**Agenda: Creating Custom Docker Images**

* Dockerfile and Building Docker Images
* Breaking down and understanding dockerfile indepth

**Creating Image from current state of Container**

Base Image 🡪 Container 🡪 Add files and Folder 🡪 Committed the Container 🡪 New Image 🡪 New Container

1. Start a container from the image nginx

docker run -d -p 8080:80 nginx

1. Get the list of containers and note the Container Id

docker ps -a

1. Also you can copy content from Host to Container

In some directory put HTML files – Our custom website build locally.

mkdir demo

cd demo

echo Hello world > hello.html

echo Testing > test.html

echo This is a Demo > demo.html

docker **cp** . <container-id>:/usr/share/nginx/html

docker diff <container-id>

1. Create an image from container ID where we created our file:

docker **commit** <container id> mynginx:v1

1. Note that the new image by name **mynginx:v1** is created

docker images

1. Start a new container using the newly created image

docker run -p 8081:80 -d **mynginx:v1**

curl http://localhost:8081/hello.html

**Creating Image using Dockerfile**

* A Dockerfile is a text file which contains a series of commands or instructions. You need to use an existing image as the starting point for your app, but you decide which one.
* These instructions are executed in the order in which they are written.
* Execution of these instructions takes place on a base image.
* On building the Dockerfile, the successive actions form a new image from the base parent image.

1. **Create a Dockerfile:**

FROM nginx

WORKDIR /usr/share/nginx/html

RUN echo "Hello, How are you…" > hello1.html

RUN echo "Hello, How are you…" > hello2.html

RUN echo "Hello, How are you…" > hello3.html

COPY . .

1. The first instruction must be **FROM**. This instruction initializes a new build stage and sets the Base Image for the remaining instructions. The multi-arch tags pull either Windows or Linux containers depending on the Docker for Windows container mode.
2. The **WORKDIR** instruction sets the working directory. If the directory doesn't exist, it's created. In this case, WORKDIR is set to the HelloWorld directory. The WORKDIR instruction **wont create a new layer** in the image but will add metadata to the image config.
3. The **COPY** instruction copies new files or directories from the source path and adds them to the destination container filesystem.
   1. COPY <src>... <dest>
   2. COPY ["<src>",... "<dest>"] (this form is required for paths containing whitespace)
4. **RUN** executes the instruction
5. **Build the Image**

docker **build** -t mynginx:v1 .

1. **Run the Image – Create the container**

docker **run** -p 8090:80 mynginx:v1

docker ps

docker exec <container ID> ls /usr/share/nginx/html

**Show the running process in the container (Container must be running and not exited)**

D:\demo>docker top <Container ID>

**Understanding CMD and ENTRYPOINT**

1. The **ENTRYPOINT** instruction allows the container to run as an executable.
   1. ENTRYPOINT should be defined when using the container as an executable.
   2. ENTRYPOINT echo "Hi, your ENTRYPOINT instruction in Shell Form !"
   3. ENTRYPOINT ["/bin/echo", "Hi, your ENTRYPOINT instruction in Exec Form !"]

**CMD vs EntryPoint**

FROM ubuntu

CMD [ "echo", "THIS IS COMMAND…" ]

**docker build -t demo .**

**Note the o/p of following commands**

* docker run demo #o/p=THIS IS COMMAND…
* docker run demo ls #ls is a command and o/p=Lists all folders

FROM ubuntu

ENTRYPOINT ["echo", "THIS IS ENTRYPOINT..."]

**docker build -t demo .**

**Note the o/p of following commands**

* docker run demo #o/p=THIS IS ENTRYPOINT…
* docker run demo ls #o/p=THIS IS ENTRYPOINT…ls

FROM ubuntu

CMD ["echo", "THIS IS COMMAND"]

ENTRYPOINT ["echo", "THIS IS ENTRYPOINT..."]

**docker build -t demo .**

**Note the o/p of following commands**

* docker inspect demo
* docker run demo #o/p=THIS IS ENTRYPOINT… echo THIS IS COMMAND
* docker run demo ls #o/p=THIS IS ENTRYPOINT…ls

**Overriding EntryPoint**

If an image has an ENTRYPOINT if you pass an argument it, while running container it won't override the existing entrypoint, it will append what you passed with the entrypoint.

To override the existing ENTRYPOINT you should user --entrypoint flag when running container.

$ docker container run **--entrypoint** "echo" demo "Hello, Welcome to my docker session! "