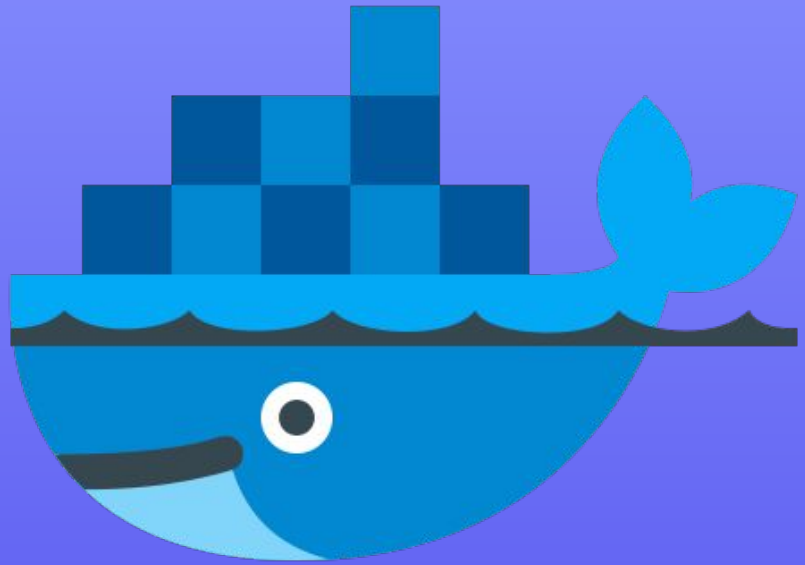
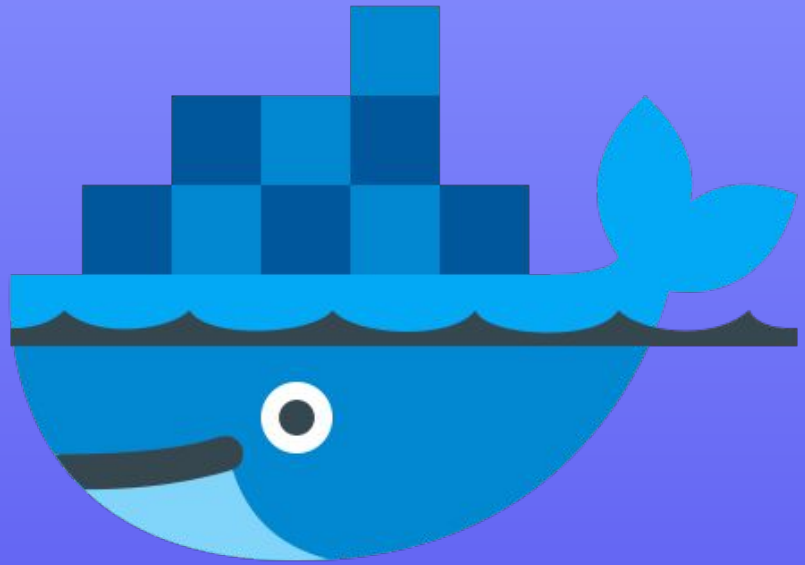


DIVING INTO DOCKER

meet.google.com

June 9 2020, 8 PM

GCES IT CLUB



ARJUN ADHIKARI

/thearjun
/thearjun
thearjun.tech

TODAY'S AGENDA

INTRO TO DOCKER

Basic intro to docker and containers.

INSTALLATION

Installation and configuration guide for Docker

COMMANDS

Basic docker commands for terminal.

DOCKERFILE

How to write a basic Dockerfile ?

INTRODUCTION TO DOCKER

an open-source project that automates the **deployment** of software applications inside **containers** by providing an additional layer of **abstraction** and **automation** of OS-level **virtualization** on Linux.

INTRODUCTION TO DOCKER

In simple words ,

Docker is a tool for running your applications inside containers.

Containers package all the dependencies and code your app needs to run into a single file, which will run the same way on any machine

DOCKER CONTAINER | CONTAINER IMAGE | HOW IT WORK ?

DOCKER CONTAINER

A container is a standard unit of software that **packages up code and all its dependencies** so the application runs quickly and reliably from one computing environment to another.

