IOx Application Hosting

This section contains the following topics:

- Application Hosting, on page 1
- Information About Application Hosting, on page 1
- Application Hosting on the IR1800 Industrial Integrated Services Router, on page 3
- How to Configure Application Hosting, on page 5
- Installing and Uninstalling Apps, on page 8
- Overriding the App Resource Configuration, on page 8
- Verifying the Application Hosting Configuration, on page 10
- Configuration Examples for Application Hosting, on page 11
- Native docker support, on page 12
- Digital IO for IOx container applications, on page 13
- Signed Application Support, on page 14

Application Hosting

A hosted application is a software as a service solution, and it can be run remotely using commands. Application hosting gives administrators a platform for leveraging their own tools and utilities.

This module describes the Application Hosting feature and how to enable it.

Information About Application Hosting

This section contains the following:

Need for Application Hosting

The move to virtual environments has given rise to the need to build applications that are reusable, portable, and scalable. Application hosting gives administrators a platform for leveraging their own tools and utilities. An application, hosted on a network device, can serve a variety of purposes. This ranges from automation, configuration management monitoring, and integration with existing tool chains.

Cisco devices support third-party off-the-shelf applications built using Linux tool chains. Users can run custom applications cross-compiled with the software development kit that Cisco provides.

IOx Overview

IOx is a Cisco-developed end-to-end application framework that provides application hosting capabilities for different application types on Cisco network platforms.

IOx architecture for the IR1800 is different compared to other Cisco platforms that use the hypervisor approach. In other platforms, IOx runs as a virtual machine. IOx is running as a process on the IR1800.

Cisco Application Hosting Overview

The IR1800 enables the user to deploy the application using the app-hosting CLIs. These app-hosting CLIs are not available on the other older platforms. There are additional ways to deploy the applications using the Local Manager and Fog Director.

Application hosting provides the following services:

- Launches designated applications in containers.
- Checks available resources (memory, CPU, and storage), and allocates and manages them.
- Provides support for console logging.
- Provides access to services via REST APIs.
- Provides a CLI endpoint.
- Provides an application hosting infrastructure referred to as Cisco Application Framework (CAF).
- Helps in the setup of platform-specific networking (packet-path) via VirtualPortGroup and management interfaces.

The container is referred to as the virtualization environment provided to run the guest application on the host operating system. The Cisco IOS-XE virtualization services provide manageability and networking models for running guest applications. The virtualization infrastructure allows the administrator to define a logical interface that specifies the connectivity between the host and the guest. IOx maps the logical interface into the Virtual Network Interface Card (vNIC) that the guest application uses.

Applications to be deployed in the containers are packaged as TAR files. The configuration that is specific to these applications is also packaged as part of the TAR file.

The management interface on the device connects the application hosting network to the IOS management interface. The Layer 3 interface of the application receives the Layer 2 bridged traffic from the IOS management interface. The management interface connects through the management bridge to the container/application interface. The IP address of the application must be on the same subnet as the management interface IP address.

IOXMAN

IOXMAN is a process that establishes a tracing infrastructure to provide logging or tracing services for guest applications, except Libvirt, that emulates serial devices. IOXMAN is based on the lifecycle of the guest application to enable and disable the tracing service, to send logging data to IOS syslog, to save tracing data to IOx tracelog, and to maintain IOx tracelog for each guest application.

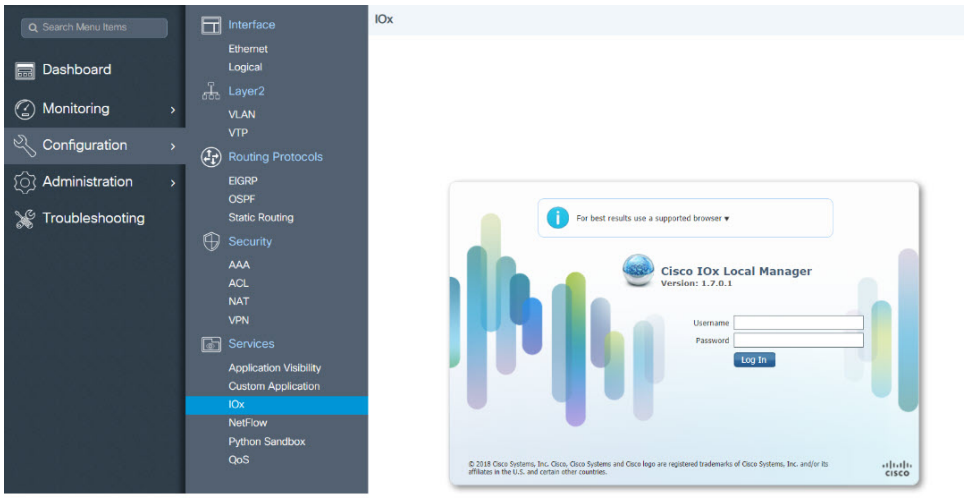
Application Hosting on the IR1800 Industrial Integrated Services Router

This section describes the application-hosting characteristics specific to the IR1800 Industrial Integrated Services Router.

