INDUSTRIAL ASSISMENT-03

Definition

CT and CD stand for continuous integration and continuous delivery/continuous deployment In very simple terms, Cl is a modern software development practice in which incremental code changes are made frequently and reliably, Automated build-and-test steps triggered by Ci ensure that code changes being merged into the repository are reliable. The code is then delivered quickly and seamlessly as a part of the CD process, in the software world, the CI/CD pipeline refers to the automation that enables incremental code changes from developers desktops to be delivered quickly and reliably to production.

What are the benefits of CI/CD?

- Automated testing enables continuous delivery, which ensures software quality and security and increases the profitability of code in production.
- CI/CD pipelines enable a much shorter time to market for new product features. creating happier customers and lowering strain on development.
- The great increase in overall speed of delivery enabled by CI/CD pipelines improves an organization's competitive edge.
- Automation frees team members to focus on what they do best, yielding the best end products.
- Organizations with a successful CI/CD pipeline can attract great talent. By moving away from traditional waterfall methods, engineers and developers are no longer bogged down with repetitive activities that are often highly dependent on the completion of other tasks.

Configuring Automated CI/CD with Jenkins & GitHub-Step by Step

Configuring automated CI/CD with Jenkins and GitHub is a simple and straightforward process and can help automate the entire workflow. Integrating Jenkins with GitHub enables the developers to pull the source code from any Git repository in a hassle-free manner.

Furthermore, GitHub also supports bi-directional integration, which will automatically initiate a trigger to Jenkins every time there is a change in the GitHub repository.

The step-by-step procedure for configuring Jenkins with GitHub is elaborated below.

+ Related Resource Learn more about **Selenium + GitHub: A Step-by-Step Testing Guide**

Installation

To configure automated CI/CD with Jenkins, the first and the most obvious step is to install Jenkins Jenkins setup, with complete instructions on how to install it, is available here

To implement it on the server or a remote computer, simply replace localhost with the IP Address of the target machine. Jenkins provides long-term support and weekly releases for several Operating Systems. For the scope of this blog we will be setting up Jenkins on a focal Ubuntu machine. To use the Ubuntu repository, execute the command in the terminal

```
wget -q-O-https://pkg.jenkins.io/debian-stable/jenkins.io.key sudo apt-key add
```

After that, add the Jenkins repository by executing this command

```
sudosh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/>

/etc/apt/sources.list.d/jenkins.list
```

Make sure to update the packages using the sudo command as shown here:

```
sudo apt-get update
```

Finally, install Jenkins:

```
sudo apt-get install jenkins
```

Make sure to install Java Runtime Environment (JRE) explicitly after installing Jenkins However, if you get a "package error, you must check for the Java version by executing the command given below:

```
Java-version.
```

If the package does not exist, then you will have to install it. For this, you will first search for the available packages.

```
sudo apt search openjdk
```

Next, pick the available version and install it.

```
sudo apt install openjdk-11-jdk
```

After the successful installation, check the Java version.

```
java-version
```

You will get a result like this:

```
openjdk version 11.0.7-ea" 2020-04-14

OpenJDK Runtime Environment (build 11.0.7-ea+9-post-Debian-1)

OpenJDK 64-Bit Server VM (build 11.0.7-ea+9-post-Debian-1, mixed mode, sharing)
```

The second step is to run Jenkins However, you can also run Jenkins on several platforms including Docker, Kubernetes, and Windows Notice here that you will need to iptali Docker or Kubernetes if you are planning to run Jenkins on these platforms to this tutorial, we will run Jenkins on localhost on Linux.

Starting the Server

After the successful installation of Jenkins, you will start it on localhost. For this step, youshould start Jenkins first using the statement command on the command line

sudosystemctl start jenkins

sudosystemctl enable jenkins

To check the status of Jenkins service, use the status command as shown below.

sudosystemctl status jenkins

The output will be rendered as "Active" if everything has been configured successfully.

