IOxClient를 사용하여 IOx 애플리케이션 구축

목차

소개

이 문서에서는 ioxclient를 사용한 애플리케이션 구축에 대해 설명합니다.

목표

이 문서에서는 ioxclient를 사용하는 애플리케이션의 구축을 이해하는 데 중점을 둡니다.

이 문서의 주안점은 전적으로 실용적이므로 더 자세한 기술적 세부 사항을 원한다면 공유된 문서를 검토하는 것이 좋습니다.

사전 요구 사항

- Cisco IOS XE 및 Cisco IOS 운영 체제에 대한 기본 지식
- Docker 및 컨테이너 라이프사이클에 대한 강력한 이해
- 기본 Linux 운영.

요구 사항

디바이스가 iox를 지원하는지 확인합니다. 호환성 매트릭스를 확인할 수 있습니다. Platform Support Matrix

또한 PC 사양에 따라 iox 클라이언트를 다운로드하십시오. 다운로드

사용되는 구성 요소

이 문서에 포함된 정보는 다음 소프트웨어 및 하드웨어 버전을 기반으로 합니다.

- ioxclient 버전 1.17.0.0
- 라우터 C8000v, 버전 17.12.3a
- Ubuntu Machine 버전 20.04
- Docker Engine 버전 24.0.9 이상을 설치합니다.

이 문서의 정보는 특정 랩 환경의 디바이스를 토대로 작성되었습니다. 이 문서에 사용된 모든 디바이스는 초기화된(기본) 컨피그레이션으로 시작되었습니다. 현재 네트워크가 작동 중인 경우 모든 명령의 잠재적인 영향을 미리 숙지하시기 바랍니다.

개요

IOx란?

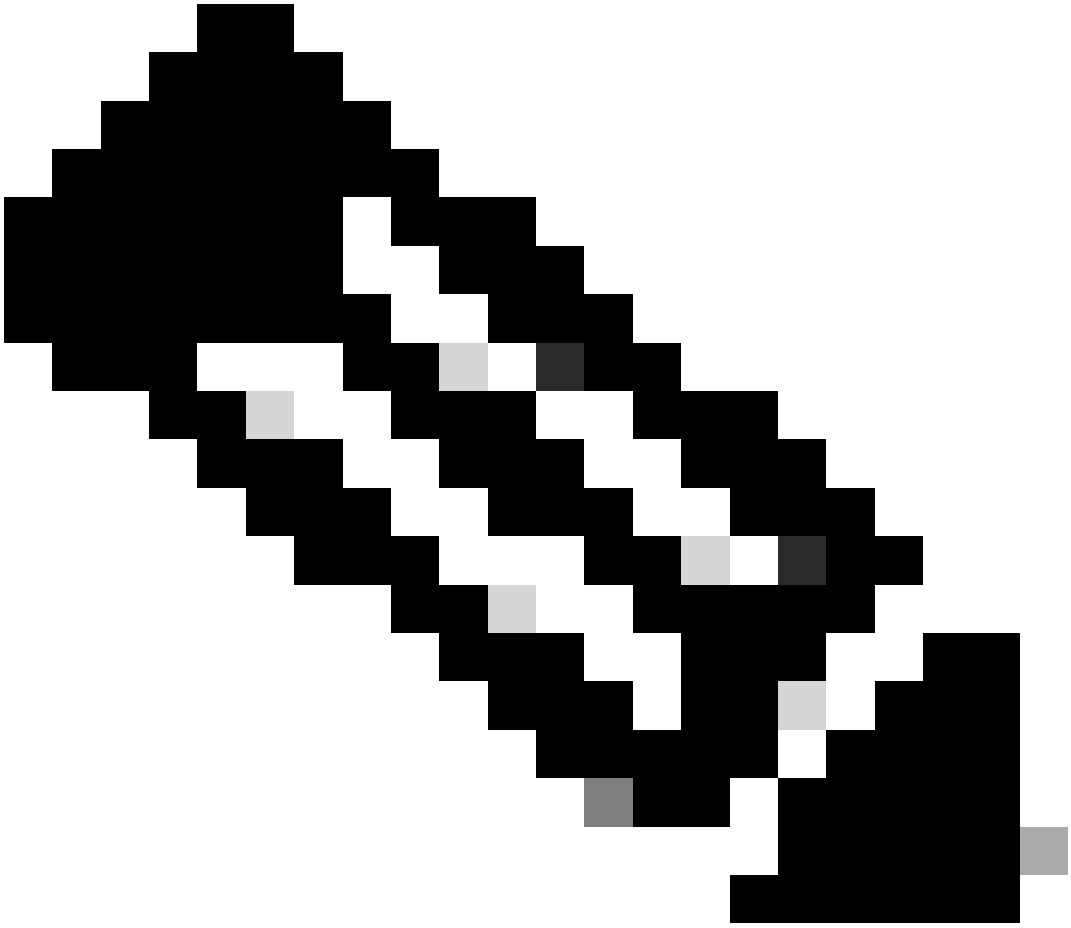
IOx는 Cisco 디바이스용 애플리케이션 환경입니다. 이 기능을 사용하면 ioxclient와 같은 툴을 사용하여 애플리케이션을 IOx와 호환되는 형식으로 패키지화할 수 있습니다.

