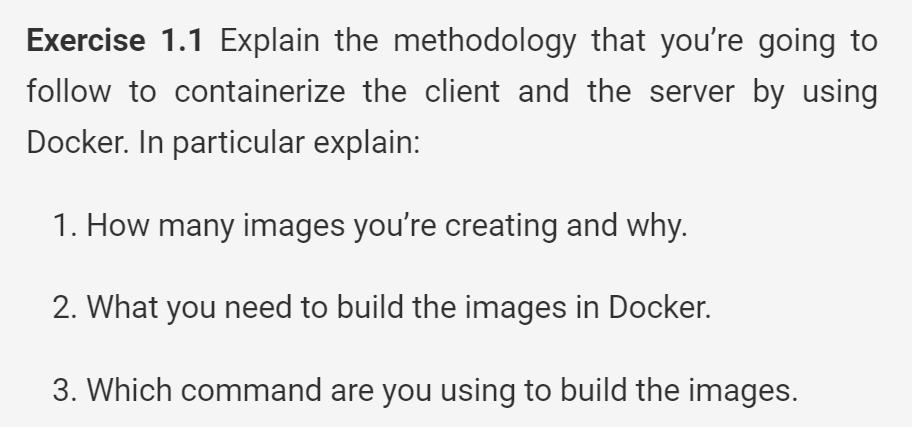
TP 1

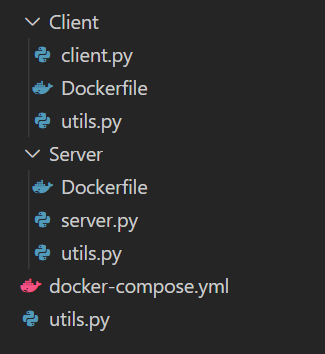


1. J’ai pu identifier 2 services différents (donc deux Docker images différentes) : client et server.

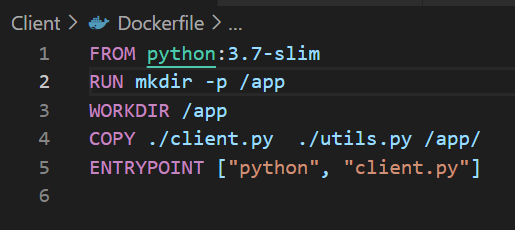
On crée donc 2 images.

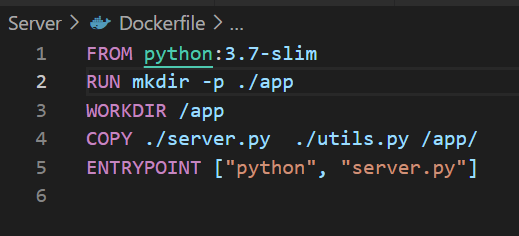
Pour le service client, on créera une instance (donc un container) à chaque fois qu’un client se connectera sur sa machine à un temps donné (le container sera donc différents s’il un même client se connecte 2h plus tard par exemple).

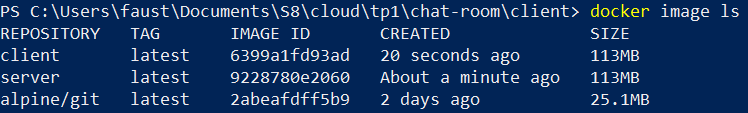
Pour le service server, on aura une instance.



1. On a besoin d’un Dockerfile par service, puis on rajoutera un Docker compose pour créer l’application multi-service sur ma machine. On prendra comme Dockerfile :



