Déployer une application IOx à l'aide d'IOxClient

Table des matières

Introduction

Objectif

Conditions préalables

- Exigences
- Composants utilisés

Aperçu

- Qu'est-ce que IOx ?
- Qu'est-ce que ioxclient ?
- Définir l'application
- Application Python
- Définir le fichier Docker
- Emballage
- Exemple YAML du référentiel de modèles Cisco DevNet IOx

Configuration

- Procédure D'Emballage
- Installation

Introduction

Ce document décrit le déploiement d'une application utilisant ioxclient.

Objectif

Ce document est dédié à la compréhension du déploiement d'une application à l'aide de ioxclient.

L'objectif de ce document est entièrement pratique, donc si vous voulez plus de détails techniques, je recommande de revoir la documentation partagée.

Conditions préalables

- Connaissances de base des systèmes d'exploitation Cisco IOS XE et Cisco IOS.
- Forte compréhension du cycle de vie des dockers et des conteneurs.
- Opérations Linux de base.

Exigences

Vérifiez si votre périphérique prend en charge iox et consultez la matrice de compatibilité : Platform Support Matrix

Aussi, s'il vous plaît télécharger client iox selon vos spécifications PC : Téléchargements

Composants utilisés

Les informations contenues dans ce document sont basées sur les versions logicielles et matérielles suivantes :

- ioxclient Version 1.17.0.0
- Routeur C8000v, version 17.12.3a
- Machine Ubuntu Version 20.04
- Installez Docker Engine version 24.0.9 ou antérieure.

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. Si votre réseau est en ligne, assurez-vous de bien comprendre l'incidence possible des commandes.

Aperçu

Qu'est-ce que IOx ?

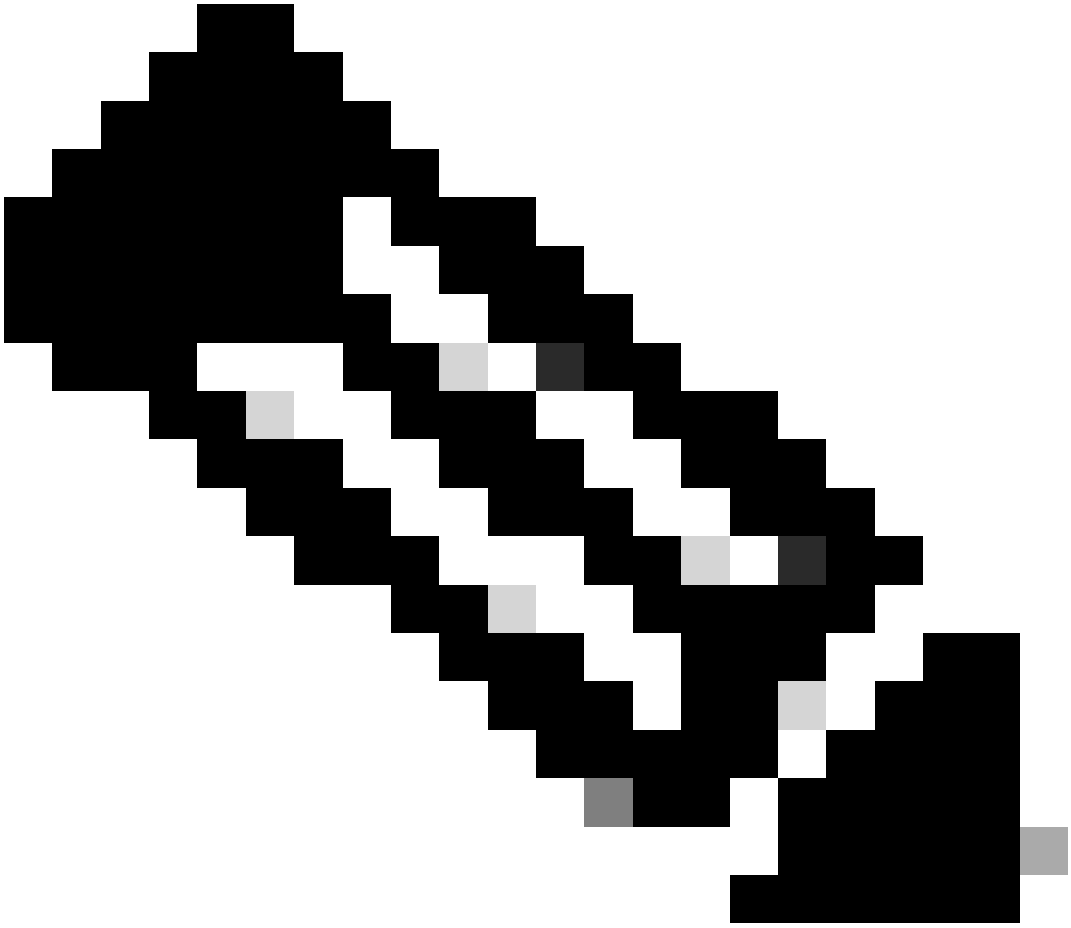
IOx est l'environnement d'applications pour les périphériques Cisco. Cette fonctionnalité nous permet d'empaqueter les applications dans un format compatible avec IOx, à l'aide d'outils tels que ioxclient.

