

Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job

Author: Oluwaseun Osunsola

LinkedIn: <https://www.linkedin.com/in/oluwaseun-osunsola-95539b175/>

Project Repository: <https://github.com/Oluwaseunoa/DevOps-Projects/tree/main/Jenkins-Projects>

Environment & Tools: AWS EC2 (Ubuntu 22.04 LTS), Jenkins 2.462.x (LTS), GitHub

Project Overview

This project demonstrates how to automate Docker image building and container deployment using a **Jenkins Declarative Pipeline Job**. The workflow integrates **GitHub webhooks** to enable **continuous integration**, ensuring that every code push automatically triggers a Jenkins pipeline that builds a Docker image and deploys a container.

This project builds upon prior Docker foundations knowledge by translating a manual Docker workflow into a fully automated CI pipeline using Jenkins Pipeline as Code.

Technologies Used

- Jenkins (Pipeline Job)
 - Docker
 - GitHub
 - Ubuntu Server
 - NGINX
 - Shell Scripting
-

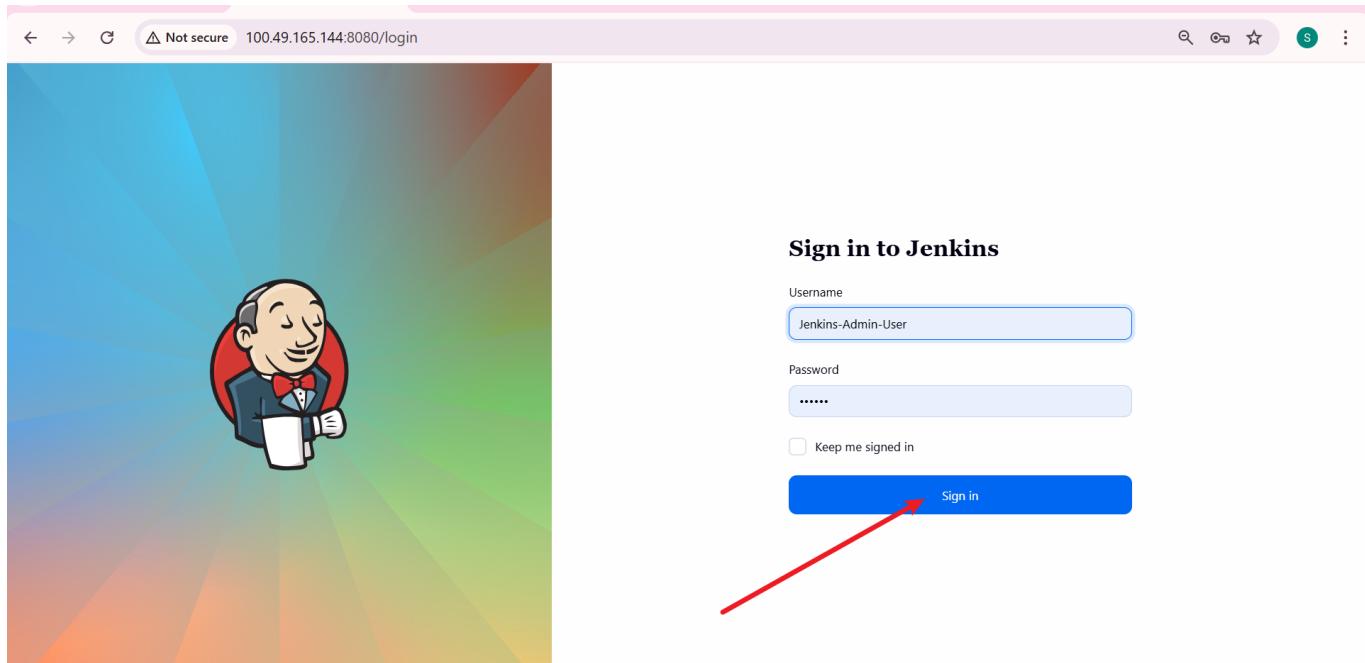
Project Prerequisites

- Jenkins installed and running on Ubuntu
 - Docker installed on the Jenkins server
 - GitHub repository created
 - Jenkins accessible via browser
 - Basic understanding of Docker and CI/CD concepts
-

Step-by-Step Project Implementation

Step 1: Sign in to Jenkins

Log in to the Jenkins dashboard using your credentials.



Step 2: Create a New Jenkins Job

Click **New Item** from the Jenkins dashboard menu.

The screenshot displays the Jenkins dashboard. At the top left, there is a 'Jenkins' icon and a 'New Item' button, which is highlighted with a red arrow. Below this, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (0/2). The main area shows a table of existing jobs:

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	Java-Pipeline	5 days 2 hr #3	N/A	8.1 sec
✓	☀️	my-first-job	1 day 3 hr #2	N/A	0.72 sec

At the bottom right, there are links for 'REST API' and 'Jenkins 2.528.3'. A red arrow also points to the 'New Item' button on the left side of the dashboard.

Step 3: Create a Pipeline Job

Enter the job name **My pipeline job**, select **Pipeline**, and click **OK**.

The screenshot shows the Jenkins 'New Item' creation interface. At the top, there's a search bar and navigation links. Below it, a form for entering an item name ('My pipeline job') and selecting an item type ('Pipeline'). The 'Pipeline' option is highlighted with a red border. At the bottom is a blue 'OK' button.

Step 4: Configure Build Trigger

Scroll to the **Build Triggers** section and check **GitHub hook trigger for GITScm polling**.

