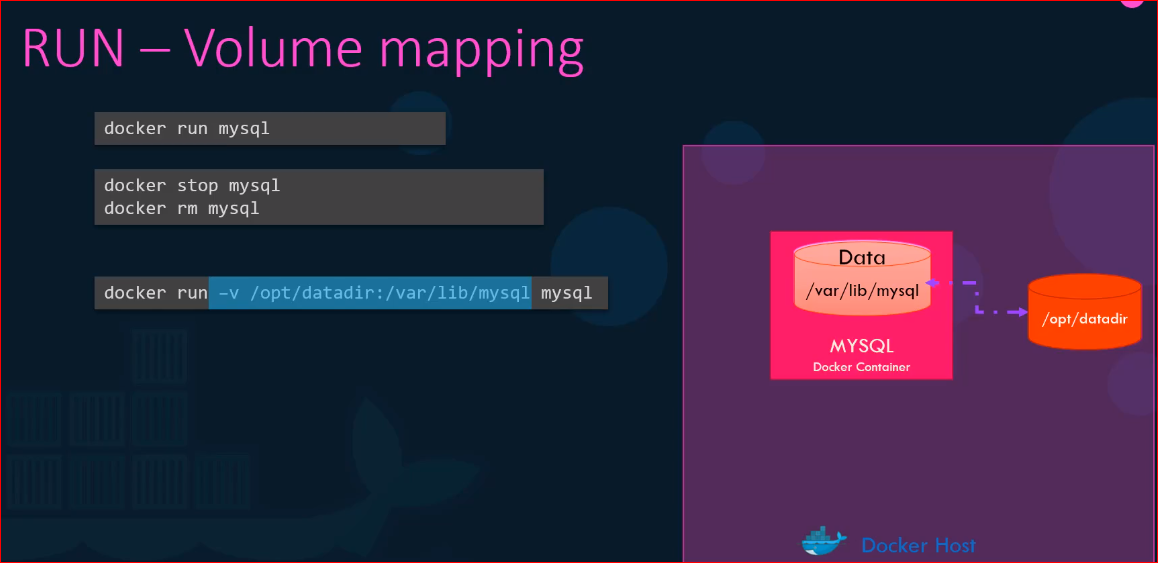
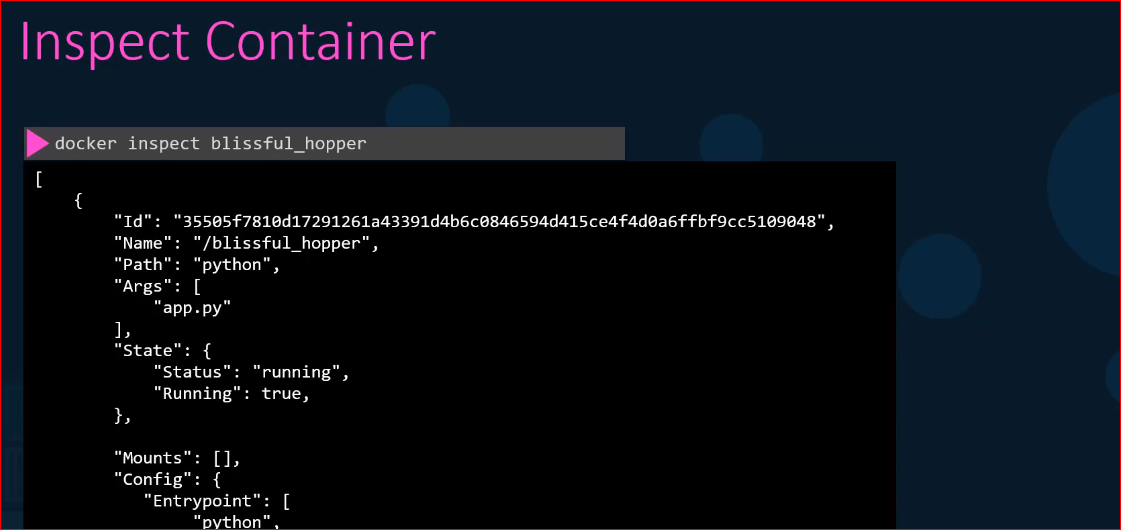
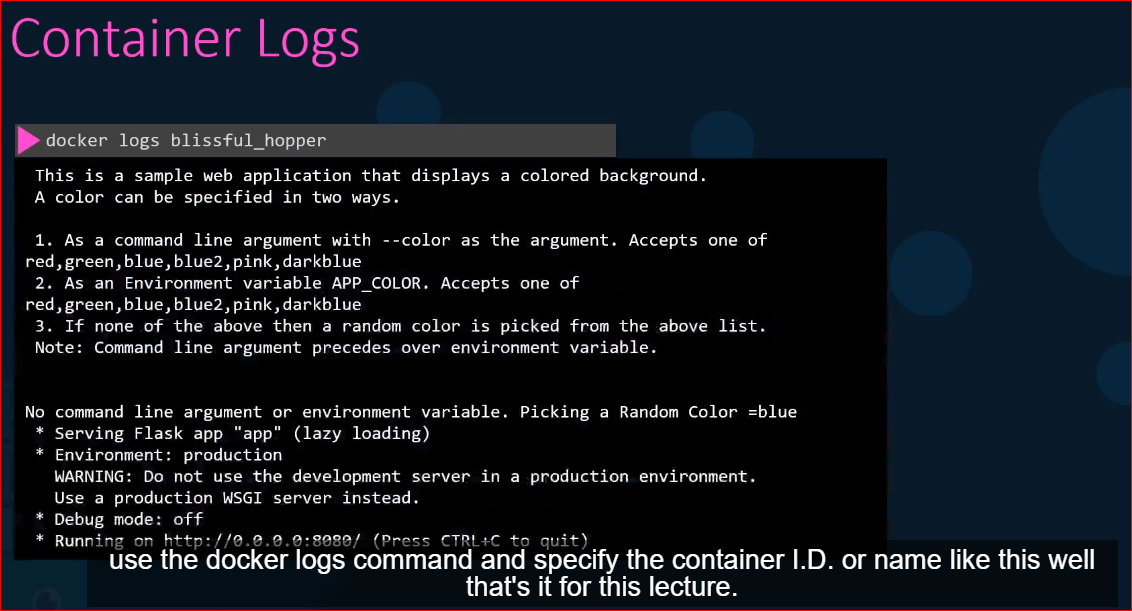


