Docker Overview and Installation

Docker Image Creation and Management

Docker Volumes and Container Deployment

MODULE OVERVIEW: CONTAINERISATION USING DOCKER

1. Evolution of Docker
2. Role of Docker in DevOps
3. Creation and Management of Docker Images
4. Deployment of Docker Container
5. Multi-container Application Deployment

Docker Overview and Installation

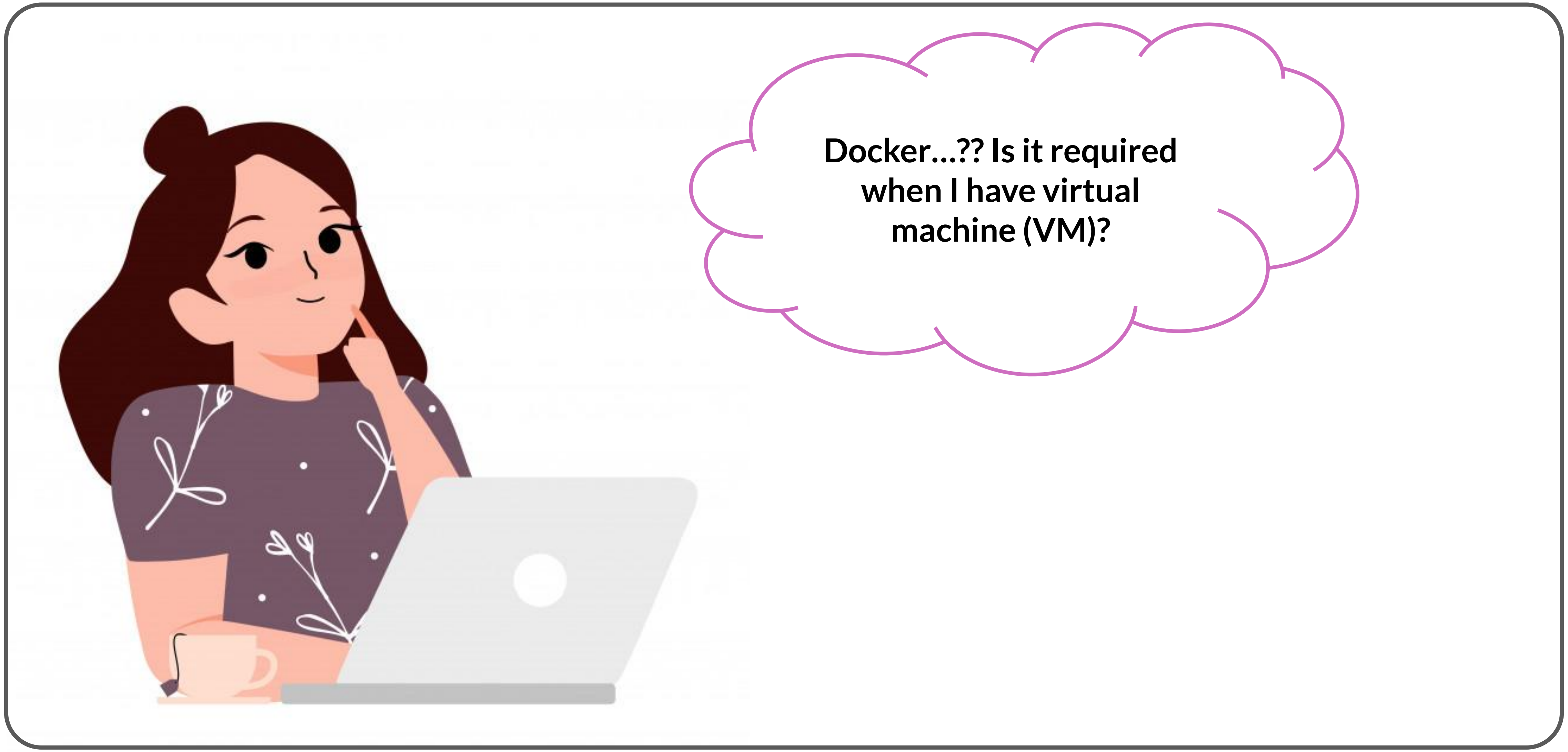
Docker Image Creation and Management

Docker Volumes and Container Deployment

SESSION OVERVIEW:
DOCKER OVERVIEW AND INSTALLATION

1. Understand Docker Journey
2. Understand Docker Advantages
3. Understand Docker Vocabulary
4. Understand Docker Architecture and its Components
5. Learn Docker Installation

WHY DOCKER?



WHY DOCKER?

But it works fine at my end

May be your machine is not the same as mine

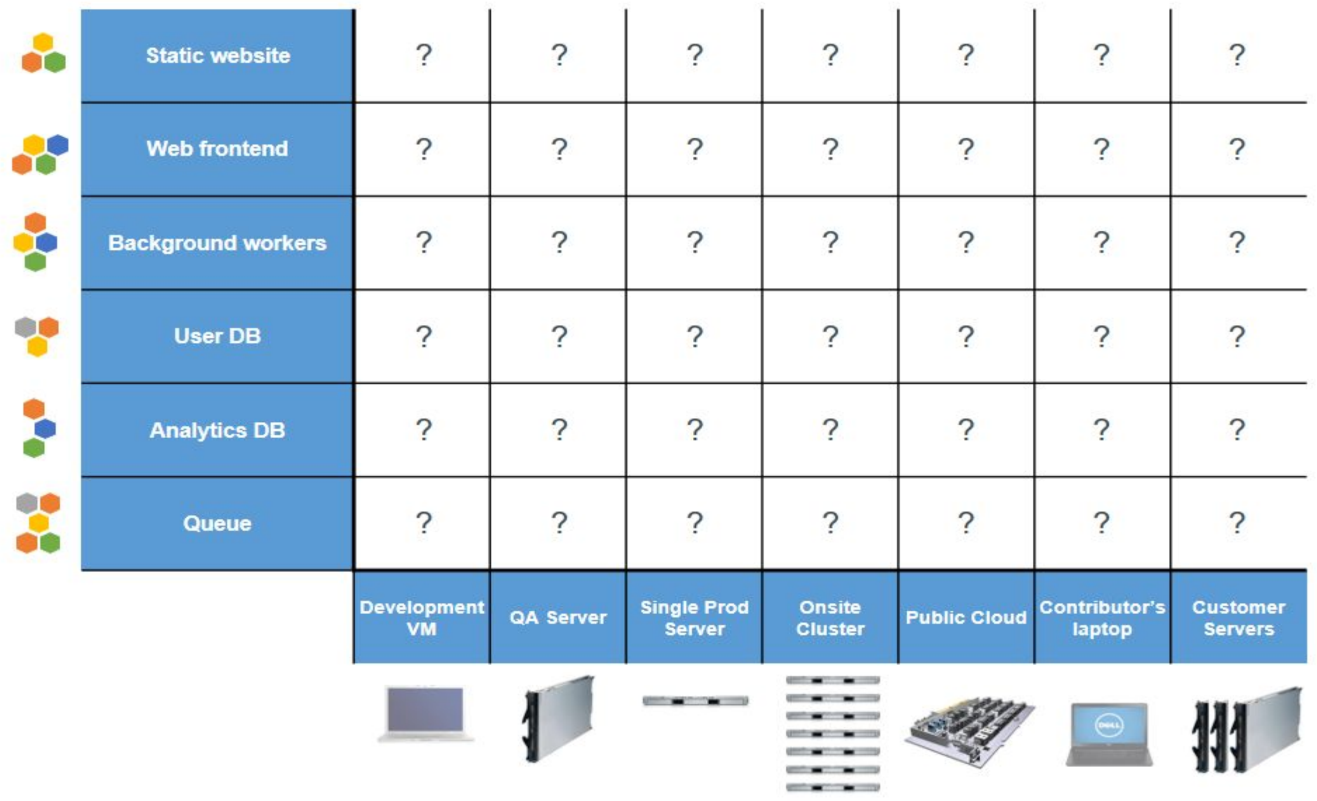
There is a problem with your application. It is not working at our end.

Developer

Operations

THE MATRIX FROM HELL

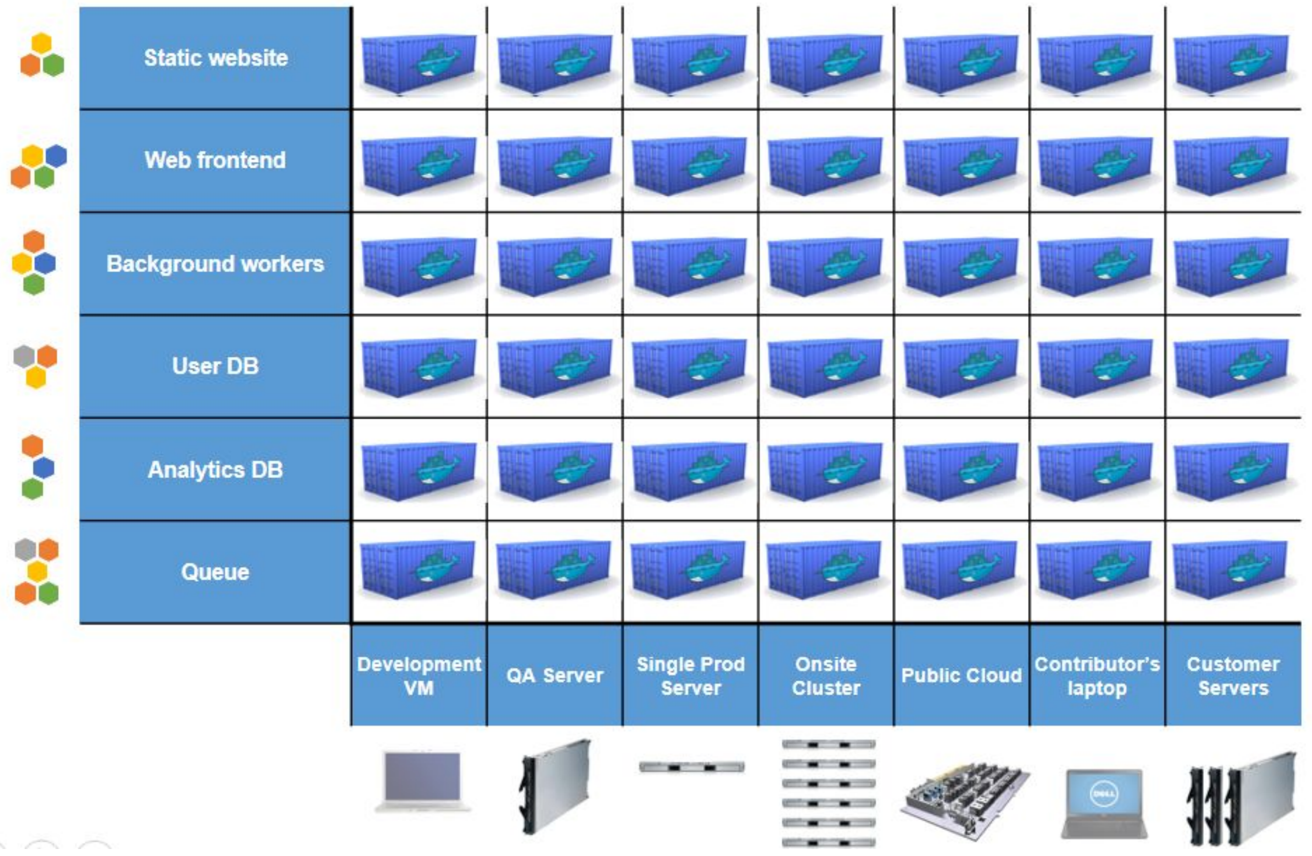
- Making sure that applications work perfectly fine with any combination of hardware/operating systems, libraries and dependencies is a challenge.
- Setting up appropriate Dev/Prod/QA environment takes a lot of time.



	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

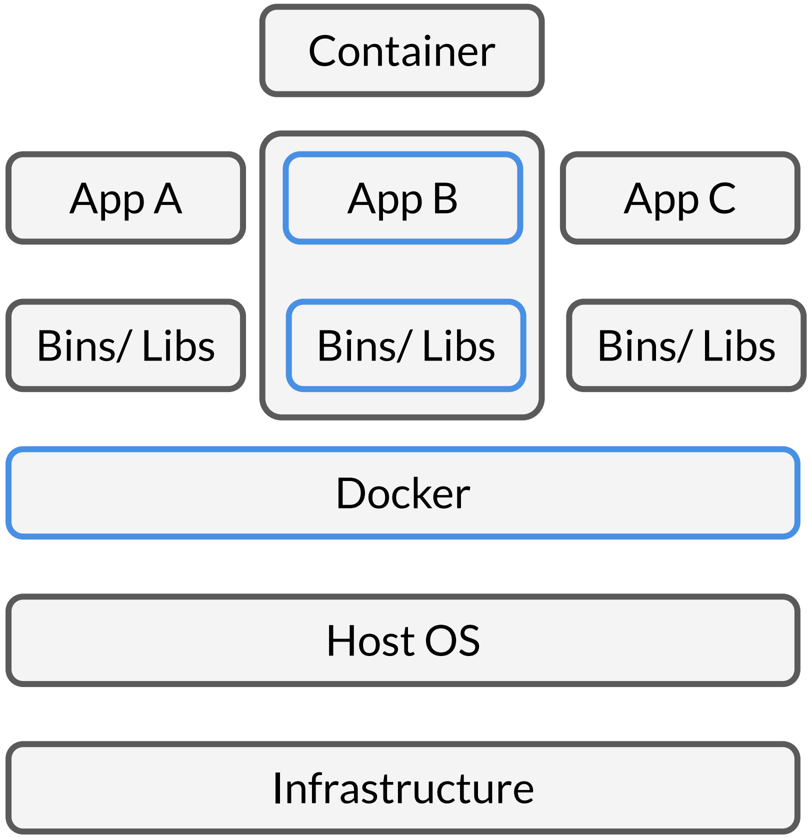
DOCKER ELIMINATES THE MATRIX FROM HELL

- Docker addresses the 'matrix from hell' by decoupling the application from the underlying operating system and hardware.
- Docker packages an application with its environment and all of its dependencies and, thus, solves **'Missing dependency problem'.**



WHAT IS DOCKER?

- Docker is a tool that helps in building and running containers that are easy to maintain and distribute.
- Docker containers are lightweight as compared to traditional virtual machines as they share the host operating system kernel.

