How to install ASELA Data Catalogue Dashboard using Docker Desktop

About

The *ASELA Data Catalogue Dashboard* is a first step toward developing an urban Digital Twin that has fully advanced analytical, visualisation and information governance capabilities. The objectives of the ASELA Data Catalogue are to:

- Provide a list of strategically relevant datasets, including details on provenance, access and contents;
- Illustrate how those datasets can be used for monitoring and evaluation as part of evidence-based policy and planning;
- Enable data and information sharing, management and visualisation.

Supporting these goals, the *ASELA Data Catalogue* has three core components:

- an inventory of more than 120 datasets pertaining to 11 themes, such as the economy, environment or housing;
- a specification of 25 key performance indicators (KPIs) that may be derived from the data residing in the datasets; the KPIs pertain to seven strategically important policy goals;
- an interactive user interface to maintain, update and visualise the datasets, KPIs, related themes and geographic sites of interest, alongside 3D visualisation functions.

We use Docker to easily deploy and run *ASELA Dashboard* on different Operating Systems (Windows, macOS and Linux) and infrastructures. *ASELA Dashboard* relies on two distinct and connected Docker containers:

- **Postgres DB with PostGIS extension**, the data access layer of the Dashboard. It contains the initial datasets, and the datasets uploaded by the users, within a predefined queryable schema. Both spatial and non-spatial data are supported;
- **ASELA ShinyApp**, the modular UI used to access and update the data catalogue developed using RShiny and accessible through any major internet browser (e.g. Safari, Chrome, Edge, Firefox).

Software and Hardware Requirements

- Docker Desktop available for Windows and MacOS
- VSCode available for Windows and MacOS
- Mapbox account and a MapboxAPI key

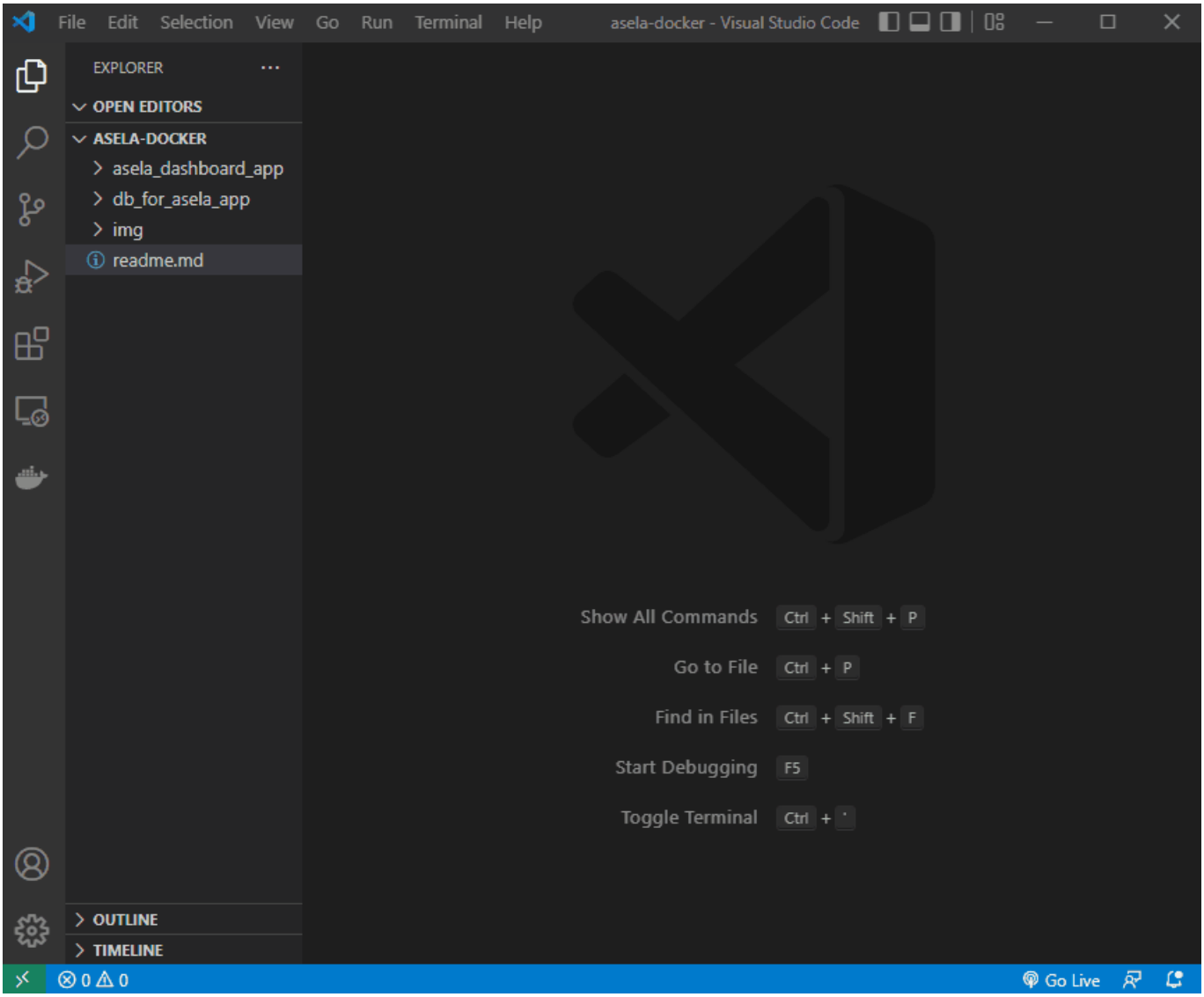
> Note: Docker Desktop is a user-friendly interface to manage and run Dockers containers. Larger enterprises (more than 250 employees OR more than $10 million USD in annual revenue) require a paid subscription. The commercial licence is limited to the use of Docker Desktop, which is not essential to run the application of any of the supported operating systems.