Qu'est-ce que ioxclient ?

ioxclient est un outil en ligne de commande qui fait partie du SDK Cisco IOx. Il est utilisé pour développer, tester et déployer des applications IOx sur des périphériques Cisco IOx.

Définir l'application

Cet exemple de code crée un serveur HTTP de base écoutant via le port 8000 ; Il sert de fonctionnalité principale de l'image de génération d'image Docker (Dockerfile).



Remarque : Un fichier Docker est uniquement requis lors du développement d'images personnalisées avec des fonctionnalités spécifiques. Une application peut être extraite d'un référentiel conteneur, exportée et utilisée comme base pour ioxclient.

Application Python

```
import http.server
import socketserver

PORT = 8000
Handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print(f"Serving at port {PORT}")
    httpd.serve_forever()
```

Définir le fichier Docker

```
FROM python:alpine3.20
WORKDIR /apps
COPY . .
EXPOSE 8000
ENTRYPOINT [ "python" ]
CMD [ "main.py" ]
```

Avec ce code, vous définissez un serveur http écoutant par le port 8000, et empaquetez l'application dans un fichier Docker.

Emballage

Il est nécessaire de créer et de remplir un fichier package.YAML avec des métadonnées et des définitions de ressources pour un déploiement correct de l'application.

Le fichier YAML est un format, ce format est attrayant en raison de la syntaxe simple, dans le fichier, nous pouvons spécifier des aspects de l'application comme variables d'environnement, ports, dépendances et ainsi de suite.

Exemple YAML du référentiel de modèles Cisco DevNet IOx

```
descriptor-schema-version: "2.2"

info:
  name: iox_docker_python
  description: "IOx Docker Python Sample Application"
  version: "1.0"
  author-link: "http://www.cisco.com"
  author-name: "Cisco Systems"

app:
  cpuarch: "x86_64"
  type: docker
  resources:
    profile: c1.small

  # Specify runtime and startup
  startup:
    rootfs: rootfs.tar
    target: ["python3 main.py"]
```

Reportez-vous à la documentation pour consulter les valeurs valides du fichier de package :

- Documentation du périphérique : IOx Package Descriptor
- Référentiel Github : CiscoDevNet / modèle d'application iox

Pour ce document, le fichier de configuration YAML contient les informations suivantes :

