Virtualisation légère – TP n° 1

Les bases de Docker

Pierre-Olivier *nemunaire* Mercier

Mercredi 7 septembre 2022

Abstract

Durant ce premier TP, nous allons apprendre à utiliser Docker, puis nous apprendrons à déployer un groupe de conteneurs !

Sommaire

1 Orchestrer un groupe de conteneurs

Maintenant que nous savons démarrer individuellement des conteneurs et les lier entre-eux, nous allons voir une première manière d'automatiser cela.

Plutôt que de lancer les commandes `docker` comme nous l'avons fait jusque-là : soit directement dans un terminal, soit via un script, nous allons décrire l'état que nous souhaitons atteindre : quelles images lancer, quels volumes créer, quels réseaux, etc. Cette description peut s'utiliser pour lancer un conteneur seul, mais elle prend tout son sens lorsqu'il faut démarrer tout un groupe de conteneurs qui fonctionnent de concert, parfois avec des dépendances (un serveur applicatif peut nécessiter d'avoir une base de données prête pour démarrer).

On parle d'orchestration, car nous allons utiliser Docker comme un chef d'orchestre : il va ordonner les créations des différents objets (volumes, réseaux, conteneurs, ...) afin d'arriver au résultat attendu, puis il va faire en sorte de maintenir ce résultat selon les événements qui pourront survenir.

Notre fil rouge dans cette partie sera la réalisation d'un système de monitoring, tel que nous pourrions le déployer chez un fournisseur de cloud.

Le résultat attendu d'ici la fin de l'exercice, est un groupe de conteneurs indépendants les uns des autres, réutilisables en fonction de besoins génériques et pouvant facilement être mis à l'échelle.

Nous collecterons les données d'utilisation de votre machine avec Telegraf. Ces données seront envoyées vers InfluxDB, puis elles seront affichées sous forme de graphique grâce à Chronograf.

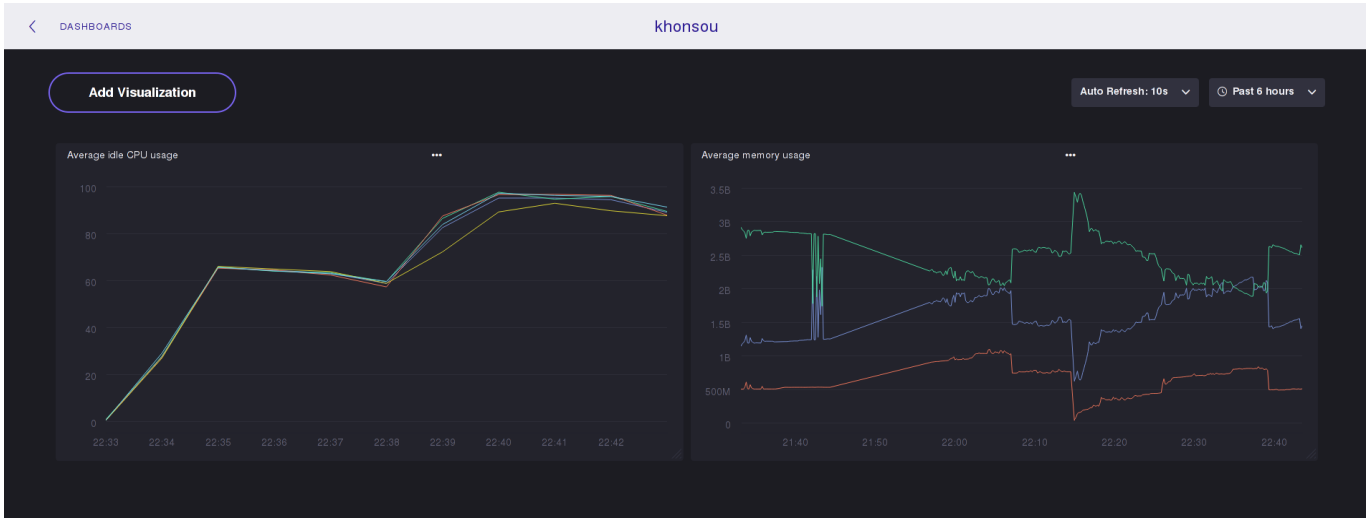

Figure 1: Dashboard de l'utilisation CPU et mémoire sur Chronograf

L'installation que nous allons réaliser est celle d'une plate-forme TICK. Il s'agit d'un mécanisme de séries temporelles (*Time Series*) moderne, que l'on peut utiliser pour stocker toute sorte de données liées à un indice temporel.

