Utilisation de variables d'environnement

Les variables d'environnement ENV

Dans ce Lab, nous couvrirons les variables d'environnement (ENV). Vous pouvez les utiliser dans votre Dockerfile pour définir des informations lors de la construction du conteneur, ce qui signifie que vous n'avez pas à aller mettre à jour de nombreuses commandes dans votre Dockerfile ou dans des scripts que vous exécutez sur le serveur.

Pour utiliser les ENV dans votre Dockerfile, vous pouvez utiliser l'instruction ENV. La structure de l'instruction ENV est la suivante:

ENV <key> <value>

ENV username admin

En utilisant Alpine Linux, nous ferons ce qui suit:

- Définir ARG pour la version de PHP que nous aimerions installer.
- Installer Apache2 et notre version PHP choisie.
- Configurer l'image pour qu'Apache2 démarre sans problème.
- Supprimer le fichier index.html par défaut et ajoutez un fichier index.php qui affiche les résultats de la commande phpinfo.
- Exposer le port 80 sur le conteneur.
- Définir Apache pour qu'il s'agisse du processus par défaut.

Notre Dockerfile ressemble à ceci:

```
[root@localhost ~]# mkdir apache-php && cd apache-php

[root@localhost apache-php]# vim Dockerfile
FROM alpine:latest
LABEL maintainer="Elies Jebri <elies.jebri@gmail.com>"
LABEL description="Cet exemple de Dockerfile installe  Apache & PHP."
ARG PHPVERSION
RUN apk add --update apache2 php${PHPVERSION}-apache2 php${PHPVERSION} && \
rm -rf /var/cache/apk/* && \
rm -rf /var/www/localhost/htdocs/index.html && \
echo "<?php phpinfo(); ?>" > /var/www/localhost/htdocs/index.php && \
chmod 755 /var/www/localhost/htdocs/index.php
EXPOSE 80/tcp
ENTRYPOINT ["httpd"]
CMD ["-D", "FOREGROUND"]
```

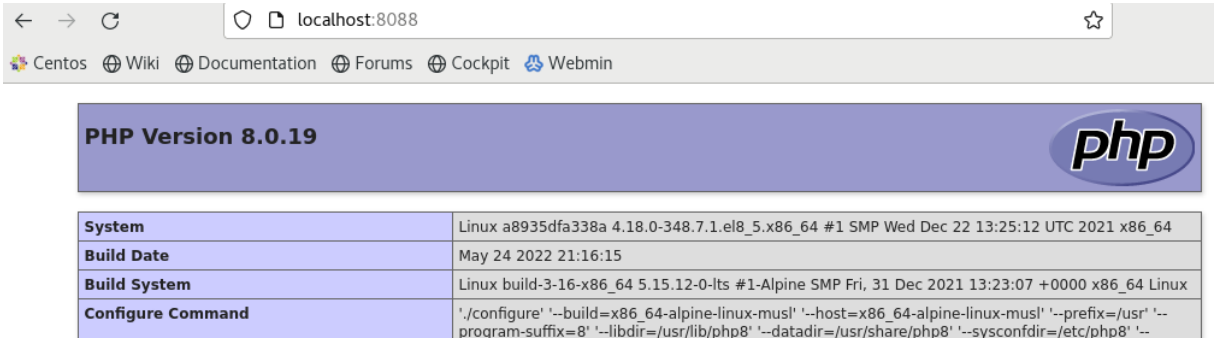
Nous pouvons construire l'image en exécutant la commande suivante:

```
[root@localhost apache-php]# docker build --tag local/apache-php:8 \
--build-arg PHPVERSION=8 .
```

Nous pouvons vérifier si tout s'est déroulé comme prévu en exécutant la commande suivante pour lancer un conteneur à l'aide de l'image :

```
[root@localhost apache-php]# docker container run -d -p 8080:80 --name
apache-php7 local/apache-php:8
```

