

Erstellen und Bereitstellen eines Docker-IOx-Pakets für die IR1101 ARM-Architektur

Inhalt

[Einführung](#)

[Voraussetzungen](#)

[Anforderungen](#)

[Verwendete Komponenten](#)

[Hintergrundinformationen](#)

[Konfigurieren](#)

[Teil 1: Erstellen des IOx-Pakets für IR1101](#)

[1. Installieren und Vorbereiten des IOx-Clients auf dem Linux-Host](#)

[2. Installieren und Vorbereiten der Docker-Umgebung auf dem Linux Build Machine](#)

[3. Installieren der QEMU-Benutzeremulationspakete](#)

[4. Testen Sie, ob ein Aarch64/ARV64v8-Container auf dem x86-Linux-System ausgeführt wird.](#)

[5. Vorbereiten von Dateien zum Erstellen des Docker-Webserver-Containers](#)

[6. Erstellen des Docker-Containers](#)

[7. Erstellen des IOx-Pakets](#)

[Teil 2. Konfigurieren des IR1101 für IOx](#)

[1. Aktivieren Sie die Webschnittstelle, IOx und den lokalen Manager.](#)

[2. Konfigurieren von IOx-Netzwerken](#)

[Teil 3. Zugriff auf den lokalen Manager und Bereitstellung der IOx-Anwendung](#)

[Überprüfen](#)

[Fehlerbehebung](#)

Einführung

In diesem Dokument wird beschrieben, wie ein docker-basiertes IOx-Paket für das ARM-basierte Internet of Things (IoT)-Gateway IR1101 vorbereitet, erstellt und bereitgestellt wird.

Voraussetzungen

Anforderungen

Cisco empfiehlt, über Kenntnisse in folgenden Bereichen zu verfügen:

- Linux
- Container
- IOx

Verwendete Komponenten

Die Informationen in diesem Dokument basieren auf den folgenden Software- und

Hardwareversionen:

- IR1101 ist über Secure Shell (SSH) erreichbar
IP-Adresse konfiguriertZugriff auf das Gerät mit einer Berechtigung für 15 Benutzer
- Linux-Host (eine minimale Debian 9 (Stretch)-Installation wird für diesen Artikel verwendet
- IOx-Client-Installationsdateien, die unter folgender Adresse heruntergeladen werden können:
<https://software.cisco.com/download/release.html?mdfid=286306005&softwareid=286306762>

Die Informationen in diesem Dokument wurden von den Geräten in einer bestimmten Laborumgebung erstellt. Alle in diesem Dokument verwendeten Geräte haben mit einer leeren (Standard-)Konfiguration begonnen. Wenn Ihr Netzwerk in Betrieb ist, stellen Sie sicher, dass Sie die potenziellen Auswirkungen eines Befehls verstehen.

Hintergrundinformationen

Die IR1101 unterscheidet sich etwas von den meisten anderen IOx-Plattformen, da diese hauptsächlich auf x86 basieren. Der IR1101 basiert auf der ARM64v8-Architektur. Daher können Container oder IOx-Pakete, die für x86 erstellt wurden, nicht direkt auf der Plattform bereitgestellt werden. Dieses Dokument startet von Grund auf und bereitet die Umgebung für den Aufbau von ARM64v8-basierten Docker-Containern vor und erläutert, wie diese mit einem x86-PC auf dem IR1101 erstellt, verpackt und bereitgestellt werden.

Als Beispiel wird ein sehr kleines Python-Skript verwendet, das ein einfacher Webserver ist, und ein Docker-Container wird umgebaut, um ihn schließlich auf dem IR1101 zu packen. Der Webserver kann nur einen vordefinierten Port (9000) abhören und eine einfache Seite zurückgeben, wenn er eine **GET**-Anforderung empfängt. Dadurch können Sie die Funktion zum Ausführen von eigenem Code testen und den Netzwerkzugriff auf die IOx-Anwendung testen, sobald diese ausgeführt wird.

Das Paket wird von den Docker-Tools unter Verwendung von Alpine Linux erstellt. Alpine Linux ist ein kleines Linux-Image (ca. 5 MB), das häufig als Basis für Docker-Container verwendet wird.

Da die meisten Desktop-, Laptop- und VM-Systeme auf x86 basieren, müssen Sie die ARM64v8-Architektur auf dem x86-basierten Computer emulieren, auf dem der Container aufgebaut ist. Dies können Sie ganz einfach mithilfe der Quick Emulator (QEMU)-Benutzeremulation durchführen. Dies ermöglicht die Ausführung von ausführbaren Dateien in einer nicht nativen Architektur, genau wie sie in der systemeigenen Architektur ausgeführt wird.

