

Créer et déployer un package IOx Docker pour l'architecture ARM IR1101

Contenu

[Introduction](#)

[Conditions préalables](#)

[Conditions requises](#)

[Components Used](#)

[Informations générales](#)

[Configuration](#)

[Partie 1. Créer le package IOx pour IR1101](#)

[1. Installer et préparer le client IOx sur l'hôte Linux](#)

[2. Installation et préparation de l'environnement Docker sur la machine de build Linux](#)

[3. Installer les packages d'émulation utilisateur QEMU](#)

[4. Vérifier si un conteneur aarch64/ARV64v8 s'exécute sur une machine Linux x86](#)

[5. Préparer les fichiers pour créer le conteneur de serveur Web Docker](#)

[6. Construire le conteneur Docker](#)

[7. Créer le package IOx](#)

[Partie 2. Configuration du IR1101 pour IOx](#)

[1. Activer l'interface Web, IOx et Local Manager](#)

[2. Configuration de la mise en réseau IOx](#)

[Partie 3. Accéder au gestionnaire local et déployer l'application IOx](#)

[Vérification](#)

[Dépannage](#)

Introduction

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

Conditions préalables

Conditions requises

Cisco vous recommande de prendre connaissance des rubriques suivantes :

- Linux
- Conteneurs
- IOx

Components Used

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 (extension) est utilisée pour cet article)
- Fichiers d'installation du client IOx téléchargeables à l'adresse :

<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 un peu différent par rapport à la plupart des autres plates-formes d'E/S car elles sont principalement basées sur x86. L'IR1101 est basé sur l'architecture ARM64v8. Vous ne pouvez donc pas déployer directement des conteneurs ou des packages IOx conçus pour x86 sur la plate-forme. Ce document commence à zéro et prépare l'environnement pour la construction de conteneurs Docker basés sur ARM64v8 et explique comment les construire, les emballer et les déployer sur le IR1101 avec l'utilisation d'un PC x86.

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

Le paquet sera 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 x86 sur laquelle 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 elle s'exécuterait sur son architecture native.

Configuration

Partie 1. Créer le package IOx pour IR1101

1. Installer et préparer le client IOx sur l'hôte Linux

Vous avez besoin d'`ioxclient` pour empaqueter le conteneur Docker en tant que paquet IOx une fois qu'il a été construit, alors préparons-le d'abord.

Copiez ou téléchargez d'abord le paquet `ioxclient`. Il est disponible à l'adresse :

<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
```

Extraire 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 l'ordinateur 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'utiliserez **ioxclient** que pour emballer le conteneur Docker, les valeurs peuvent être laissées 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 build Linux

Ce Docker est utilisé pour construire un conteneur à partir de l'image de base alpine et pour inclure les fichiers nécessaires au cas d'utilisation. Les étapes indiquées sont basées sur les guides d'installation officiels de Docker Community Edition (CE) pour Debian :

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

Mettre 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-properties-common  
Reading package lists... Done  
Building dependency tree  
...  
Processing triggers for dbus (1.10.26-0+deb9u1) ...
```

Ajoutez la clé Docker GNU Privacy Guard (GPG) en tant que 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 la référence stable Docker :

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

Mettre à jour les listes de packages à nouveau lors de l'ajout du repo 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) ...
```

Afin de pouvoir accéder/exécuter Docker en tant qu'utilisateur régulier, 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. Installer les packages d'émulation utilisateur QEMU

Après avoir installé Docker, vous devez installer les émulateurs utilisateur QEMU. Utilisez l'émulateur QEMU lié de manière statique depuis le conteneur Docker pour exécuter le conteneur 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 de manière statique 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
...
```

La première de la liste est celle dont vous avez besoin : `aarch64` est le nom d'arc pour ARM64v8 pour Linux.

4. Vérifier si un conteneur aarch64/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 l'ordinateur x86 :

```
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
```

Comme vous pouvez le voir dans la sortie, `arm64v8` conteneur alpin 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 `arch64`. Exactement comme l'arc cible du conteneur doit être pour `IR1101`.

