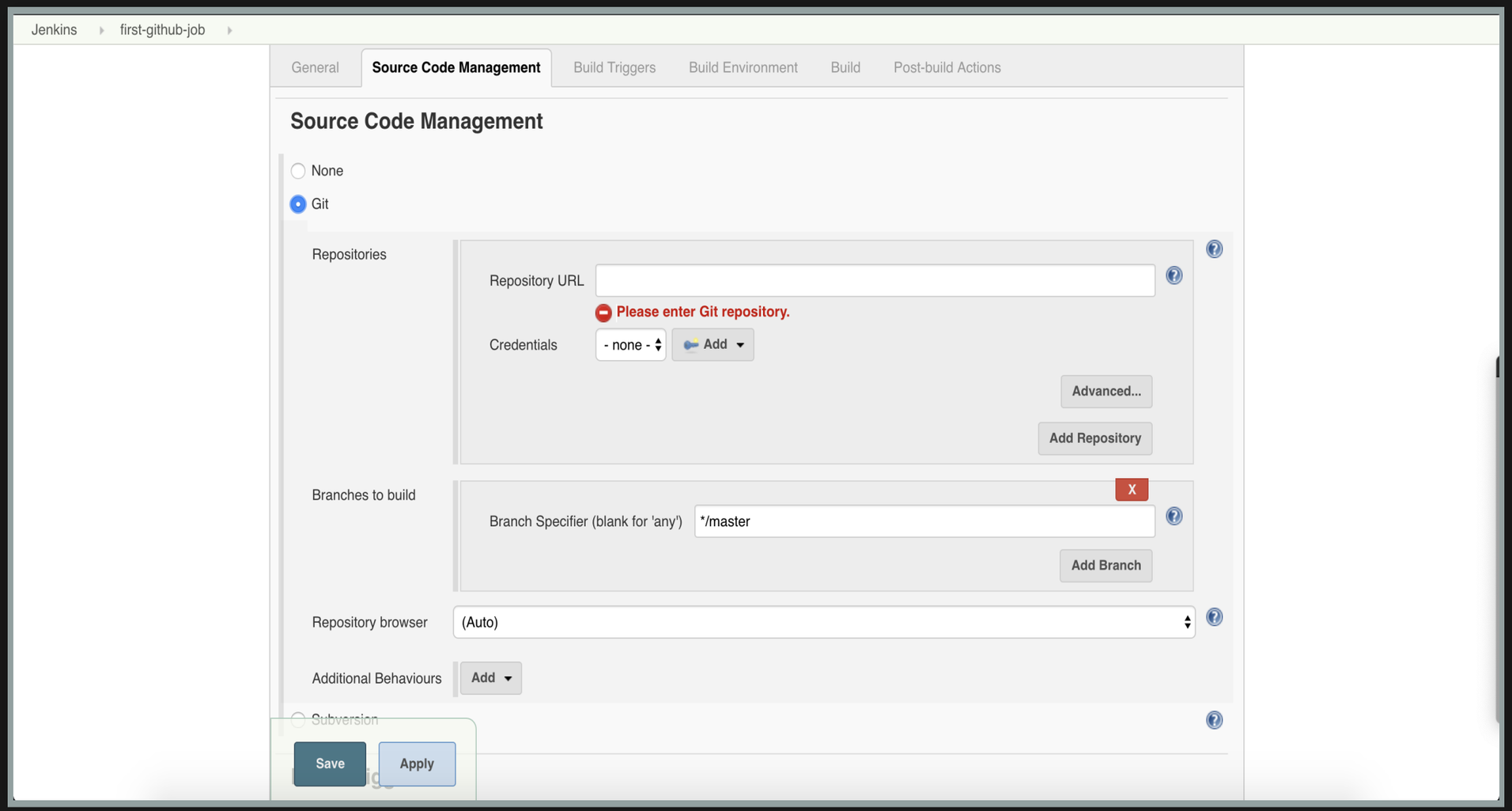
Enter the job name, select Freestyle project and Click OK



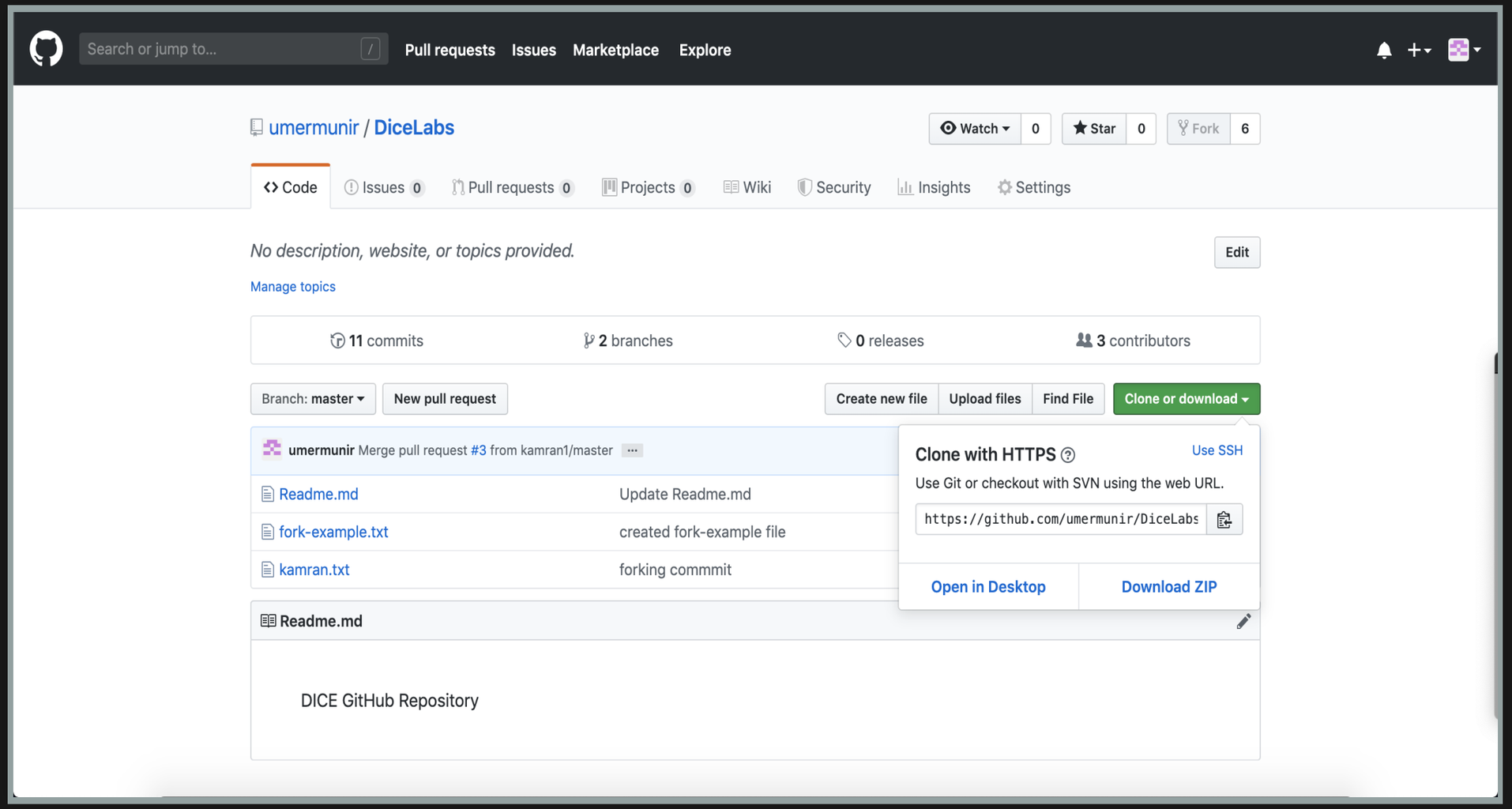
Enter job description and click Source Code Management



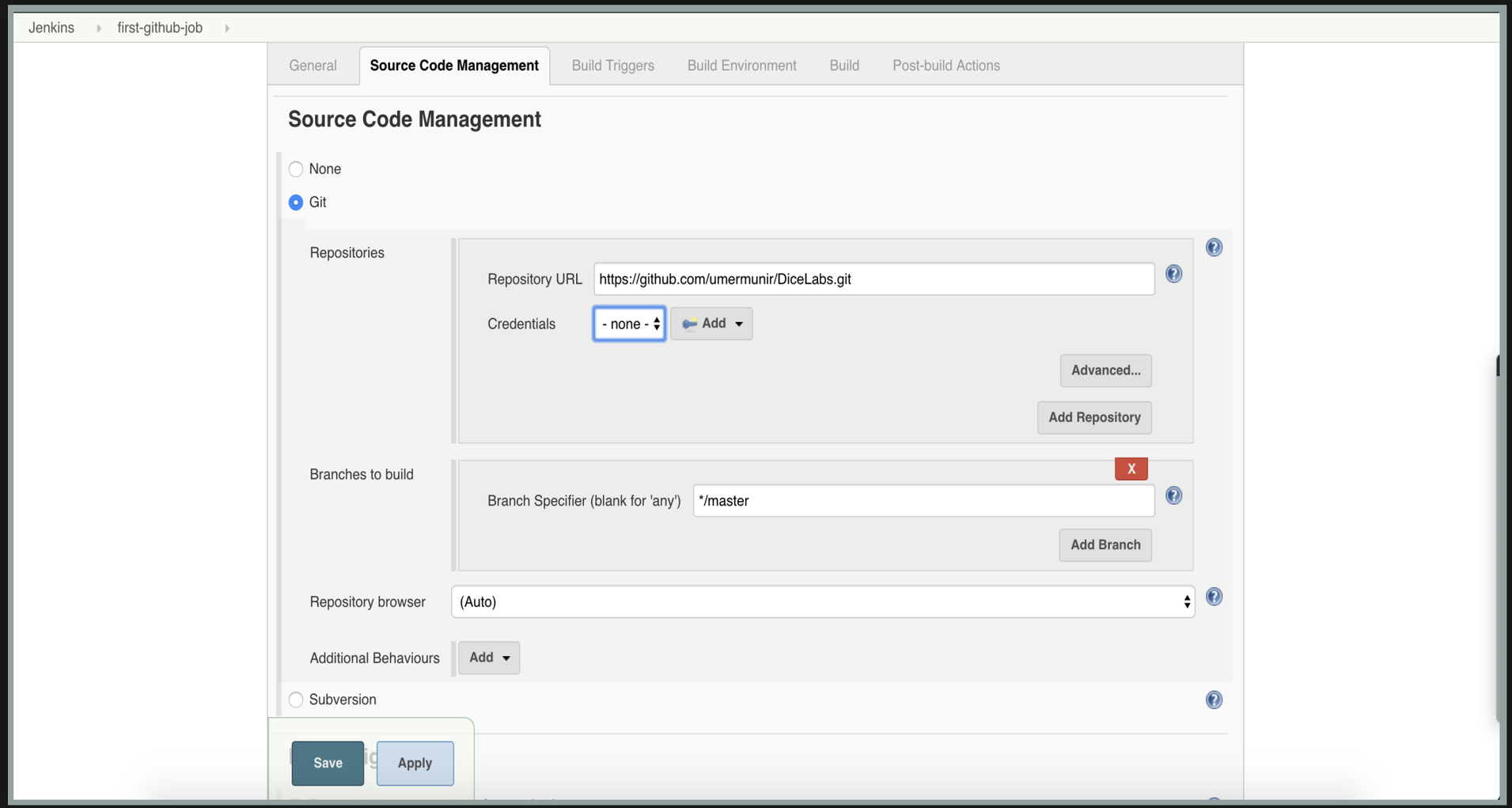
Select Git


Jenkins
first-github-job
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Source Code Management
None
Git
Repositories
Repository URL
Please enter Git repository.
Credentials
- none -
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/master
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add
Subversion
Save
Apply

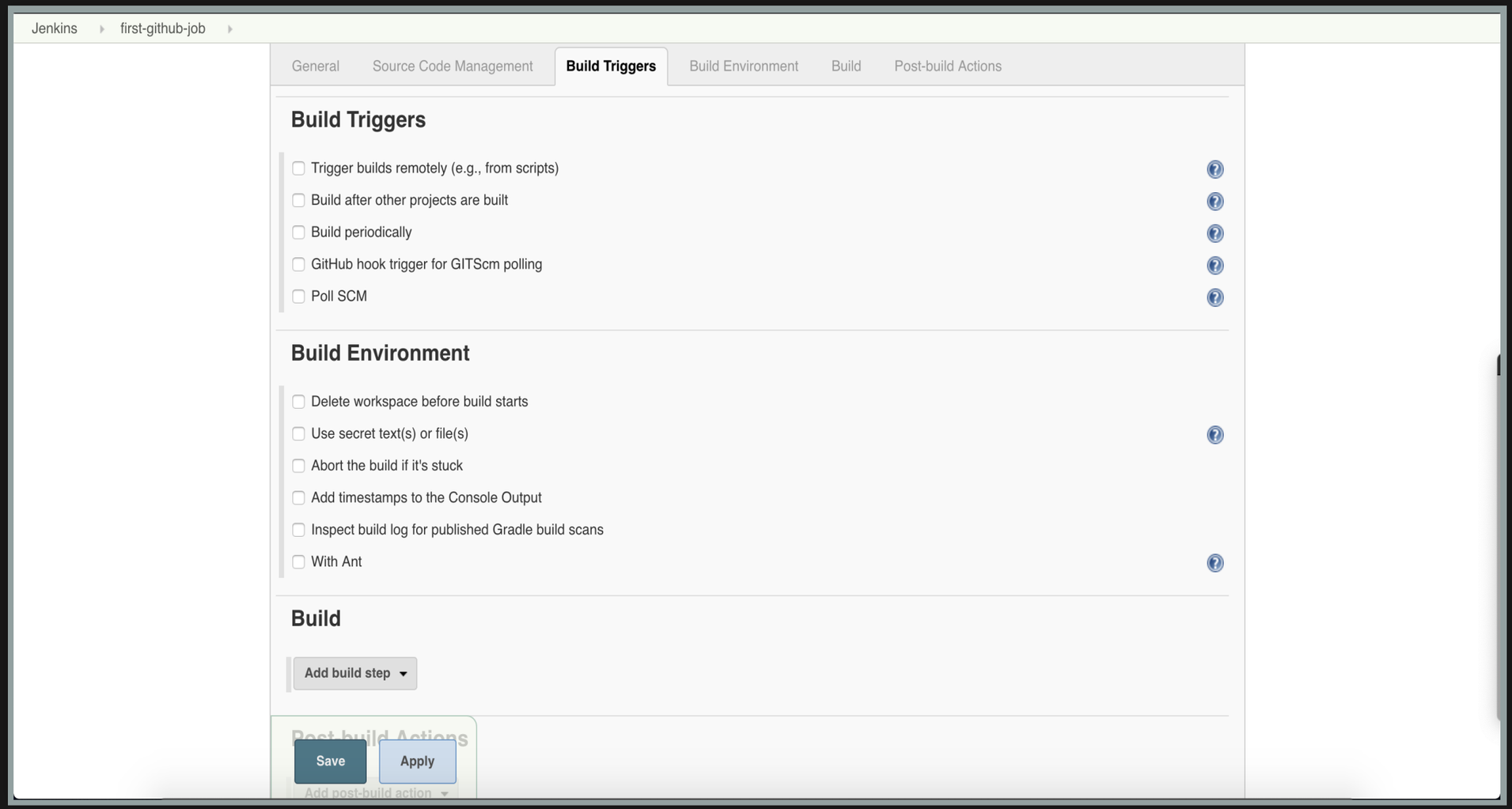
Go the repository page that you have already forked from the DiceLabs repository and copy the link as shown below



Next paste the URL in the Repository URL as shown below

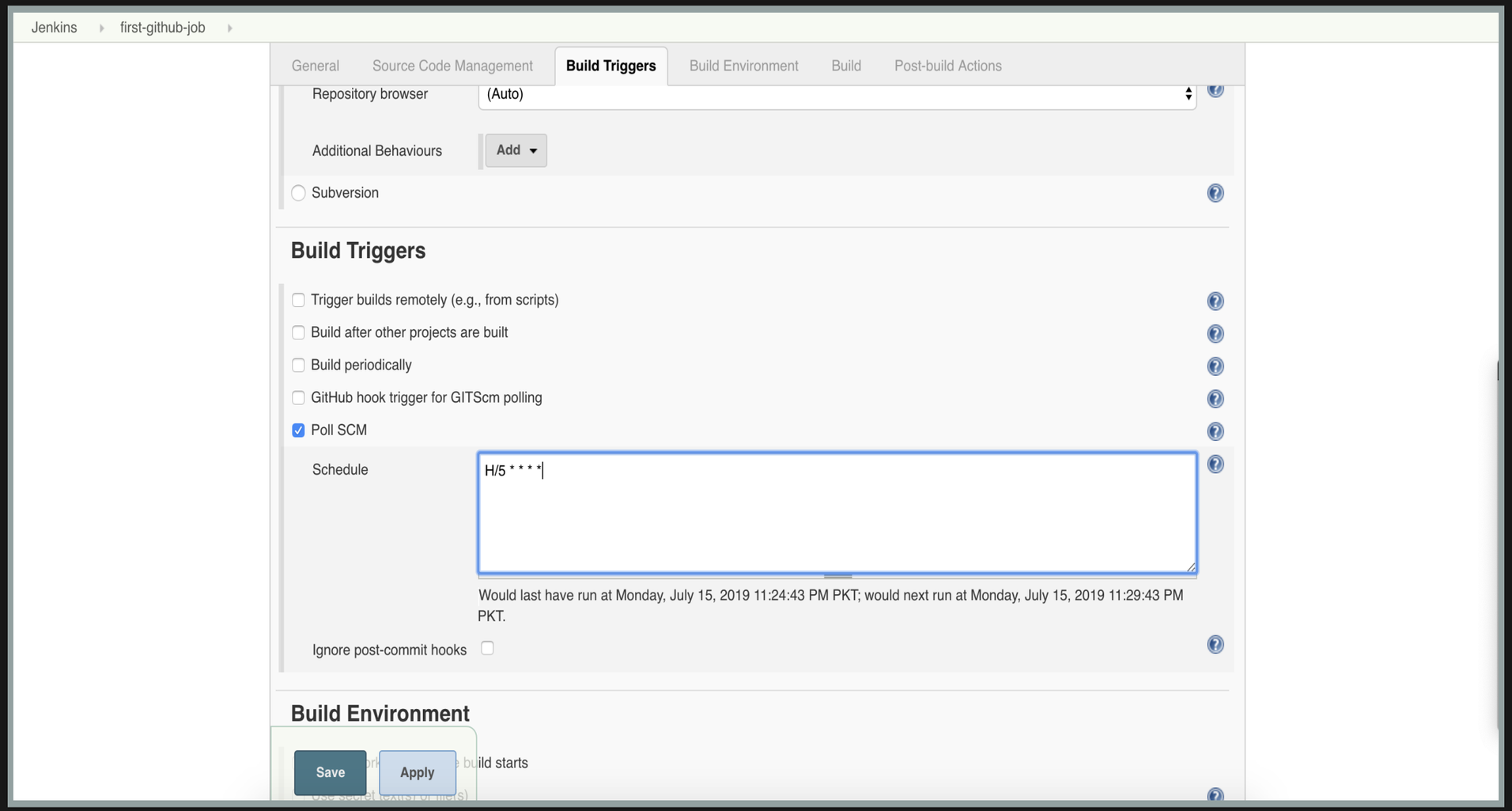


Click Build Trigger to specify the triggers for this job

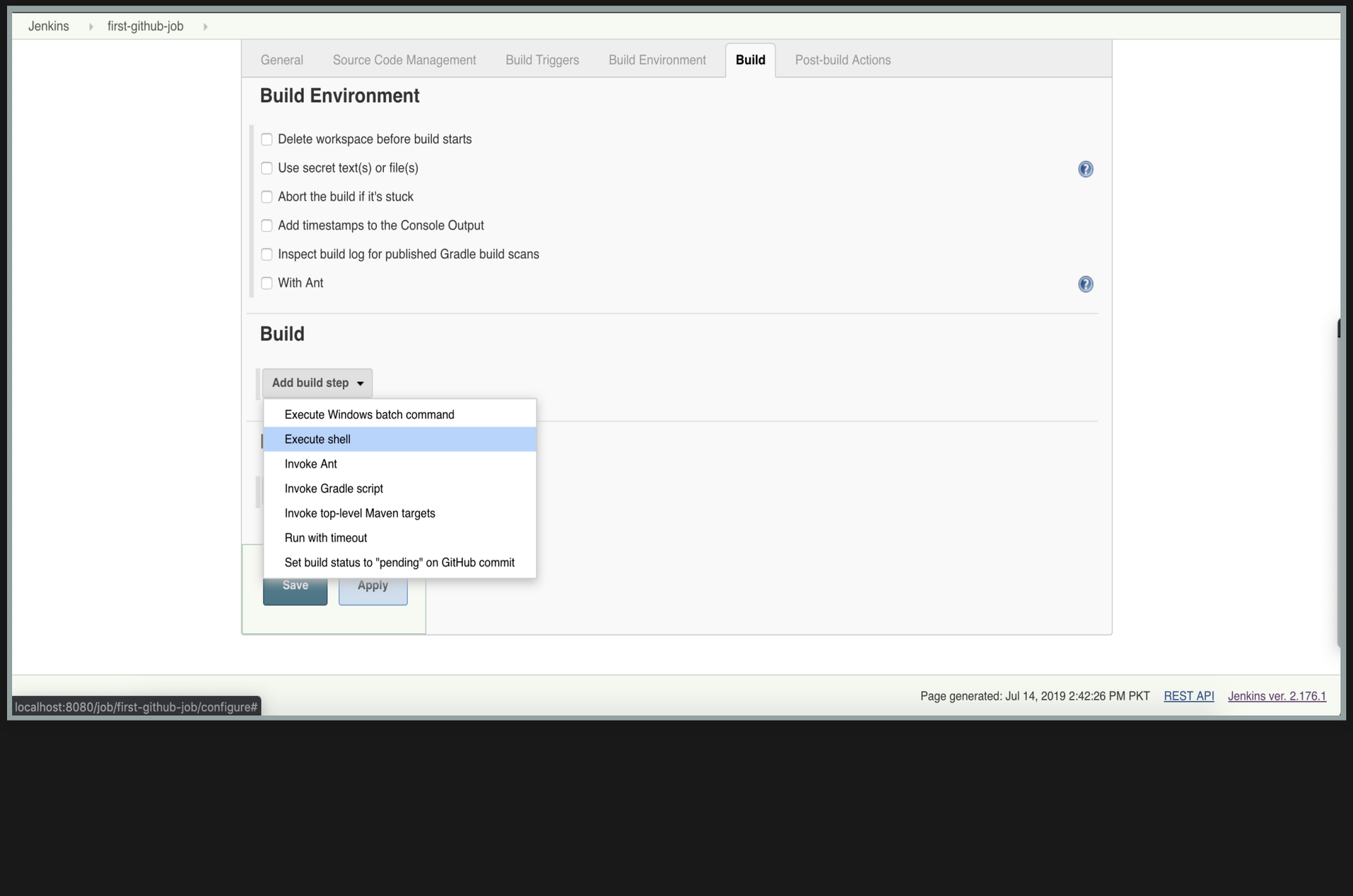


For this job you are going to select Poll SCM, which means you are going to poll GitHub as per the specified schedule. For this job we want Jenkins to poll GitHub to check for any commits every five minutes

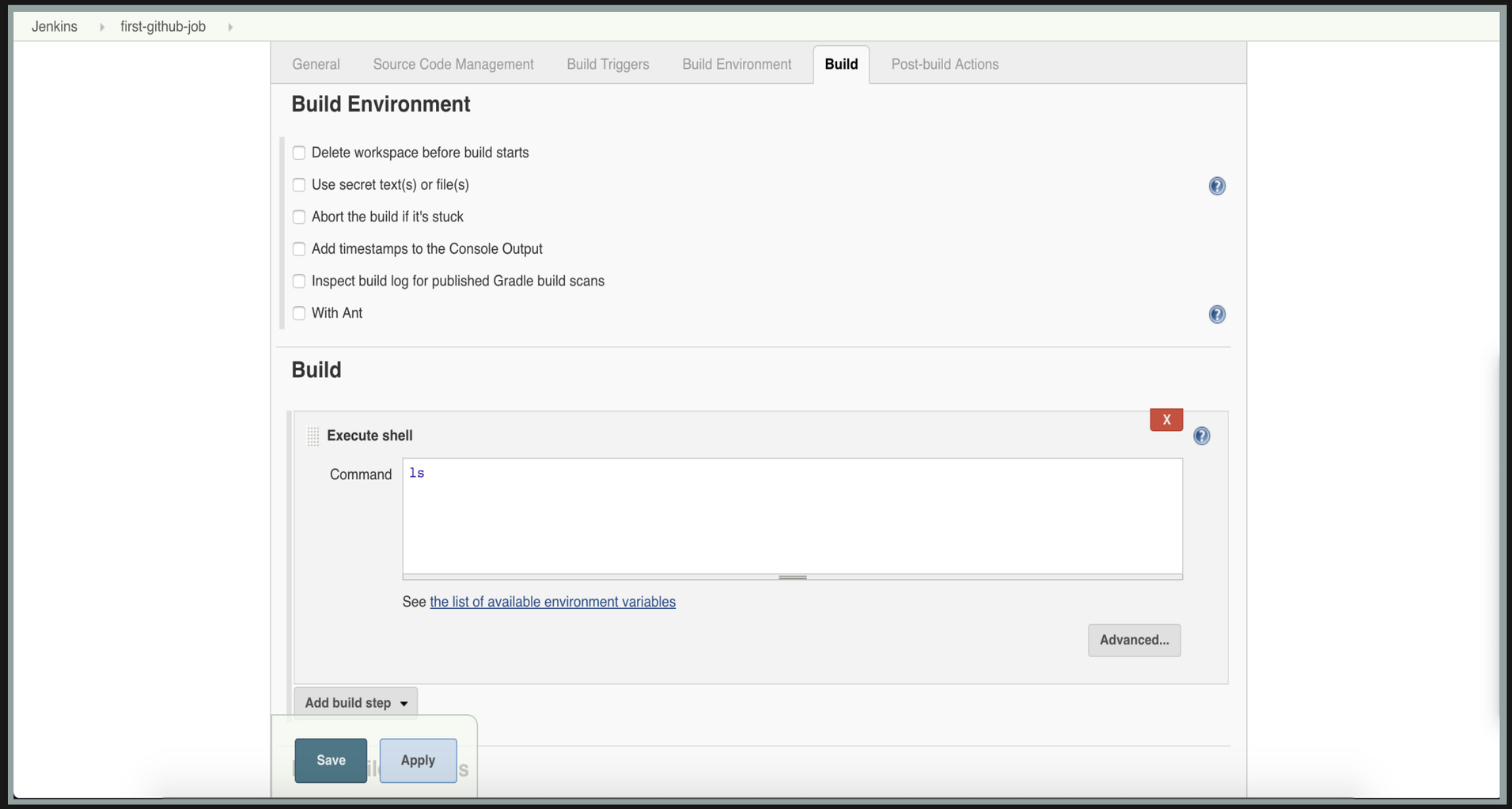
```
H/5 * * * *
```



Click Build and Select Execute Shell



Specify ls as the command and Click Save



Jenkins

search

umer | log out

Jenkins › first-github-job ›

ENABLE AUTO REFRESH

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Delete Project
- Configure
- Rename

Build History trend

find x

RSS for all RSS for failures

Project first-github-job

My First Job with GitHub Integration

edit description

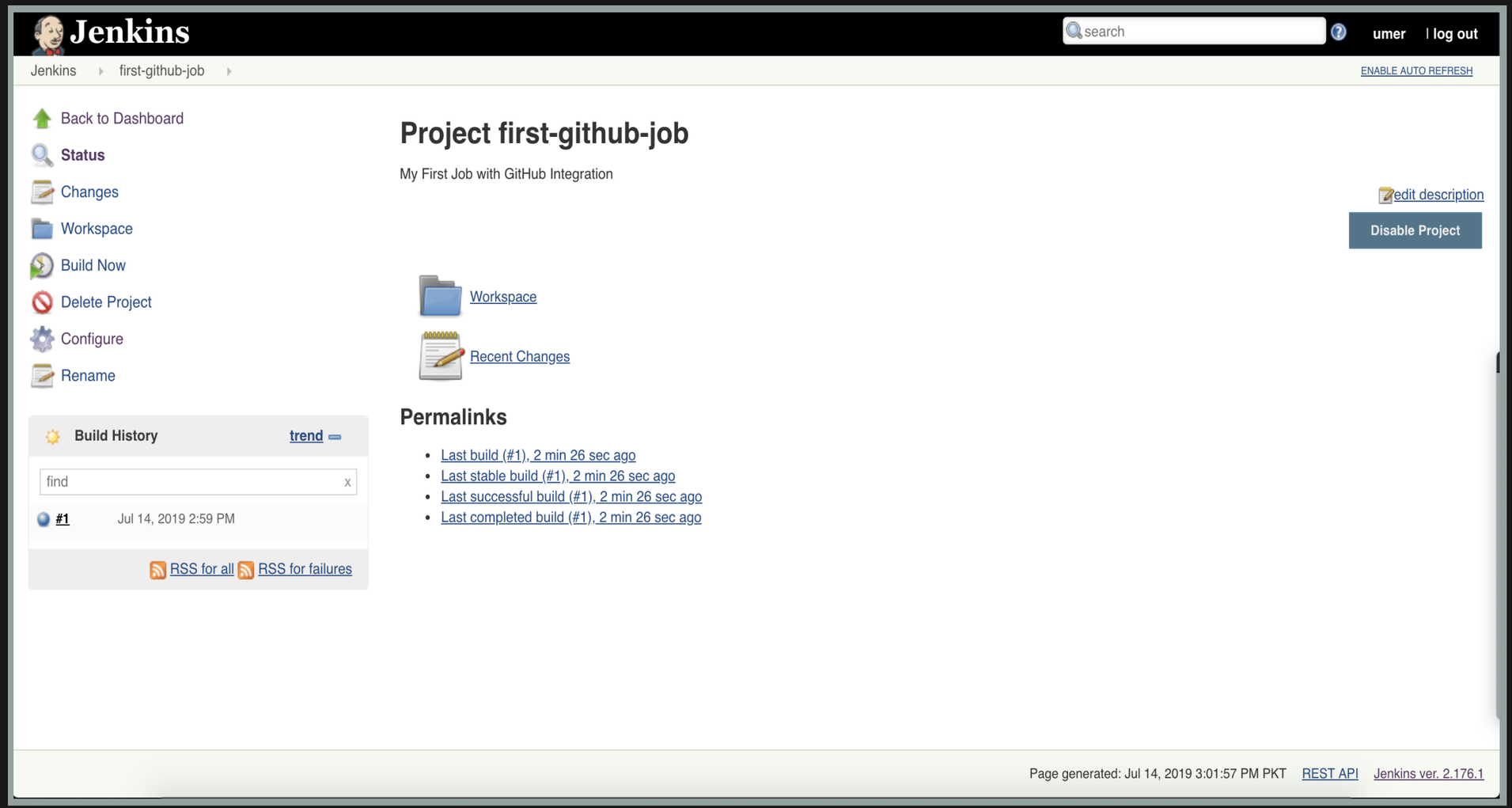
Disable Project

Workspace

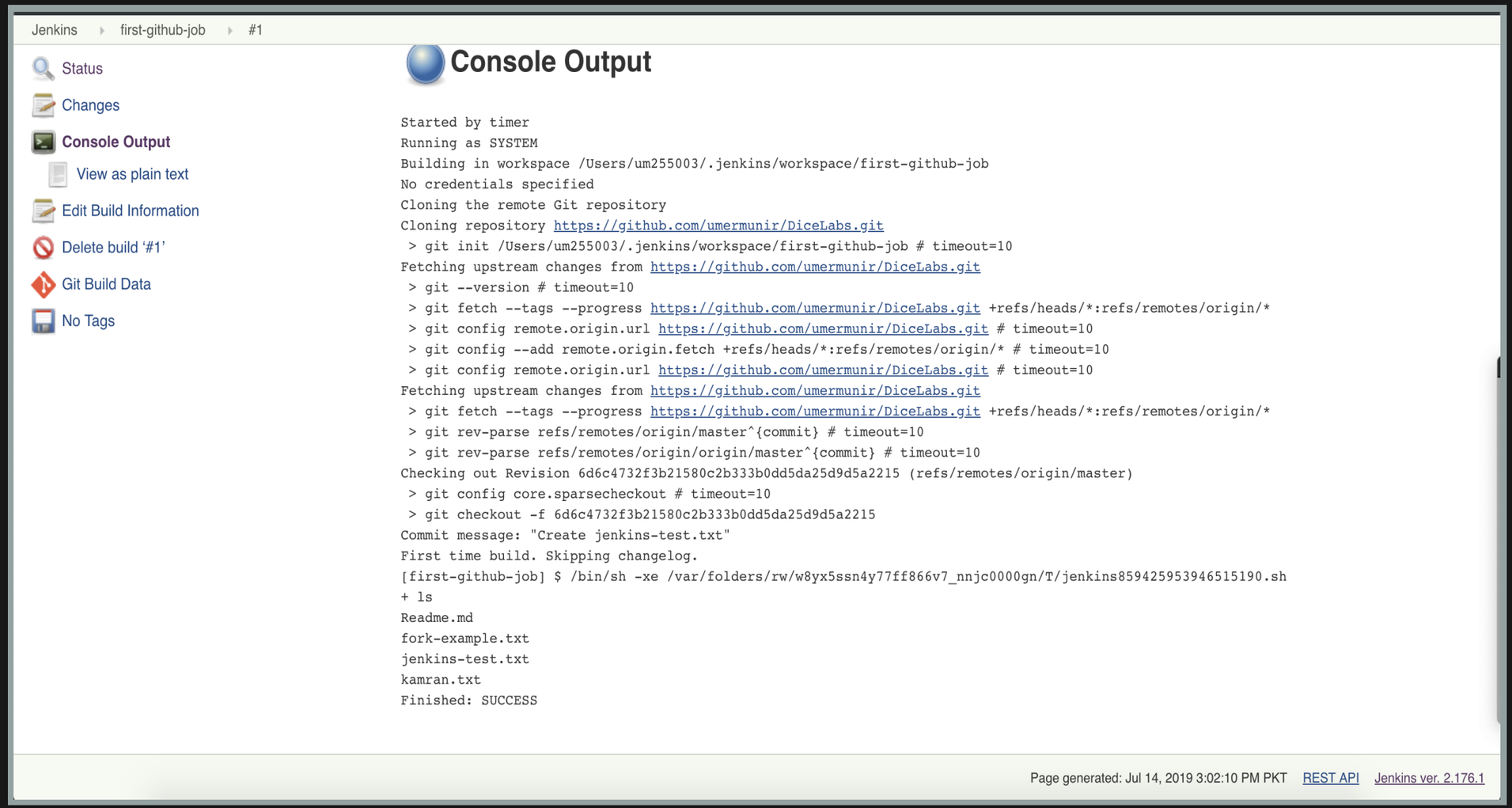
Recent Changes

Permalinks

Next create a file in the repository that you entered in the Source Code Management section and push that change to GitHub. You should see that it will trigger your job

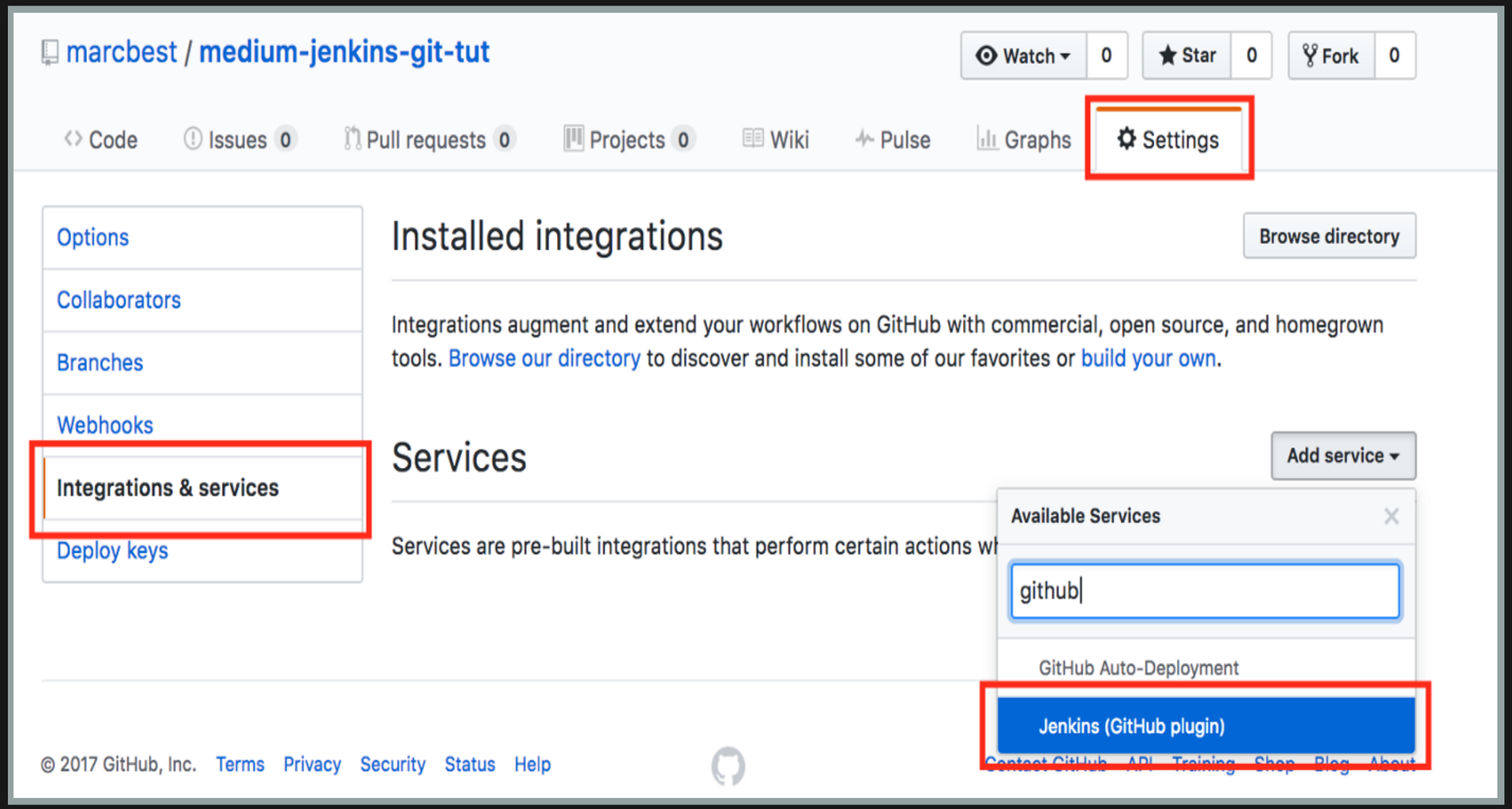


Check the Output Console of build, you should see logs as shown below

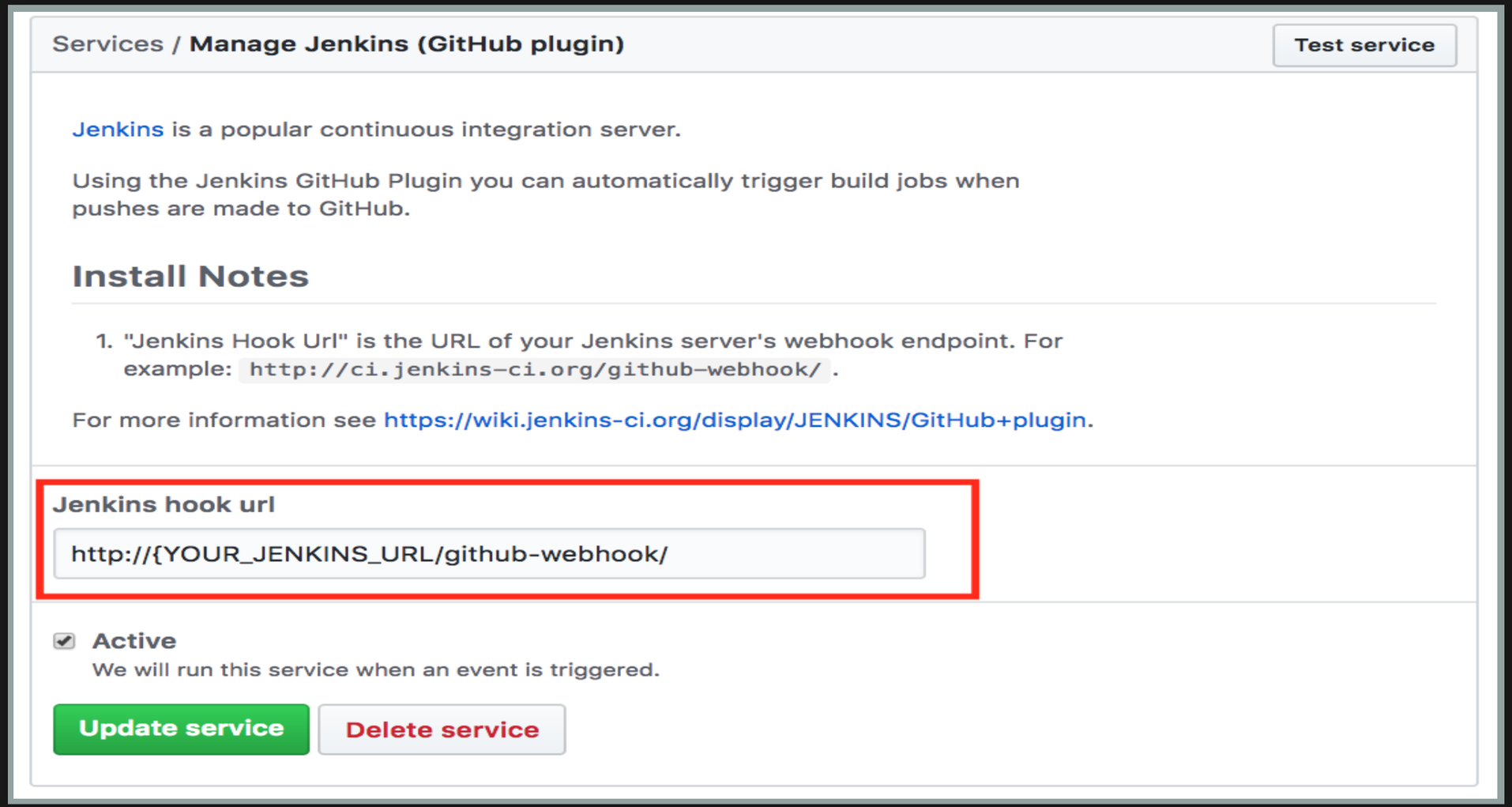


GITHUB WEBHOOKS

We need to add a service to call the Jenkins Github webhook on a push, to do this go to settings -> integrations & Services and add a new service. The Jenkins Github plugin service should be in the list of available services.



