IR1101 ARM 아키텍처용 Docker IOx 패키지 구축 및 구축

목차

소개
사전 요구 사항
요구 사항
사용되는 구성 요소
배경 정보
구성
1부. IR1101용 IOx 패키지 구축
1. Linux 호스트에 IOx 클라이언트 설치 및 준비
2. Linux 빌드 컴퓨터에 Docker 환경 설치 및 준비
3. QEMU 사용자 에뮬레이션 패키지를 설치합니다.
4. arch64/ARV64v8 컨테이너가 x86 Linux 시스템에서 실행되는지 테스트
5. Docker Webserver 컨테이너를 작성하기 위한 파일 준비
6. Docker 컨테이너 작성
7. IOx 패키지 구축
2부. IOx용 IR1101 구성
1. 웹 인터페이스, IOx 및 로컬 관리자를 활성화합니다.
2. IOx 네트워킹 구성
3부. 로컬 관리자에 액세스하고 IOx 애플리케이션 구축
다음을 확인합니다.
문제 해결

소개

이 문서에서는 IR1101 ARM 기반 IoT(Internet of Things) 게이트웨이를 위한 Docker 기반 IOx 패키지를 준비, 구축 및 구축하는 방법에 대해 설명합니다.

사전 요구 사항

요구 사항

다음 주제에 대한 지식을 보유하고 있으면 유용합니다.

- 리눅스
- 컨테이너
- IOx

사용되는 구성 요소

이 문서의 정보는 다음 소프트웨어 및 하드웨어 버전을 기반으로 합니다.

- SSH(Secure Shell)를 통해 연결 가능한 IR1101
  구성된 IP 주소권한 15명의 사용자가 있는 디바이스에 액세스
- 이 문서에 Linux 호스트(최소 Debian 9(확장) 설치가 사용됨)
- https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762
  에서 다운로드할 수 있는 IOx 클라이언트 설치 파일

이 문서의 정보는 특정 랩 환경의 디바이스를 토대로 작성되었습니다.이 문서에 사용된 모든 디바이스는 초기화된(기본) 컨피그레이션으로 시작되었습니다.현재 네트워크가 작동 중인 경우, 모든 명령어의 잠재적인 영향을 미리 숙지하시기 바랍니다.

배경 정보

IR1101은 주로 x86 기반이므로 대부분의 다른 IOx 플랫폼과 비교했을 때 약간 다릅니다.IR1101은 ARM64v8 아키텍처를 기반으로 하므로 플랫폼에 x86용으로 구축된 컨테이너 또는 IOx 패키지를 직접 구축할 수 없습니다.이 문서는 처음부터 시작하여 ARM64v8 기반 Docker 컨테이너를 구축하기 위한 환경을 준비하고 x86 PC를 사용하여 IR1101에 이를 구축, 패키징 및 구축하는 방법에 대해 설명합니다.

예를 들어 간단한 웹 서버인 매우 작은 Python 스크립트가 사용되고 Docker 컨테이너가 IR1101에서 실행되도록 패키지하기 위해 구축됩니다. 웹 서버가 수행할 수 있는 유일한 작업은 미리 정의된 포트(9000)를 듣고 **GET** 요청을 받을 때 단순 페이지를 반환하는 것입니다.이렇게 하면 자체 코드를 실행하는 기능을 테스트할 수 있으며, IOx 애플리케이션이 실행되기 시작하면 네트워크 액세스를 테스트할 수 있습니다.

이 패키지는 Docker 도구를 사용하여 Alpine Linux를 사용하여 구축됩니다.Alpine Linux는 소형 Linux 이미지(약 5MB)로서 Docker 컨테이너의 기반으로 자주 사용됩니다.

대부분의 데스크탑/랩톱/VM은 모두 x86 기반이므로 컨테이너가 구축된 x86 기반 시스템에서 ARM64v8 아키텍처를 에뮬레이션해야 합니다.QEMU(Quick Emulator) 사용자 에뮬레이션을 사용하면 쉽게 이 작업을 수행할 수 있습니다.이렇게 하면 네이티브 아키텍처에서 실행되는 것과 마찬가지로 네이티브 아키텍처에서 실행 파일을 실행할 수 있습니다.

구성

1부. IR1101용 IOx 패키지 구축

1. Linux 호스트에 IOx 클라이언트 설치 및 준비

Docker 컨테이너를 IOx 패키지로 패키징하려면 먼저 ioxclient가 필요합니다.

먼저 ioxclient 패키지를 복사하거나 다운로드합니다
.https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762에서 사용할 수 있습니다.

```
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
```

패키지를 추출합니다.

```
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
```

전체 위치를 사용하지 않고 사용할 수 있도록 **PATH** 변수에 경로를 추가합니다.시스템 또는 스위치 사용자를 재부팅하는 경우 다음 단계를 반복합니다.

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

필수 프로필을 생성하려면 처음으로 ioxclient를 실행합니다.Docker 컨테이너를 패키지하는 데 ioxclient만 사용하므로 값을 기본값으로 유지할 수 있습니다.

```
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 빌드 컴퓨터에 Docker 환경 설치 및 준비

이 Docker는 알파인 기본 이미지에서 컨테이너를 만들고 활용 사례에 필요한 파일을 포함하는 데 사용됩니다.제공된 단계는 Debian용 Docker Community Edition(CE)의 공식 설치 가이드를 기반으로 합니다. https://docs.docker.com/install/linux/docker-ce/debian/

컴퓨터에서 패키지 목록을 업데이트합니다.

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

Docker 보고서를 사용하려면 종속성을 설치합니다.

