为IR1101 ARM架构构建和部署Docker IOx软件包

目录

简介

本文档介绍如何为基于IR1101 ARM的物联网(IoT)网关准备、构建和部署基于Docker的IOx软件包。

先决条件

要求

Cisco 建议您了解以下主题：

- Linux
- 容器
- IOx

使用的组件

本文档中的信息基于以下软件和硬件版本：

- 可通过安全外壳(SSH)访问的IR1101

已配置IP地址对拥有15个用户权限的设备的访问
- Linux主机(本文使用最小的Debian 9(stretch)安装)
- IOx客户端安装文件，可从以下网址下载：https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762

本文档中的信息都是基于特定实验室环境中的设备编写的。本文档中使用的所有设备最初均采用原始（默认）配置。如果您使用的是真实网络，请确保您已经了解所有命令的潜在影响。

背景信息

与大多数其他IOx平台相比，IR1101稍有不同，因为这些平台主要基于x86。IR1101基于ARM64v8架构，因此您不能在平台上直接部署为x86构建的容器或IOx包。本文档从头开始，为构建基于ARM64v8的Docker容器准备环境，并说明如何使用x86 PC在IR1101上构建、打包和部署这些容器。

例如，使用非常小的Python脚本，即简单的Web服务器，并围绕Docker容器构建，最终将其打包以在IR1101上运行。Web服务器唯一要做的就是侦听预定义的端口(9000)，并在收到**GET请**求时返回一个简单页。这允许您测试运行自己代码的功能，并允许在IOx应用开始运行后测试对网络的访问。

该软件包将由Docker工具构建，并使用Alpine Linux。Alpine Linux是一个小型Linux映像（约5MB），通常用作Docker容器的基础。

由于周围的大多数台式机/笔记本电脑/VM都基于x86，因此您需要在构建容器的基于x86的机器上模拟ARM64v8架构。使用快速仿真器(QEMU)用户仿真可以轻松实现此目的。这允许在非本地架构中执行可执行文件，就像在其本地架构上运行一样。

配置

第1部分：为IR1101构建IOx包

1.在Linux主机上安装并准备IOx客户端

您需要**ioxclient**，以便在Docker容器构建后将其包装为IOx包，因此，我们先准备一下。

首先复制或下载ioxclient软件包。可从以下网址获取：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
```

提取包：

```
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
```

将路径添加到**PATH**变量，以使其可用，而无需使用完整位置。如果重新启动计算机或交换机用户，请不要忘记重复此步骤：

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

首次启动**ioxclient**以创建必需的配置文件。由于您仅使用ioxclient打包Docker容器，因此值可保留为默认值：

```
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.在Linux构建计算机上安装和准备Docker环境

此Docker用于从Alpine基映像构建容器并包含用例所需的文件。以下步骤基于Docker社区版(CE)针对Debian的正式安装指南：https://docs.docker.com/install/linux/docker-ce/debian/

更新计算机上的包列表：

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

安装依赖项以使用Docker回购：

```
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) ...
```

将Docker GNU隐私保护(GPG)密钥添加为有效的GPG密钥：

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

验证已安装的GPG密钥的指纹：

```
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]
```

添加Docker稳定回购：

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

在添加Docker回购时，再次更新包列表：

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

安装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) ...
```

要以普通用户身份访问/运行Docker，请将此用户添加到Docker组并刷新组成员身份：

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

3.安装QEMU用户仿真包

安装Docker后，需要安装QEMU用户仿真程序。从Docker容器内使用静态链接的QEMU仿真程序，这样您就可以在基于x86的Linux计算机上运行ARM64v8的容器，尽管目标容器将针对ARM64v8架构设计。

安装软件包：

```
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) ...
```

安装后，以下是静态链接的QEMU仿真器，可在/usr/**bin中使用**:

```
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
...
```

列表中的第一个是您需要的：aarch64是ARM64v8 for Linux的拱名。

4.测试aarch64/ARV64v8容器是否在x86 Linux计算机上运行

现在，您已安装了Docker和必要的QEMU二进制文件，您可以测试是否能够在x86计算机上运行为ARM64v8构建的Docker容器：

```
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
```

如输出所示，获取了arm64v8 Alpine容器，并使其能够通过访问仿真程序运行。

如果您请求容器的体系结构，您可以看到代码是针对aarch64编译的。与容器的目标体系结构应该针对IR1101完全一样。

5.准备文件以构建Docker Webserver容器

完成所有准备后，您可以继续为需要在IR1101上运行的Web服务器容器创建必要的文件。

第一个文件**是webserver.py**，即要在容器中运行的Python脚本。因为这只是一个示例，显然，您将用实际代码替换此代码，以便在IOx应用中运行：

