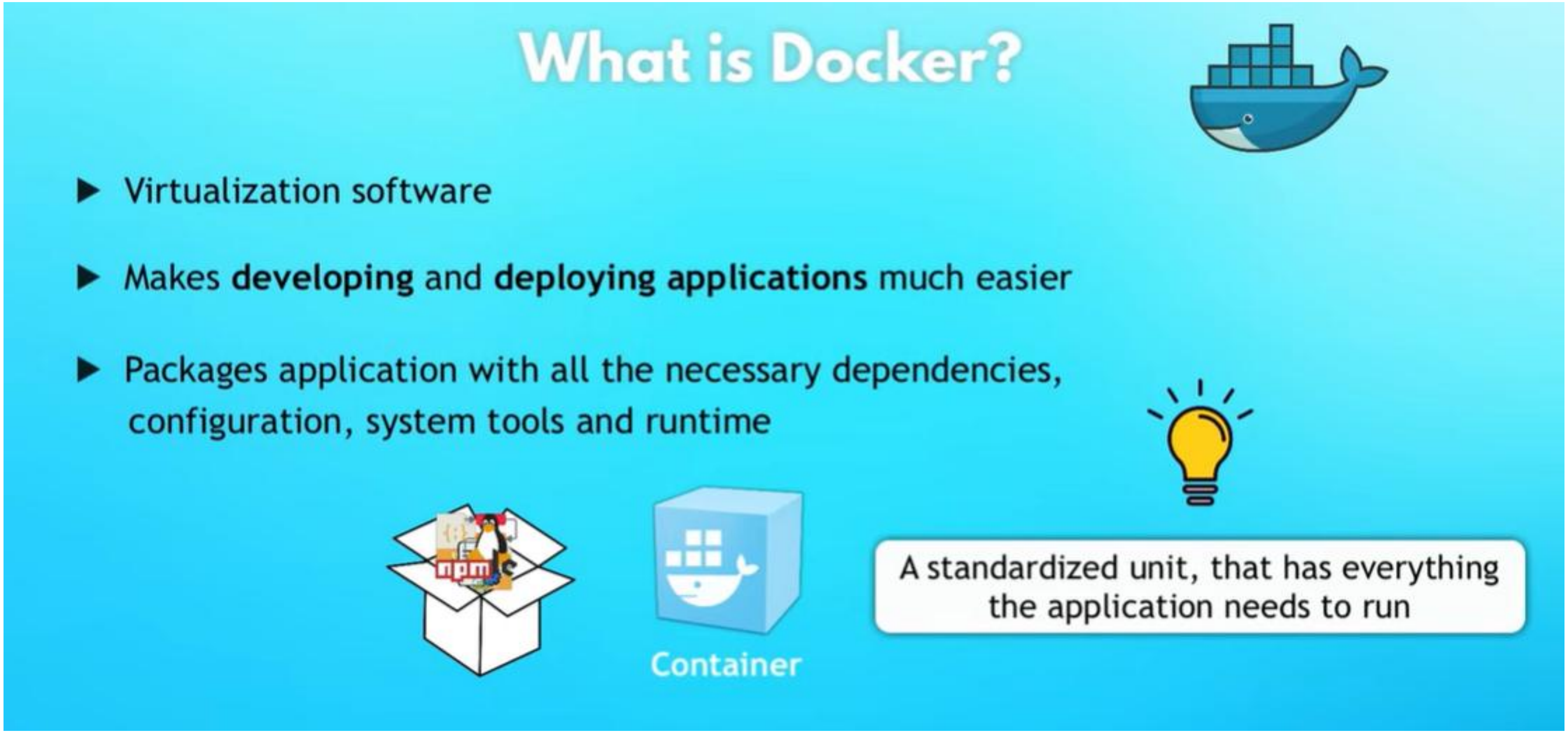
Dockers

Dr. Muhammad Safyan



What is Docker?

- Virtualization software
- Makes **developing** and **deploying applications** much easier
- Packages application with all the necessary dependencies, configuration, system tools and runtime

npm
Container

A standardized unit, that has everything the application needs to run

Why Need of docker?

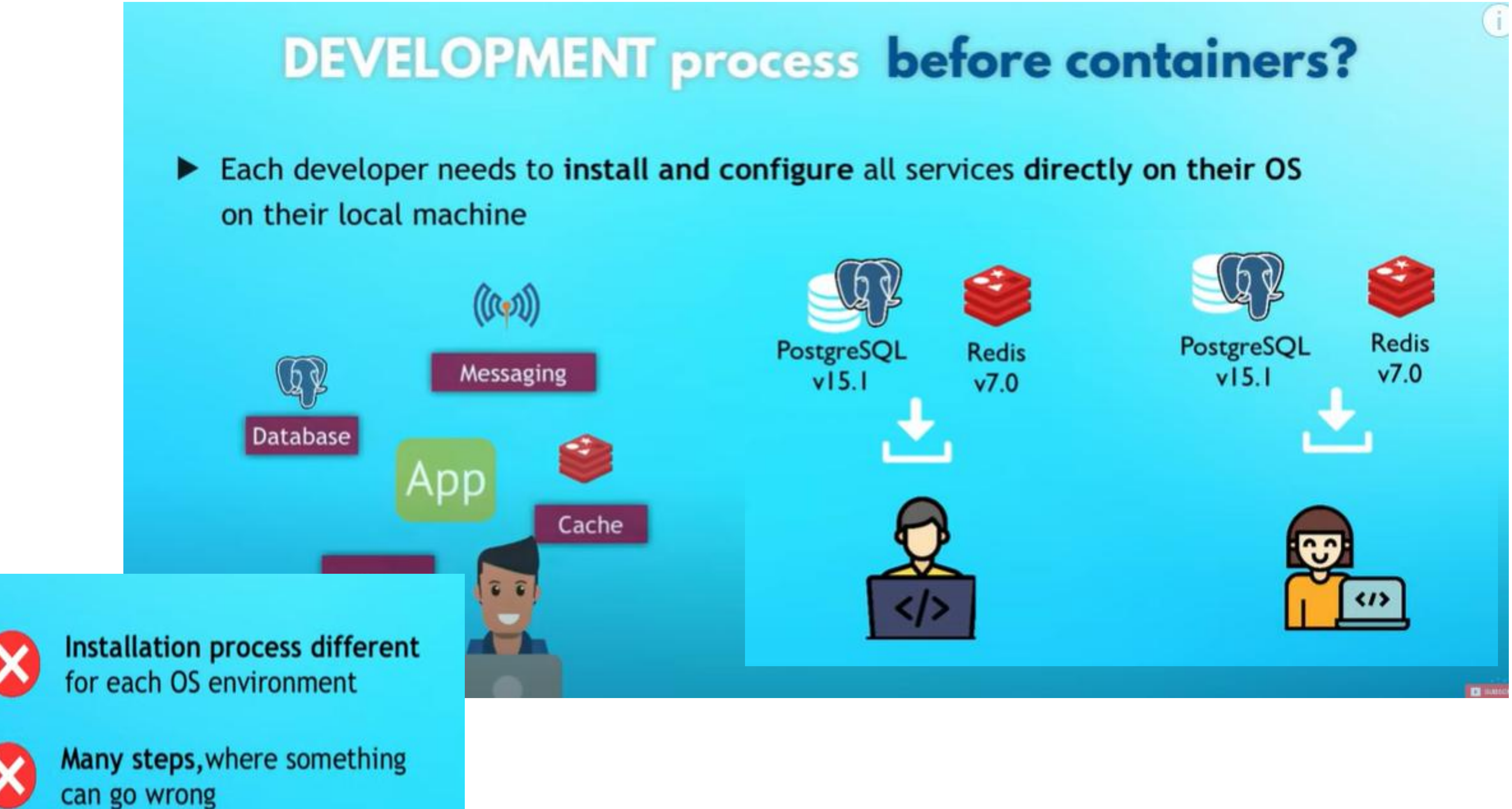
Consistency Across Environments

Problem: Applications often behave differently in development, testing, and production environments due to variations in configurations, dependencies, and infrastructure.

Solution: Docker containers encapsulate all the necessary components, ensuring the application runs consistently across all environments.

Why Need of docker?

- Isolation
- Problem: Running multiple applications on the same host can lead to conflicts, such as dependency clashes or resource contention.
- Solution: Docker provides isolated environments for each application, preventing interference and ensuring stable performance.
- Scalability
- Problem: Scaling applications to handle increased load can be challenging, requiring manual intervention and configuration.
- Solution: Docker makes it easy to scale applications horizontally by running multiple container instances, allowing for quick and efficient scaling.

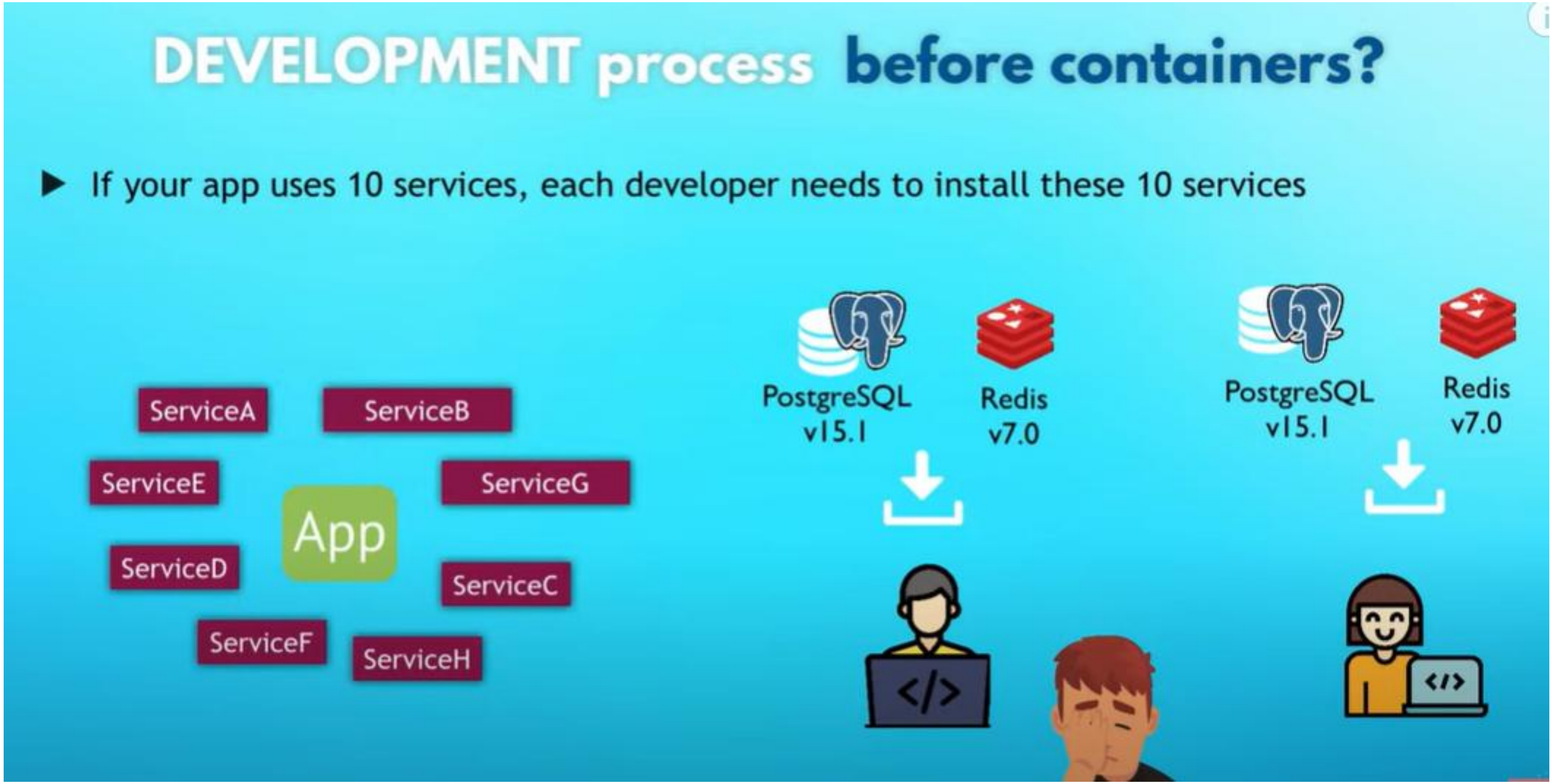


DEVELOPMENT process before containers?

- Each developer needs to **install and configure** all services **directly on their OS** on their local machine

✗ **Installation process different** for each OS environment

✗ **Many steps,** where something can go wrong



DEVELOPMENT process before containers?

- If your app uses 10 services, each developer needs to install these 10 services

ServiceA

ServiceB

ServiceE

ServiceG

App

ServiceD

ServiceC

ServiceF

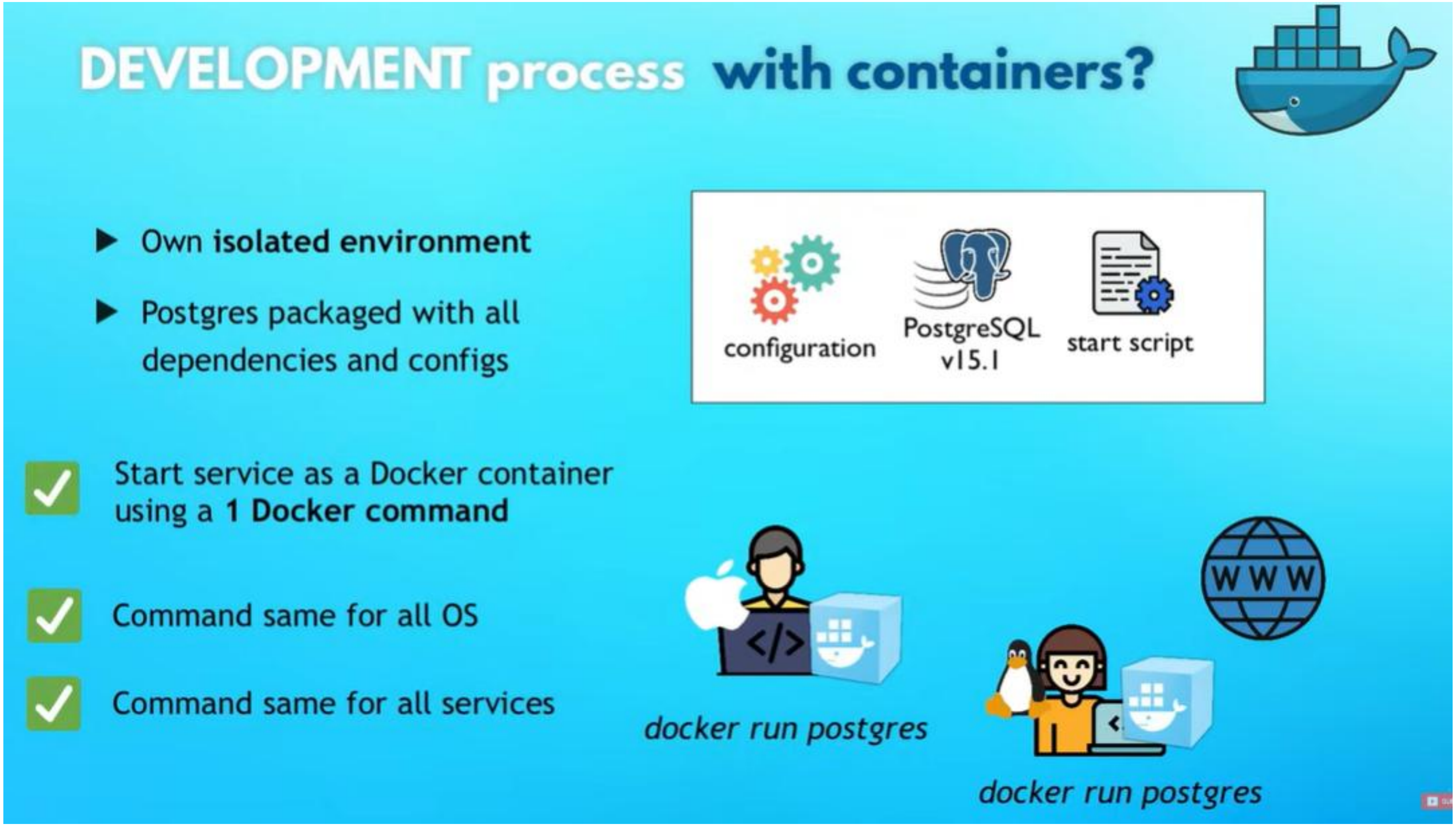
ServiceH

PostgreSQL
v15.1

Redis
v7.0

PostgreSQL
v15.1

Redis
v7.0



DEVELOPMENT process with containers?

- Own **isolated environment**
- Postgres packaged with all dependencies and configs

configuration

PostgreSQL v15.1

start script

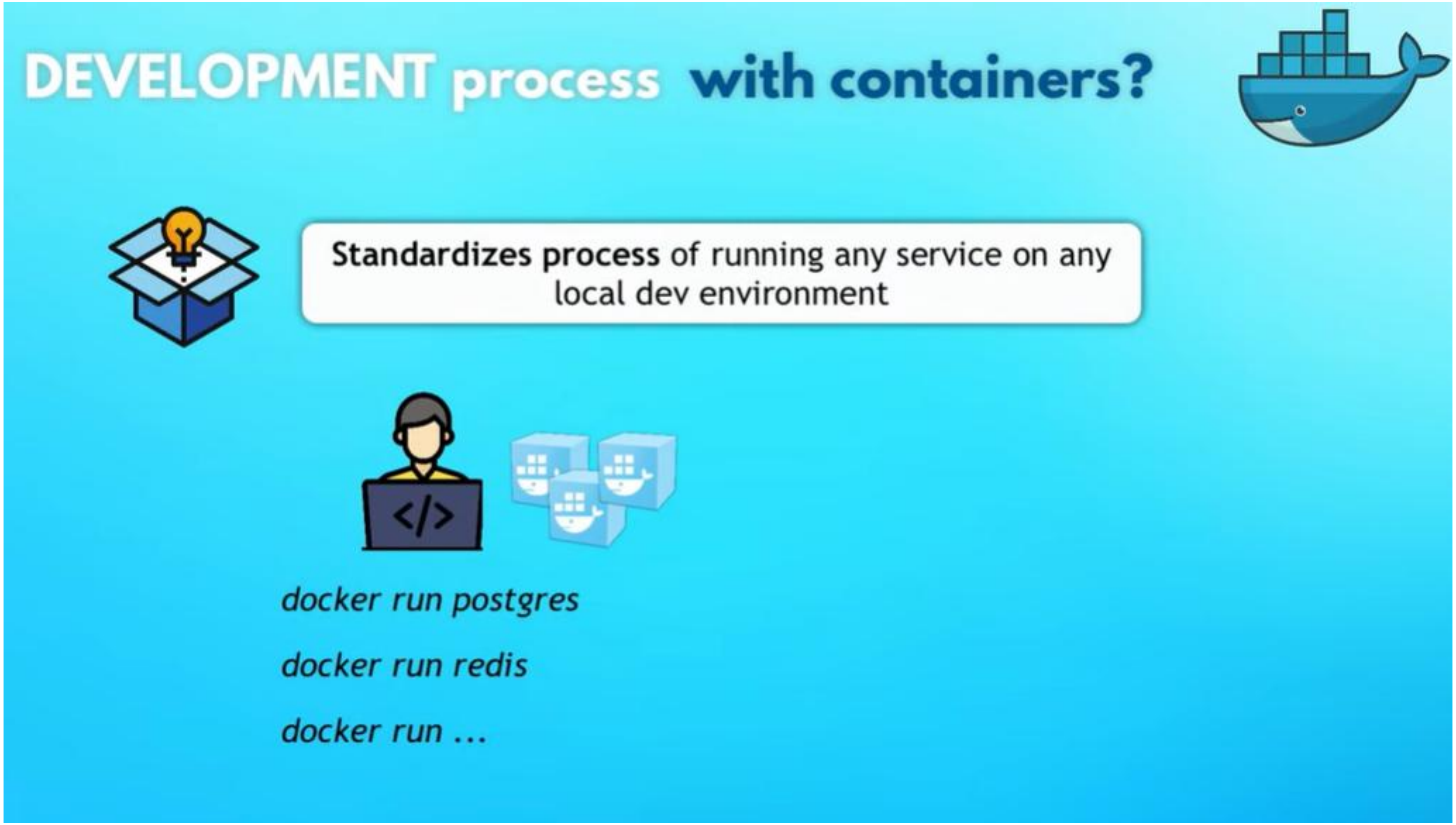
✓ Start service as a Docker container using a **1 Docker command**

✓ Command same for all OS

✓ Command same for all services

docker run postgres

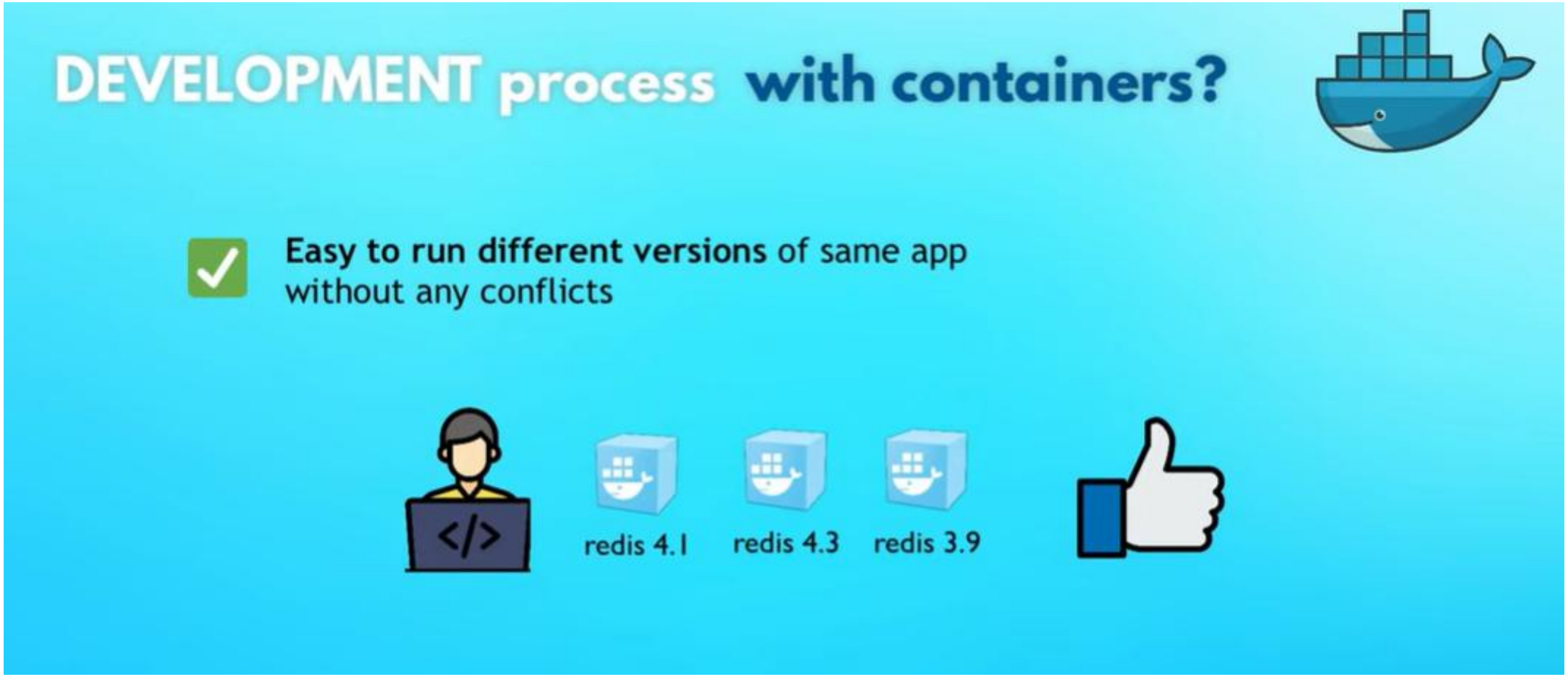
docker run postgres



DEVELOPMENT process with containers?

