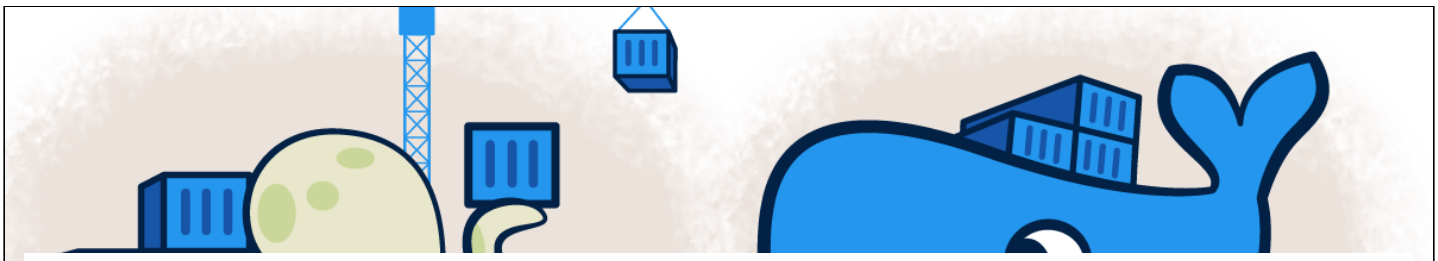


Ouvrir une connexion SSH sur un container

Cloud, Docker

Logiciel Compétences : 3

Créer des containers avec Docker facilite l'implémentation d'une application dans un environnement comme le cloud Azure. Pour accéder au container, la configuration d'une connexion SSH s'impose.

Publié mardi 28 juin 2022, 14h40
Modifié lundi 26 août 2024, 10h04

By Olivier Paudex

Introduction

L'utilisation d'un container Docker pour créer une application n'est plus à défendre, tant ceux-ci ont facilité la mise en place et la distribution des plateformes web. Et le Cloud Azure n'est pas en reste, puisqu'il permet de déployer des containers directement dans son **"App Services"**. Mais quant est-il de la connexion à ceux-ci ? Peut-on au moins se connecter sur un container et/ou sur l'application avec le protocole SSH. Et bien la réponse est oui, mais avec quelques limitations.

Le challenge

Comment créer une application web avec Docker, tout en gardant la possibilité d'accéder au serveur avec le protocole SSH.

Et dans un deuxième temps, déployer cette même application sur le cloud Azure tout en gardant les mêmes options.

Création du container web

La première tâche à réaliser est de créer le serveur web qui contient l'application. Celle-ci reste très simple pour la démonstration.

- Créez un dossier **"web_server"**.
- Créez un dossier source à l'intérieur de celui-ci.
- Créer un fichier **"dockerfile"**.
- Dans le dossier source, créez un fichier **"index.php"**.

Le fichier **"index.php"** ne contient que quelques lignes, juste pour indiquer que l'application web fonctionne.

La structure de l'application et le fichier "index.php"

Le fichier **"dockerfile"** n'est guère plus compliqué. Il va effectuer les opérations suivantes :

- Installer un serveur **"Apache"** en version 7.4.
- Copier le contenu du dossier source, à savoir le fichier **"index.php"** dans la structure web du serveur (/var/www/html).

- Ouvrir le port 80 pour permettre le trafic web.

```
FROM php:7.4-apache
# Install Apache2 server and update
RUN apt-get update -y
# Apache2
COPY source /var/www/html
WORKDIR /var/www/html
# Open port (metadata only)EXPOSE 80/tcp
```

Créez l'image du container avec la commande ci-dessous.

```
docker build -t server_web .
```

Voilà, le container web est terminé. On peut le tester en local avec la commande ci-dessous :

```
docker run -d -p 80:80 --name server_web server_web
```

puis ouvrir un navigateur et entrez l'adresse URL **"localhost"**. L'expression **"Serveur WEB"** devrait s'afficher.

