NCS 1004 Application hosting

Application hosting features

Application hosting features are IOS-XR infrastructure capabilities that

- allow third-party applications to run directly on NCS 1004 devices,
- enable extension of device functionality to complement IOS-XR features, and
- provide native support for containerized applications through Docker.

The Docker daemon is packaged with IOS-XR software on the base Linux OS. This provides native support for running applications inside docker containers on IOS-XR. Docker is the preferred way to run TPAs on IOS-XR.

App Manager

The App Manager is the infra on IOS-XR tasked with the responsibility of managing the life cycle of all container apps (third part and Cisco internal) and process scripts. App Manager runs natively on the host as an IOS-XR process. App Manager leverages the functionalities of docker, systemd and RPM for managing the lifecycle of third-party applications.

Caution: Do not use MPLS packets on Linux interfaces in Docker containers

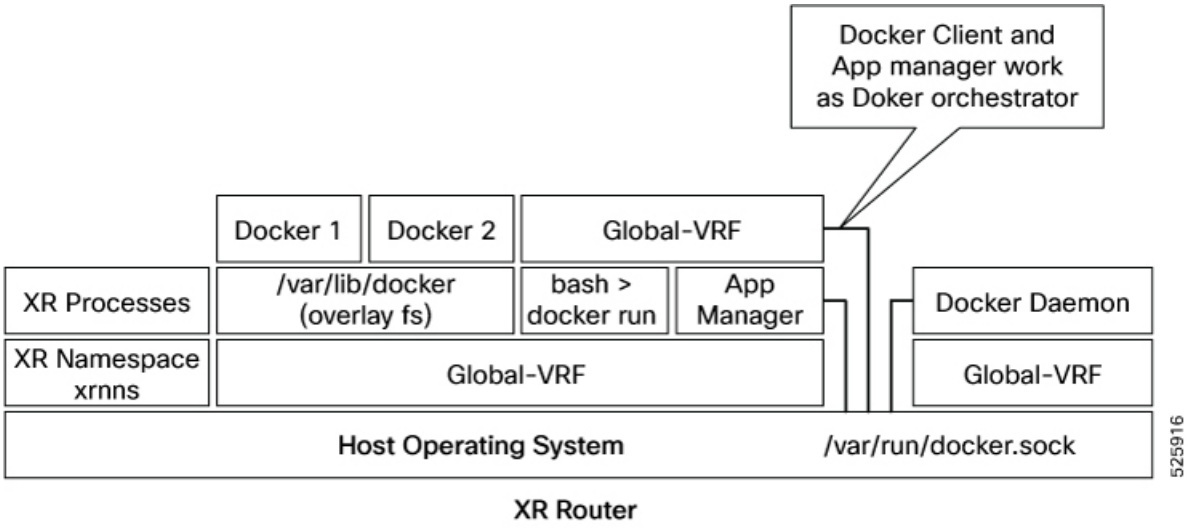
MPLS packets are not supported on Linux interfaces used for application hosting in Docker containers. You must not attempt to configure or use MPLS on these interfaces, as doing so may lead to operational instability or traffic disruption.

Docker container application hosting architectures

A Docker container application hosting architecture is a platform solution that

- utilizes Docker containers to deploy and manage applications,
- enables the App Manager to interact with containers through the Docker client and daemon, and
- facilitates application isolation and resource management across network devices.

Figure 1: Docker on IOS XR



Cisco IOS XR employs Docker to enable application hosting through a structured architecture. The App Manager internally uses the Docker client, which communicates with Trusted Platform Applications (TPAs) such as Docker 1 and Docker 2 by sending Docker commands.

The Docker client transmits commands to the Docker daemon, which then executes them. The Docker daemon uses the `docker.sock` Unix socket to communicate with the Docker containers.

When the `docker run` command is executed, a Docker container is created and started from a Docker image. Docker containers can operate within the `global-vrf` namespace, providing networking isolation.

Docker relies on overlayfs, located under the `/var/lib/docker` directory, for managing container directories and files

For detailed instructions on how to host an application in Docker containers on Cisco IOS XR, see: Hosting an Application in Docker Containers.

Recommendation: Follow hosting guidelines for Docker containers

Use the host paths listed below when specifying the --mount and --volume options with docker run:

- "/var/run/netns"
- "/var/lib/docker"
- "/misc/disk1"
- "/disk0"

- "/misc/config/grpc"
- "/etc"
- "/dev/net/tun"
- "/var/xr/config/grpc"
- "/opt/owner"

TP application resource configurations

A TP application resource configuration is a system resource management feature that

- limits the allocation of CPU, RAM, and disk space to third party applications (TPAs),
- enforces these limits through the IOS XR application manager (appmgr), and
- ensures platform security and operational integrity by monitoring application traffic and supporting application signing.

Third party applications (TPAs): Applications packaged as Docker containers and managed by the IOS XR operating system.

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.

    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.

    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.

This table 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.

TP application bring-up methods

A TP application bring-up is a deployment process that

- allows TP container applications to be launched and initialized for operational readiness,
- provides multiple configuration approaches to match different deployment needs, and
- ensures flexibility in integration with system models and container management tools.