Any recent and updated system (Windows 10 or 11, macOS version 10.15 or newer) is able to run the required software and *ASELA Data Catalogue*. Internet connection is required during the installation.

1 - Download the source code

Follow the installation instruction for Docker and VSCode for your operative system.

If you are reading this guide online, clone and unzip this repository in a new folder (e.g. `asela-docker`) on your system (e.g. Desktop). You should have two sub-folders: `asela_dashboard_app` and `db_for_asela_app`. Open `VSCode` and `File ->Open Folder` to select the main folder of the project, in this example `asela-docker`. Open a new *Terminal session* from the menu `Terminal -> New Terminal`.



2 - Create the Postgres Database Docker

Before creating the Postgres Docker, write down the name of the container and the Postgres password, you will need them in the following steps. In this example:

- Name of the container: some-postgis-container
- POSTGRES_PASSWORD: mysecretpassword

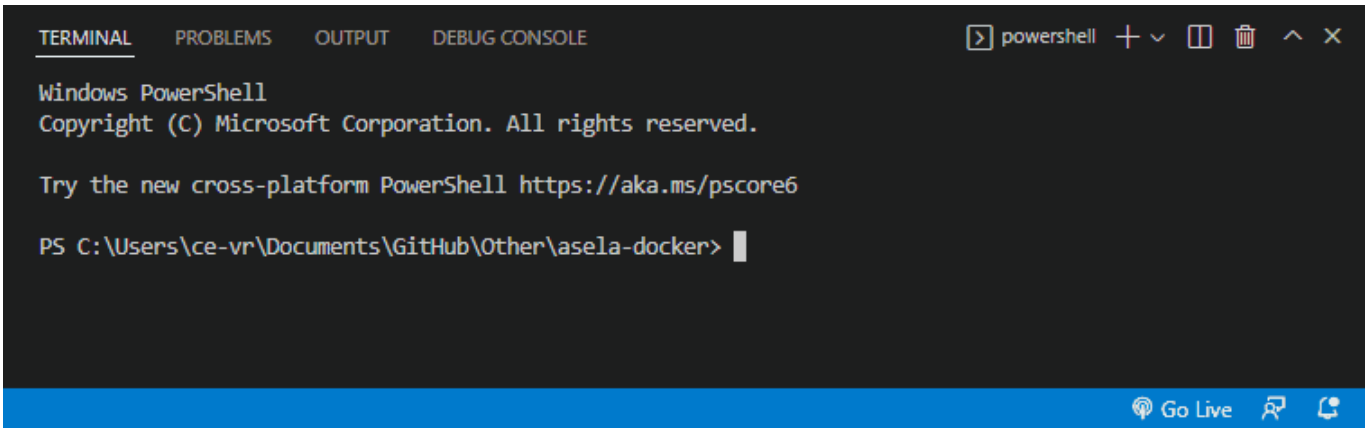
> Note: store the password in a secure location

