Operating System Virtualization
UNIT-V-Part-II

Experiments

Docker Container Host vs. Container OS for Linux and Windows Containers

http://www.floydhilton.com/docker/2017/03/31/Docker-ContainerHost-vs-ContainerOS-Linux-Windows.html

Container Host (non-windows)

- Also called the Host OS.
- The Host OS is one on which the Docker client and Docker daemon run.
- In the case of **Linux** and **non-Hyper-V** containers, the Host OS shares its kernel with running Docker containers.

Container Host

- Linux Containers on Windows (via Docker Desktop) share WSL 2's Linux kernel or the Linux VM running under Hyper-V.
- *All the containers share only one Linux kernel rather than each container having its own kernel.*

Container OS

- Also called the Base OS.
- An image that contains an OS such as Ubuntu, CentOS, or *windowsservercore*.
- We would build our own image on top of a Base OS image so that we can utilize parts of the OS.
- Windows containers require a Base OS (such as *windowsservercore* or *nanoserver*).

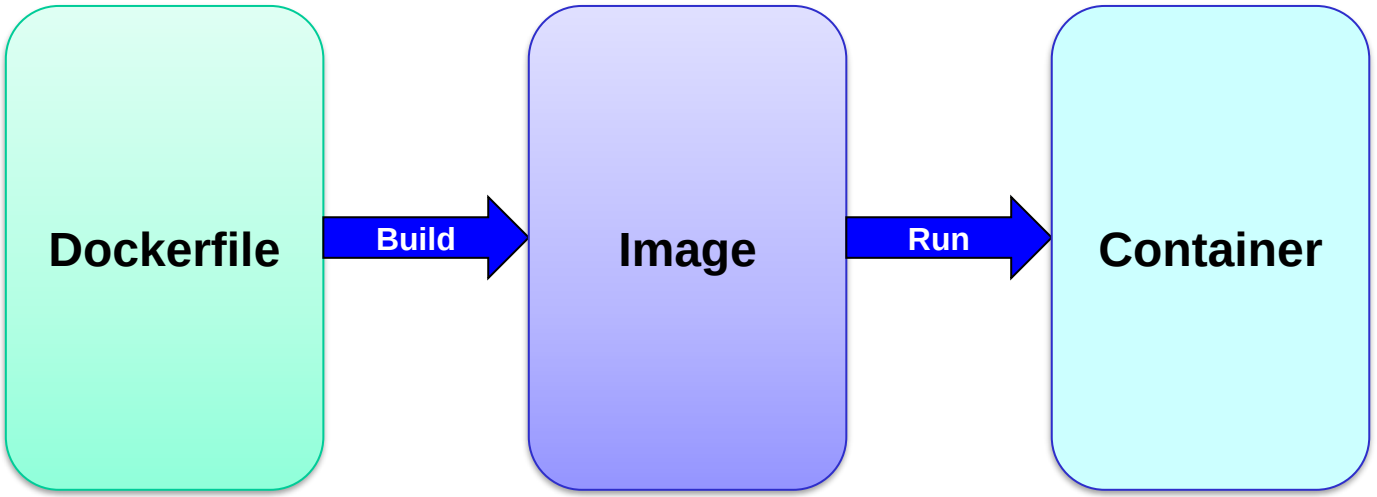
Container OS

- Linux container images typically use a base image (like Ubuntu, CentOS, or Alpine) to provide needed libraries and tools.
- *We can build a Linux container "from scratch" (using scratch base image).*
- The smallest Windows container base image is Nano Server, typically around 100–300 MB, which is significantly larger than minimal Linux images like Alpine (~5–10 MB) due to the need for Windows OS libraries and APIs.

OS Kernel

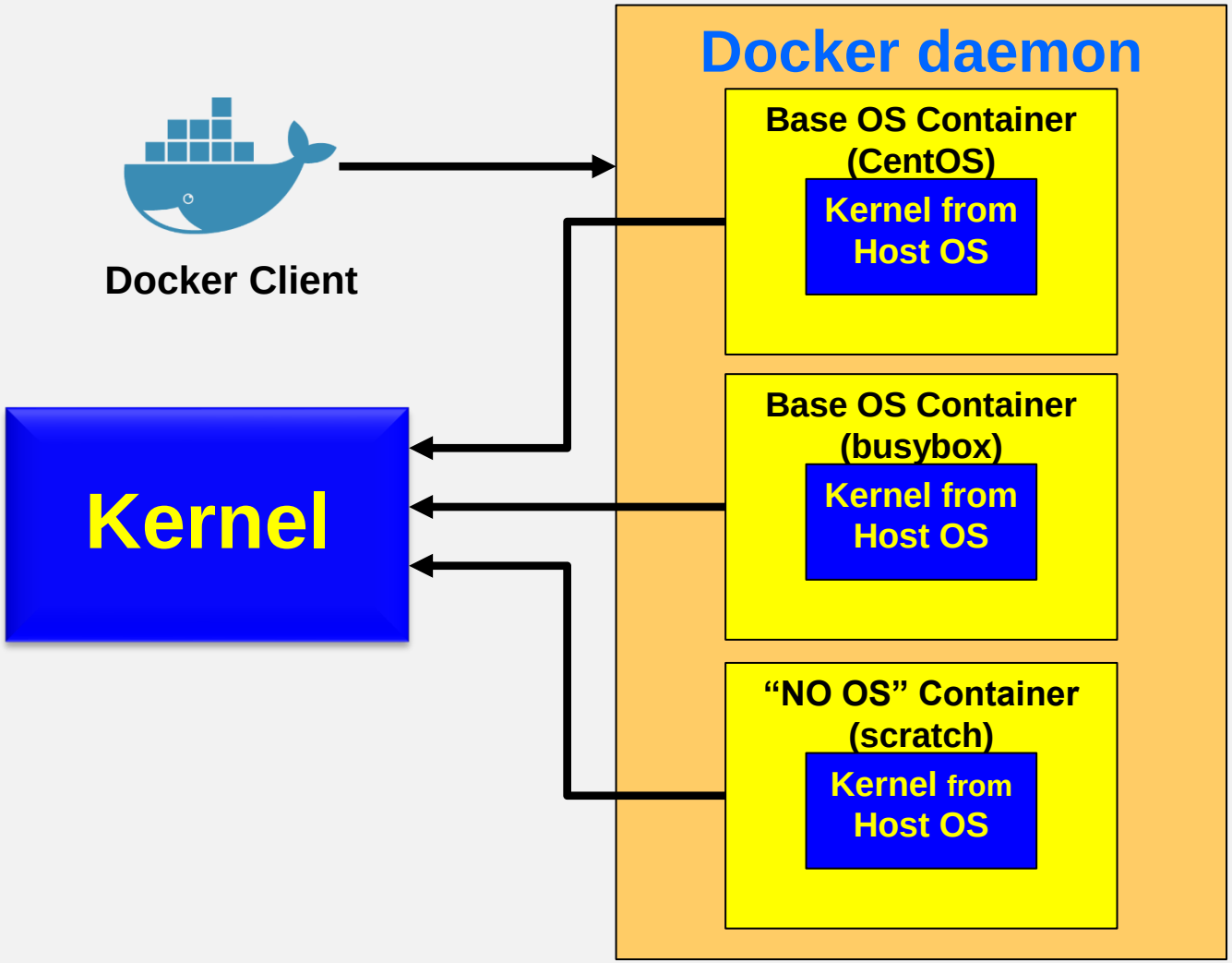
- The Kernel manages lower level functions such as:

  ✓ memory management,
  ✓ file system,
  ✓ network and process scheduling..


