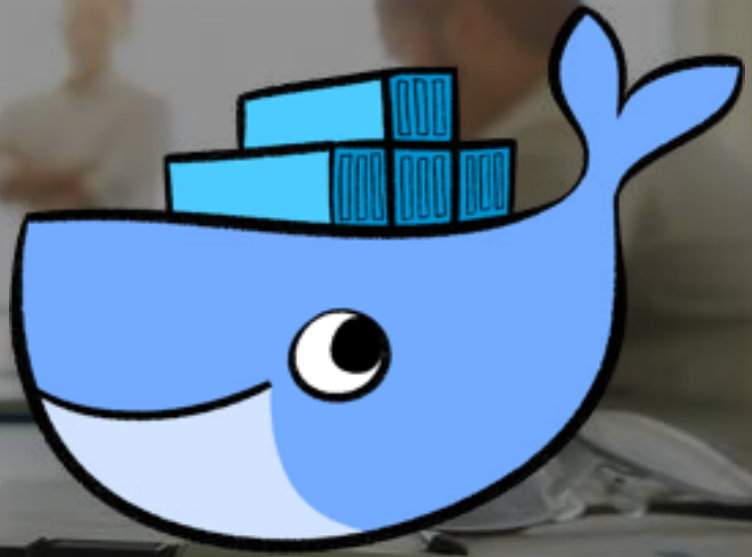
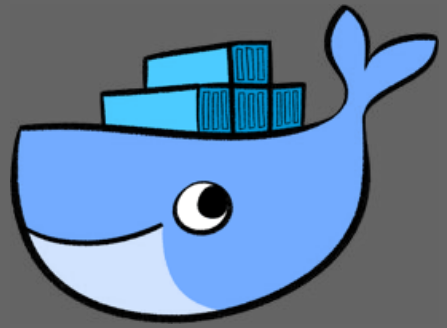
# Docker Tutorial for Beginners



IntelliPaat

# Agenda for Today's Session

- Introduction to Docker
- Common Docker Operations
- What is Dockerfile?
- Docker Volumes
- Breaking the Monolith using Docker
- What is Docker Compose?
- Container Orchestration
- Deploying a Multi Tier App on Docker Swarm

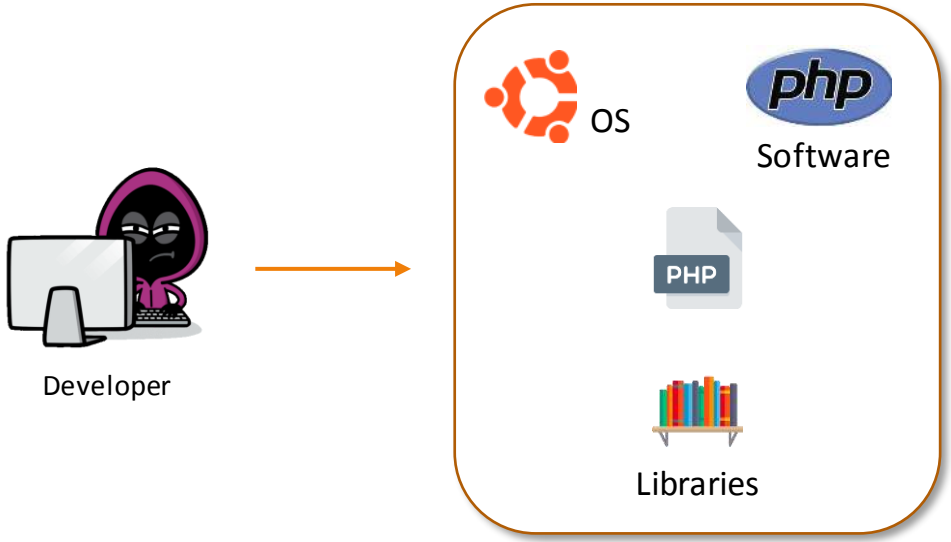


# Introduction to Docker

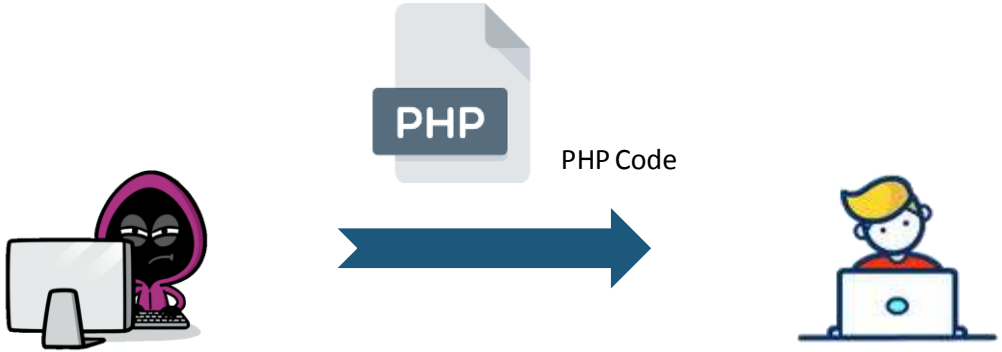
## Why Docker?

# Problems before Docker

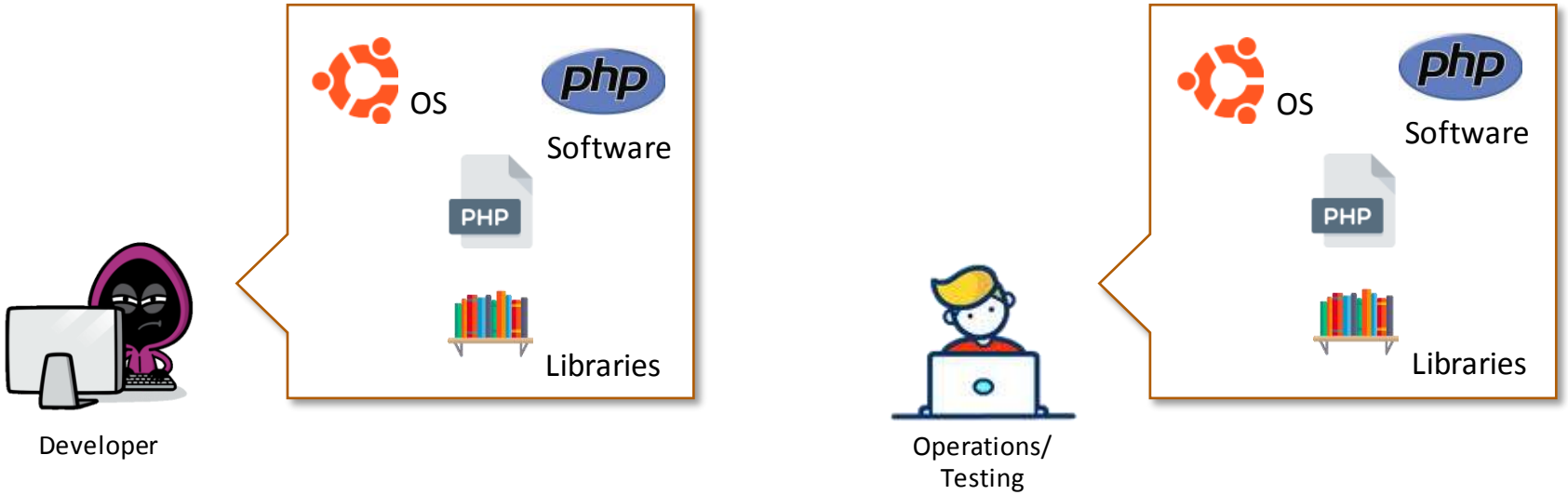
Imagine you are a Developer, and you are creating a website on PHP



# The Problem



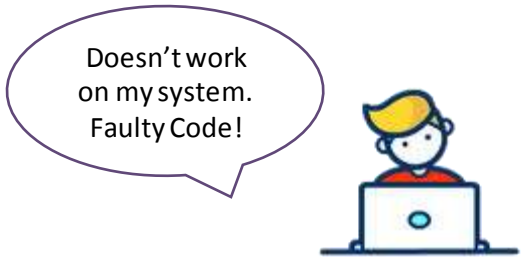
# The Problem



# Problems before Docker

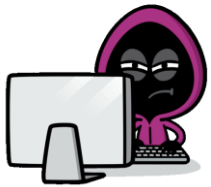
Developers used to run the code on their system, it would run perfectly. But the same code did not run on the operations team system.


