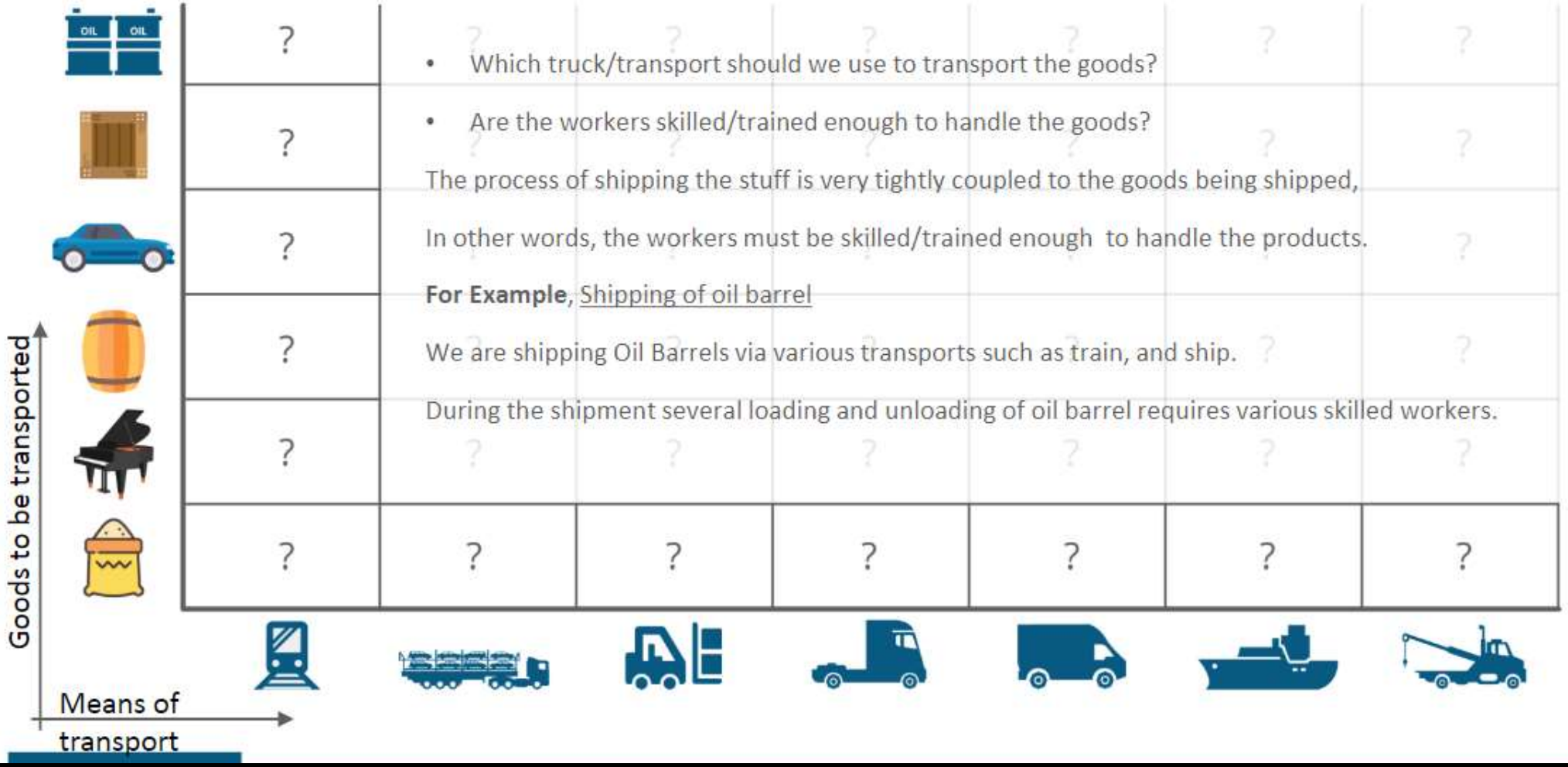
INTRODUCTION TO DOCKER

Objectives

- Shipping Transportation Challenges
- Introducing Docker
- Docker Terminology
- Architecture of Docker
- Running Hello World in Docker
- Introduction to Container
- Container Life Cycle
- Sharing and Copying

SHIPPING TRANSPORTATION CHALLENGES

Shipping Challenges



- Which truck/transport should we use to transport the goods?
- Are the workers skilled/trained enough to handle the goods?

The process of shipping the stuff is very tightly coupled to the goods being shipped,

In other words, the workers must be skilled/trained enough to handle the products.

For Example, Shipping of oil barrel

We are shipping Oil Barrels via various transports such as train, and ship.

During the shipment several loading and unloading of oil barrel requires various skilled workers.

Shipping Solution contd…

Multiplicity of Goods

A standard sized container is loaded with goods and sealed until it reaches its final destination

Do I worry about how goods interact (e.g. coffee beans next to spices)

How smooth the operations such as loading, unloading of the Container is done efficiently over long distance

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Multiplicity of methods for transporting/storing

Shipping Solution

What's inside the container doesn't matter

Challenges with System/Software Configurations

Software / Services	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's Laptop
Static Website	?	?	?	?	?	?
Web forntend	?	?	?	?	?	?
Background workers	?	?	?	?	?	?
User DB	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?
Queue	?	?	?	?	?	?

- Which system should be used to run the software/service ?
- Are the employees skilled/trained enough (as a programmer) to handle the service/software properly with required configuration?

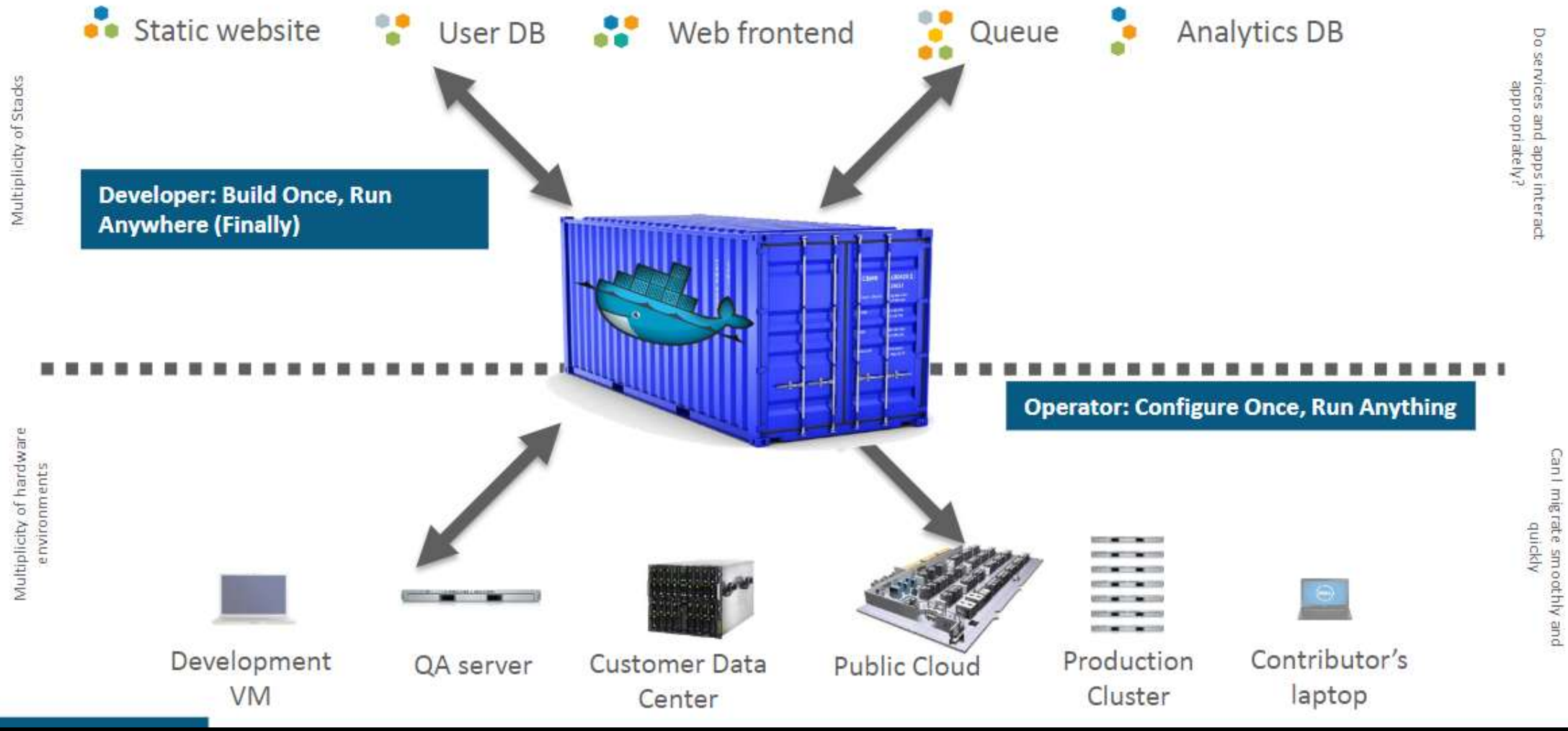
Machine / System






DOCKER TO THE RESCUE

Docker – A shipping Container for node



Introducing Drey and Dave

Drey, the Developer

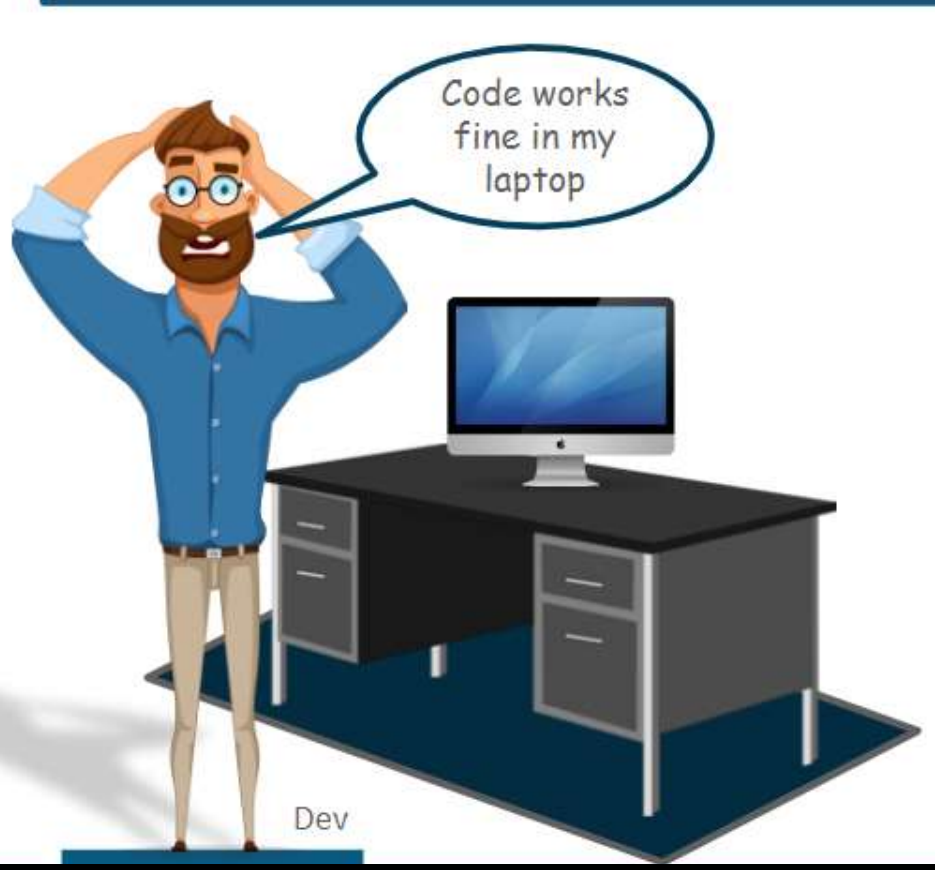
- Worries about what's inside the Container
  - Code
  - Libraries
  - Package Manager
  - Apps
  - Data

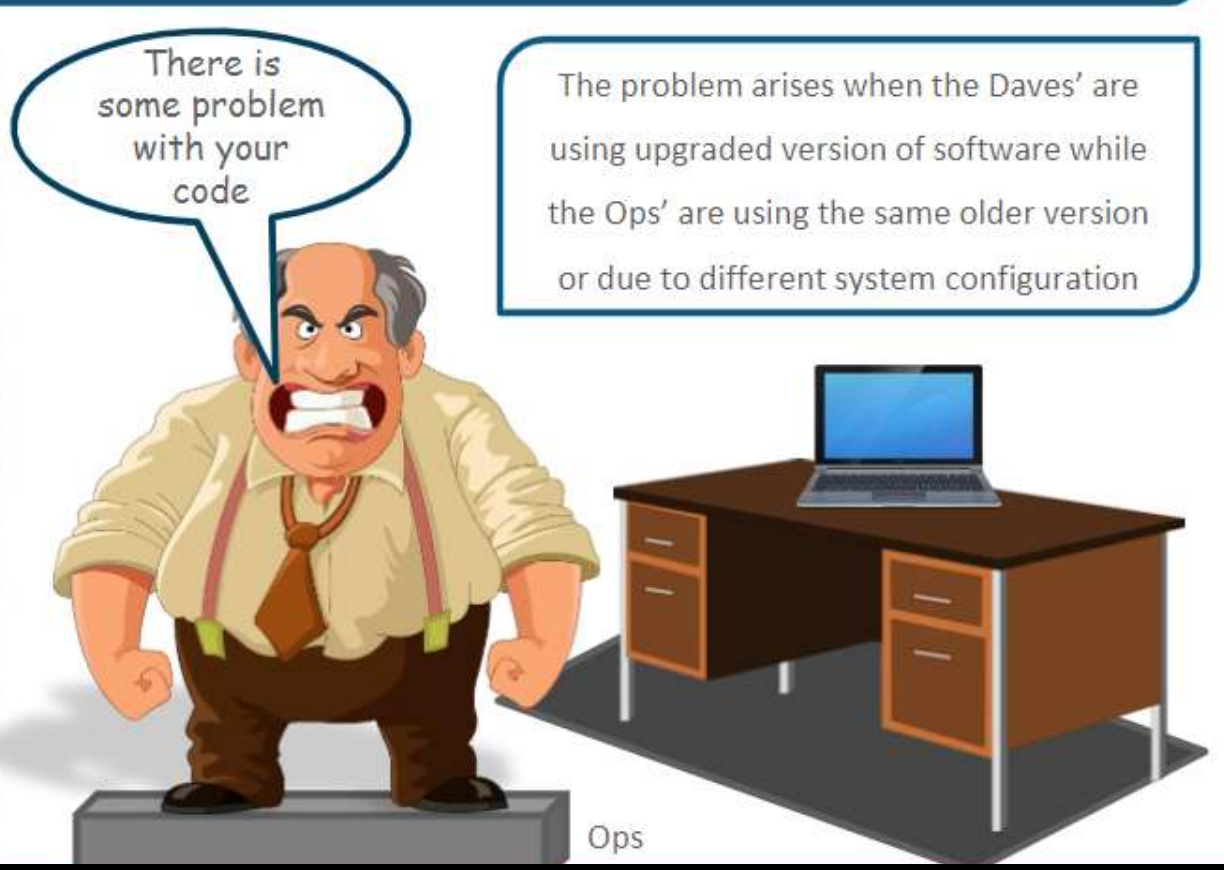
Dave, the Ops Guy

- Worries about what's outside the container
  - Logging
  - Remote Access
  - Monitoring
  - Network Config
- All containers start, stop, copy, attach, migrate the same way

Problems faced by Drey and Dave

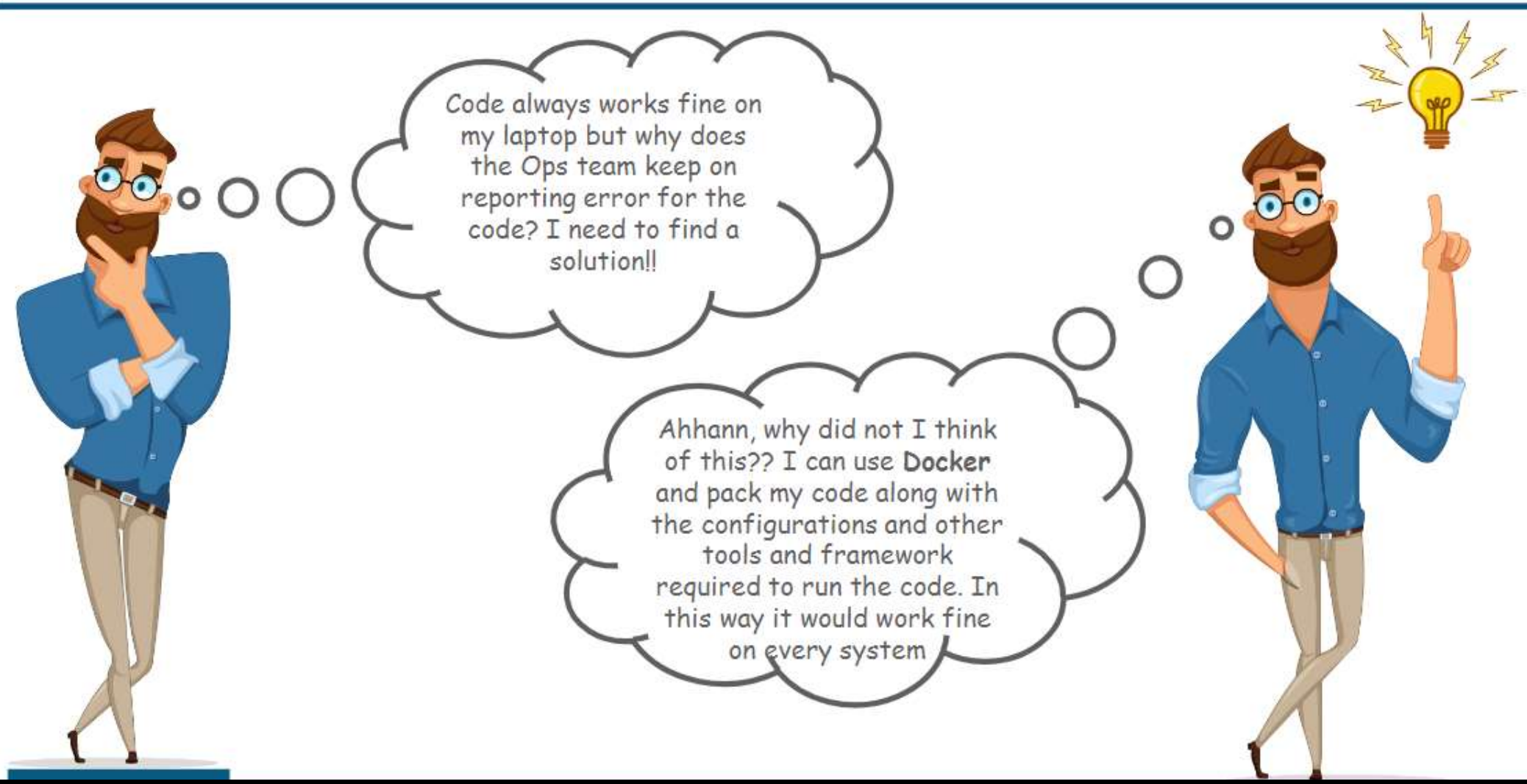
An application works in developer's laptop but not in testing or production. This is due to difference in computing environment between Dev, Test and Prod.





