<!--

Conteneurs Docker

Modularisez et maîtrisez vos applications

<!--

Docker Compose

<!--

Docker Compose

- Nous avons pu constater que lancer plusieurs conteneurs liés avec leur mapping réseau et les volumes liés implique des commandes assez lourdes. Cela devient ingérable si l'on a beaucoup d'applications microservice avec des réseaux et des volumes spécifiques.
- Pour faciliter tout cela et dans l'optique d'Infrastructure as Code, Docker introduit un outil nommé docker-compose qui permet de décrire de applications multiconteneurs grâce à des fichiers YAML.

<!--

A quoi ça ressemble, YAML ?

```yaml
- marché:
    lieu: Marché de la Défense
    jour: jeudi
    horaire:
      unité: "heure"
      min: 12
      max: 20
    fruits:
      - nom: pomme
        couleur: "verte"
        pesticide: avec

      - nom: poires
```

<!--

Syntaxe

- Alignement ! (2 espaces !!)
- ALIGNEMENT !! (comme en python)
- ALIGNEMENT !!! (le défaut du YAML, pas de correcteur syntaxique automatique, c'est bête mais vous y perdrez forcément quelques heures !
- des listes (tirets)
- des paires clé: valeur
- Un peu comme du JSON, avec cette grosse différence que le JSON se fiche de l'alignement et met des accolades et des points-virgules

```yaml
version: '2.2'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.7.1
    container_name: elasticsearch
    environment:
      - cluster.name=docker-cluster
    volumes:
      - esdata1:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - esnet
  elasticsearch2:
    ...

volumes:
  esdata1:
    driver: local
  ...
```

<!--

Un (autre) exemple de fichier Docker Compose

```
version: 3

services:

  postgres:
    image: postgres:10
    environment:
      POSTGRES_USER: rails_user
      POSTGRES_PASSWORD: rails_password
      POSTGRES_DB: rails_db

  redis:
```

<!--

Le workflow de Docker Compose

Les commandes suivantes sont couramment utilisées lorsque vous travaillez avec Compose. La plupart se passent d'explications et ont des équivalents Docker directs, mais il vaut la peine d'en être conscient·e :

- `up` démarre tous les conteneurs définis dans le fichier compose et agrège la sortie des logs. Normalement, vous voudrez utiliser l'argument -d pour exécuter Compose en arrière-plan.
- `build` reconstruit toutes les images créées à partir de Dockerfiles. La commande up ne construira pas une image à moins qu'elle n'existe pas, donc utilisez cette commande à chaque fois que vous avez besoin de mettre à jour une image (quand vous avez édité un Dockerfile).

<!--

- `ps` fournit des informations sur le statut des conteneurs gérés par Compose.
- `run` fait tourner un conteneur pour exécuter une commande unique. Cela aura aussi pour effet de faire tourner tout conteneur lié, à moins que l'argument --no-deps ne soit donné.
- De façon générale la sortie des logs est colorée et agrégée pour les conteneurs gérés par Compose.
- `stop` arrête les conteneurs sans les enlever.
- `rm` enlève les contenants à l'arrêt. N'oubliez pas d'utiliser l'argument `-v` pour supprimer tous les volumes gérés par Docker.

<!--

Visualisation des applications microservice complexes

- Certaines applications microservice peuvent avoir potentiellement des dizaines de petits conteneurs spécialisés. Le service devient alors difficile à lire dans le compose file.
- Il est possible de visualiser l'architecture d'un fichier Docker Compose en utilisant docker-compose-viz (https://github.com/pmsipilot/docker-compose-viz]
- Cet outil peut être utilisé dans un cadre d'intégration continue pour produire automatiquement la documentation pour une image en fonction du code.