Lab 4.1. Setting Up Continuous Integration with Jenkins (Additional Topics)

Optional: Integrating GitHub with Jenkins

***WARNING**: This will only work if your Jenkins server is available publicly and can be accessed from GitHub. If you are running Jenkins on your local machine (e.g. laptop) with a private IP space, this will **not** work unless you use some method to expose your local machine to the public internet. This is beyond the scope of this course, but you can look into ngrok, boring proxy, or localtunnel as possible solutions. Be aware this will have security implications.*

By the end of this exercise, you should be able to:

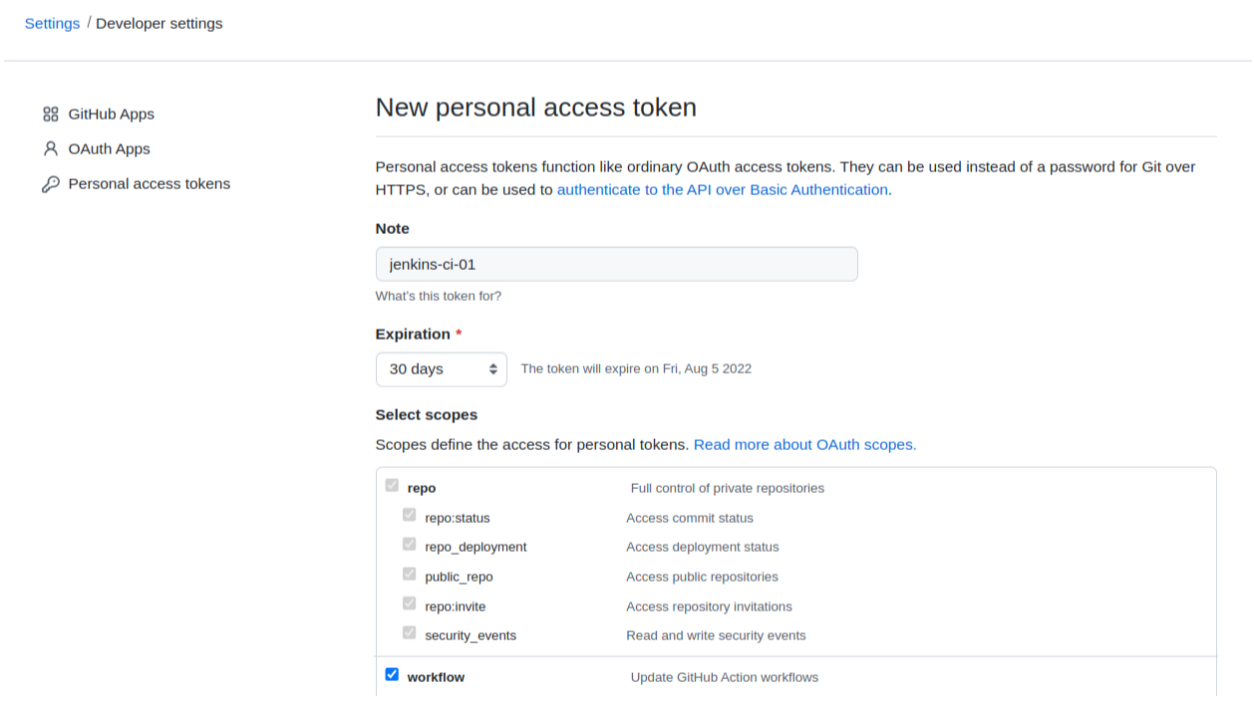
- Integrate GitHub with Jenkins.
- Trigger builds automatically using GitHub webhooks.

