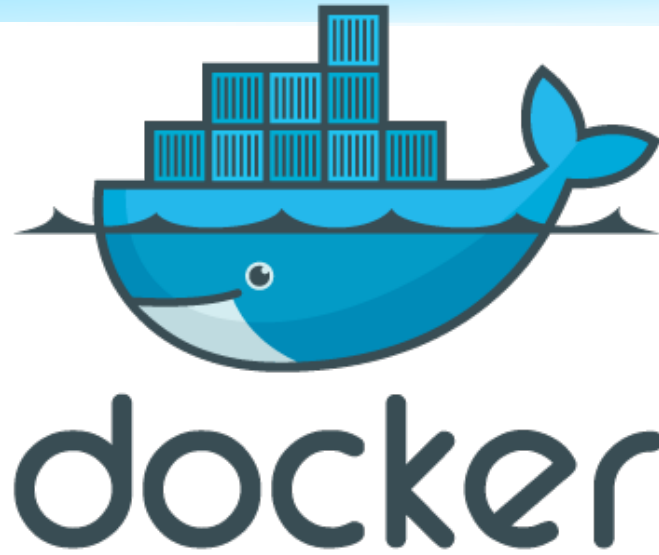


Docker Basic



Outline

- Background
- Overview
- Objects
- Docker Installation
- Basic Usage

Outline

- Background
 - cgroups
 - namespace
- Overview
- Objects
- Docker Installation
- Basic Usage

cgroups (control groups)

- A Linux kernel feature that limits, accounts for and isolates **resource** usage of collection of processes
- Resources
 - CPU
 - Memory
 - I/O
 - Network Device
 - Etc
- Functionality of cgroup
 - Resource limiting
 - Prioritization
 - Accounting
 - Control

