

OR

**DOCKER**: is used for Package the written code and TO Deploy the Package Code.

**What DOCKER will do?**

* Docker makes us to use the IMAGE. With the help of the image we can run the light weight machines
* **Main Purpose**  of DOCKER is to package and containerized applications and to ship them and to run them like anytime anywhere as many times as you want

**DOCKER IN DEVOPS:**

With DOCKER Developers and Ops team work together to transform the guide into a docker file with both of their requirements

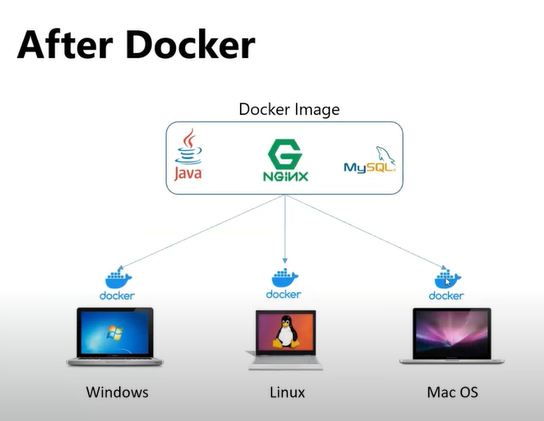
* Then the DOCKER file is used to Create image for their applications
* This Image can now run on any host with DOCKER installed on it and guaranteed to run the same way
* So the OPS team can now simply use the image to deploy the application since the image was already working it works in the same way after deployed the application

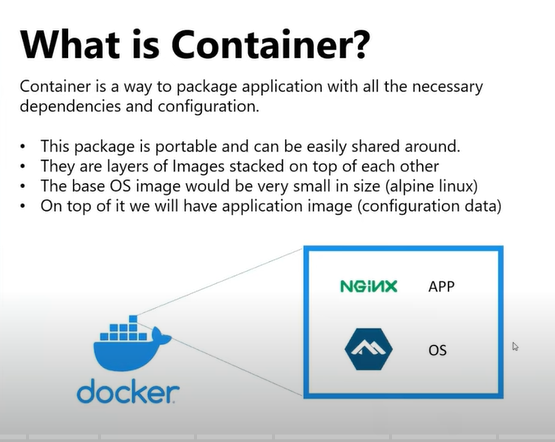
**Virtual Machines:**

* **Virtual Machines are heavy weight machines**
* **1st) install Complete OS (with/without need)**
* **2nd) Compatiblity issues**
* **We can avoid this compatibility issues with the help of the DOCKER**

**CONTAINERS: will have very light weight system**

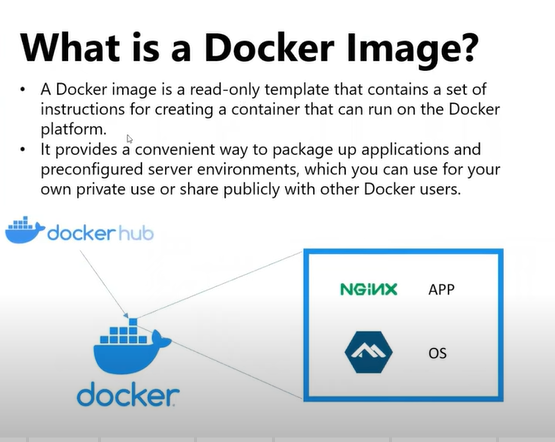
**DOCKER ENGINE: platform where we run our containers**

* When we host our containers on the DOCKER ENGINE containers will run seemlesly without any compatibility issues irrespective of machines (windows / linux/ mac os)
* So Industries are moving from Virtualization application to containerization application.
* ****
* ****
* **1st) installing DOCKER ENGINE**
* **2nd) On the DE creating Containers (This container is similar to all 3)**
* **3rd) To Run this Container we r creating an IMAGE (include All ENV lib)**

****

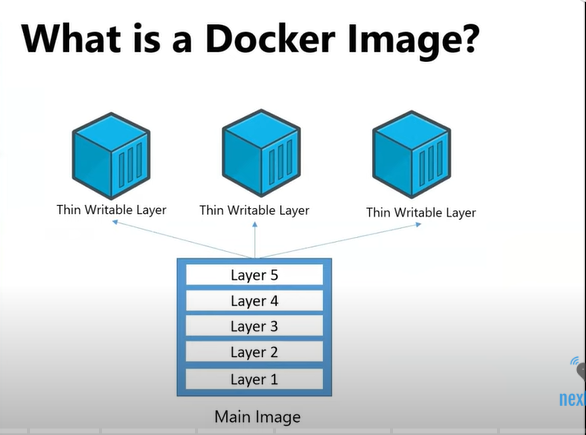
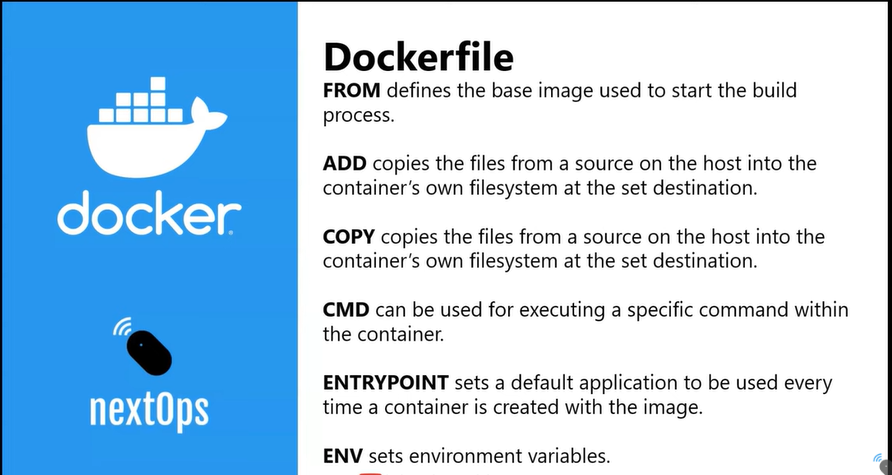
* **CONTAINER is** nothing but a machine.
* But in other machines (laptop/ server/ Virtual Machine) we need complete Operating System.
* But Containers has very light weight Operating system.
* CONTAINER will only have a required dependencies to run a specific application

**DOCKER IMAGE:**

****

* **Image** is nothing but a template which contains a set of instructions
* **With** one image we can create “n” No. of CONTAINERS
* And that image will become base
* 1st) docker pull imageName
* 2nd) after pulling image we have to create containers
* **Without image we cannot start the container**

**Difference b/w image and Container?**

* **Image is always read-only template.**
* **Image cannot start or RUN**
* **To use this image**
* **We create a container from this image**
* **After creating the container**
* ****
* **Then docker will create a writeable layer inside a container**
* **During RUN time of a container it will add writeable layer**
* **All the data of the container will store in writeable layer TEMPORARILY**
* **This writeable layer will available when CONTAINER is in running mode**
* **Inorder to store the data permanently we use DOCKER VOLUME**
* **VOLUME is nothing but persistent storage**
* **STEPS:**
* **1st) pull image from docker hub**
* **2nd) Image is in local system**
* **3rd) will Create CONTAINER using that image**
* 
* In **ADD -> you can give URL ALSO but COPY physical or destinatiom.**

 Docker components:

The three main [Docker components](https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker) are:

1. **Docker Client:**  Performs Docker build pull and run operations to open up communication with the Docker Host. The Docker command then employs Docker API to call any queries to run.
2. **Docker Host:** Contains Docker daemon, containers, and associated images. The Docker daemon establishes a connection with the Registry. The stored images are the type of metadata dedicated to containerized applications.
3. **Registry:** The place where [Docker images](https://www.simplilearn.com/tutorials/docker-tutorial/docker-images) are stored. There are two of them, a public registry and a private one. [Docker Hub](https://www.simplilearn.com/tutorials/docker-tutorial/docker-hub) and Docker Cloud are two public registries available for use by anyone.

