# Implantar um aplicativo IOx usando IOxClient

## Contents


Introdução

Objetivo

Pré-requisitos

- Requisitos
- Componentes Utilizados

Overview

- O que é IOx?
- O que é o ioxclient?
- Definir o aplicativo
- Aplicativo Python
- Definir o arquivo Docker
- Pacote
- Exemplo de YAML do Repositório de Modelos do Cisco DevNet IOx

Configuração

- Procedimento de embalagem
- Instalação


## Introdução

Este documento descreve a implantação de um aplicativo usando o ioxclient.

## Objetivo

Este documento é dedicado a entender a implantação de um aplicativo usando o ioxclient.

O foco deste documento é totalmente prático, portanto, se você quiser mais detalhes técnicos, recomendo a revisão da documentação compartilhada.

## Pré-requisitos

- Conhecimento básico dos sistemas operacionais Cisco IOS XE e Cisco IOS.
- Forte compreensão do ciclo de vida do Docker e do contêiner.
- Operações básicas do Linux.

### Requisitos

Confirme se o seu dispositivo suporta iox, você pode verificar a matriz de compatibilidade: Matriz de suporte da plataforma

Além disso, faça o download do cliente iox de acordo com suas especificações de PC: Downloads

### Componentes Utilizados

As informações contidas neste documento são baseadas nestas versões de software e hardware:

- ioxclient Versão 1.17.0.0
- Roteador C8000v, Versão 17.12.3a
- Máquina Ubuntu versão 20.04
- Instale o mecanismo Docker versão 24.0.9 ou anterior.

As informações neste documento foram criadas a partir de dispositivos em um ambiente de laboratório específico. Todos os dispositivos utilizados neste documento foram iniciados com uma configuração (padrão) inicial. Se a rede estiver ativa, certifique-se de que você entenda o impacto potencial de qualquer comando.

## Overview

### O que é IOx?

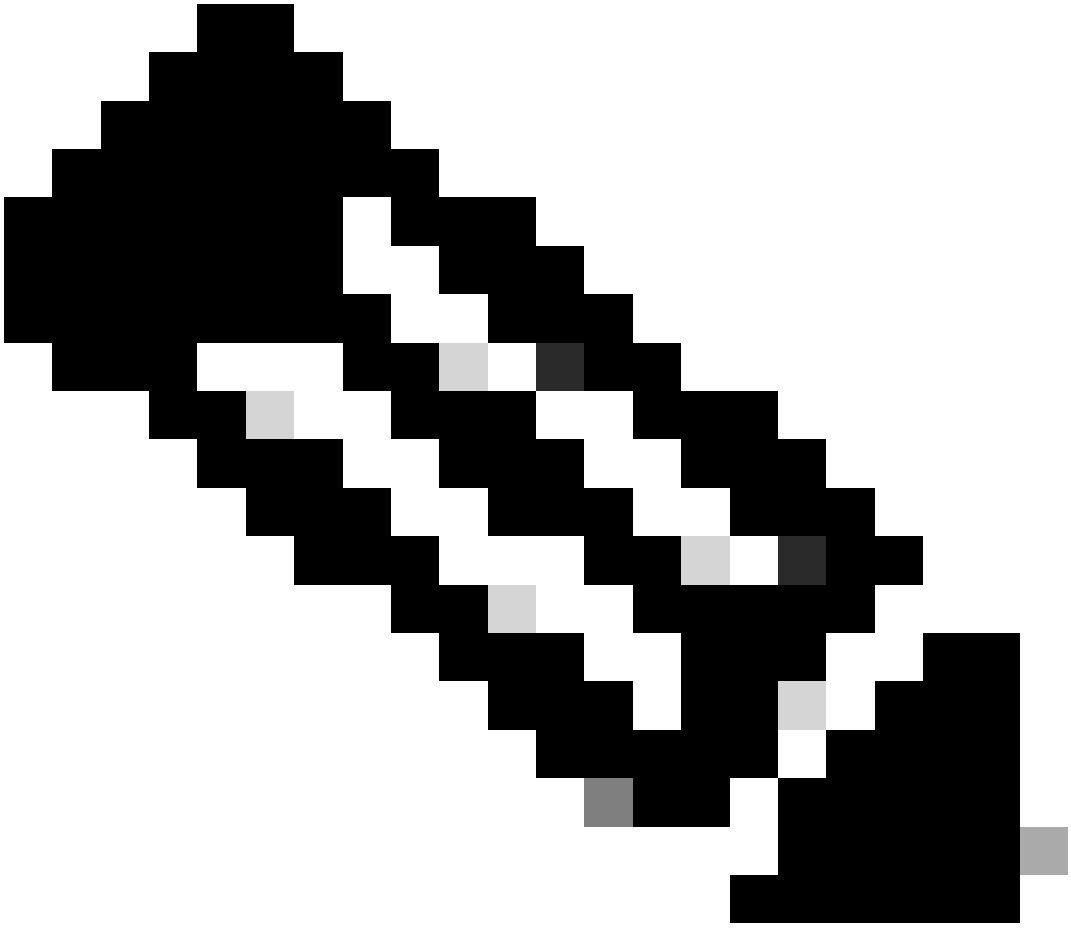
O IOx é o ambiente de aplicativos para dispositivos Cisco, esse recurso nos permite empacotar os aplicativos em um formato compatível com o IOx, usando ferramentas como o ioxclient.

