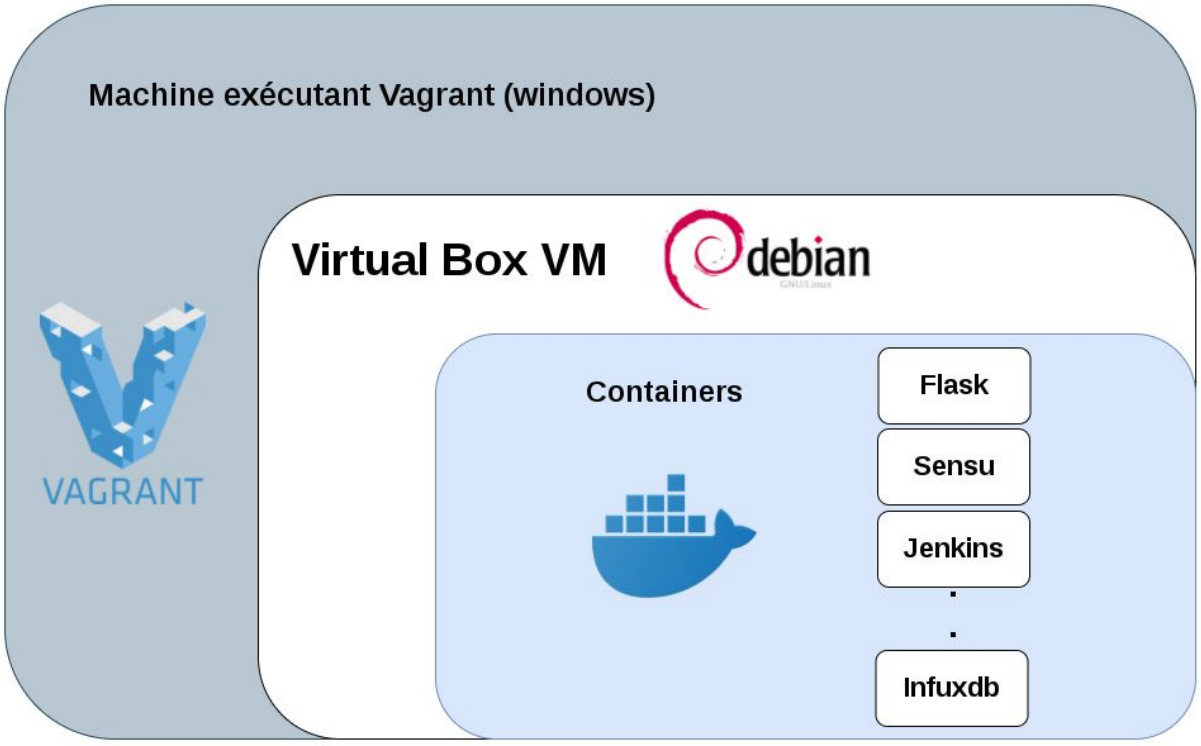
TP - DevOps sur Linux

Sylvain DAVID - Novembre 2020

Le but de ce TP est de mettre en oeuvre un ensemble de technologie DevOps Linux. Le schéma suivant décrit simplement l'architecture cible :



Dans ce TP vous allez :

- automatiser le déploiement d'un environnement de développement
- automatiser le processus de construction d'une application
- superviser une machine
- récupérer et visualiser quelques indicateurs clés de production

Note :
Vous allez avoir besoin de télécharger certains fichiers nécessaires à la réalisation du TP via le dépôt git (hebergé sur github) suivant :

- https://github.com/adrenalinux/TP-SRT-4A

Etape 1 : Création de l'environnement de déploiement

Présentation de vagrant

Vagrant est un outil permettant de créer et gérer des machines virtuelles locales facilement. Le but d'un outil comme vagrant est de pouvoir automatiser la création d'un environnement de développement ou de test via l'utilisation de la virtualisation.

Installation de Vagrant

Dans un premier temps, vous devez télécharger et installer le logiciel Vagrant sur votre machine (si cela n'est pas déjà fait) en suivant les indications contenus sur le site web de Vagrant. Il sera aussi utile pour vous de vérifier que l'outil de virtualisation VirtualBox est aussi installé.

Mise en route et configuration d'une machine virtuelle Linux

Une fois ces outils installés, créer un dossier sur votre machine qui contiendra toutes les ressources qui seront créées durant ce TP et placez vous dans ce dossier avec un terminal.

En vous aidant de la document en ligne de vagrant :

- Initialisez la box suivante : *debian/contrib-jessie64*
- Démarrez la box
- Connectez vous dessus

Questions :

- Quelles commandes avez-vous utilisées pour réaliser les étapes précédentes ?
- Que remarquez vous sous la racine de votre dossier de travail ?
- Avec quel utilisateur êtes vous connecté à la machine (par défaut) ?
- A l'aide la commande `df` (exécutée sur la machine guest) mettez en évidence un système de fichiers étrange, à quoi correspond ce système de fichiers ? Quel en est l'intérêt ?