Data persistence in docker container

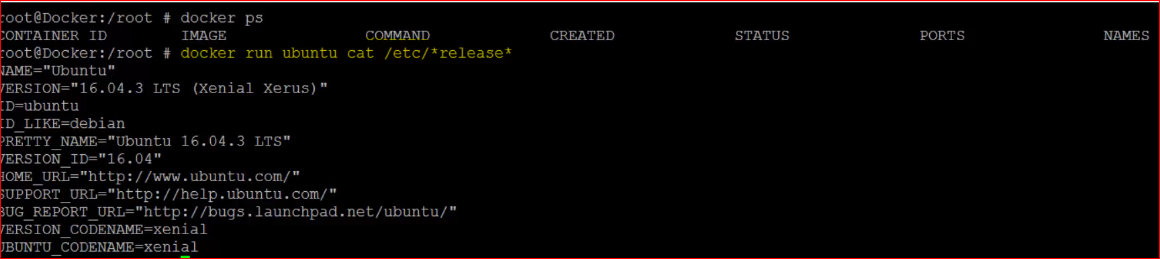




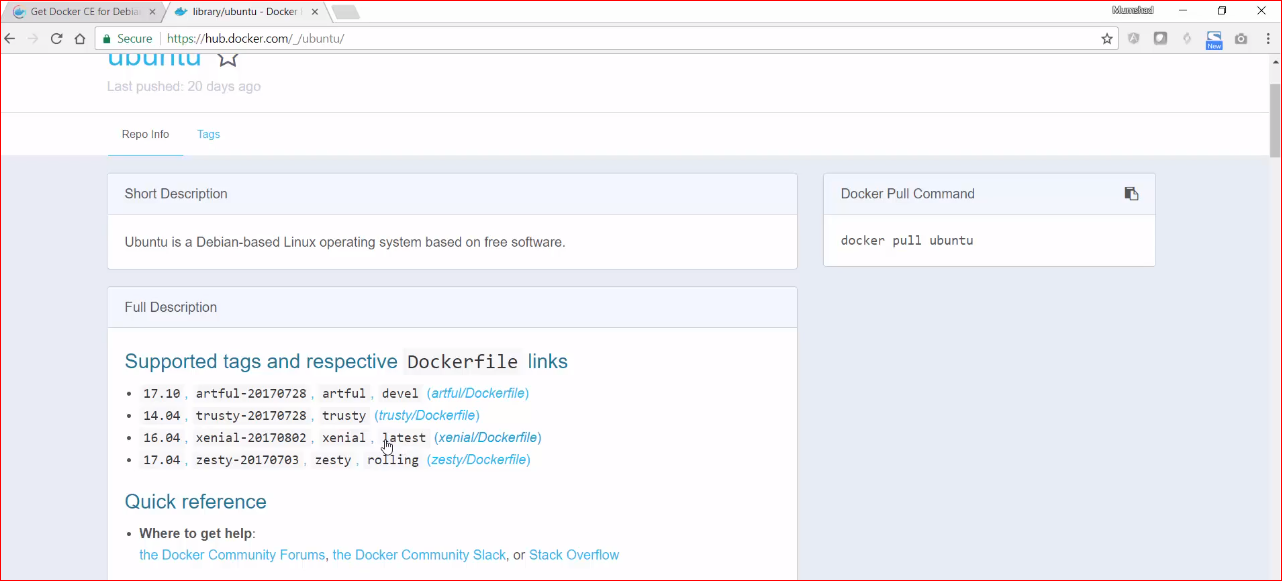
* I ran web application in container with -d option it ran the application in background in detached mode
* How do I view logs which happens to be the contents written to the standard out of that container?



docker run ubuntu cat /etc/\*release\*



But what if I want to run another version of this particular OS. First go to docker hub site, Identify the repository





$ **docker ps -a**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

5a8f89adeead ubuntu:14.04 "/bin/bash" About a minute ago Exited (0) About a minute ago agitated\_newton

$ **docker commit 5a8f89adeead newimagename**

$ **docker run -ti -v "$PWD/somedir":/somedir newimagename /bin/bash**

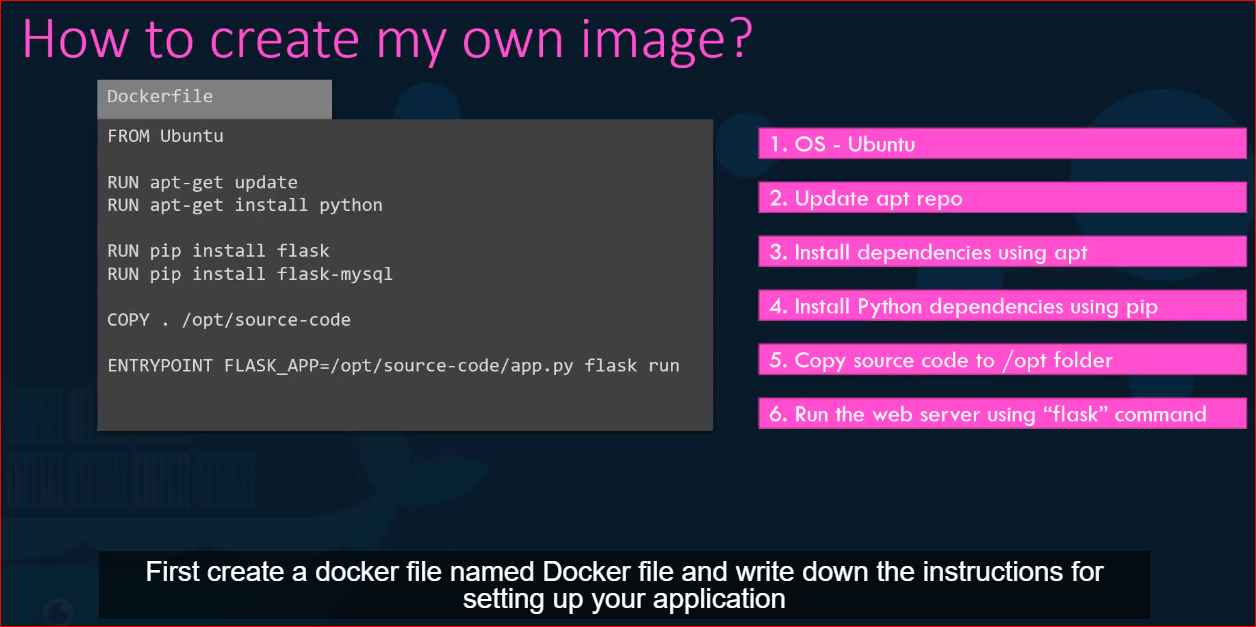
**I installed all again with persistence volume**

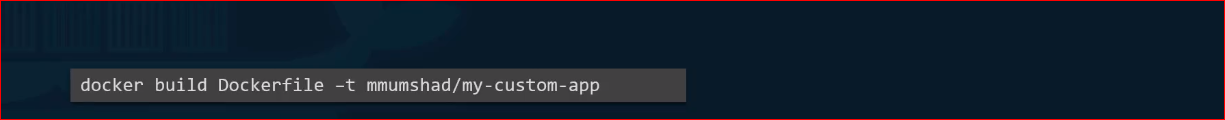
[root@localhost ~]# **docker run -p 8080:8080 -v /root/jenkins\_config:/var/jenkins\_home -u root jenkins/Jenkins**

**Docker Images:**

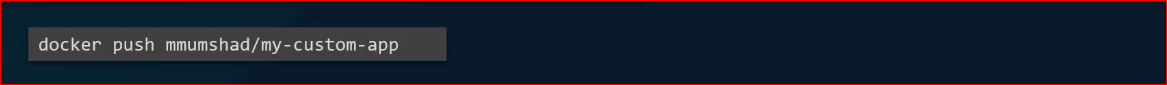
**What am I containerizing?**

* We are going to create our own image but before that why would you need to create your own image
* Maybe you cannot find a component or service that you want to use as part of your application on docker hub already or
* You or your team decide that the application you are developing will be dockerized for ease of shipping and deployment
* In this case we are going to containerize an application, a simple web application that I have built using the python flask framework
* First, we need to understand what we are containerizing?
* What application we are creating an image for?
* And how the application is built?
* Start by thinking what you will do to deploy the application manually? Write down the steps required in correct order.

