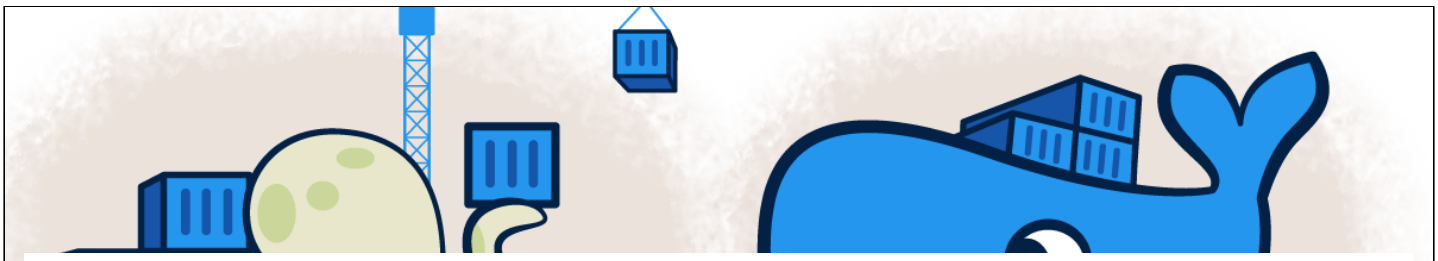


Open an SSH connection on a container

Cloud, Docker

Software Skills : 3

Creating containers with Docker makes it easier to implement an application in an environment like the Azure cloud. To access the container, an SSH connection must be configured.

Published Tuesday June 28th 2022, 14:40
Modified Monday August 26th 2024, 10:04

By Olivier Paudex

Introduction

The use of a Docker container to create an application is no longer defensible, so much so have these made it easier to build and distribute web platforms. And the Azure Cloud is not to be outdone, since it allows you to deploy containers directly in its **"App Services"**. But what about connecting to them? Can we at least connect to a container and/or the application with the SSH protocol. Well the answer is yes, but with some limitations.

The Challenge

How to create a web application with Docker, while keeping the ability to access the server with SSH protocol.

And in a second step, deploy this same application on the Azure cloud while keeping the same options.

Creating the web container

The first task to perform is to create the web server that contains the application. This one is kept very simple for the demonstration.

- Create a **"web_server"** folder.
- Create a source folder inside it.
- Create a **"dockerfile"**.
- In the source folder, create a **"index.php"** file.

The **"index.php"** file contains only a few lines, just to indicate that the web application is working.

The application structure and the "index.php"

The **"dockerfile"** file is not much more complicated. It will perform the following operations:

- Install a **"Apache"** server in version 7.4.
- Copy the contents of the source folder, namely the **"index.php"** file into the server's web structure (/var/www/html).
- Open port 80 to allow web traffic.

```
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)EXHIBIT 80/tcp
```

Create the container image with the command below.

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

Here we go, the web container is finished. We can test it locally with the command below:

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

then open a browser and enter the URL **"localhost"**. The sentence **"WEB server"** should appear.

Creating the SSH container

Second step, the creation of the SSH container. Indeed, with containers, it is recommended to separate the services. So, one container for the web application and another for the SSH service.

- Create a **"ssh_server"** folder next to the **"web_server"** folder.
- Create a **"dockerfile"** file inside it.
- Copy the contents of the **"dockerfile"** below and save.

Here are the tasks performed:

- Installation of a Debian server and update.
- Installation of an SSH server.
- Installation of different tools (ping, nslookup, telnet, vim, azure-cli).
- SSH server settings (Port, Connection authorization, encryption method, . ..).
- Changing the password for the **"root"**account.
- Generating the server keys
- Opening port 2222 on the server.
- Starting the SSH service.

```
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-ctr" >> /etc/ssh/sshd_config
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)
EXHIBIT 2222/tcp
# Start services
RUN service ssh startCMD ["/usr/sbin/sshd","-D"]
```

Create the container image with the command below.

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

Here we go, the web container is finished. We can test it locally with the command below:

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

then open a command window and enter the command **“ssh root@localhost”**. Accept the proposed key and then enter the password which is **“Docker!”**. The SSH connection should establish.

```
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:~#
```

Orchestrator usage

To create the illusion that we're going to combine our two containers into one, we'll use Docker's orchestrator, the **"docker-compose"**.

- Create a **"docker-compose.yml"** file at the same level as the two folders **"web_server"** and **"ssh_server"**.
- Copy the contents of the file below.

Here are the tasks performed by the orchestrator:

- Creating a web server
- The container name is **"server_web"**.
- The full name (FQDN) of the server is **"server_web.fuyens.ch"**.
- The docker image used is **"server_web"**.
- The local port is 80. The server port is 80.

Idem for the SSH server, except the port becomes 22 locally and 2222 on the server.

```
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
```

Create the image of the two containers with the command below:

```
docker-compose up
```

The log from the **"docker-compose"** command that correctly displays that both servers are created.

```
PS C:\olivi\Desktop\project_ssh> docker-compose up
 [+] Running 2/2
 Container server_ssh Created 0.1s
 Container server_web Created 0.2s
```

```
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: '
```

Here we go, the orchestrator has created the two containers web and ssh separately. As in the examples above, we can connect to the web server and the ssh server.

