gRPC Fundamentals

gRPC operations

gRPC operations are a set of remote procedure calls that enable clients to interact with Cisco IOS XR devices for configuration and operational data retrieval.

- They support configuration retrieval and modification.
- They provide access to operational and model data.
- They allow CLI-based and structured output retrieval.

These operations are essential for automating and managing network devices programmatically using gRPC clients.

Manageability service gRPC operations

This table defines the manageability service gRPC operations for Cisco IOS XR.

gRPC Operation	Description
GetConfig	Retrieves the configuration from the router.
GetModels	Gets the supported Yang models on the router
MergeConfig	Merges the input config with the existing device configuration.
DeleteConfig	Deletes one or more subtrees or leaves of configuration.
ReplaceConfig	Replaces part of the existing configuration with the input configuration.
CommitReplace	Replaces all existing configuration with the new configuration provided.

gRPC Operation	Description
GetOper	Retrieves operational data.
CliConfig	Invokes the input CLI configuration.
ShowCmdTextOutput	Returns the output of a show command in the text form
ShowCmdJSONOutput	Returns the output of a show command in JSON form.

gRPC operation to Get configuration

The gRPC example shows how a gRPC GetConfig request works for feature.

The client initiates a message to get the current configuration of running on the router. The router responds with the current configuration.

gRPC request (Client to Router)	gRPC response (Router to Client)

rpc GetConfig
{
"Cisco-IOS-XR-cdp-cfg:cdp": [
"cdp": "running-configuration"
]
}
 	
rpc GetConfig
{
"Cisco-IOS-XR-ethernet-lldp-cfg:lldp": [
"lldp": "running-configuration"
]
}
	{
	"Cisco-IOS-XR-cdp-cfg:cdp": {
	"timer": 50,
	"enable": true,
	"log-adjacency": [
	null
	],
	"hold-time": 180,
	"advertise-v1-only": [
	null
	]
	}
	}
	
	{
	"Cisco-IOS-XR-ethernet-lldp-cfg:lldp": {
	"timer": 60,
	"enable": true,
	"reinit": 3,
	"holdtime": 150
	}
	}

gRPC over UNIX domain sockets

gRPC over UNIX domain sockets is a method that allows establishing gRPC connections using local containers without the need for password rotations.

- Extends gRPC TCP-based connections to UNIX domain sockets for local communication.
- Eliminates the need for username/password authentication for local containers.
- Improves security and control using UNIX file permissions.

This method enhances inter-process communication and simplifies secure local access to gRPC services on Cisco routers.

Feature History Table

gRPC server initialization and service registration

When gRPC is configured on the router, the gRPC server starts and then registers services such as gNMI and gNOI. After all the gRPC server registrations are complete, the listening socket is opened to listen to incoming gRPC connection requests. Currently, a TCP listen socket is created with the IP address, VRF, or gRPC listening port.

UNIX domain sockets and dual socket listening for gNMI

With this feature, the gRPC server listens over UNIX domain sockets that must be accessible from within the container through a local connection by default. With the UNIX socket enabled, the server listens on both TCP and UNIX sockets. However, if the UNIX socket is disabled, the server listens only on the TCP socket. The socket file is located at the directory.

Configure gRPC over UNIX domain sockets

You can use local containers and scripts on the router to establish gRPC connections over UNIX domain sockets.

Before you begin

Ensure that the router supports gRPC and that you have access to the CLI in configuration mode.

Procedure

Step 1 Configure the gRPC server

Example:

```
Router(config)#grpc
Router(config-grpc)#local-connection
Router(config-grpc)#commit
```

To disable the UNIX socket use the **no** form of the command.

```
Router(config-grpc)#no local-connection
```

The gRPC server restarts after you enable or disable the UNIX socket. If you disable the socket, any active gRPC sessions are dropped and the gRPC data store is reset.

The scale of gRPC requests remains the same and is split between the TCP and Unix socket connections. The maximum session limit is 256. If you utilize the 256 sessions on Unix sockets, further connections on either TCP or UNIX sockets are rejected.

