

IR1101 ARMアーキテクチャ用のDocker IOxパッケージの構築と導入

内容

[概要](#)

[前提条件](#)

[要件](#)

[使用するコンポーネント](#)

[背景説明](#)

[設定](#)

[第1部IR1101用IOxパッケージの構築](#)

[1. LinuxホストへのIOxクライアントのインストールと準備](#)

[2. LinuxビルドマシンでのDocker環境のインストールと準備](#)

[3. QEMUユーザエミュレーションパッケージのインストール](#)

[4. aarch64/ARV64v8コンテナがx86 Linuxマシンで動作しているかどうかをテストする](#)

[5. Docker Webserverコンテナを構築するためのファイルの準備](#)

[6. Dockerコンテナの構築](#)

[7. IOxパッケージの構築](#)

[パート2:IR1101 for IOxの設定](#)

[1. Webインターフェイス、IOx、およびローカルマネージャを有効にします](#)

[2. IOxネットワーキングの設定](#)

[パート3:Local ManagerへのアクセスとIOxアプリケーションの導入](#)

[確認](#)

[トラブルシューティング](#)

概要

このドキュメントでは、IR1101 ARMベースのInternet of Things(IoT)ゲートウェイ用のDockerベースのIOxパッケージを準備、構築、および導入する方法について説明します。

前提条件

要件

次の項目に関する知識があることが推奨されます。

- Linux
- コンテナ
- IOx

使用するコンポーネント

このドキュメントの情報は、次のソフトウェアとハードウェアのバージョンに基づいています。

- セキュアシェル(SSH)経由で到達可能なIR1101
設定されたIPアドレス特権15ユーザによるデバイスへのアクセス
- Linuxホスト(この記事では最小限のDebian 9 (ストレッチ) インストールを使用しています)
- IOxクライアントのインストールファイル (ダウンロード可能)
) :<https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762>

このドキュメントの情報は、特定のラボ環境にあるデバイスに基づいて作成されました。このドキュメントで使用するすべてのデバイスは、初期 (デフォルト) 設定の状態から起動しています。対象のネットワークが実稼働中である場合には、どのようなコマンドについても、その潜在的な影響について確実に理解しておく必要があります。

背景説明

IR1101は主にx86ベースであるため、他のほとんどのIOxプラットフォームと比較して少し異なります。IR1101はARM64v8アーキテクチャに基づいているため、x86用に構築されたコンテナやIOxパッケージをプラットフォームに直接導入することはできません。このドキュメントは最初から始まり、ARM64v8ベースのDockerコンテナを構築するための環境を準備し、x86 PCを使用してIR1101上でそれらを構築、パッケージ化、および展開する方法を説明します。

例としては、単純なウェブサーバである非常に小さなPythonスクリプトを使用し、Dockerコンテナを構築して最終的にIR1101上で実行します。ウェブサーバが行うのは、事前に定義されたポート(9000)をし、単純GET要求を時時にGETをし単純GETしGETこれにより、独自のコードを実行する機能をテストし、IOxアプリケーションの実行開始後にネットワークアクセスをテストできます。

このパッケージは、Alpine Linuxを使用してDockerツールによって構築されます。Alpine Linuxは小さなLinuxイメージ (約5MB) で、Dockerコンテナのベースとしてよく使用されます。

ほとんどのデスクトップ/ラップトップ/VMはx86ベースであるため、コンテナが構築されているx86ベースのマシンでARM64v8アーキテクチャをエミュレートする必要があります。これを簡単に行うには、クイックエミュレータ(QEMU)ユーザエミュレーションを使用します。これにより、ネイティブでないアーキテクチャで実行可能ファイルを実行できるようになります。これは、ネイティブのアーキテクチャで実行するのと同じです。

設定

第1部IR1101用IOxパッケージの構築

1. LinuxホストへのIOxクライアントのインストールと準備

Dockerコンテナを構築後にIOxパッケージとしてパッケージ化するには、ioxclientが必要です。まずは準備しましょう。

最初にioxclientパッケージをコピーまたはダウンロードします。

<https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762>から入手できます。

```
jedepuyd@192.168.56.101's password:
ioxclient_1.7.0.0_linux_amd64.tar.gz
```

```
100% 4798KB 75.2MB/s 00:00
```

パッケージを展開します。

```
jedepuyd@deb9:~$ tar -xvzf ioxclient_1.7.0.0_linux_amd64.tar.gz
ioxclient_1.7.0.0_linux_amd64/ioxclient
ioxclient_1.7.0.0_linux_amd64/README.md
```

PATH変数にパスを追加し、完全な場所を使用せずにパスを使用可能にします。マシンまたはス
イッチユーザをリブートする場合は、この手順を繰り返すことを忘れないでください。

```
jedepuyd@deb9:~$ export PATH=$PATH:/home/jedepuyd/ioxclient_1.7.0.0_linux_amd64/
```

