GitLab/Git

GitLab

Reference- <https://docs.gitlab.com/user/get_started/>

GitLab is an open-source platform built around Git, the distributed version control system developed by Linus Torvalds. It provides a comprehensive set of tools and features to manage Git repositories, project planning, continuous integration/continuous deployment (CI/CD), code review, issue tracking, and more. Essentially, GitLab consolidates the entire software development lifecycle into one seamless interface, simplifying the process and making it more efficient.

Important Points about GitLab

1. At its core, GitLab is a powerful Git repository manager. It allows developers to create, clone, push, and pull repositories, providing version control capabilities that ensure changes to code are tracked, managed, and merged efficiently.

2. GitLab offers a built-in issue tracking system, enabling teams to create, assign, and track issues, bugs, and feature requests. It also provides a project management board with customizable workflows to visualise tasks and progress.

3. One of GitLab’s standout features is its integrated CI/CD pipelines. Developers can automate the testing, building, and deployment processes, ensuring code changes are thoroughly tested before being merged into the main codebase and deployed to production.

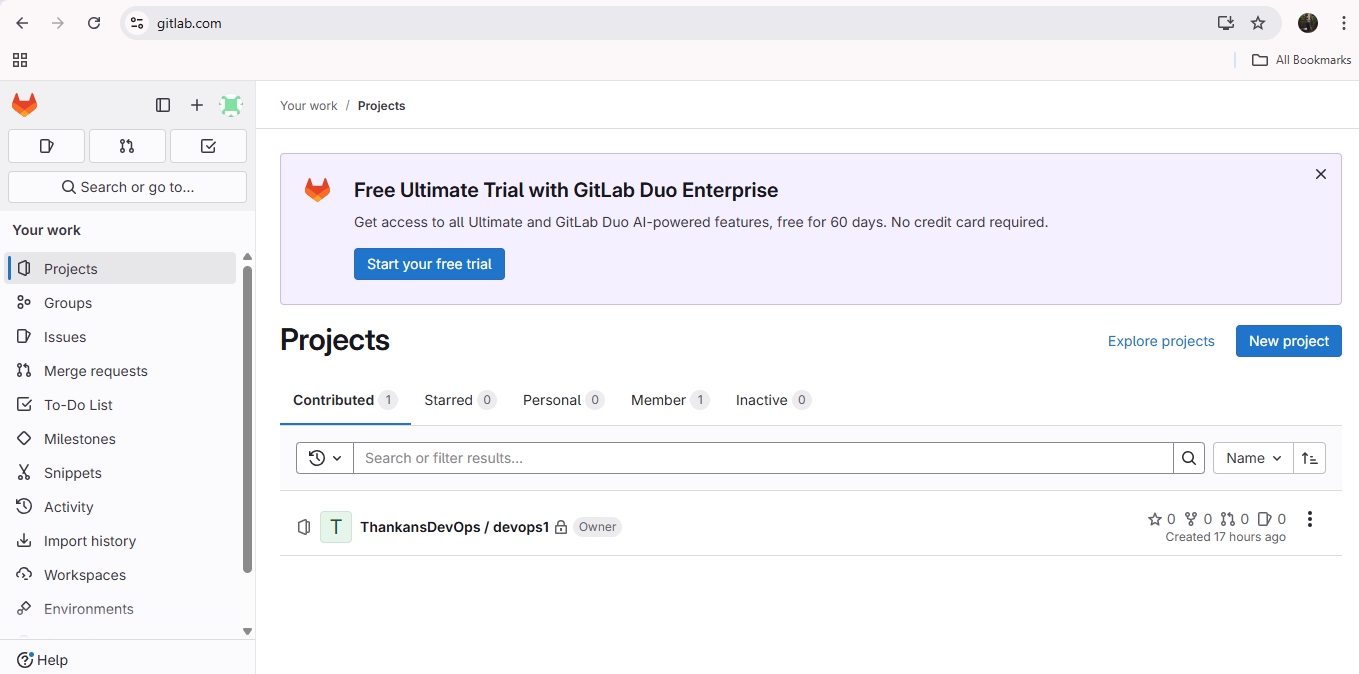
BENEFITS OF GITLAB-

* Enhanced Collaboration: GitLab centralises development activities, facilitating seamless collaboration among team members regardless of their geographical location.
* Improved Productivity: With automated CI/CD pipelines and streamlined workflows, developers can focus more on writing code and less on repetitive tasks, improving overall productivity.
* Faster Time to Market: GitLab’s built-in code review process ensures that code changes are thoroughly assessed, leading to higher-quality code and reduced bugs in the production environment.
* Transparency and Accountability: The issue tracking and project management features provide transparency on project progress and help hold team members accountable for their tasks.
* Unified Platform: GitLab replaces multiple tools, reducing context shifts, enhancing speed, and cutting DevOps toolchain costs.

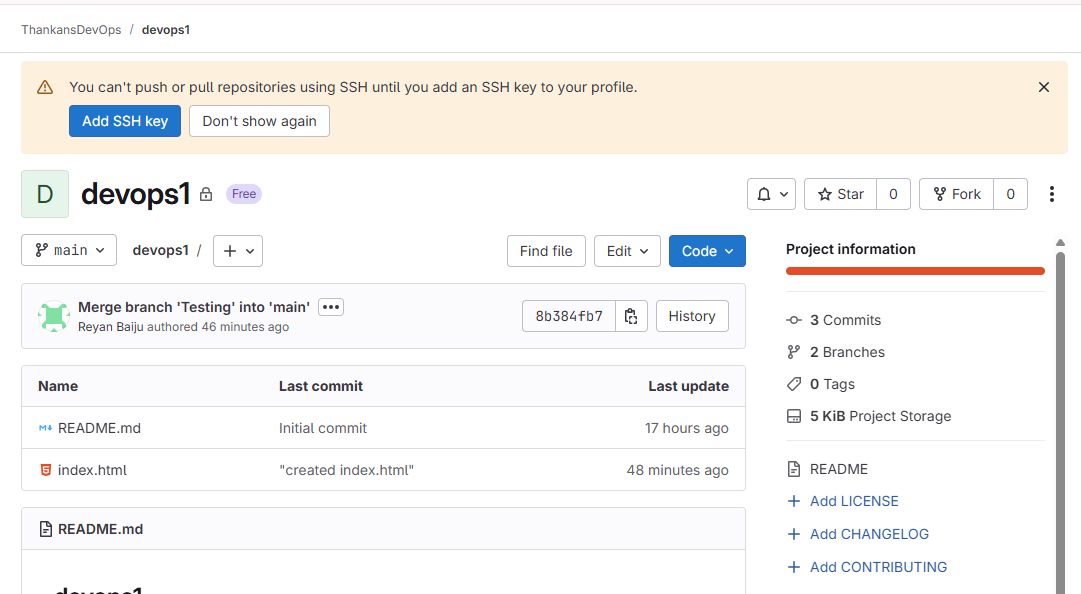
Using GitLab

GitLab can be accessed either on the gitlab cloud(on the internet) or on a self hosted server.

GitLab.com-



Repository page-

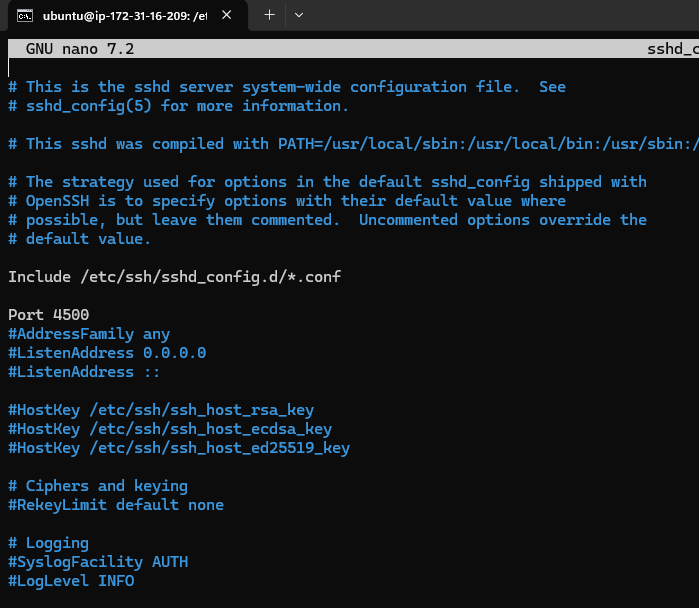


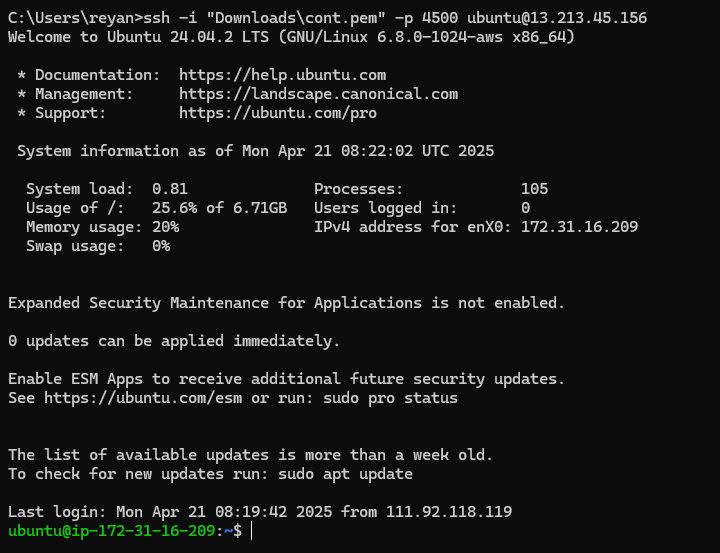
Self Hosting GitLab using Docker Compose-

Reference- <https://docs.gitlab.com/install/docker/installation/#install-gitlab-by-using-docker-compose>

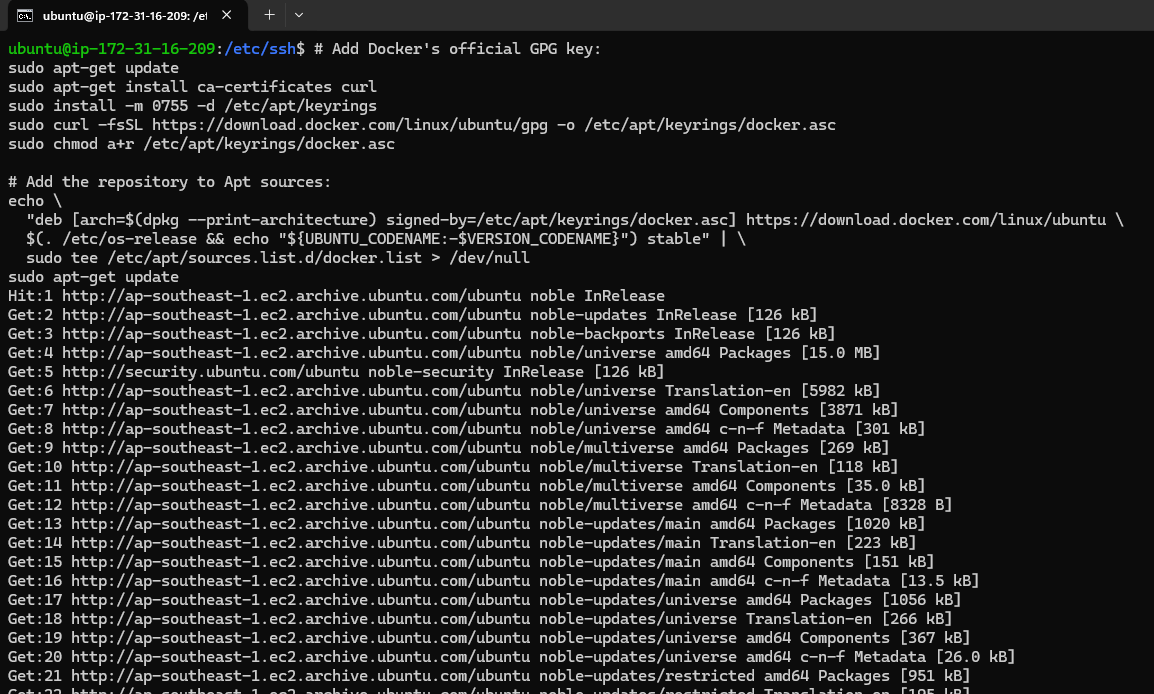
As gitlab uses port 22 to interact with git by default, I am going to map a new port for ssh into the server.

