Cloud Computing

Session 4-5

Containers

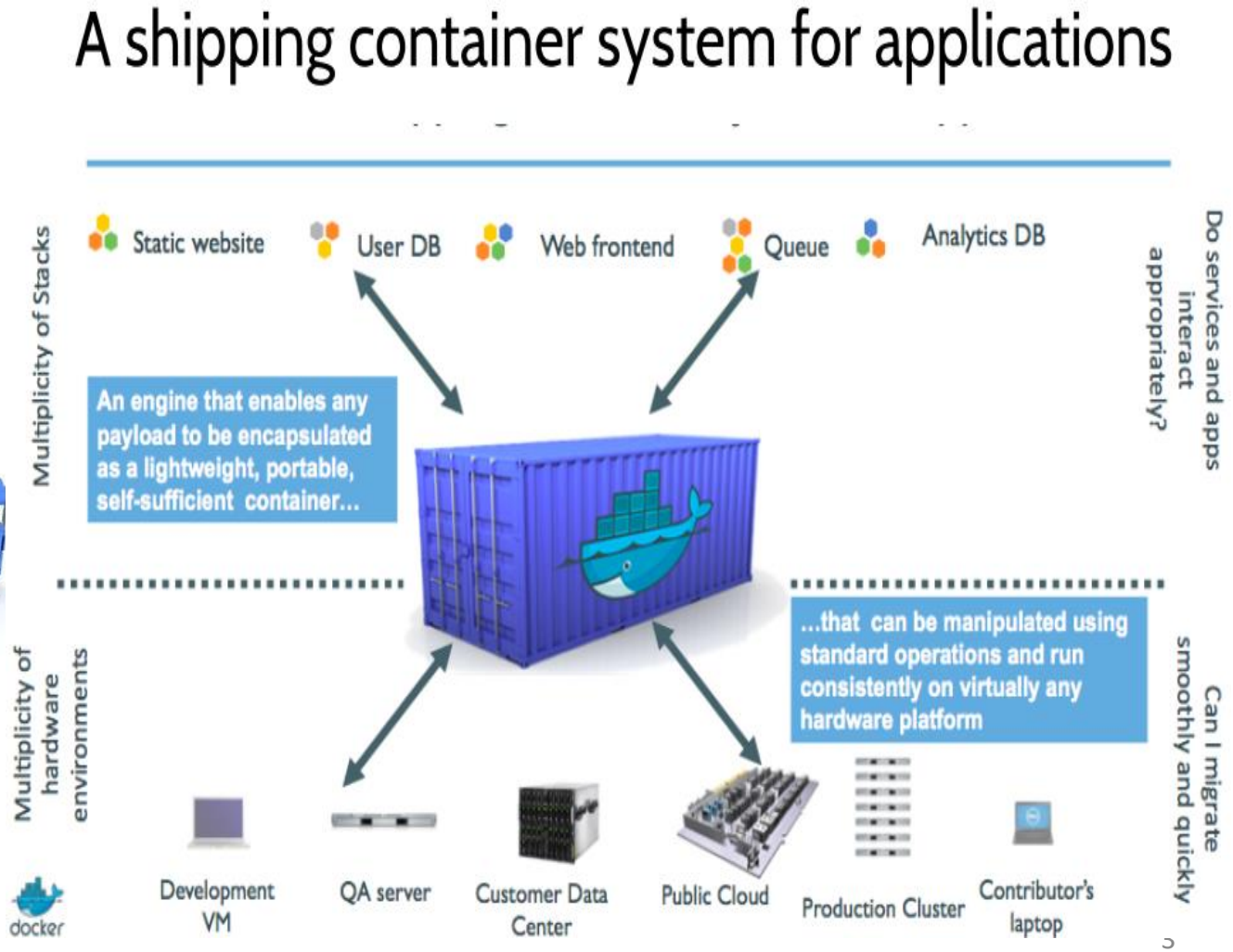
BITS Pilani

Shwetha Vittal
shwetha.vittal@pilani.bits-pilani.ac.in

Agenda

- What are containers?
- Namespaces
- Cgroups
- Virtual Machine vs Containers
- Types of Containers
- Docker

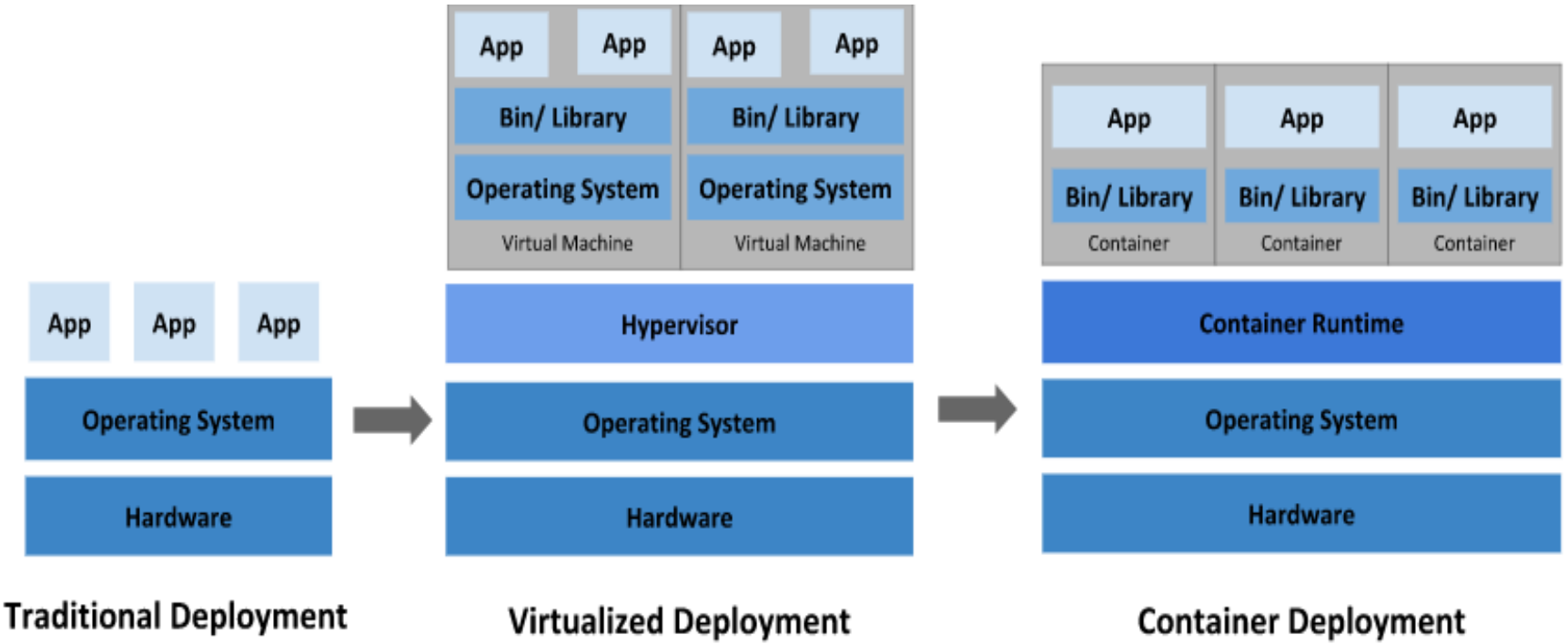
What are Containers?



