VagrantとVirtualbox/VMWareを使用したIOxアプリケーションの構築

内容

はじめに

このドキュメントでは、VagrantとVirtualboxを使用してIOxアプリケーションを構築し、IOxローカルマネージャGUIに展開する方法について説明します。

前提条件

Windows/MAC Intel/Linux

- Git
- 迷走神経
- 仮想ボックス

MAC ARMベース – M1/M2/M3

- Git
- 迷走神経
- VMWare統合
- vagrant-vmware-desktopプラグイン

ダウンロードするには:

- [迷走神経](#)
- [仮想ボックス](#)

Vagrantを使用してビルド環境を設定する手順

アクションの概要

- vagrantfile設定は、ホストマシンアーキテクチャに基づいてVM環境をセットアップします。
- アーキテクチャに応じて、VMware FusionまたはVirtualBoxを使用するようにVMを設定します
- VMに、QEMU(Quick EMUlator)、Docker、ioxclientなどの必要なソフトウェアとツールをプロビジョニングします。
- 構成により、amd64ターゲットのシスコプラットフォームデバイス用のサンプルiperfアプリケーションが自動的に構築されます。

ステップ 1：ローカルシステムでGithubリポジトリを複製します。

```
git clone https://github.com/suryasundarraj/cisco-iox-app-build.git
```

または、設定エンクロージャの内容をコピーして「Vagrantfile」に貼り付けます。これにより、ローカルシステムに「Vagrantfile」という名前のファイルが作成されます。

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure('2') do |config|
  arch = `arch`.strip()
  if arch == 'arm64'
    puts "This appears to be an ARM64 machine! ..."
    config.vm.box = 'gyptazy/ubuntu22.04-arm64'
    config.vm.boot_timeout = 600
    config.vm.provider "vmware_fusion" do |vf|
      #vf.gui = true
      vf.memory = "8192"
      vf.cpus = "4"
    end
    config.vm.define :ioxappbuild
  else
    puts "Assuming this to be an Intel x86 machine! ..."
    config.vm.box = "bento/ubuntu-22.04"
    config.vm.network "public_network", bridge: "ens192"
    config.vm.boot_timeout = 600
    config.vm.provider "virtualbox" do |vb|
      #vb.gui = true
      vb.memory = "8192"
      vb.cpus = "4"
    end
    config.vm.define :ioxappbuild
  end

  config.vm.provision "shell", inline: <<-SHELL
```

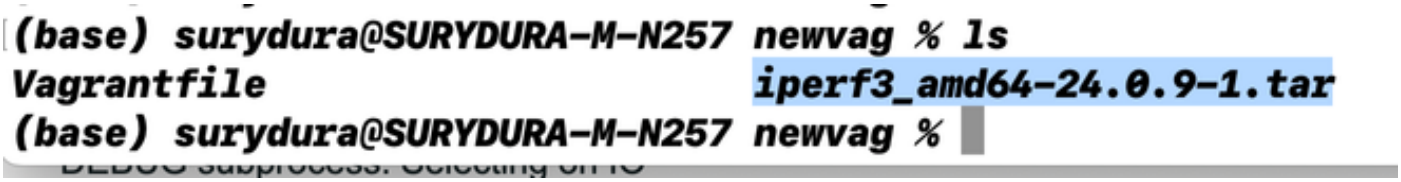
```
    #!/bin/bash
    # apt-cache madison docker-ce
    export VER="5:24.0.9-1~ubuntu.22.04~jammy"
    echo "!!! installing dependencies and packages !!!"
    apt-get update
    apt-get install -y ca-certificates curl unzip git pcregrep
    install -m 0755 -d /etc/apt/keyrings
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
    chmod a+r /etc/apt/keyrings/docker.asc
    echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://downlo
    apt-get update
    apt-get install -y qemu binfmt-support qemu-user-static
    apt-get install -y docker-ce=$VER docker-ce-cli=$VER docker-ce-rootless-extras=$VER containerd.io d
    # apt-get install -y docker.io docker-compose docker-buildx
    usermod -aG docker vagrant
    echo "!!! generating .ioxclientcfg.yaml file !!!"
    echo 'global:' > /home/vagrant/.ioxclientcfg.yaml
    echo '  version: "1.0"' >> /home/vagrant/.ioxclientcfg.yaml
    echo '  active: default' >> /home/vagrant/.ioxclientcfg.yaml
    echo '  debug: false' >> /home/vagrant/.ioxclientcfg.yaml
    echo '  fogportalprofile:' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    fogpip: ""' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    fogpport: ""' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    fogpapiprefix: ""' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    fogpurlscheme: ""' >> /home/vagrant/.ioxclientcfg.yaml
    echo '  dockerconfig:' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    server_uri: unix:///var/run/docker.sock' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    api_version: "1.22"' >> /home/vagrant/.ioxclientcfg.yaml
    echo 'author:' >> /home/vagrant/.ioxclientcfg.yaml
    echo '  name: |' >> /home/vagrant/.ioxclientcfg.yaml
    echo '    Home' >> /home/vagrant/.ioxclientcfg.yaml
    echo '  link: localhost' >> /home/vagrant/.ioxclientcfg.yaml
    echo 'profiles: {default: {host_ip: 127.0.0.1, host_port: 8443, auth_keys: cm9vdDpyb290,' >> /home/
    echo '    auth_token: "", local_repo: /software/downloads, api_prefix: /iox/api/v2/hosting/,' >> /h
    echo '    url_scheme: https, ssh_port: 2222, rsa_key: "", certificate: "", cpu_architecture: "",' >
    echo '    middleware: {mw_ip: "", mw_port: "", mw_baseuri: "", mw_urlscheme: "", mw_access_token: "
    echo '    conn_timeout: 1000, client_auth: "no", client_cert: "", client_key: ""}}' >> /home/vagran
    cp /home/vagrant/.ioxclientcfg.yaml /root/.ioxclientcfg.yaml
    chown vagrant:vagrant /home/vagrant/.ioxclientcfg.yaml
    arch=$(uname -m)
    if [[ $arch == x86_64 ]]; then
      # download page https://developer.cisco.com/docs/iox/iox-resource-downloads/
      echo "!!! downloading and extracting ioxclient for x86_64 architecture !!!"
      curl -O https://pubhub.devnetcloud.com/media/iox/docs/artifacts/ioxclient/ioxclient-v1.17.0.0/iox
      tar -xvf /home/vagrant/ioxclient_1.17.0.0_linux_amd64.tar.gz
      cp /home/vagrant/ioxclient_1.17.0.0_linux_amd64/ioxclient /usr/local/bin/ioxclient
      rm -rv /home/vagrant/ioxclient_1.17.0.0_linux_amd64
    elif  [[ $arch = aarch64 ]]; then
      # download page https://developer.cisco.com/docs/iox/iox-resource-downloads/
      echo "!!! downloading and extracting ioxclient for arm64 architecture !!!"
      curl -O https://pubhub.devnetcloud.com/media/iox/docs/artifacts/ioxclient/ioxclient-v1.17.0.0/iox
      tar -xvf /home/vagrant/ioxclient_1.17.0.0_linux_arm64.tar.gz
      cp /home/vagrant/ioxclient_1.17.0.0_linux_arm64/ioxclient /usr/local/bin/ioxclient
      rm -rv /home/vagrant/ioxclient_1.17.0.0_linux_arm64
    fi
    chown vagrant:vagrant /usr/local/bin/ioxclient
    echo "!!! pulling and packaging the app for x86_64 architecture !!!"
    docker pull --platform=linux/amd64 mlabbe/iperf3
    ioxclient docker package mlabbe/iperf3 .
    cp package.tar /vagrant/iperf3_amd64-$(echo $VER | pcregrep -o1 ':([0-9.-]+)~').tar
  SHELL
end
```

ステップ 2：「export VER="5:24.0.9-1~ubuntu.22.04~jammy"」行がコメント解除され、他のすべてのエクスポート文がコメント化されていることを確認します。これは、このVagrant環境にインストールするDocker Engineバージョンに対応します。

