Implementar una aplicación IOx mediante IOxClient

Contenido

Introducción

Objetivo

Prerequisites

- Requirements
- Componentes Utilizados

Overview

- ¿Qué es IOx?
- ¿Qué es ioxclient?
- Definir la aplicación
- Aplicación Python
- Definir el archivo Docker
- Paquete
- Ejemplo de YAML del repositorio de plantillas de Cisco DevNet IOSx

Configuración

- Procedimiento de empaquetado
- Instalación

Introducción

Este documento describe la implementación de una aplicación mediante ioxclient.

Objetivo

Este documento se dedica a entender la implementación de una aplicación usando ioxclient.

El enfoque de este documento es totalmente práctico, por lo que si desea más detalles técnicos, le recomiendo que revise la documentación compartida.

Prerequisites

- Conocimiento básico de los sistemas operativos Cisco IOS XE y Cisco IOS.
- Amplia comprensión del ciclo de vida de los contenedores y Docker.
- Operaciones básicas de Linux.

Requirements

Confirme si su dispositivo admite iox, puede comprobar la matriz de compatibilidad: Platform Support Matrix (Matriz de compatibilidad de plataformas)

También, por favor, descargue el cliente iox de acuerdo con las especificaciones de su PC: Descargas

Componentes Utilizados

La información contenida en este documento se basa en estas versiones de software y hardware:

- ioxclient Versión 1.17.0.0
- Router C8000v, versión 17.12.3a
- Ubuntu Machine Versión 20.04
- Instale Docker Engine versión 24.0.9 o posterior.

La información que contiene este documento se creó a partir de los dispositivos en un ambiente de laboratorio específico. Todos los dispositivos que se utilizan en este documento se pusieron en funcionamiento con una configuración verificada (predeterminada). Si tiene una red en vivo, asegúrese de entender el posible impacto de cualquier comando.

Overview

¿Qué es IOx?

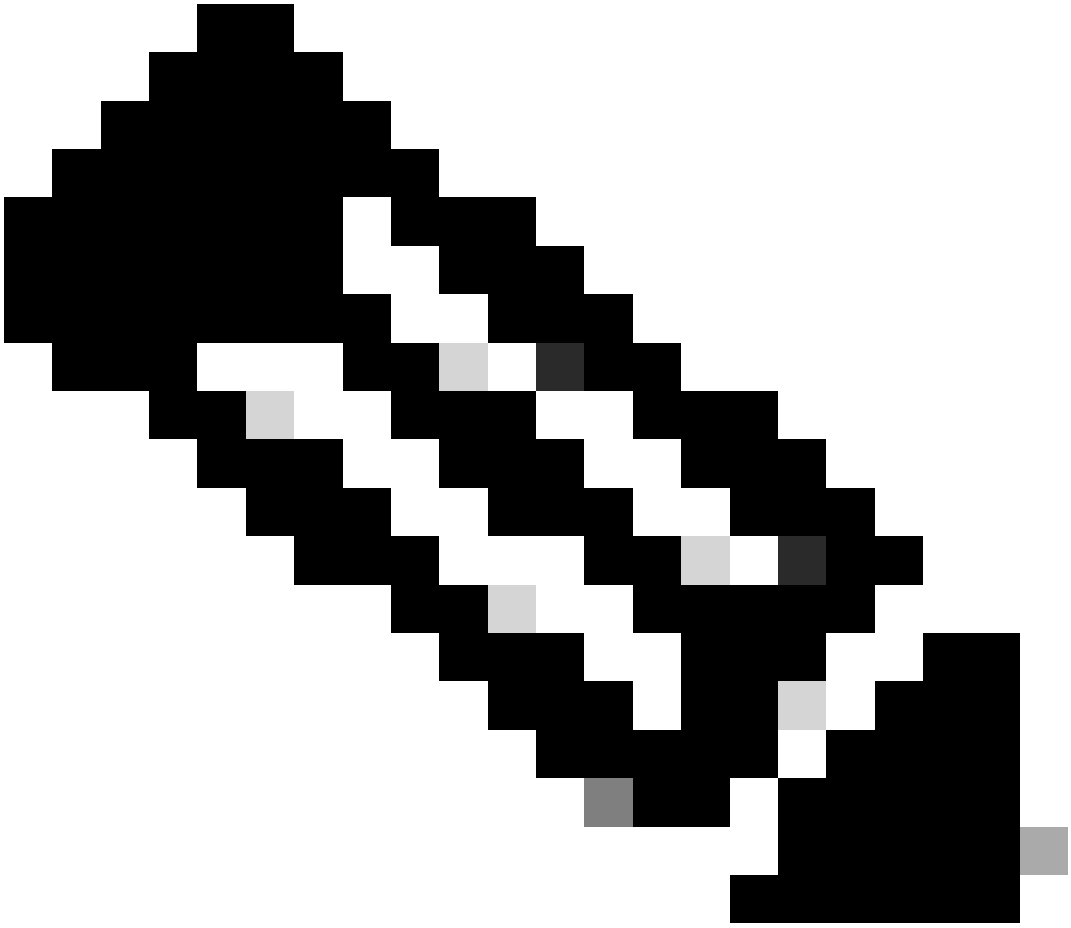
IOx es el entorno de aplicaciones para dispositivos Cisco, esta función nos permite empaquetar las aplicaciones en un formato compatible con IOx, utilizando herramientas como ioxclient.