Developer




Operations/ Testing

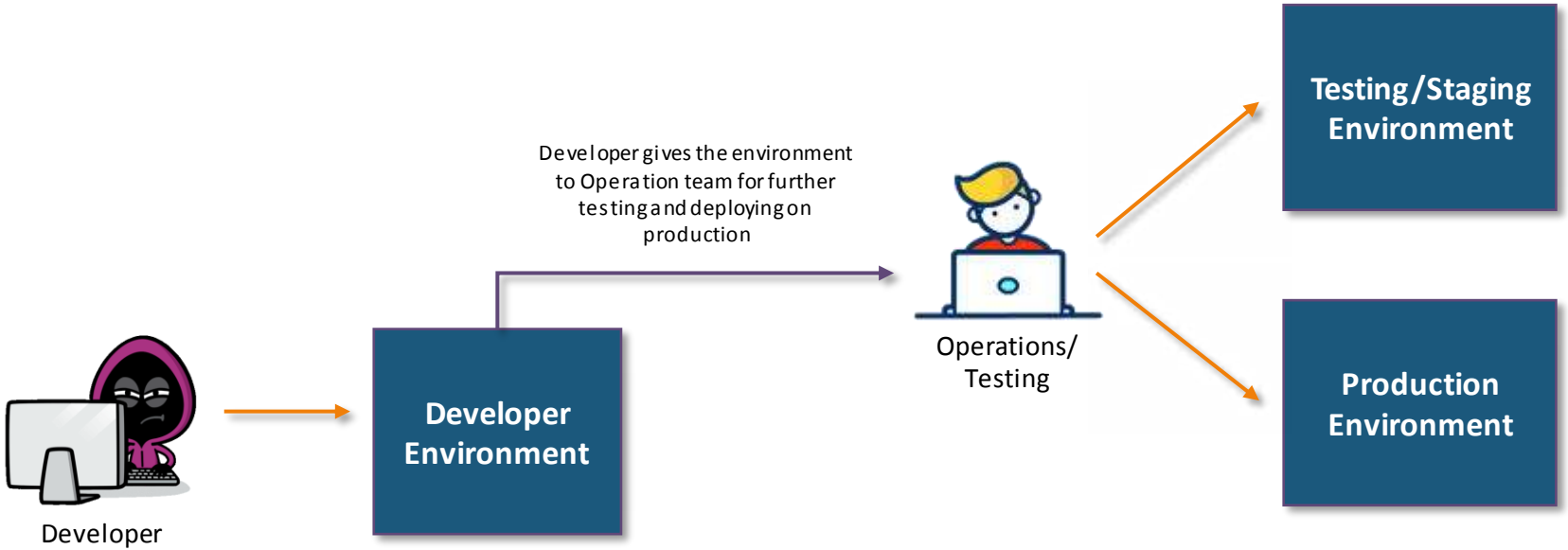
# The Problem

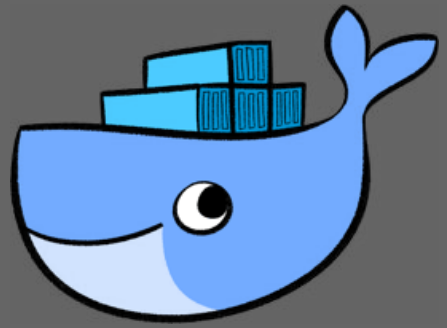


Developer


We needed an entity which can "**contain**" all the software dependencies, and can be ported on to other computers as plug & play package

# The Answer



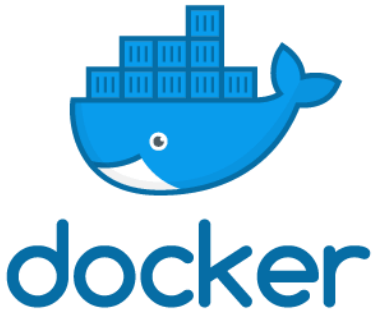


# Introduction to Docker

## What is Docker?

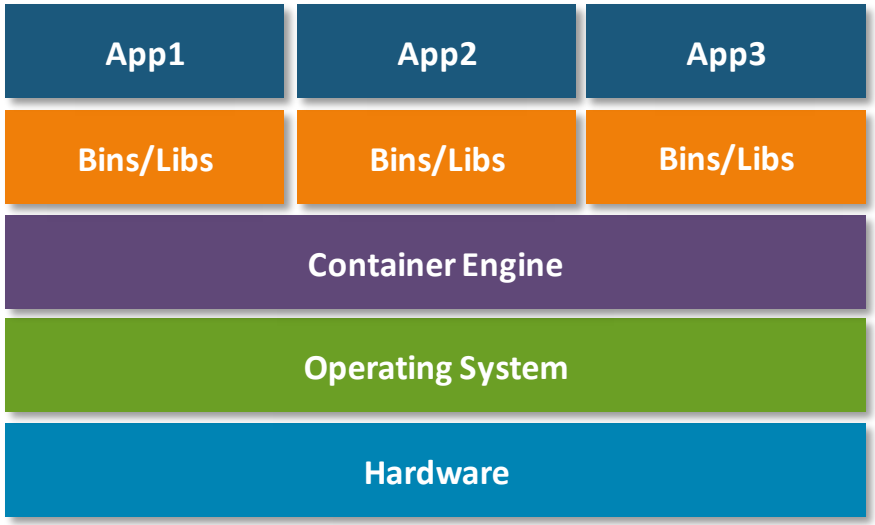
# What is Docker?

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". It was first released in 2013 and is developed by Docker, Inc. Docker is used to run software packages called "containers".

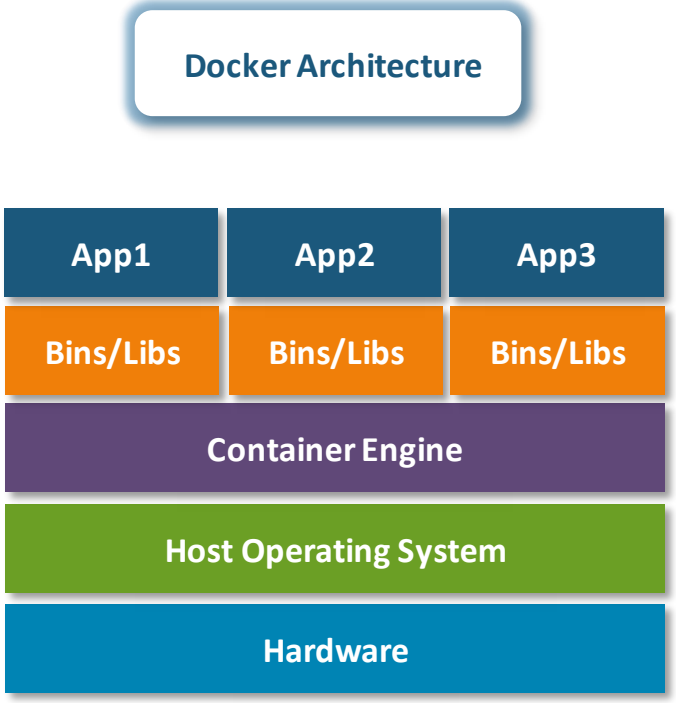


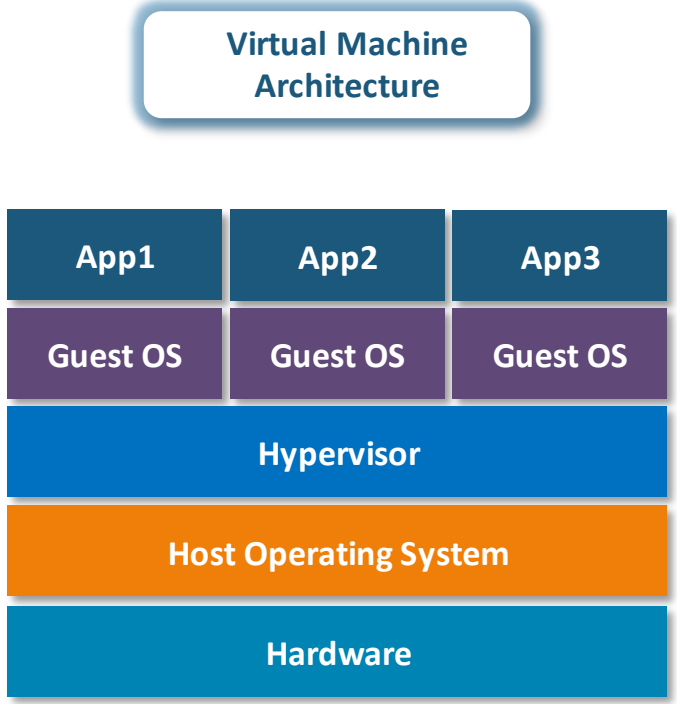
# What is Docker?