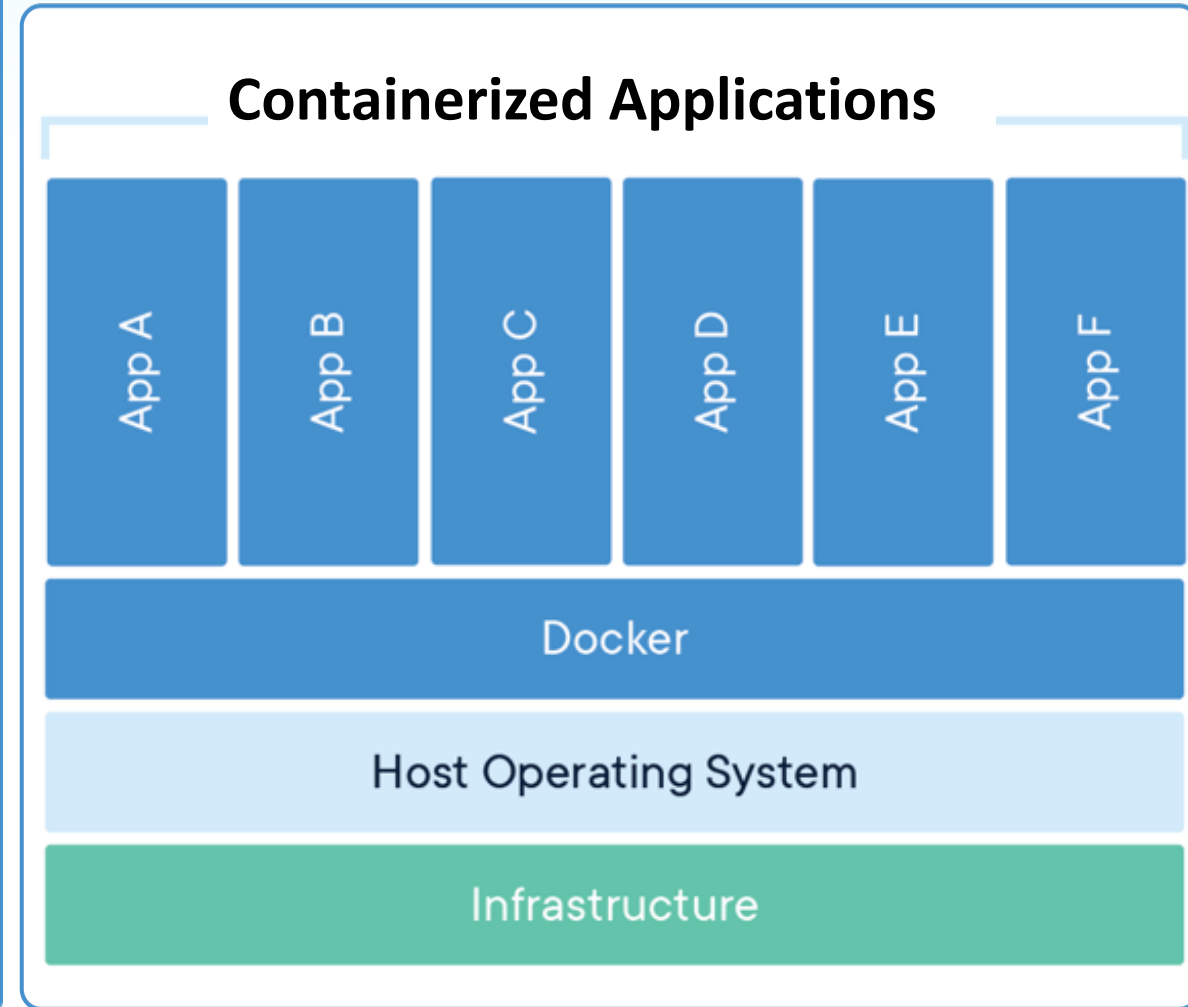
