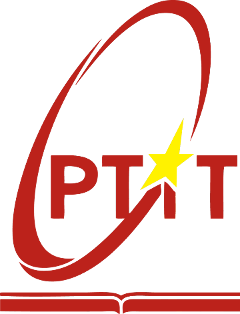
**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 2**

-----🙞🙜🕮🙞🙜-----

****

**MÔN: CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM**

**ĐỀ TÀI:**

***Giảng viên hướng dẫn: ThS. Lê Hà Thanh***

***Họ và tên: Đặng Đức Trọng***

***Mã số sinh viên: N20DCCN079***

***Lớp: D20CQCNPM01-N***

**Thành phố Hồ Chí Minh, 21/06/202****4**

Mục lục

[I. Lập kế hoạch và đánh giá 3](#_Toc169869272)

[II. Cấu hình 3](#_Toc169869273)

[1) Phần mềm cần cài 3](#_Toc169869274)

[2) Cấu hình địa chỉ ip 3](#_Toc169869275)

[3) Script và thông số thiết lập CI/CD 4](#_Toc169869276)

[a) File docker-compose.yml 4](#_Toc169869277)

[b) File .gitlab-ci.yml 6](#_Toc169869278)

[4) Liên kết tới repo chứa code sites, tài liệu, scripts 9](#_Toc169869279)

[III. Triển khai 9](#_Toc169869280)

[1) Các bước triển khai 9](#_Toc169869281)

[2) Sơ đồ hoạt động 10](#_Toc169869282)

[3) Bước 1: Tạo server gitlab và đẩy code từ local lên repository của gitlab 11](#_Toc169869283)

[4) Bước 2: Tạo server gitlab-runner-registry, cài đặt Gitlab-runner kết nối với Gitlab và Registry 18](#_Toc169869284)

[5) Bước 3: Cấu hình CICD pipeline trên gitlab và trong source 25](#_Toc169869285)

[6) Bước 4: Tạo server deploy 28](#_Toc169869286)

# Lập kế hoạch và đánh giá

* Sau khi triển khai website hoặc ứng dụng hoàn tất, thu hút được nhiều khách hàng sử dụng.
* Đặt vấn đề: Vậy mỗi lần phát triển phần mềm (tạo phiên bản mới) để thay phiên bản cũ thì website sẽ bị đóng không cho khách hàng sử dụng -> hao tổn doanh thu
* Giải pháp: CICD Pipeline sẽ giải quyết vấn đề này trong mỗi dự án. Có thể hiểu đơn giản là công việc này sẽ khiến cho những phiên bản mới được cập nhật mà phiên bản cũ vẫn chạy khi hoàn tất phiên bản mới lên cũng không hề có khoảng thời gian website bị downtime vì vậy sẽ hạn chế tối đa việc ảnh hưởng đến người dùng, không những giúp chúng ta giảm thiểu được thời gian triển khai code lên server (không còn phải triển khai thủ công bằng tay mà có triển khai thì chục nghìn vài chục nghìn service thì triển khai chay sao nổi), các job test, mà còn làm tăng hiệu quả công việc

# Cấu hình

## Phần mềm cần cài

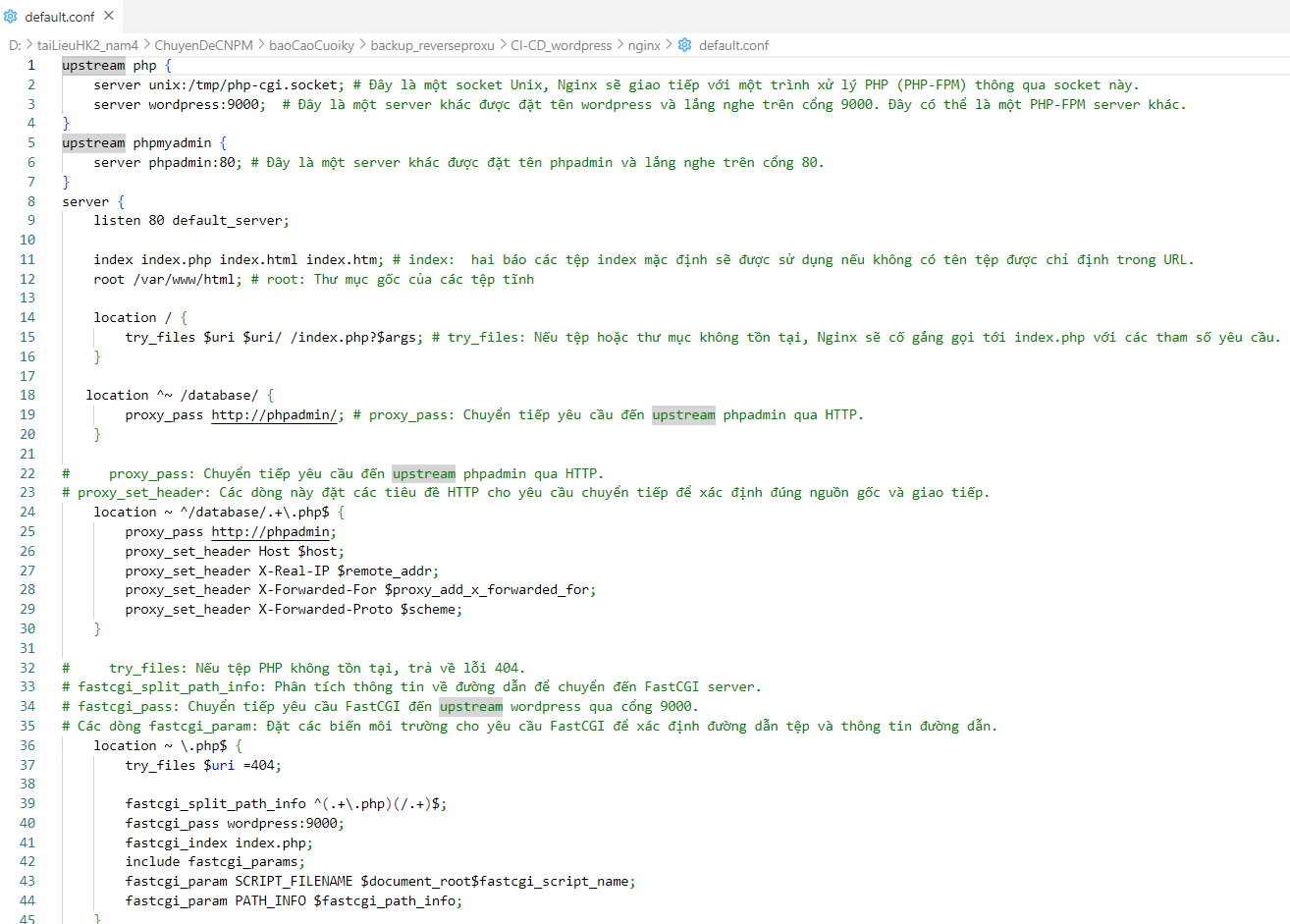
* Cài đặt các phần mềm hỗ trợ tạo máy ảo ubuntu trên windows 11:
* Vagrant
* VirtualBox

## Cấu hình địa chỉ ip

- Tham số thiết lập server: 3 server ubuntu

* Server gitlab: có địa chỉ ip là 10.32.4.111
* Server gitlab-runner-registry: có địa chỉ ip là 10.32.4.125
* Server svdeploy: có địa chỉ ip là 10.32.3.180

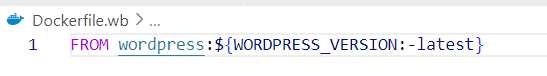
- Tham số thiết lập cấu hình Reverse Proxy



## Script và thông số thiết lập CI/CD

### File docker-compose.yml

* Tạo image wordpress:
  + Build: build image có sẵn trong docker từ file Dockerfile.wb
  + Container\_name: đặt tên container
  + Volumes: mapping không gian lưu trữ dữ liệu của container với folder trong source





* Tạo image mysql và phpmyadmin:

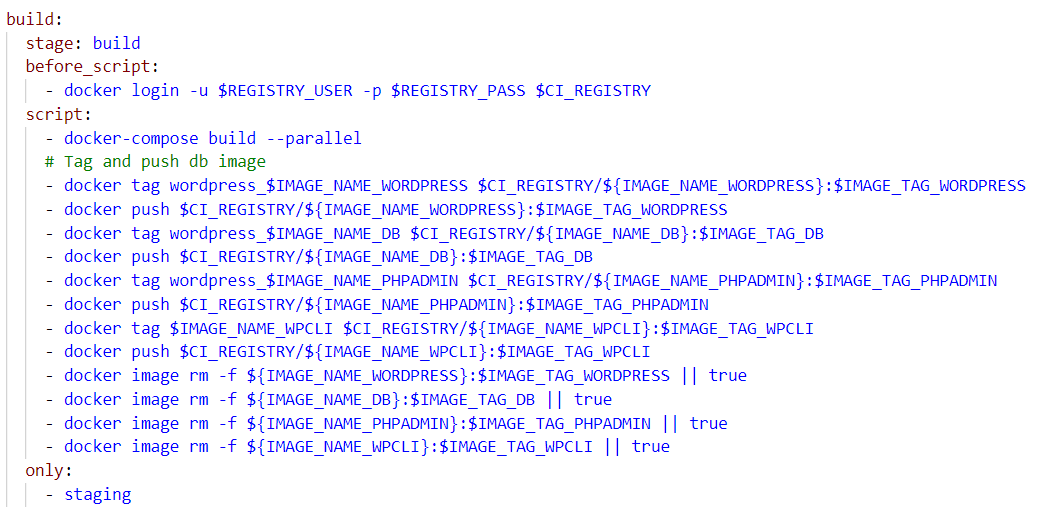


### File .gitlab-ci.yml

* Cấu hình các biến variables toàn cục (ở đây tên image và tag để đẩy image vào trong registry docker) dùng trong job build và deploy



* Job build: thực thi before\_script login vào registry và thực hiện các lệnh script
  + Docker-compose build –parallel: build các image trong file docker-compose
  + Docker tag: tạo nhãn tag cho image đã được build trên
  + Docker push: đẩy image vào repositories của registry
  + Docker image rm -f: remove các image đã được gán nhãn tag ở docker tag



* Job deploy:
  + Thực hiện shh user-deploy@SERVER\_TESTING: từ server gitlab-runner-registry gọi đến svdeploy (ssh: thông qua khóa private/public) để thực hiện các lệnh docker tạo container
  + Docker login: đăng nhập vào registry trên server gitlab-runner-registry
  + Docker pull: pull các image trong repository của registry ra từ job build đã image vào đó
  + Docker network create: tạo network kiểu bridge để các container giao tiếp được với nhau trong docker
  + Docker run: khởi tạo một container mới từ image, với parameter: -v (volumns), -d(detech), -e (environment)



## Liên kết tới repo chứa code sites, tài liệu, scripts

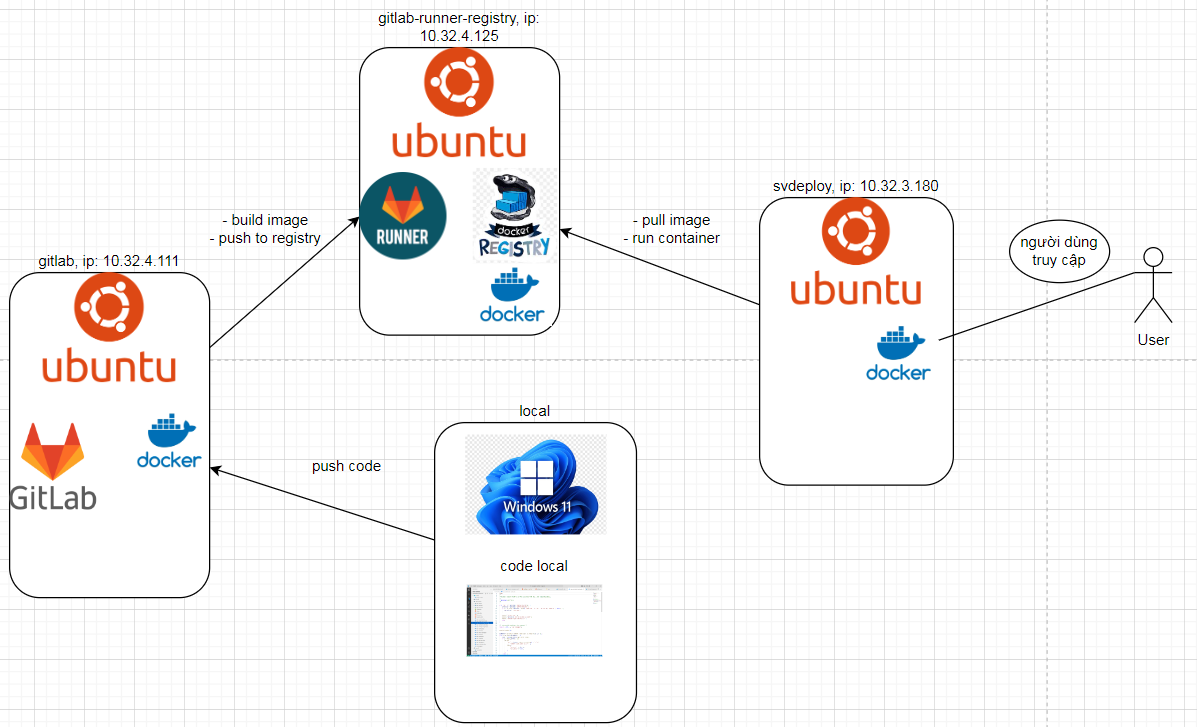
* Link github: https://github.com/ductrong6104/lab-ci-cd

# Triển khai

## Các bước triển khai

* Bước 1: Tạo server gitlab và đẩy code từ local lên repository của gitlab
  + Cài đặt gitlab bằng docker
  + Tạo dự án và đẩy code từ local lên gitlab
* Bước 2: Tạo server gitlab-runner-registry, cài đặt Gitlab-runner kết nối với Gitlab và Registry
  + Cài đặt gitlab-runner và kết nối gitlab-runner với gitlab
  + Cài đặt registry và cài đặt https cho registry
* Bước 3: Cấu hình CICD pipeline trên gitlab và trong source
  + Cấu hình .gitlab-ci.yml file để chạy CICD pipeline
  + Cấu hình các biến trong cài đặt CICD pipeline
* Bước 4: Tạo server deploy
  + Cài đặt kết nối ssh từ server gitlab-runner-registry đến server svdeploy
  + Cài đặt https từ server svdeploy đến gitlab-runner-registry

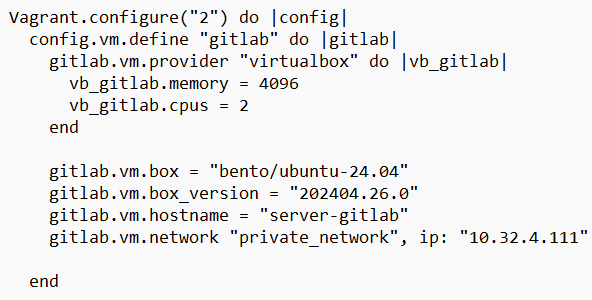
## Sơ đồ hoạt động



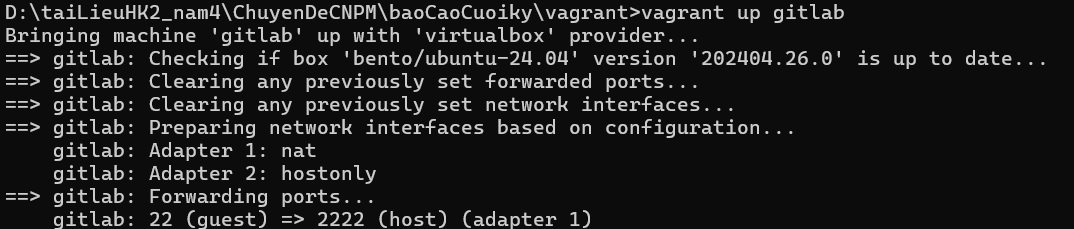
* Bạn có thể thấy ngay khi bạn push code, gitlab-runner sẽ chạy để kéo code về server, nó sẽ tiến hành chạy các job tiếp tục như build thành image và push nó lên registry (giai đoạn này là CI), tiếp tục server deploy sẽ tiến hành pull image nó và run lên container (giai đoạn này là CD) người dùng truy cập từ address mà bạn đã cấu hình.
* Vậy là chỉ duy nhất công việc của bạn là kiểm tra thật kỹ code của mình và nhấn push, hoặc merge request từ nhánh khác vào và CICD pipeline code sẽ tự động build, test, deploy.