Création du container SSH

Deuxième étape, la création du container SSH. En effet, avec les containers, il est recommandé de séparer les services. Donc, un container pour l'application web et un autre pour le service SSH.

- Créez un dossier **"ssh_server"** à côté du dossier **"web_server"**.
- Créez un fichier **"dockerfile"** à l'intérieur de celui-ci.
- Copiez le contenu du **"dockerfile"** ci-dessous et sauvegardez.

Voici les tâches effectuées :

- Installation d'un serveur Debian et mise à jour.
- Installation d'un serveur SSH.
- Installation de différents outils (ping, nslookup, telnet, vim, azure-cli).
- Paramètres du serveur SSH (Port, Authorisation de connexion, méthode d'encryption, ...).

- Modification du mot de passe du compte **“root”**.
- Génération des clés du serveur
- Ouverture du port 2222 sur le serveur.
- Démarrage du service SSH.

```
FROM debian:latest
# Install Debian server with some useful tools
RUN apt-get update -y
RUN apt-get install openssh-server -y
RUN apt-get install iputils-ping net-tools -y
RUN apt-get install telnet -y
RUN apt-get install dnsutils -y
RUN apt-get install vim -y
RUN apt-get install git -y
RUN apt-get install zip -y
RUN apt-get install unzip -y
RUN apt-get install azure-cli -y
# Create OpenSSH settings file
RUN echo "PasswordAuthentication    yes" > /etc/ssh/sshd_config
RUN echo "PermitEmptyPasswords      no" >> /etc/ssh/sshd_config
RUN echo "PermitRootLogin           yes" >> /etc/ssh/sshd_config
RUN echo "Port                      2222" >> /etc/ssh/sshd_config
RUN echo "ListenAddress             0.0.0.0" >> /etc/ssh/sshd_config
RUN echo "LoginGraceTime            180" >> /etc/ssh/sshd_config
RUN echo "X11Forwarding             yes" >> /etc/ssh/sshd_config
RUN echo "Ciphers                   aes128-cbc,3des-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256
RUN echo "MACs                      hmac-sha1,hmac-sha1-96" >> /etc/ssh/sshd_config
RUN echo "StrictModes               yes" >> /etc/ssh/sshd_config
RUN echo "SyslogFacility            DAEMON" >> /etc/ssh/sshd_config
RUN echo "Subsystem                 sftp internal-sftp" >> /etc/ssh/sshd_config
# Set root password
RUN echo "root:Docker!" | chpasswd
# Generate some keys
RUN ssh-keygen -A
# Open port (metadata only)
EXPOSE 2222/tcp
# Start services
RUN service ssh startCMD ["/usr/sbin/sshd","-D"]
```

Créez l’image du container avec la commande ci-dessous.

```
docker build -t server_ssh .
```

Voilà, le container web est terminé. On peut le tester en local avec la commande ci-dessous :

```
docker run -d -p 22:2222 --name server_ssh server_ssh
```

puis ouvrez une fenêtre de commande et entrez la commande **“ssh root@localhost”**. Acceptez la clé proposé puis entrez le mot de passe qui est **“Docker!”**. La connexion SSH devrait s’établir.

```
C:\Users\olivi>ssh root@localhost
 The authenticity of host 'localhost (::1)' can't be established.
 ECDSA key fingerprint is SHA256:qEQr3T+3R8UNg2AC5/r/tnloVb20A5OwgUA5aYQffa8.
 Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
 Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
 root@localhost's password:
 The programs included with the Debian GNU/Linux system are free software;
 the exact distribution terms for each program are described in the
 individual files in /usr/share/doc/*/copyright.
 Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
 permitted by applicable law. root@server_ssh:~#
```

Utilisation de l'orchestrateur

Pour créer l'illusion que l'on va regrouper nos deux containers en un seul, on va utiliser l'orchestrateur de Docker, le **"docker-compose"**.

