



# **Hosting Our Own GITLAB Server (Group Project)**

*4/13/2023  
OPS300*

*Yash Siraj Devani  
Syed Mujahid Hamid Ali  
Dr. Peter Callaghan*

# Table of Contents

Hosting Our Own GITLAB Server (Group Project).....	1
Summary .....	3
Introduction.....	4
Creating and Deploying a GitLab Server .....	5
Securing the Private Server .....	7
Conclusion .....	13

## **Summary**

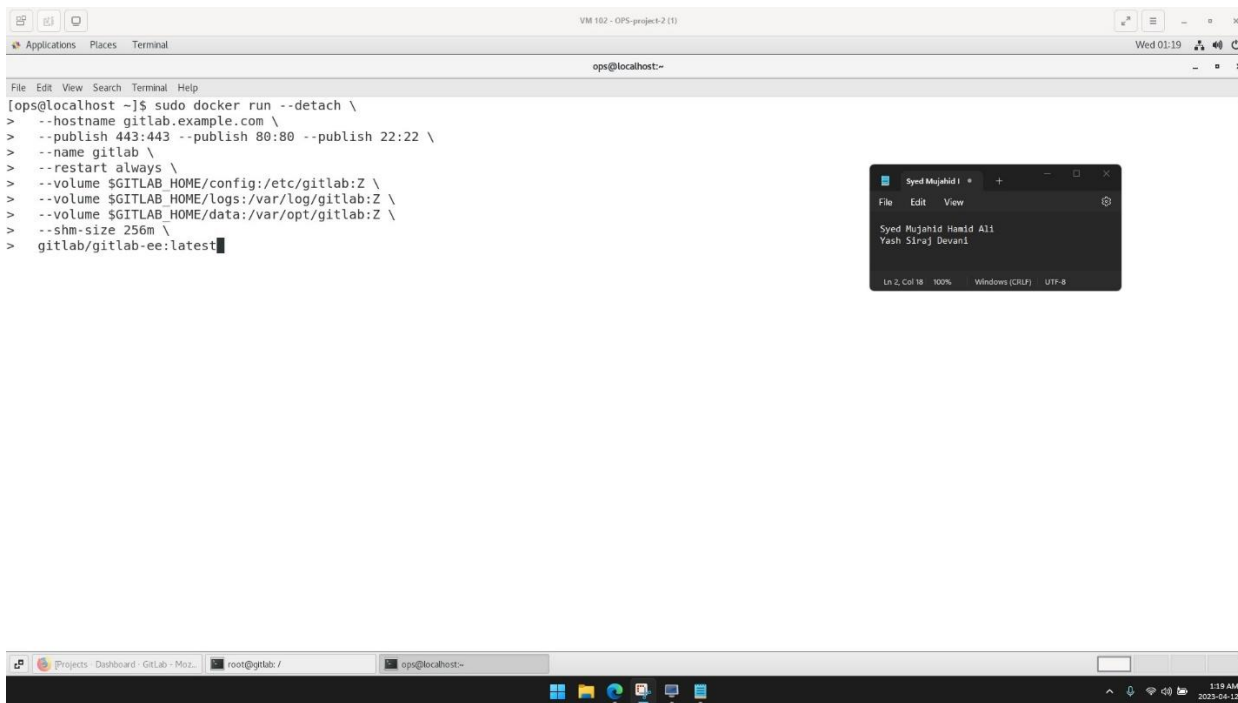
In the following project we will be hosting our own GitLab server. The main objective for us is to create and deploy the GIT server and finally we will be securing the server. Some of the tools we used for this project include Docker to pull the GitLab. Our main machine was running on CentOS. Some of the changes we made to secure GitLab include changing from http to https, enabling 2FA so that all the users are more secure and finally going through root when an account was to connect to the repository.

We created this repository for a private company and secured it so that only the authorized users can communicate with each other and share the code files.

## **Introduction**

GitLab is a web-based Git repository manager that provides a complete DevOps platform for software development teams. Like GitHub, GitLab provides a version control system (VCS) for managing code repositories, but it also includes a wide range of tools and features that support the entire software development lifecycle. These features include issue tracking, continuous integration, and delivery (CI/CD), code review, and project management tools. GitLab is designed to provide a comprehensive solution that enables teams to work more efficiently, collaborate more effectively, and streamline their workflows. With GitLab, teams can manage their entire development process in one place, from planning and coding to testing and deployment. GitLab is also open-source, meaning that developers can download and install it on their own servers, providing greater control over their code and data. GitLab has become a popular platform for both small and large development teams and is used by thousands of organizations worldwide.

