

# Applications Containerization using Docker

Ranjit Karni

VPark Innovations

[Ranjit.balu@gmail.com](mailto:Ranjit.balu@gmail.com)

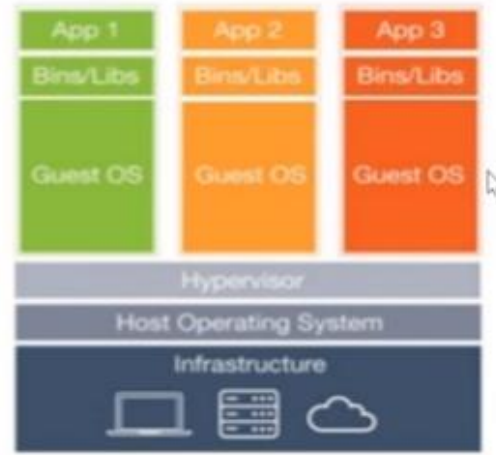
Ph.No: 9676976662

# Virtualization vs Containerization

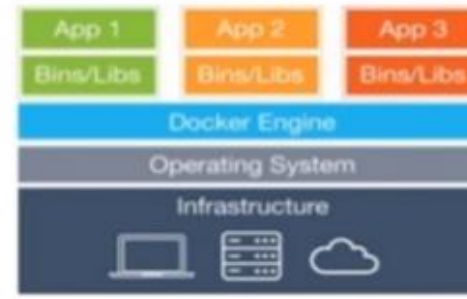
---

- Containers are lightweight because they don't need the extra load of a hypervisor.
- Containers run directly within the host machine's kernel.
- You can run more containers on a given hardware combination than if you were using virtual machines.
- You can even run Docker containers within host machines that are actually virtual machines.

## Containers vs. VMs



Virtual Machines



Containers



# What is Docker

---

- Docker is an open source platform for developing, shipping and running applications by using containers.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- The Docker platform uses the docker **Engine** to quickly build and package apps as Docker **images** created using files written in the **Dockerfile** format that then is deployed and run in a layered **container**.



# Benefits of Docker

---

**Resource Efficiency:** Docker is lightweight and fast. Process level isolation and usage of the container host's kernel is more efficient when compared to virtualizing an entire hardware server using VM.

**Fast and consistent delivery of your applications:** By taking advantages of Docker's methodologies for shipping, testing and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

**Continuous Deployment and Testing:** The ability to have consistent environments and flexibility with patching has made Docker a great choice for teams that want to move from waterfall to the modern Devops approach to software delivery.



