使用IOxClient部署IOx應用程式

目錄

簡介

目標

必要條件

- 需求
- 採用元件

概觀

- 什麼是IOx?
- 什麼是ioxclient?
- 定義應用程式
- Python應用程式
- 定義Docker檔案
- 包
- 來自Cisco DevNet IOx模板儲存庫的YAML示例

組態

- 包裝程式
- 安裝

簡介

本文檔介紹使用ioxclient部署應用程式。

目標

本文檔專用於瞭解使用ioxclient的應用程式的部署。

本文的重點非常實際，因此，如果您需要更多技術細節，我建議檢視共用的文檔。

必要條件

- Cisco IOS XE和Cisco IOS作業系統的基礎知識。
- 深入瞭解Docker和容器生命週期。
- 基本的Linux操作。

需求

確認您的裝置是否支援iox，您可以檢視相容性表：平台支援表

此外，請根據您的PC規格下載iox客戶端：下載

採用元件

本檔案中的資訊是根據以下軟體和硬體版本：

- ioxclient版本1.17.0.0
- 路由器C8000v，版本17.12.3a
- Ubuntu機器版本20.04
- 安裝Docker Engine 24.0.9版或更高版本。

本文中的資訊是根據特定實驗室環境內的裝置所建立。文中使用到的所有裝置皆從已清除（預設）的組態來啟動。如果您的網路運作中，請確保您瞭解任何指令可能造成的影響。

概觀

什麼是IOx?

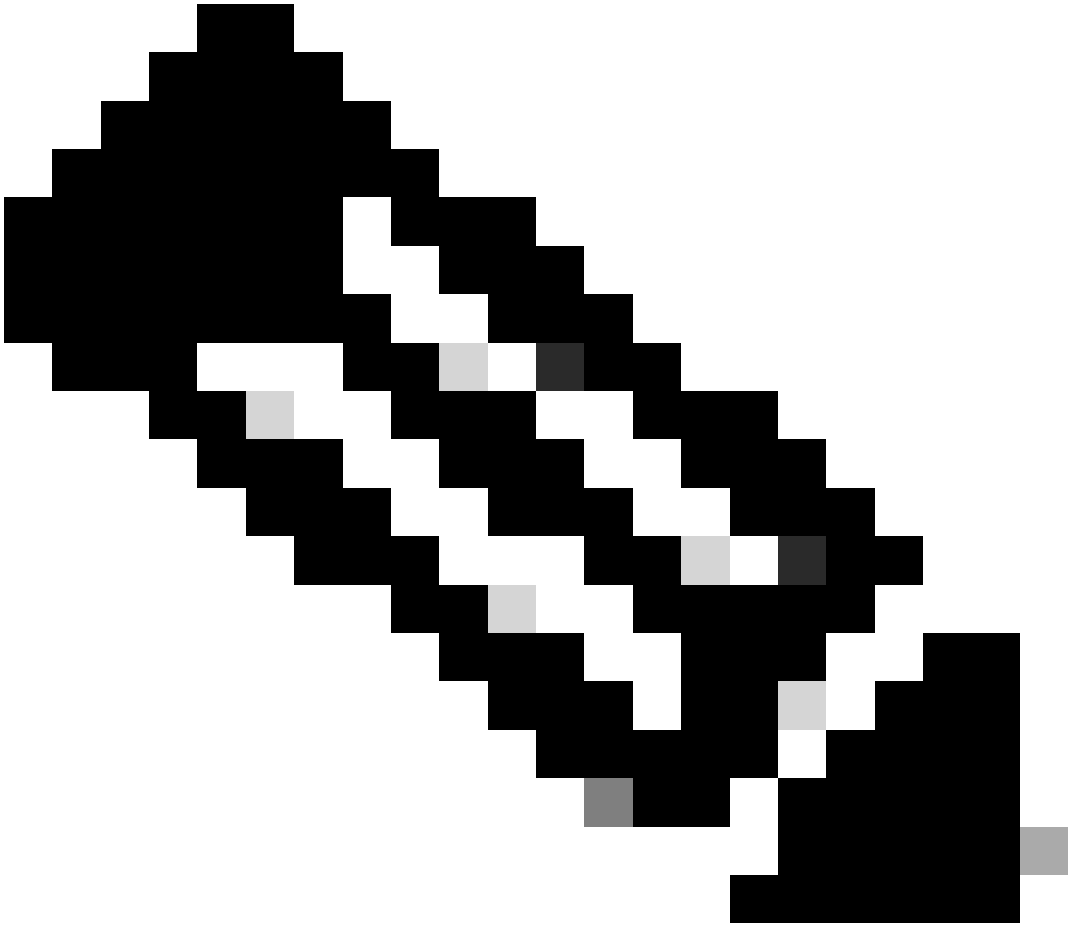
IOx是思科裝置的應用環境，此功能允許我們使用ioxclient等工具將應用打包成與IOx相容的格式。

