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

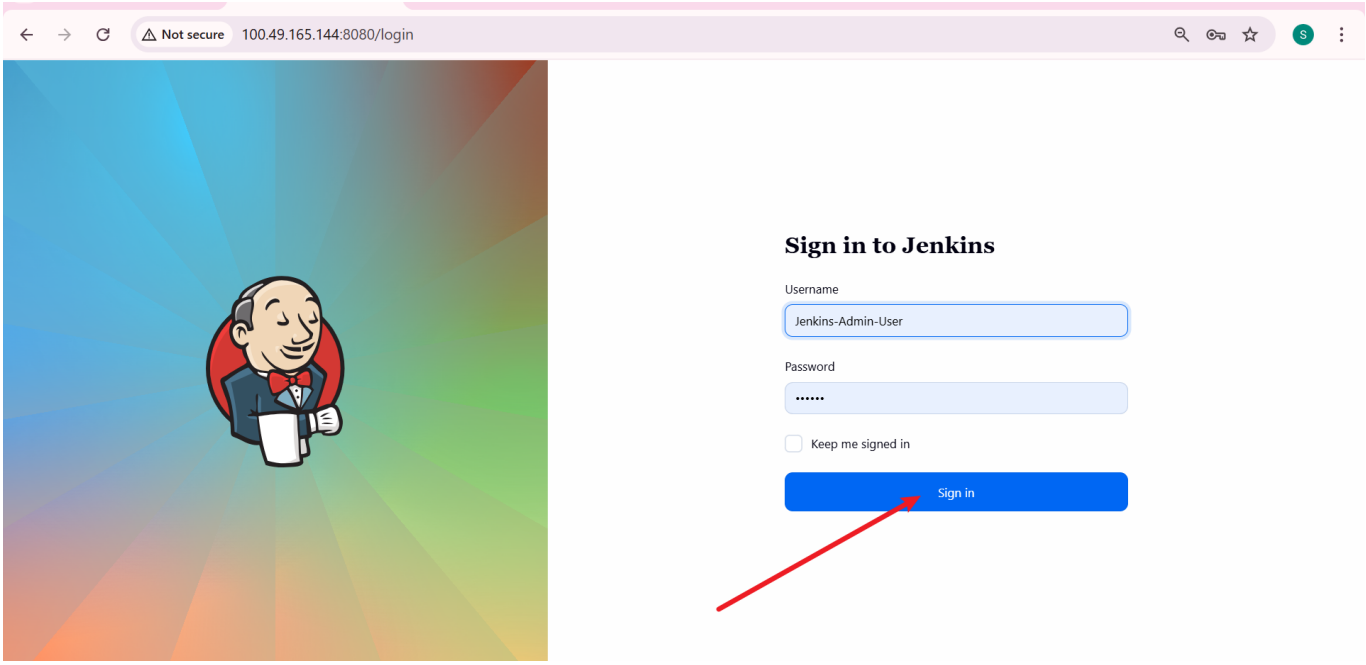
Project Prerequisites

- 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

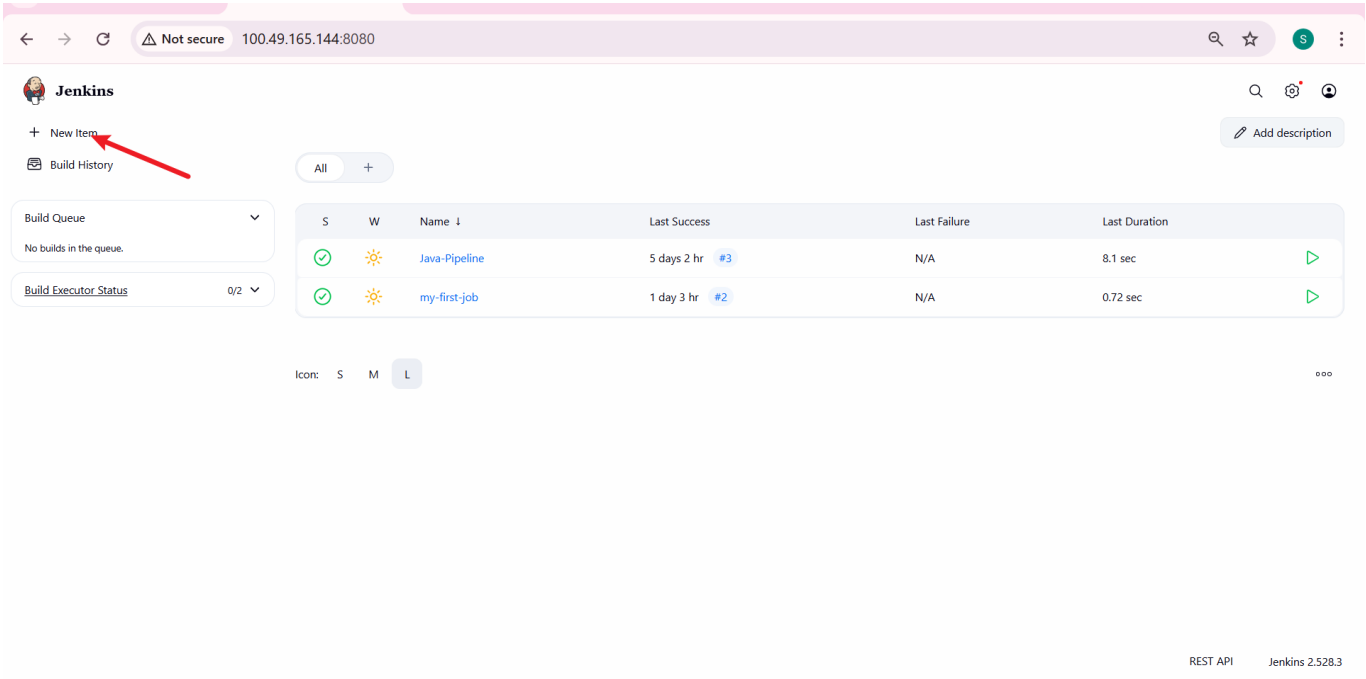
Step 1: Sign in to Jenkins

Log in to the Jenkins dashboard using your credentials.



