**Table of Contents**

**Step-1 : Gitlab Account**

**Step-2 : Create a Group & Project**

1. **Create a New Group**
2. **Create a New Project**
3. **Demo Pipeline Parallel Jobs**

**Step-3: Variables Types**

* + 1. **Predefined variables**
       1. **Predefined CI/CD Variables Reference User Defined Variables:**
          1. **Project Level Variable Define:**

**Generate the Personal Access Token on Docker Hub**

* + - * 1. **Group Level Variable Define:**

**Step-4 Gitlab Runners**

1. **Install the Gitlab Runner**
2. **Gitlab runner status**
3. **Gitlab runner Service status**
4. **Register the Gitlab Runner**

**Step-5:**

* 1. **Project Import from SSH Key-Gen on EC2 Machine**
     1. **Add the Public Key to Gitlab Account**

**Step-6 Run the Pipeline**

* + 1. **Things to keep in Mind Successful Pipeline:**
       1. **Build Job:**
       2. **Push Job:**
       3. **Deploy Job:**

**Step-7 Artifacts**

**Pipeline Config with Artifacts**

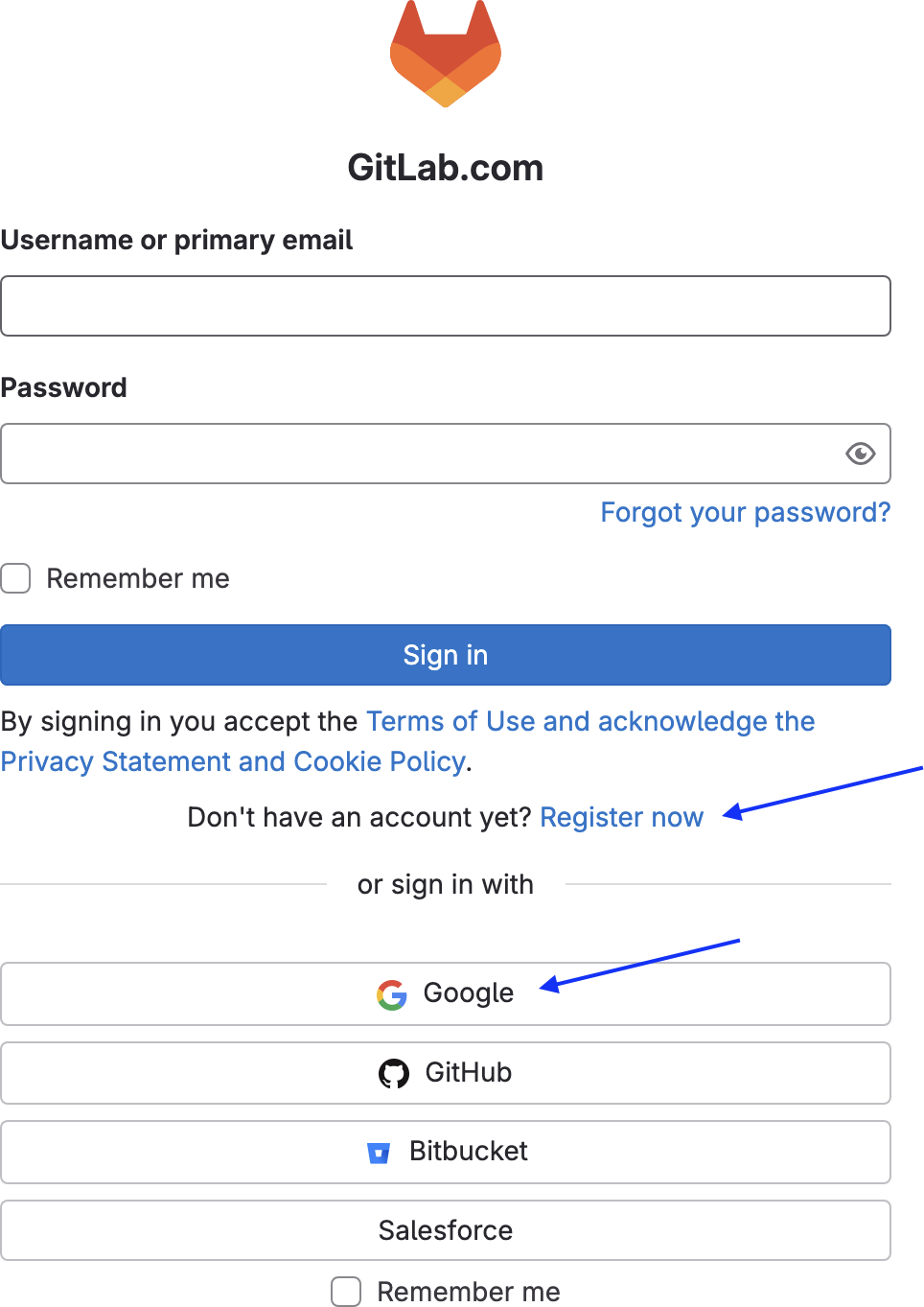
# Step-1 : Gitlab Account

### **Go to GitLab Website** – [Click here](https://gitlab.com/) **Click on "Register"** (Top right corner) **Enter Details:**

* **Username** (e.g., myusername)
* **Email Address** (Use your work or personal email)
* **Password** (Use a strong password)
* **Click "Register"** to create your account  
  **Verify Email** – GitLab will send a confirmation email. Click the link to verify your email.

✅ **Your GitLab account is now ready!** 🎉

.login with gitlab with your above information or login with google



**===============**

# Step-2 : Create a Group & Project

**Create a New Group**

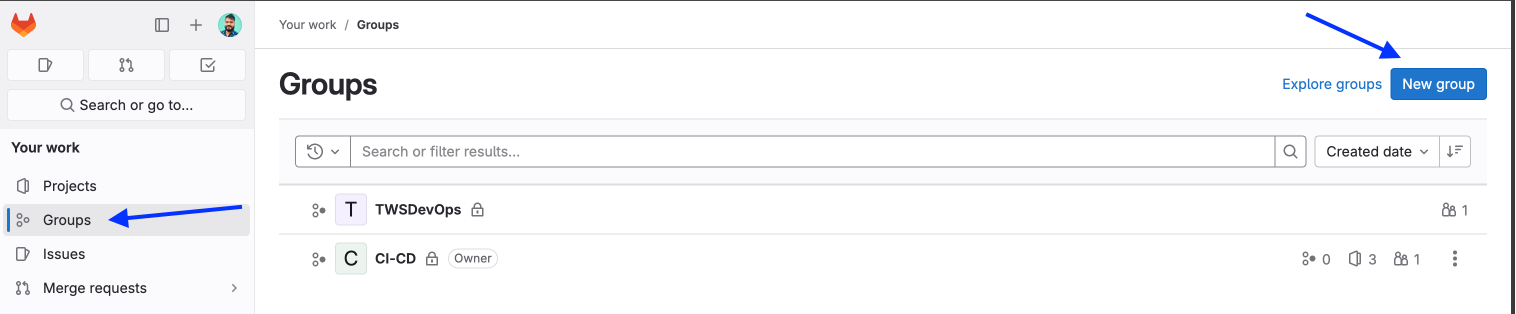
A **Group** in GitLab is used to **organize multiple projects under one team or organization**.

Click on **"Groups"** in the left sidebar  
 Click on **"Create a group"**  
 Choose **Group Type**:

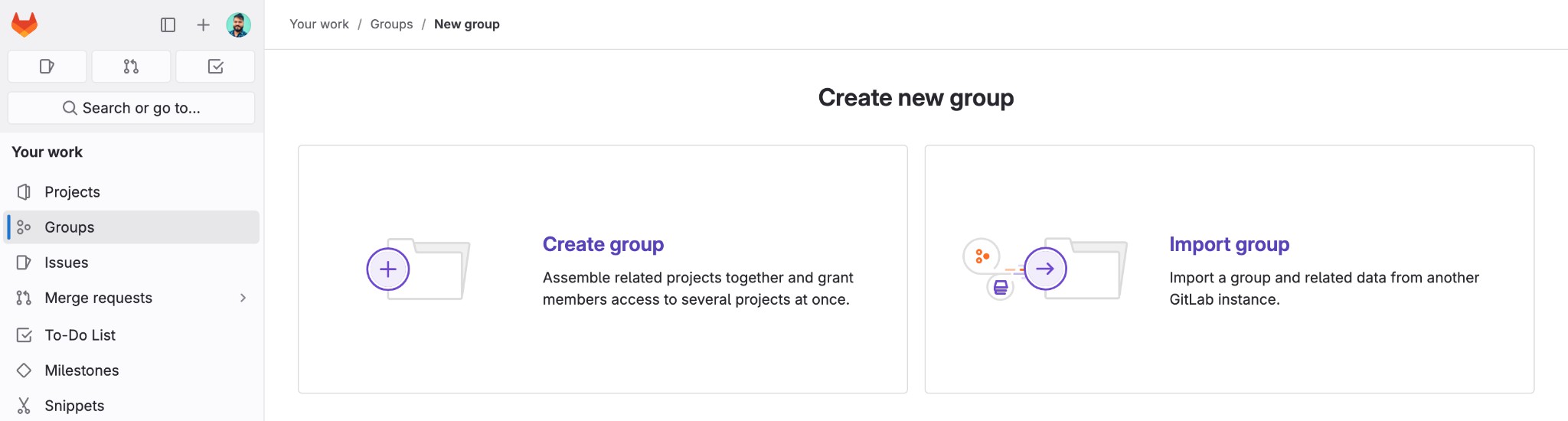
* **Create a new group** (For personal or team projects)

# # Let’s Create a New Group:

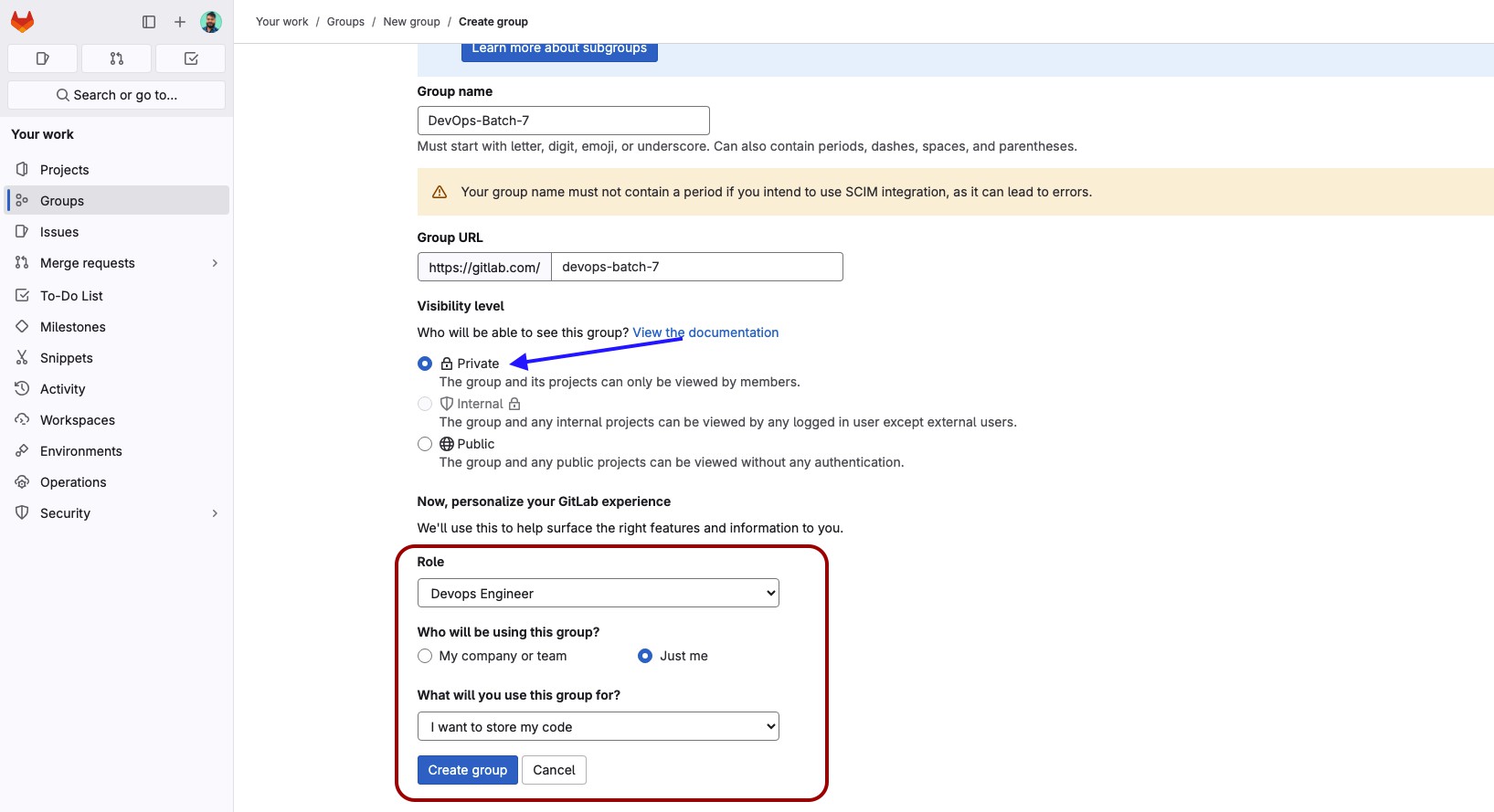
Click on Group icon —> New Group



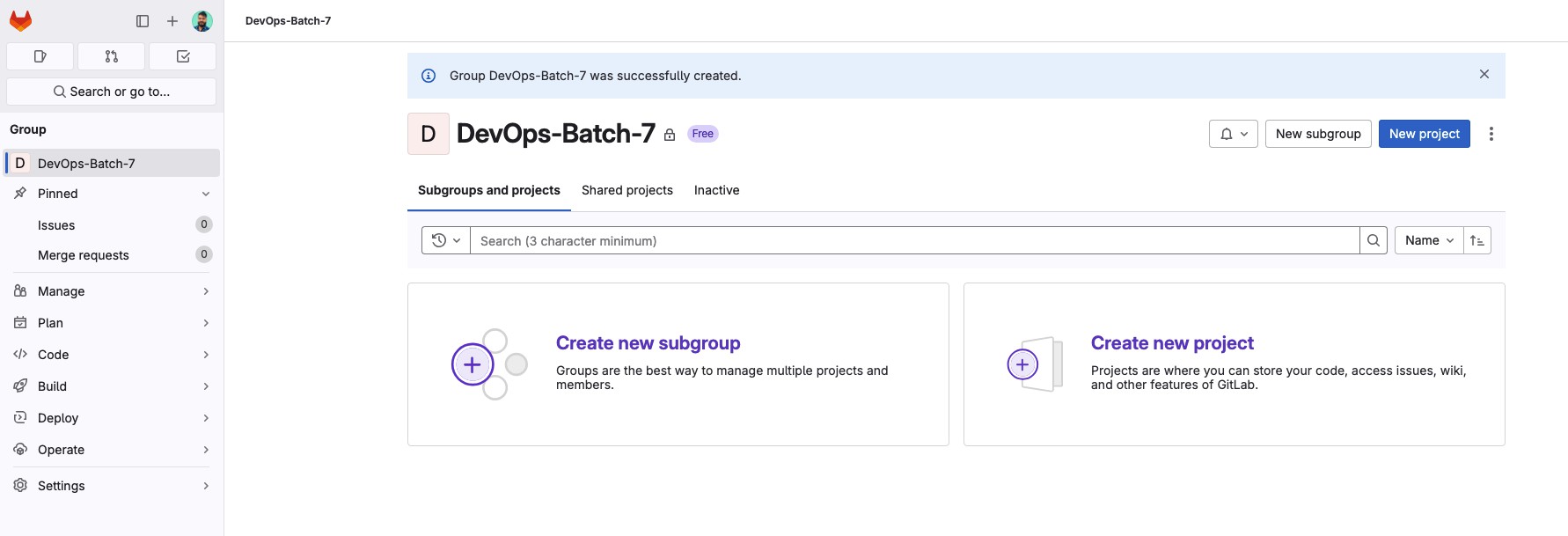
Click on **“Create Group”**



* **Enter Group Name & URL**
* Example: **Group Name** – **DevOps-Batch-7**
* Group URL: gitlab.com/ **devops-batch-7**  
  **Set Group Visibility:**
* **Private** (Only members can see)
* **Internal** (Only GitLab users can see)
* **Public** (Anyone on the internet can see)  
  Click **"Create Group"** 🎉



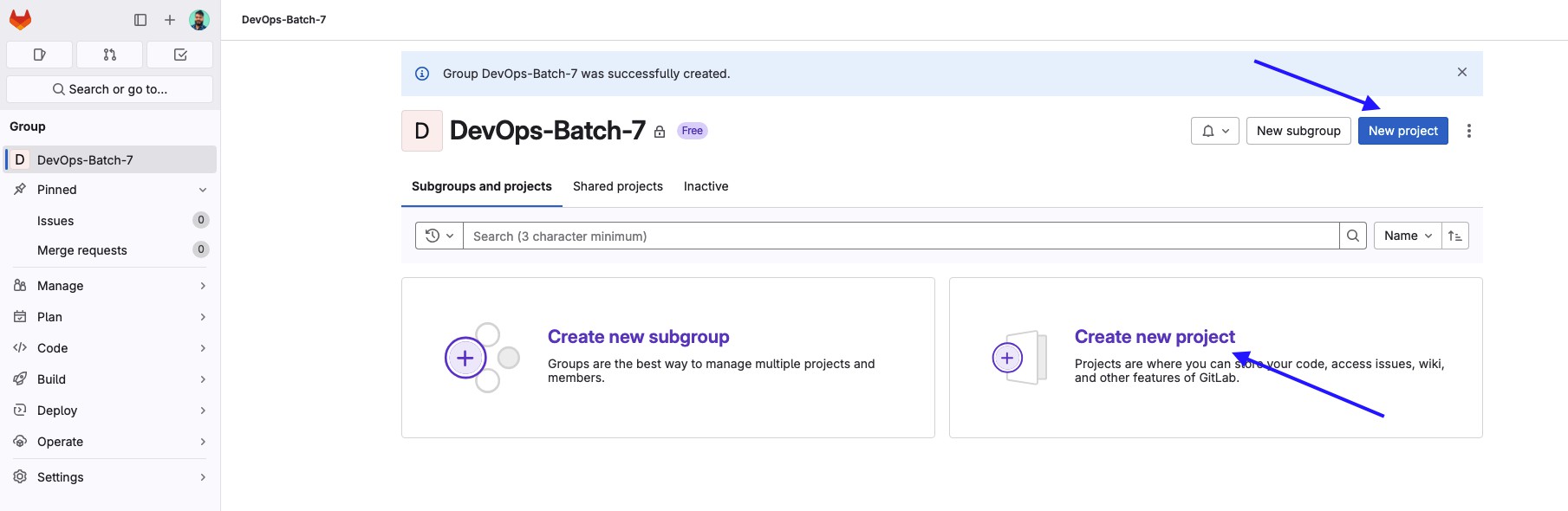
# Once created, it will give you the below screen, where you can create subgroup or New Project .



