DÉPLOYER, MANIPULER ET SÉCURISER UN SERVEUR REGISTRY DOCKER PRIVÉ

Introduction

Dans ce chapitre, nous allons nous intéresser à la partie Registry (en fr : Registre) dans Docker.

C'est quoi ?

Le Registry Docker est un **système de stockage et de distribution d'image Docker** open-source (sous la licence Apache), déployé côté serveur. Il permet aux utilisateurs d'**extraire et insérer des images Docker dans un dépôt avec les autorisations d'accès appropriées**. La même image peut avoir plusieurs versions différentes, identifiées par leurs tags.

Quand l'utiliser ?

Vous devez utiliser le Registry si vous voulez :

- Contrôler étroitement l'endroit où vos images sont stockées
- Posséder pleinement le contrôle de votre pipeline de distribution d'images
- Intégrer étroitement le stockage et la distribution des images dans votre flux de travail de développement interne

Déploiement et manipulation d'un Registry privé

Création du Registry privé

Par défaut, le moteur Docker interagit avec le DockerHub. DockerHub est le registre officiel de Docker offrant une solution prête à l'emploi, que nous avons déjà eu l'occasion d'utiliser ici . Il ne nécessite aucune maintenance et fournit un Registry gratuit, ainsi que des fonctionnalités supplémentaires telles que des comptes d'organisation, une intégration à des solutions de contrôle de source notamment Github et Bitbucket, etc. Mais dans ce cours, nous allons nous intéresser au **déploiement d'un serveur Docker Registry privé**.

Avant de pouvoir déployer un Registry, vous devez d'abord installer le moteur Docker sur la machine hôte qui hébergera vos images Docker, car un Registry n'est rien d'autre qu'une image Docker qui attend à être exécutée. Voici la commande qui permet de **créer un Docker Registry privé** :

```
docker run -d -p 5000:5000 --restart=always --name mon-registry registry:2
```

Déposer une image dans le Registry privé

Premièrement, nous allons créer une image personnalisée (alpine + git) que nous allons par la suite déposer dans notre Registry. Commencez d'abord par créer un Dockerfile et ajoutez dedans le contenu suivant :

```
FROM alpine:latest
RUN apk add --no-cache git
```

Construisez ensuite votre image :

```
docker build -t alpinegit .
```

En vue de pousser notre image dans notre Registry, il faut au préalable **créer un nouveau tag de votre image** en respectant le format suivant :

```
docker tag alpinegit <SERVER NAME REGISTRY>:<PORT SERVER REGISTRY>/<CONTAINER NAME>
```

Soit :

```
docker tag alpinegit localhost:5000/alpinegit
```

Dès à présent, vous pouvez **envoyer votre image vers le registry docker privé** :

```
docker push localhost:5000/alpinegit
```

Visualiser les images disponibles dans le Registry privé

Une fois votre image envoyée, il est possible de la visualiser grâce à l'**API du Docker Registry**. Je vais utiliser la commande curl pour manipuler l'API du Registry.

Voici l'url qui permet de **lister les différentes images dans votre Registry Docker** :

```
curl -X GET http://localhost:5000/v2/_catalog
```

Résultat :

```
{"repositories":["alpinegit"]}
```

Nous allons créer une nouvelle image, qu'on pushera ensuite dans notre registry privé, dans le but de savoir si l'API Docker Registry prend bien en compte notre nouvelle image :

Dockerfile

```
FROM alpine:latest
RUN apk add --no-cache mysql-client
ENTRYPOINT ["mysql"]
```

Vous connaissez maintenant la musique , en construit l'image, change son tag et on la push :

```
$ docker build -t alpinemysql .
$ docker tag alpinemysql localhost:5000/alpinemysql
$ docker push localhost:5000/alpinemysql
```

Vérifions ensuite si l'api nous retourne bien notre nouvelle image :

```
curl -X GET http://localhost:5000/v2/_catalog
```

Et c'est bien le cas :

```
{"repositories":["alpinegit","alpinemysql"]}
```

Si vous voulez, vous pouvez supprimer vos images localement à l'aide de la commande `docker rmi` et vous verrez que vos images seront toujours disponibles dans votre docker resigtry, car ces dernières sont stockées dans votre conteneur registry. Par la suite vous pouvez **récupérer vos images depuis votre registry privé** avec la commande suivante :

```
docker pull localhost:5000/alpinegit
```

```
docker pull localhost:5000/alpinemysql
```

Visualiser les différents tags d'une image dans le Registry privé

Dans cette partie, nous allons rajouter un nouveau tag nommé "test" sur notre image `alpinegit`,

```
docker tag alpinegit localhost:5000/alpinegit:test
```

```
docker push localhost:5000/alpinegit:test
```

Nous allons maintenant, **réutiliser l'api Docker Registry pour lister les différents tags de notre image** `alpinegit` :

```
curl -X GET http://localhost:5000/v2/alpinegit/tags/list
```

Résultat :

```
{"name":"alpinegit","tags":["test","latest"]}
```

Sécurité

Le stockage dans un Registry Docker

Par défaut aucun volume n'est créé, donc si vous quitter votre conteneur registry, vous perdrez automatiquement toutes vos images hébergées dans votre registry. Il est donc très important de **rajouter un volume**. L'exemple suivant monte le répertoire hôte **data/** vers le dossier **/var/lib/registry/** du conteneur `mon-registry`.