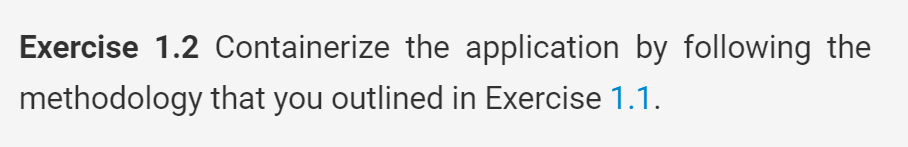




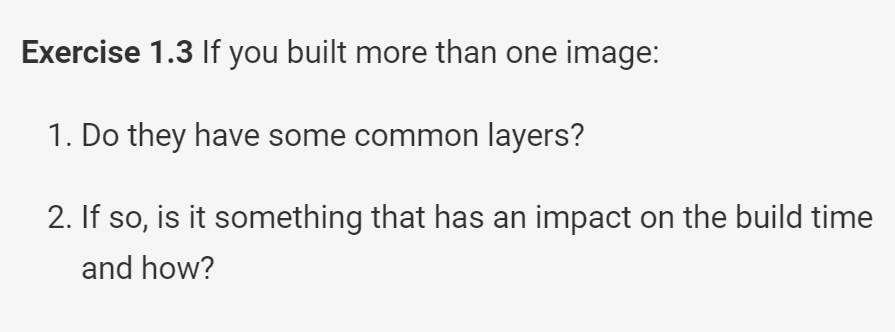
1. Les commandes que l’on lancera pour les créer sont :





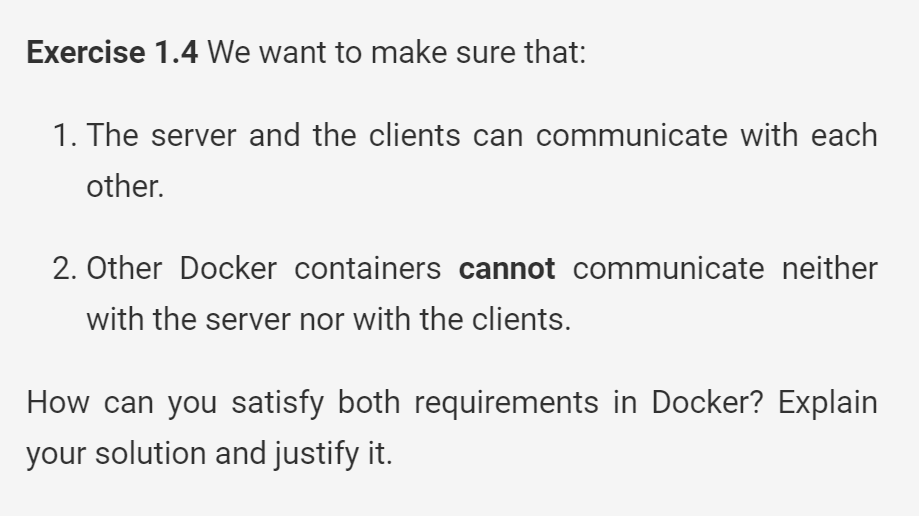


On run dans le powershell



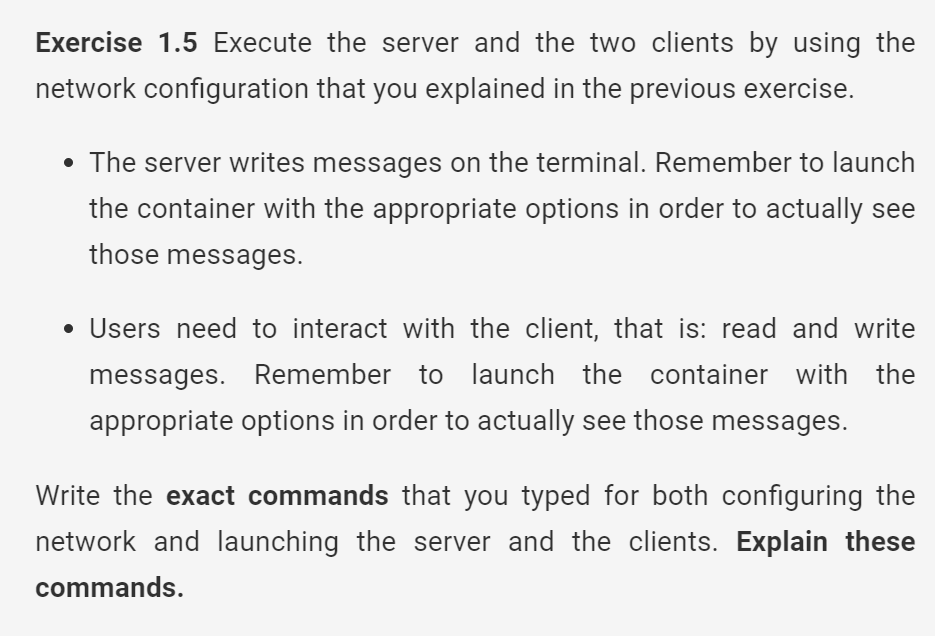
1. On regarde les lignes en commun sur les deux dockerfile : ce sont dont les 3 premières layers qui sont communes.
2. Lorsqu’on lance le 2e dockerfile, la machine a encore en cache les 3 premières layers du premier fichier, ce qui fait que le deuxième dockerfile est plus rapide à construire.

