# Workshop docker

## Prérequis :

* 1. Postman (qui permet d’envoyer des requêtes, ce qui va tester votre docker)
  2. Docker
  3. Docker-compose
  4. Portainer (utile pour le debug)

Postman : <https://www.google.fr/search?q=install+postman+fedora>

Docker : <https://docs.docker.com/engine/install/fedora/>

Docker compose : <https://docs.docker.com/compose/install/>

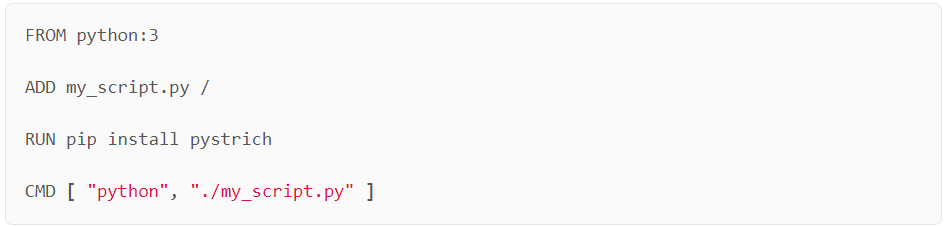
Portainer : <https://www.portainer.io/>

## Exercice 1 :

Ici il suffit simplement de compléter le dockerfile et le docker compose (le docker compose est très simple)

1. Docker étant « une machine virtuelle » très légère elle ne contient pas toutes les librairies du dump. Il faut donc les installer… (En quelle langage est l’API ?)
2. Maintenant que vous avez installé le nécessaire, pour lancer le programme dans la VM, il faut aussi s’occuper des dépendances … (lignes 8 et 9 de init.py)
3. Il faut aussi lancer le programme une fois la VM lancé.

Un exemple :



1. Il faut maintenant utiliser le docker-compose.yml. (Bien que dans le cas présent ce n’est pas nécessaire, mais vous en aurez besoin pour la suite) Ce dernier se divise en plusieurs parties :
   1. Version
   2. Services
   3. Réseau
   4. Build

Une fois que ces parties sont présente, il suffira de faire **docker-compose** *$nom*

Pour tester, il faudra lancer Postman et effectuer une requête GET sur l’adresse que vous avez donnée en fonction des ports que vous avez utilisés.

Ex : « localhost:79/ping »

Vous devriez obtenir le résultat suivant : *pong*

## Exercice 2 :

Le docker file entre l’exercice 1 et l’exercice 2 est presque identique, il peut nécessiter une variable d’environnement en plus. ([ici](https://fr.wikipedia.org/wiki/Variable_d%27environnement))

Il faudra compléter le docker-compose.yml de l’exerice 1 pour ajouter elastic-search. Il s’agit d’un gestionnaire de base de données très utilisé, en particulier en bigdata. ([ici](https://www.elastic.co/fr/elasticsearch))

Dans le docker-compose il y aura des informations supplémentaires à ajouter :

1. Volumes
2. Link

Si vous ne créez pas de link, les 2 VM ne pourront pas communiquer.

Si vous ne faites pas de volumes les informations seront supprimées quand les VM ne tourneront plus.

Conseils :

1. Elastic est lent à se lancer, l’api sera prête plus vite, lancez d’abord elastic en cas de problème.
2. Si vous avez des problèmes de connexion, utilisez portainer pour trouver l’origine du problème.
3. Si elastic-search crash au lancement exécutez **elastic.sh**
4. **Google**

Pour tester, il faudra lancer Postman et effectuer une requête GET sur l’adresse que vous avez donnée en fonction des ports que vous avez utilisés.

Ex : « localhost:79/test »

Vous devriez obtenir le résultat suivant :

*{*

*"acknowledged" : true,*

*"shards\_acknowledged" : true,*

*"index" : "$indexName"*

*}*

## Exercice 3 :

Le docker file entre l’exercice 1, 2 et 3 est presque identique, il peut nécessiter une variable d’environnement de plus.

Il faudra compléter le docker-compose.yml de l’exercice 2 pour ajouter une base de données redis. Il s’agit d’une base de données stockée dans la ram. Elle est très intéressante pour les applications de cache. ([ici](https://fr.wikipedia.org/wiki/M%C3%A9moire_cache) et [ici](https://redis.io/))

Dans le docker-compose il y aura des informations supplémentaires à ajouter :

1. Link

Conseils :

* Réfléchissez à qui utilise la base de données redis… Est-ce que c’est nécessaire de la link à l’ensemble ?

Pour tester, il faudra encore une fois lancer postman et effectuer une requête GET sur l’adresse que vous avez donnée en fonction des ports que vous avez utilisés.

Ex : « localhost:79/test\_elastic »

Vous devriez avoir le résultat suivant :

*{*

*"acknowledged" : true,*

*"shards\_acknowledged" : true,*

*"index" : "$indexName"*

*}*

Ex : « localhost:79/test\_redis »

Vous devriez avoir le résultat suivant : « ok »