

Crie e implante um pacote DOC IOx para a arquitetura IR1101 ARM

Contents

[Introduction](#)

[Prerequisites](#)

[Requirements](#)

[Componentes Utilizados](#)

[Informações de Apoio](#)

[Configurar](#)

[Parte 1. Monte o Pacote IOx para IR1101](#)

[1. Instalar e preparar o cliente IOx no host Linux](#)

[2. Instale e prepare o ambiente de docking no computador de build Linux](#)

[3. Instale os pacotes de emulação de usuário QEMU](#)

[4. Teste se um contêiner arch64/ARV64v8 é executado em máquina Linux x86](#)

[5. Preparar arquivos para criar o contêiner do docking Webserver](#)

[6. Construa o contêiner do docking station](#)

[7. Criar o pacote IOx](#)

[Parte 2. Configurar o IR1101 para IOx](#)

[1. Ative a interface da Web, o IOx e o Gerenciador local](#)

[2. Configurar rede IOx](#)

[Parte 3. Acesse o Local Manager e implante o aplicativo IOx](#)

[Verificar](#)

[Troubleshoot](#)

Introduction

Este documento descreve como preparar, criar e implantar um pacote de IOx baseado em Docker para o gateway da Internet das Coisas (IoT) baseado em IR1101 ARM.

Prerequisites

Requirements

A Cisco recomenda que você tenha conhecimento destes tópicos:

- Linux
- Contêineres
- IOx

Componentes Utilizados

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

- IR1101 acessível através do Shell Seguro (SSH)
Endereço IP configurado Acesso ao dispositivo com privilégio de 15 usuários
- Host Linux (uma instalação mínima do Debian 9 (alongamento) é usada para este artigo)
- Arquivos de instalação do cliente IOx que podem ser baixados de:
<https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762>

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, make sure that you understand the potential impact of any command.

Informações de Apoio

O IR1101 é um pouco diferente em comparação com a maioria das outras plataformas de IOx, pois elas são baseadas principalmente em x86. O IR1101 é baseado na arquitetura ARM64v8 para que você não possa implantar contêineres ou pacotes IOx construídos para x86 na plataforma diretamente. Este documento começa do zero e prepara o ambiente para a construção de contêineres Docker baseados em ARM64v8 e explica como construir, embalar e implantar no IR1101 com o uso de um PC x86.

Como exemplo, um script Python muito pequeno, que é um simples servidor web, é usado e um contêiner Docker é construído para eventualmente embalar-lo para ser executado no IR1101. A única coisa que o servidor web fará é ouvir em uma porta predefinida (9000) e retornar um pagev simples quando receber uma solicitação **GET**. Isso permite que você teste a capacidade de executar seu próprio código e permite testar o acesso à rede para o aplicativo IOx quando ele começa a ser executado.

O pacote será construído pelas ferramentas Docker, com o uso do Linux alpino. O Linux Alpino é uma pequena imagem Linux (cerca de 5 MB), que é frequentemente usada como base para contêineres Docker.

Como a maioria das VMs/desktop/laptop em torno é baseada em x86, você precisa emular a arquitetura ARM64v8 na máquina baseada em x86 onde o contêiner é criado. Você pode fazer isso facilmente com o uso da emulação de usuário do Quick Emulator (QEMU). Isso permite a execução de executáveis em uma arquitetura não-nativa da mesma forma que seria executada em sua arquitetura nativa.

Configurar

Parte 1. Monte o Pacote IOx para IR1101

1. Instalar e preparar o cliente IOx no host Linux

Você precisa do **ioxclient** para empacotar o contêiner Docker como um pacote IOx quando ele for criado, então vamos preparar isso primeiro.

Primeiro, copie ou faça o download do pacote **ioxclient**. Está disponível em:

<https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762>.

```
jedepuyd@deb9:~$ scp jedepuyd@192.168.56.101:/home/jedepuyd/ioxclient_1.7.0.0_linux_amd64.tar.gz
```

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

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

Extrair o pacote:

```
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
```

Adicione o caminho à variável **PATH** para que ela esteja disponível sem o uso do local completo. Se você reinicializar os usuários da máquina ou do switch, não se esqueça de repetir esta etapa:

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

Inicie o **ioxclient** pela primeira vez para criar um perfil obrigatório. Como você só usará o **ioxclient** para empacotar o contêiner do Docker, os valores podem ser deixados como padrão:

```
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. Instale e prepare o ambiente de docking no computador de build Linux

Este Docker é usado para criar um contêiner a partir da imagem base alpina e para incluir os arquivos necessários para o caso de uso. As etapas indicadas baseiam-se nos guias de instalação oficiais do Docker Community Edition (CE) para Debian:

<https://docs.docker.com/install/linux/docker-ce/debian/>

Atualize as listas de pacotes na sua máquina:

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

Instale as dependências para usar o acordo de docking station:

```
jedepuyd@deb9:~$ sudo apt-get install apt-transport-https ca-certificates curl gnupg2 software-properties-common
Reading package lists... Done
Building dependency tree
...
Processing triggers for dbus (1.10.26-0+deb9u1) ...
```

Adicione a chave GPG (Docker GNU Privacy Guard) como uma chave GPG válida:

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

Verifique a impressão digital da chave GPG instalada:

```
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]
```

Adicione o acordo de recompra estável do Docker:

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

Atualize as listas de pacotes novamente à medida que você adiciona o Docker repo:

```
jedepuyd@deb9:~$ sudo apt-get update
...
Reading package lists... Done
```

Instalar Docker:

```
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) ...
```

Para poder acessar/executar o Docker como um usuário regular, adicione esse usuário ao grupo Docker e atualize a associação ao grupo:

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

3. Instale os pacotes de emulação de usuário QEMU

Depois de instalar o Docker, é necessário instalar os emuladores de usuário do QEMU. Use o emulador QEMU vinculado estaticamente de dentro do contêiner Docker para executar o contêiner para ARM64v8 em nossa máquina Linux baseada em x86, embora o contêiner de destino seja projetado para a arquitetura ARM64v8.

Instale os pacotes:

```
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) ...

Após a instalação, aqui estão os emuladores QEMU vinculados estaticamente disponíveis em **/usr/bin**:

```
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
```

...

O primeiro na lista é aquele de que você precisa: aarch64 é o nome do arco para ARM64v8 para Linux.

4. Teste se um contêiner arch64/ARV64v8 é executado em máquina Linux x86

Agora que você tem o Docker e os binários QEMU necessários instalados, você pode testar se é capaz de executar um contêiner Docker criado para ARM64v8 na máquina x86:

```
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
```

Como você pode ver na saída, o contêiner alpino arm64v8 é obtido e executado com acesso ao emulador.

Se você solicitar a arquitetura do contêiner, poderá ver que o código foi compilado para aarch64. Exatamente como o arco-alvo do contêiner deve ser para IR1101.

5. Preparar arquivos para criar o contêiner do docking Webserver

Agora que toda preparação está concluída, você pode criar os arquivos necessários para o contêiner do servidor Web que precisa ser executado em IR1101.

O primeiro arquivo é **webserver.py**, o script Python que você deseja executar no contêiner. Como este é apenas um exemplo, obviamente, você substituirá isso pelo código real para executar em seu aplicativo IOx:

```
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()

```

Este código contém a lógica para gravar em um arquivo de registro, que estará disponível para consulta do Local Manager.

O segundo arquivo necessário é o Dockerfile. Isso define como o contêiner é criado:

```

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

```

O arquivo Dockerfile define como o contêiner será criado. Comece a partir da imagem base do Apline para ARM64v8, copie o emulador no contêiner, execute o aplicativo para adicionar o pacote Python e copie o script do servidor web no contêiner.

A última preparação necessária para que você possa criar o contêiner é copiar qemu-arch64-static para o diretório de onde você criará o contêiner:

```

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

```

6. Construa o contêiner do docking station

Agora que toda a preparação está concluída, você pode construir o contêiner usando o arquivo Dockerfile:

```

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

```

Como um teste, execute o contêiner que você acabou de criar e verifique se o script funciona:

```

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

```

Como você pode ver nesta saída, a arquitetura do contêiner é o arch64 de destino. E depois que você inicia o script, você vê que ele está escutando solicitações na porta 9000.

7. Criar o pacote IOx

O recipiente está pronto para ser embalado. Antes de solicitar ao ioxclient que faça isso, é necessário primeiro criar o descritor do pacote: **package.yaml**.

Este arquivo descreve como o pacote deve ser, quantos recursos ele precisa executar e o que iniciar.

```

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"]
```

Como você pode ver, a arquitetura da CPU está definida como arch64. Para obter acesso à porta TCP 9000, use **rootfs.tar** como rootfs e, no início, execute **python/webserver.py**.