There are four main approaches for bringing up TP applications, each offering distinct deployment and management strategies.

he four recommended methods for TP application bring-up are:

- App Config: Use the application configuration files to define startup parameters.
- UM Model: Employ the Unified Management model for standardized operational controls.
- Native Yang Model: Leverage native YANG data models for direct integration and configuration.
- gNOI Containerz: Use gNOI Containerz for streamlined container orchestration and lifecycle management.

Configure a TPA using application configuration

Activate and configure a TPA by specifying docker runtime settings to control process resources and verify the configuration.

Configuring a TPA with application configuration allows you to manage its runtime options, such as limiting the number of process IDs, ensuring efficient operation and resource management.

Follow these steps to configure the docker run time options.

Procedure

Step 1 Configure the docker run time option.

Use **--pids-limit** to limit the number of process IDs using appmgr.

Example:

This example shows the configuration of the docker run time option **--pids-limit** to limit the number of process IDs using appmgr.

```
RP/0/RP0/CPU0:ios#appmgr application alpine_app activate type docker source alpine docker-run-opts
"-it -pids-limit 90" docker-run-cmd "sh"
```

The number of process IDs is limited to 90.

Step 2 Verify the docker run time option configuration.

Use the **show running-config appmgr** command to verify the run time option.

Example:

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

```
RP/0/RP0/CPU0:ios#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"
 !
!
```

The TPA is activated using the specified docker runtime options, and the configuration is verified to ensure resource limits are correctly applied.

Configure TPAs with the UM model

Limit the number of process IDs (PIDs) available to a TPA (Third Party Application) running in Docker using Netconf.

Use this task to apply resource limits to Docker-based TPAs managed through Cisco's Unified Model (UM). Limiting PIDs prevents runaway processes and supports system reliability.

Follow these steps to configure the docker run time options.

Procedure

Configure the docker run time option.

Use **--pids-limit** to limit the number of process IDs using Netconf.

Example:

This example shows the configuration of 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>
```

The number of process IDs for the specified TPA is limited as configured. For example, with `--pids-limit=90`, the application will not spawn more than 90 processes.

Native model deployment for TPAs

The native deployment model for Third Party Applications (TPAs) offers direct integration and performance benefits for supported applications. This model enables users to configure advanced options, such as limiting process IDs for containerized applications using the Native YANG model.

This configuration uses the docker runtime option `--pids-limit` to restrict the number of process IDs for an application called `alpine_app`. This ensures resource control and operational stability in environments with multiple TPAs.

```
<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-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>
```

Key benefits of using the native deployment model:

- Enables direct integration for TPAs.
- Provides better resource management through advanced container options.

- Improves performance and operational reliability for supported applications.

gNOI Containerz for TPA onboarding and lifecycle management

The gNOI Containerz service on the NCS 1004 device enables the onboarding and management of third-party applications (TPAs) using standardized gNOI remote procedure calls (RPCs). This approach offers:

- A streamlined workflow for onboarding TPAs onto the device.
- Standardized methods for managing the full lifecycle of containers, including deployment, start/stop, and removal.
- Simplified and consistent administration of application containers on the device.

The Containerz - gNOI Container Service on NCS 1004 device is a workflow to onboard and manage third-party applications using gNOI RPCs.

For more information, see gNOI Containerz.

Docker run options

A Docker run option is a container configuration parameter that

- controls resource allocation such as CPU, memory, and networking,
- enables fine-tuning of security, health checks, and process behavior, and
- allows customization during container launch for precise operation.

Docker run options are used in IOS-XR environments through the Application Manager (AppMgr). These options can be configured during the launch of a docker containerized application using the `appmgr activate` command. AppMgr oversees these containers and ensures runtime options can override default configurations for aspects like CPU, security, and health checks. Configuration is flexible: you can use CLI or Netconf, but all runtime options must be added under `docker-run-opts` as needed.

Table 1: 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
--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

Docker run option	Description
--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
--publish	Publish a container's port(s) to the host
--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

Docker run option	Description
--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

Third party RPMs installation using App Manager install UI

A third party RPM is a package file that

- provides additional software features or capabilities not included with the base system,
- enables integration of external applications and tools, and
- extends system functionality through easy installation in App Manager at runtime.

App Manager's install user interface allows you to install third party RPMs during runtime while the router is operational. This feature simplifies the addition of new capabilities to your system. To install a third party RPM, select the desired package in the App Manager install UI and follow the installation steps as prompted.

Limitations for installing third-party RPMs

These limitations apply to installing third-party RPMs with the App Manager:

- RPM must not have “scriptlets”. Scriptlets allow packages to run code on installation and removal.
- RPM must not be already installed via XR Install UI.
- RPM must not be already installed via App manager UI.
- TP RPMs must install files only to the `/var/lib/docker/appmgr/` filesystem location.
- RPM Signature verification is not enforced by App Manager Install UI. It supports unsigned TP RPMs.

Install third party RPMs using App Manager install UI

Enable installation of third-party applications packaged as RPMs onto the device using the App Manager Install UI.

Use this task when you need to deploy custom or third-party Docker container applications on your device through App Manager.

Use this task to install third party RPMs using App Manager install UI.

Procedure

Step 1 Create an RPM containing the application (in the form of a docker container image).

Step 2 Use App manager RPM build tool to generate TP RPMs. See https://github.com/ios-xr/xr-appmgr-build/blob/main/README.md.

Step 3 Install the TP RPM using the App Manager Install UI command.

Example:

```
RP/0/RP0/CPU0:ios# appmgr package install rpm /harddisk\:/alpine-0.1.0-XR_7.3.1.x86_64.rpm
```

The third-party RPM package is successfully installed, making the application available through App Manager.

Uninstall third party RPMs using App Manager install UI

Remove third party RPMs to maintain system cleanliness and ensure optimal performance.

You can uninstall third party RPMs (TP RPMs) from your system using the App Manager install UI. This feature enables you to safely remove applications while NCS 1004 is running. Uninstallation can be performed in two ways: by specifying either the source name or the package name.

Attention App Manager uninstall CLI uninstalls the TP RPM at runtime, ensuring continuous operation of the NCS 1004 device.

Procedure

Step 1 Decide whether you want to uninstall the RPM using the source name or the package name.

Step 2 If uninstalling by source name, enter this command:

Example:

```
RP/0/RP0/CPU0:ios# appmgr package uninstall source alpine
```

Step 3 If uninstalling by package name, enter this command:

Example:

```
RP/0/RP0/CPU0:ios# appmgr package uninstall package alpine-0.1.0-XR_7.3.1.x86_64
```

Step 4 Confirm the removal as prompted by the CLI. The application will be uninstalled at runtime without interrupting other system operations.

Supported commands for the application manager

The application manager supports several action commands to directly manage container applications:

Action commands

App Manager action commands are used to start, stop, kill and exec shell commands inside running container.

Table 2: Action commands

Command name	Commands	Purpose
Application start	**appmgr application start name** <*name*>	Starts a stopped container or application.
Application stop	**appmgr application stop name** <*name*>	Stops a stopped running container or application.
Application kill	**appmgr application kill name** <*name*>	Kills a running container or application.
Application copy	**appmgr application copy** <*storage-path*>	Copy data between host and container.
Appliction exec	**appmgr application exec** <*name*> **docker-exec-cmd** <*cmd*>	Executes command inside TP container application (docker only).

Examples

Here are examples for the app manager action commands. For more information on the commands, see *Command Reference Guide for NCS 1004*.

Action CLI (Start): This starts a stopped container

```
RP/0/RP0/CPU0:ios# appmgr application start name alpine_app
```

Action CLI (Stop): This stops a running container

```
RP/0/RP0/CPU0:ios# appmgr application stop name alpine_app
```

Action CLI (Kill): This forcefully kills a running container

```
RP/0/RP0/CPU0:ios# appmgr application kill name alpine_app
```

Action CLI (Copy): Copy data between host and container

```
RP/0/RP0/CPU0:ios# appmgr application copy harddisk:/data.txt alpine_app:/
```

Action CLI (Exec): Execute command inside TP container app

```
RP/0/RP0/CPU0:ios# appmgr application exec name txt alpine_app docker-exec-cmd “ls -ltr”
```

Show commands

App Manager show commands shows the application or container info.

Table 3: show commands

Command name	Commands	Purpose
Source table modification	**show appmgr source-table**	Lists all third-party applications onboarded via (XR Infra / Appmgr CLI / Containerz).
Application table modification	**show appmgr application-table**	Lists all third-party applications managed via (Config / Containerz) workflow in a tabular view.
Application source name	**show appmgr source name** *<name>*	Shows the source name.
Application package install	**show appmgr packages installed**	Lists all the application manager RPM packages installed.
Appliction exec	**show appmgr application name** *<name>* **info** [**detail** | **summary**]	Shows application information at desired verbosity.
Application logs	**show appmgr application name** *<name>* **logs**	Shows application logs.
Application stats	**show appmgr application name** *name* **stats**	Shows application statistics.
Application process script table	**show appmgr process-script-table**	Shows summary status of all registered process-scripts.

Examples

This section shows the example outputs for the show appmgr commands. For more information on the show appmgr commands, see the Command Reference guide for the NCS 1004.

The example output shows the onboarded TP applications.

```
RP/0/RP0/CPU0:ios# show appmgr source-table
Sno Name               File                                Installed By

--- ----------------- -------------------- ----------------------------------
1   alpine                 alpine.tar.gz               containerz
2   hello-world        hello-world.tar.gz       app_manager
3   bonnet               bonnet.tar.gz              xr_install
```

The example output shows the Workflow column that specifies how to manage the TP applications.

```
RP/0/RP0/CPU0:ios#show appmgr application-table
Name          Type    Config State  Status                  Workflow
------------- ------ ------------ ------------- ---------------------------------
alp-cz-app    Docker  Activated    Up 2 minutes     containerz
bnt-cfg-app  Docker  Activated    Up 1 minutes     config
```

The example output shows the details of the *swan* application. The `Status` value under `Vrf Relay: <name>` indicates the running status of the relay agent. If it reports an `Exited` state or a `Restarting` state, use the relay agent logs for troubleshooting.

