Building a CI/CD Pipeline using Gitlab

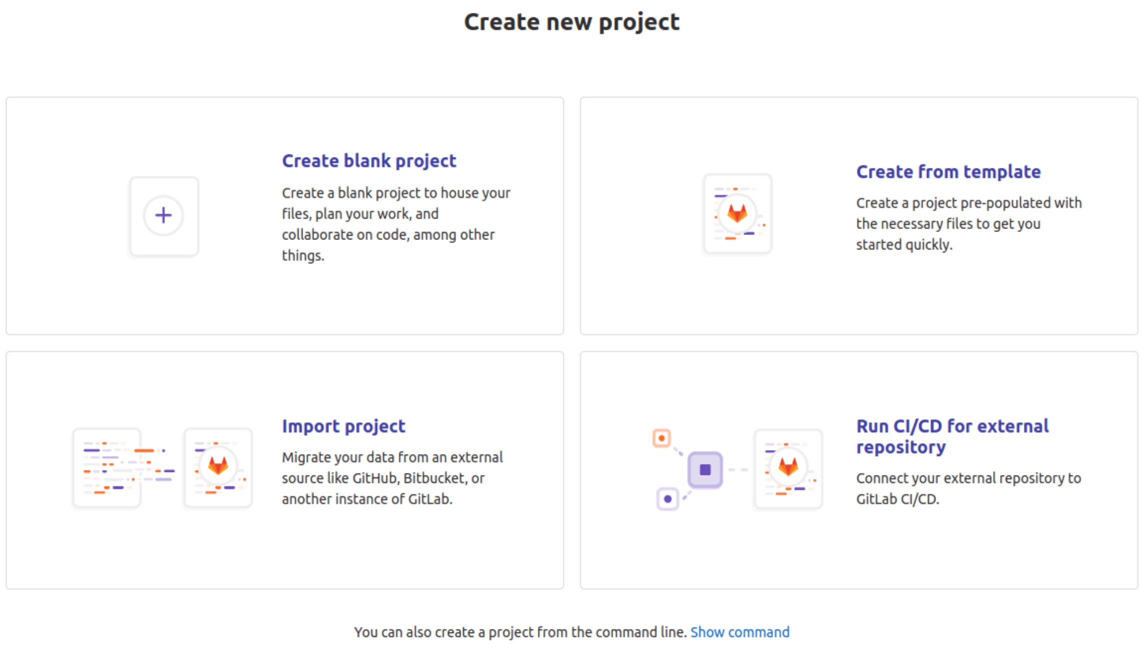
GitLab is a Git-based platform, similar to Github, that provides an array of tools and technologies for various aspects of the software development lifecycle (SDLC).
Gitlab allows users to host repositories using the popular version control system: Git. It also allows users to track proposals for new implementations, generate bug reports, review code, etc.
Another essential feature, which we will use in this tutorial, is its build-in continuous integration tool called Gitlab-ci.
In this tutorial, we are going to explore the Gitlab and Gitlab-ci by building a CI/CD pipeline and running it using the UI provided by Gitlab.

Prerequisites:

Setting up Gitlab
Building a CI/CD pipeline
Code
Conclusion
Setting up Gitlab
Setting up a Gitlab account.

Once you sign in, it will redirect you to the homepage where you can explore various options such as creating a new project, a group or just explore some public repositories.
Follow these steps to set up a new project on Gitlab:

- On the homepage, click on “create blank project” to initialize a new repository.



Create new project

Create blank project
Create a blank project to house your files, plan your work, and collaborate on code, among other things.