必須プロファイルを作成するために、初めてioxclientを起動します。ioxclientを使用してDockerコ
ンテナをパッケージ化するだけなので、値はデフォルトのままにすることができます。

```
jedepuyd@deb9:~$ ioxclient -v
ioxclient version 1.7.0.0
jedepuyd@deb9:~/iox_aarch64_webserver$ ioxclient profiles reset
Active Profile : default
Your current config details will be lost. Continue (y/N) ? : y
Current config backed up at /tmp/ioxclient731611124
Config data deleted.
jedepuyd@deb9:~/iox_aarch64_webserver$ ioxclient -v
Config file not found : /home/jedepuyd/.ioxclientcfg.yaml
Creating one time configuration..
Your / your organization's name :
Your / your organization's URL :
Your IOx platform's IP address[127.0.0.1] :
Your IOx platform's port number[8443] :
Authorized user name[root] :
Password for root :
Local repository path on IOx platform[/software/downloads]:
URL Scheme (http/https) [https]:
API Prefix[/iox/api/v2/hosting/]:
Your IOx platform's SSH Port[2222]:
Your RSA key, for signing packages, in PEM format[:
Your x.509 certificate in PEM format[:
Activating Profile default
Saving current configuration
ioxclient version 1.7.0.0
```

2. LinuxビルドマシンでのDocker環境のインストールと準備

このDockerは、Alpineベースイメージからコンテナを構築し、ユースケースに必要なファイルを含めるために使用されます。所定の手順は、Docker Community Edition(CE)for Debianの公式インストールガイドに基づいています。<https://docs.docker.com/install/linux/docker-ce/debian/>

マシンのパッケージリストを更新します。

```
jedepuyd@deb9:~$ sudo apt-get update
...
```

Docker repoを使用するには、依存関係をインストールします。

```
jedepuyd@deb9:~$ sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-
```

```
properties-common
Reading package lists... Done
Building dependency tree
...
```

Docker GNU Privacy Guard(GPG)キーを有効なGPGキーとして追加します。

```
jedepuyd@deb9:~$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
OK
```

インストールされているGPGキーのフィンガープリントを確認します。

```
jedepuyd@deb9:~$ sudo apt-key fingerprint 0EBFCD88
pub  rsa4096 2017-02-22 [SCEA]
     9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid          [ unknown] Docker Release (CE deb) <docker@docker.com>
sub  rsa4096 2017-02-22 [S]
```

Docker安定リポジトリを追加します。

```
jedepuyd@deb9:~$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/debian $(lsb_release -cs) stable"
```

Dockerリポジトリを追加するときに、パッケージリストを再度更新します。

```
jedepuyd@deb9:~$ sudo apt-get update
...
```

Reading package lists... Done
Dockerのインストール：

```
jedepuyd@deb9:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree
...
```

```
Processing triggers for systemd (232-25+deb9u9) ...
```

通常ユーザとしてDockerにアクセス/実行できるようにするには、このユーザをDockerグループに追加し、グループメンバーシップを更新します。

```
jedepuyd@deb9:~$ sudo usermod -a -G docker jedepuyd
jedepuyd@deb9:~$ newgrp docker
```

3. QEMUユーザエミュレーションパッケージのインストール

Dockerをインストールしたら、QEMUユーザエミュレータをインストールする必要があります。Dockerコンテナ内から静的にリンクされたQEMUエミュレータを使用して、x86ベースのLinuxマシンでARM64v8用のコンテナを実行できます。ただし、ターゲットコンテナはARM64v8アーキテクチャ用に設計されています。

パッケージをインストールします。

```
jedepuyd@deb9:~$ sudo apt-get install qemu-user qemu-user-static
Reading package lists... Done
Building dependency tree
```

...

Processing triggers for man-db (2.7.6.1-2) ...

インストール後、/usr/binにある静的にリンクされたQEMUエミュレータを次に示します。

```
jedepuyd@deb9:~$ ls -al /usr/bin/qemu-*static
-rwxr-xr-x 1 root root 3468784 Nov  8 16:41 /usr/bin/qemu-aarch64-static
-rwxr-xr-x 1 root root 2791408 Nov  8 16:41 /usr/bin/qemu-alpha-static
-rwxr-xr-x 1 root root 3399344 Nov  8 16:41 /usr/bin/qemu-armeb-static
-rwxr-xr-x 1 root root 3391152 Nov  8 16:41 /usr/bin/qemu-arm-static
-rwxr-xr-x 1 root root 2800400 Nov  8 16:41 /usr/bin/qemu-cris-static
```

...

リストの最初のリストは、必要なリストです。aarch64は、Linux用のARM64v8のarch-nameです。

4. aarch64/ARV64v8コンテナがx86 Linuxマシンで動作しているかどうかをテストする

Dockerと必要なQEMUバイナリがインストールされたので、x86マシンでARM64v8用にビルドされたDockerコンテナを実行できるかどうかをテストできます。

```
jedepuyd@deb9:~$ docker run -v /usr/bin/qemu-aarch64-static:/usr/bin/qemu-aarch64-static --rm -ti arm64v8/alpine:3.7
Unable to find image 'arm64v8/alpine:3.7' locally
3.7: Pulling from arm64v8/alpine
40223db5366f: Pull complete
Digest: sha256:a50c0cd3b41129046184591963a7a76822777736258e5ade8445b07c88bfdcc3
Status: Downloaded newer image for arm64v8/alpine:3.7
/ # uname -a
Linux 1dbba69b60c5 4.9.0-8-amd64 #1 SMP Debian 4.9.144-3.1 (2019-02-19) aarch64 Linux
```

出力からわかるように、arm64v8 Alpineコンテナが取得され、エミュレータにアクセスして実行されます。

コンテナのアーキテクチャを要求すると、コードがaarch64用にコンパイルされていることがわかります。コンテナのターゲットアーチとまったく同じものがIR1101用です。