Ouvrez un navigateur à l'URL http://localhost:8080



Terminez le conteneur :

```
docker container ls
docker kill 6dace3ccb831
```

Passage de variables avec --env

Les variables d'environnement peuvent aussi être passées aux processus à l'intérieur des conteneurs lors de leur exécution (docker run) grace à l'option --env (-e) :

```
[root@localhost ~]# docker run --env VARIABLE1=foobar alpine env

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=50abc9a26404
VARIABLE1=foobar
HOME=/root

[root@localhost ~]# export VARIABLE2=foobar2
[root@localhost ~]# docker run --env VARIABLE2 alpine env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=d850a388221c
VARIABLE2=foobar2
HOME=/root

[root@localhost ~]# VARIABLE3=foobar3 docker run --env VARIABLE3 alpine env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=043441d30ffc
VARIABLE3=foobar3
HOME=/root
```

Une autre solution consiste à utiliser un fichier texte pour stocker nos variables, en utilisant le format standard clé=valeur .

Définissons quelques variables dans un fichier que nous appellerons my-env.txt :

```
$ echo VARIABLE1=foobar1 > my-env.txt
$ echo VARIABLE2=foobar2 >> my-env.txt
$ echo VARIABLE3=foobar3 >> my-env.txt
```

Maintenant, injectons ce fichier dans notre conteneur Docker:

```
$ docker run --env-file my-env.txt alpine env
```

Essayons maintenant d'utiliser les variables d'environnement pour personnaliser le message affiché par PHP au lieu de la page d'info. Nous utiliserons l'image construite précédemment comme base de départ.

Notre Dockerfile ressemble à ceci:

```
[root@localhost apache-php]# vim Dockerfile-message
FROM local/apache-php:7
LABEL maintainer="Elies Jebri <elies.jebri@gmail.com>"
LABEL description="Cet exemple de Dockerfile instalte les variables
d'environnement."
COPY message-entrypoint.sh /bin
RUN     chmod 555 /bin/message-entrypoint.sh && \
rm -rf /var/www/localhost/htdocs/index.html
EXPOSE 80/tcp
ENTRYPOINT ["/bin/message-entrypoint.sh"]
```

Nous allons ensuite définir un script d'entrée lors de l'exécution du conteneur qui se chargera de récupérer le message et le placer dans index.php puis de lancer apache :

```
[root@localhost apache-php]# vim message-entrypoint.sh
#! /bin/sh
echo "<?php echo '<p>$MESSAGE</p>'; ?>" >
/var/www/localhost/htdocs/index.php
chmod 755 /var/www/localhost/htdocs/index.php
httpd -D FOREGROUND
```

Créez maintenant cette image:

```
[root@localhost apache-php]# docker build --file Dockerfile-message --tag
local/php-message .
```

Exécutez le conteneur:

```
[root@localhost apache-php]# docker container run -e MESSAGE=Hello -d -p
8080:80 --name message-hello local/php-message

[root@localhost apache-php]# curl http://localhost:8080
<p>Hello</p>
```

Gérer un conteneur MYSQL

Dans cet exercice, vous allez créer et gérer un conteneur de base de données MySQL.

Ouvrez un terminal à partir de la VM du poste de travail et exécutez la commande suivante :

```
[root@localhost ~]# docker run --name mysql-db mysql:5.7
Unable to find image 'mysql:latest' locally
Trying to pull repository docker.io/library/mysql ...
```

```
latest: Pulling from docker.io/library/mysql
68ced04f60ab: Already exists
f9748e016a5c: Pull complete

...
2020-03-18 15:31:46+00:00 [ERROR] [Entrypoint]: Database is uninitialized
and password option is not specified
You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD
and MYSQL_RANDOM_ROOT_PASSWORD
```

Cette commande télécharge l'image du conteneur de base de données MySQL et essaie de le démarrer, mais échoues. La raison est que l'image nécessite quelques variables d'environnement à fournir. La commande suivante le confirme :

```
[root@localhost ~]# docker logs mysql-db
[root@localhost ~]# docker container rm mysql-db
```