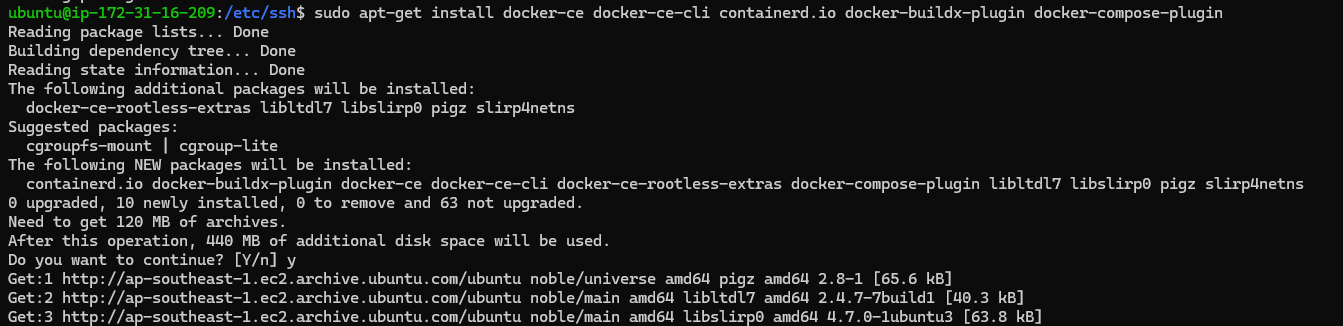
Just edit the sshd\_config file and restart the vm.





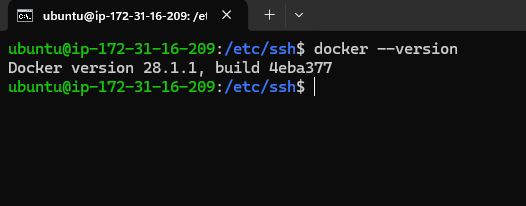
Now, we can install docker from this site- <https://docs.docker.com/engine/install/ubuntu/>





Now, docker is installed successfully.

We can check the docker version using docker –version.

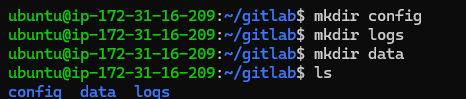


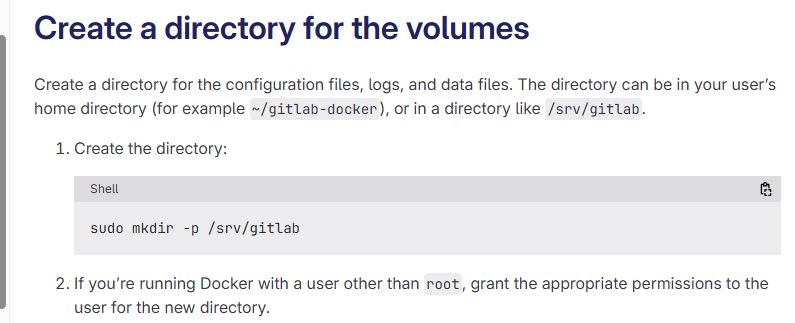
Now, we can pull the docker image from dockerhub, link- <https://hub.docker.com/r/gitlab/gitlab-ce>

But, before that, we have to create some directories to use as volumes for docker compose.

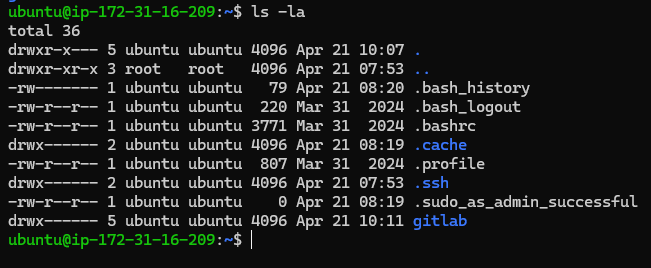
I’m creating a new directory called “gitlab” in my home directory.

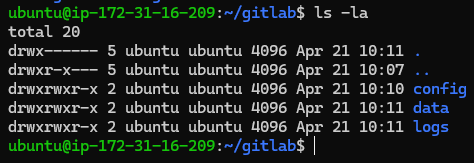
And making the required directories inside it.



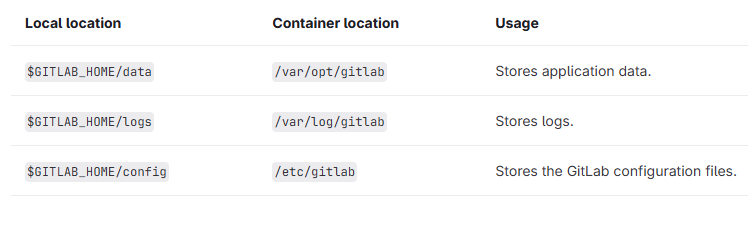


I'm giving appropriate permissions to my user to use the directory.



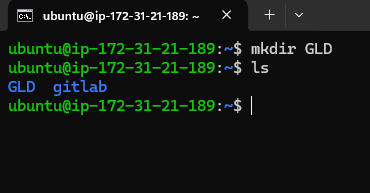


These are the uses of these directories.

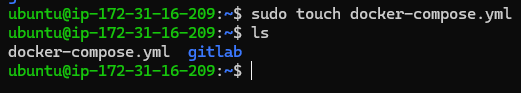


Now we can mention it in the .yaml file.

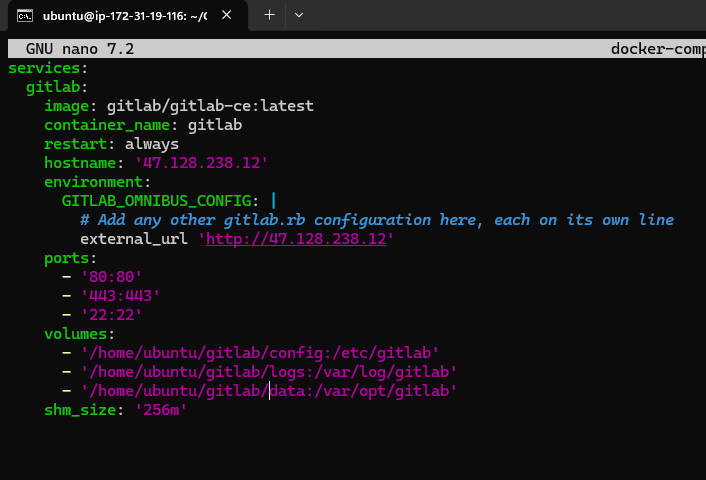
I’m creating a new directory called GLD to create the docker-compose.yml file.



Create a docker-compose.yml file.

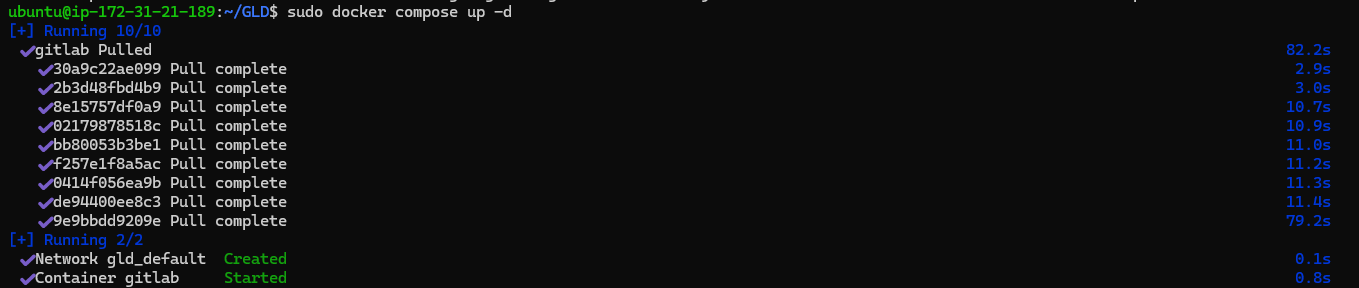


Add your configuration in that file-



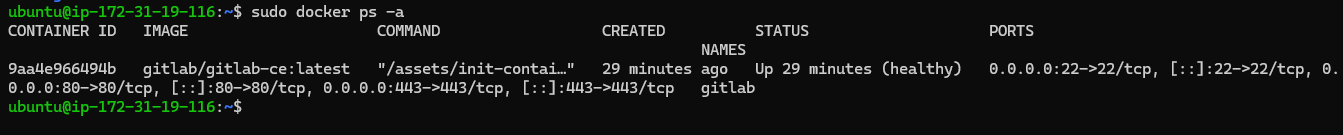
Save the file.

Now, run the command “docker compose up -d”.



Gitlab is running successfully.

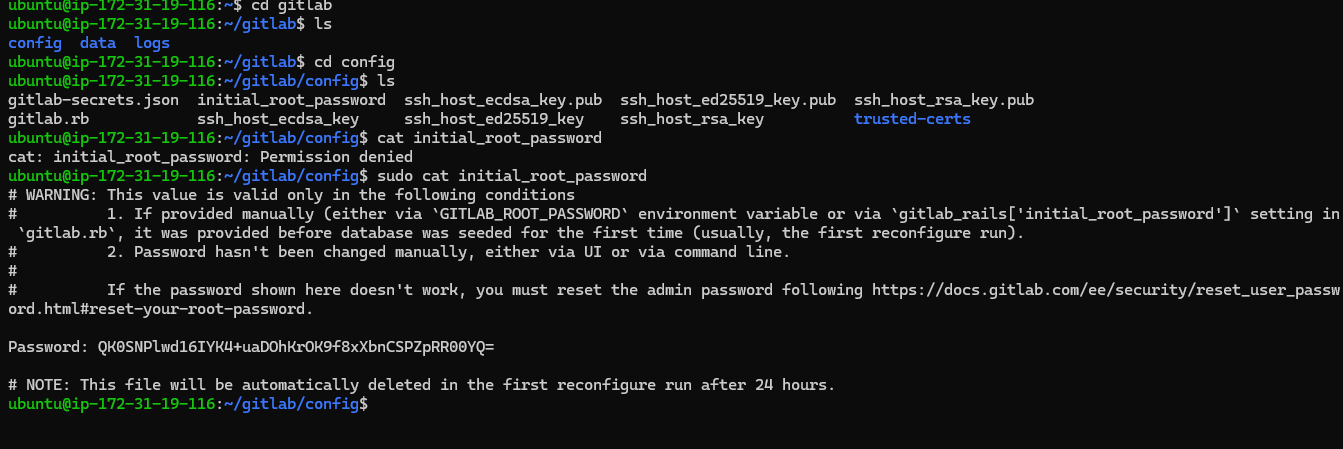
Use command sudo docker compose ps.