La pile logicielle TICK propose de collecter des métriques, en les enregistrant dans une base de données adaptées et permet ensuite de les ressortir sous forme de graphiques ou de les utiliser pour faire des alertes intelligentes.

1.1 Mise en place

Jusqu'ici, nous avons utilisé l'environnement Docker principal, qui inclut le client, le daemon et toute sa machinerie. Mais le projet Docker propose de nombreuses extensions, souvent directement trouvées dans les usages de la communauté, et parfois même appropriées par Docker.

1.1.1 `docker-compose`

Dans cette partie, nous allons avoir besoin du plugin `docker-compose`.

L'équipe en charge du projet met à disposition un exécutable que nous pouvons téléchargeant depuis https://github.com/docker/compose/releases.

Ajoutez l'exécutable dans le dossier des plugins : `$HOME/.docker/cli-plugins` (sans oublier de `chmod +x` !).

> Autrefois, `docker-compose` était un script tiers que l'on utilisait indépendamment de Docker. Le projet, historiquement écrit en Python, a été entièrement réécrit récemment afin qu'il s'intégre mieux dans l'écosystème.
> Vous trouverez encore de nombreux articles vous incitant à utiliser `docker-compose`. Dans la plupart des cas, vous pouvez simplement remplacer par des appels à `docker compose`.
> Il y a même un outil qui a spécialement été conçu pour migrer les lignes de commandes : https://github.com/docker/compose-switch

1.1.1.1 Vérification du fonctionnement

Comme avec Docker, nous pouvons vérifier le bon fonctionnement de `docker-compose` en exécutant la commande :

```
42sh$ docker compose version
Docker Compose version v2.10.2
```

Si vous obtenez une réponse similaire, c'est que vous êtes prêt à continuer ! Alors n'attendons pas, partons à l'aventure !

1.1.2 Play With Docker

Tout comme pour la partie précédente, si vous avez des difficultés pour réaliser les exercices sur votre machine, vous pouvez utiliser le projet Play With Docker qui vous donnera accès à un bac à sable avec lequel vous pourrez réaliser tous les exercices.

1.2 Lier des conteneurs

Avant de voir des méthodes plus automatiques pour déployer toute notre pile logicielle TICK, nous allons commencer par mettre en place et lier les conteneurs manuellement, de la même manière que nous avons pu le faire avec MySQL. Cela nous permettra de voir les subtilités de chaque image, ce qui nous fera gagner du temps pour ensuite en faire la description.

1.2.1 Conteneur central : la base de données

Le premier conteneur qui doit être lancé est la base de données orientée séries temporelles : InfluxDB. En effet, tous les autres conteneurs ont besoin de cette base de données pour fonctionner correctement : il serait impossible à *Chronograf* d'afficher les données sans base de données, tout comme *Telegraf* ne pourrait écrire les métriques dans une base de données à l'arrêt.

Afin d'interagir avec les données, InfluxDB expose une API REST sur le port 8086. Pour éviter d'avoir plus de configuration à réaliser, nous allons tâcher d'utiliser ce même port pour tester localement :

```
docker container run -p 8086:8086 -d --name mytsdb influxdb:1.8
```

Remarquez que nous n'utilisons pas la version 2 d'InfluxDB. Sa mise en place est plus contraignantes pour faire de simples tests. Si vous souhaitez tout de même utiliser la dernière version de la stack TICK, vous pouvez consulter le `README` du conteneur sur le Docker Hub : https://hub.docker.com/_/influxdb

Comme il s'agit d'une API REST, nous pouvons vérifier le bon fonctionnement de notre base de données en appelant :

```
42sh$ curl -f http://localhost:8086/ping
42sh$ echo $?
0
```

Notez que comme nous avons lancé le conteneur en mode détaché (option `-d`), nous ne voyons pas les logs qui sont écrits par le daemon. Pour les voir, il faut utiliser la commande `docker container logs` :

```
docker container logs mytsdb
```

Si votre influxdb répond, vous pouvez vous y connecter en utilisant directement le client officiel (le binaire s'appelle `influx`) :

```
42sh$ docker container run --rm -it --link mytsdb:influxdb influxdb:1.8 \
    influx -host influxdb
Connected to http://influxdb:8086 version 1.8.10
InfluxDB shell version: 1.8.10
> show databases
name: databases
name
----------------
_internal
```

Si vous aussi vous voyez la table `_internal`, bravo ! vous pouvez passer à la suite.

