Lecture 12: Volumes and Docker Compose

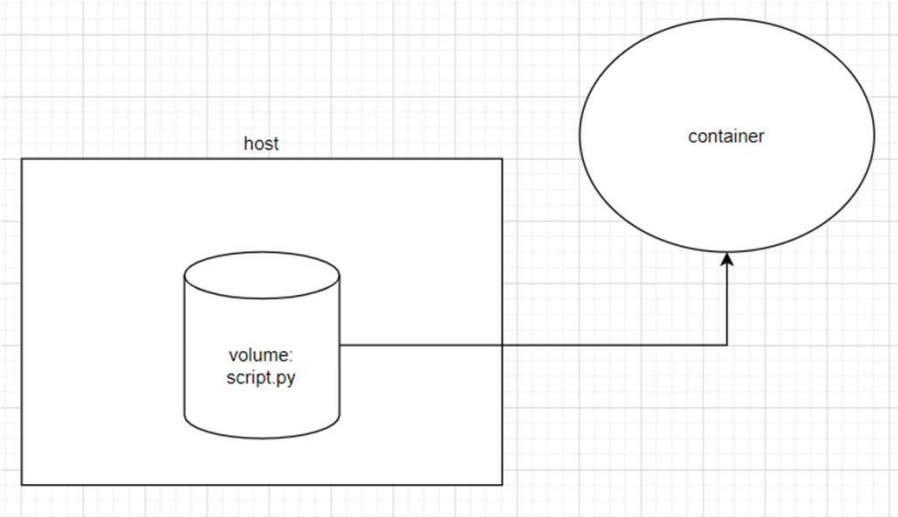
DES 422 Web and Business Application Development

Why do we need volume?

- From assignment, what if we want to change the random numbers generated from 10 numbers to 100 numbers? We have to
- delete the image
- change script.py
- rebuild the image
- let's see how long that takes

Volume

- Put our app in a volume
- File is modified directly on the host
- Just have to restart the container

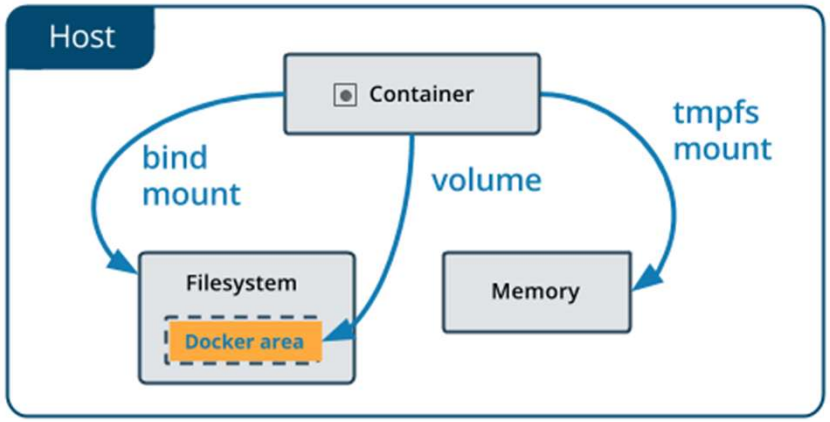


Container & persistent data

- containers are immutable and ephemeral (not permanent)
- docker follows “immutable infrastructure” principle. Only deploy/destroy containers, never change
- we must not keep persistent data inside container

Volume vs bind mount

- volume: special location in the FS
- bind mount: mount any folder on host to a container



Volume

- can create in Dockerfile using VOLUME command
- independent from any one container, can be shared among many containers
- volumes can be assigned names
- container can mount volume during run: *docker run -v /path/inside/container*

Bind mount

- map a path on host to a path on a container
- two containers can’t use the same mount path on host at the same time
- can’t be specified on Dockerfile
- docker run -v /path/on/host:/path/inside/container

Example – volume command

- docker volume ls
- docker volume create <my-vol>
- docker volume inspect <my-vol>
- docker volume rm <my-vol>

How container find volume

- *docker image inspect postgres*
- note the *volume* in json
- *docker run --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword -d postgres*
- note the *mount* in json
- compare the name to output of *docker volume ls*
- the volume name should be the same

```
},
"Mounts": [
    {
        "Type": "volume",
        "Name": "c37973069169f67aaab45c8061659a1cd3464aa06eeff660cb4ce850597d0294",
        "Source": "/var/lib/docker/volumes/c37973069169f67aaab45c8061659a1cd3464aa06eeff660cb4ce850597d0294/_data",
        "Destination": "/var/lib/postgresql/data",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