Dockerfile
Build
Image
Run
Container

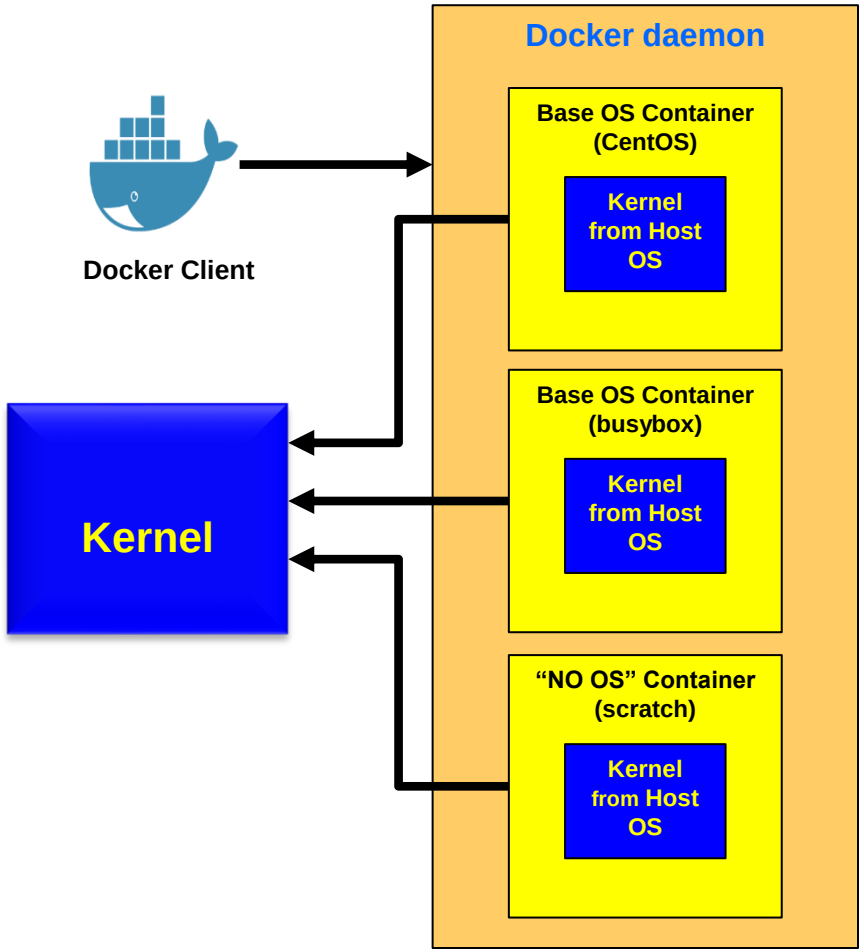
Linux Containers

Host OS Ubuntu



Host OS Ubuntu

- The Docker Client and the Docker Daemon (together called the Docker Engine) are running on the Host OS.
- Each container shares the Host OS kernel.
- CentOS is Linux Base OS images, and BusyBox is Linux utility.
- The “No OS” container demonstrates that we do not NEED a base OS to run a container in Linux.

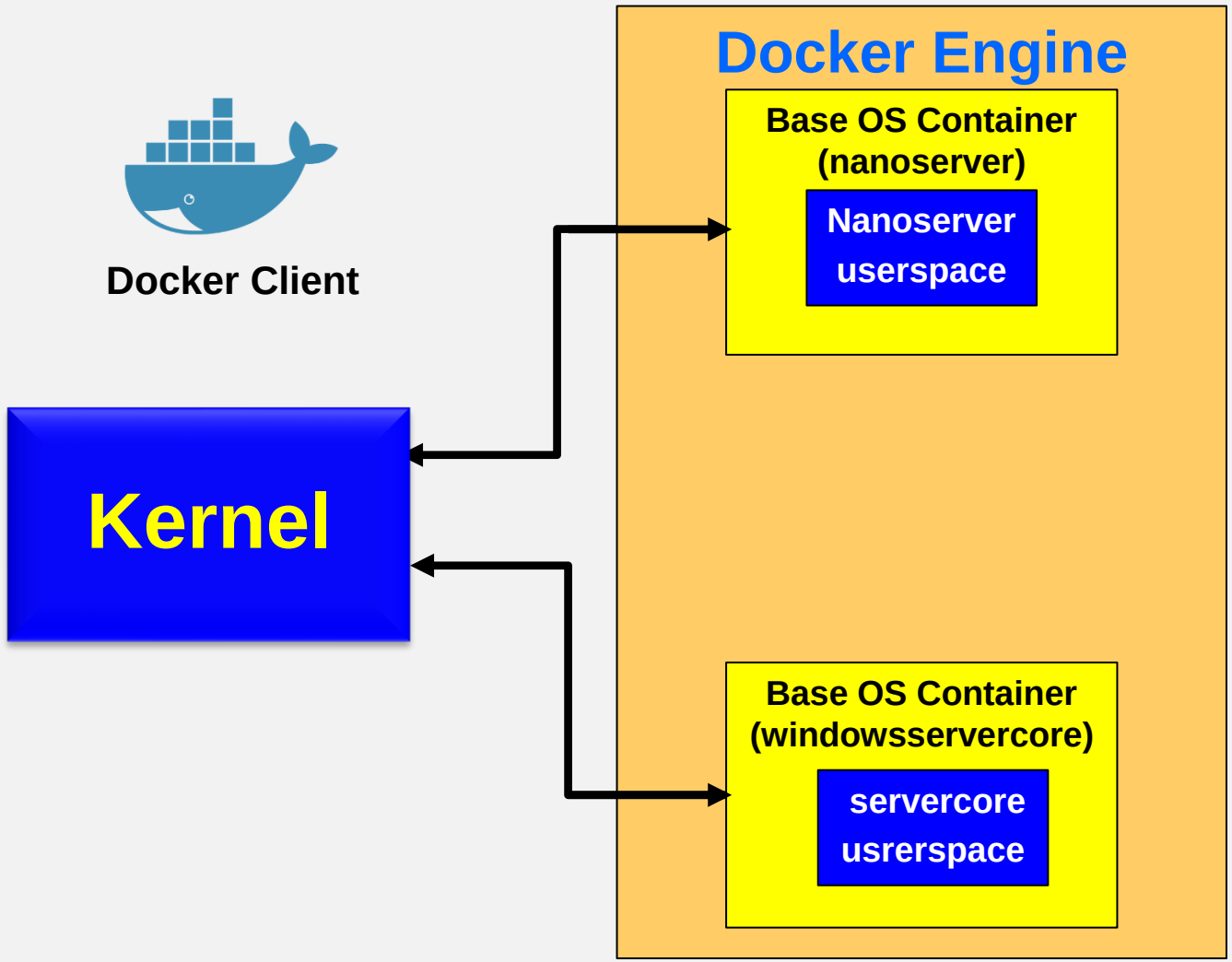


Host OS Ubuntu

- We can create a Docker file that has a base image of scratch (an explicitly empty image, especially for building images “FROM scratch”) and then runs a binary that uses the kernel directly.
- ***Windows containers cannot run without a base OS image*** because they depend on Windows kernel APIs and system libraries.
- Unlike Linux containers, there is no ‘scratch’ equivalent in Windows.

Windows server Containers
Non Hyper-V/WSL 2

Host OS Windows server or Windows 10



Host OS Windows server or Windows 10

- **Shared Kernel:**
  All containers use the **Host OS kernel** so no separate kernel per container
- **Base OS = User Space:**
  Images like **Nano Server** and **Server Core** provide only **user-space components**
- **Example:**
  - nanoserver has minimal user space
  - servercore has larger user space

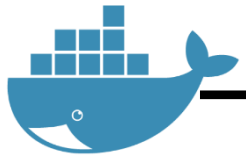
- *Kernel is NOT inside the container*

What is User Space?