* **Docker is used to Run Services and Applications**

**Why we need Docker?**

**1st)** Compatibility with the underlying OS was an issue

**We** have to ensure that different services were compatible with the version of OS we were planning to use.

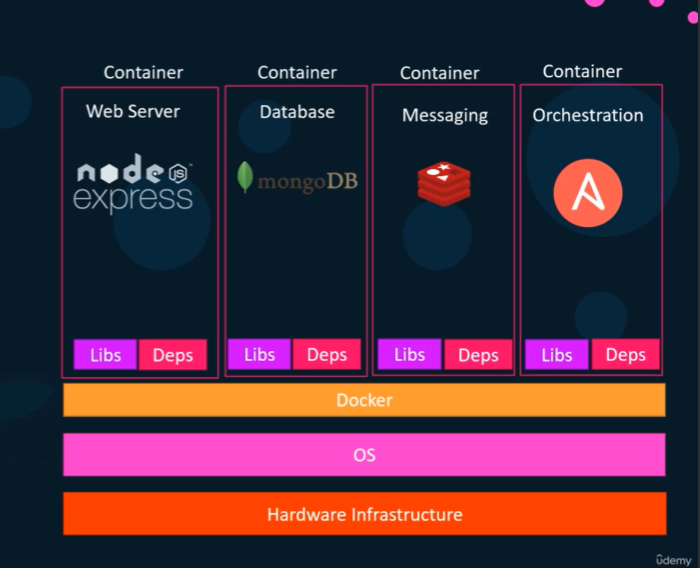
**2nd ) We** had to check the compatibility between these services and the libraries and libraries and dependencies on the OS

**We** will get issues when One service will require one version of library and another service will require another version of library

**3rd) Developer comfortable issue with OS**

**ANS) INORDER** to deal with this compatibility issue **WE** **CAME UP WITH A SOLUTION CALLED DOCKER**

**DOCKER:** With Docker we can Modify or change these components without affecting the other components and even modifying the underlying Operating System as Required.



* **With DOCKER we are able to run each component in a separate container with its own dependencies and its own libraries all are on the same VIRTUAL MACHINE and OS but within separate environments or containers**
* For this we had define our DOCKER Configuration Once Then Developer could now get started with a simple DOCKER run command.
* Irrespective of the OS they have. They just need to make sure that DOCKER has installed on their systems.
* **What are CONTAINERS?**

****

* **CONTAINERS** are completely isolated environments.
* **CONTAINERS** can have their own processes or services their own network interfaces their own mounts.
* Just like Virutal Machines they all share the same OS kernel.

**Different Types of Containers:**

**LXC, LXD, LXCFS etc..**

* **DOCKER** utilizes **LXC** Containers.
* **Main Purpose**  of DOCKER is to package and containerized applications and to ship them and to run them like anytime anywhere as many times as you want.

**OPERATING SYSTEM:**

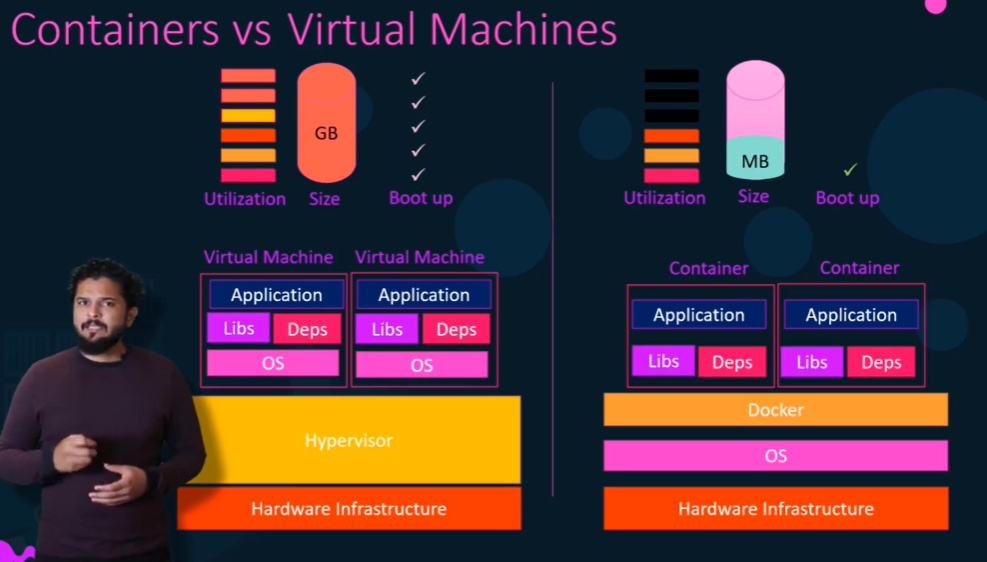
****

Operating System has two things :

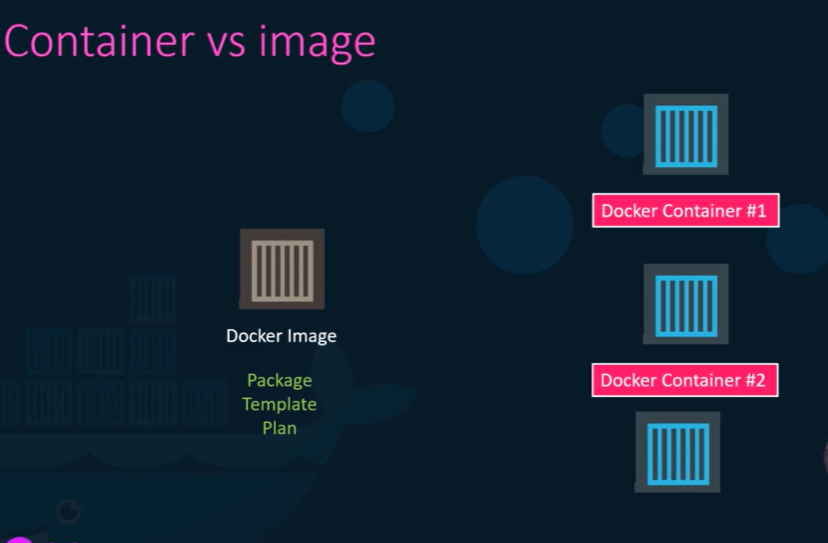
1st) OS Kernel

2nd) Set of software

**OS KERNEL:** OS Kernel is responsible for interacting with the underlying hardware while the OS kernel remains like Linux



**Container VS Image:**



* **Image :** Image is package or a template just like a VM template

Which is used to create one or more containers

* **CONTAINER:** Containers are running instances of images that are isolated and have their environments and set of processes
* **DOCKER IN DEVOPS:**

With DOCKER Developers and Ops team work together to transform the guide into a docker file with both of their requirements

* Then the DOCKER file is used to Create image for their applications
* This Image can now run on any host with DOCKER installed on it and guaranteed to run the same way
* So the OPS team can now simply use the image to deploy the application since the image was already working it works in the same way after deployed the application

**DOCKER OPERATIONS:**

DOCKER COMMANDS :

* Docker –version
* Sudo docker version
* docker login
* docker search imageName
* docker inspect image imageName

***CONATAINER OPERATIONS:***

* RUN – start a Container 🡪 docker run nginx

**OR**

* **docker create imageName**
* ps – list Containers 🡪 docker ps
* ps – with list of running containers 🡪 docker ps -a
* **docker start containerName**
* STOP – stop a Container 🡪 docker stop containerName
* Rm – Remove a Container 🡪 docker rm containerName

**Create CONTAINERS with customName**

* **docker create --name customName imageName**
* **docker start containerName**
* **docker stop containerName**
* **docker kill containerName**

**OR**

* **docker run -d --name customName imageName**
* docker exec -it containerName /bin/bash

(/bin/bash == shellName , -it 🡪 interactive Terminal)