### O que é o ioxclient?

ioxclient é uma ferramenta de linha de comando que faz parte do Cisco IOx SDK. É usado para desenvolver, testar e implantar aplicativos IOx em dispositivos Cisco IOx.

### Definir o aplicativo

Este código de exemplo cria um servidor HTTP básico que ouve através da porta 8000; Ele serve como a funcionalidade principal da imagem de construção Docker (Dockerfile).



Note: Um Dockerfile só é necessário ao desenvolver imagens personalizadas com funcionalidade específica. Um aplicativo pode ser extraído de um repositório de contêineres, exportado e usado como base para ioxclient.

## Aplicativo 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 o arquivo Docker

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

Com esse código, você define um servidor http escutando pela porta 8000 e empacota o aplicativo em um arquivo Docker.

## Pacote

É necessário criar e preencher um arquivo package.YAML com metadados e definições de recursos para a implantação adequada do aplicativo.

O arquivo YAML é um formato, este formato é atraente devido à sintaxe simples, dentro do arquivo podemos especificar aspectos do aplicativo como variáveis ambientais, portas, dependências e assim por diante.

## Exemplo de YAML do Repositório de Modelos do 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"]
```

Consulte a documentação para consultar os valores válidos no arquivo de pacote:

- Documentação do Devnet: Descritor de Pacote IOx
- Repositório Github: CiscoDevNet / modelo de aplicativo iox

Para este documento, o arquivo de configuração YAML contém as informações abaixo:

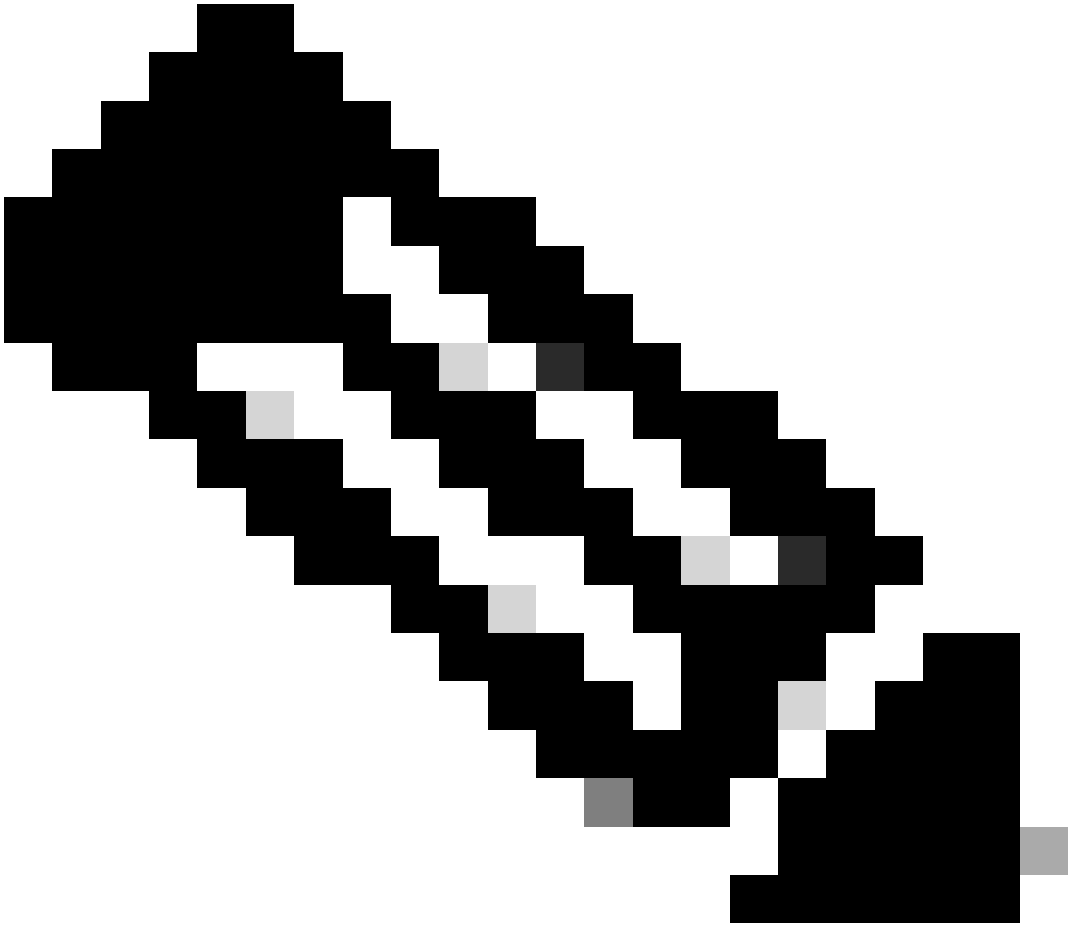
```
descriptor-schema-version: "2.2"