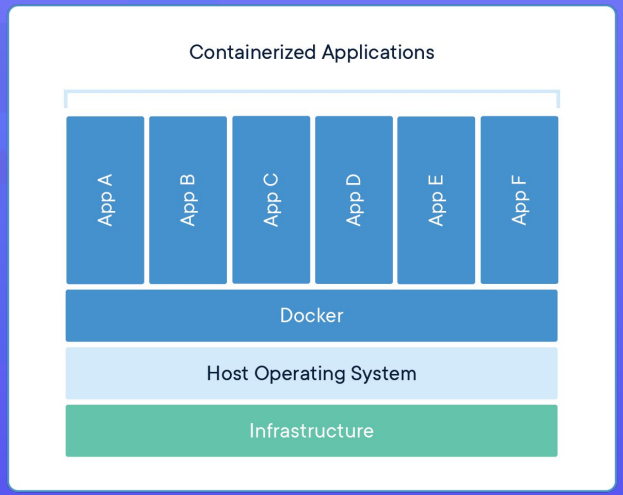
CONTAINER IMAGE

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

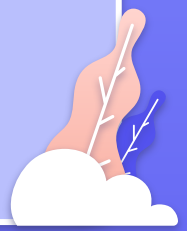
HOW CONTAINERS WORK ?

by leveraging the low-level mechanics of the host operating system, containers provide most of the isolation of virtual machines at a **fraction of the computing power**.

HOW CONTAINERS WORK ?



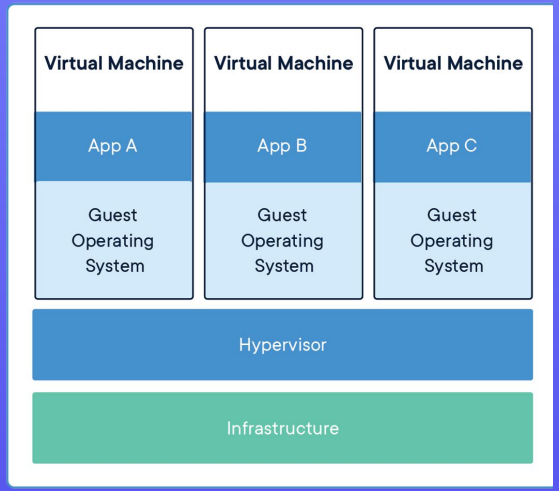
IS IT SAME AS VIRTUAL MACHINES ?



VIRTUAL MACHINES

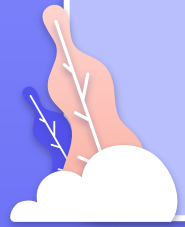
VMs are great at providing full process isolation for applications:
there are very few ways a problem in the host operating system can affect the software running in the guest operating system, and vice-versa.

HOW VIRTUAL MACHINES WORK ?

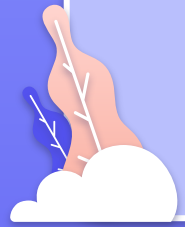


CONS OF VIRTUAL MACHINES

But this isolation comes at great cost – the computational overhead spent virtualizing hardware for a guest OS to use is substantial.



IT’S AWESOME.



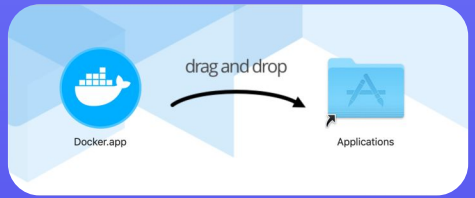
LET'S USE IT.

DOCKER : INSTALLATION

LINUX

sudo apt-get install docker

MAC



WINDOWS

https://docs.docker.com/docker-for-windows/install/

IMAGE & CONTAINER

CONTAINER

An instance of an image is called a container.

IMAGE

You have an image, which is a set of layers as you describe. If you start a image, you have a running container of that image.

IMAGE & CONTAINER

CONTAINER

- **Isolated** application platform.
- **Containers** need to run your application based on images.

IMAGE

- **Read** only templates used to create containers.
- **Build** by you or other docker users.
- **Stored** in the docker hub or your local registry.