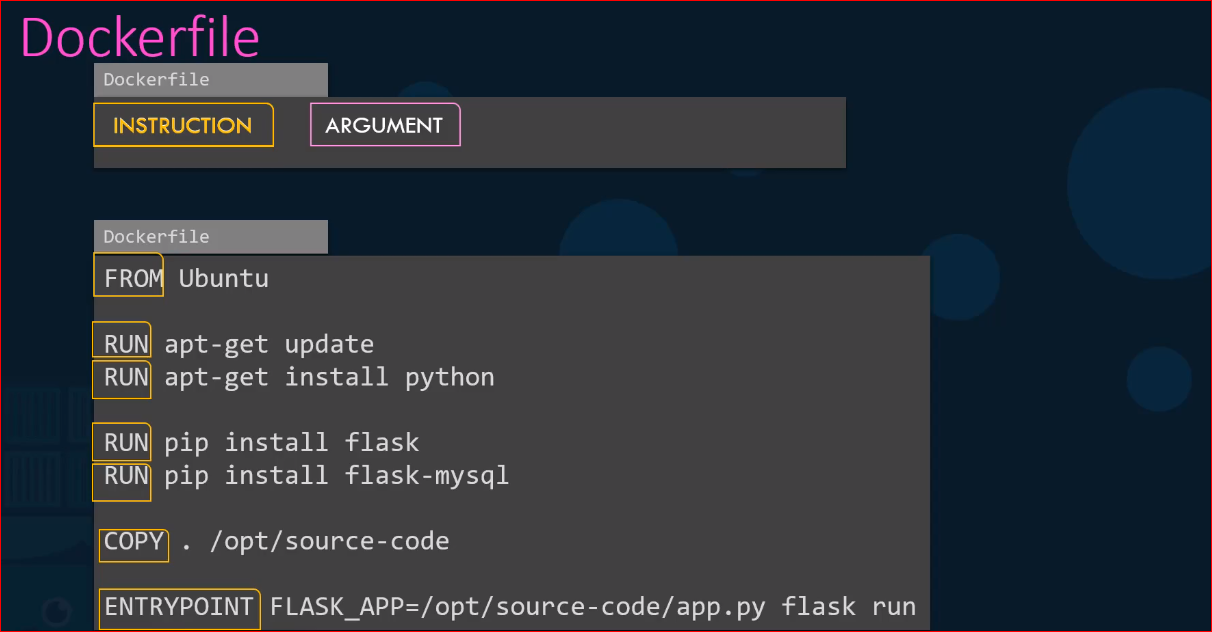




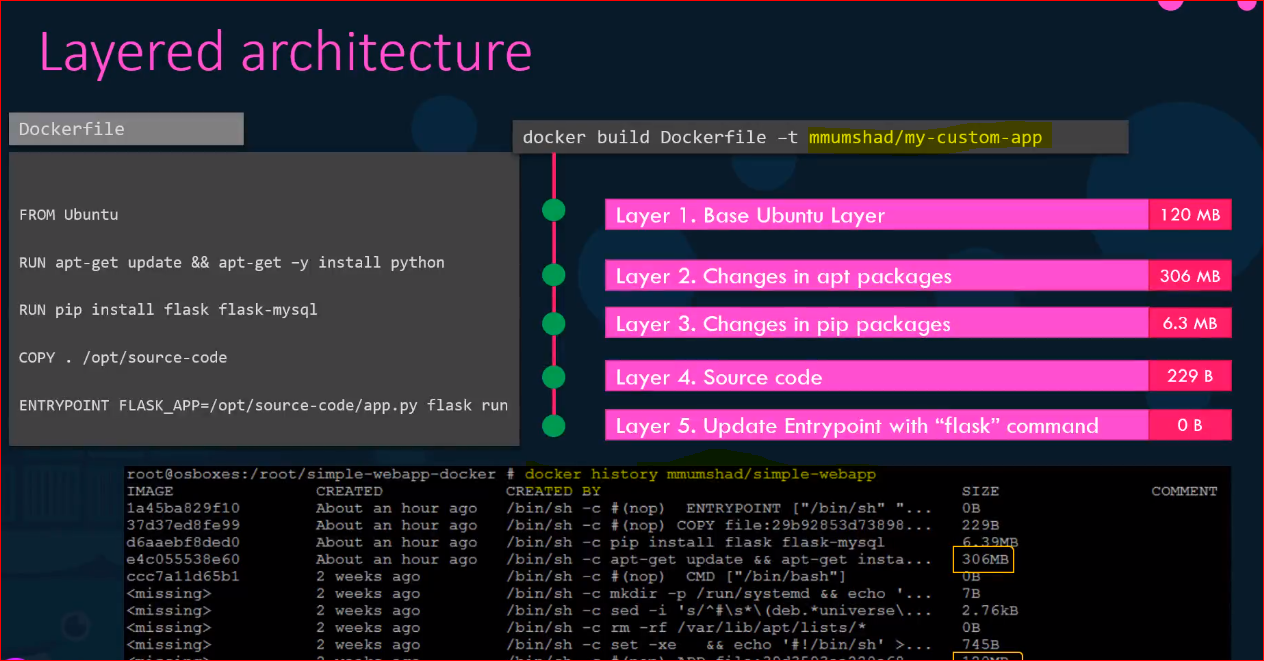
This will create image locally on your system



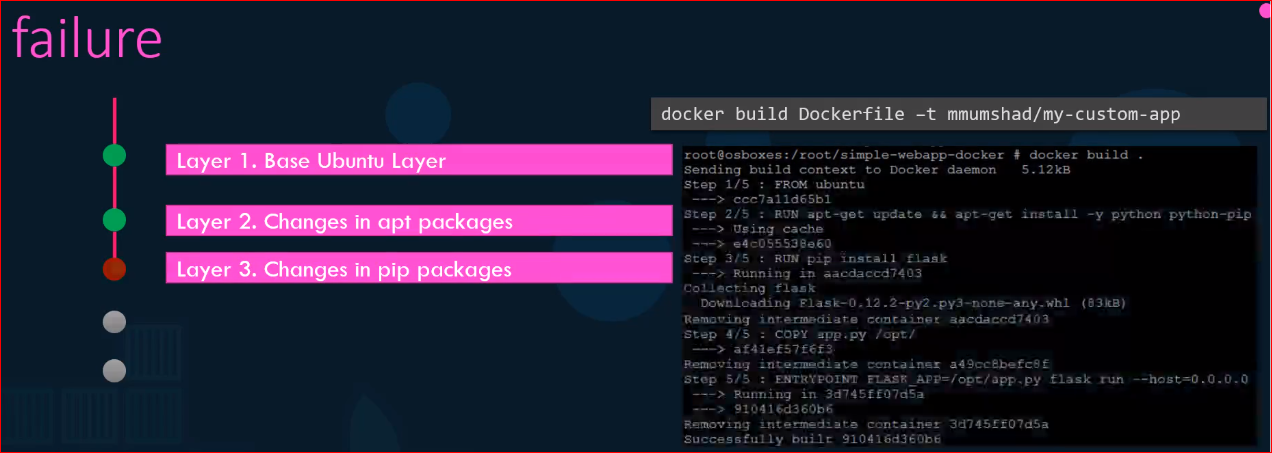
Use docker push to make your image available publicly on docker HUB registry



Instruction are written in CAPITAL letters and Arguments are the rest or command



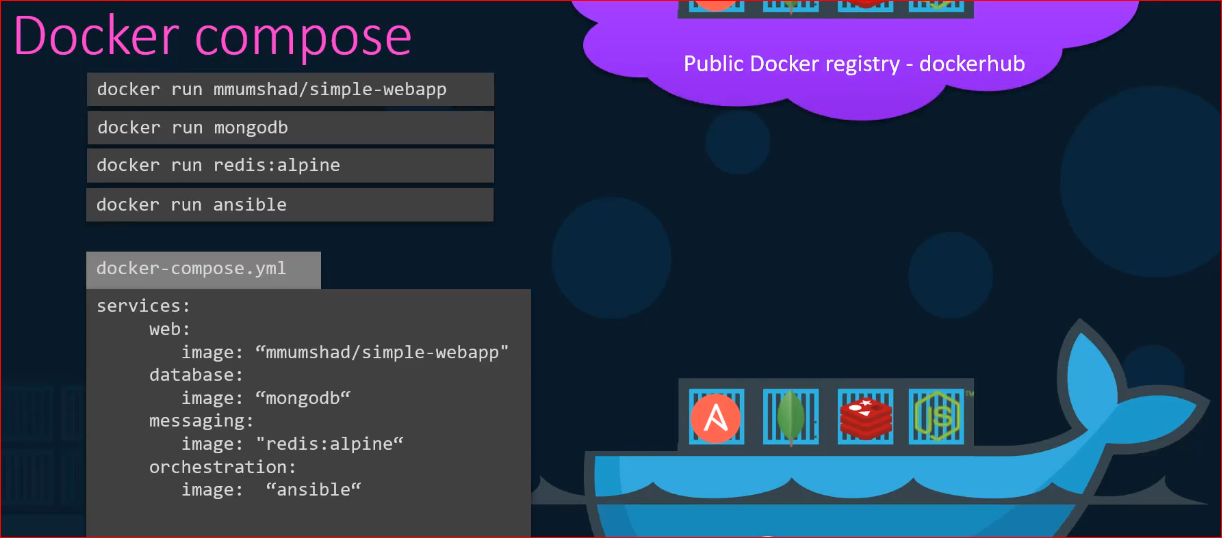
* All layers built are cached by docker so if some got failed and you re-run docker build command it will use cache to start from that step and continue to build from that step
* The same is true if you were to add additional steps in the docker file this way rebuilding the image is faster and you don’t have to rebuild the entire image each time
* Helpful especially if you want some changes in source code of your application as it may change more frequently only the layer above the updated layer is rebuild