5. Docker Webserverコンテナを構築するためのファイルの準備

すべての準備が完了したら、IR1101で実行する必要があるWebサーバコンテナに必要なファイルを作成します。

最初のファイルはwebserver.pyです。これは、コンテナで実行するPythonスクリプトです。これは単なる例であるため、IOxアプリケーションで実行するために、実際のコードで置き換えます。

```
jedepuyd@deb9:~$ mkdir iox_aarch64_webserver
jedepuyd@deb9:~$ cd iox_aarch64_webserver

jedepuyd@deb9:~/iox_aarch64_webserver$ vi webserver.py
jedepuyd@deb9:~/iox_aarch64_webserver$ cat webserver.py
#!/usr/bin/env python
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
import SocketServer
import os
```

```

class S(BaseHTTPRequestHandler):
    def _set_headers(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()

    def do_GET(self):
        self._set_headers()
        self.wfile.write("<html><body><h1>IOX python webserver on arm64v8</h1></body></html>")
        logf.write('Got GET\n')
        logf.flush()

def run(server_class=HTTPServer, handler_class=S, port=9000):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print 'Starting webserver...'
    logf.write('Starting webserver...\n')
    logf.flush()
    httpd.serve_forever()

if __name__ == "__main__":
    log_file_dir = os.getenv("CAF_APP_LOG_DIR", "/tmp")
    log_file_path = os.path.join(log_file_dir, "webserver.log")
    logf = open(log_file_path, 'w')
    run()
    logf.close()

```

このコードには、ログファイルに書き込むためのロジックが含まれています。このロジックは、Local Managerからの相談に利用できます。

2番目に必要なファイルはDockerfileです。コンテナの構築方法を定義します。

```

jedepuyd@deb9:~/iox_aarch64_webserver$ vi Dockerfile
jedepuyd@deb9:~/iox_aarch64_webserver$ cat Dockerfile
FROM arm64v8/alpine:3.7
COPY qemu-aarch64-static /usr/bin

RUN apk add --no-cache python
COPY webserver.py /webserver.py

```

Dockerfileは、コンテナの構築方法を定義します。ARM64v8のAlpineベースイメージから開始し、コンテナにエミュレータをコピーし、apkを実行してPythonパッケージを追加し、webserverスクリプトをコンテナにコピーします。

コンテナを構築する前に必要な最後の準備は、コンテナを構築するディレクトリにqemu-aarch64-staticをコピーすることです。

```

jedepuyd@deb9:~/iox_aarch64_webserver$ cp /usr/bin/qemu-aarch64-static .

```

6. Dockerコンテナの構築

準備が完了したので、Dockerfileを使用してコンテナを構築できます。

```

jedepuyd@deb9:~/iox_aarch64_webserver$ docker build -t iox_aarch64_webserver .
Sending build context to Docker daemon 3.473MB
Step 1/4 : FROM arm64v8/alpine:3.7
----> e013d5426294
Step 2/4 : COPY qemu-aarch64-static /usr/bin

```

```

---> addf4e1cc965
Step 3/4 : RUN apk add --no-cache python
---> Running in ff3768926645
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/aarch64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/aarch64/APKINDEX.tar.gz
(1/10) Installing libbz2 (1.0.6-r6)
(2/10) Installing expat (2.2.5-r0)
(3/10) Installing libffi (3.2.1-r4)
(4/10) Installing gdbm (1.13-r1)
(5/10) Installing ncurses-terminfo-base (6.0_p20171125-r1)
(6/10) Installing ncurses-terminfo (6.0_p20171125-r1)
(7/10) Installing ncurses-libs (6.0_p20171125-r1)
(8/10) Installing readline (7.0.003-r0)
(9/10) Installing sqlite-libs (3.25.3-r0)
(10/10) Installing python2 (2.7.15-r2)
Executing busybox-1.27.2-r11.trigger
OK: 51 MiB in 23 packages
Removing intermediate container ff3768926645
---> eda469dab9c6
Step 4/4 : COPY webserver.py /webserver.py
---> ccf7ee7227c9
Successfully built ccf7ee7227c9
Successfully tagged iox_aarch64_webserver:latest

```

テストとして、作成したばかりのコンテナを実行し、スクリプトが機能するかどうかを確認します。

```

jedepuyd@deb9:~/iox_aarch64_webserver$ docker run -ti iox_aarch64_webserver
/ # uname -a
Linux dae047f1a6b2 4.9.0-8-amd64 #1 SMP Debian 4.9.144-3.1 (2019-02-19) aarch64 Linux
/ # python webserver.py &
/ # Starting webserver...

/ # netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:9000            0.0.0.0:*               LISTEN      13/qemu-aarch64-
sta
/ # exit

```

この出力からわかるように、コンテナのアーキテクチャはターゲットのarch64です。スクリプトを開始すると、ポート9000で要求をリスンしていることがわかります。

7. IOxパッケージの構築

コンテナをパッケージする準備ができました。これをioxclientに依頼する前に、まずパッケージ記述子を作成する必要があります。**package.yaml**。

このファイルは、パッケージの外観、実行する必要があるリソースの数、および起動する必要があるリソースを説明します。

```

jedepuyd@deb9:~/iox_aarch64_webserver$ vi package.yaml
jedepuyd@deb9:~/iox_aarch64_webserver$ cat package.yaml
descriptor-schema-version: "2.7"

info:
  name: "iox_aarch64_webserver"
  description: "simple docker webserver for arm64v8"
  version: "1.0"
  author-link: "http://www.cisco.com"