A última coisa a fazer antes que você possa empacotar é extrair o **rootfs.tar** do contêiner do Docker:

```
jedepuyd@deb9:~/iox_aarch64_webserver$ docker save -o rootfs.tar iox_aarch64_webserver
```

Neste ponto, você pode usar o **ioxclient** para criar o pacote IOx para IR1101:

```
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
```

No momento, há um pacote para implantar no IR1101 pronto como **package.tar**. A próxima parte explica como preparar o dispositivo para implantação.

Parte 2. Configurar o IR1101 para IOx

1. Ative a interface da Web, o IOx e o Gerenciador local

O Local Manager é uma GUI para implantar, ativar, iniciar, gerenciar e solucionar problemas de aplicativos IOx. Para IR1101, ele está incorporado na interface da Web de gerenciamento regular. Então, você precisa ativar isso primeiro.

Execute estas etapas no IR1101 para ativar o IOx e a interface da Web.

```
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
```

A última linha adiciona um usuário com permissões de privilégio 15. Este usuário terá acesso à interface da Web e ao gerenciador local IOx.

2. Configurar rede IOx

Antes de acessar a interface da Web, vamos adicionar a configuração necessária para a rede IOx. Informações de fundo podem ser encontradas na documentação IR1101 para IOx:

https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b_IR1101_config/b_IR1101config_chapter_010001.html

Resumindo, os aplicativos IOx podem se comunicar com o mundo externo com o uso da interface VirtualPortGroup0 (comparável ao Gi2 em IR809 e Gi5 em interfaces IR829).

```
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
```

À medida que você configura a interface VirtualPortGroup0 como Network Address Translation (NAT) interna, você precisa adicionar a instrução `ip nat outside` na interface Gi 0/0/0 para permitir a comunicação entre e dos aplicativos IOx com o uso de NAT:

```
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
```

Para permitir o acesso à porta 9000 para o contêiner, que você pode dar a 192.168.1.15, você precisa adicionar uma porta adiante:

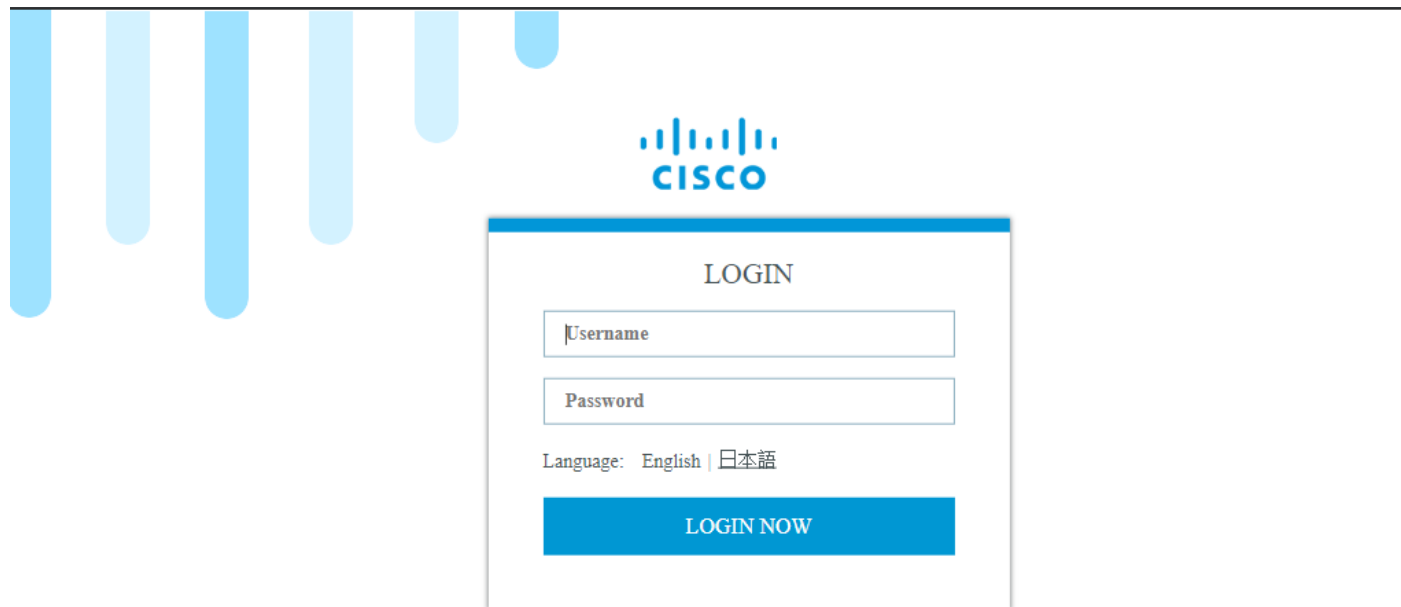
```
BRU_IR1101_20(config)#ip nat inside source static tcp 192.168.1.15 9000 interface
GigabitEthernet0/0/0 9000
```