Step 2 Verify that the local-connection is successfully enabled

Example:

```
Router#show grpc status
```

```
Thu Nov 25 16:51:30.382 UTC
*************************show gRPC status**********************
----------------------------------------------------------------
transport : grpc
access-family : tcp4
TLS : enabled
trustpoint :
listening-port : 57400
local-connection : enabled
max-request-per-user : 10
max-request-total : 128
max-streams : 32
max-streams-per-user : 32
vrf-socket-ns-path : global-vrf
min-client-keepalive-interval : 300
________________________________________________________________
```

A gRPC client must dial into the socket to send connection requests.

Here is an example of a Go client connecting to a UNIX socket.

```
const sockAddr =
  " /misc/app_host/ems/grpc.sock" // for ncs_5500
  " /var/lib/docker/ems/grpc.sock" // for cisco8000
...

func UnixConnect(addr string, t time.Duration) (net.Conn, error) {
  unix_addr, err := net.ResolveUnixAddr("unix", sockAddr)
  conn, err := net.DialUnix("unix", nil, unix_addr)
  return conn, err
}

func main() {
  ...
  opts = append(opts, grpc.WithTimeout(time.Second*time.Duration(*operTimeout)))
  opts = append(opts, grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(math.MaxInt32)))
  ...
  opts = append(opts, grpc.WithDialer(UnixConnect))
  conn, err := grpc.Dial(sockAddr, opts...)
  ...
}
```

The gRPC server is configured to accept connections over UNIX domain sockets, and clients can connect using the specified socket path.

What to do next

Monitor the gRPC sessions and ensure that the session count does not exceed the maximum limit of 256.

IANA port numbers for gRPC services

IANA port numbers for gRPC services is a configuration capability that

- enables assignment of standardized ports to gRPC-based services like gNMI, gRIBI, and P4RT
- supports both IANA-assigned and user-defined port numbers for service-level customization

- improves service isolation, routing accuracy, and protocol compliance across IPv4 and IPv6 networks.

This feature ensures secure and efficient communication by allowing precise control over port assignments for each gRPC service.

Table 1: Feature History Table

Feature Name	Release Information	Description
IANA Port Numbers For gRPC Services	Release 24.1.1	You can now efficiently manage and customize port assignments for gNMI, gRIBI, and P4RT services without port conflicts. This is possible because Cisco IOS XR now supports the Internet Assigned Numbers Authority (IANA)-assigned specific ports for P4RT (Port 9559), gRIBI (Port 9340), and gNMI (Port 9339). You can now use both IANA-assigned and user-specified ports for these gRPC services across any specified IPv4 or IPv6 addresses. As part of this support, a new submode for gNMI in gRPC is introduced. **CLI:** • **port (gRPC)** • **gnmi**

IANA port allocation and service identification

The Internet Assigned Numbers Authority (IANA) port numbers for gRPC services is a crucial aspect of network management that manages the allocation of port numbers for various protocols. These port numbers help in distinguishing different services on a network. Service names and port numbers are used to distinguish between different services that run over transport protocols such as TCP, UDP, DCCP, and SCTP. Port numbers are allocated in:

- System ports (0–1023)
- User ports (1024–49151)
- Dynamic or private ports (49152–65535)

Earlier, the gRPC server configuration on IOS-XR allowed a usable port range of 10000–57999, with a default listening port of 57400 and all services registered to the gRPC server utilized this port for connectivity. Service-based filtering of requests on any of the ports was unavailable. Hence, the request for a specific service sent on a port designated to another service (for example, gRIBI request on gNMI port) was accepted.

Service-level port customization

From Cisco IOS XR release 24.1.1, a new submode for gNMI is introduced in the configuration model to allow for service-level port customization. The existing gRPC configuration model includes submodes for P4RT and gRIBI. This submode enables you to configure specific ports for gNMI, gRIBI, and P4RT services independently. You can configure gNMI, gRIBI, and P4RT services using the gRPC submode command to

set the default port for each service. The **port** command under service submode, allows you to modify the port as needed, while adhering to the defined port range.

Disabling the **port** command causes the service to use the default or IANA port.

Service port customization

You can set custom ports for gNMI, gRIBI, and P4RT services within the defined range, including default IANA ports like 9339, 9340, and 9559 (respectively). The gRPC service will continue to maintain its default port within the specified range (57344–57999). Any changes made to the gRPC default port do not impact the service port configurations for gNMI, gRIBI, and P4RT. Requests which are sent on a port designated for a specific service (example, gRIBI request on gNMI port) are accepted. This flexibility allows for seamless communication across different service ports and the general gRPC port.