```

```
author-name: "Jens Depuydt"
```

```
app:  
  cpuarch: "aarch64"  
  type: "docker"  
  resources:  
    profile: cl.tiny  
    network:  
      -  
        interface-name: eth0  
        ports:  
          tcp: ["9000"]  
  
  startup:  
    rootfs: rootfs.tar  
    target: ["python", "/webserver.py"]
```

ご覧のように、CPUアーキテクチャはaarch64に設定されています。TCPポート9000にアクセスするには、rootfsとしてrootfs.tarを使用し、開始時にはpython/webserver.pyを実行できます。

パッケージ化する前に最後に行うことは、Dockerコンテナからrootfs.tarを抽出することです。

```
jedepuyd@deb9:~/iox_aarch64_webserver$ docker save -o rootfs.tar iox_aarch64_webserver  
この時点で、ioxclientを使用してIR1101用のIOxパッケージを構築できます。
```

```
jedepuyd@deb9:~/iox_aarch64_webserver$ ioxclient package .  
Currently active profile : default  
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/jedepuyd/iox_aarch64_webserver/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/jedepuyd/iox_aarch64_webserver/package.yaml is valid under schema version 2.7  
Created Staging directory at : /tmp/017226485  
Copying contents to staging directory  
Creating an inner envelope for application artifacts  
Generated /tmp/017226485/artifacts.tar.gz  
Calculating SHA1 checksum for package contents..  
Updated package metadata file : /tmp/017226485/.package.metadata  
Root Directory : /tmp/017226485  
Output file: /tmp/475248592  
Path: .package.metadata  
SHA1 : 95abe28fc05395fc5f71f7c28f59eceb1495bf9b  
Path: artifacts.tar.gz  
SHA1 : bdf5596a0747eae51bb0a1d2870fd09a5a16a098  
Path: package.yaml  
SHA1 : e65a6fcbe96725dd5a09b60036448106acc0c138  
Generated package manifest at package.mf  
Generating IOx Package..  
Package generated at /home/jedepuyd/iox_aarch64_webserver/package.tar
```

現在、package.tarとしてIR1101に導入するためのパッケージがあります。次のパートでは、デバイスの導入準備について説明します。

パート2:IR1101 for IOxの設定

1. Webインターフェイス、IOx、およびローカルマネージャを有効にします

Local Managerは、IOxアプリケーションの展開、アクティブ化、開始、管理、トラブルシューティングを行うためのGUIです。IR1101では、通常の管理Webインターフェイスに組み込まれています。したがって、最初にそれを有効にする必要があります。

IR1101で次の手順を実行して、IOxとWebインターフェイスを有効にします。

```
BRU_IR1101_20#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
BRU_IR1101_20(config)#iox
BRU_IR1101_20(config)#ip http server
BRU_IR1101_20(config)#ip http secure-server
BRU_IR1101_20(config)#ip http authentication local
BRU_IR1101_20(config)#username admin privilege 15 password 0 cisco
```

最後の行は、特権15権限を持つユーザを追加します。このユーザは、WebインターフェイスおよびIOxローカルマネージャにアクセスできます。

2. IOxネットワーキングの設定

Webインターフェイスにアクセスする前に、IOxネットワーキングに必要な設定を追加します。背景情報については、IR1101のIOxに関するドキュメント (https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b_IR1101config/b_IR1101config_chapter_010001.html)を参照してください。

つまり、IOxアプリケーションは、VirtualPortGroup0インターフェイス (IR809のGi2およびIR829インターフェイスのGi5に相当) を使用して外部と通信できます。

```
BRU_IR1101_20(config)#interface VirtualPortGroup0
BRU_IR1101_20(config-if)# ip address 192.168.1.1 255.255.255.0
BRU_IR1101_20(config-if)# ip nat inside
BRU_IR1101_20(config-if)# ip virtual-reassembly
BRU_IR1101_20(config-if)#exit
```

VirtualPortGroup0インターフェイスをネットワークアドレス変換(NAT)内部として設定する場合、NATを使用してIOxアプリケーションとの通信を可能にするために、Gi 0/0/0インターフェイスにip nat outsideステートメントを追加する必要があります。

```
BRU_IR1101_20(config)#interface gigabitEthernet 0/0/0
BRU_IR1101_20(config-if)#ip nat outside
BRU_IR1101_20(config-if)#ip virtual-reassembly
```

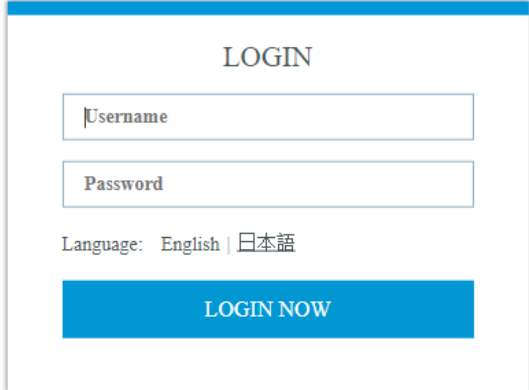
192.168.1.15を指定できるコンテナのポート9000へのアクセスを許可するには、ポート転送を追加する必要があります。

```
BRU_IR1101_20(config)#ip nat inside source static tcp 192.168.1.15 9000 interface
GigabitEthernet0/0/0 9000
```