- Créez un fichier **"docker-compose.yml"** au même niveau que les deux dossiers **"web_server"** et **"ssh_server"**.
- Copiez le contenu du fichier ci-dessous.

Voici les tâches effectuées par l'orchestrateur :

- Création d'un serveur Web
- Le nom du container est **"server_web"**.
- Le nom complet (FQDN) du serveur est **"server_web.fuyens.ch"**.
- L'image docker utilisé est **"server_web"**.
- Le port local est le 80. Le port du serveur est le 80.

Idem pour le serveur SSH, à l'exception du port qui devient le 22 en local et le 2222 sur le serveur.

```
version: '3'
services:
  web_server:
    container_name: server_web
    hostname: server_web.fuyens.ch
    image: server_web
    ports:
      - 80:80
  ssh_server:
    container_name: server_ssh
    hostname: server_ssh.fuyens.ch
    image: server_ssh
    ports:       - 22:2222
```

Créez l'image des deux containers avec la commande ci-dessous :

```
docker-compose up
```

Le log de la commande **"docker-compose"** qui affiche bien que les deux serveurs sont créées.

```
PS C:\Users\olivi\Desktop\projet_ssh> docker-compose up
 [+] Running 2/2
 Container server_ssh  Created
 Container server_web  Created
 Attaching to server_ssh, server_web
 server_web  | [Tue Jun 28 09:47:35.579871 2022] [mpm_prefork:notice] [pid 1] AH00163: Apache/2
 server_web  | [Tue Jun 28 09:47:35.580013 2022] [core:notice] [pid 1] AH00094: Command line:
```

Voilà, l'orchestrateur a créé les deux containers web et ssh séparément. Comme dans les exemples ci-dessus, on peut se connecter sur le serveur web et sur le serveur ssh.

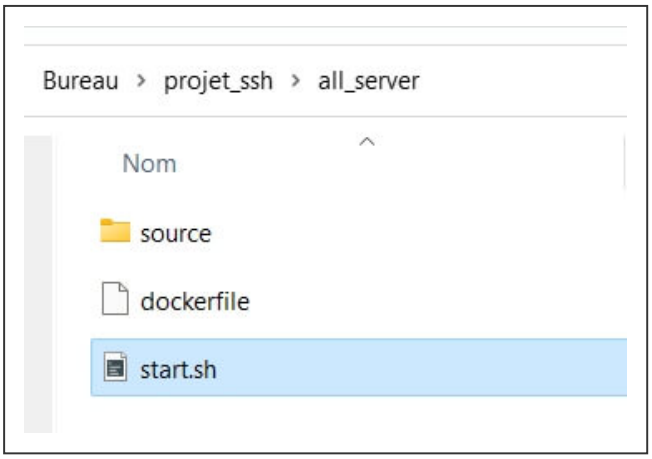
Création d'une application sur Azure

Il ne reste plus qu'à essayer de pousser les containers vers Azure et de retenter l'expérience.

> La surprise est de taille, mais Microsoft Azure, qui accepte pourtant de créer des applications avec de multiples containers, ne permet pas d'utiliser ceux-ci pour séparer le serveur SSH du serveur Web. Il va falloir **"Assembler les deux containers en un seul"** pour y remédier. Aux dernières nouvelles, Microsoft devrait corriger ceci, pour que l'utilisation d'un seul container soit possible.

