

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

## [Solution](#)

### Description

Demonstrate the continuous integration and delivery by building a Docker Jenkins Pipeline.

## Problem Statement Scenario:

You are a DevOps consultant in AchiStar Technologies. The company decided to implement DevOps to develop and deliver their products. Since it is an Agile organization, it follows Scrum methodology to develop the projects incrementally. You are working with multiple DevOps Engineers to build a Docker Jenkins Pipeline. During the sprint planning, you agreed to take the lead on this project and plan on the requirements, system configurations, and track the efficiency. The tasks you are responsible for:

- Availability of the application and its versions in the GitHub
  - Track their versions every time a code is committed to the repository
- Create a Docker Jenkins Pipeline that will create a Docker image from the Dockerfile and host it on Docker Hub
- It should also pull the Docker image and run it as a Docker container
- Build the Docker Jenkins Pipeline to demonstrate the continuous integration and continuous delivery workflow

Company goal is to deliver the product frequently to the production with high-end quality.

### **You must use the following tools:**

- Docker: To build the application from a Dockerfile and push it to Docker Hub
- Docker Hub: To store the Docker image
- GitHub: To store the application code and track its revisions

- Git: To connect and push files from the local system to GitHub
- Linux (Ubuntu): As a base operating system to start and execute the project
- Jenkins: To automate the deployment process during continuous integration

**Following requirements should be met:**

- Document the step-by-step process from the initial installation to the final stage
- Track the versions of the code in the GitHub repository
- Availability of the application in the Docker Hub
- Track the build status of Jenkins for every increment of the project

# Docker Jenkins Pipeline for Continuous Integration and Delivery

## [Repository](#)

If you are forking my repository, you will need to adjust the Jenkinsfile and Dockerfile according to Step 4 below. You will also need to get your Dockerhub access tokens for Jenkins.

All commands were run in the following ubuntu docker container, which means that the commands won't necessarily work in an ubuntu desktop, but it could definitely be adapted.

```
$ docker pull ubuntu:20.04
$ docker run -it -v /var/run/docker.sock:/var/run/docker.sock -p 8080:8080 --entrypoint=/bin/bash
--name test-u1 ubuntu:20.04
```

“-v /var/run/docker.sock:/var/run/docker.sock” - tells containers to create docker containers alongside the current container, not within the docker container

## Step 1: Install Required Software

### 1. Install Git

```
$ sudo apt update
$ sudo apt install -y git
```

If “sudo” command is not found

```
$ apt update
$ apt install sudo
```

Verify that git is installed. (version shouldn't need to match)

```
$ git --version
```

```
[root@dccffbf81c7:~# git --version
git version 2.25.1
```

### 2. Install Docker

```
$ sudo apt update
$ sudo apt install -y docker.io
```

Verify docker is installed

```
$ docker --version
```

```
root@95e847e3d9d4:~# docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
```

#### 4. Install Java

```
$ sudo apt-get install -y openjdk-11-jdk
```

Follow the on-screen instructions

Verify installation

```
$ java --version
```

```
root@dccffbf81c7:~# java --version
openjdk 11.0.20.1 2023-08-24
OpenJDK Runtime Environment (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04)
OpenJDK 64-Bit Server VM (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04, mixed mode, sharing)
```

#### 5. Install Jenkins

```
$ sudo apt install wget
$ apt-get update && apt-get install -y gnupg
$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
$ echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
$ sudo apt-get update
$ sudo apt-get -y install jenkins
```

## Step 2: Configure Jenkins

1. Start Jenkins service:

```
$ sudo service jenkins start
```

Why not `systemctl jenkins start`? This is related to the commands being run in a docker container. [Related article](#)

2. Access Jenkins on your browser by navigating to `http://localhost:8080`.

3. Retrieve the initial Jenkins password

```
$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
[root@1d9f5e23cef4:~# service jenkins start
* Starting Jenkins Automation Server jenkins
Setting up max open files limit to 8192

[root@1d9f5e23cef4:~# cat /var/lib/jenkins/secrets/initialAdminPassword
fbc54e0120b1450b937589453a7ace4f
```

4. Copy and paste the text from the previous command and paste it into the “Administrator password” prompt in the Jenkins browser. “Continue”