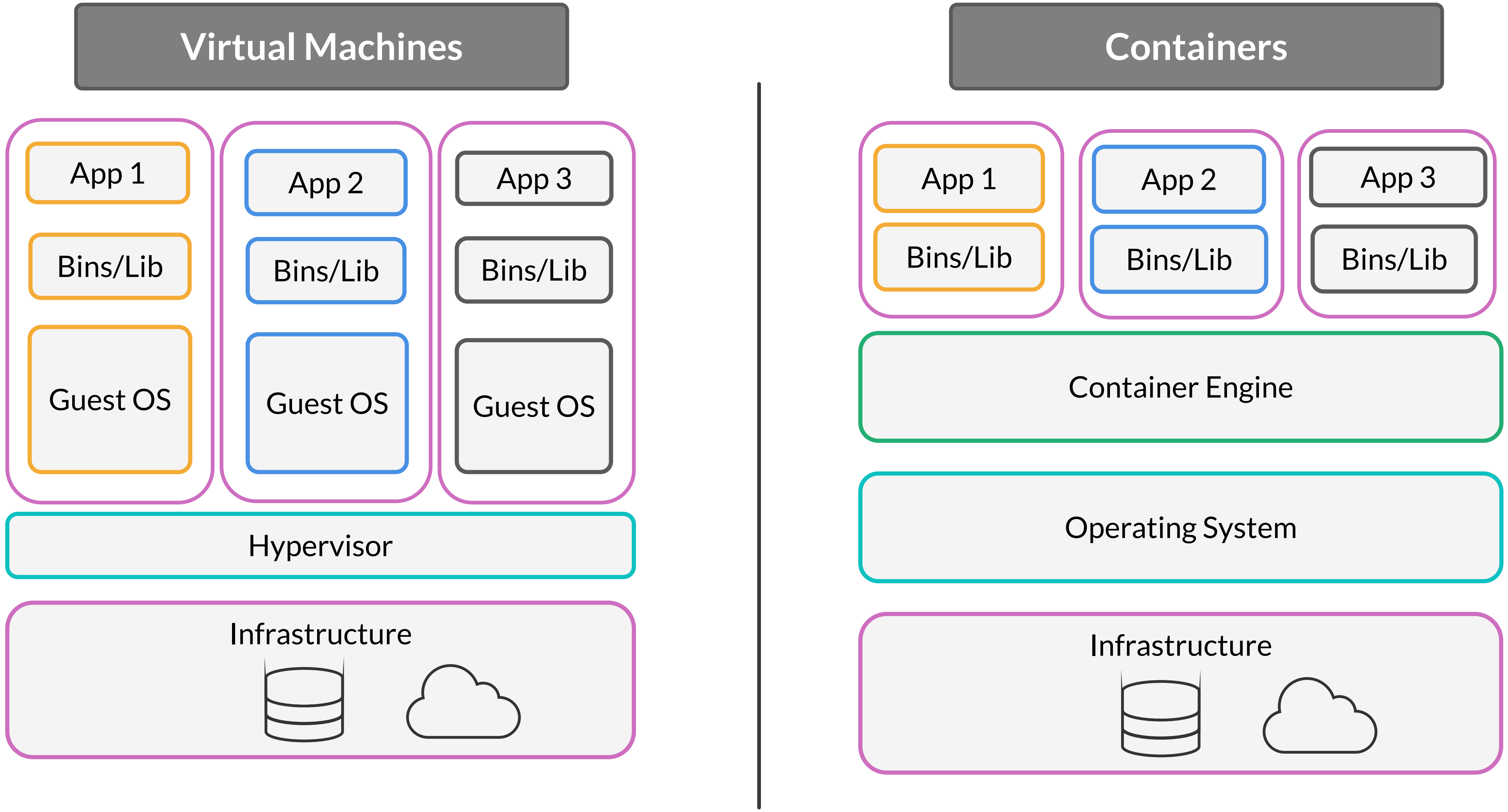


DOCKER VS VIRTUAL MACHINE

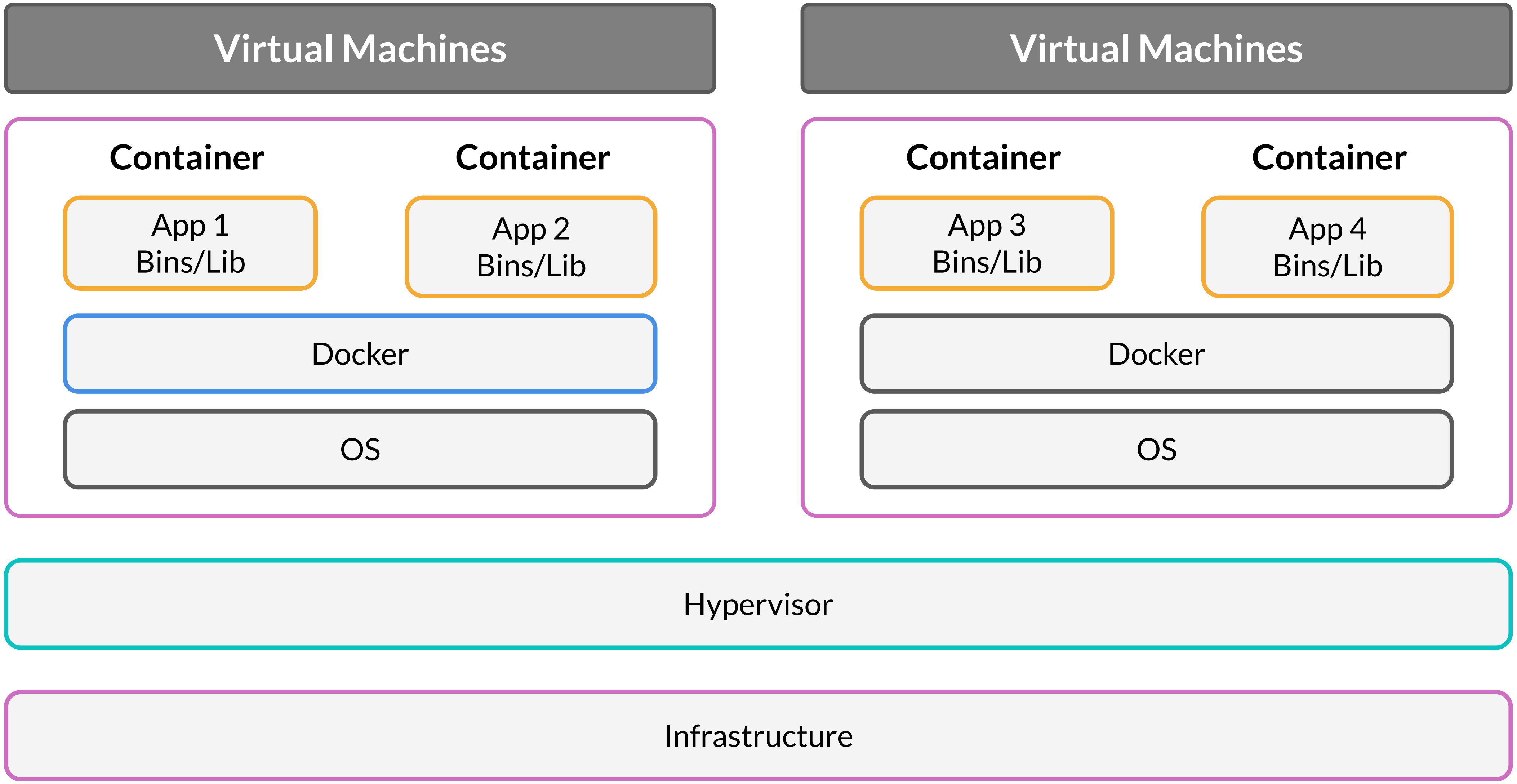
How is Docker different from Virtual Machine (VM)?

Containers virtualise the operating system, whereas virtual machine implements hardware virtualisation.

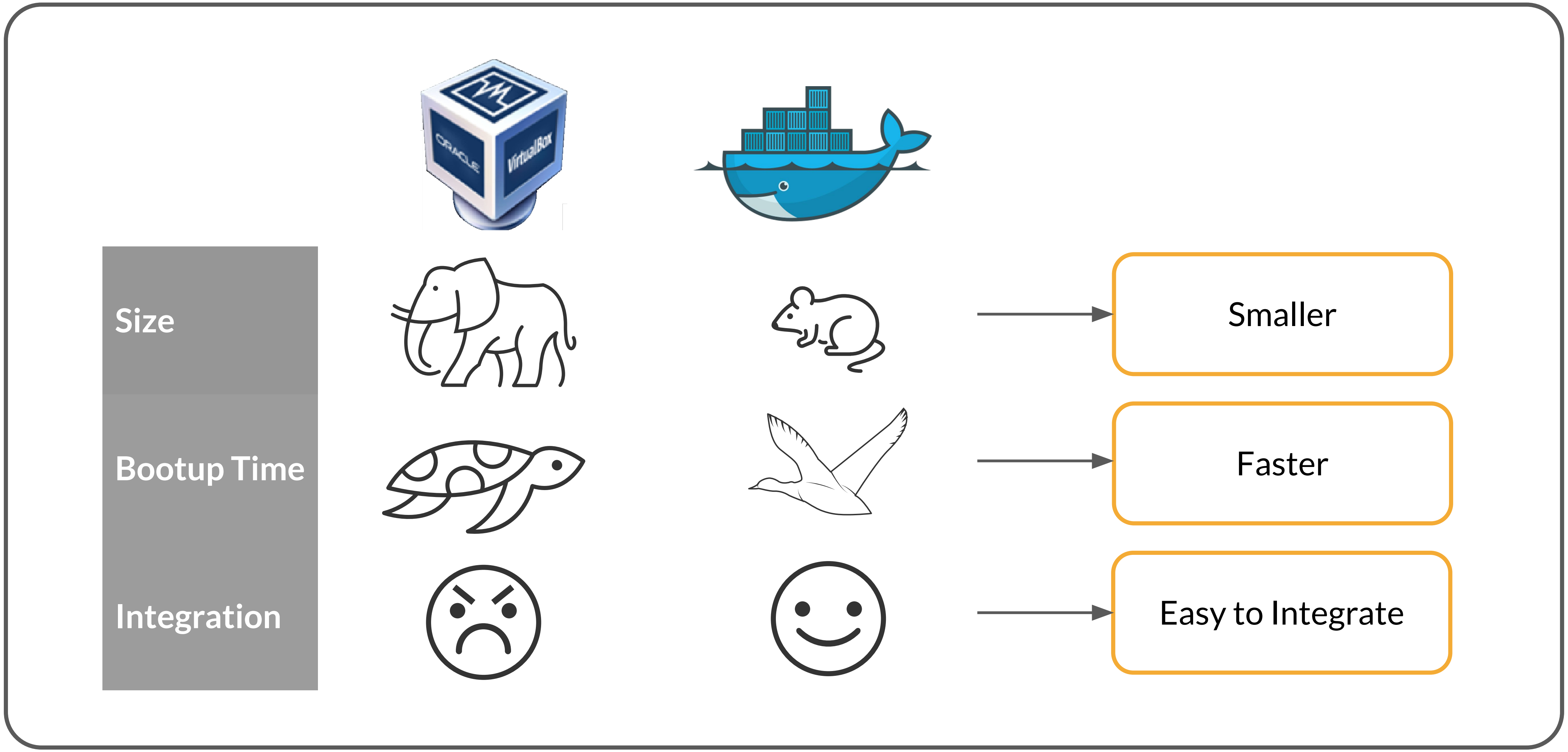
VIRTUAL MACHINES vs CONTAINERS



VIRTUAL MACHINES AND CONTAINERS

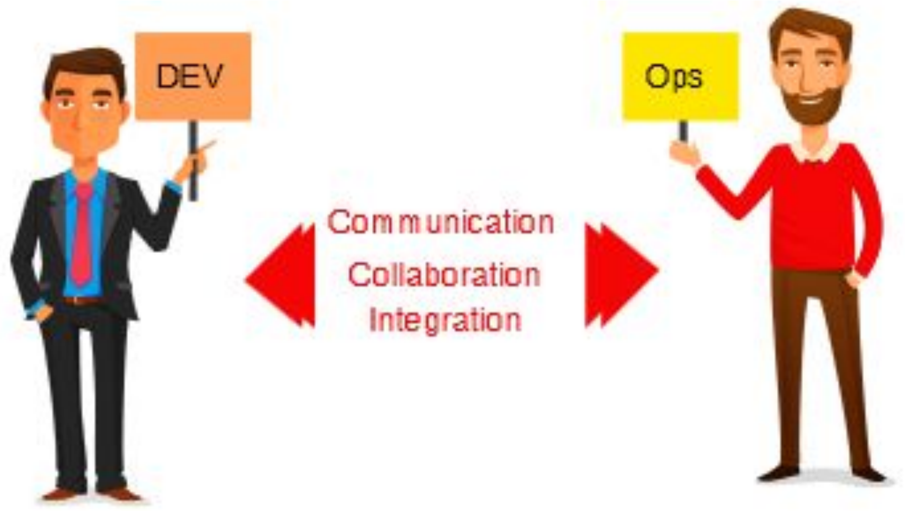


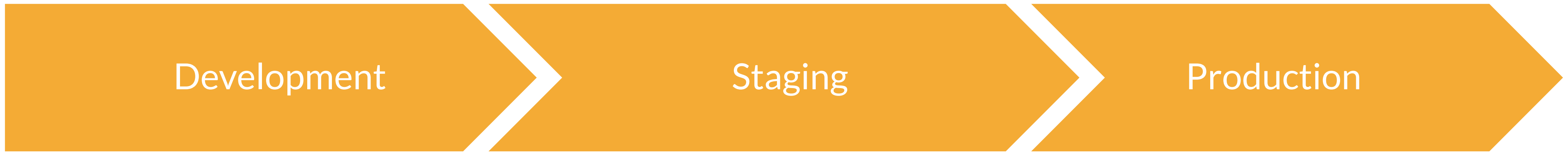
VIRTUAL MACHINEs VS DOCKER CONTAINERS



ADVANTAGES OF DOCKER FOR DEVOPS

- Docker containers are portable and sharable and, thus, solve the problems related to collaboration between development and operations teams of an organisation enabling faster software delivery.





Docker container swiftly moves through these stages in a DevOps Pipeline.

DOCKER LIMITATIONS

- Docker does not provide data storage. The files written to the container layer are not retained once the container goes off.
- Limited number of monitoring and debugging options are present with Docker. You can use Docker CLI to get the statistics, but advanced monitoring options are missing.

DOCKER VOCABULARY

Docker Image

The basis of a Docker container. It consists of multiple layers that define your container. Image is a static read-only property.

Docker Container

Runtime instance of image. By default it is read-write

Docker Engine

Creates, ships and runs Docker containers deployable on a physical or a virtual host locally, in a data centre or a cloud service provider.

Registry Service

Cloud or server-based storage and distribution service for your images.

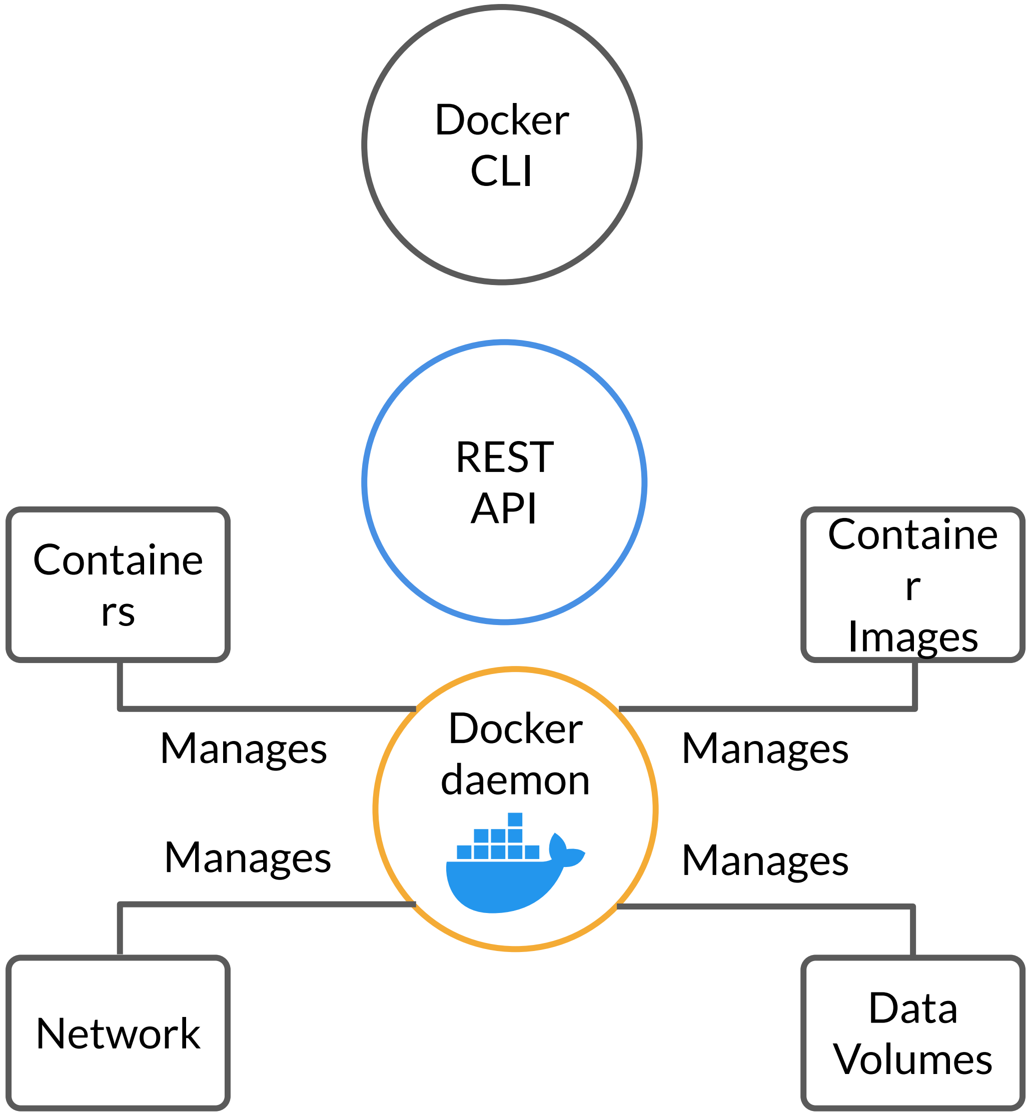
DOCKER ENGINE COMPONENTS

Docker Engine is client-server technology that **builds and runs containers** using Docker's components and services. Docker Engine needs to be installed on Docker host machine. It has multiple components as shown below.

