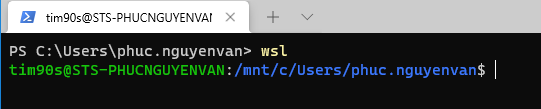
**WSL**

Lệnh sau sẽ hiển thị tất cả các linux trong máy của mình cho WSL

**wsl -l –all**

Khi ta ở cửa sổ terminal hay powershell nếu muốn vào dòng lệnh linux ta chỉ việc gõ wsl sẽ như hình bên dưới.

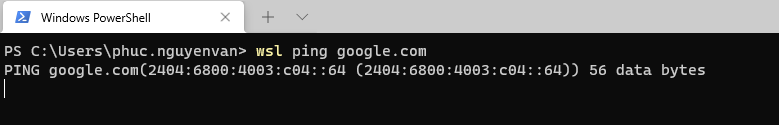


Để thoát khỏi cửa sổ linux hiện tại ta gõ lệnh **exit** nó sẽ trở về màn hình lệnh powershell hay terminal ban đầu của ta

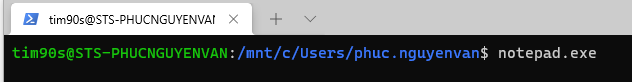
Một điểm lợi khi ta sử dụng WSL đó là ta có thể mix giữa việc sử dụng lệnh linux và lệnh window trong cùng một dòng lệnh, nó sẽ hiểu lệnh đó là của window hay linux.

**Nếu ta đang ở cửa số dòng lệnh của window ta muốn gọi đến lệnh của linux thì ta cần chỉ rõ trước lệnh đó là wsl**

**Như hình bên dưới thì ta đang gọi lệnh ping trong linux và ping tới google.com**

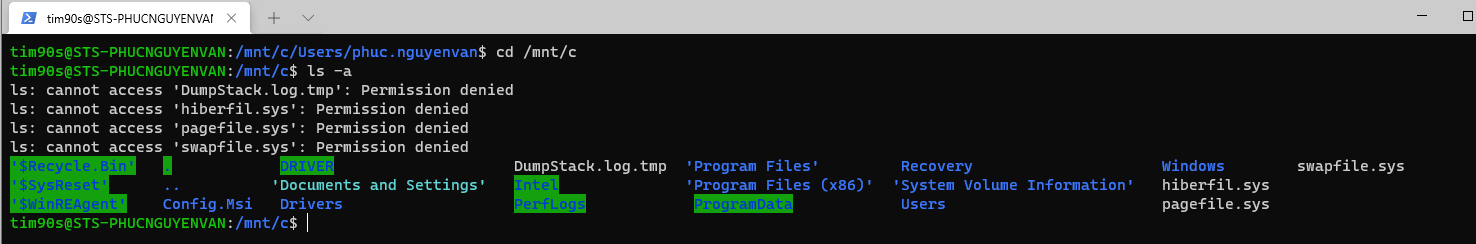


**Nếu ta đang trong linux ta gọi đến lệnh của window thì nó cũng sẽ tự hiểu và run lệnh đó. Ví dụ bên dưới là ta đang trong màn hình linux và muốn mở trình soạn thảo văn bản notepad của window.**



**Trong wsl thì nó sẽ mount ổ C của ta vào đường dẫn** **/mnt/c/**

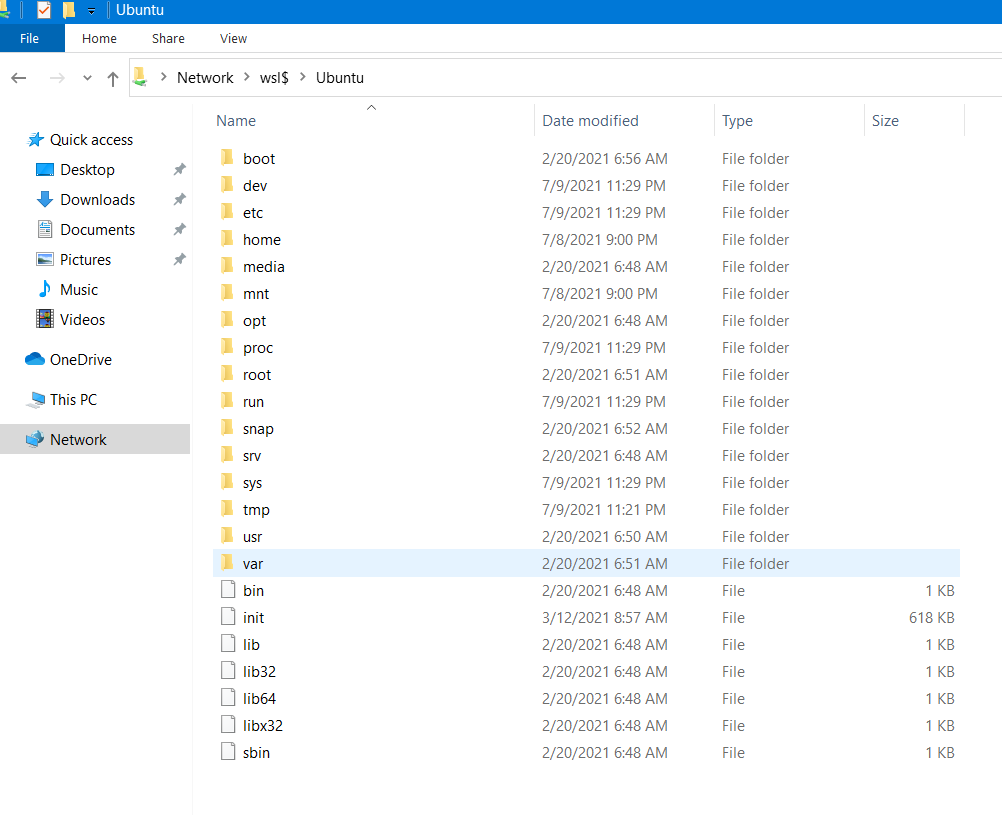
Ta có thể nhìn thấy bên dưới ta vào ổ C của windows trong linux và liệt kê các thư mục của ổ C trong windows.



Từ windows nếu ta muốn truy cập vào các file và folder của distro nào đó ta dùng lệnh sau trong file explorer

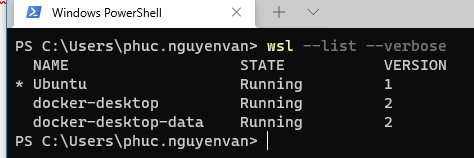
\\wsl$\Ubuntu

Ta sẽ thấy kết quả như sau:



Để kiểm tra version của wsl ta dung lệnh sau

wsl –list --verbose



Để chạy được docker ta phải là version cho distro đó là 2.

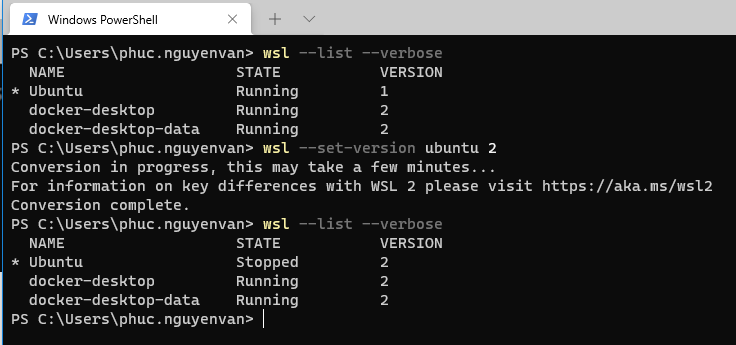
Để convert distro từ vesion 1 sang version 2 ta dung lệnh sau

**wsl —set-version ubuntu 2**

Nếu ta có 2 version chạy song song thì ta dung lệnh sau để set default về version 2

**wsl—set-default-version 2**

Sau khi chạy xong ta có thể kiểm tra và thấy kết quả như sau:



**Docker note**

1. **Linux command**

Để xem thông tin của docker ta dùng lệnh

**docker infor**

Nếu ta chạy docker với wsl thì nó sẽ sử dụng hết cpu của máy và ram của máy nó đang host khác với ta chạy docker bằng hyper V

1. Chain command

Câu lệnh sau sẽ tạo thư mục tên là test và khi tạo xong sẽ đi vào thư mục đó và xuất ra dòng chữ là done.