Getting Started

## Unlock Jenkins

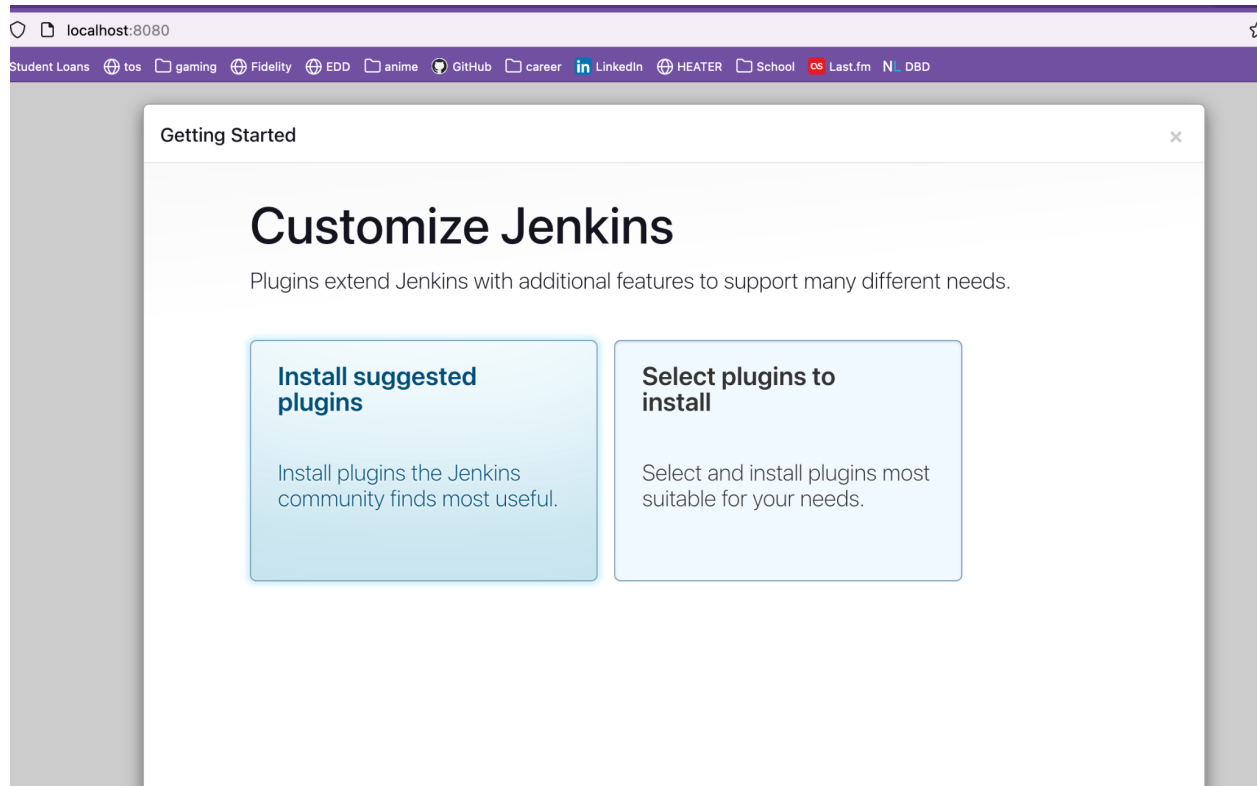
To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

5. Follow the on-screen instructions to complete the Jenkins setup, select “Install suggested plugins”



6. Fill out the following prompts. "Save and Continue". Remember your username and password for future access

# Create First Admin User

Username

admin

Password

.....

Confirm password

.....

Full name

admin

E-mail address

asdfasdfs@gmail.com

! Invalid e-mail address

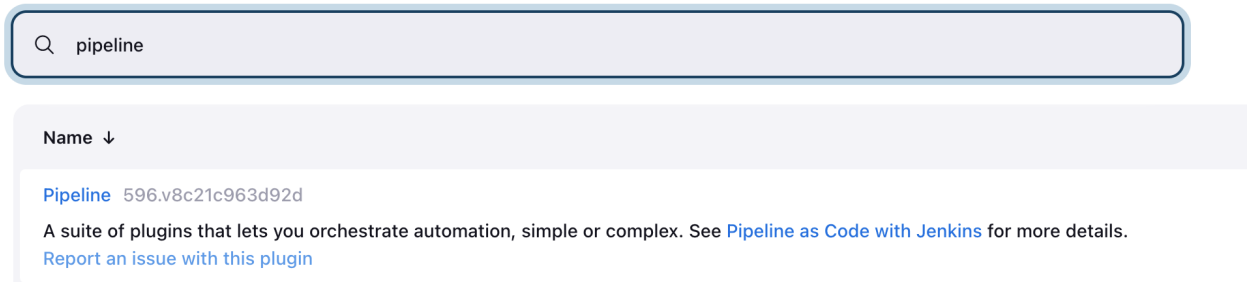
7. "Save & Continue" for localhost:8080.