# Creating and Deploying a GitLab Server

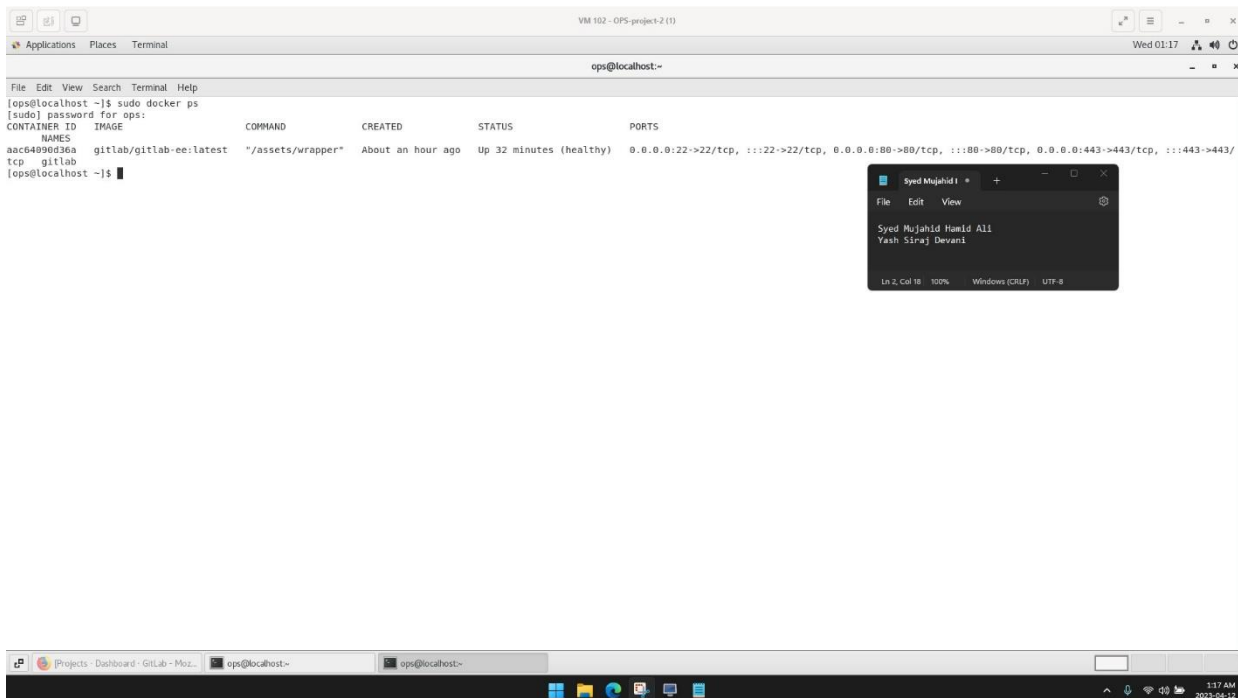


```
ops@localhost: ~]$ sudo docker run --detach \
> --hostname gitlab.example.com \
> --publish 443:443 --publish 80:80 --publish 22:22 \
> --name gitlab \
> --restart always \
> --volume $GITLAB_HOME/config:/etc/gitlab:Z \
> --volume $GITLAB_HOME/logs:/var/log/gitlab:Z \
> --volume $GITLAB_HOME/data:/var/opt/gitlab:Z \
> --shm-size 256m \
> gitlab/gitlab-ee:latest
```

This shows the command to create a GitLab container using docker. We are going to run the container on three ports:

1. SSH (22)
2. HTTP (80)
3. HTTPS (443)

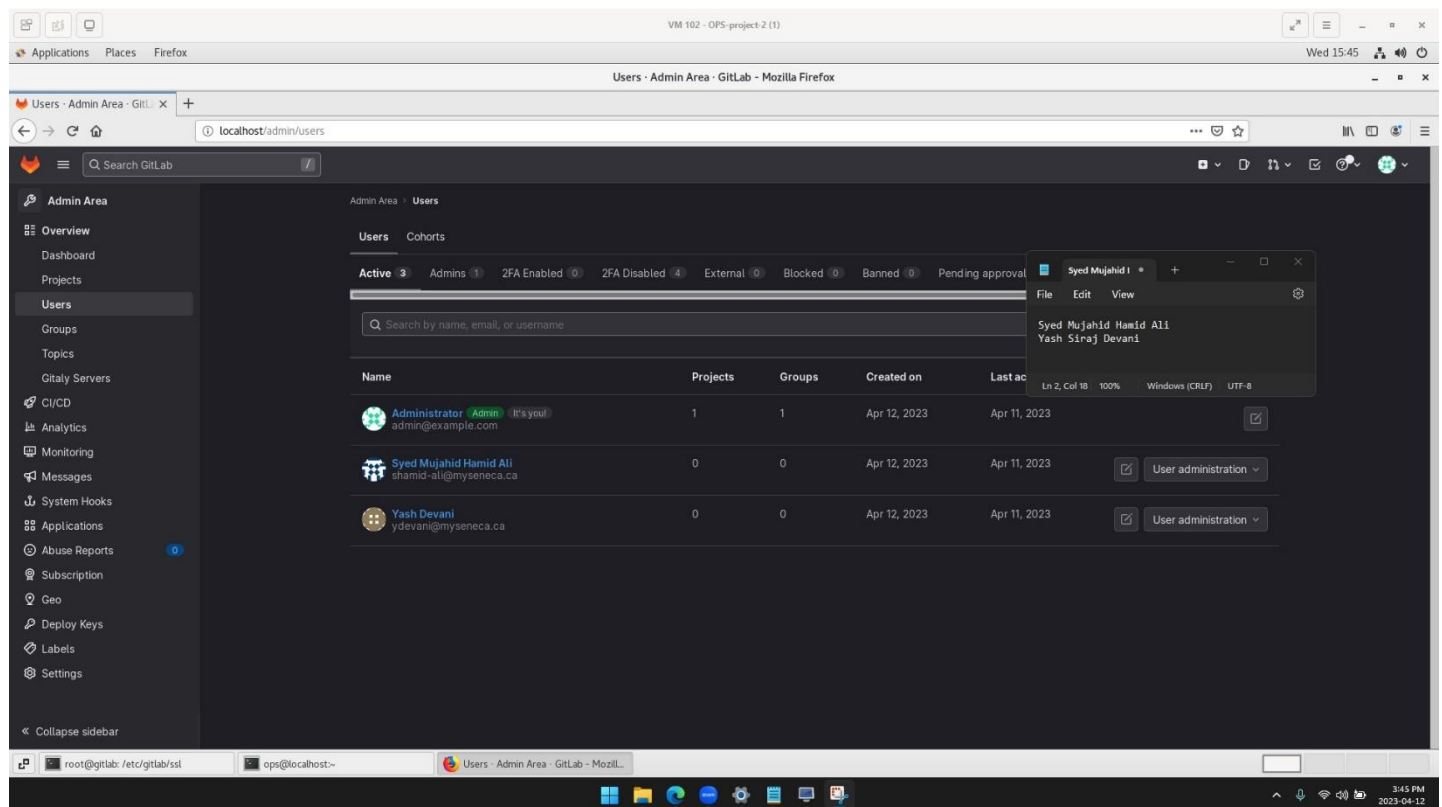
We attached separate volumes so that we can configure and log the processes for our privately created git server.



```
ops@localhost: ~]$ sudo docker ps
[sudo] password for ops:
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
aac64899d36a   gitlab/gitlab-ee:latest             "/assets/wrapper"       About an hour ago   Up 32 minutes (healthy)   0.0.0.0:22->22/tcp, :::22->22/tcp, 0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp
ops@localhost: ~]$
```

As we can see, a 'GitLab' container has been created and is running in the given ports. One issue that may arise would be running the container on ports which have some pre-existing service running on them. For us, it was openSSH running on port 22. For that, we simply disabled that service, but at an enterprise level, we would run this container or the service on some other secure port.