**mkdir test ; cd test ; echo done**

khi dùng dấu; thì gặp nỗi chỗ lệnh đó nó sẽ vẫn đi tiếp lệnh tiếp theo.

Nếu ta muốn khi gặp lỗi và dừng lại ngay tại lệnh đó và không thực hiện lệnh tiếp theo thì ta làm như sau:

**Mkdir test && cd test && echo done**

Câu lệnh bên dưới sẽ tạo ra thư mục tên là test nếu thư mục tên là test đã có sẵn rồi thì nó sẽ in ra thông báo lỗi là ‘thu muc test da co’

**mkdir test || echo ‘thu muc test da co’**

Câu lệnh bên dưới sẽ hiển thị tất cả các thư mục trong thư mục bin và hiển thị theo dạng từng trang thay vì show all hết.

Để sử dụng dược less thì bạn cần phải cài less vào. **Để thoát khỏi lệnh less tì ta nhấn phím q**

**ls /bin | less**

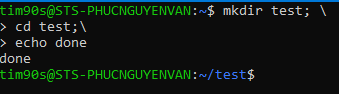
Hiển thị các thư mục ở khúc đầu trong thư mục bin

**ls /bin | head**

Hiển thị các thư mục khúc cuối trong thư mục bin

**ls /bin |tail**

để viết mỗi câu lệnh sau đó ngắt xuống dòng thay vì nằm trên 1 dòng cho dễ nhìn thì ta làm như sau (dung dấu \ sau dấu ; )



1. Environment variable

Để hiện tất cả các biến môi trường trong linux ta dung lệnh sau

**printenv**

Để hiện một giá trị của một biến môi trường ta dung lệnh như sau, ví dụ bên dưới là in ra giá trị của biến PATH

**printenv PATH**

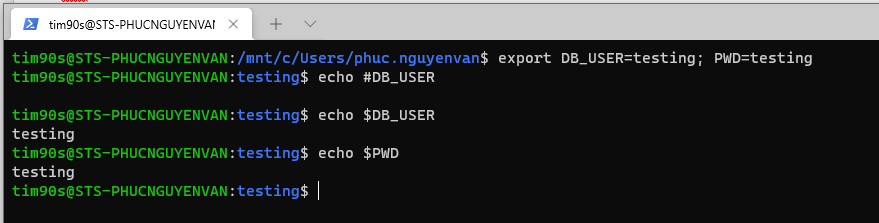
Ngoài ra ta còn có thể sử dụng lệnh echo để in ra giá trị của biến môi trường, nhưng phải thêm ký tự $ trước biến môi trường đó.

**printenv $PATH**

Để tạo ra một biến môi trường ta dùng lệnh export ( biến này sẽ chỉ có hiệu lực tại phiên làm việc hiện tại, nếu đóng phiên làm việc biến sẽ bị mất)

Ta xem xét ví dụ bên dưới là khai báo 2 biến môi trường và sau đó in ra giá trị của nó là DB\_USER có giá trị là testing và biên PWD có giá trị là testing

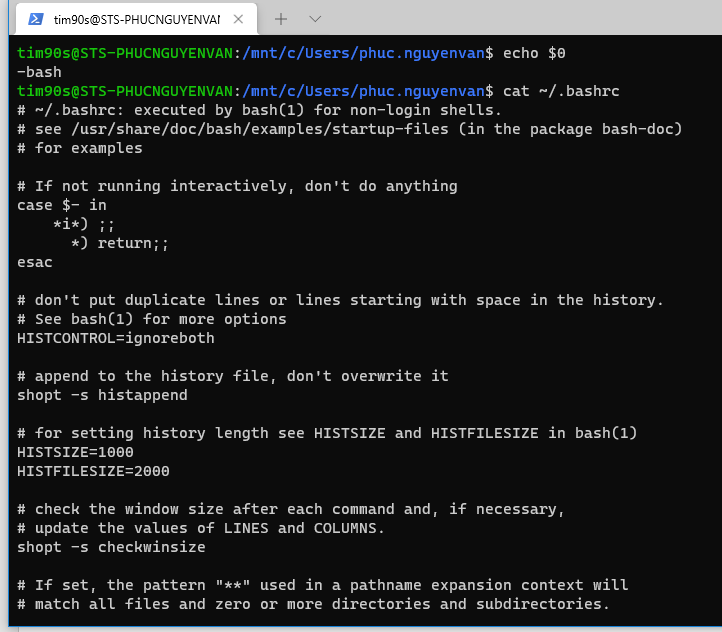
**export DB\_USER=testing; PWD=testing**



Để khai báo biến môi trường trong linux mà không bị mất khi hết phiên làm việc ta khai báo trong bash

Như hình bên dưới ta xem phiên bản linux hiện tại nó lưu biến môi trường là ở **bash hay zsh**

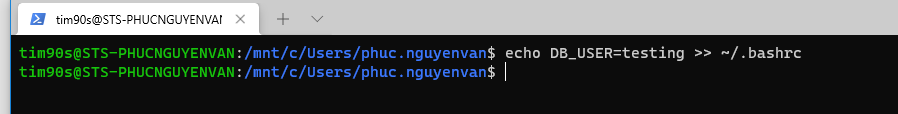
và sau đó là xem các biến môi trường đang được lưu trong bash



Tạo mới biến môi trường tên là DB\_USER và lưu nó vào trong bash để khi hết phiên làm việc sẽ không bị mất.

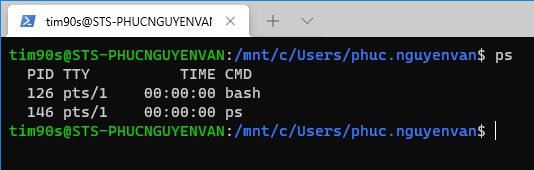
Khi kiểm tra lại biến đó đã được lưu vào trong bash hay chưa thì nếu khi in ra không thấy biến đó thì ta dung lệnh source ~/.bashrc để update lại cái terminal hiện tại

**(Lưu ý là không lưu nhưng thông nhin nhạy cảm vào thư mục này)**



1. Quản lý process trong Linux

Để list các process



Trong PID là process ID

TTY là TeleTypperiter nếu giá trị pts/1 thì có nghĩa là có 1 terminal khác đang kết nối và thực thi.

CMD là command name

Ta có lệnh **kill {PID}** để terminate một process nào đó theo PID của nó.

1. **Docker container**
2. dockerfile

**FROM :** Đâu là image sẽ được sử dụng làm nền để xây dựng nên image của bạn, các image này sẽ được tải về từ public repository (public trên docker hub) hay private repository.