Konfigurieren

Teil 1: Erstellen des IOx-Pakets für IR1101

1. Installieren und Vorbereiten des IOx-Clients auf dem Linux-Host

Sie benötigen **ioxclient**, um den Docker-Container als IOx-Paket zu packen, sobald er erstellt wurde. Lassen Sie uns das also zuerst vorbereiten.

Kopieren oder laden Sie das **ioxclient**-Paket zuerst herunter. Es ist erhältlich unter:
<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
```

Extrahieren Sie das Paket:

```
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
```

Fügen Sie den Pfad zur **PATH**-Variable hinzu, damit diese ohne Verwendung des vollständigen Speicherorts verfügbar ist. Wenn Sie den Computer neu starten oder Benutzer wechseln, sollten Sie diesen Schritt nicht wiederholen:

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

Starten Sie **ioxclient** zum ersten Mal, um ein obligatorisches Profil zu erstellen. Da Sie den Docker-Container nur mit **ioxclient** verpacken, können die Werte als Standardwert beibehalten werden:

```
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. Installieren und Vorbereiten der Docker-Umgebung auf dem Linux Build Machine

Dieser Docker wird verwendet, um einen Container aus dem Alpine-Basisbild zu erstellen und die notwendigen Dateien für den Anwendungsfall einzuschließen. Die angegebenen Schritte basieren auf den offiziellen Installationsanleitungen von Docker Community Edition (CE) für Debian:

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

Aktualisieren Sie die Paketlisten auf Ihrem Computer:

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

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

Installieren Sie die Abhängigkeiten, um den Docker-Report zu verwenden:

```
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) ...
```

Fügen Sie den GPG-Schlüssel (Docker GNU Privacy Guard) als gültigen GPG-Schlüssel hinzu:

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

Überprüfen Sie den Fingerabdruck des installierten GPG-Schlüssels:

```
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]
```

Fügen Sie den stabilen Docker-Report hinzu:

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

Aktualisieren Sie die Paketlisten erneut, während Sie den Docker-Report hinzufügen:

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

Docker installieren:

```
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) ...
```

Um Docker als regulären Benutzer aufrufen/ausführen zu können, fügen Sie diesen Benutzer der Docker-Gruppe hinzu, und aktualisieren Sie die Gruppenmitgliedschaft:

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

3. Installieren der QEMU-Benutzeremulationspakete

Nachdem Sie Docker installiert haben, müssen Sie die QEMU-Benutzeremulatoren installieren. Verwenden Sie den statisch verknüpften QEMU-Emulator aus dem Docker-Container, damit Sie den Container für ARM64v8 auf unserem x86-basierten Linux-Rechner ausführen können, obwohl der Zielcontainer für die ARM64v8-Architektur konzipiert ist.

Installieren Sie die Pakete:

```
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) ...
```

Nach der Installation sind hier die statisch verknüpften QEMU-Emulatoren verfügbar in `/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
...
```

Die erste in der Liste ist die, die Sie benötigen: `aarch64` ist der Bogen-Name für ARM64v8 für Linux.

4. Testen Sie, ob ein Aarch64/ARV64v8-Container auf dem x86-Linux-System ausgeführt wird.

Nachdem Sie Docker installiert haben und die erforderlichen QEMU-Binärdateien installiert sind, können Sie testen, ob Sie einen für ARM64v8 erstellten Docker-Container auf dem x86-Computer ausführen können:

```
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
```

Wie Sie in der Ausgabe sehen können, wird der Alpine-Container `arm64v8` erhalten und mit Zugriff auf den Emulator ausgeführt.

Wenn Sie die Architektur des Containers anfordern, können Sie sehen, dass der Code für `aarch64` kompiliert wird. Genau wie der Zielbogen für das Behältnis sollte für `IR1101` sein.

5. Vorbereiten von Dateien zum Erstellen des Docker-Webserver-Containers

Nachdem alle Vorbereitungen abgeschlossen sind, können Sie die notwendigen Dateien für den Webserver-Container erstellen, der auf `IR1101` ausgeführt werden soll.

Die erste Datei ist `webserver.py`, das Python-Skript, das Sie im Container ausführen möchten. Da dies nur ein Beispiel ist, ersetzen Sie dies natürlich durch den tatsächlichen Code, um in der IOx-Anwendung ausgeführt zu werden:

```
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()

```

Dieser Code enthält die Logik, um in eine Protokolldatei zu schreiben, die zur Abfrage durch den lokalen Manager verfügbar ist.

Die zweite Datei, die benötigt wird, ist die Dockerfile. Dadurch wird festgelegt, wie der Container erstellt wird:

```

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

```

Die Dockerfile definiert, wie der Container erstellt wird. Beginnen Sie vom Apline-Basisbild für ARM64v8, kopieren Sie den Emulator in den Container, führen Sie das apk aus, um das Python-Paket hinzuzufügen und kopieren Sie das Webserver-Skript in den Container.

Die letzte Vorbereitung, die vor dem Erstellen des Containers erforderlich ist, ist das Kopieren von qemu-aarch64-static in das Verzeichnis, aus dem Sie den Container erstellen:

```

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

```

6. Erstellen des Docker-Containers

Nachdem die gesamte Vorbereitung abgeschlossen ist, können Sie den Container mithilfe der Dockerfile erstellen:

```
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
```

Führen Sie als Test den Container aus, den Sie gerade erstellt haben, und überprüfen Sie, ob das Skript funktioniert:

```
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
```

Wie Sie in dieser Ausgabe sehen können, ist die Architektur des Containers die Zielgruppe aarch64. Und nachdem Sie das Skript gestartet haben, sehen Sie, dass es auf Anfragen an Port 9000 wartet.

7. Erstellen des IOx-Pakets

Das Behältnis ist verpackt. Bevor Sie ioxclient dazu auffordern können, müssen Sie zunächst den Paketdeskriptor erstellen: **package.yaml**.

Diese Datei beschreibt, wie das Paket aussehen soll, wie viele Ressourcen es ausführen muss und was es starten soll.

```
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"]
```

Wie Sie sehen, ist die CPU-Architektur auf "aarch64" eingestellt. Um Zugriff auf den TCP-Port 9000 zu erhalten, können Sie **rootfs.tar** als Rootfs verwenden und **python/webserver.py** starten.

Als Letztes müssen Sie das Paket packen, indem Sie die Datei **rootfs.tar** aus dem Docker-Container extrahieren:

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

An diesem Punkt können Sie **ioxclient** verwenden, um das IOx-Paket für IR1101 zu erstellen:

```
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
```

Im Moment gibt es ein Paket, um auf dem IR1101 als package.tar bereitzustellen. Im nächsten Teil wird erläutert, wie das Gerät für die Bereitstellung vorbereitet wird.

Teil 2. Konfigurieren des IR1101 für IOx

1. Aktivieren Sie die Webschnittstelle, IOx und den lokalen Manager.

Local Manager ist eine grafische Benutzeroberfläche (GUI), über die IOx-Anwendungen bereitgestellt, aktiviert, gestartet, verwaltet und Fehler bei diesen behoben werden können. Für IR1101 ist es in die reguläre Management-Webschnittstelle integriert. Sie müssen das also zuerst aktivieren.

Führen Sie diese Schritte auf dem IR1101 aus, um IOx und die Webschnittstelle zu aktivieren.

```
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
```

Die letzte Zeile fügt einen Benutzer mit der Berechtigung 15 hinzu. Dieser Benutzer hat Zugriff auf die Webschnittstelle und den lokalen IOx-Manager.

2. Konfigurieren von IOx-Netzwerken

Bevor Sie auf die Webschnittstelle zugreifen, fügen wir die erforderliche Konfiguration für das IOx-Netzwerk hinzu. Hintergrundinformationen finden Sie in der IR1101-Dokumentation für IOx:

https://www.cisco.com/c/en/us/td/docs/routers/access/1101/software/configuration/guide/b_IR1101_config/b_IR1101config_chapter_010001.html

Kurz gesagt können die IOx-Anwendungen mit der VirtualPortGroup0-Schnittstelle mit der Außenwelt kommunizieren (vergleichbar mit Gi2 für IR809 und Gi5 für IR829-Schnittstellen).

```
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
```

Wenn Sie die VirtualPortGroup0-Schnittstelle als Network Address Translation (NAT) innerhalb konfigurieren, müssen Sie die externe IP-nat-Anweisung an der Gi 0/0/0-Schnittstelle hinzufügen, um die Kommunikation mit und von den IOx-Anwendungen mithilfe von NAT zu ermöglichen:

```
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
```

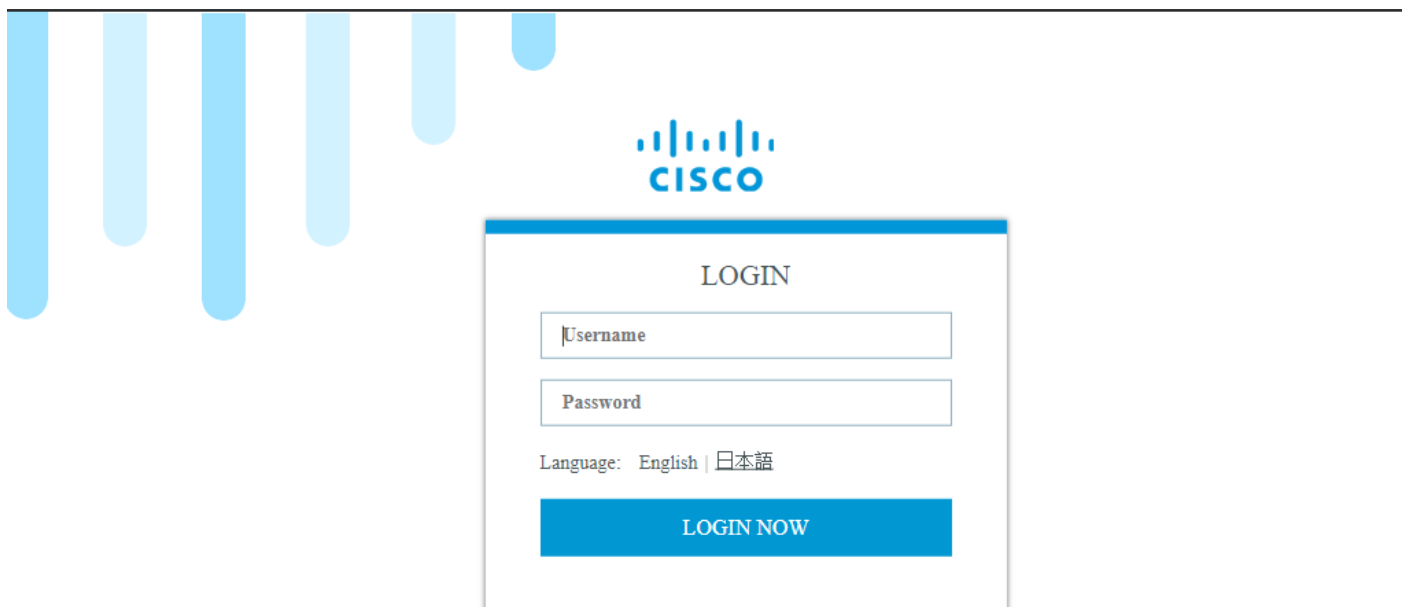
Um den Zugriff auf Port 9000 für den Container zu ermöglichen, den Sie 192.168.1.15 geben können, müssen Sie einen Port vorwärts hinzufügen:

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

Verwenden Sie für dieses Handbuch statisch konfigurierte IPs pro IOx-Anwendung. Wenn Sie den Anwendungen dynamisch IP-Adressen zuweisen möchten, müssen Sie die Konfiguration für einen DHCP-Server im Subnetz von VirtualPortGroup0 hinzufügen.

Teil 3. Zugriff auf den lokalen Manager und Bereitstellung der IOx-Anwendung

