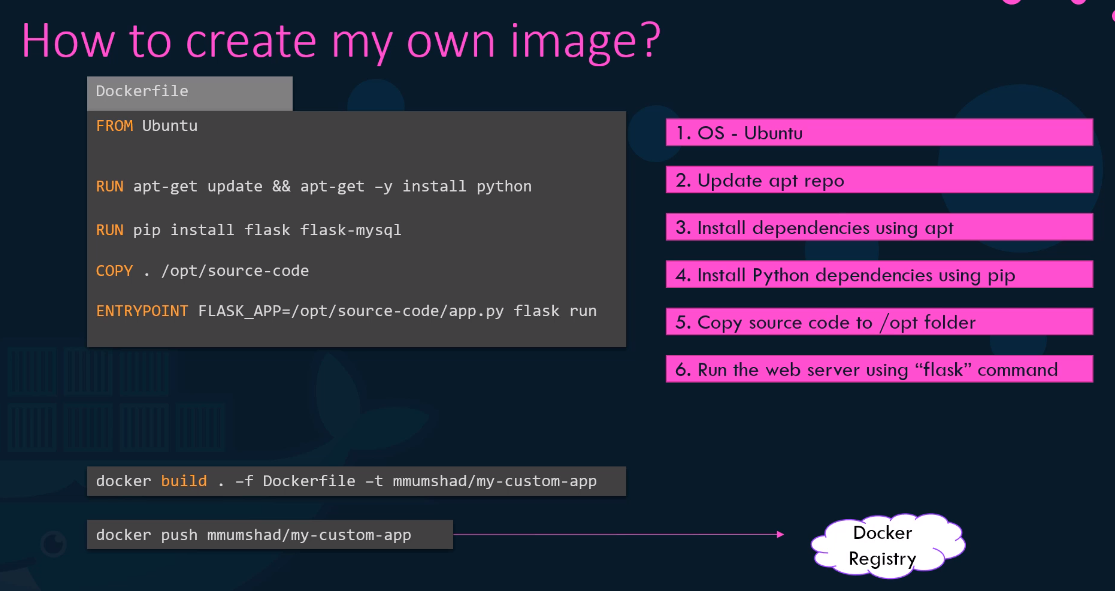
**Images:**

* Creating your own image: we need to create docker file to create an image.



* FROM : every docker file starts with FROM instruction, every new image is based on an already existed image (in our case base image =Ubuntu)
* RUN : it will run specific cmds (ie = installing dependencies).
* COPY : copies file from local system to docker image.
* ENTRYPOINT : specifies cmd that will run when image is run to make container.
* $ docker build . -f dockerfile -t “dockerhub username/name of image” : to build image from dockerfile and -t is used to give name to the newly created image. Here in tag put your Docker hub username/name of image so it goes straight under your repo on dockerhub . otherwise docker will block it
* $ docker push “name of image” : to push image to dockerhub repo , (you may need to login to your dockerhub repo first.)
* $ docker history “image-name” : to see history of image

**Environment variables:**

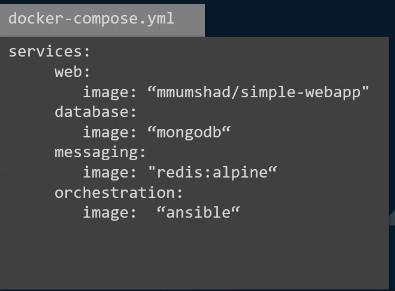
* If our application is using an environment variable , when we run docker run cmd, we can change the value of this variable as below
* $ docker run -e APP\_COLOR=blue “image name”
* $ docker inspect “image-name” : to find env-VARS being use in running contains