Multiple gRPC servers

The Extensible Manageability Services daemon (EMSd) is a mechanism that

- enables support for both OpenConfig and proprietary gRPC services
- handles network device management services, particularly streaming telemetry and gRPC, and
- provides the infrastructure to exchange configuration and operational data between a client and the network device using standardized data models.

Table 2: Feature History Table

EMSd can now host up to 16 independently configurable gRPC servers within the same process. Each server can have its own services, ports, TLS profiles, VRF assignments, DSCP settings, and listener types such as TCP or UDS. Earlier, the EMSd supported only one gRPC server. All services—OpenConfig, proprietary services, and various telemetry protocols—had to coexist on this single server, which limited flexibility. You could not isolate services with different security needs, VRFs, DSCP values, or listener types. Incompatible services could not be separated cleanly.

Benefits of multiple gRPC servers

Multiple gRPC servers within a single EMSd process provide significant benefits. This setup enables highly flexible and isolated configurations.

- Tailored service grouping: Configure specific sets of gRPC services on individual servers, allowing flexible grouping based on operational needs or compatibility rules.
- Independent network contexts: Assign each gRPC server to a distinct virtual routing and forwarding (VRF) instance, enabling simultaneous in-band paths and out-of-band communication paths.
- Granular QoS and security: Apply unique DSCP values and separate SSL profiles to each server. This enables fine-tuned traffic prioritization and robust security management.
- Flexible client connectivity: You can support diverse client requirements by configuring independent listening addresses, ports, and transport types (TCP or Unix Domain Sockets) for each server.
- Enhanced operational isolation: Isolate services, security, and network contexts on distinct gRPC servers and manage them in a single EMSd process to meet deployment needs.

Best practices for deploying multiple gRPC servers

Strategic Server Creation: Create a dedicated gRPC server in these situations:

- Services are incompatible: Separate services that cannot coexist on the same server (such as gNMI and P4RT).
- Security requirements differ: If different client groups or applications need distinct SSL profiles, certificate chains, or mutual TLS configurations, create a separate server.
- Network contexts vary: If services need to operate in different Virtual Routing and Forwarding (VRF) instances (such as in-band and out-of-band management), use separate servers.
- QoS prioritization is required: If different DSCP values are necessary for traffic prioritization, apply DSCP settings at the server level.
- Client connectivity needs differ: When varied client connection types or unique listening addresses or ports are required, create a separate server.
- Group compatible services: To improve efficiency and management, group compatible gRPC services that share operational requirements onto a single server.
- Plan for scalability: When planning your deployment, keep in mind that you can configure up to 16 gRPC servers.
- Migration from legacy: When transitioning from a single default gRPC server, use the disable-default-server option to prevent its creation and adopt the named server configuration model.
- Naming convention: Do not use `DEFAULT` as a name for any custom gRPC server because it is reserved for the legacy default server.

Restrictions for deploying multiple gRPC servers

- Maximum server count: You can configure up to 16 gRPC servers within a single EMSd process.
- Incompatible service combinations: These gRPC services cannot be configured together on the same server:
  - gNMI, gNOI, and cNMI
  - gRIBI, SL-API, and SR
  - P4RT and gNPSI
- Deprecated configurations: Do not use the legacy default gRPC server configuration that registers all services. Define services explicitly for each new server configuration.
- Deprecated commands: Do not use the following deprecated commands:
  - **config grpc gribi port**
  - **config grpc gnmi port**
  - **config grpc p4rt port**

- Default server naming: Starting with Cisco IOS XR Release 25.4.1, do not use the **config grpc name** command for the default server because it performs no operation.

Configure multiple gRPC servers

Starting with Cisco IOS XR Release 25.4.1, you can deploy and manage multiple gRPC servers within a single EMSd process. This feature enables these capabilities:

- You have granular control and isolation for each gRPC server.
- You can independently configure listening ports, TLS profiles, network instances (VRFs), and DSCP values.
- Enhanced operational flexibility and security for telemetry services.

