Bereitstellen einer IOx-Anwendung mithilfe von IOxClient

Inhalt

Einleitung

Ziel

Voraussetzungen

- Anforderungen
- Verwendete Komponenten

Überblick

- Was ist IOx?
- Was ist ioxclient?
- Definieren der Anwendung
- Python-Anwendung
- Docker-Datei definieren
- Paket
- YAML-Beispiel aus dem Cisco DevNet IOx Template Repository

Konfiguration

- Verpackungsverfahren
- Installation

Einleitung

In diesem Dokument wird die Bereitstellung einer Anwendung mit ioxclient beschrieben.

Ziel

Dieses Dokument befasst sich mit dem Verständnis der Bereitstellung einer Anwendung mit ioxclient.

Der Schwerpunkt dieses Dokuments ist sehr praktisch. Wenn Sie also weitere technische Details benötigen, empfehle ich Ihnen, die freigegebene Dokumentation zu lesen.

Voraussetzungen

- Grundkenntnisse der Betriebssysteme Cisco IOS XE und Cisco IOS
- Umfassendes Verständnis von Docker- und Container-Lebenszyklus.
- Grundlegende Linux-Prozesse.

Anforderungen

Überprüfen Sie, ob Ihr Gerät Iox unterstützt, und überprüfen Sie die Kompatibilitätsmatrix: Platform Support Matrix.

Bitte laden Sie auch den iox-Client nach Ihren PC-Spezifikationen herunter: Downloads

Verwendete Komponenten

Die in diesem Dokument enthaltenen Informationen basieren auf den folgenden Software- und Hardwareversionen:

- ioxclient Version 1.17.0.0
- Router C8000v, Version 17.12.3a
- Ubuntu-Computerversion 20.04
- Installieren Sie Docker Engine Version 24.0.9 oder älter.

Die Informationen in diesem Dokument beziehen sich auf Geräte in einer speziell eingerichteten Testumgebung. Alle Geräte, die in diesem Dokument benutzt wurden, begannen mit einer gelöschten (Nichterfüllungs) Konfiguration. Wenn Ihr Netzwerk in Betrieb ist, stellen Sie sicher, dass Sie die möglichen Auswirkungen aller Befehle kennen.

Überblick

Was ist IOx?