Post-Installation Setup

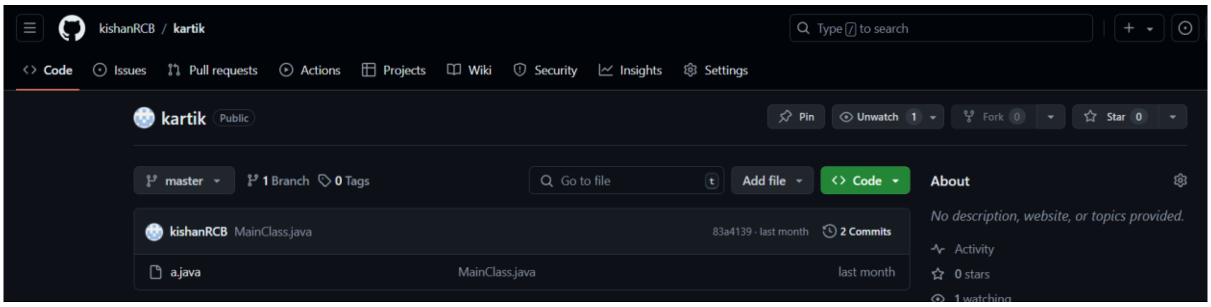
When you configure Jenkins for the first time, it is crucial to unlock it using an automatic password. For this setup, typehttp://localhost:8080 in the browser, and wait for the Getting Started page to appear The automatically-generated password stored in /var/jenkins_home/secrets/initialAdmin Password file. The file can be also accessed via the command-line interface using the cat command with the filename

Copy/paste the password in the Administrator Password field and click Continue. The dialogue box will close and that's it! You have successfully set up the user. After that, you will get the Customize Jenkins page. From this page, you can install any number of plugins you require. You can install-Suggested plugins as they are selected by default and you can also choose any additional plugins Make sure to install the Git Plugin.

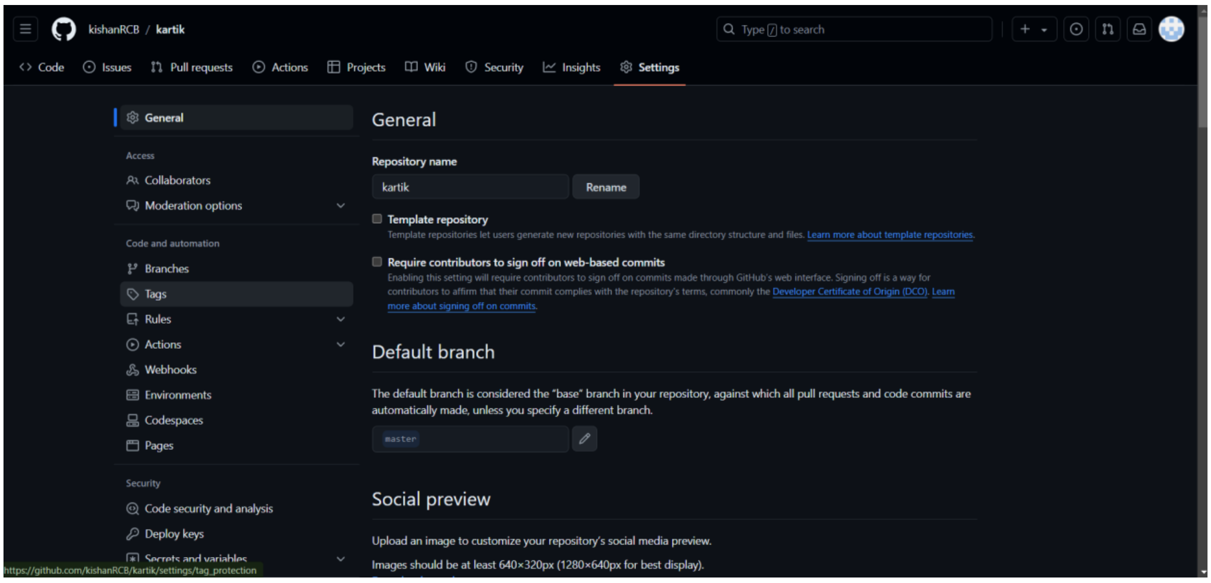
Configure GitHub

Now that you have successfully configured the user, it is time to configure the Github repository using Webhooks. Additionally, you can aho perform the step after adding the Jenkins project.

Head over to the existing GitHub repository, and click the Settings tab



After that click on the Webhooks option from the navigation panel on the left side of the screen.



General

Repository name: kartik

Template repository
Template repositories let users generate new repositories with the same directory structure and files. Learn more about template repositories.

Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

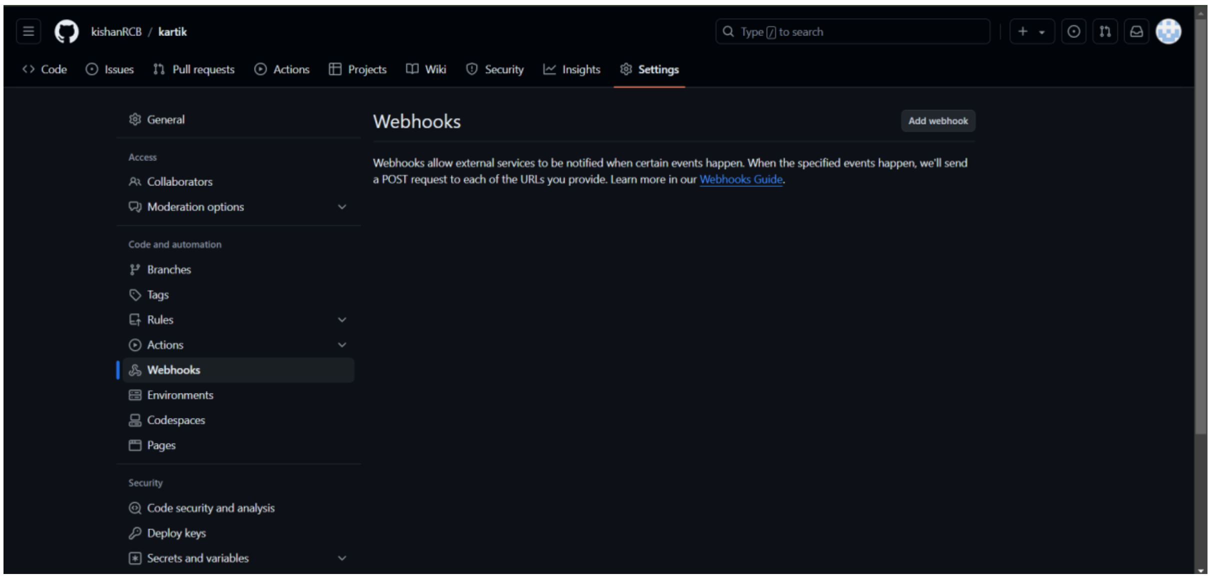
master

Social preview

Upload an image to customize your repository's social media preview.

Images should be at least 640×320px (1280×640px for best display).

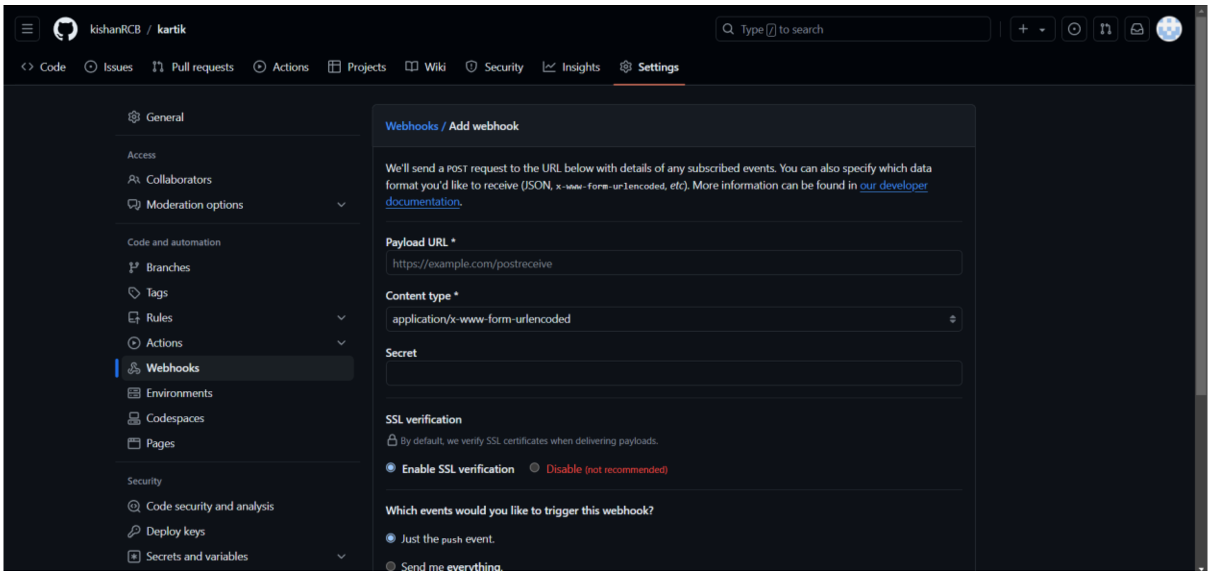
Now, click on the Add webhook button on the right side of the screen.



Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

You will see a new form appear in the wethooks section in the Payload Utt field, type in the Jenkins endpoint in our case, it is localhost: 5080 Since the localhost is behind a Firewall, make sure to make it visible to your public GitHub repository by using a webbook proxy service. In our case, we have used SocketXP. If you have a specific URL, make sure that it is publicly availatile. At the end of the URL type github webhook/



Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *

https://example.com/postreceive

Content type *

application/x-www-form-urlencoded

Secret

SSL verification

By default, we verify SSL certificates when delivering payloads.