## Step 3: Install Plug-Ins

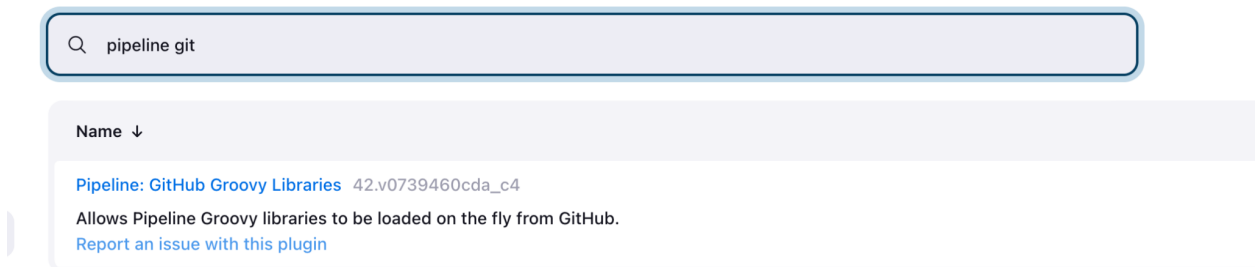
1. Inside Jenkins, click Manage Jenkins > System Configuration - Plugins > Available Plugins

The screenshot shows the Jenkins 'Manage Jenkins' interface. On the left is a sidebar with navigation links: '+ New Item', 'People', 'Build History', 'Manage Jenkins' (selected), 'My Views', 'Build Queue' (showing 'No builds in the queue.'), and 'Build Executor Status' (showing '1 Idle' and '2 Idle'). The main content area is titled 'Manage Jenkins' and includes a search bar. Below the title are three yellow warning boxes: 1) 'Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.' with buttons 'Set up agent', 'Set up cloud', and 'Dismiss'. 2) 'Jenkins URL is empty but is required for the proper operation of many Jenkins features like email notifications, PR status update, and environment variables such as BUILD\_URL. Please provide an accurate value in Jenkins configuration.' with a 'Dismiss' button. 3) 'Java 11 end of life in Jenkins' with buttons 'More info' and 'Ignore'. Below these warnings is the 'System Configuration' section, which contains four cards: 'System' (Configure global settings and paths.), 'Tools' (Configure tools, their locations and automatic installers.), 'Nodes' (Add, remove, control and monitor the various nodes that Jenkins runs on.), and 'Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins.).

2. Search for “Pipeline” and click the checkmark and “Install”



3. Search for “Pipeline git” and click the checkmark for “Pipeline: GitHub Groovy Libraries” and “Install”



4. To confirm installation, check “Installed Plugins” and make sure that “Pipeline” and “Pipeline: GitHub Groovy Libraries” are there



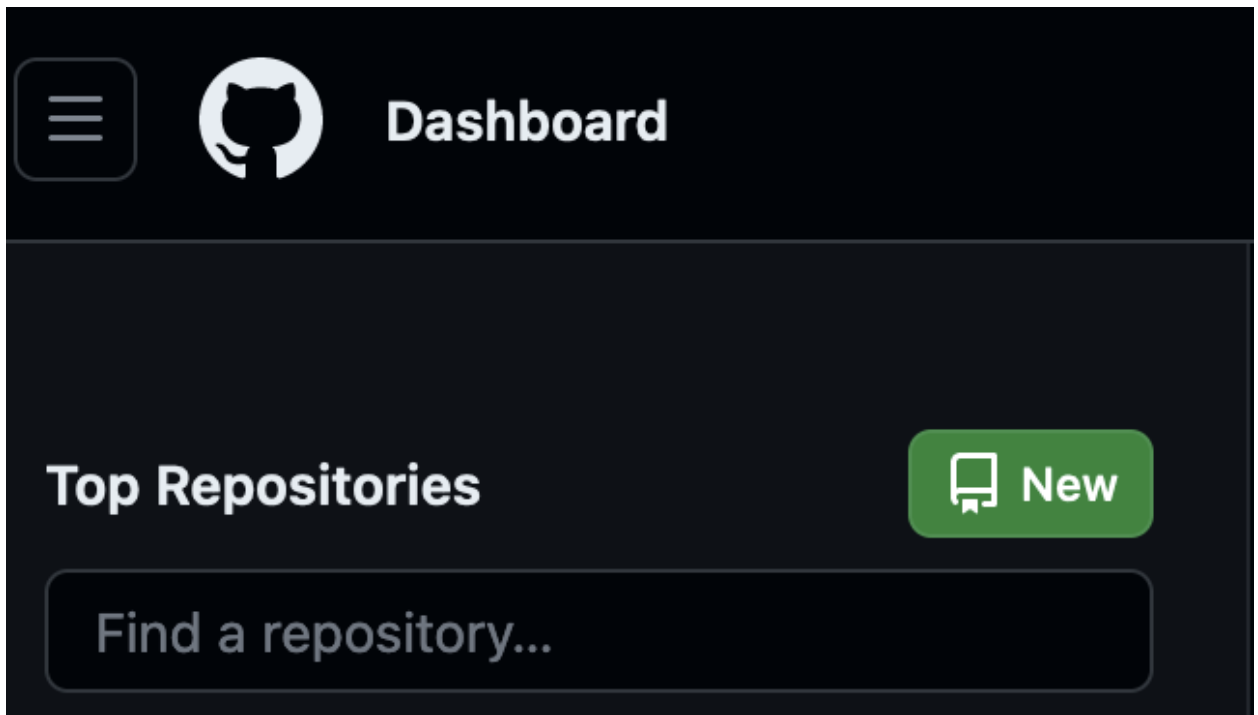
5. Repeat with the following plugins: “Docker”, “Github Integration”, “Docker Pipeline”
6. Go to ”Installed Plugins” and verify the following plugins are installed: Pipeline, Pipeline: GitHub Groovy Libraries, Docker plugin, Docker Pipeline, Github Integration plugin

