IOxClientを使用したIOxアプリケーションの導入

内容

はじめに

目的

前提条件

- 要件
- 使用するコンポーネント

概要

- IOxとは
- ioxclientとは何ですか。
- アプリケーションの定義
- Pythonアプリケーション
- Dockerファイルの定義
- パッケージ
- Cisco DevNet IOx Template RepositoryのYAMLの例

コンフィギュレーション

- 梱包手順
- 設置

はじめに

このドキュメントでは、ioxclientを使用したアプリケーションの導入について説明します。

目的

このドキュメントでは、ioxclientを使用したアプリケーションの導入について説明します。

このドキュメントの焦点は非常に実用的であるため、技術的な詳細が必要な場合は、共有されているドキュメントを確認することをお勧めします。

前提条件

- Cisco IOS XEおよびCisco IOSオペレーティングシステムに関する基本的な知識
- Dockerとコンテナのライフサイクルに関する深い理解
- 基本的なLinux操作。

要件

ご使用のデバイスがioxをサポートしているかどうかを確認するには、互換性マトリクス「プラットフォームサポートマトリクス」を参照してください。

また、PCの仕様に従ってioxクライアントをダウンロードしてください：ダウンロード

使用するコンポーネント

このドキュメントの情報は、次のソフトウェアとハードウェアのバージョンに基づくものです。

- ioxclientバージョン1.17.0.0
- ルータC8000v、バージョン17.12.3a
- Ubuntu Machineバージョン20.04
- Docker Engineバージョン24.0.9以前をインストールします。

このドキュメントの情報は、特定のラボ環境にあるデバイスに基づいて作成されました。このドキュメントで使用するすべてのデバイスは、クリアな（デフォルト）設定で作業を開始しています。本稼働中のネットワークでは、各コマンドによって起こる可能性がある影響を十分確認してください。

概要

IOxとは

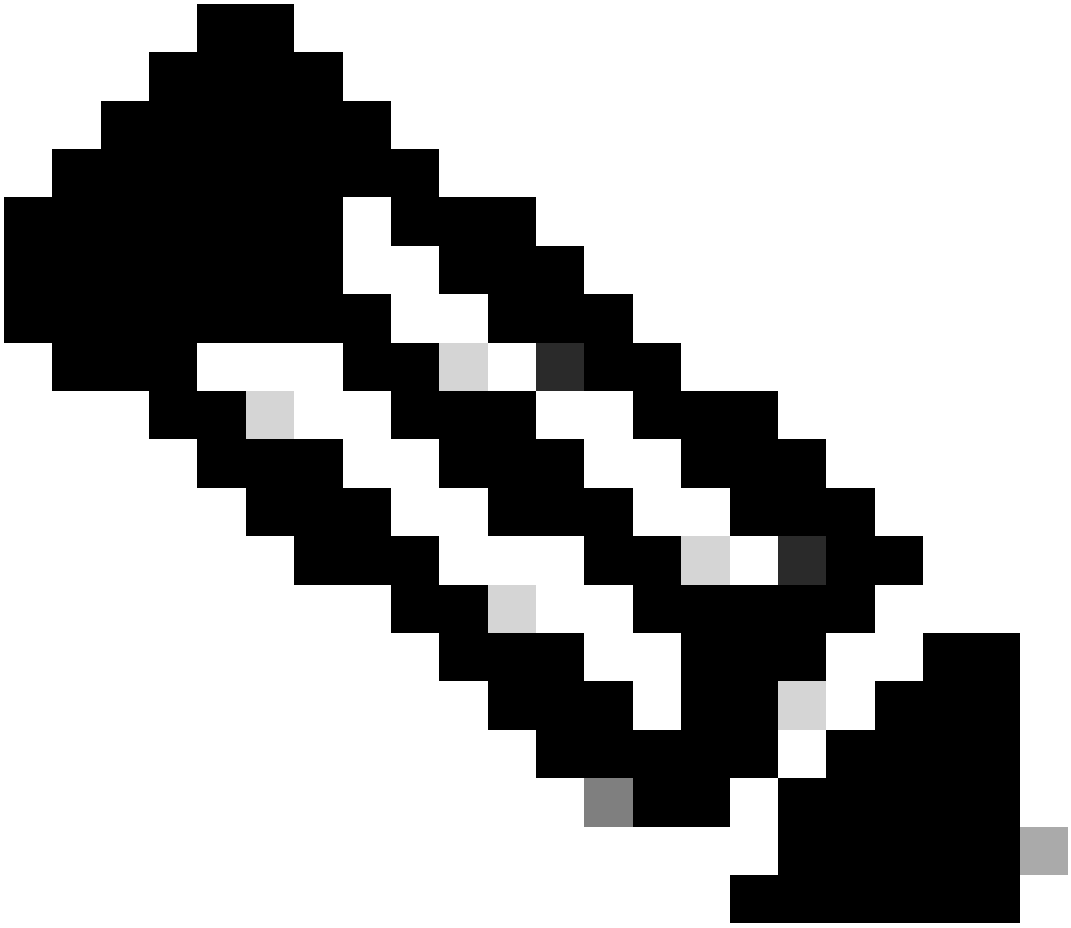
IOxはシスコデバイスのアプリケーション環境です。この機能により、ioxclientなどのツールを使用して、IOxと互換性のある形式でアプリケーションをパッケージ化できます。