```
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) ...
```
Docker GPG(GNU Privacy Guard) 키를 유효한 GPG 키로 추가합니다.

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

```
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 안정된 보고서를 추가합니다.

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

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

```
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) ...
```
Docker를 일반 사용자로 액세스/실행하려면 이 사용자를 Docker 그룹에 추가하고 그룹 구성원 자격을 새로 고치십시오.

```
jedepuyd@deb9:~$ sudo usermod -a -G docker jedepuyd
jedepuyd@deb9:~$ newgrp docker
```
3. QEMU 사용자 에뮬레이션 패키지를 설치합니다.

Docker를 설치한 후 QEMU 사용자 에뮬레이터를 설치해야 합니다.대상 컨테이너는 ARM64v8 아키텍처용으로 설계되지만 Docker 컨테이너 내에서 고정으로 연결된 QEMU 에뮬레이터를 사용하여 x86 기반 Linux 컴퓨터에서 ARM64v8용 컨테이너를 실행할 수 있습니다.

패키지를 설치합니다.

```
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) ...
```
설치 후 **/usr/bin**에서 사용할 수 있는 정적으로 연결된 QEMU 에뮬레이터는 다음과 같습니다.

```
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
...
```

목록의 첫 번째 항목은 다음과 같습니다.aarch64는 Linux용 ARM64v8의 arch 이름입니다.

4. arch64/ARV64v8 컨테이너가 x86 Linux 시스템에서 실행되는지 테스트

이제 Docker 및 필요한 QEMU 바이너리가 설치되었으므로 x86 시스템에서 ARM64v8용으로 빌드된 Docker 컨테이너를 실행할 수 있는지 테스트할 수 있습니다.

```
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:a50c0cd3b4112904618459196 3a7a76822777736258e5ade8445b07c88bfdcc3
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
```

출력에서 볼 수 있듯이 arm64v8 Alpine 컨테이너는 에뮬레이터에 대한 액세스 권한을 가지고 실행되도록 만들어집니다.

컨테이너의 아키텍처를 요청하면 코드는 arch64에 대해 컴파일됩니다. 컨테이너의 대상 검색과 정확히 동일해야 IR1101이 됩니다.

5. Docker Webserver 컨테이너를 작성하기 위한 파일 준비

이제 모든 준비가 완료되었으므로 IR1101에서 실행해야 하는 웹 서버 컨테이너에 필요한 파일을 만들 수 있습니다.

첫 번째 파일은 컨테이너에서 실행할 Python 스크립트인 webserver.py입니다.이 예시가 바로 예시이므로 IOx 애플리케이션에서 실행하기 위해 이 코드를 실제 코드로 교체해야 합니다.

```
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()
```

이 코드에는 로그 파일에 쓰기 위한 논리가 포함되어 있으며, 이 로직은 로컬 관리자에서 참조할 수 있습니다.

두 번째 파일은 Dockerfile입니다.컨테이너의 구축 방법을 정의합니다.

```
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
```

Dockerfile은 컨테이너가 작성되는 방법을 정의합니다.ARM64v8의 Apline 기본 이미지에서 시작하여 컨테이너에서 에뮬레이터를 복사하고 apk를 실행하여 Python 패키지를 추가하고 웹 서버 스크립트를 컨테이너에 복사합니다.

컨테이너를 빌드하기 전에 필요한 마지막 준비는 qemu-arch64-static을 컨테이너를 빌드할 디렉토리에 복사하는 것입니다.

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

6. Docker 컨테이너 작성

모든 준비가 완료되었으므로 Dockerfile을 사용하여 컨테이너를 작성할 수 있습니다.

```
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
```

테스트로 방금 빌드한 컨테이너를 실행하고 스크립트가 작동하는지 확인합니다.

```
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
```

이 출력에서 볼 수 있듯이 컨테이너의 아키텍처는 대상 aarch64입니다. 스크립트를 시작하면 포트 9000에서 요청을 수신합니다.

7. IOx 패키지 구축

컨테이너가 포장될 준비가 되었습니다.ioxclient에 이 작업을 요청하려면 먼저 패키지 설명자를 생성해야 합니다.**package.yaml**.

이 파일은 패키지가 어떻게 표시되어야 하는지, 패키지를 실행해야 하는 리소스 수 및 시작할 작업에 대해 설명합니다.

```
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"]
```

보시다시피 CPU 아키텍처는 aarch64로 설정되어 있습니다. TCP 포트 9000에 액세스하려면 rootfs.tar를 루트프로 사용하고 시작할 때 python/webserver.py을 실행할 수 **있습니다**.

패키지하기 전에 해야 할 마지막 작업은 Docker 컨테이너에서 rootfs.tar를 추출하는 것입니다.

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

이 시점에서 IR1101용 IOx 패키지를 구축하기 위해 iox client를 사용할 수 있습니다.

```
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
```

현재 package.tar로 준비된 IR1101에 구축할 패키지가 있습니다.다음 부분에서는 구축을 위해 디바이스를 준비하는 방법에 대해 설명합니다.

