École Polytechnique de Montréal
Département de génie informatique et génie logiciel
LOG3000 – Processus du génie logiciel

Travail pratique #4 – Infrastructure en tant que code

1. Mise en contexte théorique

L'arrivée des technologies *Infrastructure as a Service (IaaS)* tel que Amazon Web Service, Microsoft Azure ou Google Cloud a permi d'obtenir un niveau de flexibilité pour l'infrastructure sur laquelle un logiciel sera déployé qui n'avait jamais été atteint auparavant dû aux limitations des serveurs classiques. Il est maintenant possible de déployer des machines virtuelles sur demande ou encore de reconfigurer virtuellement un réseau sans jamais avoir besoin de changer physiquement l'infrastructure. Ces technologies ouvrent la porte à de tout nouveaux paradigmes de développement, entre autre en permettant à l'infrastructure d'évoluer avec le projet à très faible coût. Avant la venue des outils tels que Puppet, Vagrant, Chef ou Docker, les infrastructures flexibles devaient être configurées à la main, ce qui rendait la modification de l'infrastructure trop lourd pour être modifiée constamment.

Infrastructure as Code (IaC) offre une solution à ce problème. Ce nouveau paradigme permet de décrire l'infrastructure (machines virtuelles, espaces de stockage permanent, configuration réseau, etc.) à l'aide d'un langage de programmation déclaratif. L'infrastructure peut alors être déployée sur n'importe quel environnement compatible simplement en exécutant une commande. Dans ce travail, nous allons nous intéresser à l'une des technologies de IaC: des conteneurs (avec la technologie Docker).

2. Objectifs

Le but de ce travail pratique est de vous donner un exemple de comment utiliser Docker et comment *dockerizer* un programme existant. Les objectifs spécifiques sont de :

- Découvrir un exemple de projet déployé à l'aide de Docker
- Découvrir comment déployer un projet existant avec Docker
- Comprendre les réalités auxquelles l'industrie fait face lors des déploiements

3. Mise en contexte pratique

Votre client veut déployer une application qu'il a fait développer sur mesure par une firme de développement. Les développeurs ont fournis le code d'opération mais n'ont pas mis l'application en production. Votre client a donc fait appel à un consultant qui a commencé à travailler sur une solution de déploiement basé sur Docker. Malheureusement, celui-ci a dû arrêter de travailler sur le projet pour des raisons personnelles. Votre client s'est alors tourné vers vous pour terminer le travail de leur ancien consultant en faisant fonctionner son application avec Docker.

4. Travail à effectuer

Votre travail consiste donc à utiliser docker pour lancer un container d'une image existante, déployer localement une application existante avec docker et créer votre propre service. N'hésitez pas à faire appel au chargé de travaux pratiques qui, entre autre, donnera une introduction aux technologies. Le travail consiste donc à :

1. **Faire fonctionner docker sous Photon OS :** Docker n'étant pas disponible nativement dans les laboratoires Windows, nous pouvons l'installer sous linux dans une machine virtuelle. La procédure est décrite dans un fichier disponible sur moodle.
    - Notez que vous pouvez sauter cette étape si vous désirez travailler sur votre propre ordinateur ou via ssh sur un linux de votre choix où docker est installé.

2. **Comprendre Docker:** Dockerfile et docker-compose
3. **Répondre aux questions**

4.1 Comprendre Docker

Pour vous aidez à bien comprendre docker, voici quelques commandes à tester.

- `docker run ubuntu uname -a`

Cette commande exécute la commande “uname -a” dans un container qui a été construit à partir de l’image “ubuntu”, disponible sur le registre d’images de docker.io.

- `docker container ls --all`

Cette commande va faire la liste de tous les containers qui existent. Notez que le container que vous avez créé à l’étape précédente n’est plus actif.

- `docker run -it --rm ubuntu bash`

Cette commande va créer un container et ouvrir une session bash à l’intérieur. Vous pouvez maintenant explorer le contenu du container. Lorsque vous quitterez le container avec la commande “exit”, celui-ci n’est pas seulement désactivé mais aussi détruit à grâce à l’option --rm.

