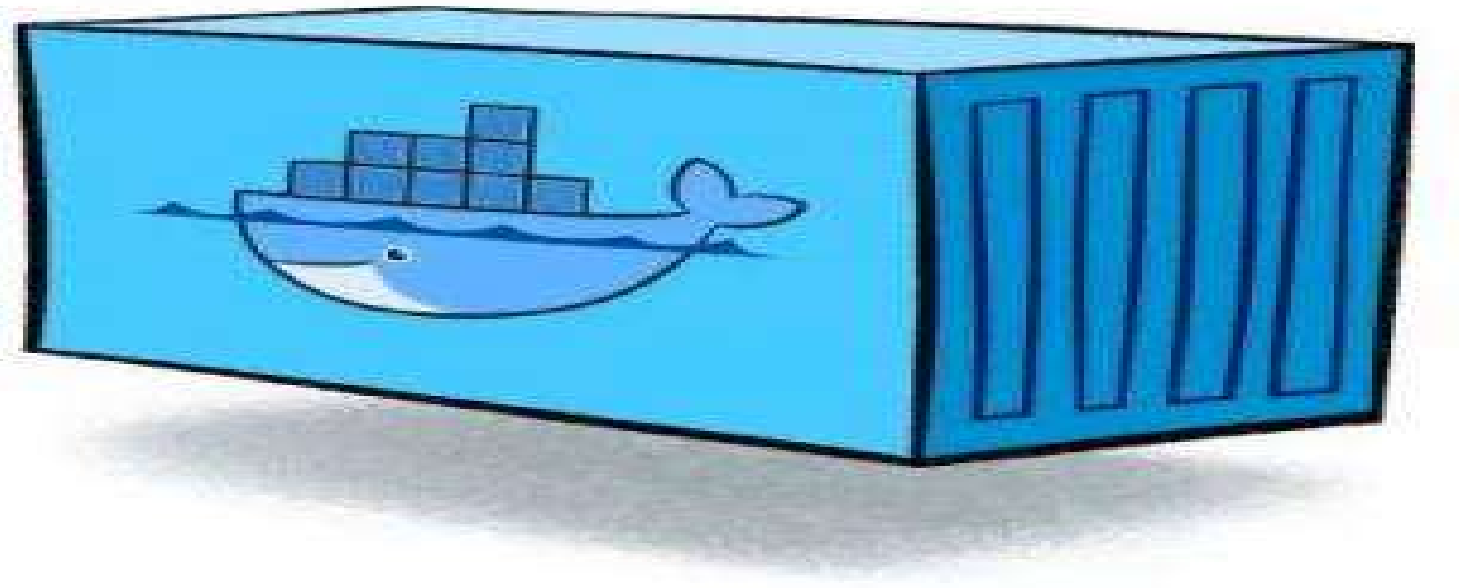
Containerizations

Unit objectives

After completing this unit, you should be able to:

- Introduction to Containers
- Introduction to Docker
- Manage Images
- Manage Containers
- Setup real scenario
  - LAMP (Linux → Apache → MySQL → PHP)

Introduction to Containers

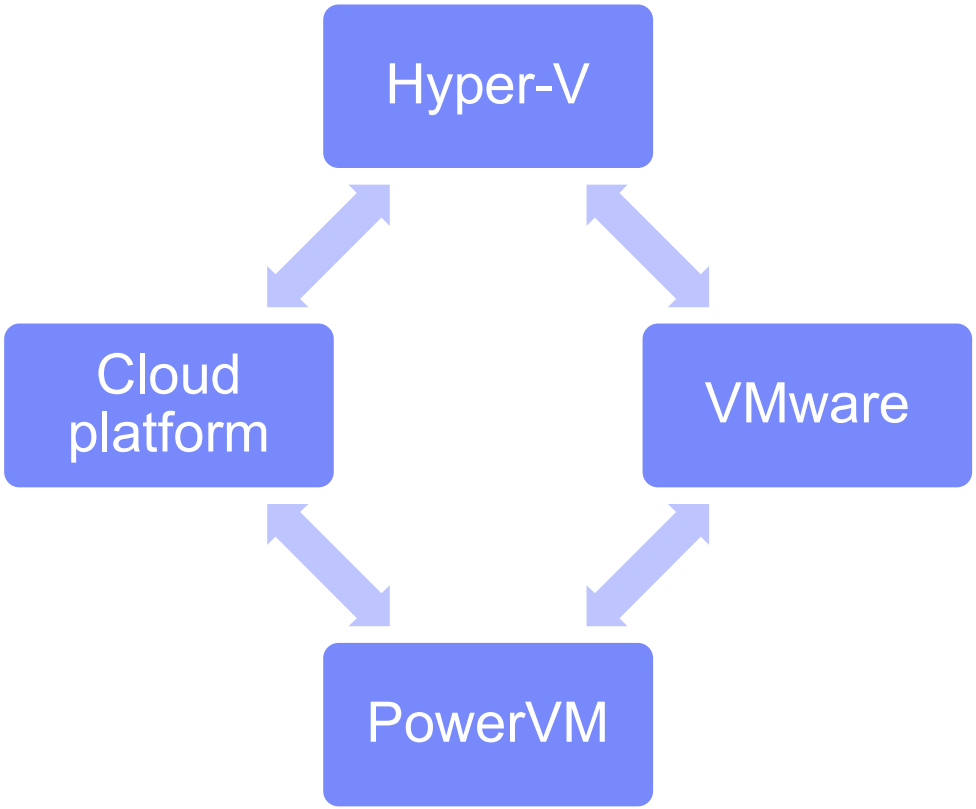


What is Container?

- is process that starts from image
  - Image consists of operating system, executables, binary codes, runtimes, libraries, configuration files, log files
- Achieve single purpose of application: database services, mailing services, web services and etc
- Lightweight, small and portable
- Environment Isolation
- Quick Deployment
- Good level of abstraction
- Reusability

Benefits of Containers

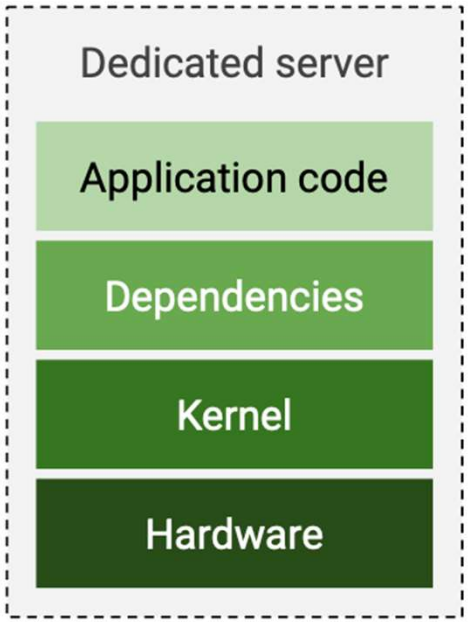
- Small and portable
- Good Level of abstraction
- Easibly migrateable between platform or environment



Individual Physical Servers

- Small and portable
- Good Level of abstraction

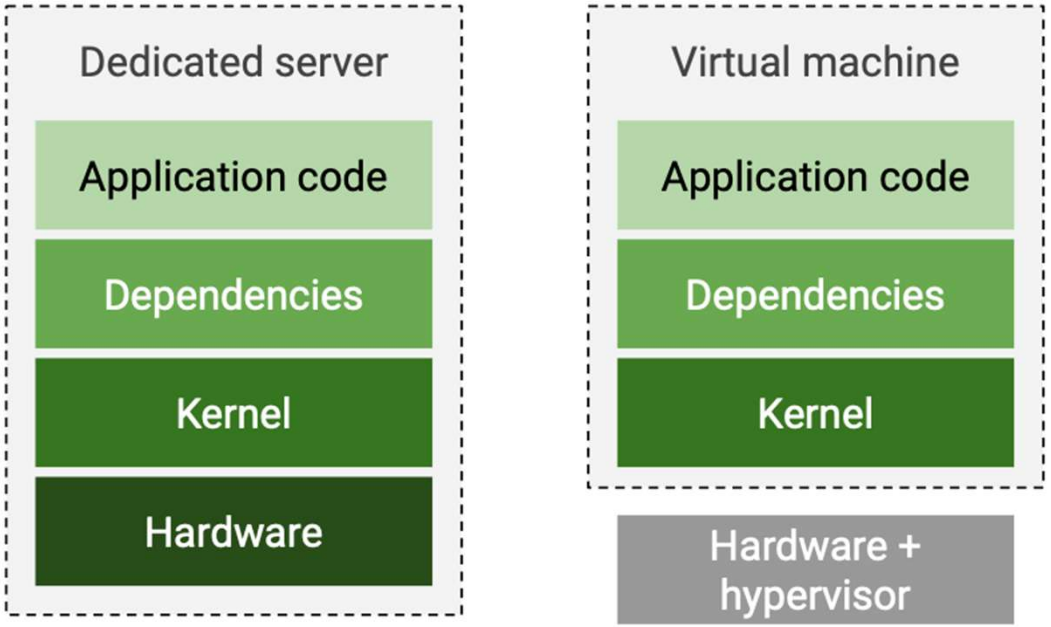
Looking back, you used to build applications on individual servers