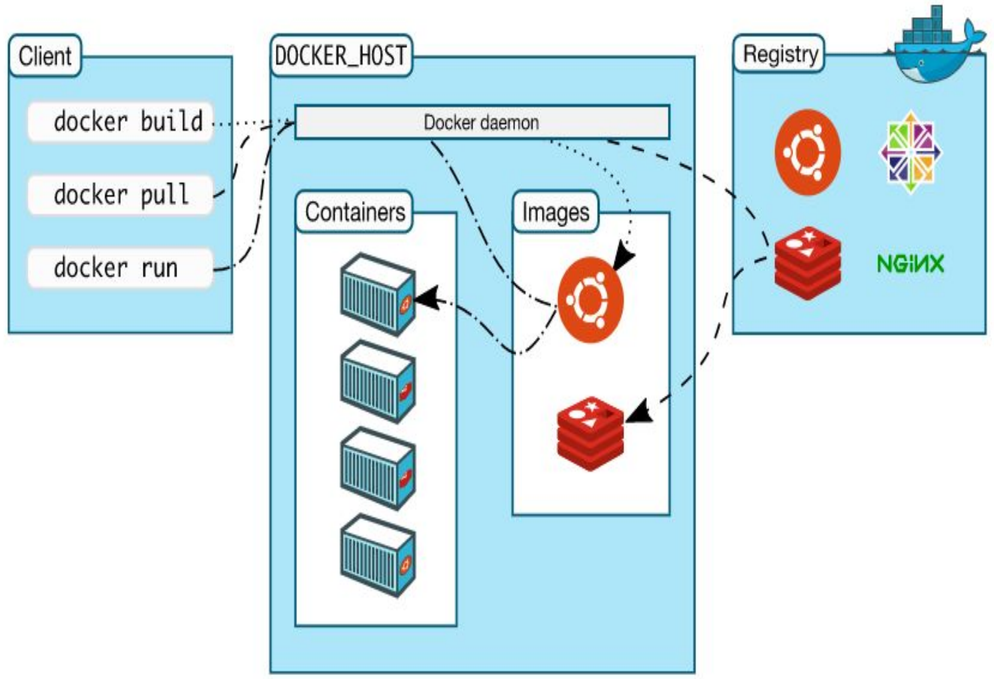
Daemon is a type of long-running program called a daemon process.

REST API specifies interfaces that programs can use to talk to the daemon and to instruct it regarding what to do.

Command Line Interface (CLI) uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands.



DOCKER ARCHITECTURE



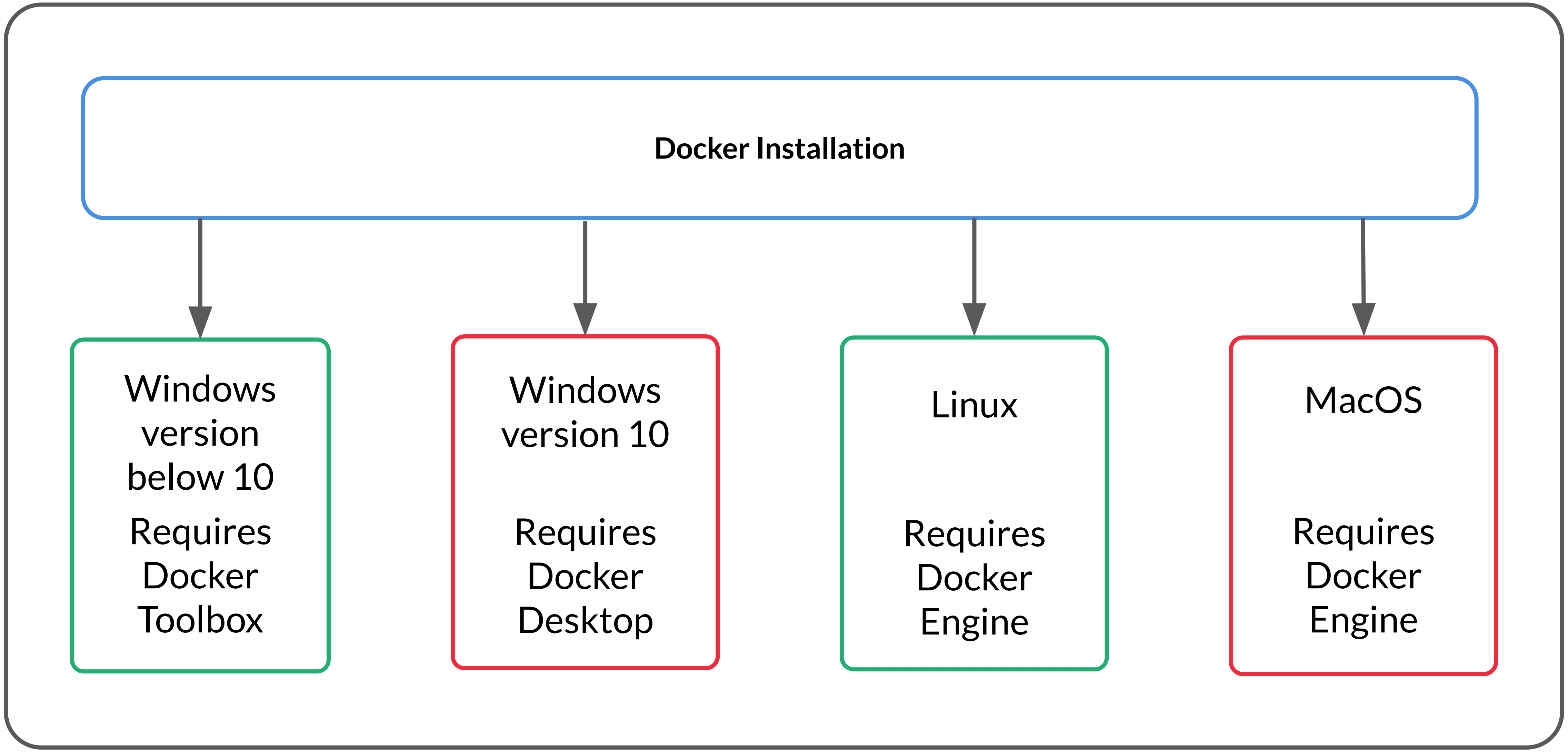
01 Docker Daemon: Runs on Docker host and handles all requests from Docker client

02 Docker Client: Interacts with Docker daemon using CLI commands

03 Docker Registry: Storage for Docker Images, which facilitates sharing of docker images

04 Docker objects: Includes Docker images, containers and volumes

DOCKER INSTALLATION



DOCKER INSTALLATION

- Docker installation is supported on all operating systems and the installation procedure mentioned at following URL can be referred: https://docs.docker.com/engine/install/

- **The docker version** command can be used to check the installation version and docker info can be used to display system wide information regarding the Docker installation. The displayed information includes the kernel version, number of containers and images.

Docker Overview and Installation

Docker Image Creation and Management

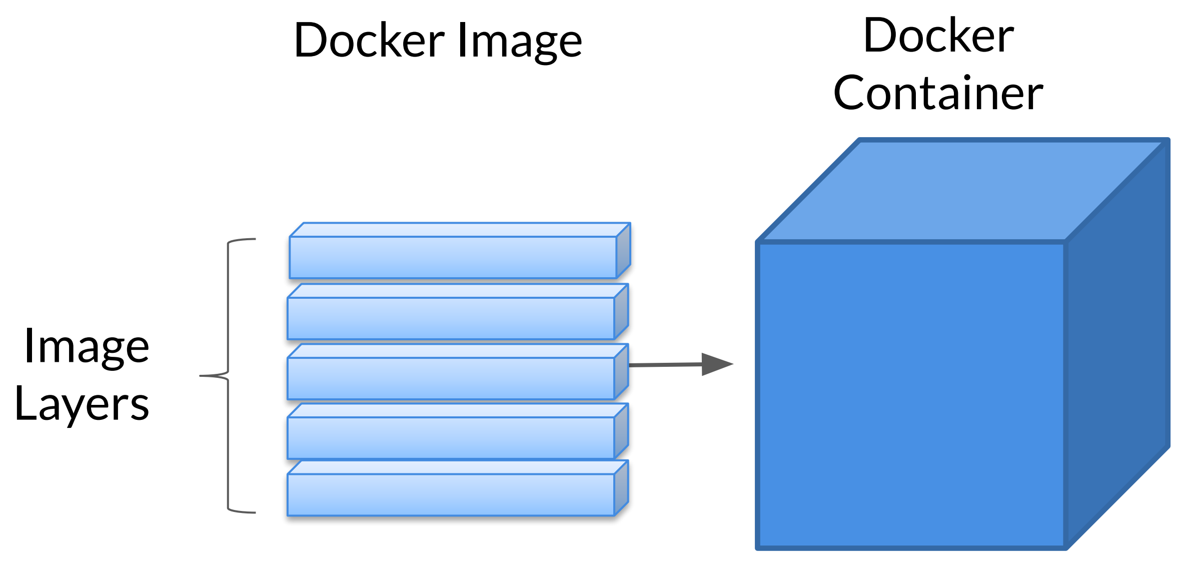
Docker Volumes and Container Deployment

SESSION OVERVIEW: DOCKER IMAGE CREATION AND MANAGEMENT

1. **Understand Docker Image**
2. **Understand Docker File and Instructions**
3. **Understand Docker Layer**
4. **Understand Docker Registry**
5. **Understand Docker Tarball Creation**

DOCKER IMAGE

- Docker Image is a set of read-only layers, where each layer indicates the actions to be performed for running Docker container.



Poll Question

Q. Docker image is a collection of read-only layers

a) True

b) False

Poll Question

Q. Docker image is a collection of read-only layers

a) True

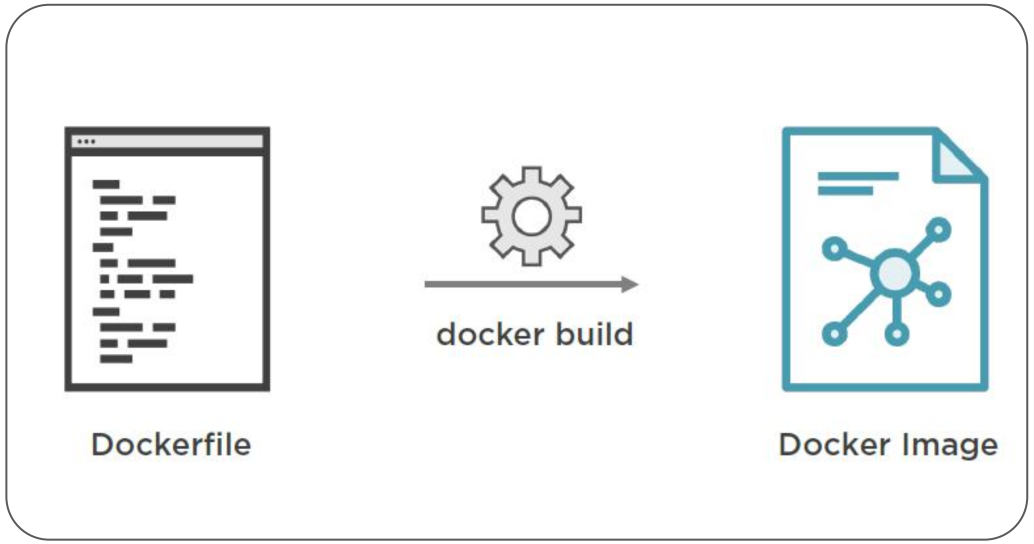
b) False

Explanation:

Docker image consists of many   read-only layers; each layers corresponds to an instruction instruction in Dockerfile.

DOCKERFILE

- Dockerfile is a text document that contains all the instructions a user could use on the command line to assemble an image. It has instructions for:
  - Inclusion of a base image
  - Addition of files or directories
  - Creation of environment variables
  - Process to run when launching container



SAMPLE DOCKER FILE

FROM: Uses a Docker image here with a jdk8 and a Linux alpine server.

MAINTAINER: Used to specify the contact information

ADD: copy the application jar in the Docker container with the name application.jar

WORKDIR: Defines the working directory of a Docker container.

ENV: Sets the value for a environment variable,here its setting PATH variable

ENTRYPOINT: Used to specify the command to execute when starting the docker container

Dockerfile

```
FROM openjdk:8-jdk-alpine
MAINTAINER upgrad
ADD build/libs/application.jar
/opt/app/application.jar
WORKDIR /opt/app
ENV PATH="${PATH}:${JAVA_HOME}/bin"
ENTRYPOINT [ "java", "-jar",
"/opt/app/application.jar"]
```

Poll Question

Q. Which of the following is not an instruction used in Dockerfile

a) ENTRYPOINT

b) FROM

c) ADD

d) REMOVE

Poll Question

Q. Which of the following is not an instruction used in Dockerfile

a) ENTRYPOINT

b) FROM

c) ADD

d) REMOVE

Explanation:

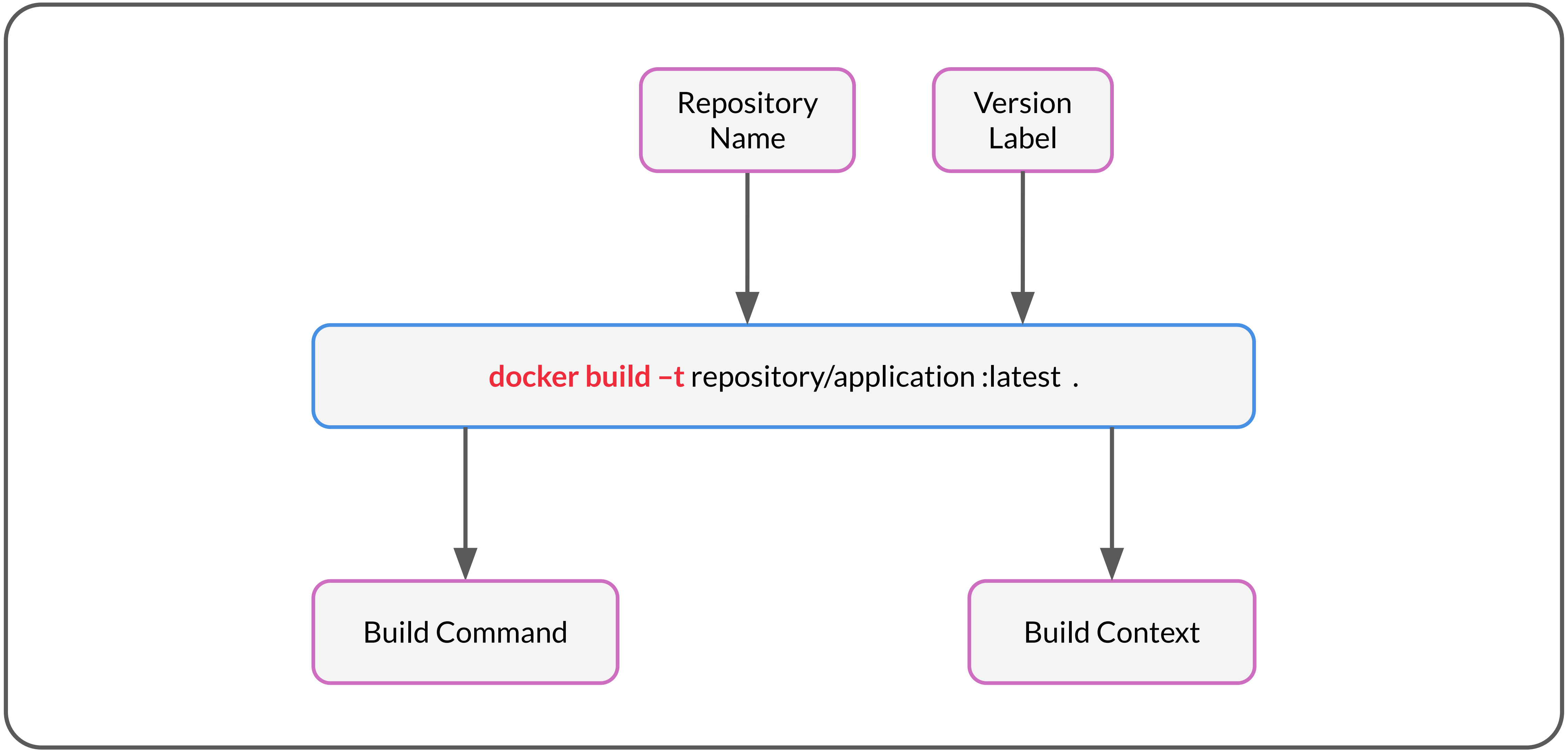
a) ENTRYPOINT: specifies a command that will be executed when the container starts. This is incorrect option

b) FROM: Include a base docker image .This is incorrect option

c) ADD: adds a file from docker host to container. This is incorrect option

d) REMOVE: this is not a docker command. This is correct option

DOCKER BUILD COMMAND USAGE



DOCKER BUILD CONTEXT

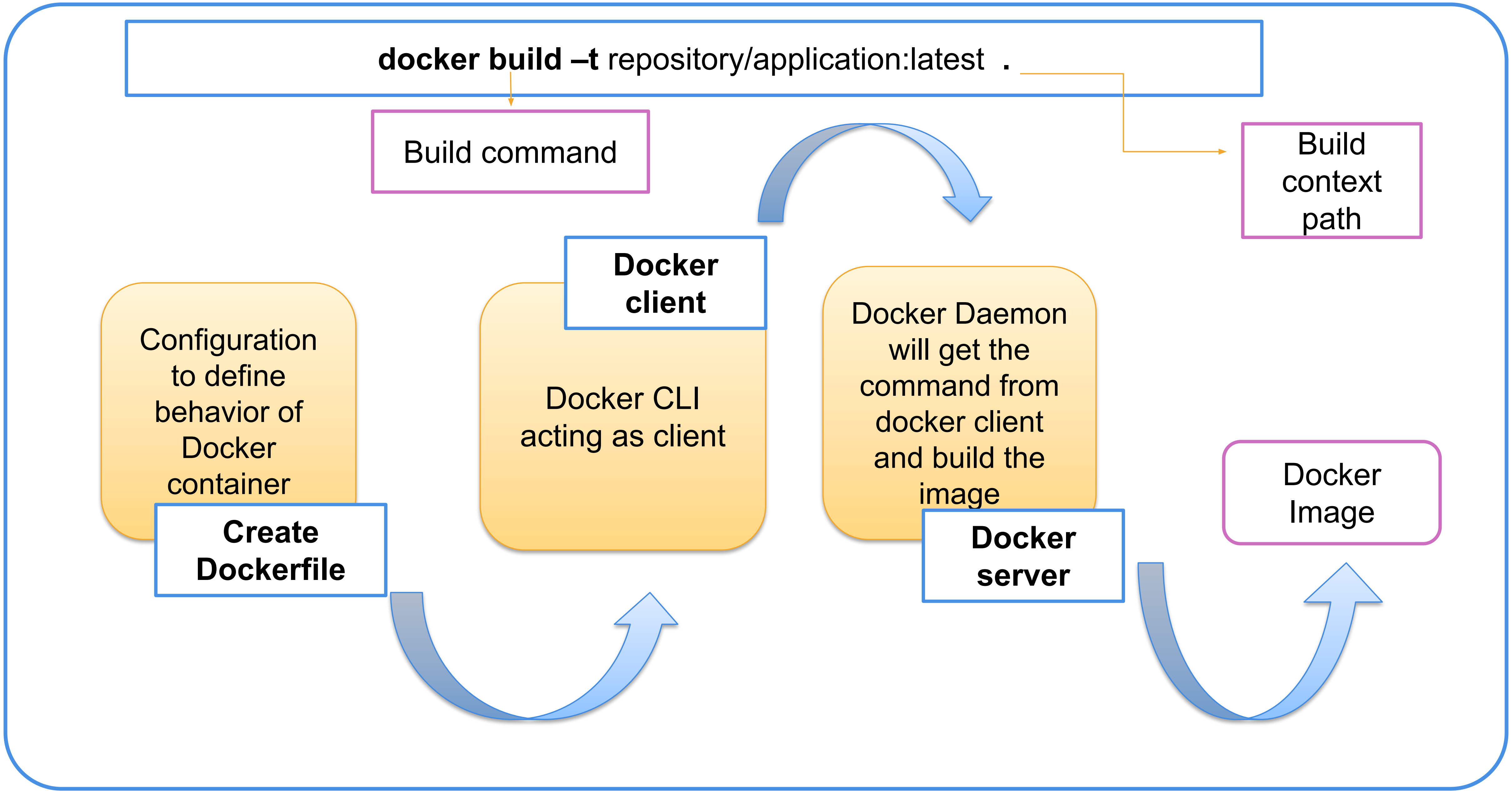
- Docker build context is the set of files located at the specific PATH on Docker host. Those files are sent to the Docker daemon during the build so it can use them in the file system of the image.

- **The docker build** command takes build context as one of the arguments.

Sample Output of **'docker build'** Command Execution

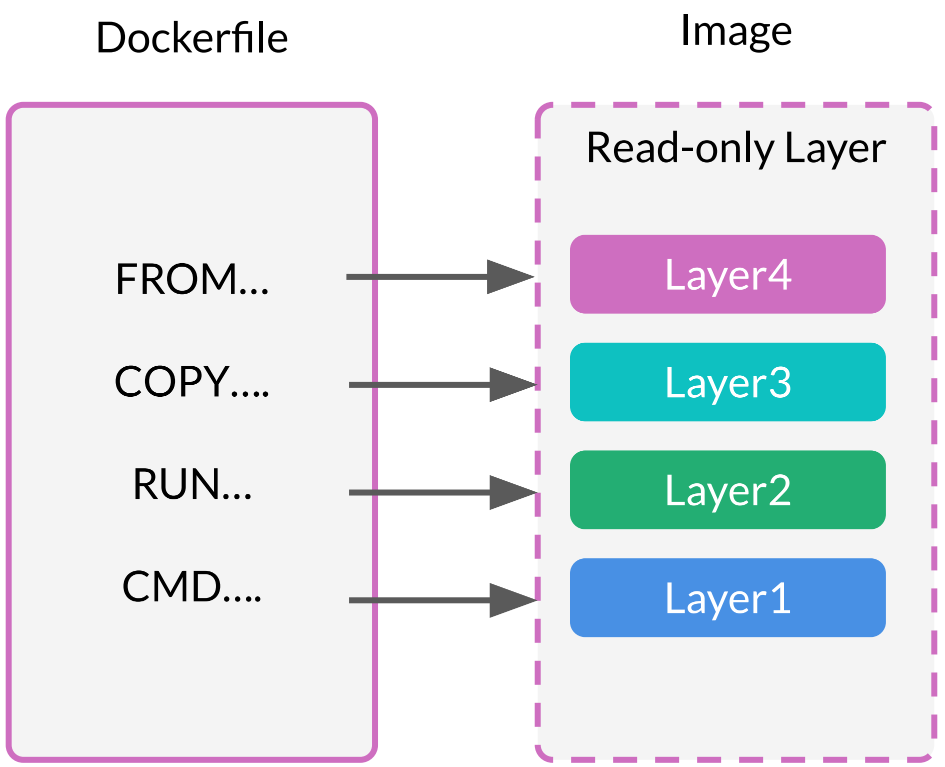
```
[opc@instance-20201221-1152 sample]$ docker build -t  example/application:latest  .
Sending build context to Docker daemon    110MB
Step 1/7 : FROM openjdk:8-jdk-alpine
 ---> a3562aa0b991
Step 2/7 : MAINTAINER example
 ---> Using cache
 ---> a96c5edf5574
Step 3/7 : VOLUME /tmp
 ---> Using cache
 ---> d9c67b9bafdd
Step 4/7 : ADD build/libs/application.jar /opt/app/application.jar
 ---> Using cache
 ---> e5646f3dd0af
Step 5/7 : WORKDIR /opt/app
 ---> Using cache
 ---> 8ca979055bfd
Step 6/7 : ENV PATH="${PATH}:${JAVA_HOME}/bin"
 ---> Using cache
 ---> 98a7f41c3919
Step 7/7 : ENTRYPOINT [ "java", "-jar", "/opt/app/application.jar"]
 ---> Using cache
 ---> 8a9eb21595a1
Successfully built 8a9eb21595a1
Successfully tagged example/application:latest
```

DOCKER BUILD OVERFLOW

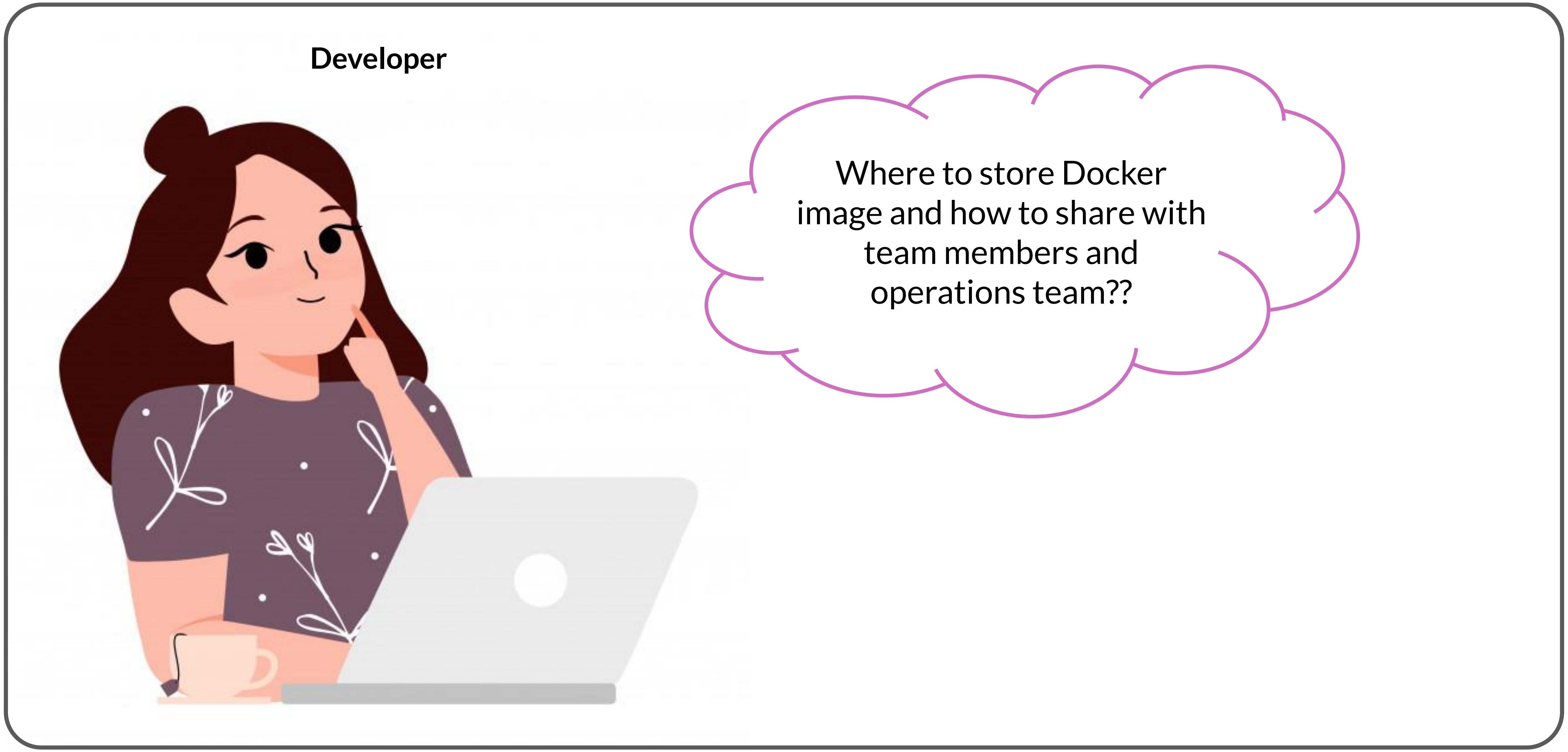

docker build –t repository/application:latest .
Build command
Build context path
Docker client
Configuration to define behavior of Docker container
Create Dockerfile
Docker CLI acting as client
Docker Daemon will get the command from docker client and build the image
Docker server
Docker Image

DOCKER LAYERS

- Docker image is made up of a series of Docker layers.
- Each layer corresponds to an instruction in Dockerfile and is read-only.
- Each layer contains the differences between the preceding layer and the current layer.
- Docker images are stored at **/var/lib/docker/overlay2** location on Linux host machine.

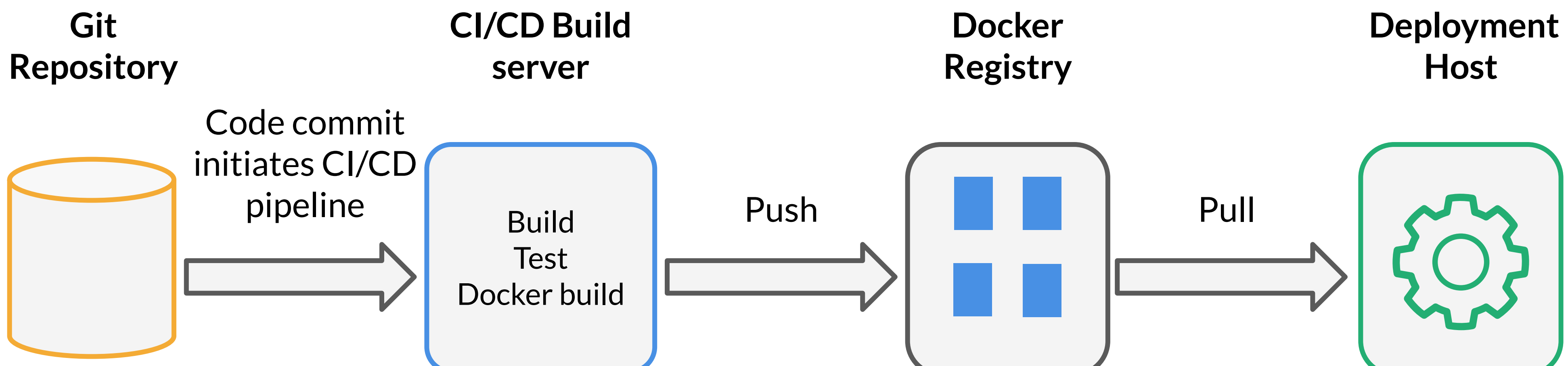


HOW TO SHARE DOCKER IMAGE?

