Sharing data between containers with `volume mount`

In this part of our lecture, we will show how to provide data sharing between **multiple containers** using the **volume mount** mechanism.

Creating images and running individual containers

In our example, we will create and run 2 containers: `app1_shared` and `app2_shared`.

Both do a simple transformation of data read from an external file, with `app2_shared` treating the result of `app1_shared`'s action as input:

```
import pandas as pd
import random

import os

print('... App 2 Started ...\n')

input_path = 'data/output_1.csv'

if os.path.isfile(input_path):
    df = pd.read_csv(input_path, header = None)
else:
    df = pd.DataFrame([[2, 43, 510], [3, 4, 315]])
    df.to_csv(input_path, index = False, header = False)

print('Original df:')
print(df)

n = random.randint(1,5)

print('\nRandom number: ', n)

df_out = df + n

output_path = 'data/output_2.csv'

df_out.to_csv(output_path, header=False, index=False)

print("")
print('Transformed df:')
print(df_out)

print('... App 2 Completed ...\n')
```

To create the `app1` image, in the `app_1` folder, run the command in terminal:

`docker build -t app1_shared:v1 .`

To run it:

`docker run -ti app1_shared:v1`.

Similarly, to create an image of `app2`, in the `app_2` folder, run the command in the terminal:

```
docker build -t app2_shared:v1 .
```

To run it:

`docker run -ti app2_shared:v1` .

To create a shared volume

The *volume mount* functionality allows you to create a local but docker-managed shared volume. Let's create such a volume using the command:

`docker volume create vol_app1_app2`.

Running individual containers with data exchange functionality

To launch individual applications in data exchange mode using the volume mount mechanism:

1. For `app1_shared` application, at the command line, run:

`docker run -ti -v vol_app1_app2:/app/data app1_shared:v1`.

2. for `app2_shared` application at the command line run:

`docker run -ti -v vol_app1_app2:/app/data app2_shared:v1`.

Note that running these applications sequentially causes the results of previous calculations to be taken into account as input for subsequent ones:

```
(base) wodecki@iMac-iMac 3.5. Kilka kontenerów z volume mount % docker run -ti -v vol_app1_app2:/app/data app1_shared:v1
... 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 ...

(base) wodecki@iMac-iMac 3.5. Kilka kontenerów z volume mount % docker run -ti -v vol_app1_app2:/app/data app2_shared:v1
... App 2 Started ...

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

Random number:  3

Transformed df:
    0   1    2
0   6  99   33
1  12  15  948
... App 2 Completed ...
```

Use cases

You can use the above mechanism in particular in the following situations:

I want to ensure the continuity of my application.

... so that subsequent runs of the container with my application can use the results of previous runs

I use docker volume for this purpose.

I want different containers running in parallel to exchange data.

I use docker volume for this purpose.

I want to store the data used by my containers with a cloud service provider.

I use docker volume for this purpose.

Useful resources

A very good presentation of the *bind mount* and *volume mount* methods can be found in the official docker documentation available here.