Création et déploiement d'un package Docker IOx pour l'architecture ARM IR1101

Table des matières

Introduction

Conditions préalables

- Exigences
- Composants utilisés

Informations générales

Configurer

- Partie 1. Création du package IOx pour IR1101
- 1. Installation et préparation du client IOx sur l'hôte Linux
- 2. Installation et préparation de l'environnement Docker sur la machine de fabrication Linux
- 3. Installation des packages d'émulation utilisateur QEMU
- 4. Testez si un conteneur arch64/ARV64v8 s'exécute sur une machine Linux x86
- 5. Préparer les fichiers pour construire le conteneur de serveur Web Docker
- 6. Créer le conteneur Docker
- 7. Création du package IOx
- Partie 2. Configuration du routeur IR1101 pour IOx
- 1. Activez l'interface Web, IOx et Local Manager
- 2. Configuration de la mise en réseau IOx
- Partie 3. Accéder à Local Manager et déployer l'application IOx

Vérifier

Dépannage

Introduction

Ce document décrit comment préparer, construire et déployer un package Iox basé sur Docker pour la passerelle IoT (Internet of Things) basée sur ARM IR1101.

Conditions préalables

Exigences

Cisco vous recommande de prendre connaissance des rubriques suivantes :

- Linux
- Conteneurs
- IOx

Composants utilisés

Les informations contenues dans ce document sont basées sur les versions de matériel et de logiciel suivantes :

- IR1101 accessible via Secure Shell (SSH)
  - Adresse IP configurée
  - Accès au périphérique avec un privilège de 15 utilisateurs
- Hôte Linux (une installation minimale de Debian 9 (extensible) est utilisée pour cet article)
- Fichiers d'installation du client IOx téléchargeables à l'adresse suivante : 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.

Informations générales

L'IR1101 est légèrement différent de la plupart des autres plates-formes IOx, car elles sont principalement basées sur x86. L'IR1101 est basé sur l'architecture ARM64v8, de sorte que vous ne pouvez pas déployer directement des conteneurs ou des packages IOx conçus pour x86 sur la plate-forme. Ce document part de zéro et prépare l'environnement pour la construction de conteneurs Docker basés sur ARM64v8 et explique comment les construire, les empaqueter et les déployer sur l'IR1101 à l'aide d'un PC x86.

Par exemple, un très petit script Python qui est un simple serveur Web est utilisé et un conteneur Docker est construit autour pour finalement le paqueter pour l'exécuter sur l'IR1101. La seule chose que fait le serveur web est d'écouter sur un port prédéfini (9000) et de retourner une simple pagev quand il reçoit une requête GET. Cela vous permet de tester la capacité à exécuter votre propre code et de tester l'accès réseau à l'application IOx une fois qu'elle commence à s'exécuter.

Le paquet est construit par les outils Docker, avec l'utilisation d'Alpine Linux. Alpine Linux est une petite image Linux (environ 5 Mo), qui est souvent utilisée comme base pour les conteneurs Docker.

Comme la plupart des ordinateurs de bureau/portables/machines virtuelles sont tous basés sur x86, vous devez émuler l'architecture ARM64v8 sur la machine basée sur x86 où le conteneur est construit. Vous pouvez le faire facilement avec l'utilisation de l'émulation utilisateur Quick Emulator (QEMU). Cela permet l'exécution de fichiers exécutables dans une architecture non native, tout comme il s'exécuterait sur son architecture native.

Configurer

Partie 1. Création du package IOx pour IR1101

1. Installation et préparation du client IOx sur l'hôte Linux

Vous avez besoin de ioxclient afin de conditionner le conteneur Docker en tant que package IOx une fois qu'il est construit, alors préparons-le d'abord.

Commencez par copier ou télécharger le package ioxclient. Il est disponible à l'adresse suivante : 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
```

Extrayez le package :

```
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
```

Ajoutez le chemin d'accès à la variable PATH afin de le rendre disponible sans utiliser l'emplacement complet. Si vous redémarrez la machine ou les utilisateurs du commutateur, n'oubliez pas de répéter cette étape :

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

Lancez ioxclient pour la première fois afin de créer un profil obligatoire. Comme vous n'utilisez ioxclient que pour empaqueter le conteneur Docker, les valeurs peuvent être conservées par défaut :

```
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. Installation et préparation de l'environnement Docker sur la machine de fabrication Linux

Ce Docker est utilisé pour construire un conteneur à partir de l'image de base Alpine et pour inclure les fichiers nécessaires pour l'exemple d'utilisation. Les étapes données sont basées sur les guides d'installation officiels de Docker Community Edition (CE) pour Debian : https://docs.docker.com/install/linux/docker-ce/debian/

Mettez à jour les listes de packages sur votre machine :

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

Installez les dépendances afin d'utiliser le repo Docker :

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

Ajoutez la clé Docker GNU Privacy Guard (GPG) comme clé GPG valide :

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

Vérifiez l'empreinte de la clé GPG installée :

```
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]
```

Ajoutez le repo stable Docker :

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

Mettez à jour à nouveau les listes de packages lorsque vous ajoutez la réserve Docker :

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

Installer 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) ...
```

Pour pouvoir accéder à Docker et l'exécuter en tant qu'utilisateur normal, ajoutez cet utilisateur au groupe Docker et actualisez l'appartenance au groupe :

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

3. Installation des packages d'émulation utilisateur QEMU

Après avoir installé Docker, vous devez installer les émulateurs utilisateur QEMU. Utilisez l'émulateur QEMU lié statiquement depuis le conteneur Docker afin de pouvoir exécuter le conteneur pour ARM64v8 sur notre machine Linux x86, bien que le conteneur cible soit conçu pour l'architecture ARM64v8.

Installez les packages :

```
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) ...
```

Après l'installation, voici les émulateurs QEMU liés statiquement disponibles dans /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
...
```

Le premier dans la liste, est celui dont vous avez besoin : arch64 est le nom d'arche pour ARM64v8 pour Linux.