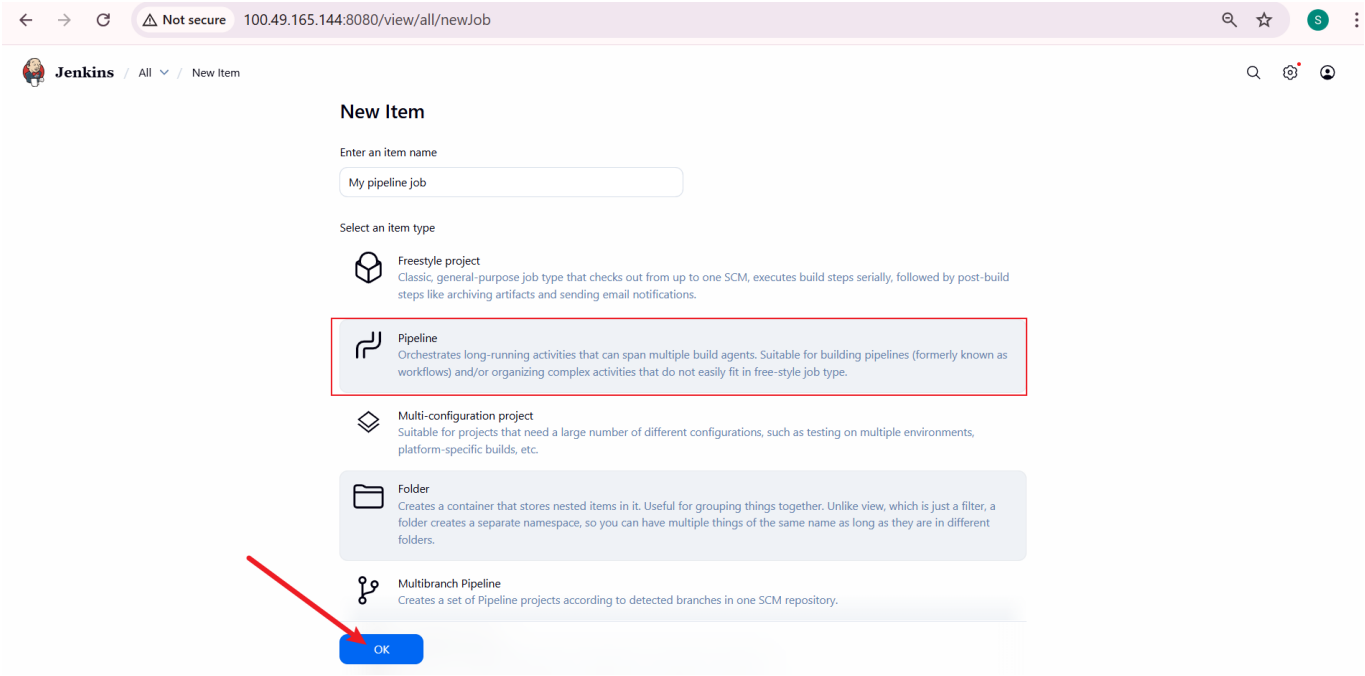
Step 2: Create a New Jenkins Job

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



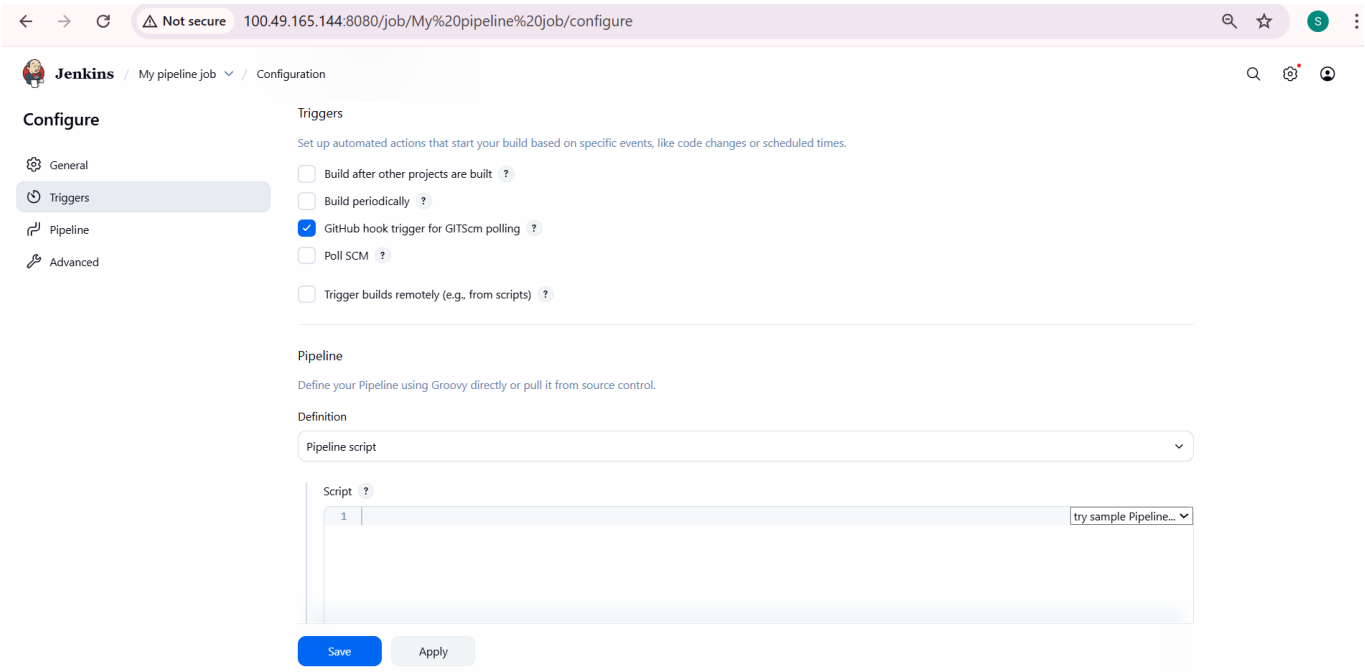
Step 3: Create a Pipeline Job

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



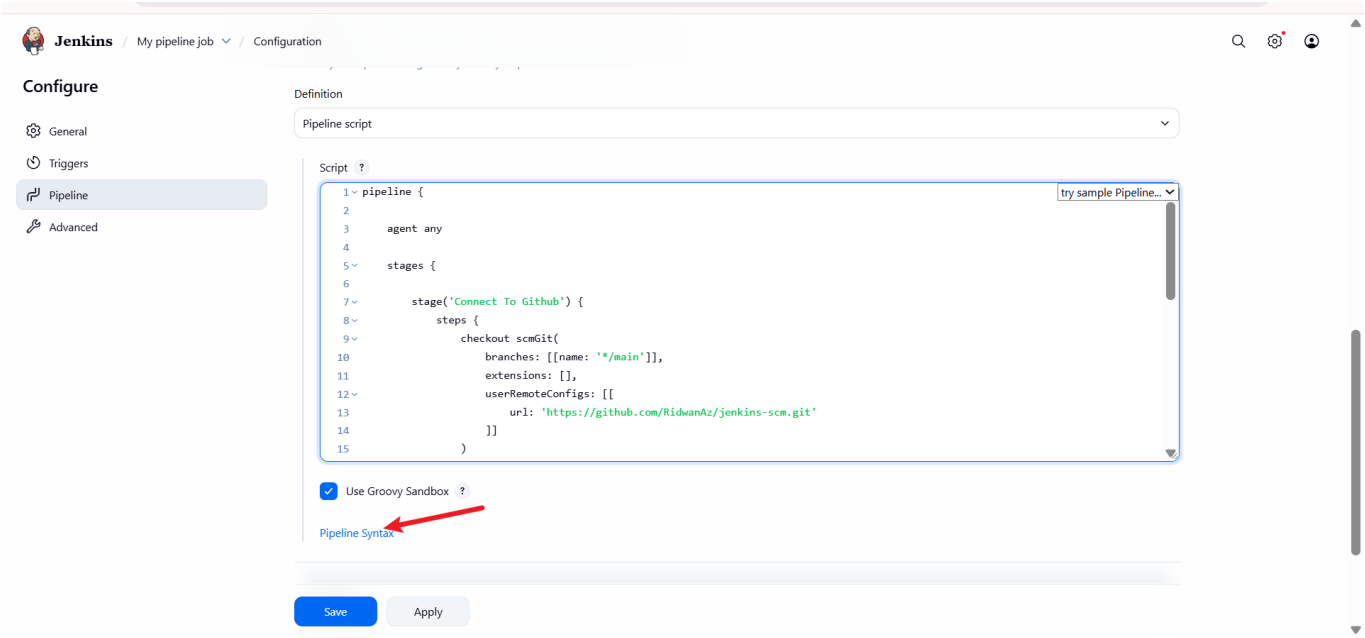
Step 4: Configure Build Trigger

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



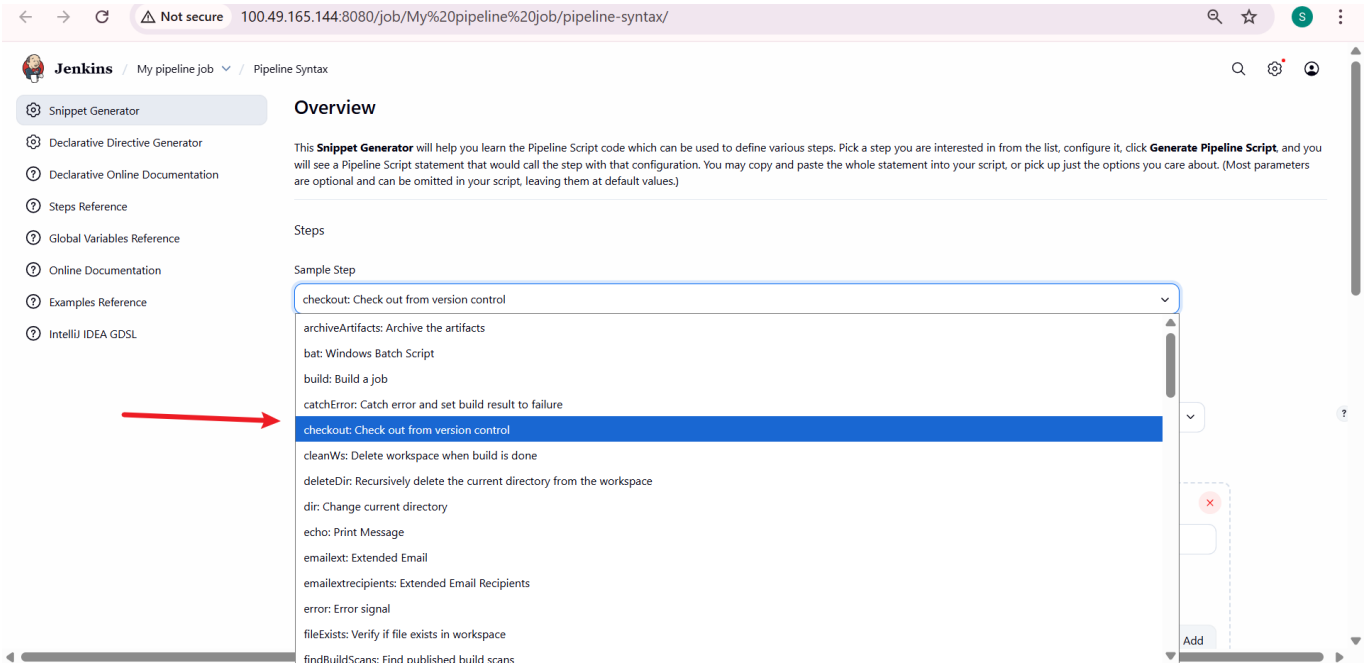
Step 5: Configure Pipeline Script Section