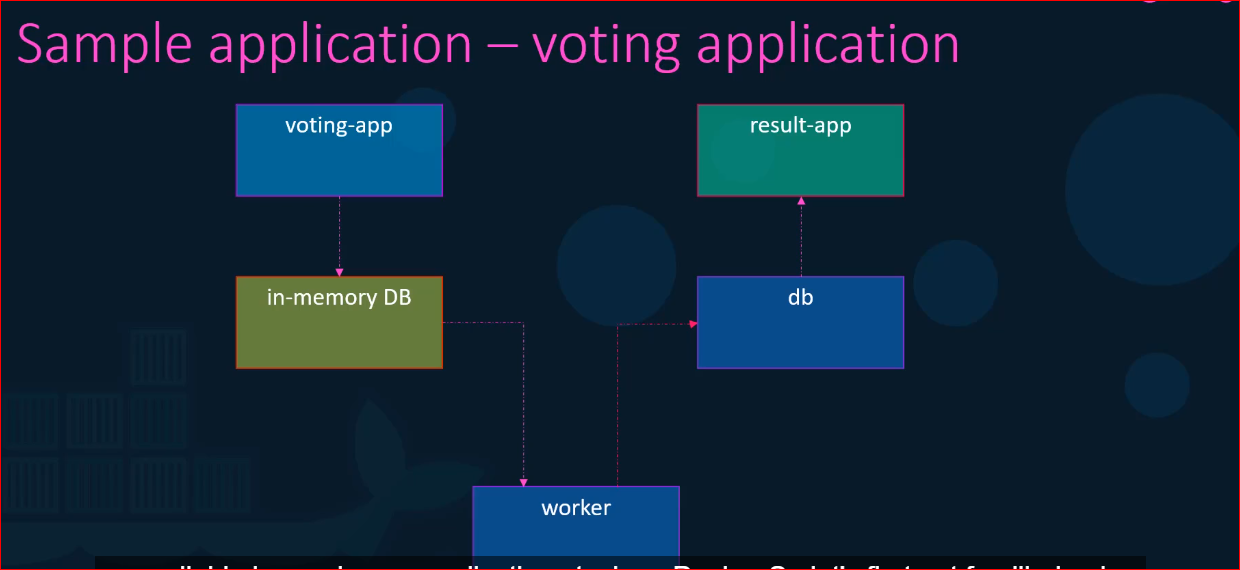
**Docker Compose (DC)**

* If we need to setup a complex application running multiple services a better way to do is by docker compose
* With DC we could create a configuration file in YAML format called docker-compose.yml and put together the different services and options specific to those to run them in this file
* Use **docker-compose up** to bring the application up and running

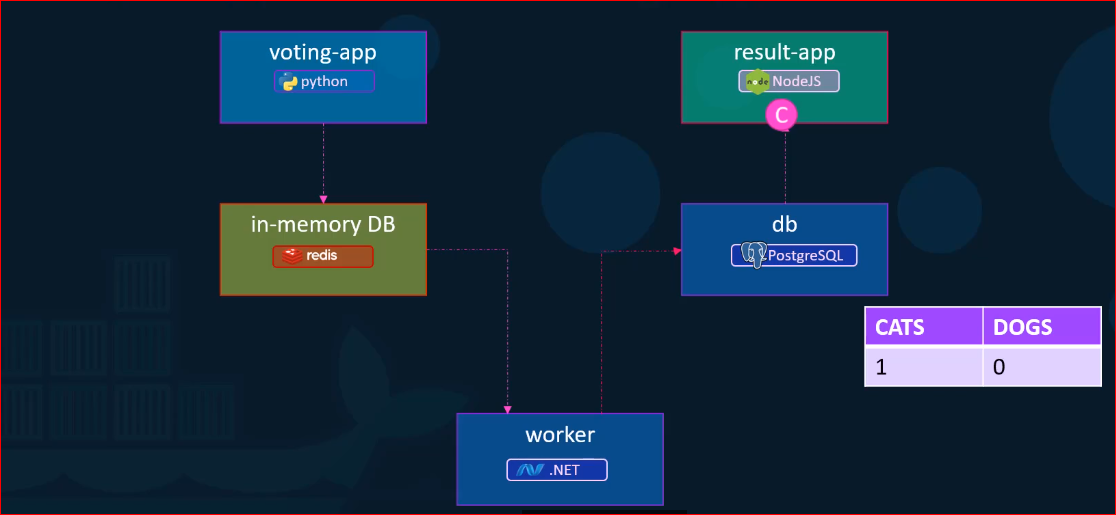


* This is easier to implement, run and maintain as all configurations are stored in docker compose configuration file
* This is all only applicable to run a container on single docker host

**Example: Voting Application**

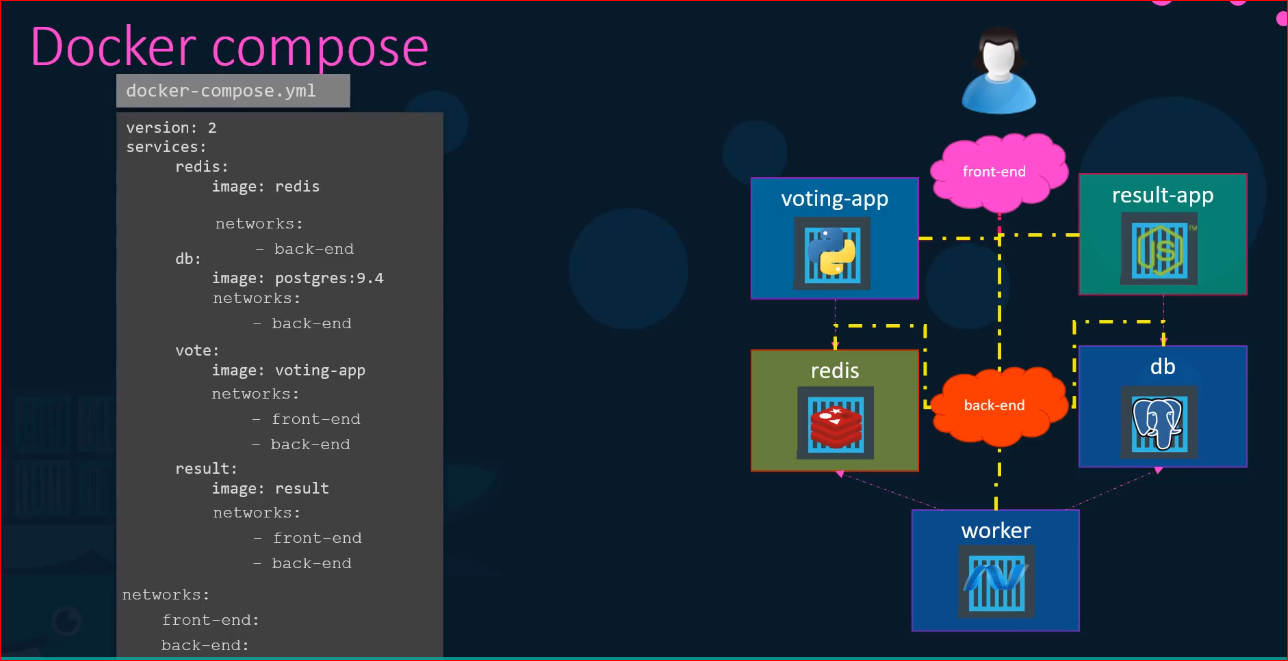


* Sample voting application which provides an interface for user to vote and another interface to show the result
* Application consist of various components such as the voting app which is the web application developed in python to provide the user with interface to choose between 2 options the cat and a dog
* When you vote the selection is stored in a Redis (in this case serve as database in memory)
* This vote is then processed by the worker which is an application written in .net
* The worker application takes the new vote and update it into persistent database which is a postgres Sql in our case
* The PostgresSQL simply has a table with the number of votes for each category
* Finally, the result of the votes is displayed in web interface which is another web application developed in NodeJS



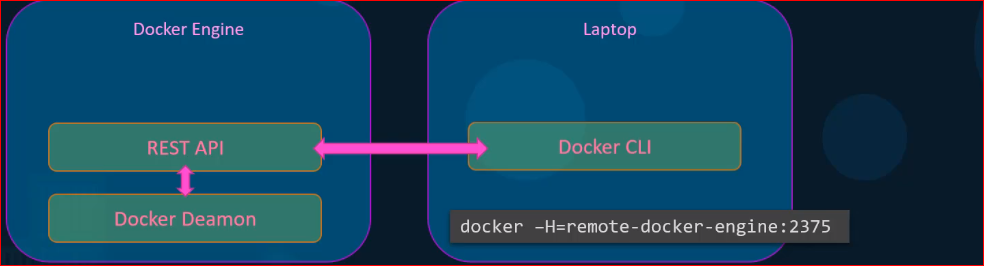
* Link is a cmd line option to link 2 container together