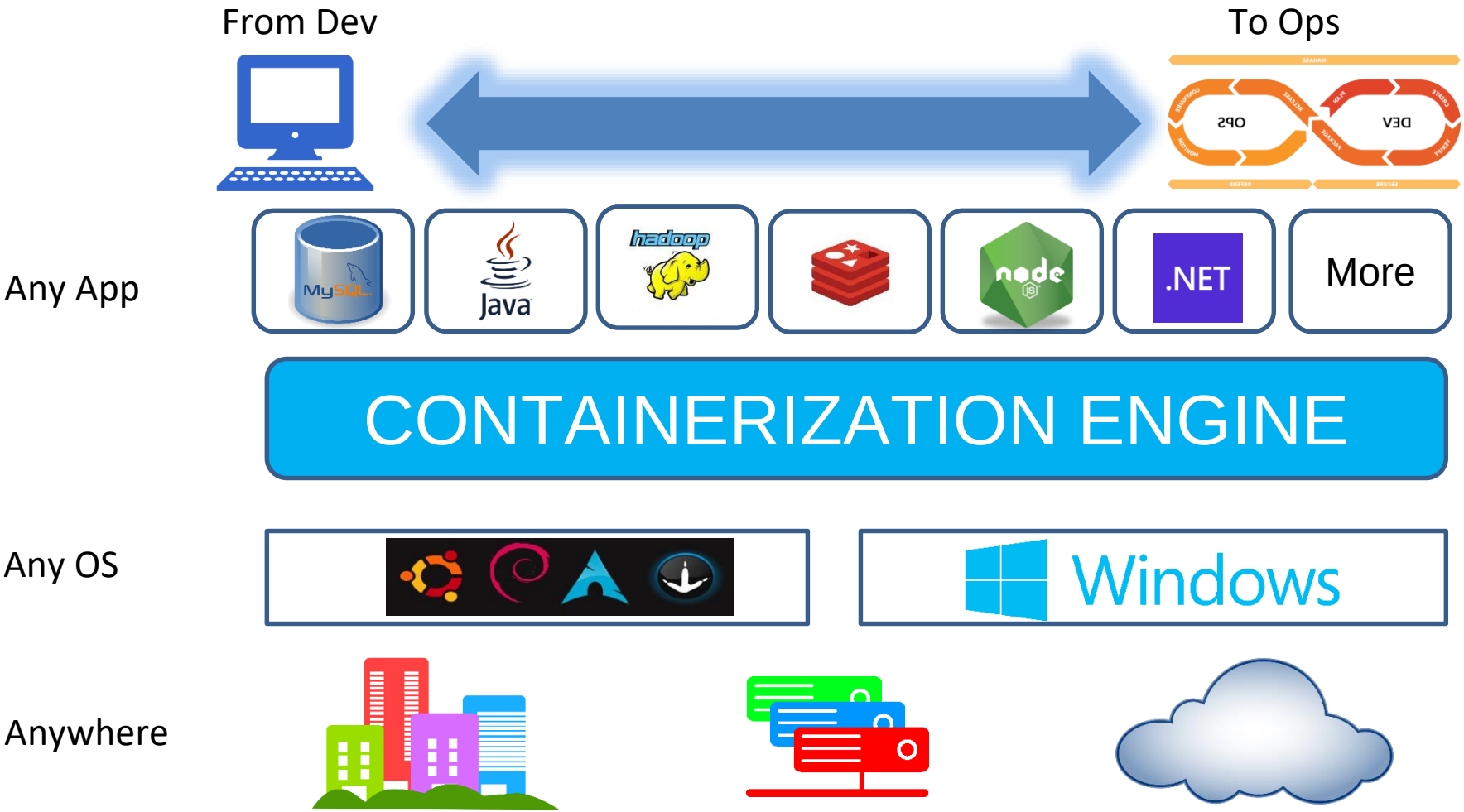
What are Containers ?

- Lightweight virtualization mechanism
- A software container is a standardized package of software.
- Everything needed for the software to run is inside the container
- The software code, runtime, system tools, system libraries, and settings are all inside a single container
- Managed by the OS kernel running on the host system
- Has its own isolated memory, CPU, storage, process table, and networking interfaces
- Faster provisioning for newer applications

Going back in Time to Now



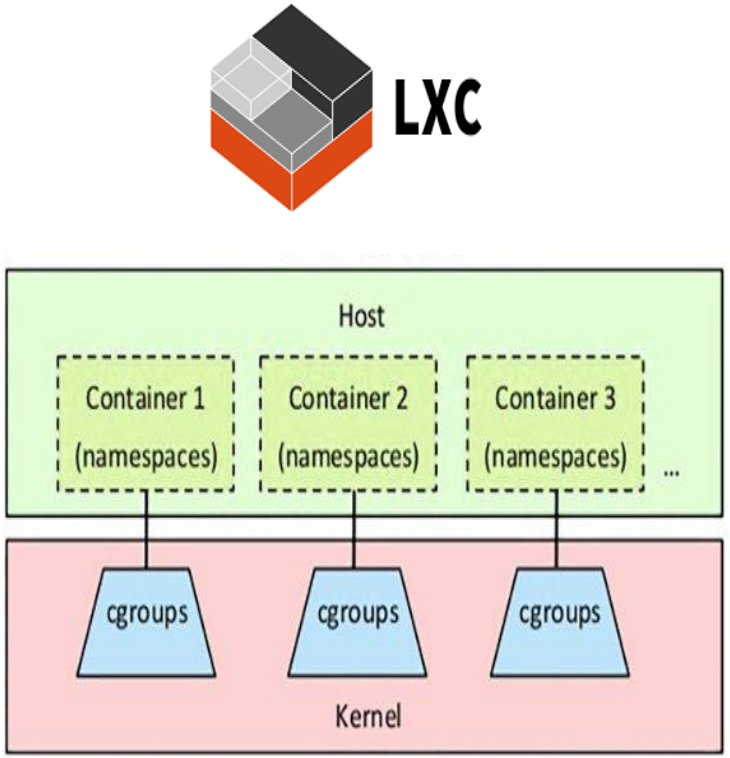
What are Containers?



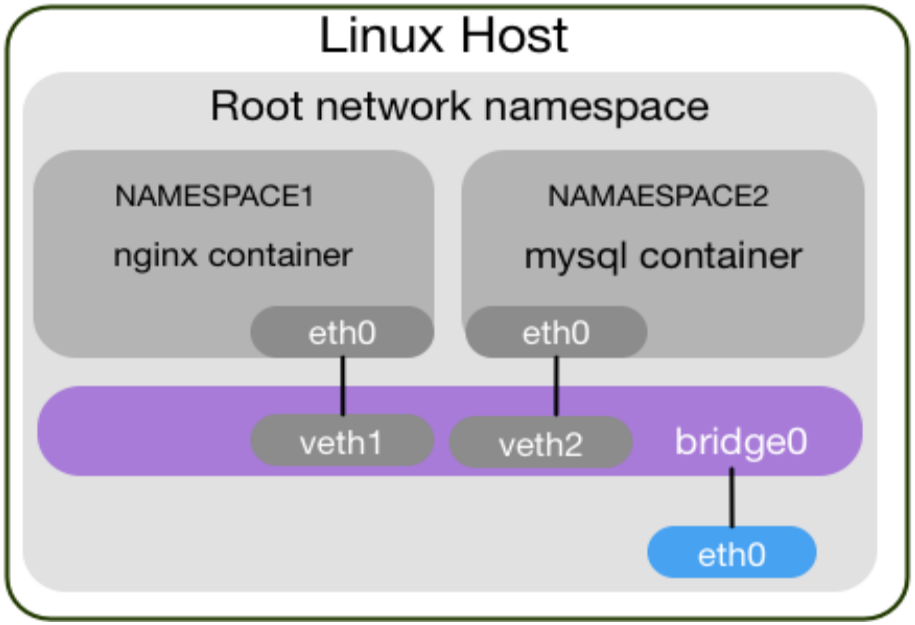
Containers

Containers are powered by two underlying Linux Kernel technologies

- Namespaces
- Cgroups



Namespaces



- Kernel mechanism for limiting the visibility that a group of processes has of the rest of a system
- Namespace merged into Linux 3.8
- Limit Visibility
  - Process trees - PIDs
  - Network interfaces
  - User IDs
  - Filesystem mounts