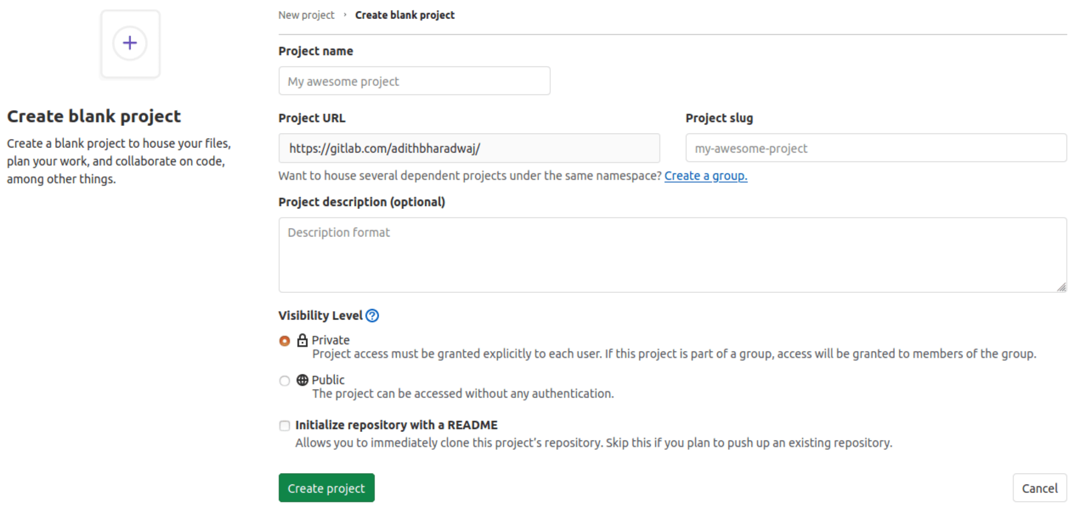
Create from template
Create a project pre-populated with the necessary files to get you started quickly.

Import project
Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.

Run CI/CD for external repository
Connect your external repository to GitLab CI/CD.

You can also create a project from the command line. Show command

- Gitlab will redirect you to a form where you fill in the details of your project, such as title, description, etc. You can also choose to make your project public or private. After you finish adding the details, click on “create project”.



New project › Create blank project

Create blank project
Create a blank project to house your files, plan your work, and collaborate on code, among other things.

Project name
My awesome project

Project URL
https://gitlab.com/adithbharadwaj/

Project slug
my-awesome-project

Want to house several dependent projects under the same namespace? Create a group.

Project description (optional)
Description format

Visibility Level
[✓ Private] Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.
Public The project can be accessed without any authentication.

Initialize repository with a README
Allows you to immediately clone this project’s repository. Skip this if you plan to push up an existing repository.

Create project
Cancel

- Click on “clone” and copy the HTTPS URL to clone the project. Open a terminal or command line and enter the following command to clone the project into your local machine:

```
git clone <https clone URL>
```

Now that you have the project on your local machine, we can start building our own CI/CD pipeline and deploy it on Gitlab using Gitlab-ci.

Building a CI/CD pipeline

Continuous Integration can be defined as a process, in the software development lifecycle, where engineers and developers push code into a git-based repository. If developers do this frequently, an automated process, to verify and build the code, is included to save time and costs.

According to the official documentation, GitLab CI/CD is a powerful tool built into GitLab that allows you to apply all the continuous methods (Continuous Integration, Delivery, and Deployment) to your software with no third-party application or integration needed.

Users can configure GitLab CI/CD by creating a file called .gitlab-ci.yml in the root directory of the project. This file is written in a simple and easy to use language called YAML. This is a good article on YAML and its basics.

Once you define the .gitlab-ci.yml file, Gitlab creates a pipeline, that runs whenever there is a change to the code in the repository. These pipelines can have single or multiple stages that run one after the other(in series).

Each stage can consist of multiple jobs that are executed in parallel by the gitlab-runner, that is an application that works with GitLab to run jobs in a pipeline.

Code

Let’s build a simple CI pipeline to run a Python script whenever we push changes to our repository.

Create a Python script called test.py in the repository and add the following lines of code into the file.