```
(base) PS C:\Users\acer\Desktop\tmp\docker-assignment2> docker volume ls
DRIVER              VOLUME NAME
local               3b5b8e678b6dcd48ae2cb3b73fb7f7dd90cb87e1cf6c88578d5e68798a579ab7
local               6ab724961e23ef623844dc40c95005a661a44995937102b89d27350ae66e225b
local               077d1aa9eeca1b5bd6e43e105c2613167d7d14334ce3ae0137fdfa619758bca3
local               86a2b36bec366a94a41fb169456956defd71033660b1dfad76e805aada303804
local               91b2bc1f8bce25f1cf40e5463863a340f451d17b2e1121bd69bdf4c0adf6d33d
local               487f1e16c85c39247b64f4c61d43eac33f62dc3c43c5e8a7a0a1729ce5a48ff2
local               7183b04ee2dc5bc0d02e674b8c258b1c388e14be048d147346a3f2220b2e1ba7
local               ab0106adc56a85f6b0aaab7e1c2ef9d21cc6c22fc9306c49080b7941aa801e7d
local               c37973069169f67aaab45c8061659a1cd3464aa06eeff660cb4ce850597d0294
local               da1914998cf29bd40705acf6a4fefa1cc8fd72bc939aa29f926675a7e32de904
local               e51895d10944ce59483eb4a6132d3643d2e4f602c631890a60f3eab53df7e3c4
local               f110baaffbbc4c13b2f88d582e76b636187cf919603610d37ea032249c29b511
local               my-vol
(base) PS C:\Users\acer\Desktop\tmp\docker-assignment2>
```

```
ENV PATH $PATH:/usr/lib/postgresql/$PG_MAJOR/bin
ENV PGDATA /var/lib/postgresql/data
# this 777 will be replaced by 700 at runtime (allow
RUN mkdir -p "$PGDATA" && chown -R postgres:postgres
VOLUME /var/lib/postgresql/data

COPY docker-entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/docker-entrypoint.sh / # bac
ENTRYPOINT ["docker-entrypoint.sh"]
```

Named Volume

- docker volume create my-vol
- docker run --name some-postgres -e POSTGRES_PASSWORD=mysecretpassword -v my-vol:/var/lib/postgresql/data -d postgres
- docker container inspect some-postgres


```
    },
    "Mounts": [
        {
            "Type": "volume",
            "Name": "my-vol",
            "Source": "/var/lib/docker/volumes/my-vol/_data",
            "Destination": "/var/lib/postgresql/data",
            "Driver": "local",
            "Mode": "z",
            "RW": true,
            "Propagation": ""
        }
    ],
    "Config": {
        "Hostname": "ce912c5305fe",
```

Volume is persistent

- docker container rm -f some-postgres
- docker run --name some-postgres2 -e POSTGRES_PASSWORD=mysecretpassword -v my-vol:/var/lib/postgresql/data -d postgres
- docker container inspect some-postgres2


```
    },
    "Mounts": [
        {
            "Type": "volume",
            "Name": "my-vol",
            "Source": "/var/lib/docker/volumes/my-vol/_data",
            "Destination": "/var/lib/postgresql/data",
            "Driver": "local",
            "Mode": "z",
            "RW": true,
            "Propagation": ""
        }
    ],
```

Path expansion

- Linux/Mac
  - $(pwd)
- Windows
  - %cd% CMD
  - ${pwd} powershell

Bind Mount

- we want to show this html page in nginx instead of the default nginx page
- we want to be able to edit this file on the host, and see the effect without re-building the image

```html
<html>
    <head>
    </head>
    <body>
        <h1>hello world</h1>
    </body>
</html>
```

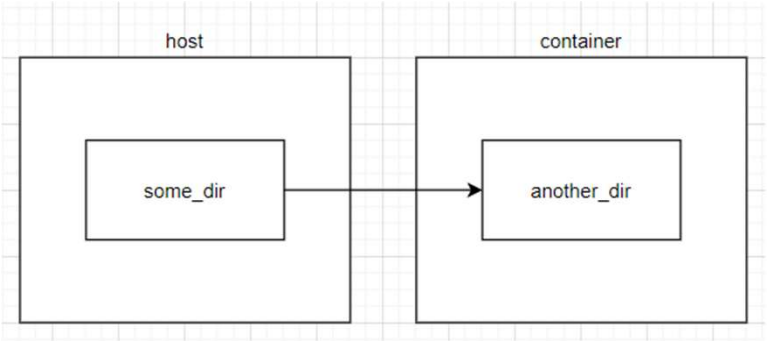
Bind Mount

- docker container run -d --name nginx -p 80:80 -v ${pwd}:/usr/share/nginx/html nginx
- modify index.html
- reload the page in the browser
- create a new file in the host
- go into the container and ls the directory /usr/share/nginx/html