4. Testez si un conteneur arch64/ARV64v8 s'exécute sur une machine Linux x86

Maintenant que Docker et les binaires QEMU nécessaires sont installés, vous pouvez tester si vous pouvez exécuter un conteneur Docker construit pour ARM64v8 sur la machine x86 :

```
jedepuyd@deb9:~$ docker run -v /usr/bin/qemu-aarch64-static:/usr/bin/qemu-aarch64-static --rm -ti arm64v
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
```

Comme vous pouvez le voir dans la sortie, arm64v8 Alpine container est obtenu et fait pour fonctionner avec l'accès à l'émulateur.

Si vous demandez l'architecture du conteneur, vous pouvez voir que le code est compilé pour aarch64. Exactement comme l'arc cible pour le conteneur doit être pour IR1101.

5. Préparer les fichiers pour construire le conteneur de serveur Web Docker

Maintenant que toute la préparation est terminée, vous pouvez créer les fichiers nécessaires pour le conteneur de serveur Web qui doit être exécuté sur IR1101.

Le premier fichier est webserver.py, le script Python que vous voulez exécuter dans le conteneur. Comme il ne s'agit que d'un exemple, vous remplacez évidemment ceci par le code réel afin de l'exécuter dans votre application 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()
```

Ce code contient la logique permettant d'écrire dans un fichier journal, qui est disponible pour consultation auprès du gestionnaire local.

Le deuxième fichier nécessaire est le fichier Dockerfile. Définit la manière dont le conteneur est construit :

```
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
```

Le Dockerfile définit comment le conteneur est construit. À partir de l'image de base Apline pour

ARM64v8, copiez l'émulateur dans le conteneur, exécutez l'apk afin d'ajouter le package Python et copiez le script webserver dans le conteneur.

La dernière préparation nécessaire avant de pouvoir construire le conteneur est de copier qemu-aarch64-static dans le répertoire à partir duquel vous construisez le conteneur :

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

6. Créer le conteneur Docker

Maintenant que toute la préparation est terminée, vous pouvez construire le conteneur avec l'utilisation du 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
```

En guise de test, exécutez le conteneur que vous venez de créer et vérifiez si le script fonctionne :

```
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 -tlpn
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
```

Comme vous pouvez le voir dans cette sortie, l'architecture du conteneur est l'arch64 cible. Et après avoir démarré le script, vous voyez qu'il écoute les requêtes sur le port 9000.

7. Création du package IOx

Le contenant est prêt à être emballé. Avant de pouvoir demander à ioxclient de le faire, vous devez d'abord créer le descripteur de package : package.yaml.

Ce fichier décrit l'aspect du package, le nombre de ressources nécessaires à son exécution et les éléments à démarrer.

```
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: c1.tiny
    network:
      -
        interface-name: eth0
        ports:
          tcp: ["9000"]

  startup:
    rootfs: rootfs.tar
    target: ["python","/webserver.py"]
```

Comme vous pouvez le voir, l'architecture du processeur est définie sur aarch64. Afin d'obtenir l'accès au port TCP 9000, utilisez rootfs.tar comme racine et au démarrage, vous pouvez exécuter python/webserver.py.

La dernière chose à faire avant de pouvoir empaqueter est d'extraire le rootfs.tar du conteneur Docker :

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

À ce stade, vous pouvez utiliser ioxclient afin de construire le package IOx pour 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 defini
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
```

Actuellement, il y a un paquet à déployer sur l'IR1101 prêt comme package.tar. La partie suivante explique comment préparer le périphérique pour le déploiement.

Partie 2. Configuration du routeur IR1101 pour IOx

1. Activez l'interface Web, IOx et Local Manager

Local Manager est une interface utilisateur graphique permettant de déployer, d'activer, de démarrer, de gérer et de dépanner des applications IOx. Pour le routeur IR1101, il est intégré à l'interface Web de gestion standard. Donc, vous devez d'abord activer cela.

Exécutez ces étapes sur le routeur IR1101 afin d'activer IOx et l'interface 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 dernière ligne ajoute un utilisateur avec des privilèges de 15 autorisations. Cet utilisateur a accès à l'interface Web et au gestionnaire local IOx.

2. Configuration de la mise en réseau IOx

Avant d'accéder à l'interface Web, ajoutons la configuration requise pour la mise en réseau IOx. Des informations de base sont disponibles dans la documentation IR1101 pour IOx : https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b-cisco-ir1101-scg.html

En bref, les applications IOx peuvent communiquer avec le monde extérieur en utilisant l'interface VirtualPortGroup0 (comparable à la Gi2 sur les interfaces IR809 et Gi5 sur les 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
```