The script prints “hello world” and “testing” 5 times. This is just an example to help you understand how to create and run pipelines. You can run complex scripts and commands as well.

```python
if __name__ == '__main__':
  print('hello world')
  for i in range(5):
    print('testing', i)
```

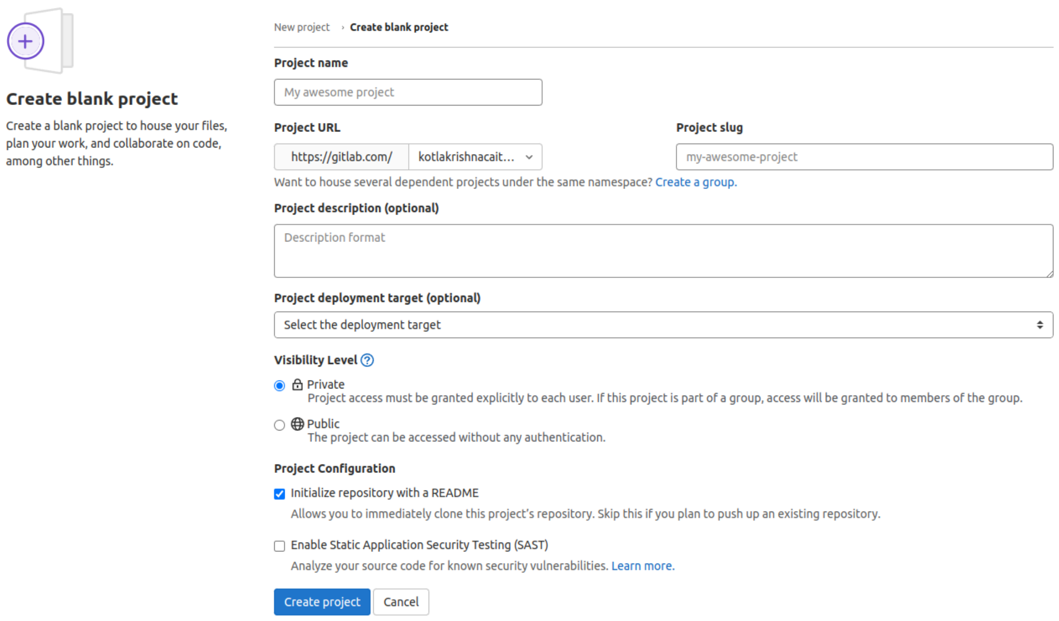
Create a file called .gitlab-ci.yml in the repository and add the following snippet of code. The variables section defines a list of variables that can be used in the pipeline.

I have declared a variable called example, whose value is set to “this is an example variable”. Variables are declared as key-value pairs (separated by a semicolon) in YAML.

The stages section defines a list of stages in the pipeline. These stages are executed one after the other, and hence, it’s important to order them based on dependencies. In other words, stages that are dependant on other stages must be declared after all of their dependencies.

The next lines define the actual stages that run in series. They start with the name of the stage followed by the tag stage, that defines which stage it is a part of. The script tag defines a shell script or command that is executed by the runner.

We can use the echo command to print something on the terminal and the cat command to display the contents of the Python script. We can also use echo to print the value of the variables that are defined in the file. These variables can be accessed by using the $ symbol.



New project › **Create blank project**

Create blank project

Create a blank project to house your files, plan your work, and collaborate on code, among other things.

Project name

My awesome project

Project URL

https://gitlab.com/ kotlakrishnacait...

Project slug

my-awesome-project

Want to house several dependent projects under the same namespace? Create a group.

Project description (optional)

Description format

Project deployment target (optional)

Select the deployment target

Visibility Level

[✓ Private] Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

Public The project can be accessed without any authentication.

Project Configuration

[✓] Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

[ ] Enable Static Application Security Testing (SAST)
Analyze your source code for known security vulnerabilities. Learn more.