```
RP/0/RP0/CPU0:ios#show appmgr application name swan info detail
Mon Nov 23 21:22:47.240 UTC
Application: swan
  Type: Docker
  Source: swanagent
  Config State: Activated
  Docker Information:
    Container ID: cd27988cd5b066d6272085e5e3ff675c94a64cb4ad06f90c2d89453a8ec4af34
    Container name: swan
    Labels:
    Image: swanagent:latest
    Command: "./agentxr"
    Created at: 2020-11-23 21:22:39 +0000 UTC
    Running for: 8 seconds ago
    Status: Up Less than a second
    Size: 0B (virtual 82.9MB)
    Ports:
    Mounts: /var/opt/cisco/iosxr/appmgr/config/docker/swanagent,/var/run/netns
    Networks: host
    LocalVolumes: 0
    Vrf Relays:
      Vrf Relay: vrf_relay.swan.70ec1f59336271ab
        Source VRF: vrf-mgmt
        Source Port: 8000
        Destination VRF: vrf-default
        Destination Port: 10000
        IP Address Range: 172.16.0.0/12
        Status: Up 10 seconds
      Vrf Relay: vrf_relay.swan.5c7373d41d0ec84f
        Source VRF: vrf-mgmt
        Source Port: 8001
        Destination VRF: vrf-default
        Destination Port: 10001
        IP Address Range: 172.16.0.0/12
        Status: Up 11 seconds
```

Application hosting use cases

Application hosting provides versatile solutions in network environments through the following use cases:

- Measuring network performance: Host performance measurement tools (such as iPerf) to assess bandwidth, throughput, and latency, enabling effective network monitoring and troubleshooting.
- Automating server management: Host configuration and management tools (including Chef and Puppet) to automate server functions such as software upgrades, resource allocation, and user account creation.

These examples illustrate how application hosting addresses real-world network challenges and enhances operational efficiency.

How deploying and activating custom scripts on NCS 1004 works

Deploying custom scripts on NCS 1004 enables advanced automation and monitoring for optical devices. The process integrates scheduler scripts managed through App manager and custom scripts provided via third-party RPM packages, allowing flexible and scalable operations.

Summary

The key components involved in the process are:

- App manager: Manages scripts and their lifecycle, providing activation, deactivation, and execution functions.
- Scheduler script: Coordinates execution and automation of tasks based on defined parameters.
- Third-party RPM package: Contains custom debug or monitoring scripts and run parameter files for deployment.

Workflow

These are the stages of deploying and activating custome scrypts

1. Preparation: Administrator ensures access credentials and obtains required scheduler and RPM scripts.
2. Activation of scheduler script: Scheduler script is configured, activated, and started using App manager to establish the automation environment.
3. Deployment of third-party RPM package: RPM files containing custom scripts are transferred to the node, installed, and their run parameters reviewed or customized.
4. Execution and verification: The system verifies that all scripts are installed, activated, and running. Logs are checked to confirm script operations.
5. Management and updates: Scripts can be updated, stopped, or reactivated as operational needs evolve.

Result

NCS 1004 provides a robust, automated framework for running custom scripts, supporting proactive monitoring, troubleshooting, and advanced automation. This process enables operators to efficiently manage and scale node operations through both built-in and custom scripting solutions.

Activate the XR scheduler script via App manager on NCS 1004

Prepare NCS 1004 to use scheduler scripts for automation and monitoring by configuring and activating them via App manager.

NCS 1004 allows automation of node operations using scheduler scripts managed by App manager. Activation and verification of these scripts enable execution of debug and monitoring tasks as needed.

Procedure

Step 1 Use the **show script status** command to check the list of the OPS scripts that are in-built in XR.

Example:

This command lists the status of *xr_script_scheduler* script. *Ready* status in the output means that the script checksum is verified and is ready to run.

```
RP/0/RP0/CPU0:ios#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 2025

 xr_data_collector.py            | exec    | Ready             | NEW         | Tue Oct 24 07:10:36 2025

 xr_script_scheduler.py          | process| Ready             | NEW         | Tue Oct 24 07:10:36 2025
=============================================================================================================
RP/0/RP0/CPU0:ios#
```

Step 2 Use the appmgr to run the XR scheduler script.

XR scheduler script contains the necessary

Example:

```
RP/0/RP0/CPU0:ios#configure
RP/0/RP0/CPU0:ios(config)#appmgr
RP/0/RP0/CPU0:ios(config-appmgr)#process-script xr_script_scheduler
RP/0/RP0/CPU0:ios(config-process-script)#executable xr_script_scheduler.py
RP/0/RP0/CPU0:ios(config-process-script)#commit
```

Step 3 Check for available process scripts in app manager.

Example:

This output highlights the *xr_script_scheduler.py* process script that is not activated.

```
RP/0/RP0/CPU0:ios#show appmgr process-script-table
Wed Oct 22 09:45:02.795 UTC
Name                 Executable            Activated  Status    Restart Policy    Config Pending
-------------------- --------------------- ---------- --------- ----------------- ----------------
xr_script_scheduler  xr_script_scheduler.py     No     Not Started   Always                 No
```

Step 4 Activate the available process script.

You can start executing a process script only after it is activated.

Example:

```
RP/0/RP0/CPU0:ios#appmgr process-script activate name xr_script_scheduler
Wed Oct 22 09:45:41.035 UTC
```

(Optional) Verify the status of the process script. This example shows the process script *xr_script_scheduler* is *Activated*.