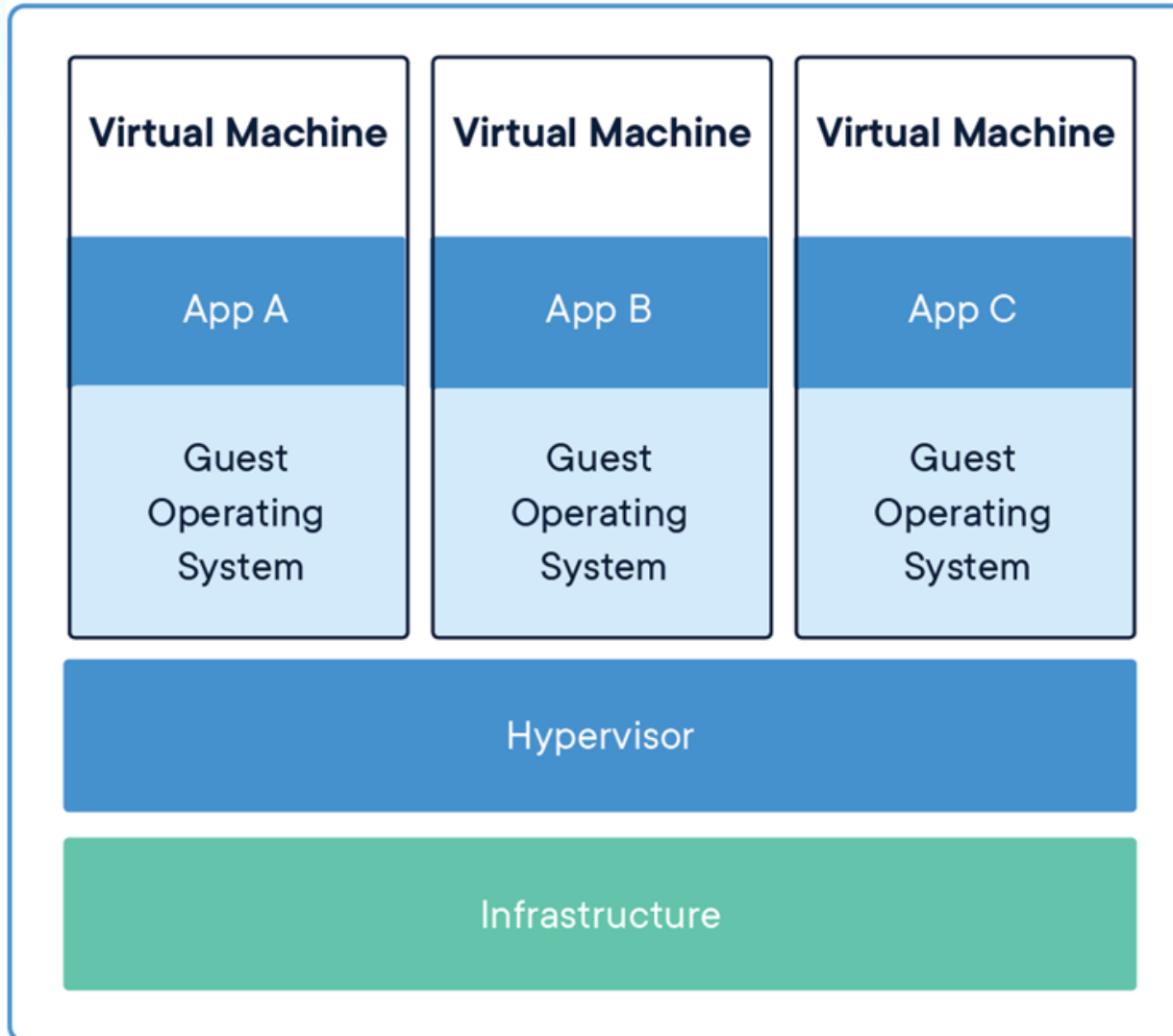
Namespace