## Step 4: Create a GitHub Repository

1. Navigate to [github.com](https://github.com) and Create an account if you don't have one

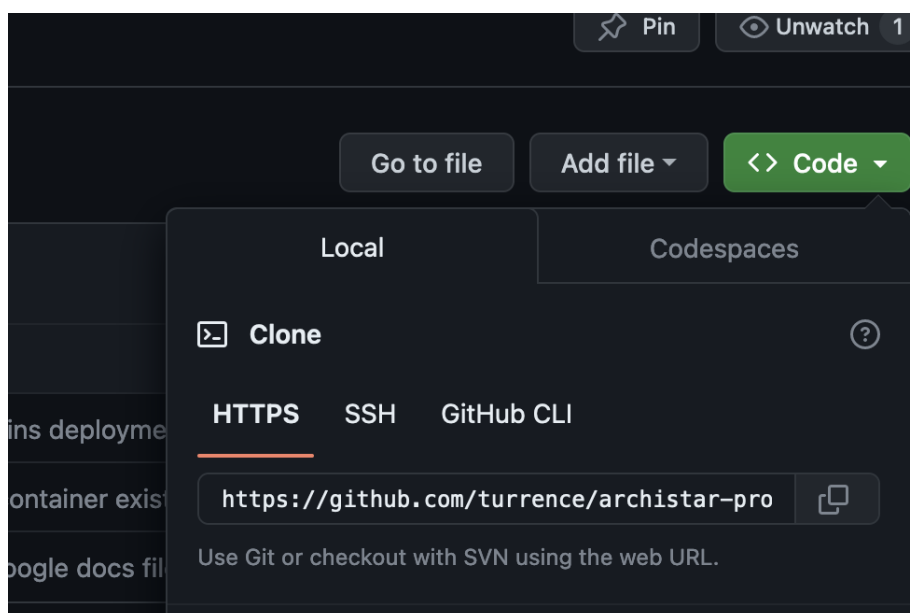


2. Click “New” to create a new GitHub Repository



3. Give the repository a name and click “Create Repository”: give it a name and add a README.
4. Navigate to the repository and go to Code. Under the HTTPS tab copy the link and paste it into the git clone command

```
$ git clone <copied_https_github_repo_link>  
$ cd <repository_name>
```



5. To create a simple web app, follow the steps in this [react tutorial](#). Verify that the web app is working, by running `npm start` and navigating to localhost:3000 in your web browser.

\*The verification may be more complex if you are running it in a docker container, you can verify the web app is working in a local dev environment\*

6. In the root directory of your repository, create a file “Dockerfile” with the following contents: Replace “my-react-app” with what you named the app you created in the previous step.

```
...  
# Use an official Node runtime as a parent image  
FROM node:18-alpine  
  
# Set the working directory in the container  
WORKDIR /my-react-app/  
  
COPY my-react-app/public/ /my-react-app/public  
COPY my-react-app/src/ /my-react-app/src  
COPY my-react-app/package.json /my-react-app/  
  
RUN npm install  
  
# Expose the port the app runs on  
EXPOSE 3000  
  
# Define the command to run the application  
RUN npm run build  
RUN npm install -g serve  
  
CMD ["serve", "-s", "build"]  
...
```