Virtualization

- Small and portable
- Good Level of abstraction

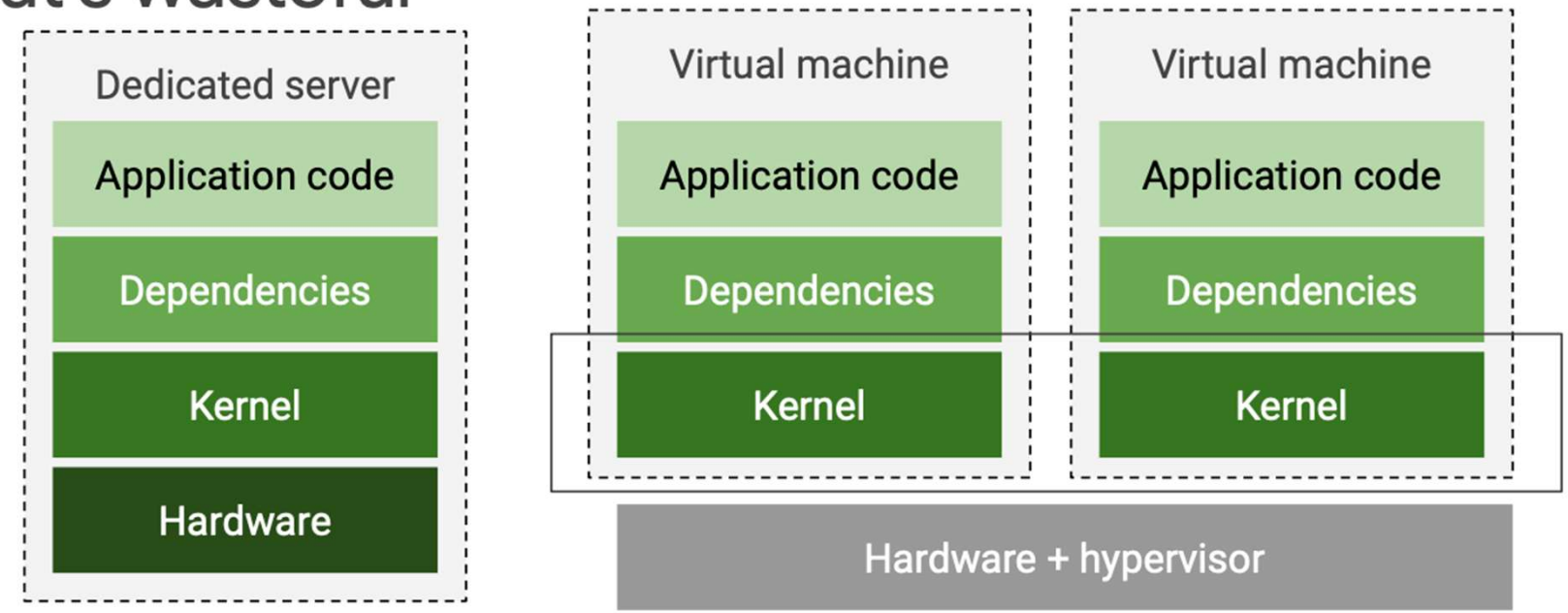
Then VMware popularized running multiple servers and operating systems on the same hardware



Virtualization

- Small and portable
- Good Level of abstraction

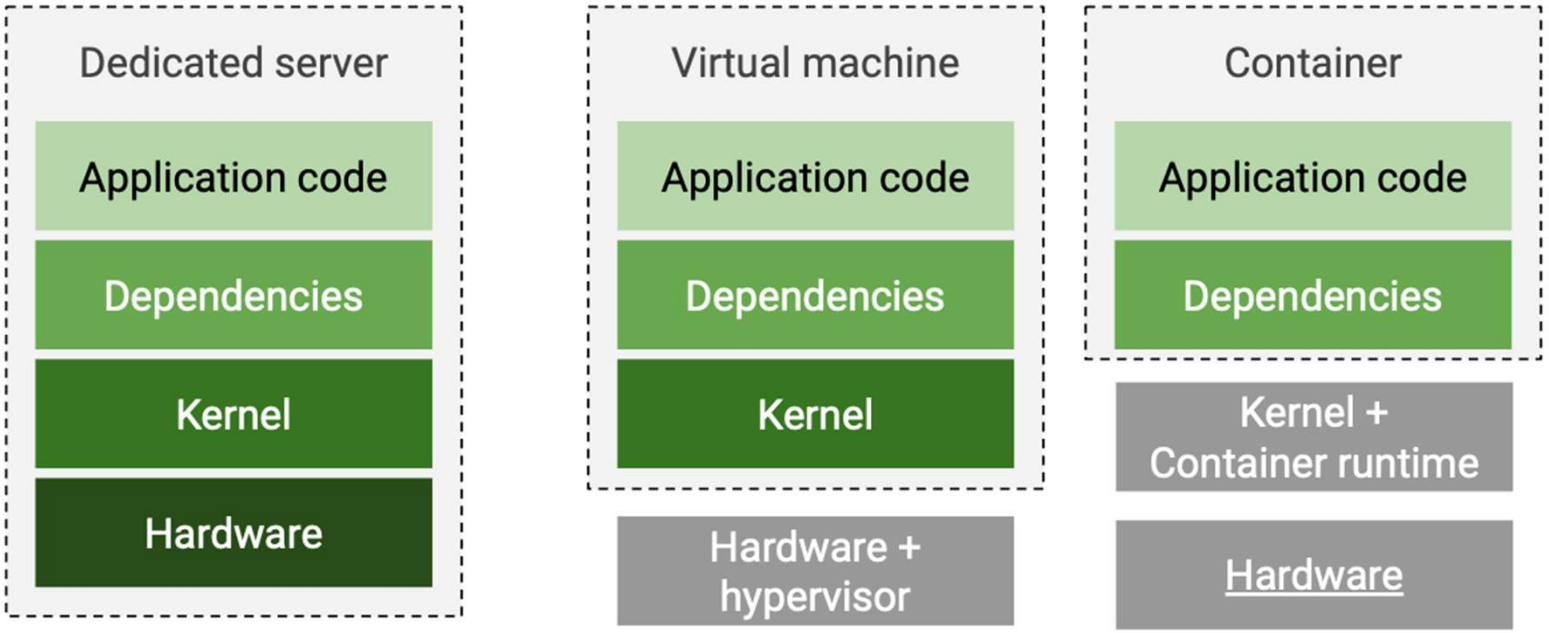
The VM-centric way to solve this is to run each app on its own server with its own dependencies, but that's wasteful



