BITS Pilani

Cloud Computing

Session 4

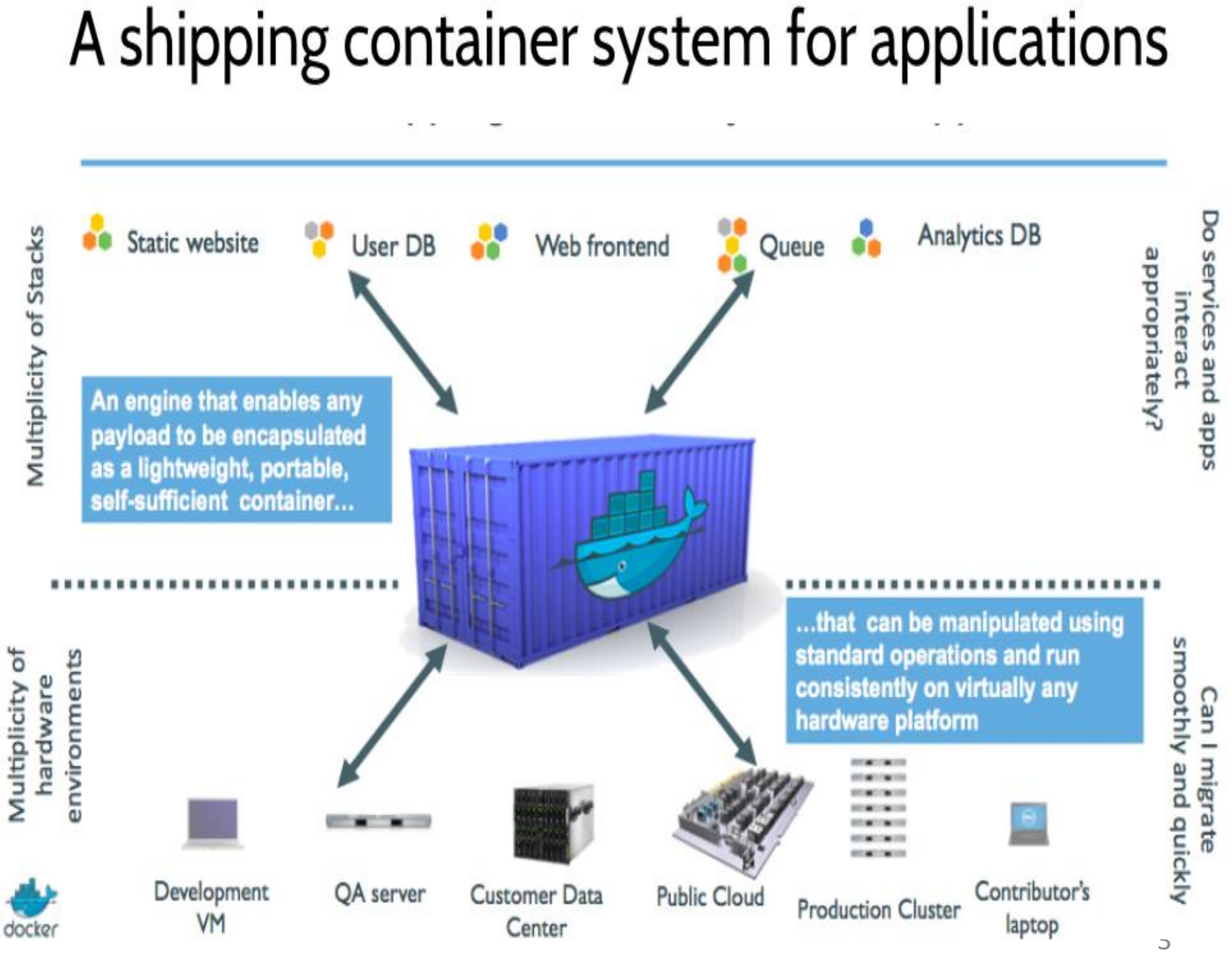
Containers

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

Agenda

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

What are Containers?



A shipping container system for applications

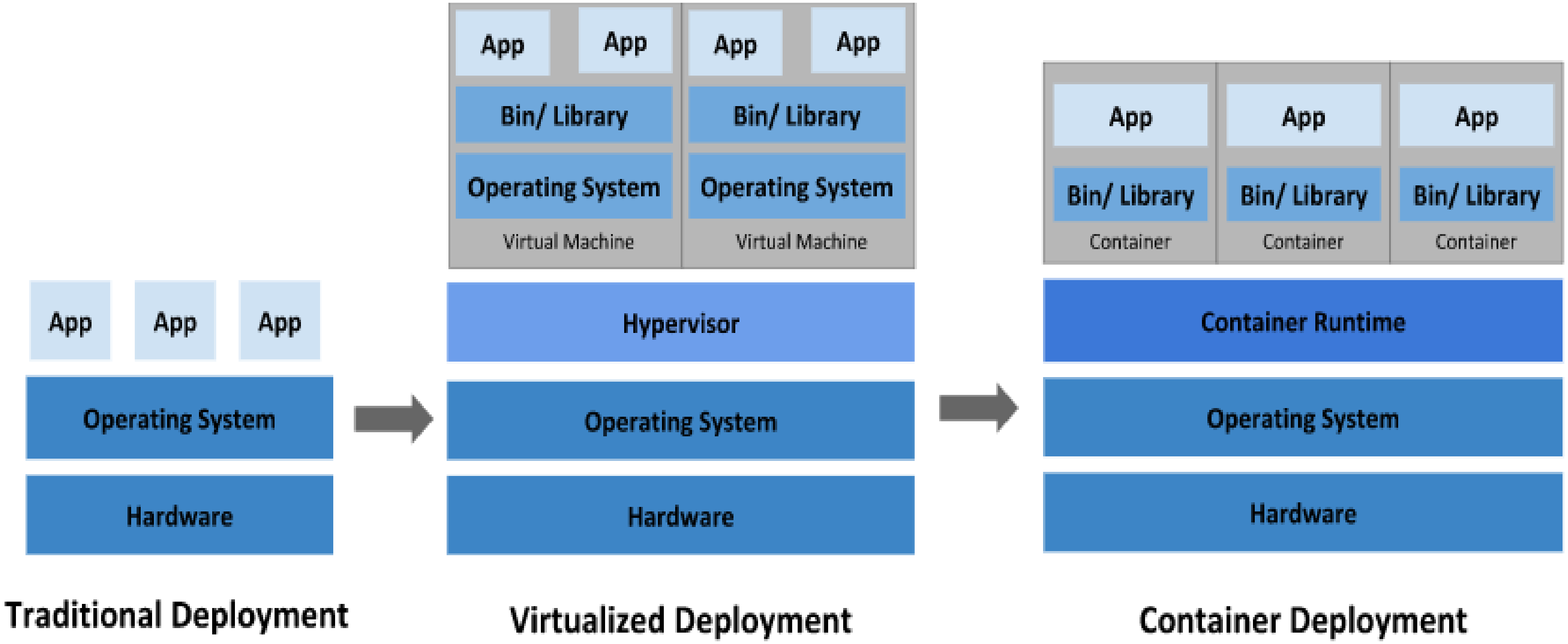
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container...

