Configuration d'une petite image de Docker Linux alpine sur IOx

Contenu

Introduction

Ce document décrit le processus de configuration afin de créer, déployer et gérer des applications basées sur Docker sur des périphériques compatibles IOx Cisco.

Conditions préalables

Conditions requises

Aucune spécification déterminée n'est requise pour ce document.

Components Used

Les informations contenues dans ce document sont basées sur les versions de matériel et de logiciel suivantes :

- Périphérique compatible IOx configuré pour IOx :
  Adresse IP configuréeSystème d'exploitation invité (GOS) et Cisco Application Framework (CAF) exécutantTraduction d'adresses de réseau (NAT) configurée pour l'accès à CAF (port 8443)NAT configuré pour l'accès au shell GOS (port 2222)
- Hôte Linux (une installation minimale de CentOS 7 est utilisée pour cet article)
- Fichiers d'installation du client IOx téléchargeables à l'adresse :
  https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

Informations générales

IOx peut héberger différents types de paquets principalement Java, Python, LXC, Virtual Machine

(VM) etc, et il peut également exécuter des conteneurs Docker. Cisco propose une image de base et un référentiel de concentrateurs Docker complet : https://devhub.cisco.com/artifactory/webapp/#/artifacts/browse/tree/General/iox-docker qui peut être utilisé pour construire des conteneurs Docker.

Voici un guide pas à pas sur la façon de construire un conteneur Docker simple avec l'utilisation d'Alpine Linux. Alpine Linux est une petite image Linux (environ 5 Mo), qui est souvent utilisée comme base pour les conteneurs Docker. Dans cet article, vous démarrez à partir d'un périphérique IOx configuré, d'une machine Linux CentOS 7 vide et vous construisez un petit serveur Web Python, le paquetez dans un conteneur Docker et déployez-le sur un périphérique IOx.

Configuration

1. Installer et préparer le client IOx sur l'hôte Linux.

Le client IOx est l'outil qui peut empaqueter les applications et communiquer avec le périphérique compatible IOx pour gérer les applications IOx.

Après avoir téléchargé le package d'installation ioxclient, il peut être installé comme suit :

```
[jedepuyd@db ~]$ ll ioxclient_1.3.0.0_linux_amd64.tar.gz
-rw-r--r--. 1 jedepuyd jedepuyd 4668259 Jun 22 09:19 ioxclient_1.3.0.0_linux_amd64.tar.gz

[jedepuyd@db ~]$ tar -xvzf ioxclient_1.3.0.0_linux_amd64.tar.gz
ioxclient_1.3.0.0_linux_amd64/ioxclient
ioxclient_1.3.0.0_linux_amd64/README.md

[jedepuyd@db ~]$ ./ioxclient_1.3.0.0_linux_amd64/ioxclient --version
Config file not found : /home/jedepuyd/.ioxclientcfg.yaml
Creating one time configuration..
Your / your organization's name : Cisco
Your / your organization's URL : www.cisco.com
Your IOx platform's IP address[127.0.0.1] : 10.48.43.197
Your IOx platform's port number[8443] :
Authorized user name[root] : admin
Password for admin :
Local repository path on IOx platform[/software/downloads]:
URL Scheme (http/https) [https]:
API Prefix[/iox/api/v2/hosting/]:
Your IOx platform's SSH Port[2222]:
Activating Profile default
Saving current configuration
ioxclient version 1.3.0.0

[jedepuyd@db ~]$ ./ioxclient_1.3.0.0_linux_amd64/ioxclient --version
ioxclient version 1.3.0.0
```

Comme vous pouvez le constater, lors du premier lancement du client IOx, un profil peut être généré pour le périphérique IOx que vous pouvez gérer avec le client IOx. Si vous souhaitez effectuer cette opération ultérieurement ou si vous voulez ajouter/modifier les paramètres, vous pouvez exécuter cette commande ultérieurement : **ioxclient profils create**

2. Installez et préparez Docker sur l'hôte Linux.

Docker est utilisé pour construire un conteneur et tester l'exécution de notre exemple

d'application.

Les étapes d'installation de Docker dépendent en grande partie du système d'exploitation Linux sur lequel vous l'installez. Pour cet article, vous pouvez utiliser CentOS 7. Pour obtenir des instructions d'installation pour différentes distributions, reportez-vous à : https://docs.docker.com/engine/installation/.