Toujours en vous aidant de la documentation en ligne de vagrant (voir Vagrantfile) réaliser les opérations suivantes :

- Ajouter une IP statique privée (par exemple: *192.168.85.85/24*) ;
- Changer le nom de la machine en "*TP-VM*" ;
- Ajouter un proxy si cela est nécessaire.

Questions :

- Fournissez les lignes correspondantes à ces changements de configuration.
- Citez deux avantages de l'usage de Vagrant par rapport à de la virtualisation classique.

Etape 2 : Automatisation de l'installation de Docker avec Ansible

Présentation de Docker

Docker est une plateforme de virtualisation par conteneur. Par rapport à la virtualisation basée sur hyperviseur (KVM, VirtualBox...) qui émule entièrement un système d'exploitation, la virtualisation par conteneur permet d'isoler un processus dans un environnement d'exécution indépendant (du système hôte, mais aussi des autres conteneurs, qui ne partagent alors que le même noyau). Cet environnement d'exécution isole donc les aspects processeurs, RAM, réseau, système de fichiers (etc ...) du conteneur.

Présentation de ansible

Ansible est une plateforme logicielle open-source pour la configuration et la gestion de machines. Cette solution combine :

- du déploiement logiciel sur une multitude de nœuds ;
- de l'exécution de tâches ad-hoc ;
- et de la gestion de configuration.

Bien que Ansible soit relativement jeune par rapport à ses concurrents (comme Puppet ou Chef), la première version datant de Février 2012, il se distingue par sa facilité d'utilisation et d'apprentissage. En particulier, Ansible repose sur une architecture sans serveur. Alors que ses prédécesseurs requièrent d'installer un daemon (agent) particulier sur les machines à administrer, Ansible a fait le choix intelligent de reposer sur le protocole SSH. Ainsi, un simple serveur SSH sur les machines à administrer suffit pour orchestrer un cluster potentiellement infini de machine. De manière plus général, Ansible permet de pousser n'importe quelle configuration vers un ensemble de nœuds, à condition de disposer d'un client SSH et de Python (le langage dans lequel est écrit Ansible).
Afin d'orchestrer cet ensemble de nœuds, Ansible utilise des modules qui permettent de réaliser différentes actions (comme exécuter une commande shell ou installer des packages). Le projet en fournit déjà pas mal (plus de 2000 actuellement) mais rien ne vous empêche de créer les vôtres. Le fichier contenant la description des tâches que Ansible doit accomplir, appelé playbook, est rédigé en YAML. YAML est un format de représentation de données (reprenant quelques concepts d'XML) simple, efficace et moins verbeux que XML. Il est ainsi très lisible par des humains.

Installation manuelle de Docker

En suivant la documentation en ligne disponible à l'adresse suivante :
https://docs.docker.com/install/linux/docker-ce/debian/#install-using-the-repository

- Installez Docker sur la machine “guest” Debian Jessie
- Testez que Docker fonctionne

Installation de Ansible et prise en main succincte

Installation de Ansible

Installez ansible avec la commande suivante :

```
# apt-get install ansible
```

Définition d’un inventaire

Nous allons essayer de pinguer notre machine locale. Pour cela, il est nécessaire de renseigner cette machine dans un fichier inventaire, par exemple : *inventory*.
Définissons le groupe “*default*” contenant la machine “TP-VM” dans notre fichier *inventory*.

```
-----------------------------------------------------------------
TP-VM  ansible_connection=local

[default]
TP-VM
-----------------------------------------------------------------
```

Enfin, on exécute le module ping d’Ansible sur notre groupe :

```
# ansible -i inventory default -m ping
```

Note sur les modules Ansible :

Il est possible de récupérer la liste des modules grâce à la commande :

```
# ansible-doc -l
```

Vous aurez également besoin de la commande suivante pour accéder à la documentation d’un module (par exemple ici on souhaite accéder à la documentation du module shell) :

```
# ansible-doc shell
```

Note sur l’utilisation des playbooks :

Bien qu’il soit possible d’exécuter une simple commande à la main comme nous venons de le faire, il est préférable la plupart du temps d’automatiser plusieurs actions à réaliser (par exemple lors du déploiement d’un serveur).

Exemple d'un playbook qui installe un serveur web Apache :

```
- hosts: webservers
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    apt:
      name: apache2
      state: latest
  - name: ensure apache is running
    service:
      name: apache2
      state: started
```

Ecriture d'un playbook ansible pour installer Docker

Afin d'automatiser l'installation (effectuée précédemment) de Docker sur la machine "guest", nous allons écrire un playbook Ansible.

