

Getting Started with Jenkins

Description

Jenkins is an open source cross platform automation server. It can be run on different environments and it is the software we will be using to configure and run the idealworks pipeline. Find out more about Jenkins on their official [website](#)

Installation

In our case, we will be running jenkins directly on top of Ubuntu. For the latest documentation steps please re-check with the [official documentation](#)

Jenkins runs on the Java Virtual Machine also commonly known as the JVM. For that purpose the installation of a modern JDK is required, preferably open-jdk 11 and later.

Java Installation (⚠️ Skip this step if you already have Java 8 and above running)

You can search for the different JDK options by running the following:

Update the apt repositories

```
dev1@idealworks:~$ sudo apt update
```

Search for all the available JDKs:

```
dev1@idealworks:~$ sudo apt search openjdk
```

Pick one option of your choice and install it; for example:

```
dev1@idealworks:~$ sudo apt install openjdk-11-jdk
```

To check if Java is now correctly installed you can always run the following:

```
dev1@idealworks:~$ java -version
```

If the installation is correct you should get the following output ('x's are replaced by the installed version number):

```
openjdk version "xx.x.xx" xxx-xx-xx
OpenJDK Runtime Environment (build xx.x.xx+Ubuntu-0ubuntu1.18.04)
OpenJDK 64-Bit Server VM (build xx.x.xx+Ubuntu-0ubuntu1.18.04, mixed mode, sharing)
```

Jenkins Installation

Now that you have Java correctly configured on your machine it is time to install Jenkins. It is recommended to run Jenkins as a service for multiple reasons:

- Easily check the status of Jenkins; for example :

```
jenkins.service - LSB: Start Jenkins at boot time
Loaded: loaded (/etc/init.d/jenkins; generated)
Active: active (exited) since Thu 2021-03-11 12:35:14 CET; 37min ago
```

- The ability to start, stop and restart jenkins with simple commands like

```
dev1@idealworks:~$ systemctl start jenkins
dev1@idealworks:~$ systemctl restart jenkins
dev1@idealworks:~$ systemctl stop jenkins
```

- When installed as a service jenkins is also configured to run at start time.

To install Jenkins on Linux, the following commands should be ran:

```
dev1@idealworks:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo dev1@idealworks:~$ apt-key
dev1@idealworks:~$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins
dev1@idealworks:~$ sudo apt-get update
dev1@idealworks:~$ sudo apt-get install jenkins
```

After the installation is complete run the following command to check Jenkin's Status:

```
dev1@idealworks:~$ systemctl status jenkins
```

And you should get something similar to:

```
jenkins.service - LSB: Start Jenkins at boot time
Loaded: loaded (/etc/init.d/jenkins; generated)
Active: active (exited) since Thu 2021-03-11 13:15:52 CET; 9min ago
  Docs: man:systemd-sysv-generator(8)
Process: 10059 ExecStop=/etc/init.d/jenkins stop (code=exited, status=0/SUCCESS)
Process: 10173 ExecStart=/etc/init.d/jenkins start (code=exited, status=0/SUCCESS)
```

Configuration

Now that jenkins is installed it is time to configure it. The configuration will be done on different levels, some of it administrative and the other related to the communication with github.

Jenkins is a service that is exposed via a web interface; the default url is generally MACHINE_IP:JENKINS_PORT . MACHINE_IP being the address you can access your machine on and JENKINS_PORT being the configured Jenkins Port. The default jenkins port is 8080.

If you are running Jenkins on a local machine you can try finding Jenkins at this [link](#).

First time setup(⚠️ This step is only relevant if you are running jenkins for the first time)

If you are accessing the Jenkins web page for the first time you will be asked to enter an OTP (One time password) to be able to access Jenkins and then create an administrator user of your own.

You should get something like this:

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password



Continue

This indicates that there is a one time password file found in the above-mentioned directory. Make sure you have the correct access right to be able to open the file.

Once you have unlocked Jenkins the next step is to install some plugins that are required.