Premièrement, on crée le dossier de stockage sur notre machine hôte :

```
mkdir data
```

Deuxièmement, on **démarre notre conteneur avec le volume adéquat** :

```
docker run -d \
    -p 5000:5000 \
    --restart=always \
    --name mon-registry \
    -v "$(pwd)"/data:/var/lib/registry \
    registry:2
```

le chiffrement

L'exécution d'un Registry accessible uniquement en local a une utilité limitée. Afin de rendre un Registry accessible aux hôtes externes, vous devez d'abord **sécuriser le Registry Docker à l'aide de TLS**.

Bien qu'il soit vivement recommandé de sécuriser votre base de registre à l'aide d'un certificat TLS émis par une autorité de certification connue, vous pouvez choisir d'utiliser des certificats auto-signé ou d'utiliser votre base de registre via une connexion HTTP non chiffrée. L'un ou l'autre de ces choix implique des compromis en matière de sécurité et des étapes de configuration supplémentaires.

Information

Le Registry prend en charge l'utilisation de Let's Encrypt afin d'obtenir automatiquement un certificat approuvé par les navigateurs.

Cependant dans notre exemple, nous allons générer des certificats auto-signés, ils ne seront pas reconnus par nos navigateurs mais au moins **la communication avec notre Registry sera chiffrée**.

Conseil

En utilisant un registre en HTTP seulement, vous exposez alors votre base de registre à des attaques triviales de type "homme au milieu" (MITM).

En premier lieu, commencez par créer un dossier nommé `certs` pour stocker nos certificats :

```
mkdir certs
```

En seconde partie, nous allons **générer nos propres certificats avec l'outil openssl** :

```
openssl req \
    -newkey rsa:4096 -nodes -sha256 -keyout certs/localhost.key \
    -x509 -days 365 -out certs/localhost.crt
```

Openssl vous demandera quelques informations, vous pouvez laisser les options par défaut en appuyant sur la touche entrée.

Pour utiliser les certificats générés, nous allons alors surcharger des variables d'environnements proposé par l'image `registry:2` et utiliser le port 443 :

```
docker run -d \
    --restart=always \
    --name mon-registry \
    -v "$(pwd)"/data:/var/lib/registry \
    -v "$(pwd)"/certs:/certs \
    -e REGISTRY_HTTP_ADDR=0.0.0.0:443 \
    -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/localhost.crt \
    -e REGISTRY_HTTP_TLS_KEY=/certs/localhost.key \
    -p 443:443 \
    registry:2
```

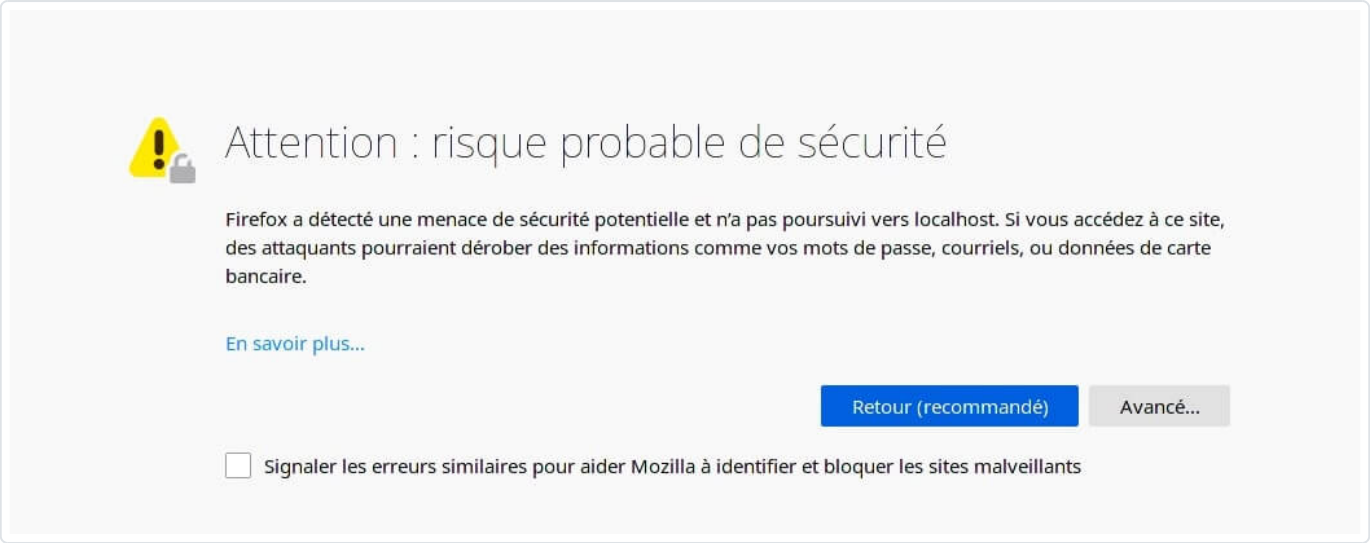
Information

La ligne `REGISTRY_HTTP_ADDR=0.0.0.0:443`, permet de rendre notre registry accessible depuis n'importe IP.

