## Docker Command Notes

* docker run
  + Run an container from image
  + Ex docker run nginx would pull a image from cloud if not locally available on host
  + Same image would be used on subsequent runs
* docker ps
  + Lists all **running** containers along with other important information like container id, image, command, created, status, ports, names
  + To display all running and not running containers use command docker ps -a
* docker stop
  + To stop a running container, use this command.
  + complete command is docker stop <container\_id> or docker stop <container name>
  + use docker ps command for referring to name or container id.
* docker rm
  + To remove the containers previously running as refleced in docker ps command, use docker rm <container\_name>
  + Now the previously running image would not reflect in docker ps -a command.
* docker rmi
  + To remove a image, use docker rmi nginx command.
  + Make sure that no container is using that image before deleting
  + Delete all dependent containers before deleting the image
* docker pull
  + What if we only want to download the image from cloud but don’t want to run the image, use docker run nginx
* A container only lives as long as the process inside the container is alive-:
  + docker run ubuntu will keep it alive only till its boots
  + but docker run ubuntu sleep 5 would keep the container alive for 5 seconds (Use this method to run a command while running an image)
* To Run a command on running container,
  + use docker exec <container\_name> command
* Run-attach and detach
  + docker run <image\_name> (This runs in attached mode or in foreground)[meaning u would be attached to the imput and ouput of this image and you would see the output on your screen].
  + Would not be able to do anything else until this docker container stops[Would not respond to your inputs]
  + Use docker run -d <image\_name> [Notice the -d would run the image in detached mode, meaning you would be able to use the terminal as u want to use.]
* To attach back to the running container, use docker attach <container\_name> or docker attach <container\_id>
* Don’t worry about the complete id of the container, just type in first few characters of the id and hit tab for auto-completion.
* docker run redis would pull the latest version from cloud repo
  + If any specific version is required use tags Ex-: docker run redis:4.0
* By default if your application in container requires std input, docker run would not wait to accept the input from the user even if you have attached your terminal with the container.
  + The Reason is that, we are running over container in non-interactive mode.For interactive mode, use -i parameter.
  + Ex-: docker run -i <image\_name>
  + Notice that -i command would ensure that you are able to give input to the container but it would not log the container output on terminal, for that use -it flag.
  + Ex-: docker run -it <image\_name>
* Port Mapping/ Port Publishing
  + We can configure a host port to route the traffic to a particular port of the container using -p parameter.
  + docker run -p 80:5000 <image\_name>
* Data Persistence in docker container
  + if you have a container runing with mysql with lot of data in it. As soon as you remove the container, all the data is gone(deleted) like
    - docker run mysql
    - docker stop mysql
    - docker rm mysql
* If you want to persist the data, you need to connect the directory with the directory outside the container in the host using -v parameter.
  + Ex-: docker run -v /opt/datadir:/var/lib/mysql mysql. This way it will mount the external directory to the container when it runs and all the data would be saved in the host directory and would not get deleted when mysql container is stopped and removed.
* Inspecting Container
  + docker ps command is good enough to get basic information about the containers like there names and ids. If you need more details, use the command
  + docker inspect <container\_name>, docker inspect<container\_name> would return all the properties of the container in the json format and you can navigate the information you want to get.
* Container logs-: (The contents written by the container on stdoutput)
  + docker logs <container\_name> / docker logs <container\_id>
* Environment Variable
  + The -e parameter in docker run command is used to set the environment variable used for passing configuration parameter externally instead of hard codding them in application.
  + Ex docker run -e APP\_COLOR=blue < container\_id / container\_name >
  + For Inspecting all the environment variables configured for the particular container,

use docker inspect < container\_name / container\_id >

* Docker file -:
  + How to create my own image?
  + Just write down the process what would you do without docker-:
    - OS Ubuntu
    - Update apt repo
    - install dependencies using apt
    - install python dependencies using pip
    - copy source code to / opt folder
    - Run the web server using flask command
  + Now convert the each line in corresponding docker command
    - FROM Ubuntu
    - RUN apt-get update
    - RUN apt-get install python
    - RUN pip install flask
    - RUN pip install flask-mysql
    - COPY . /opt/source-code
    - ENTRYPOINT FLASK\_APP=/OPT/source-code/app.py flask run
  + Now build this docker file using the build command

docker build Dockerfile -t <dockerfilename>

[This would create the image locally on the system]

