# A Comparison of GitLab and Jenkins CI

## Toto Fan Club

Computer Systems Year 3

| **Student Name** | **Student ID** |
|---|---|
| Bartosz Walusiak | R00166560 |
| David Hurley | R00150748 |
| Deirdre Connolly | R00112962 |
| PJ O'Regan | R00139430 |

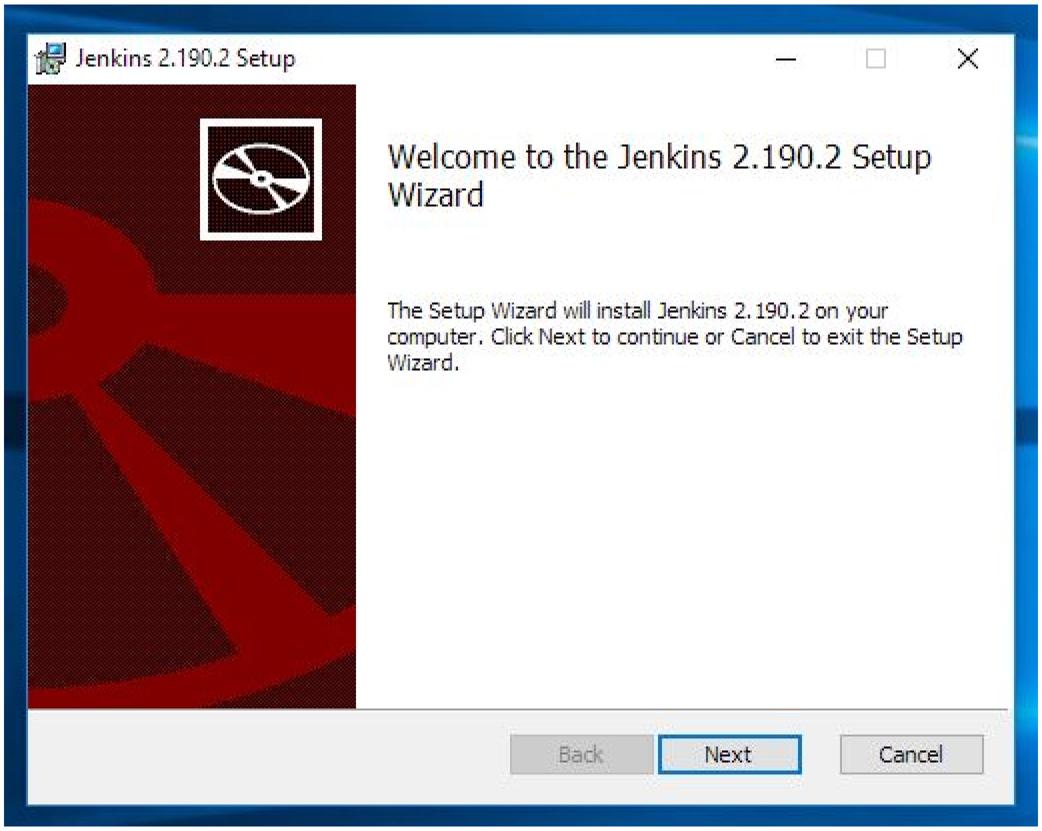
## Setup and Configuration

### Jenkins CI

### Installing Jenkins

NOTE: MY COMPUTER HAD AN IDENTITY CRISIS AND HALF OF THE TEXT IS IN A DIFFERENT LANGUAGE

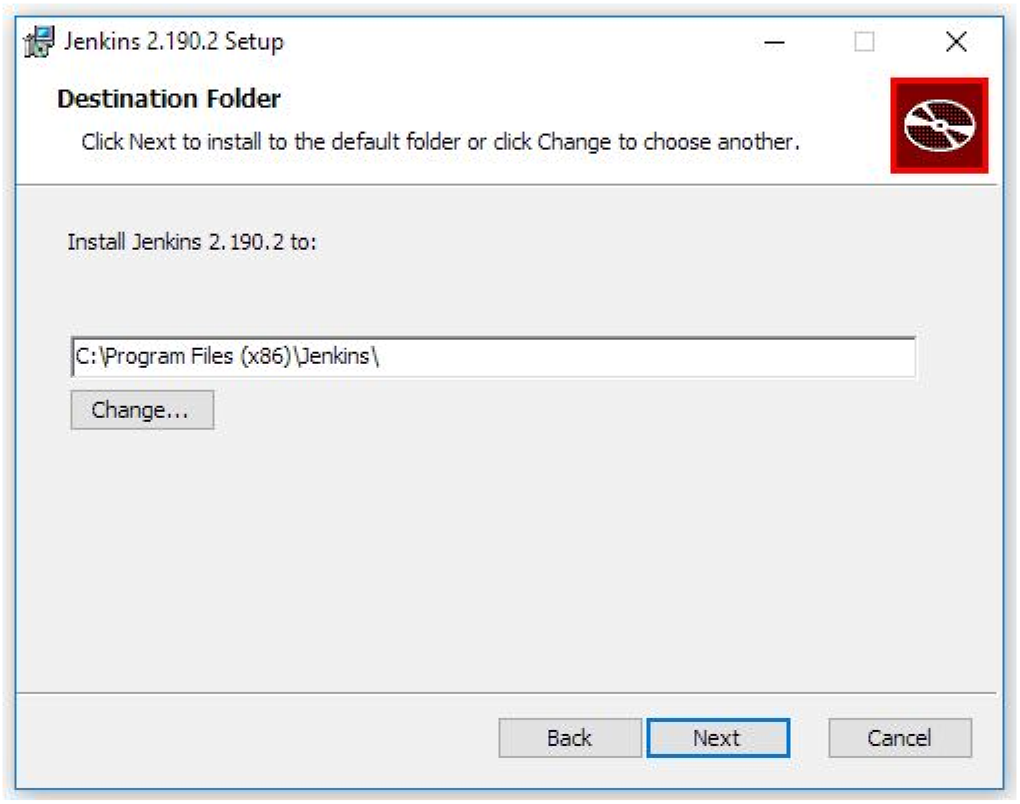
1. Install Jenkins on your Computer



Jenkins 2.190.2 Setup

Welcome to the Jenkins 2.190.2 Setup Wizard

The Setup Wizard will install Jenkins 2.190.2 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