The screenshot shows the Jenkins 'Configure' screen for the 'My pipeline job'. The 'Triggers' tab is active, displaying options like 'GitHub hook trigger for GITScm polling' which is checked. The 'Pipeline' tab is also present. At the bottom are 'Save' and 'Apply' buttons.

Step 5: Configure Pipeline Script Section

Scroll to the **Pipeline** section, paste the pipeline script, and click **Pipeline Syntax**.

The screenshot shows the Jenkins Pipeline Syntax configuration page. The left sidebar has tabs: General, Triggers, Pipeline (selected), and Advanced. The main area has tabs: Definition (Pipeline script) and Script. A code editor shows Groovy pipeline code:

```

1 v pipeline {
2   agent any
3   stages {
4     stage('Connect To Github') {
5       steps {
6         checkout scmGit(
7           branches: [[name: '/main']],
8           extensions: [],
9           userRemoteConfigs: [
10             [url: 'https://github.com/RidwanAz/jenkins-scm.git']
11           ]
12         )
13       }
14     }
15   }
16 }
```

Below the code editor are two buttons: Use Groovy Sandbox and Pipeline Syntax. A red arrow points to the Pipeline Syntax button. At the bottom are Save and Apply buttons.

Step 6: Select Checkout Sample Step

Under **Sample Steps**, select **checkout: Check out from version control**.

The screenshot shows the Jenkins Snippet Generator Overview page. The left sidebar has tabs: Snippet Generator (selected), Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main area has tabs: Overview and Steps. Under Steps, a list of sample steps is shown, with **checkout: Check out from version control** highlighted by a blue box and a red arrow pointing to it. Other steps listed include archiveArtifacts, bat, build, catchError, cleanWs, deleteDir, dir, echo, emailext, emailextrecipients, error, fileExists, and findBuildScans.

Step 7: Configure Git SCM

Under **SCM**, select **Git** and paste your GitHub repository URL.

The screenshot shows the Jenkins Pipeline Syntax configuration page. On the left sidebar, there are links for Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main area is titled 'Steps' and shows a 'Sample Step' named 'checkout: Check out from version control'. Below this, under 'SCM', it is set to 'Git'. A 'Repositories' section is expanded, showing a single repository with the 'Repository URL' set to 'https://github.com/Oluwaseunoa/jenkins-scm.git'. The 'Branches to build' section is collapsed.

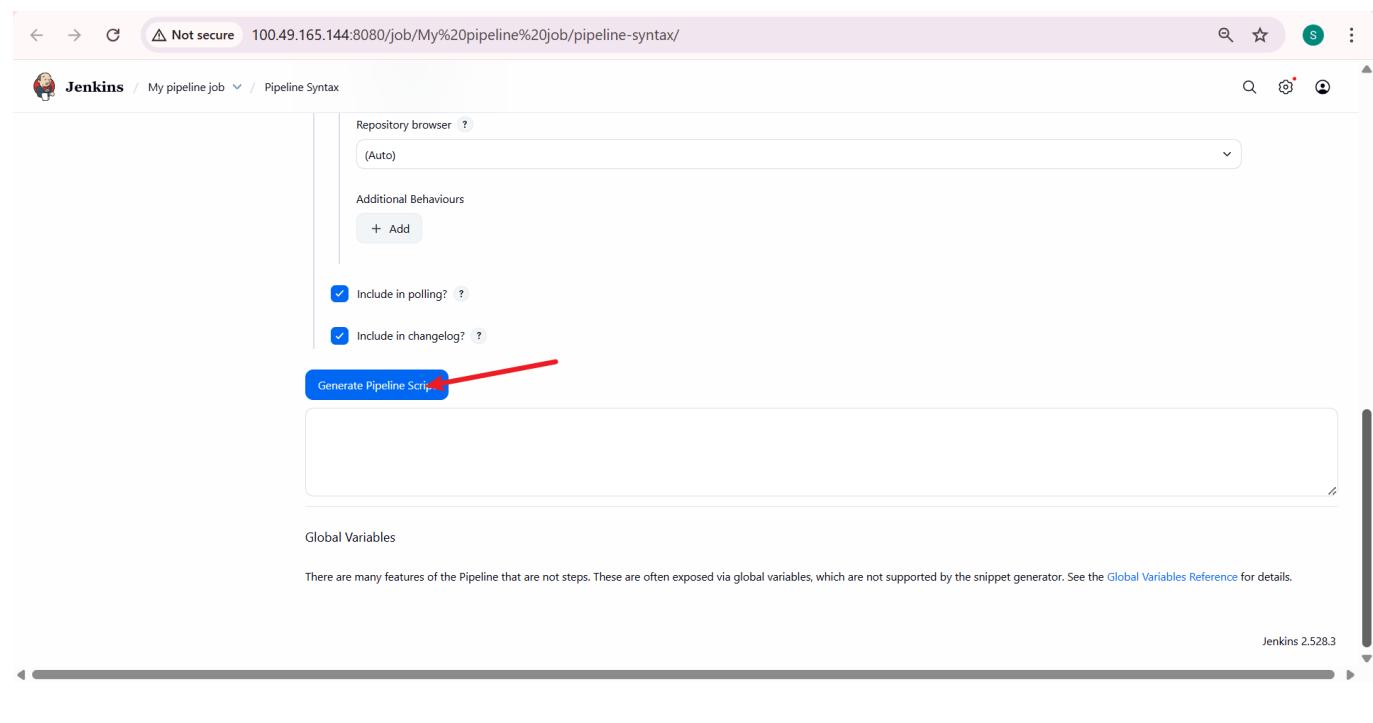
Step 8: Set Branch Specifier

Scroll and set the branch specifier to **main** (or applicable branch).