Lorsque vous configurez l'interface VirtualPortGroup0 en tant qu'interface NAT (Network Address Translation) interne, vous devez ajouter l'instruction ip nat outside sur l'interface Gi 0/0/0 afin de permettre la communication vers et depuis les applications IOx avec l'utilisation 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
```

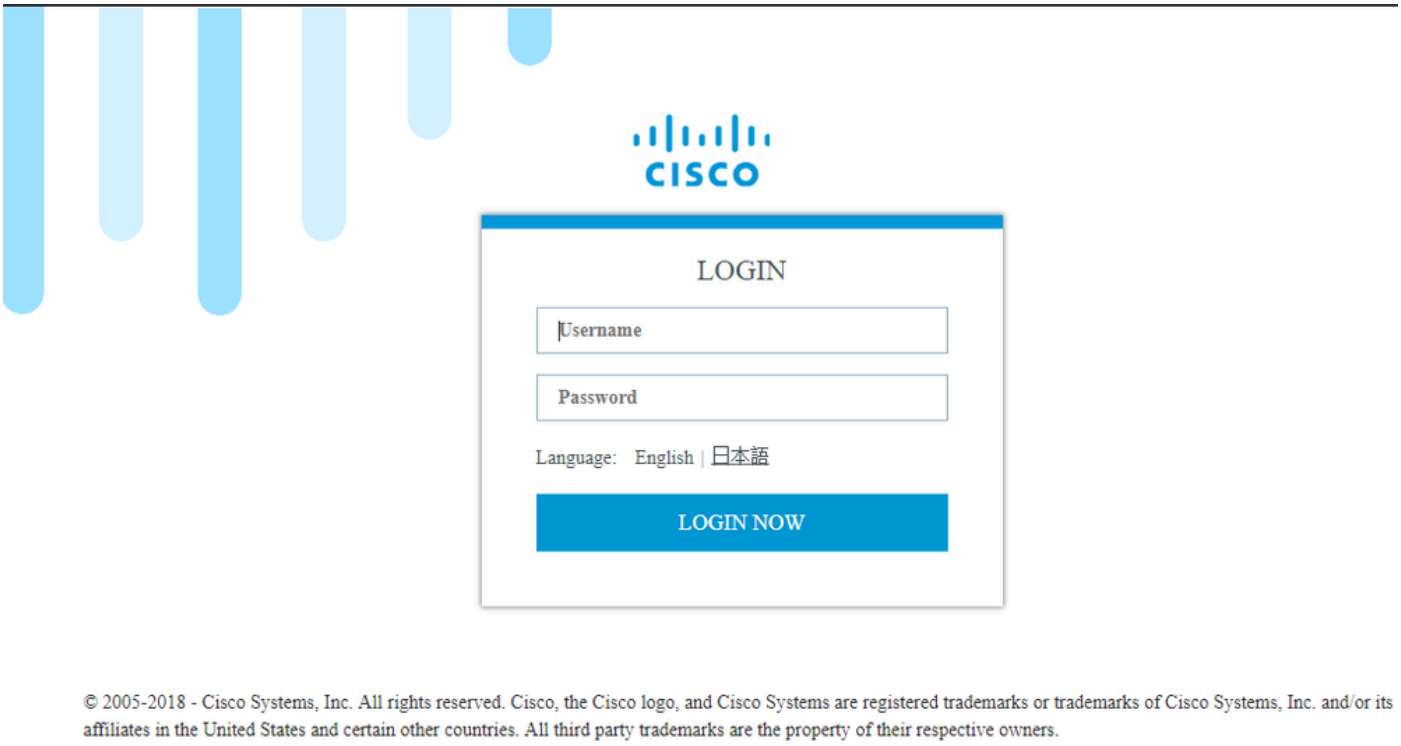
Afin d'autoriser l'accès au port 9000 pour le conteneur, que vous pouvez donner à 192.168.1.15, vous devez ajouter un port de transfert :

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