- A Linux kernel feature that partitions kernel **resources usage** of collection of processes
- Namespace kinds
 - Mount: File System mount point
 - Network: Interface, ip address, iptables, route, etc
 - UTS: Hostname and NIS domain name
 - Process ID: Process IDs
 - User: User and group IDs
 - IPC: Inter Process Communication
- Linux System starts out with a single namespace of each type
- Processes can create namespaces and join different namespaces

Outline

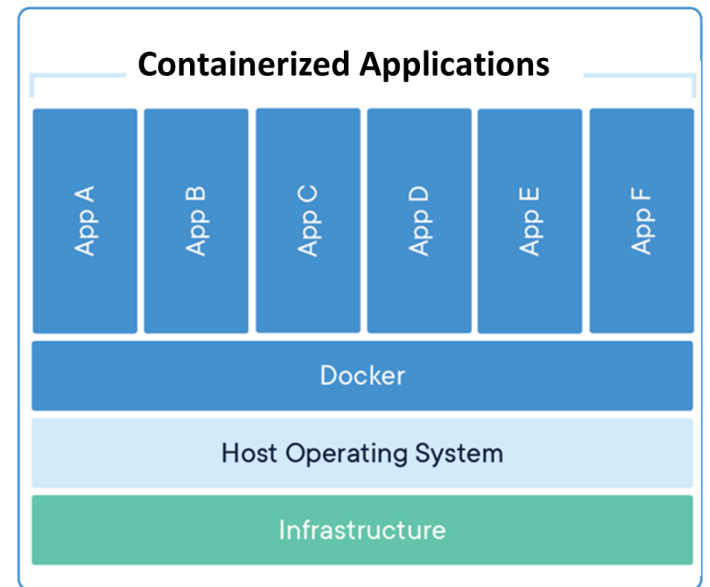
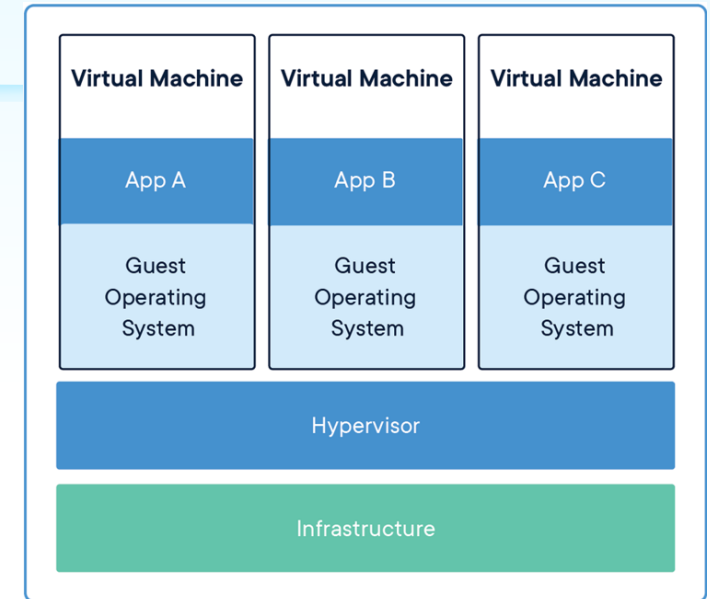
- Background
- Overview
- Objects
- Docker Installation
- Basic Usage

