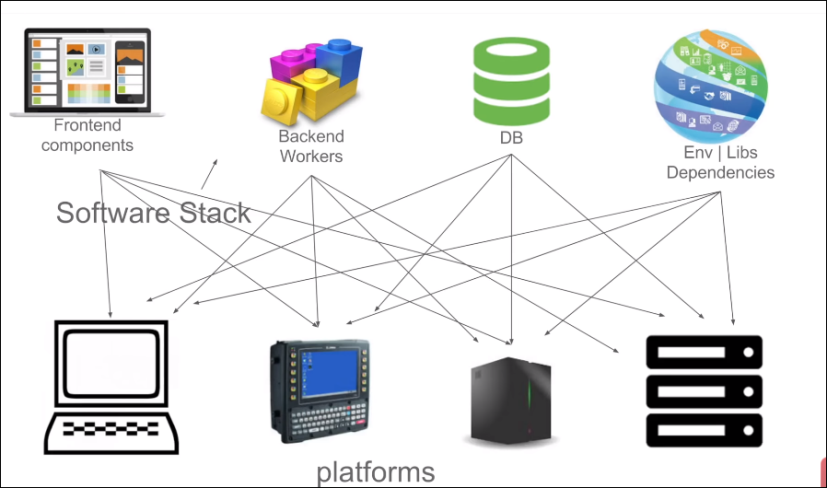
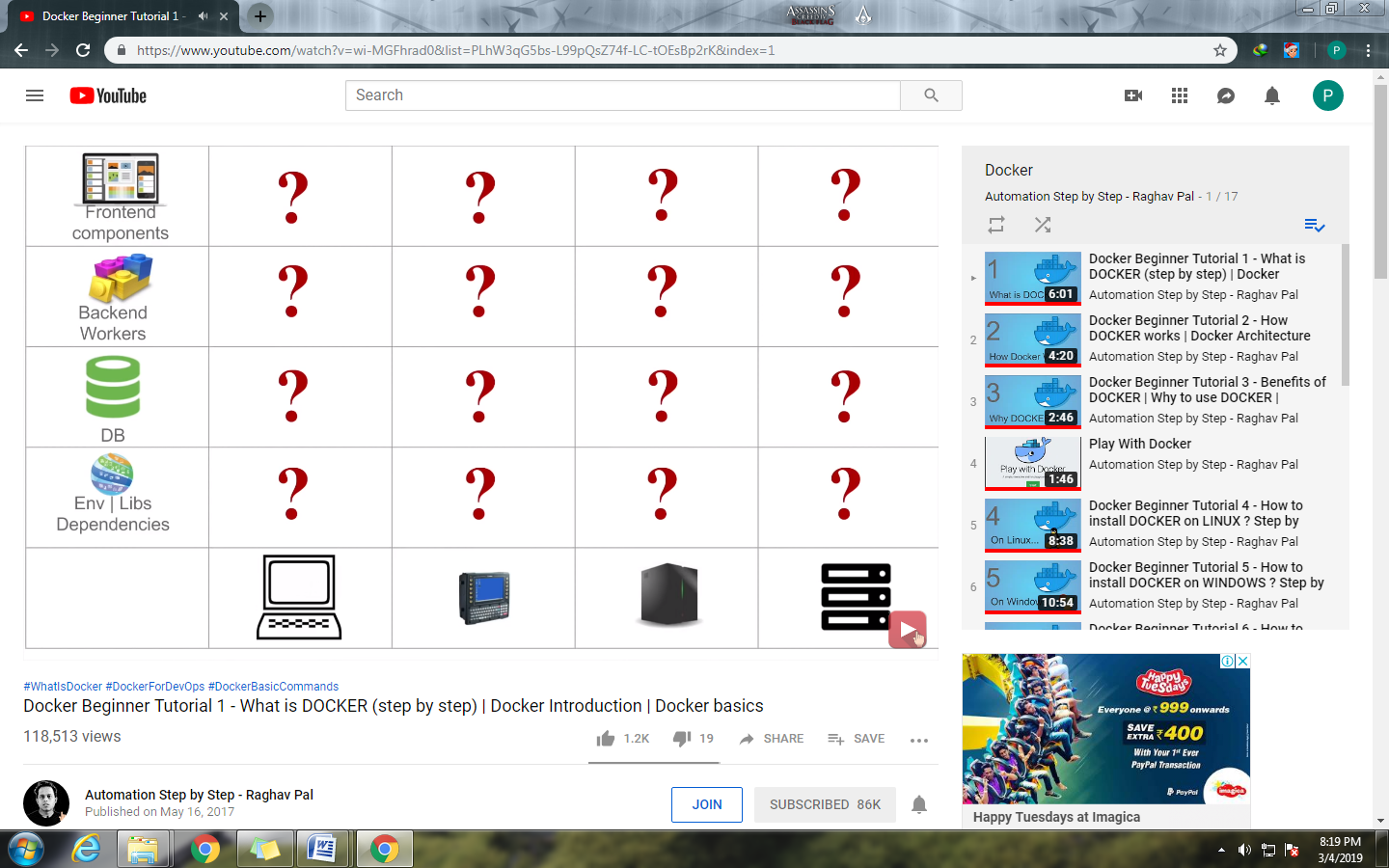
**Docker**

**Introduction:**

Docker Comes into picture at deployment stage, Docker makes the process of application deployment very easy and efficient and resolves lot of issues related to deploying applications