Configuration requise pour l'installation :

```
[jedepuyd@db ~]$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
...
Complete!
```

Ajoutez le repo Docker :

```
[jedepuyd@db ~]$ sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
Loaded plugins: fastestmirror
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to
/etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
```

Installer Docker (acceptez la vérification de la clé GPG lors de l'installation) :

```
[jedepuyd@db ~]$ sudo yum install docker-ce
...
Complete!
```

Démarrer Docker :

```
[jedepuyd@db ~]$ sudo systemctl start docker

[jedepuyd@db iox_docker_pythonweb]$ vi Dockerfile
[jedepuyd@db iox_docker_pythonweb]$ cat Dockerfile
FROM alpine:3.3

RUN apk add --no-cache python
COPY webserver.py /webserver.py
```

Afin de pouvoir accéder/exécuter Docker en tant qu'utilisateur régulier, ajoutez cet utilisateur au groupe Docker et actualisez l'appartenance au groupe :

```
[jedepuyd@db ~]$ sudo usermod -a -G docker jedepuyd
[jedepuyd@db ~]$ newgrp docker
```

Connectez-vous au Docker Hub :

Docker Hub contient l'image de base alpine que vous pouvez utiliser. Si vous n'avez pas encore d'ID Docker, vous devez vous enregistrer sur : https://hub.docker.com/.

```
[jedepuyd@db ~]$ docker login
Log in with your Docker ID to push and pull images from Docker Hub. If you do not have a Docker
ID, head over to https://hub.docker.com to create one.
Username: jensdepuydt
```

```
Password:
Login Succeeded
```

3. Créez le serveur Web Python.

Maintenant que la préparation est terminée, vous pouvez commencer à créer l'application réelle qui peut s'exécuter sur le périphérique IOx-enable.

```
[jedepuyd@db ~]$ mkdir iox_docker_pythonweb
[jedepuyd@db ~]$ cd iox_docker_pythonweb/
[jedepuyd@db iox_docker_pythonweb]$ vi webserver.py
[jedepuyd@db iox_docker_pythonweb]$ cat webserver.py
#!/usr/bin/env python
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
import SocketServer
import os

class S(BaseHTTPRequestHandler):
    def _set_headers(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_GET(self):
        self._set_headers()
        self.wfile.write("<html><body><h1>IOX python webserver</h1></body></html>")

def run(server_class=HTTPServer, handler_class=S, port=80):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print 'Starting webserver...'
    log_file_dir = os.getenv("CAF_APP_LOG_DIR", "/tmp")
    log_file_path = os.path.join(log_file_dir, "webserver.log")
    logf = open(log_file_path, 'w')
    logf.write('Starting webserver....\n')
    logf.close()

    httpd.serve_forever()

if __name__ == "__main__":
    from sys import argv

    if len(argv) == 2:
        run(port=int(argv[1]))
    else:
        run()
```

Ce code est un serveur web Python très minimal, que vous créez dans webserver.py. Le serveur web retourne simplement le serveur web python IOx dès qu'une requête GET est demandée. Le port sur lequel le serveur Web démarre peut être le port 80 ou le premier argument donné à webserver.py.

Ce code contient également, dans la fonction run, une écriture dans un fichier journal. Le fichier journal peut être consulté par le client IOx ou le gestionnaire local.

4. Créez le conteneur Dockerfile et Docker.

Maintenant que vous avez l'application (webserver.py) qui doit s'exécuter dans votre conteneur, il est temps de construire le conteneur Docker. Un conteneur est défini dans un fichier Dockerfile :

```
[jedepuyd@db iox_docker_pythonweb]$ vi Dockerfile
[jedepuyd@db iox_docker_pythonweb]$ cat Dockerfile
FROM alpine:3.3

RUN apk add --no-cache python
COPY webserver.py /webserver.py
```

Comme vous pouvez le voir, le fichier Dockerfile est également maintenu simple. Vous commencez avec l'image de base alpine, installez Python et copiez votre webserver.py à la racine du conteneur.