**WORKDIR :** Chỉ định thư mục làm việc (không ghi gì hay dấu . thì là mình sẽ làm việc rên thư mục gốc)

**COPY :** copy hết các file mà mình chỉ định vào thư mục làm việc được tạo trên WORKDIR.

**ADD :** giống với copy nhưng nó có thể lấy một file nào đó trên mạng, hoặc một cái host nào.

**RUN :** chỉ định ra sau khi mọi thứ đã sẵng sàng thì mình muốn chạy lệnh nào để start ứng dụng

**ENV :** thiết lập biến môi trường cho image đang setup

**EXPOSE :** chuyển port trong docker ra ngoài máy vật lý

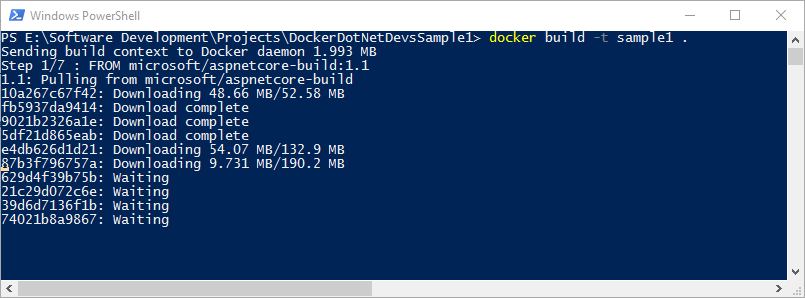
**USER :** Tạo user, xác định quyền user đó, để user đó chạy ứng dụng trong docker của ta

**CMD :** Xác định các câu lệnh cần sử dụng khi bắt đầu một containter lúc runtime

**ENTRYPOINT**

1. Docker base image

docker build -t tên\_Image đường\_dẫn\_tới\_docker\_file



Ví dụ

docker build -t {app name} . : build image với {app name} tại thư mục hiện hành (.)

-t:tag name

Link detail: <https://docs.docker.com/engine/reference/commandline/build/>

Sau khi tạo được image từ image alpine thì ta có thể chạy lên bằng lệnh sau:

docker run -it angular-app sh

1. copy file / directories

WORKDIR /app : tạo thư mục app và đi vao trong nó luôn

COPY thư\_mục\_nguồn thư\_mục\_đích

Trong đó :

thư\_mục\_nguồn:

là nơi chưa sứa source cần được copy

[“file1.json”] chỉ copy file1.json

[“file1.json”, “file2.json”] copy 2 file là file1.json và file2.json

[“file\*.json”] copy tất cả có kiểu là json và có tên bắt đầu là file

[“hello world.json”] copy file hello world.json, nếu file tên có khoản trắng thì phải làm như vầy.

Ta xét thứ tự của 2 lệnh sau:

WORKDIR /app  
 COPY . . #COPY tất cả các file thư mục hiện hành vào thư mục app bên trên

Với lệnh ADD thì nó sẽ tải file với đường dẫn ta chỉ và copy vào trong folder ta chỉ định

Ví dụ: ADD <http://abc.com/file1.json> . # có nghĩa là nó sẽ tải cái file1.json từ đường dẫn kia vào bỏ vào thư mục app

Nếu ta mốn ignore một thư mục nào để docker không copy ta tạo một file tên là .**dockerignore** và đặt vào đó tên thư mục mà ta muốn ignore

1. docker commands

Ta dùng lệnh RUN để chỉ định rằng ta muốn thực hiện lệnh nào sau khi copy source xong, ta có thể đặt nhiều lệnh khác nhau bằng cách xuống hàng với các lệnh đó và bắt đầu bằng cụm từ RUN

1. docker environment

Để khai báo một biến môi trường ta dung lệnh ENV ví dụ như bên dưới là ta khai báo biến môi trường tên là API\_PATH

có 2 cách là ta dung dấu = hoặc khoản cách

ENV API\_PATH=http://api.localhost hoặc ENV API\_PATH http://api.localhost

1. expose port
2. cài đặt user

RUN addgroup app && adduser -S -G app app #tạo group tên app, rồi tạo user tên app bỏ vào group app và user này là system user.

USER app # dùng user app vừa tạo để sử dụng trong docker

1. Docker Entrypoint

Ta move 2 line code dưới đây lên trên và nằm xuống dòng lệnh chỉ thị FROM, để cho khi nó kéo image về và tạo image thì nó sẽ tạo user và group và khi đến tới dưới thì nó sẽ tạo folder app được gán quyền cho user đó.

**RUN addgroup app && adduser -S -G app app**

**USER app**

Có 2 dạng để thực thi câu lệnh CMD

#shell form

**CMD npm start**

Execute form, dạng này tốt và chạy nhanh , đây là best practice.

**CMD [“npm” “start”]**

Nếu mà có 2 hay 3 lệnh giống nhau thì cái dòng cuối cùng sẽ được chạy

Như ví dụ bên dưới thì câu lệnh cuối cùng sẽ được chạy

**CMD [“npm” “start”]  
CMD [“npm” “start”]  
CMD [“npm” “start”]**

ENTRYPOINT và CMD về cơ bản là như nhau nhưng đối với ENTRYPOINT thì sẽ không bị override lại như CMD bằng cách truyền vào theo con đường là gọi lệnh **docker run**

Nếu muốn override lại một lệnh đã được khai báo bằng ENTRYPOINT thì ta làm như ví dụ sau:

**docker run angular-app --entrypoint Lệnh\_Muốn\_overrride**

1. Speeding

|  |  |  |
| --- | --- | --- |
| Ban đầu | Đổi thành |  |
| FROM  node:14.16.1-alpine3.13  RUN *addgroup* *app* *&&* *adduser* *-S* *-G* *app* *app*  USER *app*  WORKDIR */app*  COPY *.* *.*  RUN *npm* *install*  ENV API\_PATH=*http://api.localhost*  EXPOSE *4200*  CMD *npm* *start* | FROM  node:14.16.1-alpine3.13  RUN *addgroup* *app* *&&* *adduser* *-S* *-G* *app* *app*  USER *app*  WORKDIR */app*  COPY *package\*.json* *.*  RUN *npm* *install*  COPY *.* *.*  ENV API\_PATH=*http://api.localhost*  EXPOSE *4200*  CMD *npm* *start* | Chỉ khi nào file package.json thay đổi mới chạy lại lệnh **npm install**  **Ý nghĩa của việc này là chỉ chạy lại những thứ thay đổi còn không thì giữ lại không cần phải lúc nào cũng restore và override lại.** |

1. Removing image

Lệnh sau sẽ remove các image không cần thiết (rác)

**docker image prune**

Nhưng nếu có container nào đang chạy image rác thì lệnh bên trên thì các image rác đó sẽ không thể nào xóa đi được.

Do vậy mà ta cần chạy lệnh sau để xóa các container rác.

**docker container prune**

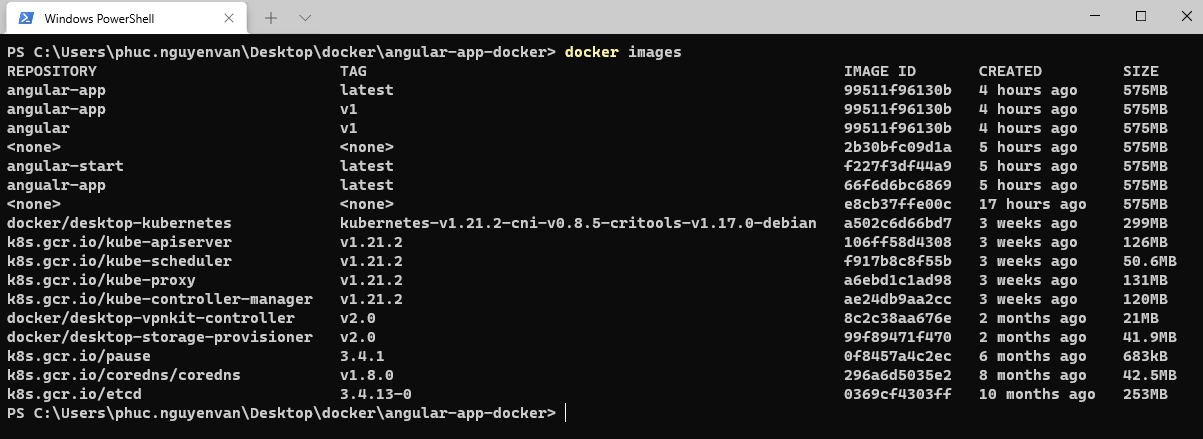
Để remove một image cụ thể nào ta dung lệnh sau

**docker image rm 434a18e7b688 (tên image hoặc ID của image đó)**

Khi ta đánh tag thì nó sẽ tạo ra một version mới của image đó theo tên tag mới.

