**Lab 6 – Manage Continuous Integration and Delivery with Jenkins Pipeline**

Contents

[Lab Evidence 2](#_Toc96761166)

[Learning Goals 2](#_Toc96761167)

[Pre-requisite 2](#_Toc96761168)

[Introduction 2](#_Toc96761169)

[Demonstrate Steps for managing Continuous Integration and Delivery with Jenkins Pipeline 4](#_Toc96761170)

[**Step 1: Terraform Structure** 5](#_Toc96761171)

[**Step 2: Terraform Code** 5](#_Toc96761172)

[**Step 3: Ansible Playbook** 5](#_Toc96761173)

[**Step 4: Terraform and Ansible in action** 5](#_Toc96761174)

[**Step 5: Connect to Jenkins** 6](#_Toc96761175)

[**Step 6: Fork GitHub Repository** 6](#_Toc96761176)

[**Step 7: Generate GitHub Access Token** 7](#_Toc96761177)

[**Step 8: Create DockerHub Account** 9](#_Toc96761178)

[**Step 9: Create Gradle Build** 10](#_Toc96761179)

[**Step 10: Create Docker File** 11](#_Toc96761180)

[**Step 11: Create Jenkins File** 12](#_Toc96761181)

[**Step 12: Install Jenkins Plugins** 14](#_Toc96761182)

[**Step 13: Configure the generated GitHub Access Key in Jenkins** 16](#_Toc96761183)

[**Step 14: Configure GitHub and DockerHub in Jenkins Credentials** 18](#_Toc96761184)

[**Step 15: Create a Multi-branch pipeline project in Jenkins** 21](#_Toc96761185)

[**Step 16: Terraform destroy** 26](#_Toc96761186)

[Note 26](#_Toc96761187)

[Conclusion 26](#_Toc96761188)

# Lab Evidence

Your report on this lab will be the basis for your grade. The purpose of the report is to demonstrate to your instructor that you have completed the lab successfully and understood the material. Please include the following screenshots as part of your evidence.

* Stage view of your pipeline after it has completed Check HTTP Response
* Stage logs for 'Check HTTP response'
* Output from docker ps command
* Web browser showing address bar with port 1233 loaded
* Your Docker Hub account, with pushed and pulled image
* After you click 'Proceed' button, stage logs for 'DeployToProduction'
* Output from docker ps command
* Web browser showing address bar with port 1233 loaded
* Output from 'Console Output'

A portion of the grade will be allocated to the quality of the report. The report should be well formatted, concise and to the point.

# Learning Goals

* Demonstrate the creation and management of continuous integration and delivery with Jenkins’s Pipeline.

# Pre-requisite

* Created AWS free tier accounts account (Check provided AWS instructions in Moodle - 02.Create AWS Free Tier Account.docx)
* Completed the following in Lab2: Manage AWS Infrastructure with Terraform
  + Downloaded and installed Terraform on your local machine link
  + Created AWS free tier accounts account<https://github.com/>
  + Downloaded and installed AWS CLI on your local machine link
  + Configured AWS Access Keys
* Completed the following in Lab3: Manage AWS Infrastructure with Terraform (IaC) and Ansible (CM)
  + Generated SSH Keys
  + Downloaded PuTTY
  + Downloaded PuTTYgen and Generated .ppk key
  + Connected to EC2 instance using PuTTY <https://github.com/>

# Introduction

You study during this week how Continuous Integration and Delivery works and how you can use Jenkins Pipeline.

* We will be utilizing what you have learned in the previous weeks and using Terraform to provision the infrastructure, Ansible Playbook to install and configure both Jenkins and Docker.
* Utilize GitHub, Docker, DockerHub, Gradle to create a multi-branch pipeline in Jenkins for React Web Application Continuous Integration and Delivery.
* The generate keys from lab 3. If you lost it, you could always return to the lab 3 instructions and generate it again.

# Demonstrate Steps for managing Continuous Integration and Delivery with Jenkins Pipeline

To accomplish this, you are going to follow these steps:

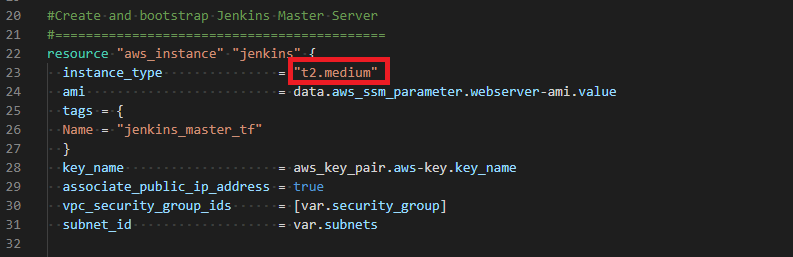
* Step 1: Terraform Structure
* Step 2: Terraform Code
* Step 3: Ansible Playbook
* Step 4: Terraform and Ansible in action
* Step 5: Connect to Jenkins
* Step 6: Fork GitHub Repository
* Step 7: Generate GitHub Access Token
* Step 8: Create DockerHub Account
* Step 9: Create Gradle Build
* Step 10: Create Docker File
* Step 11: Create Jenkins File
* Step 12: Install Jenkins Plugins
* Step 13: Configure the generated GitHub Access Key in Jenkins
* Step 14: Configure GitHub and DockerHub in Jenkins Credentials
* Step 15: Create a Multi-branch pipeline project in Jenkins
* Step 16: Terraform destroy

## **Step 1: Terraform Structure**

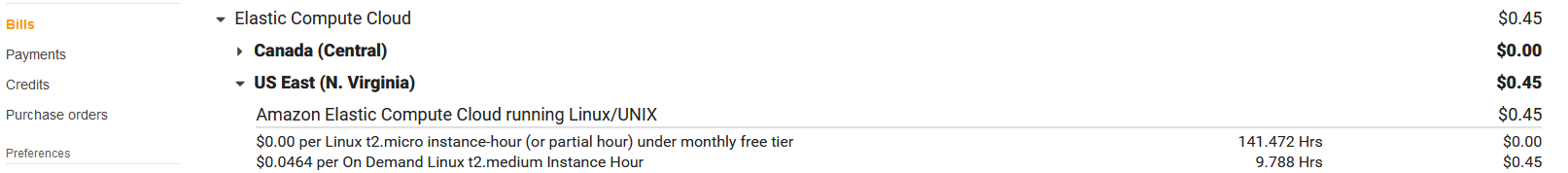
* + Refer to Lab 5 instructions from the previous module (Week5-Continuous Integration)

## **Step 2: Terraform Code**

* + Refer to Lab 5 instructions from the previous module (Week5-Continuous Integration)
  + One Change
  + This lab will need more CPU and Memory because of the JVM for Jenkins and Gradle. If you continue using t2.micro the server might stop responding and you will not be able to complete this lab and execute the CI/CD pipeline.
  + t2.medium instance doesn’t fall under the AWS Free Tier



* + At the time of writing this lab instruction, the price of t2.medium on-demand per hour around USD $0.0464; for more detail about the cost, you can refer to this [link](https://aws.amazon.com/ec2/instance-types/t2/)
  + As you can see in the screenshot from one of my AWS accounts under Billing. Around 10 hours of t2.medium compute cost me less than a dollar USD $0.45



* + The most important thing after you completes the lab, whether it was successful or not to run terraform destroy as you did in all other labs.

## **Step 3: Ansible Playbook**

* + Refer to Lab 5 instructions from the previous module (Week5-Continuous Integration)

## **Step 4: Terraform and Ansible in action**

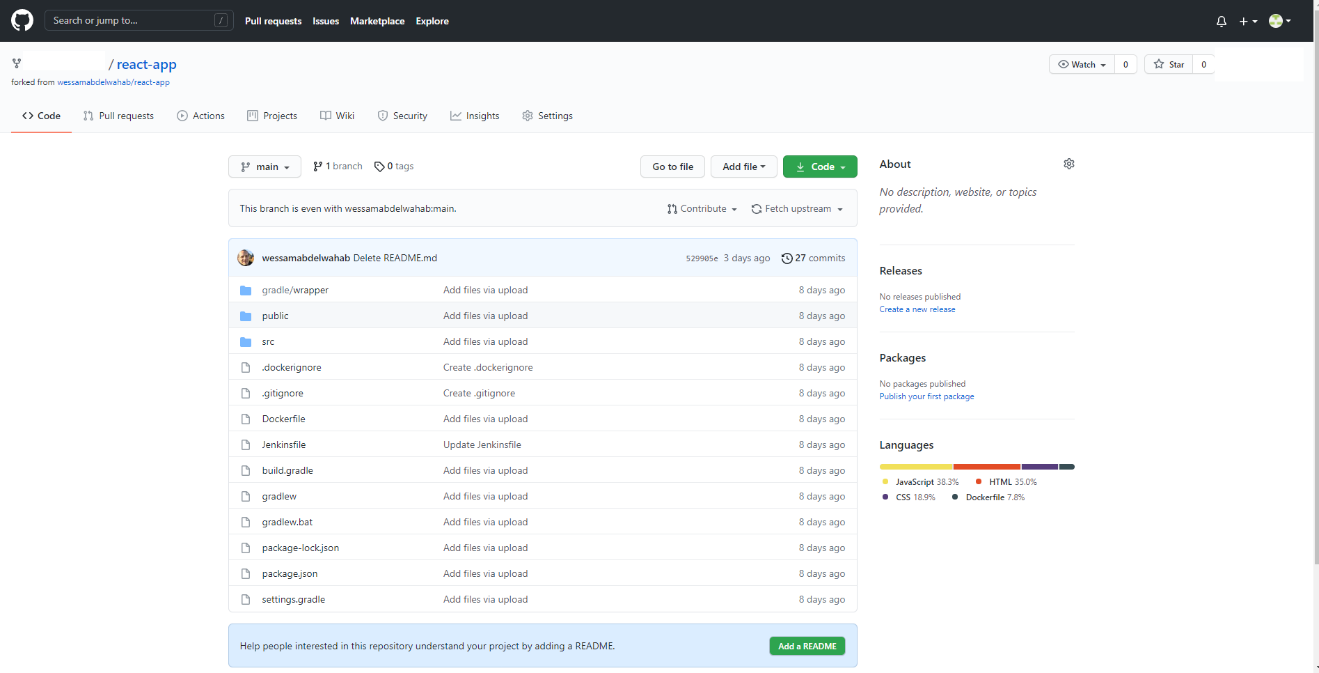
* + terraform init: Locate your working directory (Lab6- Jenkins Pipeline) and initialize terraform
  + terraform validate: Validate your terraform
  + terraform apply: Provision terraform managed infrastructure. You must confirm by tying yes if you would like to continue and perform the actions described to provision your infrastructure resources

## **Step 5: Connect to Jenkins**

* + Refer to Lab 5 instructions from the previous module (Week5-Continuous Integration)

## **Step 6: Fork GitHub Repository**

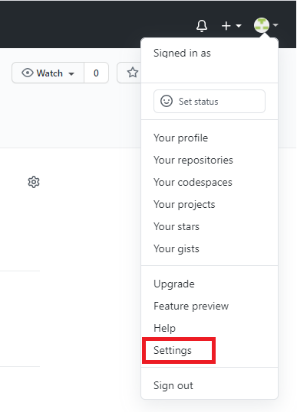
* + Log in to your GitHub account
  + Then Fork your own copy of **wessamabdelwahab/react-app** repository to your account [link](https://github.com/wessamabdelwahab/react-app)



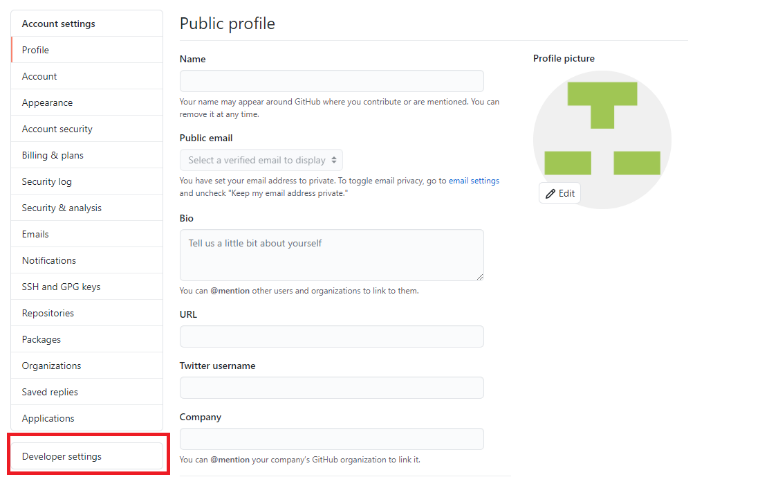
* + The forked repository link should look like this https://github.com/<Your Account>/react-app

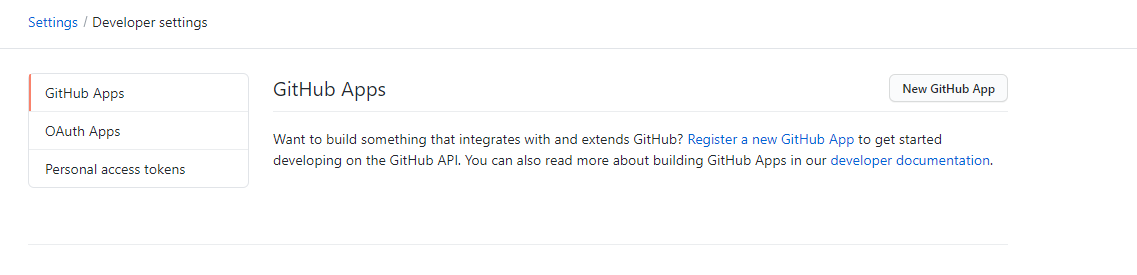
## **Step 7: Generate GitHub Access Token**