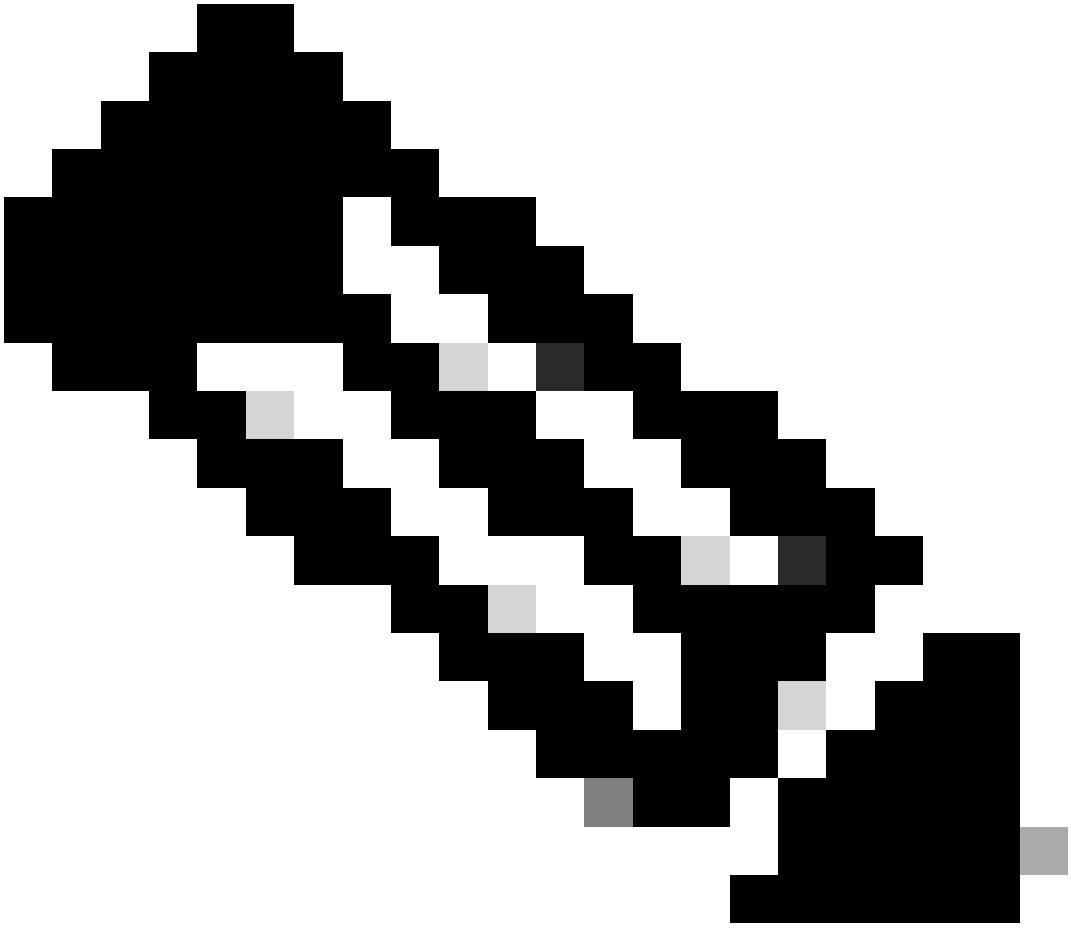
IOx ist die Anwendungsumgebung für Cisco Geräte. Mit dieser Funktion können wir die Anwendungen mithilfe von Tools wie ioxclient in einem IOx-kompatiblen Format verpacken.

Was ist ioxclient?

ioxclient ist ein Kommandozeilen-Tool und Teil des Cisco IOx SDK. Es dient zum Entwickeln, Testen und Bereitstellen von IOx-Anwendungen auf Cisco IOx-Geräten.

Definieren der Anwendung

Mit diesem Beispielcode wird ein einfacher HTTP-Server erstellt, der über Port 8000 überwacht wird. Es dient als Kernfunktionalität des Docker Image Build Images (Dockerfile).



Anmerkung: Eine Dockerdatei ist nur erforderlich, wenn benutzerdefinierte Bilder mit spezifischen Funktionen entwickelt werden. Eine Anwendung kann aus einem Container-Repository abgerufen, exportiert und als Basis für ioxclient verwendet werden.

Python-Anwendung

```
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()
```

Docker-Datei definieren

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

Mit diesem Code legen Sie einen HTTP-Server fest, der Port 8000 abhört, und packen die Anwendung in eine Docker-Datei.

Paket

Es ist erforderlich, eine package.YAML-Datei mit Metadaten und Ressourcendefinitionen zu erstellen und aufzufüllen, damit die Anwendung ordnungsgemäß bereitgestellt werden kann.

Die YAML-Datei ist ein Format, dieses Format ist aufgrund der einfachen Syntax attraktiv, innerhalb der Datei können wir Aspekte der Anwendung als Umgebungsvariablen, Ports, Abhängigkeiten usw. angeben.

