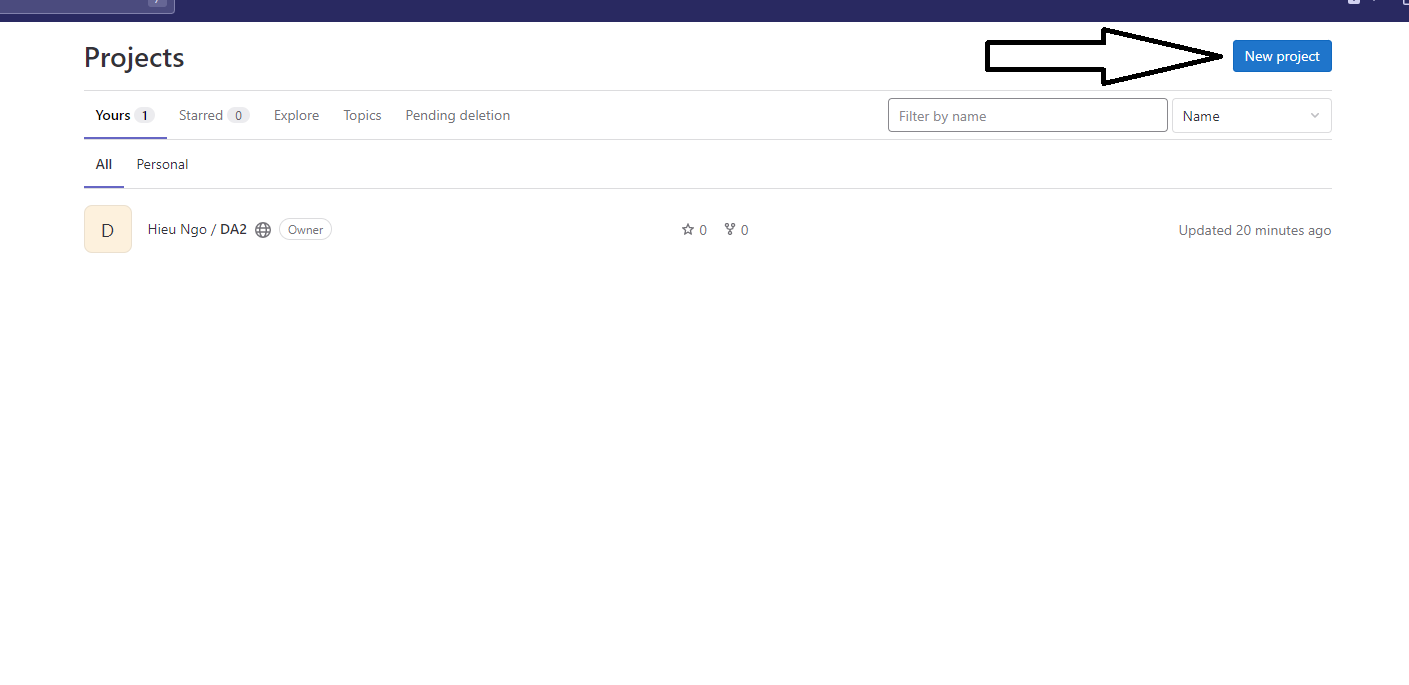
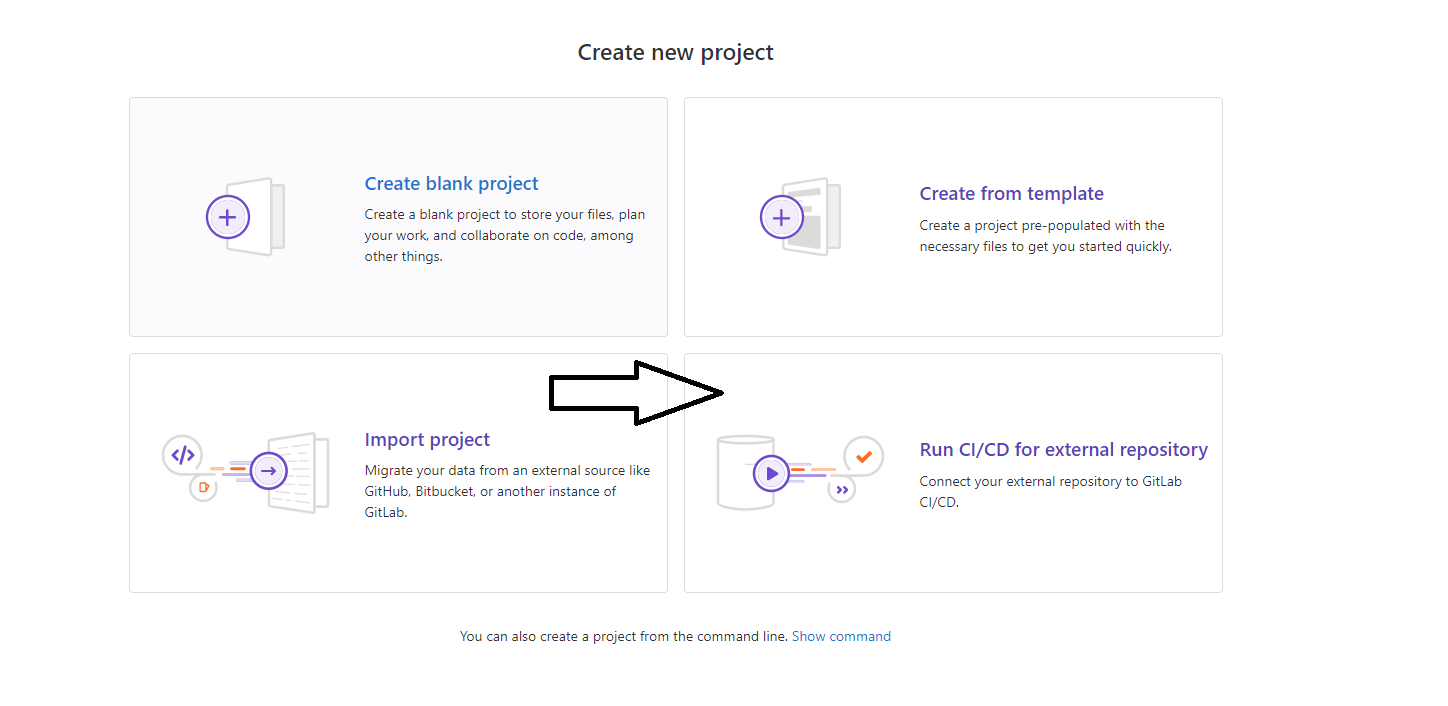
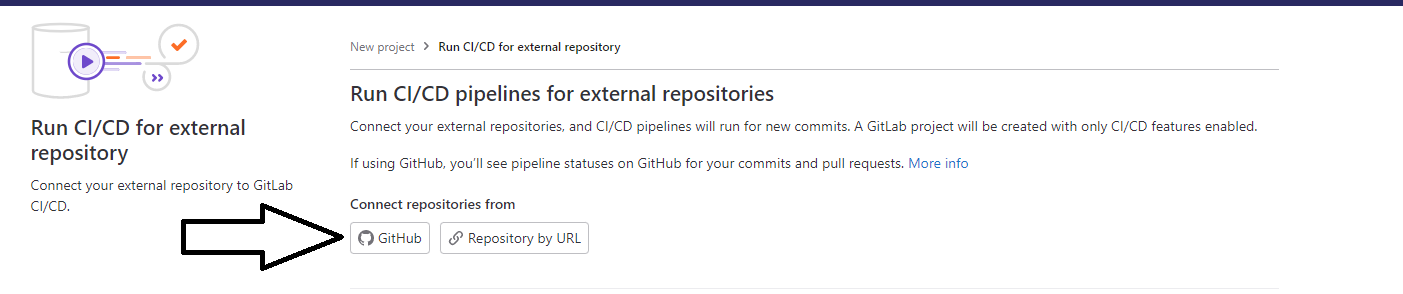
* Đầu tiên để xây dựng trang web có ứng dụng CI/CD vào trong đó thì trong dự án này sẽ sử dụng đến một nền tảng có tích hợp sẵn công cụ CI/CD trong đó là Gitlab . Ban đầu dự án này được khởi tạo ở GitHub và để kết nối GitHub sang Gitlab ta thực hiện như sau :