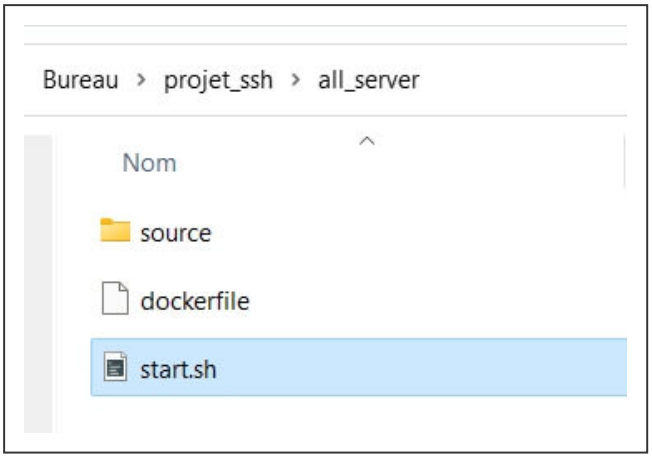
Creating an application on Azure

Now all that's left is to try pushing containers to Azure and try again.

> The surprise is that Microsoft Azure, which nevertheless accepts to create applications with multiple containers, does not allow to use these to separate the SSH server from the web server. You'll have to **"Assemble the two containers into one"**to fix this. Last I heard, Microsoft should fix this, so that using a single container is possible.

Creating a container offering both services

- Create a new **"all_server"** folder.
- Copy the source folder containing the **"index.php"** file.
- Create a new **"dockerfile"** file and copy the code below.
- Create a **"start.sh"** file that will start the two container services.