**\*\*\*\* docker rm $(docker ps -a -q)**

* docker ps -a # lts like git status
* **images – To see List images** 🡪 docker images or docker image ls
* **Rmi – Remove Images** 🡪 docker rmi imageName
* Note : Delete all dependent Containers to remove image
* **Pull** – download an image 🡪 docker run imageName
* Note: Inorder to just pull the image and Not to run the container
* **Pull – Only image Not to run Container** 🡪 docker pull imageName
* Append a Command

**Pause a Container** 🡪 docker run ubuntu 🡪 docker run ubuntu sleep 5

* Executing a command on a running Container:

**Exec – execute a Command 🡪** docker exec containerName filename

* **Run – attach and detach** 🡪 docker run kodekloud/simple-webapp
* Write CTRL + C to quit

**Detach mode** 🡪 docker run -d kodekloud/simple-webapp

Attaching smtng to Running Container :

🡪 docker attach containerName/id

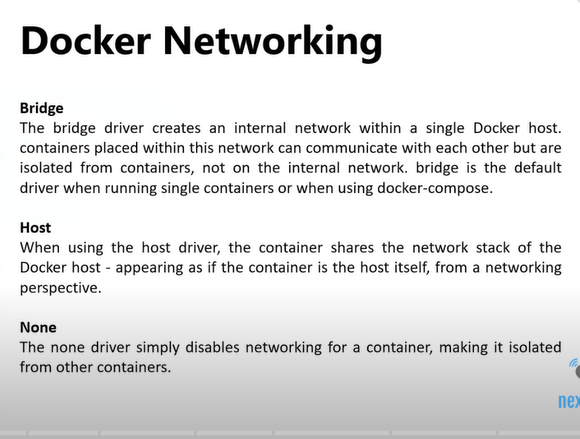
* DOCKER RUN :
* Run – tag
* Docker run redis
* docker run redis:4.0 (4.0 🡪 tag)
* Run - STDIN
* docker run kodekloud/simple-prompt-docker
* docker run -i kodekloud/simple-prompt-docker
* **docker run -it kodekloud/simple-prompt-docker** ( -it -> pseudo Terminal )

**DOCKER NETWORKING:**

**Network : collection of devices**

**Networking: Basically One or more devices communicating with each other is called networking.**

* **3 Types of networking are there**
* **1st) BRIDGE : default network**
* **2nd) Host**
* **3rd) NONE**

****

* **docker network ls**
* **docker run -d --name customName --net host imageName**
* **docker run -d --name customName --net none imageName**

**DOCKER STORAGE:**

**VOLUMES:**

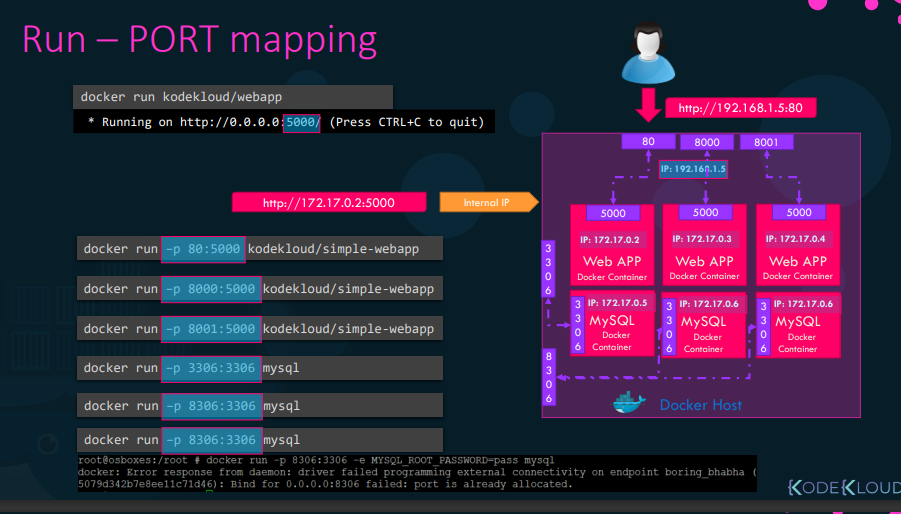
* **dockor images**
* **docker network ls**
* **docker volume ls**
* **docker volumeName/Id**
* **cd /var/lib/docker/volumes/**
* **cd volumeId**

**What is the default locations for docker volumes?**

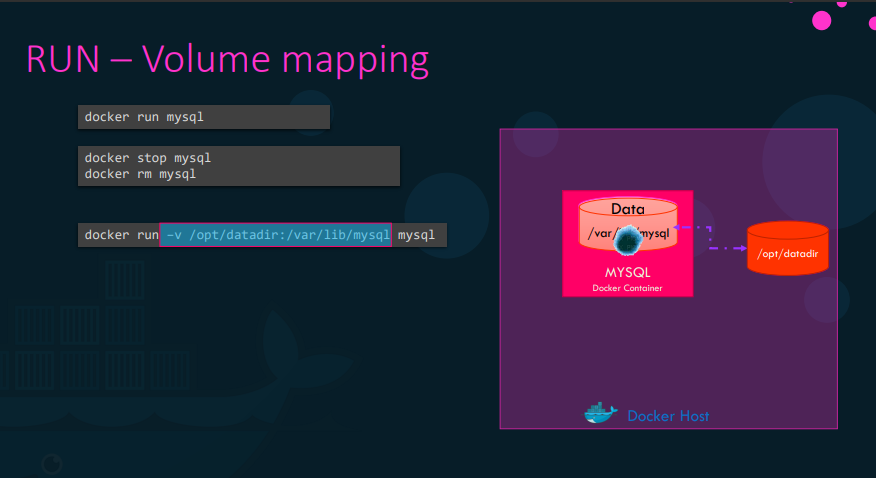
* **var ( /var/lib/docker/volumes)**
* **docker volume rm volumeId**
* **docker volume prune**
* **docker volume ls**
* **docker volume create volumeName**
* **docker volume ls**
* **docker inspect volume volumeName**
* **docker run -it --name=customName --mount source=testvol, destination=/data imageName**
* **ls**
* **touch testfile.txt**
* **echo “testdata”>> testfile.txt**
* **cat testfile.txt**
* **exit**
* **cd /volumepath/…/….**
* **Ls**

**DOCKER HOST: The Underlying host where the docker is installed is called Docker Host or Docker engine**

**NOTE: DOCKER Container can be accessible through the docker container INTERNAL IP but only can accessible in the DOCKER HOST**

****

* **docker run -p 80:5000 kodekloud/webapp**

****

* RUN – Volume Mapping
* docker run -v /opt/datadir **:** /var/lib/mysql msql

(created a directory called /opt/datadir and mapped to /var/lib/mysql inside the docker container using -v option)

* **inpsect Container 🡪 docker inspect containerName**
* **Container Logs 🡪 docker logs containerName**
* **Mapping port to DOCKER Host 🡪** docker run -p 8080:8080 containerName
* **Port Mapping 🡪** docker run -p 8080:8080 containerName

**Volume Mapping :**

* Docker run –name myjenkins -p 8080:8080 -p 50000:50000 -v /var/Jenkins\_home Jenkins
* **HOW To Create my own Image?**
* **1) OS – Ubuntu**
* **2) Update apt repo**
* **3) Install dependencies using apt**
* **4) Install Python dependencies using pip**
* **5) Copy source code to /opt folder**
* **6) Run the web server using “flask” command**

**ACTUAL COMMAND IMPLEMENTATION:**

* **Dockerfile**
* **INTRUCTION ARGUMENT**
* **FROM Ubuntu**
* **Run apt-get update && apt-get -y install python**
* **COPY . /opt/source-code**
* **ENTRYPOINT FLASK\_APP=/opt/source-code/app.py flask run**
* **(building the image using Docker)**
* **Docker build . -f Dockerfile -t mmumshad/my-custom-app**

**(above will create image locally on your system)**

**🡪( Now To make it available on the public DockerHub registry use**

**Push command )**

* **Docker push mmumshad/my-custom-app**
* **Creating a new Docker Image**
* Docker run -it ubuntu bash
* apt-get install -y python
* apt-get update ### (updating the package index)
* apt-get install -y python
* python
* exit()
* pip install flask
* apt-get install python pip
* cat > Dockerfile (We are creating a Dockerfile)
* **FROM ubuntu**
* **RUN apt-get update**
* **RUN apt-get install -y python python-pip**
* **RUN pip install flask**
* **COPY app.py /opt/app.py**
* **ENTRYPOINT FLASK\_APP=/opt/app.py flask run –host=0.0.0.0**
* **docker build .**
* **docker images**
* **docker run imageName**
* **docker push imageName ( pushing into Dockerhub public repository)**
* **docker build . -t accountName/applicationName**

Run an instance of the image webapp-color and publish port 8080 on the container to 8282 on the host.