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

Maintenant que toute la préparation est terminée, vous pouvez continuer et 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 allez évidemment remplacer ceci par le code réel afin de s'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 afin d'écrire dans un fichier journal, qui sera disponible pour consultation du gestionnaire local.

Le deuxième fichier nécessaire est le fichier Dockerfile. Cela définit la façon 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 fichier Dockerfile définit comment le conteneur sera construit. Démarrez à partir de l'image de base d'Alpine pour ARM64v8, copiez l'émulateur dans le conteneur, exécutez le paquet afin d'ajouter le paquet Python et copiez le script du serveur Web dans le conteneur.

La dernière préparation nécessaire avant de pouvoir construire le conteneur est de copier `qemu-aarch64-static` dans le répertoire d'où vous construirez le conteneur :

```

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

```

6. Construire le conteneur Docker

Maintenant que la préparation est terminée, vous pouvez construire le conteneur à l'aide du fichier

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
```

Comme 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 -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
```

Comme vous pouvez le voir dans ce résultat, l'architecture du conteneur est l'arc ciblé64. Et après avoir lancé le script, vous voyez qu'il écoute les requêtes sur le port 9000.

7. Créer le package IOx

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

Ce fichier décrit comment le package doit être présenté, le nombre de ressources nécessaires à son exécution et le démarrage.

```
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"]
```

Comme vous pouvez le voir, l'architecture du processeur est définie sur arch64. Afin d'accéder au port TCP 9000, utilisez **rootfs.tar** comme toit et au démarrage, vous pouvez exécuter **python/webserver.py**.