7. In the root directory of your repository, create a file “Jenkinsfile” with the following contents:

```
...  
pipeline {  
    agent any  
  
    environment {  
        DOCKERHUB_CREDENTIALS = credentials('docker-credentials')  
    }  
}
```

```
stages {
  stage('Build and Push') {
    steps {
      script {
        // Use withCredentials to securely pass Docker Hub credentials
        withCredentials([usernamePassword(credentialsId: 'docker-credentials',
passwordVariable: 'DOCKERHUB_PASSWORD', usernameVariable:
'DOCKERHUB_USERNAME')]) {
          // Build Docker image
          sh 'docker build -t your_dockerhub_username/your_image_name .'

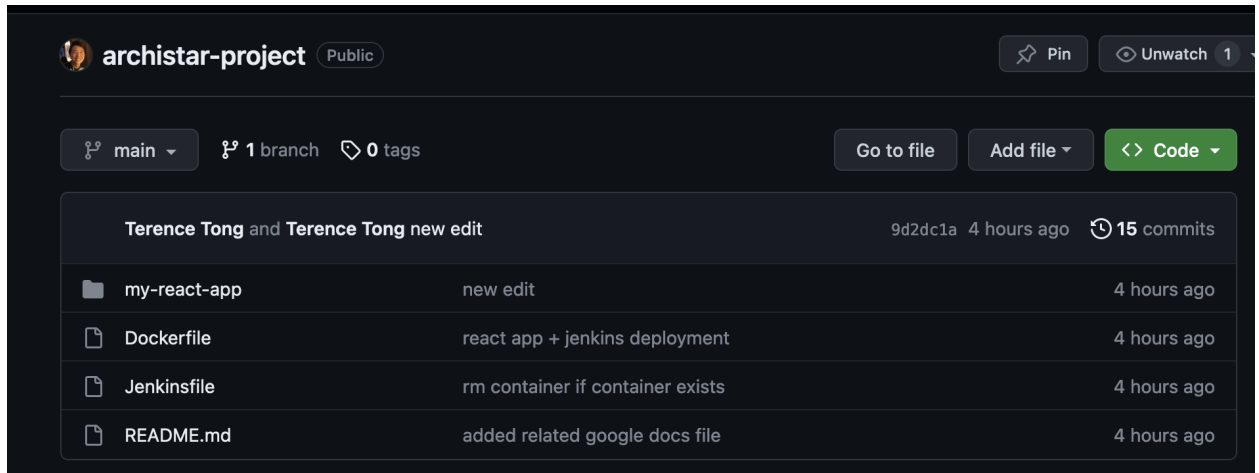
          // Login to Docker Hub
          sh 'echo $DOCKERHUB_PASSWORD | docker login -u
$DOCKERHUB_USERNAME --password-stdin'

          // Push Docker image to Docker Hub
          sh 'docker push your_dockerhub_username/your_image_name'
        }
      }
    }
    stage('Deploy') {
      steps {
        script {
          sh 'docker rm -f your_container_name || true'
          docker.image('your_dockerhub_username/your_image_name').run('-d -it -p
3000:3000 --name your_container_name')
        }
      }
    }
  }
}
```

Replace `your\_dockerhub\_username`, `your\_image\_name`, and `your\_container\_name` with your Docker Hub username, desired image name, and container name. If you do not have a DockerHub username, you can create an account [here](#). Make sure you aren't missing any quotation marks.

8. Push the changes to your repository. Run the following commands in the root of your repository.  
\$ git add .  
\$ git commit -m "jenkinsfile + dockerfile + simple react app"  
\$ git push