2부. IOx용 IR1101 구성

1. 웹 인터페이스, IOx 및 로컬 관리자를 활성화합니다.

Local Manager는 IOx 애플리케이션을 구축, 활성화, 시작, 관리 및 트러블슈팅하기 위한 GUI입니다.IR1101의 경우 일반 관리 웹 인터페이스에 포함됩니다.먼저 이를 활성화해야 합니다.

IOx 및 웹 인터페이스를 활성화하려면 IR1101에서 다음 단계를 수행합니다.

```
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
```

마지막 행은 권한 15의 사용자를 추가합니다.이 사용자는 웹 인터페이스 및 IOx 로컬 관리자에 액세스할 수 있습니다.

2. IOx 네트워킹 구성

웹 인터페이스에 액세스하기 전에 IOx 네트워킹에 필요한 컨피그레이션을 추가하겠습니다.배경 정보는 IOx용 IR1101 문서에서 찾을 수 있습니다.
https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b_IR1101config/b_IR1101config_chapter_010001.html

간단히 말해, IOx 애플리케이션은 VirtualPortGroup0 인터페이스(IR809의 Gi2 및 IR829 인터페이스의 Gi5와 비교)를 사용하여 외부 세계와 통신할 수 있습니다.

```
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 인터페이스를 내부 NAT(Network Address Translation)로 구성할 때 NAT를 사용하여 IOx 애플리케이션과의 통신을 허용하려면 Gi 0/0/0 인터페이스에 ip nat outside 문을 추가해야 합니다.