```
cisco@cisco-virtual-machine:~/Desktop/ioxappbuild$ cat Vagrantfile | grep 'export' | grep -v '#'
export VER="5:24.0.9-1~ubuntu.22.04~jammy"
```

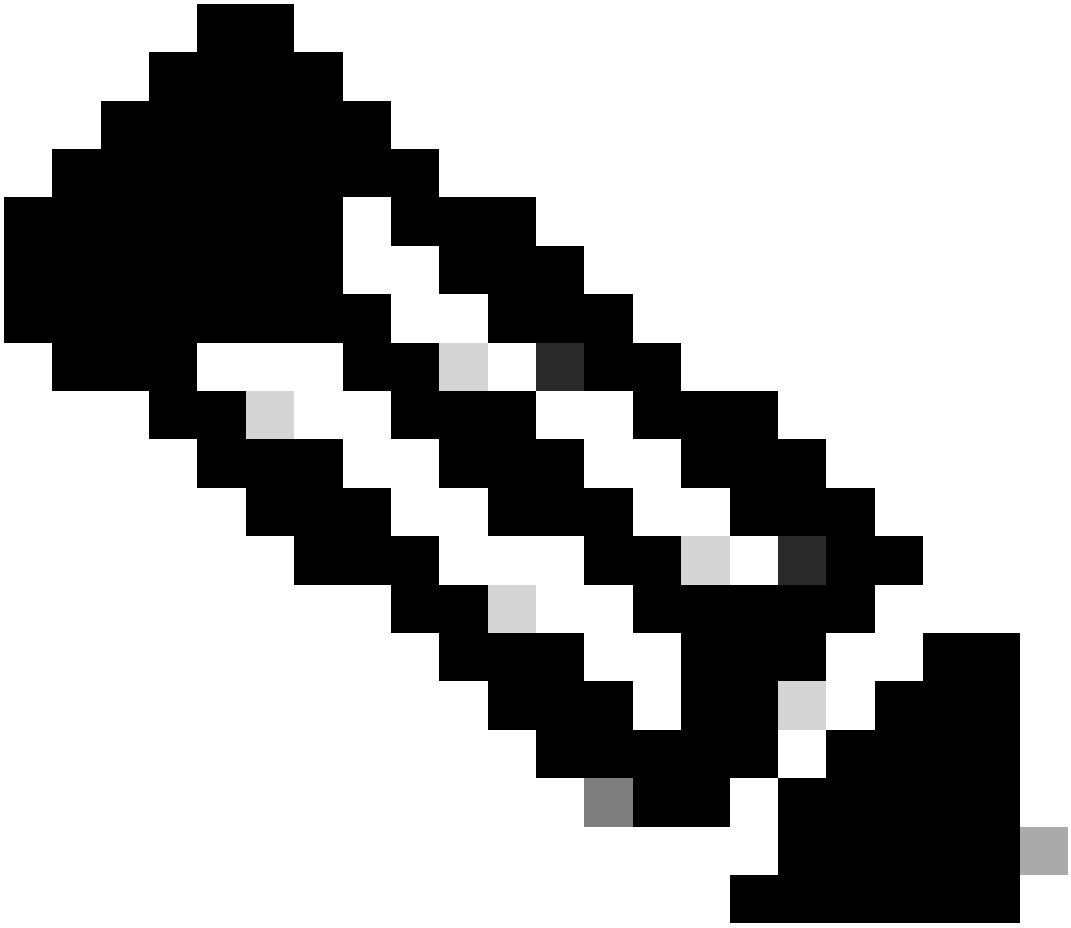
ステップ 3：Vagrantファイルが存在するディレクトリでvagrant upコマンドを使用してVagrant環境を起動し、amd64 tarファイル用のiperf IOxアプリケーションが正常に構築されたことを確認します。