* + To push to cloud repo use command

docker push <dockerfilename>

This would get pushed to the account of the user

* + Some Information about dockerfile-:
    - Dockerfile is written in a specific format which docker can understand.
    - Its format is INSTRUCTION ARGUMENT
    - Everything on left and in caps is instruction
    - Everything on right is an argument to those instrunctions
    - All docker files must start from FROM instruction.
    - RUN is used to install all the dependencies.
    - Docker builds all the instructions line by line as separate layers which is reflected in the size of the each layer also.
    - To see this layer information, use the docker history <image\_name/image\_id>
    - docker build is done in steps and each step is cached.(In case of any step failure or you edit the docker file to a new step, this caching helps in increasing the build speed so that un changes steps results are directly retrieved from the cache.
* What all can u cantainerize? Everything!!!
* CMD vs ENTRYPOINT
  + The main purpose of a CMD is to provide defaults for an executing container.
  + An ENTRYPOINT helps you to configure a container that you can run as an executable.
  + The ENTRYPOINT specifies a command that will always be executed when the container starts. The CMD specifies arguments that will be fed to the ENTRYPOINT. is a good to-the-point summary. – Jingguo Yao
* Docker Network
  + There are 3 types of network in docker
    - Bridge-: docker run ubuntu
    - none-: docker run ubuntu --network=none
    - host-: docker run ubuntu --network=host
* Use host network if you want to expose the web server externally.
* Use none network if you don’t want to connect your container to any network. They run in isolated network.
* By default, docker contains only single bridge network at ip 172.17.0.1
* If you want to make some containers on a separate bridge network, use command
  + docker network create --driver bridge --subnet 182.18.0.0/16 custom-network
  + To check the newly created network, hit docker network ls
  + To check the internal network details of the container, use docker inspect <container\_id/container\_name>
  + If there is some container say mysql and you want to connect to that in application, don’t use mysql.connect( mysql ), instead use container\_id or container name because Embedded DNS would map the container\_name/id to ip address.
  + Built in DNS server always run at 127.0.0.11
  + Docker uses container namespace to isolate different networks.
  + When we connect different containers together, docker uses virtual ethernet cables to connect each other.
* Docker Storage
  + There are 2 types of mounting
    - Volume Mounting
    - Bind Mounting
  + docker volume create data\_volume
  + docker run -v data\_volume:/var/lib/mysql mysql
  + docker run -v docker\_volume2:/var/lib/mysql mysql
  + docker run -v /data/mysql:/var/lib/mysql mysql
  + -v is old style use --mount (more verbose)
  + docker run –mount type=bind, source=/data/mysql,target=/var/lib/mysql mysql
  + docker uses Storage drivers to perform all the layered architecture functioning.
    - Some common storage drivers are
      * AUFS
      * ZFS
      * BTRFS
      * Device Mapper
      * Overlay
      * Overlay2
    - The selection of the storage driver used depends upon the underlying os used like for ubuntu is AUFS.
    - Docker would use the best storage driver available based on the underlying OS used.
    - Different storage drivers cater to different needs and use cases of different applications.
* docker compose
  + Working in configurations in yaml file
  + If you want different components in your application like simple-webapp, mongodb, redis, ansible instead of giving all the commands seperatly like-:
  + docker run abc/simple-webapp
  + docker run mongodb
  + docker run redis:alpine
  + docker run ansible
* write all the commands in .yml file and simply execute that like
* docker-compose.yml

services:

web:

image: “abc/simple-webapp”

database:

image: “mongodb”

messaging:

image: “redis:alpine”

orchestration:

image: “ansible”

* Now simple execute docker-compose up
* All will run in a single host.
* Naming the containers also important for later easy linking
  + docker run -d --name=redis redis
  + docker run -d --name=db postgres:9.4 --link db:db result-app
  + docker run -d --name=vote -p --link redis:redis 5000:80 voting-app
  + docker run -d --name=result -p 5001:80
  + docker run -d --name=worker --link db:db --link redis:redis worker

Using link this way is deprecated and support can be removed any time in future.

Instead we can make docker-compose.yml file to do the same task-:

redis:

image:redis

db:

image: postgres:9.4

vote:

image: voting-app

ports:

* + - * 5000:80

links:

redis

result:

image:resut-app

ports:

5001:80

db

worker:

image: worker

links:

redis

db

* Learn more about different versions of docker compose file.