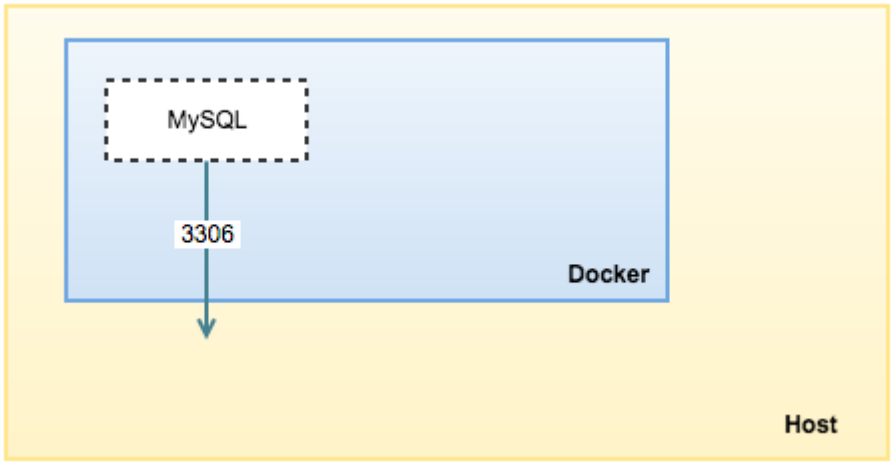
Redémarrez le conteneur en fournissant les variables requises. Donnez-lui un nom de mysql-db. Spécifiez chaque variable à l'aide du paramètre -e.

```
[root@localhost ~]# docker run --name mysql-db -d -e MYSQL_USER=user1 -e
MYSQL_PASSWORD=password -e MYSQL_DATABASE=items -e
MYSQL_ROOT_PASSWORD=password mysql:5.7

[root@localhost ~]# docker container ls
CONTAINER ID   IMAGE      COMMAND                  CREATED
ca4b1662d7ff   mysql:5.7  "docker-entrypoint..."   About a minute ago

STATUS              PORTS                 NAMES
Up About a minute   3306/tcp, 33060/tcp   mysql-db
```

Inspectez les métadonnées du conteneur pour obtenir le script d'initialisation et l'adresse IP de la base de données MySQL:

```
[root@localhost ~]# docker container inspect -f '{{ .Path }} {{
.NetworkSettings.IPAddress }}' mysql-db
docker-entrypoint.sh 172.17.0.2
```

Connectez-vous à la base de données MySQL depuis l'hôte:



```
[root@localhost ~]# yum install mysql
[root@localhost ~]# mysql -u user1 -h 172.17.0.2 -p items
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.29 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MySQL [items]>
```

Vous êtes connecté à la base de données d'articles. Créez une nouvelle table:

```
MySQL [items]> CREATE TABLE Projects (id int(11) NOT NULL, \
    -> name varchar(255) DEFAULT NULL, code varchar(255) DEFAULT NULL, \
    -> PRIMARY KEY (id));
Query OK, 0 rows affected (0.01 sec)
```

Insérez une ligne dans la table en exécutant la commande suivante:

```
MySQL [items]> insert into Projects (id, name, code) values
(1,'DevOps','DO701');
Query OK, 1 row affected (0.01 sec)

MySQL [items]> select * from items.Projects;
+----+--------+-------+
| id | name   | code  |
+----+--------+-------+
|  1 | DevOps | DO701 |
+----+--------+-------+
1 row in set (0.00 sec)

MySQL [items]> exit
Bye
```

Avant de terminer, voyons où sont stockées les données par notre conteneur :

```
[root@localhost ~]# docker inspect -f '{{ range .Mounts }}{{ .Source }}<-:->{{
.Destination }} {{ end }}' mysql-db
/var/lib/docker/volumes/8f618aad…0a168cb00696a/_data<-:->/var/lib/mysql
```

"/var/lib/docker/volumes/8f618aadec8f2f764fc02addce57ca24075265cae13c380640c0a168cb00696a/_data" représente le volume associé au conteneur sur l'hôte alors que *"/var/lib/mysql"* est le dossier par défaut du processus MySQL qui s'exécute dans le conteneur.

```
[root@localhost ~]# mkdir mysql-db
[root@localhost ~]# cd mysql-db
[root@localhost mysql-db]# cp -Rp
/var/lib/docker/volumes/8f618aadec8f2f764fc02addce57ca24075265cae13c380640c
0a168cb00696a/_data .
[root@localhost ~]# docker container stop mysql-db
[root@localhost ~]# docker container rm mysql-db
```