Application Hosting Configuration Guide for Cisco 8000 Series Routers, Cisco IOS XR Releases

Cisco 8000 Series Routers
Release: Application Hosting | Updated June 12, 2026

Topics included

Full Cisco Trademarks with Software License

Full Cisco Trademarks with Software License

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: https://www.cisco.com/c/en/us/about/legal/trademarks.html. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

1 Whats changed in this book

This cumulative guide provides a single, continuously updated version that includes all the latest IOS XR features and release updates. It simplifies your experience by letting you bookmark one link and access the complete guide, instead of navigating through multiple release-specific versions.

Specific changes or updates tied to individual releases are clearly called out within the relevant sections. For a list of features introduced in a specific release, refer to the Release Notes or the IOS XR Feature Finder.

The table lists the release numbers for which this document has been updated since its initial publication.

Table 1: Changes to this document

Date	Summary
June 2026	First published for Release 26.2.1

2 Introduction to Application Hosting

Topics:

Introduces the latest enhancements and modifications to application hosting features, including updates to Docker run options, TPA traffic prioritization, and new diagnostic tools like the CPU-Based Packet Generator.

You will learn how the application hosting framework has evolved through recent Cisco IOS XR releases, with detailed coverage of new capabilities such as customizable Docker run options, TPA traffic prioritization, and DDoS edge protection. The chapter also provides a reference for feature availability and documentation to assist in identifying the latest enhancements for your network environment.

Application hosting platforms

Explains application hosting platforms on Cisco 8000 Series Routers, which enable deployment of custom tools, diagnostic utilities, and automation agents directly on the router.

An application hosting platform is a containerized system that

- allows deployment of third-party custom tools, diagnostic utilities, and automation agents directly on Cisco 8000 series routers
- simplifies network management without requiring external controllers or additional hardware, and
- accelerates troubleshooting and optimizes network performance by enabling third-party automation directly on the router.

Application hosting bridges the Linux ecosystem with the Cisco IOS XR control plane, allowing you to integreate Third-Party Applications (TPAs) seamlessly. This extends the network operating system's functionality to meet the agility and automation demands of modern networking environments.

Components of application hosting

Lists the components that make up the application hosting solution on Cisco IOS XR and provides their roles.

The application hosting solution on Cisco IOS XR consists of several primary components, each contributing to native support for third-party application hosting. The primary components and their roles are:

- **Docker on IOS XR**: The Docker daemon is included with the 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 method for running Third Party Applications (TPAs) on IOS XR.
- **App Manager**: While the Docker daemon comes packaged with IOS XR, Docker applications can only be managed using the App Manager. The App Manager allows users to install applications packaged as RPMs and manage their lifecycle using the IOS XR CLI and programmable models.
- **PacketIO**: PacketIO is the router infrastructure that implements the packet path between TPAs and IOS XR running on the same router. It enables TPAs to leverage IOS XR forwarding for sending and receiving traffic.

Operational environments for application hosting

Lists the distinct operational environments available for hosting applications, focusing on their security, isolation, and networking capabilities.

Applications are hosted within specific operational environments that determine their security, isolation, and networking functionality:

- Sandbox Containers: Provides a secure, pre-configured environment (based on CentOS 8) tailored for third-party client applications. The Sandbox Manager enables easy deployment and integrates traffic prioritization through LPTS to ensure application reliability even during network congestion.
- Virtual Routing and Forwarding (VRF): Defines the network context for hosted applications. VRFs are synchronized into the Linux kernel as network namespaces to isolate applications within designated network segments. Key features include:
  - Traffic segregation: Separates management traffic from data plane traffic.
  - Cross-VRF communication: Uses relay agents to facilitate secure application management across separate routing instances.

Types of applications

Lists the different categories of Linux-based applications supported by the platform and describes their operational intent.

The platform supports a broad ecosystem of Linux-based tools, organized by their operational intent:

- Performance throughput testing: Tools such as iPerf are used to measure bandwidth, latency, and throughput across the network.
- Configuration management agents: Utilities like Chef or Puppet automate server functions, resource allocation, and user account management.
- Packet collection and analysis: Diagnostic tools using PacketIO capture, inspect, and generate network traffic, enabling troubleshooting and monitoring.
- Python-based applications: Custom automation scripts can be deployed and executed directly on the router, removing the need for external controllers.

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

Always avoid configuring MPLS packets on Linux interfaces within Docker container application hosting. This prevents unsupported behavior and potential failures.

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.

Application hosting use cases

Lists practical use cases for application hosting to highlight its real-world benefits.

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.

Docker container applications

Explains how Docker container applications enable flexible hosting of user-developed apps on IOS XR, allowing resource isolation and compatibility with diverse Linux libraries.

A Docker container application is a Linux-based application that

- runs inside a Docker container on IOS XR
- supports development with any Linux distribution for greater compatibility, and
- allows separate resource management for memory and CPU allocation distinct from IOS XR's root file system.

Docker container application hosting is ideal when applications require system libraries different from those provided by the IOS XR root file system. By isolating applications in their own containers, you can manage resource consumption and ensure compatibility without impacting the base operating system.

Leveraging containerization within the Cisco IOS XR environment offers enhanced operational flexibility:

- Running a custom network monitoring application developed for Ubuntu Linux inside a Docker container on IOS XR.
- Hosting microservices that depend on specific library versions not available in the IOS XR root file system.

3 Hosting Applications in Docker Containers

Topics:

Explains how to deploy and manage third-party applications and scripts on Cisco 8000 devices using Docker containers and the App Manager. It provides instructions for configuring resource limits, installing RPM packages, and automating operational tasks to extend device functionality and improve network management.

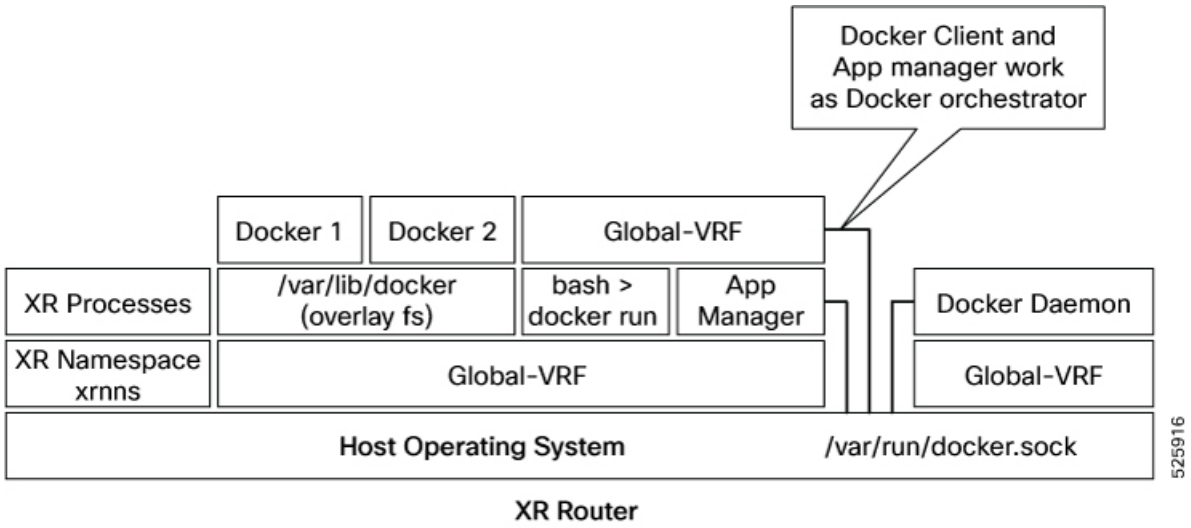
Docker container application hosting architectures

Explains how Cisco IOS XR uses Docker container application hosting architectures to enable efficient app management and execution.

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

Recommendation: Follow hosting guidelines for Docker containers

This topic outlines the key recommendations and limitations to consider when hosting 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

Explains the critical aspects of TP application resource configurations in IOS XR, emphasizing resource controls and security considerations to safeguard system operations.

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

Explore the general concepts and methodologies involved in TP application bring-up. This overview sets the foundation for specific deployment strategies and configurations.

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.

The 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

Configure a third-party application (TPA) on your device using application configuration and docker runtime options.

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

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.

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

Configure the docker run time options for TPAs using 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

Provides details about the native deployment model for TPAs, including integration steps and configuration options.

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

Provides information on the gNOI Containerz service for bringing up TPAs and managing container lifecycles on Cisco 8000 routers.

The gNOI Containerz service on the router 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 router.
- 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 is a workflow to onboard and manage third-party applications using gNOI RPCs.

Docker run options

Explains how Docker run options provide configuration flexibility for containers managed by Application Manager.

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 2: Feature History Table

Feature Name	Release Information	Description
Customize docker run options using application manager	Release 25.4.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on: - 8011-32Y8L2H2FH - 8011-12G12X4Y-A/D

Feature Name	Release Information	Description
Customize docker run options using application manager	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I
Customize docker run options using application manager	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM
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.

Table 3: 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

Docker run option	Description
--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
--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

Docker run option	Description
--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

Docker Application Management using IPv6 Address

Explains how Docker applications are managed using IPv6 addressing schemes, including configuration and operational details.

A Docker application is a software workload hosted within a container that

- is managed through virtual routing and forwarding (VRF) instances
- enables seamless integration across network namespaces within the host system, and
- supports both IPv4 and IPv6 management interfaces for enhanced accessibility.

Table 4: Feature History Table

Feature Name	Release Information	Description
Docker Application Management using IPv6 Address	Release 25.4.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8011-32Y8L2H2FH • 8011-12G12X4Y-A/D
Docker Application Management using IPv6 Address	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I
Docker Application Management using IPv6 Address	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM
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 application networking that enables traffic forwarding between VRF instances using a relay agent deployed inside 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. Configuration permits forwarding between either a single pair or multiple pairs of ports according to network requirements.

A primary use case is managing Linux-based Docker applications running in the default VRF through a management interface located in a separate VRF. This allows application management to occur seamlessly across VRF boundaries.

Starting with IOS-XR software release 7.11.1, management capabilities are enhanced: Docker applications hosted within containers can now be managed using IPv6 addresses via the Cisco 8000 router's management interface. This update improves accessibility and control compared to earlier versions that only supported IPv4 address management.

Consider a scenario where you need to manage Docker applications in the default VRF from a distinct management VRF. With IPv6 integration, you can access applications through their IPv6 addresses for configuration and monitoring, leveraging the Cisco 8000 router management interface.

Restrictions and limitations

- 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 App Manager.
- 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 for application manager

Understand the process of configuring Virtual Routing and Forwarding (VRF) for the application manager. This configuration helps in securely isolating traffic for different applications.