```html
<html>
    <head>
    </head>
    <body>
        <h1>hello world</h1>
    </body>
</html>
```

Quiz

- What is the wrong with the following command?
- docker run -v /path/to/another_dir:/path/to/some_dir some-image



Exercise – Volume

- scenario: we want to upgrade a DB
- create a new named volume *my-data*
- start postgres:9.6.15 container with named volume *my-data* (search on Docker Hub to find the path in container we're supposed to mount this volume to)
- view the log (it should be long) and stop the container
- start postgres:9.6.16 with the same named volume
- check log to see it's the same DB (log should be much shorter than 9.6.15's because it doesn't have to prepare any directory)

Exercise – bind mount

- go to https://www.free-css.com/ and download a template you like
- extract the zip file
- start nginx container, mount the folder you extracted to /usr/share/nginx/html
- go to localhost to see the website
- change something in index.html (on host)
- refresh browser to see the effect

docker-compose

docker-compose.yaml

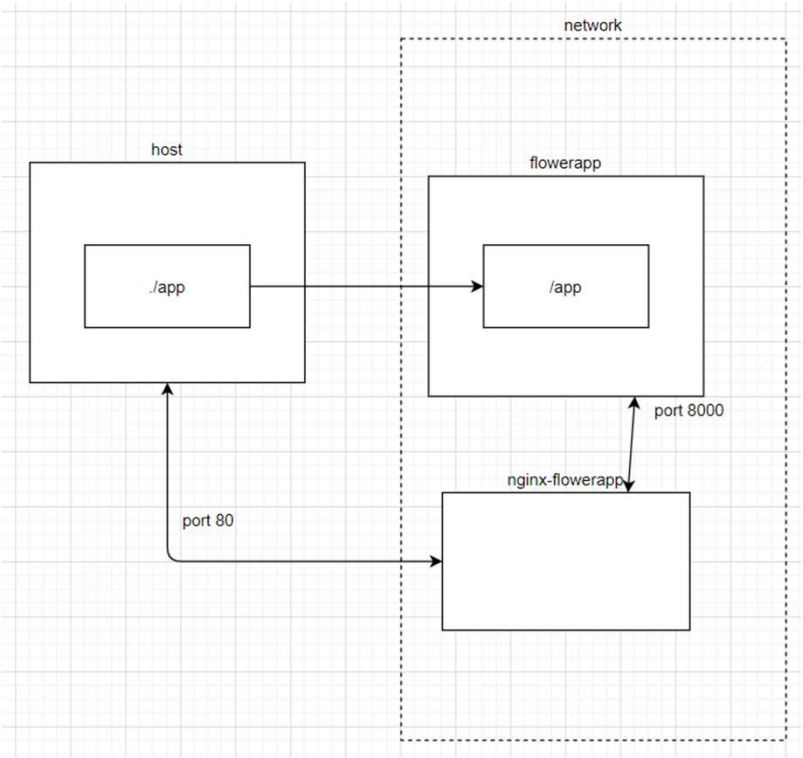
```yaml
version: '2'
services:
  flowers:
    container_name: flowersapp
    image: flowers
    build:
      context: ./app
      dockerfile: Dockerfile
    volumes:
      - ./app:/app
    expose:
      - "8000"
    command: gunicorn --bind 0.0.0.0:8000 main:app

  nginx:
    container_name: nginx-flowersapp
    restart: always
    build: ./nginx
    ports:
      - "80:80"
    depends_on:
      - flowers
```

version

- version of the syntax in compose file, not version of your app!
- latest version is 3
- not much different between V2 and V3, except Swarm

service

- container_name
- image
- build
- volumes
- ports
- command
- ports
- depends_on



docker-compose commands

- *docker-compose up*
- *docker-compose down*

Example – running docker-compose

docker-compose.yaml

```yaml
version: '3'

services:
  proxy:
    image: nginx
    ports:
      - '80:80'
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf:ro
  my_web:
    image: yeasy/simple-web
```

nginx.conf

```
server {

    listen 80;

    location / {

        proxy_pass         http://my_web;
        proxy_redirect     off;
        proxy_set_header   Host $host;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Host $server_name;

    }
}
```

HW3

- Modify the flower example
  1. Add a SQL database container to the app, using the mysql image
  2. Setup the database container in Docker compose, the four environment variables that have to be defined are:
     - `MYSQL_RANDOM_ROOT_PASSWORD`
     - `MYSQL_DATABASE`
     - `MYSQL_USER`
     - `MYSQL_PASSWORD`
  3. Make a new volume for the DB

HW3

- Modify app/main.py
  1. When the app starts, check if a table called results exists, if not, create it – use mysql-connector-python library to connect to the DB from Python
  2. Every time a query is received, after making the prediction, insert a new row into the results table

timestamp (primary key)	prediction_result
1681043479	sunflower

HW3

- Make a few requests using Postman
- Access the DB and query the results table
- Zip and submit the whole project folder when finished