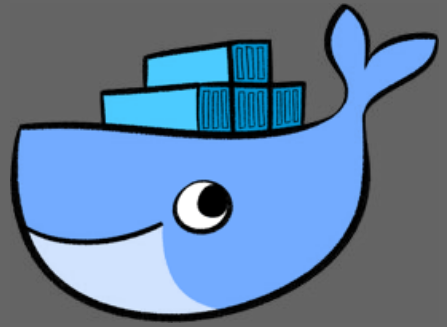
Application **containerization** is an OS-level virtualization method used to deploy and run distributed applications without launching an entire virtual machine (VM) for each app



# Docker vs Virtual Machine





# Introduction to Docker

## Docker Installation

# Docker Installation

Installation
on Mac

Installation on
Windows

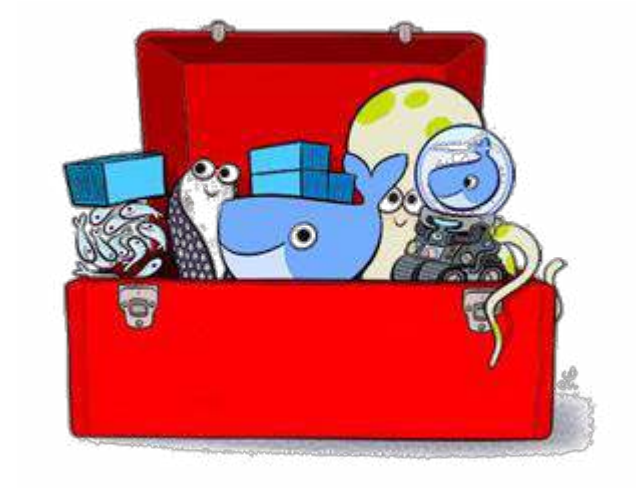
Installation
on Ubuntu

# Docker Installation – Mac

- Installation on Mac
- Installation on Windows
- Installation on Ubuntu

One needs to Install Docker Toolbox for Mac, for using Docker

https://store.docker.com/editions/community/docker-ce-desktop-mac



# Docker Installation – Windows

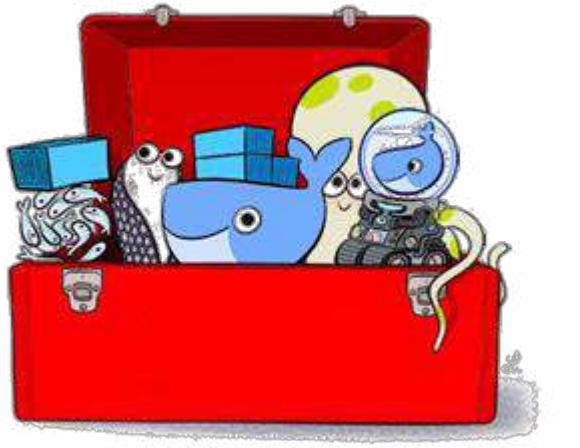
Installation on Mac

Installation on Windows

Installation on Ubuntu

One needs to Install Docker Toolbox for Windows, for using Docker

https://store.docker.com/editions/community/docker-ce-desktop-windows

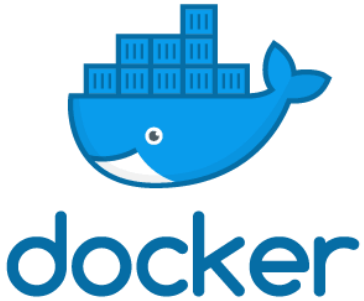


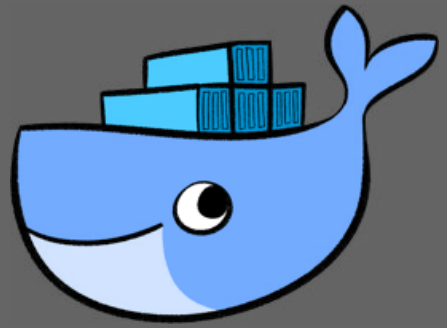
# Docker Installation – Ubuntu

- Installation on Mac
- Installation on Windows
- Installation on Ubuntu

One needs to Install Docker Toolbox for Windows, for using Docker

```
sudo apt-get update
sudo apt-get install docker.io
```



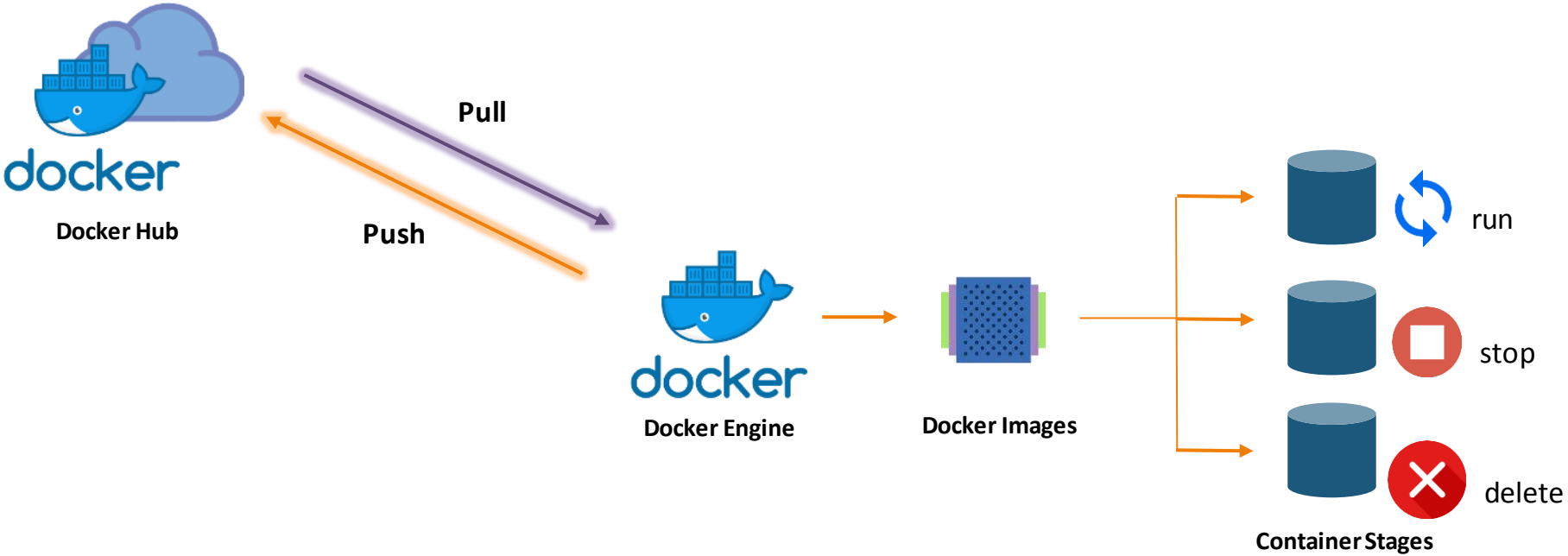


Introduction to Docker

IntelliPaat

# Docker Container Lifecycle

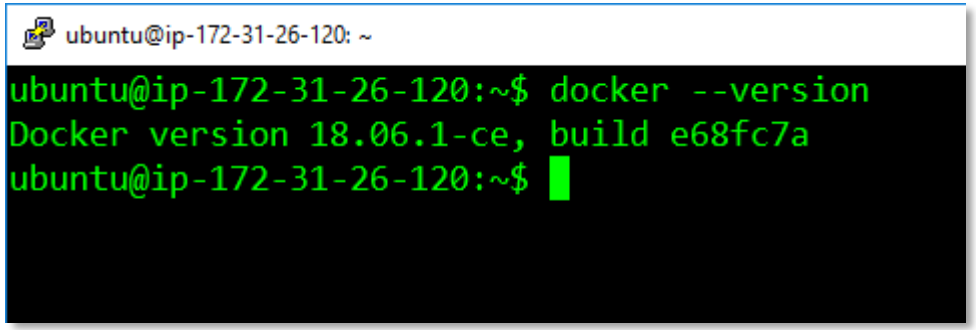
# Docker Container Lifecycle



# Common Docker Operations

# Common Docker Operations

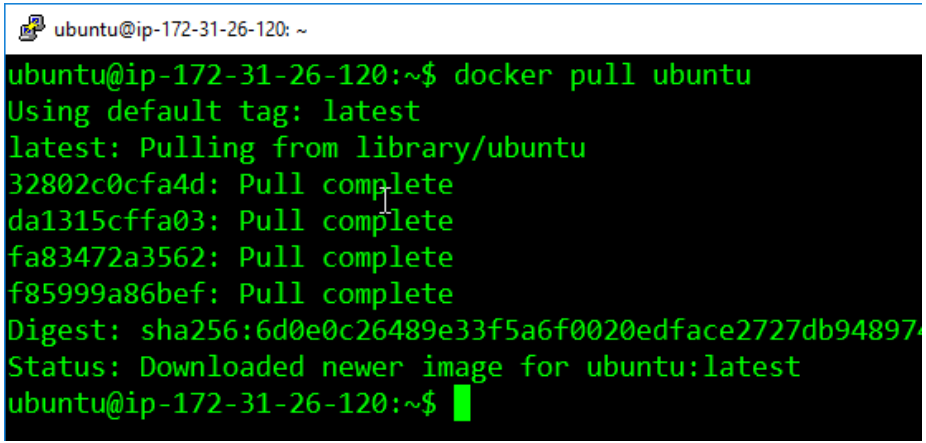
```
docker --version
```




This commands helps you know the installed version of the docker software on your system