Une fois votre fichier Dockerfile prêt, vous pouvez créer le conteneur Docker :

```
jedepuyd@db iox_docker_pythonweb]$ docker build -t ioxpythonweb:1.0 .
Sending build context to Docker daemon 3.584 kB
Step 1/3 : FROM alpine:3.3
3.3: Pulling from library/alpine
10462c29356c: Pull complete
Digest: sha256:9825fd1a7e8d5feb52a2f7b40c9c4653d477b797f9ddc05b9c2bc043016d4819
Status: Downloaded newer image for alpine:3.3
 ---> 461b3f7c318a
Step 2/3 : RUN apk add --no-cache python
 ---> Running in b057a8183250
fetch http://dl-cdn.alpinelinux.org/alpine/v3.3/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.3/community/x86_64/APKINDEX.tar.gz
(1/10) Installing libbz2 (1.0.6-r4)
(2/10) Installing expat (2.1.1-r1)
(3/10) Installing libffi (3.2.1-r2)
(4/10) Installing gdbm (1.11-r1)
(5/10) Installing ncurses-terminfo-base (6.0-r6)
(6/10) Installing ncurses-terminfo (6.0-r6)
(7/10) Installing ncurses-libs (6.0-r6)
(8/10) Installing readline (6.3.008-r4)
(9/10) Installing sqlite-libs (3.9.2-r0)
(10/10) Installing python (2.7.12-r0)
Executing busybox-1.24.2-r1.trigger
OK: 51 MiB in 21 packages
 ---> 81e98c806ee9
Removing intermediate container b057a8183250
Step 3/3 : COPY webserver.py /webserver.py
 ---> c9b7474b12b2
Removing intermediate container 4705922100e6
Successfully built c9b7474b12b2

[jedepuyd@db iox_docker_pythonweb]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ioxpythonweb        1.0                 c9b7474b12b2        11 seconds ago      43.4 MB
alpine              3.3                 461b3f7c318a        2 days ago          4.81 MB
```

La commande Docker build télécharge l'image de base et installe Python et les dépendances, comme vous l'avez demandé dans le fichier Dockerfile. La dernière commande est à des fins de vérification.

5. Testez le conteneur Docker créé.

Cette étape est facultative, mais il est bon de vérifier que votre conteneur Docker récemment construit est prêt à fonctionner comme prévu.

```
[jedepuyd@db iox_docker_pythonweb]$ docker run -ti ioxpythonweb:1.0
```

```
/ # python /webserver.py 9000 &
/ # Starting webserver...

/ # netstat -tlpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9000            0.0.0.0:*               LISTEN      7/python
/ # exit
```

Comme vous pouvez le voir dans la sortie de netstat, après avoir démarré le serveur Web.py, il écoute sur le port 9000.

6. Créez le package IOx avec le conteneur Docker.

Maintenant que vous avez vérifié la fonctionnalité de votre serveur Web dans le conteneur, il est temps de préparer et de construire le package IOx pour le déploiement. Comme le fichier Dockerfile fournit des instructions pour créer un conteneur Docker, le fichier package.yaml fournit des instructions pour le client IOx afin de créer votre package IOx.

```
jedepuyd@db iox_docker_pythonweb]$ vi package.yaml
[jedepuyd@db iox_docker_pythonweb]$ cat package.yaml
descriptor-schema-version: "2.2"

info:
  name: "iox_docker_pythonweb"
  description: "simple docker python webserver on port 9000"
  version: "1.0"
  author-link: "http://www.cisco.com"
  author-name: "Jens Depuydt"

app:
  cpuarch: "x86_64"
  type: docker
  resources:
    profile: c1.small
    network:
      -
        interface-name: eth0
        ports:
            tcp: [9000]

  startup:
    rootfs: rootfs.tar
    target: ["python","/webserver.py","9000"]
```

Pour plus d'informations sur le contenu du package.yaml, cliquez ici : https://developer.cisco.com/media/iox-dev-guide-3-10-16/concepts/package_descriptor/ .

Après avoir créé le package.yaml, vous pouvez commencer à créer le package IOx.

La première étape consiste à exporter le fichier FS racine de l'image Docker :

```
[jedepuyd@db iox_docker_pythonweb]$ docker save -o rootfs.tar ioxpythonweb:1.0
```

Ensuite, vous pouvez construire le package.tar :

```
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient package .
Currently active profile:  default
Command Name: package
```

```
Checking if package descriptor file is present.
Validating descriptor file /home/jedepuyd/iox_docker_pythonweb/package.yaml with package schema
definitions
Parsing descriptor file.
Found schema version  2.2
Loading schema file for version  2.2
Validating package descriptor file..
File /home/jedepuyd/iox_docker_pythonweb/package.yaml is valid under schema version 2.2
Created Staging directory at :  /tmp/700740789
Copying contents to staging directory
Checking for application runtime type
Couldn't detect application runtime type
Creating an inner envelope for application artifacts
Generated  /tmp/700740789/artifacts.tar.gz
Calculating SHA1 checksum for package contents..
Parsing Package Metadata file :  /tmp/700740789/.package.metadata
Wrote package metadata file :  /tmp/700740789/.package.metadata
Root Directory :  /tmp/700740789
Output file:  /tmp/335805072
Path:  .package.metadata
SHA1 : 55614e72481a64726914b89801a3276a855c728a
Path:  artifacts.tar.gz
SHA1 : 816c7bbfd8ae76af451642e652bad5cf9592370c
Path:  package.yaml
SHA1 : ae75859909f6ea6947f599fd77a3f8f04fda0709
Generated package manifest at  package.mf
Generating IOx Package..
Package generated at /home/jedepuyd/iox_docker_pythonweb/package.tar
```

Le résultat de la build est un package IOx (package.tar), qui contient le conteneur Docker, prêt à être déployé sur IOx.