The primary configuration attributes that can be customized include:

- Services: Select which telemetry or gRPC services each server supports.
- Listener types: Configure TCP, TLS, or other listener types per server.
- Network contexts: Assign servers to specific VRFs or network instances.
- Ports: Define listening ports individually per server.
- Security: Specify distinct TLS profiles and authentication settings per server.

This capability supports advanced deployment scenarios and improves maintainability compared to single-server models.

To configure multiple gRPC servers, follow these configuration tasks:

- Configure a new gRPC server with compatible services
- Configure a new gRPC server with incompatible services
- Disable the default gRPC server for migration
- Configure UDS and TCP listeners for gRPC servers
- Configure only UDS listeners for gRPC servers

Configure a new gRPC server with compatible services

Configure a new gRPC server by defining it under the `grpc` configuration mode, specifying its services, and assigning a listening port. You can create multiple named servers, each tailored to specific operational requirements.

Use this task to add a new named gRPC server. You can specify compatible services and the port. The default server remains available.

Use these steps to configure a new gRPC server with compatible services.

Procedure

Step 1 Enter configuration mode for gRPC.

Example:

```
Router#config
Router(config)#grpc
```

Step 2 Create new named servers and add services.

You can specify compatible services and the port for each server. The system allows for multiple distinct servers.

Example:

Configure `server1` and `server2`

```
ROUTER(config-grpc)#server server1
ROUTER(config-grpc-server)#services P4RT
ROUTER(config-grpc-server)#services gNPSI
ROUTER(config-grpc-server)#port 9559
ROUTER(config-grpc-server)#exit

ROUTER(config-grpc)#server server2
ROUTER(config-grpc-server)#services gNMI
ROUTER(config-grpc-server)#port 9340
ROUTER(config-grpc-server)#exit
```

This configuration assigns P4RT and gNPSI services to `server1` on port 9559 and assigns gNMI to `server2` on port 9340.

Example:

Configure `server3` with a specific VRF

To assign a gRPC server to a distinct Virtual Routing and Forwarding (VRF) instance, use the `network-instance` command within the server's configuration mode.

```
ROUTER(config-grpc)#server server3
ROUTER(config-grpc-server)#services gNMI
ROUTER(config-grpc-server)#port 9341
ROUTER(config-grpc-server)#vrf Mgmt-VRF
ROUTER(config-grpc-server)#exit
```

This example configures `server3` for gNMI on port 9341 within the `Mgmt-VRF` instance.

Step 3 Commit the configuration.

Example:

```
ROUTER(config-grpc)#commit
```

Step 4 Verify the configuration.

Example:

```
ROUTER(config-grpc)#show running-config grpc
```

Your device now hosts the configured gRPC servers. Each server is ready to receive compatible service requests on its specified port.

Configure a new gRPC server with incompatible services

The system does not allow you to configure incompatible gRPC services on a server.

Certain gRPC services cannot coexist on the same server due to functional conflicts. If you configure incompatible services, the system returns an error. Some examples of incompatible service combinations include the following:

- gNMI, gNOI, and cNMI
- gRIBI, SL-API, and SR
- P4RT and gNPSI

Use these steps to configure a new gRPC server with incompatible services.

Procedure

Step 1 Enter configuration mode for gRPC.

Example:

```
Router#config
ROUTER(config)#grpc
```

Step 2 Create a new server with incompatible services.

Example:

```
ROUTER(config-grpc)#server server-incompatible
ROUTER(config-grpc-server)#services P4RT
ROUTER(config-grpc-server)#services gNMI
```

You receive an error because P4RT and gNMI services cannot coexist. When you add a new service or service list in the OpenConfig model, it replaces the existing one. If the new list contains incompatible services, the system returns an error.

What to do next

Review the error details and remove incompatible services as needed to complete the configuration.

Disable the default gRPC server for migration

When you transition to a named server configuration model from a single default gRPC server, disable the automatic creation of the legacy default server. This action ensures only your configured gRPC servers are active.

Create a new server and remove the default server using these steps.

Procedure

Step 1 Enter configuration mode for gRPC.

Example:

```
ROUTER(config)#grpc
```

Step 2 Disable the default server.

Example:

```
ROUTER(config-grpc)#disable-default-server
```

Step 3 Create your new named server.