* + Go to GitHub Settings



* + Then go to Developer Settings from the left side menu

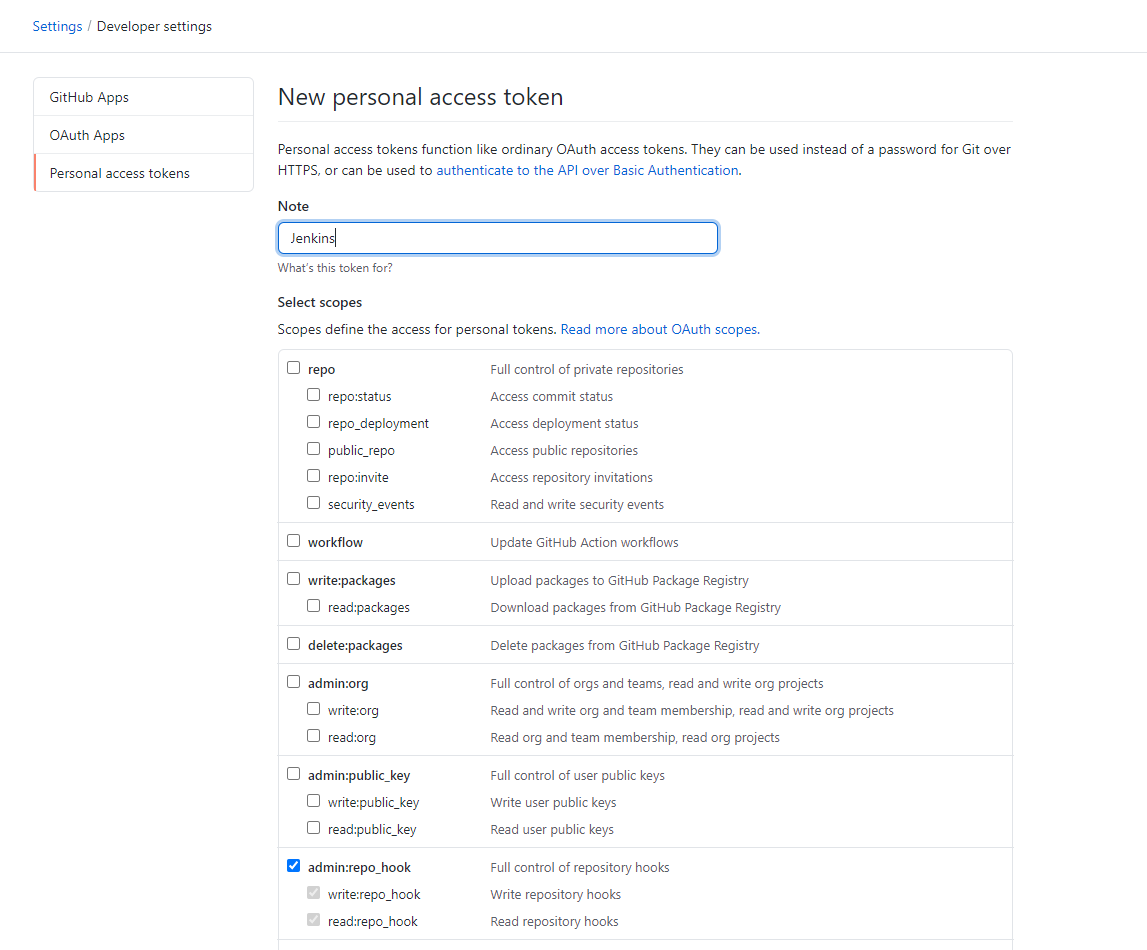




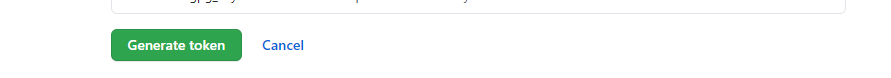
* + Then click on Personal access tokens from the left side menu



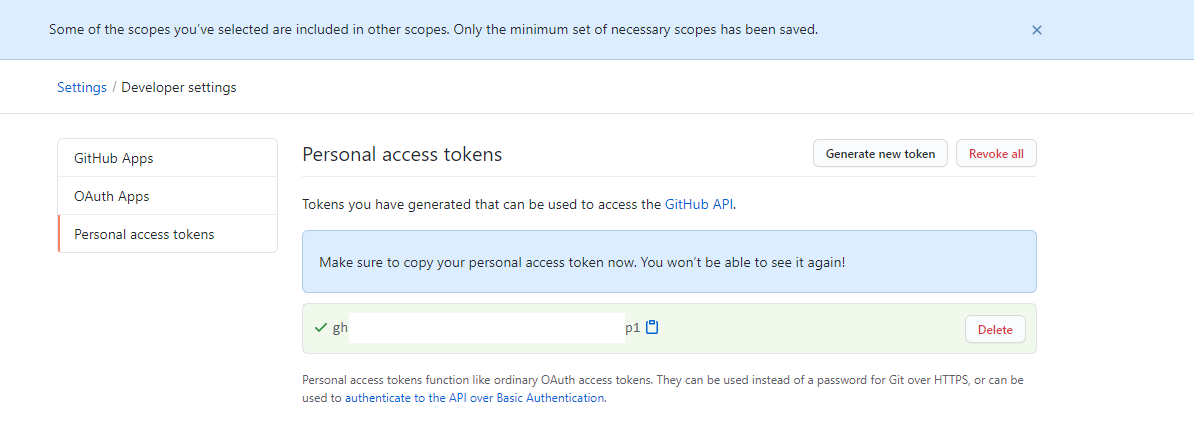
* + Choose Generate new token from the top right
  + Note: Jenkins
  + Select: admin:repo\_hook this webhook will notify Jenkins in your source code repository to start the continuous integration and delivery.



* + Then click Generate token

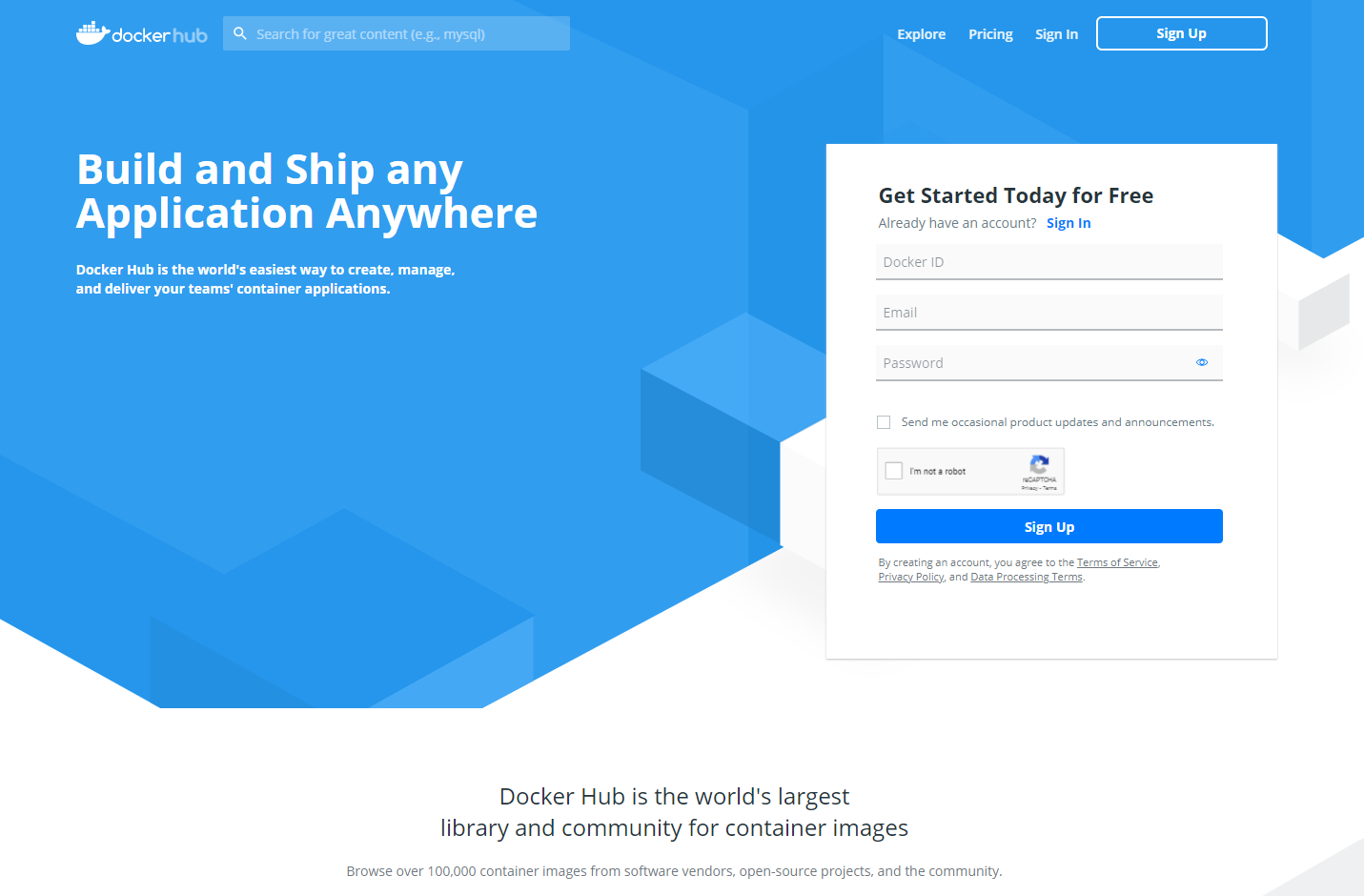


* + Make sure to copy your personal access token now and keep it in a safe place. You will not be able to see it again!



## **Step 8: Create DockerHub Account**

* + Create DockerHub Account [link](https://hub.docker.com/). You will need later to push and pull Docker images for your Pipeline
  + Enter the following:
  + Docker ID: e.g., your name
  + Email: your email
  + Password



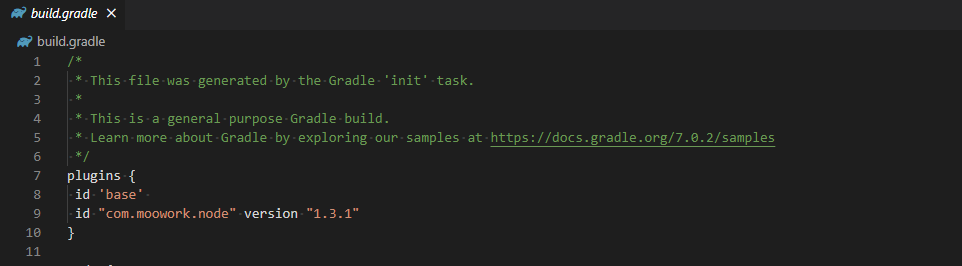
* + Please check your inbox to verify the email associated with this account. You will not be able to create a repository or configure your Docker Hub without verifying your email address.

Now let us understand the Gradle build, Docker, and Jenkins files before creating the Pipeline in Jenkins.

## **Step 9: Create Gradle Build**

You will be utilizing Gradle build automation for the react web application source code located in GitHub (build.gradle)

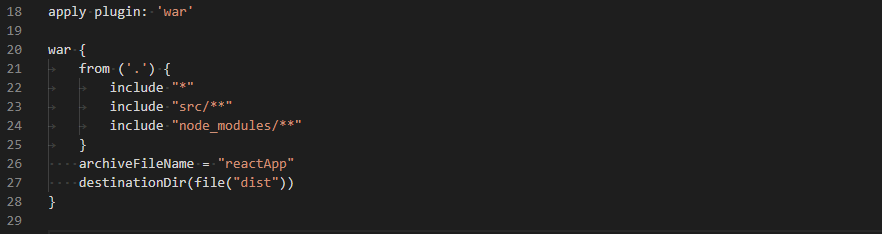
* + We applied the base plugin that gives the base set of tasks that can be used in the build file and also use the com.moowork.node plugin for executing node scripts.



* + Next, configure the node plugin to download and use node version 14.17.1 and npm version 6.14.13.



* + Then we apply the war plugin to assemble the application WAR file specifying the files to be included into the archive filename reactApp in the destination directory dist.

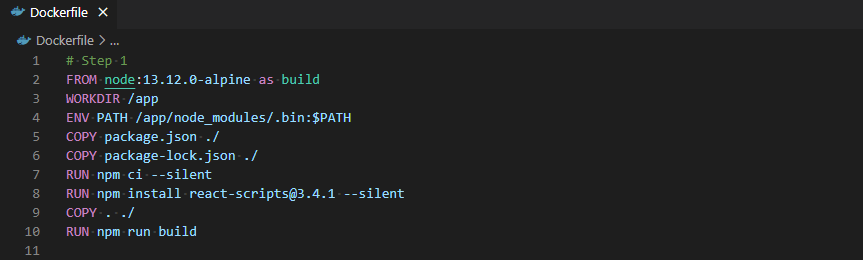


## **Step 10: Create Docker File**

In this step, we will build a Docker file to containerize the React Web App or in other words, build a container image to run in any environment. This is a multi-stage build process consist of two phases, a build step, and a run step. This allows you to selectively copy artifacts from one stage to another, leaving whatever you don’t want in the final image.

Step 1 (Build)

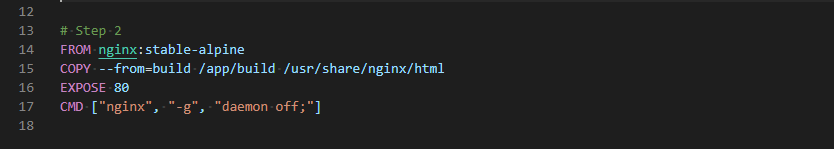
* + We are pulling the node base image from the Node Docker Hub (node:13.12.0-alpine). This will set up the Node for building the React application. The (FROM) used as instruction in the docker file for each new stage of the build. The (as build) descriptor is required for the multi-stage build as we will be using the build artifacts in the second stage of our docker file.
  + Specify the working directory to the container where the application files will run/build from (WORKDIR /app).
  + adding `/app/node\_modules/.bin` to $PATH (ENV PATH /app/node\_modules/.bin:$PATH)
  + Install the React application dependencies.
  + Copy package files (COPY package.json ./ & COPY package-lock.json ./) to the app folder.
  + (RUN npm ci --silent)
  + (npm install react-scripts@3.4.1 --silent)
  + Add all the application files (COPY . ./ )
  + (RUN npm run build) command to build the React application code so it can be deployed in the container.



Step 2 (Run)

Now the React application is built, we need to deploy and run it into Docker.

* + Nginx image will be used for the run stage (FROM nginx:stable-alpine). This will create an nginx server to deploy the react application.
  + (COPY --from=build /app/build /usr/share/nginx/html) This command will copy the React build folder from the build step to the nginx server folder.
  + Expose server at port 80 (EXPOSE 80)
  + Start app (CMD ["nginx", "-g", "daemon off;"]). This command will make nginx stay in the foreground so that docker can track the process properly; otherwise, the container will stop immediately after starting

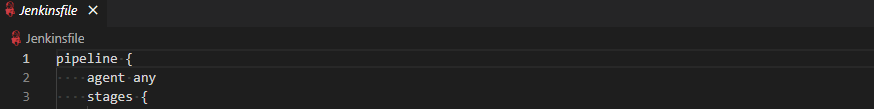


Note: I encourage you to read more about the multi-stage builds in docker.

## **Step 11: Create Jenkins File**

Create a Jenkinsfile, which is a text file that contains the definition of a Jenkins Pipeline. This file contains a declarative pipeline with multiple stages for the continuous delivery pipeline.

