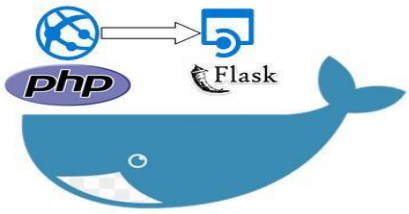
Mini-projet : Conteneuriser une application

L'application est créée en langage Python et il faut la déployer sur docker avec comme frontend en php et un backend fait en python et qui tourne en Flask.



1. Objectifs

Objectifs Pratiques : Gestion de l'Infrastructure Docker

L'objectif de ce travail pratique est de s'assurer que vous êtes capable de gérer une infrastructure Docker. Vous serez évalué sur les thèmes suivants :

- **Améliorer le Processus de Déploiement d'une Application Existant**
- **Gestion des Versions de la Release de l'Infrastructure**
- **Adopter les Meilleures Pratiques pour Implémenter une Infrastructure Docker**
- **Infrastructure as Code (IaC)**

2. Contexte

SUPMIT est une entreprise informatique située au Maroc qui développe des logiciels pour les universités.

Le département d'innovation souhaite améliorer l'infrastructure existante pour garantir qu'elle soit évolutive, facilement déployable et avec un maximum d'automatisation.

SUPMIT souhaite que vous construisiez un "POC" (Proof of Concept) pour démontrer comment Docker peut aider et à quel point cette technologie est efficace.

Pour ce POC, SUPMIT vous fournira une application et souhaite que vous construisiez une infrastructure "découplée" basée sur "Docker".

Actuellement, l'application fonctionne sur un serveur unique, sans évolutivité ni haute disponibilité.

À chaque fois que SUPMIT doit déployer une nouvelle version, quelque chose ne va pas.

En conclusion, SUPMIT a besoin d'agilité pour sa ferme logicielle.

3. **Infrastructure**

Pour ce POC, vous n'utiliserez qu'une seule machine avec Docker installé dessus.

La construction (build) et le déploiement se feront sur cette machine.

La sécurité est un aspect très critique pour la DSI de SUPMIT, donc, s'il vous plaît, **ne désactivez pas le pare-feu** ni d'autres mécanismes de sécurité, à moins que vous ne fournissiez une explication détaillée de vos raisons dans votre livrable.

4. **Application**

L'application sur laquelle vous allez travailler s'appelle "student_list". Cette application est très basique et permet à SUPMIT d'afficher la liste des étudiants avec leur âge.

L'application "student_list" se compose de deux modules :

1. Le premier module est une API REST (avec authentification de base nécessaire) qui envoie la liste souhaitée des étudiants à partir d'un fichier JSON.
2. Le second module est une application web écrite en HTML + PHP permettant aux utilisateurs finaux d'obtenir la liste des étudiants.

Votre tâche est de créer un conteneur pour chaque module et de les faire interagir entre eux.

Le code source de l'application sera partagé.

Les fichiers que vous devez fournir (dans votre livraison) sont :

- Dockerfile
- docker-compose.yml (actuellement vide)

Rôle des fichiers

- **docker-compose.yml** : pour lancer l'application (API et application web).
- **Dockerfile** : le fichier qui sera utilisé pour construire l'image de l'API.
- **requirements.txt** : contient tous les packages à installer pour faire fonctionner l'application.
- **student_age.json** : contient les noms des étudiants et leur âge au format JSON.
- **student_age.py** : contient le code source de l'API en Python.

- **index.php** : page PHP où l'utilisateur final pourra interagir avec le service pour afficher la liste des étudiants avec leur âge. Vous devez mettre à jour la ligne suivante avant de lancer le conteneur du site web afin que api_ip_or_name et port correspondent à votre déploiement :

  $url='http://<api_ip_or_name:port>/SUPMIT/api/v1.0/get_student_ages';

- **docker-compose-registry.yml** : pour lancer le registre local et sauvegarder votre API.

I. Construire (build) et tester l'API (7 points)

SUPMIT vous fournira des informations pour construire l'image de l'API.

✓ **Image de base** :

- Pour construire l'image de l'API, vous devez utiliser l'image "python:3.8-buster".

✓ **Mainteneur** :

- N'oubliez pas de spécifier les informations du mainteneur dans le Dockerfile (votre nom et prénom et votre email).