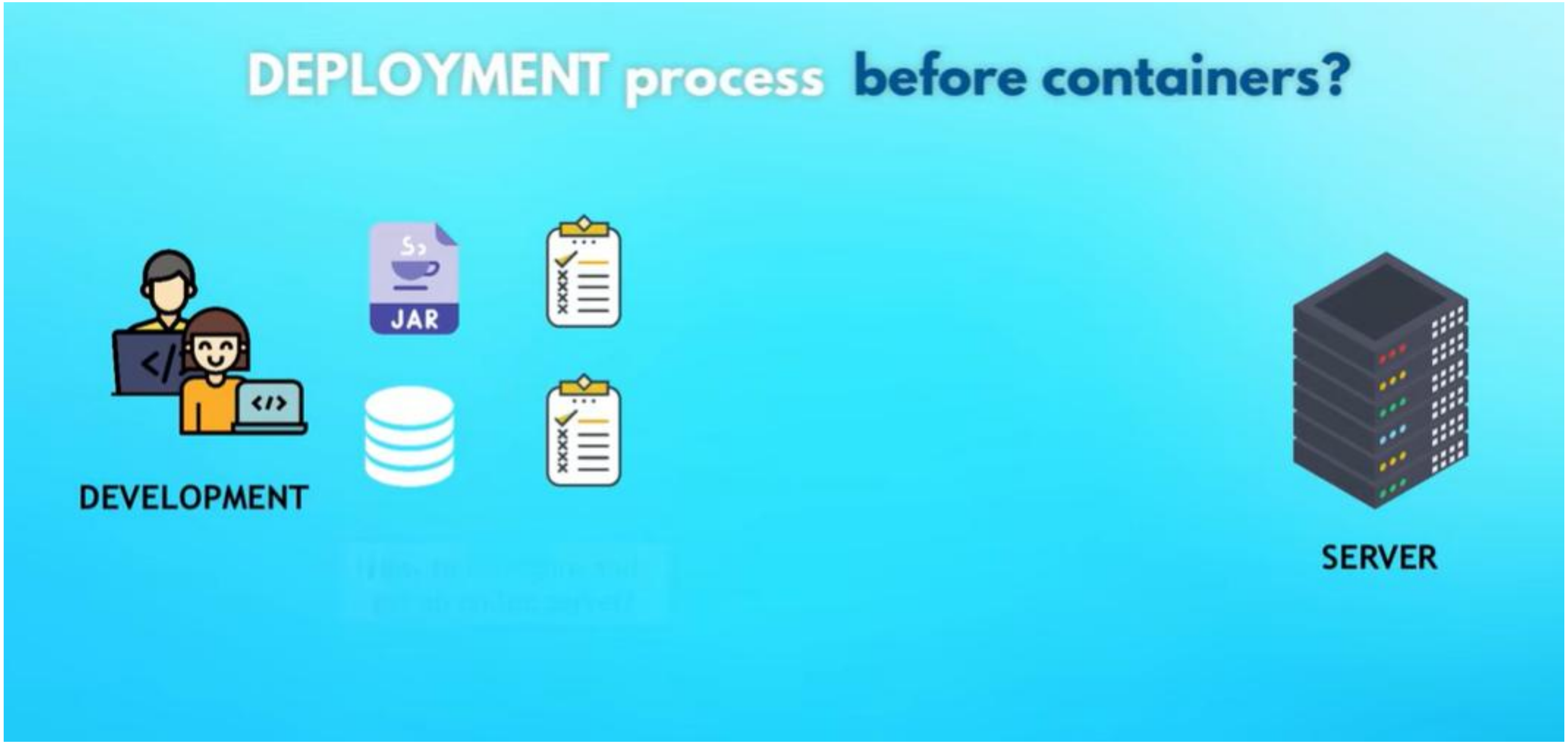
Standardizes process of running any service on any local dev environment

```
docker run postgres

docker run redis

docker run ...
```

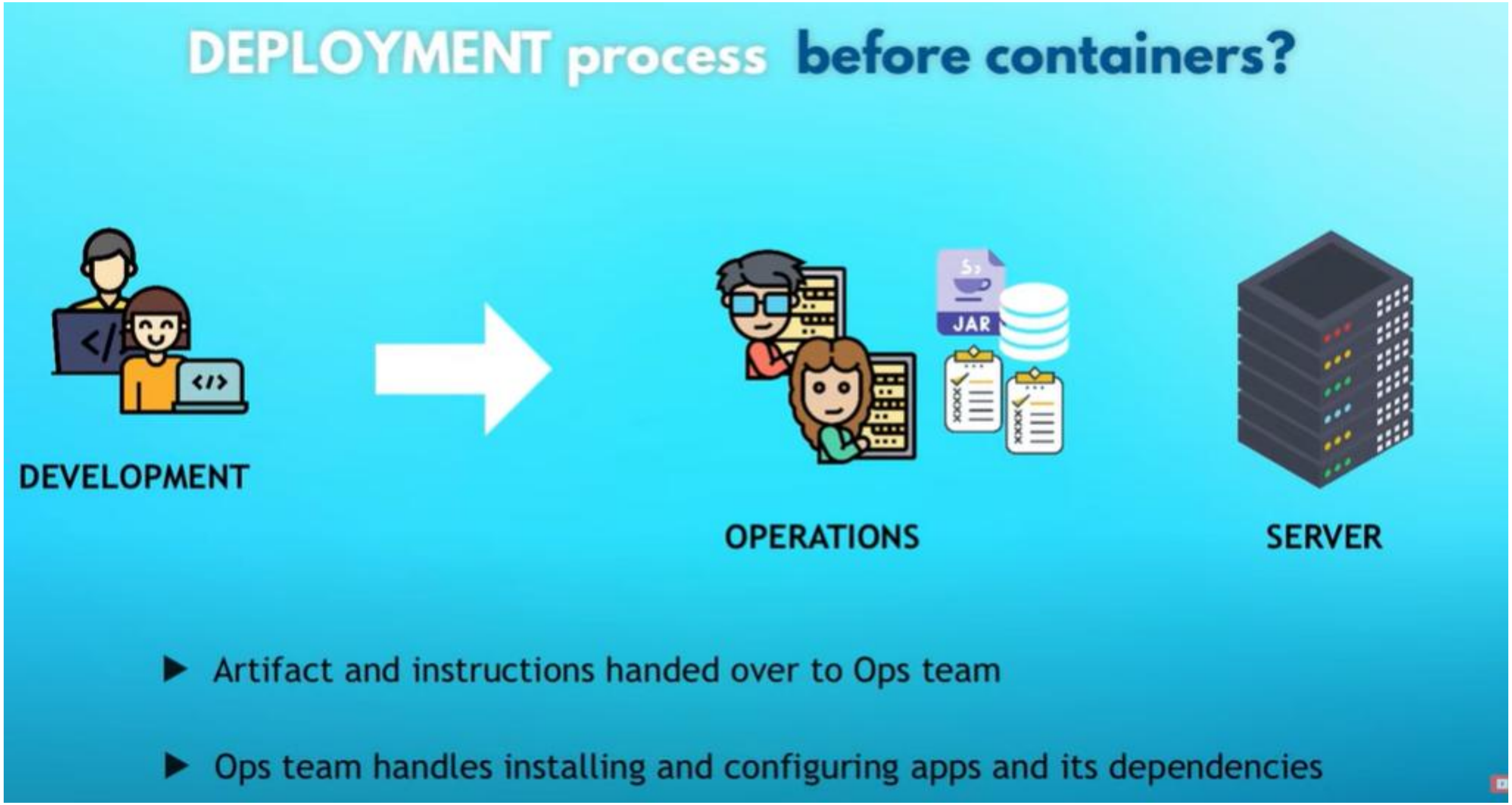


DEVELOPMENT process with containers?

☑ **Easy to run different versions** of same app without any conflicts

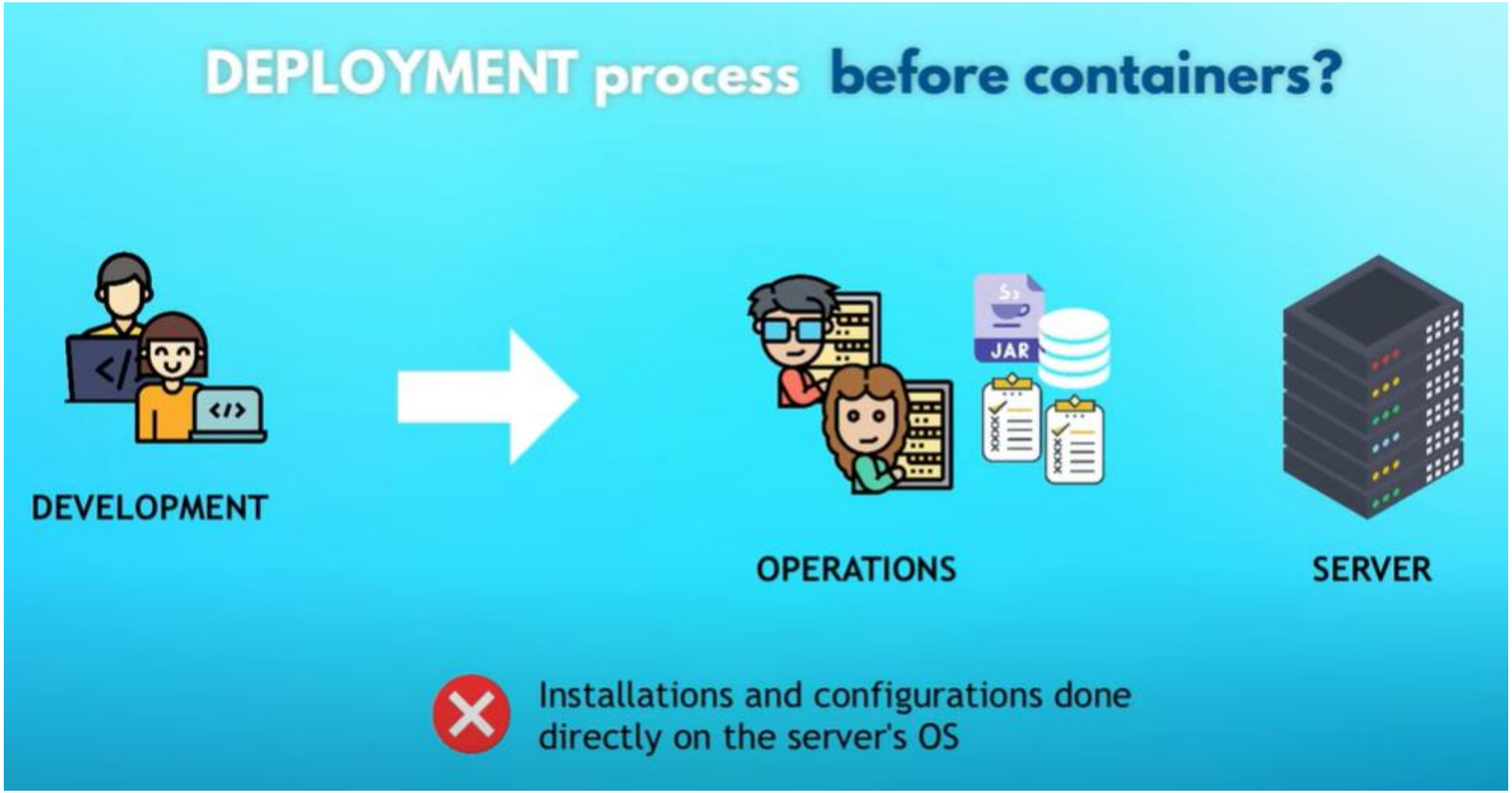


DEPLOYMENT process before containers?



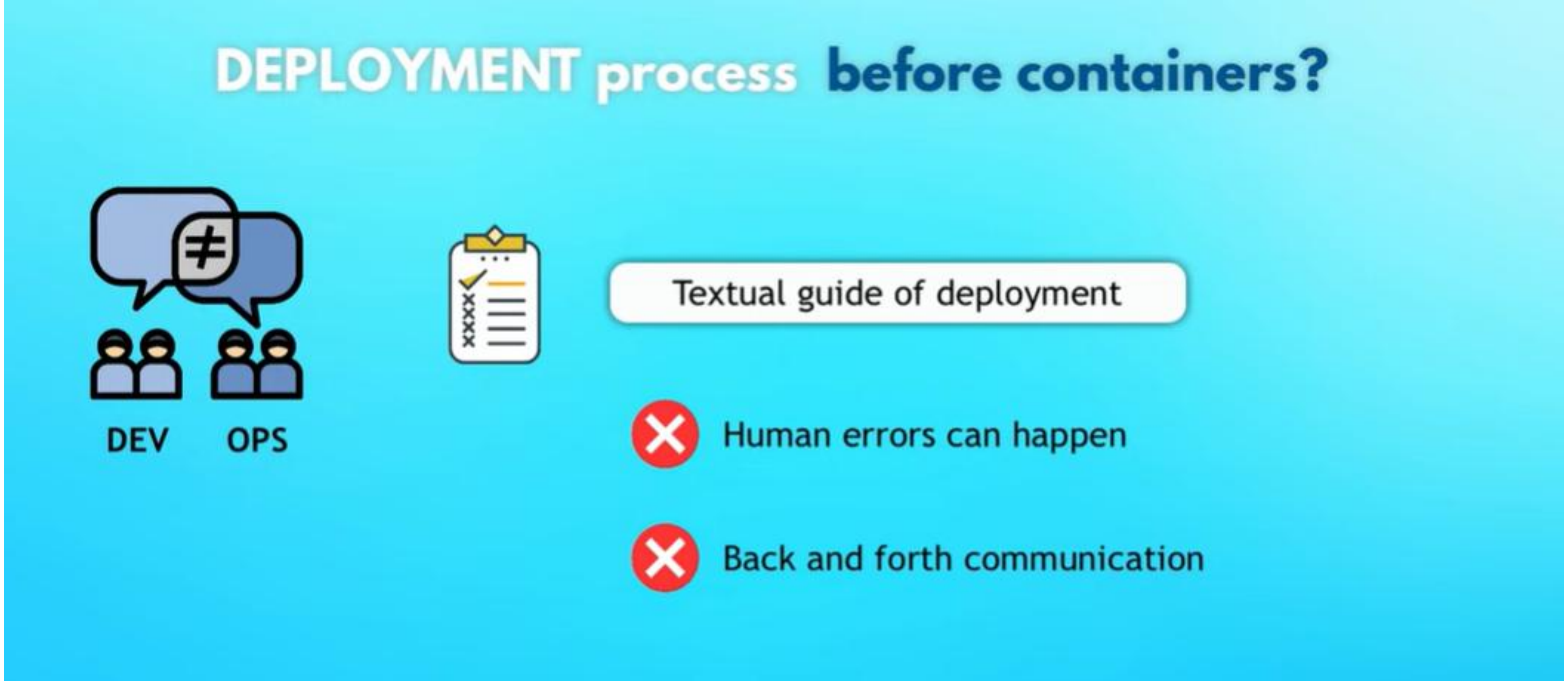
DEPLOYMENT process before containers?

- Artifact and instructions handed over to Ops team
- Ops team handles installing and configuring apps and its dependencies



DEPLOYMENT process before containers?

❌ Installations and configurations done directly on the server's OS



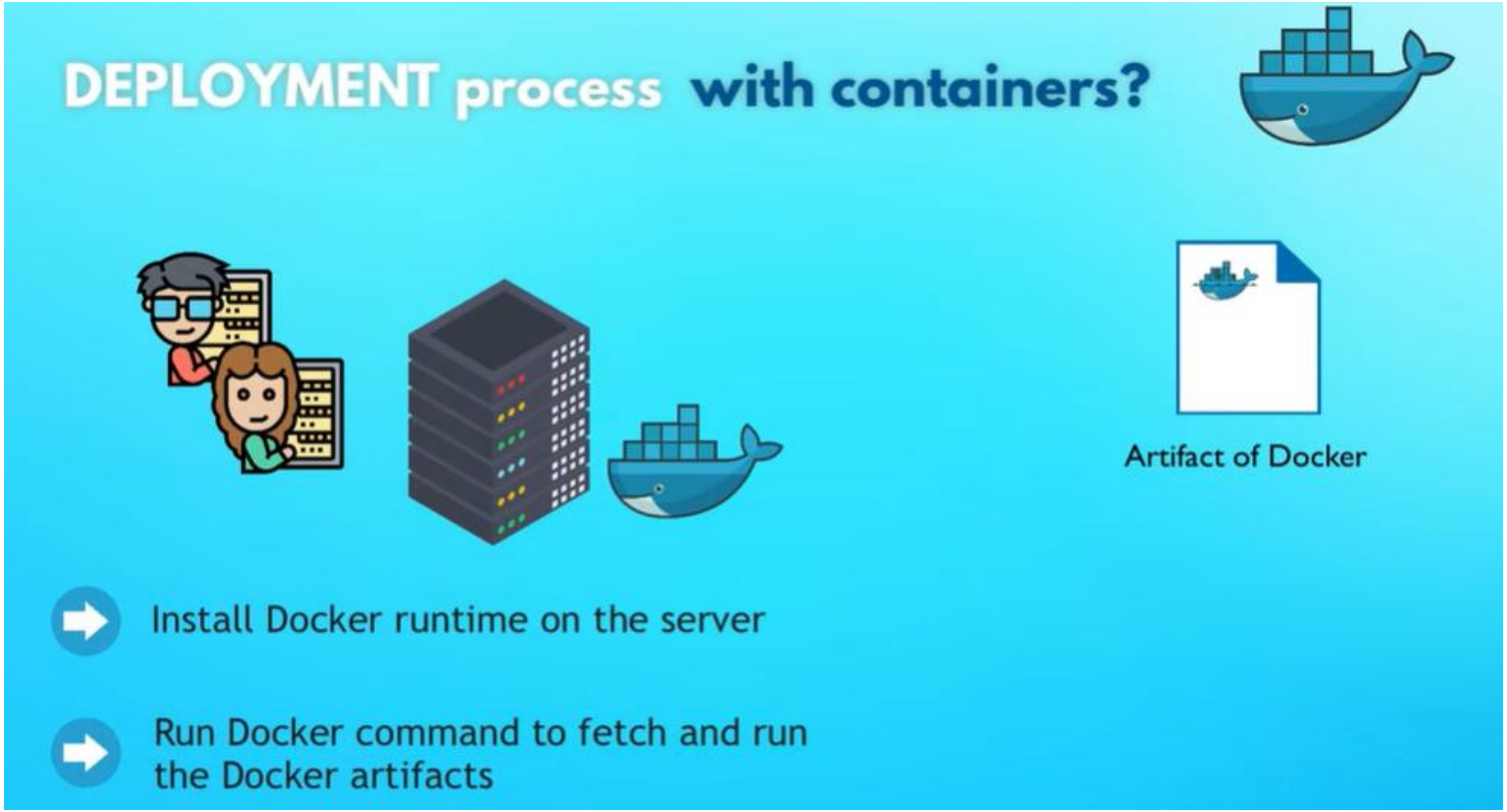
DEPLOYMENT process before containers?

