



Use gRPC Protocol to Define Network Operations with Data Models

XR devices ship with the YANG files that define the data models they support. Using a management protocol such as NETCONF or gRPC, you can programmatically query a device for the list of models it supports and retrieve the model files.

gRPC is an open-source RPC framework. It is based on Protocol Buffers (Protobuf), which is an open source binary serialization protocol. gRPC provides a flexible, efficient, automated mechanism for serializing structured data, like XML, but is smaller and simpler to use. You define the structure using protocol buffer message types in `.proto` files. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs.

gRPC encodes requests and responses in binary. gRPC is extensible to other content types along with Protobuf. The Protobuf binary data object in gRPC is transported over HTTP/2.

gRPC supports distributed applications and services between a client and server. gRPC provides the infrastructure to build a device management service to exchange configuration and operational data between a client and a server. The structure of the data is defined by YANG models.



Note All 64-bit IOS XR platforms support gRPC and TCP protocols. All 32-bit IOS XR platforms support only TCP protocol.



Note Starting from Cisco IOS XR Release 25.2.1, you can disable TCP connections using the [gRPC remote-connection disable](#) command.

Cisco gRPC IDL uses the protocol buffers interface definition language (IDL) to define service methods, and define parameters and return types as protocol buffer message types. The gRPC requests are encoded and sent to the router using JSON. Clients can invoke the RPC calls defined in the IDL to program the router.

The following example shows the syntax of the proto file for a gRPC configuration:

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {
```

```

rpc GetConfig(ConfigGetArgs) returns(stream ConfigGetReply) {};

rpc MergeConfig(ConfigArgs) returns(ConfigReply) {};

rpc DeleteConfig(ConfigArgs) returns(ConfigReply) {};

rpc ReplaceConfig(ConfigArgs) returns(ConfigReply) {};

rpc CliConfig(CliConfigArgs) returns(CliConfigReply) {};

rpc GetOper(GetOperArgs) returns(stream GetOperReply) {};

rpc CommitReplace(CommitReplaceArgs) returns(CommitReplaceReply) {};

}

message ConfigGetArgs {
    int64 ReqId = 1;
    string yangpathjson = 2;
}

message ConfigGetReply {
    int64 ResReqId = 1;
    string yangjson = 2;
    string errors = 3;
}

message GetOperArgs {
    int64 ReqId = 1;
    string yangpathjson = 2;
}

message GetOperReply {
    int64 ResReqId = 1;
    string yangjson = 2;
    string errors = 3;
}

message ConfigArgs {
    int64 ReqId = 1;
    string yangjson = 2;
}

message ConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

message CliConfigArgs {
    int64 ReqId = 1;
    string cli = 2;
}

message CliConfigReply {
    int64 ResReqId = 1;
    string errors = 2;
}

message CommitReplaceArgs {
    int64 ReqId = 1;
    string cli = 2;
    string yangjson = 3;
}

message CommitReplaceReply {

```

```

        int64 ResReqId = 1;
        string errors = 2;
    }

```

Example for gRPCExec configuration:

```

service gRPCExec {
    rpc ShowCmdTextOutput(ShowCmdArgs) returns(stream ShowCmdTextReply) {};
    rpc ShowCmdJSONOutput(ShowCmdArgs) returns(stream ShowCmdJSONReply) {};
}

message ShowCmdArgs {
    int64 ReqId = 1;
    string cli = 2;
}

message ShowCmdTextReply {
    int64 ResReqId = 1;
    string output = 2;
    string errors = 3;
}

```

Example for OpenConfiggRPC configuration:

```

service OpenConfiggRPC {
    rpc SubscribeTelemetry(SubscribeRequest) returns (stream SubscribeResponse) {};
    rpc UnSubscribeTelemetry(CancelSubscribeReq) returns (SubscribeResponse) {};
    rpc GetModels(GetModelsInput) returns (GetModelsOutput) {};
}

message GetModelsInput {
    uint64 requestId = 1;
    string name = 2;
    string namespace = 3;
    string version = 4;
    enum MODLE_REQUEST_TYPE {
        SUMMARY = 0;
        DETAIL = 1;
    }
    MODLE_REQUEST_TYPE requestType = 5;
}