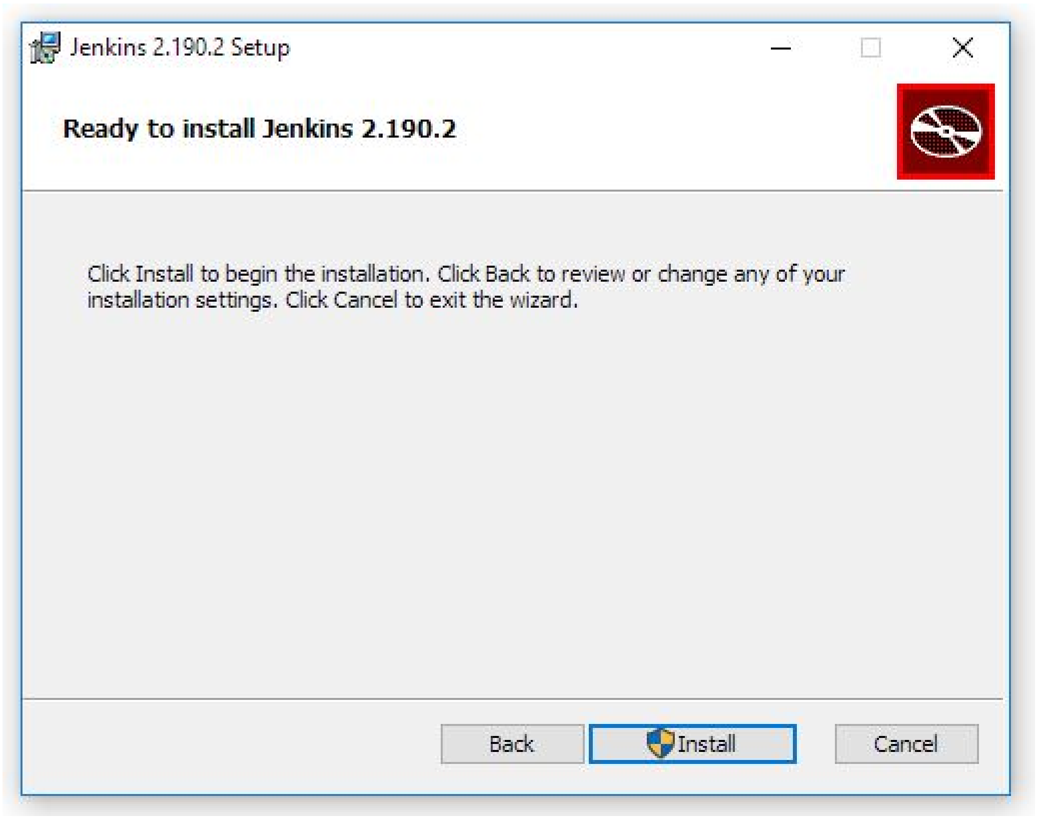

Jenkins 2.190.2 Setup

**Destination Folder**

Click Next to install to the default folder or click Change to choose another.

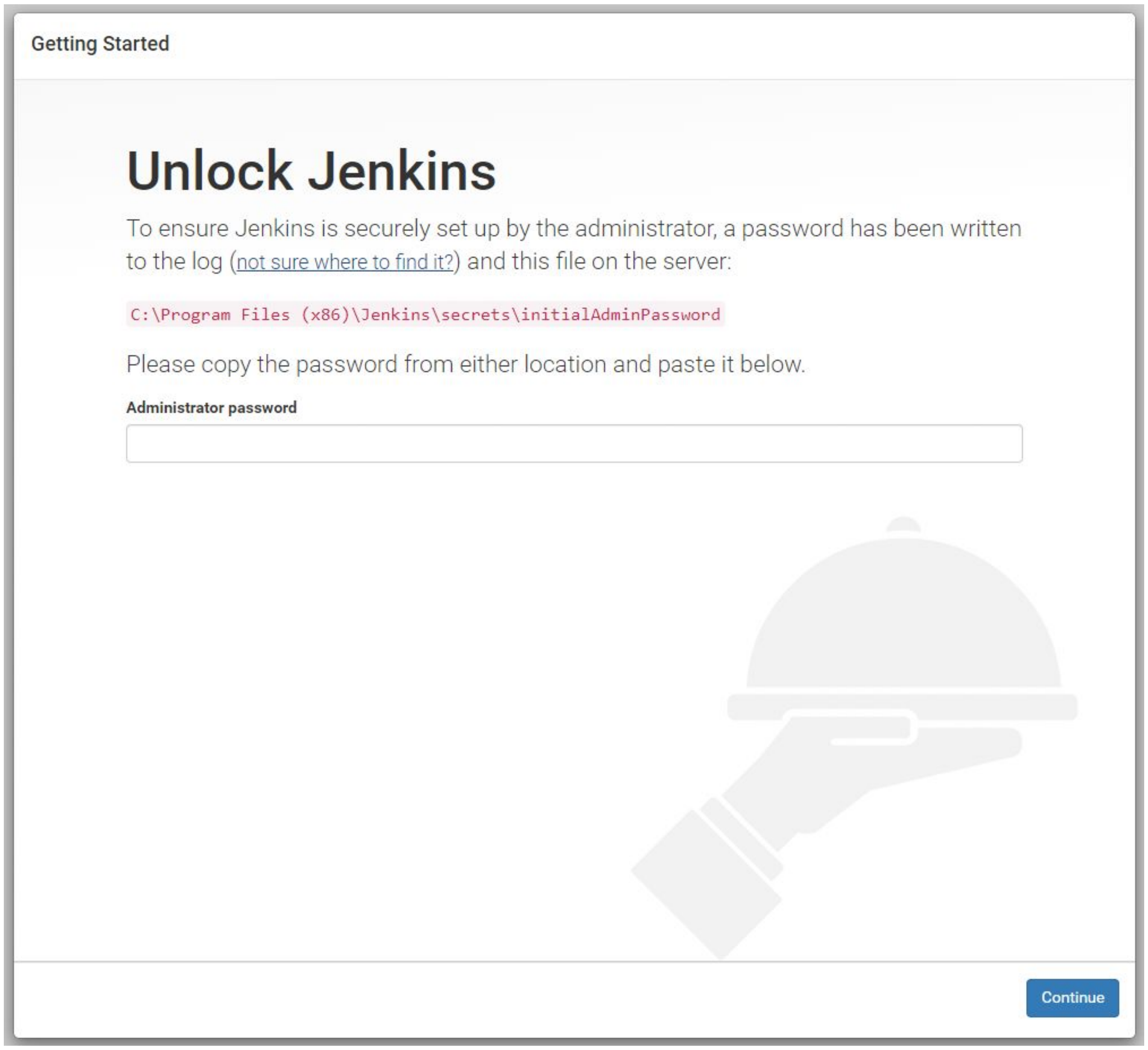
Install Jenkins 2.190.2 to:

C:\Program Files (x86)\Jenkins\


Jenkins 2.190.2 Setup

**Ready to install Jenkins 2.190.2**

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.