Create project Cancel

```
variables:
  example: this is an example variable

stages:
  - stage1
  - stage2

build:
  stage: stage1
  script:
    - echo "We are currently in stage 1"
    - echo "These are the contents of test.py"
    - cat test.py
    - echo $example

test:
  stage: stage2
  script:
    - echo "We are currently in stage 2"
    - echo "running python script"
    - python3 test.py
```

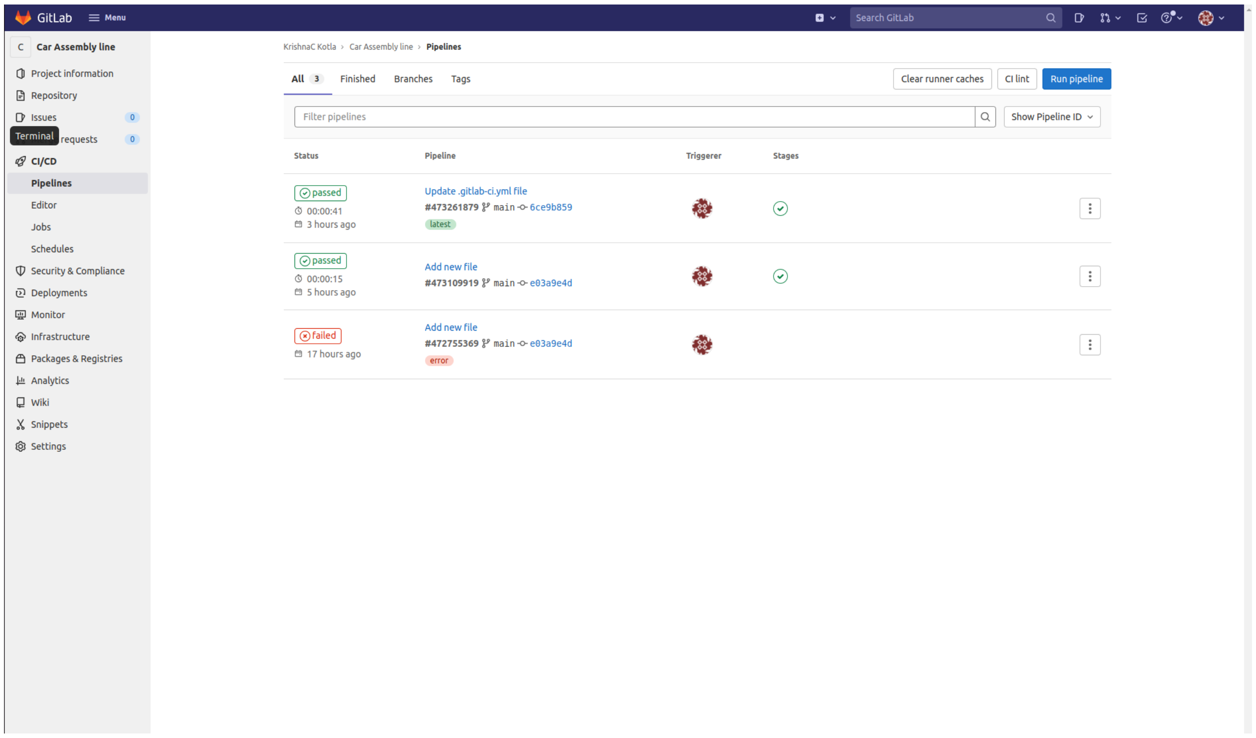
Run the following commands to push the changes to the repository:

```
git add -A
```

```
git commit -m "initial commit"
```