The screenshot shows the Jenkins Pipeline Syntax configuration page, similar to the previous one but with more fields visible. The 'Branches to build' section is expanded, showing a 'Branch Specifier (blank for 'any')' field containing '/main'. Other sections like 'Credentials', 'Advanced', 'Add Repository', 'Add Branch', 'Repository browser', and 'Additional Behaviours' are also visible.

Step 9: Generate Pipeline Script

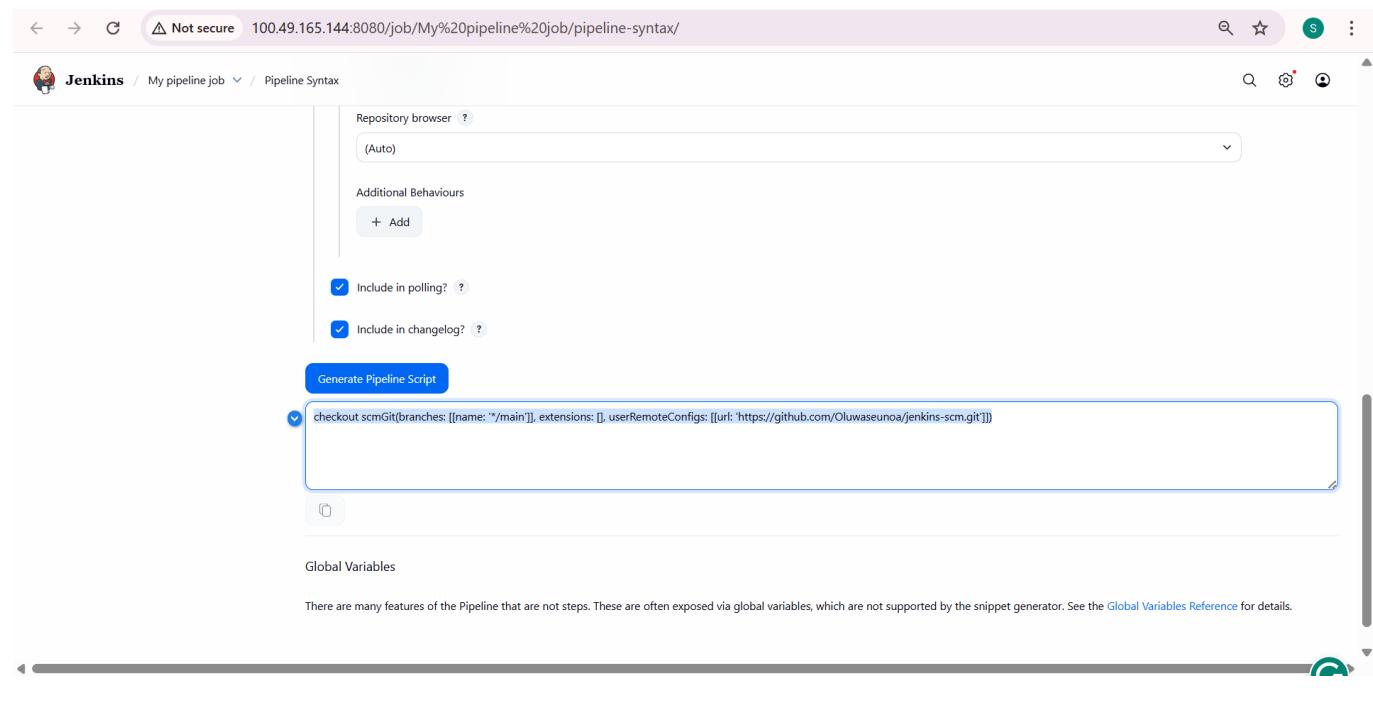
Click **Generate Pipeline Script**.



The screenshot shows the Jenkins Pipeline Syntax generator interface. At the top, it says "Not secure" and the URL "100.49.165.144:8080/job/My%20pipeline%20job/pipeline-syntax/". Below that, it says "Jenkins / My pipeline job / Pipeline Syntax". On the left, there's a sidebar with "Repository browser" set to "(Auto)", "Additional Behaviours" with a "+ Add" button, and checkboxes for "Include in polling?" and "Include in changelog?". A prominent blue button labeled "Generate Pipeline Script" is centered. A red arrow points to this button. Below the button is a large text area where the generated script will be displayed. At the bottom, it says "Global Variables" and "There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the Global Variables Reference for details." In the bottom right corner, it says "Jenkins 2.528.3".

Step 10: Copy Generated Script

Copy the generated pipeline checkout syntax.