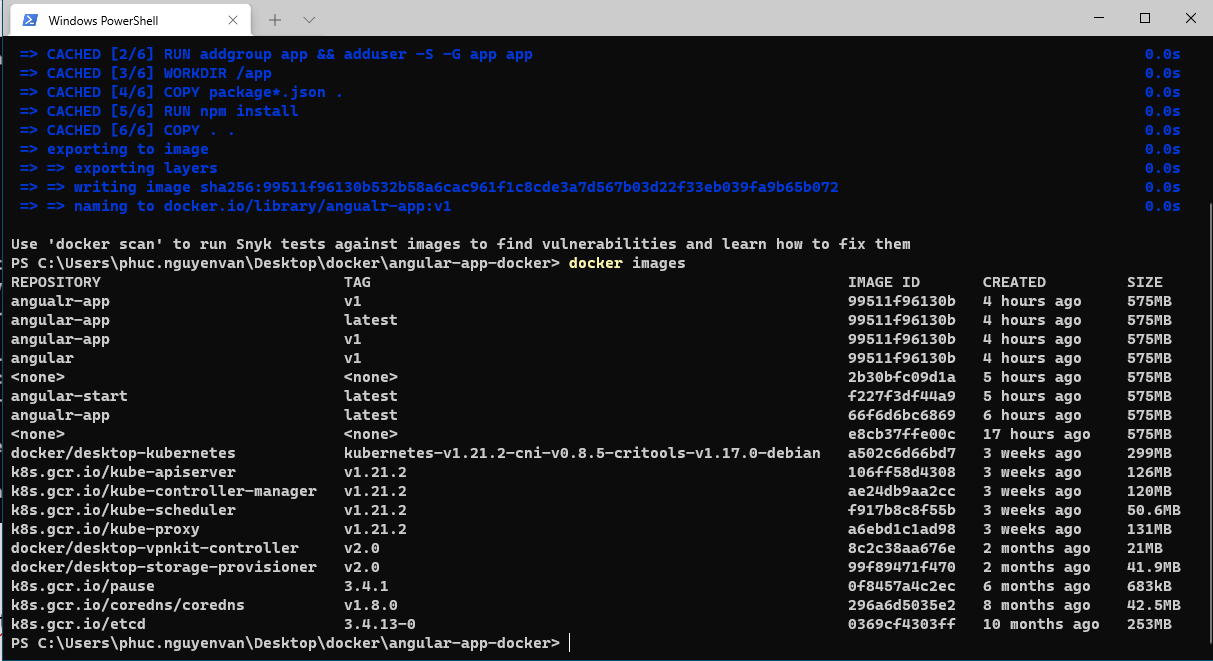
câu lệnh bên dưới nói ra từ image angular:v1 ta đánh tag để có một image khác tên là angular-app:v1

**docker image tag angular:v1 angular-app:v1**



Ta có thể đánh tag một image bằng lệnh docker build, mỗi lần build ta sẽ chỉ ra một tag/ version cho image được build đó

**docker build -t angualr-app:v1 .**



Khi mà ta đánh tag thì cái IMAGE ID sẽ trùng nhau do vậy ta muốn chọn image nào thì ta chọn theo tên rồi tới tag của nó.

Khi mà 2 image trùng image ID mà sau khi đánh tag thì điều đó có nghĩ là 2 images đó là cùng một lần build.

Câu lệnh bên dưới là đổi image có id là image\_Id thành image latest với tên là ten\_image

**docker image tag image\_Id ten\_image:latest**

1. sharing image

Dùng lệnh docker login để login

dung lệnh docker push tim90/angular-app:v2 để push một image lên trên docker hub (recommend là ta đặt tên image giống tên với repositoy của ta trên docker hub)

Repo trên docker hub

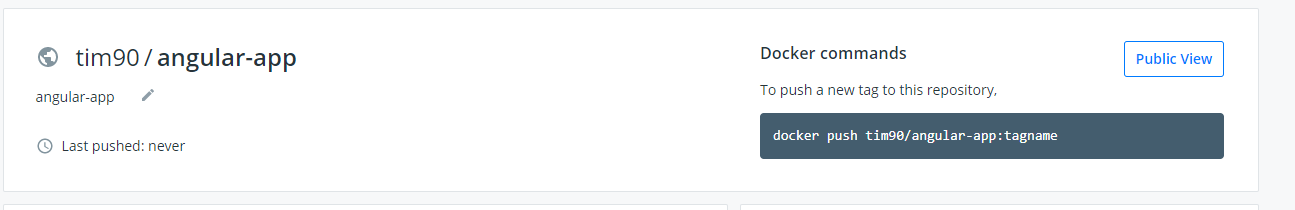
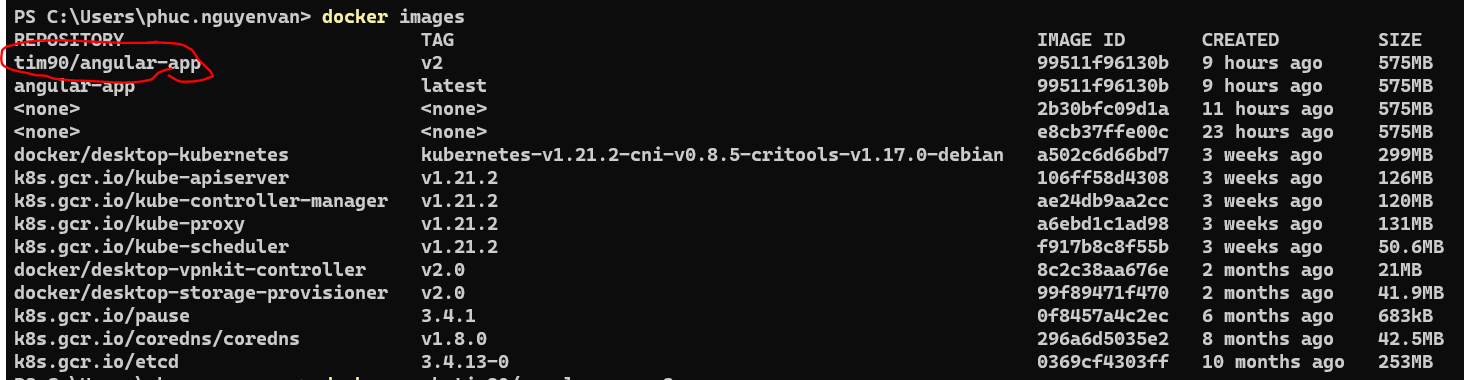


Image mà ta muốn push lên phía local

Trước đó ta đã đánh tag và tạo ra một image ta cần push lên từ image latest theo lệnh sau