# Common Docker Operations

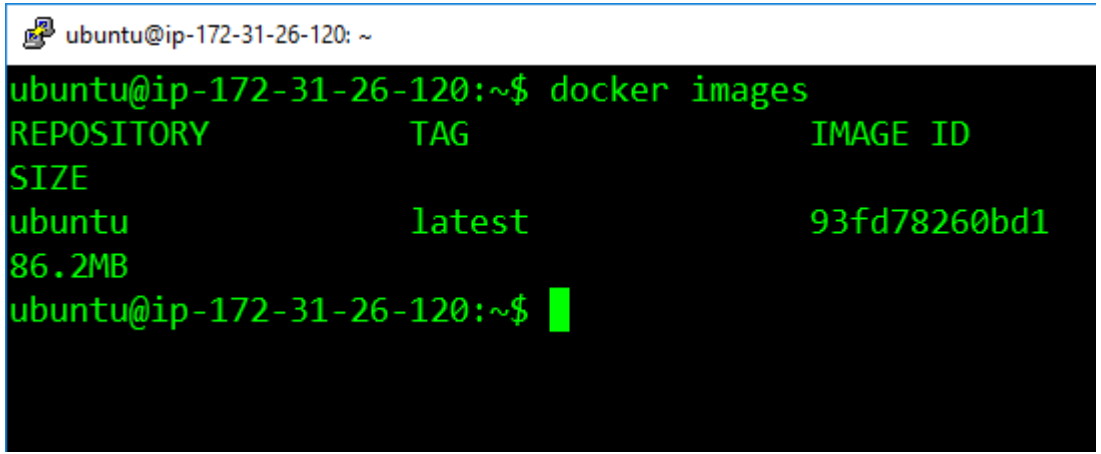
```
docker pull <image-name>
```




This commands helps you pull images from the central docker repository

# Common Docker Operations

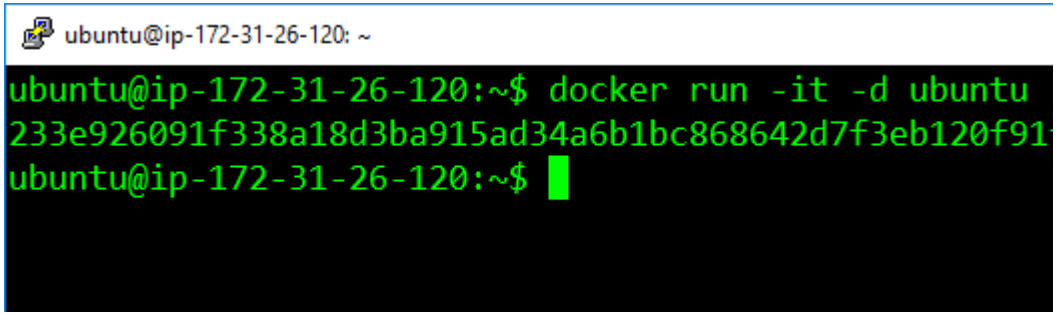
**docker images**




This command helps you in listing all the docker images, downloaded on your system

# Common Docker Operations

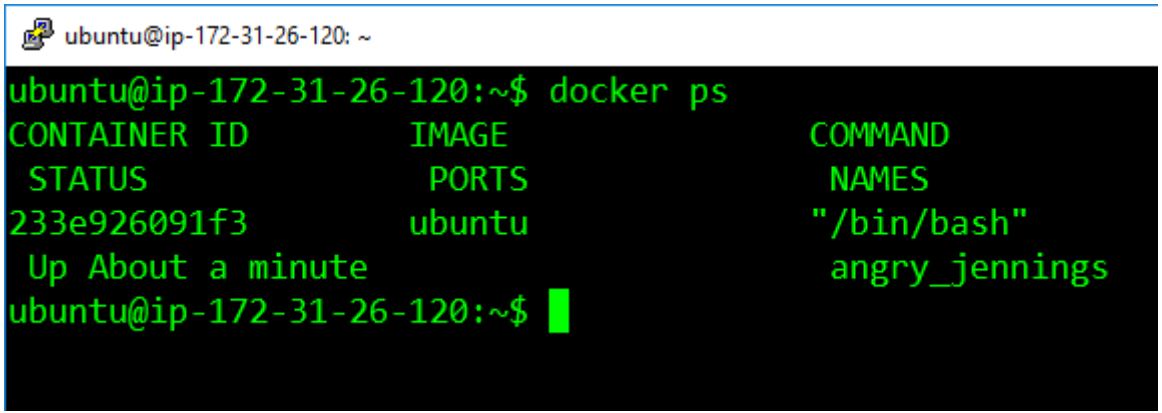
```
docker run <image-name>
```




This command helps in running containers, from their image name

# Common Docker Operations

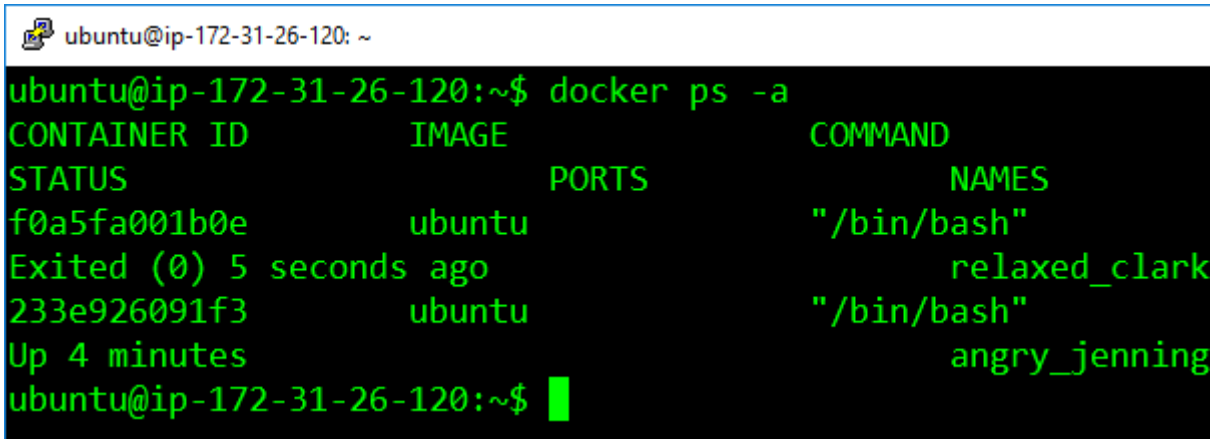
**docker ps**




This command helps in listing all the containers which are **running** in the system

# Common Docker Operations

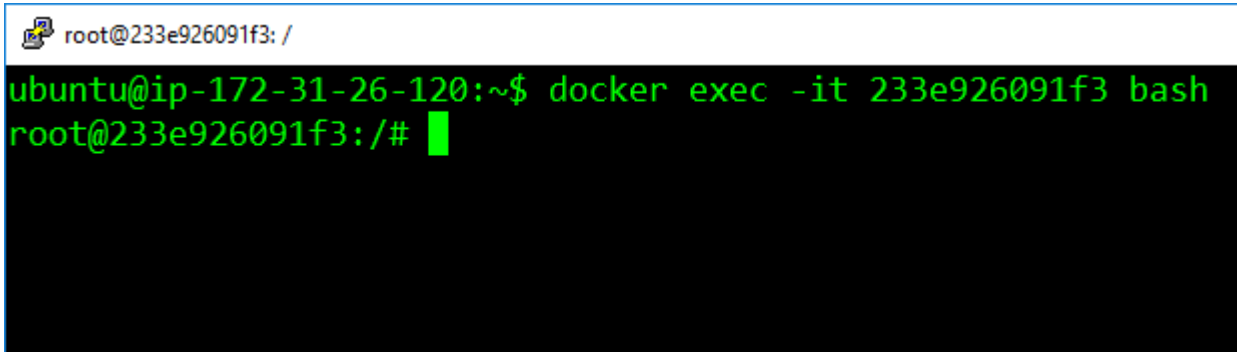
```
docker ps -a
```




If there are any stopped containers, they can be seen by adding the “-a” flag in this command