## Bước 1: Tạo server gitlab và đẩy code từ local lên repository của gitlab

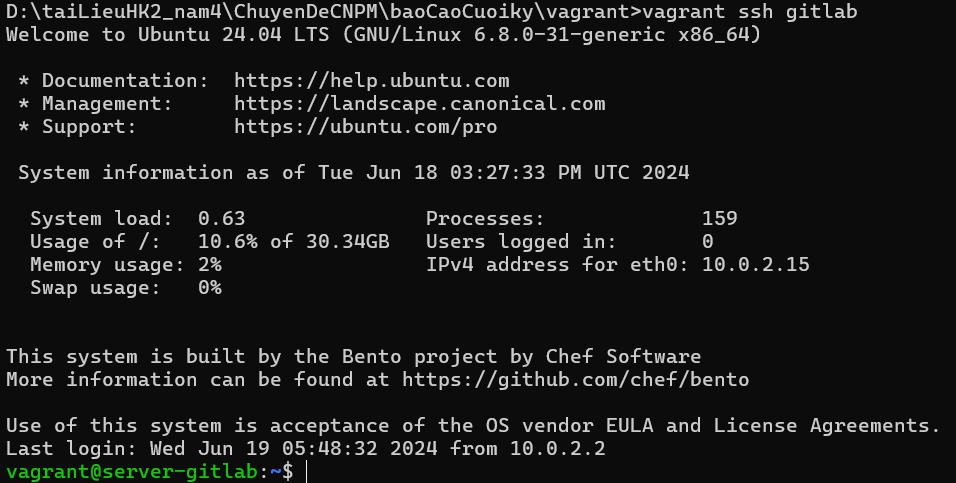
* Tạo file Vagrant ở folder bất kỳ trên máy local (windows) và cấu hình máy ảo ubuntu chạy server gitlab



* Mở terminal, cd tới folder chứa file Vagrant chạy lần lượt các lệnh:
  + Vagrant up gitlab: tạo máy ảo ubuntu



* + Vagrant ssh gitlab: truy cập vào máy ảo ubuntu



* Tiến hành cài đặt gitlab từ docker, trước tiên dùng các lệnh để cài docker trên ubuntu. **Việc cài đặt docker sẽ thực hiện ở cả ba server ubuntu**

$ apt-get update

$ apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

$add-apt-repository"deb[arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

$ apt-get update

$ apt-get install docker-ce docker-ce-cli containerd.io

$curl-L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

$ chmod +x /usr/local/bin/docker-compose

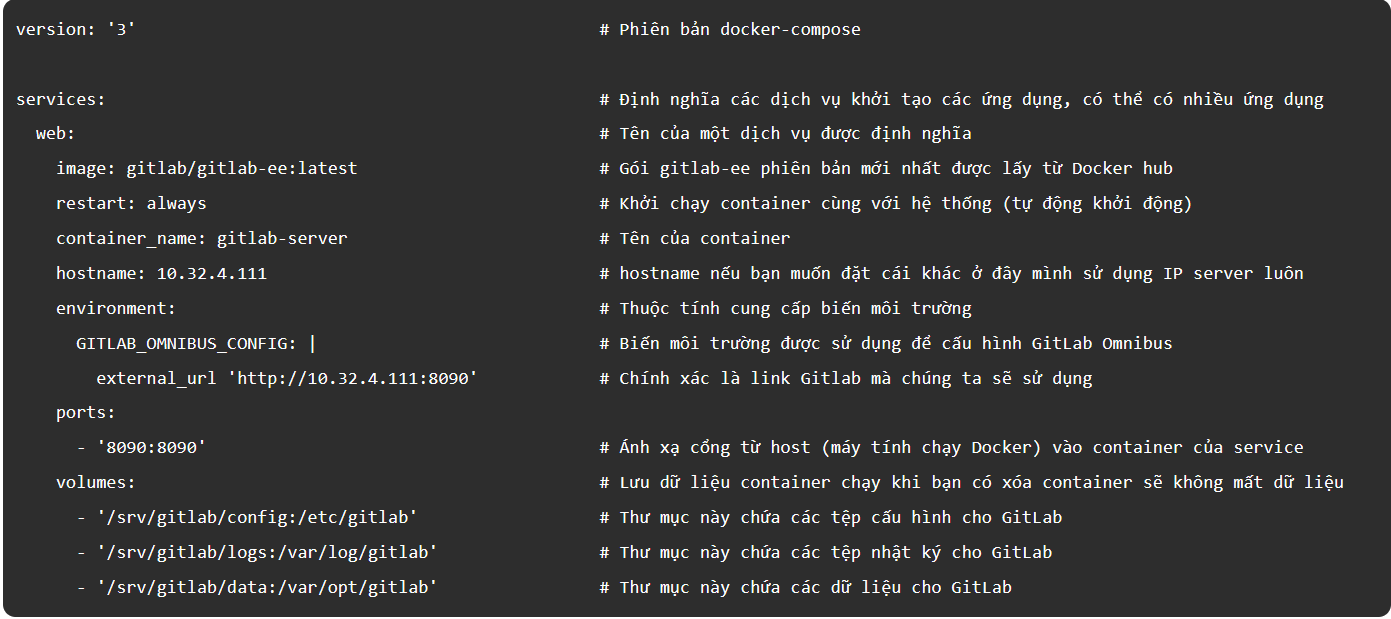
Cài đặt xong tiến hành thực hiện 2 lệnh sau và kết quản vậy là đã cài đặt docker và docker-compose thành công.

$ docker -v

$ docker-compose -v

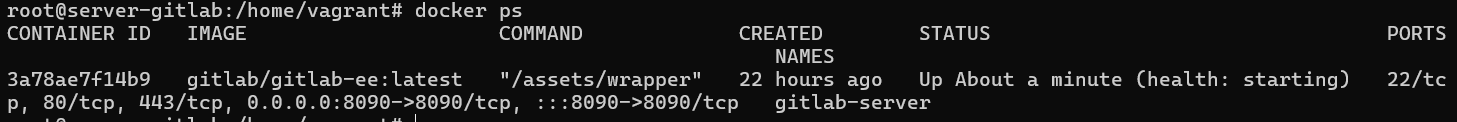
$ mkdir -p /home/docker-app/gitlab && cd /home/docker-app/gitlab tạo thư mục để lưu config triển khai gitlab và đi tới thư mục đó.

$ nano docker-compose.yml tạo file docker-compose để tiến hành build Gitlab và nội dung dưới đây



$ docker-compose up -d khởi chạy file docker-compose

$ docker ps xem kết quả của các container hiện đang chạy

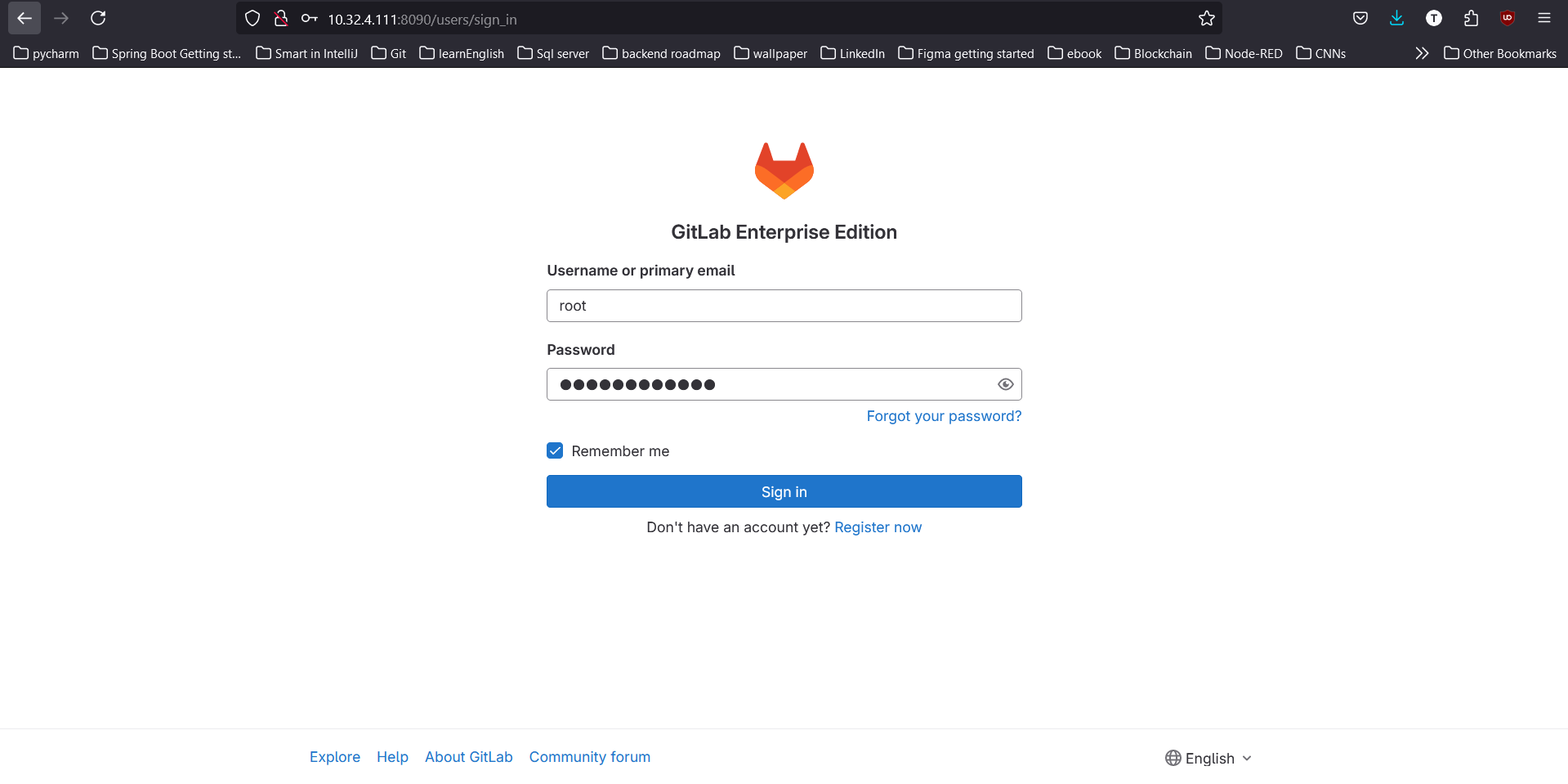


* Tiếp theo tiến hành đổi password tài khoản root cho dễ sử dụng

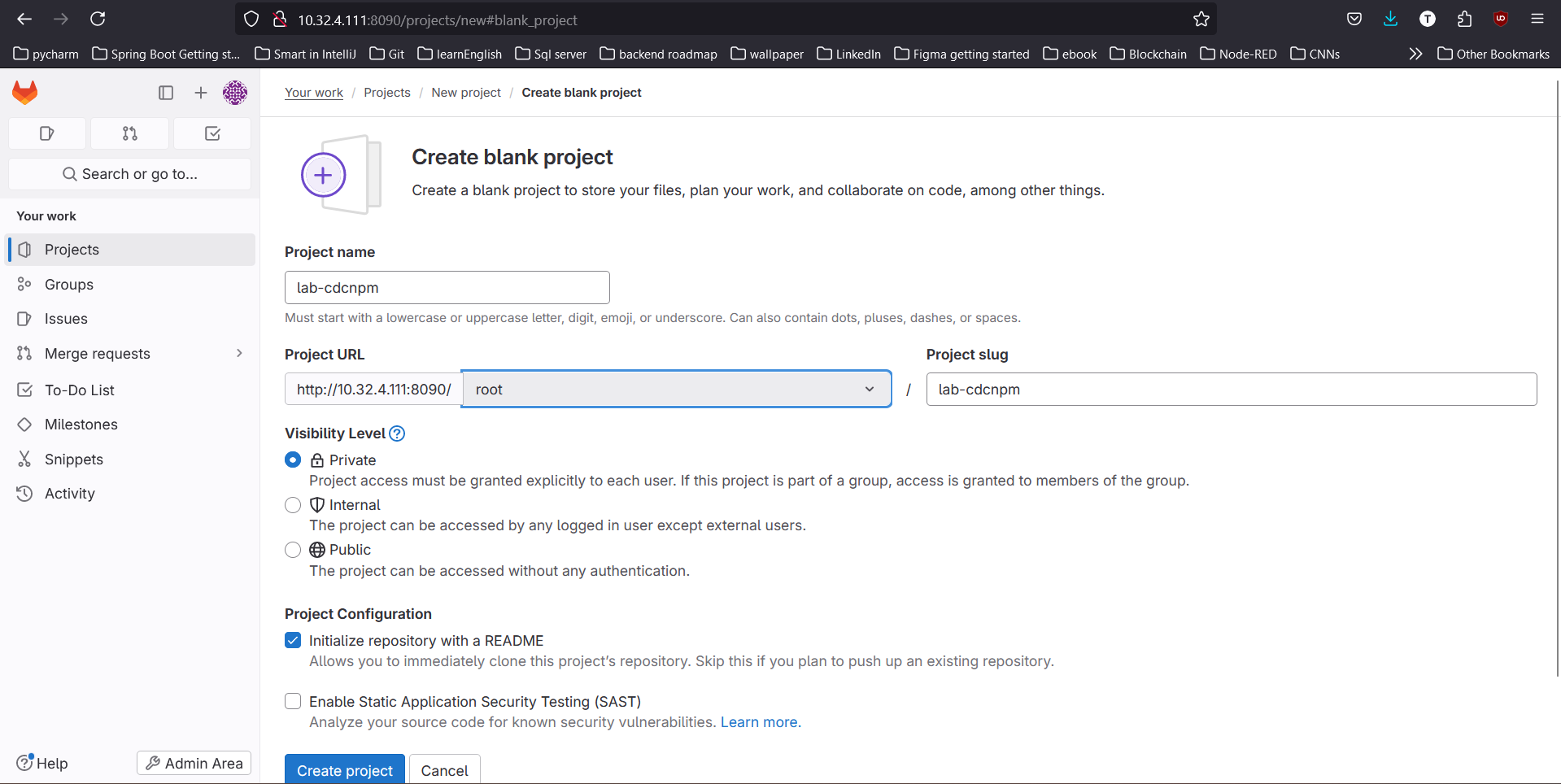
$ docker exec -it gitlab-server bash truy cập vào bên trong container đang chạy gitlab

