使用IOx客户端部署IOx应用

目录

简介

本文档介绍如何使用ioxclient部署应用。

目标

本文档专门用于了解使用ioxclient的应用的部署。

本文档的重点非常实用，因此，如果您需要更多技术详细信息，我建议查看共享的文档。

先决条件

- Cisco IOS XE和Cisco IOS操作系统的基础知识。
- 深入了解Docker和容器生命周期。
- 基本的Linux操作。

要求

确认您的设备是否支持iox，您可以查看兼容性矩阵：平台支持矩阵

此外，请根据您的PC规格下载iox客户端：下载

使用的组件

本文档中包含的信息基于以下软件和硬件版本：

- ioxclient版本1.17.0.0
- 路由器C8000v，版本17.12.3a
- Ubuntu计算机版本20.04
- 安装Docker引擎24.0.9版或更高版本。

本文档中的信息都是基于特定实验室环境中的设备编写的。本文档中使用的所有设备最初均采用原始（默认）配置。如果您的网络处于活动状态，请确保您了解所有命令的潜在影响。

概述

什么是IOx?

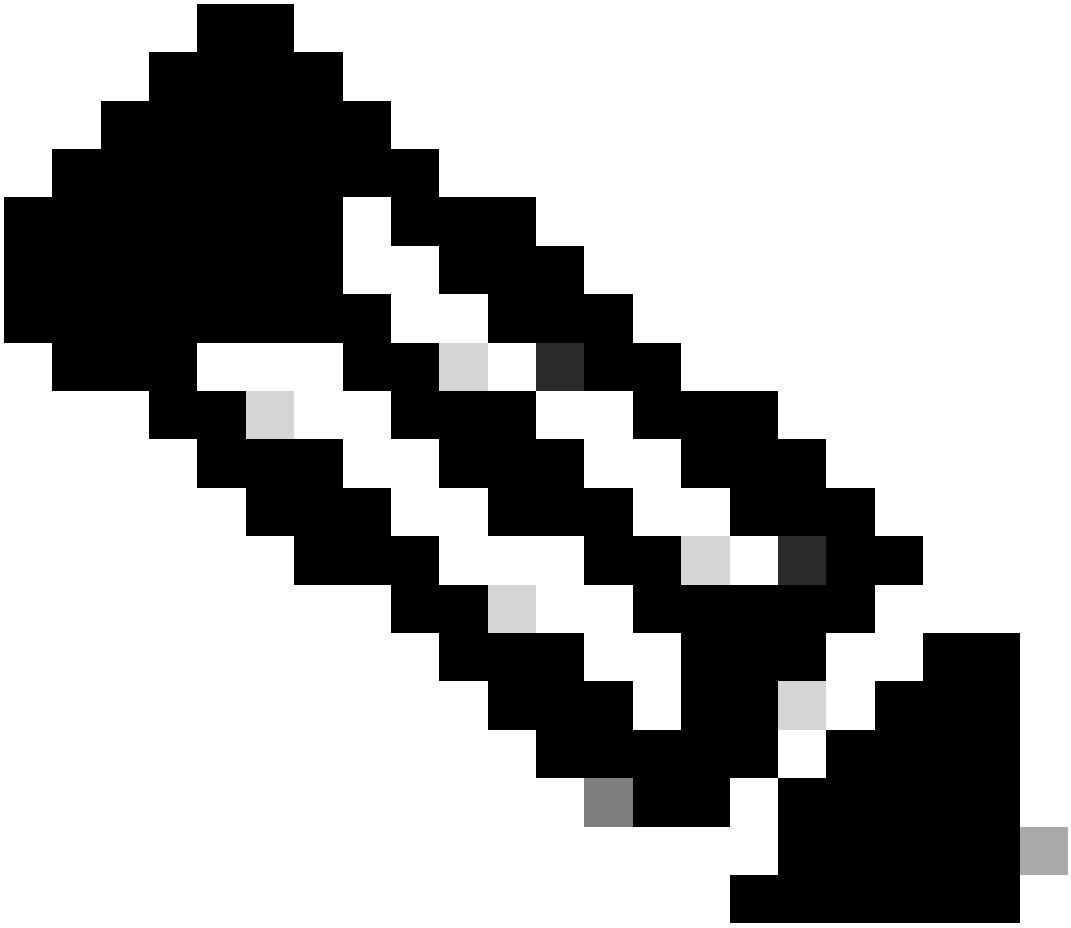
IOx是思科设备的应用环境，此功能允许我们使用ioxclient等工具以与IOx兼容的格式打包应用。