# Common Docker Operations

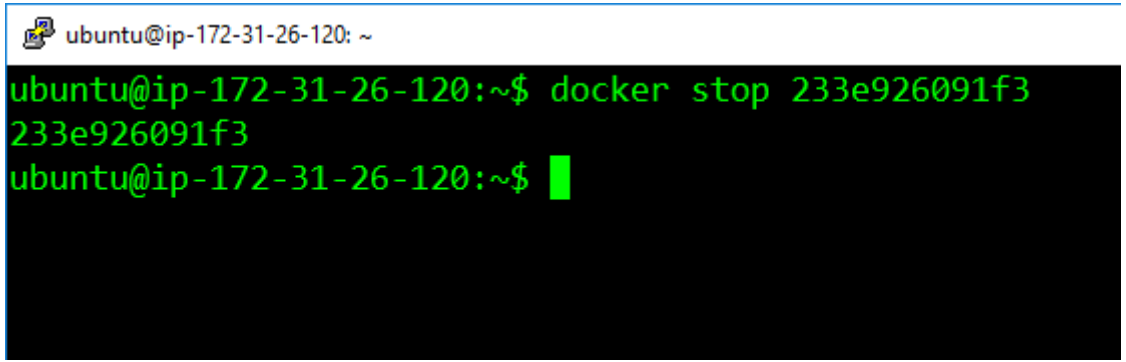
```
docker exec <container-id>
```




For logging into/accessing the container, one can use the exec command

# Common Docker Operations

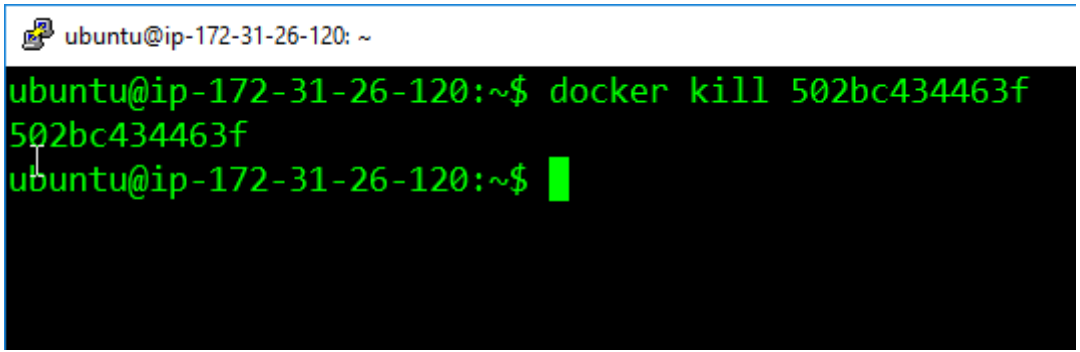
```
docker stop <container-id>
```




For stopping a running container, we use the “stop” command

# Common Docker Operations

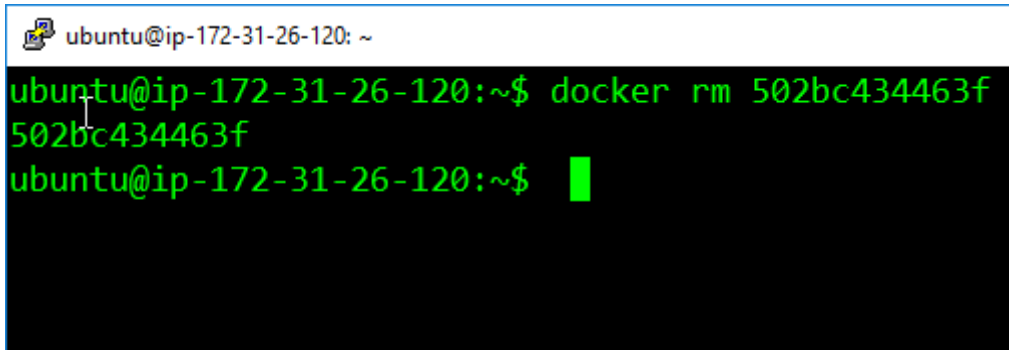
```
docker kill <container-id>
```




This command kills the container by stopping its execution immediately. The difference between 'docker kill' and 'docker stop'. 'docker stop' gives the container time to shutdown gracefully, in situations when it is taking too much time for getting the container to stop, one can opt to kill it

# Common Docker Operations

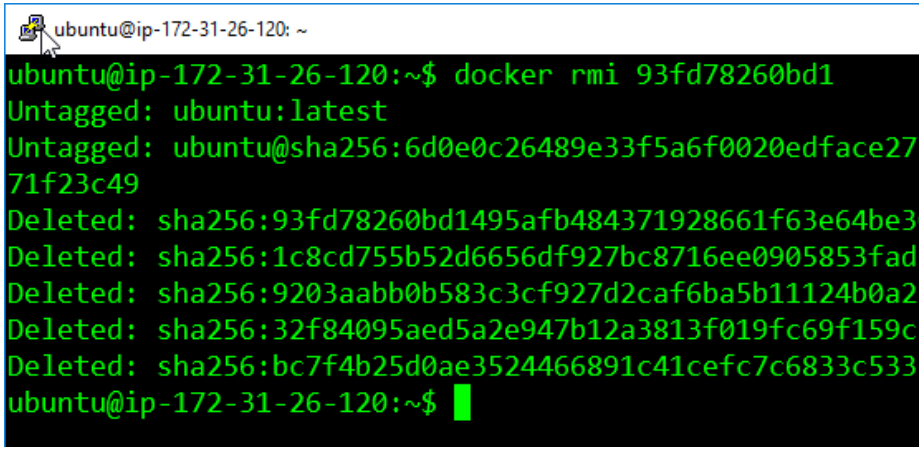
```
docker rm <container-id>
```




To remove a stopped container from the system, we use the “rm” command

# Common Docker Operations

```
docker rmi <image-id>
```



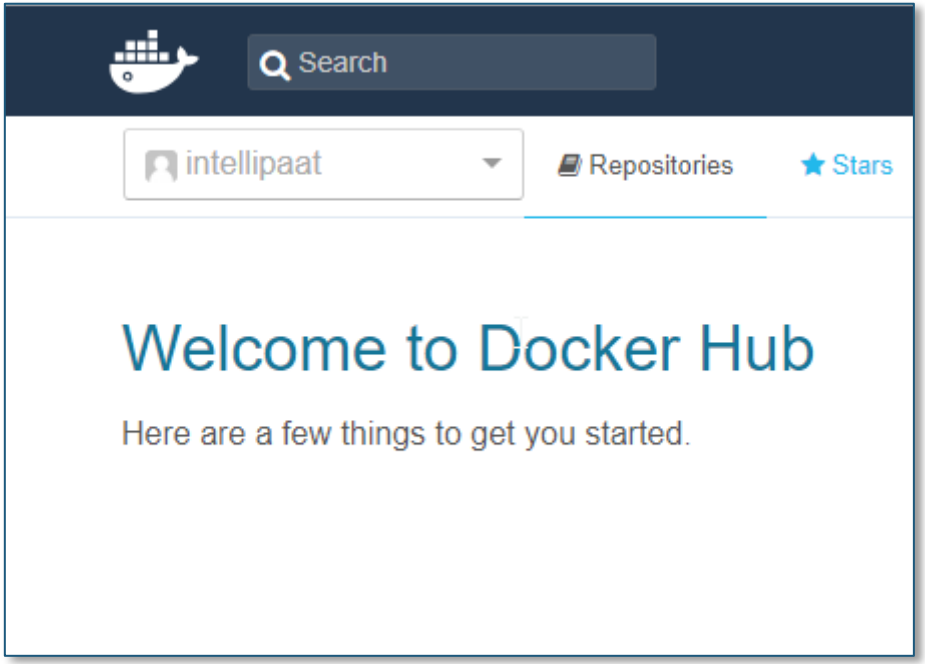

To remove an image from the system we use the command “rmi”

# Common Docker Operations

## Creating a Docker Hub Account

# Creating a Docker Hub Account

1. Navigate to https://hub.docker.com
2. Sign up on the website
3. Agree to the terms and conditions
4. Click on Sign up
5. Check your email, and verify your email by clicking the link
6. Finally, login using the credentials you provided on the sign up page