...that can be manipulated using standard operations and run consistently on virtually any hardware platform

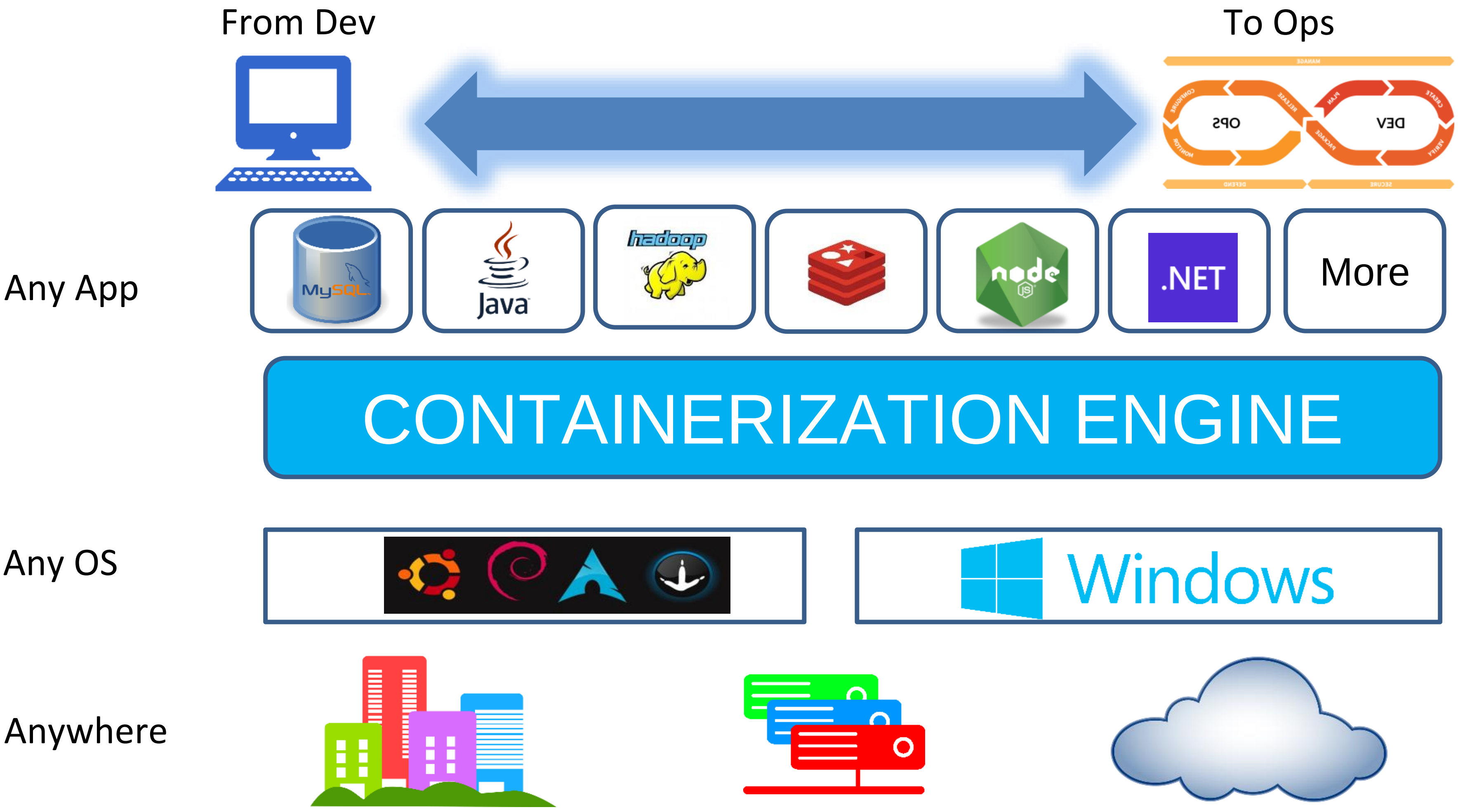
What are Containers ?

- 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?



