bỏ toolbox + integration

Basic

* **Benefits & Importances :** 
  + *Simplified Development & Deployment* : Docker simplifies the entire development process , leading to simpler development and deployment. Developers are free to concentrate (*focus*) on developing code instead of tinkering (*mày mò*) with environment configurations when they package apps together with their dependencies into portable containers. ->Faster development cycles and simpler deployments across several settings
  + *Increased Scalability and Efficiency* : Containers are lightweight and resource-efficient -> They allow for quicker application startup times and more effective resource use.
    - Scaling applications becomes effortless
  + *Improved Collaboration and Consistency* (*nhất quán*): Docker facilitates uniformity (*tạo đkiện*) among development, testing, and production environments.
    - By guaranteeing (*ensure*) everyone is using the same version of the program and its dependencies, developers may share container images, which reduces issues and expedites debugging (*đẩy nhanh qtrinh gỡ lỡi*)
  + *Cloud-Native Ready* : Docker set us up for success as cloud adoption keeps rising. Docker integrates smoothly with major cloud platforms like : AWS, Azure and GCP which makes it perfect for developing and implementing cloud-native apps.
  + *Market Demand and Career Growth* : Possessing Docker knowledge is a highly desirable ability. As the need for containerized apps grows, businesses are actively looking for engineers with experience with Docker. -> Enhance our employment prospects and lead to fascinating professional alternatives.
  + *Future-Proofing Your Skills* : Software development will move toward containerization -> Not only will we gain a useful skill by learning Docker, but we will also be investing in a set of talents that will be applicable for years to come.
* **Features and Characteristics of Docker** :
  + *Containerization* : Docker’s primary function is to create isolated, portable units that package configurations, dependencies, and code for applications.
  + *Image Management* : Docker makes it possible to create, save and distribute(*phân* *phối*) container images, guaranteeing standardized(*chuẩn* *hóa*) application delivery throughout various settings.
  + *Networking* : Docker offers fine-grained control over container networking facilitating inter-container communication and service discovery. (*detailly control to make facilitate to communication between containers and service discovery*)
  + *Volumes* : By using volumes -> developers can store data outside of the container in a persistent manner (*kiên trì*) that endures (*tồn tại*) even after the container restarts.
  + *Security* : Docker uses security mechanisms to segregate containers and manage access
  + *Scalability* : Docker makes it simple to scale apps ( Adding more containers instances to manage
  + *Orchestration* (*phối hợp)*: Docker make it possible to handle intricate deployments involving numerous containers
  + *Development Workflow* : Offer standardized environments for development, testing, production -> Docker simplifies development workflows
  + *Microservices Architecture* : Encourages modularity and scalability.
  + *Continuous Integration/Continuous Delivery* (CI/CD) : Docker easily integrates with CI/CD pipelines to automate builds ,tests and deployments.
  + *Cloud-Native Development* : Docker makes it possible to deploy and maintain applications in cloud environments effectively.
  + *Legacy Application Modernization* : Docker can be used to upgrade legacy applications by containerizing them - increase their manageability and portability (*di động*)
  + *DevOps Practices* : Simplify infrastructure management, automation and communication for containerized apps.
* **Prerequisites to learn Docker** :
  + Basic understanding of ***operating systems***
  + Comfort with the command line interface
  + Knowledge of ***networking concepts*** (IP addresses, port, protocols).
  + Experience with programming languages (python, java, go).
  + Familiarity with Version Control Systems (Git)

Overview

# **Reasons why use Docker :**

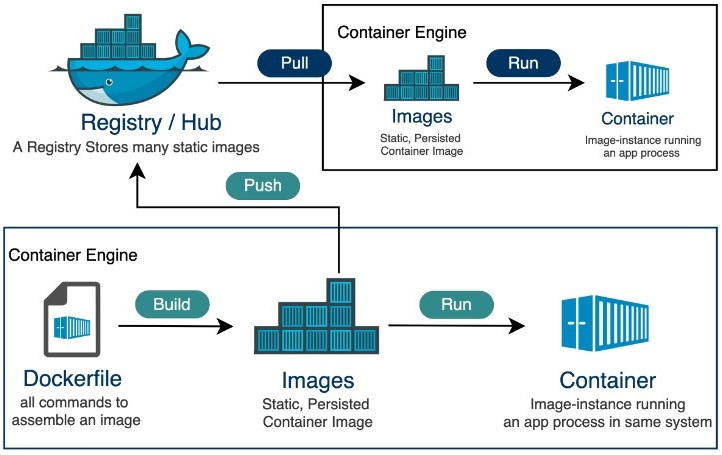
* Rapid Application Development and Delivery :
  + Docker speeds up application development cycles by providing standardized environments in the form of local containers -> These containers are integral (*0 thể thiếu*) to CI/CD workflows and they ensure fast and consistent application delivery.
* Responsive Deployment and Scaling :
  + Docker is a container-based platform -> Facilitate highly portable workloads.
  + Allow users to run applications seamlessly across various environments,
  + Its portability and lightweight nature allow for dynamic workload management.
* Maximizing Hardware Utilization :
  + Docker is a cost-effective alternative (*giải pháp tiết kiệm chi phí*) to traditional virtual machines -> Enable higher server capacity utilization.
  + Allow users to create high-density environments and perform smaller deployments.
  + Allow businesses to achieve more with limited resources.

|  | **Docker Containers** | **Virtual Machines** |
| --- | --- | --- |
| Architecture | Lightweight and portable, share the host OS kernel.  Run on top of the host OS and encapsulate the application and its dependencies | Full-fledged hardware ( including the guest OS) on top of a hypervisor.  Each VM runs its own OS instance - independent of the host OS |
| Resource Efficiency | Highly efficient since they share the host OS kernel and require fewer resources. | Consume more resources since they need to imitate an entire OS, including memory, disk space , CPU. |
| Isolation | Provide process-level isolation  ->Achieved through namespace and control group | Offer stronger isolation -> More secure but heavier. |
| Portability | Highly portable. | Less flexible. They can be portable to some extent through disk images |
| Startup Time | Spin up almost instantly (*gần như lập tức*) -> Best suitable for microservices architectures and rapid scaling. | Take longer to start |
| Use Cases | Microservices architectures, CI/CD pipelines, and application that require rapid (*nhanh chóng*) deployment and scaling | Running legacy (*old*) applications that have strict security requirements where strong isolation (*cô lập*) is necessary. |