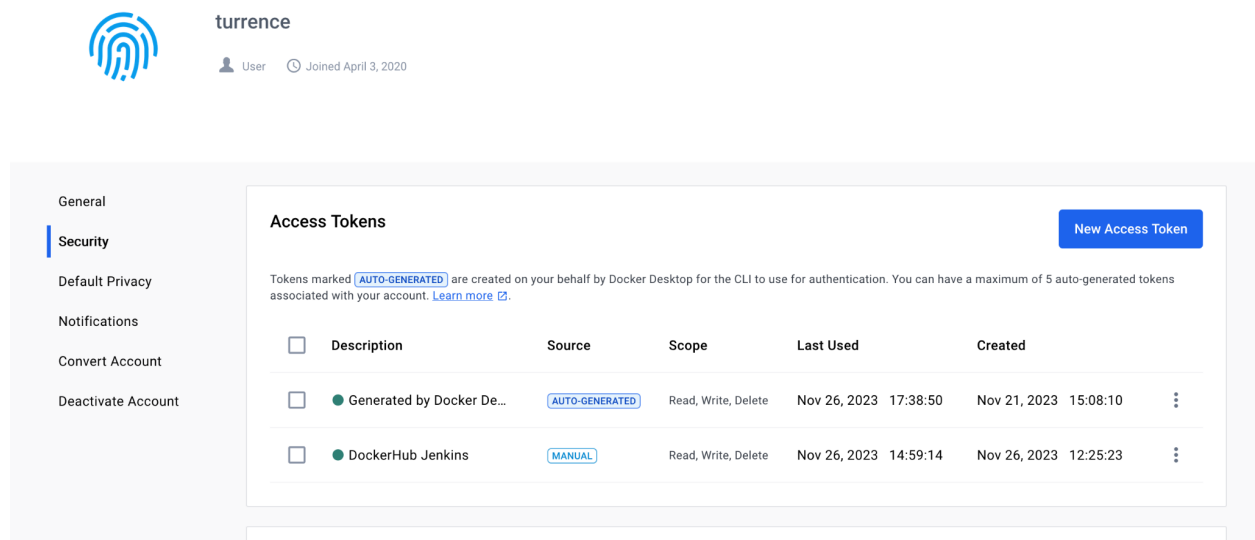
9. Verify the changes to your repository on GitHub:



## Step 5: Create and use Tokens from Docker Hub in Jenkins


Create an account on [Docker Hub](#) if you don't have one.

1. Once logged in, Account Settings > Security > New Access Token. Give the access token a name. And “Save & Copy” the access token. (Used in Step 5-6 below)



2. Go to Jenkins > Manage Jenkins > Security - Credentials. Under “Stores scoped to Jenkins” click the (global) under the Domains column

## Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

Icon: **S** M L

3. Click Add Credentials
4. Under Kind choose “Username with Password”
5. Under Username, put your dockerhub username
6. Under Password, paste your dockerhub access token from Step 5-1
7. Under ID, name it “docker-credentials” (note the relationship to the Jenkinsfile) (not shown in image below)

**New credentials**

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

username

☐ Treat username as secret ?

Password ?

.....

ID ?

Description ?

Create

8. Click "Create"

## Step 6: Create a Jenkins Pipeline Job

1. Inside Jenkins, click on "New Item."
2. Choose "Pipeline" and provide a name for your job. Select "OK"
3. In the Pipeline section, select "Pipeline script from SCM" as the Definition.
4. Choose Git as the SCM, and enter your GitHub repository URL created in Step 4.
5. Under Branches to build, make sure the branch name matches the branch that shows up when you visit your repository on github.

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/turrence/archistar-project

Credentials ?

- none -

+ Add

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

Add Branch

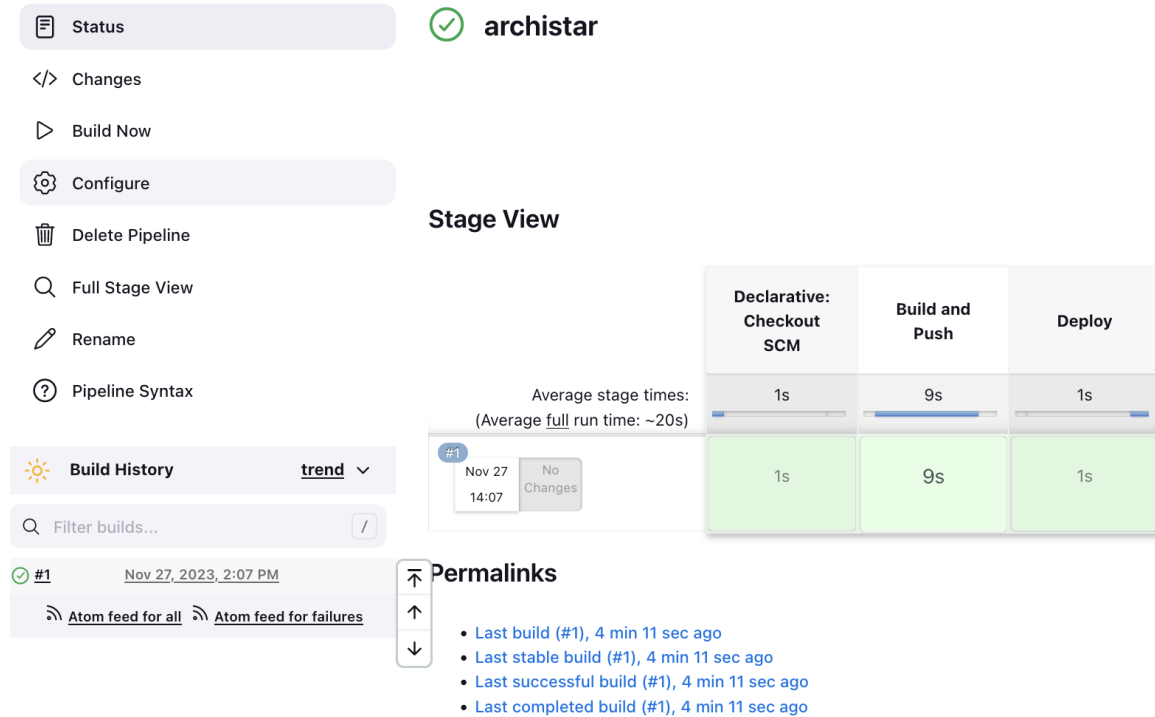
6. Save the changes.

