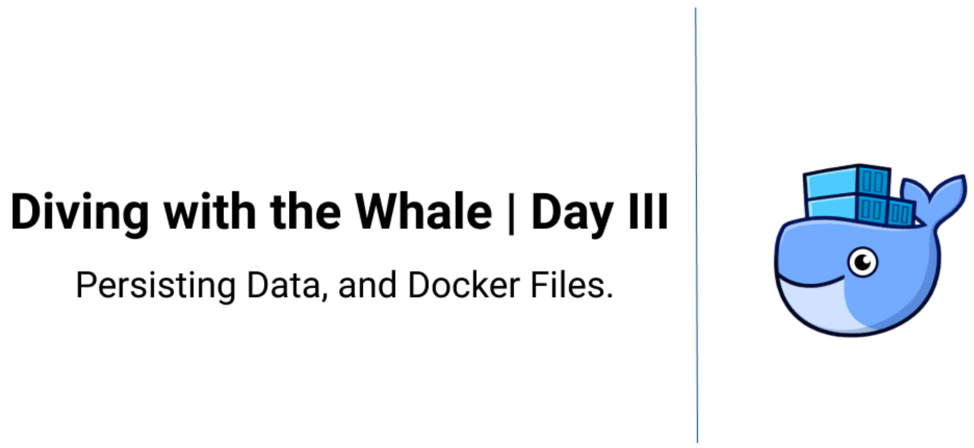

Diving with the Whale | Day III
Persisting Data, and Docker Files.

Diving with the Whale - Docker Day III - Persisting Data & Docker Files

Author : Ahmed Ayman

Remember when I said No persisting data with a really bold font? well I lied ! 😅

Persisting Data

Containers are designed to be ephemeral and isolated from the host machine. This creates problems when data is involved.

BUT, There is two ways around it actually

1. Mapping Volumes

Just the same as we mapped ports we can map directories from the host machine to a container directory, so anything happens here happens there and vice versa.

Let's say we want to run the Ubuntu image and map it's documents directory to a d

docker run -v <host-directory>:<container-directory> <image>

if the container directory does not exist, it'll create one for you

after you run the command any thing you put in this directory on the host machine appears in the container directory that's mapped to it, and vice versa!

```
pwd
# /home/ahmed
# create directory contained
mkdir contained
```

```
# docker run -v <host-directory>:<container-directory> <image>
docker run -it -v /home/ahmed/contained:/mnt/contained-vol ubuntu
# the proper way is to put it under /mnt/ but thats not really a rule
# -it to give us an interactive shell inside the container
# now check if the directory is actually there
cd /mnt/contained-vol/
# and WALAH!
# now let's create a file says "Hello From The CONTAINER Side" inside the container
echo "Hello From The CONTAINER Side" > hello.msg
# now let's head back to our host machine and check the message we got from inside this container
# ______________________________________________________________________________
# fire up a new terminal
# get inside the shared directory
cd /home/ahmed/contained/
ls
# hello.msg
# yaay it's there
# let's read it
cat hello.msg
# Hello From The CONTAINER Side
# NICE! let's say hi back
echo "Hi From The HOST Side!" > hi-back.msg
# let's head back to our container shell session to check if we got the message
# ______________________________________________________________________________
# get back to the container terminal
ls
# hello.msg  hi-back.msg
# let's read it, shall we?
cat hi-back.msg
# Hi From The HOST Side!
```

Now what happens if we stopped and removed the container? can you guess ?
HINT: Persisting!

Actually let's do it

```
# exit your container shell session
exit

# display all running containers
docker ps
# no thing running righ not

# display all containers
docker ps -a
# CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                     PORTS     NAMES
# 043808ca3998   ubuntu    "/bin/bash"   14 minutes ago   Exited (0) 25 seconds ago            gallant_pascal

# remove the container
docker rm gallant_pascal
# gallant_pascal

# display all containers again
docker ps -a
# CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                     PORTS     NAMES

# the container got removed
# now let's check our files in the contained directory
# get inside the shared directory
```

```
cd /home/ahmed/contained/
ls
# hello.msg  hi-back.msg
# both of the files still exists!
# we managed to persist data from docker container!
```

2. Docker Volumes

- the second solution is Docker Volumes, which is actually the preferred one
- These are volume mounts that are managed by docker

1. CREATE

   creates a new volume with the giving name

```
# docker volume create <name>
docker volume create contained
```

2. LS

   lists all the volumes on your system

```
docker volume ls
# DRIVER    VOLUME NAME
# local     contained
```

3. INSPECT

   gets info about the given volume

```
docker volume inspect contained
[
 {
    "CreatedAt": "2021-05-04T03:40:33+02:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/contained/_data",
    "Name": "contained",
    "Options": {},
    "Scope": "local"
  }
]
```