# Example Scenarios

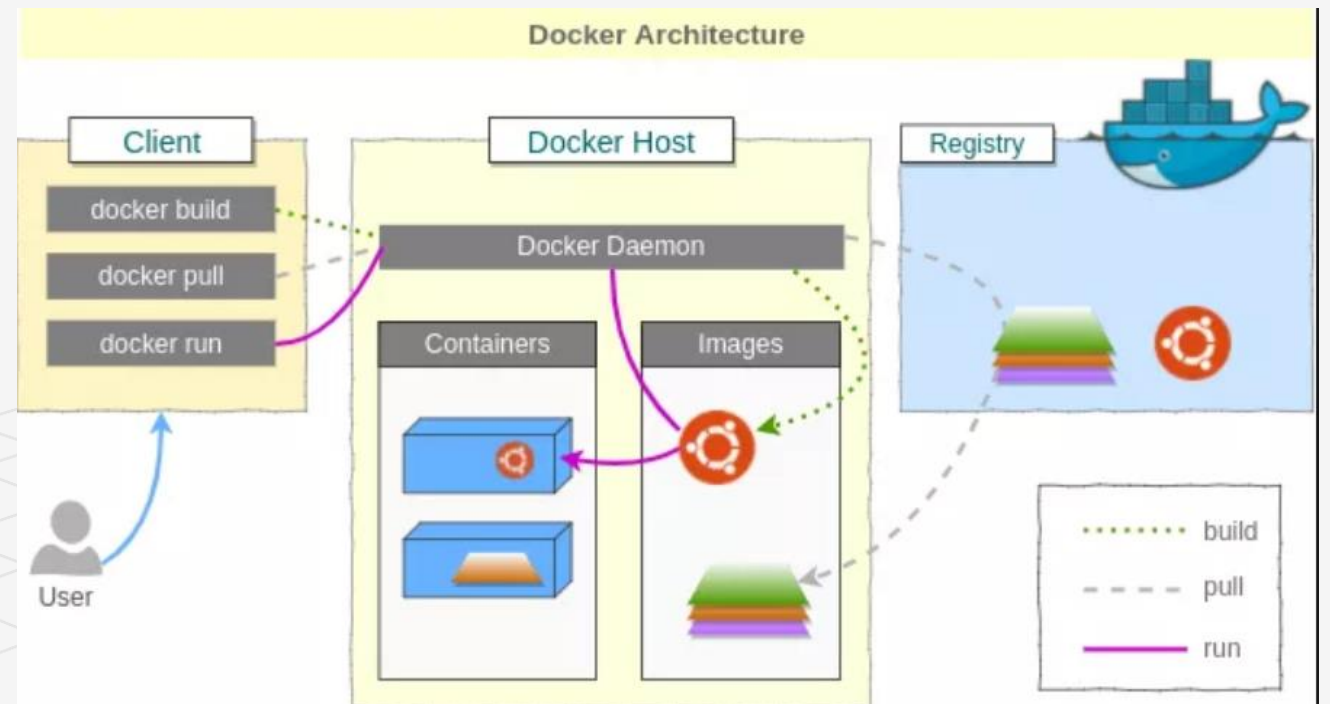
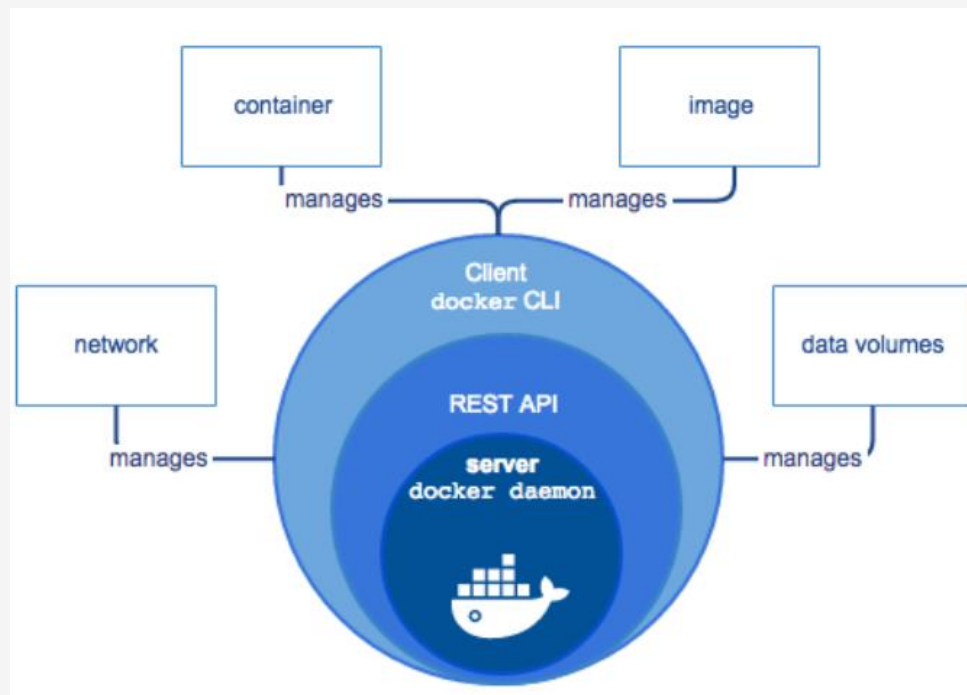
---

1. Your developers write code locally and share their work with their colleagues using Docker containers.
2. They use Docker to push their applications into a test environment and execute automated and manual tests.
3. When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation
4. When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.



# Docker Architecture

- Docker Engine is a client-server application with these major components:
  - A Server which is a type of long-running program called a **Daemon** process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client (the docker command)