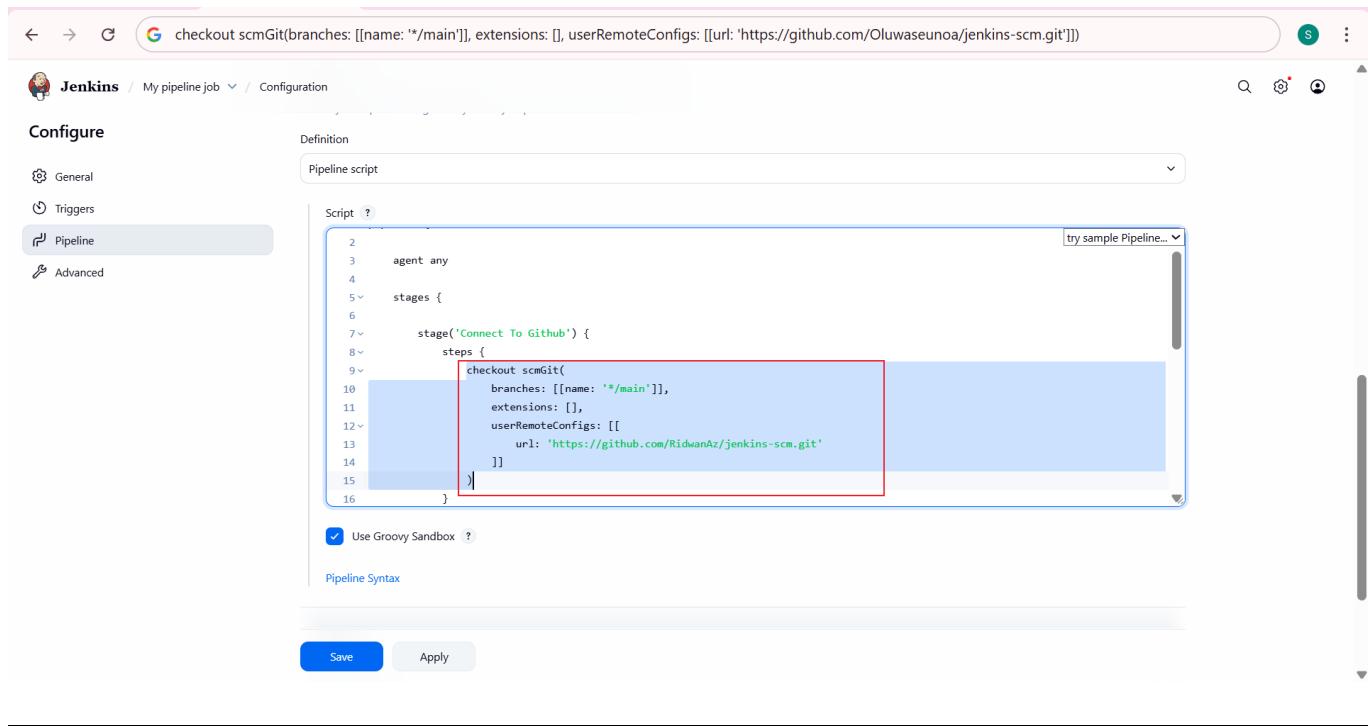
The screenshot shows the Jenkins Pipeline Syntax generator interface again. The "Generate Pipeline Script" button has been clicked, and the resulting code is displayed in a large text area. The code is:

```
checkout scmGit([branches: [[name: '/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/Oluwaseunoa/jenkins-scm.git']]])
```

 This code is highlighted with a blue box. Below the code, there's a copy icon. At the bottom, it says "Global Variables" and "There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the Global Variables Reference for details." In the bottom right corner, it says "Jenkins 2.528.3".

Step 11: Replace Checkout Section

Replace the Git checkout section of your pipeline script with the generated syntax.



Jenkins / My pipeline job / Configuration

Configure

General Triggers Pipeline Advanced

Definition Pipeline script

Script ?

```
2
3     agent any
4
5     stages {
6
7         stage('Connect To Github') {
8             steps {
9                 checkout scmGit(
10                    branches: [[name: '*/main']],
11                    extensions: [],
12                    userRemoteConfigs: [
13                        url: 'https://github.com/RidwanAz/jenkins-scm.git'
14                    ]
15                )
16            }
17        }
18    }
19
20 }
```

try sample Pipeline... ▾

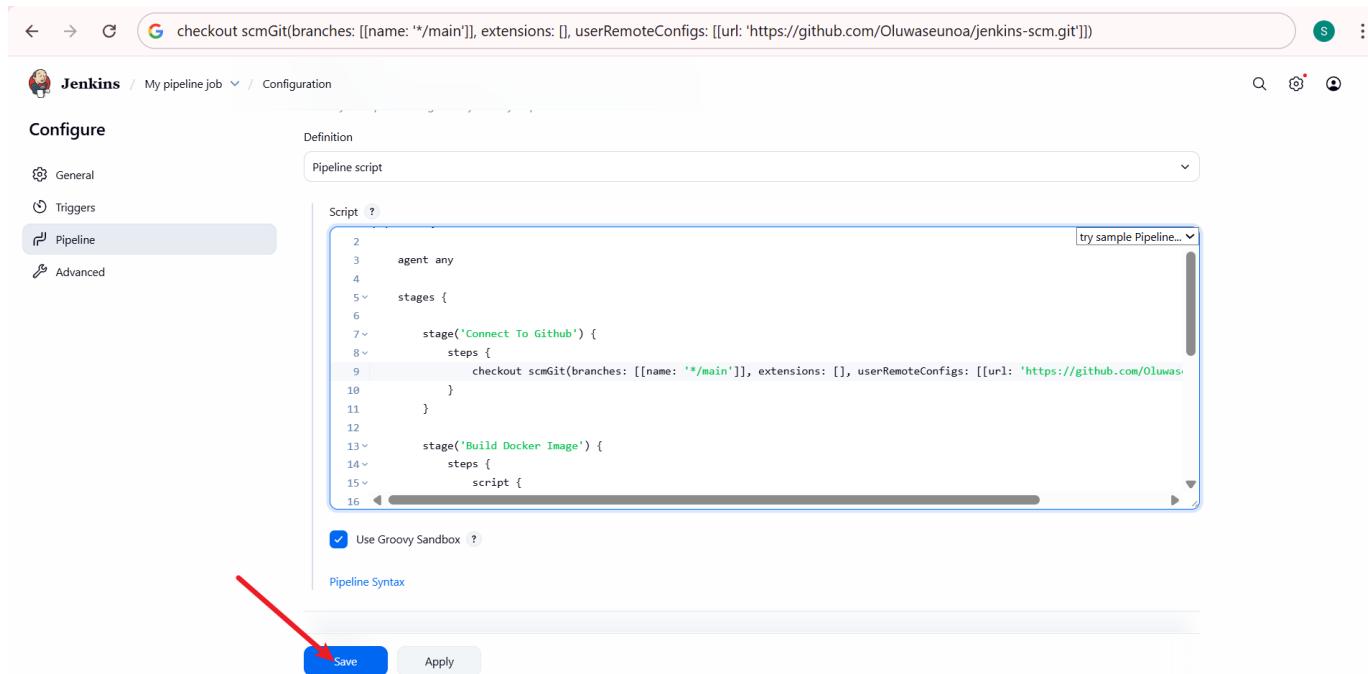
Use Groovy Sandbox ?