Enter the URL of your Jenkins instance followed by /github-webhook/



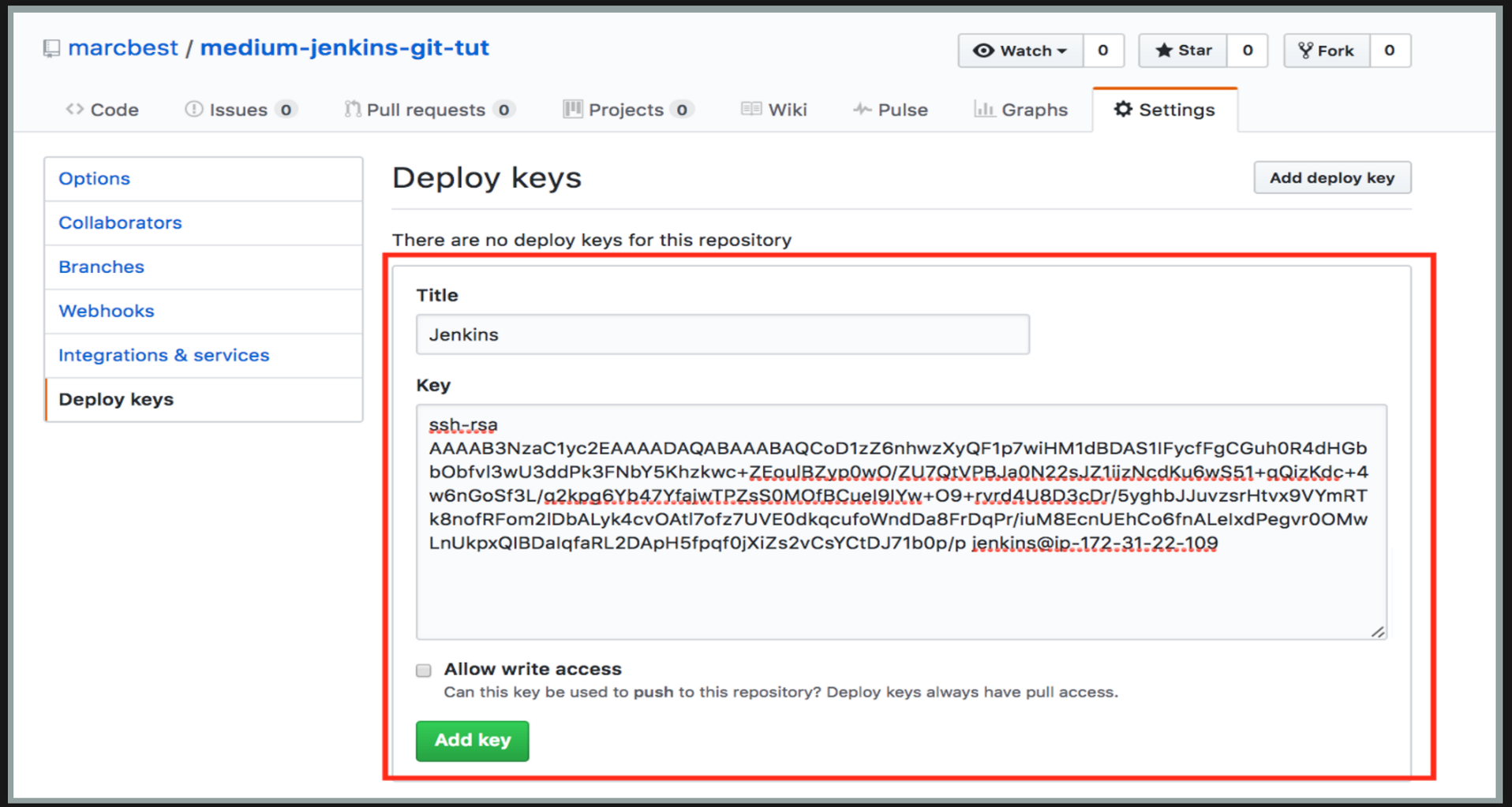
We need to give the Jenkins user access to our repository by adding a deploy key in the Github settings. For that you will create an SSH key

```
jenkins@ip:/home/ubuntu$ ssh-keygen
```

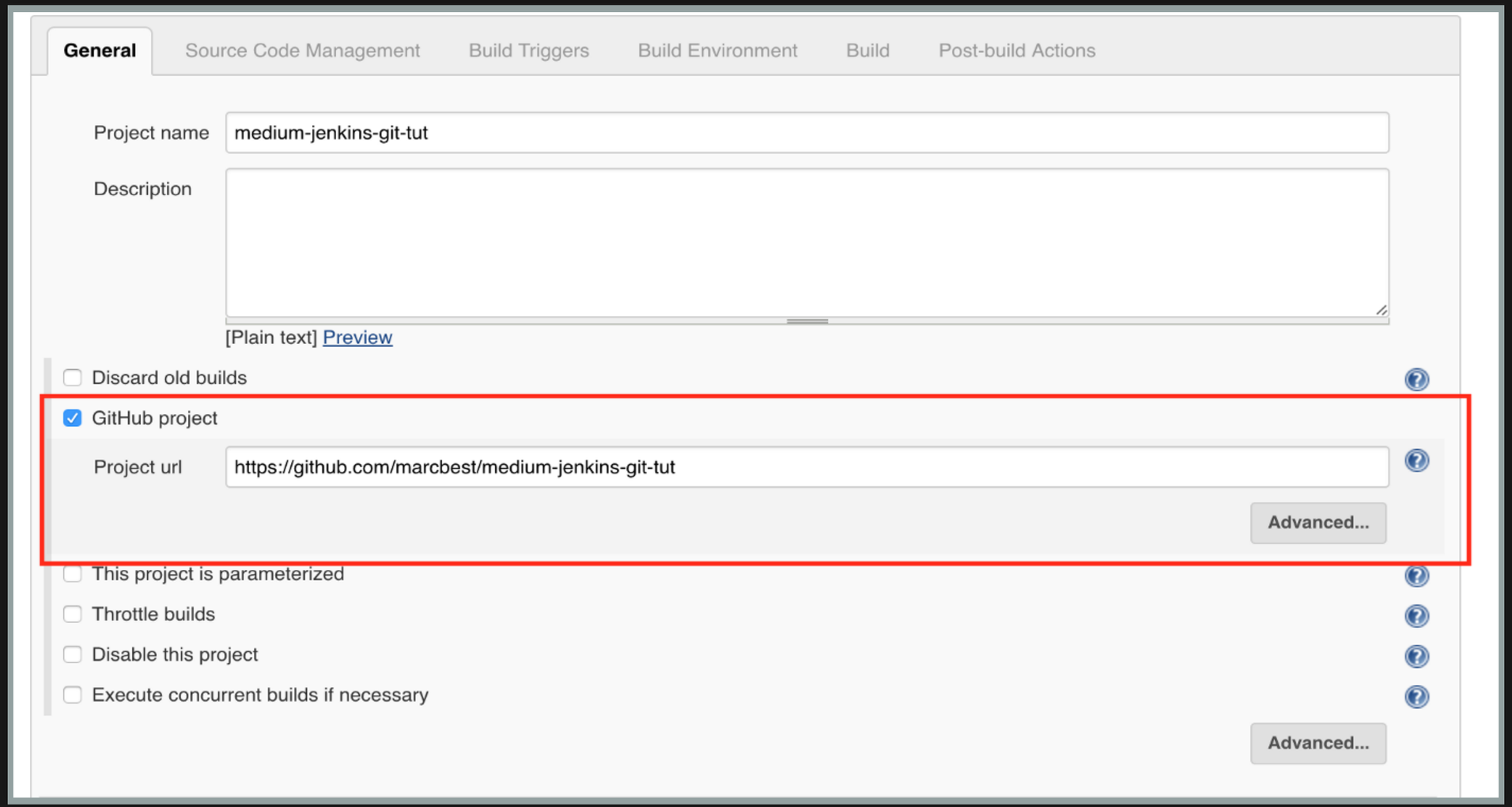
Copy the public key so that it can be added to Github

```
jenkins@ip:/home/ubuntu$ cat ~/.ssh/id_rsa.pub
```

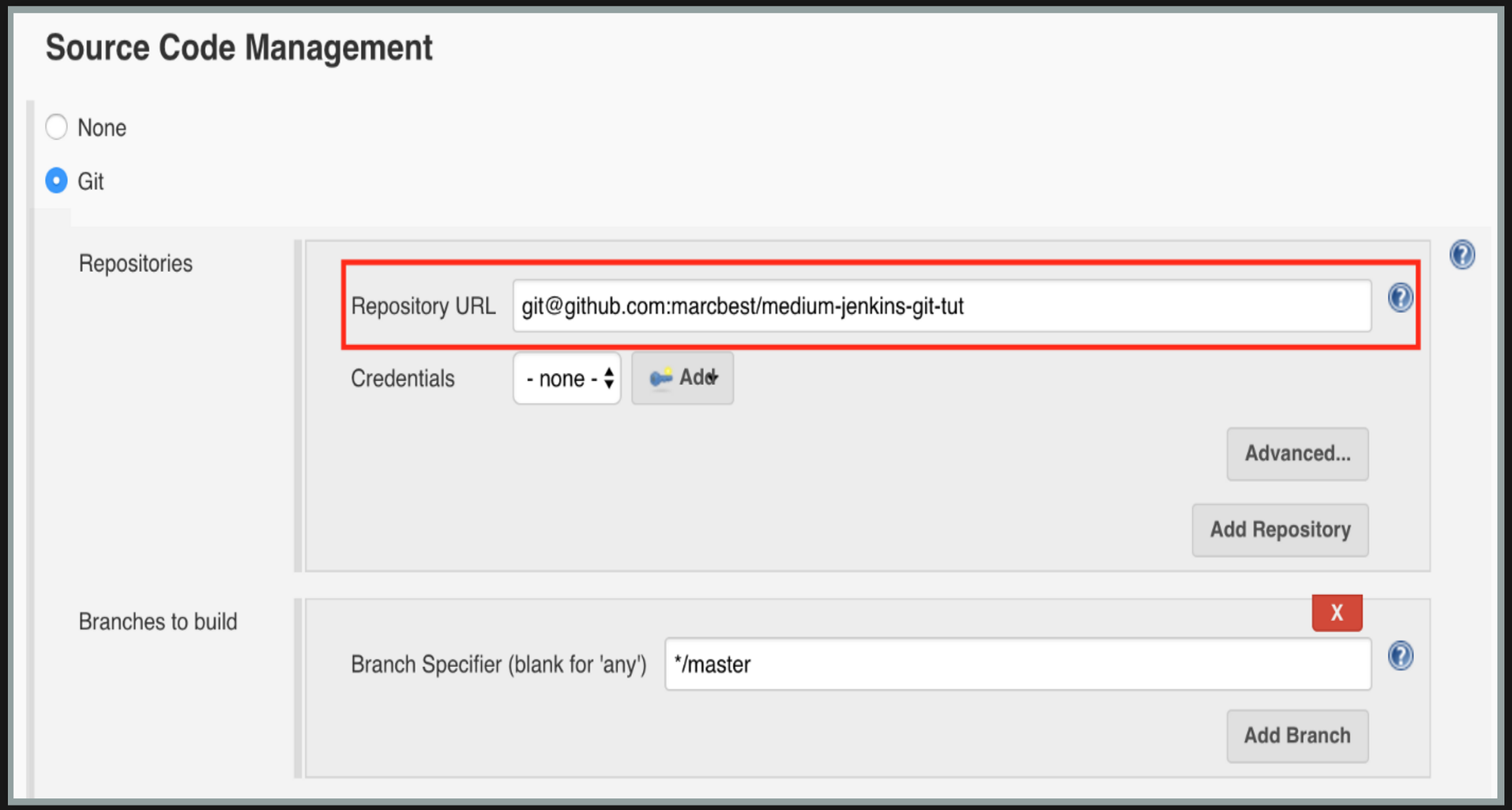
Add the key copied in the previous step to Github. To do this head to the repository settings -> Deploy keys



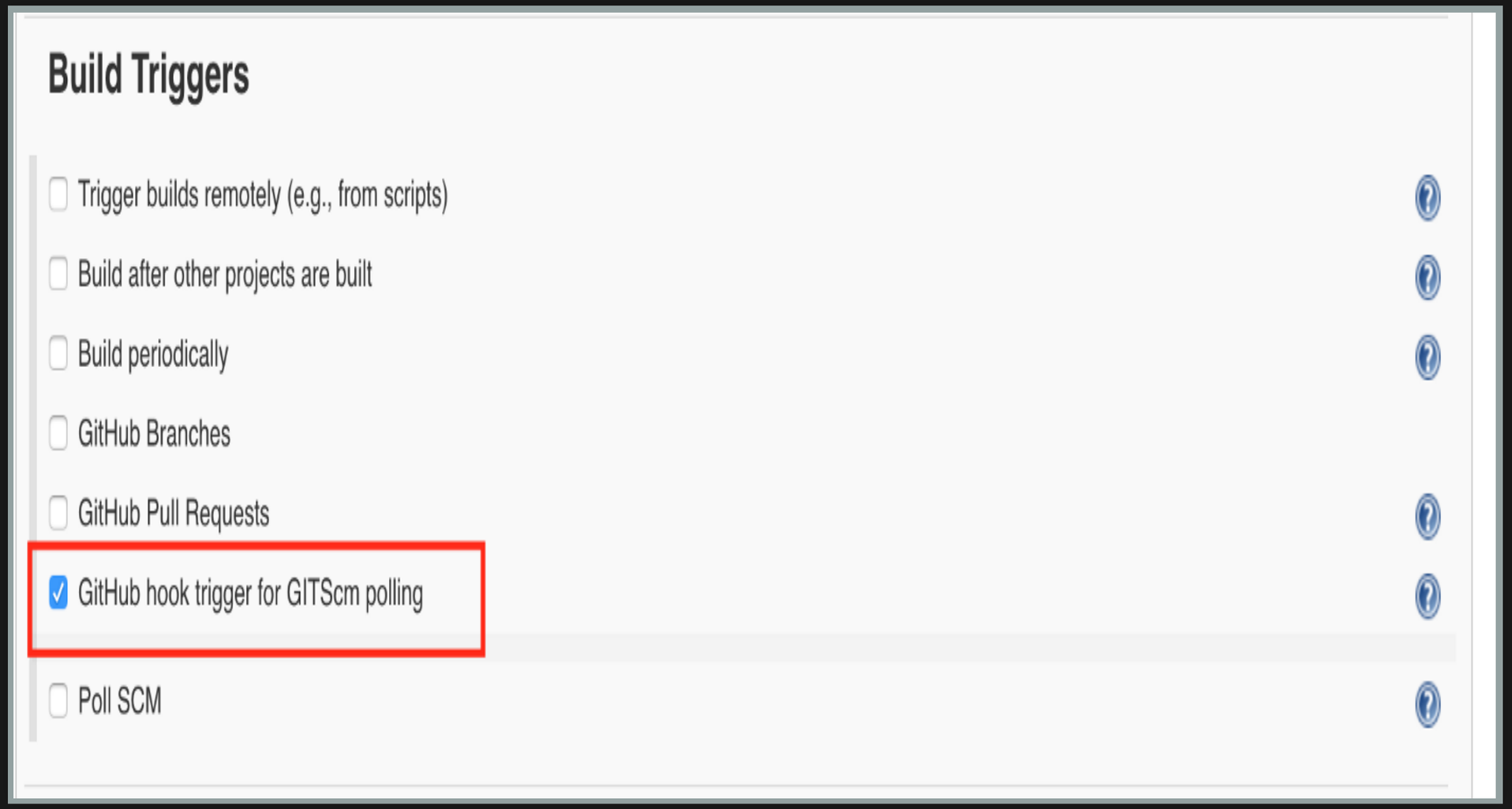
In the General section of the job configuration check the Github project tick box and enter the URL to the repository



Next update the Source Code Management section, first set the repository URL (note the format git@github.com:{YOUR_REPO}.



The last step is to tell Jenkins to build when the Github hook is called

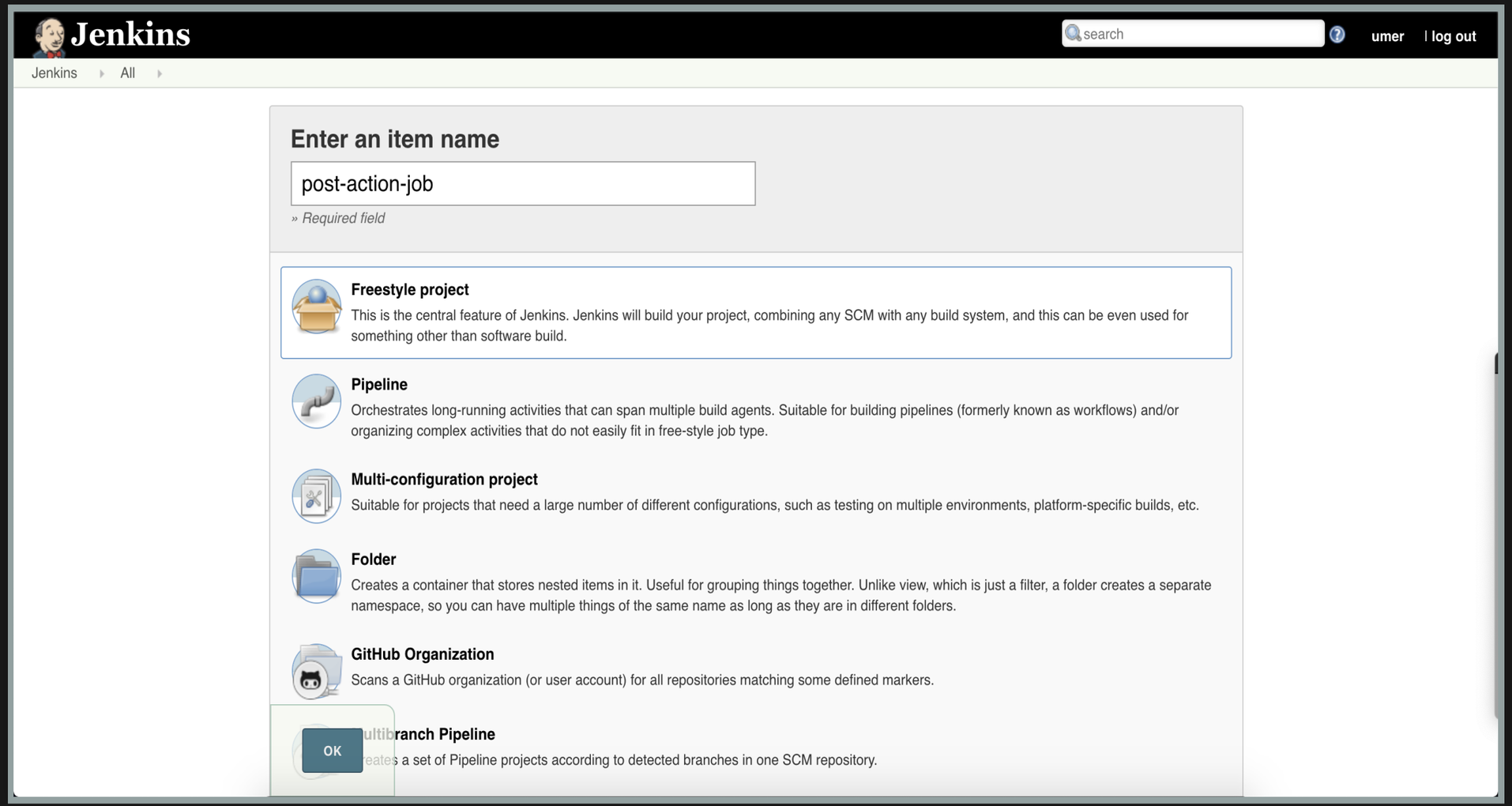


POST BUILD ACTION

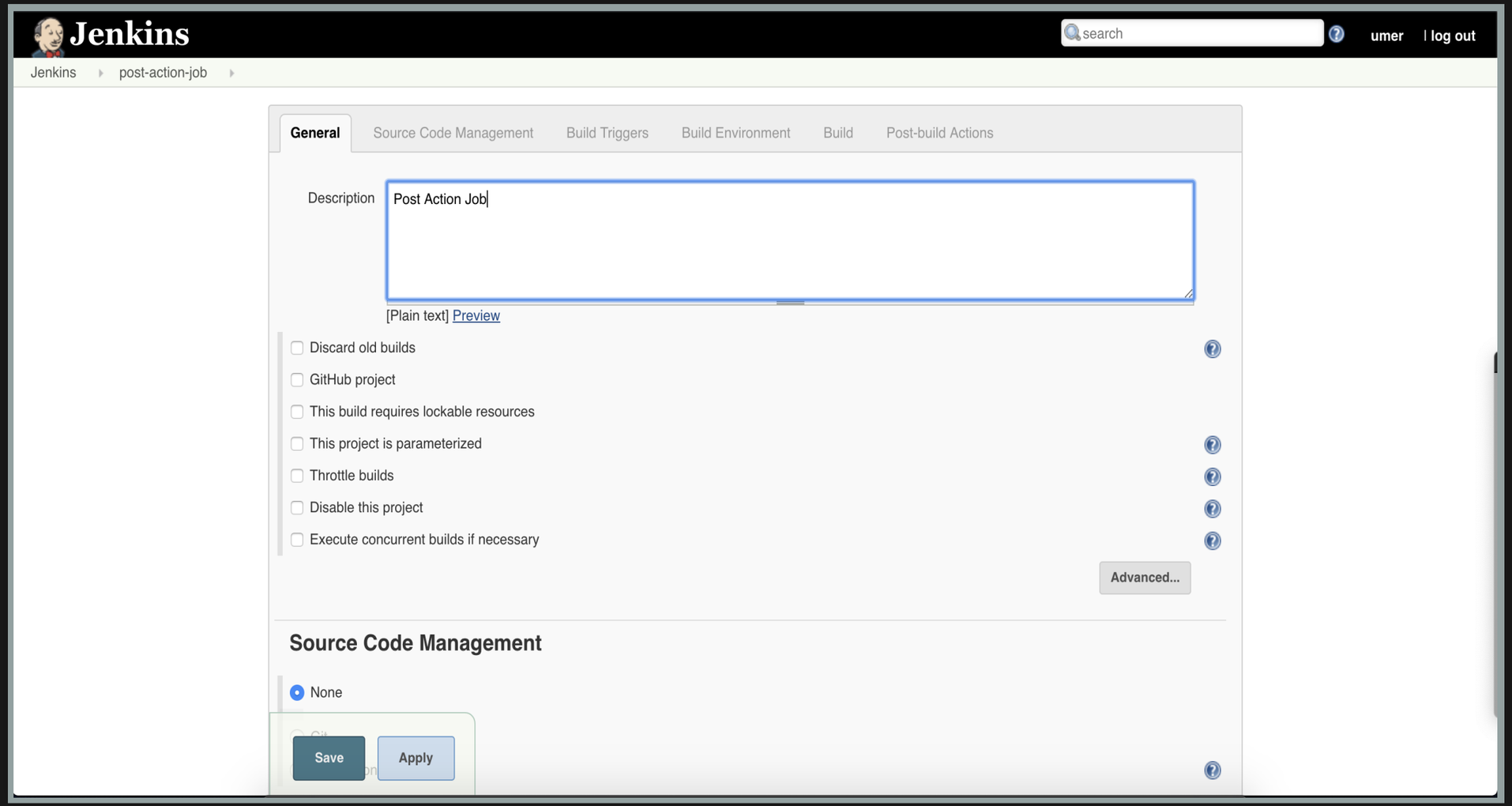
- Allows user to run tasks after build
- Rich set of Plugins
  - Email Notification
  - Slack Notification
  - Build Other Project
  - Archive Artifacts
  - Build Status on GitHub

LAB - EMAIL

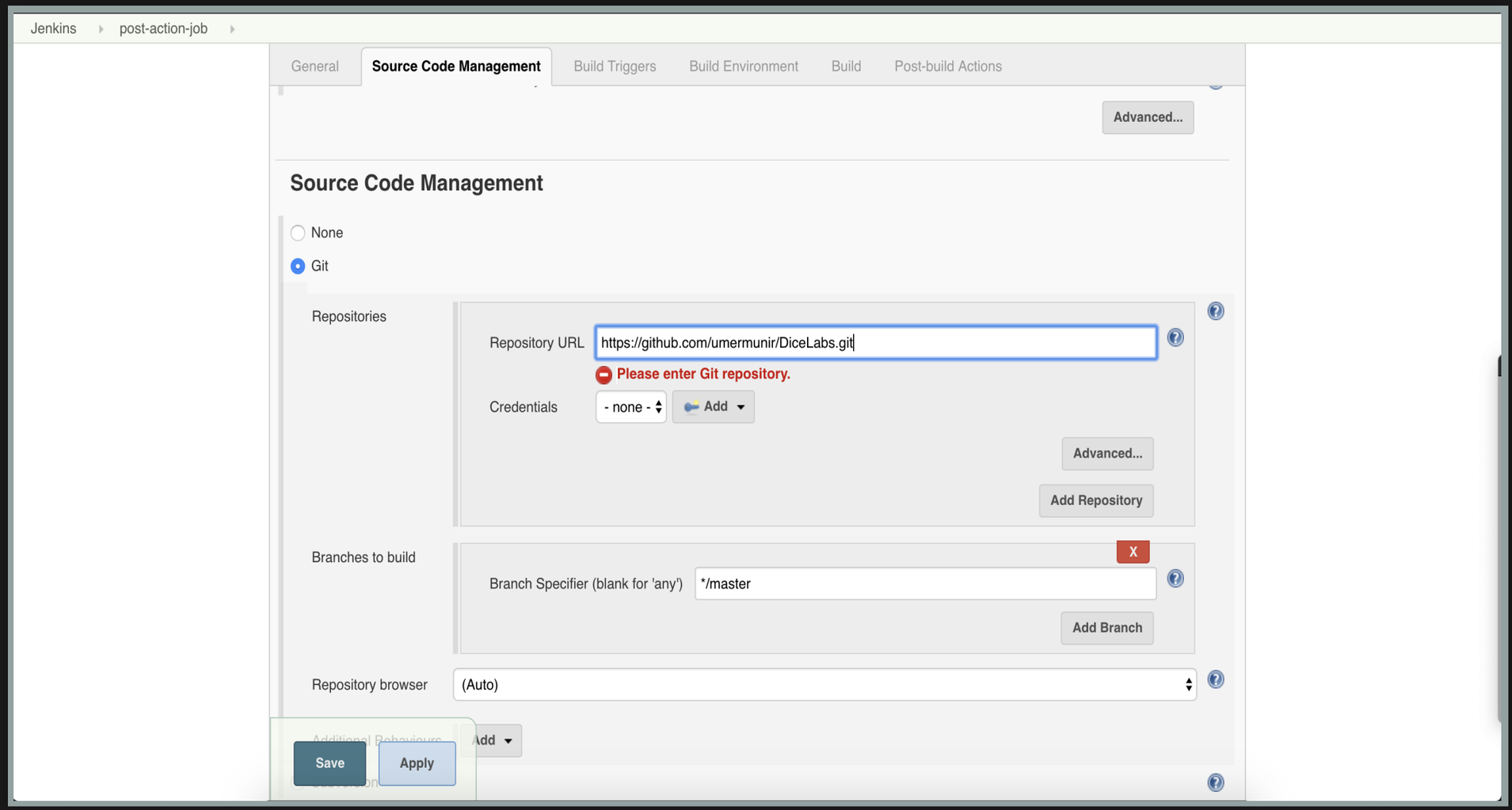
In this Lab you are going to create a job to send email notification when the build fails. Create New Item, Enter the job name, select Freestyle Project and Select OK



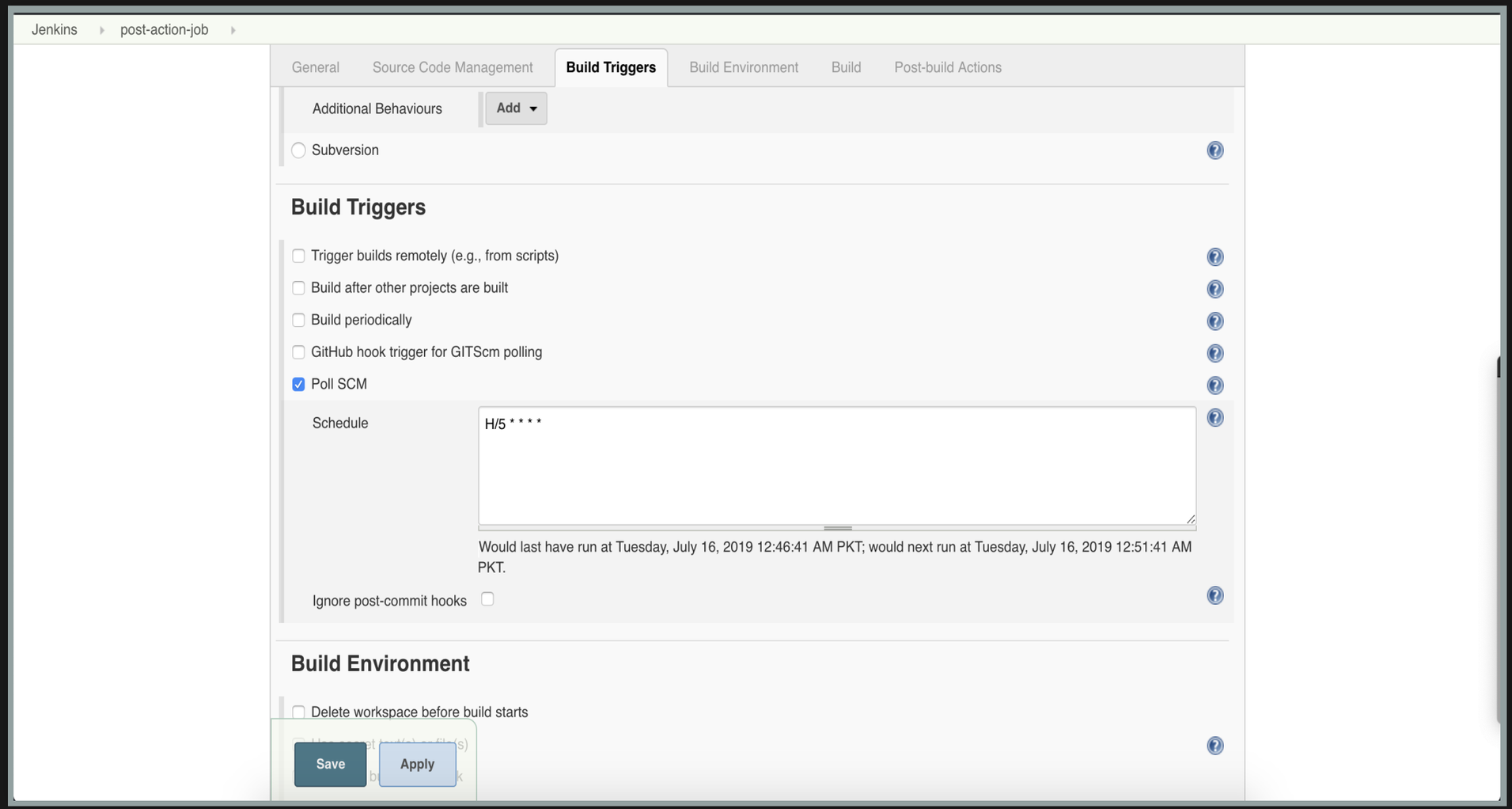
Enter Job Description and Click Source Code Management



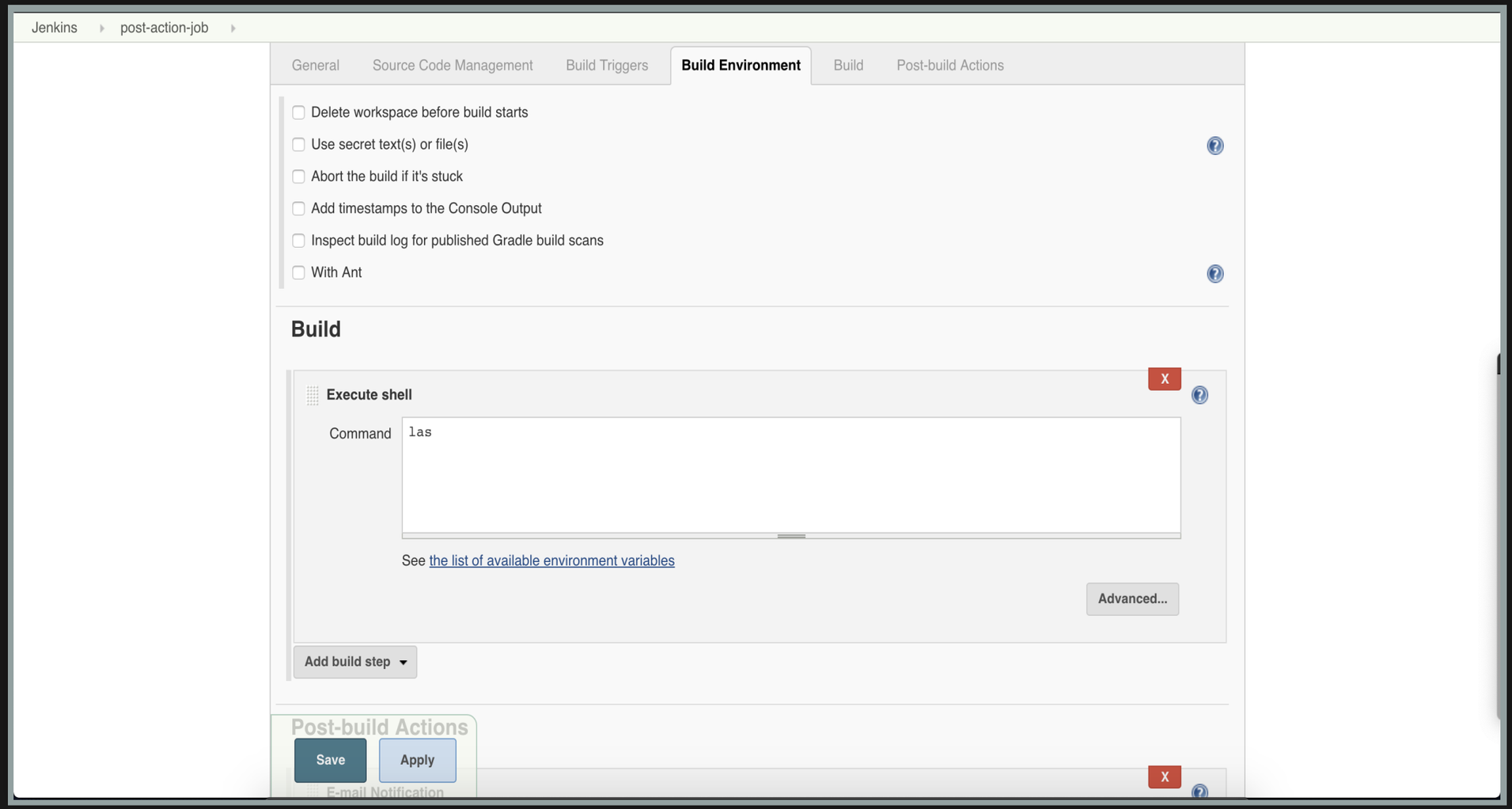
Enter the URL of repository that you forked from DiceLabs repository and Click Build Triggers



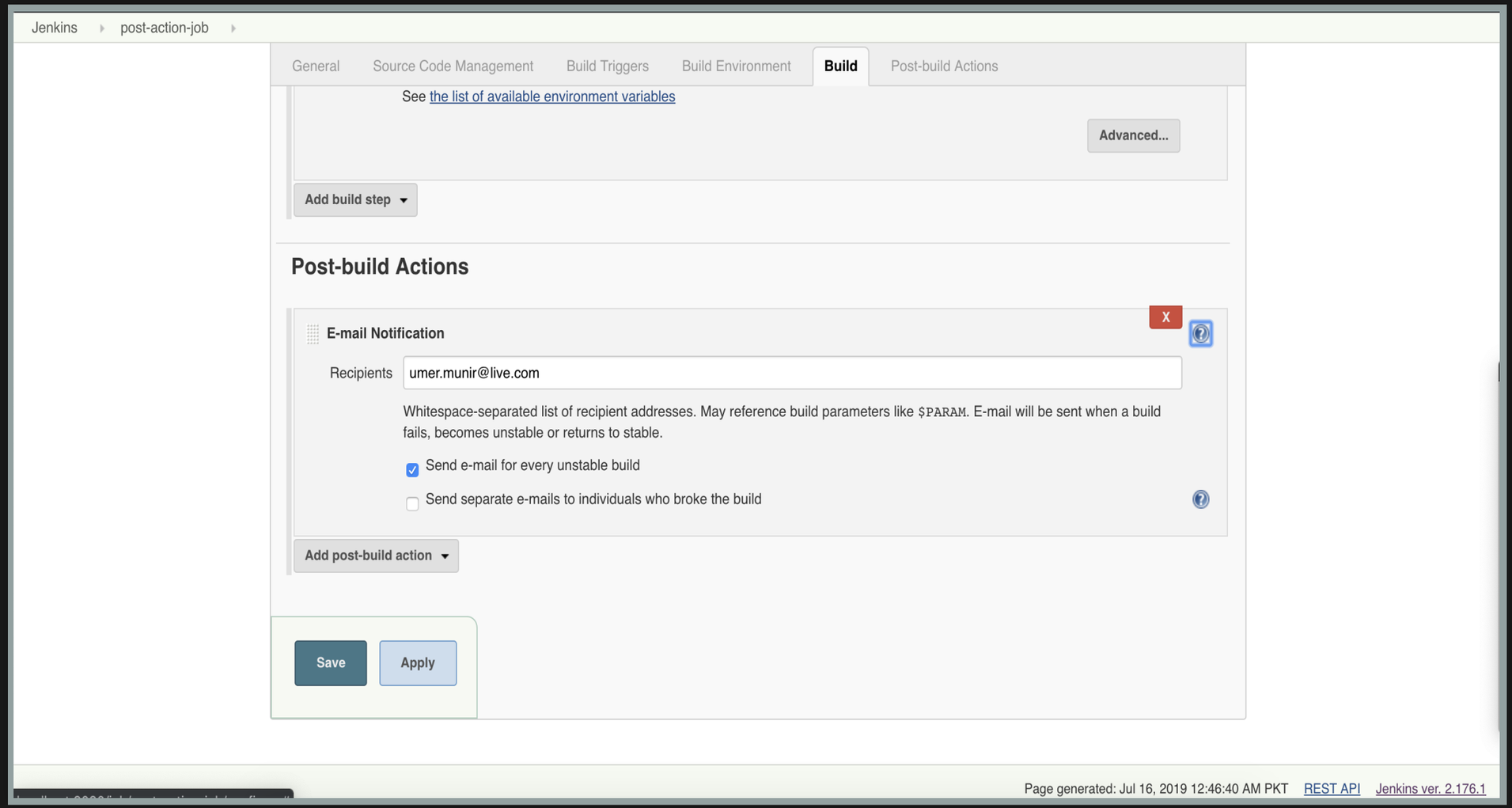
Select Poll SCM and specify the schedule as shown below and Click Build



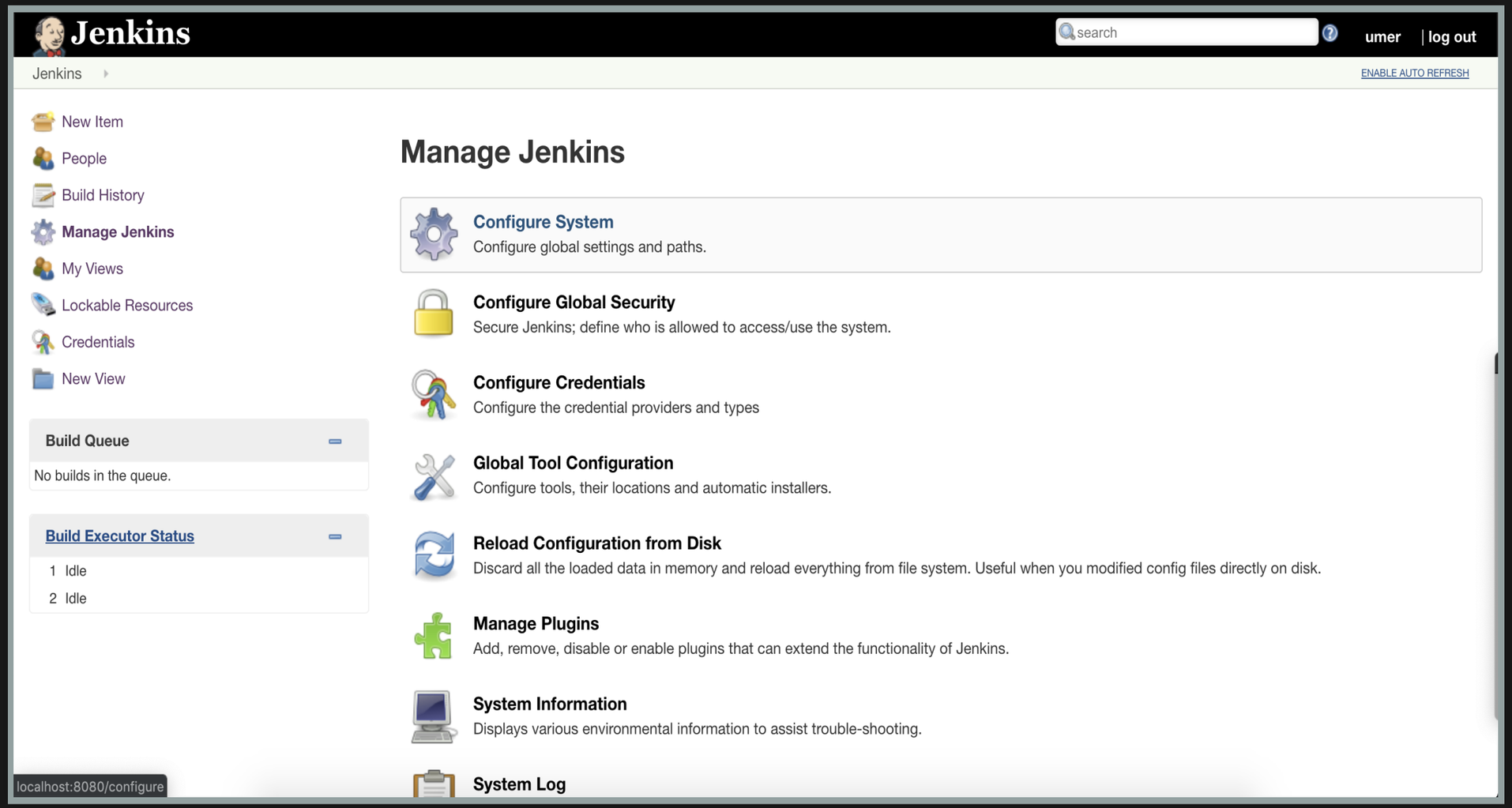
Enter a incorrect command as shown below so that we can have a failed build



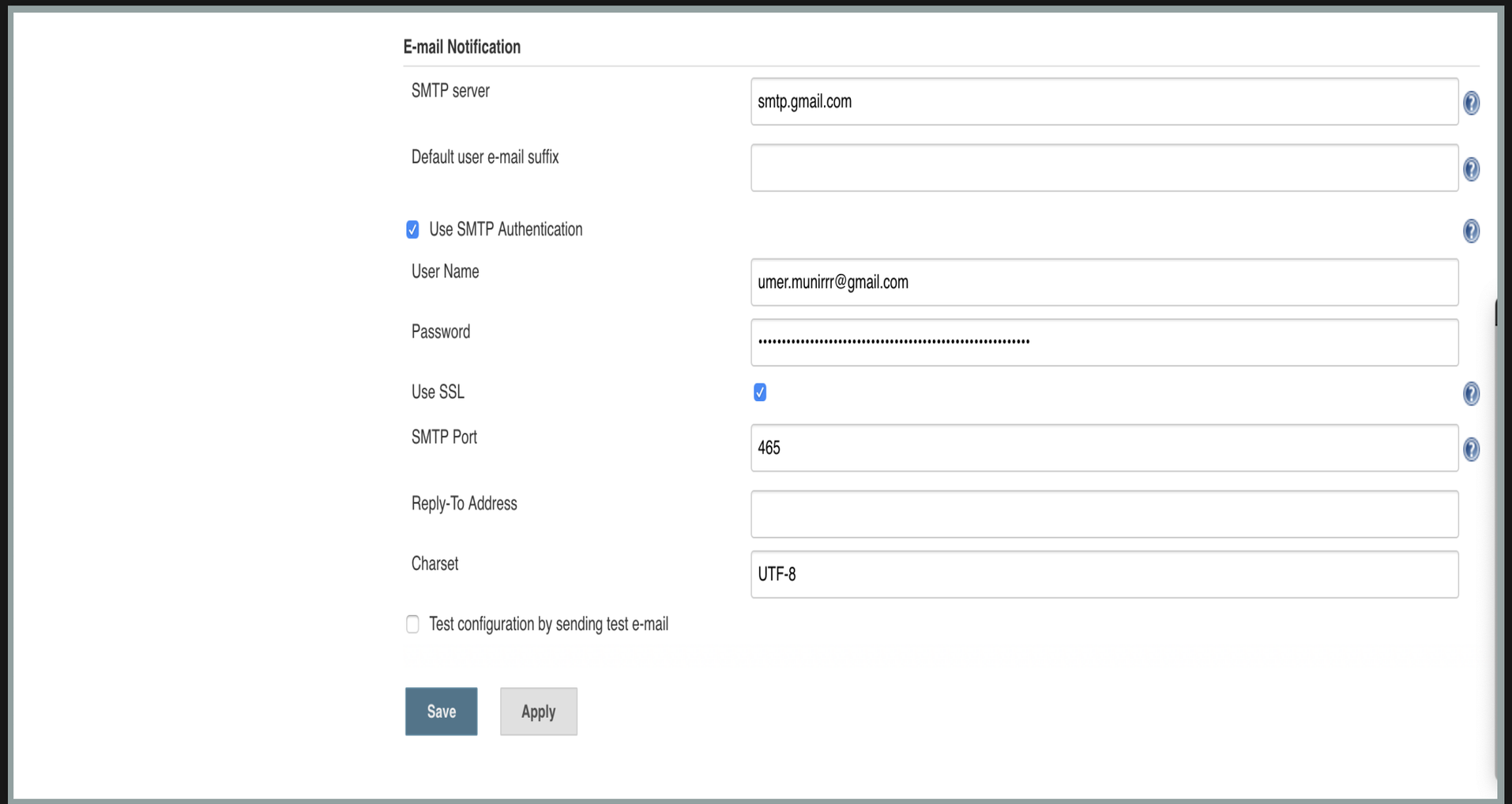
In the Post-build Actions select Email Notification, enter your email address and select 'Send email for every unstable build' as shown below



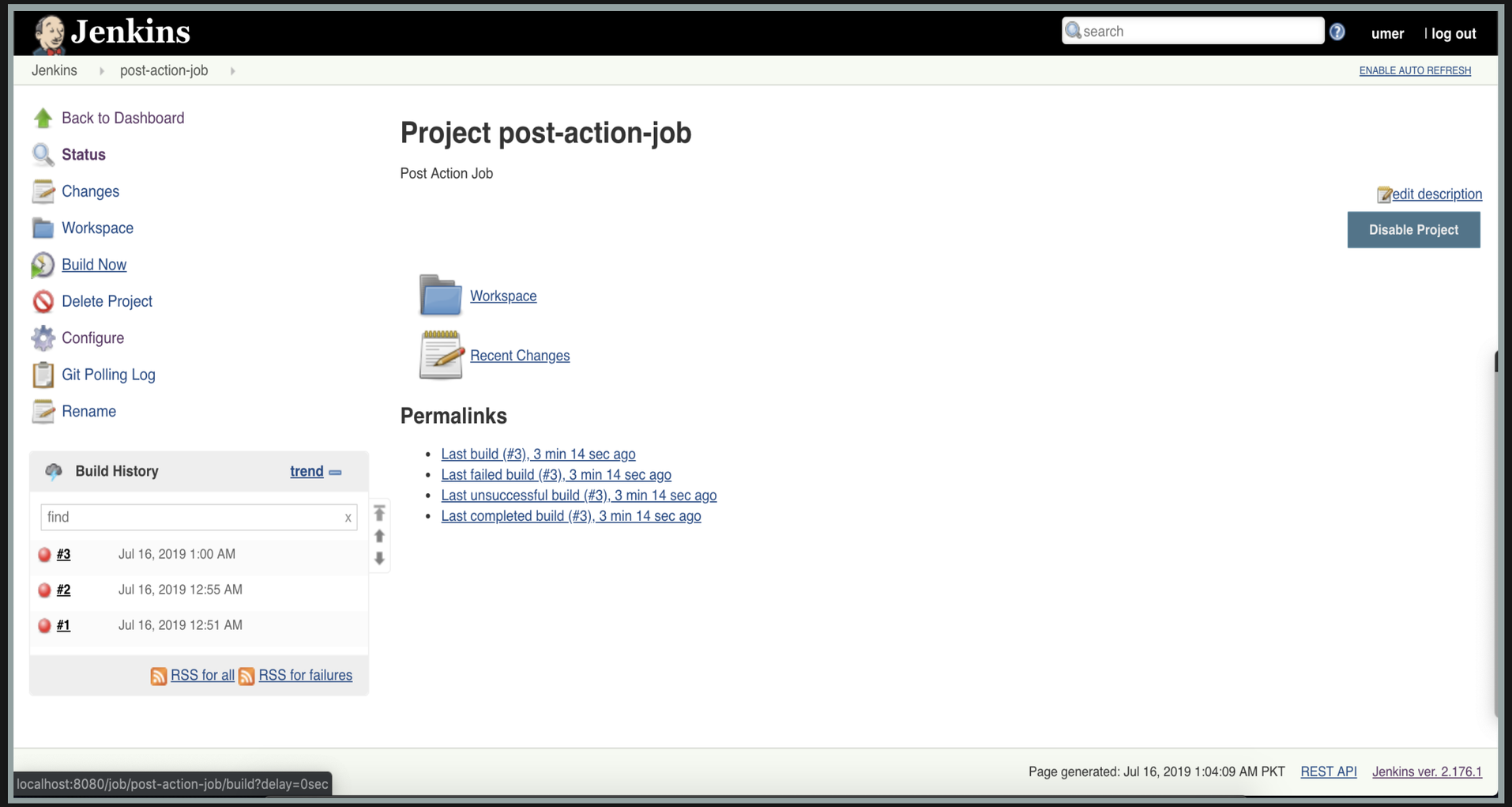
Next you are going to configure email that will be used by Jenkins to send Email Notification, for that click on Manage Jenkins and Configure System



Go to email Notification section and enter details as shown below and click Save



Next go to job and click build now. Next click on the failed build



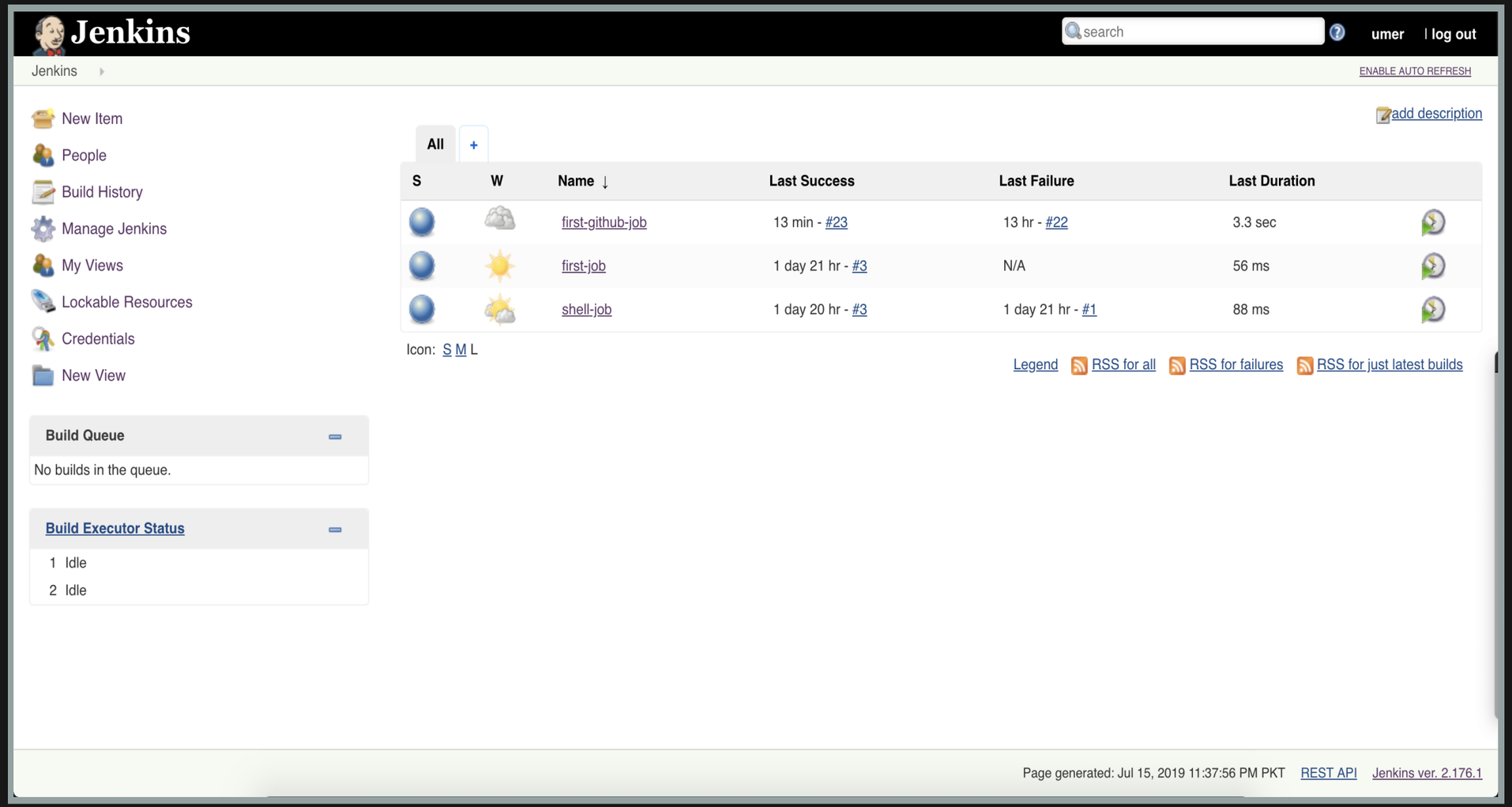
You will notice that you got an error while sending an email because Gmail account needs to be configured to send email from an untrusted source which in this case is Jenkins

JENKINS JOB DSL

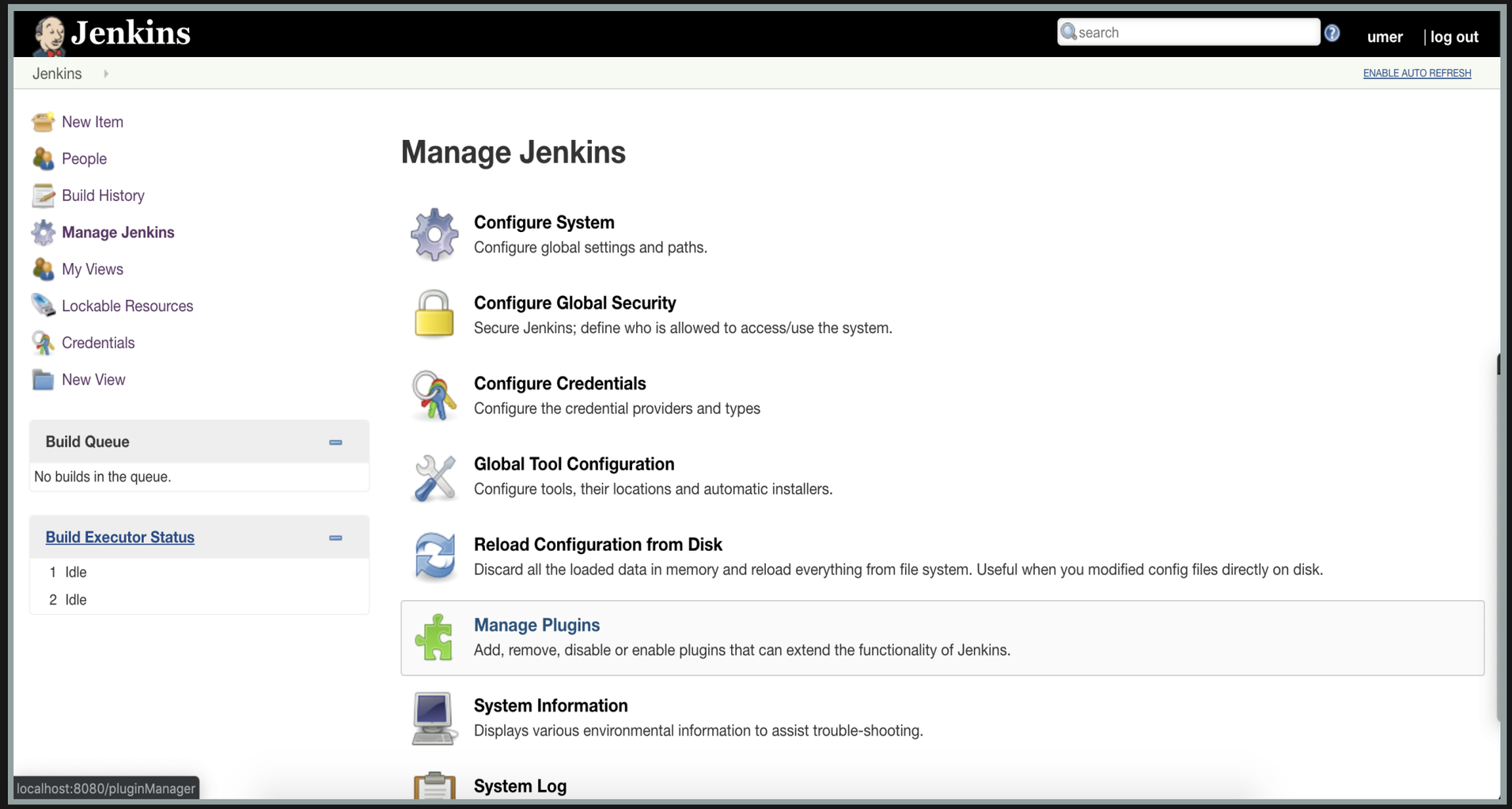
- Plugin that allows you to define jobs in programatic form
- DSL stands for Domain Specific Language
  - Groovy as scripting language
- Easier to manage jobs
  - Use UI if you have less jobs to maintain
  - When number of jobs grow use DSL
- Easier to restore
- Can use with Version Control
- Job Structure
  - Parameters
  - SCM
  - Triggers
  - Build Steps
  - Post-build actions