✓ **Ajouter le code source** :

- Vous devez copier le code source de l'API (**student_age.py**) dans le conteneur à la racine (/).

✓ **Installer les Prérequis :**

- L'API utilise le moteur FLASK, vous devez donc installer certains packages : apt update -y && apt install python3-dev libsasl2-dev libldap2-dev libssl-dev -y

- Copiez le fichier requirements.txt dans le conteneur à la racine (/) pour installer les packages nécessaires au démarrage de l'application.
  Pour lancer l'installation, utilisez cette commande :
  pip3 install -r /requirements.txt

✓ **Données persistantes (volume)** :

- Créez un dossier de données (data) à la racine (/) où les données seront stockées et déclarez-le comme un volume.
  Vous utiliserez ce dossier pour monter la liste des étudiants.

✓ **Port de l'API** :

- Pour interagir avec cette API, exposez le port **5000**.

✓ **Commande de démarrage (CMD)** :

- Lorsque le conteneur démarre, il doit exécuter le fichier student_age.py (copié à l'étape précédente). La commande doit donc être quelque chose comme : CMD ["python3", "./student_age.py"]

- ✓ **Test** :

- Construisez votre image et essayez de la lancer (n'oubliez pas de monter le fichier **student_age.json** à /data/student_age.json dans le conteneur).
- Vérifiez les logs et assurez-vous que le conteneur écoute et est prêt à répondre.

Exécutez cette commande pour vérifier que l'API répond correctement (prenez une capture d'écran pour votre livrable) :

curl -u toto:python -X GET http://<host IP>:<API exposed port>/ SUPMIT/api/v1.0/get_student_ages

Félicitations ! Vous êtes maintenant prêt pour l'étape suivante (docker-compose.yaml).

II. Infrastructure as Code (5 points)

Après avoir testé votre image d'API, vous devez tout regrouper et le déployer avec **docker-compose**.

Le fichier **docker-compose.yml** déploiera deux services :

1. **website** : l'interface utilisateur avec les caractéristiques suivantes :
    - **image** : php:apache
    - **environment** : vous fournirez le **USERNAME** et le **PASSWORD** pour permettre à l'application web d'accéder à l'API via l'authentification.
    - **volumes** : pour éviter que l'image **php:apache** ne fonctionne pas avec le site web par défaut, vous devez lier le site web fourni par SUPMIT. Vous devez avoir quelque chose comme ./website:/var/www/html.
    - **depends_on** : vous devez vous assurer que l'API démarre avant le site web.
    - **ports** : n'oubliez pas d'exposer le port.
2. **API** : l'image construite précédemment devrait être utilisée avec les spécifications suivantes :
    - **image** : le nom de l'image construite précédemment.
    - **volumes** : vous monterez le fichier **student_age.json** dans /data/student_age.json.

- **ports** : n'oubliez pas d'exposer le port.
- **networks** : n'oubliez pas d'ajouter un réseau spécifique pour votre projet.

Supprimez le conteneur que vous avez créé précédemment.

Lancez votre **docker-compose.yml**.

Enfin, accédez à votre site web et cliquez sur le bouton "List Student".

Si la liste des étudiants apparaît, vous avez réussi à dockeriser l'application SUPMIT! Félicitations (prenez une capture d'écran).

III. Docker Registry (4 points)

SUPMIT vous demande de déployer un registre privé et de stocker les images construites.

Vous devez donc déployer :

1. Un **registre Docker (**(https://hub.docker.com/_/registry/ "registry").
2. Une **interface web** (https://hub.docker.com/r/joxit/docker-registry-ui/ "interface") pour voir l'image poussée en tant que conteneur.

N'oubliez pas de pousser votre image sur votre registre privé et de les afficher dans votre livrable.

IV. Livrable (4 points)

Votre livraison doit contenir le lien de votre dépôt GitHub ou GitLab avec votre nom, contenant :

- Un fichier **README** avec vos captures d'écran et explications.
- Les fichiers de configuration utilisés (**docker-compose.yml, docker-compose-registry.yml** et **Dockerfile**).

Votre livrable sera évaluée sur :

- La qualité des explications.
- La qualité des captures d'écran (pertinence, visibilité).
- La structure de votre dépôt.