Ioxclient란 무엇입니까?

ioxclient는 Cisco IOx SDK의 일부인 명령줄 도구입니다. Cisco IOx 디바이스에서 IOx 애플리케이션을 개발, 테스트 및 구축하는 데 사용됩니다.

애플리케이션 정의

이 예제 코드는 포트 8000을 통해 수신하는 기본 HTTP 서버를 생성합니다. Docker 이미지 빌드 이미지(Dockerfile)의 핵심 기능 역할을 합니다.



참고: Dockerfile은 특정 기능이 있는 사용자 지정 이미지를 개발하는 경우에만 필요합니다. 애플리케이션은 컨테이너 저장소에서 가져와 내보내고 ioxclient의 기반으로 사용할 수 있습니다.

Python 애플리케이션

```
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 파일 정의

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

이 코드를 사용하면 포트 8000을 통해 수신하는 http 서버를 설정하고 Docker 파일에 애플리케이션을 패키징할 수 있습니다.

패키지

응용 프로그램을 제대로 배포하려면 메타데이터 및 리소스 정의를 사용하여 package.YAML 파일을 만들고 채워야 합니다.

YAML 파일은 형식이며, 이 형식은 간단한 구문 때문에 매력적입니다. 파일 내에서 환경 변수, 포트, 종속성 등으로 응용 프로그램의 측면을 지정할 수 있습니다.

Cisco DevNet IOx 템플릿 저장소의 YAML 예

```
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"]
```

패키지 파일의 유효한 값을 확인하려면 설명서를 참조하십시오.

- Devnet 설명서: IOx 패키지 설명자
- Github 저장소: CiscoDevNet / iox-app-template

이 문서의 YAML 구성 파일에는 다음 정보가 포함되어 있습니다.

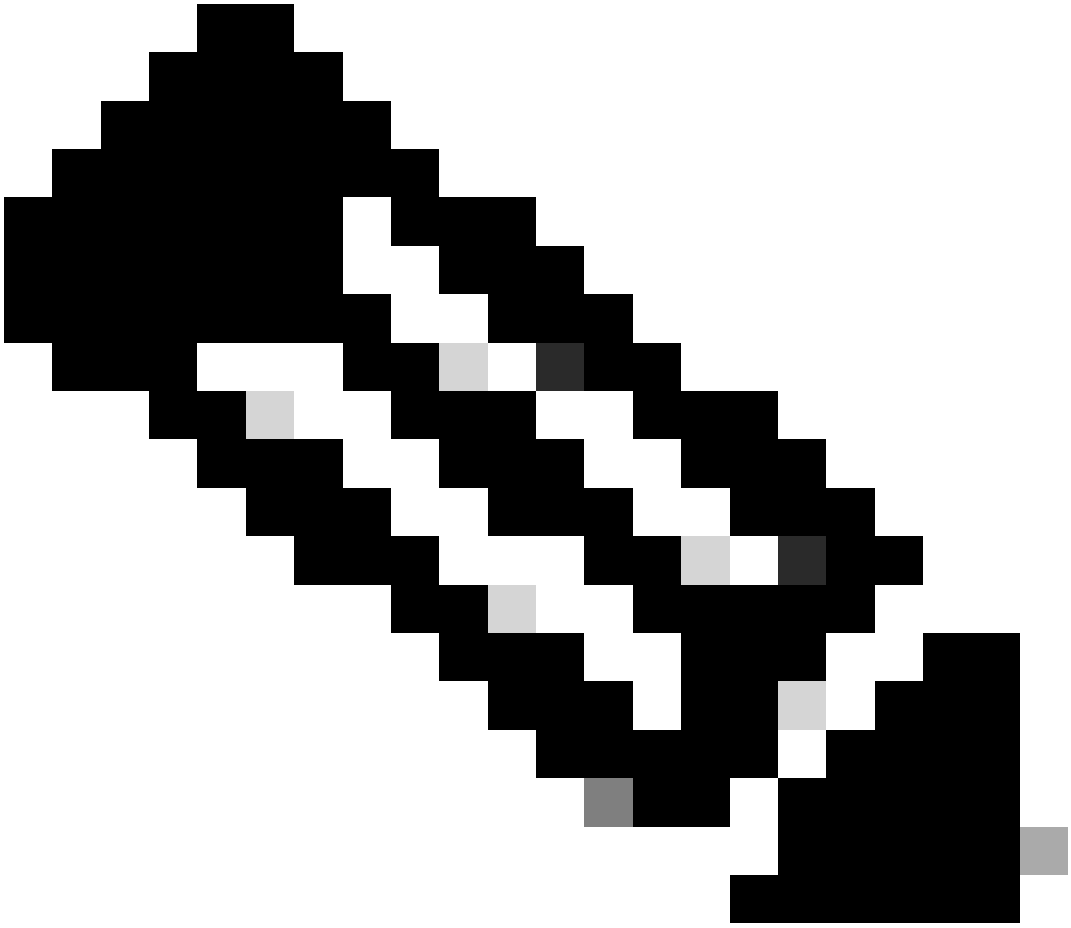
```
descriptor-schema-version: "2.2"