Para este guia, use IPs configurados estaticamente por aplicativo IOx. Se quiser atribuir endereços IP dinamicamente aos aplicativos, você precisará adicionar a configuração de um

servidor DHCP na sub-rede do VirtualPortGroup0.

Parte 3. Acesse o Local Manager e implante o aplicativo IOx

Depois de adicionar essas linhas à configuração, você pode acessar o IR1101 com o uso da interface da Web. Navegue até o endereço IP Gi 0/0/0 com o uso do navegador como mostrado na imagem.



© 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.

Use a conta de privilégio 15 criada na Etapa 1. para fazer login na interface da Web e navegar até **Configuration** - IOx como mostrado na imagem.



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

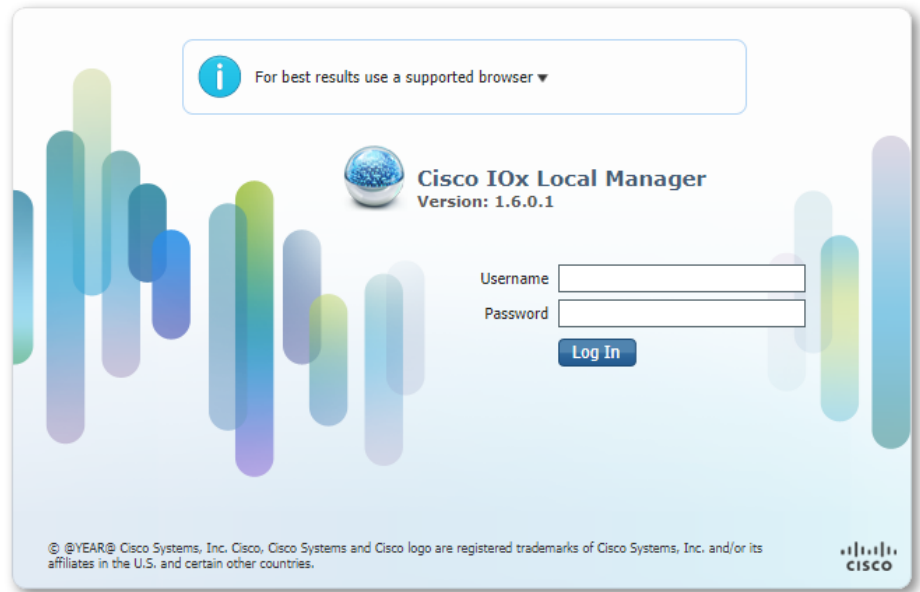
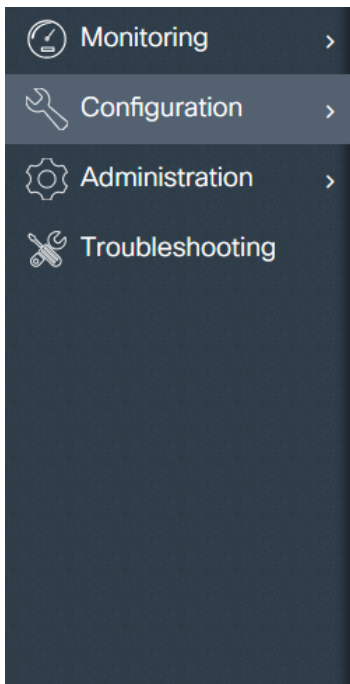
Application Visibility