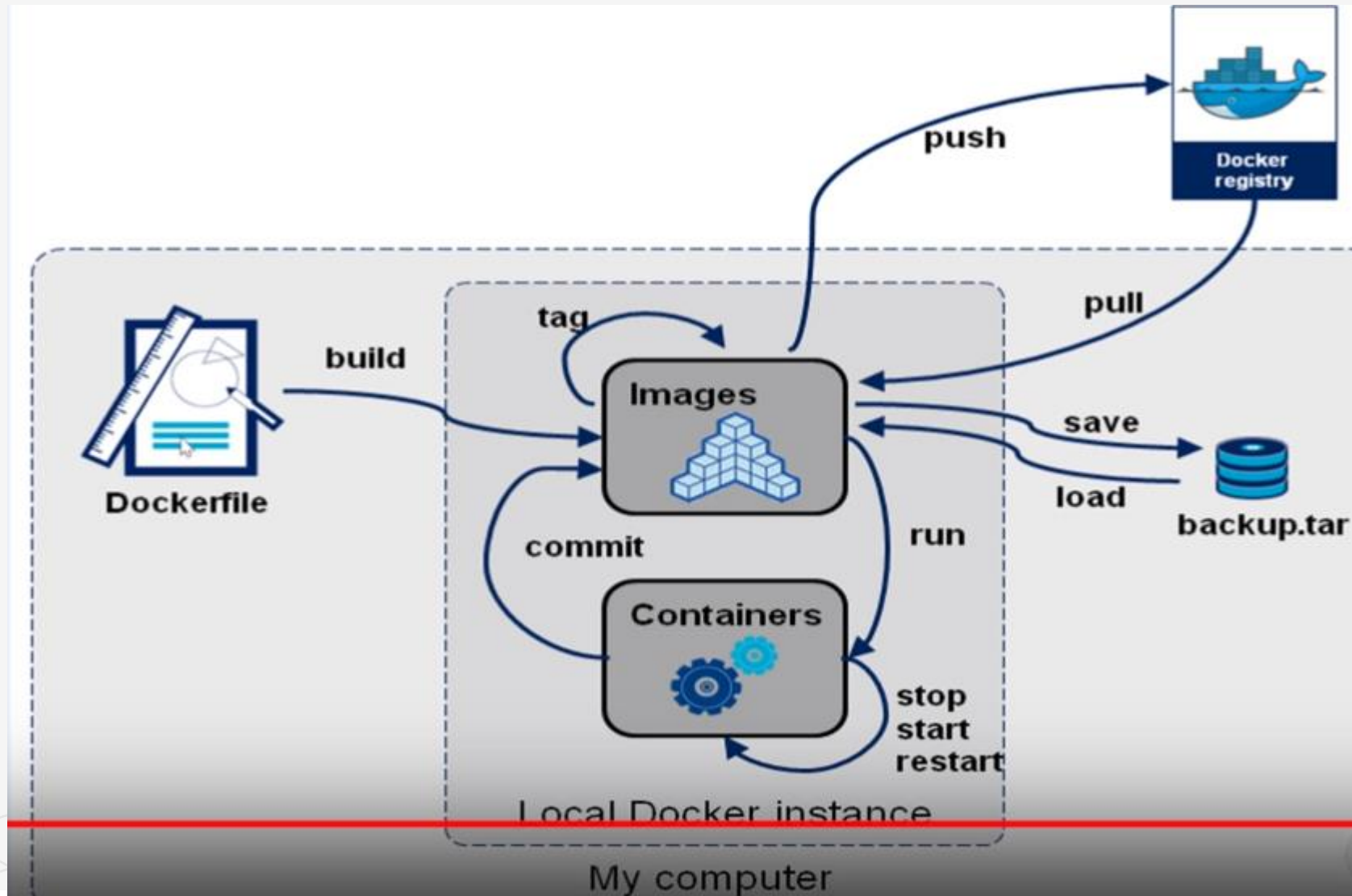


# Docker Taxonomy

---

- **Image:** The one which is going to contain the primary executable of your application along with the dependencies, configuration and also the runtime required. For Example, if you want to have a website, all the files which will make a website, along with the web server which is responsible for execution of it and along with the framework which is responsible for execution. If you want a database, even for that also, you can create an image. Docker images are not platform independent and they are always specific to a given OS they are built.
- **Container:** As a developer, you can think an image as Class and Container as Object. The image when instantiated is called as a container.
- **Repository:** Every Docker image will have certain name and that is going to create a repository. A repository is a collection of similar images that may be differentiated with the different versions or different operating systems.
- **Tags:** Every image may be associated with a tag.
- **Dockerfile:** For building an image, we need set of instructions and those instructions will be provided in Dockerfile.
- **Build:** Build is the process of creating an image from the actual source available. The process of creating an image is called Build.
- **Networking:** Docker has its own networking. Because every container is independent and isolated and how will they communicate each other. There need to be an underlying network. Docker platform has its own virtual networking through which the two containers can communicate each other.
- **Storage Volumes:** The content can be shared across multiple container instances or persisted. Container is a single instance execution. So, if a container is used for storing any data, all the data will be lost as soon as the container is terminated. Container has its own file system, but it is temporary unless it is mapped to storage volume. Unless you use storage volumes and persist your data , all your data which the container writes to its disk will be lost.
- **Docker Registry:** Docker hub and Azure Container Registry are two popular registries where you can store your private docker images. Based on the credentials I provide, other can download the image and use it.
- **Docker Compose:** Applications going to use of multiple containers like one container for web application, one container for database, one container for API, one container for Redis Cache. It is complex to start all of them in proper sequence and passing data to all of them. This can be achieved easily using yaml file. Yaml file consists of declarative syntax for the instructions to be executed by docker compose