From Dev

To Ops

Any App

More

CONTAINERIZATION ENGINE

Any OS

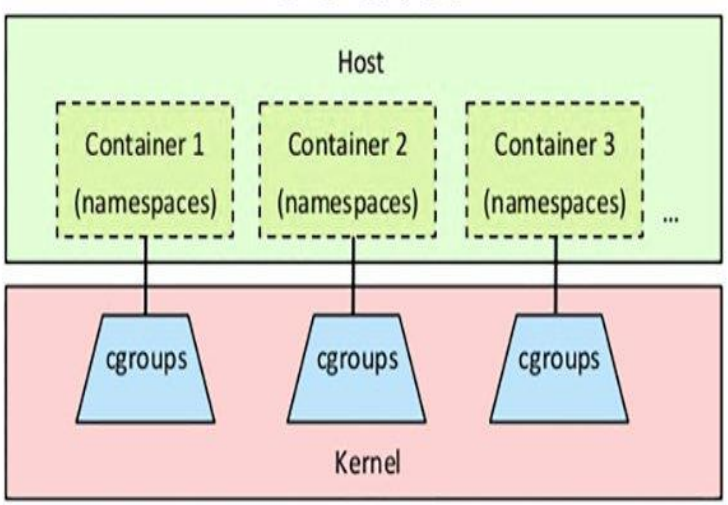
Windows

Anywhere

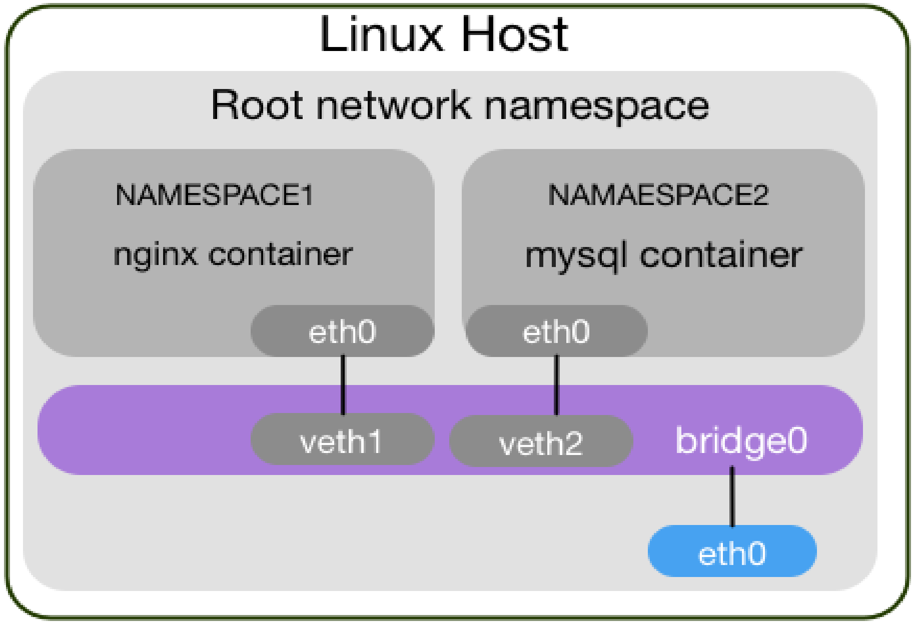
Containers

Containers are powered by two underlying Linux Kernel technologies

- Namespaces
- Cgroups



Namespaces

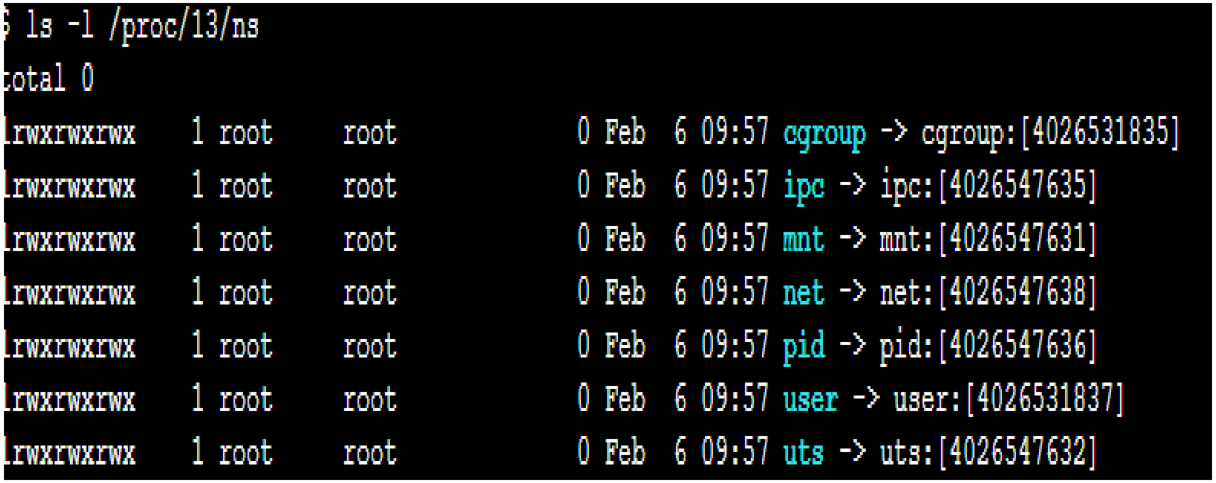


- Kernel mechanism for limiting the visibility that a group of processes has of the rest of a system
- Namespace merged into Linux 3.8
- Limit Visibility
  - Process trees - PIDs
  - Network interfaces
  - User IDs
  - Filesystem mounts

[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]](https://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]