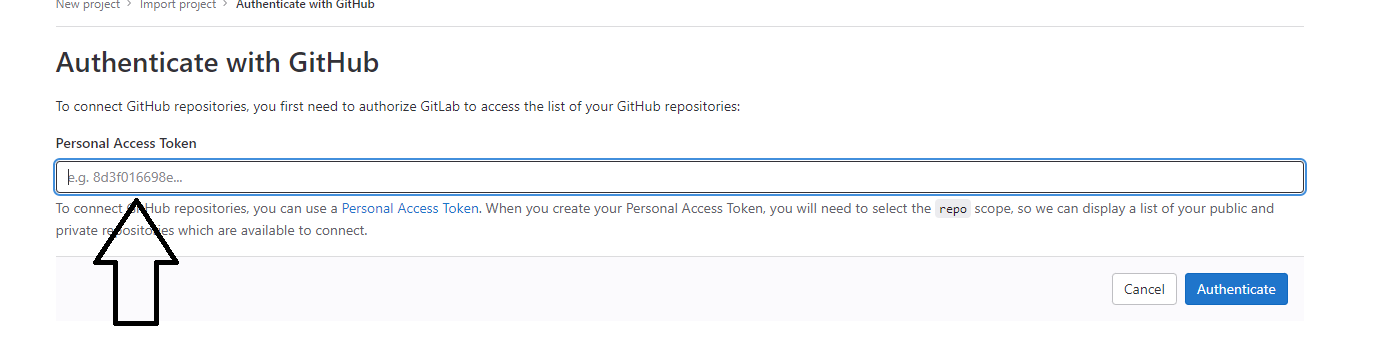
* Ta vào gitlab.com và đăng nhập vào gitlab
* Ở màn hình dự án ta chọn và tạo mới 1 dự án (New Project)



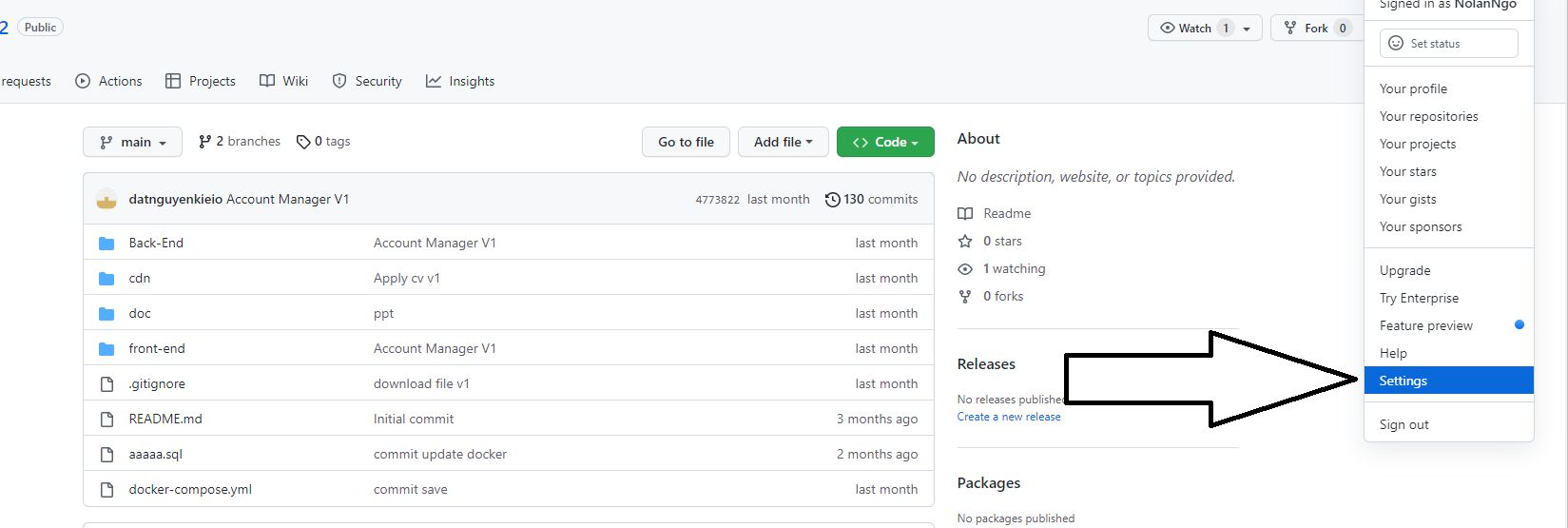
* Ở màn hình tạo mới dự án ta có các lựa chọn (option) như là :
  + Tạo ra 1 dự án trắng
  + Tạo 1 dự án trắng tạo từ mẫu có sẵn
  + Thêm dự án bằng câu lệnh
  + Và chỉ dung CI/CD để run từ dự án khác
* Ở đây ta đã có dự án sẵn ở github nên ta sẽ chọn option cuối là dùng công cụ CI/CD để tích hợp vào dự án có sẵn