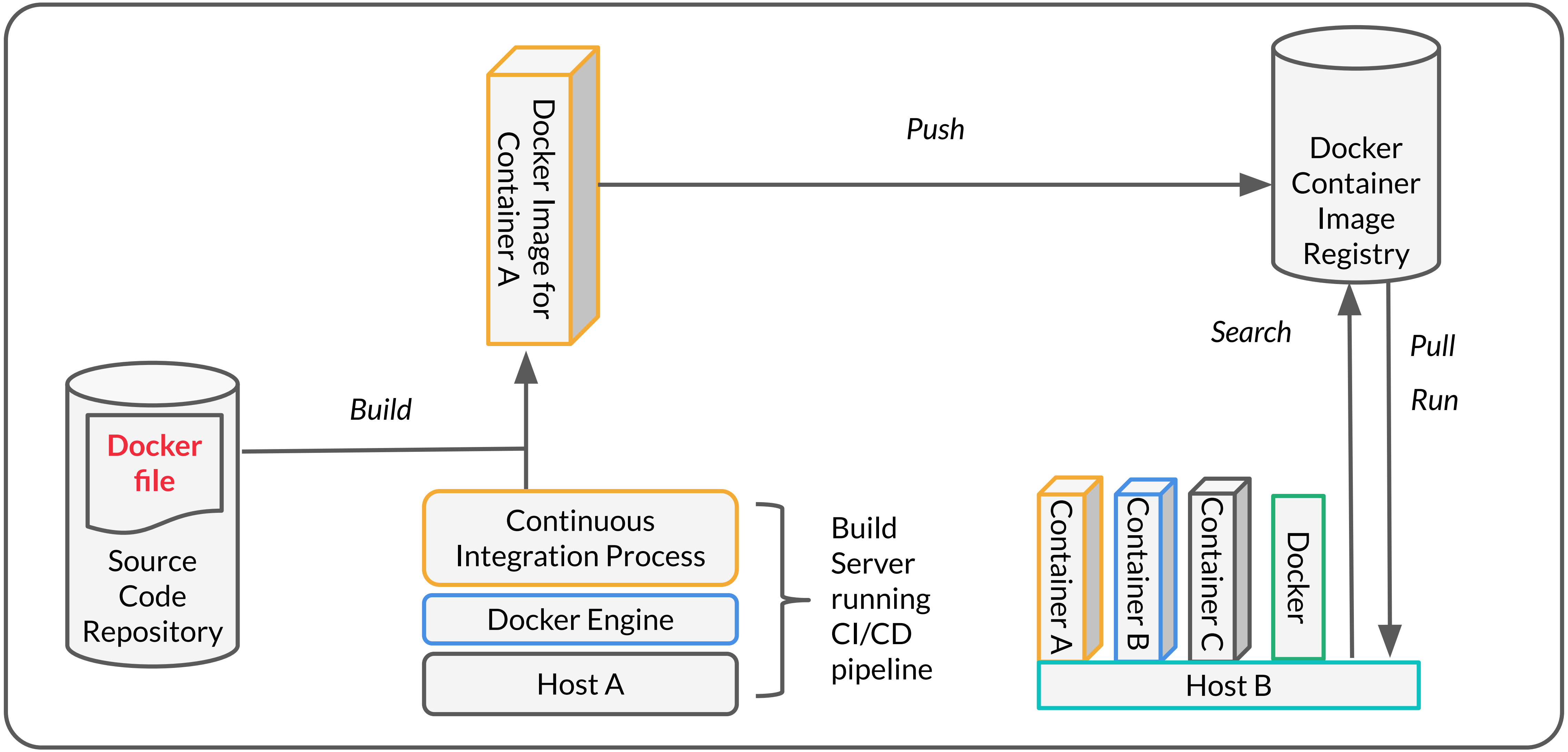

Developer
Where to store Docker image and how to share with team members and operations team??

DOCKER REGISTRY

- Docker registry is a storage system, which holds Docker images in different tagged versions. It is similar to Git repository and is used for source code management.
- It gives a way to store and share the images.
- docker push commands are used to save the image to remote registry. Similarly, docker pull command is used to fetch the image from the remote registry .
- Registry can be easily integrated with the CI/CD system.

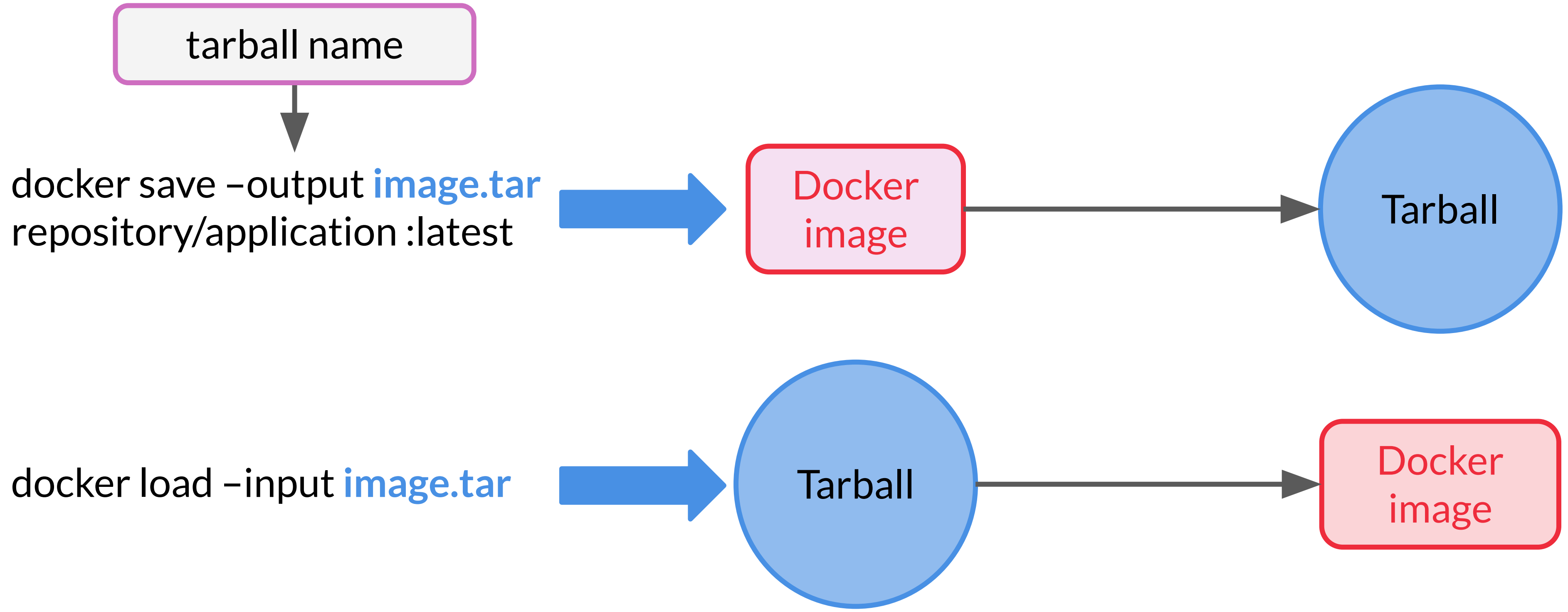


PUSH/PULL DOCKER IMAGE


Docker Image for Container A
Push
Docker Container Image Registry
Search
Pull
Run
Build
Docker file
Source Code Repository
Continuous Integration Process
Docker Engine
Host A
Build Server running CI/CD pipeline
Container A
Container B
Container C
Docker
Host B

DOCKER IMAGE AS TARBALL

- Apart from Docker registry, Docker images can be shared by creating a tarball.
- The Docker image could be shared in this manner if Docker registry is not set up or if it is temporarily inaccessible to the deployment host.



```
docker save –output image.tar
repository/application :latest
```

```
docker load –input image.tar
```

DAY1 SUMMARY: CONTAINERISATION

1. Evolution of Docker
2. Docker vs. Virtual Machine
3. Docker Advantages
4. Writing a Dockerfile
5. How to Build and Share Docker Images

Thank You.

Docker Overview and Installation

Docker Image Creation and Management

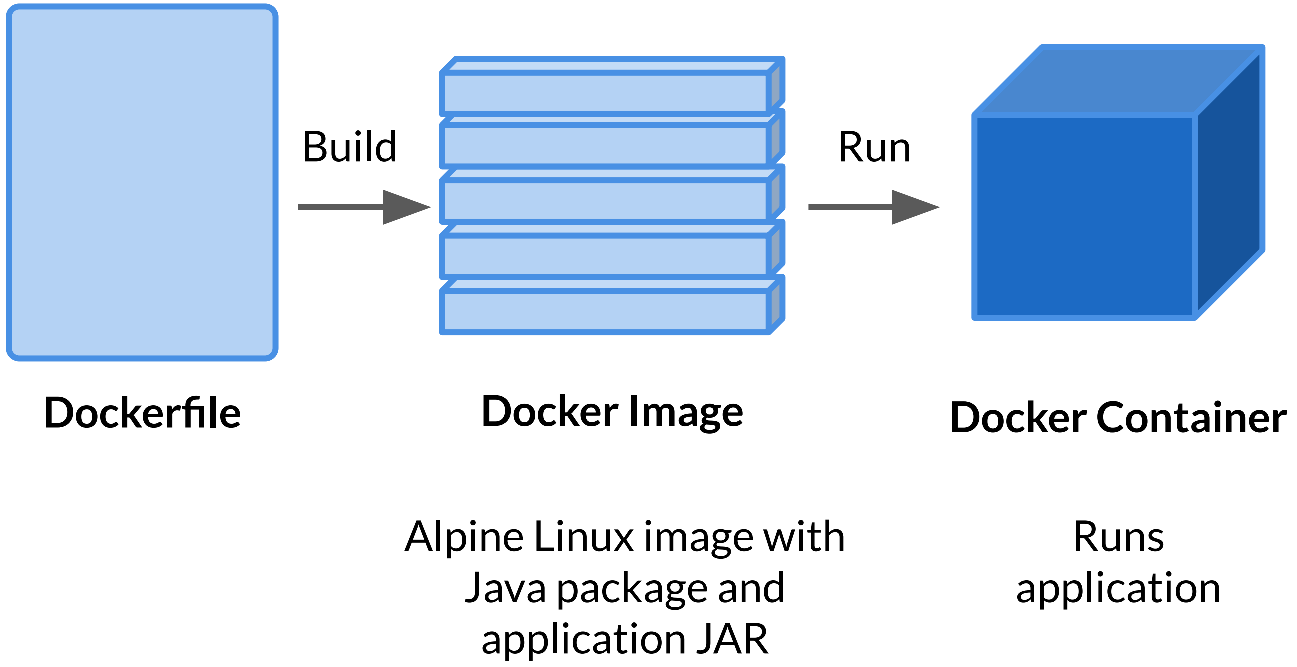
Docker Volumes and Container Deployment

SESSION OVERVIEW:
DOCKER VOLUMES AND CONTAINER DEPLOYMENT

1. Understand Docker Container Deployment
2. Understand Multi-container Deployment
3. Understand Docker-compose YAML File
4. Understand Docker Volume
5. Understand Docker Networking

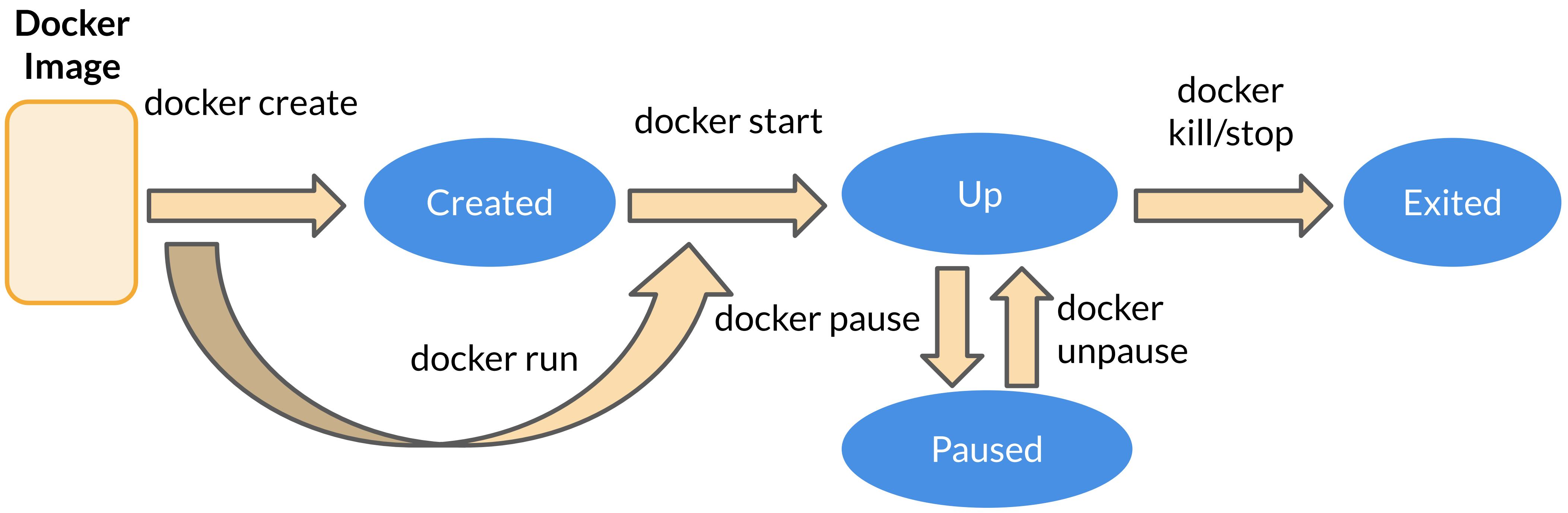
DOCKER CONTAINERS

- Docker containers are running instance of Docker images.
- **docker run** is the basic command used to launch the container. It provides various command options to specify the deployment characteristics, such as container name, host port and environment variables.



DOCKER CONTAINERS LIFECYCLE

- Docker containers goes through different stages during its lifespan. Docker commands are used to change the state of Docker container.