4. PRUNE

   removes all unused local volumes

```
docker volume prune
```

5. RM

   removes one or more volumes

   ```
   docker volume rm contained
   ```

So how to attach a volume with a container ?

```
# create a new-volume named contained
docker volume create contained

# docker run -mount type=volume,source=<volume_name>,target=<mount-point-inside-the-container> <image>
docker run -it --mount type=volume,source=contained,target=/mnt/mounted-vol ubuntu

# the proper way is to put it under /mnt/ but thats not really a rule
# -it to give us an interactive shell inside the container
# now check if the directory is actually there
cd /mnt/contained-vol/

# and WALAH!
# now let's create a file says "Hello From The CONTAINER Side" inside the container
echo "HELLO FROM THE CONTAINER, I'M ATTACHED TO THE VOLUME \"CONTAINED\"" > hello.msg

# now let's head back to our host machine and check the message we got from inside this container
# _______________________________________________________________________________
# fire up a new terminal
# get the mount point by inspecting the volume
docker volume inspect contained | grep -i "mountpoint"
# "Mountpoint": "/var/lib/docker/volumes/contained/_data"

# now we know where the shared data is
# let's get in there
cd /var/lib/docker/volumes/contained/_data
ls
# hello.msg

# read the file content
cat hello.msg
#HELLO FROM THE CONTAINER, I'M ATTACHED TO THE VOLUME "CONTAINED"

# NICE! let's say hi back
echo "Hi From The HOST Side!" > hi-back.msg

# let's head back to our container shell session to check if we got the message
# _______________________________________________________________________________
# get back to the container terminal
ls
# hello.msg  hi-back.msg
# let's read it, shall we?
cat hi-back.msg
# Hi From The HOST Side!
```

TASK

DO THIS TASK USING THE TWO OPTIONS, VOLUME MAPPING AND DOCKER VOLUMES

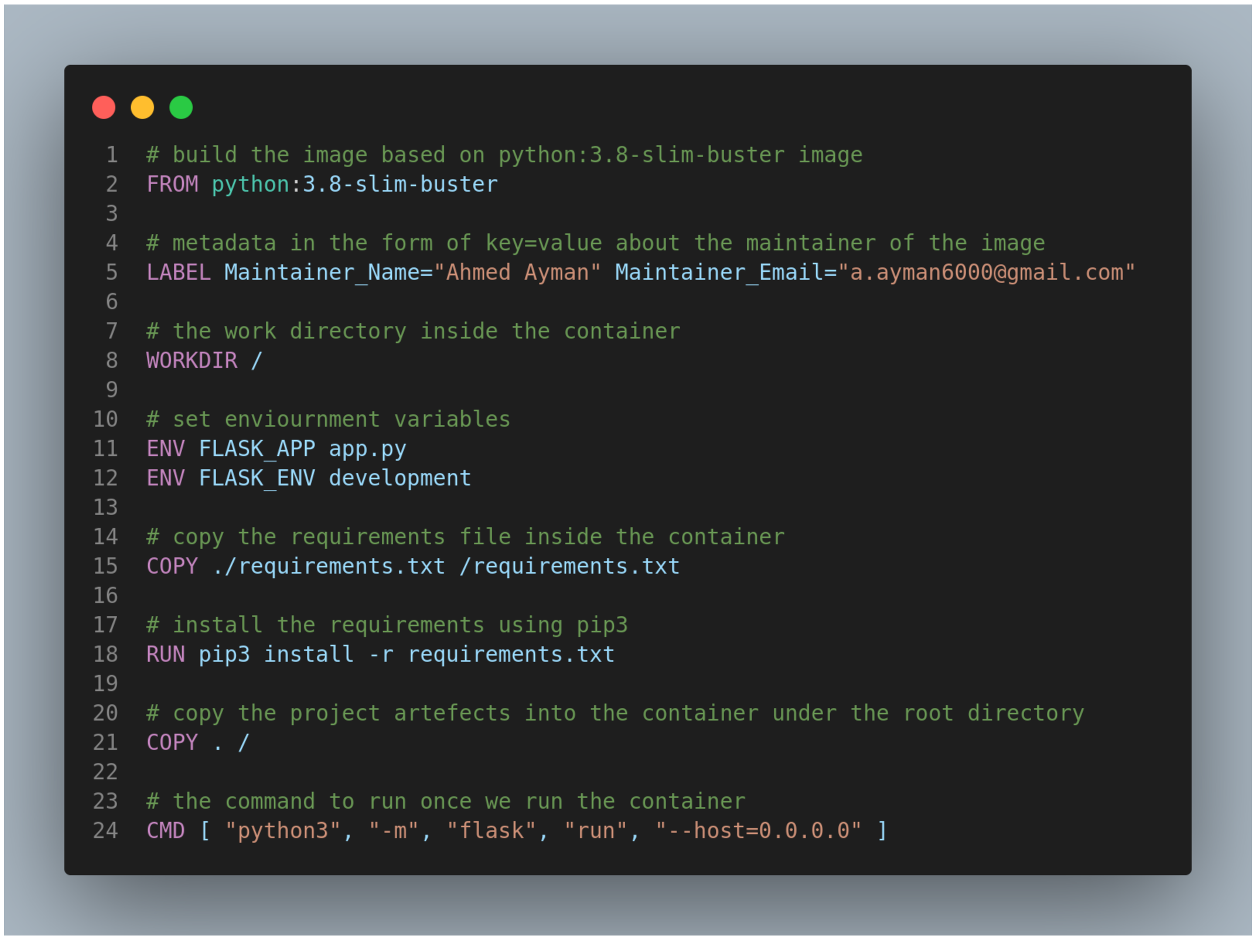
1. setup a script to send a log file (create a cron-tab for example or copy log files every 1 minute) to the mapped volume or the docker volume