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

Procedure

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

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

2. Enable Basic Forwarding Between Two Ports.

   To enable traffic forwarding between two ports in different VRFs, use the following configuration:

Example

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

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

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

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

```
RP/0/RP0/CPU0:ios#--vrf-forward vrf-mgmt:5000 vrf-default:8000 --vrf-forward
 vrf-mgmt:5003-5004 vrf-default:8003-8004
RP/0/RP0/CPU0:ios#--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.

3. Verify the VRF forwarding.

a) Use the **show appmgr application name** keyword to verify the VRF forwarding.

Example

```
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:
com.azure.dev.image.build.buildnumber=2.3.2-dev-ricabrah-partho-xr-dev.1+28,


com.azure.dev.image.build.definitionname=swanagentXR,com.azure.dev.image.build.


repository.uri=https://1Wan@dev.azure.com/1Wan/SWAN/_git/swanagentXR,com.azure.dev.


image.system.teamfoundationcollectionuri=https://dev.azure.com/1Wan/,com.azure.dev.


image.build.builduri=vstfs:///Build/Build/8518,com.azure.dev.image.build.repository.


name=swanagentXR,com.azure.dev.image.build.sourcebranchname=partho-xr-dev,
```

```
com.azure.dev.image.build.sourceversion=0ebd43521870844688660c131b0921ea7e2dcb27,com.azure.


dev.image.system.teamproject=SWAN,image.base.ref.name=mcr.microsoft.com
      /mirror/docker/library/alpine:3.15
    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
```

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

Example

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

Third-party applications in Sandbox container using Sandbox Manager

Explains how Sandbox Manager enables hosting, configuration, deployment, and management of third-party client applications in a CentOS 8-based sandbox container on Cisco IOS XR.

A third-party application in a sandbox container is a client application that

- is hosted and managed in a CentOS 8-based container environment on Cisco IOS XR
- can be configured, deployed, and controlled using Sandbox Manager and respective third-party servers over a network, and

- applications placed in the sandbox container during router bootup using Zero Touch Provisioning (ZTP) are activated automatically when Sandbox Manager is enabled, and
- automatically restarts and activates after router reloads or RP switchovers when Sandbox Manager is enabled.

Sandbox Manager activates the sandbox container using the APPMGR client library APIs. The Sandbox container operates on CentOS 8, which enables you to control the applications inside the container using standard Docker commands.

Feature history
Table 5: 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 Manager uses IOS XR commands for managing the Sandbox container.

Supported commands on Sandbox Manager

This section describes the operations and the IOS XR commands that are supported on Sandbox Manager.

- Enable and disable Sandbox Manager: Use this command to enable or disable Sandbox Manager.
  - Enable Sandbox Manager:

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

  - Disable 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: Use these commands 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 Local Packet Transport Services (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 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 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 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
systemd-journald.service    loaded active running   Journal Service
systemd-tmpfiles-setup.service loaded active exited    Create Volatile
Files and Directories
-.slice                     loaded active active    Root Slice
system.slice                loaded active active    System Slice
dbus.socket                 loaded active listening D-Bus System Message
Bus Socket
systemd-journald.socket     loaded active running   Journal 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

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 show all installed unit files use 'systemctl list-unit-files'.
```

- Access sandbox: The following command is used to access the 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 the sandbox container:

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

Third party RPMs installation using App Manager install UI

Explains how third party RPMs add new capabilities to your system by utilizing App Manager's install interface.

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

Lists the specific limitations that apply to installing third-party RPMs using the App Manager.

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

Learn how to install third party RPMs using the App Manager install UI. This task provides the necessary steps to complete the installation process efficiently.

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

1. Create an RPM containing the application (in the form of a docker container image).
2. Use App manager RPM build tool to generate TP RPMs. See https://github.com/ios-xr/xr-appmgr-build/blob/main/README.md.
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

Configure App Manager install UI to remove unwanted third party RPM applications from your system.

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 the router 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 Cisco 8000 router.

Procedure

1. Decide whether you want to uninstall the RPM using the source name or the package name.
2. If uninstalling by source name, enter this command:

Example

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

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
```

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

Lists the supported IOS XR commands for the application manager, including action and show commands, along with their operations and usage examples.

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 6: 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.

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 7: 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.

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
```

How deploying and activating custom scripts works

Describes the overall process of deploying and activating custom automation scripts, including the roles of scheduler scripts and RPM packages.

Deploying custom scripts on Cisco 8000 routers enables advanced automation and monitoring for network 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

The router 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

Configure the App Manager to activate and run the XR scheduler script for automated operations.

Prepare the router to use scheduler scripts for automation and monitoring by configuring and activating them via App Manager.

The router 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

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#
```

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
```

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
```

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
```

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
```

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 the router, ready to automate node operations and manage additional scripts.

Deploy and manage third-party RPM scripts

Transfer, install, and configure third-party RPM packages to enable custom monitoring and automation scripts.

Expand 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 Cisco 8000 routers. Proper deployment and configuration allow for flexible automation, monitoring, and debugging.

Procedure

1. Copy the third party RPM files to the Cisco 8000 router.
   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 router 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
```

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.

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/`.

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]**

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
```

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 the router, enabling custom automation and monitoring as defined by the RPM package and run parameters.

Third-party Python scripts

Explains how automated deployments of third-party Python scripts operate within the Cisco IOS XR framework, including required configurations and execution steps.

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 the router 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

Deploy and activate third-party scripts to automate operational tasks, streamline management, and reduce manual errors.

Enable automated management and execution of custom Python scripts, delivered through third-party RPM packages, using App Manager on Cisco 8000 routers.

Cisco 8000 routers support running custom scripts to automate node operations using App Manager. This capability offers flexibility and efficiency when configuring, monitoring, and debugging network devices by reducing manual actions and supporting automation.

Follow these steps to deploy and activate third party script.

Procedure

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#
```

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.

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#
```

4. Copy the third party RPM files to the router.
   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 router 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
```

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.

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/`.

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 the router, automatically activated, and running as scheduled. Device automation is enhanced, manual tasks reduced, and script logs are available for ongoing monitoring.

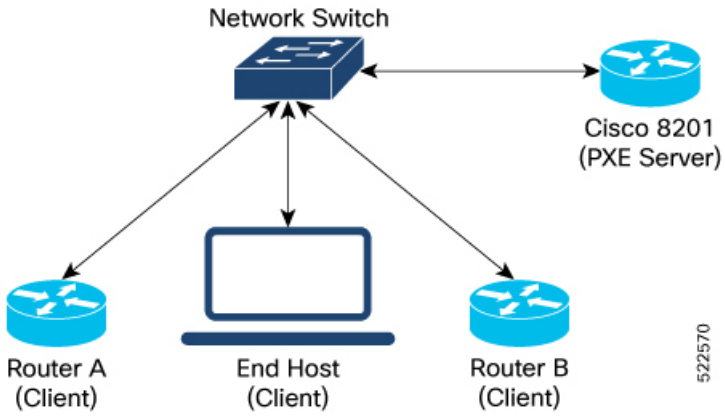
Boot devices using PXE server running in a Docker container

Provides details on the PXE server Docker container feature for Cisco IOS XR routers, including supported releases, hardware, specifications, and behavioral guidelines.

Preboot Execution Environment (PXE) is a client-server interface that enables devices in a network to download the files (like boot image, configurations and so on) from a PXE server.

The client uses DHCP protocol to receive the PXE server details and uses TFTP or HTTP protocol to download the file.

Figure 2: Client-Server Connection



PXE server Docker functionality

- The PXE server Docker feature enables PXE and iPXE functionality on routers running Cisco IOS XR software.
- It allows client devices (routers, Linux devices, VMs) to download boot images and configuration files from a Docker-hosted server managed by the application manager.
- Services provided within the PXE server Docker container:
  - DHCP
  - HTTP (iPXE)
  - TFTP (PXE)
- The feature is supported on both IPv4 and IPv6.
- The PXE server is installed on Cisco 8201 routers as a Docker container, managed by Cisco IOS XR Application Manager. The clients (routers or end-hosts such as Linux devices, VMs and so on) that are connected to this server can request and download the boot image.

PXE server Docker optional RPM

The optional RPM, xr-pxeserver, contains:

- Executables for the pxe_svr_mgr Cisco IOS XR process.
- PXE server Docker image: pxe-server-docker.rpm.
  - When installed, the Docker image is placed in the appmgr images directory: /pkg/opt/cisco/XR/appmgr/images/pxe-server-docker.rpm.
- Helper scripts: install_pxeserver.py and uninstall_pxeserver.py (located in /pkg/bin/) for manual installation/uninstallation.
  - Helper scripts are used for manual operations, otherwise use application manager commands.

Host and Activate the PXE Server Docker on Cisco 8201 Router

Host and activate the PXE server Docker container application on the Cisco 8201 router using the application manager to enable clients to download boot images and configuration files from the router.

Procedure

1. Install the optional RPM `xr-pxeserver` on the router.

   Example

   ```
   Router#install package add xr-pxeserver
   Router#install apply restart
   Router#install commit
   ```

   When the optional RPM `xr-pxeserver` is installed and activated, the `pxe_svr_mgr` process is instantiated. The `pxe_svr_mgr` process handles installation and updating the PXE server Docker RPM (`pxe-server-docker.rpm`) with the application manager. Also, when the optional RPM `xr-pxeserver` is upgraded, the `pxe_svr_mgr` process checks for the new version of the `pxe-server-docker.rpm`. If there is a change in the version number, it updates the existing version on the router to the new Docker RPM version and re-launches the PXE server Docker container for the changes to reflect.

> **Note**
>
> After activating the `xr-pxeserver` RPM package, the `pxe_svr_mgr` process installs the `pxe-server-docker.rpm` with application manager only when the ongoing install operation is committed and no more install operations are pending.

2. Verify the PXE server Docker container application package installed.

   Example

```
Router#show appmgr packages installed
Package
------------------------------------------------------------
pxe-server-2.1.0-ThinXR.x86_64
```

3. Create the following folder structure in the `/misc/disk1/` path and copy the `dhcp.conf` and `image.iso` files to their respective folders.

   Example

```
/server
|---- config
|      |---- dhcpd.conf
|      |---- dhcpd6.conf
|---- images
|      |---- Image-boot.iso
----logs
```

> **Note**
>
> The PXE server Docker container uses the `/server/` folder for its operations. This path is mounted to the PXE server Docker using the application manager configuration.
>
> In case of a dual RP system, it is recommended to create this directory structure under `/misc/disk1/mirror/`. This automatically syncs the PXE server related files to the standby RP node, making all files available on the new active RP node after RPFO. Otherwise, you must create the directory structure again and copy all the necessary files for the PXE server Docker container.