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



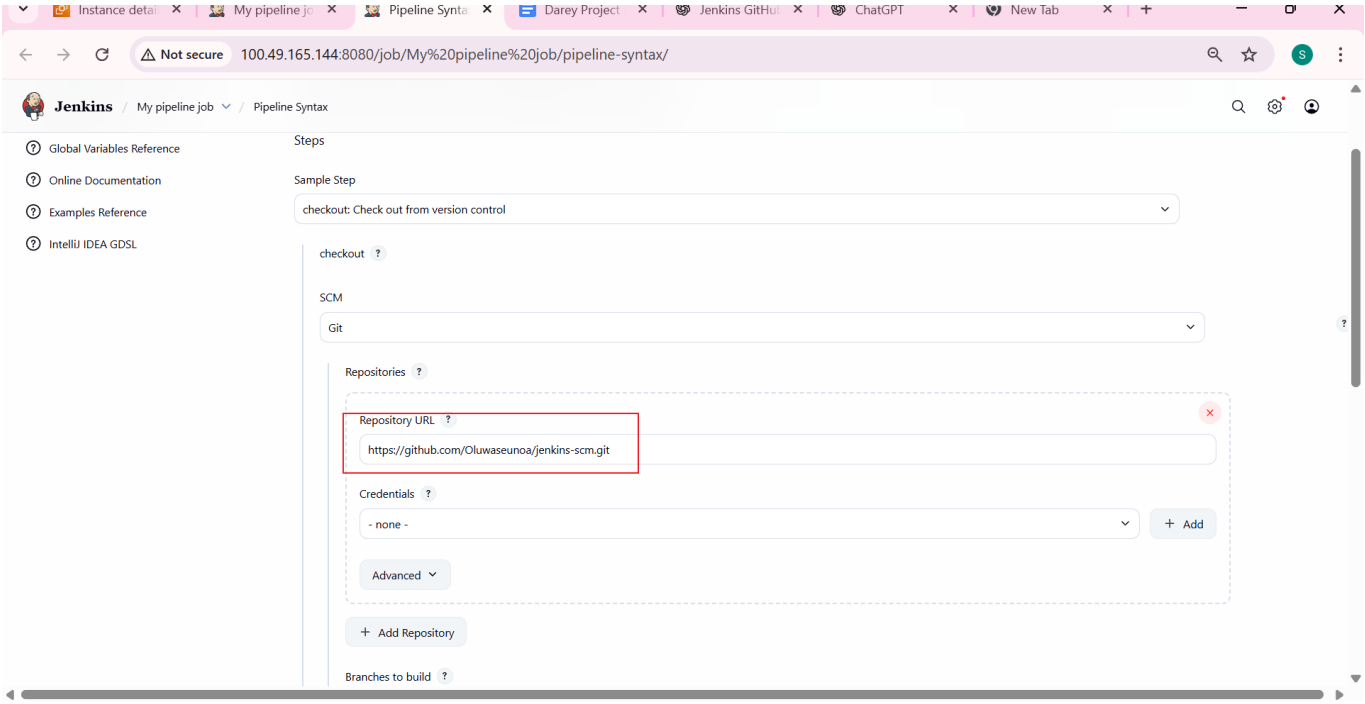
Step 6: Select Checkout Sample Step

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



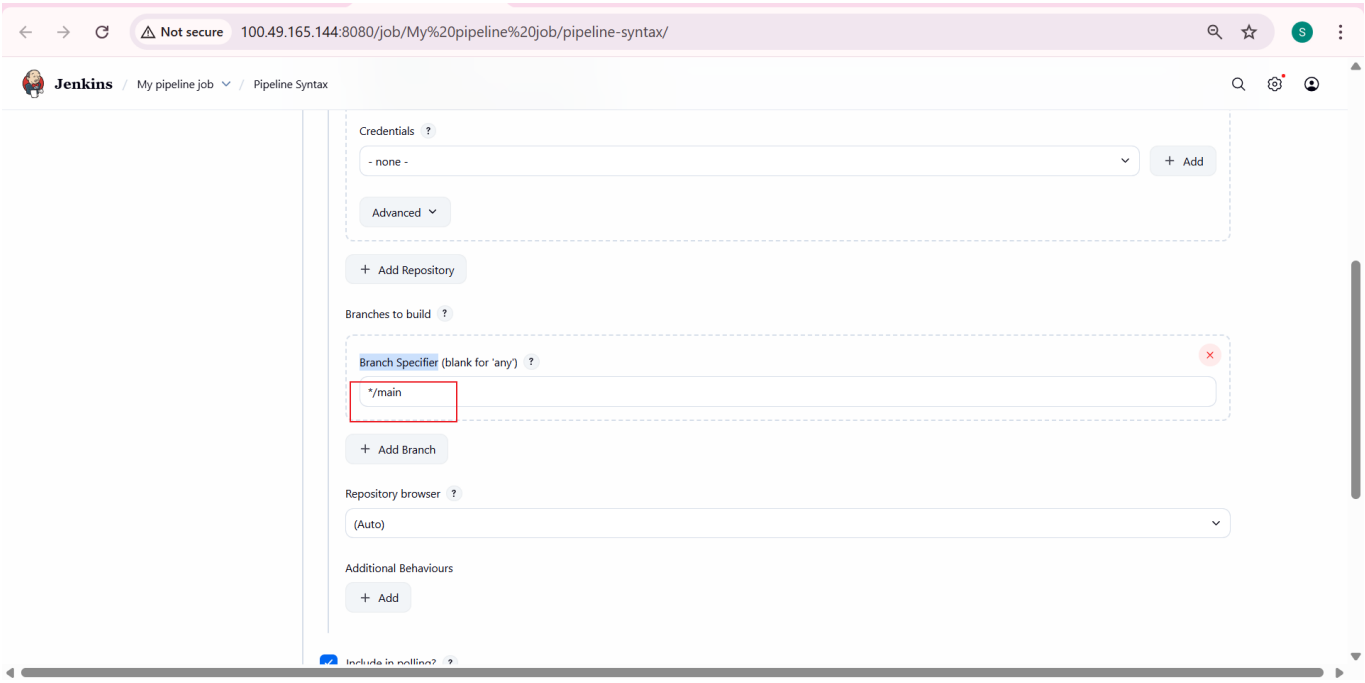
Step 7: Configure Git SCM

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



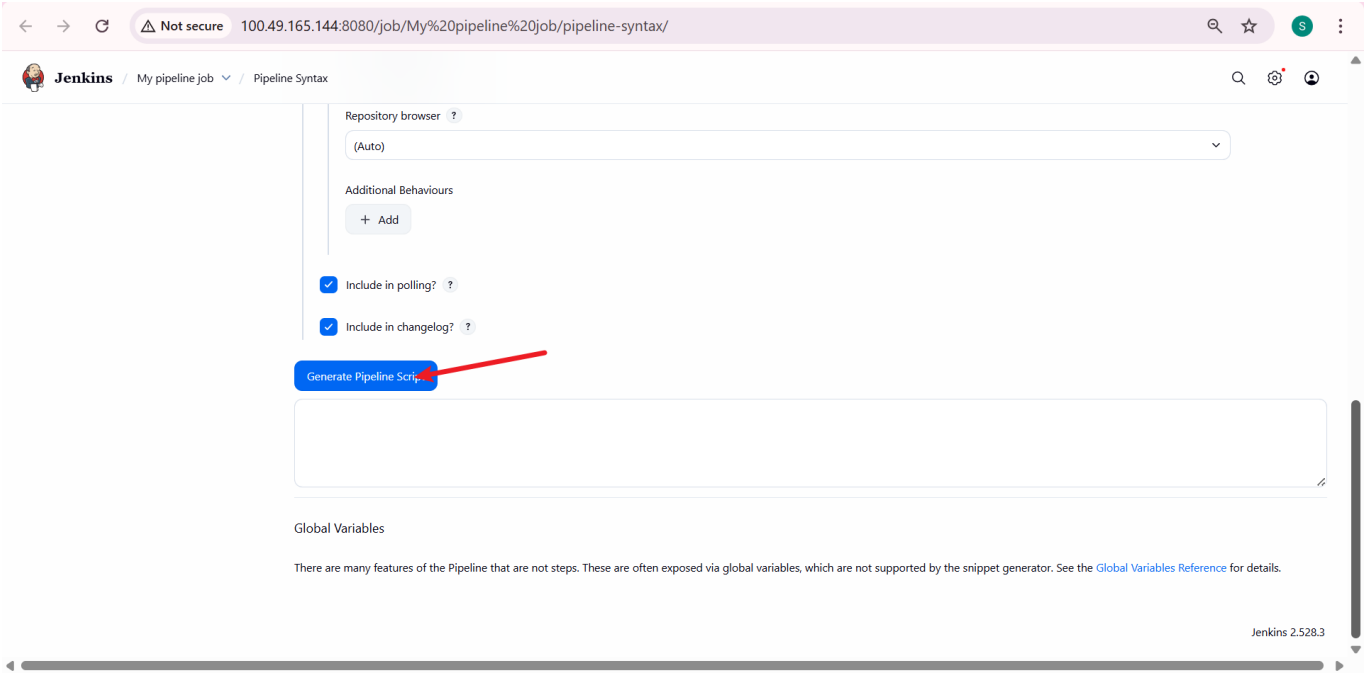
Step 8: Set Branch Specifier

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



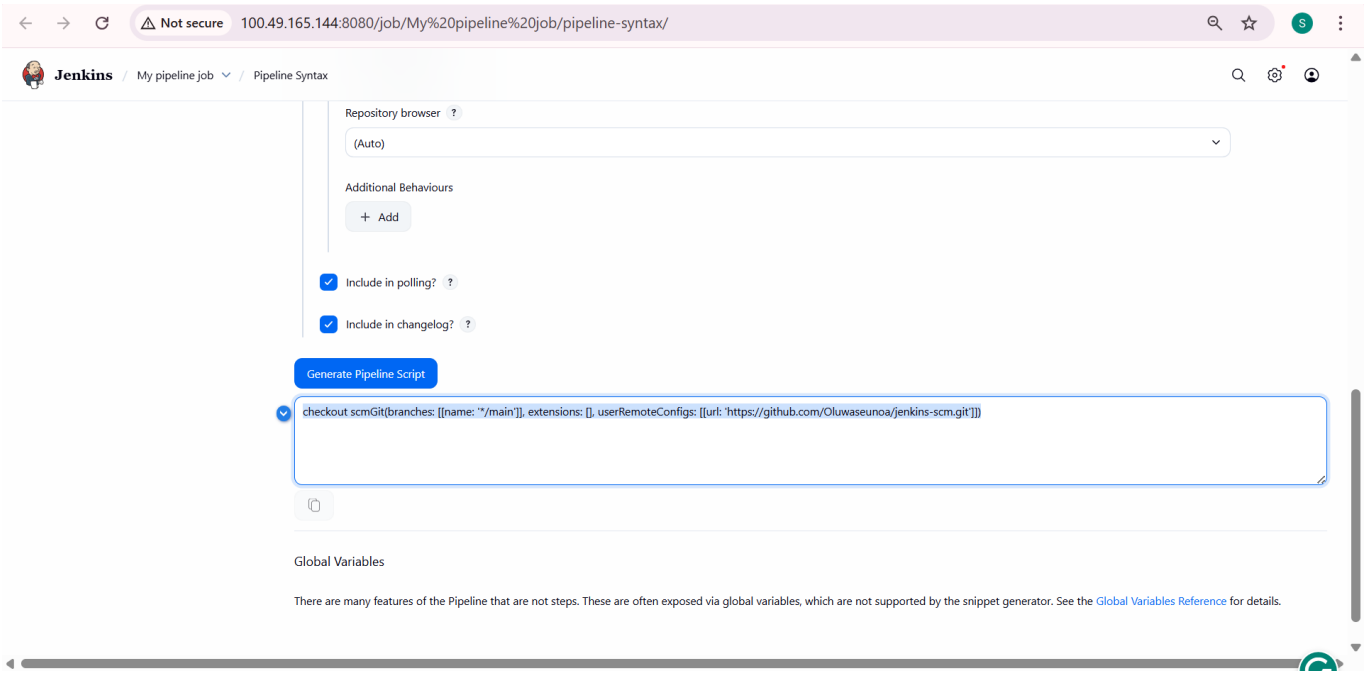
Step 9: Generate Pipeline Script

Click **Generate Pipeline Script**.



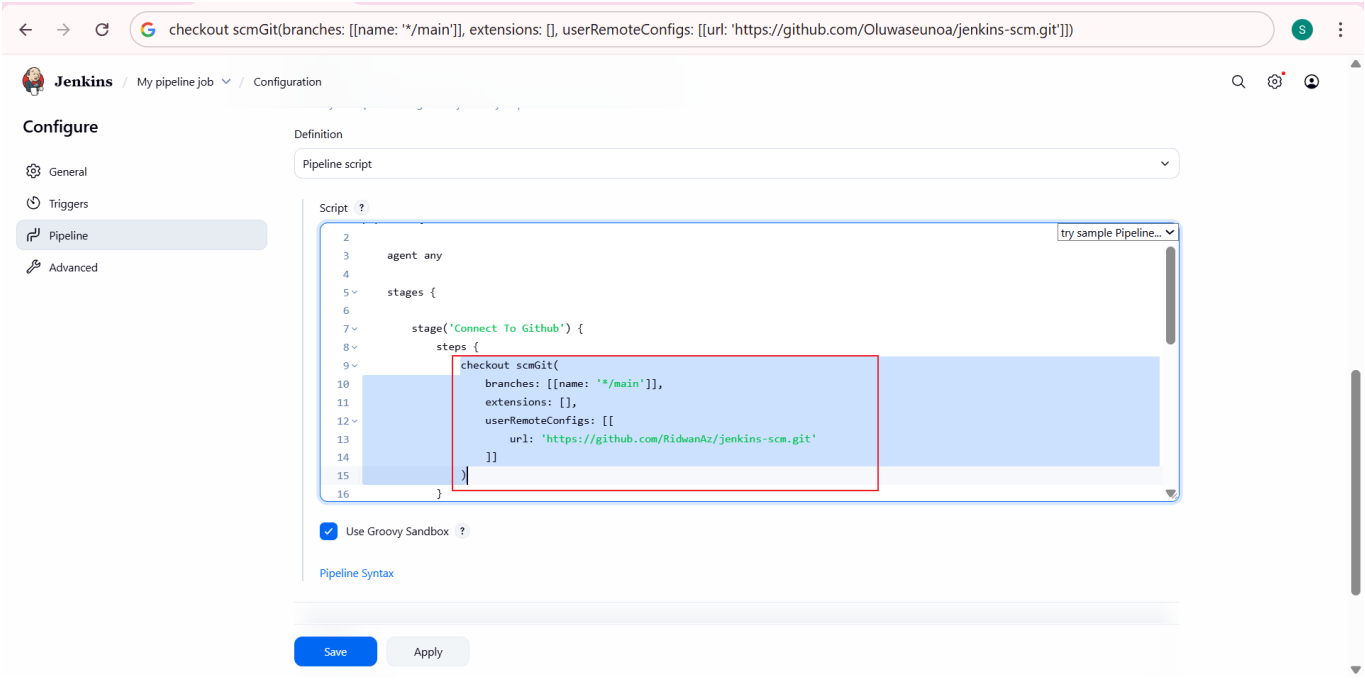
Step 10: Copy Generated Script

Copy the generated pipeline checkout syntax.