```
vagrant up
```



カスタムIOxアプリケーションを構築する手順

このセクションでは、vagrant環境を使用したカスタムIOxアプリケーションの構築方法について説明します。



注:VM内のディレクトリ「/vagrant」とホストシステム内の「Vagrantfile」を含むディレクトリは同期しています。

図に示すように、new.jsファイルはVM内に作成され、ホストシステムでもアクセス可能です。

```
vagrant@vagrant:/vagrant$ pwd
/vagrant
vagrant@vagrant:/vagrant$ touch new.js
vagrant@vagrant:/vagrant$ ls
Vagrantfile  dockerapp  iperf3_amd64-24.0.9-1.tar  new.js
vagrant@vagrant:/vagrant$
vagrant@vagrant:/vagrant$
vagrant@vagrant:/vagrant$
vagrant@vagrant:/vagrant$ exit
logout
(base) surydura@SURYDURA-M-N257 newvag %
(base) surydura@SURYDURA-M-N257 newvag %
(base) surydura@SURYDURA-M-N257 newvag % ls
Vagrantfile                 dockerapp                   iperf3_amd64-24.0.9-1.tar       new.js
(base) surydura@SURYDURA-M-N257 newvag %
```

ステップ 1：サンプルアプリケーションを、「Vagrantfile」が存在するフォルダと同じフォルダ

にクローニングします。この例では「iox-multiarch-nginx-nyancat-sample」アプリケーションを使用しています。

```
git clone https://github.com/etychon/iox-multiarch-nginx-nyancat-sample.git
```

ステップ 2：移動型マシンにSSHで接続します。

```
vagrant ssh
```

```
(base) surydura@SURYDURA-M-N257 newvag % vagrant ssh
This appears to be an ARM64 machine! ...
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-87-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Aug  5 03:21:53 PM UTC 2024

  System load:  0.23388671875      Processes:               259
  Usage of /:   37.4% of 18.01GB   Users logged in:         0
  Memory usage: 3%                 IPv4 address for ens160: 192.168.78.129
  Swap usage:   0%


Expanded Security Maintenance for Applications is not enabled.