La dernière chose à faire avant de pouvoir emballer est d'extraire **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
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
```

Pour le moment, il existe un paquet afin de le déployer sur le IR1101 prêt en tant que package.tar. La partie suivante explique comment préparer le périphérique pour le déploiement.

Partie 2. Configuration du IR1101 pour IOx

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

Local Manager est une interface utilisateur graphique qui permet de déployer, d'activer, de démarrer, de gérer et de dépanner des applications IOx. Pour IR1101, il est intégré dans 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 disposant de 15 autorisations de privilège. Cet utilisateur aura 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 le réseau IOx. Des informations générales sont disponibles dans la documentation IR1101 pour IOx :

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

En bref, les applications IOx peuvent communiquer avec le monde extérieur grâce à l'utilisation de l'interface VirtualPortGroup0 (comparable au Gi2 sur IR809 et au 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 que traduction d'adresses de réseau (NAT) 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 de permettre l'accès au port 9000 pour le conteneur, que vous pouvez donner 192.168.1.15,

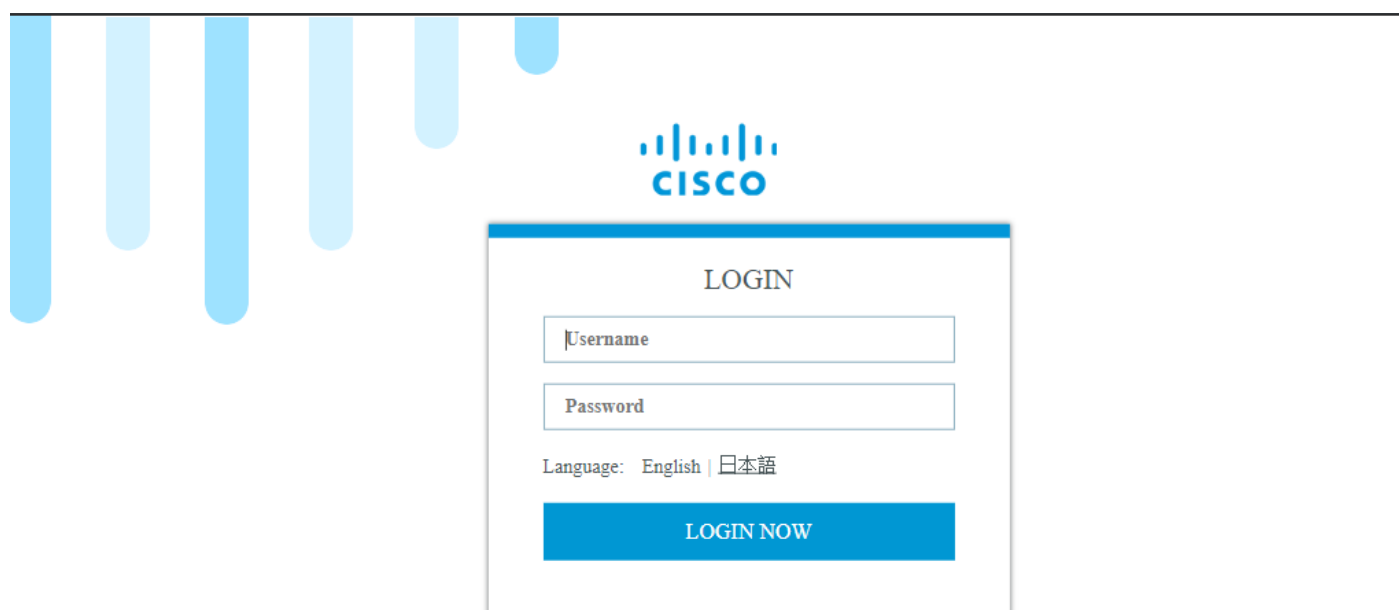
vous devez ajouter un port de transfert :