Question :

- A l'aide de la documentation en ligne d'ansible complétez le playbook "`playbook-docker.yml`" fournit dans le dépôt git puis testez que cela fonctionne correctement. Il faudra fournir la commande permettant de tester et le playbook modifié.

Maintenant que l'installation de Docker est automatisée nous allons maintenant appeler ce playbook Ansible depuis Vagrant. (Aide : il faut utiliser "le provisionner" : *ansible_local*)

Questions :

- Modifier le fichier Vagrantfile afin d'appeler automatiquement ce playbook via Vagrant.
- Quelle commande permet de "provisionner" la machine "guest" Debian ?

Etape 3 : Dockerisation d'une application Flask

Prise en main succincte de Docker

Nous allons mettre en place un conteneur basé sur une image Ubuntu. Une image correspond à un système de fichier qui contient ce que l'on a décidé d'y installer (une distribution ubuntu plus une base de donnée par exemple). Un conteneur est une instance d'une image. Nous allons récupérer une image ubuntu depuis le docker hub et vérifierons qu'elle est bien stockée localement :

```
# docker pull ubuntu:bionic-20181018
```

```
# docker images
```

À présent, nous allons charger cette image dans un conteneur qui va exécuter une commande :

```
# docker run ubuntu:bionic-20181018 echo "Hello World"
Hello World

# docker ps
```

La commande `docker ps` permet d'afficher les processus en cours.

Questions :

- Qu'affiche cette commande et pourquoi ?

Comparez la avec le résultat des commandes suivantes :

```
# docker run -d --name my_ubuntu ubuntu:bionic-20181018 tail -f /etc/passwd
# docker ps
```

- À quoi sert ce paramètre `-d` ? (Aide : la commande `docker ps -a` peut vous aider)

Il est possible de se connecter à un terminal d'un conteneur. Pour cela on utilise la commande suivante :

Si l'on souhaite créer un nouveau conteneur :
```
docker run -it ubuntu:bionic-20181018 /bin/bash
```

Si l'on souhaite se connecter sur un conteneur qui tourne :
```
docker exec -it my_ubuntu bash
```

Question :

- Que signifie l'option "`-it`" ?

Vous pouvez obtenir toutes les informations de vos conteneurs par la commande suivante :
```
# docker inspect my_ubuntu
```

Dans la suite nous verrons que les conteneurs peuvent exécuter des processus plus complexes. En particulier, ils sont adaptés aux "daemons".

Dockerisation d'une application

Nous allons construire une application Python Flask des plus minimaliste. Cette application est un mini serveur web qui affiche une page web. Vous pouvez récupérer l'application et son "Dockerfile" depuis le dépôt git du TP (fichier docker-flask/Dockerfile).

Questions :

- Expliquer très succinctement le contenu du fichier "Dockerfile".
- Quelle commande doit-on utiliser pour construire l'image ?
- Quelle commande doit-on utiliser pour démarrer le conteneur ? Pourquoi ajouter un mapping de port ? Quelle URL doit-on taper pour afficher la page web de l'application Flask ? (Aide 1 : le port par défaut du framework flask est 5000) (Aide 2 : n'oubliez pas que le container fonctionne à l'intérieur d'une machine virtuelle et que vous avez attribué une IP à cette machine)

Etape 4 : Mise en place d'un serveur de build centralisé

Présentation de Jenkins

Jenkins est un outil d'Intégration continue open source écrit en Java. Bénéficiant d'une part de marché dominante, Jenkins est utilisé par des équipes de toutes tailles, pour des projets dans des langages et des technologies variés.

Installation du Docker Jenkins

Nous allons utiliser une image Docker du serveur Jenkins. Pour cela vous pouvez récupérer le "Dockerfile" depuis le dépôt git.

Vous pouvez construire l'image "jenkins" avec la commande suivante :

```
docker build -t jenkins --no-cache .
```

Démarrer ensuite une instance avec la commande suivante :

```
docker run -d -v jenkins_home:/var/jenkins_home \
 -v /var/run/docker.sock:/var/run/docker.sock \
--mount type=bind,source=/vagrant/TP-SRT-4A/docker-flask,target=/app -p \
8080:8080 -p 50000:50000 jenkins
```

Question:
Expliquer succinctement la commande de démarrage de l'instance basée sur l'image Jenkins.

Ouvrez ensuite une page web sur *192.168.85.85:8080* puis configurez succinctement Jenkins (notez bien les informations concernant le compte administrateur car vous allez l'utiliser ultérieurement).
Aide pour "débloquer jenkins", il faut utiliser le mot de passe généré et qui est visible en utilisant la commande suivante :

```
docker logs <le nom de votre container>
```

Le nom du container est visible avec la commande :

```
docker ps
```

Lors de l'installation il sera aussi nécessaire d'installer le plugin "Docker Pipeline plugin".