message GetModelsOutput {
    uint64 requestId = 1;
    message ModelInfo {
        string name = 1;
        string namespace = 2;
        string version = 3;
        GET_MODEL_TYPE modelType = 4;
        string modelData = 5;
    }
    repeated ModelInfo models = 2;
    OC_RPC_RESPONSE_TYPE responseCode = 3;
    string msg = 4;
}

```

This article describes, with a use case to configure interfaces on a router, how data models helps in a faster programmatic and standards-based configuration of a network, as compared to CLI.

- [gRPC Operations, on page 4](#)

- [gRPC over UNIX Domain Sockets](#), on page 15
- [gRPC Network Management Interface](#), on page 16
- [gRPC Network Operations Interface](#) , on page 34
- [gRPC Network Security Interface](#) , on page 47
- [Manage certificates using Certz.proto](#), on page 68
- [gNSI EnrollZ and AttestZ](#), on page 72
- [gNOI Containerz](#), on page 75
- [P4Runtime](#), on page 80
- [IANA Port Numbers For gRPC Services](#), on page 82
- [Configure Interfaces Using Data Models in a gRPC Session](#), on page 86

gRPC Operations

The following are the defined 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.
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

This example shows how a gRPC GetConfig request works for LLDP feature.

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

gRPC Request (Client to Router)	gRPC Response (Router to Client)
<pre>rpc GetConfig { "Cisco-IOS-XR-cdp-cfg:cdp": ["cdp": "running-configuration"] } rpc GetConfig { "Cisco-IOS-XR-ethernet-lldp-cfg:lldp": ["lldp": "running-configuration"] }</pre>	<pre>{ "Cisco-IOS-XR-cdp-cfg:cdp": { "timer": 50, "enable": true, "log-adjacency": [null], "hold-time": 180, "advertise-vl-only": [null] } } { "Cisco-IOS-XR-ethernet-lldp-cfg:lldp": { "timer": 60, "enable": true, "reinit": 3, "holdtime": 150 } }</pre>

gRPC Authentication Modes

gRPC supports the following authentication modes to secure communication between clients and servers. These authentication modes help ensure that only authorized entities can access the gRPC services, like gNOI, gRIBI, and P4RT. Upon receiving a gRPC request, the device will authenticate the user and perform various authorization checks to validate the user.

The following table lists the authentication type and configuration requirements:

Table 1: gRPC Authentication Modes and Configuration Requirements

Type	Authentication Method	Authorization Method	Configuration Requirement	Requirement From Client
Metadata with TLS	username, password	username	grpc	username, password, and CA
Metadata without TLS	username, password	username	grpc no-tls	username, password
Metadata with Mutual TLS	username, password	username	grpc tls-mutual	username, password, client certificate, client key, and CA
Certificate based Authentication	client certificate's common name field	username from client certificate's common name field	grpc tls-mutual and grpc certificate authentication	client certificate, client key, and CA

Certificate based Authentication

In Extensible Manageability Services (EMS) gRPC, the certificates play a vital role in ensuring secure and authenticated communication. The EMS gRPC utilizes the following certificates for authentication:

```
/misc/config/grpc/ems.pem
/misc/config/grpc/ems.key
/misc/config/grpc/ca.cert
```



Note For clients to use the certificates, ensure to copy the certificates from **/misc/config/grpc/**
The gRPC server advertises TLS 1.2 as the maximum TLS version when using a SUDI-based SSL profile.

Generation of Certificates

These certificates are typically generated using a Certificate Authority (CA) by the device. The EMS certificates, including the server certificate (**ems.pem**), public key (**ems.key**), and CA certificate (**ca.cert**), are generated with specific parameters like the common name **ems.cisco.com** to uniquely identify the EMS server and placed in the **/misc/config/grpc/** location.

The default certificates that are generated by the server are Server-only TLS certificates and by using these certificates you can authenticate the identity of the server.

Usage of Certificates

These certificates are used for enabling secure communication through Transport Layer Security (TLS) between gRPC clients and the EMS server. The client should use **ems.pem** and **ca.cert** to initiate the TLS authentication.

To update the certificates, ensure to copy the new certificates that has been generated earlier to the location and restart the server.

Custom Certificates

If you want to use your own certificates for EMS gRPC communication, then you can follow a workflow to generate a custom certificates with the required parameters and then configure the EMS server to use these custom certificates. This process involves replacing the default EMS certificates with the custom ones and ensuring that the gRPC clients also trust the custom CA certificate. For more information on how to customize the **common-name**, see *Certificate Common-Name For Dial-in Using gRPC Protocol*.

Authenticate gRPC Services



Note Typically, gRPC clients include the username and password in the gRPC metadata fields.

Procedure

Use any one of the following configuration type to authenticate any gRPC service.

- Metadata with TLS

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

- **Metadata without TLS**