什么是ioxclient?

ioxclient是一个命令行工具，是Cisco IOx SDK的一部分。它用于在思科IOx设备上开发、测试和部署IOx应用。

定义应用程序

此示例代码创建通过端口8000进行侦听的基本HTTP服务器；它用作Docker映像构建映像(Dockerfile)的核心功能。



注意：只有在开发具有特定功能的自定义映像时，才需要Dockerfile。应用程序可以从容器存储库中提取，导出并用作ioxclient的基础。

Python应用

```
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()
```

定义Docker文件

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

使用此代码，您可以设置http服务器通过端口8000进行侦听，并将应用程序打包到Docker文件中。

包

需要使用元数据和资源定义创建和填充package.YAML文件，才能正确部署应用。

YAML文件是一种格式，此格式由于语法简单而颇具吸引力，在文件中我们可以指定应用的各个方面，如环境变量、端口、依赖项等。

来自Cisco DevNet IOx模板存储库的YAML示例

```
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"]
```

请参考文档，查阅软件包文件中的有效值：

- Devnet文档：IOx包描述符
- Github存储库： CiscoDevNet / iox-app-template

对于本文档，YAML配置文件包含以下信息：

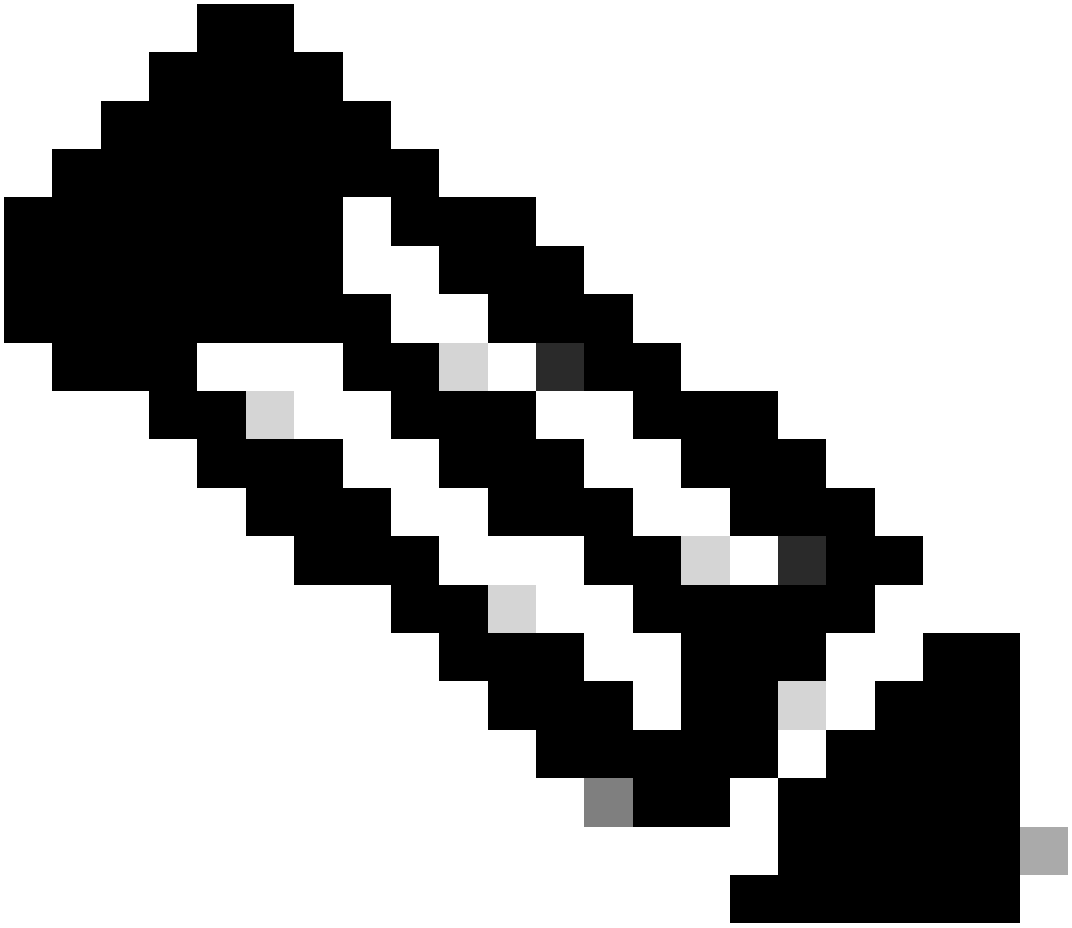
```
descriptor-schema-version: "2.2"

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
```