Custom Application

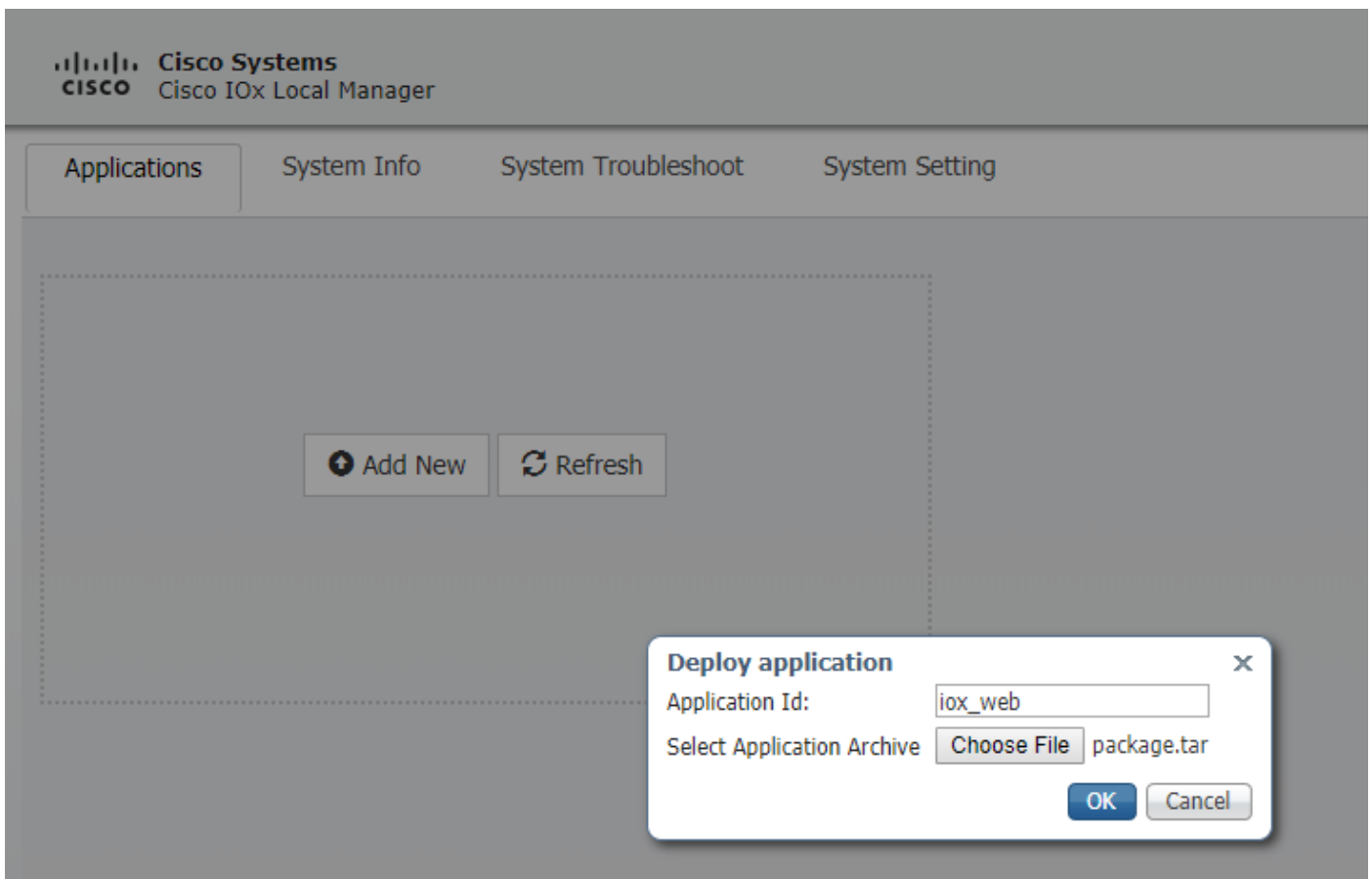
IOx

NETFLOW

No login do IOx Local Manager, use a mesma conta para continuar como mostrado na imagem.



Clique em **Adicionar novo**, selecione um nome para o aplicativo IOx e escolha o package.tar que foi criado na Parte 1 como mostrado na imagem.



