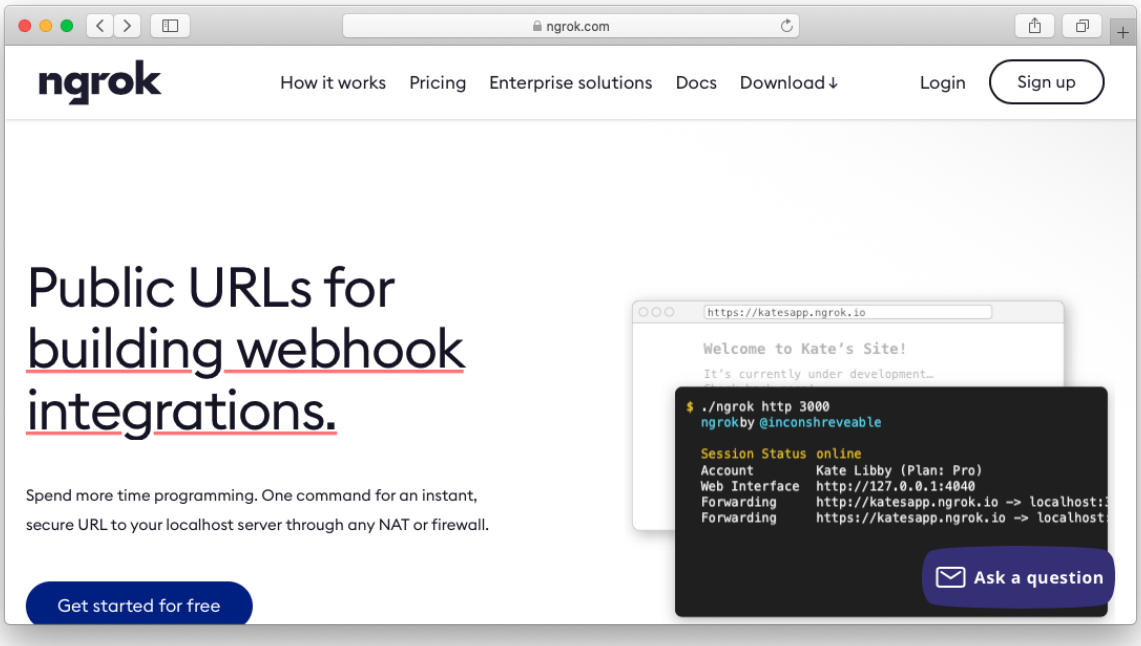
Integrating GitHub Webhooks with Jenkins to automate unit and integration test after GitHub events for CI/CD *

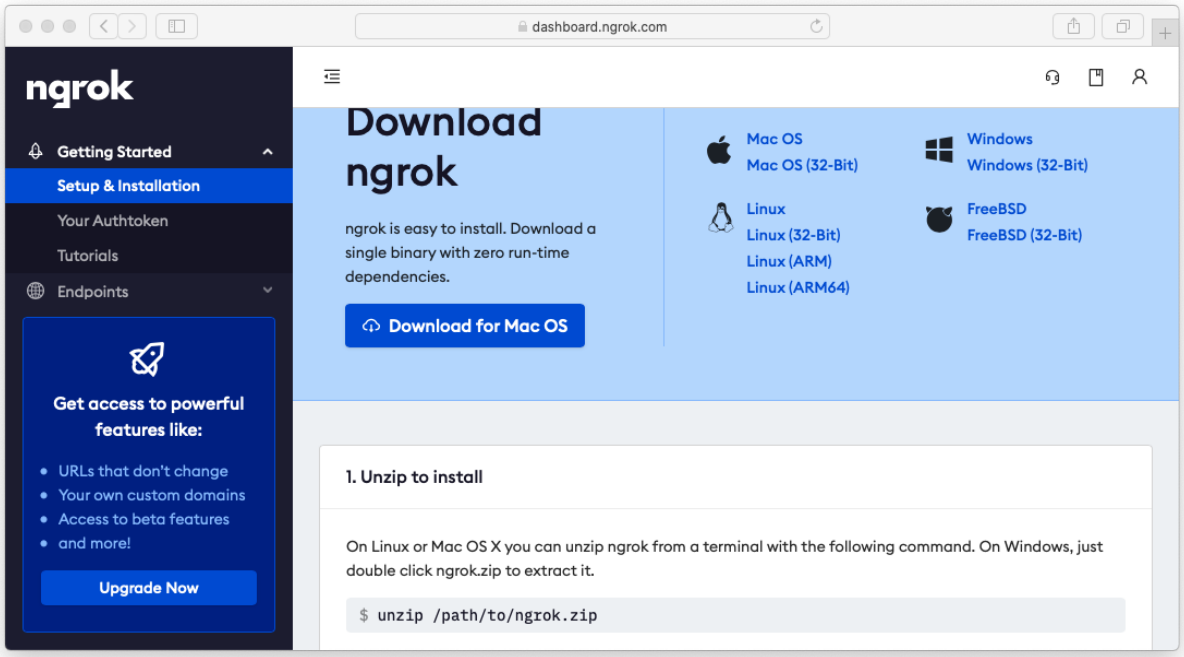
1. Exposing Jenkins to the Internet

Before beginning, this step is optional if you already have a Jenkins server exposed to the internet. If so, continue to section two, otherwise keep reading. Ok, so you are working with a local Jenkins running in some local environment, outside of any cloud provider such as Amazon EC2, and you want to use this local Jenkins server to test some workflows that can be implemented in a future formal environment without having to think on closing the cloud instance to save money. But still, you need a way to expose Jenkins to outside networks since this is a requisite to work with external services such as Webhooks.

To do so, we are going to use a service called: ngrok. Ngrok can expose local servers to the internet through a public IP. The free tier can work with one online ngrok process and up to 4 tunnels. In this case, we only need one for Jenkins, so the free tier is great for our needs.

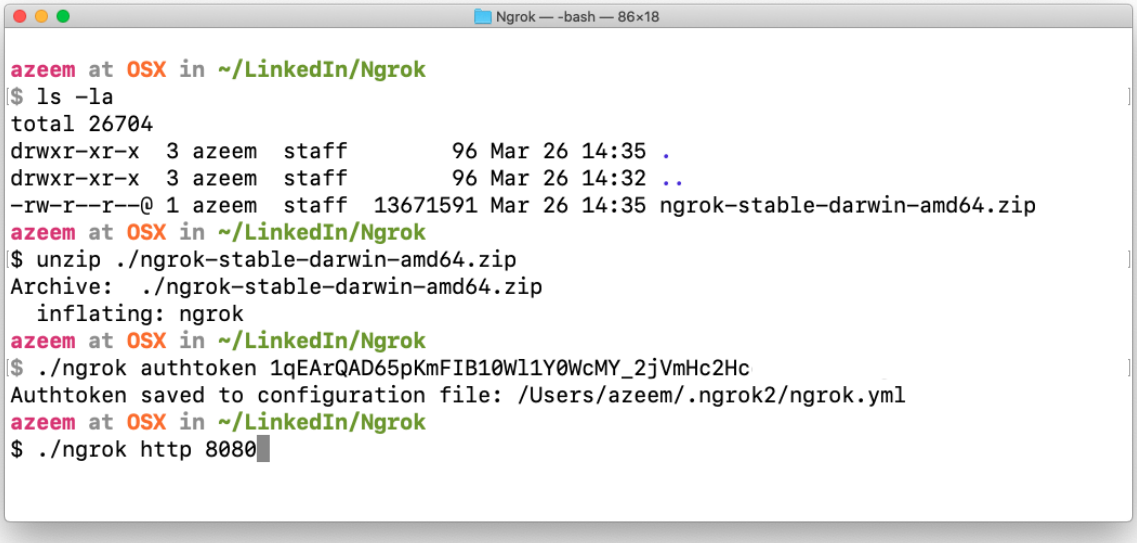


After register, we need to download the executables

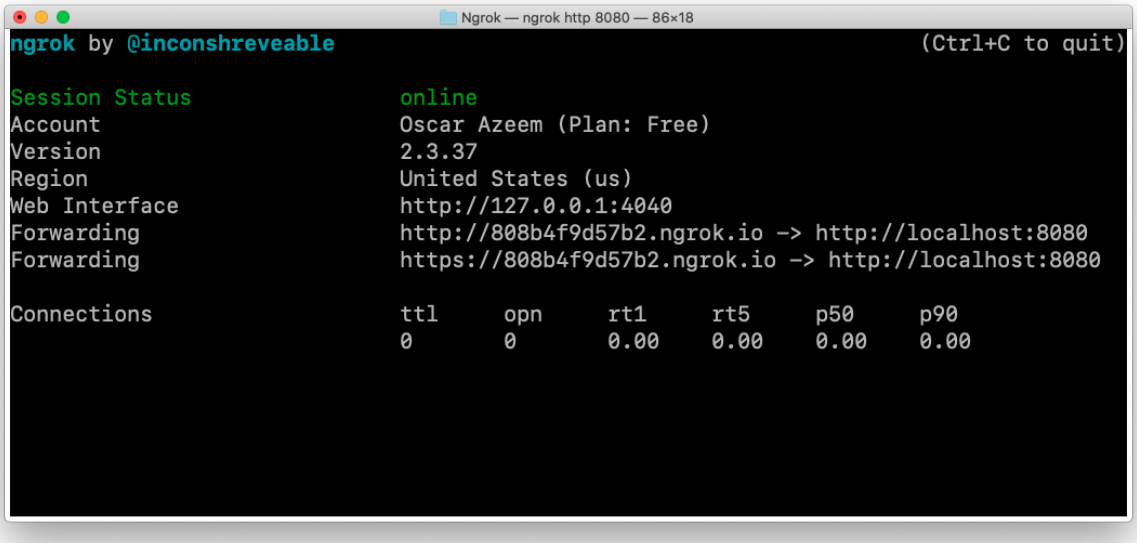


Using the terminal, inside the directory we store ngrok, we continue to unzip it, add our token from the ngrok webpage, and start the server with the 8080 port corresponding to Jenkins.