```
Router#config
Router(config)#grpc
Router(config-grpc)#no-tls
Router(config-grpc)#commit
```

- **Metadata with Mutual TLS**

```
Router#config
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#commit
```

- **Certificate based Authentication**

```
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#certificate-authentication
Router(config-grpc)#commit
```

gRPC server TLS version 1.3 support

Table 2: Feature History Table

Feature Name	Release Information	Description
gRPC server TLS version 1.3 support	Release 25.1.1	<p>Introduced in this release on: Fixed Systems (8700 [ASIC: K100])</p> <p>This feature is now supported on:</p> <ul style="list-style-type: none"> • 8712-MOD-M

Feature Name	Release Information	Description
gRPC server TLS version 1.3 support	Release 24.4.1	<p>Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100]); Centralized Systems (8600 [ASIC: Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100])</p> <p>You can now enhance your network security by enabling TLS 1.3 support for your gRPC services. The TLS 1.3 version offers stronger protection against vulnerabilities, eliminates outdated ciphers, and ensures forward secrecy in data encryption by generating a unique session key for each new network session.</p> <p>The feature introduces these changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> • tls-min-version • tls-max-version

The gRPC Remote Procedure Calls (gRPC) server Transport Layer Security (TLS) version 1.3 support is a security feature that

- provides end-to-end communications security over networks,
- prevents unauthorized access and eavesdropping, and
- protects against tampering and message forgery.

The TLS private key is encrypted before being stored on the disk. For more details on SSL or TLS version certificates, keys, and communication parameters, see [Manage certificates using Certz.proto](#).

Configure gRPC TLS version

You can configure the minimum and maximum TLS versions for gRPC, and view the TLS cipher suites.

Procedure

Step 1 Configure gRPC TLS minimum, maximum, or both versions.

Example:

- Configure gRPC TLS minimum version.

```
Router# config
Router(config)# grpc
Router(config-grpc)# tls-min-version 1.0
Router(config-grpc)# commit
```

tls-min-version can be 1.0, 1.1, 1.2, or 1.3. The default minimum version for TLS is 1.0.

- Configure gRPC TLS maximum version.


```
Router# config
Router(config)# grpc
Router(config-grpc)# tls-max-version 1.2
Router(config-grpc)# commit
```

tls-max-version can be 1.0, 1.1, 1.2, or 1.3. The default maximum version for TLS is 1.3.

Step 2 Verify the gRPC TLS minimum and maximum versions.

Example:

```
Router# show grpc
Thu Aug 29 00:49:24.428 UTC

Server name                : DEFAULT
Address family             : dual
Port                       : 57400

Service ports
  gNMI                     : none
  P4RT                     : none
  gRIBI                    : none

DSCP                       : Default
TTL                        : 64
VRF                        : global-vrf
Server                     : disabled (Unknown)
TLS                        : enabled
TLS mutual                 : disabled
Trustpoint                 : none
Certificate Authentication : disabled
Certificate common name    : ems.cisco.com
TLS v1.0                   : enabled
Maximum requests           : 128
Maximum requests per user  : 10
Maximum streams            : 32
Maximum streams per user   : 32
Maximum concurrent streams : 32
Memory limit (MB)         : 1024
Keepalive time             : 30
Keepalive timeout          : 20
Keepalive enforcement minimum time : 300
TLS Minimum Version      : TLS 1.0
TLS Maximum Version     : TLS 1.2

TLS cipher suites
  Default                  : none
  Default TLS1.3           : aes_128_gcm_sha256
                           : aes_256_gcm_sha384
                           : chacha20_poly1305_sha256

  Enable                   : none
  Disable                  : none

Operational enable         : ecdhe-rsa-chacha20-poly1305
                           : ecdhe-ecdsa-chacha20-poly1305
                           : ecdhe-rsa-aes128-gcm-sha256
                           : ecdhe-ecdsa-aes128-gcm-sha256
                           : ecdhe-rsa-aes256-gcm-sha384
                           : ecdhe-ecdsa-aes256-gcm-sha384
                           : ecdhe-rsa-aes128-sha
                           : ecdhe-ecdsa-aes128-sha
                           : ecdhe-rsa-aes256-sha
                           : ecdhe-ecdsa-aes256-sha
```