**===============**

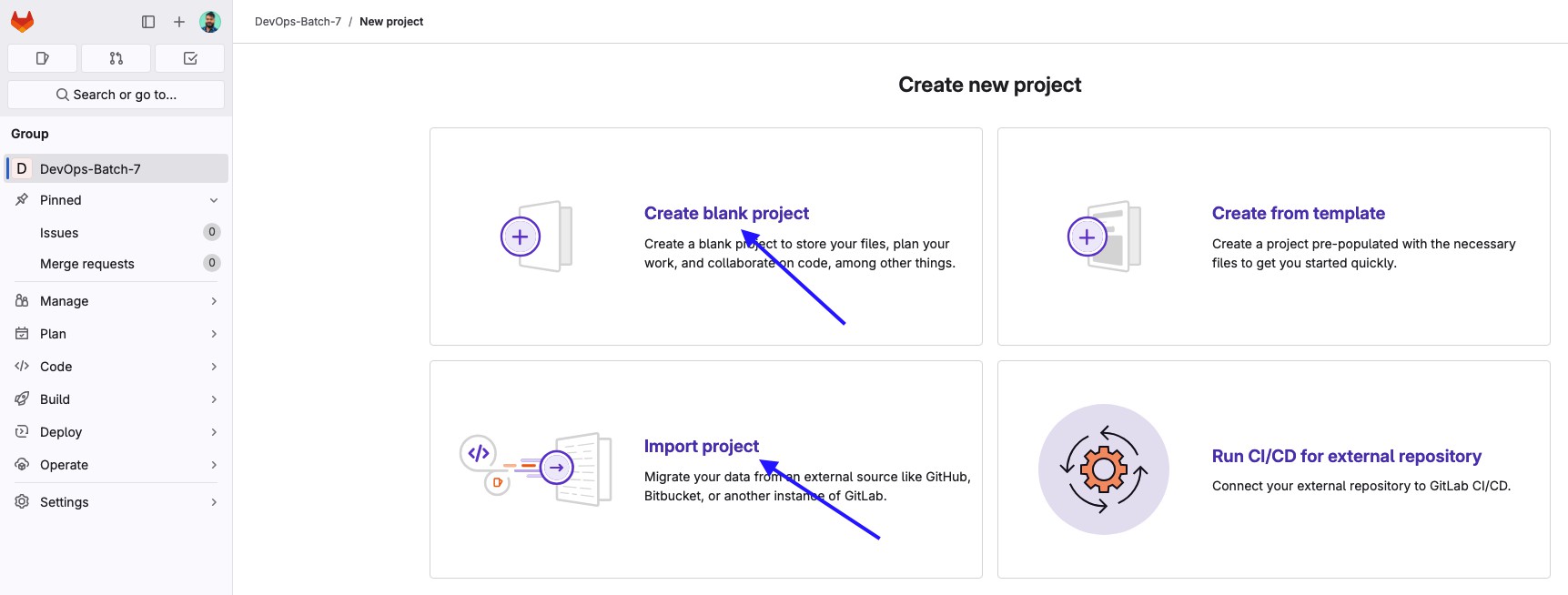
# Create a New Project

Inside the **Group**, click **"New Project"**

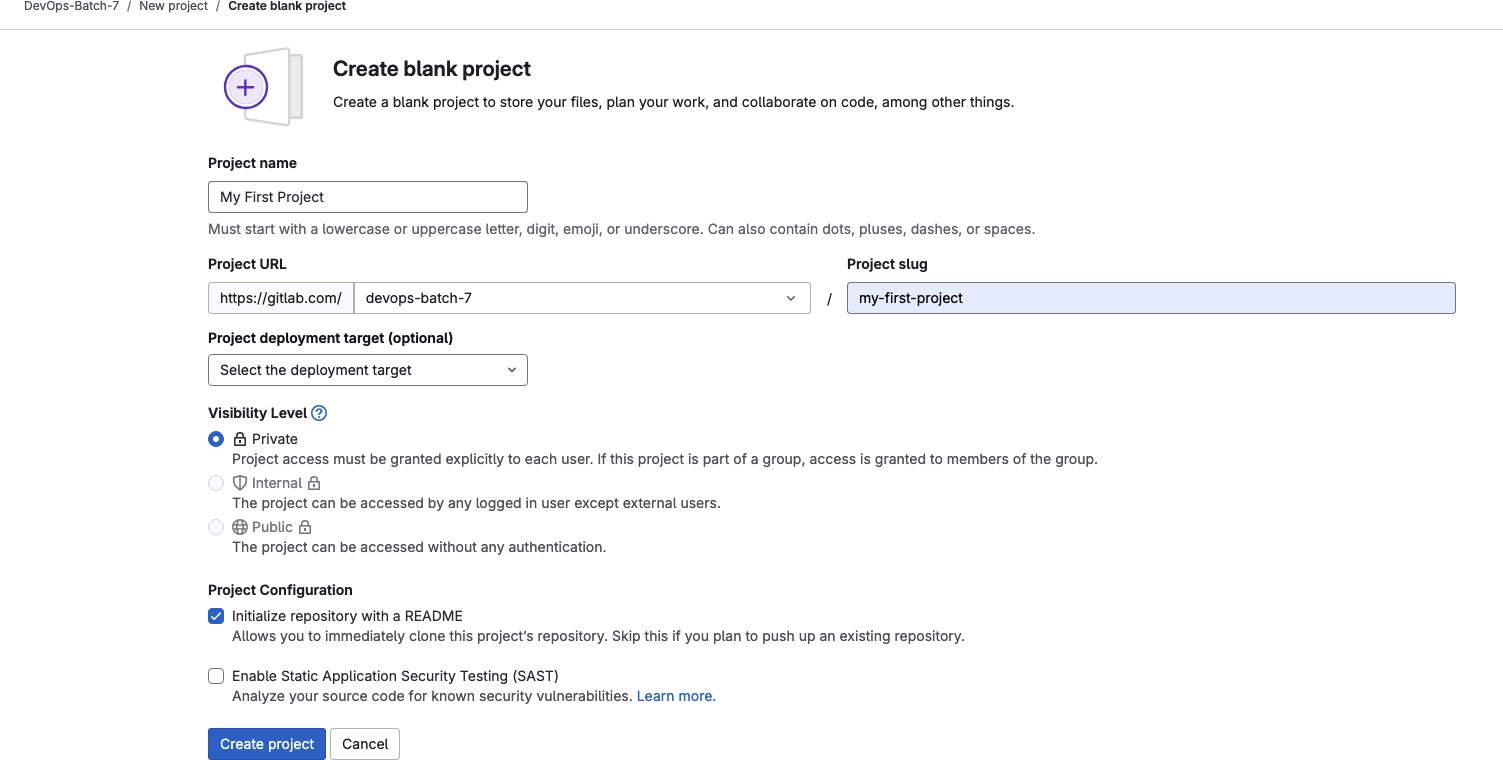


Once you click on New Project, it will give you 4 Options, us you can see below:

* 1. **Create a Blank Project :** If you want to create a New Project, then select this Option.
  2. **Import Project :** You can import the project from github, bitbucket ctc.



# Give a Name of your Project & Create a Project.



### **Add Team Members to Your Group**

Go to **Group Settings → Members**  
Click **"Invite members"**  
Enter their **GitLab username or email**  
Set their **Role & Permissions**:

* **Guest** – Read access only
* **Reporter** – Can view issues and merge requests
* **Developer** – Can push code & create merge requests
* **Maintainer** – Full project control
* **Owner** – Full group control  
   Click **"Invite"** 🎉

✅ **Your team is now added!** They can contribute to the repository.

### **Clone Your GitLab Repository (For Local Development)**

Open **Git Bash / Terminal**  
Run the following command to **clone your repo**:

git clone https://gitlab.com/MyDevOpsTeam/MyFirstProject.git

Navigate to the project directory:

cd MyFirstProject

Add your first file:

echo "# My First GitLab Project" > README.md

git add README.md

git commit -m "Initial commit"

git push origin main

✅ **Your project is now uploaded to GitLab!**

### **Set Up CI/CD Pipeline (Optional)**

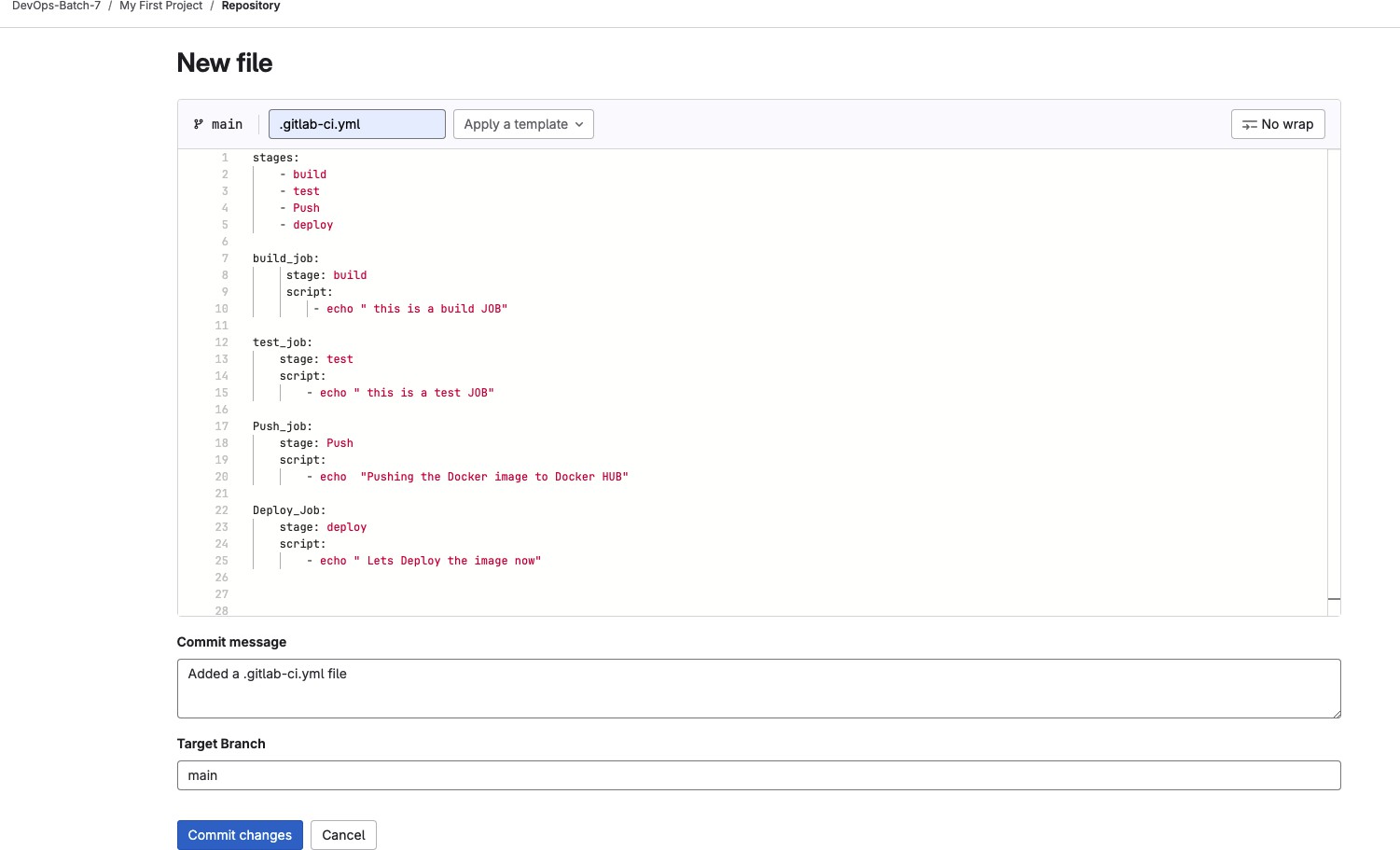
In your GitLab project, go to **CI/CD → Pipelines**  
Click **"Set up CI/CD"**  
Create a **.gitlab-ci.yml** file in your project:

Let’s Create a **.gitlab-ci.yml** file to run a pipeline.

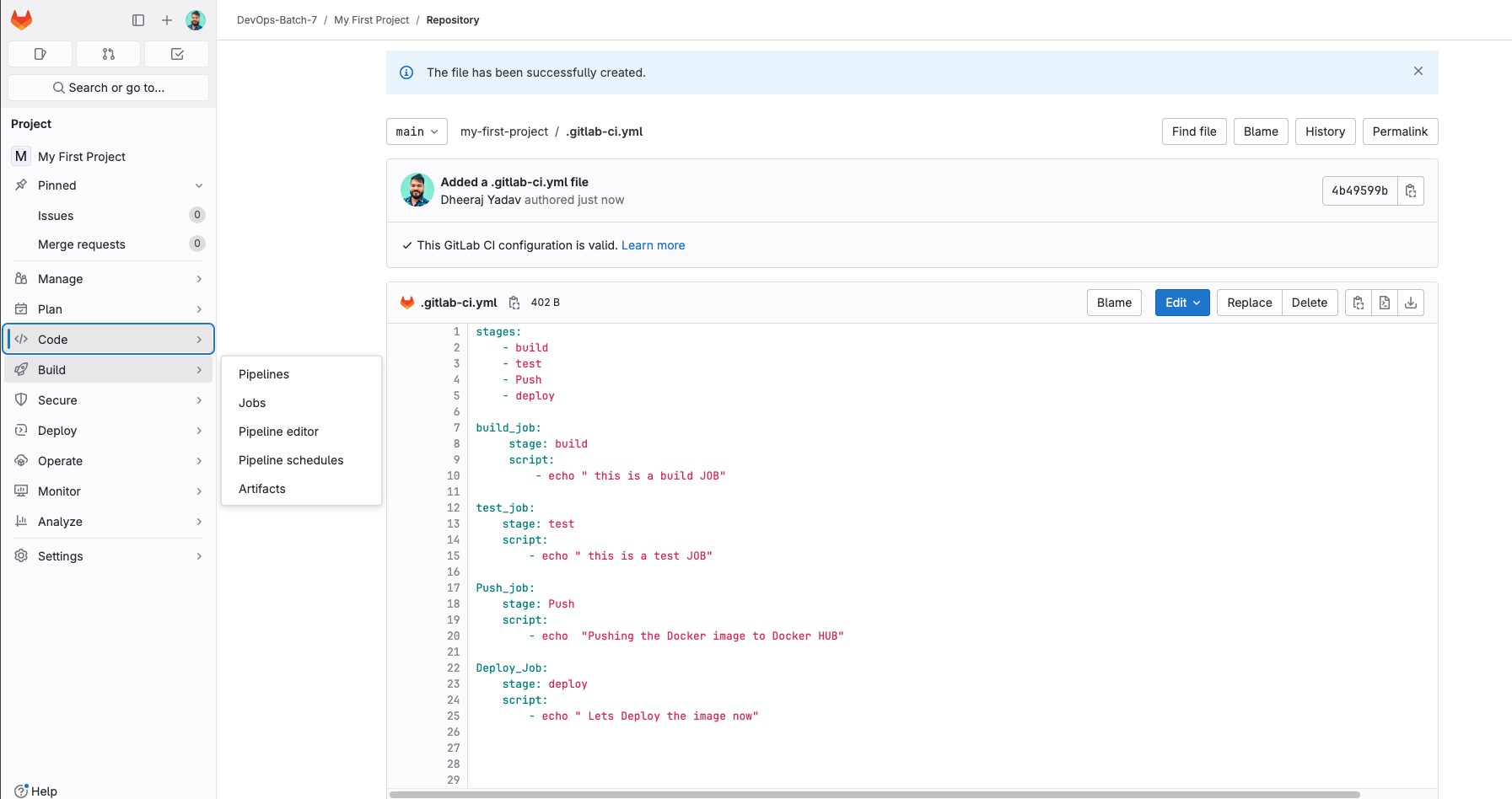
**===============**

# Demo Pipeline

**Here is the test Pipeline Config:**



As soon as you click on Commit changes , it will trigger a **Build —> Pipelines**



