PART2 : CONTAINERIZING THE APPLICATION

The multi-tiered toy store is implemented as microservices (a front-end tier and a back-end tier).

Build an Image:

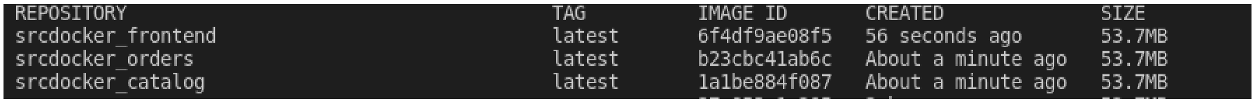
As the application is running, we have to create Dockerfiles for each service(Front-end service, Order service, Catalog service) to build images. Each Dockerfile has the instructions to assemble a Docker image. By executing the ***docker build*** command, we instruct Docker to create the Docker image by reading and executing the instructions specified, and creating a Docker image as a result. For example, to build catalog image, we simply run :

```
docker build . -t catalog
```

To see the list of images, we have to run the below command :

```
docker images
```

As part of the multi-tiered toy store, the docker images should be created for all the 3 services and should be displayed as part of the docker images.



```
REPOSITORY                                    TAG        IMAGE ID       CREATED              SIZE
srcdocker_frontend                            latest     6f4df9ae08f5   56 seconds ago       53.7MB
srcdocker_orders                              latest     b23cbc41ab6c   About a minute ago   53.7MB
srcdocker_catalog                             latest     1a1be884f087   About a minute ago   53.7MB
```

Fig 1: Docker images

Run Containers:

To check if the application is working as expected, we will run these images inside the containers. In Docker terms, a container is a runnable instance of an image. It can be created, started, stopped, moved, or deleted using the Docker API. A container is by default relatively well isolated from other containers and the host machine. When a container is removed, any changes to its state that are not kept in persistent storage will also be removed.

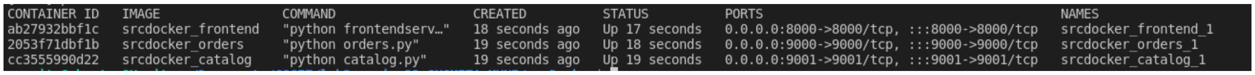
Containers are defined by their images and any configuration options they are given when they are created or started. To persist the results outside the container, we need to mount a directory on the host machine as a volume in the container. The volume instruction creates a mount point with the specified name that holds externally mounted volumes from native other containers or hosts. In order to persist the data in orderlog.csv, we have to provide the volume instruction in docker-compose, so whenever the container starts it persists the previous version of the orderlog.csv file. Similarly, we need to provide the volume instruction in order to persist the data in toyStoreData.json file.

We use ***docker run*** command to run an image inside of a container. For example, to run the catalog image, we simply run(-d parameter means the containers should be started in the background (detached mode)):

```
docker run --name catalog -d catalog
```

To view the list of containers that are running , run the below command:

docker ps



```
CONTAINER ID   IMAGE               COMMAND                  CREATED          STATUS          PORTS                                       NAMES
ab27932bbf1c   srcdocker_frontend  "python frontendserv…"   18 seconds ago   Up 17 seconds   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   srcdocker_frontend_1
2053f71dbf1b   srcdocker_orders    "python orders.py"       19 seconds ago   Up 18 seconds   0.0.0.0:9000->9000/tcp, :::9000->9000/tcp   srcdocker_orders_1
cc3555990d22   srcdocker_catalog   "python catalog.py"      19 seconds ago   Up 19 seconds   0.0.0.0:9001->9001/tcp, :::9001->9001/tcp   srcdocker_catalog_1
```

Fig 2: list of running containers

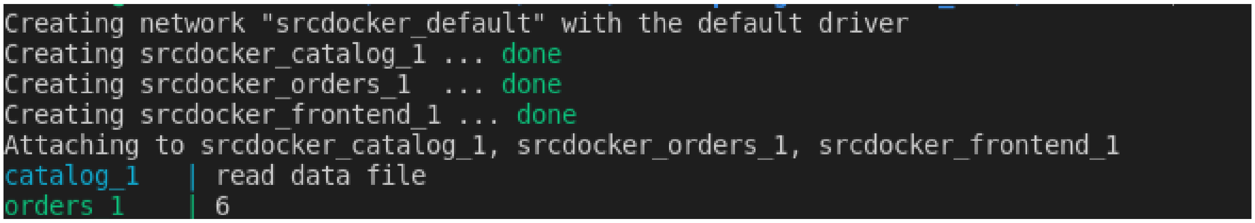
Managing Multiple Docker Containers using Docker Compose:

Using compose, we can define and run multi-container docker applications. By using a .yaml file to configure the applications services, we can start and stop all the services with a single command. As part of multi-tiered toy store, we can run the three containers(frontendservice, orders, catalog) by providing the following command:

docker-compose up

As we run the above command, any containers discovered from previous runs are copied to the newly created container,ensuring that any data you’ve created in volumes is preserved. This ensures that we don’t lose any data in orderlog.csv file when the orders container is stopped.

As the containers are up and running, each container will be able to communicate to the service present in the other container using the hostname. Frontend service is connecting to catalog and order services using the respective services hostnames.

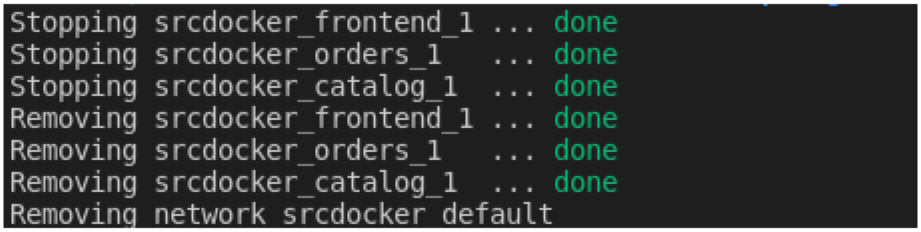


```
Creating network "srcdocker_default" with the default driver
Creating srcdocker_catalog_1 ... done
Creating srcdocker_orders_1  ... done
Creating srcdocker_frontend_1 ... done
Attaching to srcdocker_catalog_1, srcdocker_orders_1, srcdocker_frontend_1
catalog_1    | read data file
orders_1     | 6
```

Fig 3: Figure showing that the containers are up and running

To stop and remove these containers in the application, we can simply run the below command, instead of docker stop and docker run commands :

docker-compose down



```
Stopping srcdocker_frontend_1 ... done
Stopping srcdocker_orders_1   ... done
Stopping srcdocker_catalog_1  ... done
Removing srcdocker_frontend_1 ... done
Removing srcdocker_orders_1   ... done
Removing srcdocker_catalog_1  ... done
Removing network srcdocker_default
```

Fig 4: Figure showing that the containers have stopped and removed