```

: aes128-gcm-sha256
: aes256-gcm-sha384
: aes128-sha
: aes256-sha
Operational disable : none
Listen addresses    : ANY

```

The TLS 1.3 cipher suites are not configurable, they are either fixed or static.

SPIFFE ID-Based Authentication and Authorization Services for gRPC Services

Table 3: Feature History Table

Feature Name	Release Information	Description
SPIFFE ID-Based Authentication and Authorization Services for gRPC Services	Release 24.2.11	<p>You can now securely manage service identities for workloads that communicate over gRPC. This capability is critical for environments such as distributed systems, where workloads move across different platforms.</p> <p>This security measure is feasible because workloads can use the Secure Production Identity Framework for Everyone (SPIFFE) ID and SPIFFE Verifiable Identity Document (SVID) to encrypt and authenticate gRPC traffic.</p> <p>This feature introduces the following changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> • aaa map-to username <p>Yang Data Models:</p> <ul style="list-style-type: none"> • New XPaths for <code>Cisco-IOS-XR-um-aaa-task-user-cfg.yang</code> • New XPaths for <code>Cisco-IOS-XR-aaa-locald-cfg.yang</code> <p>(see GitHub, YANG Data Models Navigator)</p>

The SPIFFE standard specifies a framework that can bootstrap and issue identities to services across diverse environments and organizational boundaries. SPIFFE assigns a unique identity to each workload with a SPIFFE ID and securely encapsulates it within a SPIFFE Verifiable Identity Document (SVID). The SVID, which is short-lived, corresponds exclusively to its SPIFFE ID and can be encoded either as an X.509 certificate or as a JSON Web Token (JWT). This dual-format capability facilitates robust identity verification.

This feature provides a mechanism for mapping a SPIFFE ID to an XR user for authorization purposes. This feature enables Extensible Manageability Services (EMS) to use the SVID, which are certificates that essentially contain SPIFFE IDs, to perform the following operations:

- Authentication via mTLS
- AuthZ authorization using the SVID

The XR authorization occurs with the XR user which is mapped to the SPIFFE ID. Mapping the SPIFFE ID to a username is required for gRPC services to perform IOS XR authentication and authorization before executing any operations on the device. If the authz evaluation is successful then only the connection request is processed; otherwise, access is denied.

Workflow for SPIFFE ID-Based Authentication and Authorization for gRPC Services

The high-level workflow of SPIFFE ID-based authentication and authorization for gRPC services involves the following steps:

1. The EMS starts searching for the *spiffe-user-map.json* file at the location `/misc/config/grpc/gnsi/credentialz/spiffe-user-map.json`.
2. If the file exists, it is parsed, and the mapping is stored globally in the `aaa/auth` package.
3. If the file does not exist or parsing is unsuccessful, the mapping will be empty.
4. The EMS registers with the configuration manager to receive updates for the `aaa` configuration.
5. When processing requests in the Authentication interceptor, the spiffe-user mapping API checks for the SPIFFE ID mapping in the map created in [step 2](#).
6. If the mapping exists, the API responds with the corresponding username.
7. If the mapping does not exist but the `aaa` configuration exists, the API responds with the configured username.
8. If neither the mapping nor the `aaa` configuration is present, the API responds with an empty string.
9. Upon a client connecting to the server, the server interceptor extracts the SPIFFE ID from the client's certificate and uses the mapping stored in the `aaa/auth` package to find the corresponding username.
10. The username identifies it and then includes the metadata into the context.
11. gRPC services that require XR Authorization will later verify the access rights for the username identified in the previous step when handling the request.
12. If the mapping is unsuccessful, the request is passed to the relevant service, such as gNMI, which then decides whether to grant or deny access based on its authorization requirements.

Authenticate and Authorize gRPC Service Requests Using the SPIFFE Standard

Before you begin

Before authenticating and authorizing gRPC service requests using the SPIFFE standard, ensure the following prerequisites are met:

- Enable mutual TLS authentication with the `tls-mutual` command.
- Enable certificate authentication with the `certificate-authentication` command to facilitate SPIFFE ID recognition. For more information, see [Authenticate gRPC Services, on page 6](#).

- Configure the gNSI Authz policy by setting the principal to the SPIFFE-ID for service-level authorization (gNSI AuthZ).

After establishing the connection, the gRPC server extracts the SPIFFE ID from the client's certificate.

To authenticate and authorize gRPC service requests using the SPIFFE standard, follow these steps:

Procedure

Step 1 Configure the username in the system.

Example:

```
Router#show running-config aaa
Thu Oct 12 11:43:15.771 UTC
username cisco
  group root-lr
  group cisco-support
  password 7 104D000A061843595F
!
```

Step 2 Map the SPIFFE ID to a username using the **aaa map-to username** command. This command assigns a default username to any SPIFFE ID.

