

Creación e implementación de un paquete de Ex Docker para la arquitectura ARM IR1101

Contenido

[Introducción](#)

[Prerequisites](#)

[Requirements](#)

[Componentes Utilizados](#)

[Antecedentes](#)

[Configurar](#)

[Parte 1. Cree el paquete IOx para IR1101](#)

[1. Instalación y preparación del cliente IOx en el host Linux](#)

[2. Instalación y preparación del entorno Docker en la máquina de generación de Linux](#)

[3. Instalación de los Paquetes de Emulación de Usuarios de QEMU](#)

[4. Prueba si un contenedor aarch64/ARV64v8 se ejecuta en una máquina x86 Linux](#)

[5. Preparar archivos para crear el contenedor del servidor web Docker](#)

[6. Construir el contenedor del acoplador](#)

[7. Creación del paquete IOx](#)

[Parte 2. Configure el IR1101 para IOx](#)

[1. Habilitar la interfaz web, Ex y el administrador local](#)

[2. Configuración de redes IOx](#)

[Parte 3. Acceda al Local Manager e implemente la aplicación IOx](#)

[Verificación](#)

[Troubleshoot](#)

Introducción

Este documento describe cómo preparar, construir e implementar un paquete IOx basado en Docker para el gateway Internet of Things (IoT) basado en ARM IR1101.

Prerequisites

Requirements

Cisco recomienda que tenga conocimiento sobre estos temas:

- Linux
- Contenedores
- IOx

Componentes Utilizados

La información que contiene este documento se basa en las siguientes versiones de software y

hardware.

- IR1101 que se puede alcanzar a través de Secure Shell (SSH)
Dirección IP configurada Acceso al dispositivo con privilegios 15 usuarios
- Servidor Linux (se utiliza una instalación mínima de Debian 9 (versión) para este artículo)
- Archivos de instalación del cliente IOx que se pueden descargar desde:

<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.

Antecedentes

El IR1101 es un poco diferente en comparación con la mayoría de las otras plataformas IOx, ya que se basan principalmente en x86. El IR1101 se basa en la arquitectura ARM64v8, por lo que no puede implementar contenedores o paquetes IOx construidos para x86 directamente en la plataforma. Este documento comienza desde el principio y prepara el entorno para la construcción de contenedores Docker basados en ARM64v8 y explica cómo construirlos, empaquetarlos e implementarlos en el IR1101 con el uso de un PC x86.

A modo de ejemplo, se utiliza un script Python muy pequeño que es un servidor web simple y se crea un contenedor Docker para empaquetarlo finalmente para que se ejecute en el IR1101. Lo único que el servidor web hará es escuchar en un puerto predefinido (9000) y devolver una página simple cuando reciba una solicitud **GET**. Esto le permite probar la capacidad para ejecutar su propio código y probar el acceso a la red a la aplicación IOx una vez que comience a ejecutarse.

El paquete será construido por las herramientas Docker, con el uso de Alpine Linux. Alpine Linux es una pequeña imagen Linux (alrededor de 5 MB), que a menudo se utiliza como base para los contenedores Docker.

Como la mayoría de las máquinas virtuales/portátiles/de escritorio están basadas en x86, debe emular la arquitectura ARM64v8 en la máquina basada en x86 donde se construye el contenedor. Puede hacerlo fácilmente con el uso de la emulación de usuario de Quick Emulator (QEMU). Esto permite la ejecución de ejecutables en una arquitectura no nativa de la misma manera que se ejecutaría en su arquitectura nativa.

Configurar

Parte 1. Cree el paquete IOx para IR1101

1. Instalación y preparación del cliente IOx en el host Linux

Necesitas **ioxclient** para empaquetar el contenedor Docker como un paquete IOx una vez que se construya, así que preparémoslo primero.