$ gitlab-rake "gitlab:password:reset[root]" đổi password mới

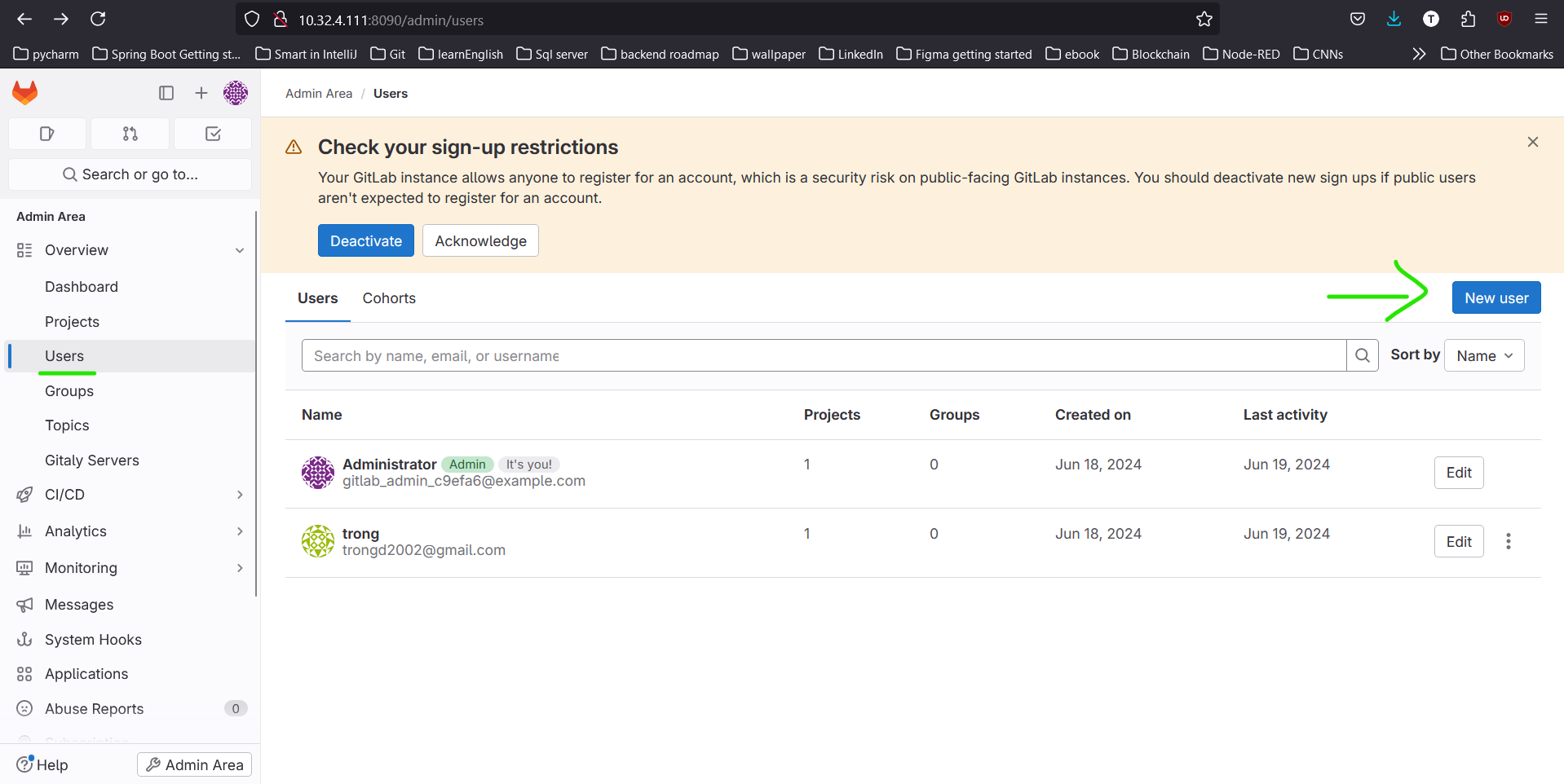
* Tiến hành đăng nhập vào Gitlab với IP và Port đã thiết lập: 10.32.4.111:8090



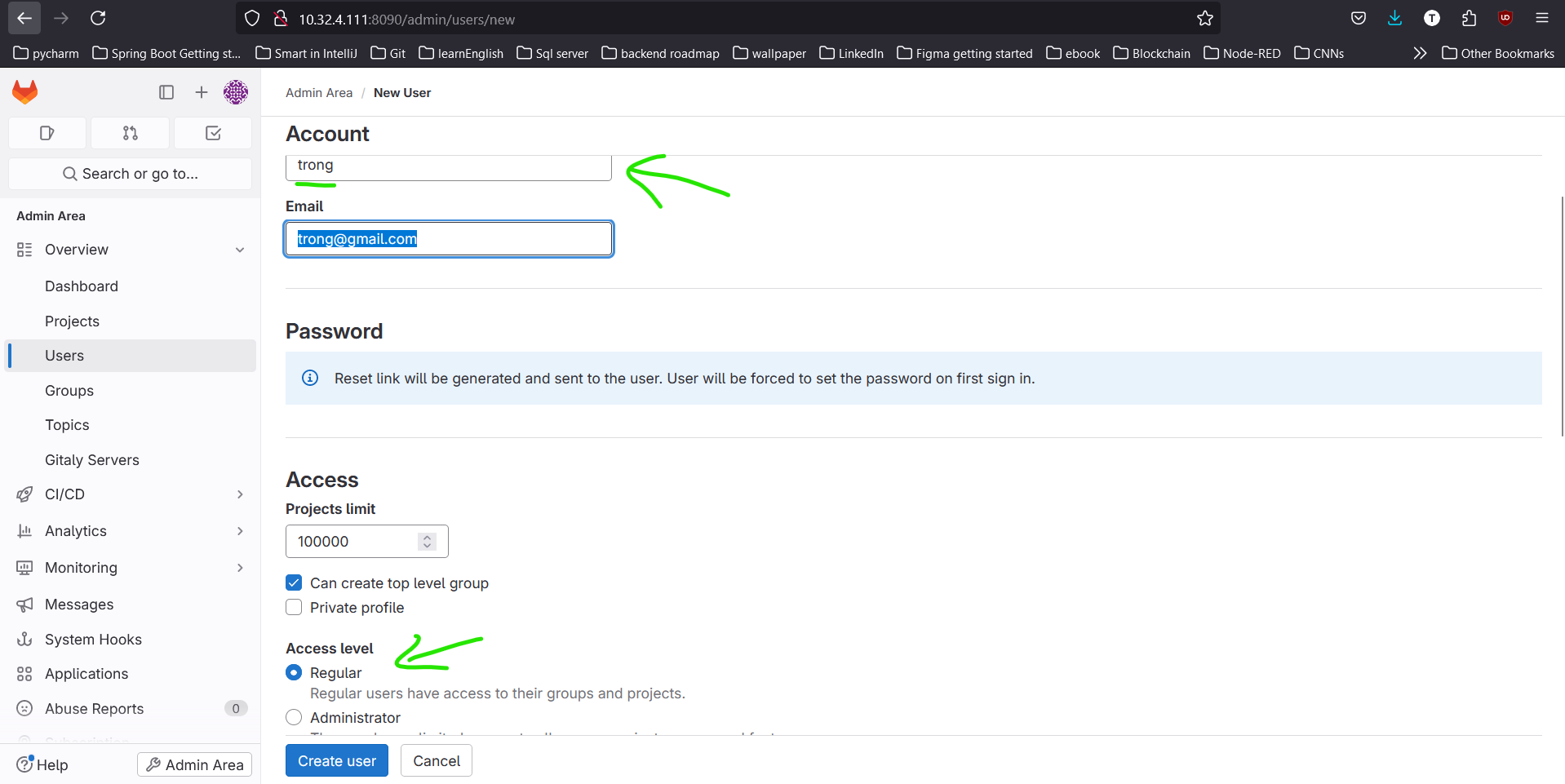
* Tiến hành tạo dự án mới có tên là lab-cdcnpm chẳng hạn:



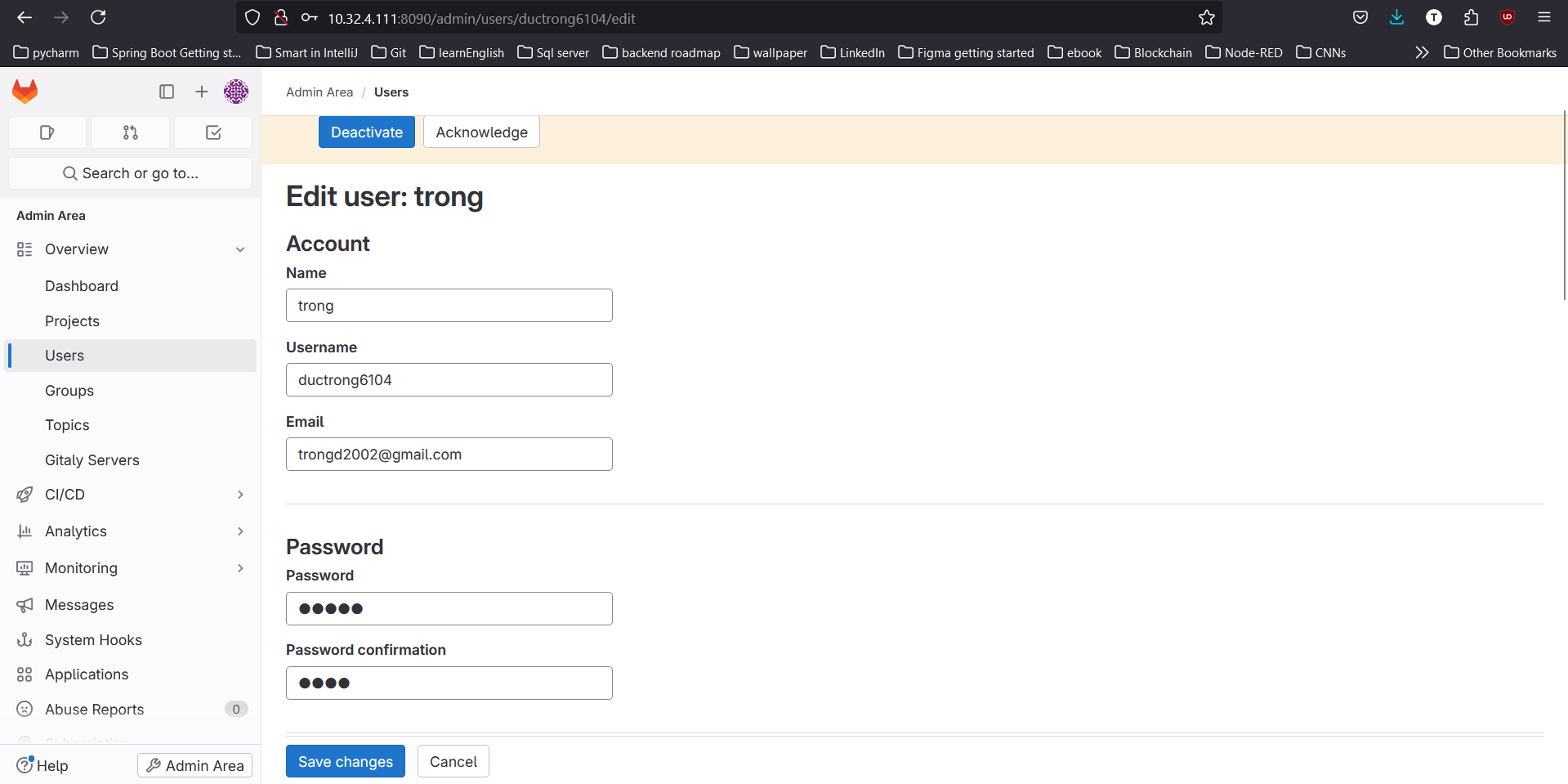
* Tạo dự án thành công và tiến hành tạo người dùng mới (tương ứng với thành viên trong team)



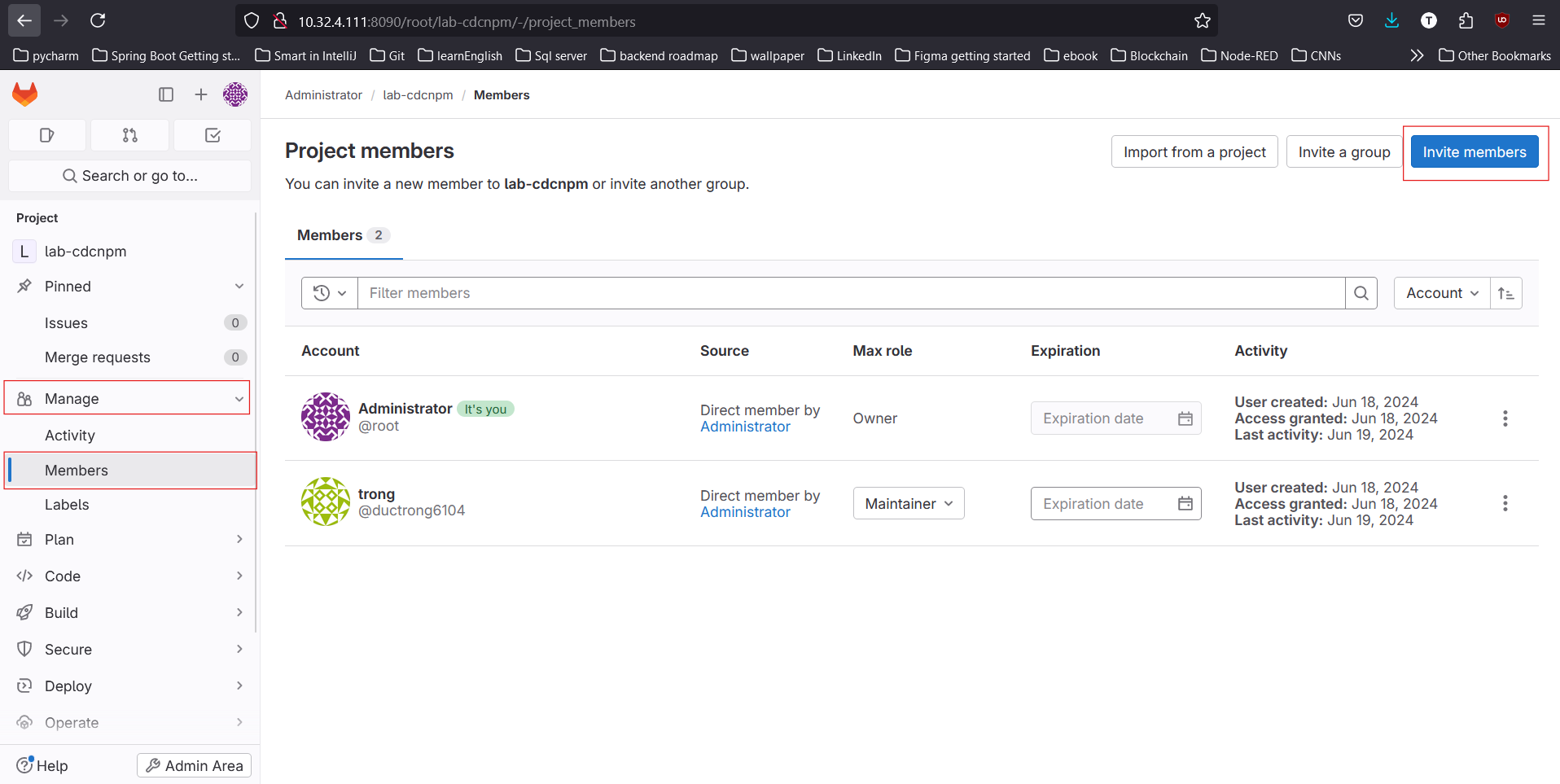
* Mình tạo một tài khoản tên ductrong6104 và role regular