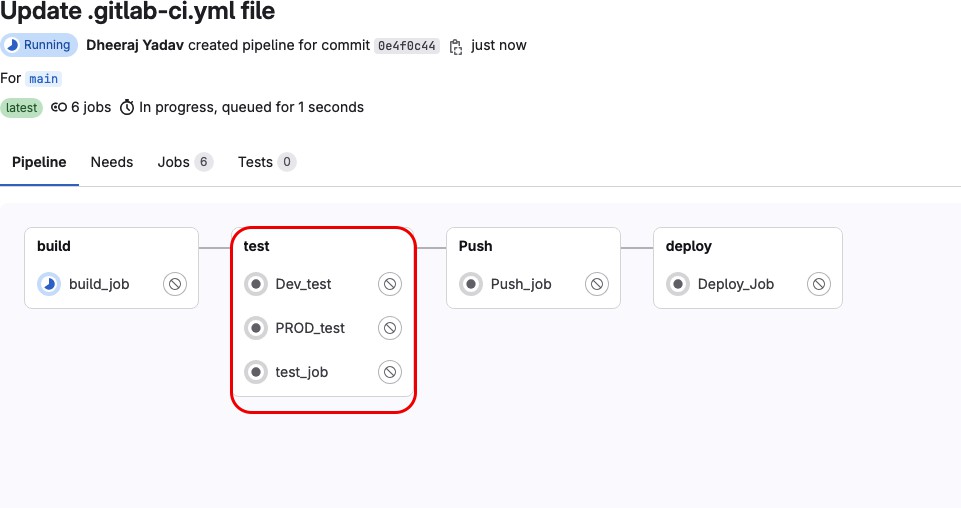
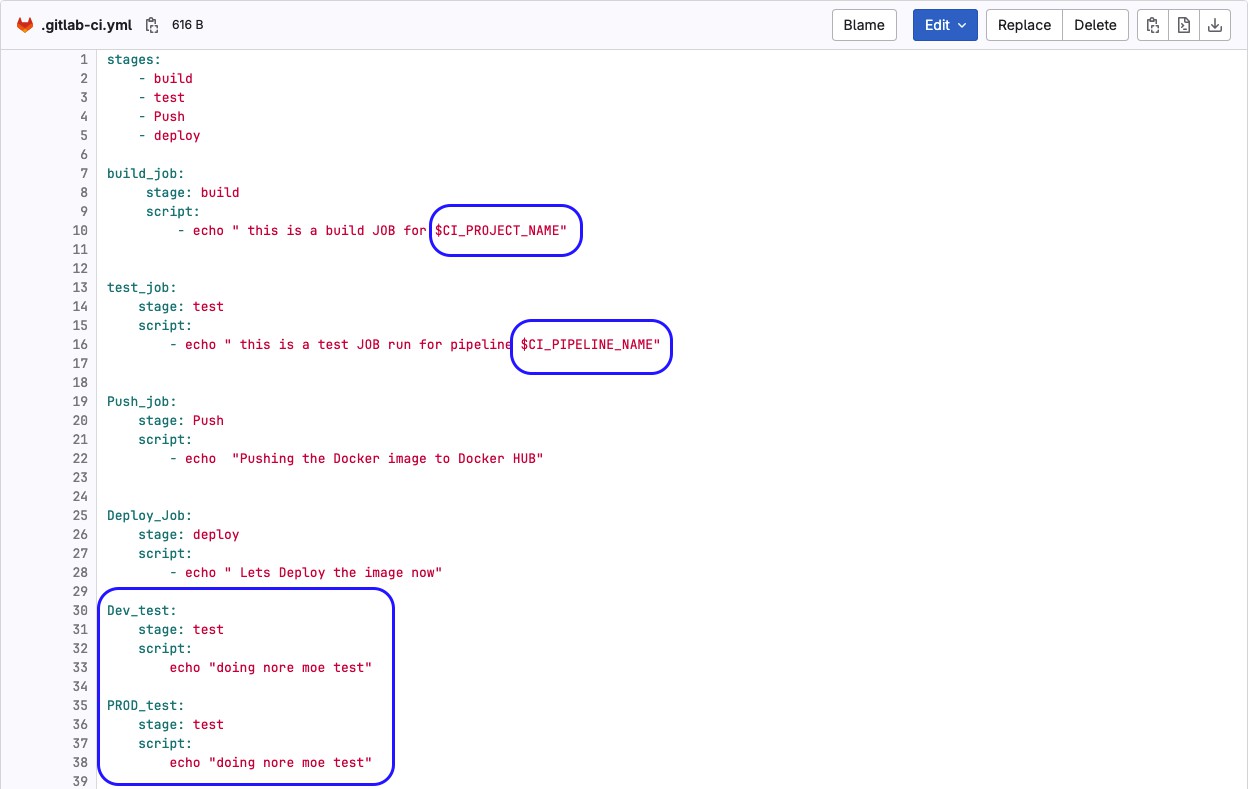
Pipeline has been triggered & Completed successfully.



**===============**

# Parallel Jobs

Create 3 test which will run parallel, as you can see below.



**===============**

# Step-3: Variables

**GitLab CI/CD Variables** are **environment variables** that store sensitive information, credentials, and configuration settings used in **CI/CD pipelines**. These variables allow developers to securely manage secrets and avoid hardcoding values in scripts.

## **🔹 Types of GitLab CI/CD Variables**

GitLab provides **two main types** of variables:

| **Variable Type** | **Description** |
| --- | --- |
| **Predefined Variables** | Built-in system variables provided by GitLab |
| **Custom Variables** | User-defined environment variables (e.g., API keys, credentials) |

## **1. Predefined GitLab CI/CD Variables**

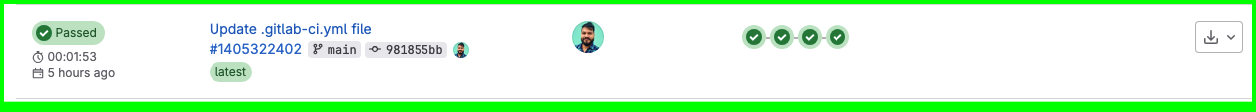
GitLab provides **default environment variables** that can be used inside .gitlab-ci.yml pipelines.

### ✅ **Common Predefined Variables**

| **Variable** | **Description** |
| --- | --- |
| $CI\_COMMIT\_REF\_NAME | Branch name of the commit |
| $CI\_COMMIT\_SHA | Full commit SHA |
| $CI\_COMMIT\_SHORT\_SHA | Short commit SHA |
| $CI\_PROJECT\_NAME | Name of the project |
| $CI\_PROJECT\_NAMESPACE | Namespace of the project |
| $CI\_PIPELINE\_ID | Unique ID of the pipeline |
| $CI\_JOB\_ID | Unique ID of the current job |
| $CI\_JOB\_STAGE | Current job’s stage name |
| $CI\_REGISTRY\_IMAGE | Docker registry image path |

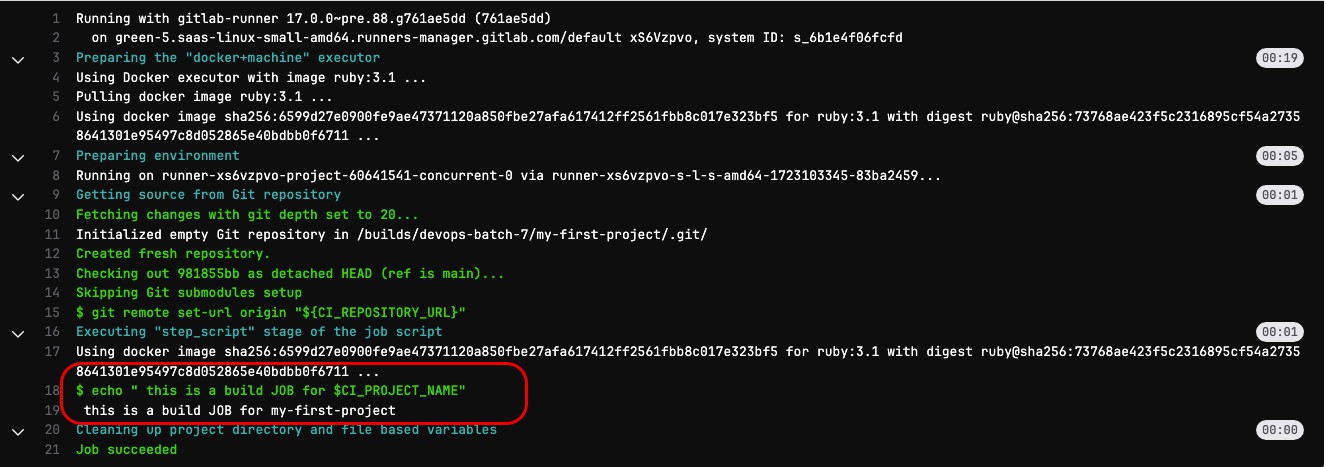


Run the Pipeline & it’s successful.



Let’s check the output

# This way we can use the Predefined Variable to your pipelines.



**Predefined CI/CD Variables Reference**

https://docs.gitlab.com/ci/variables/predefined\_variables/

**===============**

## **2. Creating Custom Variables in GitLab**

You can create **custom environment variables** for **storing secrets, API keys, and configurations**.

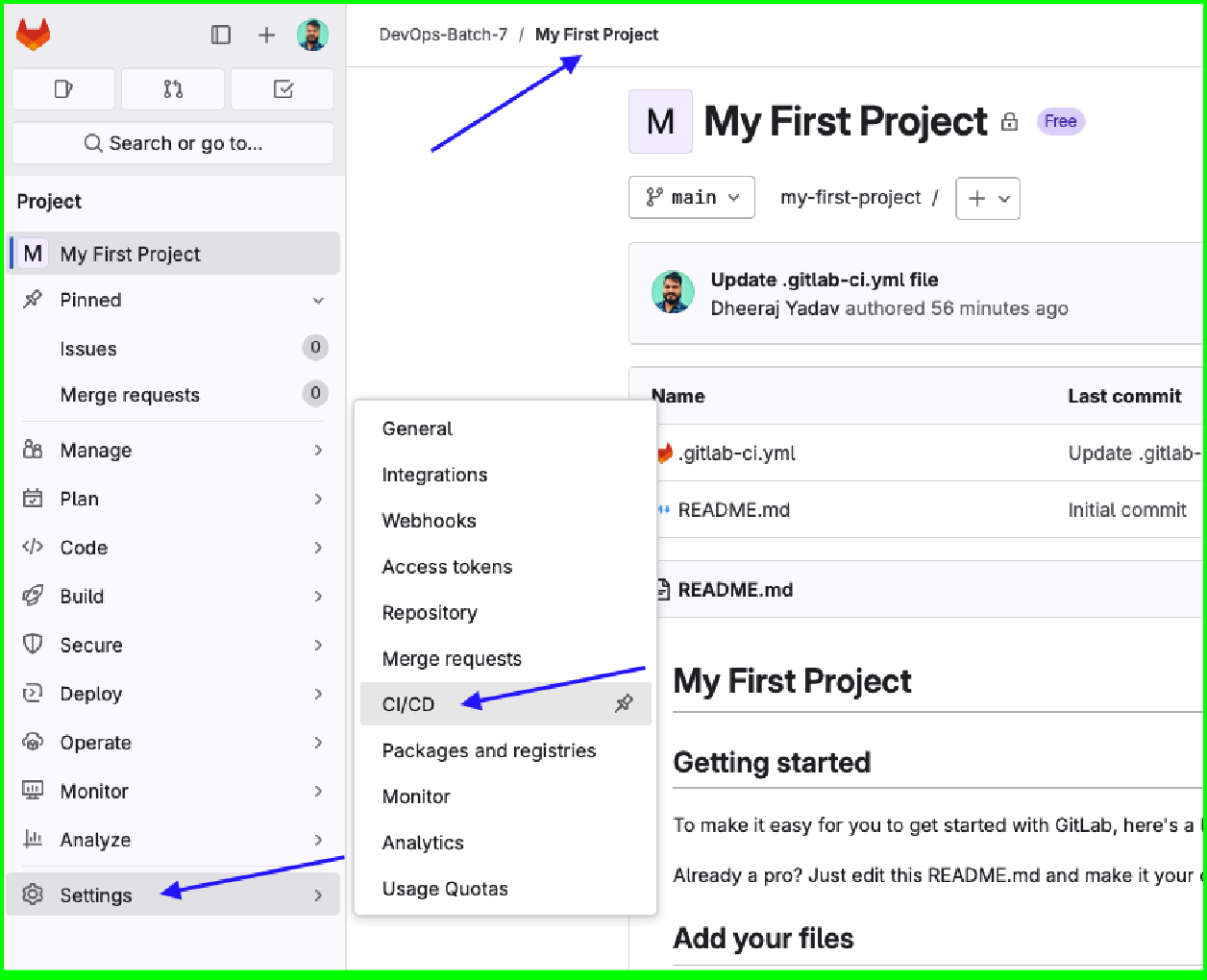
### ✅ **How to Add Custom Variables in GitLab**

User Defined Variables:

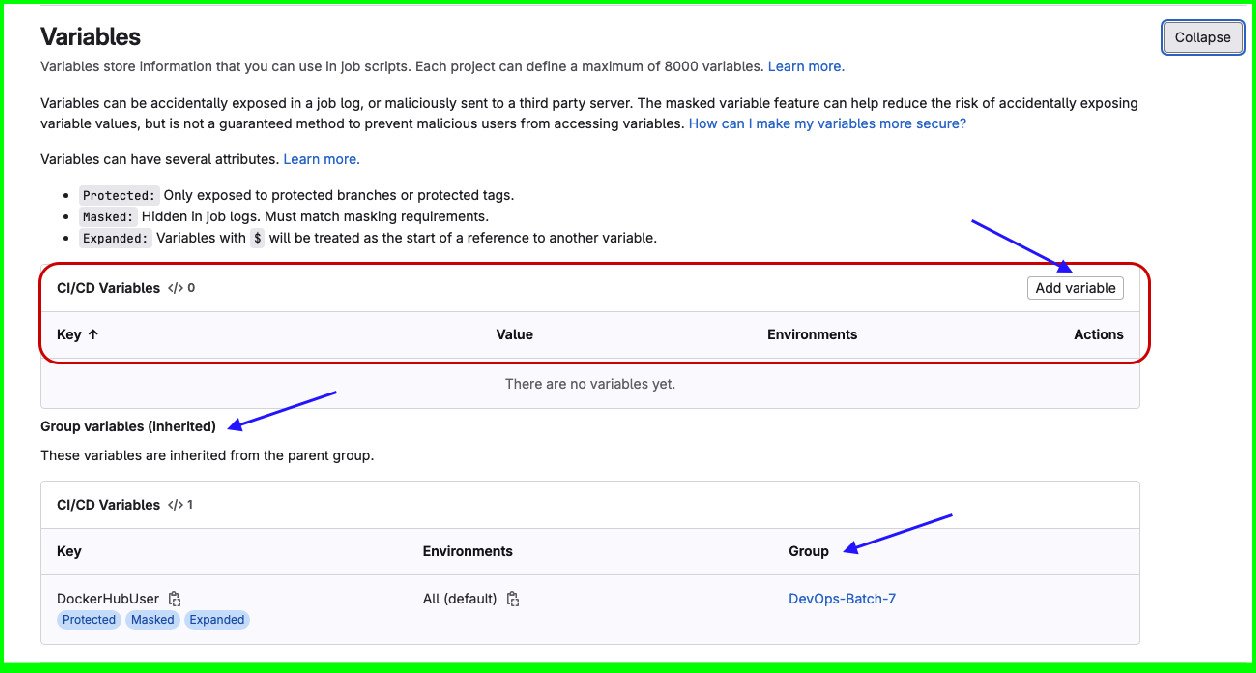
We can Define User Defined Variables either on **Project Level or Group Level.**

# Project Level Variable Define:

**Under Project—-> Setting —-> CI/CD —> Variables —> Expand**

****

# Add the Variable by clicking on **Add Variables** & there, user Defined Variables will be defined on Project Level.



# Add Variable:

We will be adding the DockerHub Username & Token to push the created docker image to DockerHub.

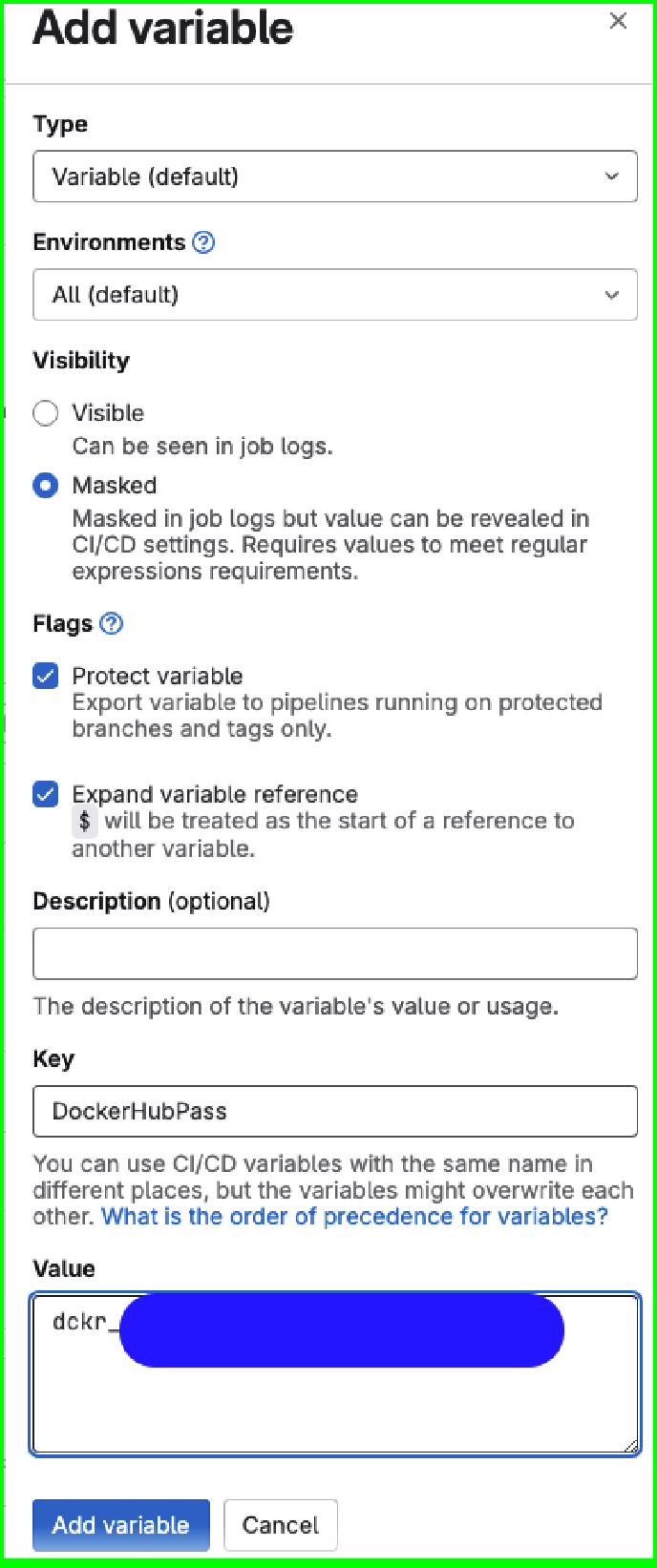
**======**

**Key :** Variable name “You can define as DockerHubUser”

**Value:** Docker Hub token

How to generate docker hub token step by step:

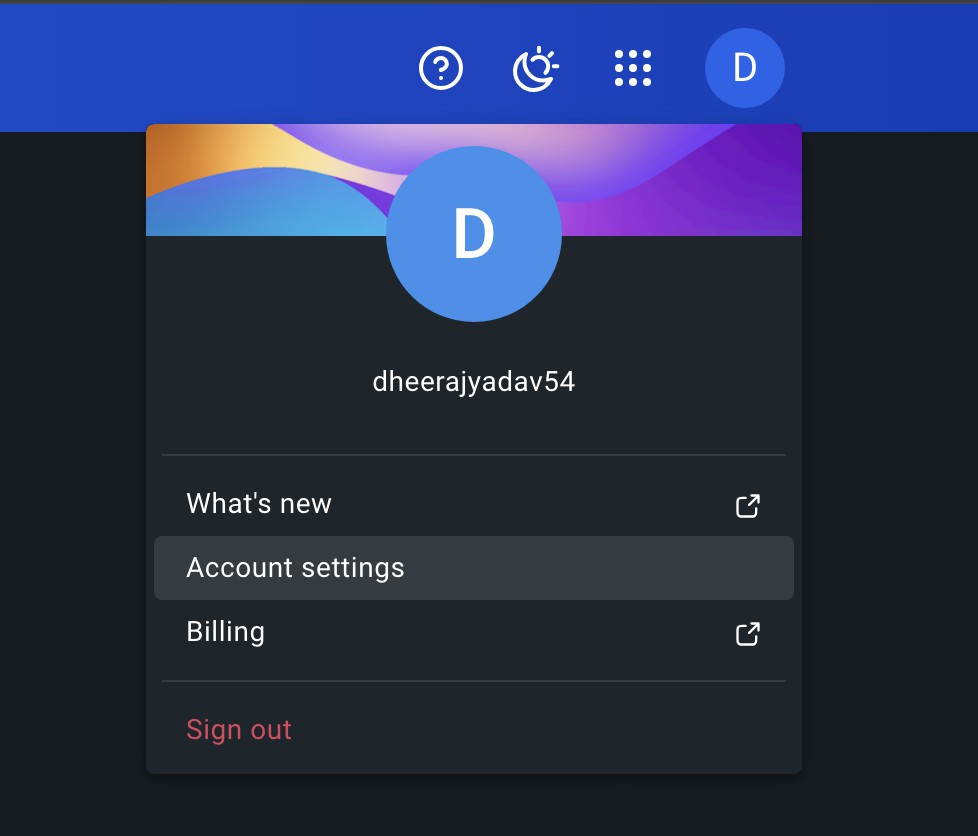
Follow the below “**Generate the Personal Access Token on Docker Hub”**



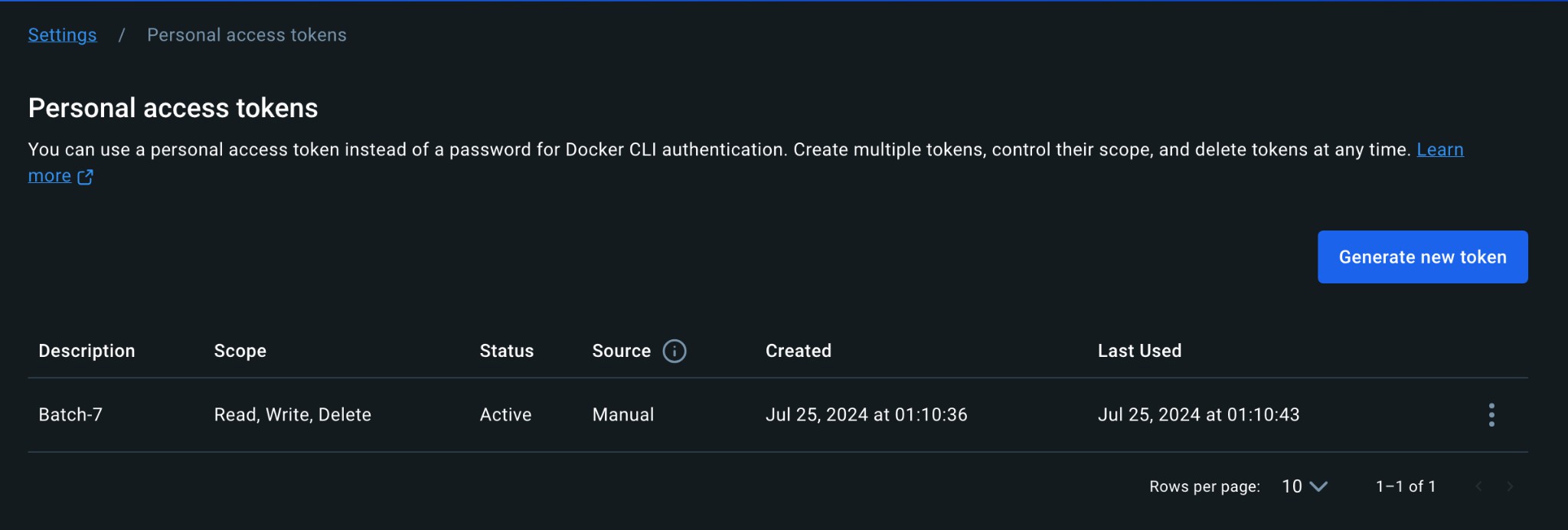
**===============**

# Generate the Personal Access Token on Docker Hub

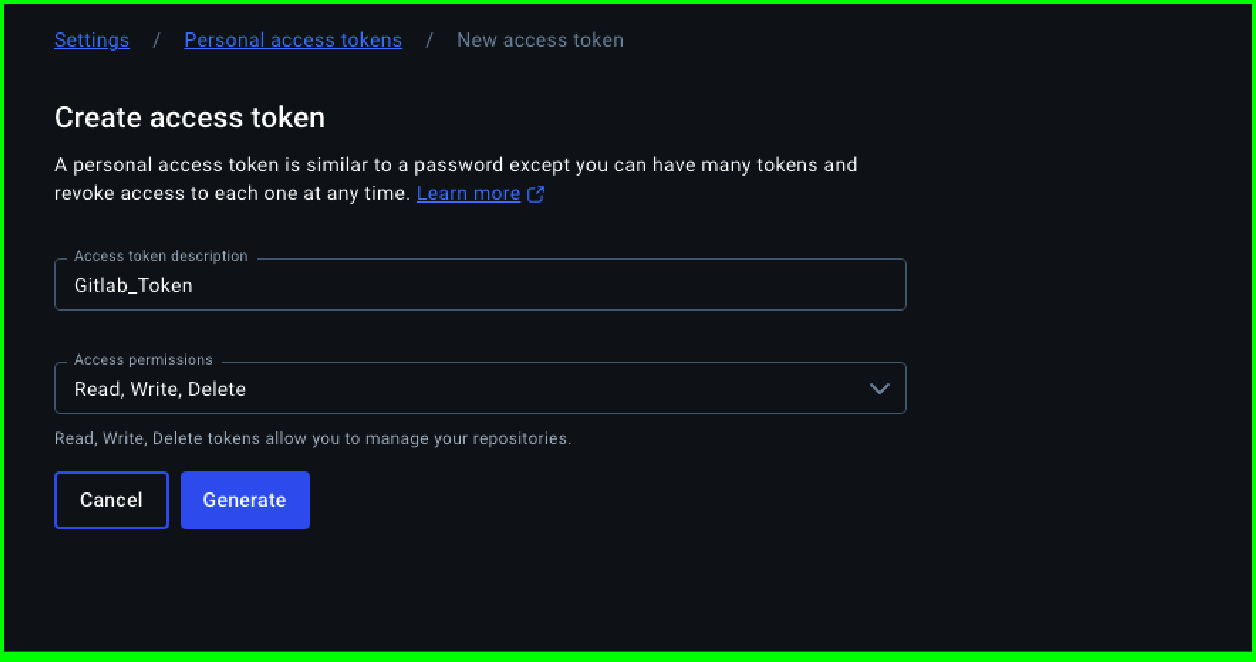
u) Docker Hub Account —-> Account Setting



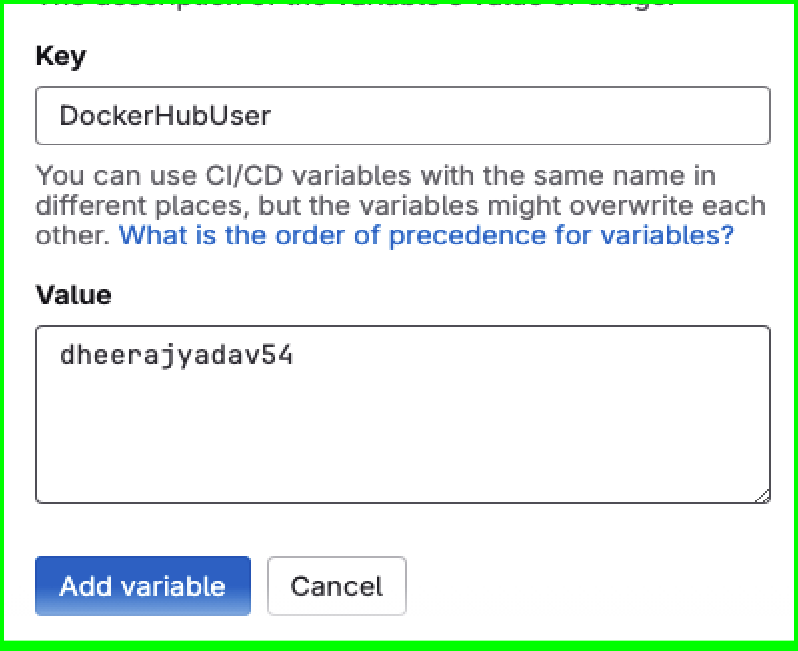
b) Personal access tokens → Generate new token



# Click on Generate & Copy the token. Once you Copied the Token, go to Jenkins Credentials page & paste under Password Section.

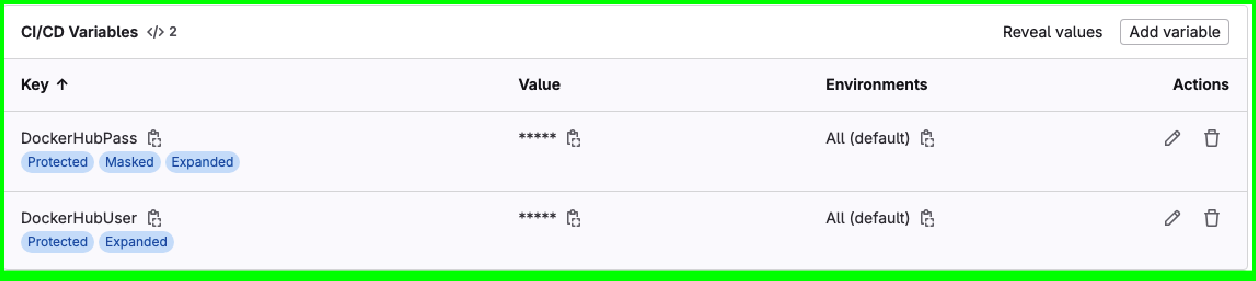


# Add the DockerHubUser Variable.



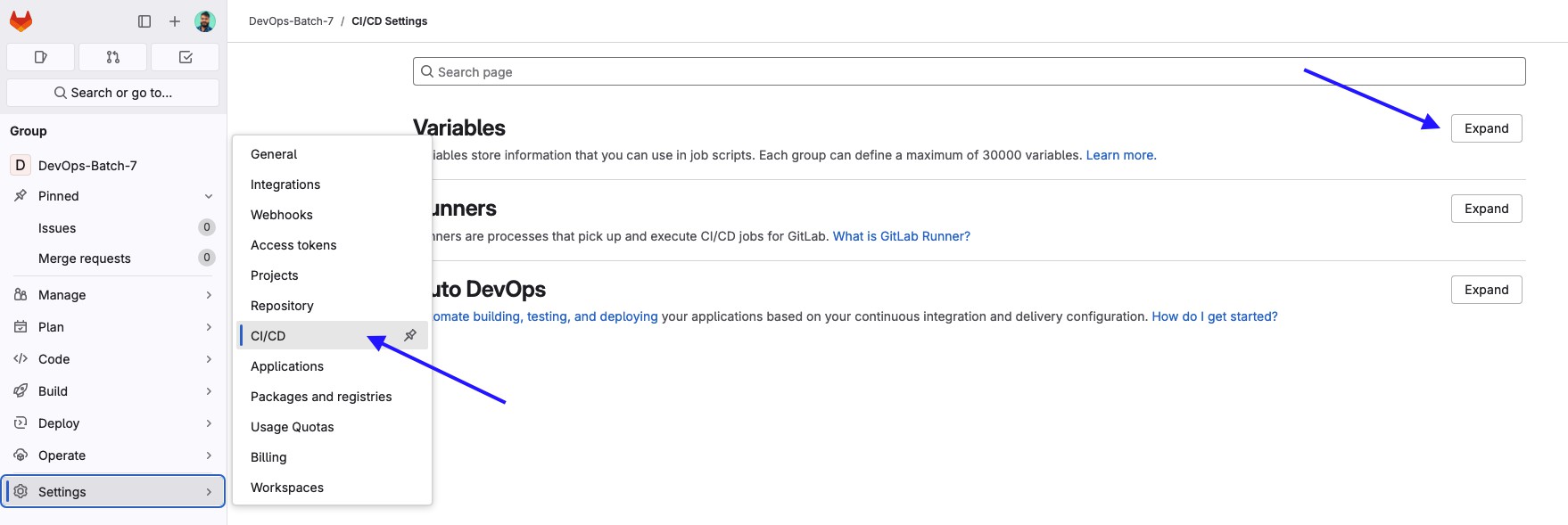
# Added both DockerHubUser, DockerHubPass User defined Variables, Let’s use these in the pipelines.

# Note: These are User Defined, so can be any Variable Name.

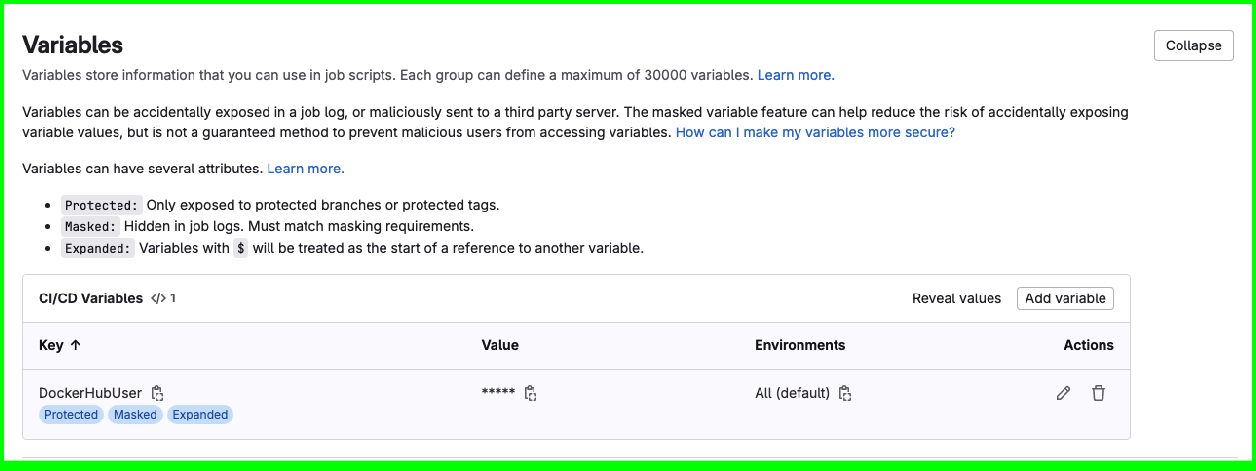
****

* + 1. **Group Level Variable Define:**

**#** Creating the Variables, follow the same process as shown on Project Level. Group—> Setting→ CI/CD —> Variables → Expand



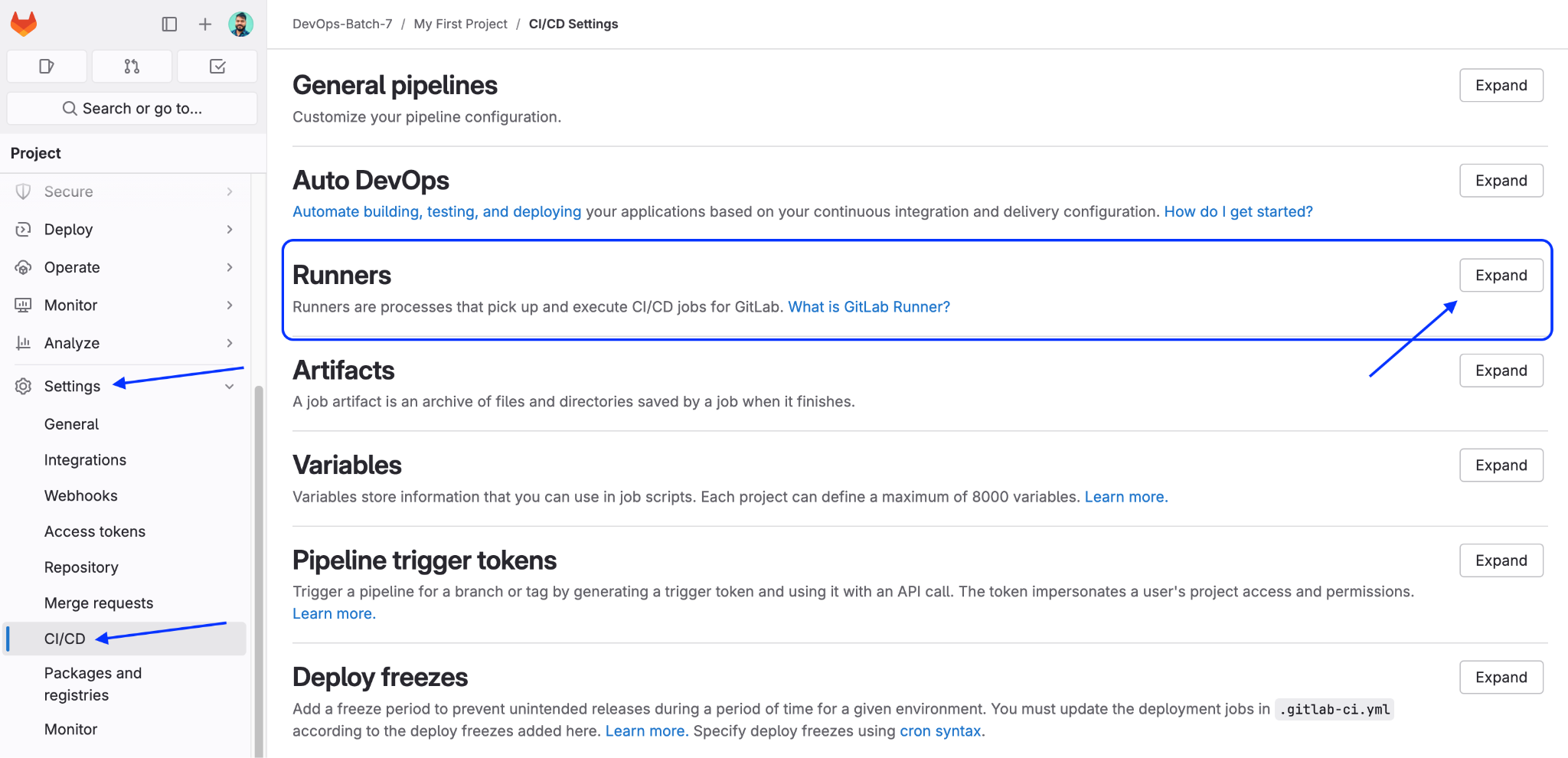
**#** Once you add the Variables on Group Level, it will be assigned to all your Projects.