```
Router(config)#aaa map-to username cisco spiffe-id any
Router(config)#commit
```

Note

Each SPIFFE ID supports only one username.

Step 3 Evaluate the client's SPIFFE ID against the service-level authorization policy (gNSI AuthZ). For more information about gNSI authz policies, see [gRPC Network Security Interface](#) , on page 47.

Certificate Common-Name For Dial-in Using gRPC Protocol

Table 4: Feature History Table

Feature Name	Release Information	Description
Certificate Common-Name For Dial-in Using gRPC Protocol	Release 24.1.1	<p>You can now specify a common-name for the certificate generated by the router while using gRPC dial-in. Earlier, the common-name in the certificate was fixed as <i>ems.cisco.com</i> and was not configurable. Using a specified common-name avoids potential certification failures where you may specify a hostname different from the fixed common name to connect to the router.</p> <p>The feature introduces these changes:</p> <p>CLI:</p> <ul style="list-style-type: none">• grpc certificate common-name <p>YANG Data Model:</p> <ul style="list-style-type: none">• New XPath for <code>Cisco-IOS-XR-um-grpc-cfg.yang</code>• New XPath for <code>Cisco-IOS-XR-man-ems-cfg</code> <p>(see GitHub, YANG Data Models Navigator)</p>

When using gRPC dial-in on Cisco IOS-XR router, the **common-name** associated with the certificate generated by the router is fixed as *ems.cisco.com* and this caused failure during certificate verification.

From Cisco IOS XR Release 24.1.1, you can now have the flexibility of specifying the common-name in the certificate using the **grpc certificate common-name** command. This allows gRPC clients to verify if the domain name in the certificate matches the domain name of the gRPC server being accessed.

Configure Certificate Common Name For Dial-in

Configure a common name to be used in EMSD certificates for gRPC dial-in.

Procedure

Step 1 Configure a common name.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#certificate common-name cisco.com
Router(config-grpc)#commit
```

Use the show command to verify the common name:

```
Router#show grpc
Certificate common name           : cisco.com
```

Note

For the above configuration to be successful, ensure to regenerate the certificate. so that the new EMSD certificates include the configured common name.

To **regenerate** the self-signed certificate, perform the following steps.

Step 2

Remove the certificates: /misc/config/grpc/ems.pem, /misc/config/grpc/ems.key, and /misc/config/grpc/ca.cert from /misc/config/grpc file.

Example:

```
Router#run ls -ltr /misc/config/grpc/

total 16
drwx-----. 2 root root 4096 Feb 14 09:17 dialout
-rw-rw-rw-. 1 root root 1505 Feb 14 10:58 ems.pem
-rw-----. 1 root root 1675 Feb 14 10:58 ems.key
-rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert

Router#run rm -rf /misc/config/grpc/ems.pem /misc/config/grpc/ems.key
Router#run ls -ltr /misc/config/grpc/

total 8
drwx-----. 2 root root 4096 Feb 14 09:17 dialout
-rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
```

Step 3

Restart gRPC server by toggling the TLS configuration.

Configure gRPC with non TLS and then re-configure with TLS.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#no-tls
Router(config-grpc)#commit

Router#run ls -ltr /misc/config/grpc/

total 8
drwx-----. 2 root root 4096 Feb 14 09:17 dialout
-rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert

Router#config
Router(config)#grpc
Router(config-grpc)#no no-tls
Router(config-grpc)#commit

Router#run ls -ltr /misc/config/grpc/

total 16
drwx-----. 2 root root 4096 Feb 14 09:17 dialout
-rw-rw-rw-. 1 root root 1505 Feb 14 14:23 ems.pem
```