Setting Up Build Triggers Using GitHub Webhooks

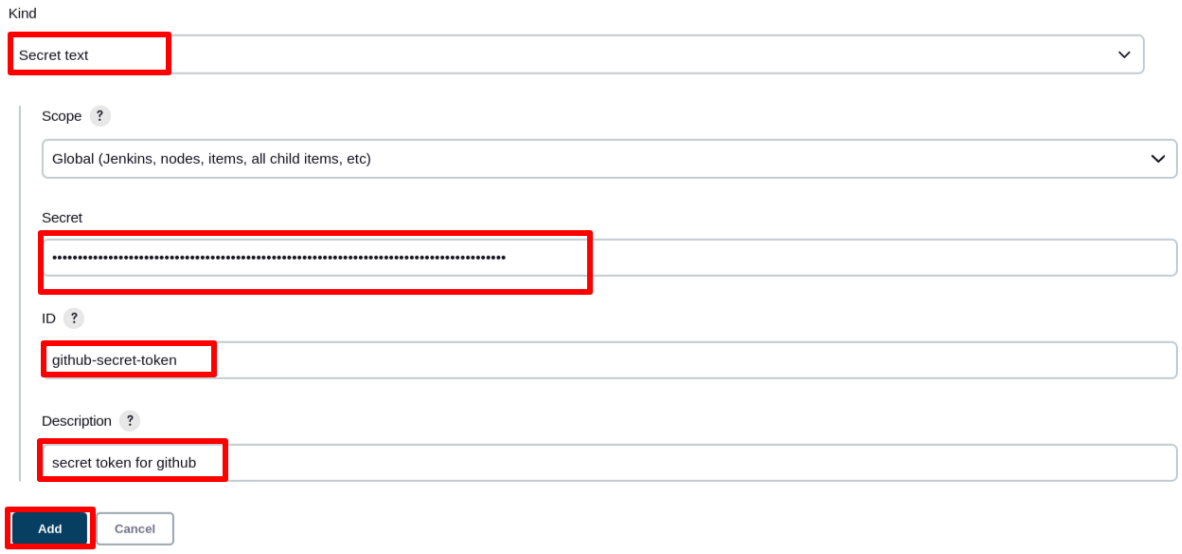
You can trigger your job when there is a commit change in the master branch on GitHub. Webhooks will trigger your job from the GitHub side and you can send the status back to GitHub, so you don't need polling from Jenkins.

Follow these steps to set up the Webhook.

To set up a GitHub token, go to the user page and select ***Settings > Developer settings > Personal access token***. In the **Personal access token** page select **Generate new token**, add a note, check *all* boxes, and generate the token. Once you generate your token, save it somewhere immediately because you can't view it again.

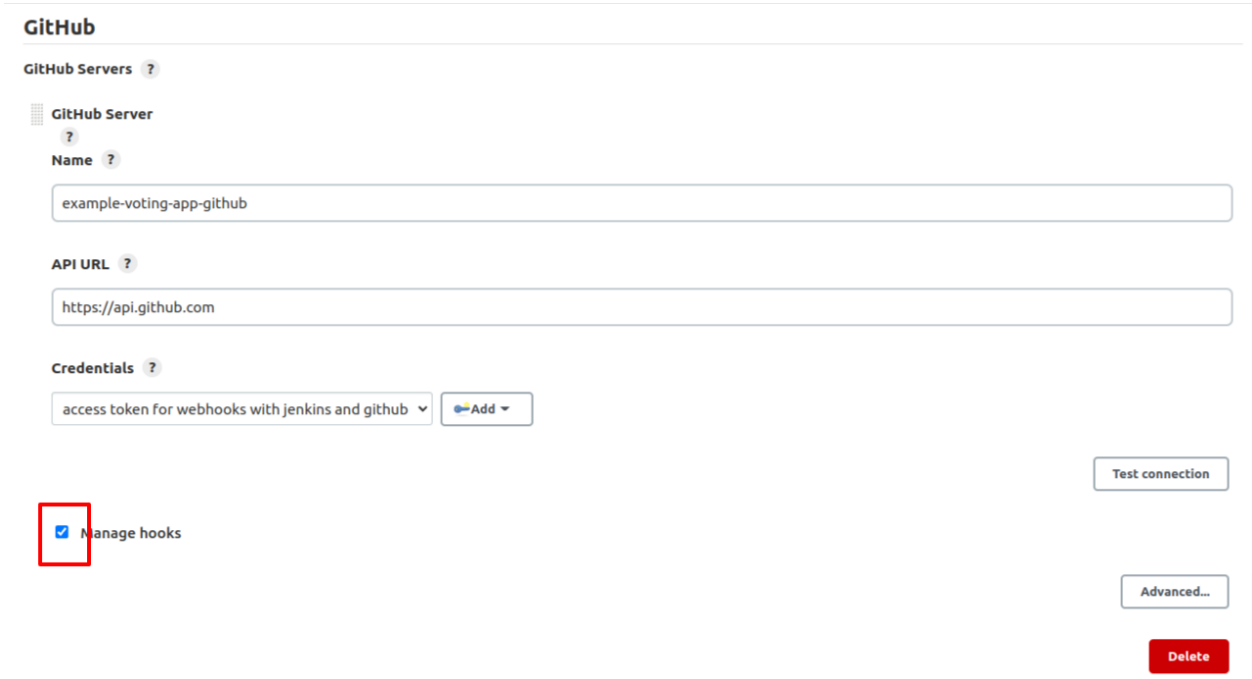


Now go to **Jenkins page > Manage Jenkins > Configure System**. Scroll to **GitHub** and click **Add GitHub Server**. Provide any name for the server. This does not matter. Click **Add** under **Credentials** which brings up the following:

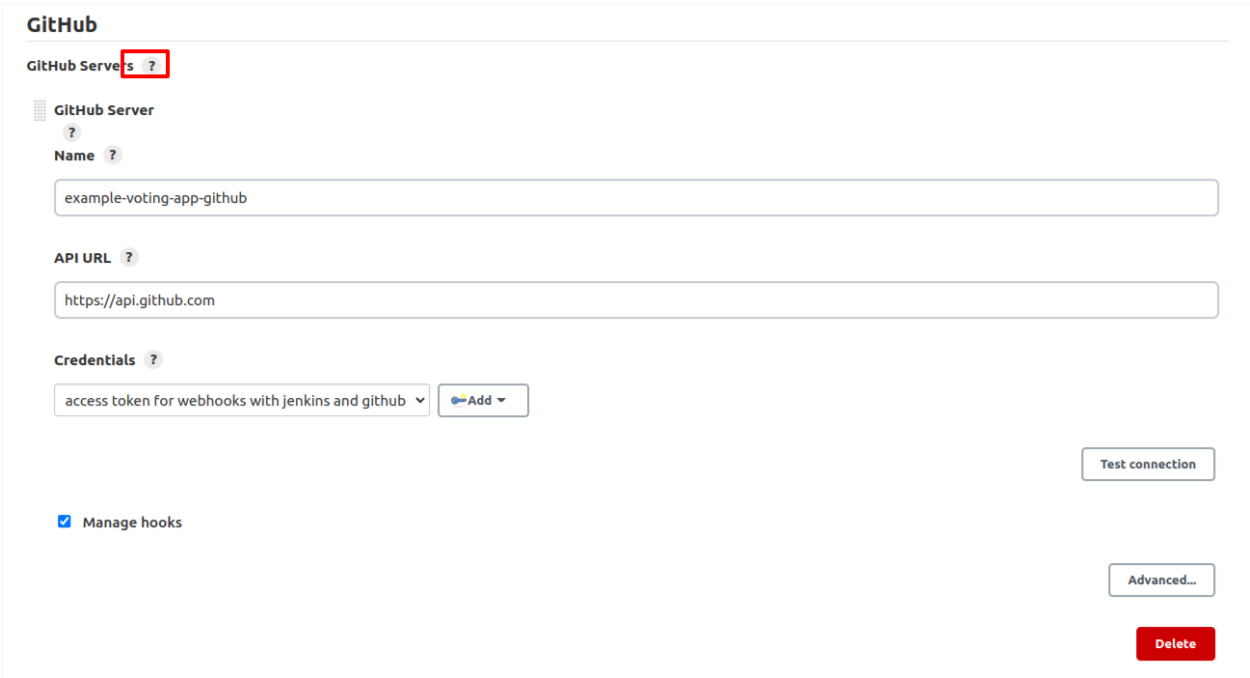


The secret is your GitHub access token that you generated. Provide an id description. Save it.