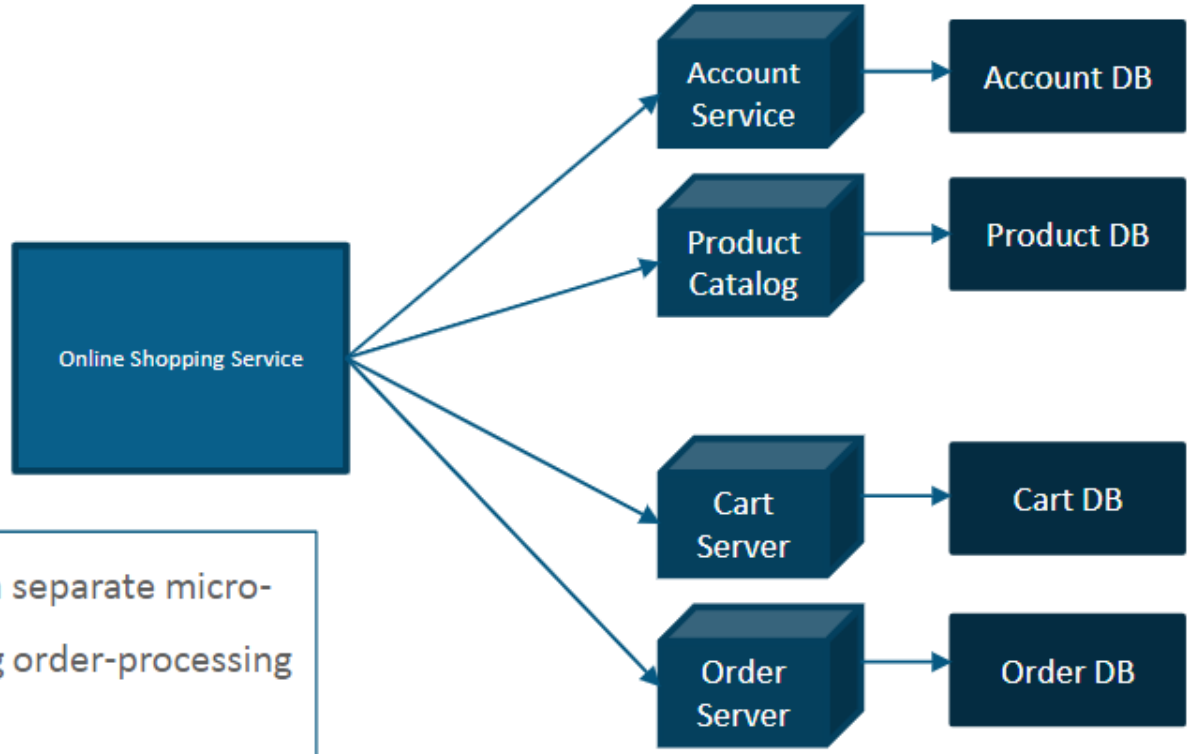
The problem arises when the Daves' are using upgraded version of software while the Ops' are using the same older version or due to different system configuration

Docker to rescue of Drey and Dave



Use Case – Online Portal

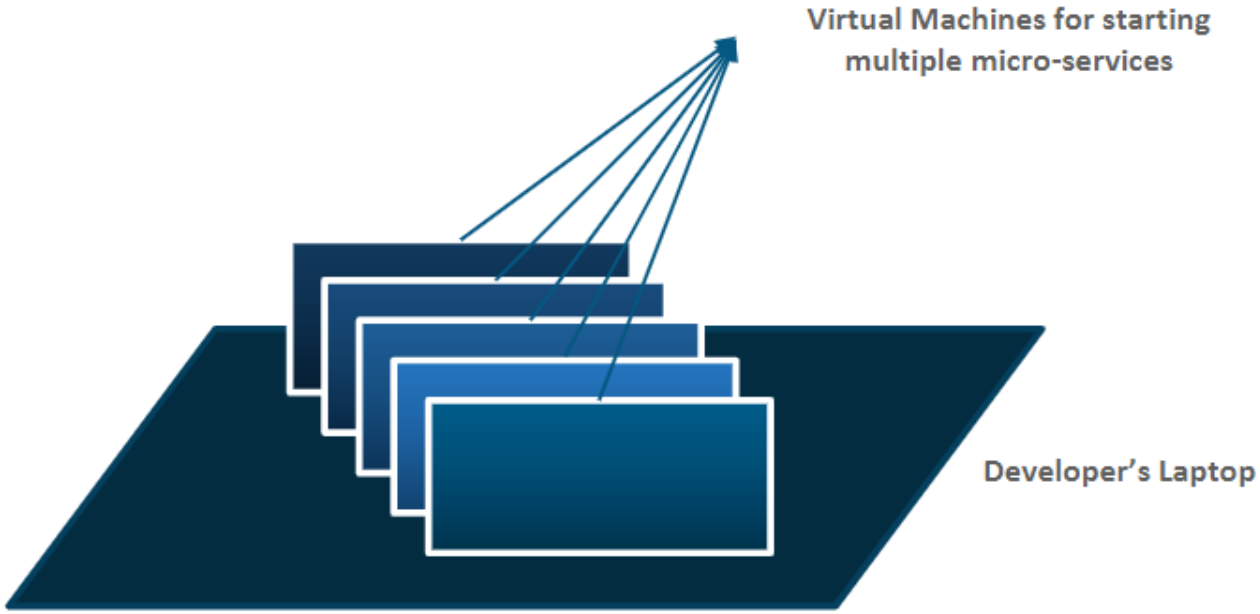
Applications are easier to build and maintain when broken down to smaller, composable pieces which work together (micro-services). Each component is developed separately and the application is the sum of its constituent.



For example, Imagine an online shop with separate micro-services for user-accounts, product-catalog order-processing and shopping carts

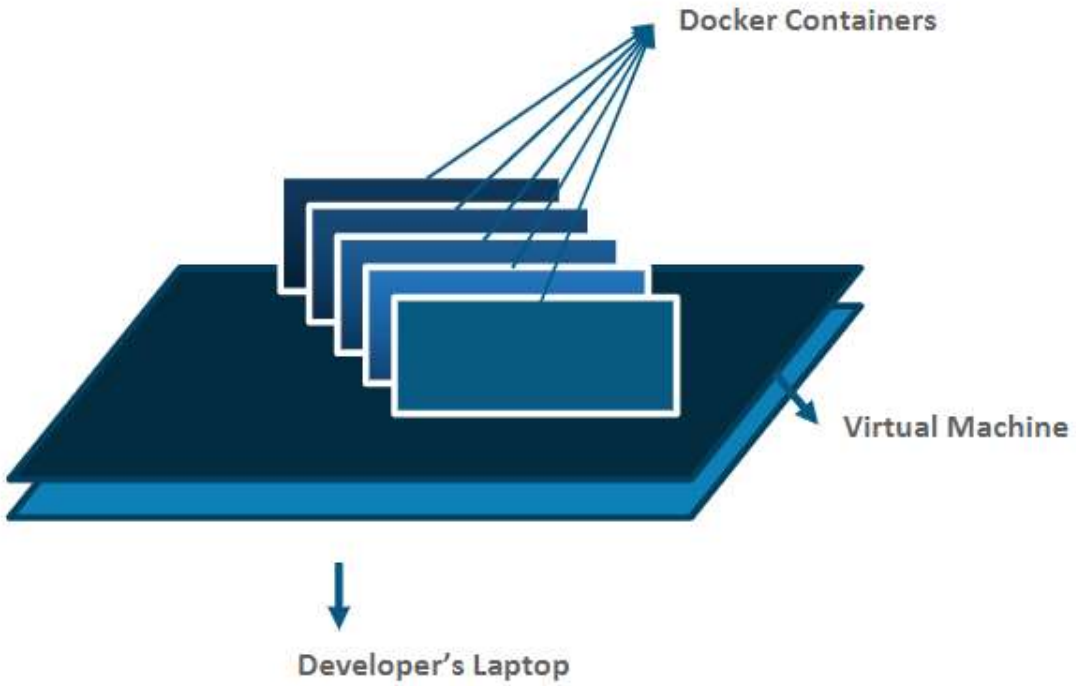
Online Portal – Handling micro-services

Developing an application requires starting several of micro-services in one machine. So if you are starting five of those services you require five VMs on that machine.

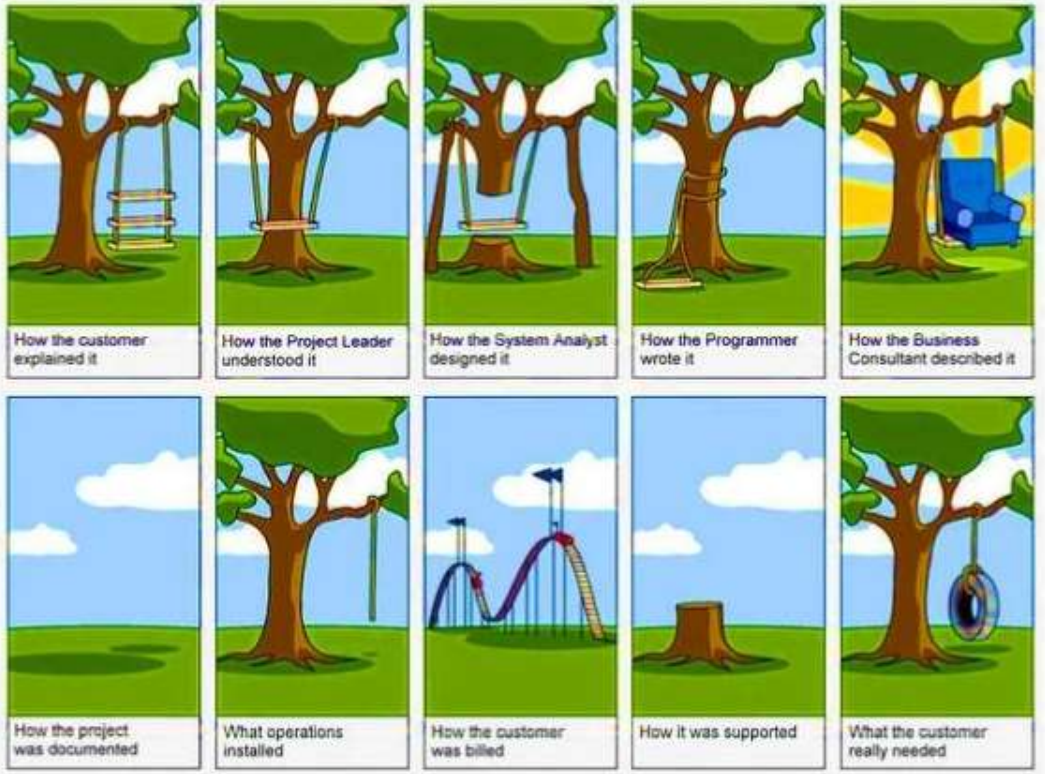


Handling micro-services with Docker

You can run several micro-services in the same VM by running various Docker containers for each micro-service.



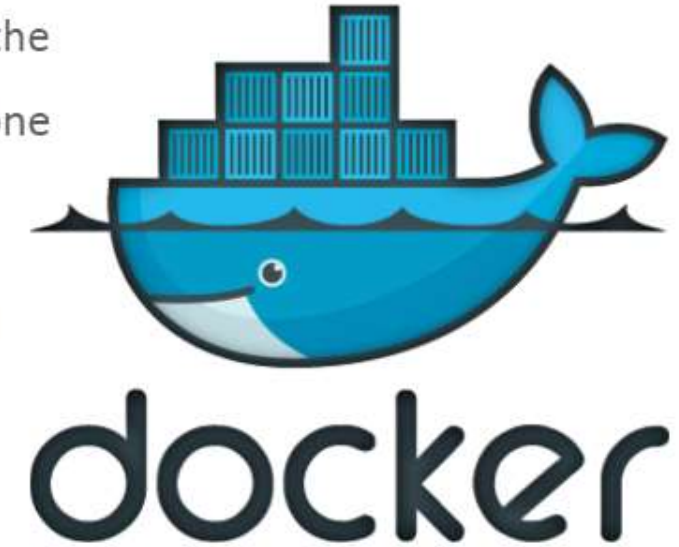
Provides a consistent computing environment throughout the whole SDLC.



Let us see what is Docker!

What is Docker?

- Docker is a tool designed to create, deploy, and run applications with ease by using containers.
- It allows a developer to package up an application with all of the requirements such as libraries and other dependencies and ship it all as one package.
- It ensure that your application works seamlessly in any environment; be it Development, Test or Production.



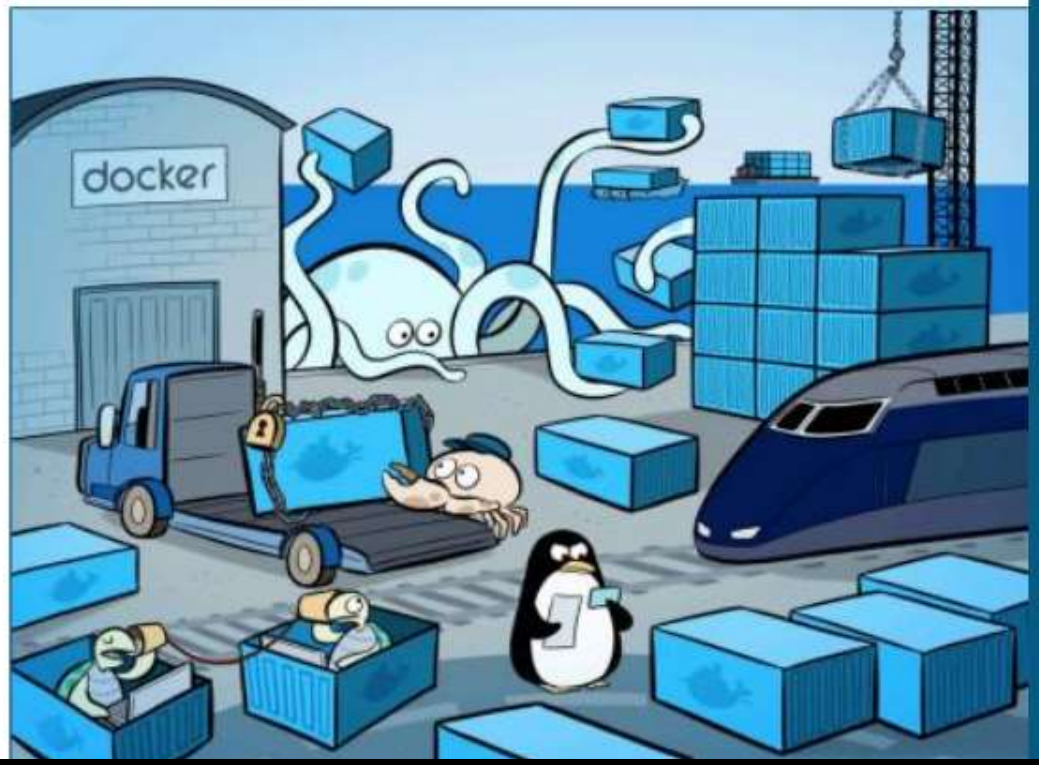
"BUILD, SHIP & RUN ANY SOFTWARE ANY WHERE"

Who uses Docker?