Getting Started

Customize Jenkins

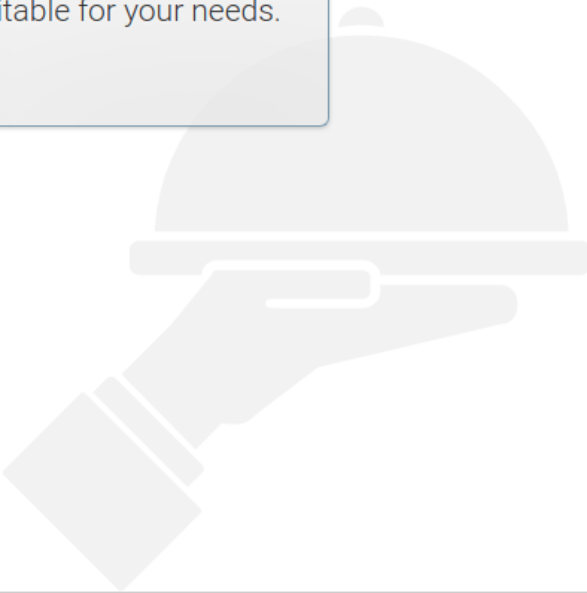
Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.



Jenkins 2.277.1

The recommended choice is to **Install Selected Plugins**. If you are experienced with Jenkins and now exactly which plugins to install, I don't think you should be reading this anyway 😏.

Jenkins Plugins

As a small side-note Jenkins plugins are the secret behind Jenkins power and popularity. Plugins are usually written by 3rd party organizations or individuals which extend the power of Jenkins Immensily. They allow for additional functionality that doesn't come with Jenkins by default.

Creating the First Admin User

In this step you are required the Jenkins' admin credentials. Make sure to save them somewhere external and safe because resetting the administrator user can be a daunting task.

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Jenkins 2.277.1

Skip and continue as admin

Save and Continue

Setting up the Jenkins URL

In this step you can modify the Jenkins URL if you want to do so. For users that are not familiar with Jenkins, it is recommended to leave this value as is.

Getting Started

Instance Configuration

Jenkins URL:

http://localhost:8080/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.277.1

Not now

Save and Finish

Pipelines

Now that the Jenkins Installation is out of the way we can start by creating Jenkins Jobs. In our case the Job will be a Jenkins Pipeline.

What we aim to achieve in this project is to have a Jenkins pipeline that will run everytime a pull request is created. This pipeline will serve as a check mechanism to indicate whether the pull request is **safe to merge**.

How do we know if the pipeline passed the check?

We will configure the pipeline to be able to communicate with Github. When a pull request is submitted; a prompt will appear on the Github page showing the pipeline state:

- All checks are completed; code is safe to merge. (Pipeline outcome SUCCESS)
- Pending (Pipeline still running)
- Some/All checks have failed. (Pipeline failed)

What is a Jenkins pipeline ?

If we are to vulgarize the definition a bit, a Jenkins pipeline is, simply put, a sequence of running programs or scripts that are defined in a specific environment, in a specific order.

A Jenkins pipeline could be as simple as calling the echo command 2 times in a row.

Creating a Jenkins pipeline

- In order to create a Jenkins pipeline, one must login with their Jenkins user to the Jenkins web interface.
- Once logged in, you will be redirected to the Jenkins Dashboard.
- On the top left you will find a Button named **New Item**. Click On it.
- Enter the job name you wish to create. After that select Pipeline. Press **OK** to finish creating the pipeline.
- Congrats, you have created your first pipeline 🐉🐉

Configuring your first pipeline

Since we want the pipeline to be able to communicate with Github and communicate whether it has failed or not. We will be needing a plugin called "GitHub Pull Request Builder"