什麼是ioxclient?

ioxclient是一個命令列工具，它是Cisco IOx SDK的一部分。它用於開發、測試和部署Cisco IOx裝置上的IOx應用。

定義應用程式

此示例代碼建立通過埠8000偵聽的基本HTTP伺服器；它充當Docker映像構建映像(Dockerfile)的核心功能。



附註：只有在開發具有特定功能的自定義映像時，才需要Dockerfile。應用程式可以從容器儲存庫中拉出，匯出並用作ioxclient的基礎。

Python應用程式

```
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()
```

定義Docker檔案

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

使用此代碼，您可以設定一個http伺服器通過埠8000進行偵聽，並將應用程式打包到Docker檔案中。

包

為正確部署應用程式，需要使用後設資料和資源定義建立和填充package.YAML檔案。

YAML檔案是一種格式，這種格式由於簡單的語法而頗具吸引力，在檔案中我們可以指定應用的各個方面，如環境變數、埠、依賴項等。

來自Cisco DevNet IOx模板儲存庫的YAML示例

```
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"]
```

請參考文檔，查閱軟體包檔案中的有效值：

- Devnet文檔：IOx包描述符
- Github儲存庫： CiscoDevNet/iox-app-template

對於此文檔，YAML配置檔案包含以下資訊：

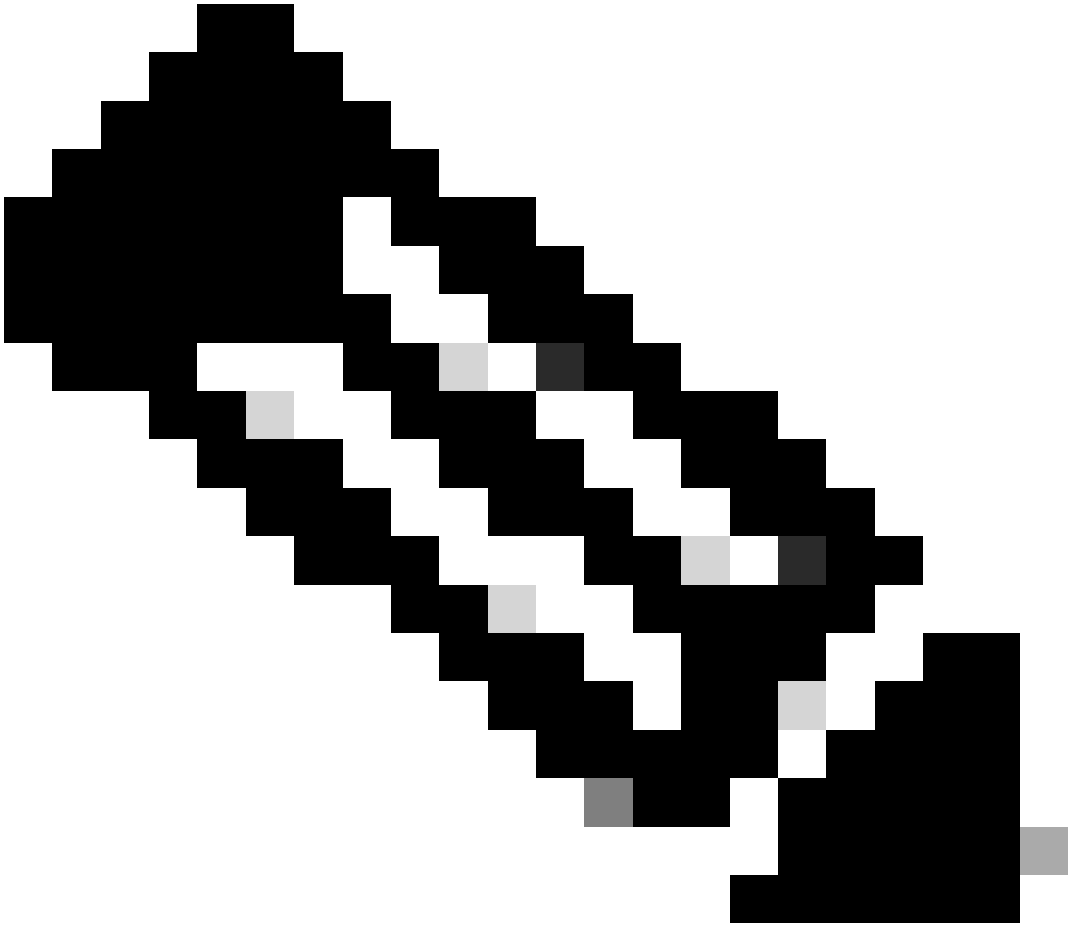
```
descriptor-schema-version: "2.2"

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
```