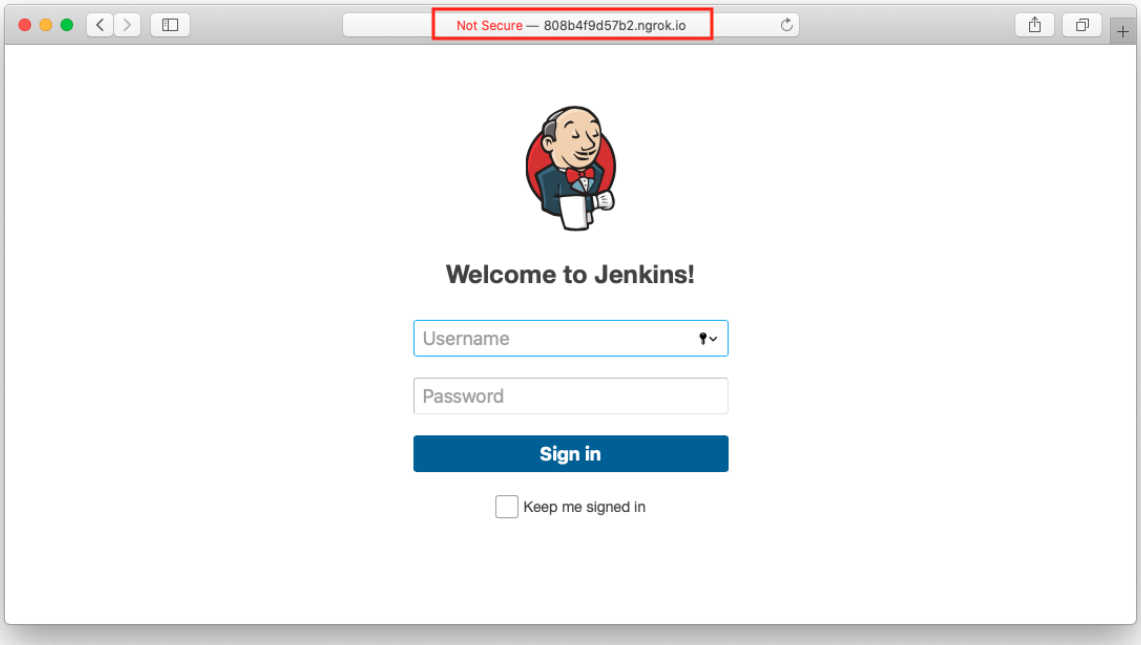
Using the terminal, inside the directory we store ngrok, we continue to unzip it, add our token from the ngrok webpage, and start the server with the 8080 port corresponding to Jenkins.



Once we press enter, the ngrok server starts and gives us a public IP, which forwards any petitions to our localhost 8080 port.



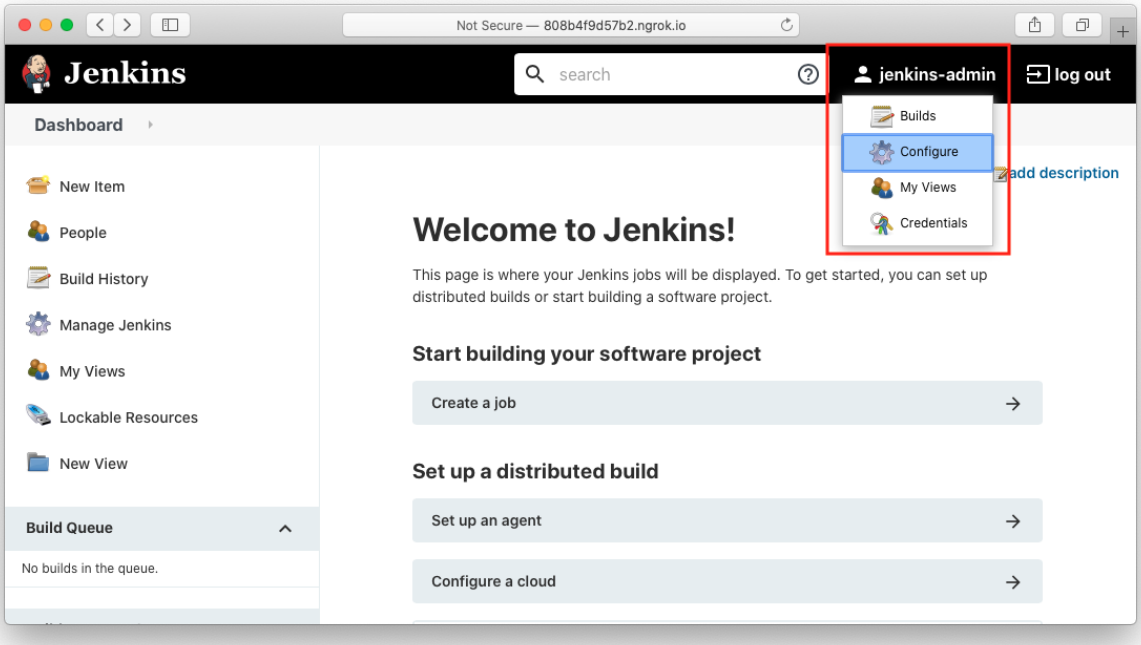
Using this direction, we can access our localhost 8080 port (Jenkins) from any outside network with our web explorer.



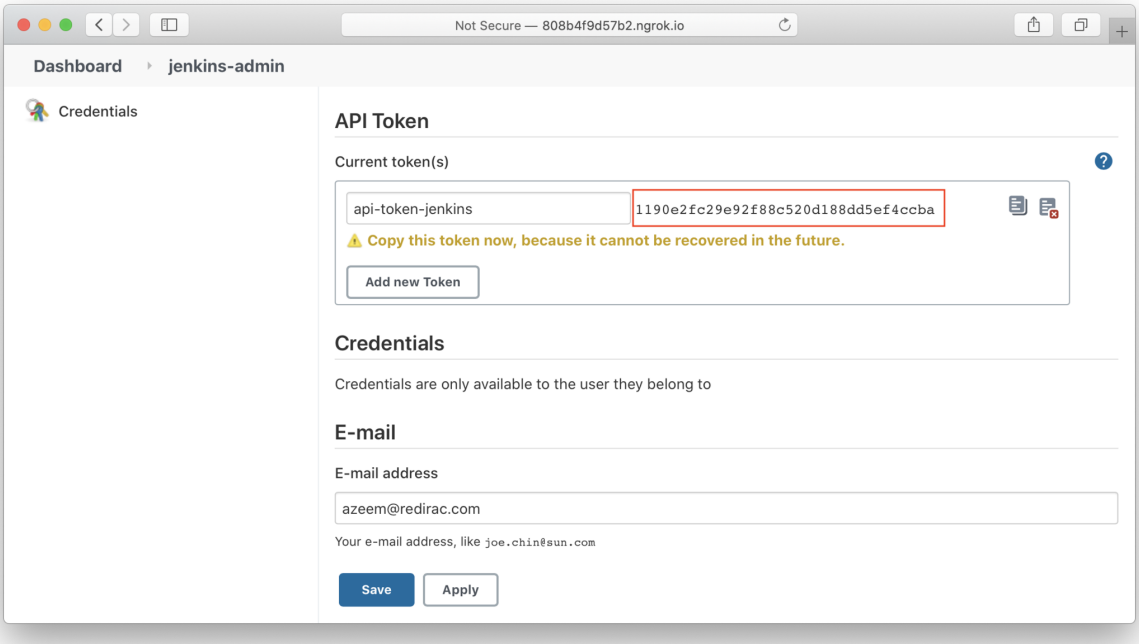
So far so good, using ngrok we have exposed our local Jenkins to the internet. The only step remaining it's to login to Jenkins and continue with the next section. 🖥️

2. Create a Jenkins API Token

To connect our Jenkins server with our Webhook, we will first need to create an API Token to authenticate from the version control provider (Github in this case). To do so, we need to click on our account and then on the configure option.



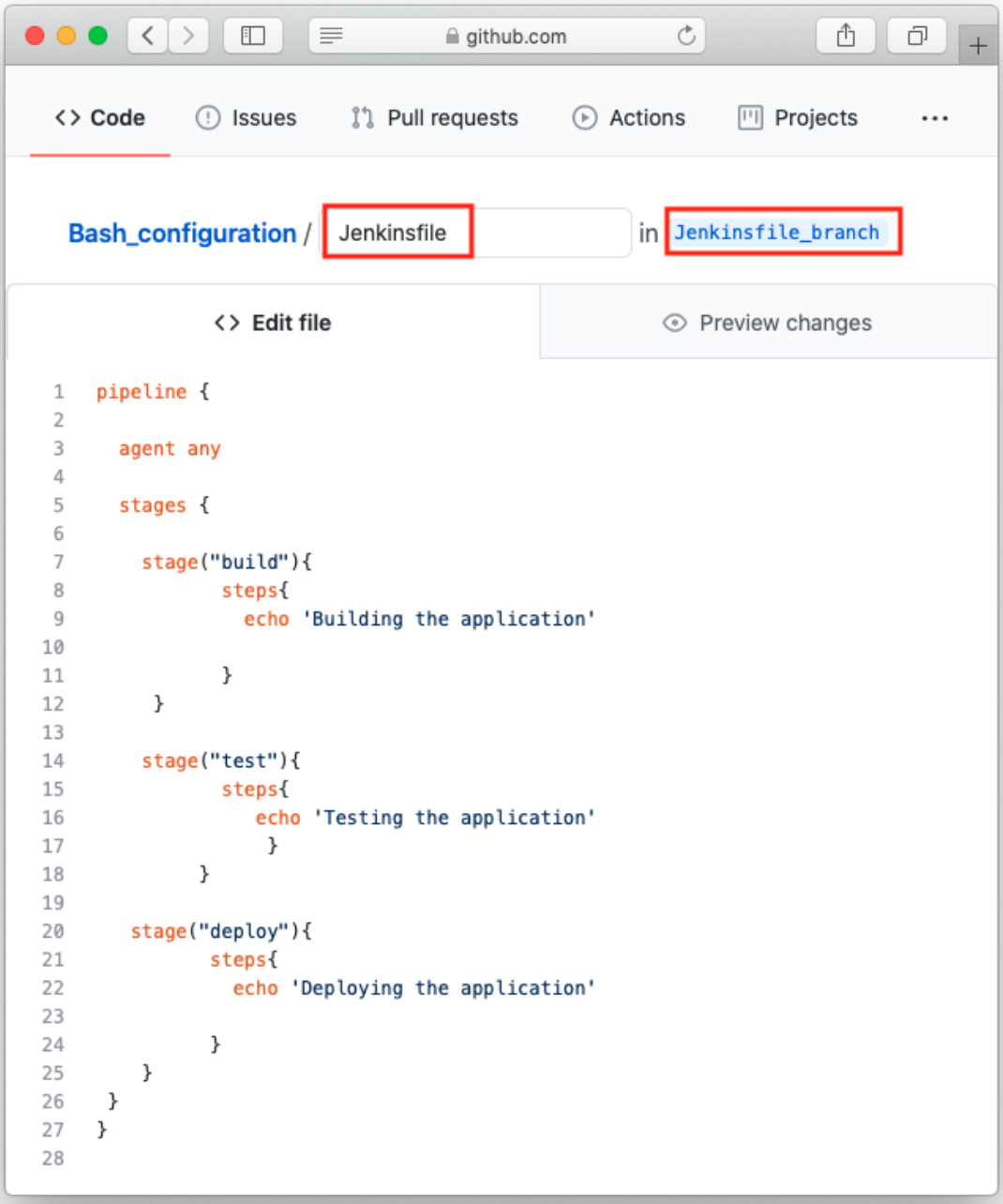
Click on Add new Token, write a name for it and then press Generate. We need to store this Token in a safe place because there is no way to recover it in the future unless creating a new one.