* Once we have already installed docker desktop but we don't have docker quickstart terminal -> It already on OS’s terminal

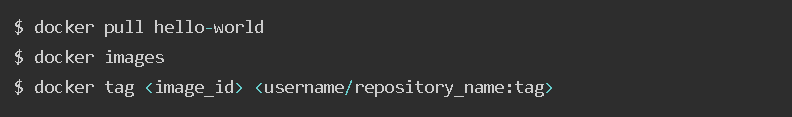
Docker-Hub

# **Docker - Hub :**

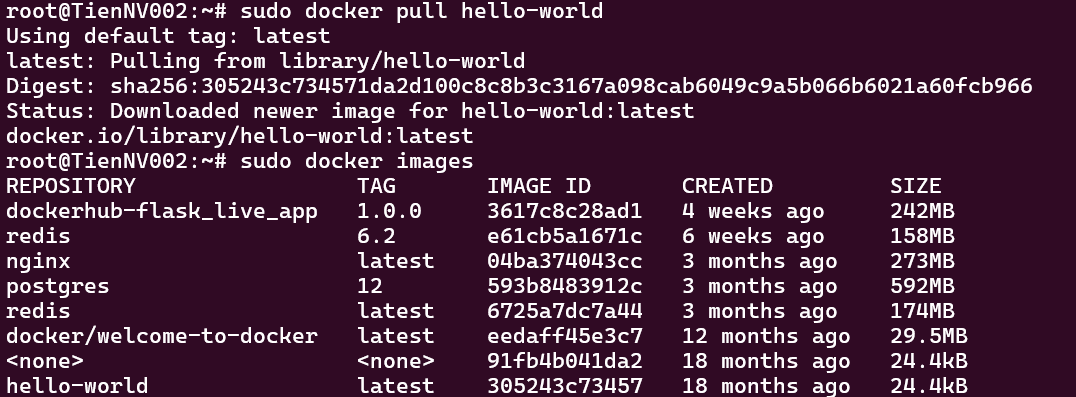
* **Docker Hub** : Is a cloud-based repository service that allows users to store, share and manage Docker container images.
  + Developers can package their apps and dependencies into lightweight, portable containers.
  + Those containers can operate consistently in various environments -> Applications can then be deployed and scaled more easily.
* **Docker Hub** is a central location where Docker users can find, share and work together on containerized applications.
* Providing : Databases, web servers, programming languages and a plethora of other software and services.
* **Features and Benefits :** 
  + **Centralized Repository** : Docker Hub allows users to search, access and share containerized apps and services.
  + **Vast Library of Images** : Provides access to a huge library of pre-built Docker images. We can find and select images based on our unique requirements in the vast collection.
  + **Open Collaboration** : Docker Hub promotes an environment of open collaboration -> Allow developers to share their own Docker images with the community.
  + **Automation tools** : Offer tools for automating the build, test and deployment of Docker images. Providing support for automated builds, which start builds automatically whenever changes are pushed to a repo.
  + **Versioning and Tagging** : Simplifying the management and tracking of various iterations of a service or application over time. Making it easier to roll back to earlier versions if necessary and guarantees consistency and reproducibility across various environments.
  + **Access Control and Permissions** : Allow businesses to regulate who can view, edit, and share Docker images.
  + **Scalability and Performance** : Provides high-performance infrastructure and scalability for hosting and distributing Docker images. -> Guarantee dependable and quick access to container images irrespective of the repo’s size or level of popularity.
  + **Integration with Docker Ecosystem** : Offer a unified (*thống nhất*) platform for developing, launching and overseeing containerized applications from development to production.

# **Push / Pull images from Docker Hub :**

1. Pushing Images to Docker Hub : (**Upload**)
   1. *Tag the Image* : -> Copy an image and give it a new tag.

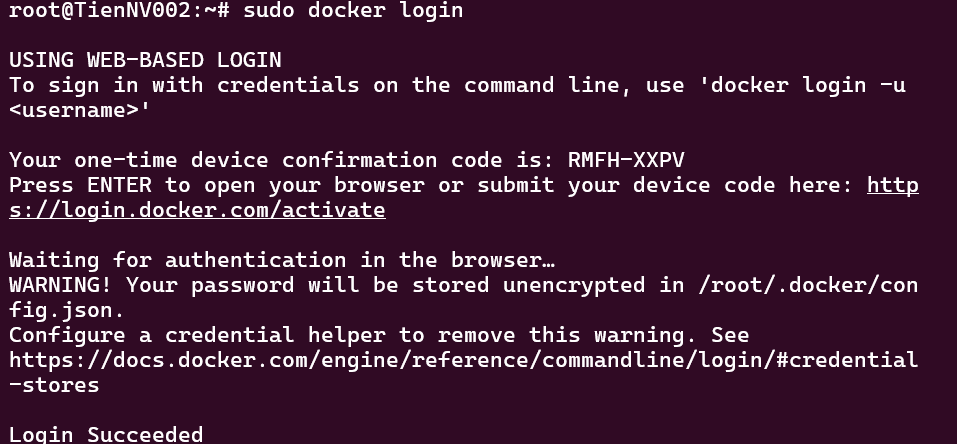


* Example :