Containerization

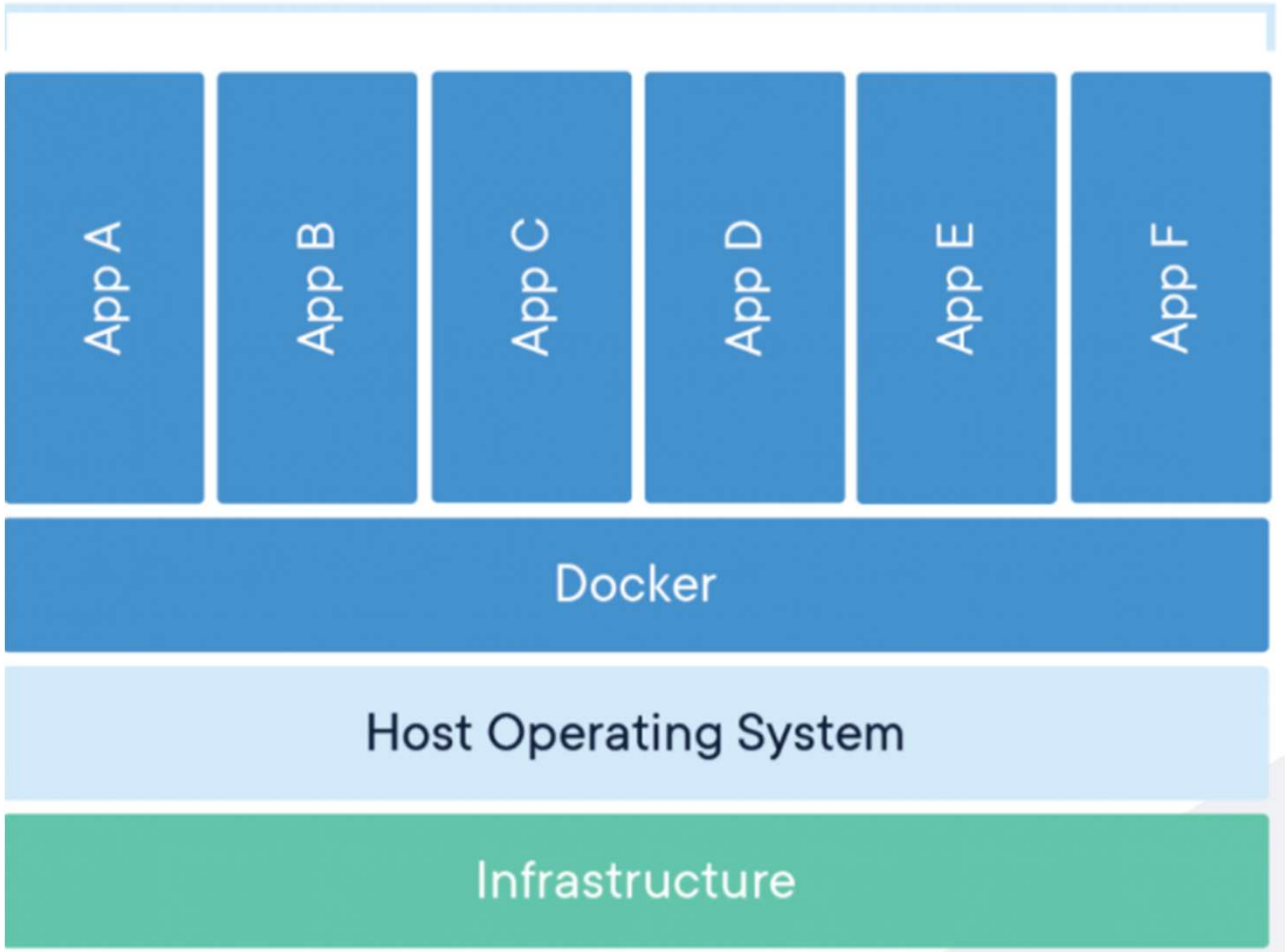
- Small and portable
- Good Level of abstraction

So you raise the abstraction one more level and virtualize the OS



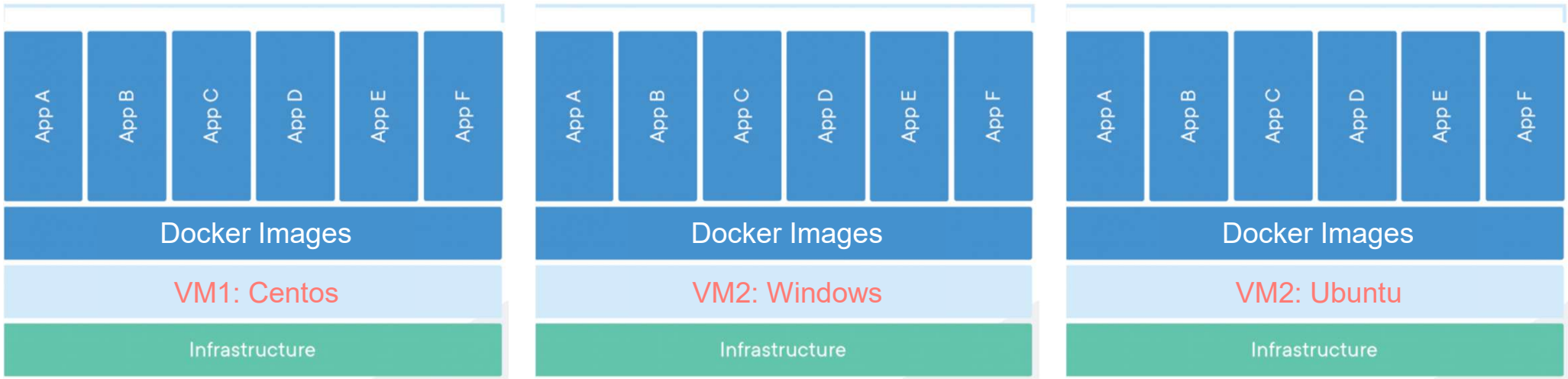
Containers on single node

- Open source
- Standardized image format
- Easily package, distribute, and manage applications within containers



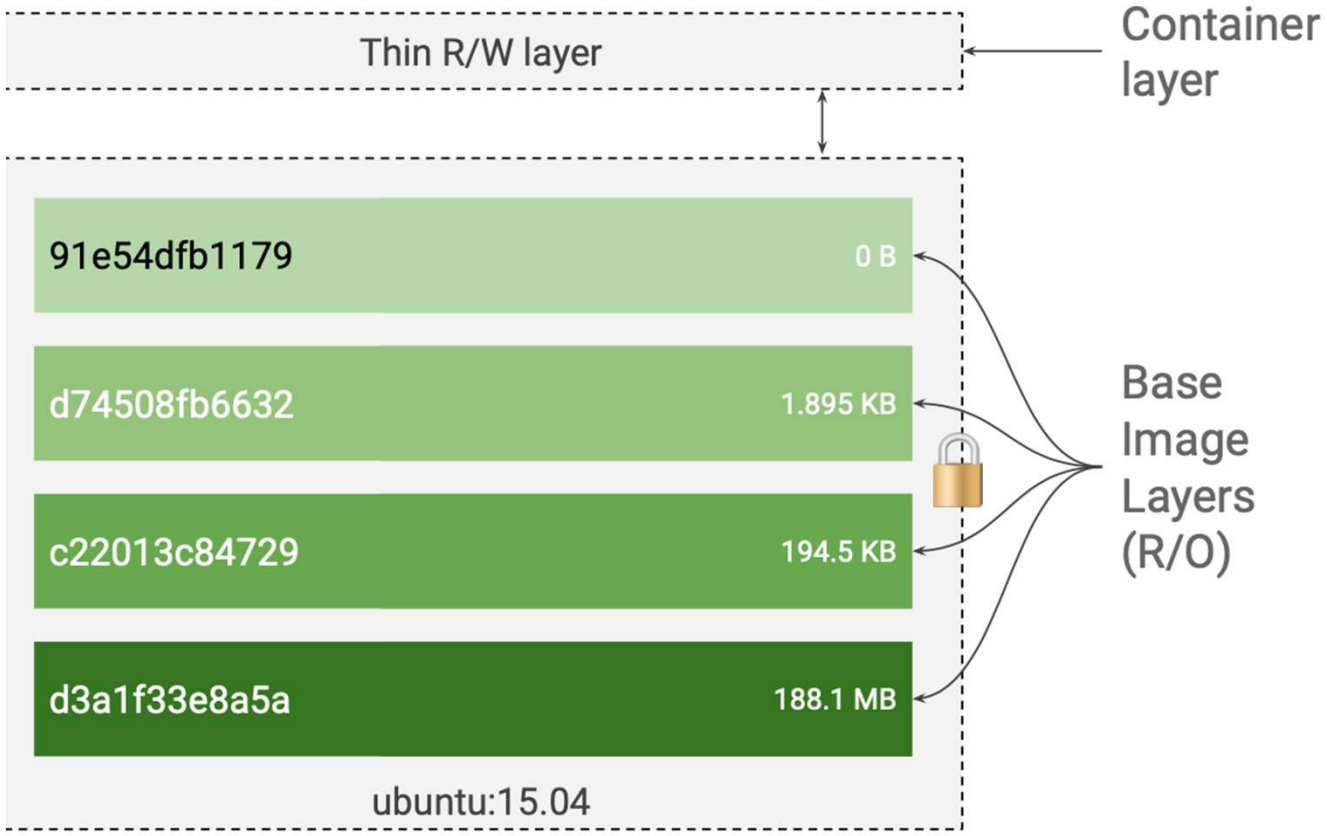
Containers on different platform/OS

- Open source
- Standardized image format
- Easily package, distribute, and manage applications within containers



