**DOCKER**

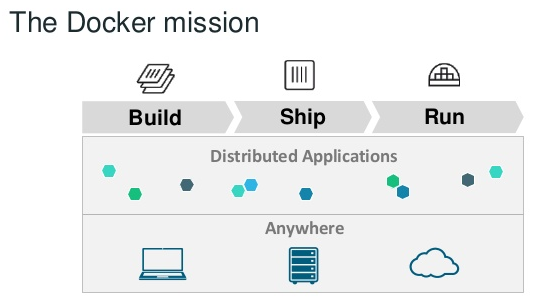
**Docker**

=======

Docker is a container management service.

The keywords of Docker are **develop, ship** and **run** anywhere.

The whole idea of Docker is for developers to **easily develop applications**, **ship them into containers** which can then be **deployed anywhere**.



Release of Docker was in March 2013 and since then, it has become the buzzword for modern world development.

**Features of Docker**

**=================**

* Docker has the ability to **reduce the size of development** by providing a **smaller footprint of the operating system via containers**.
* With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
* You can **deploy Docker containers anywhere**, on any physical and virtual machines and even on the cloud.
* Since Docker containers are pretty lightweight, they are easily scalable.

**Why Virtualization ??**

==================

* Hardware Utilization
* To reduce no of physical servers
* Reduce Cost
* More different OS

Your whole design of virtualization, is to target the Applications.  
  
We are focusing more on Hardware, Virtualization and OS.  
But no one is focusing on Application side,

On Application side we need two fundamental characteristics:

Data Isolation & Data Protection  
  
Let say we are having 3 VM’s and minimum requirements for this system are:

1 CPU & 1 GB RAM now like this i need 3 CPU and 3 GB RAM  
 If the same things is needed for like 1000+ machines, it becomes more cumbersome.

So **Docker** took advantage of this, by using **CONTAINERIZATION**.

Using Docker we can build up entire application with OS.

Sometimes it happens like this application works fine on Linux OS, but doesn’t work on Unix and Win, these kind of problems can be avoided by using Docker

Using docker we can create entire application with OS itslef(OS dependent files).

Docker uses special file system,

  Layered File system[COW - Copy On Write]

  Box → VM's

  AMI     → Instances

  Images → Containers

Three important things to check in docker:

Docker Container, Docker Images & Docker Registry.

Docker Container: is a running instance of an OS image.

      Run time object

**Installation**

**============**

**# get.docker.com**

  # sudo usermod -aG docker <user-name>

  Docker comes in two components SERVER & CLIENT

  # **systemctl start docker**

  # sudo docker <options>

  First thing we need to have is images, from those images will create containers.

  # rpm -qa | grep docker

  # sudo systemctl status docker

  Let’s see do we have any images

  # sudo docker images

**So where do i get images from ??**

==========================

hub.docker.com {search for nexus, jenkins, tomcat and centos}

# docker pull centos {this is how we get image from internet}

# sudo docker images

{ unique image id and size is also very less coz its limited OS }

**# docker info**

# docker images

**I want to see how many containers are running ??**

# sudo docker ps

# open two sessions of the same instance {we can do things simultaneously}

# s1 - sudo docker ps {shows running containers}

# s2 - sudo docker run -it centos /bin/bash

{now we are inside container}

it - terminal interactive

# s1 - sudo docker ps {shows running containers}

This container is created from centos image

If we use container it will take at least 2 min but. here its 3 seconds

# s1 - top {so many tasks}

# s2 - top {literally 2 process}

# s2 - ps -ef {same 2 process}

# s2 - cat /etc/hosts {my hostname is container\_id}

s1 - sudo docker ps

# s1 - sudo docker ps

# s2 - exit {bash is finished - container is gone}

# s1 - sudo docker ps

# docker images

# docker run -it { attached mode runs in **foreground** }

# docker run -dt { detached mode runs in **background** }

# docker exec -it <container-id> bash

**Controlling the container**

**=====================**

**naming container**

**================**

s2 - # sudo docker run --rm -ti --name "web-server01" docker.io/centos /bin/bash

s1 - sudo docker ps

s2 - exit

**setting hostname**

**================**

s2 - sudo docker run --rm -ti --name "web-server1" --hostname "web-server" docker.io/centos /bin/bash

s2 - cat /etc/hostname

s2 - hostname

s2 - exit

s1 - sudo docker ps

Now let’s do something interesting

===========================

**# docker run -it --name demo1 ubuntu /bin/bash**

# apt-get update

My image here is ubuntu, am i downloading the image again ??