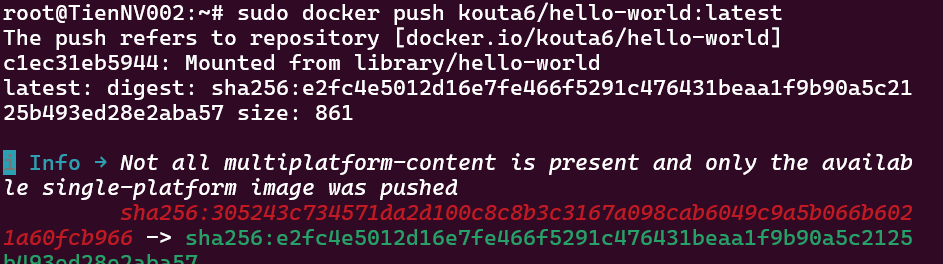




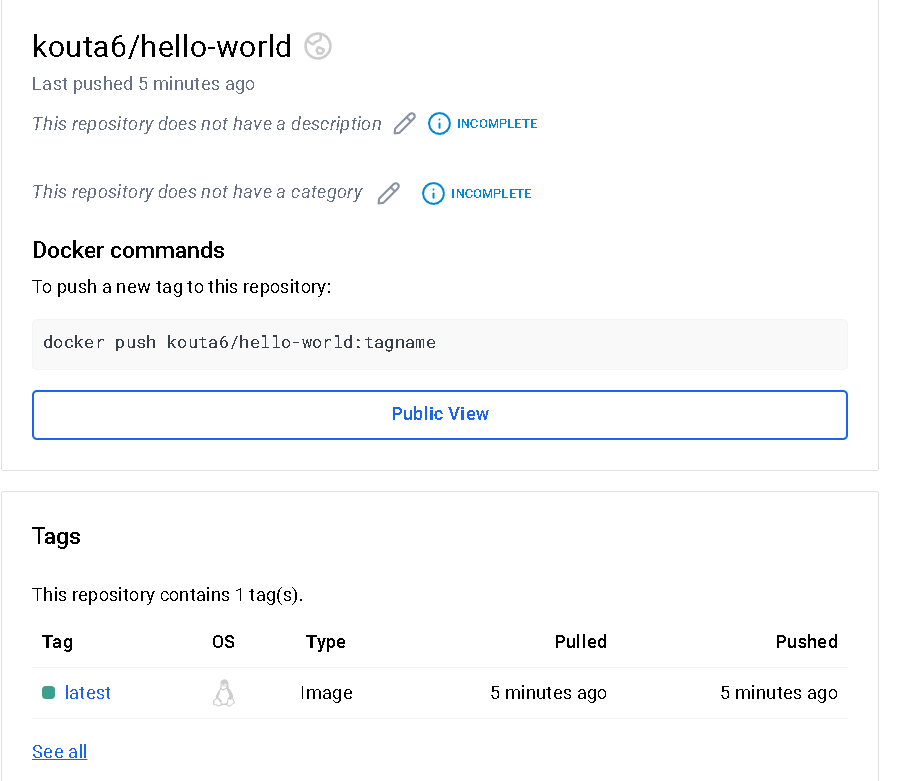
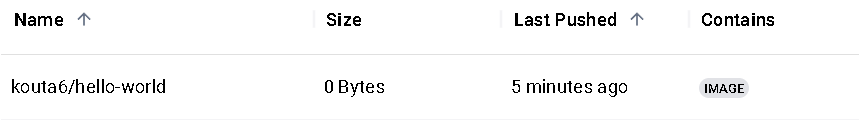
* The form :   
  Docker tag <IMAGE\_ID> <USERNAME/REPOSITORY:TAG>
  1. *Log in to Docker Hub* :
* Before Pull or Push images from the private repo -> Log in to Docker Hub using the command line



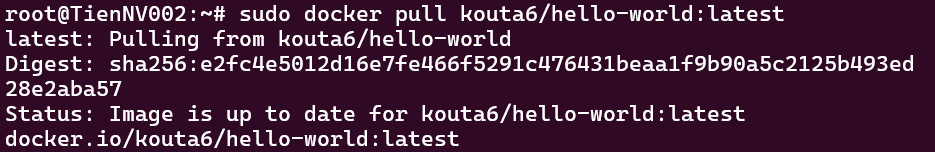
* Follow the request and log in.
  1. *Push the Image* :



* We have to sign in on the docker repo before push.
* -> Verify the Pushed imaged in the docker hub repo :



1. Pulling Images from Docker Hub : (**Download**)



***\*\*\*\* : The step follow : Pull - Tag - Push***

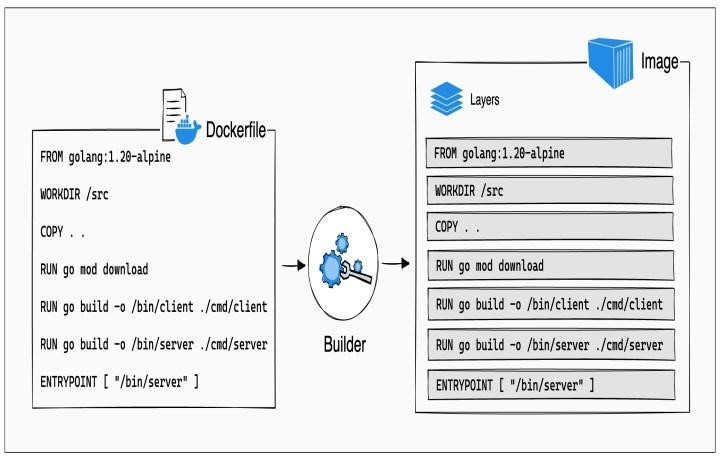
1. Key Components of Docker Hub :

* **Repositories** :
* **Webhooks** : allow automated responses to events that take place inside of repo. -> Can be set by users to initiate specific events.
* **Service Accounts** : Authenticate and grant access to Docker Hub services and APIs, service accounts are needed -> Make it possible to automated procedures to securely communicate with Docker Hub without the need for human intervention.
* **Vulnerability Scanning** : Find security flaws in Docker images, Docker Hub provides vulnerability scanning capabilities. -> Help users mitigate possible security risks by scanning images for known vulnerabilities in their dependencies and offering useful insights.
* **Mirroring** : Organizations can mirror Docker Hub repo to theirown private infrastructure or internal registries.
* **Automated Builds**: Start builds automatically whenever changes are pushed to a linked repo -> Automated builds simplify the process of making Docker images -> Guarantee images are continuously updated and created with the most recent modifications.
* **User Interface (UI) and API** : For interacting with repo, managing images, carrying out administrative tasks -> Docker Hub provides an intuitive web interface. -> Docker offers a strong API for programmatic access to its features, facilitating easy integration with outside tools and automation processes.

Docker - Images

# **Docker Images :**

* + Self-contained templates that are used to build containers.
  + Make use of a tiered file system to store data effectively.
  + Each layer - which contains instructions such as downloading software packages or transferring configuration files - represents a particular phase in the image generation process.
  + A text file known as a Dockerfile forms the basis of a Docker image - which contain instructions for creating image layer by layer.
    - In most cases, an instruction begins with “**FROM**” to identify the base image.
    - Commands like “**RUN**” are used to carry out particular operations within a layer.