```
-rw-----. 1 root root 1675 Feb 14 14:23 ems.key
-rw-r--r--. 1 root root 1505 Feb 14 14:23 ca.cert
```

Copy the newly generated `/misc/config/grpc/ems.pem` certificate in this path (from the device) to the gRPC client.

gRPC over UNIX Domain Sockets

Table 5: Feature History Table

Feature Name	Release Information	Description
gRPC Connections over UNIX domain sockets for Enhanced Security and Control	Release 7.5.1	<p>This feature allows local containers and scripts on the router to establish gRPC connections over UNIX domain sockets. These sockets provide better inter-process communication eliminating the need to manage passwords for local communications. Configuring communication over UNIX domain sockets also gives you better control of permissions and security because UNIX file permissions come into force.</p> <p>This feature introduces the grpc local-connection command.</p>

You can use local containers to establish gRPC connections via a TCP protocol where authentication using username and password is mandatory. This functionality is extended to establish gRPC connections over UNIX domain sockets, eliminating the need to manage password rotations for local communications.

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. With this feature, the gRPC server listens over UNIX domain sockets that must be accessible from within the container via a local connection by default. With the UNIX socket enabled, the server listens on both TCP and UNIX sockets. However, if disable the UNIX socket, the server listens only on the TCP socket. The socket file is located at `/var/lib/docker/ems/grpc.sock` directory.

The following process shows the configuration changes required to enable or disable gRPC over UNIX domain sockets.

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 following 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 is 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.

The following is an example of a Go client connecting to UNIX socket:

```
const sockAddr =
"/var/lib/docker/ems/grpc.sock"
...
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...)
    ...
}
```

gRPC Network Management Interface

gRPC Network Management Interface (gNMI) is a gRPC-based network management protocol used to modify, install or delete configuration from network devices. It is also used to view operational data, control and

generate telemetry streams from a target device to a data collection system. It uses a single protocol to manage configurations and stream telemetry data from network devices.

The subscription in a gNMI does not require prior sensor path configuration on the target device. Sensor paths are requested by the collector (such as pipeline), and the subscription mode can be specified for each path. gNMI uses gRPC as the transport protocol and the configuration is same as that of gRPC.

gNMI Operations

gNMI Operation	Supported Release	Description	Additional Details
Capabilities	Release 7.0.1	Retrieves the metadata of the network device.	—
Get	Release 7.0.1	Retrieve state data, configuration, and operational information from a network device	—
Set	Release 7.0.1	You can modify the state of a network device such as router's configuration, replace router's entire configuration sections, or delete specific parts of the configuration using the Set operation.	—
Subscribe	Release 24.2.1	Subscribes to a stream of updates for specific paths within the device's data model.	Stream Telemetry Data for LLDP Statistics



Note The gNMI **Get** operation is not supported for Sysadmin YANG models.

gNMI Wildcard in Schema Path

Table 6: Feature History Table

Feature Name	Release Information	Description
Use gNMI Get Request With Wildcard Key to Retrieve Data	Release 7.5.2	<p>You use a gRPC Network Management Interface (gNMI) <code>Get</code> request with wildcard key to retrieve the configuration and operational data of all the elements in the data model schema paths. In earlier releases, you had to specify the correct key to retrieve data. The router returned a JSON error message if the key wasn't specified in a list node.</p> <p>For more information about using wildcard search in gNMI requests, see the Github repository.</p>

gNMI protocol supports wildcards to indicate all elements at a given subtree in the schema. These wildcards are used for telemetry subscriptions or gNMI `Get` requests. The encoding of the path in gNMI uses a structured format. This format consists of a set of elements such as the path name and keys. The keys are represented as string values, regardless of their type within the schema that describes the data. gNMI supports the following options to retrieve data using wildcard search:

- **Single-level wildcard:** The name of a path element is specified as an asterisk (*). The following sample shows a wildcard as the key name. This operation returns the description for all interfaces on a device.

```
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "*"
    }
  }
  elem {
    name: "config"
  }
  elem {
    name: "description"
  }
}
```

- **Multi-level wildcard:** The name of the path element is specified as an ellipsis (...). The following example shows a wildcard search that returns all fields with a description available under `/interfaces` path.

```
path {
  elem {
    name: "interfaces"
  }
}
```

```

    elem {
      name: "..."/>

```

Example: gNMI Get Request with Unique Path to a Leaf

The following is a sample `Get` request to fetch the operational state of `GigabitEthernet0/0/0/0` interface in particular.