```
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()
```

此代码包含用于写入日志文件的逻辑，该日志文件可供本地管理器咨询。

需要的第二个文件是Dockerfile。这定义了容器的构建方式：

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

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

Dockerfile定义如何构建容器。从ARM64v8的Apline基映像开始，在容器中复制仿真程序，运行apk以添加Python包并将Web服务器脚本复制到容器中。

在构建容器之前，最后需要做的准备是将qemu-aarch64-static复制到要构建容器的目录：

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

6.构建Docker容器

完成所有准备后，您可以使用Dockerfile构建容器：

```
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
```

作为测试，请运行刚构建的容器并检查脚本是否有效：

```
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
```

如您在此输出中所见，容器的体系结构是目标arch64。启动脚本后，您会看到它正在侦听端口9000上的请求。

7.构建IOx产品包

容器已准备好包装。在要求ioxclient执行此操作之前，首先需要创建包描述符：**package.yaml**。

此文件描述软件包的外观、需要运行的资源数量和启动内容。

```
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"]
```

如您所见，CPU体系结构设置为arch64。要获得对TCP端口9000的访问权，请使用**rootfs.tar**作为rootfs，然后开始运行**python/webserver.py**。

在打包之前，最后一件事是从Docker容**器中提**取rootfs.tar:

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

此时，您可以使用**ioxclient**为IR1101构建IOx包：

```
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
```

现在，有一个软件包可以部署在IR1101上，作为package.tar。下一部分介绍如何为部署准备设备。

第2部分：为IOx配置IR1101

1.启用Web接口、IOx和本地管理器

本地管理器是一个GUI，用于部署、激活、启动、管理IOx应用并排除其故障。对于IR1101，它嵌入到常规管理Web界面中。因此，您需要先启用该功能。

在IR1101上执行这些步骤以启用IOx和Web界面。

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

最后一行添加具有15权限的用户。此用户将有权访问Web界面和IOx本地管理器。

2.配置IOx网络

在您访问Web界面之前，我们先添加IOx网络所需的配置。背景信息可在IOx的IR1101文档中找到：https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b_IR1101config/b_IR1101config_chapter_010001.html

简而言之，IOx应用可以使用VirtualPortGroup0接口（与IR809上的Gi2和IR829接口上的Gi5相当）与外界通信。

```
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
```

将VirtualPortGroup0接口配置为内部网络地址转换(NAT)时，需要在Gi 0/0/0接口上添加ip nat outside语句，以便通过使用NAT来允许与IOx应用进行通信：

