# All notes

## Sec 3 - Working with Docker

Setup the Docker desktop

Managing venv is very important and it can be done via conda/pip

### File contents:

1. Dockerfile: This has instructions to execute the env

L1: FROM <https://stackoverflow.com/questions/54000157/purpose-of-from-command-docker-file>

The alpine is the Linux OS version

L2: COPY <https://phoenixnap.com/kb/docker-add-vs-copy>

COPY <src> … <dest>

We basically copy all files in the folder and create an app/ folder on Docker

L3: Set this as a working directory

Rest are project dependent

1. Other files for our app. Like python and requirements file

### 2. Build a Docker app

We need to build Docker image and Container

**Docker image**

Tag and add all files for our image

docker build -t <image-tag> .

Check all the images built on Docker with

docker images --all

Now, on Docker Desktop, we can only see it in the image, not in a container. So, we now have to run it in a container and set the right port to it

docker run --name <container-name> -p 5000:5000 <image-tag>

--publish , -p | Publish a container’s port(s) to the host

### Notes -

### Common Docker Commands

Cheatsheet: <https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf>

* Get info about the container

docker ps

* Get useful commands

docker exec --help

* Get Debugging useful tools

docker logs --help

* How to connect back to a container?

On Docker container, you can open on browser or CLI by using 1st two icons on Docker container.

* Delete container and images commands respectively

docker rm

Docker rmi

* Docker Hub login

docker login

* Stopping a container

docker kill

## Section 4 - DockerHub

Easy sharing of images

### Docker Push Image

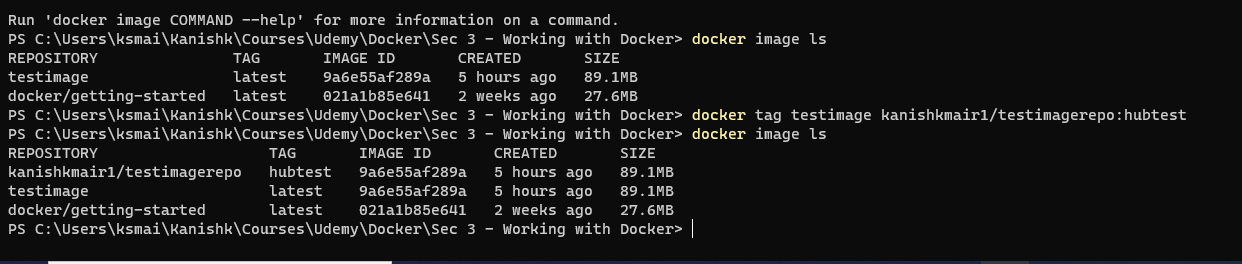
First create a repo on the UI to push our image. For example, we create ***testimagerepo***

Refer: <https://docs.docker.com/engine/reference/commandline/tag/>

docker tag <imagename> <username>/<commitreponame>:<tagname>

We can tag an image to our repo or image w/ a tag to the repo (for later, add a colon and the tag name)

On running ***docker image ls*** we get to see our commit and the tag name

We have repo with the given tag added above

Finally, push the commit to DockerHub

docker push <username>/<commitreponame>:<tagname>

### Docker Pull Image

Find trusted images from Docker Hub on Explore. For ex: We pulled Alpine in the very beginning

docker pull <reponame>

### Update Image

To update our solution, let’s say we update our code. The recommended way is we:

* **Let the Docker container stay immutable as it is**
* **Make changes only to the Docker image and commit changes to it**

### Extending an Image

After pulling, we want to extend code on the image we pull. This can be done directly in our **Dockerfile** to build our base image. Ex:

FROM <username>/<commitreponame>:<tagname>

RUN …

This basically pulls the image first and then add our own extensions to it. We can check this image in our Docker Desktop.

### Volumes



Used to share information between containers.

Docker images are read only layers. We apply changes but initial changes can be lost if original container is deleted.

We can use:

1. Volumes (Preferred)
2. Mount

Volumes are easier to manage and back up across multiple containers

**Commands:**

<https://docs.docker.com/storage/volumes/>

# Create vol

docker volume create <volname>

# Check vol

docker volume ls

# Inspect vol for debugging

docker volume inspect <volname>

# Remove

Docker volume rm <volname>

Let’s setup a file in a volume

-it interactive mode

--name for our container

--mount a more explicit version of -v (volume cmd)

docker run -it --name=newvol --mount source=testvolume,destination=/testvolume alpine

Pulls the alpine image and the interactive mode allows us to communicate directly on CLI mode.

Running `ls` in this Alpine, we can see the testvolume is present in the image.

We can add some data in this volume. After closing and running the above *docker run -it…* command, we can find all our data still present.

## Sec 5 - Challenge

**To build a docker image with:**

requests

Scikit-learn

pandas

Tensorflow==1.0.0

Ubuntu:18.04

## Sec 6 - Automated Builds

Practical workflows use NGINX which is basically a web server for all load balancing, http cache, etc.

Auto-build feature allows building automatically once we push to Git. It is like Flask in the sense that it needs it a directory (ex: of HTML files) to serve.

We use **bind volumes in Docker** to do that. In <https://docs.docker.com/storage/volumes/> it is said in the link that:

”While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker.”