On aurait pu construire les deux dockerfile avec une ligne en plus (COPY utils, puis copy les autres file). Cela aurait l’avantage d’aller plus vite à la construction d’après ce que l’on a dit plus haut. Cependant, avec une ligne en plus on aurait aussi une layer en plus et donc des images plus lourdes, et on a dit dans le sujet que ce n’était pas ce que l’on voulait.



On met ces deux images et leurs containers dans un même network différent de build. Ils pourront donc communiquer tous les deux.

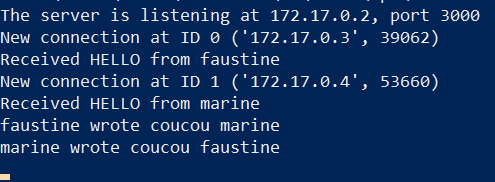
Si on n’ajoute aucun autre container dans ce network, alors personne d’autres ne pourra communiquer ave eux, ce qui correspond à cette deuxième exigence.



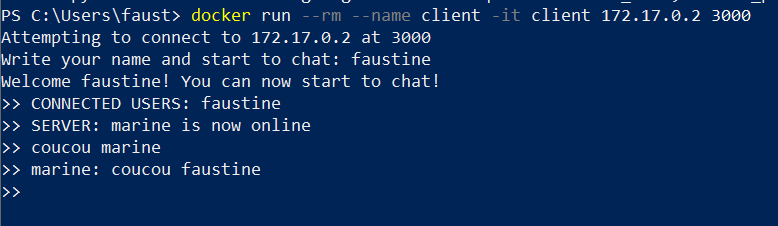


 dans le premier terminal

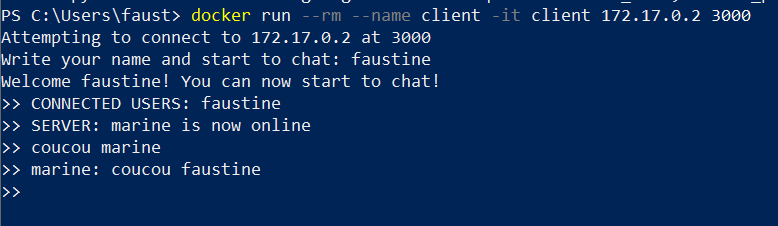
Le -it sert à avoir un terminal linux qui ne se ferme pas pour pouvoir monitorer ce qu’il se passe dans le chat genre :

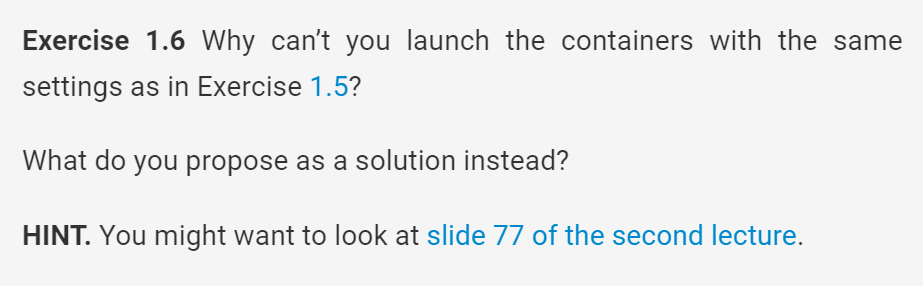


Avec dans un deuxième terminal :