171 updates can be applied immediately.
106 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Fri Oct 20 16:12:20 2023 from 192.168.139.1
vagrant@vagrant:~$
```

ステップ 3：アプリケーションを構築します。

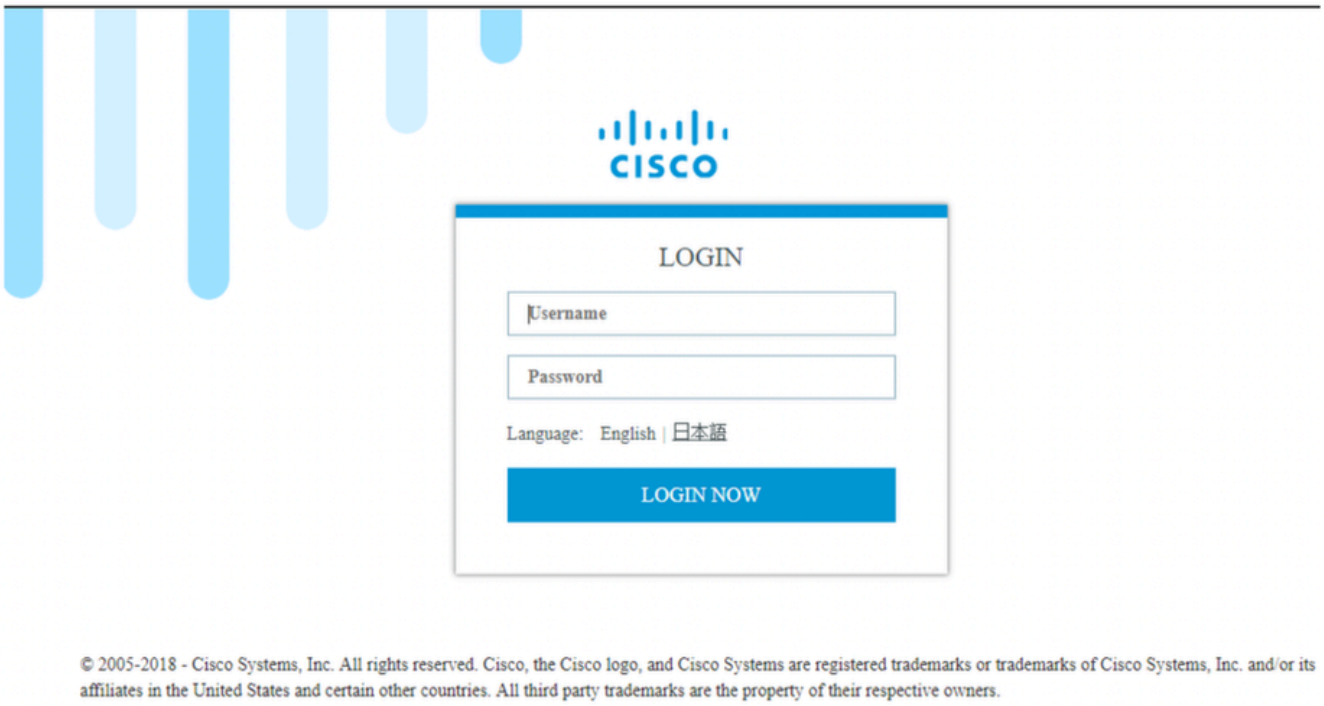
```
cd /vagrant/iox-multiarch-nginx-nyancat-sample/
chmod +x build
sh ./build
```

構築プロセスが完了すると、2つのIOxアプリケーションを導入可能な状態になります(amd64の場合「iox-amd64-nginx-nyancat-sample.tar.gz」、ターゲットプラットフォームの場合「iox-arm64-nginx-nyancat-sample.tar.gz」)。

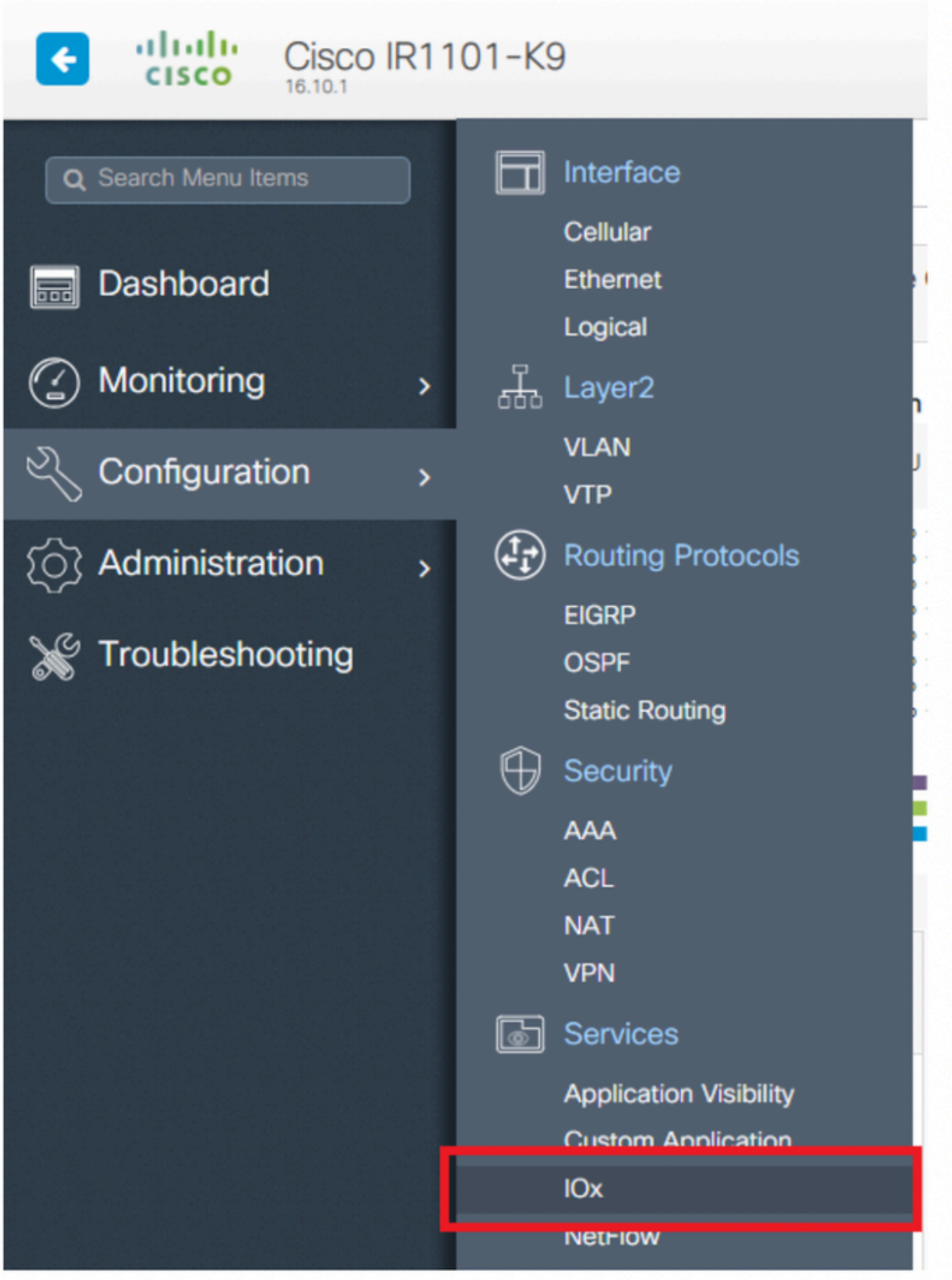
```