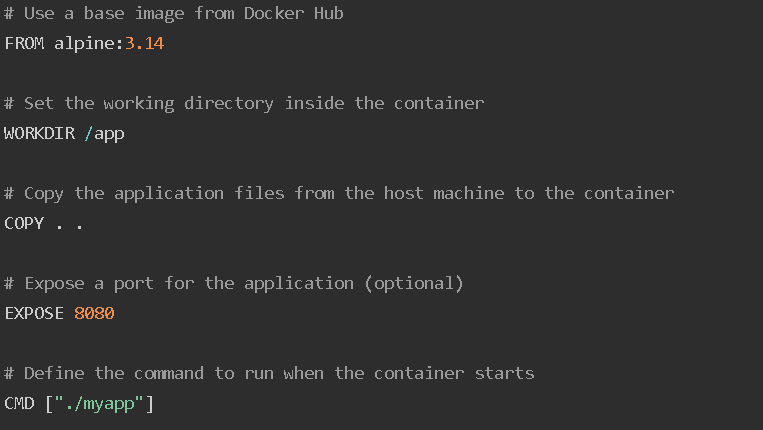
* + Docker images are read-only templates -> Any changes we make to the running program happen inside a container, not to the image itself => A clear division is maintained between the runtime state (container) and the application definition (image).

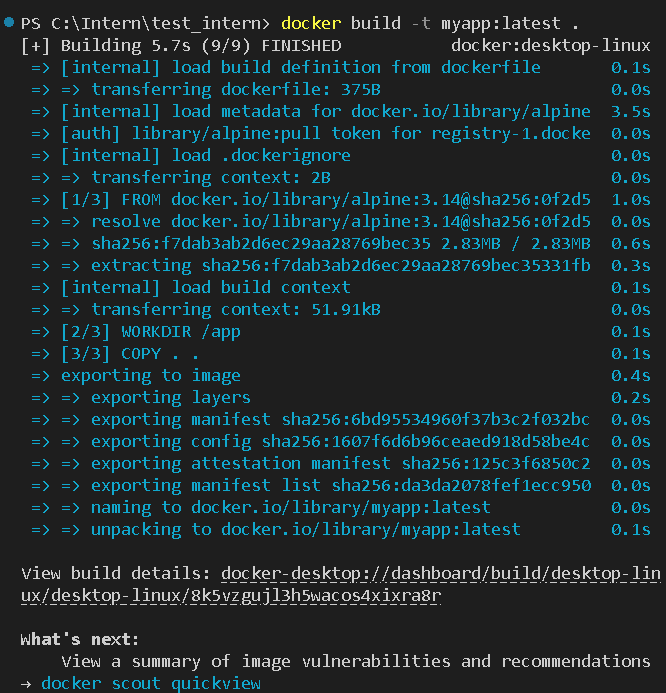
# **Key components and concepts of Docker Images :**

* + **Layers** : unchangeable -> Immutable
    - Docker images consist of several layers.
    - Every layer denotes (biểu thị) a collection of filesystem modifications.
    - Each Dockerfile instruction adds a layer on top of the previous one while building a Docker image.
    - Since its immutability, Docker can effectively reuse layers during image builds and deploys -> Speed up build times and use less disk space.
  + **Base image** :
    - Base image has the bare minimum runtime environment and operating system needed to complete the application.
    - Base images from CentOS, Ubuntu, Debian, and Alpine Linux are frequently used.
  + **Dockerfile** :
    - Dockerfile is a text document with a set of instructions for creating Docker images -> Which describe how to create the basic image, add files and directories, install dependencies, adjust settings, and define the container’s entry point.
  + **Image Registry** :
    - Docker images can be stored in either public or private registries.
    - Registries offer a centralized area for managing, sharing and distributing Docker images.
    - Providing image scanning for security flaws, versioning and access control.
  + **Tagging** :
    - A repository name and a tag combine to form a unique identification for Docker images.
    - Used to distinguish between various image versions.
    - When no tag is given -> Docker uses the “latest” tag by default.
  + **Image Pulling and Pushing** :
    - Docker pull command can be used to download Docker images to a local system from a registry
    - Docker push command can be used to push images from a local machine to a registry.
  + **Layer Caching** : for performance optimization (tối ưu hóa hiệu suất)
    - If the Dockerfile instructions haven’t changed -> Docker leverages (tận dụng) previously built cached layers when we construct an image.

# **Useful Docker Image Commands :**

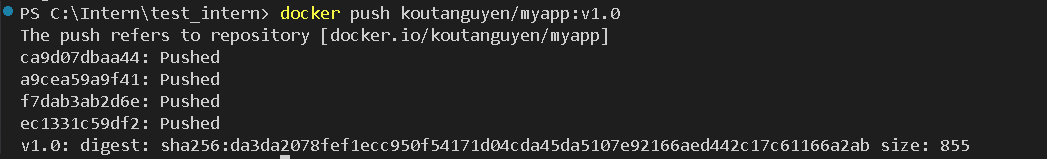
* + List all Docker images :
    - *$ docker images*
    - If we want to just display the IDs:   
      *$ docker image ls -q* (“-**quite**”)
  + Pulling Docker Images :
    - *$ docker pull ubuntu:*20.04 - if no tag is specified -> Docker will automatically pull the “latest”
  + Building Docker Images from Dockerfile :
    - Docker build command creates a Docker image from a Dockerfile placed at the provided path.
    - During the build process, Docker follows the instructions in the Dockerfile to generate layers and assemble (lắp ráp) the final image.
  + Dockerfile :
    - A dockerfile will have a form like :



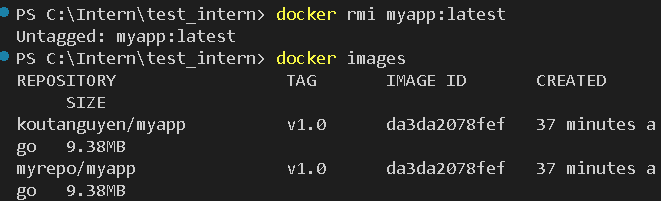
* + - Then we can build an image using :   
      *$ docker build -t myapp:latest .*   
      
  + **Tagging Docker Images :** 
    - This allows us to label and reference multiple versions of an image.  
      *$ docker tag repo\_name:old\_tag new\_repo:new\_tag  
      $ docker tag myapp:latest myrepo/myapp:v1.0*