[Namespaces in operation, part 1: namespaces overview [LWN.net]]

Types of Namespaces

- Wrap a particular global system resource in an abstraction
- **Illusion:** Makes it appear to the processes within the namespace that they have **their own isolated instance of the global resource**.
- 6 Main Namespaces
  - Mount namespace
  - UTS namespace
  - IPC namespace
  - PID namespace
  - Network namespace
  - User namespace

```
$ ls -l /proc/13/ns
total 0
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 cgroup -> cgroup:[4026531835]
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 ipc -> ipc:[4026547635]
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 mnt -> mnt:[4026547631]
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 net -> net:[4026547638]
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 pid -> pid:[4026547636]
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 user -> user:[4026531837]
lrwxrwxrwx    1 root     root             0 Feb  6 09:57 uts -> uts:[4026547632]
```

[Namespaces in operation, part 1: namespaces overview [LWN.net]]

Mount Namespace

- Isolate the set of filesystem mount points seen by a group of processes.
- Processes across different mount namespaces (ns) have different views of the filesystem hierarchy.
- mount(), umount()
- Shared or Private mount points
  - Shared mount points propagated to all namespaces across process hierarchy / other processes.
  - Private is not
- Every container – has a custom root file system to start with.
- Any new child process without any shared ns by its parent, will start with empty root filesystem

Namespaces in operation, part 1: namespaces overview [LWN.net]

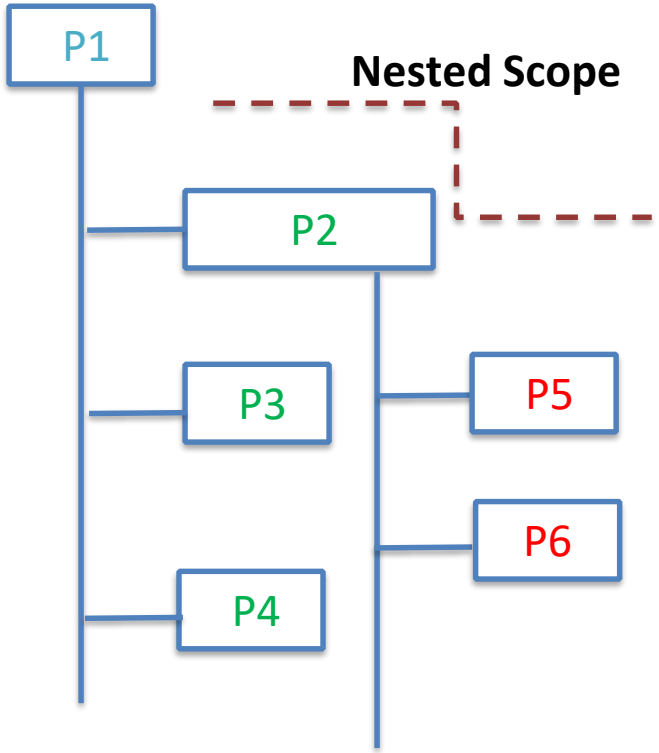
IPC Namespace

- Isolate certain inter process communication (IPC) resources
  - System V IPC objects
  - POSIX message queues.
- Have a private set of IPC objects (sem, shm, msg) inside namespace.

Namespaces in operation, part 1: namespaces overview [LWN.net]

PID Namespace

- Isolate the process ID number space.
- Processes in different PID namespaces can have the same PID.
  - Containers can be migrated between hosts while keeping the same process IDs for the processes inside the container.
  - Allow each container to have its own init – PID 1
- Nested Scope
  - Ancestor->…->Parent -> child



Namespaces in operation, part 1: namespaces overview [LWN.net]
PID namespaces in the 2.6.24 kernel [LWN.net]

Network Namespace

- Provide isolation of the network resources
- Each network namespace has its own network devices, IP addresses, IP routing tables, /proc/net directory

List the network ns(es)

```
ubuntu@ip-172-31-31-148:~$ ls -l /var/run/netns;
total 0
-r--r--r-- 1 root root 0 Feb  6 10:19 mynetworkns
-r--r--r-- 1 root root 0 Feb  6 10:17 testns
```

```
ubuntu@ip-172-31-31-148:~$ ip netns
testns
mynetworkns
```

Adding network ns

```
ubuntu@ip-172-31-31-148:~$ sudo ip netns add mynetworkns
ubuntu@ip-172-31-31-148:~$
```

[Namespaces in operation, part 7: Network namespaces [LWN.net]]

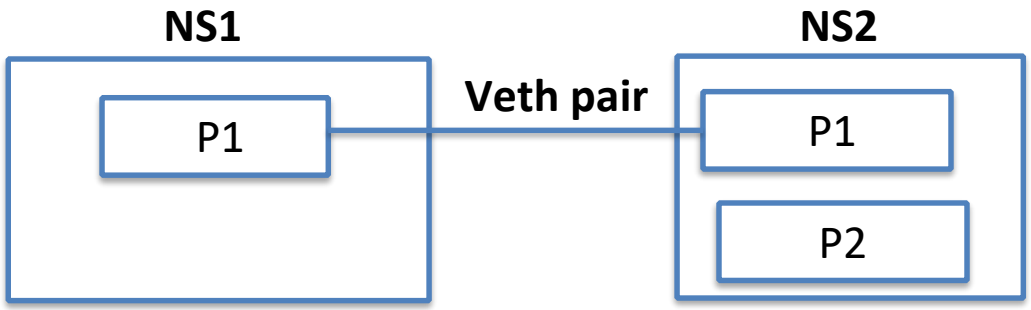
Network Namespace

Network interfaces on host

```
ubuntu@ip-172-31-31-148:~$ ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enX0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 0a:ff:ed:60:2c:e5 brd ff:ff:ff:ff:ff:ff
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:75:31:cf:1a brd ff:ff:ff:ff:ff:ff
```