[✓ Enable SSL verification] Disable (not recommended)

Which events would you like to trigger this webhook?

[✓ Just the push event.]

Send me everything.

Choose application son as Content Type from the drop down menu, select just the push event, and click the Add webhook button

For now, this repository will only call Jenkins on the pash even other events, you will have to select the Let me select individual events option instead of Just the push event

That's all! You are done with the configurations on the Github side

Jenkins Project and Adding GitHub

After successfully logging in, the Jenkins dashboard will appear on the screen Hero, all the CI/CD pipelines and their summary are visible to a user. O the Jenkins dashboard, click on "New Item". After that, enter the name of your project, choose "freestyle project" the list, and hit the enter key.

The next step is to configure the project on the Jenkins dashboard. In the tieneral tabs, choose your GitHub Project, and enter the GitHub Repository URL. Now, head over to the Source Code Management tab to add the credentials.

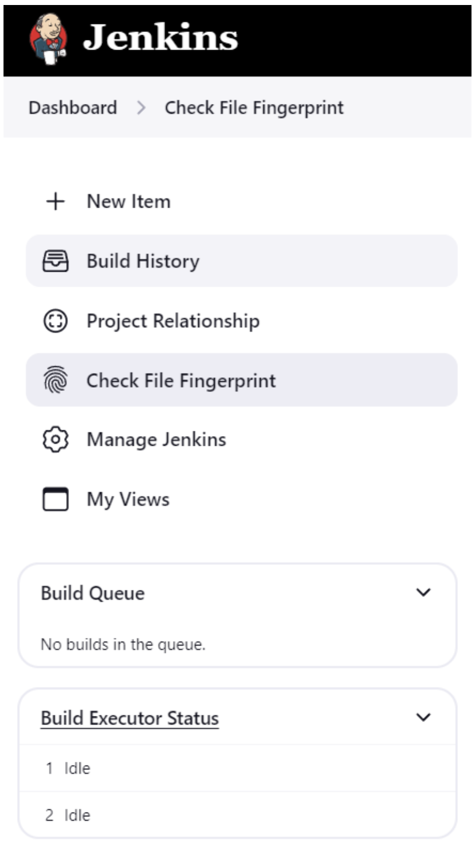
For this step, press the Add button and type Username and Password. Press Add again to close the dialogue box. Be sure to select the main branch in the Branch Specifier.

Now, head over to the Build Triggers section to set the triggers for Jenkins. The purpose of triggers is to necessarily indicate to Jenkins when to build the project Select Poll SCM section, which queues VCS on the predefined schedule. The predefined schedule is which represents every minute. That means, our Jenkins job will check for the change in our repository every single minute.

Build Jenkins Project

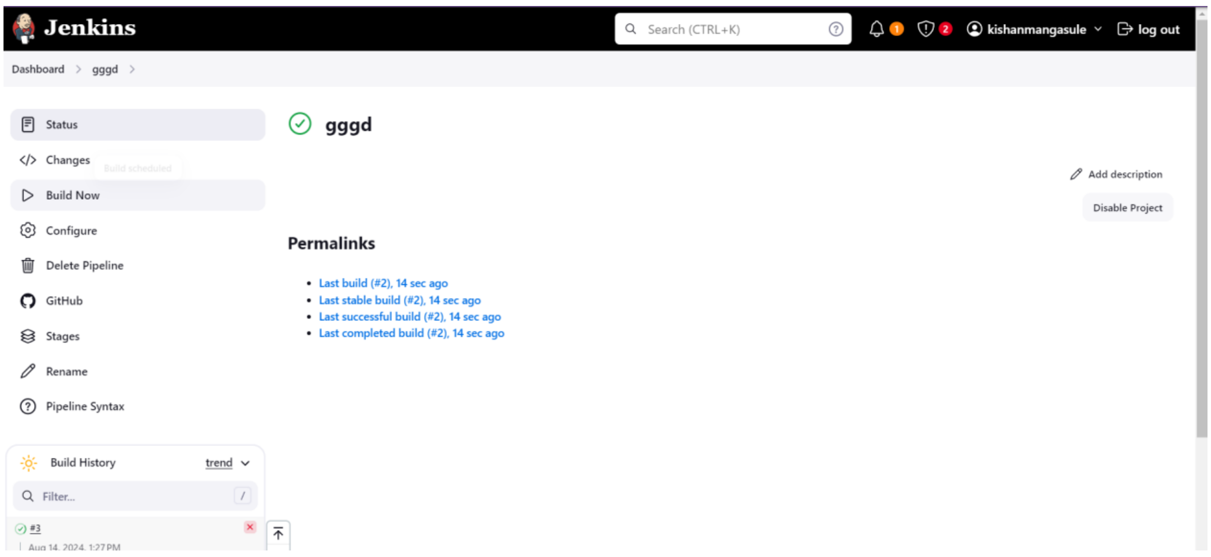
Now go back to the repository that you configured, and make a sample file, or edit any existing file and commit the changes,

After that, head over to Jenkins and click on the "Build Now" button from the navigation bar.