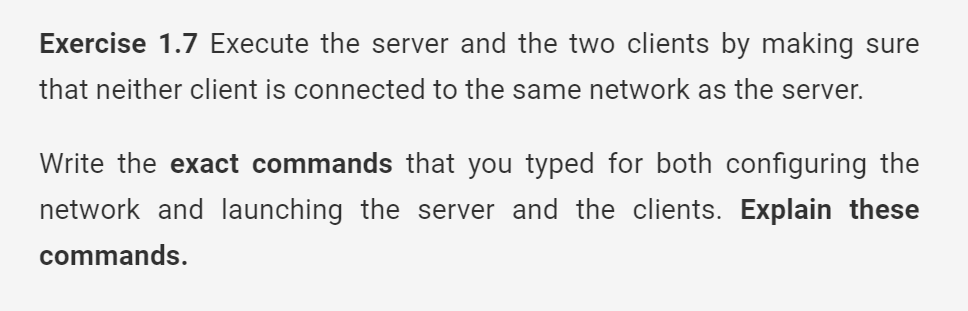
Avec dans un troisième :

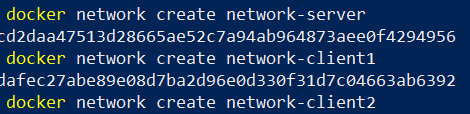




Parce que en 1.5 ils étaient dans le même network, donc ça ne fonctionne pas ici.

On doit créer deux networks : l’un pour le server, et l’autre pour les clients (on peut même créer un network pour chaque client, ce qui serait plus représentatif de la réalité puisqu’il n’y a aucune raison que les clients se trouvent sur le même réseau en réalité).

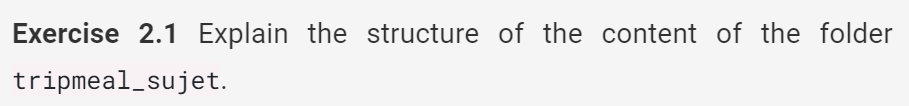






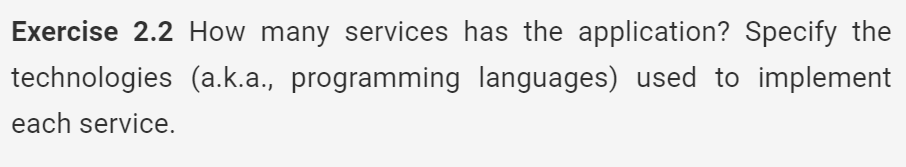






**web**. This is the **frontend** (the part the user interacts with) of the application. It consists of HTML and JavaScript code. Il y a en plus une partie en scripts python qui permettent de lier à la base de données, ainsi qu’une liste de prérequis pour lancer l’application.

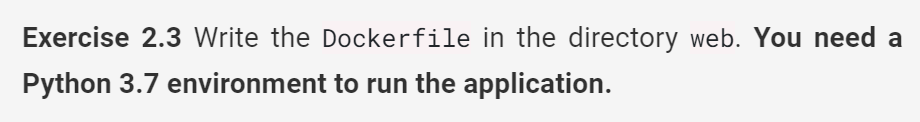
**db**. This is the **backend** (the part hidden to the user). It is a [SQL](https://www.postgresql.org/) (relational) database.



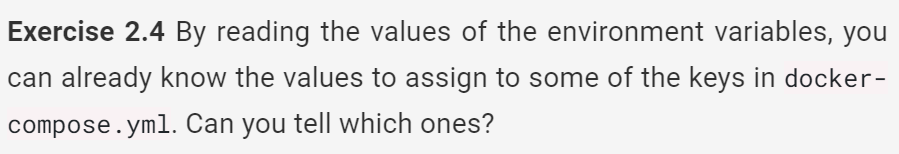
2 services (web et db)

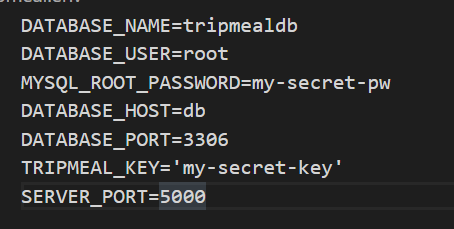
Db : sql

Web : python, html, js, css



Okok c’est bon, cf vscode

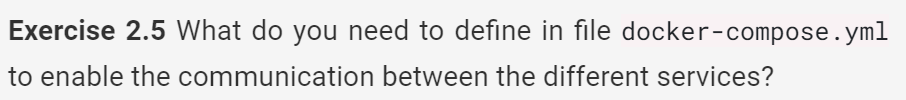




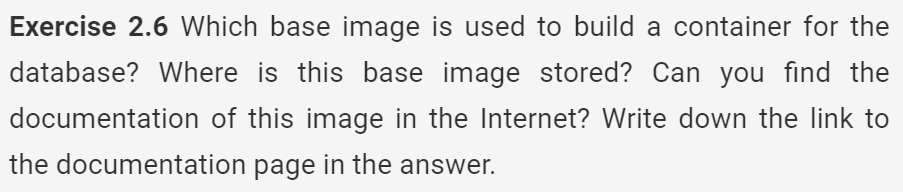
On peut donc avoir accès au port pour database