Network interfaces inside ns

```
ubuntu@ip-172-31-31-148:~$ sudo ip netns exec mynetworkns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```



[Namespaces in operation, part 7: Network namespaces [LWN.net]]

User Namespace

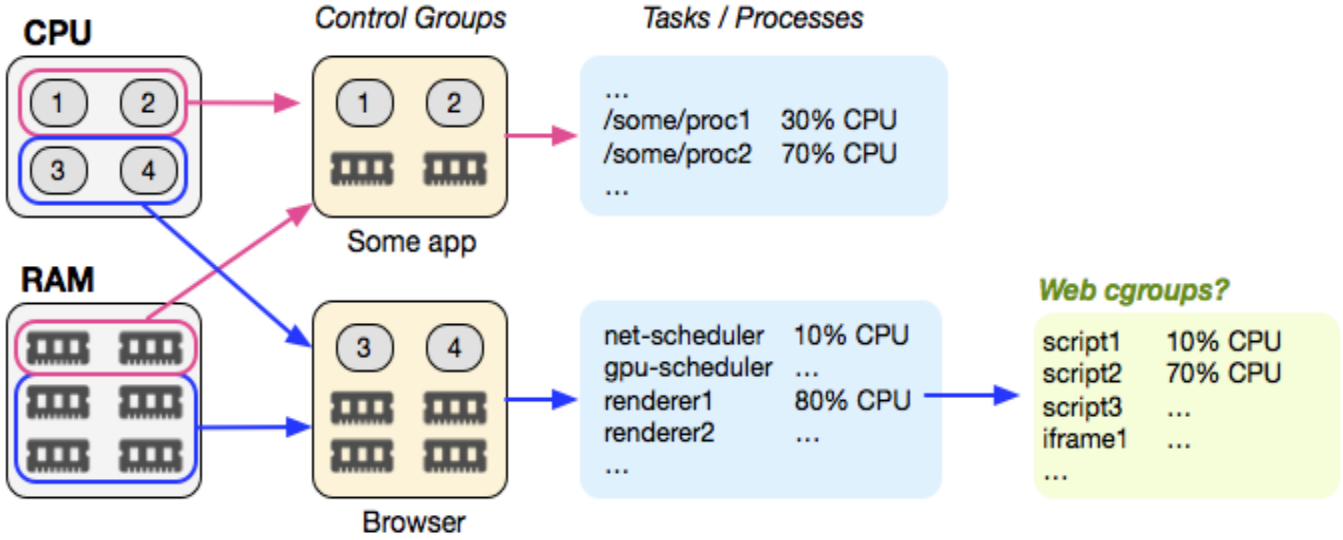
- Isolate the user and group ID number spaces
- A process's user and group IDs can be different inside and outside a user namespace.
  - A process has full root privileges for operations inside the user namespace,
  - But is unprivileged for operations outside the namespace.

Namespaces in operation, part 7: Network namespaces [LWN.net]

Namespace APIs

- System Calls
  - clone(): Create a new process and place it into a new namespace.
  - unshare(): Creates a new namespace and places calling process into it.
  - setns(): Join an existing namespace.
- Commands
  - lsns - all namespaces in the system
  - /proc/PID/ns - which namespace a process belongs to.

Cgroups

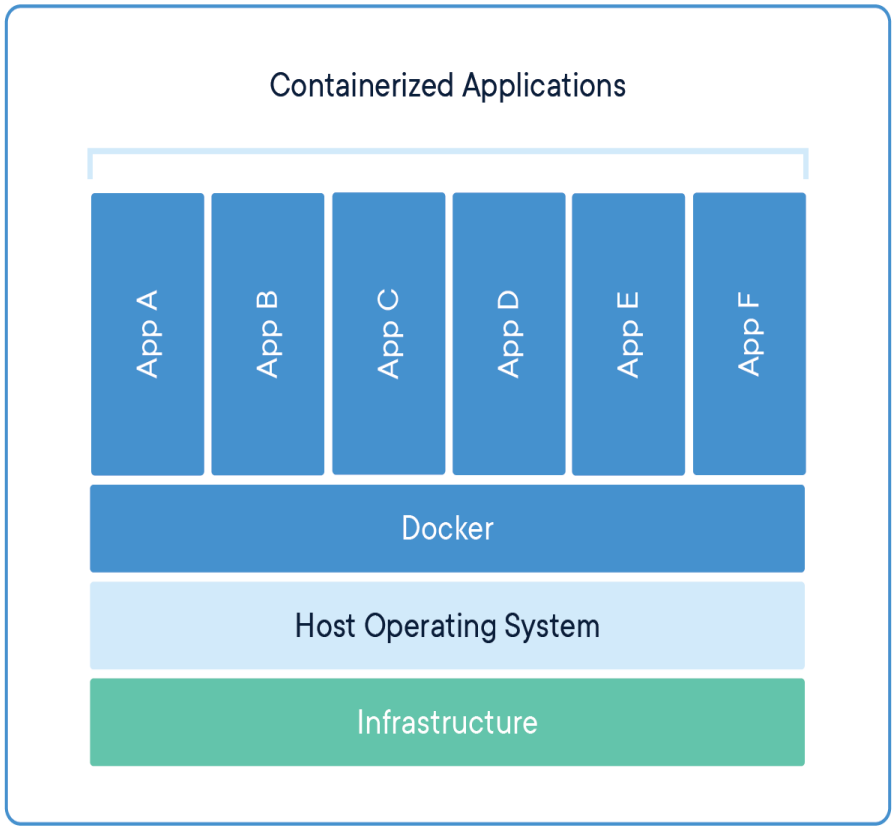


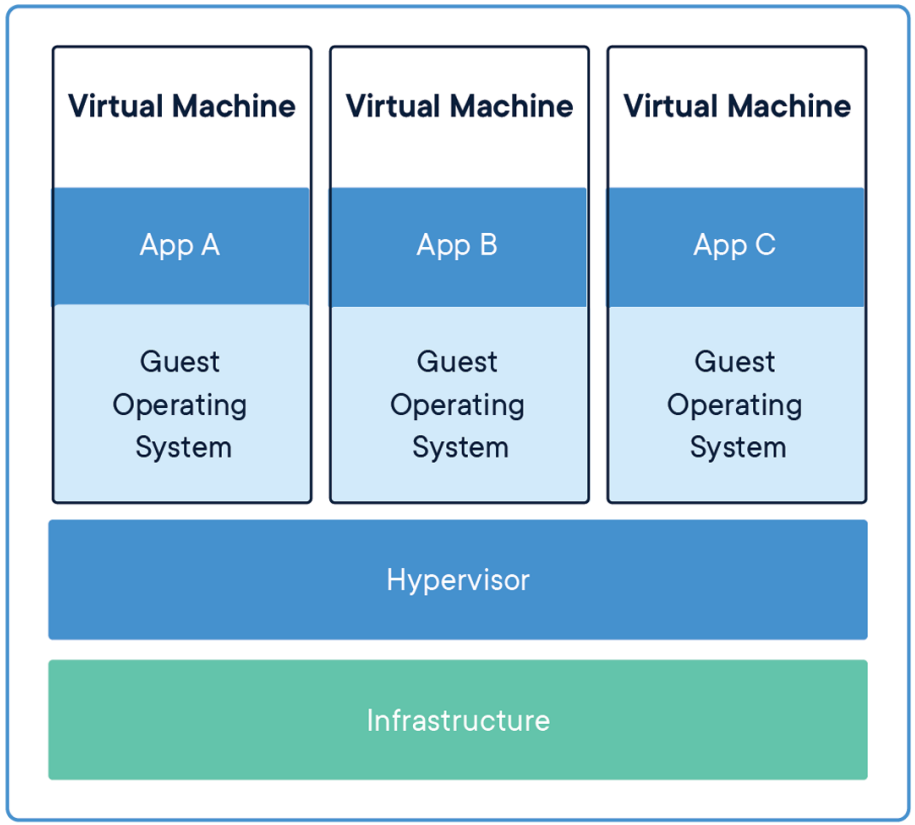
cgroups – Control groups

- A kernel mechanism for limiting and measuring the total resources used by a group of processes running on a system
- Processes can be applied with CPU, memory, network or IO quotas

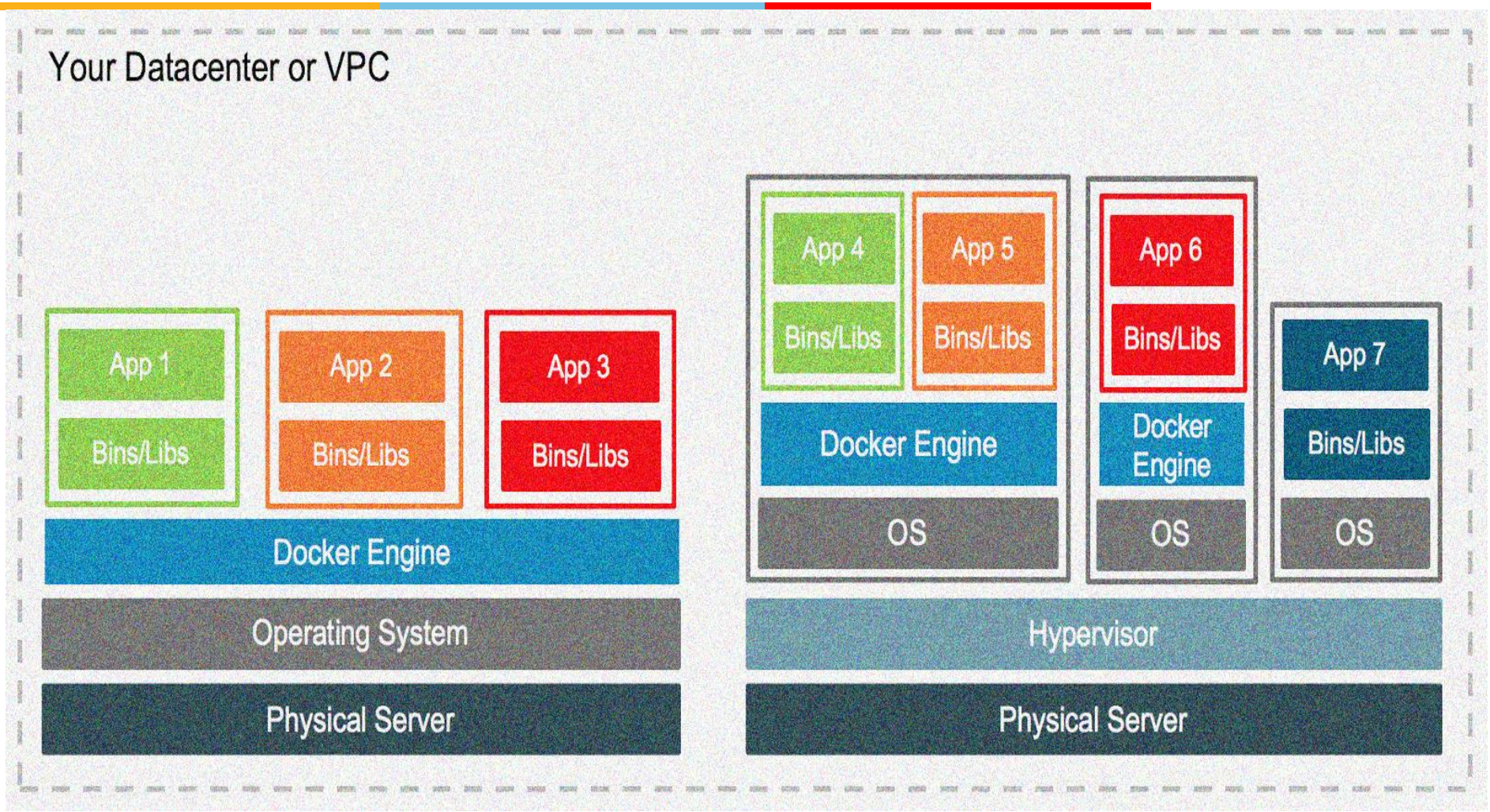
Cgroup merged into Linux 2.6.24

Containers Vs Virtual Machines





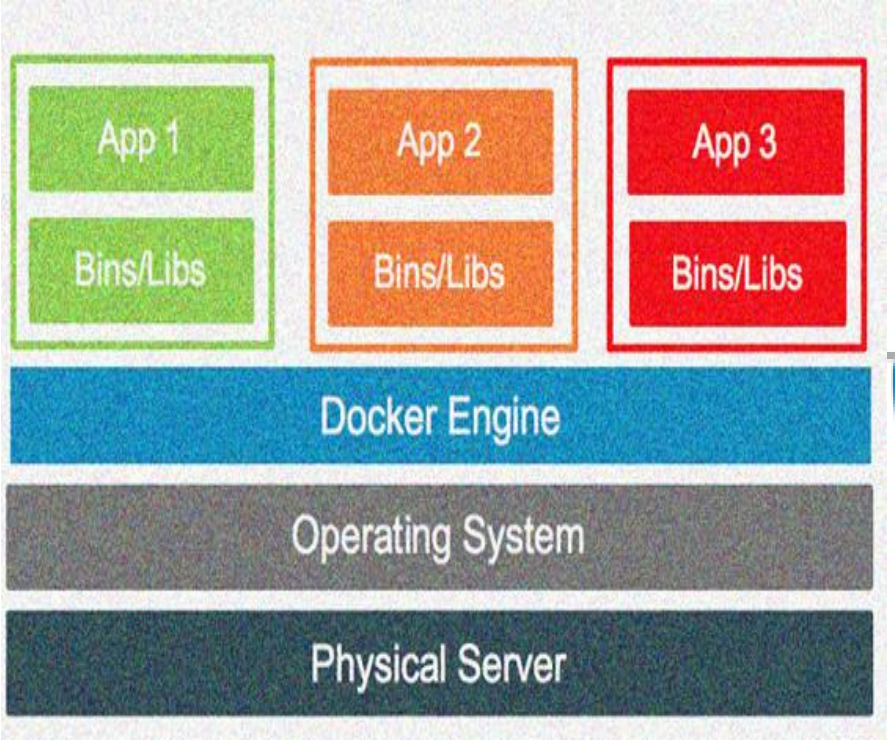
Containers on Virtual Machines ?



Docker

Docker Platform