Quando o pacote for carregado, você poderá ativá-lo conforme mostrado na imagem.

iox_web

DEPLOYED

simple docker webserver for arm64v8

TYPE	VERSION	PROFILE
docker	1.0	c1.tiny

Memory *

6.3%

CPU *

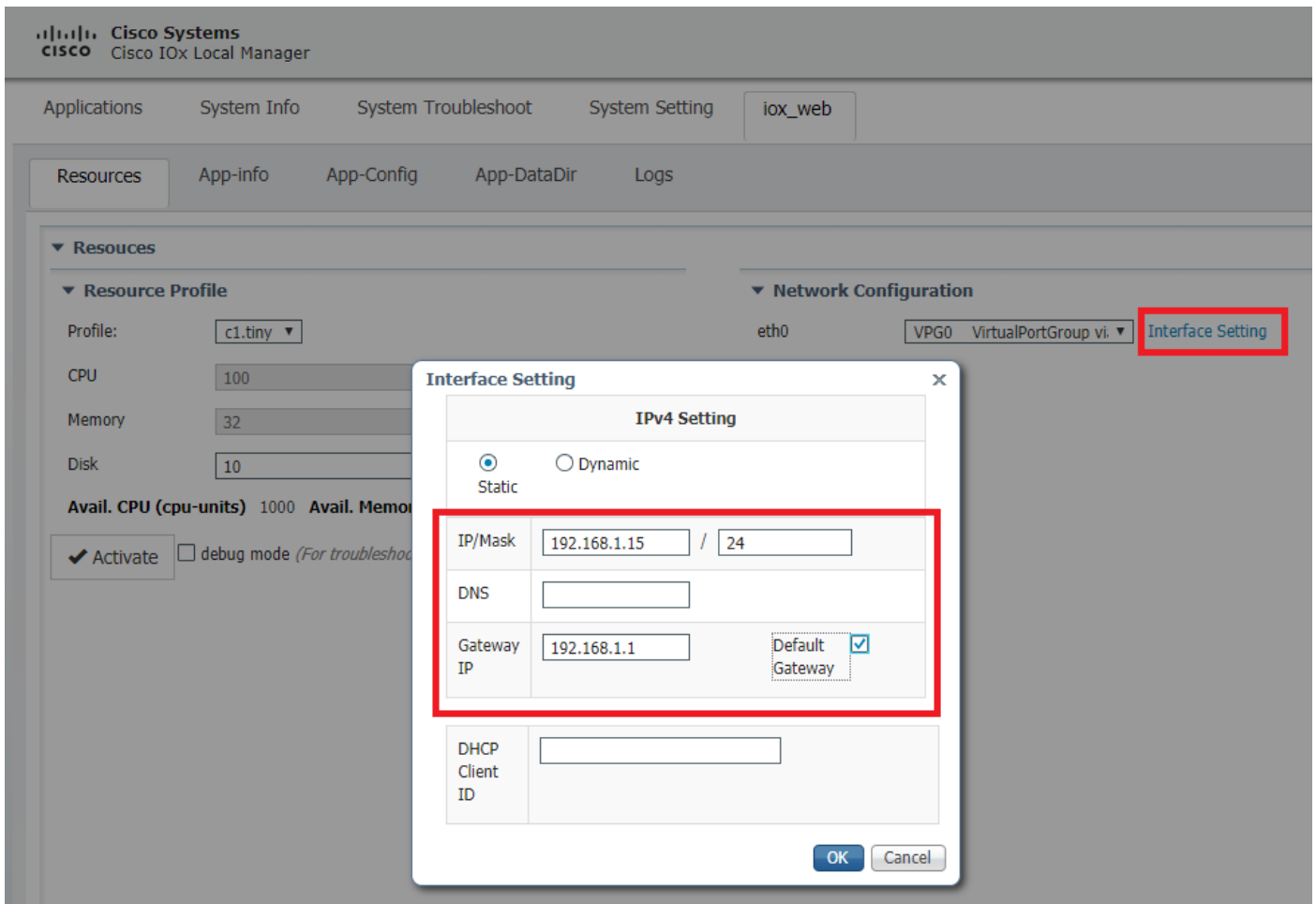
10.0%

✓ Activate

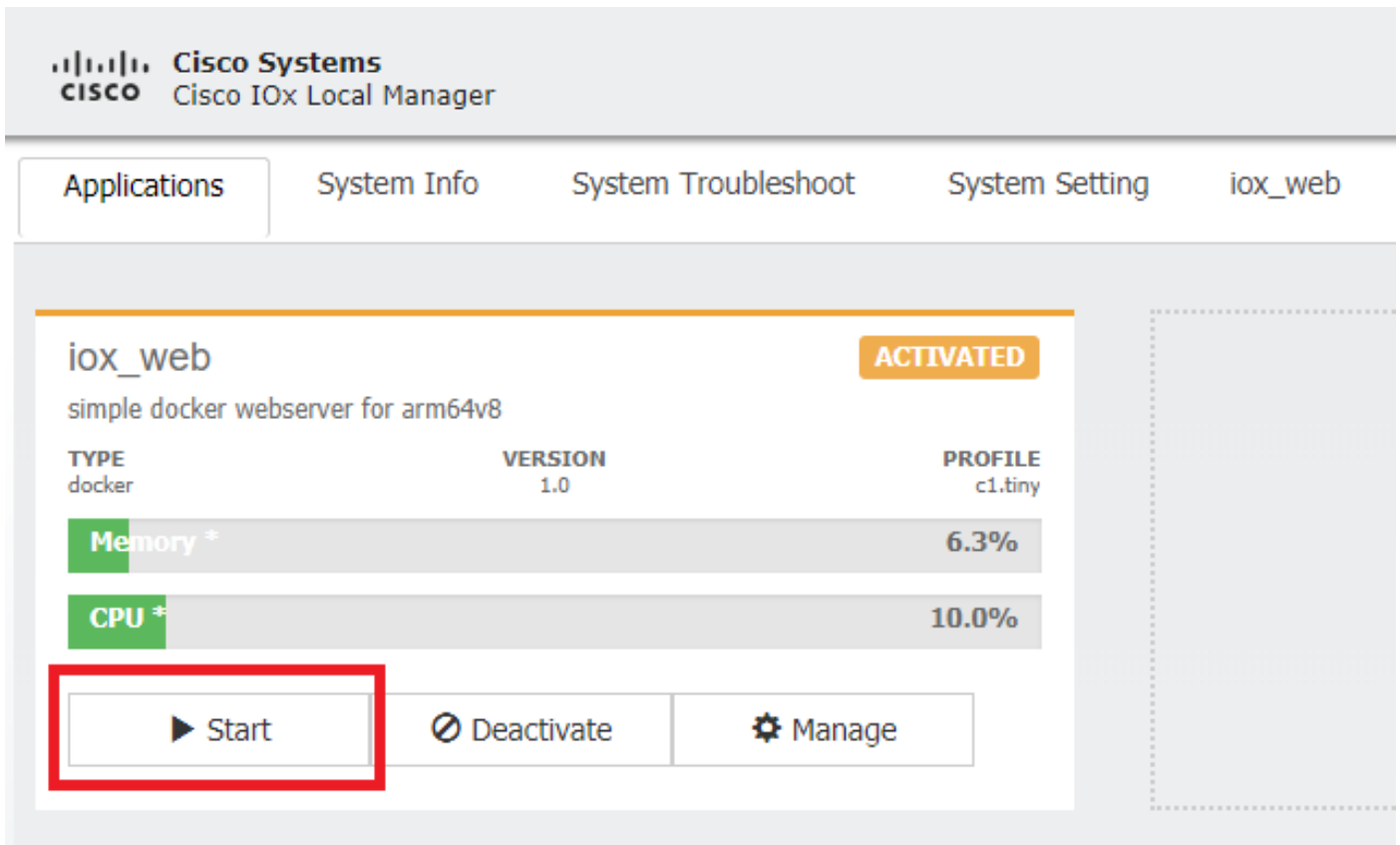
Upgrade

Delete

Na guia **Resources**, abra a configuração da interface para especificar o IP fixo que você deseja atribuir ao aplicativo como mostrado na imagem.