> **Note**: IOxclient peut également exécuter la commande docker save en une étape. Sur CentOS, cela entraîne l'exportation vers rootfs.img par défaut au lieu de rootfs.tar, ce qui donne des problèmes plus tard dans le processus. La seule étape à créer peut être effectuée à l'aide de : IOx client docker package IOxpythonweb:1.0.

8. Déployez, activez et démarrez le package sur le périphérique IOx.

Les dernières étapes consistent à déployer le package IOx sur le périphérique IOx, à l'activer et à le démarrer. Ces étapes peuvent être effectuées à l'aide d'un client IOx, d'un gestionnaire local ou d'un directeur de réseau de brouillard. Pour cet article, vous pouvez utiliser le client IOx.

Afin de déployer le package sur le périphérique IOx, utilisez le nom python_web :

```
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient app install
python_web package.tar
Currently active profile:  default
Command Name: application-install
Installation Successful. App is available at:
https://10.48.43.197:8443/iox/api/v2/hosting/apps/python_web
Successfully deployed
```

Avant de pouvoir activer l'application, vous devez définir la configuration du réseau. Pour ce faire, vous devez créer un fichier JSON. Lorsque vous activez, il peut être joint à la demande d'activation.

```
[jedepuyd@db iox_docker_pythonweb]$ vi activate.json
```

```
[jedepuyd@db iox_docker_pythonweb]$ cat activate.json
{
    "resources": {
        "profile": "c1.small",
        "network": [{"interface-name": "eth0", "network-name": "iox-nat0","port_map": {"mode":
"1to1"},"ports":{"tcp":9000}}]
    }
}
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient app activate
python_web --payload activate.json
Currently active profile :  default
Command Name: application-activate
Payload file : activate.json. Will pass it as application/json in request body..
App python_web is Activated
```

La dernière action consiste à démarrer l'application que vous venez de déployer et d'activer :

```
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient app start
python_web
Currently active profile :  default
Command Name: application-start
App python_web is Started
```

Puisque vous avez configuré votre application IOx pour écouter les requêtes HTTP de tor sur le port 9000, vous devez toujours transférer ce port de votre périphérique IOx vers le conteneur, car le conteneur est derrière NAT. Procédez comme suit sur Cisco IOS®.

```
BRU-IOT-809-1#sh iox host list det | i IPV4
    IPV4 Address of Host:       192.168.1.2
BRU-IOT-809-1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
BRU-IOT-809-1(config)#ip nat inside source static tcp 192.168.1.2 9000 interface
GigabitEthernet0 9000
BRU-IOT-809-1(config)#exit
```

La première commande répertorie l'adresse IP interne de GOS (responsable du démarrage/de l'arrêt/de l'exécution des conteneurs IOx).

La deuxième commande configure un port statique pour le port 9000 sur l'interface Gi0 du côté IOS vers GOS. Si votre périphérique est connecté via un port de couche 2 (ce qui est probablement le cas sur IR829), vous devez remplacer l'interface Gi0 par le VLAN correct qui a l'instruction ip nat outside configurée.

Vérification

Utilisez cette section pour confirmer que votre configuration fonctionne correctement.

Afin de vérifier si le serveur Web s'exécute et répond correctement, vous pouvez essayer d'accéder au serveur Web à l'aide de cette commande.