Pipeline Syntax

Save Apply

Step 12: Save the Pipeline Job

Click **Save** to apply all configurations.



Jenkins / My pipeline job / Configuration

Configure

General Triggers Pipeline Advanced

Definition Pipeline script

Script ?

```
2
3     agent any
4
5     stages {
6
7         stage('Connect To Github') {
8             steps {
9                 checkout scmGit(bran...
```

try sample Pipeline... ▾

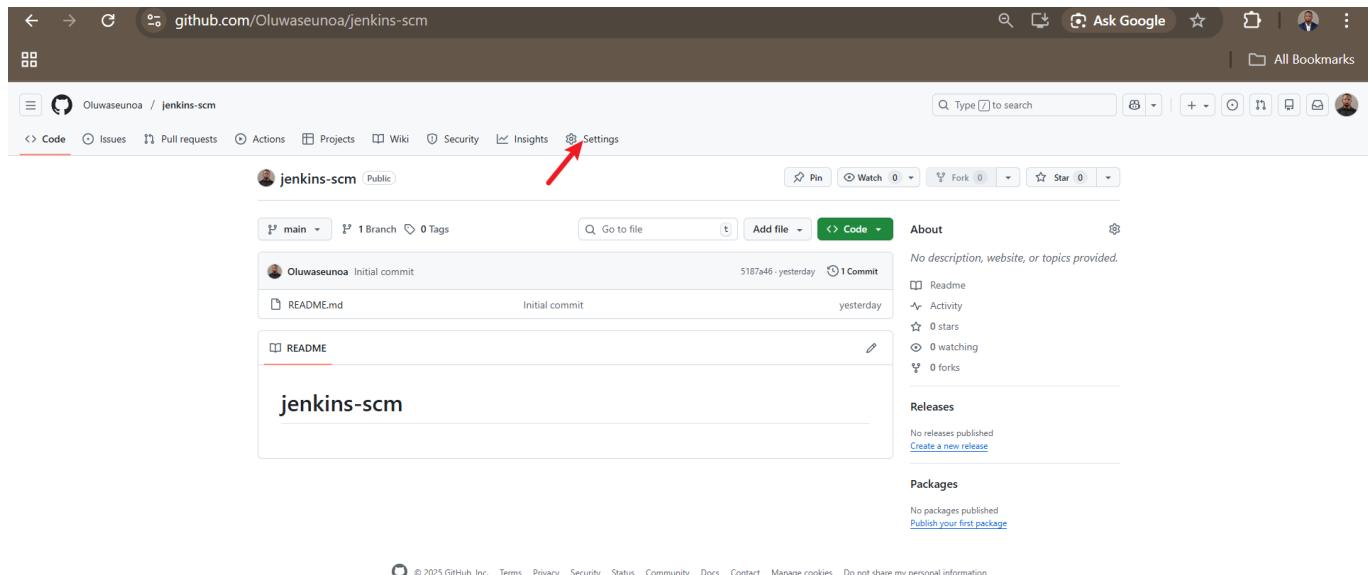
Use Groovy Sandbox ?

Pipeline Syntax

Save Apply

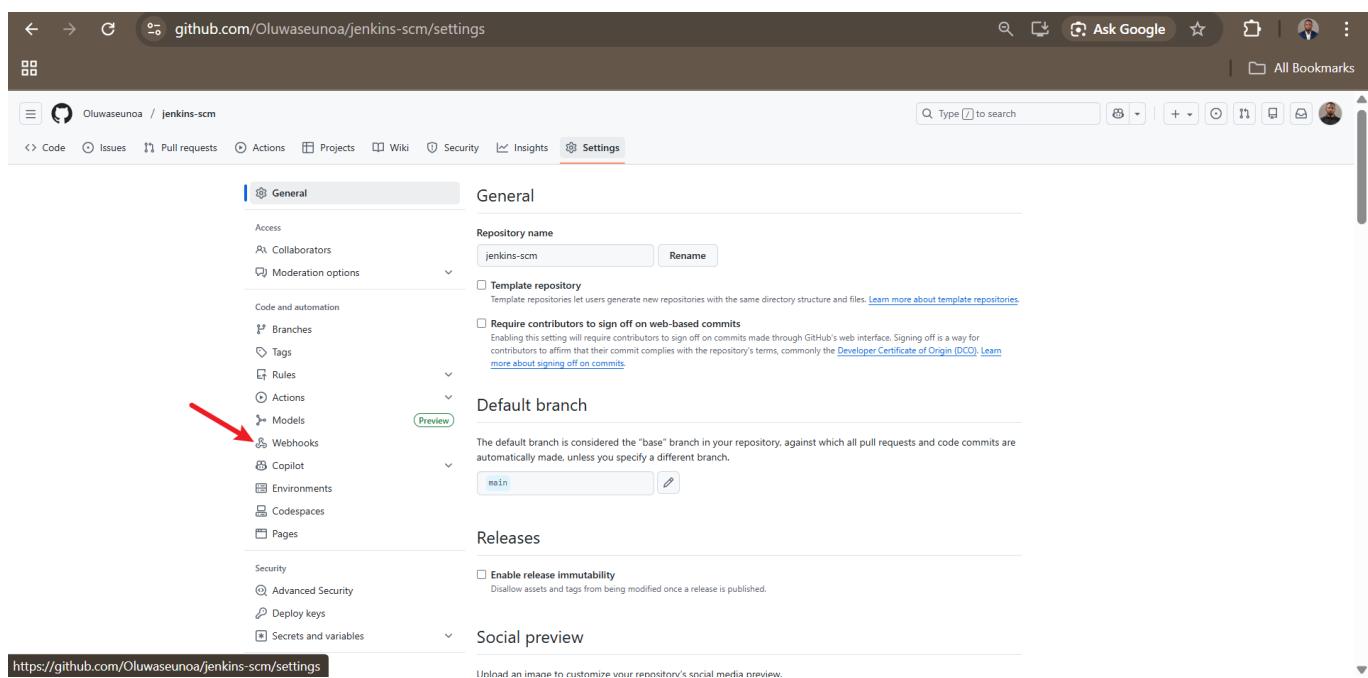
Step 13: Open GitHub Repository Settings