3. Configuring our Github repository

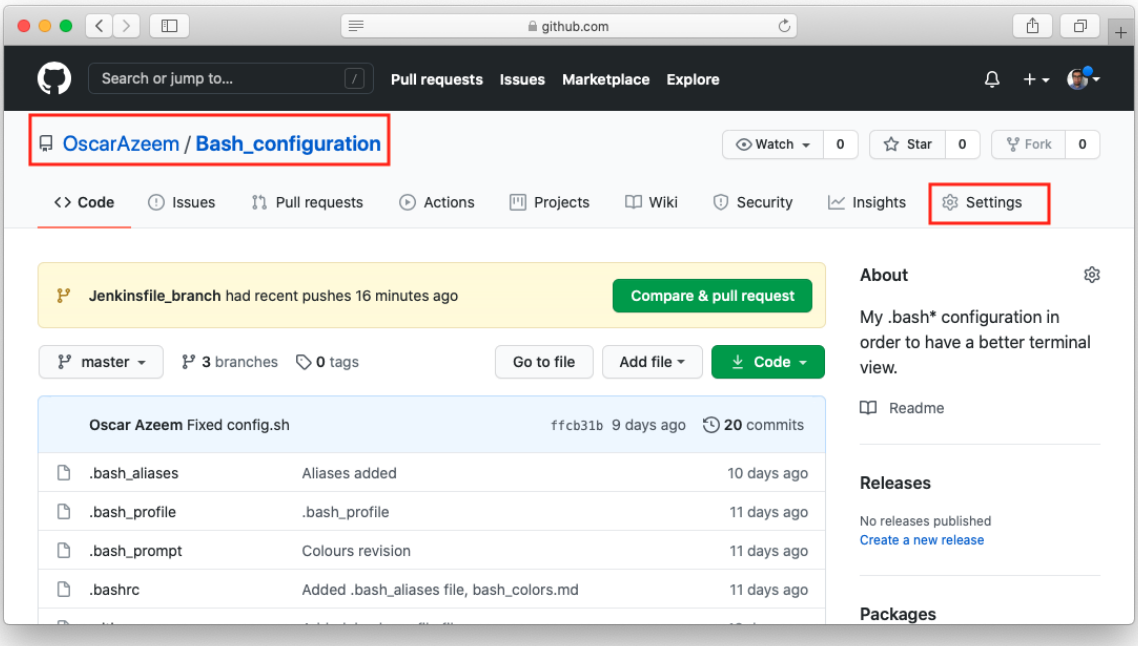
3.1 Adding a Jenkinsfile

Before creating a Jenkins pipeline to be triggered with the webhook, it's good practice to include a Jenkins file inside our git project so that there are already declared all the stages and steps to be validated on each run. The following is an example with three stages: build, test, and deploy if you don't already have a Jenkinsfile in your project. Each one of the three stages has a single step echoing the current step.

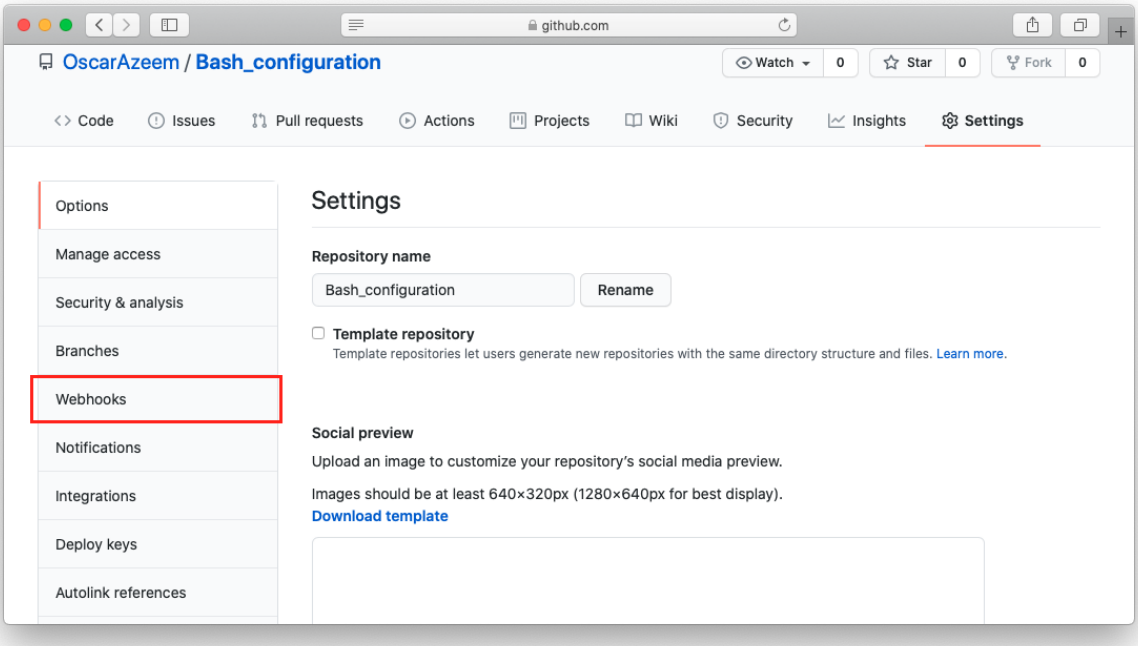


3.2 Creating the Github webhook

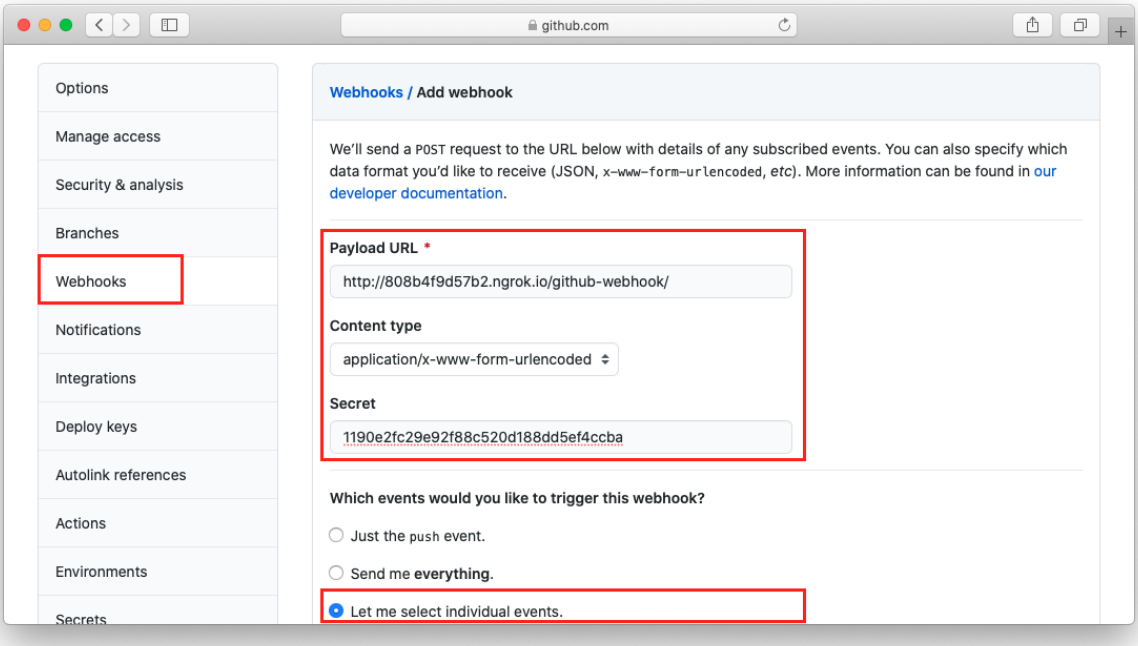
To create our GitHub Webhook, we need to open our repository and click on settings.



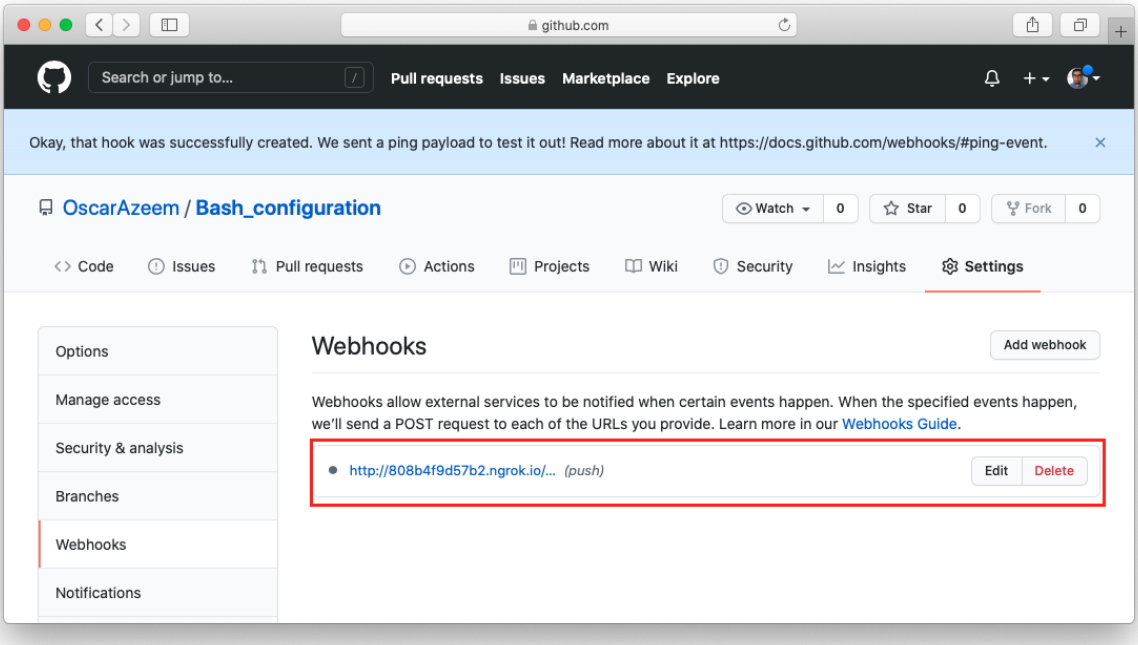
Then click on Webhooks and Add webhook



Inside the Payload URL text box, we need to write the direction to our Jenkins Server, plus: /github-webhook/. If you created a ngrok account, it'd be obtained at the end of section one. The payload must have both / to work, otherwise it'll throw some error. Inside the Secret box, we'll paste the Jenkins API Token from section two. Finally, click on: Let me select individual elements to choose the desired webhook events for our needs to trigger Jenkins.