LAB - JOB DSL

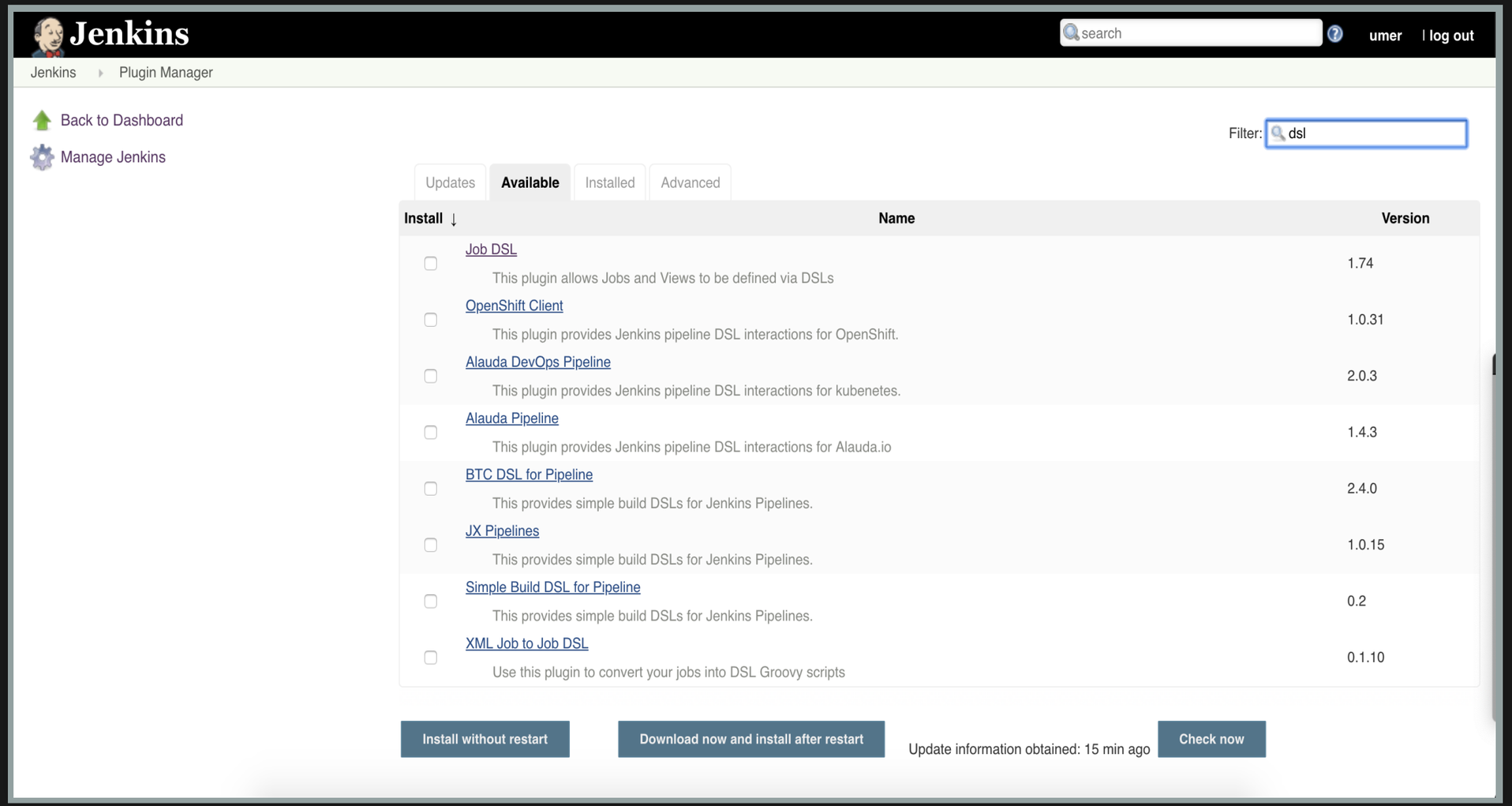
In this Lab, you are going to create your job using a DSL job. First we are going to install Job DSL plugin. Go to the landing page and Click Manage Jenkins



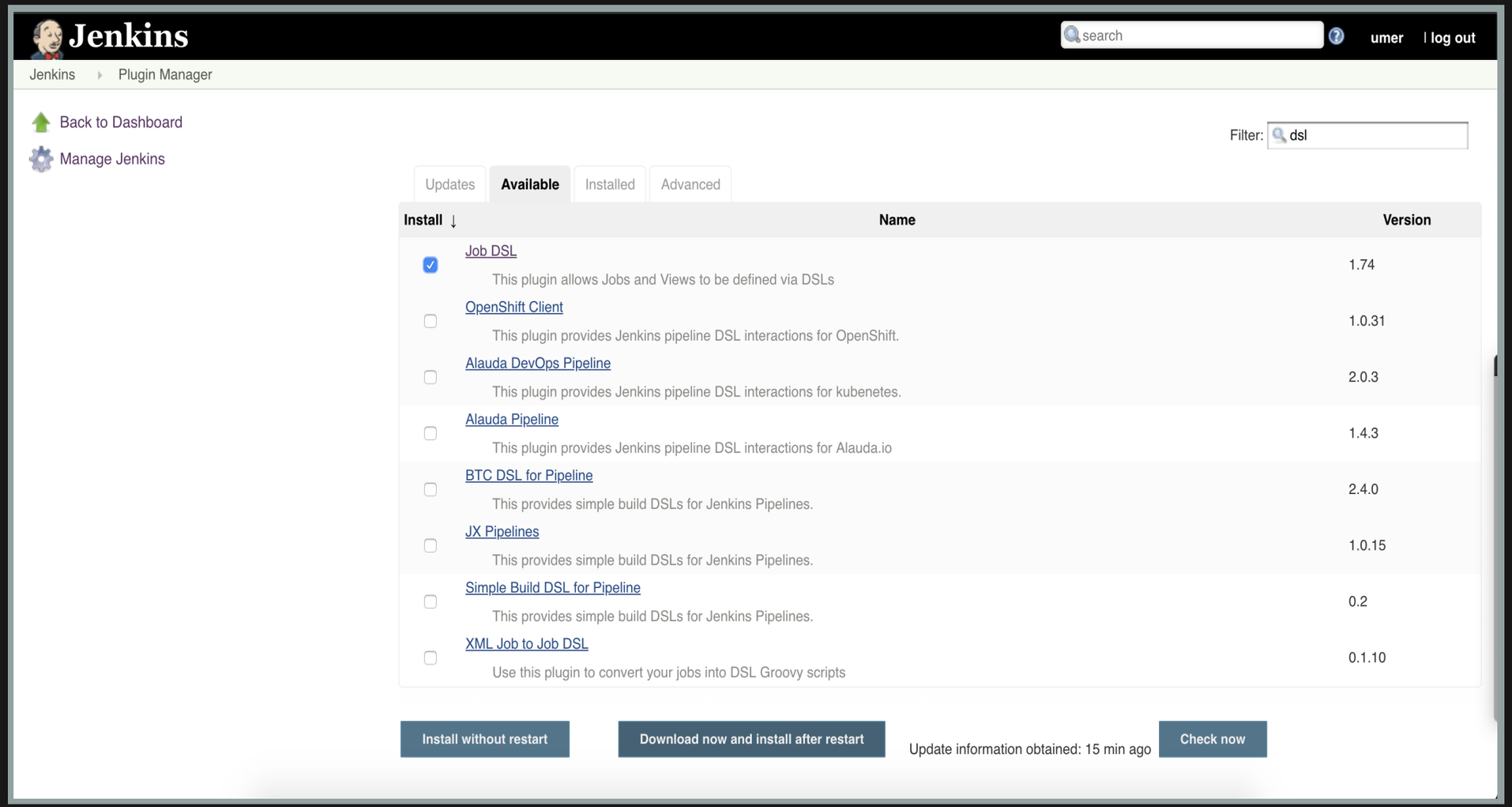
Next, click Manage Plugins



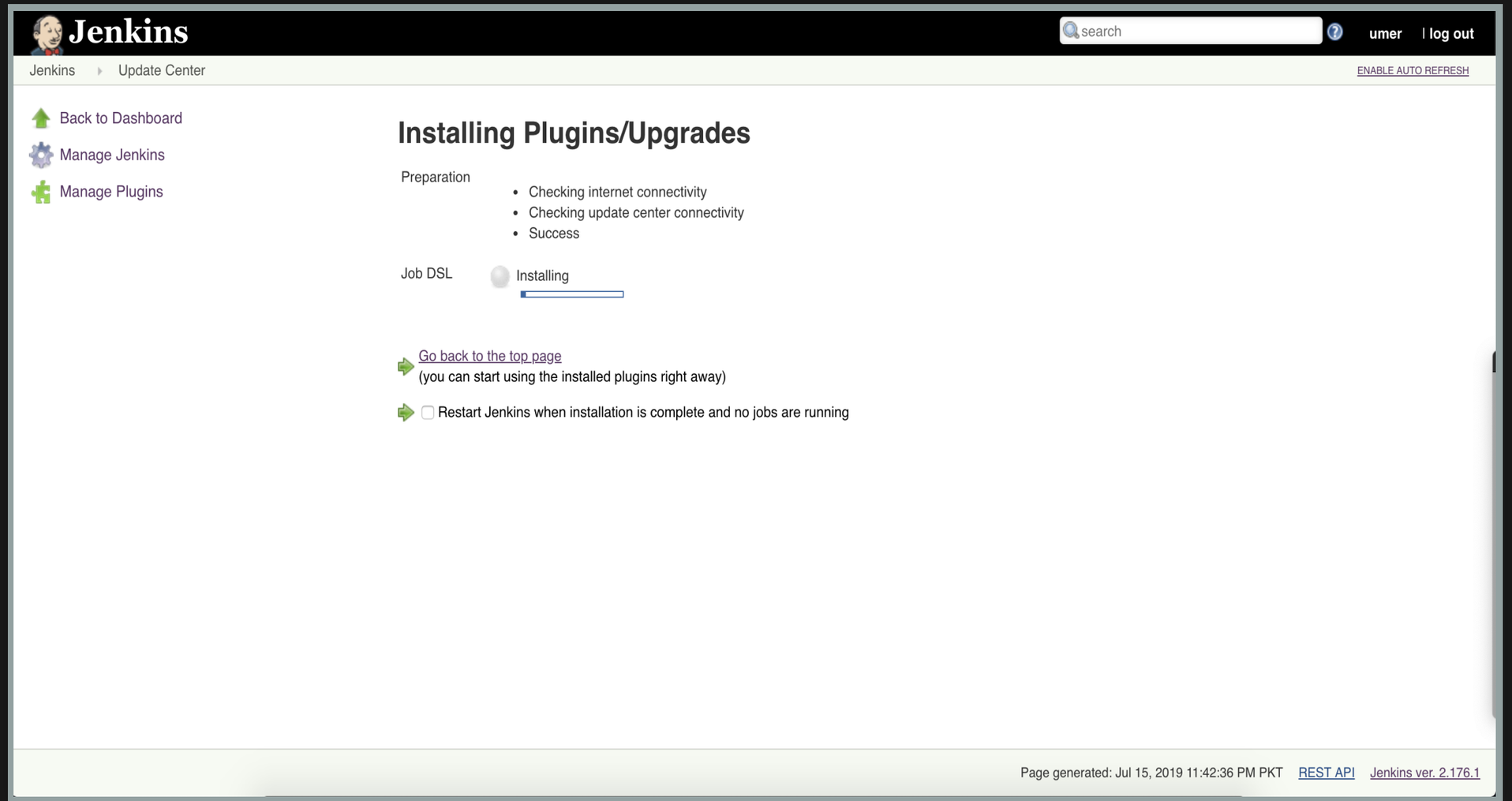
Click Availabe and search for 'dsl' as shown below



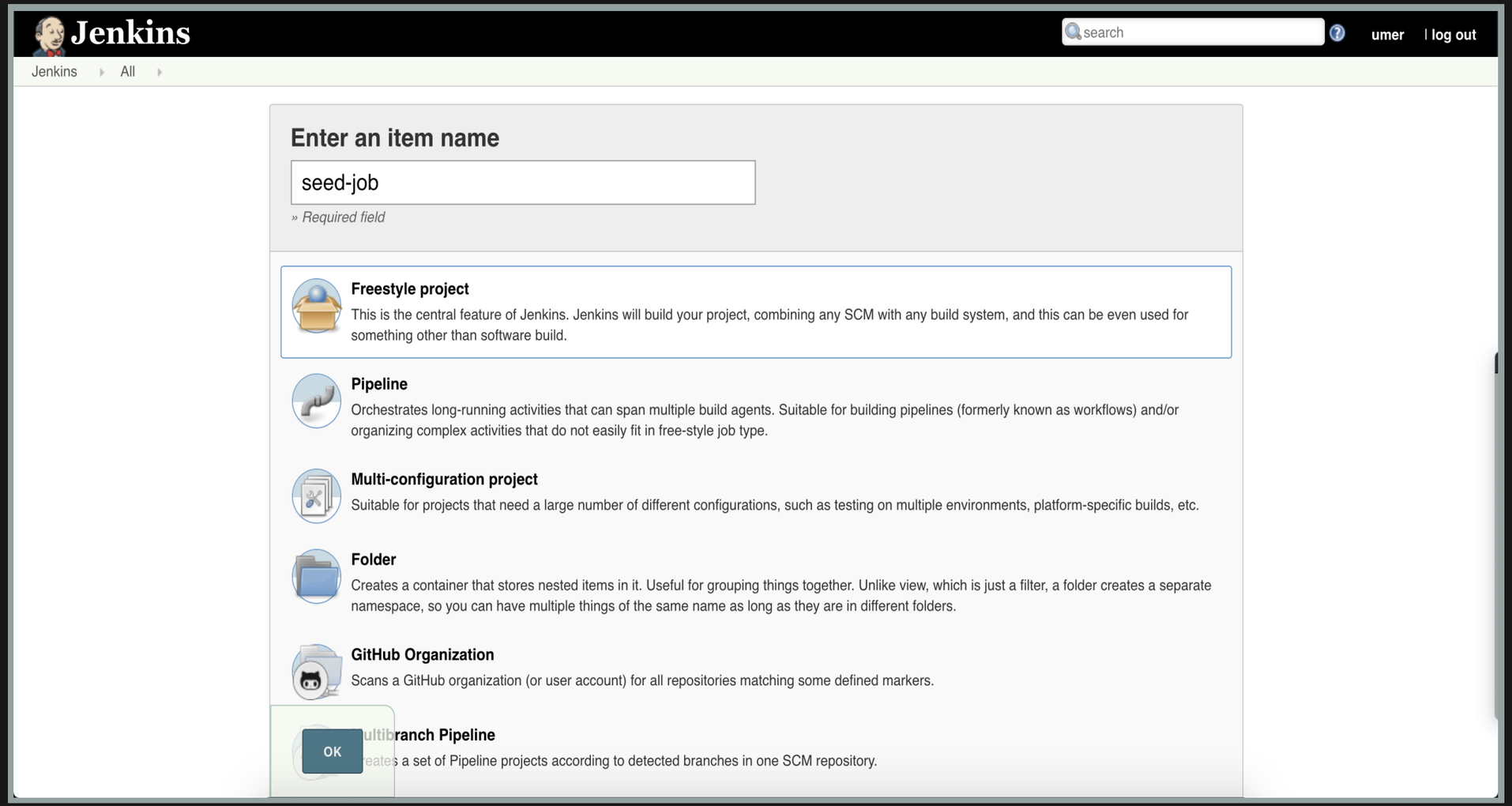
Select 'Job DSL' and click 'Download now and install after restart'



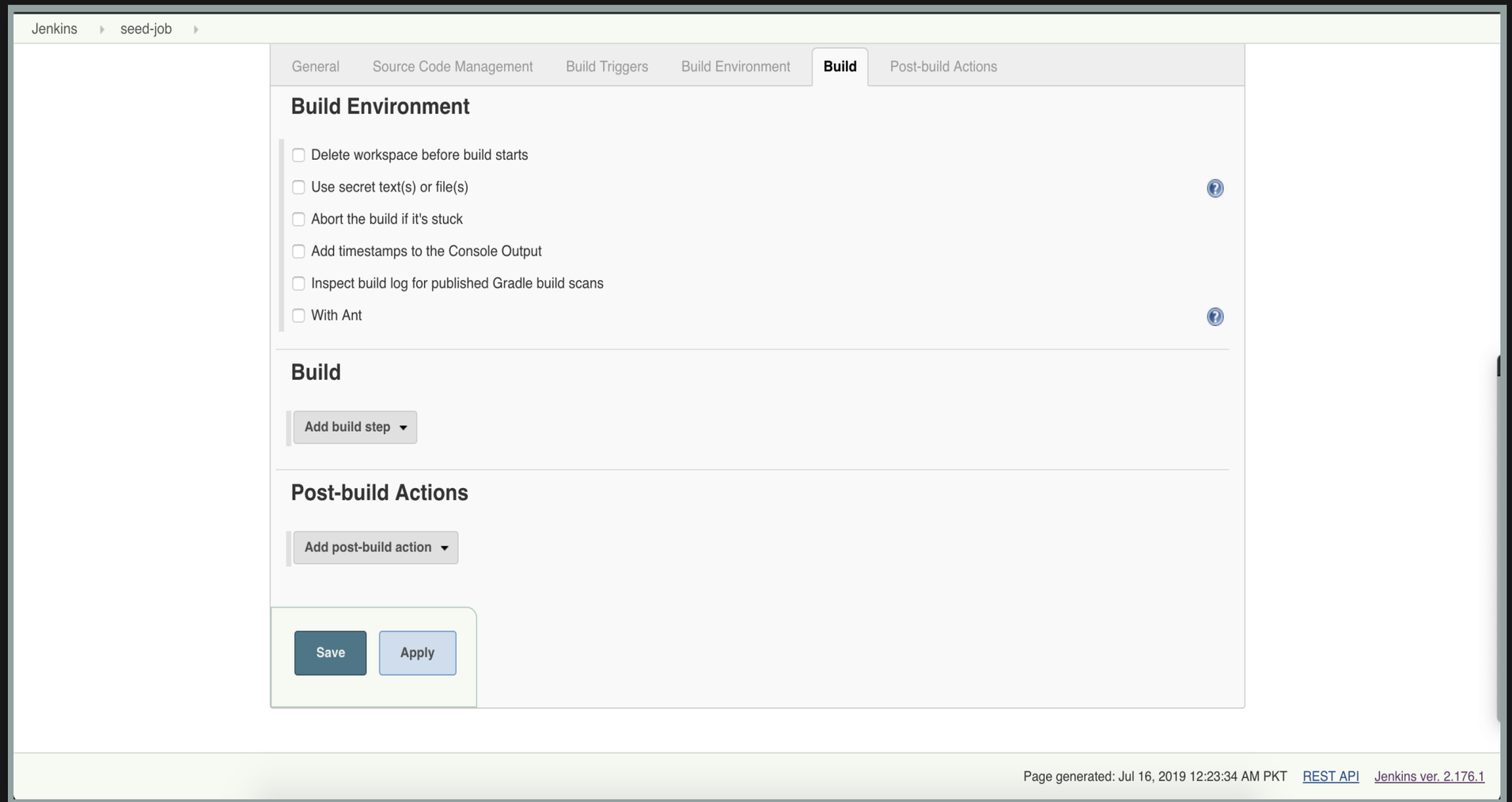
Wait for the plugin to installed and jenkins to be restarted



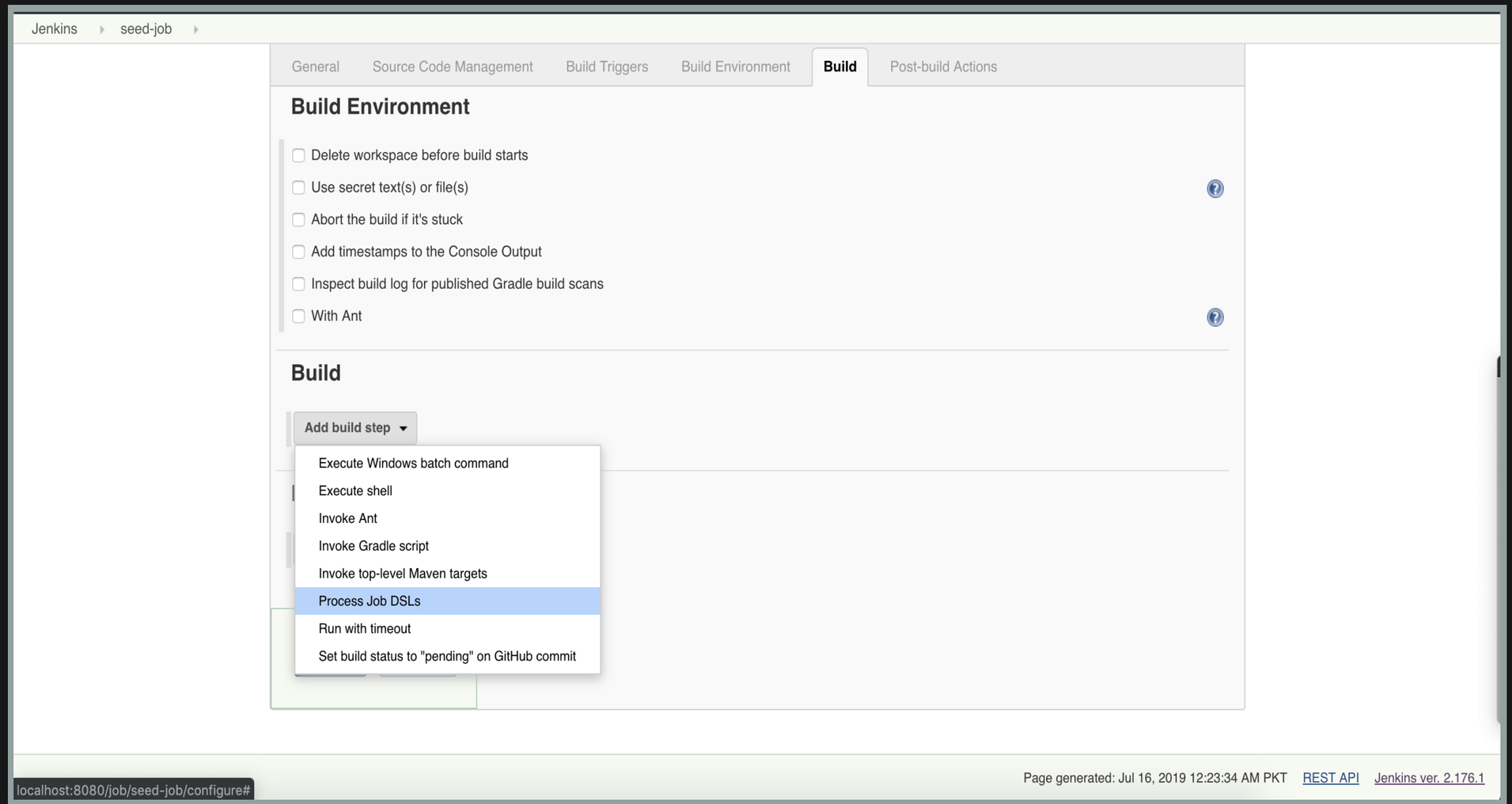
Once restarted and create new item as shown below



Enter job description and Click Build



Select 'Process Job DSLs'

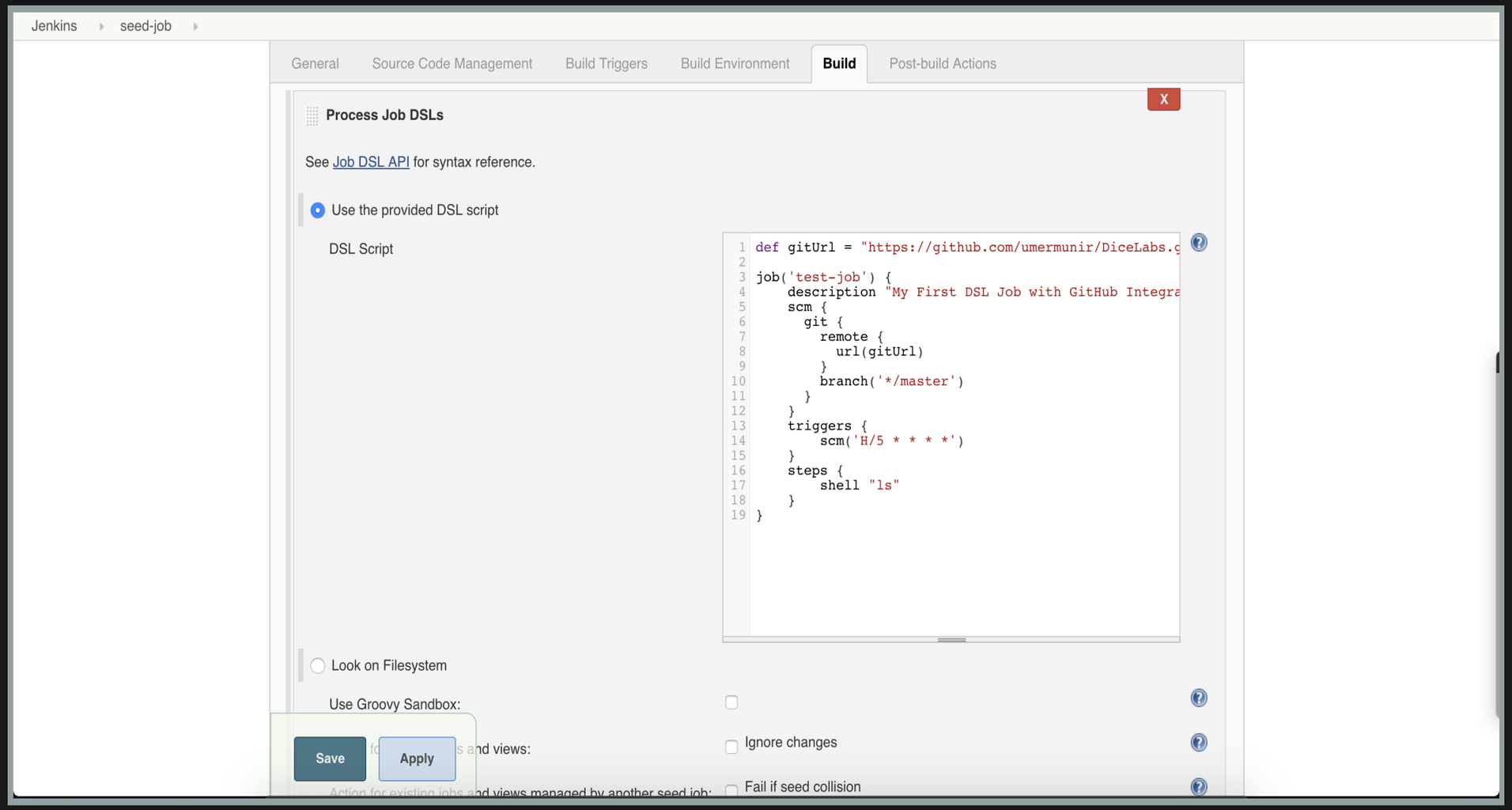


Enter the DSL job

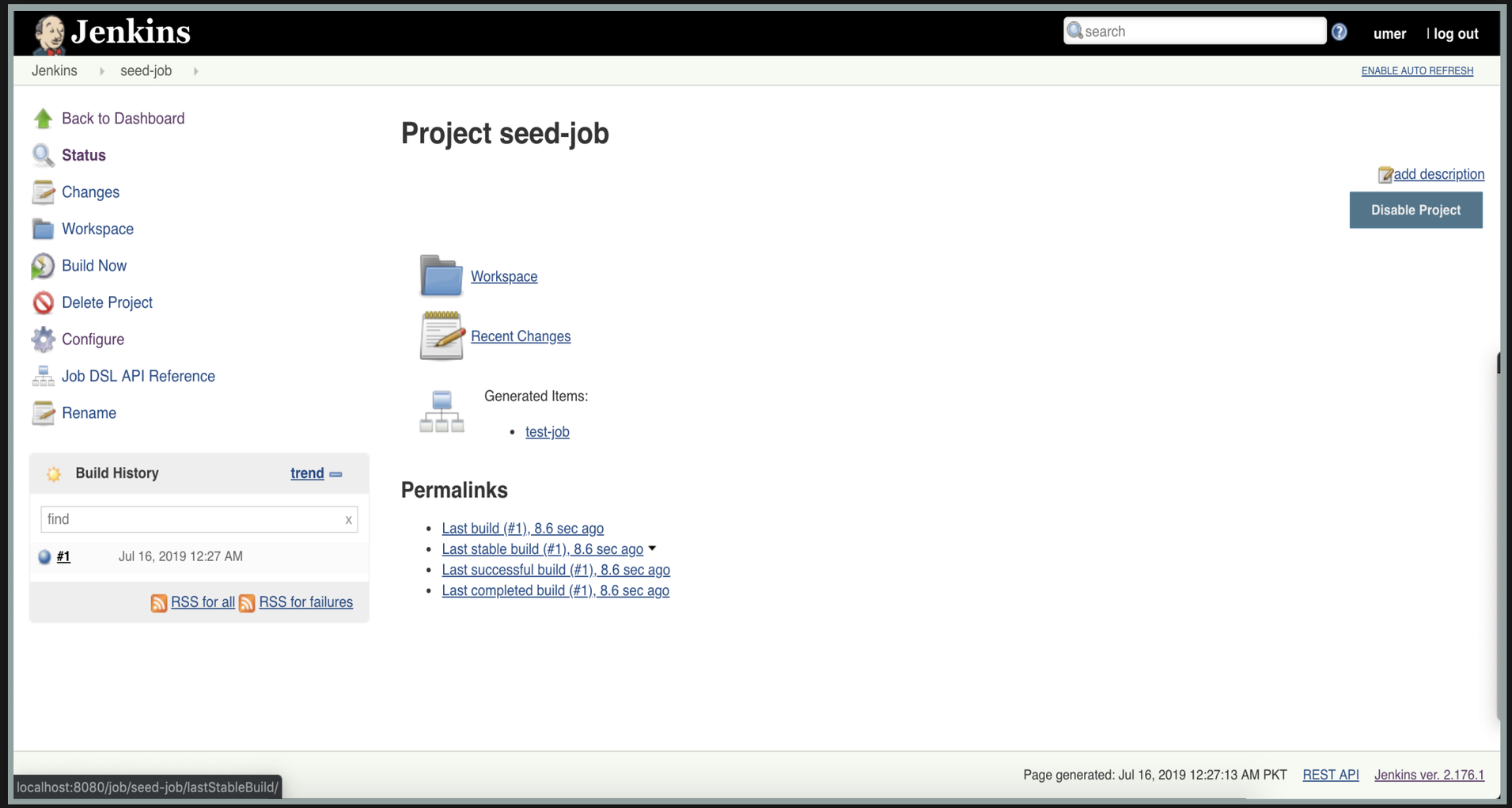
```
def gitUrl = "https://github.com/umermunir/DiceLabs.git"

job('test-job') {
    description "My First DSL Job with GitHub Integration"
    scm {
      git {
        remote {
          url(gitUrl)
        }
        branch('*/master')
      }
    }
    triggers {
        scm('H/5 * * * *')
    }
    steps {
        shell "ls"
    }
}
```

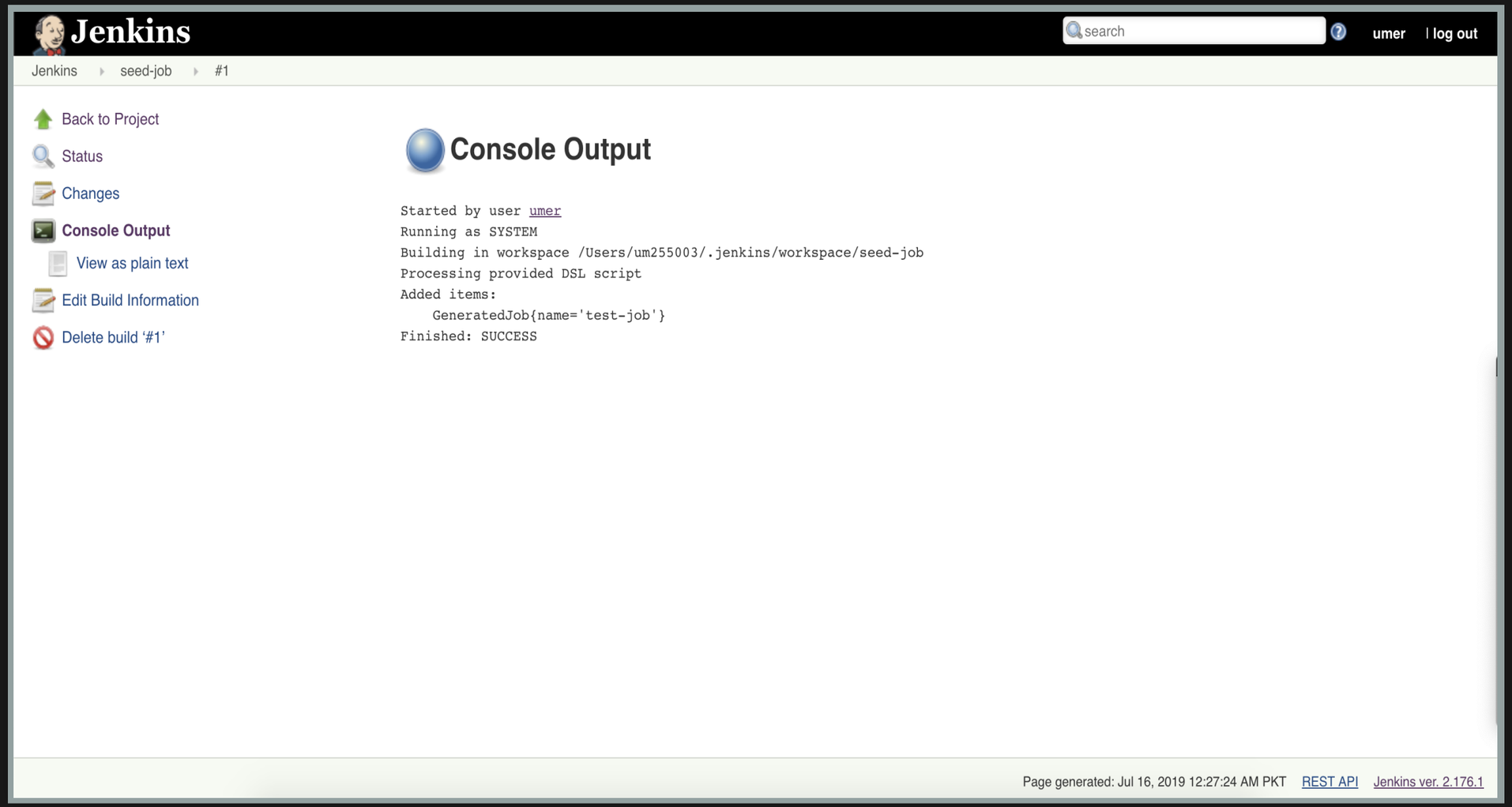
Click Save


Jenkins
seed-job
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Process Job DSLs
See Job DSL API for syntax reference.
Use the provided DSL script
DSL Script
def gitUrl = "https://github.com/umermunir/DiceLabs.g
job('test-job') {
description "My First DSL Job with GitHub Integra
scm {
git {
remote {
url(gitUrl)
}
branch('*/master')
}
}
triggers {
scm('H/5 * * * *')
}
steps {
shell "ls"
}
}
Look on Filesystem
Use Groovy Sandbox:
Ignore changes
Fail if seed collision
Save
Apply

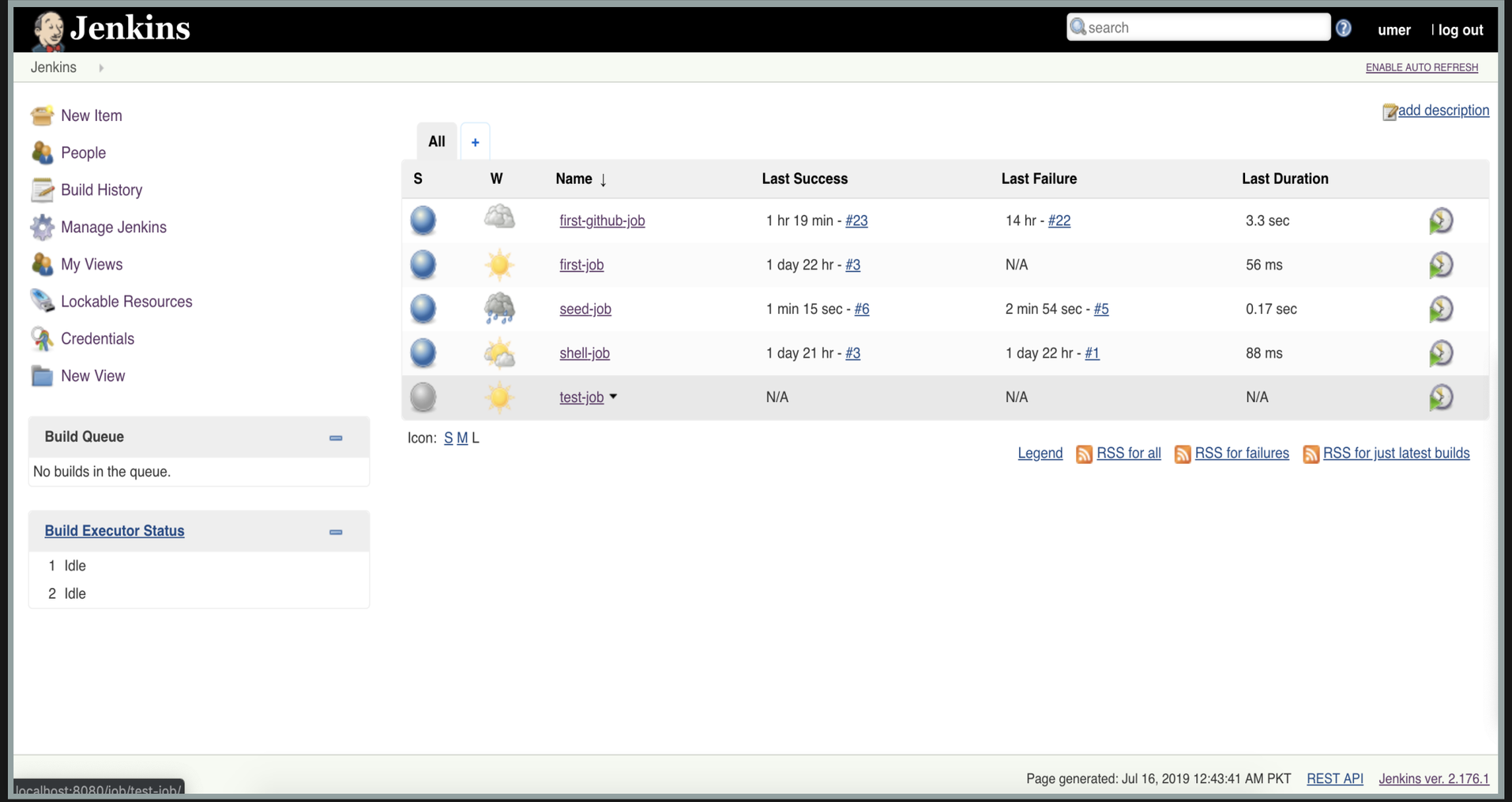
Click Build Now to run the job



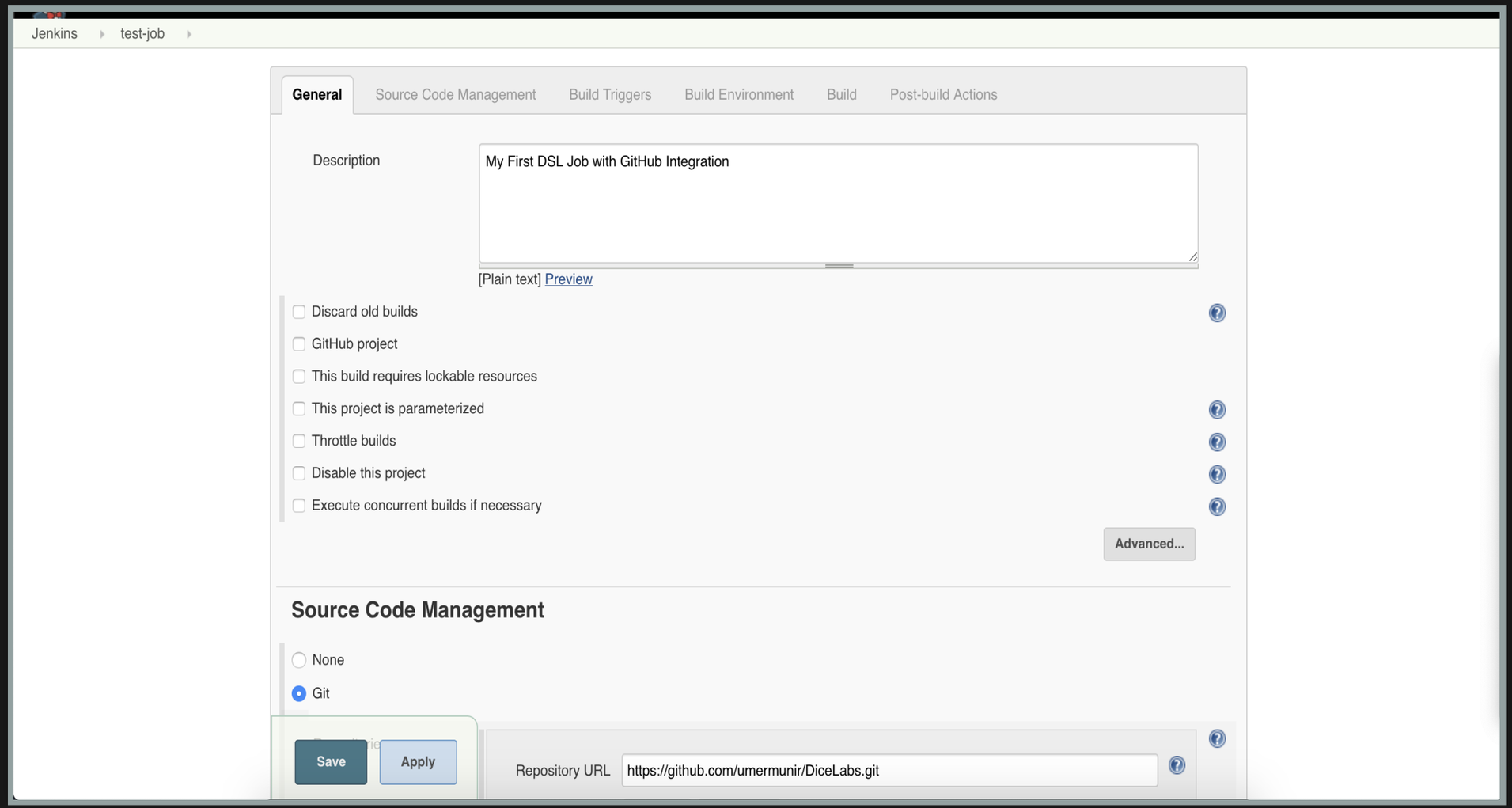
Check Console Output to verify if the new job has been created



Check if the job has been created by going to the landing page



Click on newly created job to check configuration



Jenkins ▸ test-job ▸

General | Source Code Management | Build Triggers | **Build Environment** | Build | Post-build Actions

- [ ] Use secret text(s) or file(s)
- [ ] Abort the build if it's stuck
- [ ] Add timestamps to the Console Output
- [ ] Inspect build log for published Gradle build scans
- [ ] With Ant

Build

Execute shell

Command

```
ls
```

See the list of available environment variables

Advanced...