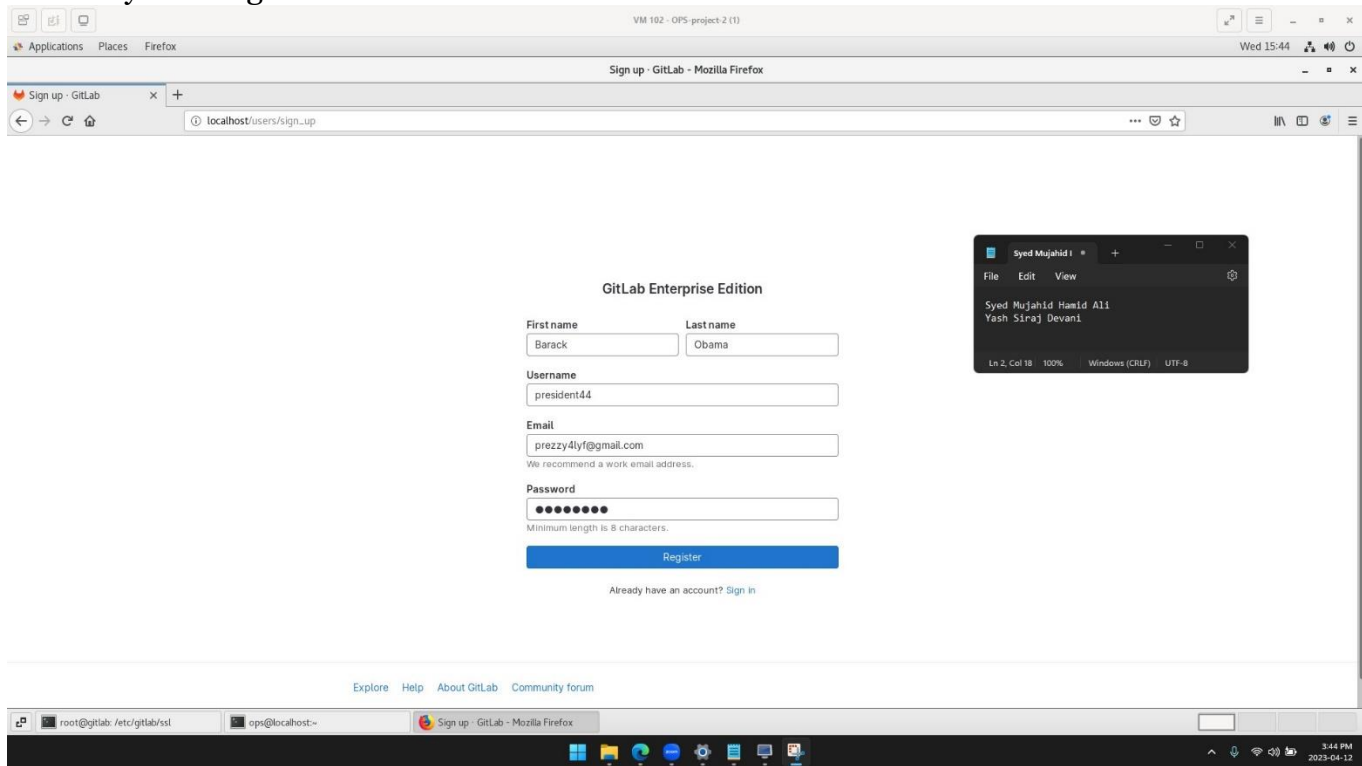
Then when we access localhost on the browser, we should be greeted with a GitLab signup/login page. Here, you can sign in as root and the password should be given in the log files that we mounted to the volume given while creating the container. Then you can log back out and create your own user. When you first sign up; you will be notified saying that the admin/root should allow you and add the user created to approved users.



For this project, we (Syed Mujahid Hamid Ali and Yash Siraj Devani) were the only authorized users and any other users created should have read permissions, unless they have been added to a project or a group.