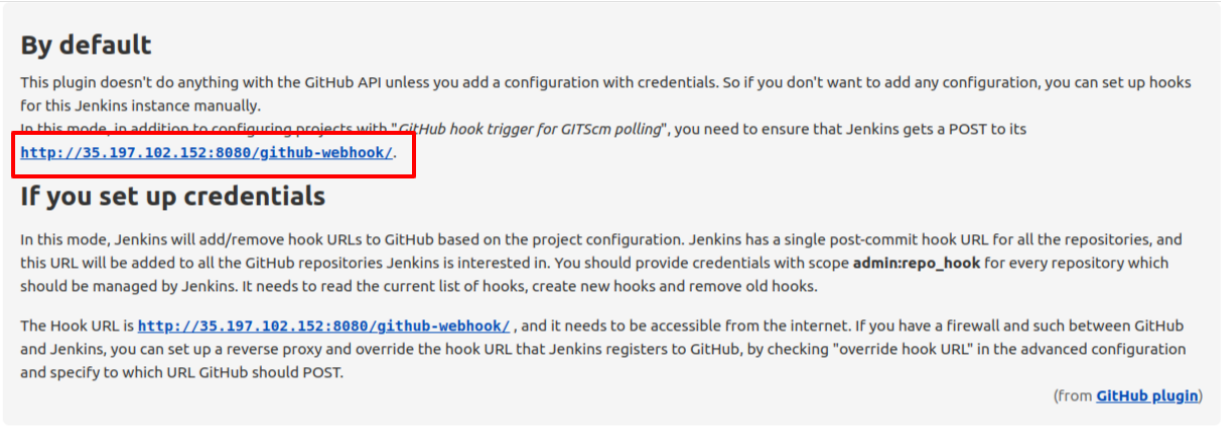
Choose “secret token for github” under credentials and check the **Manage hooks** box. Save the configuration.



Now click the question mark next to **GitHub Server**:

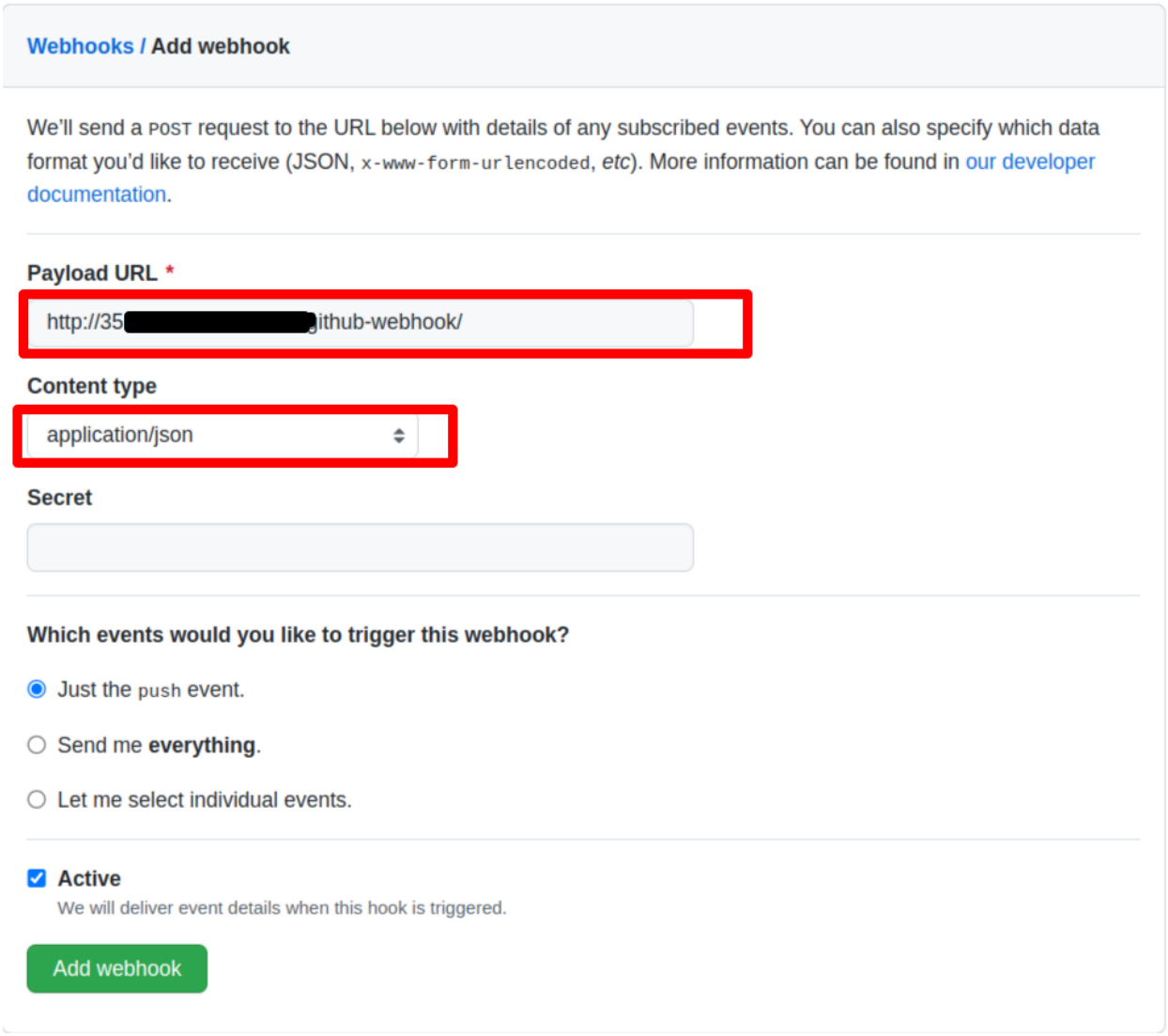


Copy the web hook address and save it for later. We will need to give this GitHub in a few steps:



Go to your **`worker-build`** configuration page, choose “github project” and provide the repository URL. Remove **Poll SCM** and choose **GitHub hook trigger** for gitscm polling, save the changes.

Again go to the GitHub repository and choose ***Settings > Webhooks***. Add “Payload URL” and “Content type” as “application/json”. Save it (refer to the example image given below).



Now go to the Git repository, add some file in your repository and commit the changes.

Once you make the commit in your repository, it will automatically trigger your build. You can see it by using the build pipeline you created earlier.

This is how you integrate GitHub with Jenkins.

Optional: Adding Jenkins Status Badges to GitHub

***WARNING**: This only works if the Jenkins server is accessible from GitHub.*

By the end of this exercise, you should be able to:

- Set up two way communication between Jenkins and GitHub.

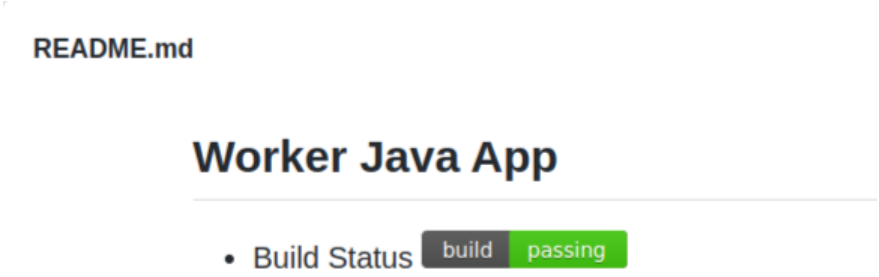
- Send the status of the build from Jenkins to GitHub and display it as a badge.

You need to install the **Embeddable Build Status** plugin from *Manage Jenkins > Manage Plugin > Available*. Once the installation is done, you can see **Embeddable build status** on every job page.

Follow the next steps to add the Jenkins status badge to GitHub.

Go to the `worker-build` job page, select **Embeddable Build Status**. On the **Embeddable Build Status** page, scroll to **Links** and under **Markdown** copy the **unprotected** link. Paste it in your GitHub repository `README.md` file which you have created in the previous lab and commit the changes.

Once you commit the changes, you should see the build status being listed as passing on your `README.md` file.



Go to the `worker-build` configuration page. Under the **Build** section change the root file to `pom.xml` instead of `worker/pom.xml` and save the changes.

Build your `worker-build` job; it will fail your build. Now go to the GitHub repository page. There you can see the build's failed status on the `README.md`.

Correct the `worker-build` by switching `pom.xml` back to `worker/pom.xml`.

If you wish, you can add the embeddable build status unprotected markdown link of `worker-test` and `worker-package` to your GitHub repository `README.md` file as well.

This is how you add the Jenkins status badge to GitHub.