ioxclientとは何ですか。

ioxclientは、Cisco IOx SDKの一部であるコマンドラインツールです。Cisco IOxデバイス上でIOxアプリケーションを開発、テスト、導入するために使用されます。

アプリケーションの定義

このサンプルコードは、ポート8000を介してリスニングする基本的なHTTPサーバを作成します。これは、Dockerイメージ構築イメージ(Dockerfile)のコア機能として機能します。



注: Dockerfileは、特定の機能を持つカスタムイメージを作成する場合にのみ必要です。アプリケーションは、コンテナリポジトリから取得してエクスポートし、ioxclientのベースとして使用できます。

Pythonアプリケーション

```
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ファイルの定義

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

このコードでは、ポート8000を介してhttpサーバがリッスンするように設定し、アプリケーションをDockerファイルにパッケージ化します。

パッケージ

アプリケーションを適切に配備するには、package.YAMLファイルを作成し、メタデータとリソース定義を設定する必要があります。

YAMLファイルは形式であり、この形式は単純な構文のために魅力的です。ファイル内では、環境変数、ポート、依存関係などのアプリケーションの側面を指定できます。

Cisco DevNet IOx Template RepositoryのYAMLの例

```
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"]
```

ドキュメントを参照して、パッケージファイルの有効な値を調べてください。

- Devnetドキュメント：IOxパッケージ記述子
- Githubリポジトリ：CiscoDevNet / iox-app-template

このドキュメントでは、YAML構成ファイルに次の情報が含まれています。

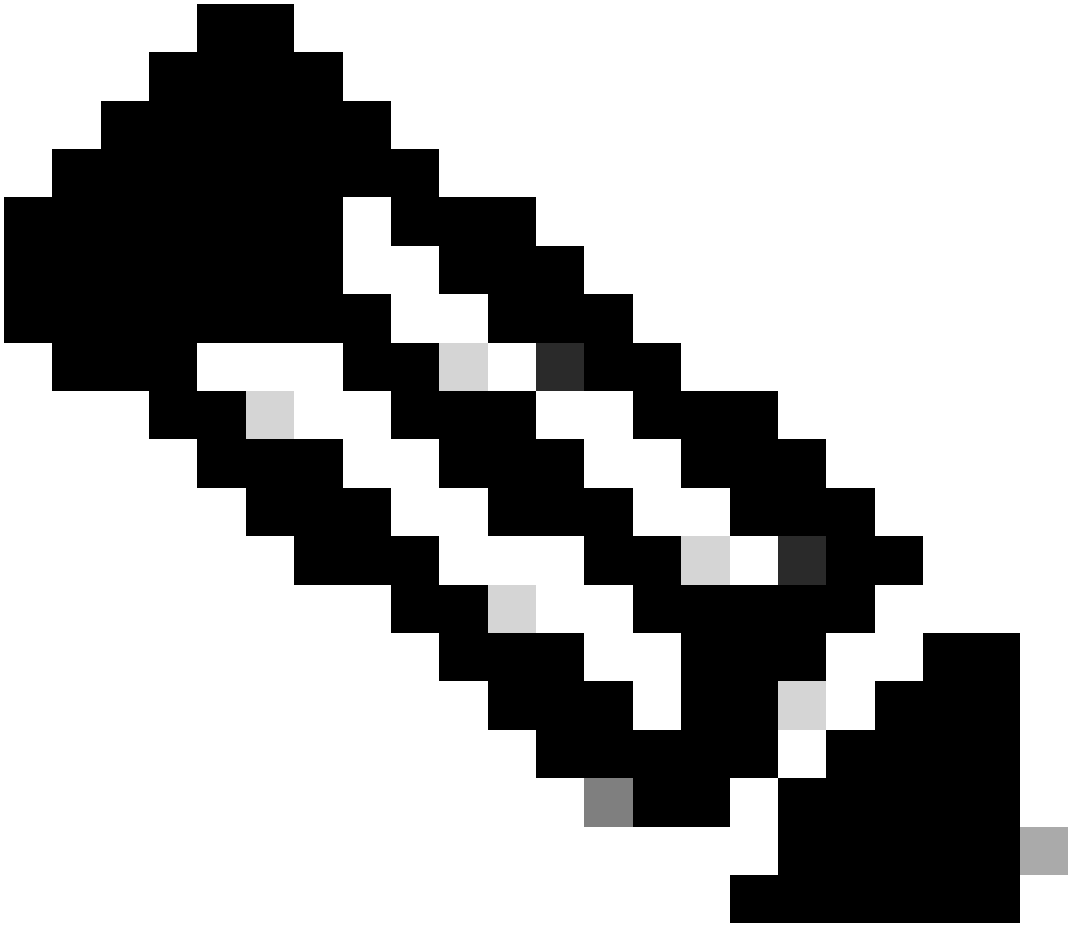
```
descriptor-schema-version: "2.2"