Clique em **OK** e em **Ativar**. Quando a ação for concluída, navegue de volta para a página principal do Local Manager (botão **Aplicativos** no menu superior) e inicie o aplicativo como mostrado na imagem.



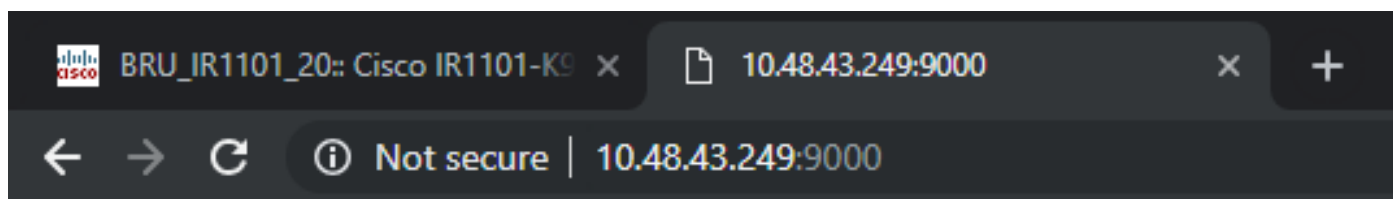
Depois de passar por essas etapas, seu aplicativo deve ser executado e estar disponível através da porta 9000 com o uso da interface Gi 0/0/0 do IR1101.

