Configure Jenkins to build docker image Successfully deploy spring boot app as a Docker container using the Jenkins Pipeline. At the last stage in the pipeline, push the image to the docker hub with the appropriate tag.

To configure Jenkins to build a Docker image, deploy a Spring Boot app as a Docker container, and push the image to Docker Hub using a Jenkins Pipeline, follow these steps:

# Step1 - Prerequisites:

1. Ensure you have Docker and Jenkins installed and running.
2. Install necessary Jenkins plugins, such as "Github Integration","Docker" and "Pipeline," if not already installed.

# Step 2 - Configure Docker and Github Credentials :

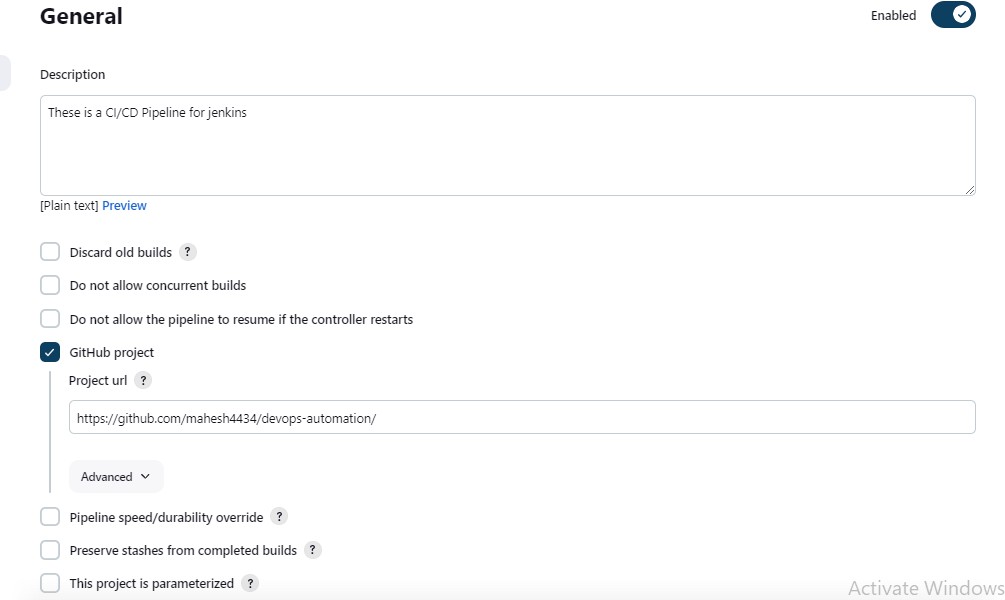
1. You can define your docker hub credentials into your pipeline code or
2. In Jenkins, navigate to "Credentials" in the left-hand menu.
3. Click on "Global credentials" and then "Add Credentials."
4. Choose "Username with password" as the kind.
5. Enter your Docker Hub username and password, and provide an ID (e.g., "docker-hub-credentials").
6. Click "OK" to save your Docker Hub credentials.
7. Same for the process for Github clicking on the github tab and enter all required information

8)

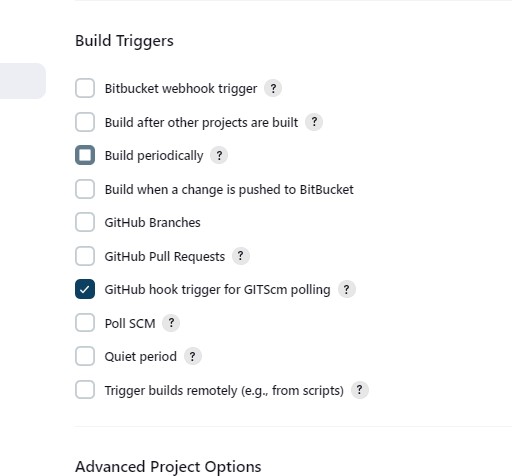
# Step 3 - Set up a Jenkins Pipeline for Your Spring Boot App:

1. Create a new Jenkins Pipeline job or open an existing one.
2. In the Pipeline section, click on pipeline and enter your pipeline name
3. After that click on configure button and follow the steps as below:

**Step 1) Add Description and your Github project link**



# Step 2) Enable Github Hook trigger



**Step 3) Start writing Pipeline Script**

Here is my pipeline code:

**pipeline { agent any tools {**

**maven 'Maven-Install'**

**}**

**stages {**

**stage('Build Maven') { steps {**

**script {**

**// Checkout the code from the Git repository checkout([**

**$class: 'GitSCM', branches: [[name: 'main']],**

**userRemoteConfigs: [[url: 'https://github.com/mahesh4434/devops-automation']]**

**])**

**// Build the Maven project sh 'mvn clean install'**

**}**

**}**

**}**

**stage('Docker Build and Run') { steps {**

**sh 'docker stop node-app-todo || true' sh 'docker rm node-app-todo || true'**

**sh 'docker rmi mahesh4434/myapp || true' sh 'docker build -t mahesh4434/myapp .'**

**sh 'docker run -d -p 8081:8081 --name node-app-todo mahesh4434/myapp'**

**}**

**}**

**stage('push') { steps {**

**script {**

**withCredentials([string(credentialsId: 'dockerhubpwd', variable: 'dockerhubpwd')]) { sh "docker login -u** [**maheshchaudhari511@gmail.com**](mailto:maheshchaudhari511@gmail.com) **-p ${dockerhubpwd}"**

**}**

**sh "docker push mahesh4434/myapp"**

**}**

**}**

**}**

These are the Pipeline Script Into these

# Stage1: Build Maven Stage:

* 1. This is the first stage in the pipeline and is responsible for building a Maven-based Java project.
  2. checkout: This step checks out the code from a Git repository located at 'https://github.com/mahesh4434/devops-automation'.
  3. sh 'mvn clean install': This step executes the Maven build command to clean and install the project.

# Stage 2: Docker Build and Run

* 1. stage('Docker Build and Run'): This stage is responsible for building a Docker image and running a Docker container for the application.
  2. sh commands are used to run various Docker commands:
  3. docker stop node-app-todo || true: Stops a Docker container named 'node-app-todo' if it exists.
  4. docker rm node-app-todo || true: Removes the Docker container 'node-app-todo' if it exists.
  5. docker rmi mahesh4434/myapp || true: Removes the Docker image 'mahesh4434/myapp' if it exists.
  6. docker build -t mahesh4434/myapp .: Builds a Docker image from the current directory (context) and tags it as 'mahesh4434/myapp'.
  7. docker run -d -p 8081:8081 --name node-app-todo mahesh4434/myapp: Runs a Docker container from the 'mahesh4434/myapp' image in detached mode (-d), mapping port 8081 on the host to port 8081 in the container, and naming it 'node-app-todo'.

