**Docker**

* Docker is one of the tools that used the idea of the isolated resources to create a container that allows applications to be packaged with all the dependencies installed and ran wherever we wanted.
* Docker can only run on Linux machines this means I cant install Docker directly on Windows or any other OS.
* If I want install Docker on windows then I need to run a Linux VM in windows on top that I need to run Docker.

**Virtualization (VM)**

* VM is way of running virtual OS on top a host OS using a special software called Hypervisor.
* VM directly shares the hardware of the host OS.

|  |  |
| --- | --- |
| Virtualization at hardware level | Virtualization at OS level |
| Heavyweight - consume more host resources | Lightweight |
| VM uses hypervisor | containerisation tool is used |
| limited performance - Boot up time is more which is in minutes | Native performance - usually boot fast in seconds. |
| Consumes more storage | Shares OS storage means only uses required storage. |
| Supports all OS | Supports on Linux |

**VM                                                            Containerisation**

**host machine**  
    This is the machine in which docker is running **Docker image**

Docker definition: A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, configuration and other settings.  
  
Like: Docker image as a digital lunchbox. It holds everything your software needs to work - like ingredients for a sandwich. This lunchbox is super handy because it keeps your software fresh and ready to go, no matter where you take it. It's like having your favorite meal prepped and packed, ready to enjoy anytime, anywhere!  
  
**List docker images**  
    docker images   
    docker image ls   
  
**Download/Pull images from docker hub**   
    docker pull <repo>:<tag>  
    Note: If we wont provide any <tag> then default tag 'latest' is considered  
  
**Delete docker images in the host machine**  
    docker rmi <repo>:<tag>  
    docker image rm <repo>:<tag>  
  
**Delete all images in docker**  
    docker images -q | xargs -I{} docker rmi {}  
    docker rmi $(docker images -q)

**Tag a existing docker image**  
    docker tag <old\_image> <username>/<repo\_name>:<tagname>  
  
**Upload/Push images to registry (Docker to authenticate with a Docker registry.)**  
    1. Need to prepare image which matches the same name as repo   
        (image name syntax: <username>/<repo\_name>:<tagname>)  
        docker tag <old\_image> <username>/<repo\_name>:<tagname>

    2. To connect to docker hub account   
        docker login

    3. Then push the above tagged image to hub   
        docker push <username>/<repo\_name>:<tagname>  
  
**TRY: 1.** **Amazon Elastic Container Registry (ECR)**:  
            aws ecr get-login-password --region <AWS\_REGION> | docker login --username AWS --password-stdin <AWS\_ACCOUNT\_ID>.dkr.ecr.<AWS\_REGION>.amazonaws.com  
            **NOTE:** <AWS\_REGION> and <AWS\_ACCOUNT\_ID> with your AWS region and account ID

**2.** **JFrog Artifactory Container Registry**:  
            docker login <ARTIFACTORY\_URL>

**Delete all stopped container**  
        docker rm $(docker ps --filter "status=exited" -q)  
        docker container prune

**Create container from image (Run a container from image)**  
        docker run -it -d --name <container\_name> <image>  
            -i - Interactive mode   
            -t - enable tty in the terminal   
            -d - To create container in detached mode (in background)  
            --name - To assign a custom name to the container   
            --rm - Using this will automatically deletes the container when it stop/exited

**Docker custom image / Dockerfile / Docker instructions**

A blue whale with blocks and blue text

Description automatically generated with medium confidence      
**Dockerfile**  
        Dockerfile is used to create custom images by using any stock image or other image as base image.  
        In Dockerfile we can write some set of instructions to update any image.  
      
        To create image from Dockerfile  
            docker build -t my\_ubuntu .  
**FROM ubuntu**  
        FROM is the first instruction in the every Dockerfile  
        FROM is used to specify the base image on top which all the other   
            instruction will run in the same Dockerfile.  
              
        FROM <image\_name>:<tag>  
          
**RUN**  
        Normal shell command or the commands supported by the base image are executed using this instruction.  
        we can have n number of RUN in a single Dockerfile.  
          