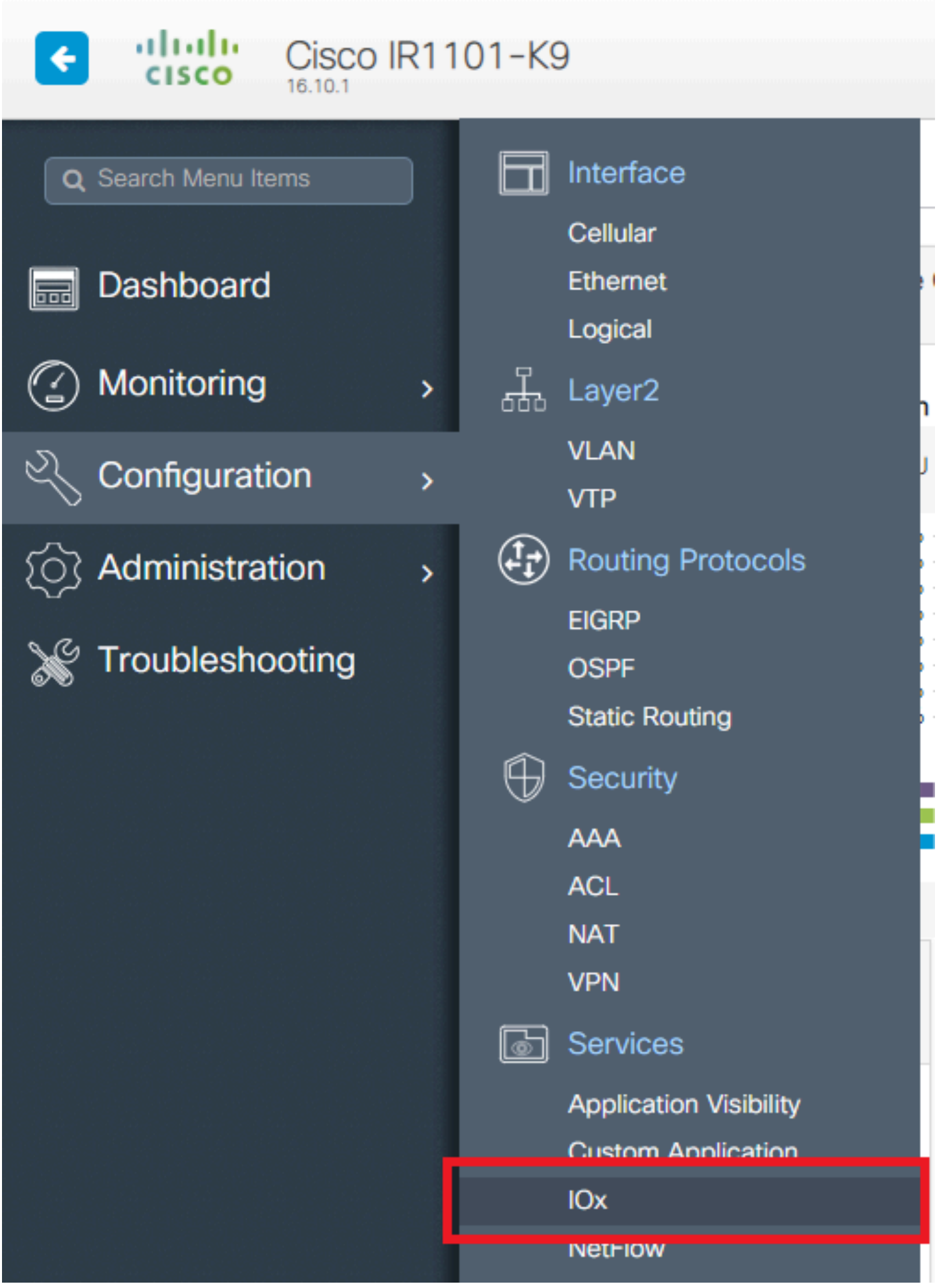
Pour ce guide, utilisez des adresses IP configurées de manière statique par application IOx. Si vous souhaitez attribuer dynamiquement des adresses IP aux applications, vous devez ajouter la configuration d'un serveur DHCP dans le sous-réseau de VirtualPortGroup0.

Partie 3. Accéder à Local Manager et déployer l'application IOx

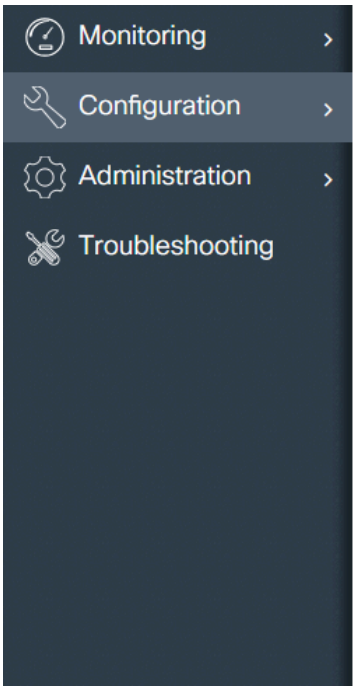
Après avoir ajouté ces lignes à la configuration, vous pouvez accéder à l'IR1101 à l'aide de l'interface Web. Accédez à l'adresse IP Gi 0/0/0 à l'aide de votre navigateur, comme illustré dans l'image.