Navigate to your GitHub repository and click **Settings**.



Step 14: Open Webhooks Configuration

Click **Webhooks** from the repository settings menu.



Step 15: Add a New Webhook

Click **Add webhook**.

The screenshot shows the GitHub settings interface for a repository named 'jenkins-scm'. The left sidebar has 'Webhooks' selected under the 'General' category. The main content area is titled 'Webhooks' and contains a brief description of what webhooks are. A red arrow points to the 'Add webhook' button at the top right of the page.

Step 16: Configure Webhook Details

Paste the Jenkins webhook URL, set content type to **application/json**, choose **Just the push event**, and click **Add webhook**.

The screenshot shows the 'Add webhook' form on GitHub. It includes fields for 'Payload URL' (set to 'http://100.49.165.144:8080/github-webhook/'), 'Content type' (set to 'application/json'), and 'SSL verification' (set to 'Enable SSL verification'). Under 'Which events would you like to trigger this webhook?', the 'Just the push event.' option is selected. At the bottom, the 'Active' checkbox is checked, and a red arrow points to the 'Add webhook' button.

Step 17: Webhook Added Successfully

Confirmation that the webhook was added successfully.

The screenshot shows the GitHub settings page for the repository 'jenkins-scm'. The 'Webhooks' section is highlighted with a red border. It displays a single webhook configuration with the URL 'http://100.49.165.144:8080/github... (push)'. A message at the bottom states 'Last delivery was successful.'

Step 18: Create Docker Installation Script

SSH into the Ubuntu server and create a `docker.sh` file using `nano`.



Step 19: Paste Docker Installation Script

Paste the Docker installation commands, save, and exit the file.

```

GNU nano 7.2                               docker.sh *

sudo apt-get update -y
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu \
  $(lsb_release -sc) | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -sc) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update -y
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y
sudo systemctl status docker
  
```

Step 20: Make Script Executable

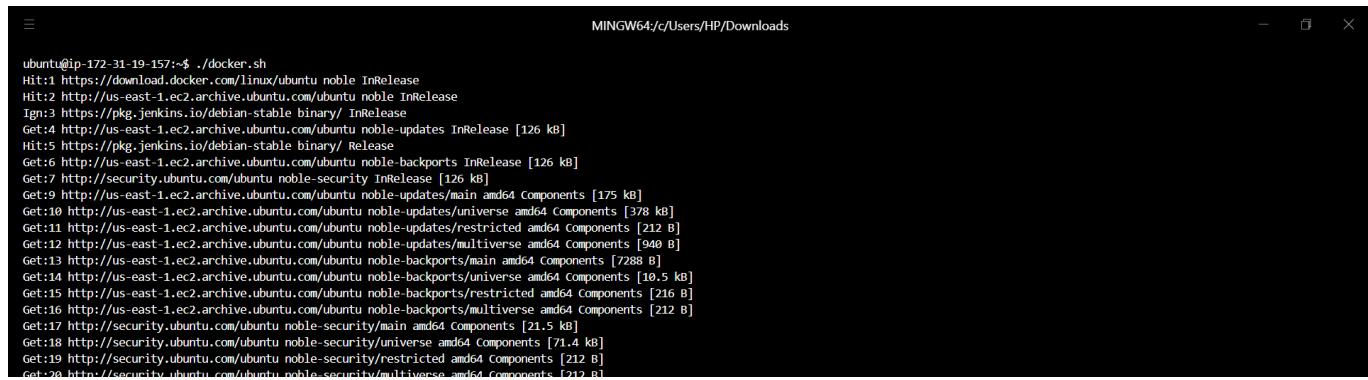
Make the script executable using `chmod`.



```
ubuntu@ip-172-31-19-157:~$ chmod u+x docker.sh
ubuntu@ip-172-31-19-157:~$
```

Step 21: Execute Docker Installation Script

Run the script to install Docker on the Jenkins server.



```
ubuntu@ip-172-31-19-157:~$ ./docker.sh
Hit:1 https://download.docker.com/linux/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Ign:3 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:5 https://pkg.jenkins.io/debian-stable binary/ Release
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [175 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [378 kB]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [946 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [7288 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [10.5 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 kB]
Get:17 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:18 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [71.4 kB]
Get:19 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 kB]
Get:20 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 kB]
```

Step 22: Create Dockerfile

In the project repository, create a `Dockerfile` using NGINX as the base image.