Primero copie o descargue el paquete **ioxclient**. Puede consultarse en:

<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
```

Extraer el paquete:

```
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
```

Agregue la trayectoria a la variable **PATH** para tenerla disponible sin el uso de la ubicación completa. Si reinicia la máquina o conmuta a los usuarios, no olvide repetir este paso:

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

Inicie **ioxclient** por primera vez para crear un perfil obligatorio. Como sólo utilizará **ioxclient** para empaquetar el contenedor Docker, los valores se pueden dejar como predeterminados:

```
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. Instalación y preparación del entorno Docker en la máquina de generación de Linux

Este acoplador se utiliza para generar un contenedor a partir de la imagen base alpine e incluir los archivos necesarios para el caso práctico. Los pasos dados se basan en las guías oficiales de instalación de Docker Community Edition (CE) para Debian:

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

Actualice las listas de paquetes en su equipo:

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

```
Reading package lists... Done
```

Instale las dependencias para utilizar la repo de Docker:

```
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) ...
```

Agregue la clave Docker GNU Privacy Guard (GPG) como una clave GPG válida:

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

Verifique la huella dactilar de la clave 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]
```

Agregue la repo estable de Docker:

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

Actualice de nuevo las listas de paquetes a medida que agrega la repo de Docker:

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

```
...
```

```
Reading package lists... Done
```

Instale el acoplador:

```
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 acceder/ejecutar Docker como usuario normal, agregue este usuario al grupo Docker y actualice la pertenencia al grupo:

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

```
jedepuyd@deb9:~$ newgrp docker
```

3. Instalación de los Paquetes de Emulación de Usuarios de QEMU

Después de instalar Docker, debe instalar los emuladores de usuario de QEMU. Utilice el emulador QEMU enlazado estáticamente desde dentro del contenedor Docker para que pueda ejecutar el contenedor para ARM64v8 en nuestra máquina Linux basada en x86, aunque el contenedor de destino se diseñará para la arquitectura ARM64v8.

Instale los paquetes:

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

Después de la instalación, aquí están los emuladores QEMU enlazados estáticamente disponibles en **/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
...
```

El primero de la lista es el que necesita: **aarch64** es el nombre en arco para ARM64v8 para Linux.

4. Prueba si un contenedor aarch64/ARV64v8 se ejecuta en una máquina x86 Linux

Ahora que tiene Docker y los binarios QEMU necesarios instalados, puede probar si puede ejecutar un contenedor Docker construido para ARM64v8 en el equipo 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 puede ver en el resultado, se obtiene y se hace funcionar un contenedor alpino arm64v8 con acceso al emulador.

Si solicita la arquitectura del contenedor, puede ver que el código se compila para aarch64. Exactamente como el arco de destino para el contenedor debe ser para IR1101.

5. Preparar archivos para crear el contenedor del servidor web Docker

Ahora que se ha hecho toda la preparación, puede continuar y crear los archivos necesarios para el contenedor del servidor web que se debe ejecutar en IR1101.

El primer archivo es **webserver.py**, el script Python que desea ejecutar en el contenedor. Como este es sólo un ejemplo, obviamente, reemplazará esto por el código real para ejecutarse en su aplicación 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 contiene la lógica para escribir en un archivo de registro, que estará disponible para consulta desde el Administrador local.

El segundo archivo que se necesita es el archivo Dockerfile. Esto define cómo se construye el contenedor:

```

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

```

El archivo Dockerfile define cómo se generará el contenedor. Empiece desde la imagen base de Apline para ARM64v8, copie el emulador en el contenedor, ejecute el paquete para agregar el paquete Python y copie el script del servidor web en el contenedor.

La última preparación que se necesita para poder construir el contenedor es copiar qemu-aarch64-static en el directorio desde donde se construirá el contenedor:

```

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

```

6. Construir el contenedor del acoplador

Ahora que se ha realizado toda la preparación, puede generar el contenedor con el uso del archivo 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 prueba, ejecute el contenedor que acaba de generar y verifique si el 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 puede ver en este resultado, la arquitectura del contenedor es el aarch64 objetivo. Y después de iniciar el script, verá que está escuchando las solicitudes en el puerto 9000.

7. Creación del paquete IOx

El envase está listo para ser empaquetado. Antes de poder solicitar a ioxclient que haga esto, primero debe crear el descriptor del paquete: **package.yaml**.

Este archivo describe cómo debe ser el paquete, cuántos recursos necesita ejecutar y qué 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 puede ver, la arquitectura de la CPU se establece en aarch64. Para obtener acceso al puerto TCP 9000, utilice **rootfs.tar** como los rootfs y al inicio, puede ejecutar **python/webserver.py**.

Lo último que hay que hacer antes de empaquetar es extraer **rootfs.tar** del contenedor Docker:

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

En este punto, puede utilizar **ioxclient** para construir el paquete 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
```

En este momento, hay un paquete para implementarlo en el IR1101 listo como **package.tar**. En la siguiente parte se explica cómo preparar el dispositivo para la implementación.

