Introduction:

This documentation provides a comprehensive guide for setting up and managing infrastructure provisioning and continuous integration (CI) processes using Jenkins in both Windows and Linux environments. It is designed to help developers streamline their workflows, automate infrastructure management, and ensure efficient deployment pipelines.

Deployment:

**Refer to Github repo for example project + template:** [Hiverlab-Brian/flask\_docker\_jenkins\_example](https://github.com/Hiverlab-Brian/flask_docker_jenkins_example)

**Guide:** [Jenkins Handbook](https://www.jenkins.io/doc/book/), [Setting Jenkins CI for python application](https://mdyzma.github.io/2017/10/14/python-app-and-jenkins/)

Setting up Jenkins

# Install and setup Jenkins on windows machine

**Guide:** [Windows (jenkins.io)](https://www.jenkins.io/doc/book/installing/windows/)

1. Install Jenkins from [Windows Stable installer (jenkins.io)](https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable/)run Jenkins.msi
2. On <http://localhost:8080/>, follow the file path to file the initialAdminPassword to reset and create administrator user account for the Jenkins program
3. Install suggested plugins

# Install and setup Jenkins on VM environment

Recommended Configuration:

1. Java 17
2. 256MB free memory
3. 1GB+ free disk space

Run the following command:

sudo apt install openjdk-17-jre -y

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo 'deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/' | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update && sudo apt-get install jenkins -y

sudo service jenkins restart

1. On [http://<vm-ip-address>t:8080/](http://localhost:8080/), follow the file path to file the initialAdminPassword to reset and create administrator user account for the Jenkins program
2. Install suggested plugins

# Configuration

**Recommended Plugins to install: Dashboard > Manage Jenkins > Plugins**

1. Docker
2. Docker Pipeline
3. Cobertura

Take note to tick the checkbox “Restart Jenkins when installation is complete and no jobs are running” to have the Jenkins agent restart itself

This recommendation to provide a seamless test environment that allows the integration docker image within the CI/CD pipeline

Credentials Setup

**Guide:** [Using credentials (jenkins.io)](https://www.jenkins.io/doc/book/using/using-credentials/)

**Recommended** to use Credentials for any authorisation requirements:   
**Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted)**

1. Click on **Add Credentials**
2. Given a private github repository, ensure that a **Github Personal Access Token (**[**Personal Access Tokens (Classic)**](https://github.com/settings/tokens)**)** that is associated with one of its collaborator accounts has been created
3. 