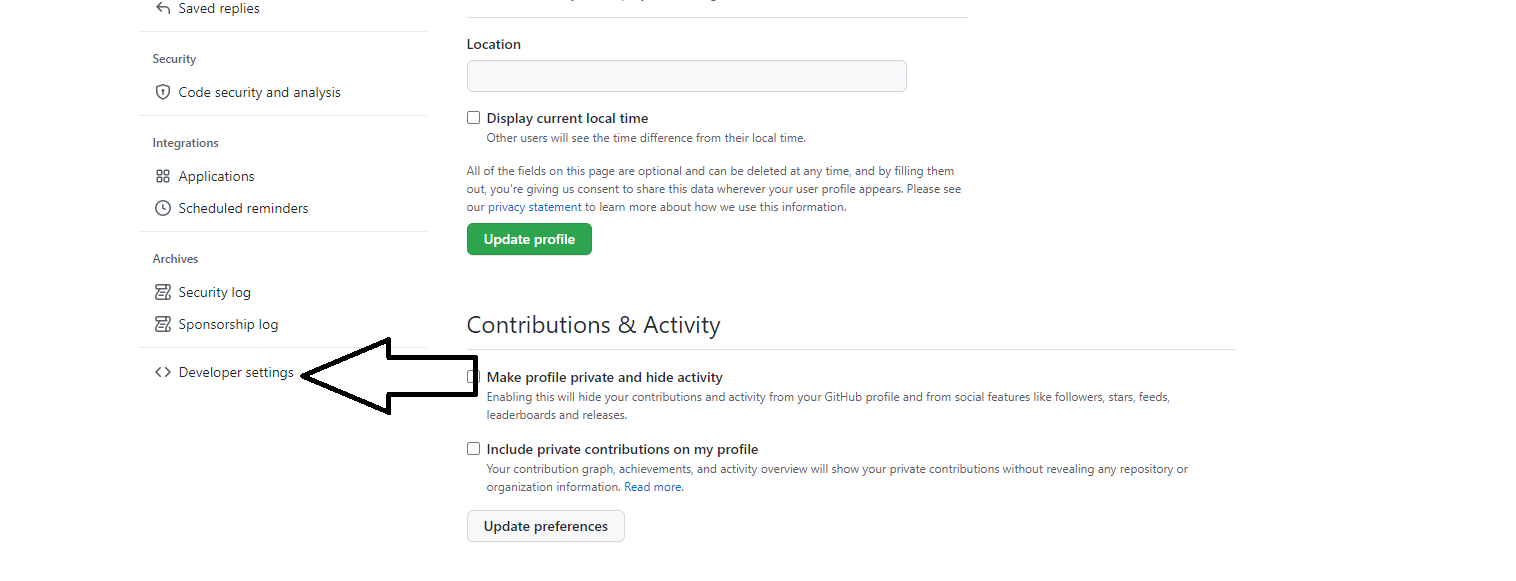
* Tiếp đến ta sẽ chọn kết nối dự án từ nguồn nào thì ta sẽ chọn github ở màn hình này



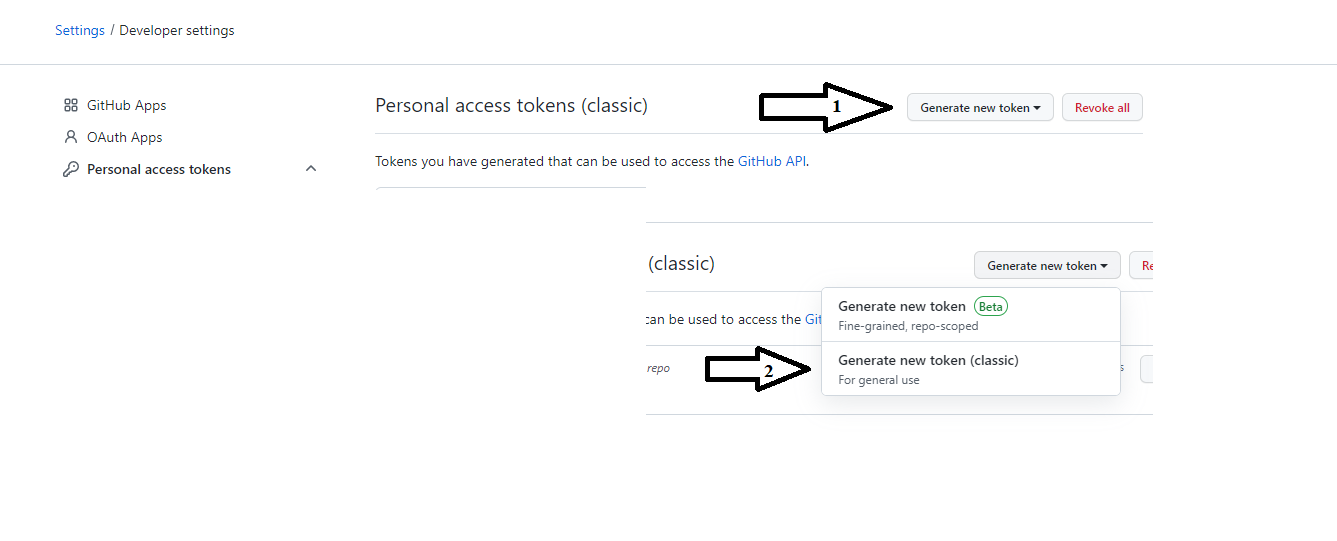
* Ở màn hình này cần token chấp thuận cho gitlab sẽ có quyền thao tác với dự án. Tại đây ta sẽ để màn hình ở đây và quay lại nền tảng github và cấu hình vài thứ để ủy quyền cho gitlab