On pourra aussi les rajouter au docker compose sous l’intitulé su service db

Quid des volumes ?

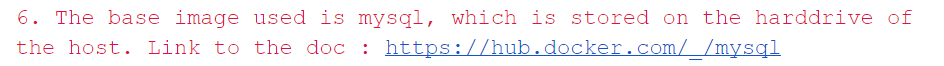


On doit les mettre sur le même network (par exemple backend) pour qu’ils puissent communiquer, et le spécifier en plus sur le docker-compose.



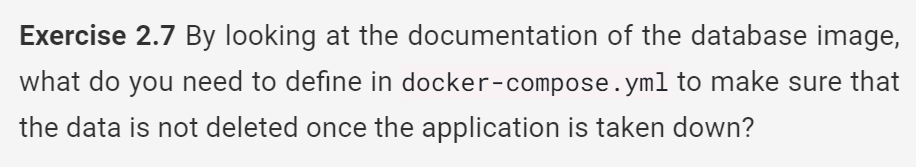
Base image : dans le « from » du dockerfile, donc ici : mysql.

C’est une image de base donc c’est dans le docker hub

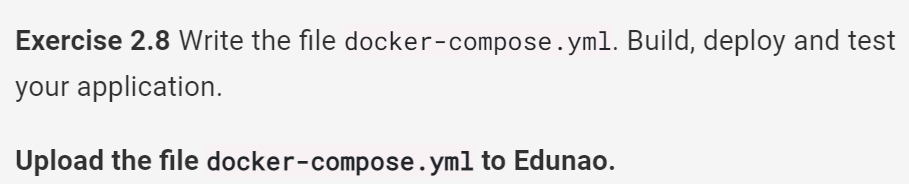


registry.hub.docker.com

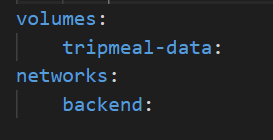
<https://hub.docker.com/_/mysql/>

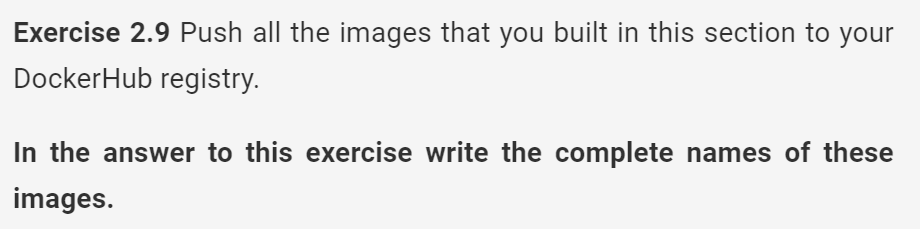


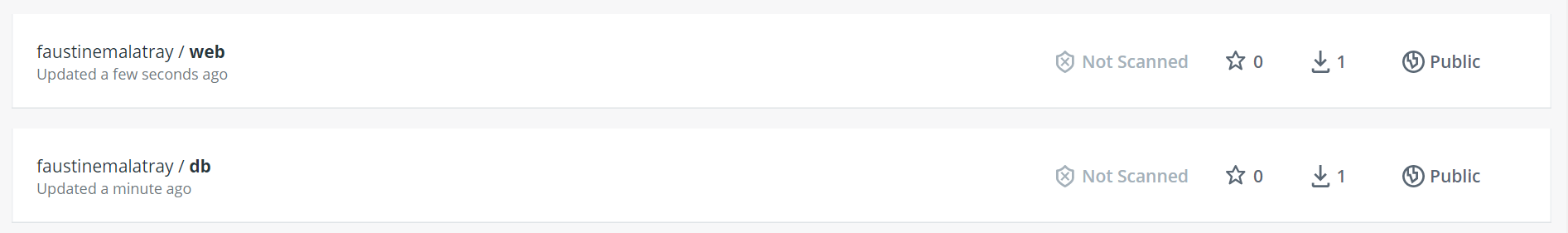
A volume ? pour stocker les données permanentes