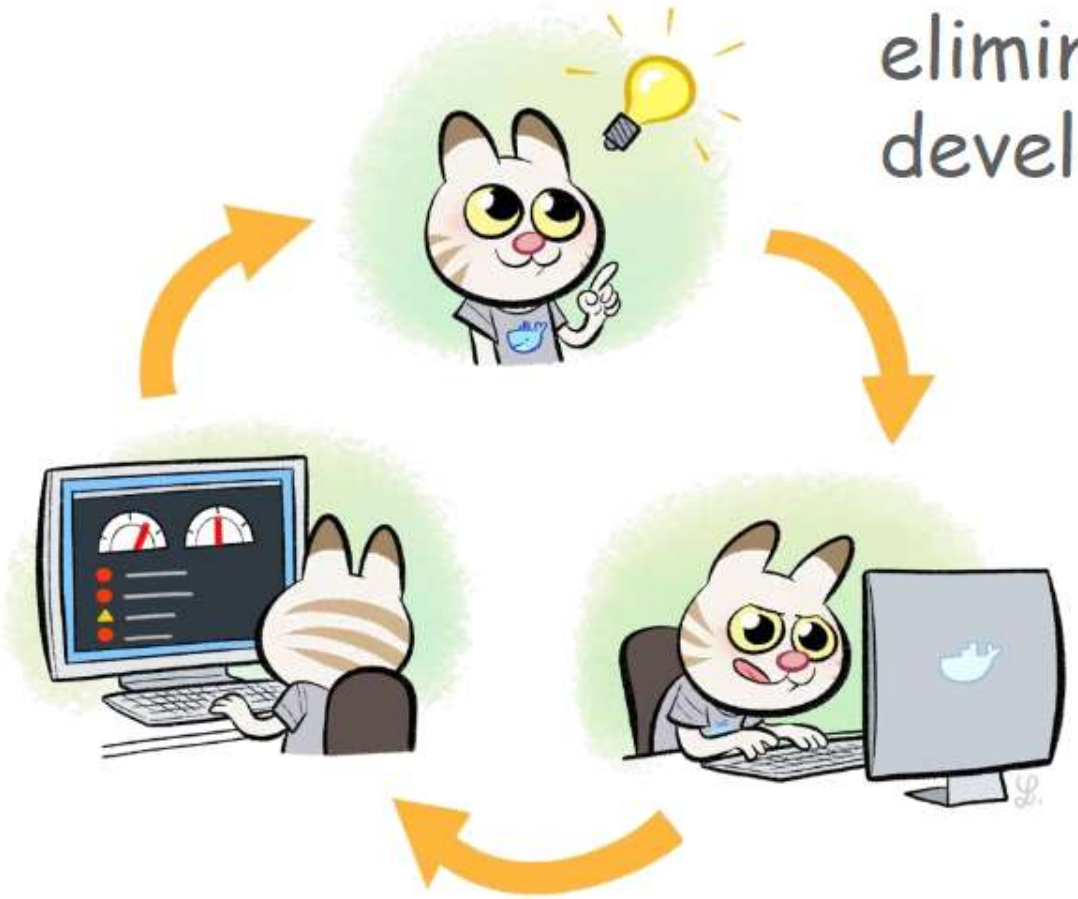
Developer: Docker helps the developer to focus only on building great software by automating the repetitive tasks of setting up and configuring development environment.

Sysadmin: Docker helps the sysadmin to streamline the software delivery, such as develop and deploy bug fixes and new features without any roadblock.

Enterprise: Docker is at the heart of the modern app platform, bridging developer and IT, Linux and Windows. It works in the cloud just as well as on premise; and supports both traditional and microservices architectures.



Why Docker?

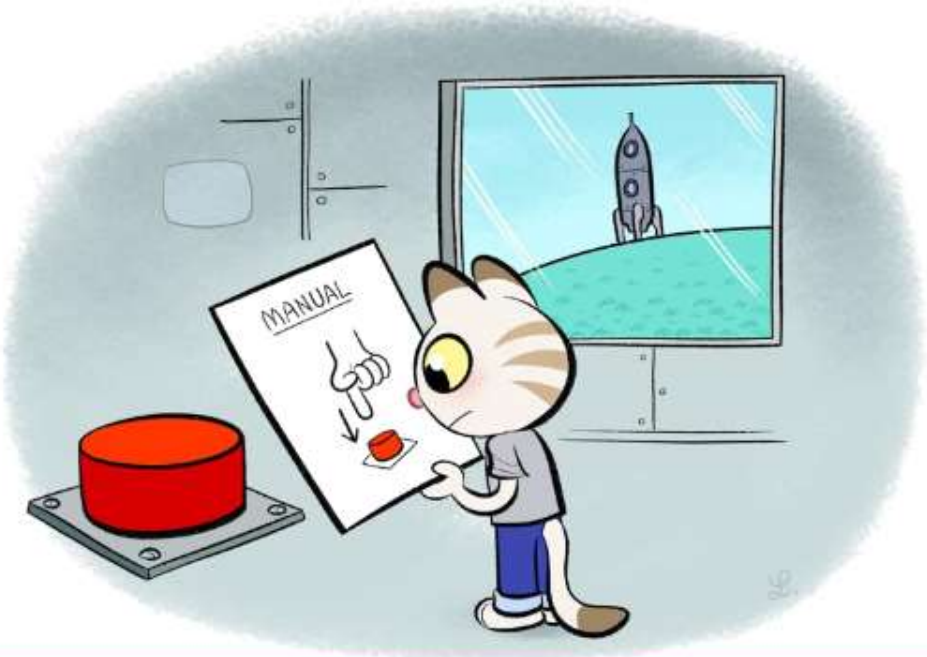


eliminate friction in the development cycle

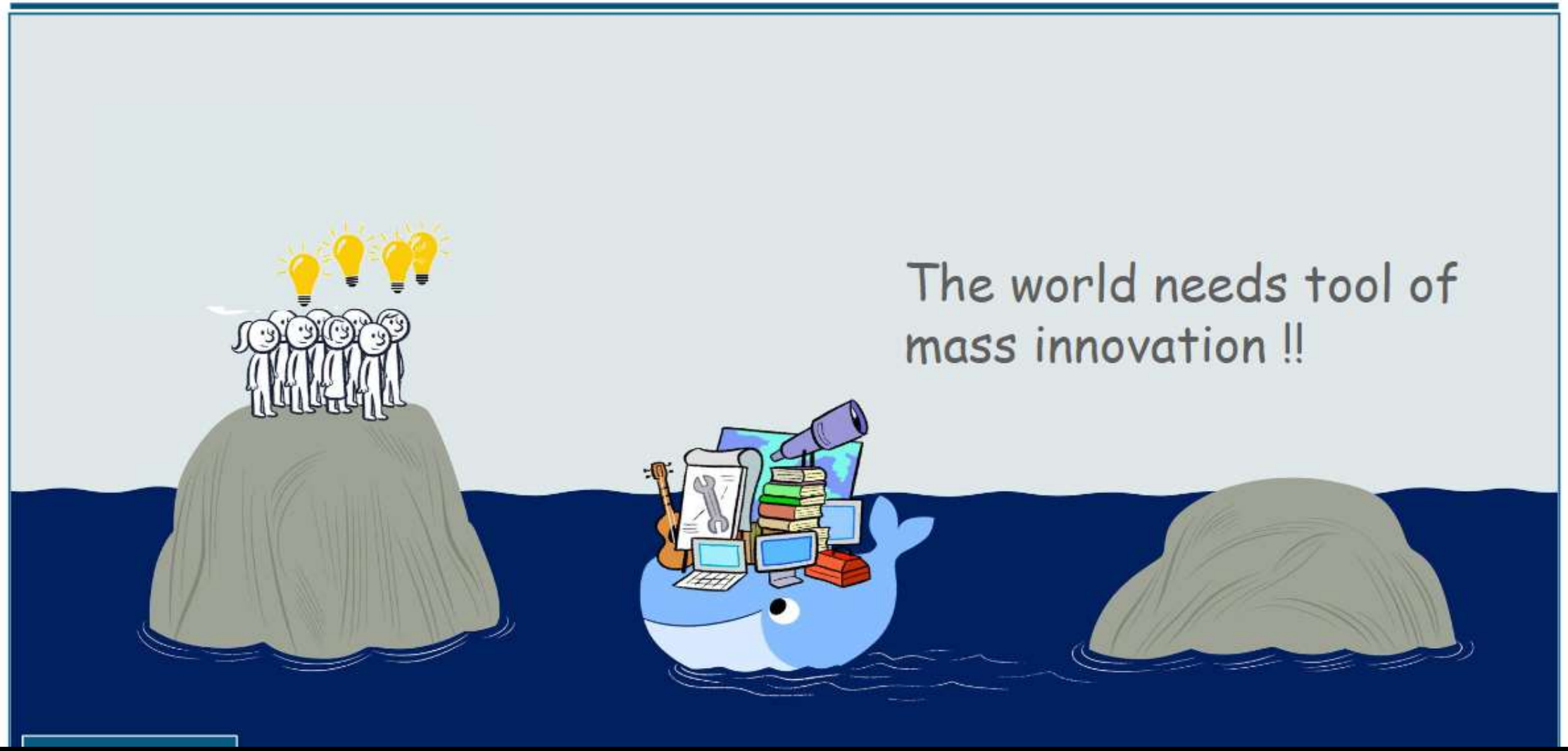
Why Docker?

Easily adapts to your working environment

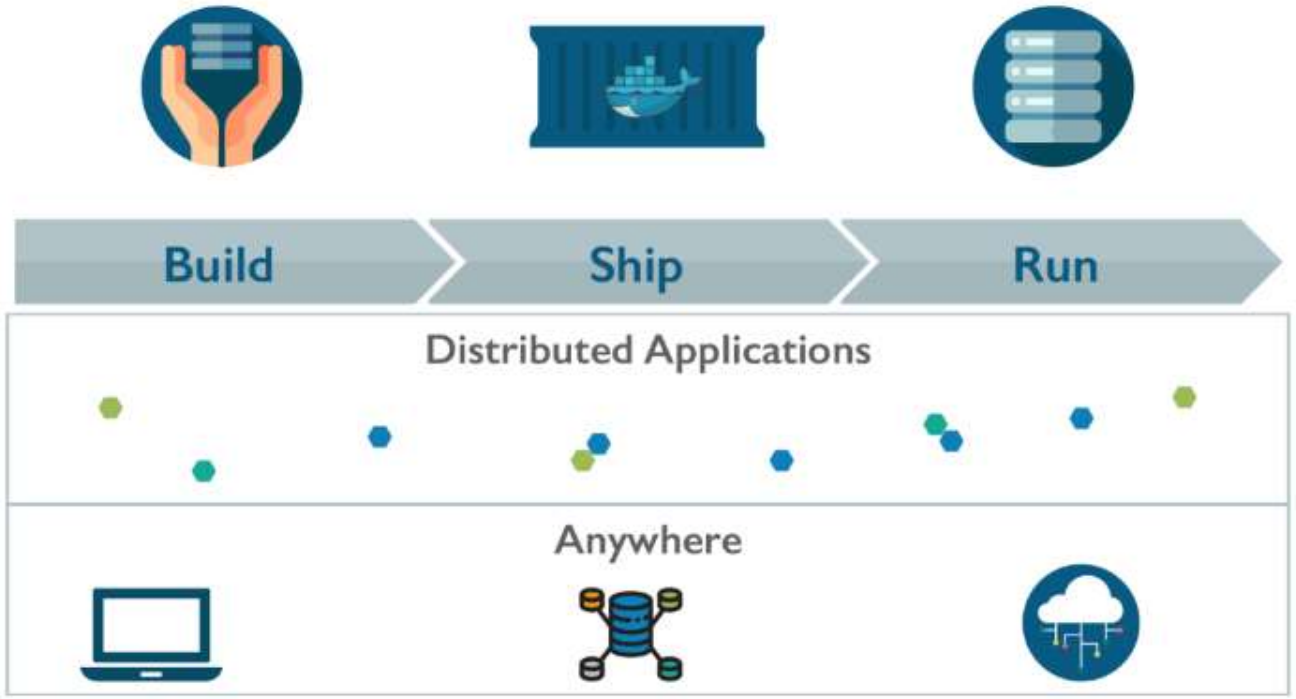
Make the powerful simple



Why Docker?



Docker containers are popular!



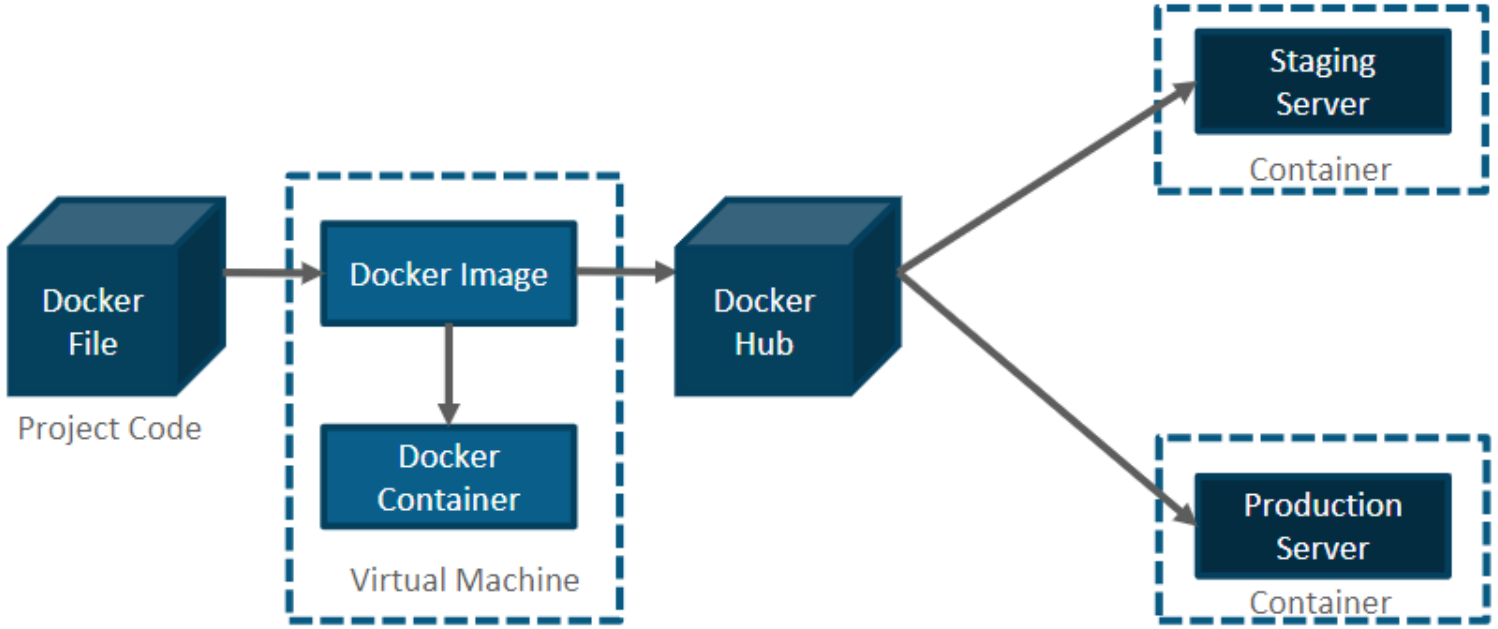
Develop an app using Docker containers with any language and any toolchain.

Ship the "Dockerized" app and dependencies anywhere - to QA, teammates, or the cloud - without breaking anything.

Scale to 1000s of nodes, move between data centers and clouds, update with zero downtime and more.