```
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
```

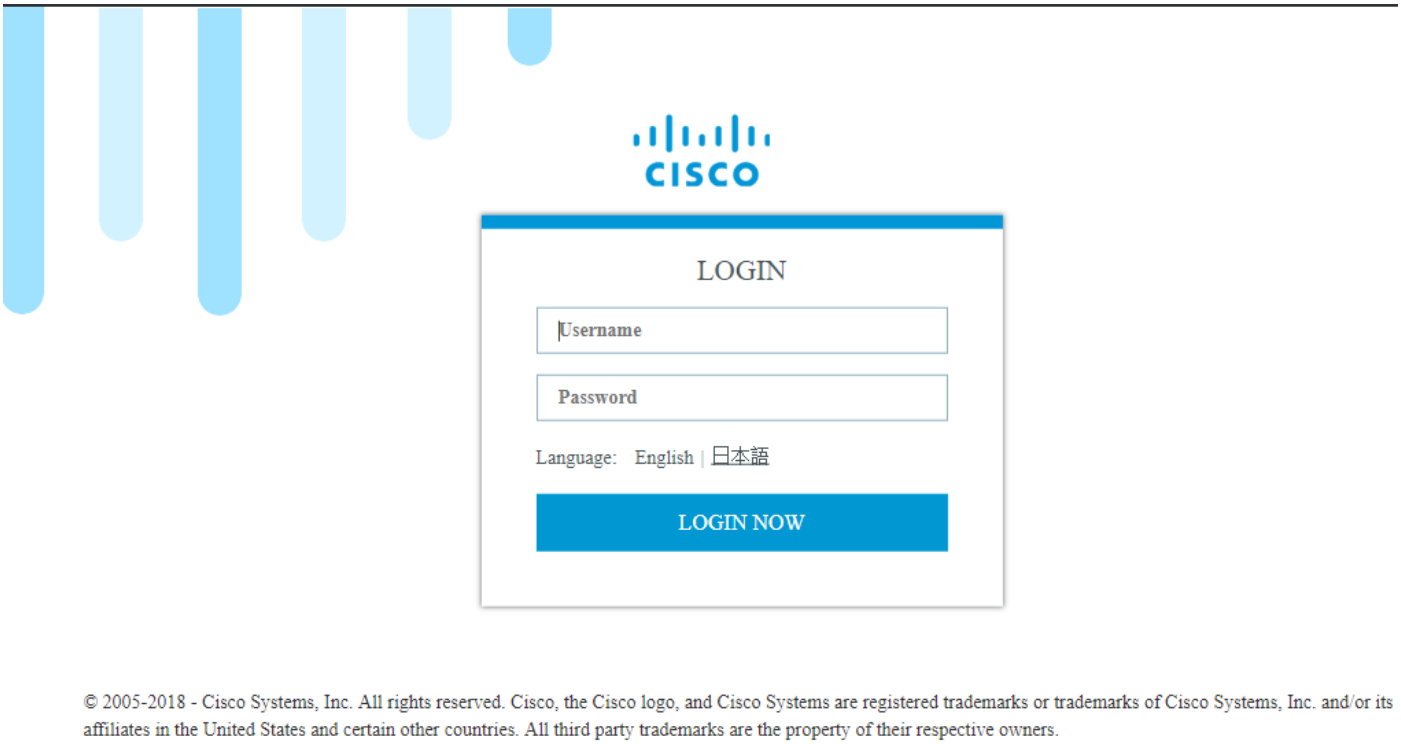
为了允许对容器的端口9000进行访问，您需要向前添加一个端口：

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

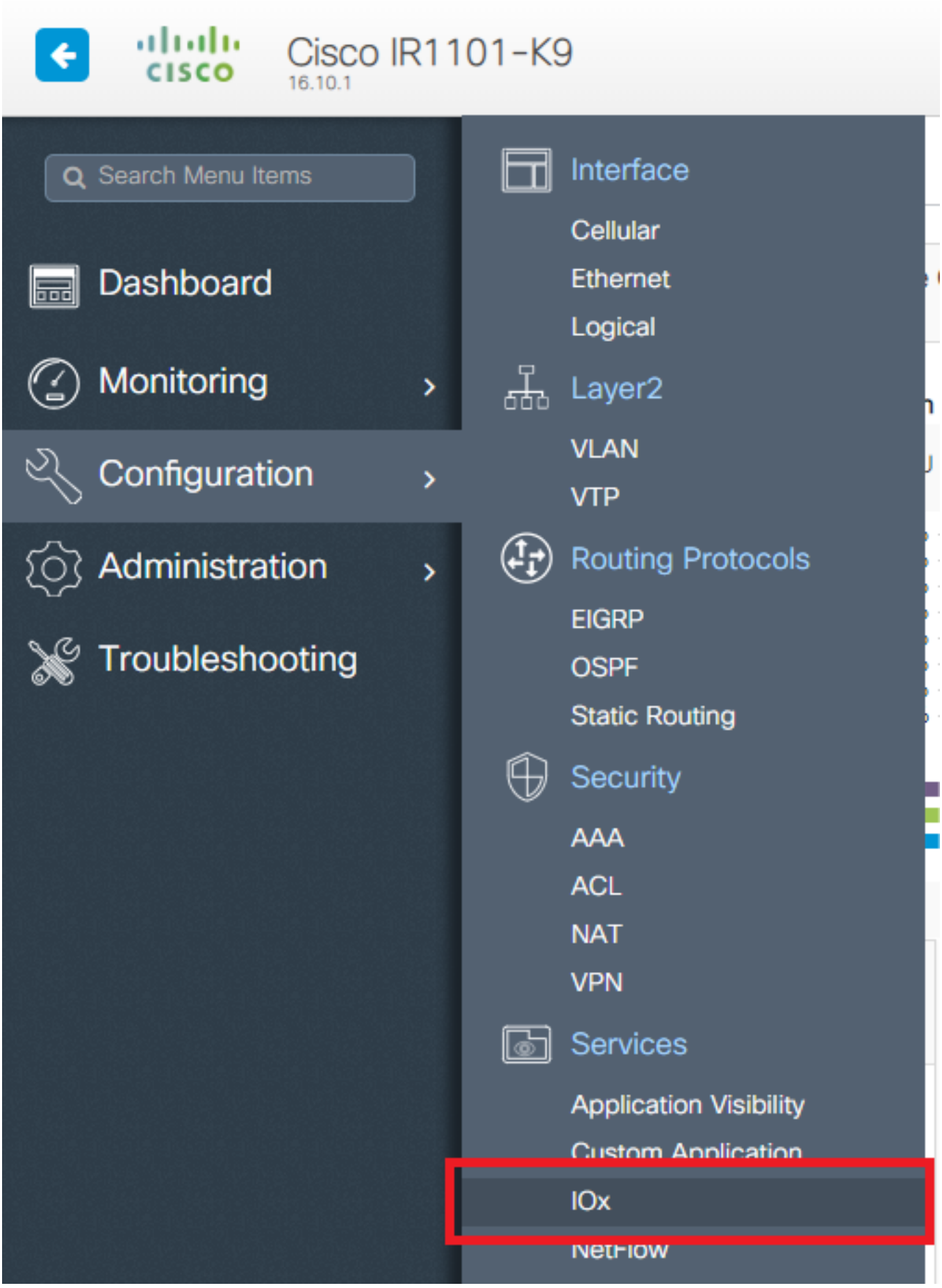