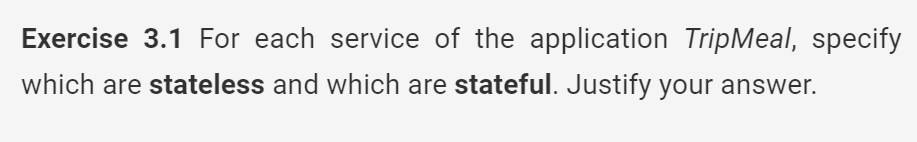










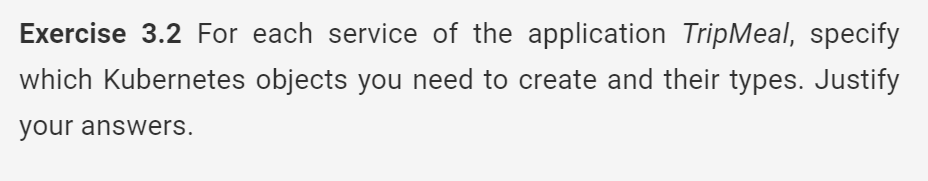


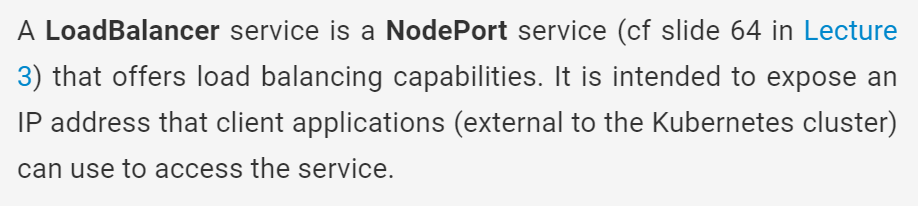
Stateful service: one that manages permanent data and is bound to that data.

Stateless service: one that is not bound to permanent data

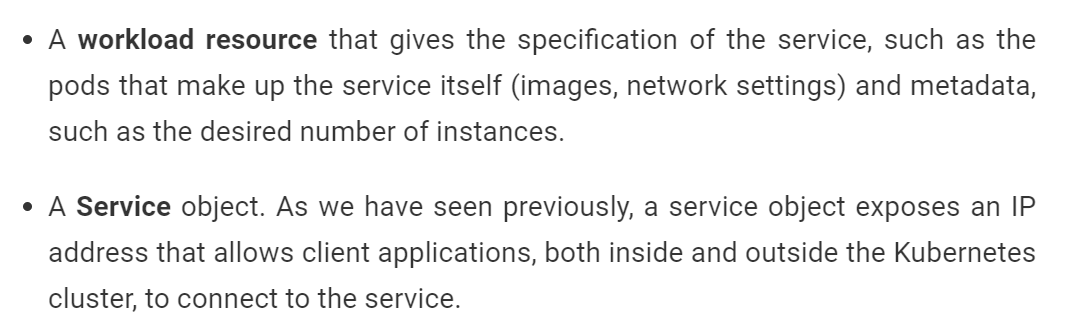
The service web is stateless because it does not have to be tightly bound to the data. In other words, the service web uses the data that is stored in the backend database, but it can be scheduled on any node of the cluster independently of that data.

The service db is stateful because it directly manages permanent data.