- Docker is an open platform
- Docker separates applications from hardware infrastructure
- **Containers** are used to package and run an application

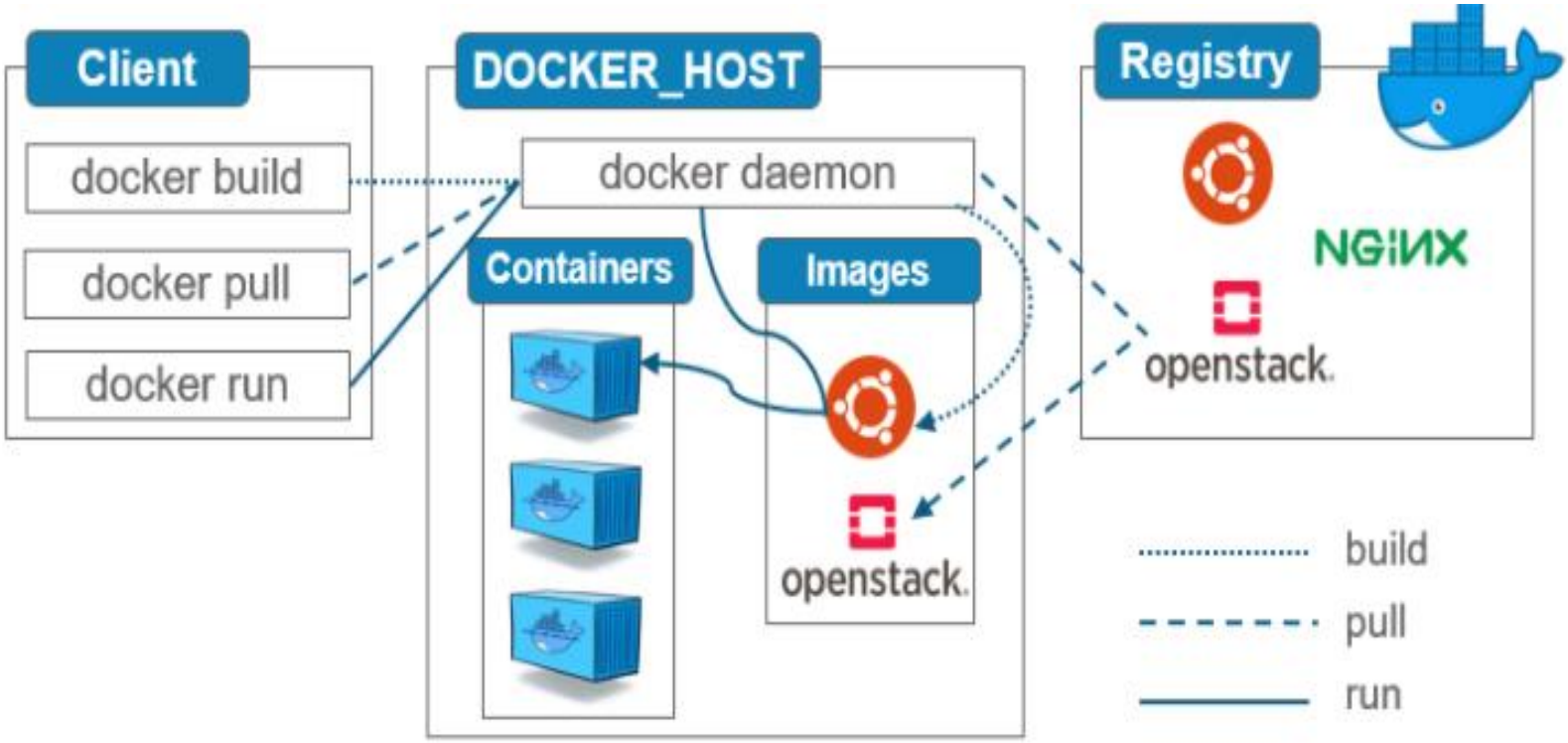


- A single host can run many containers simultaneously
- Containers are lightweight and contain everything needed to run the application

Docker Platform

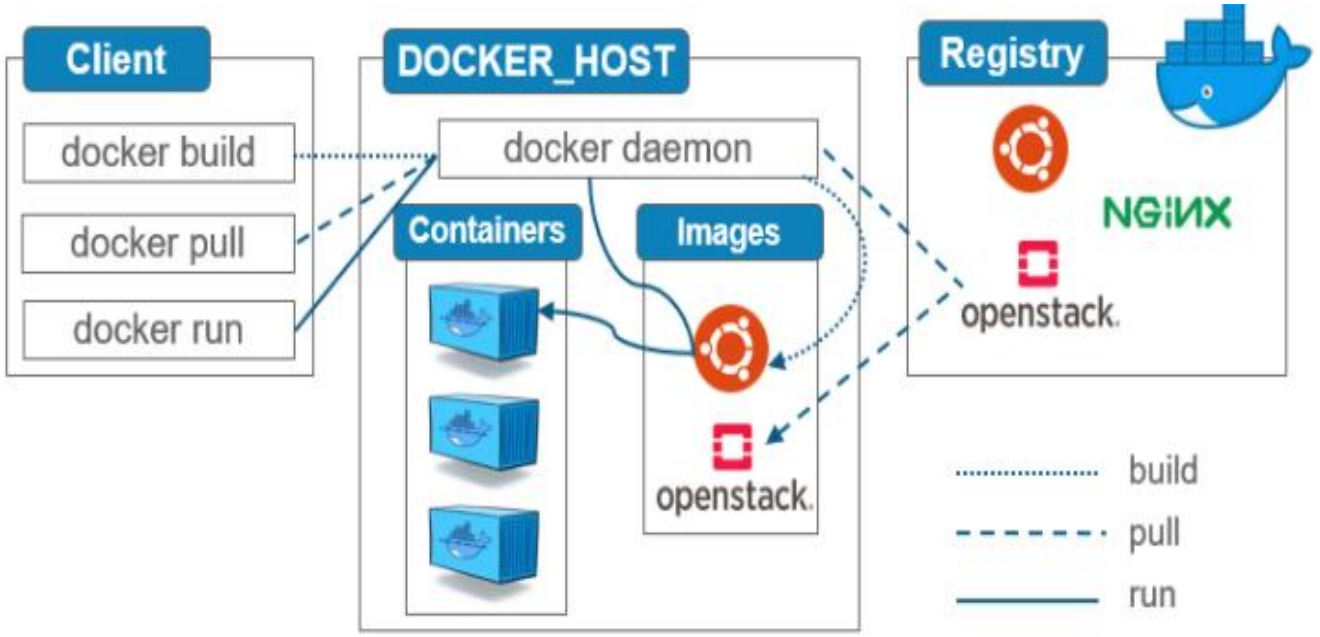
- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Develop application(s)
  - Distribute & test
  - Deploy into production environment, as a container or an orchestrated service.
- Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Docker Architecture



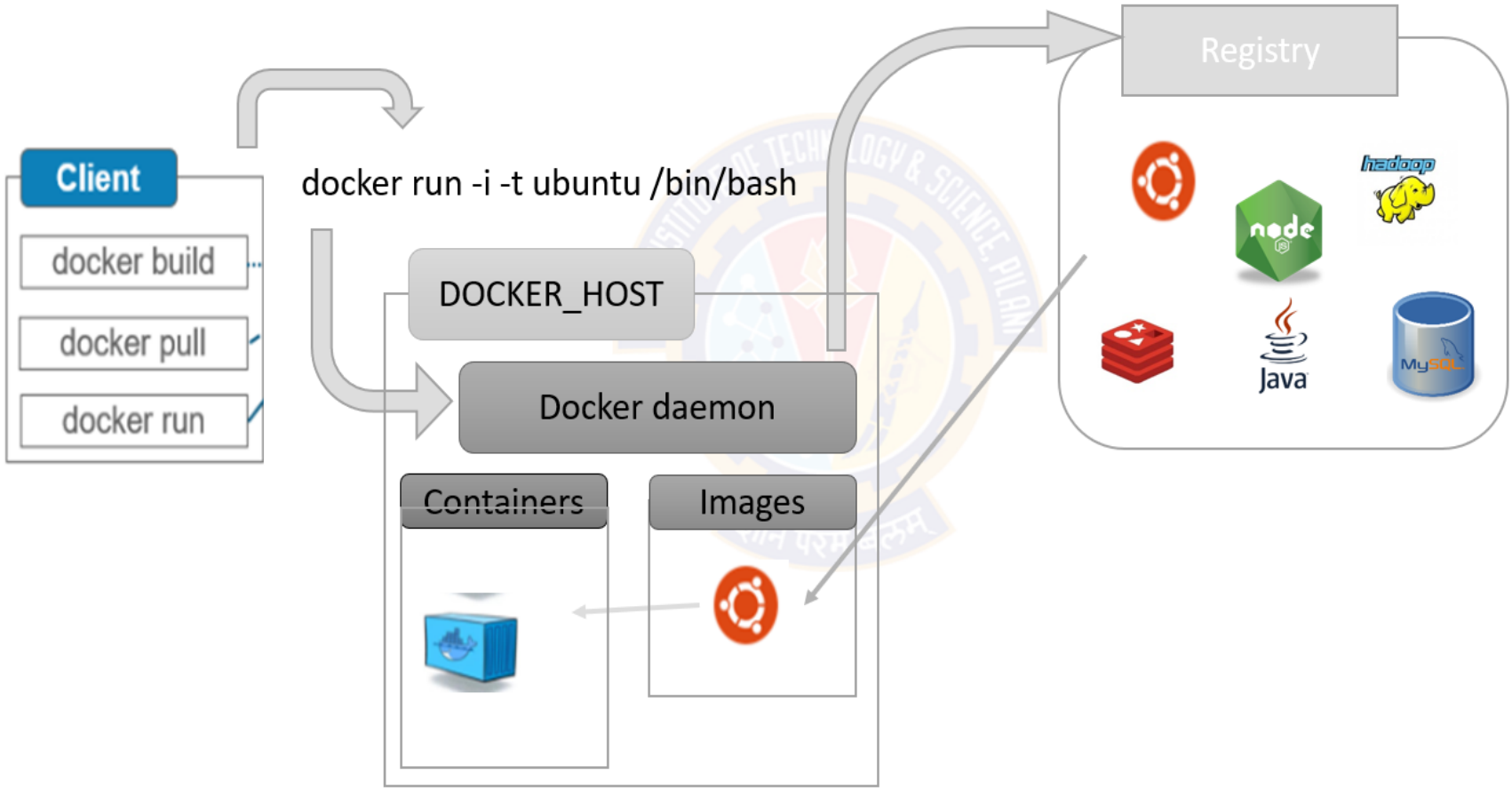
- Docker uses a client-server architecture.
- The Docker daemon
- The Docker client
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Docker Architecture



- **Docker Registries**
  - Stores Docker images.
  - Docker Hub is a public registry that anyone can use.
  - Docker is configured to look for images on Docker Hub by default
- **Docker Objects**
  - IMAGES: An *image* is a read-only template with instructions for creating a Docker container.
  - CONTAINERS: A container is a runnable instance of an image

Running a Docker Container



Docker Commands

- A container is a runtime instance of a docker image
- Create and run a container from an image, with a custom name:

  docker run –name <container name> <image name>

  docker run –name mylinuxserver Ubuntu