```
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
```

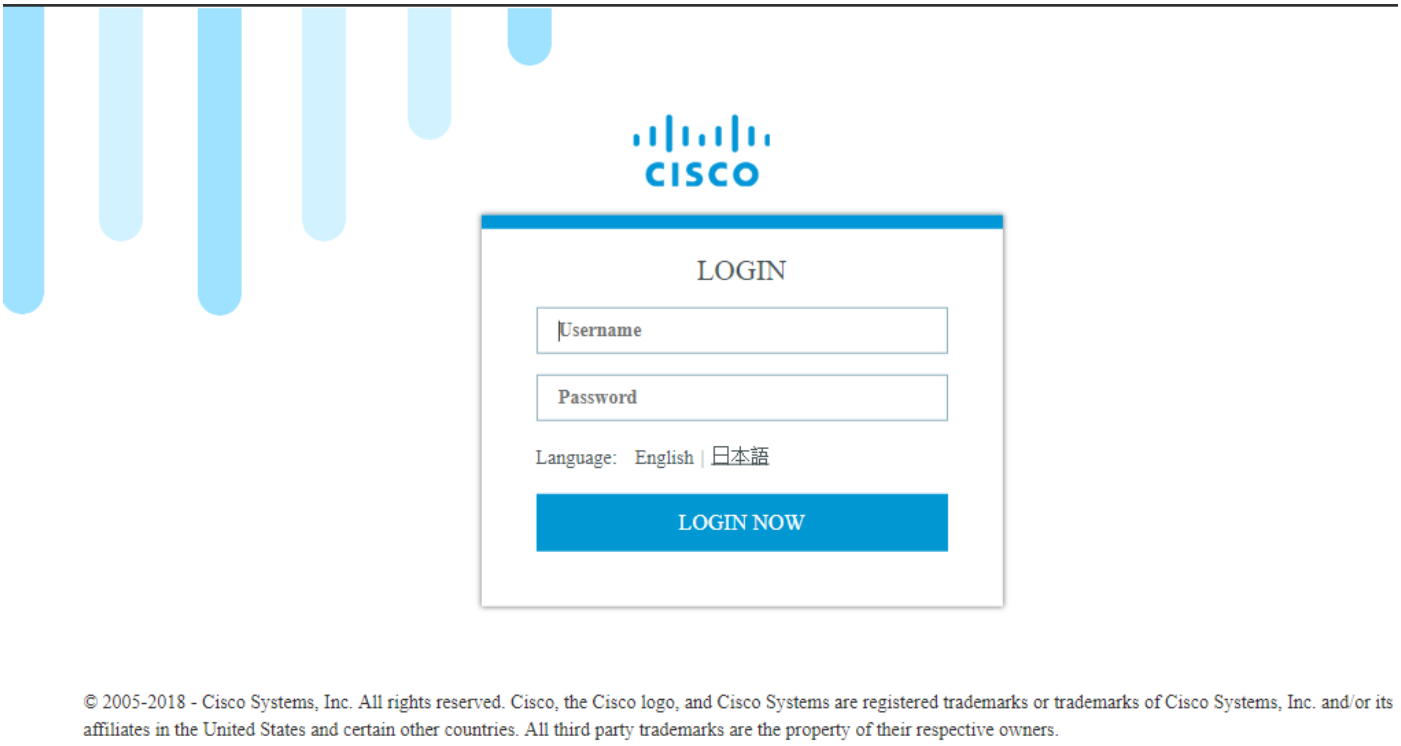
컨테이너의 포트 9000에 대한 액세스를 허용하려면 192.168.1.15을 제공하려면 포워드를 추가해야 합니다.

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

이 가이드에서는 IOx 애플리케이션당 정적으로 구성된 IP를 사용합니다.애플리케이션에 IP 주소를 동적으로 할당하려면 VirtualPortGroup0 서브넷의 DHCP 서버에 대한 컨피그레이션을 추가해야 합니다.

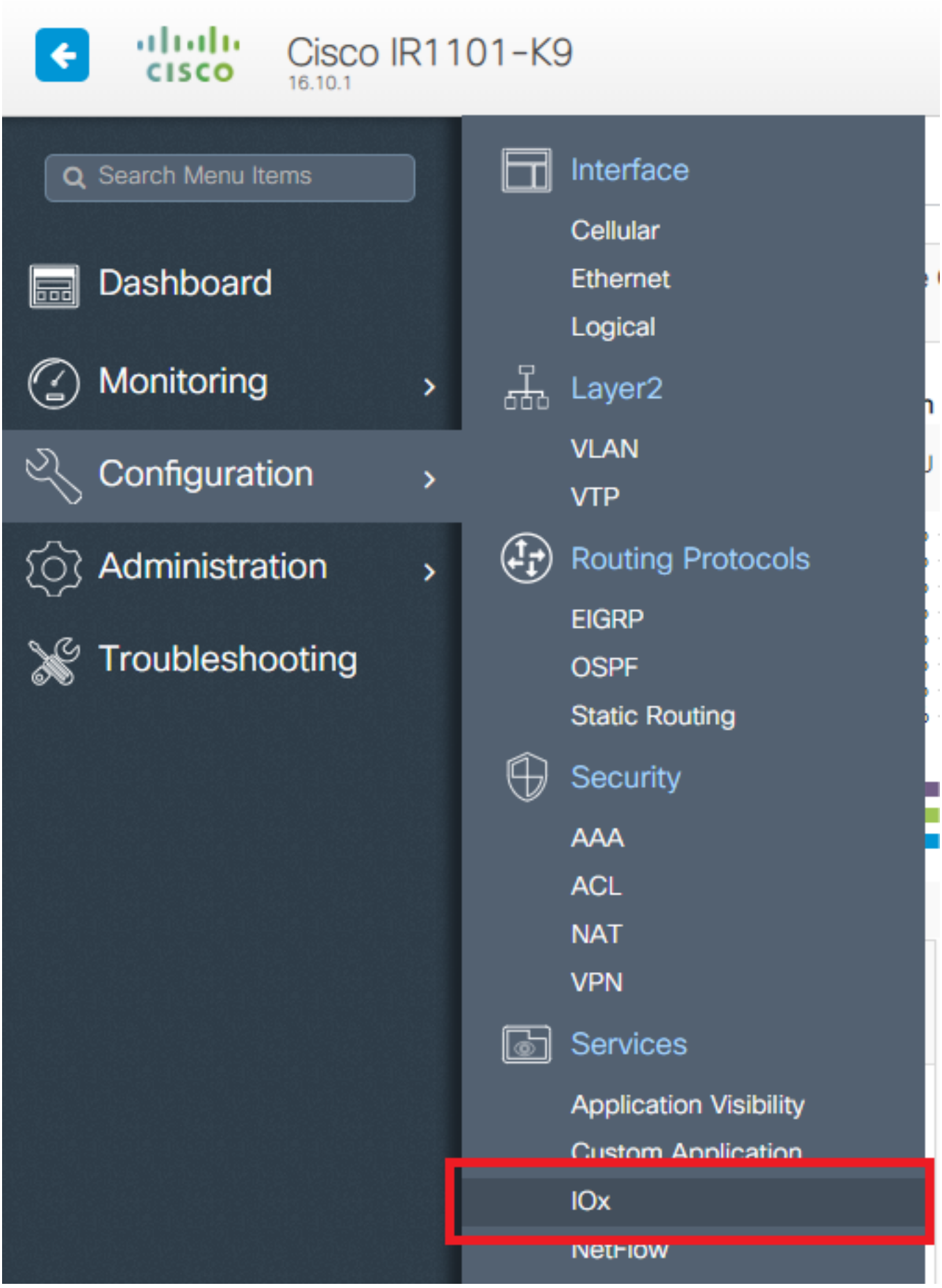
3부. 로컬 관리자에 액세스하고 IOx 애플리케이션 구축