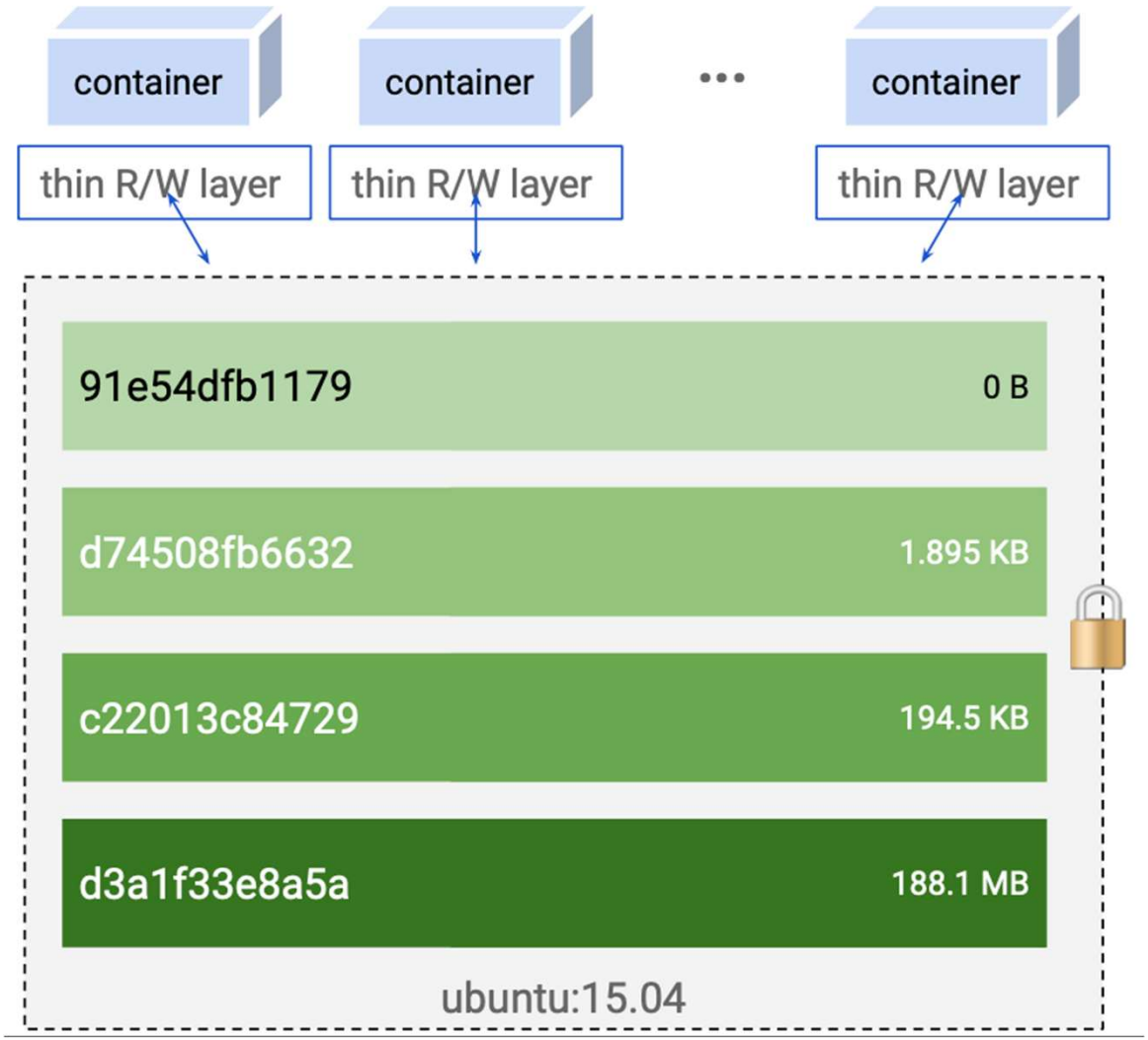
Containers use Layered File System

- With top layer writable
- Start multiple containers from same container image



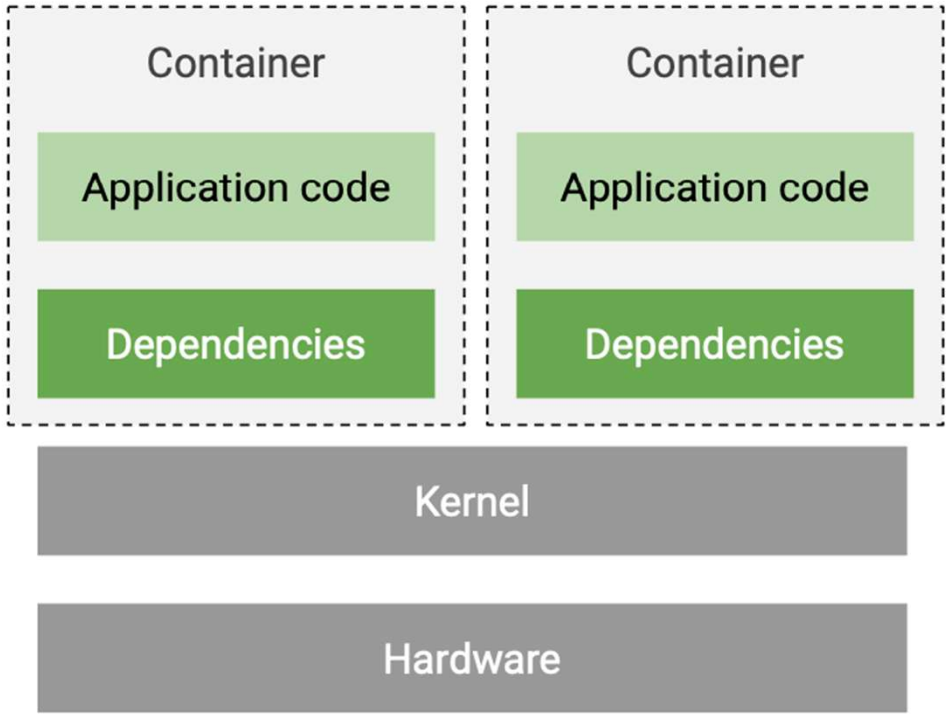
Containers promotes smaller shared images

- Base image size about 200 MB.
- As container spawn up, it may consumed only 100-200 KB.
- Instead of copy whole image, it creates layer with delta data only.
- Fast boot time