Note The IR1800 CPU is not based on x86 architecture like other Routers. Therefore, this requires the application to comply with the ARM 64-bits architecture.

Application hosting can be achieved using the app-hosting cli's as well using the Local Manager and Fog Director. Application hosting using Local Manager is done through the WebUI. In order to deply the applications using Local Manager, WebUI should be enabled and then login to the Local Manager.

Figure 1: Local Manager



1. From the WebUI, click on **Configuration > Services > IOx**
2. Login using the username and password configured.
3. Follow the steps for the application lifecycle in the **Cisco IOx Local Manager Reference Guide** using this link: https://www.cisco.com/c/en/us/td/docs/routers/access/800/software/guides/iox/lm/reference-guide/1-7/b_iox_lm_ref_guide_1_7/b_iox_lm_ref_guide_1_7_chapter_011.html

The next section explains the deployment of an application using the app-hosting cli's.

VirtualPortGroup

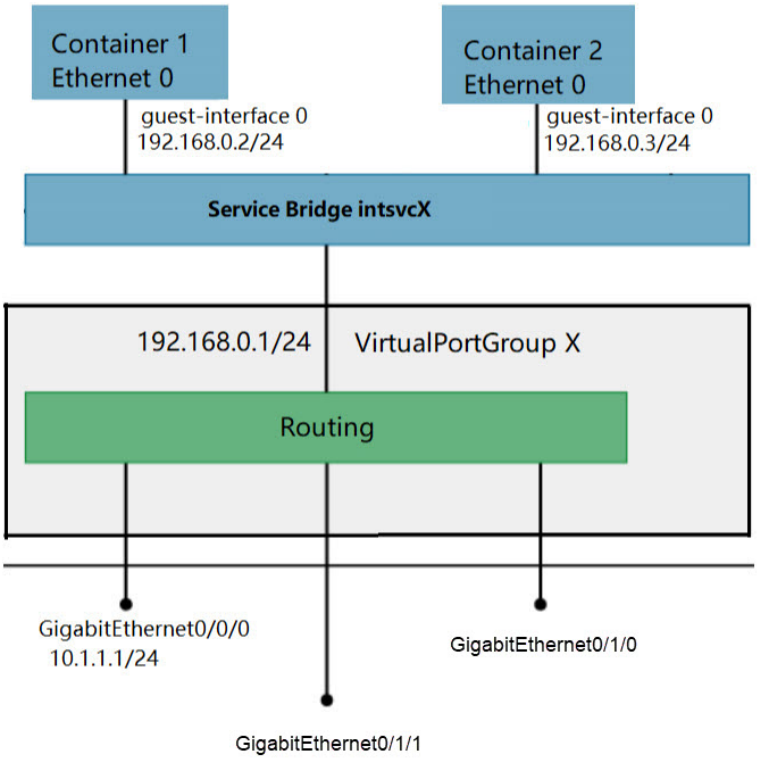
The VirtualPortGroup is a software construct on Cisco IOS that maps to a Linux bridge IP address. As such, the VirtualPortGroup represents the switch virtual interface (SVI) of the Linux container. Each bridge can contain multiple interfaces; each mapping to a different container. Each container can also have multiple interfaces.

VirtualPortGroup interfaces are configured by using the interface virtualportgroup command. Once these interfaces are created, IP address and other resources are allocated.

The VirtualPortGroup interface connects the application hosting network to the IOS routing domain. The Layer 3 interface of the application receives routed traffic from IOS. The VirtualPortGroup interface connects through the SVC Bridge to the container/application interface.

The following graphic helps to understand the relationship between the VirtualPortGroup and other interfaces, as it is different than the IR8x9 routers.

Figure 2: Virtual Port Group Mapping



vNIC

For the container life cycle management, the Layer 3 routing model that supports one container per internal logical interface is used. This means that a virtual Ethernet pair is created for each application; and one interface of this pair, called vNIC is part of the application container. The other interface, called vpgX is part of the host system.

NIC is the standard Ethernet interface inside the container that connects to the platform dataplane for the sending and receiving of packets. IOx is responsible for the gateway (VirtualPortGroup interface), IP address, and unique MAC address assignment for each vNIC in the container.

