Application Hosting Overview

In today's networking environment, there is a need for simplifying and automating network management processes. Application hosting gives administrators a platform for leveraging their own tools and utilities for network management. Cisco IOS XR supports third-party, off-the-shelf applications that are built using Linux tool chains. With the software development kit that Cisco provides, users can cross-compile and run custom applications.

When you manage network devices with applications, you are freed of the task of focusing only on the CLI based configurations. Because of the abstraction provided by the applications, while the applications do their job, you can now focus on design and implementation aspects of the network.

The purpose of this chapter is to develop an understanding of the application hosting infrastructure, and the wide range of use cases that may be right for your need.

Docker Container Application Hosting

You can create your own container on IOS XR, and host applications within the container. The applications can be developed using any Linux distribution. Docker container application hosting is suited for applications that use system libraries that are different from those libraries provided by the IOS XR root file system.

In docker container application hosting, you can manage the amount of resources (memory and CPU) consumed by the hosted applications.

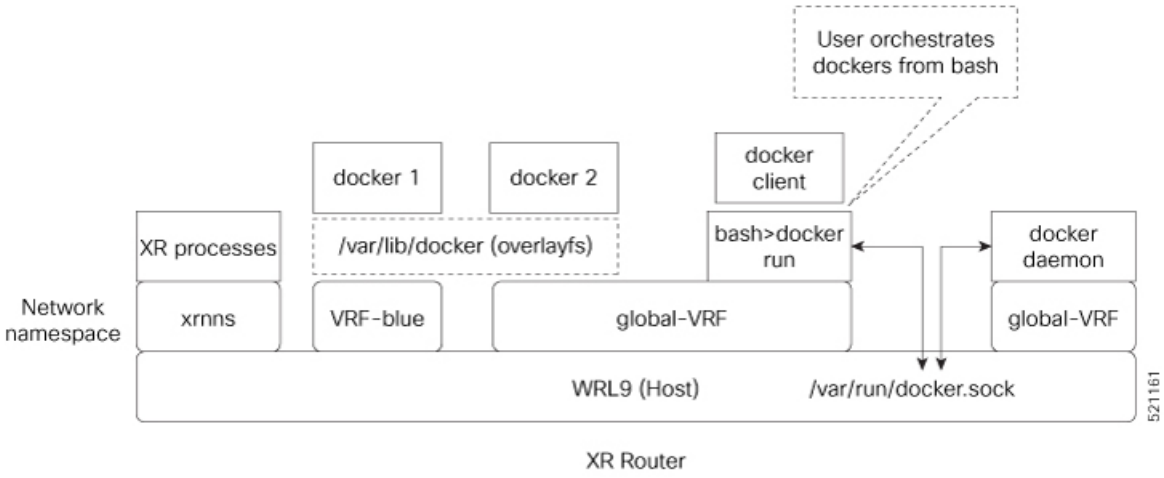
Restrictions

MPLS packets are not supported on Linux interfaces.

Docker Container Application Hosting Architecture

This section describes the docker container application hosting architecture.

Figure 1: Docker on IOS XR



The **docker client**, run from the bash shell, interacts with dockers (docker 1 and docker 2) by using the docker commands. The docker client sends the docker commands to **docker daemon**, which, then, executes the commands. The docker daemon uses the **docker.sock** Unix socket to communicate with the dockers.

When the **docker run** command is executed, a docker container is created and started from the docker image. Docker containers can be either in **global-vrf** namespace or any other defined namespace (for example, VRF-blue).

The docker utilizes overlayfs under the **/var/lib/docker** folder for managing the directories.

To host an application in docker containers, see Hosting an Application in Docker Containers.

App Hosting Components on IOS XR

The following are the components of App Hosting:

- Docker on IOS XR: The Docker daemon is included with the IOS XR software on the base Linux OS. This inclusion provides native support for running applications inside Docker containers on IOS XR. Docker is the preferred method for running TPAs on IOS XR.
- Appmgr: While the Docker daemon comes packaged with IOS XR, Docker applications can only be managed using appmgr. Appmgr allows users to install applications packaged as RPMs and then manage their lifecycle using the IOS XR CLI and programmable models.
- PacketIO: This is the router infrastructure that implements the packet path between TPAs and IOS XR running on the same router. It enables TPAs to leverage XR forwarding for sending and receiving traffic.

TPA Security

IOS XR is equipped with inherent safeguards to prevent third party applications from interfering with its role as a Network OS.

- Although IOS XR doesn't impose a limit on the number of TPAs that can run concurrently, it does impose constraints on the resources allocated to the Docker daemon, based on the following parameters:
  - CPU: By default, ¼ of the CPU per core available in the platform.

Starting from IOS XR Release 24.4.1, you can hard limit the default CPU usage in the range between 25-75% of the total system CPU using the **appmgr resources containers limit cpu** *value* command. This configuration restricts the TPAs from using more CPU than the set hard limit value irrespective of the CPU usage by other XR processes.