Parte 2. Configure el IR1101 para IOx

1. Habilitar la interfaz web, Ex y el administrador local

Local Manager es una GUI para implementar, activar, iniciar, administrar y resolver problemas de aplicaciones IOx. Para el IR1101, está integrado en la interfaz web de administración regular. Así que primero hay que habilitar eso.

Realice estos pasos en el IR1101 para habilitar IOx y la interfaz 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
```

La última línea agrega un usuario con privilegios de 15 permisos. Este usuario tendrá acceso a la interfaz web y al administrador local IOx.

2. Configuración de redes IOx

Antes de acceder a la interfaz web, agreguemos la configuración necesaria para la red IOx. Puede encontrar información de fondo en la documentación de 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

En resumen, las aplicaciones IOx pueden comunicarse con el mundo exterior con el uso de la interfaz VirtualPortGroup0 (comparable con la Gi2 en las interfaces IR809 y Gi5 en las 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
```

Al configurar la interfaz VirtualPortGroup0 como Traducción de dirección de red (NAT) interna, debe agregar la instrucción `ip nat outside` en la interfaz Gi 0/0/0 para permitir la comunicación hacia y desde las aplicaciones IOx con el 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 el acceso al puerto 9000 para el contenedor, que puede dar 192.168.1.15, necesita agregar un puerto hacia adelante:

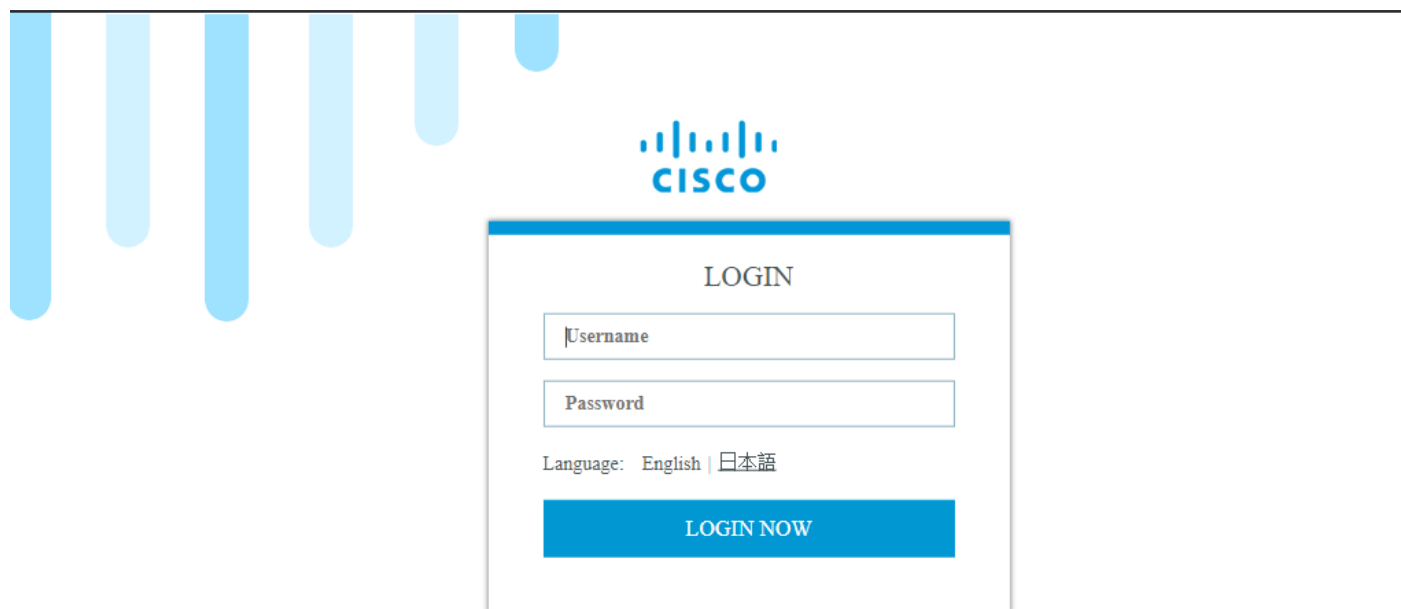
```
BRU_IR1101_20(config)#ip nat inside source static tcp 192.168.1.15 9000 interface
GigabitEthernet0/0/0 9000
```

Para esta guía, utilice IPs estáticamente configuradas por aplicación IOx. Si desea asignar

dinámicamente direcciones IP a las aplicaciones, necesitará agregar la configuración para un servidor DHCP en la subred de VirtualPortGroup0.