Nachdem Sie diese Leitungen zur Konfiguration hinzugefügt haben, können Sie mithilfe der Webschnittstelle auf die IR1101 zugreifen. Navigieren Sie mithilfe Ihres Browsers zur IP-Adresse Gi 0/0/0, wie im Bild gezeigt.



© 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.

Verwenden Sie das in Schritt 1 erstellte Konto mit der Berechtigung 15. um sich bei der Webschnittstelle anzumelden und zu **Configuration - IOx** zu navigieren, wie im Bild gezeigt.



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

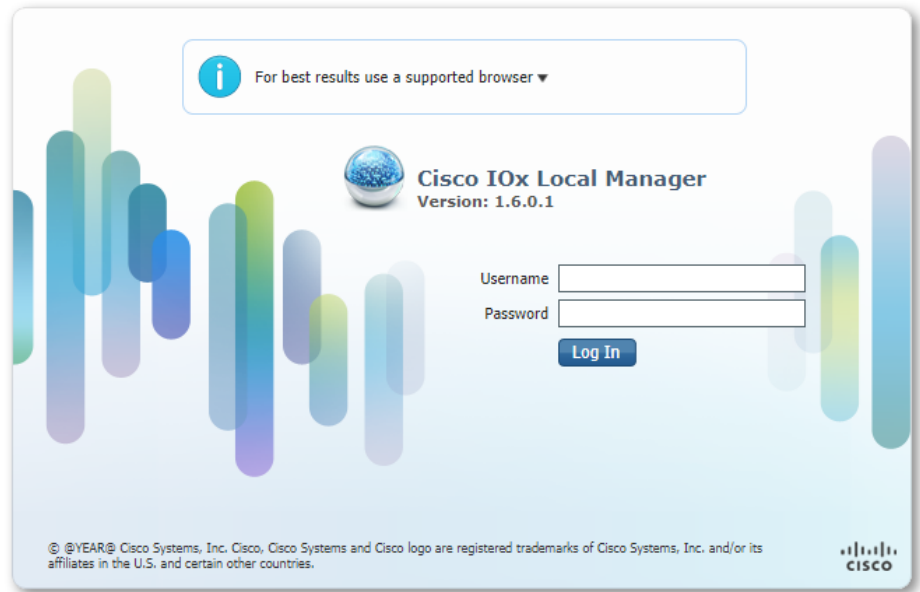
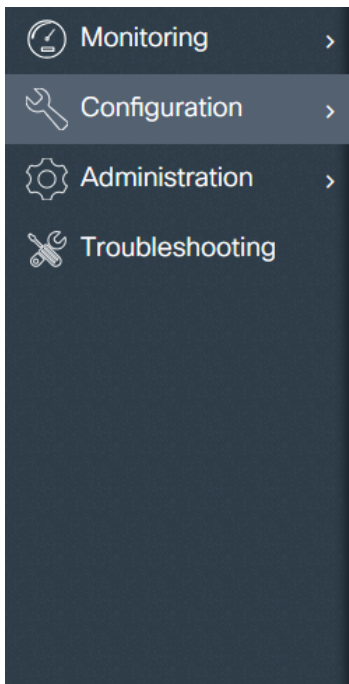
Application Visibility

Custom Application

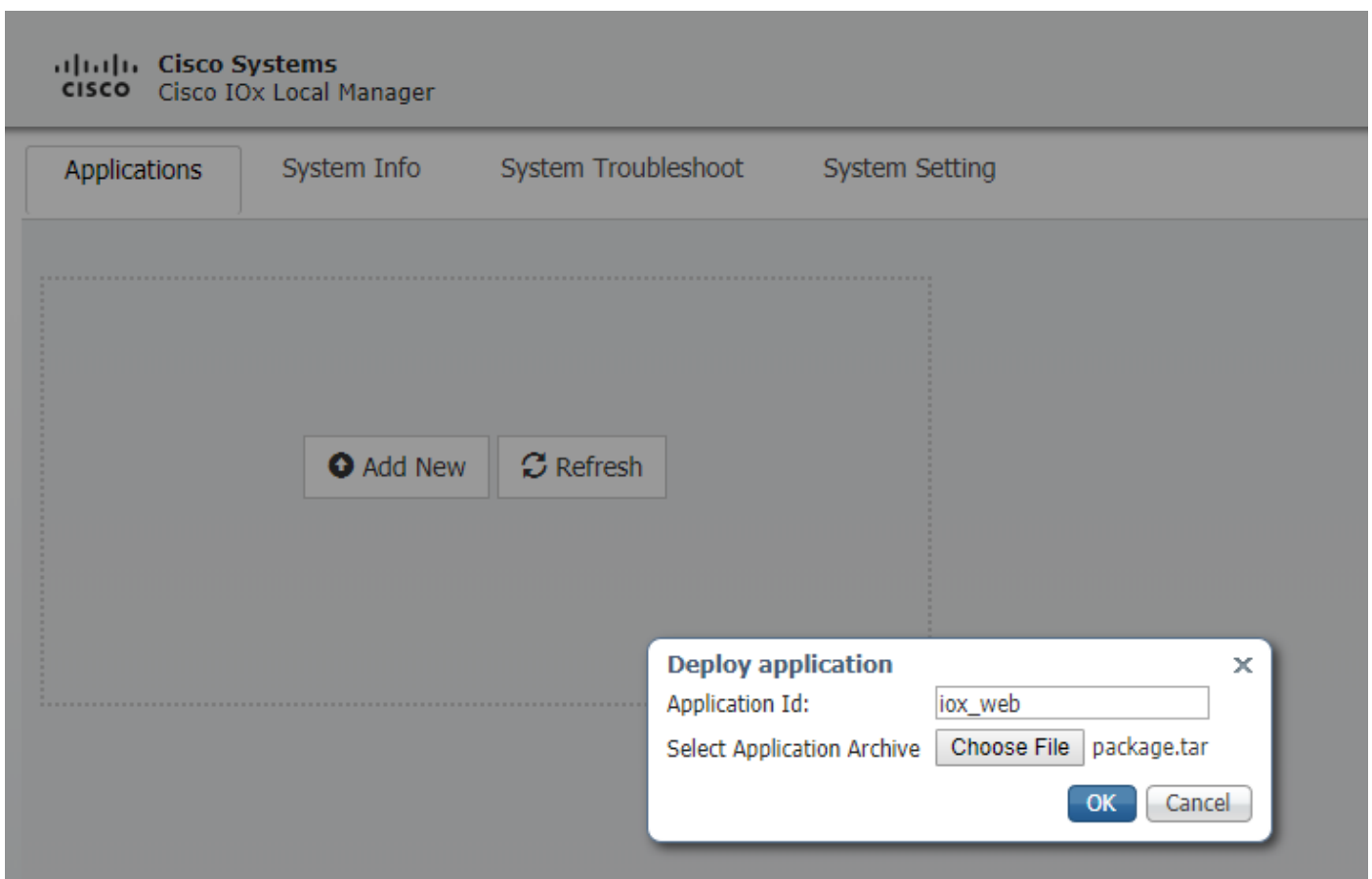
IOx

NETFLOW

