# Managing images and containers

Understanding the basics of how images and containers work is critically important to their effective use in Data Science projects. We'll cover some key concepts below, illustrating them with simple examples.

## Image management

Most often, the first step in the process of creating a new image is to gather the relevant data files and scripts in one place and test their correct operation.

In the next step, we create a Dockerfile, specifying the scope and configuration of running images.

In our example, we will start with the following Dockerfile:

```
#syntax=docker/dockerfile:1

FROM python:3.8-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD ["python3","app_1.py"]
```

Create the image with the command:

`docker build -t app1:v1 .`.

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker build -t app1:v1 .
[+] Building 15.8s (17/17) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 478B                                                                                0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => resolve image config for docker.io/docker/dockerfile:1                                                          9.2s
 => [auth] docker/dockerfile:pull token for registry-1.docker.io                                                    0.0s
 => CACHED docker-image://docker.io/docker/dockerfile:1@sha256:443aab4ca21183e069e7d8b2dc68006594f40bddf1b15bbd83f5  0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => [internal] load build definition from Dockerfile                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim-buster                                           2.0s
 => [auth] library/python:pull token for registry-1.docker.io                                                       0.0s
 => [1/5] FROM docker.io/library/python:3.8-slim-buster@sha256:ed38d1a70cd28a628a58cecaa87bb95dba7180fffc0afd314f9f  4.3s
 => => resolve docker.io/library/python:3.8-slim-buster@sha256:ed38d1a70cd28a628a58cecaa87bb95dba7180fffc0afd314f9f  4.3s
 => [internal] load build context                                                                                   0.0s
 => => transferring context: 88.43kB                                                                                0.0s
 => [auth] library/python:pull token for registry-1.docker.io                                                       0.0s
 => CACHED [2/5] WORKDIR /app                                                                                       0.0s
 => CACHED [3/5] COPY requirements.txt requirements.txt                                                             0.0s
 => CACHED [4/5] RUN pip3 install -r requirements.txt                                                               0.0s
 => [5/5] COPY . .                                                                                                  0.0s
 => exporting to image                                                                                              0.0s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:5eb1ce1e69d53569195cfe466eddd8a08c3f334ab60006034e0cfc872ccf4e3b                        0.0s
 => => naming to docker.io/library/app1:v1                                                                          0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami %
```

We can now check if a new image has appeared on the system using the command (we use this to redirect the pipeline to the `grep` function to show only images named `app1`) :

`docker images | grep app1`.

As you can see, a new image has appeared on our system.

We can also delete images (`docker images rm < container ID>`), rename them, etc.

# Container management

Command:

`docker ps`.

displays all **actually running** containers.

To see all containers on the system, including stopped ones, you need to run this command with the -a flag:

`docker ps –a`.

Running these two commands on a "clean" system should return an empty result.

Let's see what happens when we run our image using the command:

`docker run –ti app1:v1`.

The program runs correctly:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker run -ti app1:v1
... App 1 Started ...

Original df:
   0   1    2
0  1  32   10
1  3   4  315

Random number:  1

Transformed df:
   0   1    2
0  1  32   10
1  3   4  315

... App 1 Completed ...
```

The `docker ps` command returns an empty result, but already `docker ps –a` displays on the screen:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND             CREATED             STATUS                          PORTS     NAMES
fb008f913850   app1:v1   "python3 app_1.py"   About a minute ago   Exited (0) About a minute ago             brave_hertz
```

As you can see, the **container after running has stopped**: it is no longer active (empty `docker ps` message), but it exists in the container repository.

This is because in the last line of our Dockerfile, we called the *CMD ["python3", "app_1.py"]* command to run the Python script: the container gets a signal to stop after the script is executed correctly. The absence of this command:

1. on the one hand, it would not run the app_1.py script. Its launch would only be possible "from inside" the container
2. ... But on the other hand, it does not stop the container from running.

Now let's try to run the container again, using once again the command:

`docker run -ti app1:v1`.

As before, the `docker ps` command returns an empty result, but `docker ps -a` displays on the screen:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND             CREATED          STATUS                      PORTS     NAMES
902771bee841   app1:v1   "python3 app_1.py"   37 seconds ago   Exited (0) 35 seconds ago             competent_heisenberg
fb008f913850   app1:v1   "python3 app_1.py"   4 minutes ago    Exited (0) 4 minutes ago              brave_hertz
```

This shows that **the effect of restarting the container with the `docker run ...` command was to create a new container.

An important observation follows: **multiple use of the `docker run ...` function generates multiple containers, separate for each run.