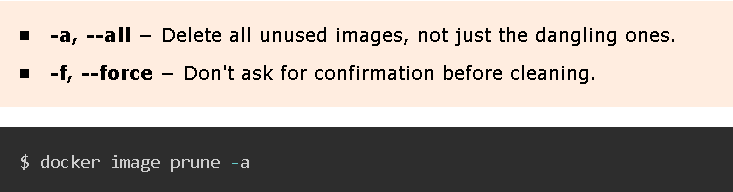
* + Pushing Docker Images :
    - Command :   
       *$ docker push new\_repo:new\_tag*

**

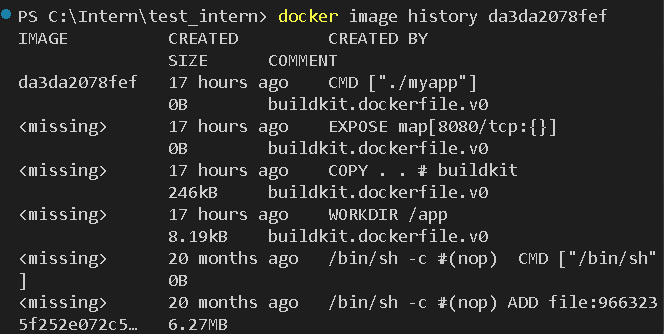
* + - If we want to push, ensure that the repo that we created has the same repo name with the name which we created when doing tagging < Summarize : just different about the tag >.
  + Removing Docker Images :
    - Docker “rmi” command removes 1 or more Docker images from the local machine ->Can provide either image name or the image ID.  
       *$ docker rmi myapp:latest*

**

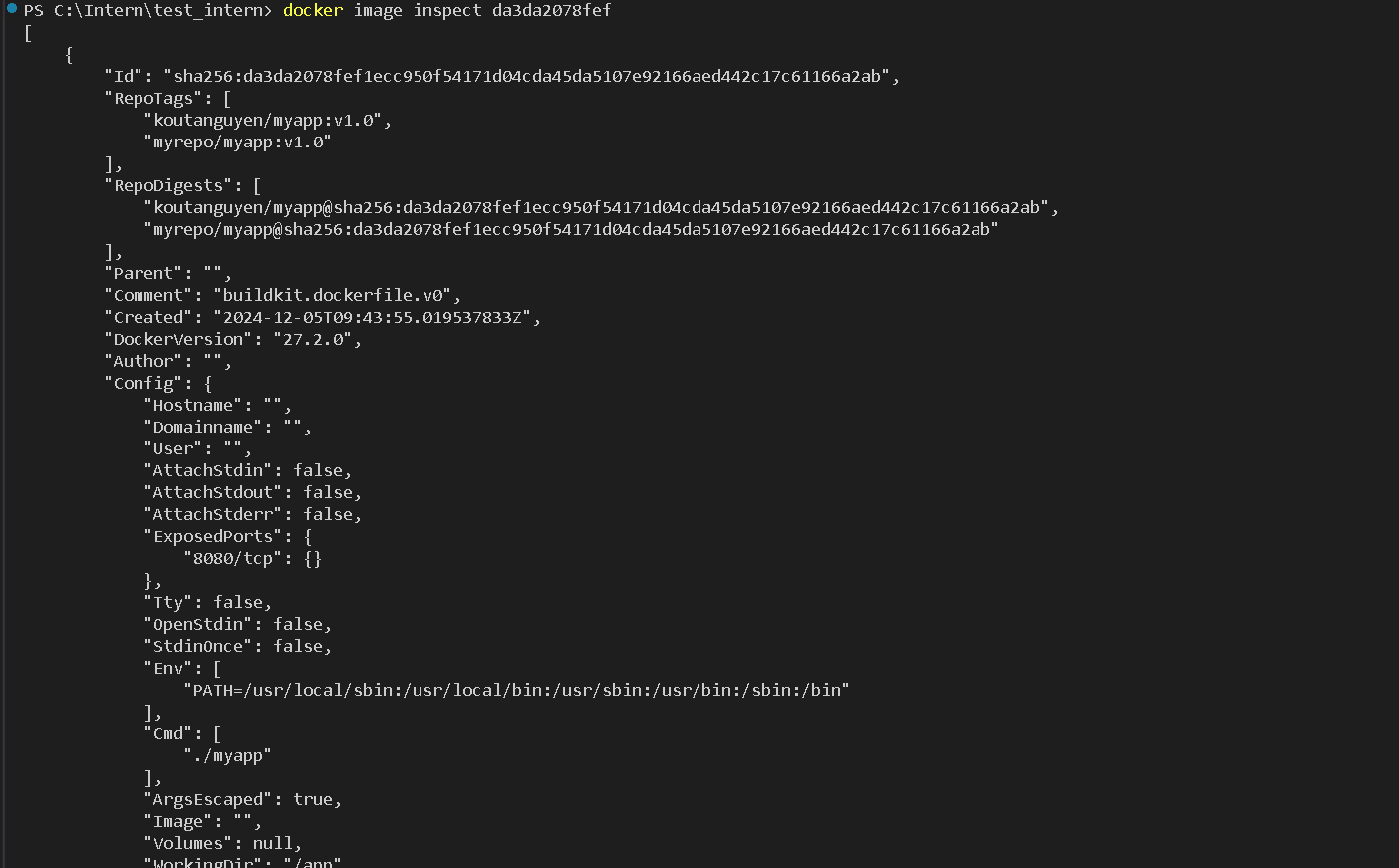
* + Pruning (cắt tỉa) Docker Images :
    - Docker image prune command removes unused Docker images from the local machine.
    - Useful for freeing up disk space by removing images that are no longer associated with any containers or tags.
  + Options :



* + Viewing Docker Images History :
    - Show the history of a Docker image , including : the commands and the instructions used during the image-building process.
    - Useful for determining how an image is formed and diagnosing problems.  
      *$ docker image history myimage:latest  
      $ docker image history da3da2078fef <the ID>*

**

* + Inspecting Docker Images :
    - Return extensive information about a Docker image in JSON format.
    - Including : the settings, layers, environment variables.
    - Can use the --format option to format the output .  
      *$ docker image inspect myimage:latest  
      $ docker image inspect da3da2078fef <the ID>*

**

* + Delete all Docker Images at Once :
    - Use a chain of 2 commands that removes all the images after listing all the image IDs :   
      *$ docker rmi $(docker image ls -q)*

Docker - Containers

# **Beginning :**

* Docker container is a runtime instance (trường hợp) of a Docker image.
* Can be created by instantiating the image.
* Docker containers bundle (bó) an application along with all of its dependencies into a compact, light package.
* Can operate (vận hành) reliably in a range of computing environments by using virtualization at the operating system level.
* Images are essentially blueprints that contain all the files, libraries and configurations required to run a particular application.