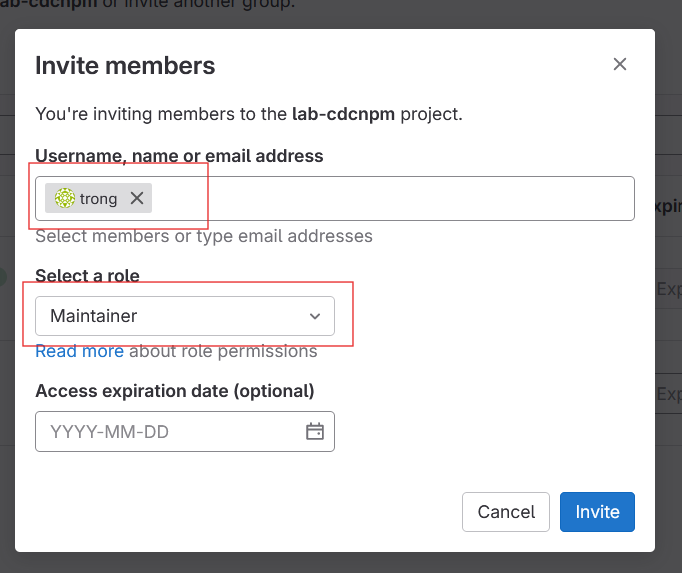
* Tiến hành cập nhật password



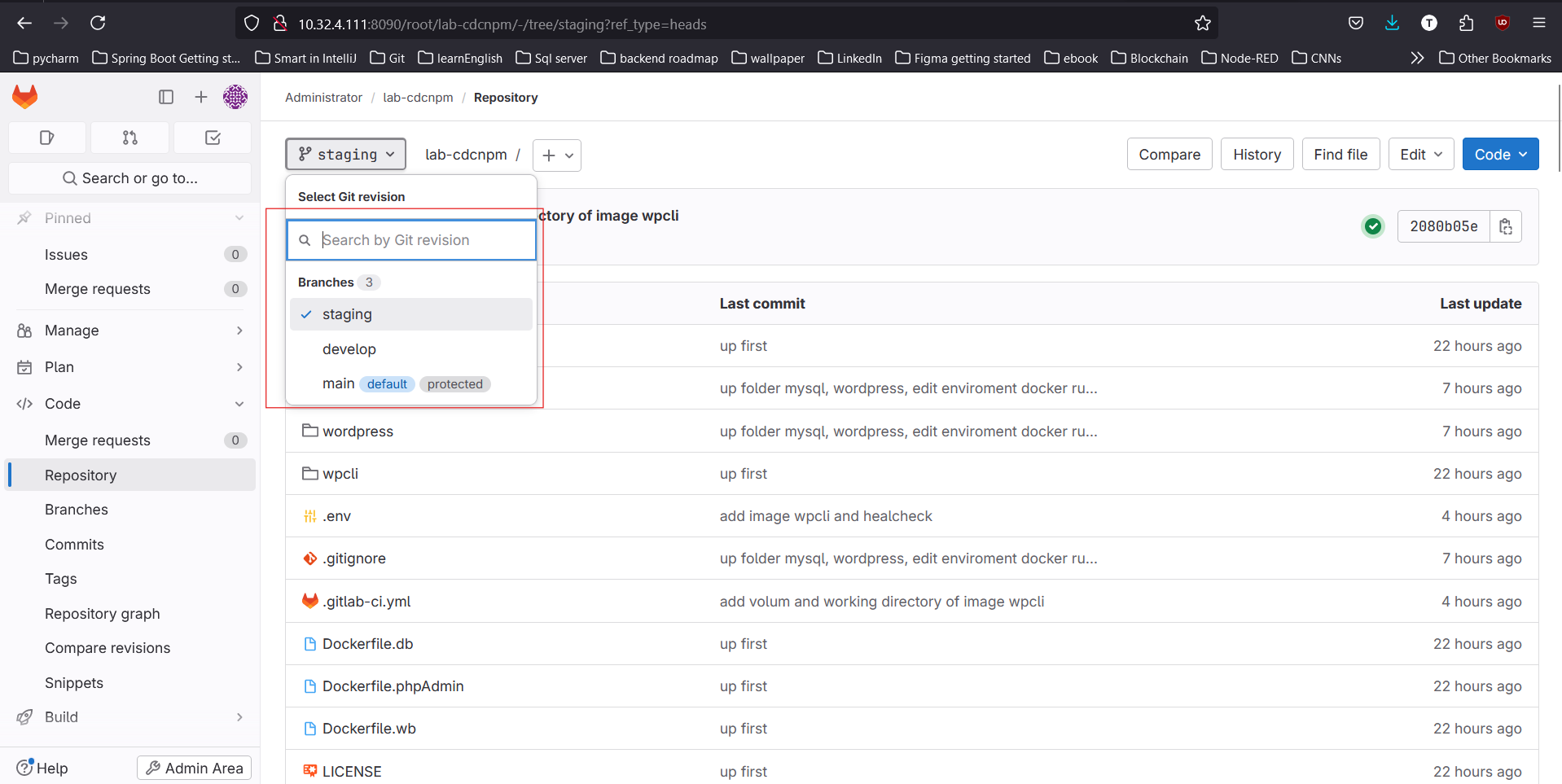
* Qua trang admin tiến hành thêm người dùng đó vào dự án



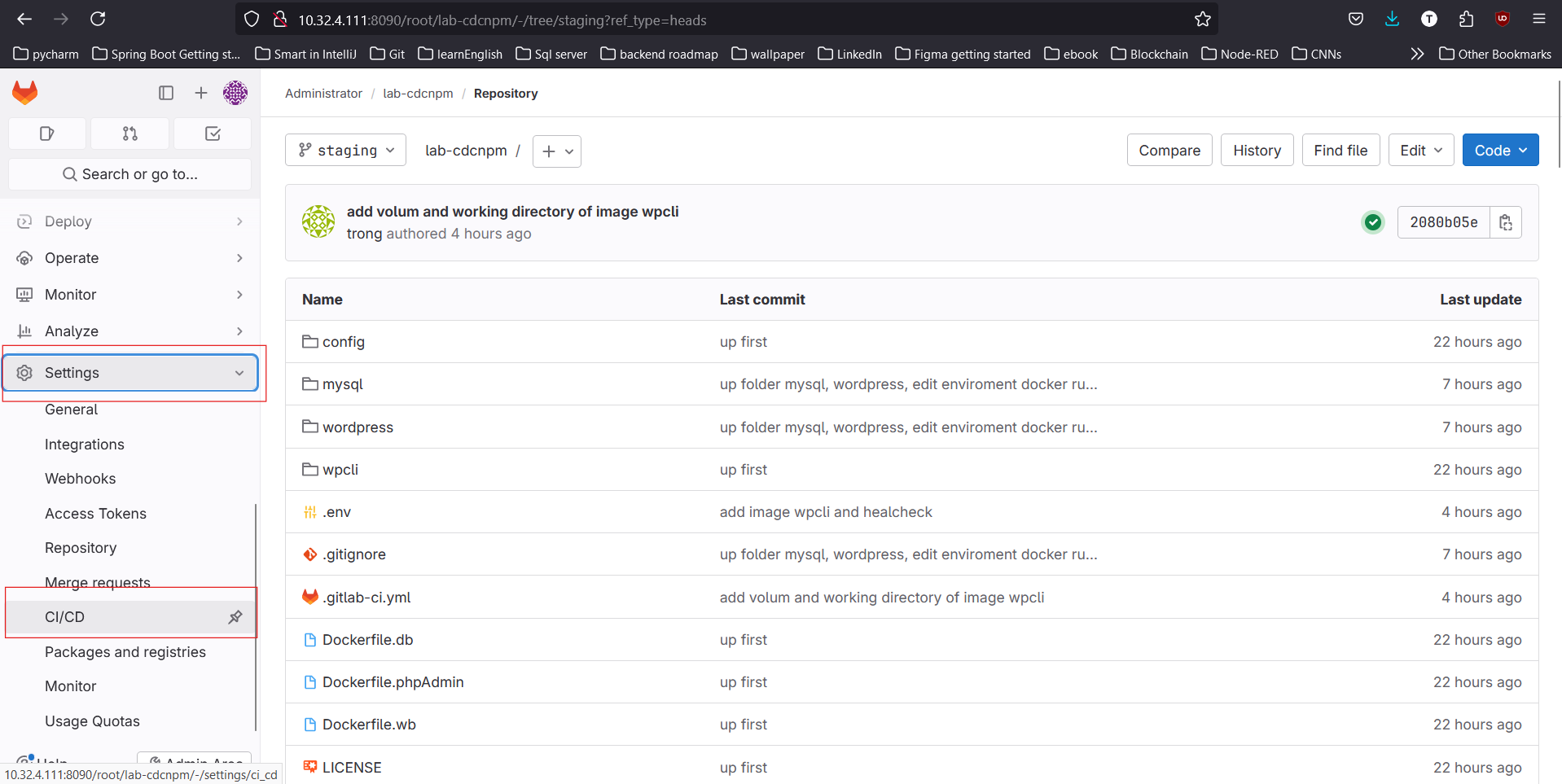
* Đây sẽ là tài khoản để triển khai, bảo trì dự án nên bạn phải đặt quyền là Maintainer nhé



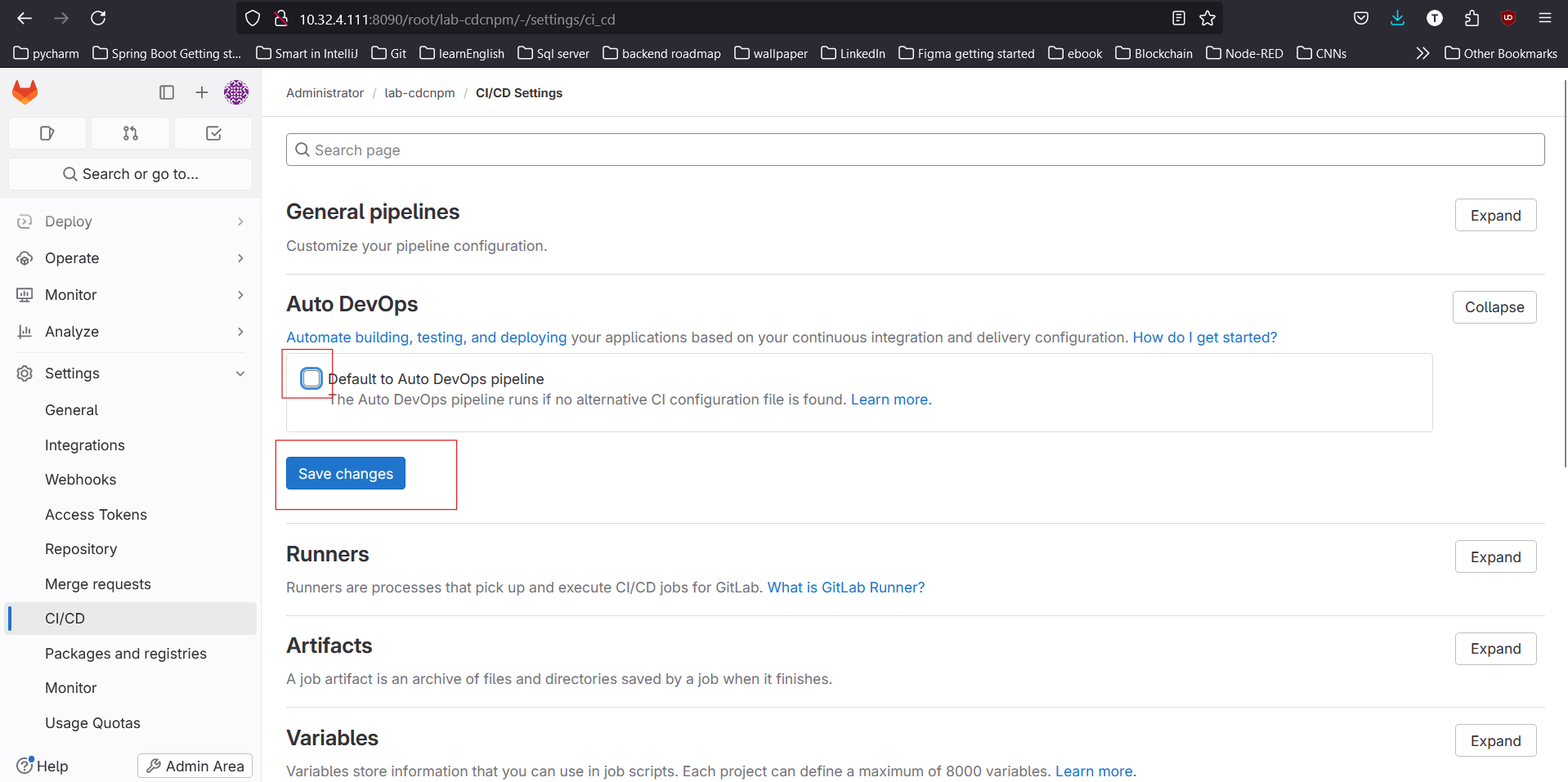
* Login tài khoản vừa tạo tiến hành push code lên và tiến hành tạo thêm nhánh develop và staging (main là nhánh mặc định)



* tắt Auto DevOps của Gitlab tiến hành truy cập Setting để tắt



* Bỏ chọn như dưới đây và lưu lại



* Vậy đến đây chúng ta đã setup xong Step 1 trong phần CICD pipeline này với tạo Gitlab và đưa dự án lên trên Gitlab, tiếp theo sẽ đến bước vô cùng quan trọng bạn chú ý nhé