Création d'un container offrant les deux services

- Créez un nouveau dossier **"all_server"**.
- Copiez le dossier source contenant le fichier **"index.php"**.
- Créez un nouveau fichier **"dockerfile"** et copiez le code ci-dessous.
- Créez un fichier **"start.sh"** qui va permettre de démarrer les deux services du container.




```
#!/bin/sh
service ssh start
/usr/sbin/sshd
/usr/sbin/apache2ctl -D FOREGROUND
```

La structure du nouveau container et le script de démarrage des services

```
FROM php:7.4-apache
# Install Debian server with some useful tools
RUN apt-get update -y
RUN apt-get install openssh-server -y
RUN apt-get install iputils-ping net-tools -y
RUN apt-get install telnet -y
RUN apt-get install dnsutils -y
RUN apt-get install vim -y
RUN apt-get install git -y
RUN apt-get install zip -y
RUN apt-get install unzip -y
RUN apt-get install azure-cli -y
# Create OpenSSH settings file
RUN echo "PasswordAuthentication    yes" > /etc/ssh/sshd_config
RUN echo "PermitEmptyPasswords      no" >> /etc/ssh/sshd_config
RUN echo "PermitRootLogin           yes" >> /etc/ssh/sshd_config
RUN echo "Port                      2222" >> /etc/ssh/sshd_config
RUN echo "ListenAddress             0.0.0.0" >> /etc/ssh/sshd_config
RUN echo "LoginGraceTime            180" >> /etc/ssh/sshd_config
RUN echo "X11Forwarding             yes" >> /etc/ssh/sshd_config
RUN echo "Ciphers                   aes128-cbc,3des-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes25
RUN echo "MACs                      hmac-sha1,hmac-sha1-96" >> /etc/ssh/sshd_config
RUN echo "StrictModes               yes" >> /etc/ssh/sshd_config
RUN echo "SyslogFacility            DAEMON" >> /etc/ssh/sshd_config
RUN echo "Subsystem                 sftp internal-sftp" >> /etc/ssh/sshd_config
# Set root password
RUN echo "root:Docker!" | chpasswd
# Generate some keys
RUN ssh-keygen -A
# Apache2
COPY source /var/www/html
WORKDIR /var/www/html
```

```
# Open port (metadata only)
EXPOSE 2222/tcp
EXPOSE 80/tcp
# Start services
COPY start.sh /start.sh
RUN chmod 777 /start.shCMD /start.sh
```

Créez l'image du container avec la commande ci-dessous.

```
docker build -t server_all .
```

Voilà, le nouveau container offrant les deux services est disponible. Il ne reste plus qu'à le pousser vers le Cloud Azure et créer une application.

Le registre de containers du Cloud Azure

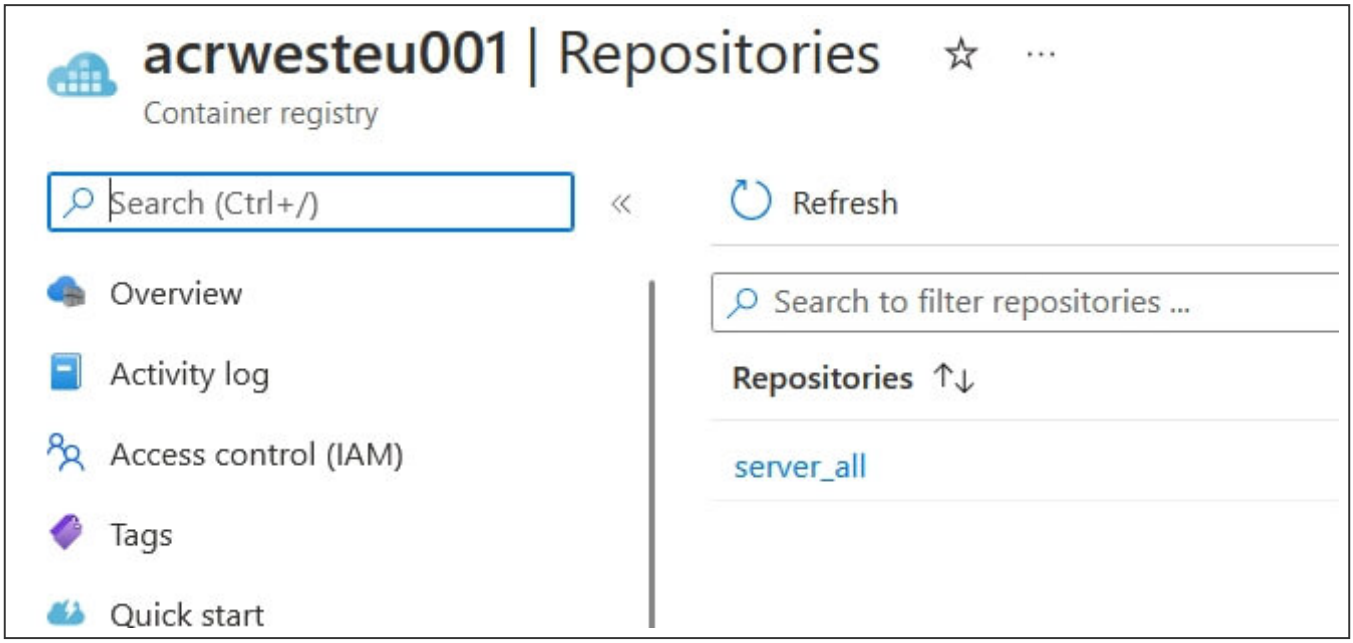
- Ouvrez la console Azure et connectez-vous.
- Créez un **"Azure Container Registry"** et nommez-le. Dans l'exemple ci-dessous, il se nomme **"acrwesteu001"**.
- Depuis VSCode, connectez-vous au registre.

```
az acr login --name acrwesteu001
```