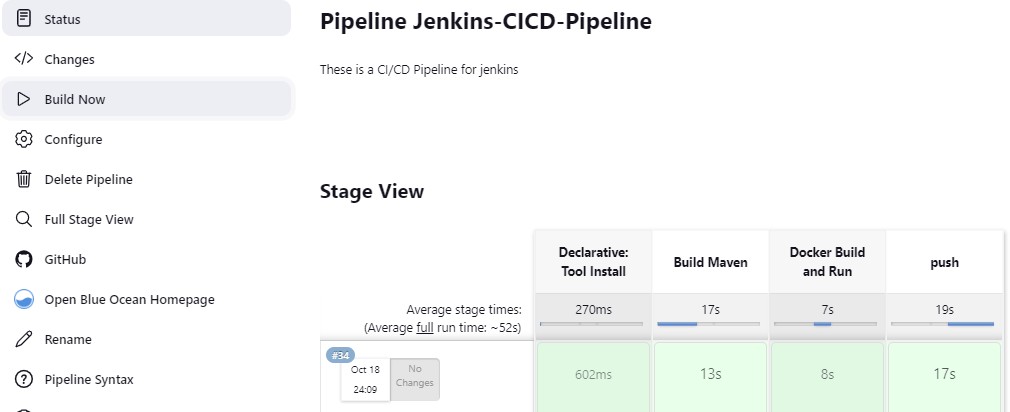
# Stage 3: Push to Docker Hub

* 1. stage('push'): This stage is responsible for pushing the Docker image to Docker Hub.
  2. withCredentials: It securely provides Docker Hub credentials stored in Jenkins as 'dockerhubpwd'.
  3. sh "docker login -u [maheshchaudhari511@gmail.com](mailto:maheshchaudhari511@gmail.com) -p ${dockerhubpwd}": Logs in to Docker Hub using the provided credentials.
  4. sh "docker push mahesh4434/myapp": Pushes the Docker image 'mahesh4434/myapp' to Docker Hub.

After completing these steps click on save button

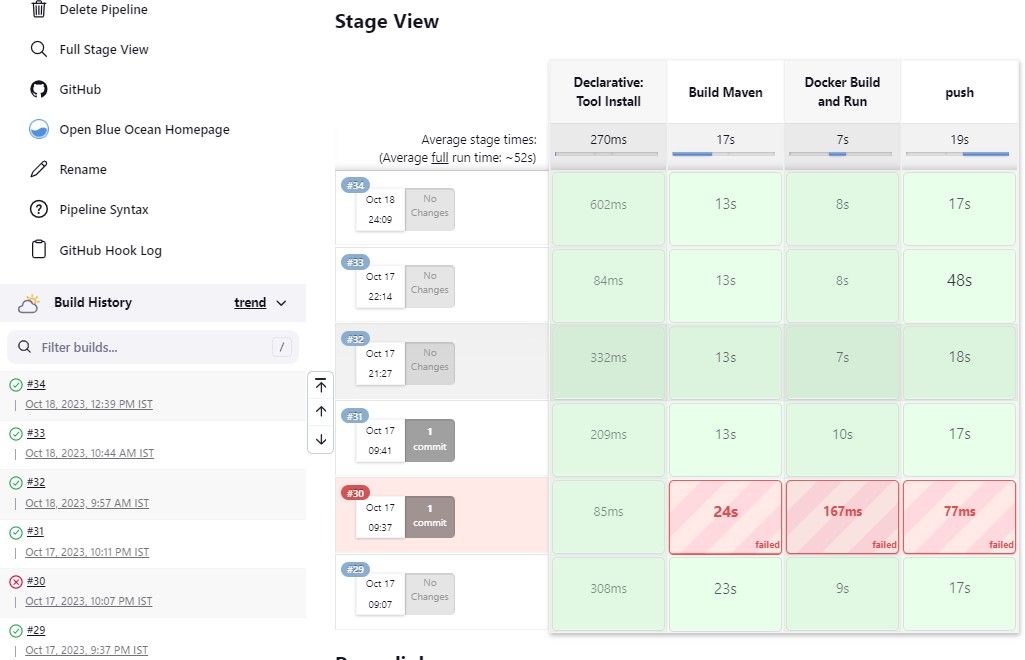
# Step 4) Build now

After clicking on your pipeline name you can get build now option at the right side of the UI



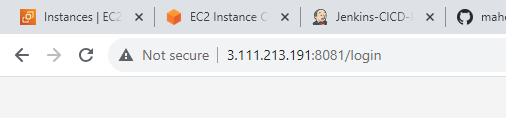
# Step 5) Monitoring the pipeline stages and console output

You can monitor your pipeline outputs and stages on the same window of build now UI



