Tâches automatiques avec les jobs Cron

- 9.0 - Écrire un job cron qui tourne en tant que `votreutilisateur` et toutes les minutes met le résultat de la commande `date` dans `/home/votreutilisateur/date`
- 9.1 - Écrire un job cron qui tourne en tant que `root` et toutes les minutes remplace le contenu de `/etc/resolv.conf` par `nameserver 89.234.141.66`
- 9.2 - Écrire un petit script nommé dans `autoupdate` que l'on mettra dans `/etc/cron.weekly` qui lancera `apt update` et `apt dist-upgrade -y` une fois par semaine. (NB: dans la vraie vie on ne ferait pas ça ! La problématique de mises à jour automatiques et plus compliquée que ça. Il existe des paquets comme `unattended-upgrades` pour gérer ça)
- 9.3 - Sur votre serveur, écrire un job cron qui tourne en tant que `votreutilisateur` et toutes les heures (mais le temps de tester, disons toutes les minutes!), créer un backup des données de votre nextcloud vers son home à l'aide de `tar` . (Les données de Nextcloud sont stockées dans `/var/www/nextcloud/data`)

Initiation à Docker pour le déploiement d'applications

- 10.1 - Installer Docker en suivant les instructions sur le site officiel : https://docs.docker.com/engine/install/debian/

Cela devrait se résumer à faire (en root)

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archi
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/n

apt update
apt install docker-ce
```

Vous devriez maintenant constater qu'un service `docker` tourne sur la machine. Vous devriez aussi voir qu'il existe une nouvelle interface réseau `docker0` dans la sortie de `ip a` .

- 10.2 - Installons aussi Docker Compose, qui est une surcouche à Docker, et permet de manipuler plus simplement des ensembles de conteneurs. Pour avoir une version récente de ce logiciel, nous utilisons `pip` , le gestionnaire de paquet du language python (qui est le langage utilisé pour développer Docker Compose).

```
apt install python3-pip
pip3 install docker-compose
```

- 10.3 - Récupérons le code de l'application en utilisant `git` :

```
apt install git
cd /var/www/
git clone https://github.com/e-lie/microblog
```

- 10.4 - Rentrons dans le dossier `/var/www/microblog/` . Notons l'existence d'un fichier `Dockerfile` . Dans ce fichier, identifier les différentes étapes de la construction de l'image:
    - image de départ
    - création d'un utilisateur système
    - installation des dépendances de l'app avec pip
    - copie des fichiers de l'app (code source et autre) dans l'image
    - déclaration du port sur lequel fonctionnera l'application
    - déclaration de l'"entrypoint", c'est à dire le programme ou script qui se lance au démarrage du conteneur
- 10.5 - Récupérer le fichier `docker-compose.yml` auprès du formateur. Ce fichier est destiné à être mis dans `/var/www/microblog/` . Dans ce fichier, constater que :
    - la première section corresponds à l'application à proprement parler (`microblog`). L'image correspondante sera construite à partir du Dockerfile précédemment étudié. Ce conteneur exposera l'application sur le port 5000 dans la machine. C'est le port que nous utiliserons plus tard pour faire l'interface avec nginx.
    - la deuxième section corresponds à la base de donnée de l'application (`db`). L'image correspondate (mariadb) existe déjà et sera récupérée sur le docker hub. Par contre, on précise via des variables quels sont les identifiants de DB à créér.
    - les identiants de la base de données de `db` sont repris dans la variable `DATABASE_URL` du conteneur `microblog` .
    - les données effectives de la base de données sont stockées dans un volume `db-data` , c'est-à-dire de données persistentes, stockées **en dehors** du conteneur.
- 10.6 - Déclencher le build du container microblog avec `docker-compose build microblog` (notons ici qu'il n'y a pas besoin de dire à `docker-compose` de se baser sur le fichier `docker-compose.yml` , car le nom est standardisé et il utilise implicitement le fichier portant ce nom dans le répertoire courant).
- 10.7 - Démarrer le conteneur de base de donnée avec `docker-compose up -d db` .
    - Regarder avec `docker-compose logs db` les logs du démarrage du conteneur pour vérifier que ça s'est bien passé.
    - Constater aussi que `docker-compose ps` (commande qui permet de lister l'état des conteneurs, similaire à `docker ps`) liste bien le conteneur dans l'état 'Up'.
    - Dans la machine hôte, lancer `ps -ef --forest` pour voir le processus `mysqld` en cours d'execution, fils d'un processus nommé `/usr/bin/containerd(...)` .
    - Ouvrir un shell bash dans le conteneur à l'aide de `docker-compose exec db bash` . Dans ce shell qui tourne dans le conteneur, lancer `ps -ef --forest` pour voir le processus `mysqld` en cours d'execution. (Vous devriez voir qu'il n'y a vraiment pas beaucoup de processus dans le conteneur !)
    - (Quitter le shell du conteneur)
- 10.8 - Démarrer le conteneur de l'application avec `docker-compose up -d microblog` .
    - Regarder avec `docker-compose logs microblog` les logs du démarrage du conteneur pour vérifier que ça s'est bien passé.

    - Constater aussi que `docker-compose ps` , qui permet de lister l'état des conteneurs, qu'il est bien dans l'état 'Up'. Vous devriez également voir que le port 5000 est exposé dans la machine (voir même publiquement !?)
    - Dans la machine hôte, lancer `ps -ef --forest` pour voir le processus `python gunicorn` en cours d'execution, fils d'un processus nommé `/usr/bin/containerd(...)` .
    - Ouvrir un shell bash dans le conteneur à l'aide de `docker-compose exec db sh` (NB: `sh` ! car `bash` n'est pas disponible dans ce conteneur...). Dans ce shell qui tourne dans le conteneur, lancer `ps -ef` (pas d'option `--forest` ! snif) pour voir le processus `python gunicorn` en cours d'execution. (Vous devriez voir qu'il n'y a vraiment pas beaucoup de processus dans le conteneur !)
    - (Quitter le shell du conteneur)

- 10.9 - Ajouter le bout de configuration nginx pour exposer l'application via le serveur web. Notez que l'application n'est pas conçue pour être installée sous un sous-chemin comme `/microblog` (à la différence de Nextcloud) ... nous sommes obligé de l'exposer *à la racine du domaine* (`location /`), assurez-vous qu'il n'y a pas d'autre bout de conf qui conflicte ! :

```
location / {
    proxy_pass http://127.0.0.1:5000;
}
```

- 10.10 - Après avoir rechargé nginx, vous devriez avoir accès à l'application sur `http://votre.domaine.tld/` . Il vous faudra commencer par créer un compte via le lien de création de compte vers le bas de la page ("New User? Click to Register!"). (Inutile de mettre une vraie addresse mail !). S'amuser ensuite à poster quelques messages dans l'interface.

- 10.11 - Faites `docker-compose down` pour stopper et supprimer les conteneurs. Constater avec `docker-compose ps` qu'il n'y a effectivement plus de conteneur. Faire `docker-compose up -d` pour recréer les conteneurs. Tentez de vous re-logger sur l'interface et ainsi de retrouver vos données. Savez-vous expliquer pourquoi vos données n'ont pas été supprimées ?

- 10.12 - Refaire la même manipulation, mais cette fois avec l'option `-v` de `docker-compose down` qui **supprime les volumes** (dans notre cas, la base de donnée). Après avoir recréé les conteneurs, les données sont-elles toujours là ?