Docker in a Nutshell

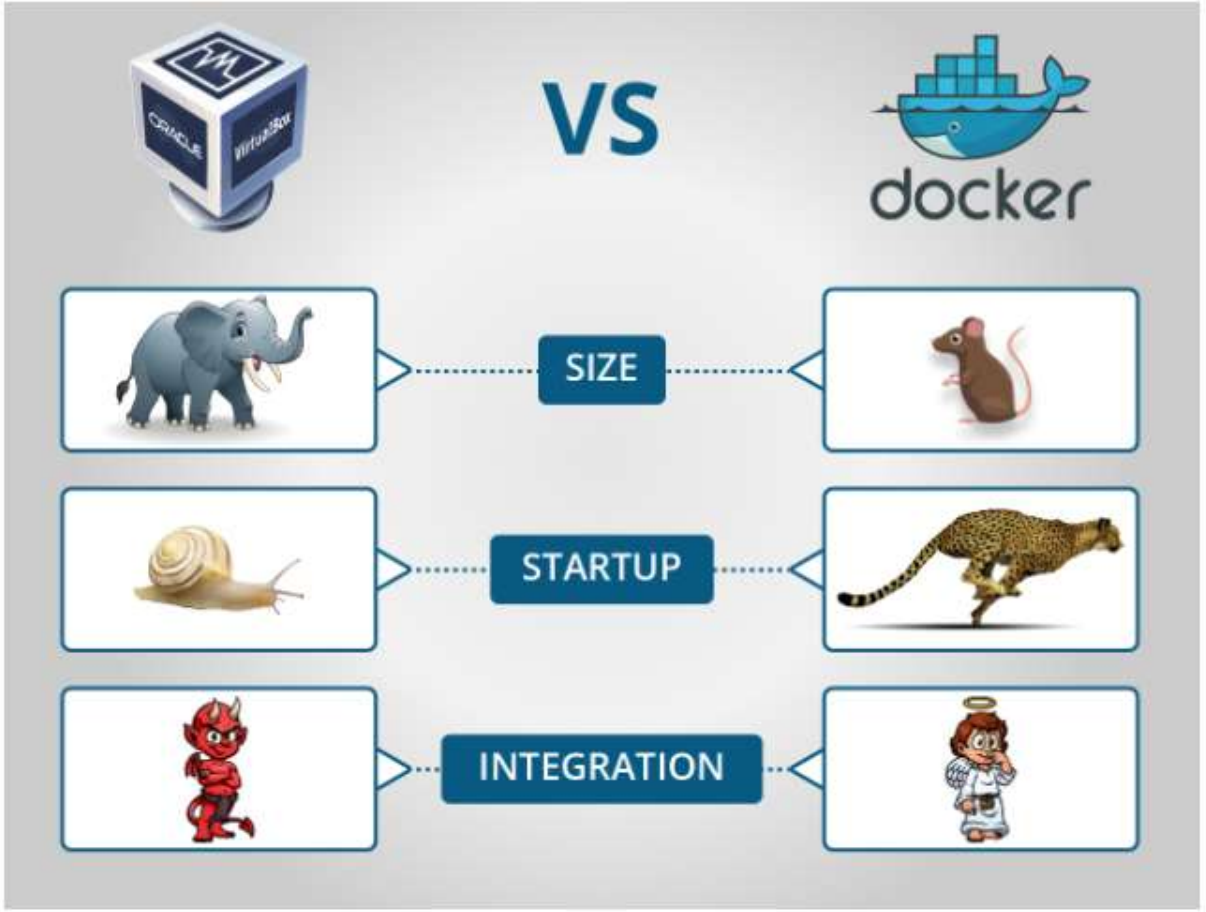
- Docker file builds a Docker image and that image contains all the project's code
- You can run that image to create as many docker containers as you want
- The created Images can be uploaded on Docker hub from where the image can be pulled and built in a container



Programming the world with Docker

BENEFITS OF DOCKER OVER VM

VM vs. Docker

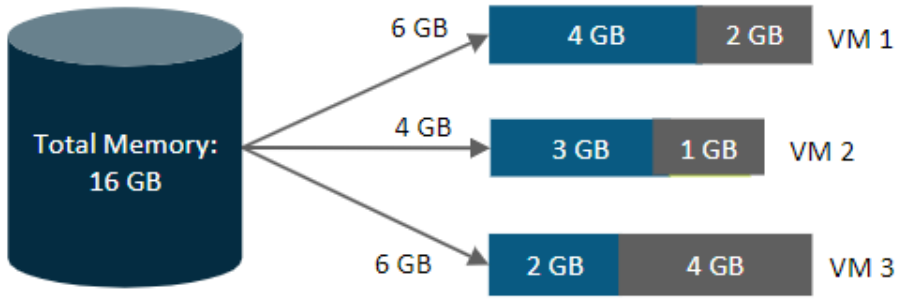


VM vs Docker

1. Size
2. Startup
3. Integration

Resource / Memory Management

In case of Virtual Machines



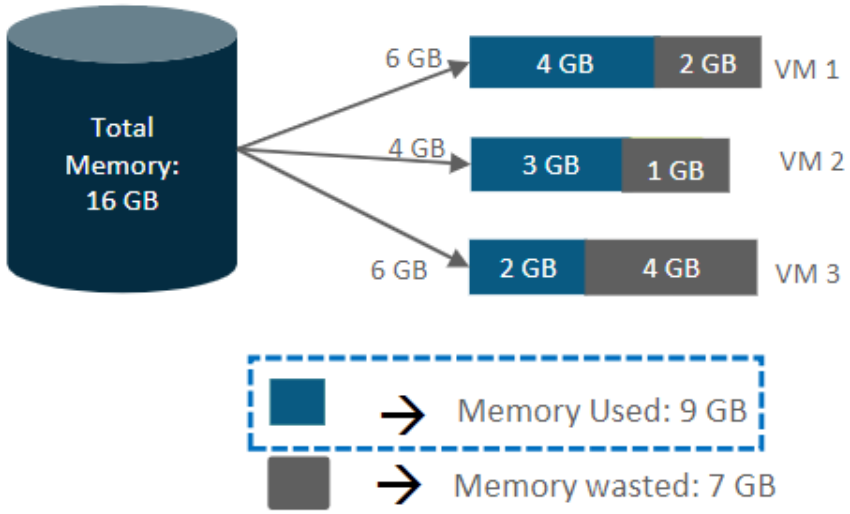
7 GB of memory is blocked and cannot be allotted to a new VM

VM vs Docker

1. Size
2. Startup
3. Integration

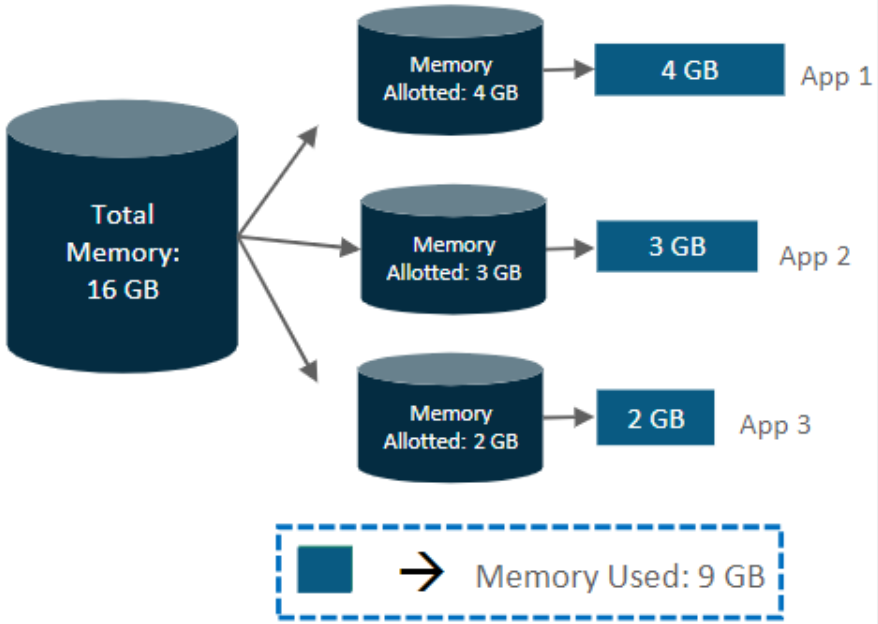
Resource / Memory Management

In case of Virtual Machines



7 GB of memory is blocked and cannot be allotted to a new VM

In case of Docker



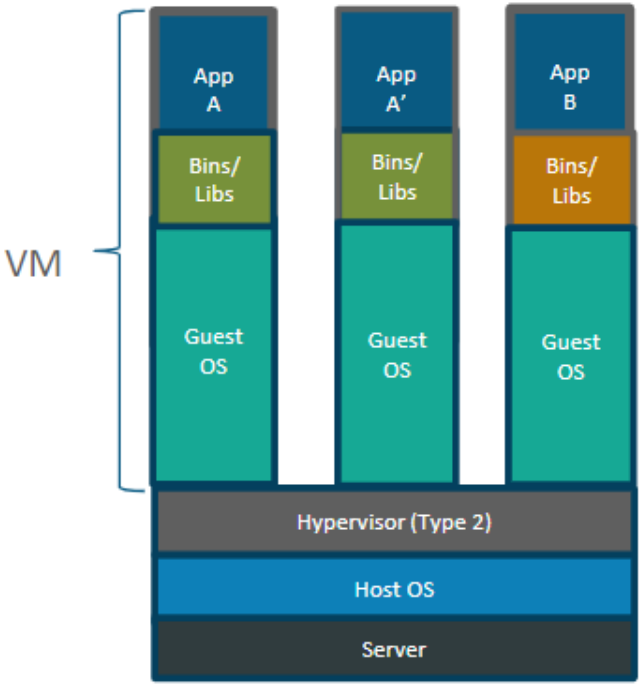
Only 9 GB memory utilized;
7 GB can be allotted to a new Container

VM vs Docker

1. Size
2. Startup
3. Integration

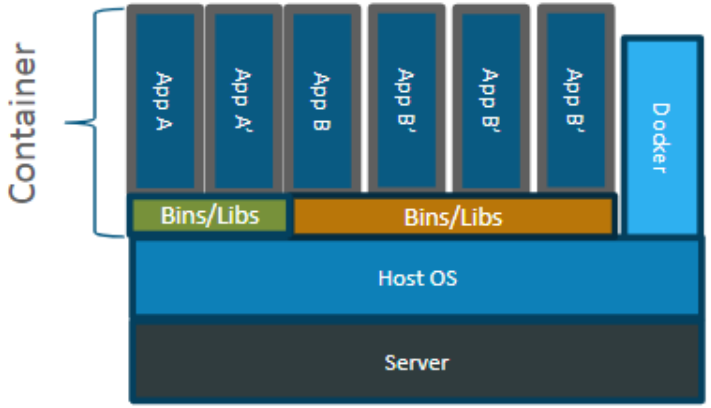
Building & Deployment

In case of Virtual Machines



New Builds → **Multiple OS** → Separate Libraries → Heavy → **More Time**

In case of Docker

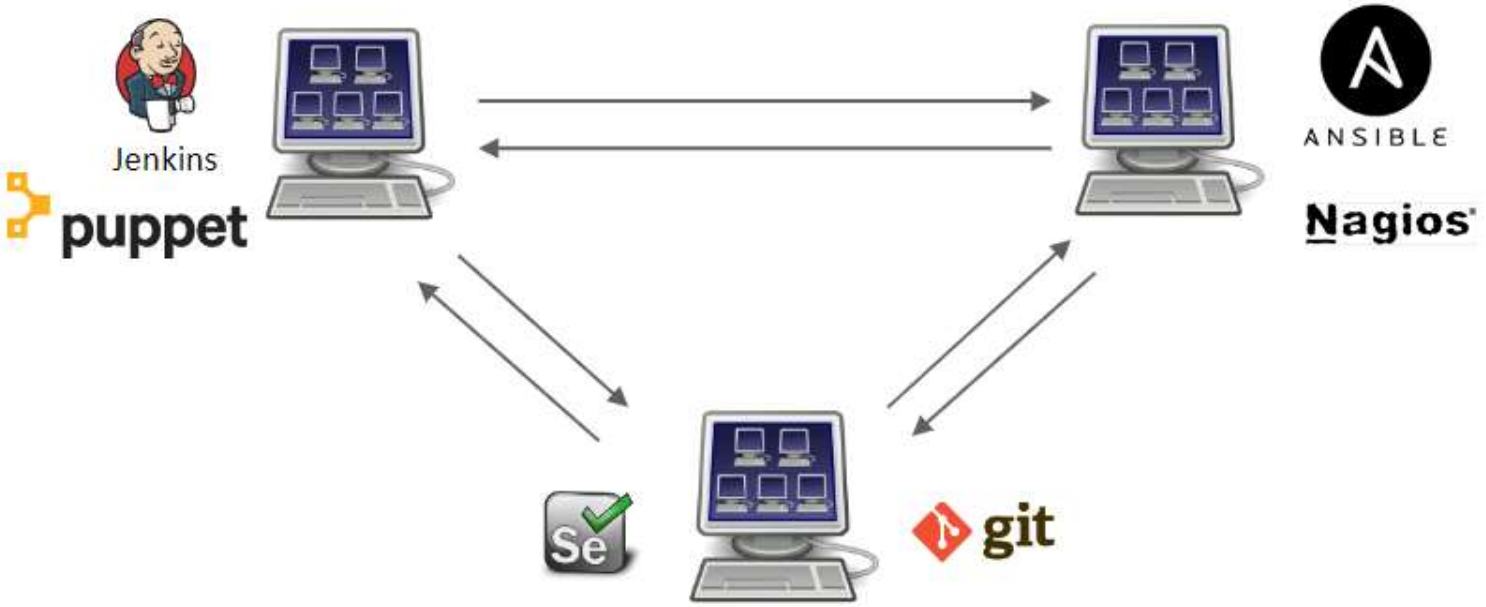


New Builds → **Same OS** → Separate Libraries → Lightweight → **Less Time**

Integration in VMs

Integration in virtual machines is possible, but:

- Costly due to infrastructure requirements
- Not easily scalable



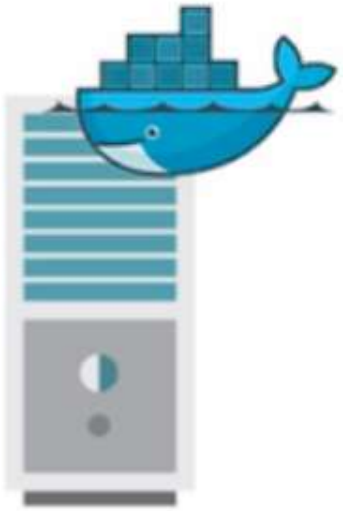
DOCKER TERMINOLOGY

Docker Components

- Docker Daemon
- Docker Client
- Docker Registry
- Docker Images
- Docker Containers

Docker Daemon

- The Docker daemon runs on a host machine.
- The user uses the Docker client to interact with the daemon.



Why Use Docker Daemon?

- Responsible for creating, running, and monitoring containers
- Building and storing images

Docker Daemon

- On a typical installation the Docker daemon is started by a system utility, not manually by a user. This makes it easier to automatically start Docker when the machine reboots.
- If you don’t want to use a system utility to manage the Docker daemon, or just want to test things out, you can manually run it using the “dockerd” command. You may need to use “sudo” depending on your operating system configuration.
- When you start Docker this way, it runs in the foreground and sends its logs directly to your terminal.
- To stop Docker when you have started it manually, issue a “Ctrl+C” in your terminal.

Configure the Docker Daemon

- There are two ways to configure the Docker daemon:
  - Use a JSON configuration file. This is the preferred option, since it keeps all configurations in a single place.
  - Use flags when starting “dockerd”
- You can use both of these options together as long as you don’t specify the same option both as a flag and in the JSON file. If that happens, the Docker daemon won’t start and prints an error message.
- To configure the Docker daemon using a JSON file, create a file at /etc/docker/daemon.json on Linux systems. Here’s what the configuration file looks like:

```
{
  "debug": true,
  "tls": true,
  "tlscert": "/var/docker/server.pem",
  "tlskey": "/var/docker/serverkey.pem",
  "hosts": ["tcp://192.168.59.3:2376"]
}
```

With this configuration the Docker daemon runs in debug mode, uses TLS, and listens for traffic routed to `192.168.59.3` on port `2376`. You can learn what configuration options are available in the dockerd reference docs

You can also start the Docker daemon manually and configure it using flags. This can be useful for troubleshooting problems.

Configure the Docker Daemon

Here's an example of how to manually start the Docker daemon, using the same configurations as above:

```
dockerd --debug \
  --tls=true \
  --tlscert=/var/docker/server.pem \
  --tlskey=/var/docker/serverkey.pem \
  --host tcp://192.168.59.3:2376
```

Docker Client

- The Docker client is the primary UI to Docker.
- It accepts commands and configuration flags and communicates with a Docker daemon via HTTP.

Why Use Docker Client?

- Since all communication has to be done over HTTP, it is easy to connect to remote Docker
- The API used for communication with daemon allows developers to write programs that interface directly with the daemon, without using Docker

Docker Registry

- Docker Registry is a storage component for Docker Images
- We can store the Images in either Public/Private repositories
- Docker Hub is Docker's very own cloud repository

Why Use Docker Registries?

- Control where your images are being stored
- Integrate image storage with your in-house development workflow

Private or Public Registry?

When choosing between these, some points to consider are:

- Performance, depending mainly on roll-out frequency and cluster size
- Security issues such as access control and digitally signing Docker image

With a private registry,

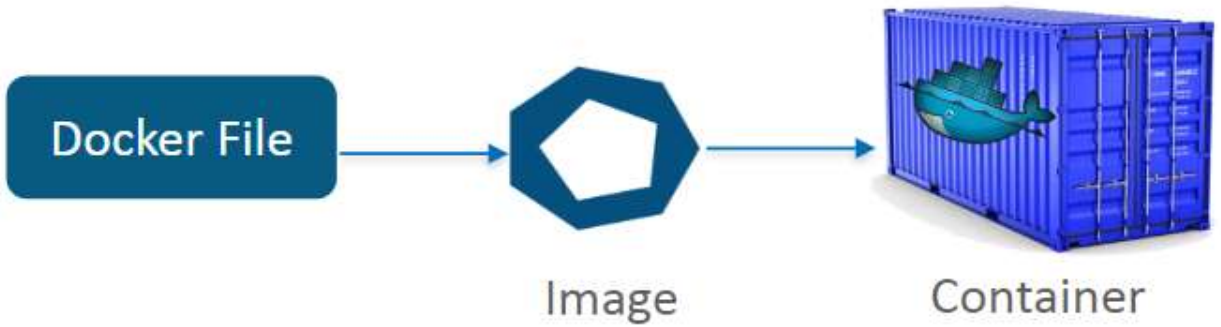
- You are in full control.
- No external dependencies in the CD pipeline, so your build is faster (or appear to be faster)
- Have to manage storage yourself – and this can increase drastically as the adoption of DevOps and number of build increase.