## Step 7: Run the Jenkins Pipeline

1. In the terminal, execute the following command

```
$ chmod 666 /var/run/docker.sock
```

## 2. Manually trigger the Jenkins job to run the pipeline. Using the “Build Now” button



The screenshot shows the Jenkins interface for a job named 'archistar'. On the left, a sidebar contains navigation links: Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, and Pipeline Syntax. The main area displays the 'Stage View' of the pipeline. It shows three stages: 'Declarative: Checkout SCM' (1s), 'Build and Push' (9s), and 'Deploy' (1s). Below the stages, a table shows the duration of each stage for the current build (#1). The 'Build History' section on the left shows a single build (#1) from Nov 27, 2023, at 2:07 PM. The 'Permalinks' section on the right provides links to the last build, last stable build, last successful build, and last completed build, all of which are the same build (#1) from 4 minutes ago.

**Stage View**

Average stage times:  
(Average full run time: ~20s)

Stage	Duration
Declarative: Checkout SCM	1s
Build and Push	9s
Deploy	1s

**Build History** **trend** **Filter builds...**

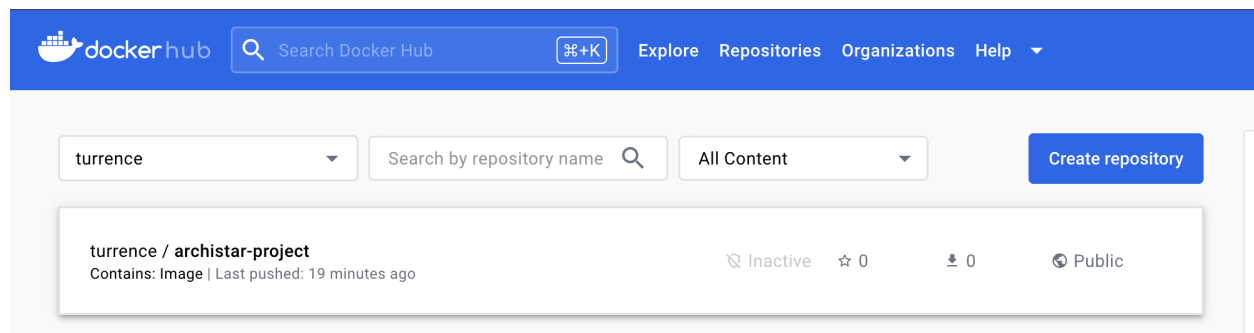
**#1** Nov 27, 2023, 2:07 PM

**Permalinks**

- Last build (#1), 4 min 11 sec ago
- Last stable build (#1), 4 min 11 sec ago
- Last successful build (#1), 4 min 11 sec ago
- Last completed build (#1), 4 min 11 sec ago