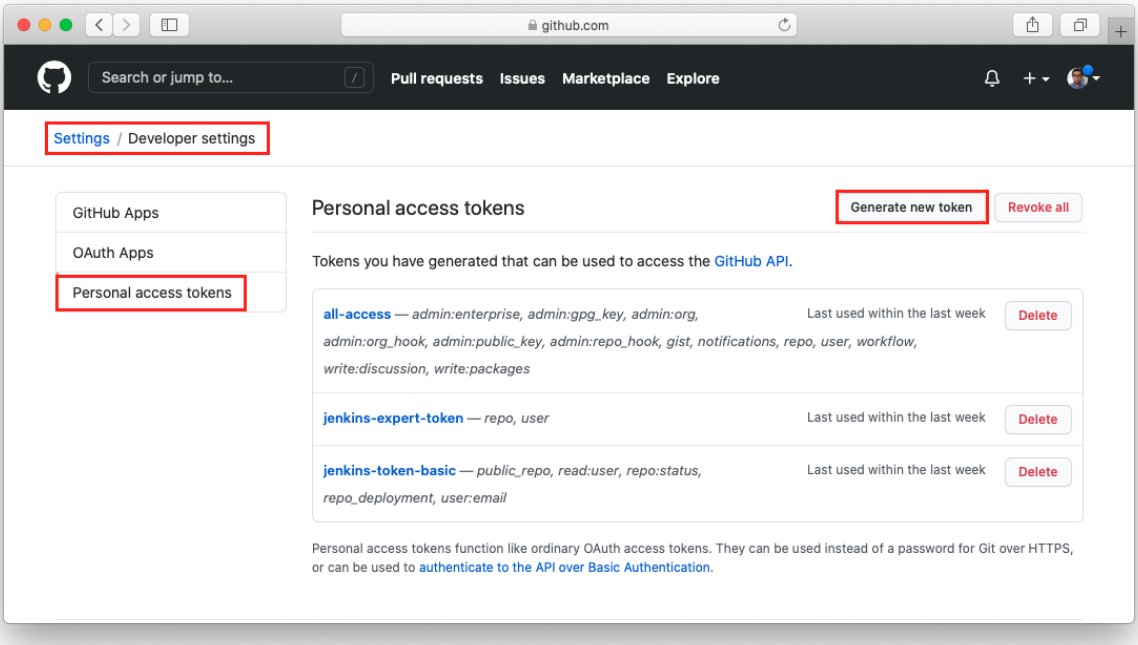
Once created the webhook, it'll look like the following image.



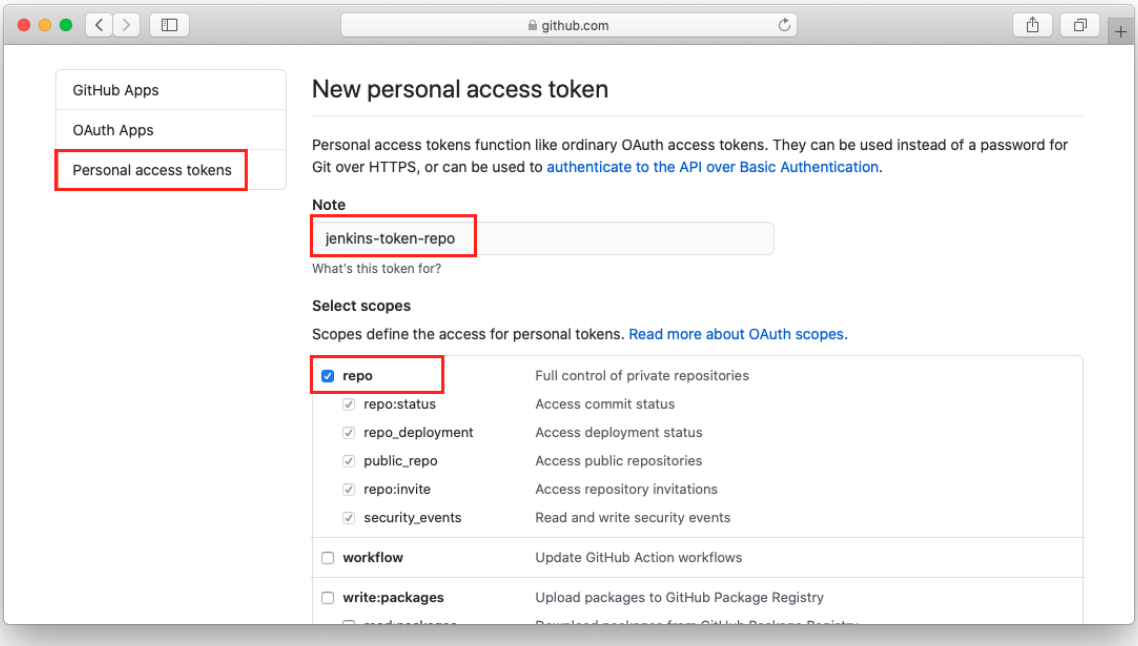
3.2 Creating a GitHub Token

Since early 2021 GitHub restricts API user authentication to work only with a token or private ssh key. There is no longer possible to authenticate using our user and password, and such is the case working with Jenkins. If you are not using GitHub, continue to section 4. Otherwise, keep reading.

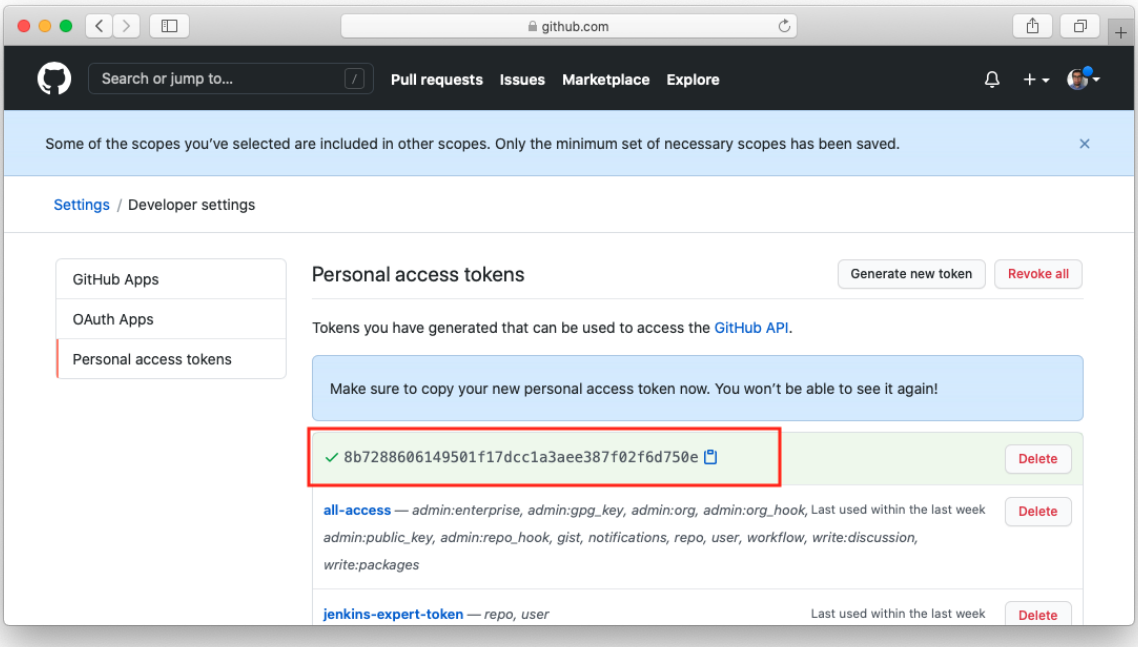
To create a token to be authenticated from Jenkins, we need to go to our GitHub profile, then click on Settings, Developer Settings, Personal access tokens (or go to this direction: https://github.com/settings/tokens). Once there, click on Generate new token



We give it a name plus checking the two boxes: repo and user, finally click on Generate Token.



Once the token has been generated, it'll look like the following. Again, these tokens are meant to be kept in a safe place, there is no way to recover them once lost unless creating a new one.