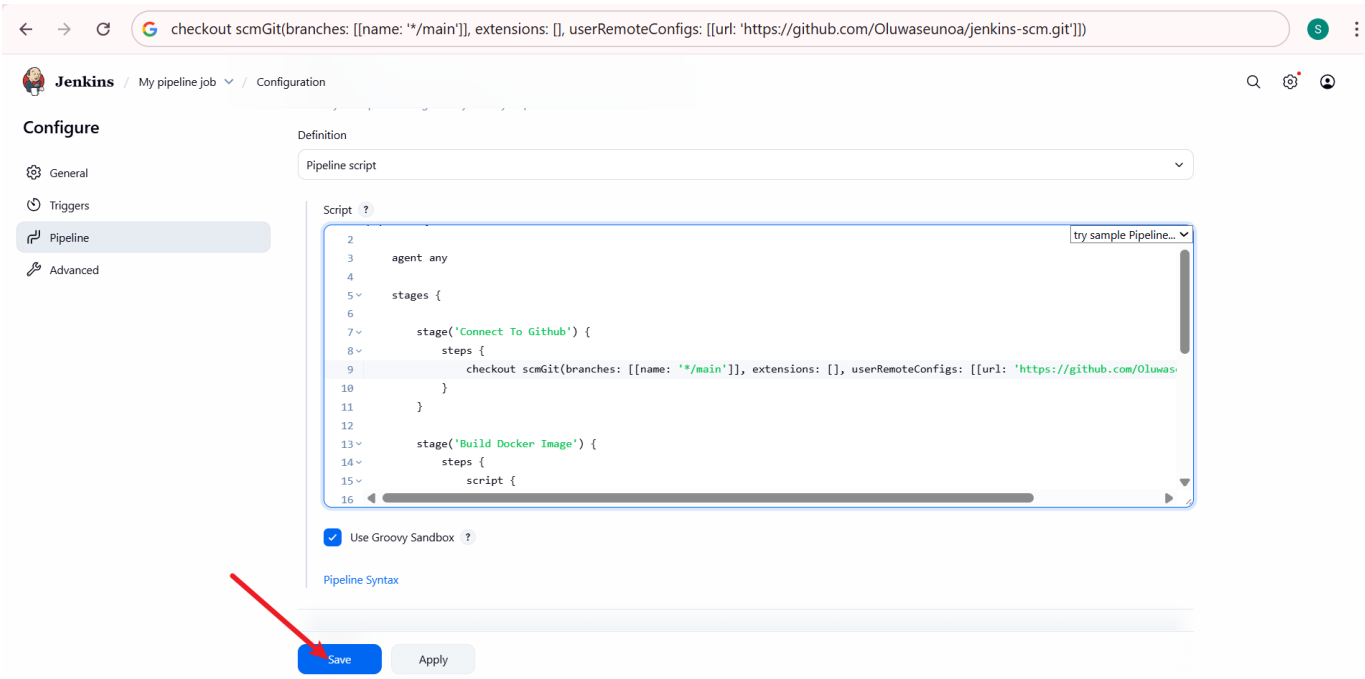
Step 11: Replace Checkout Section

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



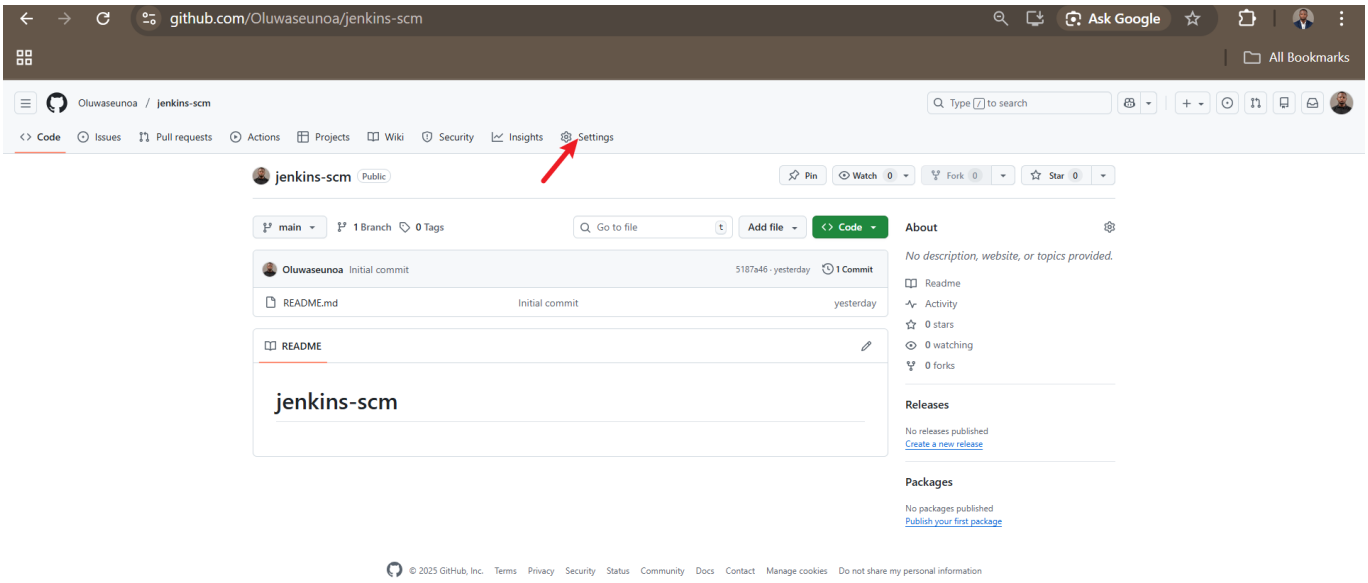
Step 12: Save the Pipeline Job

Click **Save** to apply all configurations.



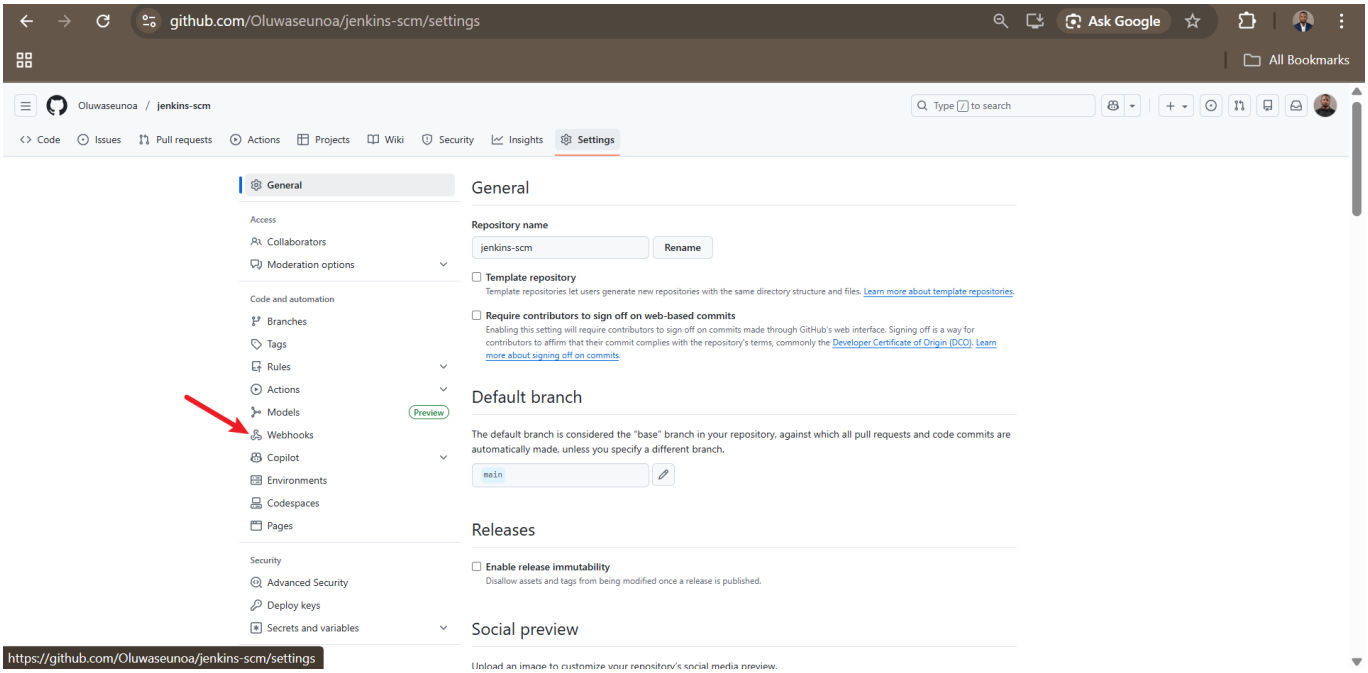
Step 13: Open GitHub Repository Settings