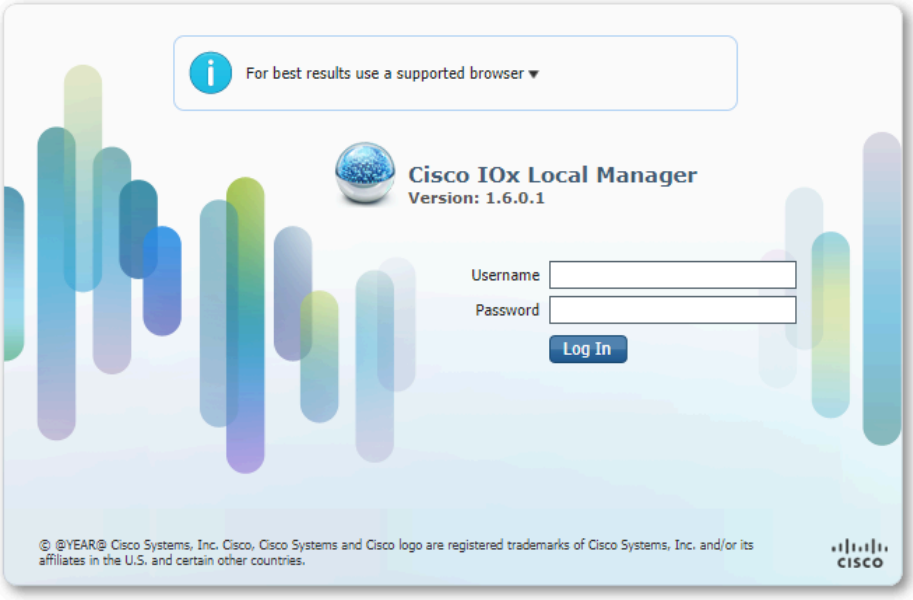


Utilisez le compte de privilège 15 créé à l'étape 1. afin de vous connecter à l'interface Web et naviguer vers Configuration - IOx comme indiqué dans l'image.

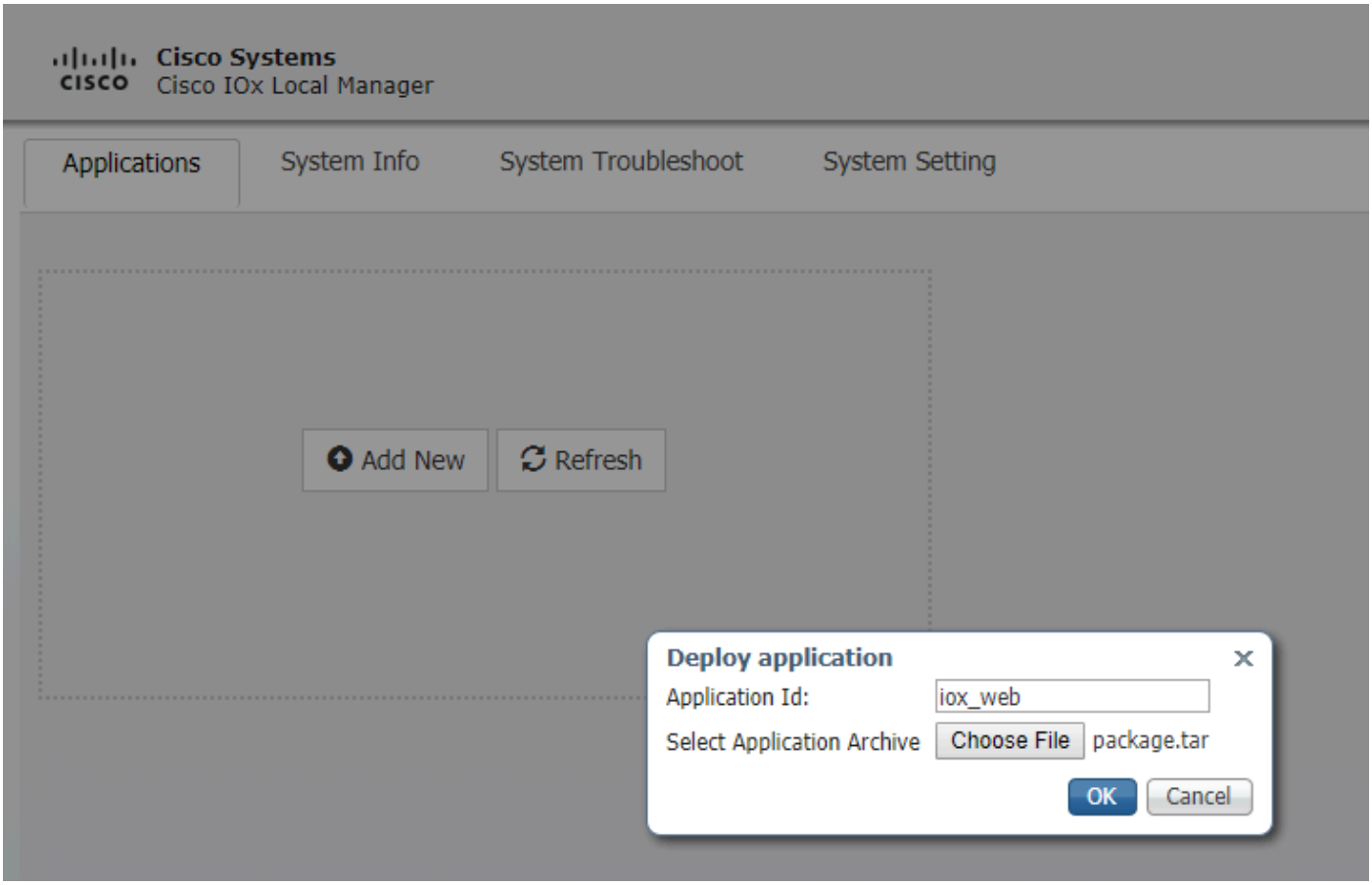

Cisco IR1101-K9
16.10.1
Search Menu Items
Dashboard
Monitoring
Configuration
Administration
Troubleshooting
Interface
Cellular
Ethernet
Logical
Layer2
VLAN
VTP
Routing Protocols
EIGRP
OSPF
Static Routing
Security
AAA
ACL
NAT
VPN
Services
Application Visibility
Custom Application
IOx
NetFlow