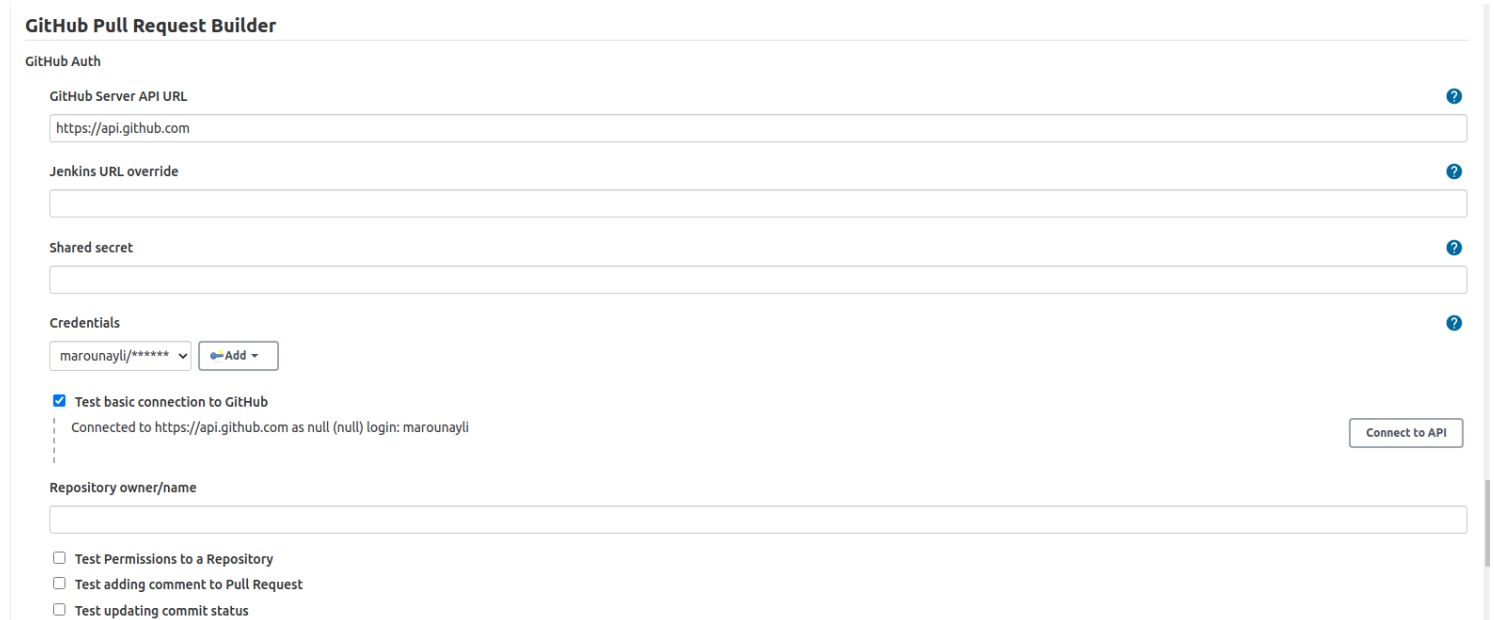
To Download the plugin:

- On the top left corner go to dashboard.
- On the left menu-bar select **Manage Jenkins**
- Click on **Manage Plugins**
- Go to the **Available** section and search for *Github Pull Request Builder*
- Tick the checkbox and on the bottom of the page click **Install without restart**

Now that the plugin has been installed; we need to configure it.


- Once again go to **Manage Jenkins** and click on **Configure System**
- Scroll down to the **GitHub Pull Request Builder**
- If you are not using GitHub enterprise to host your repo Leave the GitHub Server API URL as is
- Leave the Jenkins URL override field empty, especially if you are not using a Load Balancer for Jenkins.
- Should you want extra security; you can configure the Shared secret. You'll need to configure it on GitHub too.
- For the credentials; it is recommended that you create a Jenkins user for the GitHub Repository and that user will be the one to be authenticated. When you want to create the credentials, use the email associated with the account as a username. Do not use the username itself. For the password field; it needs to be a Github API Token with the correct access rights.

You can always test your connection to the Github API. A helping screenshot is provided:



Now that the plugin installed we can go back to the pipeline configuration

General

- Go to dashboard, you'll find your newly created pipeline there. Click on it and then go to  Configure.
- Enter your pipeline description in the **Description Box**
- Feel free to tinker with the options in the General Options; however what is required to correctly configure the project is to tick the GitHub project checkbox and specify the URL for the github repo.

Build Triggers

- Tick the GitHub Pull Request Builder Option
- Specify the Admin list (the usernames of the repo owners)
- Tick the option **Use github hooks for build triggering**
- Click Advanced
- Whitelist; you can add the users that will trigger the pipeline with their pull request. In this case we are aiming for everyone with access rights on the repository. We can supply the name of the organization in this case.
- There are many options to explore for further scrutiny

Pipeline

- Definition : Select Pipeline script from SCM
- SCM : Git
- Repository URL : Your Github repo URL
- Add the credentials for the Jenkins Account
- Script Path : Relative Path of the Jenkins File to the repository root

Connecting GitHub To a Local Jenkins Server

If you are new to Jenkins, chances are that you are still running Jenkins locally and you want to do some testing.