Here we are! Docker Files

Dockerfile Reference

Docker File Best Practices

Docker files are specific instructions on how to build the docker image, things like libraries, environments, system updates, copying project files, and basically everything Docker Engine needs to build up the docker-image.



```dockerfile
# build the image based on python:3.8-slim-buster image
FROM python:3.8-slim-buster

# metadata in the form of key=value about the maintainer of the image
LABEL Maintainer_Name="Ahmed Ayman" Maintainer_Email="a.ayman6000@gmail.com"

# the work directory inside the container
WORKDIR /

# set enviournment variables
ENV FLASK_APP app.py
ENV FLASK_ENV development

# copy the requirements file inside the container
COPY ./requirements.txt /requirements.txt

# install the requirements using pip3
RUN pip3 install -r requirements.txt

# copy the project artefects into the container under the root directory
COPY . /

# the command to run once we run the container
CMD [ "python3", "-m", "flask", "run", "--host=0.0.0.0" ]
```

Docker File Anatomy - Commands

usually the naming convention is **Dockerfile** or **Dockerfile.ubuntu**

FROM <image>

MUST BE THE FIRST COMMAND IN THE DOCKERFILE

ONLY ONE FROM COMMAND

Specify the image to base your image from so you wouldn't have to start from scratch.
For example in the above Dockerfile we're basing our image on a python image that would have most of the libraries already installed and the environment is ready, so we don't need to do this from scratch!

```
FROM ubuntu:18.04
```

RUN <COMMAND>

Runs Linux commands inside the container

For example what is the first thing you do when you acquire a new Linux machine?
YES you update and upgrade the repository! so that's a thing you would normally want the docker image to run as a command

```
RUN sudo apt update
```

LABEL <key=value>

adding metadata like the author data or info about the image

```
LABEL Author="Ahmed"
```

CMD [<"command">, <"argument-1">, ..]

ONLY ONE CMD COMMAND

Actually I lied, you can have more than one CMD command, but only the last one will get executed!

Sets defaults for running a container

it tells the engine exactly how to run a container from this image if now commands are specified (if any commands were specified it will overwrite the CMD command)

```
CMD ["python3", "app.py"]
```

ADD <host file(s)(or)directory(or)URL*> <path in the image*>

Copies files into the image from the host machine or remote URL into the image

```
ADD "https://www.notion.so/Diving-with-the-Whale-Docker-Day-III-97af6b7cda5249ab82df3656ea18e1da" /Documents
```

COPY <host file(s)(or)directory*> <path in the image*>

Copies files into the image from the host machine or remote URL into the image

```
COPY ~/myapp /
```

ENV <variable*> <value>

Sets environment variables inside the image

```
ENV DATABASE_USER="dbuser"
# we can use or don't use the = sign
ENV DATABASE_HOST "host.com"
```

EXPOSE <port*/protocol>

Expose a port to the outer world!

for example if we want to run a web-app we need to expose port 80 for HTTP and perhaps port 443 for HTTPS

```
EXPOSE 80/tcp
```

ENTRYPOINT <command*>

Configure the container to run as an executable

```
ENTRYPOINT ["/bin/bash"]
```

WORKDIR <path>

Sets the path of the work directory, basically this changes the working directory to the path you're giving to it

```
WORKDIR /var/www/html
# now all the next commands (until the next RUN cd or WORKDIR) will be executed inside this dir
```

it's the same as

```
# RUN cd <path>
RUN cd /var/www/html
```

BUILD IT!

now from our docker file we want to actually build the image that we can run to get a running container, so how can we build a docker file

```
# change your directory to the path where your Dockerfile is
cd project/
# run the build command
docker build -t flask-app:latest .
# -t to specify a tag for your image
# flask-app:latest is the tag we chose
# . is the path of the directory contains the docker file you want to build
```

TASK

- Create a bash script says hello from container!
- Then create a docker file based on Ubuntu, copy the script into the container and make the script run once you run the container.
- Build the image!
- Run and test your first image
- **errors**? debug and find the issue
- Run again
- **Congratulations You Created Your First Docker Image!** 🎉

Advanced Topics

Create your own base image from scratch