- User space includes:
  - Applications (Python, nginx, bash)
  - Libraries (glibc, .NET, etc.)
  - System tools (ls, ps, apt)
- It cannot directly access hardware
- It talks to kernel via system calls

Windows Hyper-V/WSL 2 Containers

Host OS Windows server or Windows 11



Docker Client

Host OS / Hypervisor

Hyper-V VM (lightweight VM)

Extra Isolation with VM

Kernel

Base OS Container (nanoserver)

Nanoserver userspace

Hyper-V VM (lightweight VM)

Extra Isolation with VM

Kernel

Base OS Container (windowsservercore)

Servercore userspace

In Hyper-V isolation, each container runs inside a lightweight virtual machine with its own kernel. The host OS provides the hypervisor layer, and each VM provides strong isolation compared to process-based containers.

Host OS Windows server or Windows 11

- The Host OS is Windows 11 or Windows Server.
- Each container is hosted in its own light weight Hyper-V VM.
- Each container uses the kernel inside the Hyper-V VM which provides an extra layer of separation between containers.
- All windows containers require a user space of either nanoserver or windowsservercore.

Prerequisites (Common to Both)

1. Docker Hub Account (Optional but Recommended)

   - Required? Not mandatory to install Docker.
   - Recommended? Yes, for pulling/pushing images, managing private repositories.
   - Create at: https://hub.docker.com/signup

Section A: Running Docker on Windows 11

- **Requirements**
- **Windows 11 Home or Pro**
- **WSL 2** (Windows Subsystem for Linux) — needed for Home edition
- **Hyper-V** — enabled on Pro edition
- **PowerShell** (already preinstalled in Windows 11)
- If not, install it

PowerShell Vs WSL terminal

1. Windows PowerShell (PowerShell CLI)

- What it is
  - Windows-native command line
  - Works with:
    - Windows commands
    - Windows filesystem
    - Windows services

2. Windows Subsystem for Linux CLI (WSL terminal)

- What it is
  - Real Linux environment inside Windows
  - You get:
    - Bash shell
    - Linux commands
    - Linux filesystem

PowerShell Vs WSL terminal

Feature	PowerShell CLI	WSL CLI (Linux)
OS environment	Windows	Linux
Shell	PowerShell	Bash
Commands	Windows commands	Linux commands
File paths	C:\Users\...	/home/user/...
Docker backend	Connects to WSL	Runs inside WSL

Section A: Running Docker on Windows 11

- Open PowerShell **as Administrator** (right click on PowerShell) and run:
  - `dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart`
  - `dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart`
- Then **restart your PC**.

Section A: Running Docker on Windows 11

2. Install WSL2 and Ubuntu

- Open **PowerShell (admin)** again:
  - `wsl --install`
- This installs:
- *WSL 2 backend*
- *Ubuntu (default distribution)*
- Restart again if prompted. To check:
- Open **cmd** and type powershell (not in ubuntu)
  - `wsl --list --verbose`

Section A: Running Docker on Windows 11

3. Download and Install Docker Desktop

- Go to `https://www.docker.com/products/docker-desktop/`
- Click Download for Windows.
- Run the installer:
- Make sure “Use WSL 2” is checked.
- Accept default options.
- *Before running any docker command open Docker Desktop*

Section A: Running Docker on Windows 11

4. Test Docker Installation

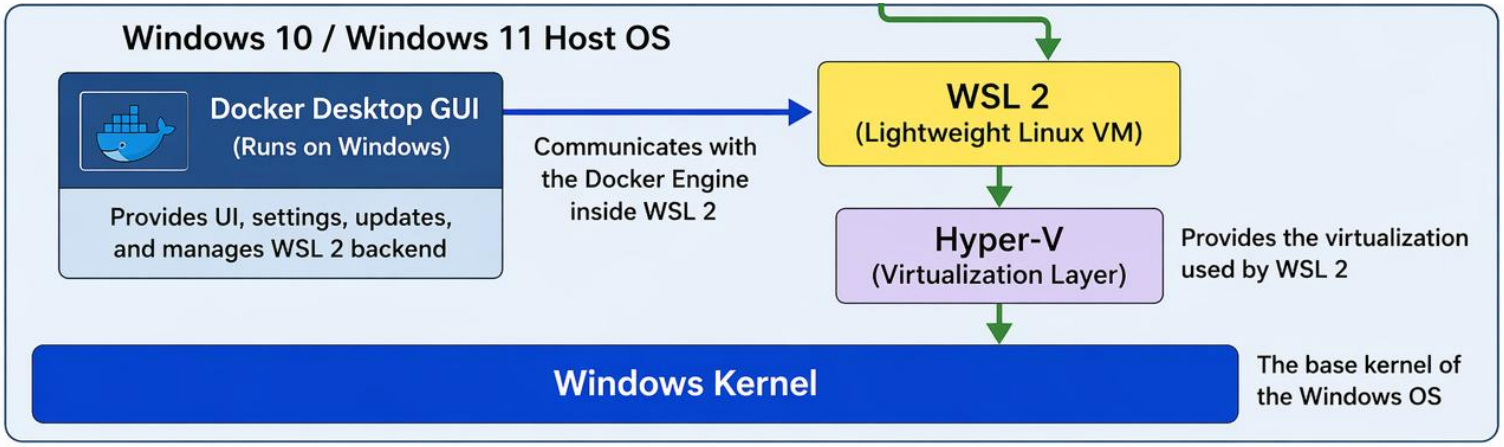
- Open either:

- PowerShell or
- Command Prompt or
- **WSL terminal (like Ubuntu) (preferable)**
  - `docker --version`
  - `docker run hello-world`
- If successful, Docker is installed correctly.