Username: <Collaborator Account Username>

Password: <Collaborator Personal Access Token>

ID: <Meaningful name for credentials>

Description: <Meaning description>

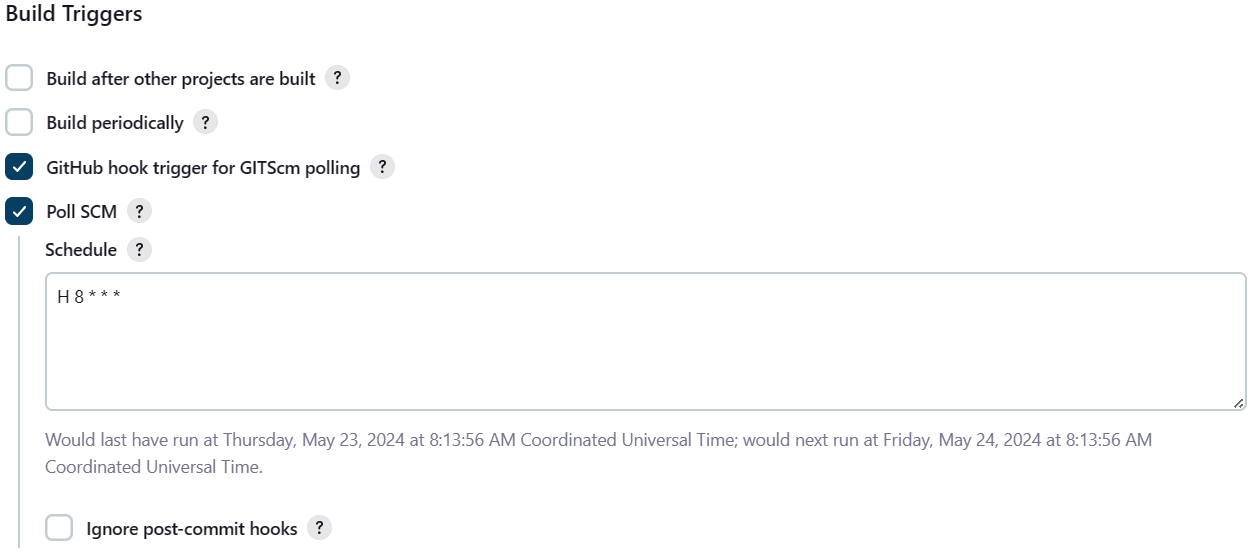
1. Save credentials

Implementing Jenkins pipeline in Github repository

**Guide:** [Working with projects (jenkins.io)](https://www.jenkins.io/doc/book/using/working-with-projects/)

**Recommended** to use appropriate **Pipeline** job according to design of **Github repository**:

For Example: **Using the Pipeline job**

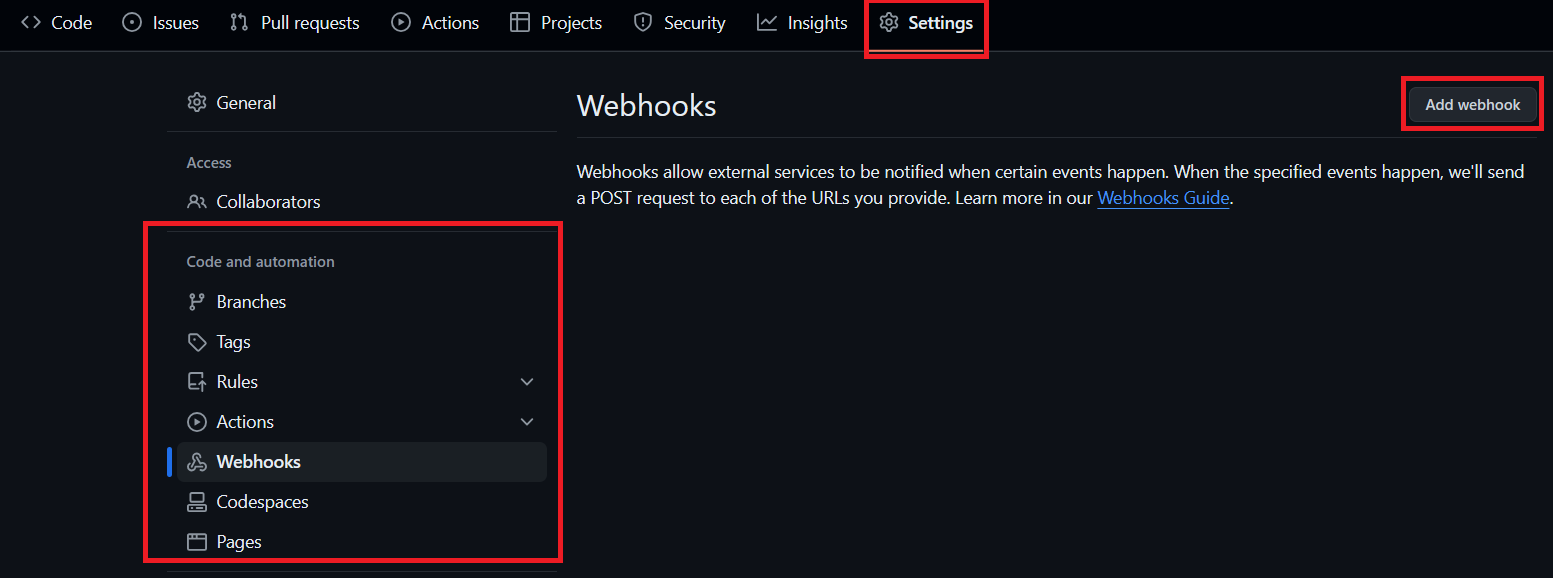
1. Click on **New Item** on the dashboard
2. Click on **Pipeline** and give an appropriate name <Demo Project>
3. Ensure discarding of old builds with **max # of builds to keep** limited to 5
4. Ensure **GitHub hook trigger for GITScm polling** is enabled. Github webhook integration can be found below  
   