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

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 au gestionnaire local et déployer l'application IOx

Après avoir ajouté ces lignes à la configuration, vous pouvez accéder au routeur 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.



© 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.

Utilisez le compte privilégié 15 créé à l'étape 1. afin de se connecter à l'interface web et naviguer jusqu'à **Configuration** - IOx comme indiqué dans l'image.



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

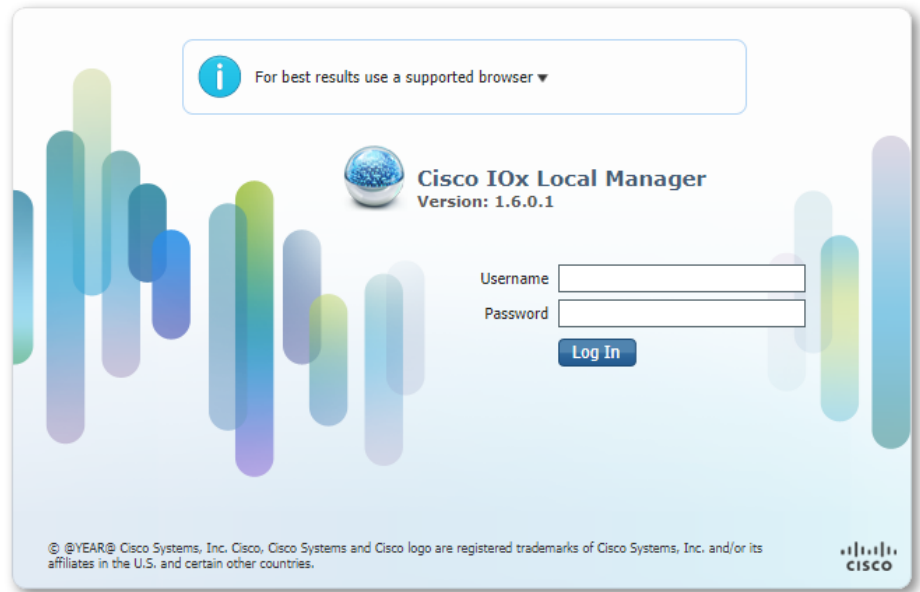
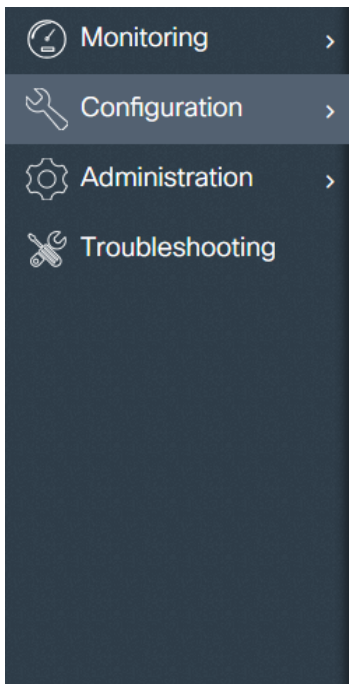
Application Visibility

Custom Application