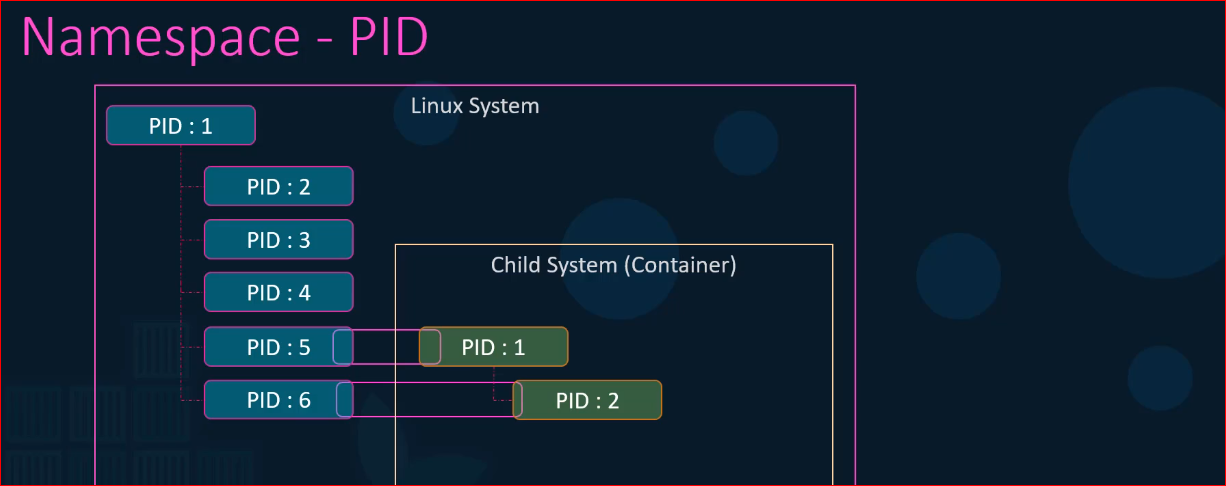




**Docker Engine:**

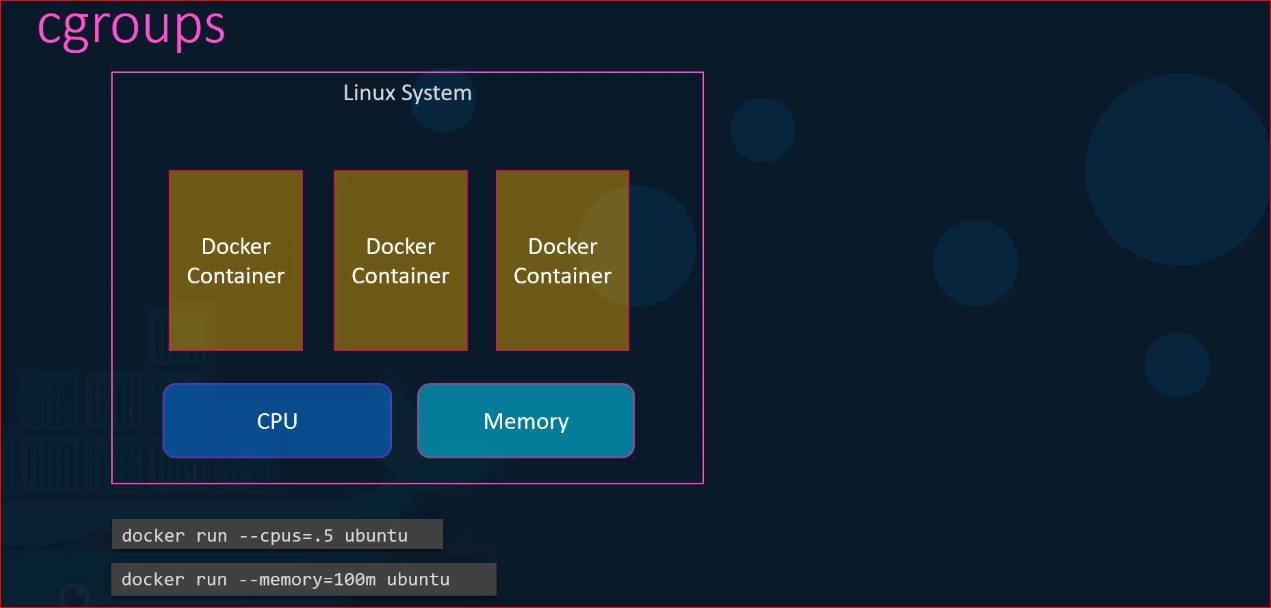
* Installing docker on linux means installing 3 different competency:
  + Docker CLI
  + REST API
  + Docker daemon
* **Docker daemon** 
  + It is a background processes that manages docker objects such as the images, containers, network, and volumes
* **Docker REST API**
  + It is the API interface that programs can use to talk to the daemons and provide instruction
  + You could create your own tool using this REST API
* **Docker CLI**
  + It uses REST API to interact with docker daemon
  + Docker CLI need not necessary to be on the same host
  + It could be on your laptop and can still work with remote docker engine



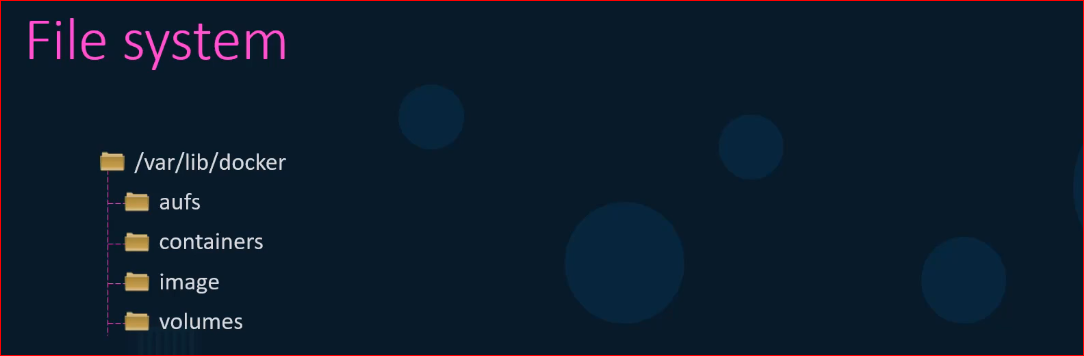


2 processes cannot have same process id so to avoid that we use namespace

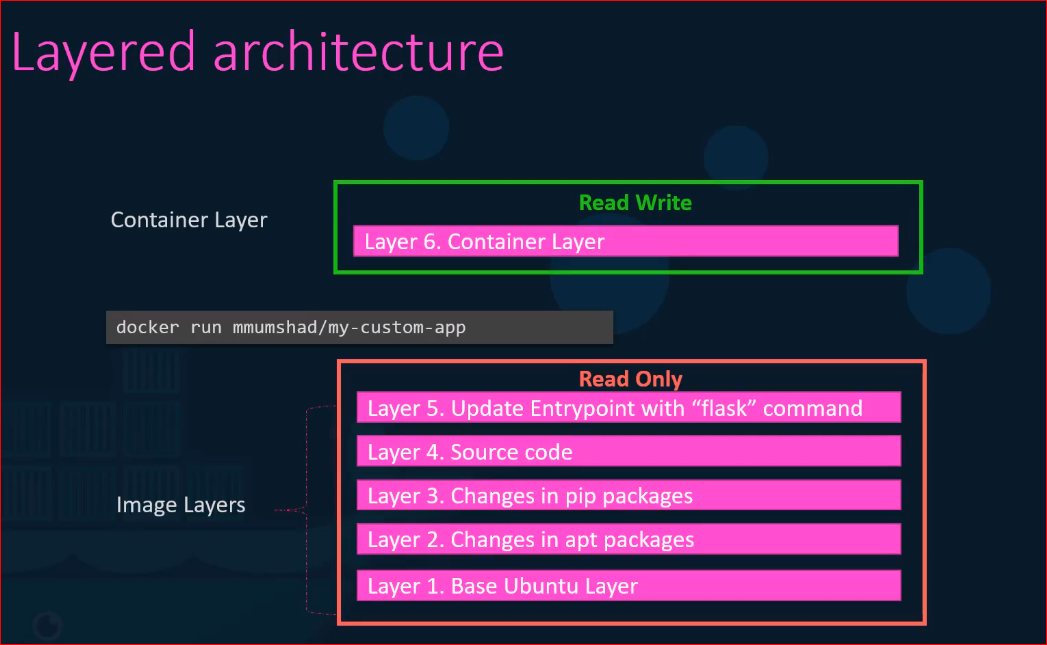
* Docker host as well as container share the same system resources such as CPU and memory
* How much of the resources are dedicated to the resource and containers?
* How does docker manages share the resources between the container?
* By default, there is no restriction as of how much of resources a container can use and hence a container may end up utilizing all the resources of the underlying host
* But there is a way to restrict the amount of CPU and memory a container can use.
* Docker uses 3 groups or control groups to restrict the amount of hardware resources allocated to each container
* You can use below 2 option in docker com:
* **Docker run - -cpus=.5 ubuntu**
  + Ensure that container does not take up more than 50% of the host
* **Docker run - -memory=100m ubuntu**
  + Limits the amount of memory a container can use to 100 megabytes