CheckCompleteIncomplete

* Container with image 'webapp-color'
* Container Port: 8080
* Host Port: 8282
* **ANS) docker run -d -p 8282:8080 webapp-color**

**Build a docker image using the Dockerfile and name it webapp-color. No tag to be specified.**

**$ cd /root/webapp-color/**

**$ pwd**

**/root/webapp-color**

**$ docker build -t webapp-color .**

**Sending build context to Docker daemon 125.4kB**

**Step 1/6 : FROM python:3.6**

**3.6: Pulling from library/python**

**0e29546d541c: Pull complete**

**9b829c73b52b: Pull complete**

**cb5b7ae36172: Pull complete**

**6494e4811622: Pull complete**

**6f9f74896dfa: Pull complete**

**5e3b1213efc5: Pull complete**

**9fddfdc56334: Pull complete**

**404f02044bac: Pull complete**

**c4f42be2be53: Pull complete**

**Digest: sha256:f8652afaf88c25f0d22354d547d892591067aa4026a7fa9a6819df9f300af6fc**

**Status: Downloaded newer image for python:3.6**

**---> 54260638d07c**

**Step 2/6 : RUN pip install flask**

**---> Running in d263843f6272**

**Collecting flask**

**Downloading Flask-2.0.3-py3-none-any.whl (95 kB)**

**Collecting itsdangerous>=2.0**

**Downloading itsdangerous-2.0.1-py3-none-any.whl (18 kB)**

**Collecting Jinja2>=3.0**

**Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)**

**Collecting Werkzeug>=2.0**

**Downloading Werkzeug-2.0.3-py3-none-any.whl (289 kB)**

**Collecting click>=7.1.2**

**Downloading click-8.0.4-py3-none-any.whl (97 kB)**

**Collecting importlib-metadata**

**Downloading importlib\_metadata-4.8.3-py3-none-any.whl (17 kB)**

**Collecting MarkupSafe>=2.0**

**Downloading MarkupSafe-2.0.1-cp36-cp36m-manylinux\_2\_5\_x86\_64.manylinux1\_x86\_64.manylinux\_2\_12\_x86\_64.manylinux2010\_x86\_64.whl (30 kB)**

**Collecting dataclasses**

**Downloading dataclasses-0.8-py3-none-any.whl (19 kB)**

**Collecting typing-extensions>=3.6.4**

**Downloading typing\_extensions-4.1.1-py3-none-any.whl (26 kB)**

**Collecting zipp>=0.5**

**Downloading zipp-3.6.0-py3-none-any.whl (5.3 kB)**

**Installing collected packages: zipp, typing-extensions, MarkupSafe, importlib-metadata, dataclasses, Werkzeug, Jinja2, itsdangerous, click, flask**

**Successfully installed Jinja2-3.0.3 MarkupSafe-2.0.1 Werkzeug-2.0.3 click-8.0.4 dataclasses-0.8 flask-2.0.3 importlib-metadata-4.8.3 itsdangerous-2.0.1 typing-extensions-4.1.1 zipp-3.6.0**

**WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv**

**WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.**

**You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.**

**Removing intermediate container d263843f6272**

**---> 6bab4188096f**

**Step 3/6 : COPY . /opt/**

**---> 7ef25fa33ab1**

**Step 4/6 : EXPOSE 8080**

**---> Running in ae99db11d77d**

**Removing intermediate container ae99db11d77d**

**---> 7c23c2e8c86f**

**Step 5/6 : WORKDIR /opt**

**---> Running in 9948e7040968**

**Removing intermediate container 9948e7040968**

**---> 2beb4e989a27**

**Step 6/6 : ENTRYPOINT ["python", "app.py"]**

**---> Running in 142e80eaab47**

**Removing intermediate container 142e80eaab47**

**---> adcc2788ddaf**

**Successfully built adcc2788ddaf**

**Successfully tagged webapp-color:latest**

**$**

**ENVIRONMENTAL VARIABLES:**

* **ENV Variables in Docker**
* Docker run -e APP\_COLOR=blue simple-webapp-color
* docker run -e APP\_COLOR=green simple-webapp-color
* docker run -e APP\_COLOR=pink simple-webapp-color
* **How can you find the environmental Variable set on a container that’s already running?**

**ANS)** Use the inspect command to know the properties of a running container

* **docker inspect containerName**

**(** In the config : { } ull find the environmental variables**)**

**How to inspect the environmental variables?**

* **docker inspect -f '{{range .Config.Env}}{{println .}}{{end}}' <container\_id\_or\_name> | grep APP\_COLOR**
* **> docker inspect -f '{{range .Config.Env}}{{println .}}{{end}}' <container\_id\_or\_name>: This part of the command inspects the container and extracts all the environment variables.**
* **| grep APP\_COLOR: This part of the command uses grep to filter the output and display only the line that contains the APP\_COLOR environment variable and its value.**

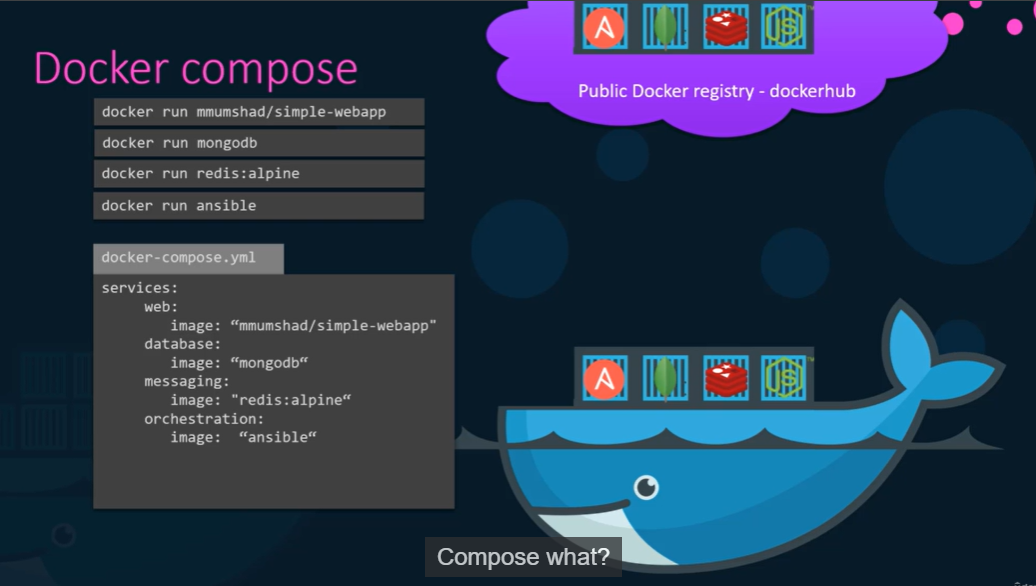
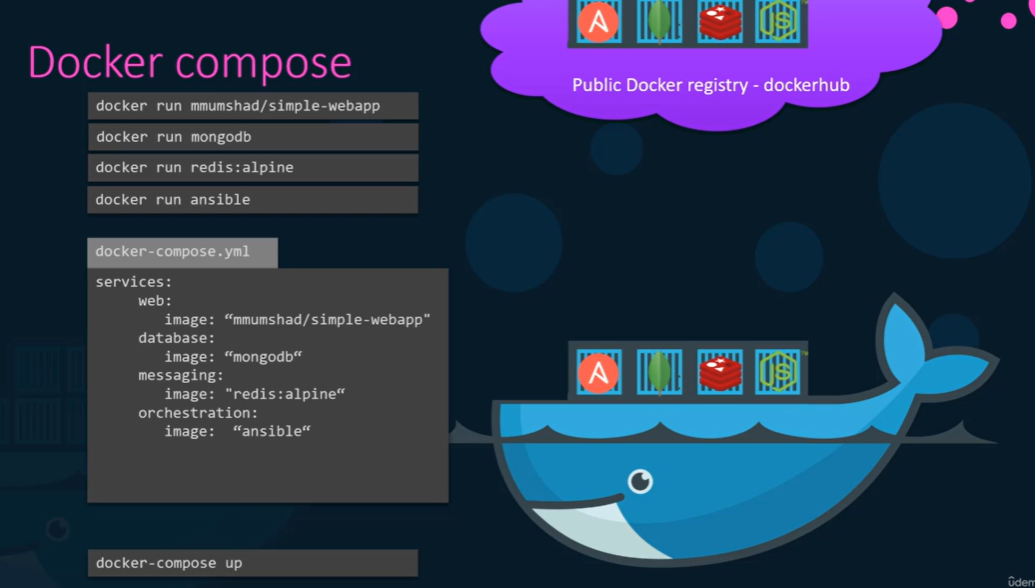
**COMMAND VS ENTRYPOINT**