## Bước 2: Tạo server gitlab-runner-registry, cài đặt Gitlab-runner kết nối với Gitlab và Registry

* Đầu tiên chúng ta phải hiểu gitlab-runner là gì. Bạn có thể hiểu đơn giản gitlab-runner tương ứng chính là người lấy code từ trên server gitlab về và thực thi các câu lệnh của bạn. Vậy tác dụng chính xác của gitlab-runner ở đây sẽ là clone code từ gitlab tiến hành chạy công việc CICD pipeline như build dự án đóng gói nó và triển khai lên server deploy cùng với registry. Registry bạn có thể hiểu đơn giản giống như docker hub là một nơi lưu trữ những image và đơn giản hơn nữa nó giống như gitlab vậy khác là gitlab lưu trữ các source code registry lưu trữ các image docker.
* Như ở trên server này có IP là 10.32.4.125 chúng ta cũng tiến hành cài đặt gitlab-runner và registry

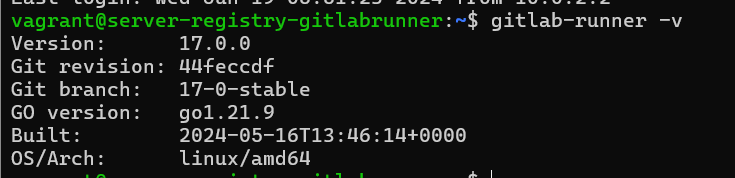
$ apt-get update

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

$ apt-get install gitlab-runner

$ apt-cache madison gitlab-runner

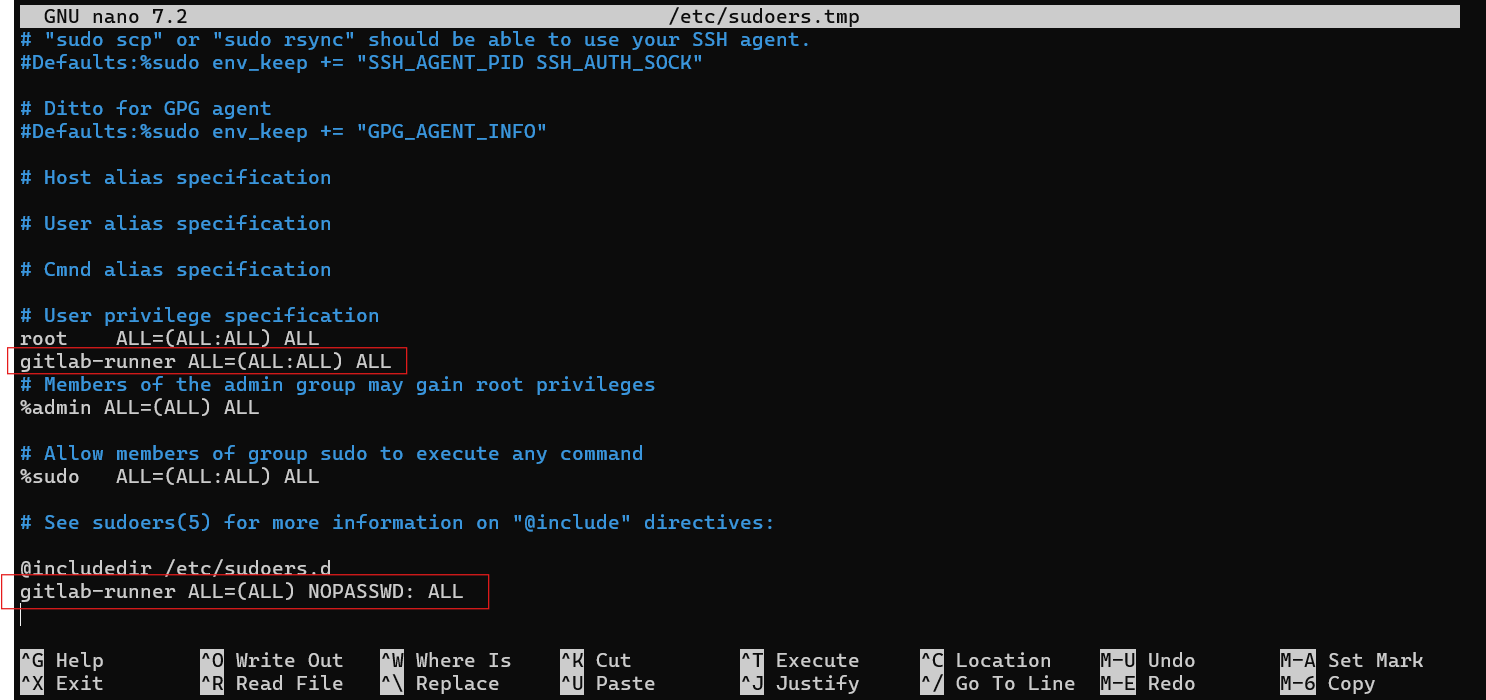
$ gitlab-runner -version



$ usermod -aG docker gitlab-runner

$ usermod -aG sudo gitlab-runner

$ visudo chỉnh sửa tệp sudoers, tệp này xác định người dùng và nhóm có quyền quản trị viên thêm 2 dòng gitlab-runner như dưới đây



* Sau khi đã cài đặt xong gitlab-runner mình sẽ tiến hành kết nối gitlab-runner với gitlab
* $ gitlab-runner register khởi tạo gitlab-runner mới theo thứ tự

Server gitlab (ở đây của mình là **10.32.4.111:8090**)

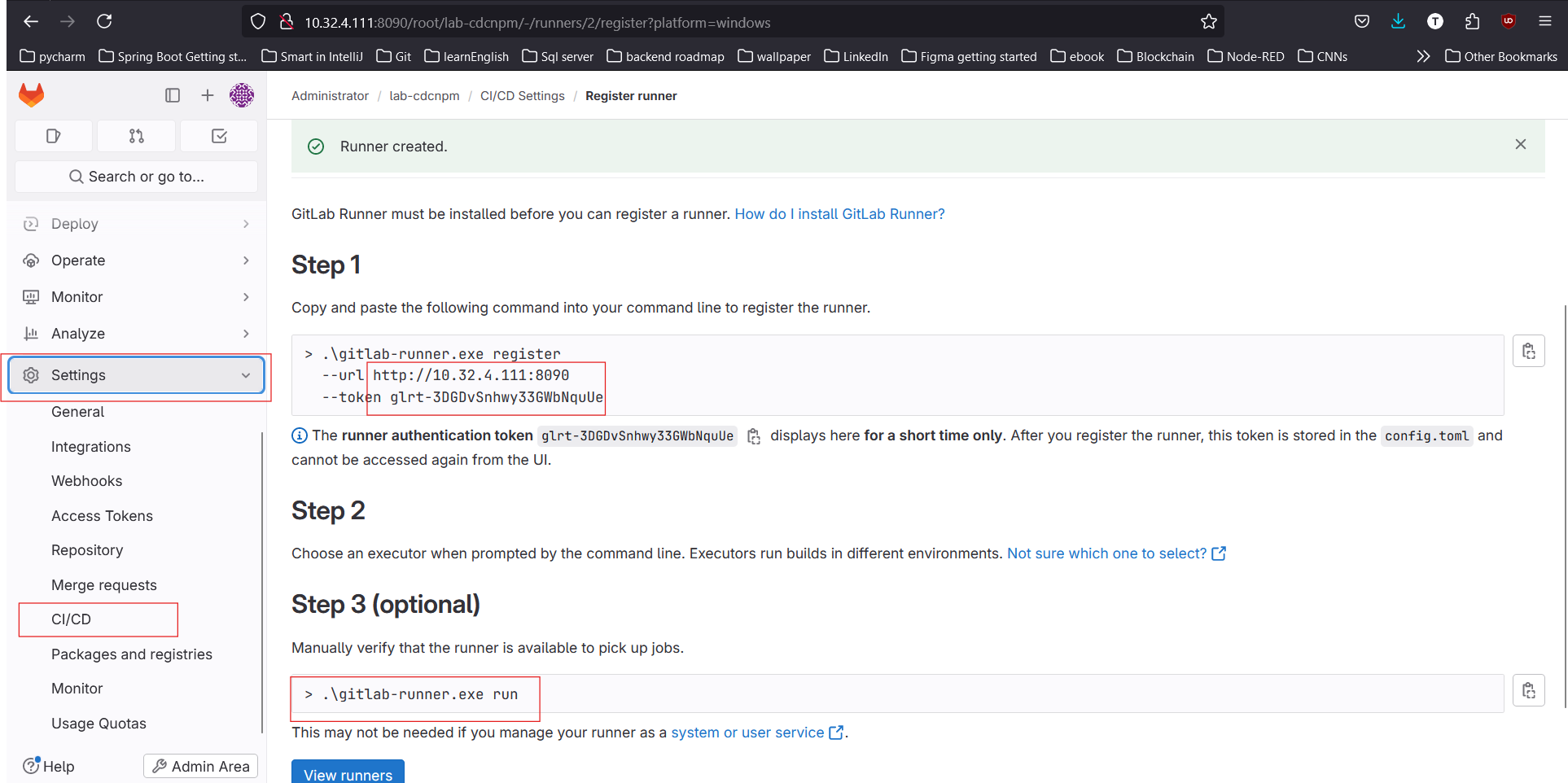
Nhập token project lấy từ setting cicd trong dự án trên gitlab như dưới đây (dự án của mình là **glrt-3DGDvSnhwy33GWbNquUe**)

Nhập mô tả, bạn có thể thêm mô tả cho runner này (như mình **ENTER** qua và mặc định lấy hostname server)

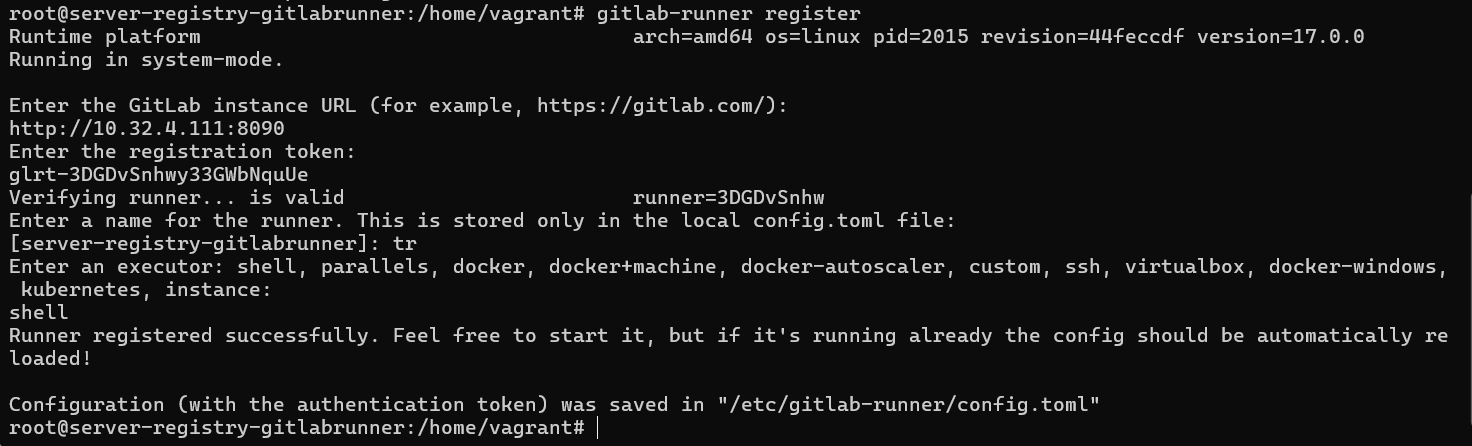
Nhập tag, tag của runner có thể được chỉ định trong cài đặt, ví dụ chỉ đích danh tag này thì runner mới chạy (ở đây mình điền **mr**)

Nhập ghi chú (Phần này **ENTER** bỏ qua cũng được)

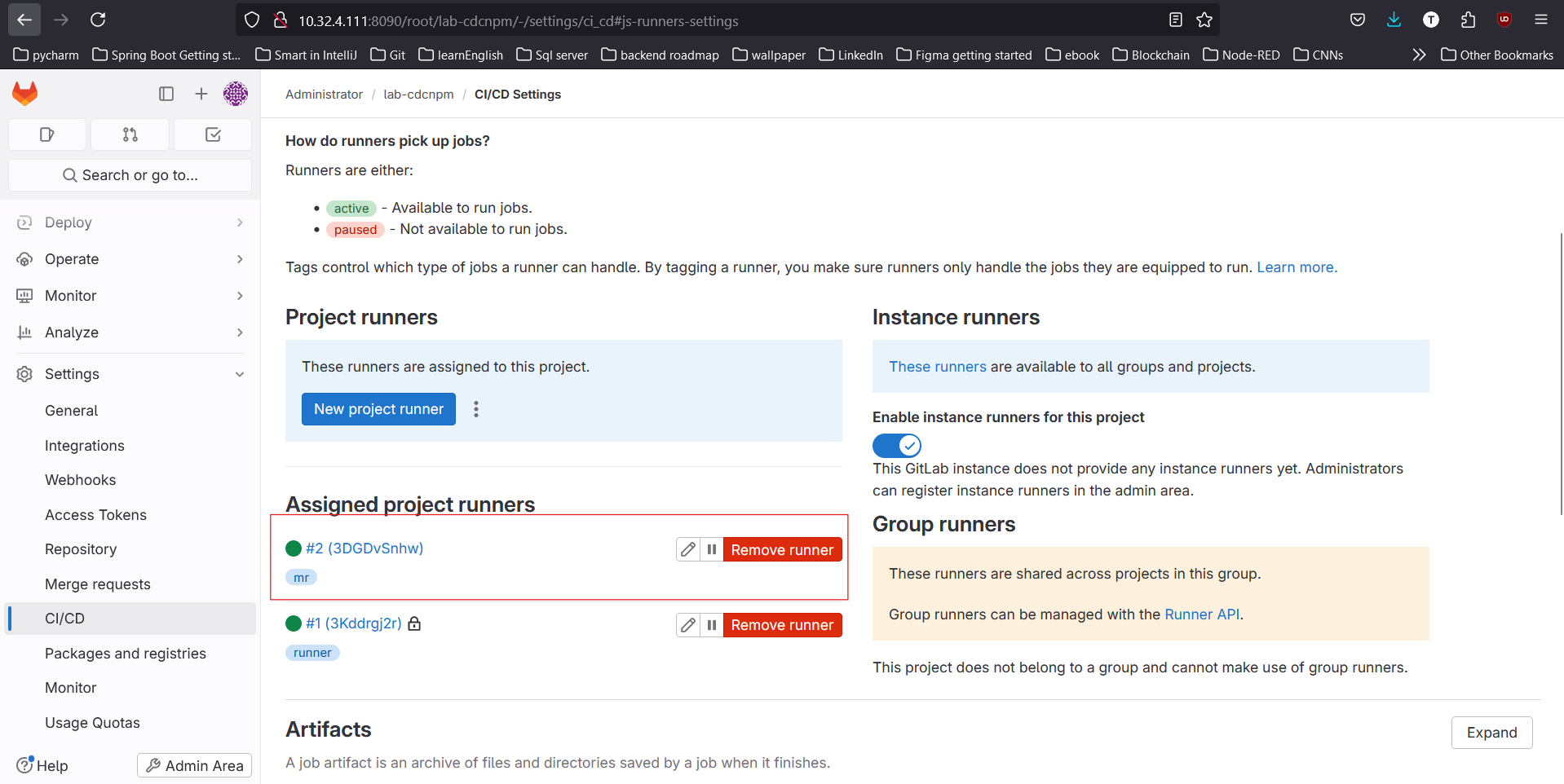
Lựa chọn cách chạy (ở đây mình ghi **shell**)