Example:

```
ROUTER(config-grpc)#server server1
ROUTER(config-grpc-server)#services gNMI
ROUTER(config-grpc-server)#port 9340
ROUTER(config-grpc-server)#exit
```

Step 4 Commit the configuration.

Example:

```
ROUTER(config-grpc-server)#commit
```

The system creates only `server1` (or any named servers you configure). The default gRPC server is not active.

Configure UDS and TCP listeners for gRPC servers

Configure gRPC servers to listen on Transmission Control Protocol (TCP) and Unix Domain Sockets (UDS), or only on UDS. This configuration supports a range of client connectivity requirements.

Enable both TCP and UDS listeners for gRPC servers using these steps.

Procedure

Step 1 Enter configuration mode for gRPC.

Example:

```
ROUTER(config)#grpc
```

Step 2 Enable global local connection (UDS).

Example:

```
ROUTER(config-grpc)#local-connection
```

Step 3 Create a server, or select an existing one, and configure the server's services and port.

Example:

```
ROUTER(config-grpc)#server server1
ROUTER(config-grpc-server)#services gNMI
ROUTER(config-grpc-server)#port 9340
```

Step 4 Enable local connection specifically for this server.

Example:

```
ROUTER(config-grpc-server)#local-connection
```

Step 5 Commit the configuration.

Example:

```
ROUTER(config-grpc-server)#commit
```

This configuration creates both TCP and UDS listeners for `server1`. The UDS listener for `server1` will use the socket at `/var/lib/docker/ems/server1`.

Configure only UDS listeners for gRPC servers

You configure a gRPC server to use only UDS.

Procedure

Step 1 Enter configuration mode for gRPC.

Example:

```
ROUTER(config)#grpc
```

Step 2 Enable global local connection (UDS).

Example:

```
ROUTER(config-grpc)#local-connection
```

Step 3 Disable global remote connection (TCP).

Example:

```
ROUTER(config-grpc)#remote-connection disable
```

Step 4 Create a server or select an existing server. Then configure the server's services and port.

Example:

```
ROUTER(config-grpc)#server server1
ROUTER(config-grpc-server)#services gNMI
ROUTER(config-grpc-server)#port 9340
```

Step 5 Enable local connection and disable remote connection specifically for this server.

Example:

```
ROUTER(config-grpc-server)#local-connection
ROUTER(config-grpc-server)#remote-connection disable
```

Step 6 Commit the configuration.

Example:

```
ROUTER(config-grpc-server)#commit
```

When you complete the configuration, server1 with the gNMI service becomes available. Only UDS listeners remain active.

Transceiver optics resets

Transceiver optics reset is a optics management operation that

- remotely power-cycles an optical transceiver module on a Cisco router
- restores normal operation or resolves faults without manual intervention, and
- allows execution using a single CLI command or remotely using gNOI RPC.

Table 3: Feature History Table

Feature Name	Release Information	Feature Description
Transceiver optics resets	Release 25.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100], 8010 [ASIC: A100]); Centralized Systems (8600 [ASIC: Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) You can now achieve rapid, automated recovery of optical modules, significantly reducing manual intervention and improving network uptime. With this feature, you can remotely power-cycle transceiver optics using a single gNOI Reboot RPC API call or CLI command, ensuring optimal performance and quick fault resolution at scale.

Restoring optimal operation in optical modules

Optical transceivers in modern routers, especially those operating at 400G and above, can experience degraded performance or errors due to their complex internal components. These issues may occasionally require resetting the optics modules to restore optimal operation. Providing the capability to reset optics modules with a single command or through network automation interfaces enables operators to troubleshoot and maintain transceivers.

Usage guidelines for optics reset

Preconditions

Do not attempt an optics reset using the CLI action or gNOI reboot if the transceiver is disabled from the config namespace, as the operation will fail.

Removal Procedure

- Use the Optics R/S/I/P naming convention to reload transceivers through the CLI.
- For gNOI-based optics reload, use the Speed R/S/I/P naming convention for grey optics and the Optics R/S/I/P naming convention for ZR/ZRP optics.

Request Scope

Submit a gNOI reboot request for only one optic at a time.

Transceiver optics reset using gNOI Reboot RPC

Remotely reset an optical transceiver module using the gNOI Reboot RPC API for automation or integration with network management systems.