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



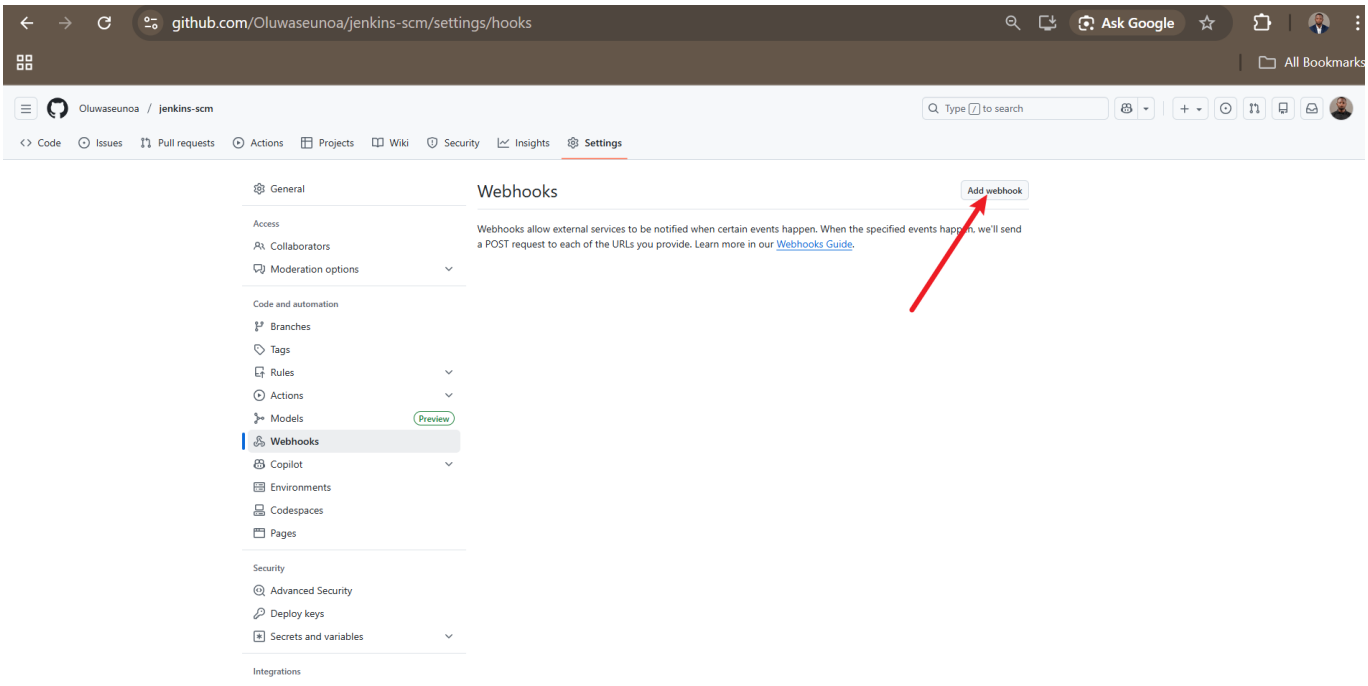
Step 14: Open Webhooks Configuration

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



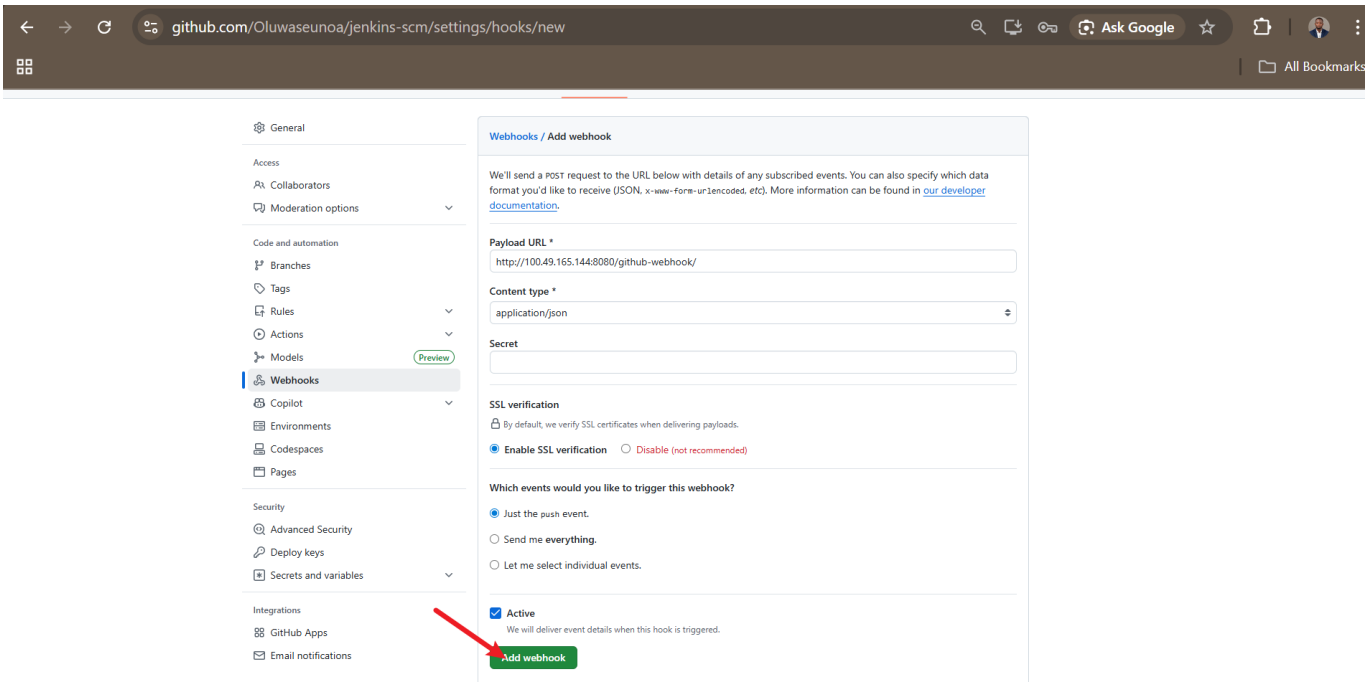
Step 15: Add a New Webhook

Click **Add webhook**.



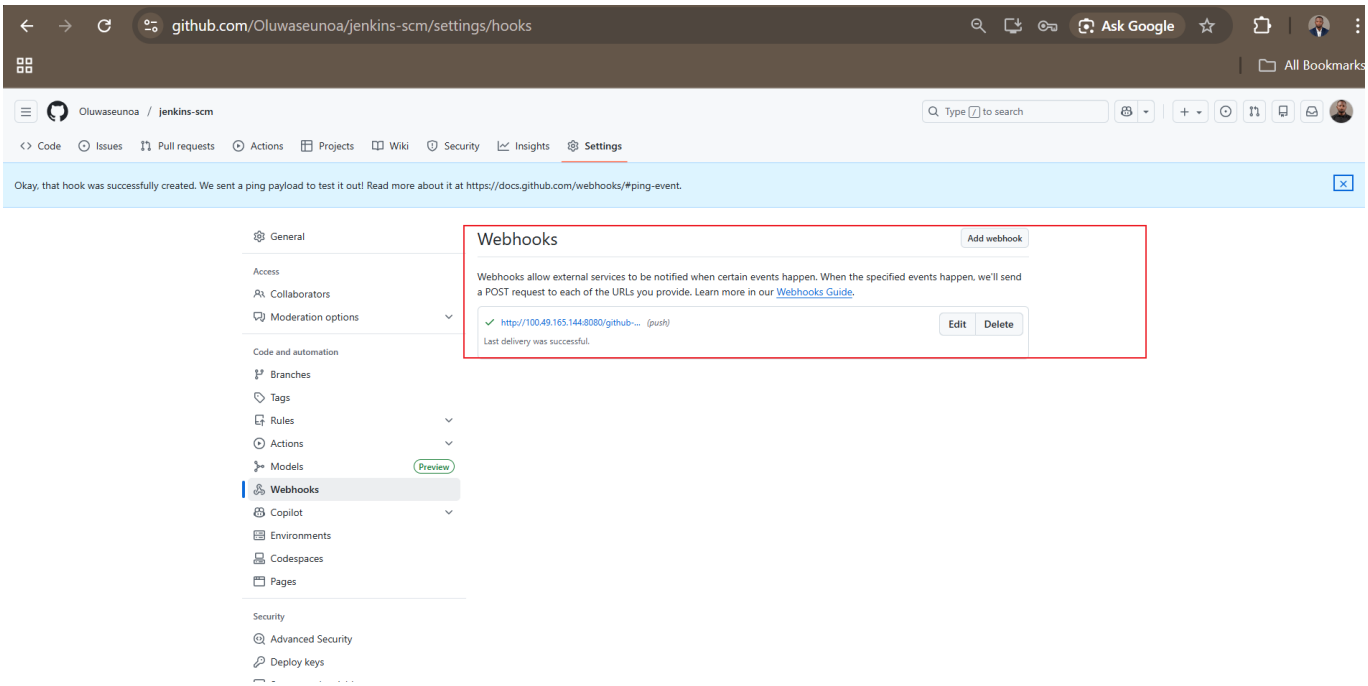
Step 16: Configure Webhook Details

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



Step 17: Webhook Added Successfully

Confirmation that the webhook was added successfully.



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

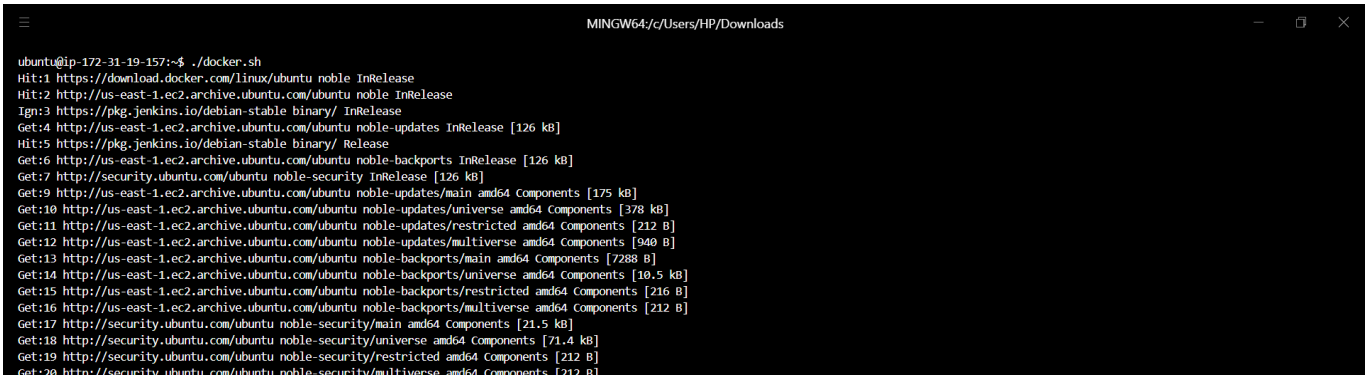
Paste the Docker installation commands, save, and exit the file.

Step 20: Make Script Executable

Make the script executable using `chmod`.