**Problem statement for docker:**

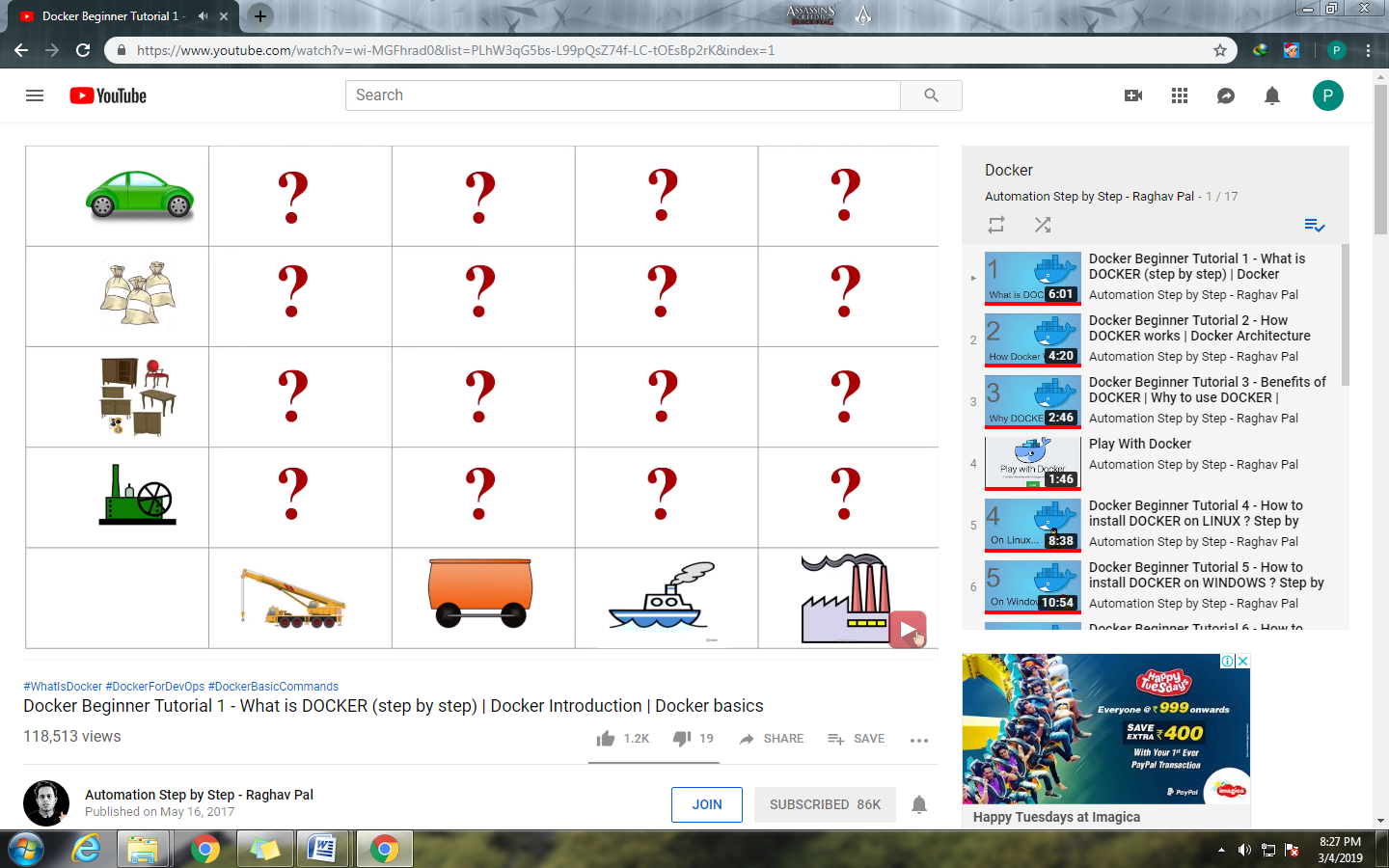
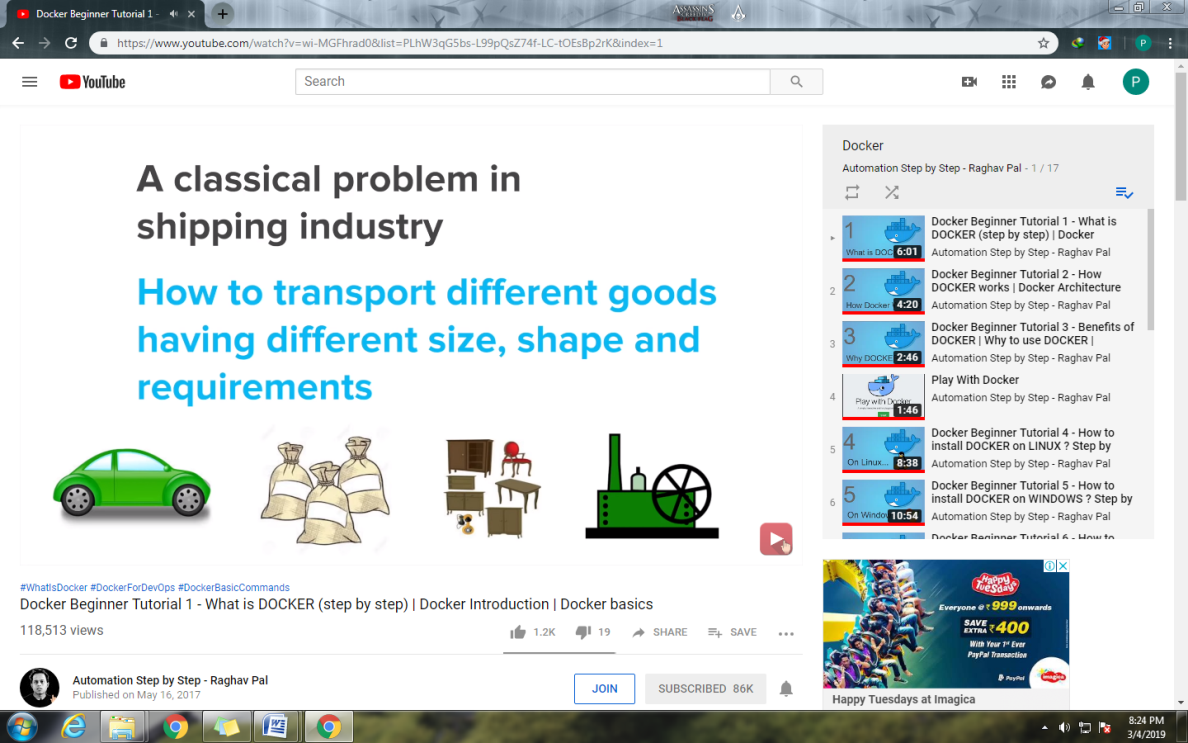
A typical software application consist of frontend component, back end, databases env, libs/dependencies. we have to make sure that these component should run on different types of platforms like desktop, laptop etc. It looks like very complicated structure like below, and it is very difficult to ensure software application run on each platform.