```
RP/0/RP0/CPU0:ios#show appmgr process-script-table
Wed Oct 22 09:45:47.275 UTC
Name                 Executable            Activated  Status    Restart Policy    Config Pending
-------------------- --------------------- ---------- --------- ----------------- ----------------
xr_script_scheduler  xr_script_scheduler.py     Yes     Not Started   Always                 No
```

Step 5 Use the **appmgr process-script start** command to start the available process script.

Example:

xr_script_scheduler is the only available process script.

This command starts the process script *xr_script_scheduler*.

```
RP/0/RP0/CPU0:ios#appmgr process-script start name xr_script_scheduler
Wed Oct 22 09:46:08.273 UTC
```

(Optional) Verify the status of the process script after activation. This example shows the process script *xr_script_scheduler* is *Activated* and *Started*.

```
RP/0/RP0/CPU0:ios#show appmgr process-script-table
Wed Oct 22 09:46:24.679 UTC
Name                 Executable            Activated  Status    Restart Policy    Config Pending
-------------------- --------------------- ---------- --------- ----------------- ----------------
xr_script_scheduler  xr_script_scheduler.py       Yes     Started   Always                  No
```

Step 6 Verify the scheduler script is running.

a) Run the **show script execution** command to verify the functioning of the debug and monitoring scripts.

Example:

This command displays a list of OPS scripts currently running.

```
RP/0/RP0/CPU0:ios# 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 2025 | 253.32s    |
 None   | Started
==========================================================================================================================
RP/0/RP0/CPU0:ios#
```

b) Use the **show script execution details** command to verify if the scheduler script is running.

Example:

This command displays a list of OPS scripts currently running. If the scheduler script is correctly configured and activated, the scheduler script execution detail appears in the output.

```
RP/0/RP0/CPU0:ios#show script execution details
Tue Oct 25 18:01:56.590 UTC
==========================================================================================================================

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

 1698170509| xr_script_scheduler.py (process)           | Tue Oct 25 18:01:49 2023 | 7.68s      |
 None   | Started
--------------------------------------------------------------------------------------------------------------------------

 Execution Details:
 ------------------
 Script Name  : xr_script_scheduler.py
 Version      : 25.3.1.14Iv1.0.0
 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 25 18:01:49 2025
```

```
      Time Elapsed : 0.00s Seconds
      Description  : Started by Appmgr
 2.   Event        : Started
      Time         : Tue Oct 25 18:01:49 2025
      Time Elapsed : 0.11s Seconds
      Description  : Script execution started. PID (15985)
==============================================================================
RP/0/RP0/CPU0:ios#
```

The XR scheduler script is configured, activated, and running on NCS 1004, ready to automate node operations and manage additional scripts.

Deploy and manage third-party RPM scripts on NCS 1004

Expand NCS 1004 automation by deploying and managing third-party Python scripts through RPM package installation.

RPM packages can contain custom scripts and run parameter files for use in NCS 1004. Proper deployment and configuration allow for flexible automation, monitoring, and debugging.

Procedure

Step 1 Copy the third party RPM files to the NCS 1004 node.

a) Use any of the file transfer mechanisms to copy third-party RPM.

Example:

This example shows copying the RPM to the harddisk of the NCS 1004 node using **scp**.

```
RP/0/RP0/CPU0:ios#scp
user@171.xx.xxx.xxx:/users/user/rpm-factory/RPMS/x86_64/nms-1.1-25.3.1.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

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

b) (Optional) Verify the RPM files using **dir <filepath>**.

Example:

```
RP/0/RP0/CPU0:ios#dir harddisk:/nms-1.1-24.1.1.x86_64.rpm
Wed Oct  24 19:53:54.041 UTC
```

Step 2 Install the third party RPM files to use the required debug and monitoring python scripts.

The third party RPM files have the customized scripts to be executed. The third-party RPM contains two types of files:

- One or more python scripts—For more information on developing python scripts, see IOS XR Programmability with Python and xr-python-scripts.
- Run parameter JSON file—*xr_script_scheduler.json* has instruction for the scheduler script.

Example:

This is an example *xr_script_scheduler.json* file. Customize this file as per your requirements.

```
[
    {
        "name": "__template_entry__.py",
        "description": ["**This is a template entry for documentation purpose. This entry will be
ignored**",
                        "name : Name of the python script to be executed",
                        "       [string][mandatory]",
                        "description:  Description of the script",
                        "              [string or list of strings][optional: default empty string]",

                        "cmd_line_parameters: Script command line parameters" ,
                        "                     [list of strings][optional: default Null][Example: ",
                        "env_variables: Enviromental variables to be set in script run shell",
                        "               [list of key value pairs][optional: default Null][Example:
[['INT_NAME': 'hu0/1/0/1']]",
                        "run_policy: Script restart policy when script exits",
                        "            [string: one of always/once/stop][optional: default 'always']",

                        "            always: restart the script every time it exits",
                        "            once:  do not restart the script if it exits",
                        "            stop:  stop an existing script run "
                        ],
        "cmd_line_parameters": [],
        "env_variables": [],
        "run_policy": "always"
    },
    {
        "name": "monitor_int_rx_cntr.py",
        "description": "Monitoring mgmt interface for Rx threshold of 100 ",
        "cmd_line_parameters": ["MgmtEth0/RP0/CPU0/0", "200", "-log", "debug"],
        "env_variables": [["INT_NAME", "FourHundredGigE0/9/0/0"], ["INT_NAME2",
"FourHundredGigE0/10/0/0"]],
        "run_policy": "always"
    },
    {
        "name": "monitor_int_rx_cntr.py",
        "description": "Monitoring Fo0/0/0/0 interface for Rx threshold of 1000000 ",
        "cmd_line_parameters": ["FourHundredGigE0/0/0/0", "1000000"],
        "run_policy": "once"
    },
    {
        "name": "monitor_int_rx_cntr2.py",
        "description": "Monitoring Fo0/0/0/1 interface for Rx threshold of 5000000 ",
        "cmd_line_parameters": ["FourHundredGigE0/0/0/1", "5000000"],
        "run_policy": "always"
    },
    {
        "name": "monitor_int_rx_cntr2.py",
        "description": "Monitoring Fo0/0/0/2 interface for Rx threshold of 5000000 ",
        "cmd_line_parameters": ["FourHundredGigE0/0/0/2", "8000000"],
        "run_policy": "stop"
    }
]
```

Example:

Use the **appmgr package install rpm <full RPM file path>** command to install the third-party RPMs.

```
RP/0/RP0/CPU0:ios#appmgr package install rpm /harddisk:/nms-1.1-25.3.1.x86_64.rpm
Tue Oct 24 18:03:26.685 UTC
RP/0/RP0/CPU0:ios#
RP/0/RP0/CPU0:ios#show appmgr packages installed
```

```
Tue Oct 24 19:42:07.967 UTC
Sno Package
--- ------------------------------------------------------------
1   nms-1.1-25.3.1.x86_64
RP/0/RP0/CPU0:ios#
```

After the scripts and the run parameters file become ready, build the RPM and configure the RPM to install files at `<default exr appmgr rpm install path>/ops-script-repo/exec/<rpm name>/`. RPM build tool for TPA is available at RPM Build Tool.

Note
Install the scripts in directories named after the RPM for smoother execution.

Step 3 Use the **show script status** command to verify that the scripts and the run parameter files contained in the RPM are all installed successfully and added to the script management repository.

Example:

This output shows the status that two scripts (monitor_int_xr_cntr.py and monitor_int_rx_cntr2.py) and a run parameter file (xr_script_scheduler.json) file were installed in the third-party RPM named “nms”.

```
RP/0/RP0/CPU0:ios#show script status

Tue Oct 24 19:41:10.696 UTC
=============================================================================================================

 Name                              | Type    | Status           | Last Action | Action Time

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

 nms/monitor_int_rx_cntr.py        | exec    | Ready            | NEW         | Tue Oct 24 19:38:41 2023

 nms/monitor_int_rx_cntr2.py       | exec    | Ready            | NEW         | Tue Oct 24 19:38:41 2023

 nms/xr_script_scheduler.json      | exec    | Ready            | NEW         | Tue Oct 24 19:38:41 2023

 show_interfaces_counters_ecn.py | exec    | Ready            | NEW         | Tue Oct 24 19:33:52 2023

 xr_data_collector.py            | exec    | Ready            | NEW         | Tue Oct 24 19:33:52 2023

 xr_script_scheduler.py          | process| Ready            | NEW         | Tue Oct 24 19:33:52 2023

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

After the scripts are installed, the scheduler script starts reading the run parameter JSON file and executes the required debug and monitoring scripts.

The logs generated by the scripts are available in the directory `/harddisk\:/mirror/script-mgmt/logs/`.

Step 4 Verify that the debug and monitoring scripts are running.

Example:

Use the **show script execution** command to verify that the scripts are running.

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

(Optional) Use the **show script execution** [**name***script-name***detail [output][error]**]

Step 5 Verify all the active packages are installed.

Example:

```
RP/0/RP0/CPU0:ios#show install active summary
Fri Nov 14 12:52:39.322 IST
Label : 25.3.1.31I-iso

    Active Packages: 2
        ncs1004-xr-25.3.1.31I version=25.3.1.31I [Boot image]
        ncs1004-cosm-1.0.0.0-r253131I
```

Step 6 (Optional) Use the **appmgr process-script stop** command to stop the process script.

Example:

This command stops the execution of the process script *xr_script_scheduler*.

```
RP/0/RP0/CPU0:ios#appmgr process-script stop name xr_script_scheduler
Wed Oct 22 09:46:35.110 UTC
```

(Optional) Verify the status of the process script after stopping it. This example shows the process script *xr_script_scheduler* is *Activated* and *Stopped*.