**Cmds, Arguments and Entry points:**

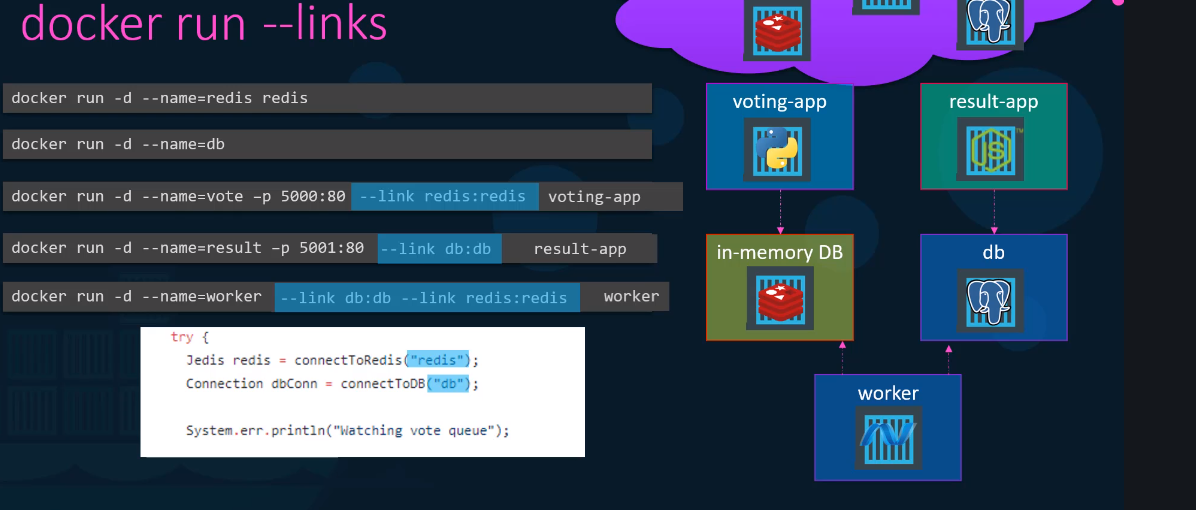
* $ docker built -t ubuntu-sleeper : to make “ubuntu-sleeper” image
* Now you can modify this image as per your need (ie. Image needs to start sleep cmd as beginning)
* 
* $ docker run ubuntu-sleeper 10 : now “ubuntu-sleeper” image will run for 10 sec (if we do not specify 10, then by default 5 sec as mention above)

**Docker Compose:**

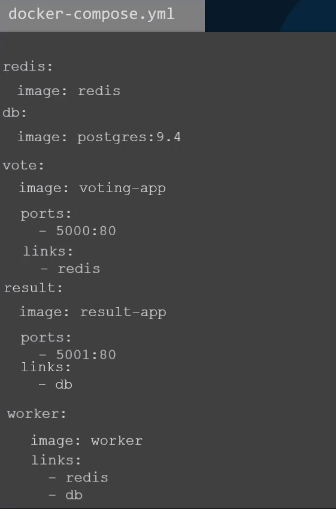
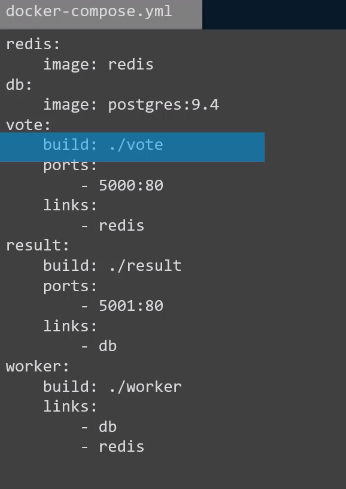
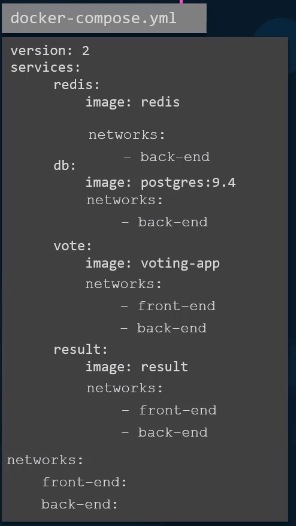
* If we want to run multiple images in docker (Application stack), we use docker compose.
* It is a .yaml file which has all the list of the images to run.



* $ docker-compose up : to run the compose.yaml
* We often needs to link the application with each other (ie : here voting app depends upon redis DB to store data, so there has to be a link b/w them)



* $ docker run “image1” --link “containername2: hostname2” : to link application 1 with application 2
* But this old way, new way is to use docker composed file.

If images are ready if we need to build image first version:2

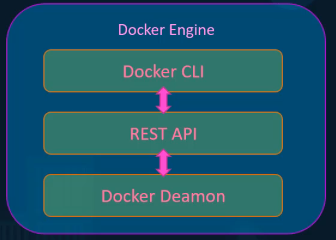
* No need to specify links in version 2 or 3 (as they provide n/w for all the services inherently)

**Docker Registry:**

* Central repo of all docker images. (i.e Docker Hub)