Before doing any decisions,

- Evaluate how many builds/images are being pushed and the average increase in size
- Also factor in growth rates for your applications/builds
- These will give you metrics' for determining network bandwidth and storage requirements

What is an Image?

- An image is a text file with a set of pre-written commands, usually called as a Docker file
- Docker Images are made up of multiple layers which are read-only filesystem
- A layer is created for each instruction in a Docker file and placed on top of the previous layer
- When an image is turned into a container the Docker engine takes the image and adds a read-write filesystem on top (as well as initializing various settings such as the IP address, name, ID, and resource limits)



Docker Images & Containers

Docker Images

- Read only template used to create containers
- Built by Docker users
- Stored in DockerHub or your local registry

Docker Images & Containers

run

Docker Images

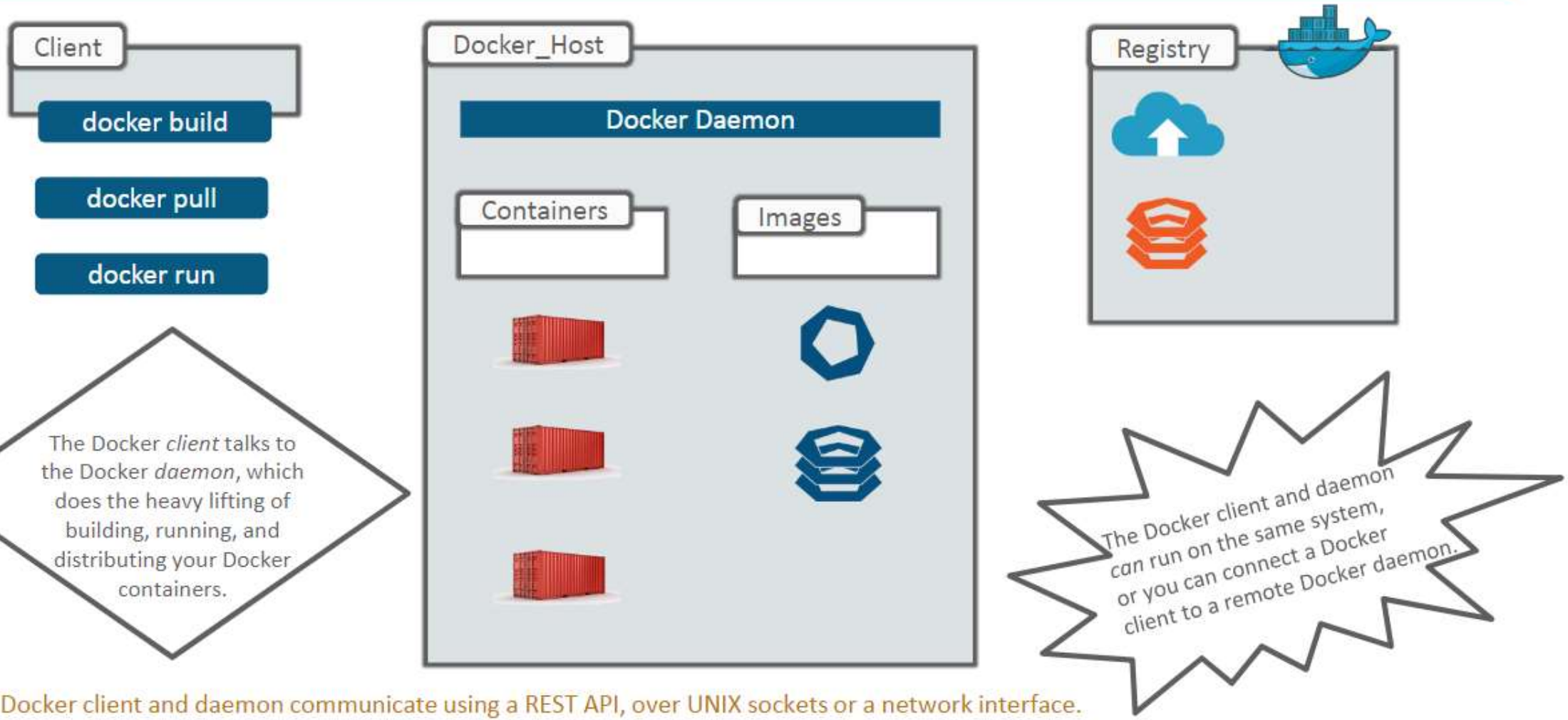
- Read only template used to create containers
- Built by Docker users
- Stored in DockerHub or your local registry

Docker Containers

- Isolated application platform
- Contains everything needed to run the application
- Built from one or more images

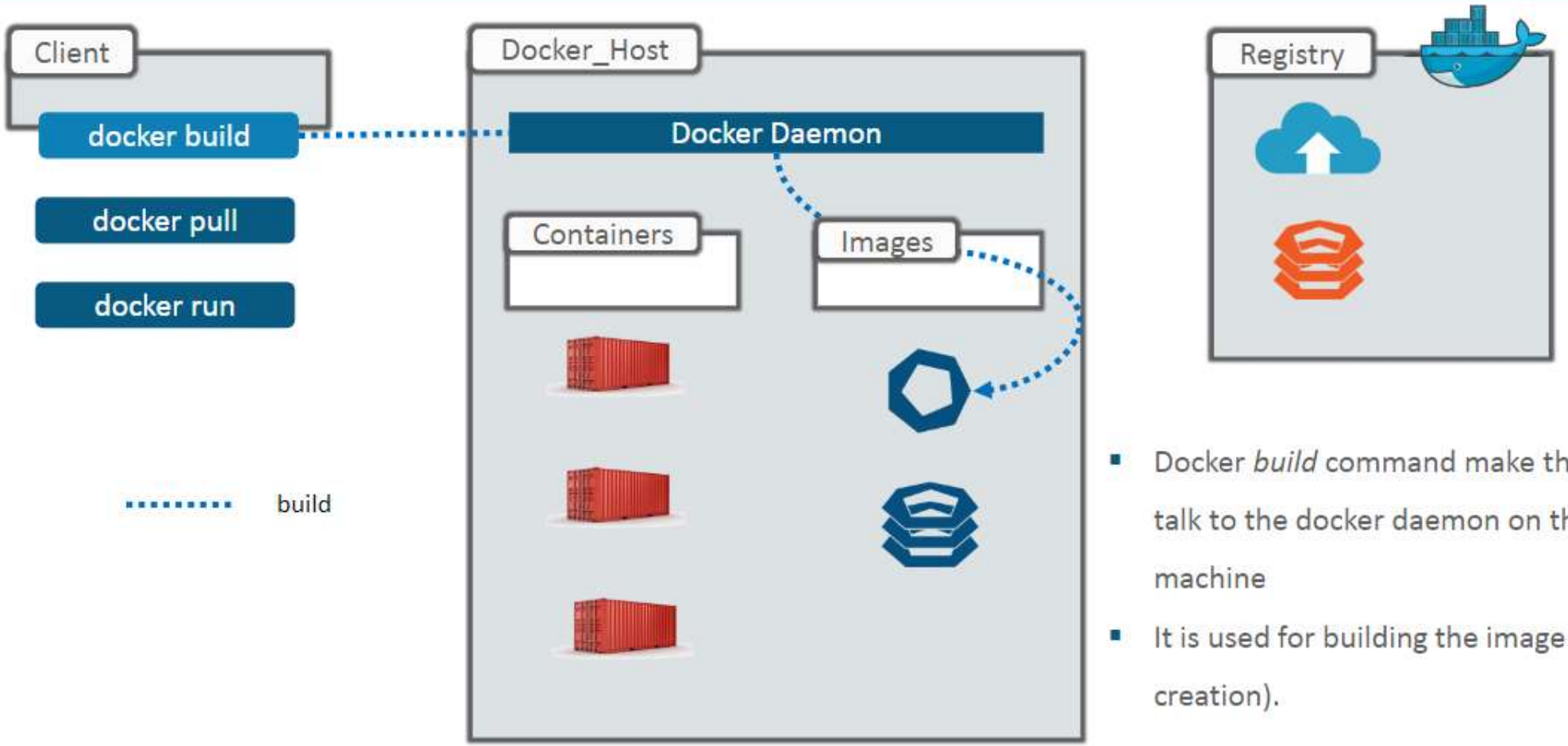
DOCKER ARCHITECTURE

Docker Architecture



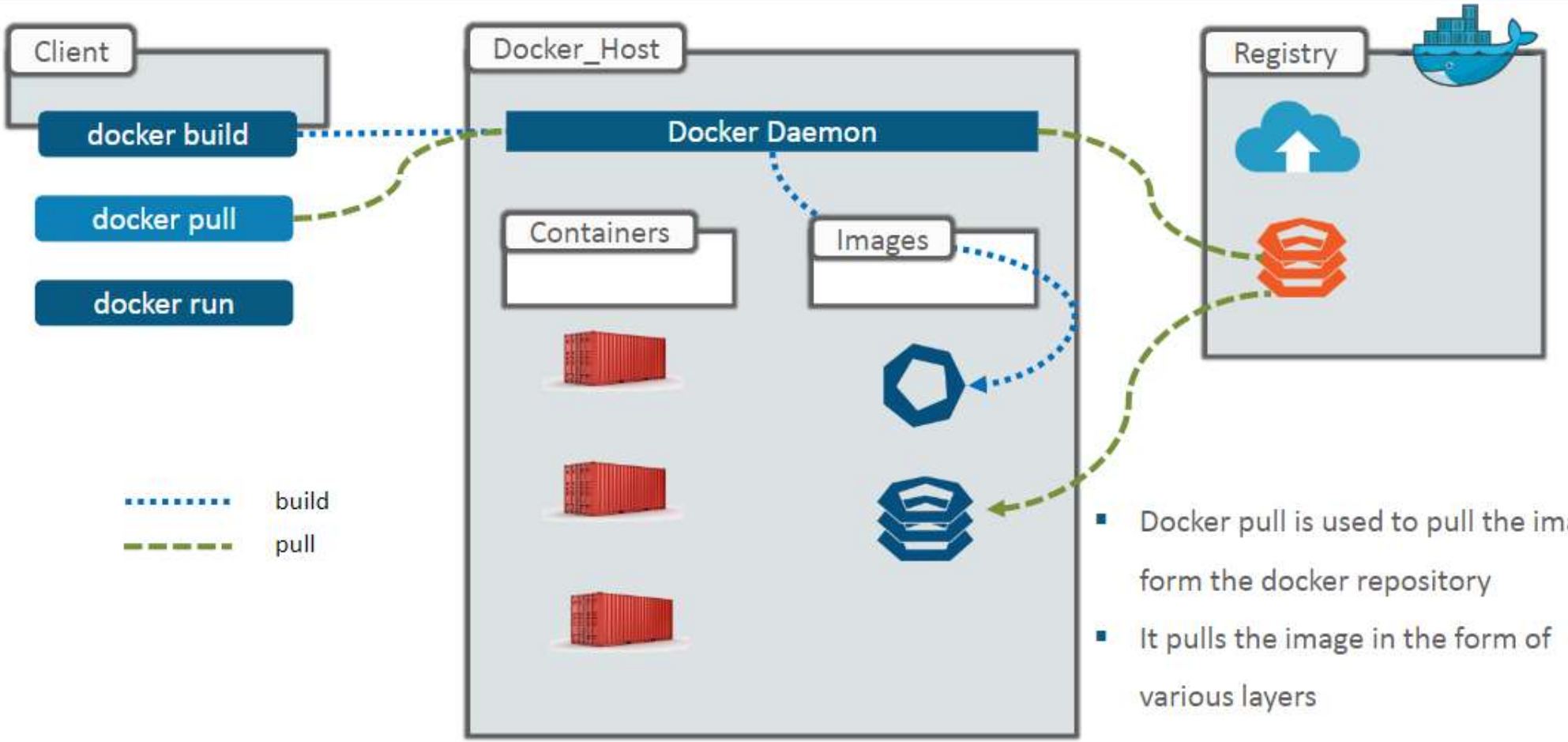
The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Docker Architecture In Action



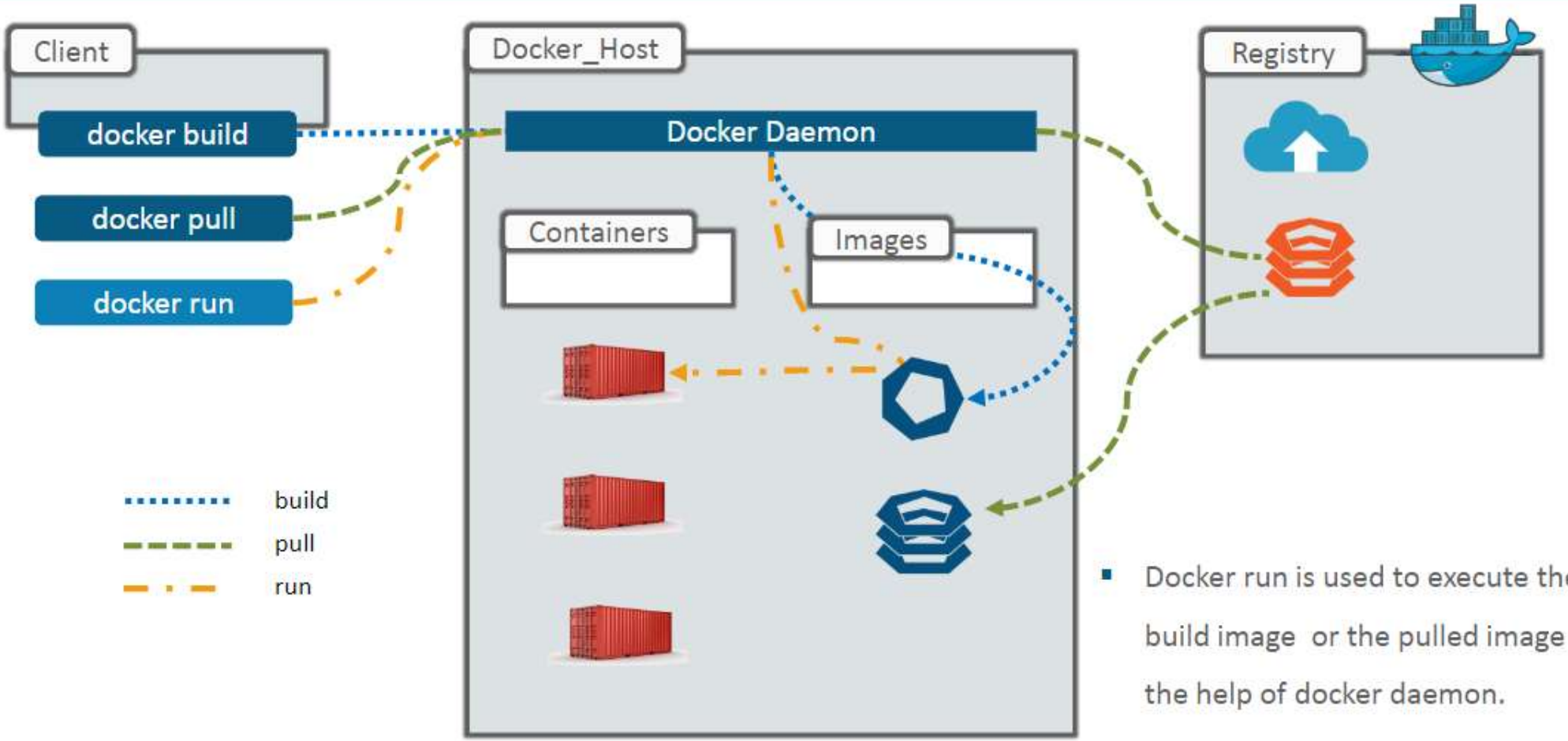
- Docker *build* command make the client talk to the docker daemon on the host machine
- It is used for building the image (image creation).