```
RP/0/RP0/CPU0:ios#show appmgr process-script-table
Wed Oct 22 09:46:41.245 UT
Name                 Executable            Activated  Status    Restart Policy    Config Pending
-------------------- --------------------- ---------- --------- ----------------- ----------------
xr_script_scheduler  xr_script_scheduler.py       Yes     Stopped   Always                  No
```

Third-party scripts are installed and running on NCS 1004, enabling custom automation and monitoring as defined by the RPM package and run parameters.

Third-party Python scripts

A deployment of third-party Python scripts is a network automation capability that

- enables automation scripts to run directly on the router without relying on an external controller,
- leverages Python libraries and provides direct access to critical router information, and
- accelerates script execution while enhancing reliability by removing dependencies on network speed and reachability of external controllers.

Efficient network automation is pivotal for managing extensive cloud-computing networks. The Cisco IOS XR infrastructure supports automation by allowing API calls and execution of scripts directly on the router.

Traditionally, an external controller managed automation by communicating with NCS 1004 via interfaces such as NETCONF, SNMP, and SSH.

This feature streamlines operational workflows by enabling scripts to execute on the router itself, eliminating the dependency on external controller reachability and speed. Scripts can take advantage of Python libraries and interact directly with router-specific information, improving both execution speed and reliability.

When a third-party RPM is installed, the xr_script_scheduler.py script automatically executes the third-party Python script. To enable this automation, App manager configuration is required to activate xr_script_scheduler.py and ensure the third-party scripts are run post-installation.

Deploy and activate third-party scripts

Enable automated management and execution of custom Python scripts, delivered via third-party RPM packages, using AppManager on NCS 1004 optical devices.

NCS 1004 supports running custom scripts to automate node operations using AppManager. This capability offers flexibility and efficiency when configuring, monitoring, and debugging optical devices by reducing manual actions and supporting automation.

Follow these steps to deploy and activate third party script.

Procedure

Step 1 Use the **show script status** command to check the list of the OPS scripts that are in-built in XR.

Example:

This command lists the status of *xr_script_scheduler* script. *Ready* status in the output means that the script checksum is verified and is ready to run.

```
RP/0/RP0/CPU0:ios#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 2025

 xr_data_collector.py            | exec    | Ready             | NEW         | Tue Oct 24 07:10:36 2025

 xr_script_scheduler.py          | process| Ready             | NEW         | Tue Oct 24 07:10:36 2025
=============================================================================================================
RP/0/RP0/CPU0:ios#
```

Step 2 Use the appmgr to automatically run the XR scheduler script.

Activate the scheduler script automatically using the "autorun" option with the configuration.

Example:

```
RP/0/RP0/CPU0:ios#configure
RP/0/RP0/CPU0:ios(config)#appmgr
RP/0/RP0/CPU0:ios(config-appmgr)#process-script xr_script_scheduler
RP/0/RP0/CPU0:ios(config-process-script)#executable xr_script_scheduler.py
```

```
RP/0/RP0/CPU0:ios(config-process-script)#autorun
RP/0/RP0/CPU0:ios(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, skip the 'autorun' configuration line.

Step 3 Verify the scheduler script is running.

a) Run the **show script execution** command to verify the functioning of the debug and monitoring scripts.

Example:

This command displays a list of OPS scripts currently running.

```
RP/0/RP0/CPU0:ios# 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 2025 | 253.32s    |
 None   | Started

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

b) Use the **show script execution details** command to verify if the scheduler script is running.

Example:

This command displays a list of OPS scripts currently running. If the scheduler script is correctly configured and activated, the scheduler script execution detail appears in the output.

```
RP/0/RP0/CPU0:ios#show script execution details
Tue Oct 25 18:01:56.590 UTC
==========================================================================================================================

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

 1698170509| xr_script_scheduler.py (process)             | Tue Oct 25 18:01:49 2023 | 7.68s      |
 None   | Started
--------------------------------------------------------------------------------------------------------------------------

 Execution Details:
 ------------------
 Script Name  : xr_script_scheduler.py
 Version      : 25.3.1.14Iv1.0.0
 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 25 18:01:49 2025
      Time Elapsed : 0.00s Seconds
      Description  : Started by Appmgr
 2.   Event        : Started
      Time         : Tue Oct 25 18:01:49 2025
      Time Elapsed : 0.11s Seconds
      Description  : Script execution started. PID (15985)
```

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

Step 4 Copy the third party RPM files to the NCS 1004 node.

a) Use any of the file transfer mechanisms to copy third-party RPM.

Example:

This example shows copying the RPM to the harddisk of the NCS 1004 node using **scp**.

```
RP/0/RP0/CPU0:ios#scp
user@171.xx.xxx.xxx:/users/user/rpm-factory/RPMS/x86_64/nms-1.1-25.3.1.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

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

b) (Optional) Verify the RPM files using **dir <filepath>**.

Example:

```
RP/0/RP0/CPU0:ios#dir harddisk:/nms-1.1-24.1.1.x86_64.rpm
Wed Oct  24 19:53:54.041 UTC
```

Step 5 Install the third party RPM files to use the required debug and monitoring python scripts.

The third party RPM files have the customized scripts to be executed. The third-party RPM contains two types of files:

- One or more python scripts—For more information on developing python scripts, see IOS XR Programmability with Python and xr-python-scripts.
- Run parameter JSON file—*xr_script_scheduler.json* has instruction for the scheduler script.