# Common Docker Operations

## Saving changes to a container

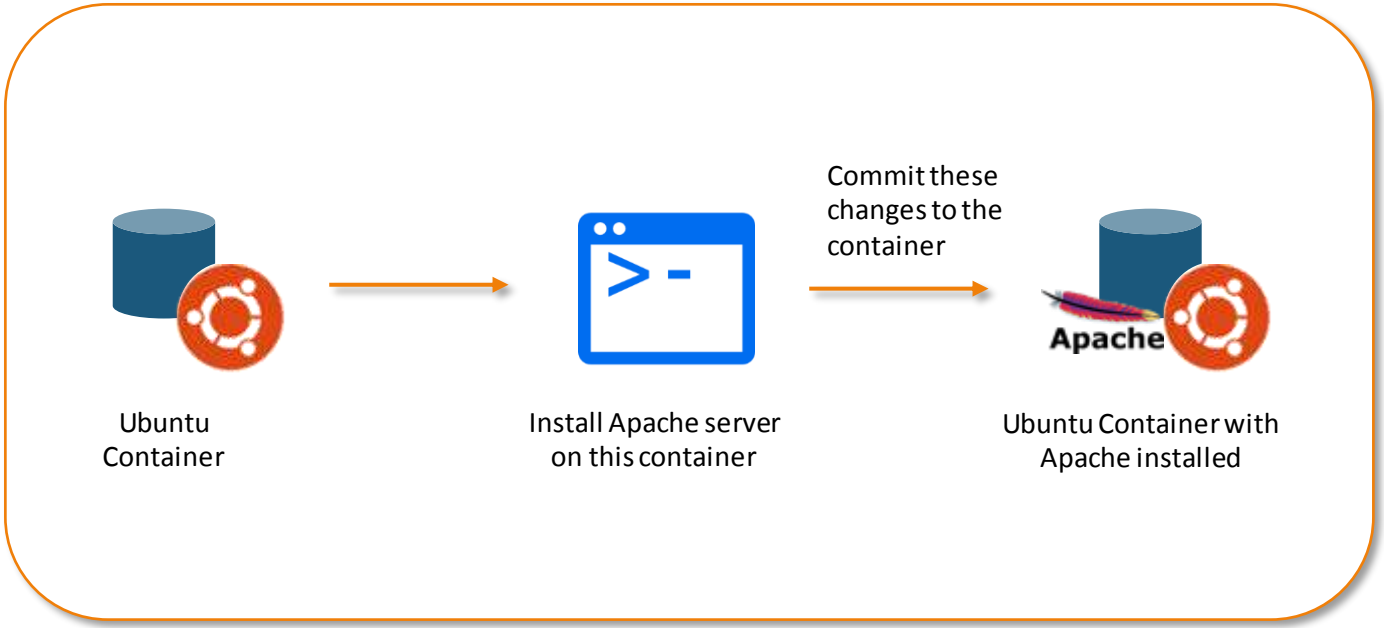
# Saving Changes to a Container

```
docker commit <container-id> <name-for-image>
```

```
~$docker commit b5593da4cc72 new
sha256:4e5bc9bf0e89471361a5a3f70187c5a3b45b3b04e9875832ea55d3f45b91a6ea
~$
```

With this command, a new image is created which can be seen under **docker images** with the same name as passed in the command

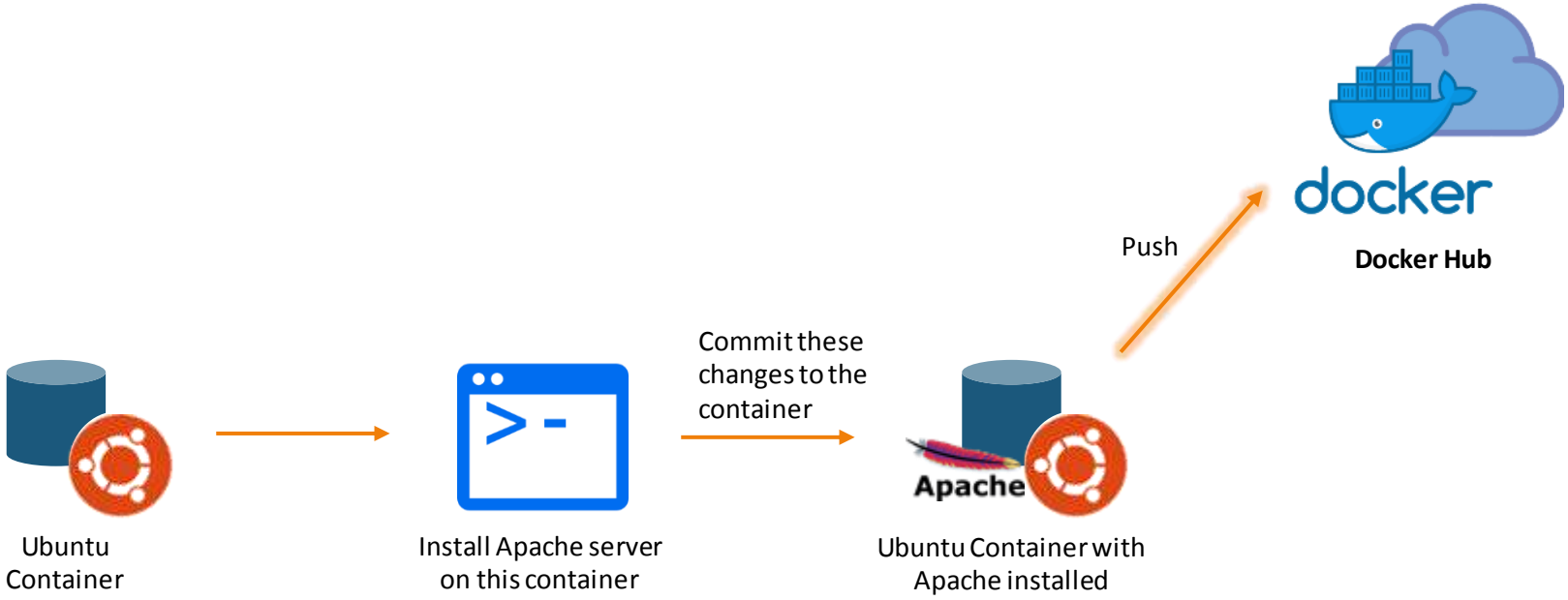
# Saving Changes to a Container



# Common Docker Operations

## Pushing to DockerHub

# Pushing a Container to Docker Hub

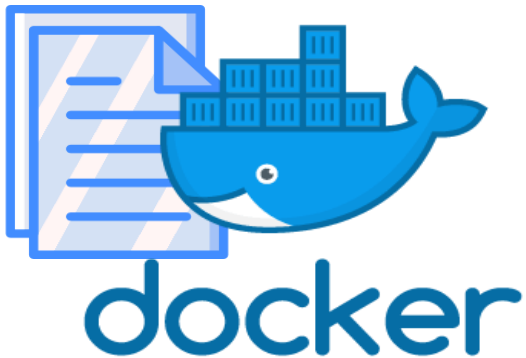


## What is a Dockerfile?

# Introduction to Dockerfile

# Introduction to Dockerfile

A **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image. Using **docker** build users can create an automated build that executes several command-line instructions in succession.



# Various Commands in Dockerfile

- **FROM**
- ADD
- RUN
- CMD
- ENTRYPOINT
- ENV

The FROM keyword is used to define the base image, on which we will be building

Example

```
FROM ubuntu
```

Dockerfile

# Various Commands in Dockerfile

- FROM
- ADD
- RUN
- CMD
- ENTRYPOINT
- ENV

The ADD keyword is used to add files to the container being built. The syntax which is followed is

**ADD <source> <destination in container>**

Example

```
FROM ubuntu
ADD .  /var/www/html
```

Dockerfile

# Various Commands in Dockerfile

- FROM
- ADD
- **RUN**
- CMD
- ENTRYPOINT
- ENV

The **RUN** keyword is used to add layers to the base image, by installing components. Each RUN statement, adds a new layer to the docker image

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD .  /var/www/html
```

Dockerfile

# Various Commands in Dockerfile

- FROM
- ADD
- RUN
- **CMD**
- ENTRYPOINT
- ENV

The **CMD** keyword is used to run commands on the start of the container. These commands run only when there is no argument specified while running the container

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD .  /var/www/html
CMD apachectl –D FOREGROUND
```

Dockerfile

# Various Commands in Dockerfile

- FROM
- ADD
- RUN
- CMD
- **ENTRYPOINT**
- ENV

The **ENTRYPOINT** keyword is used strictly run commands the moment the container initializes. The difference between CMD and ENTRYPOINT is, ENTRYPOINT will run irrespective of the fact whether argument is specified or not

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD .  /var/www/html
ENTRYPOINT apachectl –D FOREGROUND
```

Dockerfile

# Various Commands in Dockerfile

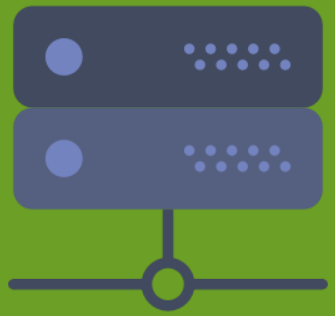
- FROM
- ADD
- RUN
- CMD
- ENTRYPOINT
- ENV

The ENV keyword is used to define environment variables in the container run-time.

Example

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install apache2
ADD .  /var/www/html
ENTRYPOINT apachectl –D FOREGROUND
ENV name Devops Intellipaat
```