This example provides the CPU hard limit configuration.

```
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit cpu ?
  <25-75>  In Percentage
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit cpu 25
```

- RAM: By default, 1 GB of memory is available.

  Starting from IOS XR Release 24.4.1, you can hard limit the default memory usage in the range between 1-25% of the overall system memory using the **appmgr resources containers limit memory** *value* command. This configuration restricts the TPAs from using more memory than the set hard limit value.

  This example provides the memory hard limit configuration.

```
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit memory ?
  <1-25>  In Percentage
RP/0/RSP0/CPU0:ios(config)#appmgr resources containers limit memory 20
```

- Disk space is restricted by the partition size, which varies by platform and can be checked by executing "run df -h" and examining the size of the /misc/app_host or /var/lib/docker mounts.

- All traffic to and from the application is monitored by the XR control protection, LPTS.
- Signed Applications are supported on IOS XR. Users have the option to sign their own applications by onboarding an Owner Certificate (OC) through Ownership Voucher-based workflows as described in RFC 8366. Once an Owner Certificate is onboarded, users can sign applications with GPG keys based on the Owner Certificate, which can then be authenticated during the application installation process on the router.

The table below shows the various functions performed by appmgr.

Package Manager	Lifecyle Manager	Monitoring and Debugging

• Handles installation of docker images packaged as RPMs.
• Syncs the required state to standby to restart apps in cases of switchover, etc
	• Handles application start/stop/kill operations.
	• Handles automatic application reload on:
	  • Router reboot
	  • Container crash
	  • Switchover
		• Logging, stats, application health check.
		• Forwards docker deamon logs to XR syslog.
		• Allows to execute into docker shell of running application.

Customize Docker Run Options Using Application Manager

Table 1: Feature History Table

Feature Name	Release Information	Description
Customize Docker Run Options Using Application Manager	Release 24.1.1	You can now leverage Application Manager to efficiently overwrite default docker runtime configurations, tailoring them to specific parameters like CPU usage, security settings, and health checks. You can thus optimize application performance, maintain fair resource allocation among multiple dockers, and establish non-default network security settings to meet specific security requirements. Additionally, you can accurately monitor and reflect the health of individual applications. This feature modifies the **docker-run-opts** option command.

With this feature, runtime options for docker containerized applications on IOS-XR can be configured during launch using the **appmgr activate"** command. AppMgr, which oversees docker containerized applications, ensures that these runtime options can effectively override default configurations, covering aspects like CPU, security, and health checks during the container launch.

This feature introduces multiple runtime options that allow users to customize different parameters of docker containers. The configuration of these runtime options is flexible, as users can use either command or Netconf for the configuration process. Regardless of the chosen method, runtime options must be added to **docker-run-opts** as needed.

The following are the docker run option commands introduced in IOS-XR software release 24.1.1.

Table 2: Docker Run Options

Docker Run Option	Description
--cpus	Number of CPUs
--cpuset-cpus	CPUs in which to allow execution (0-3, 0,1)
--cap-drop	Drop Linux capabilities
--user, -u	Sets the username or UID
--group-add	Add additional groups to run
--health-cmd	Run to check health
--health-interval	Time between running the check

Docker Run Option	Description
--health-retries	Consecutive failures needed to report unhealthy
--health-start-period	Start period for the container to initialize before starting health-retries countdown
--health-timeout	Maximum time to allow one check to run
--no-healthcheck	Disable any container-specified HEALTHCHECK
--add-host	Add a custom host-to-IP mapping (host:ip)
--dns	Set custom DNS servers
--dns-opt	Set DNS options
--dns-search	Set custom DNS search domains
--domainname	Container NIS domain name
--oom-score-adj	Tune host's OOM preferences (-1000 to 1000)
--shm-size	Option to set the size of /dev/shm
--init	Run an init inside the container that forwards signals and reaps processes
--label, -l	Set meta data on a container
--label-file	Read in a line delimited file of labels
--pids-limit	Tune container pids limit (set -1 for unlimited)
--work-dir	Working directory inside the container
--ulimit	Ulimit options
--read-only	Mount the container's root filesystem as read only
--volumes-from	Mount volumes from the specified container(s)
--stop-signal	Signal to stop the container
--stop-timeout	Timeout (in seconds) to stop a container
--cap-addNET_RAW	Enable NET_RAW capabilities

Prior to IOS-XR software release 24.1.1, only the below mentioned docker run option commands were supported.