Getting Started

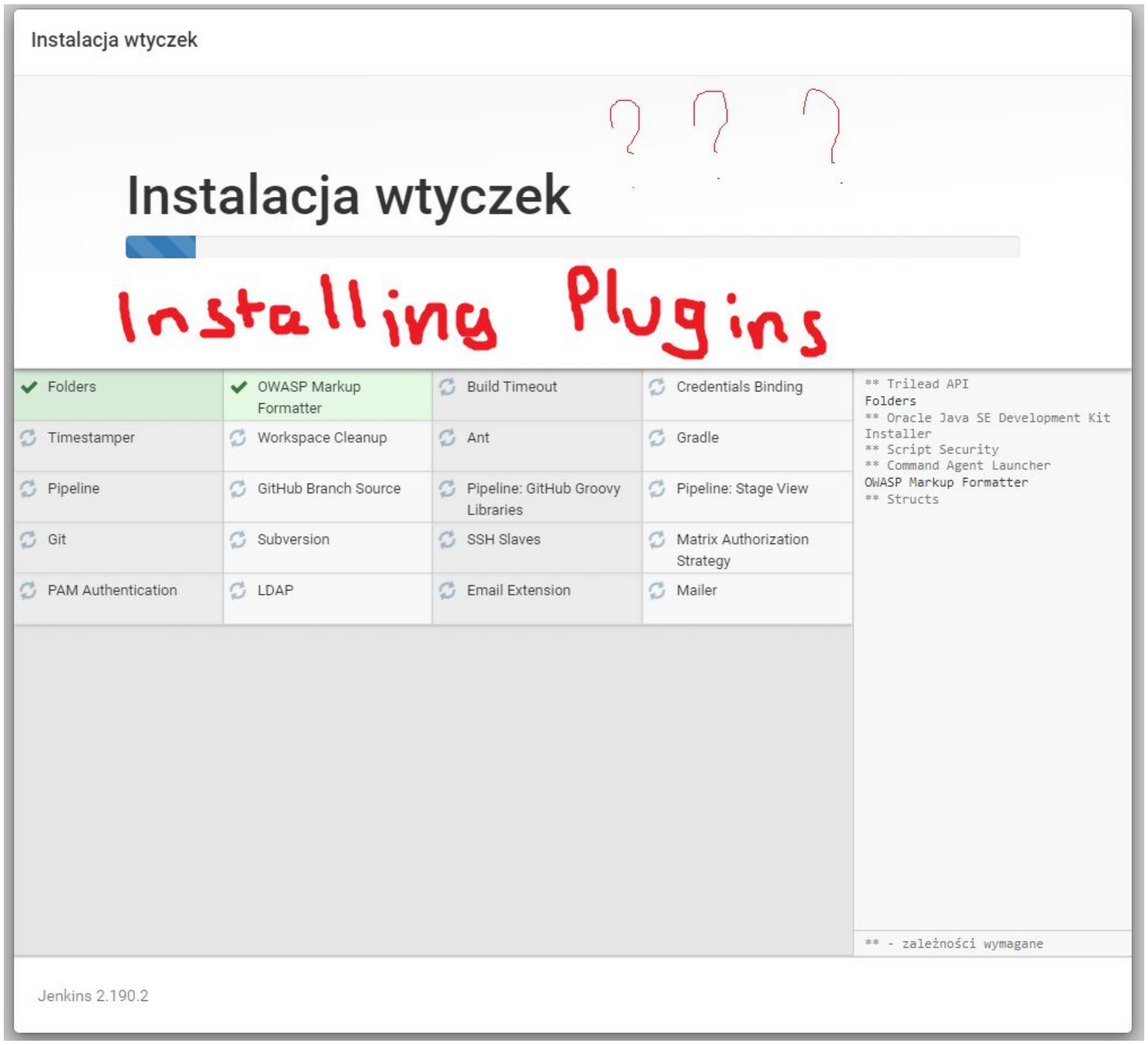
Unlock Jenkins

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:

`C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword`

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

Administrator password



Instalacja wtyczek

Instalacja wtyczek

? ? ?