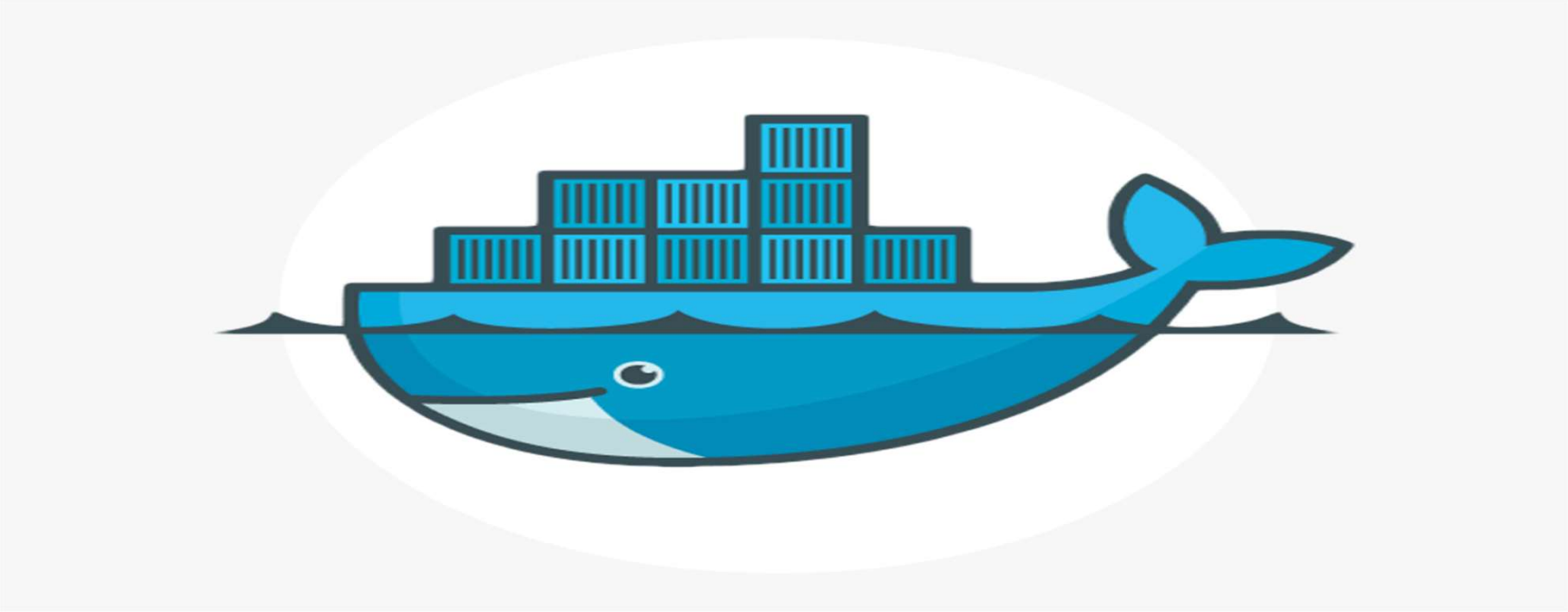


Why Developers love Containers?

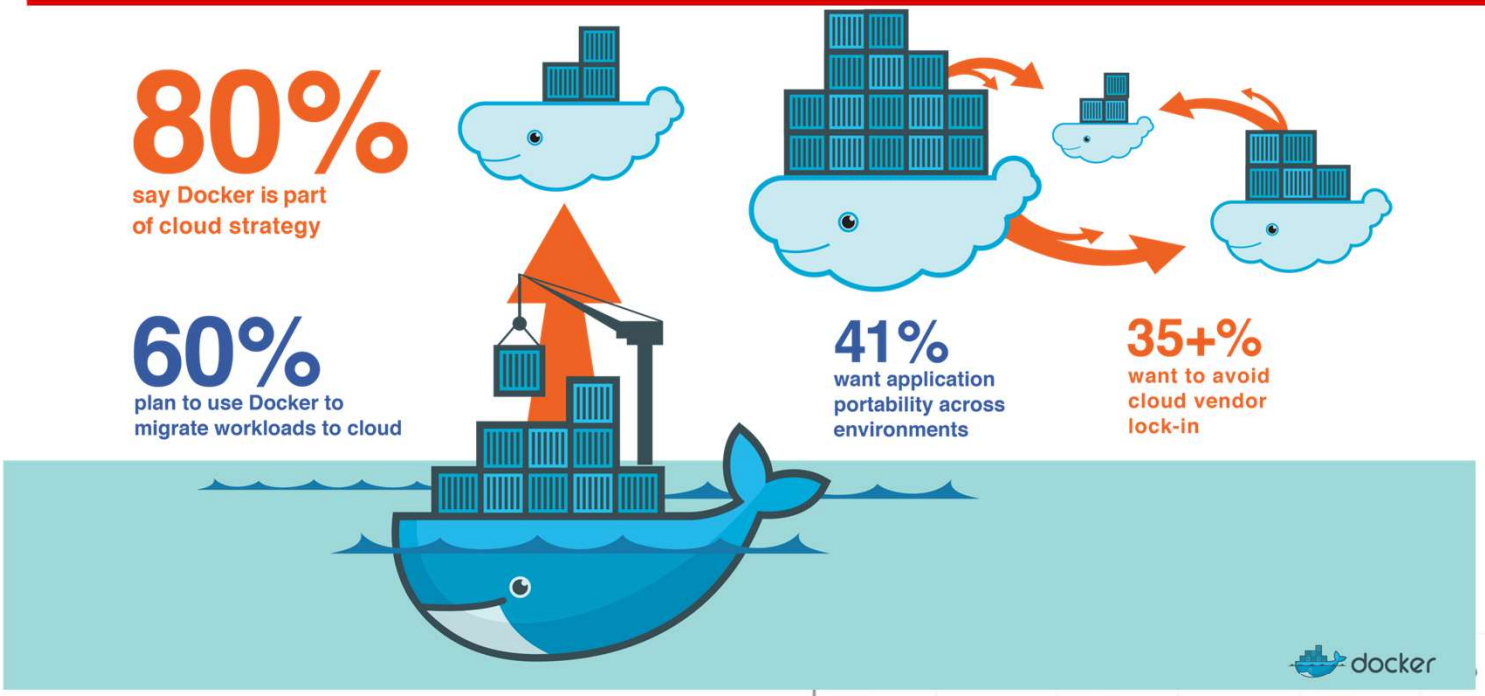
- Code works the same everywhere:
  - Across dev, test & production
  - Across bare-metal, VMs, cloud
- Packaged apps speed development
  - Rapid creation and deployment
  - CI/CD
  - Single file copy
- Provide best path to micro-services environment
- Isolated & elastic