4. Configure and activate the PXE server Docker container application on the interface BVI301.

   Example

```
Router#config
Router(config)#appmgr
Router(config-appmgr)#application pxeserver
Router(config-application)#activate type docker source pxe-server
docker-run-opts "-it --restart always --cap-add=NET_ADMIN --net=host --log-opt
 max-size=20m --log-opt max-file=3 -v /misc/disk1/mirror/server:/server "
docker-run-cmd "-i BV301 -4 172.16.0.0/12 -6 2001:DB8::/48 -l /server/images
```

```
 -t"
Router(config-application)#commit
```

The parameters are:

- `--cap-add=NET_ADMIN --net=host` – Mandatory
- `-i <interface>` – Mandatory
- `-4 <secondary ipv4 address of BVI>`
- `-6 <ipv6 address of BVI>`
- `-l <location of image stored>` – Default: `/server/images`
- `-t <1 - tftp enabled, 0 - tftp disabled>`

5. Verify the PXE server Docker container status.

Example

```
Router(config)#appmgr application exec name pxeserver docker-exec-cmd status
dhcpd                            RUNNING   pid 94, uptime 0:00:05
dhcpd6                           RUNNING   pid 95, uptime 0:00:05
monitor                          RUNNING   pid 96, uptime 0:00:05
nginx                            RUNNING   pid 97, uptime 0:00:05
syslogd                          RUNNING   pid 98, uptime 0:00:05
tftp-hpa4                        RUNNING   pid 101, uptime 0:00:05
tftp-hpa6                        RUNNING   pid 104, uptime 0:00:05
```

What to do next

The PXE server Docker container is active and the clients can now download the boot image and the configuration file from the PXE server (Cisco 8201 router).

4 Cisco Secure DDoS Edge Protection

Topics:

Details the implementation of Cisco Secure DDoS Edge Protection at the network ingress, describing how to install detector containers and configure router settings to identify and mitigate malicious traffic.

Cisco Secure DDoS Edge Protection

Describes how Cisco Secure DDoS Edge Protection identifies and mitigates distributed denial-of-service (DDoS) attacks at the network edge, enhancing overall security and minimizing impact on core bandwidth.

A Cisco Secure DDoS Edge Protection feature is a network security mechanism that

- enables routers to respond immediately to DDoS threats at the network edge
- allows malicious traffic to be identified and counteracted directly on the router, and
- minimizes network and application impact without affecting core bandwidth by avoiding backhaul of malicious traffic.

A centralized controller manages DDoS mitigation using information from detectors deployed on routers. These detectors analyze IPv4 and IPv6 traffic in real time to identify DDoS attacks. Upon detection, the controller enforces deny ACLs to block malicious traffic while allowing legitimate traffic.

Feature history

This table provides the feature history for Cisco Secure DDoS Edge Protection:

Table 8: Feature history table

Feature name	Release information	Description
Cisco Secure DDoS Edge Protection	Release 25.4.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100]) (select variants only*) *This feature is supported on: • 8011-32Y8L2H2FH • 8011-12G12X4Y-A/D
Cisco Secure DDoS Edge Protection	Release 25.2.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100], 8010 [ASIC: A100]) (select variants only*); Centralized Systems (8600 [ASIC: Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) (select variants only*) You can now enable the router to detect DDoS attacks targeting MPLS traffic using DDoS edge protection. The router analyzes MPLS flows to identify malicious traffic patterns, ensuring the availability and performance of services traversing MPLS networks.
Cisco Secure DDoS Edge Protection	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100]) (select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I

Feature name	Release information	Description
Cisco Secure DDoS Edge Protection	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100]) (select variants only*); Modular Systems (8800 [LC ASIC: P100]) (select variants only*) *This feature is now supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM
Cisco Secure DDoS Edge Protection	Release 24.1.1	You can now efficiently block malicious traffic, safeguarding your network's performance and availability. Protection against DDoS attacks is implemented at the network edge, deployed at the ingress point where external network traffic enters. A centralized controller manages DDoS mitigation using information from detectors deployed on the routers. These detectors analyze IPv4 and IPv6 traffic in real time to identify DDoS attacks. Upon detection, the controller enforces deny ACLs to block malicious traffic while allowing legitimate traffic.

Key components and advantages of Cisco Secure DDoS Edge Protection

Provides a summary of the main components, mitigation workflow, and benefits of Cisco Secure DDoS Edge Protection for edge network security.

Cisco Secure DDoS Edge Protection uses a centralized controller and detectors deployed on routers to identify DDoS attacks at the network edge and mitigate malicious traffic at the ingress point.

Cisco Secure DDoS Edge Protection consists of these components:

Table 9: Cisco Secure DDoS Edge Protection components

Component	Description
Centralized controller	A highly available centralized controller, cloud-based or on-premises, that manages a collection of detectors deployed on routers. Key functions include: • Managing the container lifecycle for detectors • Configuring and editing detector profiles and security settings • Checking detector health, displaying real-time attack forensics and threat intelligence analyses • Controlling DDoS attack mitigation at the network ingress point • Providing real-time and historical event reporting • Operational control and incident response

Component	Description
Detector	A resource-efficient application container for real-time DDoS detection, deployed on edge or peering routers and managed by the centralized controller. It analyzes IPv4 and IPv6 traffic on each ingress interface to identify DDoS attacks as they occur. Starting from Cisco IOS XR Release 25.2.1, MPLS traffic is also analyzed using extracted information from the IPv4 and IPv6 payload to detect DDoS attacks accurately.

Upon detecting a DDoS attack, the centralized controller promptly enforces a deny ACL to block the attack traffic while still allowing legitimate traffic to pass through.

Cisco Secure DDoS Edge Protection provides these benefits:

- Stops DDoS attacks at the network ingress.
- Requires no additional hardware or facilities such as power, rack space, and cooling.
- Requires no changes to the network architecture.
- Avoids overprovisioning facilities such as links and routers to account for attack traffic.
- Prevents backhauling of malicious traffic.
- Minimizes network outages and optimizes the end-user experience.
- Meets low-latency application requirements.

DDoS edge protection restrictions

Adhere to these restrictions when deploying Cisco Secure DDoS Edge Protection to ensure correct traffic analysis, VRF configuration, and NetFlow exporter compatibility.

- The DDoS edge protection does not support tunnel traffic.
- The system supports only the default VRF configuration, and it applies solely to the management port. For effective communication between the Docker and the controller, you must configure the management port to operate within the default VRF exclusively. This setup guarantees that the Docker can reliably interact with the controller without any network interruptions.
- The version **protobuf** of flow exporter-map is only supported for Monitor Map configured with record mpls ipv4-ipv6-fields.

Install and configure DDoS edge protection

Install the Cisco Secure DDoS Edge Protection application through the DDoS edge protection controller and configure the router with the required UDF, Loopback, ACL, SSH, and TPA settings to enable controller-managed threat detection and mitigation.

Use this procedure to install the DDoS edge protection detector on the router using the DDoS edge protection controller, and to configure all required router settings that enable the controller to monitor traffic and enforce mitigation.

Before you begin

Before you begin:

- Configure the management interface to reach the DDoS controller IP address.
- Manually configure the base ACL, NetFlow, and SSH configurations.

Also review the restrictions in DDoS edge protection restrictions before proceeding.

Procedure

1. Install and download the DDoS edge protection controller software package.

   Download the controller software package from the Software Download page. After the controller installation is complete, you can access the user interface. Log in to the controller services instance to monitor, manage, and control the device.

   For more information on installing the DDoS controller, see the Cisco Secure DDoS Edge Protection Installation Guide.

2. Configure loopback interfaces on the router.

   Example

   ```
   Router(config)# interface Loopback100
   Router(config-if)#  ipv4 address 15.1.1.2 255.255.255.255
   Router(config-if)#  exit
   Router(config)# interface Loopback101
   Router(config-if)#  ipv4 address 17.1.1.2 255.255.255.255
   Router(config-if)# commit
   ```

3. Configure an ACL on the router.

   Example

   ```
   Router(config)# ipv4 access-list myACL
   Router(config-ipv4-acl)#  1301 permit ipv4 any any
   Router(config-ipv4-acl)#  exit
   Router(config)# ipv6 access-list myACL
   Router(config-ipv6-acl)#  1301 permit ipv6 any any
   Router(config-ipv6-acl)# exit
   Router(config)# commit
   ```

   The first 1301 entries are reserved for DDoS mitigation purposes. When a DDoS attack is detected, the controller automatically enforces deny ACL rules to block malicious traffic. The following is a sample configuration that the controller uses to deny attacker traffic:

   ```
   1 deny udp any eq 19 host 45.0.0.1 eq 0 packet-length eq 128 ttl eq 64
   2 deny tcp any host 45.0.0.1 eq www match-all -established -fin -psh +syn
   -urg packet-length eq 60 ttl eq 64
   1301 permit ipv4 any any
   ```

   For more information on implementing access lists, see Understanding Access-List.

4. Configure SSH on the router.

   Example

   ```
   Router(config)# ssh server v2
   Router(config)# ssh server netconf
   Router(config)# netconf agent tty
   Router(config-netconf-tty)# netconf-yang agent ssh
   ```

```
Router(config)# ssh timeout 120
Router(config)# ssh server rate-limit 600
Router(config)# ssh server session-limit 110
Router(config)# ssh server vrf default
Router(config)# ssh server netconf vrf default
Router(config)# commit
```

5. Configure TPA on the router.

> **Note**
>
> TPA configuration is not required for Cisco 8000 routers. This step applies to NCS 5500 platforms only. Reload the router after configuring the `hw-module profile`.

Example

```
Router(config)# tpa
Router(config-tpa)# linux networking
Router(config-tpa-vrf)# vrf default
Router(config-tpa-vrf)# east-west Loopback101
Router(config-tpa-vrf)# address-family ipv4
Router(config-tpa-vrf-afi)# default-route software-forwarding
Router(config-tpa-vrf-afi)# source-hint default-route interface Loopback100
```

6. Run the **ping** command to verify router connectivity to the DDoS edge protection controller.

Example

```
Router# ping 10.105.237.54
Thu Jun  1 07:16:43.654 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.105.237.54 timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/4 ms

RP/0/RP0/CPU0:Router# bash
[Router:~]$ ping 10.105.237.54
PING 10.105.237.54 (10.105.237.54) 56(84) bytes of data.
64 bytes from 10.105.237.54: icmp_seq=1 ttl=63 time=1.73 ms
64 bytes from 10.105.237.54: icmp_seq=2 ttl=63 time=1.29 ms
64 bytes from 10.105.237.54: icmp_seq=3 ttl=63 time=1.27 ms
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.270/1.510/1.751/0.230 ms
```

A 100% success rate in the ping output confirms that the router can reach the DDoS controller IP address. If pings fail, check the management interface configuration and routing.