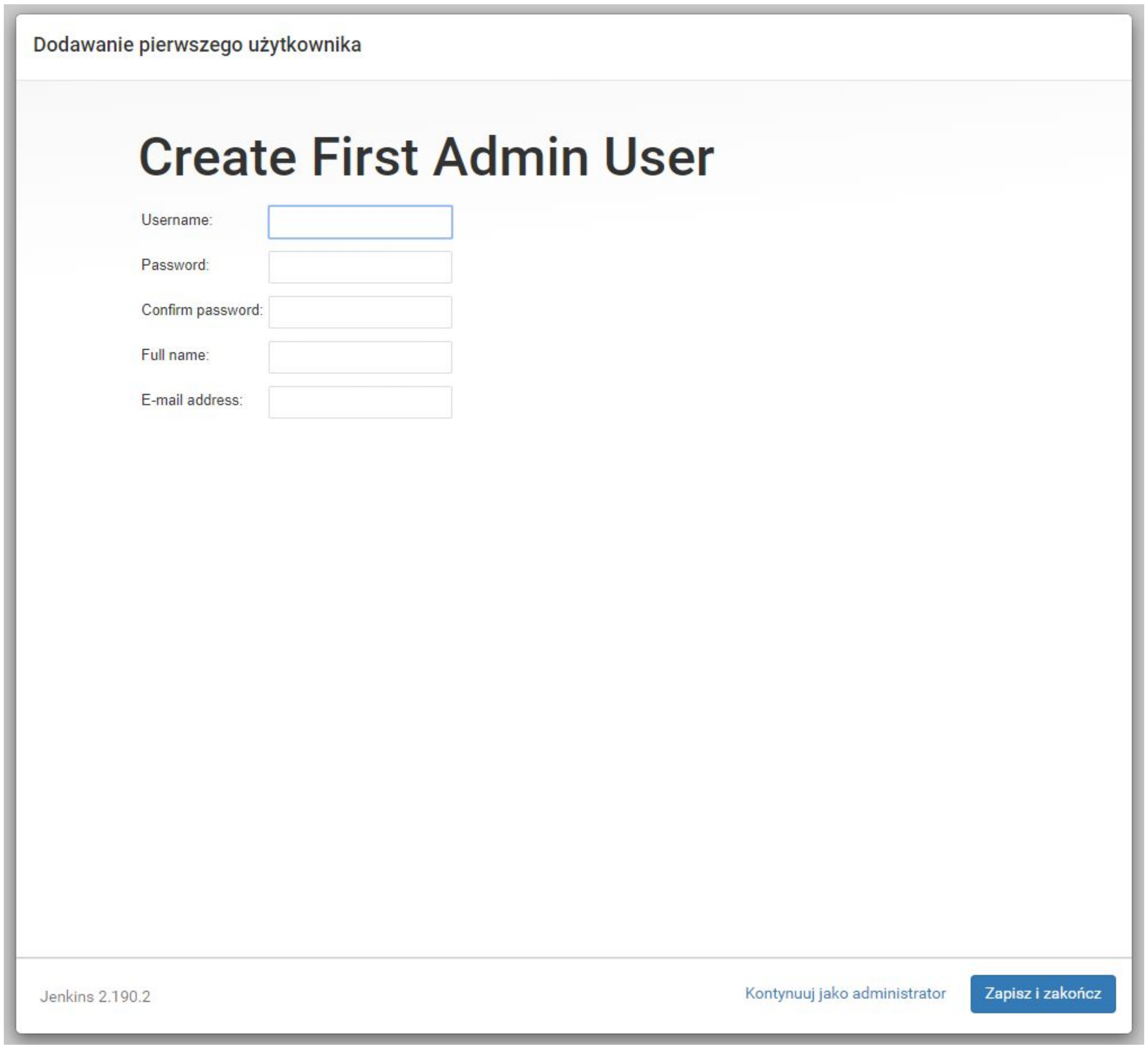
Installing Plugins

| | | | |
|---|---|---|---|
| ✓ Folders | ✓ OWASP Markup Formatter | Build Timeout | Credentials Binding |
| Timestamper | Workspace Cleanup | Ant | Gradle |
| Pipeline | GitHub Branch Source | Pipeline: GitHub Groovy Libraries | Pipeline: Stage View |
| Git | Subversion | SSH Slaves | Matrix Authorization Strategy |
| PAM Authentication | LDAP | Email Extension | Mailer |

```
** Trilead API
Folders
** Oracle Java SE Development Kit
Installer
** Script Security
** Command Agent Launcher
OWASP Markup Formatter
** Structs
```

```
** - zależności wymagane
```

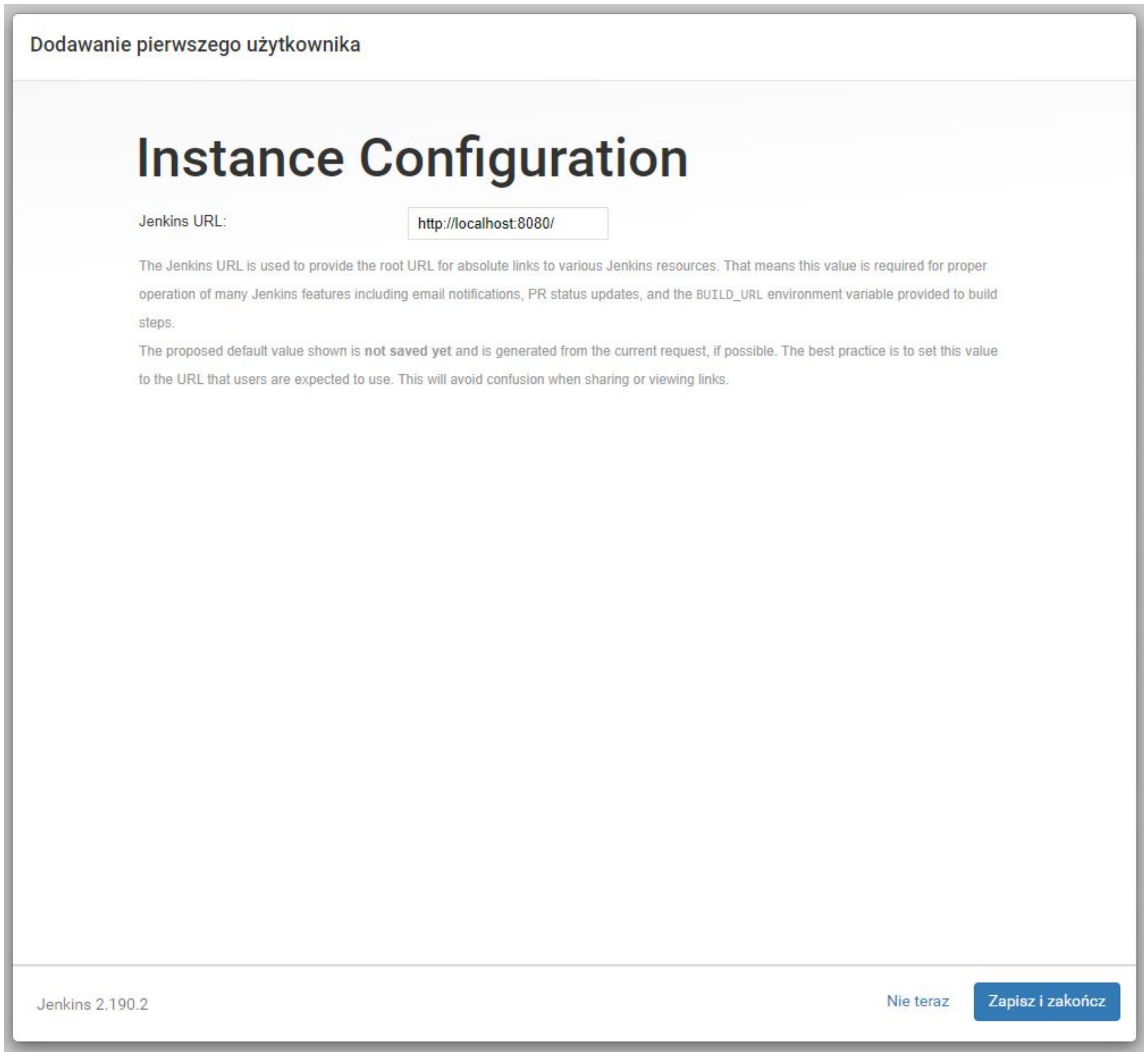
Jenkins 2.190.2



Dodawanie pierwszego użytkownika

Create First Admin User

| | |
|---|---|
| Username: | |
| Password: | |
| Confirm password: | |
| Full name: | |
| E-mail address: | |



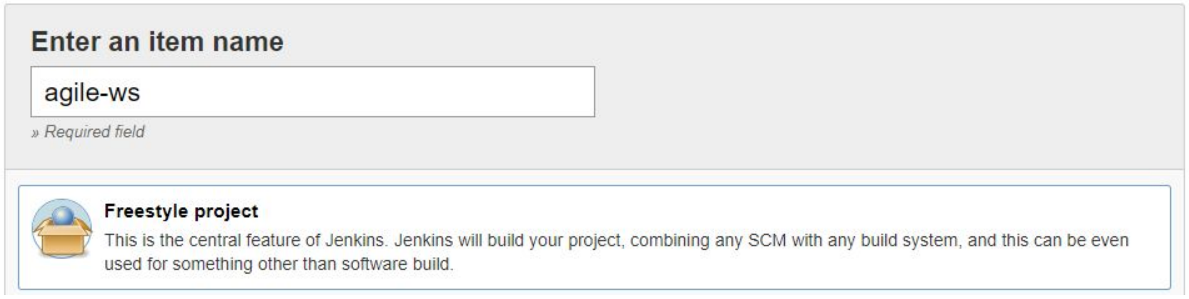
Instance Configuration

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.