Add build step

Post-build Actions

Save | Apply

JENKINS PIPELINE

- Set of plugins which support implementing and integrating continuous delivery pipelines into Jenkins
- Is a job type to perform sequence of steps
  - Build
  - Test
  - Deploy
- A Pipeline can be created
  - Through UI
  - In SCM
- Types
  - Scripted Pipeline
  - Declarative Pipeline

SCRIPTED PIPELINE

Follows a more imperative programming model built with Groovy.

```
node {
    stage ('Build') {
        //...
    }
    stage ('Test') {
        //...
    }
    stage ('Deploy') {
        //...
    }
}
```

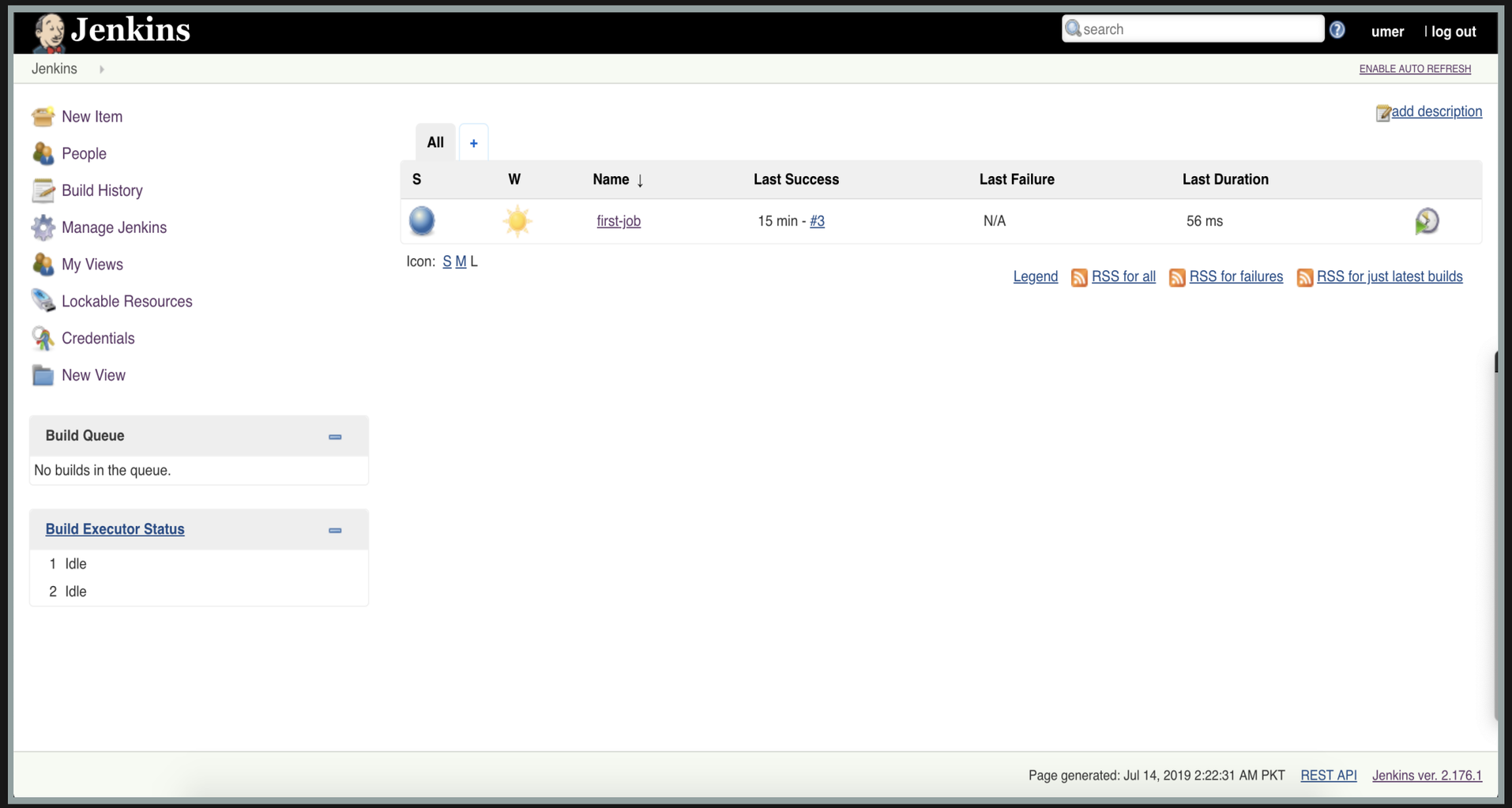
DECLARATIVE PIPELINE

Presents a more simplified and opinionated syntax on top of the Pipeline sub-systems

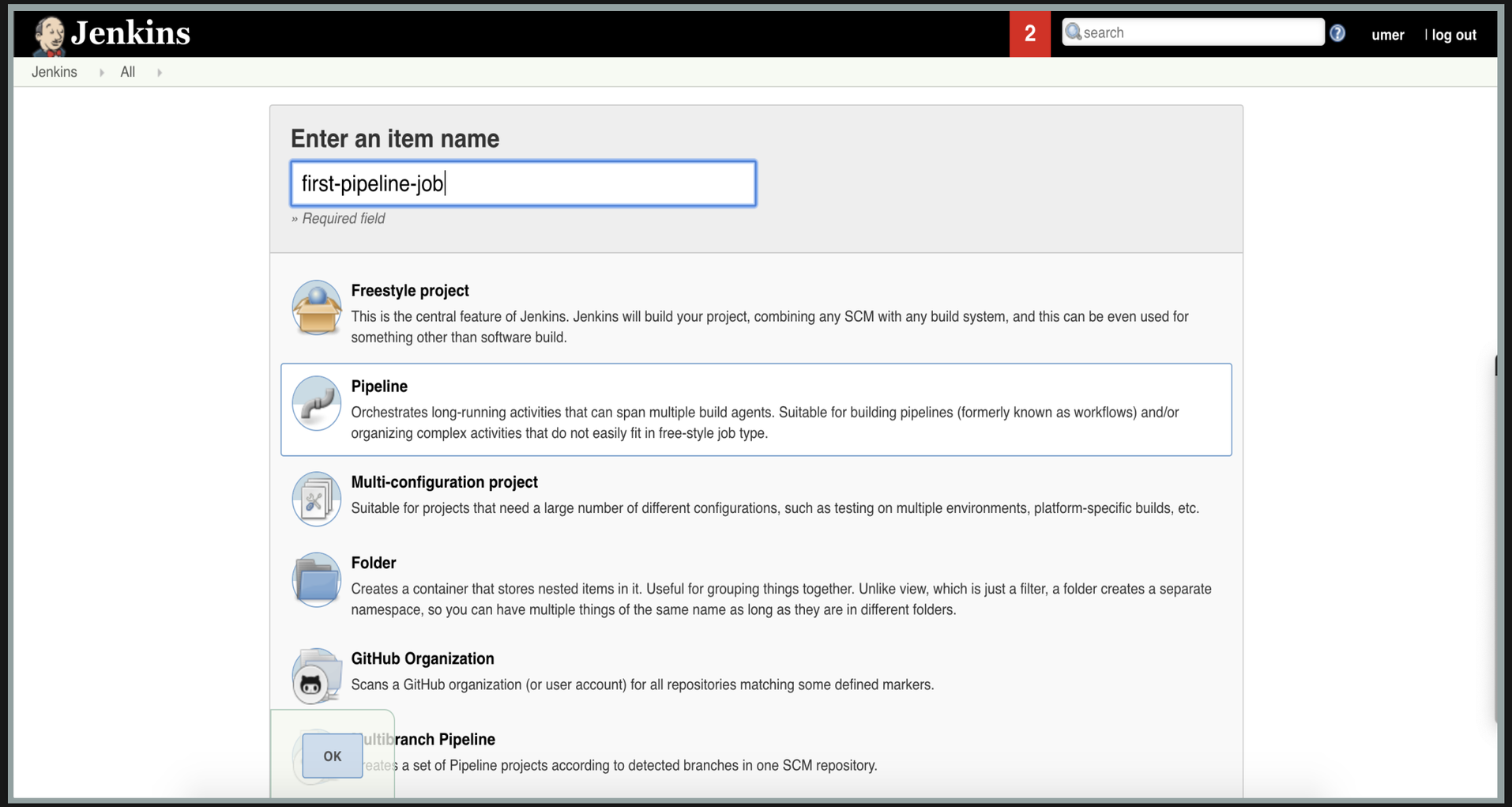
```
pipeline {
    agent any
    stages {
    stage('Build') {
        steps {
            //..
        }
      }
    stage ('Test') {
      steps {
          //..
        }
      }
  }
}
```

LAB - FIRST PIPELINE

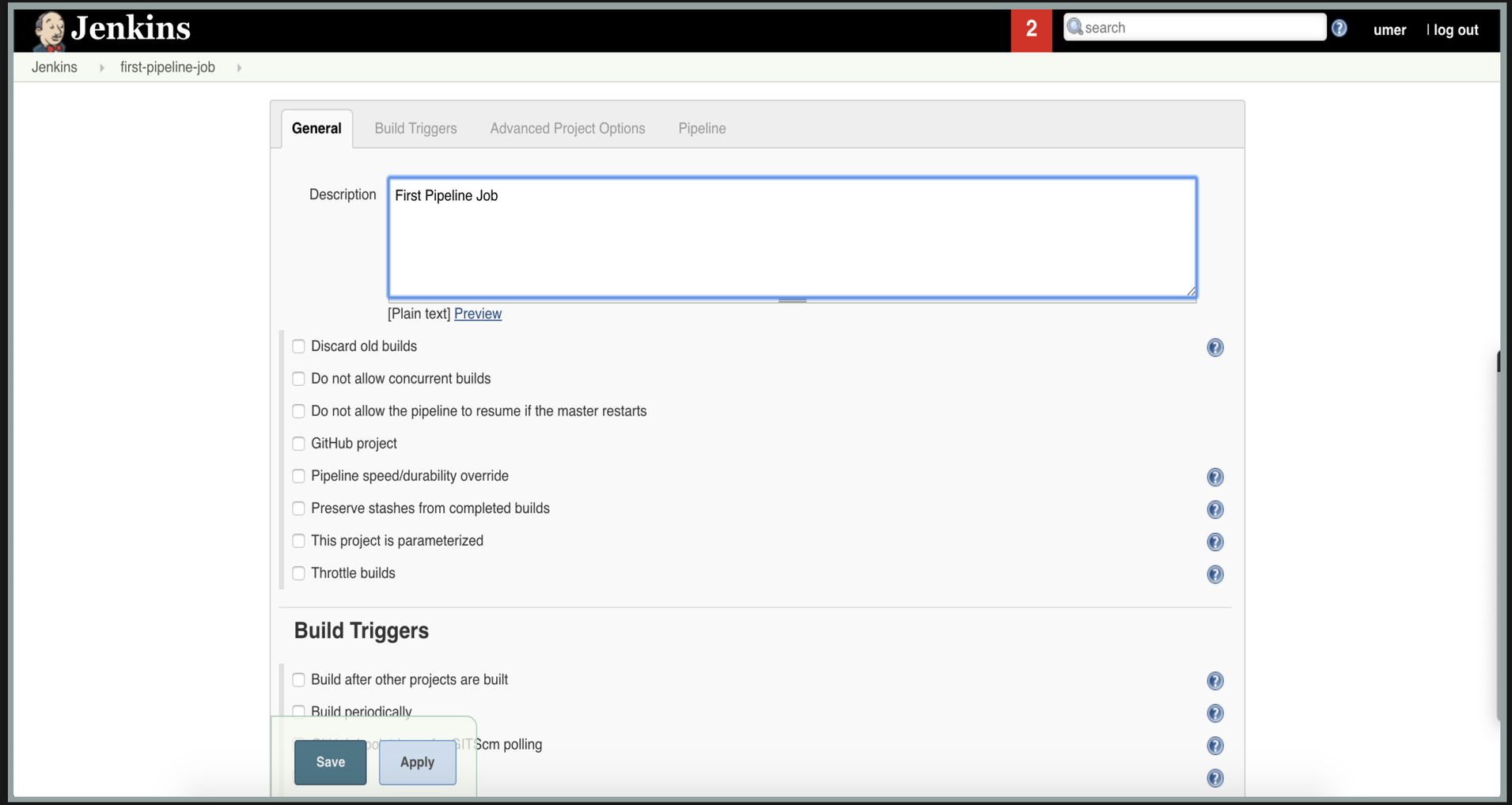
In this Lab, you are going to create a Jenkins Pipeline Job without any SCM integration. Go to Jenkins landing page by clicking on Jenkins on the left top corner and click New Item.



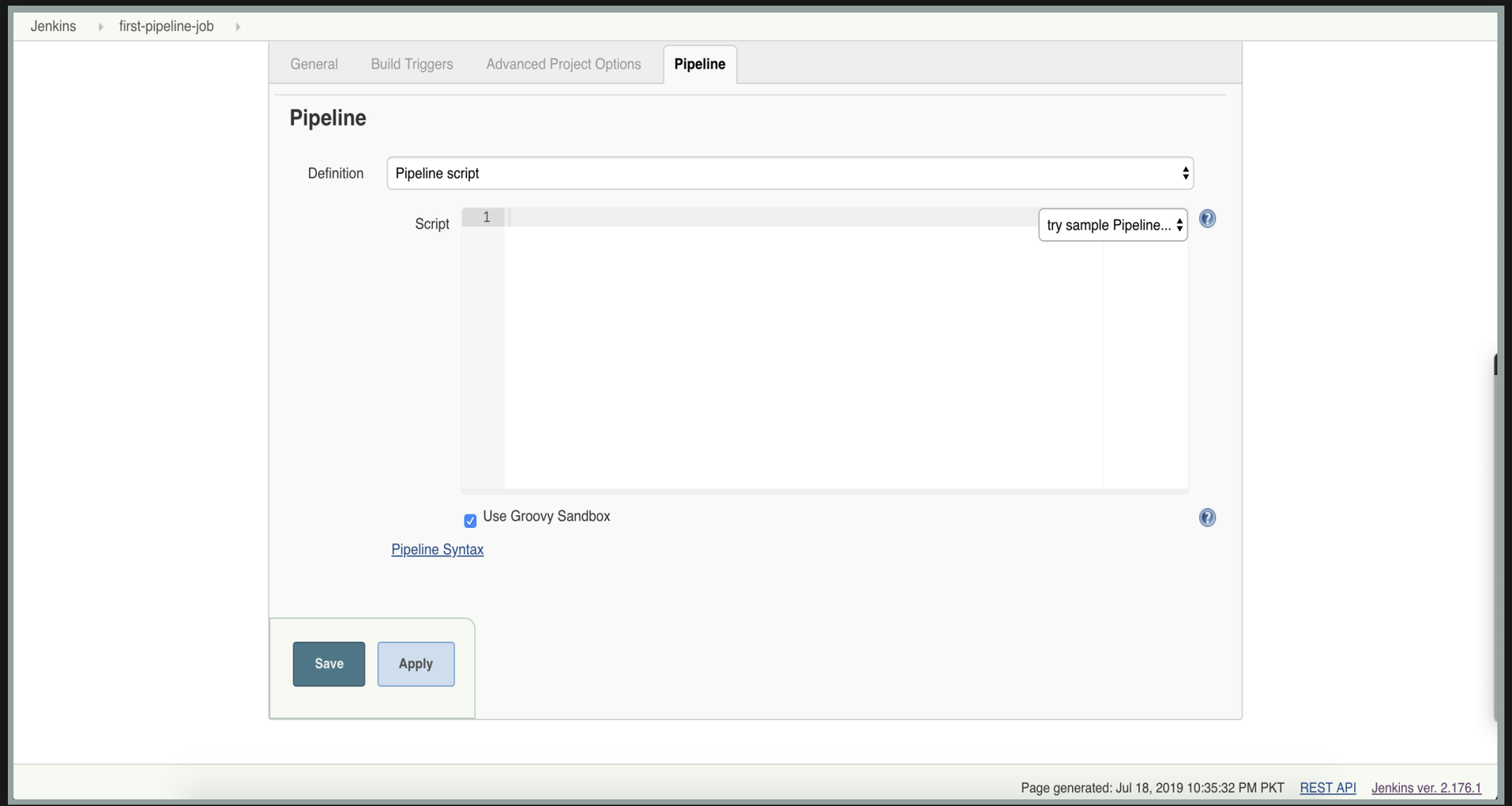
Enter the name of the job and select Pipeline and Click OK



Enter the description and Click Pipeline.



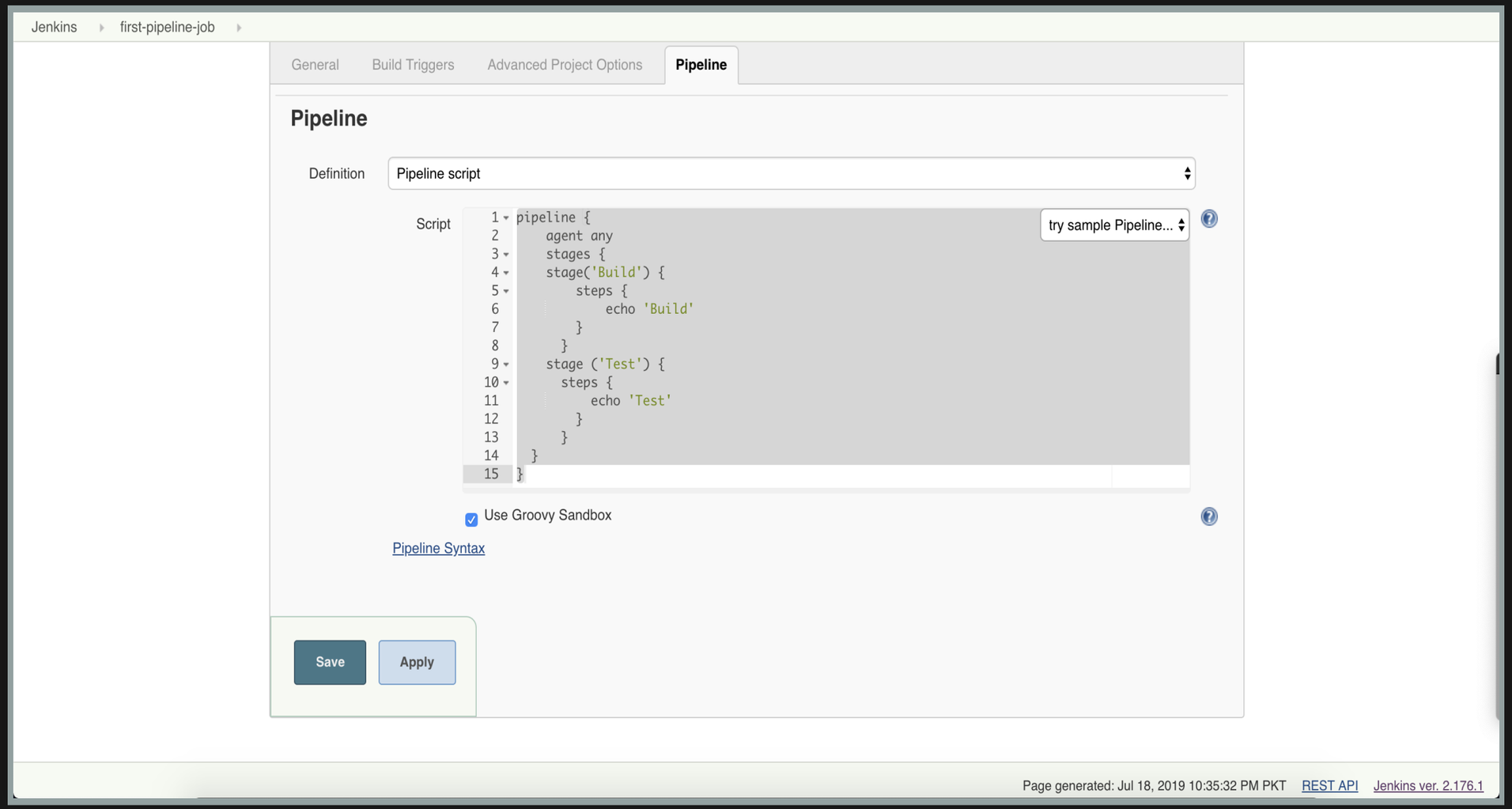
Select Pipeline script in the Definition drop box.



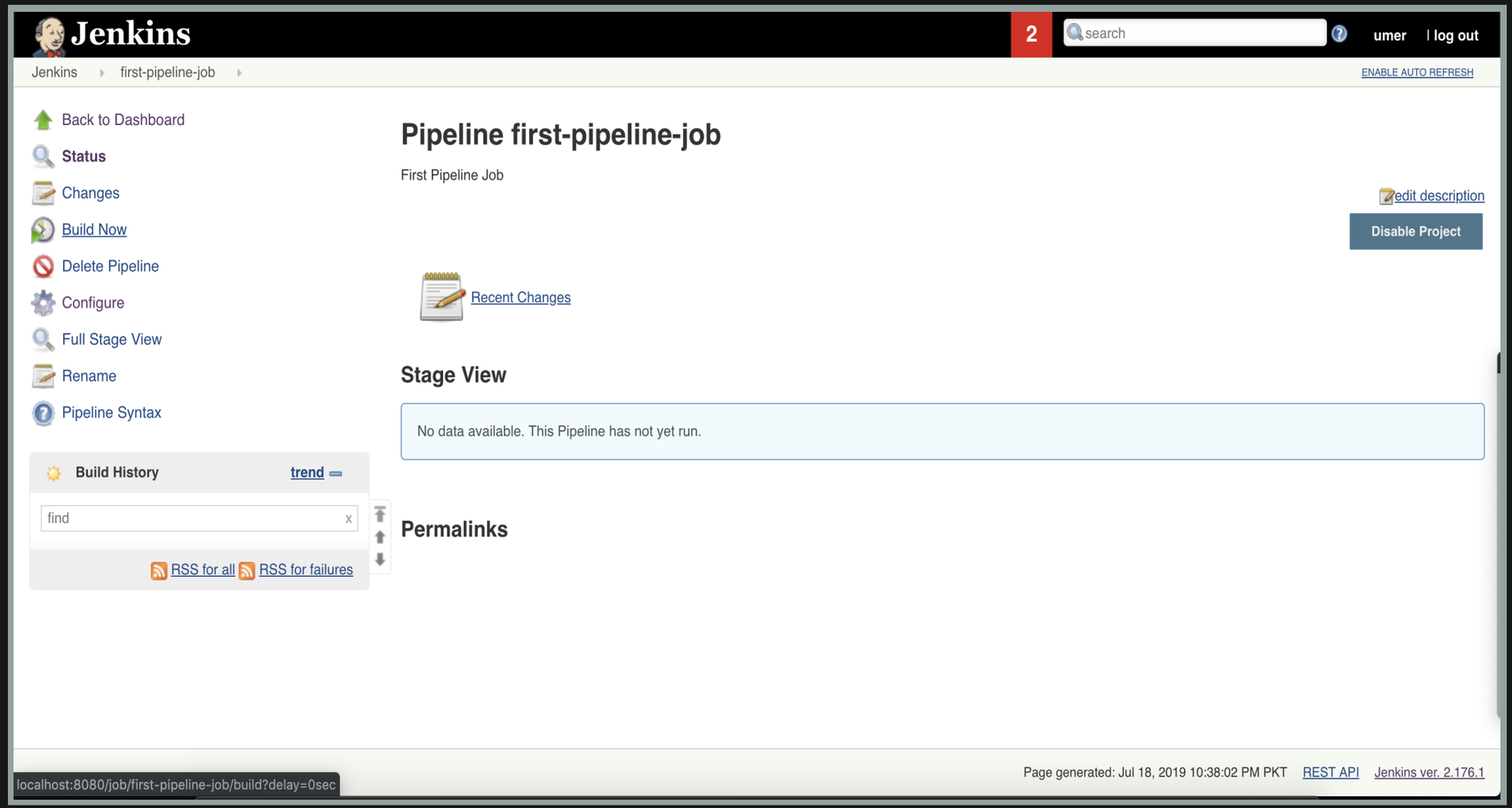
Enter the below script in the Script section

```
pipeline {
    agent any
    stages {
    stage('Build') {
        steps {
            echo 'Build'
        }
    }
    stage ('Test') {
      steps {
          echo 'Test'
        }
      }
  }
}
```

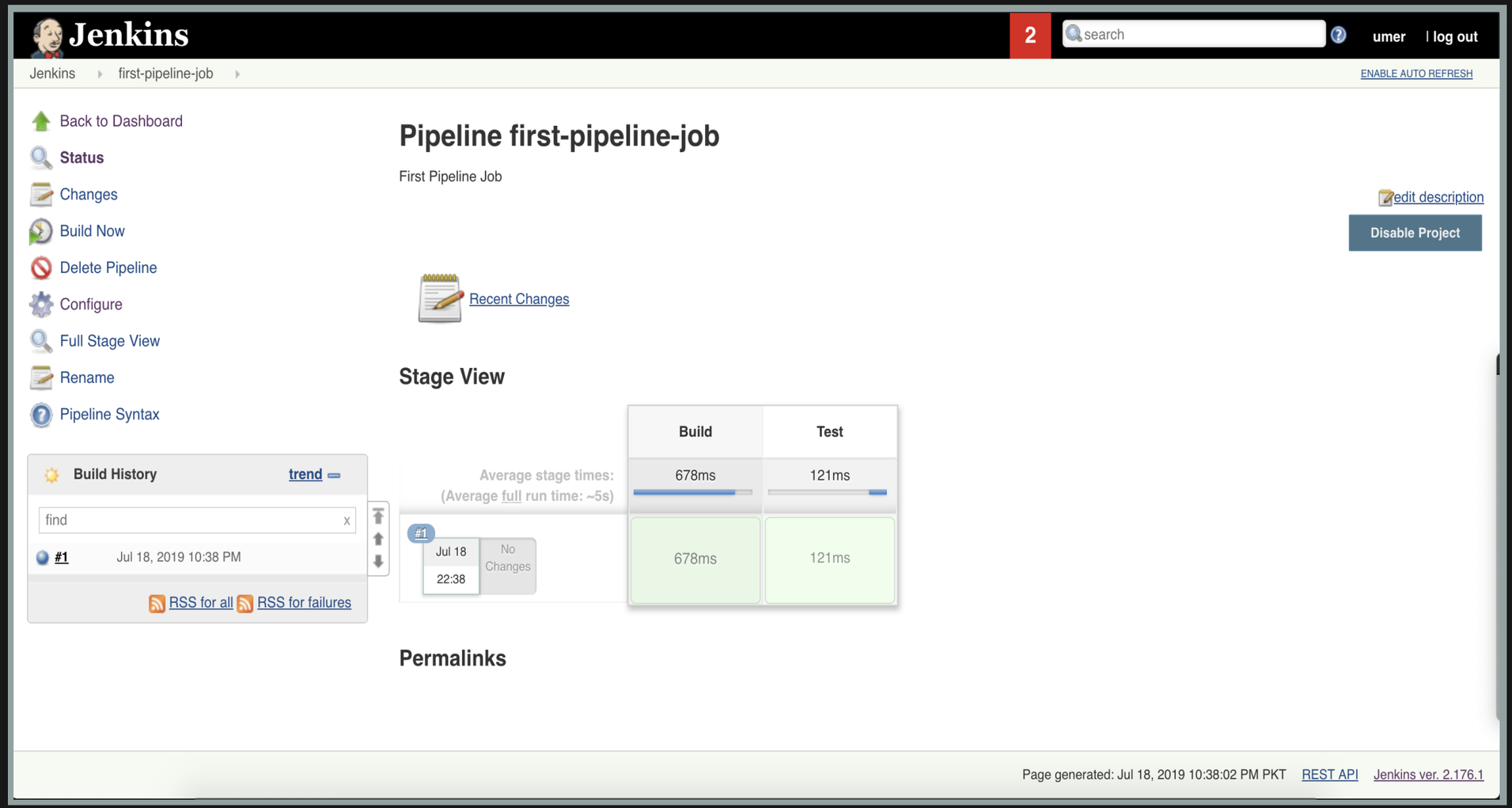
Click Save


Jenkins
first-pipeline-job
General
Build Triggers
Advanced Project Options
Pipeline
Pipeline
Definition
Pipeline script
Script
try sample Pipeline...
pipeline {
agent any
stages {
stage('Build') {
steps {
echo 'Build'
}
}
stage ('Test') {
steps {
echo 'Test'
}
}
}
}
Use Groovy Sandbox
Pipeline Syntax
Save
Apply
Page generated: Jul 18, 2019 10:35:32 PM PKT
REST API
Jenkins ver. 2.176.1

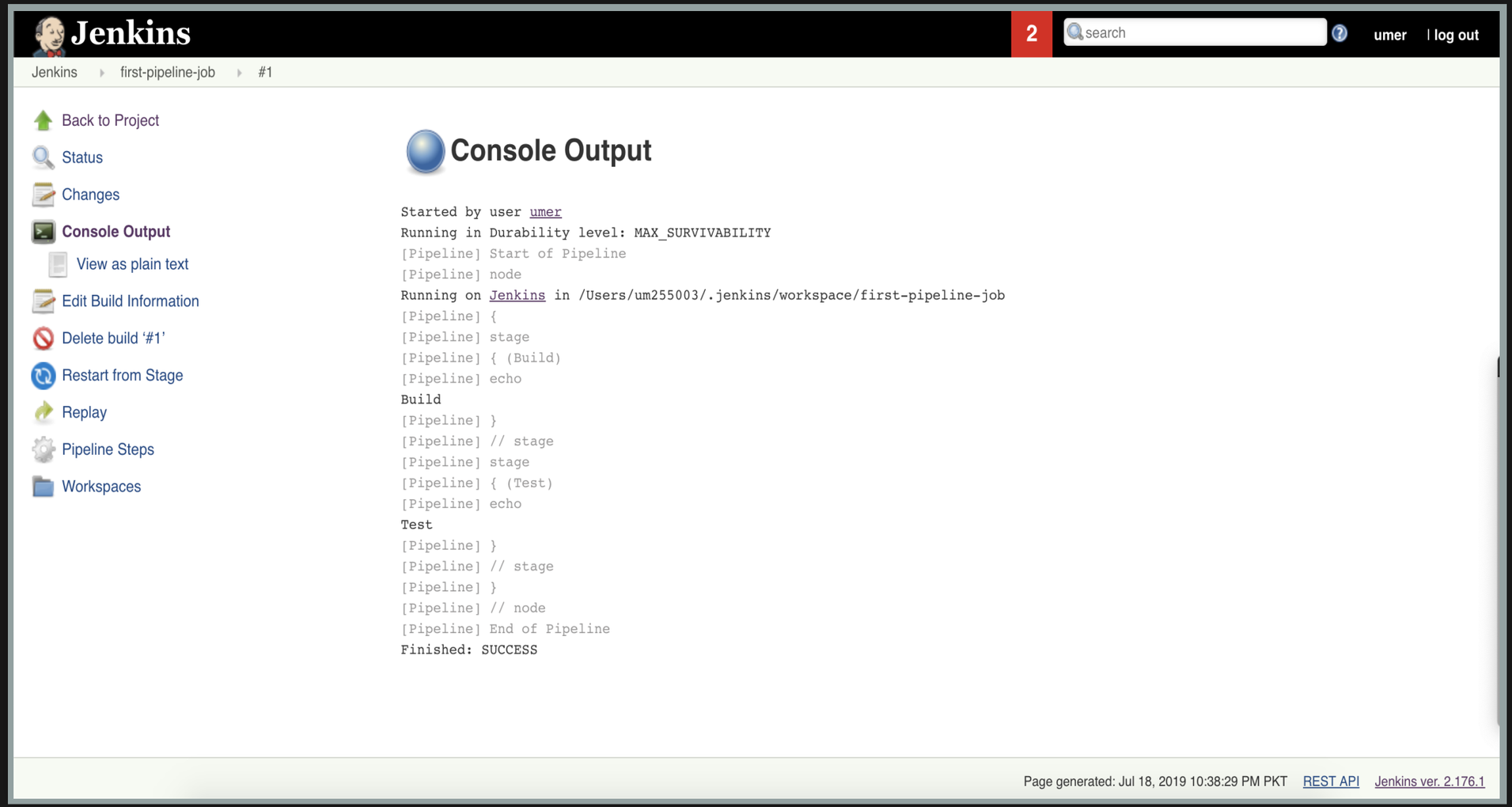
Click Build to run the Pipeline Job



You can see the Build and Test stages in the screenshot below



Check Console Output, you should see the logs as shown below



JENKINSFILE

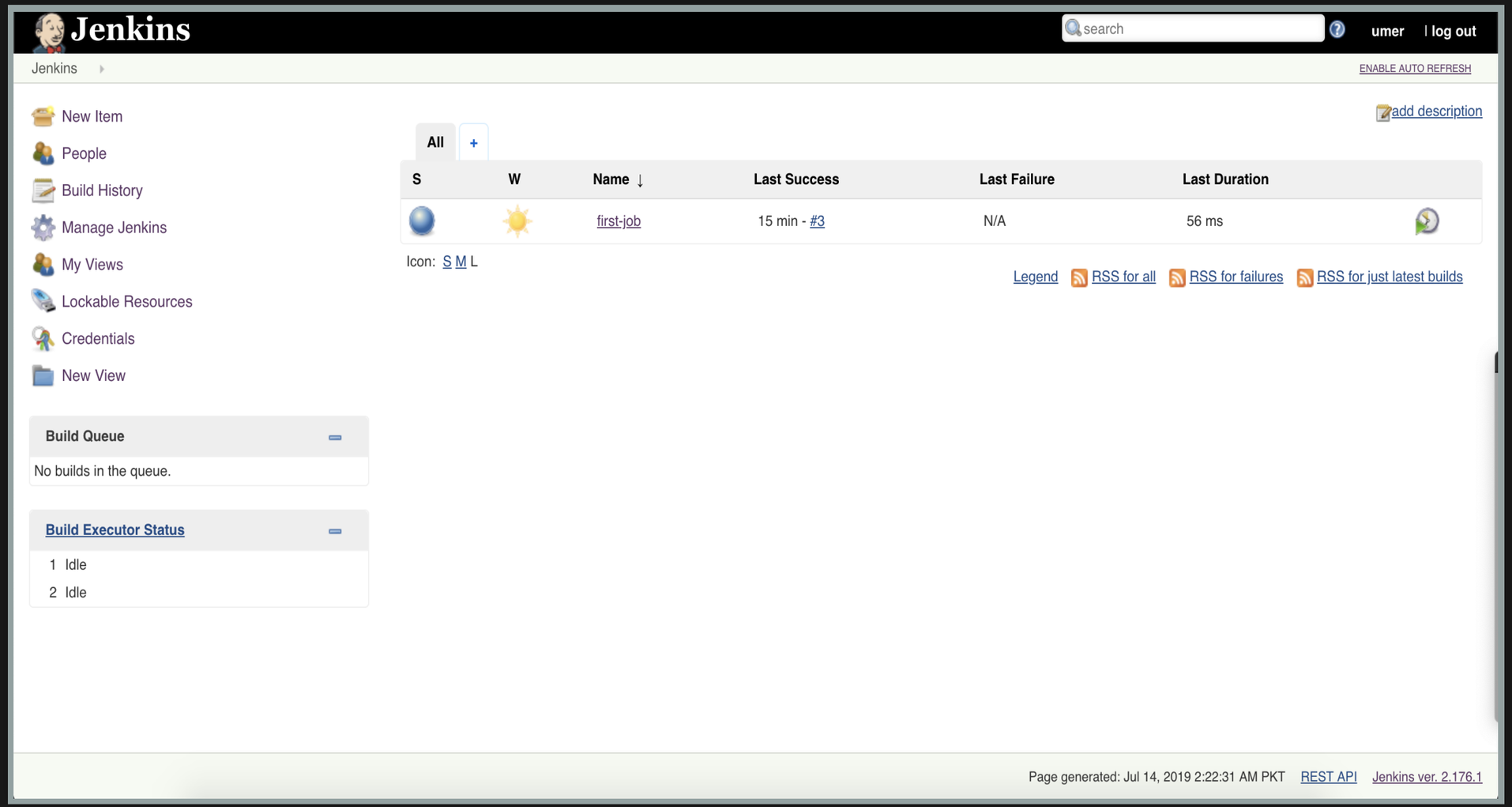
- File that contains the definition of Jenkins Pipeline and checked into source control
- Pipeline as code
- Benefits
  - Version control
  - Code review on the Pipeline
  - Audit trail for the Pipeline
  - Single source of truth for the Pipeline

LAB - JENKINSFILE

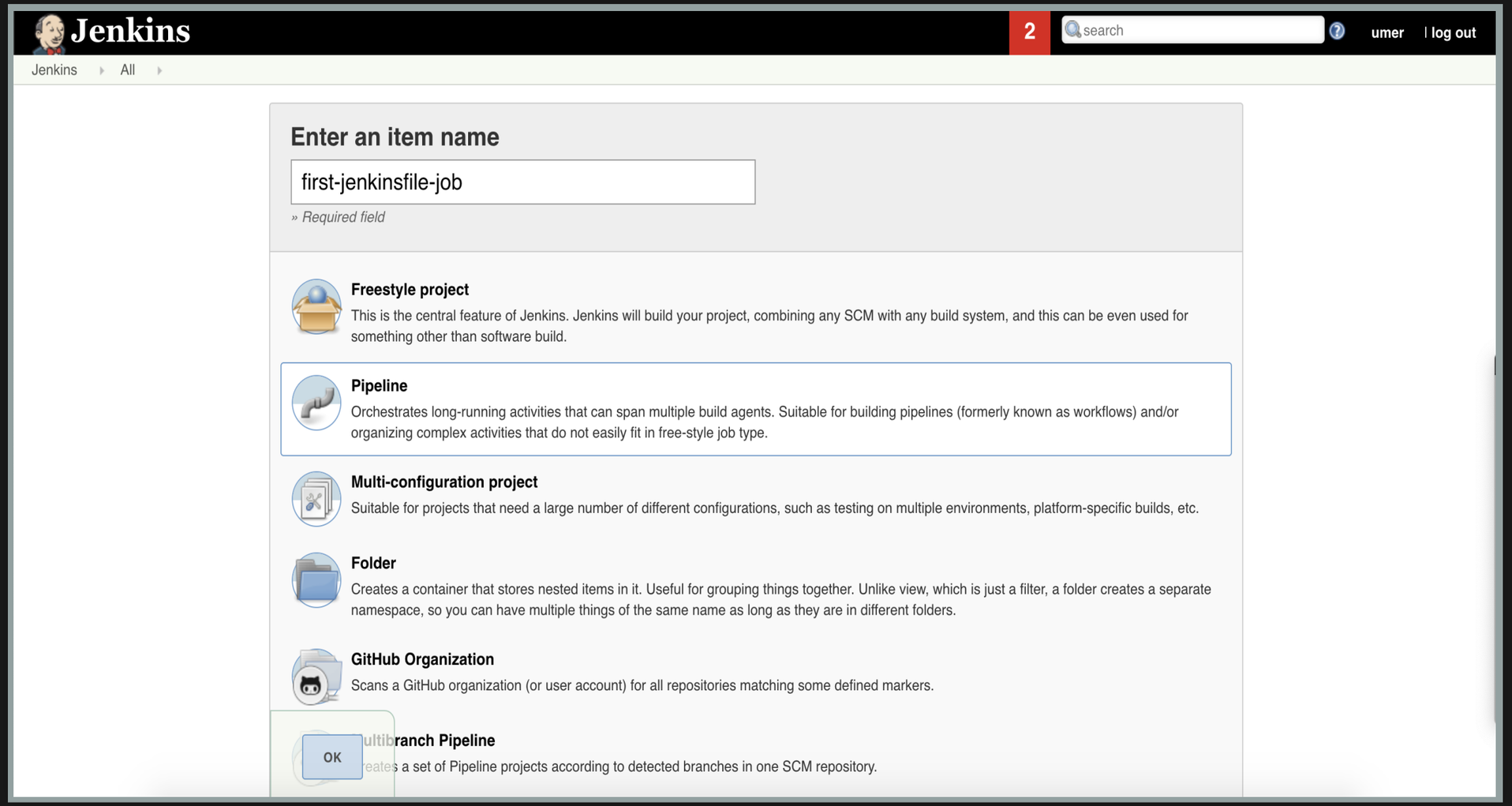
In this Lab, you are going to create a Jenkinsfile, push to the GitHub repository and then create a Jenkins job to run the steps defined in the Jenkinsfile. First create a Jenkinsfile at the root of repository that was forked from DiceLabs with the following script. Name of the file should be Jenkinsfile. Push this file to the remote repository.

```
pipeline {
    agent any
    stages {
    stage('Build') {
        steps {
            echo 'Build'
        }
      }
    stage ('Test') {
      steps {
          echo 'Test'
        }
      }
  }
}
```

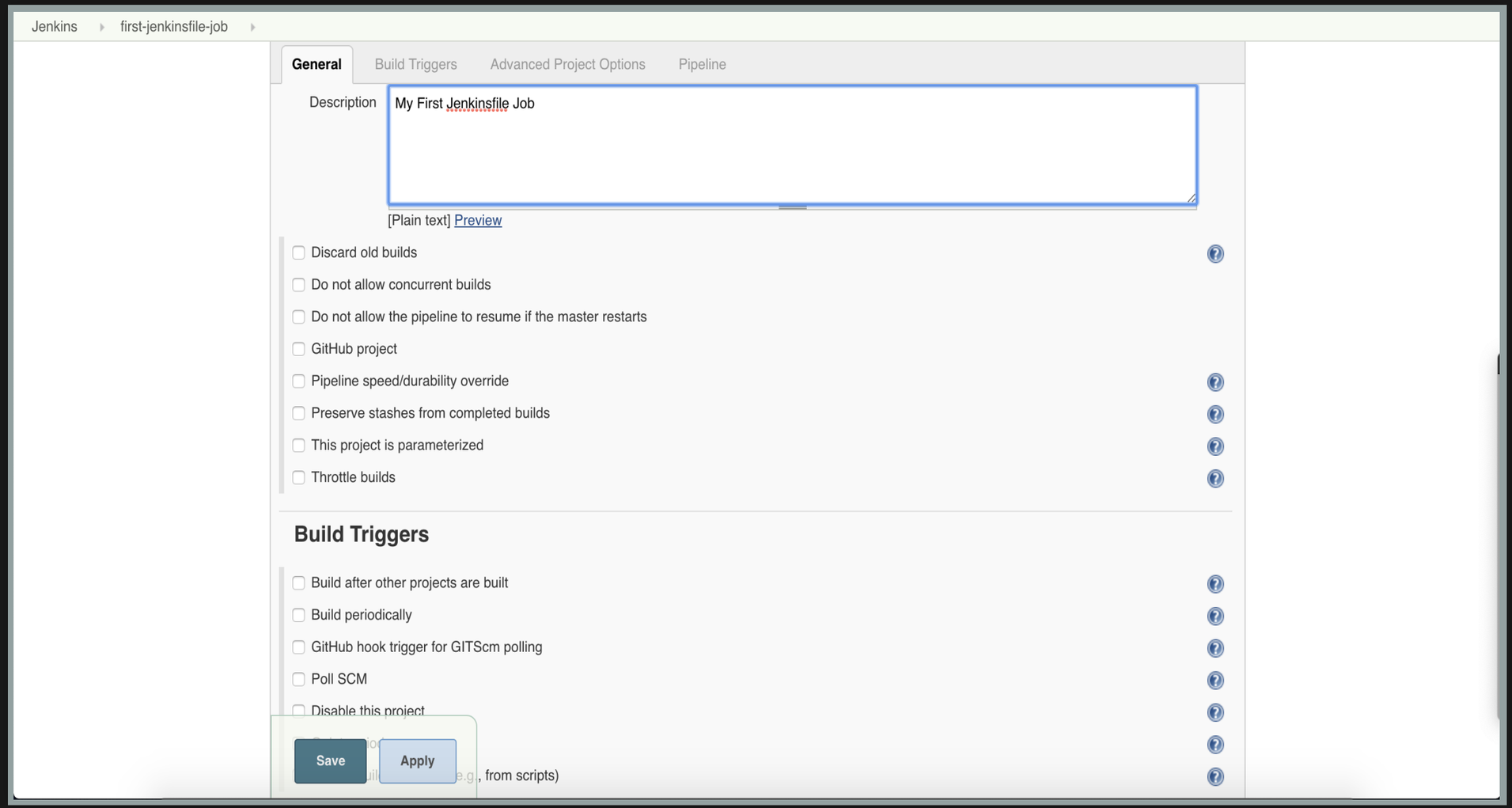
Go to the Jenkins landing page. Click New Item



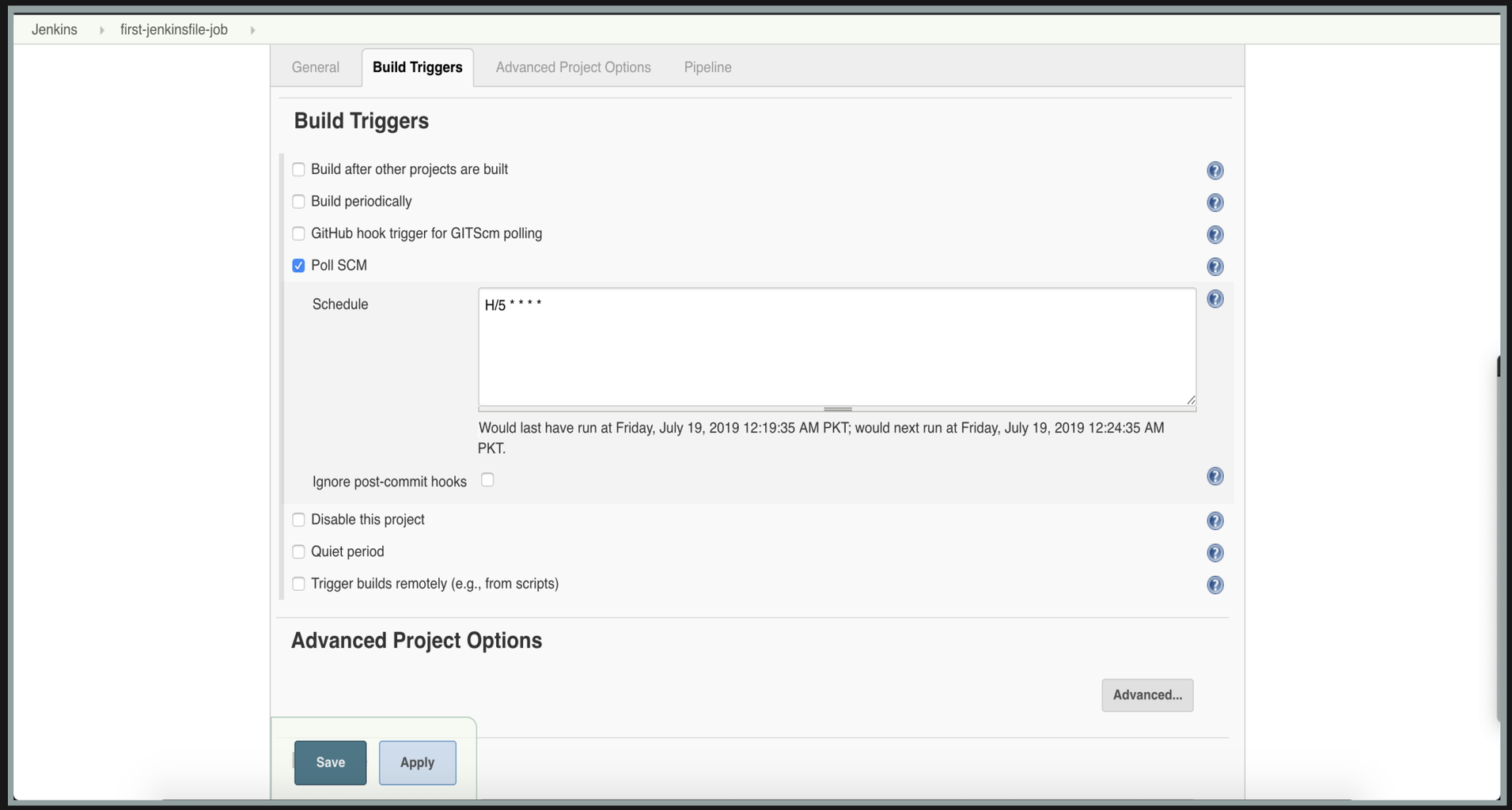
Enter job details, select Pipeline and Click OK



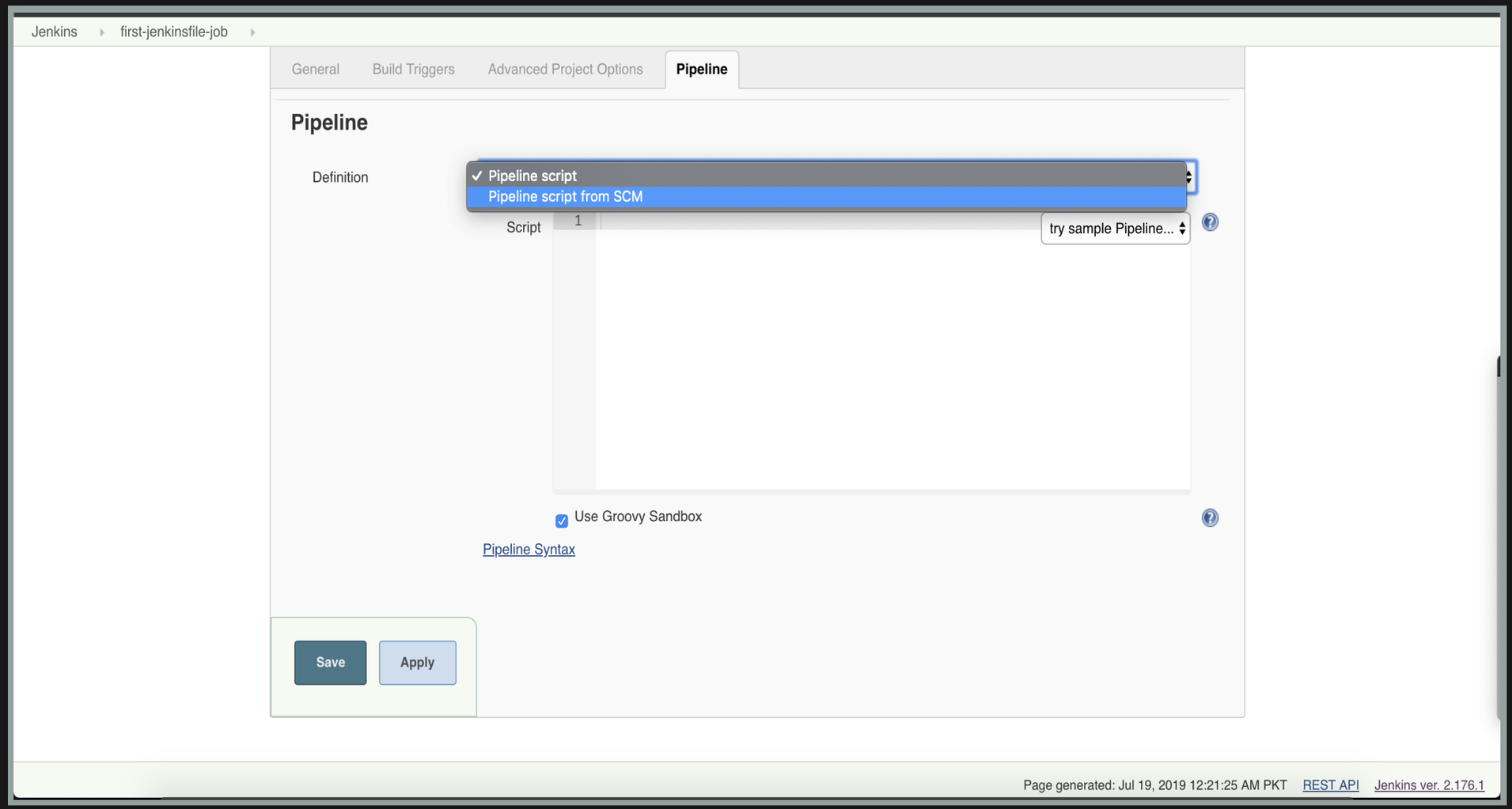
Enter description and click Build Triggers



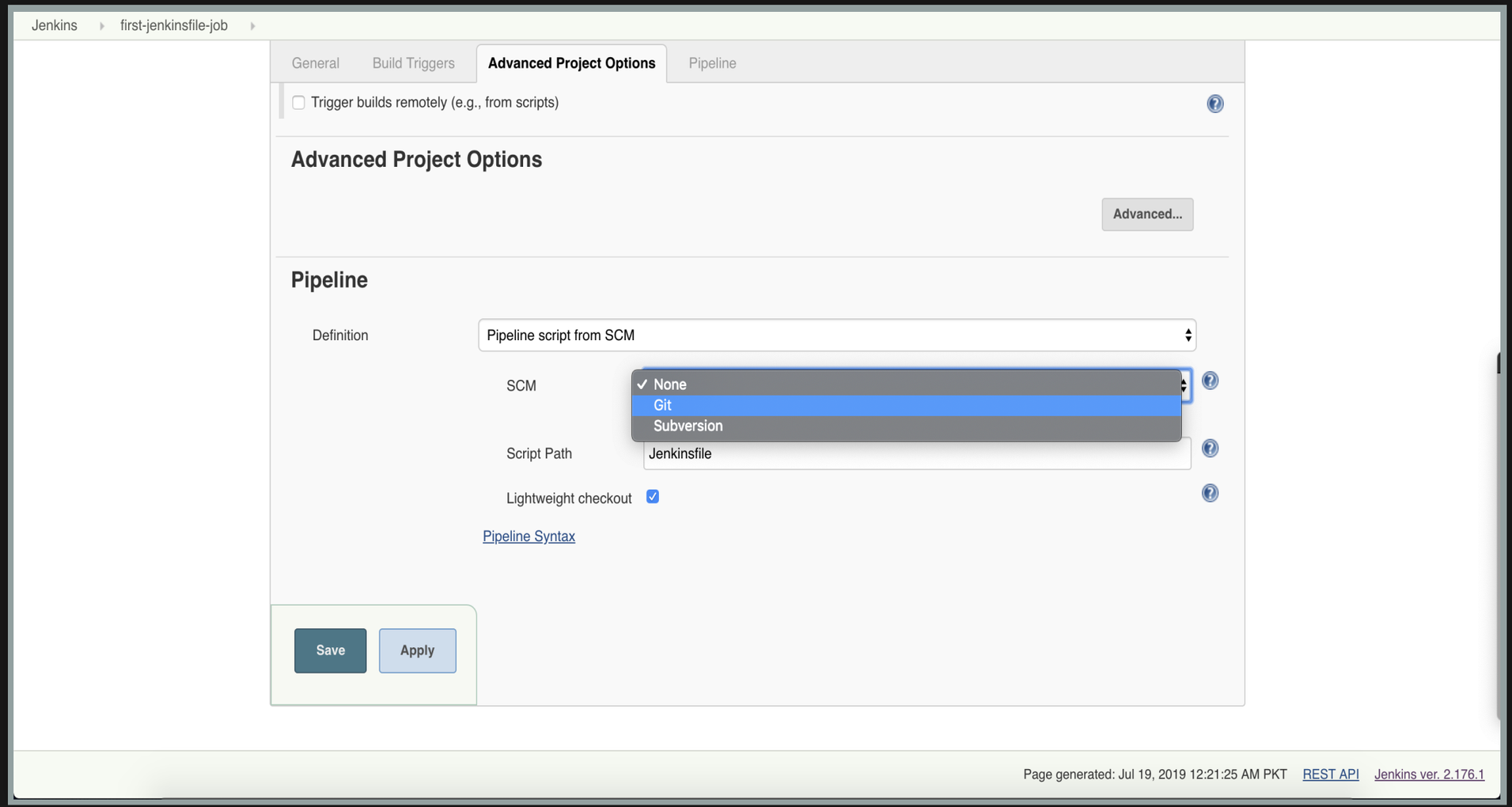
Select Poll SCM and enter schedule as shown below. Next click Pipeline