* + The definition filed indicates the pipeline script option.
  + The agent directive instructs Jenkins to allocate an executor and workspace for the Pipeline. Without an agent directive, the declarative Pipeline is not valid and will not work. By default, the agent directive ensures that the source repository is checked out and made available for steps in the subsequent stages.
  + The stages directive is required for a valid declarative pipeline. It contains a sequence of one or more stage directives. The stages section is where the bulk of the work described by a Pipeline will be located.



* + The stages and steps directives are also required for a valid declarative pipeline as they instruct Jenkins on what to execute and in which stage it should be executed.
  + The steps section defines a series of one or more steps to be executed in a given stage.
  + The Build Stage has the following shell steps
  + Set the execution permission for gradle wrapper (gradlew)
  + Then run gradle build (./gradlew build)
  + archiveArtifacts captures the files built matching and saves them to the Jenkins controller for later retrieval. Notice this must match the destination directory and file archive name from the gradle build file.



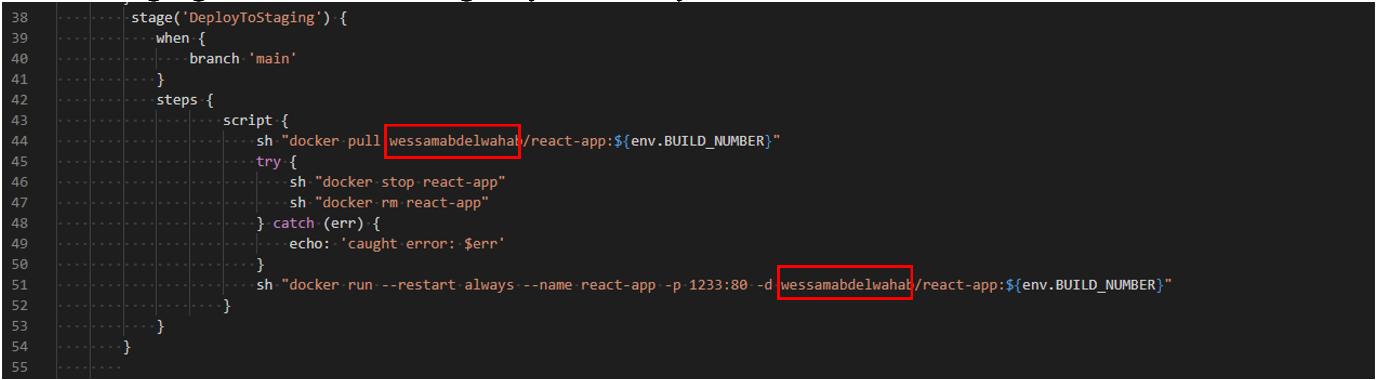
* + Build Docker image Stage when I am on the main branch, docker build command will be run to build the image. Note: Change my name to your Docker account name.



* + Push the docker image to your Docker Hub; after successful Docker image build command, the image will be pushed to your docker hub. We will configure later in Jenkins credentials your DockerHub account; it is important to keep the ‘dockerhub\_login’ ID match the ID of the credential in Jenkins.



* + Pull the docker image that matches the build number from Docker Hub and deploy it in the staging area. Note: Change my name to your Docker account name.



* + Test the HTTP response code for your containerized sample react web application in the staging area, and based on the response code; you could approve or abort the deployment of your containerized sample react web application to production.

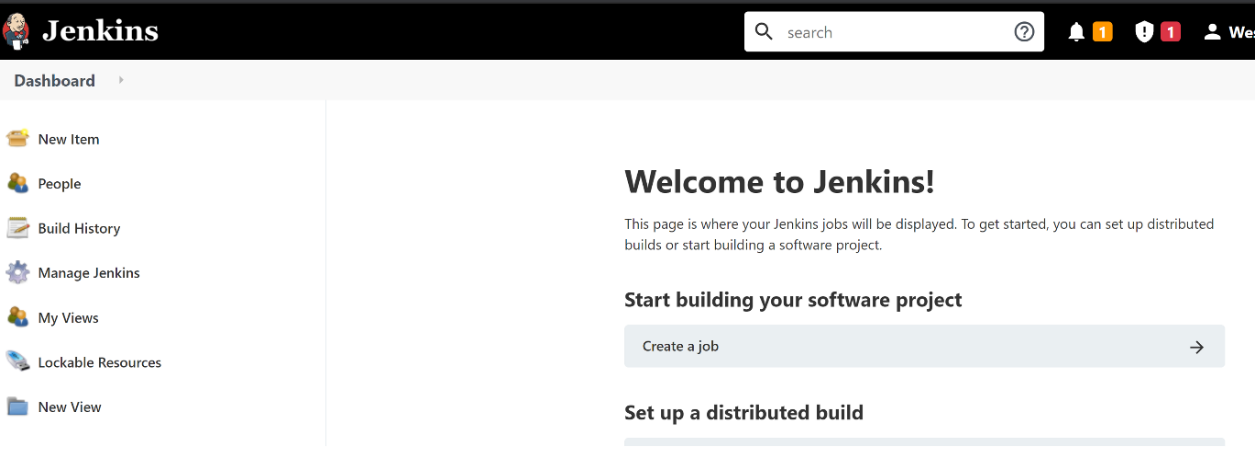


* + Deploy the docker image to the production environment if you receive a 200 response in the staging environment. Note: Change my name to your Docker account name.

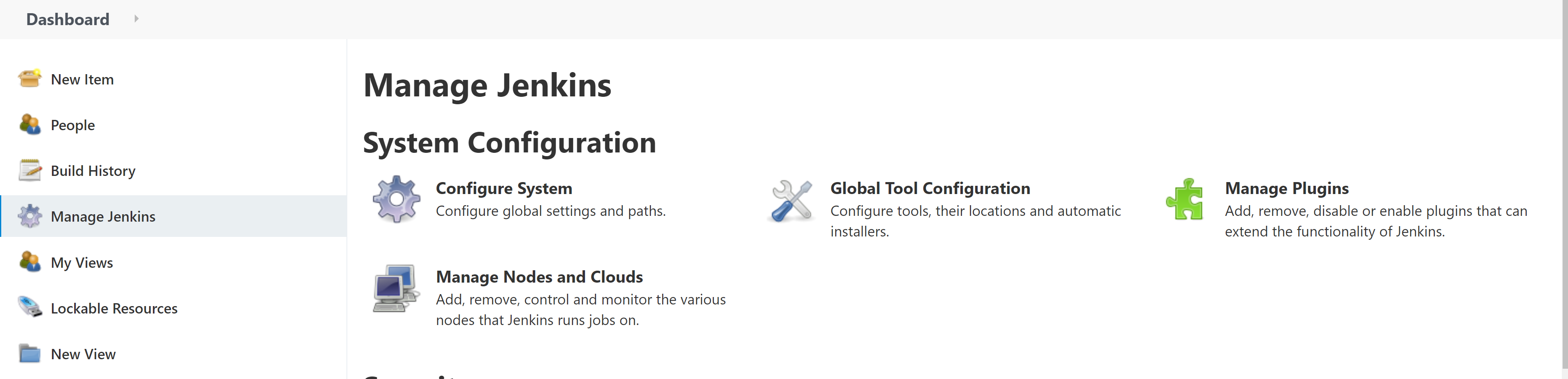


## **Step 12: Install Jenkins Plugins**

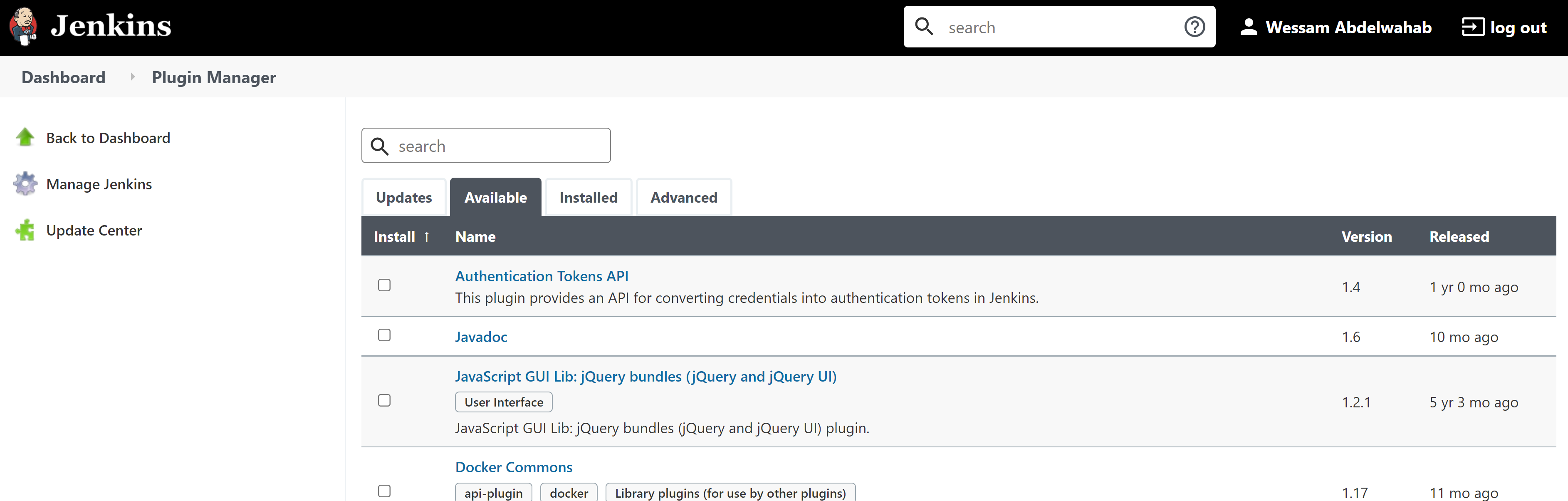
* + Login to Jenkins



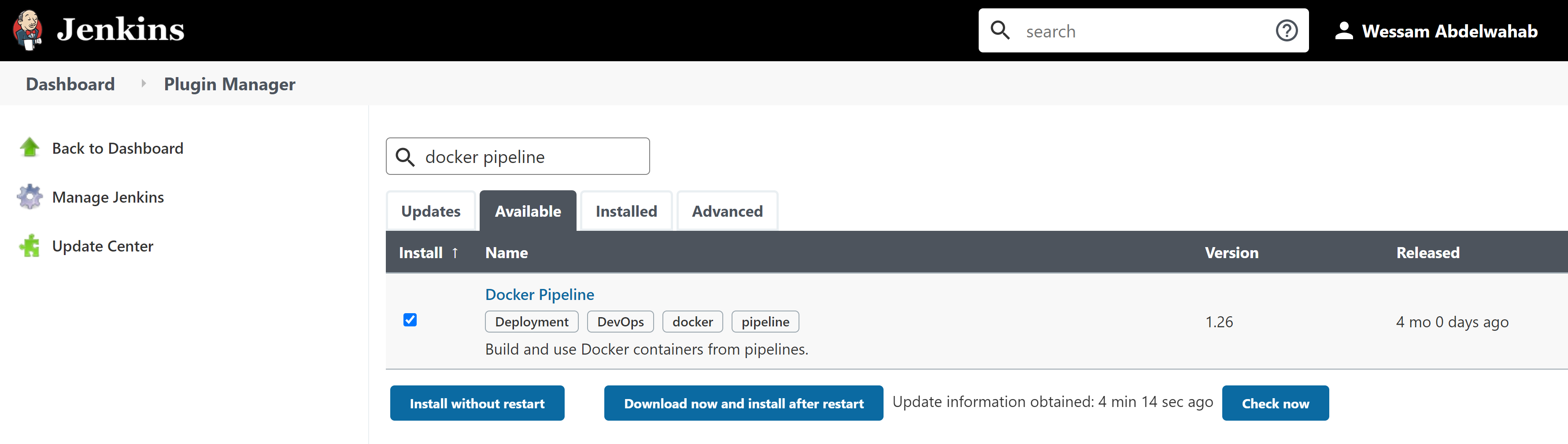
* + Go to Manage Jenkins on the left side menu, then go to Manage Plugins



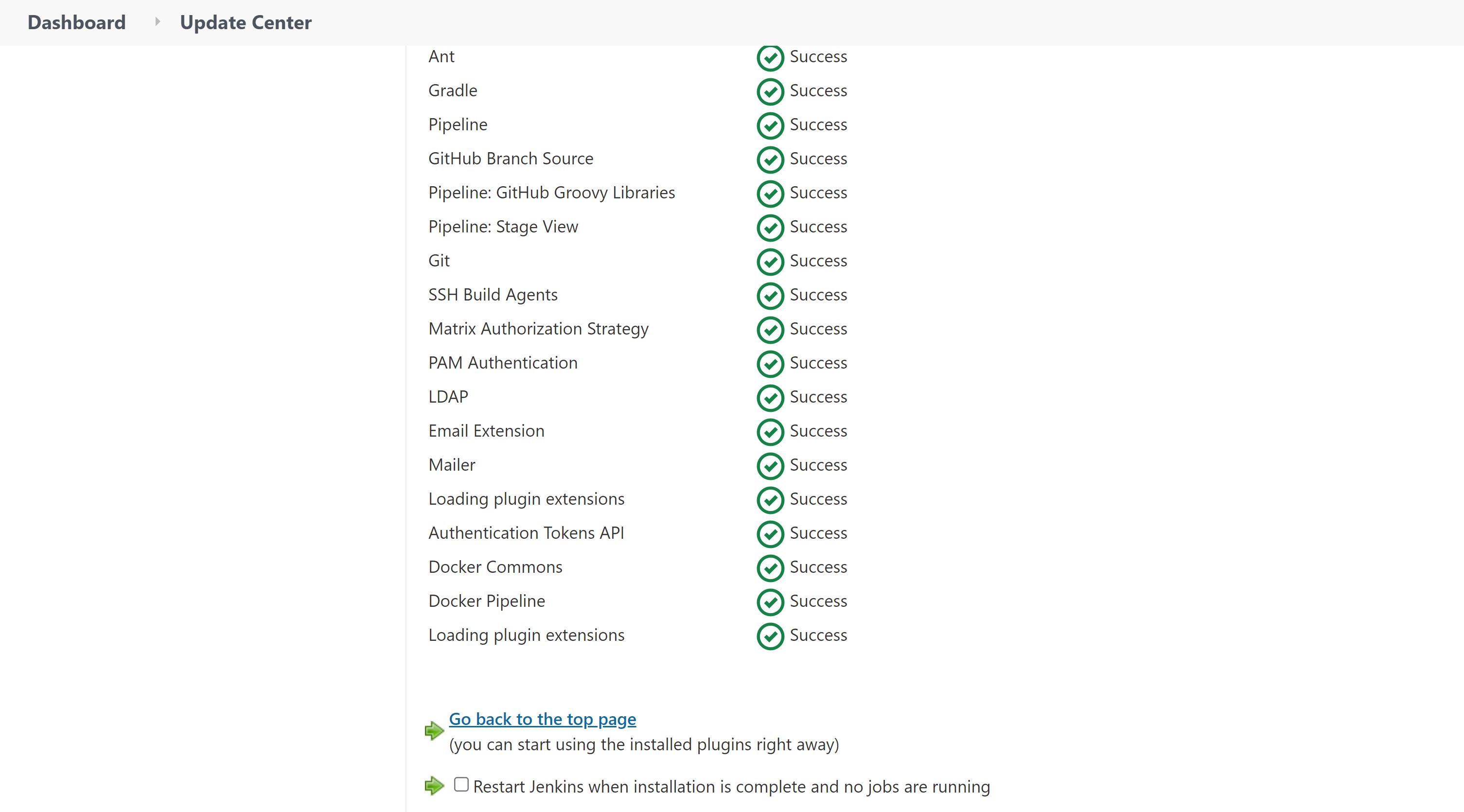
* + Go to Available



* + Search for docker pipeline in the available plugins. This plugin will allow Jenkins to build and use Docker containers from pipelines