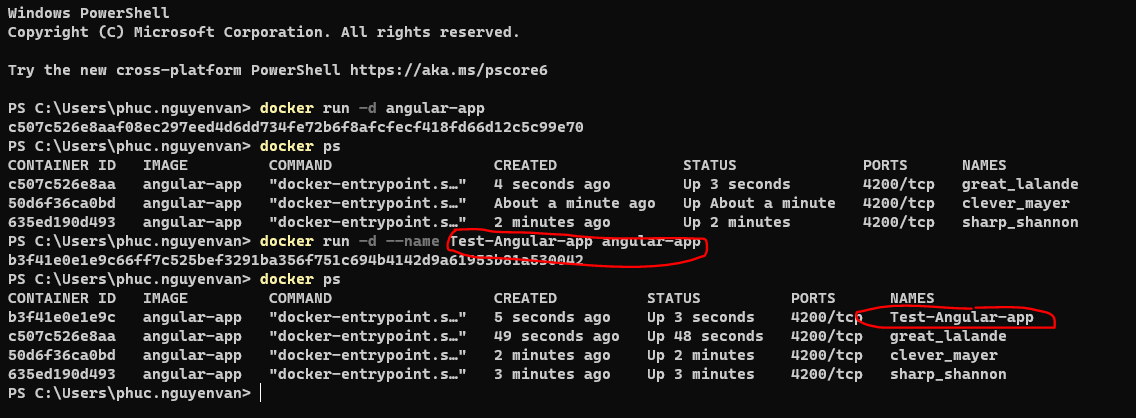
**docker image tag 995 tim90/angular-app:v2**



1. starting container

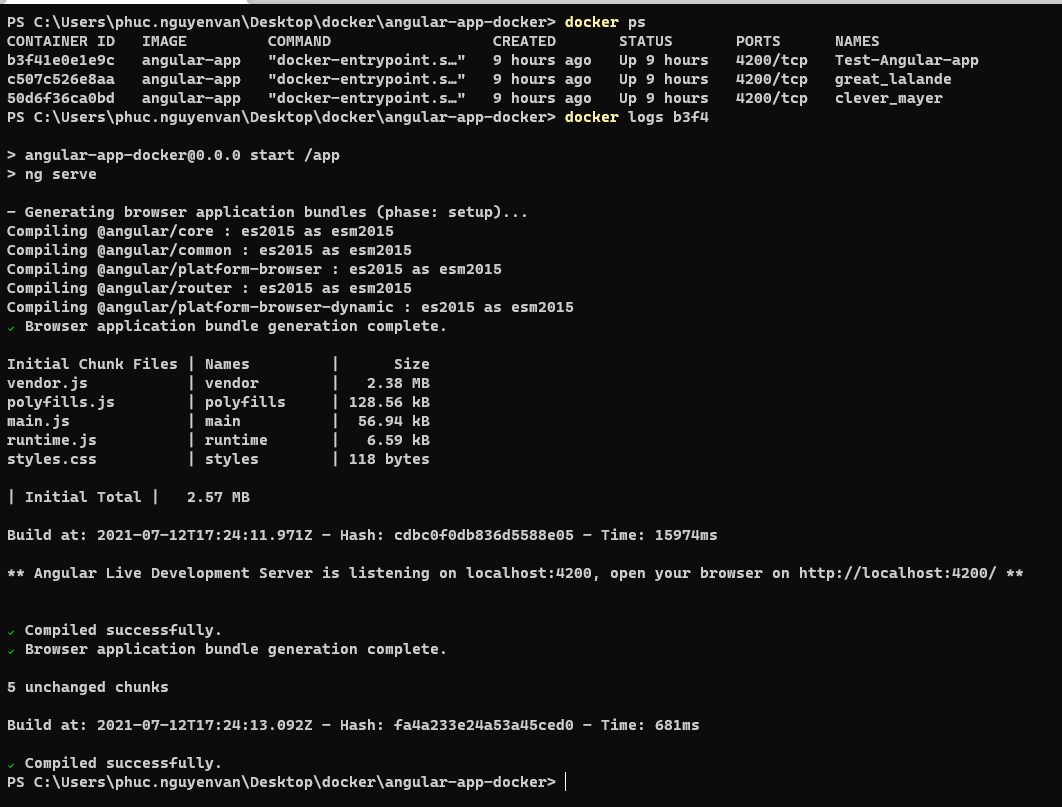
Chạy mới một container ở chế độ dectect mode (không bị treo màn hình terminal khi chạy) với NAMES là Test-Angular-app

**docker run -d --name Test-Angular-app angular-app**



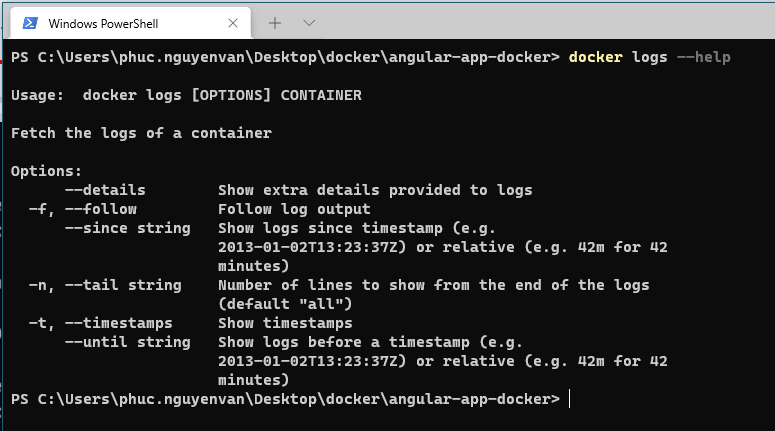
1. view log

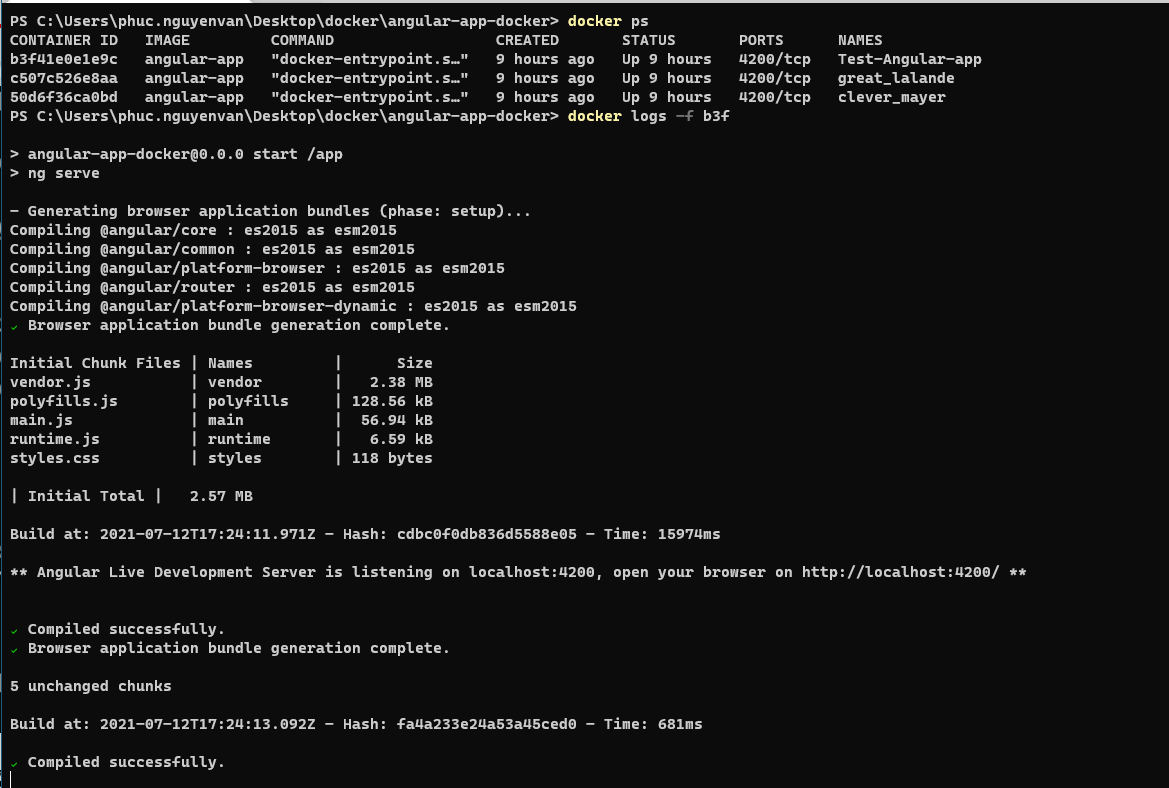
Để xem log ta dùng lệnh angular-app-docker **docker logs ContainterID**