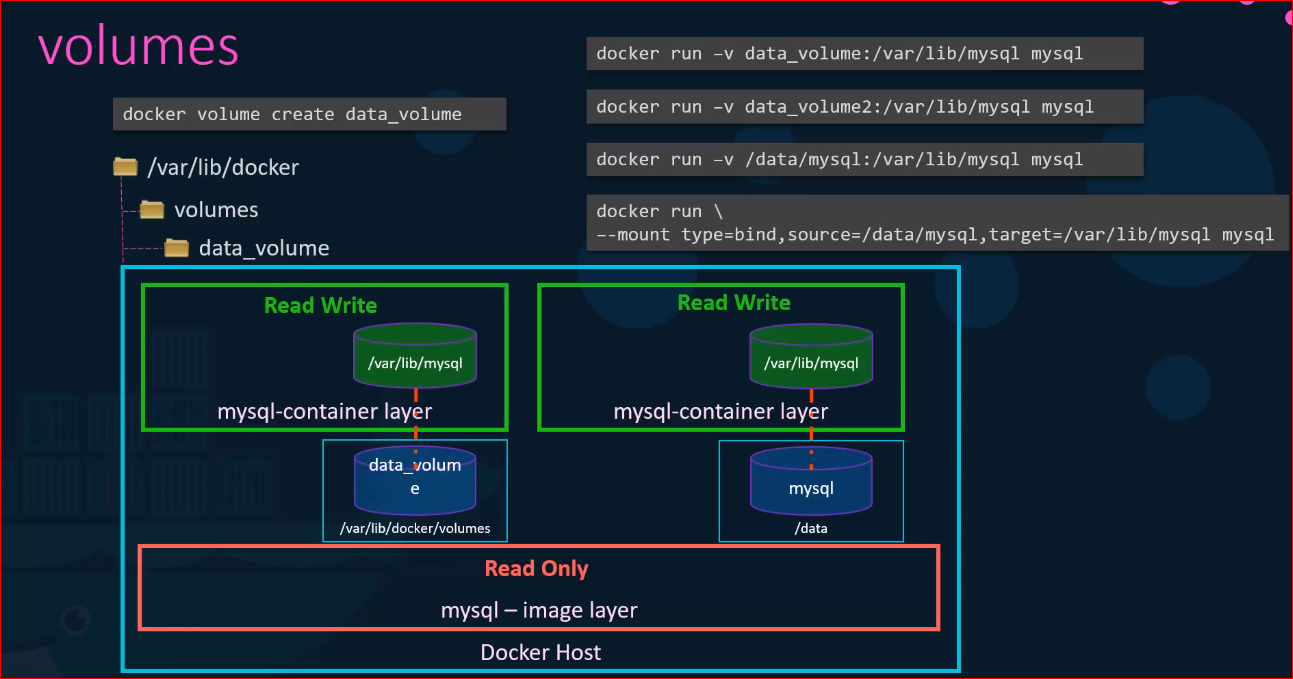


**Docker Storage:**



* This is were docker stores it all data by default
* Data here means files related to images and containers running on the docker host





* **Docker uses storage drivers to enable layered architecture**



* Storage driver’s selection depends on the underlying OS
* For example, **Ubuntu**. The default storage drivers is **ufs** and it’s not available on other OS such as **fedora** and **centOS**

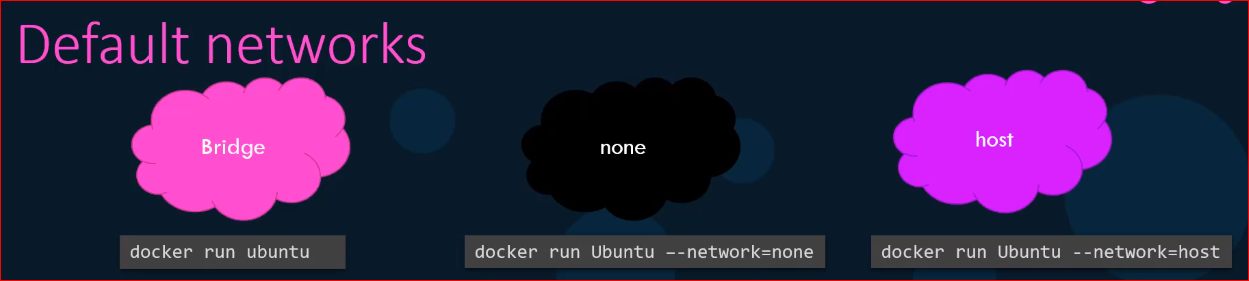
Read more details here:

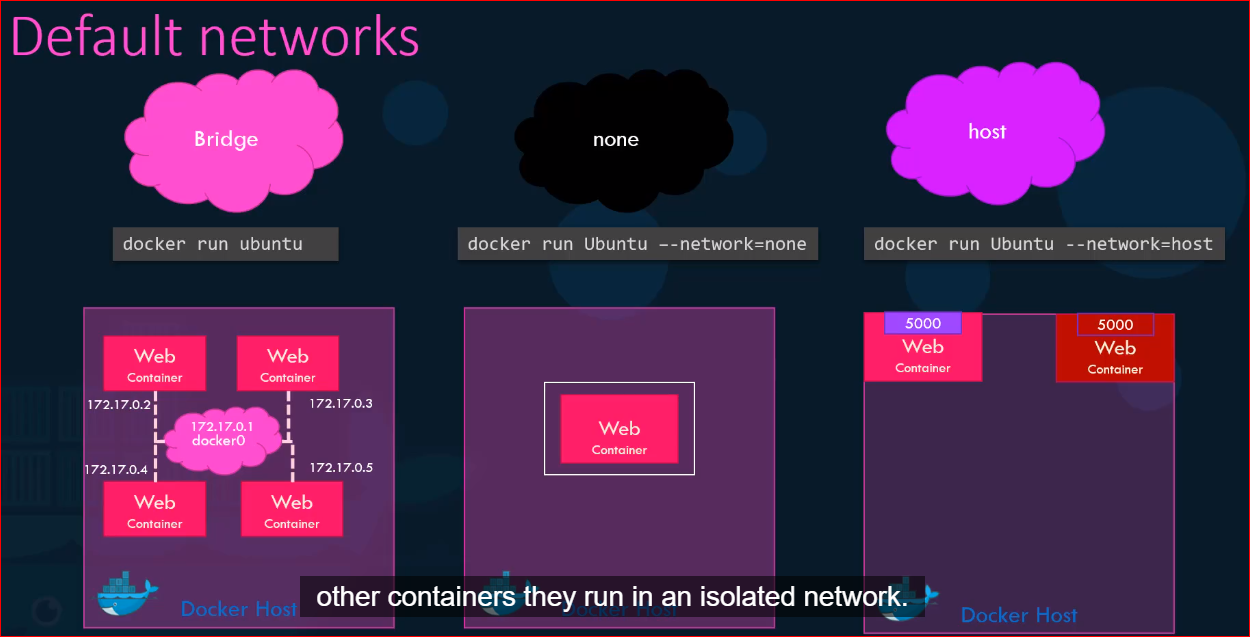
Storage in Docker : <https://docs.docker.com/storage/>