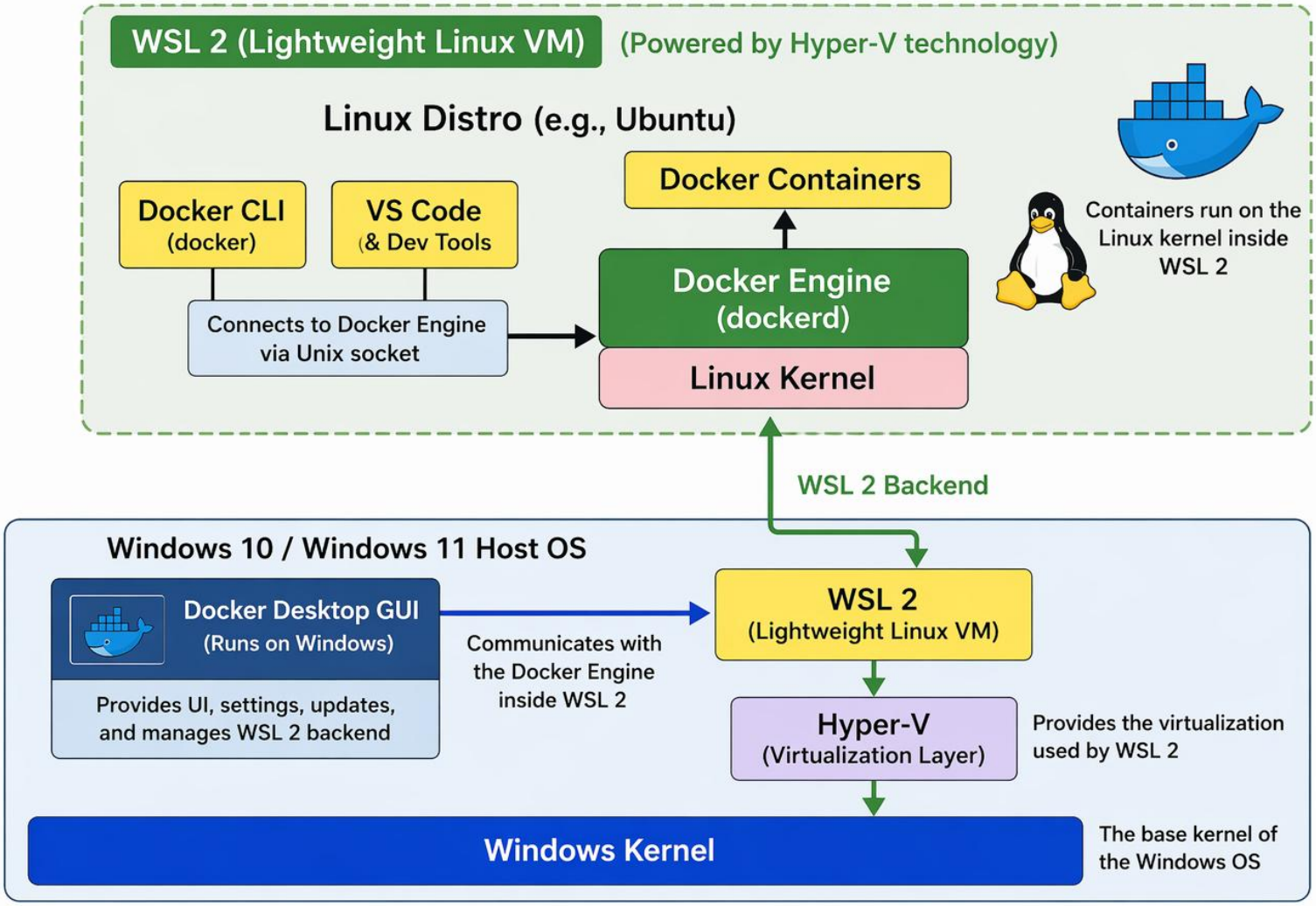
Section B: Running Docker Ubuntu or EC2

- Install *docker desktop* or install it on your *ububtu* EC2 instance using
- *sudo apt install docker.io -y*
- Check, if running or not
  - *sudo systemctl status docker*
  - *docker run hello-world*
  - *sudo usermod –aG docker ubuntu*
- (if earlier command doesn't work)
  - *logout*
- again login and run (will work)

How Docker Actually Runs Inside WSL



How Docker Actually Runs Inside WSL



To do list

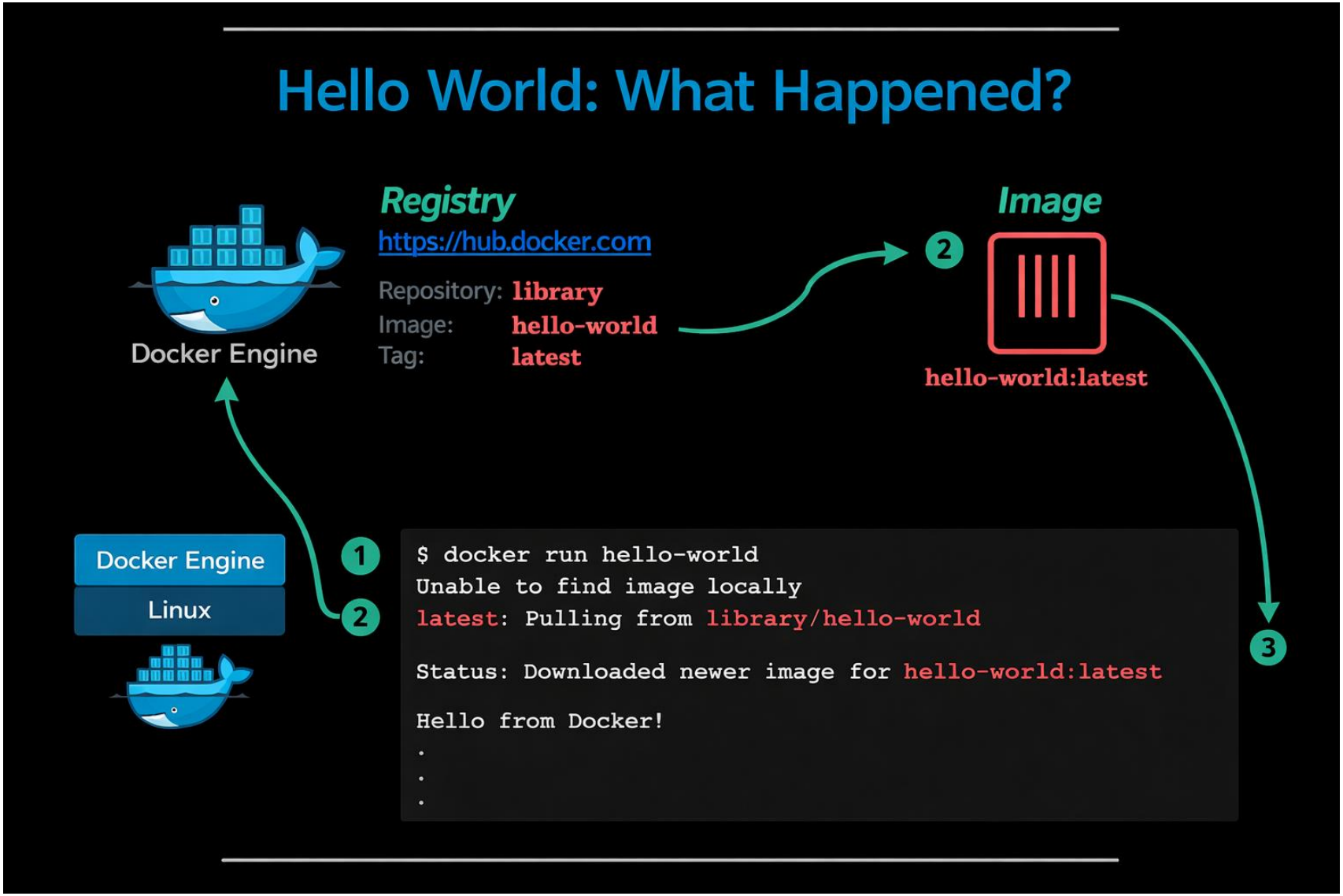
- Go to https://hub.docker.com/ and sign up for an account.
- Find your confirmation email and active your account.
- Explore images from docker hub.
- Understanding official Images, Tags Search images on docker hub.

https://labs.iximiuz.com/

Running first container

- *docker container run hello-world*
- Since we just started, there are no images stored locally (Unable to find image...) so Docker engine goes to its default Docker registry, which is Docker Hub, to look for an image named “hello-world”.
- It finds the image there, pulls it down, and then runs it in a container.

Running first container



What happens?

This Dockerfile uses the 'scratch' base image, which is an empty image FROM scratch

Copy the hello executable into the container

COPY hello /

Set the executable as the container's entry point

CMD ["/hello"]

Explanation:

- Base Image (FROM scratch): The scratch image is an empty image, serving as a foundation for building minimal containers.
- Copy Command (COPY hello /): This line copies the compiled hello binary into the root directory of the container.
- Command (CMD ["/hello"]): Sets the default command to execute the hello binary when the container runs.
- The *hello* binary is a simple program that outputs a greeting message. Its source code is written in C .

Commands…

$ docker image pull alpine

- The pull command fetches the alpine image from the Docker registry and saves it in our system.

$ docker image ls

- docker image command to see a list of all images on our system.