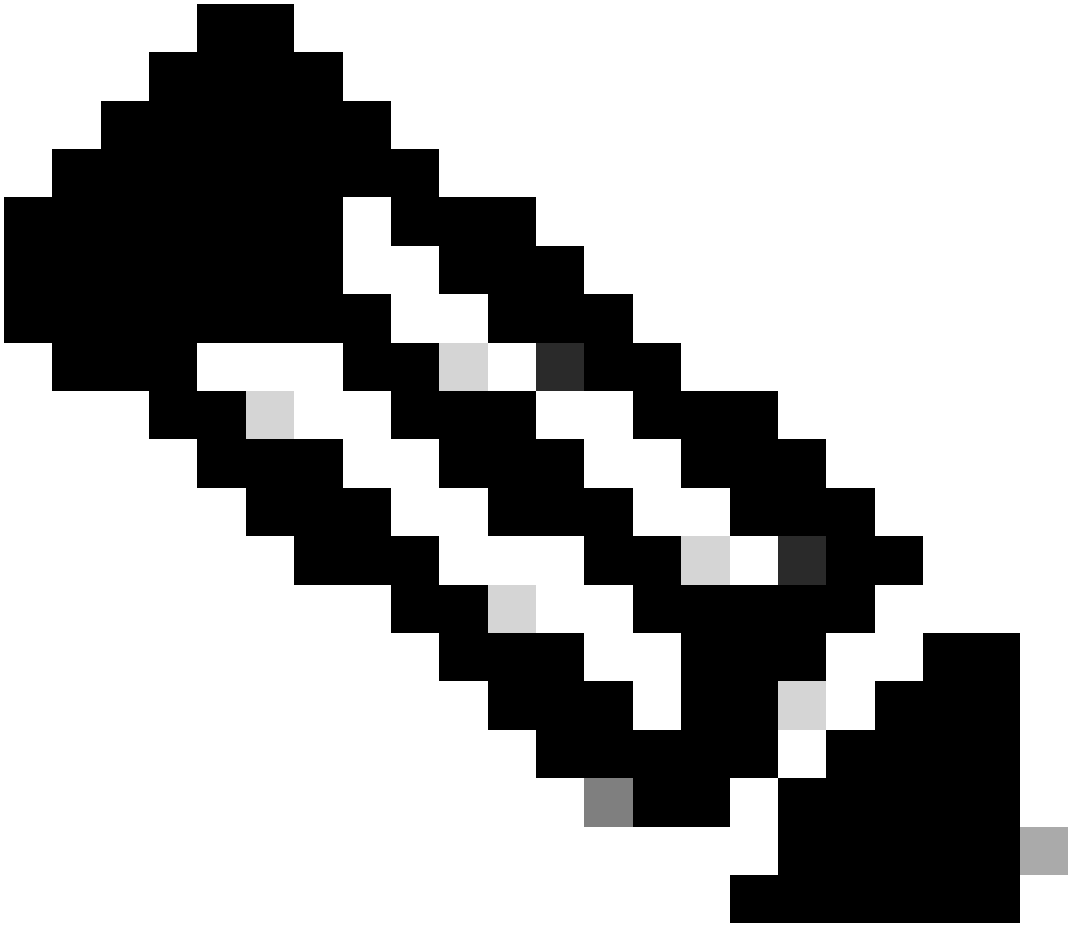
```
descriptor-schema-version: "2.2"

info:
  name: "tac_app"
  description: "tac_app"
  version: "1.0"
  author-name: "TAC-TEST"

app:
  cpuarch: x86_64
  type: docker
  resources:
    profile: "custom"
    cpu: 100                # CPU en MHz assigned to the application.
    disk: 50               # Storage in MB for the disk
    memory: 128           # Memory en MB assigned to the application.
    network:
      -
        interface-name: eth0
        ports:
          tcp:
            - 8000
  startup:
    rootfs: "rootfs.tar"         # Container file system
    target: "python main.py"     # Command to start the application
```

En raison d'une incompatibilité entre Docker Engine version 25.0 et ioxclient, l'approche recommandée est d'utiliser une distribution Linux qui prend en charge Docker Engine version 24.0.9 ou antérieure, car la version 24.0.9 est la dernière version prise en charge pour la compatibilité avec ioxclient.

Dans cet exemple, l'image Docker utilisée pour démontrer la fonctionnalité du client IOx a été créée sur une machine virtuelle basée sur Ubuntu exécutant la version 20.04, choisie spécifiquement parce que les fichiers binaires .deb du moteur Docker sont disponibles pour cette distribution/version.



Remarque : La seule façon d'installer des versions plus anciennes de Docker est de l'installer à travers les fichiers bin. Ces fichiers binaires sont des versions spécifiques du logiciel qui sont déjà préparées pour s'exécuter directement sur un système d'exploitation particulier.