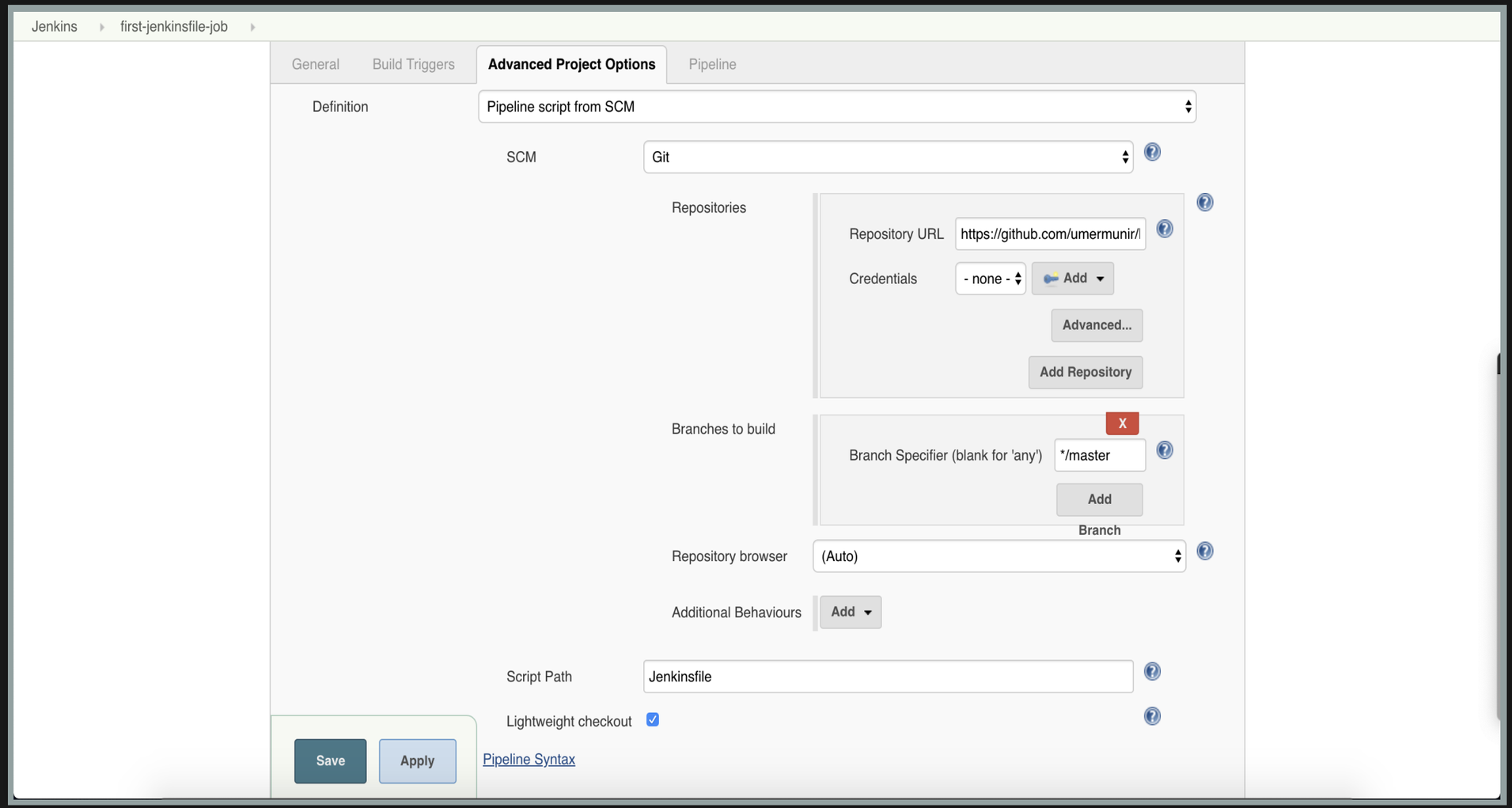
Select 'Pipleline script from SCM'



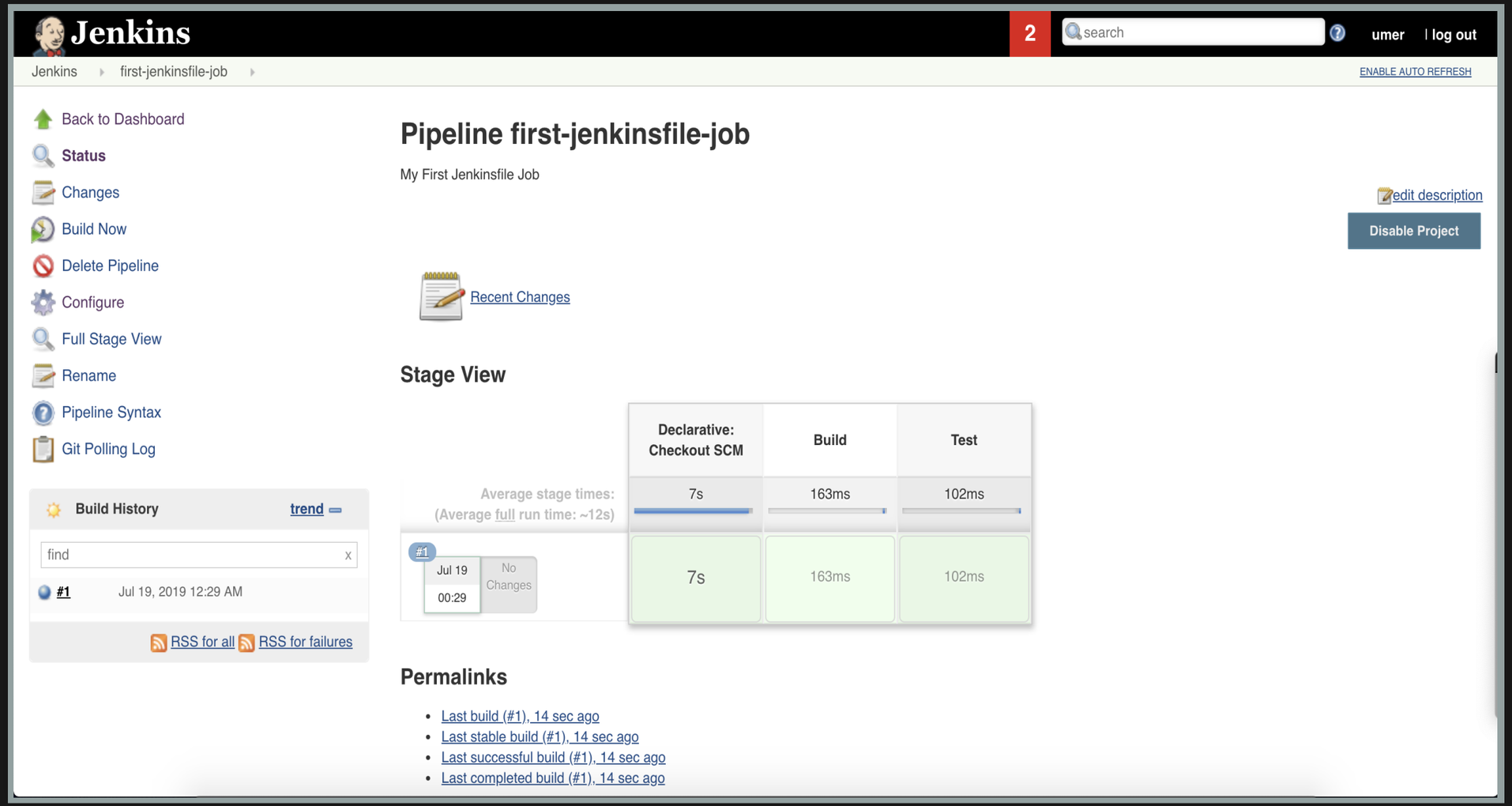
In the SCM, select Git



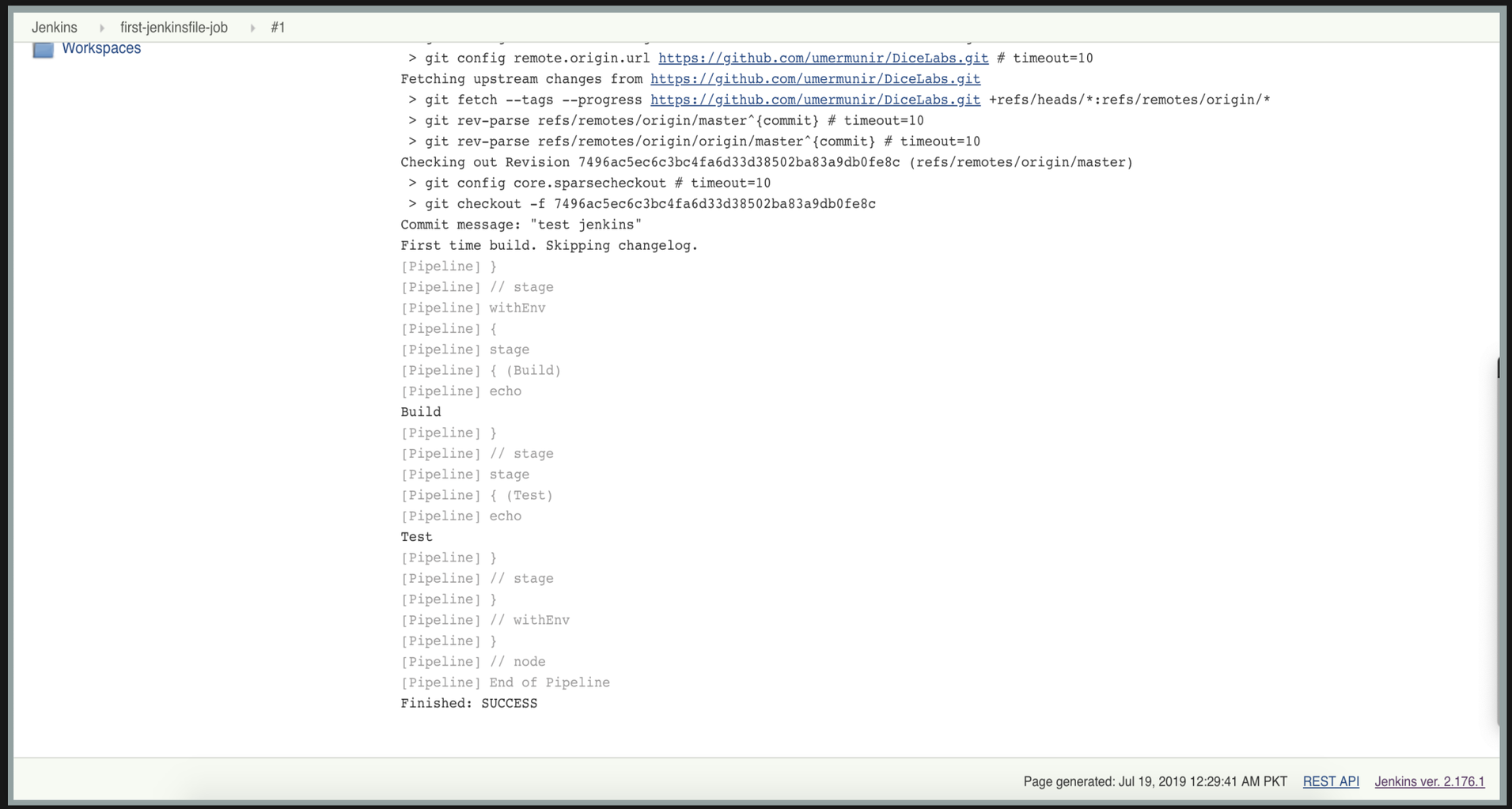
Enter the URL of repository that you forked from DiceLabs. Click Save



Next create a new file and push to remote repository and wait for the job to be triggered



Check Console Output

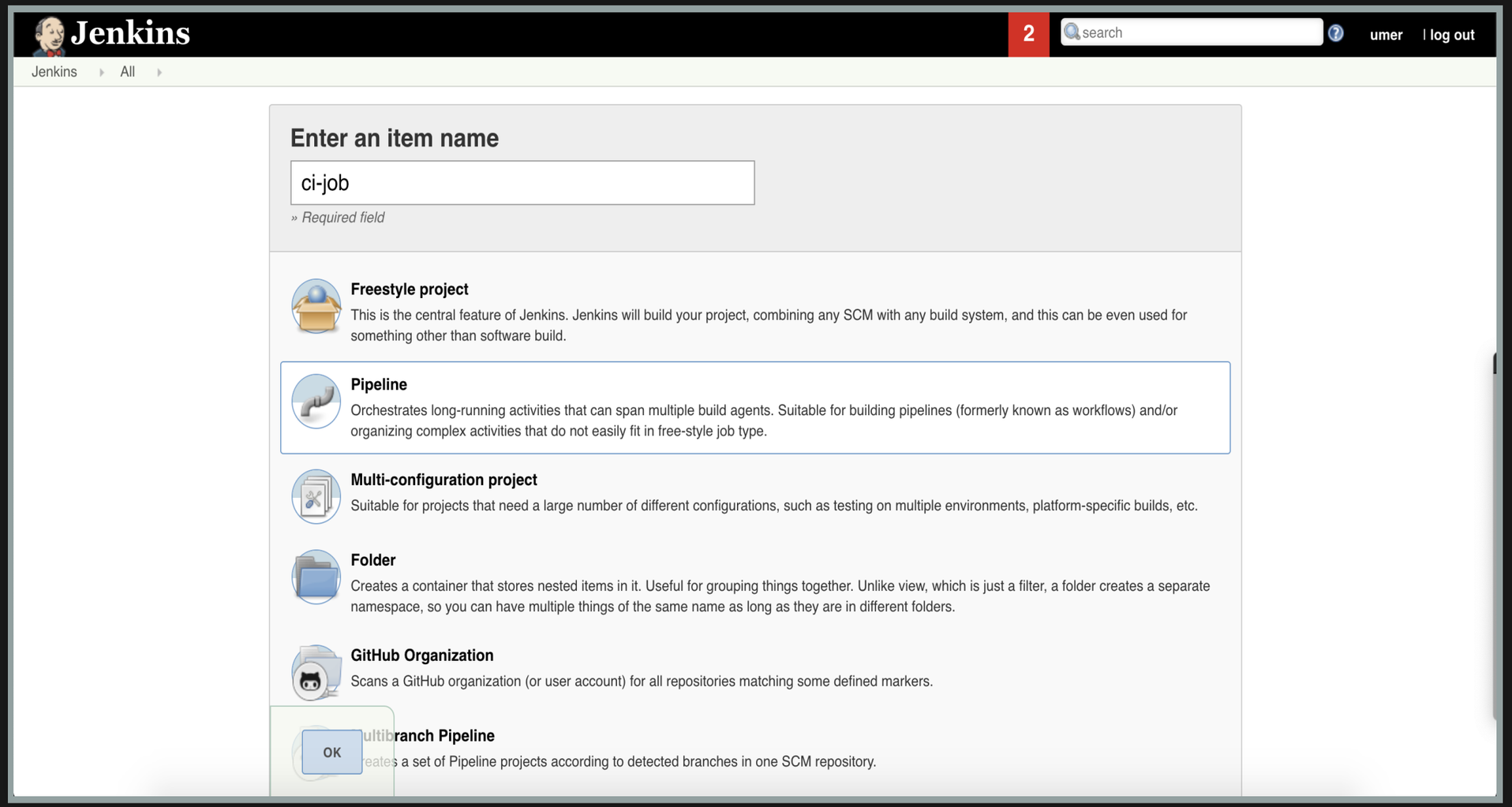


CONTINOUS INTEGRATION

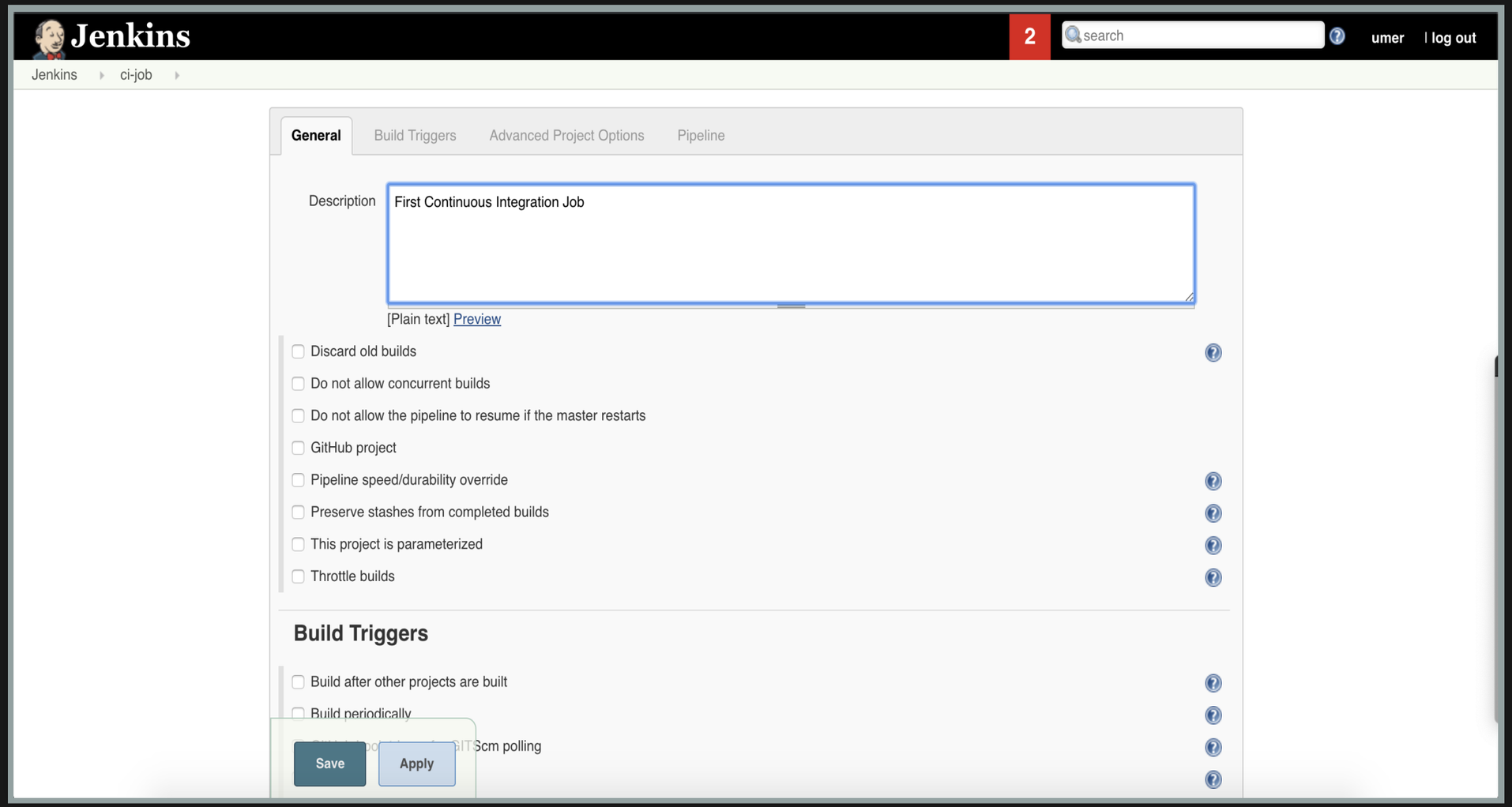
- Merge code changes from different developers into a central repository
- Automated builds and tests are run
- Key goals are to fail fast and find and address bugs quicker
- Benefits Developers most, Less merge conflicts

LAB - CONTINOUS INTERGRATION

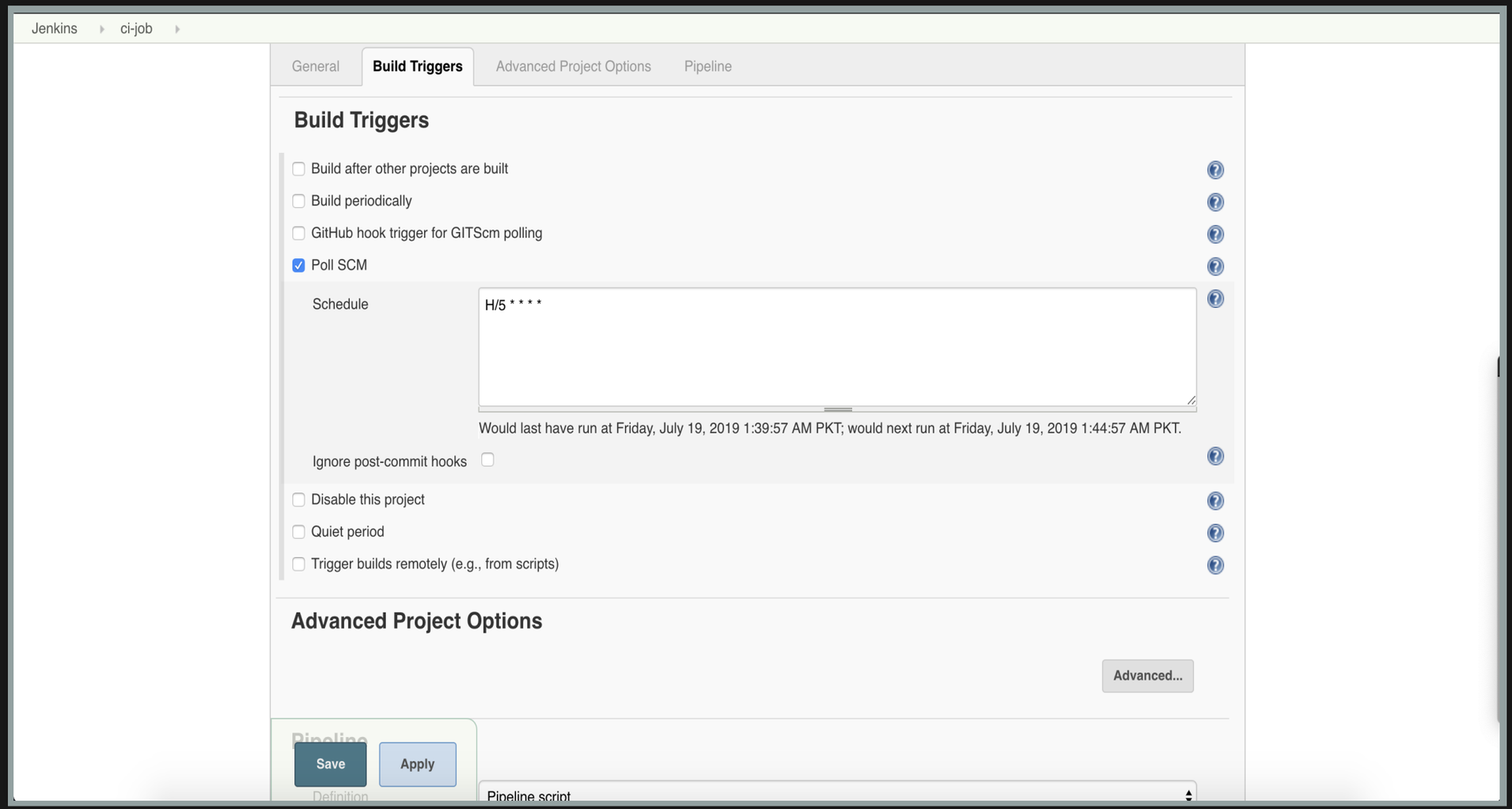
In this Lab, you are going to fork a simple java application, clone it, create a jenkins file in that, push to remote repository and then create a CI job. First create a new job



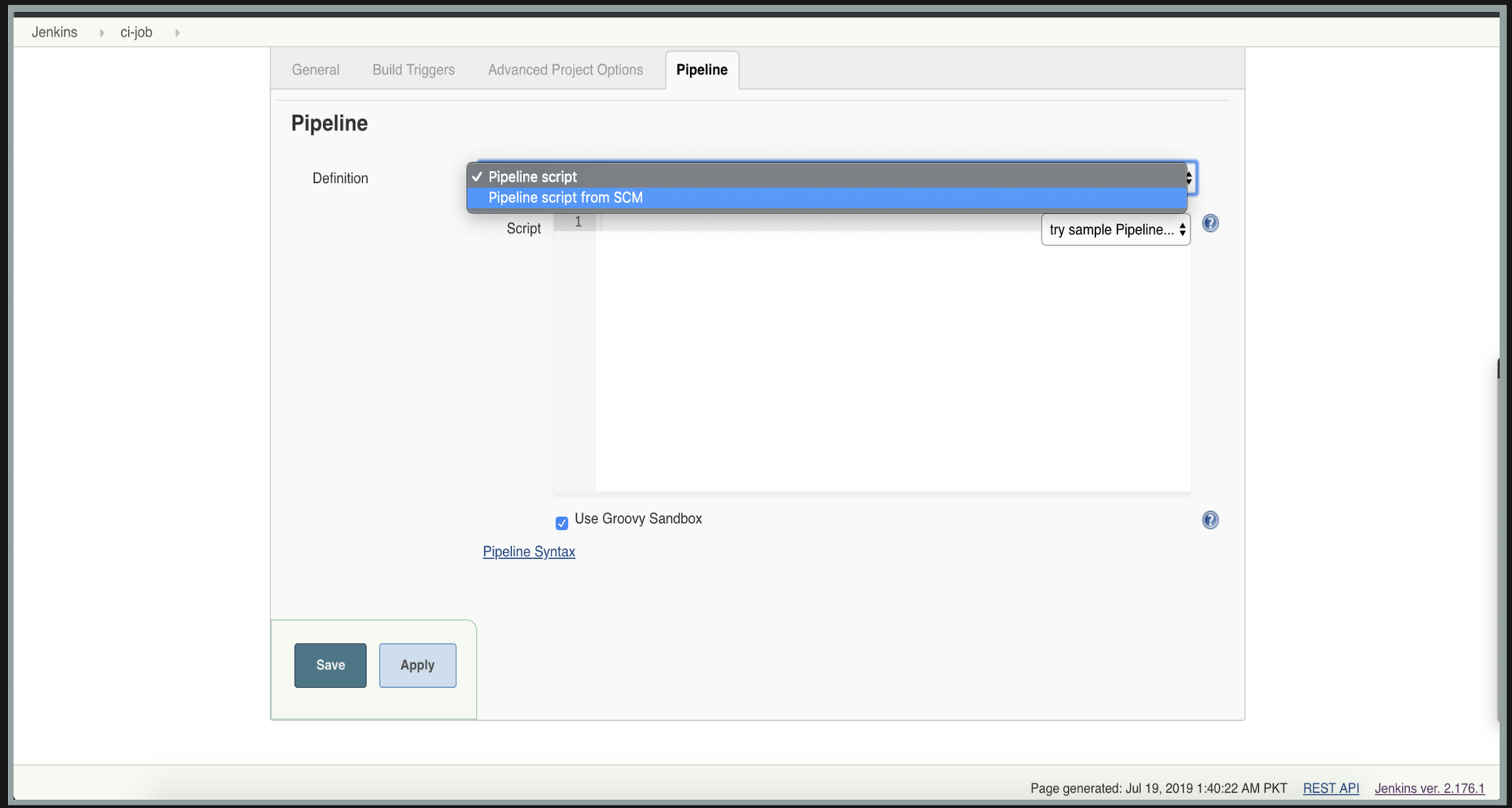
Enter description and Click Build Triggers



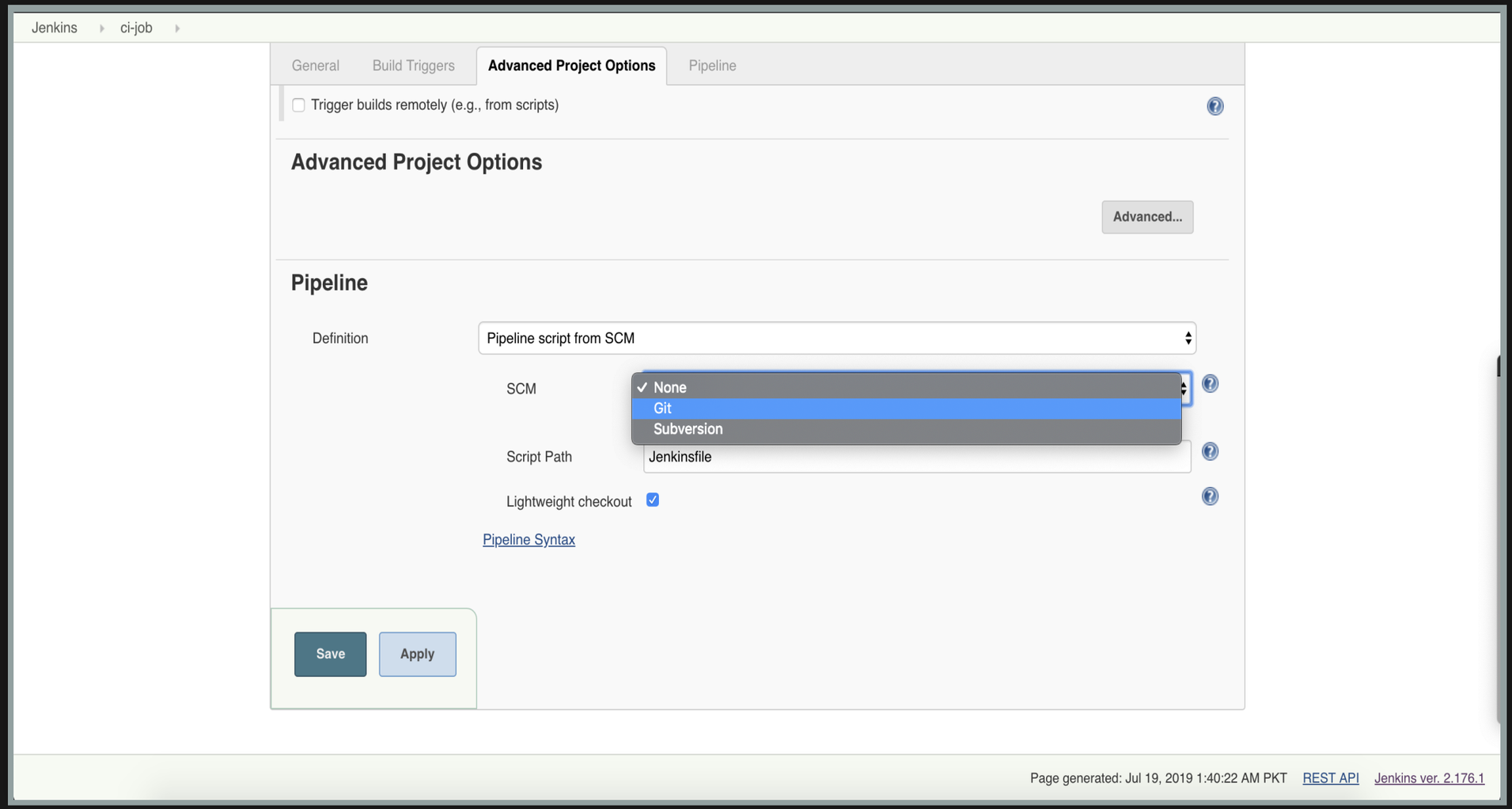
Select Poll SCM, specify the schedule and Click Pipeline



Select 'Pipeline script from SCM'



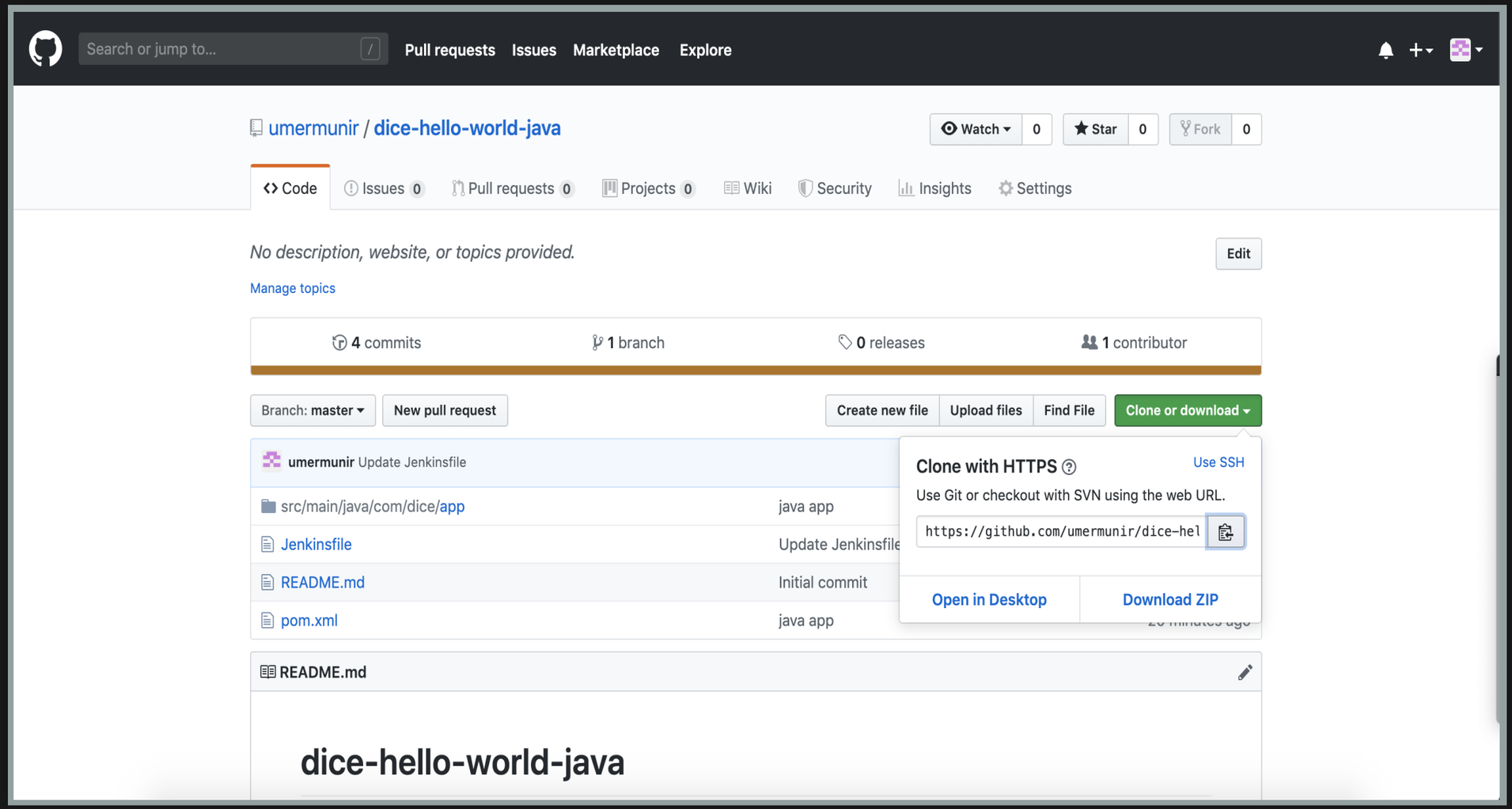
In the SCM, select Git



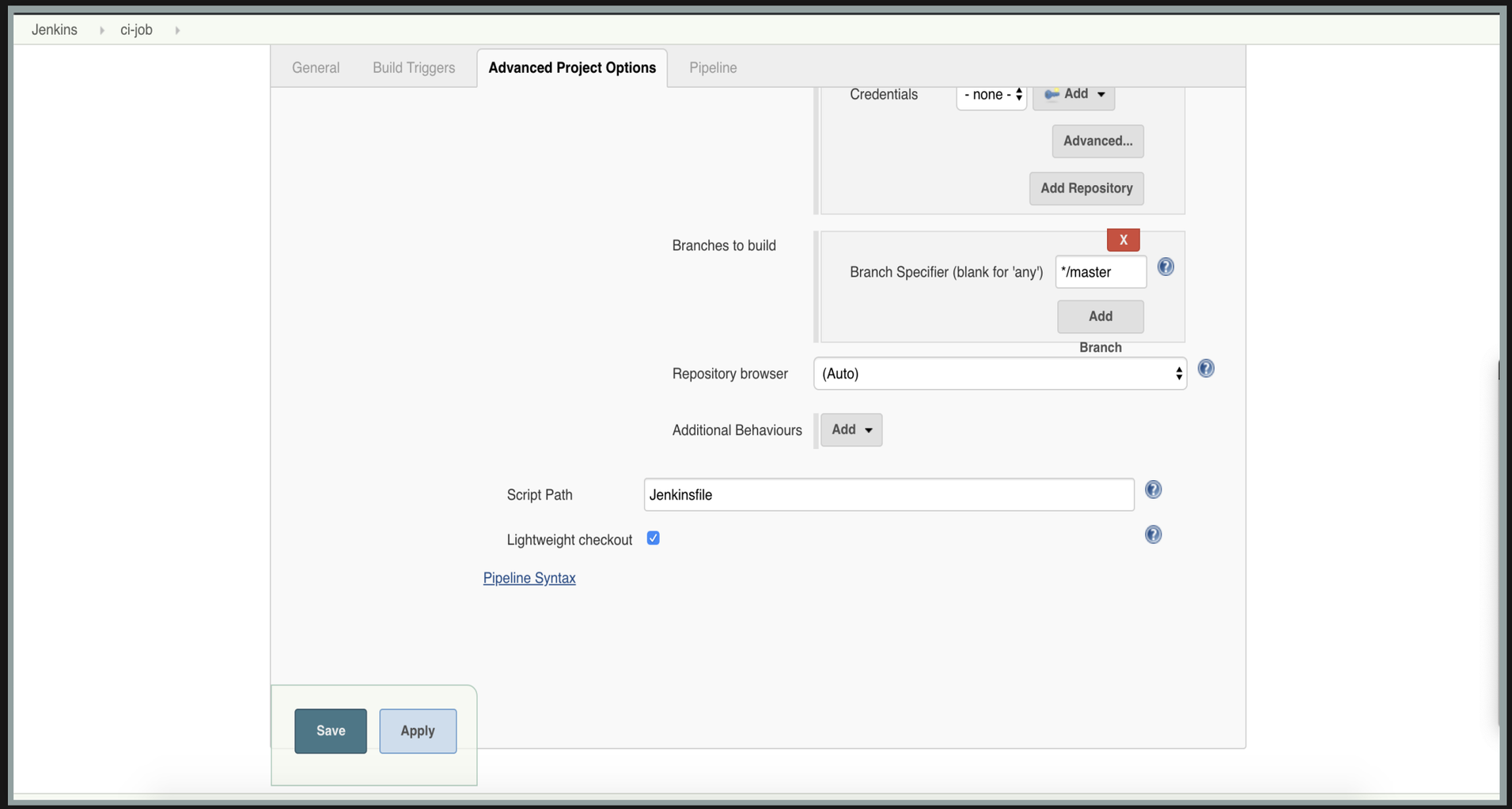
Fork the DICE Java Repo, clone it on your system. Create a Jenkinsfile in the root folder of the cloned repository and push it to remote repository

```
pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
        }
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'java -jar target/hello-world-1.jar'
            }
        }
    }
}
```

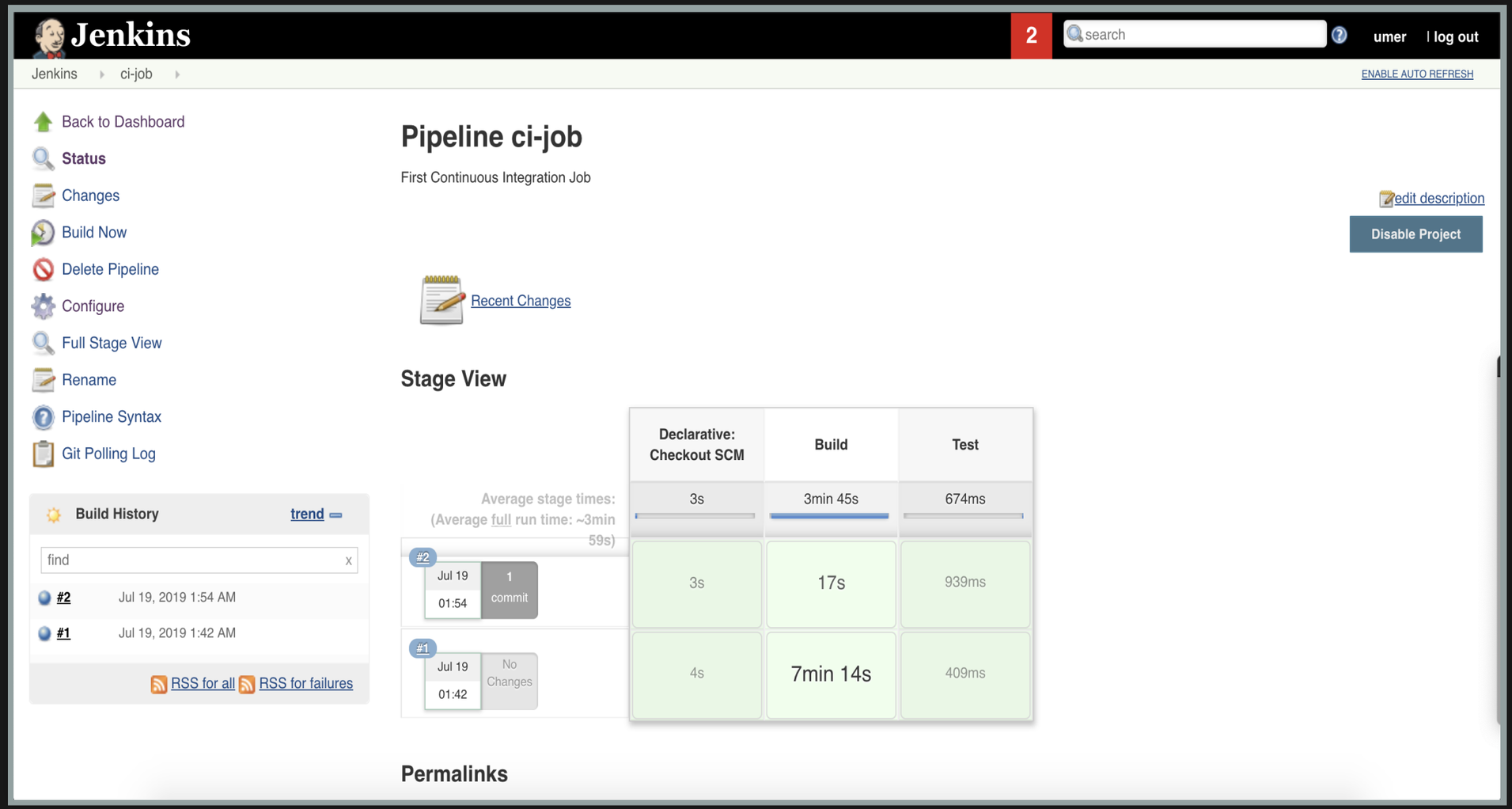
Copy the URL of forked repository as shown below



Enter the Repository URL in the Pipeline section and Click Save



Now create a new file in the forked repository and push it to remote repository that should trigger the build