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

The structure of the new container and the script to start the 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,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
# Apache2
COPY source /var/www/html
WORKDIR /var/www/html
# Open port (metadata only)
EXHIBIT 2222/tcp
EXHIBIT 80/tcp
# Start services
COPY start.sh /start.sh
RUN chmod 777 /start.shCMD /start.sh
```

Create the container image with the command below.

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

Here we go, the new container offering both services is available. All that's left is to push it to the Azure Cloud and create an application.

The Azure Cloud Container Registry

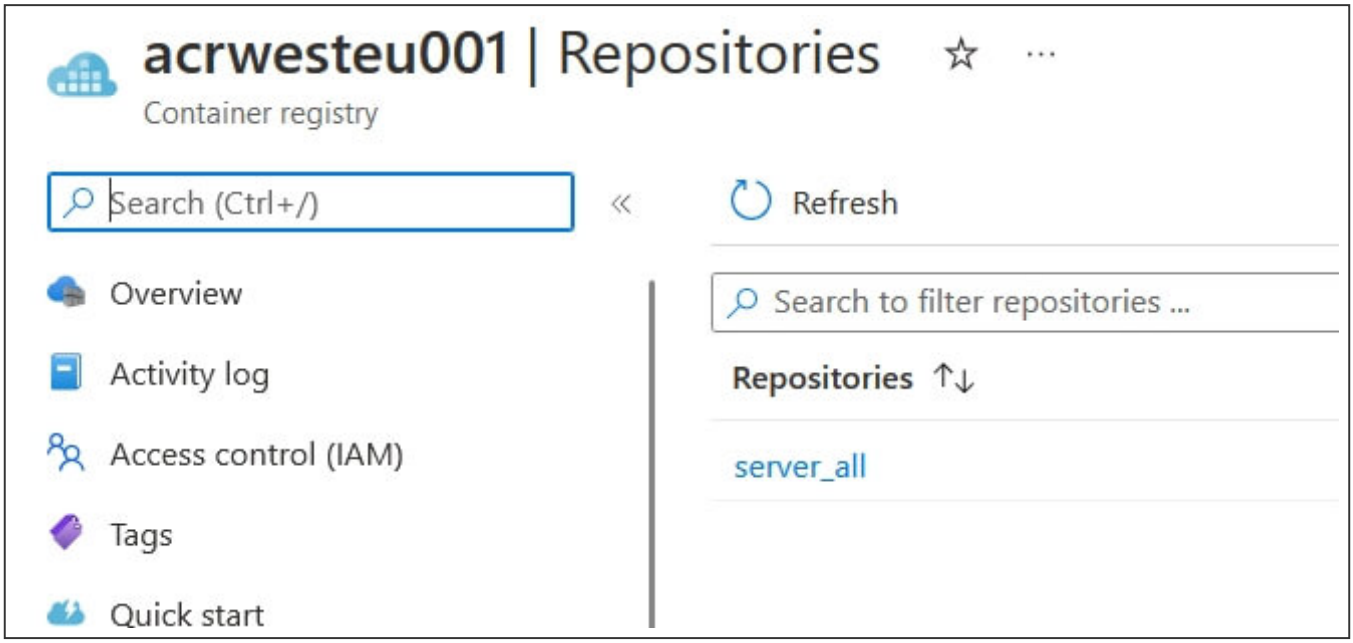
- Open the Azure console and log in.
- Create an **"Azure Container Registry"** and name it. In the example below, it is called **"acrwesteu001"**.
- From VSCode, connect to the registry.

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

Create the **"server_all"** container tag, then push it to the Azure Cloud registry.

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

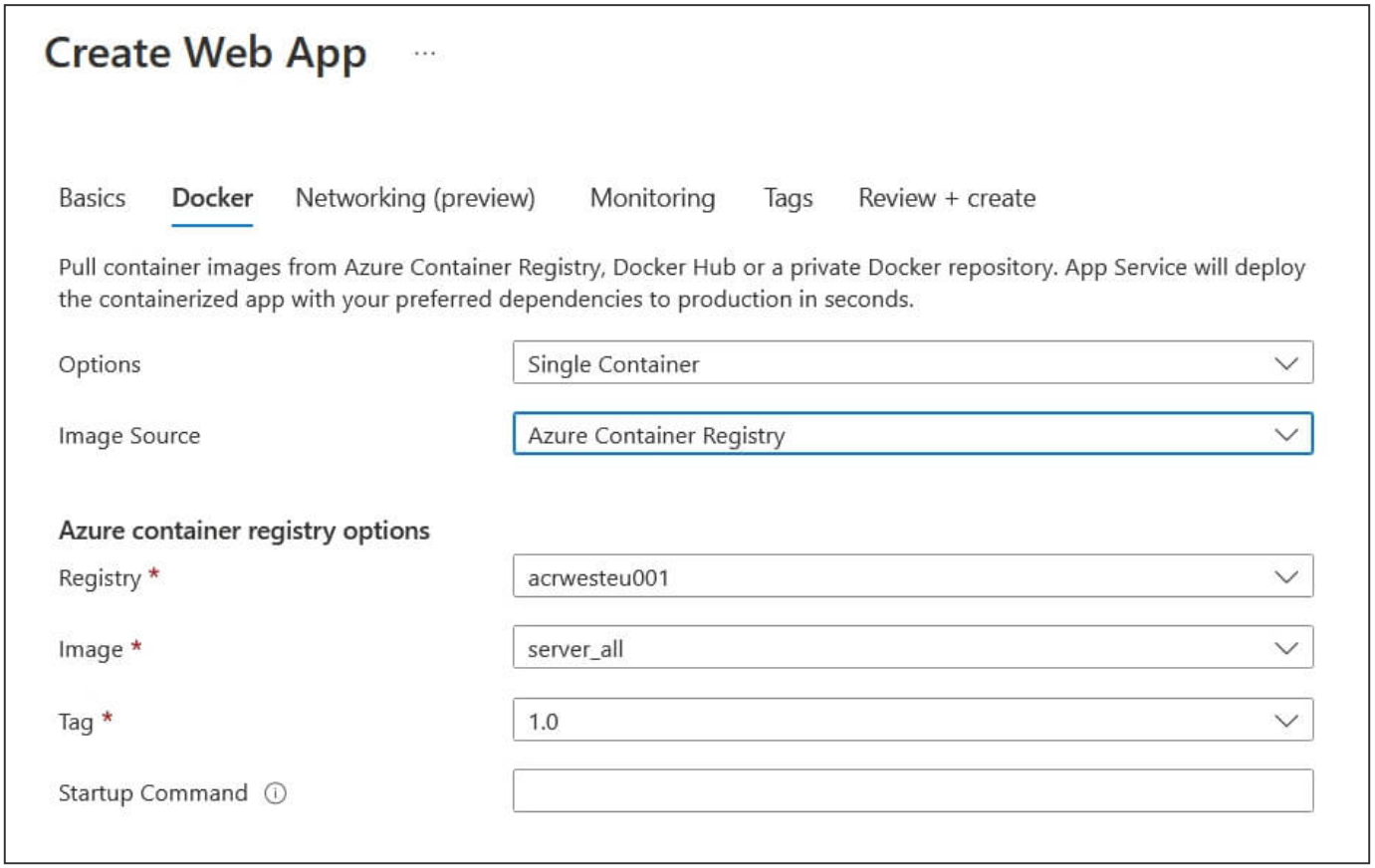
Check that the container is in the registry.



The Azure Cloud Container Registry

The Azure Cloud Applications Service

- Create an application with **"App Services"**.
- In the **"Docker"** section, select the **"Single Container"** option.
- In the configuration file, select the **"server_all"** image file in turn.
- In both cases, select version **"1.0"**.
- Save.



Creating an application using a container

Connect to the web server with the URL:

```
https://[application name].azurewebsites.net
```

> The first connection takes a while. Don't panic!"

If everything works well, the **"WEB Server"** message should be displayed.

The SSH service

To connect to SSH from a terminal, it's a little more complicated.

Go back into VSCode and enter the Azure command below

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

The terminal should return a connection port

```
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
```

It is now possible to connect using the command below:

```
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

Here's a little exercise you won't be able to escape if you run into problems with your application. So rather than tattooing around looking for the error, pushing the container 50x to the Azure cloud to find the problem, a little SSH connection comes in handy.

It's also worth noting that it's possible to connect via SSH directly from the Azure console by selecting**"the SSH tab"** from the application.