Few docker commands to try

- Check Docker Version and Info
  - $ *docker version*
  - $ *docker info*
- Working with Images
  - $ *docker pull ubuntu*
  - $ *docker images //get all images listings*
  - $ *docker rmi ubuntu //remove image*
- Running Containers
  - $ *docker run [OPTIONS] <image_name>*
    - **-d : Run container in detached mode (in the background).**
    - **-it : Run container interactively with a terminal.**

Few docker commands to try

$ *docker ps*

$ *docker ps -a*

- Interacting with Running Containers

  $ *docker exec -it my-ubuntu-container*

- Stopping and Removing Containers

  $ *docker stop my-ubuntu-container*

  $ *docker rm my-ubuntu-container*

Create your image and Container

- **Step 1: Create a Python Script (hi.py) through wsl**
- Suppose you are at wankarcs@DESKTOP-6780OLS:~$ // it is WSL terminal

  $ mkdir docker-python-lab

  $ cd docker-python-lab

  $ nano hifi.py

- This will create a file named **hifi.py** with the following content:

  #!/usr/bin/env python3

  print("Hello Virtualization Students from Rajeev")

Create your image and Container

- **Step 2: Create a Dockerfile**
- In the same directory (docker-python-lab)

$ nano Dockerfile

- Create a file named **Dockerfile** (with no extension) in the same directory as hifi.py, and add the following content:

Remember to check: python3 --version

Create your image and Container

```
# Use the official Ubuntu image as the base image
FROM ubuntu:latest

# Update package repository and install Python 3, it is mandatory
RUN apt-get update && apt-get install -y python3

# Copy the hi.py file into the container
COPY hifi.py /usr/src/app/hifi.py

# Set the working directory
WORKDIR /usr/src/app

# Set the default command to execute the Python script
CMD ["python3", "hifi.py"]
```

Userspaces

- We cannot use ***apt-get*** without a base image, that is exactly because user space (**tools, libraries, package manager**) is missing, and only the kernel is not enough to run those commands

- Windows user space includes:
  - Large number of DLLs
  - Windows API layers
  - Compatibility components
- Linux containers can be extremely small because user space is modular, but Windows containers require a larger user space due to OS complexity.

Create your image and Container

- *Explanation:*
- **FROM ubuntu:latest**: Uses the latest Ubuntu image as the base.
- **RUN apt-get update && apt-get install -y python3**: Updates the package list and installs Python 3.
- **COPY hi.py /usr/src/app/hifi.py**: Copies your Python script into the container.
- **WORKDIR /usr/src/app**: Sets the working directory.
- **CMD ["python3", "hifi.py"]**: Specifies the command to run when the container starts.

Create your image and Container

- **Step 3: Build the Docker Image**

1. Open your terminal or command prompt and navigate to the directory containing **hifi.py** and the **Dockerfile**.
2. Run the following command to build the Docker image (replace *hifi* with your desired image name):

docker build -t hifi .

Since no tag is specified, it defaults to latest. The . at the end specifies the build context, indicating that Docker should use the current directory. –t is for tag.

Create your image and Container

- **Step 4: Run the Docker Container**
- After building the image, run a container from it with this command:

  docker run --rm hifi
- *Explanation:*
- **docker run**: Runs a new container.
- **--rm**: Automatically removes the container when it exits.
- **hifi**: Specifies the image name.

  – *Hello Virtualization Students from Rajeev*

Push to docker hub (optional)

- **Step 1: Create a free Docker Hub Account (if you don’t already have one)**
- **Step 2: Log in from the Command Line**
  - Open your Command Prompt, PowerShell, or terminal.
  - Run:

    docker login

Push to docker hub (optional)

- **Step 3: Tag Your Image for Docker Hub**
- Docker images are pushed to a repository on Docker Hub. You need to tag your image with your Docker Hub username and the repository name.
- Suppose your Docker Hub username is yourusername and your image is called myhi, you can tag it like this:

docker tag hifi yourusername/hifiv1:latest

Push to docker hub (optional)

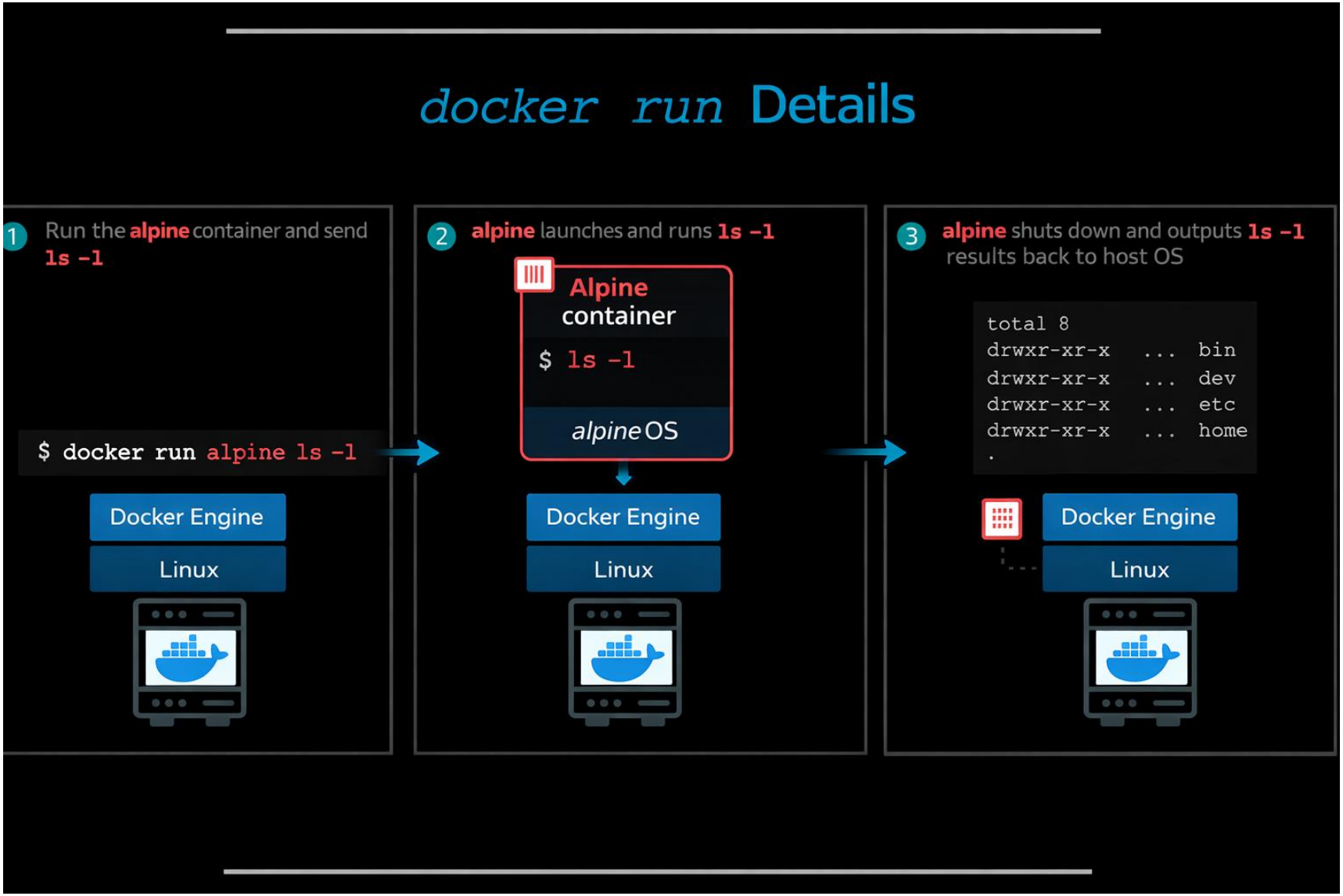
- **Step 4: Push the Image to Docker Hub**
- Use the docker push command to upload your tagged image:

docker push yourusername/hifiv1:latest

Commands…

- *docker container run alpine ls -l*
- When we call run, the Docker client finds the image (alpine in this case), creates the container and then runs a command in that container.
- When we run docker container run alpine, we provided a command (ls -l), so Docker executed this command inside the container for which we saw the directory listing. After the ls command finished, the container shuts down.