Để xem log tự động, có nghĩa là khi thay đổi thì log nó sẽ hiển thị ra màn hình

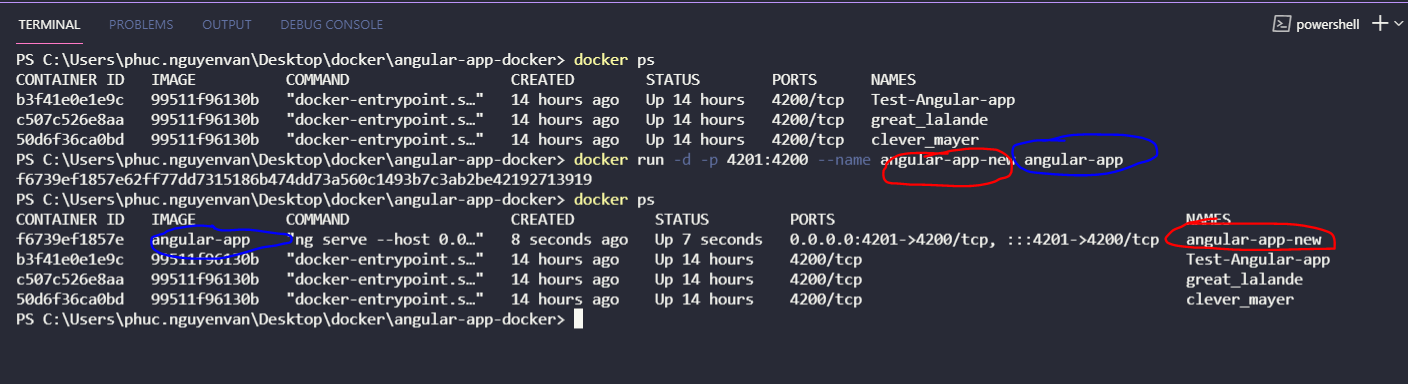
bên dưới là tất cả tham số của việc xem log





1. Publishing port

**docker run -d -p 4201:4200 angular-app-new** (tên image mới) **angular-app-new** (tên image cũ)

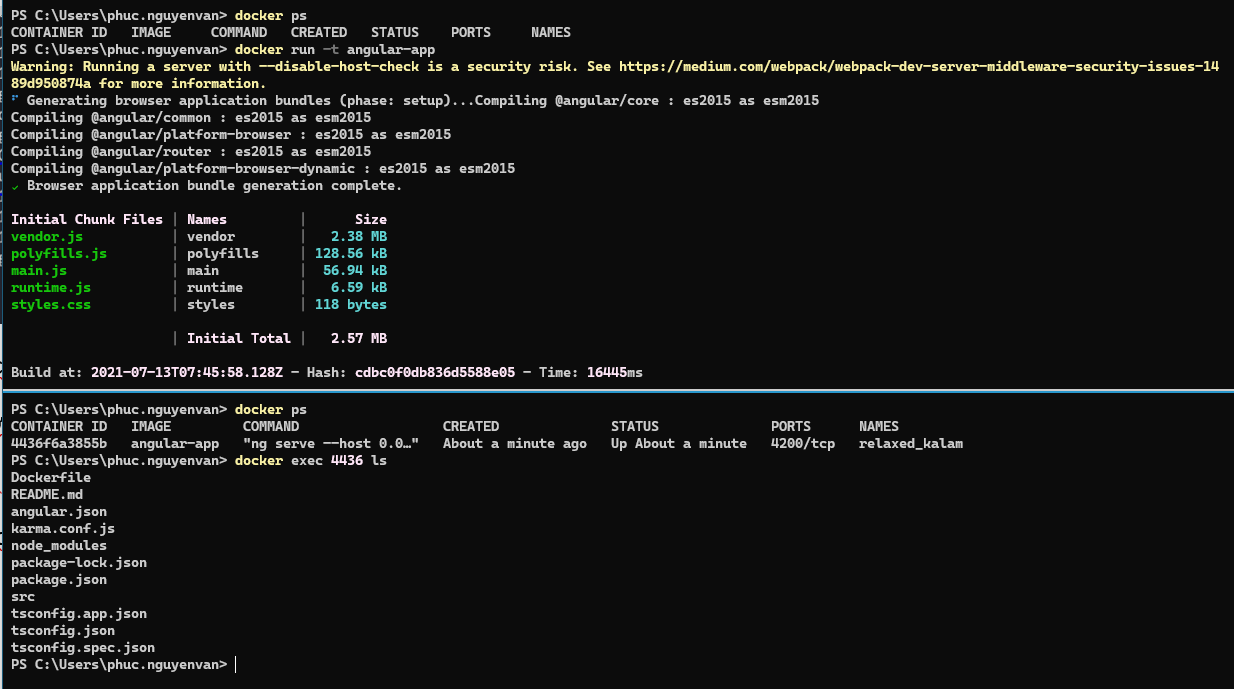
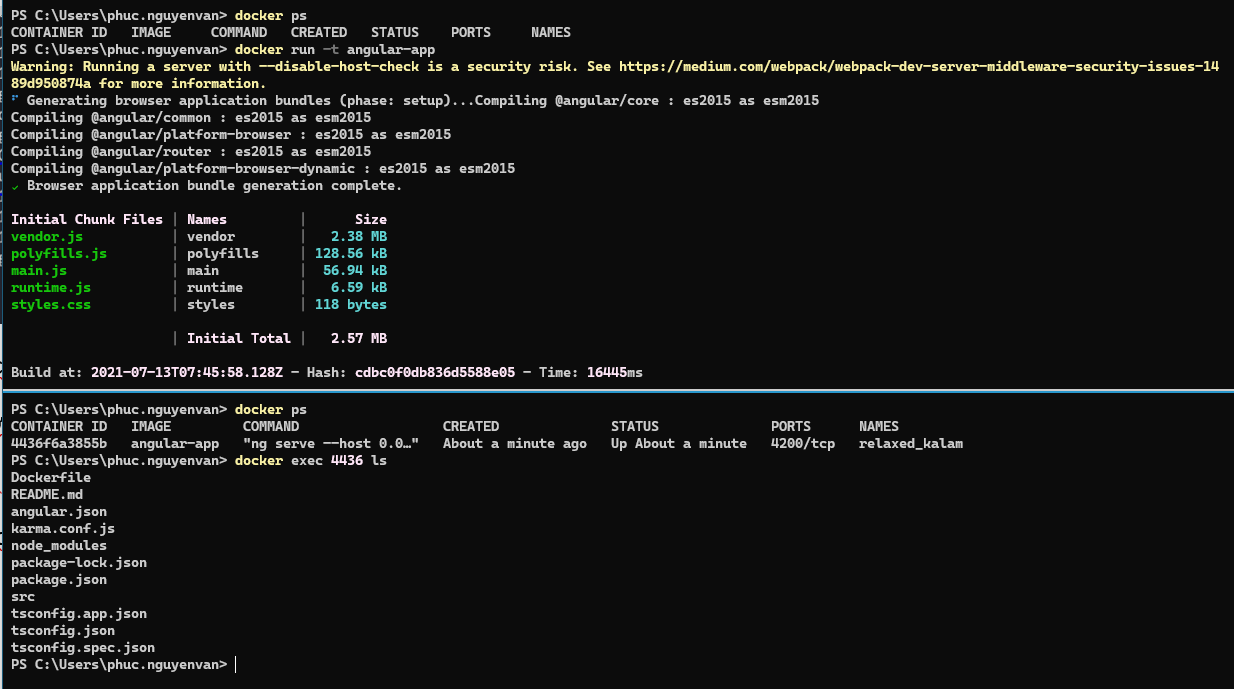


1. Thao tác với command trong running container

có 2 cách để tương tác bằng lệnh vào một container là

docker run: khởi chạy ứng dụng, khi container chưa start nó sẽ start luôn

docker exec: sẽ chạy command với container đang chạy, nghĩa là thực hiện command trên một container đang chạy.