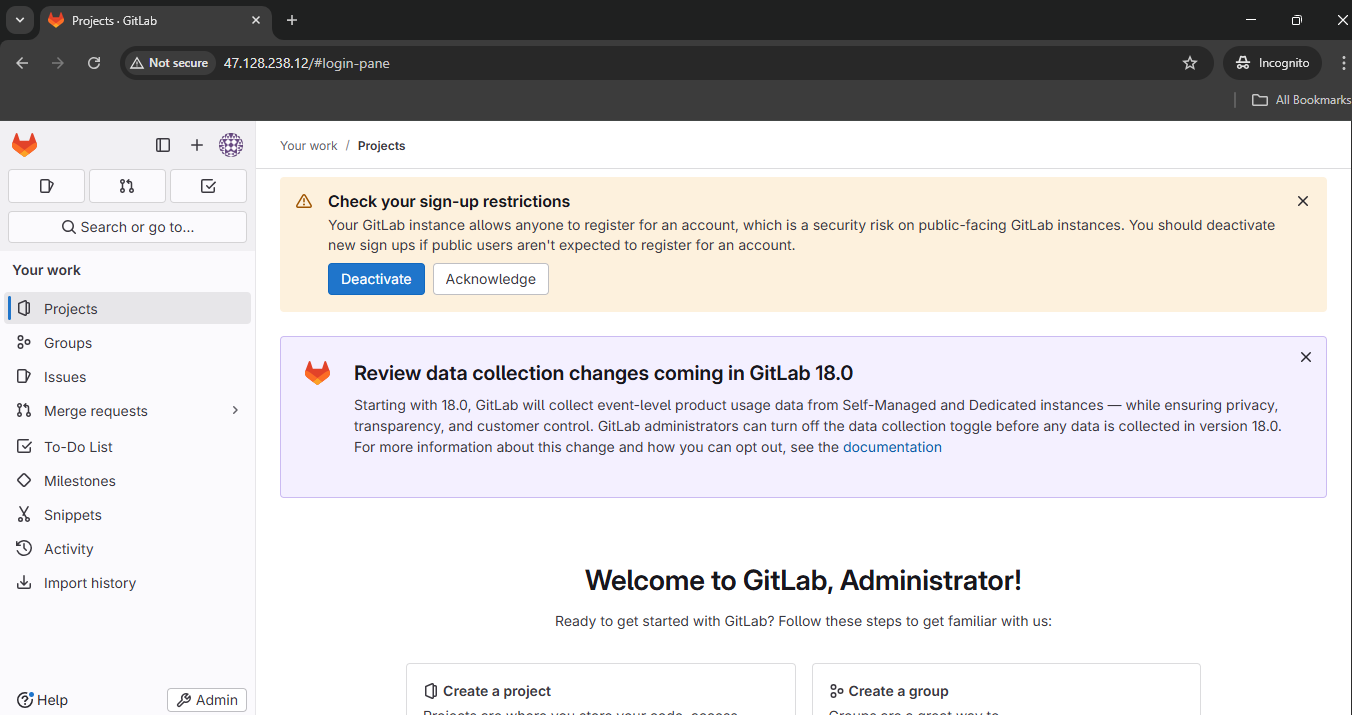
Now, we can access it from the web browser using the url <http://ipaddress:80>.

The initial password will be in the initial\_root\_

Password in the described volume.

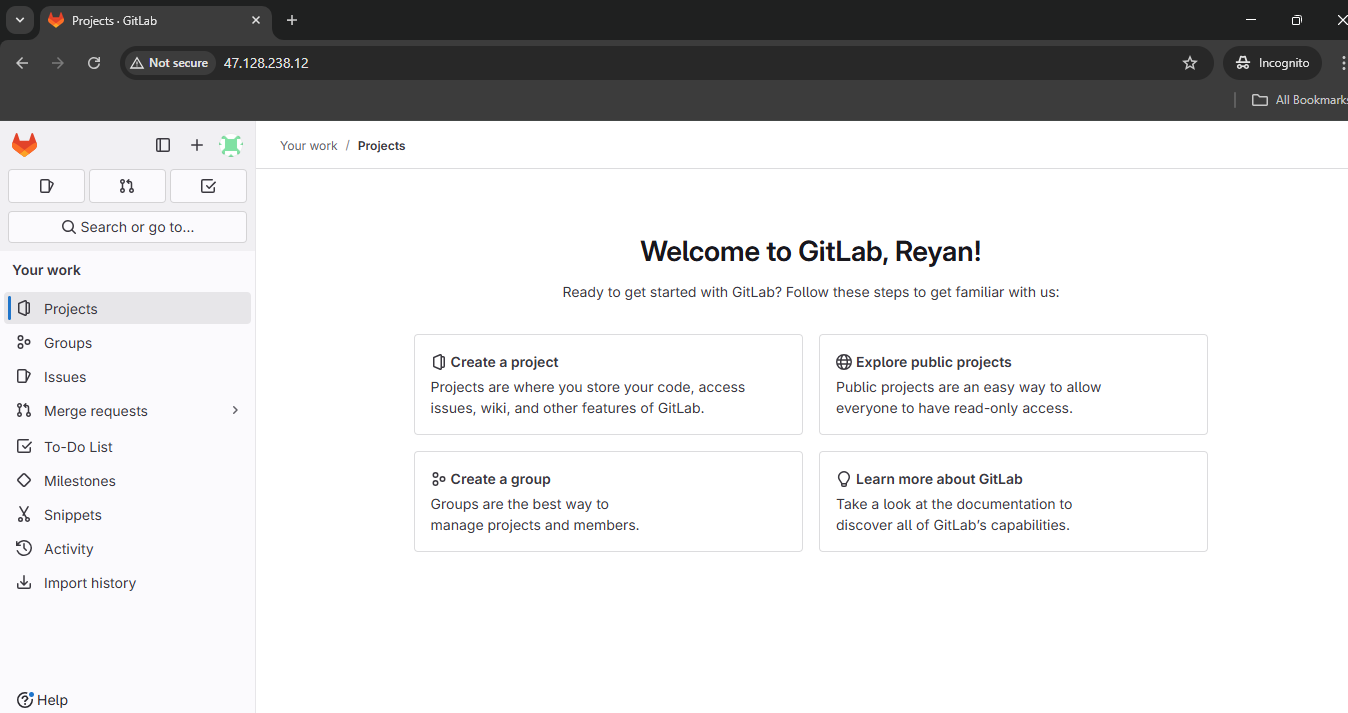


Login using the username “root” and password.



Users can now login and be approved by the administrators.

I added a user ReyaneBaiju and approved it.

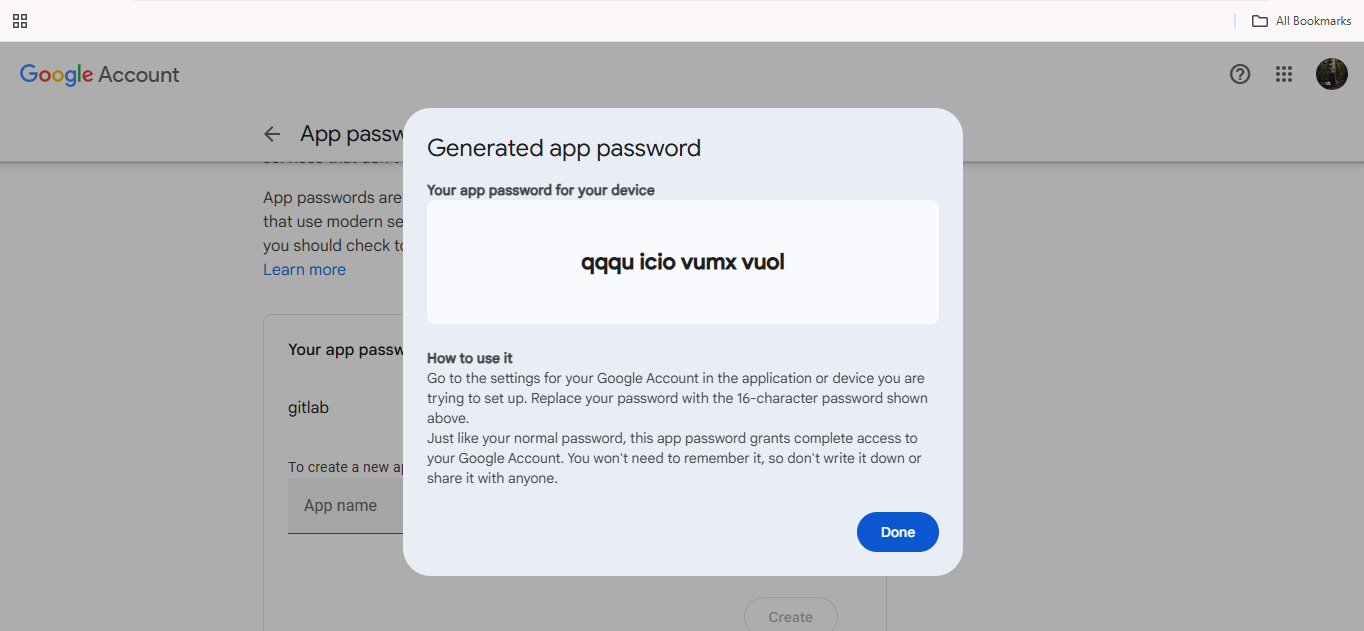


Setting up email alerts-

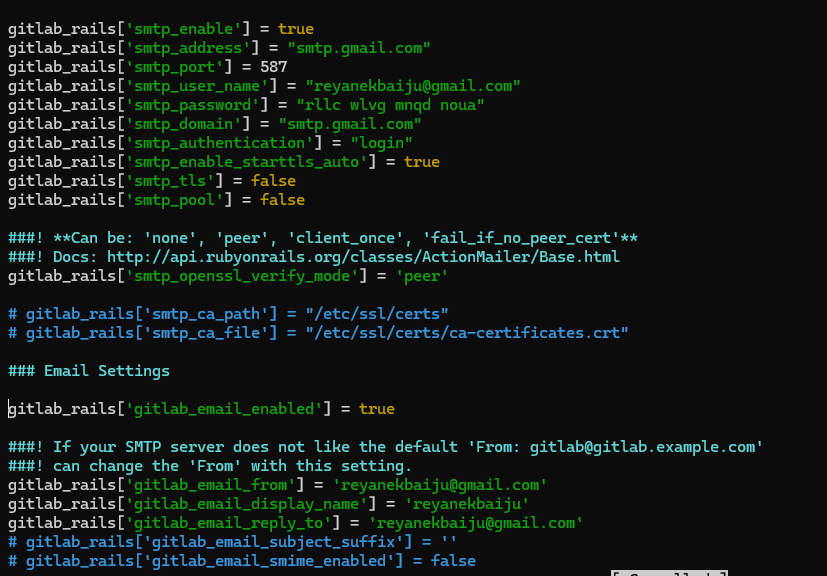
Reference- <https://docs.gitlab.com/omnibus/settings/smtp/#gmail>

To set up email, we need to configure SMTP using gmail.

First, get your gmail app password.



Then, we have to edit the /home/ubuntu/gitlab/config and edit the gitlab.rb file.