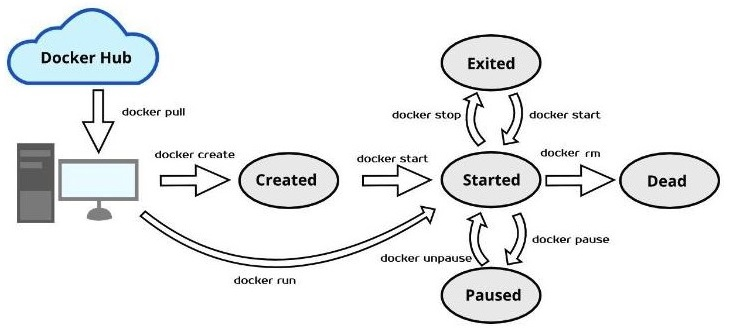
# **Key Concepts of Docker Containers :**

* Containerization :
  + Container function based on the concept of containerization, which is packing an application together with all of its dependencies into a single package.
  + This package, referred (đc xem) to as a container image, includes all of the necessary runtime environments, libraries, and other components needed to run the application.
* Isolation (cách ly) :
  + Operating system-level virtualization is used by Docker containers to offer applications isolation.
  + With its filesystem, network interface, and process space, each space, each container operates independently of the host system as a separate process.
  + By maintaining their independence from another -> Containers are kept from interfering with another’s operation thanks to isolation.
* Docker Engine :
  + The brains behind containers : builds, launches, maintains them.
  + Docker daemon - operates in the background
  + Docker client - lets users communicate with the Docker daemon via commands

=> These are 2 of parts that make up the Docker Engine

* Image and Container Lifecycle :
  + The creation of a container image is the first step in the lifecycle of a Docker container.
  + A dockerfile is used to build this image
  + The image can be used to instantiate (init) containers - which are instances of the image that are running after it has been created.
  + It is possible to start, stop, pause, and restart containers as one.
* Resource Management :
  + Because of their shared kernel (hạt nhân) architecture and lightweight design -> Docker containers provide effective resource management.
  + Since containers share the operating system kernel of the host system -> Overhead is decreased and startup times are accelerated (tăng tốc)
  + Docker offers tools for resource usage monitoring (giám sát) and control to ensure maximum performance and scalability.
* Portability :
  + Container images are self-contained units that are easily deployable and distributed throughout various environments, ranging from production to testing and development.
  + Streamlines (hợp lý hóa) the deployment process and decreases the possibility of compatibility problems by enabling “Build once, run anywhere”

# **Docker container Lifecycle :**

****

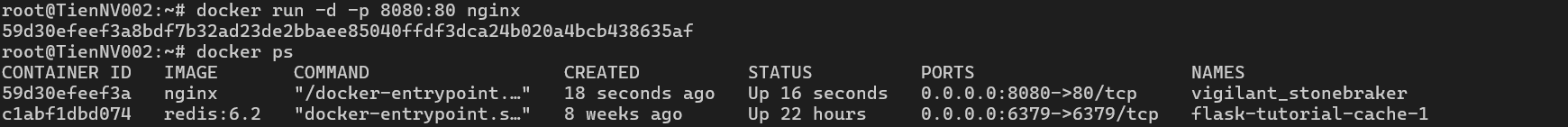
* The Created state : the first stage .
  + When a container is created with the docker create command or comparable API call -> It reaches this phase.
  + The container is not yet running when it is in the “created” state, but it does exist as a static entity with all of its configuration settings defined.
  + Docker reserves the storage columns and network interfaces that the container needs, but the processes inside the container have not yet begun.
* The Started State : The next stage .
  + When a container is started with the docker start command or an equivalent API call, it enters this stage.
  + In this stage, its processes are launched and it starts running the service or application that is specified in its image.
  + While they carry out their assigned tasks, containers actively use CPU, memory, and other system resources.
* The Paused State :
  + When a container is paused with the docker pause command, its processes are suspended (cấm),thereby stopping its execution.
  + A container that is paused keeps its resource allotments(sự phân bổ) and configuration setting but it not in use.
  + This stage helps with resource conservation and debugging by momentarily (trong chốc lát) stopping container execution without completely stopping it.
* The Exited State :
  + A container in the “exited” state has finished executing and has left its primary process.
  + Containers can enter this state when they finish the tasks they are intended to complete or when they run into errors that force them to terminate.
  + A container that has been “exited” stays stopped, keeping its resources and configuration settings but ceasing to run any processes.
  + Containers can be completely deleted with the docker “*rm*” command or restarted with the docker “*start*” command.
* The Dead state :
  + A container that is in “dead” state has either experienced an irreversible error (error k thể khắc phục) or been abruptly terminated ( chấm dứt đột ngột).
  + Including :
    - Critical errors in the containerized application
    - Problems with the host system underneath
    - Manual intervention (can thiệp chủ công).
  + When a container is “dead” -> it is not in use and the Docker daemon usually releases or reclaims its resources.
  + The containers are in “dead” -> they can not be restarted -> to free up system resources, containers need to be deleted using the “***docker rm command***”

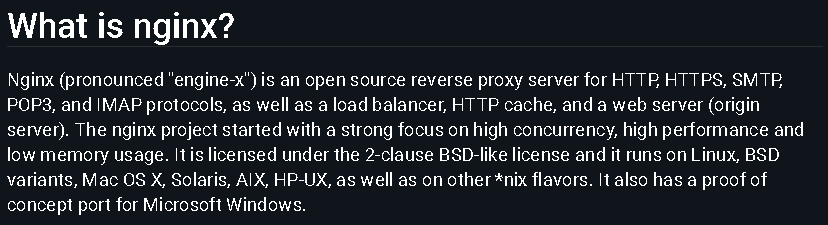
# **Important Docker Container Commands :**

