Séance 3

mardi 22 novembre 2022 07:43

Exo pour comprendre le principe **Network** :

docker network create myNetwork : on crée un network nommé myNetwork
docker run --name c1 -it busybox sh : on crée un conteneur nommé c1 et on ouvre un shell dessus
docker network connect myNetwork c1 : on lie le conteneur et le network
Ip addr : dans le conteneur, affiche les adresses

```
11: eth1@if12: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc
 noqueue
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth1
       valid_lft forever preferred_lft forever
/ #
```

En vrai, ce n'est pas pratique du tout !
Autre méthode :
docker run --name c1 -it --network myNetwork busybox sh : la même mais en une ligne !
Par contre, on voit que l'adresse liée à l'hote (par defaut quand il n'y a pas de network), n'est pas créé du tout !

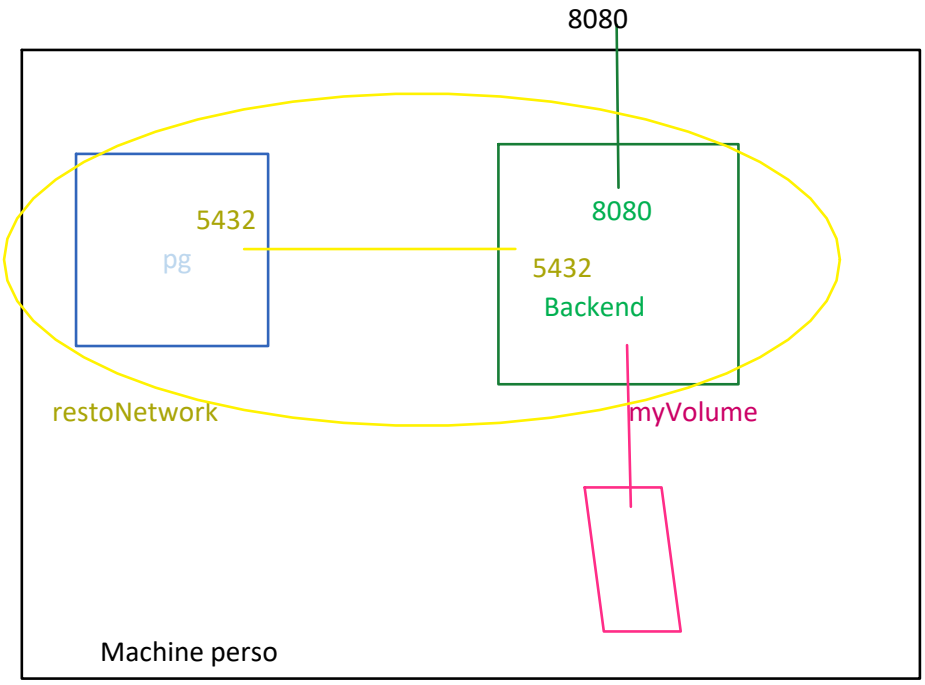
On peut créer deux conteneurs qui sont sur le même network. Chacun peut ping l'autre directement avec le nom du docker :

```
/ # ping c1
PING c1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.069 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.054 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.053 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.141 ms
^C
--- c1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.053/0.079/0.141 ms
/ #
```

NB : c1 et c2 ne sont pas accessibles depuis notre machine

Projet bibliothèque :

Actuellement, le port 5432 de notre pc est exposé.
Nous allons changer ça :
En faisant un lien directement entre la bdd et le back

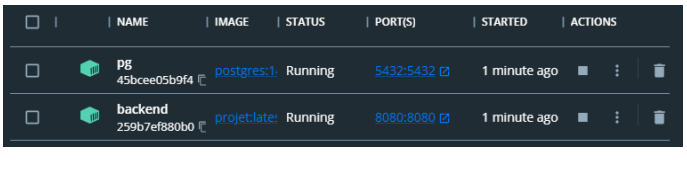


Nous allons donc créer un network entre la bdd et le back !

docker network create restoNetwork

docker run --name pg -d -p 5432:5432 --env POSTGRES_DB=myDatabase --env POSTGRES_USER=databaseUser --env POSTGRES_PASSWORD=databasePassword -v myVolume:/var/lib/postgresql/data:rw --network restoNetwork postgres:14

docker run --name backend -d -p 8080:8080 --env SPRING_DATASOURCE_URL="jdbc:postgresql://**pg**/myDatabase" --env SPRING_DATASOURCE_USERNAME=databaseUser --env SPRING_DATASOURCE_PASSWORD=databasePassword --network restoNetwork projet



Ce n'est plus notre ip mais le nom du conteneur bdd

On passe de la bibliothèque au resto pour plus d'uniformisation (prob avec les url)

On veut maintenant lier le front au reste !