このガイドでは、IOxアプリケーションごとに静的に設定されたIPを使用します。アプリケーションにIPアドレスを動的に割り当てる場合は、VirtualPortGroup0のサブネットにDHCPサーバの設定を追加する必要があります。

パート3:Local ManagerへのアクセスとIOxアプリケーションの導入

これらの行を設定に追加したら、Webインターフェイスを使用してIR1101にアクセスできます。図に示すように、ブラウザを使用してGi 0/0/0 IPアドレスに移動します。



The image shows a Cisco login interface. At the top center is the Cisco logo, consisting of a stylized signal icon above the word "CISCO". Below the logo is a white rectangular box with a blue border and a blue header bar containing the word "LOGIN". Inside the box, there are two input fields: the first is labeled "Username" and the second is labeled "Password". Below these fields, there is a language selection option: "Language: English | 日本語". At the bottom of the box is a blue button with the text "LOGIN NOW" in white. The background of the page features several vertical blue bars of varying heights and shades on the left side.

© 2005-2018 - Cisco Systems, Inc. All rights reserved. Cisco, the Cisco logo, and Cisco Systems are registered trademarks or trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries. All third party trademarks are the property of their respective owners.

ステップ1.で作成したprivilege 15アカウントを使用して、Webインターフェイスにログインし、図に示すように[Configuration - IOx]に移動します。



Search Menu Items



Dashboard



Monitoring



Configuration



Administration



Troubleshooting



Interface

Cellular

Ethernet

Logical



Layer2

VLAN

VTP



Routing Protocols

EIGRP

OSPF

Static Routing



Security

AAA

ACL

NAT

VPN



Services

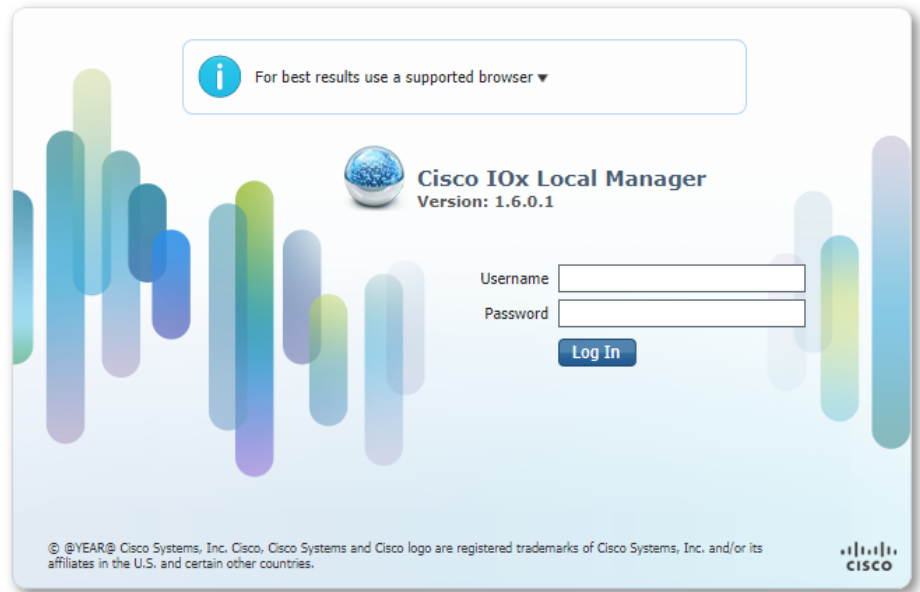
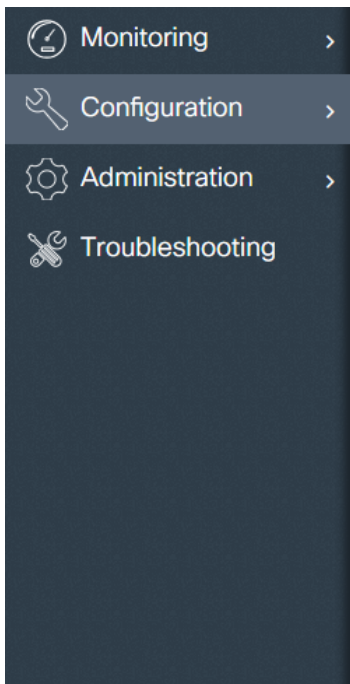
Application Visibility