DEV OPS

Textual guide of deployment

❌ Human errors can happen

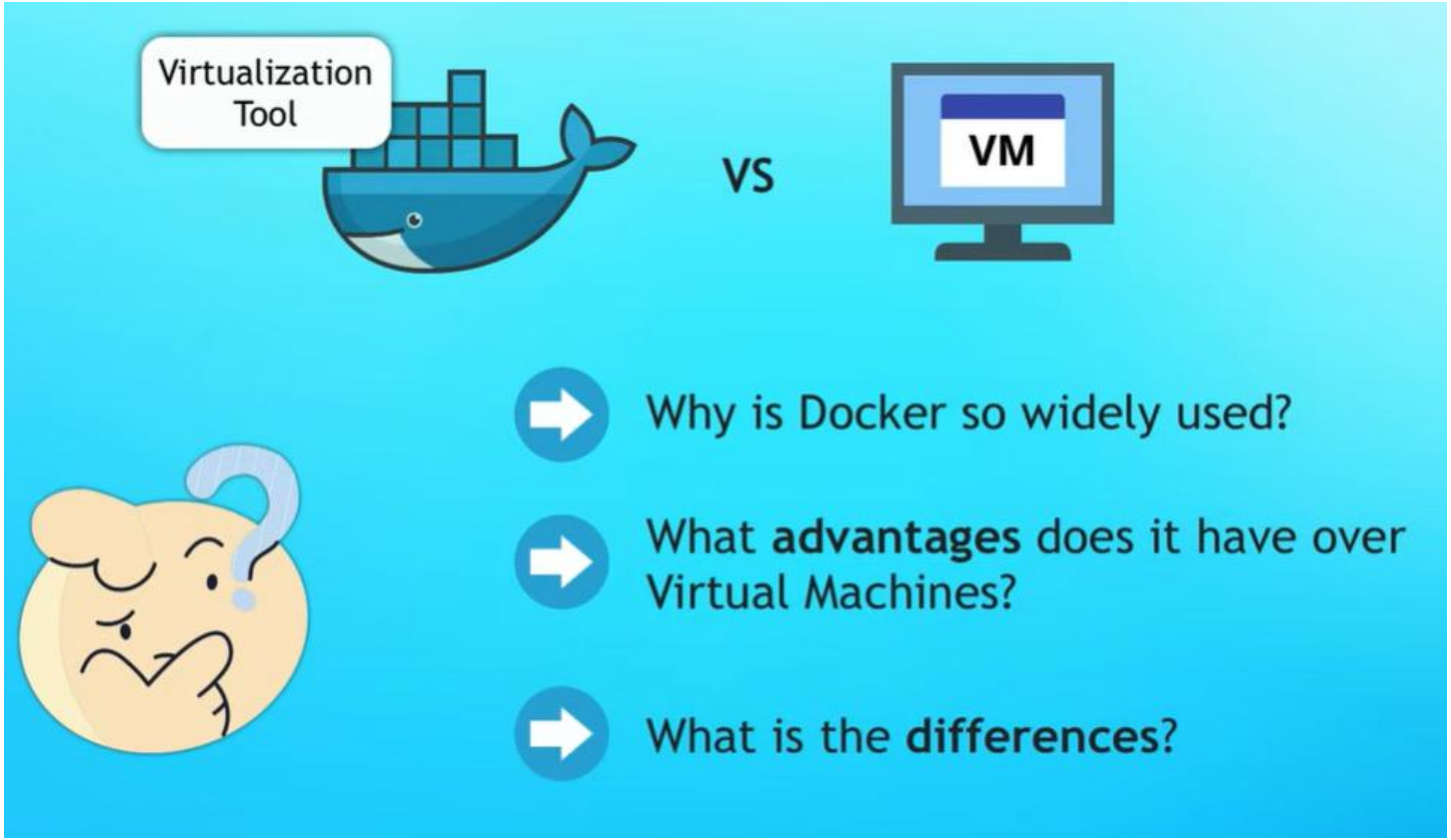
❌ Back and forth communication



DEPLOYMENT process with containers?

Artifact of Docker

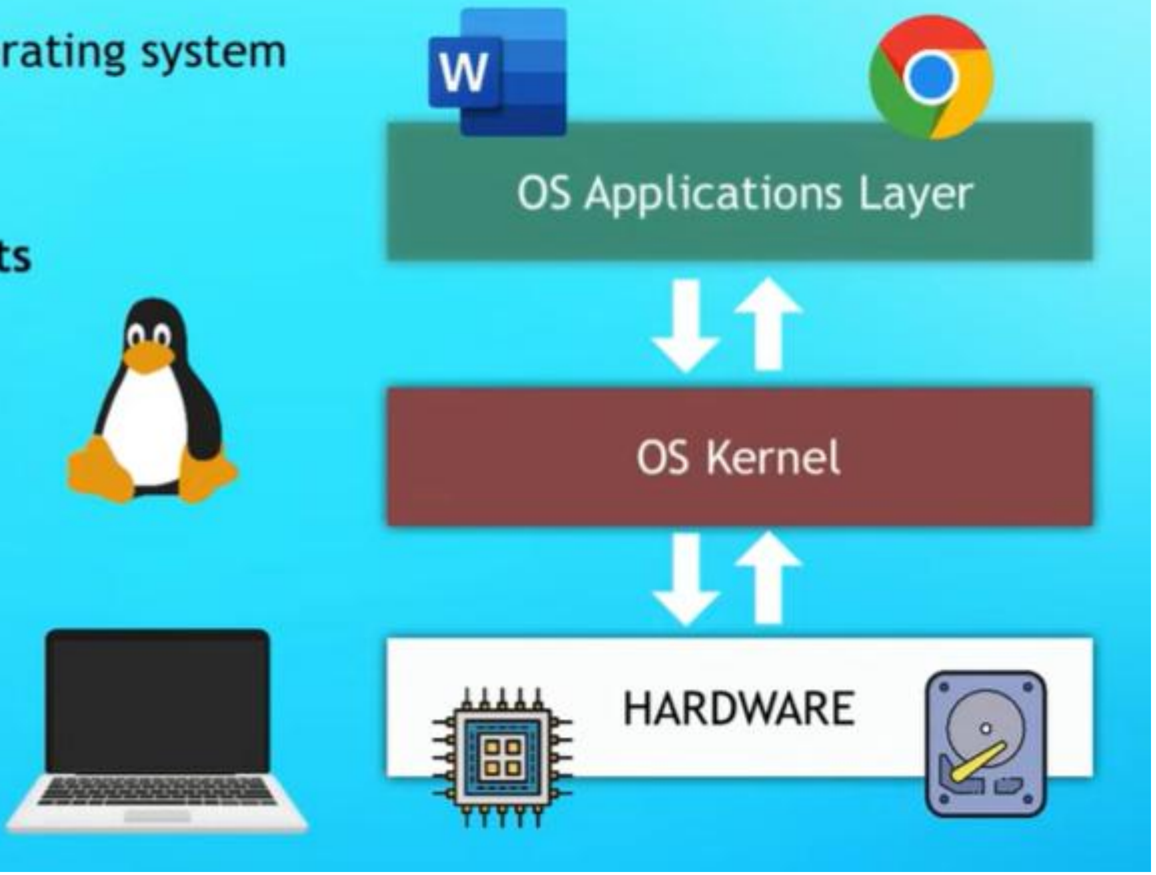
- Install Docker runtime on the server
- Run Docker command to fetch and run the Docker artifacts



- Why is Docker so widely used?
- What **advantages** does it have over Virtual Machines?
- What is the **differences**?

How an OS is made up

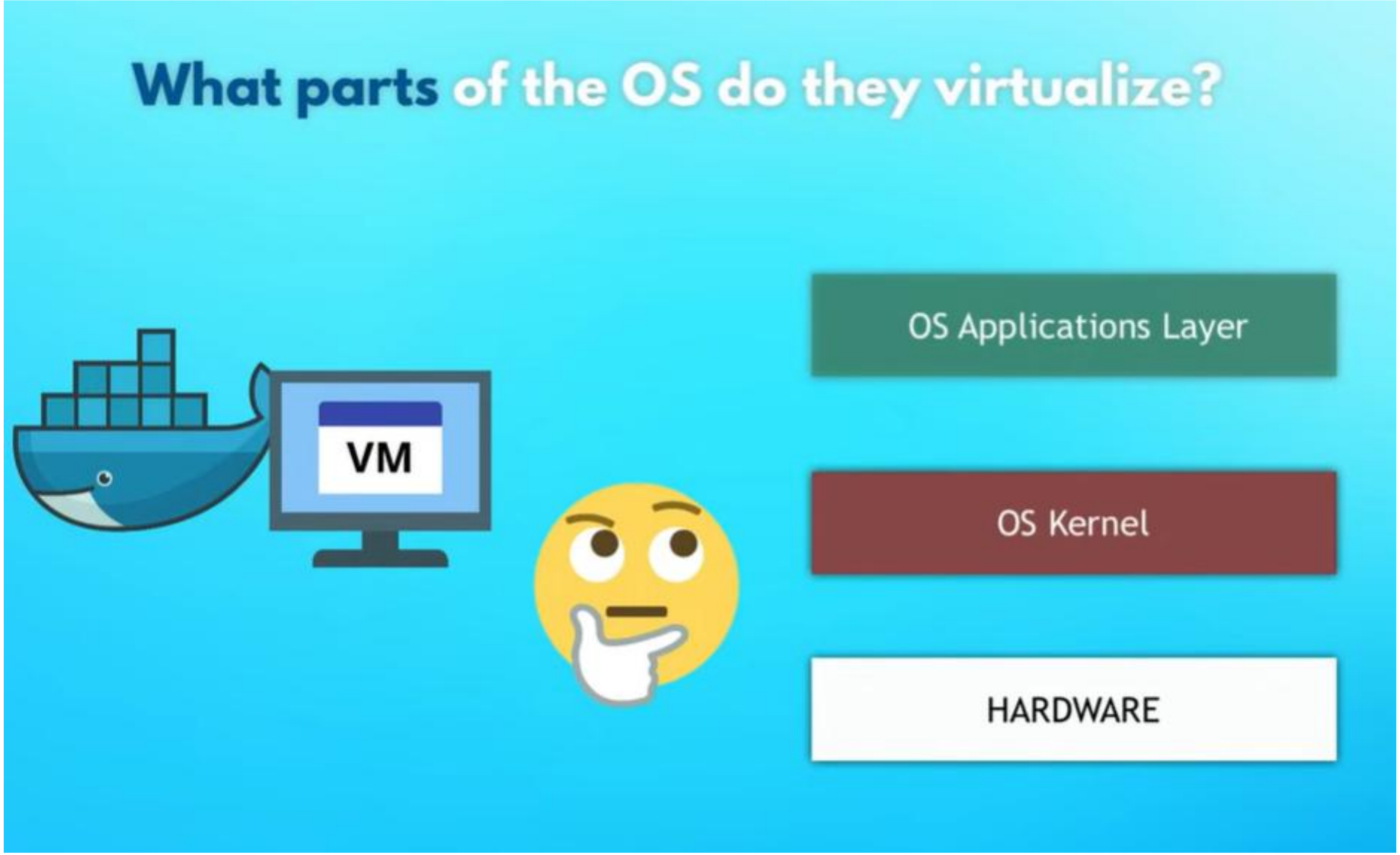
- Kernel is at the **core** of every operating system
- Kernel **interacts between hardware & software components**



OS Applications Layer

OS Kernel

HARDWARE

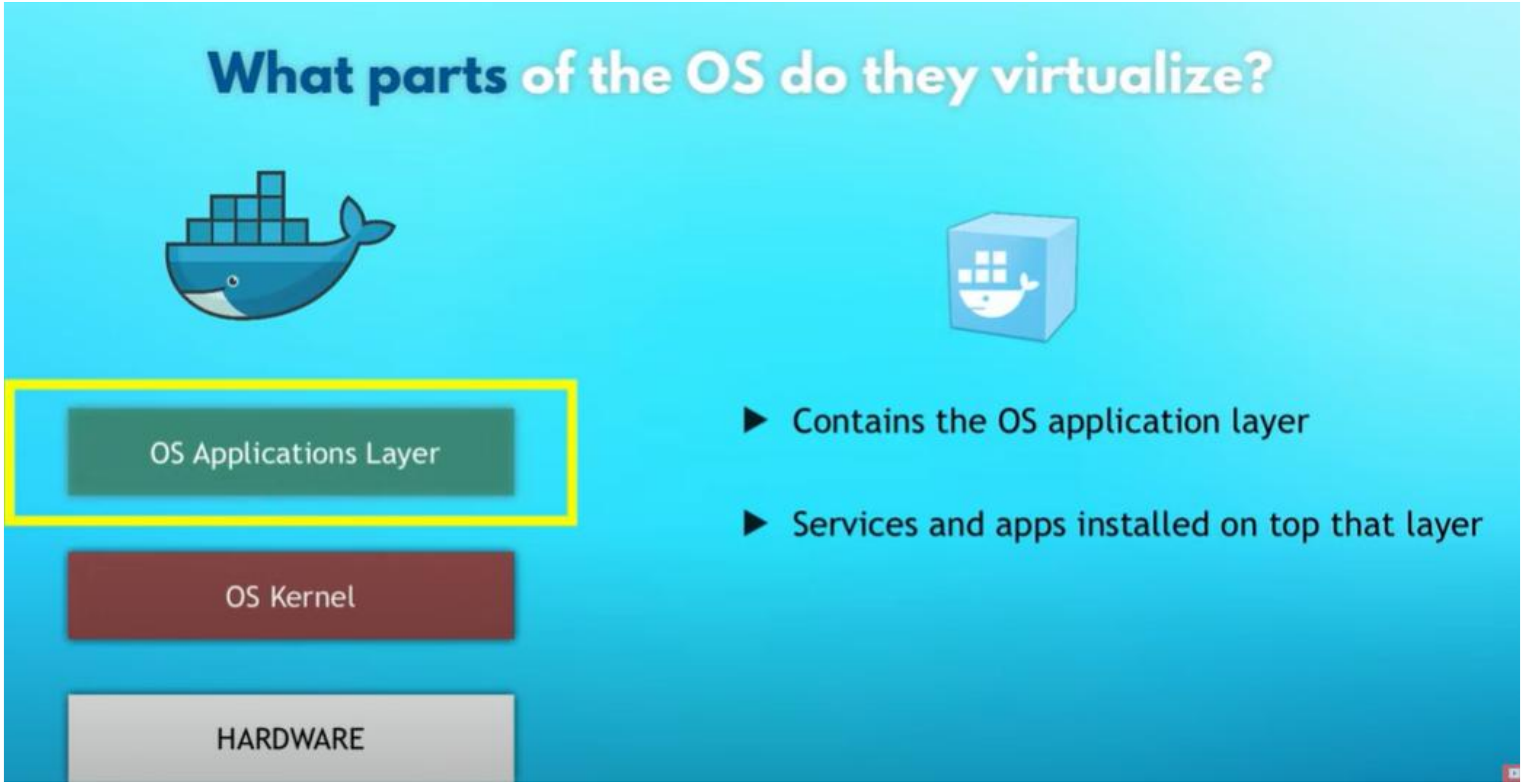


What parts of the OS do they virtualize?

OS Applications Layer

OS Kernel

HARDWARE



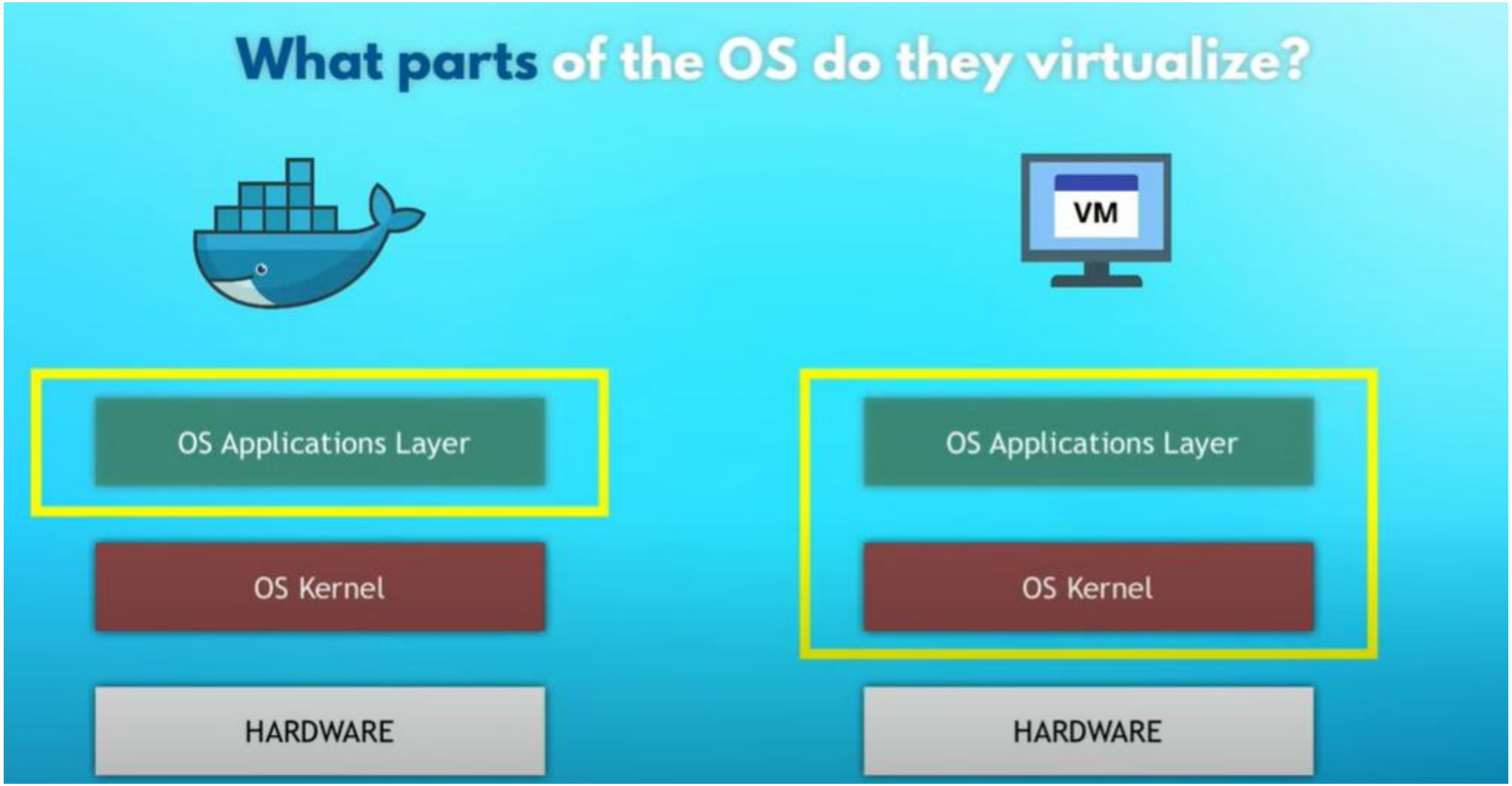
What parts of the OS do they virtualize?

OS Applications Layer

OS Kernel

HARDWARE

- Contains the OS application layer
- Services and apps installed on top that layer



What parts of the OS do they virtualize?