5. Ensure that **pipeline** follows **Pipeline script from SCM** > **SCM** follows **Git**
6. Input **Repository URL** and accompanying **Credentials** that was created earlier



1. Change **Branch Specifier** to \*/main
2. Ensure **Script Path** is **Jenkinsfile**

Github Webhook integration

1. Within the chosen github repository, in the settings page’s code and automation section, click add webhook



1. Set payload URL as the <jenkins\_url>/github-webhook/
2. Set the ‘Pull Requests’ and ‘Push’ as the trigger for this webhook
3. A tick should appear with the message ‘Last delivery was successful’
4. The Jenkinsfile would need to be configured to in order to complete webhook integration

Configuring Jenkinsfile

**Guide:** [Using a Jenkinsfile](https://www.jenkins.io/doc/book/pipeline/jenkinsfile/)

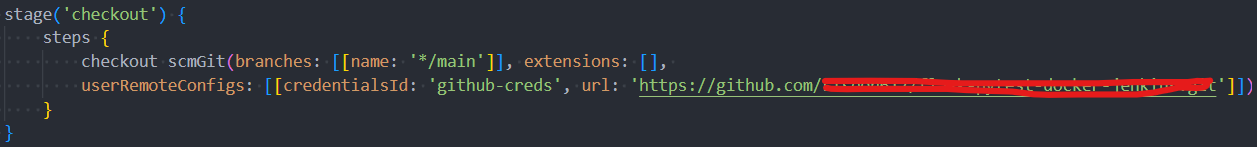
Using the Pipeline Syntax generator at the bottom of the page and the template provided by the guide, we will configure the Jenkinsfile for the project. Generally a Jenkinsfile is broken up into several stages

# Checkout Stage: Finishing up Github webhook integration:

1. Choose checkout: Check out from version control and configure the Git SCM section same as what was done previously

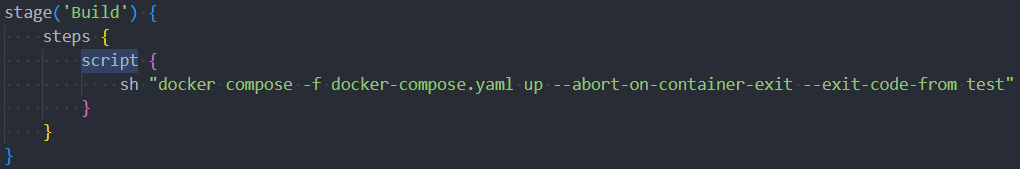


1. Generate Pipeline Script and copy the output script, placing it in the Checkout step



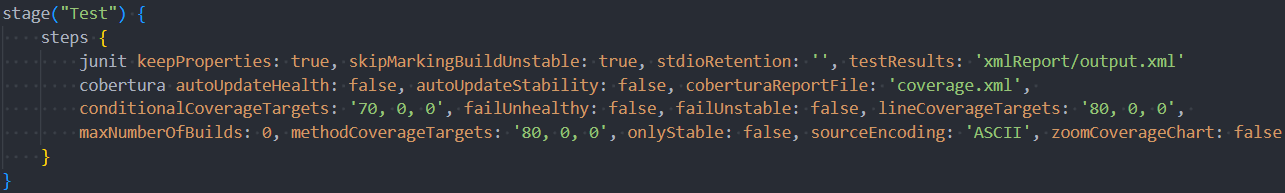
# Build Stage:

Building and running the project to ensure that it runs when built when new commits are made



# Test Stage:

Testing would consist of the various forms of testing implemented within the project, alongside the verification methods to ensure the project is fully tested