YAML-Beispiel aus dem Cisco DevNet IOx Template Repository

```
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"]
```

Die gültigen Werte in der Paketdatei finden Sie in der Dokumentation:

- Gerätedokumentation: IOx-Paketbeschreibung
- Github-Archiv: CiscoDevNet/iox-App-Vorlage

Die YAML-Konfigurationsdatei für dieses Dokument enthält die folgenden Informationen:

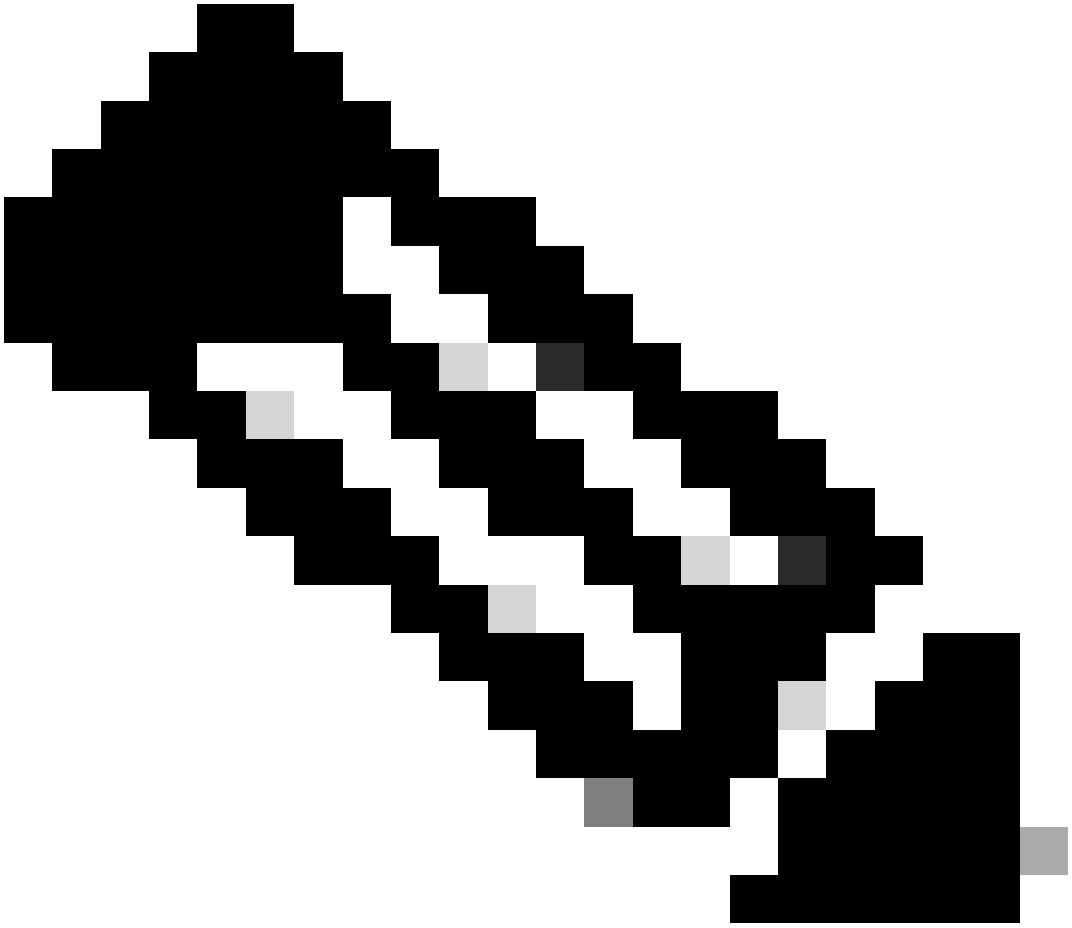
```
descriptor-schema-version: "2.2"

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
```

Aufgrund einer Inkompatibilität zwischen Docker Engine Version 25.0 und ioxclient wird empfohlen, eine Linux-Distribution zu verwenden, die Docker Engine Version 24.0.9 oder früher unterstützt, da Version 24.0.9 die neueste unterstützte Version für die Kompatibilität mit ioxclient ist.

In diesem Beispiel wurde das Docker-Image, das zur Demonstration der IOx-Client-Funktionalität verwendet wurde, auf einem Ubuntu-basierten virtuellen System mit Version 20.04 erstellt. Dieses Image wurde speziell deshalb ausgewählt, weil die .deb-Binärdateien der Docker-Engine für diese Distribution/Version verfügbar sind.