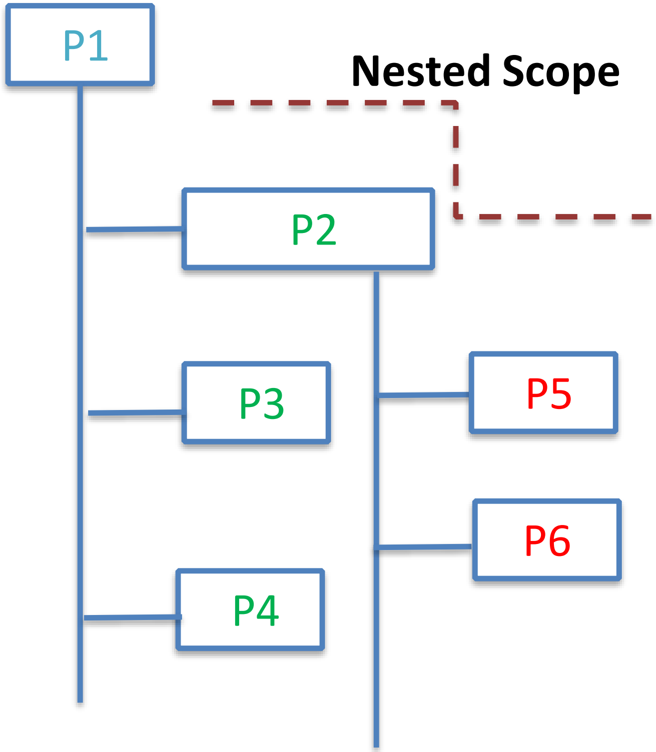
IPC Namespace

- 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



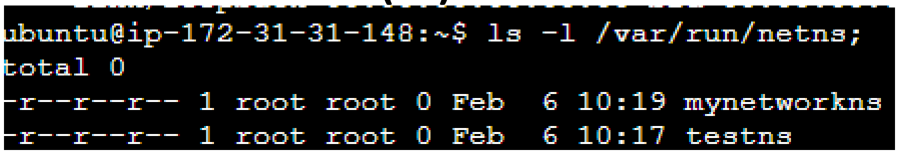
[Namespaces in operation, part 1: namespaces overview [LWN.net]]

[PID namespaces in the 2.6.24 kernel [LWN.net]]

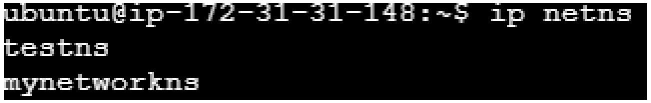
Network Namespace

- 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)

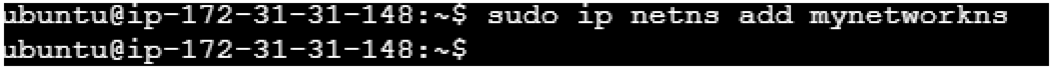


```
ubuntu@ip-172-31-31-148:~$ ls -l /var/run/netns;
total 0
-r--r--r-- 1 root root 0 Feb  6 10:19 mynetworkns
-r--r--r-- 1 root root 0 Feb  6 10:17 testns
```



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

Adding network ns

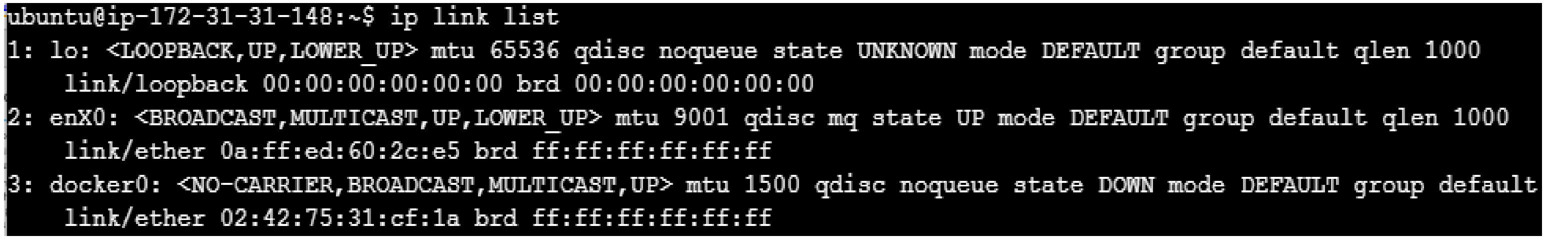


```
ubuntu@ip-172-31-31-148:~$ sudo ip netns add mynetworkns
ubuntu@ip-172-31-31-148:~$
```