Verwenden Sie bei der Anmeldung beim IOx-lokalen Manager dasselbe Konto, um fortzufahren, wie im Bild gezeigt.



Klicken Sie auf **Add New**, wählen Sie einen Namen für die IOx-Anwendung aus, und wählen Sie die in Teil 1 erstellte package.tar aus, wie im Bild gezeigt.



Nachdem das Paket hochgeladen wurde, können Sie es wie im Bild gezeigt aktivieren.

iox_web

DEPLOYED

simple docker webserver for arm64v8

TYPE	VERSION	PROFILE
docker	1.0	c1.tiny

Memory *

6.3%

CPU *

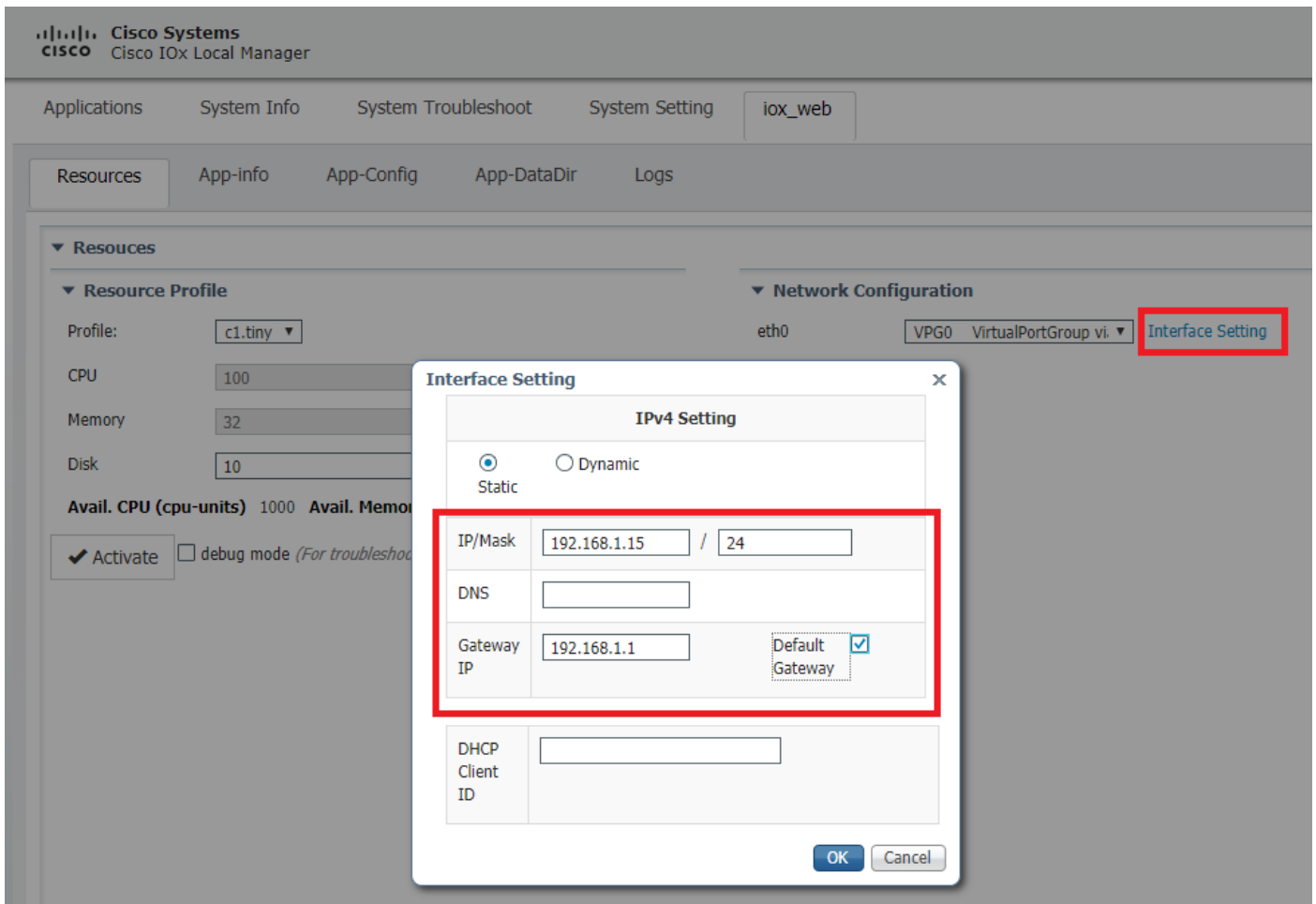
10.0%

✓ Activate

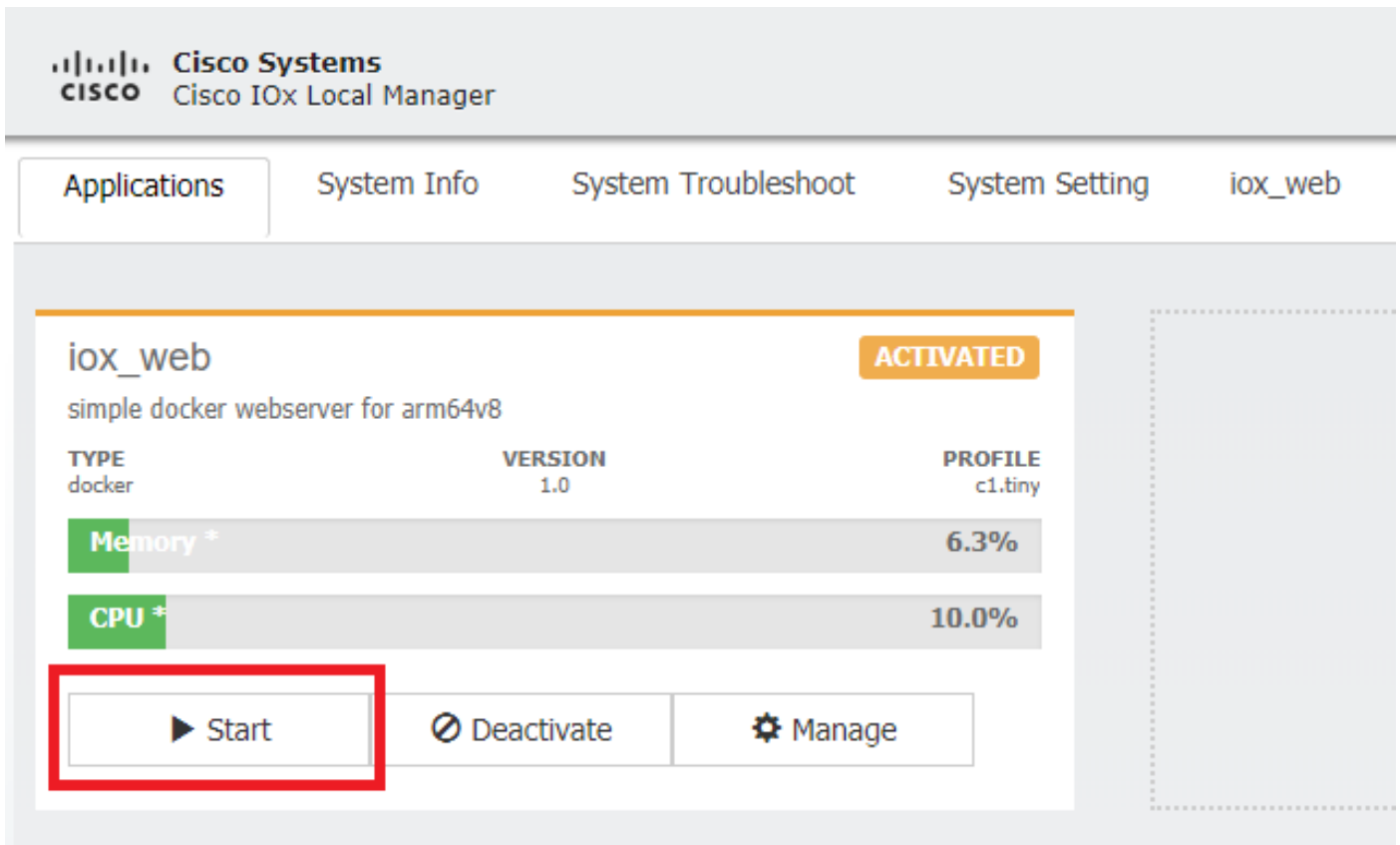
Upgrade

Delete

Öffnen Sie auf der Registerkarte **Resources** (Ressourcen) die Schnittstelleneinstellung, um die feste IP anzugeben, die Sie der App zuweisen möchten, wie im Bild gezeigt.