Virtual Machines and Containers



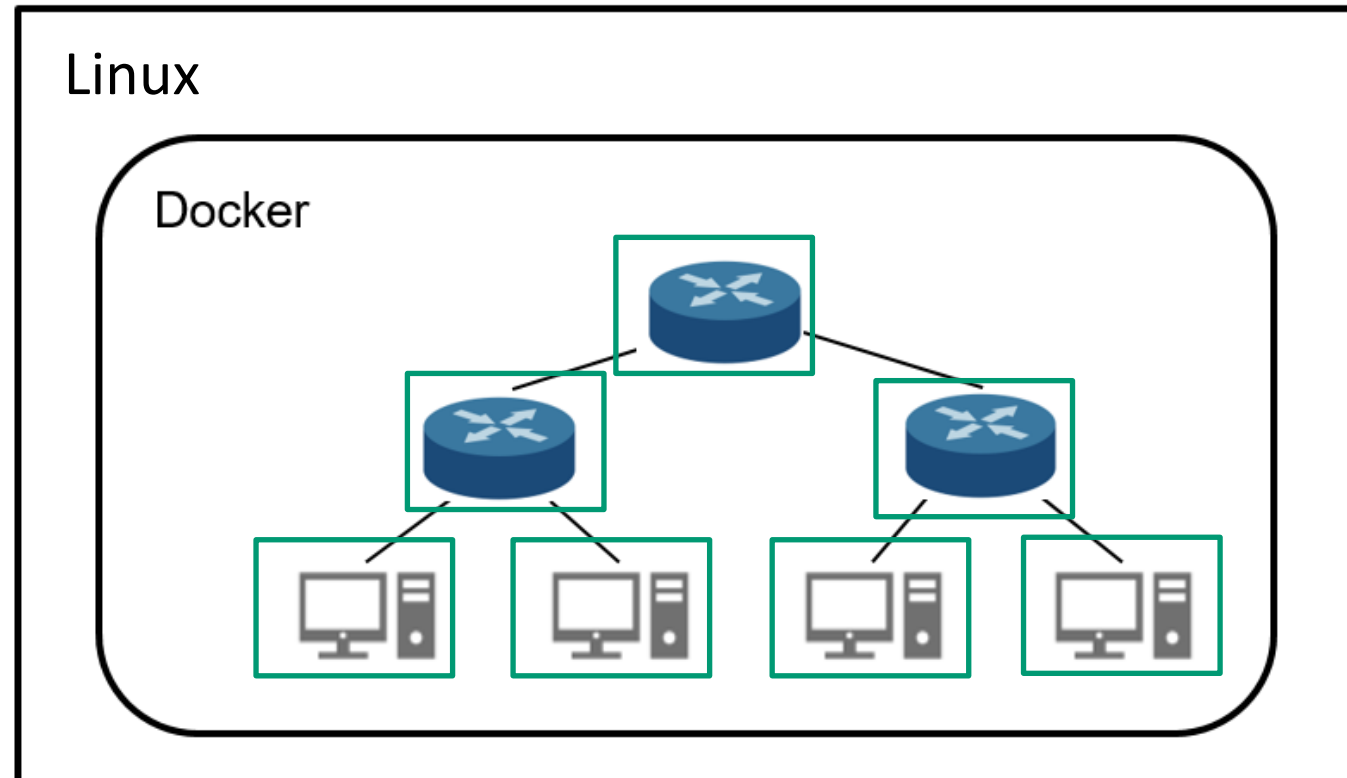
VMs vs. Containers

- VM
 - Runs a unique guest operating system
 - **Hypervisor** creates and runs VMs
- Containers
 - Run directly on the host OS.
 - Utilizing Linux kernel functions
 - **namespace**, chroot (change root directory)
 - Process isolation
 - **cgroup** (control groups)
 - Physical resource limitation
 - **Container engine** hosts the containers
 - Docker is a popular container engine



Overview

- Docker
 - A software platform that allows user to build, test and deploy applications in packages called containers
- Example Environment

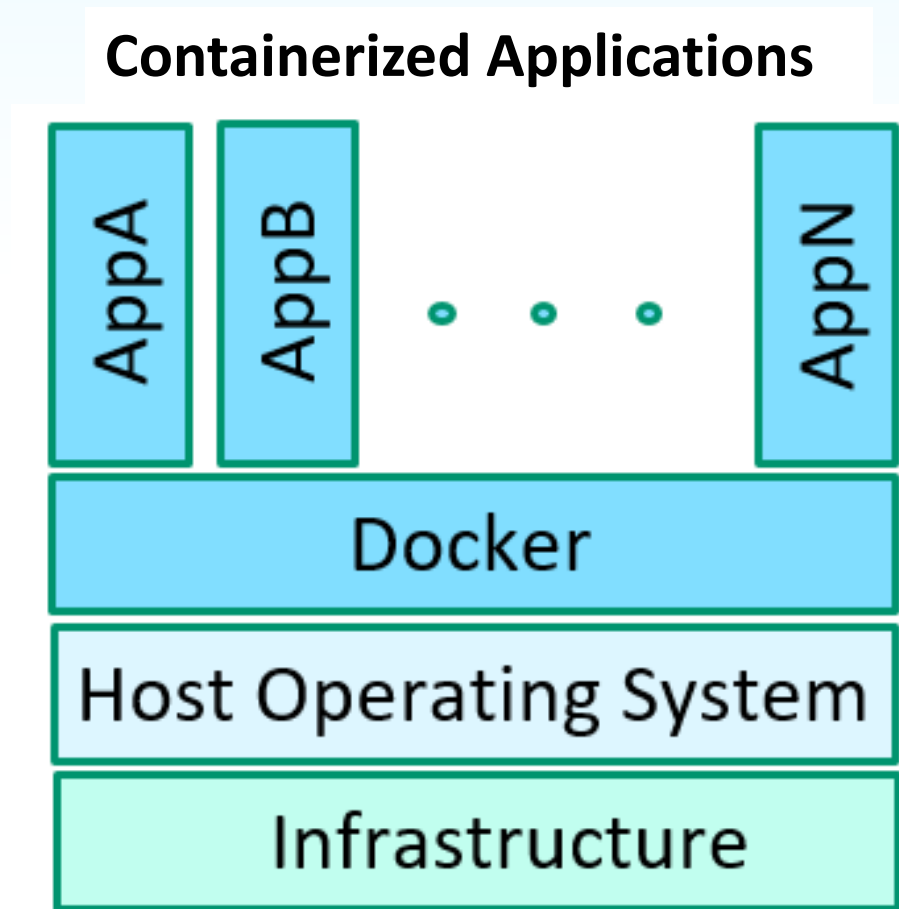


Outline

- Background
- Overview
- **Objects**
- Docker Installation
- Basic Usage

Docker Objects

- Docker Image
 - A read-only template with for creating a Docker container
 - Could be based on another image, with some additional customization
- Docker Containers
 - A runnable instance of a Docker image
 - Use **cgroups** and **namespace** to implement resource isolation