Table 3: Docker Run Options

Docker Run Option	Description
--publish	Publish a container's port(s) to the host

Docker Run Option	Description
--entrypoint	Overwrite the default ENTRYPOINT of the image
--expose	Expose a port or a range of ports
--link	Add link to another container
--env	Set environment variables
--env-file	Read in a file of environment variables
--network	Connect a container to a network
--hostname	Container host name
--interactive	Keep STDIN open even if not attached
--tty	Allocate a pseudo-TTY
--publish-all	Publish all exposed ports to random ports
--volume	Bind mount a volume
--mount	Attach a filesystem mount to the container
--restart	Restart policy to apply when a container exits
--cap-add	Add Linux capabilities
--log-driver	Logging driver for the container
--log-opt	Log driver options
--detach	Run container in background and print container ID
--memory	Memory limit
--memory-reservation	Memory soft limit
--cpu-shares	CPU shares (relative weight)
--sysctl	Sysctl options

Guidelines and Limitations

- For the options `--mount` and `--volume`, only the following values can be configured:
  - "/var/run/netns"
  - "/var/lib/docker"
  - "/misc/disk1"
  - "/disk0"
- The maximum allowed size for shm-size option is 64 Mb.

- Prior to Release 24.4.1, all container logs were recorded with an info severity level (sev-6), regardless of the docker run time options used. From Release 24.4.1, if the docker run time option `-it` is used, the container logs are generated with an info severity level (sev-6). However, if the `--it` option is not included, the logs are produced with an error severity level (sev-3).
- From Release 24.4.1, you can use the rsyslog daemon to forward syslog messages to remote syslog servers. To know more, see Support for logging functionality on third-party applications.

Configuration

This section provides the information on how to configure the docker run time options.

In this example we configure the docker run time option **--pids-limit** to limit the number of process IDs using appmgr.

```
Router#appmgr application alpine_app activate type docker source alpine docker-run-opts
"-it –pids-limit 90" docker-run-cmd "sh"
Router#
```

In this example we configure the docker run time option **--pids-limit** to limit the number of process IDs using Netconf.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <appmgr xmlns=http://cisco.com/ns/yang/Cisco-IOS-XR-um-appmgr-cfg>

        <applications>
          <application>
            <application-name>alpine_app</application-name>
            <activate>
              <type>docker</type>
                                              <source-name>alpine</source-name>
                                              <docker-run-cmd>/bin/sh</docker-run-cmd>
                                              <docker-run-opts>-it
--pids-limit=90</docker-run-opts>
            </activate>
          </application>
        </applications>
      </appmgr>
    </config>
  </edit-config>
```

Verification

This example shows how to verify the docker run time option configuration.

```
Router# show running-config appmgr
Thu Mar 23 08:22:47.014 UTC
appmgr
 application alpine_app
  activate type docker source alpine docker-run-opts "-it –pids-limit 90" docker-run-cmd
"sh"
 !
!
```

You can also use **docker inspect** *container id* to verify the docker run time option configuration.

```
Router# docker inspect 25f3c30eb424
[
    {
                "PidsLimit": 90,
    }
]
```

Prioritize Traffic for TPAs in Sandbox Environments

Table 4: Feature History Table

Feature Name	Release Information	Description
Prioritize Traffic for TPAs in Sandbox Environments	Release 24.1.1	You can now optimize network performance, implement traffic segregation, and prevent packet drops due to congestion for Third Party Application (TPA) within the Sandbox environment, improving reliability and efficiency. This is achieved through enhanced LPTS-based traffic prioritization for TPAs hosted within a sandbox container. This feature introduces these changes: **CLI**: • **sandbox flow TPA-APPMGR-HIGH ports** • **sandbox flow TPA-APPMGR-MEDIUM ports** • **sandbox flow TPA-APPMGR-LOW ports**

With this enhancement, you have the flexibility to categorize traffic flows from TPAs hosted in a sandbox based on priority levels, offering better granular control over traffic handling. Prior to this release, traffic from TPAs hosted in a sandbox flowed through a single queue, leading to policer overload and subsequent packet drop.

Configuring Traffic Prioritization for TPA in a Sandbox

During the configuration of a TPA port, you can now set the priority for the port as High, Medium, or Low.

Configuring high priority traffic port

This example shows how to configure TPA traffic in port 2018 to high LPTS flow priority.

```
Router(config)# sandbox flow TPA-APPMGR-HIGH ports 2018
```

Configuring medium priority traffic port

This example shows how to configure TPA traffic in port 6666 to medium LPTS flow priority.

```
Router(config)# sandbox flow TPA-APPMGR-MEDIUM ports 6666
```

Configuring low priority traffic port

This example shows how to configure TPA traffic in port 60100 to low LPTS flow priority.

```
Router(config)# sandbox flow TPA-APPMGR-LOW ports 60100
```

Verification

This example shows how to verify TPA traffic prioritization.