Custom Application

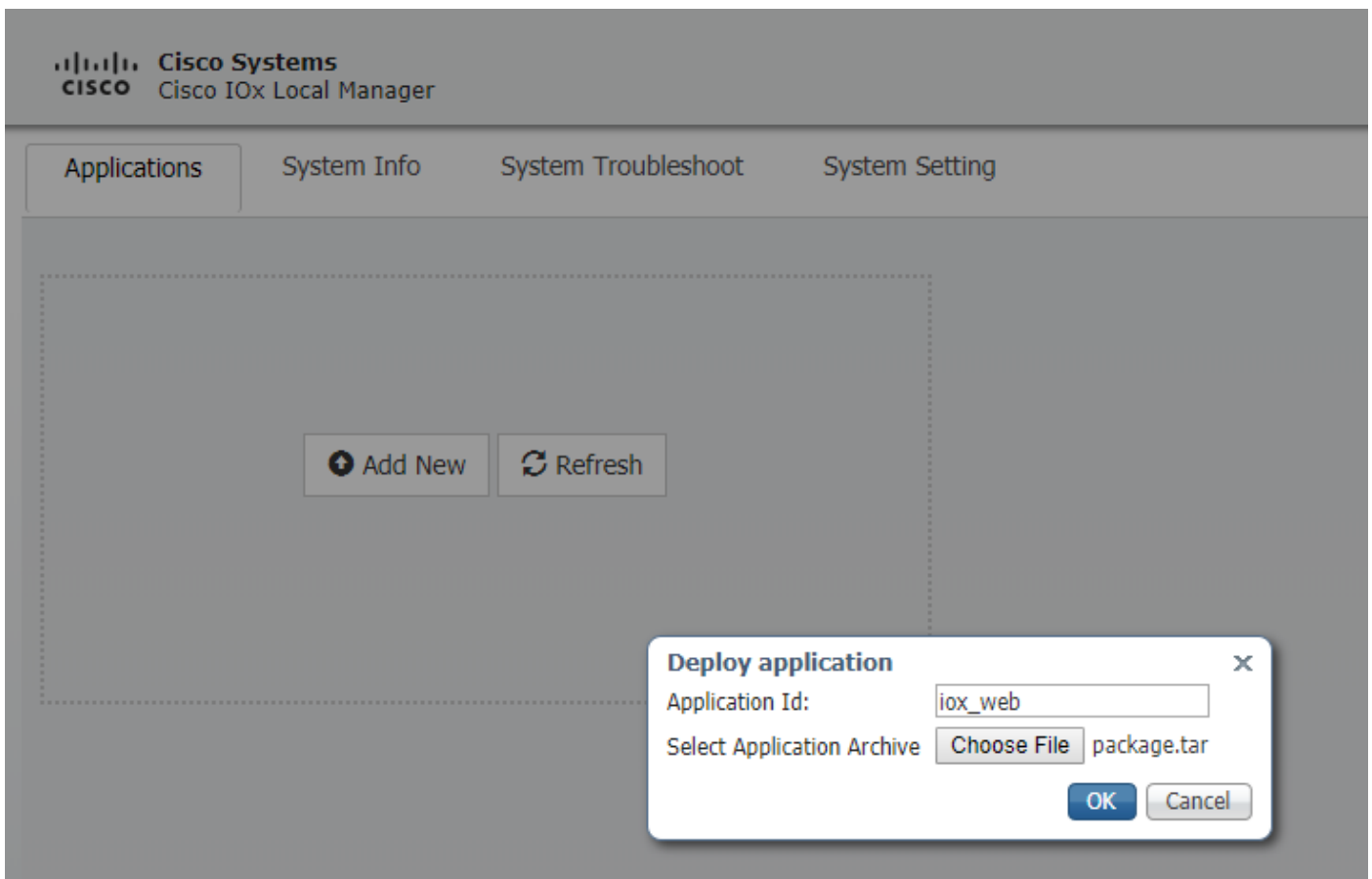
IOx

NETFLOW

IOx Local Managerログインで、図に示すように同じアカウントを使用して続行します。



[Add New] をクリックし、IOxアプリケーションの名前を選択し、図に示すようにPart 1に組み込まれたpackage.tarを選択します。



パッケージがアップロードされたら、図に示すようにアクティブにできます。

iox_web

DEPLOYED

simple docker webserver for arm64v8

TYPE	VERSION	PROFILE
docker	1.0	c1.tiny

Memory *	6.3%