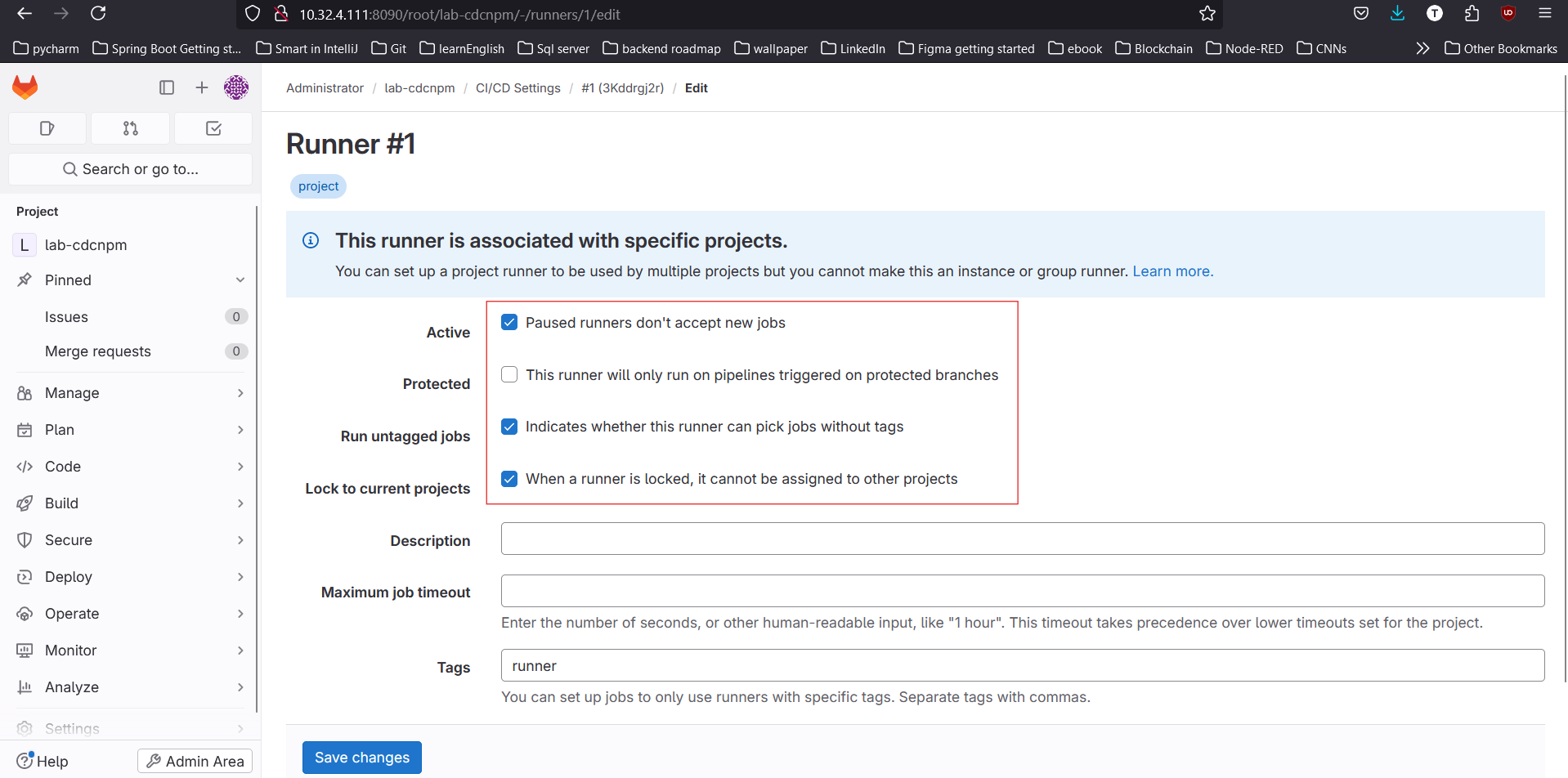
* Làm đúng các bước trên và hoàn thành được kết quả, vậy là chúng ta setup xong gitlab-runner và kết nối với gitlab trong bài CICD pipeline này



* Reload lại trang gitlab vào setting CICD pipeline sẽ thấy có 1 runner trạng thái đang chạy



* Nhấn vào biểu tượng edit cấu hình CICD pipeline tiến hành vào chỉnh sửa một chút
  + Dòng đầu tiên **chọn** để runner nhận chạy công việc
  + Dòng thứ hai chọn để chạy trên các nhánh bảo vệ, phần này sẽ thêm lý thuyết mới đầu bạn cũng chưa cần quan tâm nên có thể **bỏ tích chọn**
  + Dòng thứ ba như mình có nói ở trên nếu t **không tích chọn** nó sẽ chạy khi được chỉ định tag chính xác.
  + Dòng thứ tư khóa runner này chỉ chạy trong dự án hiện tại nên **chọn**, vì các runner có thể được share giữa các dự án nhưng trường hợp này mình chọn tức là không cho dự án khác sử dụng runner này.



* Mình sẽ tiến hành cài đặt docker và docker-compose như đã hướng dẫn ở trên trong server này (docker sẽ được cài đặt trên cả 3 server)
* Tiếp theo mình sẽ setup registry bằng docker

$ mkdir -p /home/docker-app/registry && chmod -R 777 /home/docker-app/registry && cd /home/docker-app/registry tạo thư mục để lưu config triển khai registry và đi tới thư mục đó

$ mkdir certs data tạo 2 thư mục, certs để lưu thông tin key, data để lưu dữ liệu được

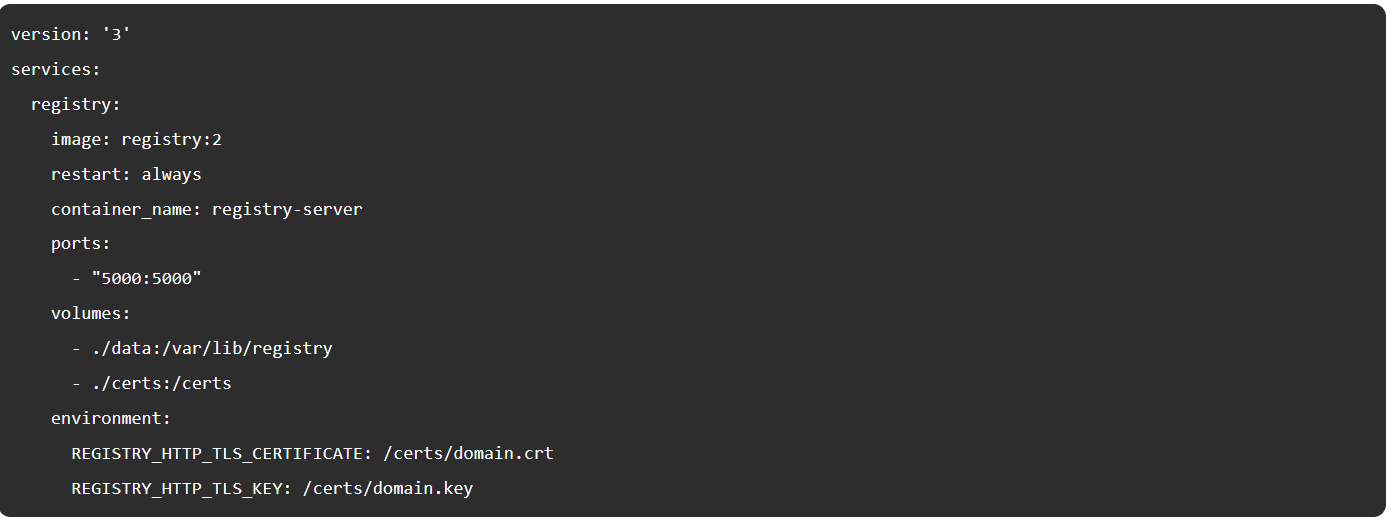
$ apt get update

$ apt-get install openssl cài đặt gói của openssl (như đã nói bài viết hướng tới sự “miễn phí” nên mình sử dụng OpenSSL để tạo chứng chỉ SSL/TLS tự ký để registry có thể sử dụng https)

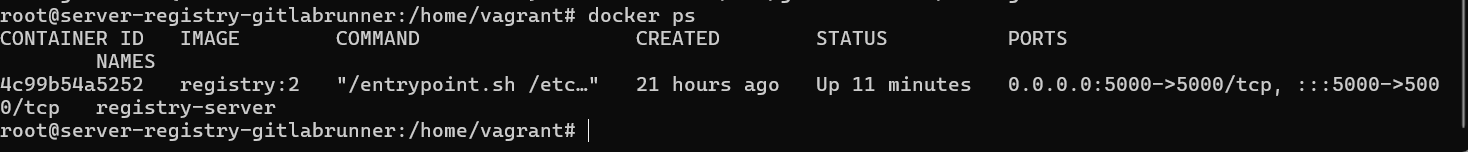
$ openssl req -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key -subj "/CN=10.32.4.125" -addext "subjectAltName = DNS:10.32.4.125,IP:10.32.4.125" -x509 -days 365 -out certs/domain.crt

* Trong đó:
  + **req**: tùy chọn này chỉ định rằng chúng ta muốn sử dụng tiện ích yêu cầu chứng chỉ của OpenSSL.
  + **-x509**: Tùy chọn này cho biết chúng ta muốn tạo một chứng chỉ tự ký thay vì tạo yêu cầu ký chứng chỉ (CSR).
  + **-newkey rsa:4096**: Tùy chọn này tạo ra một khóa RSA mới với độ dài 4096 bit. Đây là một khóa đối xứng, có thể được sử dụng để mã hóa và giải mã thông tin, cũng như ký và xác minh chữ ký số.
  + **-nodes**: Tùy chọn này chỉ định rằng khóa không được bảo vệ bằng mật khẩu. Nếu không có tùy chọn này, OpenSSL sẽ yêu cầu nhập mật khẩu để bảo vệ khóa riêng tư.
  + **-keyout certs/domain.key**: Tùy chọn này chỉ định rằng khóa riêng tư sẽ được lưu trữ tệp domain.key trong thư mục certs vừa tạo.
  + **-out certs/domain.crt**: Tùy chọn này chỉ định rằng chứng chỉ sẽ được lưu trữ tệp domain.crt trong thư mục certs.
  + **-days 365**: Tùy chọn này chỉ định thời hạn của chứng chỉ là 1 năm (365 ngày).
  + **-subj “/CN=10.32.4.125”**: Tùy chọn này chỉ định thông tin chủ chứng chỉ, bao gồm tên chính (Common Name – CN) là địa chỉ IP của máy chủ.
  + **-addext “subjectAltName = IP:10.32.4.125”**: Tùy chọn này thêm một phần mở rộng cho chứng chỉ, chứa địa chỉ IP của máy chủ trong phần Alternative Names. Việc thêm phần mở rộng này giúp tránh tình trạng một số trình duyệt không nhận diện được chứng chỉ nếu chỉ có tên miền trong phần Common Name.
    - Thêm nữa, nếu không có Alternative Names thì khả năng lỗi của bạn rất cao khi một server khác tiến hành xác thực registry của bạn nên bạn hãy làm theo hướng dẫn nhé.
* Được kết quả tạo thành công chứng chỉ SSL/TLs tự ký (chúng ta có thể config với website để có thể nhận https)

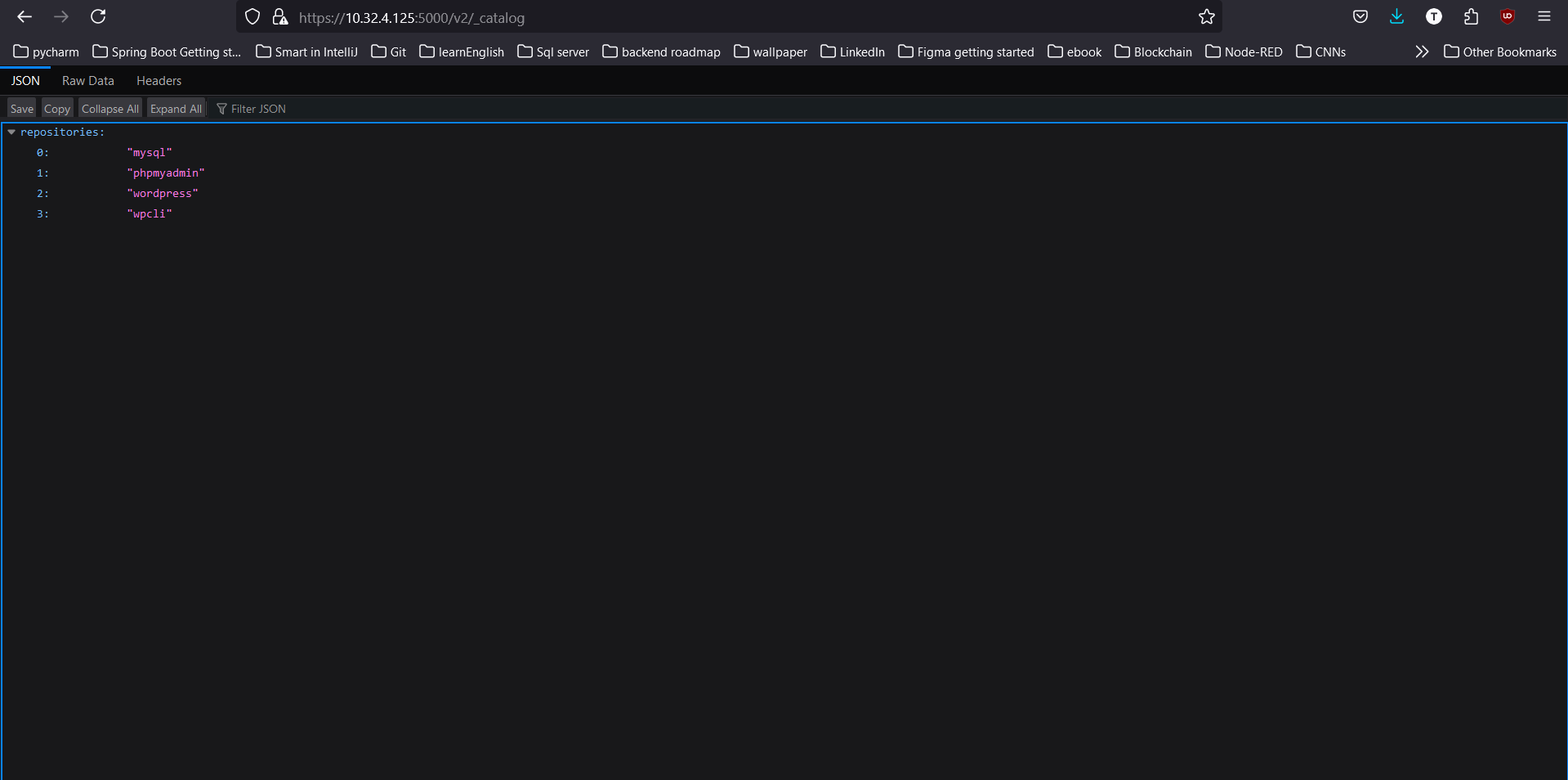
$ nano docker-compose.yml sau khi tạo xong chứng chỉ tiến hành tạo file docker-compose để chạy registry với nội dung