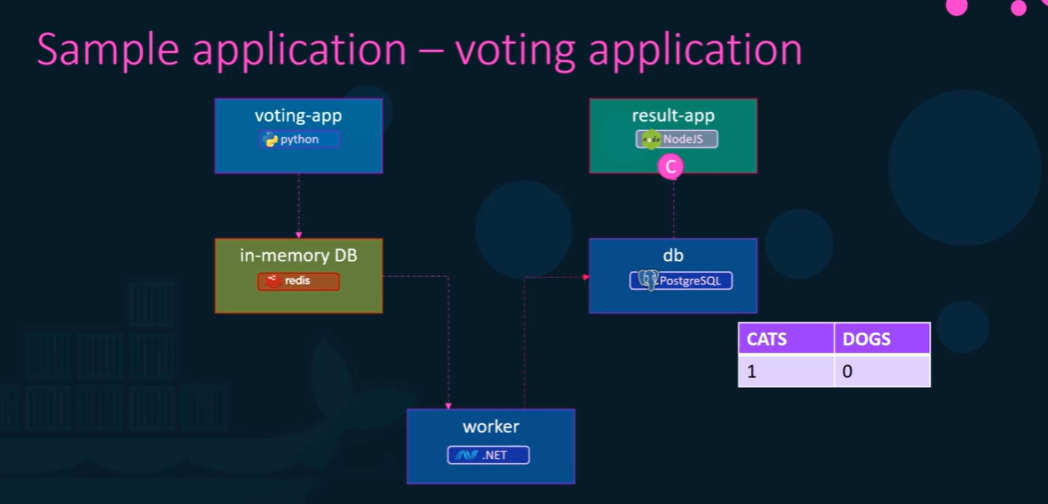
* **Unlike virtual machines containers are meant to run the Operating Systems.**
* **CONTAINERS are meant to run a specific task or process**
* **Once Task is completed Containers will exit**
* **CONTAINER will only alive as long as the process takes place.**
* docker run ubuntu [ COMMAND]
* docker run ubuntu sleep 5

Creating OWN image from the base ubuntu image

* FROM Ubuntu
* CMD sleep 5 🡪 docker run ubuntu-sleeper sleep 10
* **FROM Ubuntu**
* **ENTRYPOINT [“sleep”] 🡪 docker run ubuntu-sleeper 10**
* **COMMAND AND ENTRYPOINT are better to represent in JSON FORMAT**
* **FROM Ubuntu**
* **ENTRYPOINT [“sleep”]**
* **CMD [“5”]**

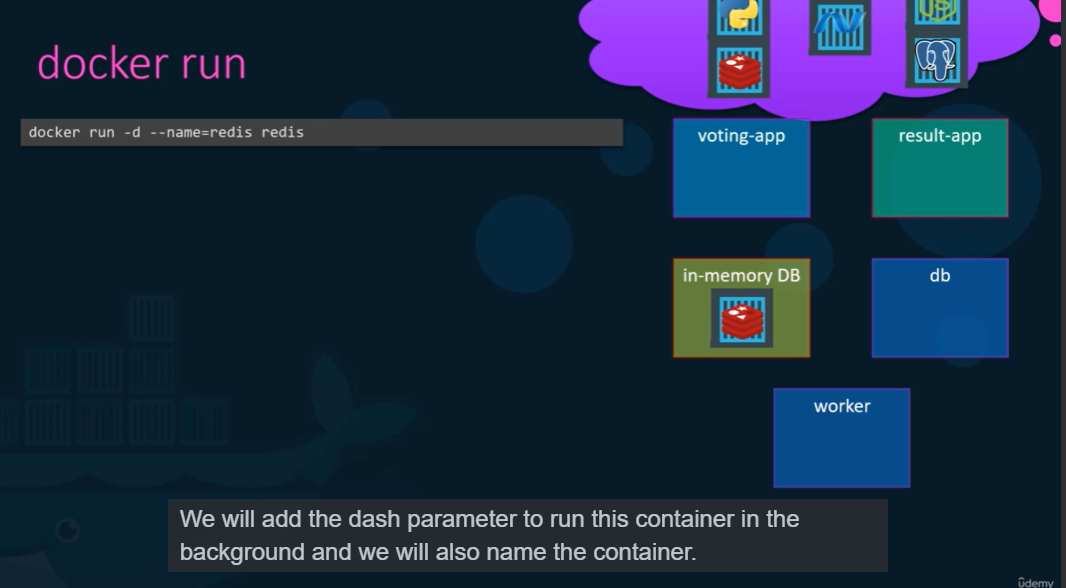
**DOCKER COMPOSE**

* ****
* If we needed to setup a complex application running multiple services then its better to use DOCKER COMPOSE
* We create a Configuration file called dockern-compose.yml
* ****
* **docker-compose up 🡪 will bring entire application stack**