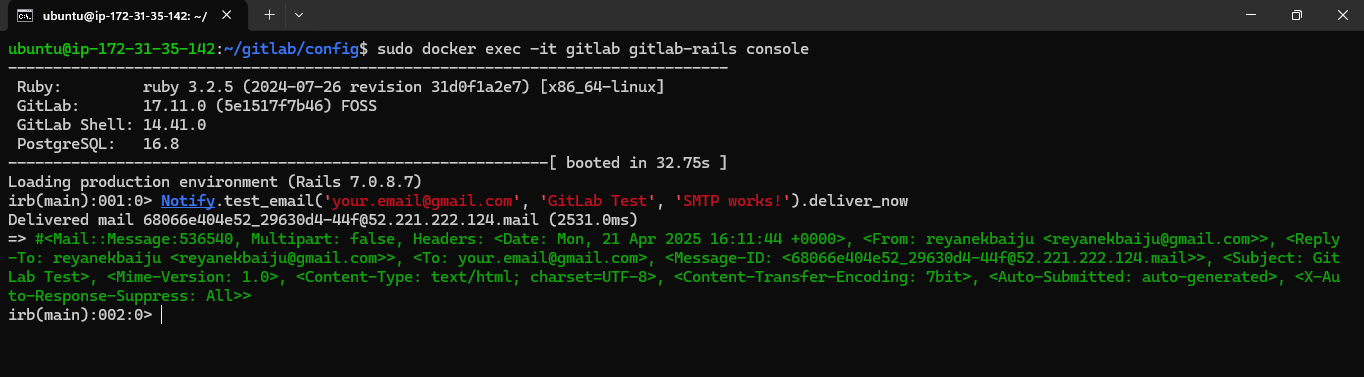


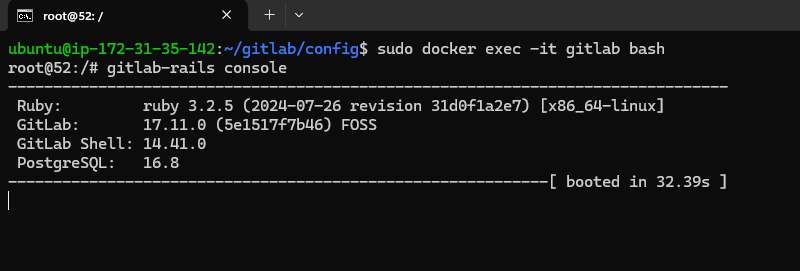
Now save and restart gitlab using both-

1. sudo docker exec -it gitlab gitlab-ctl reconfigure
2. sudo docker restart gitlab

Sending Test Email-

To send test email, we have to get into the gitlab ruby rails console, either use “sudo docker exec -it gitlab gitlab-rails console” or “sudo docker exec -it gitlab bash”





and use the send mail command-

“Notify.test\_email('destination\_email@address.com', 'Message Subject', 'Message Body').deliver\_now”

We will get the email in our mail-



Extra info- to use git clone on self hosted gitlab servers, remember this command-

ssh://git@gitlab.example.com/user/project.git

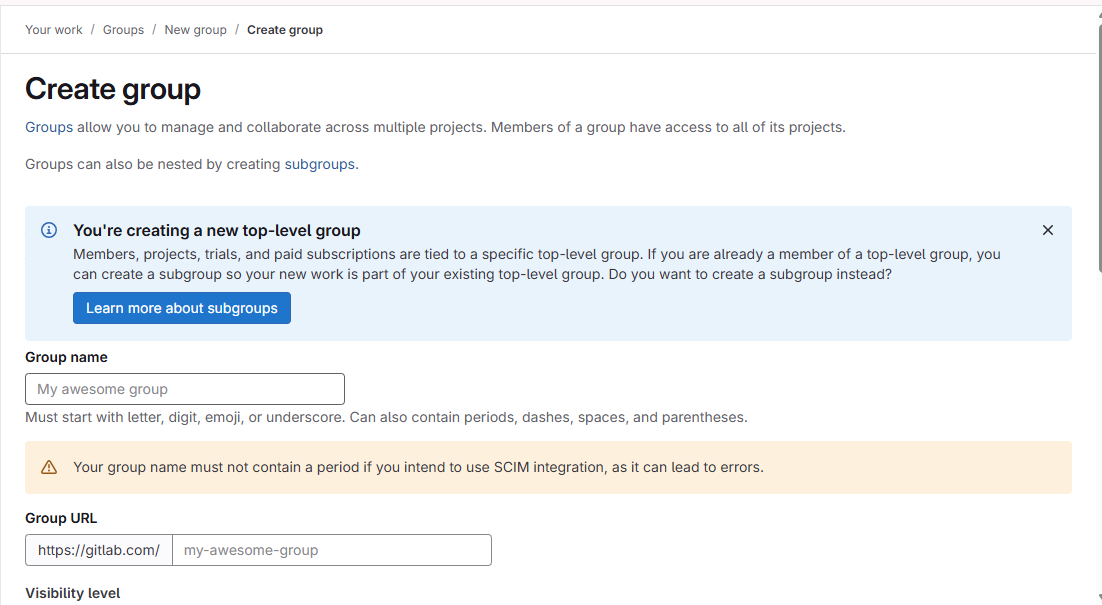
By default, runners are not available with self-hosted servers and won't be able to run CI/CD jobs. You can add your own managed runners by registering it with GitLab.

Important GitLab Terminologies

1. Gitlab Groups-

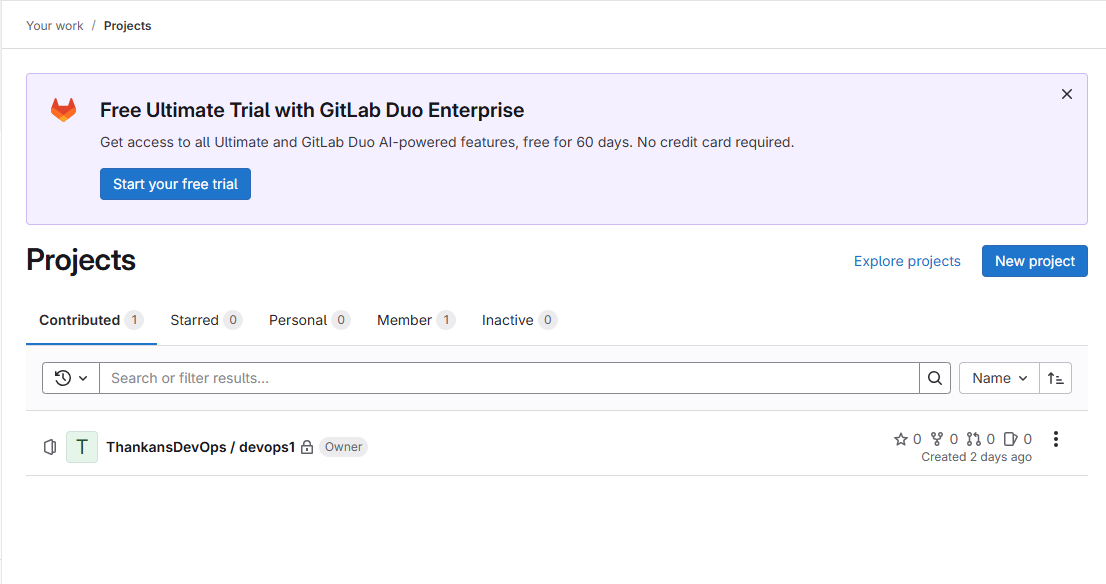
Gitlab groups are a number of different projects and repositories that are related together.

They can be used to manage these projects and its users.



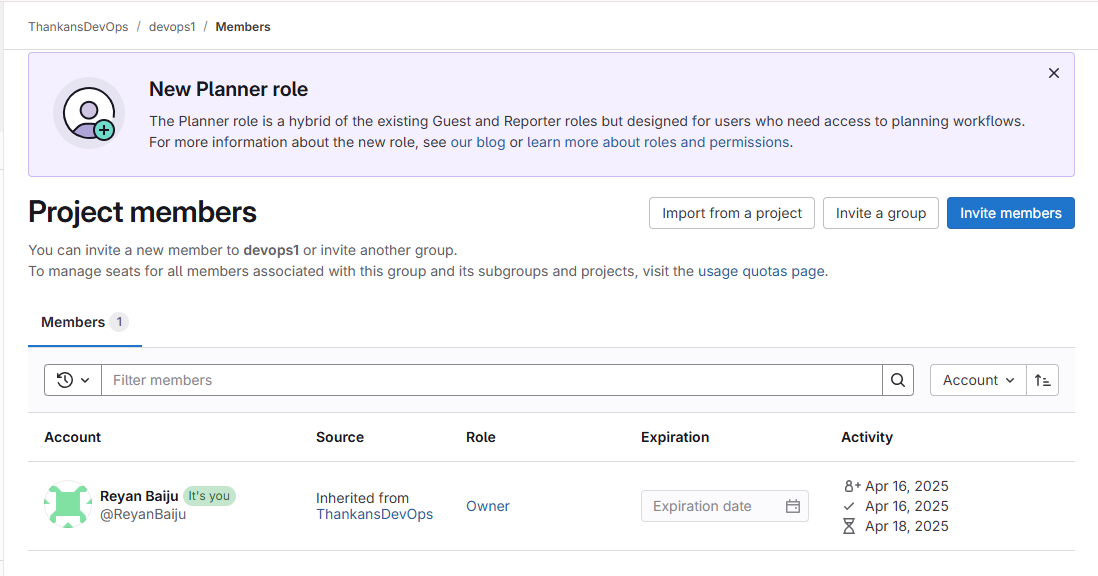
1. Gitlab project-

A gitlab project is a hub that serves as a complete environment where your team can store code, do CI/CD, check for issues etc. lt is like a github repository.



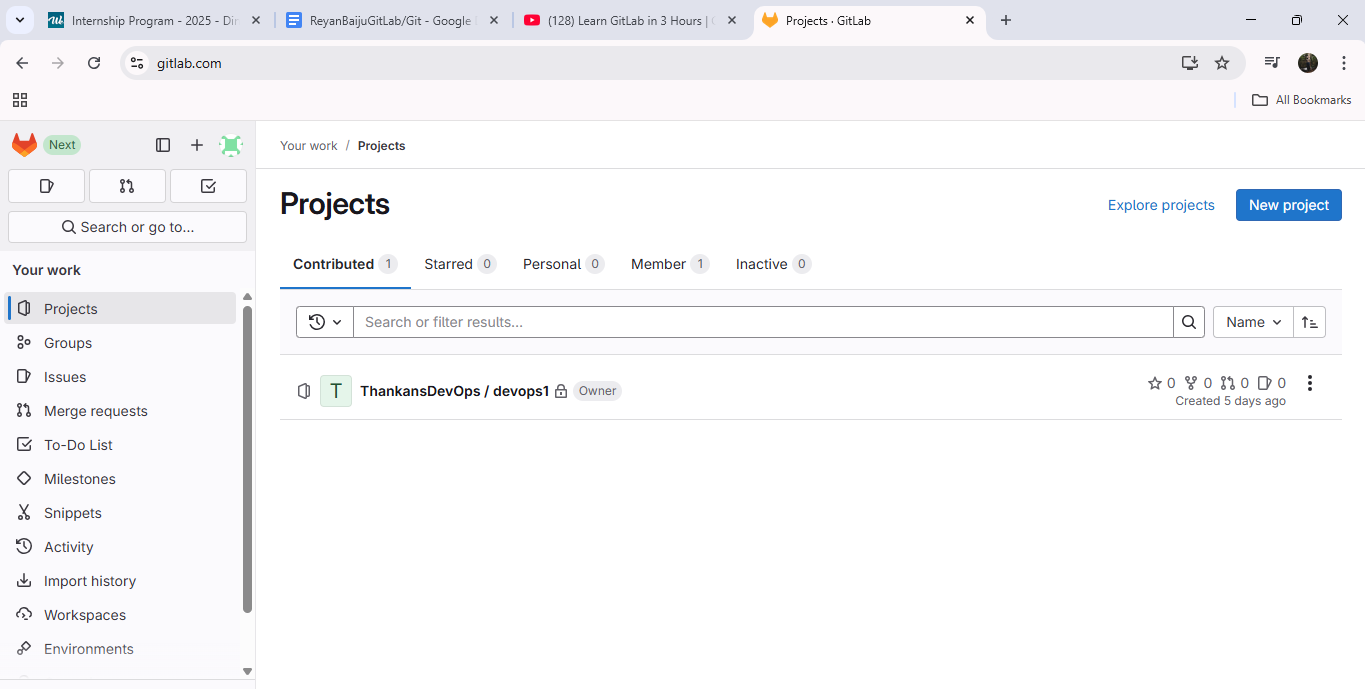
1. Gitlab Members

Gitlab members are users that have access to a gitlab project, we can specify the roles and what kind of access is allowed for the member.

