

# Séance 2

vendredi 18 novembre 2022 08:21

## Commandes intéressantes :

**docker exec** : taper des commandes dans le docker directement

**docker exec -it hopeful\_goodall bash** : ouvrir un terminal sur le conteneur

**psql -U databaseUser -W myDatabase** : on se connecte sur la base de donnée

**\q** : quitter la bdd

**exit** : quitter le terminal

**docker inspect hopeful\_goodall** : avoir toutes les infos sur le conteneur (json)

**docker run -p 5432:5432** : Mapping de ports : on peut lier un conteneur au localhost de notre machine

EXPOSE : ajoute des infos (visibles avec inspect)

## Projet bibliothèque :

FROM maven:3.8-openjdk-18

WORKDIR usr/src/app

COPY ..

CMD ["mvn", "spring-boot:run"]

Prob : outils orienté dev -> surcouche outillage non nécessaire pour démarrer l'appli + le mvn run compile, dl les dépendances et run le projet

On peut remplacer les action par un fat-jar (créable avec docker aussi) et le run avec java

Run le fatjar grâce à java  
(fait à chaque run)

FROM maven:3.8-openjdk-18 as builder  
WORKDIR usr/src/app  
COPY ..  
RUN mvn clean package -DskipTests=true

Création fat jar grâce à mvn  
(fait que au build)

FROM openjdk:18-jdk  
WORKDIR usr/src/app  
COPY --from=builder usr/src/app/target/fs-back-\*.jar project.jar  
CMD ["java", "-jar", "project.jar"]

Build : **docker build --tag projet**

Run : **docker run --rm projet**

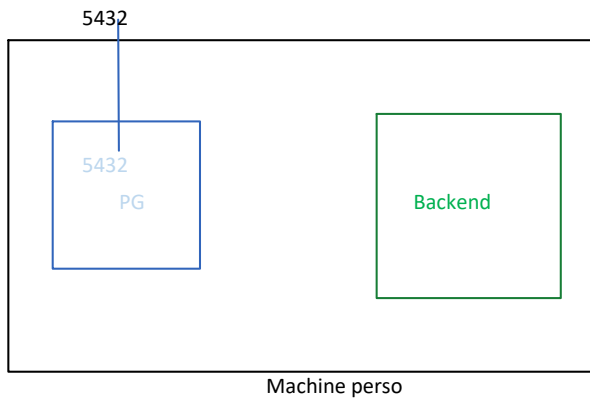
On veut maintenant lier notre base de donnée :

On va donc run un conteneur postgres avec toutes les bonnes variable d'environnement (comme le

password par exemple) : **docker run -d --env POSTGRES\_DB=myDatabase --env**

**POSTGRES\_USER=databaseUser --env POSTGRES\_PASSWORD=databasePassword postgres:14**

(retourne l'id du conteneur grace au -d)



On veut faire le lien entre le backend et le reste :

`SPRING_DATASOURCE_URL=jdbc:postgresql://host.docker.internal:5432/myDatabase`

`SPRING_DATASOURCE_USERNAME=databaseUser`

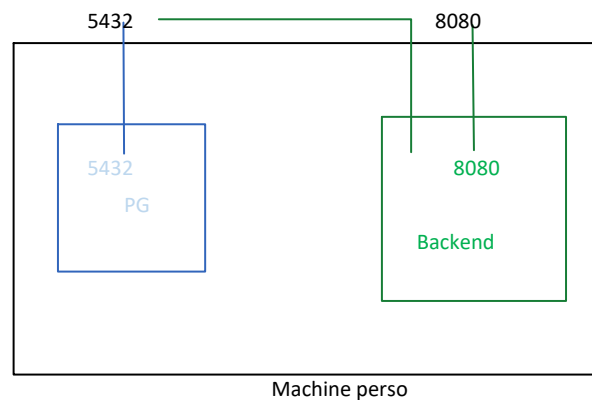
`SPRING_DATASOURCE_PASSWORD=databasePassword`

c'est exactement les infos dans ressources/application.properties mais point de vue boîte backend

Attention : j'ai eu un problème avec le host.docker.internal -> 172.17.0.1 (Gateway)

	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	<b>sad_wozniak</b> c59863027cff	postgres:14	Running	5432:5432	11 minutes ag	
<input type="checkbox"/>	<b>serene_maxwell</b> fe6fd09ac95a	projet:latest	Running	8080:8080	6 minutes agc	

localhost:8080/livres -> retourne tableau de livres (potentiellement vide)