After some time, there may be so many containers that it is necessary to remove them. This can be done "manually" using the `docker container rm <containerID>` command:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker container rm 902771bee841
902771bee841
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS                     PORTS     NAMES
fb008f913850   app1:v1   "python3 app_1.py"   9 minutes ago   Exited (0) 8 minutes ago             brave_hertz
```

In an extreme version, you can also use the `docker container prune` command to remove all containers.

# Restarting stopped containers

To **start a stopped container again**, run:

`docker container start < container ID>`:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS                     PORTS     NAMES
8b732fc483b6   app1:v1   "python3 app_1.py"   3 seconds ago   Exited (0) 2 seconds ago             lucid_zhukovsky
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker container start -a 8b732fc483b6
... App 1 Started ...

Original df:
   0   1    2
0  1  32   10
1  3   4  315

Random number:  5

Transformed df:
    0    1     2
0   5  160    50
1  15   20  1575

... App 1 Completed ...
```

Here we used the -a (attach) flag to be able to display messages from the container.

As before, after running it is stopped, **and all the data it generates is lost.** We'll talk about how to preserve it in the section on file exchange between containers and the host.

# Running a command inside a running container.

To run a command inside a container, you can use the EXEC command:

`docker exec -ti < container ID> COMMAND`.

Note: the example below shows that **this is only possible when the container is running**:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker ps -a
CONTAINER ID   IMAGE     COMMAND             CREATED         STATUS                      PORTS     NAMES
8b732fc483b6   app1:v1   "python3 app_1.py"   2 minutes ago   Exited (0) 2 minutes ago            lucid_zhukovsky
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker exec -ti 8b732fc483b6 sh ls
Error response from daemon: Container 8b732fc483b655640a0b51e31c60e1b5edf0f368acc74f849b2a2be832ffdc8b is not running
```

How to cause our container not to be closed immediately after startup?

We have presented the first option above: **we just make sure that our Dockerfile does not contain the command that runs the script** at the end, as in the example below:

```
#syntax=docker/dockerfile:1

FROM python:3.8-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .
```

Another possibility is **running the container with the bash command at the end of the command line**:

`docker run -ti app1:v1 bash`.

With it, we start the container and get access to its bash shell. While in it, we can run our script, view and modify files, etc:

```
(SUML2) wodecki@iMac-iMac 5. demo - zarządzanie obrazami i kontenerami % docker run -ti app1:v1 bash
root@619a5c855731:/app# ls
 Dockerfile  'Podstawowe pojęcia.md'  'Zarządzanie obrazami i kontenerami.md'   data    requirements.txt
'Icon'$'\r'   README.md                app_1.py                                 media
root@619a5c855731:/app# python app_1.py
... App 1 Started ...

Original df:
   0   1    2
0  1  32   10
1  3   4  315

Random number:  1

Transformed df:
   0   1    2
0  1  32   10
1  3   4  315

... App 1 Completed ...

root@619a5c855731:/app# more data/input_1.csv
1,32,10
3,4,315
root@619a5c855731:/app#
```

This time the `docker ps` command already indicates that our container is active:

```
(base) wodecki@iMac-iMac ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED              STATUS              PORTS     NAMES
619a5c855731   app1:v1   "bash"    About a minute ago   Up About a minute             priceless_dhawan
(base) wodecki@iMac-iMac ~ %
```

As a result, you can already run various programs in it using the `exec` command:

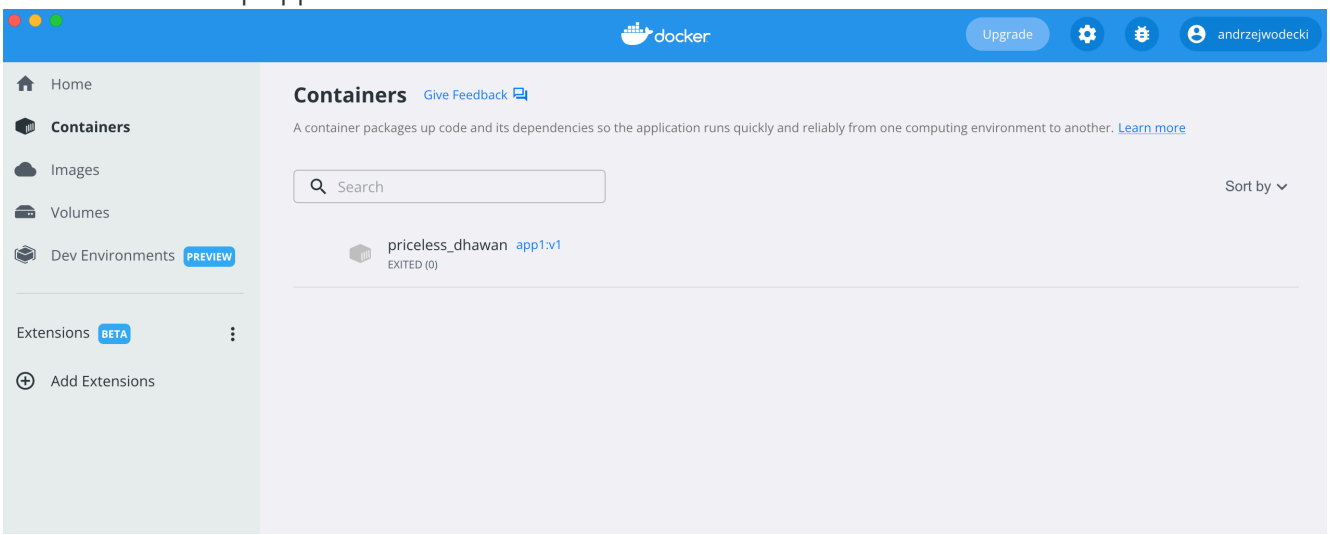
```
(base) wodecki@iMac-iMac ~ % docker exec -ti 619a5c855731 sh -c ls
 Dockerfile                          app_1.py
'Icon'$'\r'                          data
'Podstawowe pojęcia.md'              media
 README.md                           requirements.txt
'Zarządzanie obrazami i kontenerami.md'
(base) wodecki@iMac-iMac ~ % docker exec -ti 619a5c855731 sh -c "python app_1.py"
... App 1 Started ...