对于本指南，请对每个IOx应用使用静态配置的IP。如果要向应用动态分配IP地址，则需要在VirtualPortGroup0子网中添加DHCP服务器的配置。

第3部分：访问本地管理器并部署IOx应用

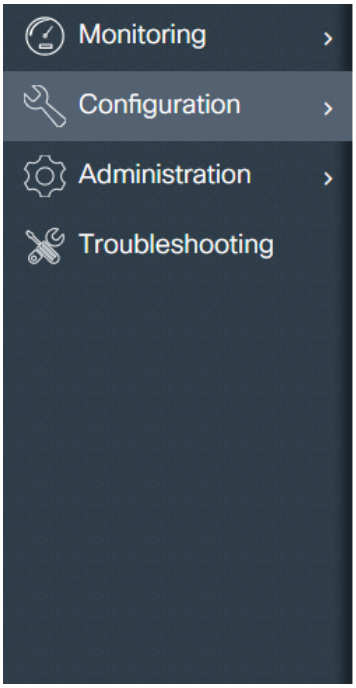
在将这些行添加到配置后，您可以使用Web界面访问IR1101。使用浏览器导航至Gi 0/0/0 IP地址，如图所示。

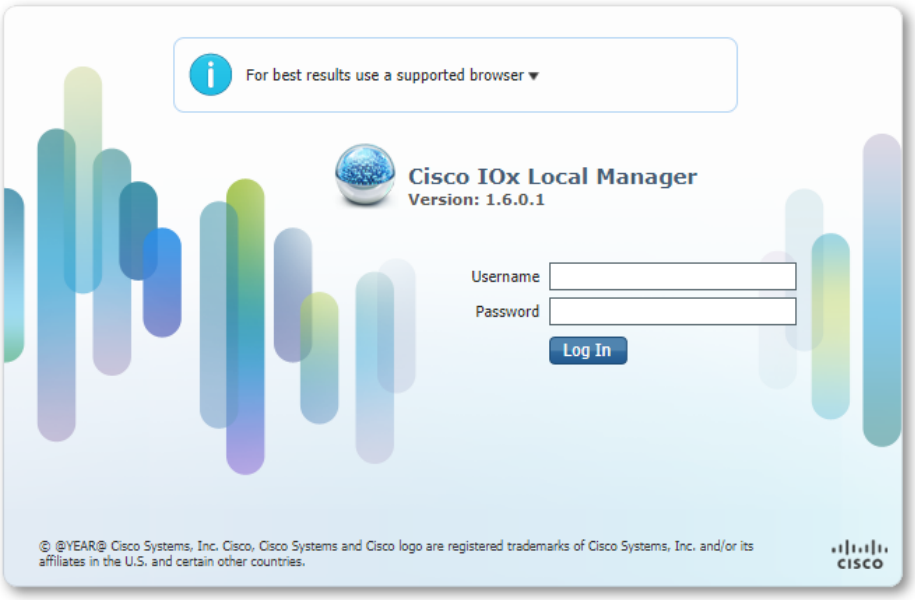


使用在步骤1中创建的权限15帐户登录Web界面并导航至**Configuration** - IOx，如图所示。

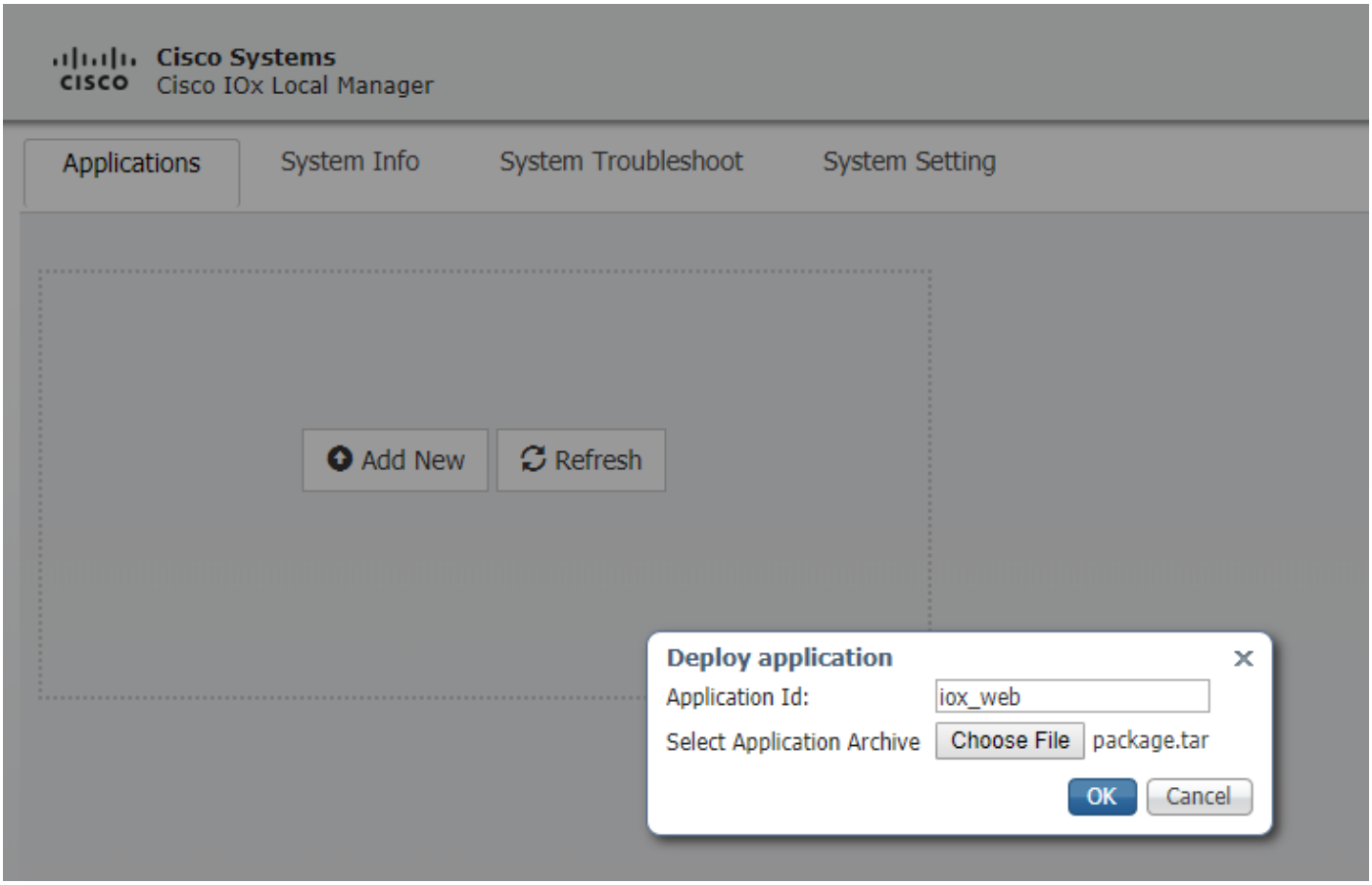


在IOx本地管理器登录中，使用与图中所示相同的帐户继续。

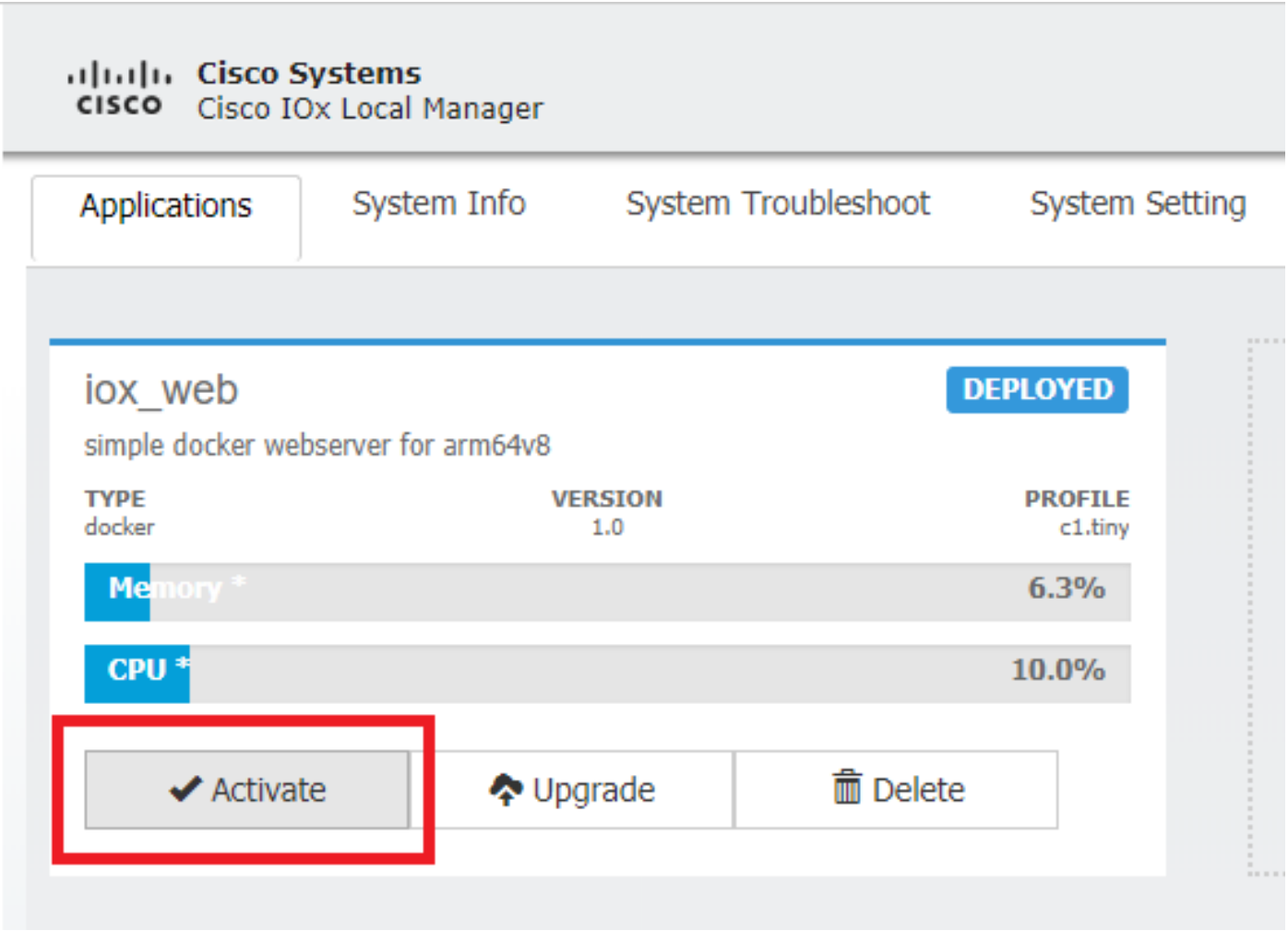