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 버전 25.0과 ioxclient 간의 비호환성 때문에 권장되는 접근 방식은 버전 24.0.9가 ioxclient와의 호환성을 위해 지원되는 최신 버전이므로 Docker Engine 버전 24.0.9 이하를 지원하는 Linux 배포판을 사용하는 것입니다.

이 예에서는 IOx 클라이언트 기능을 시연하는 데 사용되는 Docker 이미지를 버전 20.04를 실행하는 Ubuntu 기반 가상 머신에 구축했습니다. 이 배포/버전에서는 Docker Engine .deb 바이너리를 사용할 수 있기 때문입니다.



참고: 이전 버전의 Docker를 설치하는 유일한 방법은 bin 파일을 통해 설치하는 것입니다. 이러한 바이너리는 특정 운영 체제에서 직접 실행할 준비가 된 소프트웨어의 특정 버전입니다.

설정

위에서 설명한 사양으로 VM을 준비하려면 이전 버전의 Docker에서 이진 파일을 설치합니다.

```
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
```

모든 파일이 설치되면 시스템은 iox 애플리케이션을 패키지화할 준비가 됩니다.

포장 절차

파이썬 코드와 Dockerfile을 가상 머신으로 전송하고 두 파일이 동일한 디렉토리에 있는지 확인한 다음 Docker 이미지를 구축합니다.

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

로컬 시스템 리포지토리에서 사용할 수 있는 Docker 이미지를 나열하려면 아래에 표시된 명령을 실행합니다.

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

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

여기서부터 2가지 대안이 있습니다

1 - Docker 이미지 및 설명자 파일 package.yaml를 사용하여 응용 프로그램을 패키지합니다.

2 - 이미지를 루트 파일 시스템으로 내보내고 설명자 YAML 파일로 압축

옵션 1 - Docker 이미지 및 YAML 파일 패키징:

이미지를 패키지할 대상 디렉토리와 YAML 파일로 이동합니다.

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

그런 다음 다음 다음 명령을 실행하여 파일을 패키지합니다.

```
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: 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$ |
```

이 작업 집합은 패키지 tar 번들의 생성을 담당했습니다. 패키지 내용을 검사하기 위해 tar 유틸리티를 사용하여 압축을 풀 수 있습니다

```
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
```

옵션 2 - Docker 이미지를 루트 파일 시스템으로 내보내고 설명자 YAML 파일로 패키지합니다.

이미지를 생성할 디렉토리에서 명령을 실행합니다.

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

이 명령은 Docker 이미지를 컨테이너에 / 마운트된 루트 파일 시스템이 포함된 번들로 내보냅니다.

package.YAML 파일을 지정된 위치로 이동합니다. 완료되면 다음과 같이 디렉토리 구조가 나타나야 합니다.

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

마지막 단계에서는 다음 명령을 실행하여 Docker 이미지를 패키징합니다.

```
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
```

이러한 작업의 결과로 package.tar 파일이 생성되고 배포를 준비합니다. 패키지의 내용을 검사하려면 다음 명령을 실행합니다.

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

설치

애플리케이션이 준비된 후 마지막 단계에서는 다음과 같이 특별 권한 EXEC 모드에서 명령을 실행하여 대상 디바이스에 설치합니다.

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

약 1분 정도 기다린 후 애플리케이션이 성공적으로 실행되고 있는지 확인합니다.

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

이 번역에 관하여

Cisco는 전 세계 사용자에게 다양한 언어로 지원 콘텐츠를 제공하기 위해 기계 번역 기술과 수작업 번역을 병행하여 이 문서를 번역했습니다. 아무리 품질이 높은 기계 번역이라도 전문 번역가의 번역 결과물만큼 정확하지는 않습니다. Cisco Systems, Inc.는 이 같은 번역에 대해 어떠한 책임도 지지 않으며 항상 원본 영문 문서(링크 제공됨)를 참조할 것을 권장합니다.