Distribuire un'applicazione IOx utilizzando IOxClient

Sommario

[Introduzione](#)

[Obiettivo](#)

[Prerequisiti](#)

- [Requisiti](#)
- [Componenti usati](#)

[Panoramica](#)

- [Che cos'è IOx?](#)
- [Cos'è ioxclient?](#)
- [Definire l'applicazione](#)
- [Applicazione Python](#)
- [Definire il file Docker](#)
- [Pacchetto](#)
- [Esempio YAML dal repository dei modelli Cisco DevNet IOx](#)

[Configurazione](#)

- [Procedura di imballaggio](#)
- [Installazione](#)

Introduzione

Questo documento descrive la distribuzione di un'applicazione mediante ioxclient.

Obiettivo

Questo documento è dedicato alla comprensione della distribuzione di un'applicazione utilizzando ioxclient.

L'obiettivo di questo documento è assolutamente pratico, quindi se desiderate maggiori dettagli tecnici, vi consiglio di rivedere la documentazione condivisa.

Prerequisiti

- Conoscenze base dei sistemi operativi Cisco IOS XE e Cisco IOS.
- Conoscenza approfondita del ciclo di vita del Docker e del contenitore.
- Operazioni Linux di base.

Requisiti

Se il dispositivo supporta iox, verificare la matrice di compatibilità: matrice di supporto della piattaforma

Inoltre, si prega di scaricare il client iox secondo le specifiche del PC: Download

Componenti usati

Le informazioni fornite in questo documento si basano sulle seguenti versioni software e hardware:

- ioxclient versione 1.17.0.0
- Router C800v, versione 17.12.3a
- Computer Ubuntu versione 20.04
- Installare Docker Engine versione 24.0.9 o precedente.

Le informazioni discusse in questo documento fanno riferimento a dispositivi usati in uno specifico ambiente di emulazione. Su tutti i dispositivi menzionati nel documento la configurazione è stata ripristinata ai valori predefiniti. Se la rete è operativa, valutare attentamente eventuali conseguenze derivanti dall'uso dei comandi.

Panoramica

Che cos'è IOx?

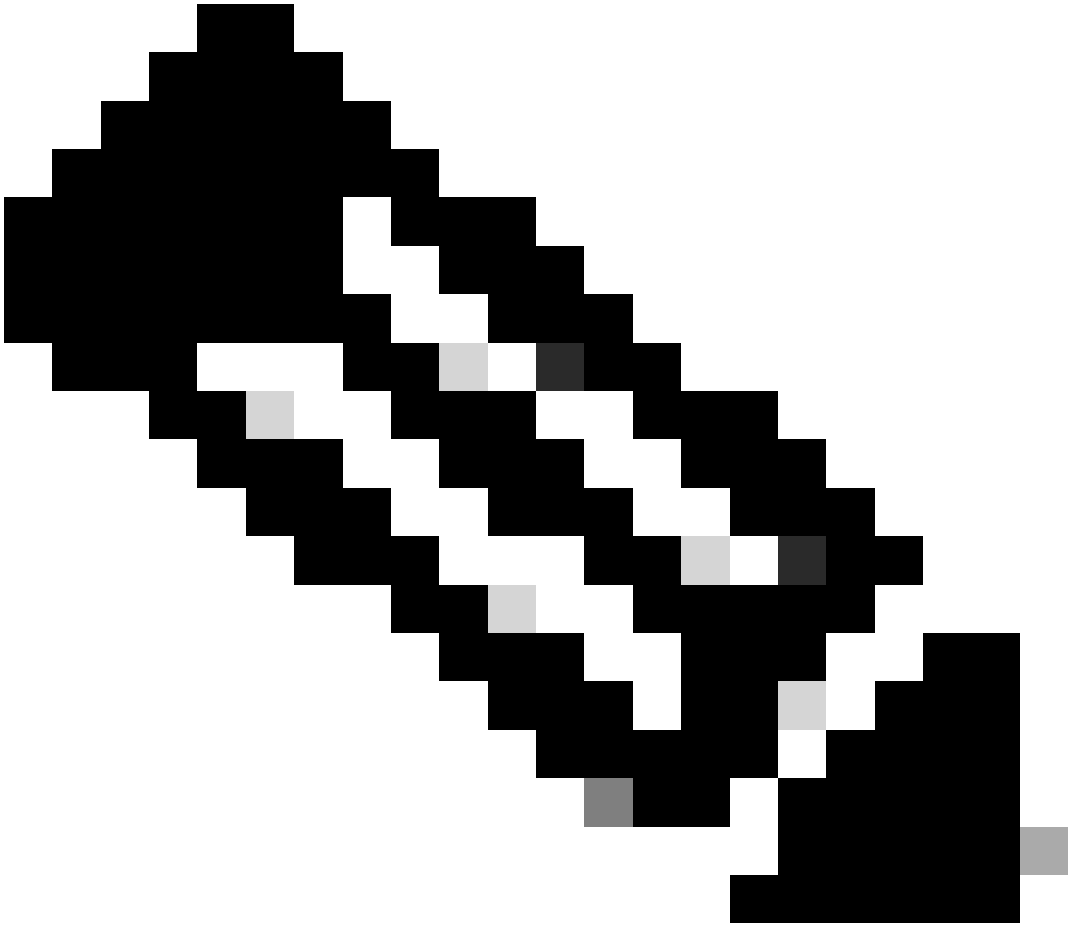
IOx è l'ambiente applicativo per i dispositivi Cisco, questa funzionalità ci consente di assemblare le applicazioni in un formato compatibile con IOx, utilizzando strumenti come ioxclient.