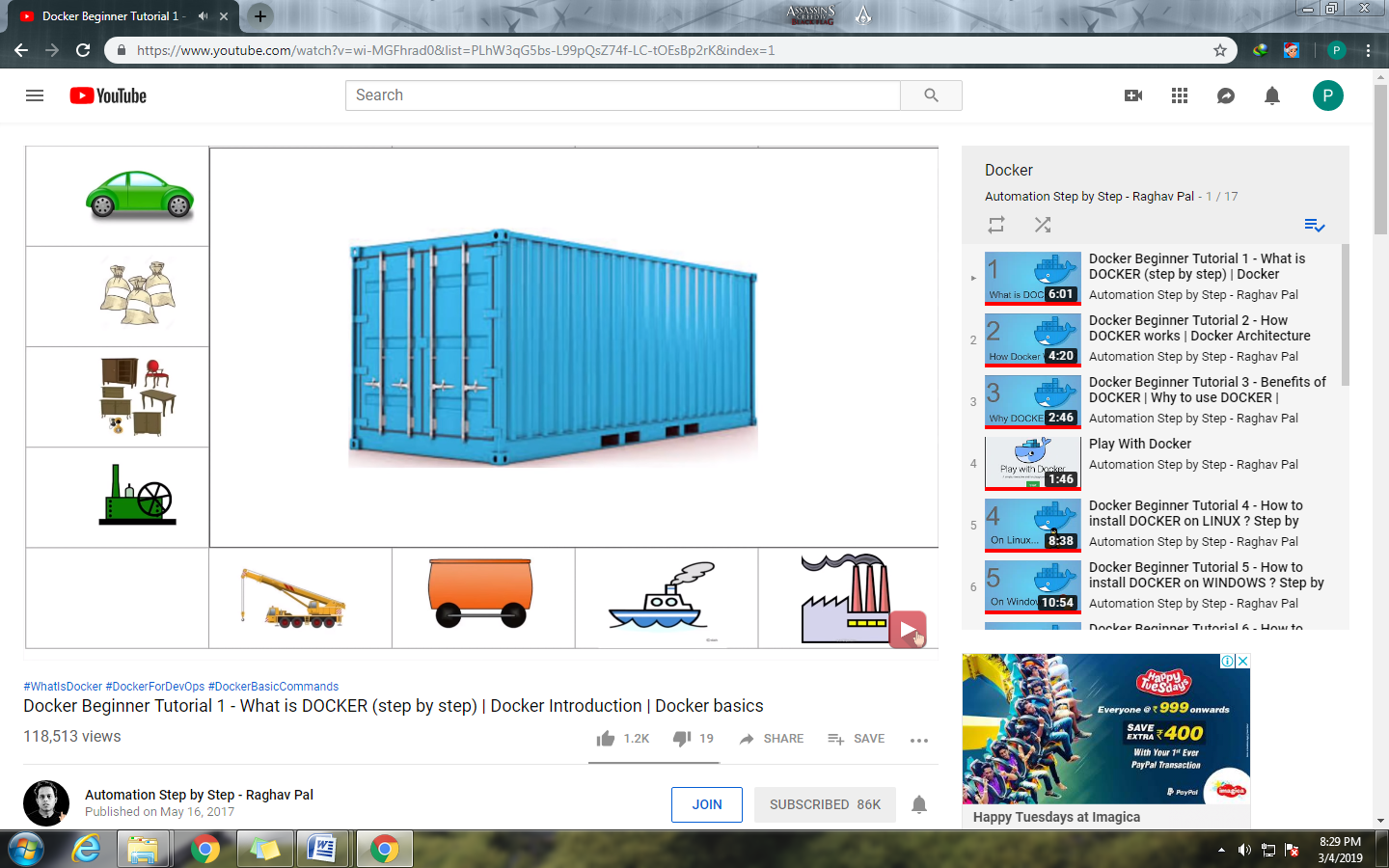


Now a days in many organizations developers and operations teams spend lots of time on ensuring the application run on each platform

**Solution to these problem:**

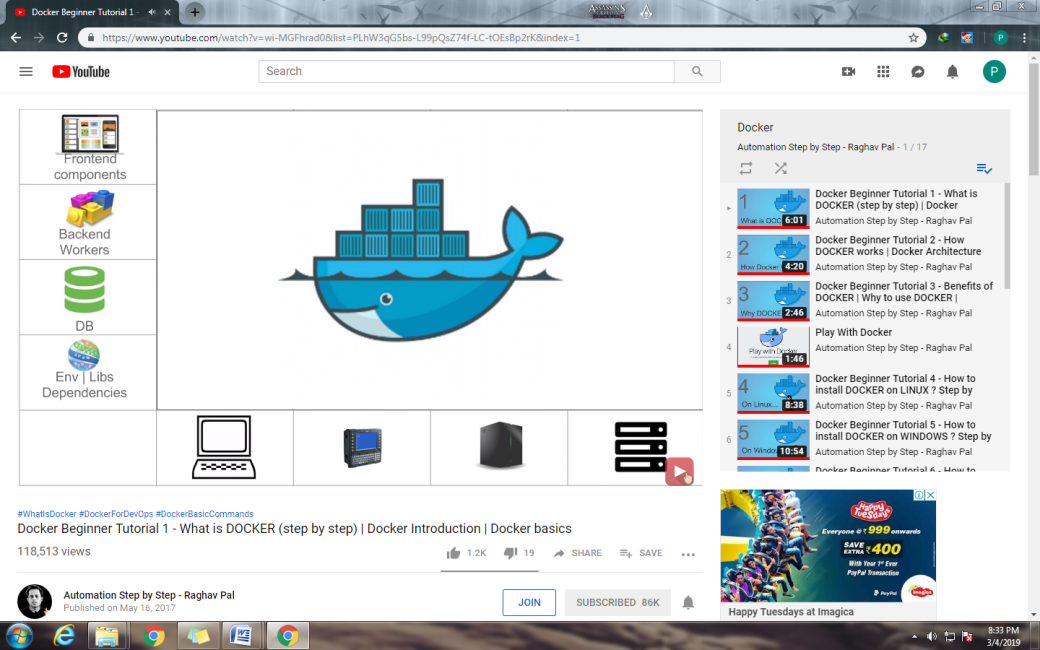
Ex: shipping problem

To solve above shipping problem containers are used as shown below:



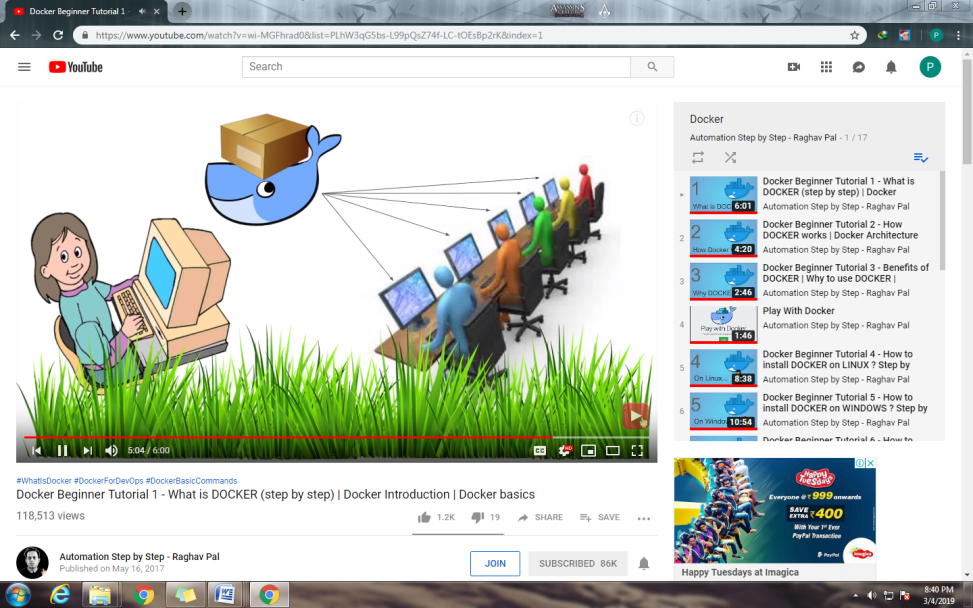
If you want to transfer the furniture, you only have worry about how to packaging the furniture in container. You have no need to worry about the how its transfer by road, water, & no need to worry about the expertise of workers who handle this containers.