Dockerfile

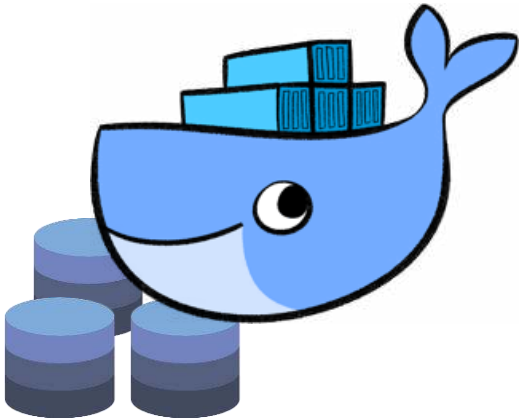


Docker Volumes

# Introduction to Docker Volumes

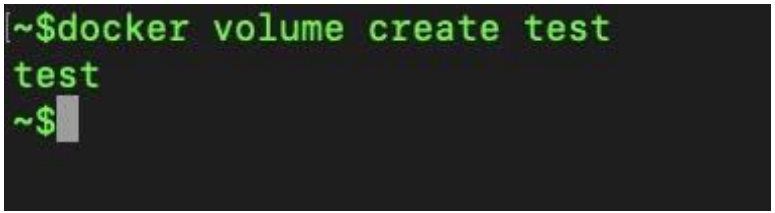
# Introduction to Docker Volumes

Docker Volumes are used to persist data across the lifetime of a container.



# Creating a Docker Volume

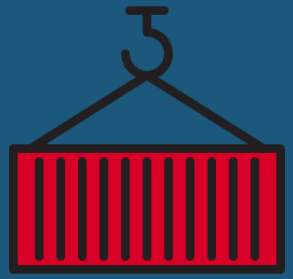
```
docker volume create <name-of-volume>
```



# Attaching it to a Container

```
docker run -it --mount source=<name-of-volume>,target=<path-to-directory> -d <image-name>
```

```
~$docker run -it --volume source=test,target=/app -d ubuntu
2bc10d68532ea5c186d78197faebd53fd9aa155a2399157d343940d580c88036
~$
```

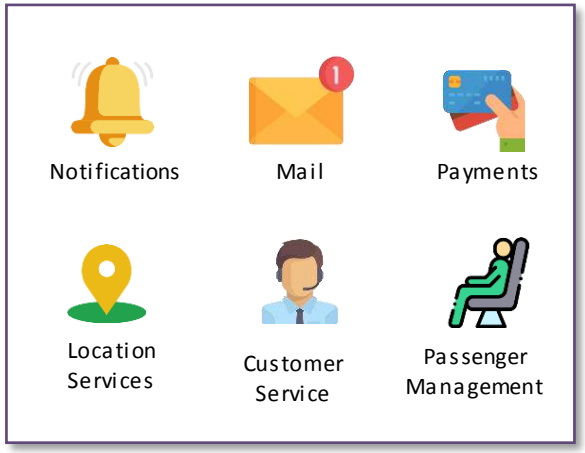


# Breaking the Monolith using Docker

## Microservices

# What is a Monolithic Application?

A **Monolithic** application is a single-tiered software application in which different components are combined into a single program which resides in a single platform.



# Disadvantages of a Monolithic Application

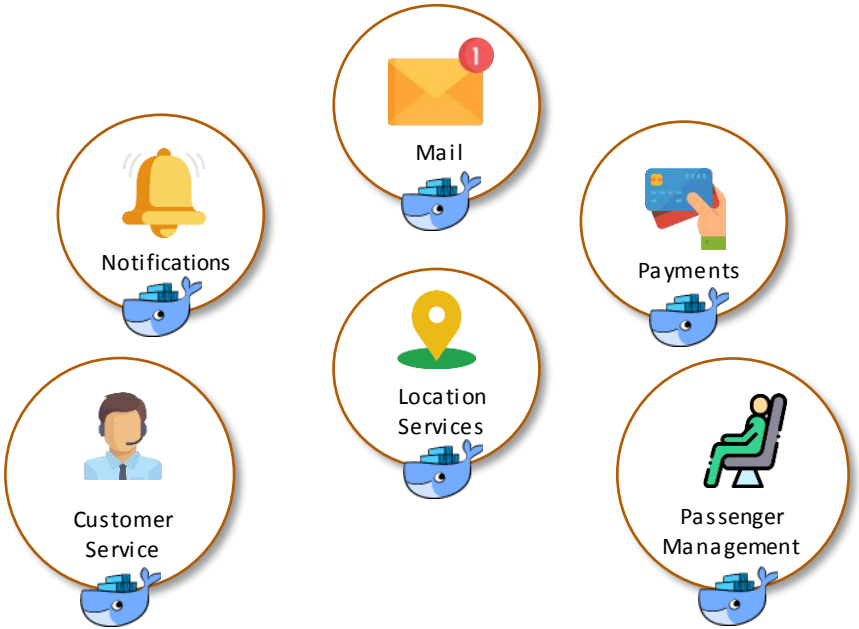
- Application is large and complex to understand
- Entire Application has to re-deployed on an application update
- Bug in any module, can bring down entire application
- Has a barrier to adopting new technologies

# What are Microservices?

Microservices are a software development architectural style that structures an application as a collection of loosely coupled services.

# What are Microservices?

Microservices are a software development architectural style that structures an application as a collection of loosely coupled services.



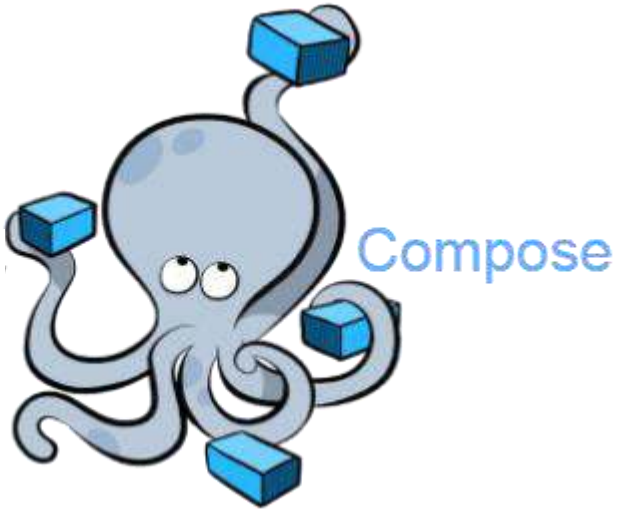
# Advantages of Microservices

- Application is distributed, hence easy to understand
- The code of only the Microservice which is supposed to be updated is changed
- Bug in one service, does not affect other services
- No barrier to any specific technology

# What is Docker Compose?

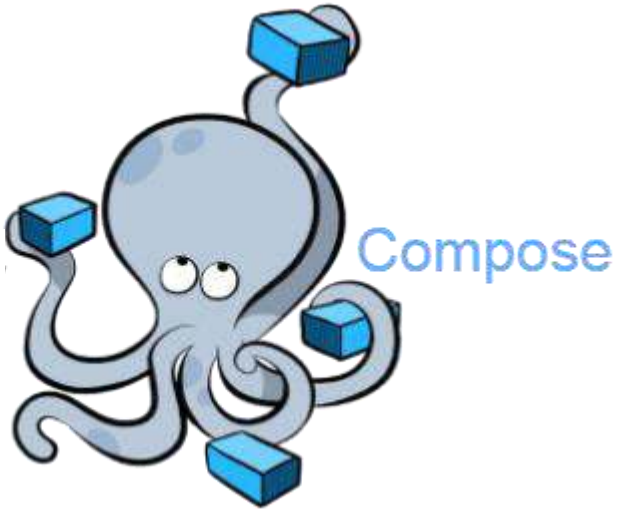
# What is Docker Compose?

**Compose** is a tool for defining and running multi-container **Docker** applications. With **Compose**, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. Run **docker-compose** up and **compose** starts and runs your entire app.



# What is Docker Compose?

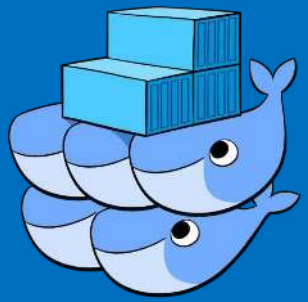
**Compose** is a tool for defining and running multi-container **Docker** applications. With **Compose**, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. Run **docker-compose** up and **compose** starts and runs your entire app.