Check Console Output

Jenkins ▸ ci-job ▸ #2

```
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ hello-world ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ hello-world ---
[INFO] Building jar: /Users/um255003/.jenkins/workspace/ci-job/target/hello-world-1.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13.268 s
[INFO] Finished at: 2019-07-18T20:54:35Z
[INFO] Final Memory: 16M/151M
[INFO] ------------------------------------------------------------------------
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] sh
+ java -jar target/hello-world-1.jar
Hello World!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker stop --time=1 3ebe94b02e8c1653703878f881d4003e8ebac957ef4ad38d85bc6a4cc96cce06
$ docker rm -f 3ebe94b02e8c1653703878f881d4003e8ebac957ef4ad38d85bc6a4cc96cce06
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Page generated: Jul 19, 2019 1:55:52 AM PKT REST API Jenkins ver. 2.176.1

CONTINOUS INSPECTION

- Can be used for Continous Inspection
  - Static Code Analysis
  - Measure Code Quality
  - Identify non-complaint code
  - Fix code quality issue
  - Used to generate reports
- Plugins
  - Warnings Next Generation
  - SonarQube

CONTINOUS DELIVERY

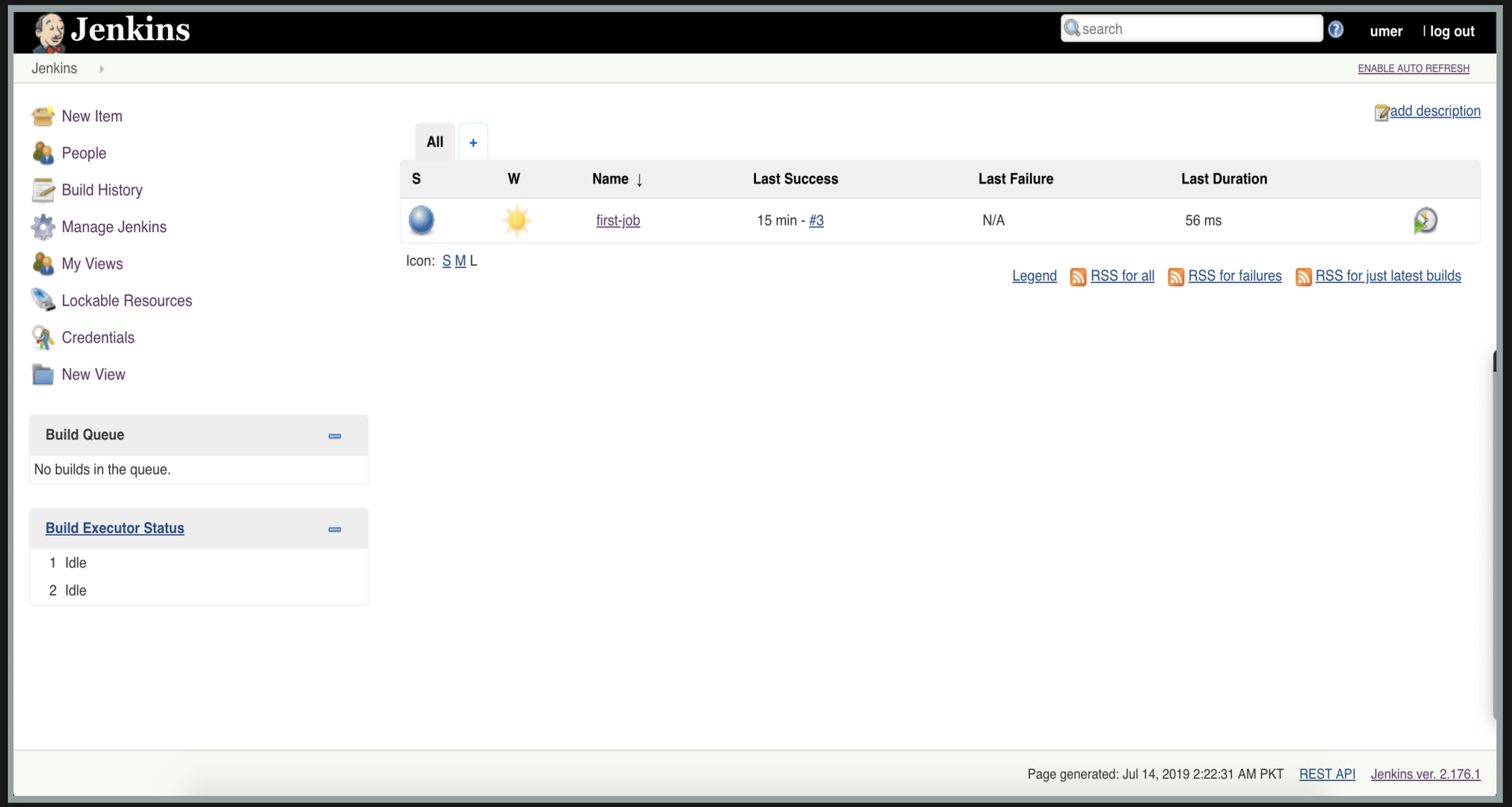
- CI stage is approved
- A small build cycle for short sprints for releasing small features
- Code changes are automatically built & tested
- Can be deployed to a test environment
- Can use branching strategy (other than master)
- Mindset to always have a deployment-ready build artifact.

LAB - CONTINOUS DELIVERY

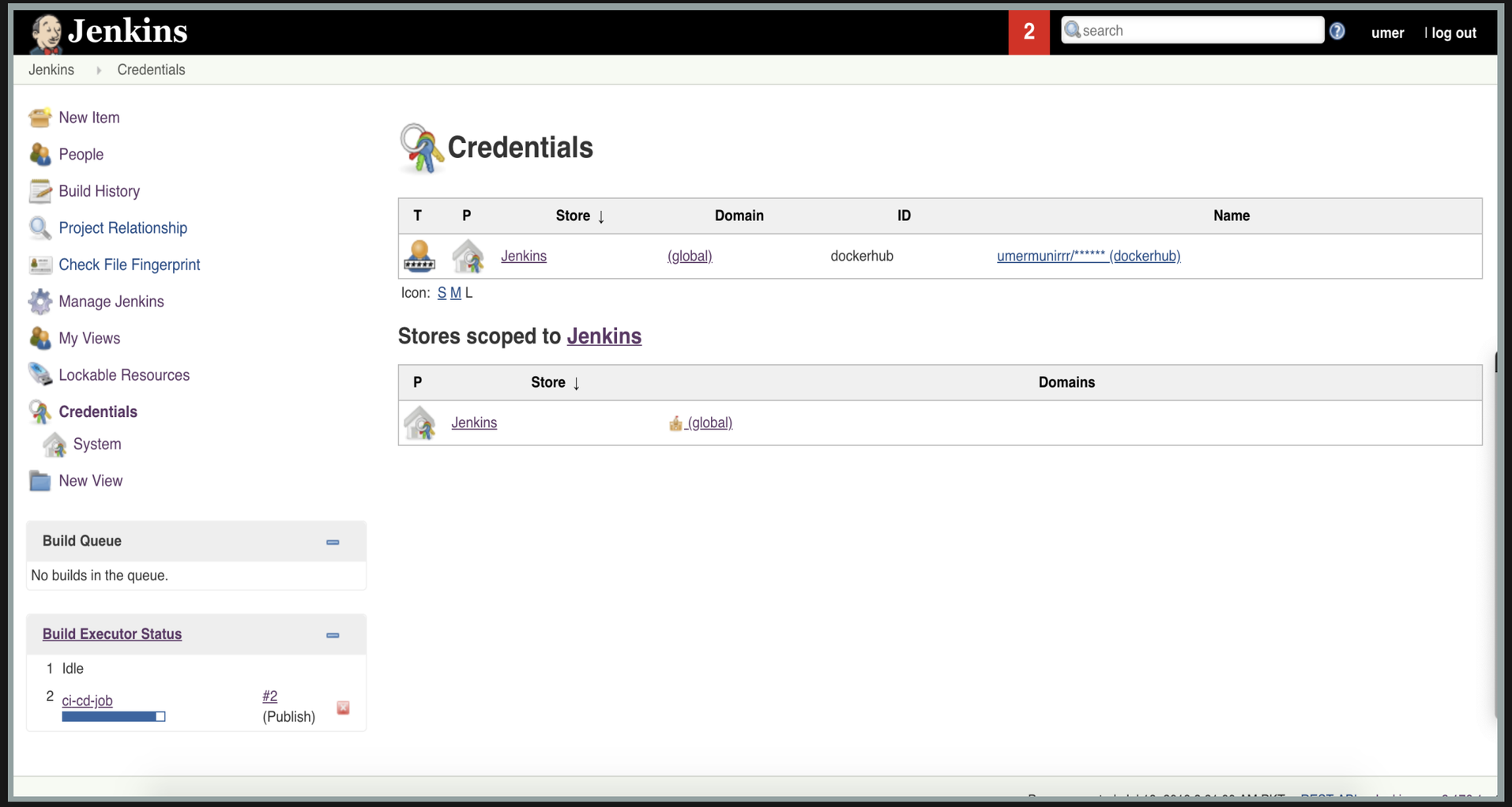
- Docker
- Python

DOCKER

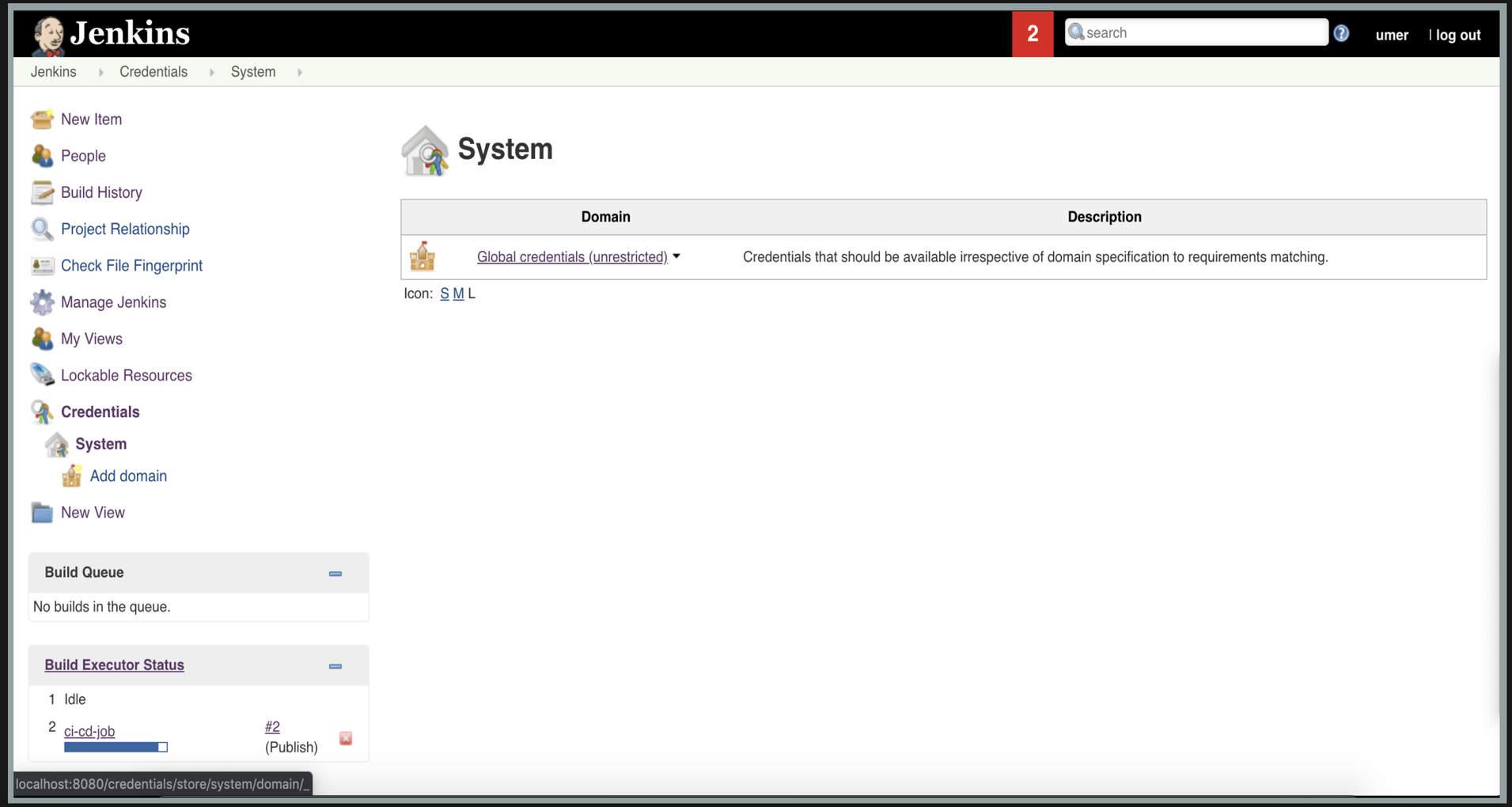
In this Lab, you are going to fork a repository, clone it, create a jenkins file, push to remote repository and then create CI/CD job. You will use the jenkins pipeline to create docker image, test it and then push to docker hub. First you are going to add dockerhub by clicking Credentials



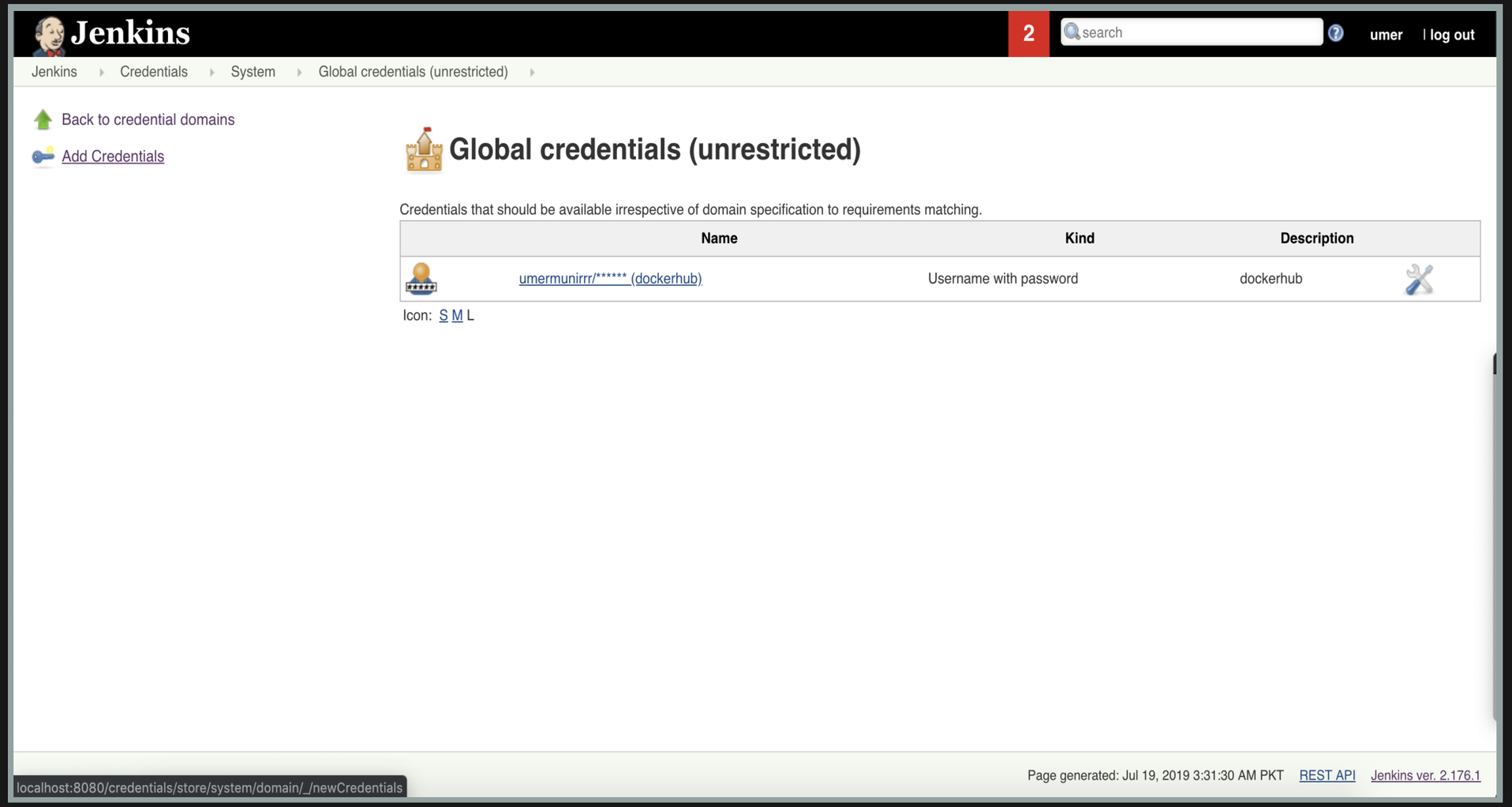
Click System



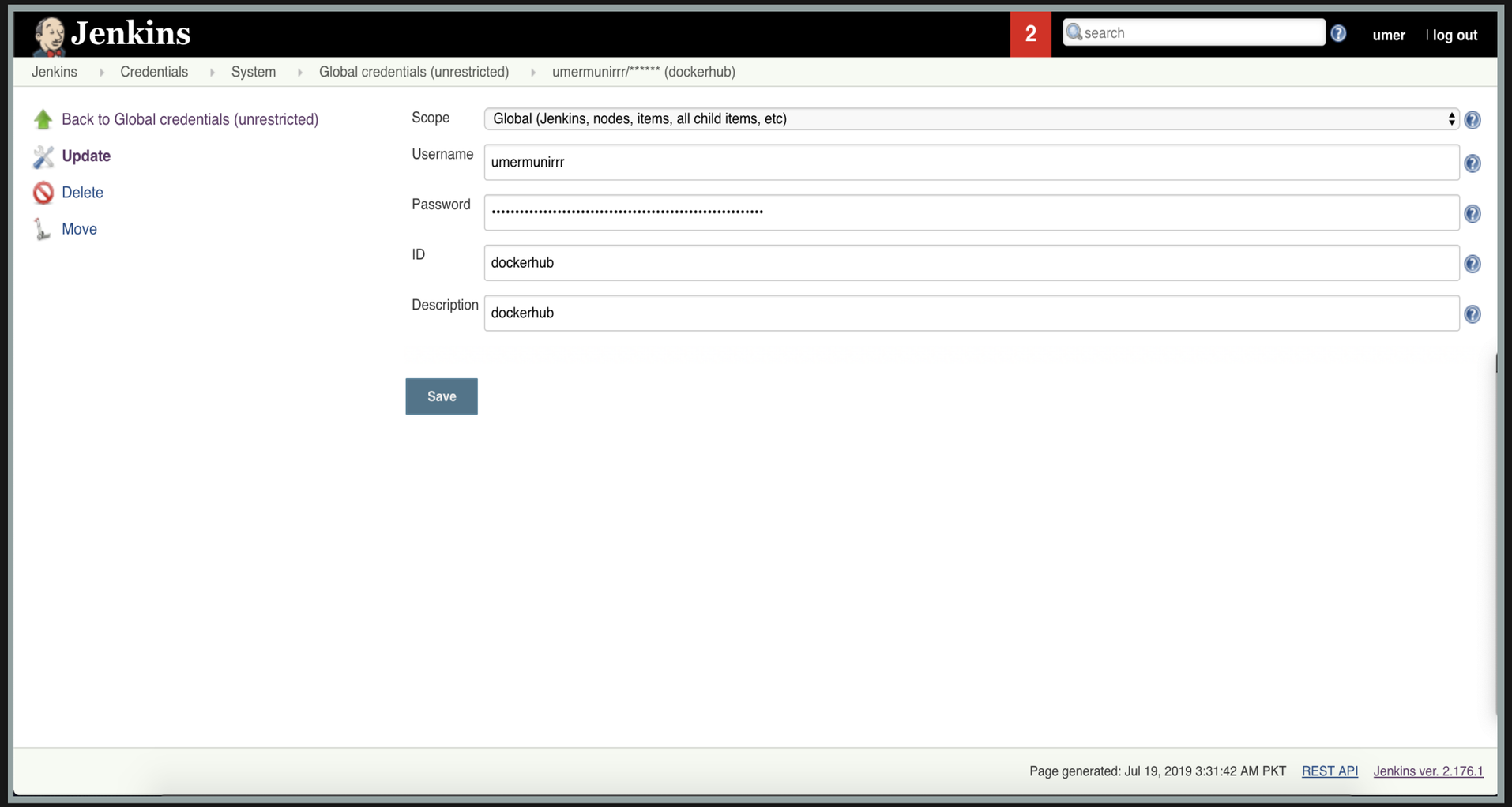
Click Global credentials



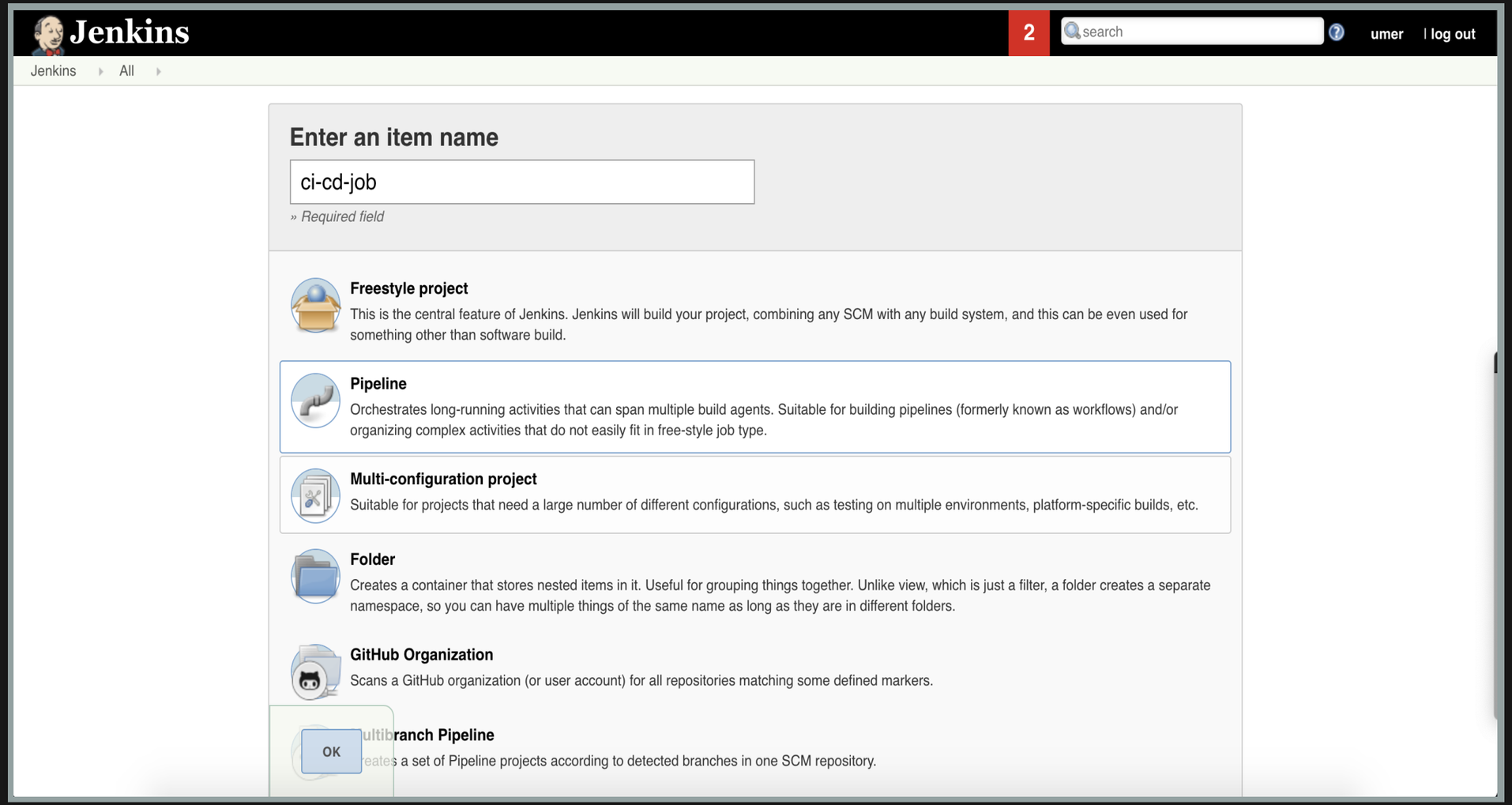
Click Add Credentials



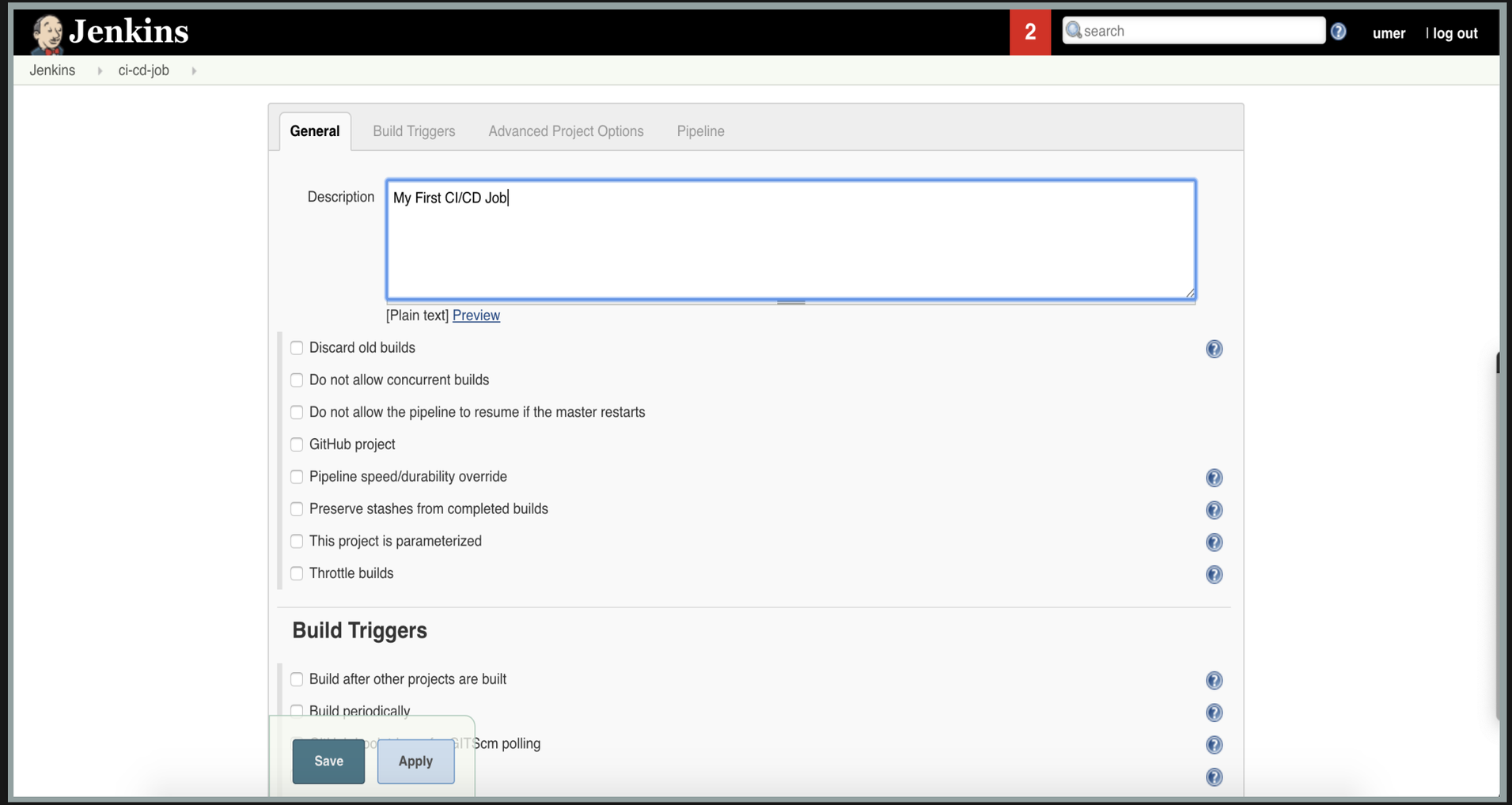
Specify your dockerhub username, password and credential id as shown below. You will be using this credential id in Jenkins file



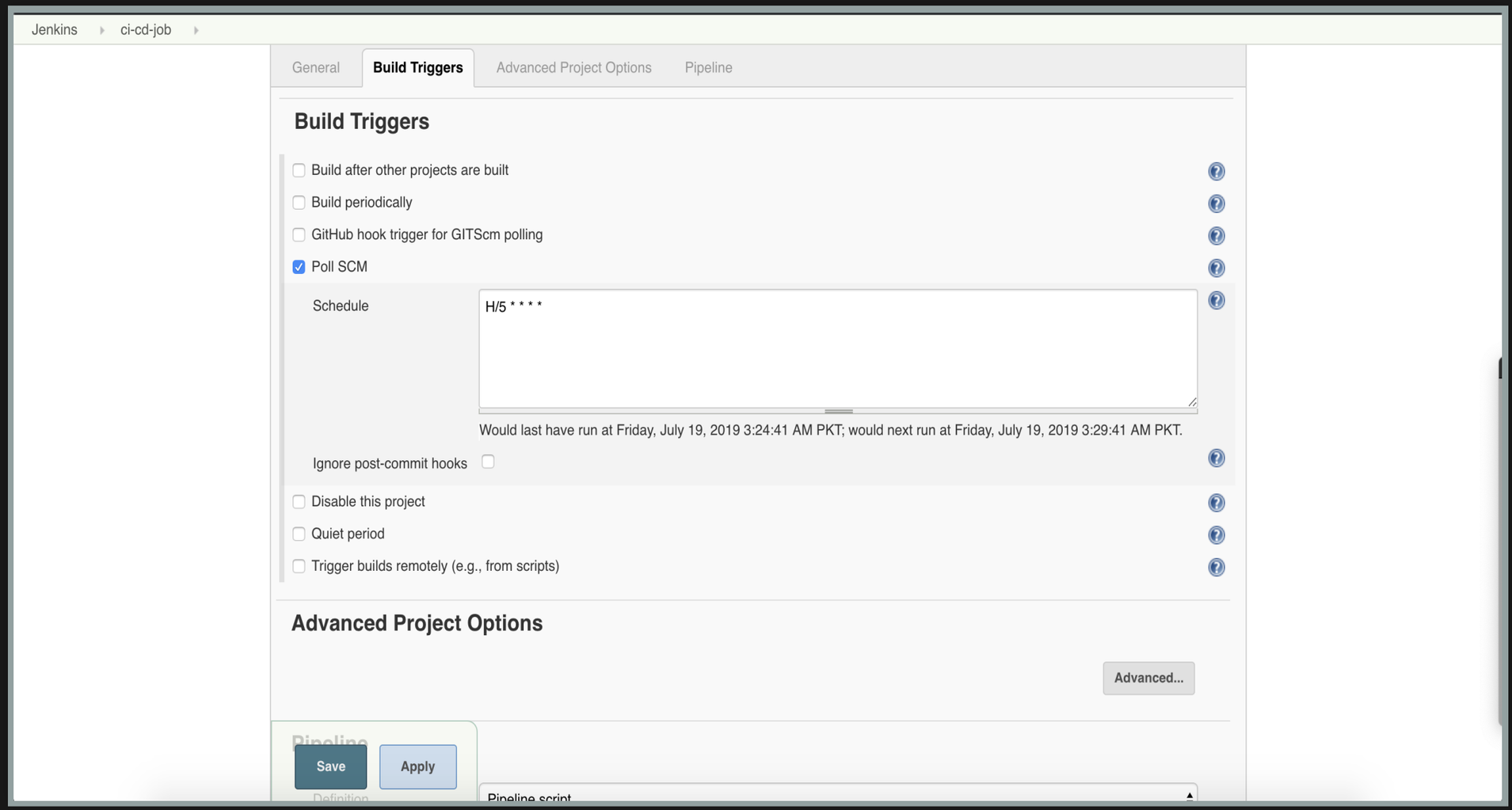
Go to landing page, click New Item and Enter job details



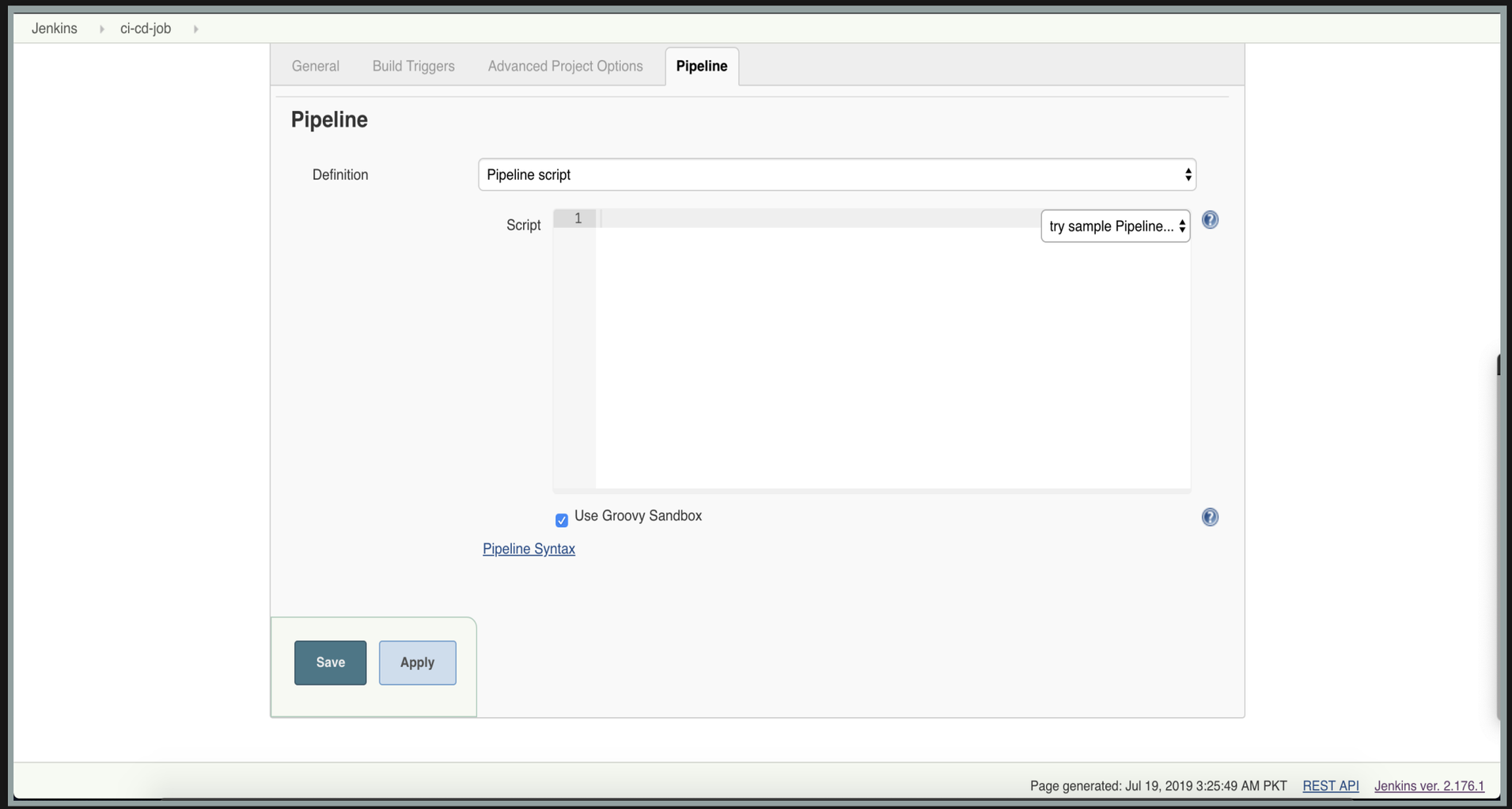
Enter job description and Click Build Triggers



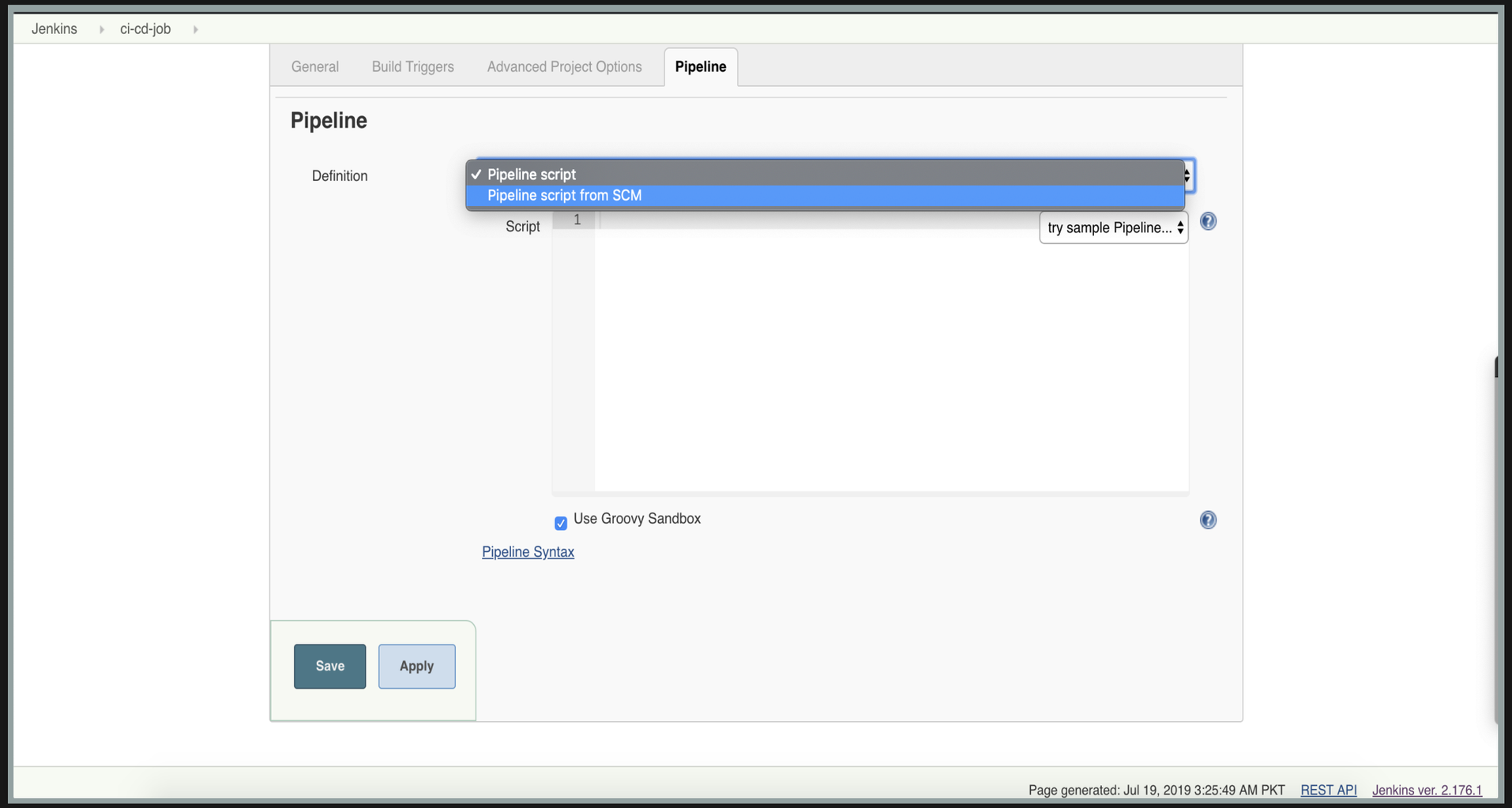
Select Poll SCM and specify schedule as shown below



Select Pipeline



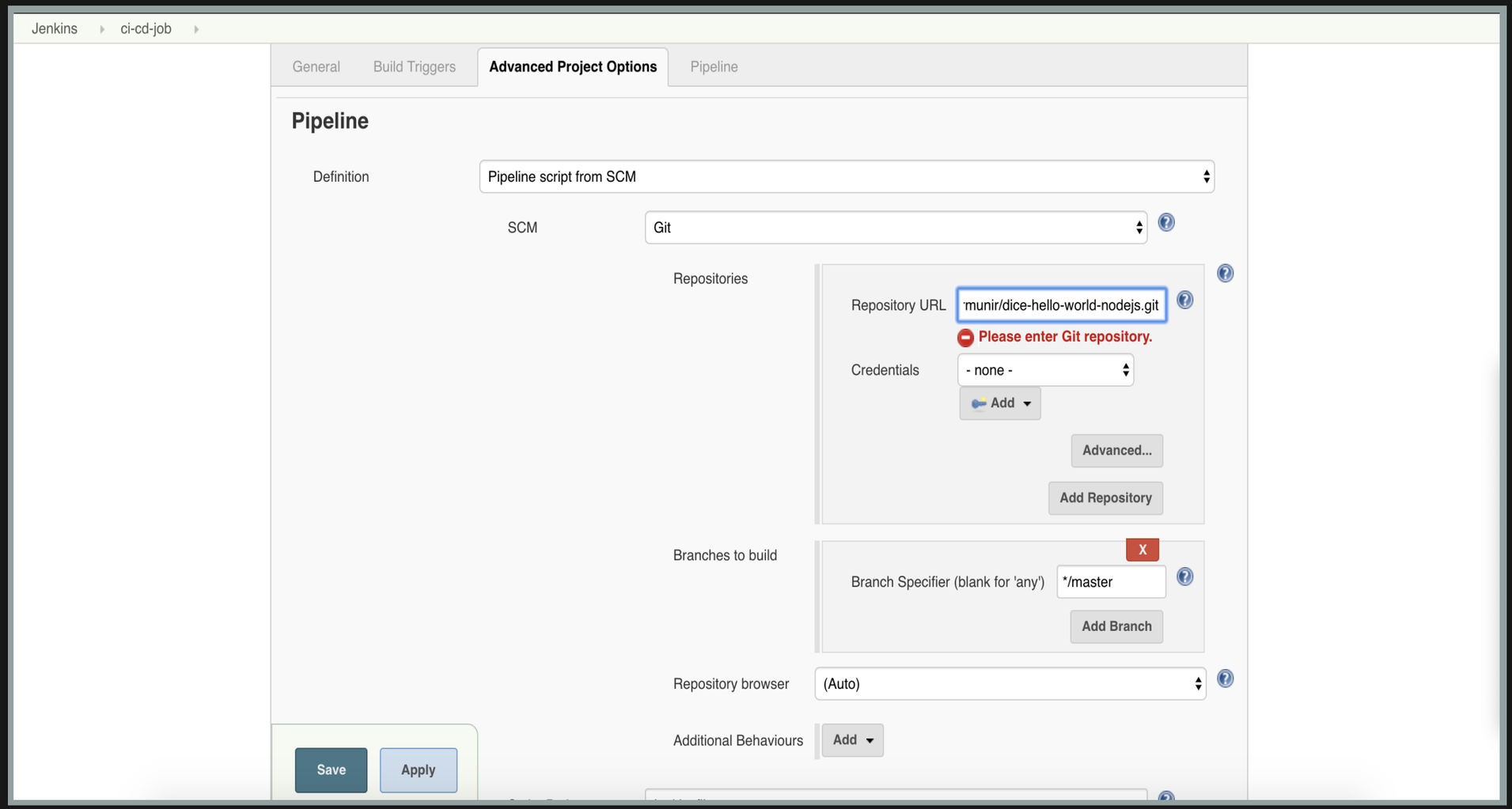
In the Definition select 'Pipeline script from SCM' and select 'Git' in SCM



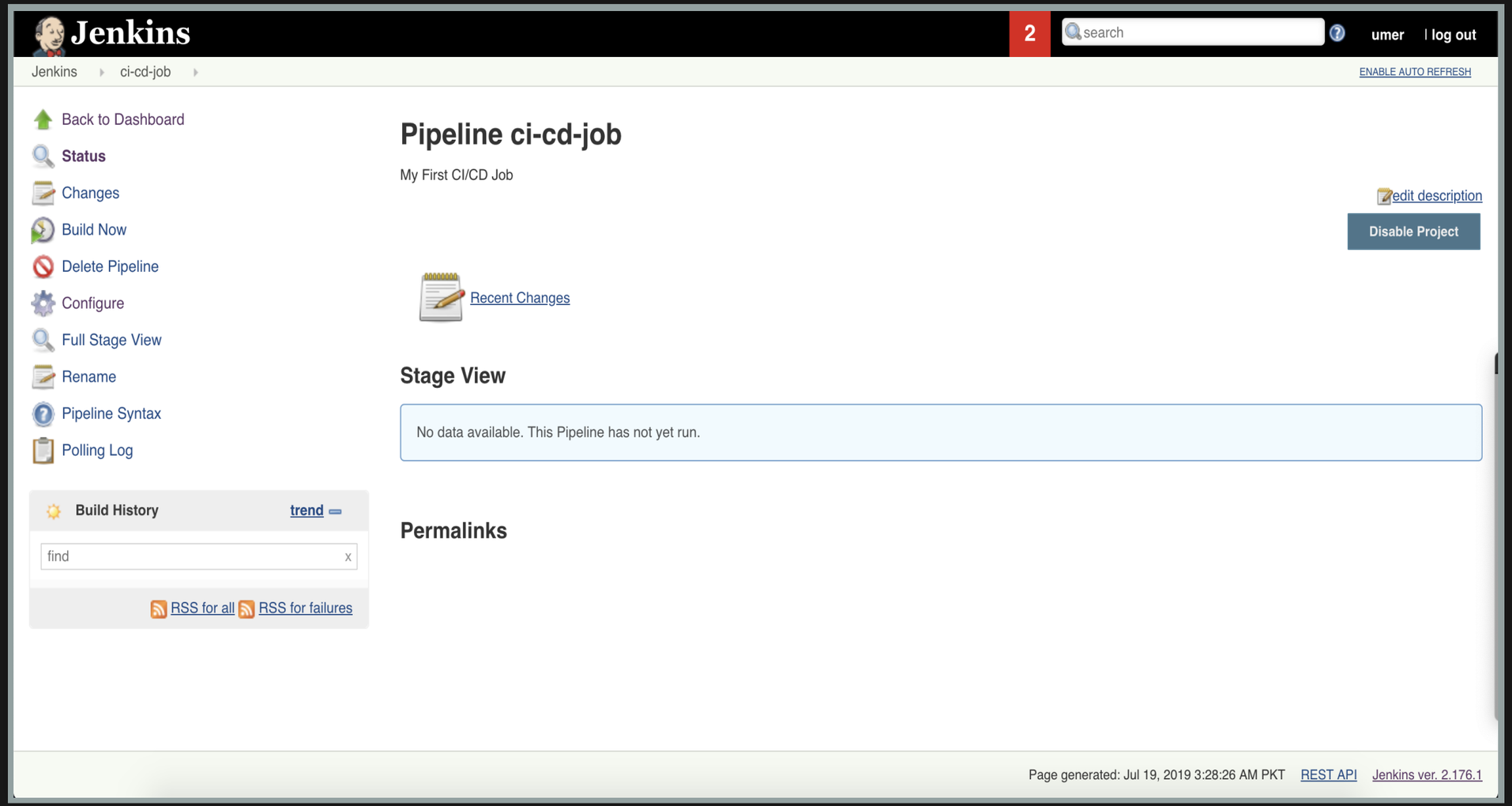
Fork DICE Docker Img repository, clone it, create a Jenkins file below and then push it.

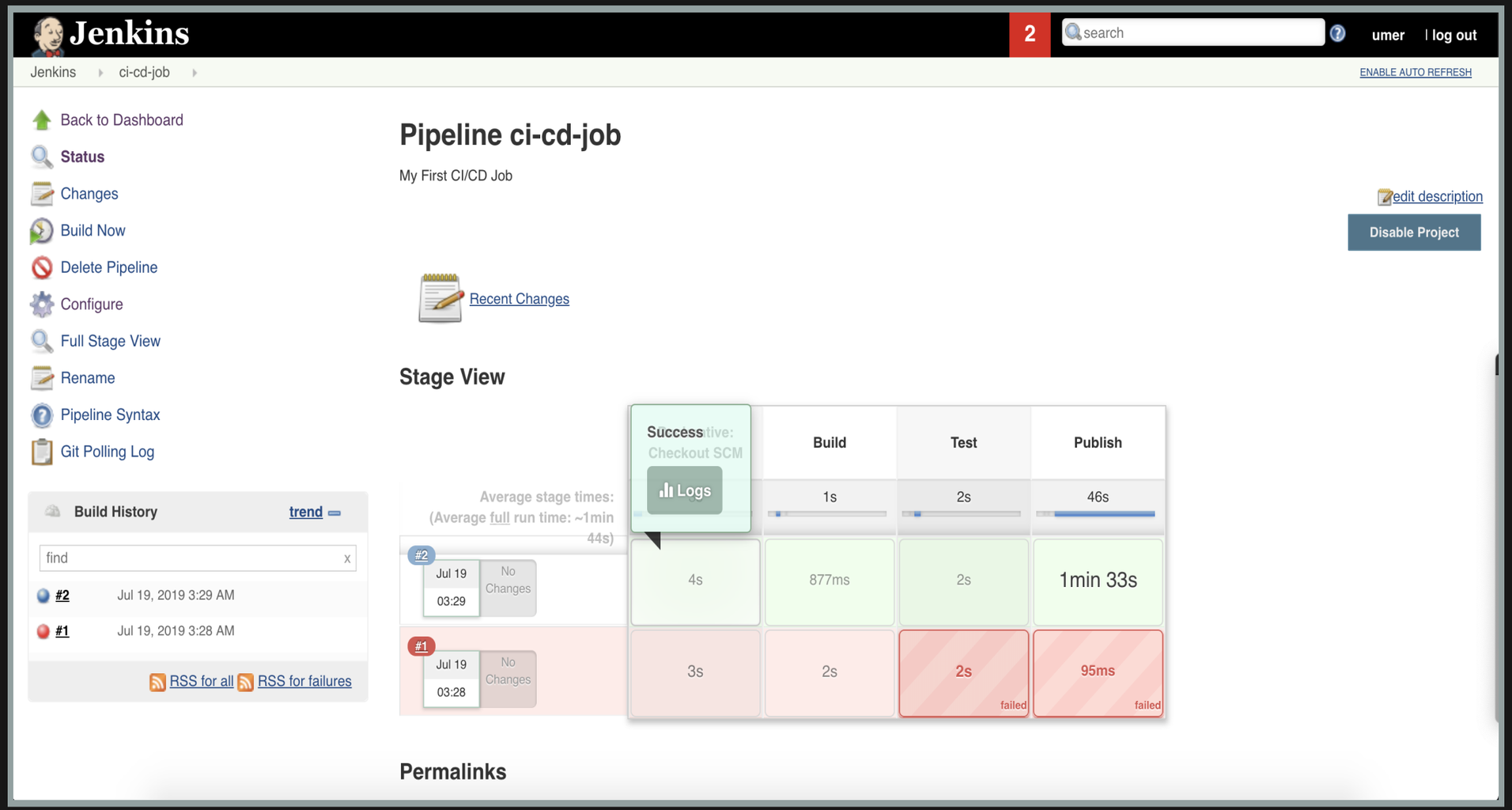
```
pipeline {
  agent any
  environment {
    registryCredential = 'dockerhub'
  }
  stages {
    stage('Build') {
      steps {
        sh 'docker build -t umermunirrr/test-node-app .'
      }
    }
    stage('Test') {
      steps {
        sh 'docker container rm -f node || true'
        sh 'docker container run -p 8001:8080 --name node -d umermunirrr/test-node-app'
        sh 'curl -I http://localhost:8001'
      }
    }
    stage('Publish') {
        steps{
            script {
                docker.withRegistry( '', registryCredential ) {
                  sh 'docker push umermunirrr/test-node-app:latest'
                }
            }
        }
      }
    }
}
```

Copy the URL of forked repository, specify that in the Repository URL and click Save



Create a new file in the forked repository and wait for it to trigger the build




Jenkins
2
search
umer
log out
Jenkins
ci-cd-job
ENABLE AUTO REFRESH
Back to Dashboard
Status
Changes
Build Now
Delete Pipeline
Configure
Full Stage View
Rename
Pipeline Syntax
Git Polling Log
Build History
trend
find
#2 Jul 19, 2019 3:29 AM
#1 Jul 19, 2019 3:28 AM
RSS for all
RSS for failures
Pipeline ci-cd-job
My First CI/CD Job
edit description
Disable Project
Recent Changes
Stage View
Success
Checkout SCM
Logs
Build
Test
Publish
Average stage times:
(Average full run time: ~1min 44s)
1s
2s
46s
#2
Jul 19
03:29
No Changes
4s
877ms
2s
1min 33s
#1
Jul 19
03:28
No Changes
3s
2s
2s
failed
95ms
failed
Permalinks

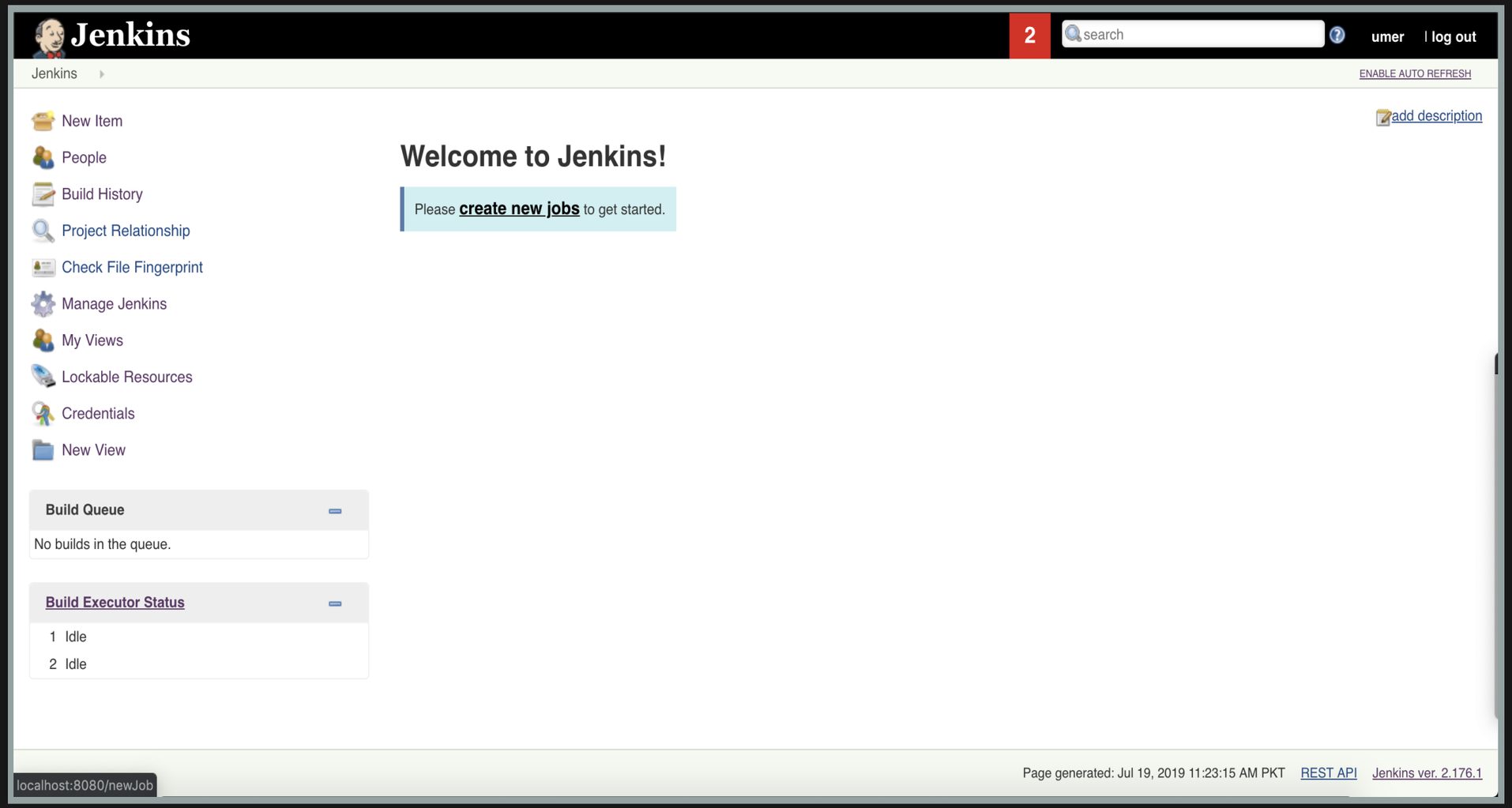
Check Console Output

Jenkins ▸ ci-cd-job ▸ #2

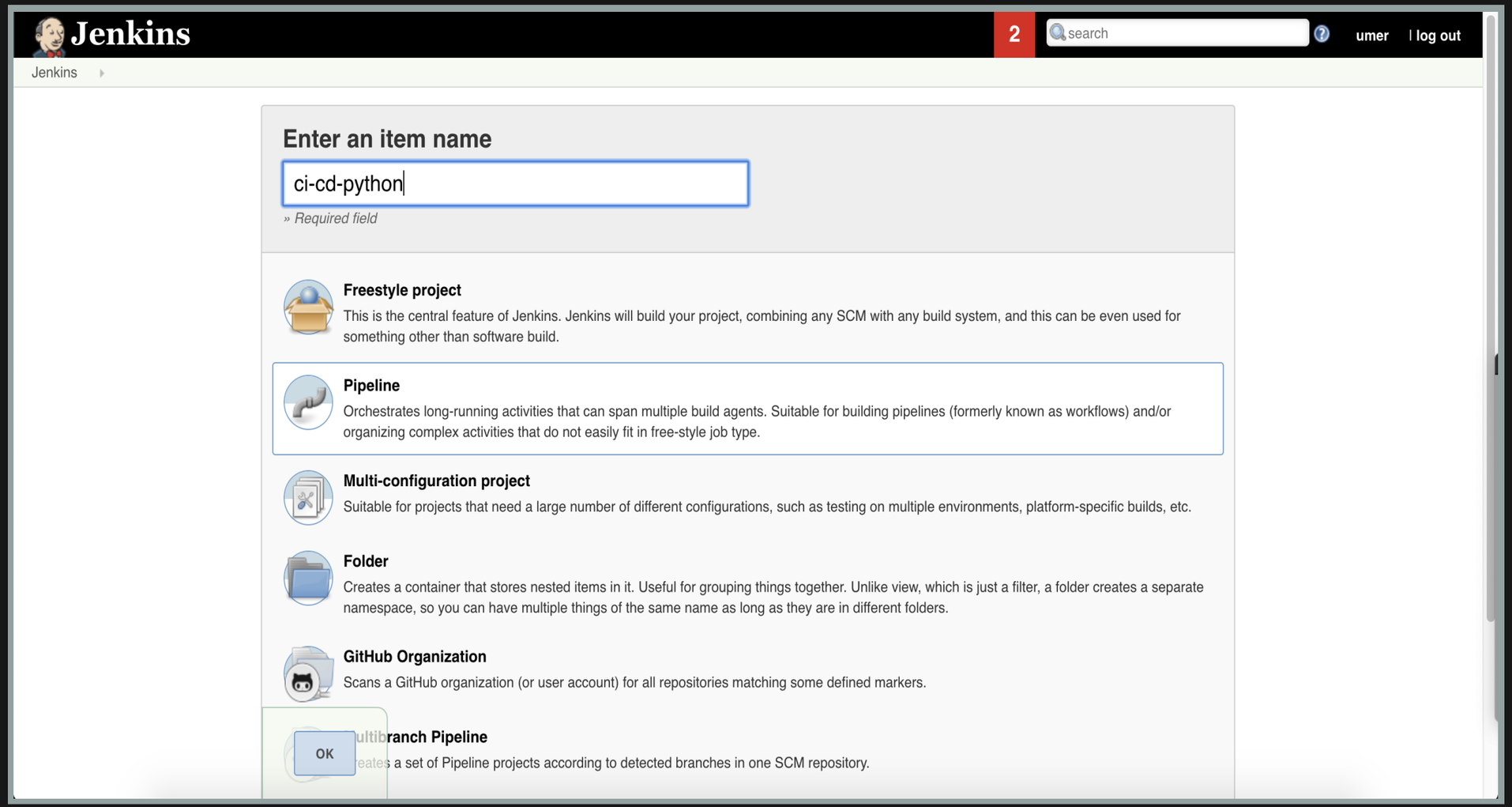
```
15210a41d4ee: Waiting
e2a8a00a83b2: Waiting
7bff69ee8682: Pushed
76f4ec530e1a: Pushed
452bed668936: Pushed
a0822d54fb0d: Pushed
f8313885dabb: Pushed
787062cd94fb: Mounted from library/node
b27cb3611f76: Mounted from library/node
29fbe5747f9a: Mounted from library/node
533be996568a: Mounted from library/node
a4e797bc3f15: Mounted from library/node
392f356944ff: Mounted from library/node
15210a41d4ee: Mounted from library/node
e2a8a00a83b2: Mounted from library/node
latest: digest: sha256:88382b2cef5a656abed73f8bdb2d499f87cd5300e743e2f4949fd43e93b30da6 size: 3049
[Pipeline] }
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