So same thing can be applied to above software matrix similar like shipping industry containers so concept of dockers comes into picture

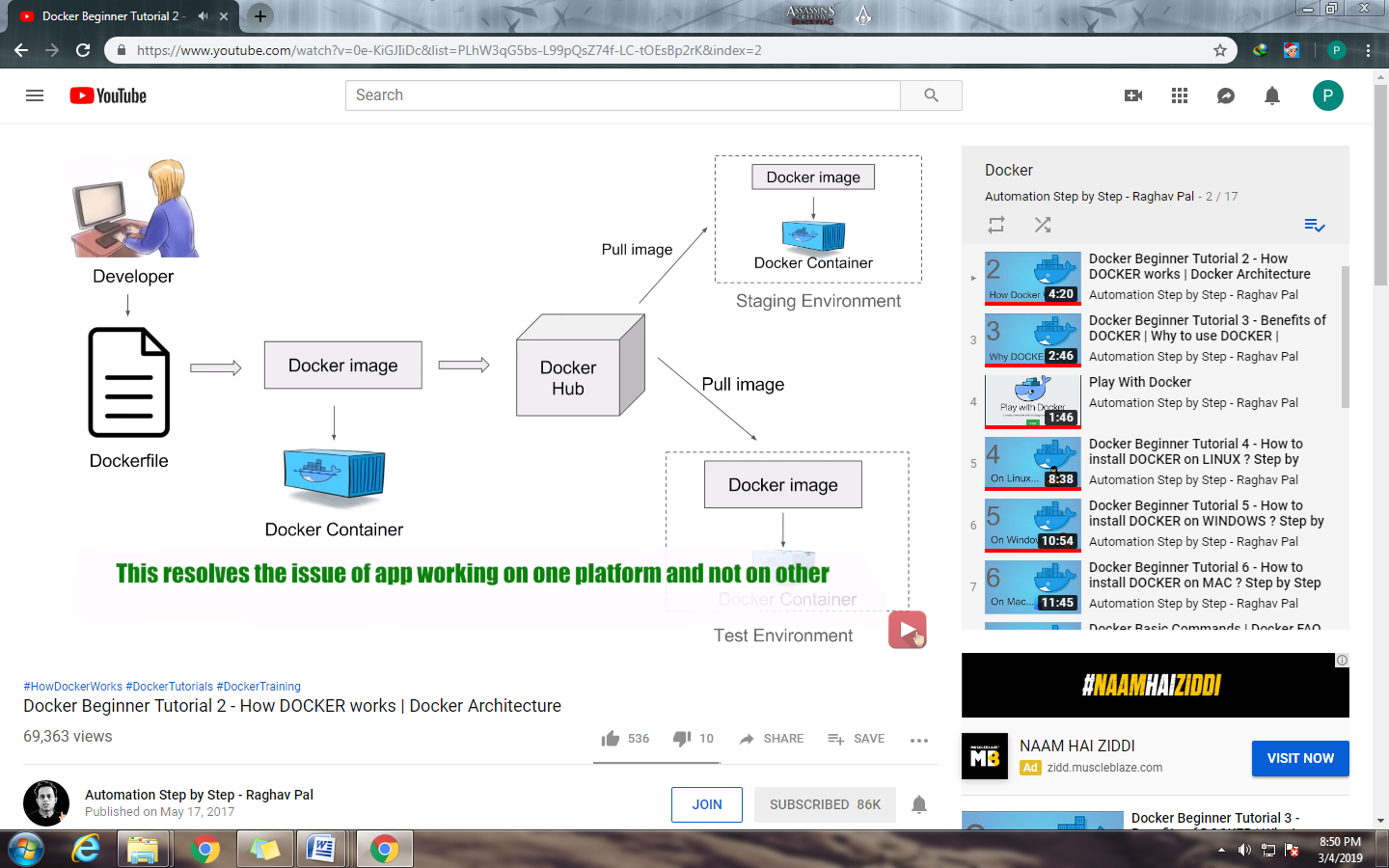


**What is Docker?**

* Docker is a containerization platform which packages application and all it's dependencies together in form of container.
* Docker containers wrap a piece of software in complete file system that contains everything needed to run: code, runtime, system tools, system libraries etc.
* This guarantees that the software will always run the same, regardless of environment.



**General workflow of Docker:**



In Docker workflow a developer defines all the application and its dependency and requirement into a file called Dockerfile. This Dockerfile can be used to create an docker images. so in a docker image you will have all the applications its requirement and dependencies. When you run a docker images we get docker container so Docker container is the runtime instance of Docker image and this images can also be stored in an online cloud repository called Docker Hub. In Docker Hub we will get lot of publically available images and we can store our own docker images as well. You can also stored your docker image in your own repository or version control system. Now this images can be pulled to create container in any environment. So we can create a docker container in test env or in any other env. And we can show that our application will be run on the same way either of any environment using Docker container.

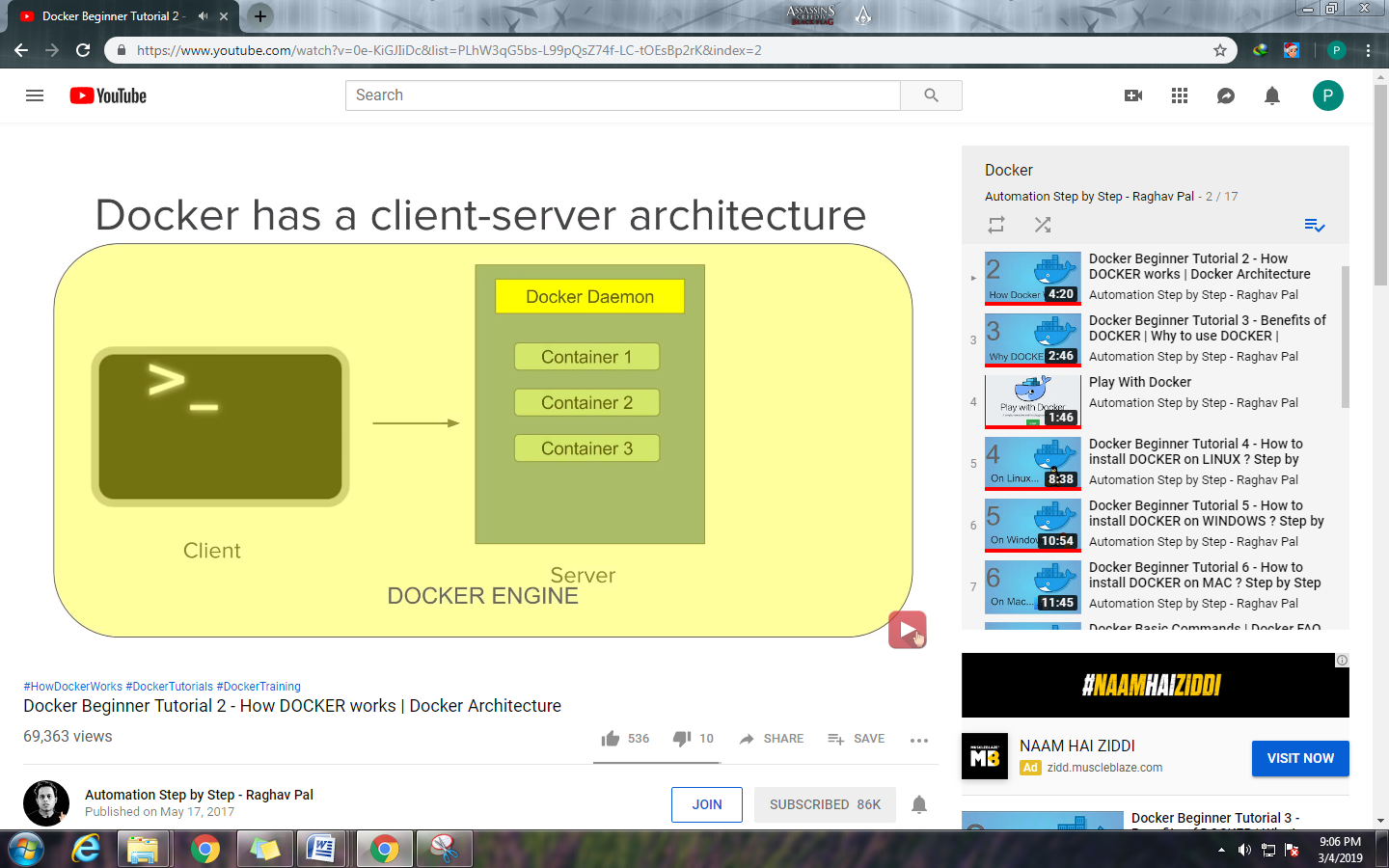
* **Dockerfile** :-It describes the step to create a Docker image. it’s like a recipe with all ingredients and steps necessary in making your dish.
* Container will have application with all its dependencies
* Docker is a Container platform

**Hypervisor :** In the concept of Virtual machine there is a software called **Hypervisor** which is used to create and run Virtual machine .Using Hypervisor we can create multiple VM’s on a host operating system. Now these VM’s will have their own operating system and it does not uses the host’s operating system. So there can be an overhead on the host platform. Also in case of VM we have to allocate a fixed memory and space to every machine so there is a lot of wastage of memory and space.