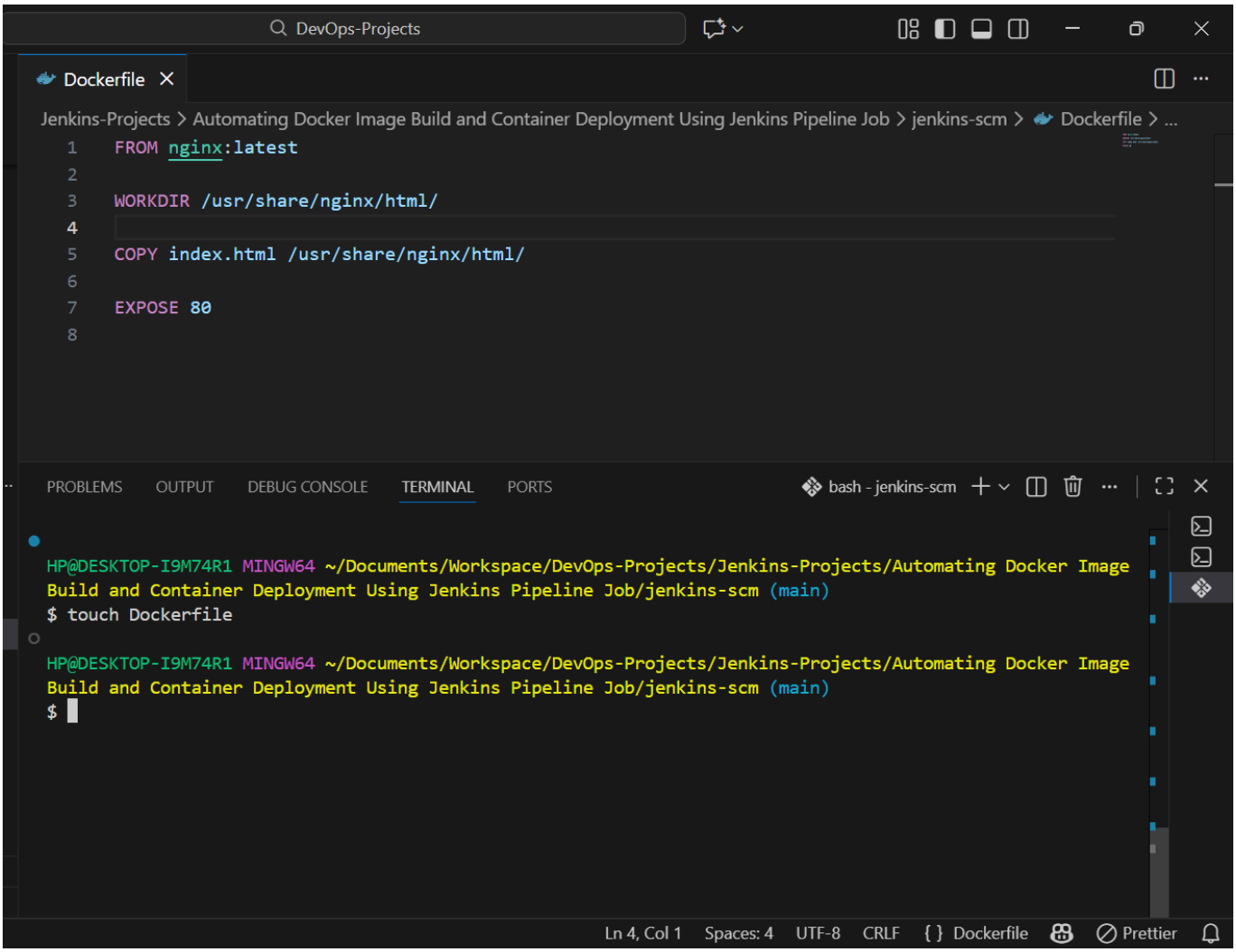
Step 21: Execute Docker Installation Script

Run the script to install Docker on the Jenkins server.