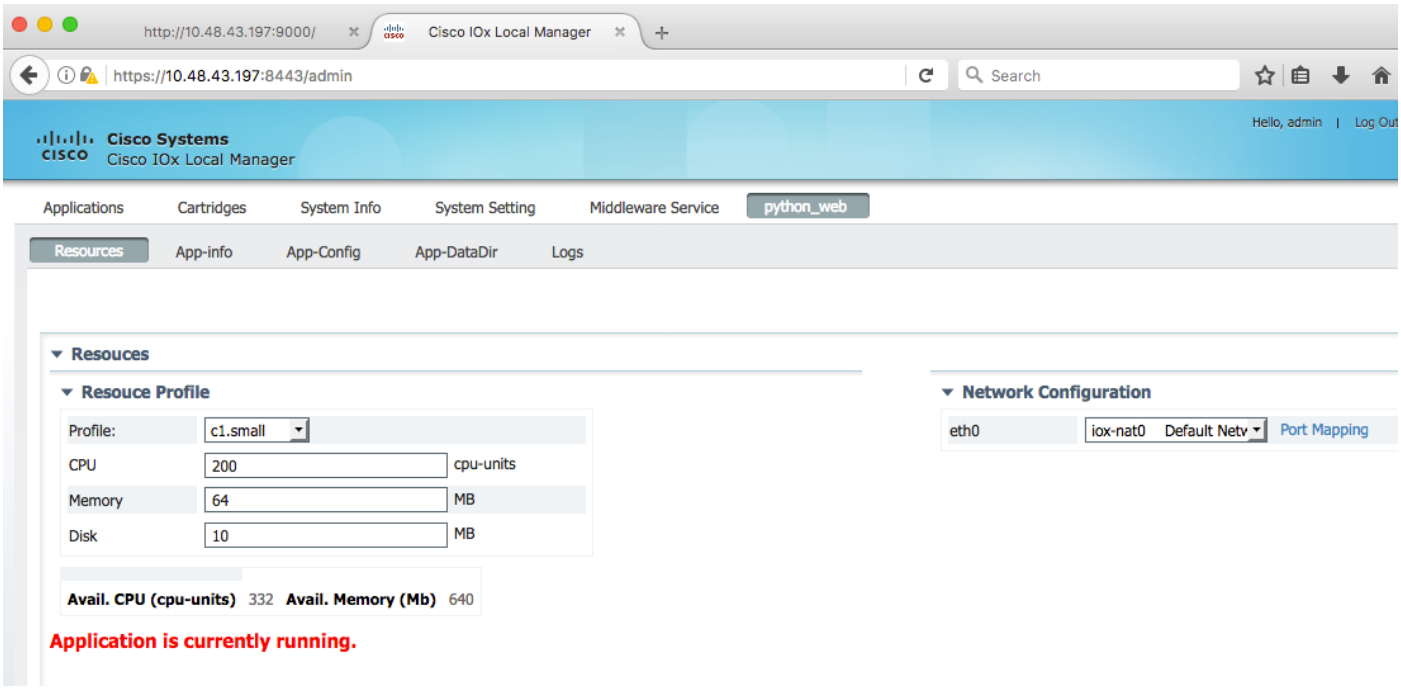
```
[jedepuyd@db iox_docker_pythonweb]$ curl http://10.48.43.197:9000/
<html><body><h1>IOX python webserver</h1></body></html>
```

Ou, à partir d'un vrai navigateur comme illustré dans l'image.

Vous pouvez également vérifier l'état de l'application à partir de l'interface de ligne de commande IOxclient :

```
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient app status
python_web
Currently active profile :  default
Command Name: application-status
Saving current configuration
App python_web is RUNNING
```

et vous pouvez également vérifier l'état de l'application à partir de l'interface utilisateur graphique de Local Manager, comme illustré dans l'image.



Afin de consulter le fichier journal dans lequel vous écrivez dans webserver.py :

```
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient app logs info
python_web
Currently active profile :  default
Command Name: application-logs-info

Log file information for :  python_web
Size_bytes : 711
Download_link : /admin/download/logs?filename=python_web-watchDog.log
Timestamp : Thu Jun 22 08:21:18 2017
Filename : watchDog.log
```

```
Size_bytes : 23
Download_link : /admin/download/logs?filename=python_web-webserver.log
Timestamp : Thu Jun 22 08:21:23 2017
Filename : webserver.log

Size_bytes : 2220
Download_link : /admin/download/logs?filename=python_web-container_log_python_web.log
Timestamp : Thu Jun 22 08:21:09 2017
Filename : container_log_python_web.log
```

Dépannage

Cette section fournit des informations que vous pouvez utiliser pour dépanner votre configuration.

Afin de dépanner l'application et/ou le conteneur, le moyen le plus simple est de se connecter à la console de l'application qui s'exécute :

```
[jedepuyd@db iox_docker_pythonweb]$ ../ioxclient_1.3.0.0_linux_amd64/ioxclient app console
python_web
Currently active profile:  default
Command Name: application-console
Console setup is complete..
Running command: [ssh -p 2222 -i python_web.pem appconsole@10.48.43.197]
The authenticity of host '[10.48.43.197]:2222 ([10.48.43.197]:2222)' can't be established.
ECDSA key fingerprint is 1d:e4:1e:e1:99:8b:1d:d5:ca:43:69:6a:a3:20:6d:56.
Are you sure you want to continue connecting (yes/no)? yes
/ # netstat -tlpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9000            0.0.0.0:*               LISTEN      19/python
/ # ps aux | grep python
   19 root       0:00 python /webserver.py 9000
```