        Normal command format   
            RUN <command>

**List only stopped**  
        docker ps --filter "status=exited"

**To stop, start, restart, pause containers**

        docker stop <container\_id1> <container\_id2> .... <container\_idn>  
        docker start <container\_id1> <container\_id2> .... <container\_idn>  
        docker restart <container\_id1> <container\_id2> .... <container\_idn>  
        docker pause <container\_id1> <container\_id2> .... <container\_idn>

**To check the resource utilized by all the container**  
        docker stats           
              
**To list all the process running inside a container**  
        docker top <container\_id>

**Container Logs**

* Docker collects logs from containers primarily through its logging drivers.
* When a container generates output to stdout (standard output) and stderr (standard error), Docker captures this output at */var/lib/docker/containers/<container-id>/*
* We can retrieve logs from a running or stopped container using the **docker logs** command

**To fetch the logs generated by a Docker container.**  
        docker logs <container\_id>

**To fetch the logs in Real-Time (Follow).**docker logs -f <container\_id>

**To fetch the logs with Timestamps.**docker logs --timestamps <container\_id>

**A diagram of a container command

Description automatically generated**

**How to login to a container**  
        **1. To attach to a container**  
            docker attach <container\_id>

**To detach from a container**

* press (Ctrl + p) followed by (Ctrl + q).
* This sequence tells Docker to detach from the container without stopping it.
* After detaching, you'll return to your host shell while leaving the container running in the background.

**2. To use exec to create a new shell**  
            docker exec -it <container\_id> /bin/bash

A diagram of a diagram

Description automatically generated

**COPY and ADD**  
        Both copy and add instruction is used to copy files and directories from host machine build location to the image and the container created from it.  
      
**ADD supports extra source formats**

* If the source is a compressed file then ADD will automatically uncompress it in the destination.
* If the source is a downloadable link then ADD will automatically download the file in the destination.

        COPY <source\_path\_from\_build\_context> <destination\_inside\_image>        
        ADD <source\_path\_from\_build\_context> <destination\_inside\_image>

A diagram of a docker image

Description automatically generated

A diagram of a software development

Description automatically generated with medium confidence

**ENV**

* This instruction is used to set the environment variable inside the container.
* Using this instruction we can create env variables at build time which means in the docker images.

ex:

   1. For individual variable

       ENV <variable\_name> <value>

     (OR)

       ENV <variable\_name>=<value>

   2. For multiple variable

       ENV <variable\_name1>=<value1> <variable\_name2>=<value2> .....

**To create environment variables at run time (means in containers)**

1. With the docker run command

    docker run -e <variable\_name>=<value> -e <variable\_name>=<value>

2. With a list of variables in a file (.env file)

    docker run --env-file <file\_path> ...

**CMD vs ENTRYPOINT**

* Both CMD and ENTRYPOINT are used to define the default execution command of the container (the command which will be executed in the container as main process).
* If we use multiple CMD or ENTRYPOINT in the same Dockerfile only the last one will be considered.
* If we use both CMD and ENTRYPOINT in the same Dockerfile, then ENTRYPOINT gets the highest priority and the command defined using CMD will be as parameters to ENTRYPOINT.

CMD ["command","param1","param2",...]

ENTRYPOINT ["command","param1","param2",...]

**Difference**

* CMD can be completely overridden at the runtime (with docker run at the end we can provide the command to override the CMD).
* ENTRYPOINT can't be overridden at the runtime but the command passed at the runtime will become parameters to ENTRYPOINT command defined in Dockerfile.  
            
  Syntax: we can define command in 2 ways   
        1. shell format   
               CMD "ls -lrt"  
            
        2. EXEC format   
  + - * Always first element is command.
      * Except first element all the other elements are parameters to command.

                  CMD ["ls","-lrt"]

**MAINTAINER**

* The MAINTAINER instruction is used to update the Author of the docker images
* The MAINTAINER instruction has been deprecated in favor of using LABEL for metadata since Docker version 1.13. Instead of MAINTAINER

MAINTAINER <Your Name>

(OR)

LABEL maintainer="Your Name <your.email@example.com>"

**USER**

* The USER instruction is used to set the username or UID (User Identifier) for the commands following it in the Dockerfile.
* The USER instruction is used to specify which user the container should run as

USER <username or UID>