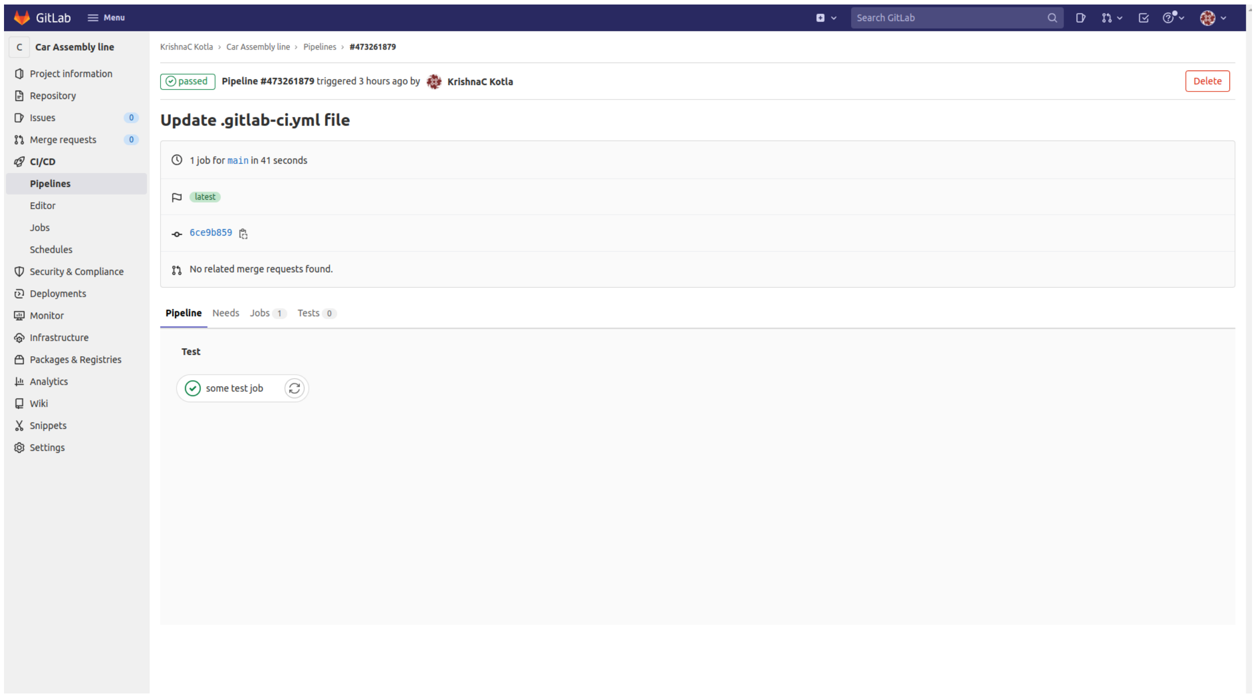
```
git push origin master
```

You will be prompted to enter your Gitlab username and password. Once you correctly enter the username and password, the changes will be pushed to the repository and Gitlab will instantly run the code in the .gitlab-ci file by creating a pipeline.



Status	Pipeline	Triggerer	Stages
passed 00:00:41 3 hours ago	Update .gitlab-ci.yml file #473261879 main 6ce9b859 latest		✓
passed 00:00:15 5 hours ago	Add new file #473109919 main e03a9e4d		✓
failed 17 hours ago	Add new file #472755369 main e03a9e4d error		

Navigate to the CI/CD section on the left navbar to view a list of pipelines. The latest one will be at the top and will have the “latest” tag. It will also show us whether the pipeline passed or failed. Click on “passed” or “failed” to view the stages of the pipeline and all the steps that were executed.