PYTHON

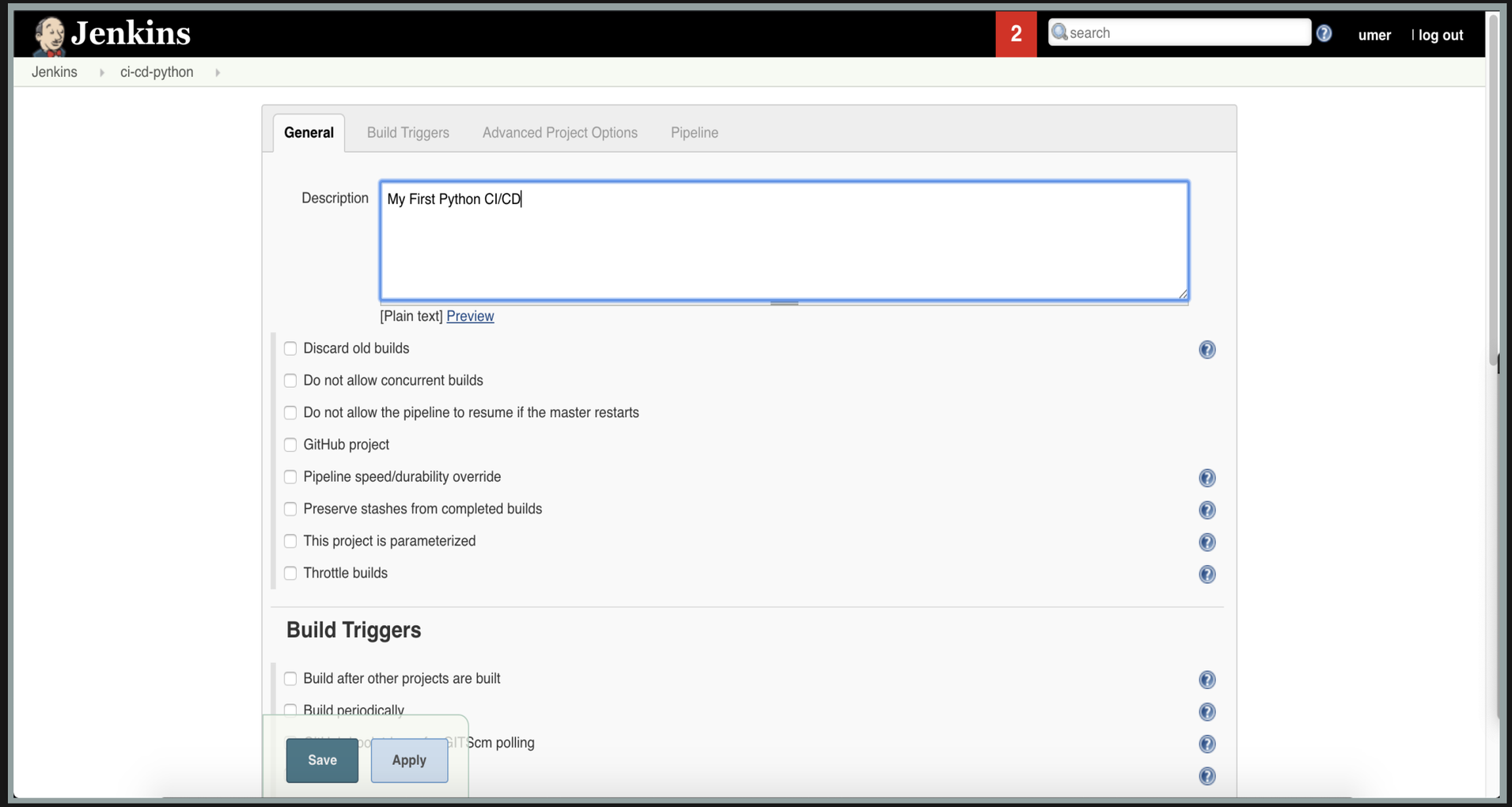
In this Lab, you are going to fork a repository, clone it, create a jenkins file, push to remote repository and then create CI/CD job. You will use the jenkins pipeline to compile python script and then create artifact. First Create a New Job



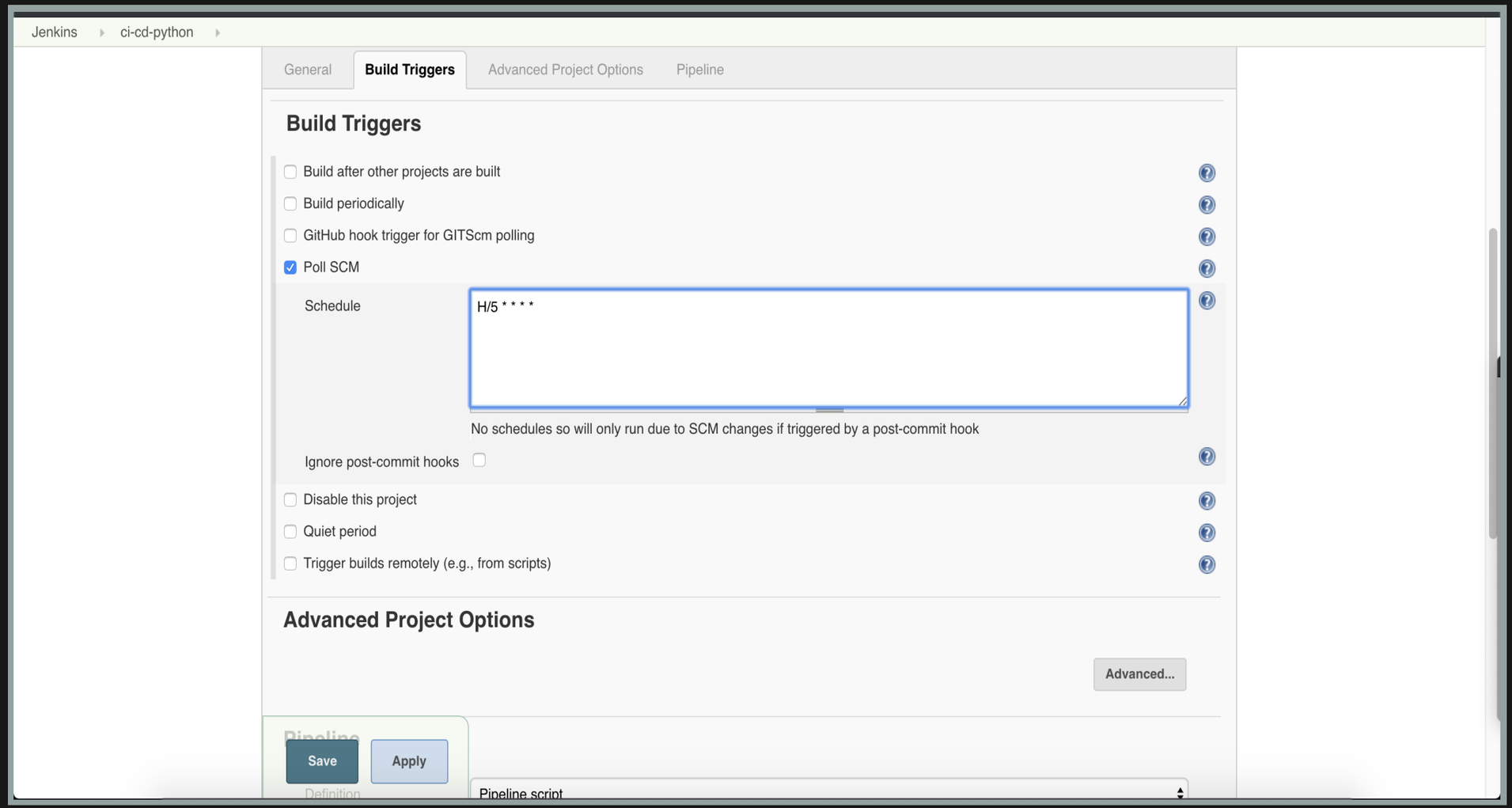
Enter job name, select Pipeline and Click OK as shown below



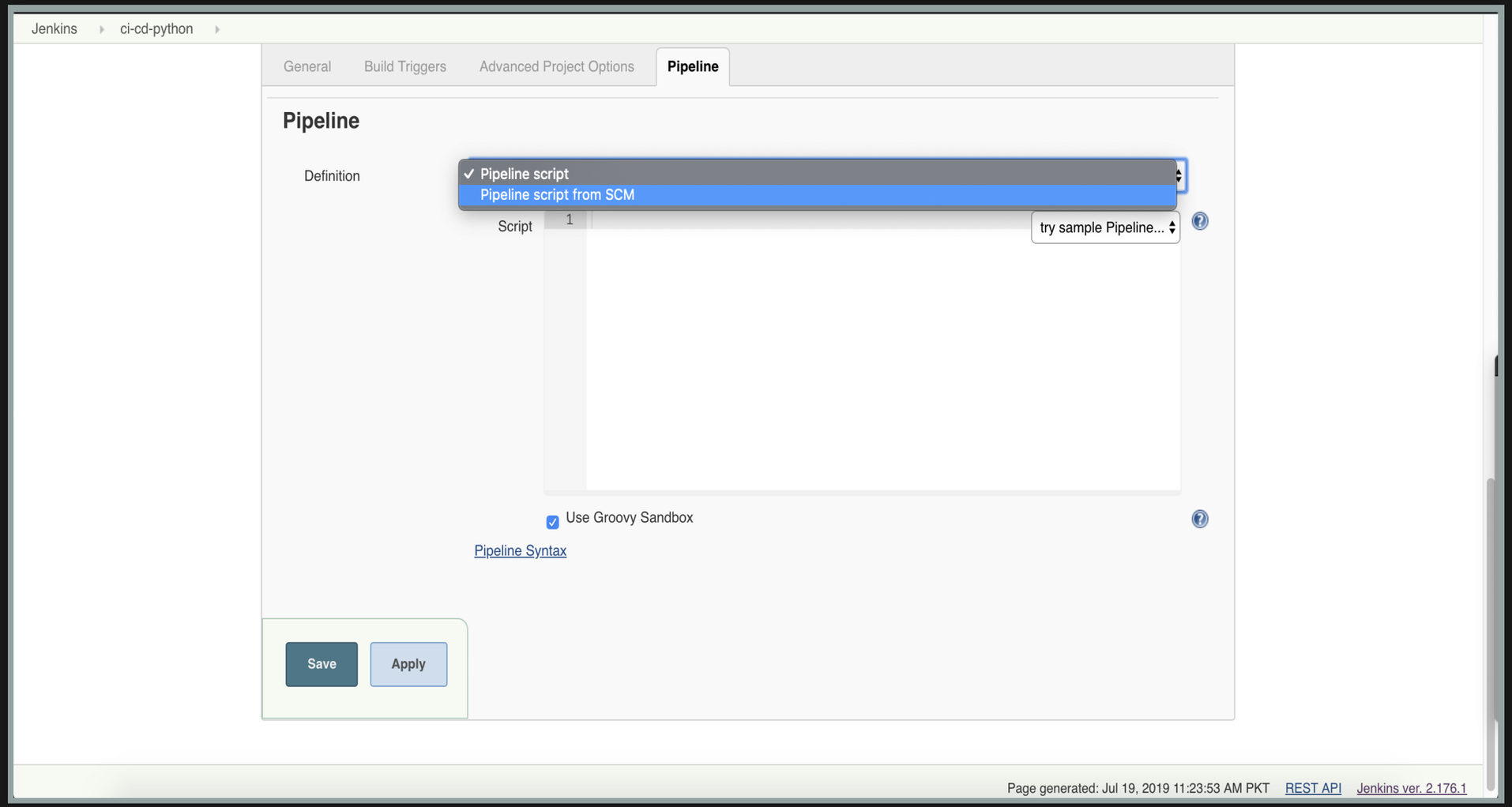
Enter job description and Click Build Triggers



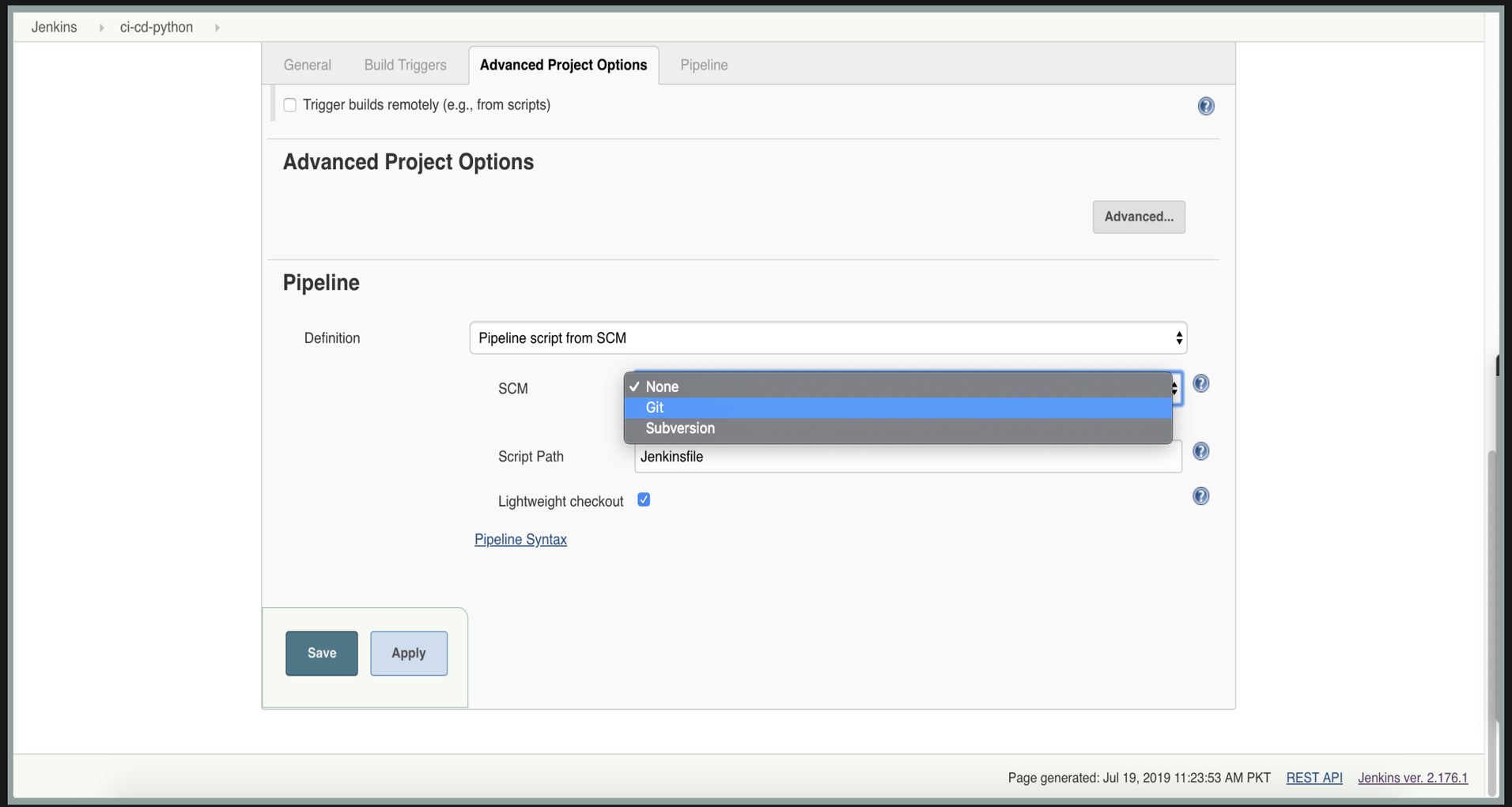
Select Poll SCM, specify schedule as shown below



Select 'Pipeline script from SCM'



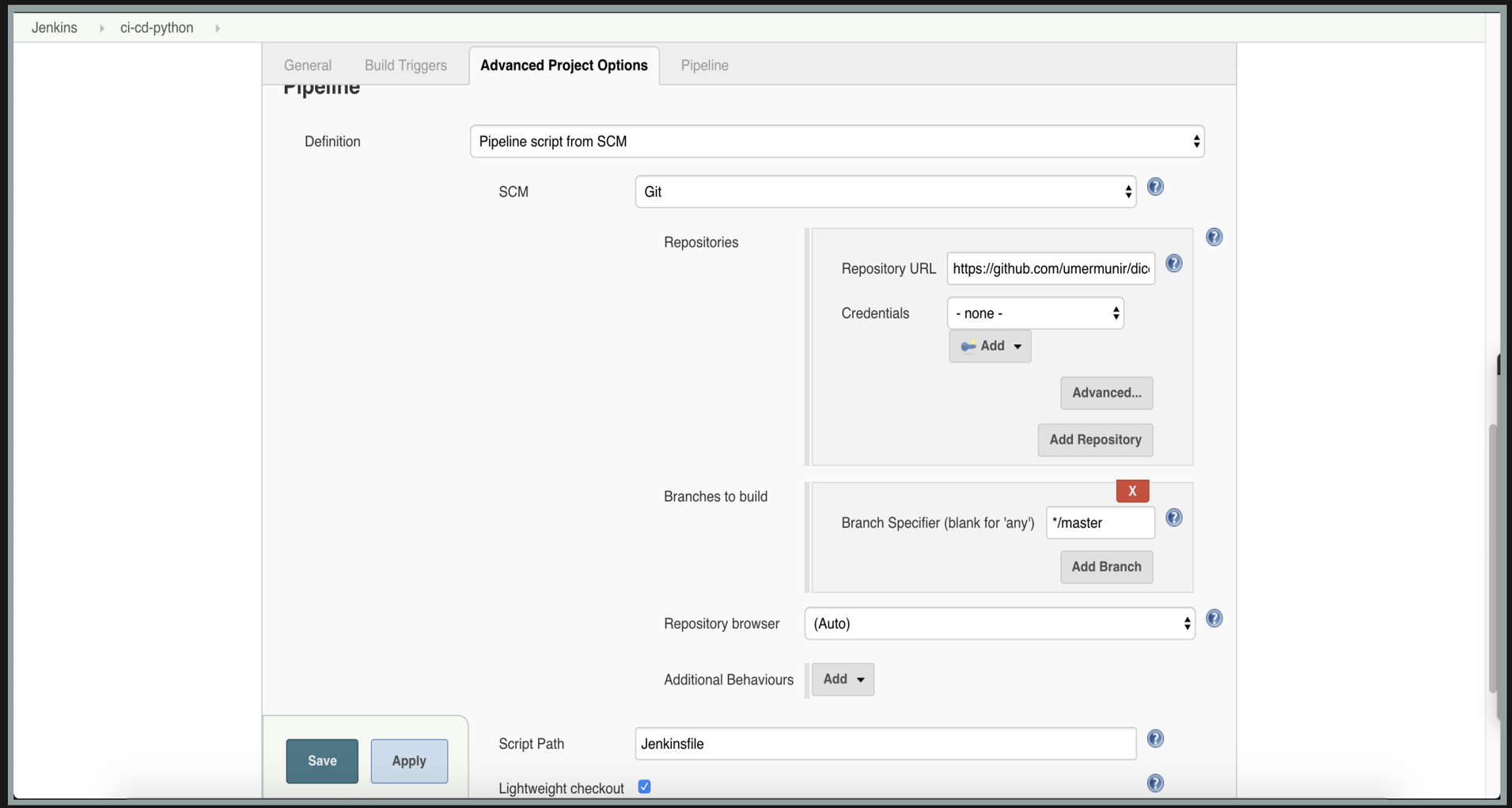
Select 'Git'


Jenkins
ci-cd-python
General
Build Triggers
Advanced Project Options
Pipeline
Trigger builds remotely (e.g., from scripts)
Advanced Project Options
Advanced...
Pipeline
Definition
Pipeline script from SCM
SCM
None
Git
Subversion
Script Path
Jenkinsfile
Lightweight checkout
Pipeline Syntax
Save
Apply
Page generated: Jul 19, 2019 11:23:53 AM PKT
REST API
Jenkins ver. 2.176.1

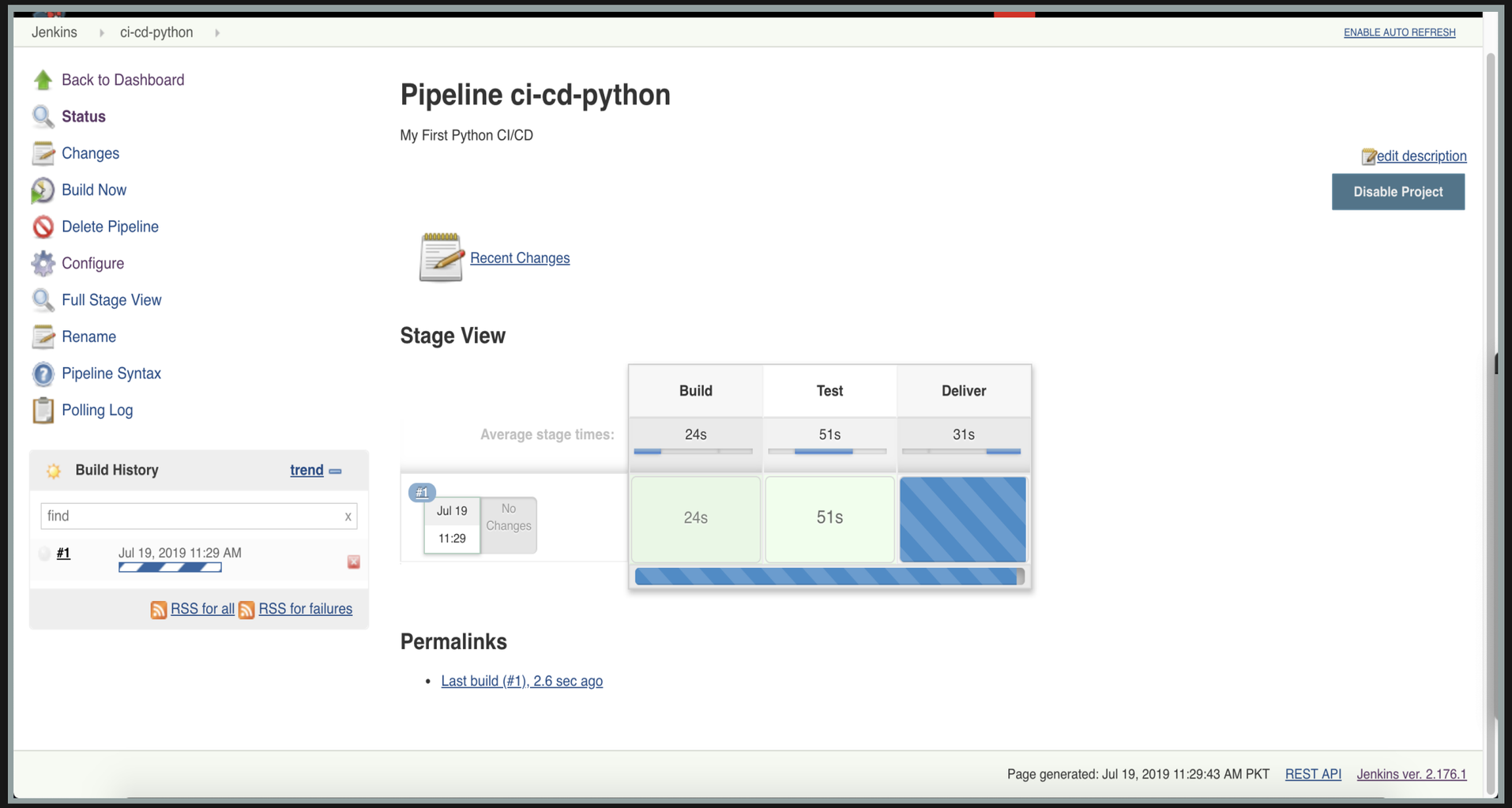
Fork DICE Python, clone it and then create a JenkinsFile below and push to repository

```
pipeline {
    agent none
    stages {
        stage('Build') {
            agent { docker { image 'python:2-alpine' } }
            steps {
                sh 'python -m py_compile sources/add2vals.py sources/calc.py'
            }
        }
        stage('Test') {
            agent { docker { image 'qnib/pytest' } }
            steps {
                sh 'py.test --verbose --junit-xml test-reports/results.xml sources/test_calc.py'
            }
        }
        stage('Deliver') {
            agent { docker { image 'cdrx/pyinstaller-linux:python2' } }
            steps {
                sh 'pyinstaller --onefile sources/add2vals.py'
            }
            post {
                success {
                    archiveArtifacts 'dist/add2vals'
                }
            }
        }
    }
}
```

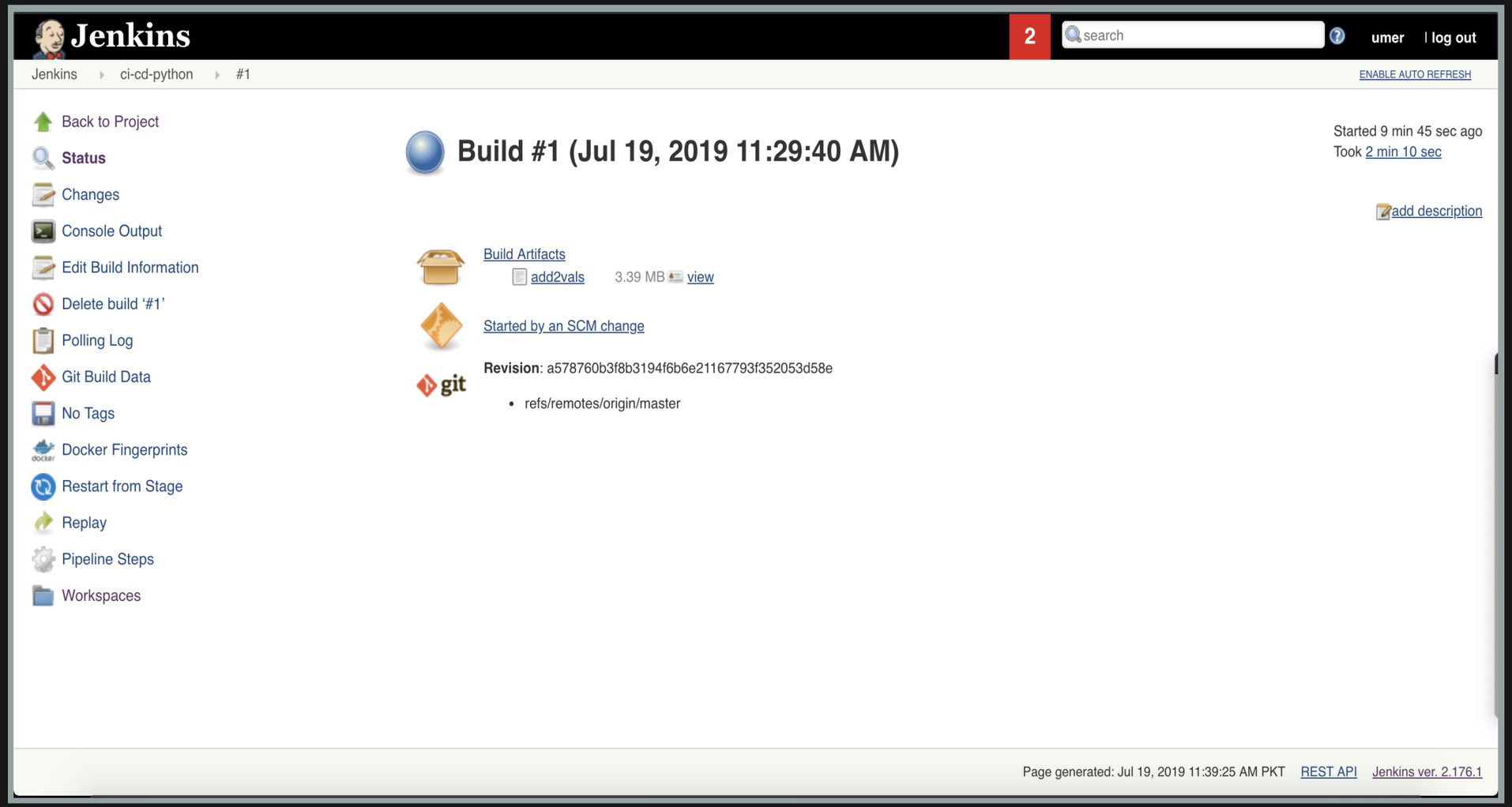
Enter your Repository URL and Click Save



Make a change in local repository and push to remote repository to trigger the job



Click the Build and you should see the details as shown below.

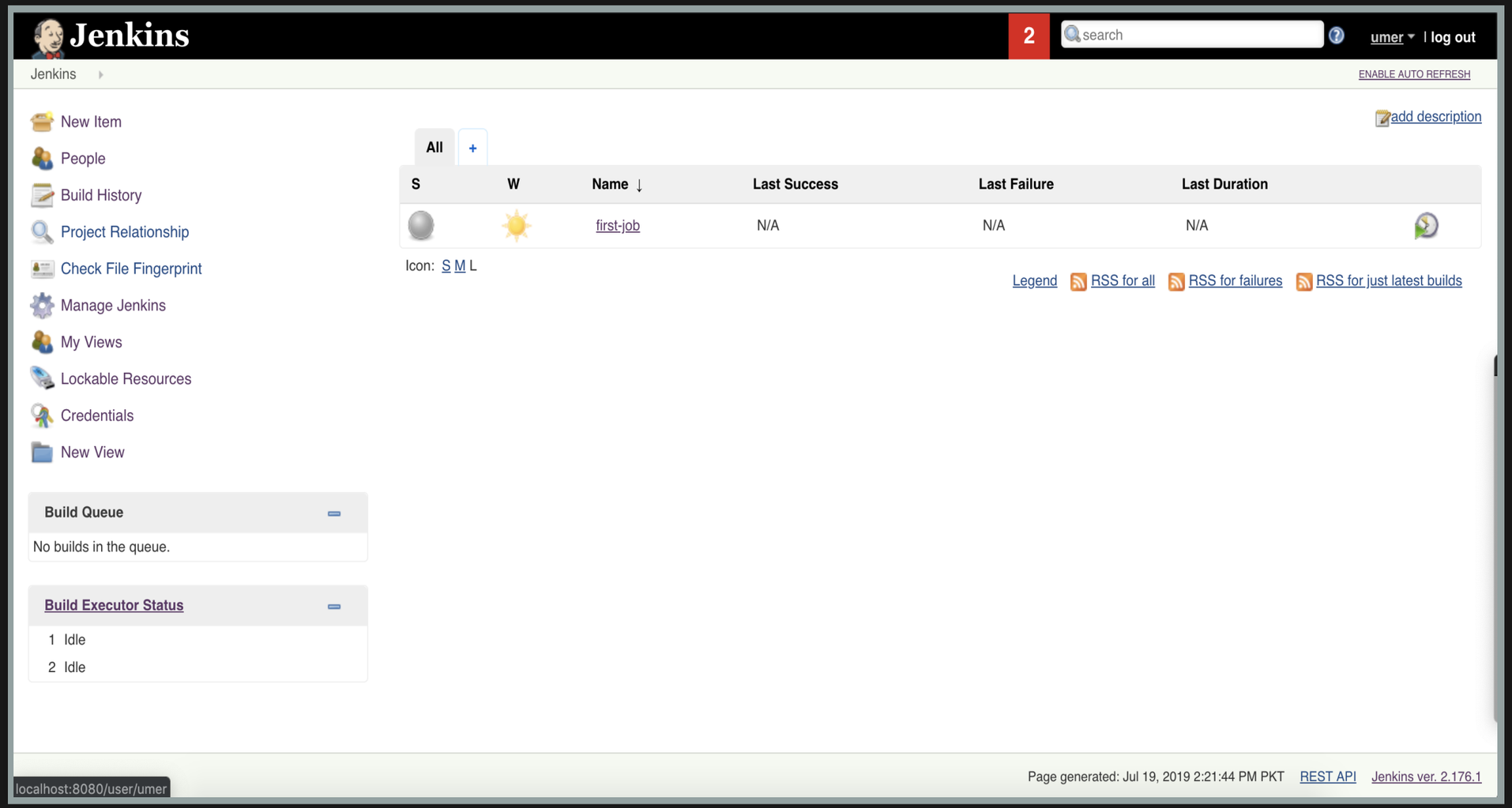


TRIGGER JOBS FROM BASH SCRIPT

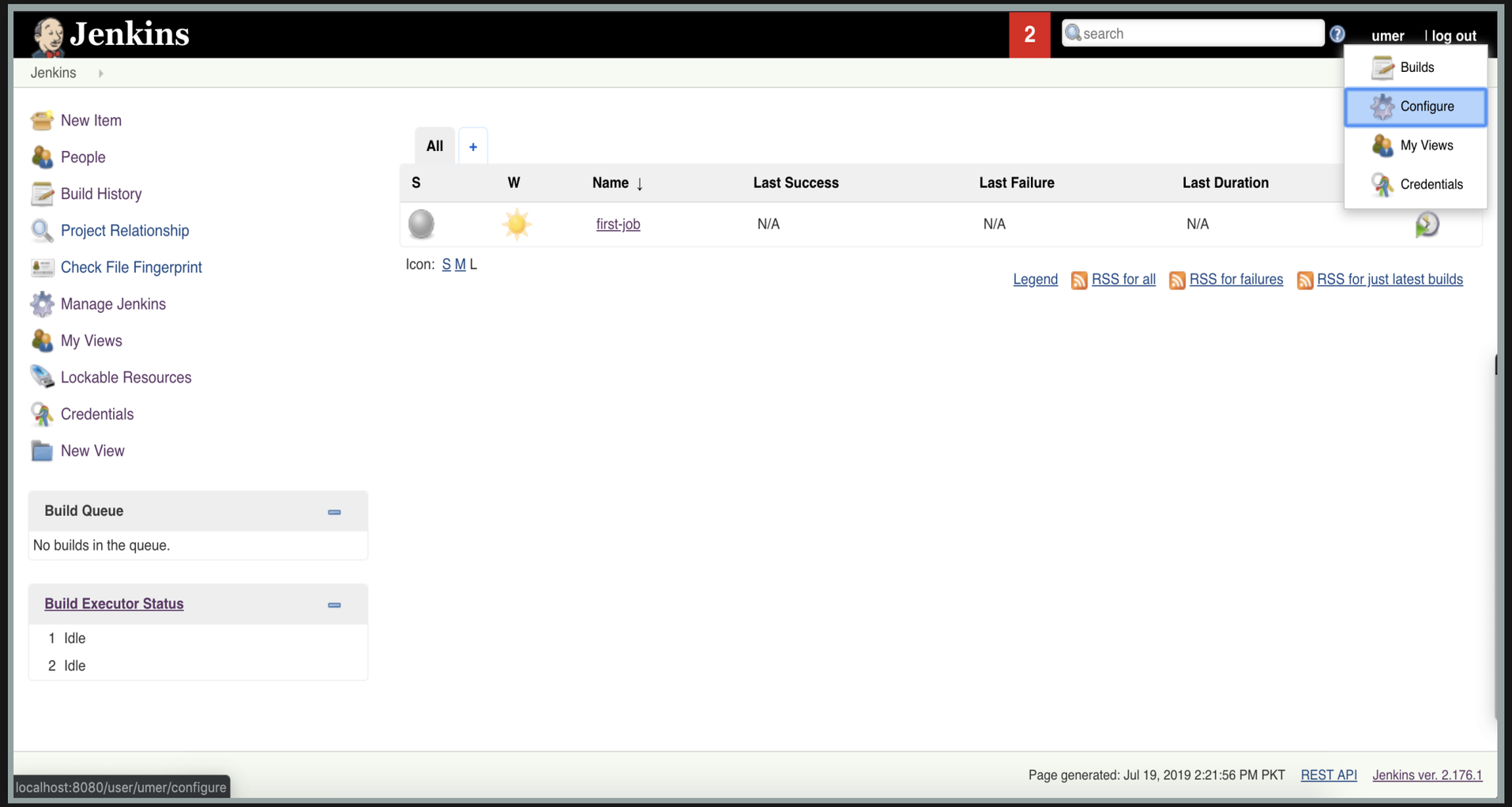
- REST API
- Jenkins CLI
- Trigger Job with Email

REST API

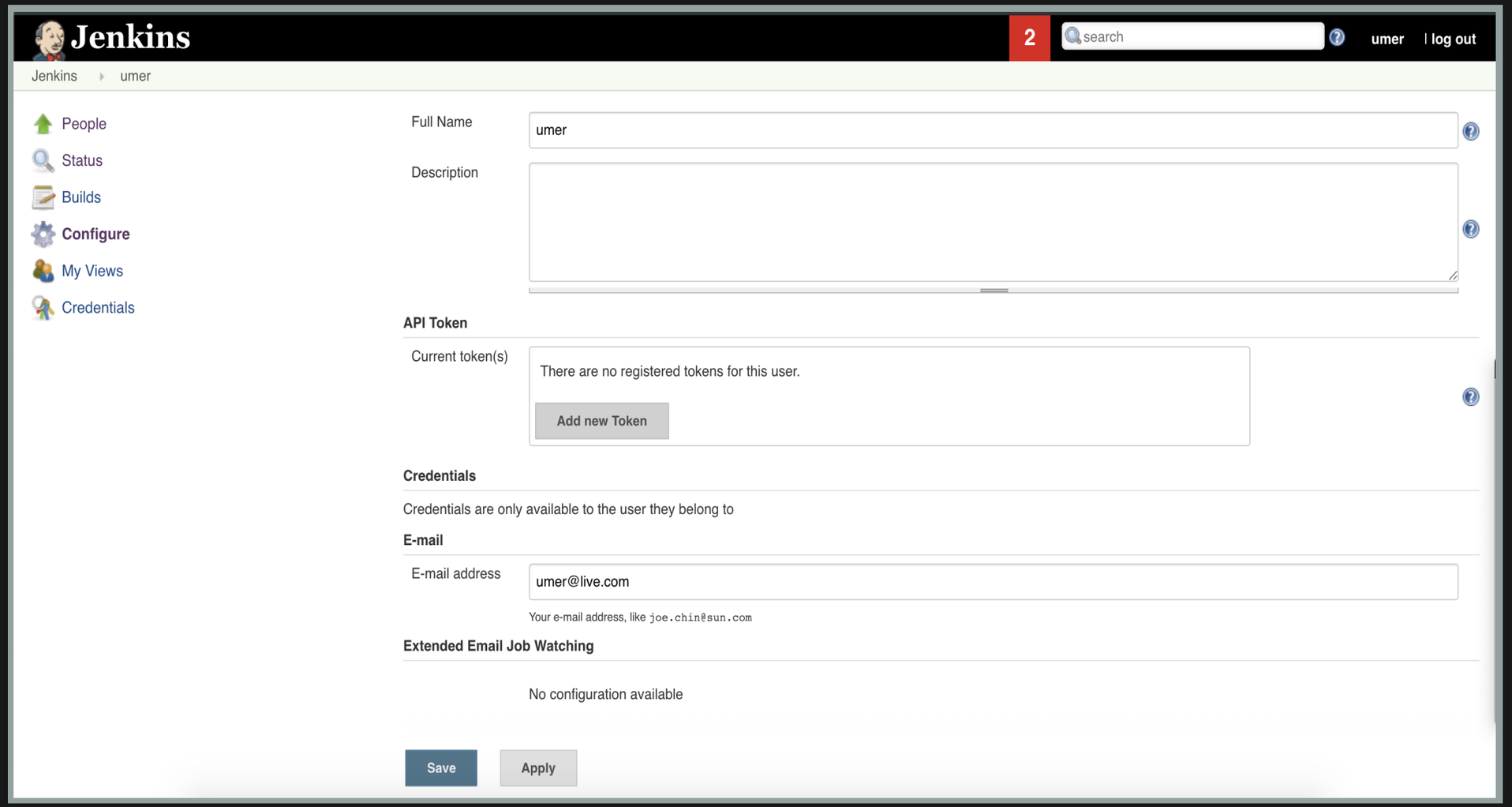
In this Lab, you are going to generate a token and use that to trigger job remotely using REST API. Go to the landing Page and Click the arrow next to your user



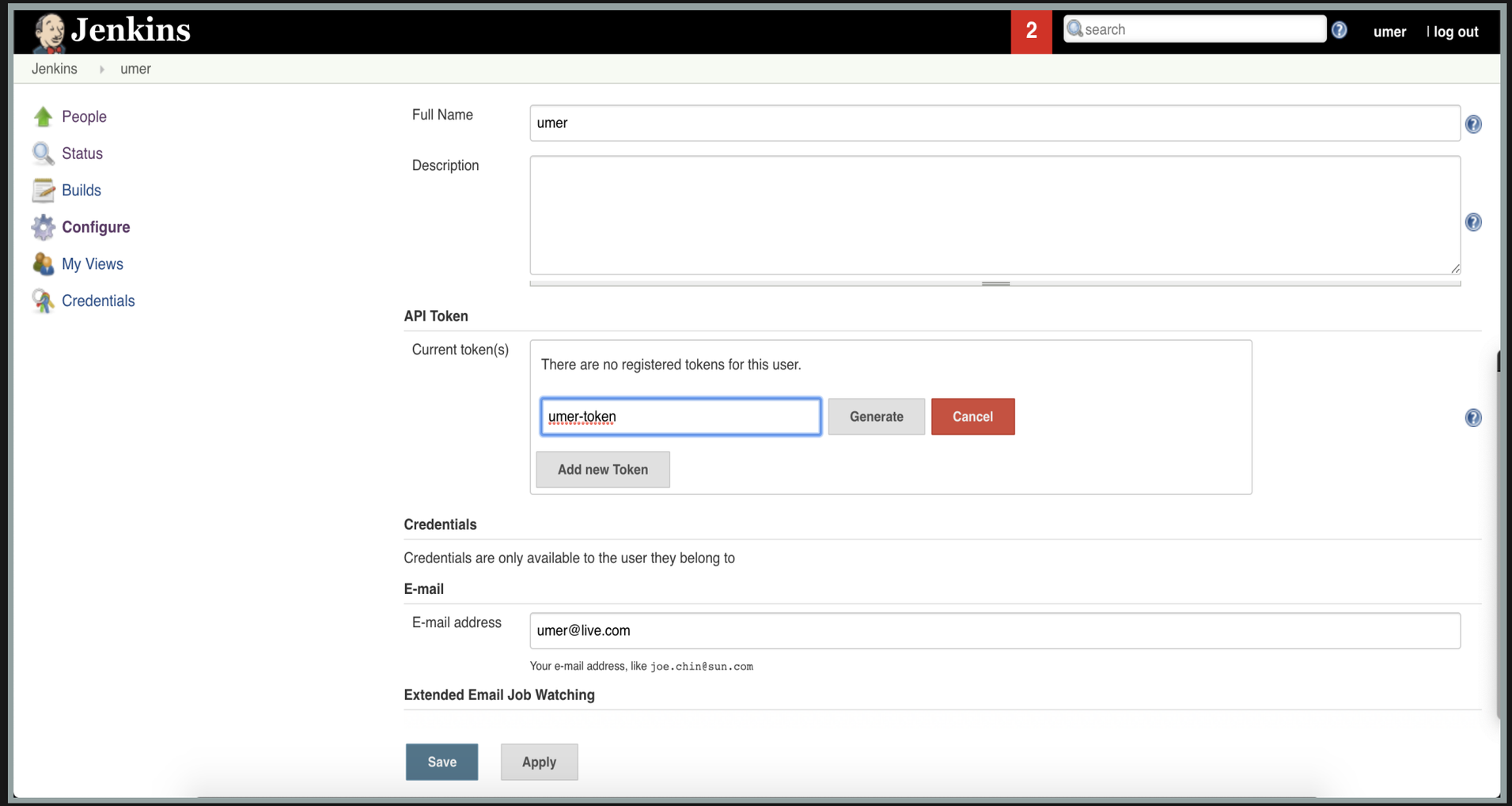
Select Configure



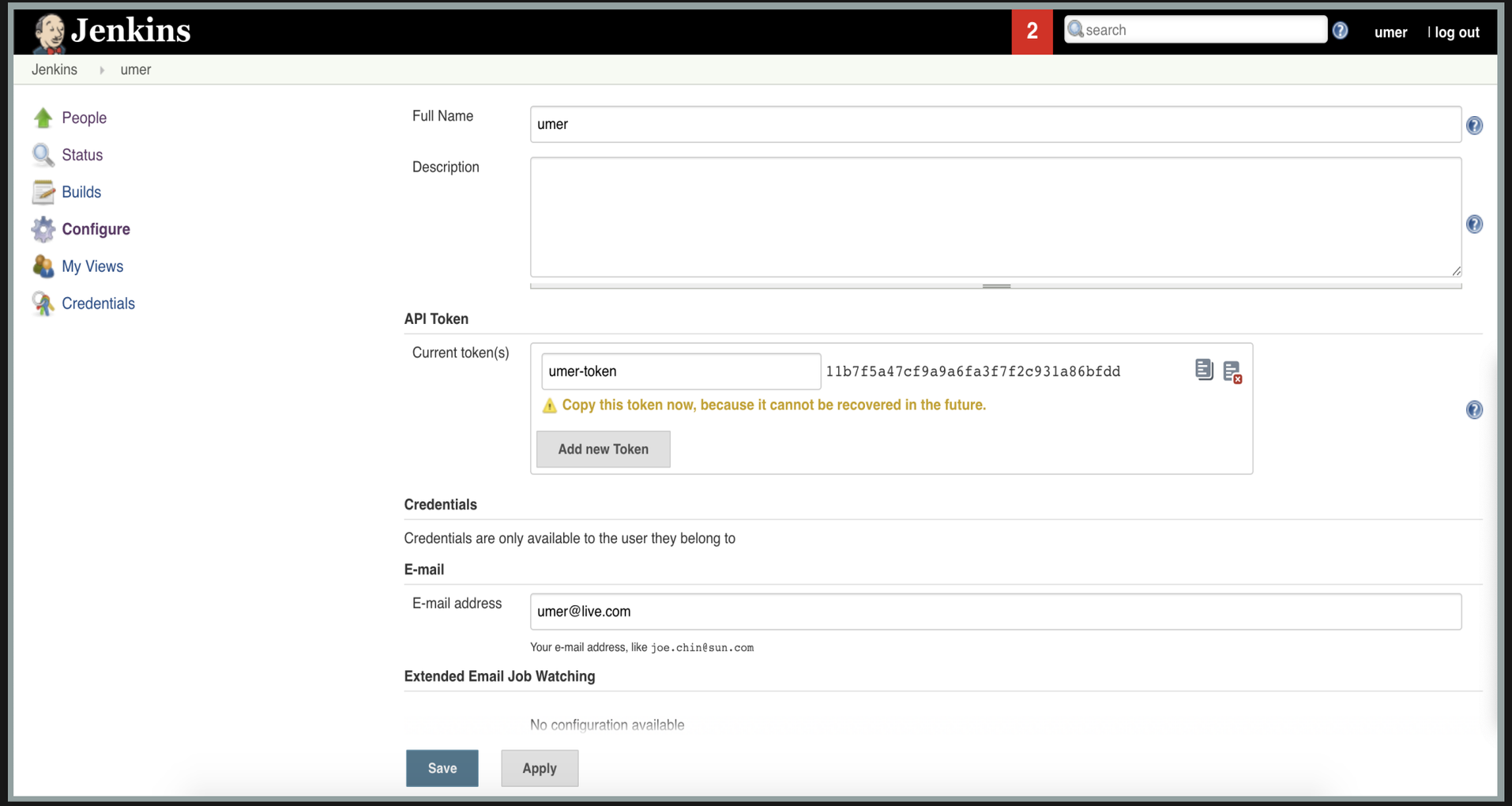
Click Add Token



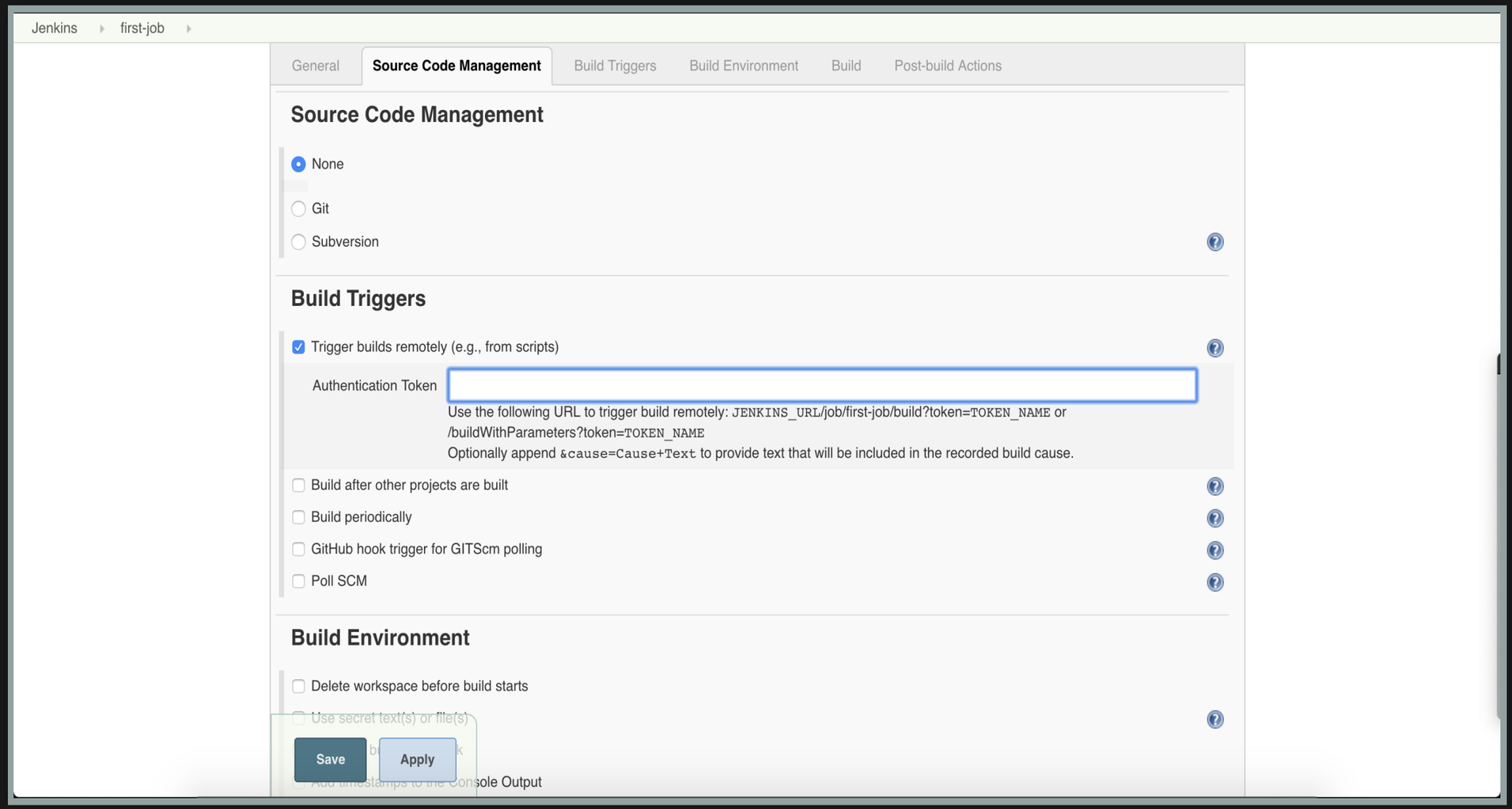
Specify name of the token



Copy token generated



Next go to the job you want to trigger and in the Build Triggers section, select 'Trigger builds remotely' and paste the copied token and Click Save.