7. Enter the device details into the DDoS edge protection controller panel and verify that the Deployment, Container, and Configuration indicators all display green.

After completing this procedure, the controller automatically applies the following NetFlow configuration on the router to enable traffic sampling and export to the detector container:

```
// Configuring Monitor Map
flow monitor-map DetectPro_Monitor_IPV6
 record ipv6 extended
 exporter DetectPro_GPB
 cache entries 1000000
 cache timeout active 1
 cache timeout inactive 1
 cache timeout rate-limit 1000000
!
flow monitor-map DetectPro_Monitor_IPV4
 record ipv4 extended
 exporter DetectPro_GPB
 cache entries 1000000
 cache timeout active 1
 cache timeout inactive 1
 cache timeout rate-limit 1000000
!
// Configuring Monitor Map MPLS (Release 25.2.1 and later)
flow monitor-map mon_mpls_ipv4_ipv6
 record mpls ipv4-ipv6-fields
 exporter DetectPro_MPLS
 cache entries 1000000
 cache timeout active 1
 cache timeout inactive 1
 cache timeout rate-limit 1000000
!
// Configuring Exporter Map
flow exporter-map DetectPro_GPB
 version protobuf
 transport udp 5005
 source TenGigE0/0/0/16
 destination 15.1.1.2
!
// Configuring Exporter Map MPLS (Release 25.2.1 and later)
flow exporter-map DetectPro_MPLS
 version protobuf
 transport udp 5005
 source loopback101
 destination 15.1.1.2
!
// Configuring Sampler Map
sampler-map DetectPro_NFv9
 random 1 out-of 100
!
// Configuring Interface MPLS (Release 25.2.1 and later)
interface TenGigE0/0/0/8
 ipv4 address 7.7.1.1 255.255.0.0
 flow mpls monitor mon_mpls_ipv4_ipv6 sampler samp_mpls ingress
!
```

For more information on Cisco Secure DDoS Edge Protection, see the Cisco Secure DDoS Edge Protection Data Sheet.

What to do next

Verify the DDoS edge protection deployment by running the verification procedure. See Verify DDoS edge protection application configuration.

Verify DDoS edge protection

Verify that the DDoS edge protection controller has successfully applied all configurations to the router by checking the AppMgr configuration, flow monitor cache, flow exporter status, and Docker application state.

Use this procedure to confirm that the DDoS controller has applied the required configuration to the router and that the detector container is running correctly.

Procedure

1. Run the **show running-config appmgr** command to verify the AppMgr configuration.

   Example

   ```
   RP/0/RP0/CPU0:Router# show running-config appmgr
   Thu Jun  1 07:33:36.741 UTC
   appmgr
    application esentryd
     activate type docker source esentryd-cisco-20230431633 docker-run-opts
   "--env-file /harddisk:/ENV_6478443711ac6830700d1aeb --net=host"
    !
   !
   ```

   The output confirms that the `esentryd` application is activated as a Docker container with the correct source and run options. The `--net=host` option is required for the container to communicate with the DDoS controller.

2. Run the **show flow monitor** to confirm that the monitor maps are active and exporting flows.

   Example

   ```
   Router# show flow monitor DetectPro_Monitor_IPV4 cache location 0/0/CPU0
   Cache summary for Flow Monitor DetectPro_Monitor_IPV4:
   Cache size:                         1000000
   Current entries:                          0
   Flows added:                     2243884200
   Flows not added:                          0
   Ager Polls:                      2243884200
     - Active timeout                        0
     - Inactive timeout                      0
     - Total                        2243884200
   Flows exported                   2243884200
   Matching entries:                         0
   !

   Router# show flow monitor DetectPro_Monitor_IPV6 cache location 0/0/CPU0
   Cache summary for Flow Monitor DetectPro_Monitor_IPV6:
   Cache size:                         1000000
   Current entries:                          0
   Flows added:                          59971
   Flows not added:                          0
   Ager Polls:                           94437
     - Active timeout                    59971
     - Total                             59971
   Flows exported                        59971
   Matching entries:                         0

   Router# show flow monitor mon_mpls_ipv4_ipv6 cache format record location
   0/RP0/CPU0
   ```

```
Cache summary for Flow Monitor mon_mpls_ipv4_ipv6:
Cache size:                         1000000
Current entries:                          0
Flows added:                            963
Flows exported                          963
!
```

The output shows that flows are being added and exported for both `DetectPro_Monitor_IPV4` and `DetectPro_Monitor_IPV6` monitor maps. From Release 25.2.1, verify the `mon_mpls_ipv4_ipv6` monitor map as well. Non-zero `Flows exported` values confirm that the monitor maps are actively exporting traffic data to the detector container.

3. Run the **show flow exporter** command to confirm that traffic data is being exported to the detector container destination.

   Example

```
RP/0/RP0/CPU0:Router# show flow exporter DetectPro_GPB location 0/0/CPU0
Thu Nov 16 06:13:58.059 UTC
Flow Exporter: DetectPro_GPB
Export Protocol: protobuf
Flow Exporter memory usage: 5265344
Used by flow monitors: DetectPro_Monitor_IPV4
                       DetectPro_Monitor_IPV6
Status: Disabled
Transport:   UDP
Destination: 15.1.1.2        (5005) VRF default
Source:      0.0.0.0         (54482)
Flows exported:                                  0 (0 bytes)
Packets exported:                         20355756 (27716506821 bytes)
Packets dropped:                                 0 (0 bytes)
```

From Release 25.2.1, also verify the MPLS exporter:

```
Router# show flow exporter exp_mpls_ipv4_ipv6 location 0/RP0/CPU0
Flow Exporter: exp_mpls_ipv4_ipv6
Export Protocol: protobuf
Used by flow monitors: mon_mpls_ipv4_ipv6
Status: Normal
Transport:   UDP
Destination: 15.1.1.2        (5005) VRF default
Source:      17.1.1.1        (54341)
Flows exported:                                963 (159519 bytes)
Packets exported:                              122 (159519 bytes)
Packets dropped:                                 0 (0 bytes)
```

The output confirms the exporter destination IP (`15.1.1.2`), UDP port (`5005`), and protocol (`protobuf`). A non-zero `Packets exported` value confirms that traffic is being sent to the detector container. Zero `Packets dropped` confirms no export failures.

4. Run the **show appmgr application-table** command to confirm the Docker application is running.

   Example

```
RP/0/RP0/CPU0:Router# show appmgr application-table
Thu Nov 16 06:13:58.059 UTC
Name     Type   Config State Status
-------- ------ ------------ ------------------------------------------------
esentryd Docker  Activated   Up 8 minutes
RP/0/RP0/CPU0:Router#
```

The output confirms that the `esentryd` Docker application is in the `Activated` config state and the status shows `Up`, indicating that the detector container is running successfully.

All verification checks confirm that the DDoS edge protection controller has applied the required configuration to the router, the detector container is running and active, and flow records are being exported correctly for DDoS traffic analysis.

5 Packet IO Communication

Topics:

Describes the Packet I/O framework that exposes Cisco IOS XR network interfaces to the Linux kernel, enabling the use of standard Linux networking tools, VRF isolation, and CPU-based packet generation for diagnostics.

Packet I/O communications

Explains how Packet I/O provides communications between IOS XR interfaces and Linux applications using the Linux networking stack.

A Packet I/O communication is a framework that

- exposes IOS XR network interfaces as standard Linux kernel interfaces
- enables Linux applications running on the router to send, receive, and manage traffic as if operating on a regular Linux system, and
- allows off-the-shelf Linux tools to run alongside IOS XR without reconfiguration.
- Exposed XR interfaces (EXIs): Interfaces that inherit IP address, MTU, and MAC address settings from the corresponding IOS XR configuration.
- to_xr interface: Connects the Linux network stack to the IOS XR routing and forwarding infrastructure, enabling default reachability based on XR's Routing Information Base (RIB) and Forwarding Information Base (FIB).

Virtual IP address support

Virtual IP addresses allow a single IP address to connect to the active route processor (RP) after an RP switchover event. This supports third-party applications and IOS XR applications using the Linux networking stack, enabling management access from a network with a single virtual IP address through gRPC or similar protocols.

Automatic synchronization of secondary IP addresses

Secondary IPv4 and IPv6 addresses configured on IOS XR interfaces are automatically synchronized with the Linux operating system, removing the need for manual configuration in Linux. Third-party applications deployed on IOS XR can use these secondary addresses directly.

Supported interface types:

- Exposed XR interfaces (EXIs): Secondary IP addresses are added to the corresponding Linux interface.
- Address-only interfaces: Secondary IP addresses are added to the Linux loopback device.

> **Note**
>
> Secondary IPv4 addresses are not synchronized from Linux to IOS XR for Linux-managed interfaces. Use the `ip addr show` command to view all configured IPv4 and IPv6 addresses; the `ifconfig` command displays only the first IPv4 address.

For more information on secondary IPv4 and IPv6 addresses, see ipv4 address (network) and ipv6 address.

Packet I/O infrastructure reachability

Describes how IOS XR interfaces are exposed to the Linux kernel and how the global virtual routing and forwarding (VRF) namespace reaches IOS XR routing through the `to_xr` interface.

Packet I/O infrastructure reachability is a networking capability that

- exposes IOS XR interfaces as Linux kernel interfaces with inherited IP address, MTU, and MAC address values

- enables access to the global virtual routing and forwarding (VRF) namespace through the bash command in the Linux shell, and
- connects Linux routing to the IOS XR default VRF using the `to_xr` interface.

Accessing the global VRF network namespace through the **bash** command reflects the IOS XR default VRF in the kernel, enabling default reachability based on IOS XR routing and Packet I/O infrastructure.