DOCKER DAEMON

The background service running on the host that manages building, running and distributing Docker containers.

The daemon is the process that runs in the operating system which clients talk to.

DOCKER CLIENT

The command line tool that allows the user to interact with the daemon.

More generally, there can be other forms of clients too - such as **Kitematic** which provide a GUI to the users.

DOCKER HUB

A registry of Docker images.

You can think of the registry as a directory of all available Docker images.

If required, one can host their own Docker registries and can use them for pulling images.

DOCKER : HELLO WORLD

```
$ docker run hello-world
```

DOCKER : HELLO WORLD

```
$ docker run hello-world
```

If the image of then name <CUSTOM_NAME> is not available on your system, it pulls the image from Docker Hub.

By default it pulls the image of latest version from Docker Hub.

pull command pulls the image, and **run** command does (pull + execute).

DOCKER : RUNNING BUSYBOX

```
$ docker run -it busybox
```

DOCKER : RUNNING BUSYBOX

```
$ docker run -it busybox
```

BusyBox is a software suite that provides several Unix utilities in a single executable file.

Simply here, we call it a “small Linux”.

-i provides interactive STDIN mode.
-t provides pseudo terminal.
Remember it, we will be using -it many times throughout our **Docker**ous journey.

DOCKER : RUNNING BUSYBOX

```
$ docker run busybox echo "hello from busybox"
```

In this case, the Docker client dutifully runs the echo command in our busybox container and then exits it.

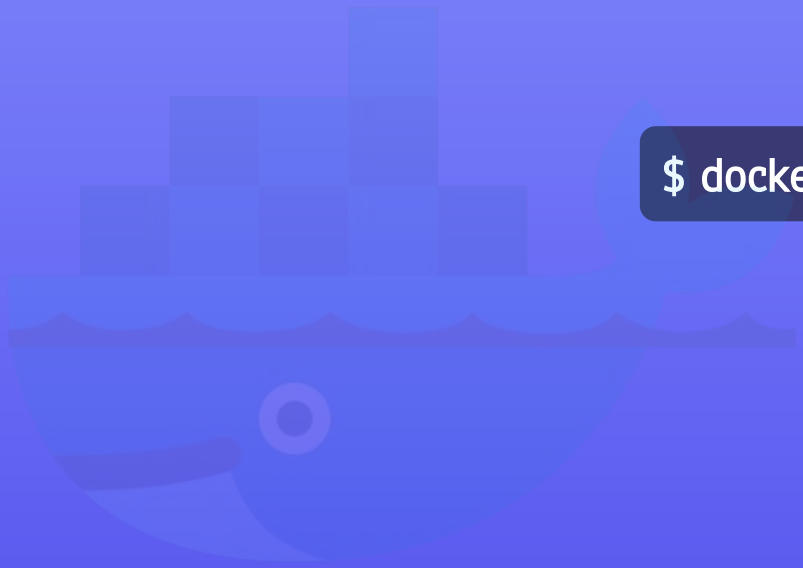
LET'S EXECUTE IT ▶

DOCKER : RUNNING BUSYBOX

```
$ docker run busybox echo "hello from busybox"
```

If you've noticed, all of that happened pretty quickly. Imagine booting up a virtual machine, running a command and then killing it. Now you know why they say containers are fast!

DOCKER : LIST CONTAINERS



```
$ docker ps
```

Lists only the running containers on the system.

DOCKER : LIST CONTAINERS

```
$ docker ps -aq
```

Lists all containers (Only IDs).

DOCKER : LIST CONTAINERS

```
$ docker ps -a
```

Lists all the containers on the system.

DOCKER : STOP RUNNING CONTAINERS

```
docker stop $(docker ps -aq)
```

Stop all running containers.
Although we can kill all the running images forcefully with **kill** command, but it's not recommended.

DOCKER : REMOVE CONTAINER

```
$ docker rm <CONTAINER_ID>
```

Remove container of specific ID in the system.
We can get ID through **$ docker ps -a**

DOCKER : REMOVE ALL CONTAINERS

```
docker rm $(docker ps -aq)
```

Remove all containers in the system.