In **Containerization** we have a container engine. and we don’t have a separate operating system. But we have the containers where we will have the applications and all its dependencies and it will use the Host operating system. Here the space memory and other resources are not fixed it will be taken as per the needs of application.so there is no overhead it is very lightweight and very fast.

|  |  |
| --- | --- |
| Virtualization | Containerization |
|  |  |
| Higher utilization of resources | Low utilization of resources |
| Consumes higher memory as on each VM has its own OS. | Consumes less memory because they uses the Host OS |
| Each OS has its own kernel | Uses shared kernel of Host OS |
| Boot up very slow | Boot up faster |
| Different types of OS on hypervisior | Different types of containers on docker engine. |
| Different Kernels per guest OS are used using shared hypervisor | It uses special features of Unix file system to create each isolated environment. |

|  |  |  |
| --- | --- | --- |
| **Docker vs Hypervisors** | | |
| **Features** | **Hypervisors** | **Docker** |
| *Default Security Support* | To a great degree | To a slightly less degree |
| *Memory on disk required* | Complete OS plus apps | Application requirement only |
| *Time Taken to start up* | Substantially longer as it requires boot of OS plus app loading | Substantially shorter as apps only need to start as the kernel is already running |
| *Portability* | Portable with proper preparation | Portable within image format; typically smaller |
| *Operating System* | Supports multiple OS | It uses the host OS |



Docker Client:

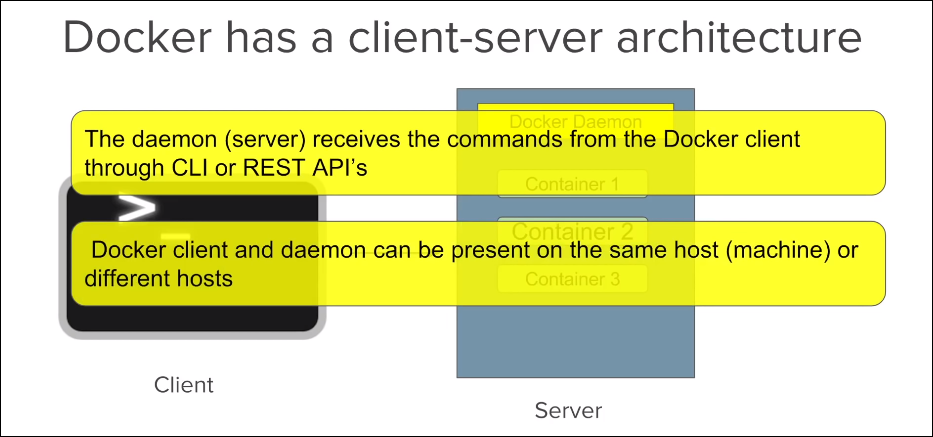
* Command line interface is the client

Docker Server or Docker Daemon:

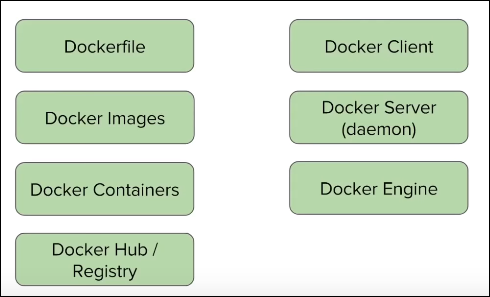
* Which contains all the containers.
* Docker server receives command from docker client in the form of commands or rest api request.

Docker Engine:

* Docker client and Doker Daemon/Server together called as Docker Engine.



Basics of Docker:



Advantages of Docker:

**Build app only once**: An application inside a container can run on any system that has docker installed. So there is no need to build and configure app multiple times on different platforms.

with docker you test your application inside a container and ship it inside a container. This means the environment you test is identical to one on the which the app will run in production.

**Portability**: Docker container can run on any platform. It can run on local machine Amazon ec2, virtual box, rackspace, google cloud platform etc.

ex: A container running created on AWS EC2 machine can easily be ported to virtual box.

**Version controlling :** You can do the Version Controlling for Docker also. If you make any changes in the images commit them and version control them

**Isolation:**

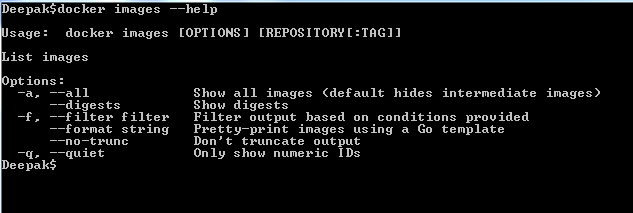
* With docker every application works in isolation in its own container and does not interferes with other application running on same system.
* For removal also you can simply delete the container and it will not leave behind any files or traces on system.

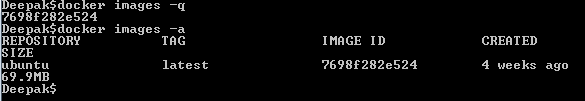
**Productivity:**

Docker allows faster and more efficient deployments without worrying about the running application on each environment.

**Docker basic commands :**

1. **docker version** : gives the information about docker client and docker server ie docker engine
2. **docker** –v /docker –version : gives docker installed version
3. **docker info** : gives detailed information about docker like noof container, Running, Paused, Stopped ,Images etc
4. **docker --help** : to get the option
5. **docker login** : To log into docker hub repository
6. **docker images** : list of images we have
7. **docker pull Ubuntu** :to pull a docker image named Ubuntu from docker hub
8. **docker images --help** : to get all options we can use with docker images



1. **docker images –q** : gives only the image id
2. **docker images –a** : gives all images
3. **docker rmi** :used to delete an image
4. **docker rmi imageid** : this particular image will be deleted
5. **docker ps** : gives all the container available
6. **docker run Ubuntu** : to run a container having image Ubuntu , if that image is not available first it will be pulled from docker hub and then it will run in the container. but still that image is not available in the container.
7. **docker start containerid** : to start a container -🡪 once this container is started then only Ubuntu image will be shown for that container
8. **docker stop containerid** : to stop a container
9. **docker stats** : gives us the detailed of running container like memory uses the input output and so on
10. **docker system df**: Used to check the disk uses of docker
11. **docker system prune** : remove unused data

* Dangling images means those images which are not associated with running container
* 

**What are images?**

Images are the template used to create Docker Containers. Container is running instance of image.