Package docker image iox-arm64-nginx-nyancat-sample at /vagrant/iox-multiarch-nginx-nyancat-sample/iox-arm64-nginx-nyancat-sample.tar.gz
vagrant@vagrant:/vagrant/iox-multiarch-nginx-nyancat-sample$ ls
Dockerfile  README.md  images                        iox-arm64-nginx-nyancat-sample.tar.gz  nyan-cat      package.yaml.amd64
LICENSE     build      iox-amd64-nginx-nyancat-sample.tar.gz  loop.sh                        package.yaml  package.yaml.arm64
vagrant@vagrant:/vagrant/iox-multiarch-nginx-nyancat-sample$ exit
logout
(base) surydura@SURYDURA-M-N257 newvag % cd iox-multiarch-nginx-nyancat-sample
(base) surydura@SURYDURA-M-N257 iox-multiarch-nginx-nyancat-sample % ls
Dockerfile                      images                          nyan-cat
LICENSE                         iox-amd64-nginx-nyancat-sample.tar.gz   package.yaml
README.md                       iox-arm64-nginx-nyancat-sample.tar.gz   package.yaml.amd64
build                           loop.sh                         package.yaml.arm64
(base) surydura@SURYDURA-M-N257 iox-multiarch-nginx-nyancat-sample %