Feature history

Table 10: Feature history table

Feature name	Release information	Description
Virtual IP address in the Linux networking stack	Release 25.4.1	Introduced in this release on fixed systems (8010 [ASIC: A100]) for select variants. This feature is supported on: • 8011-32Y8L2H2FH • 8011-12G12X4Y-A/D
Virtual IP address in the Linux networking stack	Release 25.1.1	Introduced in this release on fixed systems (8700 [ASIC: K100], 8010 [ASIC: A100]) for select variants. This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I
Virtual IP address in the Linux networking stack	Release 24.4.1	Introduced in this release on fixed systems (8200 [ASIC: P100], 8700 [ASIC: P100]) and modular systems (8800 [LC ASIC: P100]) for select variants. This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Feature name	Release information	Description
Virtual IP address in the Linux networking stack	Release 7.5.2	Virtual IP addresses allow a single IP address to connect to the current active route processor (RP) after an RP switchover event. This functionality enables the network stack to support virtual IP addresses for third-party applications and IOS XR applications using the Linux networking stack. These commands are modified: • ipv4 virtual address • ipv6 virtual address • show linux networking interfaces address-only
Automatic synchronization of secondary IPv4 addresses from IOS XR to Linux operating system	Release 25.4.1	Introduced in this release on fixed systems (8010 [ASIC: A100]) for select variants. This feature is supported on: • 8011-32Y8L2H2FH • 8011-12G12X4Y-A/D
Automatic synchronization of secondary IPv4 addresses from IOS XR to Linux operating system	Release 25.1.1	Introduced in this release on fixed systems (8700 [ASIC: K100], 8010 [ASIC: A100]) for select variants. This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I
Automatic synchronization of secondary IPv4 addresses from IOS XR to Linux operating system	Release 24.4.1	Introduced in this release on fixed systems (8200 [ASIC: P100], 8700 [ASIC: P100]) and modular systems (8800 [LC ASIC: P100]) for select variants. This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Feature name	Release information	Description
Automatic synchronization of secondary IPv6 addresses from IOS XR to Linux operating system	Release 7.11.1	Configured interface secondary IPv6 addresses on IOS XR are automatically synchronized to the Linux operating system. Third-party applications can use secondary IPv6 addresses without manual intervention.
Automatic synchronization of secondary IPv4 addresses from IOS XR to Linux operating system	Release 7.5.3	Configured interface secondary IPv4 addresses on IOS XR are automatically synchronized to the Linux operating system. Third-party applications can use secondary IPv4 addresses without manual intervention.

Verify the reachability of IOS XR and Packet I/O infrastructure

Verify Packet I/O reachability by checking IOS XR interface state in Linux, confirming global VRF access from bash, and reviewing default routes.

Use this task to confirm that IOS XR interfaces are visible from the Linux networking stack and that the Packet I/O infrastructure delivers default reachability through IOS XR routing.

Before you begin

- Obtain access credentials for both the IOS XR console and Linux shell.
- Confirm SSH connectivity from your Linux host to the IOS XR router.

Follow these steps to verify reachability:

Procedure

1. Access the IOS XR console from your Linux host using SSH.

   Example

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
Router#
```

2. View Ethernet interfaces on IOS XR to confirm their status.

   Example

```
Router#show ip interface brief
Interface                      IP-Address      Status          Protocol
Vrf-Name
FourHundredGigE0/0/0/0         unassigned      Shutdown        Down     default
...
HundredGigE0/0/0/24            10.1.1.10       Up              Up       default
MgmtEth0/RP0/CPU0/0            192.168.122.22  Up              Up       default
```

> **Note**
>
> Use the **ip addr show** or **ip link show** commands to view all corresponding interfaces in Linux. Interfaces in the IOS XR `admin-down` state also show a `Down` state in the Linux kernel.

3. Check the IP and MAC addresses of interfaces in the `Up` state. In this example, `HundredGigE0/0/0/24` and `MgmtEth0/RP0/CPU0/0` are in the `Up` state.

   Example

```
Router#show interfaces HundredGigE0/0/0/24
HundredGigE0/0/0/24 is up, line protocol is up
  Hardware is HundredGigE0/0/0/24, address is 0000.5e00.5301 (bia
0000.5e00.5301)
  Internet address is 10.1.1.1/24
  MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
```

4. Verify that the **bash** command runs in the global VRF to view network interfaces.

   Example

```
Router#bash -c ifconfig
Hu0_0_0_24  Link encap:Ethernet  HWaddr 00:00:5e:00:53:01
            inet addr:10.1.1.10  Bcast:0.0.0.0  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
to_xr       Link encap:UNSPEC  HWaddr
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
            UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
```

   The `to_xr` interface indicates access to the global VRF.

5. Access the Linux shell.

   Example

```
Router#bash
[ios:~]$
```

6. (Optional) View the IP routes associated with the `to_xr` interface in Linux.

   Example

```
[ios:~]$ip route
default dev to_xr scope link metric 2048
192.0.2.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.0.2.41
10.1.1.0/24  dev Hu0_0_0_0     proto kernel scope link src 10.1.1.1
172.17.0.0/16 dev docker0     proto kernel scope link src 172.17.0.1 linkdown
```

> **Note**
>
> You can enter the global VRF directly after logging in to IOS XR using the **run ip netns exec vrf-default bash** command.

IOS XR interfaces appear in the Linux kernel, the global VRF is accessible from the bash shell, and the `to_xr` interface is present in the routing table, confirming that Packet I/O infrastructure is operational.

Programme routes in the kernel

Programme default and connected routes into the Linux kernel to allow Linux applications to send and receive traffic through the IOS XR network stack using the `to_xr` interface.

Use this procedure to configure the Linux kernel routing table so that Linux applications can communicate over the IOS XR network stack. Because IOS XR acts as the network stack for the system, the Linux kernel needs basic routes to connect into and use the IOS XR network stack.

Two types of routes can be programmed in the kernel:

- Default route: Sends traffic destined to unknown subnets out of the kernel through the `to_xr` interface to IOS XR for routing. Because `to_xr` has no associated IP address, a source hint is required to set the source IP address for outgoing packets.
- Local or connected routes: Routes associated with the subnet configured on interfaces. For example, the `10.1.1.0/24` network is associated with the `Hu0_0_0_24` interface.

Procedure

1. View the current routes from the bash shell.

   Example

   ```
   [ios:~]$ip route
   default dev to_xr scope link src 10.1.1.10 metric 2048
   10.1.1.0/24 dev Hu0_0_0_24 proto kernel scope link src 10.1.1.10
   192.168.122.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.168.122.22
   ```

2. Update the source hint for the default route to use the management port IP address.

   Example

   ```
   Router#bash

   [ios:~]$ip route replace default dev to_xr scope link src 192.168.122.22
   metric 2048

   [ios:~]$ip route
   default dev to_xr scope link src 192.168.122.22 metric 2048
   ```

```
10.1.1.0/24 dev Hu0_0_0_24 proto kernel scope link src 10.1.1.10
192.168.122.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.168.122.22
```

With this updated source hint, any default traffic exiting the system uses the management port IP address as the source IP address.

The Linux kernel routing table is updated with a default route through `to_xr` and connected routes for each configured interface, enabling Linux applications to send and receive traffic through the IOS XR network stack.

Configure VRFs in the kernel

Configure a custom VRF in IOS XR and verify that it is automatically synchronized to the Linux kernel as a network namespace, enabling Linux applications to operate within isolated VRF contexts.

Use this procedure to configure IOS XR VRFs so that corresponding Linux network namespaces are created automatically. This capability isolates Linux applications or processes into specific VRFs, such as an out-of-band management VRF, and allows those applications to open sockets and send or receive traffic only on interfaces in that VRF.

VRFs configured in IOS XR are automatically synchronized to the Linux kernel as network namespaces (netns). Each globally configured VRF creates a Linux network namespace with the prefix `vrf` added to the VRF name. The default VRF in IOS XR is named `default` and appears in Linux as `vrf-default`.

Procedure

---

1. Identify the current network namespace or VRF.

   Example

   ```
   [ios:~]$ip netns identify $$
   vrf-default
   global-vrf
   ```

2. Configure a custom VRF named `blue` in IOS XR.

   Example

   ```
   Router#conf t
   Router(config)#vrf blue
   Router(config-vrf)#commit
   ```

3. Verify that VRF `blue` is configured in IOS XR.

   Example

   ```
   Router#show run vrf
   vrf blue
   !
   ```

4. Verify that VRF `blue` is created as a network namespace in the Linux kernel.

Example

```
Router#bash
[ios:~]$ls -l /var/run/netns
total 0
-r--r--r--. 1 root root 0 Jul 30 04:17 default
-r--r--r--. 1 root root 0 Jul 30 04:17 global-vrf
-r--r--r--. 1 root root 0 Jul 30 04:17 tpnns
-r--r--r--. 1 root root 0 Aug  1 17:01 vrf-blue
-r--r--r--. 1 root root 0 Jul 30 04:17 vrf-default
-r--r--r--. 1 root root 0 Jul 30 04:17 xrnns
```

5. Access VRF `blue` to launch and run processes from the new network namespace.

Example

```
[ios:~]$ip netns exec vrf-blue bash
[ios:~]$
[ios:~]$ip netns identify $$
vrf-blue
[ios:~]$
```

Running **ifconfig** shows only the default `to_xr` interface because no IOS XR interface is currently assigned to this VRF.

6. Configure interface `HundredGigE 0/0/0/24` in VRF `blue` from IOS XR. The interface is automatically configured in the `vrf-blue` namespace in the Linux kernel.

Example

```
Router#conf t
Router(config)#int HundredGigE 0/0/0/24
Router(config-if)#no ipv4 address
Router(config-if)#vrf blue
Router(config-if)#ipv4 address 10.1.1.10/24
Router(config-if)#commit
```

7. Verify that interface `HundredGigE 0/0/0/24` is configured in VRF `blue` in IOS XR.

Example

```
Router#show run int HundredGigE 0/0/0/24
interface HundredGigE0/0/0/24
 vrf blue
 ipv4 address 10.1.1.10 255.255.255.0
!
```

8. Verify that the interface is configured in VRF `blue` in the Linux kernel.

Example

```
Router#bash
[ios:~]$ip netns exec vrf-blue bash
[ios:~]$ifconfig
Hu0_0_0_24  Link encap:Ethernet  HWaddr 78:e7:e8:d3:20:c0
            inet addr:10.1.1.10  Bcast:0.0.0.0  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

The interface appears in the `vrf-blue` network namespace, confirming successful synchronization between IOS XR and the Linux kernel.

The VRF `blue` is created as a Linux network namespace `vrf-blue`, the IOS XR interface is visible in that namespace, and Linux processes can be isolated to this VRF context.

Open Linux sockets

Open TCP or UDP Linux sockets from the bash shell and verify that the socket entries are programmed into the Local Packet Transport Services (LPTS) infrastructure to distribute traffic to the hosted application.

Use this procedure to open Linux sockets so that applications can receive traffic from IOS XR interfaces. Socket entries are programmed into the LPTS infrastructure, which distributes the information through the line cards. Any packet received on a line card interface triggers an LPTS lookup to direct the packet to the application that opened the socket.