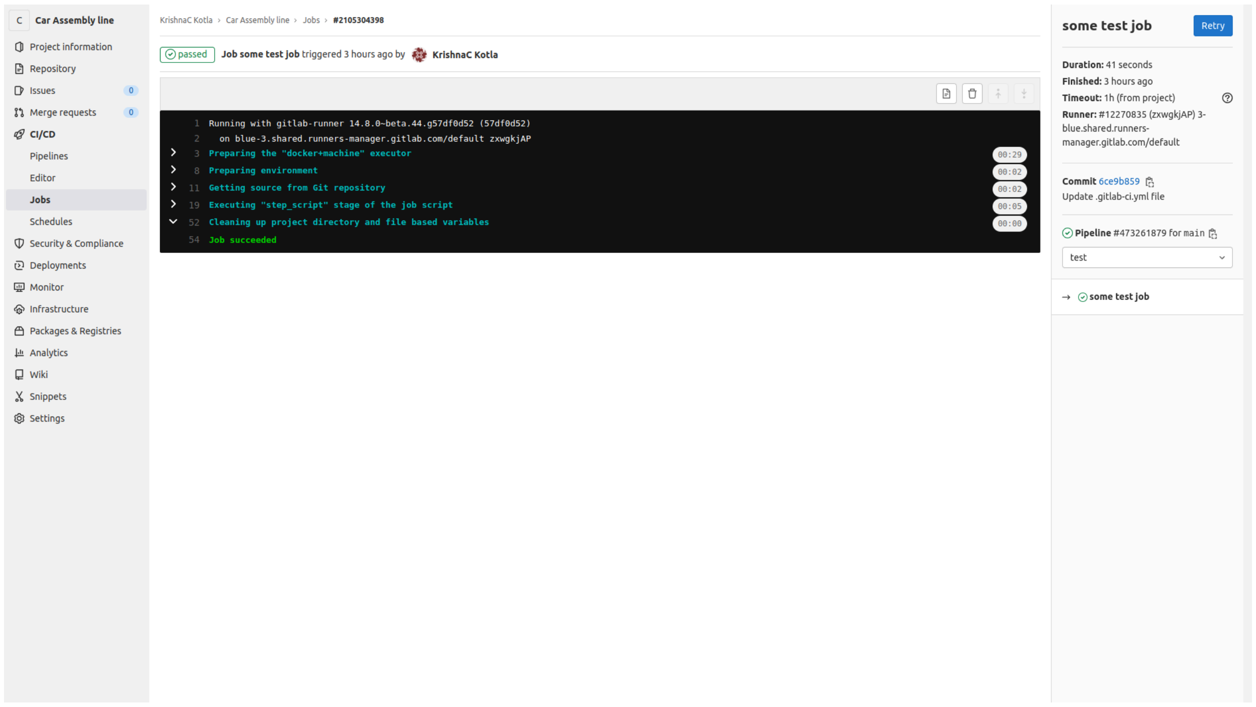
Pipeline #473261879 triggered 3 hours ago by KrishnaC Kotla

Update .gitlab-ci.yml file

1 job for main in 41 seconds

No related merge requests found.

You will see the different stages that you defined and whether each stage passed or not. You can click on the individual stages to view the jobs executed in each stage in detail.



passed **Job some test job** triggered 3 hours ago by **KrishnaC Kotla**

```
Running with gitlab-runner 14.8.0~beta.44.g57df0d52 (57df0d52)
  on blue-3.shared.runners-manager.gitlab.com/default zxwgkjAP
Preparing the "docker+machine" executor                          00:29
Preparing environment                                            00:02
Getting source from Git repository                               00:02
Executing "step_script" stage of the job script                  00:05
Cleaning up project directory and file based variables           00:00
Job succeeded
```

some test job

Duration: 41 seconds
Finished: 3 hours ago
Timeout: 1h (from project)
Runner: #12270835 (zxwgkjAP) 3-blue.shared.runners-manager.gitlab.com/default

Commit 6ce9b859
Update .gitlab-ci.yml file

Pipeline #473261879 for main
test

→ **some test job**

Conclusion

In conclusion, Gitlab-ci is an impeccable tool that makes the lives of software engineers easier by providing a robust solution without requiring any third-party library or tool.

In this tutorial, we understood the basics of GitLab and GitLab-ci by building a simple CI pipeline that automates the process of testing and building your code without the need for human intervention.

Now that you have grasped the basics, you can go ahead and build your own pipeline. Whether you are a software developer, a system administrator, or a DevOps engineer, GitLab-ci is an indispensable part of your toolkit.