Dans l'ouverture de session IOx Local Manager, utilisez le même compte pour continuer comme indiqué dans l'image.

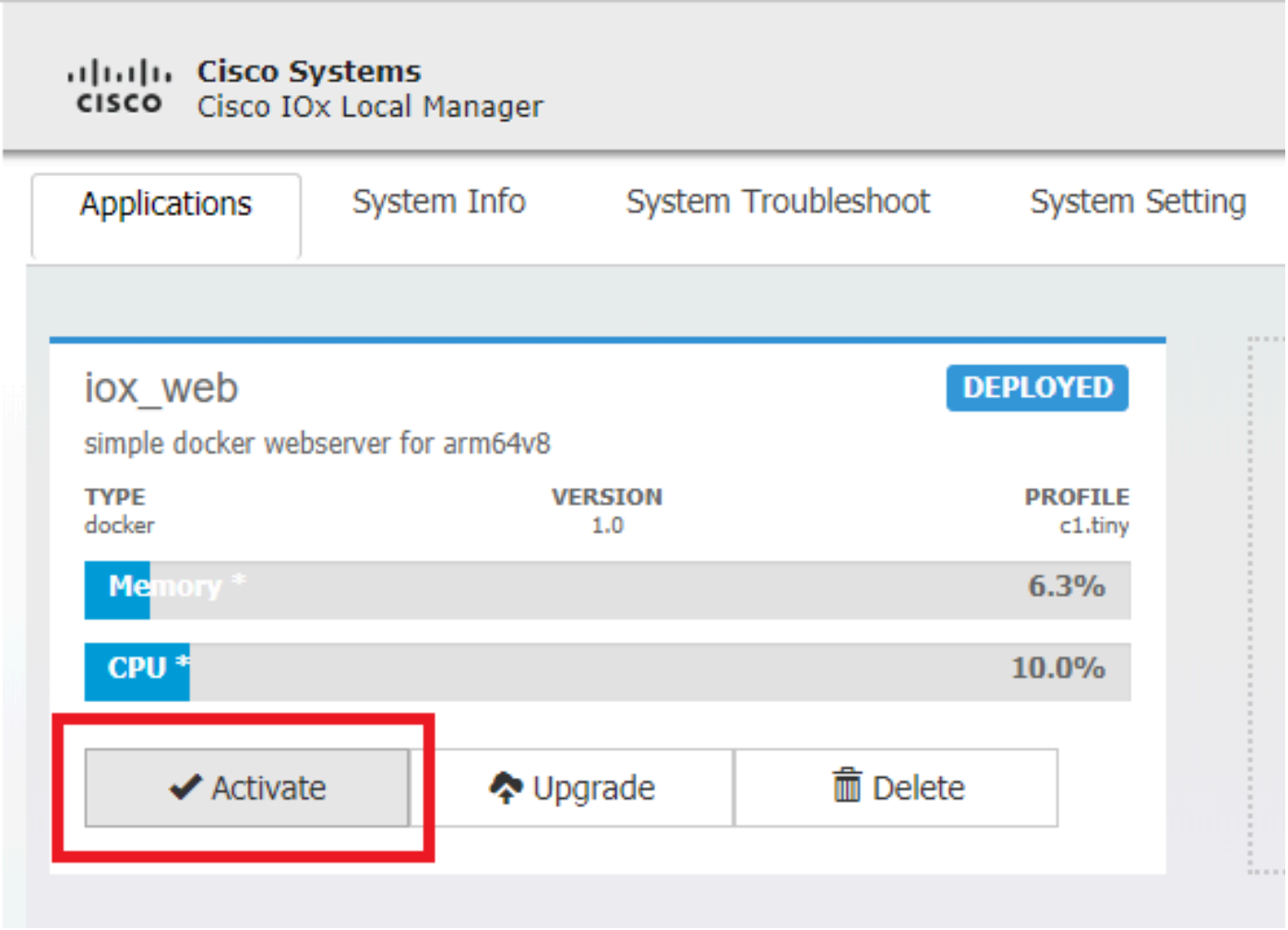




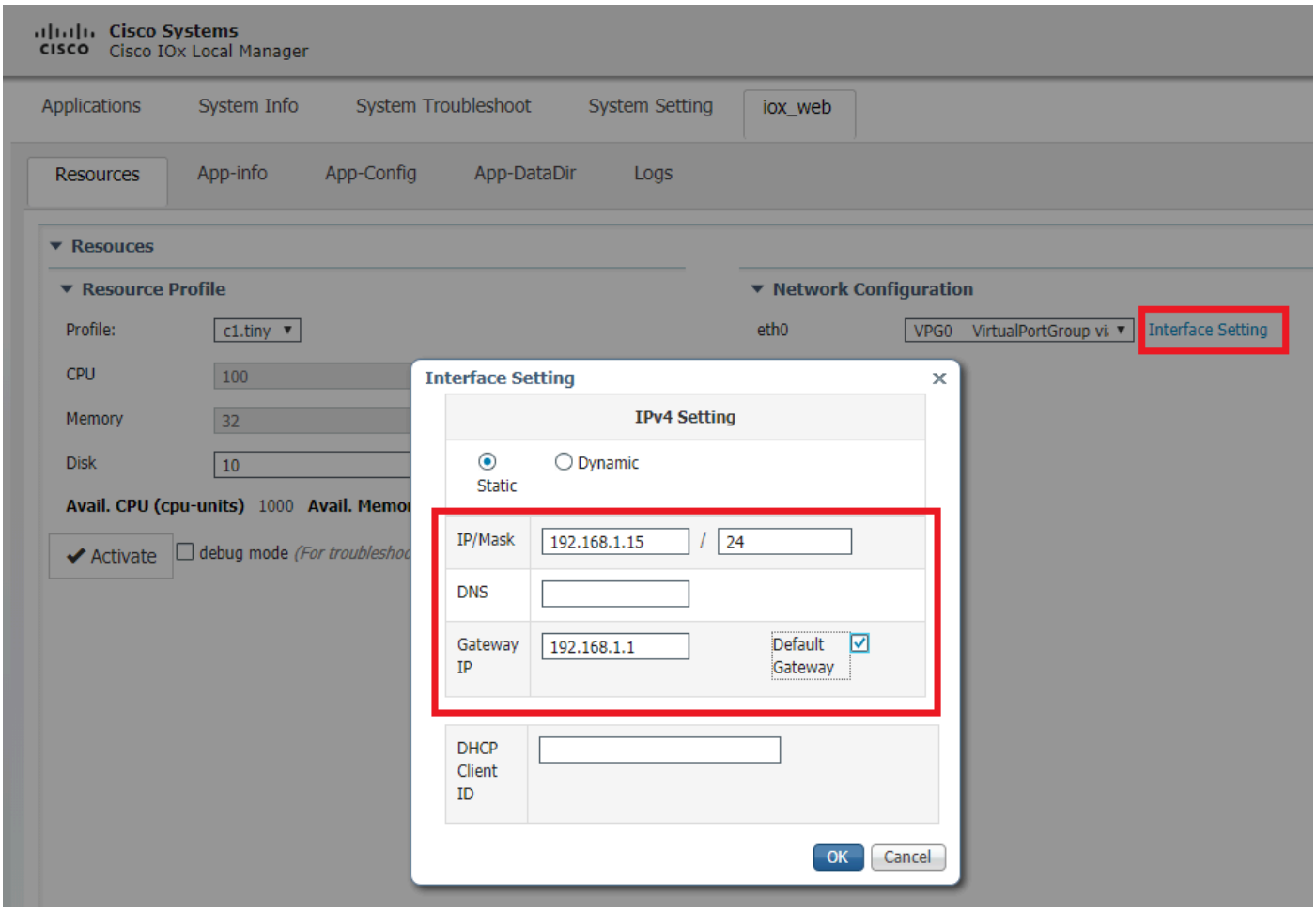
Cliquez sur Add New, sélectionnez un nom pour l'application IOx et choisissez le package.tar qui a été construit dans la Partie 1 comme illustré dans l'image.



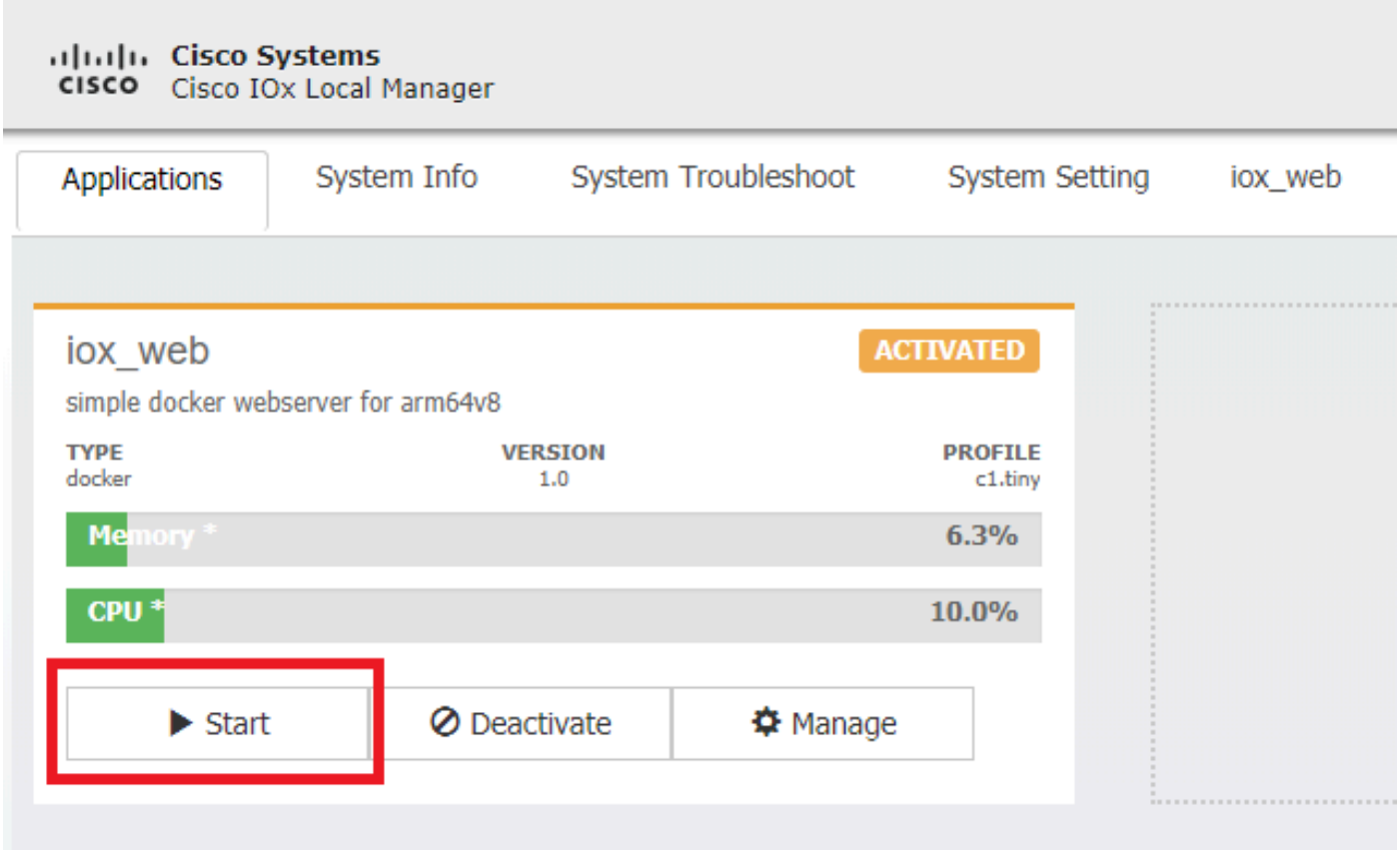
Une fois le package téléchargé, vous pouvez l'activer comme indiqué dans l'image.



Dans l'onglet Resources, ouvrez le paramètre d'interface afin de spécifier l'IP fixe que vous voulez attribuer à l'application comme montré dans l'image.



