| | |
|---|---|
| esprit Se former autrement HONORIS UNITED UNIVERSITIES | Année Universitaire : 2023-2024<br><br>**Applications Web Distribuées**<br><br>**UP_WEB** |

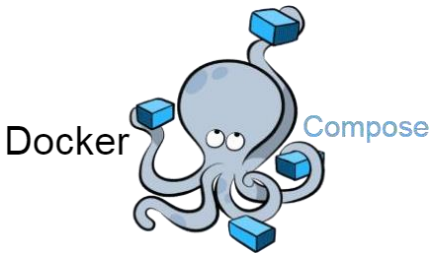
# Application multi-containers

## Mise en situation

- Dans le cas d'une application qui répond à une montée en charge et dans le cas de plusieurs instances de chaque Microservice, il devient nécessaire d'automatiser les tâches de paramétrage, de builds des images et de run des différentes instances, de définir les dépendances entre les microservices…

- Docker-Compose est un outil permettant de définir et d'exécuter des applications Docker multi-conteneurs. Avec Docker-Compose, on utilise un fichier YAML pour configurer les servicesde notre application. Ensuite, avec une seule commande, nous créons et démarrons tous les services à partir de cette configuration.

- Docker Compose permet en effet d'orchestrer les conteneurs, et ainsi de simplifier les déploiements sur de multiples environnements. Docker Compose est un outil écrit en Python qui permet de décrire, dans un fichier YAML, plusieurs conteneurs comme un ensemble de services.




## Objectifs

- Ecrire un script docker-compose
- Lancer une application multi-containers

# Etapes

## Etape 1: Ecrire le script de configuration

Dans un fichier nommé **docker-compose.yml**, ajouter le script d'automatisation. Ce script permet de faire le build, le run … et d'autres configurations relatives aux conteneurs Docker. Ci-après un exemple de script qui fait le build des 3 microservices déjà développés dans l'atelier précédent (candidate-service, job-service et discovery-service). Il fait aussi le run de 2 instances du MS candidate, 1 du MS jobs et 1 du service de découverte eureka.

```
version: "3.9"
services:

  discovery:
    build: ../../discovery-server
    ports:
      - "8761:8761"
    image: "discovery-service"

    candidat:
    build: ../../candidateMS
    environment:
      - eureka.client.serviceUrl.defaultZone=http://discovery:8761/eureka/
    ports:
      - "3000:8080"
    image: "candidate-service"
    depends_on:
      - job_A

   job_A:
    build: ../../job-MS
    environment:
      - eureka.client.serviceUrl.defaultZone=http://discovery:8761/eureka/
    ports:
      - "8180:8180"
    image: "job-service"
```

**port:** permet d'exposer plusieurs ports de conteneurs à une machine parente. Par exemple ports: "3000:8080".
Elle permet d'exposer le port 8080 au port 3000 de la machine parente.Nous naviguerons ici vers localhost:3000.

**build :** permet de définir le lien vers le dossier racine contenant le Dockerfile pour réaliser le build

**image :** permet de définir le nom de l'image à générer

**depends_on :** permet de définir les autres services dont le service dépend. Nous avons supposé ici que le service candidat dépend du service job. Ceci permet de contrôler l'ordre de démarrage et d'arrêt du service.

Il existe d'autres configurations possibles, telles que volumes, links, command…. [https://github.com/compose-spec/compose-spec/blob/master/spec.md ] [https://docs.docker.com/compose/ ]

Il faudrait d'autre part changer dans le fichier application.properties du microservice à enregistrer localhost par le nom attribué au serveur de découverte comme suit :

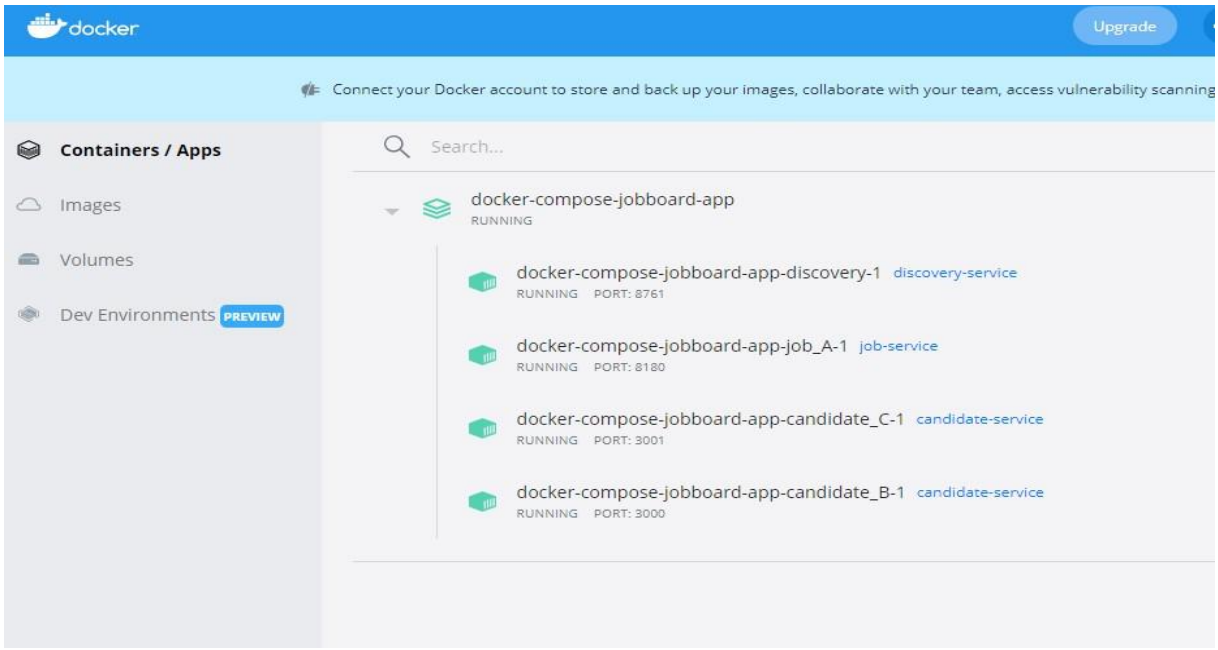
```
eureka.client.server-url.default-zone=http://discovery:8761/eureka/
```

# Etape 2: Lancer l'application multi-containers

Pour lancer l'application multi-containers, il suffit de lancer la commande :

```
docker-compose up
```

Nous avons ainsi les images créées et démarrées :




Pour appeler les différents services lancés, consulter les liens :

http://localhost:8761/
http://localhost:8080/candidats/
http://localhost:8180/jobs

Tous les services sont listés et référencés par le serveur de découverte Eureka :

Pour stopper les applications, il suffit de lancer la commande :

```
docker-compose down
```

Taper la commande #**docker images ls** pour lister les images Docker locales.