Klicken Sie auf **OK** und dann auf **Aktivieren**. Navigieren Sie nach Abschluss der Aktion zurück zur Hauptseite für den lokalen Manager (Schaltfläche **Anwendungen** im oberen Menü), und starten Sie die Anwendung wie im Bild gezeigt.



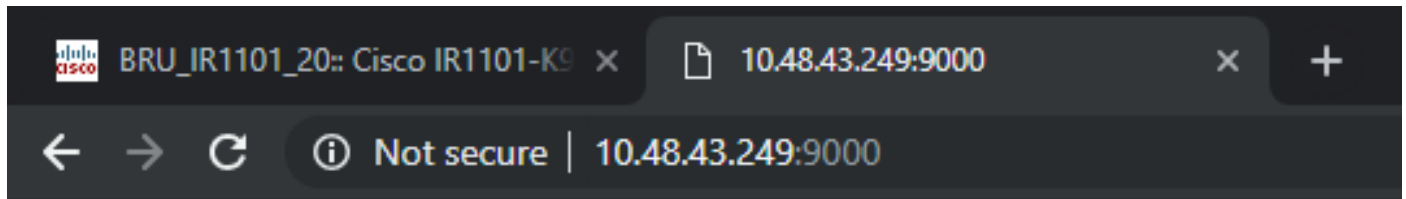
Nachdem Sie diese Schritte ausgeführt haben, sollte die Anwendung über Port 9000 mit der Gi 0/0/0-Schnittstelle des IR1101 ausgeführt und verfügbar sein.

Überprüfen

In diesem Abschnitt überprüfen Sie, ob Ihre Konfiguration ordnungsgemäß funktioniert.

Um dies zu überprüfen, können Sie mit Port 9000 auf die IP-Adresse der Gi 0/0/0-Schnittstelle des IR1101 zugreifen.