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

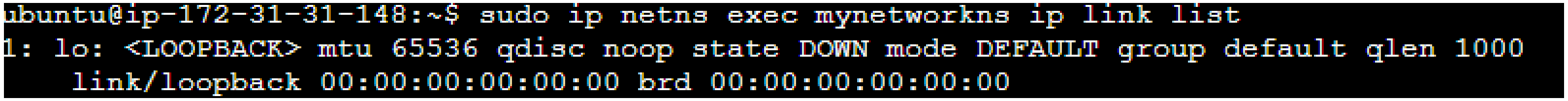
Network Namespace

Network interfaces on host

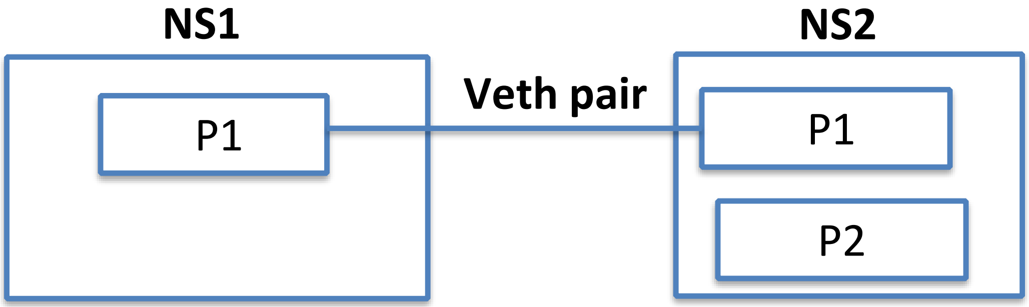


```
ubuntu@ip-172-31-31-148:~$ ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enX0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP mode DEFAULT group default 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

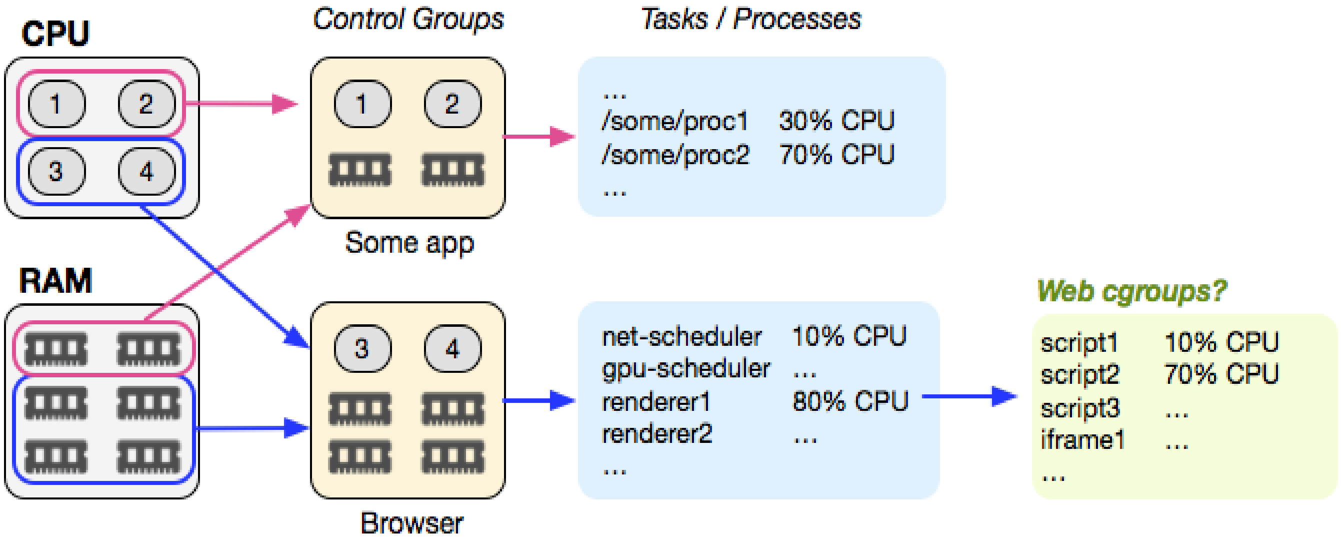
- Isolate the user and group ID number spaces
- A process's user and group IDs can be different inside and outside a user namespace.