* Listing all Docker Containers :   
  ***$ docker ps (-a or --all)***
  + List the Docker host’s running containers.
  + Can use the --format to show all containers, including stopped ones, as it only shows running containers by default.
  + Display the IDs, names, statuses and other pertinent details of all containers that are currently running.
  + Return an empty list if no containers are in use.
* Running a Docker Container :  
  ***$ docker run ….***
  + This command is for starting and creating Docker containers.
  + If the image isn’t already available locally -> Docker pulls it from a registry when we run this command -> Start a fresh container instance by generating 1 based on that image.
  + We can specify several options, including :
    - Volume mounts
    - Environment variables
    - Port mappings

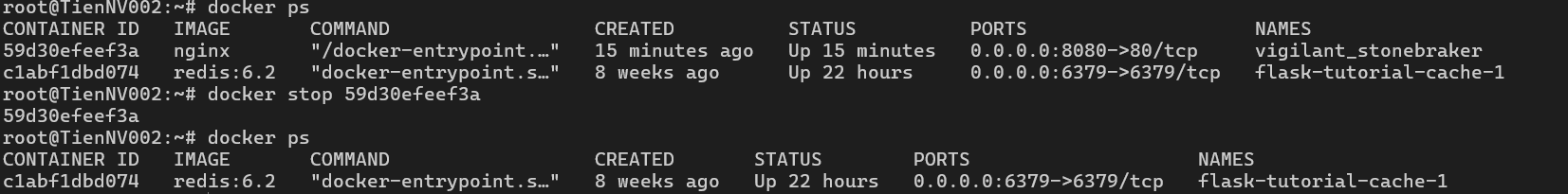
=> To tailor (điều chỉnh) the container’s configuration(Cấu hình)

* + Ex :   
    ***$ docker run -d -p 8080:80 nginx***
  + The detached(*tách rời)* mode (*-d*) of the docker run creates a new container based on the “*nginx*” image and runs it in the background.
  + This maps host port 8080 to container port 80 (-p 8080:80), granting(provide) access to the NGINX web server housed within the container.

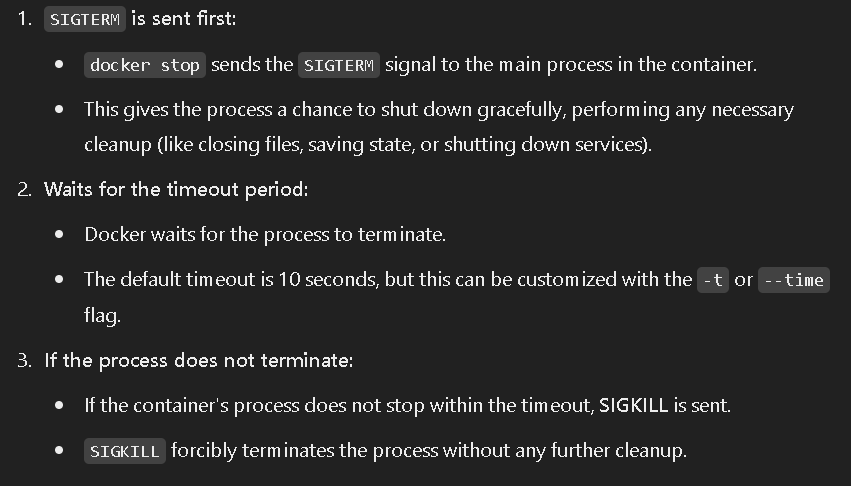




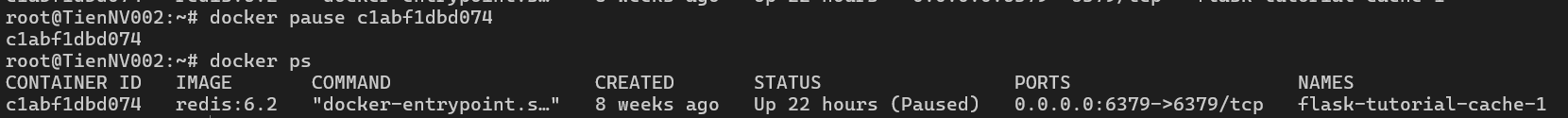
* Stopping a Docker Container :   
  ***$ docker stop -t time-second ID\_container***
  + A container can be stopped by using the ***docker stop*** command, which signals the container’s main process with a SIGTERM.
  + This enables the containers to finish any cleanup operation : saving state or cutting off network connections before shutting down.



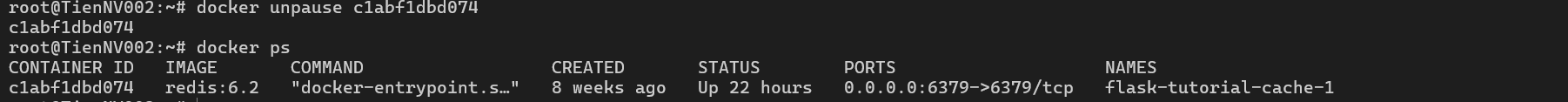
* + This command stops the specific “container”that is currently operating.
  + Docker will wait for the container to gracefully end its life for a configurable duration( 10s by default).
  + Docker will automatically terminate the container with a SIGKILL signal if it does not stop within this time limit.



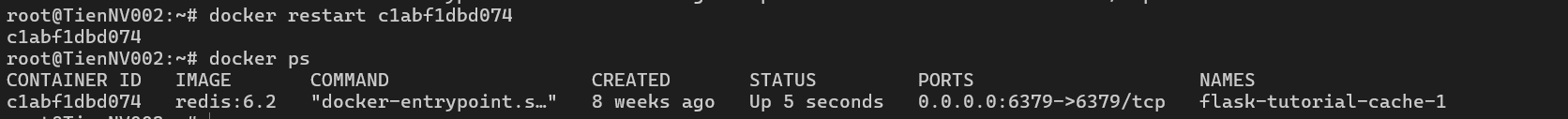
* Pausing a Running Container :   
  ***$ docker pause ID\_container***
  + A running container’s processes can be momentarily suspended, or its execution paused, within the “***docker pause command***”.
  + Helpful for temporarily freeing up system resources, debugging and troubleshooting problems.