OS Applications Layer

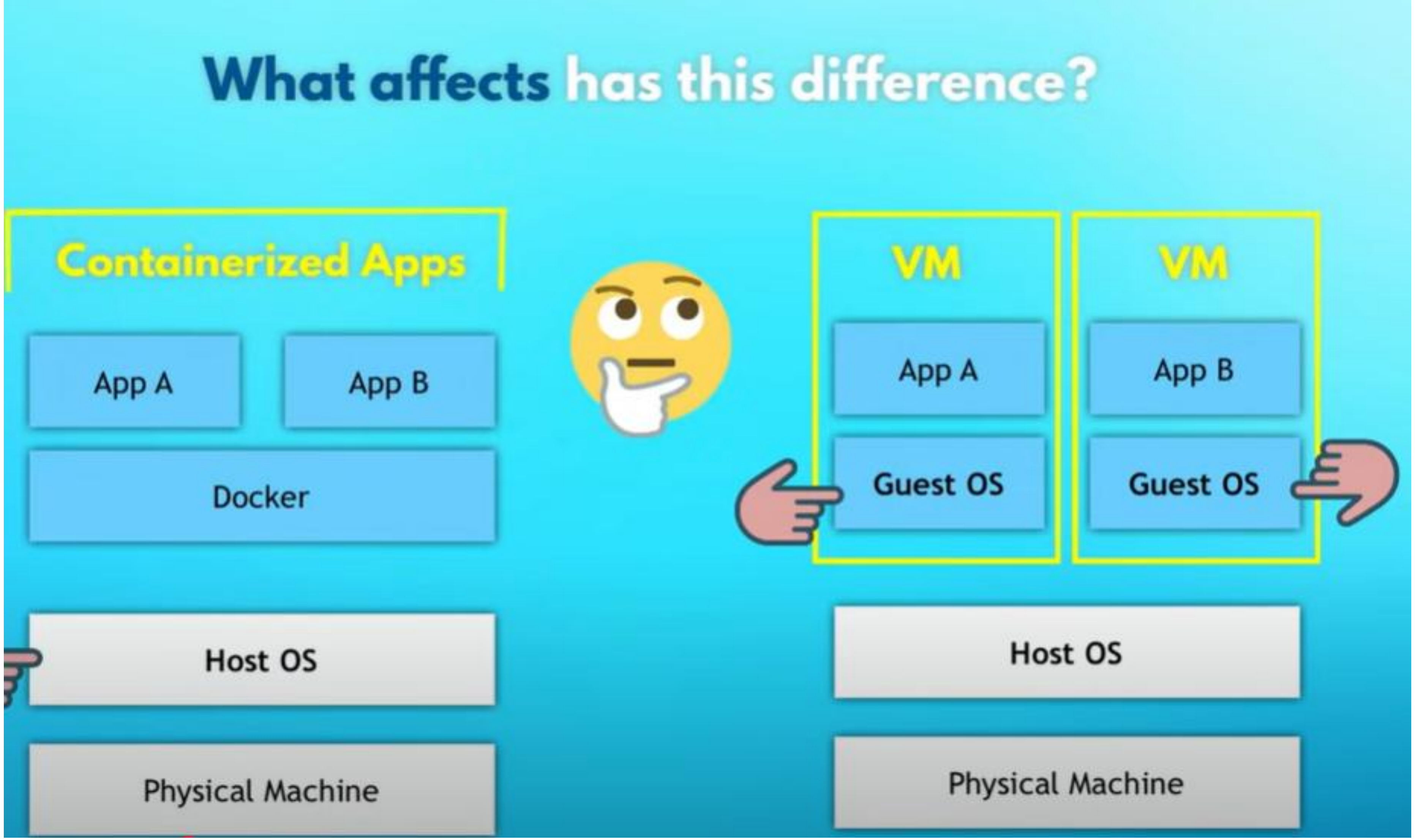
OS Kernel

HARDWARE

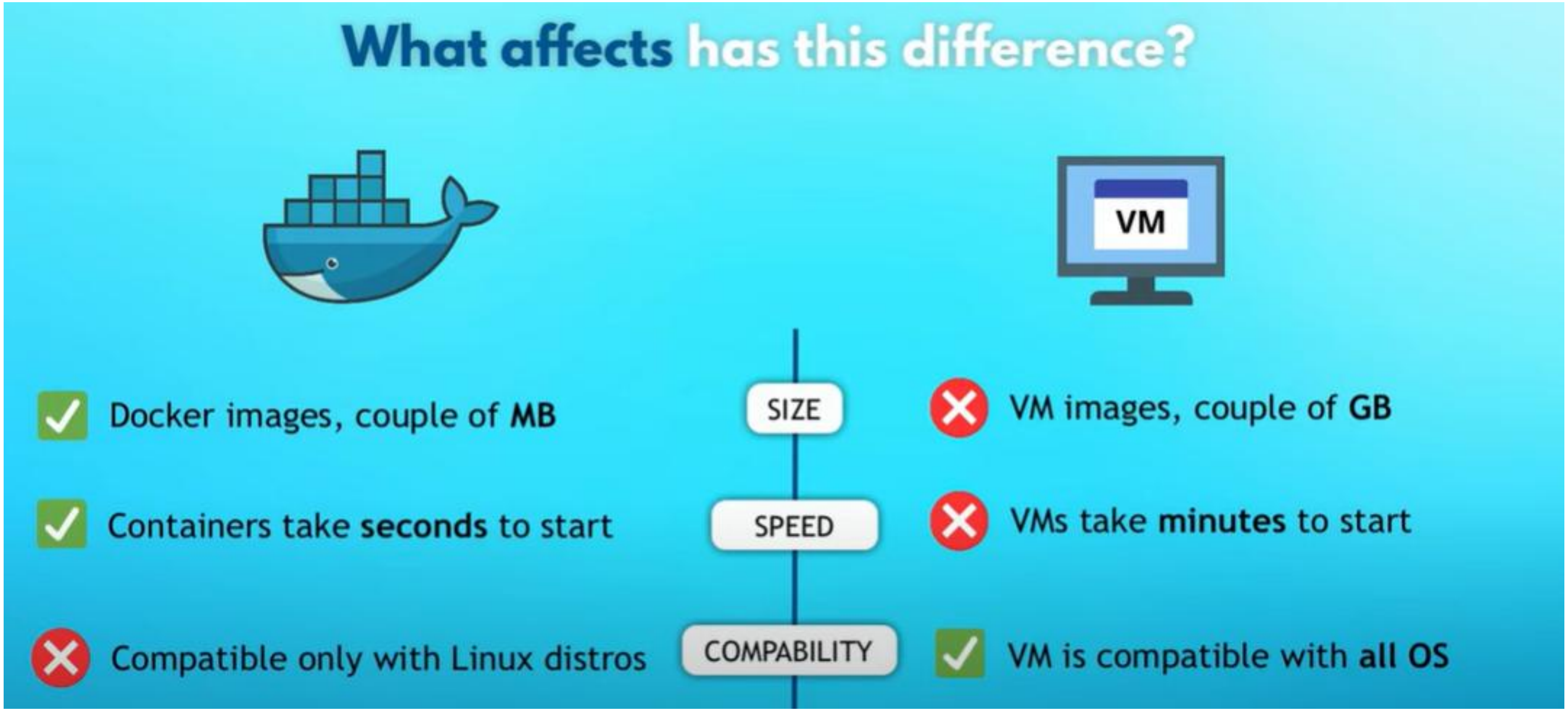
OS Applications Layer

OS Kernel

HARDWARE

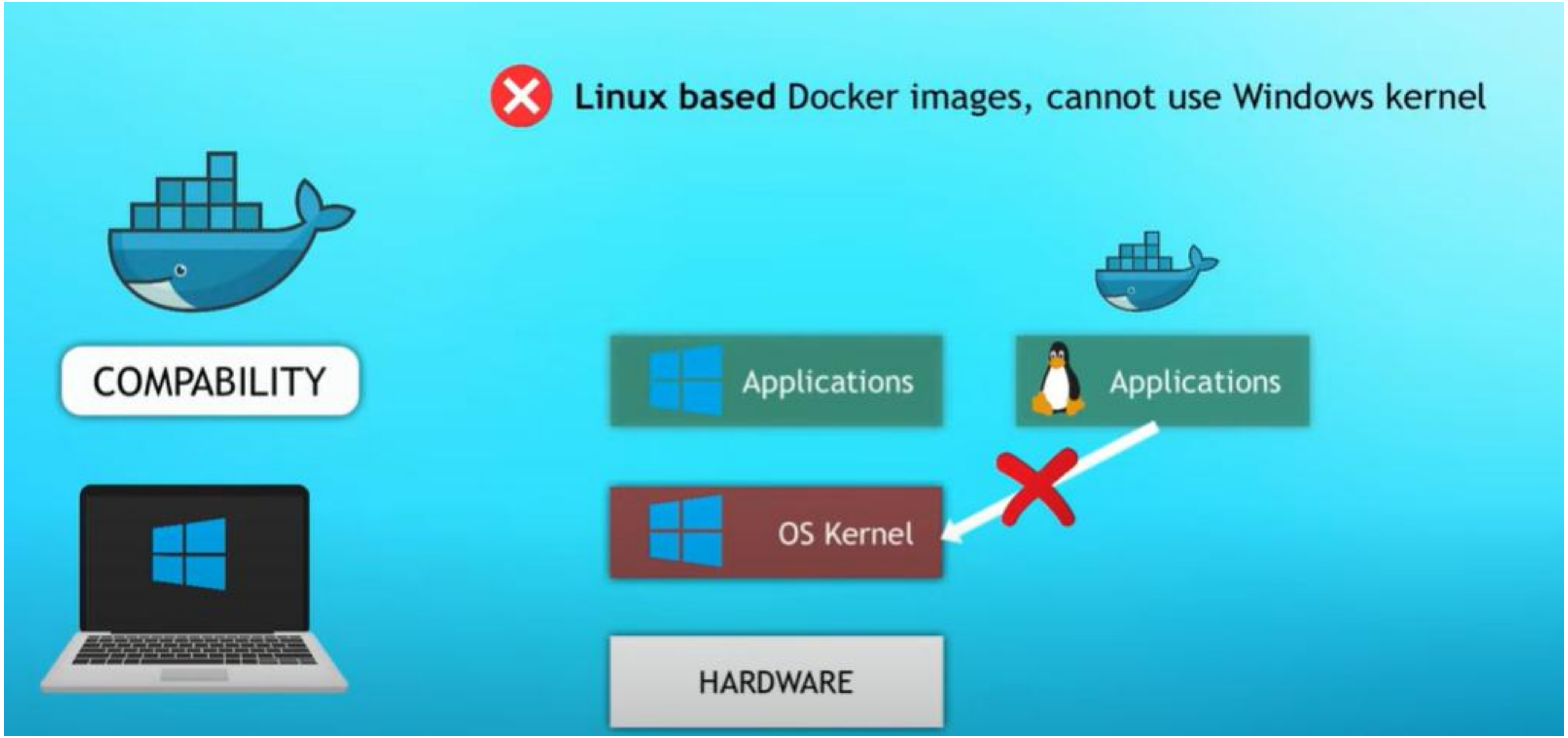


What affects has this difference?

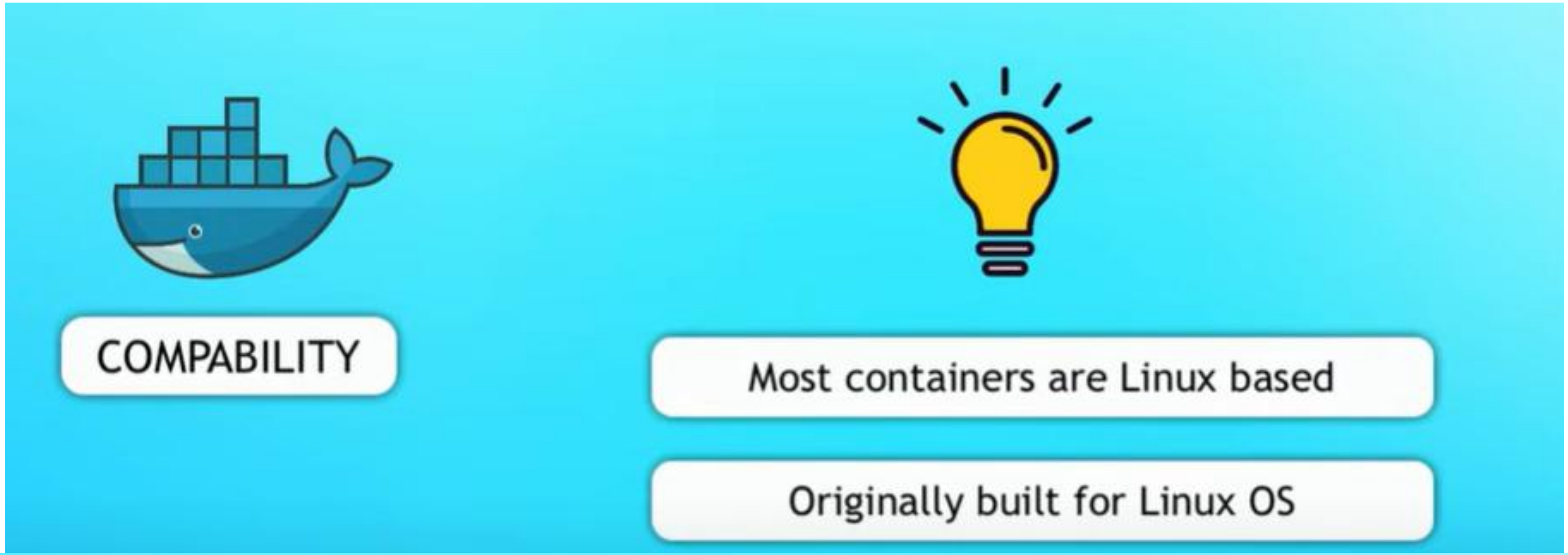


What affects has this difference?

✓ Docker images, couple of **MB**	SIZE	✗ VM images, couple of **GB**
✓ Containers take **seconds** to start	SPEED	✗ VMs take **minutes** to start
✗ Compatible only with Linux distros	COMPABILITY	✓ VM is compatible with **all OS**



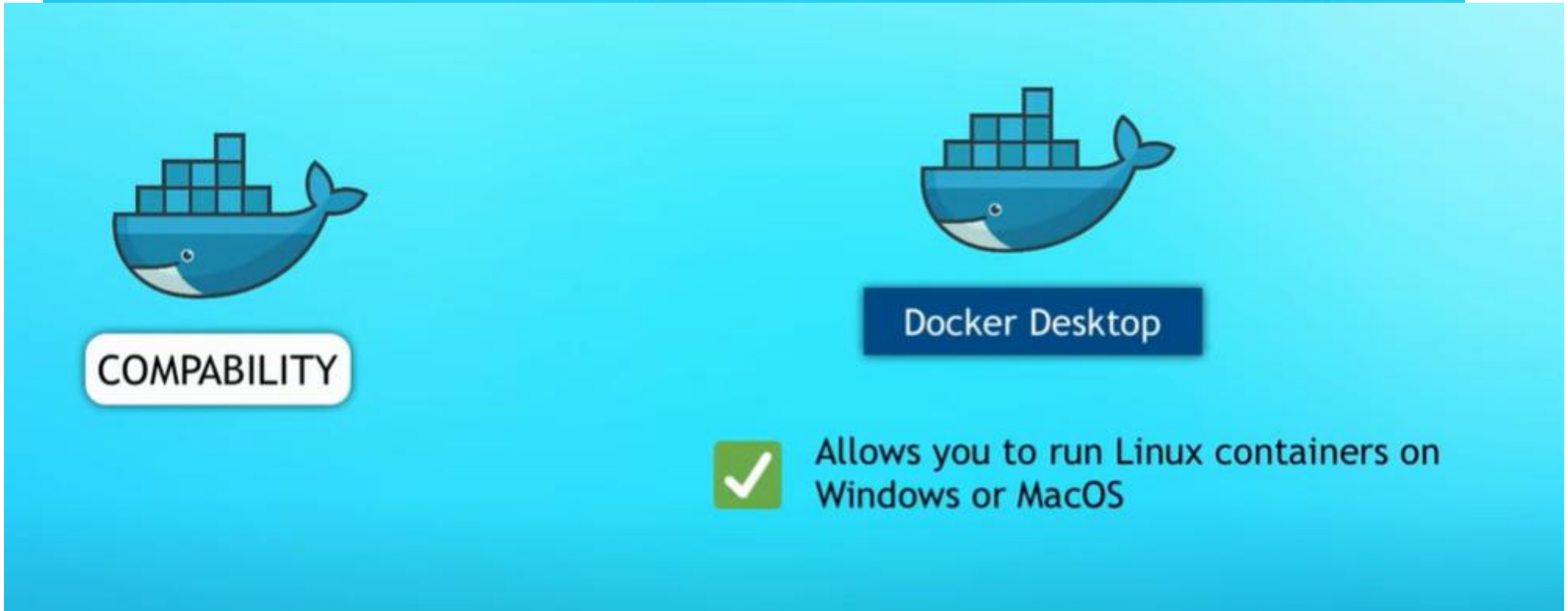
Linux based Docker images, cannot use Windows kernel



COMPABILITY

Most containers are Linux based

Originally built for Linux OS



COMPABILITY

Docker Desktop

✓ Allows you to run Linux containers on Windows or MacOS

Abstract Architecture of Docker

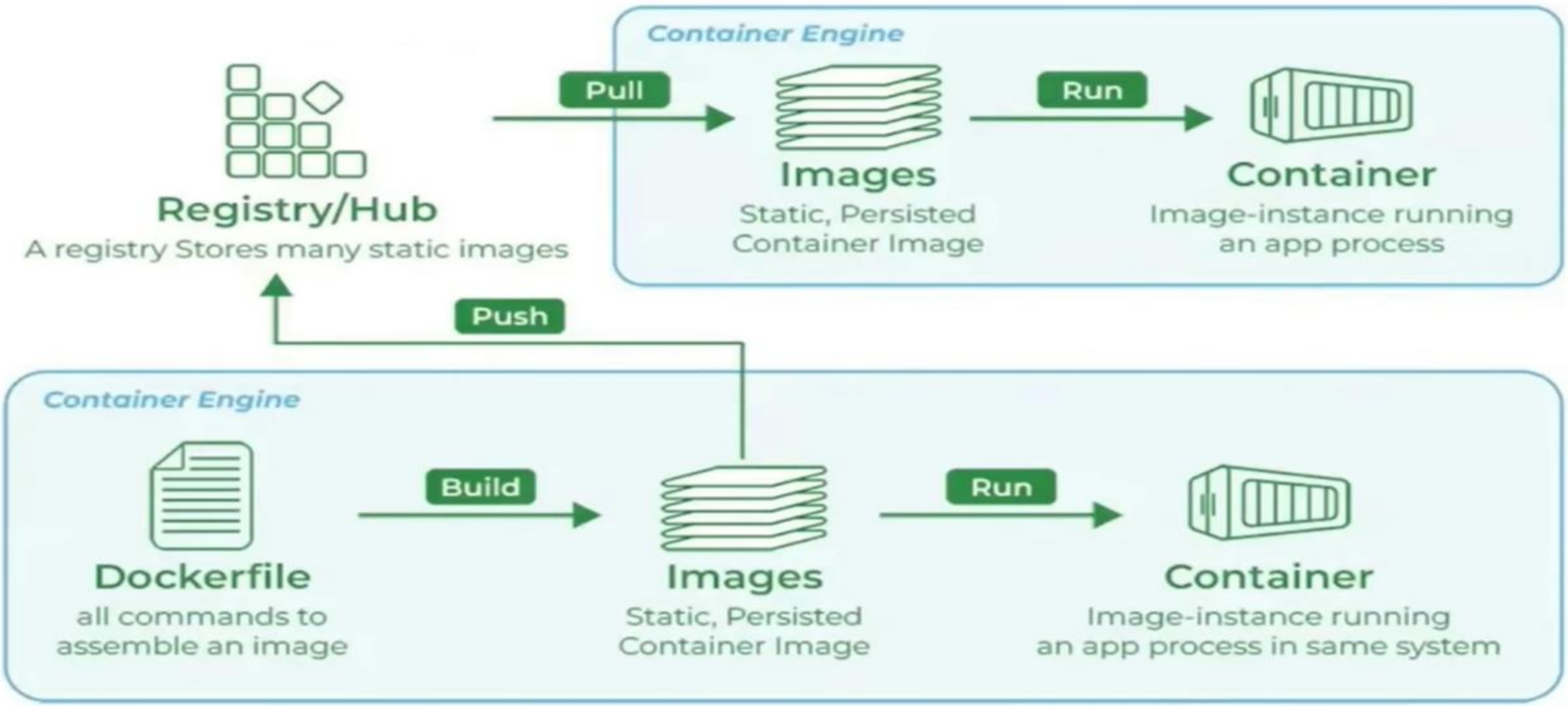
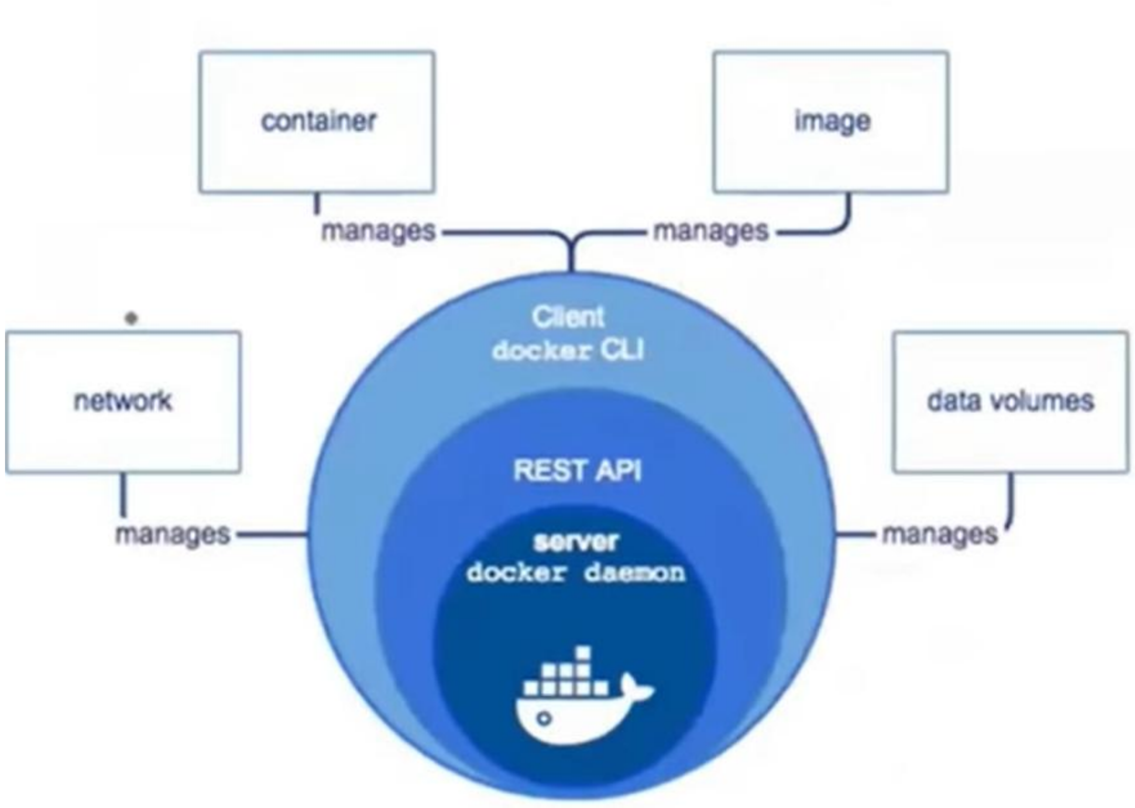


Image credit: Geeksforgeeks

- **Docker Engine**
- Docker Engine is the core component of the Docker platform, responsible for creating, running, and managing Docker containers. It serves as the runtime that powers Docker's containerization capabilities. Here’s an in-depth look at the Docker Engine.



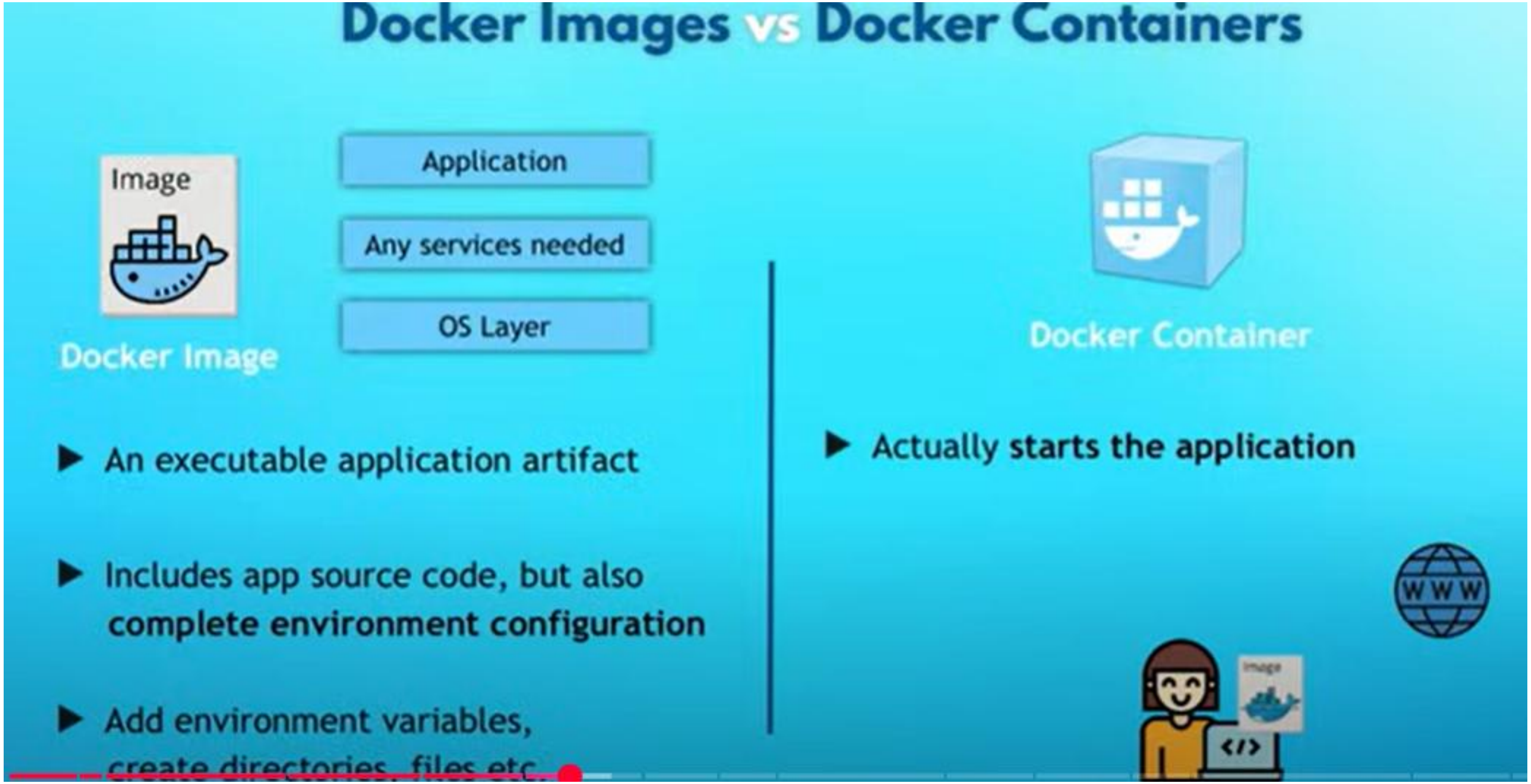
Components of Docker Engine

- **Docker Daemon (dockerd):**
- **Function:** The Docker daemon is the background service running on the host machine. It manages Docker objects such as images, containers, networks, and volumes.
- **Interaction:** It listens for Docker API requests and processes them, handling container lifecycle operations (start, stop, restart, etc.).
- **Docker CLI (docker):**
- **Function:** The Docker Command Line Interface (CLI) is the tool that users interact with to communicate with the Docker daemon.
- **Usage:** Users run Docker commands through the CLI to perform tasks like building images, running containers, and managing Docker resources.