- Run a container with and publish a container's port(s) to the host.

  docker run -p <host port>:<container port> <image name>

  docker run –p 8080:80 nginx
- Run a container in the background

  docker run –d <image name>

  docker run –d –p 8080:80 nginx

Docker Commands

- Start or stop an existing container:
  docker start/stop <container name> (or <container id>)
  docker stop 11ed (or mynginx)

- Remove a stopped container:
  docker rm <container name> (or <container id>)
  docker rm –f 11ed (or mynginx)

- Open a shell inside a running container:
  docker exec -it <container name> sh
  docker exec –it myubuntu bash

Docker Commands

- Fetch and follow the logs of a container: ***docker logs -f <container name>***
- To inspect a running container: ***docker inspect <container id> (or) <container name>***
- To list currently running containers: ***docker ps***
- List all docker containers (running and stopped): ***docker ps --all***
- View resource usage stats: ***docker container stats***

Docker Image

- A lightweight, standalone, executable package of software
- includes
  code,
  runtime,
  system tools,
  system libraries

- Build an Image from a Dockerfile: ***docker build -t <image name>***
- List local images: ***docker images ls***
- Delete an Image: ***docker rmi <image name>***
- Remove all unused images: ***docker image prune***

Build & Run Customized Image

Dockerfile: File with instructions to build a docker container image