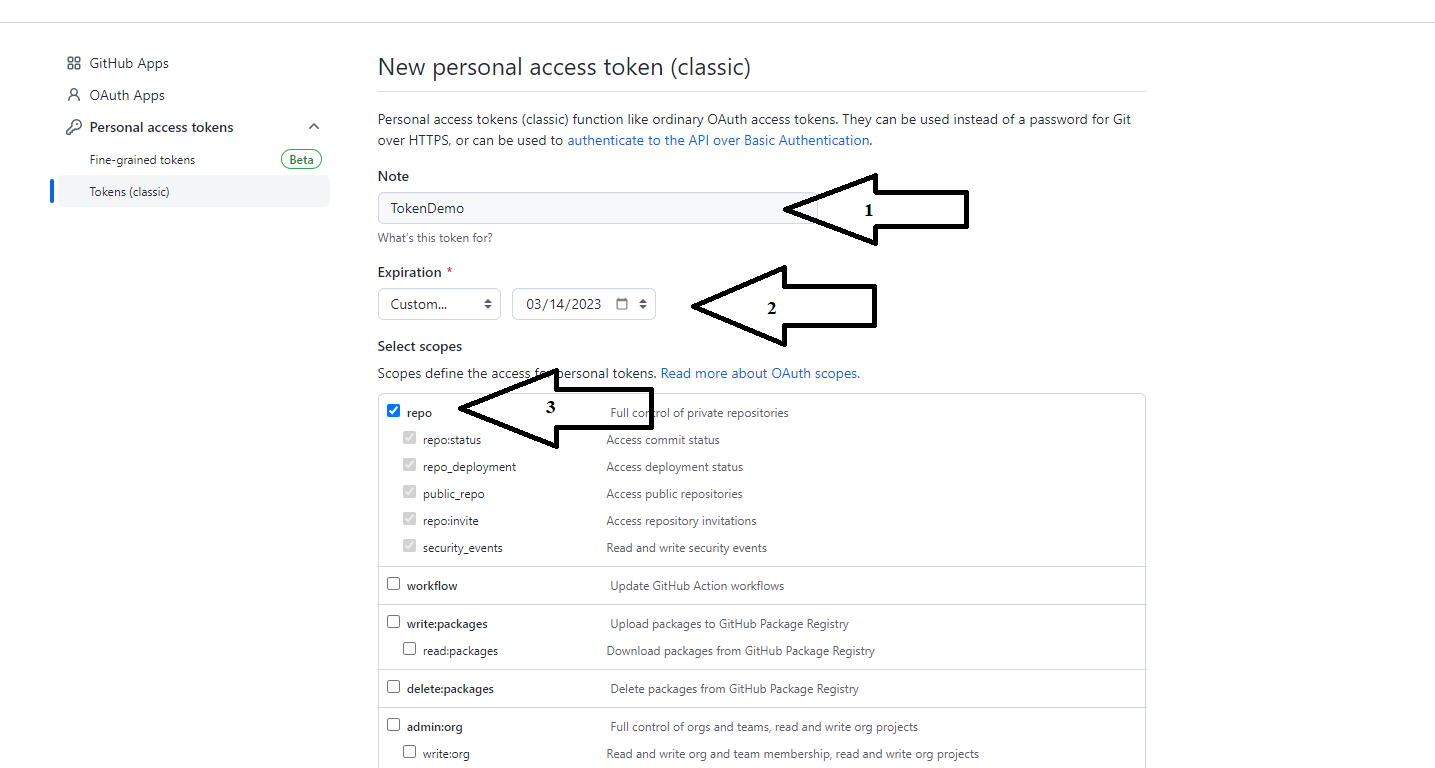
* Ở nền tảng gitlab ta chọn vào settings