# Deploy Stage:

Deployment would need to be configured to users specification, either using Nginx configured for application deployment or pulling from docker hub. This stage would cement the Continuous Deployment (CD) portion of the pipeline

# Post Block:

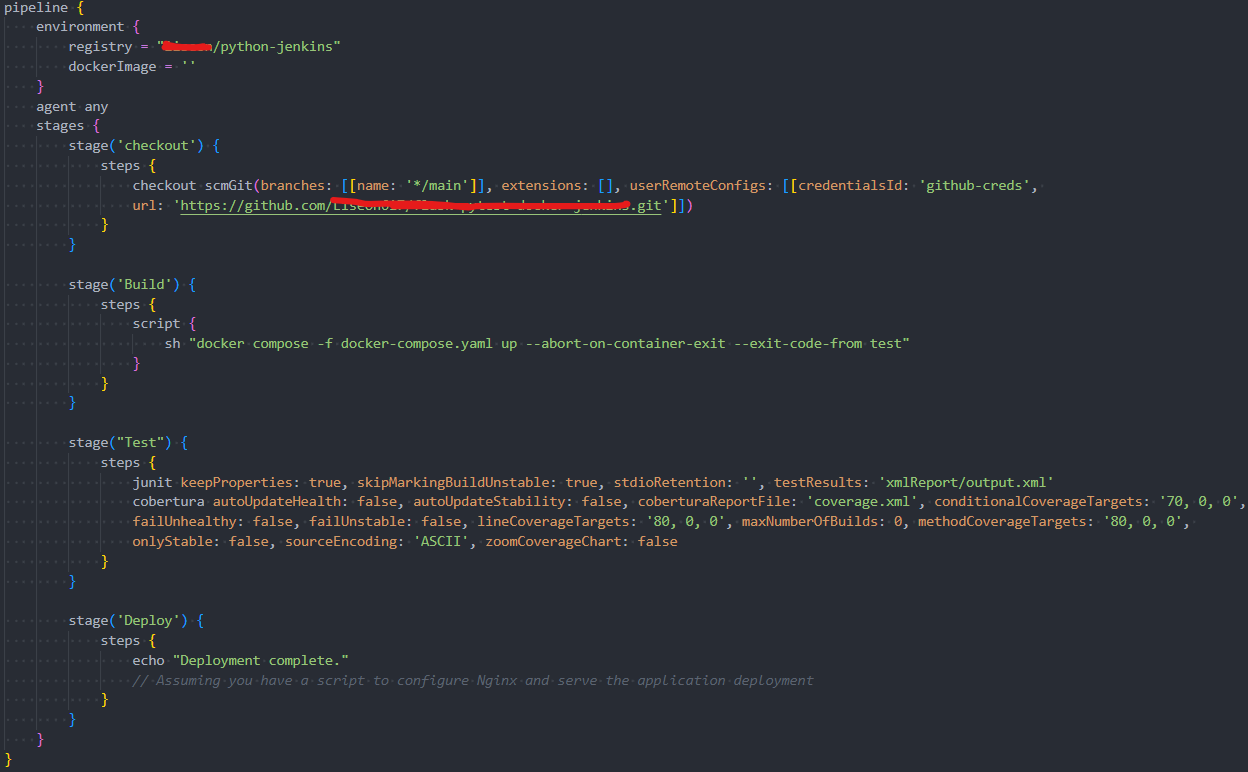
This block is used to handle actions that should be taken after the pipeline execution. It can include three common conditions: ***`always`***, ***`success*`**, and ***`failure`***.

***`always`*:** executes regardless of the pipeline outcome, useful for tasks like archiving logs or cleanup.

**`*success`*:** runs only if the pipeline is completed without errors, ideal for archiving successful builds.

***`failure`*:** activates if the pipeline fails, which is useful for notifying the team about issues via email or Slack, and potentially rolling back to the last successful build if integrated with the deployment process.

# Jenkinsfile example



The full example template for Jenkinsfile of a flask + docker project can be found in the repository