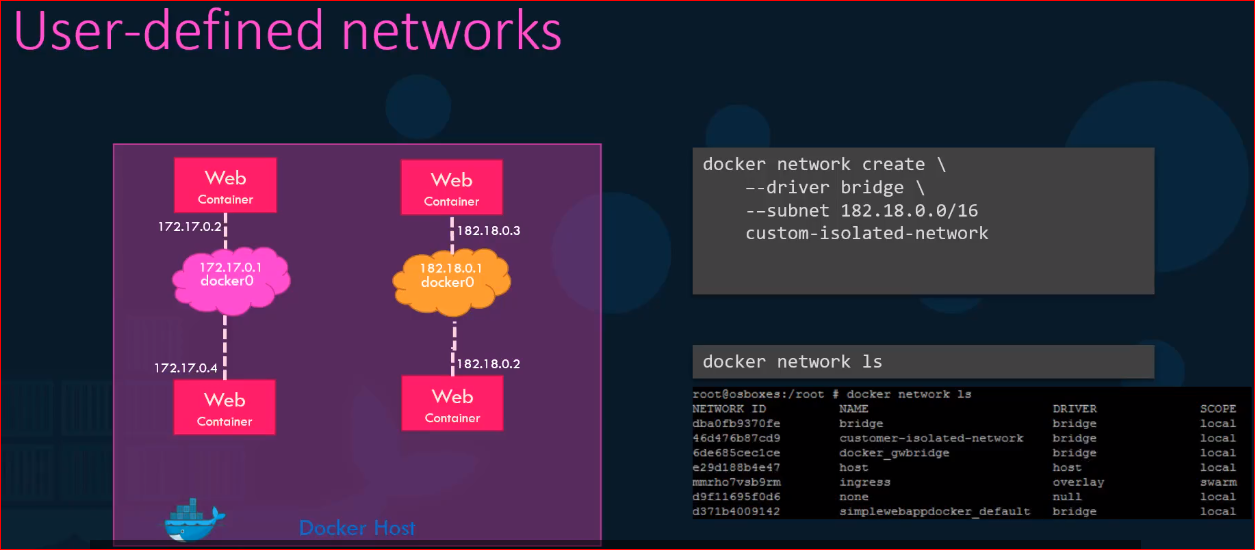
Storage Drivers: <https://docs.docker.com/storage/storagedriver/>

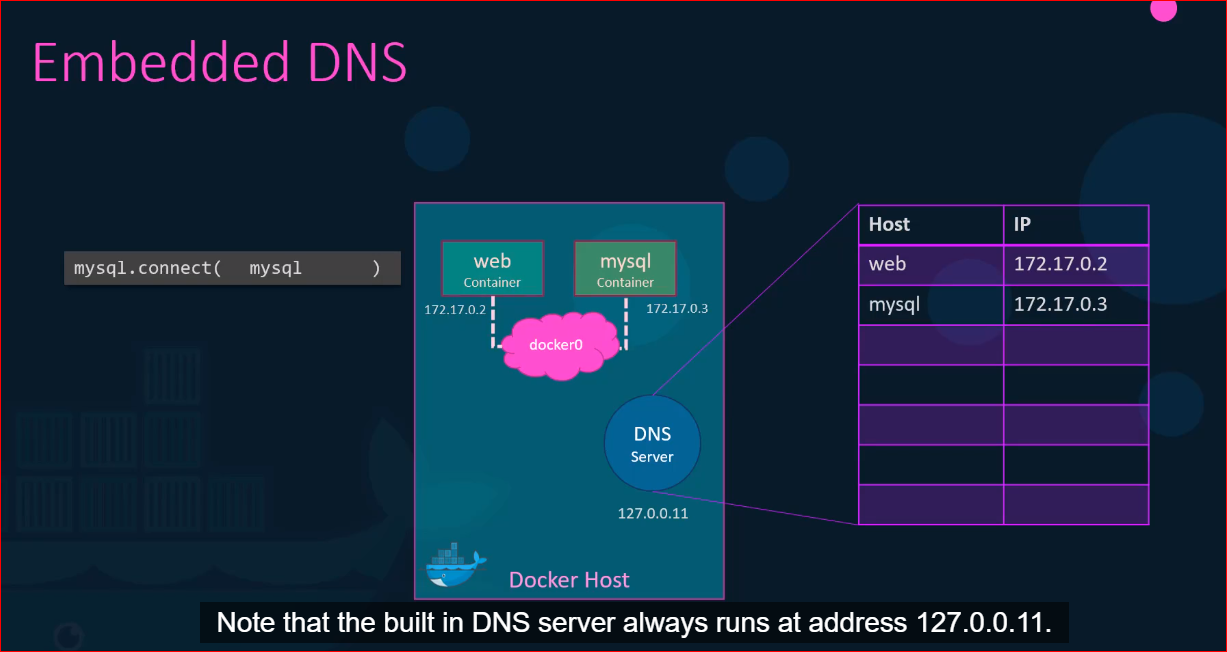
**Docker Networking:**

* When docker is installed it creates 3 networks automatically
  + Bridge (default)
  + None
  + Host









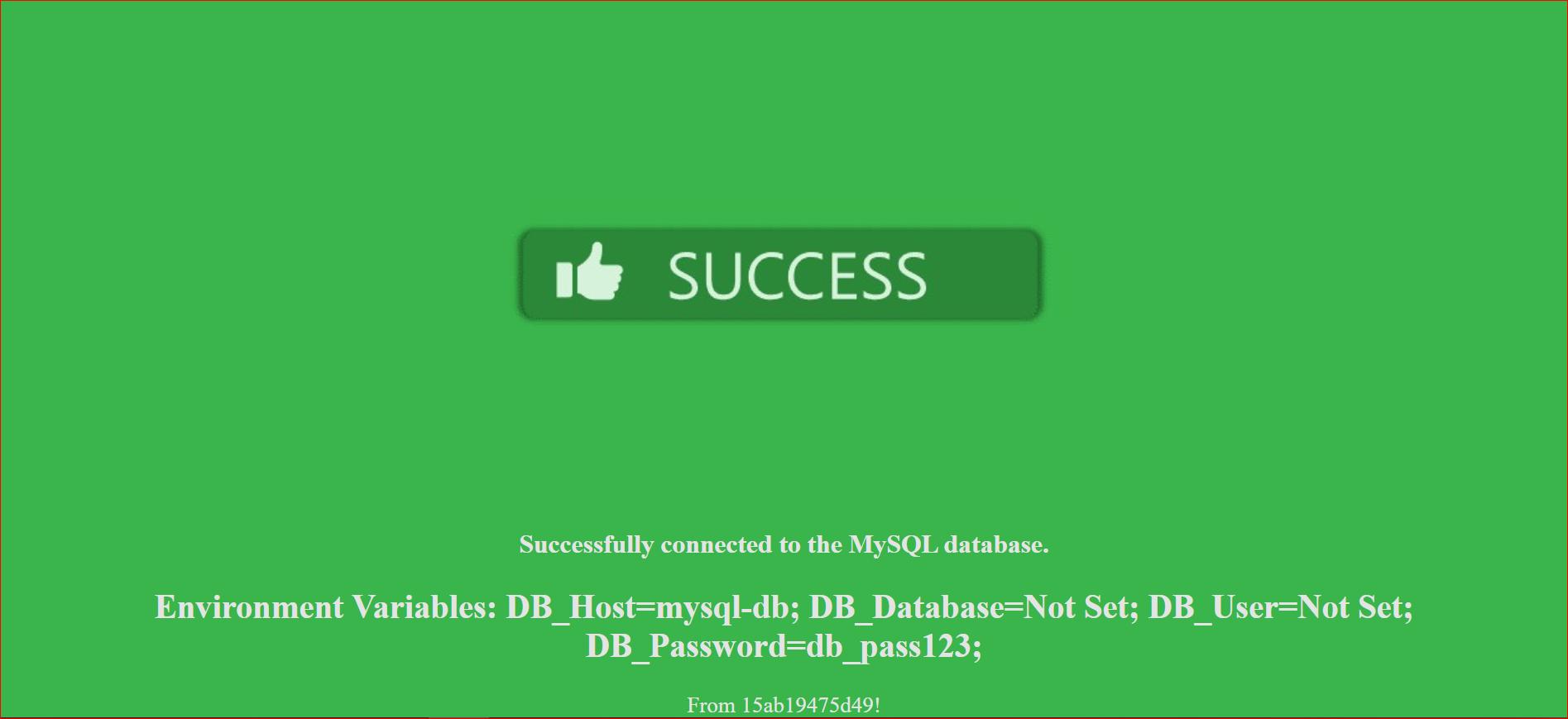
**Lab:**

Deploy a web application named webapp using the kodekloud/simple-webapp-mysql image. Expose the port to 38080 on the host.  
  