Créez le tag du container **"server_all"**, puis poussez-le vers le registre du Cloud Azure.

```
docker tag server_web acrwesteu001.azurecr.io/server_all:1.0
docker push acrwesteu001.azurecr.io/server_all:1.0
```

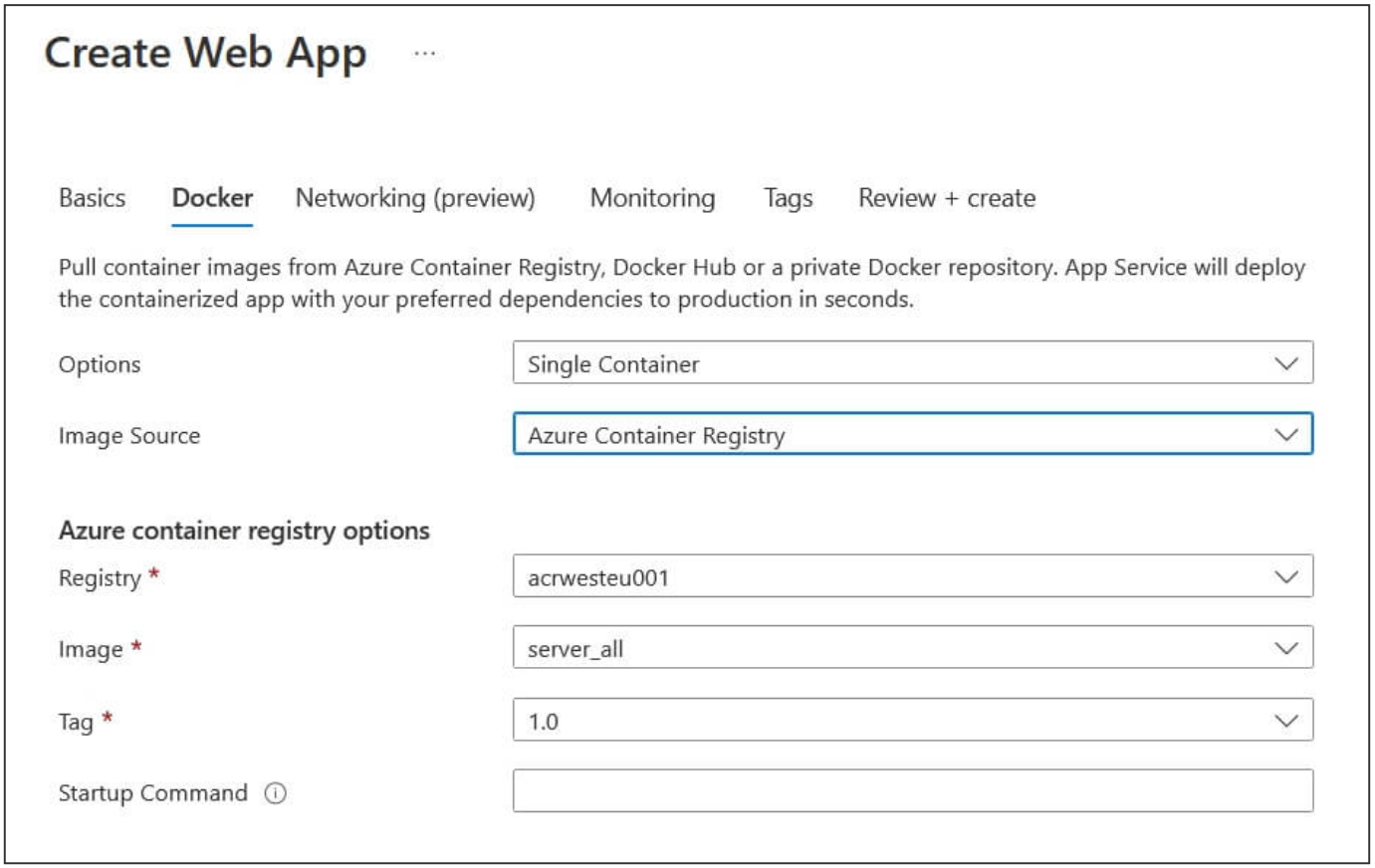
Vérifiez que le container se trouve bien dans le registre.