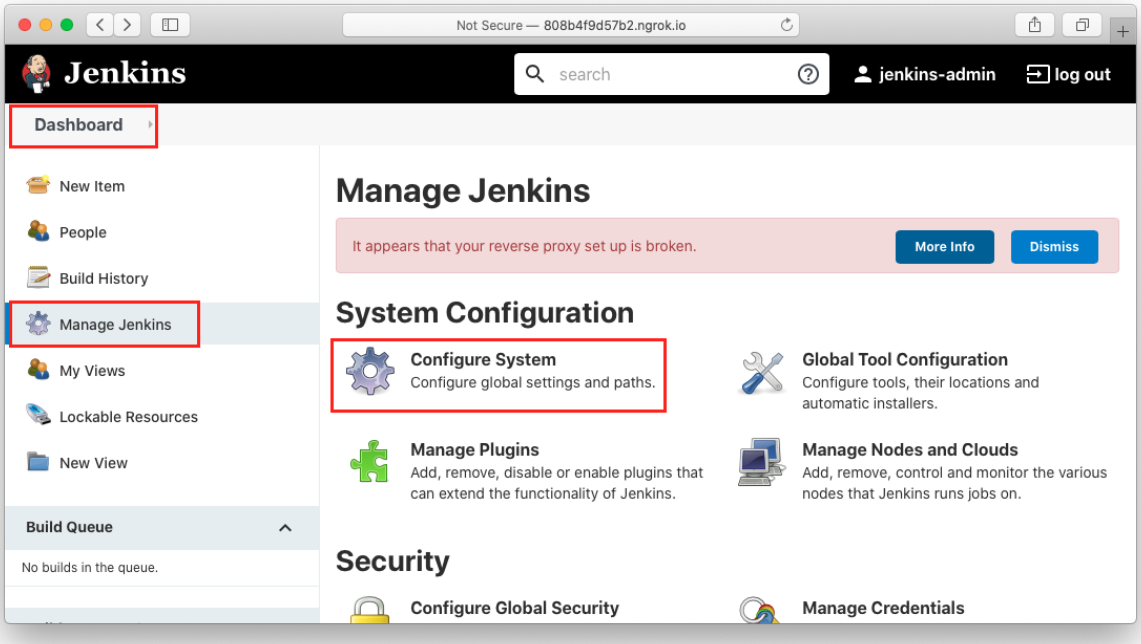


4. Configuring Jenkins

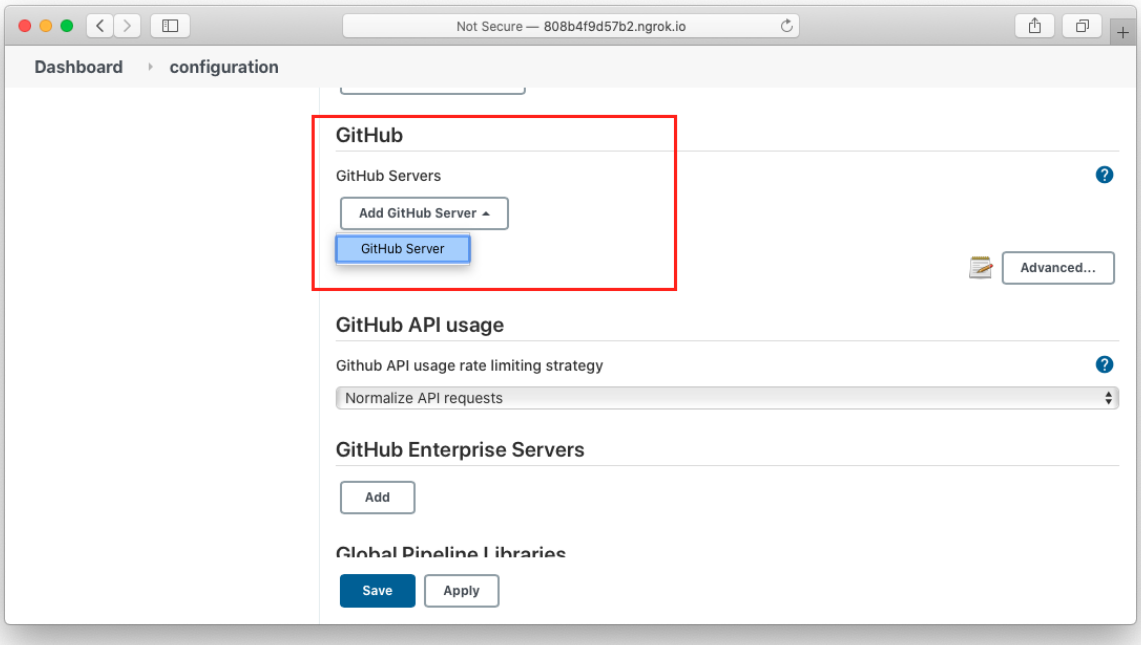
Let's recapitulate a little. We have our Jenkins server exposed to the internet using ngrok. Then created a Jenkins API Token so we can be authenticated with the GitHub webhook. Later we create a Jenkinsfile for our project (if there hasn't been any yet) and configure our GitHub webhook and token (for the Github Jenkins API). Now we need to add these credentials to Jenkins and create a pipeline to run the Jenkinsfile tests for our remote repository.

4.1. Adding GitHub Credentials

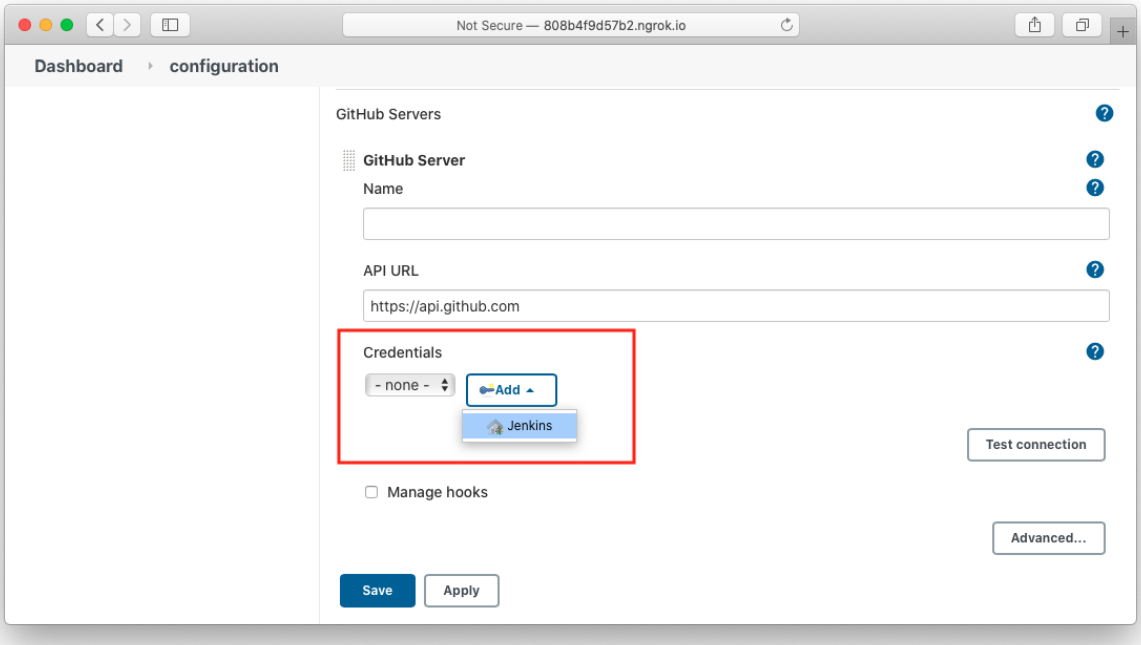
To add our Github credentials on Jenkins, we need to go to the dashboard, Manage Jenkins, and Configure System. If a message appears saying that our reverse proxy set up is broken, just click on dismiss. This message is due to the proxy connection using ngrok.



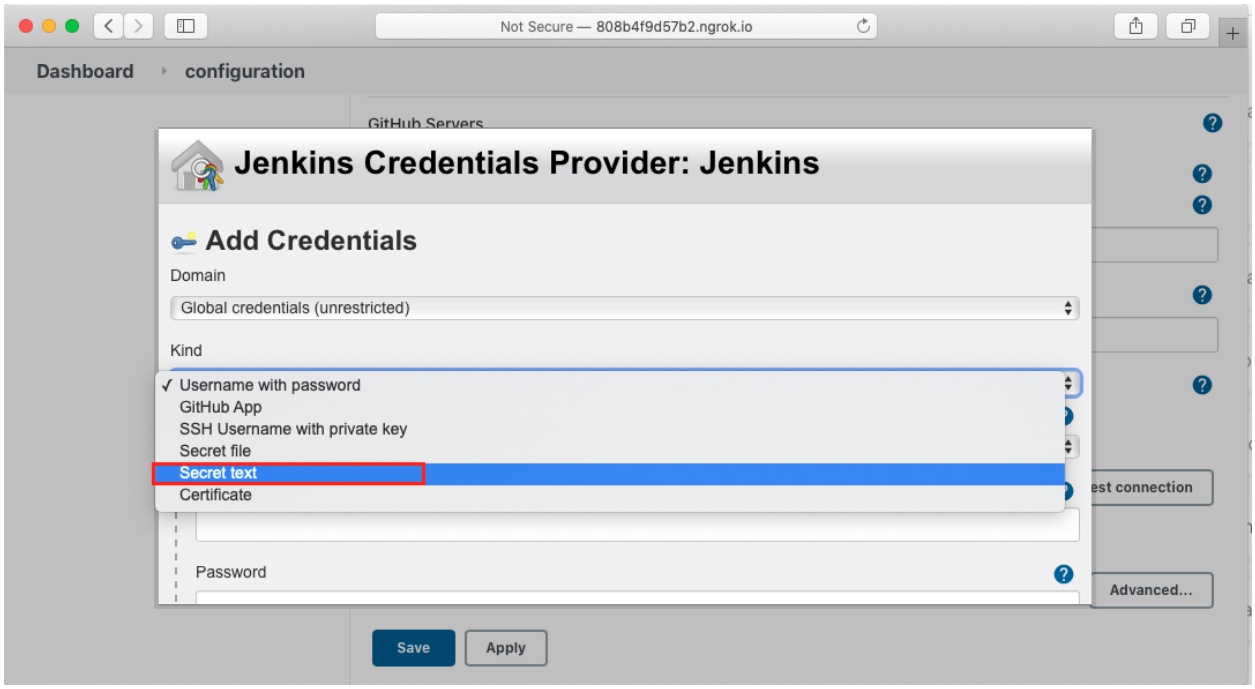
Scroll down to the GitHub section (you need first to have the GitHub plugin installed on Jenkins; if you installed all the recommended plugins at Jenkins installation, it should be there) click on Add GitHub Server, GitHub Server



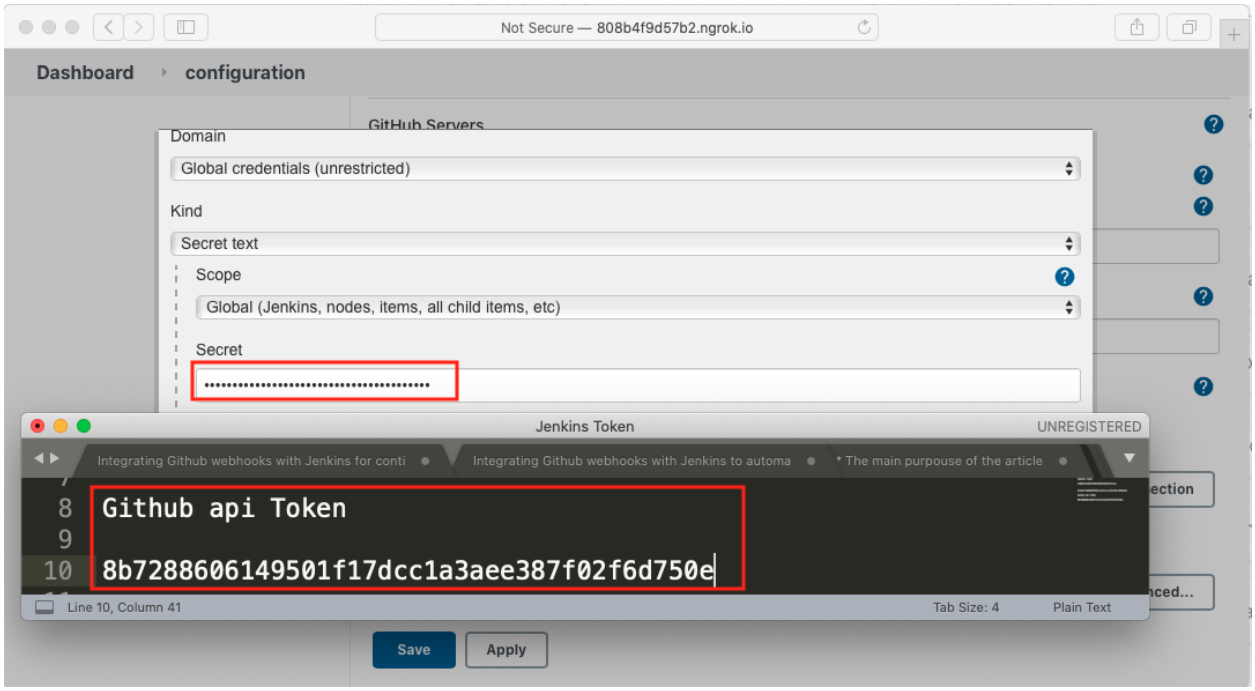
Inside the credentials section click on add, Jenkins.



