**Docker**

**Docker Commands:**

* docker login - It will prompt for username and password
* docker version – *To find the version of docker*
* docker info - Provides more information about the docker engine.
* Docker -D info – This will also provide more details about docker
* docker container run busybox echo hello world – *First check container is available or not and then download container from store and launch busybox container and print hello world*
* docker container run –it ubuntu bash (or docker container run -i -t ubuntu bash)-  *This will launch Ubuntu bash machine in Ubuntu container.* ***‘i’*** *tells docker to connect to container’s stdin and* ***‘t’*** *tell docker that we need pseudo terminal.*
* dpkg -l | wc -l : dpkg -l lists packages installed in container and wc -l counts them (runs in debian and Ubuntu machines)

<https://stackoverflow.com/questions/51247609/what-is-the-difference-between-docker-run-and-docker-container-run>

* apt-get update – Update the Ubuntu package
* apt-get install figlet – install the figlet package
* another way to run $apt-get update && apt-get install figlet
* docker container run jpetazzo/clock – this will run the container with clock and show output. If you want to come out of the running log press Ctrl+C
* docker container run –d jpetazzo/clock – This run the container clock without showing the running log. It will run as a demon container
* docker ps - This will show the current running containers with all details
* docker ps -a - This will list all the container including the stopped containers.

We can run this also as **docker ps -all**

* docker ps -l – this will list the last started container with all details
* docker ps -q – List the running container with only container Id’s
* docker ps -lq – List the last running container with only container ID.
* docker logs <container\_id> - to see the entire logs of the container.
* docker logs --tail 3 <container\_id> - To see the last 3 lines of the container log
* docker logs --tail 3 --follow <container\_id> - Displays the last 3 lines of the logs and show the running logs. To cancel the real time logs press ctrl+c
* docker kill <container\_id> - this will kill the container instantly.
* docker image pull <image> - To pull the image from docker registry
* docker image pull -a <image> - This will pull all the versions of the specific image.
* Docker search <image> - This will search the images in docker registry and display the result in sorted order, mostly with no of stars for images.
* docker search –limit 3 <image> - This will list the 3 images only.
* docker image ls - To show the current images
* docker image ls -ltr - To show the latest pulled image from the bottom
* docker container start <container\_id> - To start the stopped/killed container
* docker container start $(docker container ls -aq) – This is like query in query like sql . This will start all the containers that we wanted to start. First it will execute inline container and next it will start them.
* docker container start -a <container\_id> - To start the stopped/killed container and print the result
* docker container pause <cont\_id> - this will pause the running container
* docker container unpause <cont\_id> - This will unpause the paused container
* docker system prune - This will delete all the stopped containers.
* Docker container run -it -rm ubuntu bash – This is used for cleanup process. As soon as docker container is stopped, the container will be cleaned up.
* docker system df - This will show the list of containers images etc with memory and other details.
* docker container commit <container\_id> - This will commit the changes which we have done to the container but this won’t create any image id
* docker image tag <img\_id> <new\_img\_name> - This will create tag for an image
* docker network ls - To display the current networks
* docker network create mynetwork - This will create a new network with mynetwork name
* docker container diff <container\_id> - This will show the difference for the container.
* docker system df - To know the docker disk usage.
* docker images - This will list all images.
* docker tag <new\_img\_id> <name> - This will tag the image with a name. Next time when you want to run conatiner you can call with the created name
* vim Dockerfile - to create a docker file
  + Dockerfile is used to run set of instructions on the docker cli
  + Eg Dockerfile
    - FROM ubuntu
    - RUN apt-get update
    - RUN apt-get install figlet
  + Same the docker file
* CMD -f figlet script hi Madhu - CMD is the default command to run when nothing is given. If we return multiple CMD commands it will overwrite the previous one
* figlet -f script hello - This will run the figlet container with fancy font. Hello is the message.
* docker build -t <name of image> . - This is to build the docker file created. (.) represents the directory is as the current one.
  + docker build -t figlet . - figlet is the image name
* docker container run -it figlet - when you run this it will print the whatever it is written for CMD.
  + - FROM ubuntu
    - RUN apt-get update
    - RUN apt-get install figlet
    - CMD figlet -f script hi Madhu
* docker container run -it figlet bash - This will override the CMD command and enter into the bash mode.
* ENTRYPOINT [“figlet”, “-f”, “script”] - This will run the customer parameters given when running the container.
  + - FROM ubuntu
    - RUN apt-get update
    - RUN apt-get install figlet
    - ENTRYPOINT [“figlet”, “-f”, “script”]
  + ENTRYPOINT defines a base command (and its parameters) for the container.
  + The command line arguments are appended to those parameters.
  + Like CMD, ENTRYPOINT can appear anywhere, and replaces the previous value
* docker build -t figlet . and docker container run figlet madhuanandam - This will print the custom parameter madhuanadnam
* Using the CMD and ENTRYPOINT together
  + - FROM ubuntu
    - RUN apt-get update
    - RUN apt-get install figlet
    - ENTRYPOINT [“figlet”, “-f”, “script”]
    - CMD hello world
  + ENTRYPOINT defines a base command (and its parameters) for the container.
  + If we don't specify extra command-line arguments when starting the

Container, the value of CMD is appended.