The vNIC inside the container/application are considered as standard Ethernet interfaces.

How to Configure Application Hosting

This section contains the following:

Enabling IOx

Perform this task to enable access to the IOx Local Manager. The IOx Local Manager provides a web-based user interface that you can use to manage, administer, monitor, and troubleshoot apps on the host system, and to perform a variety of related activities

Note In the steps that follow, IP HTTP commands do not enable IOX, but allow the user to access the WebUI to connect the IOX Local Manager.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device>enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device#configure terminal`	Enters global configuration mode.
Step 3	**iox** **Example:** `Device(config)#iox`	Enables IOx.
Step 4	**ip http server** **Example:** `Device(config)#ip http server`	Enables the HTTP server on your IP or IPv6 system.
Step 5	**ip http secure-server** **Example:** `Device(config)#ip http secure-server`	Enables a secure HTTP (HTTPS) server.
Step 6	**username** *name* **privilege** *level* **password** {**0** | **7** | *user-password* }*encrypted-password* **Example:** `Device(config)#username cisco privilege 15 password 0 cisco`	Establishes a username-based authentication system and privilege level for the user. The username privilege level must be configured as 15.

	Command or Action	Purpose
Step 7	**end** **Example:** `Device(config-if)#end`	Exits interface configuration mode and returns to privileged EXEC mode

Configuring a VirtualPortGroup to a Layer 3 Data Port

Multiple Layer 3 data ports can be routed to one or more VirtualPortGroups or containers. VirutalPortGroups and Layer 3 data ports must be on different subnets.

Enable the **ip routing** command to allow external routing on the Layer 3 data-port.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device>enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device#configure terminal`	Enters global configuration mode.
Step 3	**ip routing** **Example:** `Device(config)#ip routing`	Enables IP routing. The **ip routing** command must be enabled to allow external routing on Layer 3 data ports.
Step 4	**interface** *type number* **Example:** `Device(config)#interface gigabitethernet 0/0/0`	Configures an interface and enters interface configuration mode.
Step 5	**no switchport** **Example:** `Device(config)#no switchport`	Places the interface in Layer 3 mode, and makes it operate more like a router interface rather than a switch port.
Step 6	**ip address** *ip-address mask* **Example:** `Device(config)#ip address 10.1.1.1 255.255.255.0`	Configures an IP address for the interface.
Step 7	**exit** **Example:** `Device(config-if)#exit`	Exits interface configuration mode and returns to global configuration mode.

	Command or Action	Purpose
Step 8	**interface** *type number* **Example:** `Device(config)#interface virtualportgroup 0`	Configures an interface and enters interface configuration mode.
Step 9	**ip address** *ip-address mask* **Example:** `Device(config-if)#ip address 192.168.0.1 255.255.255.0`	Configures an IP address for the interface.
Step 10	**end** **Example:** `Device(config-if)#end`	Exits interface configuration mode and returns to privileged EXEC mode
Step 11	**configure terminal** **Example:** `Device#configure terminal`	Enters global configuration mode.
Step 12	**app-hosting appid** *app_number* **Example:** `Device(config)#app-hosting appid app1`	Configures the application and enters the application configuration mode.
Step 13	**app-vnic gateway0 virtualportgroup 0 guest-interface 0** **Example:** `Device(config-app-hosting)#app-vnic gateway0 virtualportgroup 0 guest-interface 0`	Configures the application interface and the gateway of the application.
Step 14	**guest-ipaddress** *ip_address* **netmask** *netmask* **Example:** `Device(config-app-hosting-gateway0)#guest-ipaddress 192.168.0.2 netmask 255.255.255.0`	Configures the application Ethernet interface ip address.
Step 15	**app-default-gateway** *ip_address* **guest-interface 0** **Example:** `Device(config-app-hosting-gateway0)#app-default-gateway 192.168.0.1 guest-interface 0`	Configures the default gateway for the application.
Step 16	**end** **Example:** `Device#end`	Exits interface configuration mode and returns to privileged EXEC mode

Installing and Uninstalling Apps

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device>enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device#configure terminal`	Enters global configuration mode.
Step 3	**app-hosting install appid** *application-name* **package** *package-path* **Example:** `Device#app-hosting install appid lxc_app package flash:my_iox_app.tar`	Installs an app from the specified location. The app can be installed from any local storage location such as, flash, bootflash, and usbflash0.
Step 4	**app-hosting start appid** *application-name* **Example:** `Device#app-hosting start appid app1`	Starts the application. Application start-up scripts are activated.
Step 5	**app-hosting stop appid** *application-name* **Example:** `Device#app-hosting stop appid app1`	Stops the application.
Step 6	**app-hosting deactivate appid** *application-name* **Example:** `Device#app-hosting deactivate appid app1`	Deactivates all resources allocated for the application.
Step 7	**app-hosting uninstall appid** *application-name* **Example:** `Device#app-hosting uninstall appid app1`	Uninstalls the application. Uninstalls all packaging and images stored. All changes and updates to the application are also removed.