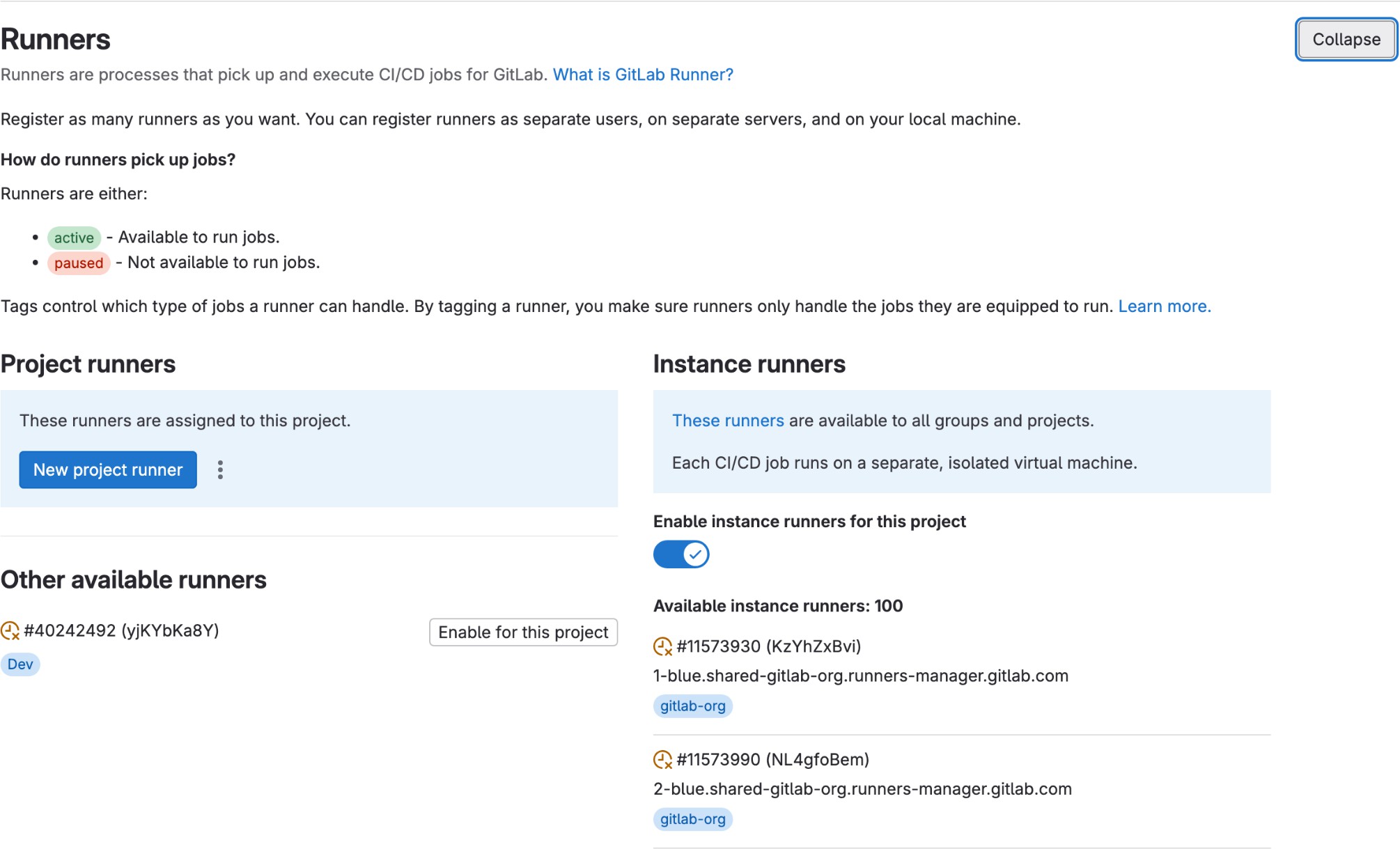
**===============**

# Step-4 Gitlab Runners

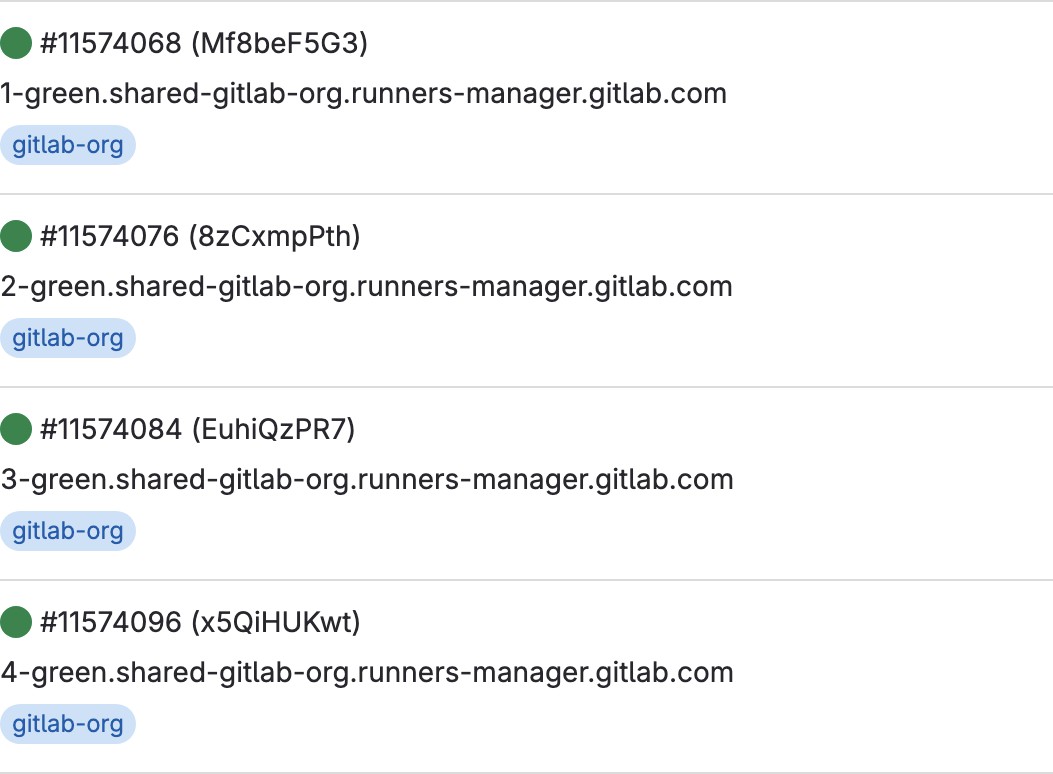
Click on Setting—> CICD —> Runners—> Click on Expand

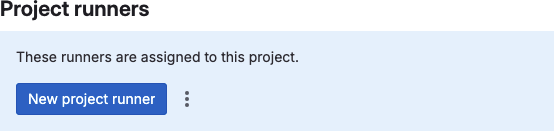


You can Create your Own runner or you can use “Instance Runners” provided by Gitlab for free.

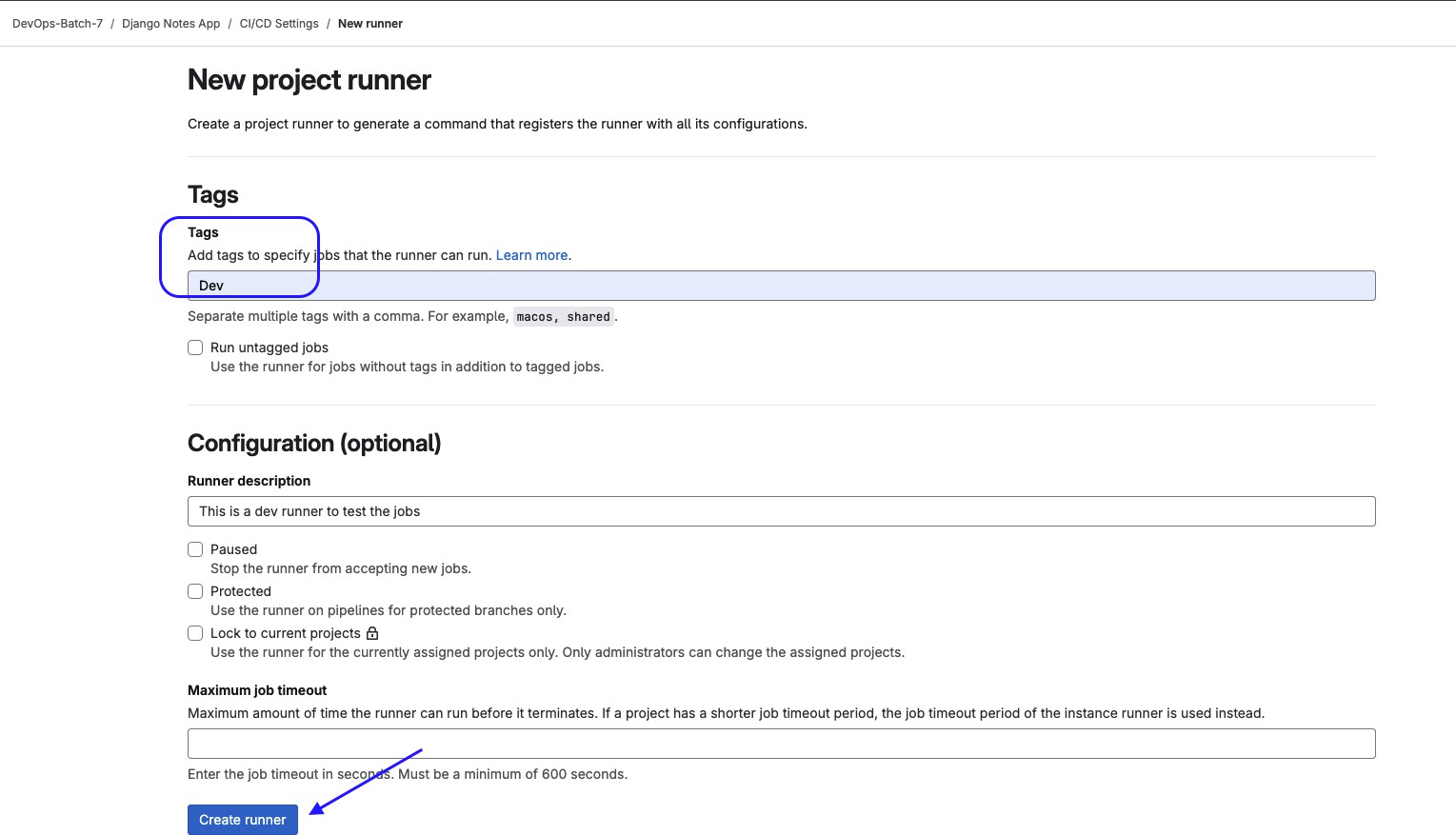


These are the shared Gitlab Provided runners.

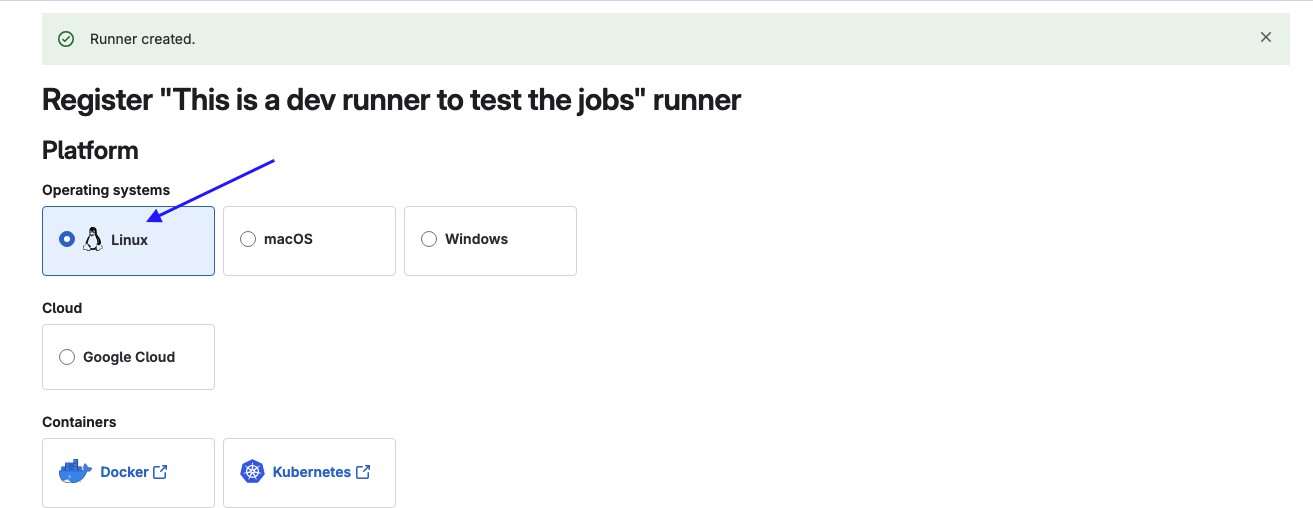


Let’s add the “New ProjectRunner”

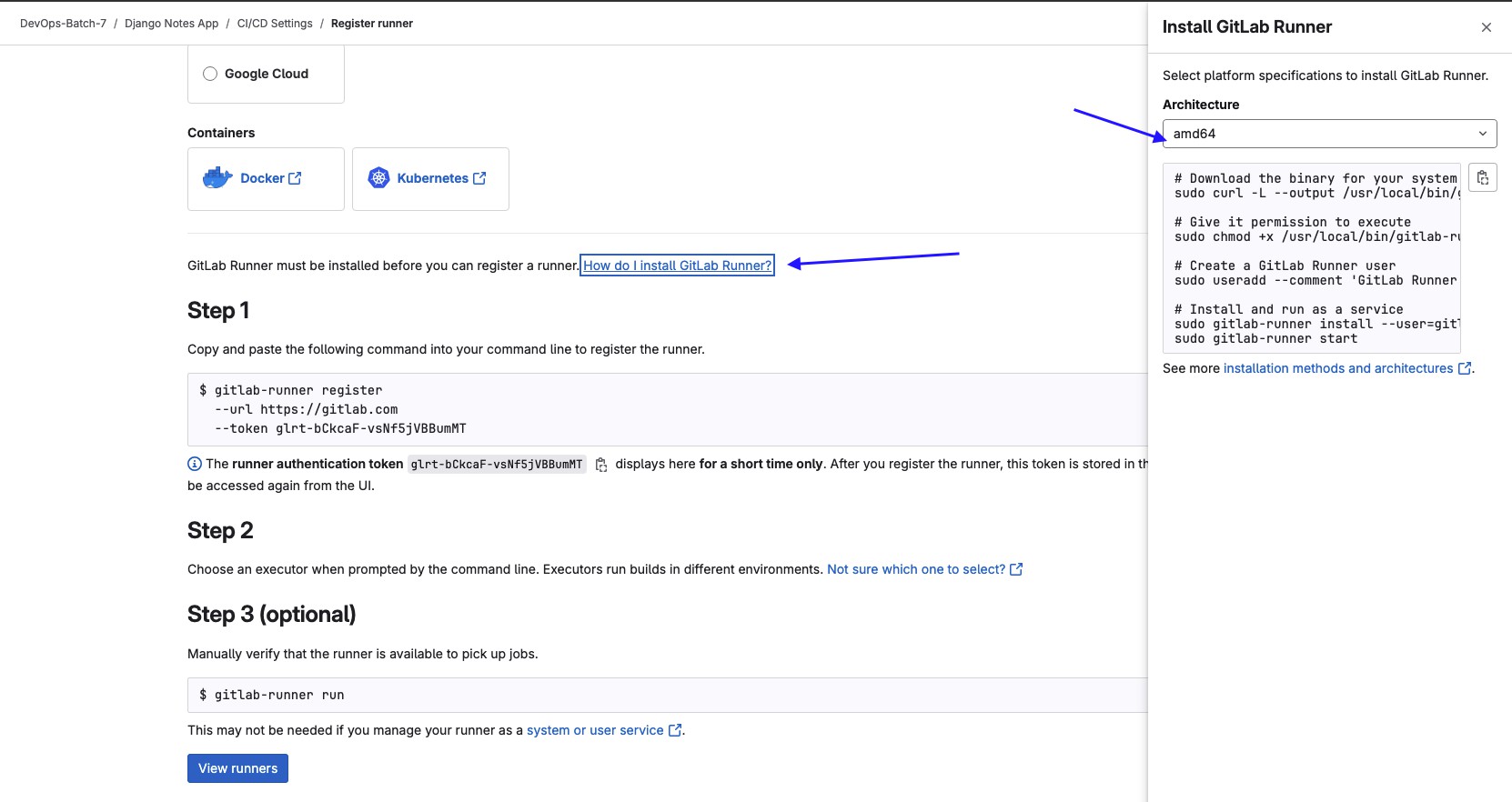
**# Tag :** Value which we will define in the Pipeline to instruct which runner to use.



Note : We can use different platforms to run runners, but I will be using Linux Platform.