```
FROM ubuntu:latest
RUN mkdir /app
RUN apt update
RUN apt install vim g++ -y
WORKDIR /app
ENTRYPOINT ["/bin/bash"]
```

Hands On

```
docker build -t myappimage .
docker run -it myappimage
```

Container Storage

- The container's filesystem
- Each container also gets its own "scratch space" to create/update/remove files.
- Any changes won't be seen in another container, *even if* they are using the same image
- Docker containers use the following for persistent storage of data

1. **Volumes**
2. **Bind mounts**

Container Storage - Volume

- ***Preferred mechanism*** to ***store persisting data*** generated by and used by containers
- Managed by docker itself.
- The data inside volume is not stored in the container's file system, but stored in the host m/c's filesystem.
- Can be more safely shared among multiple containers.
- Volumes provide the ability to connect specific filesystem paths of the container back to the host machine
- Volume path on Linux: /var/lib/docker/volumes/

Container Storage - Volume

Create a volume by using the docker volume create command

docker volume create mydb

```
docker volume create testvol
estvol
node1] (local) root@192.168.0.13 ~
ls -lrt /var/lib/docker/volumes/
otal 24
rw-------    1 root     root        8,  16 Feb  9 07:15 backingFsBlockDev
rwx-----x    3 root     root            19 Feb  9 08:03 testvol
```

Start the container with mount

docker run -it --mount type=volume,src=mydb,target=/etc/myappdb ubuntu

docker volume inspect **mydb**