Dockerfile du frontend :

```
FROM node:16 AS builder

WORKDIR /usr/src/app

COPY . .

RUN npm install && npm run build

FROM nginx

COPY nginx/ /etc/ngix/

COPY --from=builder /usr/src/app/dist /usr/share/nginx/html

COPY entrypoint.sh entrypoint.sh

EXPOSE 80

ENV $BACKEND_URL_ENV_VAR=localhost

ENTRYPOINT ["entrypoint.sh"]
```

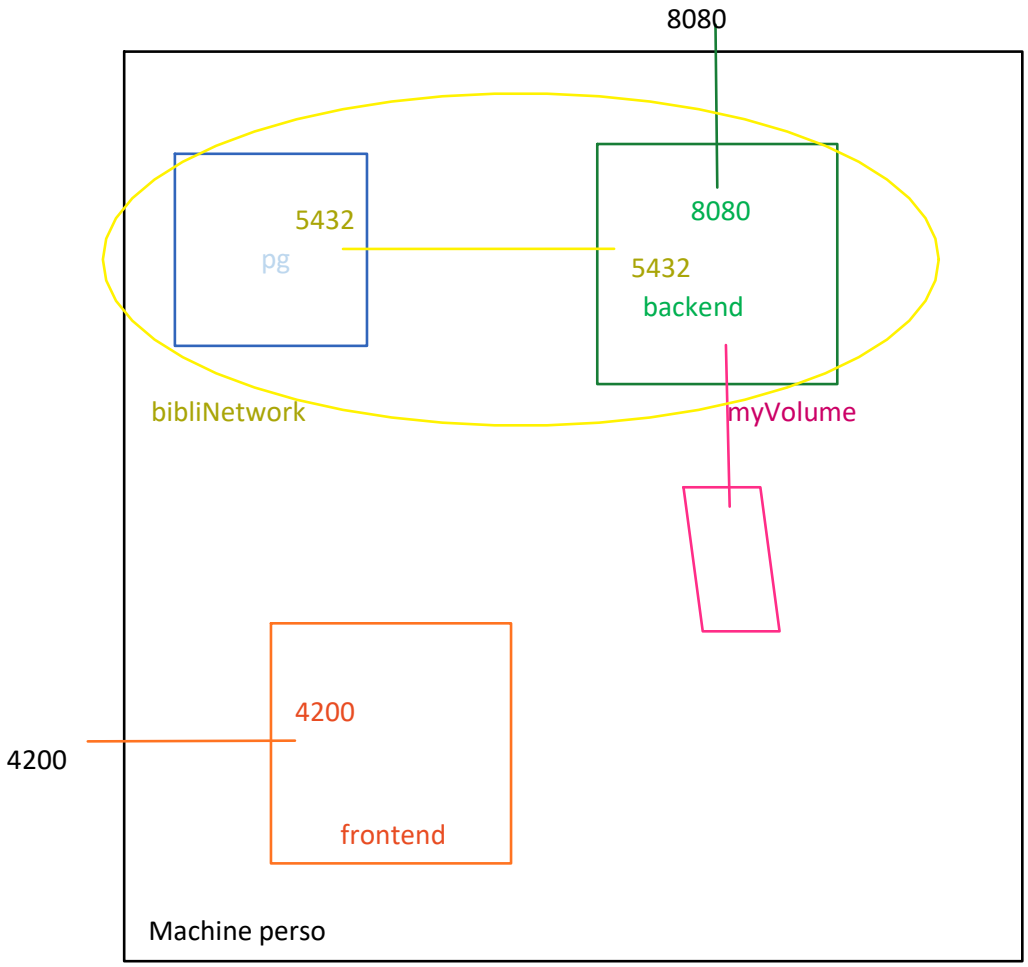
On veut pouvoir changer l'url d'acces facilement sans rentrer dans le code.
Pour cela, on utilise un placeholder dans le fichier environement.prod.ts :
backendUrl : '{{ BACKEND_URL }}'

En parallèle, on crée un fichier entrypoint.sh :

```
#!/bin/sh
sed-i's/{{BACKEND_URL}}/$BACKEND_URL_ENV_VAR/g'/usr/share/nginx/html/*
exec"$@"
```

Il sera appelé par le dockerfile (grace au ENTRYPOINT)

Une fois l'image build **docker build --tag projet_front .**, on peut la run en précisant les variables d'environnement
docker run --name frontend -d -p 4200:4200 --env BACKEND_URL_ENV_VAR="localhost:8080" projet_front



On crée maintenant un fichier yaml pour eviter d'avoir à toujours tapper les commandes de run des conteneurs, c'est un docker compose :
(exemple avec postgres :)

```
volumes:
	postgres:

networks:
	backend:

services:
	postgres:
		image: postgres:14
		container_name: pg
		volumes:
			-postgres:/var/lib/postgresql/data:rw
		networks:
			-backend
		environment:
			-POSTGRES_DB=myDatabase
			-POSTGRES_USER=databaseUser
			-POSTGRES_PASSWORD=databasePassword
```

Il se place à la racine du front et du back pour acceder aux deux.

Pour l'executer : **docker-compose up -d :**
Pour les stoper et les supprimer : **docker-compose down**