Parte 3. Acceda al Local Manager e implemente la aplicación IOx

Después de agregar estas líneas a la configuración, puede acceder al IR1101 con el uso de la interfaz web. Navegue hasta la dirección IP Gi 0/0/0 con el uso de su navegador como se muestra en la imagen.



© 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.

Utilice la cuenta de privilegio 15 creada en el paso 1. para iniciar sesión en la interfaz web y navegar a **Configuration** - IOx como se muestra en la imagen.



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

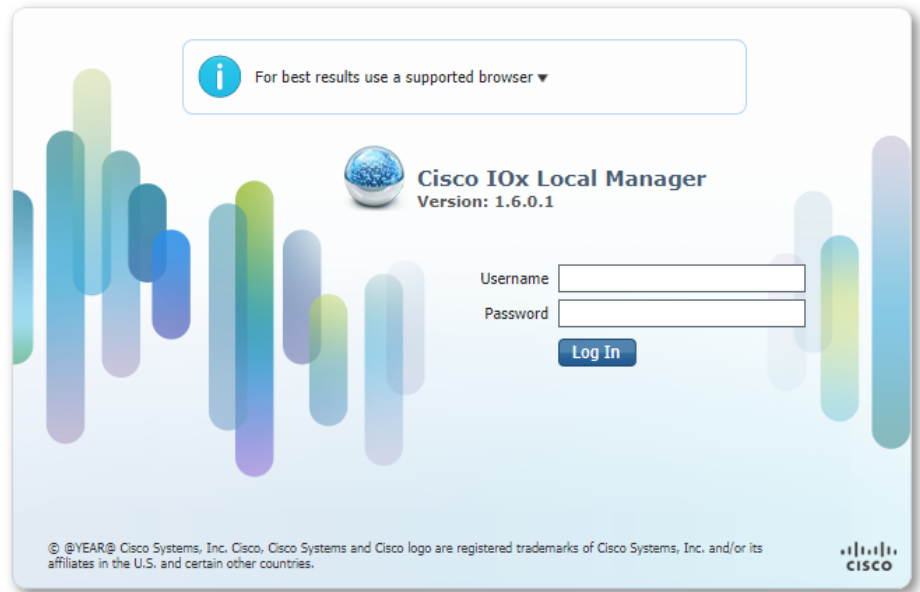
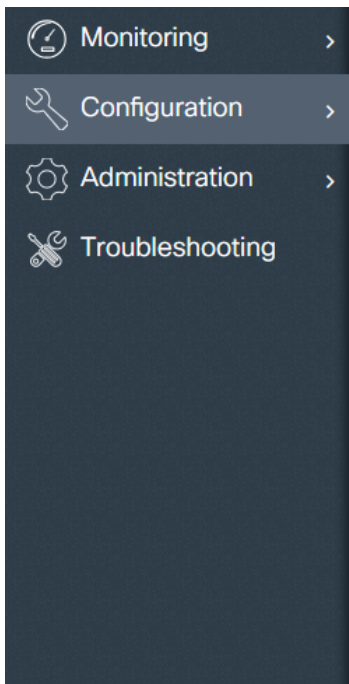
Application Visibility