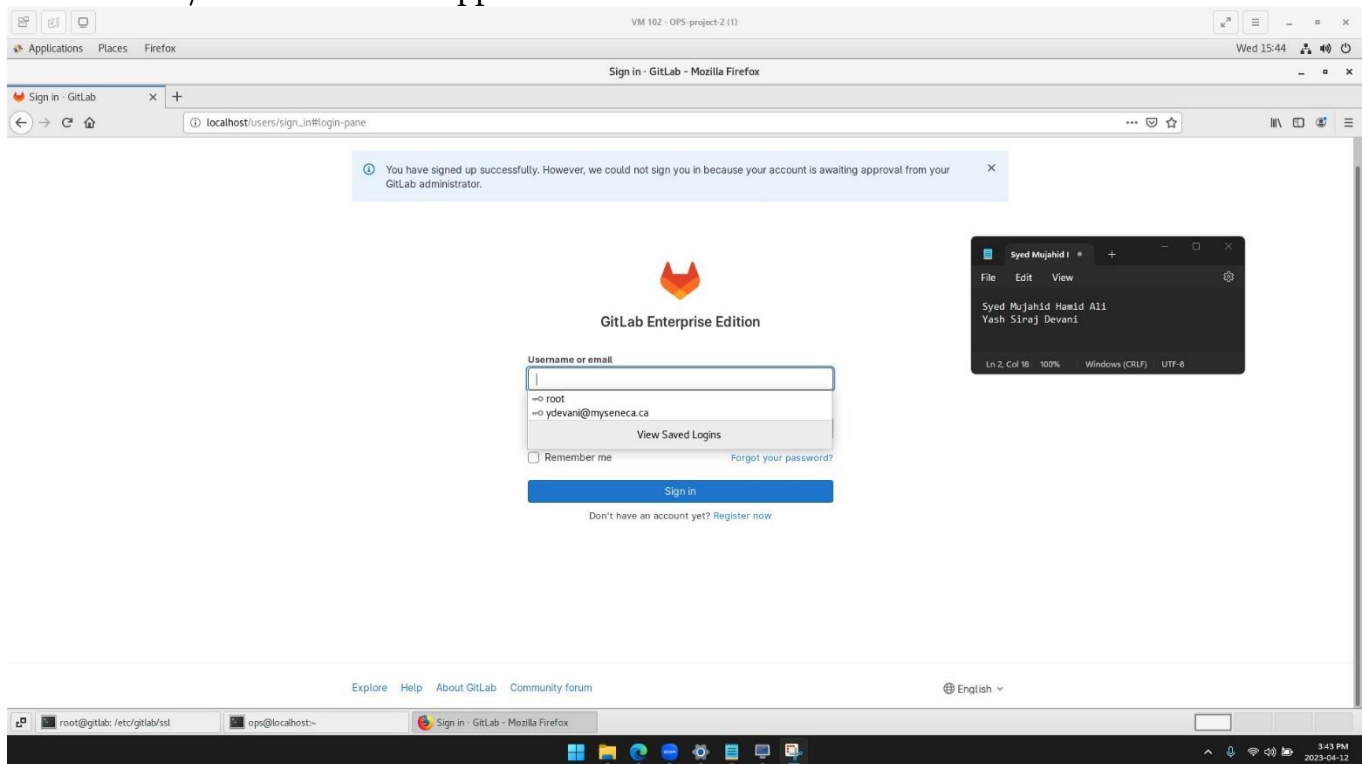
# Securing the Private Server

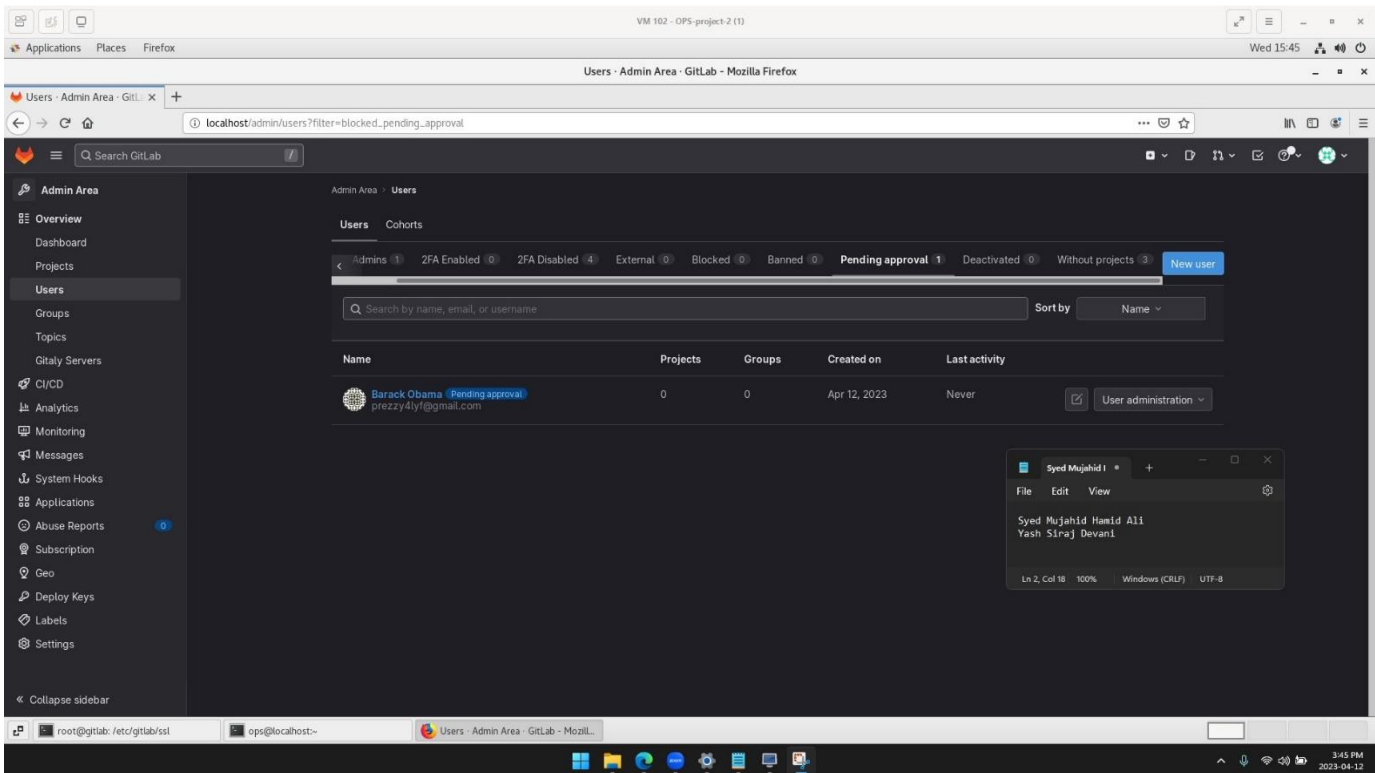
Let us try creating a fake ID and then check how it is different from an authorized user.