# Execute the below steps on Linux EC2 Instance to register your Gitlab Runner. Click on **“How do i install Gitlab runner”**



**===============**

# Install the Gitlab Runner

# Run the below command on an EC2 Linux instance to install the Gitlab Runner.

# Download the binary for your system

sudo curl -L --output /usr/local/bin/gitlab-Runner

https://gitlab-Runner-Downloads.s3.amazonaws.com/latest/binaries/gitlab-Runner-linux-amd64

# Give it permission to execute

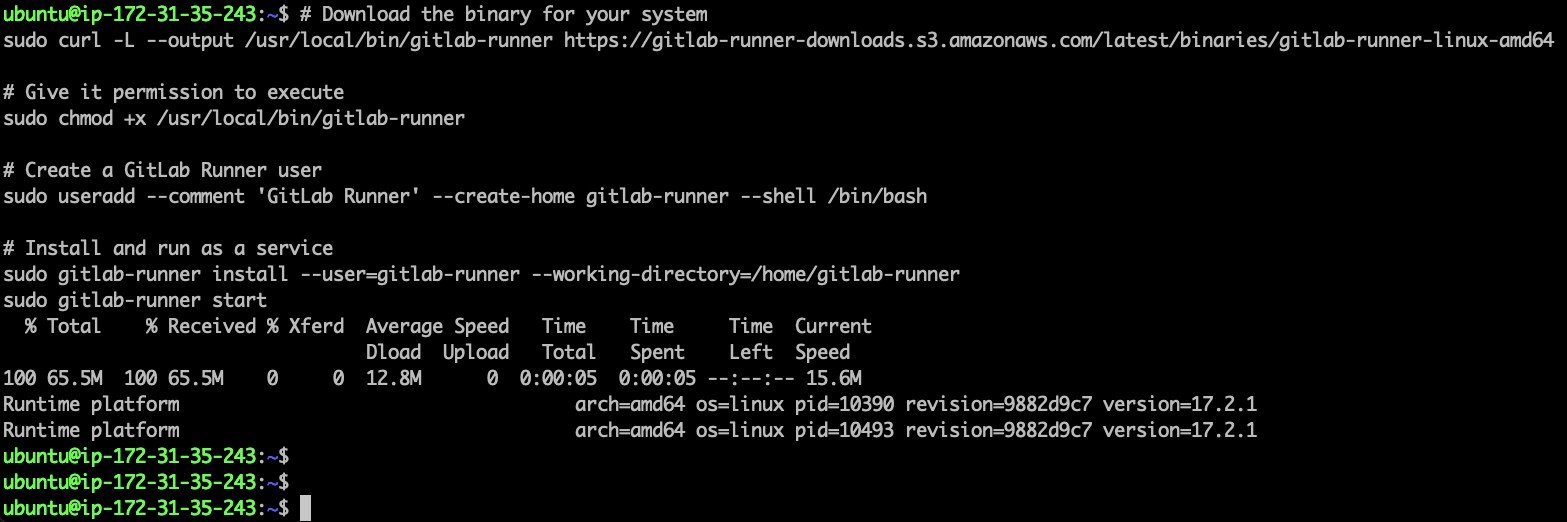
sudo chmod +x /usr/local/bin/gitlab-Runner

# Createau Gitlab Runner User

sudo Useradd --comment 'Gitlab Runner' --create-home gitlab-Runner --shell /bin/bash

# Install and run as u service

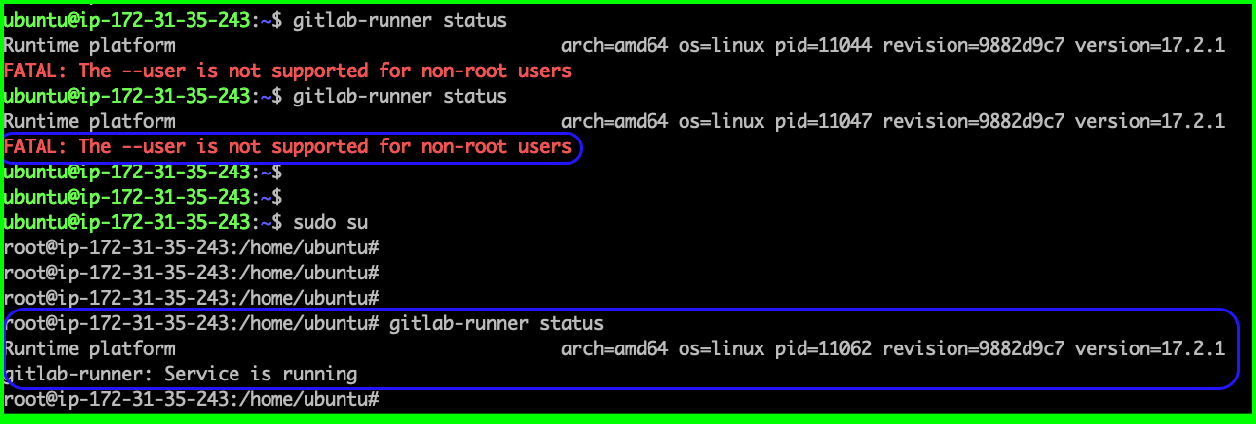
sudo gitlab-Runner install --User=gitlab-Runner --working-directory=/home/gitlab-Runner sudo gitlab-Runner start



**===============**

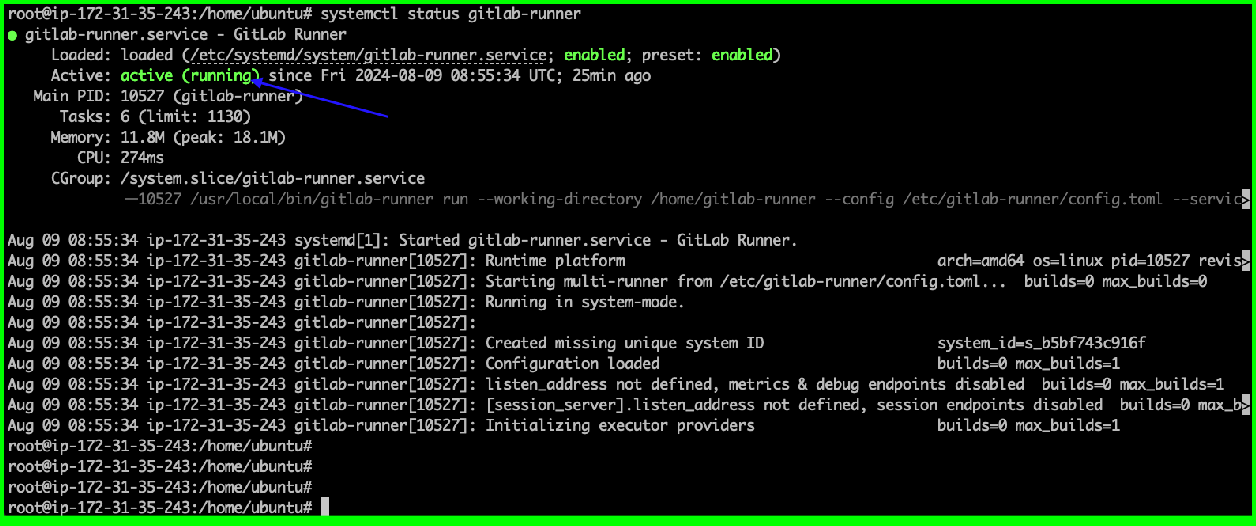
# Gitlab runner status

To Check the gitlab Runner status, you have to be a root User else it will give you below warning.



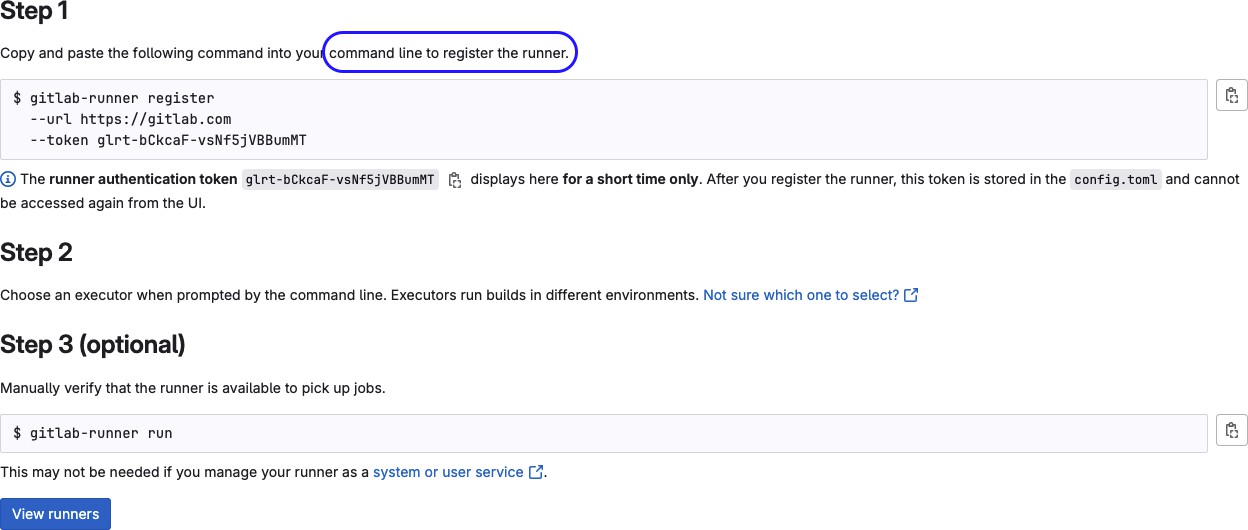
**===============**

# Gitlab runner Service status

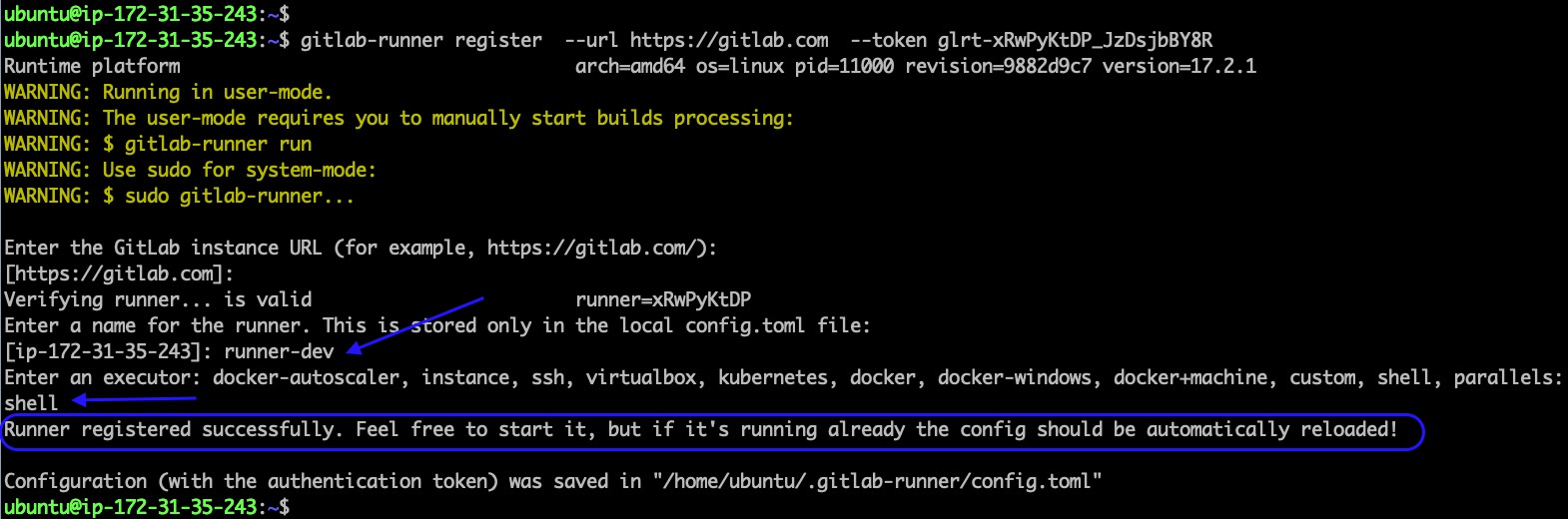
****

**===============**

# Register the Gitlab Runner

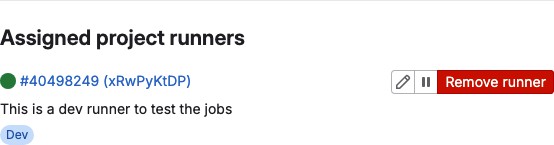


# Run Step-1 Command to register the Runner with your Project.



# After Step-1 Command, Runner registered with the Project.

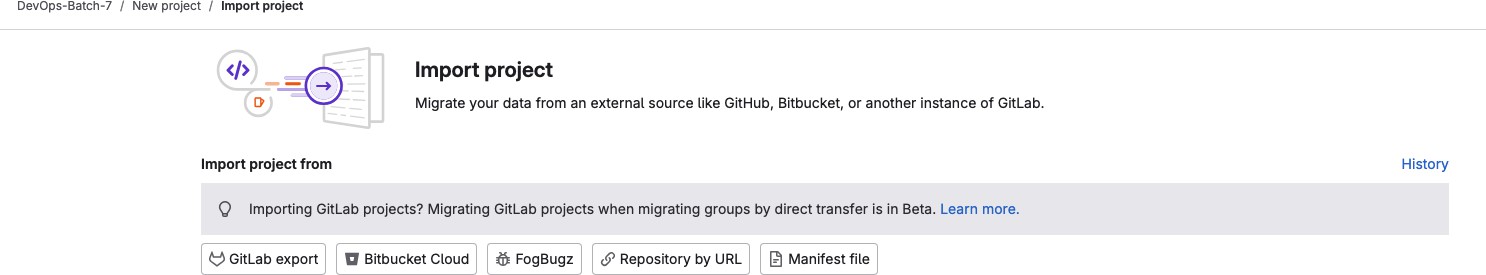
Click on View Runner, here you can see newly created Runner with Tag Name : Dev



**===============**

# Step-5: Project Import from

Here are the Options to import your projects into Gitlab.

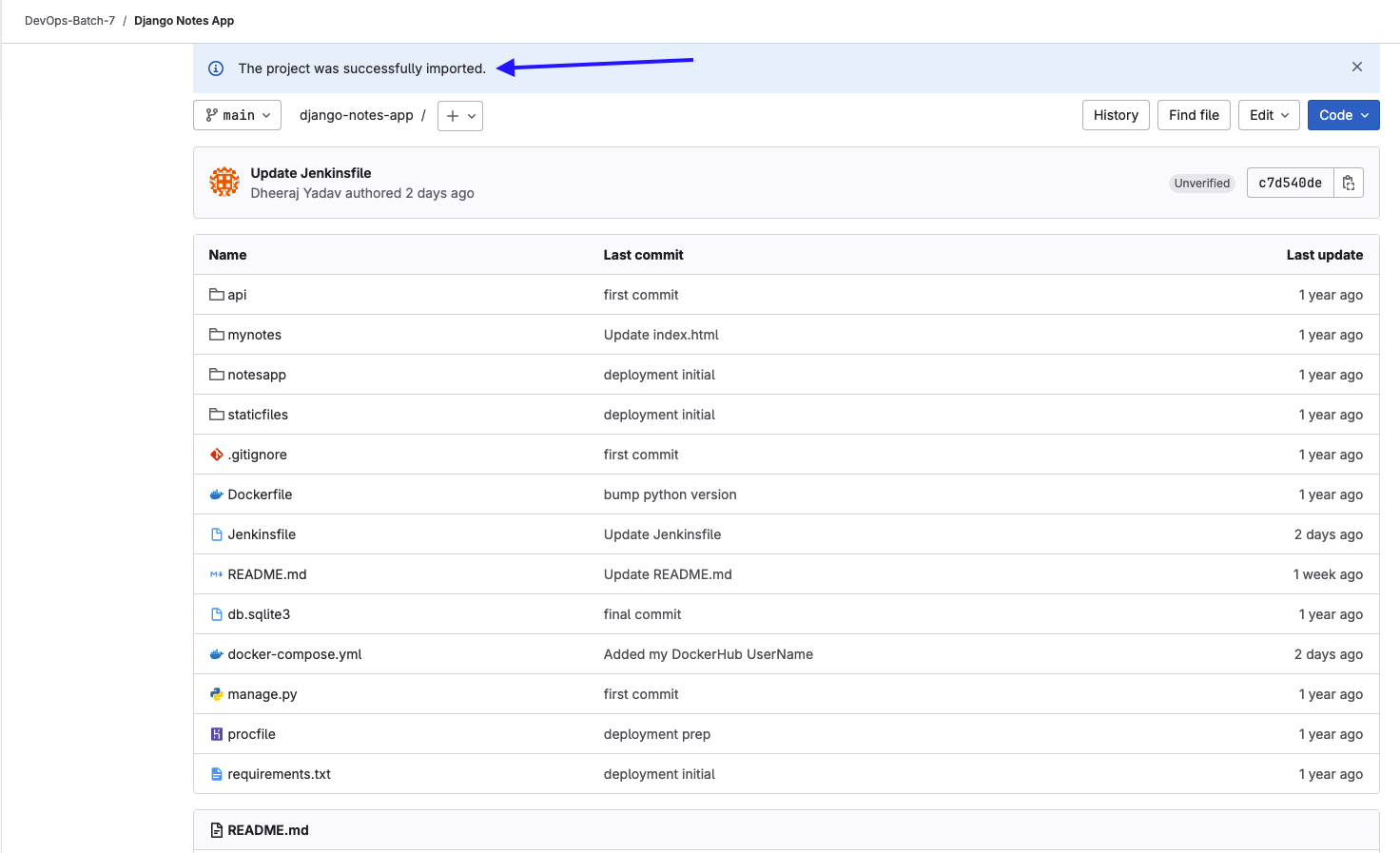


We will be using **Repository by URL** Option which is imported from GitHub.

**Mirror Repository :** Check this box hence no need for Manual Update, it will be Mirroring the changes from Source “**GitHub**”



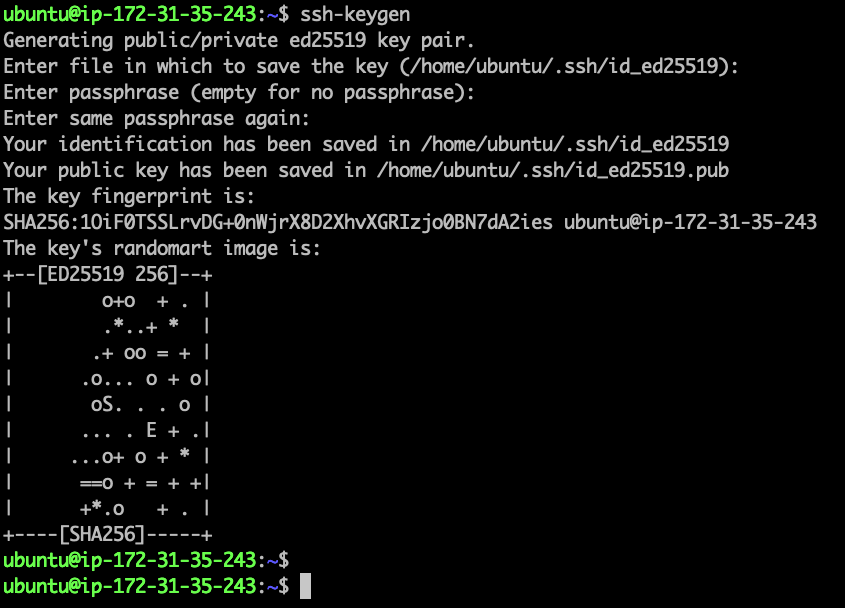
# Project has been imported successfully.