* + Otherwise, our extra command-line arguments are used instead of CMD.
* docker container run -it --entrypoint bash figlet - This will override the entry point command in the Dockerfile . We have written bash before image name is because if we write bash after image name in this case it will print the bash instead of executing the shell.
* docker run --name Madhu ubuntu - This is used to name the container which is being pulled.
* docker run -p 8080:80 nginx - This is to map a port for docker container (p is small)
* docker run -P nginx - This is to map all the ports for docker container (P is capital)
* docker container port <cont\_id> - This will list the ports of the container.
* docker rm <container\_name> - This will remove the container which is stopped. Will throw error to stop container if we try to run this on running container.
* docker rm -f <container\_name> - This will remove the container which is running.(we can use container ID as well in place of container name)
* docker container prune - This will delete the stopped containers
* docker pull nginx - To download an image. This will not run any containesrs.
* docker rmi <image> - To delete the image.
* docker rmi -f <image\_name/ID> - This is force remove the image.
* Docker container run --publish 80:80 nginx - This to run the docker image nginx. Publish or -p is used to expose the ports on the container. 80:80 means opened the port on host IP and routes the traffic to container IP 80.
* Docker container top <container\_id/Conatiner\_name> - This will list the processes running inside the container.
* docker container inspect <container\_id/Conatiner\_name> - To inspect the docker container what code it consists.
* docker container stats <container\_id/Conatiner\_name> - This to give the performance of the CPU when running the container.
* docker container run -it --name ubu ubuntu bash- This will run the Ubuntu image in interactive bash shell mode. -i is for interactive and -t is for tty.

Once we execute our commands inside bash and try to exit the command, the container stops and nothing will run.

* apt-get update && apt-get install -y curl - This will update the Ubuntu package and installs the curl inside the container.
* curl google.com - this will connect the google.com and return the code.

Once we exit from the container the container will die. If we want to run the same curl again on the same Ubuntu then we have to run the below

* docker container start -ai ubu - This will start the stopped container ubu from the previous state. This will have the curl installed. -a is for attach and -i for interactive.
* Docker container attach <container> - This will attach and display to the running container.
* docker container port webhost - It shows the port details of the webhost container.
* docker container inspect --format ‘{{ .NetworkSettings.IPAddress }}’ webhost - This will give the ip address of the container. Docker inspect gives the full details, if we want a specific details in the config file, then use the --format.
* Docker system prune -a and docker system prune – The difference is
  + Docker system prune – a : will remove the below
    - All stopped containers
    - All networks not used at least by one container
    - All images without atleast one container associated to them.
    - All build cache
  + Docker system prune : will remove the below
    - All stopped containers
    - All networks not used at least by one container
    - All dangling images
    - All dangling build cache.
* Docker stats <container\_id> - this will give the status of the container like what is the memory used, cpu used etc.,
* Docker container run -d -m 512m jpetazzo/clock – This will limit the usage of the container
* Docker container run -d -m 512m –cpu-quota=50000 jpetazzo/clock – This will allocate only 50000 cpu quota to the container. **Pease note the total quota for PCU is 100000.**

Dockerfile commands:

* FROM – Every dockerfile starts with FROM command, FROM command will give the base image for the docker image
* MAINTAINER – This command is used to mention the author of the dockerfile
* RUN – run command is used to run the shell commands. These commands will run on top of the base image layer. Run commands executed during the build time of the docker image. You can use command any number of times in the docker file
* CMD – The CMD command doesn’t execute during the build time, it executes after the creation of the container. There can be only one cmd command in docker file. If you add more cmd commands, then the last cmd command will gets executed and rest all will be skipped. Whatever you are mentioned with cmd command in dockerfile can be overwritten with docker run command. If there is ENTRYPOINT command in docker file, then cmd will gets executed after entrypoint. With cmd, you can pass arguments to entrypoint command. CMD [“param1”, “param2”] – This param1 and param2 will be passed as arguments to entrypoint command.
* ENTRYPOINT ["/decodingdevops.sh"]
* CMD ["postgress"]

in the above example, postgress is the first argument to entrypoint command this is like running the shell script with one argument. decodingdevops.sh postgress executable form:

CMD [“python”, “app.py”]

python app.py will be run in the container after the creation of the container.

* EXPOSE – After creating the container, the container will listen on this port.
* ENV – This command is used to set environment variable in the container.
* COPY – This will copy the files and directories from the host machine to the container. If the destination does not exists, it will create the directory.

COPY <SRC> <DEST>

COPY . /usr/src/app – Here dot (.) means all files in the host machine dockerfile directory , will be copied into container /usr/src/app directory.

* ADD –

ADD <SRC> <DEST> - it will copy the files and directories from the host machine to the container and its support copy the files from remote URLs.

* USER – This command will sent the userid or username when running the image.
* WORKDIR – This is just like cd command in linux If you mention any path after workdir the shell will be changed into this directory. The next mentioned commands like run,cmd,entrypoint commands will be executed in this directory.

WORKDIR /devops

RUN npm install

here npm install command will run on devops directory.

* ENTRYPOINT -

The first command that will execute in the container is entrypoint command. Entrypoint command will be executed before the cmd command. If you mentioned more than one, only the last one will be executed and remaining all will be skipped. cmd command paraments are arguments to entrypoint command.

ENTRYPOINT ["/decodingdevops.sh"]

CMD ["postgress"]

above commands will run as /decodingdevops.sh postgress

here postgress is the first argument to script decodingdevops.sh

* docker swarm init –advertise-addr $(hostname -i) – Create the docker swarm
* docker node ls – lists the nodes
* docker container run -it –name mysql2 -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=true -v mysql-db:/var/opt/mysql mysql – This will create a named volume for mysql db
* docker container run -it –name mysql2 -e MYSQL\_ALLOW\_EMPTY\_PASSWORD=true -v /usr/libs/jenkins:/var/opt/mysql mysql – Thiis is called bind mounts. Here you are mounting host directory to the container.