Example:

This is an example *xr_script_scheduler.json* file. Customize this file as per your requirements.

```
[
    {
        "name": "__template_entry__.py",
        "description": ["**This is a template entry for documentation purpose. This entry will be
ignored**",
                        "name : Name of the python script to be executed",
                        "       [string][mandatory]",
                        "description:  Description of the script",
                        "              [string or list of strings][optional: default empty string]",

                        "cmd_line_parameters: Script command line parameters" ,
                        "                     [list of strings][optional: default Null][Example: ",
                        "env_variables: Enviromental variables to be set in script run shell",
                        "               [list of key value pairs][optional: default Null][Example:
[['INT_NAME': 'hu0/1/0/1']]",
                        "run_policy: Script restart policy when script exits",
                        "            [string: one of always/once/stop][optional: default 'always']",

                        "            always: restart the script every time it exits",
                        "            once:  do not restart the script if it exits",
                        "            stop:  stop an existing script run "
                        ],
        "cmd_line_parameters": [],
        "env_variables": [],
        "run_policy": "always"
```

```
    },
    {
        "name": "monitor_int_rx_cntr.py",
        "description": "Monitoring mgmt interface for Rx threshold of 100 ",
        "cmd_line_parameters": ["MgmtEth0/RP0/CPU0/0", "200", "-log", "debug"],
        "env_variables": [["INT_NAME", "FourHundredGigE0/9/0/0"], ["INT_NAME2",
"FourHundredGigE0/10/0/0"]],
        "run_policy": "always"
    },
    {
        "name": "monitor_int_rx_cntr.py",
        "description": "Monitoring Fo0/0/0/0 interface for Rx threshold of 1000000 ",
        "cmd_line_parameters": ["FourHundredGigE0/0/0/0", "1000000"],
        "run_policy": "once"
    },
    {
        "name": "monitor_int_rx_cntr2.py",
        "description": "Monitoring Fo0/0/0/1 interface for Rx threshold of 5000000 ",
        "cmd_line_parameters": ["FourHundredGigE0/0/0/1", "5000000"],
        "run_policy": "always"
    },
    {
        "name": "monitor_int_rx_cntr2.py",
        "description": "Monitoring Fo0/0/0/2 interface for Rx threshold of 5000000 ",
        "cmd_line_parameters": ["FourHundredGigE0/0/0/2", "8000000"],
        "run_policy": "stop"
    }
]
```

Example:

Use the **appmgr package install rpm <full RPM file path>** command to install the third-party RPMs.

```
RP/0/RP0/CPU0:ios#appmgr package install rpm /harddisk:/nms-1.1-25.3.1.x86_64.rpm
Tue Oct 24 18:03:26.685 UTC
RP/0/RP0/CPU0:ios#
RP/0/RP0/CPU0:ios#show appmgr packages installed
Tue Oct 24 19:42:07.967 UTC
Sno Package
--- ------------------------------------------------------------
1   nms-1.1-25.3.1.x86_64
RP/0/RP0/CPU0:ios#
```

After the scripts and the run parameters file become ready, build the RPM and configure the RPM to install files at `<default exr appmgr rpm install path>/ops-script-repo/exec/<rpm name>/`. RPM build tool for TPA is available at RPM Build Tool.

Note
Install the scripts in directories named after the RPM for smoother execution.

Step 6 Use the **show script status** command to verify that the scripts and the run parameter files contained in the RPM are all installed successfully and added to the script management repository.

Example:

This output shows the status that two scripts (monitor_int_xr_cntr.py and monitor_int_rx_cntr2.py) and a run parameter file (xr_script_scheduler.json) file were installed in the third-party RPM named “nms”.

```
RP/0/RP0/CPU0:ios#show script status

Tue Oct 24 19:41:10.696 UTC
=============================================================================================================

 Name                              | Type    | Status           | Last Action | Action Time
```

```
------------------------------------------------------------------------------------------------------------

 nms/monitor_int_rx_cntr.py      | exec   | Ready           | NEW         | Tue Oct 24 19:38:41 2023

 nms/monitor_int_rx_cntr2.py     | exec   | Ready           | NEW         | Tue Oct 24 19:38:41 2023

 nms/xr_script_scheduler.json    | exec   | Ready           | NEW         | Tue Oct 24 19:38:41 2023

 show_interfaces_counters_ecn.py | exec   | Ready           | NEW         | Tue Oct 24 19:33:52 2023

 xr_data_collector.py            | exec   | Ready           | NEW         | Tue Oct 24 19:33:52 2023

 xr_script_scheduler.py          | process| Ready           | NEW         | Tue Oct 24 19:33:52 2023

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

After the scripts are installed, the scheduler script starts reading the run parameter JSON file and executes the required debug and monitoring scripts.

The logs generated by the scripts are available in the directory `/harddisk\:/mirror/script-mgmt/logs/`.

Step 7 Verify that the debug and monitoring scripts are running.

Example:

Use the **show script execution** command to verify that the scripts are running.

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

(Optional) Use the **show script execution** [**name***script-name***detail [output][error]**]

Third-party Python scripts and their configuration files are deployed on NCS 1004, automatically activated, and running as scheduled. Device automation is enhanced, manual tasks reduced, and script logs are available for ongoing monitoring.