On peut maintenant faire des requêtes :

POST

localhost:8080/livres

Params

Authorization

Headers (8)

Body

Pre-request Scr

none

form-data

x-www-form-urlencoded

raw

bin

```
1 {
2   "titre": "title",
3   "commentaires": []
4 }
```

localhost:8080/livres

JSON

Données brutes

En-têtes

Enregistrer Copier Tout réduire Tout développer

Filtrer le JSON

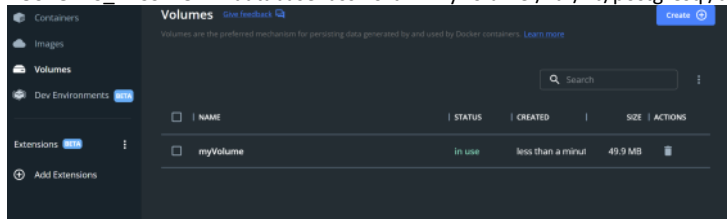
```
0: {
  id: 1
  titre: "string"
  commentaires: []
}
1: {
  id: 2
  titre: "test"
  commentaires: []
}
2: {
  id: 3
  titre: "string"
  commentaires: []
}
3: {
  id: 4
  titre: "title"
  commentaires: []
}
```

Si le docker de bdd se stop, il faut pouvoir récupérer nos données !

On va donc faire un docker de volume qui stockera les données de notre base :

On le crée directement à la création du docker postgres

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

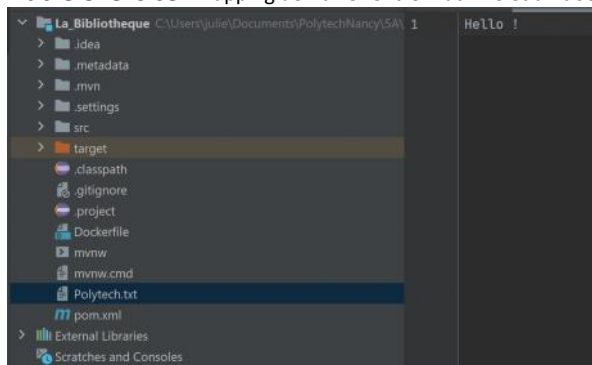


On démarre ensuite notre docker back

```
docker run -d -p 8080:8080 --env SPRING_DATASOURCE_URL="jdbc:postgresql://172.17.0.1:5432/myDatabase" --env SPRING_DATASOURCE_USERNAME=databaseUser --env SPRING_DATASOURCE_PASSWORD=databasePassword projet
```

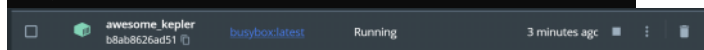
C'est tout bon ! Même en stoppant et supprimant le conteneur postgres, si on le redémarre, les données sont toujours là !

### Autre exercice : mapping de fichier entre machine et un docker :



```
docker run --rm -it -v $PWD/Polytech.txt:/tmp/Polytech.txt busybox sh
```

```
PS C:\Users\julie\Documents\PolytechNancy\5A\ProgrammationFullStack\Back\La_Bibliotheque> docker run --rm -it -v $PWD/Polytech.txt:/tmp/Polytech.txt busybox sh
/ # ls
bin dev etc home proc root sys tmp usr var
/ # cd tmp/
/tmp # ls
Polytech.txt
/tmp # |
```



En gros, on a un fichier partagé entre notre machine et un docker ! (et ce de façon dynamique !!)

PS : cela fonctionne aussi avec des répertoires

### Autre exercice : Docker de volume

```
PS C:\Users\julie\Documents\PolytechNancy\5A\ProgrammationFullStack\Back\La_Bibliotheque> docker volume create myVolume
myVolume
PS C:\Users\julie\Documents\PolytechNancy\5A\ProgrammationFullStack\Back\La_Bibliotheque> docker volume ls
DRIVER      VOLUME NAME
local       bd3eb07b9c458c7ae49032eb8d911f7f4aff32f32ff766826cdf12adba525a5b2
local       myVolume
PS C:\Users\julie\Documents\PolytechNancy\5A\ProgrammationFullStack\Back\La_Bibliotheque> docker inspect myVolume
[
  {
    "CreatedAt": "2022-11-18T10:48:52Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/myVolume/_data",
    "Name": "myVolume",
    "Options": {},
    "Scope": "local"
  }
]
```

(on peut en faire deux et voir qu'ils "communiquent" entre eux !)