I’m using the same image to build another container,

so now my container is **ubuntu + apt-get update.**

**Everytime i work on an image, my container has an update.**

**Whenever u start working on an image, that is when u start building a container.**

Now i’m doing new thing

# apt-get install apache2

Now my container is  **demo + apt update + install apache2**

I’ll do exit

# exit

Now what is command to see the exit ed containers:

# docker ps -a

**Container lifetime and Persistent data  
================================  
Containers are usually immutable** and **ephemeral**, **just fancy buzzwords for unchanging and temporary or disposable**, but the idea here is that we can just throw away the container and create a new one from an image right!!!!.  
 **Containers are Ephemeral and once a container is removed, it is gone.**

What about scenarios where you want the applications running inside the container to write to some files/data and then ensure that the data is still present. For e.g. let’s say that you are running an application that is generating data and it creates files or writes to a database and so on. Now, even if the container is removed and in the future you launch another container, you would like that data to still be there.

In other words, the fundamental thing that we are trying to get over here is to separate out the container lifecycle from the data. Ideally we want to keep these separate so that the data generated is not destroyed or tied to the container lifecycle and can thus be reused. This is done via Volumes, which we shall see via several examples.

So we are not talking about actual limitation of containers, but more of design goal or best practise, this is the idea of immutable infrastructure

So docker has two solutions to this problem known as **Volumes** and **Bind mounts**.

**Working with volumes    
===================**There are three main use cases for Docker data volumes:

1. To keep data around when a container is removed
2. To share data between the host filesystem and the Docker container

By **Docker Volumes**, we are essentially going to look at how to **manage data within your Docker containers**.

**Few points about volumes  
======================**

* A data volume is a specially designed directory in the container.
* It is initialized when the container is created. By default, it is not deleted when the container is stopped. It is not even garbage collected when there is no container referencing the volume.
* The data volumes are independently updated. Data volumes can be shared across containers too.

**Mounting a Data volume  
====================**We are going to use the -v [/VolumeName] as an option to mount a volume for our container.  
  
 # docker run -it -v /data --name container1 ubuntu bash

In the container do # ls  
 Notice that a volume named **data** is visible now.  
  
Let us a do a cd inside the data volume and create a file named file1.txt  
  
# cd data  
# touch file1.txt

So what we have done so far is to mount a volume /data in the container.  
Now, let us exit the container by typing exit  
  
# exit  
# docker ps  
 # docker ps -a  
# docker inspect container1

# sudo ls var/lib/docker/volumes

You will see that it shows our **file1.txt** that we created.  
Now that the container is stopped i.e. exited, let us restart the container (**container1**) and see if our volume is still available and that **file1.txt** exists.

# docker start container1

# docker exec -it container1 bash

Now, let’s do an interesting thing. Exit the container and remove the container.

# docker rm container1

# sudo ls var/lib/docker/volumes  
  
This shows to you that though you have removed the **container1**, the data volume is still present on the host. This is a dangling or ghost volume and could remain there on your machine consuming space.  
  
Do remember to clean up if you want. Alternatively, there is also a **-v** option while removing the container.

# docker volume rm data { **Removes the volume data** }  
  
We got three different implementations in volumes:

1. Anonymous volumes
2. Named Volumes
3. Bind Mounts

**Anonymous volumes**: don’t have a name it’s not so easy to work with Anonymous volumes if there are multiple Anonymous volumes.

#  docker run -it -v /my-data --name container1 ubuntu bash

**Named volumes**: have a name to identify them so it’s easy to work with names if there are multiple volumes we can easily identify them with names.

#  docker run -it -v vol1:/my-data --name container1 ubuntu bash

**Bind Mounts**: same process of mounting a volume but this time **we will mount an existing host folder in the Docker container**. This is an interesting concept and is **very useful** if you are looking **to do some development where you regularly modify a file** in a folder

#  docker run -it -v /home/path:/my-data --name container1 ubuntu bash

**MySQL Volumes Example  
=====================**# goto hub.docker.com  
# search for mysql and goto → Details → Click on latest Dockerfile → Scroll down and you can see **VOLUME /var/lib/mysql,** this is the default location of MySQL Databases.  
  
This **mysql image** is programmed in a way to tell docker, when we start a new container from it, it actually **creates a new volume location** and **assign it to** this directory **/var/lib/mysql,** in the container,  
  
 Which means any files we put in the container will outlive the container, until we manually delete the volume.   
  
Volumes need manual deletion, you can’t clean them up just by removing the container that’s an extra setup with volumes, the whole point of volume command is to say that this data is particularly important at least much more important than container itself.  
  
**Note: You might wanna do  
 # docker volume prune**

**to cleanup unused volumes and make it easier to see what you’re doing.**# docker volume ls

Let’s run a container from it:

# docker container run -d --name mysql -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True mysql  
# docker container ls  
# docker volume ls  
# docker container inspect mysql { you can see **Volumes /var/lib/mysql** }

And if you go up in the output, you can see **Mounts,** and this is actually the running container

so actually the **container actually thinks it’s getting data or writing data is from /var/lib/mysql**,

But in this case, we can see the **data is actually living** in **Source** above line of /var/lib/mysql on the host.  
  