----------	------

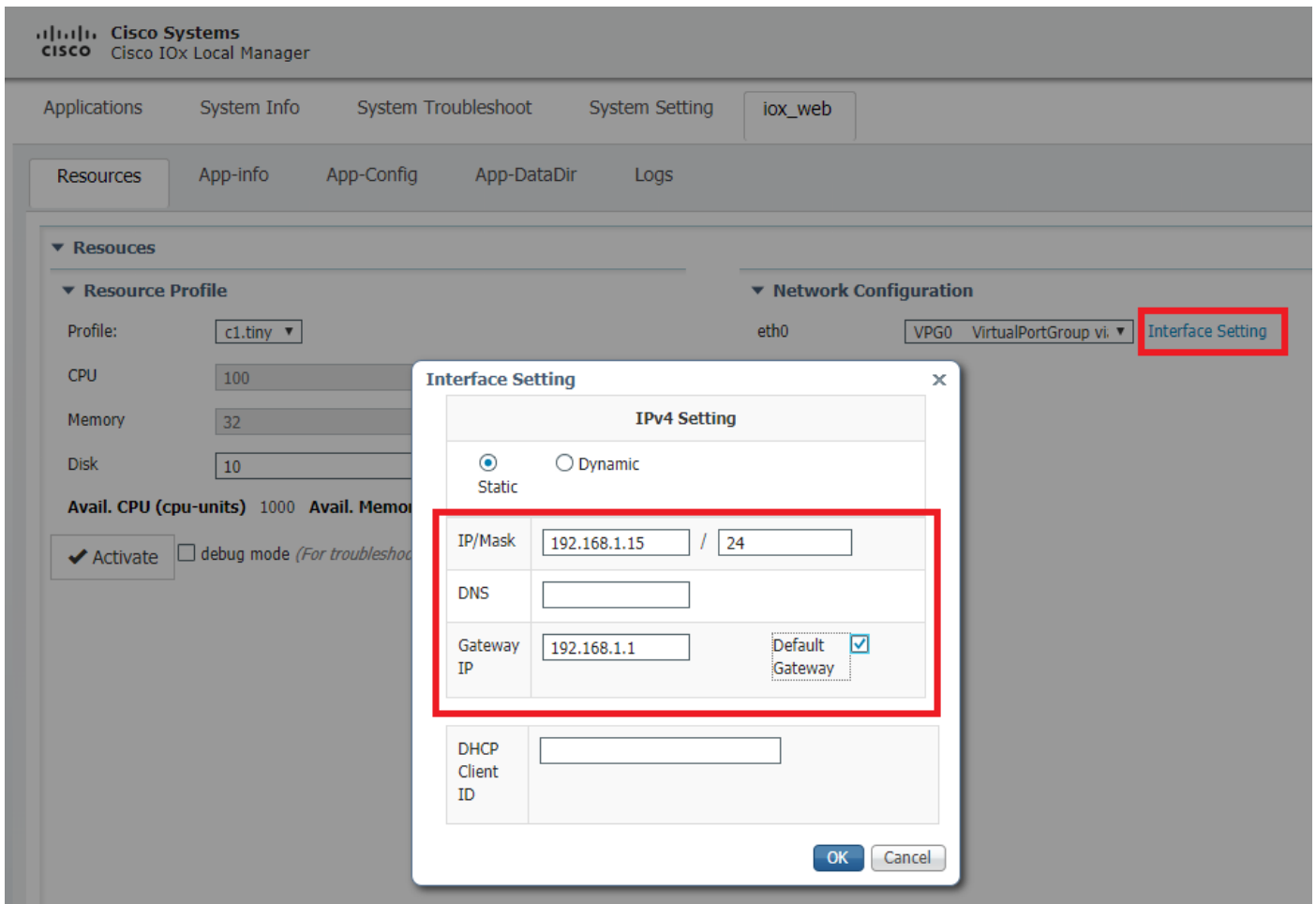
CPU *	10.0%
-------	-------

✓ Activate

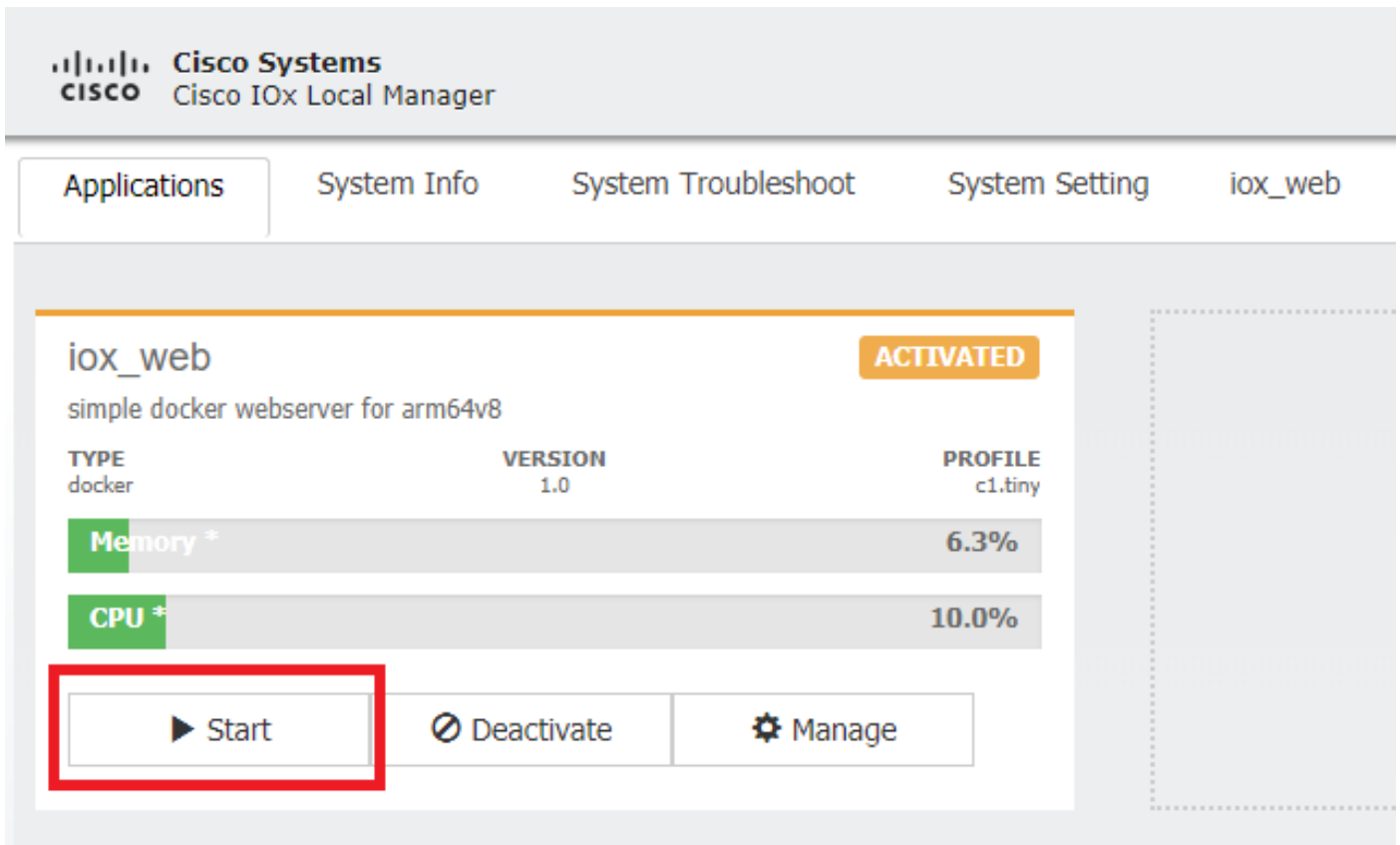
Upgrade

Delete

[Resources] タブでインターフェイス設定を開き、図に示すように、アプリケーションに割り当てる固定IPを指定します。



[OK]をクリックし、[Activate]をクリックします。操作が完了したら、メインの[Local Manager]ページ(トップメニューの[Applications] ボタン)に戻り、図に示すようにアプリケーションを起動します。