Docker Architecture In Action



- Docker pull is used to pull the image form the docker repository
- It pulls the image in the form of various layers

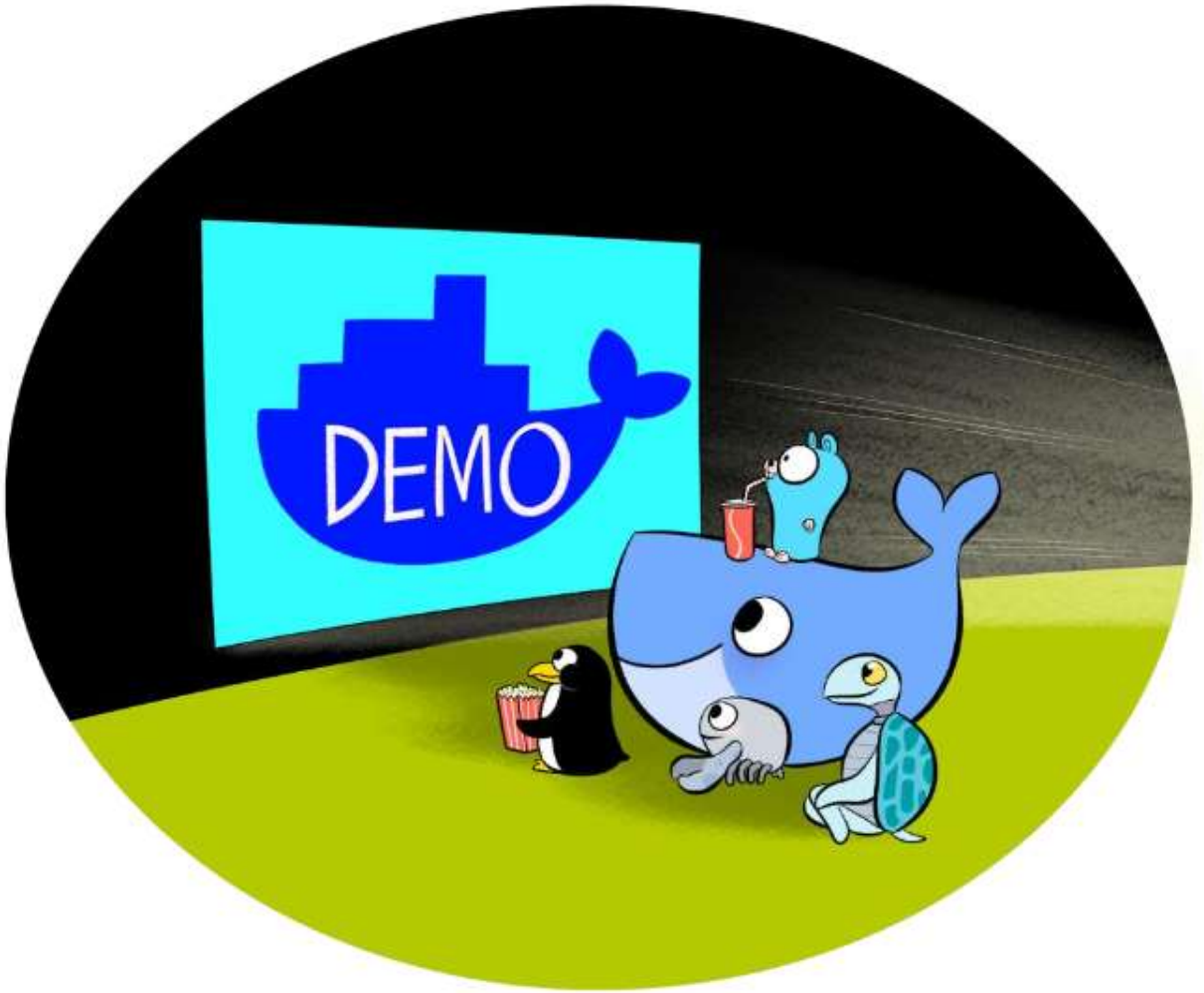
Docker Architecture In Action



- Docker run is used to execute the build image or the pulled image with the help of docker daemon.

Docker Hands - On

- Version Check
- Searching hello-world
- Pull hello-world
- Execute hello-world



Version Check

```
root@test01:~# docker version
Client:
 Version:      1.12.3
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   6b644ec
 Built:        Wed Oct 26 21:44:32 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.12.3
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   6b644ec
 Built:        Wed Oct 26 21:44:32 2016
 OS/Arch:      linux/amd64
root@test01:~#
```

Few Basic Commands

- $ docker help - Displays all the useful commands for Docker and other general help commands
- $ docker images - Displays a list of existing images in Docker system. It also displays the following details:
  - REPOSITORY: Name of the repository
  - TAG: Every image has an attached tag.
  - IMAGE ID: Each image is assigned a unique ID
  - CREATED: The date when the image was created
  - SIZE: The size of the image

Few Basic Commands contd…

- $ docker ps - Displays the list of active containers. It also displays the following details:
  - CONTAINER ID: Each container is assigned a unique ID
  - IMAGE: Every image has an attached tag.
  - COMMAND: Each image is assigned a unique ID
  - CREATED : The date when the image was created
  - STATUS: This shows whether the container is active or not
  - PORTS: The number of exposed ports (needed for networking)
  - NAMES: Name of container which is automatically assigned by Docker. It contains first name, "_" and last name
- $ docker ps -a - Displays the list of all the container processes which are running or have run in the past

Create Our First Image: “Hello-World”

The following actions will be performed to pull the image:

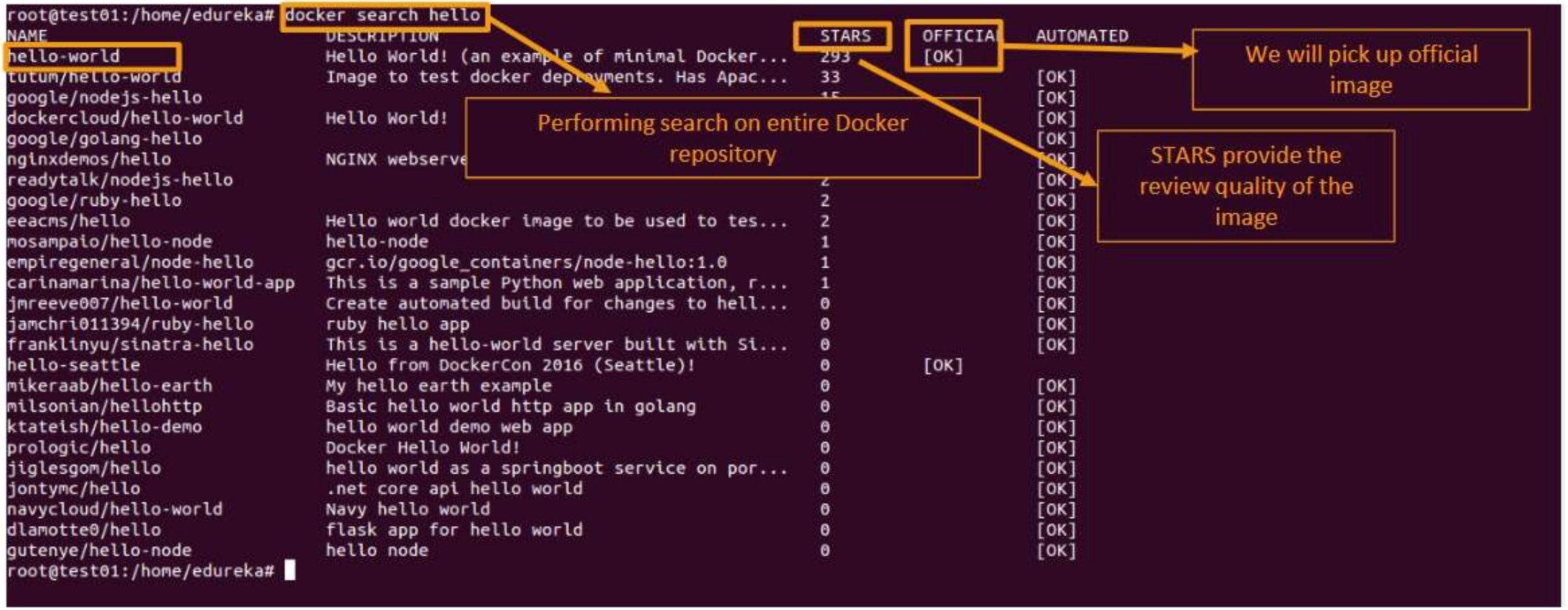
Step 1: Search for images which start with “hello-world” word from Docker Repository

Step 2: Pull the selected image from Docker Hub

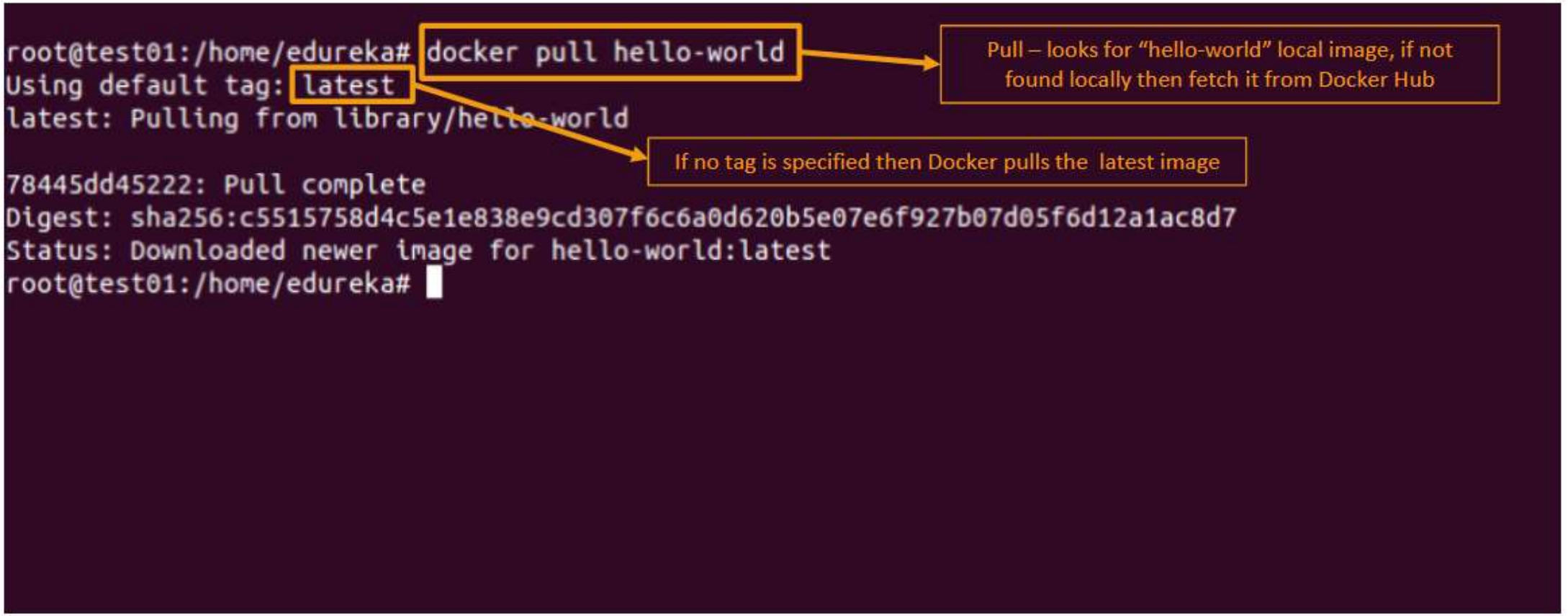
Step 3: Search for the copy of the image in our local repository

Step 4: Execute the “Hello-world” pulled from Docker Repository

Step 1: Search Hello-World



Step 2: Pull Hello-World



Step 3: Searching Local Repository

```
root@test01:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              48b5124b2768        3 months ago        1.84 kB
ubuntu              latest              4ca3a192ff2a        4 months ago        128.2 MB
root@test01:~#
```

Step 4: Execute Hello-World

INTRODUCTION TO CONTAINER

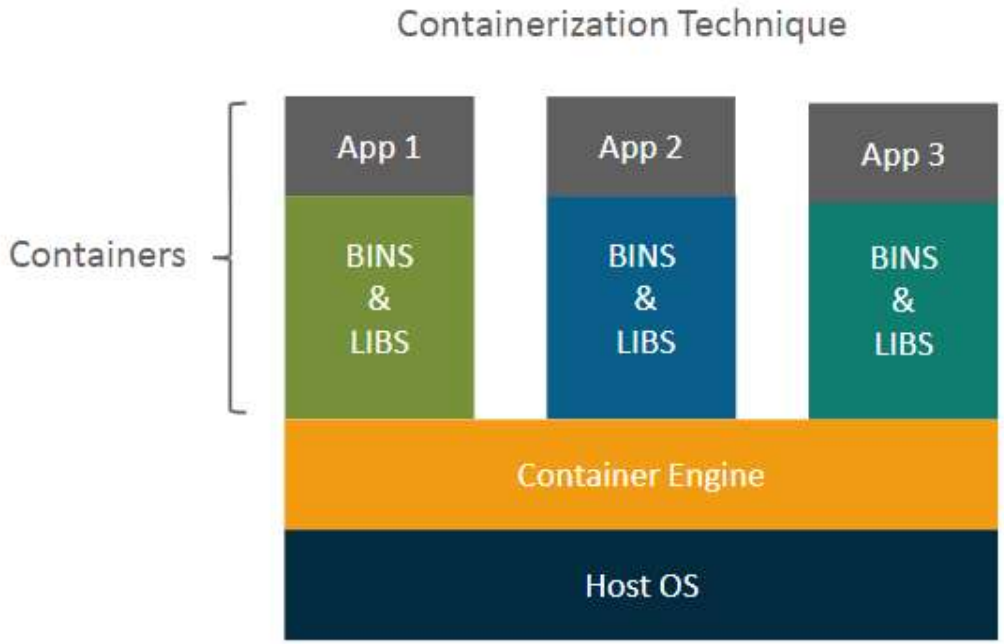
What is Container?

- Container is basically a VM without OS
- In other words, it is packaged with everything required to make the software run, except the OS, hence making the application portable
- All you need is libraries and settings to make the software work on any system
- It makes the container efficient, self-contained, lightweight system and guarantees that the packaged software will always run, regardless of where it is deployed

Why Containers Matter ?

	Physical Containers	Docker Container
Content Agnostic	• The same container can hold almost any type of cargo	• Can encapsulate any payload and its dependencies
Hardware Agnostic	• Standard shape and interface allow same container to move from ship to train to semi-truck to warehouse to crane without being modified or opened	• Using operating system primitives (e.g. LXC) can run consistently on virtually any hardware—VMs, bare metal, openstack, public IAAS, etc.—without modification
Content Isolation and Interaction	• No worry about anvils crushing bananas. Containers can be stacked and shipped together	• Resource, network, and content isolation. Avoids dependency
Automation	• Standard interfaces make it easy to automate loading, unloading, moving, etc.	• Standard operations to run, start, stop, commit, search, etc. Perfect for devops: CI, CD, autoscaling, hybrid clouds
Highly efficient	• No opening or modification, quick to move between waypoints	• Lightweight, virtually no start-up penalty, quick to move and manipulate
Separation of duties	• Shipper worries about inside of box, carrier worries about outside of box	• Developer worries about code. Ops worries about infrastructure.

Containerization

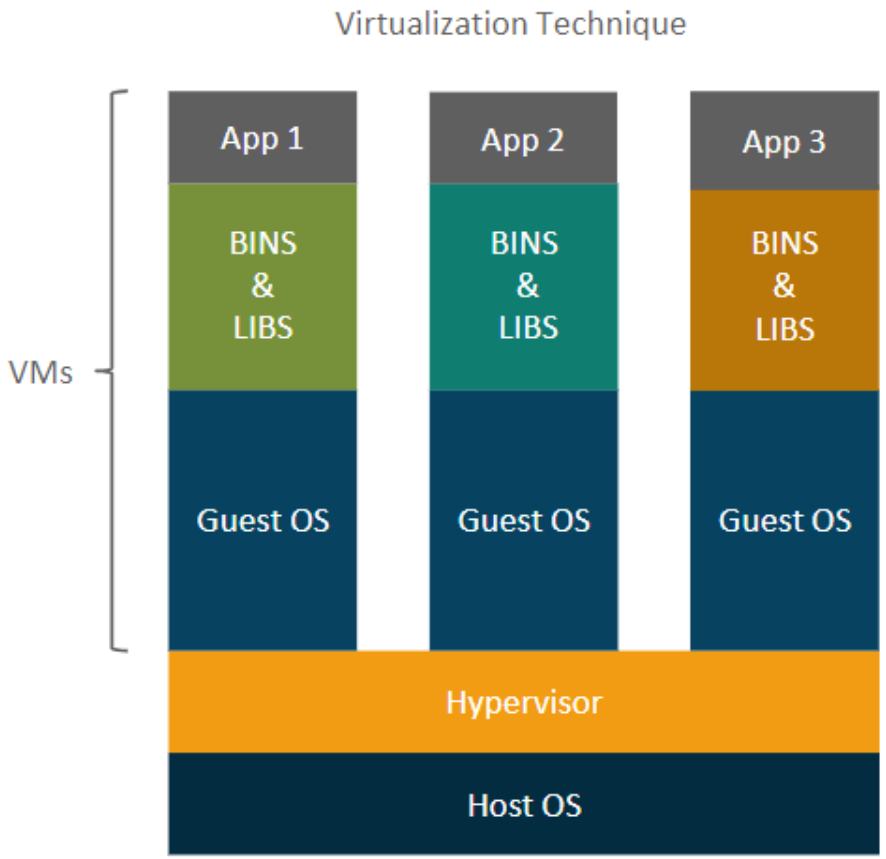


Advantages over Virtualization

- Containers on same OS kernel are lighter & smaller
- Better resource utilization compared to VMs
- Short boot-up process ($1/20^{th}$ of a second)

Containerization is just virtualization at the OS level

Virtualization



Advantages

- Multiple OS in same machine
- Easy maintenance & recovery
- Lower total cost of ownership