Use this method for automated, large-scale, or remote operations where CLI access is not practical.

Before you begin

- Ensure the correct optics module is installed in the desired port.

Procedure

Step 1 Send the RebootRequest using your gNOI client or automation system to the router.

Example:

```
> python src/gnoi_DT_client.py reboot 172.26.228.212 9589 --user ***** --passwd ******
--slot=Optics0/0/0/6 --rebootmethod COLD --reboot_message "GNOI demo" --reboot_delay 1000000000
##############################
RPC to 172.26.228.212:9589
RPC start time: 10:27:59.755231
========Reboot Request=============
RPC start time: 10:27:59.755309
Reboot method set is  1
method: COLD
delay: 1000000000
message: "GNOI demo"
subcomponents {
  elem {
    name: "openconfig-platform"
  }
  elem {
    name: "components"
  }
  elem {
    name: "component"
    key {
      key: "name"
      value: "Optics0/0/0/6"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "location"
  }
}
```

```
False
========Reboot Response=============
RPC end time: 10:28:00.513444
```

Step 2 Use the gNOI RebootStatus RPC with the transceiver path/identifier to check the progress or result.

Example:

```
> python src/gnoi_DT_client.py rebootstatus 172.26.228.212 9589 --user ***** --passwd *****
--slot=Optics0/0/0/6
##############################
RPC to 172.26.228.212:9589
RPC start time: 10:30:51.100883
RPC start time: 10:30:51.100945
========Reboot-Status Request==============
subcomponents {
  elem {
    name: "openconfig-platform"
  }
  elem {
    name: "components"
  }
  elem {
    name: "component"
    key {
      key: "name"
      value: "Optics0/0/0/6"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "location"
  }
}

RPC end time: 10:30:51.819921
========Reboot-status Response=============
Active : False
Wait : 0
When : 1765256281641617310
Reason : GNOI demo
Count : 1
Method : COLD
Status : STATUS_SUCCESS
```

Transceiver optics reset using CLI

Restore normal operation or recover from a fault by power-cycling an optical transceiver module directly from the command line.

Use this task if a transceiver module is not functioning correctly or as part of standard troubleshooting procedures.

Procedure

Step 1 Identify the controller-name (*r/s/i/p*) of the transceiver you want to reset and enter the reset controller command.

Example:

```
Router#reload transceiver optics 0/0/0/4
```

Step 2 Verify the reload from the syslog.

Example:

```
RP/0/RP0/CPU0:Dec  9 04:58:01.644 UTC: optics_driver[349]: %L2-OPTICS-6-TRANS_RELOAD_STATUS :
Transceiver reload started for Optics0/0/0/6
RP/0/RP0/CPU0:Dec  9 04:58:01.644 UTC: optics_driver[349]: %L2-OPTICS-6-TRANSCEIVER_DISABLED : Optics
 module FourHundredGigE0/0/0/6 type QSFPDD 400G ZRP serial number 201800148 disabled
RP/0/RP0/CPU0:Dec  9 04:58:07.007 UTC: optics_driver[349]: %L2-OPTICS-6-TRANSCEIVER_ENABLED : Optics
 module FourHundredGigE0/0/0/6 type QSFPDD 400G ZRP serial number 201800148 enabled
RP/0/RP0/CPU0:Dec  9 04:58:07.054 UTC: optics_driver[349]: %L2-OPTICS-6-TRANS_RELOAD_STATUS :
Transceiver reload done for Optics0/0/0/6
```

gNOI RemoveContainer RPC

The RemoveContainer RPC is a remote procedure call that

- allows users to remove containers managed by App Manager through the Containerz workflow
- supports both standard and forced removal of containers, and
- integrates with the gNOI protocol for automation and remote operations.

Users can use this RPC to remove containers that are no longer needed or to automate container lifecycle management in integration with external systems.

Table 4: Feature History Table

Feature Name	Release Information	Feature Description
gNOI RemoveContainer RPC	Release 25.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100], 8010 [ASIC: A100]); Centralized Systems (8600 [ASIC: Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) We have introduced support for RemoveContainer RPC to give you direct, automated control over the removal of application containers managed by App Manager, allowing you to efficiently decommission unneeded or malfunctioning containers and maintain a clean, resource-optimized system.