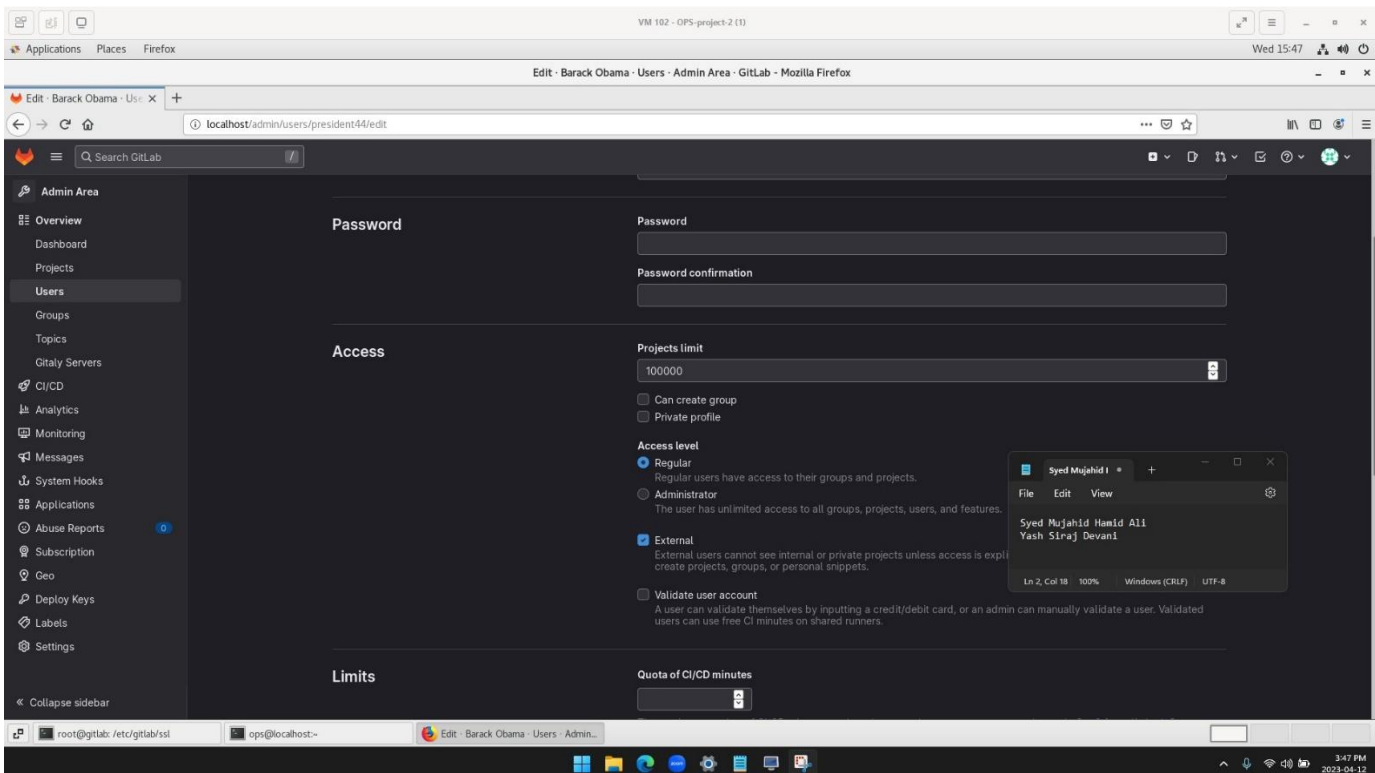
This shows the details that we decided to use for the signup page.

Next thing we get is this screen (below). As we can see, the user is not able to access the Git server yet, as the admin/root will need to approve of the user.



