Walkthrough

INTRODUCTION TO DOCKER

Presented by- Soumya Pralhad

What is Docker?

Docker is a software platform that allows you to build, test, and deploy applications quickly, **packaging software** into standardized units called **containers**. It is a containerization platform.

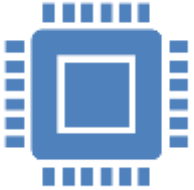
History Before Containerization

- **Virtualization** means machines run multiple applications on the same physical hardware
- But virtualization had few drawbacks

Problems of Virtualization

virtual machines were bulky in size

running multiple virtual machines lead to unstable performance



boot up process would usually take a long time and VM's would not solve the problems like portability, software updates, or continuous integration and continuous delivery

Solution: Containers

- Run many apps in the same virtual machine
- These apps share the OS and its overhead
- But these apps can’t interfere with each other
- Can’t access each other’s resources without explicit permission
- Multiple containers run on one operating system on a virtual/physical machine
- All containers share the operating system
- Containers are isolated
- Own file system/data, own networking

What is a container?

- Container ≠ VM
- Isolated
- Share OS
- and sometimes bins/libs

Containers Advantages

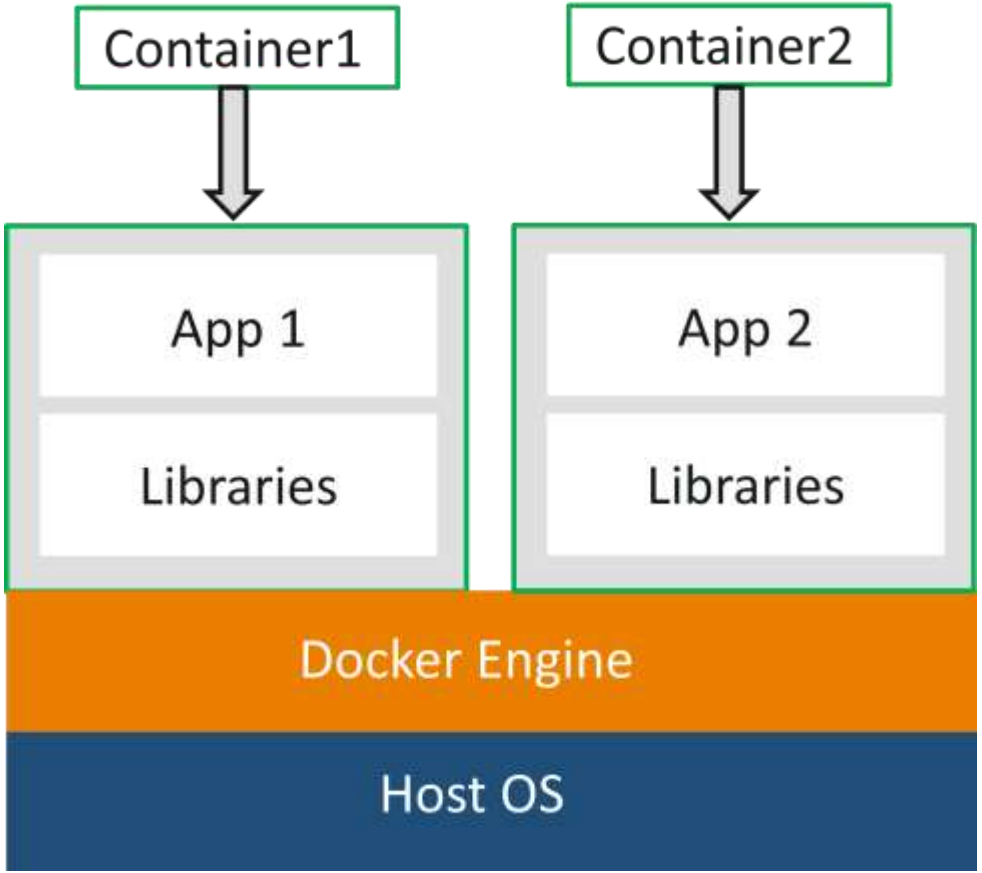
Containers have all the good properties of VMs

- Come complete with all files and data that you need to run
- Multiple copies can be run on the same machine or different machine
- Scalable
- Same image can run on a personal machine, in a data center or in a cloud
- Operating system resources can be restricted or unrestricted as designed at container build time
- Isolation: For example, “Show Process” (ps on Linux) command in a container will show only the processes in the container
- Can be stopped.
- Saved and moved to another machine or for later run

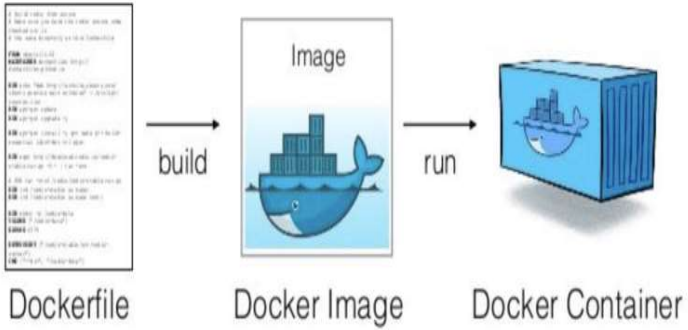
Docker

- Provides the isolation among containers
- Helps them share the OS
- Manage containers
- Developed initially by Docker.com
  Downloadable for Linux, Windows, and Mac from Docker.com
- Customizable with replacement modules from others