由于Docker引擎25.0版和ioxclient之间不兼容，推荐的方法是使用支持Docker引擎24.0.9版或更早版本的Linux发行版，因为24.0.9版是与ioxclient兼容的最新受支持版本。

在本示例中，用于演示IOx客户端功能的Docker映像构建于运行版本20.04的基于Ubuntu的虚拟机上，之所以选择此映像是因为此分发/版本可以使用Docker引擎.deb二进制文件。



注意：安装旧版本Docker的唯一方法是通过bin文件安装。这些二进制文件是已经准备在特定操作系统上直接运行的软件的特定版本。

配置

要使用上述规范准备VM，请继续从旧版本的Docker安装二进制文件：

```
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
```

安装完所有文件后，计算机就可以打包该iox应用程序了。

包装程序

将python代码和Dockerfile传输到虚拟机，验证这两个文件是否位于同一目录中，然后继续构建Docker映像：

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

要列出本地计算机存储库中可用的Docker映像，请运行如下所示的命令：

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

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

这里有2个替代方案

1 — 使用Docker映像和描述符文件package.yaml打包应用程序

2 — 将映像导出为根文件系统，并将其打包为描述符YAML文件

选项1 — 打包Docker映像和YAML文件：

移动到要打包映像和YAML文件的目标目录。

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

然后，通过运行以下命令打包文件：

```
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 1Ox 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/tesO/.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: 5Oc922f1O3ddcO1a5dc7a98d6cacefb167f4a2c692dfc521231bb42fOc3dcf55 Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211630719
Path: artifacts.mf
SHA256: 511OO8aa2d1418daf1770768fb79c9Of16814ff7789dO3beb4f4ea1bf4fae8f2 Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211634941
Path: artifacts.tar.gz
SHA256: Occ3f69af5Ocf0aO1ec9a14OOc44Of6OaOdff55369bd3O9b6dfc69715302425+ Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211634952
Path: envelope_package.tar.gz
SHA256: d492deO9441a241f879cd268cd1b3424ee79a58a9495aa77ae5b11cab2fd55da Timestamp before SHA256: 17482
Timestamp after SHA256: 1748211634963
Path: package.yaml
SHA256: d8dc7253443ff3adO8Oc42bc8d82db4c3ea7ae9b2dOe2f827fbaf2bc8O245f62 Generated package manifest at
Generating IOx Package..
Package docker image tac_app at /home/ubuntu/tesO/package.tar
ubuntu@ip-172-31-3O-249:~/tesO$ |
```

该组操作负责生成程序包tar包。为了检查程序包内容，可以使用tar实用程序将其解压缩

```
ubuntu@ip-172-31-3O-249:~/tesO$ tar -tf package.tar
package.yaml
artifacts.mf
.package.metadata
package.mf
```

```
envelope_package.tar.gz
artifacts.tar.gz
```

选项2 — 将Docker映像导出为根文件系统并将其打包为描述符YAML文件。

在要创建映像的目录中运行命令：

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

此命令将Docker映像导出为包含根文件系统的捆绑包，该根文件系统装载在容器上/容器内。

将package.YAML文件移动到指定位置。完成后，目录结构必须如下所示：

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

最后一步是通过执行以下命令打包Docker映像：

```
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
```

这些操作会生成文件package.tar并准备进行部署。要检查软件包的内容，请运行所示的命令：

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

安装

在准备好应用后，最后一步是在特权EXEC模式下运行命令在目标设备上安装应用，如下所示：

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

等待约1分钟，并确认应用是否已成功运行：

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

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。