Création d'un pipeline Jenkins

Nous allons maintenant ajouter un pipeline (`nouveau item -> Pipeline`) afin de builder automatiquement l'image du Docker Flask à chaque commit.

Configurez le pipeline en activant les options suivantes :

- `Scrutation de l'outil de gestion de version`
- `Déclencher les builds à distance` (par exemple, à partir de scripts)
    - et définissez un jeton d'authentification (notez bien le token utilisé)

Définissez le Pipeline sur :

- `Pipeline script from SCM`
- `SCM choisissez "git"`

Puis mettez comme "Repository URL" (attention aux 3 '/'):

- `file:///app`

Puis initialiser le dépôt git de l'application flask (à la racine du répertoire docker-flask) en utilisant les commandes git suivantes (à compléter):

- `git init ...`
- `git add ...`
- `git commit ...`

Questions :

- Lancer un build depuis l'interface puis observer ce qui se passe, où sont définis les étapes du build ?

Il est possible de lancer un build avec la commande http suivante (il sera nécessaire d'installer curl sur la machine host "debian" avec la commande suivante: *apt install curl*):

```
curl --user <user>:<pwd> http://<ip>:8080//job/<projet-jenkins>/build?token=<token>
```

- Ajoutez la commande ci-dessus dans un "hook post-commit" du dépôt git de l'application flask (s'aider de la documentation git si besoin).
- Puis ajoutez/modifiez votre dépôt en ajoutant un commit.
- Quel est le chemin du fichier "hook" à ajouter ?
- Qu'observez vous lors d'un commit ?

- Quelle est l'intérêt d'avoir une API pour un logiciel comme Jenkins ?
- Citer 2 avantages procurés par l'utilisation d'un serveur de build automatique.
- Quelles autres actions seraient avantageuses à ajouter ?

Etape 5 : Mise en place d'un serveur de monitoring

Présentation de sensu

Sensu est une solution de monitoring d'infrastructure permettant aussi bien de résoudre des problèmes de surveillance sur une grande quantité de machines que sur des environnements plus réduits. Cette large couverture est obtenue grâce à l'utilisation de deux primitives de surveillance simples mais puissantes: les contrôles de service (les checks) et le traitement des événements (handlers). Avec ces primitives il est possible de construire des pipelines extensibles à l'infini pour composer une solution de surveillance des plus complète.

Mise en place d'une solution de monitoring

Dans le dépôt git vous pourrez trouver sous le répertoire "`monitoring-dockers`" un fichier `docker-compose.yml` qui permet l'utilisation d'un ensemble de conteneurs.

Questions :

- A quoi sert `docker-compose` et quel est l'intérêt de l'utiliser ?
- Décrivez succinctement le rôle des différents conteneurs utilisés.
- Comment démarrer ces conteneurs ?
- Que remarquez vous au niveau des volumes Docker utilisés ?

Une fois les conteneurs dockers lancés, connectez vous sur l'interface utilisateur de sensu (port 3000):

- Quels sont les "checks" définis et dans quels fichiers de configuration sont définis ces checks?

On souhaiterait maintenant monitorer aussi la machine "host", pour cela il faut ajouter le démon `sensu-client` sur la machine "host" Debian.

A l'aide du playbook "`playbook-sensu.yml`" installer le client *sensu*.

- Expliquer succinctement la notion de "rôle" utilisé dans ce playbook Ansible et son intérêt ici.

Vérifier que la machine “guest” Debian est bien visible dans l’interface utilisateur de sensu.

Essayez la commande suivante depuis la machine “guest” :

```
# curl -s http://localhost:4567/events | jq .
```

- Quelle est l’intérêt d’avoir une API comme celle-çi ?

Etape 5 : Stockage de métriques et visualisation (Bonus)

Grafana et influxdb sont deux outils qui ont été installés conjointement avec Sensu.

Questions :

- Expliquez très succinctement leurs fonctionnements et leurs utilités ?

A l’aide de la documentation en ligne de grafana et d’influxdb, configurer grafana (ajout d’une source de données de type influxdb) et ajouter un nouveau dashboard contenant les graphiques suivants :

- Nb. d’image Docker en fonction du temps
- Nb. de conteneurs en cours d'utilisation en fonction du temps
- Volume d’espace disque utilisé en fonction du temps
- Charge moyenne par minute

Pour chaque graphique, il sera nécessaire de fournir la requête exécutée sur le serveur influxdb.

Notes pour l’accès et la configuration de l’outil Grafana:

- Grafana utilise le port TCP 5555 donc l’accès se fait par l’URL http://192.168.85.85:5555
- L’accès par défaut est *admin /secret*
- L’URL d’accès pour la “data source” influxdb est : http://rozofs-influxdb:8086
- Le nom de la database influxdb est: *sensu*