¿Qué es ioxclient?

ioxclient es una herramienta de línea de comandos que forma parte del SDK de Cisco IOSx. Se utiliza para desarrollar, probar e implementar aplicaciones IOx en dispositivos Cisco IOx.

Definir la aplicación

Este código de ejemplo crea un servidor HTTP básico que escucha a través del puerto 8000; Sirve como la funcionalidad principal de la imagen de creación de Docker (Dockerfile).



Nota: Solo se requiere un archivo Dockerfile cuando se desarrollan imágenes personalizadas con funcionalidad específica. Una aplicación puede extraerse de un repositorio contenedor, exportarse y utilizarse como base para ioxclient.

Aplicación 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()
```

Definir el archivo Docker

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

Con este código usted configura un servidor http que escucha a través del puerto 8000, y empaqueta la aplicación en un archivo Docker.

Paquete

Es necesario crear y rellenar un archivo package.YAML con definiciones de metadatos y recursos para la correcta implementación de la aplicación.

El archivo YAML es un formato, este formato es atractivo debido a la sintaxis simple, dentro del archivo podemos especificar aspectos de la aplicación como variables ambientales, puertos, dependencias, etc.

Ejemplo de YAML del repositorio de plantillas de Cisco DevNet IOSx

```
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"]
```

Consulte la documentación para consultar los valores válidos en el archivo de paquete:

- Documentación de Devnet: Descriptor del Paquete de IOSx
- Repositorio de Github: CiscoDevNet/iox-app-template

Para este documento, el archivo de configuración YAML contiene la siguiente información:

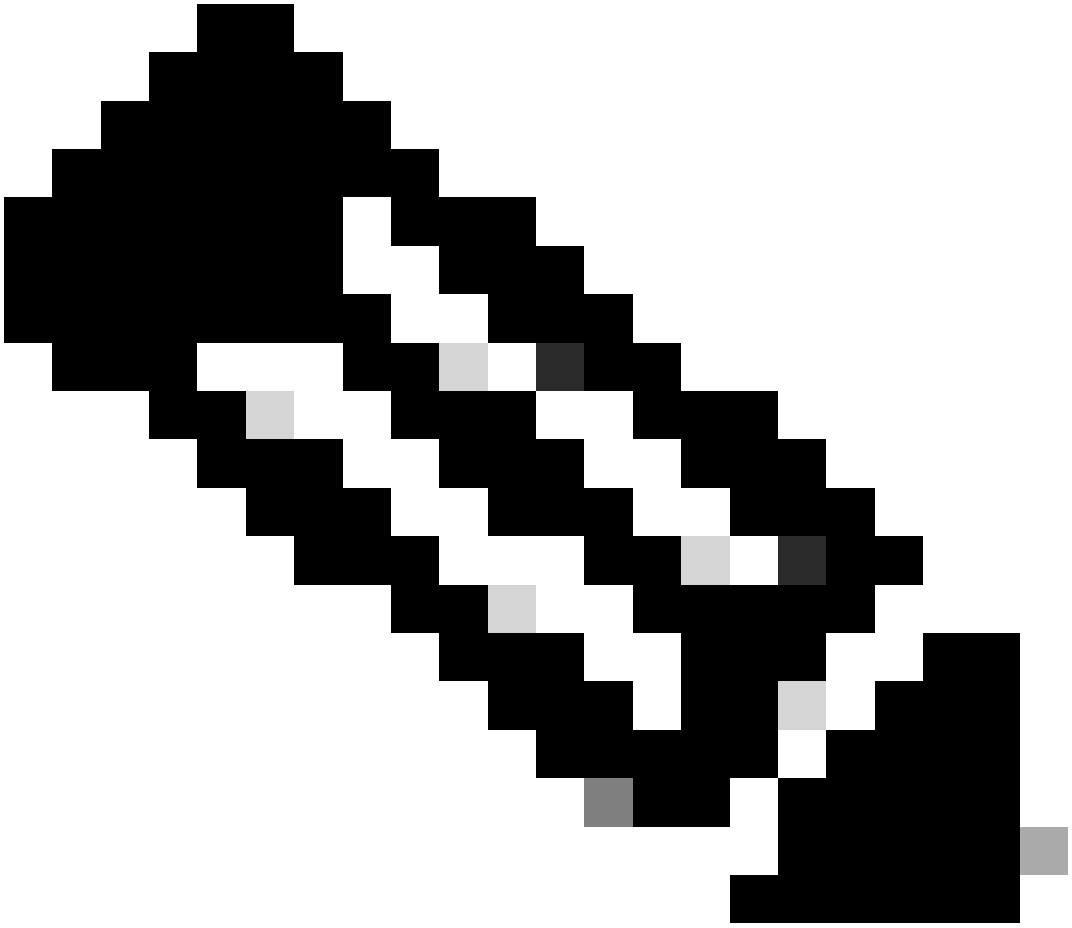
```
descriptor-schema-version: "2.2"