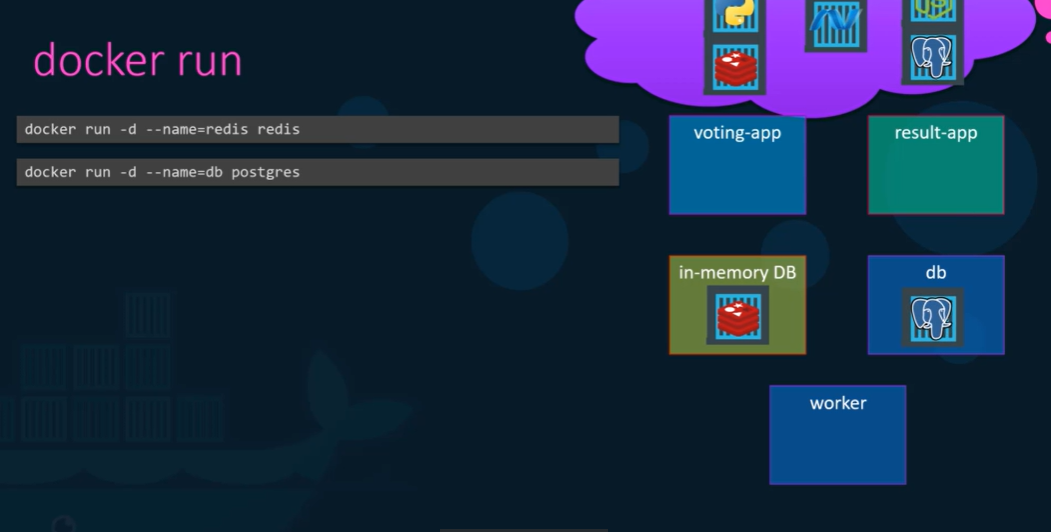
****

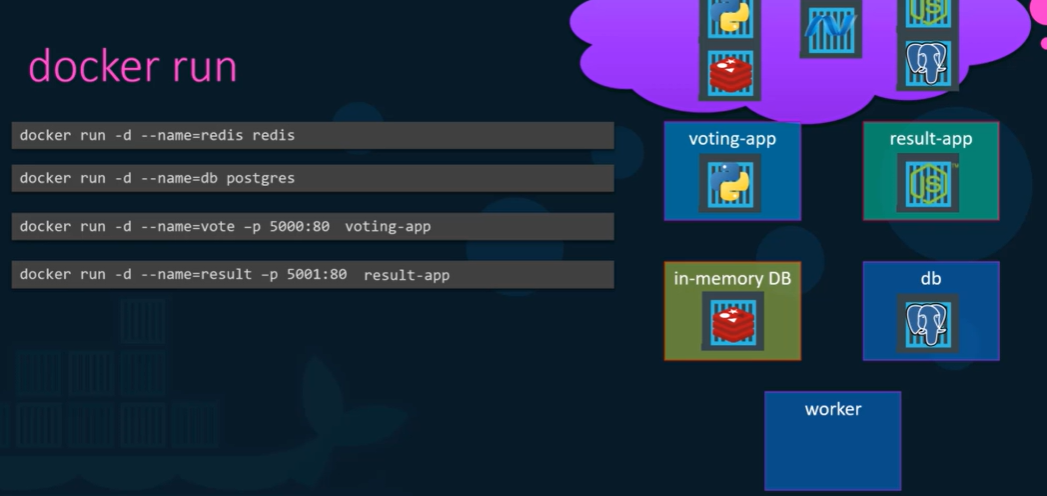
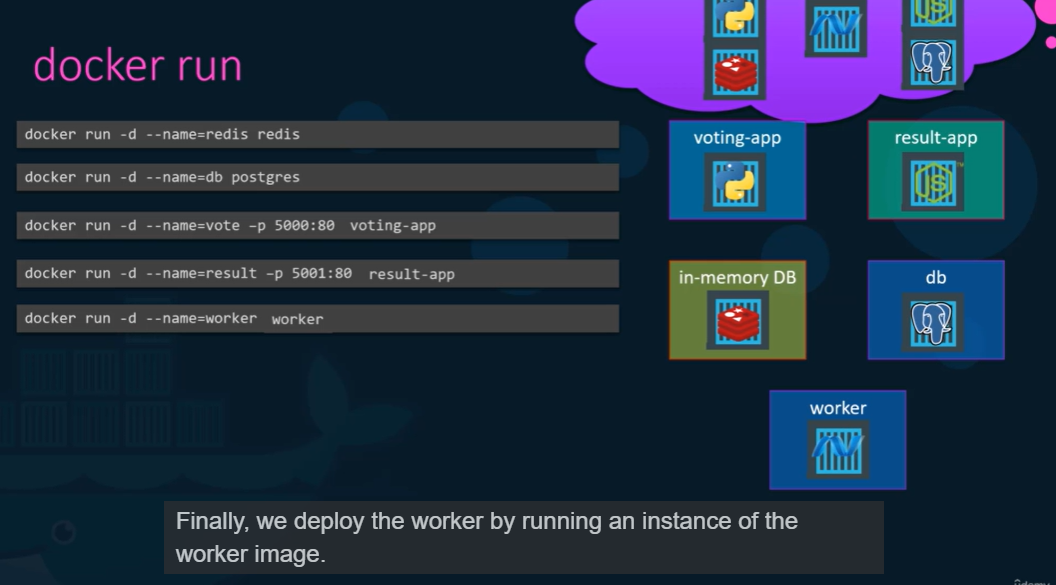
* **We trying to run this whole application stack on a single Docker engine using….**

**First docker run commands and then Docker Compose**



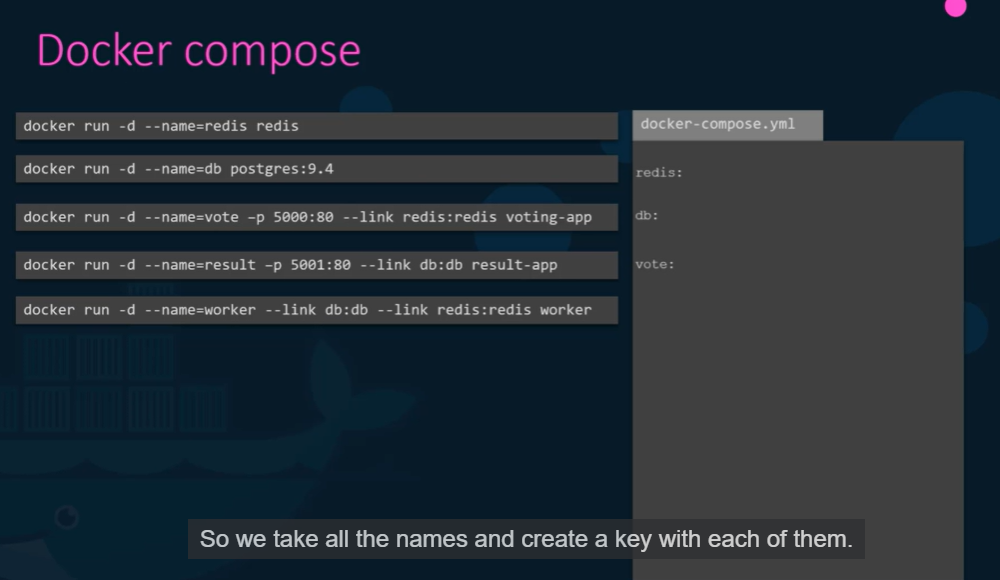
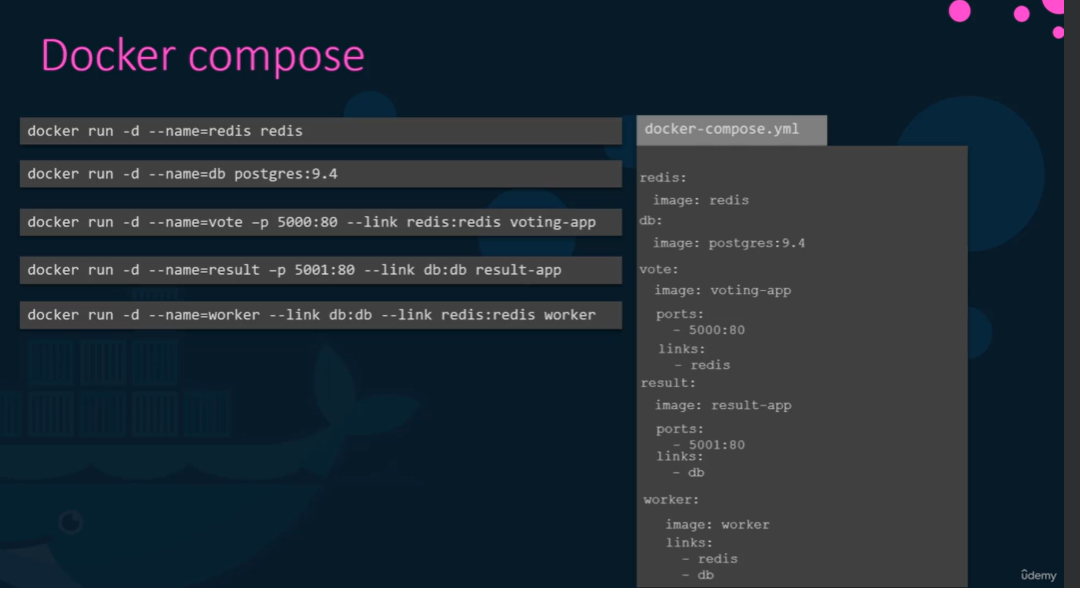
* **docker run -d –name=radis redis**