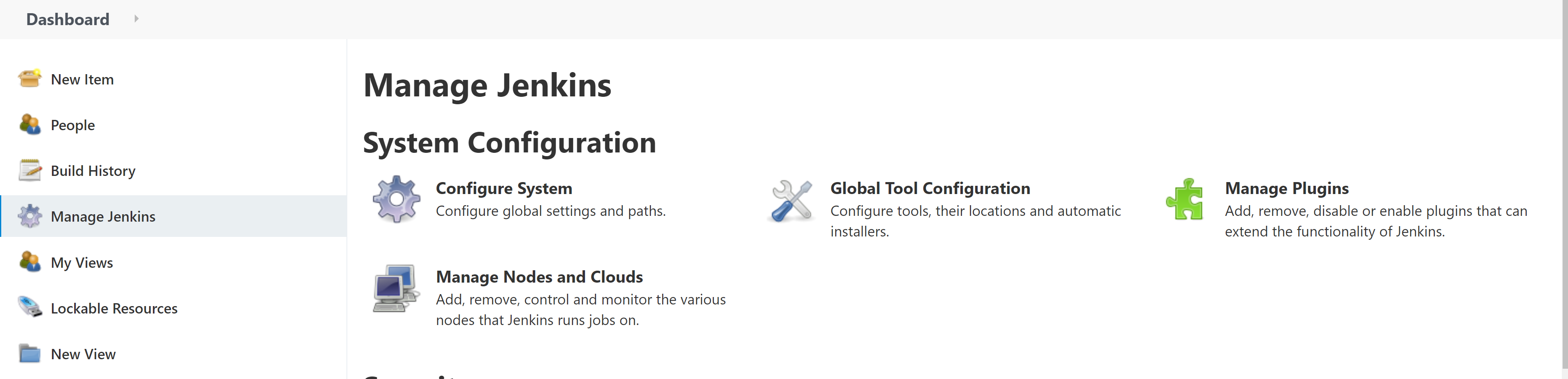


* + Click on Install without restart button

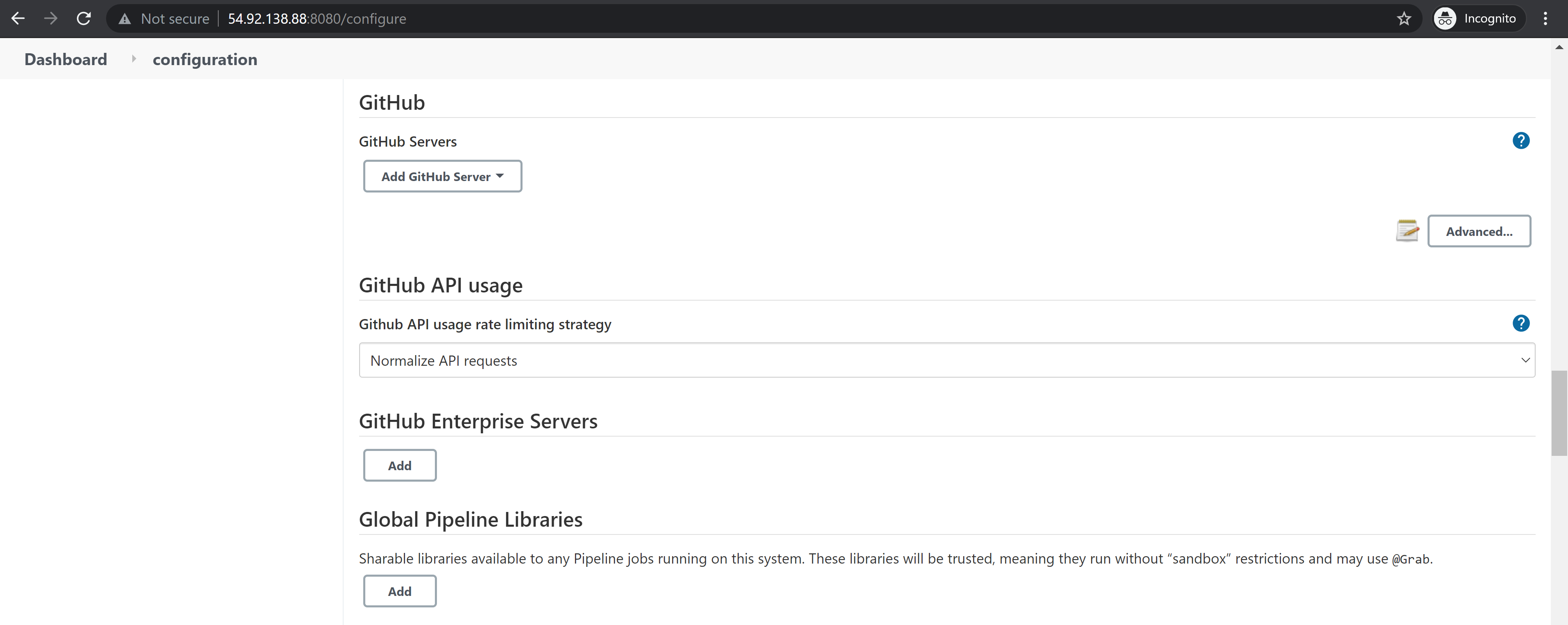


## **Step 13: Configure the generated GitHub Access Key in Jenkins**

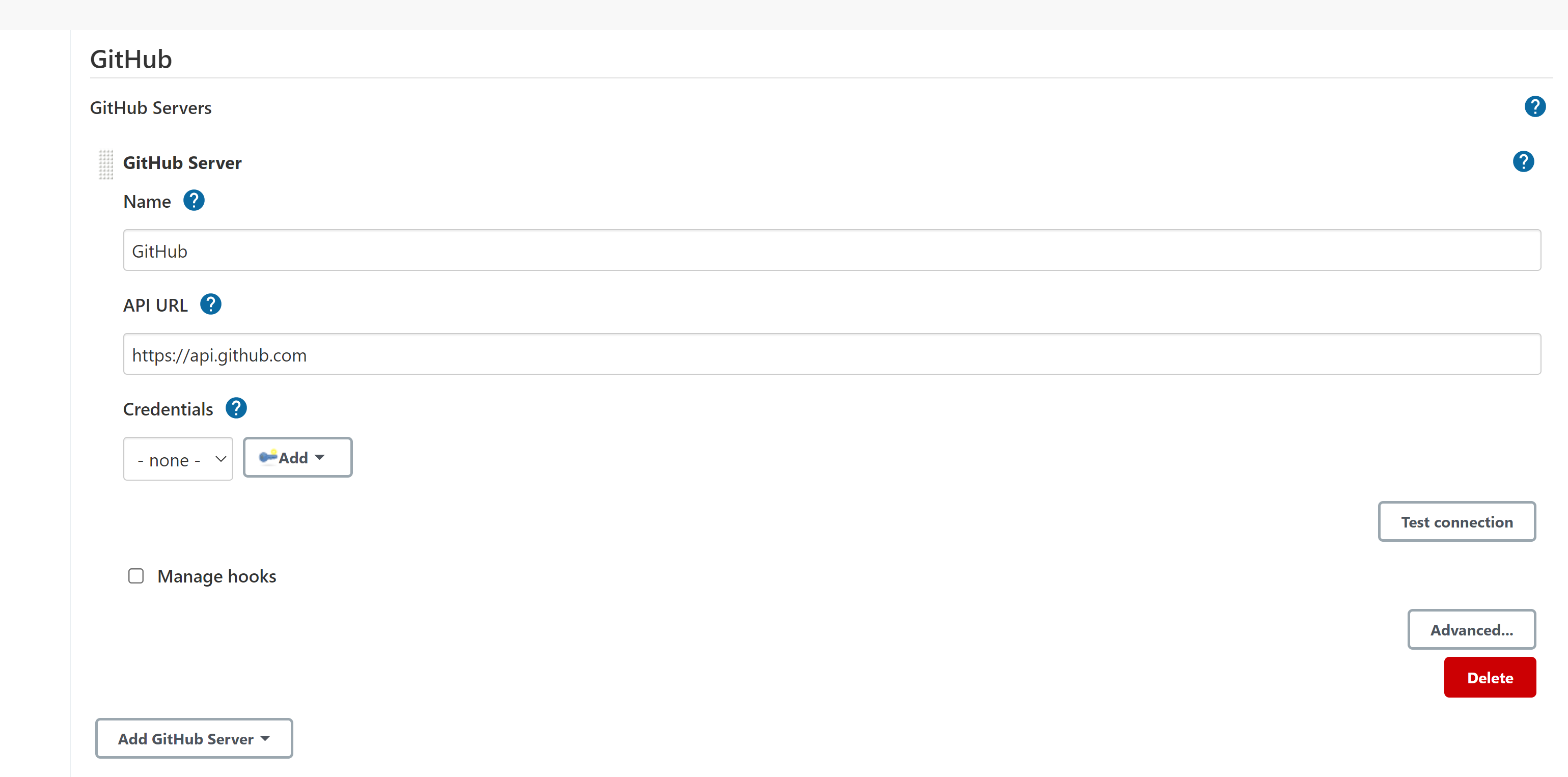
* + Go to Manage Jenkins on the left side menu, then go to Configure System



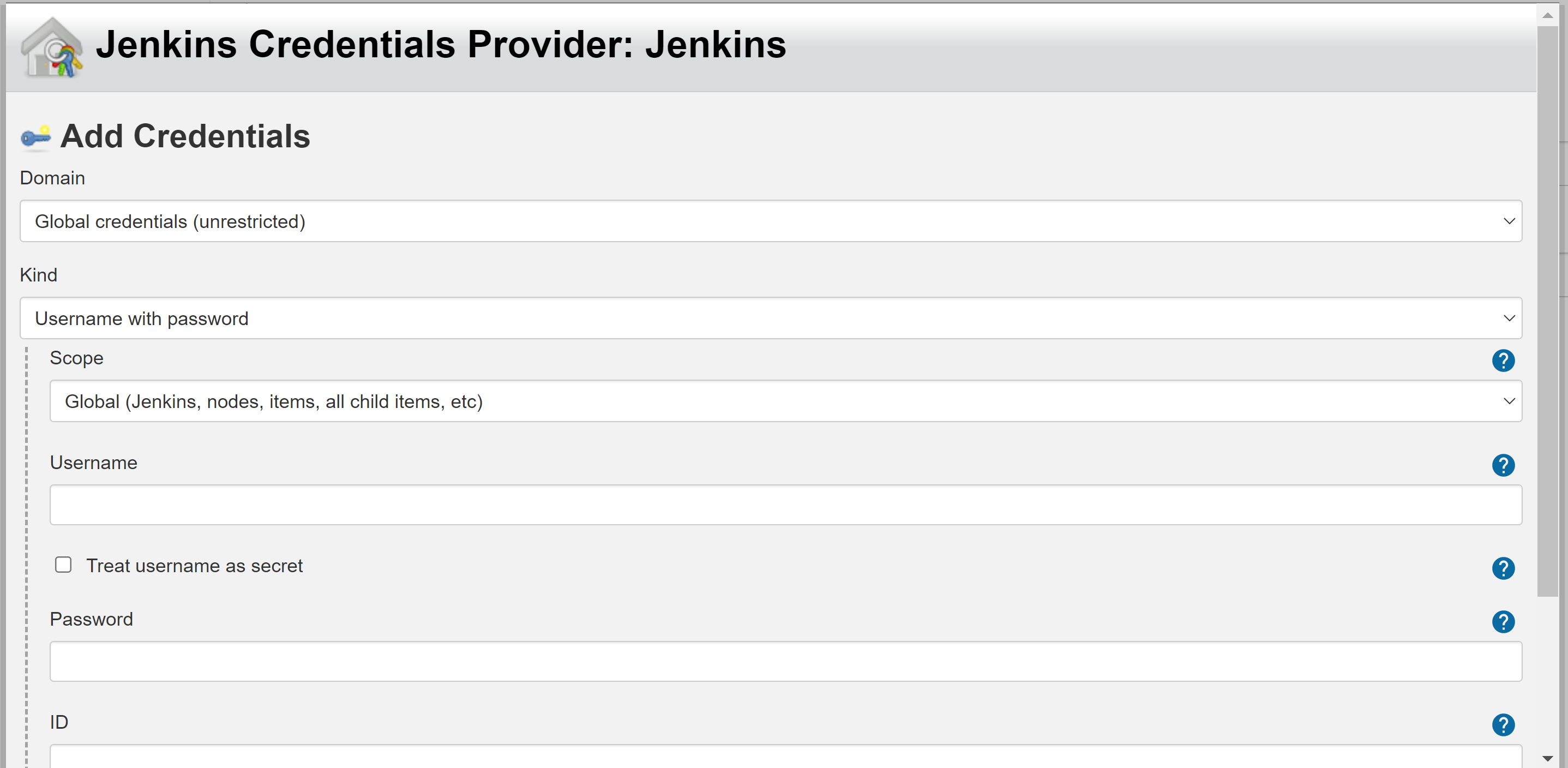
* + Scroll down to GitHub -> Select Add GitHub Server -> Github Server