Ie image is a file which has information about what all things are required to create container.

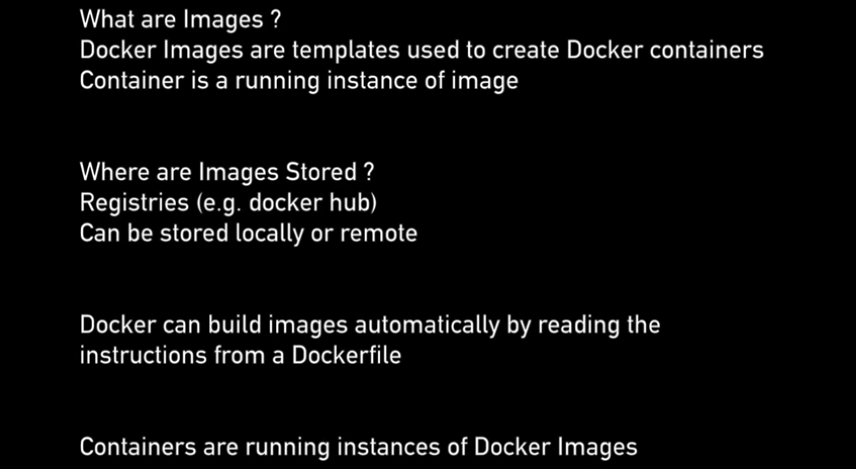
Images are stored on local or registry (hub.docker.com)

To pull docker image from registry use below command with options:

**docker pull <image\_name> ex: docker pull Ubuntu**

**docker pull <imagename:tag> ex: docker pull Ubuntu:18.04**

|  |  |
| --- | --- |
| Command | Description |
| docker images --help | Gives info about all the available options of docker image command |
| docker images | List all the available images. |
| docker images -q | Only shows numeric ids |
| docker images –f “dangling=false” | Filter output on given condition.  Dangling image is one that is not tagged and is not referenced by any container. |
| docker inspect<image\_name|ID>  docker inspect Ubuntu :18.04 | List the details of images |
| docker rmi <image\_name|ID>:<tag(optional)>  docker rmi Ubuntu:18.04 | To remove docker image use –f option to remove image forcefully |
| docker history <image\_name> | Shows the history of image |
| Docker images –f “dangling=false” -q | To show only the numeric id for all the images which is associated with any container. |



**What are container?**

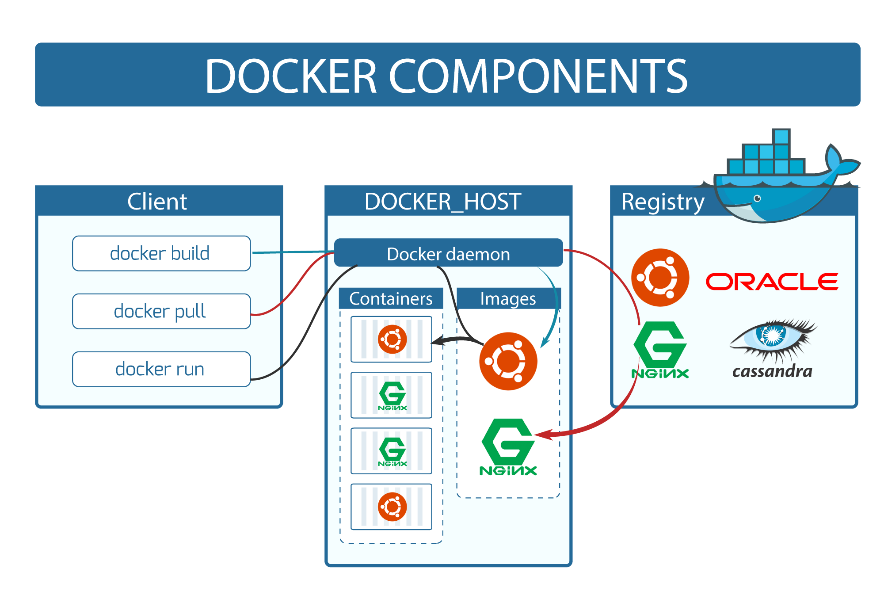
Running instances of docker images.

A container image is lightweight, stand-alone, executable package of piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.

Features:

1. Lightweight
2. Uses less resources
3. Booting of container is very fast
4. Can start stop kill remove containers easily and quickly
5. OS resources shared within docker
6. Container run on same machine sharing same OS kernel. This makes it faster

**Docker architecture:**



**Docker Client:** console used for running docker commands is called as docker client. In case of windows command prompt, for mac it is terminal, and for linux it is bash shell

**Docker Host:** It consist of-

1. Docker demon
2. Container
3. Images

**Note :** When we execute **docker pull** command it searches an image from Docker registry pull it from registry and keep it into our local system on the docker host. And if we execute **docker run** command first it will check that image availability into our local and run it and creates a container from that image, if that image is not available into our local system then first it will pull it from Registry to local and then started to run it to create a container.

**docker build** is used to create image using docker file and create a container

**Registry:** it can be on local or on remote (eg. hub.docker.com).it has all the images.

Commands:

|  |  |
| --- | --- |
| Command | Description |
| docker run –name myContainer -it <image\_name>  eg: **docker run –name MyUbuntu1 –it ubuntu** | It will run the image in interactive mode, it will first check image locally. If image not found locally then it searches in registry (hub.docker.com) and downloads that image and then run that image to create container and newly created container name will be |
| docker ps | List all the running containers |
| docker ps –a | List of all availablecontainer |
| docker pull image\_name | Used to pull image from registry  (eg. hub.docker.com) |
| docker start <container\_name|ID> | Starts the container |
| docker stop<container\_name|ID> | Stops the container |
| docker pause <container\_name|ID> | Pause the running container, so it will not perform any operations. |
| docker unpause <container\_name|ID> | Unpauses the running container, so it will starts performing operations again. |
| docker top <container\_name|ID> | Shows top processes of container |
| docker stats <container\_name|ID> | Gives stats of running container i.e. gives us the detailed of running container like memory uses the input output and so on |
| docker attach <container\_name|ID> | To attach running container to current command prompt |
| docker kill <container\_name|ID> | Kills the running container |
| docker rm <container\_name|ID> | Remove the container |

**Create Jenkins on Docker:**

Step 1: pull jenkin image from docker hub (hub.docker.com)

**docker pull jenkins**

Step 2: run the jenkin container :

**docker run -p 8080:8080 -p 50000:50000 jenkins** 🡪

-p command is to expose 8080 on the host system which corresponds to 8080 port for the docker server and also we are exporting 50000 in local (which is for Jenkins API) corresponds to 50000 port for docer server and image name is Jenkins.