Custom Application

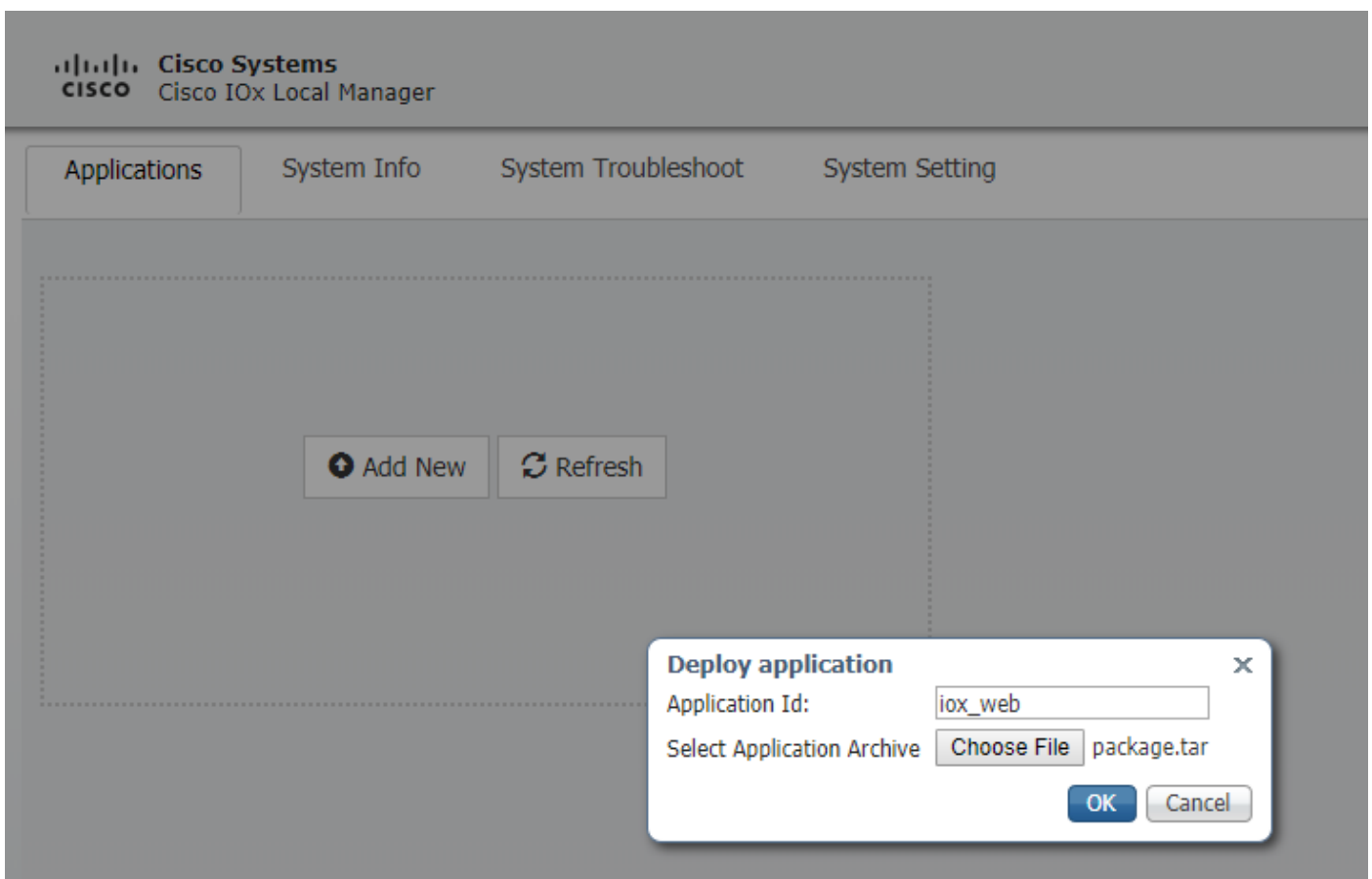
IOx

NETFLOW

En el inicio de sesión de IOSx Local Manager, utilice la misma cuenta para continuar como se muestra en la imagen.



Haga clic en **Add New**, seleccione un nombre para la aplicación IOx y elija el package.tar que se generó en la Parte 1 como se muestra en la imagen.