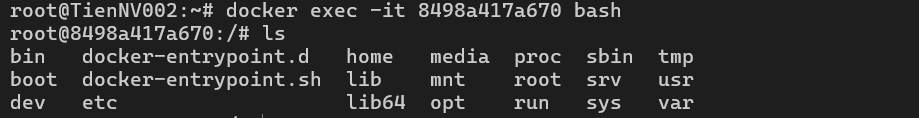
* + This command stops the specific “container” from running.
  + The container uses no CPU or memory when it is paused because its processes are frozen.
  + The container still keeps its resource allocation and configuration setting.
* Resuming a Docker Container :   
  ***$ docker unpause ID\_container***
  + When a container is paused-> its processes can be carried out again by using the “***docker unpause command***”
  + Using this command -> Container returns to its initial state and undoes the effects of the docker pause command.



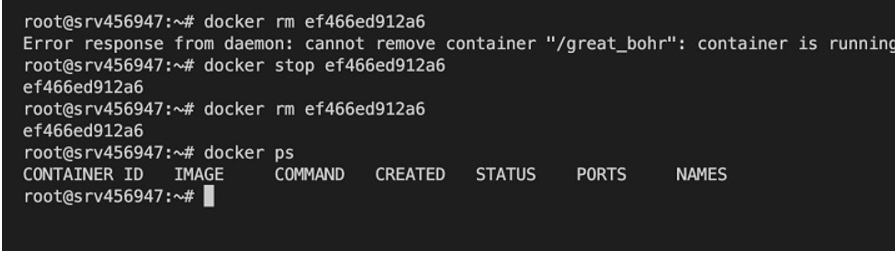
* Restarting a Container :
  + Easy to quickly stop and restart an operating container.
  + Used to force a container to reinitialize after experiencing problems (gặp sự cố) or to apply changes to the configuration of a running container.



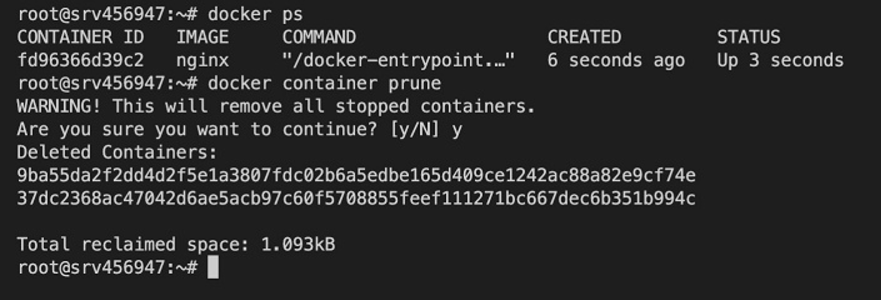
* + This command pauses and then resumes the execution of the specific container.
  + The processes inside the container are stopped and then restarted upon restarting, enabling any modification to take effect.
* Executing commands in a Running Docker Container :   
  ***$ docker exec -it ID\_container bash***
  + This command allows users to go inside an already-running container
  + Enable users to run arbitrary (optional) commands, like starting a shell session or carrying out a particular program, inside the environment of a container.



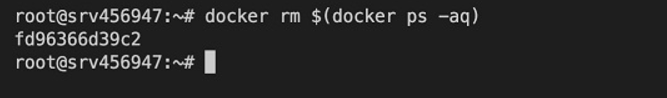
* + The “***-it***” flags allocate a pseudo-TTY and maintain STDIN open even when it is not attacked in order to enable interactive input/output.
* Removing a Docker Container :   
  ***$ docker rm ID\_container***
  + This command only removes stopped containers by default, to forcefully remove running containers -> We can use the -f or --force flag.



* + The container must stop running before being removed unless the “***-f****”* flash is used to force removal.
  + After removing, to clear up disk space, we can use “***docker container prune command***” to remove all stopped containers.



* + Docker will ask for confirmation before continuing, but we can ignore this prompt by passing it with the “***-f or --force***” flag.
  + Another hand, if we want to remove all containers together instead 1 by 1 :   
    ***$docker rm $(docker ps -aq)***



* When removing a container, we cannot take it back => Before removing :
  + Saving the container state :   
    using “***docker commit <ID\_container> <new\_image\_name>***”

Docker - Private Registries

1. **Beginning** :

* Docker registries are central image repositories strictly meant to store and distribute Docker images
* Make it easy for individual developers or teams to share their work and deploy their containerized applications in different environments.

1. **Types of Docker Registries** :

| **Public Registries** | **Private Registries** |
| --- | --- |
| They are made publicly available and usually contain many pre-build images.  The most popular public registry is ***DockerHub***, which contains thousands of official and community-contributed images. | Those that are hosted on the infrastructure or cloud, offering limited or restricted access.  These are useful when storing proprietary(unique) images sensitive(nhạy cảm) data, or images with unique compliance requirements.  Some popular private registry options include : Docker Registry (open source), Harbor (open source), Nexus Repo, JFrog Artifactory. |

1. **Popular Docker Registry Option** :

* Docker Hub (Official Registry)
  + Is the official cloud-based registry service for all Docker images.
  + Is a central location where developers can store and have access to share container images.
  + Offer public and private repositories -> It is an all-around solution for both open-source projects and private enterprise applications
  + Including some features :
    - **Large Ecosystem** (hệ thống sinh thái): Docker Hub operates millions of container images shared by a vast developer community, offering official images from Docker and verified publishers.
    - **Automated Builds** : Docker Hub can automatically build images from source code hosted on GitHub or Bitbucket.
    - **Webhooks** : allow for the triggering (kích hoạt) of actions after a successful push to a repo.
    - **Content Trust** : Authenticity (xác thực) and integrity (tin cậy) of images.
    - **Seamlessly integrated** (tích hợp liền mạch) **with the Docker CLI** : can pull and push images from the command line interface.
* Self-Hosted Registries :
  + Providing significantly more control over the storage and distribution of Docker images, especially for organizations with strict security, compliance, or performance requirements.
* Docker Registry (open source):
  + Is a basic open-source project that many other registries, such as Docker Hub, are based on.
  + A simple way to host the very own private registry.
  + **Simplicity** : Deploy and configure easily
  + **Customization** : Highly customizable through its configuration files.
  + **Scalability** : Can scale horizontally (theo chiều ngang) using a distributed storage backend.
  + **Security** : It provides means (Phương tiện) for TLS-encrypted communication and basic authentication.
  + **Extensible** : Additional functionalities(chức năng bổ sung) can be added through middleware.