```
Router(config)# show lpts pifib hardware police location

TPA-APPMGR-HIGH         103     np      NPU             1940      1000      0             0
            0
TPA-APPMGR-HIGH         103     np      NPU             1940      1000      1456          0
            1
TPA-APPMGR-MED          104     np      NPU             1940      1000      0             0
            0
TPA-APPMGR-MED          104     np      NPU             1940      1000      1455          0
            1
TPA-APPMGR-LOW          105     np      NPU             1940      1000      0             0
            0
TPA-APPMGR-LOW          105     np      NPU             1940      1000      1456          0
            1
```

Docker Application Management using IPv6 Address

Table 5: Feature History Table

Feature Name	Release Information	Description
Docker Application Management using IPv6 Address	Release 24.4.1	Introduced in this release on: Fixed Systems(8200, 8700);Modular Systems (8800 [LC ASIC: P100]) (select variants only*) *This feature is now supported on: • 8212-32FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E

Feature Name	Release Information	Description
Docker Application Management using IPv6 Address	Release 7.11.1	In this release, you gain the ability to manage Docker applications within containers using IPv6 addresses via the router's management interface. Leveraging IPv6 addresses provides expanded addressing options, enhances network scalability, and enables better segmentation and isolation of applications within the network. Prior to this update, only IPv4 addresses could be used to manage docker applications.

The Application Manager in IOS-XR software release 7.3.15 introduces support for an application networking feature that facilitates traffic forwarding across Virtual Routing and Forwarding (VRF) instances. This feature is implemented through the deployment of a relay agent contained within an independent docker container.

The relay agent acts as a bridge, connecting two network namespaces within the host system and actively transferring traffic between them. Configurations can be made to establish forwarding between either a single pair of ports or multiple pairs, based on your network requirements.

One of the main uses of this feature is to allow the management of Linux-based Docker applications that are running in the default VRF through a management interface. This management interface can be located in a separate VRF. This feature ensures that Docker applications can be managed seamlessly across different VRFs.

In the IOS-XR software release 7.11.1, enhanced management capabilities are offered for docker applications. Now, you can leverage IPv6 addresses to manage applications within docker containers via the management interface of the Cisco 8000 router. This update provides improved accessibility and control over your Docker applications using IPv6 addressing. Prior to the IOS-XR software release 7.11.1, application management for docker containers could only be conducted using IPv4 addresses.

Restrictions and Limitations

In configuring your setup, please consider the following restrictions and limitations:

- **VRF Forwarding Limitation**: The Virtual Routing and Forwarding (VRF) is only supported for Docker apps with host networking.
- **Relay Agent Availability and Management**: The relay agent container is designed to be highly available. It will be managed by the Application Manager (App Mgr).
- **Relay Agent Creation**: For each pair of forwarded ports, one relay agent container will be created.
- **Port Limitation per Application**: The total effective number of ports for each application is limited to a maximum of 10.

Configure VRF Forwarding

To manage a Docker application using the Application Manager through the Management Interface, follow these steps:

Procedure

Step 1 **Configure the app manager**: The application manager is configured to access the docker application. Use the **appmgr application***application-name* keyword to enable and specify configuration parameters for the VRF forwarding. A typical example would look like this:

Example:

```
Router#appmgr
Router#application Testapp
```

Note
The VRF forwarding related run options like **--vrf-forward** and **--vrf-forward-ip-range** will not be passed to the Docker engine when the app container is run.

Step 2 **Enable Basic Forwarding Between Two Ports**: To enable traffic forwarding between two ports in different VRFs, use the following configuration:

Example:

```
Router#activate type docker source swanagent docker-run-opts "--vrf-forward vrf-mgmt:5001
vrf-default:8001 --net=host -it"
```

This command enables traffic on port 5000 at all addresses in vrf-mgmt to be forwarded to the destination veth device in vrf-default on port 8000.

To enable VRF forwarding between multiple ports, follow the steps below:

- **Enable Forwarding Between a Range of Ports**: To enable traffic forwarding between port ranges in different VRFs, use the following configuration:

  ```
  Router#--vrf-forward vrf-mgmt:5000-5002 vrf-default:8000-8002
  ```

  This command enables traffic on ports 5000, 5001, and 5002 at all addresses in vrf-mgmt to be forwarded to the destination veth device in vrf-default on ports 8000, 8001, and 8002 respectively.

- **Enable Forwarding Between Multiple VRF Pairs or Port Ranges**: To enable traffic forwarding between multiple VRF pairs, use multiple **--vrf-forward** command.

  ```
  Router#--vrf-forward vrf-mgmt:5000 vrf-default:8000 --vrf-forward vrf-mgmt:5003-5004
  vrf-default:8003-8004
  Router#--vrf-forward vrf-mgmt1:5000 vrf-default:8000 --vrf-forward vrf-mgmt2:5000 vrf-default:8001
  ```

  You can provide any number of --vrf-forward options, but the total number of port pairs involved should not exceed 10.

Verifying VRF Forwarding for Application Manager

Use the **show appmgr application name** keyword to verify the VRF forwarding. A typical example would look like this:

```
RP/0/RP0/CPU0:ios#show appmgr application name swan info detail
Thu Oct 26 11:59:32.798 UTC
Application: swan
  Type: Docker
```

```
  Source: swanagent
  Config State: Activated
  Docker Information:
    Container ID: f230a2396b85f6b3eeb01a8a4450a47e5bd8499fe5cfdb141c2d0fba905b63ec
    Container name: swan
    Labels:
[illegible]