$ docker-compose up -d khởi chạy registry



* Tiến hành truy cập vào website như đã config trong file docker-compose là [**https://10.32.4.125:5000**](https://10.32.4.125:5000)
* Chấp nhận cảnh báo vì trình duyệt cho biết là đây chứng chỉ tự ký.
* Như vậy là mình đã cài đặt thành công registry, đây sẽ là nơi lưu trữ image (dự án đã được đóng gói thành image để sẵn sàng chạy)



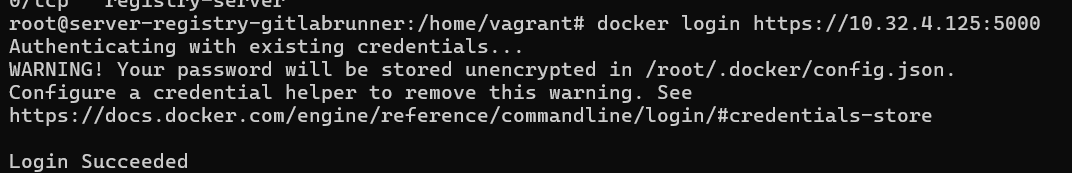
* Tuy nhiên nếu sử dụng docker login thì vẫn sẽ chưa được vì đây là chứng chỉ tự ký nên server sẽ chặn vì vậy mình phải làm thêm 1 bước là xác thực key, bạn làm theo những bước dưới đây

$ mkdir -p /etc/docker/certs.d/10.32.4.125:5000/ tạo thư mục để lưu trữ xác thực

$ cp /home/docker-app/registry/certs/domain.crt /etc/docker/certs.d/10.32.4.125:5000/ca.crt copy khóa riêng nãy chúng ta tạo vào thư mục vừa tạo

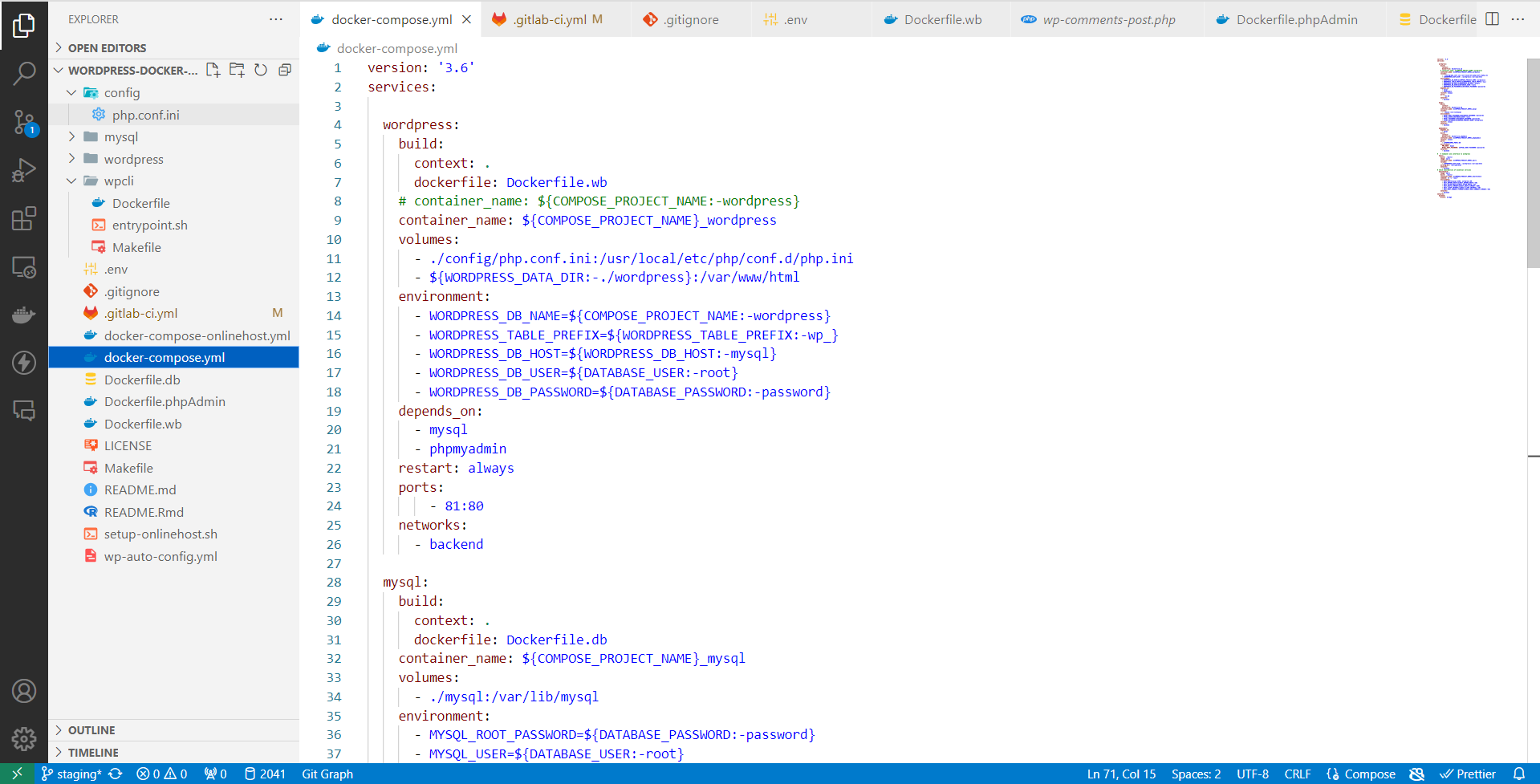
$ systemctl restart docker restart lại docker

$ docker login 10.32.4.125:5000 tiến hành login thử thành công và được kết quả (chú ý ở đây bạn có thể nhập username password bất kỳ hoặc muốn xác thực bạn có thể tìm kiếm cụm từ **basic authentication** cũng rất dễ dàng mình sẽ nhường bạn làm để tăng khả năng research)

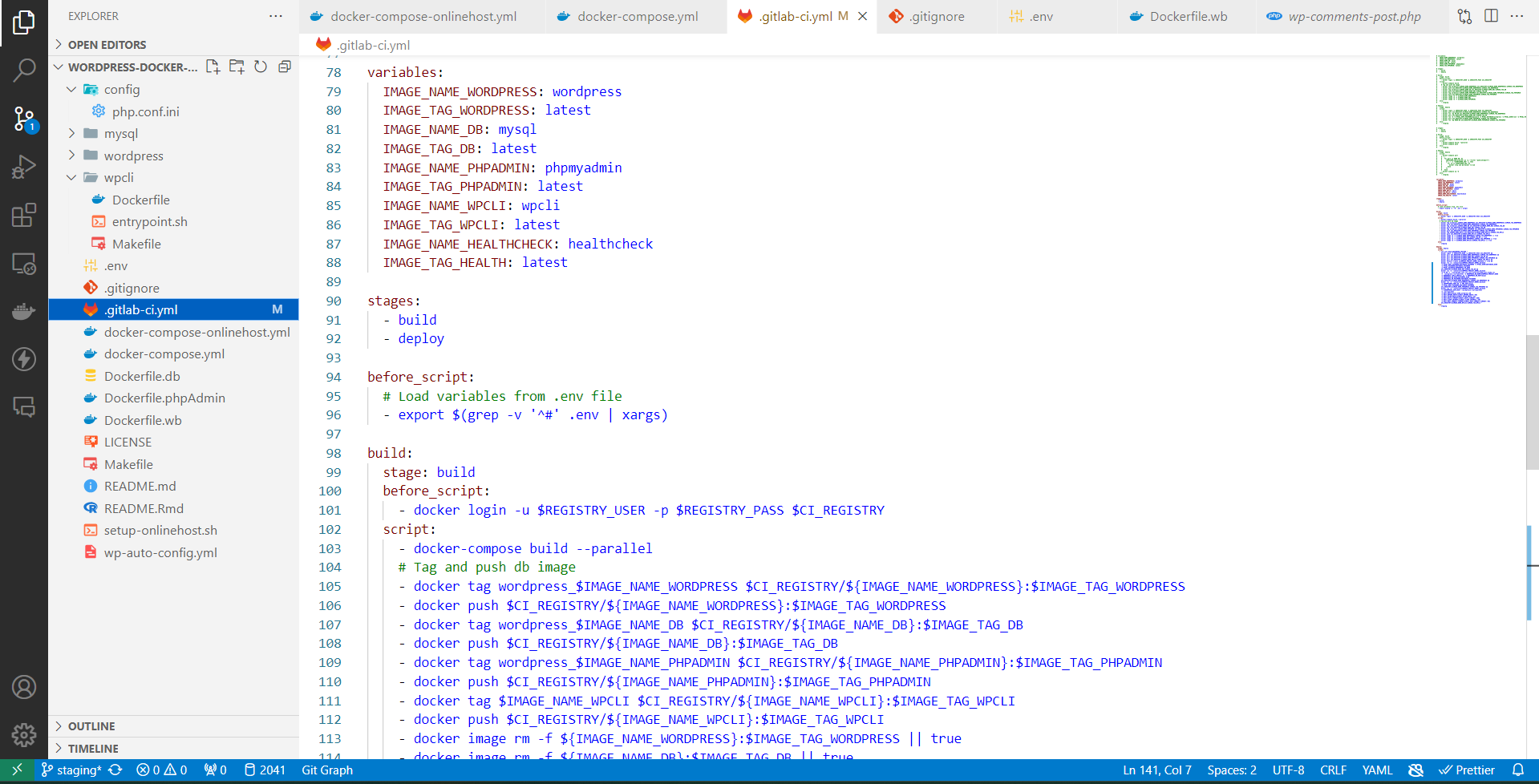


## Bước 3: Cấu hình CICD pipeline trên gitlab và trong source

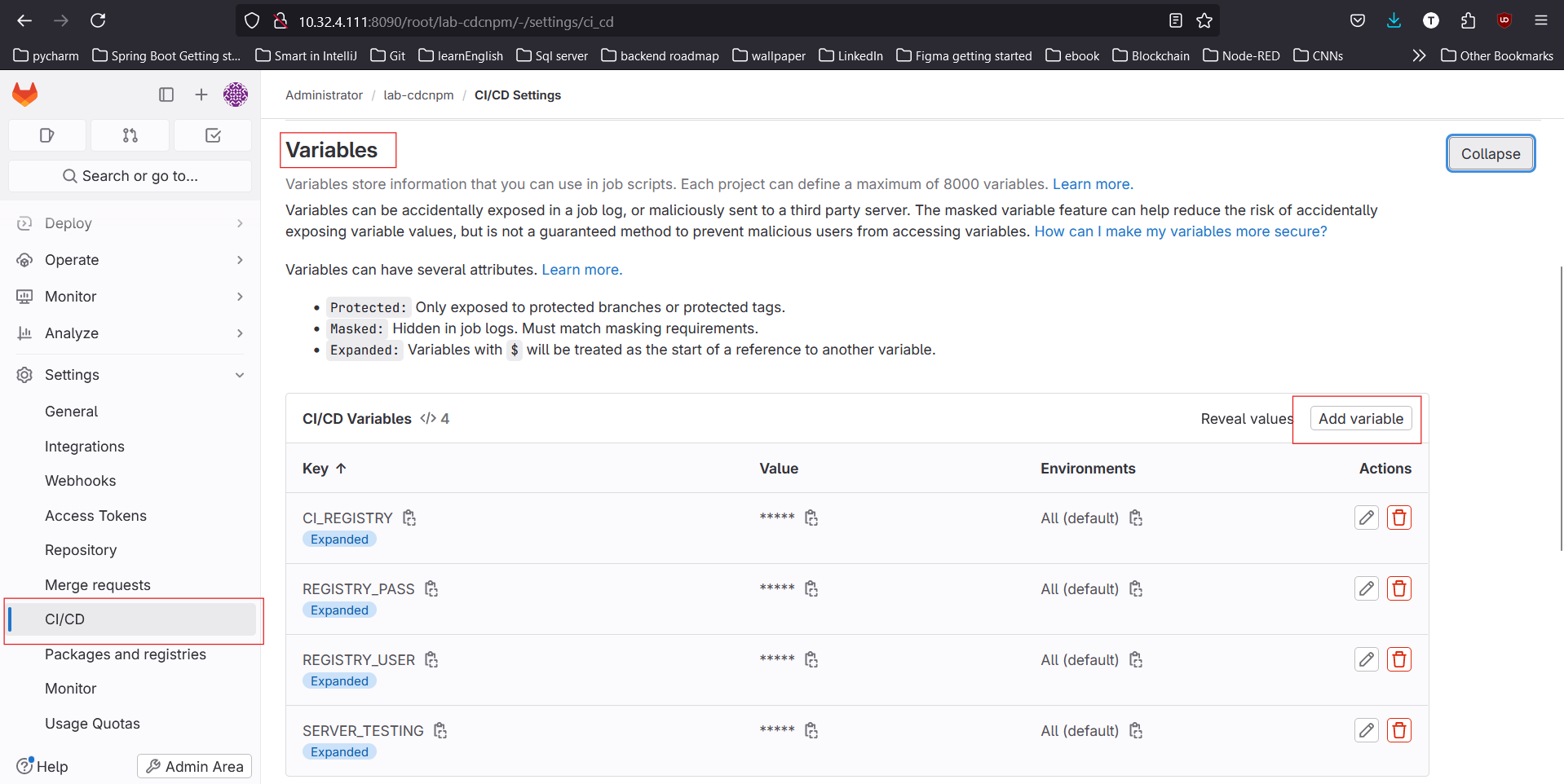
* Đây chính là bước setup trong dự án để có thể chạy CICD pipeline yêu cầu bạn phải biết về docker và viết yml file.
* Tạo Dockerfile, docker-compose.yml cho dự án wordpress như dưới đây:



* Tiếp theo tiến hành tạo .gitlab-ci.yml file để cấu hình các công việc chạy CICD pipeline chi tiết dưới đây
* variables: tạo các biến môi trường
* stages: khai báo các giai đoạn của CICD pipeline ở đây mình có 2 giai đoạn build và deploy
* build: khai báo gia đoạn build
  + before\_script: là khối lệnh này sẽ được thực thi trước khi chạy script là các công việc chính trong giai đoạn đó, ở đây mình tiến hành login vào registry (bạn sẽ biết ngay dưới đây)
  + script: các công việc trong giai đoạn đó, ở đây mình tiến hành các bước như build image, gắn tag image vừa build và tiến hành đẩy lên registry, và mình xóa image mà vừa build đi
* only: chỉ định đúng là nhánh đó thì job CICD pipeline mới chạy công việc, ở đây mình để là staging
* deploy: tương tự vậy đây là 1 giai đoạn và giai đoạn này thực hiện như sau: từ server gitlab runner login vào server deploy và từ server deploy tiến hành  login vào registry và pull image về tiến hành chạy container ý lên với port là 8081



* Bạn có thấy những phần có $ đó không, đó chính là các biến như $REGISTRY\_USER, $REGISTRY\_PASS,.. và bạn có hỏi nó nằm ở đâu không, chính là đây



* Chi tiết như sau:
  + CI\_REGISTRY: là registry chúng ta mới cài đặt ở trên tức là server thứ 2 với port 5000 chính là 10.32.4.125:5000
  + REGISTRY\_USER: hiện bạn có thể lưu account server thứ 2
  + REGISTRY\_PASS: tương tự bạn có thể lưu password server thứ 2
  + SERVER\_TESTING: chính là server deploy tức là server thứ 3 mình có đề cập là 10.32.3.180

## Bước 4: Tạo server deploy

* Để làm được điều đó chúng ta phải tạo một key với người dùng là gitlab-runner của server 2 và tiến hành thêm vào server 3. thao tác tiếp tục tại server 2 như sau:

$ passwd gitlab-runner cài đặt mật khẩu cho gitlab-runner

$ exit tiến hành logout tài khoản hiện tại

$ ssh gitlab-runner@10.32.4.125 tiến hành login với người dùng là gitlab-runner

$ ssh-keygen -t rsa tạo keygen

$ ssh-copy-id user-deploy@10.32.3.180 copy keygen của tài khoản gitlab-runner sang server thứ 3 (server deploy) nhập password server thứ 3 như dưới đây. Bên server svdeploy tạo user mới đặt tên là user-deploy

* Vậy là người dùng gitlab-runner có thể ssh sang server deploy, tiếp tục chúng ta sẽ cấu hình như server 2 vì bản chất chứng chỉ của chúng ta là chứng chỉ tự ký bạn làm theo các bước dưới đây:

$ mkdir -p /home/sshkey/10.32.4.125/ tạo thư mục để lưu key SSL của server 2 (mình tạo ở đây để tường minh sau các bạn có thể sử dụng cho nhiều server khác)

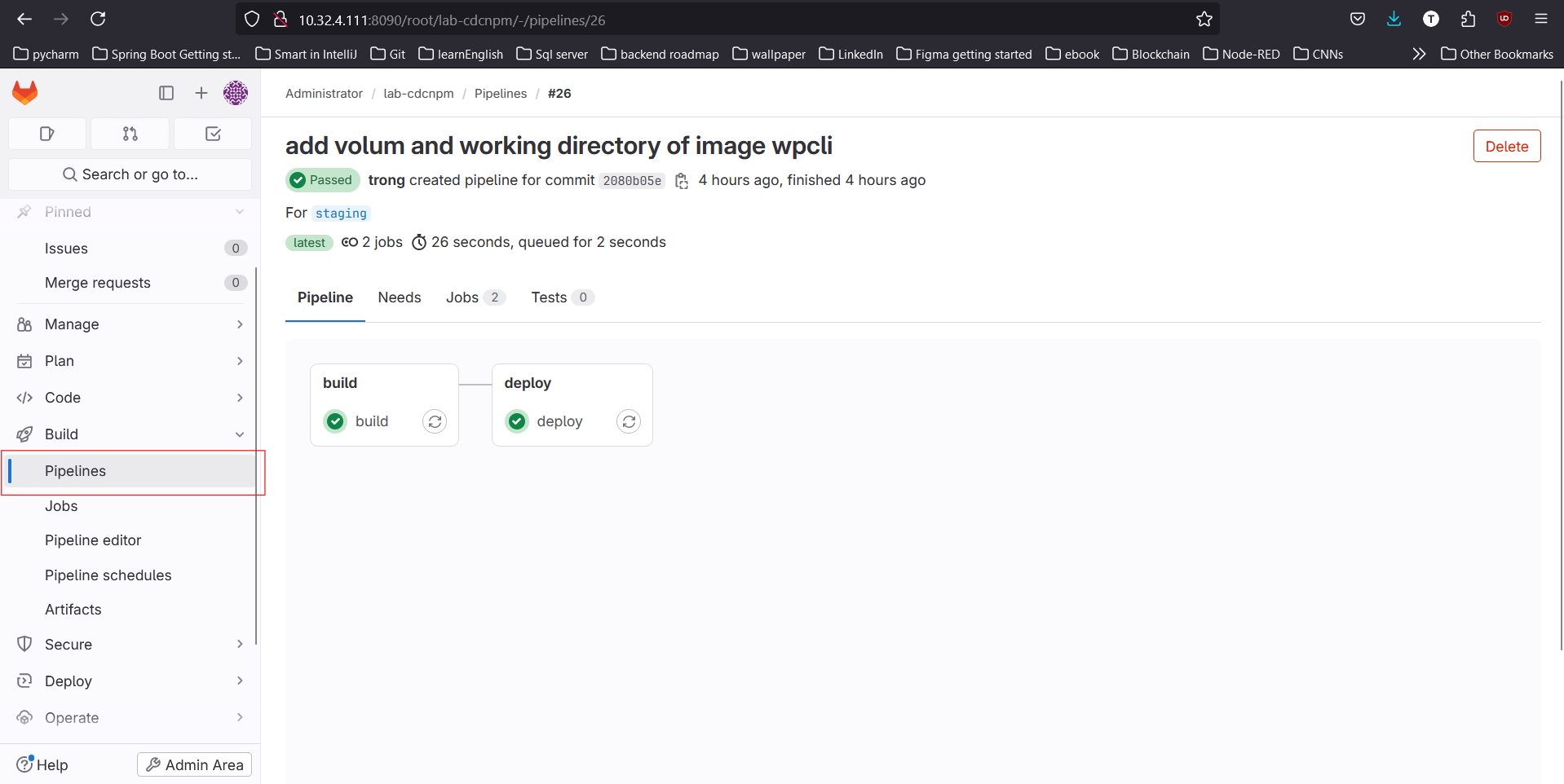
$ scp root@10.32.4.125:/home/docker-app/registry/certs/domain.crt /home/sshkey/10.32.4.125/ tiến hành copy .crt file vào thư mục vừa tạo

$ mkdir -p /etc/docker/certs.d/10.32.4.125:5000/ tương tự tạo một thư mục như ở server 2

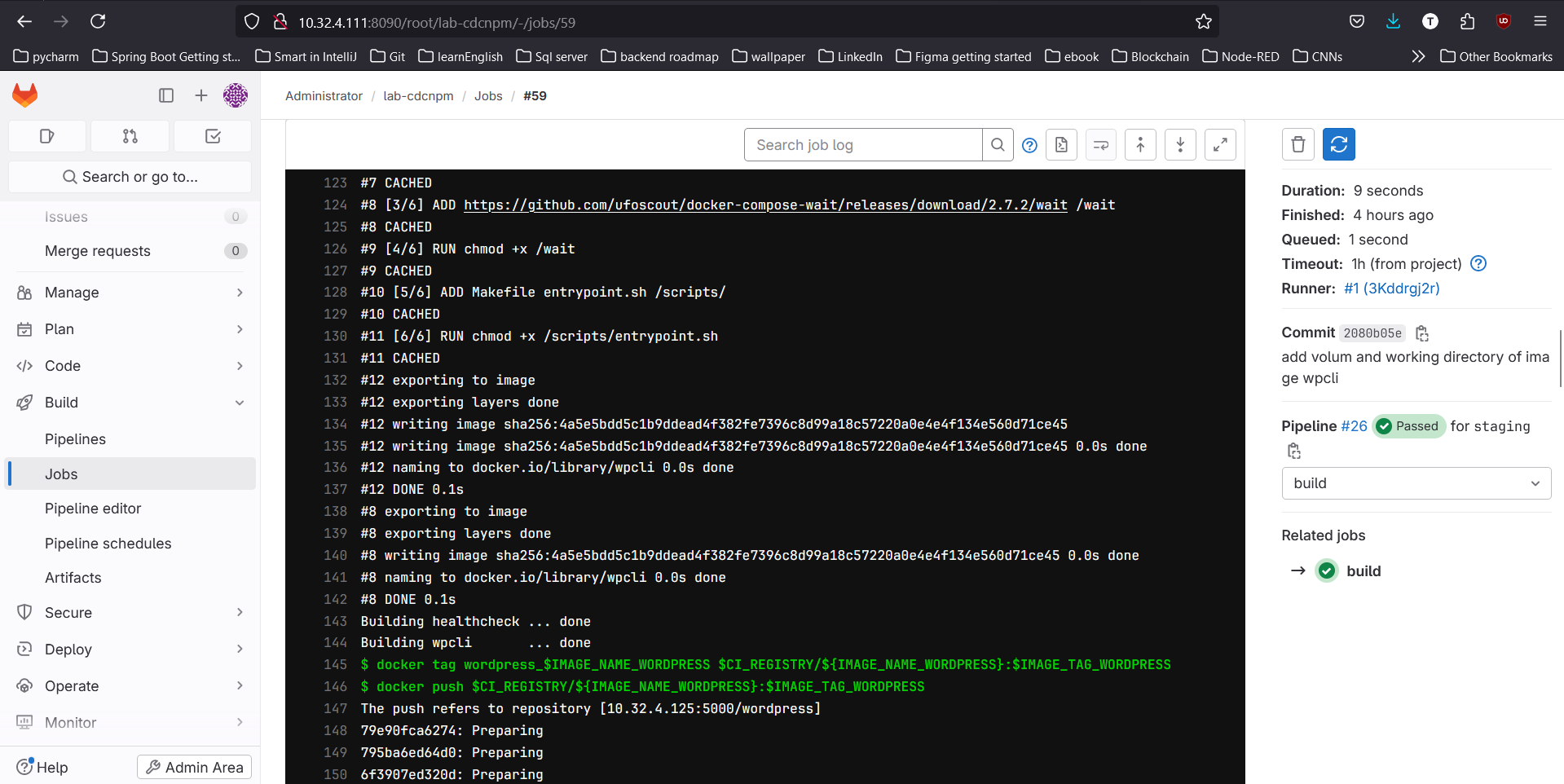
$ cp /home/sshkey/10.32.4.125/domain.crt /etc/docker/certs.d/10.32.4.125:5000/ca.crt copy file .crt vừa copy từ server 2 vào thư mục vừa tạo trong docker

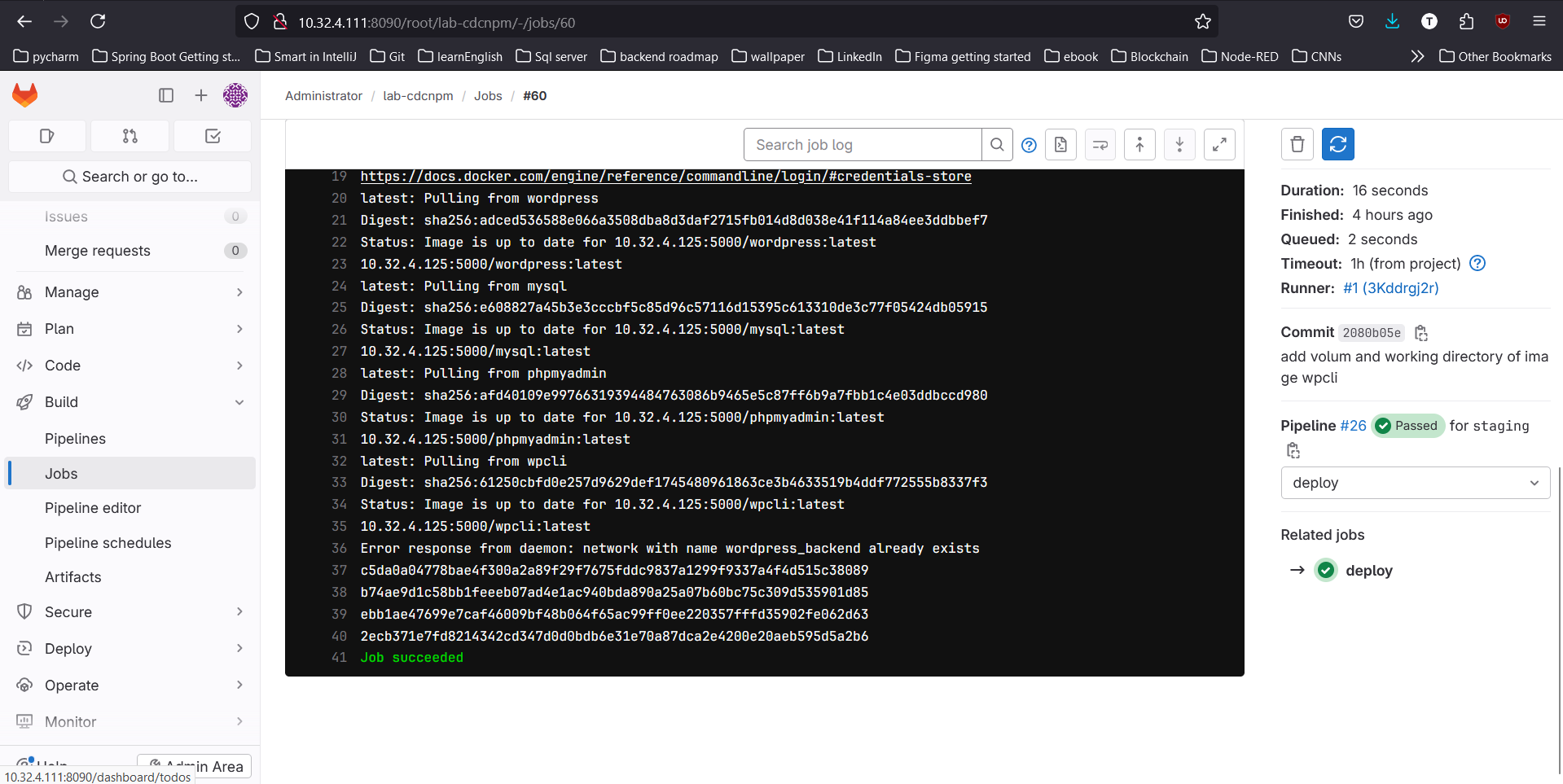
$ systemctl restart docker restart lại service docker và được kết quả

* Như vậy chúng ta đã setup xong giờ chỉ cần push code từ máy bạn lên



* Xem chi tiết job chạy





* Kiểm tra tại registry đã có image của chúng ta tải lên