Disadvantages

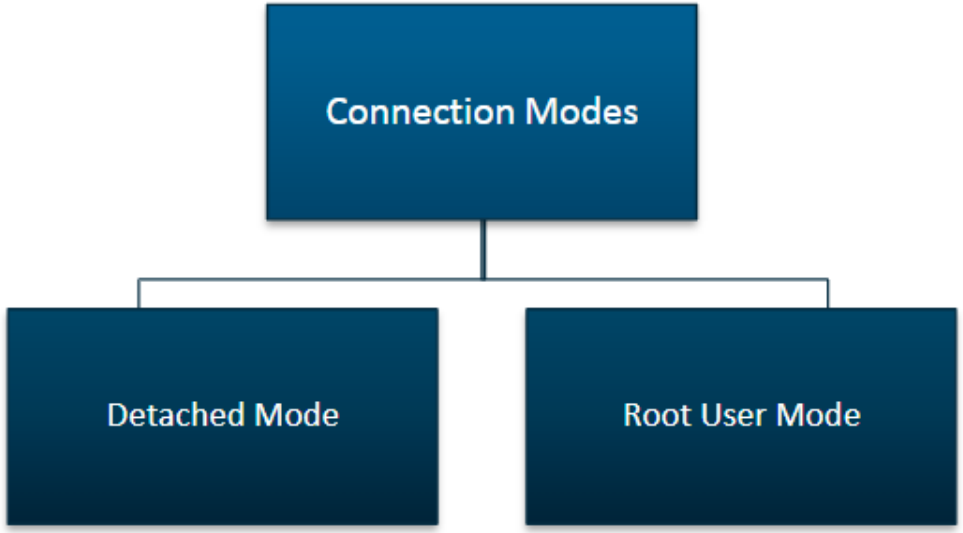
- Multiple VMs lead to unstable performance
- Hypervisors are not as efficient as host OS
- Long boot-up process (approx. 1 minute)

Container v/s VM

Parameters	VM	Container
Work Scope	• If you want to run multiple applications, use VM	• If you want to run multiple copies of a single app, use Containers
Portability	• Working with VMs are platform independent	• Working with containers is platform dependent
Resources	• VMs take up a lot of system resources	• When resources are a constraint, use Containers
Security	• You need not have sudo privileges to import subsystems	• You need to have sudo privileges to import subsystems

Container - Connection Modes

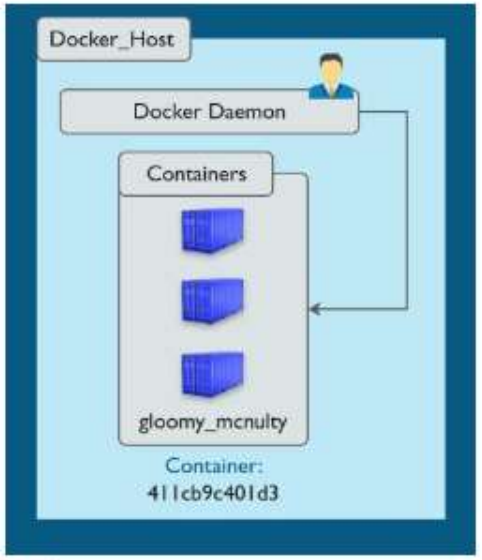
- Container can be connected in the following two modes:



Detached Mode Vs Root Mode

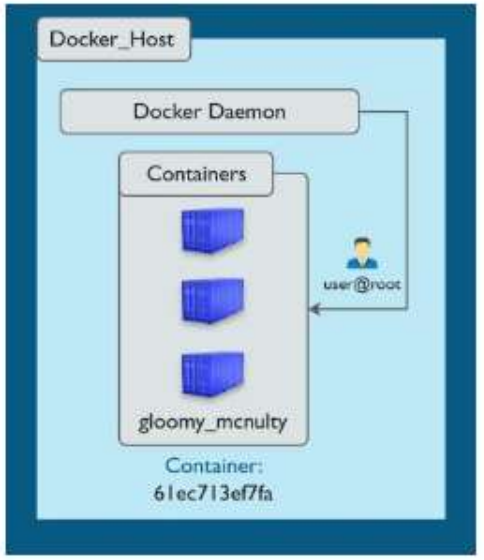
Detached Mode

- Command: `$ docker run -itd ubuntu:xenial`
  - I – Interactive
  - T – Connected to terminal
  - D – Detached mode



Root User Mode

- Command: `$ docker run -it ubuntu:xenial`
  - I – Interactive
  - T – Connected to terminal



Detached Mode Vs Root Mode(Contd..)

Detached Mode

- User manages from Daemon
- Container does not exit after the process within the container is over
- Container could be stopped at a later stage
- Using the $docker attach command user can attach as a root user
- Allows control over other containers

Root User Mode

- User manages from the Root
- Container exits after the process within the container is over
- Container can be restarted at a later stage
- Using the $docker exit command user can attach to the daemon
- Allows control over the container to which user is attached

Pulling an Ubuntu image

docker pull ubuntu: This pulls the ubuntu image from docker hub repository with the tag: latest

```
root@test01:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
aafe6b5e13de: Pull complete
0a2b43a72660: Pull complete
18bdd1e546d2: Pull complete
8198342c3e05: Pull complete
f56970a44fd4: Pull complete
Digest: sha256:f3a61450ae43896c4332bda5e78b453f4a93179045f20c8181043b26b5e79028
Status: Downloaded newer image for ubuntu:latest
```

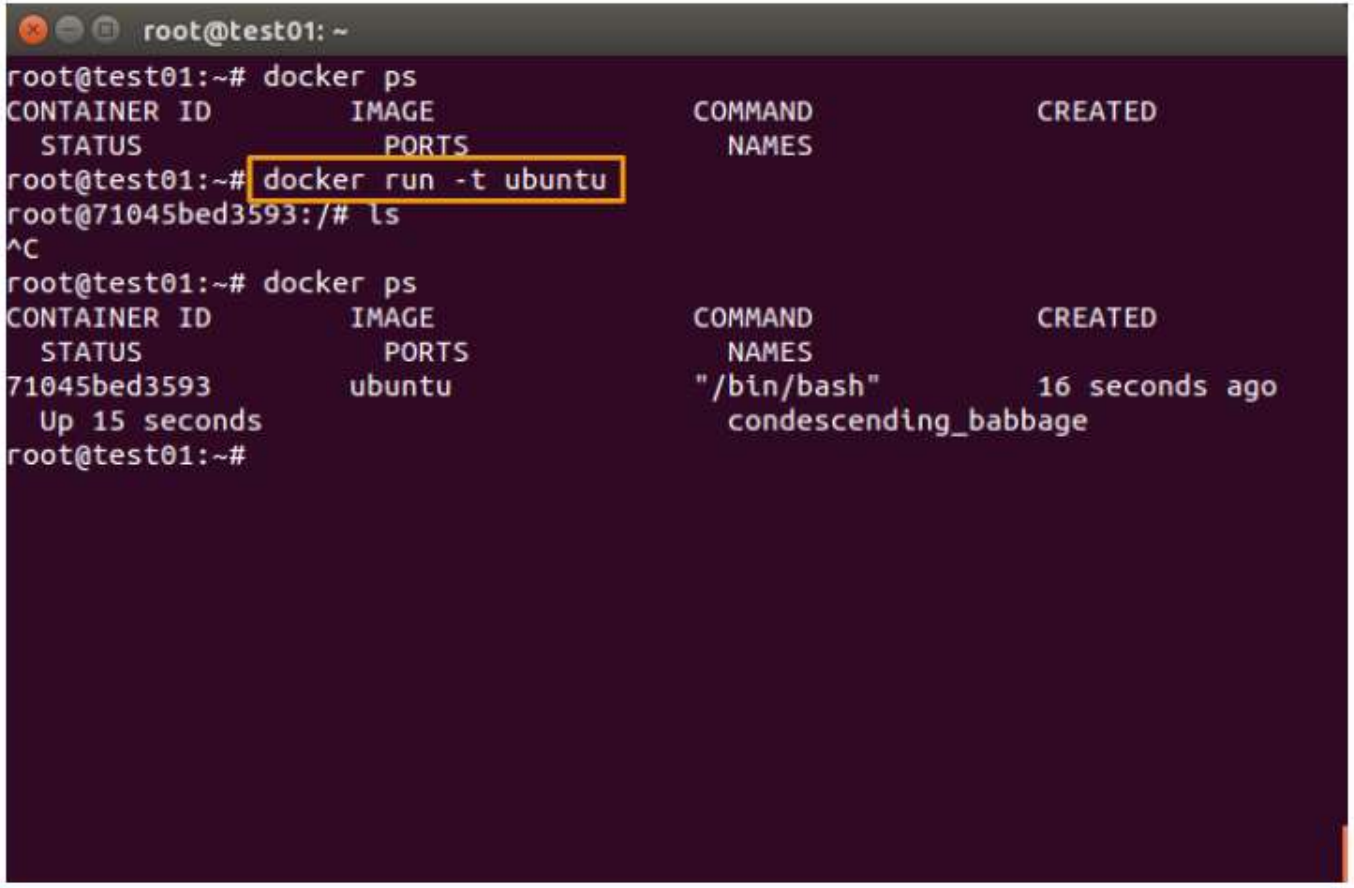
Docker run : Interactive mode

- docker run -i <image-name> : This command helps you to get inside the container

```
root@test01:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
aafe6b5e13de: Pull complete
0a2b43a72660: Pull complete
18bdd1e546d2: Pull complete
8198342c3e05: Pull complete
f56970a44fd4: Pull complete
Digest: sha256:f3a61450ae43896c4332bda5e78b453f4a93179045f20c8181043b26b5e7902
Status: Downloaded newer image for ubuntu:latest
root@test01:~# docker run -i ubuntu
ls
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
```

Docker run : Connected to Terminal

- docker run –t <image-name> : The -t command line options calls a terminal from inside the container. This prevents the container from exiting.



What does the -t or --tty do in Docker?

TTY basically means “a console”. When you are in Windows or Linux and open a terminal to start interacting with the OS via typing some text and getting some text responses, that’s TTY. It’s *pseudo*-TTY in docker because it’s been a pseudo-TTY since being able to connect to an arbitrary computer remotely created the need for a common “terminal” experience independent of what shell the user is running and what the abilities of the physical device I’m connecting to are.

As for why it’s important in docker, try the following:

On the first **docker run**, nothing shows up because we didn't use -t, no TTY (console) so the output from the shell is not rendered anywhere.

On the second **docker run**, I provide **-t** and see some output in my newly allocated TTY. But note when I type **whoami** and hit enter, I see my TeleTYping, but I don’t see anything come back from the shell. Because my typing isn’t attached to the container. (Note: the ^C is me hitting Ctrl-C)

To attach my TTY to the shell, I need the **-i**, short for —**interactive.** In our last one, our TTY is stdin to our shell, the **-i** keeps that stdin open, and all is well.

```
$ docker run busybox /bin/sh

$ docker run -t busybox /bin/sh
/ # whoami
^C

$ docker run -it busybox /bin/sh
/ # whoami
root
/ #
```

Docker run : Interactive Connected to Terminal

- docker run –it <image-name> : This allows the docker container to run in the interactive mode as well as prevent it from exiting

```
root@test01:~# docker run -it ubuntu




root@5dd016c345b1:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@5dd016c345b1:/# hostname
5dd016c345b1
root@5dd016c345b1:/# exit
exit
root@test01:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
  STATUS              PORTS               NAMES
71045bed3593        ubuntu              "/bin/bash"         6 minutes ago
  Up 6 minutes                          condescending_babbage
root@test01:~# docker run -it ubuntu
root@ef6531512f53:/# root@test01:~#
root@test01:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
  STATUS              PORTS               NAMES
ef6531512f53        ubuntu              "/bin/bash"         16 seconds ago
  Up 16 seconds                         serene_galileo
71045bed3593        ubuntu              "/bin/bash"         7 minutes ago
  Up 7 minutes                          condescending_babbage
root@test01:~#
```

Use ctrl+P+Q to get out of container without exiting

Docker run : Detached Mode

- docker run –d <image-name> : This allows the container to run a service in the background
- Containers started in detached mode exits when the root process used to run the container exits.
- A container in detached mode cannot be automatically be removed when it stops, that is - -rm command cannot be used with –d

```
root@test01:~# docker run -d ubuntu
bf0bc4e12f37fa46dff3d45d60338b18c688cd7547c882330eadf5afdf211815
root@test01:~#
```

Connection Modes – Use Cases

Detached Mode

- Containers started in detached mode exit when the root process used to run the container exits.
- A container in detached mode cannot be automatically removed when it stops.
- If this is not your use case and you do wish to automatically remove them, then you would not use the detached mode option.
- The detached mode is used when you want to run one command and (possibly) have the output of the commands sent to some shared data volume for processing later.

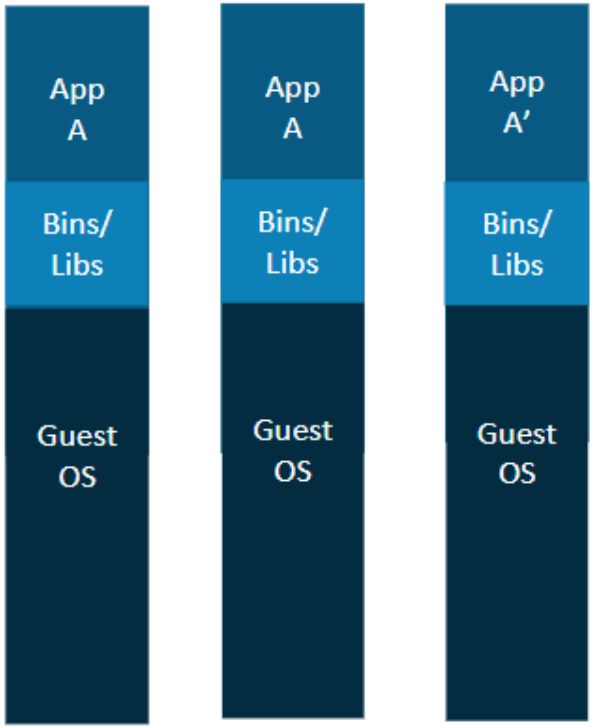
Connection Modes – Use Cases

Root Mode

- 2 examples that do not use detached mode is the use of Dockers plugins in Jenkins where we want to remove the containers after a Jenkins job is executed. This runs in a root user mode with attaching to the foreground and pretending to be a pseudo terminal and on completion close the container.
- Another example will be the starting of a service – where we need to start the service. However, using "service nginx start" with the –d option starts the nginx server but this cannot be used as the container stops after the command is executed.

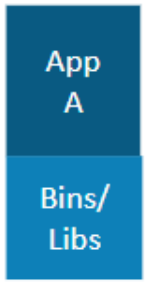
Why are Docker containers lightweight?

VMs

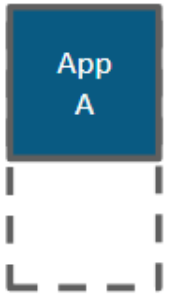


VMs
Every app, every copy of an app, and every slight modification of the app requires a new virtual server

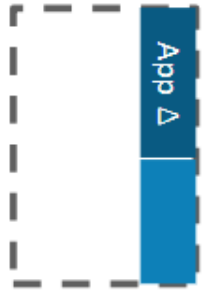
Containers



Original App
(No OS to take up space, resources, or require restart)



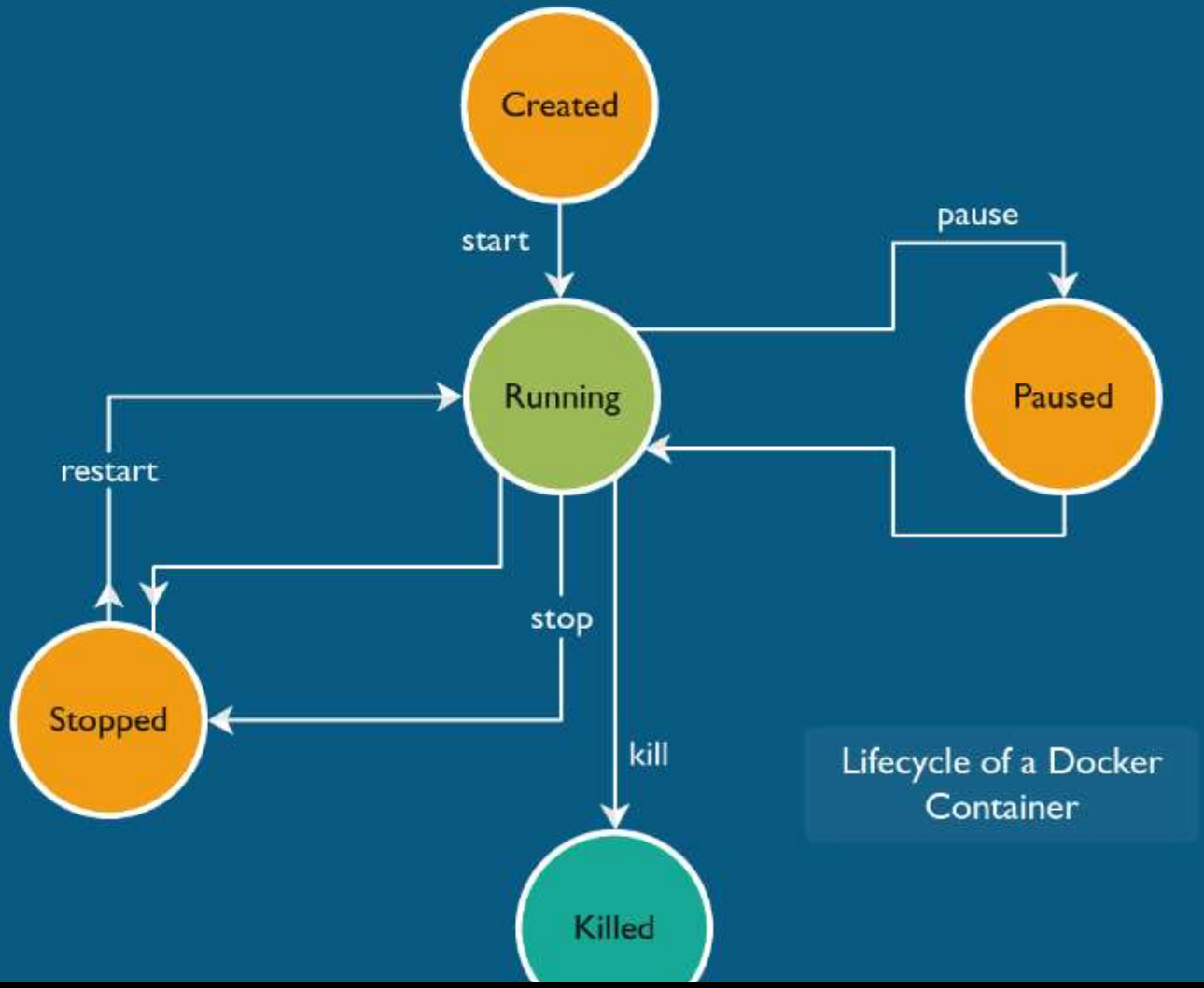
Copy of App
No OS. Can Share bins/libs



Modified App
Union file system allows us to only save the diffs Between container A and container A'