Cont....


docker run Details
1 Run the alpine container and send ls -1
$ docker run alpine ls -1
Docker Engine
Linux
2 alpine launches and runs ls -1
Alpine container
$ ls -1
alpine OS
Docker Engine
Linux
3 alpine shuts down and outputs ls -1 results back to host OS
total 8
drwxr-xr-x ... bin
drwxr-xr-x ... dev
drwxr-xr-x ... etc
drwxr-xr-x ... home
.
Docker Engine
Linux

Commands…

- *docker container run -it alpine /bin/sh*

- Inside the container running a Linux shell and we can try out a few commands like *ls -l, uname -a* and others.
- *Note that Alpine is a small Linux OS so several commands might be missing.*

Commands…

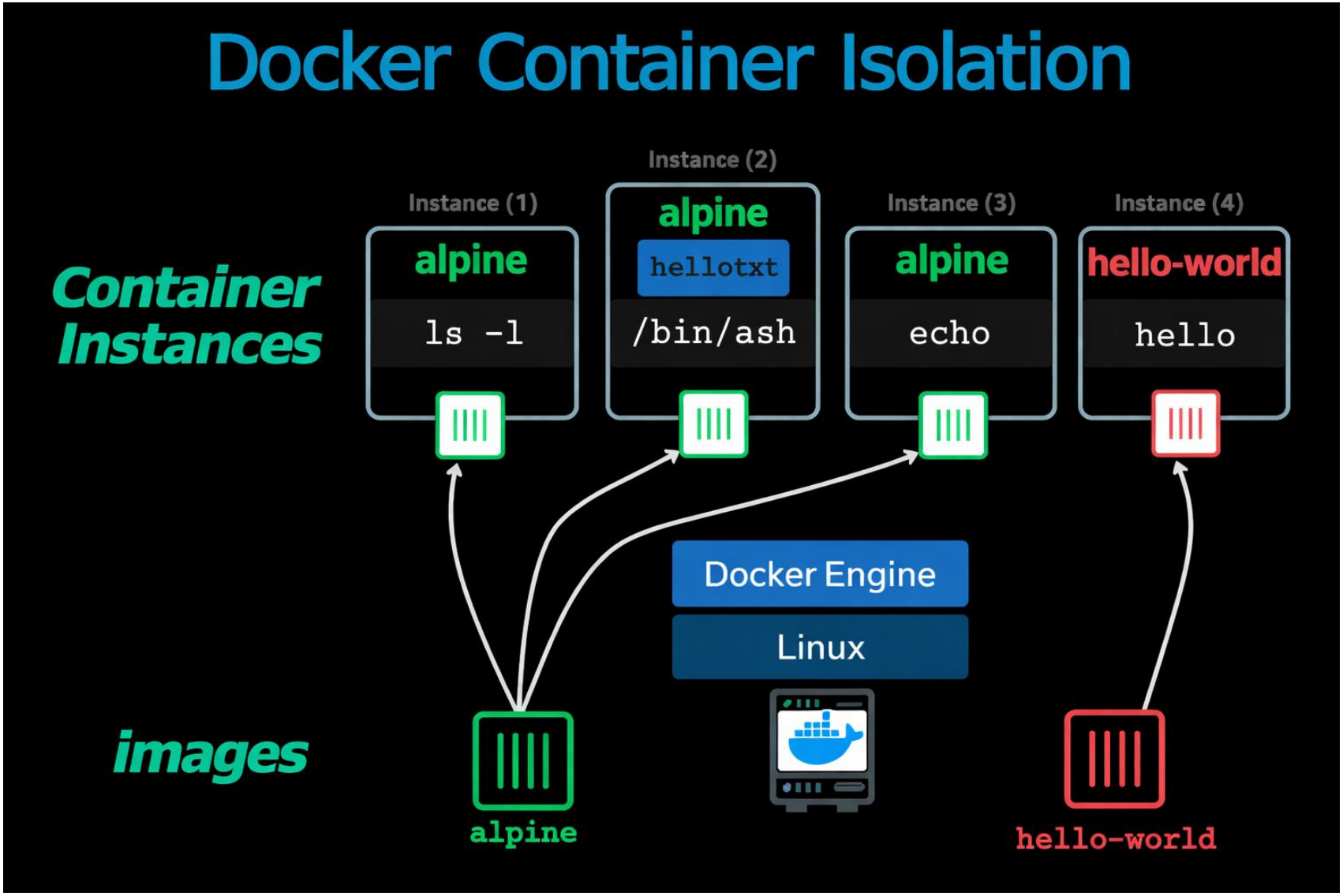
- *docker container ls*
- We can see these instances using the docker container ls command.
- The docker container ls command by itself shows you all containers that are currently running:

Commands…

- *docker container ls -a*
- Lists all containers, including those that are stopped or exited.
- The -a flag stands for "all," showing every container regardless of its state

- *by default a container has no way of interacting with other containers*

Isolation…


Docker Container Isolation
Container Instances
Instance (1)
alpine
ls -l
Instance (2)
alpine
hellotxt
/bin/ash
Instance (3)
alpine
echo
Instance (4)
hello-world
hello
Docker Engine
Linux
images
alpine
hello-world

Cheatsheet for Docker CLI

Run a new Container

Start a new Container from an Image
```
docker run IMAGE
docker run nginx
```

...and assign it a name
```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port
```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports
```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background
```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname
```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry
```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container
```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/:/usr/share/nginx/html nginx
```

...but change the entrypoint
```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

Manage Containers

Show a list of running containers
```
docker ps
```

Show a list of all containers
```
docker ps -a
```

Delete a container
```
docker rm CONTAINER
docker rm web
```

Delete a running container
```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers
```
docker container prune
```

Stop a running container
```
docker stop CONTAINER
docker stop web
```

Start a stopped container
```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host
```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container
```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container
```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container
```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container
```
docker commit CONTAINER
docker commit web
```

Manage Images

Download an image
```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository
```
docker push IMAGE
docker push myimage:1.0
```

Delete an image
```
docker rmi IMAGE
```

Show a list of all Images
```
docker images
```

Delete dangling images
```
docker image prune
```

Delete all unused images
```
docker image prune -a
```

Build an image from a Dockerfile
```
docker build DIRECTORY
docker build .
```

Tag an image
```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile
```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file
```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file
```
docker load -i TARFILE
docker load -i nginx.tar
```

Info & Stats

Show the logs of a container
```
docker logs CONTAINER
docker logs web
```

Show stats of running containers
```
docker stats
```

Show processes of container
```
docker top CONTAINER
docker top web
```

Show installed docker version
```
docker version
```

Get detailed info about an object
```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container
```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container
```
docker port CONTAINER
docker port web
```

https://dockerlabs.collabnix.com/docker/cheatsheet/

Image creation from a container

- Type the following content into a file named *index.js*. You can use vi, vim or several other Linux editors in this exercise.

```
var os = require("os");
var hostname = os.hostname();
console.log("hello from " + hostname);
```

Image creation from a container

- Create a file named *Dockerfile* and copy the following content into it.