- REST API:
- Function: The Docker REST API allows communication between the Docker CLI and the Docker daemon. It also enables programmatic interaction with Docker.
- Usage: Developers can use the API to automate Docker operations or integrate Docker functionality into their applications.

- Docker Image
- A Docker image is a lightweight, stand-alone, and executable software package that includes everything needed to run a piece of software, such as the code, runtime, libraries, environment variables, and configuration files. Images are used to create Docker containers, which are instances of these images.
- Components of a Docker Image
- Base Image: The starting point for building an image. It could be a minimal OS image like alpine, a full-fledged OS like ubuntu, or even another application image like python or node.
- Application Code: The actual code and files necessary for the application to run.
- Dependencies: Libraries, frameworks, and packages required by the application.
- Metadata: Information about the image, such as environment variables, labels, and exposed

- Docker Image Lifecycle
- Creation: Images are created using the docker build command, which processes the instructions in a Dockerfile to create the image layers.
- Storage: Images are stored locally on the host machine. They can also be pushed to and pulled from Docker registries like Docker Hub, AWS ECR, or Google Container Registry.
- Distribution: Images can be shared by pushing them to a Docker registry, allowing others to pull and use the same image.
- Execution: Images are executed by running containers, which are instances of these images.
-



Docker Images vs Docker Containers

Application

Any services needed

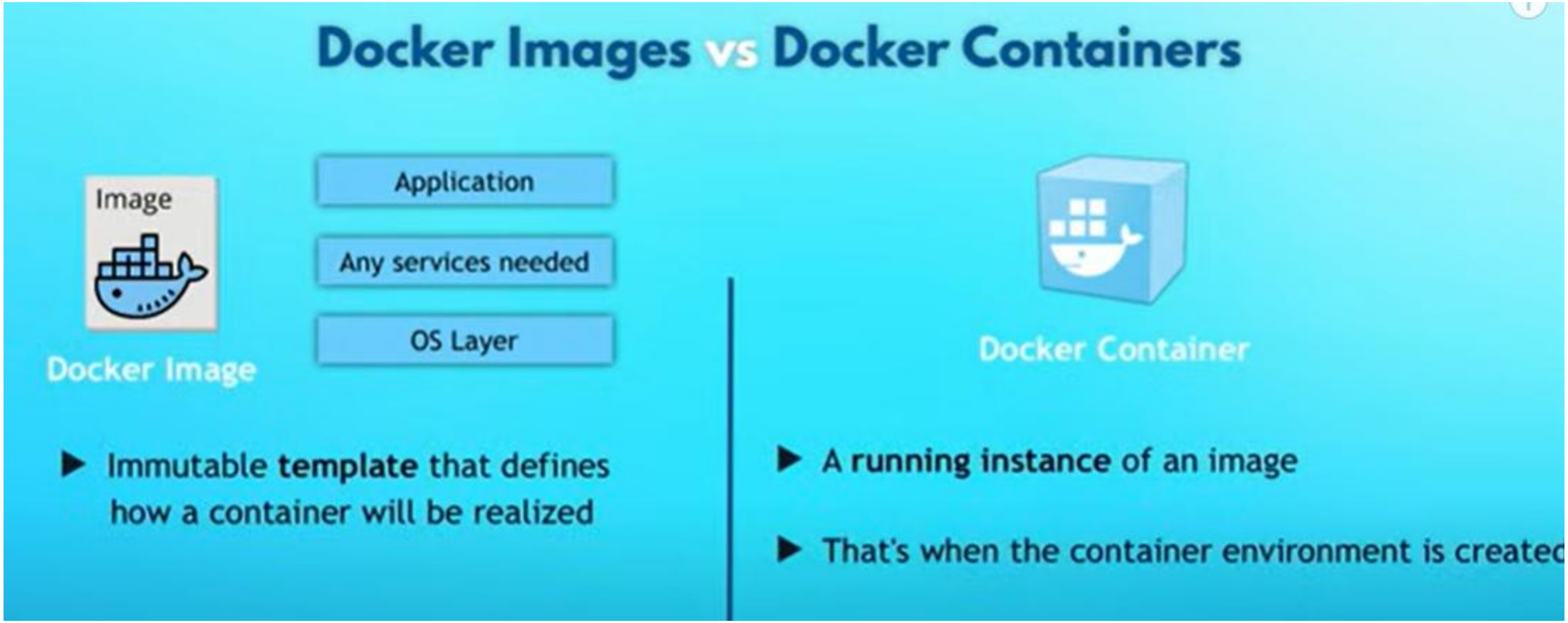
OS Layer

Docker Image

Docker Container

- An executable application artifact
- Includes app source code, but also **complete environment configuration**
- Add environment variables, create directories, files etc.

- Actually **starts the application**



Docker Images vs Docker Containers

Image

Docker Image

Application

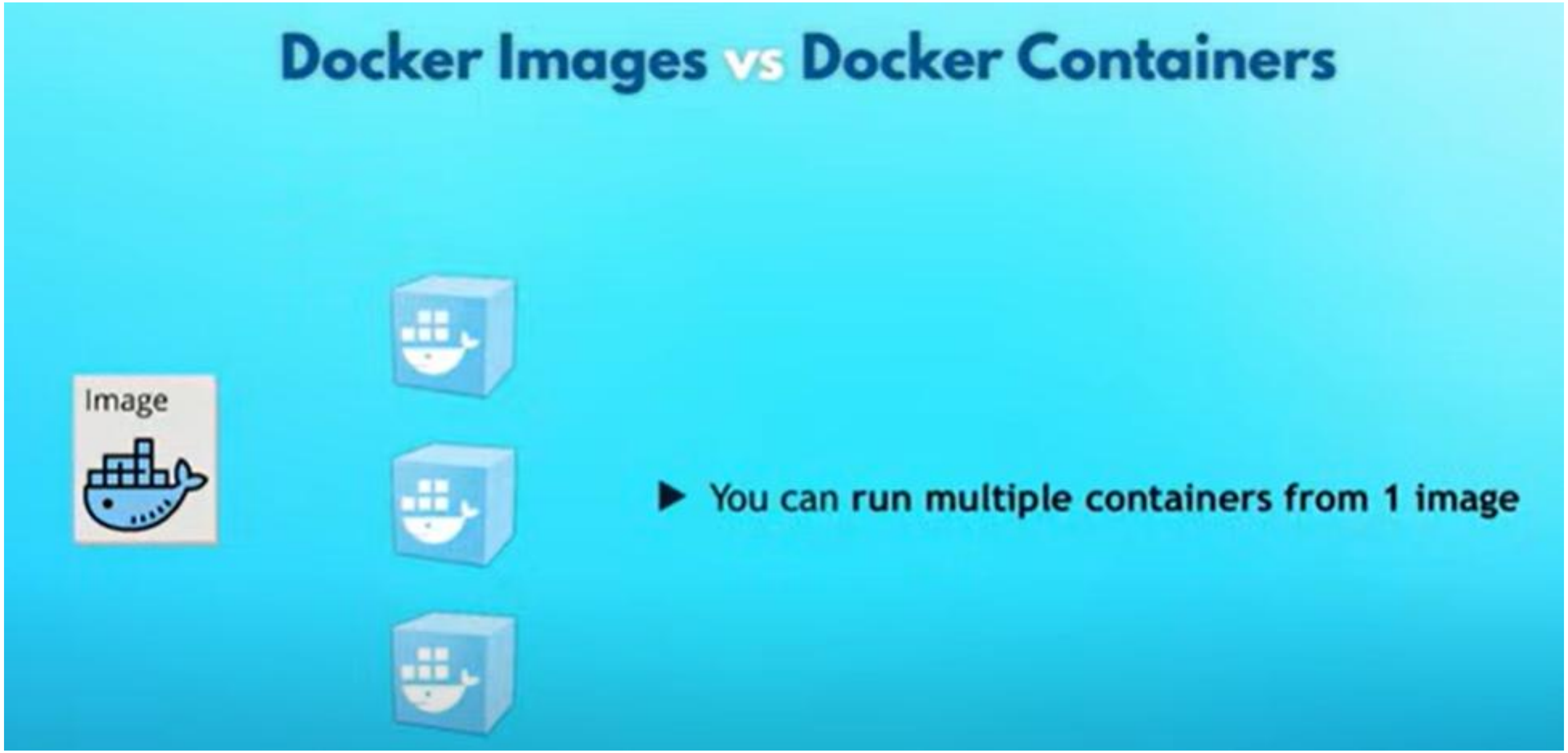
Any services needed

OS Layer

- Immutable **template** that defines how a container will be realized

Docker Container

- A **running instance** of an image
- That's when the container environment is createc



Docker Images vs Docker Containers

- You can **run multiple containers from 1 image**

- Dockerfile
- A Dockerfile is a text file that contains a series of instructions used to build a Docker image. Each instruction in a Dockerfile creates a layer in the image, allowing for efficient image creation and reuse of layers. Dockerfiles are used to automate the image creation process, ensuring consistency and reproducibility.
- Key Components of a Dockerfile

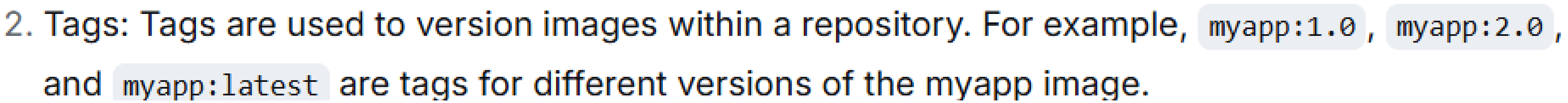
1. Base Image (FROM): Specifies the starting point for the image, which could be a minimal operating system, a specific version of a language runtime, or another image. Example:
   ```
   FROM ubuntu:20.04
   ```
2. Labels (LABEL): Adds metadata to the image, such as version, description, or maintainer. Example:
   ```
   LABEL version="1.0" description="My application"
   ```
3. Run Commands (RUN): Executes commands in the image during the build process, typically used to install software packages. Example:
   ```
   RUN apt-get update && apt-get install -y python3
   ```
4. Copy Files (COPY): Copies files or directories from the host system to the image. Example:
   ```
   COPY ./app
   ```
5. Environment Variables (ENV): Sets environment variables in the image. Example:
   ```
   ENV PATH /app/bin:$PATH
   ```
6. Work Directory (WORKDIR): Sets the working directory for subsequent instructions. Example:
   ```
   WORKDIR /app
   ```

7. Expose Ports (EXPOSE) - Informs Docker that the container listens on specified network ports. Example: EXPOSE 8080
8. Command (CMD) - Provides a default command to run when the container starts.
   Example: CMD ["python", "app.py"]
9. Volume (VOLUME) - Creates a mount point with a specified path and marks it as holding externally mounted volumes from the host or other containers. Example: VOLUME "/data"
10. Arguments (ARG) - Defines build-time variables. Example: ARG VERSION=1.0

- Docker Container
- A Docker container is a lightweight, portable, and isolated environment that encapsulates an application and its dependencies, allowing it to run consistently across different computing environments. Containers are created from Docker images, which are immutable and contain all the necessary components for the application to run.
- Registry
- A Docker registry is a service that stores and distributes Docker images. It acts as a repository where users can push, pull, and manage Docker images. Docker Hub is the most well-known public registry, but private registries can also be set up to securely store and manage images within an organization.

myapp:1.0, myapp:2.0, and myapp:latest are tags for different versions of the myapp image

- Key Components of a Docker Registry
- Repositories: A repository is a collection of related Docker images, typically different versions of the same application. Each repository can hold multiple tags, representing different versions of an image.

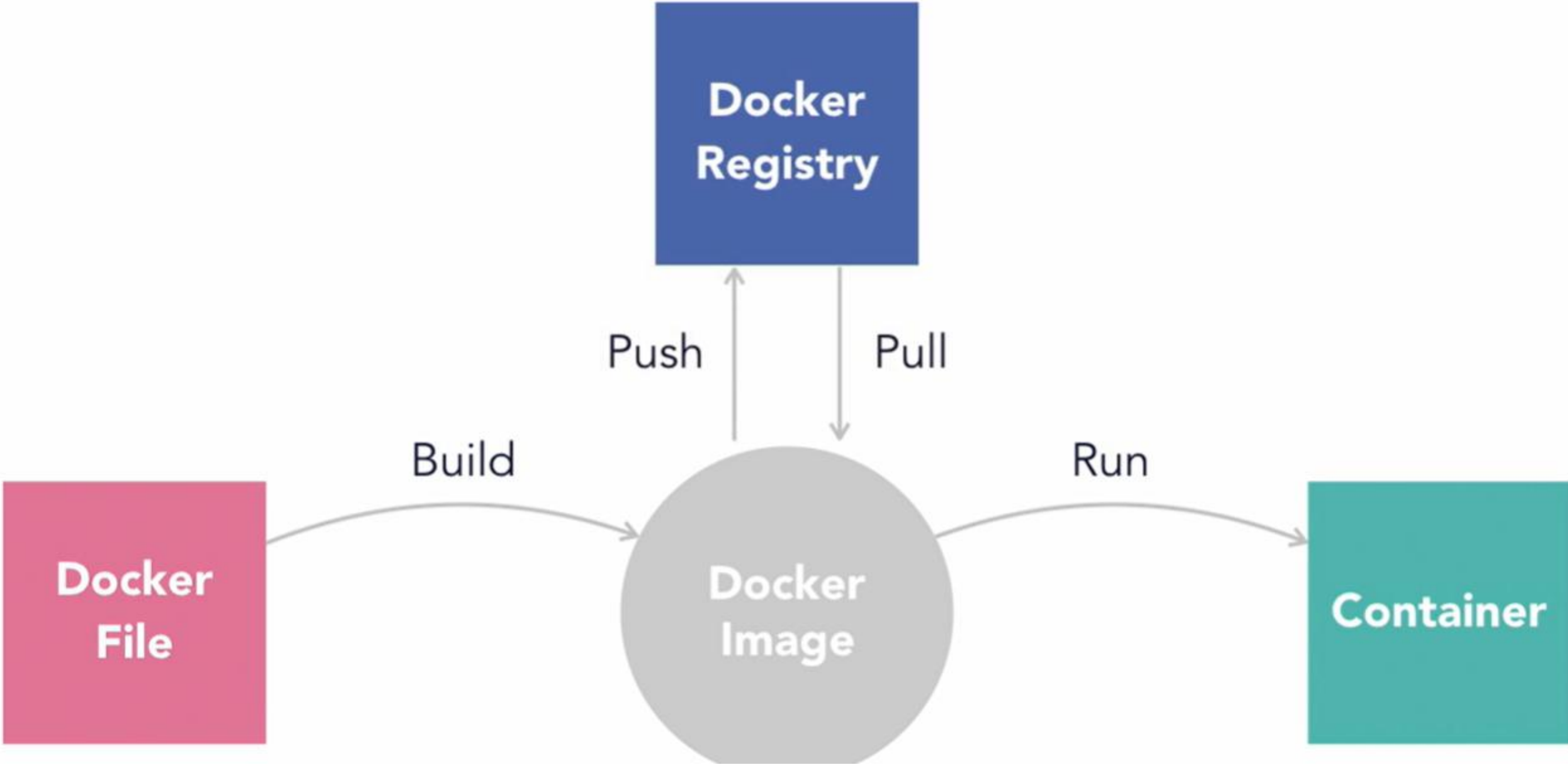


2. Tags: Tags are used to version images within a repository. For example, `myapp:1.0`, `myapp:2.0`, and `myapp:latest` are tags for different versions of the myapp image.

- Types of Docker Registries
- Docker Hub:
- Description: The default public registry provided by Docker, which hosts a vast number of public images and also supports private repositories.
- URL: hub.docker.com
- Use Case: Publicly sharing images and accessing a large collection of pre-built images from the community and official repositories.
- Private Registries:
- Description: Custom registries set up by organizations to securely store and manage their own Docker images.
- Use Case: Ensuring security and control over image distribution within an organization.
- Third-Party Registries:
- Examples: Amazon Elastic Container Registry (ECR), Google Container Registry (GCR), Azure Container Registry (ACR).
- Use Case: Integrating with cloud platforms for seamless deployment and management of images within cloud infrastructure.
-

- Benefits of Using Docker Registries
- Centralized Image Management: Registries provide a centralized location to store and manage Docker images, making it easier to organize and distribute them.
- Version Control: Using tags, registries allow version control of images, enabling users to easily roll back to previous versions if needed.
- Collaboration: Public registries like Docker Hub facilitate collaboration by allowing users to share images with the community or within teams.
- Security: Private registries ensure that sensitive images are stored securely and access is controlled within an organization.
- Integration with CI/CD: Registries integrate seamlessly with CI/CD pipelines, automating the process of building, storing, and deploying Docker images.
-

Docker Registry



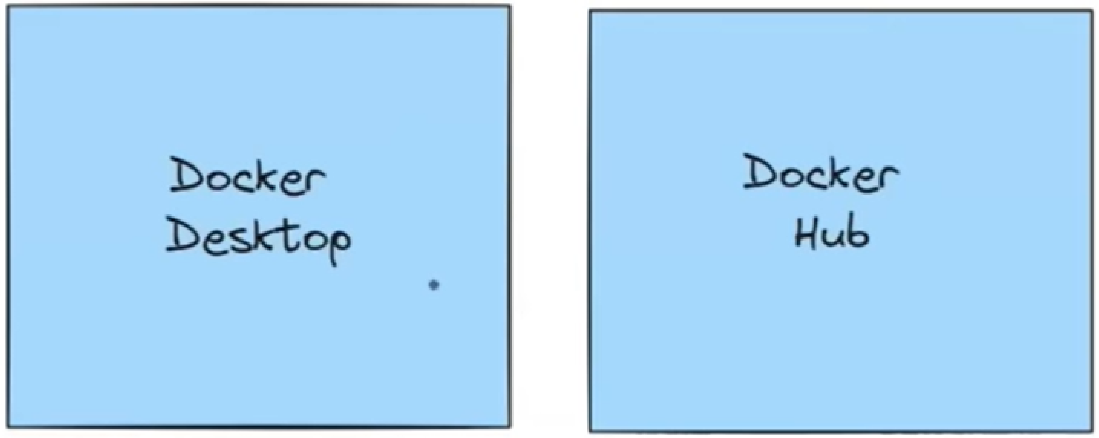
Use-cases

- Microservices Architecture
- Description: Microservices break down applications into smaller, independent services, each running in its own container.
- Benefits: Simplifies deployment, scaling, and maintenance. Each service can be developed, updated, and deployed independently.
- Continuous Integration and Continuous Deployment (CI/CD)
- Description: Docker ensures a consistent environment from development through testing to production.
- Benefits: Streamlines the CI/CD pipeline, reduces discrepancies between environments, and speeds up testing and deployment processes.

- Cloud Migration
- Description: Containerizing applications to move them to the cloud.
- Benefits: Simplifies the migration process, allows applications to run consistently across different cloud providers, and optimizes resource usage.
- Scalable Web Applications
- Description: Deploying web applications in containers for easy scaling.
- Benefits: Simplifies scaling up or down based on traffic, ensures consistent deployment, and enhances resource utilization.

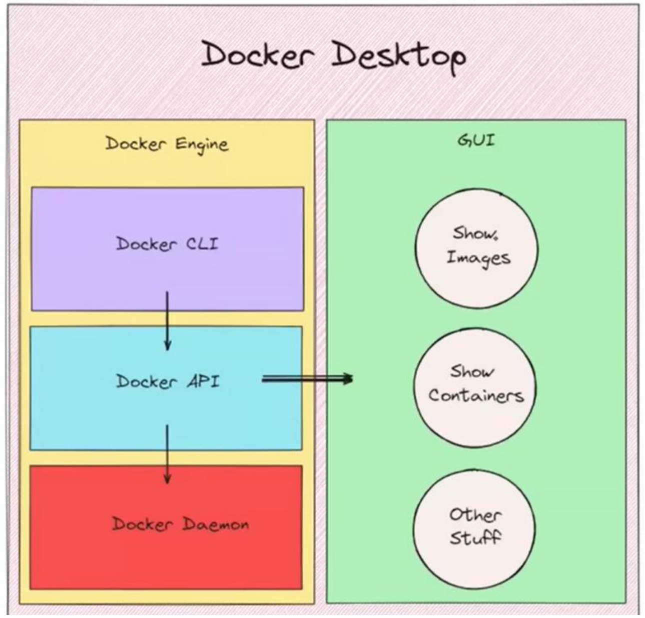
- Testing and QA
- Description: Creating consistent environments for testing applications.
- Benefits: Ensures tests are run in environments identical to production, speeds up the setup of test environments, and facilitates automated testing.
- Machine Learning and AI
- Description: Deploying machine learning models and AI applications in containers.
- Benefits: Ensures consistency in the runtime environment, simplifies scaling of model training and inference, and facilitates collaboration and reproducibility.

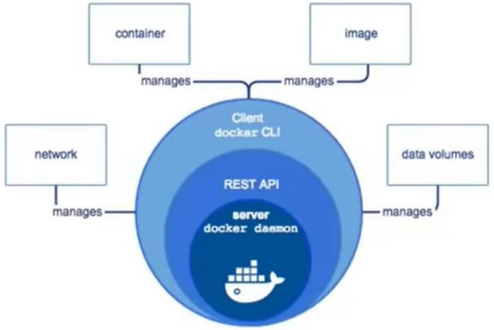
- API Development and Deployment
- Description: Developing and deploying APIs in containers.
- Benefits: Ensures APIs run consistently across environments, simplifies scaling, and improves deployment speed and reliability.