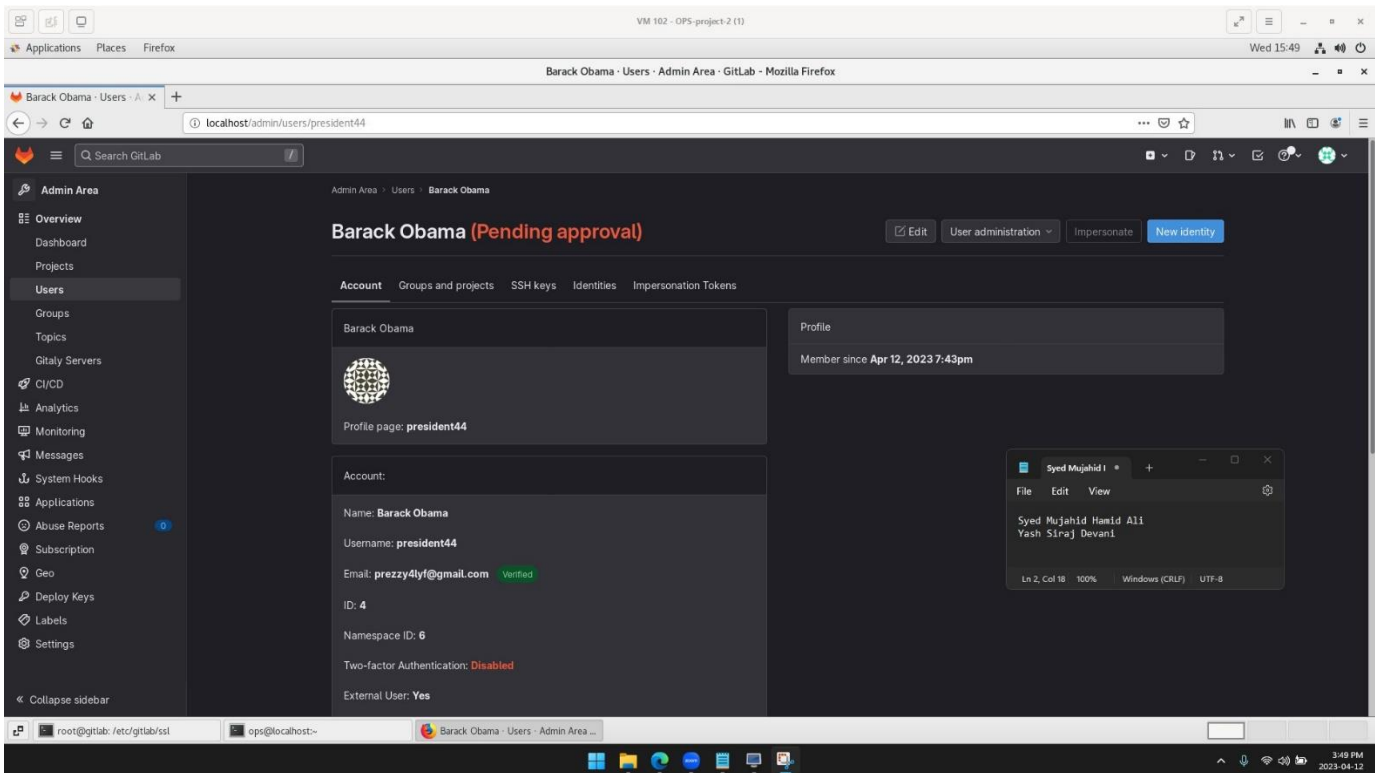


The request for the above user can be seen on the admin's page, waiting for approval. But as we know, this user is neither of the authorized users, which is us groupmates.



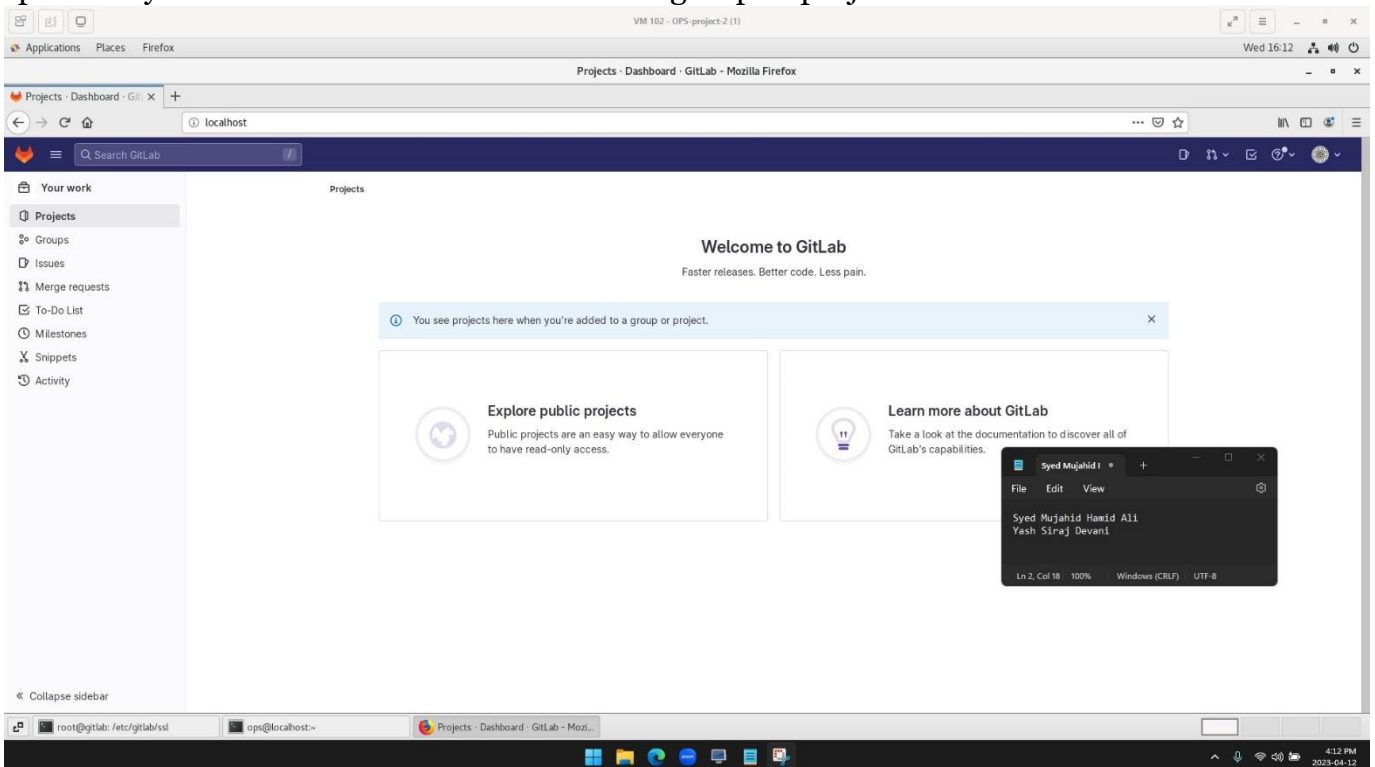
We will make this user external, so that this user is only able to access the server, not write anything. He will be able to do so only when the authorized users create a group or project and add him to it.