  - A process has full root privileges for operations inside the user namespace,
  - But is unprivileged for operations outside the namespace.

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

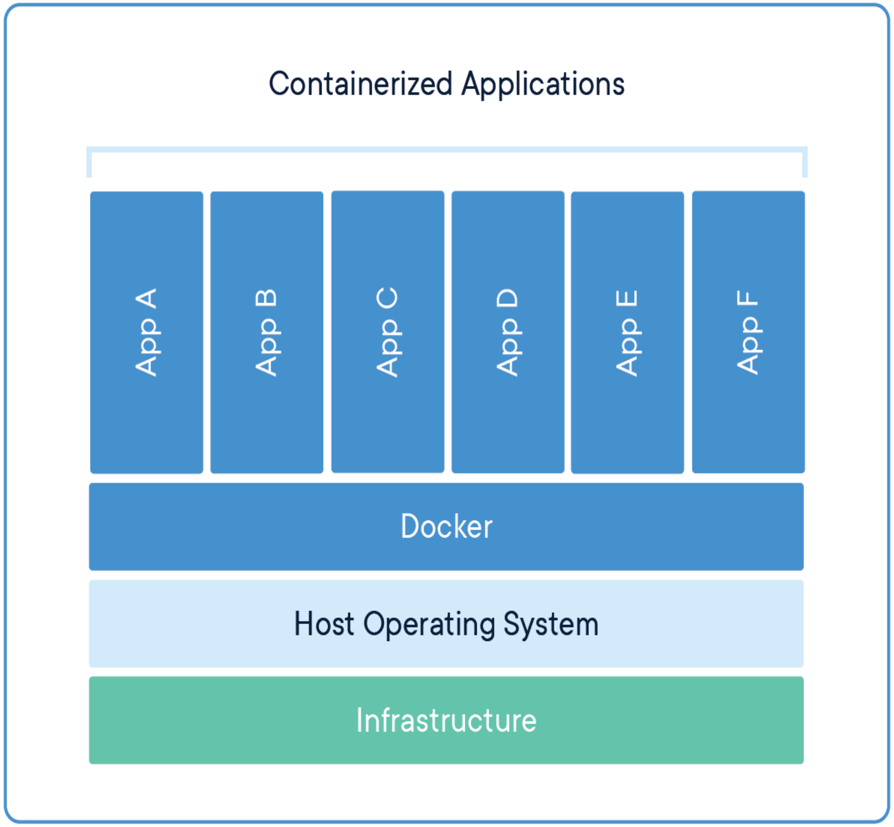


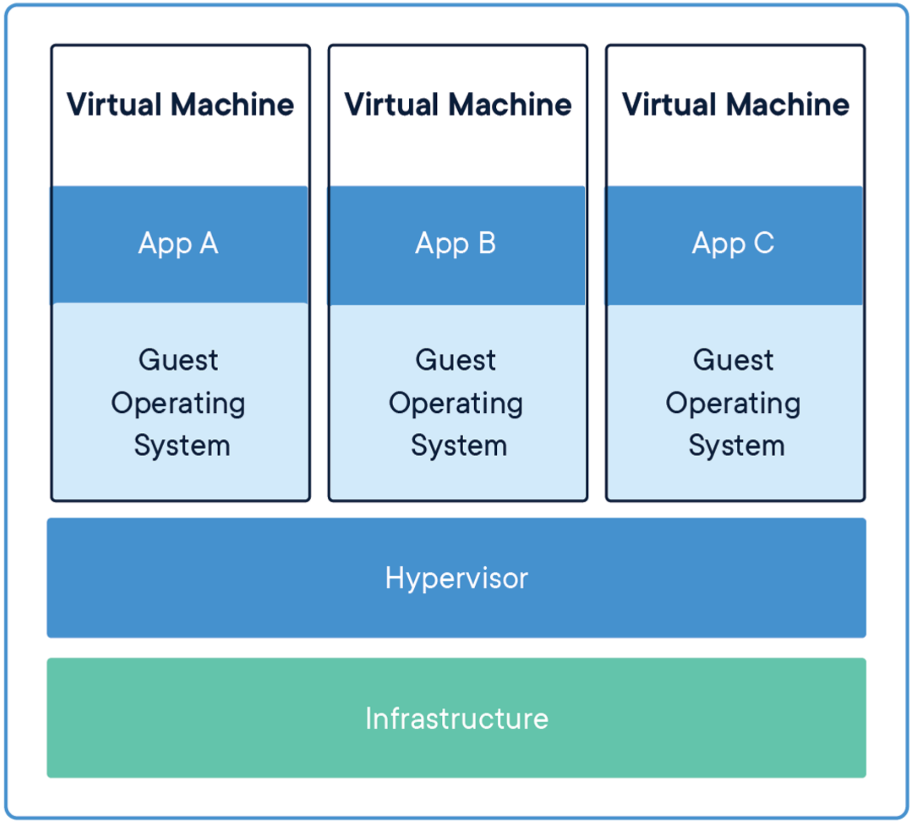
cgroups – Control groups

- 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

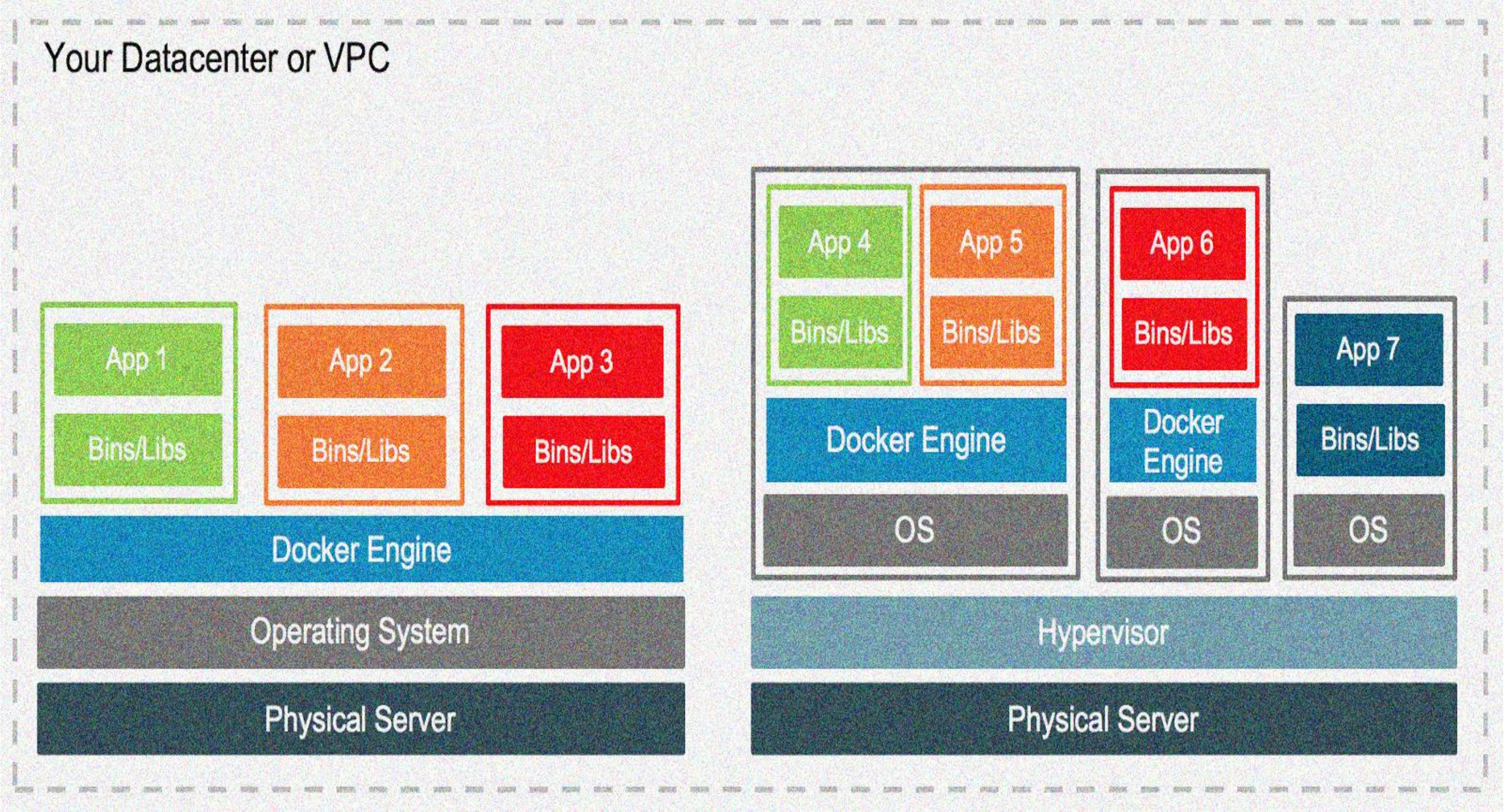
Cgroup merged into Linux 2.6.24

Containers Vs Virtual Machines





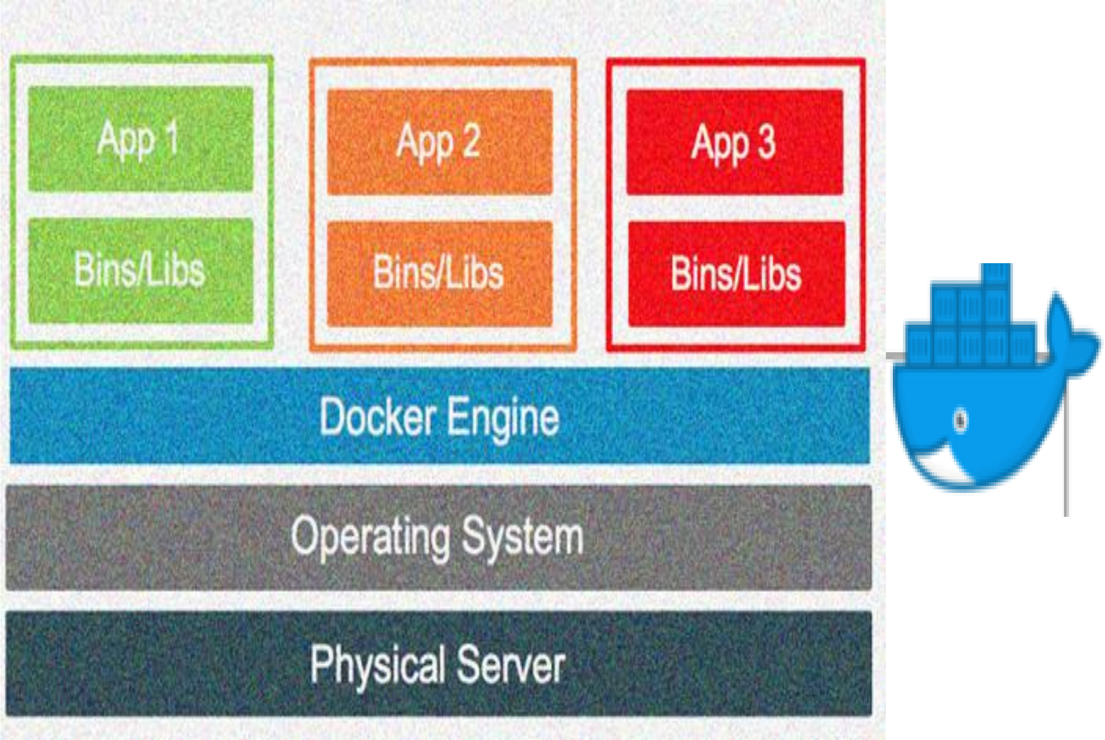