- `docker run -d --name db mongo:latest`

Cette commande va créer et lancer un container nommé “db” à partir de l’image “mongo”. Le container est lancé en mode détaché, ce qui signifie qu’il est exécuté en arrière-plan. À l’aide de la commande `docker ps` vous pourrez voir qu’il est en effet actif.

- `docker top db`

Cette commande affiche les processus dans le container “db”

- `docker exec -it db mongo --version`

Cette commande exécute dans le container la commande “mongo --version” et affiche le résultat.

- `docker exec -it db bash`

Cette commande ouvre une session bash interactive dans le container. Vous pouvez explorer son contenu et quitter avec la commande “exit” et noter que le container reste actif.

4.2 Configurer l’application

Dans cette section, vous allez commencer à faire fonctionner l’application de TP4

1. Aller chercher le code avec:
   - `git clone https://github.com/foundjem/log3000.git`

Explorer un peu le contenu du projet. Comprenez comment le code source et le fichier docker-compose.yml fonctionne. Votre tâche est de compléter les fichiers Dockerfile et docker-compos et installez toutes les dépendances.

2. Lancez le stack après avoir apporté les modifications:
   - `docker-compose up --build`

3. Votre application doit être en mesure d'échelle jusqu'à un maximum de 10 conteneurs.

5. Questions

5.1 Questions d'analyse (à répondre brièvement)

1. Quelle est la différence entre une image et un conteneur?
2. Comment faire pour effacer un volume dans docker ?
3. Quels sont les avantages d'un conteneur par rapport à une machine virtuelle?
4. Décrivez les avantages de déployer des applications avec un système d'IaC tel que Docker par rapport à une machine virtuelle?
5. Quel est l'avantage du fichier docker-compose.yml vis-à-vis l'utilisation de simples Dockerfiles? Utilisez un tableau pour comparer les fonctionnalités principales de chacun et expliquez les avantages et désavantages entre les deux solutions.
6. Quelle est la différence entre un volume et un "bind mount" dans docker?
7. Quels sont les avantages d'exécuter une application de microservice sur docker?
8. Est-il conseillé d'exécuter plusieurs applications sur docker, pourquoi/pourquoi pas?
9. Quel type de volume utiliserez-vous dans cette application? Répertoriez et expliquez tous les types de volumes de docker existants.
10. Mise à l'échelle de votre application à un maximum de 10 conteneurs?

5.3 Question de rétroaction

Nous travaillons à l'amélioration continue des travaux pratiques de LOG3000. Cette question peut être répondue très brièvement.

1. Combien de temps avez-vous passé sur le travail pratique, en heures-personnes, en sachant que deux personnes travaillant pendant trois heures correspondent à six heures-personnes. Est-ce que l'effort demandé pour ce laboratoire est adéquat ?
2. Quelles difficultés avez vous rencontré lors de ce laboratoire?

6. Livrable à remettre, procédure de remise et retard

Les fichiers à remettre sont les suivants :

- Votre rapport en pdf contenant les réponses aux questions posées dans la section 5.
- Toute modification que vous avez apportée au code source; Dockerfile ou docker-compose des fichiers

Seule une remise électronique est exigée. La remise doit se faire sur moodle
avant dimanche, le **31 mars à 23h55 PM**.

Annexe A: Docker

Voici quelques commandes utiles pour utiliser docker et docker-compose.

Créer une image avec le répertoire courant
```
docker build .
```

Lister les images
```
docker images
```

Lancer un container
```
docker run -d IMAGE
```

Lister les containers actifs
```
docker ps
```

Se connecter à un container actif
```
docker run -it CONTAINER /bin/sh
```

Clean et build un stack
```
docker-compose build --no-cache
```

Lancer un stack
```
docker-compose up
```

Annexe B: Photon OS

Trouver l'adresse IP de la VM
```
ifconfig eth0 | grep "inet addr" | cut -c 21-34
```

Annexe C: Tutoriels et références

Dockerfiles:

- https://docs.docker.com/get-started/part2/
- https://docs.docker.com/engine/reference/builder/#from
- https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images

Docker-compose:

- https://docs.docker.com/get-started/part3/#introduction
- https://docs.docker.com/compose/compose-file/
- https://docs.docker.com/compose/gettingstarted/#step-2-create-a-dockerfile