1. Start và stop một container

Stop:

docker stop container\_ID/ container\_name

Start:

docker start container\_ID/ container\_name

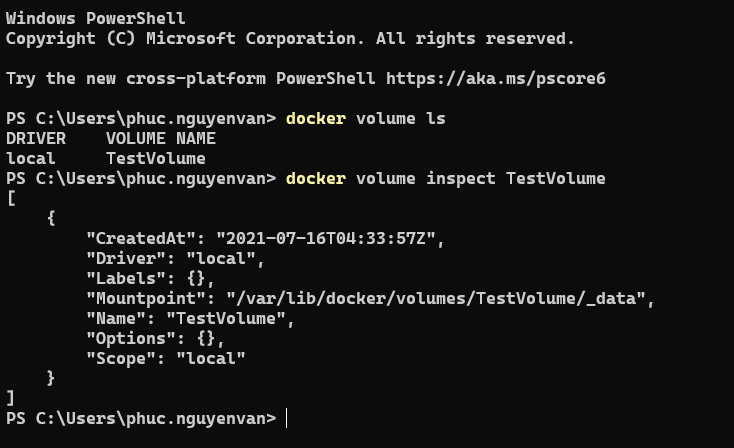
1. Tạo docker volume

Lệnh tạo mới volume

docker volume create TestVolume

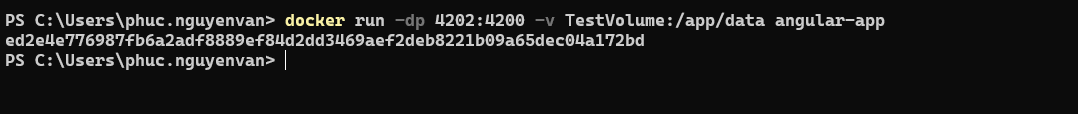
lệnh để xem chi tiết về volume

docker volume inspect TenVolume

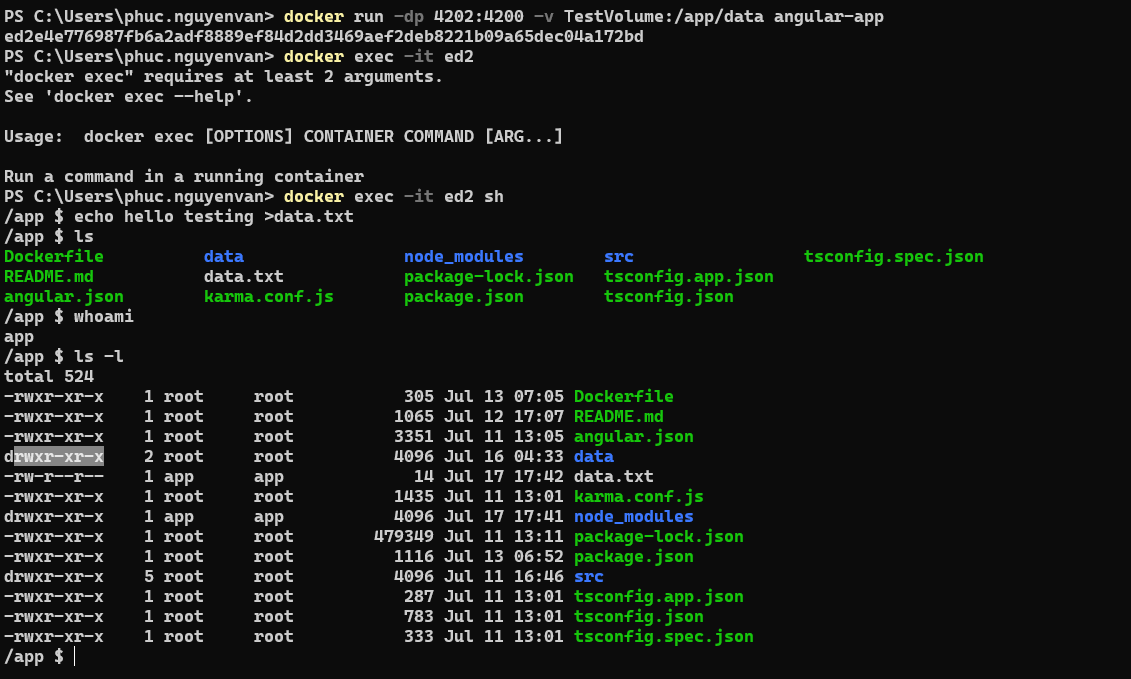


chạy mới một container và map vào một volume

ví dụ bên dưới là sẽ map ứng dụng của container angular-app vào volume TestVolume



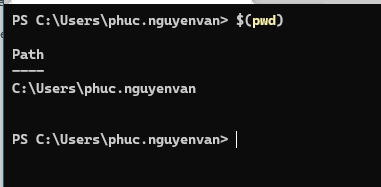
Như hình bên dưới thì thư mục ta mới tạo là data đã được cấp quyền execute



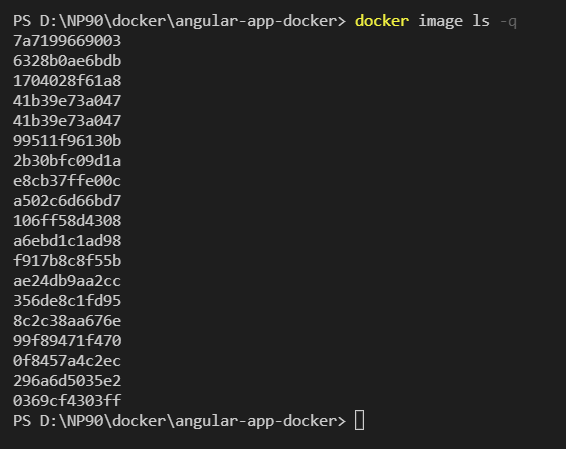
Nếu mà ta gặp vấn đề khi tạo folder data trong thư mục app thì ta sẽ khai báo lại trong file docker sau khi tạo thư mục workdir là app thì ta tạo thêm thư mục data, lưu ý là cả hai lệnh tạo thư mục này phải dưới lệnh user app để nó cấp quyền đầy đủ bằng user này.

Khi mà ta map một volume khi chạy một container thì khi ta thoát khỏi container đó hay remove container đó thì dữ liệu trong volume đó vẫn còn.