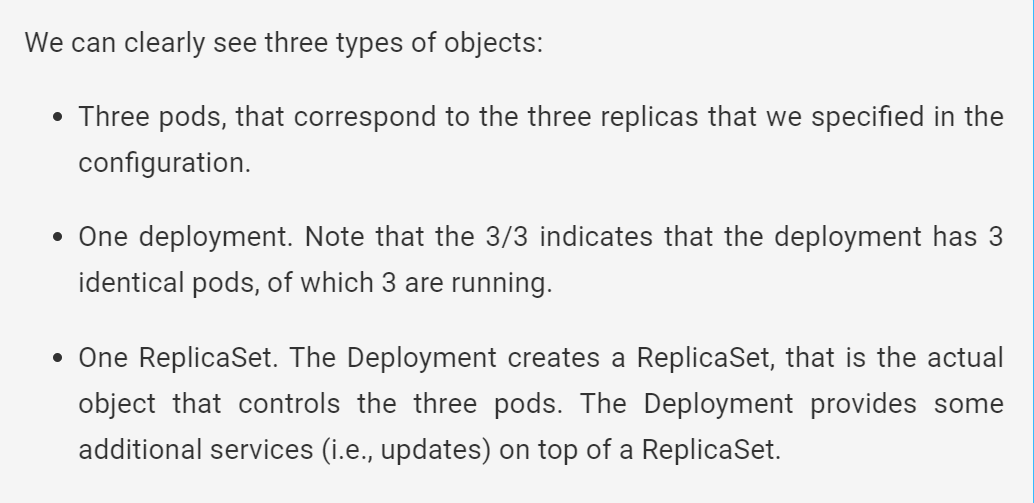




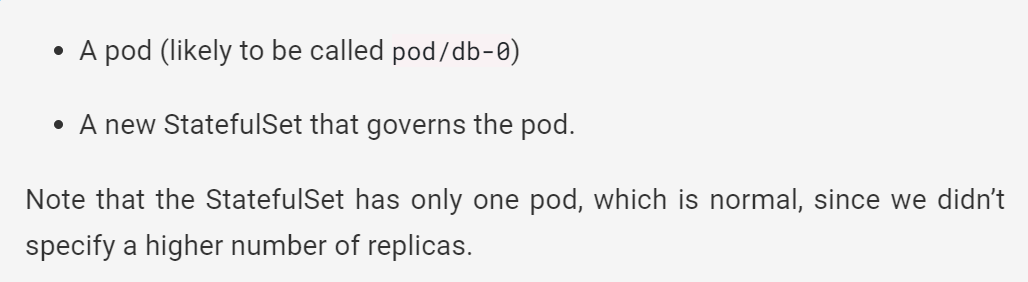
2 types of objects for each service:



Pour web, c’est un déploiement qu’il faut créer, qui ensuite créer les objets suivants :



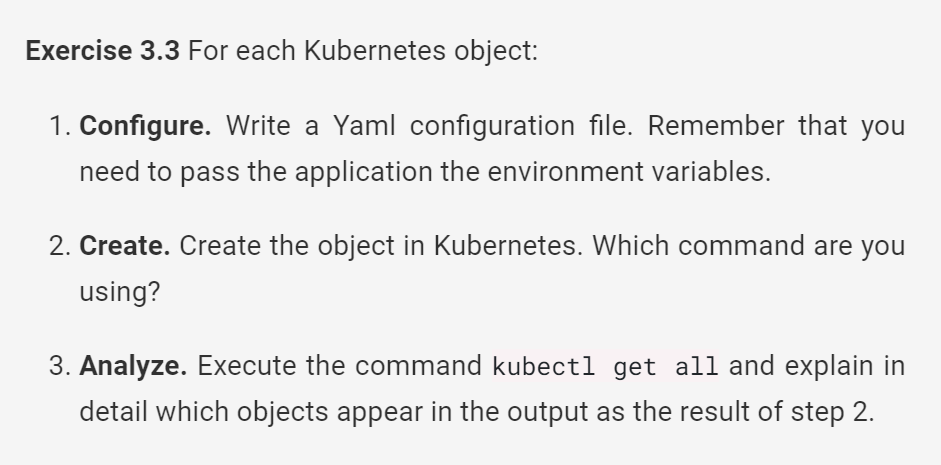
Pour database, c’est un déploiement qu’il faut créer, qui lui-même créer les objets suivants :