**===============**

# SSH Key-Gen on EC2 Machine

**3 ssh-keygen** : Use this command to generate the Private & Public Key Pair on EC2 Instance.



# Copy the Public key from EC2 Instance to gitlab Account

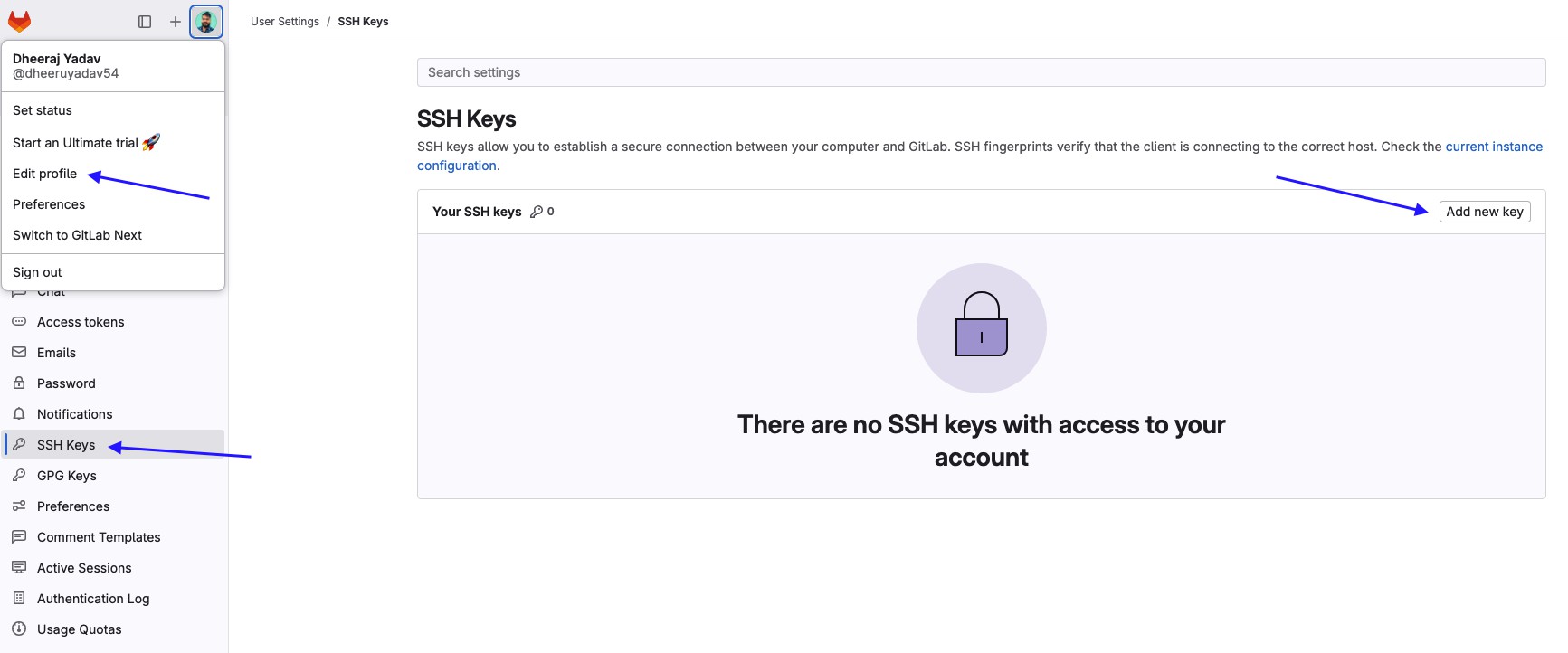
# cat /home/ubuntu/.ssh/id\_ed255fi9.pub



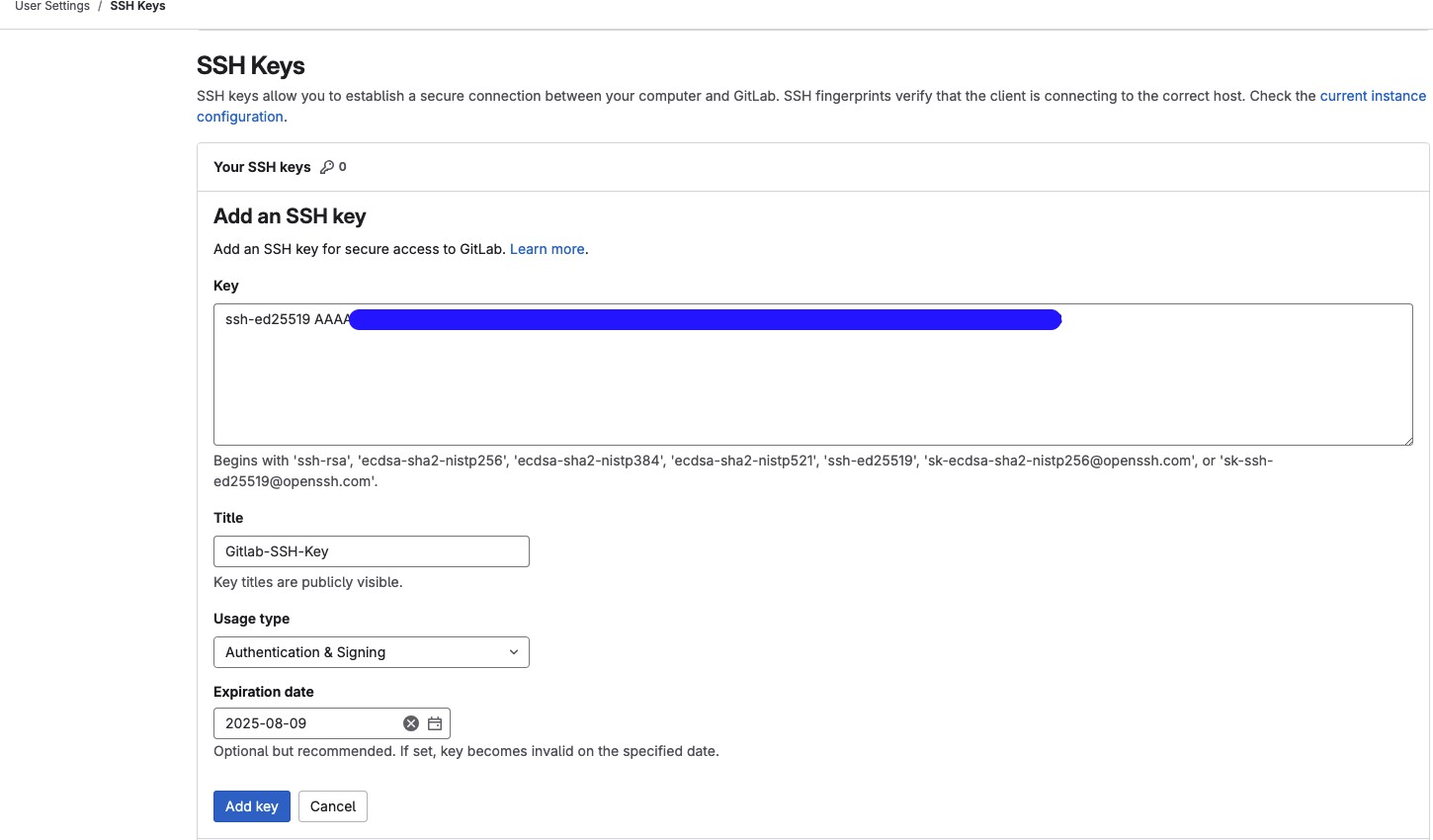
**===============**

# Add the Public Key to Gitlab Account

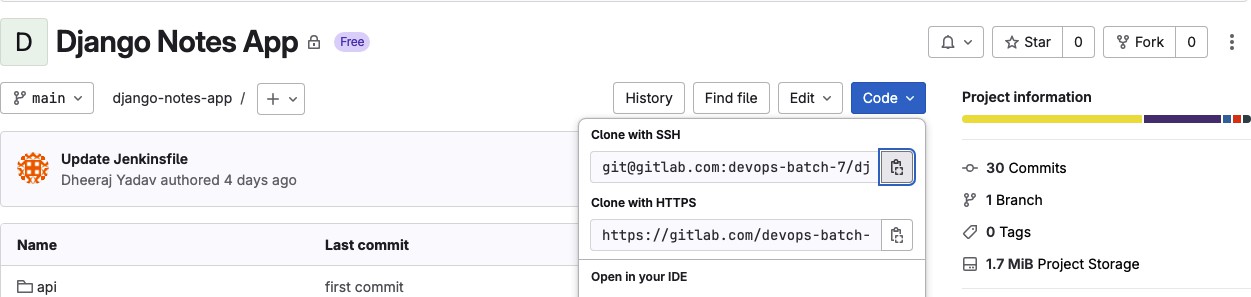
**Click on your profile icon —-> Edit Profile —-> SSH Keys —-> Add new key**



Paste the Copied Public key here & hit add key.



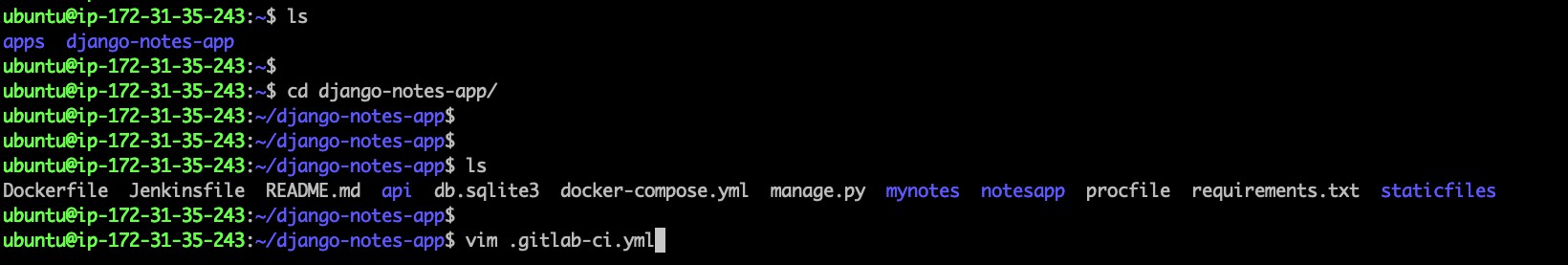
# 3 git clone [git@gitlab.com](mailto:git@gitlab.com):devops-batch-7/django-notes-app.git : Use SSH URL



# After adding the Public key into gitlab, we can clone our project locally on EC2 instance.



Let’s Create a .gitlab-ci.yml file



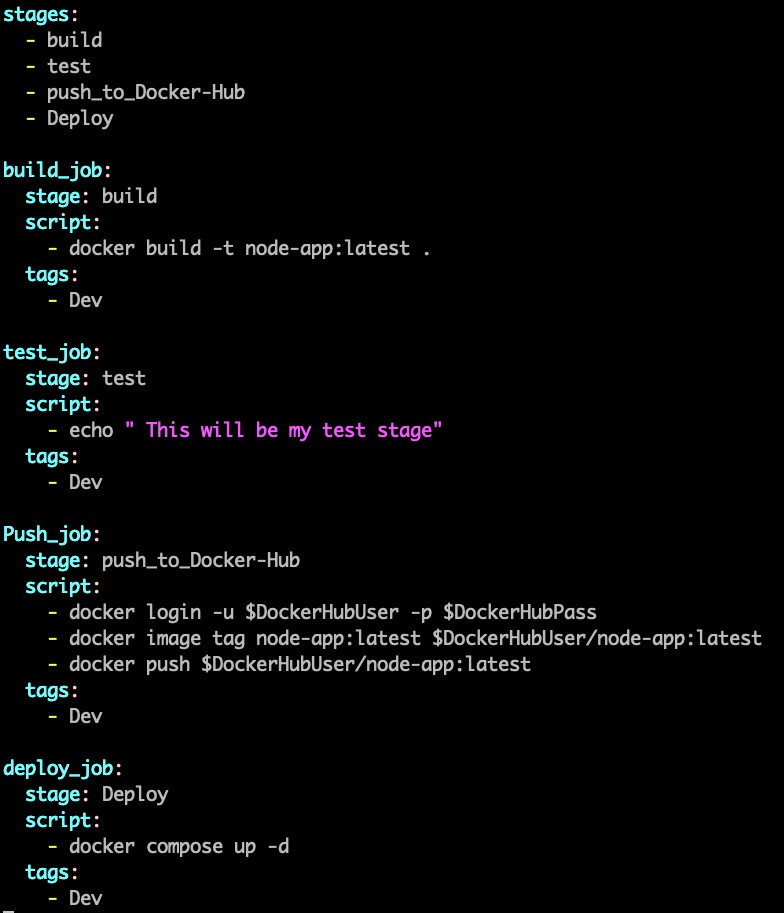
# Added .gitlab-ci.yml file with the four stages with Runner tug: Dev

# Prerequisites

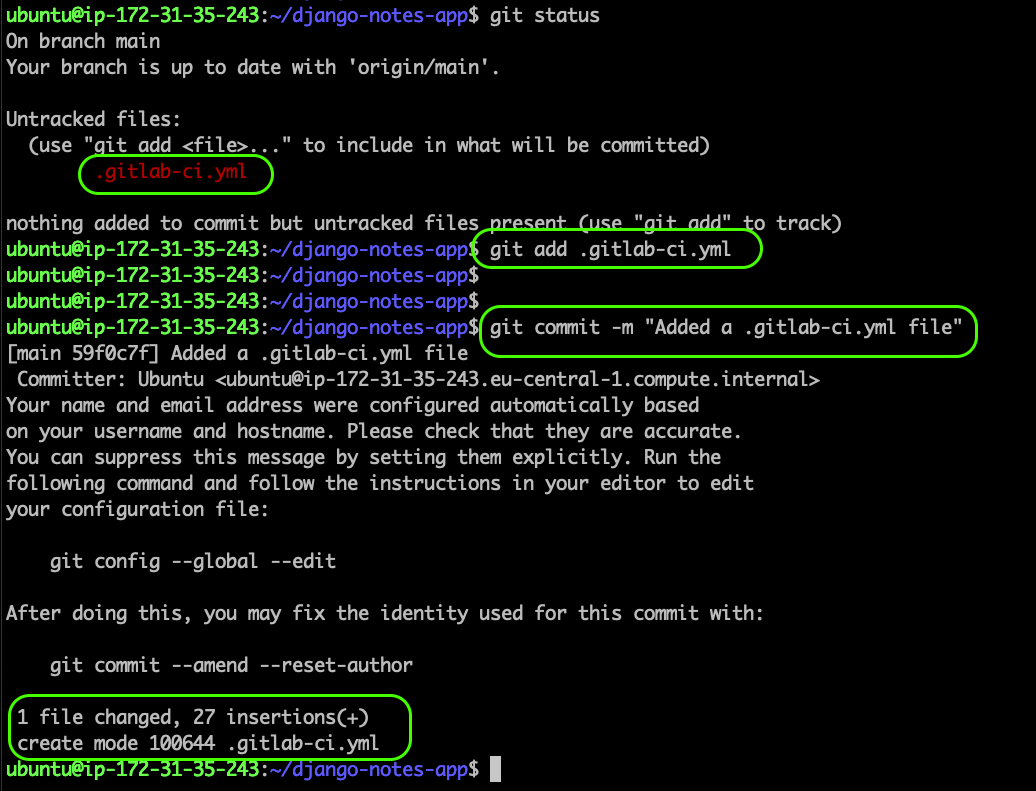
- docker & compose command should have installed

**===============**

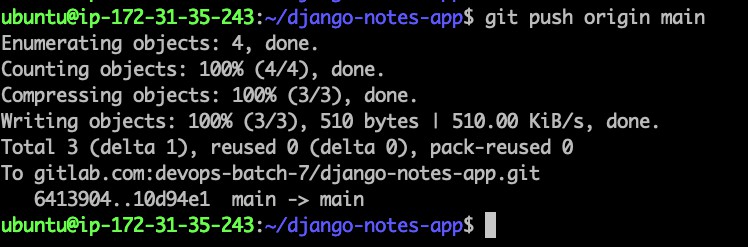
# Step-6 Run the Pipeline

****

# Pushed the ncw chungcs to gitlab & mudc u ncw commit.



# Let’s push the code to Origin , in case you have made the changes on gitlab directly , “git push will fail” you need to do “git pull origin –rebase”

****

**===============**

# Things to keep in Mind

* Your gitlab-Runner User should be a Part of Docker Group

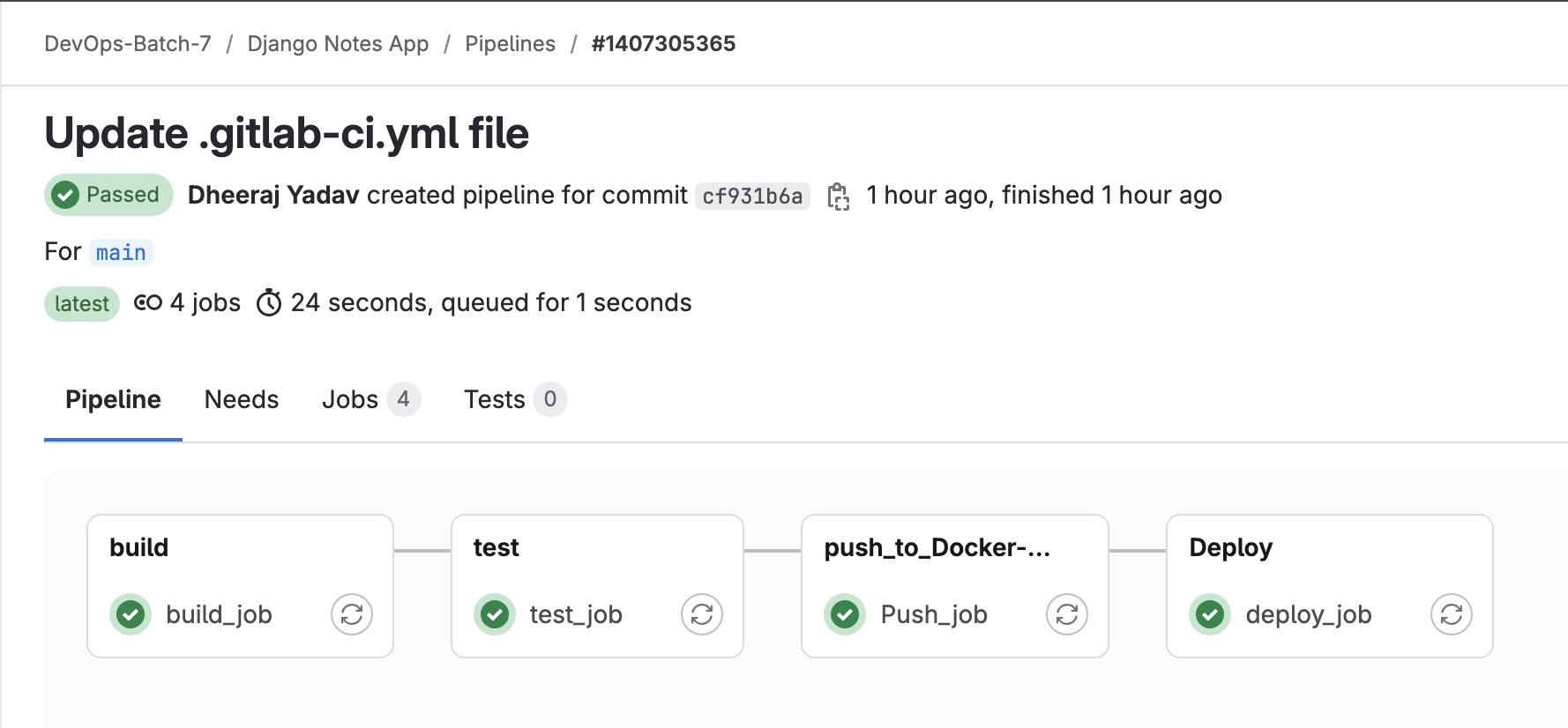
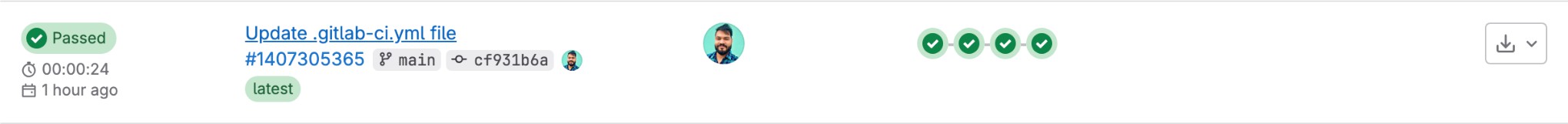


* Runner should be running & enabled for your Project.