****

* **docker run -d –name=db postgres**
* ****
* **docker run -d –name=vote -p 5000:80 voting-app ( # 80: port , 5000: host system)**
* ****
* **docker run -d –name=result -p 5001:80 result-app**
* ****
* **docker run -d –name=worker worker**

**Links : Link is a command line option which can be used to link Container together**

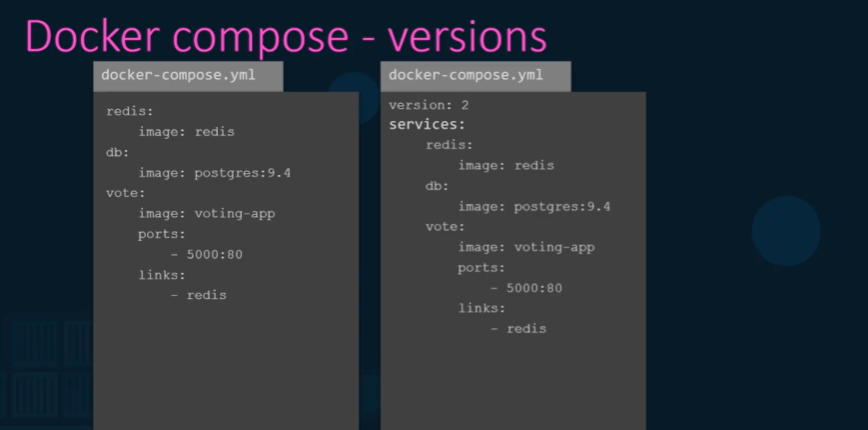
****

* **we making the voting app to aware of redis app using link.**
* ****
* **The Key is the image and value is name of the image**
* ****
* **docker-compose up 🡪 will bring up the docker-compose.yml file**

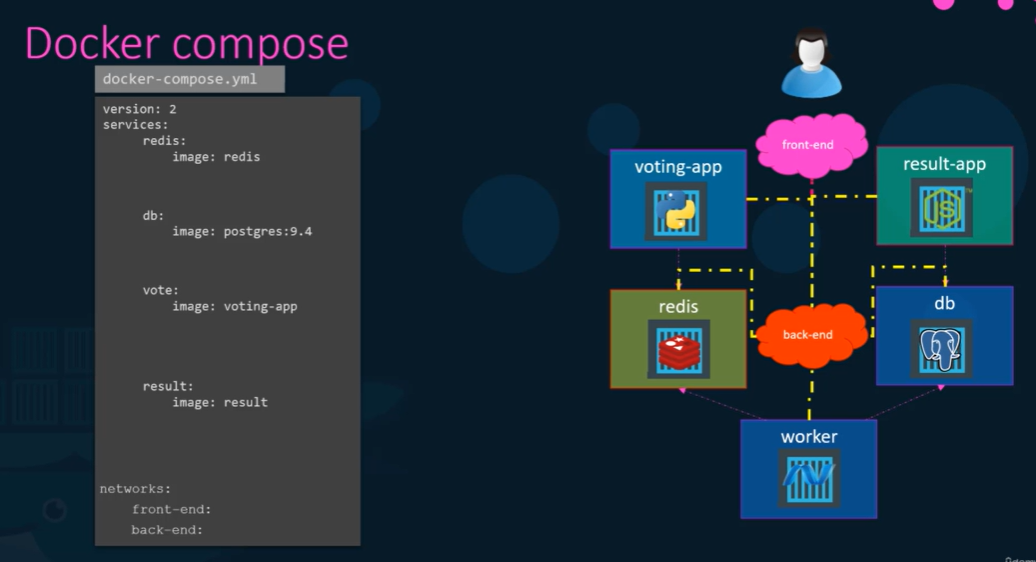
**DOCKER COMPOSE FILE – VERSIONS**

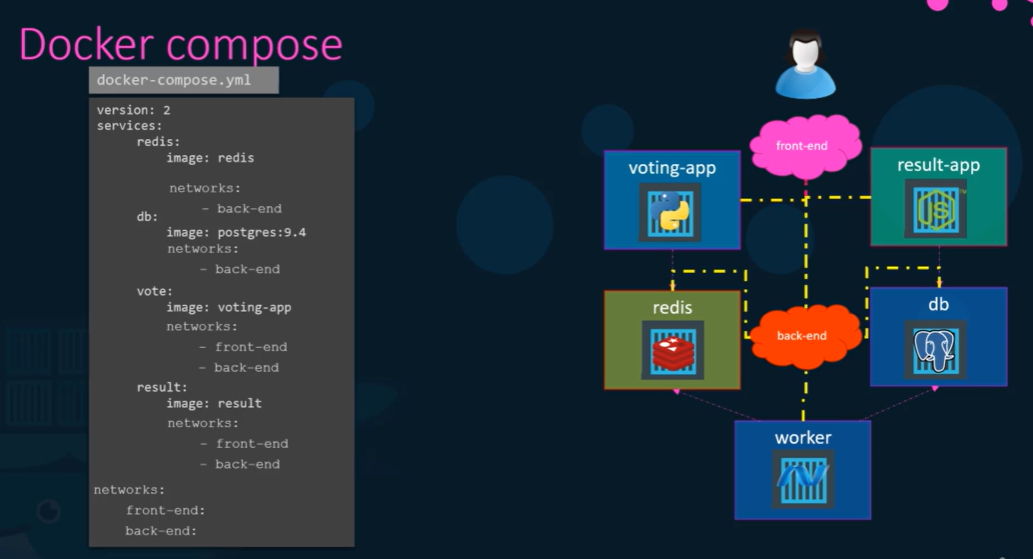
* ****

****

****

* **We need to specify the version at the top of the file**
* **All Containers are communicate to each other using each others service name**
* **So there is noo need of links in version 2.**
* ****

****

****

**Github link:**

* **git clone** [**https://github.com/dockersamples/example-voting-app.git**](https://github.com/dockersamples/example-voting-app.git)
* **Ls**
* **cd vote/**
* **ls**
* **cat Dockerfile**
* **ls**
* **docker images**
* **docker run -p 5000:80 voting-app**
* **docker run -d –name=redis redis**
* **docker ps**
* **docker run -p 5000:80 –link redis:redis voting-app**
* **docker run -d –name=db postgres:9.4**
* **docker images**
* **docker run –link redis:redis --link db:db worker-app**
* **docker ps**
* **cd sample-application/**
* **ls**
* **cd result/**
* **ls**
* **ls -l**
* **docker build . -t result-app**
* **docker images**
* **docker run -p 5001:80 –link db:db result-app**

**With Docker Compose**

* **docker ps**
* **docker stop 34 45 656 78**
* **> docker ps**
* **cat > docker-compose.yml**
* **redis:**

**db:**

**vote:**

**worker:**

**result:**

* **vi docker-composer.yml**

**version: “3”**

**services:**

**redis:**

**image: redis**

**db:**

**image: postgres:9.4**

**environment:**

**POSTGRES\_USER: postgres**

**POSTGRES\_PASSWORD: postgres**

**vote:**

**image: voting-app**

**ports:**

**-5000:80**

**Links:**

* + - **db**
    - **redis**

**Worker:**

**Image: worker-app**

**result:**

**image: result-app**

**ports:**

**-5001:80**

**Links:**

* + - **db**
* **docker ps**
* **docker-compose up**

**DOCKER REGISTRY**

* **CONTAINERS == RAIN then REGISTRY == CLOUD**
* **Images are stored in the Registry**
* **It’s a central Repository for all the Docker Images**
* **Image :**
* **Image: nginx (nginx == image/repository)**

**User/Account**

* **Image: docker.io/nginx/nginx**

**Registry image/Repository**

* **PRIVATE REGISTRY:**

Docker login private-registry.io

* **Calling an image from the private Registry:**

Docker run private-registry.io/apps/internal-app

* **How to Deploy private Registry:**

docker run -d -p 5000:5000 –name registry registry:2

* **How to push an image to the private Registry:**

docker image tag my-image localhost:5000/my-image

* docker push localhost:5000/imageName
* **How to pull an image to the private Registry:**