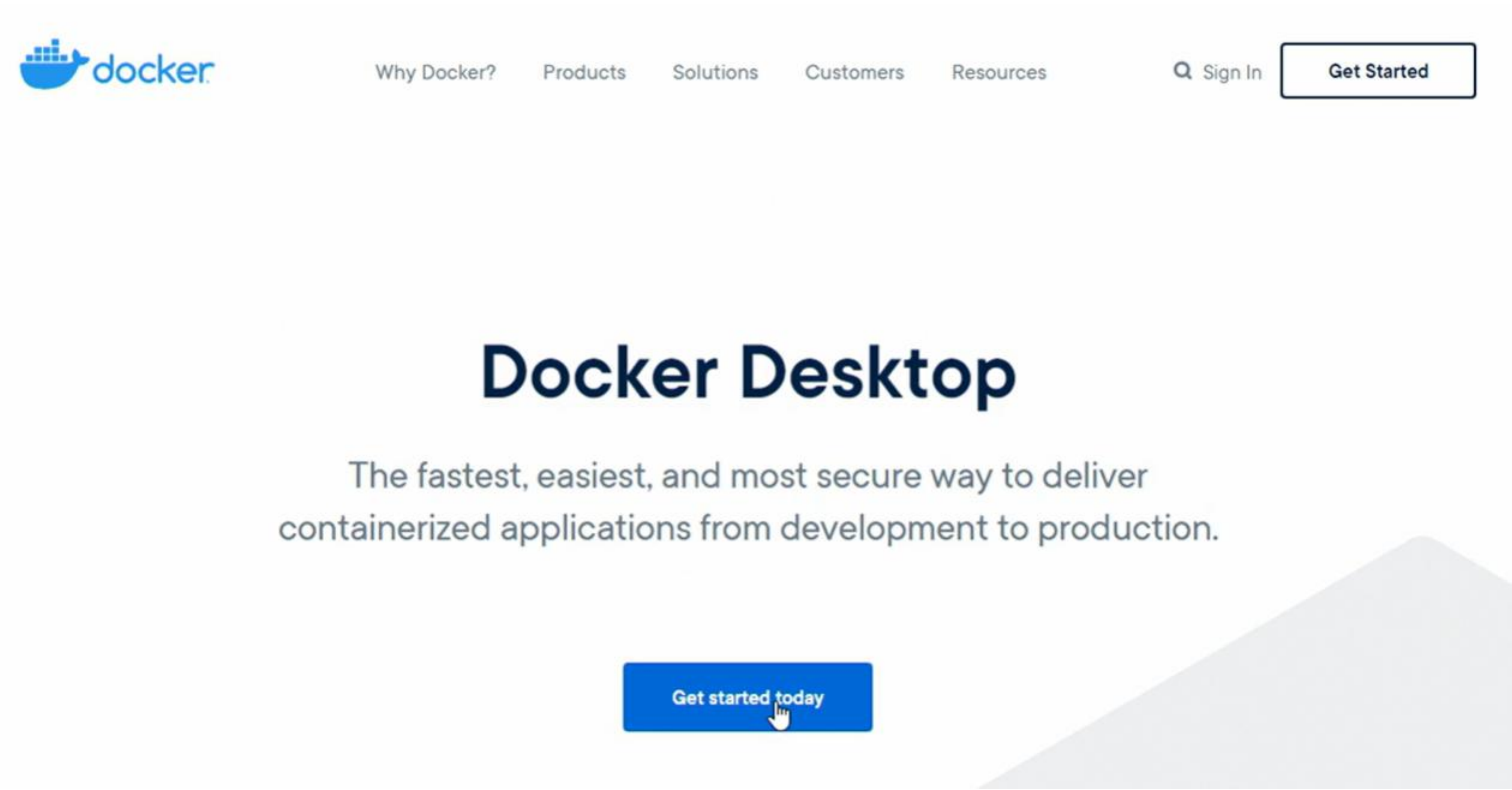


Docker Desktop

Docker Hub

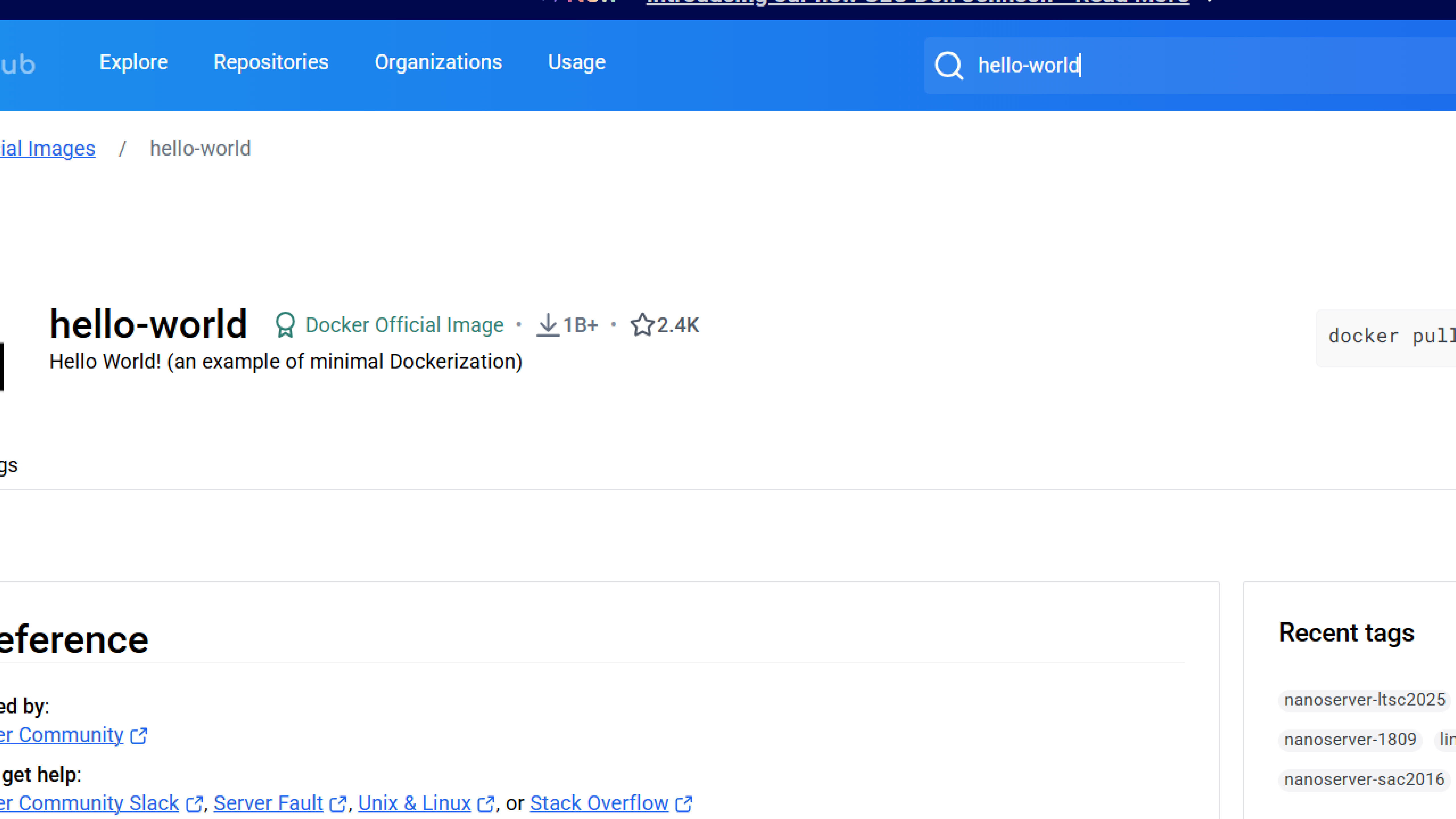

Docker Desktop
Docker Engine
Docker CLI
Docker API
Docker Daemon
GUI
Show Images
Show Containers
Other Stuff


container
image
manages
manages
network
data volumes
manages
manages
Client
docker CLI
REST API
server
docker daemon



Docker Desktop

The fastest, easiest, and most secure way to deliver containerized applications from development to production.



ial Images / hello-world

hello-world Docker Official Image • 1B+ • 2.4K

Hello World! (an example of minimal Dockerization)

```
docker pull
```

eference

ed by:

er Community

get help:

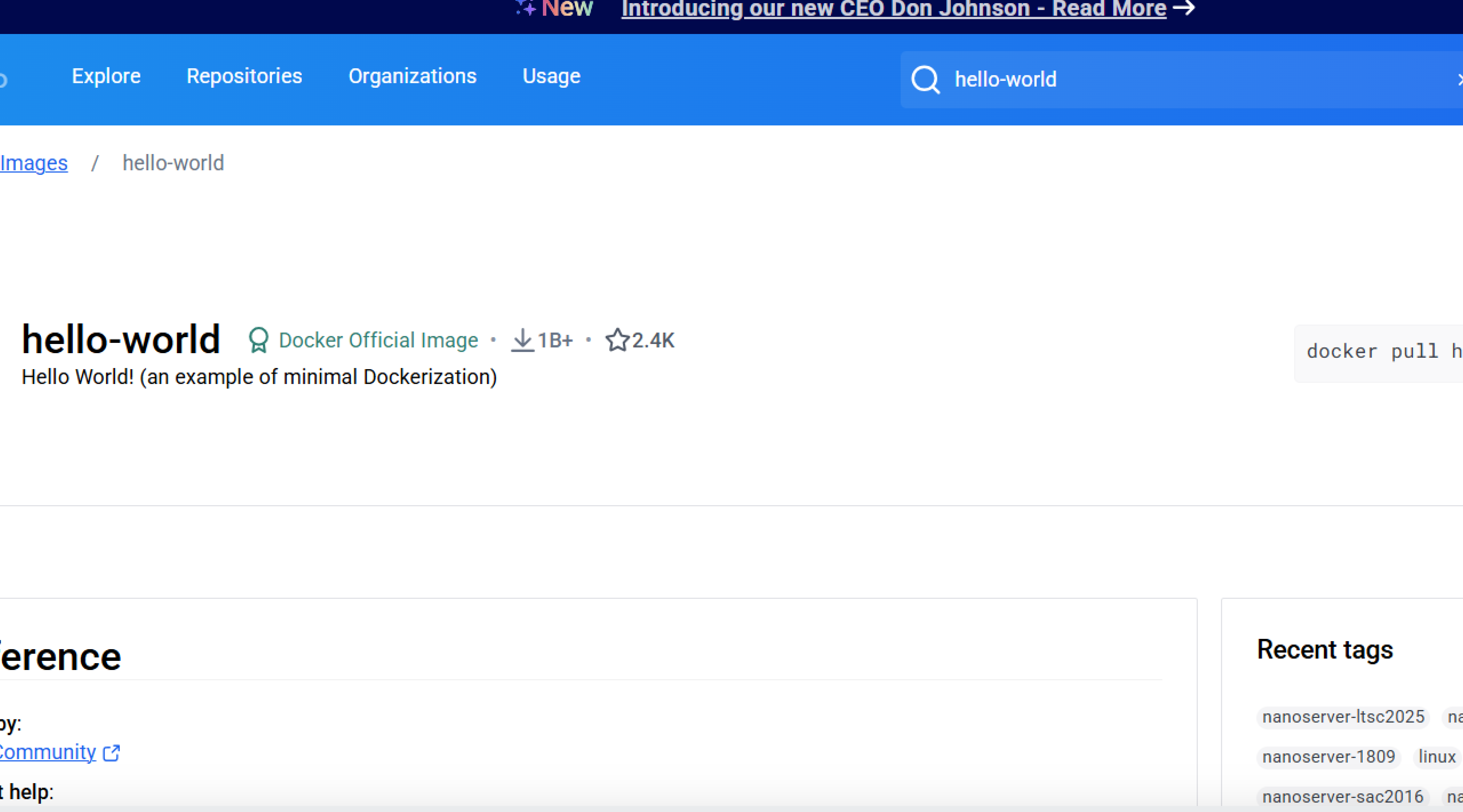
er Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Recent tags

nanoserver-ltsc2025

nanoserver-1809 lin

nanoserver-sac2016



hello-world

Docker Official Image • 1B+ • 2.4K

Hello World! (an example of minimal Dockerization)

```
docker pull h
```

erence

by:

Community

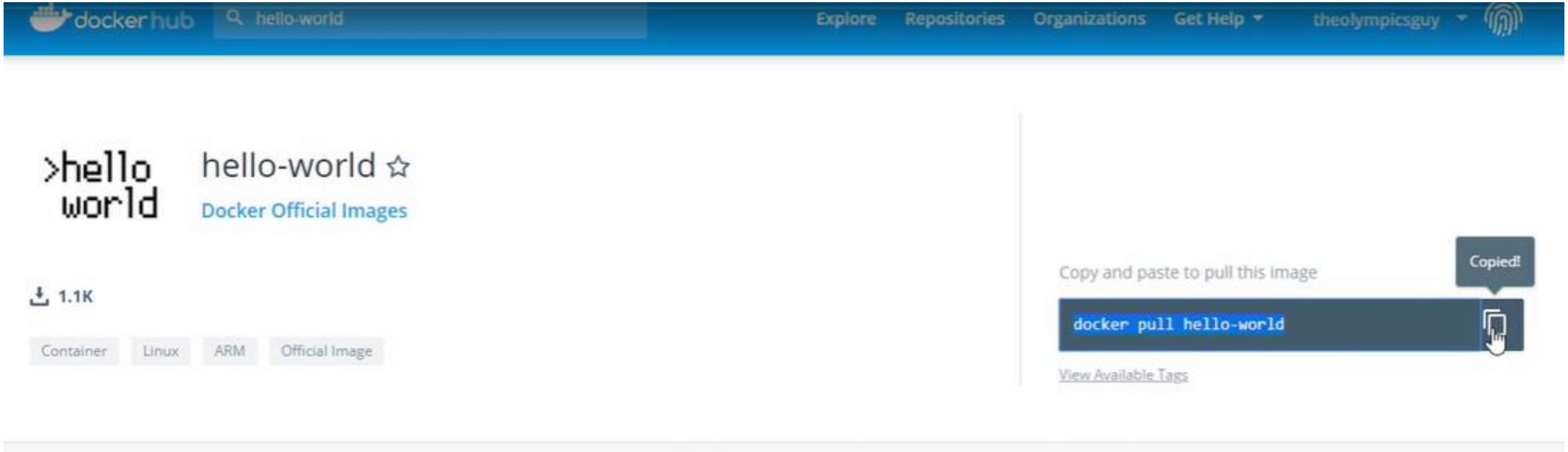
t help:

Recent tags

nanoserver-ltsc2025 na

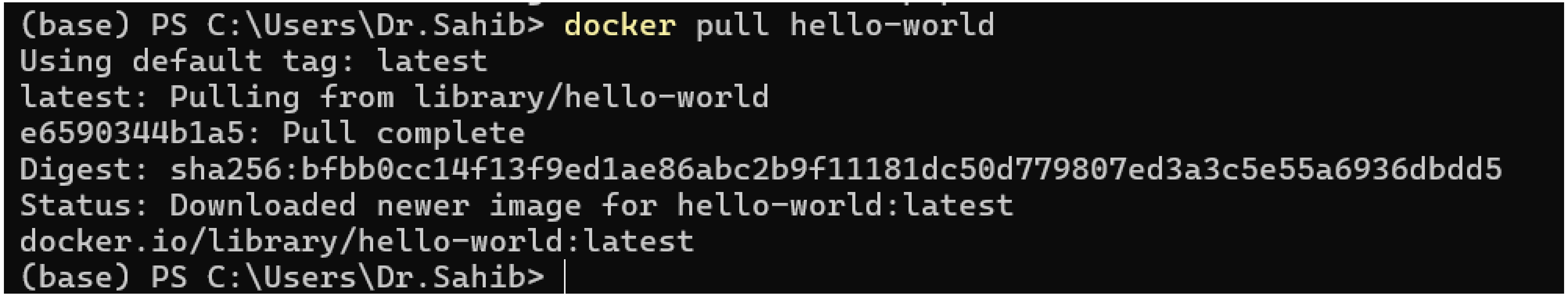
nanoserver-1809 linux

nanoserver-sac2016 na

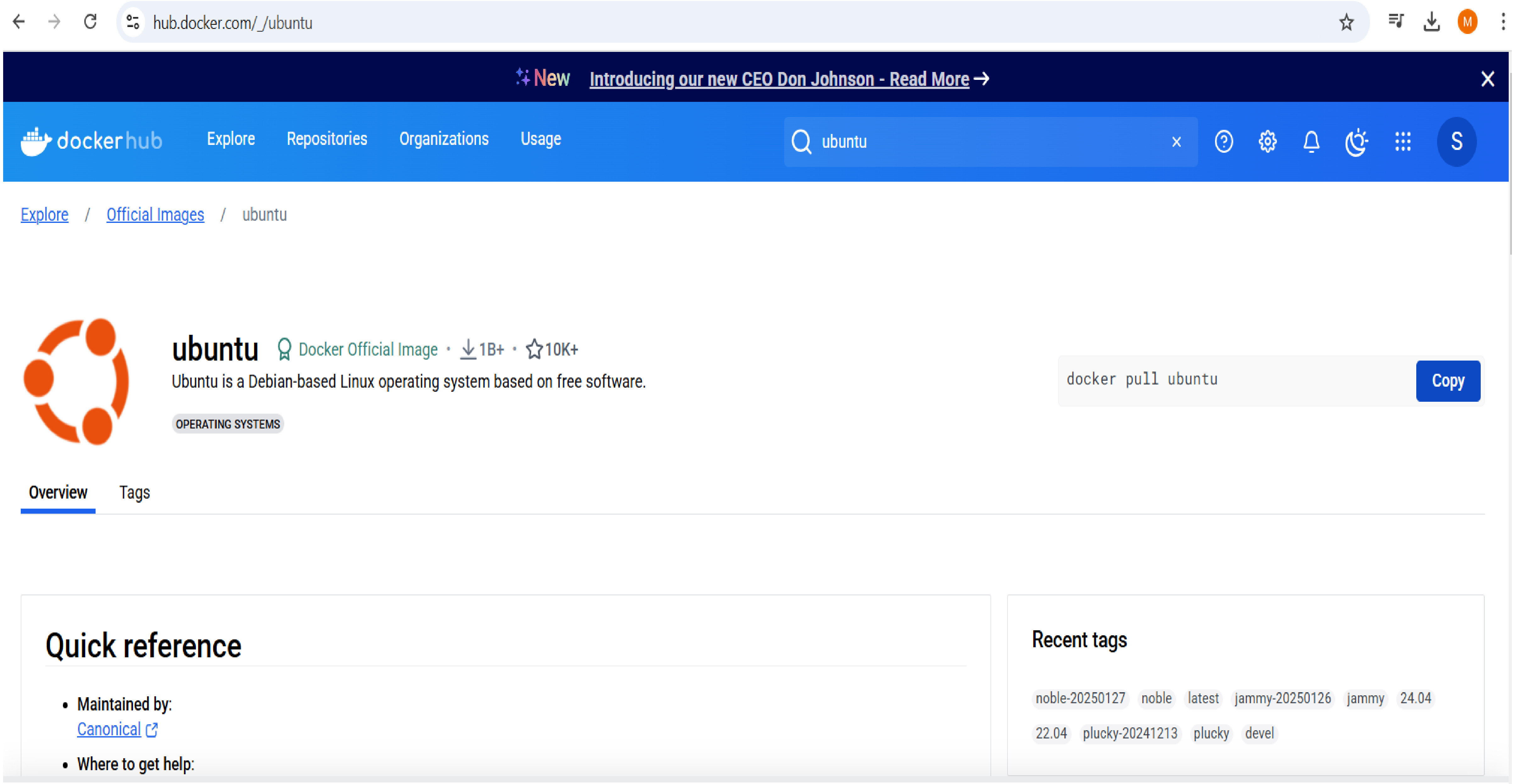


Copy and paste to pull this image

```
docker pull hello-world
```



```
(base) PS C:\Users\Dr.Sahib> docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:bfbb0cc14f13f9ed1ae86abc2b9f11181dc50d779807ed3a3c5e55a6936dbdd5
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
(base) PS C:\Users\Dr.Sahib>
```



ubuntu

Docker Official Image • 1B+ • 10K+

Ubuntu is a Debian-based Linux operating system based on free software.