Le registre de containers du Cloud Azure

Le service applications du Cloud Azure

- Créez une application avec **"App Services"**.
- Dans la section **"Docker"**, sélectionnez l'option **"Single Container"**.
- Dans le fichier de configuration, sélectionnez tour à tour le fichier image **"server_all"**.
- Dans les deux cas, sélectionnez la version **"1.0"**.
- Sauvegardez.



La création d'une application à l'aide d'un container

Connectez-vous au serveur web avec l'URL :

```
https://[nom de l'application].azurewebsites.net
```

> La première connection prend du temps. Pas de panique !

Si tout fonctionne bien, le message **"Serveur WEB"** devrait s'afficher.

Le service SSH

Pour se connecter en SSH depuis un terminal, c'est un peu plus compliqué.

Retournez dans VSCode et entrez la commande Azure ci-dessous

```
az webapp create-remote-connection
  -- subscription [id]
  -- resource-group [name of the resource-group]  -- name [name of the application]
```

Le terminal devrait retourner un port de connexion

```
Verifying if app is running….
 App is running. Trying to establish tunnel connection…
 Opening tunnel on port: 50818
 SSH is available { username: root, password: Docker! } Ctrl + C to close
```

Il est maintenant possible de se connecter en utilisant la commande ci-dessous :

```
ssh root@localhost -p 50818
```

```
C:\Users\olivi>ssh root@localhost -p 50818
 root@localhost's password:
 The programs included with the Debian GNU/Linux system are free software;
 the exact distribution terms for each program are described in the
 individual files in /usr/share/doc/*/copyright.
 Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
 permitted by applicable law. root@c1a48fdd6ef8:~#
```

Conclusion

Voilà un petit exercice auquel vous n'échapperez pas si vous rencontrez des problèmes avec votre application. Alors, plutôt que de tatonner à la recherche de l'erreur, de repousser 50x le container vers le cloud Azure afin de trouver le problème, une petite connexion SSH est bien pratique.

Il est a noté également qu'il est possible de se connecter en SSH directement depuis la console Azure en sélectionnant **"l'onglet SSH"** depuis l'application.