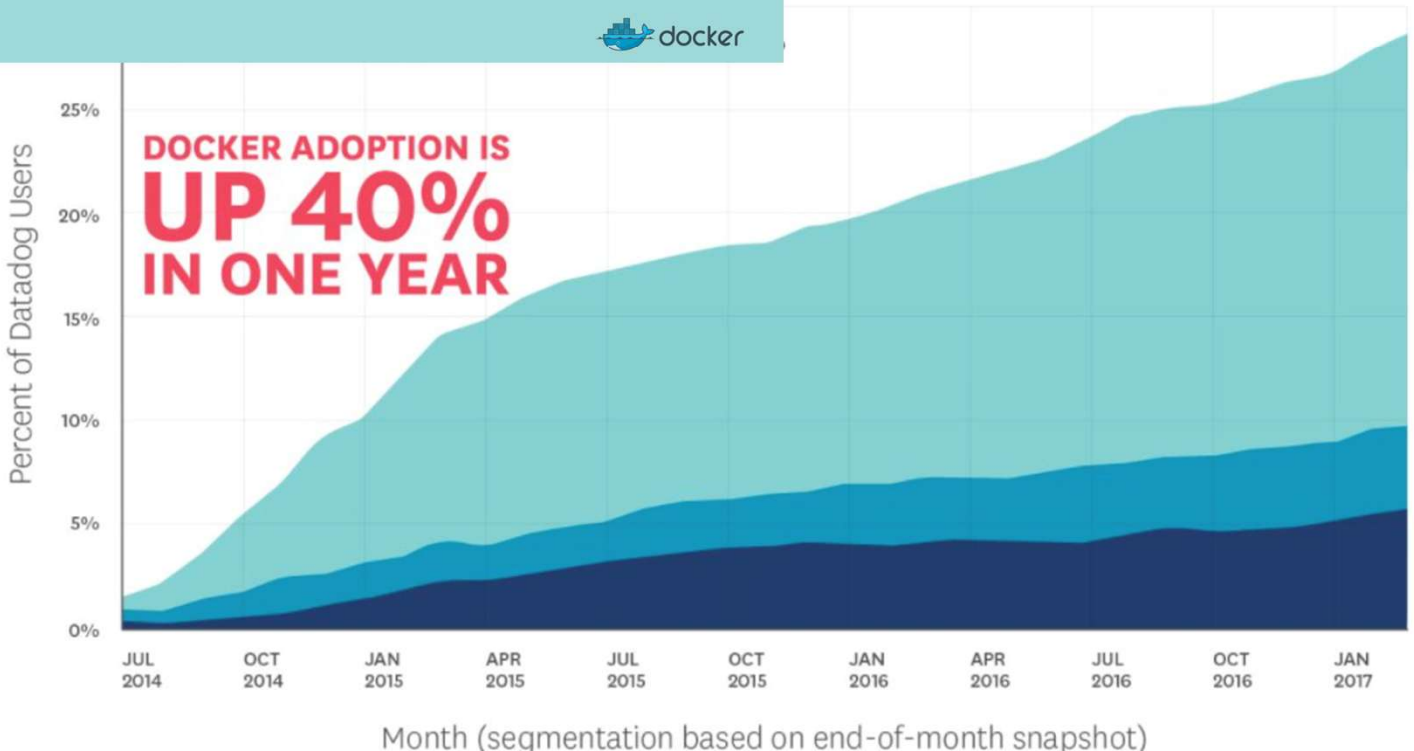


Introduction to Docker



Docker – Fastest growing technology





Management

- Docker Registry
  - docker.io
  - redhat.io
  - gcr.io
- In v7 and before, docker engine is to installed
  - docker command
- in v8, docker engine is built-in the kernel. no need installation
  - podman command
- For backward compatibility
  - alias docker="podman"
- Search for image
  - skopeo command (must install first)
  - or podman search

Search for image

- Explore docker hub

- Searching for images – unqualified-registries

```
# podman search ubuntu
# podman search lamp
```

- Search and limit number of result

```
# podman search ubuntu --limit 5
```

- Search based on conditions

```
# podman search --filter stars=1000 ubuntu --limit 5
# podman search --filter is-official mysql
```

Inspect image

- Using podman (images must be downloaded first)

```
# podman inspect docker.io/library/ubuntu
```

- Using skopeo

```
# dnf –y install skopeo
# skopeo inspect docker://docker.io/ubuntu
# skopeo list-tag docker://quay.io/jason_wong76/webserver
```

- Using skopeo to download

```
# skopeo copy docker://quay.io/jason_wong76/webserver:sport80 \
  containers-storage:localhost/myimage:1.0
# skopeo inspect containers-storage:localhost/myimage:1.0
```

Search for image

- Explore docker hub

- Using podman

```
# podman search ubuntu
# podman search lamp
```

- Install and use skopeo

```
# dnf –y install skopeo
# skopeo inspect docker://docker.io/library/mysql
# skopeo inspect docker://docker.io/library/python
# skopeo inspect docker://docker.io/library/ubuntu
# skopeo inspect docker://docker.io/library/centos
```

Download image

- Search for any repo and download latest Ubuntu

```
# podman pull ubuntu
```

- Search for specific repo and download latest CentOS

```
# podman pull docker.io/library/centos
# podman pull docker.io/library/nginx
```

- Download latest but specific tag of Ubuntu

```
# podman pull docker.io/library/centos:zesty
```

- Download apache built with Fedora Core libraries only

Manage image

- List all downloaded images

```
# podman images [-a]
```

- Inspect image

```
# podman inspect ubuntu | more
# podman inspect ubuntu:zesty | more
```

- Remove image

```
# podman image rm ubuntu:zesty  or
# podman rmi ubunty:zesty
```

- Remove all un-used images

```
# podman image rm -a
```

Start container from image

- Boot into ubuntu, view /etc/passwd file, quickly remove upon completion

```
# podman run --rm ubuntu cat /etc/*release
# podman run --rm nginx cat /proc/cpuinfo
```

- Boot into nginx with interactive shell

```
# podman run -it nginx /bin/bash
nginx-id:/# uname –a
nginx-id:/# ls /dev
nginx-id:/# apt list
nginx-id:/# exit
```

Inspect container

- List all running containers

```
# podman ps
```

Inspect a container for more information

```
# podman inspect <container-id> | more
```

- List all processes running in the container

```
# podman top <container-id>
```

- Fetch logs from the container

```
# podman logs <container-id>
```

Multiple login session into container

- On Terminal 1, start up a container on centos

```
# podman run -it centos /bin/bash
centos-id# cat /etc/*release
centos-id# tty
```

- On Terminal 2, access the same centos container

```
# podman ps
# podman exec -it <container-id> /bin/bash
centos-id# echo "test" > /dev/pts/0
centos-id# top
```

- On both terminal , exit out

Root vs Rootless container

Root Container

- Start by root account
- Has unlimited access to resources
  - Can access all network ports
  - Container starts processes using rootID
- Not good idea: compromise on container can lead to malicious attack on main host

Rootless Container

- Start by non-root account
- Does not required root privileges
- Has limited access to resources
  - Network port >=1024
  - User mapping to PV is required
- Compromise on container won't affect host machine

Manage Container Resources

- Environment Variables
- DNS Resolution
- For storage resources in large deployment of containers:
  - Sophisticated storage solutions
  - RedHat Openshift Container Platform
- For storage resources in small deployments of containers:
  - Persistent Storage / Volume (PV)
- For network resources:
  - Port Mapping

Environment Variables

- Required or optional configuration data to start an application
- Example: Database container such as MariaDB or MYSQL requires following parameters to be configured:
  - MYSQL_ROOT_PASSWORD=<something>
  - MYSQL_DATABASE=<name of database>
  - MYSQL_USER=<non root account>
  - MYSQL_PASSWORD=<password for the non root account>
  - More variables from vendor / official site
- If not specify, will get error when start the container

podman run --name db01 -d docker.io/library/mysql

podman log db01

```
You need to specify one of the following as an environment variable:
- MYSQL_ROOT_PASSWORD
- MYSQL_ALLOW_EMPTY_PASSWORD
- MYSQL_RANDOM_ROOT_PASSWORD
```

Environment Variables

- Start new MYSQL container with variable

```
# podman run --name db01 \
    -e MYSQL_ROOT_PASSWORD=r00t \
    -e MYSQL_DATABASE=salesdb \
    -e MYSQL_USER=user01 \
    -e MYSQL_PASSWORD=user01pass \
    -d docker.io/library/mysql
```

- Verify successful deployment

```
# podman ps
# podman exec -it db01 bash
bash# mysql -uuser01 -puser01pass
mysql> show databases;
```

Container Persistent Storage

- By default, use ephemeral storage
- Consider file-system level mount
- Configure ACL, SELinux
- Configure user and group owner
  - MARIADB → mysql user/group
  - Can use root account; but not preferable
  - Rootless container don’t use root

Container Persistent Storage

- By default, use ephemeral storage
- Consider file-system level mount
- Configure ACL, SELinux
- Configure user and group owner
  - MARIADB → mysql user/group
  - Can use root account; but not preferable
  - Rootless container don't use root
- To obtain UID mapping for user namespace

```
[user@host ~]$ podman unshare cat /proc/self/uid_map
         0       1000          1
         1     100000      65536
[user@host ~]$ podman unshare cat /proc/self/gid_map
         0       1000          1
         1     100000      65536
```

Configure Persistent Storage

- View mysql user UID and GID inside container

```
[user@host ~]$ podman exec -it db01 grep mysql /etc/passwd
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin/nologin
```

- Change ownership of local folder / mount point

```
[user@host ~]$ mkdir /home/user/db_data
[user@host ~]$ podman unshare chown 27:27 /home/user/db_data
```

- Mount the local folder as persistent volume to container (without SELinux)

```
# podman run --name db01 \
  -v /home/user/db_data:/var/lib/mysql \
  -d registry.lab.example.com/rhel8/mariadb-105
```