이러한 행을 구성에 추가한 후에는 웹 인터페이스를 사용하여 IR1101에 액세스할 수 있습니다.이미지에 표시된 대로 브라우저를 사용하여 Gi 0/0/0 IP 주소로 이동합니다.

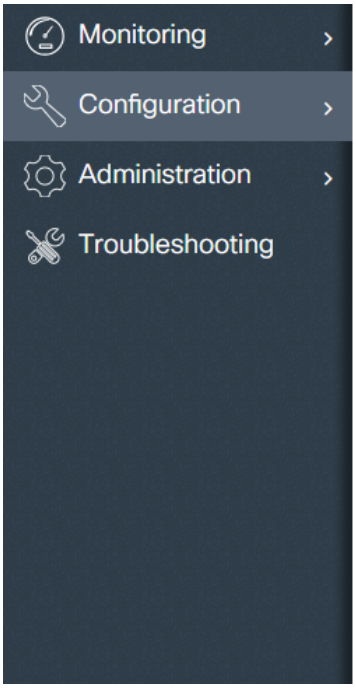


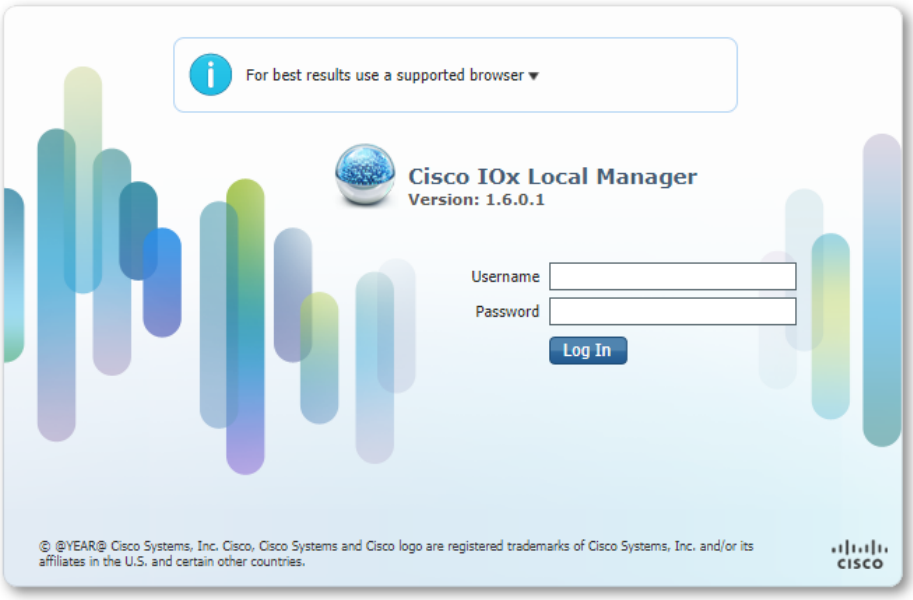
© 2005-2018 - Cisco Systems, Inc. All rights reserved. Cisco, the Cisco logo, and Cisco Systems are registered trademarks or trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries. All third party trademarks are the property of their respective owners.