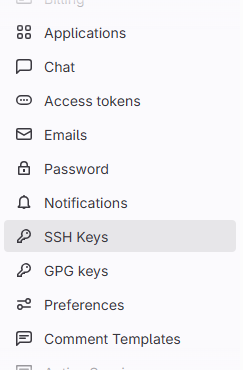


USING GITLAB-

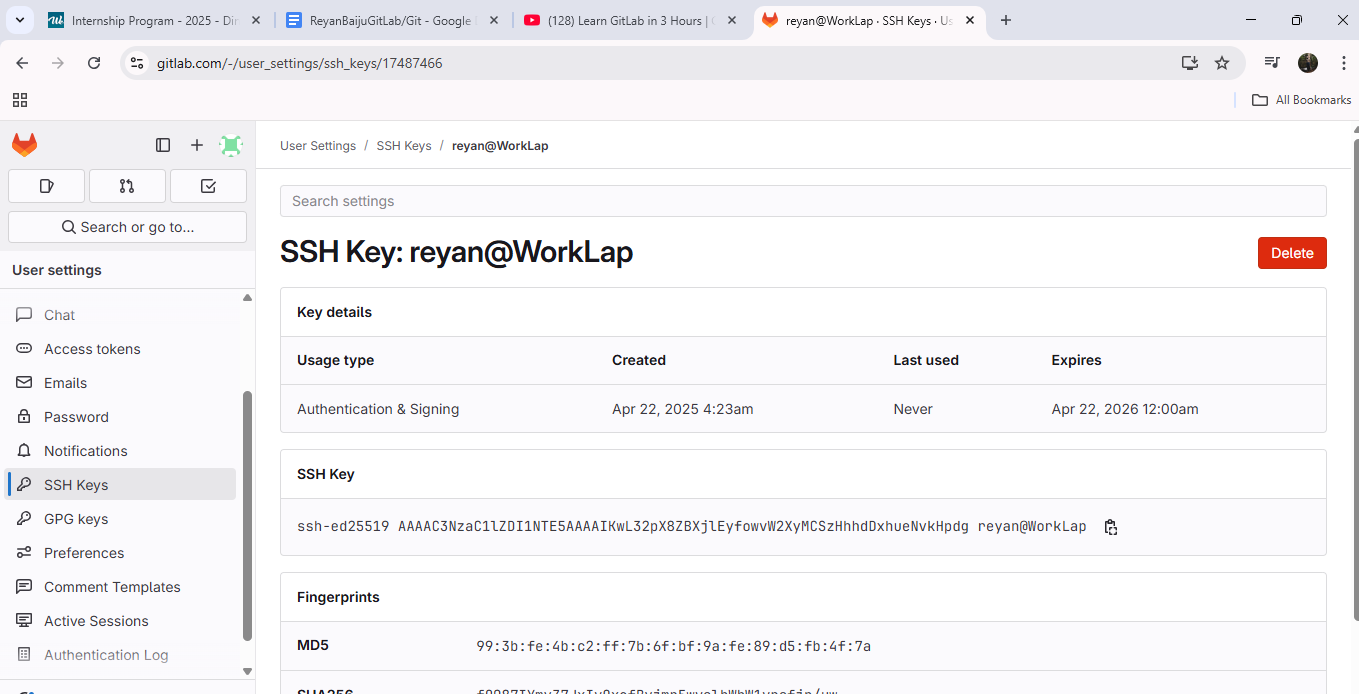
The default homepage of gitlab is-



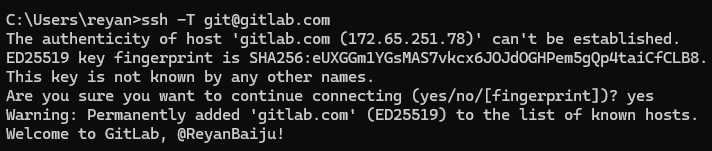
You can go to settings to set your preferred authentication method. There are personal access tokens and also ssh keys.



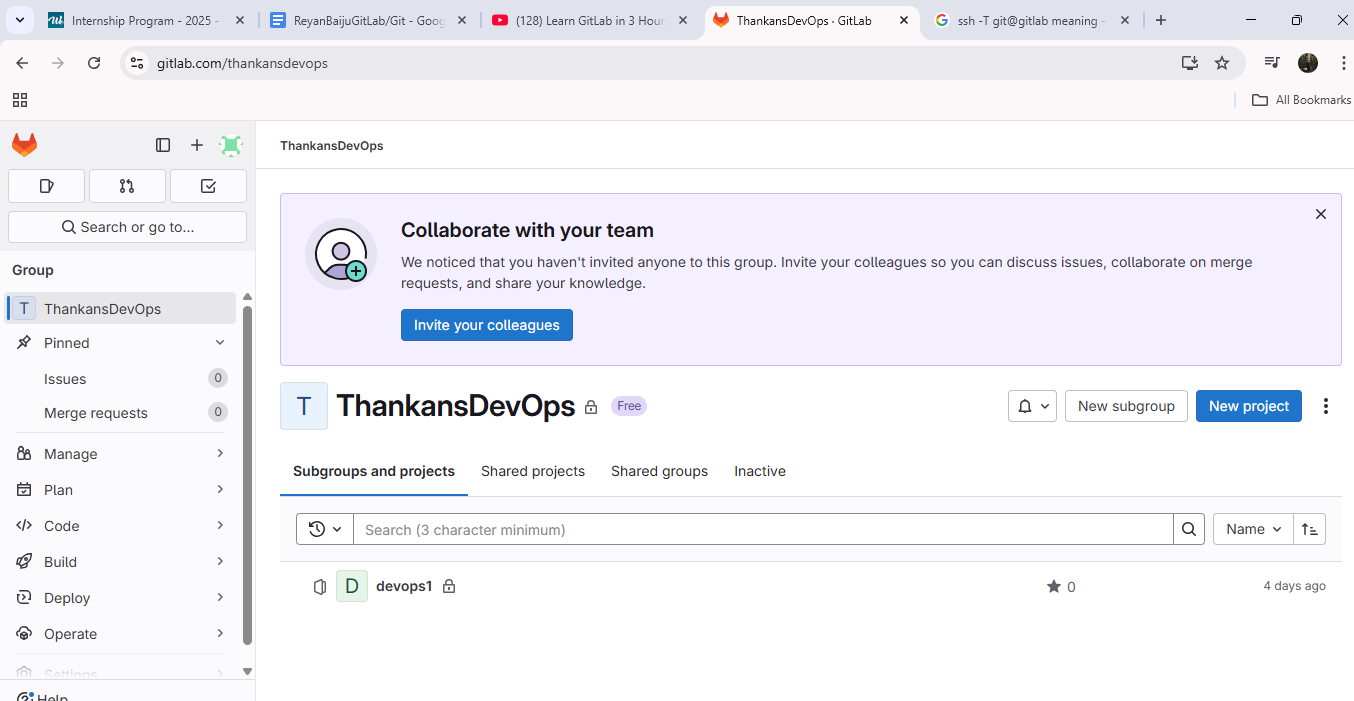
I have added SSH key to my account.



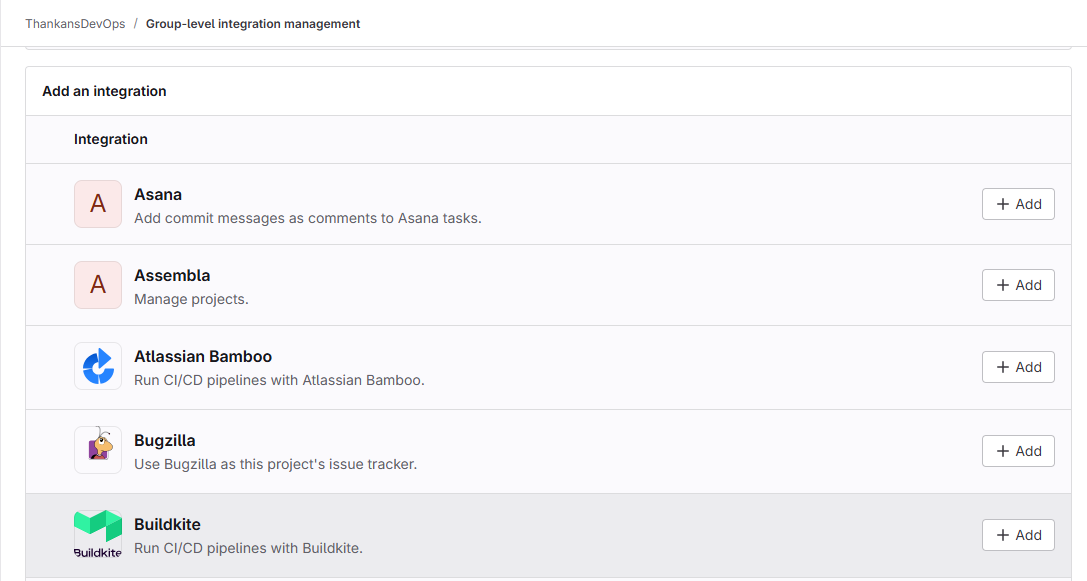
To check if it worked, use the command “ssh -T [git@gitlab.com](mailto:git@gitlab.com)”



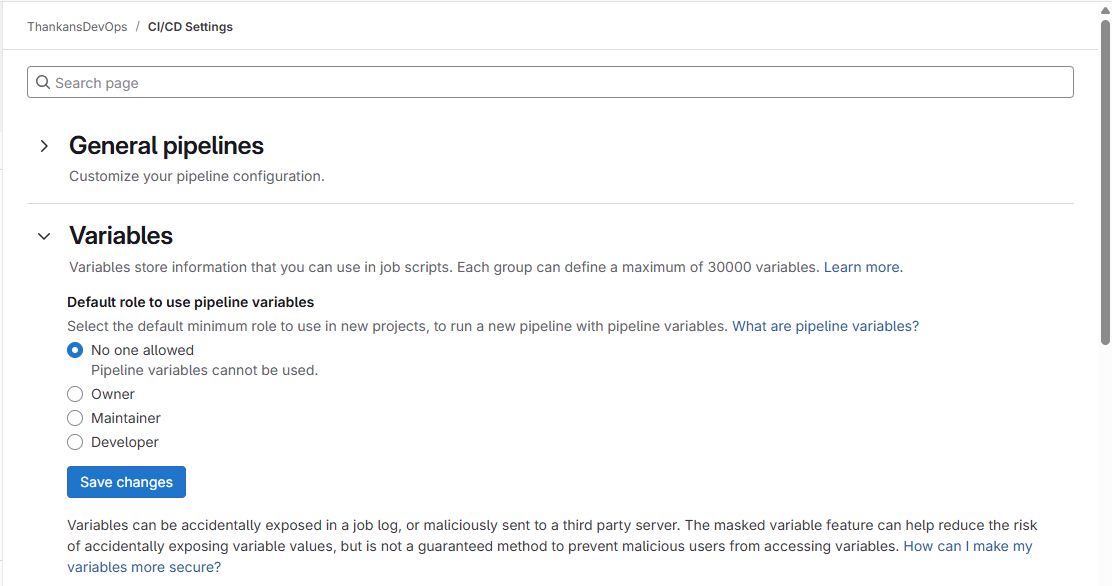
You can create Groups and integrate various tools like jira, atlassian bamboo etc.