Usage guidelines for removing containers

Identification and Selection

- Always specify the container name when removing containers; do not use image IDs or other identifiers.
- Always use the ListContainer RPC to retrieve the correct container name before performing a removal.

Removal Procedure

- Do not attempt to remove a running container unless you specify the force option; otherwise, removal will fail and the RPC returns an error.
- Always review the status codes and messages returned by the RemoveContainer RPC to diagnose removal issues.
- You cannot remove Docker images directly by image ID using the RemoveContainer RPC.

CLI guidelines

- The App Manager CLI has been updated to support additional sub-configuration and new sensitive host paths under the unified `docker` hierarchy.
- The `allow-sensitive-paths` CLI grants containers managed by the Containerz workflow access to sensitive host paths, including the Docker socket (for example, `/var/run/docker.sock` or `/misc/app_host/docker.sock`, depending on the platform), as well as other paths such as `/tmp` and `/var/run`.

- The `appmgr docker vrf <vrf-name>` CLI starts the Docker daemon in the specified non-default VRF, so previously running containers and any new containers managed by AppMgr/Containerz use that VRF for their network connectivity.

Containerz.proto restrictions

- Plugin support defined in the v0.6.1 `containerz.proto` is not available for this feature. Plugin-related fields and workflows are not supported.
- The `devices` field in `StartContainerRequest` is not supported. If you use this field, the RPC returns an error indicating this option is not available.

Configure RemoveContainer RPC

Remove a running or stopped container using the RemoveContainer RPC workflow.

Use this task to manage the lifecycle of third-party, partner, or customer application containers using the gNOI API.

Before you begin

- Ensure that the container is stopped before removing. If the container is running, set the force option to `true` to forcibly stop and remove it

Procedure

Step 1 Use the ListContainer RPC to verify the name and status of the target container.

Example:

```
[user@host ~]$ ContZ_Client --addr 192.168.122.254:57400 container list
ID      Name    Image           State   Labels  HASH
14d12   alp-1   alpine:3.20     RUNNING map[]   method:SHA256 hash:"..."
15e35   alp-2   alpine:3.20     RUNNING map[]   method:SHA256 hash:"..."
```

Step 2 Specify the container name using RemoveContainer RPC.

Example:

```
[user@host ~]$ ContZ_Client --addr 192.168.122.254:57400 container remove --instance alp-2 --force
Successfully removed alp-2
```

Step 3 Use ListContainer RPC to verify that the container is removed from the list.

Example:

```
[user@host ~]$ ContZ_Client --addr 192.168.122.254:57400 container list
ID      Name    Image           State   Labels  HASH
14d12   alp-1   alpine:3.20     RUNNING map[]   method:SHA256 hash:"..."
```

Configure Docker access for Containerz containers

Configure App Manager to grant containers managed by the Containerz workflow access to sensitive host paths, including the Docker socket, and optionally run Docker in a non-default VRF.

Use this task to enable access to host resources such as `/var/run/docker.sock`, `/misc/app_host/docker.sock`, `/tmp`, and `/var/run` for trusted containers, and to place the Docker daemon in a non-default VRF.

Procedure

Step 1 Enter App Manager docker configuration mode.

Example:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# appmgr
RP/0/RP0/CPU0:router(config-appmgr)# docker
```

Step 2 Enable access to sensitive host paths for Containerz-managed containers.

Example:

```
RP/0/RP0/CPU0:router(config-appmgr-docker)# allow-sensitive-paths
```

The `allow-sensitive-paths` CLI grants containers managed by the Containerz workflow access to sensitive host paths, including the Docker socket (for example, `/var/run/docker.sock` or `/misc/app_host/docker.sock`, depending on the platform), as well as other paths such as `/tmp` and `/var/run`.

Step 3 Optionally, start the Docker daemon in a non-default VRF.

Example:

```
RP/0/RP0/CPU0:router(config-appmgr-docker)# vrf <vrf-name>
```

The `appmgr docker vrf <vrf-name>` command starts the Docker daemon in the specified non-default VRF instead of the default global VRF. When this configuration is present, previously running containers and any new containers managed by AppMgr/Containerz use that VRF for their network connectivity.