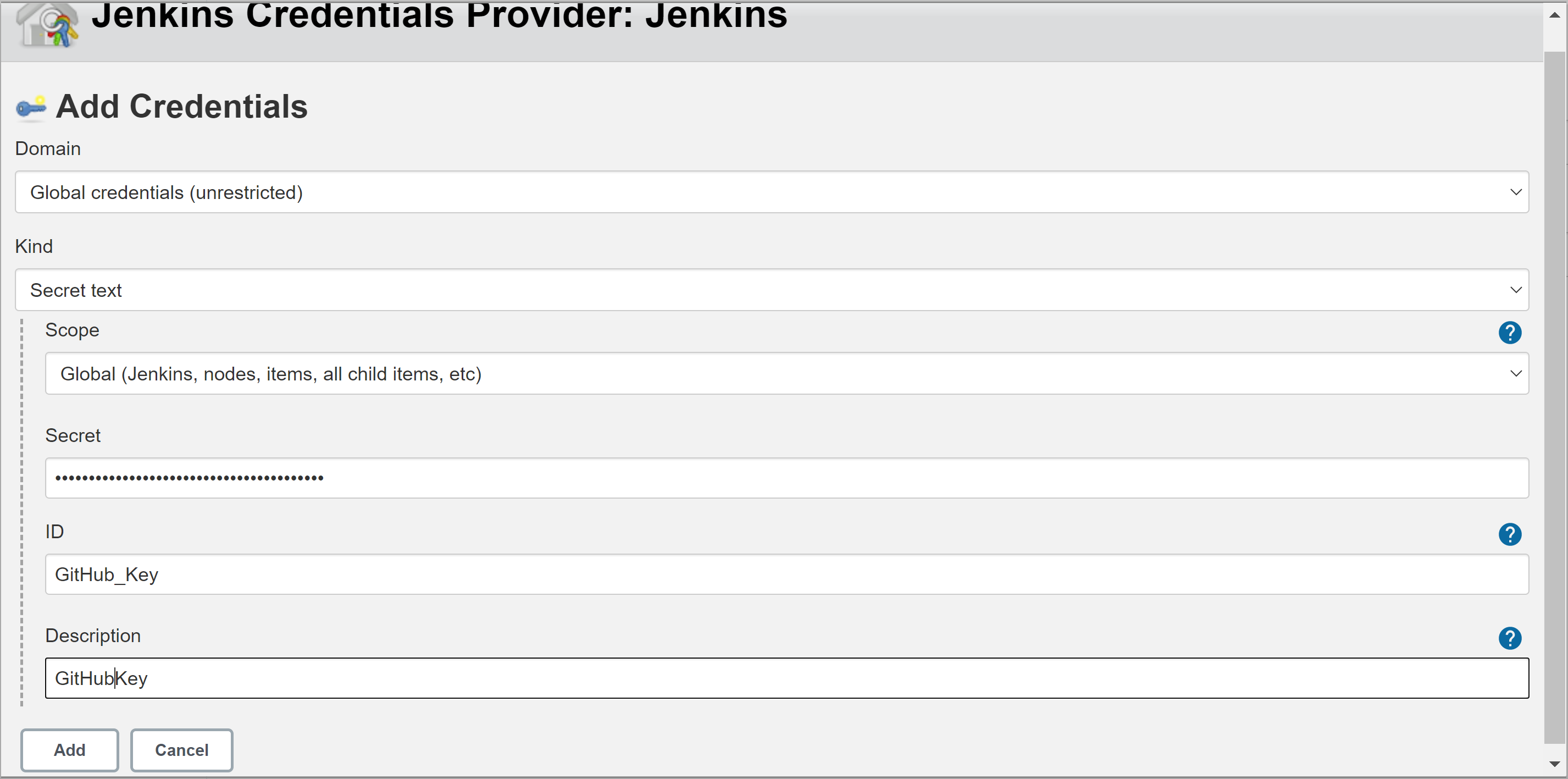
* + Enter the Name: GitHub



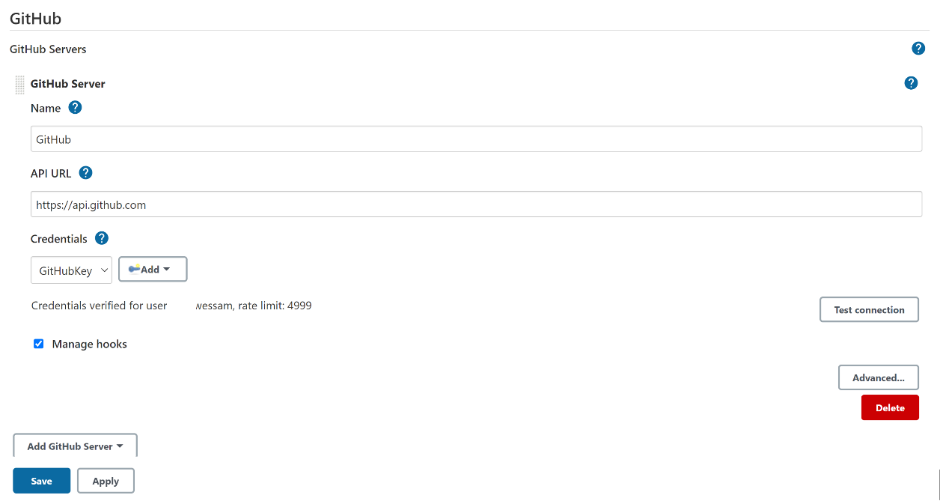
* + Click Add -> Jenkins



* + Select Kind: Secret text
  + Enter the following, then click Add
  + Secret: Enter the generated GitHub Access Key
  + ID: GitHub\_Key
  + Description: GitHubKey



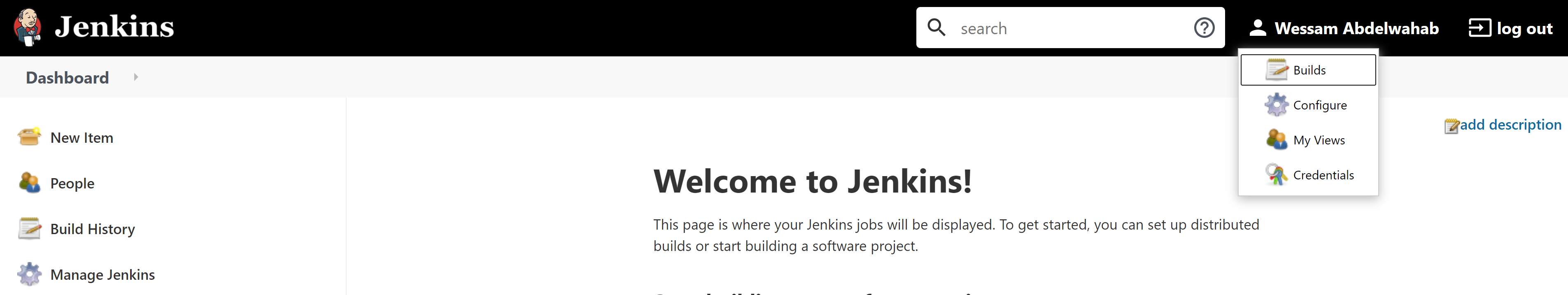
* + Enter the following, then click Add
  + Select GitHubKey from the drop-down under Credentials, select Manage hooks (This will allow webhooks to work whenever any change is made on your GitHub repository), and then click Test connection

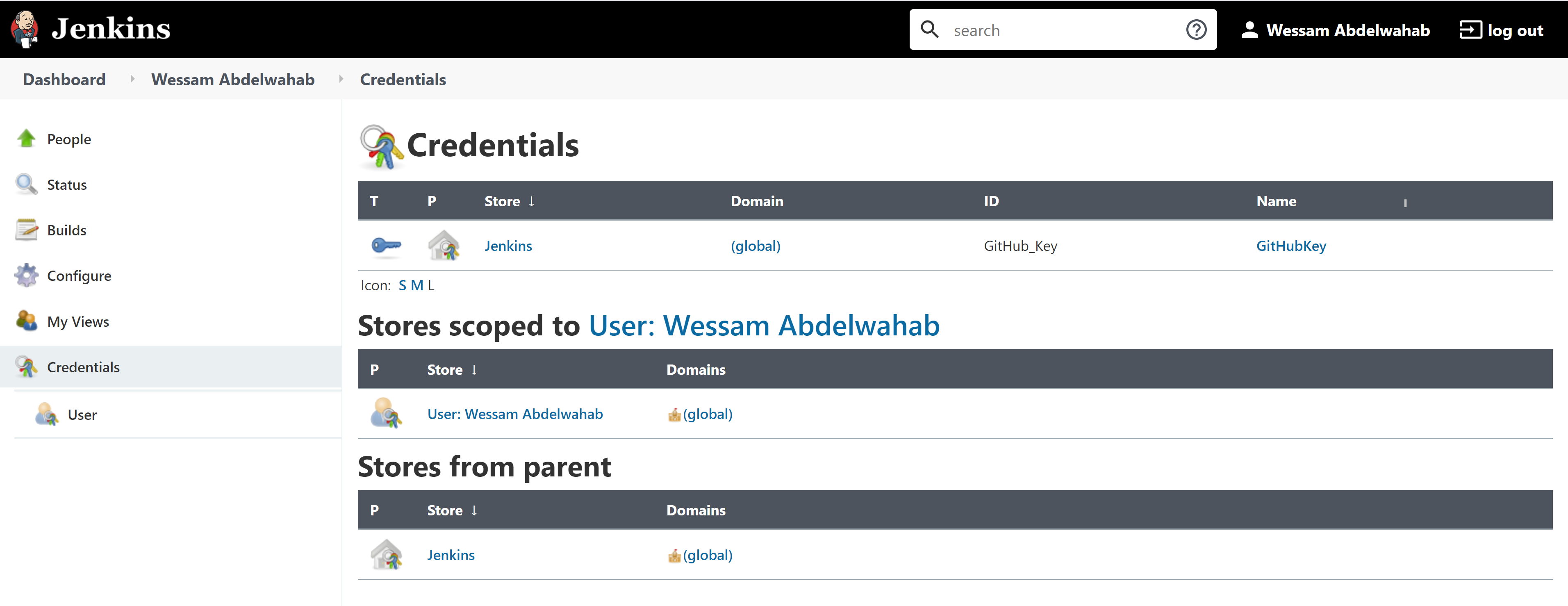


* + Click Save

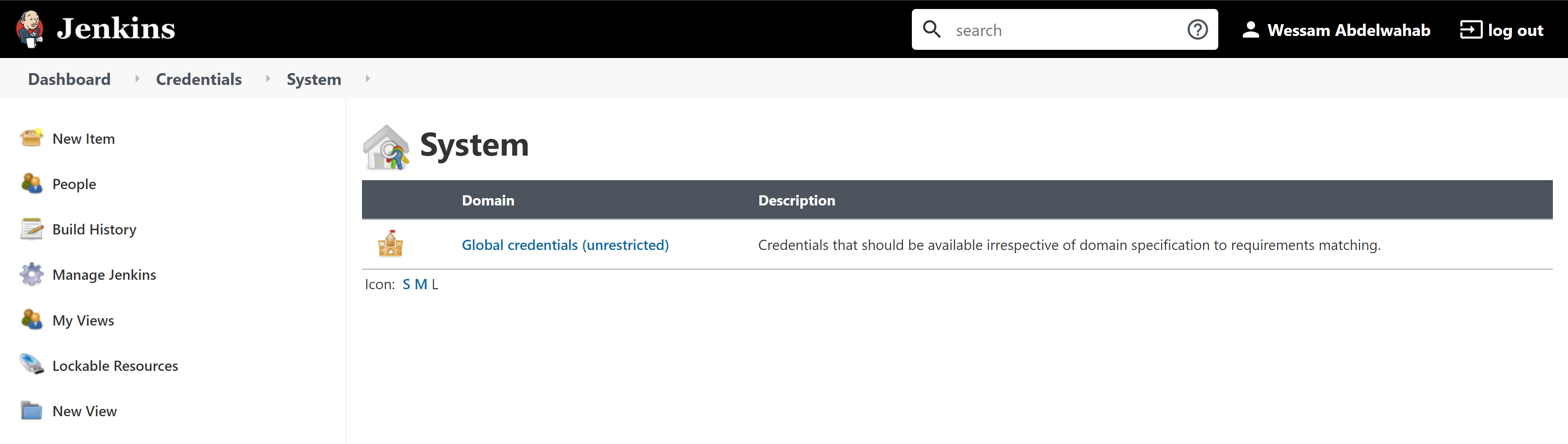
## **Step 14: Configure GitHub and DockerHub in Jenkins Credentials**

* + Click on the small arrow under your user and select Credentials

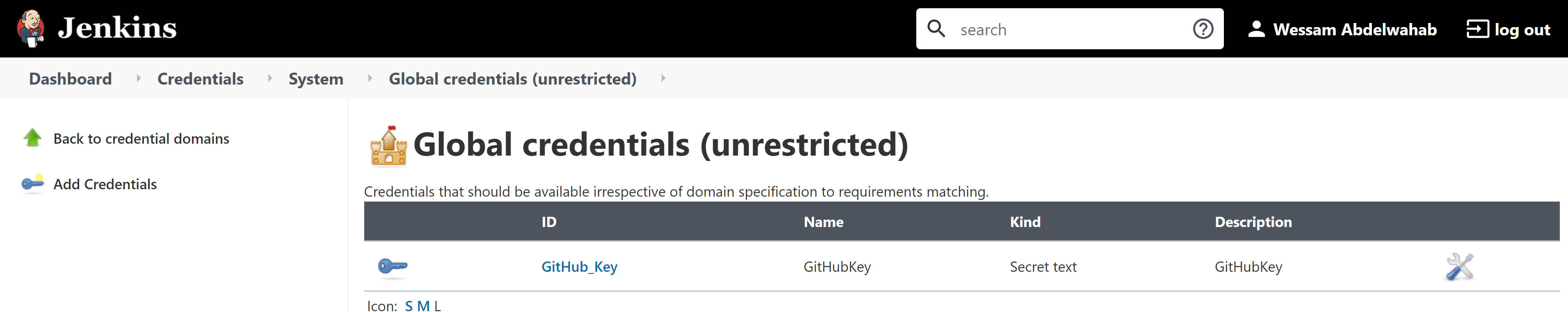




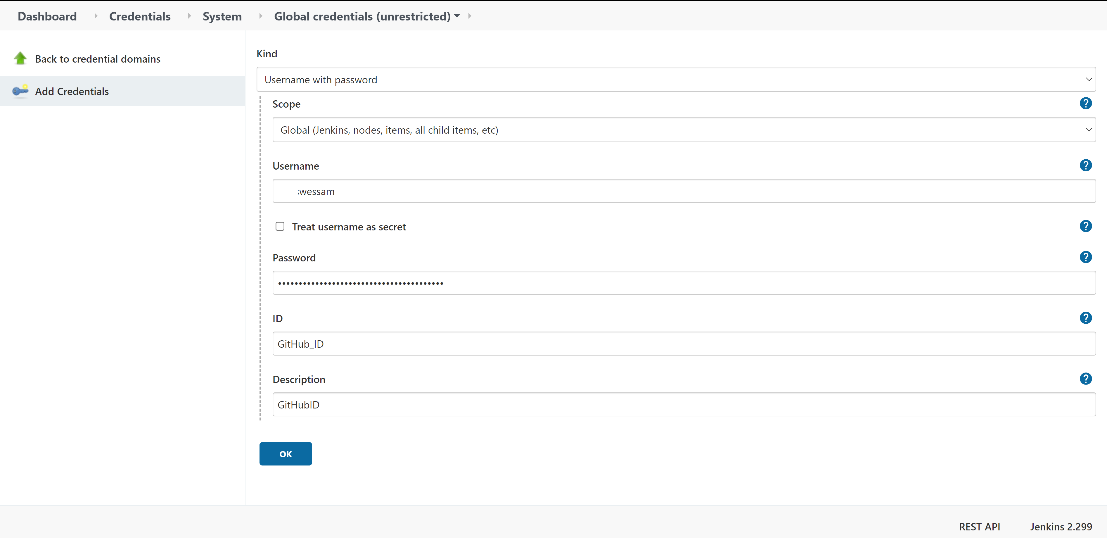
* + Select Jenkins under Stores from parent