Cliquez sur OK, puis sur Activate. Une fois l'action terminée, revenez à la page Gestionnaire local principale (bouton Applications dans le menu supérieur), puis démarrez l'application comme indiqué dans l'image.



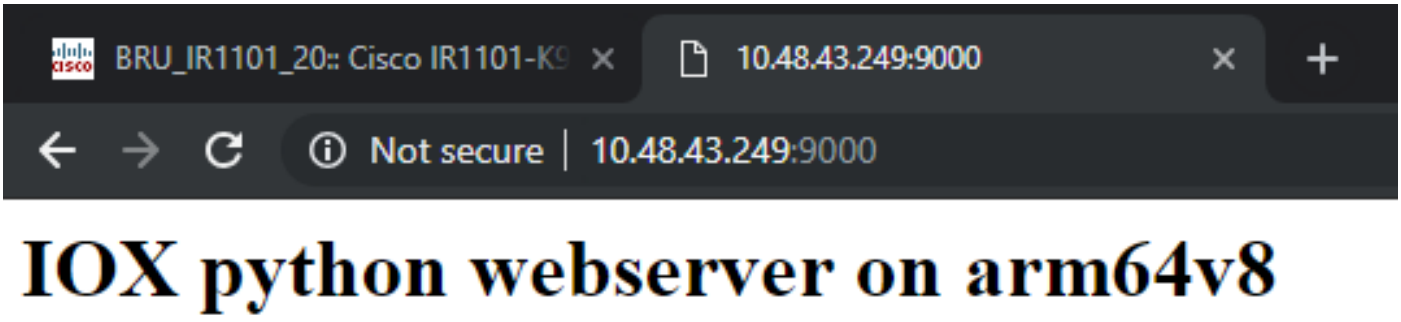
Une fois ces étapes effectuées, votre application s'exécute et est disponible via le port 9000 à l'aide de l'interface Gi 0/0/0 de l'IR1101.

Vérifier

Utilisez cette section pour confirmer que votre configuration fonctionne correctement.

Afin de vérifier, vous pouvez accéder à l'adresse IP de l'interface Gi 0/0/0 sur l'IR1101 en utilisant le port 9000.