Mais quelle était cette commande magique ? Oui, prenons quelques minutes pour l'analyser…

L'option `--link` permet de lier deux conteneurs. Ici nous souhaitons lancer un nouveau conteneur pour notre client, en ayant un accès réseau au conteneur `mytsdb` que nous avons créé juste avant. L'option `--link` prend en argument le nom d'un conteneur en cours d'exécution (ici `mytsdb`, nom que l'on a attribué à notre conteneur exécutant la base de données influxdb via l'option `--name`), après les `:`, on précise le nom d'hôte que l'on souhaite attribuer à cette liaison, au sein de notre nouveau conteneur.

`influx -host influxdb` que nous avons placé après le nom de l'image, correspond à la première commande que l'on va exécuter dans notre conteneur, à la place du daemon `influxdb`. Ici nous voulons exécuter le client, il s'appelle `influx`, auquel nous ajoutons le nom de l'hôte dans lequel nous souhaitons nous connecter : grâce à l'option `--link`, le nom d'hôte à utiliser est `influxdb`.

L'option `--link` peut être particulièrement intéressante lorsque l'on souhaite sauvegarder une base, car en plus de définir un nom d'hôte, cette option fait hériter le nouveau conteneur des variables d'environnement du conteneur auquel elle est liée : on a donc accès aux `$MYSQL_USER`, `$MYSQL_PASSWORD`, ...

On aurait aussi pu créer un réseau entre nos deux conteneurs (via `docker network`), ou bien encore, on aurait pu exécuter notre client `influx` directement dans le conteneur de sa base de données :

```
42sh$ docker container exec mytsdb influx -execute "show databases"
name: databases
name
----------------
_internal
```

Dans ce dernier exemple, nous n'avons pas besoin de préciser l'hôte, car `influx` va tenter `localhost` par défaut, et puisque nous lançons la commande dans notre conteneur `mytsdb`, il s'agit bien du conteneur où s'exécute localement `influxdb`.

Ajoutez à la ligne de commande de lancement du conteneur les bon(s) volume(s) qui permettront de ne pas perdre les données d'influxDB si nous devions redémarrer le conteneur. Aidez-vous pour cela de la documentation du conteneur.

1.2.2 Collecter les données locales

Tentons maintenant de remplir notre base de données avec les métriques du système. Pour cela, on commence par télécharger *Telegraf* :

```
V=1.23.4
P=telegraf-${V}_linux_$(uname -m)
curl https://dl.influxdata.com/telegraf/releases/${P}.tar.gz | \
tar xzv -C /tmp
```

Puis, lançons *Telegraf* :

```
cd /tmp/telegraf
./usr/bin/telegraf --config etc/telegraf/telegraf.conf
```

La configuration par défaut va collecter les données de la machine locale et les envoyer sur le serveur situé à http://localhost:8086. Ici, cela fonctionne parce que l'on a fait en sorte de rediriger le port de notre conteneur sur notre machine locale (option `-p`).

Et observons ensuite :

```
42sh$ docker container run --rm -it --link mytsdb:zelda influxdb:1.8 \
    influx -host zelda
```

```
InfluxDB shell version: 1.8.10
> show databases
name: databases
name
---------------
_internal
telegraf

> use telegraf
Using database telegraf

> show measurements
name: measurements
name
------------------
cpu
disk
diskio
kernel
mem
processes
swap
system
```

La nouvelle base a donc bien été créée et tant que nous laissons *Telegraf* lancé, celui-ci va régulièrement envoyer des métriques de cette machine.

1.2.3 Afficher les données collectées

À vous de jouer pour lancer le conteneur *Chronograf*.

InfluxDB v2 Chronograf n'existe plus en tant que projet indépendant dans la version 2, si vous êtes parti sur cette version, vous retrouverez les tableaux de bord directement dans l'interface d'InfluxDB, sur le port 8086.

L'interface de *Chronograf* est disponible sur le port 8888.

Consultez la documentation du conteneur si besoin.

La page d'accueil est vide au démarrage, pour savoir si vous avez réussi, rendez-vous sous l'onglet *Hosts*, le nom de votre machine devrait y apparaître. En cliquant dessus, vous obtiendrez des graphiques similaires à ceux ci-après :

Figure 2: Résultat obtenu

1.3 Composition de conteneurs

1.3.1 Automatiser le lancement de conteneurs

Au lieu de faire un script pour construire et lancer tous vos conteneurs, nous allons définir à la racine de notre projet un fichier `docker-compose.yml` qui contiendra les paramètres d'exécution.

```
version: "3.9"
services:
  influxdb:
    ...
  chronograf:
    build: grafana/
    image: nginx
    ports:
      - "3000:3000"
    volumes:
      - ./:/tmp/toto
    links:
      - influxdb
```

Ce fichier est un condensé des options que nous passons habituellement au `docker container run`.