Steps for Creating Docker Containers

1. **Build Docker images of desired OS distribution and dependency of applications**
2. **Store Docker images in a Docker Registry**
 - Public Registry
 - Docker Hub, default
 - Private Registry
 - Local host, Private image server, or ...
3. **Run Docker to build containers of images**

Outline

- Background
- Overview
- Object
- Installation
- Basic Usage

Install Docker on Ubuntu

- Update apt

```
bash$ sudo apt-get update && sudo apt-get upgrade -y
```

- Install curl for data transfer

```
bash$ sudo apt-get install -y curl
```

- Download and run a **Docker installation script** `get-docker.sh`

```
bash$ sudo curl -ssl https://get.docker.com | sh
```

- -ssl use SSL for the connection

- Pipe `get-docker.sh` script to shell

Permission Setup

- Need root privilege to run Docker command
- To create a supplementary group for a user to run Docker without root privilege

1. Add a group named **docker**

```
bash $ sudo groupadd docker #Add a group named docker
```

2. Add a user into the docker group

```
bash $ sudo usermod -aG docker $USER
```

-a Add the user to the supplementary **group(s)**. Use only with the **-G** option.

-G, --groups *GROUP1[,GROUP2,...[,GROUPN]]*

- remove the user from a supplementary group if the group not listed

(-a appends the user to the current supplementary group list)

3. Reboot your operating system to enable new configuration

Outline

- Background
- Overview
- Object
- Installation
- Basic Usage
 - Pull Docker image
 - Build Docker image
 - Docker run
 - Docker exec
 - Connect container with veth

Pull Docker Image

- Usage

```
bash$ sudo docker pull [NAME]:[TAG]
```

- E.g., pull ubuntu:18.04 from Docker Hub Registry

```
bash$ sudo docker pull ubuntu:18.04
```

- List images to show pull result

```
bash$ sudo docker images
```

```
jln@ubuntu:~$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
SIZE			
ubuntu	18.04	c090eaba6b94	2 weeks ago
63.3MB			

- Background
- Overview
- Object
- Docker Installation
- Basic Usage
 - Pull Docker image
 - Build Docker image
 - Create Dockerfile
 - Build image
 - Docker run
 - Docker exec
 - Connect container with veth

Create Dockerfile for an Image

- Create a txt file named Dockerfile, consisting of
 - A base image and
 - Custom instructions
- Example
 - Create a Dockerfile at directory ~/Desktop

```

1 FROM ubuntu:18.04
2
3 MAINTAINER jin
4
5 RUN apt-get update
6
7 RUN apt-get install iptables -y
8 RUN apt-get install iputils-ping -y
9 RUN apt-get install net-tools -y
10 RUN apt-get install iproute2 -y
11 RUN apt-get install tcpdump -y
12 RUN apt-get install vim -y
13 RUN apt-get install sudo -y
14 RUN apt-get install git -y
15 RUN apt-get install isc-dhcp-server -y
16 RUN apt-get install isc-dhcp-client -y
17 RUN apt-get install mininet -y

```

Base Image

Custom Commands

Build Image

- Usage

```
bash$ docker build -t [image_name] [directory of Dockerfile]
```

- Example

```
jln@ubuntu:~/Desktop$ sudo docker build -t test .
```

– Show Images to check build result

