* Creating a Docker container from Dockerfiles. *

- Building an image from our own Dockerfile.

Sample docker file:

```
FROM ubuntu
```
↳ which existing Dockerimage to base our image off of.

```
LABEL maintainer = "Anant <anantk.singh...>
```
↳ additional info

```
USER root
```
↳ tells which user to use for commands below
↳ default

```
COPY ./entrypoint.bash /
```
↳ copies the file from path to container

↳ to customize our image

```
RUN apt-y update
RUN apt -y install curl bash
RUN chmod 755 /entrypoint.bash
```
} install additional softwares and configure files

↳ changes user (so that we don't change host files)

```
USER nobody
```

```
ENTRYPOINT ["/entrypoint.bash"]
```
↳ what command containers created from this image should run

Q How to turn Dockerfile into a Docker Image?

> docker build → builds an image

- docker build [OPTIONS] PATH | URL | - (or)

Aliases: docker image build, docker buildx b, etc. (using buildkit)

↳ Commonly used option:

- -t, --tag → Assigns a name to image, can be used in place of image ID.

- -f, --file → to provide the name of the dockerfile if it's not "Dockerfile"

→ PATH | URL | - are context → folder containing files that docker will include in the image

for working directory → .

* Note:

Every docker image has an image ID.

When building an image docker will look for a "Dockerfile" in the directory to run. If the name is diff. we have to provide it after options.

→ Then just run the Image built!

Q. How to run and interact with containers that do not immediately exit?

↳ Something like a server

Let's try running a server.Dockerfile ↗ doesn't exit!

> docker build --file server.Dockerfile --tag server .
↳ as the file name is diff. ↗ alias

> docker run server ↗ attaches the container to our terminal
↳ there is a problem here!

server need "ctrl+c" to stop. But by default, docker containers are not interactive.

- what to do then ??

> docker run -d server ↗ run without attaching to terminal
↳ detach, run in background

then → > docker exec ...

- docker exec [OPTIONS] CONTAINER COMMAND --[ARG...] ↗ to run command within container

↳ Useful options:-

-i, --interactive → keep STDIN open

-t, --tty → Allocate a pseudo-TTY

∴ We can use exec with options to interact with containers.

* Troubleshoot *

How to stop uninteractive containers that are running like a server?

Open a new terminal →
run > Docker ps, and get it's C-ID
run > Docker kill C-ID, to kill the respective container.

Q. How to stop and remove container?

> docker stop [OPTIONS] CONTAINER [CONTAINER...]

↳ useful options

- -s, --signal string → signal to send to container
- -t, --timeout int → secs to wait before killing the container

*Note:-

This command might take some time to stop a container.

To mitigate this, we can forcefully stop a container.

> docker stop -t 0 C-ID

Q Then, how to remove stoped containers from ps?

> docker rm C-ID
↳remove

This won't remove a running container.

To do that:-

> docker rm -f C-ID
↳ removes a running container by steping them first with SIGKILL

Q How to remove a list of Containers?

We can run a loop to remove all containers using there ID.

pipe

> docker ps -aq | xargs docker rm

docker ps -aq : only listID

xargs : take each element from left and run given command.

Q How to remove Docker images?

> docker rmi IMAGE

* Note :-

To stop a running image, we need to stop it first.

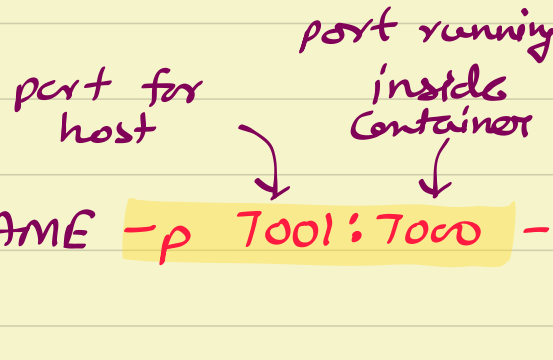
Q How to access container network services?

We can use port binding.

So, by default if a server is running inside a container we won't be able to access it as our host has no access point.

i.e. if the server is running on lc:7000, we won't be able to see it on our machine.

For this's to work :-



Now we can access our server from lc:7001 from host browser.

Q How to make sure that the data remains even when the container is removed?

We can use volume mounting.

By default, all the data inside the container is deleted when the container is removed.

To change this, we can mount a file/folder outside the container, which will store all the copy of data inside the container its mounted.

mounted

> docker run -v tmp/file_outside : tmp/file_inside CONTAINER

Now, if the container makes any changes in the file_inside, they will all be reflected in the file_outside.

Even after the container is removed

- -v out : in

or

--volume __ : __