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
```

Docker Engineバージョン25.0とioxclientの間には互換性がないため、推奨されるアプローチは、Docker Engineバージョン24.0.9以前をサポートするLinuxディストリビューションを使用することです。これは、ioxclientとの互換性を保つために、バージョン24.0.9が最新のサポートバージョンであるためです。

この例では、IOxクライアント機能のデモンストレーションに使用するDockerイメージは、バージョン20.04を実行するUbuntuベースの仮想マシン上に構築されています。この仮想マシンは、Docker Engine .debバイナリがこの配布/バージョンで利用可能であることによって選択されています。



注：古いバージョンのDockerをインストールする唯一の方法は、binファイルを使用してインストールすることです。これらのバイナリは、特定のオペレーティングシステム上で直接実行できるように準備されているソフトウェアの特定のバージョンです。

コンフィギュレーション

前述の仕様でVMを準備するには、古いバージョンのDockerからバイナリファイルをインストールします。

```
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
```

すべてのファイルがインストールされたら、マシンはioxアプリケーションをパッケージ化する準備が整います。

梱包手順

PythonコードとDockerfileを仮想マシンに転送し、両方のファイルが同じディレクトリにあることを確認してから、Dockerイメージの構築に進みます。

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

ローカルマシンリポジトリで使用可能なDockerイメージを一覧表示するには、次に示すコマンドを実行します。

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

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

ここから2つの選択肢があります

1 - Dockerイメージと記述子ファイルpackage.yamlを使用してアプリケーションをパッケージ化します。

2 – イメージをルートファイルシステムとしてエクスポートし、記述子YAMLファイルと一緒にパックする

オプション1 - DockerイメージとYAMLファイルのパッケージ化：

イメージとYAMLファイルをパッケージ化するターゲットディレクトリに移動します。

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

次に、次のコマンドを実行してファイルをパッケージ化します。

```
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$ |
```

この一連のアクションが、パッケージtarバンドルの生成を行います。 パッケージの内容を確認するために、tarユーティリティを使用して解凍できます

```
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
```

オプション2 - Dockerイメージをルートファイルシステムとしてエクスポートし、記述子YAMLファイルでパッケージ化する。

イメージを作成するディレクトリで、次のコマンドを実行します。

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

このコマンドは、コンテナ内の/にマウントされるルートファイルシステムを含むバンドルとしてDockerイメージをエクスポートします。

package.YAMLファイルを指定した場所に移動します。完了すると、ディレクトリ構造は次のようになります。

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

最後のステップでは、次のコマンドを実行してDockerイメージをパッケージ化します。

```
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
```

これらのアクションの結果として、package.tarファイルが生成され、展開の準備が行われます。パッケージの内容を確認するには、次に示すコマンドを実行します。

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

設置

アプリケーションの準備が完了したら、次に示すように、特権EXECモードでコマンドを実行してターゲットデバイスにインストールします。

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

1分ほど待機し、アプリケーションが正常に実行されているかどうかを確認します。

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

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。