Wenn alles gut geht, sollten Sie dies wie folgt sehen, wie es im Python-Skript erstellt wurde.



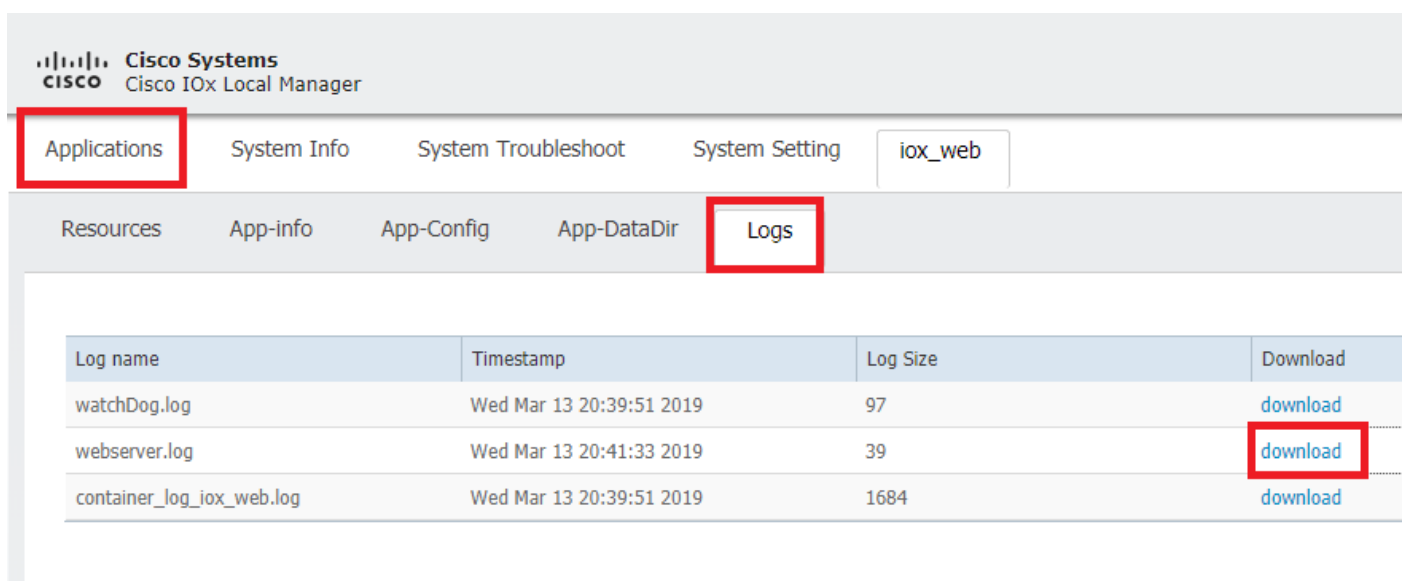
IOX python webserver on arm64v8

Fehlerbehebung

Dieser Abschnitt enthält Informationen, die Sie zur Fehlerbehebung bei Ihrer Konfiguration verwenden können.

Zur Fehlerbehebung können Sie die im Python-Skript erstellte Protokolldatei mithilfe eines lokalen Managers überprüfen.

Navigieren Sie zu **Anwendungen**, klicken Sie auf **Verwalten** in der Anwendung **iox_web**, und wählen Sie dann die Registerkarte **Protokolle** aus, wie im Bild gezeigt.



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