**Docker Engine:**

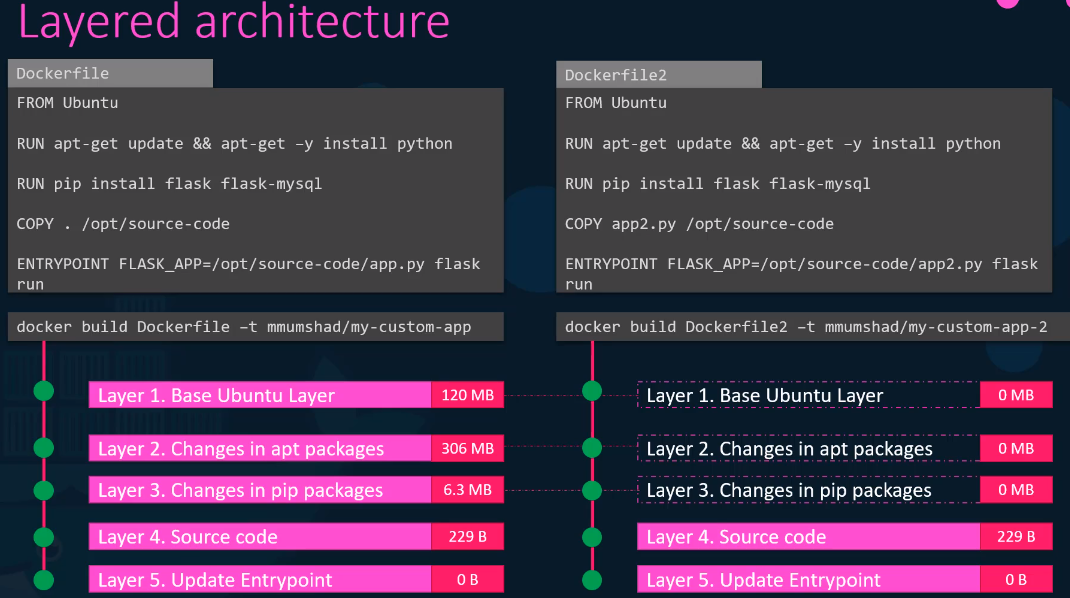
* it consist three things inside of it.
* Docker uses namespaces to isolate the workspaces .
* Underlying Docker host and all the containers uses resources out of same available pool (CPU and Memory)



* Namespace process id (PID) is one of the technique to provide different name spaces to different workspaces.
* $ docker exec “container id” ps -eaf : it will show all the processes running on the container

**Docker Storage: “persistence volume”**

* Docker uses layered architecture to save and optimize space and time of execution.

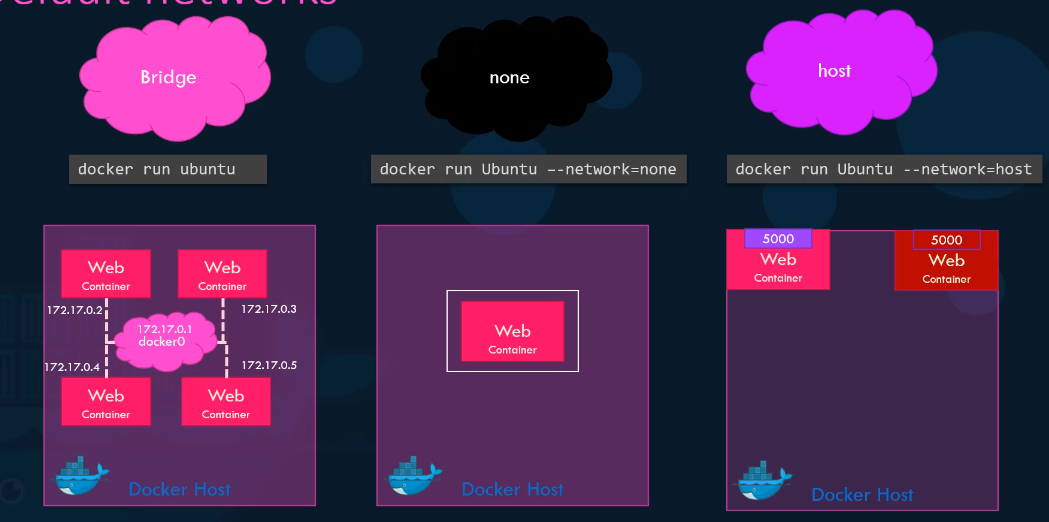


* When building 2 image (from dockerfile2) > docker sees that first 3 steps are already built while building 1 image so it skips to step 4 and starts building > saves storage and TOE
* if we would like to preserve the data created by container even after container gets destroyed, attach volume with it.
* $ docker volume create “name of volume” : to create a volume
* $ docker run -v data\_volume:/var/lib/mysql mysql : it will create **“data\_volume”** volume and will attach it to container
* $ docker run \

--mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql : new method to attach volume.

**Docker Networking:**

* When we install docker, it creates three spaces automatically
* Bridge : default n/w
* None : $ docker run ubuntu --network=none
* Host : $ docker run ubuntu --network=Host



* What if we want to assign different internal n/w to docker container , run below cmd
* $ docker network create \

--driver bridge \

--subnet 182.18.0.0/16

“n/w-name”