```

IOxアプリケーションの導入

ステップ 1：Webインターフェイスを使用してIR1101にアクセスします。

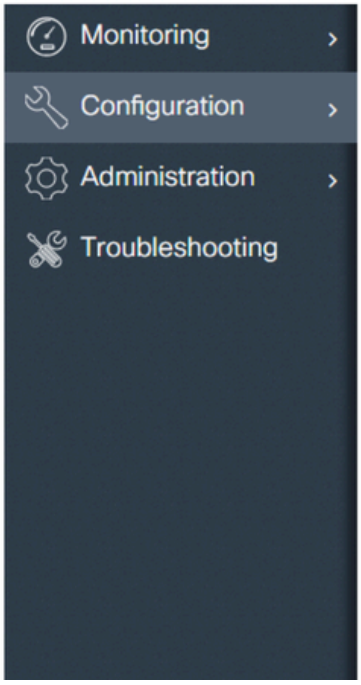


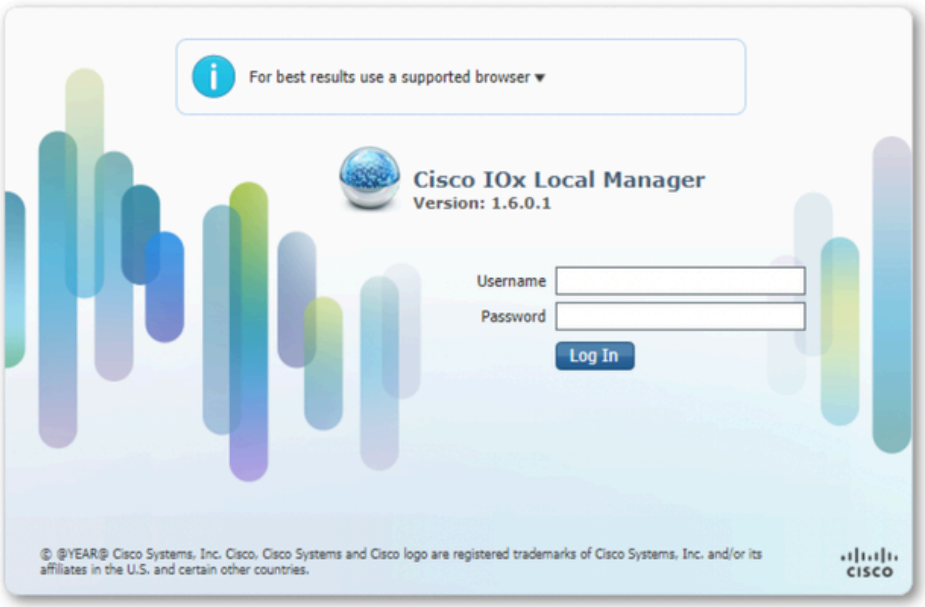
ステップ 2：特権15アカウントを使用します。



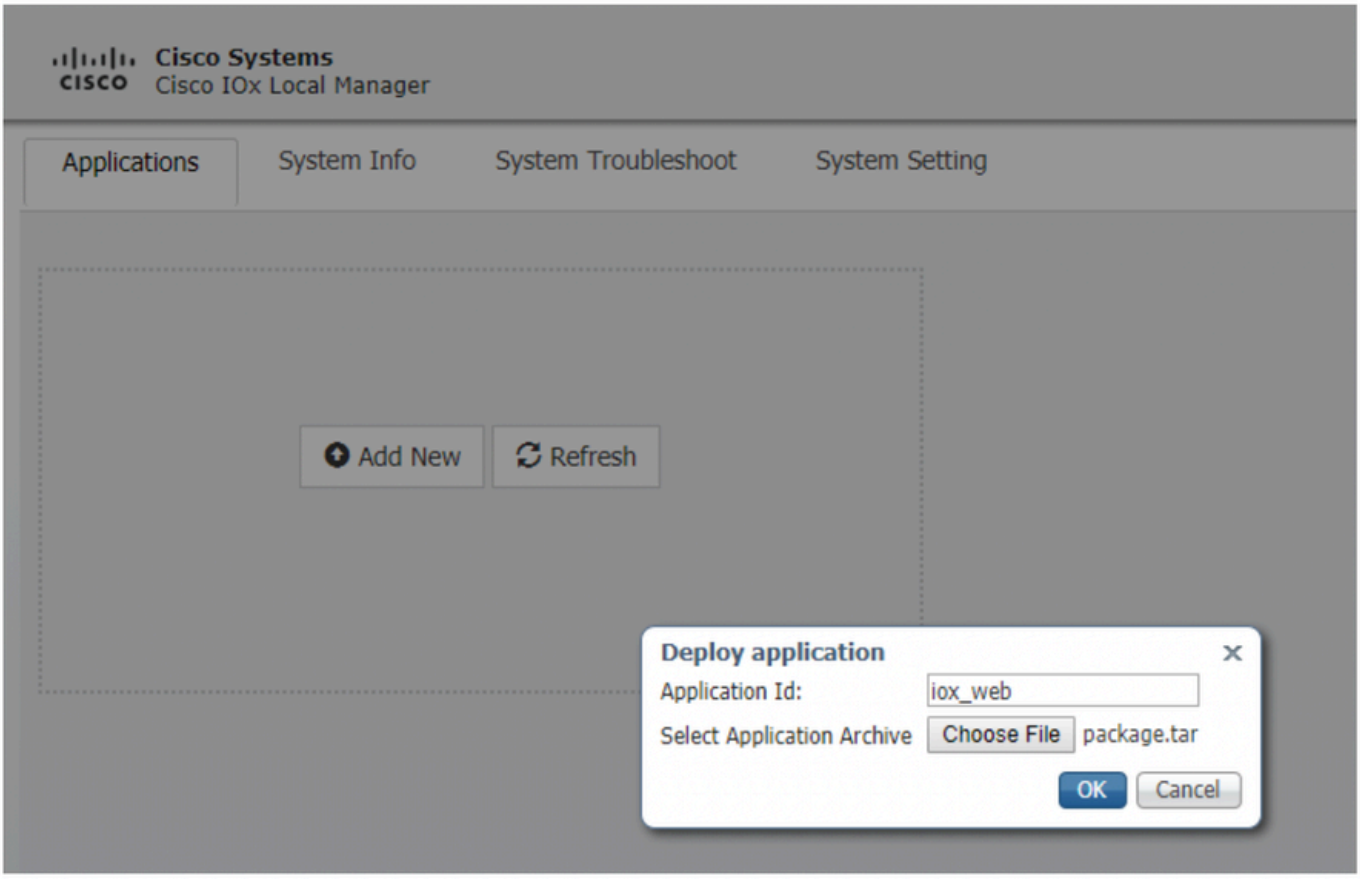
ステップ 3：図に示すように、IOx Local Managerのログインで、同じアカウントを使用して続行

します。

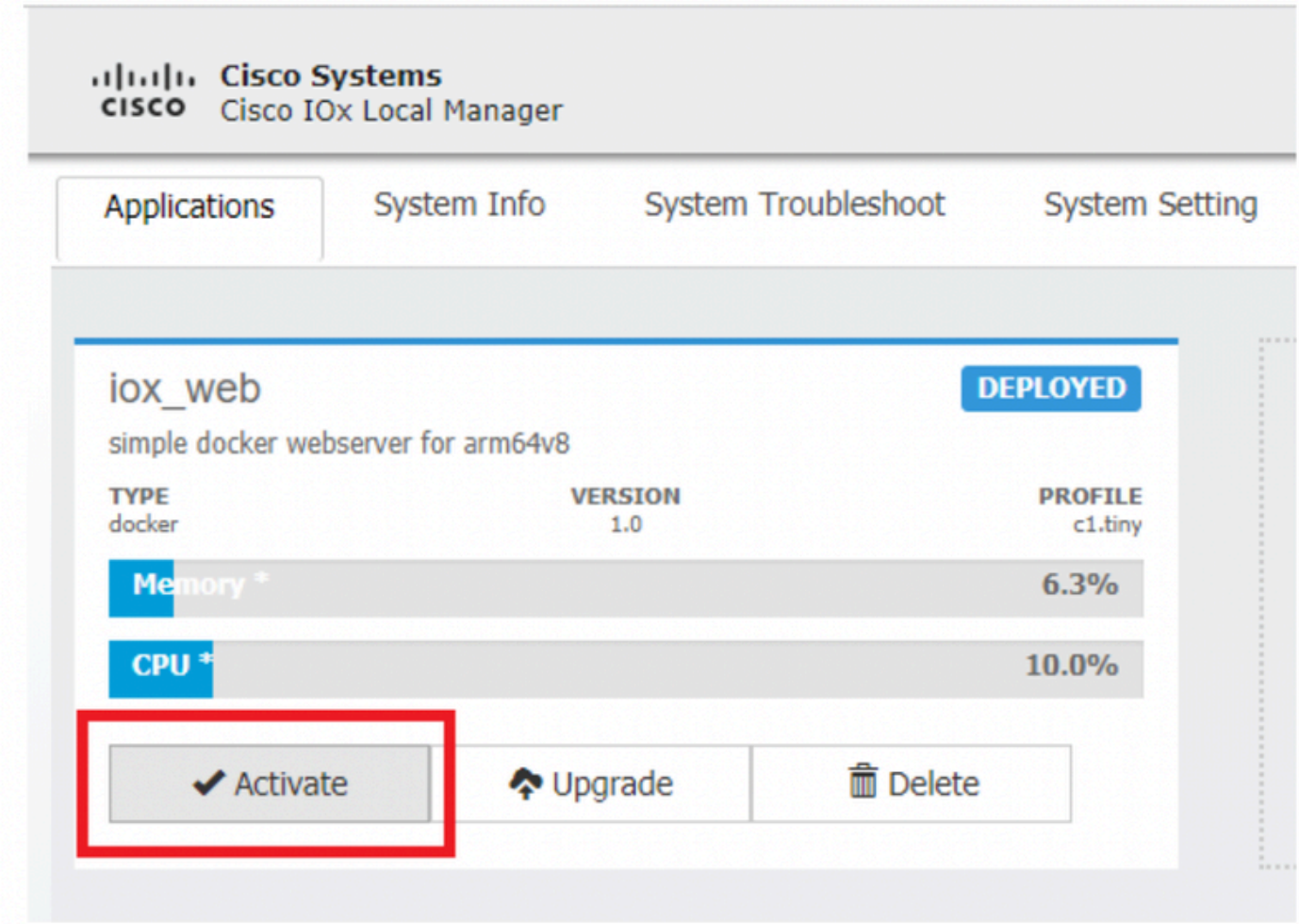




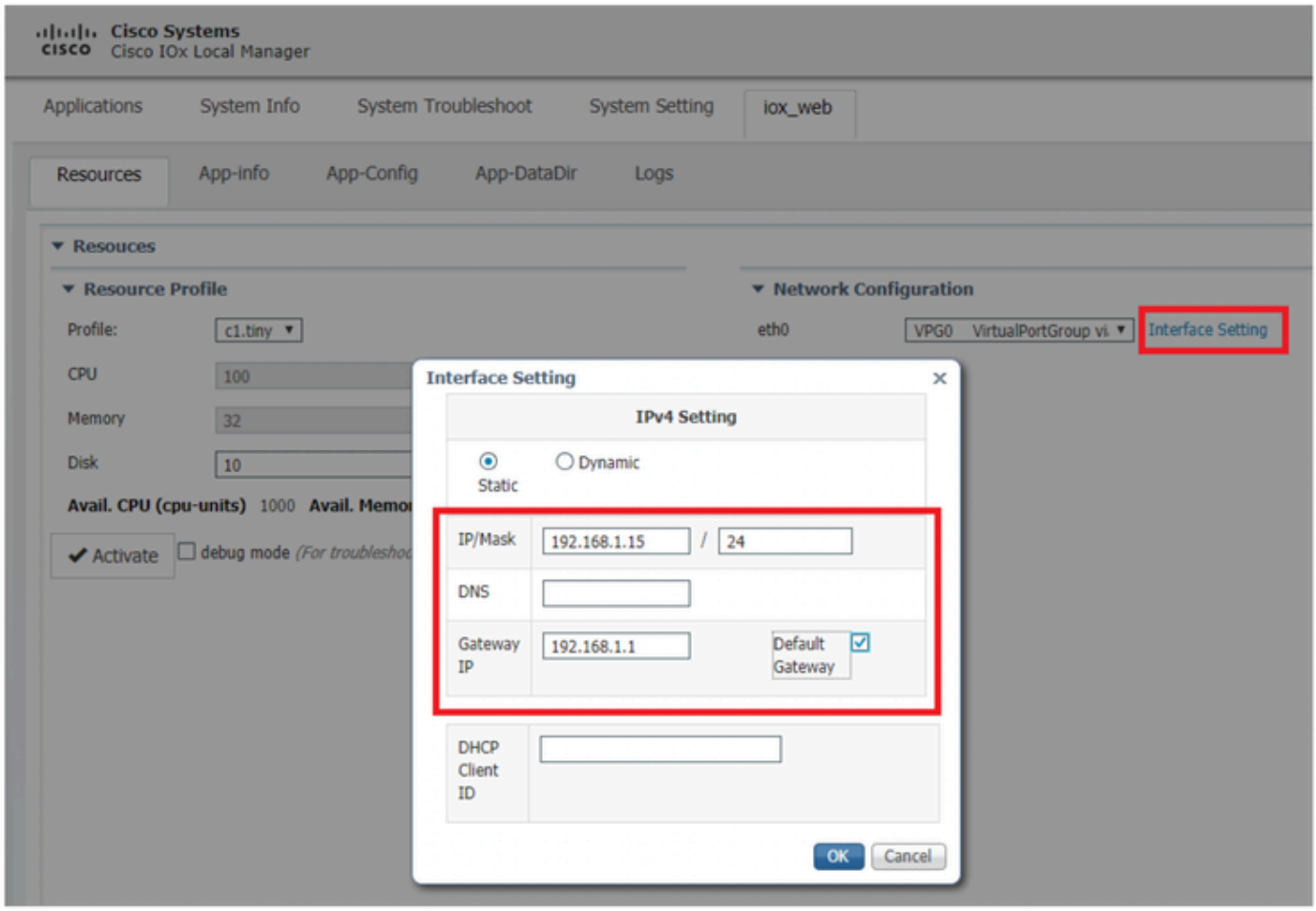
ステップ 4：Add Newをクリックし、IOxアプリケーションの名前を選択し、「Vagrantを使用してビルド環境をセットアップする手順」セクションのステップ3で構築したpackage.tarを選択します（次の図を参照）。



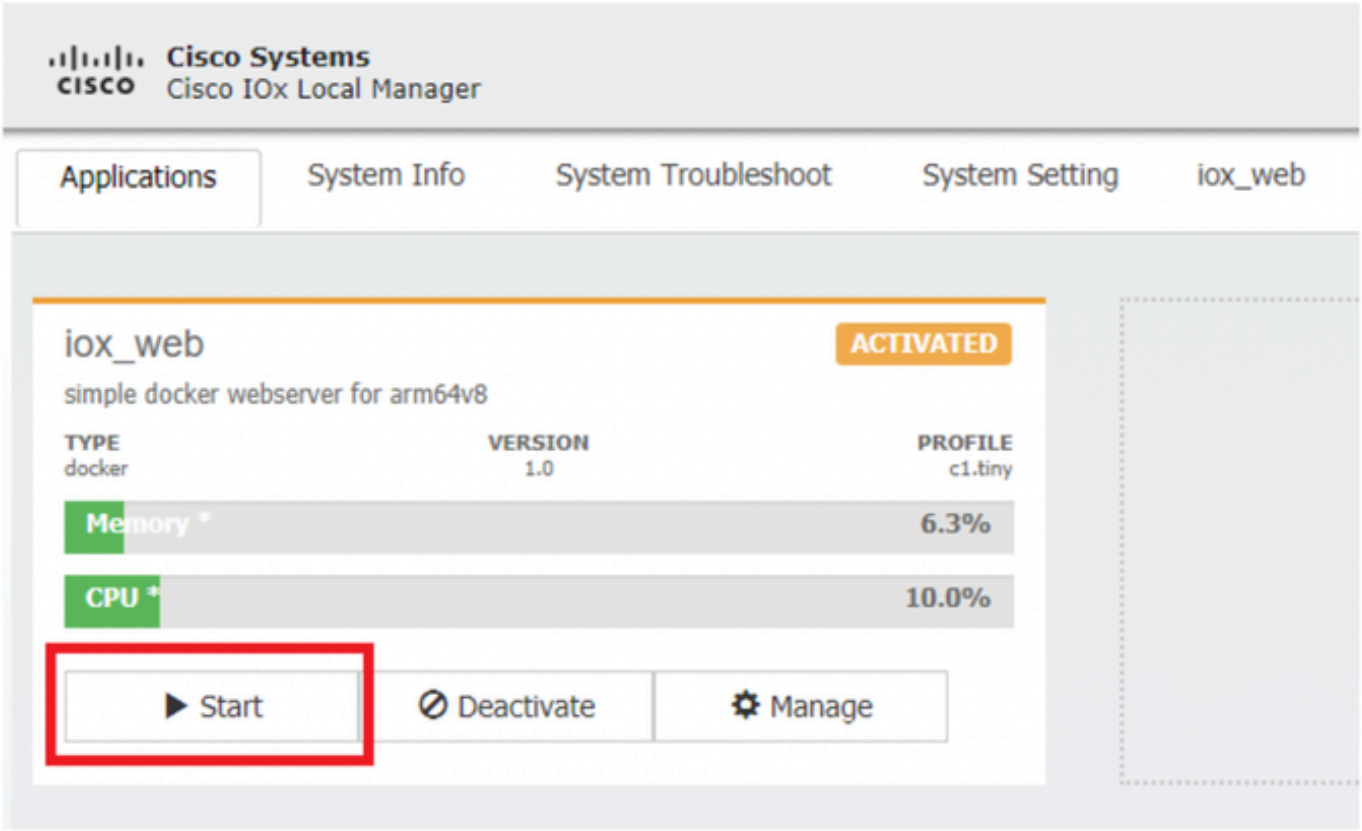