If you have some networking knowledge; you'd know that Github cannot directly communicate with your **localhost** because of the multiple NATs on the way.

We need to expose the Jenkins Server to the Internet   

Ngrok

To achieve the above-mentioned we will be needing the help of a software called ngrok.

Ngrok is able to expose one of the ports of our local machine to the internet. To achieve this it uses a concept called Port Forwarding. To learn more about port forwarding please refer to the following [link](#)

Installing and running ngrok is really simple. All we have to do is to create a free account on the ngrok website to get an access token. When we get the access token all we have to do is:

Unzip the downloaded package:

```
dev1@idealworks~$ unzip /path/to/ngrok.zip
dev1@idealworks~$ ./ngrok authtoken <auth_token>
```

And then to finally create an HTTP tunnel between localhost and the internet on port 8080 we run:

```
dev1@idealworks~$ ./ngrok http 8080
```

If the tunnel creation is successful, the output should be similar to:

```
ngrok by @inconshreveable
Session Status      online
Account             dwijdiqow (Plan: Free)
Version             2.3.35
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://04380aface7e.ngrok.io -> http://localhost:8080
Forwarding           https://04380aface7e.ngrok.io -> http://localhost:8080





Connections         ttl    opn    rt1    rt5    p50    p90
                   38     0      0.00   0.01   5.32   14.80
```

Now our jenkins is exposed to the internet via the following url : <http://04380aface7e.ngrok.io> . This value obviously depends on the output. Don't try to copy this one...

GitHub

The last step is to link Github's API to our Jenkins Instance. We will be using github's webhooks in order to inform jenkins of the different events to trigger pipeline build.

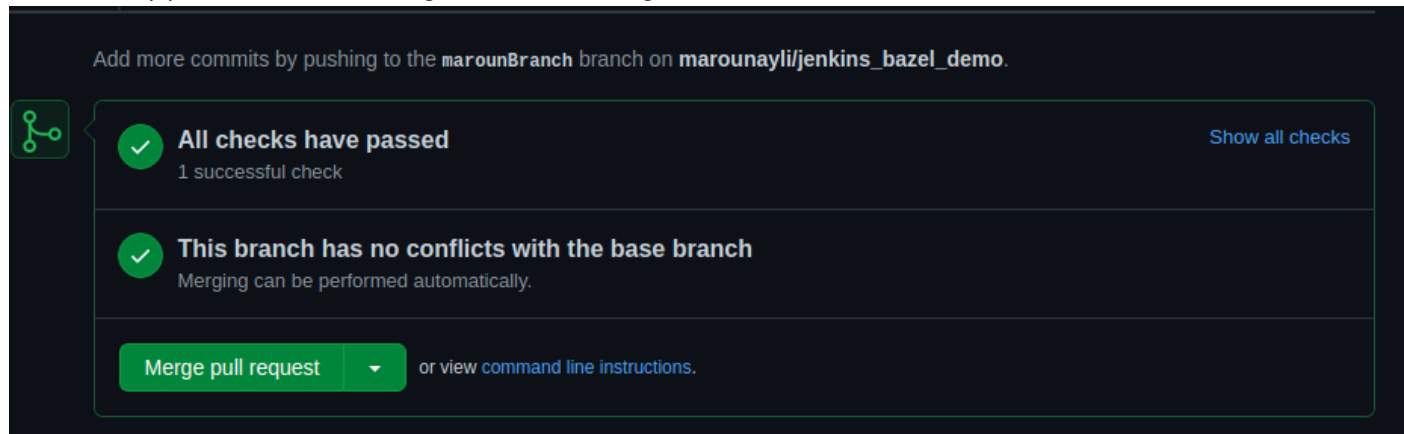
Creating a Github Webhook

- Navigate to your **GitHub Repository**
- Go to  Settings  **Webhooks**
- Click on **Add webhook**
- In the **Payload URL** enter the following : `<generated_ngrok_url>/ghprbhook/` ( Please don't forget the trailing / . If you remove it will cause the hook to fail. )
- Select application/json for the **Content Type**
- **Secret:** You can add the shared secret that you have specified when configuring the **GitHub Pull Request Builder**
- **Which events would you like to trigger this webhook?** : Select Individual events and select **Pull Requests** and remove **Pushes** which comes by default
- Make sure the active check-box is ticked.