```
jln@ubuntu:~/Desktop$ sudo docker images
```

REPOSITORY SIZE	TAG	IMAGE ID	CREATED
test 268MB	latest	94fa41d40498	11 minutes ago
ubuntu 63.3MB	18.04	c090eaba6b94	2 weeks ago

- Background
- Overview
- Object
- Docker Installation
- Basic Usage
 - Pull Docker image
 - Build Docker image
 - Docker run
 - Docker exec
 - Connect container with veth

Docker run

- Run a command in a new container
- Usage

```
bash$ docker run [OPTIONS] [IMAGE:TAG] [COMMAND] [ARG..]
```

Create and Run a new container

A Command runs in the new container

- E.g., Create and Run a container “sample”

```
bash$ sudo docker run -d -it --name sample ubuntu:18.04
```

-it: Interactive process (like a shell)

--name: Assign a name to the container

-d: Detached (like a daemon in background)

List Containers

- List containers

```
bash$ sudo docker ps -a
```

-a: Show all containers

Container ID (a hashed value)

```
jin@ubuntu:~/Desktop$ sudo docker run -d -it --name sample ubuntu:18.04
[sudo] password for jin:
be95d7141f15663ff624d226f08c95a99f7946467ac36b24329cfe7cdf1bf517
```

```
jin@ubuntu:~/Desktop$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
be95d7141f15	ubuntu:18.04	"/bin/bash"	16 seconds ago
Up 13 seconds		sample	

- Background
- Overview
- Object
- Docker Installation
- Basic Usage
 - Pull Docker image
 - Build Docker image
 - Docker run
 - Docker exec
 - Connect container with veth

Docker exec

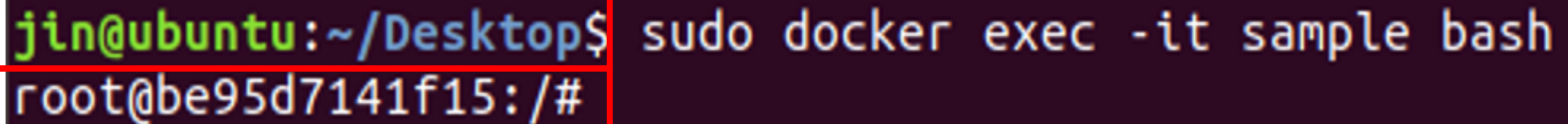
- Execute a command in a **running** container
- Usage

```
bash$ sudo docker exec [OPTIONS] [CONTAINER] [COMMAND]
```

- Exec *bash* shell in the running container “sample”

```
bash$ sudo docker exec -it sample bash
```