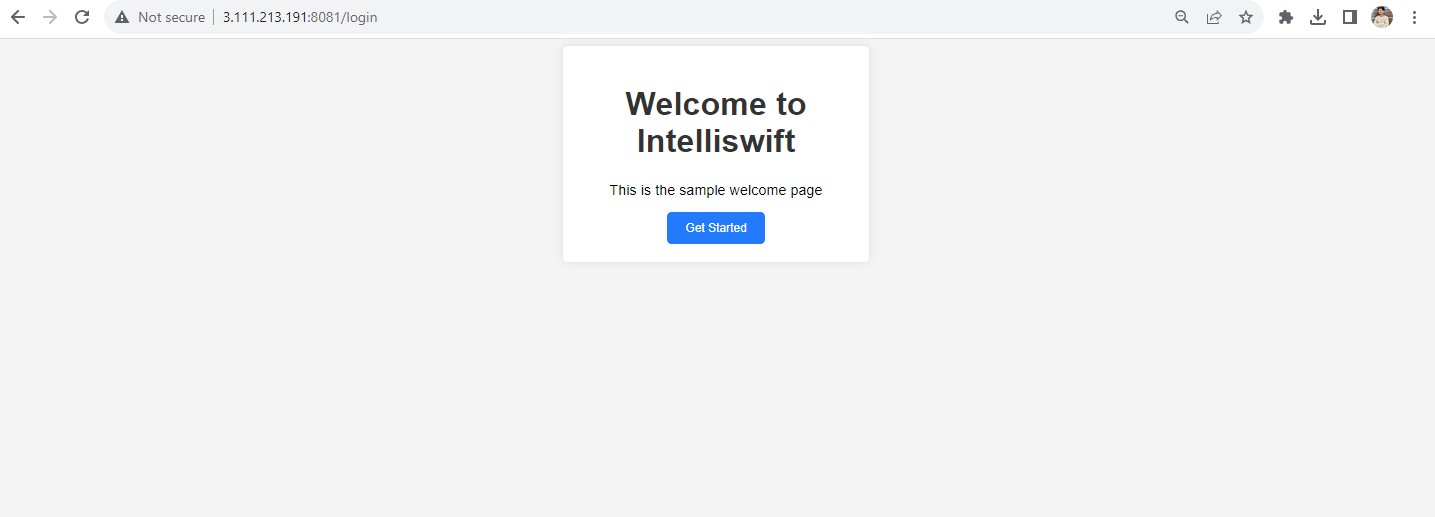
# Step 5) Accessing UI page of SpringBoot Application

After successful build of pipeline you can access spring boot UI page through publicIp of ec2 instance and mentioned port number of application into th application.properties



URL- PublicIp:port number/api

# Step 6) Rendering UI Page of SpringBoot Application



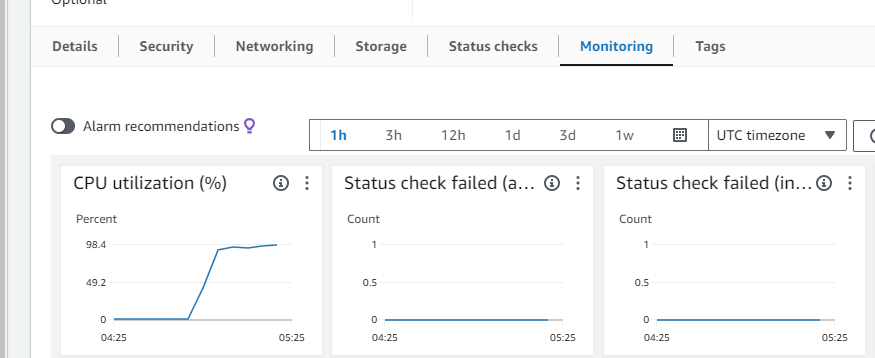
Here is my github SpringBoot project repository link

**Problems faced during Project**

# Excessive use of CPU Utilization

I created an EC2 instance on AWS using a Terraform file, and I installed Jenkins on that EC2 instance through a Terraform file. I have also written a CI/CD pipeline for the deployment of a Spring Boot project on the server. My EC2 instance's CPU utilization is at 98%, and I am unable to access Jenkins on port 8080.

This problem occurs due to t2.micro free tier instance which has only 1GB RAM for instance.



# Solution:

I created a new AWS Linux instance and used the memory swapping concept of linux to use the hard disk memory of the instance as RAM.