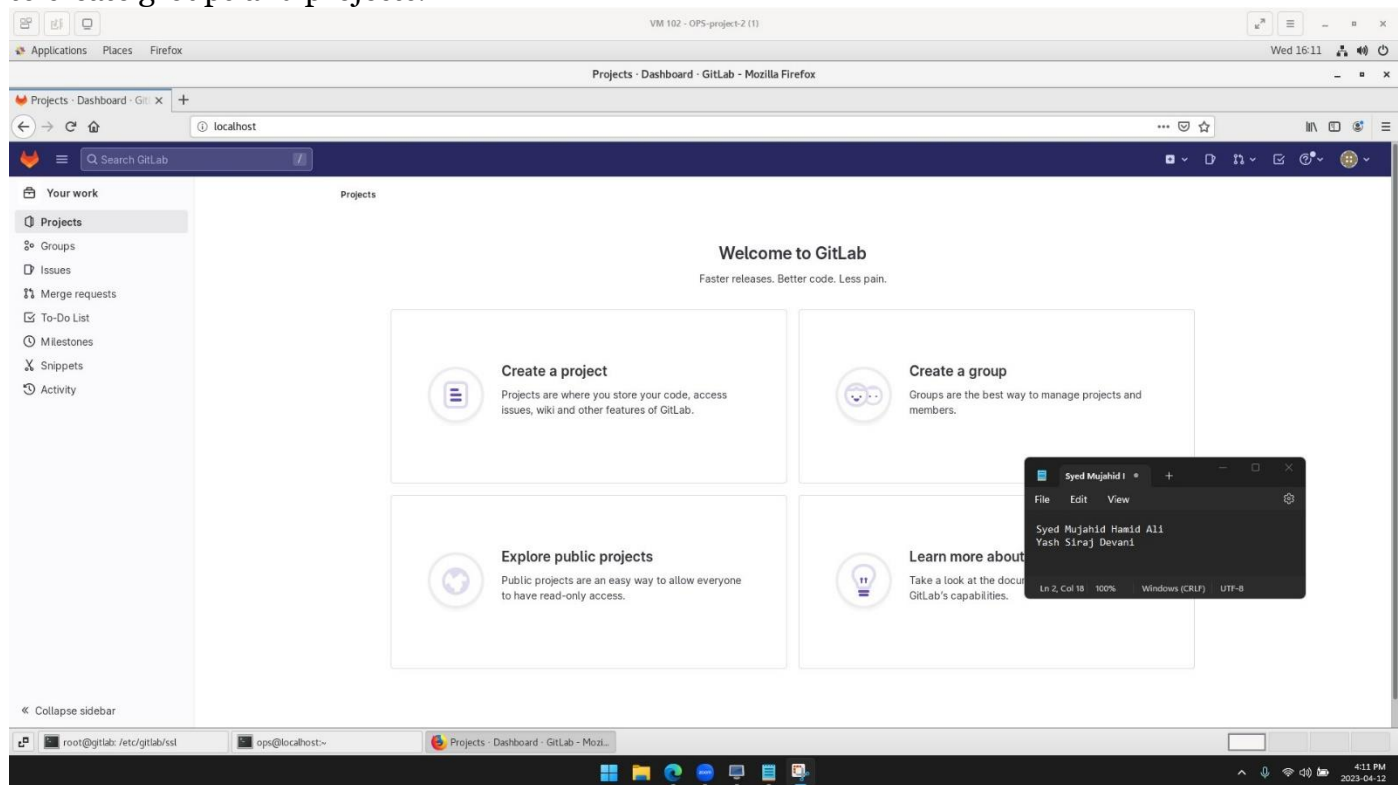


This shows an overview of the account, and as we can see, it shows that 2FA is disabled. This is because when we create a group, we will then be able to enable 2FA. 2FA in GitLab works in such a way that only groups that have it enabled will need 2FA.

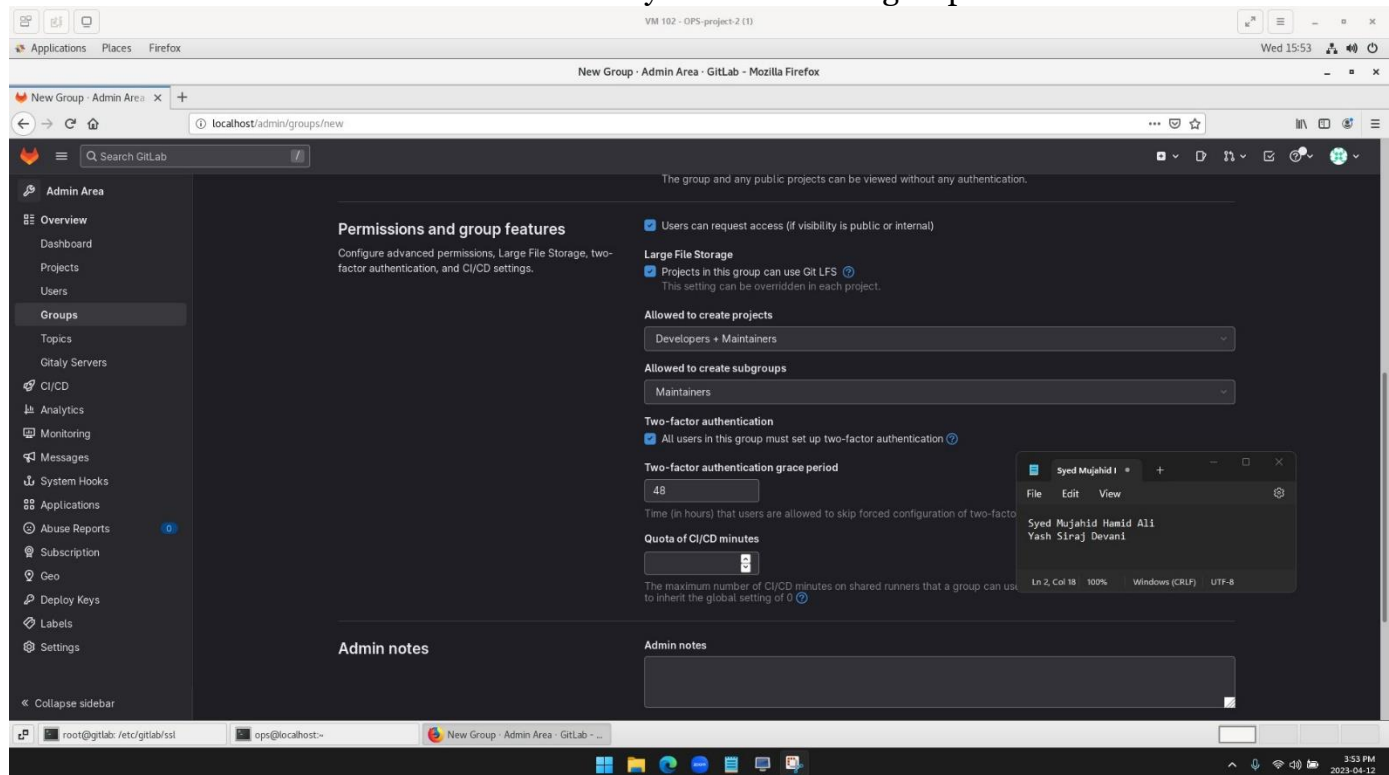
Once we log in successfully using this account, we can see that the interface is a bit different. This user specifically does not have the access to create a group or project.