* On running the above command our Jenkins server will be started on docker container and we will be able to access Jenkins by using url **localhost:8080** on our system

**docker run --name MyJenkins1 -p 8080:8080 -p 50000:50000 -v /your/home:/var/jenkins\_home Jenkins :** We are creating a new container named MyJenkins1 which is mapped the port 8080 of server to 8080 of our local and 50000 of server to 50000 of local machine. Also **–v command** is used to map Jenkins home directory with our local computer directory

eg: **docker run --name MyJenkins1 -p 8080:8080 -p 50000:50000 -v /Users/Deepak/Desktop/Jenkins\_Home:/var/jenkins\_home Jenkins 🡪 H**ere A folder named Jenkins\_Home will be creted on Desktop and will mapped with Jenkins war Jenkins home directory

**eg : docker run --name JenkinsOnDocker -p 9090:8080 -p 60000:50000 –v /Users/Deepak/Desktop/Jenkins\_DockerHome:/var/jenkins\_home jenkins**

Step 3: how to set jenkins home on Docker volume and Host machine

docker run --name myjenkins -p 8080:8080 -p 50000:50000 -v /var/jenkins\_home jenkins

docker volume create myjenkins

docker run -p 8080:8080 -p 50000:50000 -v myjenkins:/var/jenkins\_home jenkins

Volume is helpful when you want to share between containers keep your data unchanged.

**Docker file:**

A text file with instructions to build image, automation of docker image creation. And when we do the docker build for this docker file an image gets created.

We have lot of images in docker hub so we can pull that image to use but also we can create a new image with the help of dockerfile.so in ther words we can say **Dockerfile is Automation of Docker Image Creation.**

Step 1: **Create a the file named Dockerfile** :Whenever we run the docker build command docker searches for a file named **dockerfile** however it is not compulsory we can also give some different name and we can tell the Docker that this particular file is a docker file.

Step 2: **Write instructions**

**FROM**: getting base image (like ubuntu) you can use existing image or empty image(like scratch )

ex: FROM Ubuntu or FROM scratch (scratch is used for empty image from hub.docker.com)

**MAINTAINER**: mail id of dockerfile maintainer

**RUN**: it will executed during building of image.

ex : RUN apt-get update (gets the update from docker hub for base image)

**CMD**: command that only gets executed when you create container out of that image.

Ex: CMD [“echo”,”Hello World…! From my first docker image”]

Difference between RUN and CMD is Run gets executed during the building of the image where as **CMD**: command is executed only when you create container out of that image.

Step 3: **build dockerfile to create image**

docker build –t imageName:tag directoryofDockerfile

Step 4: **Run image to create container**.

docker images

docker run <imageID>

reference: [https://github.com/wsargent/docker-ch...](https://www.youtube.com/redirect?q=https%3A%2F%2Fgithub.com%2Fwsargent%2Fdocker-cheat-sheet%23dockerfile&redir_token=lfu-hdErx5XacvLdVHJ_QqLEfvZ8MTU1MTc2MjU4NUAxNTUxNjc2MTg1&v=LQjaJINkQXY&event=video_description)

[https://docs.docker.com/engine/refere...](https://www.youtube.com/redirect?q=https%3A%2F%2Fdocs.docker.com%2Fengine%2Freference%2Fbuilder%2F%23environment-replacement&redir_token=lfu-hdErx5XacvLdVHJ_QqLEfvZ8MTU1MTc2MjU4NUAxNTUxNjc2MTg1&v=LQjaJINkQXY&event=video_description)

**What is Docker Compose?**

* Tool for defining & running multi container docker applications
* Use yml files to configure application services (**docker-compose.yml**)
* Can start all services with single command: **docker compose up**
* Can stop all services with single command: **docker compose down**
* Can scale up selected services whenever required.

Step 1: install docker compose (already installed on windows and mac with docker**) docker-compose -v**

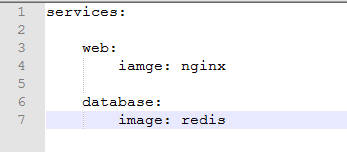
**For Linux installation** : 2 Ways to install docker on linux

1. [https://github.com/docker/compose/rel...](https://www.youtube.com/redirect?q=https%3A%2F%2Fgithub.com%2Fdocker%2Fcompose%2Freleases&event=video_description&v=HUpIoF_conA&redir_token=FdCB7d66IuJ5eaeeEavzJv5hFXB8MTU1MTc2NDA3MUAxNTUxNjc3Njcx)

2. Using PIP pip install -U docker-compose

Step 2: Create docker compose file at any location on your system

Standard name of file is : **docker-compose.yml**



Step 3: Check the validity of file by command **docker-compose config**

You can see the content of your yml file means yes it is now ok



Step 4: Run **docker-compose.yml** file by command

**docker-compose up -d** 🡪 -d is used to start in detached mode so that I will have available command prompt to run another command

so here we can check the running container is :::



Means two containers are available

Step 5: Bring down application by command **docker-compose down**

By running this command both the container is down now

**TIPS How to scale services —**

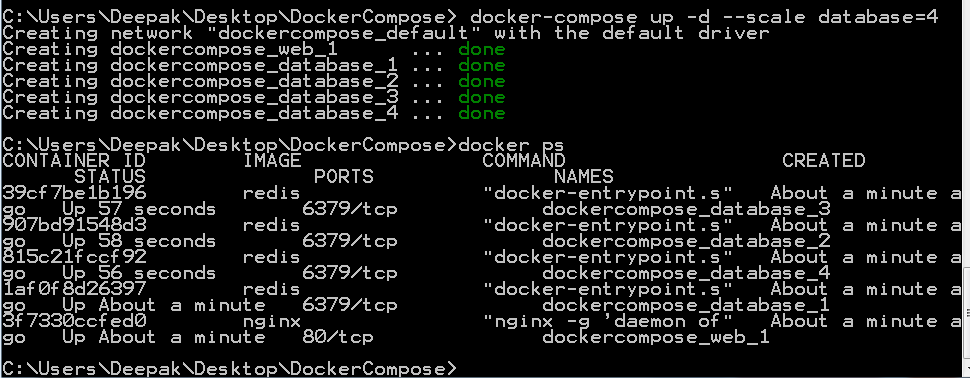
There is a command named **scale** to increase or decrease the number of instances for microservice