**===============**

# Successful Pipeline:

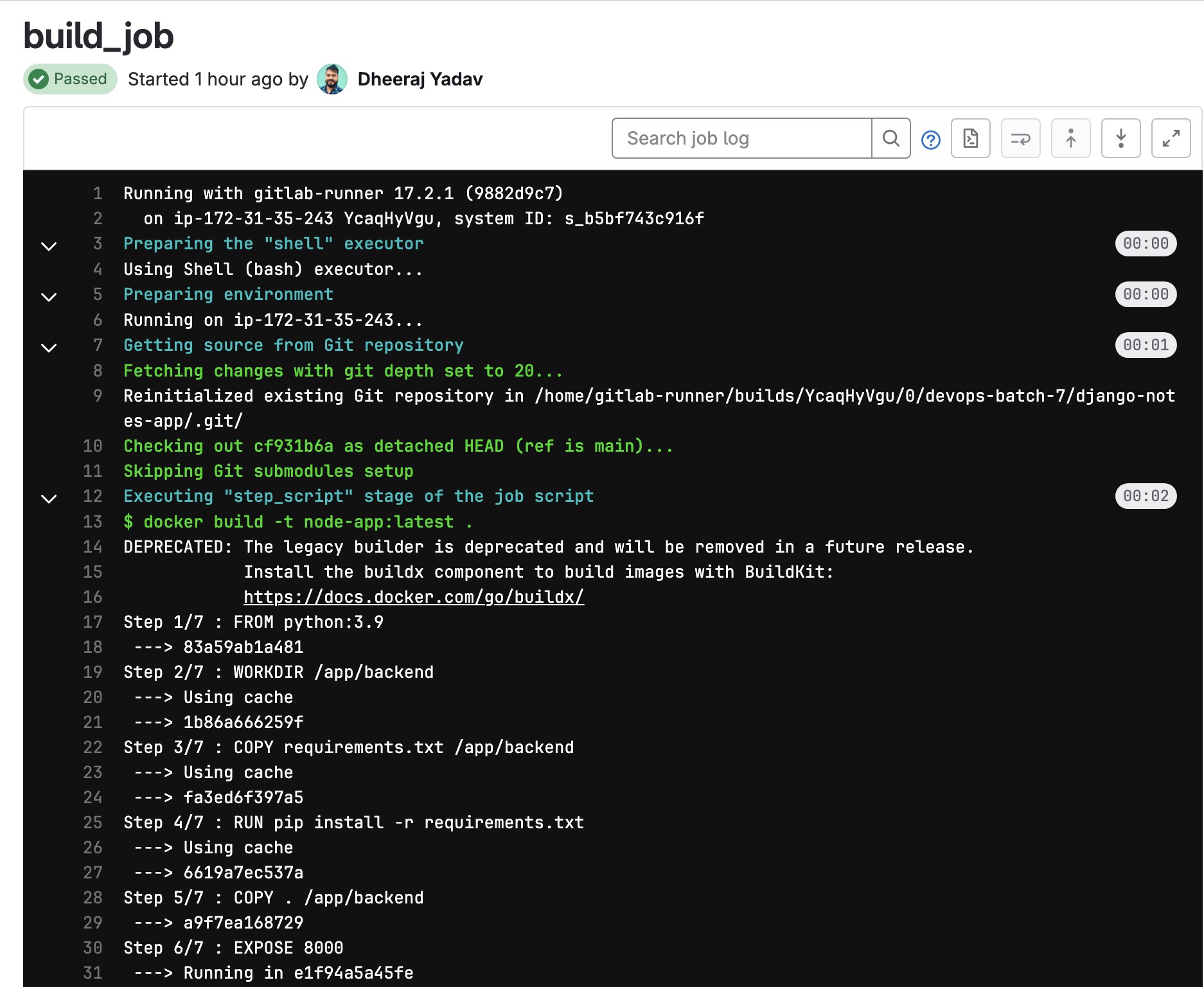
Pipeline has 4 stages & we have successfully uploaded the Docker image to DockerHub & deployed it.



**===============**

**Build Job:**

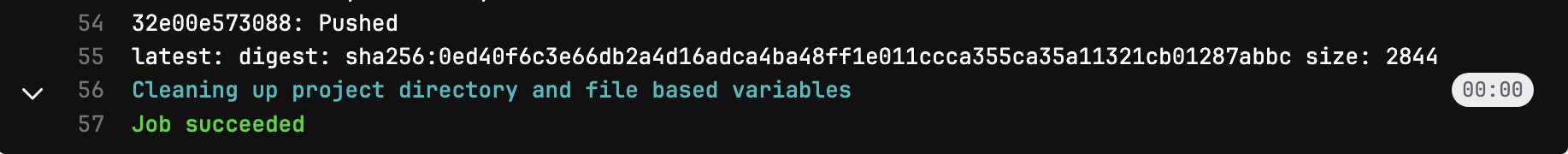
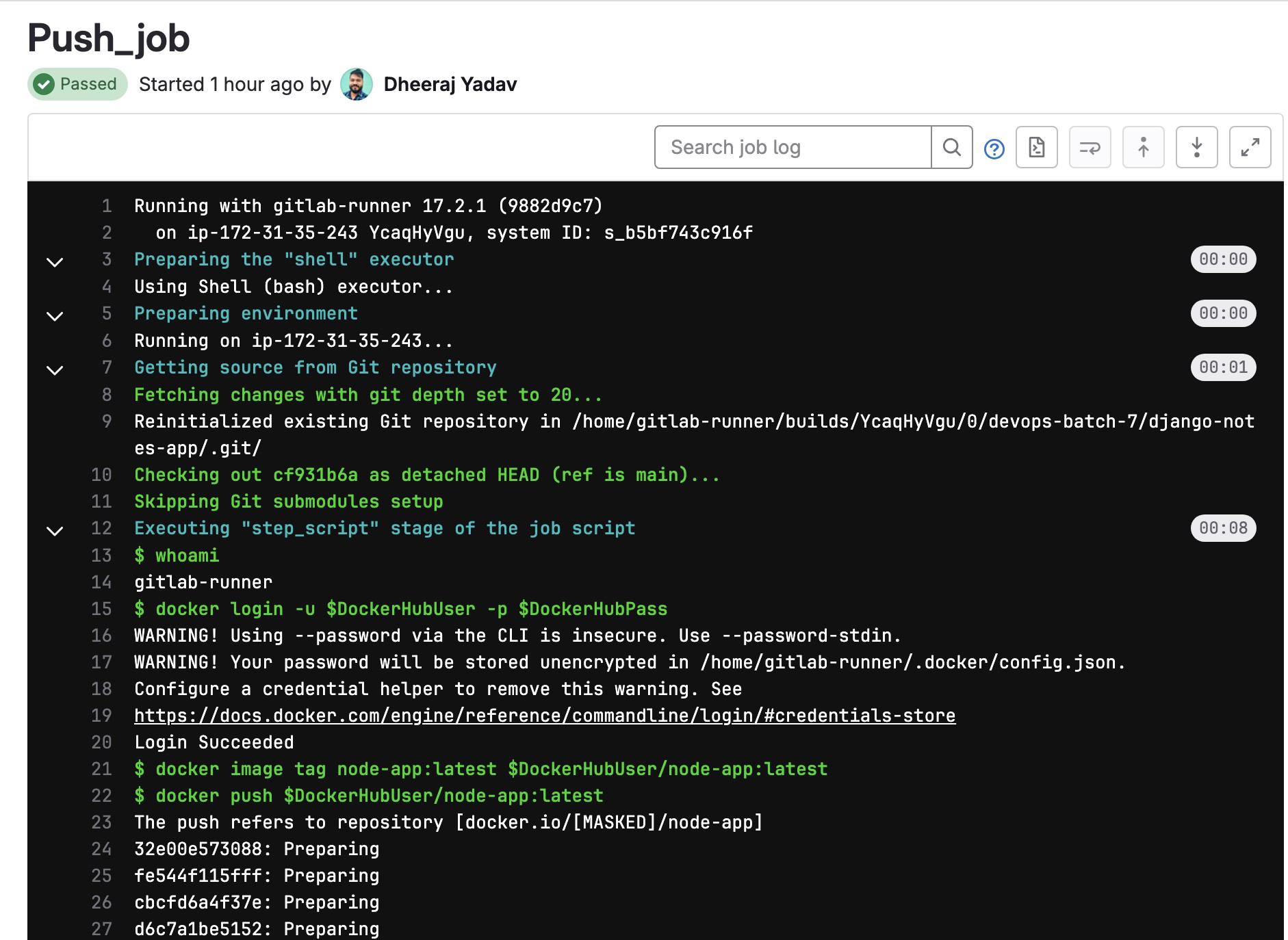
**3 docker build -t node-app:latest .** : Run this Docker command to build the docker image.



**===============**

**Push Job:**

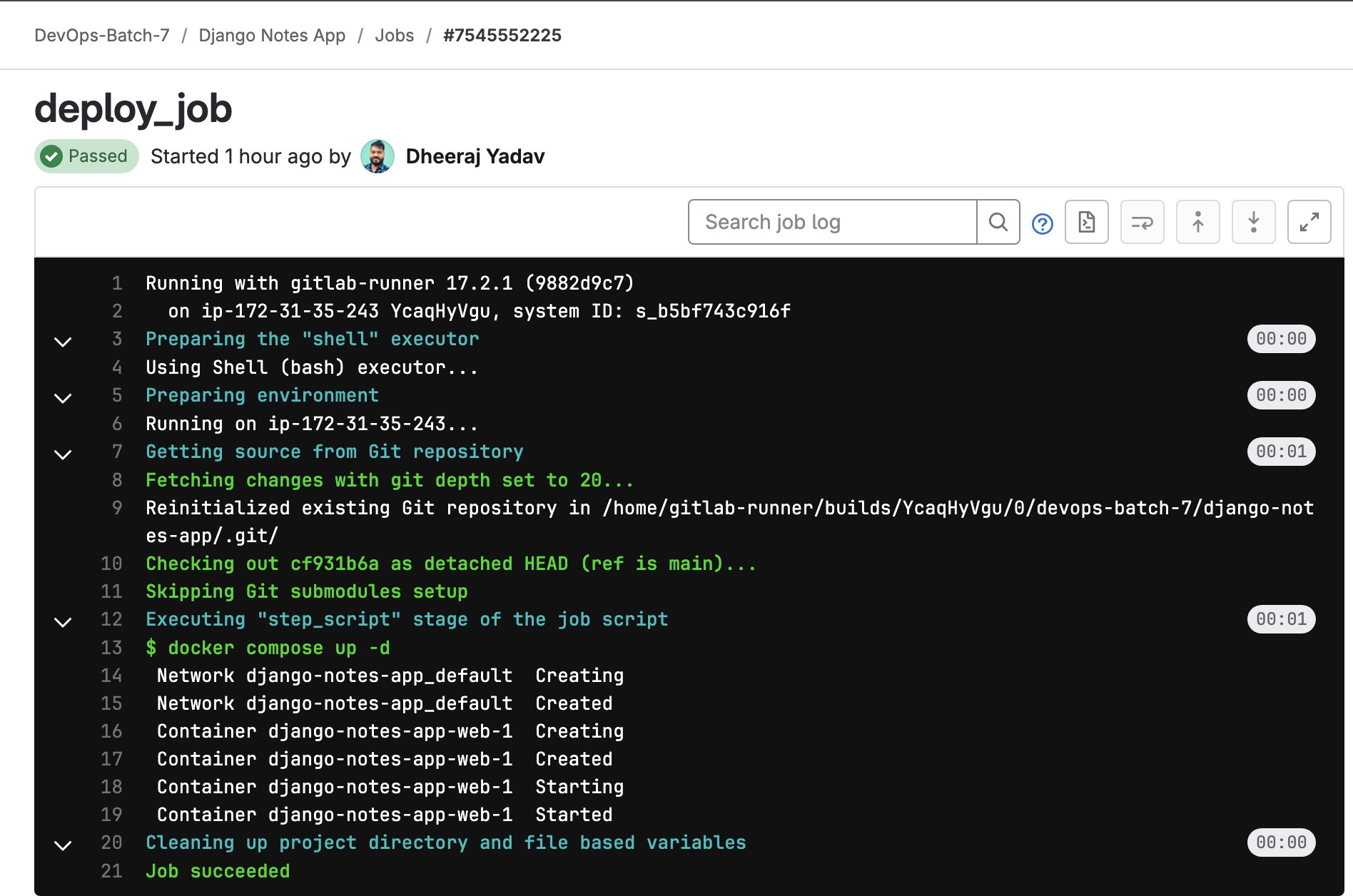
# Pushed the build image to DockerHub by using the User Defined Variables in the gitlab.



**===============**

**Deploy Job:**

# Pushed the image to DockerHub.

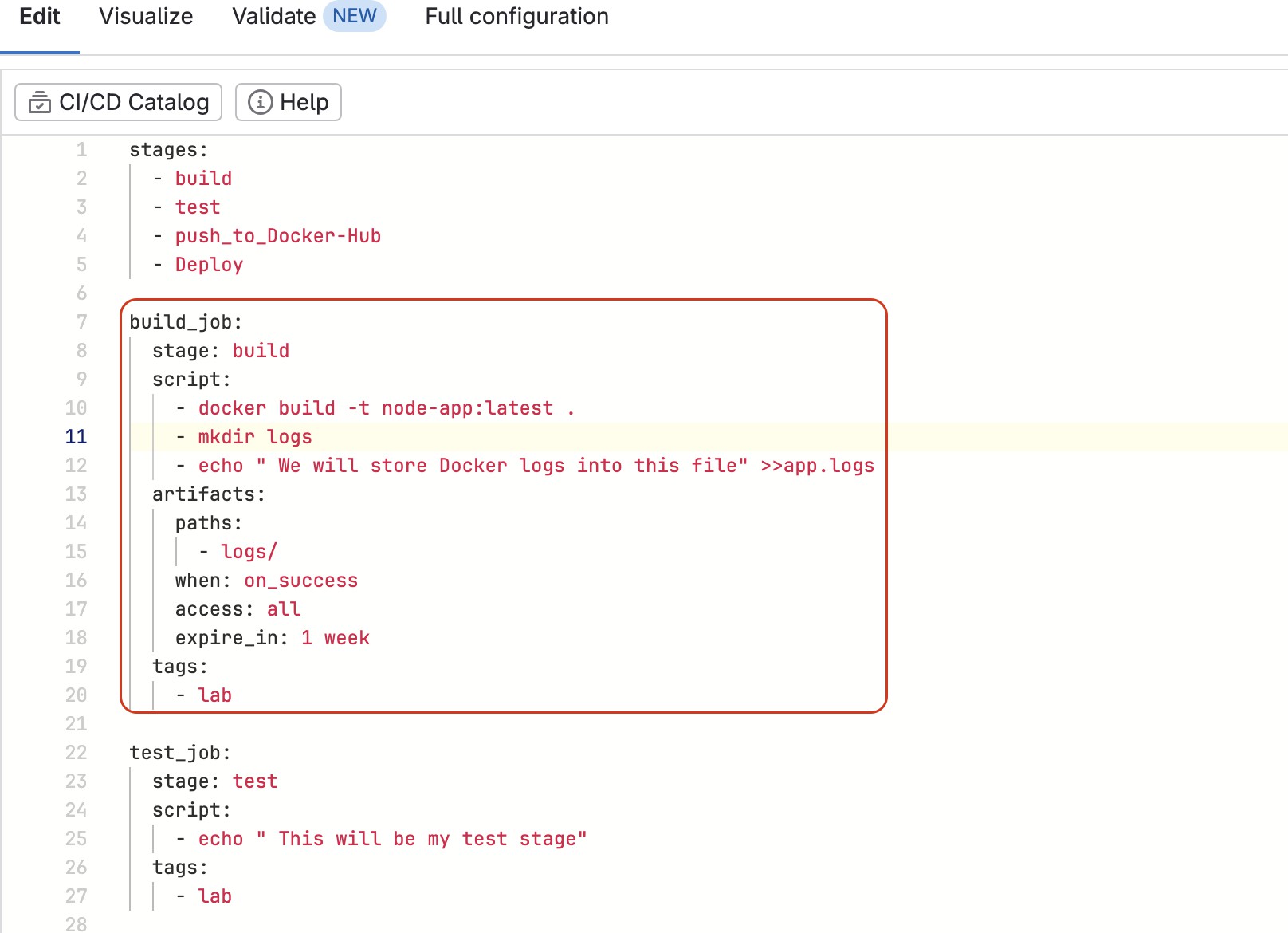


**===============**

# Step-7 Artifacts

# We will be Storing the Artifacts for 1 week, Added the output into Artifacts Logs Folder.

# Pipeline Config with Artifacts



Pipcline was succcssful & we can Download the Artifacts file on the right side.