Si tout se passe bien, cela montre, comme il a été créé dans le script Python.

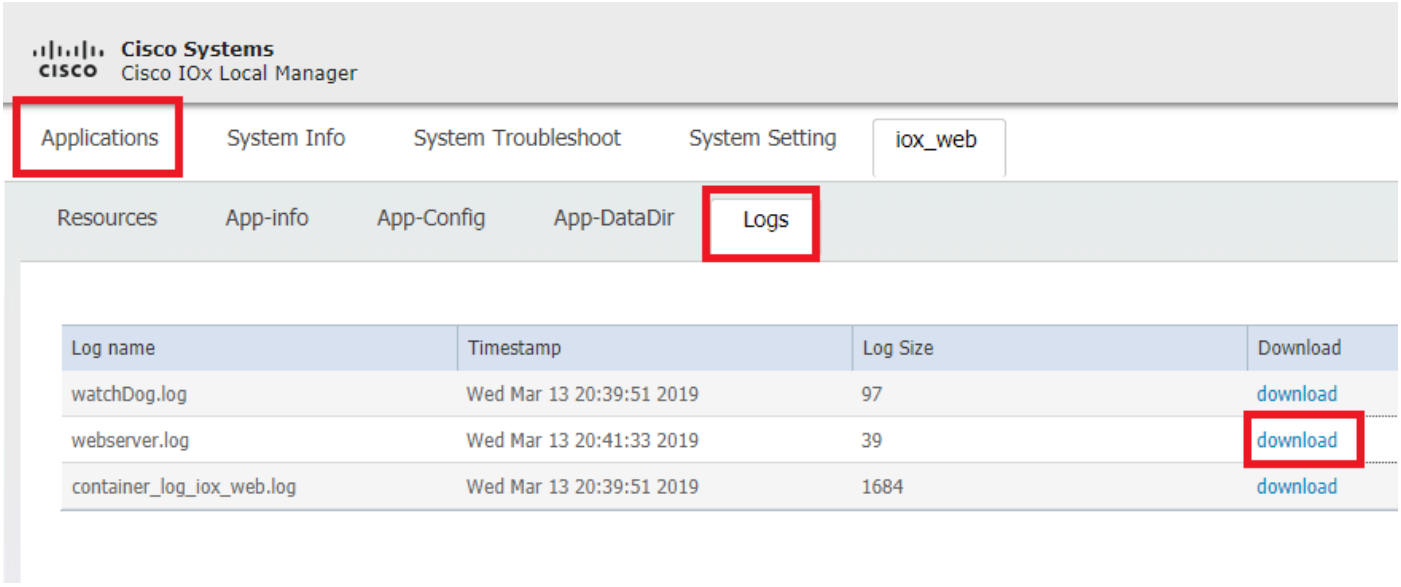


Dépannage

Cette section fournit des informations que vous pouvez utiliser pour dépanner votre configuration.

Afin de dépanner, vous pouvez vérifier le fichier journal que vous créez dans le script Python avec l'utilisation d'un gestionnaire local.

Accédez à Applications, cliquez sur Manage dans l'application iox_web, puis sélectionnez l'onglet Logs comme illustré dans l'image.



À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.