The screenshot shows a code editor interface with a dark theme. At the top, there's a search bar with the text "DevOps-Projects". Below it is a tab bar with "Dockerfile X" selected. The main area displays a Dockerfile content:

```
1 FROM nginx:latest
2
3 WORKDIR /usr/share/nginx/html/
4
5 COPY index.html /usr/share/nginx/html/
6
7 EXPOSE 80
8
```

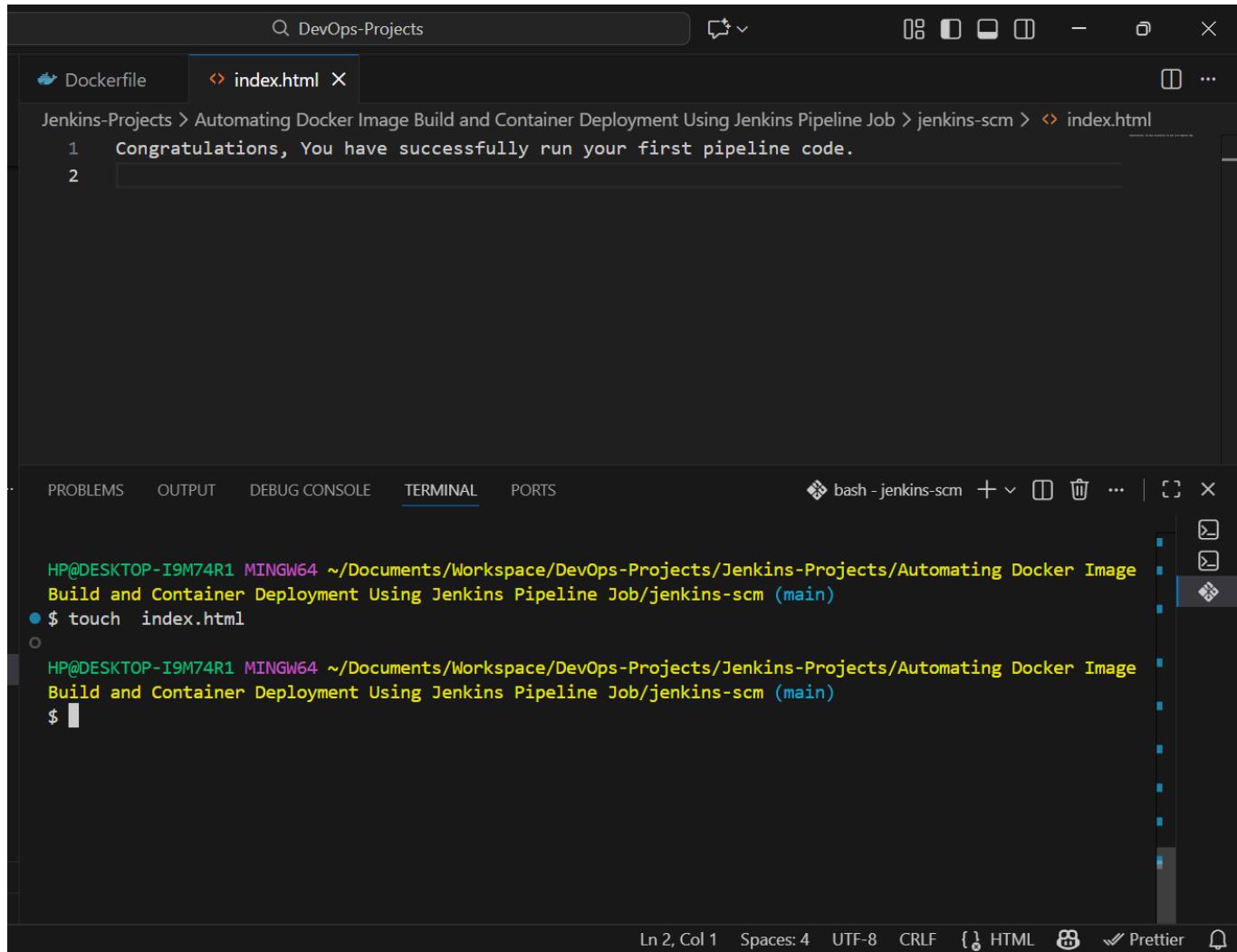
Below the code editor, there's a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected, showing the following terminal history:

- HP@DESKTOP-I9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Jenkins-Projects/Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job/jenkins-scm (main)
\$ touch Dockerfile
- HP@DESKTOP-I9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Jenkins-Projects/Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job/jenkins-scm (main)
\$ █

At the bottom right of the terminal area, there are status indicators: Ln 4, Col 1, Spaces: 4, UTF-8, CRLF, { } Dockerfile, a Prettier icon, and a bell icon.

Step 23: Create index.html File

Create an `index.html` file and add the application content.



The screenshot shows a terminal window in VS Code with the title bar "DevOps-Projects". The tab bar has "Dockerfile" and "index.html X". The status bar at the bottom shows "Ln 2, Col 1" and "Spaces: 4". The terminal content is as follows:

```
HP@DESKTOP-I9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Jenkins-Projects/Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job/jenkins-scm (main)
$ touch index.html
HP@DESKTOP-I9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Jenkins-Projects/Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job/jenkins-scm (main)
$
```

Step 24: Commit and Push Changes

Stage, commit, and push the Dockerfile and index.html to GitHub.