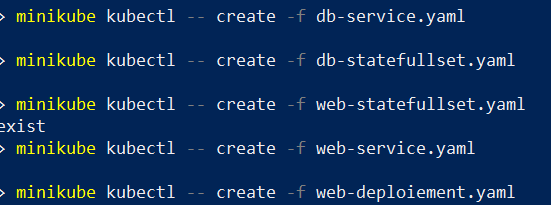


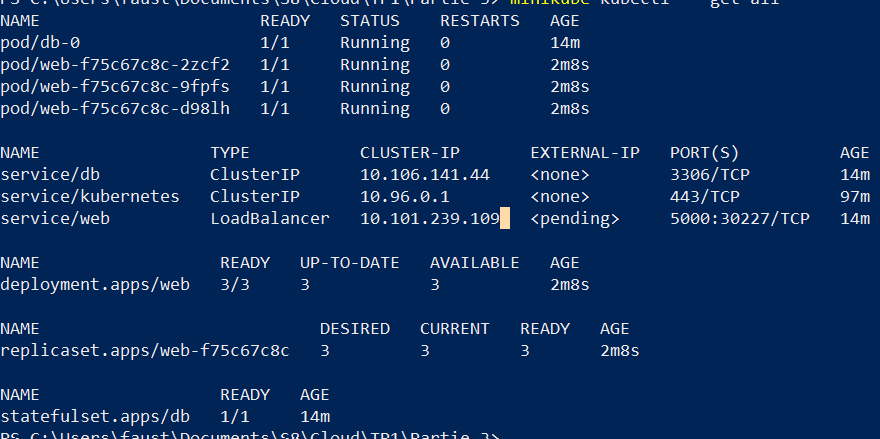
C’est un déploiement qui ne créer pas de déploiement lui-même (plus ou moins inclus dans statefulset), puisque ce serait lourd et inutile de répliquer la db à chaque fois en entier dans chaque pod. Ici, la solution est plutôt d’avoir une réplique commune pour tous les pods.

En résumé, en fait on créer 4 objets :

* Le déploiement de web
* Le « déploiement » de db
* Le lien web/client
* Le lien db/web

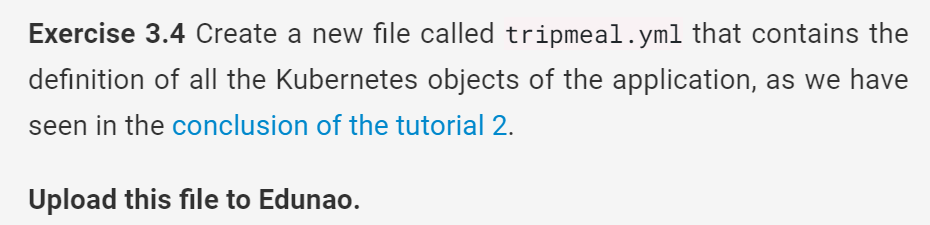




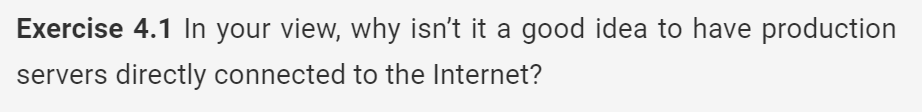


On voit bien que pour le service web, on a les 3 pods, avec un déploiement et un replicaset

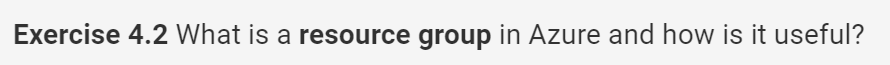
On voit également bien que pour le service db, on a un pod, un service, et un statefulsest



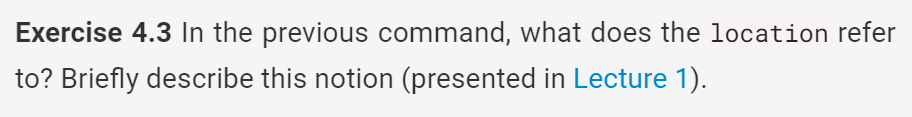
Okok



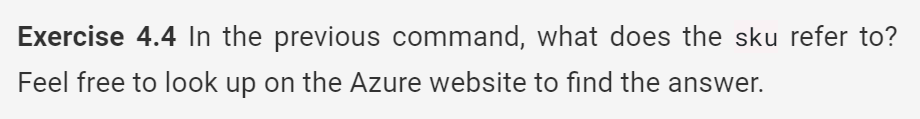
On va dire ok



On va dire ok aussi



Plus o moins ok aussi



/