First, navigate to the `db_for_asela_app` using the *Terminal session* opened in `VSCode`

```
cd db_for_asela_app
```

run the following docker command

```
docker run --name some-postgis-container --restart always -e
POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 -v postgres-
data:/var/lib/postgresql/data -d postgis/postgis
```



This command creates and runs a new Docker container named `some-postgis-container`; it restarts the container in case it stops (e.g. reboot of the machine. It is not essential in the local environment); it sets the password to the Database `mysecretpassword`; it also exposes the port 5432 of the container; it creates a persistent volume named `postgres-data`. The container is created using the image `postgis/postgis` located on the Docker Hub.

3 - Set up the Database

Create a new database in the running postgres container (in this example is named `some-postgis-container`). Run the following command in the *Terminal session*

```
docker exec -ti some-postgis-container psql -U postgres
```

This command is opening a `psql` console in the same *Terminal session* using the user `postgres` (the terminal should show the version of psql and the new prompt `postgres=#`). Create the database with the name `asela_dash` and enable postgis extension on it

```
CREATE DATABASE asela_dash;
```

Change your psql CLI to be operating within that database

```
\c asela_dash
```

Enable PostGIS extension on that database

```
CREATE EXTENSION postgis;
```

> Note: exit the *psql* console using `\q`

4 - Restore or add entries to the Database

The easiest way to add data to the Database is to perform a restore of a Database dump file if exists (the dump file is not shared on the online repository, but it is available on the archive version provided via a secure link).

The Database dump file needs to be copied from the host to the Docker container. The destination folder inside the Docker container must exist (e.g. `/tmp` or `/var/backups`). Run the following command in the *Terminal session* of `VSCode`

```
docker cp your_db.dump some-postgis-container:/tmp/your_db.dump
```

Once the process is completed, log into the docker container in an interactive bash session using the following command (The result is a new prompt `root@xxxxxx:/#`)

```
docker exec -ti some-postgis-container bash
```

Navigate to the directory that contains the Database dump file (e.g. `cd /tmp` or `cd /var/backups`) and restore it using `pg_restore command` (this process might take some minutes depending on the size of the Database)

> Note: This command will strip out all ownership and privilege information. You will need to add all the users and roles necessary later as instructed in step *5*.

```
pg_restore -U postgres -d asela_dash -j 3 -O -x your_db.dump
```

for example

```
pg_restore -U postgres -d asela_dash -j 3 -O -x /tmp/asela_dash_db.dump
```

Exit the Docker bash using `exit` in the bash command line.

5 - Roles and Users

There are two roles on the Database, one with writing access and one with reading only access. Roles and access are created and added using the bash script `psql_grant_user_readonly_command_writer.sh` and the text file `all_schema_names.txt` (the text file contains the Database schema for `asela_dash`).

Copy both files from the host system (current folder `db_for_asela_app`) in the same folder of the docker container (e.g. `/tmp/`)

```
docker cp ./psql_grant_user_readonly_command_writer.sh some-postgis-
container:/tmp/psql_grant_user_readonly_command_writer.sh
```

```
docker cp ./all_schema_names.txt some-postgis-container:/tmp/all_schema_names.txt
```

Enter the console bash of the Docker container

```
docker exec -ti some-postgis-container bash
```

run the command

```
cd /tmp
```

and run the script

```
sh psql_grant_user_readonly_command_writer.sh
```

> Note: The privilege granter will have to be rerun if there are ever changes to the database schema in future.