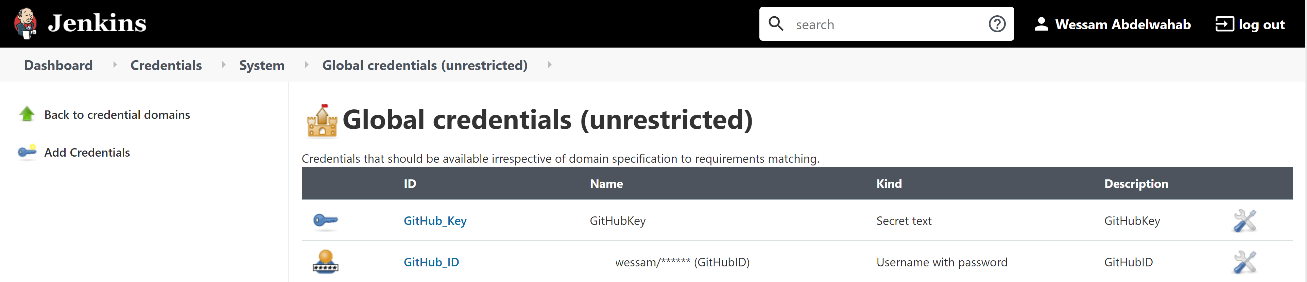
* + Then click on Global credentials (unrestricted)



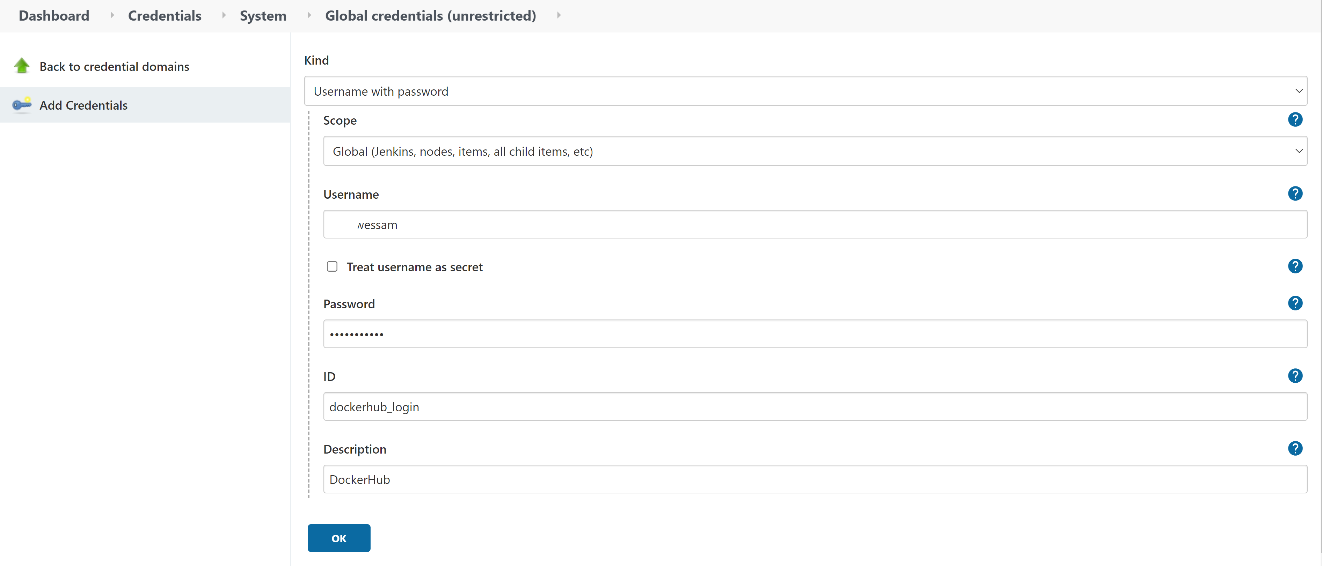
* + Then select Add Credentials from the left side menu
  + Add your GitHub Account by entering the following:
  + Username: Your Github User Name
  + Password: Enter the generated GitHub Access Key
  + ID: GitHub\_ID
  + Description: GitHubID



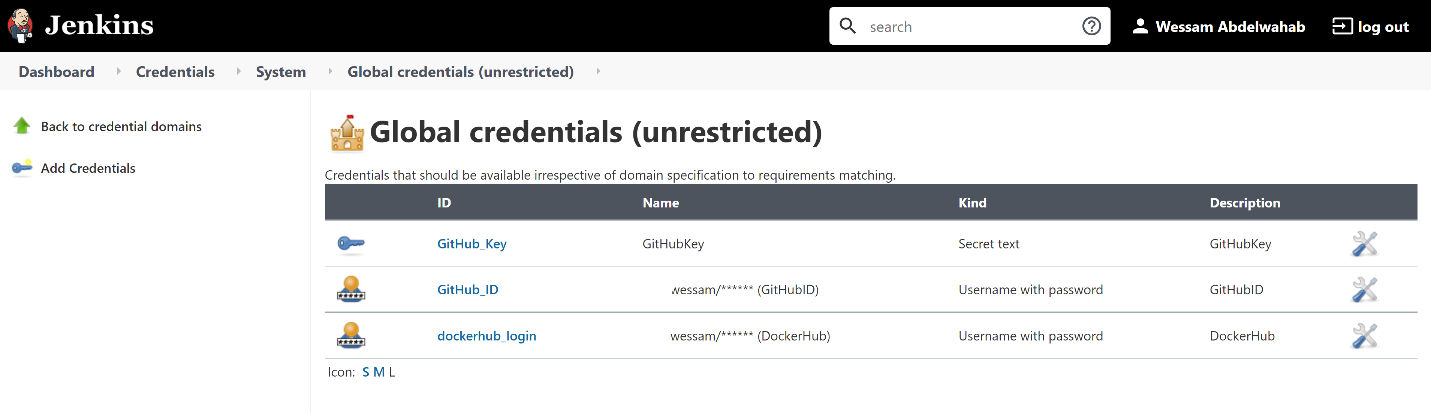
* + Then Click OK



* + Then select again Add Credentials from the left side menu
  + Add your GitHub Account by entering the following:
  + Username: Your Docker Hub User Name
  + Password: Enter your Docker Hub Password
  + ID: dockerhub\_login (It is important the keep the exact ID name as we are referring to it in Jenkinsfile)
  + Description: DockerHub