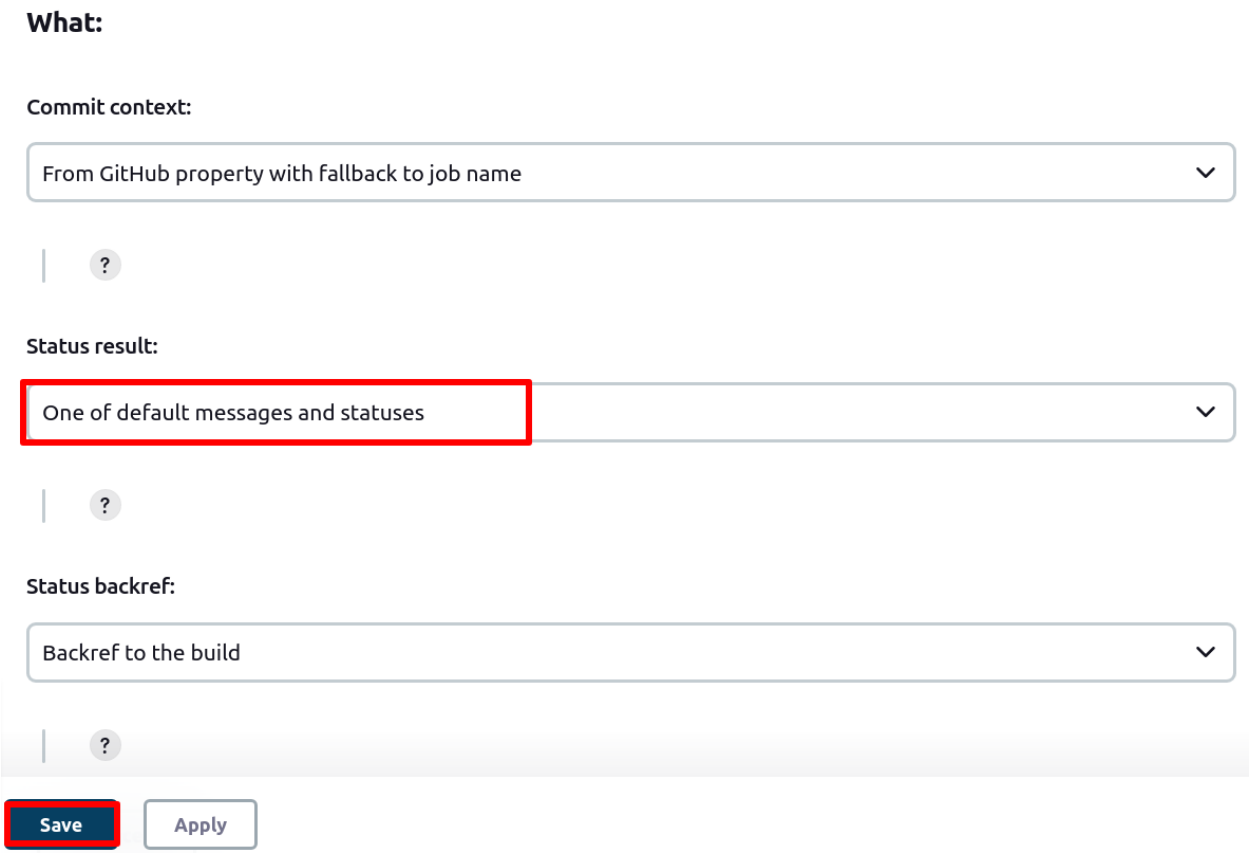
Optional: Configuring Job Status with Commit Messages

By the end of this exercise, you should be able to:

- Add build and test job statuses to GitHub commit messages.

Follow these steps to configure the job status.

Go to your `worker-build` job configuration page, under **Post-build Actions**, choose "Set GitHub commit status (universal)" and set the **What > Status Result** to *One of default messages and statuses*. Save the changes.



Go to the `worker-test` configuration page and paste the GitHub project URL into the **Source Code Management** section. Add the post-build action "Set GitHub commit status (universal)" in the same way that you did for the `worker-build` job. Be sure **What > Status** is set to *One of default messages and statuses*.

Now go to your `worker-test` job page. There, select the **Embeddable Build Status**. In the **Embeddable status** page, copy the markdown unprotected link under **Links** and paste it on your GitHub repository `README.md` file. Add `subject=Unittest` to your `worker-test` link as follows:

The unprotected markdown link will look similar to the following:

```
[![Build
```

```
Status](http://IPADDRESS:8080/buildStatus/icon?job=job-02)](http://IPA
DDRESS:8080/job/job-02/)
```

You need to add `subject=Unittest` like so:

```
[![Build
Status](http://IPADDRESS:8080/buildStatus/icon?job=job-02&subject=Unit
Test)](http://IPADDRESS:8080/job/job-02/)
```

Refer to the example image given below.

Once you commit the changes, it will automatically build the triggers. You can see the builds in the pipeline view and the status will be updated on GitHub.

If you check your commit messages, it shows them as check-marked. You can see all the current build statuses with build number on the git commit message. If you click **Details** in the commit message, it will take you to the Jenkins page.

This is how you configure the job status and `git commit` messages.