OPERATING SYSTEMS

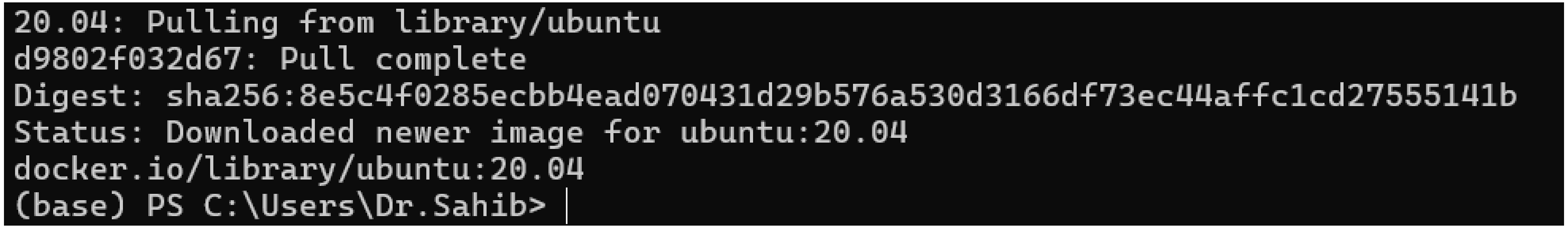
```
docker pull ubuntu
```

Quick reference

- **Maintained by:**
  Canonical
- **Where to get help:**

Recent tags

noble-20250127 noble latest jammy-20250126 jammy 24.04 22.04 plucky-20241213 plucky devel



```
20.04: Pulling from library/ubuntu
d9802f032d67: Pull complete
Digest: sha256:8e5c4f0285ecbb4ead070431d29b576a530d3166df73ec44affc1cd27555141b
Status: Downloaded newer image for ubuntu:20.04
docker.io/library/ubuntu:20.04
(base) PS C:\Users\Dr.Sahib>
```

Important Docker Image Commands

- `docker image ls -a`
- `docker image rm <image id>`
- `docker image rm $(docker image ls -a -q)`

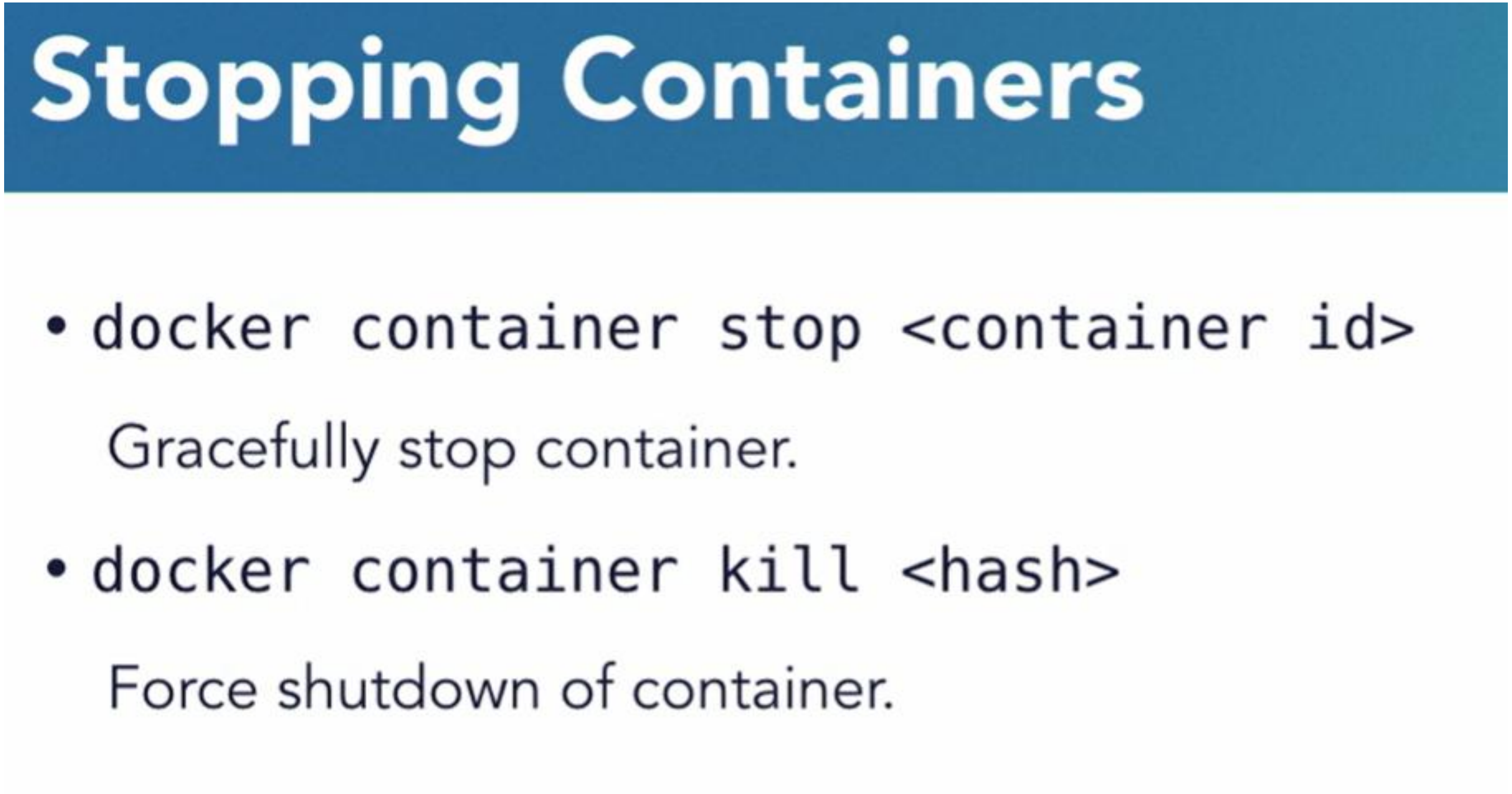
Listing Containers

- `docker container ls`

  List all running containers.

- `docker container ls -a`

  List all containers, even those not running.



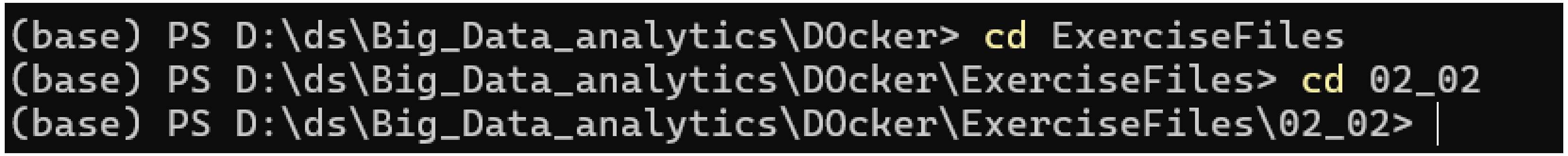
Stopping Containers

-
  ```
  docker container stop <container id>
  ```

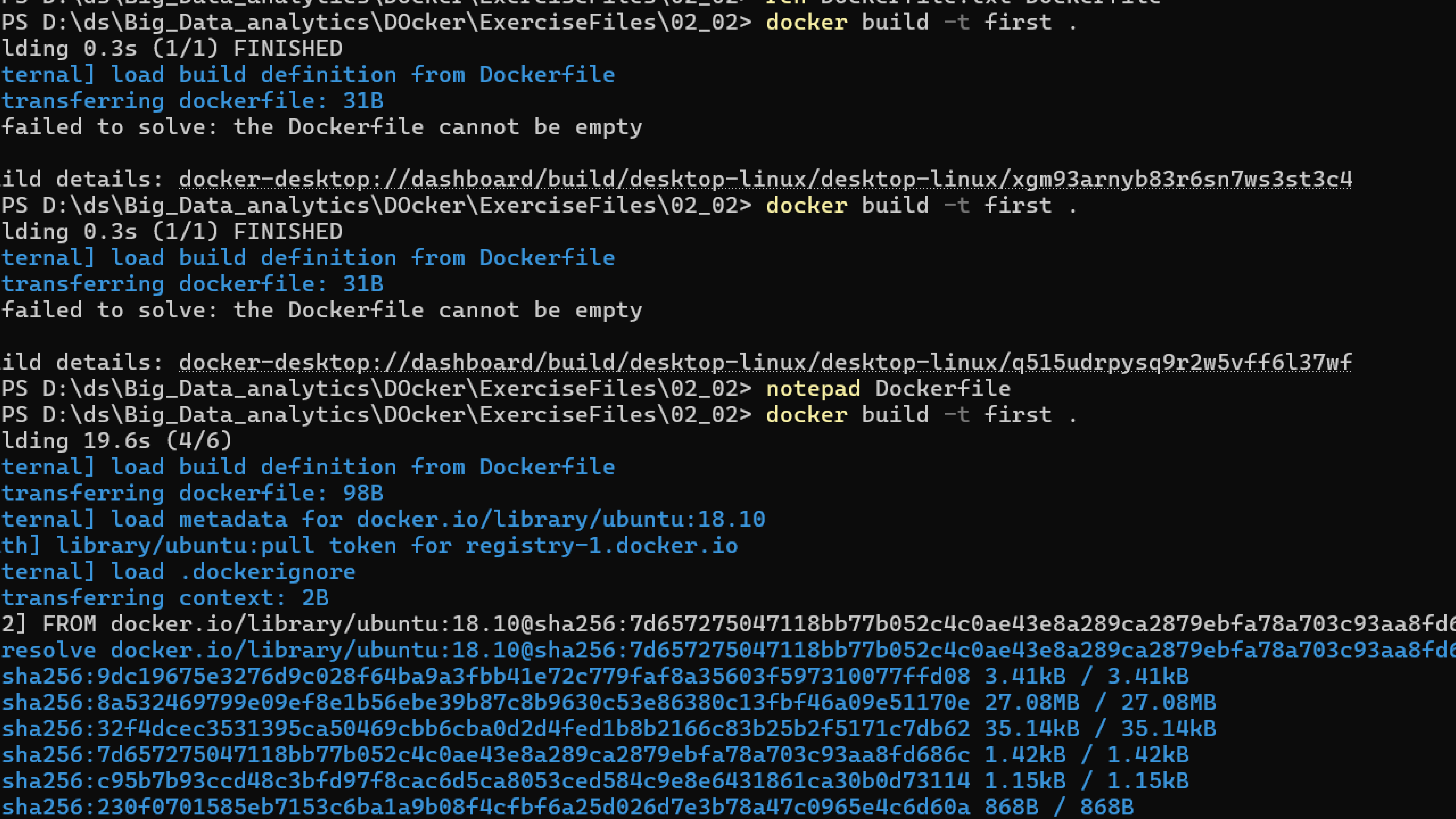
  Gracefully stop container.

-
  ```
  docker container kill <hash>
  ```

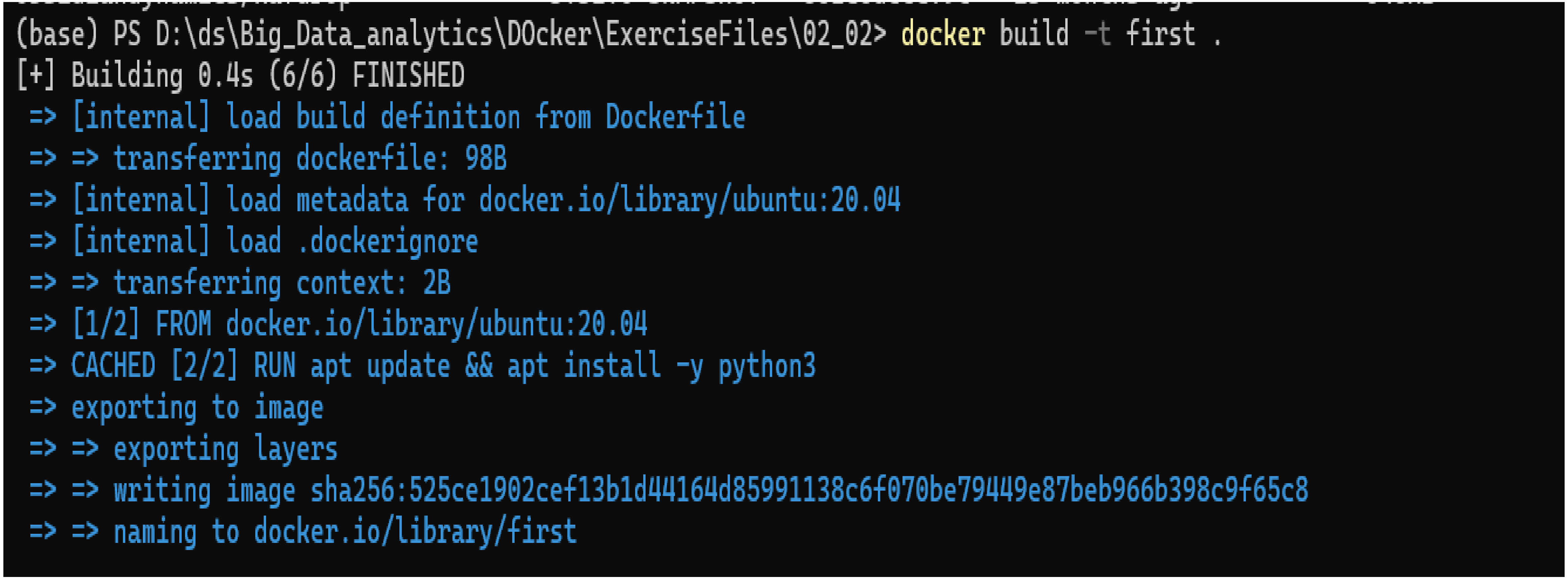
  Force shutdown of container.



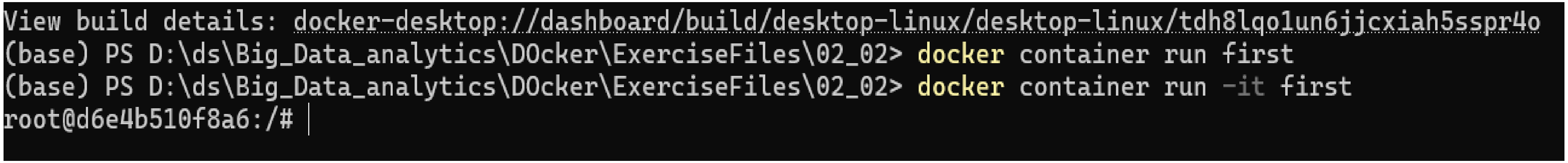
```
(base) PS D:\ds\Big_Data_analytics\DOcker> cd ExerciseFiles
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles> cd 02_02
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02>
```



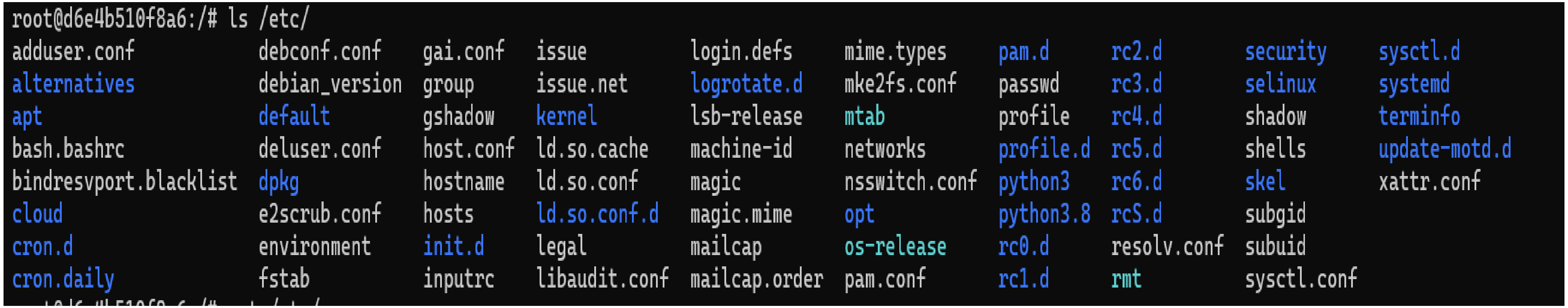
```
PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> ren Dockerfile.txt Dockerfile
PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker build -t first .
lding 0.3s (1/1) FINISHED
ternal] load build definition from Dockerfile
transferring dockerfile: 31B
failed to solve: the Dockerfile cannot be empty

ild details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/xgm93arnyb83r6sn7ws3st3c4
PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker build -t first .
lding 0.3s (1/1) FINISHED
ternal] load build definition from Dockerfile
transferring dockerfile: 31B
failed to solve: the Dockerfile cannot be empty

ild details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/q515udrpysq9r2w5vff6l37wf
PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> notepad Dockerfile
PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker build -t first .
lding 19.6s (4/6)
ternal] load build definition from Dockerfile
transferring dockerfile: 98B
ternal] load metadata for docker.io/library/ubuntu:18.10
th] library/ubuntu:pull token for registry-1.docker.io
ternal] load .dockerignore
transferring context: 2B
2] FROM docker.io/library/ubuntu:18.10@sha256:7d657275047118bb77b052c4c0ae43e8a289ca2879ebfa78a703c93aa8fd6
resolve docker.io/library/ubuntu:18.10@sha256:7d657275047118bb77b052c4c0ae43e8a289ca2879ebfa78a703c93aa8fd6
sha256:9dc19675e3276d9c028f64ba9a3fbb41e72c779faf8a35603f597310077ffd08 3.41kB / 3.41kB
sha256:8a532469799e09ef8e1b56ebe39b87c8b9630c53e86380c13fbf46a09e51170e 27.08MB / 27.08MB
sha256:32f4dcec3531395ca50469cbb6cba0d2d4fed1b8b2166c83b25b2f5171c7db62 35.14kB / 35.14kB
sha256:7d657275047118bb77b052c4c0ae43e8a289ca2879ebfa78a703c93aa8fd686c 1.42kB / 1.42kB
sha256:c95b7b93ccd48c3bfd97f8cac6d5ca8053ced584c9e8e6431861ca30b0d73114 1.15kB / 1.15kB
sha256:230f0701585eb7153c6ba1a9b08f4cfbf6a25d026d7e3b78a47c0965e4c6d60a 868B / 868B
```



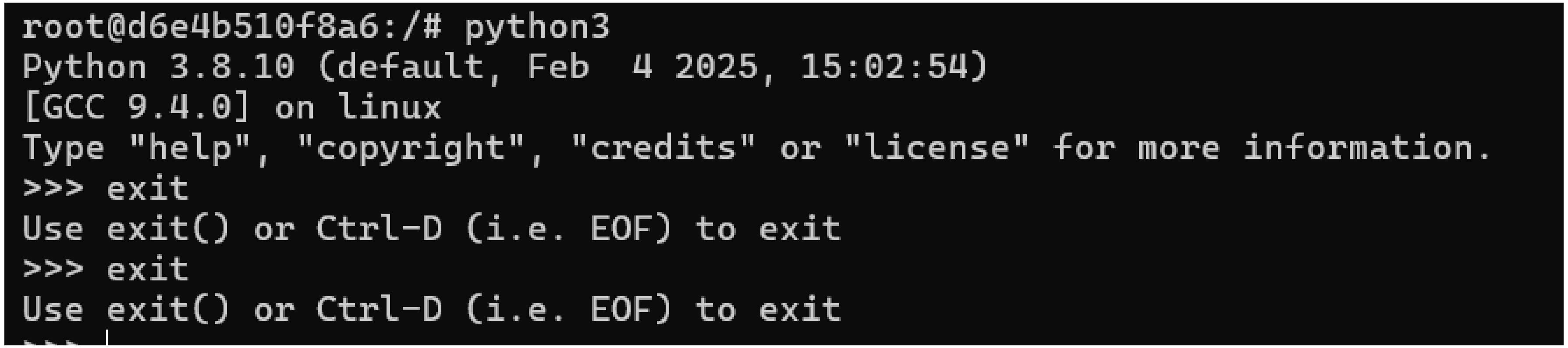
```
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker build -t first .
[+] Building 0.4s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 98B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/ubuntu:20.04
 => CACHED [2/2] RUN apt update && apt install -y python3
 => exporting to image
 => => exporting layers
 => => writing image sha256:525ce1902cef13b1d44164d85991138c6f070be79449e87beb966b398c9f65c8
 => => naming to docker.io/library/first
```



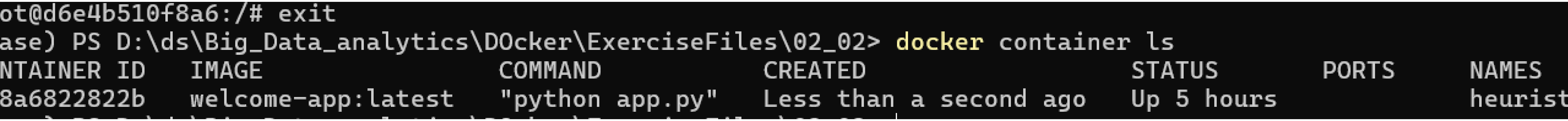
```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/tdh8lqo1un6jjcxiah5sspr4o
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker container run first
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker container run -it first
root@d6e4b510f8a6:/#
```



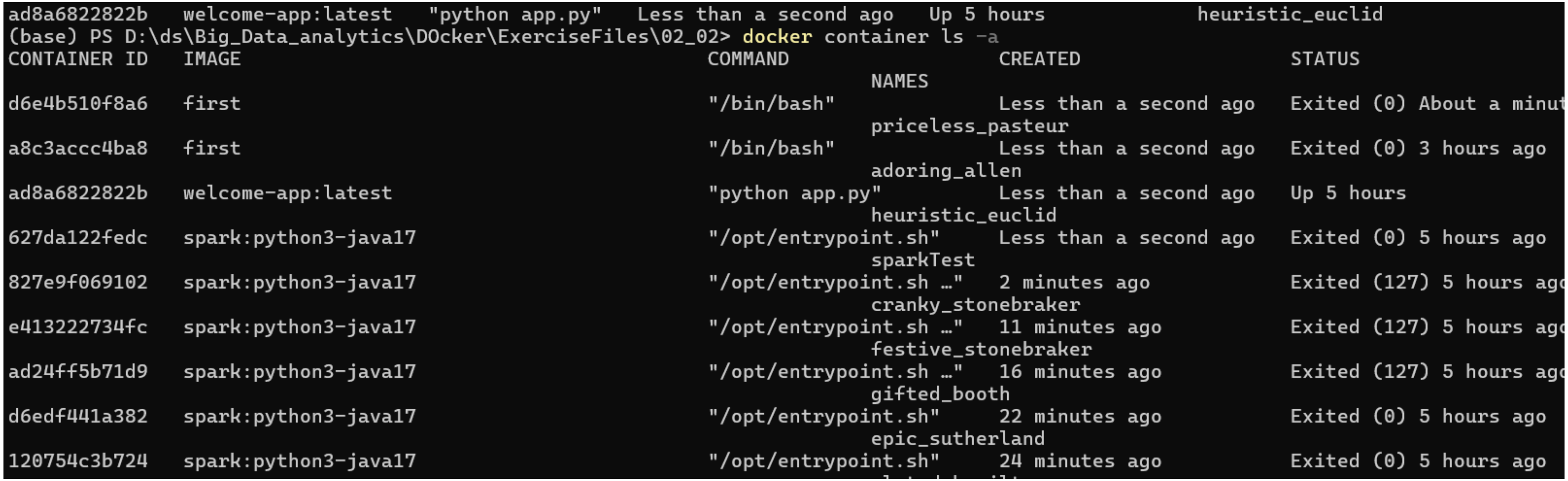
```
root@d6e4b510f8a6:/# ls /etc/
adduser.conf            debconf.conf    gai.conf   issue          login.defs     mime.types     pam.d       rc2.d        security   sysctl.d
alternatives            debian_version  group      issue.net      logrotate.d    mke2fs.conf    passwd      rc3.d        selinux    systemd
apt                     default         gshadow    kernel         lsb-release    mtab           profile     rc4.d        shadow     terminfo
bash.bashrc             deluser.conf    host.conf  ld.so.cache    machine-id     networks       profile.d   rc5.d        shells     update-motd.d
bindresvport.blacklist  dpkg            hostname   ld.so.conf     magic          nsswitch.conf  python3     rc6.d        skel       xattr.conf
cloud                   e2scrub.conf    hosts      ld.so.conf.d   magic.mime     opt            python3.8   rcS.d        subgid
cron.d                  environment     init.d     legal          mailcap        os-release     rc0.d       resolv.conf  subuid
cron.daily              fstab           inputrc    libaudit.conf  mailcap.order  pam.conf       rc1.d       rmt          sysctl.conf
```



