24/07/2020

Le Coz Théo

Etudiant en informatique de confiance 2eme année

Rapport de projet :

Service de déploiement automatisé de conteneur

**Table des matières**

[Concept : 2](#_Toc46819841)

[Objectif : 2](#_Toc46819842)

[Réalisation : 2](#_Toc46819843)

[Interface API 2](#_Toc46819844)

[Utilisation du moteur *Docker* 3](#_Toc46819845)

[Gestion des projets 3](#_Toc46819846)

[Exposition des conteneurs 3](#_Toc46819847)

[Monitoring 3](#_Toc46819848)

[Mise à l’échelle 4](#_Toc46819849)

[Sécurité : 4](#_Toc46819850)

[Les dangers ? 4](#_Toc46819851)

[Le moteur docker 4](#_Toc46819852)

[L’API 4](#_Toc46819853)

[Les conteneurs 4](#_Toc46819854)

[Retour d’expérience 4](#_Toc46819855)

# Concept

Il y a, à la base de ce projet, le constat qu’il n’y a pas de plateforme permettant de simplifier et d’automatiser le déploiement de projet micro-service. Pour ce genre de projet les solutions à disposition sont les services cloud qui propose des services d’hébergement de container même si des serveurs virtuels avec un moteur *Docker* est généralement privilégié afin d’avoir plus de contrôle.

Le but était donc d’imaginer un service permettant facilement à l’aide d’un simple fichier de déployer un projet micro-service, à l’image de *Docker*-*Compose*.

# Objectif

Afin de répondre aux attentes énoncées plus tôt nous avons besoin d’un certain nombre de fonctionnalités. Tout d’abord une API recevant les requêtes et faisant appel à des services internes pour répondre aux demandes. Ensuite dans les services internes il faut :

* Un service de création d’image.
* Un service de déploiement de conteneur sur le moteur *Docker.*
* Un service de gestion de projet afin de gérer les différents conteneurs du projet.
* Un service d’exposition des conteneurs pour pouvoir dédier une adresse à un service du projet afin qu’il puisse être accessible de l’extérieur.
* Un service de monitoring permettant de récolter les données d’exploitation comme la consommation CPU, RAM et réseaux et de les exposer au client.
* Un système permettant de mettre à l’échelle les projets en taille mais aussi les localiser géographiquement.

Ces 6 services internes sont, pour moi, les services minimums permettant de répondre à la demande initiale.

# Réalisation

Fort de ces observations, j’ai commencé la réalisation d’une preuve de concept permettant de valider la réalisation d’un tel service. Au vu de la nature du projet je me suis orienté sur une architecture de type micro-service, le tout hébergée dans des conteneurs.

## Interface API

J’ai commencé par la réalisation de l’interface API qui reçoit les différentes requêtes. Cette partie étant relativement simple car dépendant des autres fonctions disponible, je suis passé à la partie authentification et l’inscription des utilisateurs. J’ai voulu créer ces fonctions le plus simplement possible n’étant pas le cœur du projet.

Un utilisateur s’authentifie avec son nom d’utilisateur et son mot de passe qui sont ses seules informations données lors de l’inscription. Une fois authentifié l’utilisateur reçoit un jeton qu’il donnera avec son nom d’utilisateur pour tout requête à l’API.

## Utilisation du moteur *Docker*

J’ai ensuite commencé la partie création d’image et déploiement de conteneur. Etant donné que ces services sont eux-mêmes dans des conteneurs nous avons besoin qu’ils aient accès au moteur de l’hôte qui les héberge. Pour cela nous utilisons la fonctionnalité volume de *Docker* qui permet de copier un fichier de l’hôte, ici le socket du moteur *Docker*, dans le conteneur permettant à celui-ci de faire appel au moteur docker comme s’il était installé dans le conteneur.

Une fois cela fait les fonctions ont été facile à réaliser car *Docker* fournie un SDK facilitant l’utilisation du socket pour le développement.

## Gestion des projets

Viens ensuite la gestion de projet et des conteneurs. Cette partie fut la plus longue car demandant la coordination des deux services précédant en plus du service d’exposition.

Le principe est de pouvoir à partir d’un fichier type *Docker-Compose* ou en décrivant soi-même les options et avec un répertoire *GitHub* de pouvoir créer une ou plusieurs images et de les déployer sur l’hôte.