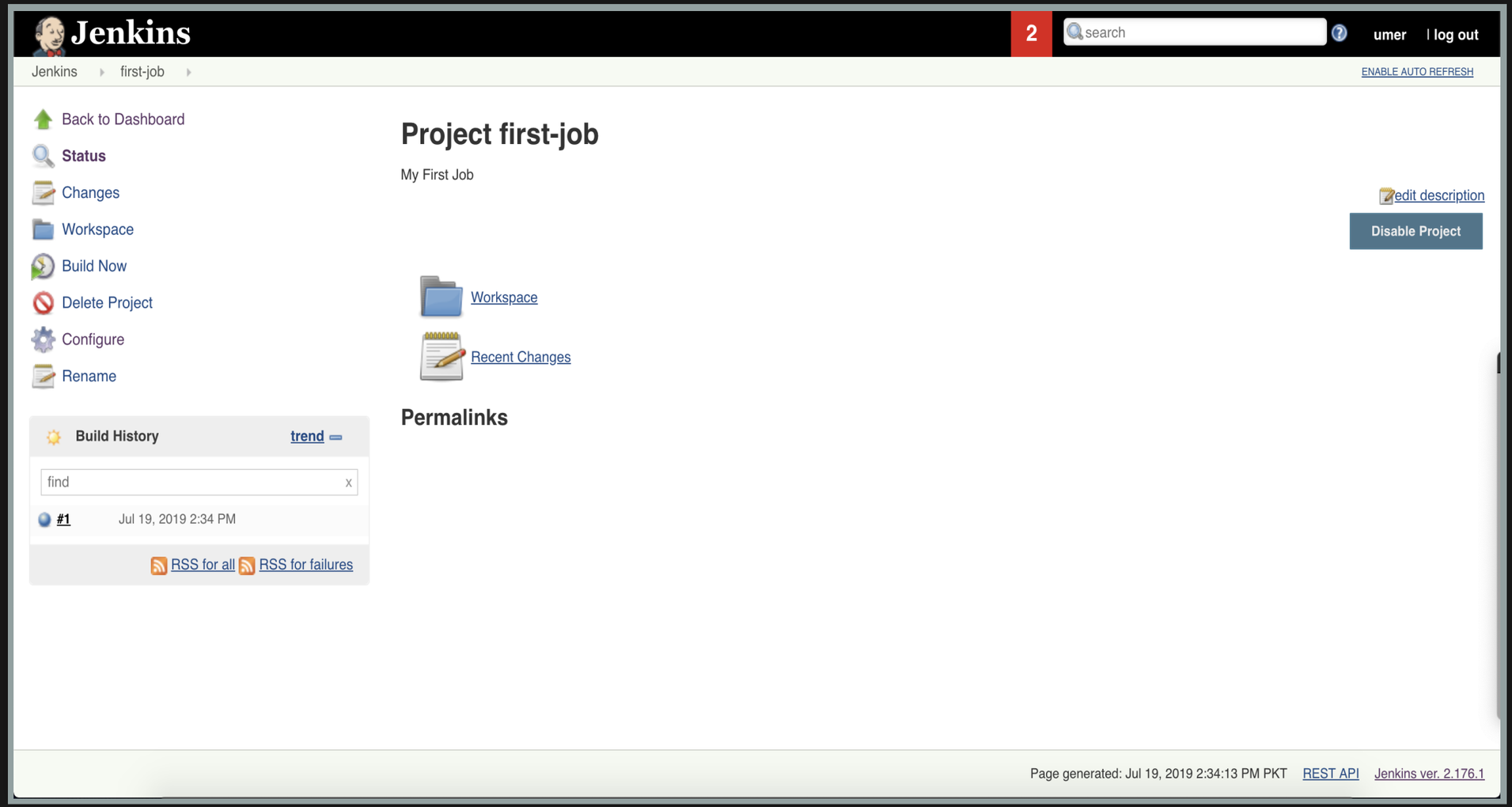


Run the following command from your terminal

```
curl -u username:token http://localhost:8080/job/first-job/build?token=token
```

where "username" is your Jenkins user, "token" is the token you generated earlier and first-job is the job you want to trigger

You should the triggered build as shown below



JENKINS CLI

First you are going to download jenkins-cli by running following command

```
wget http://localhost:8080/jnlpJars/jenkins-cli.jar
```

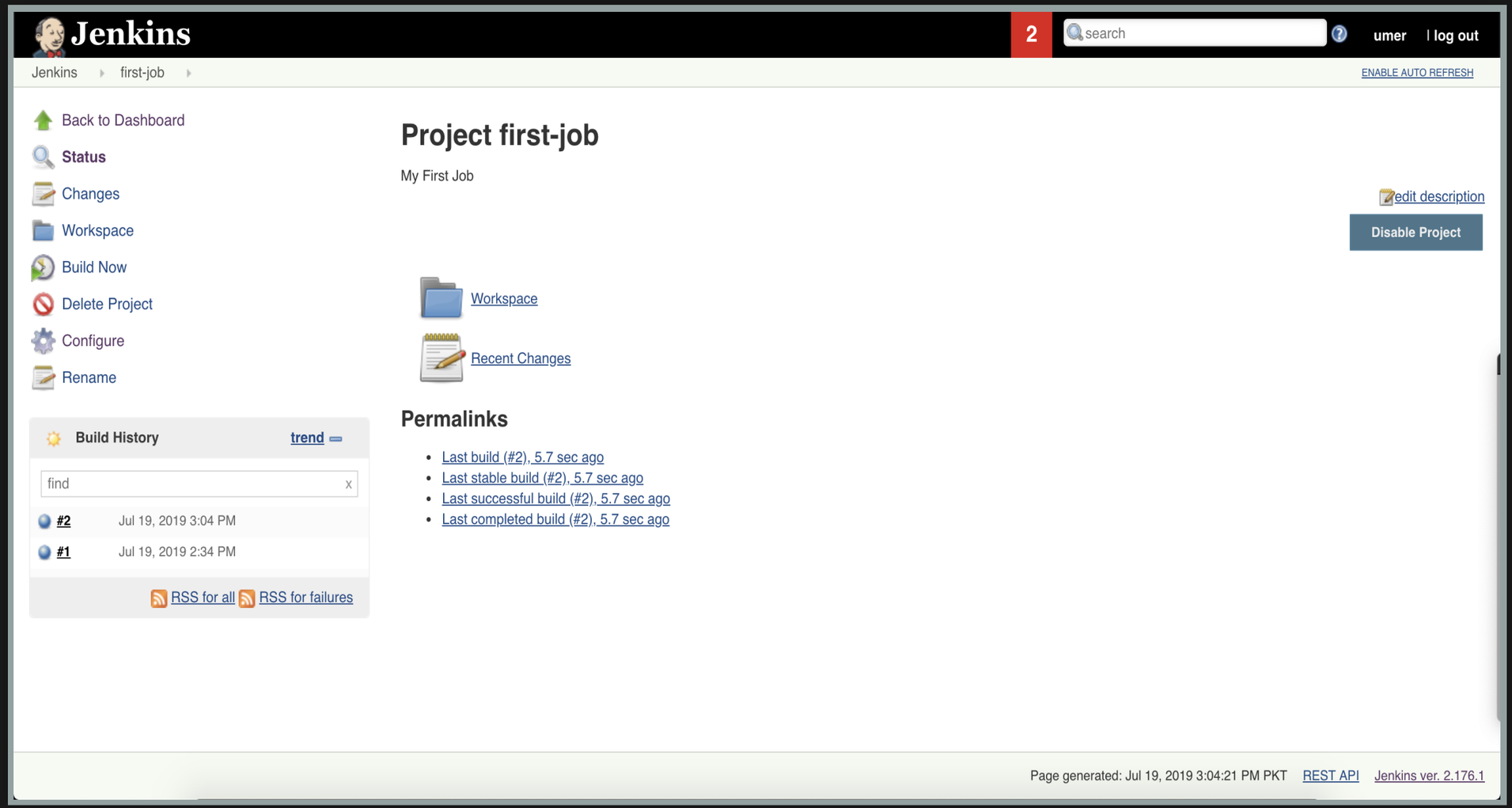
where localhost:8080 is the URL of Jenkins running on your system

Trigger your job by running the following command. Replace "username" with your user in Jenkins and "password" with your password.

```
java -jar jenkins-cli.jar -s "http://localhost:8080" -auth username:password build "first-job"
```

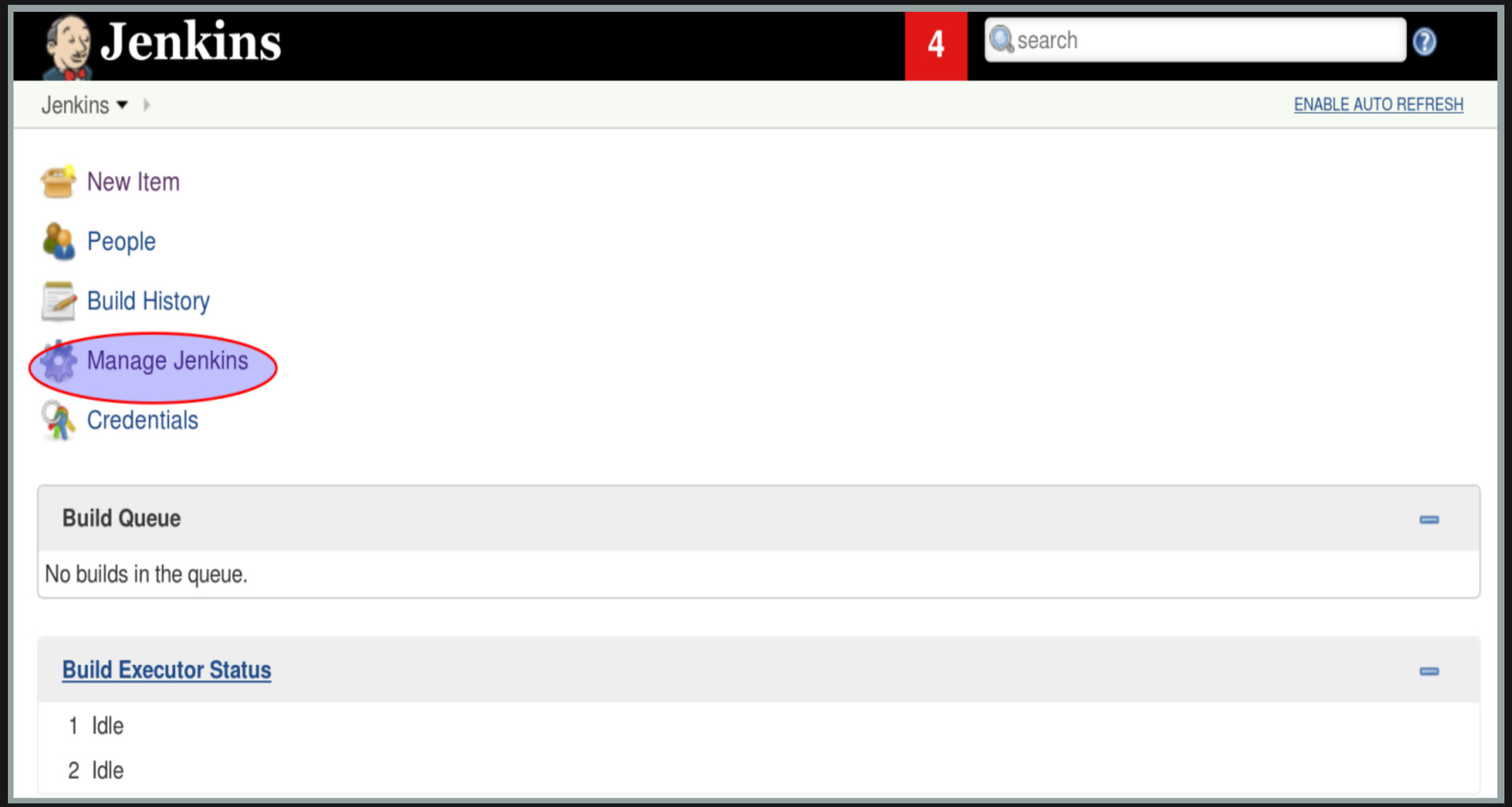
Where is localhost:8080 is the Jenkins URL and "first-job" is the job you want to trigger

You should see the job being triggered as shown below

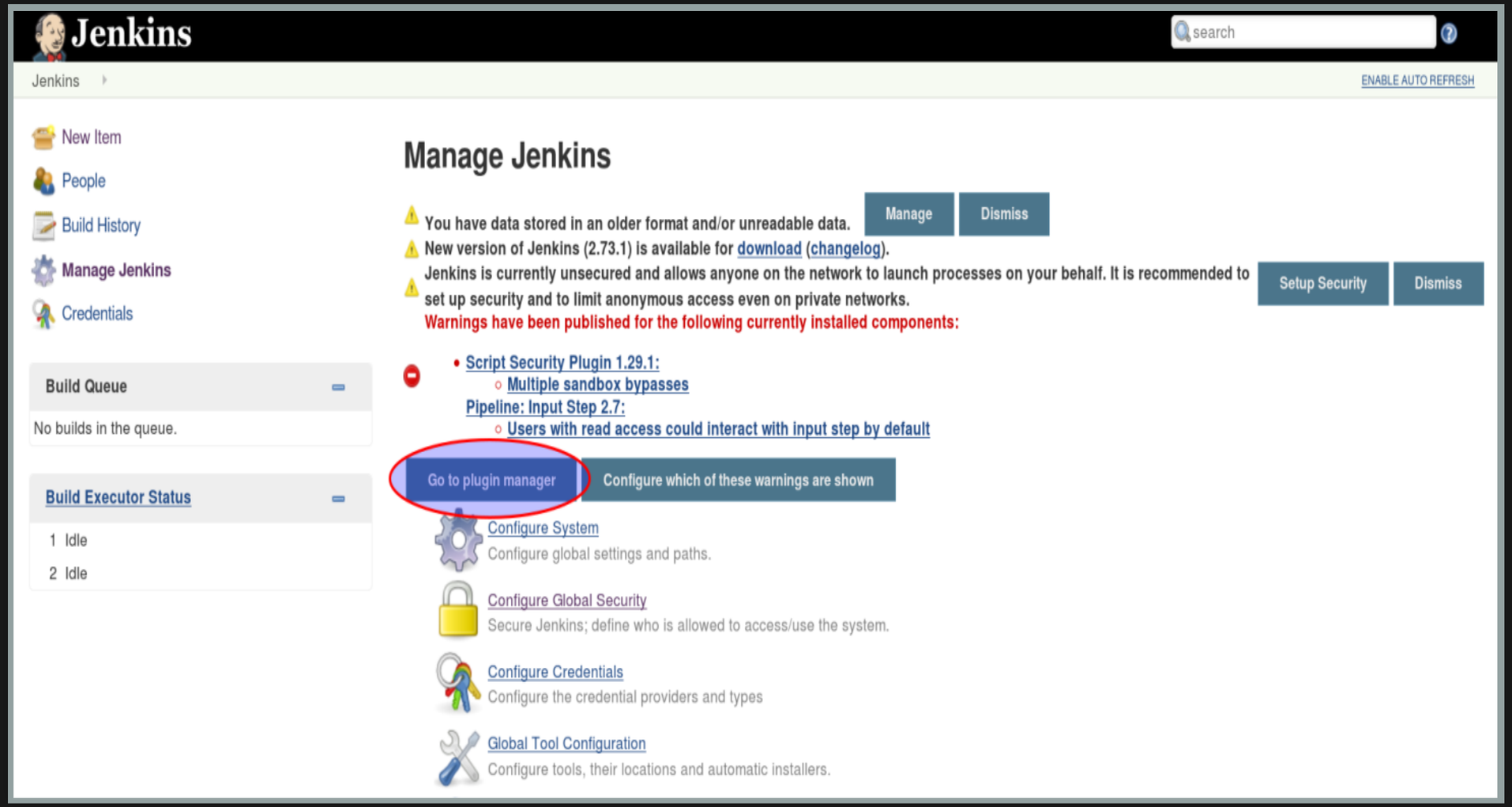


TRIGGER JOB WITH EMAIL

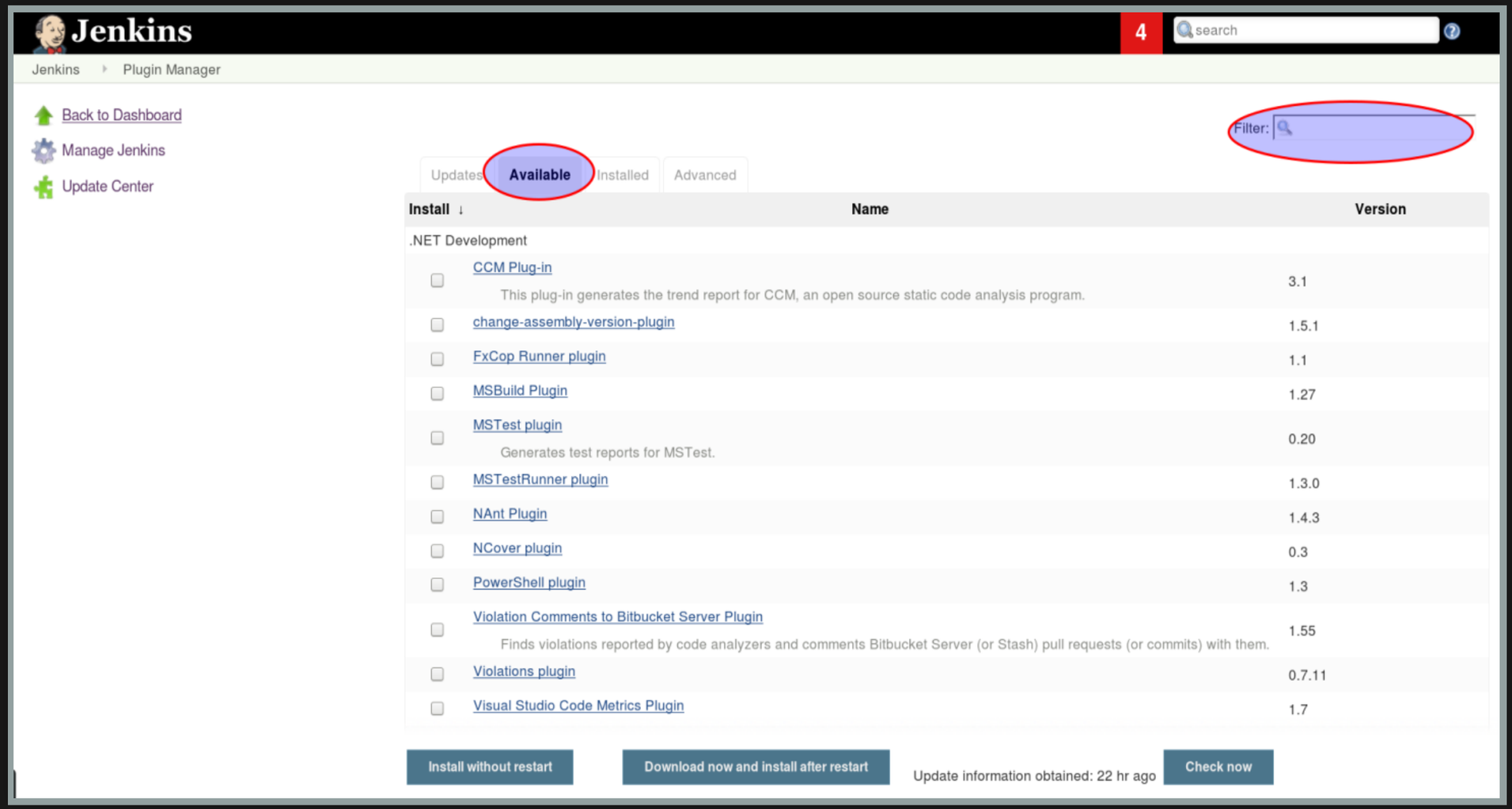
On your Jenkins home page, click Manage Jenkins.



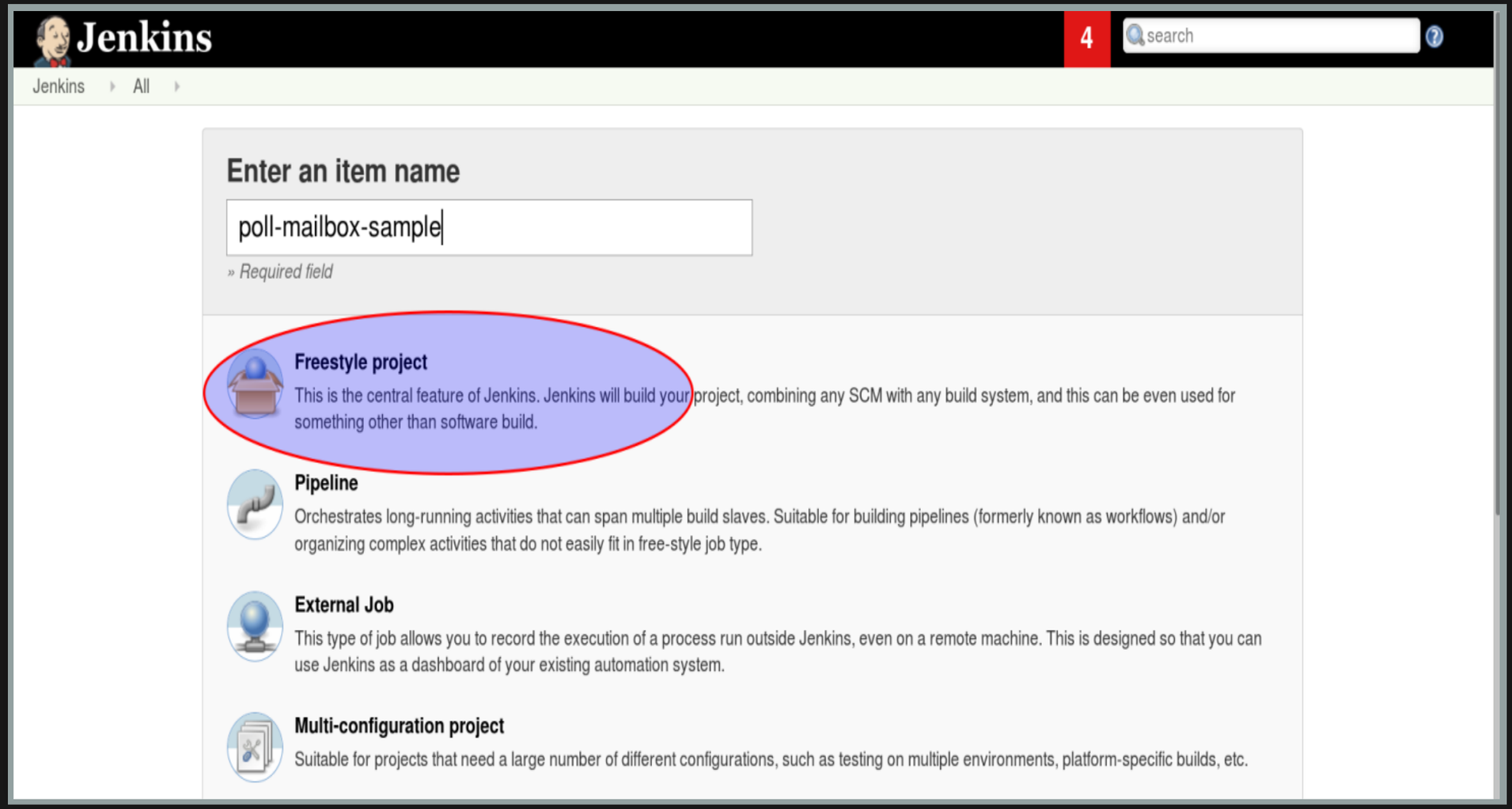
Click on Go to plugin manager button



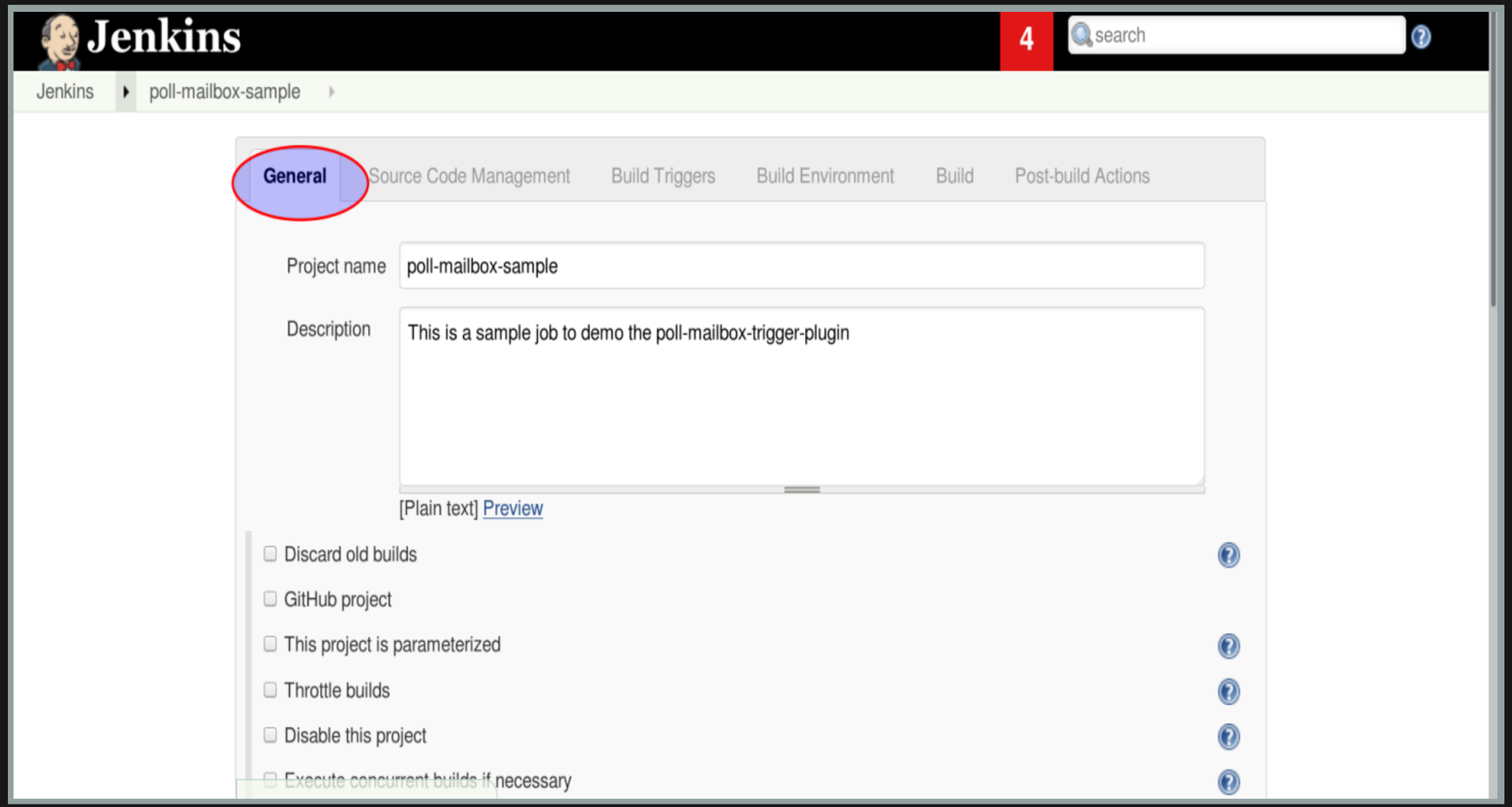
Select the Available tag and using the Filter search box, search for Poll Mailbox Trigger Plugin, select it and choose Install without restart.



On the Jenkins homepage, select New item, and then create a new freestyle project as shown below

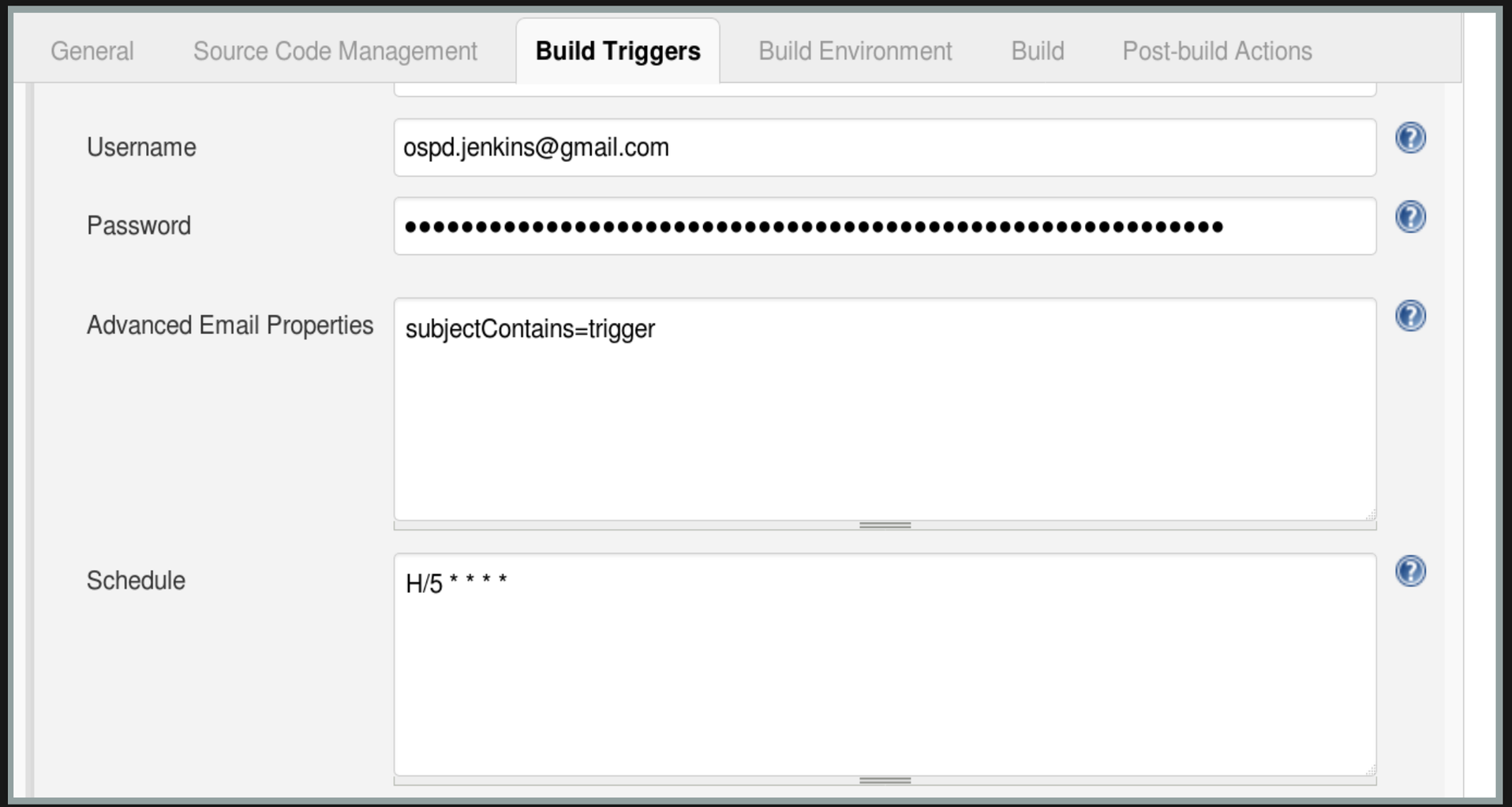


Enter description and Click Build Triggers

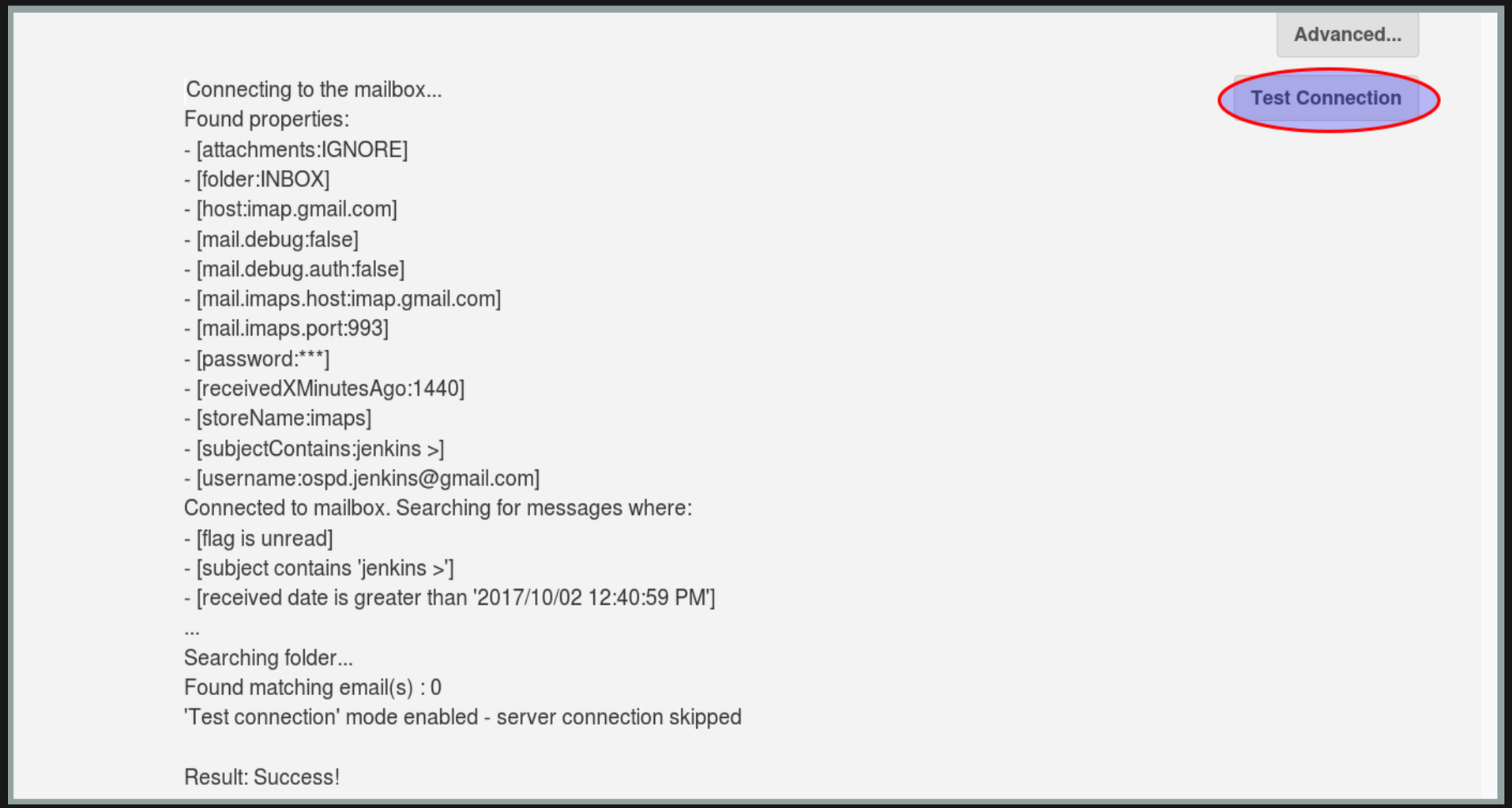


Select Poll Mailbox Trigger. If you want to use a GMAIL account, enter GMAIL email address in the username and the password as shown below. We need some advanced configuration as well so click on that

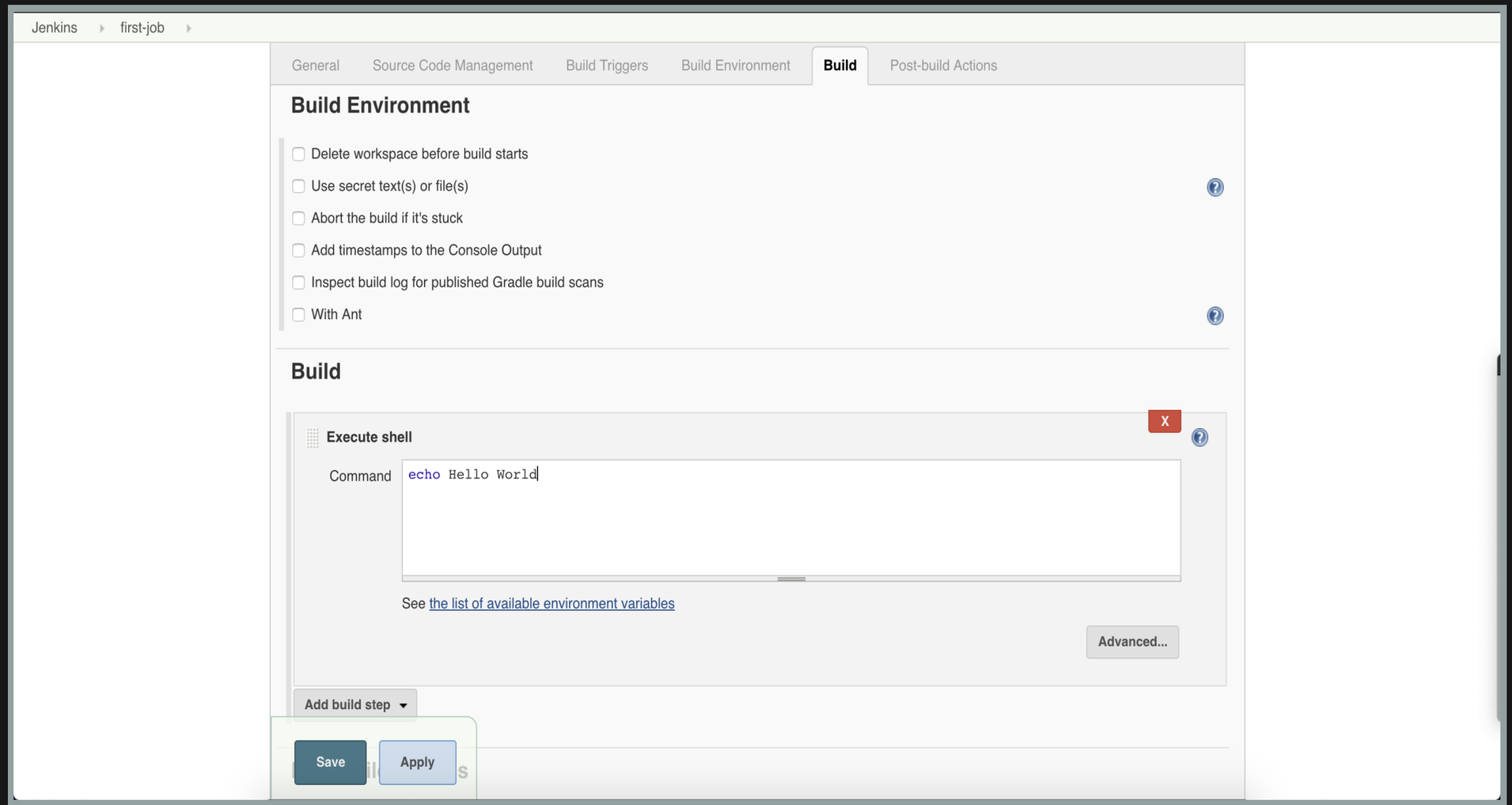
We can use the subjectContains property to set the string in the subject line that the would need to be present for the job to be triggered. We can also configure the interval at which the plugin scans the inbox for unread messages containing the subjectContains property in the subject line.



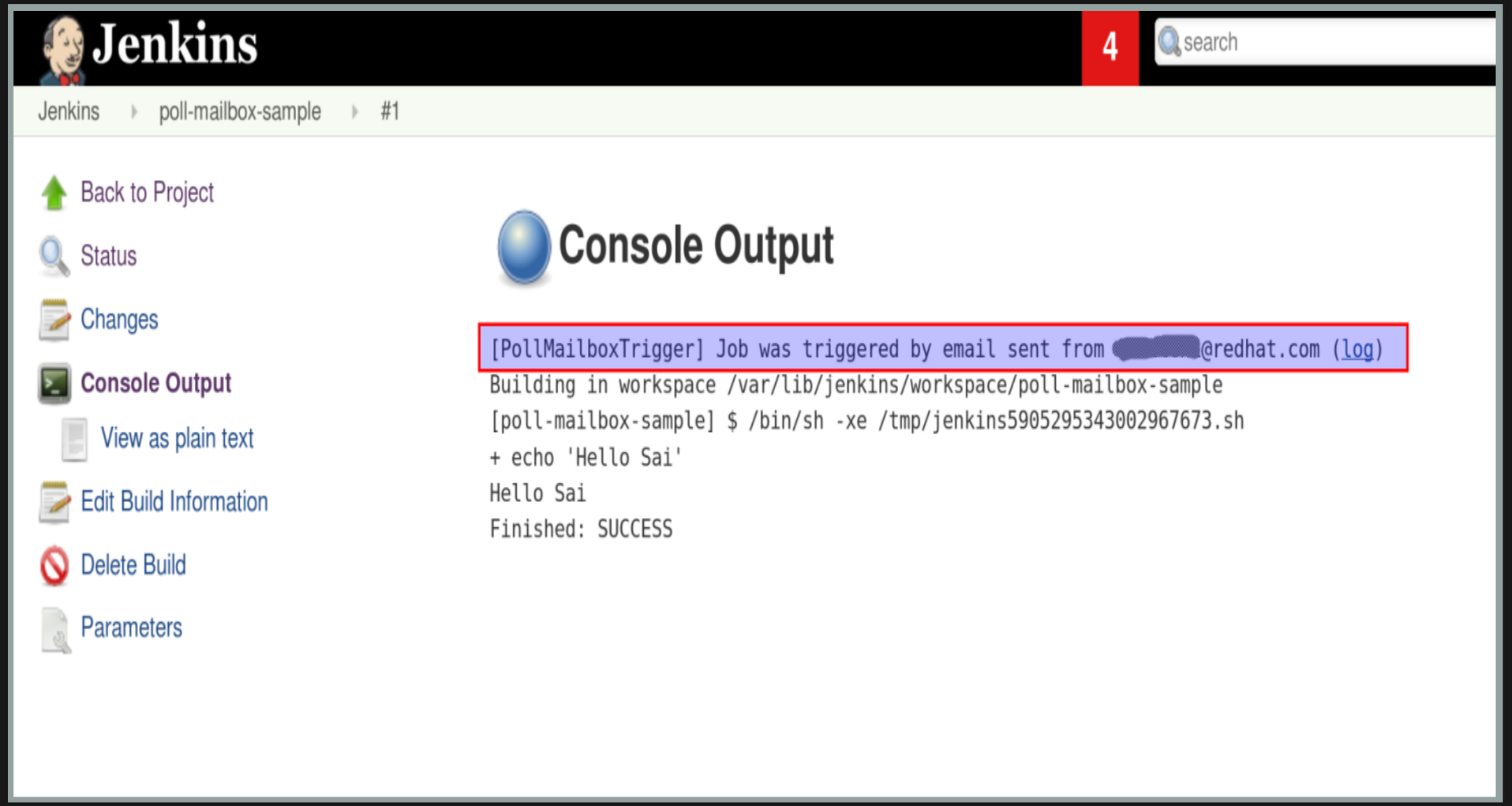
After configuring everything, hit on the Test Connection button to verify



Click on Build section and the command as shown below



Send an email with the subject line "trigger", this should trigger your job



JENKINS SECURITY

- Keep Jenkins shielded from the internet using firewall rules
- You will need to whitelist github IP for push requests
- Keep Jenkins up-to-date
  - Use LTS (Long Term Support) edition
- Always keep the plugins up to date
- Configure authentication/authorization
  - Administator can decrypt credentials
- In larger systems, don't build on the master
- Limit project names to a sane (e.g. alphanumeric) character set
- Avoid scheduling all jobs to start at the same time
- Archive unused jobs before removing them