# Sample Docker Compose File

```
version: '3.3'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
volumes:
  db_data:
```

**docker-compose.yaml**

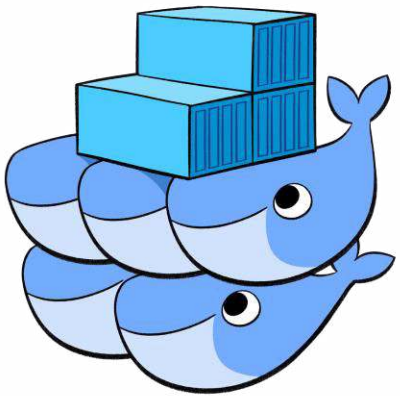


Container Orchestration

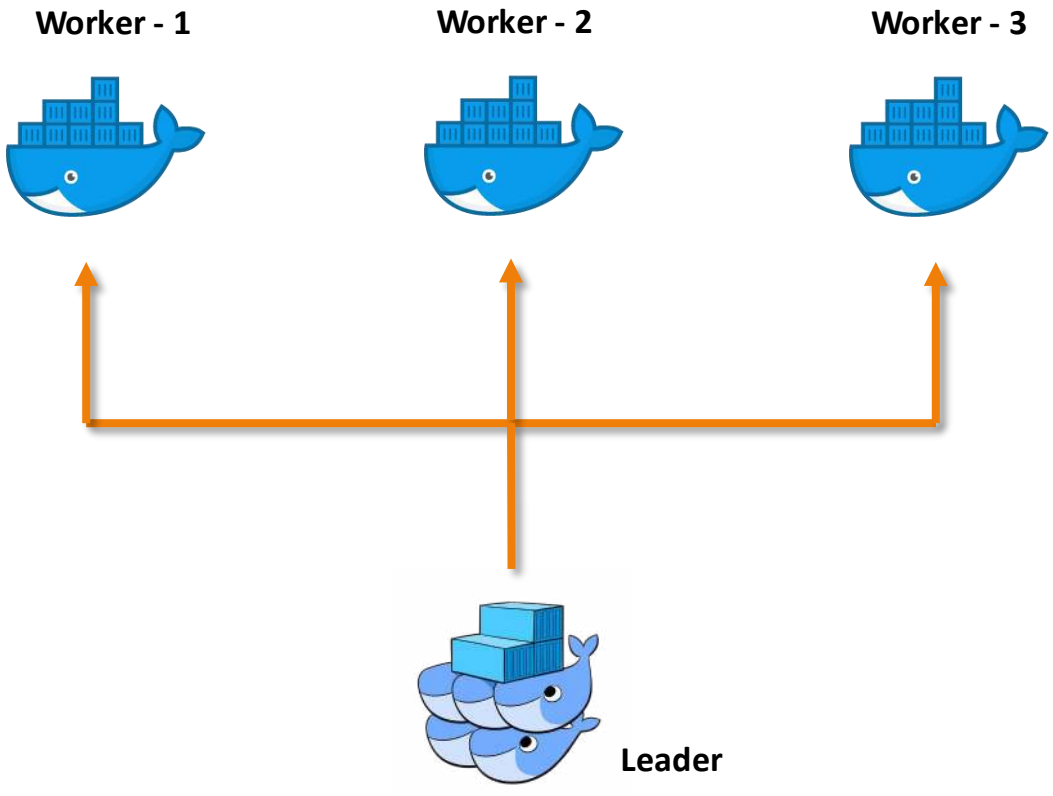
# What is Docker Swarm?

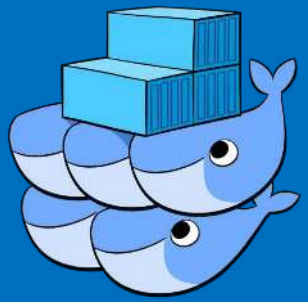
# What is Docker Swarm?

**Docker Swarm** is a clustering and scheduling tool for **Docker** containers. With **Swarm**, IT administrators and developers can establish and manage a cluster of **Docker** nodes as a single virtual system.



# What is Docker Swarm?





Container Orchestration

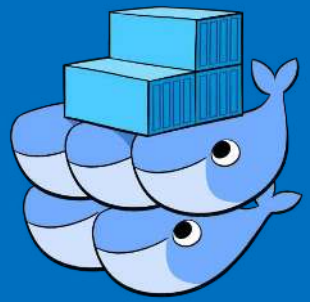
# Creating a Docker Swarm Cluster

# Creating a Docker Swarm Cluster

```
docker swarm init --advertise-addr=<ip-address-of-leader>
```

```
ubuntu@ip-172-31-26-120:~/wordpress$ docker swarm init --advertise-addr=172.31.2
Swarm initialized: current node (ptde8fg2vbxp8py931vrxdbpp) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2m8bntbbysh354anwigivubiqwf21kq6xkww4kjnq
.26.120:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow

ubuntu@ip-172-31-26-120:~/wordpress$
```

This command should be passed on the worker node, to join the docker swarm cluster

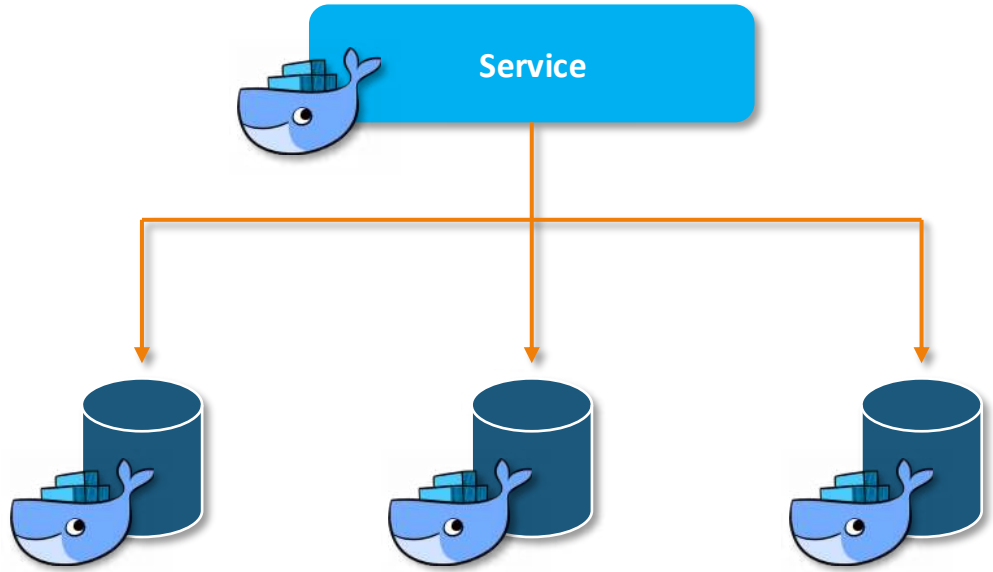


Container Orchestration

# Deploying an app on Docker Swarm

# What is a Service?

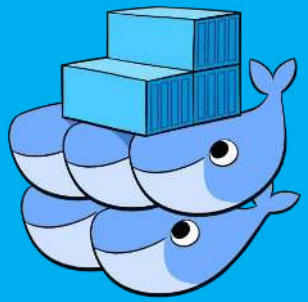
Containers on the cluster are deployed using **services** on Docker Swarm. A **service** is a long-running **Docker** container that can be deployed to any node worker.



# Creating a Service

```
docker service create -name <name-of-service> --replicas <number-of-replicas> <image-name>
```

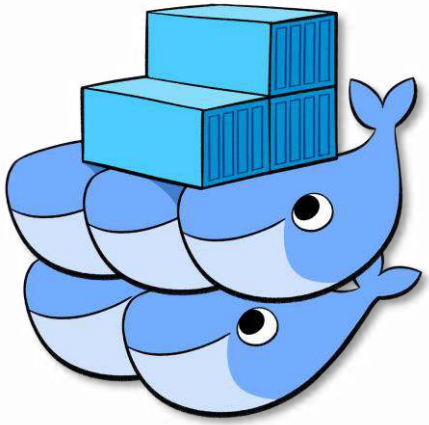
```
ubuntu@ip-172-31-26-120:~$ docker service create --name apache --replicas 3 -p 80:80 hshar/webapp
osftoz95rma0dkbsganqk0f3o
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-26-120:~$
```



# Deploying a Multi Tier App on Docker Swarm

# Deploying a Multi Tier app on Docker Swarm

1. Deploy a MySQL service on the Cluster
2. Deploy a website service on this Cluster
3. The MySQL service should not be exposed on the system
4. This will require an Overlay Network. Create an Overlay network
5. Verify if website service can interact with the database service



India : +91-7847955955

US : 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24X7 Chat with our Course Advisor