No builds in the queue.

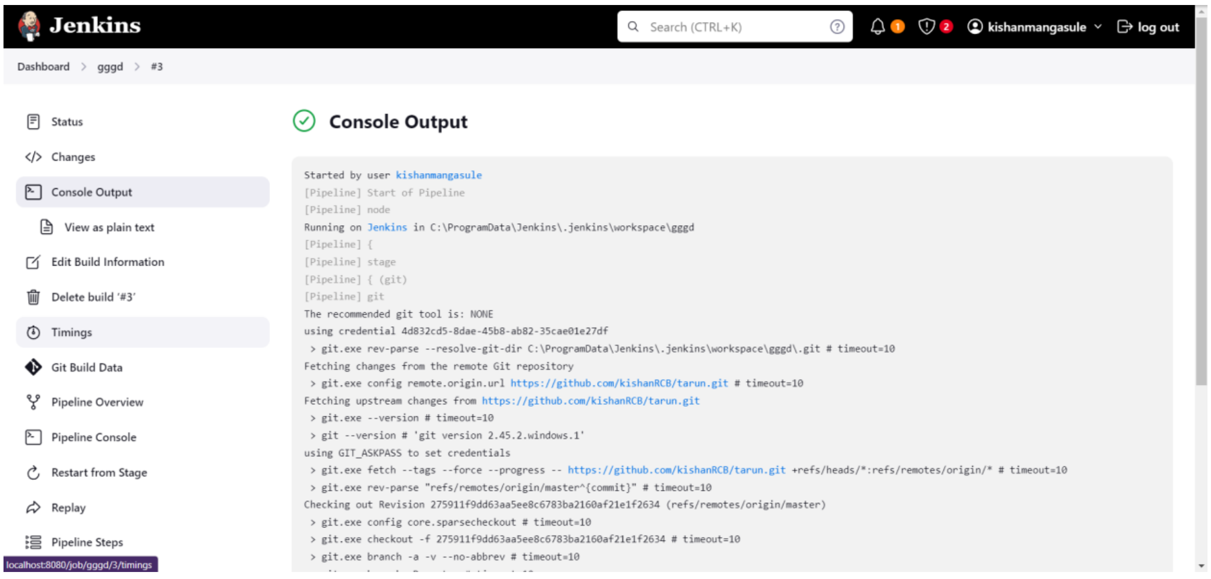
The build number will display a green tick if the build is successful and the app will be published application on the localhost:8080 You can also use the output console todebug the application.



- Last build (#2), 14 sec ago
- Last stable build (#2), 14 sec ago
- Last successful build (#2), 14 sec ago
- Last completed build (#2), 14 sec ago

#3
Aug 14, 2024, 1:27 PM

As you can see. Jenkins pulled the changes from the GitHub repository and reflected them in its console.



Console Output

```
Started by user kishanmangasule
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\gggd
[Pipeline] {
[Pipeline] stage
[Pipeline] { (git)
[Pipeline] git
The recommended git tool is: NONE
using credential 4d832cd5-8dae-45b8-ab82-35cae01e27df
 > git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\gggd\.git # timeout=10
Fetching changes from the remote Git repository
 > git.exe config remote.origin.url https://github.com/kishanRCB/tarun.git # timeout=10
Fetching upstream changes from https://github.com/kishanRCB/tarun.git
 > git.exe --version # timeout=10
 > git --version # 'git version 2.45.2.windows.1'
using GIT_ASKPASS to set credentials
 > git.exe fetch --tags --force --progress -- https://github.com/kishanRCB/tarun.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
Checking out Revision 275911f9dd63aa5ee8c6783ba2160af21e1f2634 (refs/remotes/origin/master)
 > git.exe config core.sparsecheckout # timeout=10
 > git.exe checkout -f 275911f9dd63aa5ee8c6783ba2160af21e1f2634 # timeout=10
 > git.exe branch -a -v --no-abbrev # timeout=10
```

Finally, you have successfully integrated Jenkins with the GitHub repository With each push, commit, or update, GitHub will trigger the Jenkins job in tum will execute the necessary steps to deploy the changes.