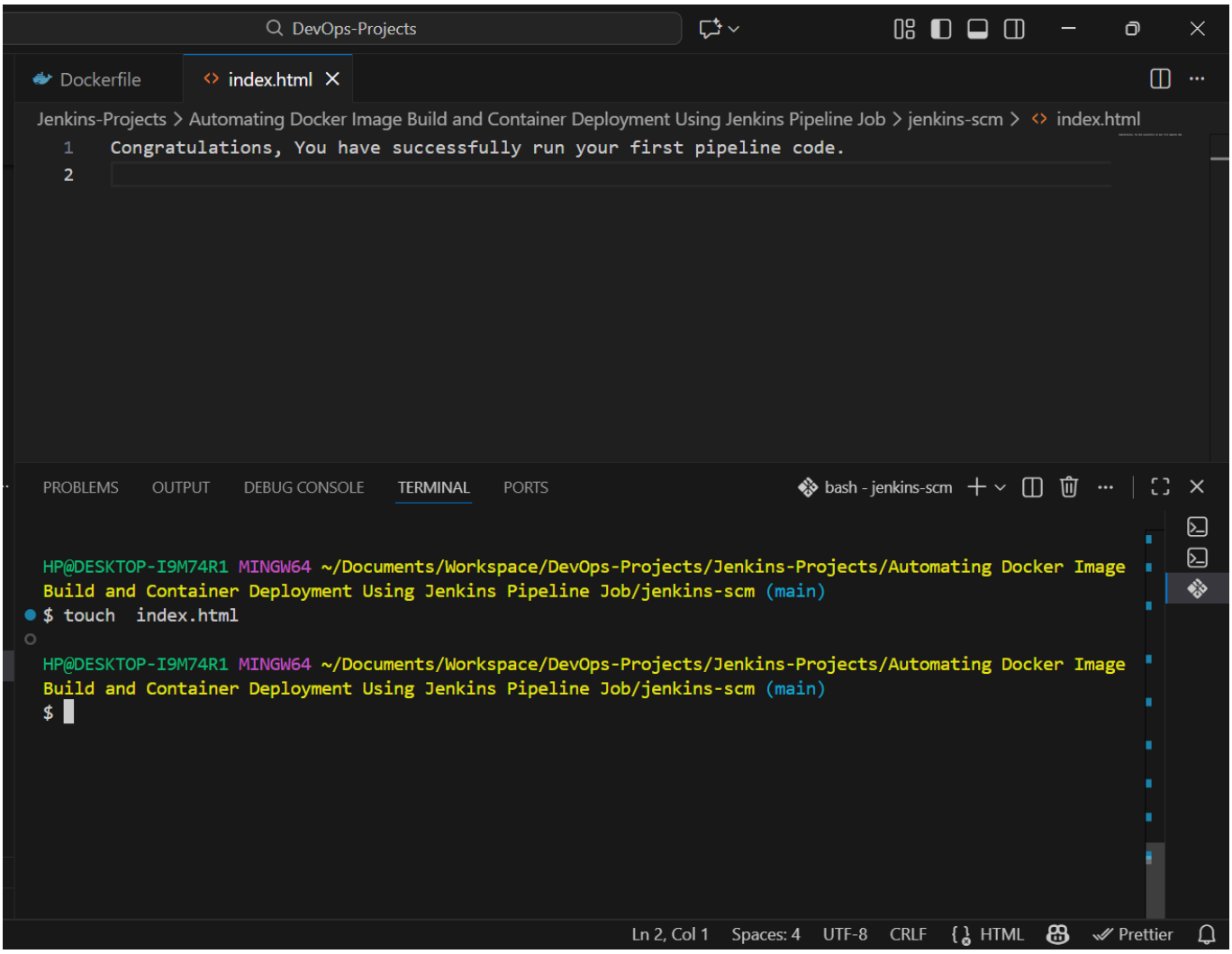
Step 22: Create Dockerfile

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



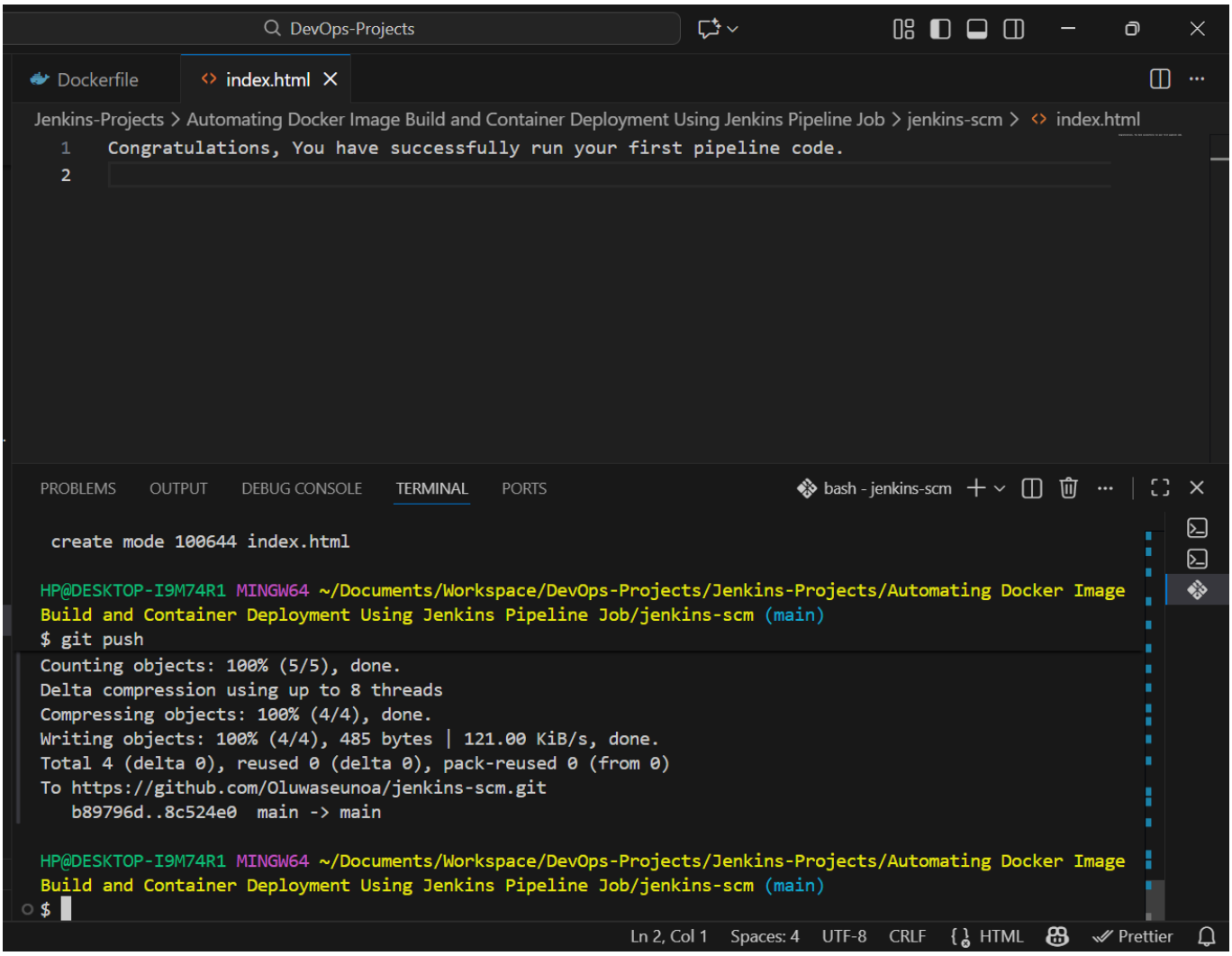
Step 23: Create index.html File

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



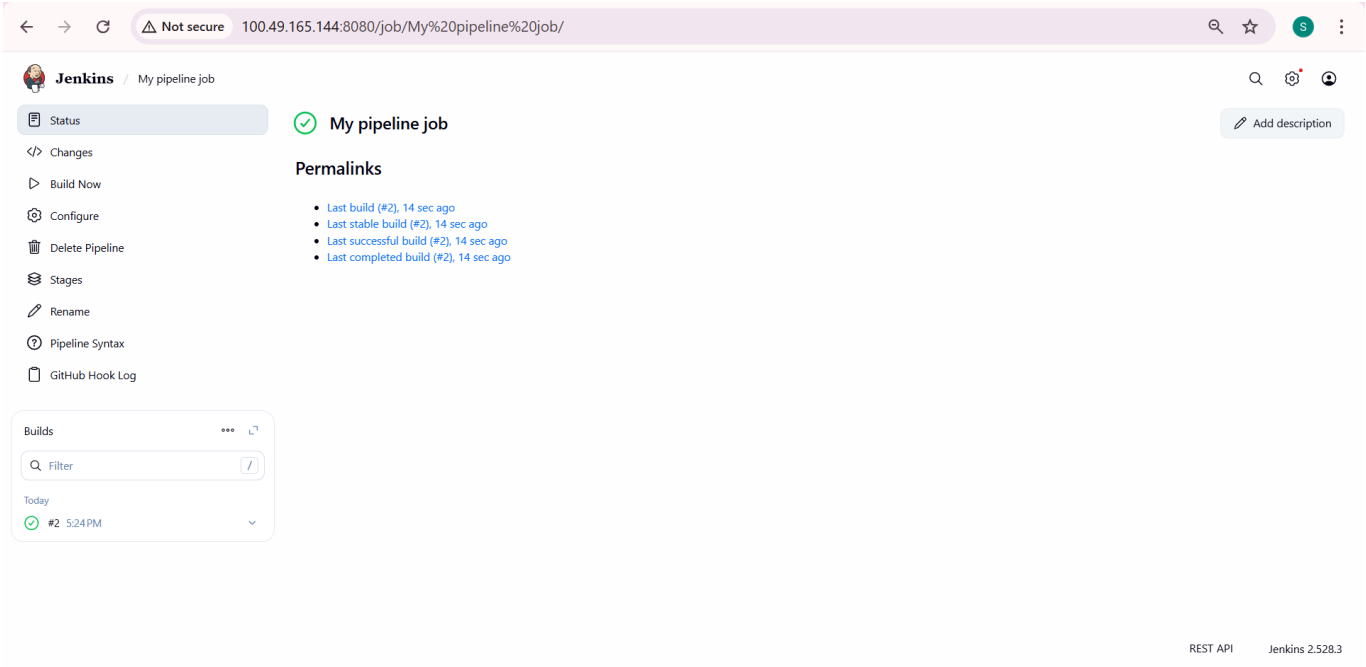
Step 24: Commit and Push Changes

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



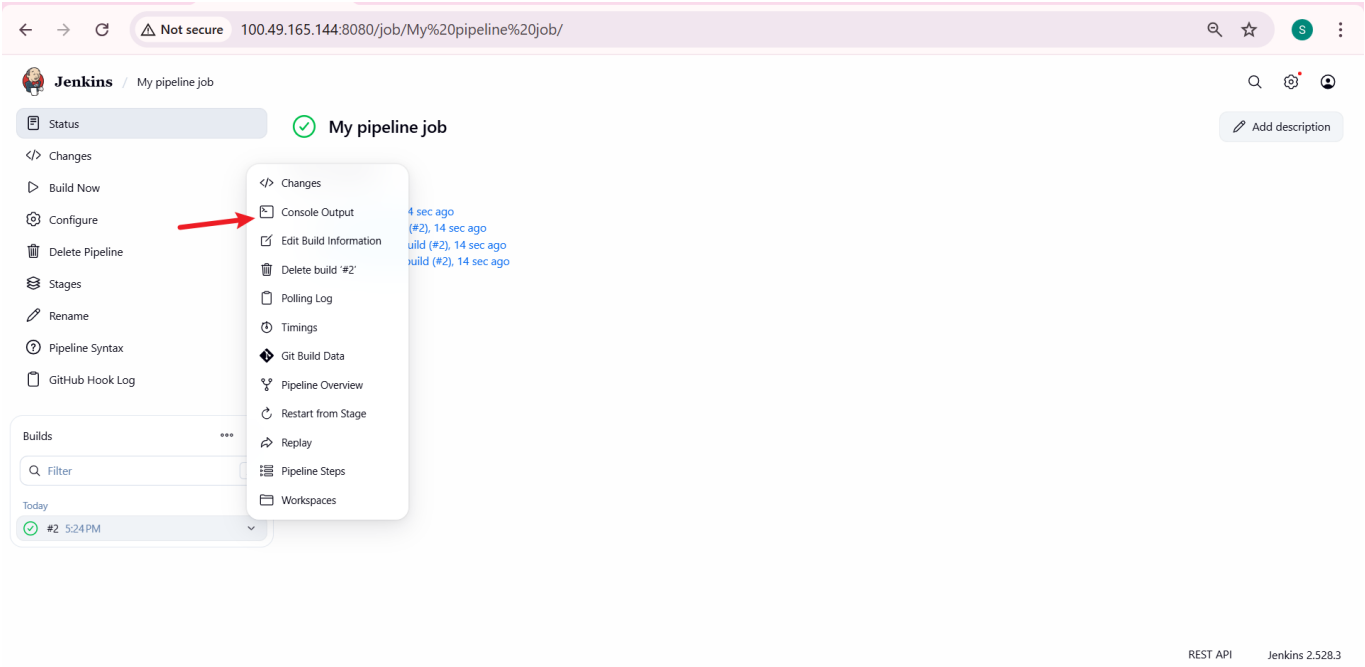
Step 25: Pipeline Triggered Automatically

The GitHub push automatically triggers the Jenkins pipeline.



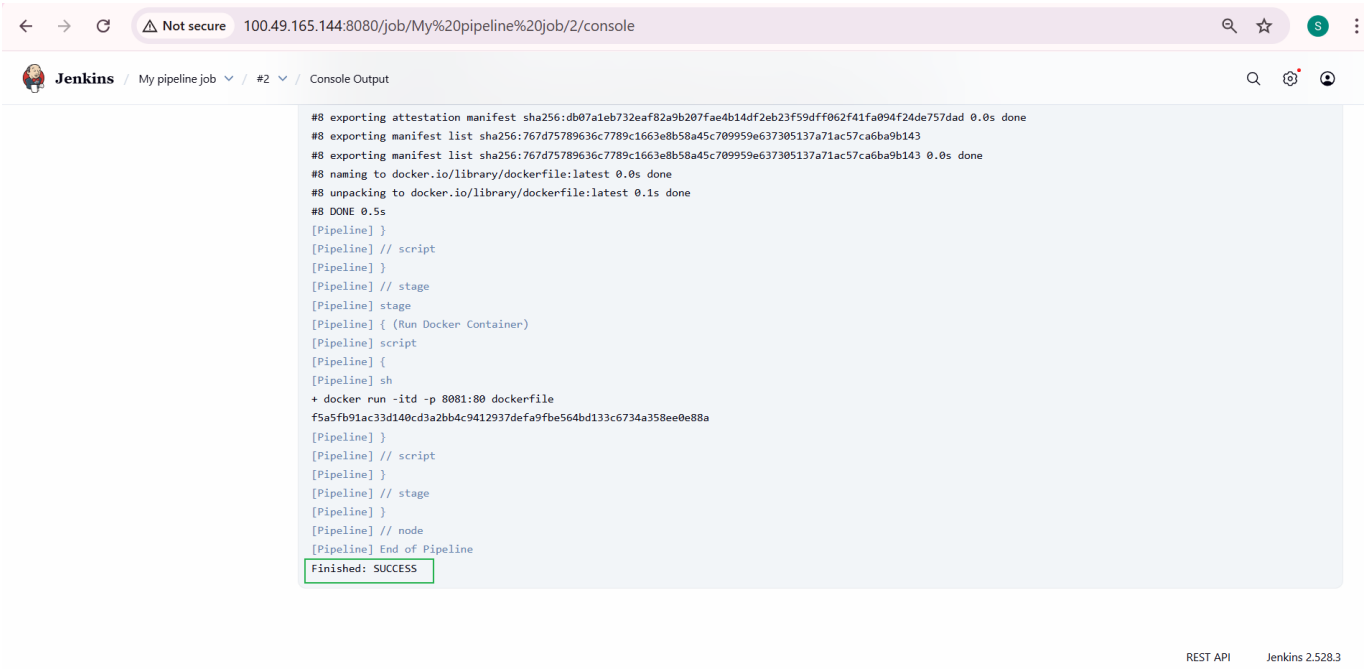
Step 26: View Console Output

Open the console output to monitor pipeline execution.



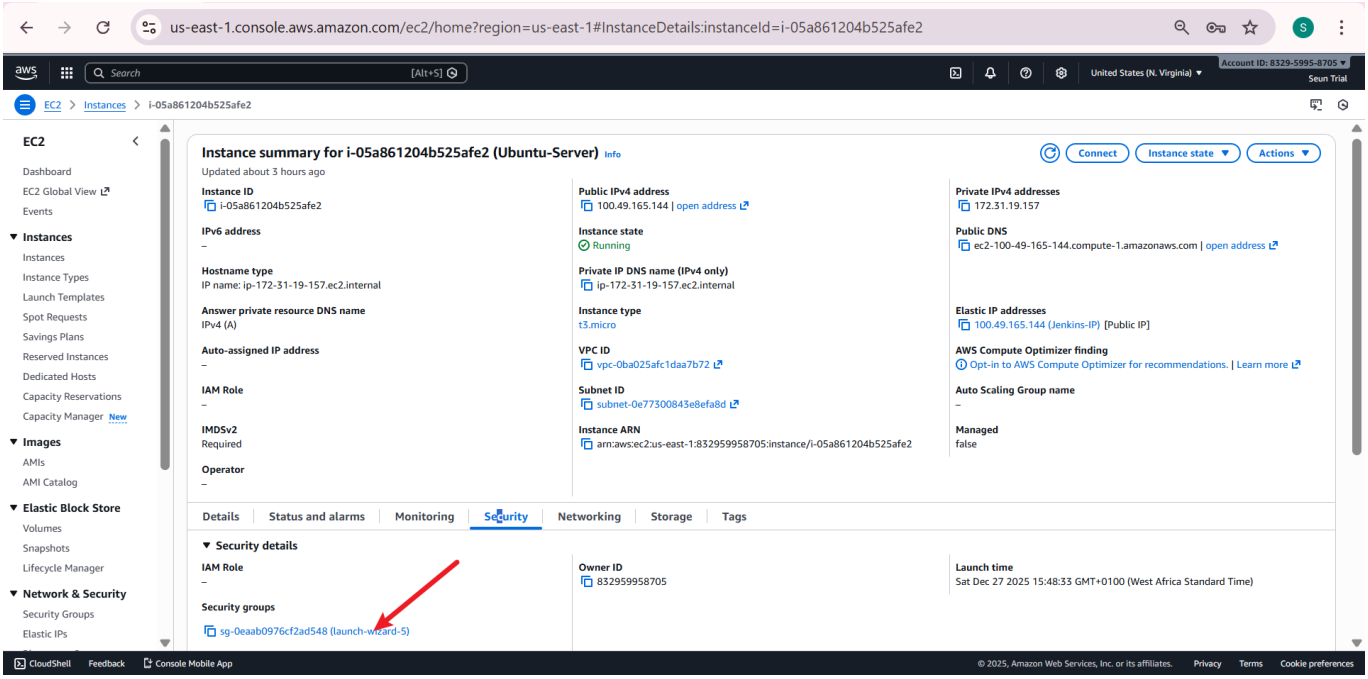
Step 27: Pipeline Finished Successfully

Pipeline execution completed successfully.