1. Create a new project



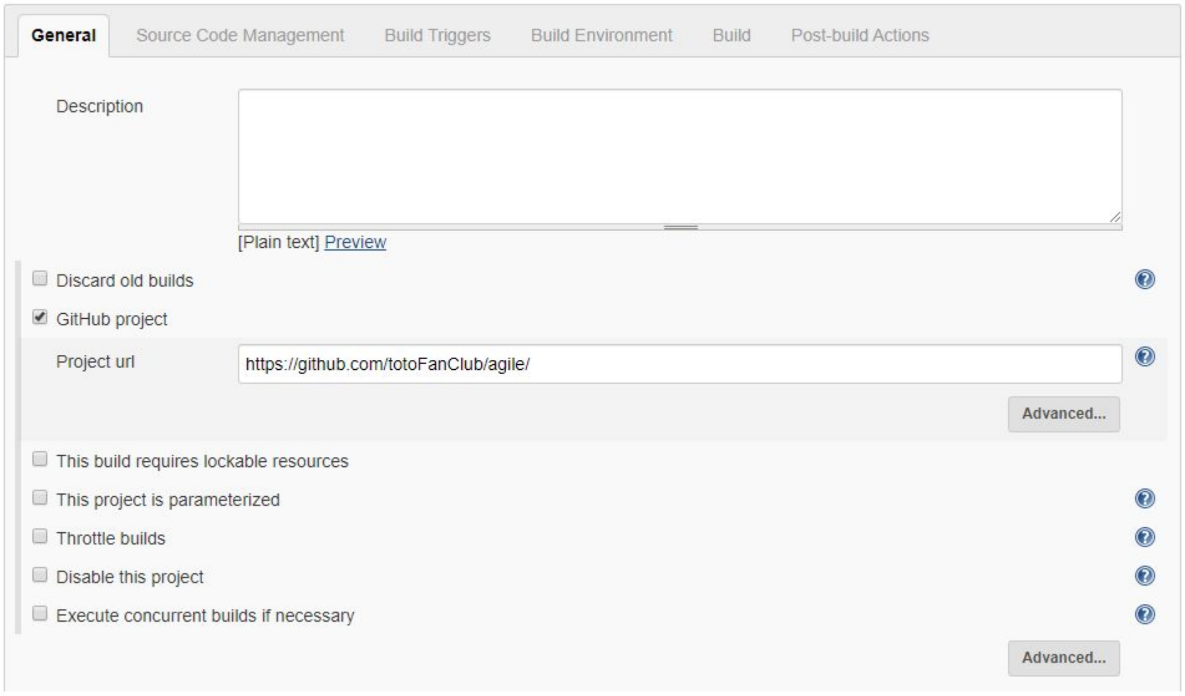
Enter an item name

agile-ws

» Required field

**Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

2. Link to GitHub project



Description

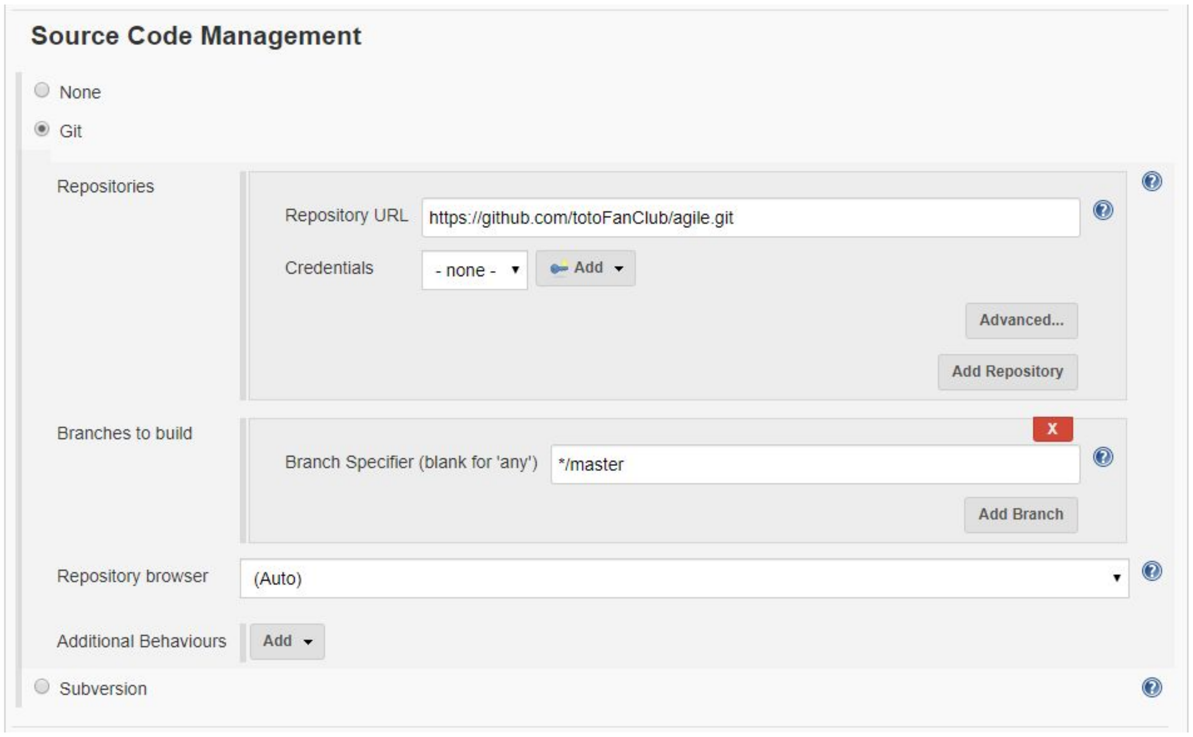
[Plain text] Preview

- [ ] Discard old builds
- [x] GitHub project

Project url: https://github.com/totoFanClub/agile/

- [ ] This build requires lockable resources
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Disable this project
- [ ] Execute concurrent builds if necessary

## 3. Connect Git


Source Code Management

( ) None
(●) Git

Repositories
Repository URL: https://github.com/totoFanClub/agile.git
Credentials: - none -