Verify Mounted Container Persistent Storage

- View processes

```
[user@host ~]$ podman ps -a
CONTAINER ID  IMAGE                                                COMMAND
 CREATED         STATUS                          PORTS       NAMES
dfdc20cf9a7e  registry.lab.example.com/rhel8/mariadb-105:latest  run-mysqld
 29 seconds ago  Exited (1) 29 seconds ago                     db01
```

- Upon checking on container logs, reveals permission denied error

```
[user@host ~]$ podman container logs db01
...output omitted...
---> 16:41:25     Initializing database ...
---> 16:41:25     Running mysql_install_db ...
mkdir: cannot create directory '/var/lib/mysql/data': Permission denied
Fatal error Can't create database directory '/var/lib/mysql/data'
```

SELinux contexts for Container Storage

- Must use container_file_t SELinux context type
- Approach 1: Old method

```
# semanage fcontext -at container_file_t "/home/user/db_data(/.*)?"
# restorecon
```

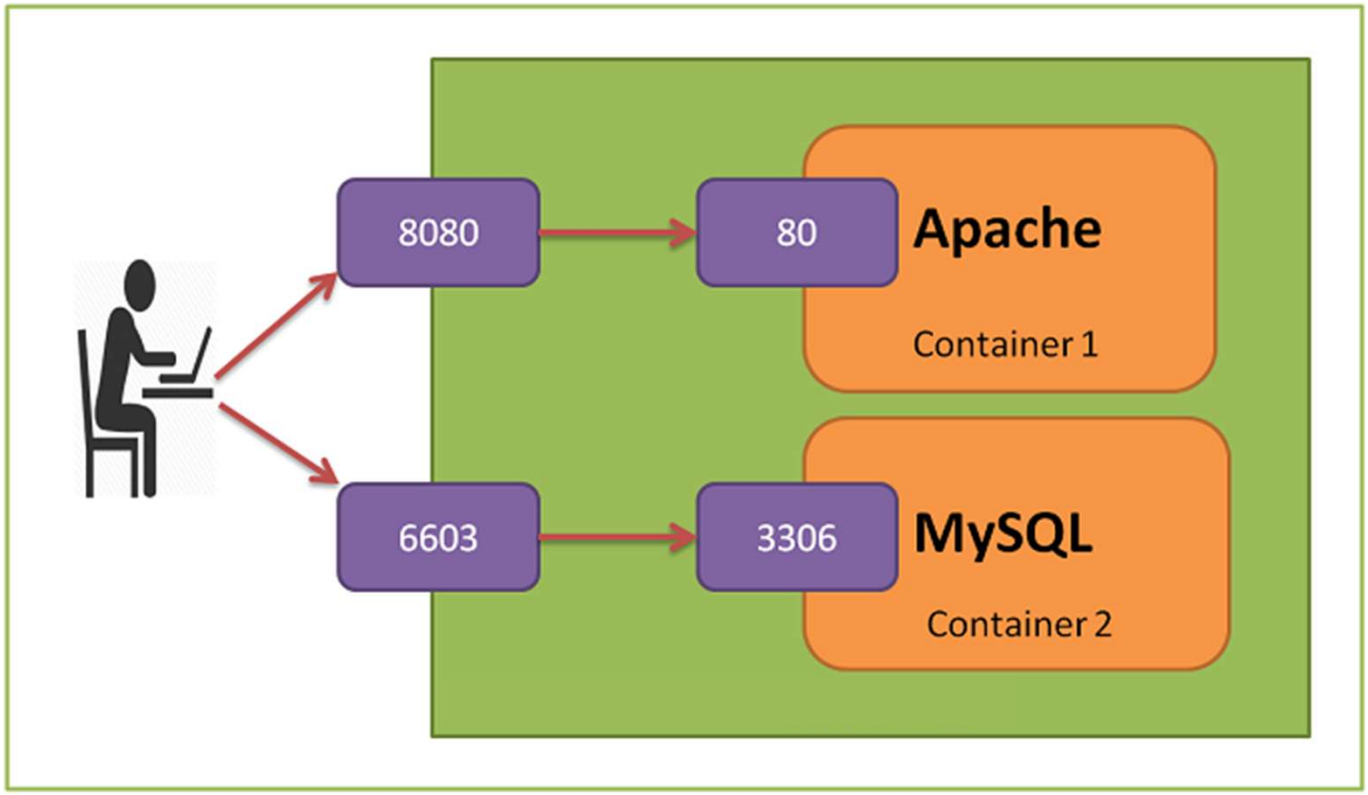
- Approach 2: Preferred method

```
# podman run --name db01 \
   -v /home/user/db_data:/var/lib/mysql:Z \
   -d registry.lab.example.com/rhel8/mariadb-105
```

- Verify

```
# ls –Z /home/user
system_u:object_r:container_file_t:s0:c81,c1009 dbfiles
…output omitted…
```

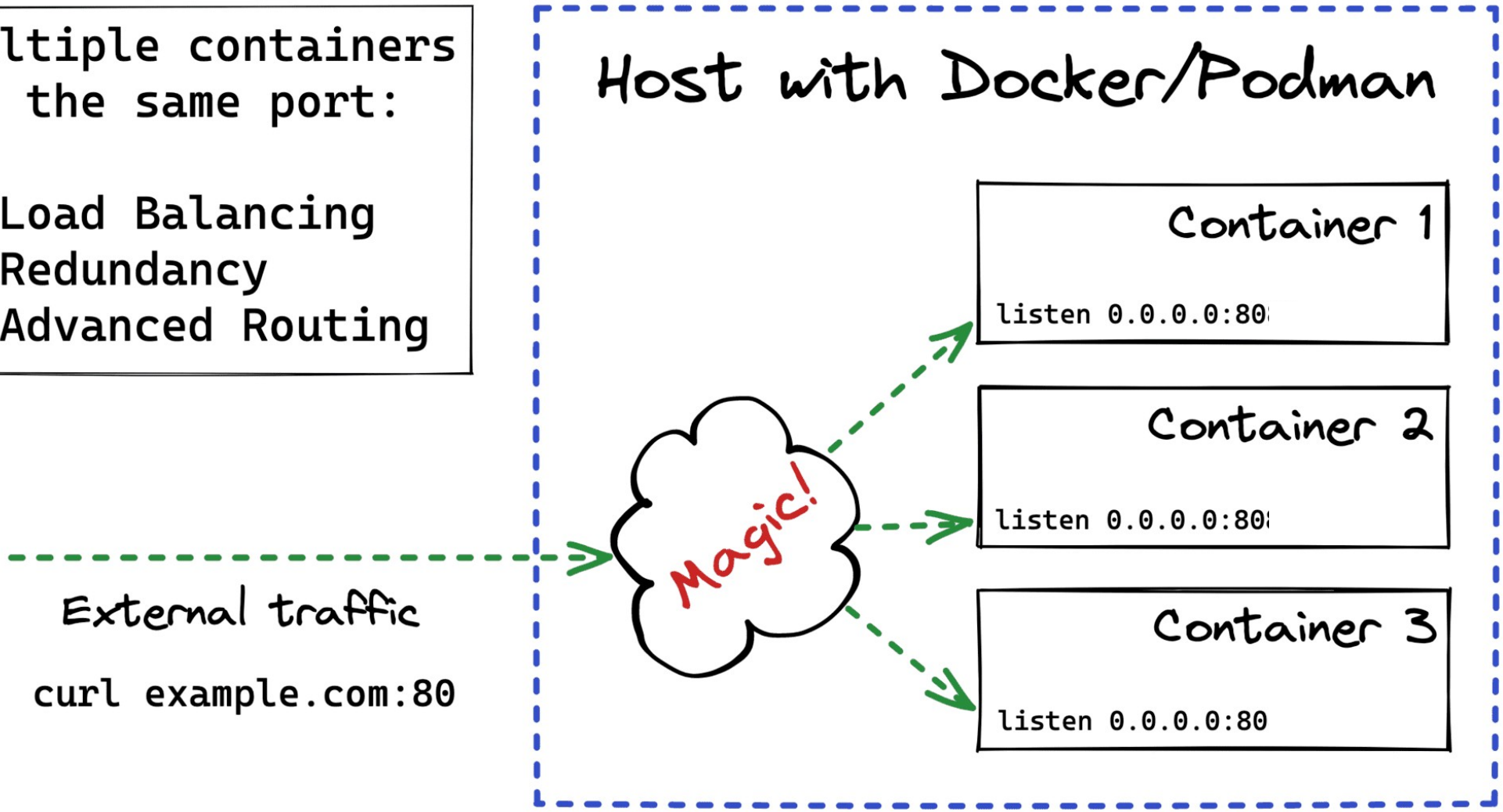
Port Mapping to Containers



Why multiple containers of same services?