ステップ 5：パッケージがアップロードされたら、次の図に示すようにアクティブにします。



手順 6：図に示すように、Resourcesタブでインターフェイス設定を開き、アプリケーションに割り当てる固定IPを指定します。



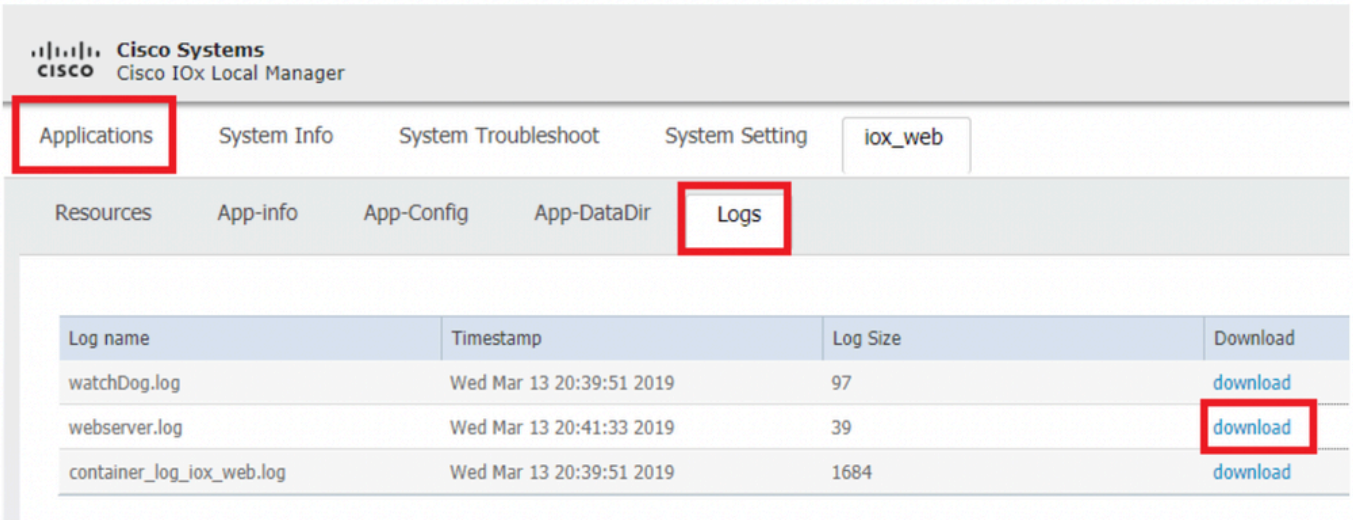
手順 7：OKをクリックしてから、Activateをクリックします。アクションが完了したら、メインのLocal Managerページ(トップメニューのApplicationsボタン)に戻り、図に示すようにアプリケーションを起動します。



これらの手順を実行すると、アプリケーションを実行する準備が整います。

トラブルシュート

設定のトラブルシューティングを行うには、Pythonスクリプトでローカルマネージャを使用して作成したログファイルを確認します。Applicationsに移動し、iox_webアプリケーションでManageをクリックしてから、図に示すようにLogsタブを選択します。



翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。