Configuration

Pour préparer la machine virtuelle avec les spécifications mentionnées, procédez à l'installation du fichier binaires à partir d'une ancienne version de Docker :

```
wget https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/docker-ce-cli_24.0.9-1~ubun
wget https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/docker-ce_24.0.9-1~ubuntu.2
wget https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/docker-buildx-plugin_0.11.2
wget https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/docker-compose-plugin_2.21.
wget https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/containerd.io_1.7.19-1_amd6

And installed them:
```

```
sudo dpkg -i ./containerd.io_1.7.19-1_amd64.deb \
./docker-ce_24.0.9-1~ubuntu.20.04~focal_amd64.deb \
./docker-ce-cli_24.0.9-1~ubuntu.20.04~focal_amd64.deb \
./docker-buildx-plugin_0.11.2-1~ubuntu.20.04~focal_amd64.deb \
./docker-compose-plugin_2.21.0-1~ubuntu.20.04~focal_amd64.deb
```

Une fois tous les fichiers installés, la machine est prête à empaqueter l'application iox.

Procédure D'Emballage

Transférez le code python et le fichier Docker vers la machine virtuelle, vérifiez que les deux fichiers se trouvent dans le même répertoire, puis continuez à construire l'image Docker :

```
sudo docker build -t tac_app .
```

Pour répertorier les images Docker disponibles dans le référentiel de l'ordinateur local, exécutez la commande ci-dessous :

```
ubuntu@ip-172-31-30-249:~$ sudo docker images

REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
tac_app       latest    94a1c2ba4b08   19 seconds ago   1.78GB
```

D'ici il y a 2 alternatives

1 - Créez le package de l'application à l'aide de l'image Docker et du fichier descripteur package.yaml

2 - Exportez l'image en tant que système de fichiers racine et compressez-la avec le fichier YAML de descripteur

Option 1 - Emballage de l'image Docker et du fichier YAML :

Déplacez-vous vers le répertoire cible où vous souhaitez empaqueter l'image et le fichier YAML.

```
ubuntu@ip-172-31-30-249:~/tes0$ ls
package.yaml
```

Ensuite, empaquetez le fichier en exécutant cette commande :

```
ioxclient docker package tac_app package.yaml
...

Example:
ubuntu@ip-172-31-30-249:~/tese$ sudo /home/ubuntu/ioxclient_1.17.0.0_linux_amd64/ioxclient docker packa
Currently active profile: default
Secure client authentication: no
Command Name: docker-package
Timestamp at DockerPackage start: 1748211382584
Using the package descriptor file in the project dir
Validating descriptor file package.yaml with package schema definitions
Parsing descriptor file..
Found schema version 2.7
Loading schema file for version 2.7
Validating package descriptor file..
File package.yaml is valid under schema version 2.7
Generating IOx package of type docker with layers as rootfs
Replacing symbolically linked layers in docker rootfs, if any
No symbolically linked layers found in rootfs. No changes made in rootfs
Removing emulation layers in docker rootfs, if any
The docker image is better left in it's pristine state
Updated package metadata file :/home/ubuntu/tes0/.package.metadata
No rsa key and/or certificate files provided to sign the package
------------------------------------------------------------------
Generating the envelope package
------------------------------------------------------------------
Checking if package descriptor file is present..
Skipping descriptor schema validation..
Created Staging directory at : /tmp/1093485025
Copying contents to staging directory
Timestamp before CopyTree: 1748211503878
Timestamp after CopyTree: 1748211575671
Creating artifacts manifest file
Creating an inner envelope for application artifacts
Including rootfs.tar
Generated /tmp/1093485025/artifacts.tar.gz
Parsing Package Metadata file /tmp/1093485025/.package.metadata
Updated package metadata file /tmp/1093485025/.package.metadata
Calculating SHA256 checksum for package contents..
Timestamp before SHA256: 1748211630718
Timestamp after SHA256: 1748211630718
Path:.package.metadata
SHA256: 50c922f103ddc01a5dc7a98d6cacefb167f4a2c692dfc521231bb42f0c3dcf55 Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211630719
Path: artifacts.mf
SHA256: 511008aa2d1418daf1770768fb79c90f16814ff7789d03beb4f4ea1bf4fae8f2 Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211634941
Path: artifacts.tar.gz
SHA256: 0cc3f69af50cf0a01ec9a1400c440f60a0dff55369bd309b6dfc69715302425+ Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211634952
Path: envelope_package.tar.gz
SHA256: d492de09441a241f879cd268cd1b3424ee79a58a9495aa77ae5b11cab2fd55da Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211634963
Path: package.yaml
SHA256: d8dc7253443ff3ad080c42bc8d82db4c3ea7ae9b2d0e2f827fbaf2bc80245f62 Generated package manifest at
Generating IOx Package..
Package docker image tac_app at /home/ubuntu/tes0/package.tar
ubuntu@ip-172-31-30-249:~/tes0$ |
```