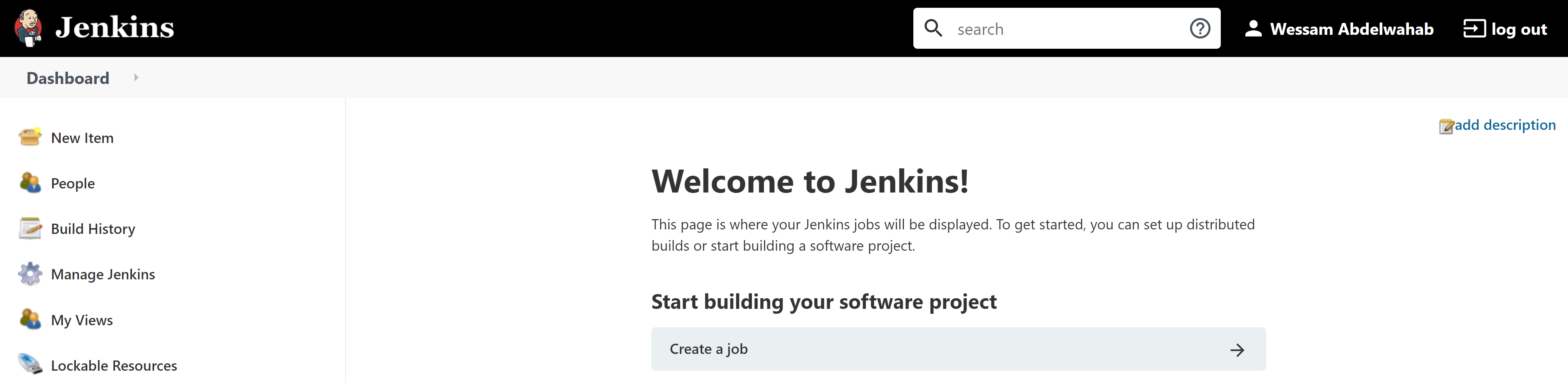


* + Then Click OK

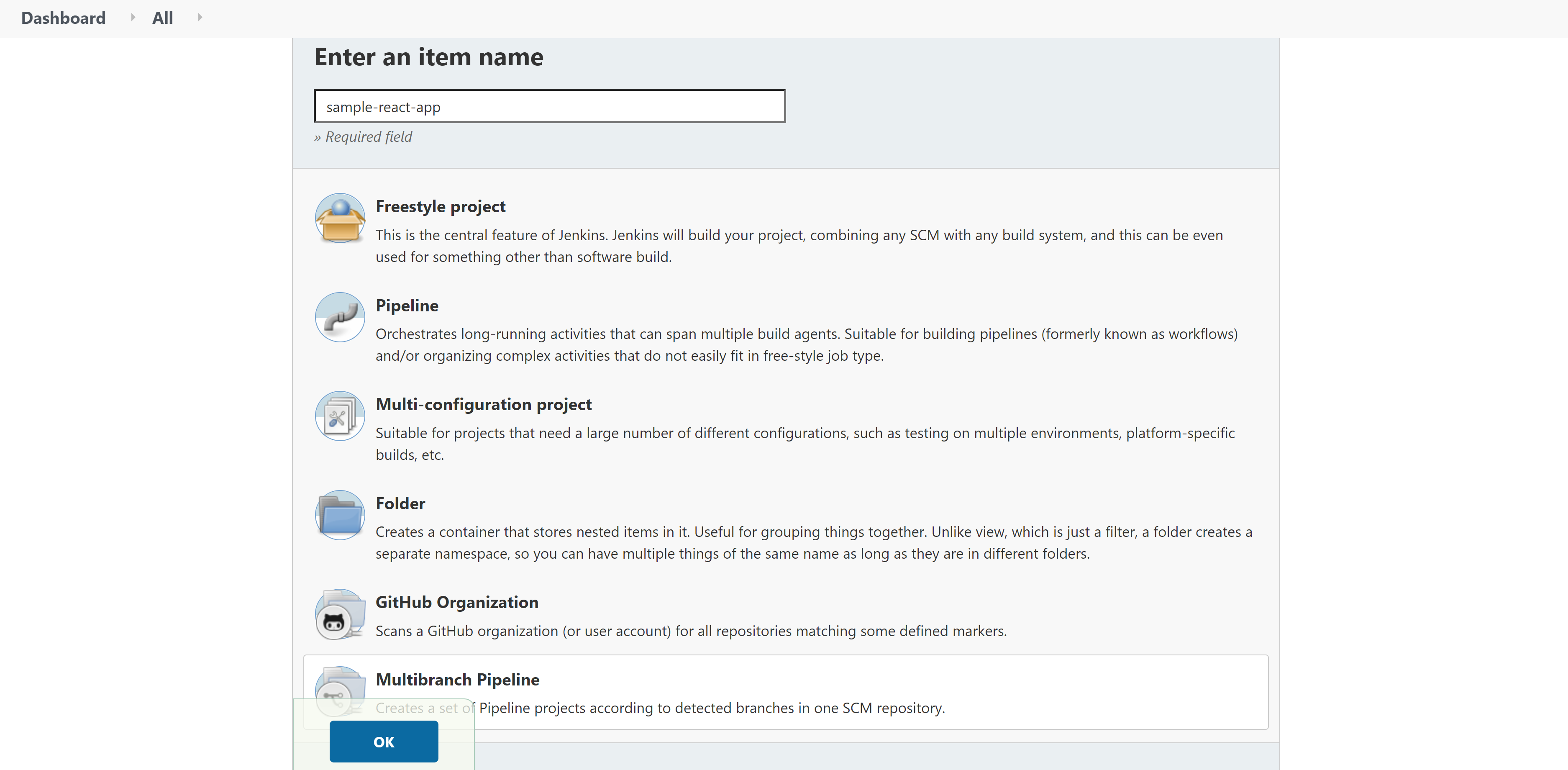


## **Step 15: Create a Multi-branch pipeline project in Jenkins**

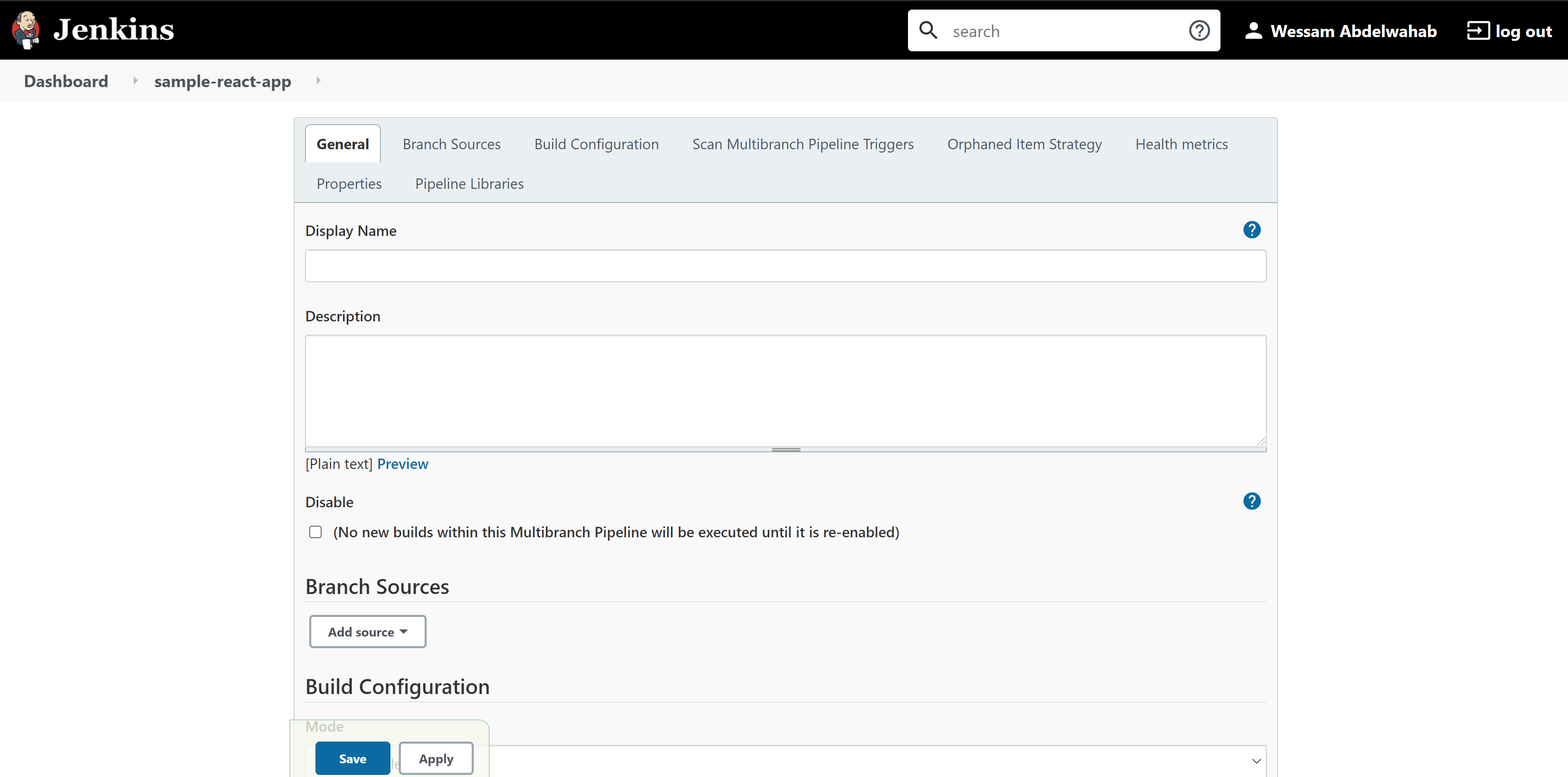
* + Navigate back to the Dashboard



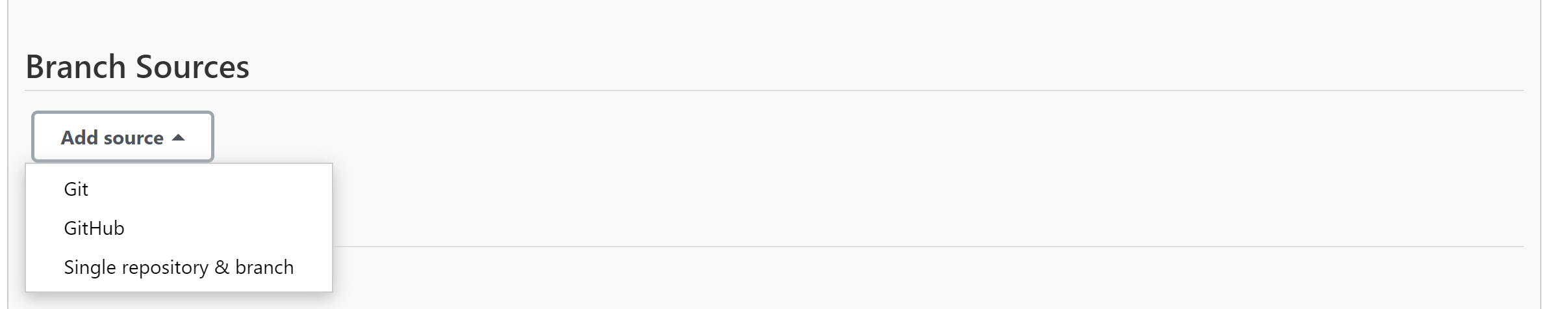
* + Now let us create your first freestyle project in Jenkins
  + Click on New Item on the left side menu
  + Enter an item name: sample-react-app
  + Select Multibranch Pipeline



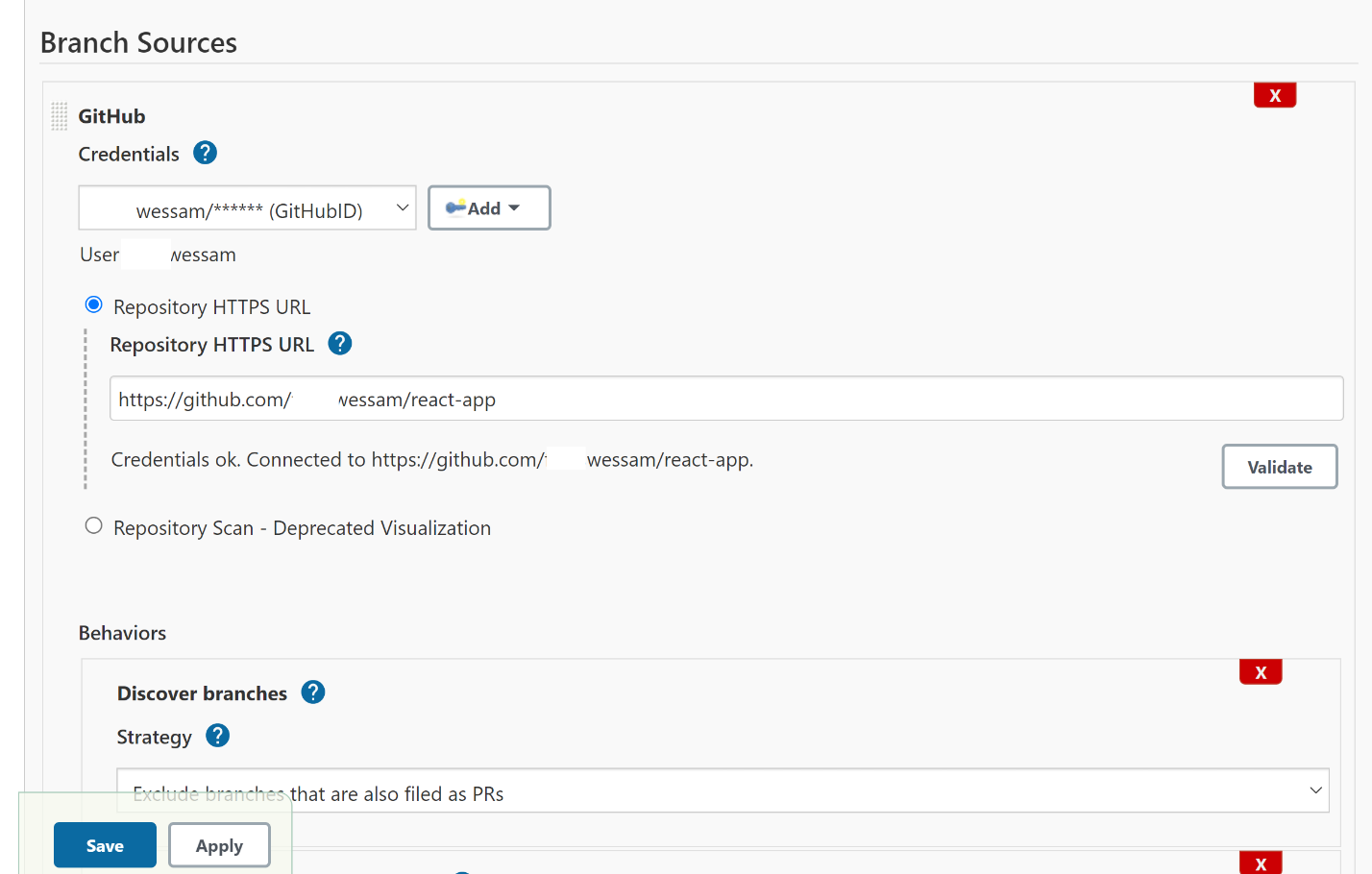
* + Then Click OK



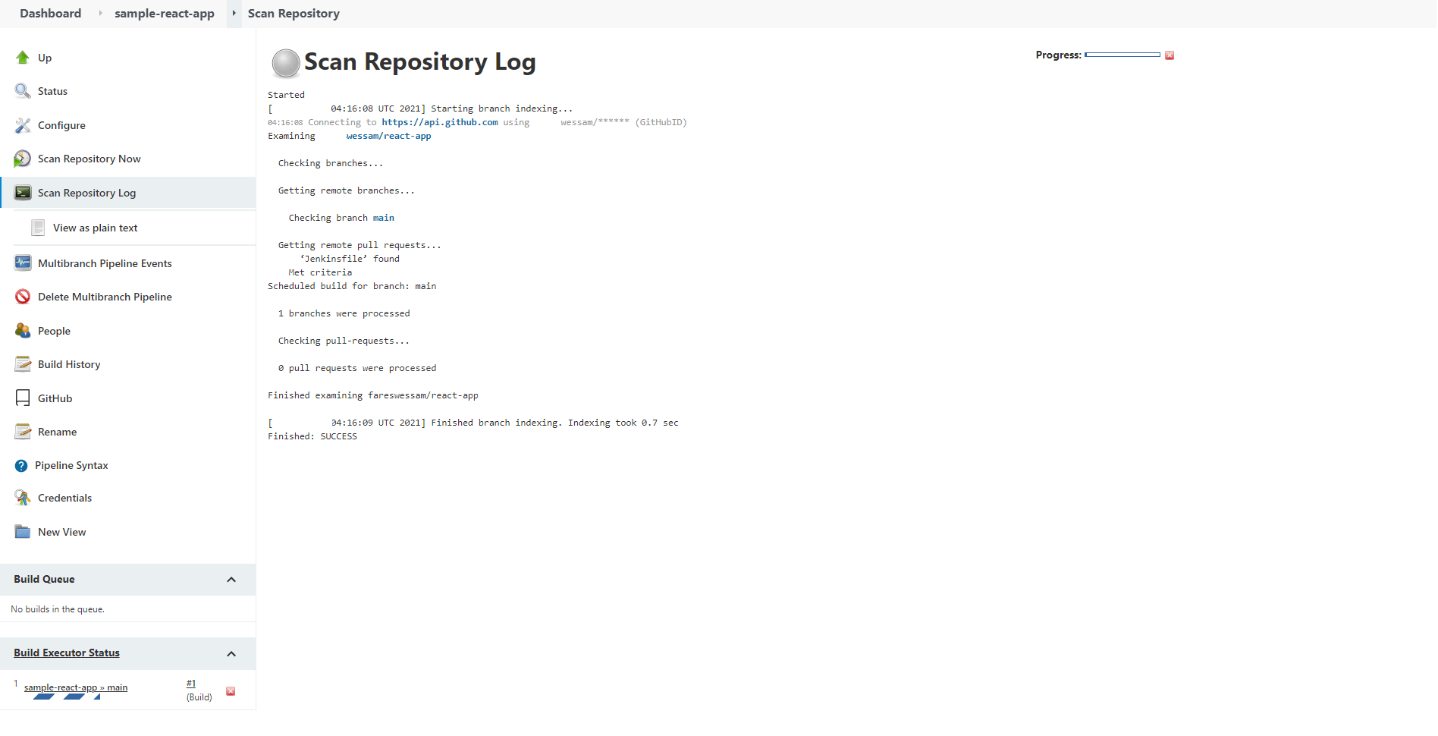
* + Under Branch Sources -> Add source -> GitHub



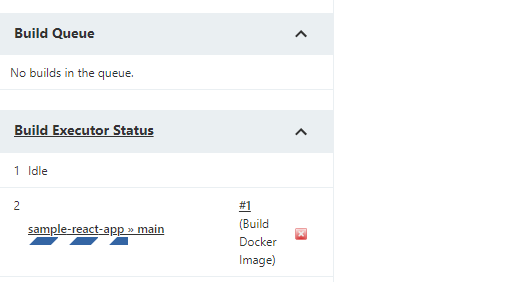
* + Under Credentials
  + Select GitHubID from the drop-down list
  + Add the react-app Repository HTTPS URL from your GitHub e.g., https://github.com/<Your Account>/react-app
  + Then Click Validate



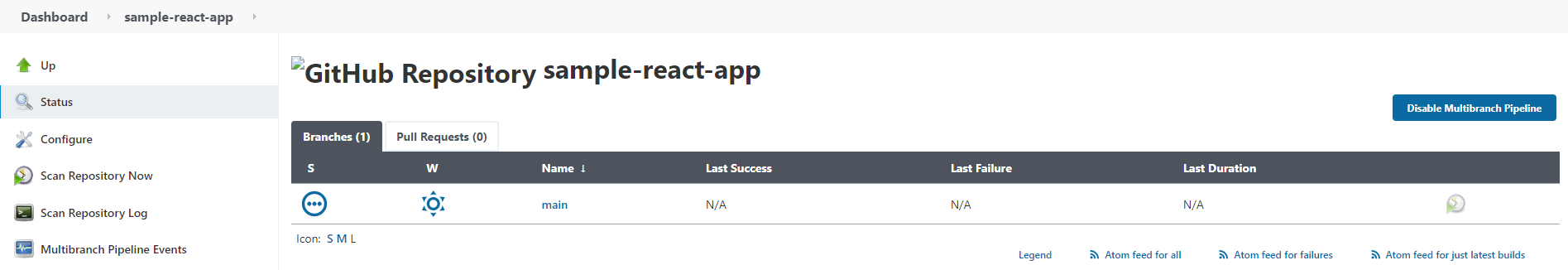
* + Then click Save



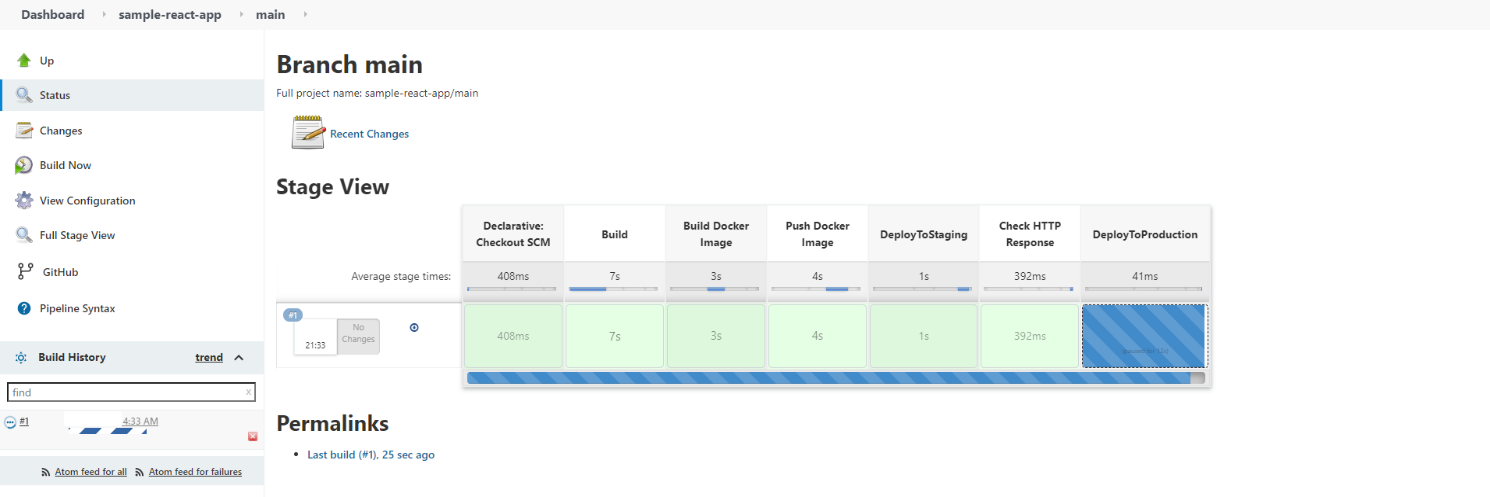
* + Click on the sample-react-app >> main under Build Executor Status