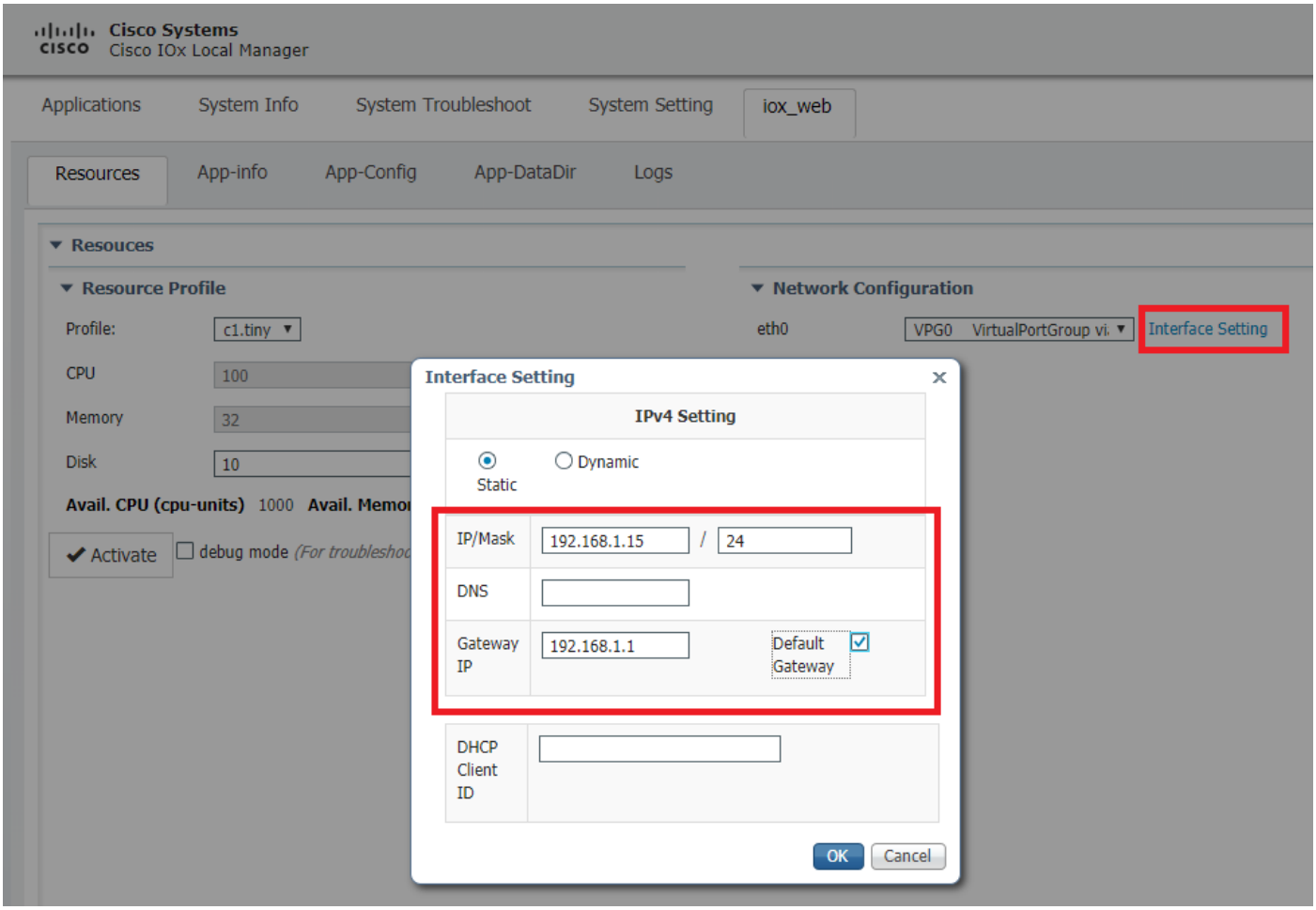
单击**Add New**，为IOx应用选择一个名称，然后选择在第1部分中构建的package.tar，如图所示。



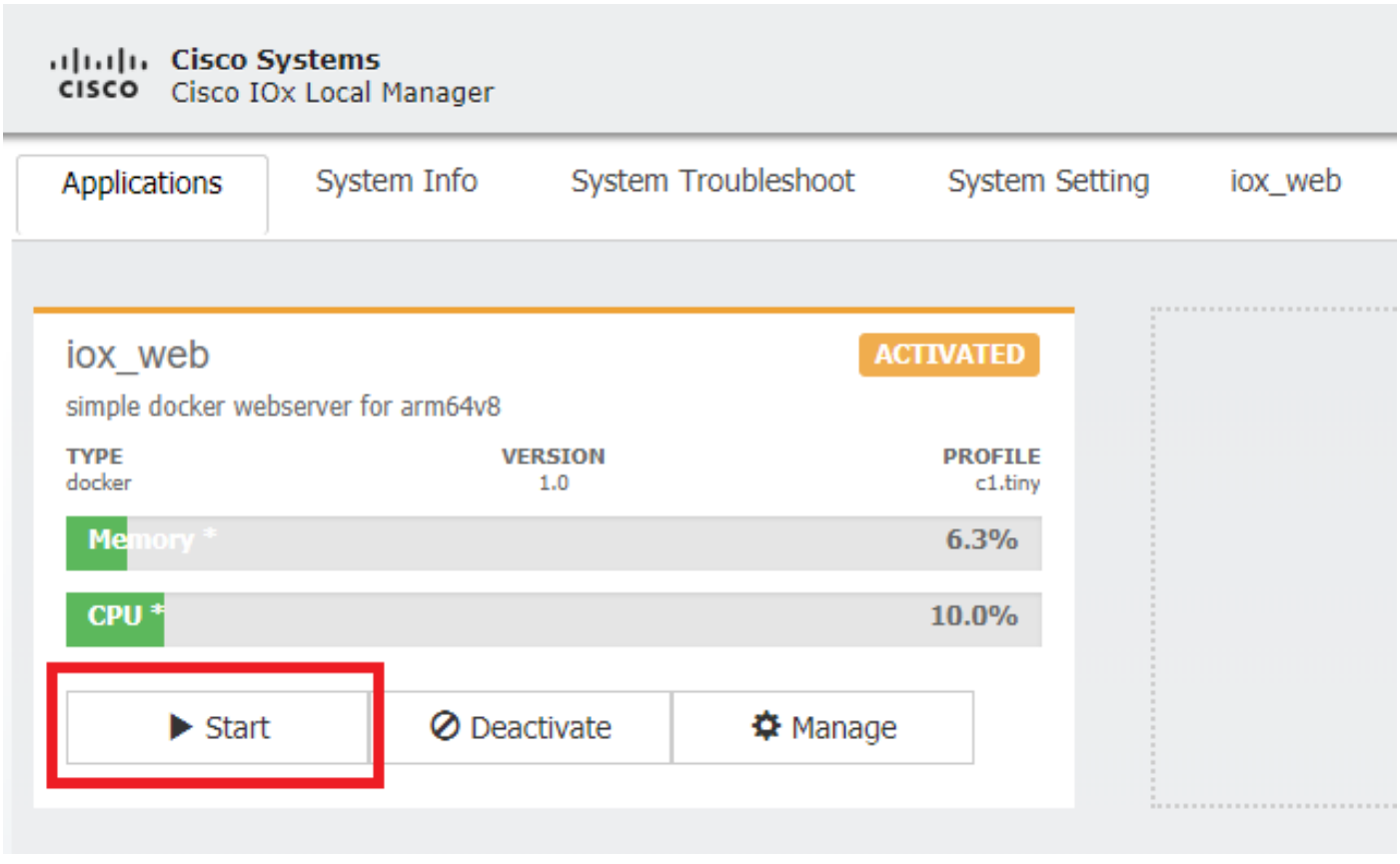
上传包后，您可以激活它，如图所示。



在**资源**选项卡中，打开接口设置，以指定要分配给应用的固定IP，如图所示。