Cos'è ioxclient?

ioxclient è uno strumento da riga di comando che fa parte di Cisco IOx SDK. Viene utilizzato per sviluppare, testare e distribuire applicazioni IOx su dispositivi Cisco IOx.

Definire l'applicazione

Questo codice di esempio crea un server HTTP di base in ascolto tramite la porta 8000; Funge da funzionalità di base dell'immagine Docker (Dockerfile).



Nota: Un Dockerfile è necessario solo per lo sviluppo di immagini personalizzate con funzionalità specifiche. È possibile estrarre un'applicazione da un repository contenitore, esportarla e utilizzarla come base per ioxclient.

Applicazione 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()
```

Definire il file Docker

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

Con questo codice si imposta un server http in ascolto attraverso la porta 8000 e si raccoglie l'applicazione in un file Docker.

Pacchetto

È necessario creare e popolare un file package.YAML con metadati e definizioni di risorse per una corretta distribuzione dell'applicazione.

Il file YAML è un formato, questo formato è attraente a causa della sintassi semplice, all'interno del file possiamo specificare aspetti dell'applicazione come variabili ambientali, porte, dipendenze e così via.

Esempio YAML dal repository dei modelli Cisco DevNet IOx

```
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"]
```

Consultare la documentazione per consultare i valori validi nel file del pacchetto:

- Documentazione del dispositivo: descrittore pacchetto IOx
- Repository Github: Cisco DevNet/iox-app-template

Per questo documento, il file di configurazione YAML contiene le seguenti informazioni:

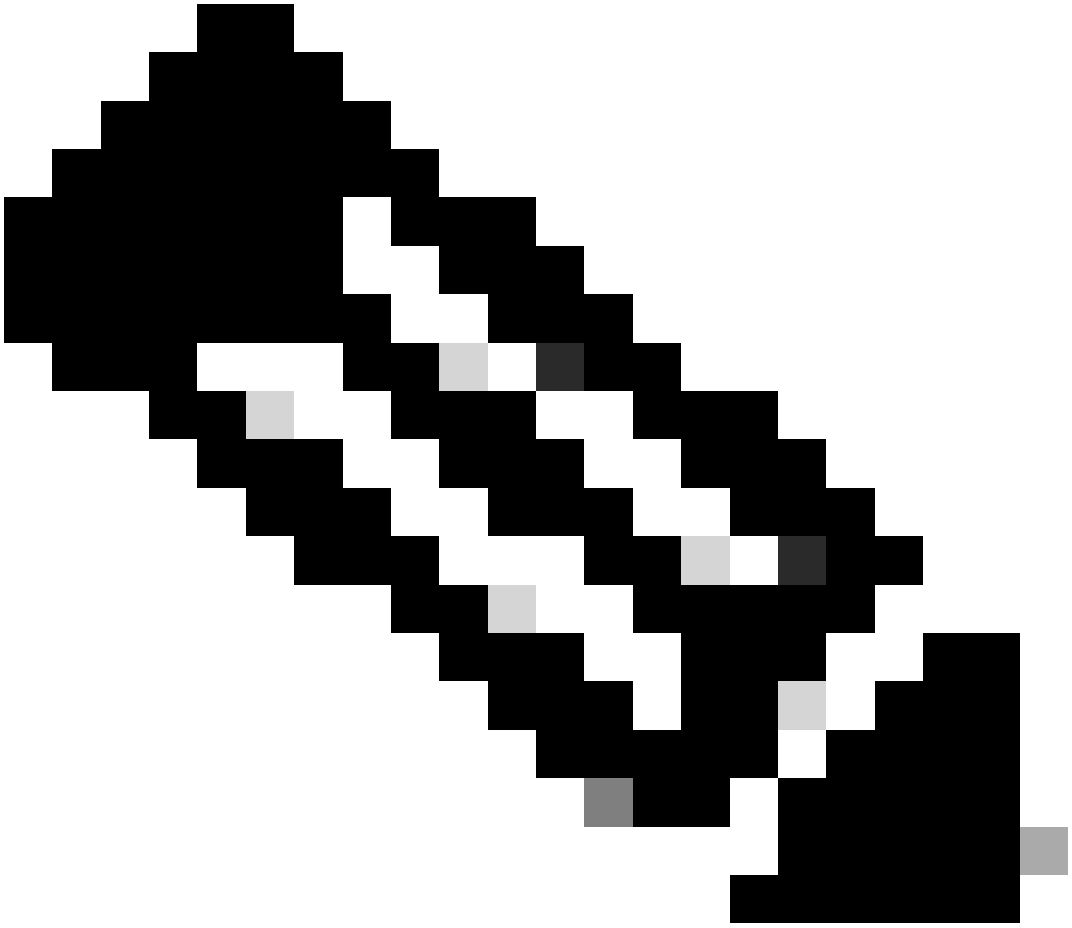
```
descriptor-schema-version: "2.2"