## Step 8: Verify

1. Check Jenkins for the build status and console output.
2. Verify the Docker image on Docker Hub.



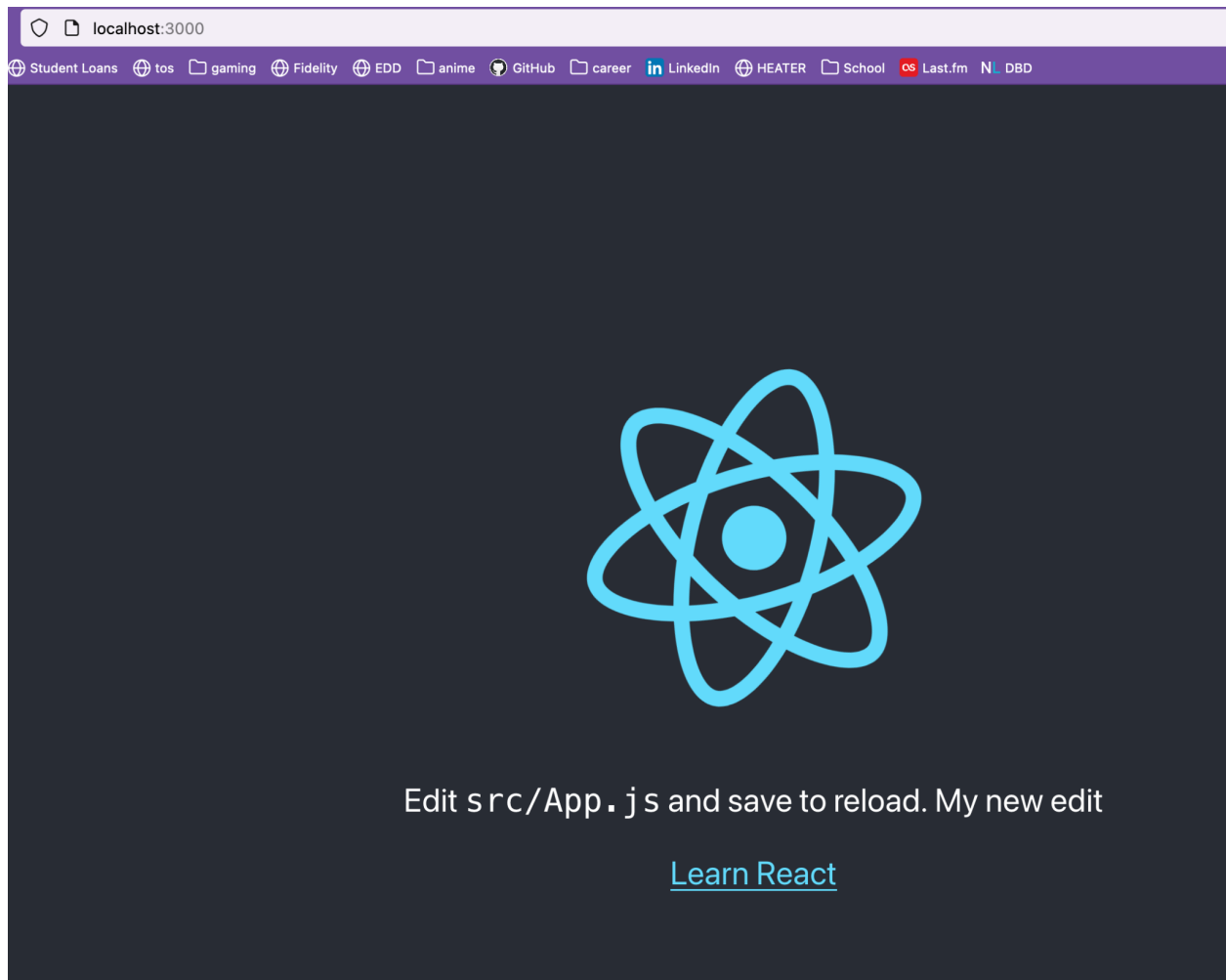
The screenshot shows the Docker Hub interface for a repository named 'turrence/archistar-project'. The repository is currently 'Inactive' and has 0 stars and 0 downloads. It is a public repository. The page also shows the repository name 'turrence' and a search bar for 'Search by repository name'. The repository contains an image and was last pushed 19 minutes ago.

**turrence / archistar-project**

Contains: Image | Last pushed: 19 minutes ago

Inactive ☆ 0 📄 0 🌐 Public

3. Access your application using the app port (e.g., `localhost:3000`) to ensure it's running.



## Step 9: Automate the Deployment


The following steps allow you to automate the deployment process to a set interval.


1. Go to the pipeline you create and go to “Configure”
2. General - Poll Triggers > Check Poll and in Schedule put the following: H/10 \* \* \* \*




Dashboard > archistar-pipeline > Configuration

### Configure

 General

 Advanced Project Options

 Pipeline

### Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☐ GitHub Branches

☐ GitHub Pull Requests ?

☐ GitHub hook trigger for GITScm polling ?

☒ Poll SCM ?

Schedule ?

H/10 \* \* \* \*

Would last have run at Sunday, November 26, 2023 at 1 Time.

☐ Ignore post-commit hooks ?

☐ Quiet period ?

☐ Trigger builds remotely (e.g., from scripts) ?

3. Click “Save” and now Jenkins will check your GitHub Repository every 10 mins for changes

You have now set up a Jenkins CI/CD that builds images and pushes them to dockerhub. To see changes to your web app, make changes to the react app in your repository and commit and push those changes. Then head to Jenkins and either wait for the 10 minutes period to pass for the automatic deployment, or hit the “Build Now” button in the pipeline. Once the build has finished you should see the changes reflected at `localhost:3000`.