Final Steps:

Now that everything is correctly set up, we can test the whole workflow.

On your repository, create a merge request from a branch to another. The pipeline should run on the newest branch. The success or failure of the pipeline will be shown on github in the following form:



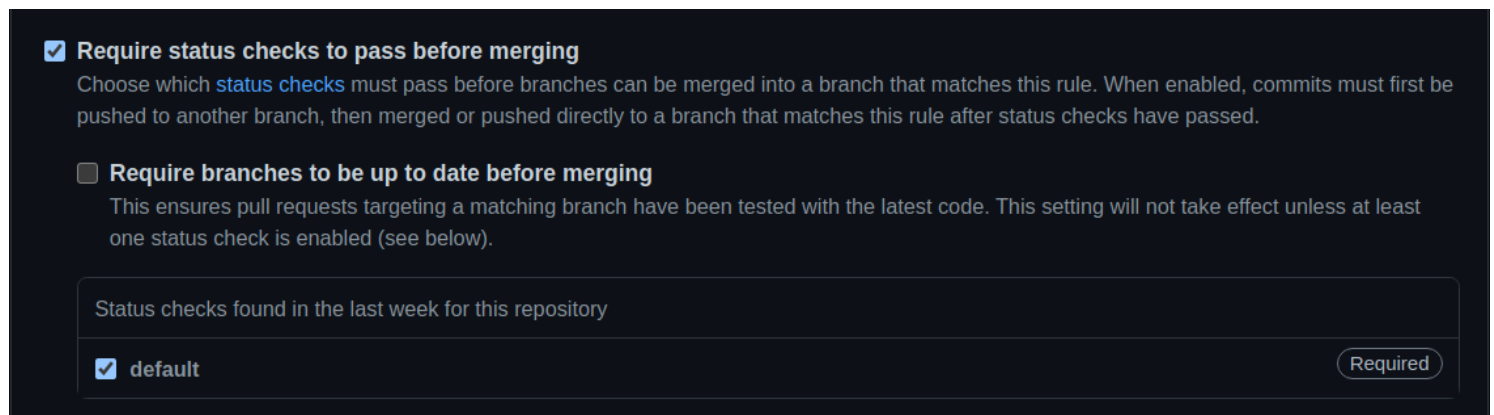
Managing a branch protection rule

Branch protection rules are created to enforce certain workflows for one or more branches, such as passing status checks for all pull requests merged into the protected branch.

A name pattern specified with fnmatch syntax creates a branch protection rule in a repository for a specific branch, all branches, or any branch that matches it. For example, to protect any branches containing the word feature, you can create a branch rule for `*feature*`.


Creating a branch protection rule


1. On GitHub, navigate to the main page of the repository.
2. Under your repository name, click Settings.
3. In the left menu, click Branches.
4. Next to "Branch protection rules", click Add rule.
5. Under "Branch name pattern", type the branch name or pattern you want to protect.




6. Enable required status checks by selecting Require status checks to pass before merging
7. From the list of available status checks, select the checks you want to require.


Now, before every pull request, checks are required to pass



**Some checks haven't completed yet**[Hide all checks](#)


1 pending check

**default** Pending — Build triggered for merge commit.[Required](#)


**Required statuses must pass before merging**


All required [statuses](#) and check runs on this pull request must run successfully to enable automatic merging.

As an administrator, you may still merge this pull request.



[Merge pull request](#)  or view [command line instructions](#).


If checks are verified, the merge can be done.




**All checks have passed**[Hide all checks](#)

1 successful check


**default** — Build finished.[Required](#) [Details](#)


**This branch has no conflicts with the base branch**

Merging can be performed automatically.



[Merge pull request](#)  or view [command line instructions](#).


Else, the pull request can't be accepted until changes are done and checks pass.



**All checks have failed**[Hide all checks](#)


1 failing check

**default** — Build finished.[Required](#) [Details](#)

**Required statuses must pass before merging**

All required [statuses](#) and check runs on this pull request must run successfully to enable automatic merging.

As an administrator, you may still merge this pull request.

[Merge pull request](#)  or view [command line instructions](#).