CONTAINER LIFE CYCLE

Lifecycle of a Container



Docker File System - Initiate A Container

- Command : docker run hello-world:latest

```
root@test01:~# docker run hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Container can be initiated using both the “Image name” as well as “Image ID” along with the required tag

Examining Existing Container

- Command : docker ps

```
root@test01:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
root@test01:~#
```

Question: What happened to the container we initiated from our image “hello-world?

- Command : docker ps -a

```
root@test01:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
7bffd209ccfd        hello-world:latest  "/hello"            13 minutes ago      Exited (0) 13 minutes ago                       elated_colden
13cd653434d9        hello-world:latest  "/hello"            13 minutes ago      Exited (0) 13 minutes ago                       small_swartz
216187fe4a0c        busybox             "echo hello-world"  2 hours ago         Exited (0) 2 hours ago                          sleepy_murdock
ed80c141e5f4        ubuntu              "echo hello-world"  2 hours ago         Exited (0) 2 hours ago                          lonely_almeida
6009b99b40da        7c226dc91bb2        "echo 'hello world'"  2 hours ago       Exited (0) 2 hours ago                          test
235b950f9f7f        hello-world         "/hello"            25 hours ago        Exited (0) 25 hours ago                         nostalgic_austin
root@test01:~#
```

Display inference: This container with the name “elated_colden” , with CONTAINER ID (7bffd209ccfd) got created 13 minutes ago and exited 13 minutes ago as soon as its processes were executed.

Naming Container

- Docker assigns default names to the container. The usual format would be firstname_secondname.
- Docker gives us the privilege to name our containers. This can be done with the use of the –name command

```
root@test01:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              7c226dc91bb2        2 hours ago         128.2 MB
busybox             latest              00f017a8c2a6        6 weeks ago         1.11 MB
hello-world         latest              48b5124b2768        3 months ago        1.84 kB
ubuntu              latest              4ca3a192ff2a        4 months ago        128.2 MB
root@test01:~# docker run -d --name=edureka-container hello-world:latest
5a337d8167a13d0287c5e198cebf913e36a8d306d29c336aa8c24abca8a3e12a
root@test01:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
root@test01:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
5a337d8167a1        hello-world:latest  "/hello"            47 seconds ago      Exited (0) 43 seconds ago                       edureka-container
7bffd209ccfd        hello-world:latest  "/hello"            29 minutes ago      Exited (0) 28 minutes ago                       elated_colden
13cd653434d9        hello-world:latest  "/hello"            29 minutes ago      Exited (0) 29 minutes ago                       small_swartz
216187fe4a0c        busybox             "echo hello-world"  2 hours ago         Exited (0) 2 hours ago                          sleepy_murdock
ed80c141e5f4        ubuntu              "echo hello-world"  2 hours ago         Exited (0) 2 hours ago                          lonely_almeida
6009b99b40da        7c226dc91bb2        "echo 'hello world'"  2 hours ago       Exited (0) 2 hours ago                          test
235b950f9f7f        hello-world         "/hello"            26 hours ago        Exited (0) 26 hours ago                         nostalgic_austin
root@test01:~#
```

The container would be named as edureka-container

Getting Attached To a Container

- When we are running containers in Detached mode (Daemonised Mode) we can still attach to the container if required.

```
root@ubuntu:~# docker run -itd ubuntu
07770217e5850902c275dcb24e50743c298e26ae88dd7d55cdf9a3404c561212
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
                    PORTS               NAMES
07770217e585        ubuntu              "/bin/bash"         10 seconds ago      Up 9 se
conds                                   infallible_aryabhata
5f6c86b91b86        hello-world         "/hello"            33 minutes ago      Restart
ing (0) 8 minutes ago                   hello-world


root@ubuntu:~# docker attach 07770217e585
root@07770217e585:/#
```

Stopping a container: Default Mode

When container is initiated in detached mode. It keeps running. It can be stopped in following two ways:

- Get attached to the root and exit
- Stop the container using the “stop” command : docker stop <container-id>

```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              a8493f5f50ff        2 weeks ago         192.5 M
B
busybox             latest              00f017a8c2a6        6 weeks ago         1.11 MB
hello-world         latest              48b5124b2768        3 months ago        1.84 kB
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE                   COMMAND             CREATED              STATUS
                    PORTS                   NAMES
5f6c86b91b86        hello-world             "/hello"            About a minute ago   Restar
ting (0) 25 seconds ago                     hello-world
root@ubuntu:~# docker stop 5f6c86b91b86
5f6c86b91b86
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
            PORTS               NAMES
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
                    PORTS               NAMES
5f6c86b91b86        hello-world         "/hello"            2 minutes ago       Exited
(0) 15 seconds ago                      hello-world
root@ubuntu:~#
```

By default a container exists as soon as its processes are executed

Restarting a Container

- Start a stopped container by using the “start” command: $ docker start

```
root@ubuntu:~# docker stop 5f6c86b91b86
5f6c86b91b86
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
             PORTS               NAMES
root@ubuntu:~# docker start 5f6c86b91b86
5f6c86b91b86
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
                    PORTS               NAMES
5f6c86b91b86        hello-world         "/hello"            12 minutes ago      Restart
ing (0) Less than a second ago                          hello-world
root@ubuntu:~#
```

Inspect Container

- Inspect containers and to view important information regarding config, IP address etc., where both, the running and stopped containers can be inspected by the command: $ docker inspect

```
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
                    PORTS               NAMES
5f6c86b91b86        hello-world         "/hello"            12 minutes ago      Restart
ing (0) Less than a second ago                              hello-world
root@ubuntu:~# docker inspect 5f6c86b91b86
[
    {
        "Id": "5f6c86b91b86cd07f4262c530abb2a40530fb61bc1a9d1085bdf0b59a12d412c",
        "Created": "2017-04-27T12:33:53.037035206Z",
        "Path": "/hello",
        "Args": [],
        "State": {
            "Status": "restarting",
            "Running": true,
            "Paused": false,
            "Restarting": true,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2017-04-27T12:47:42.960898398Z",
            "FinishedAt": "2017-04-27T12:47:42.971435916Z"
        },
        "Image": "sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf02
33",
        "ResolvConfPath": "/var/lib/docker/containers/5f6c86b91b86cd07f4262c530abb2a405
30fb61bc1a9d1085bdf0b59a12d412c/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/5f6c86b91b86cd07f4262c530abb2a40530
fb61bc1a9d1085bdf0b59a12d412c/hostname",
```

REMOVING IMAGES

Removing Images

1. With Associated Containers
2. Without Associated Containers
3. Removing by force

Removing Images with attached container:

- Docker will not allow you to remove an image if there are containers attached with the image
- The running containers will have to be removed first and then the image can be removed
- Using –f command, the images can be removed forcefully making the associated containers orphan which is not advisable though.

```
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
            PORTS               NAMES
07770217e585        ubuntu              "/bin/bash"         17 hours ago        Up 2 mi
nutes                           infallible_aryabhata
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              f7b3f317ec73        3 days ago          117.3 M
B
centos              latest              a8493f5f50ff        3 weeks ago         192.5 M
B
busybox             latest              00f017a8c2a6        7 weeks ago         1.11 MB
hello-world         latest              48b5124b2768        3 months ago        1.84 kB
root@ubuntu:~# docker rmi f7b3f317ec73
Error response from daemon: conflict: unable to delete f7b3f317ec73 (cannot be forced)
- image is being used by running container 07770217e585
root@ubuntu:~#
```

Removing Images

1. With Associated Containers
2. Without Associated Containers
3. Removing by force

Command: $ docker rmi <Image Name/ID>

- One liner to remove all the Docekr container
  - docker stop $(docker ps -a -q)
  - docker rm $(docker ps -a -q)

```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              f7b3f317ec73        3 days ago          117.3 M
B
centos              latest              a8493f5f50ff        3 weeks ago         192.5 M
B
busybox             latest              00f017a8c2a6        7 weeks ago         1.11 MB
hello-world         latest              48b5124b2768        3 months ago        1.84 kB
root@ubuntu:~# docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a
1ac8d7
Deleted: sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf0233
Deleted: sha256:98c944e98de8d35097100ff70a31083ec57704be0991a92c51700465e4544d08
root@ubuntu:~#
```

Removing Images

1. With Associated Containers
2. Without Associated Containers
3. Removing by force

- Docker allows to remove images with associated containers forcefully. This is not recommended though.
- You can use the –f command to perform this action.

```
root@ubuntu:~# docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world

Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Image is up to date for hello-world:latest
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              f7b3f317ec73        3 days ago          117.3 M
B
centos              latest              a8493f5f50ff        3 weeks ago         192.5 M
B
busybox             latest              00f017a8c2a6        7 weeks ago         1.11 MB
hello-world         latest              48b5124b2768        3 months ago        1.84 kB
root@ubuntu:~# docker run -d hello-world
da3f52aa43ad5163d89b7e815212e4499d723df148aeb436308fe667ec49872d
root@ubuntu:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
          PORTS               NAMES
07770217e585        ubuntu              "/bin/bash"         17 hours ago        Up 5 mi
nutes                           infallible_aryabhata
root@ubuntu:~# docker rmi hello-world
Error response from daemon: conflict: unable to remove repository reference "hello-worl
d" (must force) - container 5f6c86b91b86 is using its referenced image 48b5124b2768
root@ubuntu:~# docker rmi 48b5124b2768
Error response from daemon: conflict: unable to delete 48b5124b2768 (must be forced) -
image is being used by stopped container 5f6c86b91b86
root@ubuntu:~# docker rmi -f 48b5124b2768
Untagged: hello-world:latest
Untagged: hello-world@sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a
1ac8d7
Deleted: sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf0233
root@ubuntu:~#
```

When to use remove with –f option?

- Use this only for images that have containers.
- As a best practice, containers should be disposed when they are done.
- If this is not possible, you will need to script to dispose the containers before removing the image.
- REITERATED WARNING – Use of the –f option is never a good option, unless you know that the image itself warranted removal because of
    - Poor performance
    - Security issues

SHARING & COPYING

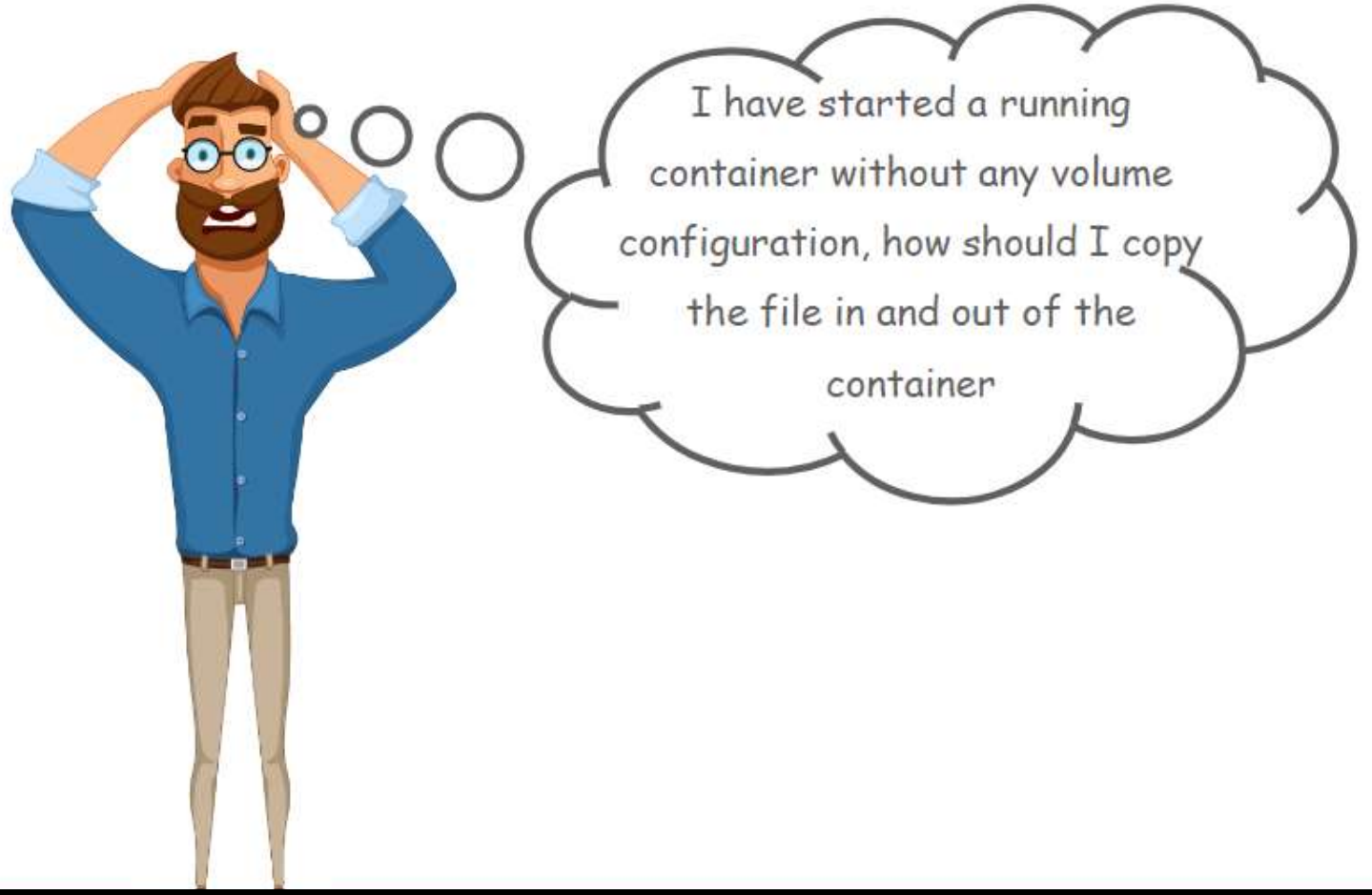
Sharing Docker Host data with Containers

- Use the -v option of docker run to mount a host volume into a container.
- Sharing the working directory of the host within edureka/directory in a container.
- Mount the working directory in the host into the /edureka directory in the container.
- Creating files or directories within the container, let the changes to be written directly to the host working directory

```
root@test01:~# ls
bindthis  cont1.txt   Dockerfile~  foobar       hello-world~  python
busybox   Dockerfile  file.txt     hello-world  host.txt      python~
root@test01:~# docker run -it -v "$PWD":/edureka ubuntu /bin/bash
root@0a4c3cd83641:/# ls
bin   dev      etc   lib    media  opt   root  sbin  sys  usr
boot  edureka  home  lib64  mnt    proc  run   srv   tmp  var
root@0a4c3cd83641:/# ls edureka
Dockerfile   bindthis  cont1.txt  foobar       hello-world~  python
Dockerfile~  busybox   file.txt   hello-world  host.txt      python~
root@0a4c3cd83641:/# touch /edureka/edurekaDocker
root@0a4c3cd83641:/# exit
exit
root@test01:~# ls -l edurekaDocker
-rw-r--r-- 1 root root 0 May 11 10:54 edurekaDocker
root@test01:~#
```

By default docker mounts the volume in read-write mode

Docker Problem I



Solution: Copying Data to and from Containers

- Use docker cp command to copy files from a running container to the docker host
- The command allows you to copy files to and from the host to container
- The usage of docker cp command is as follows :

```
root@test01:~# docker cp
"docker cp" requires exactly 2 argument(s).
See 'docker cp --help'.

Usage:  docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
        docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH

Copy files/folders between a container and the local filesystem
root@test01:~#
```

Solution: Copying Data to and from Containers contd…

- Copy data from Container to Host

```
root@test01:~# docker run -d --name testcopy ubuntu sleep 360
ab6e79055874b95ce464d453b87d9c6c3ad6de5861567539d3d43c92e0880e49
root@test01:~# docker exec -it testcopy /bin/bash
root@ab6e79055874:/# cd /root
root@ab6e79055874:~# echo 'I am in the container' >file.txt
root@ab6e79055874:~# exit
exit
root@test01:~# docker cp testcopy:/root/file.txt .
root@test01:~# cat file.txt
I am in the container
root@test01:~#
```

Solution: Copying Data to and from Containers contd…

- Copy data from Host to Container

```
root@test01:~# echo 'I am in the host' > host.txt
root@test01:~# docker cp host.txt testcopy:/root/host.txt
root@test01:~#
```