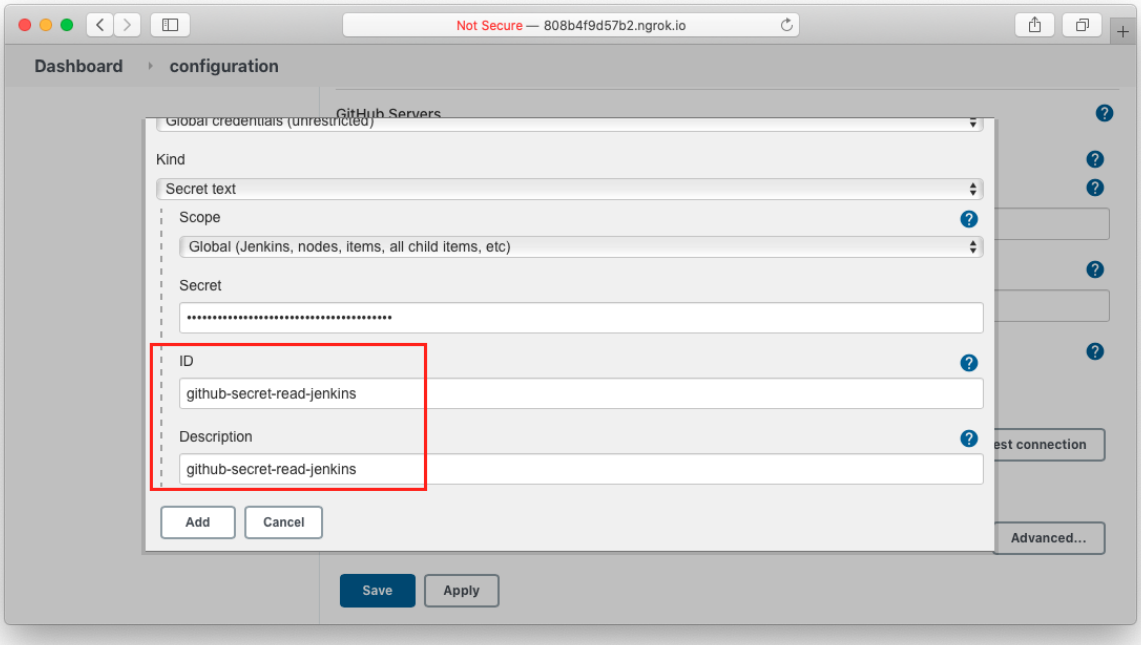
It'll appear a new window. On this window, we should select Secret text



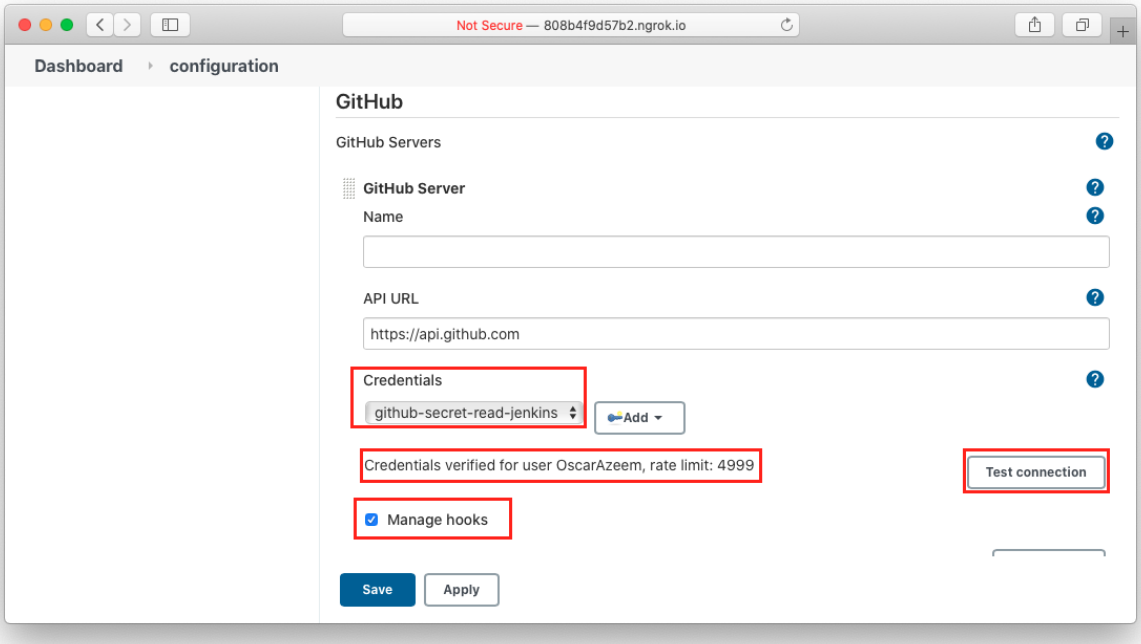
Inside the Secret box, we paste the GitHub API token obtained in section 3.2



Give it some ID and Description and click on ADD

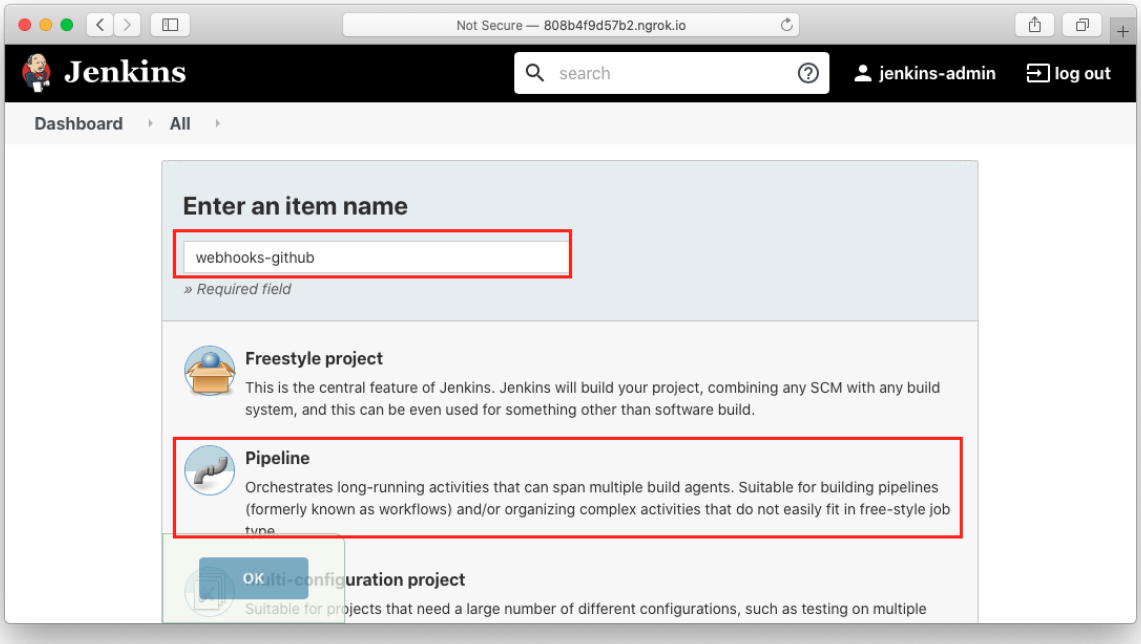


Now we can see this new credential inside the credentials option box, select it, and then click on Test connection. If everything goes right, it should say, Credentials verified for user <My-Github-User>, rate limit: <Number>. If you receive some error, recheck your GitHub API token. Finally, enable the Manage Hooks option and click on save.