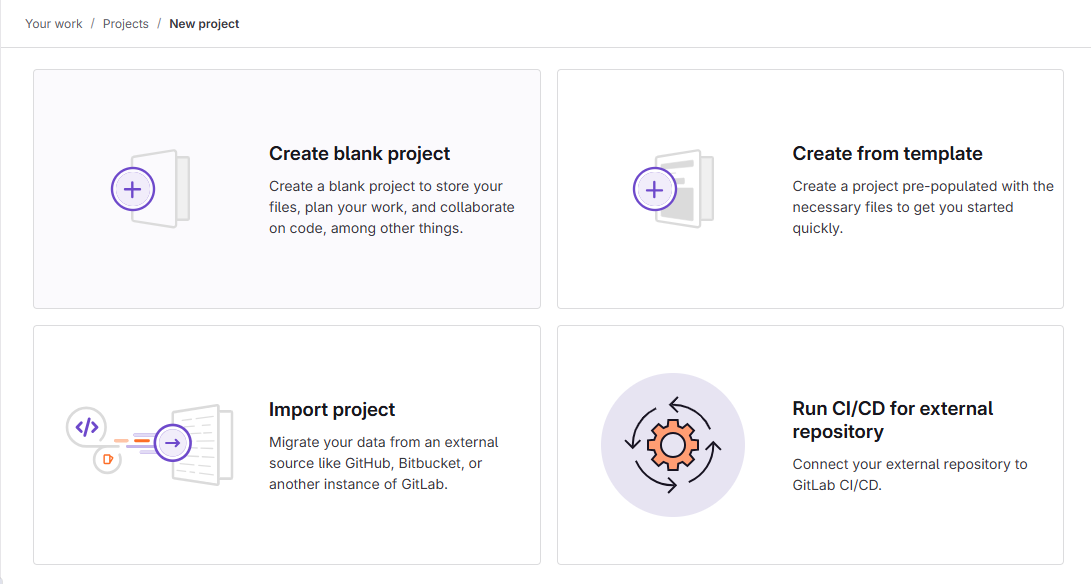
Integrations page-



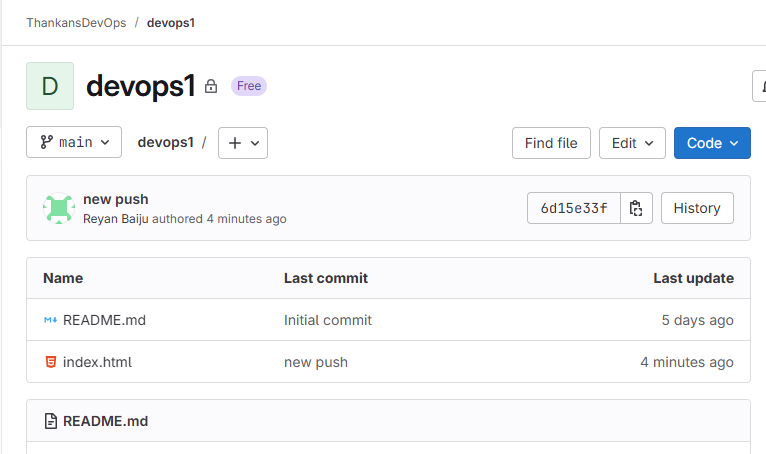
You can also added group level CI/CD variables-



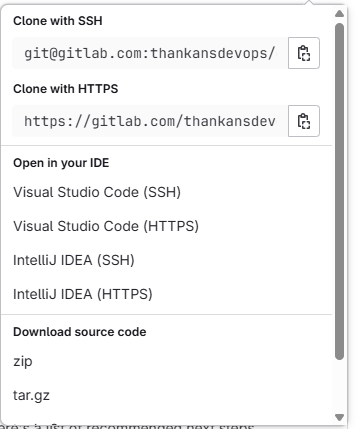
TIP- WE CAN USE GITLAB LIKE A CI/CD AGENT LIKE JENKINS WITH AN EXTERNAL REPOSITORY



After creating a project, you can use the repository. I created few files-

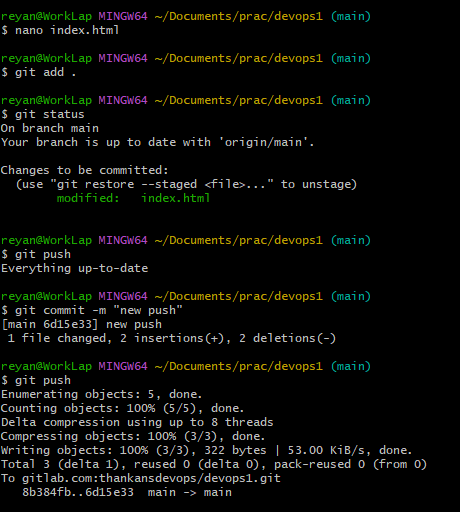


Im cloning it using http and adding it in my git bash.

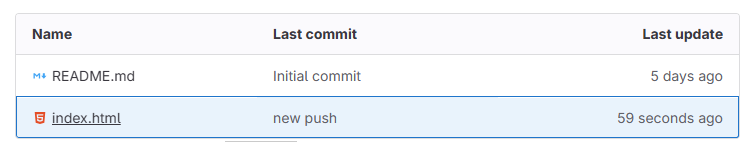




Editing and pushing the file.

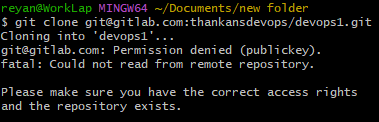


The commit is successful-



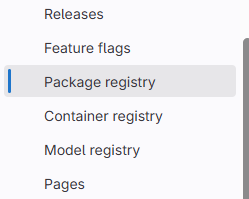
Imp- IF IT IS PRIVATE REPO AND YOU USE SSH KEYS TO AUTHENTICATE, REMEMBER TO PLACE THE PRIVATE KEY IN THE .SSH FOLDER.

ERROR-

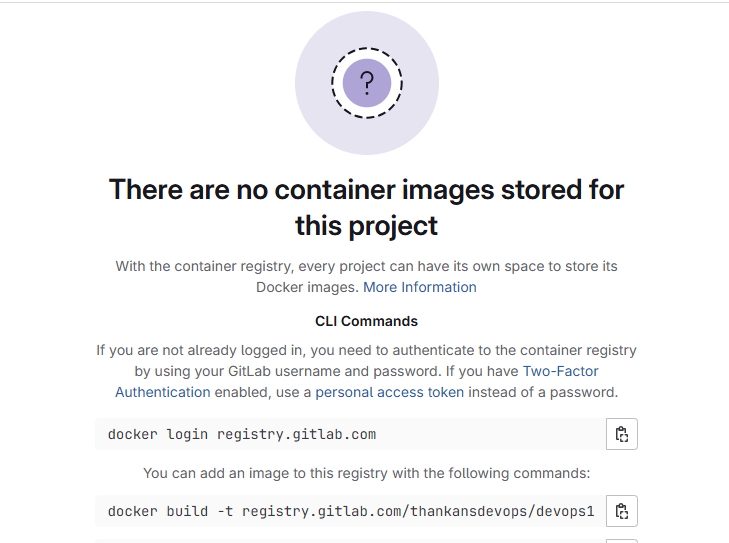


OTHER CAPABILITIES-

You also have container and package registry-

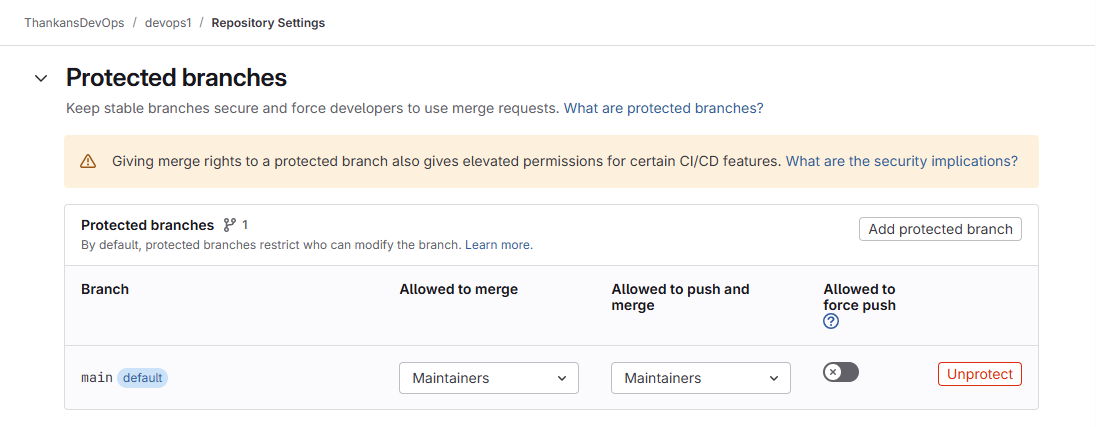


Container registry-



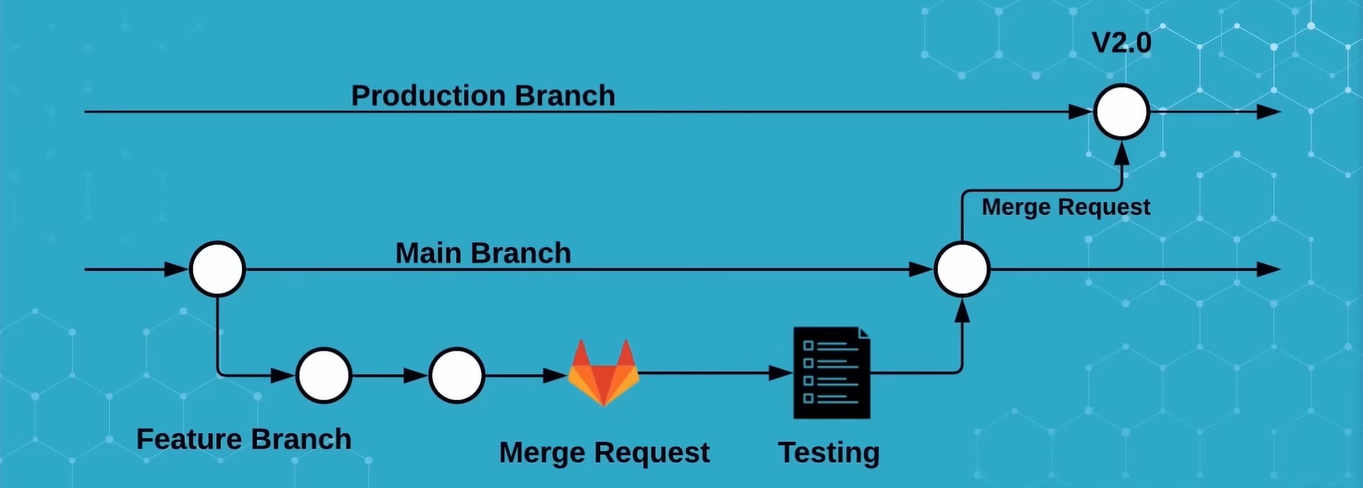
Protected branches-

Usually reserved for production branch-



GITLAB FLOW-

This is the most common gitlab flow, where a production branch is only updated after a feature branch is worked on.



GITLAB CI/CD

Important terminology-

Gitlab runners are open source programs that run on your vm, cloud etc that execute instructions in your ci/cd.