    Image: swancr.azurecr.io/swanagentxr-iosxr:2.4.0-0ebd435
    Command: "./agentxr"
    Created at: 2023-10-26 11:58:45 +0000 UTC
    Running for: 48 seconds ago
    Status: Up 47 seconds
    Size: 0B (virtual 29.3MB)
    Ports:
    Mounts:
/var/lib/docker/appmgr/config/swanagent/hostname,/var/lib/docker/appmgr/config/swanagent,/var/lib/docker/ems/grpc.sock,/var/run/netns

    Networks: host
    LocalVolumes: 0
    Vrf Relays:
      Vrf Relay: vrf_relay.swan.6a98f0ed060bffa
        Source VRF: vrf-management
        Source Port: 11111
        Destination VRF: vrf-default
        Destination Port: 10000
        IP Address Range: 172.16.0.0/12
        Status: Up 45 seconds
```

Use the **show running-config appmgr** keyword to check the running configuration.

```
Router#show running-config appmgr
Thu Oct 26 12:04:06.063 UTC
appmgr
 application swan
  activate type docker source swanagent docker-run-opts "--vrf-forward vrf-management:11111
 vrf-default:10000 -it --restart always --cap-add=SYS_ADMIN --net=host --log-opt max-size=20m
 --log-opt max-file=3 -e HOSTNAME=$HOSTNAME -v /var/run/netns:/var/run/netns -v
{app_install_root}/config/swanagent:/root/config -v
{app_install_root}/config/swanagent/hostname:/etc/hostname -v
/var/lib/docker/ems/grpc.sock:/root/grpc.sock"
 !
!
```

Hosting Third Party Applications in Sandbox Container Using Sandbox Manager

Table 6: Feature History Table

Feature Name	Release Information	Feature Description
Hosting Third Party Applications in Sandbox Container Using Sandbox Manager	7.5.3	This release introduces Sandbox Manager for hosting and functioning third-party client application in the CentOS 8 based Sandbox Container. The Sandbox container supports configuration, deployment, and management of third-party client applications from the third-party server. The Sandbox manger uses IOS XR commands for managing the Sandbox container.

The Sandbox container enables you to configure, deploy, and manage third-party client applications through the respective third-party server over a network. The Sandbox manager activates the Sandbox container using the APPMGR client library APIs. During the router bootup, the third-party client applications are placed in the Sandbox container using ZTP and get activated when the sandbox manger is enabled. The third-party client applications can then connect to the respective server for installing or upgrading applications in the Sandbox container. Sandbox container operates on CentOS 8, this enables you to control the applications in the container using the docker commands. All the activated third-party client applications can restart automatically after a router reload or an RP switchover.

Supported Commands on Sandbox Manager

This section describes the operations and the IOS XR commands that are supported on the sandbox manager:

- **Enable and disable sandbox manager**: This command is used to enable or disable sandbox manager:
  - Enable—

    The following command enables the Sandbox Manager:

    ```
    RP/0/RP0/CPU0:ios#conf
    RP/0/RP0/CPU0:ios(config)#sandbox enable
    RP/0/RP0/CPU0:ios(config)#commit
    ```

  - Disable—

    The following command disables the Sandbox Manager:

    ```
    RP/0/RP0/CPU0:ios#conf
    RP/0/RP0/CPU0:ios(config)# no sandbox enable
    RP/0/RP0/CPU0:ios(config)#commit
    ```

- **TPA traffic flow prioritization**: These commands are used to configure traffic priority for third party applications within a Sandbox container:
  - High priority traffic—

    The following command configures TPA traffic in port 2018 to high LPTS flow priority

    ```
    Router(config)# sandbox flow TPA-APPMGR-HIGH ports 2018
    ```

- Medium priority traffic—

  The following command configures TPA traffic in port 6666 to medium LPTS flow priority

  ```
  Router(config)# sandbox flow TPA-APPMGR-MEDIUM ports 6666
  ```

- Low priority traffic—

  The following command configures TPA traffic in port 60100 to low LPTS flow priority

  ```
  Router(config)# sandbox flow TPA-APPMGR-LOW ports 60100
  ```

- **Show commands**
  - Info—

    The following command shows the Sandbox Manager and application info:

    ```
    RP/0/RP0/CPU0:ios#show sandbox info
    Thu Jun 30 06:56:45.593 UTC

    Sandbox Config State: Enabled

    APP INFO:
      Image: /pkg/opt/cisco/XR/appmgr/images/sandbox-centos.tar.gz
      Config state: Activated
      Container state: Running
    ```

  - Detail—

    The following command shows the Sandbox Manager and application details:

    ```
    RP/0/RP0/CPU0:ios#show sandbox detail
    Thu Jun 30 06:57:46.724 UTC

    Sandbox Config State: Enabled

    APP INFO:
      Image: /pkg/opt/cisco/XR/appmgr/images/sandbox-centos.tar.gz
      Run Options:
     --restart always
     --cap-add SYS_ADMIN --cap-add NET_ADMIN
     --log-opt max-size=10m --log-opt max-file=3
     --net host
     --mount type=bind,source=/sys/fs/cgroup,target=/sys/fs/cgroup,readonly
     --mount type=bind,source=/var/run/netns,target=/netns,bind-propagation=shared
     --mount type=bind,source=/opt/sandbox,target=/opt/sandbox,bind-propagation=shared

     --mount type=bind,source=/misc/disk1/sandbox,target=/host,bind-propagation=shared

      Config state: Activated
      Container state: Running

    STATS INFO:
      Cpu Percentage: 0.01%
      Memory Usage: 13.57MiB / 19.42GiB
      Net IO: 0B / 0B
      Block IO: 0B / 1.2MB
      Memory Percentage: 0.07%
      pids: 2
    ```

  - Services—

    The following command shows the Sandbox Manager and application services:

```
RP/0/RP0/CPU0:ios#show sandbox services
Wed Jul  6 05:59:16.446 UTC
UNIT                          LOAD   ACTIVE SUB       DESCRIPTION
-.mount                       loaded active mounted   /
dev-mqueue.mount              loaded active mounted   POSIX Message Queue File Sys
etc-hostname.mount            loaded active mounted   /etc/hostname
etc-hosts.mount               loaded active mounted   /etc/hosts
etc-resolv.conf.mount         loaded active mounted   /etc/resolv.conf
host.mount                    loaded active mounted   /host
netns-default.mount           loaded active mounted   /netns/default
netns-global\x2dvrf.mount     loaded active mounted   /netns/global-vrf
netns-vrf\x2dblue.mount       loaded active mounted   /netns/vrf-blue
netns-vrf\x2ddefault.mount    loaded active mounted   /netns/vrf-default
netns-vrf\x2dmanagement.mount loaded active mounted   /netns/vrf-management
netns-vrf\x2dred.mount        loaded active mounted   /netns/vrf-red
netns-xrnns.mount             loaded active mounted   /netns/xrnns
netns.mount                   loaded active mounted   /netns
proc-acpi.mount               loaded active mounted   /proc/acpi
proc-bus.mount                loaded active mounted   /proc/bus
proc-fs.mount                 loaded active mounted   /proc/fs
proc-irq.mount                loaded active mounted   /proc/irq
proc-kcore.mount              loaded active mounted   /proc/kcore
proc-keys.mount               loaded active mounted   /proc/keys
proc-latency_stats.mount      loaded active mounted   /proc/latency_stats
proc-sched_debug.mount        loaded active mounted   /proc/sched_debug
proc-scsi.mount               loaded active mounted   /proc/scsi
proc-sysrq\x2dtrigger.mount loaded active mounted   /proc/sysrq-trigger
proc-timer_list.mount         loaded active mounted   /proc/timer_list
sys-firmware.mount            loaded active mounted   /sys/firmware
systemd-journald.service      loaded active running   Journal Service
systemd-tmpfiles-setup.service loaded active exited    Create Volatile Files and
 Di
-.slice                       loaded active active    Root Slice
system.slice                  loaded active active    System Slice
dbus.socket                   loaded active listening D-Bus System Message Bus Soc
systemd-journald.socket       loaded active running   Journal Socket
systemd-shutdownd.socket      loaded active listening Delayed Shutdown Socket
basic.target                  loaded active active    Basic System
local-fs.target               loaded active active    Local File Systems
multi-user.target             loaded active active    Multi-User System
paths.target                  loaded active active    Paths
slices.target                 loaded active active    Slices
sockets.target                loaded active active    Sockets
swap.target                   loaded active active    Swap
sysinit.target                loaded active active    System Initialization
timers.target                 loaded active active    Timers
systemd-tmpfiles-clean.timer loaded active waiting   Daily Cleanup of Temporary
D

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

43 loaded units listed. Pass --all to see loaded but inactive units, to
o.
To show all installed unit files use 'systemctl list-unit-files'.
```

- Access Sandbox—

  The following command is used to access sandbox container:

```
RP/0/RP0/CPU0:ios#bash sandbox
root@ios:/data# exit
```

```
exit
RP/0/RP0/CPU0:ios#
```

- Linux commands—

  The following command is used to run linux commands inside sandbox container:

```
RP/0/RP0/CPU0:ios#bash sandbox -c linux-command
RP/0/RP0/CPU0:ios#
```

Top Use Cases for Application Hosting

Some of the top use cases for application hosting are:

- **Measure Network Performance**: An application can be hosted to measure the bandwidth, throughput and latency of the network and monitor the performance. An example of such an application is the iPerf tool.
- **Automate Server Management**: An application can be hosted to automate the server functions like upgrading software, allocation of resources, creating user accounts, and so on. Examples of such an application are the Chef and Puppet configuration management tools.

Automated Deployment of Third Party Python Scripts

Feature Name	Release Information	Description
Automated Deployment of Third Party Python Scripts	Release 24.2.1	When you deploy custom or third-party Python scripts on routers running IOS XR software using third-party RPMs, these scripts are automatically executed. This streamlines the deployment process and enhances the speed of script execution. Traditionally, script deployment required an external controller, which used interfaces like NETCONF, SNMP, and SSH to communicate with the router. This feature eliminates the need for such external controllers, simplifying the workflow and improving efficiency.

Efficient network automation is pivotal in handling extensive cloud-computing networks. The Cisco IOS XR infrastructure plays a crucial role by enabling automation through the initiation of API calls and execution of scripts. Traditionally, an external controller is used for this purpose, utilizing interfaces like NETCONF, SNMP, and SSH to communicate with the router.

This feature streamlines the operational structure by executing automation scripts directly on the router, thus eliminating the need for an external controller. It allows scripts to leverage Python libraries and access underlying router information. This approach not only accelerates the execution of various types of scripts

but also enhances reliability by removing dependencies on the speed and network reachability of an external controller.

The third party script is automatically executed by the xr_script_scheduler.py script upon the installation of third-party RPMs. No specific configuration is required to run these scripts after installation.

The below steps provide the information on how to deploy and activate third party script:

Procedure

Step 1 Adding and Activating Scheduler Script - Add and activate scheduler script in in-built script repository - Copy the "xr_script_scheduler.py" scheduler script to the In-Built Script Repository, and simultaneously activate it using the following commands:

Example:

```
cp /path/to/xr_script_scheduler.py /opt/cisco/install-iosxr/base/opt/cisco/ops-script-repo/process/
appmgr activate script name xr_script_scheduler.py
Router#
```

Replace "/path/to/xr_script_scheduler.py" with the actual path of the script. This command copies the script to the specified directory and activates it in the XR configuration mode.

This step ensures the script is added to the repository and activated for continuous execution.

Step 2 Verify the Status of Scheduler Script - To confirm the availability of the scheduler script, run the following command on the router.

Example:

```
Router# show script status
Tue Oct 24 18:03:09.220 UTC
=============================================================================================================

 Name                              | Type    | Status            | Last Action | Action Time

-------------------------------------------------------------------------------------------------------------

 show_interfaces_counters_ecn.py | exec    | Ready             | NEW         | Tue Oct 24 07:10:36 2023

 xr_data_collector.py            | exec    | Ready             | NEW         | Tue Oct 24 07:10:36 2023

 xr_script_scheduler.py          | process| Ready             | NEW         | Tue Oct 24 07:10:36 2023

=============================================================================================================
Router#
```

Ensure that the output displays "Ready" for the "xr_script_scheduler.py" script, indicating that the script checksum is verified and it is ready to run. This single step provides a quick verification of the scheduler script's status.

Step 3 Configure appmgr to Automatically Run the Scheduler Script - Activate the scheduler script automatically using the "autorun" option with the following configuration:

Example:

```
Router(config)#appmgr
Router(config-appmgr)#process-script xr_script_scheduler
Router(config-process-script)#executable xr_script_scheduler.py
Router(config-process-script)#autorun
Router(config-process-script)#commit
```

The 'autorun' configuration has been added to enable automatic activation of the process script. If you prefer manual activation/deactivation using cli, the 'autorun' configuration line can be skipped.

Step 4 Verify scheduler script is running - To verify if the scheduler script is running, execute the **show script execution** command. This command will display a list of OPS scripts currently running. If the scheduler script has been correctly configured and activated, the scheduler script execution detail will appear in the output.

Example:

```
Router# show script execution
Tue Oct 24 18:01:56.590 UTC
==========================================================================================================================

 Req. ID   | Name (type)                                | Start                    | Duration   |
Return | Status
--------------------------------------------------------------------------------------------------------------------------

 1698170509| xr_script_scheduler.py (process)          | Tue Oct 24 18:01:49 2023 | 7.68s      | None
   | Started
--------------------------------------------------------------------------------------------------------------------------

 Execution Details:
 ------------------
 Script Name  : xr_script_scheduler.py
 Version      : 7.3.6.14Iv1.0.0
 Log location : /harddisk:/mirror/script-mgmt/logs/xr_script_scheduler.py_process_xr_script_scheduler

 Arguments    :
 Run Options  : Logging level - INFO, Max. Runtime - 0s, Mode - Background
 Events:
 -------
 1.   Event        : New
      Time         : Tue Oct 24 18:01:49 2023
      Time Elapsed : 0.00s Seconds
      Description  : Started by Appmgr
 2.   Event        : Started
      Time         : Tue Oct 24 18:01:49 2023
      Time Elapsed : 0.11s Seconds
      Description  : Script execution started. PID (15985)
==========================================================================================================================
Router#
```

Step 5 Transfer of Third-Party RPM with Debug/Monitoring Scripts - Transfer the third-party RPM containing debug/monitoring scripts onto the router. This RPM includes Python scripts for debugging/monitoring and a run parameters JSON file.

Example:

```
Router# scp user@171.68.251.248:/users/savinaya/rpm-factory/RPMS/x86_64/nms-1.1-24.1.1.x86_64.rpm
/harddisk:

Tue Oct 24 18:02:42.400 UTC
<snip>
Password:
nms-1.1-24.1.1.x86_64.rpm                                          100% 9664   881.5KB/s   00:00

Router#
Router# dir harddisk:/nms-1.1-24.1.1.x86_64.rpm
```

Step 6 Install the third-party RPM - Use the **appmgr package install** CLI command for the installation of the RPM.

Example:

```
Router# appmgr package install rpm /harddisk:/nms-1.1-24.1.1.x86_64.rpm
Tue Oct 24 18:03:26.685 UTC
```

```
Router# show appmgr packages installed
Tue Oct 24 19:42:07.967 UTC
Sno Package
--- ------------------------------------------------------------
1   nms-1.1-24.1.1.x86_64
Router#
```

Step 7 Verify the operation of the debug/monitoring scripts - You can verify that these scripts are functioning by executing the **show script execution** command.

Example:

```
Router# show script execution
Tue Oct 24 19:41:15.882 UTC
=============================================================================================================================

 Req. ID   | Name (type)                                  | Start                    | Duration   |
Return | Status
-----------------------------------------------------------------------------------------------------------------------------

 1698176223| xr_script_scheduler.py (process)          | Tue Oct 24 19:37:02 2023 | 253.32s    | None
   | Started
 1698176224| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:38:43 2023 | 152.46s    | None
   | Started
 1698176225| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:38:44 2023 | 152.03s    | None
   | Started
 1698176226| nms/monitor_int_rx_cntr2.py (exec)        | Tue Oct 24 19:38:44 2023 | 151.63s    | None
   | Started
=============================================================================================================================
Router#
```

Step 8 Stopping the scheduler script - Stop the scheduler using the **appmgr process-script stop** command.

Example:

```
Router# show script execution
Tue Oct 24 20:04:22.021 UTC
=============================================================================================================================

 Req. ID   | Name (type)                                  | Start                    | Duration   |
Return | Status
-----------------------------------------------------------------------------------------------------------------------------

 1698176224| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:38:43 2023 | 234.21s    | -9
     | Stopped
 1698176225| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:38:44 2023 | 234.43s    | -9
     | Stopped
 1698176226| nms/monitor_int_rx_cntr2.py (exec)        | Tue Oct 24 19:38:44 2023 | 234.67s    | -9
     | Stopped
 1698176227| ops/monitor_int_rx_cntr3.py (exec)        | Tue Oct 24 19:41:35 2023 | 97.56s     | -9
     | Stopped
 1698176228| ops/monitor_int_rx_cntr4.py (exec)        | Tue Oct 24 19:41:36 2023 | 97.19s     | -9
     | Stopped
 1698176229| ops/monitor_int_rx_cntr5.py (exec)        | Tue Oct 24 19:41:36 2023 | 96.48s     | -9
     | Stopped
 1698176231| ops/monitor_int_rx_cntr3.py (exec)        | Tue Oct 24 19:43:44 2023 | 760.88s    | -9
     | Stopped
 1698176232| ops/monitor_int_rx_cntr4.py (exec)        | Tue Oct 24 19:43:44 2023 | 760.53s    | -9
     | Stopped
 1698176233| ops/monitor_int_rx_cntr5.py (exec)        | Tue Oct 24 19:43:44 2023 | 760.20s    | -9
     | Stopped
 1698176234| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:44:15 2023 | 202.88s    | -9
     | Stopped
 1698176235| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:44:15 2023 | 203.01s    | -9
     | Stopped
```

```
1698176236| nms/monitor_int_rx_cntr2.py (exec)        | Tue Oct 24 19:44:16 2023 | 203.17s    | -9
    | Stopped
1698176237| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:53:41 2023 | 163.99s    | -9
    | Stopped
1698176238| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 19:53:41 2023 | 163.52s    | -9
    | Stopped
1698176239| nms/monitor_int_rx_cntr2.py (exec)        | Tue Oct 24 19:53:42 2023 | 163.11s    | -9
    | Stopped
1698176252| xr_script_scheduler.py (process)          | Tue Oct 24 20:00:20 2023 | 220.61s    | -15
   | Stopped
1698176253| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 20:00:21 2023 | 222.11s    | -9
    | Stopped
1698176254| nms/monitor_int_rx_cntr.py (exec)         | Tue Oct 24 20:00:21 2023 | 221.76s    | -9
    | Stopped
1698176255| nms/monitor_int_rx_cntr2.py (exec)        | Tue Oct 24 20:00:22 2023 | 221.39s    | -9
    | Stopped
1698176256| ops/monitor_int_rx_cntr3.py (exec)        | Tue Oct 24 20:00:22 2023 | 221.08s    | -9
    | Stopped
1698176257| ops/monitor_int_rx_cntr4.py (exec)        | Tue Oct 24 20:00:23 2023 | 131.46s    | -9
    | Stopped
1698176258| ops/monitor_int_rx_cntr5.py (exec)        | Tue Oct 24 20:00:23 2023 | 220.30s    | -9
    | Stopped
=============================================================================================================
Router#
```