Multiple containers
on the same port:

+ Load Balancing
+ Redundancy
+ Advanced Routing



Port Mapping to Containers

- Assign port 8080 to container 1

podman run --name container1 –p 8080:80 -d docker.io/library/httpd

- Assign port 8181 to container 2

podman run --name container2 –p 8181:80 -d docker.io/library/httpd

- Assign port 8282 to container 3

podman run --name container3 –p 8282:80 -d docker.io/library/httpd

Verify all ports

podman port -a

podman ps

Configure Firewall and SELinux

```
# firewall-cmd --add-port={8080,8181,8282}/tcp --permanent
# firewall-cmd --reload

# semanage port -at httpd_cache_port_t –p tcp 8080
# semanage port -at httpd_cache_port_t –p tcp 8181
# semanage port -at httpd_cache_port_t –p tcp 8282

#
```

Port Mapping to Containers

- View all port mappings

podman port –a

```
25ee41f8ab93    80/tcp -> 0.0.0.0:8181
a6eb6f0e07ec    80/tcp -> 0.0.0.0:8080
fa21e0645e1a    80/tcp -> 0.0.0.0:8282
```

curl localhost:8080

… showing web content on container1…

curl localhost:8181

… showing web content on container2…

Other topics continue on with PDF

- DNS configuration in Container
- Manage Containers as Root with systemd

Scenario: Setting up multiple database systems

- Use open source database : MariaDB
- First deployment
  - Name: db01
  - Port: map to local port 33061
  - Storage: /home/student/storage/db01
- Second deployment
  - Name: db02
  - Port: map to local port 33062
  - Storage: /home/student/storage/db02

Scenario: Setting up multiple database systems

- Use open source database : MariaDB
- First deployment
  - Name: **db01**
  - Port: map to local port **33061**
  - Storage: /home/student/storage/**db01**
  - User: student (Password: student)
  - Root password: redhat
  - Database: testdb
- The setup

```
# podman pull mariadb
# podman run --name db01 -v “/home/student/storage/db01:/var/lib/mysql” -p 33061:3306 \
  -e MARIADB_USER=student -e MARIADB_PASSWORD=student \
  -e MARIADB_ROOT_PASSWORD=redhat –e MARIADB_DATABASE=testdb \
  -d mariadb
#
```

Scenario: Setting up multiple database systems

- Use open source database : MariaDB
- Second deployment
  - Name: **db02**
  - Port: map to local port **33062**
  - Storage: /home/student/storage/**db02**
  - User: student (Password: student)
  - Root password: redhat
  - Database: testdb
- The setup

```
# podman pull mariadb
# podman run --name db02 -v “/home/student/storage/db01:/var/lib/mysql” -p 33062:3306 \
  -e MARIADB_USER=student -e MARIADB_PASSWORD=student \
  -e MARIADB_ROOT_PASSWORD=redhat  -e MARIADB_DATABASE=testdb \
  -d mariadb
#
```

Verify deployments

- List content in the persistent storage

ls /home/student/storage/db0?

testdb will appear alongside other subdirectories created by MariaDB

- List containers

podman ps

- Query databases

podman exec db01 mysql -ustudent -pstudent -e “show databases”

- Create additional database

podman exec db01 mysql -uroot -predhat -e “create database mydb”

Create table interactively

- Do following

```
# podman exec –it db01 sh
db01:/# mysql –uroot -predhat
MariaDB [(none)]> show databases;
MariaDB [(none)]> use mydb;
MariaDB [(mydb)]> create table users (
name varchar(50) NULL,
age varchar(50) NULL,
address varchar(50) NULL
);
MariaDB [(mydb)]> show tables;
```

Insert rows into table

- Still with db01 container, do following

```
MariaDB [(mydb)]> show tables;

MariaDB [(mydb)]> insert into users (name,age,address) values ("Ali","35", "PJ");

MariaDB [(mydb)]> insert into users (name, age,address) values("Sam", "66", "PJ");

MariaDB [(mydb)]> insert into users (name, age,address) values("Mary", "28", "KL");
```

- Display inserted rows

```
MariaDB [(mydb)]> select * from users;
```

Example : Setup a LAMP

- Linux with Apache, MySQL, PHP

- On terminal1, download and run lamp

```
# podman search lamp
# podman pull docker.io/mattrayner/lamp
# podman run --name server1 -d lamp
# podman run --name server2 -d lamp
```

Still with terminal1 verify running lamp servers

```
# podman ps –a
```

Create database non-interactively

- On Terminal2, Perform some database stuff

podman ps

podman logs <container-id>

podman exec <container-id> mysql –uroot –e "show databases"

podman exec <container-id> mysql –uroot –e "create database mydb"

Create table interactively

- Still with Terminal2, do following

```
# podman exec –it <container-id> /bin/bash
container-id:/# mysql –u root
mysql> show databases;
mysql> use mydb;
mysql> create table users (
• name varchar(50) NULL,
• age varchar(50) NULL,
• address varchar(50) NULL
• );
```

Insert rows into table

- Still with Terminal2, do following

```
mysql> show tables;
mysql> insert into users (name,age,address) values
(“James","35","sunnyvale");
mysql> insert into users (name, age,address)
values("barbara","59","sunnyvale");
mysql> insert into users (name, age,address)
values("nicholas","19","frankfurt");
```

- Now show all rows

```
mysql> select * from users;
```

Log into phpMyAdmin

We need the graphical

```
# systemctl isolate graphical.target
```

Get container's ip

```
# podman inspect <container-id> | grep 10.88
10.88.0.5
```

Launch firefox

Enter url ➔ http://10.88.0.5

Change url ➔ http://10.88.0.5/phpmyadmin

Login as admin with password shown earlier

Checkpoint

1. Why Docker so popular?
   a) It the most widely used open-source operating system in the whole wide world
   b) It has standardized format, industry leading in containerization
   c) It is a multi-billion company that provide cloud technology
   d) Its a leading virtualization technology on par with VMware, Hyper and so on

2. Which statements are accurate? [choose two]
   a) Docker image provides shared content to containers
   b) Container image provides shared content to Docker
   c) Docker is read writable wherelse Container is read-only image
   d) Container is read writable wherelse Docker is read-only image

3. How do you quickly spawn up new container from a downloaded Ubuntu image and get an interactive shell?
   a) docker run –it ubuntu /bin/bash
   b) docker pull docker.io/ubuntu
   c) podman run –it ubuntu /bin/bash
   d) podman pull docker.io/ubuntu

4. True or False: Best thing that developers loved about container is micro-services design

Checkpoint

1. Why Docker so popular?
   a) It the most widely used open-source operating system in the whole wide world
   b) It has standardized format, industry leading in containerization
   c) It is a multi-billion company that provide cloud technology
   d) Its a leading virtualization technology on par with VMware, Hyper and so on

2. Which statements are accurate? [choose two]
   a) Docker image provides shared content to containers
   b) Container image provides shared content to Docker
   c) Docker is read writable wherelse Container is read-only image
   d) Container is read writable wherelse Docker is read-only image

3. How do you quickly spawn up new container from a downloaded Ubuntu image and get an interactive shell?
   a) docker run –it ubuntu /bin/bash
   b) docker pull docker.io/ubuntu
   c) podman run –it ubuntu /bin/bash
   d) podman pull docker.io/ubuntu

4. True or False: Best thing that developers loved about container is micro-services design

Unit summary

Having completed this unit, you should be able to:

- Introduction to Containers
- Introduction to Docker
- Manage Images
- Manage Containers
- Setup real scenario
  - LAMP (Linux → Apache → MySQL → PHP)