1.3.1.1 `version` Notons toutefois la présence d'une ligne `version` ; il ne s'agit pas de la version de vos conteneurs, mais de la version du format de fichier `docker-compose` qui sera utilisé. Sans indication de version, la version originale sera utilisée, ne vous permettant pas d'utiliser les dernières fonctionnalités de Docker.

1.3.1.2 `services` Cette section énumère la liste des services (ou conteneurs) qui seront gérés par `docker compose`.

Ils peuvent dépendre d'une image à construire localement, dans ce cas ils auront un fils `build`. Ou ils peuvent utiliser une image déjà existante, dans ce cas ils auront un fils `image`.

Les autres fils sont les paramètres classiques que l'on va passer à `docker run`.

1.3.1.3 `volumes`

Cette section est le pendant de la commande `docker volume`.

On déclare les volumes simplement en leur donnant un nom et un driver comme suit :

```
volumes:
  mysql-data:
    driver: local
```

Pour les utiliser avec un conteneur, on référence le nom ainsi que l'emplacement à partager :

```
[...]
  mysql:
    [...]
    volumes:
      - mysql-data:/var/lib/mysql
```

1.3.1.4 `network`

Cette section est le pendant de la commande `docker network`.

Par défaut, Docker relie tous les conteneurs sur un bridge et fait du NAT pour que les conteneurs puissent accéder à l'Internet. Mais ce n'est pas le seul mode possible !

De la même manière que pour les `volumes`, cette section déclare les réseaux qui pourront être utilisés par les `services`. On pourrait donc avoir :

```
networks:
  knotdns-slave-net:
    driver: bridge
```

1.3.1.4.1 Driver `host`

Le driver `host` réutilise la pile réseau de la machine hôte. Le conteneur pourra donc directement accéder au réseau, sans NAT et sans redirection de port. Les ports alloués par le conteneur ne devront pas entrer en conflit avec les ports ouverts par la machine hôte.

1.3.1.4.2 Driver `null`

Avec le driver `null`, la pile réseau est recréée et aucune interface (autre que l'interface de loopback) n'est présente. Le conteneur ne peut donc pas accéder à Internet, ni aux autres conteneurs, …

Lorsque l'on exécute un conteneur qui n'a pas besoin d'accéder au réseau, c'est le driver à utiliser. Par exemple pour un conteneur dont le but est de vérifier un backup de base de données.

1.3.1.4.3 Driver `bridge`

Le driver `bridge` crée un nouveau bridge qui sera partagé entre tous les conteneurs qui la référencent.

Avec cette configuration, les conteneurs ont accès à une résolution DNS des noms de conteneurs qui partagent leur bridge. Ainsi, sans avoir à utiliser la fonctionnalité de `link` au moment du `run`, il est possible de se retrouver lié, même après que l'on ait démarré. La résolution se fera dynamiquement.

1.3.1.5 Utiliser le `docker-compose.yml`

Consultez la documentation[1] pour une liste exhaustive des options que nous pouvons utiliser.

Une fois que notre `docker-compose.yml` est prêt, nous pouvons lancer la commande suivante et admirer le résultat :

```
docker compose up
```

Encore une fois, testez la bonne connexion entre `chronograf` (accessible sur http://localhost:8888) et `influxdb`.

[1] La documentation des `docker-compose.yml` : https://docs.docker.com/compose/compose-file/

2 Projet et rendu

2.1 Projet

Réalisez le `docker-compose.yml` permettant de lancer toute notre stack de monitoring, d'un simple :

```
42sh$ docker compose up
```

Vous intégrerez les trois images (`influxdb`, `chronograf`[2] et `telegraf`), mettrez en place les *volumes* et *networks* nécessaires au bon fonctionnement de la stack.

Le résultat final attendu doit permettre d'afficher dans `chronograf` l'hôte auto-monitoré par la stack, sans plus de configuration. Vous aurez pour cela besoin de placer des fichiers de configuration à côté de votre `docker-compose.yml`, afin de pouvoir inclure ces configurations dans les conteneurs, sans avoir besoin de reconstruire ces conteneurs.

2.2 Arborescence attendue

Tous les fichiers identifiés comme étant à rendre sont à placer dans un dépôt Git privé, que vous partagerez avec votre professeur.

Voici une arborescence type (vous pourriez avoir des fichiers supplémentaires) :

```
./
./docker-compose.yml
./...                  # Pour les fichiers de configuration
```

Votre rendu sera pris en compte en faisant un tag **signé par votre clef PGP**. Consultez les détails du rendu (nom du tag, ...) sur la page dédiée au projet sur la plateforme de rendu.

[2] N'ajoutez pas chronograf dans votre `docker-compose.yml` si vous avez opté pour la version 2 d'InfluxDB.