单击**确定**，然后单击**激活**。操作完成后，导航回Local Manager主页(顶部菜单上的**Applications**按钮)，然后启动应用程序，如图所示。



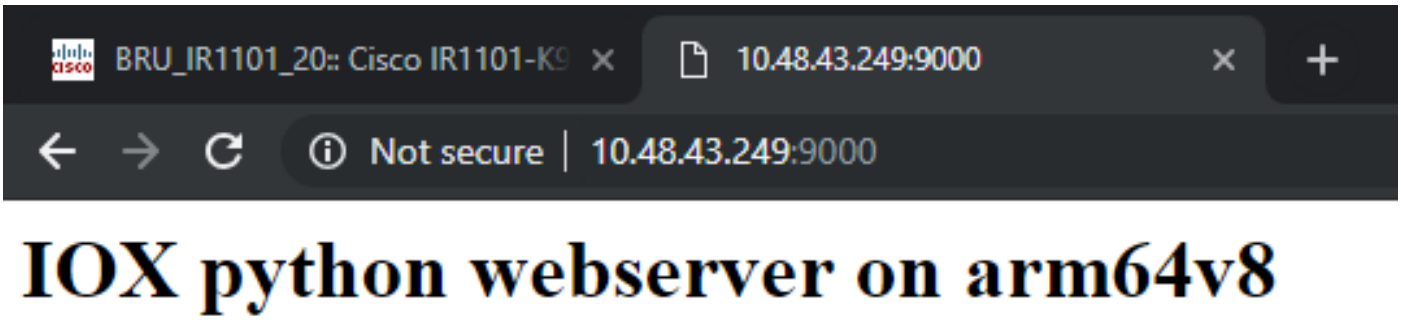
完成这些步骤后，应用程序应运行并通过端口9000使用IR1101的Gi 0/0/0接口可用。

验证

使用本部分可确认配置能否正常运行。

为了进行验证，您可以使用端口9000访问IR1101上Gi 0/0/0接口的IP地址。

如果一切顺利，您应该看到如下，因为它是在Python脚本中创建的。



故障排除

本部分提供了可用于对配置进行故障排除的信息。

为了进行故障排除，您可以使用本地管理器检查在Python脚本中创建的日志文件。

导航至**Applications**，在**iox_web**应用程序上单击**Manage**，然后选择**Logs**选项卡，如图所示。