Maintenant on est capable d'envoyer nos images vers notre nouveau registry chiffrée :

```
$ docker tag alpinegit localhost:443/alpinegit
$ docker tag alpinemysql localhost:443/alpinemysql
$ docker push localhost:443/alpinegit
$ docker push localhost:443/alpinemysql
```

La prochaine étape consiste à visiter la page suivante https://localhost/v2/_catalog. Si votre navigateur vous le permet, alors il vous demandera de rajouter le certificat en question car ce dernier n'est pas reconnu par une CA. Dans mon cas j'utilise Firefox, et voici ce que j'obtiens comme résultat :



Cliquez ensuite sur le bouton "Avancé", puis sur "Accepter le risque et poursuivre" et si tout se passe comme prévu alors vous obtiendrez le résultat suivant :

Restriction d'accès

Pour rajouter une autre couche de sécurité dans le fonctionnement de vos registres sur des réseaux locaux sécurisés, vous pouvez implémenter des restrictions d'accès.

Pour le moment, nous allons **mettre en place une Authentification de base avec un utilisateur et un mot de passe globaux** à l'aide du fichier **htpasswd**.

Heureusement que notre baleine (mascotte de docker) a pensé à nous, car depuis l'image `registry`, il est possible de générer le fichier **htpasswd**.

On commence d'abord par créer un dossier **auth** pour stocker notre fichier :

```
mkdir auth
```

Par la suite on génére notre fichier **htpasswd** :

```
docker run --rm \
    --entrypoint htpasswd \
    registry:2 -Bbn testuser testpassword > auth/htpasswd
```

Démarrer ensuite votre conteneur en créant un volume qui prend en compte le fichier **htpasswd** :

```
docker run -d \
    -p 5000:5000 \
    --restart=always \
    --name mon-registry \
    -v "$(pwd)"/data:/var/lib/registry \
    -v "$(pwd)"/auth:/auth \
    -e "REGISTRY_AUTH=htpasswd" \
    -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
    -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
    -v "$(pwd)"/certs:/certs \
    -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/localhost.crt \
    -e REGISTRY_HTTP_TLS_KEY=/certs/localhost.key \
    registry:2
```

Maintenant, il faut **se connecter sur votre Docker Registry privé** :

```
docker login localhost:5000
```

```
Username: testuser
Password: testpassword
```

Si tout se déroule comme prévu, vous devriez avoir le message suivant :

```
Login Succeeded
```

Une fois authentifié, vous pouvez alors envoyer votre image dans votre Registry privé :

```
docker push localhost:5000/alpinemysql
```

Docker Compose

Normalement, si vous avez suivi ce chapitre depuis le début, vous devez vous retrouver avec l'arborescence suivante :

```
|___ auth
|
|___ htpasswd
|___ certs
|   |___ localhost.crt
|
|___ localhost.key

|___  data
```

Reprenons les fonctionnalités que nous venons de mettre en place. Actuellement nous avons :

- Un volume pour stocker nos images envoyées par l'utilisateur
- Une communication chiffrée

- Un système d'authentification basique

Nous allons reprendre toutes ces fonctionnalités et les rajouter dans un Docker Compose :

```
version: '3.7'
services:
    registry:
        restart: always
        image: registry:2
        container_name: my-web-container
        ports:
            - 5000:5000
        environment:
            REGISTRY_HTTP_TLS_CERTIFICATE: /certs/localhost.crt
            REGISTRY_HTTP_TLS_KEY: /certs/localhost.key
            REGISTRY_AUTH: htpasswd
            REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
            REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
        volumes:
            - ./data/:/var/lib/registry
            - ./certs/:/certs
            - ./auth/:/auth
```

Conclusion

Nous avons réussi à mettre en place un Registry Dokcer privé en rajoutant quelques couches de sécurité. Il est possible bien sûr de perfectionner encore plus ce registry basique en prenant par exemple en charge l'envoi de notifications Webhook en réponse aux événements se produisant dans le registre , ou en rajoutant des autorisations d'accés comme le readonly pour certains clients qui ne seront pas autorisés à écrire dans le registre etc ... Je vous conseille donc vivement de jeter un coup d'œil à la documentation officielle des Registry Docker.