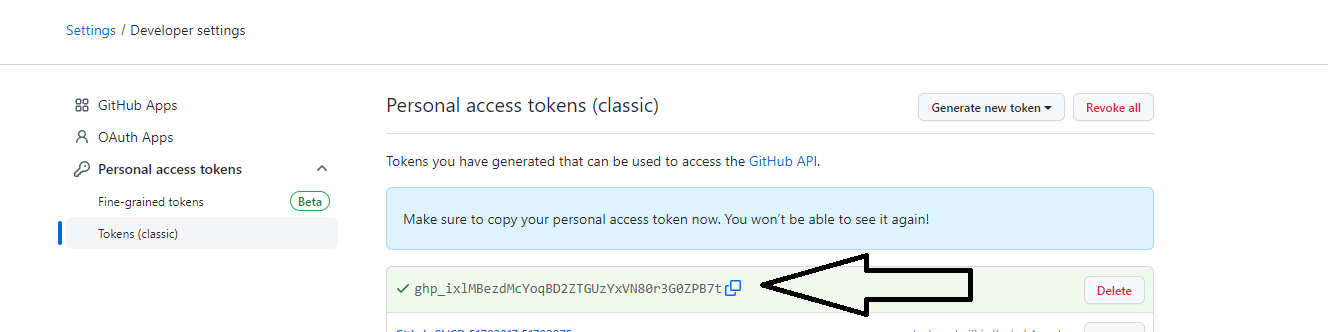
* Tiếp tục ta chọn Developer settings



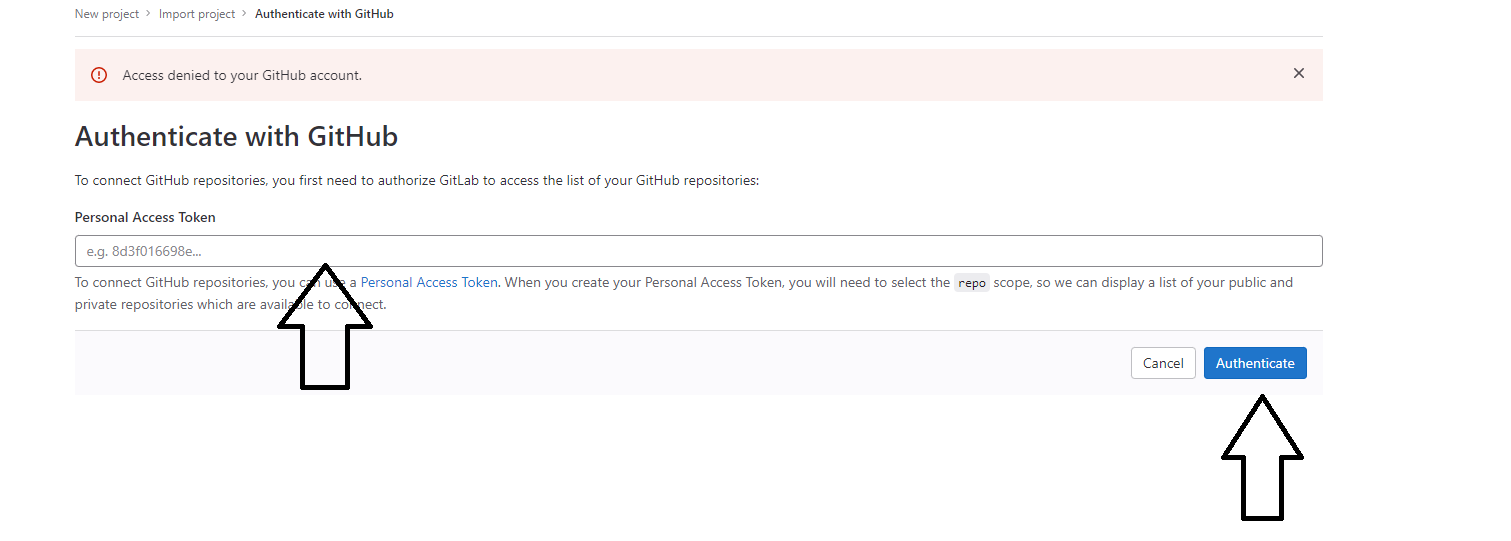
* Tại đây ta thực hiện chọn “Generate new token” để tạo mã ủy quyền cho gitlab (1), Chọn tiếp lựa chọn Generate new token (classic) (2)



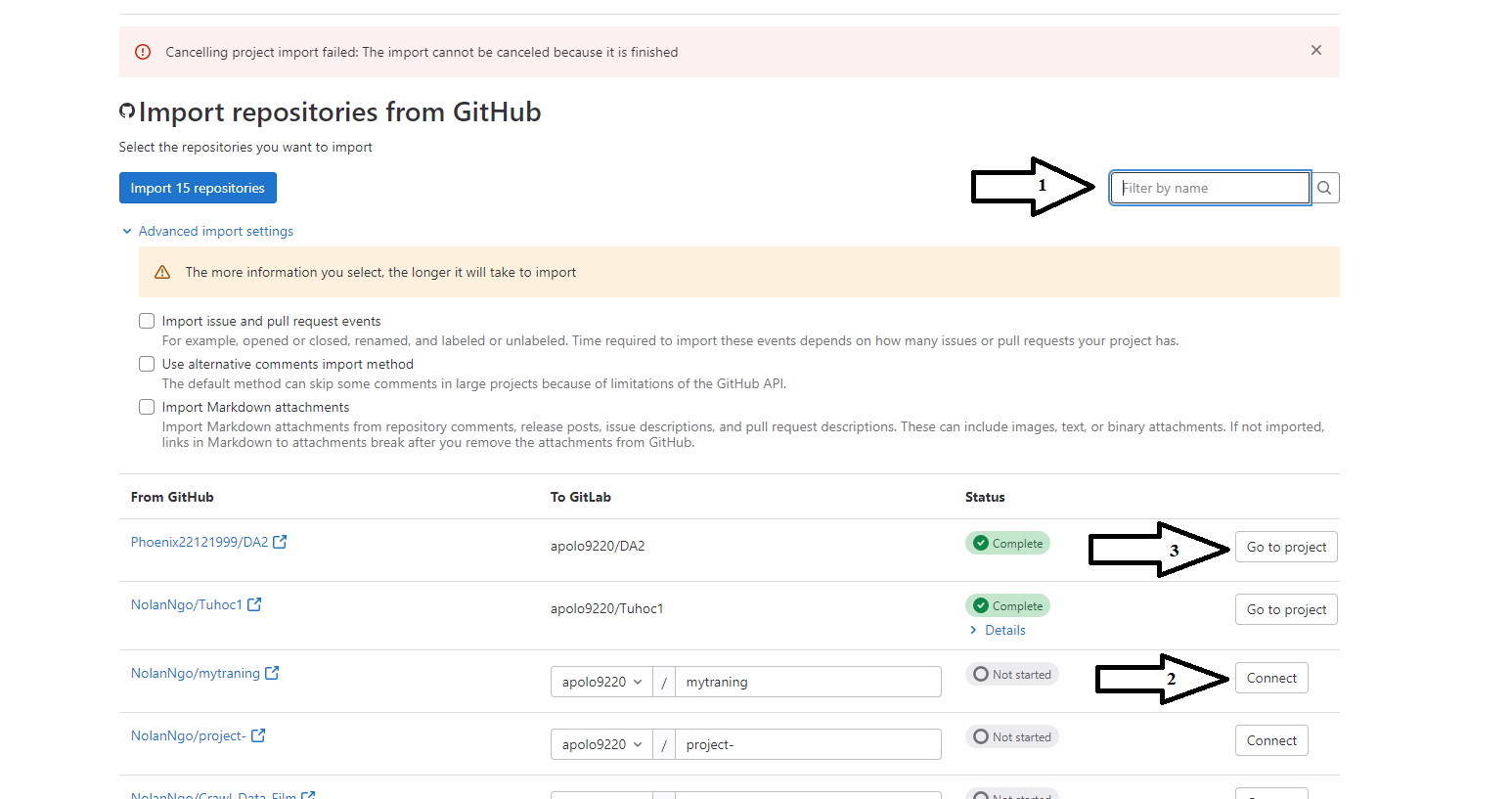
* Ta sẽ đặt tên cho Token (1) , Chọn thời hạn hết hạn của Token này (2) và cuối cùng là lựa chọn các quyền thao tác với dự án của Token này (3)