Shell in ubuntu host



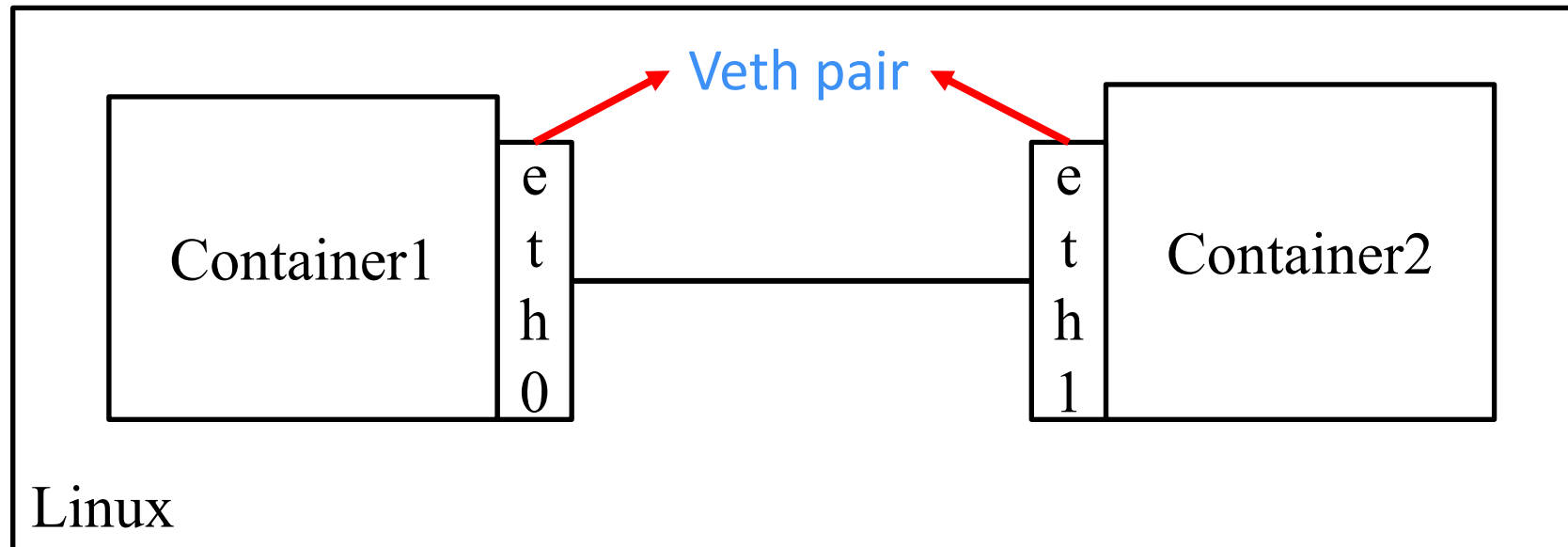
```
jin@ubuntu:~/Desktop$ sudo docker exec -it sample bash  
root@be95d7141f15:/#
```

Shell in sample container

- Background
- Overview
- Object
- Docker Installation
- Basic Usage
 - Pull Docker image
 - Build Docker image
 - Docker run
 - Docker exec
 - Connect Containers with Veth Pair

virtual Ethernet (Veth) Pair

- veth: virtual Ethernet device
 - Linux virtual device
 - A local Ethernet tunnel
 - Created in pair
 - Can connect two different network namespaces
- Use a veth pair to connect containers



Create Two Containers without Network

- E.g., Create two containers named left and right

```
bash$ docker run -it --cap-add=NET_ADMIN --name left --net=none --privileged  
test
```

```
bash$ docker run -it --cap-add=NET_ADMIN --name right --net=none --privileged  
test
```

--cap-add=NET_ADMIN: Add Linux capabilities to modify network interfaces

--privileged: Give extended privilege to this container

--net=none: do not use docker network

Create veth Pair and Put inside Container

- Create veth pair at Linux host named **leftVeth** and **rightVeth**

```
bash$ sudo ip link add leftVeth type veth peer name rightVeth
```

- Put veth pairs inside containers **left** and **right**

```
bash$ sudo ip link set leftVeth netns $(sudo docker inspect -f '{{.State.Pid}}' left)
```

```
bash$ sudo ip link set rightVeth netns $(sudo docker inspect -f '{{.State.Pid}}' right)
```

- Enter each container and set veth of each container

- Set veth in left container

```
left bash$ ip link set leftVeth up
```

```
left bash$ ifconfig
```

- Repeat for **right** container

```
root@1c27c6b58ca7:/# ip link set leftVeth up
root@1c27c6b58ca7:/# ifconfig
leftVeth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    ether 96:43:84:b0:5a:e7 txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0 carrier 0 collisions 0
```

Assign IP Addresses to veth Interfaces in Containers

- Manually, set IP address for veth pair

```
Left bash$ ip addr add 10.0.0.1/8 dev leftVeth
```

```
Right bash$ ip addr add 10.0.0.2/8 dev rightVeth
```

- Alternatively, run dhcp in corresponding container to acquire an IP for veth.
- Check reachability

```
root@1c27c6b58ca7:/# ping 10.0.0.2 -c 5
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.061 ms
```

Q & A