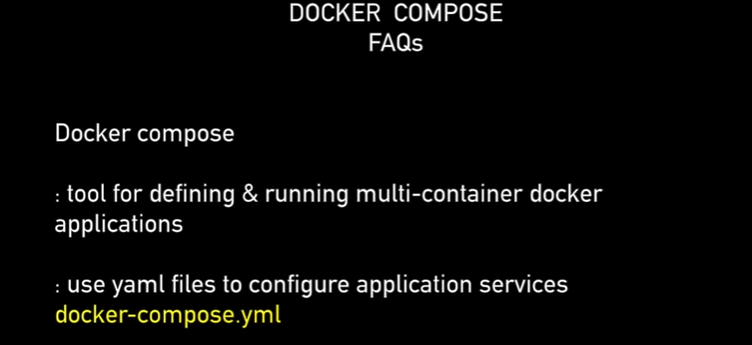
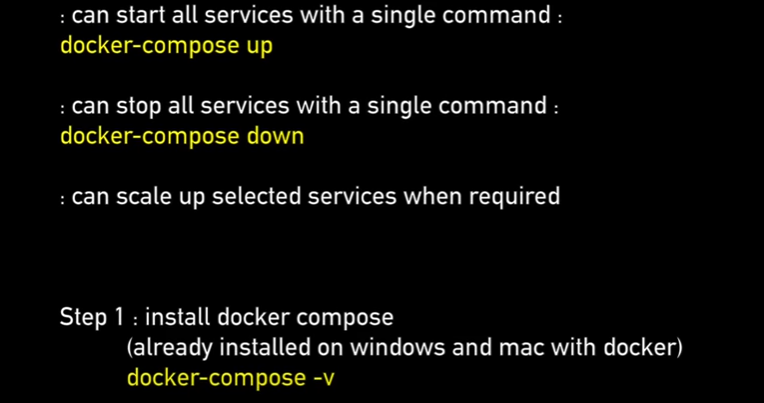
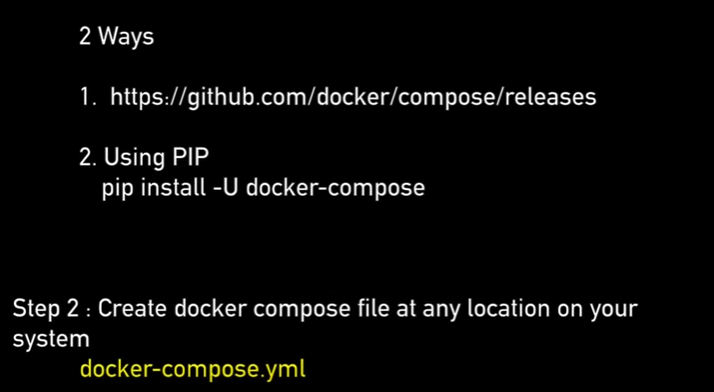
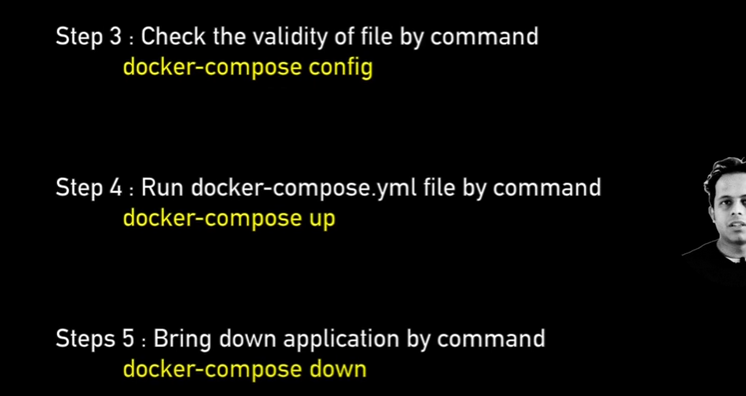
Scale : Set number of containers for a service

Q-> In our Microservice application we have two services web and database as discussed in docker-compose file. If we need 4 containers of database which command we need to run

Ans🡪

**docker-compose up -d --scale database=4**



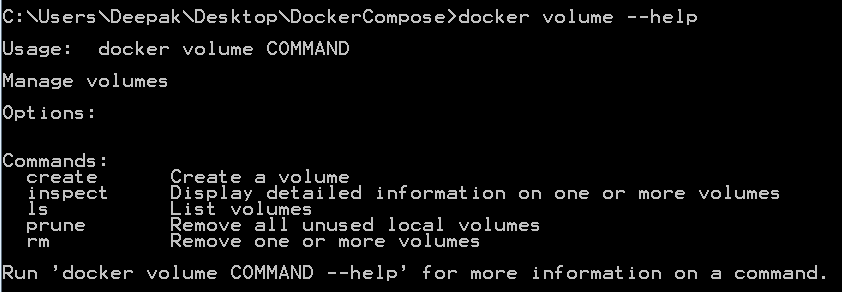
   

**Docker Volume :**

In docker whenever we create a container there has to be some place where data for the container will be stored. now if we don’t provide any explicit location for that data to be stored it gets within the container and when we delete the container or remove the container the data is also lost. however when we work for an enterprise projects we want the data is not lost we can actually remove the container but still persist the data. and in case it is required to create more container with the old data or to share the data between the container it should be possible.

Use of volumes :

* **It is used in decoupling container from storage** : It means storage is separated from the container and even if we delete or remove the container the storage is available
* It is used to share volume or data among different containers
* We can attach a volume to a container
* We can delete container but the volume will not be deleted



We create a volume named myvol1 : **docker volume create myvol1**

To get full details about newly created volume :

**docker volume inspect myvol1**



Mountpoint gives the volume location in our local system where data is stored so this place cant be edited by any function locally that is one of the advantages because this volume is safe now

How can we use this volume ?

First we pull Jenkins image

C:\Users\Deepak\Desktop\DockerCompose> **docker run --name DeepakJenkins1 -v myvol1**

**:/var/jenkins\_home -p 8080:8080 -p 50000:50000 jenkins**



We just run this command in which ewe creatae a new image named MyJenkins1 which will have a volume named vol1 which will get data from home directory of Jenkins and also it will map the port 8080of Jenkins under docker to port 8080 of our local amcjhine and same 50000 for Jenkins api

The above command will run jenkin server on docker container and its admin password will be :

**ae3810685aa4417ba1cff763c259e13a**

**now a jekins on port 8080 is stared .so we create a job named** TestJob .

Now we run the same command with changing the container name (**DeepakJenkins2**) and port number (**9090 and 60000**)

and run the command

We can see that the job created by first Jenkins started on port number 8080 is already available on port started on 9090

References: [https://hub.docker.com](https://www.youtube.com/redirect?q=https%3A%2F%2Fhub.docker.com&event=video_description&v=HUpIoF_conA&redir_token=FdCB7d66IuJ5eaeeEavzJv5hFXB8MTU1MTc2NDA3MUAxNTUxNjc3Njcx)[https://github.com/docker/compose/rel...](https://www.youtube.com/redirect?q=https%3A%2F%2Fgithub.com%2Fdocker%2Fcompose%2Freleases&event=video_description&v=HUpIoF_conA&redir_token=FdCB7d66IuJ5eaeeEavzJv5hFXB8MTU1MTc2NDA3MUAxNTUxNjc3Njcx)[https://docs.docker.com/compose/compo...](https://www.youtube.com/redirect?q=https%3A%2F%2Fdocs.docker.com%2Fcompose%2Fcompose-file%2F&event=video_description&v=HUpIoF_conA&redir_token=FdCB7d66IuJ5eaeeEavzJv5hFXB8MTU1MTc2NDA3MUAxNTUxNjc3Njcx)

Docker Commands: more info refer https://docs.docker.com/engine/reference/commandline/ps/

Docker Basic Commands:

|  |  |
| --- | --- |
| Command | Description |
| Basic | |
| docker version | Give info about docker client and docker server which is the engine. |
| docker -v / docker --version | Gives the version of docker |
| docker info | Gives detailed info about docker installed on your sysytem. |
| docker --help | Gives info about all docker commands. use docker [command\_name] --help to fetch details about particular docker command. |
| docker login | Used to login in https://hub.docker.com |
| Images | |
| docker images | Gives the list of images. |
| docker pull | Used to get the images from repository (code repository or https://hub.docker.com) |
| docker rmi | Used to remove one or more images. |
| Containers | |
| docker ps | List all the running containers. |
| docker run <images name> | To run the container. If image not present locally then it downloads the image from docker hub and then it starts the container |
| docker start <container\_id> | Starts container by container id |
| docker stop <container\_id> | Stops container by container id |
| System | |
| docker stats | Shows memory usage by container |
| docker system df | Shows disk usage of docker |
| Docker system prune --a | Removes all the unused images, container from docker. Use this command very carefully. |