Pour ce faire il faut réaliser un nommage précis des images et conteneurs car ils doivent être unique, tout en restant transparent pour l’utilisateur qui doit pouvoir avoir le nommage le plus simple et intuitif possible.

## Exposition des conteneurs

Une fois cela réalisé je suis passé à l’exposition dans conteneur vers l’extérieur. Pour cela j’ai utilisé le projet open source *Traefik* qui est un reverse proxy dédié à la conteneurisation.

Son implémentation n’a pas été trop complexe étant donné son principe de fonctionnement qui repose sur du paramétrage directement avec des labels qui sont donné au lancement du conteneur et qui sont alors récupéré par *Traefik*. Une fois ces informations récupérées, *Traefik* créé une route pour chaque conteneur dynamiquement en faisant appel aux APIs de notre provider DNS.

## Monitoring

Concernant la partie monitoring je n’ai pu qu’esquisser sa mise en place grâce à plusieurs projets open source. On peut découper cette partie en trois agents :

L’agent de récolte qui comme son nom l’indique récolte les données à monitorer comme la charge CPU, RAM et réseaux.

Ces données sont alors transférées dans une base de données dite orientée séries chronologiques, ici *influxDB*, qui permet le stockage d’informations avec chronologie.

Enfin il faut un agent permettant de restituer ces données sous forme graphique afin d’être exploitées par l’utilisateur.

Les deux premiers agents ont été relativement facile à mettre en place car ayant une certaine compatibilité avec la conteneurisation. Cependant le dernier agent reste complexe à mettre en place et n’a pas pu être développé entièrement, car les données doivent être séparées entre chaque utilisateur pour éviter qu’un utilisateur puisse avoir accès à des données d’un autre utilisateur.

## Mise à l’échelle

Enfin pour la mise à l’échelle des projets, même si cela n’a pas pu être mis en place il existe *Docker Swarm* qui est incorporé à *Docker* qui permet de créer des clusters d’hôte *Docker* et ainsi démultiplier le nombre de conteneur avec une gestion interne des réplicas.

# Sécurité

## Les dangers ?

Étant donné sa structure qu’on pourrait qualifier de mutualiser, si une personne venait à compromettre le service il pourrait, à partir d’un projet d’un utilisateur, accéder aux données des autres utilisateurs ce qui est problématique. Serait alors accessible non seulement les données des comptes utilisateurs mais aussi les conteneurs. Ceux-ci pourraient être compromis en remplaçant les images utilisées ou en les modifiants. De plus les données des conteneurs sont potentiellement accessibles si l’attaquant a un accès à la machine hôte.

## Le moteur docker

Etant donné que le socket docker est utilisé par les conteneurs par l’intermédiaire d’un simple fichier, il reste très exposé aux attaques et à la compromission des conteneurs qui l’utilise. Pour pallier cela il faudrait mettre en place une connexion par socket TLC ou même par SSH.

## L’API

L’un des points les plus exposé est l’API. En effet, celle-ci est accessible par tout le monde et à directement accès au service qui effectue les actions.

Il faut donc avoir des protections renforcées pour les différents types d’injection et le vol de session. De plus il est nécessaire d’avoir une surveillance accrue de ce service afin de repérer les actions suspectes.

## Les conteneurs

Une autre source de compromission sont les conteneurs eux-mêmes. En effet un attaquant peut lui-même déployer un projet qui contiendrait un conteneur utilisant une faille dans le moteur Docker ou dans ces librairies permettent d’outre passer l’isolation des conteneurs et ainsi accéder à d’autre conteneur ou même à l’hôte.

Pour éviter ce genre de chose il faut analyser les images et répertoire utilisé afin d’identifier de potentiel agent infectant. En plus de cela il faut évidemment rester au courant des différentes failles découvertes sur Docker et d’appliquer systématiquement les mises à jour de sécurités.

Retour d’expérience

Ce projet m’a permis d’expérimenter en condition réelle la conception d’un projet de A à Z. J’ai pu expérimenter le fait de planifier le développement et d’essayer de respecter ce planning. Coté développement j’ai pu acquérir de nouvelle compétence dans le développement notamment sur la gestion de projet avec git et sur le langage python. J’ai pu me familiariser avec la technologie de conteneurisation notamment avec Docker qui a été le centre de ce projet. J’ai pu rencontrer des difficultés dans le respect des dates intermédiaire car j’ai souvent sous-évalué le travaille et le temps de développement.