```
FROM alpine
RUN apk update && apk add nodejs
COPY . /app
WORKDIR /app
CMD ["node","index.js"]
```

Image creation from a container

- Let's build our image out of this Dockerfile and name it *hello:v0.1*:

- *docker image build -t hello:v0.1 .*

The -t (or --tty) flag tells Docker to allocate a virtual terminal session within the container. This is commonly used with the -i (or --interactive) option, which keeps STDIN open even if running in detached mode

Image creation from a container

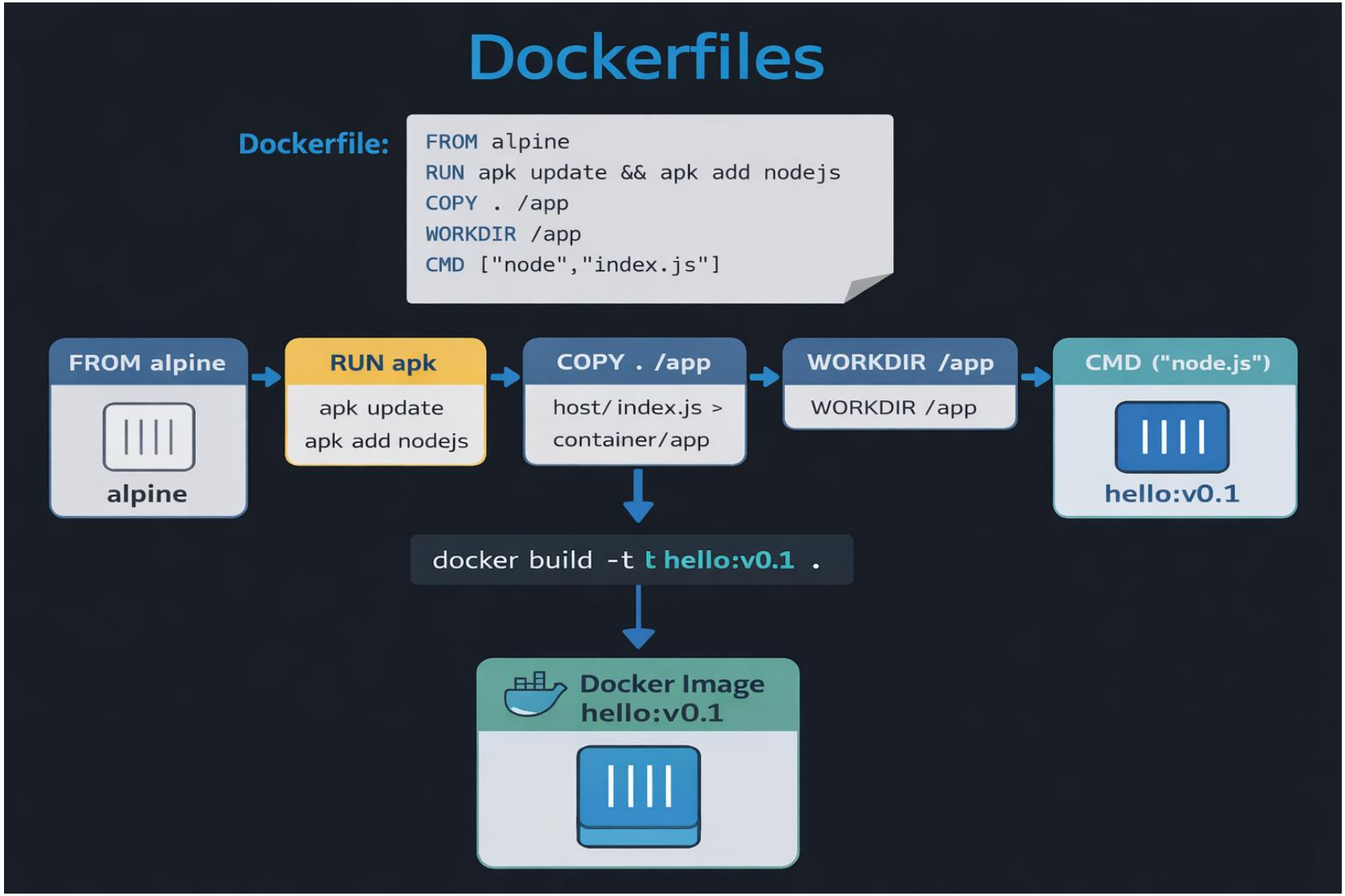


Image creation from a container

- We then start a container to check that our applications runs correctly:
- *docker container run hello:v0.1*
- We should then have an output similar to the following one (the ID will be different though).
- hello from 92d79b6de29f

Image creation from a container

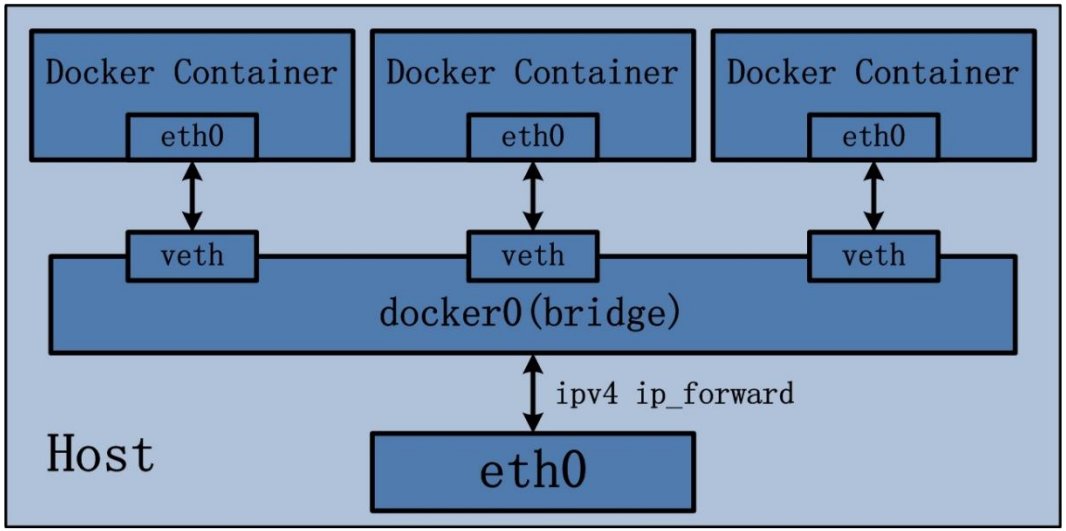
- **What just happened?** We created two files: our application code (index.js) is a simple bit of javascript code that prints out a message.
- *Dockerfile* is the instructions for Docker engine to create our custom container.
- This *Dockerfile* does the following:
  - Specifies a base image to pull **FROM** - the ***alpine*** image we used in earlier labs.

Image creation from a container

- Then it **RUN**s two commands (*apk update* and *apk add*) inside that container which installs the Node.js server.
- Then we told it to **COPY** files from our working directory in to the container. The only file we have right now is our *index.js*.
- Next we specify the **WORKDIR** - the directory the container should use when it starts up
- And finally, we gave our container a command (**CMD**) to run when the container starts.

- docker ps
- docker ps -a
- All running containers

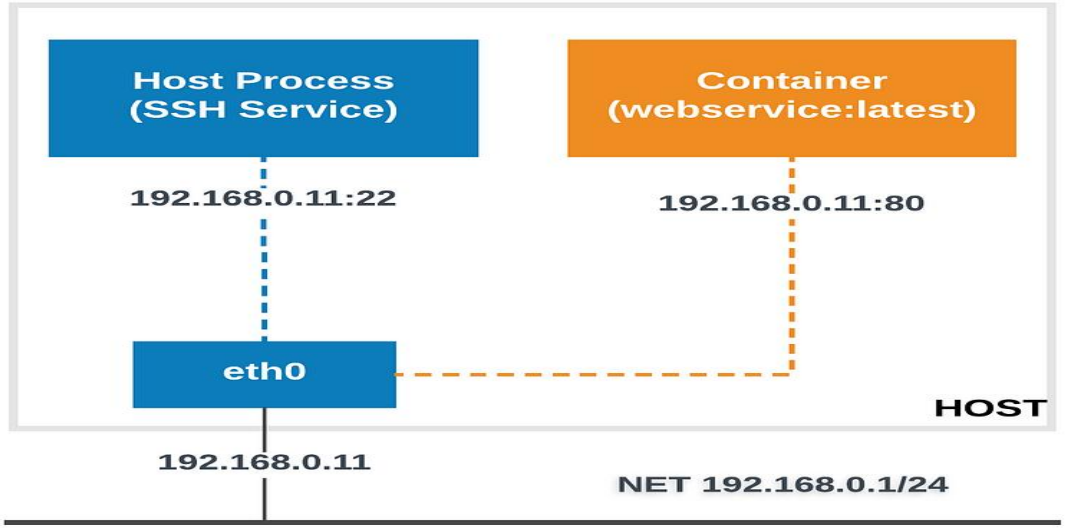
Bridge Networking



- The bridge network represents the docker0 network present in all Docker installations.
  - *docker network ls*
  - *docker network inspect 5caefda1a88e*

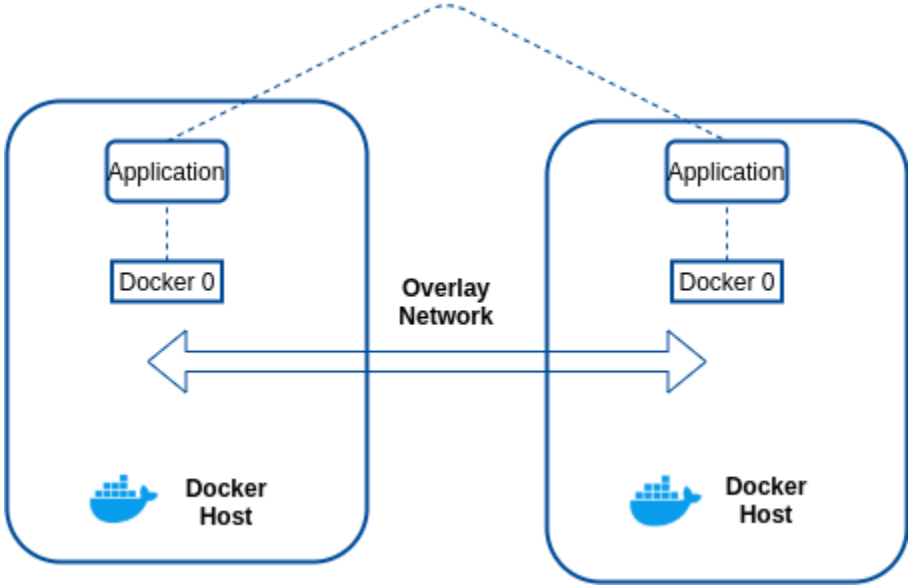
In case logical isolation is required between the containers on the same host then Custom Bridge Network is used

Host Mode Networking



- Container shares the networking namespace of the host, it is directly exposed to the public network. It will use the host's IP address and host's TCP port space

Overlay Network



- It provides connectivity when multiple Docker hosts are running. Bridge network only works on the same host but in production, we run a Docker container in multiple hosts using Docker Swarm.

Network Lab-Same Network

Step 1: Open wsl terminal

Step 2: Create the network using the command