Una vez cargado el paquete, puede activarlo como se muestra en la imagen.

iox_web

DEPLOYED

simple docker webserver for arm64v8

TYPE	VERSION	PROFILE
docker	1.0	c1.tiny

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

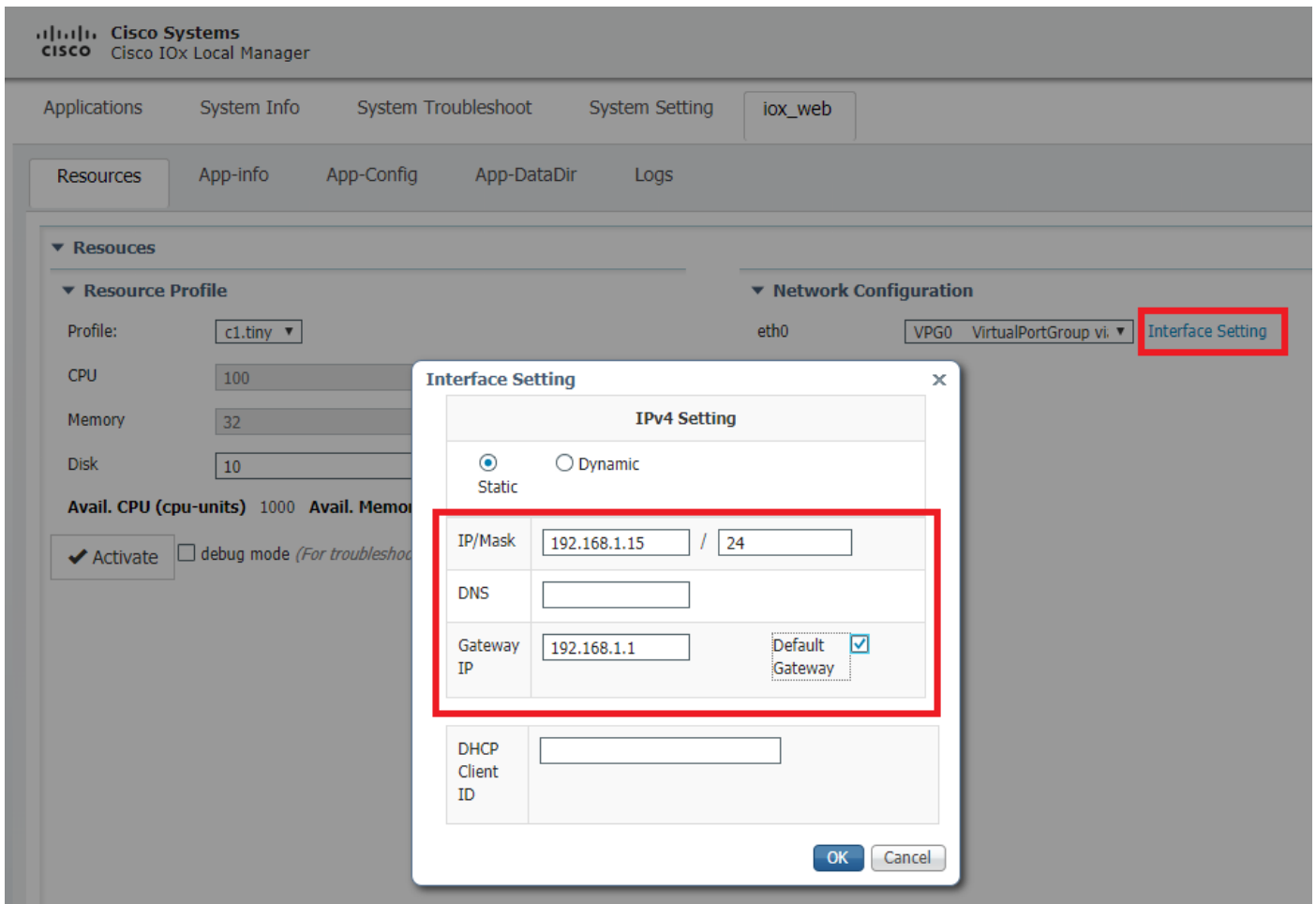
CPU *	10.0%
-------	-------

✓ Activate

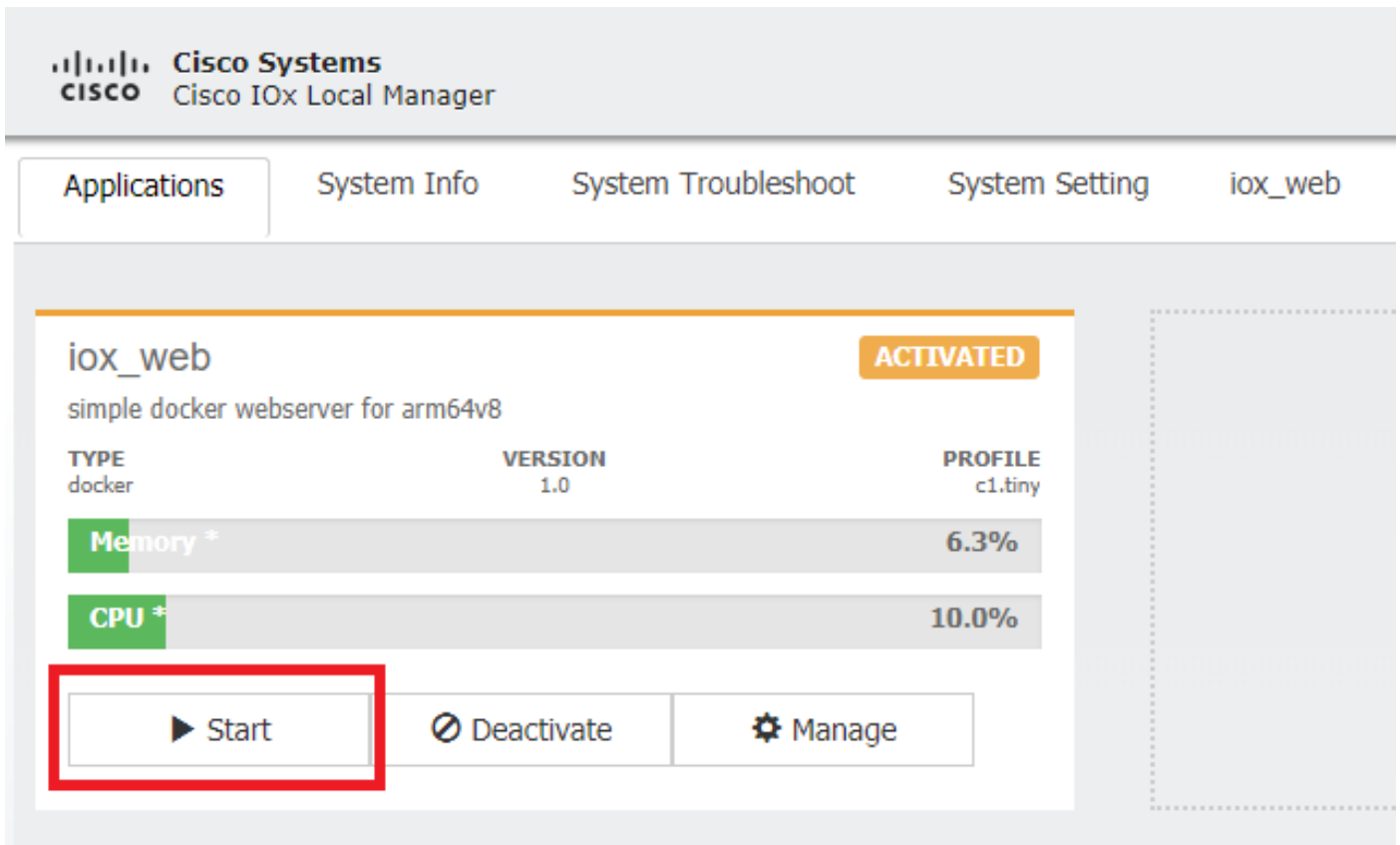
Upgrade

Delete

En la pestaña **Recursos**, abra la configuración de la interfaz para especificar la IP fija que desea asignar a la aplicación como se muestra en la imagen.



Haga clic en **Aceptar** y, a continuación, **Activar**. Una vez finalizada la acción, desplácese a la página principal de Local Manager (botón **Aplicaciones** del menú superior) y, a continuación, inicie la aplicación como se muestra en la imagen.