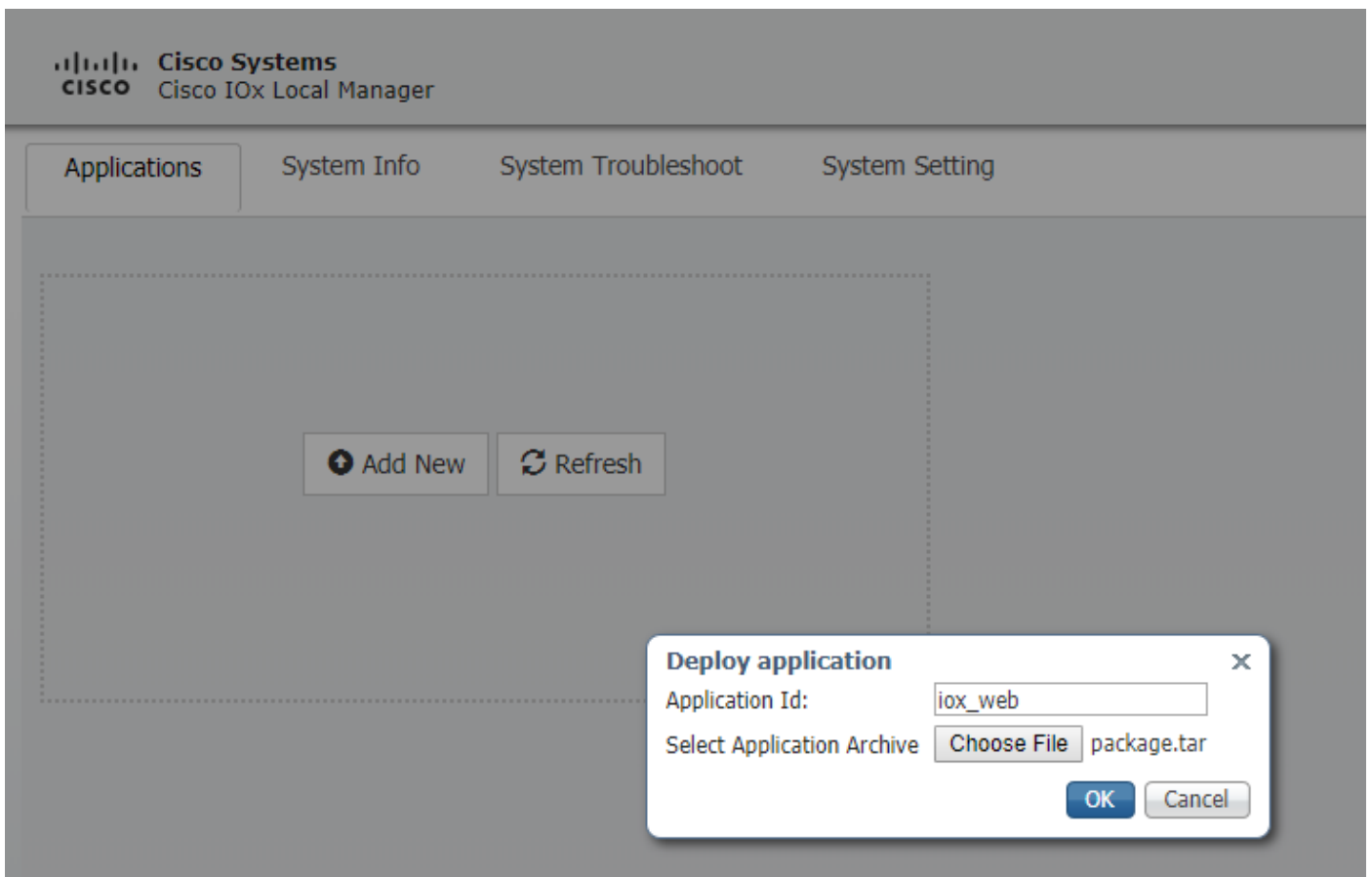
IOx

NETFLOW

Dans la connexion IOx Local Manager, utilisez le même compte pour continuer comme indiqué dans l'image.



Cliquez sur **Ajouter nouveau**, sélectionnez un nom pour l'application IOx et choisissez le package.tar qui a été construit dans la Partie 1 comme indiqué dans l'image.



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

iox_web

DEPLOYED

simple docker webserver for arm64v8

TYPE	VERSION	PROFILE
docker	1.0	c1.tiny

Memory *	6.3%
----------	------

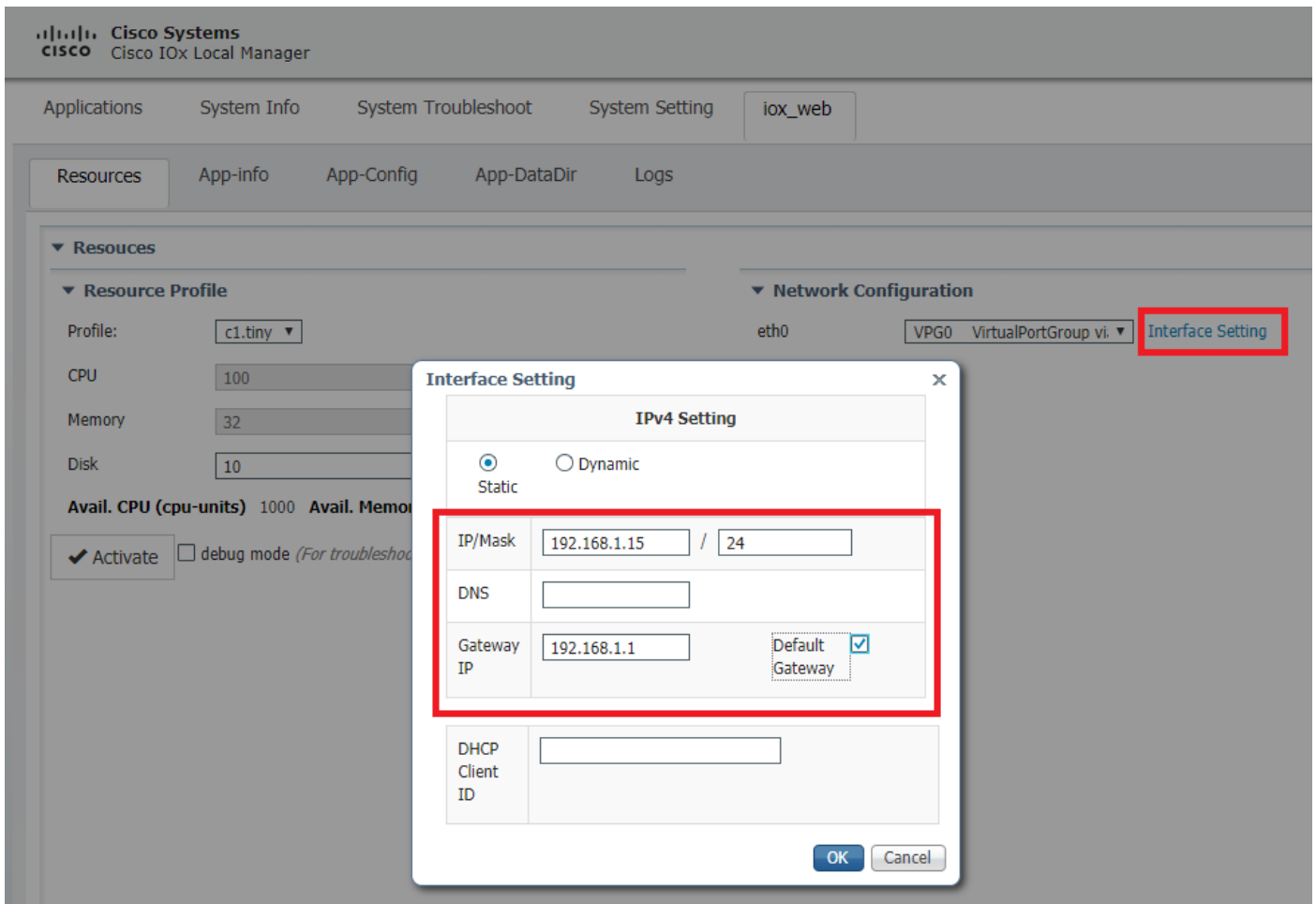
CPU *	10.0%
-------	-------

✓ Activate

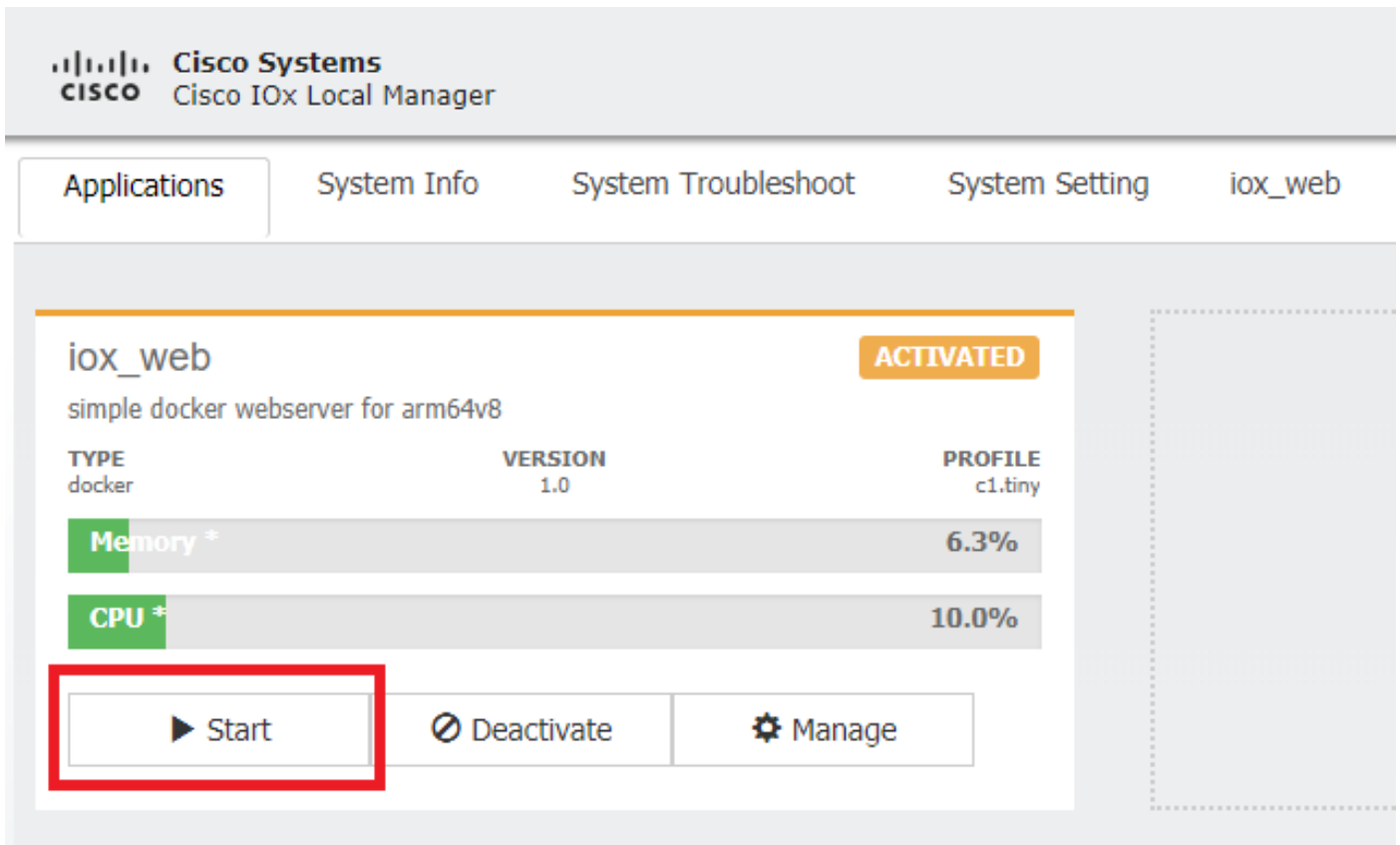
Upgrade

Delete

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