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
```

Devido a uma incompatibilidade entre o Docker Engine versão 25.0 e ioxclient, a abordagem recomendada é usar uma distribuição Linux que suporte o Docker Engine versão 24.0.9 ou anterior, como a versão 24.0.9 é a versão mais recente suportada para compatibilidade com ioxclient.

Neste exemplo, a imagem do Docker usada para demonstrar a funcionalidade do cliente IOx foi criada em uma máquina virtual baseada em Ubuntu executando a versão 20.04, escolhida especificamente porque os binários .deb do Docker Engine estão disponíveis para essa distribuição/versão.



Note: A única maneira de instalar versões mais antigas do Docker é instalá-lo através dos arquivos bin. Esses binários são versões específicas do software que já estão preparadas para serem executadas diretamente em um sistema operacional específico.

## Configuração

Para preparar a VM com as especificações mencionadas, continue para instalar o arquivo binários de uma versão antiga do 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
```

Depois que todos os arquivos estiverem instalados, a máquina estará pronta para empacotar o aplicativo iox.

## Procedimento de embalagem

Transfira o código python e o arquivo Docker para a máquina virtual, verifique se ambos os arquivos estão no mesmo diretório e continue para criar a imagem Docker:

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

Para listar as imagens do Docker disponíveis no repositório da máquina local, execute o comando mostrado abaixo:

```
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 daqui, há 2 alternativas

1 - Empacotar o aplicativo usando a imagem Docker e o arquivo de descrição package.yaml

2 - Exportar a imagem como um sistema de arquivos raiz e compactá-la com o arquivo YAML do descritor

Opção 1 - Empacotando a imagem do Docker e o arquivo YAML:

Vá para o diretório de destino onde deseja empacotar a imagem e o arquivo YAML.

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

Em seguida, empacote o arquivo executando 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 ações foi responsável pela geração do pacote tar. Para inspecionar o conteúdo

do pacote, podemos descompactá-lo usando o utilitário 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
```

Opção 2 - Exportar a imagem do Docker como um sistema de arquivos raiz e empacotar com o arquivo YAML do descritor.

Execute o comando no diretório onde a imagem será criada:

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

Este comando exporta a imagem do Docker como um pacote que contém o sistema de arquivos raiz que monta em / dentro do contêiner.

Mova o arquivo package.YAML para o local especificado. Uma vez concluída, a estrutura do diretório deve parecer-se com o exibido:

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

A etapa final envolve empacotar a imagem do Docker executando 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 dessas ações, o arquivo package.tar é gerado e preparado para implantação. Para examinar o conteúdo do pacote, execute o 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
```

## Instalação

Depois que o aplicativo tiver sido preparado, a etapa final envolve instalá-lo no dispositivo de destino executando o comando no modo EXEC privilegiado como mostrado:

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

Aguarde cerca de 1 minuto e confirme se o aplicativo está sendo executado com êxito:

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

Sobre esta tradução
A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês (link fornecido) seja sempre consultado.

Sobre esta tradução
A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês (link fornecido) seja sempre consultado.