Procedure

---

1. From the bash shell, open a TCP socket using **netcat** and verify that applications open sockets.

   Example

   ```
   Router#bash
   [ios:~]$ nc -l 0.0.0.0 -p 5000 &
   [1] 1160
   [ios:~]$
   [ios:~]$netstat -nlp
   Active Internet connections (only servers)
   Proto Recv-Q Send-Q Local Address  Foreign Address  State   PID/Program name
   tcp        0      0 0.0.0.0:5000   0.0.0.0:*        LISTEN  1160/nc
   tcp        0      0 0.0.0.0:57777  0.0.0.0:*        LISTEN  14723/emsd
   tcp        0      0 0.0.0.0:22     0.0.0.0:*        LISTEN  8875/ssh_server
   tcp6       0      0 :::22          :::*              LISTEN  8875/ssh_server
   udp        0      0 0.0.0.0:68     0.0.0.0:*                13235/xr_dhcpcd
   [ios:~]$exit
   Logout
   Router#
   ```

   Netcat starts listening on port 5000, which appears as an IPv4 TCP socket in the **netstat** output, as in a typical Linux kernel. This socket is programmed to LPTS, which creates a corresponding hardware entry to look up TCP port 5000. Incoming traffic is redirected to the kernel of the active RP where netcat runs.

2. Verify that the socket is programmed into the LPTS infrastructure.

   Example

   ```
   Router#show lpts pifib brief | i 5000
   Thu Aug 1 17:16:00.938 UTC
   IPv4 default  TCP  any  0/RP0/CPU0  any,5000  any
   ```

   The LPTS entry for TCP port 5000 confirms that traffic destined for that port is correctly directed to the Linux application through the Packet I/O infrastructure.

The Linux socket is open and the LPTS entry confirms that the IOS XR packet infrastructure will direct matching traffic to the application.

Send and receive traffic through a Linux socket

Send traffic from an external server to an open Linux socket on the router to verify that the Packet I/O infrastructure correctly delivers packets to hosted Linux applications.

Use this procedure to confirm end-to-end traffic delivery through the Packet I/O infrastructure after opening a Linux socket. Connect from an external server to the netcat socket that was started in the `vrf-default` network namespace, using an interface in the same VRF.

Before you begin

Before you begin, complete the steps in Open Linux sockets to start a listening socket on the router.

Procedure

From an external server, connect to the netcat socket on the router.

Example

```
[root@localhost ~]#nc -vz 192.168.122.22 5000
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 192.168.122.22:5000.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
```

The successful connection confirms that the Packet I/O infrastructure is delivering traffic from the external server to the Linux application running on the router. Connect over an interface that is in the same VRF as the socket (`vrf-default`) to ensure reachability.

Traffic sent from the external server reaches the Linux socket on the router, confirming that the Packet I/O infrastructure is correctly routing packets to the hosted application.

Manage IOS XR interfaces through Linux

Linux-managed interfaces allow standard Linux automation tools to configure and manage IOS XR interfaces, enabling IP address changes, MTU settings, and state management to propagate bidirectionally between Linux and IOS XR.

A Linux-managed interface is an IOS XR exposed XR interface (EXI) that is configured to accept management input from Linux rather than from the IOS XR configuration. Key characteristics include:

- By default, all IOS XR interfaces are managed through IOS XR configuration (CLI or YANG models), with attributes such as IP address, MTU, and state inherited from the IOS XR configuration and interface state.
- When an interface is set to Linux-managed mode, standard Linux automation tools on Linux servers can manage IOS XR interfaces directly, and configuration changes made from Linux propagate back to IOS XR.
- Secondary IPv4 addresses cannot be managed from Linux; only primary addresses are supported in Linux-managed mode.

Linux-managed interface capabilities

Each network namespace in the Linux system contains a set of interfaces that map to IOS XR interfaces. When an interface is Linux-managed, you can:

- Configure new IP addresses on the interface from Linux using the **ip addr add** command.
- Configure a custom MTU using the **ip link set mtu** command, which propagates the change back to the IOS XR configuration.
- Bring the interface up or down using **ifconfig** commands, with state changes reflected in IOS XR.

Configure an interface to be Linux-managed

This task shows you how to configure a router interface to be Linux-managed on a Cisco IOS XR device using the **linux networking exposed-interfaces** command, and verify the configuration from both the XR CLI and the Linux bash shell.

Use this procedure to expose a router interface to Linux so that the interface is accessible for packet management, routing, and diagnostics from the IOS XR bash shell.

Procedure

1. Check the available exposed-interfaces in the system.

   Example

```
Router(config)#linux networking exposed-interfaces interface ?

  Bundle-Ether     Aggregated Ethernet interface(s) | short name is BE
  FiftyGigE        FiftyGigabitEthernet/IEEE 802.3 interface(s) | short name
 is Fi
  FortyGigE        FortyGigabitEthernet/IEEE 802.3 interface(s) | short name
 is Fo
  FourHundredGigE  FourHundredGigabitEthernet/IEEE 802.3 interface(s) | short
 name is FH
  GigabitEthernet  GigabitEthernet/IEEE 802.3 interface(s) | short name is
Gi
  HundredGigE      HundredGigabitEthernet/IEEE 802.3 interface(s) | short
name is Hu
  Loopback         Loopback interface(s) | short name is Lo
  MgmtEth          Ethernet/IEEE 802.3 interface(s) | short name is Mg
  TenGigE          TenGigabitEthernet/IEEE 802.3 interface(s) | short name
is Te
  TwentyFiveGigE   TwentyFiveGigabitEthernet/IEEE 802.3 interface(s) | short
 name is TF
  TwoHundredGigE   TwoHundredGigabitEthernet/IEEE 802.3 interface(s) | short
 name is TH
```

2. Configure the interface to be managed by Linux.

   Example

   The following example shows how to configure a HundredGigE interface to be managed by Linux:

```
Router#configure
Router(config)#linux networking exposed-interfaces interface HundredGigE
0/0/0/24 linux-managed
Router(config-exi-if)#commit
```

3. View the interface details and the VRF.

   Example

   The following example shows the information for HundredGigE interface:

```
Router#show run interface HundredGigE0/0/0/24
interface HundredGigE0/0/0/24
mtu 4110
vrf blue
ipv4 mtu 4096
ipv4 address 10.1.1.10 255.255.255.0
ipv6 mtu 4096
ipv6 address fe80::7ae7:e8ff:fed3:20c0 link-local
!
```

4. Verify the configuration in XR.

   Example

   The following example shows the configuration for HundredGigE interface:

```
Router#show running-config linux networking
linux networking
 exposed-interfaces
  interface HundredGigE0/0/0/24 linux-managed
  !
 !
!
```

5. Verify the configuration from Linux.

   Example

   The following example shows the configuration for HundredGigE interface:

```
Router#bash
Router:Aug 1 17:40:02.873 UTC: bash_cmd[67805]: %INFRA-INFRA_MSG-5-RUN_LOGIN
 : User vagrant logged into shell from vty0

[ios:~]$ip netns exec vrf-blue bash

[ios:~]$ifconfig
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

[ios:~]$ifconfig -a
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
```

```
BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Configure a new IP address on a Linux-managed interface

Add a new IP address to a Linux-managed IOS XR interface using the Linux **ip addr add** command, which automatically propagates the change to the IOS XR configuration.

Use this procedure to configure a new IP address on a Linux-managed interface from the Linux shell. Changes made from Linux are propagated automatically to the IOS XR configuration for the interface.

Before you begin

Before you begin, configure the interface as Linux-managed. For details, see Configure an interface to be Linux-managed.

Procedure

1. Configure the IP address on the interface from the Linux shell.

   Example

   ```
   [ios:~]$ip addr add 10.1.1.10/24 dev Hu0_0_0_24
   [ios:~]$Router:Aug 1 17:41:11.546 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT
   : Configuration
   committed by user 'system'. Use 'show configuration commit changes 1000000021'
    to view the changes.
   ```

   The system log confirms that the IP address change has been committed to the IOS XR configuration automatically.

2. Verify that the new IP address is configured on the interface.

   Example

   ```
   [ios:~]$ifconfig Hu0_0_0_24
   Hu0_0_0_24  Link encap:Ethernet  HWaddr 78:e7:e8:d3:20:c0
               inet addr:10.1.1.10  Bcast:0.0.0.0  Mask:255.255.255.0
               BROADCAST MULTICAST  MTU:1500  Metric:1
   ```

The IP address `10.1.1.10` appears on the interface, confirming that the configuration was applied successfully.

The new IP address is configured on the Linux-managed interface and the change is automatically committed to the IOS XR configuration.

Configure custom MTU setting

This task shows you how to bring up a Linux-managed interface and configure a custom MTU value using the **ip link set** command, then verify the change from both the Linux shell and the IOS XR running configuration.

Use this procedure to set a custom maximum transmission unit (MTU) on a Linux-managed interface and confirm that the updated MTU is reflected in both the Linux interface configuration and the IOS XR running configuration.

Procedure

---

1. Configure the MTU setting.

   Example

```
[ios:~]$ifconfig Hu0_0_0_24 up

[ios:~]$Router:Aug 1 17:41:54.824 UTC: ifmgr[266]: %PKT_INFRA-LINK-3-UPDOWN
: Interface
HundredGigE0/0/0/24, changed state to Down
Router:Aug 1 17:41:54.824 UTC: ifmgr[266]: %PKT_INFRA-LINEPROTO-5-UPDOWN :
Line protocol on
Interface HundredGigE0/0/0/24, changed state to Down
Router:Aug 1 17:41:56.448 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT :
Configuration committed by
user 'system'. Use 'show configuration commit changes 1000000022' to view the
 changes.
Router:Aug 1 17:41:56.471 UTC: ifmgr[266]: %PKT_INFRA-LINK-3-UPDOWN : Interface
HundredGigE0/0/0/24, changed state to Up
Router:Aug 1 17:41:56.484 UTC: ifmgr[266]: %PKT_INFRA-LINEPROTO-5-UPDOWN :
Line protocol on
Interface HundredGigE0/0/0/24, changed state to Up
Router:Aug 1 17:41:58.493 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT :
Configuration committed by
user 'system'. Use 'show configuration commit changes 1000000023' to view the
 changes.

[ios:~]$
[ios:~]$ip link set dev Hu0_0_0_24 mtu 4096
[ios:~]$
[ios:~]$Router:Aug 1 17:42:46.830 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT
: Configuration
committed by user 'system'. Use 'show configuration commit changes 1000000024'
 to view the changes.
```

2. Verify that the MTU setting has been updated in Linux.

   Example

```
[ios:~]$ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
```

```
inet6 addr: fe80::7ae7:e8ff:fed3:20c0/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:4096 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:648 (648.0 B)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

3. Check the effect on the IOS XR configuration with the change in MTU setting on this interface.

Example

```
Router#show running-config int HundredGigE0/0/0/24
interface HundredGigE0/0/0/24
 mtu 4110
  vrf blue
  ipv4 mtu 4096
  ipv4 address 10.1.1.10 255.255.255.0
  ipv6 mtu 4096
  ipv6 address fe80::7ae7:e8ff:fed3:20c0 link-local
  !
 !
!
Router#
Router#show ip int br | i HundredGigE0/0/0/24
HundredGigE0/0/0/24 10.1.1.10 Up Up blue
```

The output indicates that the interface acts as a regular Linux interface, and IOS XR configuration receives inputs from Linux.