To install gitlab runner in a system, use the commands-

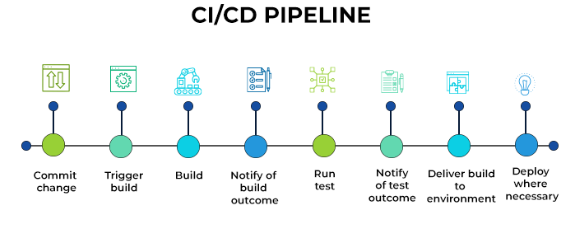
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash

sudo apt install gitlab-runner

After installing a runner in your machine, you can register it in gitlab. The details are in this documentation site- <https://docs.gitlab.com/runner/register/>

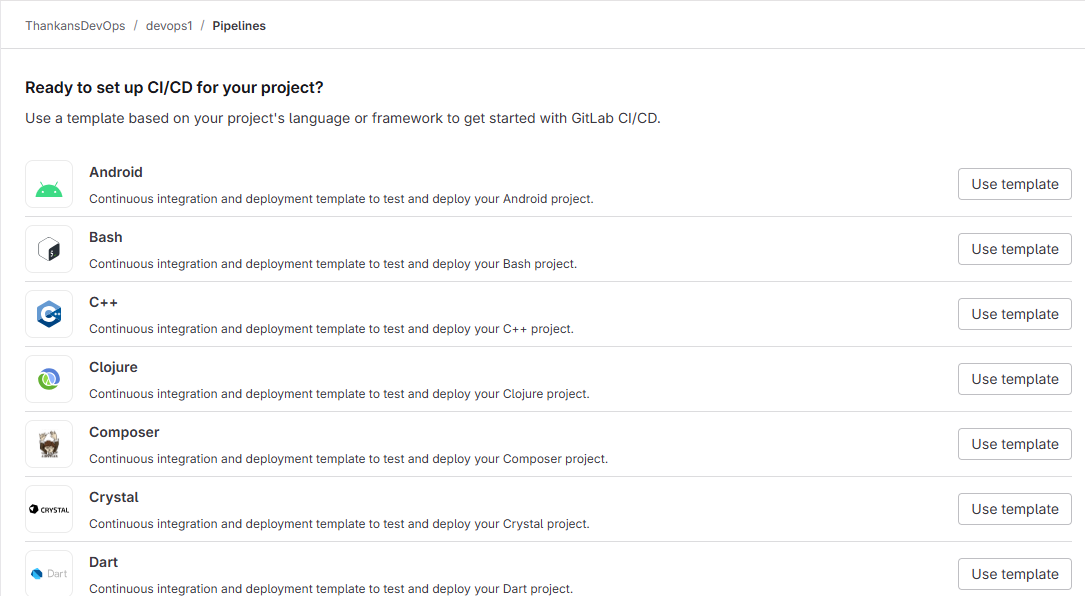
SETTING UP CI/CD

Basic CI/CD steps-



To carry out a CI/CD job, you have to define a .gitlab-ci.yml file in the root directory.

You can either manually create a .yml file for ci/cd or go to the dedicated gitlab ci/cd pipeline section and select.



This is a sample CI/CD configuration file-

stages: *# List of stages for jobs, and their order of execution*

- build

- test

- deploy

build-job: *# This job runs in the build stage, which runs first.*

stage: build

script:

- echo "Compiling the code..."

- echo "Compile complete."

unit-test-job: *# This job runs in the test stage.*

stage: test *# It only starts when the job in the build stage completes successfully.*

script:

- echo "Running unit tests... This will take about 60 seconds."

- sleep 60

- echo "Code coverage is 90%"

lint-test-job: *# This job also runs in the test stage.*

stage: test *# It can run at the same time as unit-test-job (in parallel).*

script:

- echo "Linting code... This will take about 10 seconds."

- sleep 10

- echo "No lint issues found."

deploy-job: *# This job runs in the deploy stage.*

stage: deploy *# It only runs when \*both\* jobs in the test stage complete successfully.*

environment: production

script:

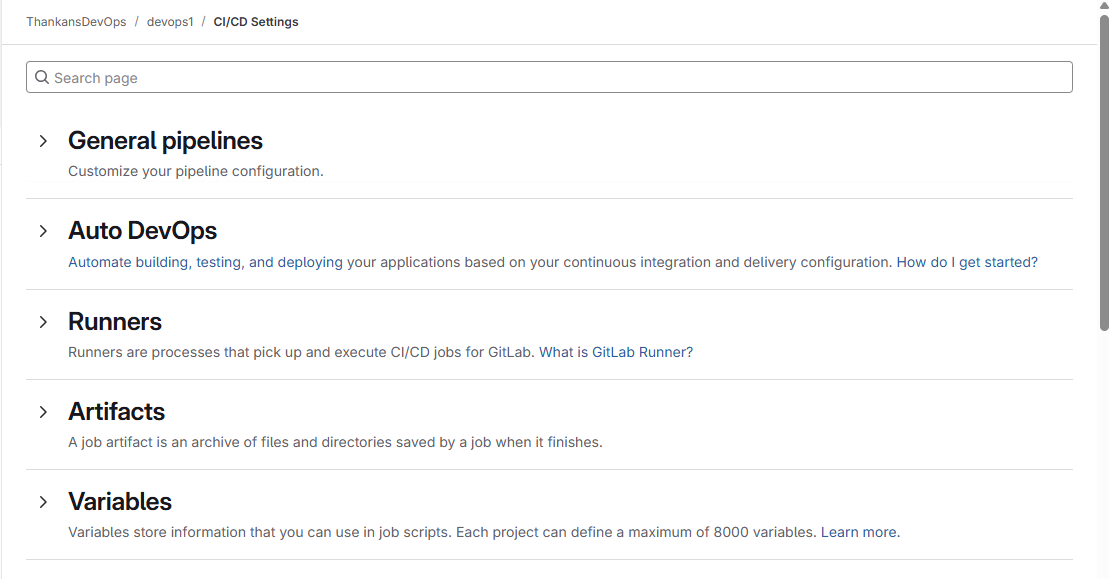
- echo "Deploying application..."

- echo "Application successfully deployed."

Example Docker CI/CD-



You can access CI/CD settings and configure it-



Imp- Don't hard code your credentials into the CI/CD configuration, use variables of the protected kind.

In the CI/CD code, this part of the code is responsible for checking if any new code is pushed to the branch to trigger the pipeline.

rules:

- if: $CI\_COMMIT\_BRANCH

exists:

- Dockerfile

To learn about creating docker CI/CD, go to this link- [**https://www.youtube.com/watch?v=qP8kir2GUgo**](https://www.youtube.com/watch?v=qP8kir2GUgo)

GIT

Reference- <https://git-scm.com/docs> <https://www.atlassian.com/git/tutorials/setting-up-a-repository>

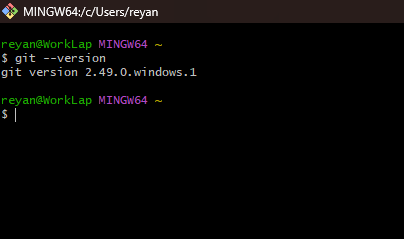
Git is a distributed version control system (VCS) used to track changes in files, allowing developers to collaborate efficiently and manage different versions of code.

A VCS tracks and records changes to any file (or a group of files) allowing you to recall specific iterations later on or as needed. VCSs are sometimes called source code management (SCM) or revision control systems (RCS).

Version control allows numerous team members to work collaboratively on a project, even if they’re not in the same room or even country.

To install GIT, go to the <https://git-scm.com/downloads> page, and select your os.

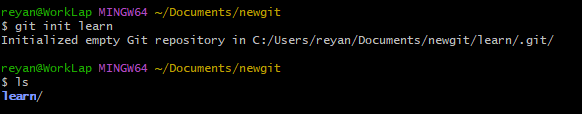
After installing GIT, you can open the GIT Bash terminal and start using GIT.



Most important GIT commands-

1. **Git init**- initialize current directory as the git repository.

Git init <repo-name>- Create a subdirectory in the same directory that is initialized as the git repository.



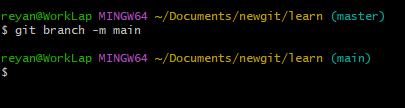
1. **Git config**- Git config is used to set configuration values of Git.

The main 3 flags that are used are- –local (only configured for current repository), –global (configured for user in that OS only),

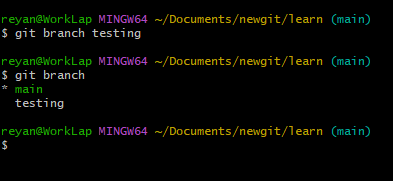
Ex-

1. Git config –global init.defaultBranch main- Used to set the default branch as main.
2. git config --global user.email "reyanekbaiju@gmail.com"
3. Git config --global user.name "reyane"

3. **Git Branch**- Used to create a new branch



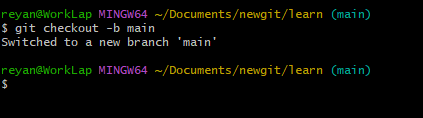
We can see all the branches by typing git branch.



To delete a branch, use the command git branch -d

4. **Git checkout**- Used to switch to a branch.

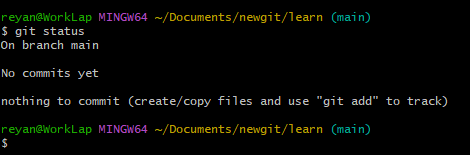
Ex- git checkout -b <branchname> to create a new branch and switch to that.



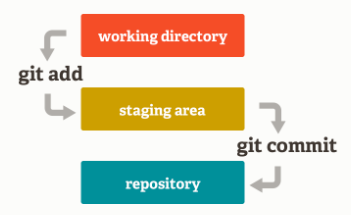


IMP- When we switch to a new branch, the changes that we make in that branch stay in that branch only. To merge the changes to the main branch, we can use the git merge command.

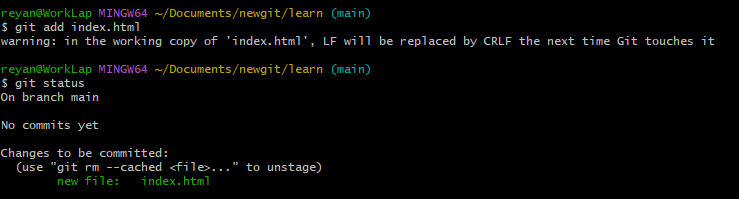
5. **Git status**- Used to check the current status of the current directory. Shows all the changes in all the files in the directory.



6. **Git add**- Git has 2 stage commit to track changes in files. We have to add files from the working directory to the staging area.



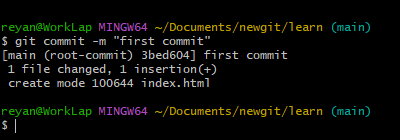
We can use   
git add <files/directories>” to add files to the staging area.



To unstage files- use the command- "git rm --cached <file>”

7. **Git commit**- Used to commit changes that we are making. If we just type git commit, it will bring up the editor to type in a commit message.

We can use the git commit -m “message here”.