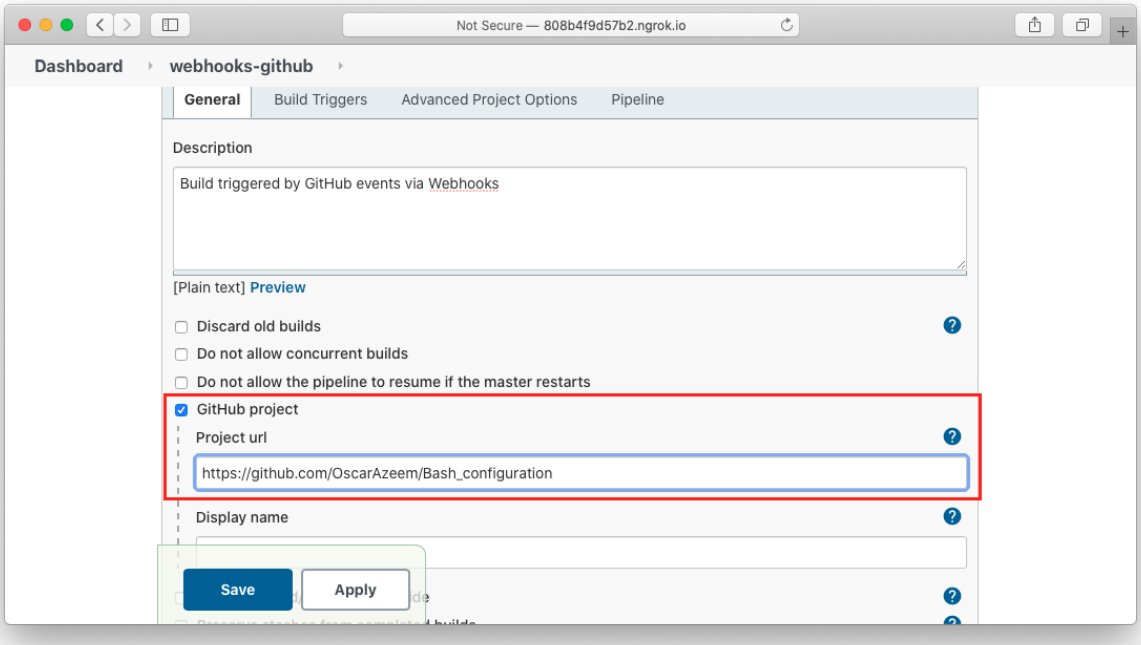


4.2 Creating a Jenkins Pipeline

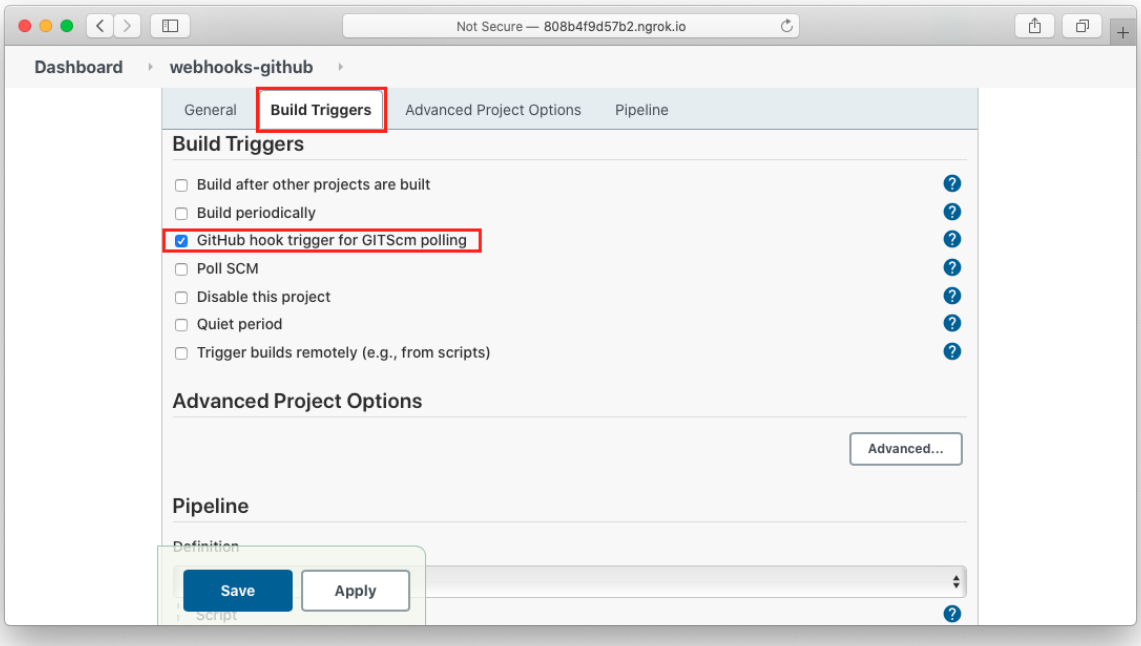
To create a Jenkins pipeline to test our webhook, we go to the dashboard, new item, give it a name and click on the Pipeline job.



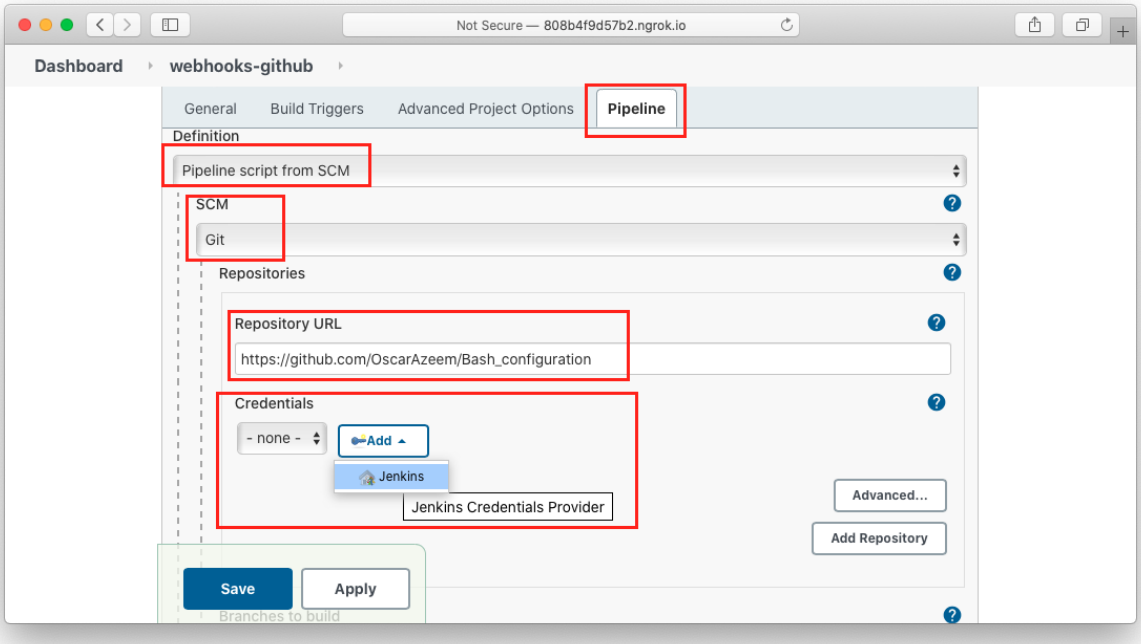
In the General Tab add a description for this pipeline, check the option: GitHub project, and inside the Project Url text box, write the URL for the remote repository to be tested.



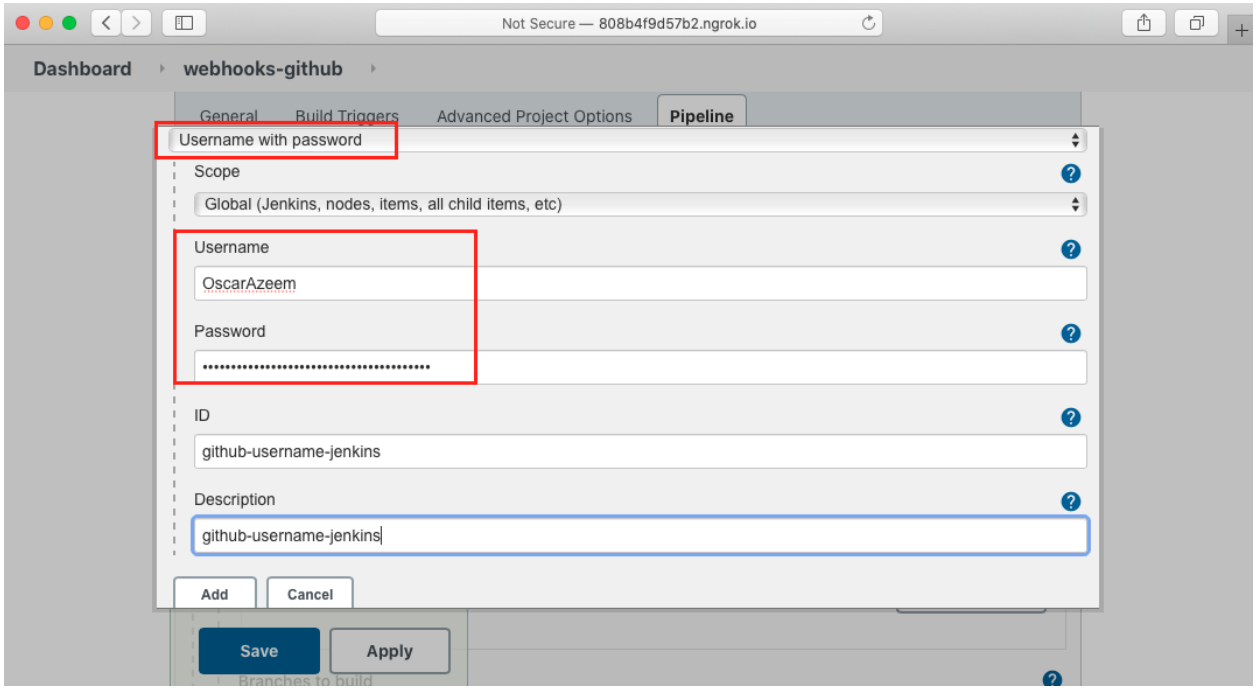
In the Build Triggers tab, check the box: GitHub hook trigger for GITScm polling. This option will enable this build to run after the Webhook sends the POST request.



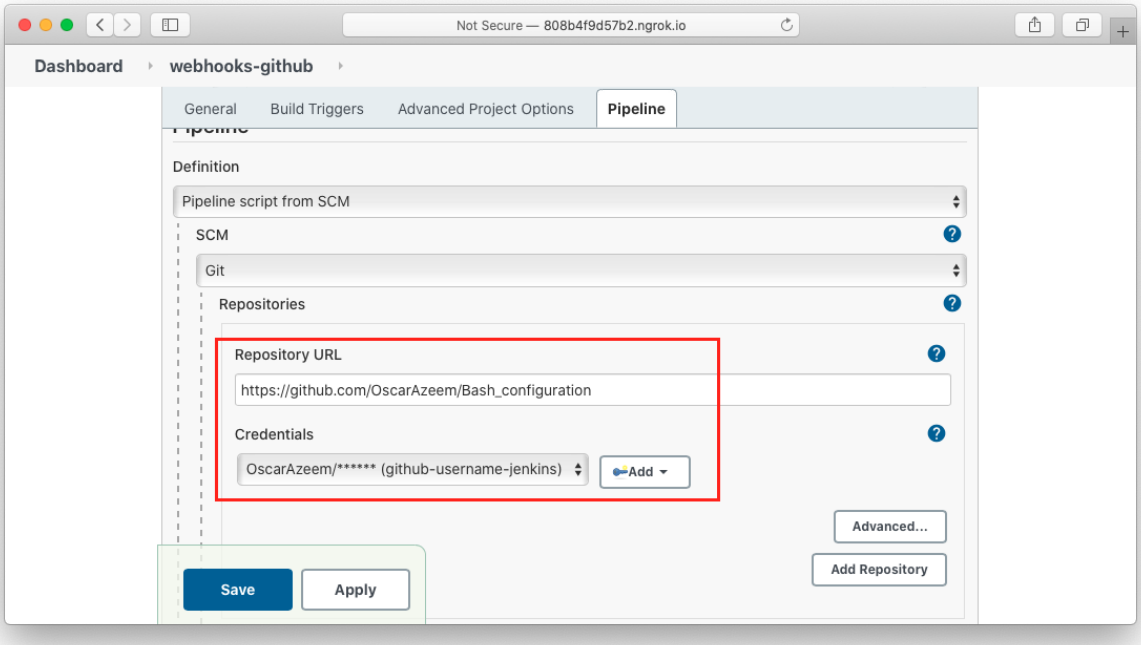
In the pipeline tab, choose: Pipeline script from SCM option to execute the steps declared in the Jenkinsfile for our project, select Git and paste your repository URL. Then we need to add our credentials. This step is not completely necessary but will enable GitHub to set a mark if our test passed or failed.