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



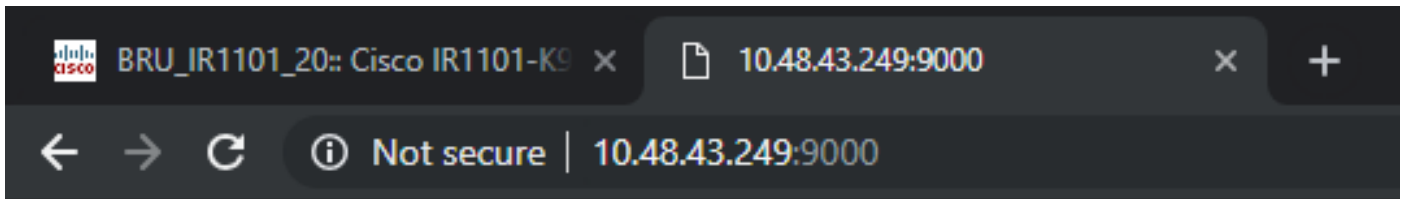
Après avoir suivi ces étapes, votre application doit être exécutée et disponible via le port 9000 avec l'utilisation de l'interface Gi 0/0/0 du IR1101.

Vérification

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 le IR1101 avec l'utilisation du port 9000.

Si tout va bien, vous devriez voir ceci comme suit, comme il a été créé dans le script Python.



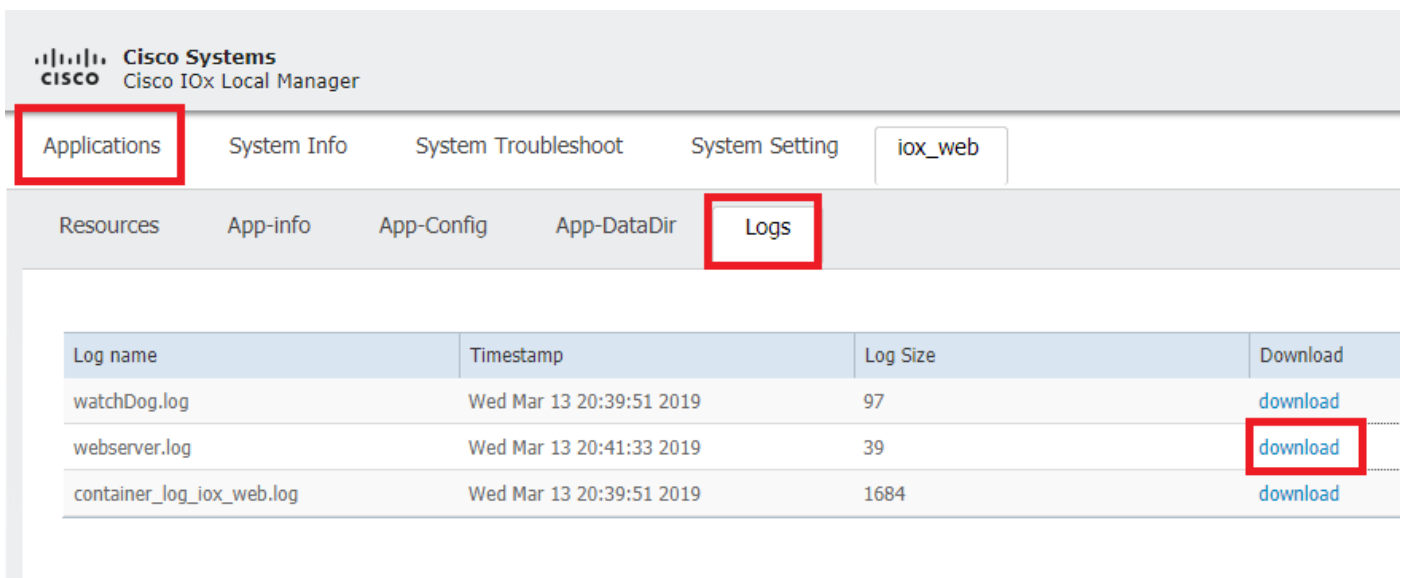
IOX python webserver on arm64v8

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 à l'aide d'un gestionnaire local.

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



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