1단계에서 생성한 권한 15 어카운트를 사용하여 웹 인터페이스에 로그인하고 이미지에 표시된 **Configuration** - IOx로 이동합니다.

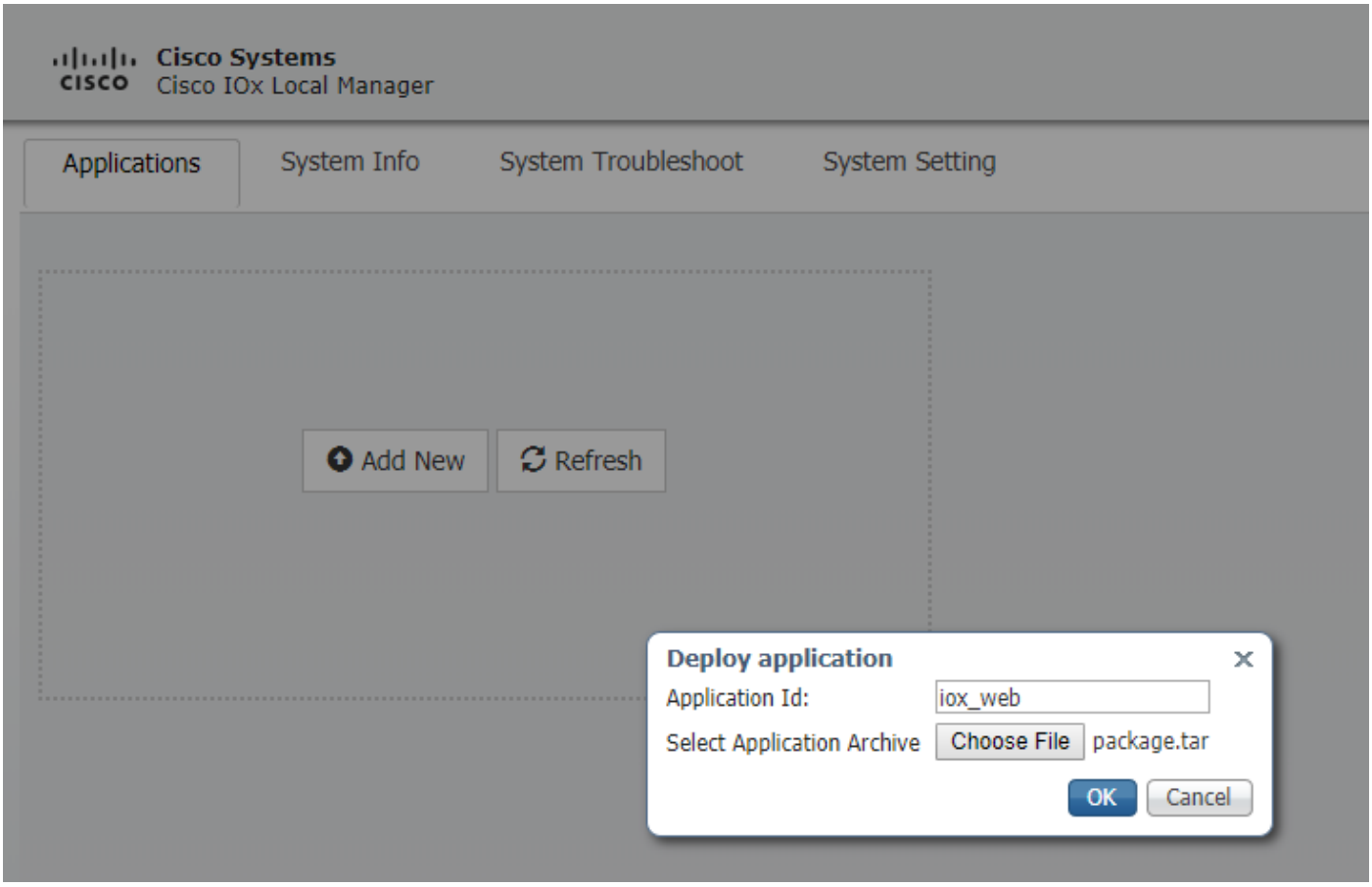


IOx 로컬 관리자 로그인에서 동일한 계정을 사용하여 이미지에 표시된 대로 계속합니다.

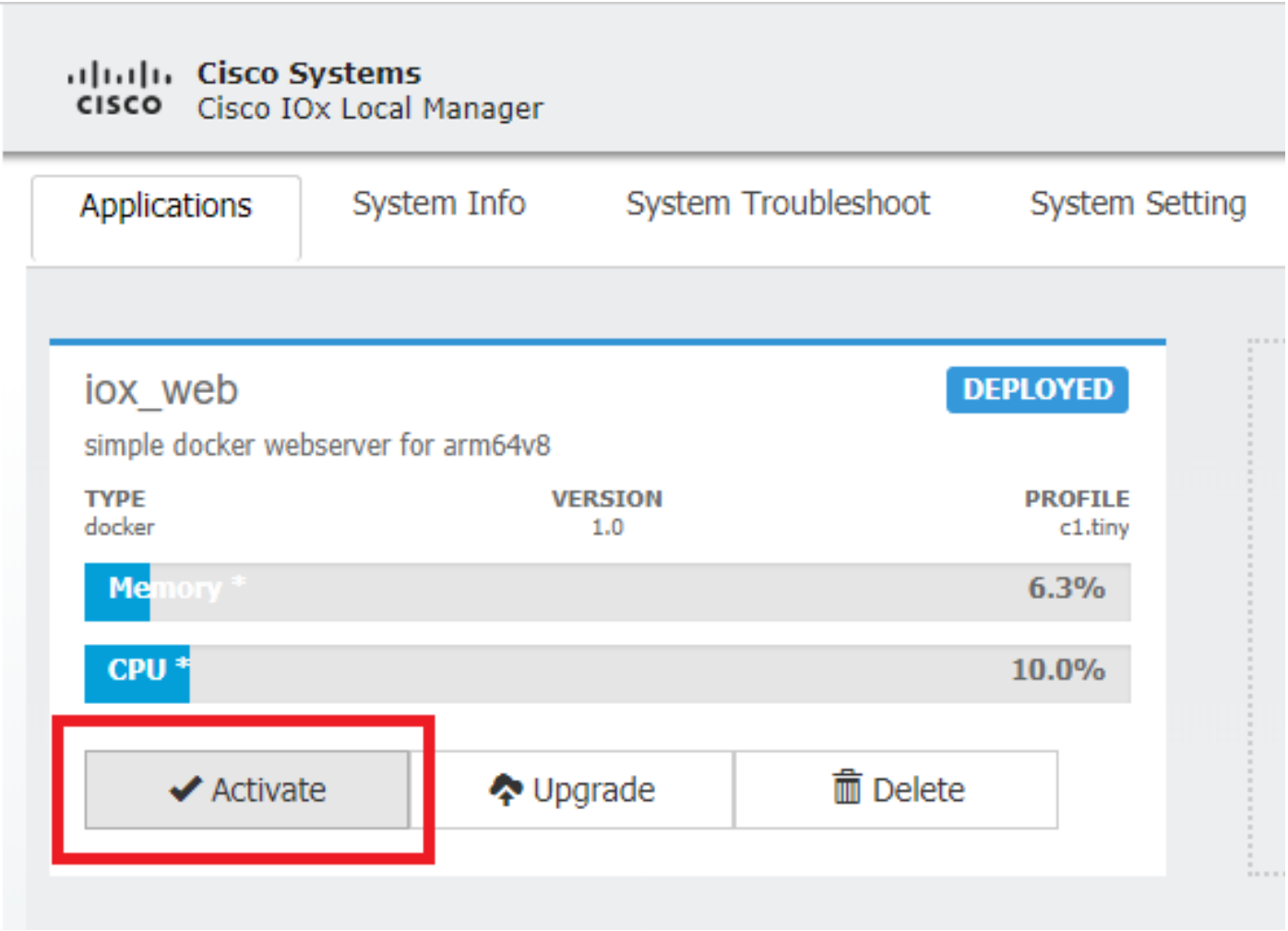




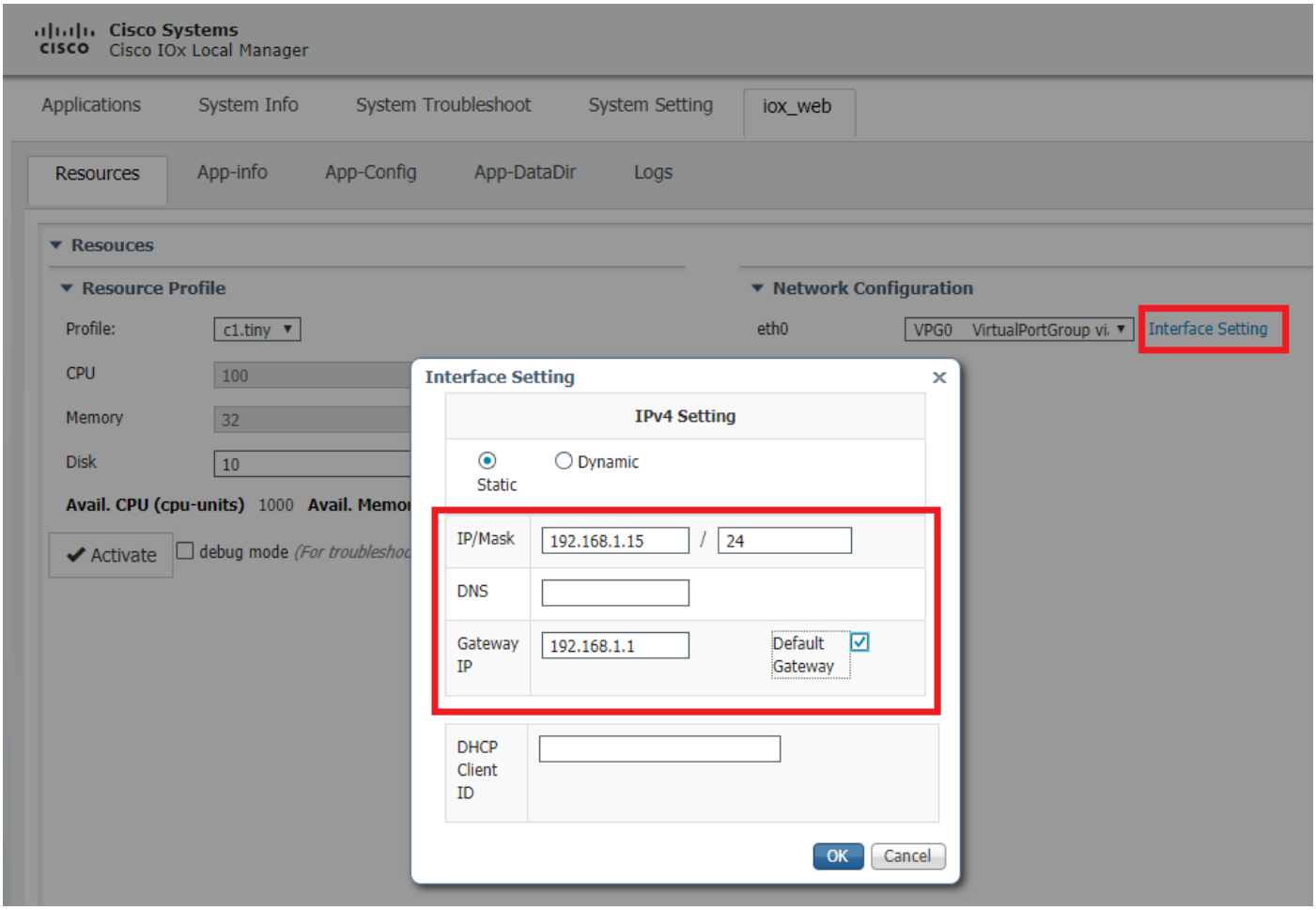
Add **New(새로 추가)**를 클릭하고 IOx 애플리케이션의 이름을 선택하고 이미지에 표시된 대로 Part 1에 빌드된 package.tar를 선택합니다.