Verificar

Use esta seção para confirmar se a sua configuração funciona corretamente.

Para verificar, você pode acessar o endereço IP da interface Gi 0/0/0 no IR1101 com o uso da porta 9000.

Se tudo correr bem, você verá isto da seguinte forma, como foi criado no script Python.



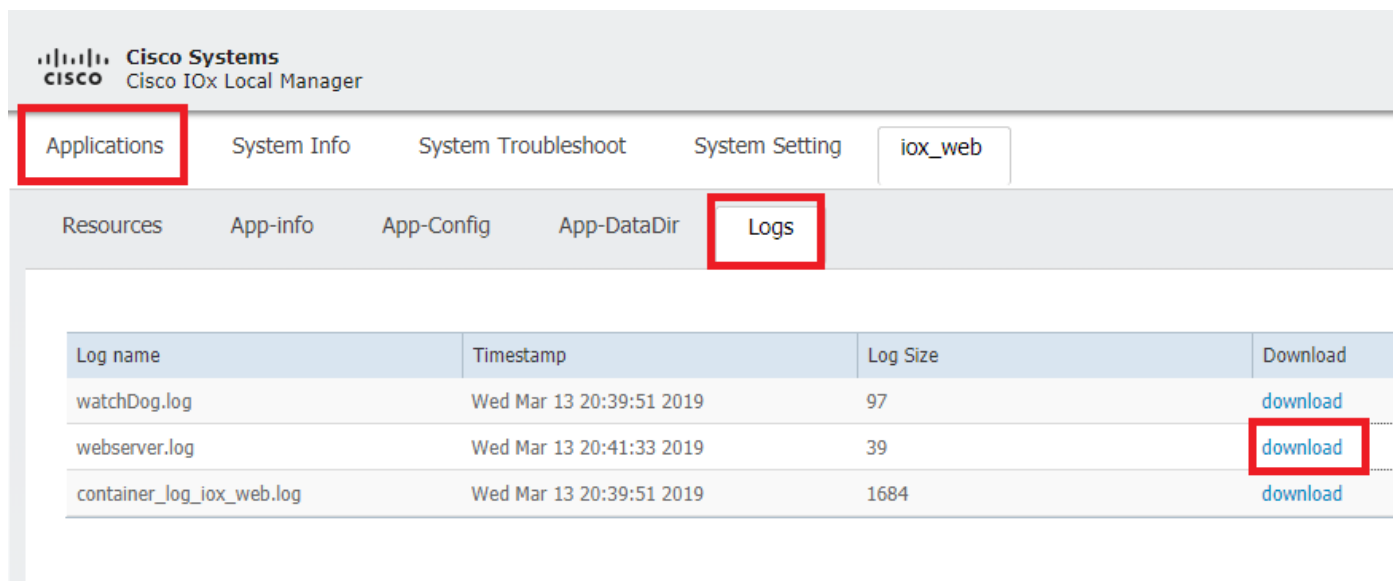
IOX python webserver on arm64v8

Troubleshoot

Esta seção disponibiliza informações para a solução de problemas de configuração.

Para solucionar problemas, você pode verificar o arquivo de registro que você criou no script Python com o uso de um gerenciador local.

Navegue para **Aplicativos**, clique em **Gerenciar** no aplicativo **iox_web** e selecione a guia **Logs** como mostrado na imagem.

A screenshot of the Cisco IOX Local Manager web interface. The top navigation bar shows 'Cisco Systems' and 'Cisco IOX Local Manager'. Below it, there are tabs for 'Applications', 'System Info', 'System Troubleshoot', 'System Setting', and 'iox_web'. The 'Applications' tab is selected and highlighted with a red box. Underneath, there are sub-tabs for 'Resources', 'App-info', 'App-Config', 'App-DataDir', and 'Logs'. The 'Logs' sub-tab is also selected and highlighted with a red box. Below the sub-tabs is a table with columns for 'Log name', 'Timestamp', 'Log Size', and 'Download'. The table contains three rows of log entries, with the 'download' link in the third row highlighted by a red box.

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