Multi-container applications

Docker allows you to create multi-container applications that share data with each other.

They can be run independently or sequentially.

The **docker compose** module is used to design and run such applications, while the configuration specification is stored in the `docker-compose.yaml` file.

In our case, this file has the following structure:

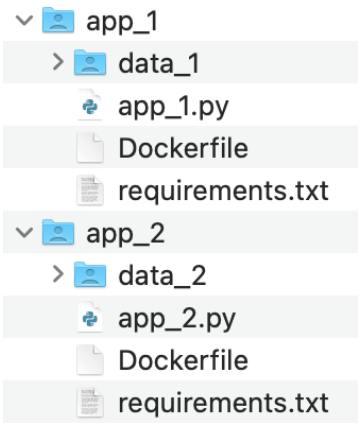
```yaml
version: "3.9"

volumes:
  shared_data:

services:
  app1:
    build: app_1/.
    container_name: app1
    volumes:
      - shared_data:/app/data_1
  app2:
    build: app_2/.
    container_name: app2
    #depends_on:
    #  - "app1"
    volumes:
      - shared_data:/app/data_2
```

Note:

1. Scripts `app1` and `app2` use differently named directories, `/data_1` and `/data_2` respectively



2. At the beginning of the `docker-compose.yaml` file, we defined a shared volume `shared_data` (you can read about this and other file sharing mechanism between containers, e.g. here).
3. In the specification of `app1` and `app2` services, we indicated the mechanism for mapping the docker volume with the volumes of each container.

Building and running

To build and run such an application, at the command line run:

`docker compose up`.

To run one of the selected applications:

`docker compose run app1` < runs *app1*.

`docker compose run app2` < runs *app2*.

As a result, the results of sequentially running the services show that the data is propagated correctly:

```
(base) wodecki@iMac-iMac ! newly % docker compose run app1
... 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 ...

(base) wodecki@iMac-iMac ! newly % docker compose run app2
... App 2 Started ...

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

Random number:  4

Transformed df:
    0    1     2
0   9  164    54
1  19   24  1579
... App 2 Completed ...
```

To run in terminal detached mode:

`docker compose up -d`.

In this case, to stop applications:

`docker compose stop`.

`docker compose down --volumes` < also delete all shared volumes.

To check the list of active docker compose processes:

`docker compose ps`.

If you have modified the source files in the meantime (e.g. Python scripts), rebuild the corresponding images:

`docker compose build`.

`docker compose` vs `docker-compose`

NOTE: in the latest versions Docker implemented `compose` module directly in the package. As a result, it can be run as `docker compose ...`.

In older versions, using this service is possible after installing a separate `docker-compose` library, while running it requires typing `docker-compose ...` (< note the `-` sign in the middle).

You can also configure this in Docker settings:

Useful resources

Documentation of docker compose functionality can be found here.

A very good description of file exchange between applications can be found here.