Branches to build
Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

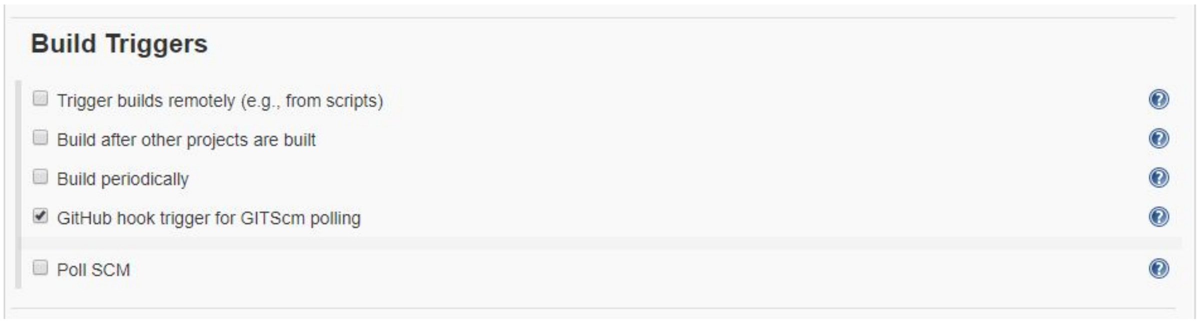
Additional Behaviours

( ) Subversion

Here we encountered an error where the Repository couldn't be found because we originally made it private and changing it to public made it work.

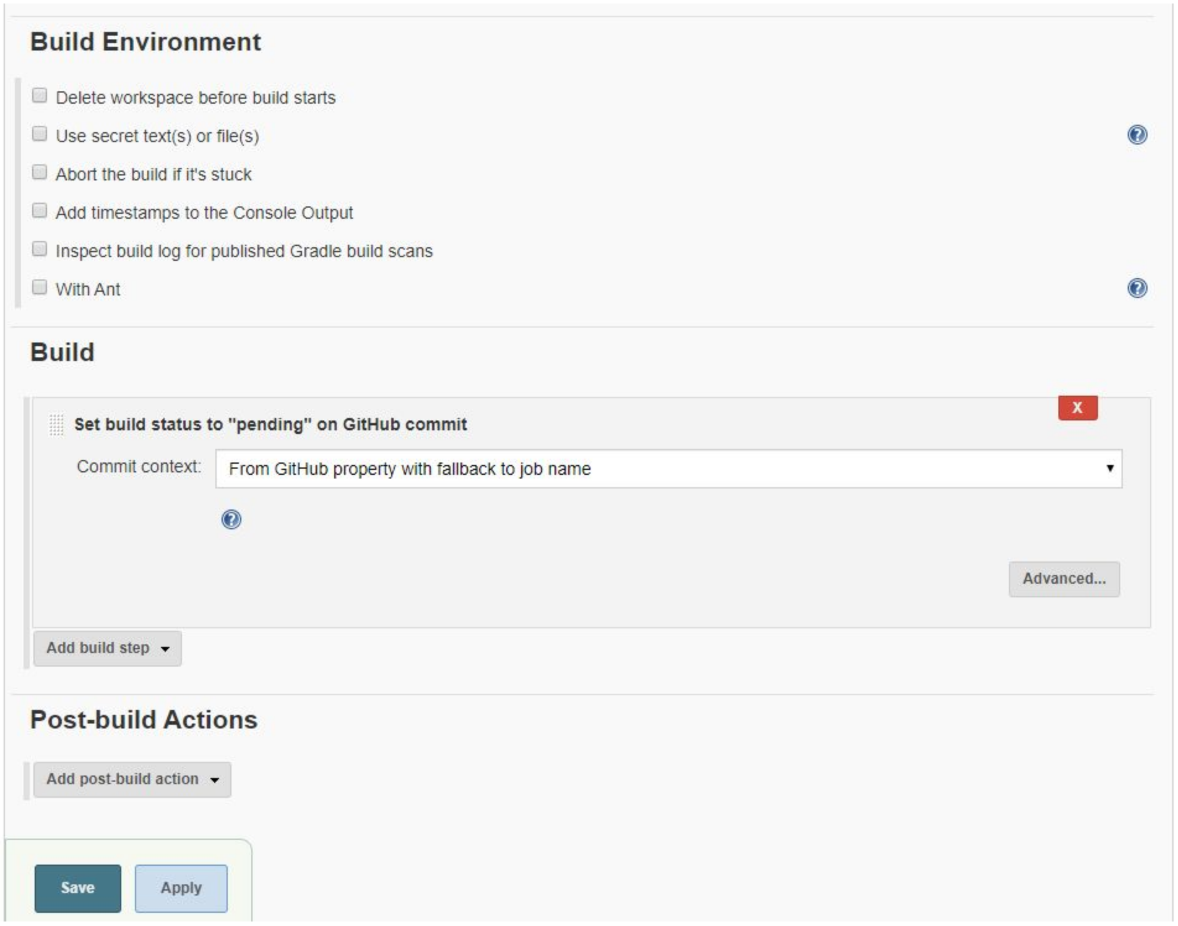
Here we also encountered an error where Jenkins did not recognise that Git was installed and manually pointing it to the *git.exe* file was required.

## 4. Set build trigger



**Build Triggers**

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [x] GitHub hook trigger for GITScm polling
- [ ] Poll SCM



**Build Environment**

- [ ] Delete workspace before build starts
- [ ] Use secret text(s) or file(s)
- [ ] Abort the build if it's stuck
- [ ] Add timestamps to the Console Output
- [ ] Inspect build log for published Gradle build scans
- [ ] With Ant