**docker pull localhost:5000/imageName**

**or**

**docker pull 192.168.56.100:5000/imageName**

**DOCKER ENGINE**

* **Docker engine: is referred to host with Docker installed on it.**
* **What is Docker DEAMON?**

Docker Deamon is the background process that manages Docker objcets such as the images containers Volumes and Networkds

* **What is DOCKER REST API SERVER?**

Docker Rest API Server is the API interface that programs can use to talk to the deamon and provides instructions

* You can create your own tools using REST API
* **What is DOCKER CLI?**

Docker command line interface is used perform actions such as running a container stopping a container

**docker ps -eaf 🡪 to see list of processes inside a container**

**Docker STORAGE:**

* **Volumes: docker volume create data\_volume**
* **docker run -v data\_volume:/var/lib/mysql mysql**
* **docker run -v data\_volume2:/var/lib/mysql mysql**

**( This is called Volume Mounting)**

**STORAGE DRIVERS:**

* **AUFS**
* **ZFS**
* **BTRFS**
* **DEVICE MAPPER**
* **OVERLAY**
* **OVERLAY2**

**DOCKER VOLUMES**

* **docker images**
* **docker ps -a**
* **docker network ls**
* **docker volume ls**
* **cd /var/lib/docker/volumes/volumeId**
* **ls**
* **cd volumeId**
* **ls**
* **cd \_data**
* **ls**
* **Default LOCATIONS FOR DOCKER VOLUME?**

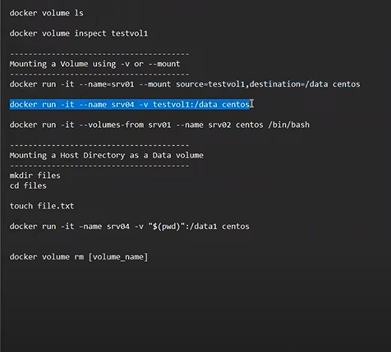
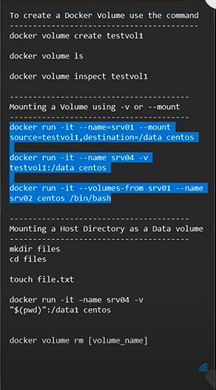
**/var/lib/docker/volumes/**

* **Docker rm $(docker ps -a -q)**
* **docker ps -a**
* **docker volume rm volumeId**
* **docker volume prune**
* **docker volume rm volumeId**

**How to CREATE VOLUME?**

* **docker volume create testvol1**

**HOW TO MAP A VOLUME TO THE CONTAINER?**

* **docker run -it –name=kk –mount source=testvol1, destination=/data centos**
* **docker ps**
* **docker ps -a**
* **ls**
* **cd data/**
* **touch testfile.txt**
* **echo “testdata”>> testfile.txt**
* **cat testfile.txt**
* **docker ps -a**
* **ls**
* **cat testfile.txt**
* **docker rm srv01**
* **docker volume ls**
* **ls**
* **cat testfile.txt**
* **cd ..**
* **cd ..**
* **cd ..**
* **docker run -it –name-srv02 –mount source=testvol1, destination=/data centos**
* **ls**
* **cd data/**
* **ls**
* ****
* ****
* **Volume is nothing but absolute path**
* **docker run -it –name=srv05 -v “$(pwd)”:/data1 centos**

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

**What is the command run at startup when the ubuntu image is run?**

**#**

**# Ubuntu Dockerfile**

**#**

**# https://github.com/dockerfile/ubuntu**

**#**

**# Pull base image.**

**FROM ubuntu:14.04**

**# Install.**

**RUN \**

**sed -i 's/# \(.\*multiverse$\)/\1/g' /etc/apt/sources.list && \**

**apt-get update && \**

**apt-get -y upgrade && \**

**apt-get install -y build-essential && \**

**apt-get install -y software-properties-common && \**

**apt-get install -y byobu curl git htop man unzip vim wget && \**

**rm -rf /var/lib/apt/lists/\***

**# Add files.**

**ADD root/.bashrc /root/.bashrc**

**ADD root/.gitconfig /root/.gitconfig**

**ADD root/.scripts /root/.scripts**

**$ vi /root/Dockerfile-ubuntu**

**Now its time to push some images to our registry server. Let's push two images for now .i.e. nginx:latest and httpd:latest.**

**Note: Don't forget to pull them first.**

**To check the list of images pushed , use curl -X GET localhost:5000/v2/\_catalog**

**$ docker pull nginx:latest httpd:latest**

**"docker pull" requires exactly 1 argument.**

**See 'docker pull --help'.**

**Usage: docker pull [OPTIONS] NAME[:TAG|@DIGEST]**

**Pull an image or a repository from a registry**

**$ docker pull httpd:latest**

**latest: Pulling from library/httpd**

**578acb154839: Pull complete**

**c1a8c8567b78: Pull complete**

**10b9ab03bf45: Pull complete**

**74dbedf7ddc0: Pull complete**

**6a3b76b70f73: Pull complete**

**Digest: sha256:4e24356b4b0aa7a961e7dfb9e1e5025ca3874c532fa5d999f13f8fc33c09d1b7**

**Status: Downloaded newer image for httpd:latest**

**docker.io/library/httpd:latest**

**$ docker pull nginx:latest**

**\latest: Pulling from library/nginx**

**578acb154839: Already exists**

**e398db710407: Pull complete**

**85c41ebe6d66: Pull complete**

**7170a263b582: Pull complete**

**8f28d06e2e2e: Pull complete**

**6f837de2f887: Pull complete**

**c1dfc7e1671e: Pull complete**

**Digest: sha256:86e53c4c16a6a276b204b0fd3a8143d86547c967dc8258b3d47c3a21bb68d3c6**

**Status: Downloaded newer image for nginx:latest**

**docker.io/library/nginx:latest**

**$ docker image tag nginx:latest localhost:5000/nginx:latest**

**$ docker push localhost:5000/nginx:latest**

**The push refers to repository [localhost:5000/nginx]**

**Get "http://localhost:5000/v2/": dial tcp 127.0.0.1:5000: connect: connection refused**

**$ docker image tag nginx:latest nginx:latest**

**$ docker image tag httpd:latest localhost:5000/httpd:latest**

**$ httpd:latest localhost:5000/httpd:latest**

**-bash: httpd:latest: command not found**

**$ docker push localhost:5000/httpd:latest**

**The push refers to repository [localhost:5000/httpd]**

**Get "http://localhost:5000/v2/": dial tcp 127.0.0.1:5000: connect: connection refused**

**$**

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

**What is the command run at startup when the ubuntu image is run?**

**# Install.**

**RUN \**

**sed -i 's/# \(.\*multiverse$\)/\1/g' /etc/apt/sources.list && \**

**apt-get update && \**

**apt-get -y upgrade && \**

**apt-get install -y build-essential && \**

**apt-get install -y software-properties-common && \**

**apt-get install -y byobu curl git htop man unzip vim wget && \**

**rm -rf /var/lib/apt/lists/\***

**# Add files.**

**ADD root/.bashrc /root/.bashrc**

**ADD root/.gitconfig /root/.gitconfig**

**ADD root/.scripts /root/.scripts**

**# Set environment variables.**

**ENV HOME /root**

**# Define working directory.**

**WORKDIR /root**

**# Define default command.**

**CMD ["bash"]**

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

**Create a docker-compose.yml file under the directory /root/clickcounter. Once done, run docker-compose up.**

**The compose file should have the exact specification as follows -**

**redis service specification - Image name should be redis:alpine.**

**clickcounter service specification - Image name should be kodekloud/click-counter, app is run on port 5000 and expose it on the host port 8085 in the compose file.**

**$ cd /root/clickcounter**

**$ touch docker-compose.yml**

**$ vi docker-compose.yml**

**$ docker-compose up -d**

**Creating network "clickcounter\_default" with the default driver**

**Creating clickcounter\_clickcounter\_1 ... done**

**Creating clickcounter\_redis\_1 ... done**

**$ ^C**

**$**

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