# Docker Container Life Cycle





# Docker Images for .NET Core

---

- .NET code is a general purpose development platform maintained by Microsoft and the .NET community on **GitHub**.
- It is a cross-platform, supporting Windows, macOS and Linux.
- It can be used in device, cloud and embedded /IOT scenarios.
- A complete en-to-end application can be developed using .NET core.
- When you want to build images with very high performance, very light weight even when compared to .NET framework
  - Microsoft/dotnet
  - Microsoft/dotnet:2.1-sdk
  - Microsoft/dotnet-2.1-runtime
  - Microsoft/dotnet-2.1-deps

# Docker CLI Commands

---

## **Images CLI Command**

Docker images -a

Docker images ls -a

Docker rmi d62ae1319d0a

Docker prune

## **Container CLI Command**

Docker run -it -rm Microsoft/nanoserver cmd

Docker container run -d -p 8080:80 --name iis Microsoft/iis

Docker inspect iis

Docker ps -a

Docker container ls -a

Docker container rm d62ae1319d0a

Docker container prune

# Docker CLI Commands

---

Docker --version

Docker info

Docker search debian

Docker pull debian

Docker pull Ubuntu:14.04

docker run -it -p 9000:80 --name=debian\_container1 debian

Docker images

Docker ps

Docker ps -a

docker run -it -d -p 9001:80 --name=web\_container centos:7

docker attach web\_container

docker stop db\_container1

Docker stats

docker stats --no-stream

# Docker CLI Commands

---

```
docker export web_container -o web_container.tar
```

```
docker import web_container.tar
```

```
docker logs {container_name_or_container_id}
```

```
docker top {Container_Name_OR_ID}
```

```
docker inspect web_container | grep IPAddress | cut -d '"' -f 4
```

```
docker inspect -f "{{.NetworkSettings.IPAddress}}" web_container
```

# DockerFile

---

## FROM

Usage:

- FROM <image>
- FROM <image>:<tag>
- FROM <image>@<digest>

Information:

- FROM must be the first non-comment instruction in the Dockerfile.
- FROM can appear multiple times within a single Dockerfile in order to create multiple images. Simply make a note of the last image ID output by the commit before each new FROMcommand.
- The tag or digest values are optional. If you omit either of them, the builder assumes a latest by default. The builder returns an error if it cannot match the tag value.

## MAINTAINER

Usage:

- MAINTAINER <name>

The MAINTAINER instruction allows you to set the Author field of the generated images.

## RUN

Usage:

- RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows)
- RUN ["<executable>", "<param1>", "<param2>"] (exec form)

Information:

- The exec form makes it possible to avoid shell string munging, and to RUN commands using a base image that does not contain the specified shell executable.
- The default shell for the shell form can be changed using the SHELL command.
- Normal shell processing does not occur when using the exec form. For example, RUN ["echo", "\$HOME"] will not do variable substitution on \$HOME.

# DockerFile

---

## **LABEL**

Usage:

- LABEL <key>=<value> [<key>=<value> ...]

Information:

- The LABEL instruction adds metadata to an image.
- To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing.
- Labels are additive including LABELs in FROM images.
- If Docker encounters a label/key that already exists, the new value overrides any previous labels with identical keys.
- To view an image's labels, use the docker inspect command. They will be under the "Labels" JSON attribute.

## **EXPOSE**

Usage:

- EXPOSE <port> [<port> ...]

Information:

- Informs Docker that the container listens on the specified network port(s) at runtime.
- EXPOSE does not make the ports of the container accessible to the host.

## **ENV**

Usage:

- ENV <key> <value>
- ENV <key>=<value> [<key>=<value> ...]


Information:

- The ENV instruction sets the environment variable <key> to the value <value>.
- The value will be in the environment of all "descendant" Dockerfile commands and can be replaced inline as well.
- The environment variables set using ENV will persist when a container is run from the resulting image.
- The first form will set a single variable to a value with the entire string after the first space being treated as the <value> - including characters such as spaces and quotes.

# Example

---

FROM ubuntu:15.04  
COPY . /app  
RUN make /app  
CMD python /app/app.py



# Questions?

Ranjit Karni

Vpark Innovations

[Ranjit.balu@gmail.com](mailto:Ranjit.balu@gmail.com)

Ph No: +91-9676976662