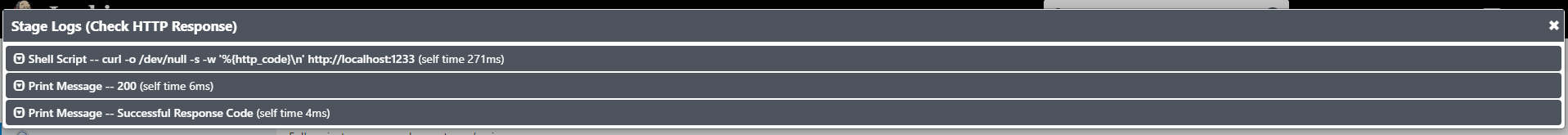
* + Select main under branches



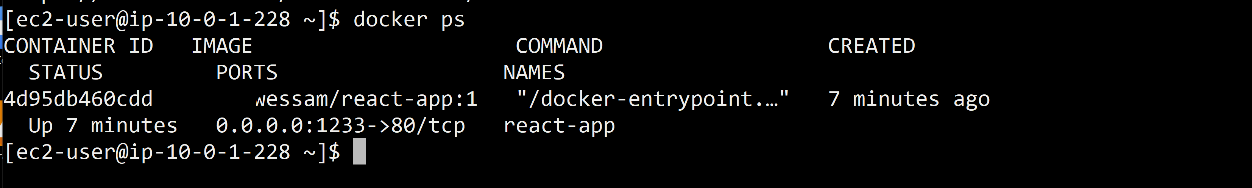
* + You will see the different steps of your Pipeline



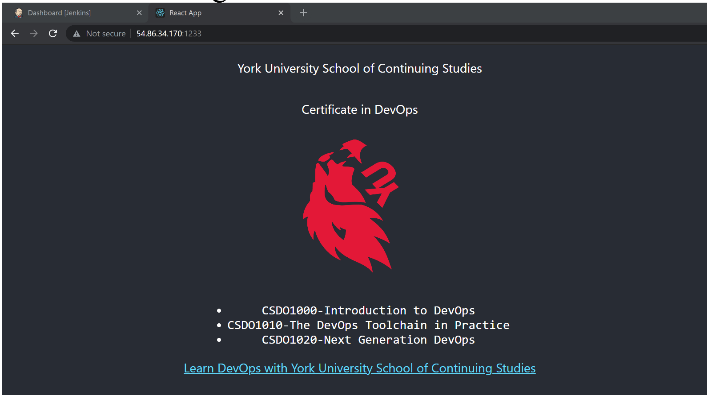
* + Click on the green box under Check HTTP Response and Select Logs. You will be able to see the status of your HTTP response (200)



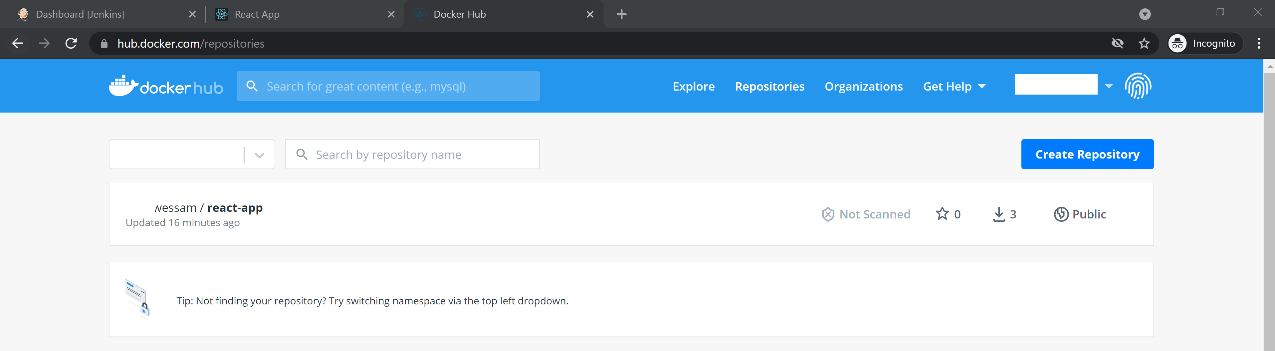
* + You could log in to the server and run docker ps to check your running container



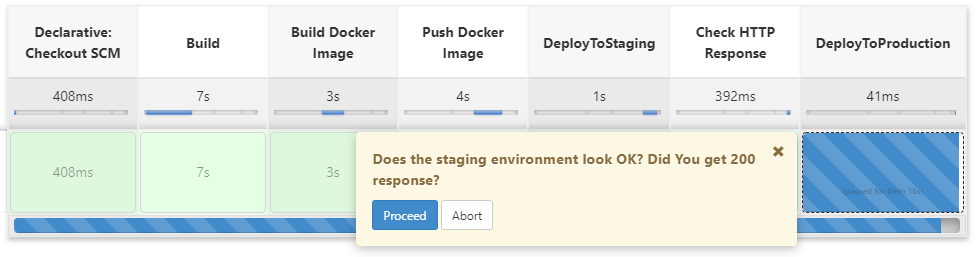
* + Go to the public IP of your server and port 1233 <Public IP>:1233. The sample react application should be running.

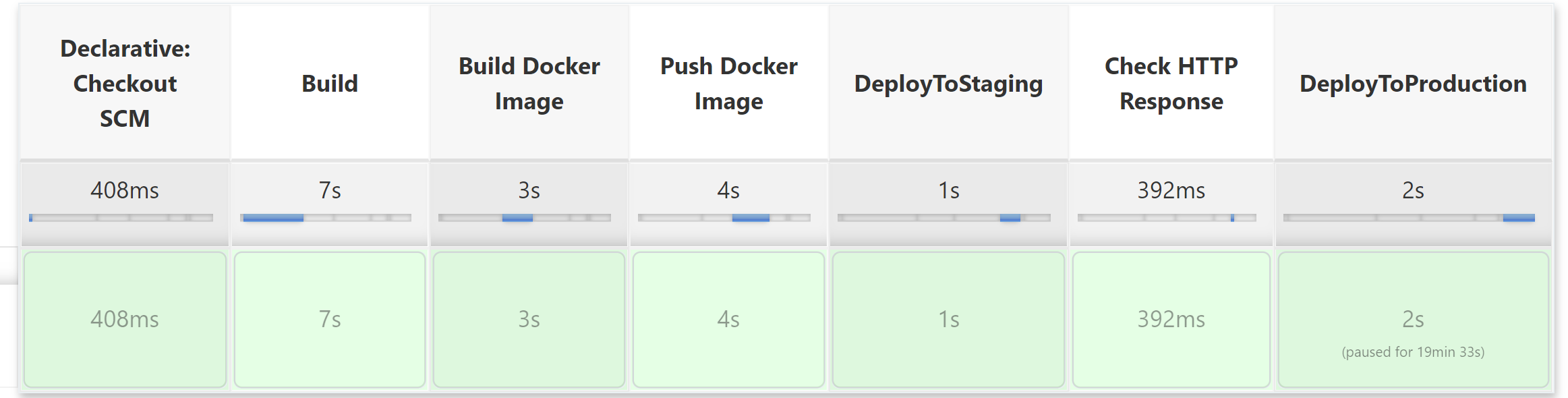


* + Go to your Docker Hub; you should be able to see the pushed and pulled image under Repositories

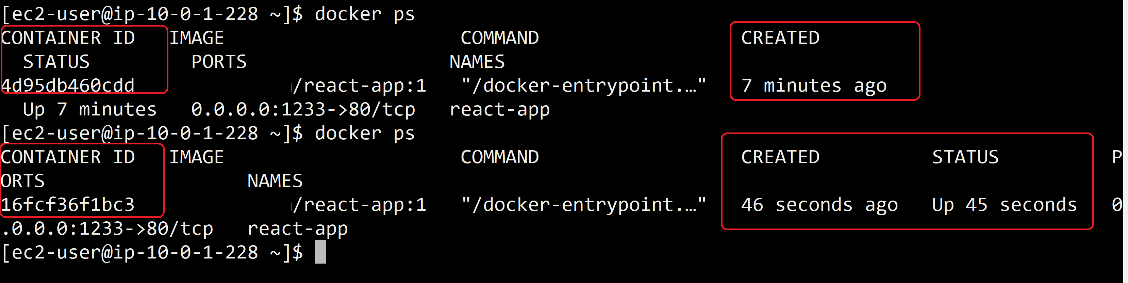


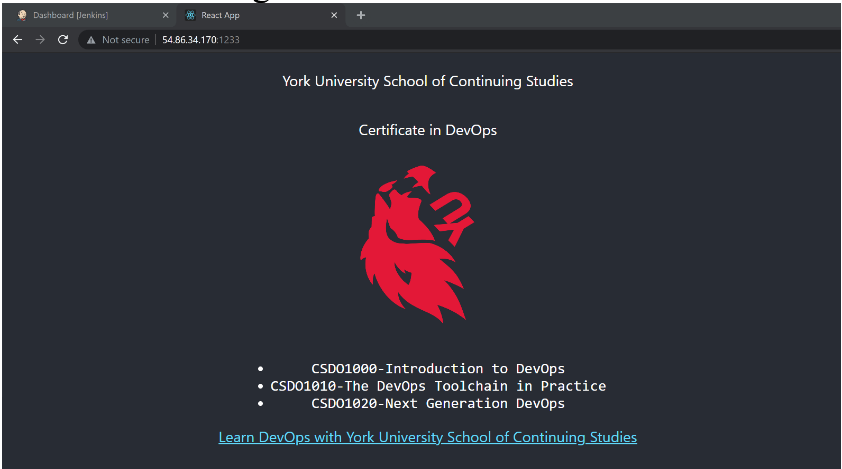
* + Now back to Jenkins and click on the blue box under DeployToProduction and Select Proceed.





* + Now run docker ps again to check your running container. Notice the image deployed to your production as you can see the different Container ID and Time. In our case, we are using one server for both staging and production.

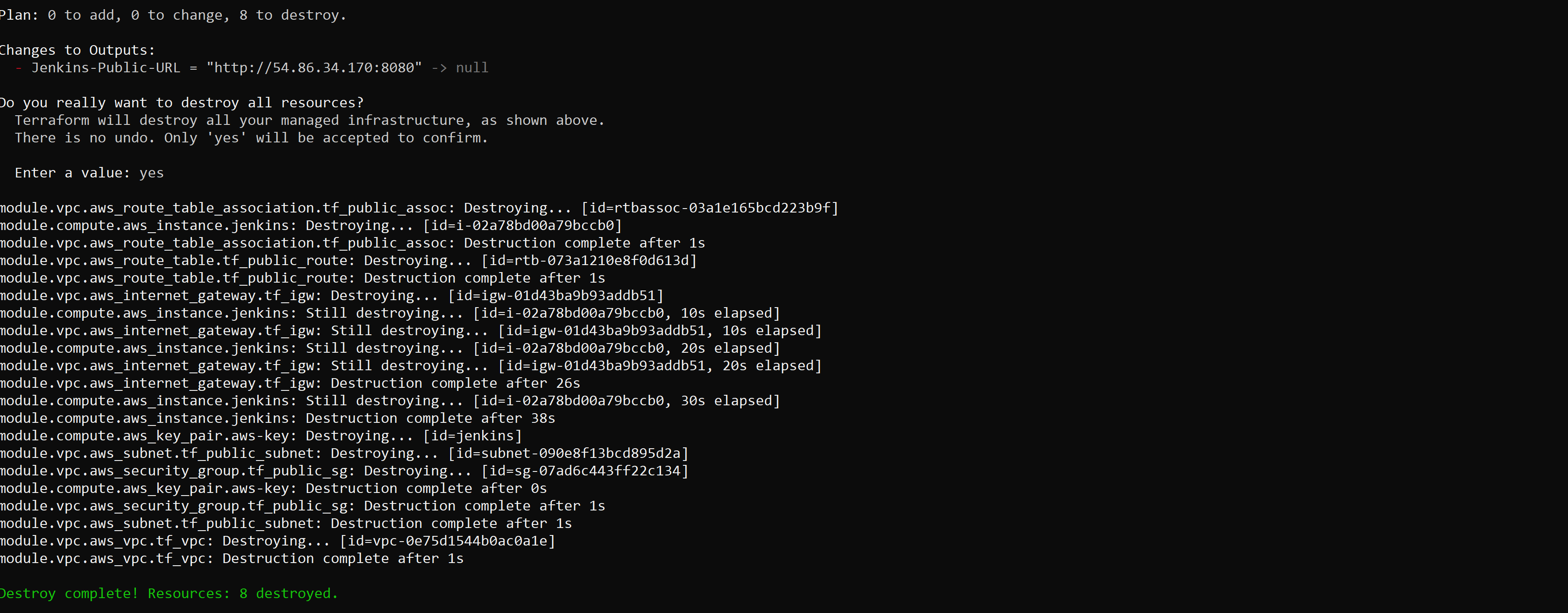


* + Again, go to the public IP of your server and port 1233 <Public IP>:1233. The sample react application should be running.
  + You could also view the Console Output for more details.



## **Step 16: Terraform destroy**

* + Once you are done with the lab, you must destroy the provisioned infrastructure resources to avoid any risks of encountering costs in the future.
  + terraform destroy: Destroy terraform managed infrastructure. You must confirm by tying yes



# Note

Do not delete the keys you generated on your local machines. You will need it for subsequent labs.

# Conclusion

Congrats on creating and managing your continuous integration and delivery with Jenkins’s Pipeline!