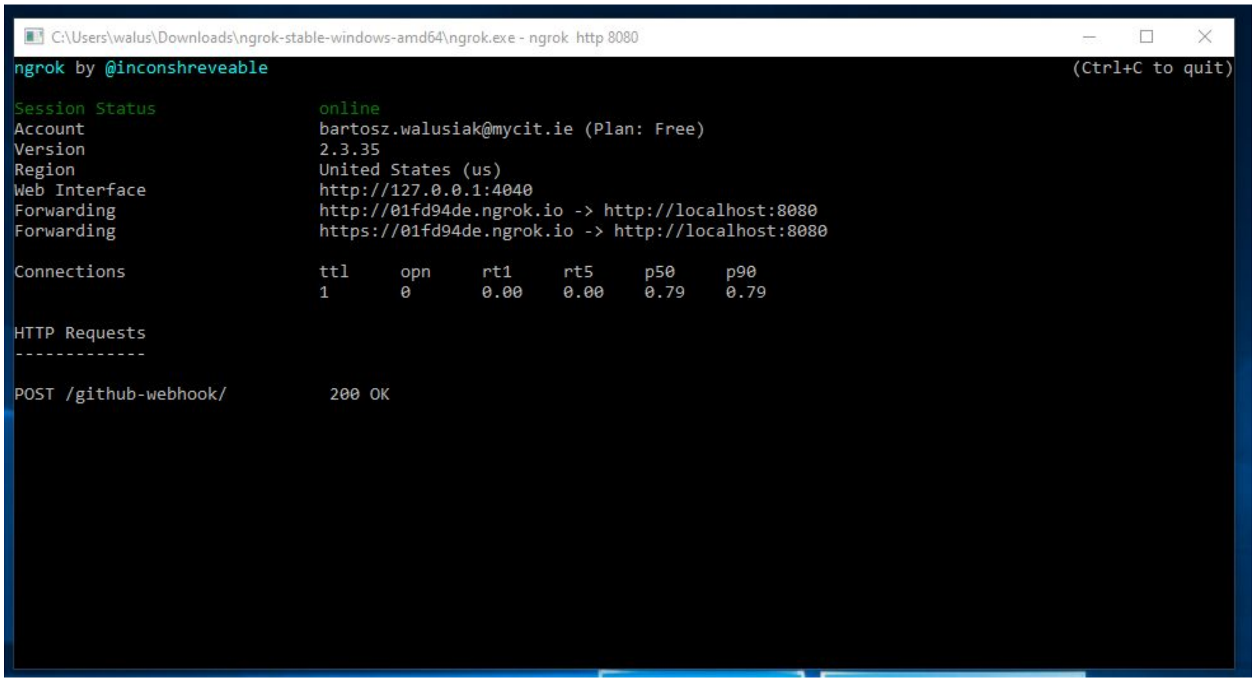
**Build**

**Set build status to "pending" on GitHub commit**

Commit context: From GitHub property with fallback to job name

**Post-build Actions**

## 5. Set up Ngrok to port 8080

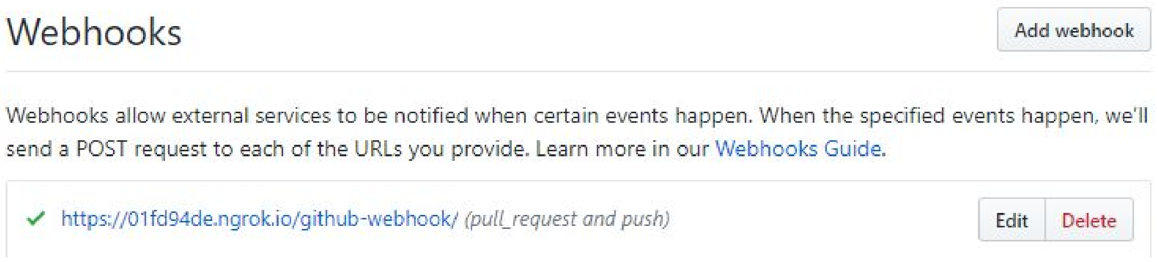


```
C:\Users\walus\Downloads\ngrok-stable-windows-amd64\ngrok.exe - ngrok  http 8080
ngrok by @inconshreveable                                                     (Ctrl+C to quit)

Session Status                online
Account                       bartosz.walusiak@mycit.ie (Plan: Free)
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://01fd94de.ngrok.io -> http://localhost:8080
Forwarding                    https://01fd94de.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              1       0       0.00    0.00    0.79    0.79

HTTP Requests
-------------

POST /github-webhook/          200 OK
```

## 6. Add that link to webhooks in the GitHub project

Here we forwarded the http port that Jenkins was running on that allowed us to assign it to a GitHub

## 7. Webhook



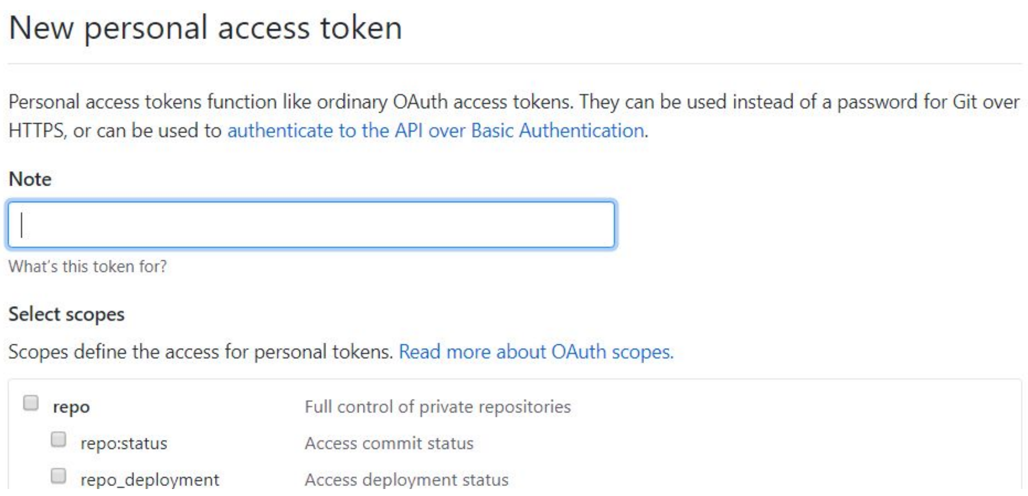
Webhooks

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.

✓ https://01fd94de.ngrok.io/github-webhook/ *(pull_request and push)*

## GitLab

1. Firstly, you must set up a GitHub and GitLab account.
2. Next you create a personal access token in GitHub. This is used to create a connection between GitHub and GitLab.



New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

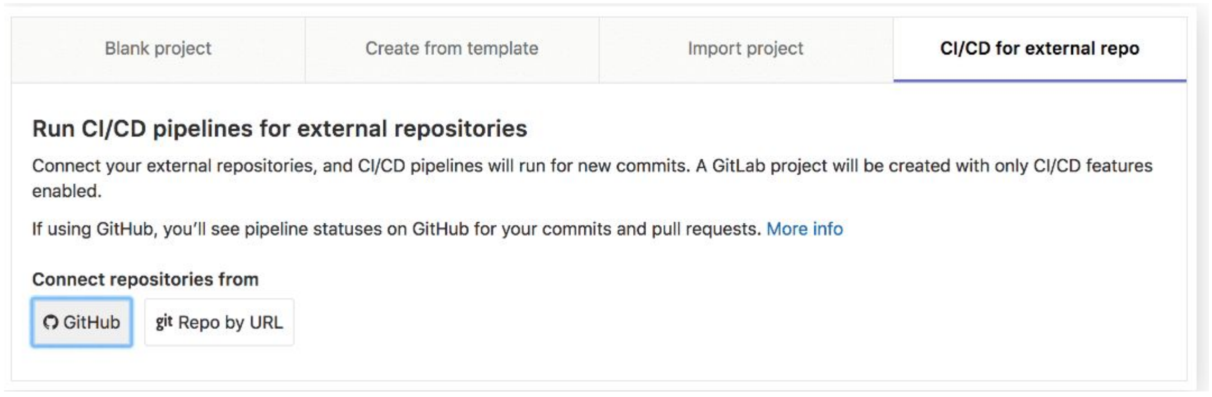
What's this token for?