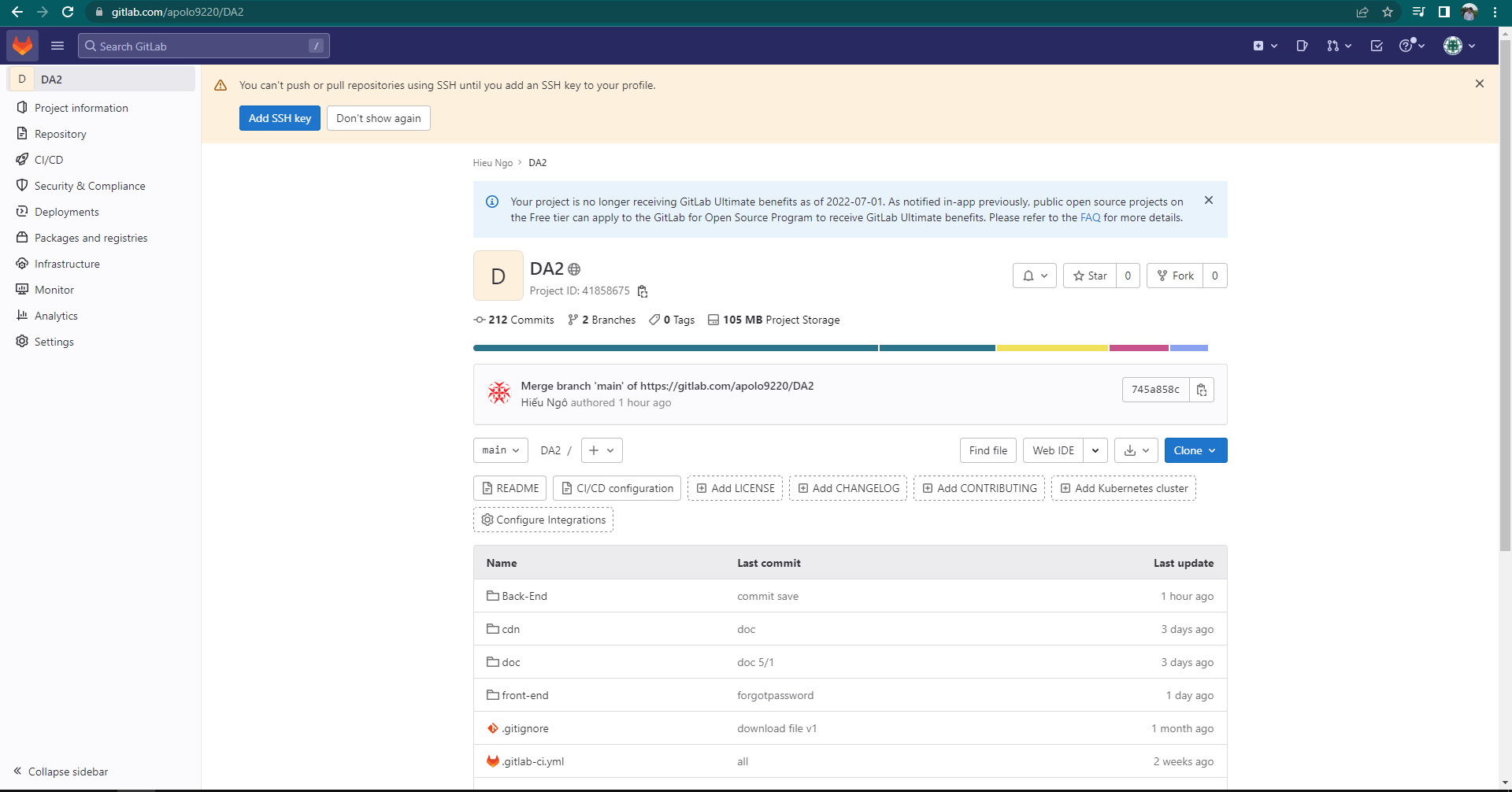
* Sau khi ta tạo xong thì Github sẽ trả cho ta 1 Access Token



- Ta sẽ quay lại màn hình của gitlab lúc nảy yêu cầu Token và đưa Token này vào và chọn Authenticate



* Sau khi ủy quyền xong thì sẽ hiện ra danh sách các dự án mà tài khoản github tạo Access Token có
* Nếu có nhiều dự án ta sẽ sử dụng bộ lọc để lấy được dự án ta cần dùng CI/CD (1), Nếu có dự án ta sẽ chọn Connect và đợi khoản vài phút để Gitlab sẽ lấy dự án từ Github về (2) còn nếu đã kết nối dự án rồi ta sẽ chọn vào Go to project để chuyển hướng đến dự án của ta cần cấu hình CI/CD (3)



* Và như thế ta hoàn thành xong bước đầu tiên là lấy dự án từ Github sang Gitlab sau đó ta thực hiện việc cấu hình chạy CI/CD trên dự án này với nền tảng Gitlab
* Sau khi ta có được dự án ở nền tảng Gitlab ta sẽ tiến hành thiết lập công cụ CI/CD vào trong dự án

Ta sẽ tạo 1 tập tin với tên là .gitlab-ci.yml : Đây sẽ là tập tin chưa mã nguồn được viết theo chuẩn của CI/CD để mỗi lần có sự thay đổi về mã nguồn của dự án mà căng cứ vào đó mà CI/CD dựa vào tập tin này để build cũng như deploy

**variables:**

**IMAGES\_NAME : nolanngo20031999**

**IMAGES\_TAG\_BACK\_END: da2\_backend**

**IMAGES\_TAG\_FRONT\_END : da2\_frontent**

* Tag variables : là nơi ta sẽ khai báo các biến dùng chung bao gồm:
  + IMAGES\_NAME : Tên trên dockerhub
  + IMAGES\_TAG\_BACK\_END : Tên của tag để chỉ nơi lưu trữ images khi build của API
  + IMAGES\_TAG\_FRONT\_END : Tên của tag để chỉ nơi lưu trữ images khi build của Front-end