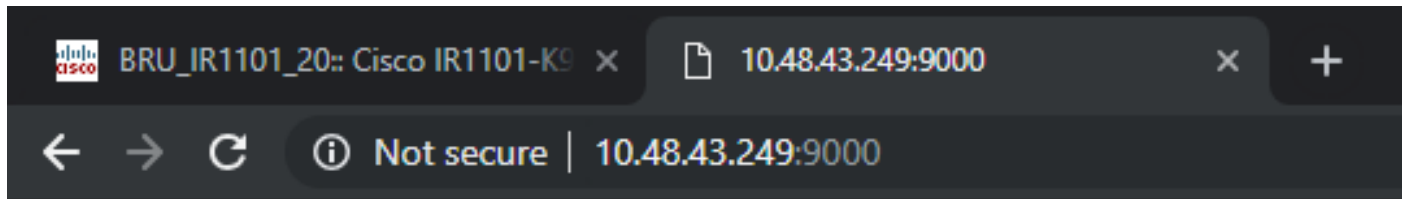
Después de seguir estos pasos, la aplicación debe ejecutarse y estar disponible a través del puerto 9000 con el uso de la interfaz Gi 0/0/0 del IR1101.

Verificación

Utilice esta sección para confirmar que su configuración funcione correctamente.

Para verificar, puede acceder a la dirección IP de la interfaz Gi 0/0/0 en el IR1101 con el uso del puerto 9000.

Si todo va bien, debería ver esto de la siguiente manera, ya que fue creado en el guión de Python.



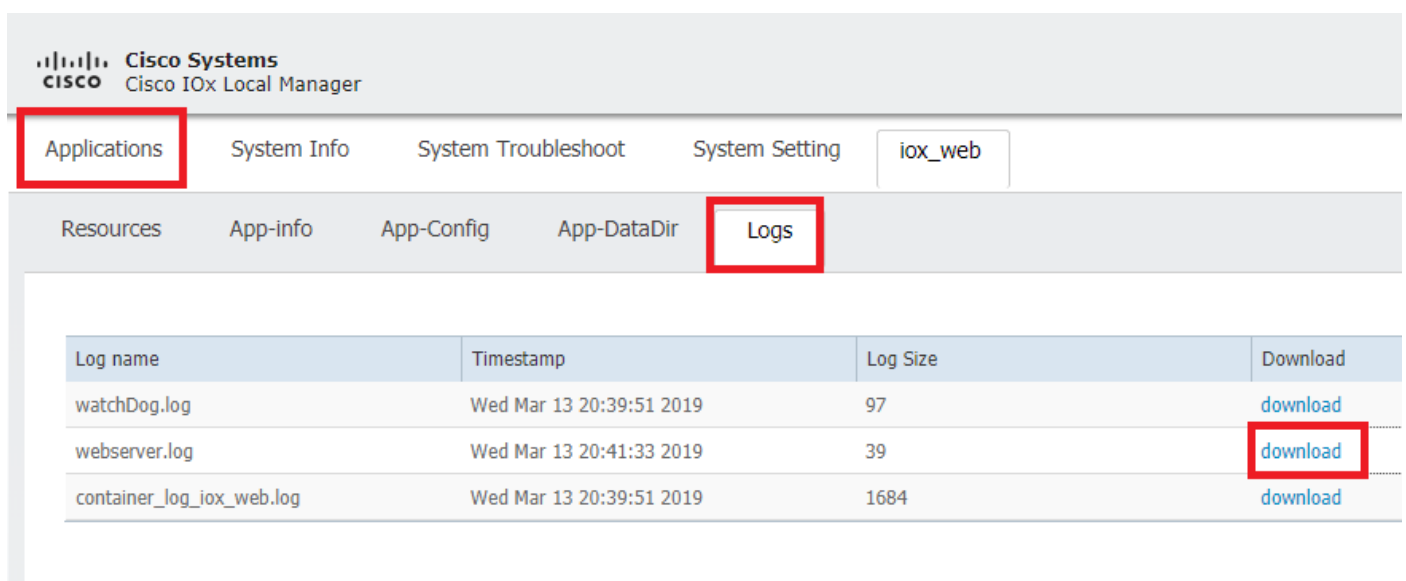
IOX python webserver on arm64v8

Troubleshoot

En esta sección se brinda información que puede utilizar para resolver problemas en su configuración.

Para resolver problemas, puede verificar el archivo de registro que crea en la secuencia de comandos de Python con el uso de un administrador local.

Navegue hasta **Aplicaciones**, haga clic en **Administrar** en la aplicación **iox_web** y, a continuación, seleccione la pestaña **Registros** como se muestra en la imagen.



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