由於Docker Engine版本25.0和ioxclient之間不相容，推薦的方法是使用支援Docker Engine版本24.0.9或更低版本的Linux發行版，因為24.0.9版是支援與ioxclient相容的最新版本。

在本示例中，用於演示IOx客戶端功能的Docker映像是在運行版本20.04的基於Ubuntu的虛擬機器上構建的，之所以選擇此映像，是因為Docker Engine .deb二進位制檔案可用於此分發/版本。



附註：安裝舊版Docker的唯一方法是通過bin檔案安裝。這些二進位制檔案是已經準備在特定作業系統上直接運行的特定軟體版本。

組態

要使用上述規範準備VM，請繼續從舊版本的Docker安裝二進位制檔案：

```
wget https://download.docker.com/linux/ubuntu/dists/focal/pool/stable/amd64/docker-ce-cli_24.0.9-1~ubunt
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
```

安裝完所有檔案後，電腦就可以打包該iox應用程式了。

包裝程式

將python代碼和Dockerfile傳輸到虛擬機器，驗證這兩個檔案是否位於同一個目錄中，然後繼續構建Docker映像：

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

要列出本地電腦儲存庫中可用的Docker映像，請運行以下命令：

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

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

這裡有2種選擇

1 — 使用Docker映像和描述符檔案package.yaml打包應用程式

2 — 將映像匯出為根檔案系統，並將其打包為描述符YAML檔案

選項1 — 打包Docker映像和YAML檔案：

移動到要打包影象和YAML檔案的目標目錄。

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

然後，通過運行以下命令打包檔案：

```
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: 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$ |
```

此組操作負責生成程式包tar捆綁包。為了檢查程式包內容，可以使用tar實用程式對其進行解壓縮

```
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
```

選項2 — 將Docker映像匯出為根檔案系統，並將其打包為描述符YAML檔案。

在要建立映像的目錄中運行命令：

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

此命令將Docker映像匯出為包含裝載在容器上/容器內的根檔案系統的捆綁包。

將package.YAML檔案移動到指定位置。完成後，目錄結構必須如下所示：

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

最後一步是通過執行以下命令來打包Docker映像：

```
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
```

執行這些操作後，將生成檔案package.tar並為部署做好準備。要檢查軟體包的內容，請運行所示的命令：

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

安裝

準備好應用程式後，最後一步是在特權EXEC模式下運行命令，將應用程式安裝到目標裝置上，如下所示：

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

等待約1分鐘，並確認應用程式是否已成功運行：

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

關於此翻譯

思科已使用電腦和人工技術翻譯本文件，讓全世界的使用者能夠以自己的語言理解支援內容。請注意，即使是最佳機器翻譯，也不如專業譯者翻譯的內容準確。Cisco Systems, Inc. 對這些翻譯的準確度概不負責，並建議一律查看原始英文文件（提供連結）。