Deploy an IOx Application Using IOxClient

Contents

Introduction

Objective

Prerequisites

- Requirements
- Components Used

Overview

- What is IOx?
- What is ioxclient?
- Define the Application
- Python Application
- Define the Docker File
- Package
- YAML Example from the Cisco DevNet IOx Template Repository

Configuration

- Packaging Procedure
- Installation

Introduction

This document describes the deployment of an application using ioxclient.

Objective

This document is dedicated to understanding the deployment of an application using ioxclient.

The focus of this document is entirely practical, so if you want more technical details, I recommend reviewing the documentation shared.

Prerequisites

- Basic knowledge of the Cisco IOS XE and Cisco IOS operating systems.
- Strong understanding of Docker and container lifecycle.
- Basic Linux operations.

Requirements

Confirm if your device supports iox, you can check the compatibility matrix: Platform Support Matrix

Also, please download iox client according to your PC specifications: Downloads

Components Used

The information contained in this document is based on these software and hardware versions:

- ioxclient Version 1.17.0.0
- Router C8000v, Version 17.12.3a
- Ubuntu Machine Version 20.04
- Install Docker Engine version 24.0.9 or older.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Overview

What is IOx?

IOx is the application environment for Cisco devices, this feature allows us to package the applications in a format that is compatible with IOx, using tools like ioxclient.

What is ioxclient?

ioxclient is a command-line tool that is part of the Cisco IOx SDK. It is used to develop, test, and deploy IOx applications on Cisco IOx devices.

Define the Application

This example code creates a basic HTTP server listening through port 8000; It serves as the core functionality of the Docker image build image (Dockerfile).

Note: A Dockerfile is only required when developing custom images with specific functionality. An application can be pulled from a container repository, exported and used as base for ioxclient.

Python Application

```
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()
```

Define the Docker File

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

With this code you set a http server listening through port 8000, and package the application in a Docker file.

Package

It is required to create and populate a package.YAML file with metadata and resource definitions for proper deployment of the application.

The YAML file is a format, this format is attractive due to the simple syntax, within the file we can specify aspects of the application as environmental variables, ports, dependencies and so on.

YAML Example from the Cisco DevNet IOx Template Repository

```
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"]
```

Please refer to the documentation to consult the valid values in the package file:

- **Devnet documentation:** IOx Package Descriptor
- **Github repository:** CiscoDevNet / iox-app-template

For this document, the YAML configuration file contains the information below:

```
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
```

Due to an incompatibility between Docker Engine version 25.0 and ioxclient, the recommended approach is to use a Linux distribution that supports Docker Engine version 24.0.9 or earlier, as version 24.0.9 is the latest supported version for compatibility with ioxclient.

In this example, the Docker image used to demonstrate IOx Client functionality was built on an Ubuntu-based virtual machine running version 20.04, chosen specifically because the Docker Engine .deb binaries are available for this distribution/version.

Note: The only way to install older versions of Docker is to install it through the bin files. These binaries are specific versions of the software that are already prepared to run directly on a particular operating system.

Configuration

To prepare the VM with the specifications mentioned, proceed to install the binaries file from an old version of 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
```

Once all the files are installed, the machine is ready to package the iox application.

Packaging Procedure

Transfer the python code, and the Dockerfile to the virtual machine, verify both files are in the same directory, then proceed to build the Docker image:

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

To list the Docker images available in the local machine repository, run the command shown below:

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

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

From here there are 2 alternatives

1 - Package the application using the Docker image and the descriptor file package.yaml

2 - Export the image as a root file system and pack it with the descriptor YAML file

Option 1 – Packaging the Docker image and the YAML file:

Move to the target directory where you desire to package the image, and the YAML file.

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

Then, package the file by running this command:

```
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$ |
```

This set of actions were responsible for the generation of the package tar bundle. In order to inspect the package contents, we can decompress it by using the tar utility

```
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
```

Option 2 - Exporting the Docker image as a root file system and package it with the descriptor YAML file.

Run the command in the directory where the image is to be created:

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

This command exports the Docker image as a bundle containing the root filesystem that mounts on / within the container.

Move the package.YAML file to the specified location. Once completed, the directory structure must resemble appear as shown:

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

The final step involves packaging the Docker image by executing this command:

```
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
```

As a result of these actions, the file package.tar is generated and prepared for deployment. To examine the contents of the package, run the command shown:

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

Installation

After the application has been prepared, the final step involves installing it on the target device by running the command in privileged EXEC mode as shown:

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

Wait around 1 minute and confirm if the application is running successfully:

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