Overriding the App Resource Configuration

Resource changes will take effect only after the app-hosting activate command is configured.

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device>enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device#configure terminal`	Enters global configuration mode.
Step 3	**app-hosting appid** *name* **Example:** `Device(config-app-hosting)#app-resource profile custom`	Enables application hosting and enters application hosting configuration mode.
Step 4	**app-resource profile** *name* **Example:** `Device#app-hosting start appid app1`	Configures the custom application resource profile, and enters custom application resource profile configuration mode. Only the custom profile name is supported.
Step 5	**cpu** *unit* **Example:** `Device(config-app-resource-profile-custom)# cpu 800`	Changes the default CPU allocation for the application. Resource values are application-specific, and any adjustment to these values must ensure that the application can run reliably with the changes.
Step 6	**memory** *memory* **Example:** `Device(config-app-resource-profile-custom)# memory 512`	Changes the default memory allocation.
Step 7	**vcpu** *number* **Example:** `Device(config-app-resource-profile-custom)# vcpu 2`	Changes the virtual CPU (vCPU) allocation for the application.
Step 8	**end** **Example:** `Device(config-app-resource-profile-custom)# end`	Exits custom application resource profile configuration mode and returns to privileged EXEC mode.

Verifying the Application Hosting Configuration

Procedure

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device>enable`	Enables privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device#configure terminal`	Enters global configuration mode.

Step 3	**show iox-service**
	Example:
	`Device(config)#show iox-service`
	`IOx Infrastructure Summary:`
	`---`
	`IOx service (CAF) 1.8.0.2 : Running`
	`IOx service (HA) : Not Supported`
	`IOx service (IOxman) : Running`
	`Libvirtd 1.3.4 : Running`
	`Device#`
		Displays the status of all IOx services.

Step 4	**show app-hosting detail**
	Example:
	`Device#show app-hosting detail`
	`App id : app1`
	`Owner : iox`
	`State : RUNNING`
	`Application`
	`Type : lxc`
	`Name : nt08-stress`
	`Version : 0.1`
	`Description : Stress Testing`
	`Application`
	`Path : usbflash0:my_iox_app.tar`
	`Activated profile name : custom`
	`Resource reservation`
	`Memory : 64 MB`
	`Disk : 2 MB`
	`CPU : 500 units`
	`Attached devices`
	`Type Name Alias`
	`---`
	`serial/shell iox_console_shell serial0`
	`serial/aux iox_console_aux serial1`
		Displays detailed information about the application.

	Command or Action	Purpose
	serial/syslog iox_syslog serial2 serial/trace iox_trace serial3 Network interfaces ---------- eth0: MAC address : 52:54:dd:fa:25:ee	
Step 5	**show app-hosting list** **Example:** Device#**show app-hosting list** App id State ---------- app1 RUNNING	Displays the list of applications and their status.

Configuration Examples for Application Hosting

See the following examples:

Example: Enabling IOx

```
Device> enable
Device# configure terminal
Device(config)# iox
Device(config)# ip http server
Device(config)# ip http secure-server
Device(config)# username cisco privilege 15 password 0 cisco
Device(config)# end
```

Example: Configuring a VirtualPortGroup to a Layer 3 Data Port

```
Device> enable
Device# configure terminal
Device(config)# ip routing
Device(config)# interface gigabitethernet 0/0/0
Device(config-if)# no switchport
Device(config-if)# ip address 10.1.1.1 255.255.255.0
Device(config-if)# exit
Device(config)# interface virtualportgroup 0
Device(config-if)# ip address 192.168.0.1 255.255.255.0
Device(config-if)# end
```

Example: Installing and Uninstalling Apps

```
Device> enable
```

```
Device# app-hosting install appid app1 package flash:my_iox_app.tar
Device# app-hosting activate appid app1
Device# app-hosting start appid app1
Device# app-hosting stop appid app1
Device# app-hosting deactivate appid app1
Device# app-hosting uninstall appid app1
```