Original df:
   0   1    2
0  1  32   10
1  3   4  315

Random number:  3

Transformed df:
   0   1    2
0  3  96   30
1  9  12  945

... App 1 Completed ...
```

This is convenient in that **after each such run I return to our local shell**. This can have very interesting applications in production runtime (managing container launches via shell scripts).

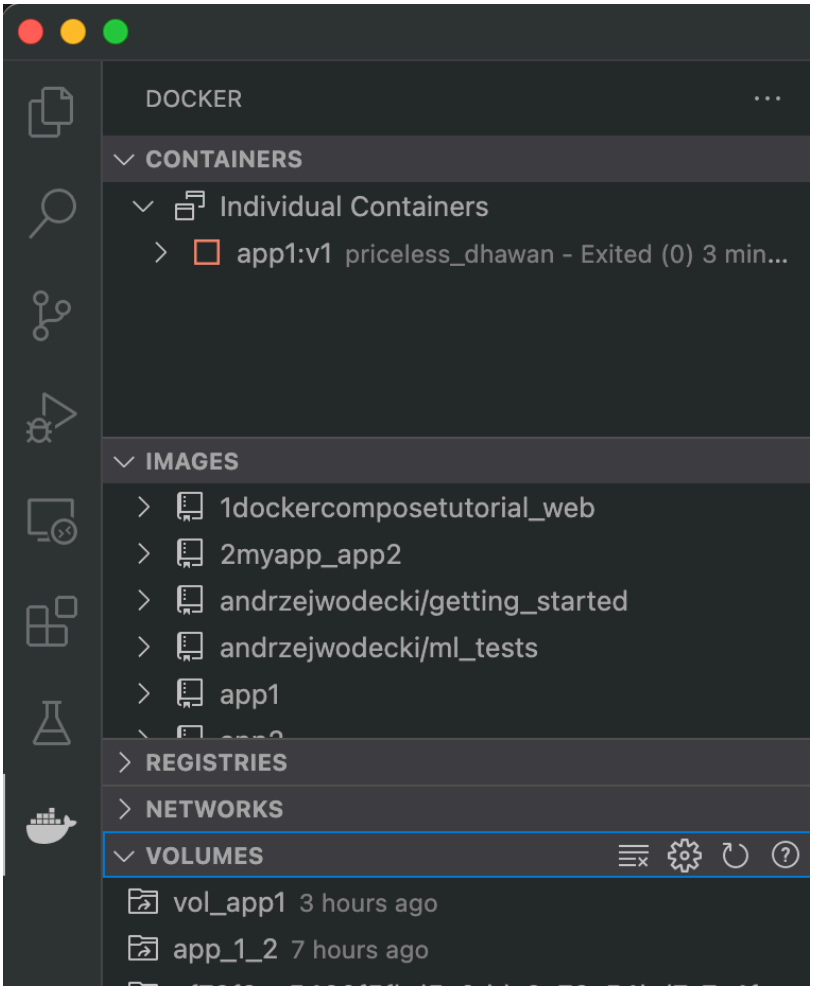
# Managing images and containers using Docker Desktop and IDE applications.

Command line is not the only way to inspect and manage images and containers. You can successfully use:

1. the Docker Desktop application:




2. IDE environment, such as MS Visual Studio Code:


DOCKER
CONTAINERS
Individual Containers
app1:v1 priceless_dhawan - Exited (0) 3 min...
IMAGES
1dockercomposetutorial_web
2myapp_app2
andrzejwodecki/getting_started
andrzejwodecki/ml_tests
app1
REGISTRIES
NETWORKS
VOLUMES
vol_app1 3 hours ago
app_1_2 7 hours ago