Configure traffic protection for Linux networking

Configure Linux firewall rules from IOS XR to restrict traffic reaching Linux applications, allowing you to permit or deny traffic based on protocol, port, and ingress interface.

Use this procedure to configure traffic protection rules in IOS XR to control which traffic reaches Linux applications. Traffic protection provides a mechanism to configure Linux firewalls using IOS XR configuration, and applies rules based on any combination of remote address, local address, and ingress interface.

Any combination of remote address, local address, and ingress interface can be specified as rules to either allow or deny traffic. At least one parameter must be specified for a traffic protection rule to be valid.

> **Note**
>
> Do not use both native Linux firewalls and IOS XR Linux traffic protection simultaneously. If traffic is received on a protocol or port combination with no traffic protection rules configured, all traffic is allowed by default.

Procedure

1. Configure the traffic protection rule. The following example configures a rule that denies all TCP traffic on port 999 except traffic arriving on interface `HundredGigE0/0/0/25`.

   Example

   ```
   Router(config)#linux networking vrf default address-family ipv4 protection
   protocol
   tcp local-port 999 default-action deny permit hundredgigE0/0/0/25
   Router(config)#commit
   ```

   Key parameters in this command:

   - address-family: Configuration for a particular IPv4 or IPv6 address family.
   - protection: Configure traffic protection for Linux networking.
   - protocol: Select the supported protocol – TCP or UDP.
   - local-port: L4 port number (1-65535 or `all`) for the traffic protection rule.
   - default-action deny: Drop all packets for this service unless a permit rule applies.
   - permit: Allow packets to reach the Linux application for the specified interface or address.

2. Verify that the traffic protection rule is applied.

   Example

   ```
   Router#show run linux networking
   linux networking
    vrf default
     address-family ipv4
      protection
       protocol tcp local-port 999 default-action deny
        permit interface HundredGigE0/0/0/25
        !
       !
      !
    !
   ```

   The running configuration confirms that the traffic protection rule is active. TCP traffic on port 999 is denied by default, and traffic arriving on interface `HundredGigE0/0/0/25` is permitted.

The traffic protection rule is configured and active. Linux applications receive only the traffic that is permitted by the defined rules, and all other traffic on the specified port is denied.

Synchronize statistics between IOS XR and Linux

Configure IOS XR to periodically synchronize interface packet statistics from IOS XR to the Linux kernel so that Linux counters reflect the full traffic flow on the interface.

Use this procedure to configure statistics synchronization between IOS XR and Linux. The packet and byte counters maintained by Linux for IOS XR interfaces display only the traffic sourced in Linux by default. Configuring periodic synchronization pushes IOS XR interface statistics to the Linux kernel, providing a complete view of interface traffic from Linux.

Procedure

1. Configure the statistics synchronization direction and interval in global configuration or exposed-interface configuration.

   Example

   The following example configures statistics synchronization globally:

   ```
   Router(config)#linux networking statistics-synchronization from-xr every 30s
   ```

   The following example configures statistics synchronization for a specific exposed interface:

   ```
   Router(config)#linux networking exposed-interfaces interface
   bundle-ether 1 statistics-synchronization from-xr every 10s
   ```

   Key parameters:

   - from-xr: Pushes interface packet statistics from IOS XR to the Linux kernel.
   - every: Synchronization interval. Global configuration supports `30s` and `60s`. Exposed-interface configuration supports `5s`, `10s`, `30s`, and `60s`.

2. Verify that the statistics synchronization configuration is applied.

   Example

   ```
   Router#show run linux networking
   linux networking
    vrf default
     address-family ipv4
      protection
      protocol tcp local-port all default-action deny
       permit interface bundle-ether 1
       !
      !
     !
    !
   exposed-interfaces
    interface bundle-ether 1 linux-managed
     statistics-synchronization from-xr every 10s
     !
    !
   !
   ```

The running configuration confirms that statistics synchronization is active for the specified interface at the configured interval.

For troubleshooting purposes, use the **show tech-support linux networking** command to display debugging information.

IOS XR interface statistics are periodically synchronized to the Linux kernel. Linux counters now reflect the full traffic flow on the interface, not only traffic sourced in Linux.

CPU-based packet generators

The CPU-based packet generator is a built-in Linux application that generates diverse traffic streams directly within the production environment, enabling in-production network diagnostics without physically isolating routers.

A CPU-based packet generator is a Linux application that

- generates a wide range of traffic streams for network diagnostics in production environments
- allows direct in-production diagnosis without physically isolating routers, and
- integrates with IOS XR interfaces to send packets using the Packet I/O infrastructure.

Table 11: Feature History

Feature Name	Release Information	Feature Description
CPU-Based Packet Generator	Release 25.4.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select vonlyariants *) *This feature is supported on: • 8011-32Y8L2H2FH • 8011-12G12X4Y-A/D
CPU-Based Packet Generator	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) This feature is supported on Cisco 8011-4G24Y4H-I routers.
CPU-Based Packet Generator	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is supported on: 8212-48FH-M, 8711-32FH-M, 8712-MOD-M, 88-LC1-36EH, 88-LC1-12TH24FH-E, 88-LC1-52Y8H-EM.

Feature Name	Release Information	Feature Description
CPU-Based Packet Generator	Release 24.2.1	Use a CPU-based packet generator for IOS XR routers to simplify diagnostics without physically isolating routers. The feature introduces the **packetgen** command with different options to generate different types of packets.

Benefits

- Versatile traffic crafting: Create complex nested packets, such as IPinIPinIPinIP, to test and diagnose various scenarios.
- In-production diagnosis: Diagnose routers in a problem state directly, without disrupting the network setup.

Capabilities

- Support for diverse packet types: The CPU-based packet generator supports ARP, TCP, UDP, GRE, MPLS, IPinIP, ICMPv4, and ICMPv6 packet types.
- Corrupt or error packet generation: Can intentionally create packets with known errors for debugging, such as IPv4 packets with TTL 0, incorrect checksum, or mismatched IP option length fields.

Restrictions for CPU-based packet generators

Outline the key restrictions of CPU-based packet generators so you can understand their performance limitations and potential impact on router operation.

You must follow these restrictions for CPU-based packet generators:

- CPU-based packet generators are not optimized for high-speed packet processing and may not match the performance of NPU-based packet generators.
- CPU-based packet generators can introduce higher CPU loads during operation, which may affect router performance.
- The probe packet rate is limited to 40 kpps.

Topology of the CPU-based packet generator

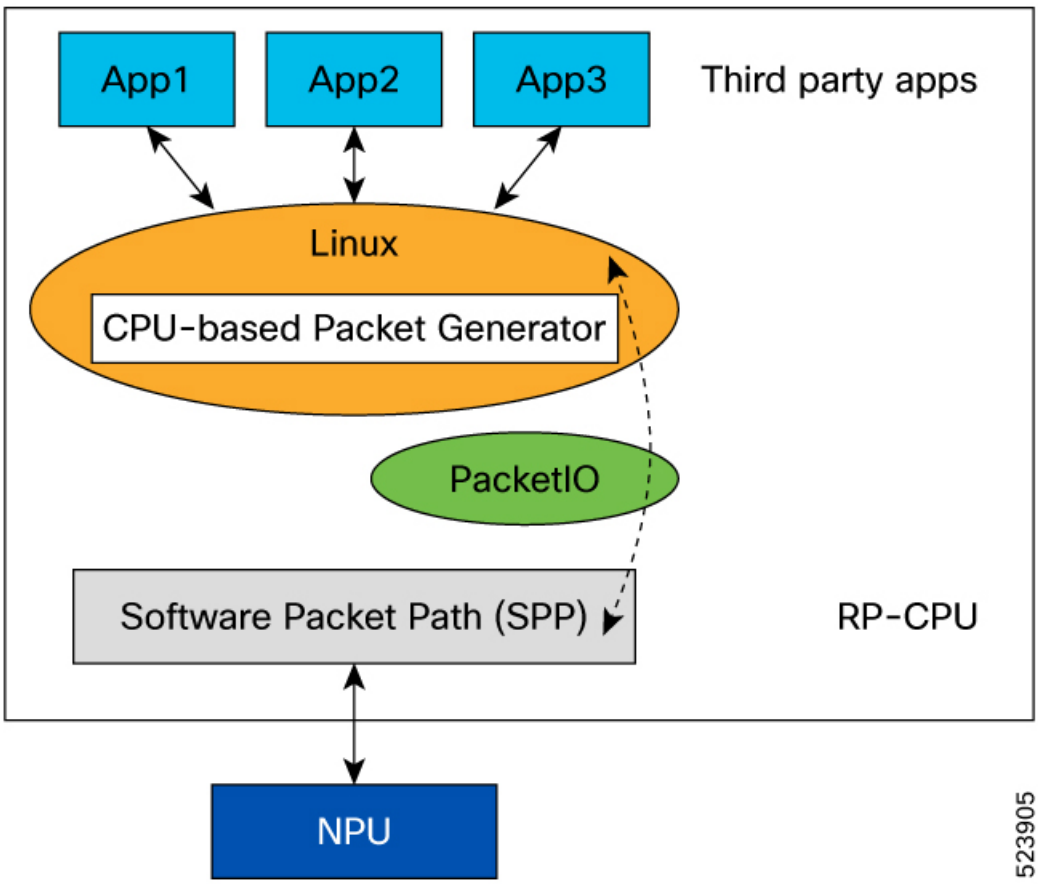
Provides details on the infrastructure and operation of the CPU-based packet generator within the Cisco IOS XR platform.

The CPU-based packet generator leverages the Cisco IOS XR Packet I/O framework, which serves as a host for third-party applications on the XR platform. The Packet I/O infrastructure facilitates packet transport and manages interactions between the Linux environment and IOS XR, enabling seamless communication between processes.

Implemented as a Linux application packaged within the IOS XR platform base image, the CPU-based packet generator is distributed as part of the platform, ensuring consistent deployment and integration.

The Linux infrastructure maintains a database of all IOS XR interfaces, including bundle interfaces. The CPU-based packet generator uses this database to send a specific packet type over a chosen interface, providing flexibility for testing and diagnostics.

Figure 3: Topology of the CPU-based packet generator



Packetgen CLI options and sample commands

The **packetgen** command supports multiple options to generate specific packet types and control injection behavior from the IOS XR bash environment.