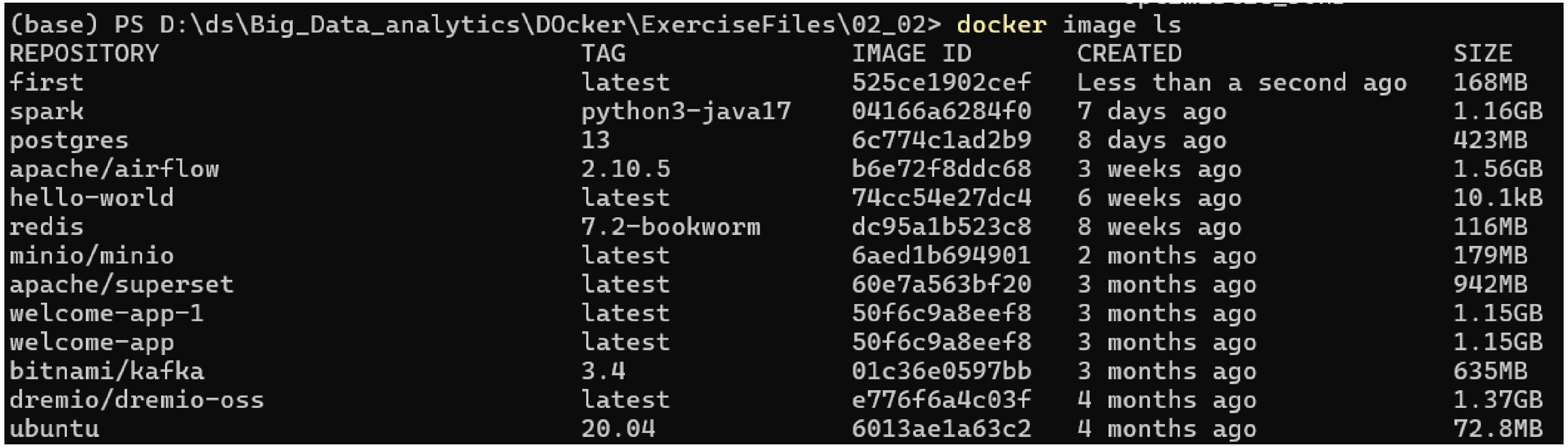
```
root@d6e4b510f8a6:/# python3
Python 3.8.10 (default, Feb  4 2025, 15:02:54)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
```



```
ot@d6e4b510f8a6:/# exit
ase) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker container ls
NTAINER ID   IMAGE                COMMAND           CREATED                  STATUS       PORTS     NAMES
8a6822822b   welcome-app:latest   "python app.py"   Less than a second ago   Up 5 hours             heurist
```

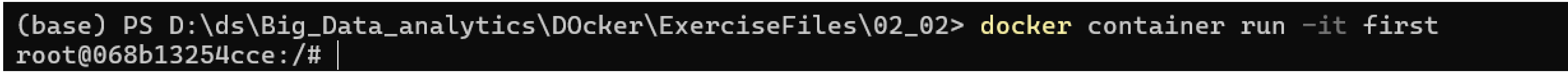


```
ad8a6822822b   welcome-app:latest   "python app.py"   Less than a second ago   Up 5 hours            heuristic_euclid
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker container ls -a
CONTAINER ID   IMAGE                                           COMMAND                  CREATED                  STATUS
                                                                           NAMES
d6e4b510f8a6   first                                           "/bin/bash"              Less than a second ago   Exited (0) About a minut
                                                                           priceless_pasteur
a8c3accc4ba8   first                                           "/bin/bash"              Less than a second ago   Exited (0) 3 hours ago
                                                                           adoring_allen
ad8a6822822b   welcome-app:latest                              "python app.py"          Less than a second ago   Up 5 hours
                                                                           heuristic_euclid
627da122fedc   spark:python3-java17                            "/opt/entrypoint.sh"     Less than a second ago   Exited (0) 5 hours ago
                                                                           sparkTest
827e9f069102   spark:python3-java17                            "/opt/entrypoint.sh …"   2 minutes ago            Exited (127) 5 hours ago
                                                                           cranky_stonebraker
e413222734fc   spark:python3-java17                            "/opt/entrypoint.sh …"   11 minutes ago           Exited (127) 5 hours ago
                                                                           festive_stonebraker
ad24ff5b71d9   spark:python3-java17                            "/opt/entrypoint.sh …"   16 minutes ago           Exited (127) 5 hours ago
                                                                           gifted_booth
d6edf441a382   spark:python3-java17                            "/opt/entrypoint.sh"     22 minutes ago           Exited (0) 5 hours ago
                                                                           epic_sutherland
120754c3b724   spark:python3-java17                            "/opt/entrypoint.sh"     24 minutes ago           Exited (0) 5 hours ago
```

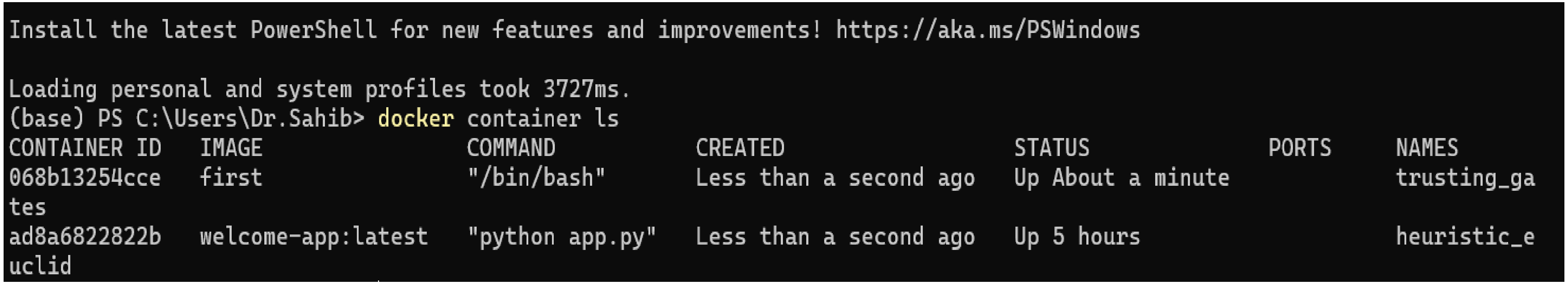


```
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker image ls
REPOSITORY                     TAG                IMAGE ID       CREATED                  SIZE
first                          latest             525ce1902cef   Less than a second ago   168MB
spark                          python3-java17     04166a6284f0   7 days ago               1.16GB
postgres                       13                 6c774c1ad2b9   8 days ago               423MB
apache/airflow                 2.10.5             b6e72f8ddc68   3 weeks ago              1.56GB
hello-world                    latest             74cc54e27dc4   6 weeks ago              10.1kB
redis                          7.2-bookworm       dc95a1b523c8   8 weeks ago              116MB
minio/minio                    latest             6aed1b694901   2 months ago             179MB
apache/superset                latest             60e7a563bf20   3 months ago             942MB
welcome-app-1                  latest             50f6c9a8eef8   3 months ago             1.15GB
welcome-app                    latest             50f6c9a8eef8   3 months ago             1.15GB
bitnami/kafka                  3.4                01c36e0597bb   3 months ago             635MB
dremio/dremio-oss              latest             e776f6a4c03f   4 months ago             1.37GB
ubuntu                         20.04              6013ae1a63c2   4 months ago             72.8MB
```

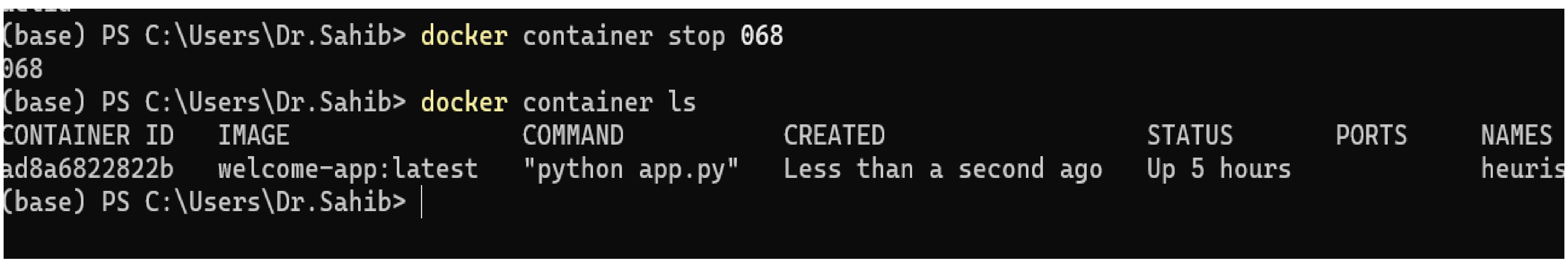
Open two terminal



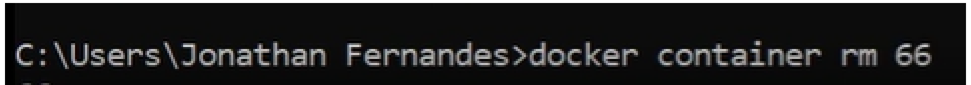
```
(base) PS D:\ds\Big_Data_analytics\DOcker\ExerciseFiles\02_02> docker container run -it first
root@068b13254cce:/# 
```



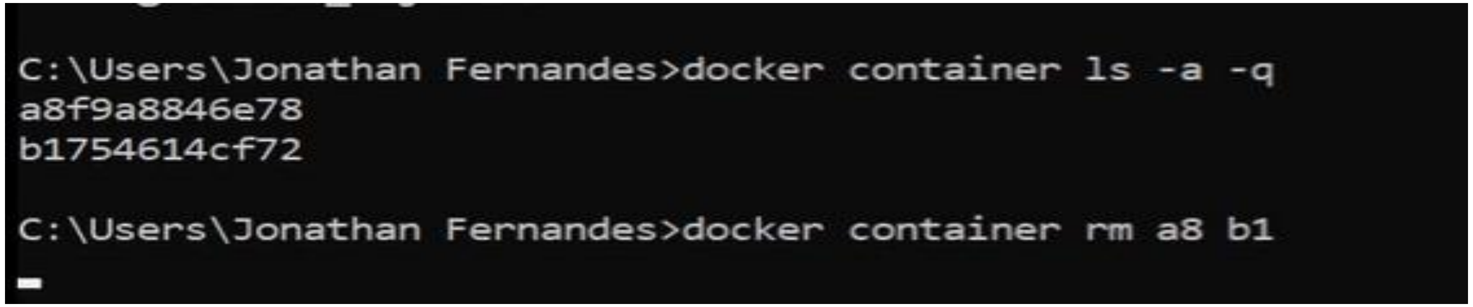
```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

Loading personal and system profiles took 3727ms.
(base) PS C:\Users\Dr.Sahib> docker container ls
CONTAINER ID   IMAGE                 COMMAND           CREATED                  STATUS               PORTS     NAMES
068b13254cce   first                 "/bin/bash"       Less than a second ago   Up About a minute              trusting_ga
tes
ad8a6822822b   welcome-app:latest    "python app.py"   Less than a second ago   Up 5 hours                     heuristic_e
uclid
```



```
(base) PS C:\Users\Dr.Sahib> docker container stop 068
068
(base) PS C:\Users\Dr.Sahib> docker container ls
CONTAINER ID   IMAGE                 COMMAND           CREATED                  STATUS       PORTS     NAMES
ad8a6822822b   welcome-app:latest    "python app.py"   Less than a second ago   Up 5 hours             heuris
(base) PS C:\Users\Dr.Sahib> 
```



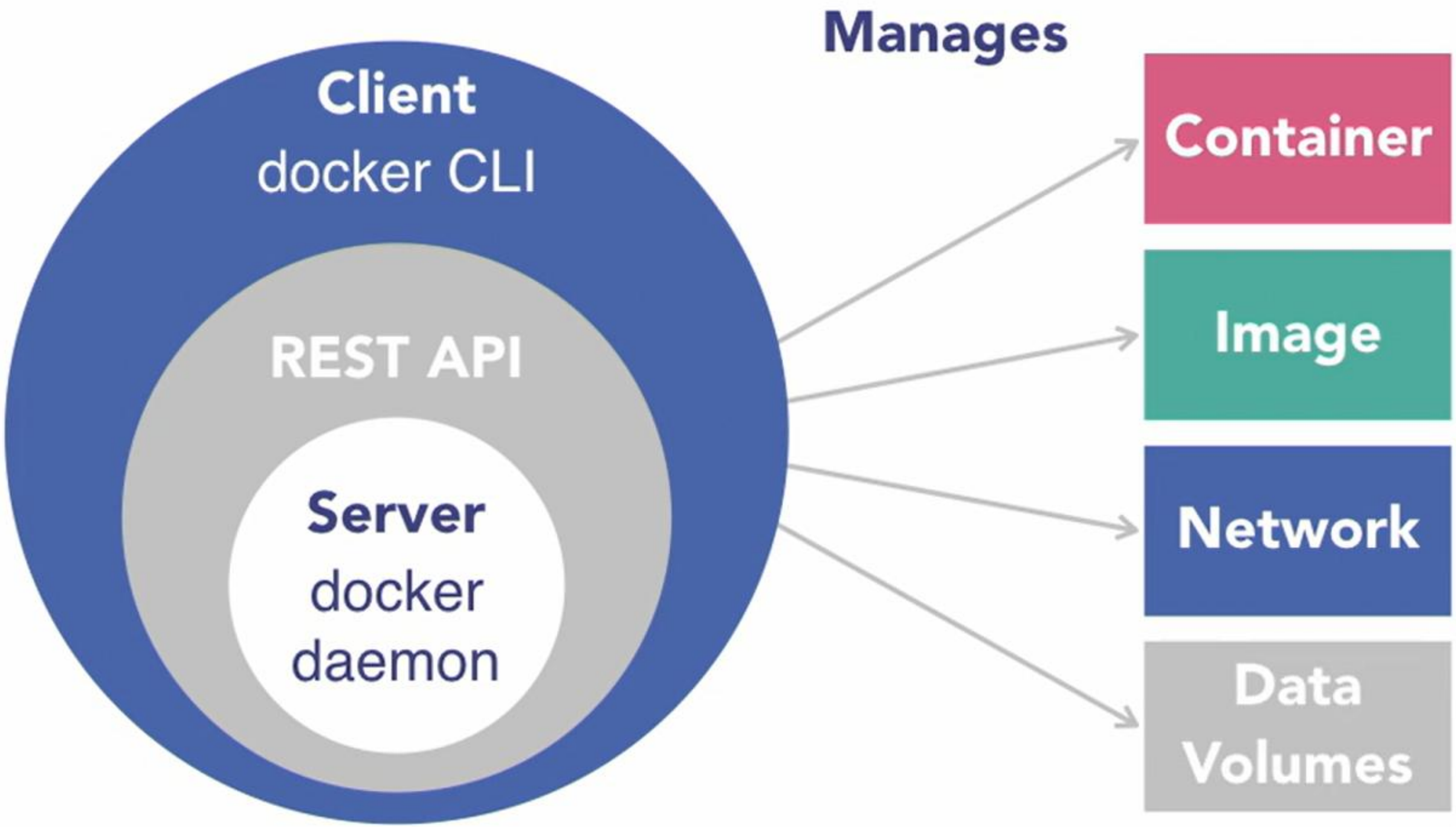
```
C:\Users\Jonathan Fernandes>docker container rm 66
```



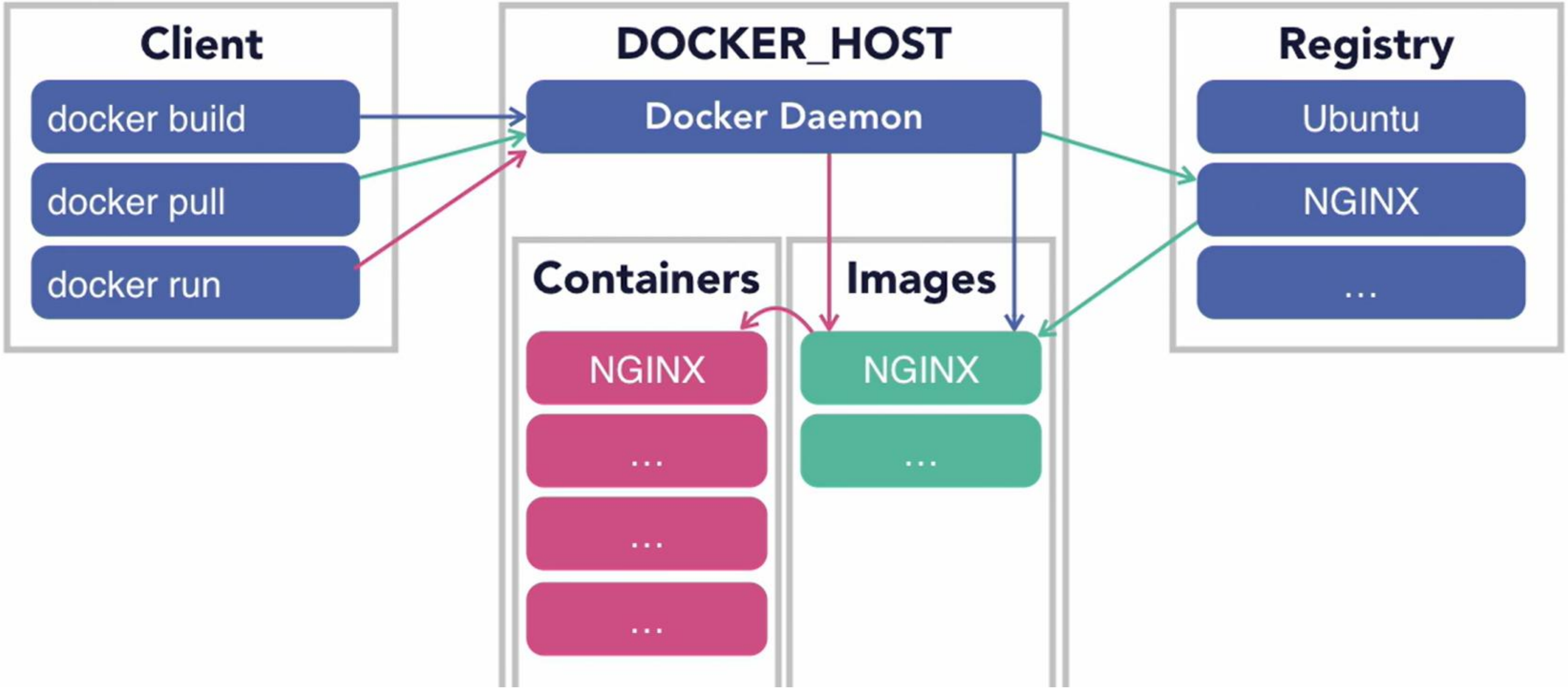
```
C:\Users\Jonathan Fernandes>docker container ls -a -q
a8f9a8846e78
b1754614cf72

C:\Users\Jonathan Fernandes>docker container rm a8 b1
```

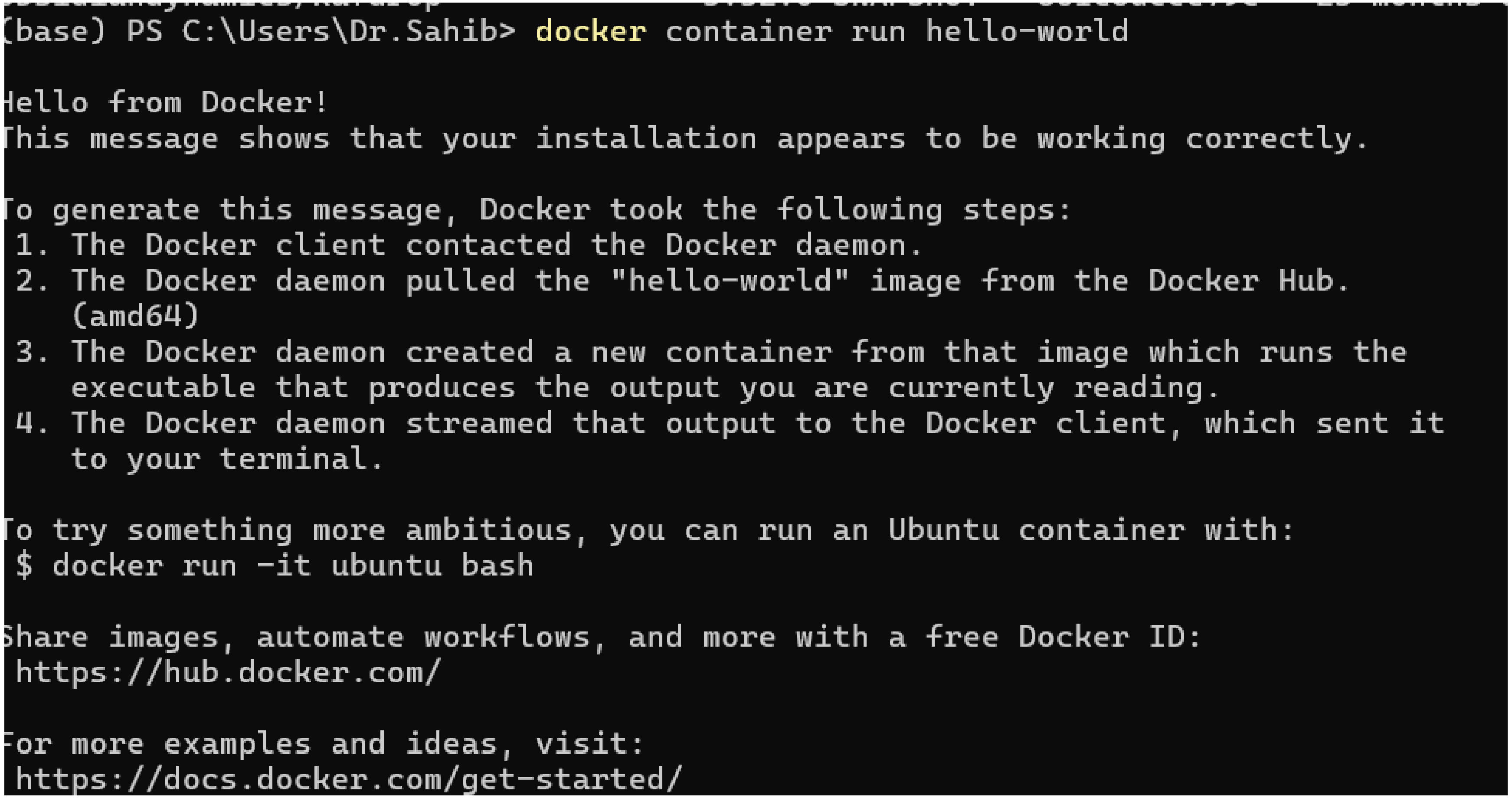
- Docker Daemon: The Docker daemon runs on your host operating system as a background process, managing Docker objects like images, containers, networks, and volumes.
- Client-Server Architecture: Docker uses a client-server architecture where the Docker CLI (client) communicates with the Docker daemon (server) via network sockets.
- Command Execution: Commands like docker build, docker pull, and docker run are processed by the Docker daemon, which performs tasks such as building images, pulling images from Docker Hub, and running containers.

Build, Pull, and Run



Build, Pull, and Run





```
(base) PS C:\Users\Dr.Sahib> docker container run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```