Poll Question

Q. Which of the following commands is used to suspend a running container?

a) docker stop

b) docker kill

c) docker pause

d) docker rm

Poll Question

Q. Which of the following commands is used to suspend a running container?

a) docker stop
b) docker kill
c) docker pause
d) docker rm

Explanation:

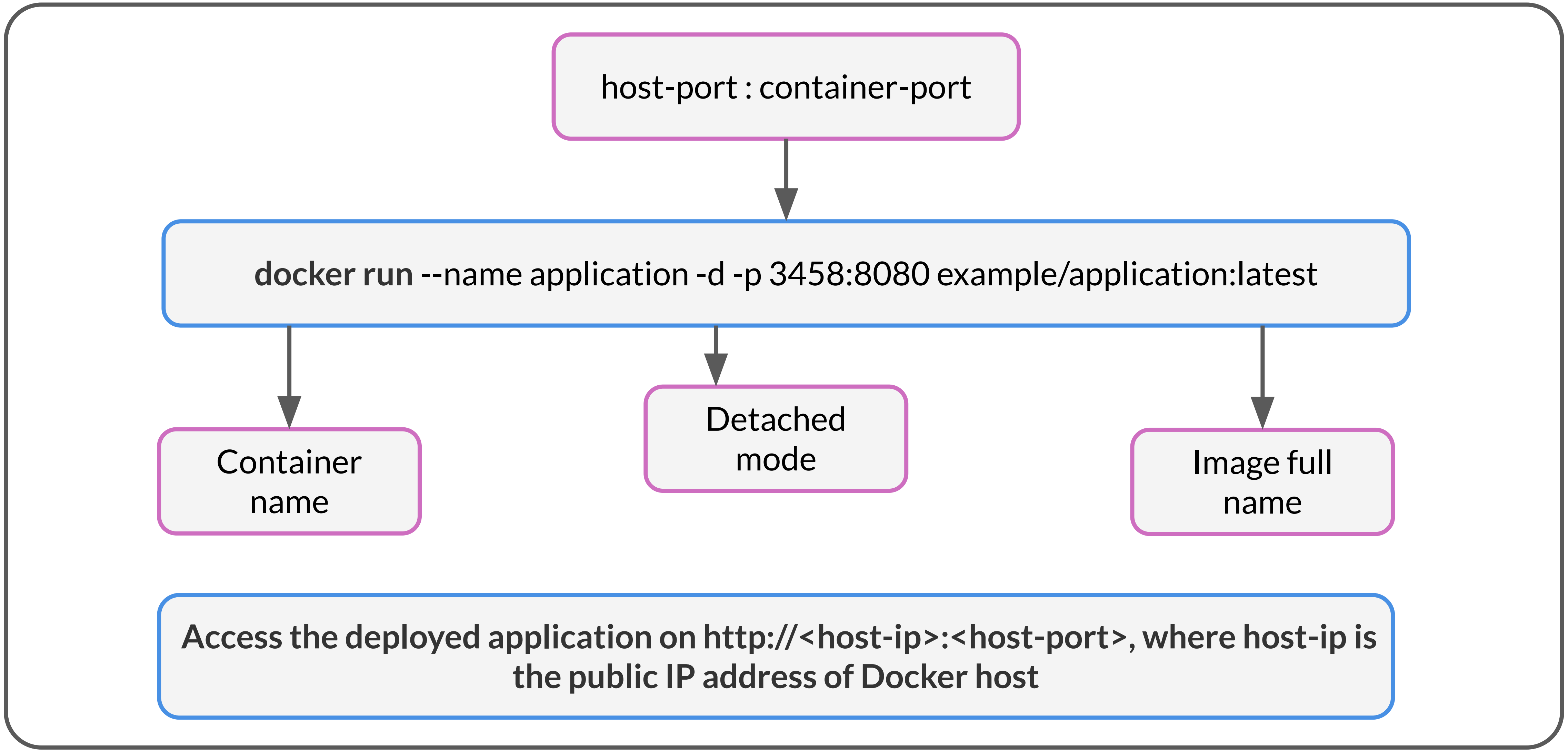
a) docker stop command is used to stop a running container; it sends SIGTERM and then sends SIGKILL after sometime to ensure graceful shutdown of the containerized application. So this is incorrect option

b) docker kill command is used to kill a running container; it sends a SIGKILL to a docker container. Therefore, this is incorrect option.

c) docker pause is used to suspend all processes in specified container and could be used to pause a container rather than stopping the container completely. **'docker pause'** & **'docker unpause'** are the commands particularly useful in case where a container needs to be suspended for a small duration just for debugging purpose. Therefore, this is the correct option

d) docker rm is used to remove an already stopped docker container. So this is an incorrect option

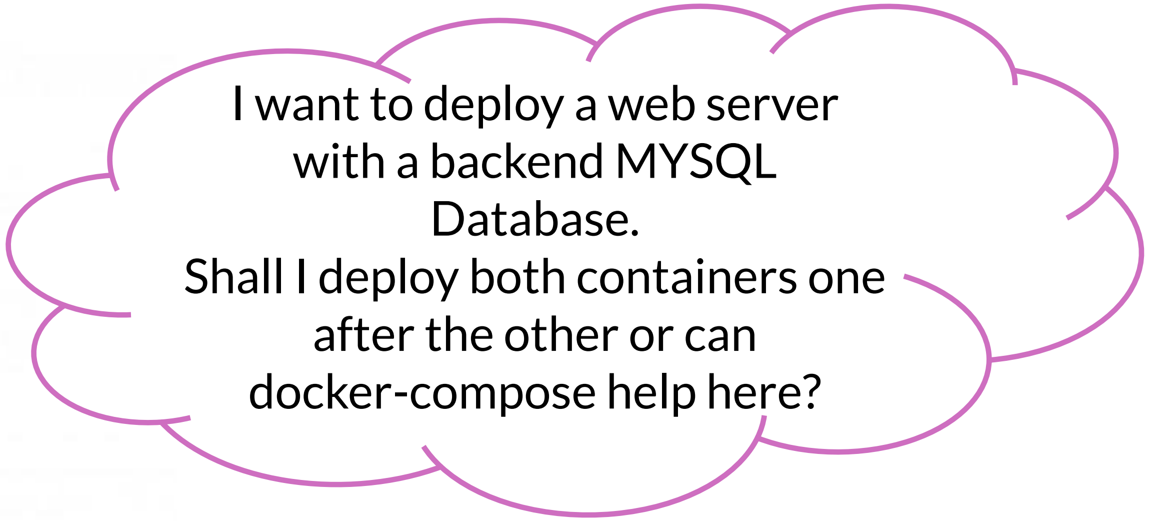
DOCKER RUN COMMAND USAGE



```
docker run --name application -d -p 3458:8080 example/application:latest
```

Access the deployed application on http://<host-ip>:<host-port>, where host-ip is the public IP address of Docker host

MULTI-CONTAINER DEPLOYMENT

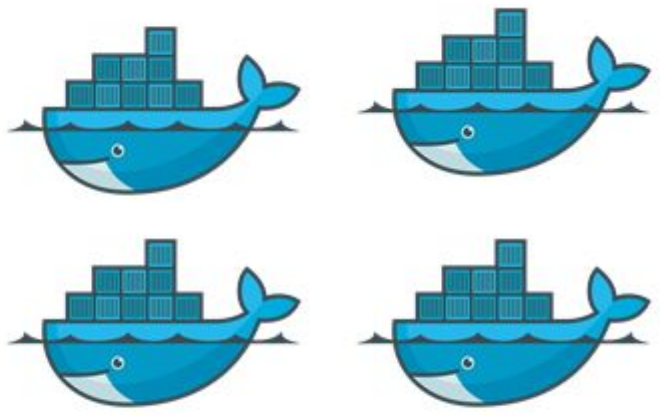

I want to deploy a web server with a backend MYSQL Database.
Shall I deploy both containers one after the other or can docker-compose help here?

DOCKER COMPOSE

- Docker Compose is a tool that is used for defining and running multi-container Docker applications. Docker-compose YAML file is used to configure your application's services. Then, with a single command, you create and start all the services from your configuration.
- Docker Compose Installation:

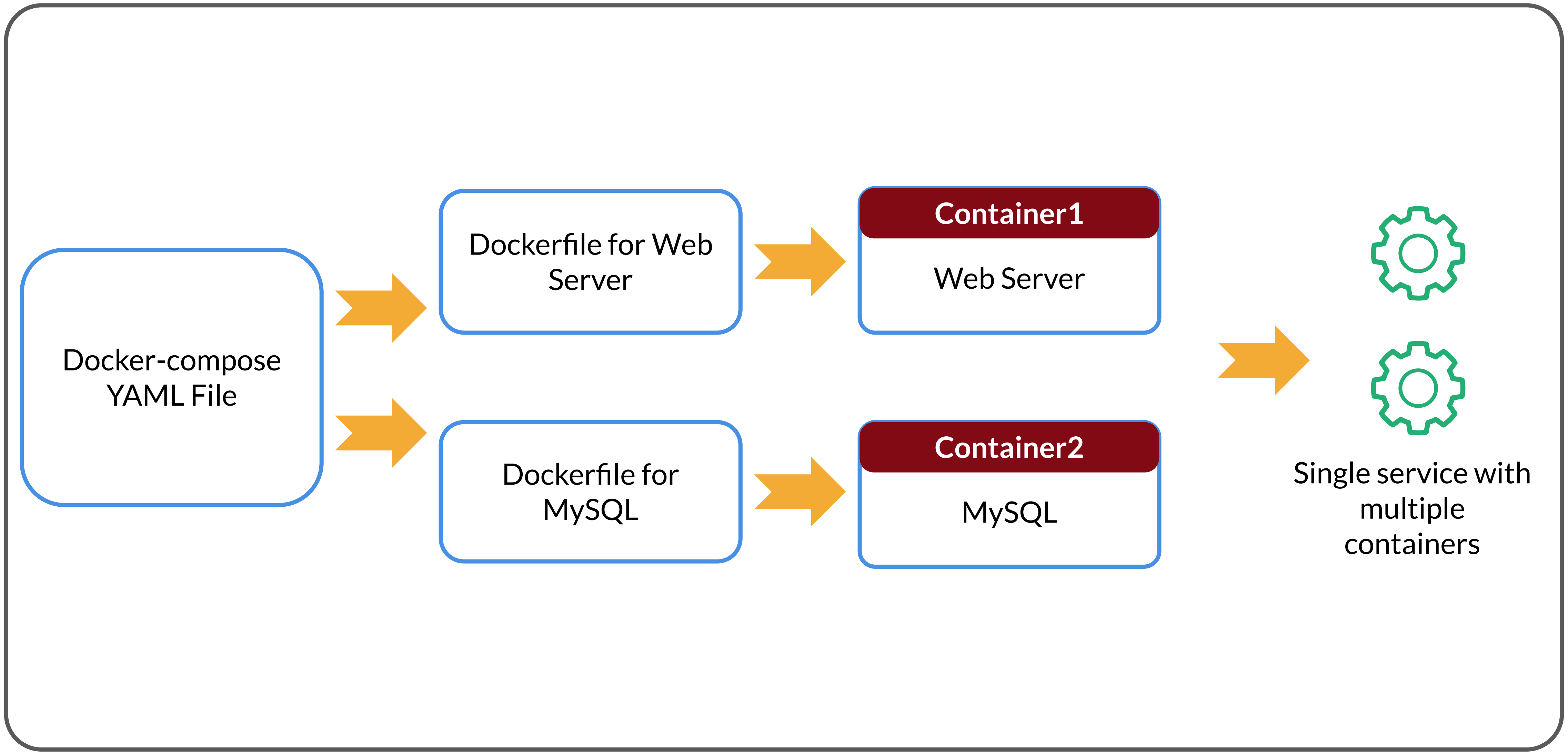
Windows and MacOS → Docker Compose is included in Docker Desktop

Linux → Follow the following link for installation: https://docs.docker.com/compose/install/



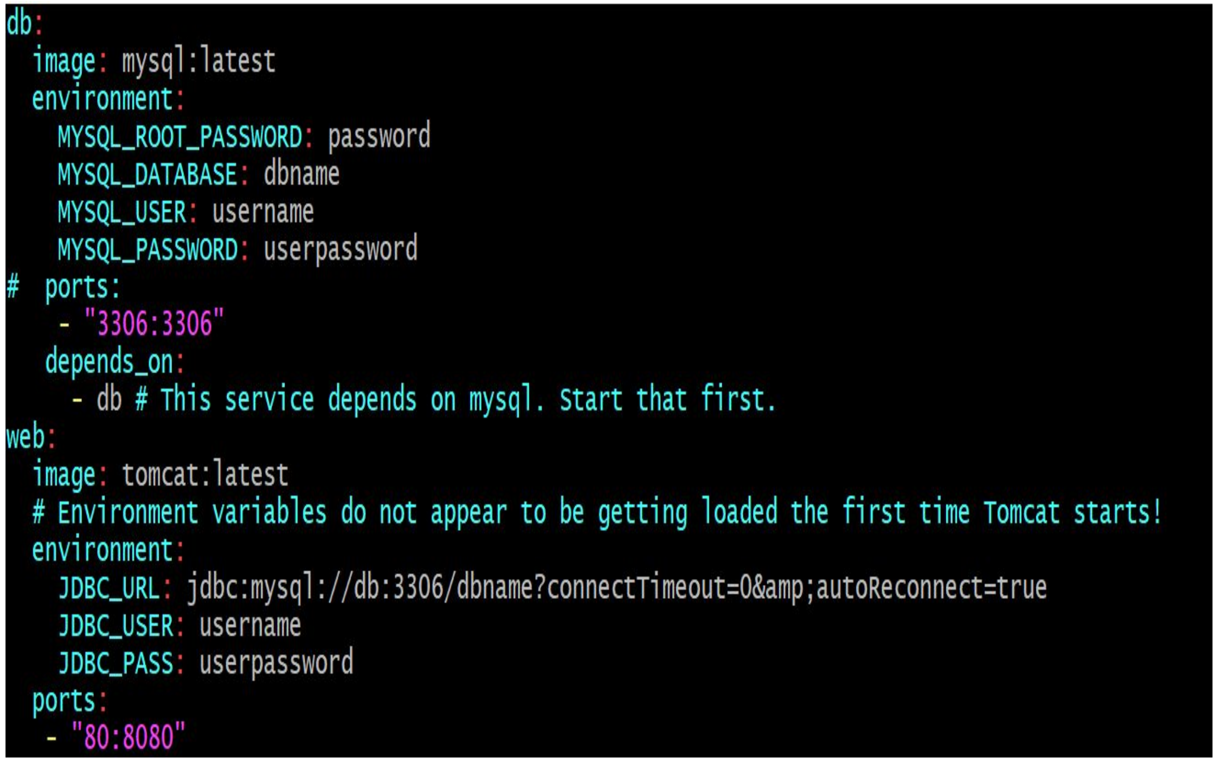
Docker-compose

DOCKER-COMPOSE WORKFLOW (WEB SERVER + MYSQL)



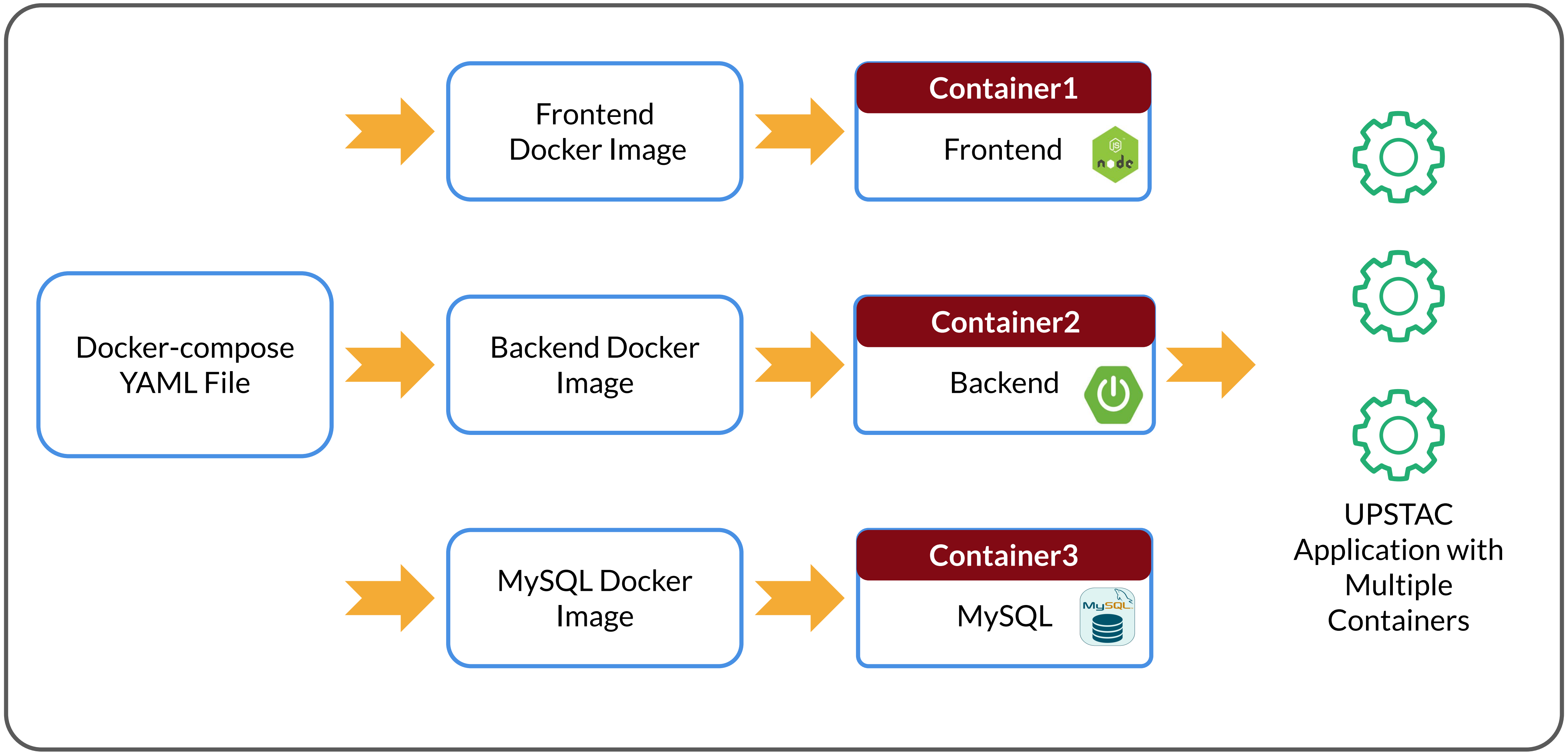
SAMPLE DOCKER-COMPOSE.YML (WEB SERVER + MYSQL)