Use the **packetgen** command with the following options from the XR bash environment. You can also replay a pcap file using:

```
packetgen -i interface_name -pcap pcap_file
```

Table 12: Packetgen CLI Options

Option	Description
`-accounting`	Turn on accounting for packets. Only works if packets come back to the packet generator.
`-arp-destination-hw-address string`	ARP target hardware address (default: uses interface MAC address).
`-arp-destination-ip-address string`	ARP target IP address (default: 127.0.0.1 or ::1).
`-arp-operation uint`	ARP operation (1: request, 2: reply, 3: rarp).
`-arp-source-hw-address string`	ARP sender hardware address (default: uses interface MAC).

Option	Description
`-arp-source-ip-address string`	ARP sender IP address (default: uses interface IP).
`-burst int`	Number of packets to inject at a time. Use with `-sleep`.
`-count int`	Number of packets to generate.
`-data-type string`	Payload type: constant, incrementing, random (default: no payload).
`-ethernet-dmac string`	Destination MAC address (default: ff:ff:ff:ff:ff:ff).
`-ethernet-smac string`	Source MAC address (default: interface MAC address).
`-file string`	Write packets to file.
`-gre`	Enable GRE.
`-gre-checksum-present`	Enable GRE checksum present bit.
`-gre-key-present`	Enable GRE key present bit.
`-gre-over-mpls`	Enable GRE over MPLS.
`-gre-protocol uint`	Set the protocol type of the GRE payload (default: 0x0800 (IP)).
`-gre-seq-present`	Enable GRE sequence number present bit.
`-gre-version uint`	Set the GRE version number (default: 0).
`-header string`	Custom header for all packets.
`-hex`	Print hex dump of packets.
`-i string`	Interface name for packet injection.
`-icmp-code uint`	ICMP code (default: 0).
`-icmp-type uint`	ICMP type (default: 0).
`-inc-dmac`	Increment destination MAC.
`-inc-smac`	Increment source MAC.
`-inner-ethernet-dmac string`	Inner Ethernet destination MAC address (default: ff:ff:ff:ff:ff:ff).
`-inner-ethernet-smac string`	Inner Ethernet source MAC address (default: ff:ff:ff:ff:ff:ff).
`-inner-ip-checksum uint`	Inner IP checksum (default: compute automatically).
`-inner-ip-dont-fragment uint`	Set inner IP Don't Fragment flag as 1.

Option	Description
`-inner-ip-dst string`	Inner destination IP address (default: 127.0.0.1 or ::1).
`-inner-ip-flow-label uint`	Inner IPv6 Flow Label value (default: 0).
`-inner-ip-frag-offset uint`	Inner IP fragment offset in units of 64-bits (for example, 1 = 64 bits).
`-inner-ip-protocol string`	Inner IP protocol. Supports protocol text (TCP, UDP) and code (63 for TCP) (default: TCP).
`-inner-ip-src string`	Inner source IP address (default: 127.0.0.1 or ::1).
`-inner-ip-tos uint`	Inner IP Type Of Service (TOS) value (default: 0).
`-inner-ip-traffic-class uint`	Inner IP traffic-class value (default: 0).
`-inner-ip-ttl uint`	Inner IP time to live (TTL) (default: 64).
`-inner-ip-version int`	Inner IP version (default: 4).
`-inner-vlan-id uint`	Inner VLAN ID (default: 0).
`-inner-vlan-tpid uint`	Inner VLAN Ethernet type (default: 33024, Dot1Q).
`-inner-vlan-vpri uint`	Inner VLAN priority (default: 0).
`-ip-checksum string`	IP checksum (default: compute automatically).
`-ip-dont-fragment string`	Set IP flag: 0 → 000 (nothing set), 1 → 001 (More Fragments), 2 → 010 (Don't Fragment), 4 → 100 (reserved bit).
`-ip-dst string`	Destination IP address (default: 127.0.0.1 or ::1).
`-ip-flow-label string`	IPv6 Flow Label value (default: 0).
`-ip-frag-offset string`	Fragment offset in units of 64-bits (1 = 64 bits).
`-ip-protocol string`	IP protocol. Supports protocol text (TCP, UDP, GRE, VXLAN, ICMP, NDP) and code (default: TCP).
`-ip-src string`	Source IP address (default: interface IP).
`-ip-tos string`	IP Type Of Service value (default: 0).

Option	Description
`-ip-traffic-class string`	IP traffic-class value (default: 0).
`-ip-ttl string`	IP time to live (TTL) (default: 64).
`-ip-version string`	IP version. Set for accurate IP packet creation (default: 4).
`-mpls-exp string`	Comma-separated MPLS EXP (Experimental) value (default: 0).
`-mpls-label string`	Comma-separated list of MPLS labels to add to the packet, from top to bottom.
`-mpls-ttl string`	Comma-separated MPLS TTL value (default: 64).
`-ndp string`	Specify the Neighbor Discovery Protocol: `nbr-solicit` or `nbr-advt`.
`-ndp-target-address string`	NDP target address (default: for advertisement, source IP; for solicitation, destination IP).
`-pcap string`	File to replay pcap.
`-progress`	Display a progress bar.
`-seed int`	Seed for pseudo-random payload generator.
`-size int`	Size of payload.
`-sleep string`	Time duration to sleep during each burst. Use with `-burst`.
`-stdout`	Print packets to stdout.
`-tcp-dport int`	TCP destination port (default: 40000).
`-tcp-flags string`	Set TCP control flags: U (Urgent), A (Acknowledgement), P (Push), R (Reset), S (Synchronize), F (Finish).
`-tcp-sport int`	TCP source port (default: 40000).
`-udp-dport int`	UDP destination port (default: 40000).
`-udp-sport int`	UDP source port (default: 40000).
`-vlan-id uint`	VLAN ID (default: 0).
`-vlan-tpid uint`	VLAN Ethernet type (default: 33024, Dot1Q).
`-vlan-vpri uint`	VLAN priority (default: 0).
`-vxlan-udp-dport int`	UDP destination port for VXLAN (default: 4789).
`-vxlan-udp-sport int`	UDP source port for VXLAN (default: 0).
`-vxlan-vni uint`	VXLAN VNI (default: 0).

Table 13: Sample Packetgen Commands

Packet Type	Sample Command
ARP	`packetgen -i enp0s8 -ip-ttl 32 -arp-operation 1 -progress -count 10000 -inc-smac -arp-destination-ip-address 192.168.56.1`
TCP	`packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac`
UDP	`packetgen -i enp0s8 -ip-ttl 32 -udp-sport 40000 -progress -count 10000 -inc-smac`
ICMP - PING	`packetgen -i enp0s8 -ip-ttl 32 -icmp-type 8 -progress -count 10000 -ip-dst 192.168.56.1`
GRE	`packetgen -i enp0s8 -ip-ttl 32 -gre -count 100 -inner-ip-ttl 32 -tcp-sport 3222 -progress`
IP in IP	`packetgen -i enp0s8 -count 100 -tcp-sport 3222 -progress -ip-src="1.1.1.1,2.2.2.2"`
ETHER-IP	`packetgen -i enp0s8 -ip-ttl 32 -count 100 -inner-ip-version 6 -tcp-sport 3222 -progress -inner-ethernet-smac ff:ff:ff:ff:ff:ff`
VLAN	`packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac -vlan-id 2`
QinQ	`packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac -vlan-id 2 -inner-vlan-id 2`
VXLAN	`packetgen -i enp0s8 -ip-ttl 32 -tcp-sport 40000 -progress -count 10000 -inc-smac -vxlan-vni 3 -vxlan-udp-sport 4444 -inner-ip-version 4 -inner-ethernet-smac ff:ff:ff:ff:ff:ff -data-type constant`
NDP	`packetgen -i enp0s8 -ip-version 6 -ndp nbr-advt -count 100 -ip-checksum 1 -progress`
MPLS	`packetgen -i enp0s8 -ip-version 4 -mpls-label 1,2,3,4,5 -tcp-sport 4556 -count 1000 -progress`

Generate packets using the CPU-based packet generator

Use the **packetgen** command from the IOS XR bash shell to generate and inject packets for in-production network diagnostics.

Run the CPU-based packet generator from the IOS XR bash environment to inject packets over IOS XR interfaces for network diagnostics.

The **packetgen** command enables you to generate a wide range of packet types directly on a router without physically isolating it from the production network. IOS XR interfaces appear as Linux interfaces in the bash environment, allowing direct use of interface names.

Before you begin

Review the CLI options and sample commands. For details, see Packetgen CLI options and sample commands.

Follow these steps to generate packets and verify injection through the CPU-based packet generator.

Procedure

1. Access the bash shell on the router.

   Example

   ```
   Router#bash
   [ios:~]$
   ```

2. Run **packetgen** with the required options to generate packets. This example sends 50 ICMP ping requests from source address `10.0.0.1` to destination address `10.0.0.2` over interface `Hu0_0_0_25`.

   Example

   ```
   [ios:~]$packetgen -i Hu0_0_0_25 -ip-ttl 32 -progress -count 50 -icmp-type 8
   -ip-dst 10.0.0.2 -ip-src 10.0.0.1 --ethernet-smac 78:c5:51:84:48:c4
   --ethernet-dmac 00:00:00:1e:ca:fc
   INFO[0000] [ETH IP ICMP]
   INFO[0000] Setting SRC IP to 10.0.0.1
   INFO[0000] Setting DST IP to 10.0.0.2
   INFO[0000] Opening Handle Hu0_0_0_25
   INFO[0000] Opened Handle Hu0_0_0_25
   INFO[0000] Starting Packet Injection
   Sending Packets... 2% | | (1/50, 254 packet/s) [0s:0s] /* Truncated output.
   */

   Address Age Hardware Addr State Type Interface
   10.0.0.1 - 78c5.5184.48c4
   Interface ARPA HundredGigE0/0/0/25
   10.0.0.2 00:50:23 0000.001e.cafc Dynamic ARPA HundredGigE0/0/0/25

   Source stats:
   Stat Name      Port Name            Control Packet Tx.  Control Packet Rx.
   Ping Reply Tx.
   20.0.0.2/
   Card01/Port01  Ethernet - VM - 001  51                  51
   50

   Interface stats:
   Input    Punt XIPC  InputQ     XIPC         PuntQ
   ClientID Drop/Total Drop/Total Cur/High/Max Cur/High/Max
   ipv6_icmp 0/0        0/0       0/0/1000     0/0/1000
   icmp      0/50       0/0       0/15/1000    0/0/1000
   ```

3. Verify packet delivery by reviewing the interface statistics in the output.

   The statistics confirm that 50 ICMP packets were injected and received on interface `Hu0_0_0_25`, verifying successful packet injection through the Packet I/O infrastructure.

The **packetgen** command injects specified packets over the selected IOS XR interface. Using interface statistics output, you can verify packet delivery and diagnose forwarding issues.