```
$ docker volume inspect mydb
[
    {
        "CreatedAt": "2025-02-09T08:06:30Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/mydb/_data",
        "Name": "mydb",
        "Options": null,
        "Scope": "local"
    }
]
```

Container Storage – Bind mount

- Share a directory from the host's filesystem into the container.
- The container sees the changes you make to the code immediately, as soon as you save a file.
- ***docker run -it --mount type=bind,src="$(pwd)",target=/src ubuntu bash***
- The --***mount*** option tells Docker to create a bind mount
- ***src*** is the current working directory on your host machine (getting-started/app)
- ***target*** is where that directory should appear inside the container (/***src***)

Bind mounts are dependent on the directory structure and OS of the host machine

Hands On

Docker Compose

Run multiple container services together

Create deployment file - .yaml/.yml

Include details about individual services to run. E.g service name, image, ports/network, volume

Bring up services

docker-compose up

docker-compose up –d (Detached mode)

docker-compose –verbose up

Attach to individual service

docker attach <container name>

Status of services

docker-compose ps --all

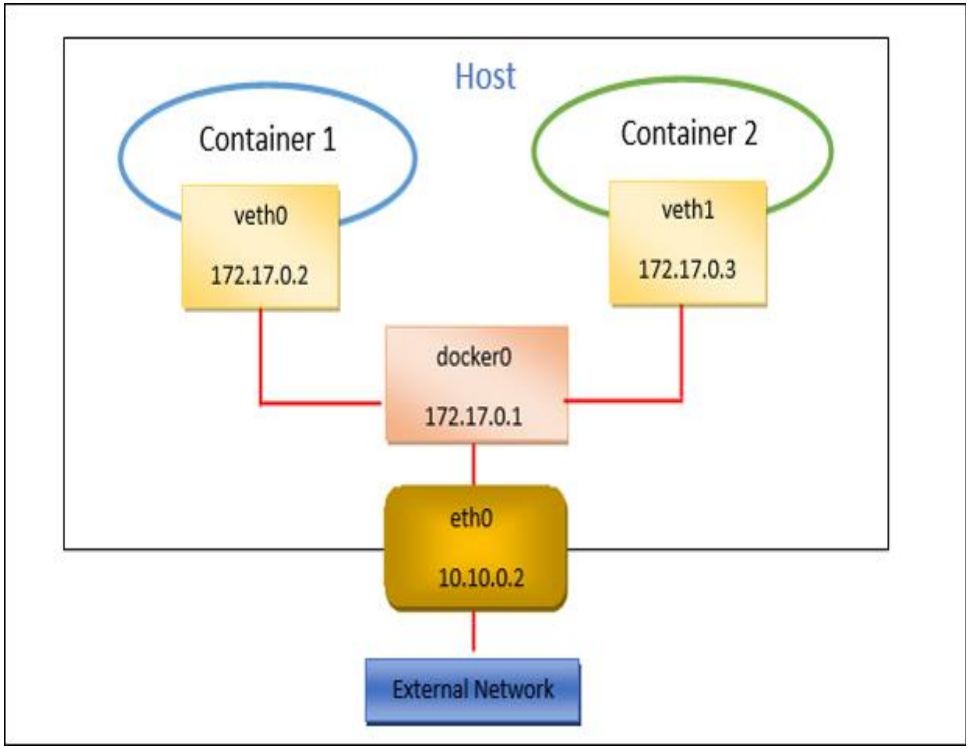
Bring down services

docker-compose down

Docker Networking

- Docker containers and services are powerful
- Connect them together, or connect them to non-Docker workloads.
- Platform-agnostic way
- Docker's networking subsystem is pluggable, using drivers.
- Provide core networking functionality
  - Bridge
  - Host
  - Overlay

Network Drivers - Bridge



- The default network driver.
- **Standalone containers that need to communicate inside a single host.**

Network Drivers - Bridge

```
docker network ls                                   #command to list all the
networks(drivers) in Docker
Docker network create mynet
docker run -dit --rm --name alpine1 alpine bash     #command to run a
container
docker run -dit --rm --name alpine2 alpine bash
docker ps                                           #list all the containers
docker network inspect bridge                       #details of bridge network driver
docker attach alpine1                               #attach to the container (get
console)
$ip addr show
$ping -c 2 google.com
$ping -c 2 alpine2
docker stop alpine1 alpine2                         #to stop the containers
```

Network Drivers - Bridge

docker attach alpine1
ping -c 2 alpine2
ping -c 2 alpine3 (not pingable)

Detach from alpine1 using detach sequence, CTRL + p CTRL + q (hold down CTRL and type p followed by q).
ping -c 2 google.com
docker stop alpine1 alpine2 alpine3 alpine4#stop and remove containers
docker rm alpine1 alpine2 alpine3 alpine4
docker network rm mynet   #remove user-defined bridge network

Hands On

Network Drivers - Host

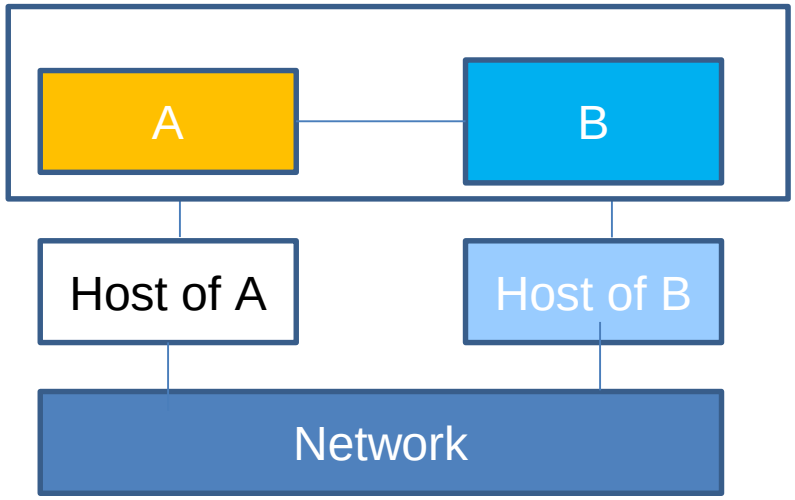
- From a networking point of view, as if the container process were running directly on the Docker host and not in a container.
- However, in all other ways, such as storage, process namespace, and user namespace, the container process is isolated from the host.
- **The host networking driver only works on Linux hosts**, and is not supported on Docker Desktop for Mac, Docker Desktop for Windows, or Docker EE for Windows Server.

Example

docker run --rm -d --network host --name mynginx nginx
sudo netstat -tulpn | grep :80 #Verify which process is bound to port 80
Also curl <ip>:80 / ip on browser
docker container stop mynginx

This procedure requires port 80 to be available on the Docker host.

Network Drivers - Overlay



- Overlay networks connect multiple Docker daemons running in multiple hosts together
- Enable swarm services to communicate with each other.
- You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons.

Summary

- Introduction to Containers - Lightweight Virtualization
- Key Building blocks
  - Namespaces
    - Provide isolation
    - 6 key namespaces: Mount, UTS, IPC, PID, Network, User
  - Cgroups
    - **Resource limiting, Prioritization**
    - CPU, memory
- Containers Vs Virtual Machines
- Docker Containers
  - Docker Architecture: Client-server, Objects, Registry
  - Building Customized image – Dockerfile
  - Docker Volume – volume, bind-mount
  - Docker Compose – Running multiple services/containers
  - Docker Networking

Lab - TCP Chat on Docker Container

You can use https://labs.play-with-docker.com/ if you don't have docker locally.

Create two separate programs TCP Client and Server in your favorite programming language – C/C++/Python

Refer the programs from Tutorial topic, if you need them.

Create custom image for each program with respective dockerfile

Execute them individually on two separate docker containers. Remember client has to know where the server is running – Which IP and Port ?

Now create docker-compose with both the services specified in a single docker-compose.yml

Observe the difference in events/steps executed between individual running of docker container and using docker-compose

Lab - Web Service and DB on Docker

1. Deploy nginx web service on docker on your PC or via https://labs.play-with-docker.com/
    1. Once it is executing, try accessing it using curl <ip>:port from terminal or if you have browser locally use localhost:port
2. Deploy mysql DB on docker container
    1. Show all the steps you use to get the successful deployment and executing mysql DB on the container.
    2. Inside the docker container shell, access the mysql DB using mysql CLI client (mysql –p <username>)
    3. Verify the data persistency using Volumes and bind mounts by creating entries in a table inside the DB. Compare the execution without using Volume/bind mount

Capture all your observations and discuss with peers and instructor.

References

- Namespaces in operation, part 1: namespaces overview [LWN.net]
- Chapter 1. Introduction to Linux Containers | Red Hat Product Documentation
- Docker Engine | Docker Docs

Additional Slides

Network Drivers - Overlay

Create the swarm

On manager. initialize the swarm.

```
docker swarm init
docker swarm join --token <TOKEN> \
 --advertise-addr <IP-ADDRESS-OF-WORKER-1> \        #optional
 <IP-ADDRESS-OF-MANAGER>:2377

docker node ls                          #on manager
docker network ls
```

The docker_gwbridge connects the ingress network to the Docker host's network interface so that traffic can flow to and from swarm managers and workers.