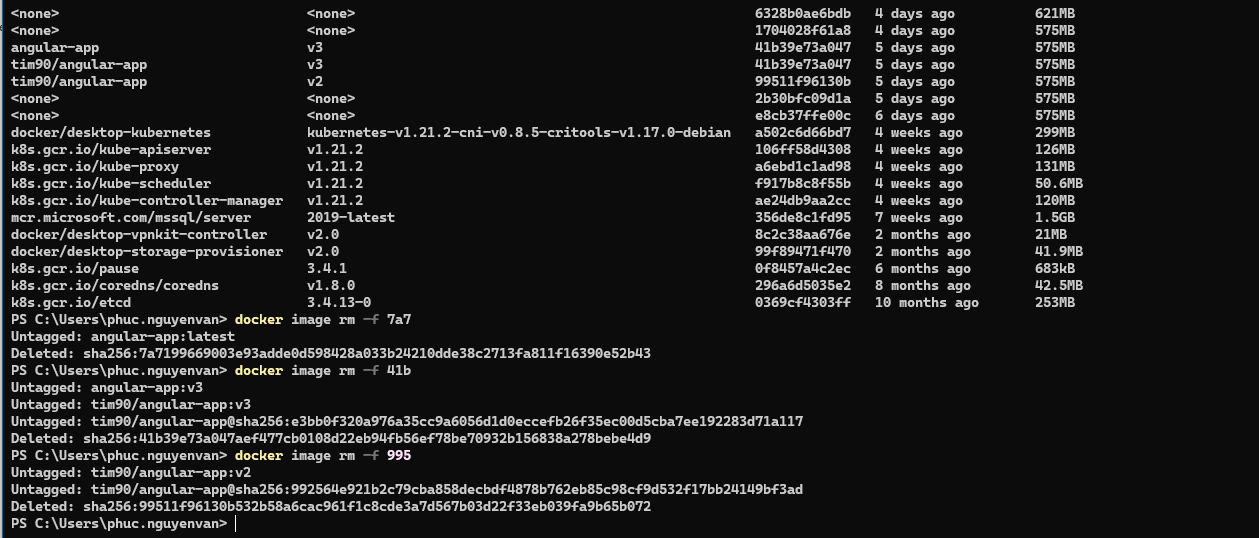
Lấy thư mục hiện tại $(pwp)



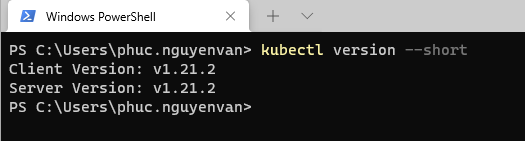
Liệt kê tất cả các image id trong docker



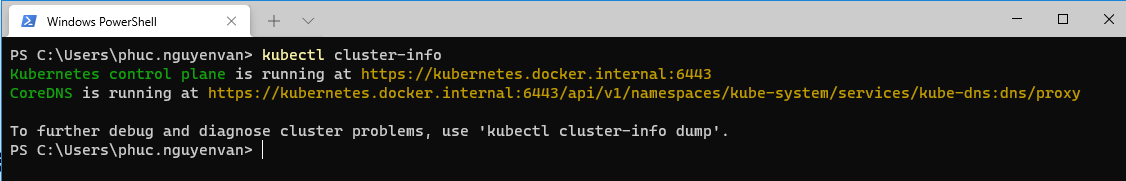
remove một image với tham số -f để force xóa luôn các image đang chạy



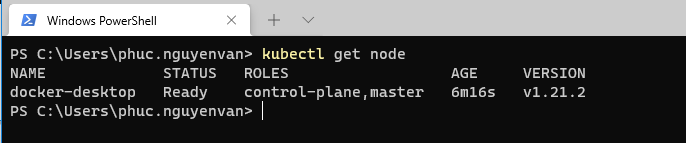
**Kubernetes Note**



Xem thông tin về Kubernetes cluster.

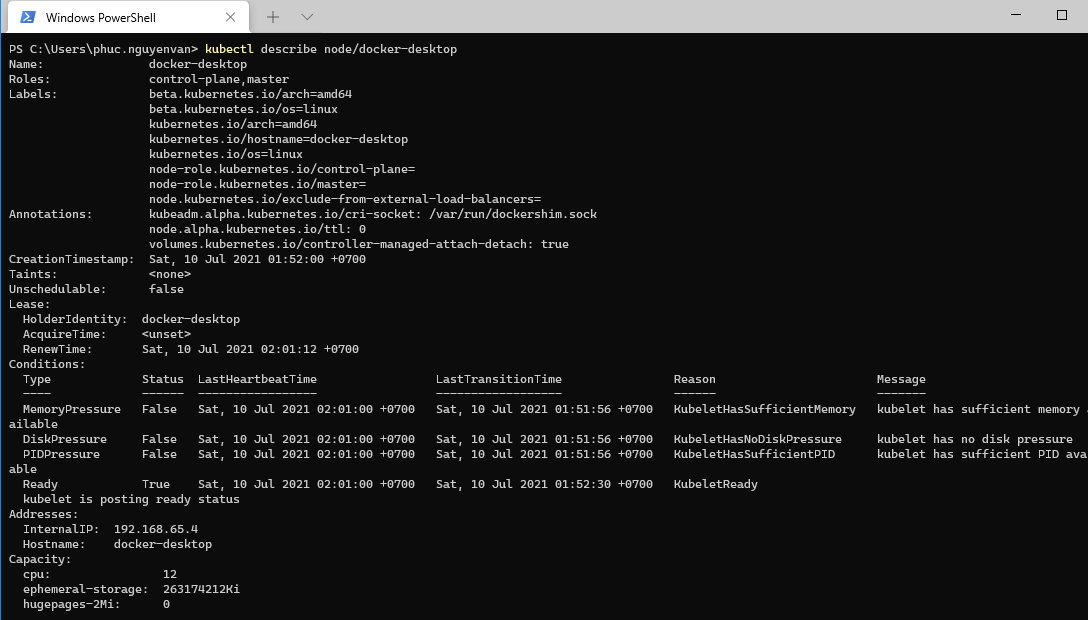


Kiểm tra trong kubenate này có những cái máy cái slot nào



Để xem thông tin chi tiết của slot nào đó ta dung lệnh sau

**kubectl describe node/docker-desktop**

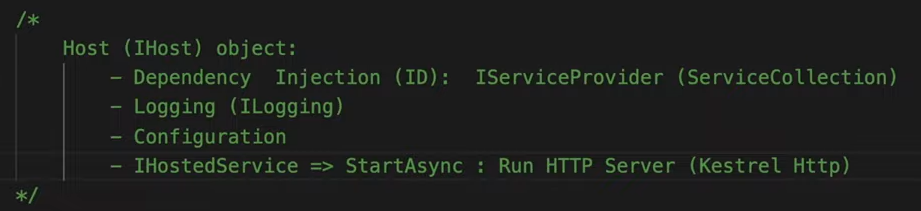


ASP.NET CORE

Để chạy một ứng dụng asp.net ta cần

Host ( được triên khia bởi interface IHost)

* trong thằng này nó có DI :
  + IServiceProvider (ServiceCollection)
* Logging (ILogging)
* Configuration
* IHostedService -> StartAsync: khi thằng này khởi chạy nó sẽ run HTTP Server (Kestrel Http)



Thứ tự ta cần làm sau đây



Ở bước 3 khi mà host.run() thì nó sẽ gọi đến phương thức StartAsync() để run cái http server(kestrel)

Cách tạo một custome middleware

Cách 1 ta tạo một class bình thường sau đó ta làm các việc sau

* Trong hàm khởi tạo ta cần truyền vào một biến là next có kiểu là RequestDelegate, biến next này là dùng để biết mà có gọi đến middleware tiếp theo không.
* Ta override lại một thương phước gọi là InvokeAssync với tham số là một biến tên là context có kiểu Httpcontext, biến này sẽ nhận và xử lý request, sau khi xử lý hết tất cả nếu ta muốn đi đến middleware tiếp theo ta gọi biến next bên trên như sau \_next(context) để gọi đến middleware tiếp theo
* Muốn đăng ký middle của ta thì trong Configure ta gọi đến middle ware của ta như sau: app.UseMiddleware<Your\_Middleware>

Cách để đăng ký middle của ta như là một service trong ứng dụng

* Ta tạo một class tĩnh
* ta viết một phương thức mở rộng với tham số là this IApplicationBuilder
* Khi ta sử dụng ta chỉ cần gọi trong method configure như sau: app.YourMiddleware.