Containers on Virtual Machines ?



Docker

Docker Platform

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

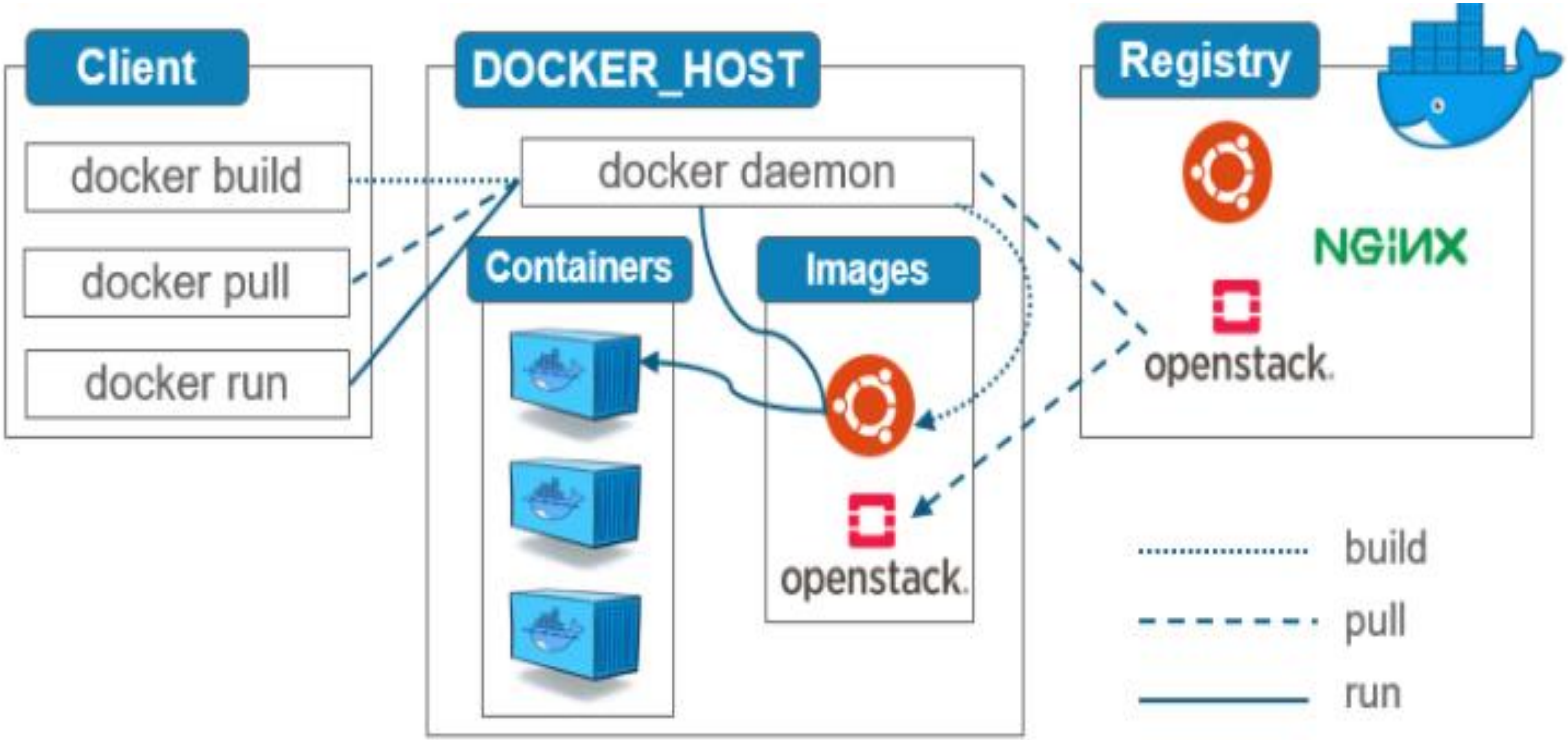


- A single host can run many containers simultaneously
- Containers are lightweight and contain everything needed to run the application

Docker Platform

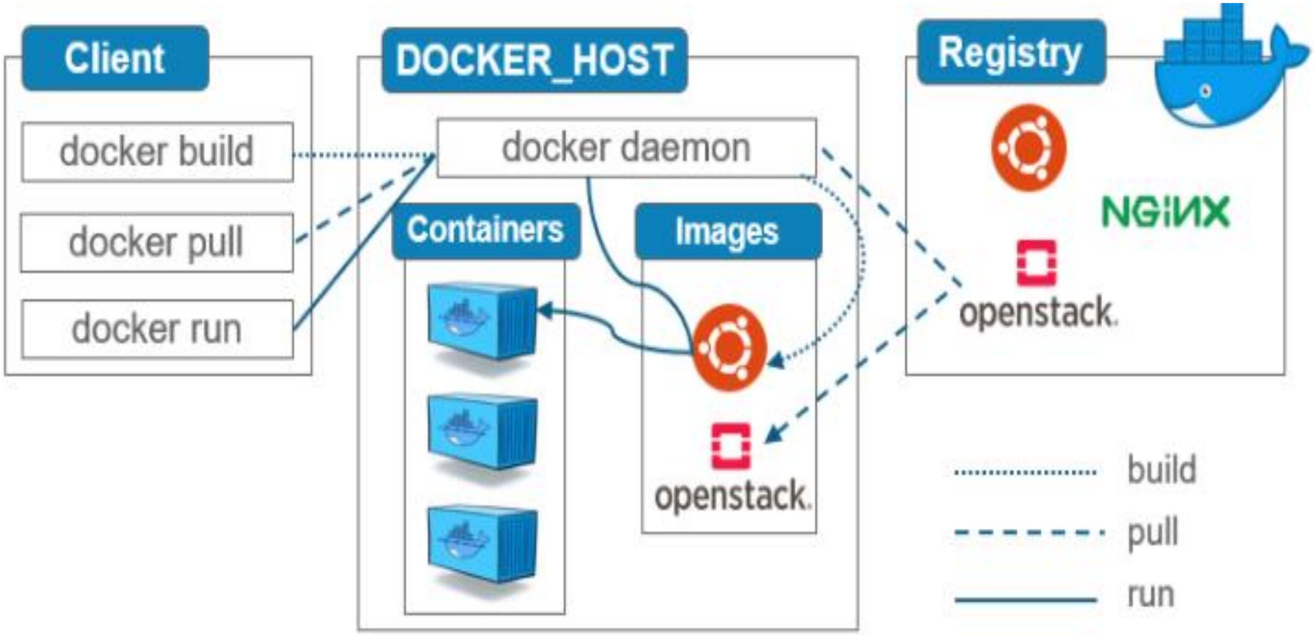
- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Develop application(s)
  - Distribute & test
  - Deploy into production environment, as a container or an orchestrated service.
- Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

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.

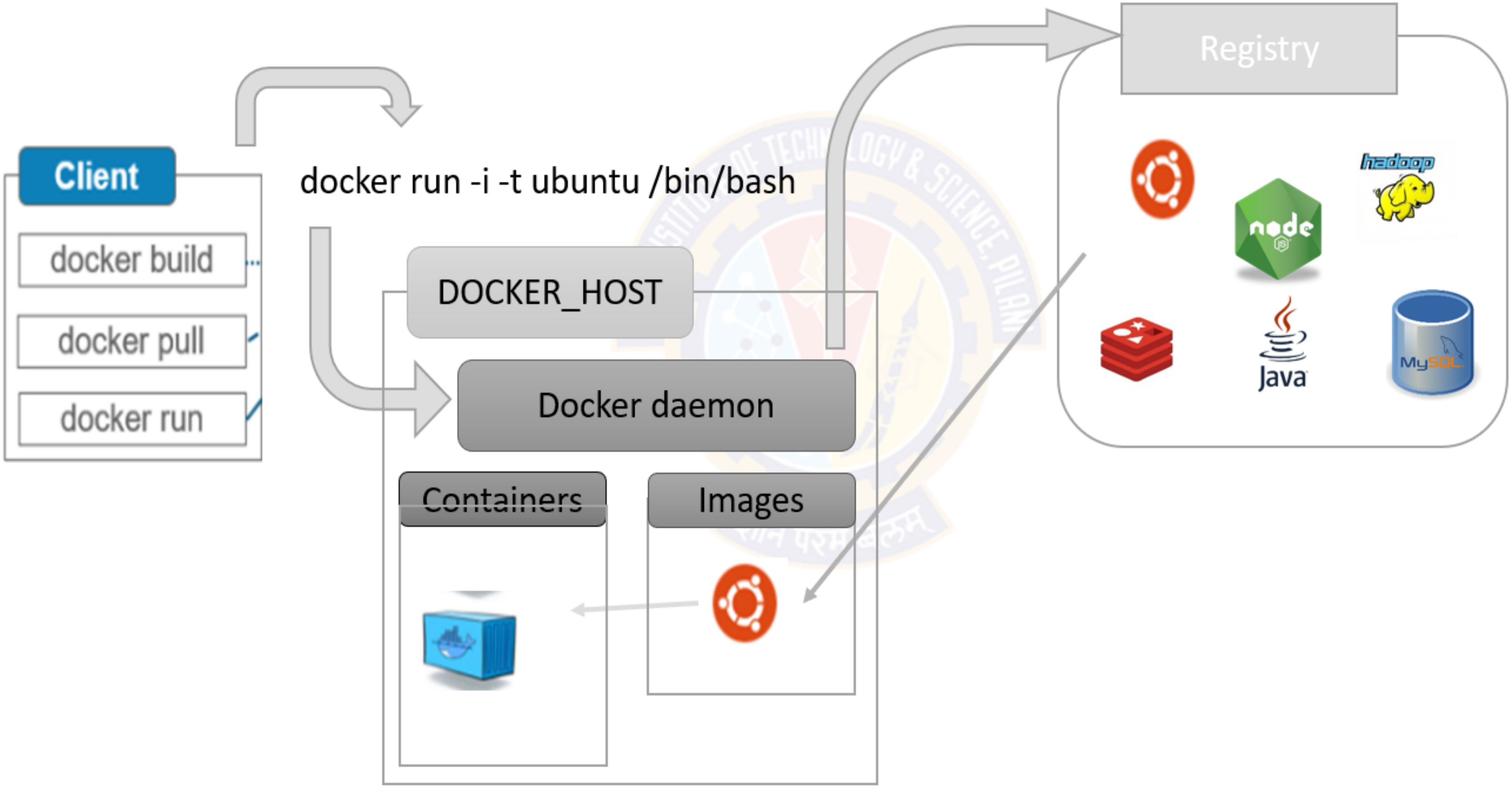
Docker Architecture



- **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
```

Summary – So Far

- 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

Lab - Web Service 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

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
- The Resilience of Virtual Machines: Why VMs Are Still Vital in the Age of Docker Dominance | Effective Programmer | Effective Programmer