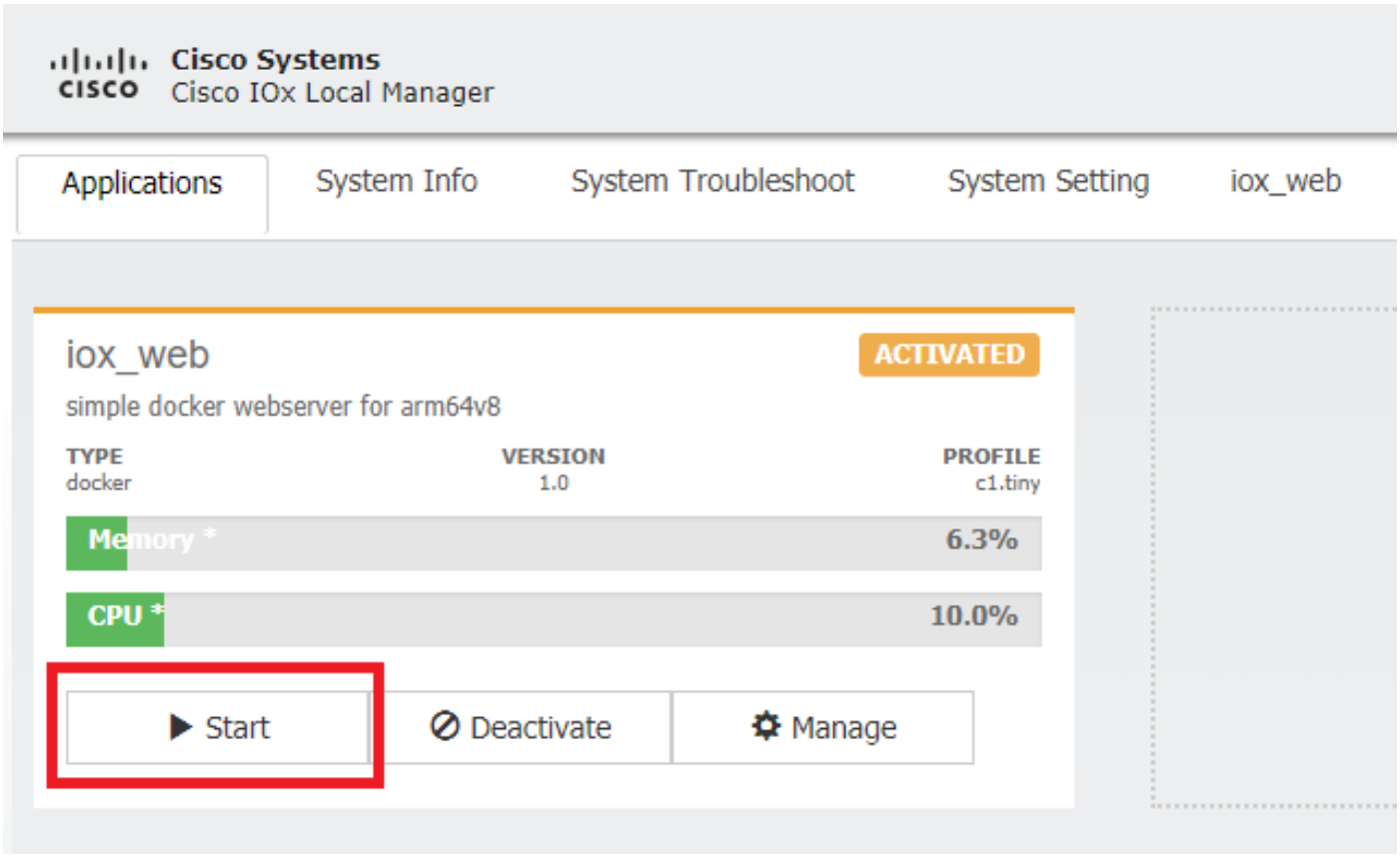
패키지가 업로드되면 이미지에 표시된 대로 활성화할 수 있습니다.



Resources(리소스) 탭에서 인터페이스 설정을 열어 이미지에 표시된 대로 앱에 할당할 고정 IP를 지정합니다.



확인을 클릭한 다음 **활성화를 클릭합니다**.작업이 완료되면 기본 Local Manager 페이지(상단 메뉴의 **Applications** 버튼)로 돌아가 이미지에 표시된 대로 애플리케이션을 시작합니다.



이러한 단계를 거친 후 IR1101의 Gi 0/0/0 인터페이스를 사용하여 애플리케이션을 실행하고 포트

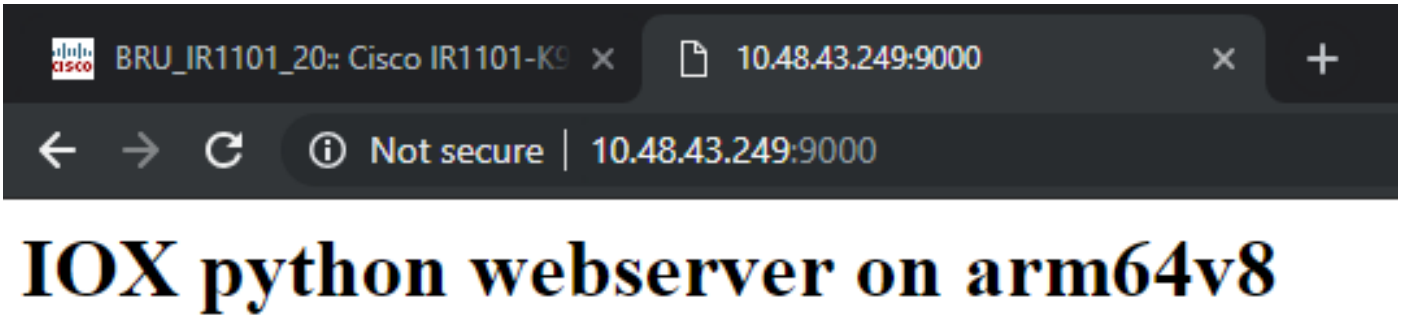
9000을 통해 사용할 수 있어야 합니다.

다음을 확인합니다.

이 섹션을 사용하여 컨피그레이션이 제대로 작동하는지 확인합니다.

확인하려면 포트 9000을 사용하여 IR1101에서 Gi 0/0/0 인터페이스의 IP 주소에 액세스할 수 있습니다.

모든 작업이 제대로 진행되면 Python 스크립트에서 만든 것처럼 다음과 같이 표시되어야 합니다.



문제 해결

이 섹션에서는 컨피그레이션 문제를 해결하는 데 사용할 수 있는 정보를 제공합니다.

문제를 해결하려면 로컬 관리자를 사용하여 Python 스크립트에서 만든 로그 파일을 확인할 수 있습니다.

Applications(**애플리케이션)로** 이동하고 **Manage**(관리)를 **클릭하여 iox_web** 애플리케이션에서 **Manage(관리)를** 클릭한 다음 이미지에 표시된 대로 Logs(로그) 탭을 선택합니다.