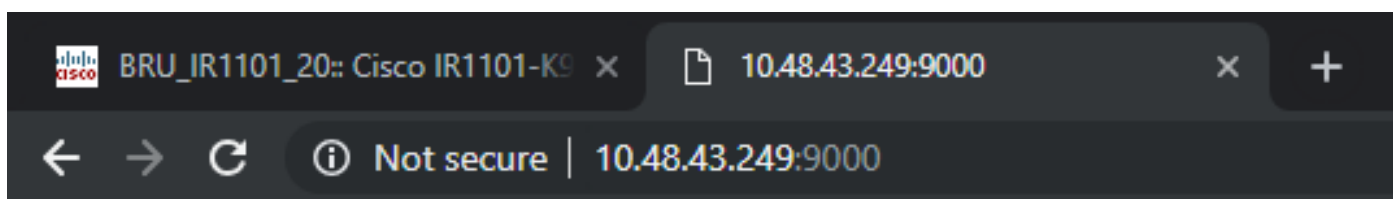
これらの手順を実行した後、IR1101のGi 0/0/0インターフェイスを使用して、アプリケーションが実行され、ポート9000を介して使用可能になります。

確認

ここでは、設定が正常に機能しているかどうかを確認します。

確認するには、ポート9000を使用して、IR1101のGi 0/0/0インターフェイスのIPアドレスにアクセスできます。

すべてが正常に動作している場合は、Pythonスクリプトで作成されたように、次のように表示されます。



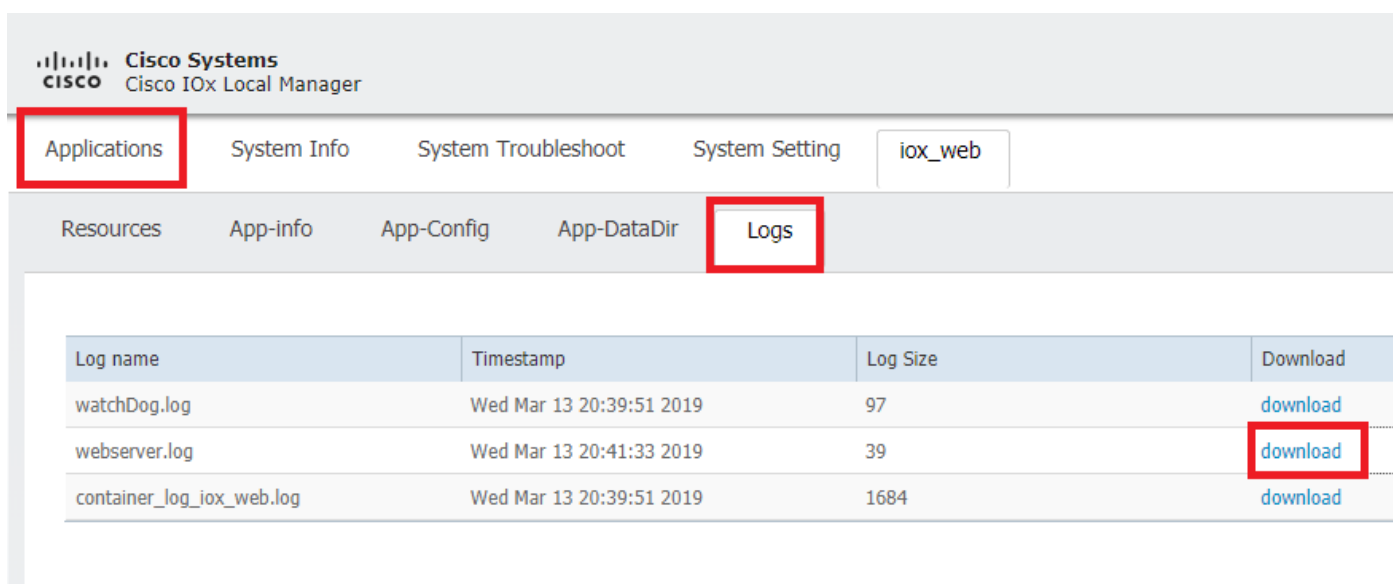
IOX python webserver on arm64v8

トラブルシューティング

ここでは、設定のトラブルシューティングに使用できる情報を示します。

トラブルシューティングを行うには、ローカルマネージャを使用して、Pythonスクリプトで作成したログファイルを確認します。

[Applications] に移動し、iox_webアプリケーションで[Manage]をクリックし、図に示すように[Logs]タブを選択します。



Log name	Timestamp	Log Size	Download
watchDog.log	Wed Mar 13 20:39:51 2019	97	download
webserver.log	Wed Mar 13 20:41:33 2019	39	download
container_log_iox_web.log	Wed Mar 13 20:39:51 2019	1684	download