Exit the Docker bash using `exit` in the bash command line.

To add users with connecting privileges and passwords and give them read-only access use the command below, but substitute names for real usernames and a proper password.

From the psql console of the Docker container

```
docker exec -ti some-postgis-container psql -U postgres
```

run the command

```
CREATE USER all_asela_user WITH PASSWORD 'password_here';
```

Allow it to connect to the db:

```
GRANT CONNECT ON DATABASE asela_dash TO all_asela_user;
```

For writing access inherit `all_data_dash_user` privileges to the new user. For readonly access inherit `all_data_dash_reader` privileges to the new user.

```
GRANT all_data_dash_user TO all_asela_user;
```

> Note: exit the *psql* console using `\q`

6 - Create a shared private networks

The shared private network is used to connect Docker containers between them, in this case, the Postgresql Docker Container with the Dashboard Docker Container (that will be created in the next step). Create a Docker network using the command `docker network create NAME_OF_THE_NETWORK`, e.g.:

```
docker network create asela-docker-net
```

then, connect the Postgres container to the network using the command docker network connect `NAME_OF_THE_NETWORK NAME_OF_THE_CONTAINER`

```
docker network connect asela-docker-net some-postgis-container
```

7 - Create Dashboard Docker image

The source code to install the dashboard can be found in the second sub-folder `asela_dashboard_app`. For security reasons, the login data file has not been added to the repository. You will need to create a new file named `.Renviron` (note the `.` is part of the filename) that contains login data used for the Postgres Docker Container. It will also need an API key from Mapbox for serving the basemap. One API key will work for the entire app, and as long as the app isn't used really heavily it will always stay under the free level. Sign up and get the key here: https://www.mapbox.com/. The `.Renviron` file should have the following parameters in it (values might be different depending on the previous steps, e.g. DB Name, UserID and Password of the DB, name of the container).

In `VSCode`, right click on the folder `asela_dashboard_app` on the left-hand side and create a `New File` named `.Renviron` and add the following information

```
#mapbox api token
MAPBOX_API_TOKEN_OWNBRAND=the_api_created_in_the_MAPBOX_website

#db connection info

#DATABASE AND USERNAMES
USERDBNAME=asela_dash
USERUSERNAME=all_asela_user
USERPASSNAME=password_here


#Shared private networks
DOCKHOSTNAME=some-postgis-container
DOCKPORTNO=5432
```

> Note: The values added to the `.Renviron` **MUST** match the ones used previously to create the PostgreSQL container.

Save the file and build the Docker image. If the Docker container needs to run in a *ShinyProxy* environment (this is generally the case if the system is running on a remote server), the name of the image, defined by the option `-t` must be the same as the one defined in the *shinyproxy* `application.yml` file. If the container run on the local environment, e.g. using Docker desktop, any name and tag of the Docker image will work.

Run the following command in the folder `asela_dashboard_app`

> Note: if the prompt shows the previous folder `.\asela-docker>`, first go back to the main directory using `cd ..` and then in the `asela_dashboard_app` using `cd asela_dashboard_app`

```
docker build -t name_of_your_image_here -f Dockerfile .
```

e.g.

```
docker build -t asela_r_app_shinyserv_ver:latest -f Dockerfile .
```

> Note: the first time the process can last up to 30 minutes or more, depending on the system used

Finally, we can create and run the container `asela-dashboard` using the built image `asela_r_app_shinyserv_ver:latest` on the shared network `asela-docker-net`); the env-file `.Renviron` contains the login information.

```
docker run --name asela-dashboard --net asela-docker-net --env-file ./.Renviron -
it -p 3838:3838 asela_r_app_shinyserv_ver:latest
```

8 - Run and stop the Asela Dashboard Docker

At this point, the dashboard is accessible from any browser at the address `http://localhost:3838`

To stop the dashboard exit Docker Desktop or, from Docker desktop, stop the two containers created. From Docker Desktop interface it is also possible to remove the application by deleting, in this order, the two docker containers, the two images and the volume.