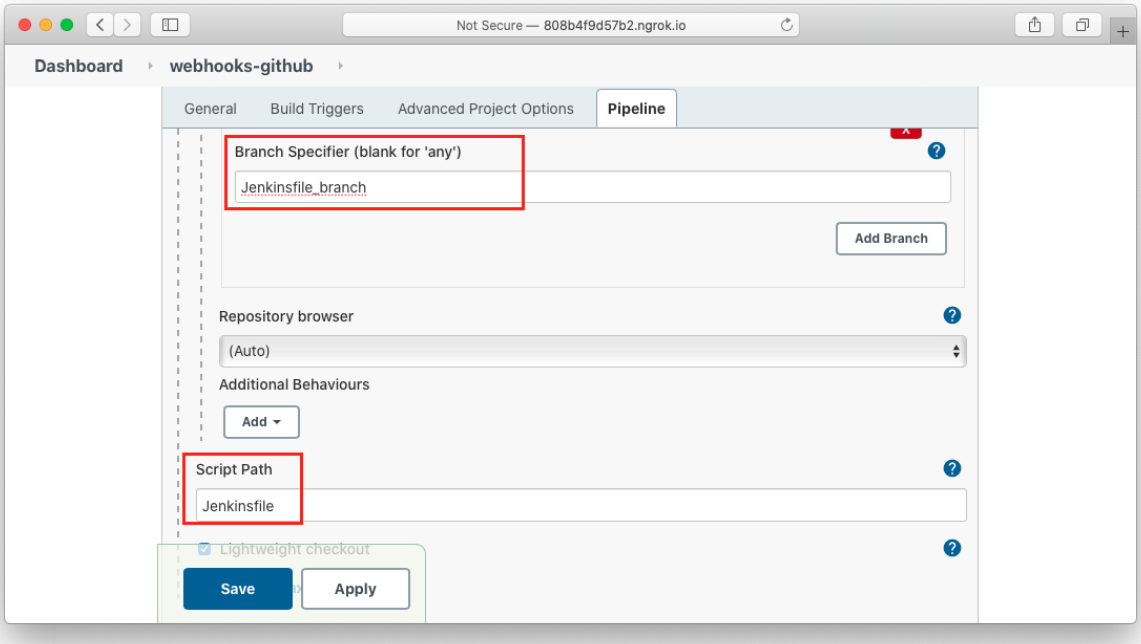
After clicking on add, it'll appear the credentials window. This time choose Username and Password. Write your GitHub username, but in the password text box, paste your GitHub API token, not your current password to access your GitHub account through the webpage.



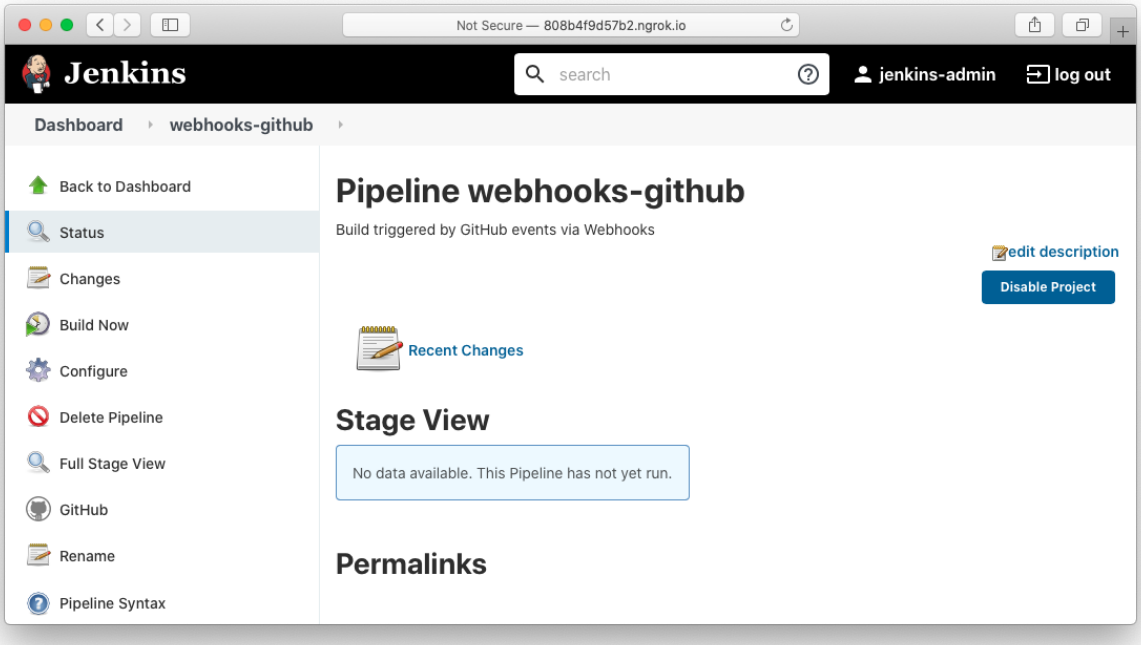
Now your credential should be visible from the options.



Finally, we choose our branch to build the pipeline and the path for the Jenkinsfile. Click on save

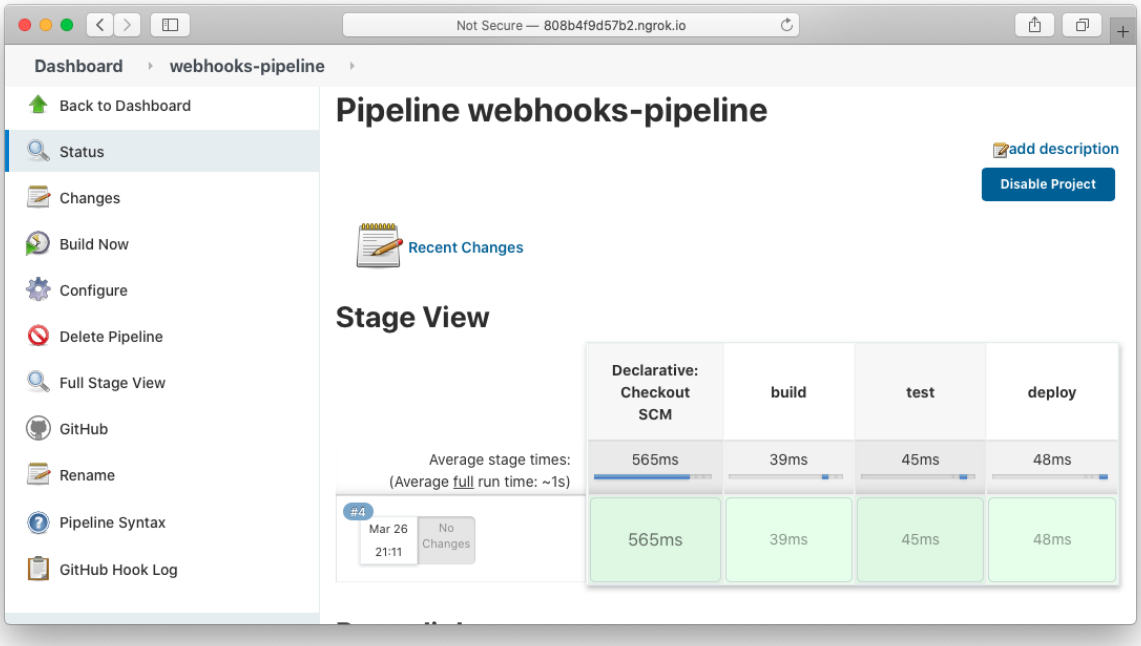


Our pipeline should look like the following image.

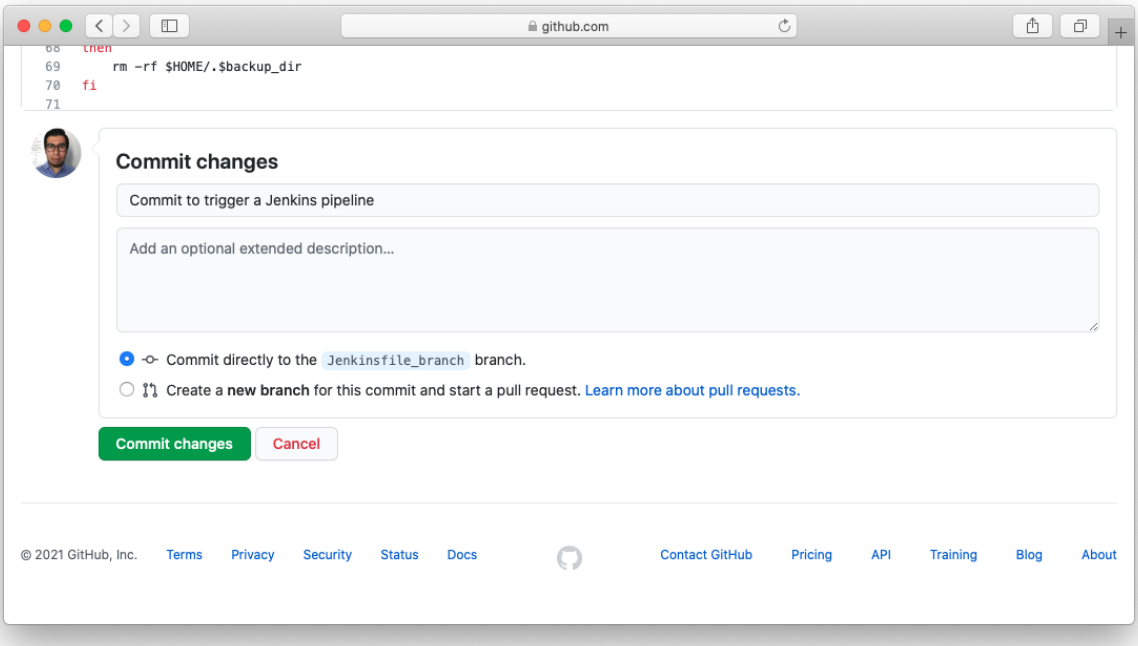


5. Pushing a commit to our GitHub repository and triggering the build

This process's main idea is to automate a Jenkins pipeline to run testing validations after some changes have been made in a remote repository. At this point, all the configurations have been realized successfully, so it's time to test it. Before committing a new change in our GitHub repository, we need to run manually at least one time our Jenkins job to be triggered by the webhook.



After our first run, let's go to our GitHub repository and create a new commit,



This commit will trigger the build automatically