**stages :**

**- build**

**- deploy**

* Tag stages : là nơi ta sẽ định nghĩa các giai đoạn của quá trình CI/CD thao tác trên dự án, Với dự án trên thì ta sẽ khai báo 2 giai đoạn là giai đoạn đóng gói các tài nguyên (Build) và giai đoạn triển khai đưa dự án lên server (Deploy)

**run\_build\_jobportal\_backend:**

**stage : build**

**only :**

**refs:**

**- main**

**variables:**

**- $CI\_COMMIT\_DESCRIPTION =~ /(api|all).\*/**

**- $CI\_COMMIT\_MESSAGE =~ /(api|all).\*/**

**image : docker:20.10.16**

**services :**

**- docker:20.10.16-dind**

**variables:**

**DOCKER\_TLC\_CERTDIR : "/certs"**

**before\_script :**

**- docker login -u $REGISTRY\_USER -p $REGISTRY\_PASS**

**script:**

**- echo "Runing build image Back End"**

**- cd Back-End**

**- docker build -t  $IMAGES\_NAME/$IMAGES\_TAG\_BACK\_END .**

**- docker push $IMAGES\_NAME/$IMAGES\_TAG\_BACK\_END**

**- echo "Build image Back-End success"**

* Đối với api thì ta sẽ khai báo 1 tên đại diện cho quá trình này
  + Tên này ta sẽ đặt tùy ý nhưng phải thế hiện chức năng của nó hoạt động là gì
  + Stage : ta sẽ khai báo là quá trình này đang trong giai đoạn nào của CI/CD
  + Only : refs : - main có nghĩa là sẽ sử dụng nhánh main trong dự án để tiến hành đóng gói tài nguyên
  + Variables trong quá trình này sẽ khai báo biến và khi CI/CD nhận thấy sự thay đổi của mã nguồn cùng với tình nhắn khi gọp nhánh (merge) , đưa code lên nhánh (push) thì sẽ thực hiện quá trình này đối với phía Back-End
  + Với việc đóng gói (Build images) ta sẽ sử dụng 1 gói khác thuộc về docker để build image này và sử dụng tiện ích dind để build Image của dự án trong chính Image của docker
  + ở tag before\_script : Ta tiến hành login vào docker hub để sau khi tiến hành build images xong thì ta sẽ đẩy images đó lên docker hub để mượn docker hub sẽ lưu trữ images của dự án
  + ở tag script : những đoạn mã trong tag này sẽ hoạt động tuần tự từ trên xuống dưới
  + echo sẽ đẩy ra màn hình message

**run\_deploy\_jobportal\_backend:**

**stage : deploy**

**only :**

**refs:**

**- main**

**variables:**

**- $CI\_COMMIT\_DESCRIPTION =~ /(api|all).\*/**

**- $CI\_COMMIT\_MESSAGE =~ /(api|all).\*/**

**before\_script:**

**- chmod 400 $SSH\_KEY**

**script:**

**- echo "Runing deploy api"**

**- echo "Runing deploy api"**

**- ssh -o StrictHostKeyChecking=no -i $SSH\_KEY root@159.223.54.199 "**

**docker stop container backendda2 || true && docker rm backendda2 || true &&**

**docker rmi $IMAGES\_NAME/$IMAGES\_TAG\_BACK\_END || true &&**

**docker login -u $REGISTRY\_USER -p $REGISTRY\_PASS &&**

**docker run -d -p 4000:4000 --name backendda2 --hostname backendda2 $IMAGES\_NAME/$IMAGES\_TAG\_BACK\_END"**

* Quá trình tiếp theo thì cũng khai báo tương tự như quá trình build như thay đổi ở phần stage : sẽ là giai đoạn deploy của CI/CD
* Trước khi chạy đoạn script ta sẽ khai bảo mở quyền trên server linux đã thuê bằng câu lệnh **chmod 400 $SSH\_KEY với $SSH\_KEY** là key mà ta sử dụng ssh\_key để tạo và connect với server và quyền 400 là chỉ đọc và không được thực hiện các thao tác ghi gì cả
* Khi bắt đầu deploy ta sẽ thực hiện 1 loạt các câu lệnh để không xảy ra lỗi là ssh **StrictHostKeyChecking = no** sẽ giúp ta trỏ đến server và không bị server kiểm tra key đăng nhập
* Thực hiện dùng docker để tạm ngưng container của API , Xóa container đó đi , và đồng thời xóa images của API có ở server
* Ta Tiến hành đăng nhập vào docker hub
* Và chạy lại container với image khi chạy docker sẽ kiểm tra nếu ở máy chủ chưa có images nào tên đó sẽ thực hiện thao tác là pull images từ docker hub về và build image đó thành container tại máy chủ và như thế sẽ kết thúc quá trình deploy 1 image từ 1 sự thay đổi của mã nguồn dự án do developer thay đổi