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
```

Debido a una incompatibilidad entre Docker Engine versión 25.0 y ioxclient, el enfoque recomendado es utilizar una distribución Linux que soporte Docker Engine versión 24.0.9 o anterior, ya que la versión 24.0.9 es la última versión compatible para la compatibilidad con ioxclient.

En este ejemplo, la imagen Docker utilizada para demostrar la funcionalidad del cliente IOx se creó en una máquina virtual basada en Ubuntu que ejecuta la versión 20.04, elegida específicamente porque los binarios Docker Engine .deb están disponibles para esta distribución/versión.



Nota: La única manera de instalar versiones anteriores de Docker es instalarlo a través de los archivos bin. Estos binarios son versiones específicas del software que ya están preparadas para ejecutarse directamente en un sistema operativo determinado.

Configuración

Para preparar la máquina virtual con las especificaciones mencionadas, proceda a instalar el archivo binario desde una versión anterior 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
```

Una vez instalados todos los archivos, el equipo está listo para empaquetar la aplicación iox.

Procedimiento de empaquetado

Transfiera el código de Python y el archivo Docker a la máquina virtual, verifique que ambos archivos estén en el mismo directorio y, a continuación, proceda a crear la imagen Docker:

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

Para mostrar las imágenes de Docker disponibles en el repositorio del equipo local, ejecute el comando que se muestra a continuación:

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

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

A partir de aquí hay 2 alternativas

1 - Empaquetar la aplicación utilizando la imagen Docker y el archivo descriptor package.yaml

2 - Exportar la imagen como un sistema de archivos root y empaquetarla con el archivo YAML descriptor

Opción 1 - Empaquetar la imagen de Docker y el archivo YAML:

Vaya al directorio de destino donde desea empaquetar la imagen y el archivo YAML.

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

Luego, empaquete el archivo ejecutando este comando:

```
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 10x 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: 51100 8aa2d1418daf1770768fb79c90f16814ff7789d03beb4f4ea1bf4fae8f2 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$ |
```

Este conjunto de acciones fue responsable de la generación del paquete tar. Para inspeccionar el contenido del paquete, podemos descomprimirlo usando la utilidad 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
```

Opción 2: exportar la imagen Docker como sistema de archivos raíz y empaquetarla con el archivo YAML descriptor.

Ejecute el comando en el directorio donde se va a crear la imagen:

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

Este comando exporta la imagen Docker como un conjunto que contiene el sistema de archivos raíz que se monta en / dentro del contenedor.

Mueva el archivo package.YAML a la ubicación especificada. Una vez completada, la estructura de directorios debe tener el aspecto siguiente:

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

El último paso implica empaquetar la imagen Docker mediante la ejecución de este comando:

```
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
```

Como resultado de estas acciones, se genera el archivo package.tar y se prepara para su implementación. Para examinar el contenido del paquete, ejecute el comando mostrado:

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

Instalación

Una vez preparada la aplicación, el paso final consiste en instalarla en el dispositivo de destino mediante la ejecución del comando en el modo EXEC privilegiado, como se muestra a continuación:

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

Espere aproximadamente 1 minuto y confirme si la aplicación se está ejecutando correctamente:

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

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).