The screenshot shows a dark-themed VS Code interface. At the top, there's a search bar with 'DevOps-Projects' and a tab bar with 'Dockerfile' and 'index.html X'. Below the tabs, a breadcrumb navigation path is visible: Jenkins-Projects > Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job > jenkins-scm > index.html. The main editor area contains the following Jenkins pipeline code:

```
1 Congratulations, You have successfully run your first pipeline code.  
2
```

Below the editor, the terminal tab is active, showing the command line output of a GitHub push:

```
create mode 100644 index.html  
  
HP@DESKTOP-I9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Jenkins-Projects/Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job/jenkins-scm (main)  
$ git push  
Counting objects: 100% (5/5), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (4/4), 485 bytes | 121.00 KiB/s, done.  
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)  
To https://github.com/Oluwaseun0/jenkins-scm.git  
 b89796d..8c524e0 main -> main  
  
HP@DESKTOP-I9M74R1 MINGW64 ~/Documents/Workspace/DevOps-Projects/Jenkins-Projects/Automating Docker Image Build and Container Deployment Using Jenkins Pipeline Job/jenkins-scm (main)  
$
```

The bottom status bar shows 'Ln 2, Col 1' and other terminal settings.

Step 25: Pipeline Triggered Automatically

The GitHub push automatically triggers the Jenkins pipeline.

The screenshot shows the Jenkins web interface. The URL is '100.49.165.144:8080/job/My%20pipeline%20job/'. The left sidebar has a 'Status' tab selected, showing a green checkmark icon next to 'My pipeline job'. Other options include 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Stages', 'Rename', 'Pipeline Syntax', and 'GitHub Hook Log'. The main content area shows the 'My pipeline job' page with a 'Permalinks' section listing four builds: 'Last build (#2), 14 sec ago', 'Last stable build (#2), 14 sec ago', 'Last successful build (#2), 14 sec ago', and 'Last completed build (#2), 14 sec ago'. Below this is a 'Builds' section with a 'Filter' input field and a table showing one build entry: 'Today #2 5:24PM'. The bottom right corner of the page footer indicates 'REST API' and 'Jenkins 2.528.3'.

Step 26: View Console Output

Open the console output to monitor pipeline execution.

The screenshot shows the Jenkins interface for a pipeline job named 'My pipeline job'. On the left, there's a sidebar with various options like 'Status', 'Changes', 'Build Now', 'Configure', etc. The main area shows a build history with build #2 highlighted. A context menu is open over build #2, listing options such as 'Console Output', 'Edit Build Information', 'Delete build #2', 'Polling Log', 'Timings', 'Git Build Data', 'Pipeline Overview', 'Restart from Stage', 'Replay', 'Pipeline Steps', and 'Workspaces'. The 'Console Output' option is specifically highlighted with a red arrow.

REST API Jenkins 2.528.3

Step 27: Pipeline Finished Successfully

Pipeline execution completed successfully.

The screenshot shows the Jenkins 'Console Output' page for build #2. The output log is displayed in a large text area. The log shows the execution of a pipeline script, including steps like exporting an attestation manifest, naming the Docker image to 'docker.io/library/dockerfile:latest', unpacking the Dockerfile, and running it with Docker. The final message in the log is 'Finished: SUCCESS', which is highlighted with a green box.

REST API Jenkins 2.528.3

Step 28: Open Security Group

Navigate to the instance dashboard and click the **Security Group**.

Instance summary for i-05a861204b525afe2 (Ubuntu-Server)

Security details

- IAM Role: -
- Security groups: sg-0eaab0976cf2ad548 (launch-wizard-5)

Step 29: Edit Inbound Rules

Click **Edit inbound rules**.

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
sgr-08aaa80d1e6256ab6	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-	
sgr-024262e02e5cdff6f	IPv4	SSH	TCP	22	0.0.0.0/0	-	

Step 30: Add Port 8081 Rule

Add a **Custom TCP rule** for port **8081** and allow access from anywhere.

The screenshot shows the 'Edit inbound rules' step of the AWS EC2 Security Groups wizard. A new rule is being added with the following details:

Type	Protocol	Port range	Source	Description
Custom TCP	TCP	8081	Anywhere...	0.0.0.0/0

A red box highlights the 'Add rule' button at the bottom left. A yellow warning message at the bottom center states: '⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.' To the right of the message are 'Cancel', 'Preview changes', and a prominent orange 'Save rules' button.

Step 31: Inbound Rule Added Successfully

Inbound security group rules updated successfully.

The screenshot shows the 'Details' page for the security group 'sg-0eaab0976cf2ad548'. The 'Inbound rules' tab is selected, displaying the following table:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-08aaa80d1e6256ab6	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	-
-	sgr-0fd241e5c14d1ee73	IPv4	Custom TCP	TCP	8081	0.0.0.0/0	-
-	sgr-024262e02e5cdff6f	IPv4	SSH	TCP	22	0.0.0.0/0	-

Step 32: Access the Application

Visit the application using the server IP on port **8081**.

```
http://<server-ip>:8081
```

A screenshot of a web browser window. The address bar shows the URL as 100.49.165.144:8081. A message in the center of the page says "Congratulations, You have successfully run your first pipeline code." The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with icons for star, save, and more.

Congratulations, You have successfully run your first pipeline code.

Project Outcome

- Jenkins Pipeline Job successfully created
 - GitHub webhook integrated
 - Docker image built automatically
 - Docker container deployed automatically
 - Application accessible via web browser
-

Key Learning Outcomes

- Jenkins Pipeline Job configuration
- Writing Declarative Pipeline scripts
- GitHub webhook integration
- Automating Docker builds with Jenkins
- Implementing Continuous Integration (CI)

Author

Oluwaseun DevOps / Cloud Engineering Project