Anmerkung: Die einzige Möglichkeit, ältere Versionen von Docker zu installieren, besteht darin, es über die bin-Dateien zu installieren. Bei diesen Binärdateien handelt es sich um bestimmte Versionen der Software, die bereits für die direkte Ausführung auf einem bestimmten Betriebssystem vorbereitet sind.

Konfiguration

Um die VM mit den genannten Spezifikationen vorzubereiten, fahren Sie mit der Installation der Binärdatei von einer alten Version von Docker fort:

```
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
```

Sobald alle Dateien installiert sind, ist der Computer bereit, die iox-Anwendung zu packen.

Verpackungsverfahren

Übertragen Sie den Python-Code und die Docker-Datei auf das virtuelle System. Überprüfen Sie, ob sich beide Dateien im gleichen Verzeichnis befinden, und fahren Sie dann mit der Erstellung des Docker-Abbilds fort:

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

Führen Sie den folgenden Befehl aus, um die im Repository des lokalen Systems verfügbaren Docker-Images aufzulisten:

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

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

Von hier gibt es 2 Alternativen

1 - Verpacken Sie die Anwendung mit dem Docker-Bild und der Beschreibungsdatei package.yaml

2 - Exportieren Sie das Bild als Root-Dateisystem und packen Sie es mit der Beschreibung YAML-Datei

Option 1 - Verpacken des Docker-Bilds und der YAML-Datei:

Wechseln Sie zum Zielverzeichnis, in das Sie das Bild und die YAML-Datei packen möchten.

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

Packen Sie die Datei dann mit dem folgenden Befehl ein:

```
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$ |
```

Diese Aktionen waren für die Generierung des Paket-TAR-Pakets verantwortlich. Um den Paketinhalt zu überprüfen, können wir ihn mithilfe des TAR-Dienstprogramms dekomprimieren

```
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 - Exportieren des Docker-Bildes als Root-Dateisystem und Verpacken mit dem Deskriptor YAML-Datei.

Führen Sie den Befehl in dem Verzeichnis aus, in dem das Abbild erstellt werden soll:

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

Dieser Befehl exportiert das Docker-Image als Bündel, das das Root-Dateisystem enthält, das auf / im Container gemountet wird.

Verschieben Sie die Datei package.YAML an den angegebenen Speicherort. Nach Abschluss des Vorgangs muss die Verzeichnisstruktur so aussehen, als würde sie wie folgt angezeigt:

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

Im letzten Schritt wird das Docker-Bild mit dem folgenden Befehl verpackt:

```
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
```

Als Ergebnis dieser Aktionen wird die Datei package.tar generiert und für die Bereitstellung vorbereitet. Führen Sie den folgenden Befehl aus, um den Inhalt des Pakets zu überprüfen:

```
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

Nachdem die Anwendung vorbereitet wurde, besteht der letzte Schritt darin, sie auf dem Zielgerät zu installieren, indem der Befehl im privilegierten EXEC-Modus wie folgt ausgeführt wird:

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

Warten Sie etwa eine Minute, und überprüfen Sie, ob die Anwendung erfolgreich ausgeführt wird:

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

Informationen zu dieser Übersetzung

Cisco hat dieses Dokument maschinell übersetzen und von einem menschlichen Übersetzer editieren und korrigieren lassen, um unseren Benutzern auf der ganzen Welt Support-Inhalte in ihrer eigenen Sprache zu bieten. Bitte beachten Sie, dass selbst die beste maschinelle Übersetzung nicht so genau ist wie eine von einem professionellen Übersetzer angefertigte. Cisco Systems, Inc. übernimmt keine Haftung für die Richtigkeit dieser Übersetzungen und empfiehlt, immer das englische Originaldokument (siehe bereitgestellter Link) heranzuziehen.