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
```

A causa di un'incompatibilità tra Docker Engine versione 25.0 e ioxclient, si consiglia di utilizzare una distribuzione Linux che supporti Docker Engine versione 24.0.9 o precedente, poiché la versione 24.0.9 è la più recente versione supportata per la compatibilità con ioxclient.

In questo esempio, l'immagine Docker utilizzata per dimostrare la funzionalità del client IOx è stata creata su una macchina virtuale basata su Ubuntu che esegue la versione 20.04, scelta in modo specifico perché i file binari .deb di Docker Engine sono disponibili per questa distribuzione/versione.



Nota: L'unico modo per installare le versioni precedenti di Docker consiste nell'installarle tramite i file bin. Questi file binari sono versioni specifiche del software già predisposte per l'esecuzione diretta su un particolare sistema operativo.

Configurazione

Per preparare la VM con le specifiche indicate, procedere con l'installazione del file binario da una versione precedente di 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 volta installati tutti i file, il computer è pronto per creare il pacchetto dell'applicazione iox.

Procedura di imballaggio

Trasferire il codice Python e il file Docker nella macchina virtuale, verificare che entrambi i file si trovino nella stessa directory, quindi procedere con la creazione dell'immagine Docker:

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

Per elencare le immagini Docker disponibili nel repository del computer locale, eseguire il comando riportato di seguito:

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

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

Da qui ci sono 2 alternative

1 - Creare il pacchetto dell'applicazione utilizzando l'immagine Docker e il file descrittore package.yaml

2 - Esportare l'immagine come file system radice e comprimerla con il file descrittore YAML

Opzione 1 - Creazione del package dell'immagine Docker e del file YAML:

Spostarsi nella directory di destinazione in cui si desidera collocare l'immagine e il file YAML.

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

Creare quindi il pacchetto del file eseguendo questo 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$ |
```

Questo insieme di azioni è stato responsabile della generazione del pacchetto tar. Per esaminare il contenuto del pacchetto, è possibile decomprimerlo utilizzando l'utilità 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
```

Opzione 2 - Esportazione dell'immagine Docker come file system radice e relativo pacchetto con il file YAML descrittore.

Eseguire il comando nella directory in cui deve essere creata l'immagine:

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

Questo comando esporta l'immagine Docker come un bundle contenente il file system radice che si installa su / all'interno del contenitore.

Spostare il file package.YAML nella posizione specificata. Al termine, la struttura della directory deve essere simile a quella mostrata di seguito:

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

Il passaggio finale prevede il packaging dell'immagine Docker mediante l'esecuzione di questo 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
```

A seguito di queste azioni, il file package.tar viene generato e preparato per la distribuzione. Per esaminare il contenuto del pacchetto, eseguire il comando riportato di seguito:

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

Installazione

Dopo aver preparato l'applicazione, il passaggio finale consiste nell'installarla sul dispositivo di destinazione eseguendo il comando in modalità di esecuzione privilegiata, come mostrato:

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

Attendere circa 1 minuto e verificare se l'applicazione è in esecuzione correttamente:

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

Informazioni su questa traduzione

Cisco ha tradotto questo documento utilizzando una combinazione di tecnologie automatiche e umane per offrire ai nostri utenti in tutto il mondo contenuti di supporto nella propria lingua. Si noti che anche la migliore traduzione automatica non sarà mai accurata come quella fornita da un traduttore professionista. Cisco Systems, Inc. non si assume alcuna responsabilità per l’accuratezza di queste traduzioni e consiglia di consultare sempre il documento originale in inglese (disponibile al link fornito).