Cet ensemble d'actions était responsable de la génération du paquet tar bundle. Afin d'inspecter le contenu du paquet, nous pouvons le décompresser en utilisant l'utilitaire tar

```
ubuntu@ip-172-31-30-249:~/tes0$ tar -tf package.tar
package.yaml
artifacts.mf
.package.metadata
package.mf
envelope_package.tar.gz
artifacts.tar.gz
```

Option 2 : exportation de l'image Docker en tant que système de fichiers racine et création d'un package avec le fichier YAML de descripteur.

Exécutez la commande dans le répertoire dans lequel l'image doit être créée :

```
ubuntu@ip-172-31-30-249:~/tac_app$ sudo docker save tac_app -o rootfs.tar
```

Cette commande exporte l'image Docker en tant que bundle contenant le système de fichiers racine qui se monte sur / dans le conteneur.

Déplacez le fichier package.YAML vers l'emplacement spécifié. Une fois terminée, la structure de répertoires doit ressembler à celle illustrée ci-dessous :

```
ubuntu@ip-172-31-30-249:~/tac_app$ ls
package.yaml   rootfs.tar
```

La dernière étape consiste à empaqueter l'image Docker en exécutant la commande suivante :

```
ioxclient docker package tac_app package.yaml
...

ubuntu@ip-172-31-30-249:~/tac_app$ ioxclient package .
Currently active profile :  default
Secure client authentication:  no
Command Name:  package
No rsa key and/or certificate files provided to sign the package
Checking if package descriptor file is present..
Validating descriptor file /home/ubuntu/tac_app/package.yaml with package schema definitions
Parsing descriptor file..
Found schema version  2.7
Loading schema file for version  2.7
Validating package descriptor file..
File /home/ubuntu/tac_app/package.yaml is valid under schema version 2.7
```

```
Created Staging directory at :  /tmp/2119895371
Copying contents to staging directory
Timestamp before CopyTree: 1748374177879


Timestamp after CopyTree: 1748374357306
Creating artifacts manifest file
Creating an inner envelope for application artifacts

Generated  /tmp/2119895371/artifacts.tar.gz
Updated package metadata file :  /tmp/2119895371/.package.metadata
Calculating SHA256 checksum for package contents..
Timestamp before SHA256: 1748374566796
Timestamp after SHA256: 1748374566796
Path:  .package.metadata
SHA256 : 4fad07c3ac4d817db17bacc8563b4c632bc408d2a9cbdcb5e7a526c1c5c6e04e
Timestamp before SHA256: 1748374566796
Timestamp after SHA256: 1748374566809
Path:  artifacts.mf
SHA256 : d448a678ae952f9fe74dc19172aba17e283a5e268aca817fefc78b585f02b492
Timestamp before SHA256: 1748374566809
Timestamp after SHA256: 1748374575477
Path:  artifacts.tar.gz
SHA256 : 64d70f43be692e3cee61d906036efef45ba29e945437237e1870628ce64d5147
Timestamp before SHA256: 1748374575477
Timestamp after SHA256: 1748374575489
Path:  package.yaml
SHA256 : d8dc7253443ff3ad080c42bc8d82db4c3ea7ae9b2d0e2f827fbaf2bc80245f62
Generated package manifest at  package.mf
Generating IOx Package..
Package generated at /home/ubuntu/tac_app/package.tar
```

À la suite de ces actions, le fichier package.tar est généré et préparé pour le déploiement. Pour examiner le contenu du package, exécutez la commande suivante :

```
ubuntu@ip-172-31-30-249:~/tac_app$ tar -tf  tac_app.tar
package.yaml
artifacts.mf
.package.metadata
package.mf
artifacts.tar.gz
```

Installation

Une fois l'application préparée, l'étape finale consiste à l'installer sur le périphérique cible en exécutant la commande en mode d'exécution privilégié, comme indiqué ci-dessous :

```
 app-hosting install appid tacapp package bootflash:package.tar
```

Attendez environ 1 minute et vérifiez si l'application s'exécute correctement :

```
Router# show app-hosting list
App id                                   State
---------------------------------------------------------
  tacapp                                 RUNNING
```

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.