DOCKER : REMOVE ALL IMAGES

```
docker rmi $(docker images -q)
```

Remove all the images in the system.
NOTE : Remove containers first before removing the images.

DOCKERFILE

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

LET'S CREATE IT ▶

DOCKERFILE COMMAND : FROM

FROM UBUNTU:18.04

The first part is the FROM command, which tells us what image to base this off of.
In this instance, it's using the ubuntu:18.04 Docker image, which again references a Dockerfile to automate the build process.

DOCKERFILE COMMAND : RUN

RUN <COMMAND>

The next set of calls are the RUN commands.
This is what runs within the container at build time.

In this command, package information from all configured sources are downloaded inside Docker container.

DOCKERFILE COMMAND : RUN

RUN ["<executable>", "<param1>", "<param2>"]

(shell form, the command is run in a shell,
which by default is
/bin/sh -c on Linux
or
cmd /S /C on Windows)

DOCKERFILE COMMAND : CMD

```
CMD ["<executable>", "<param1>", "<param2>"]
```

The main purpose of a CMD is to provide defaults for an executing container.

DOCKERFILE COMMAND : ENTRYPOINT

ENTRYPOINT ["<executable>","<param1>","<param2>"]

- If the ENTRYPOINT isn't specified, Docker will use /bin/sh -c as the default.
- However, if you want to override some of the system defaults, you can specify your own entrypoint and therefore manipulate the environment.
- The ENTRYPOINT command also allows arguments to be set from the Docker run command as well, whereas CMD can't.

DOCKERFILE COMMANDS : WORKDIR

WORKDIR </path/to/workdir>

- Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it.
- II can be used multiple times in the one Dockerfile.
- If a relative path is provided, it will be relative to the path of the previous WORKDIR instruction.

DOCKERFILE COMMANDS : MAINTAINER

MAINTAINER Arjun

Allows you to set the Author field of the generated images.

DOCKERFILE COMMAND : LABEL

```
LABEL maintainer="Arjun"
```

Allows you to set metadata for the image.

DOCKERFILE COMMANDS : EXPOSE

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

- **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.

DOCKERFILE COMMAND : ENV

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

The ENV instruction sets the environment variable <key> to the value <value>.

DOCKERFILE COMMAND : COPY

COPY <src> [<src> ...] <dest>
COPY ["<src>", ... "<dest>"] (for paths containing whitespace)

DOCKERFILE COMMAND : COPY

- Copies new files or directories from <src> and adds them to the filesystem of the image at the path <dest>.
- <src> must be relative to the source directory that is being built (the context of the build).
- <dest> is an absolute path, or a path relative to WORKDIR.

DOCKERFILE EXECUTION IN ORDER

STEP BY STEP

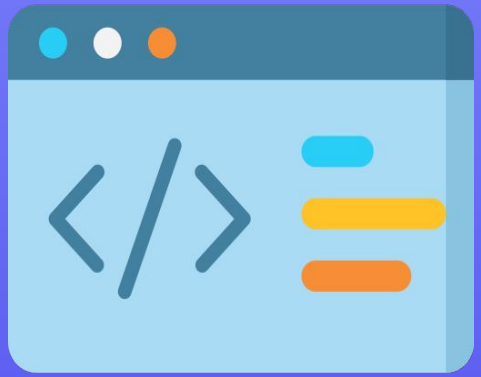
01 Getting the base image.

02 Installing the programs, libraries and dependencies.

03 Configuring the system ready for execution. (Not always done)

04 Execution of instructions.

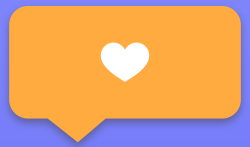
DOCKERFILE : PRACTICAL SESSION



1. Hello World

2. Lint code from Github Repository

3. Serve a static site



“ If it works during development, then it should work on production too.

No more ‘It worked on my system’ stuff. ”

— UNKNOWN

THANKS!

Do you have any questions?

Thanks to GCES IT CLUB and attendees for making this session successful.

IT'S NOT HARD TO REACH ANYONE IN DIGITAL WORLD

mailarjunadhikari@gmail.com
fb.com/thearjun.io