Example: Overriding the App Resource Configuration

```
Device# configure terminal
Device(config)# app-hosting appid app1
Device(config-app-hosting)# app-resource profile custom
Device(config-app-resource-profile-custom)# cpu 800
Device(config-app-resource-profile-custom)# memory 512
Device(config-app-resource-profile-custom)# vcpu 2
Device(config-app-resource-profile-custom)# end
```

Native docker support

Native Docker Support enables users to deploy the docker applications on the IR1800. The application lifecycle process is similar to the procedure in the Installing and Uninstalling Apps section. For docker applications, entry point configuration is required as part of the application configuration. Please refer to the following example for the entry point configuration.

```
Router#conf t
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#app-hosting appid app3
Router(config-app-hosting)#app-vnic gateway0 virtualportgroup 0 guest-interface 0
Router(config-app-hosting-gateway0)#guest-ipaddress 192.168.0.7 netmask 255.255.255.0
Router(config-app-hosting-gateway0)#app-default-gateway 192.168.0.1 guest-interface 0
Router(config-app-hosting)#app-resource docker
Router(config-app-hosting-docker)#run-opts 1 "--entrypoint '/bin/sleep 10000'"
Router(config-app-hosting-docker)#end
Router#
```

The output for docker applications is shown in the following example:

```
Router#show app-hosting detail
App id : app1
Owner : iox
State : RUNNING
Application
Type : docker
Name : aarch64/busybox
Version : latest
Description :
Path : bootflash:busybox.tar
Activated profile name : custom
Resource reservation
Memory : 431 MB
Disk : 10 MB
CPU : 577 units
VCPU : 1
Attached devices
Type Name Alias
---------------------------------------------
serial/shell iox_console_shell serial0
serial/aux iox_console_aux serial1
```

```
serial/syslog iox_syslog serial2
serial/trace iox_trace serial3
Network interfaces
---------------------------------------
eth0:
MAC address : 52:54:dd:e9:ab:7a
IPv4 address : 192.168.0.7
Network name : VPG0
Docker
------
Run-time information
Command :
Entry-point : /bin/sleep 10000
Run options in use : --entrypoint '/bin/sleep 10000'
Application health information
Status : 0
Last probe error :
Last probe output :
Router#
```

Digital IO for IOx container applications

IOx container applications are able to access the digital IO. There is a CLI for alarm contact command.

```
Router(config)# alarm contact ?
  <0-4>          Alarm contact number (0: Alarm port, 1-4: Digital I/O)
  attach-to-iox  Enable Digital IO Ports access from IOX

Router (config)# alarm contact attach-to-iox
```

Enabling the **attach-to-iox** command will provide complete control of all Digital IO ports to IOx. The ports will be exposed as four character devices /dev/dio-[1-4] to IOX applications. You can use read/write functions to get/set values of the Digital IO ports.

If you wish to update the mode, you can write the mode value to the character device file. This is accomplished by IOCTL calls to read/write the state, change mode, and read the true analog voltage of the port. Following this method, you can attach analog sensors to the IR1800. All ports are initially set to Input mode with voltage pulled up to 3.3v.

The following are examples of IOCTL calls:

Read Digital IO Port:

```
cat /dev/dio-1
```

Write to Digital IO Port:

```
echo 0 > /dev/dio-1
echo 1 > /dev/dio-1
```

Change mode:

```
echo out > /dev/dio-1
echo in > /dev/dio-1
```

List of IOCTLs supported:

```
DIO_GET_STATE = 0x1001
DIO_SET_STATE = 0x1002
DIO_GET_MODE = 0x1003
DIO_SET_MODE_OUTPUT = 0x1004
DIO_SET_MODE_INPUT = 0x1005
```

```
DIO_GET_THRESHOLD 0x1006
DIO_SET_THRESHOLD = 0x1007
DIO_GET_VOLTAGE = 0x1009
```

Read State using IOCTL:

```
import fcntl, array
file = open("/dev/dio-1","rw")
state = array.array('L',[0])
fcntl.ioctl(file, DIO_GET_STATE, state)
print(state[0])
```

Change mode using IOCTL:

```
import fcntl
file = open("/dev/dio-1","rw")
fcntl.ioctl(file, DIO_SET_MODE_OUTPUT, 0)
```

Signed Application Support

Cisco Signed applications are now supported on the IR1800. In order to install a signed application, signed verification has to be enabled on the device. Signed verification can be enabled by following the following instructions.

```
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#
Router(config)#app-hosting signed-verification
Router(config)#
Router(config)#exit
```

After enabling the signed verification, follow the instructions in the Installing and Uninstalling Apps section under IOx Application Hosting in order to install the application.