Isolation


Container1
Container2
App 1
Libraries
App 2
Libraries
Docker Engine
Host OS

Dockerfile, Images and Containers



- Dockerfile is built, it becomes a Docker Image and when we run the Docker Image then it finally becomes a Docker Container

Layers

- Each image has many layers Image is built layer by layer
- Layers in an image can be inspected by Docker commands
- Each layer has its own 256-bit hash
- Layers can be shared among many containers

Dockerfile

- A Dockerfile is a text document which contains all the commands that a user can call on the command line to assemble an image.
- So, Docker can build images automatically by reading the instructions from a Dockerfile.
- You can use docker build to create an automated build to execute several command-line instructions in succession.

Dockerfile

It is possible to build your own images reading instructions from a `Dockerfile`

```
FROM centos:7
RUN yum install -y python-devel python-virtualenv
RUN virtualenv /opt/indico/venv
COPY entrypoint.sh /opt/indico/entrypoint.sh
EXPOSE 8000
ENTRYPOINT /opt/indico/entrypoint.sh
```

Dockerfile commands

FROM

ENV

WORKDIR

ENTRYPOINT

CMD

COPY

ADD

RUN

EXPOSE

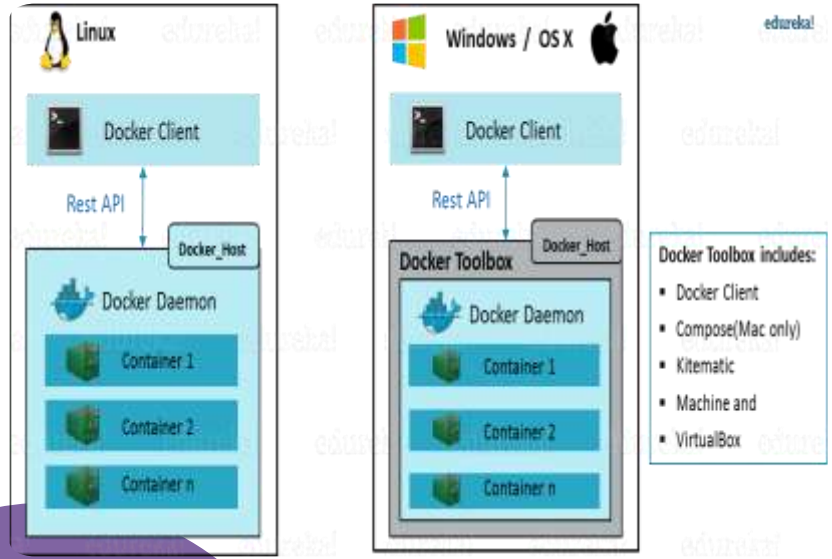
Docker Images

- Docker Image can be compared to a template which is used to create Docker Containers.
- So, these read-only templates are the building blocks of a Container.
- You can use docker run to run the image and create a container.
- Docker Images are stored in the Docker Registry. It can be either a user's local repository or a public repository like a Docker Hub which allows multiple users to collaborate in building an application.

Docker Containers

- It is a running instance of a Docker Image as they hold the entire package needed to run the application.
- So, these are basically the ready applications created from Docker Images which is the ultimate utility of Docker.
- Layered file system

Docker Architecture



- client server architecture
- REST API is used for communication between the CLI client and Docker Daemon
- As per the above image, in a Linux Operating system, there is a client which can be accessed from the terminal and a Host which runs the Daemon.
- We build our images and run containers by passing commands from the CLI client to the Daemon.

Image Registries

- Containers are built from images and can be saves as images
- Images are stored in registries
- Local registry on the same host
- Docker Hub Registry: Globally shared
- Private registry on Docker.com
- Any component not found in the local registry is downloaded from specified location
- Official Docker Registry: Images vetted by Docker
- Each image has several tags, e.g., v2, latest, ...

Docker registry

https://hub.docker.com/

Docker commands

docker pull: Pull the image from local/image registry
docker container run: Run the specified image
docker ps: list running containers
docker ps –a: running and exited containers
docker images: list the images
docker build: build image from dockerfile
docker container exec: run a new process inside a container
docker container stop: Stop a container
docker container start: Start a stopped container
docker container rm: Delete a stopped container
docker container inspect: Show information about a container

docker-compose

Allows to run multi-container Docker applications reading instructions from a `docker-compose.yml` file

```yaml
version: "2"
services:
  my-application:
    build: ./
    ports:
      - "8000:8000"
    environment:
      - CONFIG_FILE
  db:
    image: postgres
  redis:
    image: redis
    command: redis-server --save "" --appendonly no
    ports:
      - "6379"
```

Docker compose commands

docker-compose build command is used to build/ rebuild the services

docker-compose up command is used to create/ start the containers

Thank You!