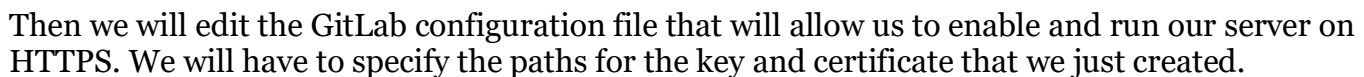
The screenshot below shows the same interface, but for an authorized user. These users have the right to create groups and projects.



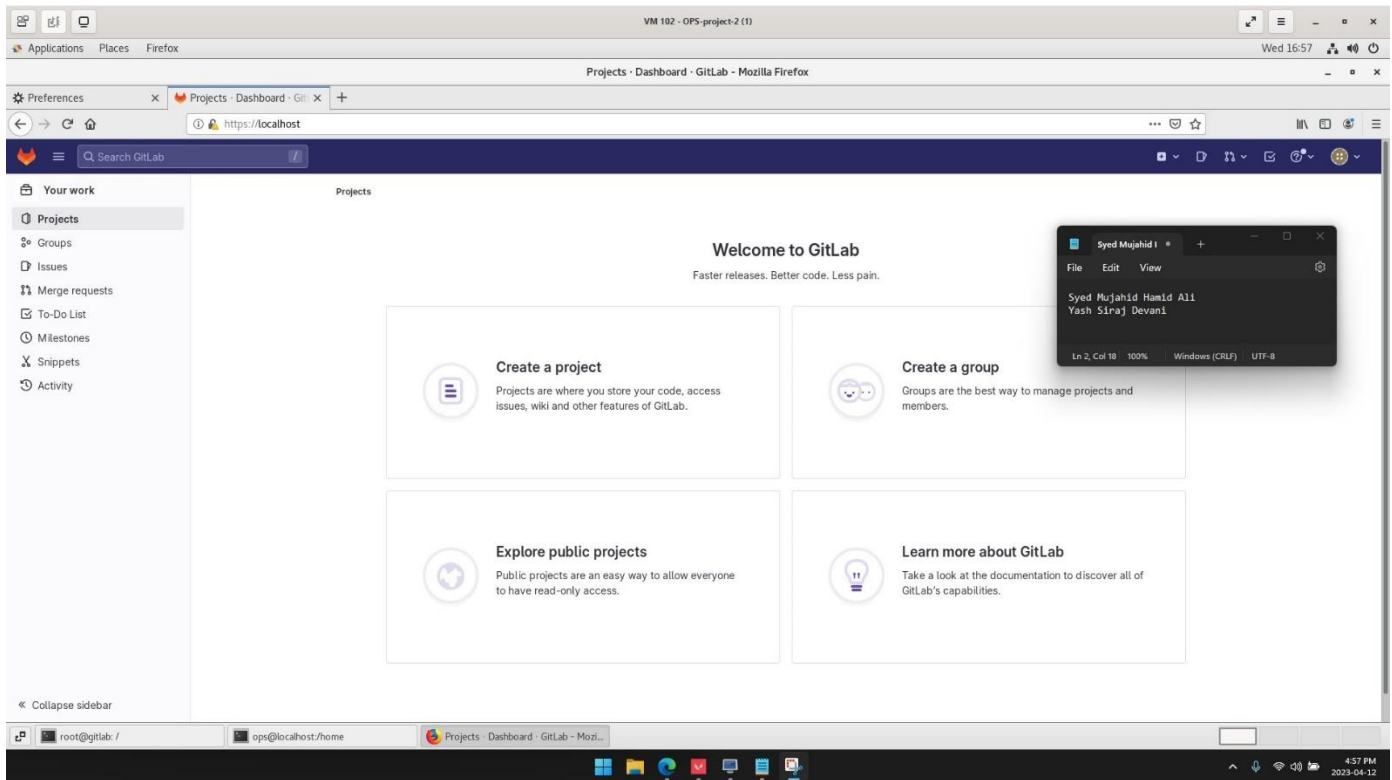
To show the 2FA process, we will create a group, just to test and then we will be able to see an option that will allow us to force enable 2FA for every member of the group.



We will start by generating a key and certificate pair that will be self-signed.



Lastly, we will add our keys to Firefox (the browser that we are using). What this will allow us to do is let the browser know that the keys are authorized, and it is okay for it to be recognized. Then we will be able to run the server on HTTPS.



## **Conclusion**

In conclusion, creating, deploying, and securing a GitLab server for a private company is crucial to protecting their code and ensuring safe communication among team members. By following the steps outlined in our process, including setting up secure passwords, enabling two-factor authentication, and configuring firewalls, companies can reduce the risk of attacks and safeguard their sensitive information. It is important to continually monitor and update security measures to stay ahead of potential threats and ensure the ongoing protection of the company's assets.

## **References**

"GitLab Docker images | GitLab," *docs.gitlab.com*.

<https://docs.gitlab.com/ee/install/docker.html#>

"Get Docker," *Docker Documentation*, May 14, 2022. <https://docs.docker.com/get-docker/>

"Swarm mode overview," *Docker Documentation*, Dec. 12, 2019.

<https://docs.docker.com/engine/swarm/>