The application makes use of two environment variable:  
1: DB\_Host with the value mysql-db.  
2: DB\_Password with the value db\_pass123.  
Make sure to attach it to the newly created network called wp-mysql-network.

Also make sure to link the MySQL and the webapp container.

**Docker cmd:**

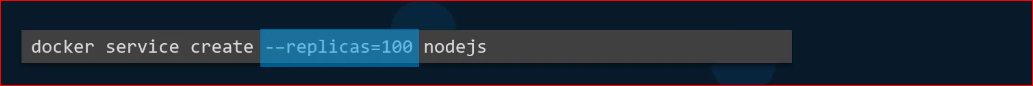
docker run --network=wp-mysql-network -e DB\_Host=mysql-db -e DB\_Password=db\_pass123 -p 38080:8080 --name webapp --link mysql-db:mysql-db -d kodekloud/simple-webapp-mysql





**Container Orchestration:**

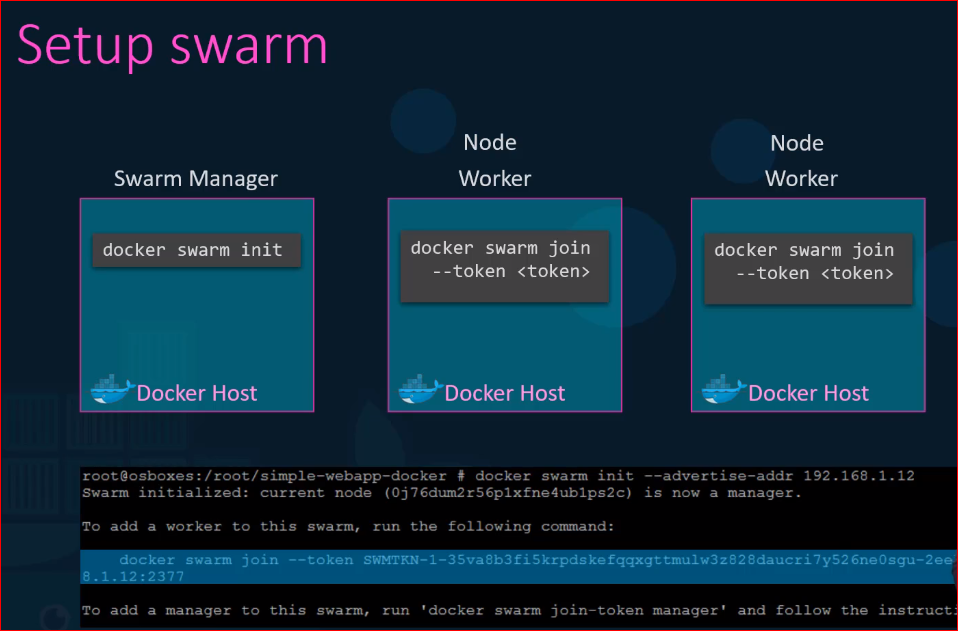
* It is a solution that consist of set of tools and scripts that can help host containers in the production environment
* Container orchestration solution consist of multiple docker hosts that can host containers that way if one container fail
* The application is still accessible through the other.
* A containers orchestration solution allows you to deploy hundreds or thousands of instances of your application with a single command



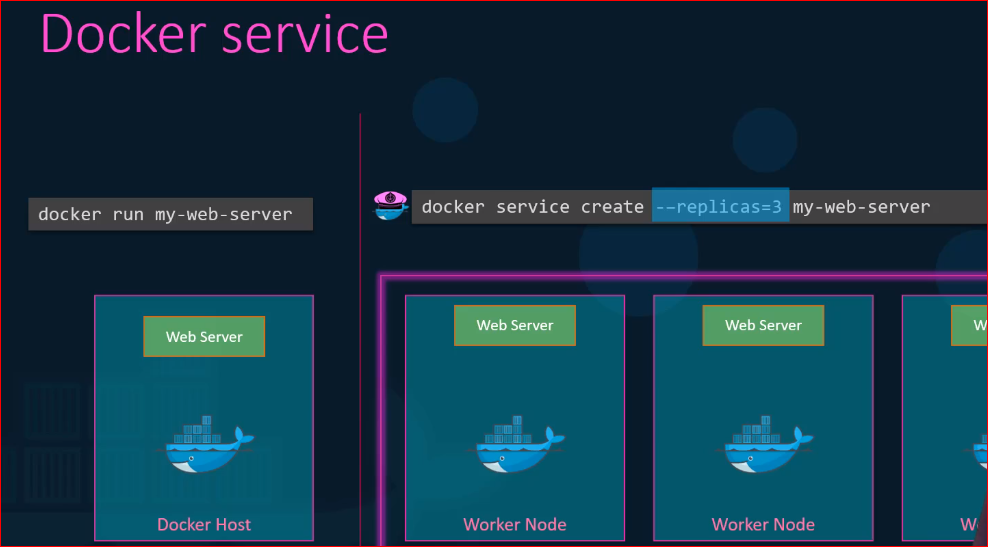
* It also helps you to scale up and scale down your number of instances depending on the number of users increases or decreases
* Some solution can even help you in adding additional hosts to support the user load and not just clustering and scaling the container orchestration
* Also provides solution for advanced networking between these containers across different hosts as well load balancing users request on different hosts
* They also provides support for sharing storage between the host as well as support for configuration management and security within the cluster
* There are multiple docker containerization solution available today
  + **Docker swarm** from **Docker**
  + **Kubernetes** from **Google**
  + **Mesos** from **Apache**
* While Docker Swarm is very easy to setup and get started
* It lacks some of the advanced Auto scaling features required for complex production and great application
* Mesos on the other hand is quite difficult to setup and get started but support many advanced features
* Kubernetes most popular of it all is a bit difficult to start and get started but provides a lot of option to customize deployment and has support for many different vendors.
* Kubernetes that is now supported on all public cloud service providers like GCP, Azure and AWS
* The kubernetes project are one of the top ranked projects on GitHub

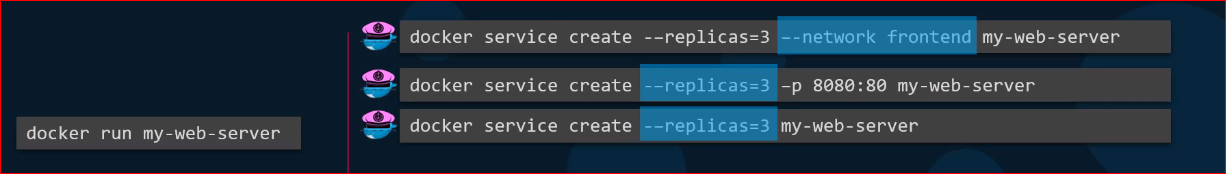
Docker swarm:

* With DS you could combine multiple docker machines together into a single cluster docker
* It takes care of distributing your services and applications instance into separate hosts for high availability and for load balancing across different systems and hardware
* To setup docker swarm you must first have multiple hosts with docker installed on them
* Then designate one host to be manager or master or swarm manager as it is called and others as slave or worker node



* Once done run docker swarm init command on manager and it will initialize the docker swarm manager
* The output also provide the cmd to be run on worker node
* Copy the cmd and run on worker node to join the manager
* The key component of swarm orchestration is the **docker service**
* Docker services are one or more instances of a single application or service that runs across the site the nodes in the swarm cluster for example





* The docker service cmd should be run **only on the manager node** not on the worker node