So let’s do:  
 # docker volume ls

If you are doing this on a linux machine, You can actually navigate to the volume **Source** location { **/var/lib/docker** } and can see the data, i.e some databases.  
  
And if i do just hit an up arrow and create a multiple mysql container:

# docker container run -d --name mysql2 -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True mysql  
# docker volume ls

# docker container run -d --name mysql3 -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True mysql  
# docker volume ls

# docker container run -d --name mysql4 -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True mysql  
# docker volume ls  
  
 We can see **two volumes** but we can see the problem right ??  
There is **no easy way to tell which volume belongs to which container**.  
  
# docker container stop mysql  
# docker container stop mysql2

# docker container ls  
# docker volume ls  
 # docker container rm -f mysql mysql2 mysql3  
  
# docker volume ls

**{ my volumes are still there, my data is still safe, so we solved one prob }**

**So how we make little more user friendly ??**

That is where **named volumes** come in, the ability for us to specify names for docker volumes.

**Named** **volume** [ i can put a name in front of it with **: that is known as named vol** ]  
  
# docker container run -d --name mysql -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True **-v mysql-db:/var/lib/mysql** mysql

# docker volume ls

{ you can see my new container is using a new volume and it’s using a friendly name }  
  
# docker volume inspect **mysql-db {** this is easier to use here**}**And if i removed my container:  
 #  docker container rm -f mysql { -f coz it’s still running, if not stop & remove }

And if i run another container with some other name and same volume:  
 # docker container run -d --name **mysql3** -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=True **-v mysql-db:/var/lib/mysql** mysql  
  
# docker volume ls { **only mysql-db is there** }  
  
You can see that we haven’t created a new volume, but still using the same mysql-db volume from earlier.  
  
# docker container inspect mysql3

{ and we can see Volumes, changed the **Source** location to be a little friendlier as well }

**More on Mysql - Volumes  
=====================**Running mysql container with password:

# docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=mypassword -d mysql

The following command line will give you a bash shell inside your mysql container:

# docker exec -it some-mysql bash  
 # mysql -u root -p

The MySQL Server log is available through Docker's container log:

# docker logs some-mysql

Removing volumes along with container:  
 # docker container rm -f -v some-mysql

**Persistent Data: Bind Mounting  
==========================**

Look at the same process of mounting a volume but this time **we will mount an existing host folder in the Docker container**. This is an interesting concept and is **very useful** if you are looking **to do some development where you regularly modify a file** in a folder outside and expect the container to take note or even sharing the volumes across different containers.

Bind mounts are actually cool, this helps how to use docker for local development.  
  
So really a bind mount is just a mapping of the host files or directories into a container file or directory.  
In background, it’s just having two locations pointing to the same physical location(file) on the disk.  
  
**Full path rather than just a name** like volumes, the way actually docker can tell the difference between named volume and bind mount, is that **bind mounts starts with a forward slash /**  { root }  
  
Now where it really comes to shine is with development and running services inside your container, that are accessing the files you are using on your host which you are changing. So let’s do that with nginx:

**installation of Nginx  
==================**

# sudo docker run -d -P --name web-server nginx { **-P** **random ports** }

**Nginx  
 =====**# docker container run -d --name nginx1 -p 8080:80 nginx  
# docker container run -d --name nginx2 -p 80:80 nginx  
  
Check the site in host machine<http://ip-address>:8080  
The site which we are seeing is default nginx html file, actually coming from default nginx image.

**Nginx with Mountpoints example  
========================**# docker container run -d --name nginx -p 80:80 -v ~/website:/usr/share/nginx/html nginx

**Nginx Example ( Volumes )**

**======================**

# mkdir nginxlogs

# sudo docker run -d -v ~/nginxlogs:/var/log/nginx -p 80:80 nginx

I do this because every time i don’t want to go into the container and check the logs.

Now i have all the logs in host machine itself rather than container.

**Using multiple volumes on Single Container**

**====================================**

Saving both logs and website data

# docker container run -v **~/nginx-logs:/var/log/nginx** -v ~/website:/usr/share/nginx/html nginx

**Wordpress example - Backup**

**========================**

**## Creating DB**

# docker run -d --name=**wp-mysql** -e MYSQL\_ROOT\_PASSWORD=mypassword -v ~/mysql-data:/var/lib/mysql mysql  
# docker exec -it wp-mysql bash

# mysql -u root -p  
 # create database wordpress;

**## Creating WP**

# docker run --name my-wordpress --link wp-mysql:mysql -p 8080:80 -d -v ~/wp-data:/var/www/html wordpress

For **Custom Docker Images** go with following url

<https://github.com/ravi2krishna/Node-Js-Sample-App>