Docker File

* API :
* FROM node:14.17.3-alpine as da2
* WORKDIR /app
* COPY package\*.json ./
* COPY prisma ./prisma/
* RUN  npm install
* COPY . .
* ENV PORT=4000
* ENV ACCESS\_TOKEN="51702075DAT\_51702017HIEU"
* ENV CDN\_URL="http://159.223.54.199:3002/"
* ENV DATABASE\_URL="sqlserver://159.223.54.199:1433;database=QLHS\_TDTU;user=sa;password=51702017Hieu51702075Dat;encrypt=true;trustServerCertificate=true;"
* ENV CLIENT\_ID\_TDTU="454375430021-ins1u408up6msulgvnffrl4f9hn1nlt5.apps.googleusercontent.com"
* ENV CLIENT\_ID\_NORMAL="772360898971-ufhi78bgjm44o9r1megrah02lnfb8f8s.apps.googleusercontent.com"
* EXPOSE 4000
* CMD ["npm" , "run" , "start"]
* Ở mỗi Back-end, Front-end , CDN ta sẽ đính với mỗi site là 1 file Dockerfile thì và nhiệm vụ của nó là mỗi khi thao tác với docker để tiến hành đóng gói thì docker sẽ đọc nội dùng trong Dockerfile ở site cần đóng gói và thực hiện các đoạn mã được viết trong đó để đóng gói và lấy những tài nguyên cần có để đính kèm vào trong gói mà ta cần để chạy nó ở máy khác hoặc ở server mà không bị lỗi. Ý nghĩa của từng dòng như sau
  + From :node 14.17.3-alpine as da2 có nghĩa là ta sẽ sửa dụng tài nguyên cho site này là node phiên bản 14.17.3 và đặt tên cho nó là da2
  + WORKDIR /app có nghĩa là nơi mà thực hiện thao đóng gói sẽ là thư mục /app
  + COPY package\*.json/ được hiểu là sẽ tiến hành đưa các gói tài nguyên được khai báo trong package và package-lock vào trong gói
  + COPY prisma ./prisma/ Ở Dự án này sử dụng công cụ ORM là prisma được đặt trong thư mục ./prisma nên lệnh này được dùng để copy tài nguyên của prisma vào trong gói
  + Npm install khi sử dụng node ở bất kì máy nào ta sẽ phải cài đặt bằng câu lệnh này nên khi đóng gói ta sẽ tiến hành cài đặt trước các tài nguyên cho dự án và sau đó đóng gói phần tài nguyên này kèm theo site dự án
  + Các biến có ENV = có nghĩa là sẽ khai báo biến môi trường để khi gói được chạy sẽ sử dụng các biến môi trường này vì file .env khai báo biến ở nodejs nhưng khi ta đóng gói docker sẽ không đóng gói file .env này nên ta cần khai báo biến
  + Expose 4000 có nghĩa là khi docker đóng gói xong site dự án này sẽ được đẩy ra là cổng (port) 4000
  + CMD [“npm” , “run” , “start”]: đặt các lệnh có thể hoạt động trong quá trình docker chạy gói

FROM node:14.17.3-alpine AS da2frontend

WORKDIR /app

COPY package\*.json ./

RUN yarn install

COPY . .

RUN yarn

RUN yarn build

FROM nginx:stable-alpine

COPY --from=da2frontend /app/build /usr/share/nginx/html

RUN rm /etc/nginx/conf.d/default.conf

COPY --from=da2frontend /app/nginx/nginx.conf /etc/nginx/conf.d

# Containers run nginx with global directives and daemon off

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

* Ở site Front-end sẽ có một chút thay đổi do Front-end chạy bằng ReactJs nên DockerFile của Front-end sẽ như sau
* From cũng như Back-end sẽ sử dụng bản node là 14.17.3
* Địa chỉ làm việc sẽ là /app
* Tiến hành việc lấy nhưng tài nguyên trong package , package-lock vào trong gói
* Tiến hành yarn install để cài đặt các tài nguyên được khai báo trong packge.json
* Và copy hết các tài nguyên vừa install đó
* Và Ở reactJS khi muốn deploy ta sẽ không sử dụng là npm run start , yarn start mà ta sẽ sử dụng lệnh build lúc này ReactJS sẽ build dự án của ta thành 1 folder build trong đó sẽ nén mã nguồn của ta lại thành các file là .html , css , .js để khi react hoạt động chỉ trỏ đến thư mục public và lấy ra tất cả file .html .css và .js để chạy trên trình duyệt web
* Sau khi build thì ta tiến hành giai đoạn tiếp theo trong Docker file thì để deploy được dự án thì sẽ sử dụng một nền tảng khác đó là nginx
* Từ gói ta đã đặt tên docker sẽ copy toàn bộ file build ở gói sang thư mục mặc định mà khi ta start nginx lên là nginx sẽ trỏ thẳng vào thư mục mặc định này ở đây là usr/share/nginx/html
* Trong nginx sẽ có 1 file thiết lập mặc định nên ta sẽ tiến hành xóa file thiết lập mặc định này và tiếp theo là lấy toàn bộ file thiết lập của bản thân ra và đưa vào đó
* Gói sau khi đóng sẽ được start ở port 80
* Và các coogn lệnh cho phép hoạt động trong suốt quá trình chạy docker là CMD [“nginx “ “-g” “daemon-off”]

Phía trên là file thiết lập của từng site là Front-end và Back-end khi start dự án sẽ tiến hành start cả 2 site cùng một lúc thì khi này ở dự án nơi chứa cả 2 site ta sẽ tạo thêm 1 file là Docker-Compose.yml

version: '2.2'

services:

    backend:

      build:

        context : ./Back-End

        dockerfile : ./Dockerfile

      # image : 'backendda2'

      container\_name: backendda2

      ports :

        - 4000:4000

    frontent:

      build :

        context : ./front-end

        dockerfile: ./Dockerfile

      # image : 'frontda2'

      container\_name: frontda2

      ports :

        - 80:80

      links :

        - "backend:be"

    cdn:

      build:

        context : ./cdn

        dockerfile : ./Dockerfile

      container\_name: cdnda2

      # image : 'cdnda2'

      ports :

        - 3002:3002

* docker-compose.yml đóng vai trò sẽ chữa các thiết lập mà ta sẽ khai báo khi ta muốn khởi động dự án với những gói nào và các gói riêng biệt được start ở cổng (port) số mấy
  + version tag được dùng với mục đích là định danh cho quá trình khởi động dự án
  + services tag là nơi ta sẽ thiết lập các đoạn mã để build cũng như run các image
  + Với dự án này file docker-compose bao gồm 3 services gồm các site là back-end , front-end , cdn (Là nơi chứa các tập tin được upload hoặc là hình ảnh)
  + Ví dụ như với site back-end sẽ tiến hành build ở địa chỉ là /back-end và file được docker trỏ đến đọc và build image là Dockerfile được đặt ở site back-end và tương tự các site khác ta khai báo tương tự như vậy và kèm theo port mà người cấu hình docker-compose này muốn site đó được start ở cổng (port) số mấy với phần port ta đơn giản hiểu là 3001 : 3001 có nghĩa là ánh xạ cổng 3001 của máy chủ vào cổng 3001 của docker