- The **db** and **web** keywords are used to define two separate services.
- The **image** keyword is used to specify the docker images of MySQL and Tomcat web server.
- The **ports** keyword is used to mention the mapping of the container port to the host machine's port where the service is exposed.
- The **version** keyword indicates the version of Docker-compose being used.
- The **build** keyword indicates the location of Dockerfile of the service.



```yaml
db:
  image: mysql:latest
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: dbname
    MYSQL_USER: username
    MYSQL_PASSWORD: userpassword
#  ports:
    - "3306:3306"
   depends_on:
     - db # This service depends on mysql. Start that first.
web:
  image: tomcat:latest
  # Environment variables do not appear to be getting loaded the first time Tomcat starts!
  environment:
    JDBC_URL: jdbc:mysql://db:3306/dbname?connectTimeout=0&amp;autoReconnect=true
    JDBC_USER: username
    JDBC_PASS: userpassword
  ports:
   - "80:8080"
```

DOCKER-COMPOSE WORKFLOW FOR UPSTAC APPLICATION



BASIC DOCKER-COMPOSE COMMANDS

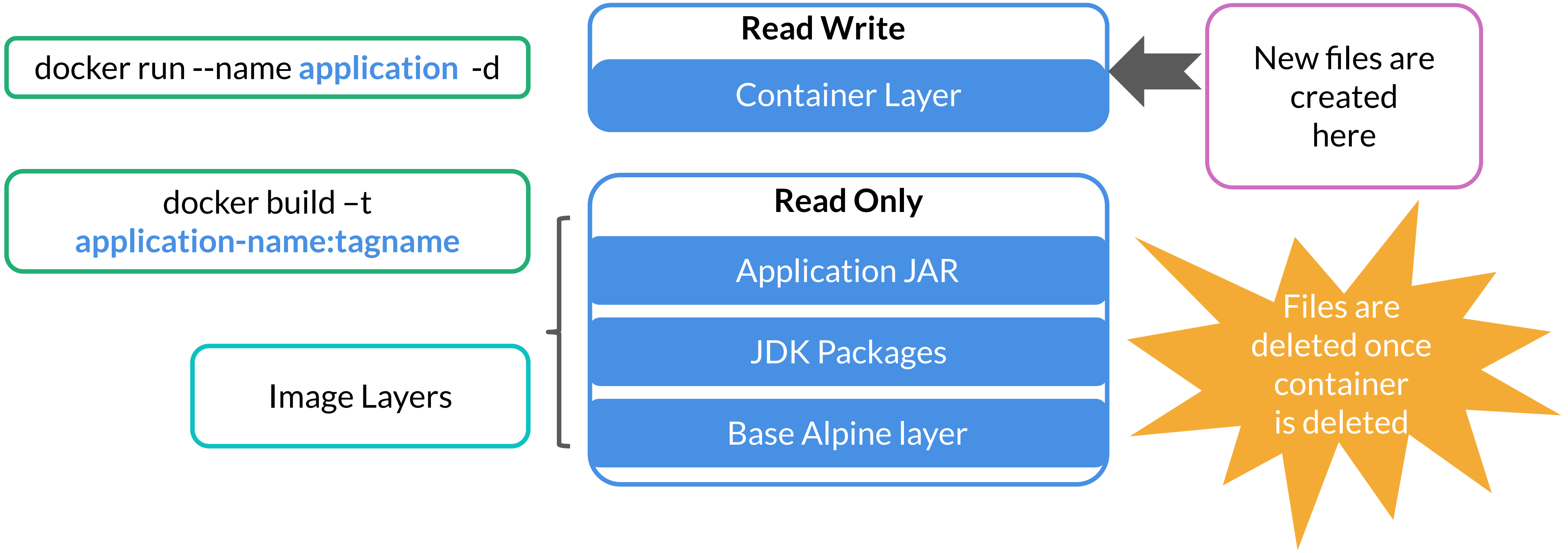
Command	Description
docker-compose up	Starts all the containers
docker-compose ps	Can be used to verify the status of running containers
docker-compose stop	Can be used to stop the containers
docker-compose logs	Can be used to check the logs of the containers
docker-compose down	Can be used to remove the containers

DOCKER-COMPOSE ADVANTAGE

- Single command to deploy a multi-container application based on container definitions in the docker-compose YAML file.
- Docker-compose has a range of commands for the management of application lifecycle.
- Persistent storage can also be specified for application containers.

STORAGE IN DOCKER CONTAINER

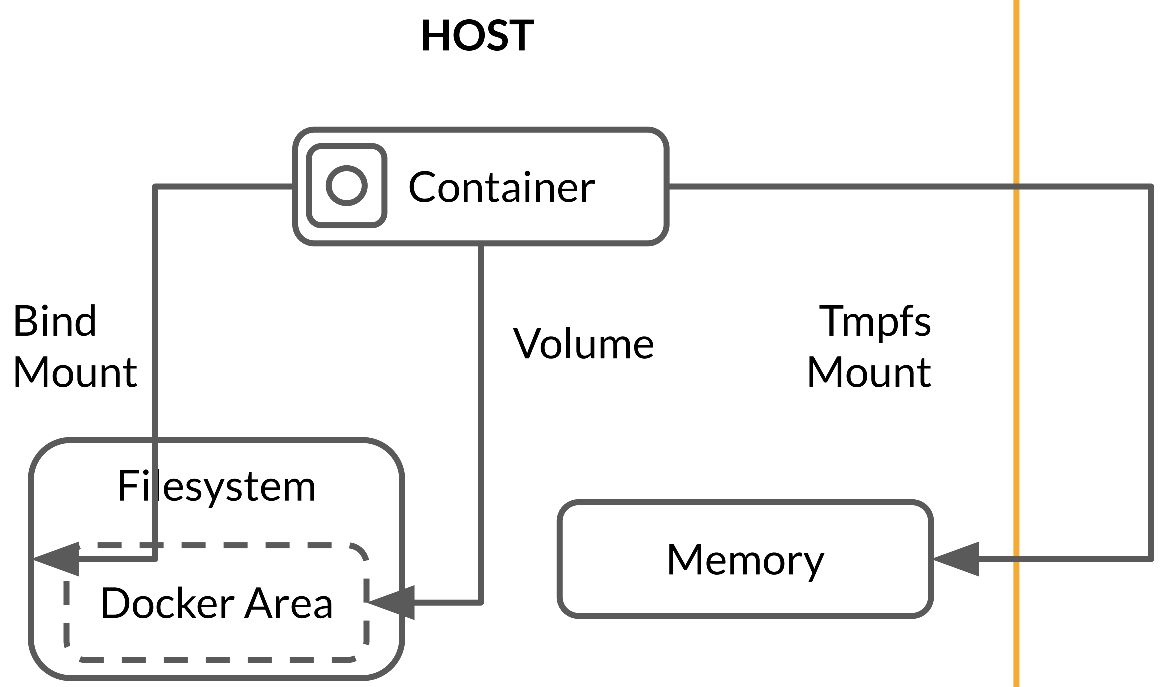
- Any file/directory created inside a container persists only till the time container is alive.


docker run --name application -d
docker build –t application-name:tagname
Image Layers
Read Write
Container Layer
Read Only
Application JAR
JDK Packages
Base Alpine layer
New files are created here
Files are deleted once container is deleted

STORAGE IN DOCKER CONTAINER

STORAGE IN DOCKER CONTAINER

- All files created inside a container are stored on a writable container layer and they persist only till the time the container is alive.
- **Volumes** and **bind** mounts are most commonly used options to store data on a host machine . Apart from these two options, **tmpfs** mount is possible on Linux host machine and, in case of Windows, the **named pipe** option can be used.
- **tmpfs** mounts are stored in the host system's memory only, and are never written to the host system's file system.
- **Volumes** is the best way to persist data in Docker container and are stored at path/var/lib/docker/volumes/ on Linux. On the other hand, **bind** mounts may be stored anywhere on the host system.



Poll Question

Q. Identify if the following statement is True or False:

Statement: “/var/lib/docker/volumes” location is used to store the docker volume on linux operating system ?

a) True

b) False

Poll Question

Q. Identify if the following statement is True or False:

Statement: “/var/lib/docker/volumes” location is used to store the docker volume on linux operating system ?

a) True

b) False

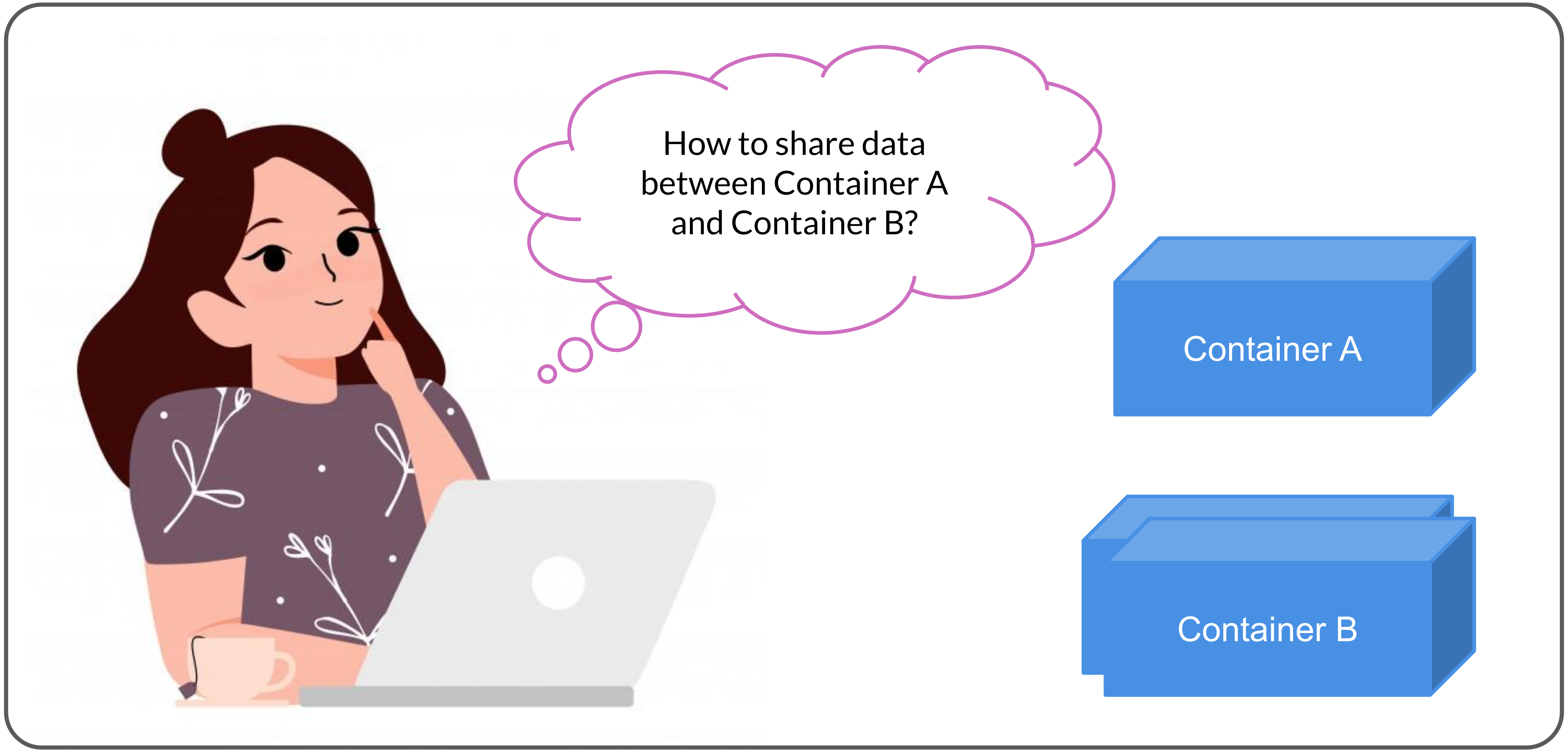
Explanation:

“/var/lib/docker/volumes” location is used to store the docker volumes on docker host with linux operating system.

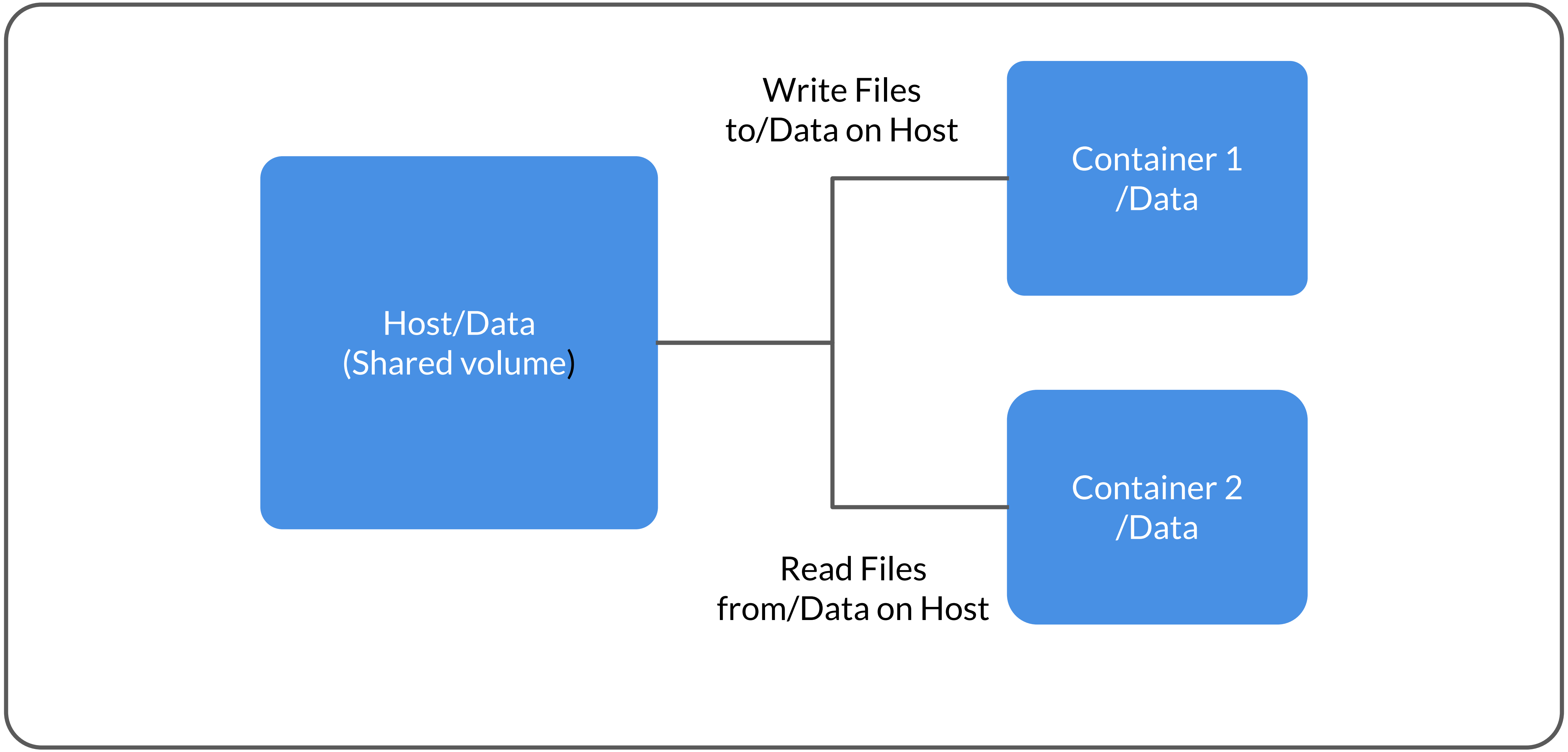
DOCKER VOLUME

- Docker volume can be used for the long-term storage of your container data by mapping a directory in the container to a directory on the host machine.
- Docker volume can also be used to share data among containers.
- Docker volumes significantly reduce the chances of data loss due to a failed container.
- Data is available on host machine even when a container is not alive. Logs and backups of the application container can be stored in data volumes.