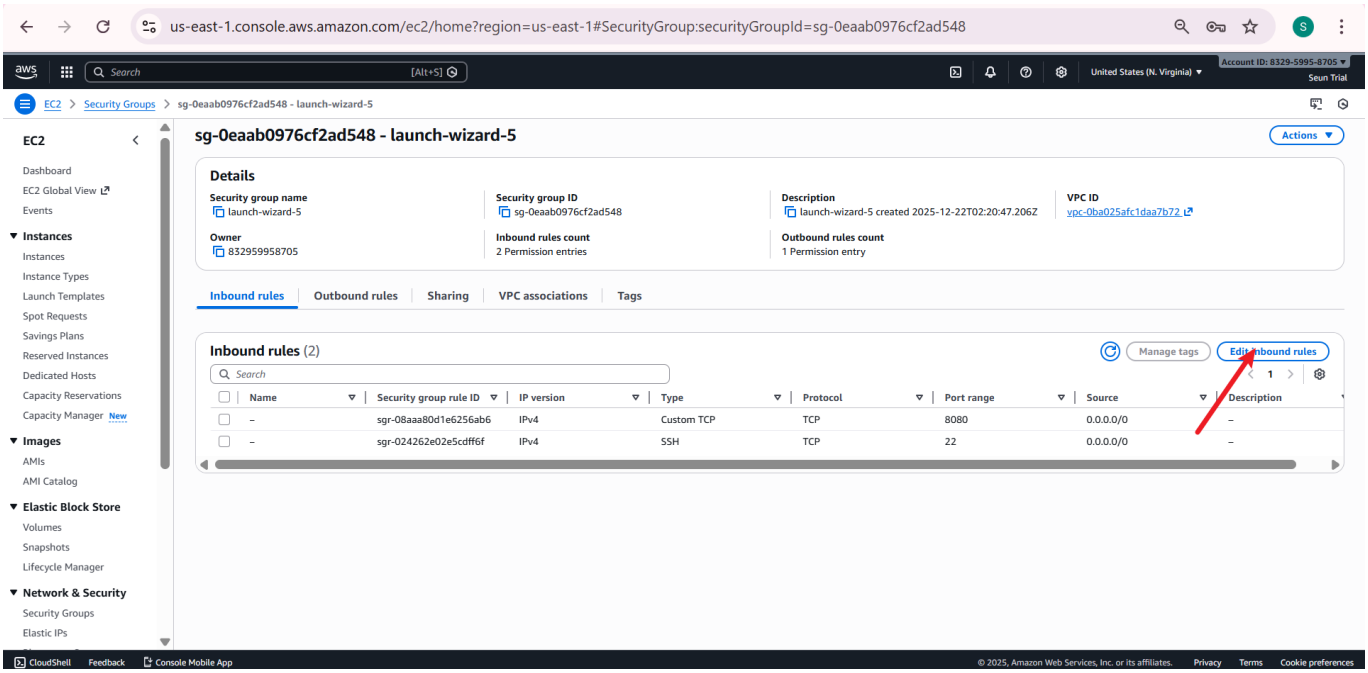
Step 28: Open Security Group

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



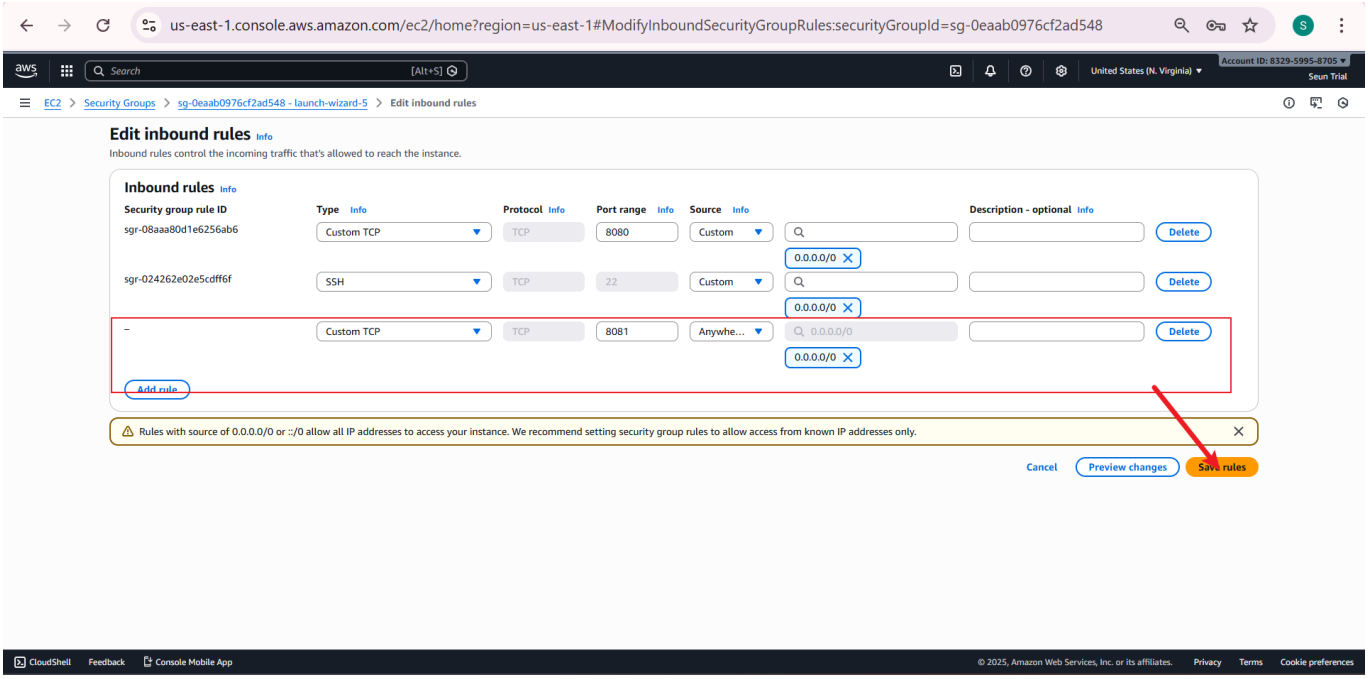
Step 29: Edit Inbound Rules

Click **Edit inbound rules**.



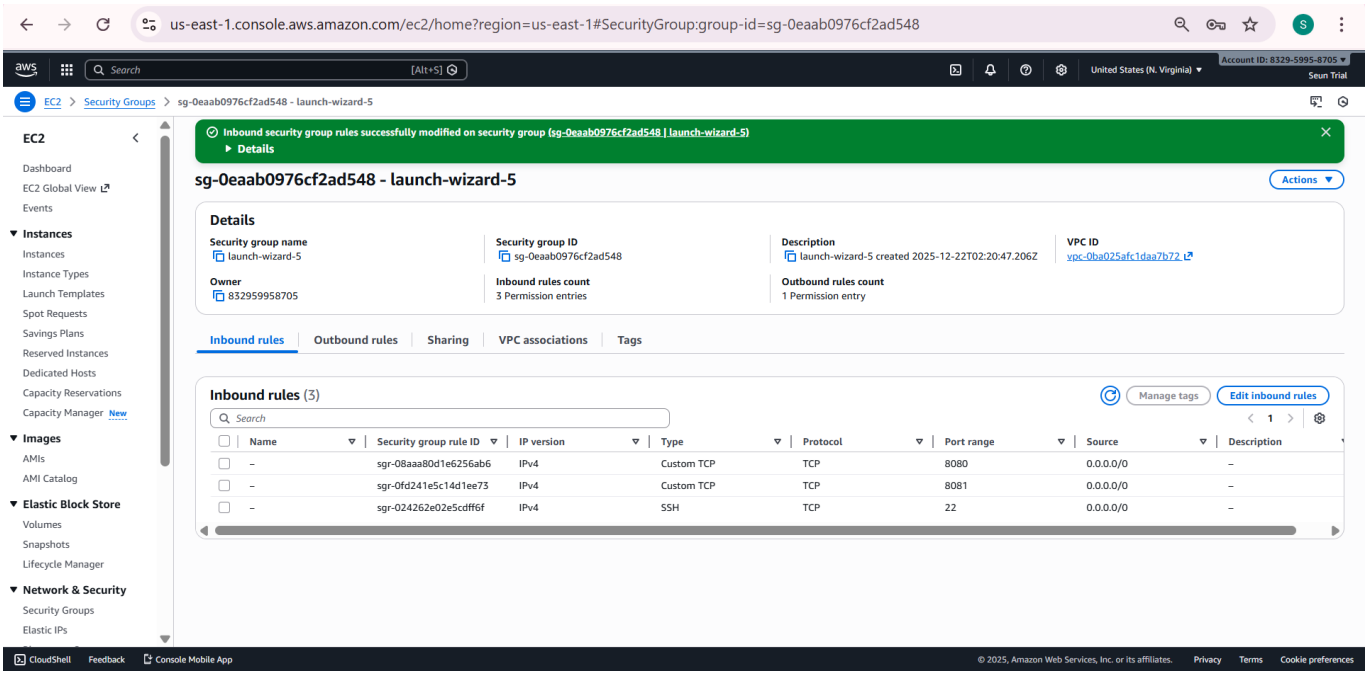
Step 30: Add Port 8081 Rule

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



Step 31: Inbound Rule Added Successfully

Inbound security group rules updated successfully.



Step 32: Access the Application

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

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

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