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

| | |
|---|---|
| repo | Full control of private repositories |
| repo:status | Access commit status |
| repo_deployment | Access deployment status |

3. Next you navigate to GitLab and create a new CI/CD pipeline.



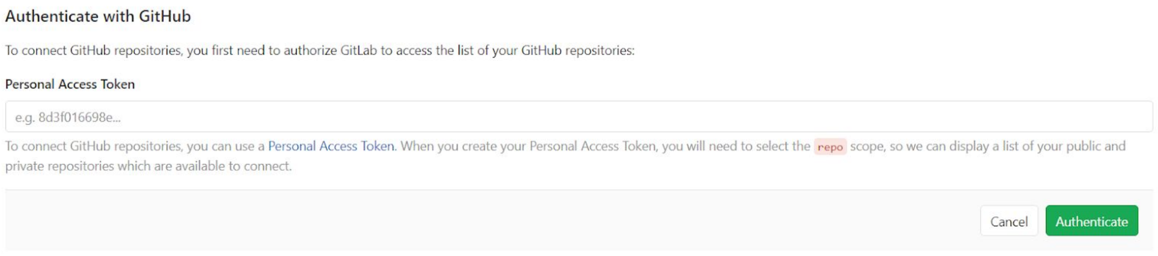
Run CI/CD pipelines for external repositories

Connect your external repositories, and CI/CD pipelines will run for new commits. A GitLab project will be created with only CI/CD features enabled.

If using GitHub, you'll see pipeline statuses on GitHub for your commits and pull requests. More info

Connect repositories from

4. Once you paste your personal access key it will authenticate and create the pipeline.



Authenticate with GitHub

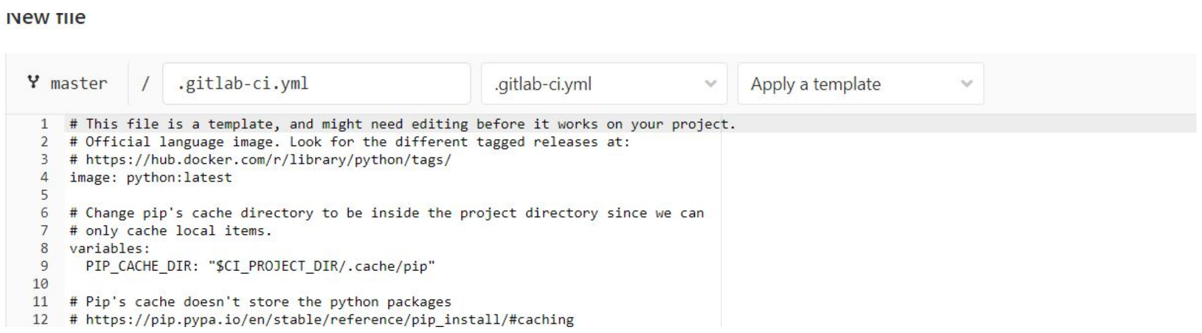
To connect GitHub repositories, you first need to authorize GitLab to access the list of your GitHub repositories:

Personal Access Token

e.g. 8d3f016698e...

To connect GitHub repositories, you can use a Personal Access Token. When you create your Personal Access Token, you will need to select the `repo` scope, so we can display a list of your public and private repositories which are available to connect.

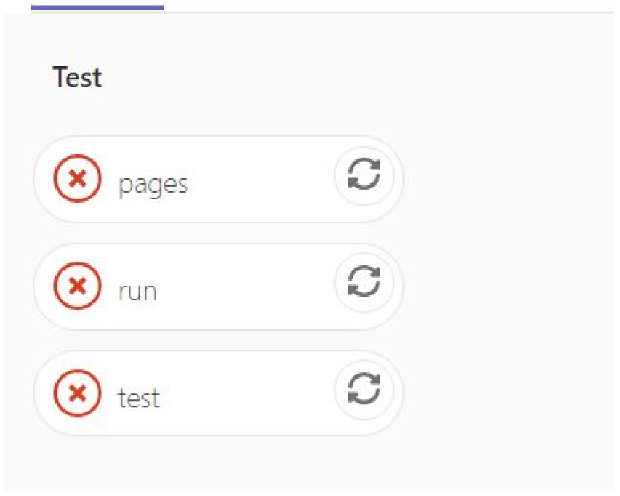
5. After this you must make and configure your .YML file. You can find a template for yml files in GitLab and you must copy this file into your GitHub. For us it's a python template.



New file

master / .gitlab-ci.yml

```
# This file is a template, and might need editing before it works on your project.
# Official language image. Look for the different tagged releases at:
# https://hub.docker.com/r/library/python/tags/
image: python:latest

# Change pip's cache directory to be inside the project directory since we can
# only cache local items.
variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

# Pip's cache doesn't store the python packages
# https://pip.pypa.io/en/stable/reference/pip_install/#caching
```



## Evaluation

Creating the connection between GitHub and GitLab was very simple. Once we followed the above steps the connection was created, and changes propagated through GitHub to GitLab.

The problem we ran into was with the YML file. For us the template didn't work as intended and the documentation about how to configure and set them up in the GitLab documentation was very confusing. This meant that our pipeline set up correctly and our changes propagated correctly but GitLab ran into errors in the pipeline.

Overall we found Jenkins to be much better documented, and even though the setup process was longer and more complex, it was easier as the documentation guided us through all of the steps.

We encountered three problems detailed in the step-by-step, and each time there was plenty of community guides on how to fix those errors, on StackOverflow etc.

## Recommendation

Based on our research, the Continuous Integration tool we recommend using is GitLab. Compared to Jenkins CI, GitLab was easier to set up, particularly for small-medium sized projects.

Jenkins requires an existing installation and an activated pipeline plug-in when setting up. Jenkins does not provide repositories, so the source-repo must be defined. Jenkins maintains a file called jenkinsfile which saves the phases and stages about the build configuration. If further features are required, plugins can be utilised. When running through the pipeline, the terminal shows whether the stages have passed or failed.

GitLab is simple to setup as it offers repositories. Stages are described in the gitlab-ci.yml configuration file. First, a sequence of stages will be executed in a specified order. Following that, each job is described and configured with different options, and can run parallel with other jobs in the same stage.

GitLab allows us to view the status of every job inside a stage, whereas Jenkins shows the stages as a whole. This means GitLab is more efficient to debug. It is easy to integrate new jobs into a stage, and is scalable due to concurrent runners. It offers more flexibility as a continuous integration tool. GitLab is better suited to small and medium-sized projects, yet is still a contender for larger projects. Jenkins requires more configuration that you must do yourself. However, it may be the better option for large-scale projects and deployment jobs.