```
docker network create mynet
```

Step 3: Run these containers nginx and alpine using

```
docker run -d --name web --network mynet nginx
docker run -it --name client --network mynet alpine sh
```

Step 4: ping web

```
ping web
```

Step 5: Remove the network

```
docker network ls
docker network rm mynet
docker network ls
```

Network Lab-Isolation

Step 1: Open wsl terminal

Step 2: Create the network using the command

docker network create net1

docker network create net2

Step 3: Run these containers nginx and alpine using

docker run -d --name c1 --network net1 nginx

docker run -it --name c2 --network net2 alpine sh

Step 4: ping web

ping c1 (it fails as they are different networks)

Step 5: Remove the network

docker network rm c1

docker network rm c2

docker network ls

Volumes in Docker

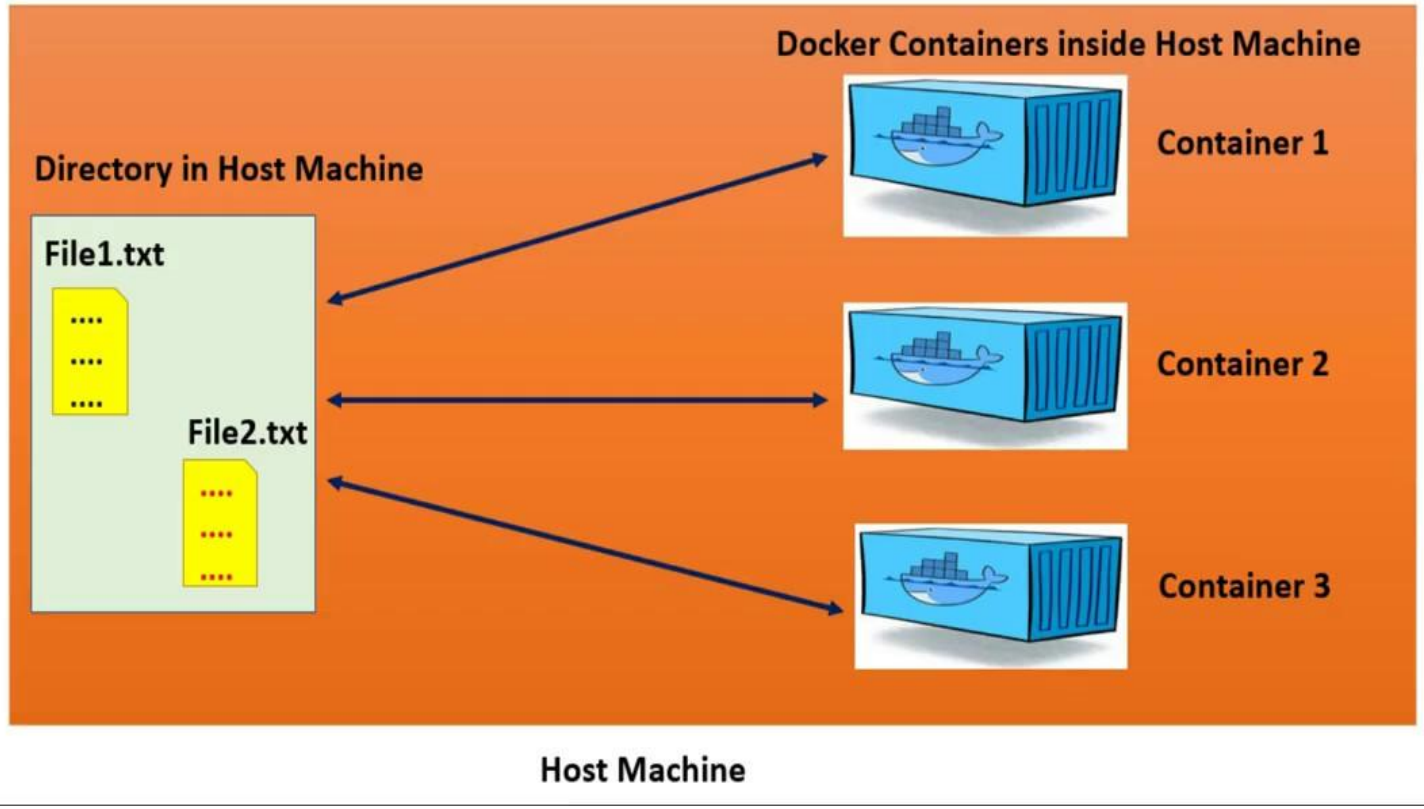
- Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container.
- The volumes are stored on the host, independent of the container life cycle.
- Volumes allows users to back up data and share file systems between containers easily.
- Volumes can be more safely shared among multiple containers.

Volumes in Docker

- Volumes are often a better choice than persisting data in a container's writable layer, because a volume does not increase the size of the containers using it.

- Volume's contents exist outside the lifecycle of a given container. Using volumes we can Backup a container.

Volumes



https://medium.com/@infosecnubes/mount-volume-with-read-only-mode-in-docker-ad9f566ef2b9

Volume Lab

Step 1: Create a volume

```
docker volume create rwvol
```

Step 2: Run container and write data

```
docker run -it --name c1 -v rwvol:/data alpine sh
//Inside container:
echo "Hello Students" > /data/file.txt
exit
```

Step 3: Remove container

```
docker rm c1
//Container is removed
```

Step 4: Run new container with same volume

```
docker run -it --name c2 -v rwvol:/data alpine sh
//Inside container:
cat /data/file.txt
docker volume rm myvol
```