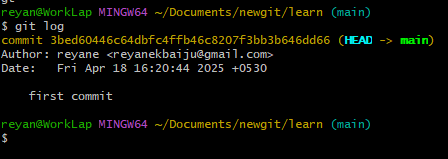


When committing for the first time, GIT will ask for user metadata, like this-

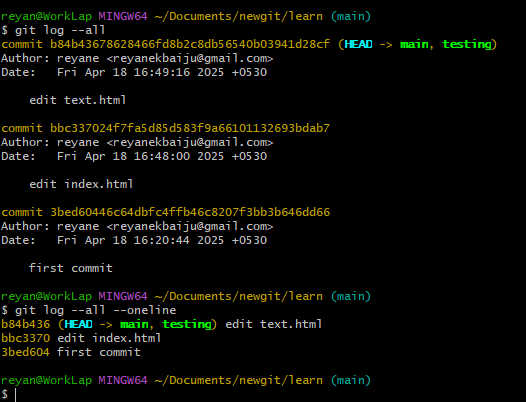
git config --global user.email "reyanekbaiju@gmail.com"

git config --global user.name "reyane"

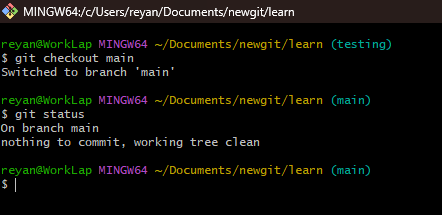
8. **Git log**- Shows history of git commits-

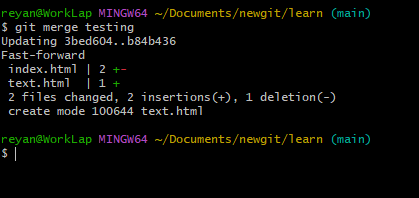


Use the flag –all to see changes to all branches, –oneline to show one line summary.

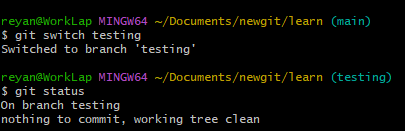


9. **Git merge**- To use git merge, we must change to the target branch(master or main), and then use git merge <source branch>

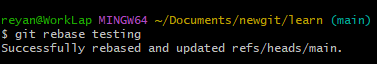




10. **Git switch**- To switch to a new branch. Basically, a newer version of checkout.



11. **Git rebase**- Like merge, used to merge source branch to target branch. Rebase will present conflicts one commit at a time, whereas merge will present them all at once. It’s easier to handle the conflicts, but reverting a rebase is more difficult than reverting a merge if there are many conflicts.



In rebase, Git takes the latest version of main, then re-applies all the changes from testing one by one on top of it.

So it’s like testing is rebuilt starting from main.

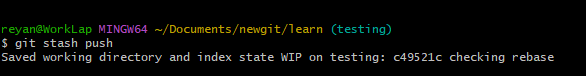
Always checkout to the testing branch before doing rebase.

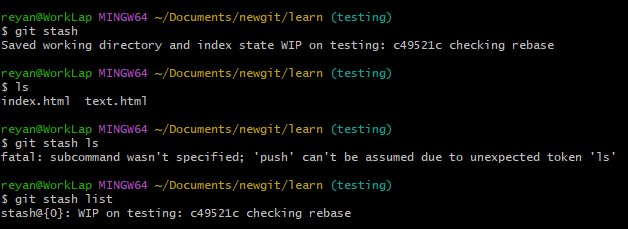
“git checkout testing”

“git rebase main”

12. **Git Stash**- Take all the changes and stash it outside of the git repository history.

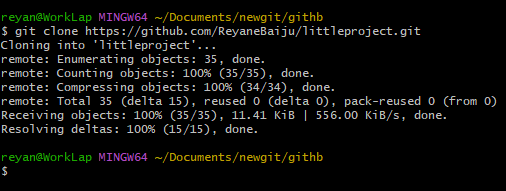
To add to the stash, use git stash push





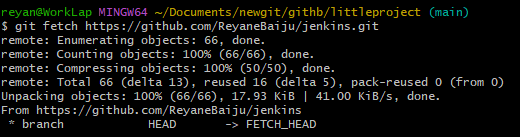
To bring the changes to the original branch, use git stash apply or git stash pop (pop removes the changes from the list).

13. **Git clone**- Git clone is used to clone a repository into your machine. Example of a git clone command looks like this- git clone <https://github.com/ReyaneBaiju/littleproject.git>

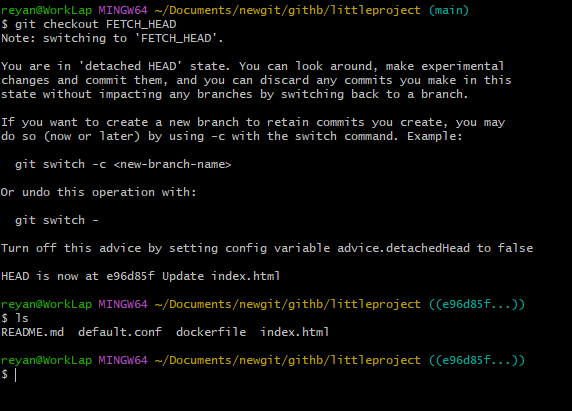


14. **Git fetch**- Git fetch can be used to download commits, files, and refs from a remote repository into your local repo. It can fetch branches and files from the same repo or different repo.

Different Repo example-

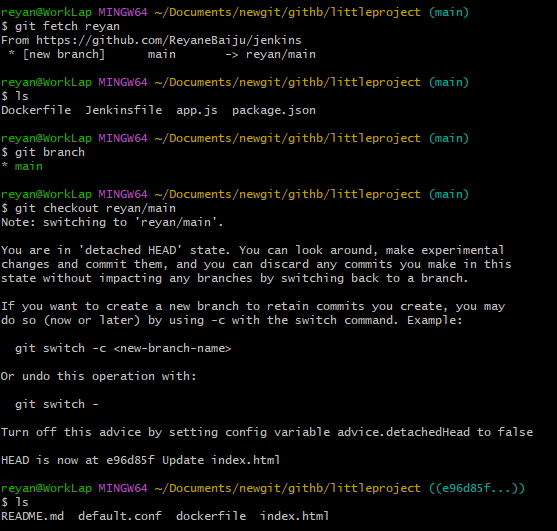


You can access the fetched files by using the command git checkout FETCH\_HEAD for this example.



SAME REPO-

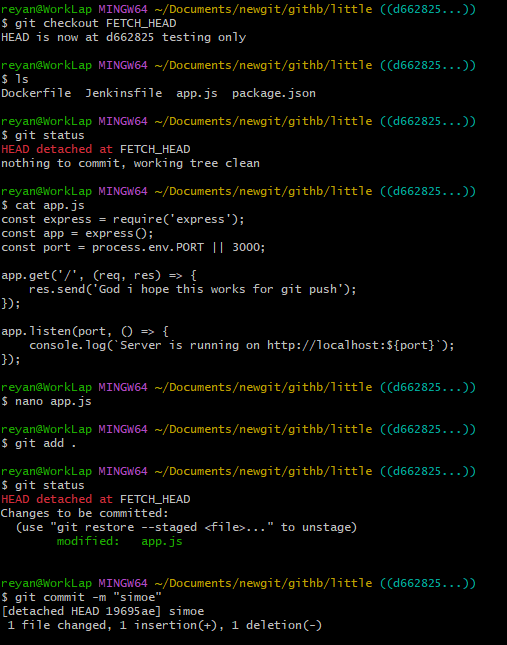
After you add your remote repository, you can fetch another branch.



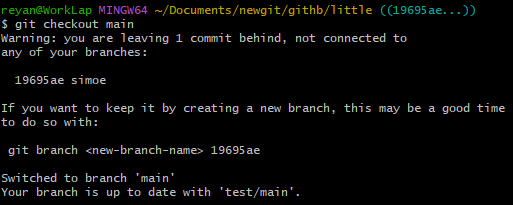
HOW TO USE GIT FETCH

First ,we can add our remote repository and fetch the branch that we want. It will be available in FETCH\_HEAD.

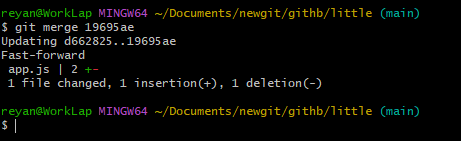
We can now switch or checkout to that branch and edit the files.



After adding and committing, we can go back to the main branch and merge it-

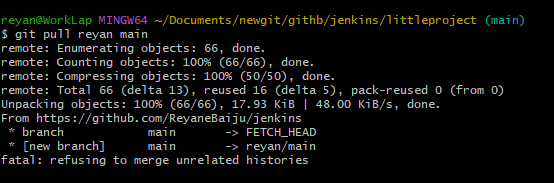


We will get the code- use it-

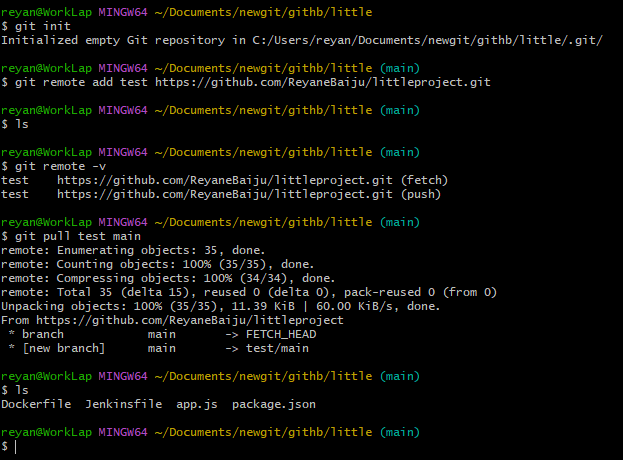


15. **Git Pull**- Git pull is a combination of the git fetch and git merge command.

The command example- git pull <remote> <branch>

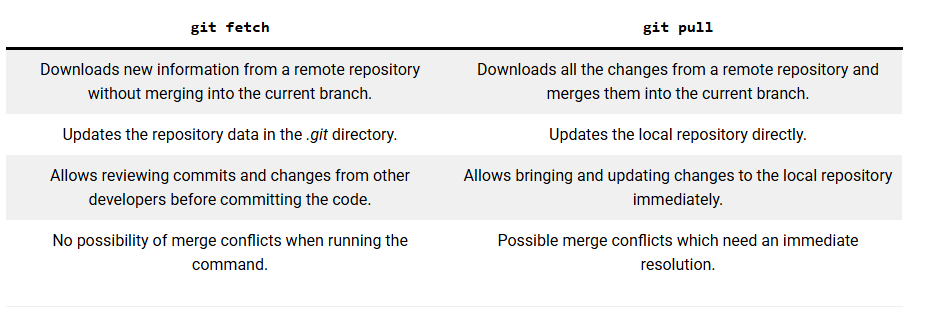


Git pull example-



Merging example-



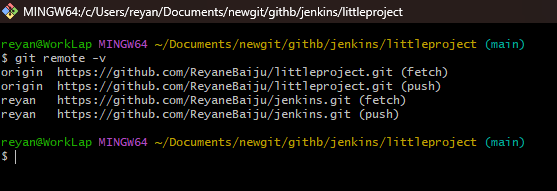


16. **Git Remote** - Git remote is used to manage remote repositories.

You can add a remote repository using the command git remote add <customname> <repourl>



You can see the added repositories using the git remote -v command.

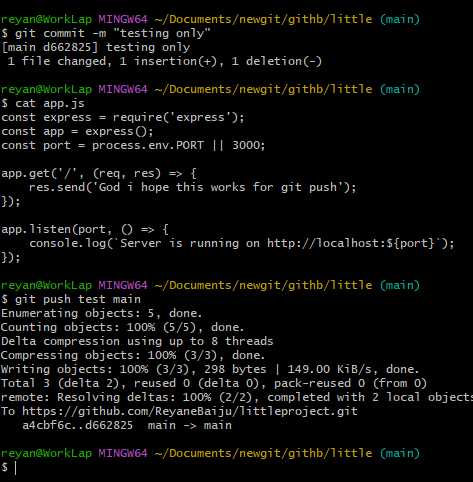


17. **Git Push**- Git push is used to push changes to the remote repository.

Command- git push <remote> <branch>.

You can use the –force tag to force a push to a branch.

You can push after editing, adding the file to the staging area and committing.





18. **Git Revert**

Git Revert is used to undo commits to a branch. The syntax of the git revert command is git revert <commithash>