SHARE DATA BETWEEN DOCKER CONTAINERS



DOCKER VOLUME - SHARE DATA BETWEEN CONTAINERS



WAYS TO CREATE DOCKER VOLUME

- Docker Volume can be created by using docker command or can be created implicitly during container or service creation

01 Create data volume while running service

docker run --name application -d -p 3458:8080 –v sharedvolume:/data example/application: latest

sharedvolume :data volume on host machine

/data is mounted in the container.

WAYS TO CREATE DOCKER VOLUME

02 Create data volume and mount on container

sharedvolume :data volume on host machine

sharedvolume :data volume on host machine

```
docker volume create sharedvolume

docker run --name application  -d -p 3458:8080 –mount source=sharedvolume
,destination=/data example/application: latest
```

/data. Path in the container

DOCKER NETWORKING

How Docker containers communicate with each other and with external world?

DOCKER NETWORKING

- Docker installation creates three networks by default bridge (named **docker0**), Host and None. Apart from these three, there are two additional networks as well, that is, Overlay and Macvlan.
- Bridge networks are used when applications run in standalone containers that need to communicate and the default network type is used by the containers unless specified explicitly using docker run –net <NETWORK> option.
- Host networks are used for standalone containers, removing network isolation between the container and the Docker host and directly using the host’s networking. For instance, a container that binds to port 80 and where Docker network is the host, the container’s application is available on port 80 on the host’s IP address.
- Launching the container with None network disables networking stack on a container, that is, eth0 is not created on container.
- Overlay network is used when it is required to run containers on different Docker hosts. Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other.
- Macvlan assigns a MAC address to a container so that it acts as a physical device on the network. Docker daemon routes traffic to containers using their MAC addresses.

DOCKER BRIDGE NETWORK

- The **docker0** bridge network is the default network used by the containers. It uses a default private subnet **172.17.0.0/16** for container networking, with 172.17.0.1 as a default gateway.
- When a container is launched, a virtual Ethernet device (veth) is created on docker0 bridge and it maps to eth0 in a container that is assigned a private IP address on docker0 network.
- Containers communicate with each other via docker0 bridge. Docker keeps a mapping of the container name and its IP address. This allows communication using container name against an IP address.
- Docker uses port forwarding to map the traffic between container IP address and specific port and host IP address and port. For this, every time a Docker container is launched, new NAT rules are created for routing the traffic from host IP address and port to container IP address and port.

Poll Question

Q. docker0 bridge is created when docker engine is installed on docker host

a) True

b) False

Poll Question

Q. docker0 bridge is created when docker engine is installed on docker host

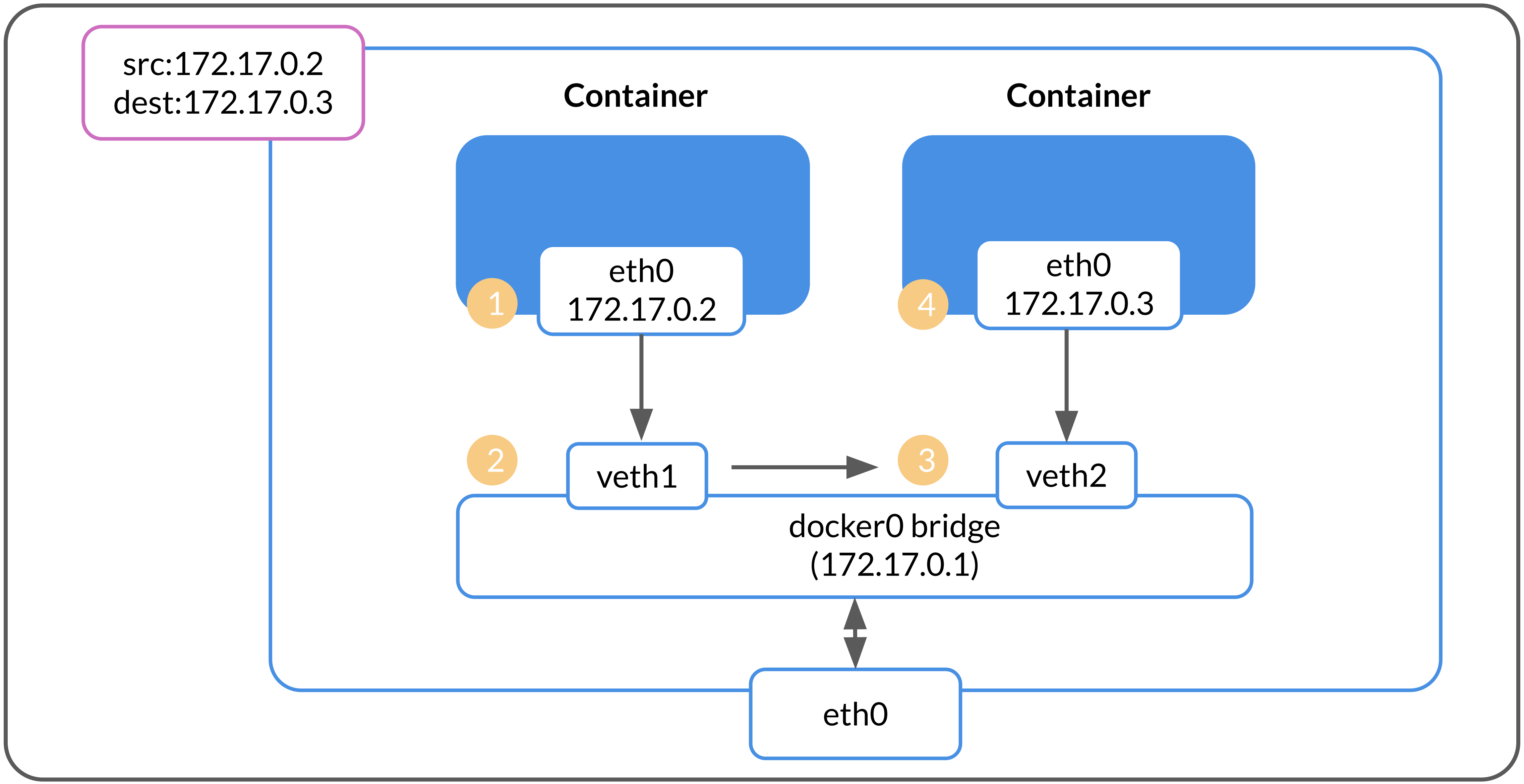
a) True

b) False

Explanation:

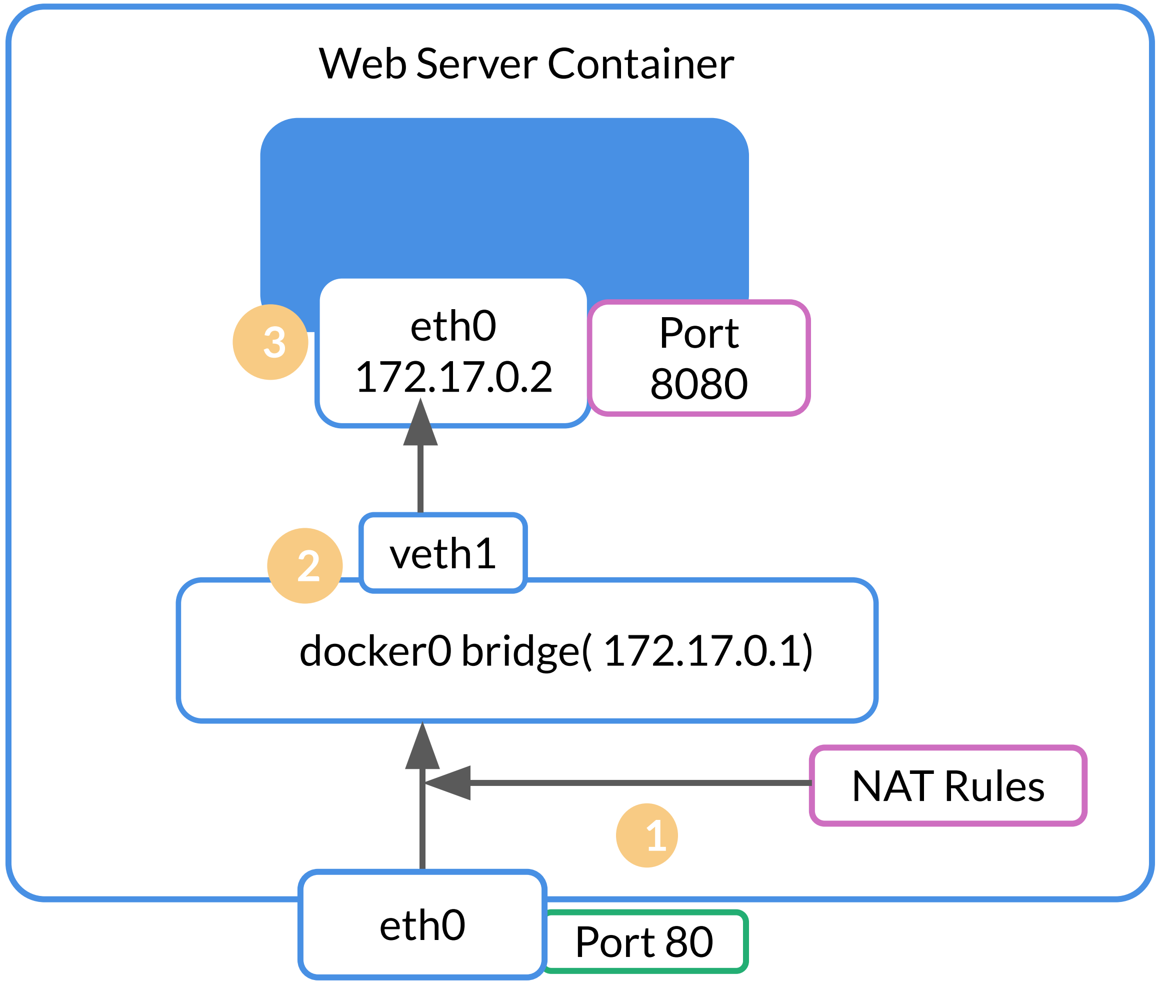
On docker installation, docker0 Linux bridge is created on the host machine

CONTAINER TO CONTAINER COMMUNICATION – BRIDGE NETWORK



CONTAINER COMMUNICATION WITH OUTSIDE WORLD - BRIDGE NETWORK

- Packets received on host machine are forwarded to docker0 interface after evaluation of NAT rules.
- Further, with the help of NAT rules, destination IP address (Docker host IP) and port (80) of packet is changed to the IP Address 172.17.0.2 and port (8080) of the container.
- Thereafter, veth tunneling is used to send the packets from docker0 interface (172.17.0.1) to container eth0 interface (172.17.0.2).
- Web server at port 8080 answers all the requests and sends the response through the same path in the opposite direction.



Poll Question

Q. Which of the following is not a docker network type.

a) host

b) none

c) Overlay

d) tmpfs mount

Poll Question

Q. Which of the following is not a docker network type.

a) host

b) none

c) Overlay

d) tmpfs mount

Explanation:

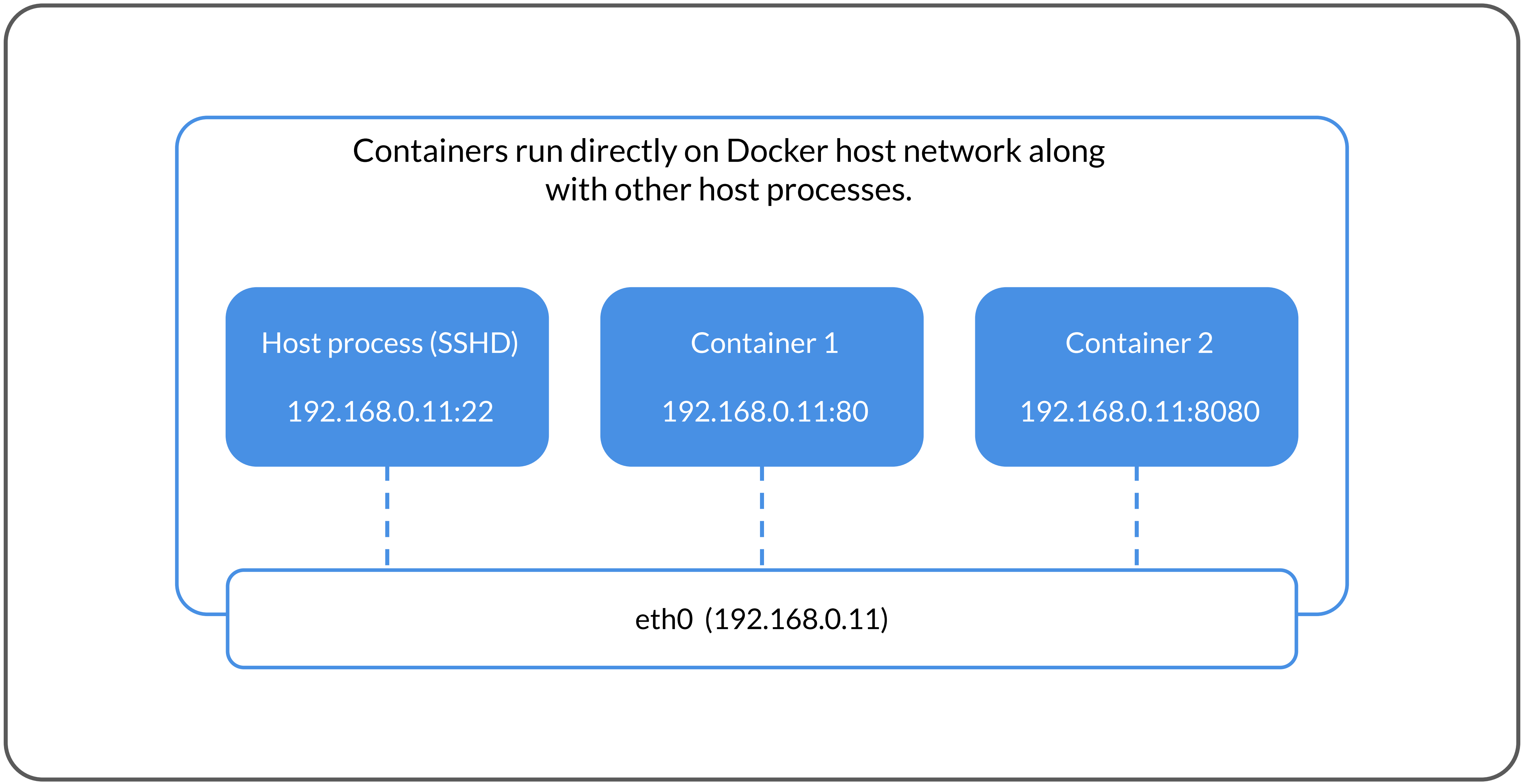
a) host is a type of docker network. This is an incorrect option

b) none is a type of docker network. This is an incorrect option

c) overlay is a type of docker network. This is an incorrect option

d) tmpfs mount is the temporary storage for persisting the container data. This is the correct option

DOCKER HOST NETWORK



MODULE SUMMARY: CONTAINERISATION USING DOCKER

1. **Evolution of Docker**
2. **Docker Advantages**
3. **How to Build and Share Docker Images**
4. **Deployment of Docker Container**
5. **Multi-container Application Deployment**

Thank You.