```

path: <
  origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
  elem: <
    name: "interfaces"
  >
  elem: <
    name: "interface-xr"
  >
  elem: <
    name: "interface"
    key: <
      key: "interface-name"
      value: "\"GigabitEthernet0/0/0/0\""
    >
  >
  elem: <
    name: "state"
  >
>
type: OPERATIONAL
encoding: JSON_IETF

```

The following is a sample `Get` response:

```

notification: <
  timestamp: 1597974202517298341
  update: <
    path: <
      origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface-xr"
      >
      elem: <
        name: "interface"
        key: <
          key: "interface-name"
          value: "\"GigabitEthernet0/0/0/0\""
        >
      >
      elem: <
        name: "state"
      >
    >
    val: <
      json_ietf_val: im-state-admin-down
    >
  >
>

```

```
error: <
>
```

Example: gNMI Get Request Without a Key Specified in the Schema Path

The following is a sample Get request to fetch the operational state of all interfaces.

```
path: <
  origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
  elem: <
    name: "interfaces"
  >
  elem: <
    name: "interface-xr"
  >
  elem: <
    name: "interface"
  >
  elem: <
    name: "state"
  >
>
type: OPERATIONAL
encoding: JSON_IETF
```

The following is a sample Get response:

```
path: <
  origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
  elem: <
    name: "interfaces"
  >
  elem: <
    name: "interface-xr"
  >
  elem: <
    name: "interface"
  >
  elem: <
    name: "state"
  >
>
type: OPERATIONAL
encoding: JSON_IETF
notification: <
  timestamp: 1597974202517298341
  update: <
    path: <
      origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
      elem: <
        name: "interfaces"
      >
      elem: <
        name: "interface-xr"
      >
      elem: <
        name: "interface"
        key: <
          key: "interface-name"
          value: "\"GigabitEthernet0/0/0/0\""
        >
      >
    >
    elem: <
      name: "state"
```

```

    >
  >
  val: <
    json_ietf_val: im-state-admin-down
  >
>
update: <
  path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
    elem: <
      name: "interfaces"
    >
    elem: <
      name: "interface-xr"
    >
    elem: <
      name: "interface"
      key: <
        key: "interface-name"
        value: "\"GigabitEthernet0/0/0/1\""
      >
    >
    elem: <
      name: "state"
    >
  >
  val: <
    json_ietf_val: im-state-admin-down
  >
>
update: <
  path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
    elem: <
      name: "interfaces"
    >
    elem: <
      name: "interface-xr"
    >
    elem: <
      name: "interface"
      key: <
        key: "interface-name"
        value: "\"GigabitEthernet0/0/0/2\""
      >
    >
    elem: <
      name: "state"
    >
  >
  val: <
    json_ietf_val: im-state-admin-down
  >
>
update: <
  path: <
    origin: "Cisco-IOS-XR-pfi-im-cmd-oper"
    elem: <
      name: "interfaces"
    >
    elem: <
      name: "interface-xr"
    >
    elem: <

```

```

        name: "interface"
        key: <
          key: "interface-name"
          value: "\"MgmtEth0/RP0/CPU0/0\""
        >
      >
    elem: <
      name: "state"
    >
  >
val: <
  json_ietf_val: im-state-admin-down
>
>

```

Example: gNMI Get Request with Unique Path to a CLI

The following is a sample Get request to fetch the system updates through CLI.

```

path: <
  origin: "cli"
  elem: <
    name: "show version"
  >
>
type: ALL
encoding: ASCII

```

The following is a sample Get response.

```

path: <
  origin: "cli"
  elem: <
    name: "show version"
  >
>

type: ALL
...
...

[
  {
    "source": "unix:///var/run/test_env.sock",
    "timestamp": 1730123328800447525,
    "time": "2024-10-28T06:48:48.800447525-07:00",
    "updates": [
      {
        "Path": "show version",
        "values": {
          "show version":
            "----- show version -----
Cisco IOS XR Software, Version 24.4.1.37I
Copyright (c) 2013-2024 by Cisco Systems, Inc.
Build Information:\n Built By      : swtools
Built On       : Mon Oct 21 03:16:32 PDT 2024
Built Host     : iox-lnx-121\n Workspace   :
/auto/iox-lnx-121-san2/prod/24.4.1.37I.SIT_IMAGE/ncs5500/ws
Version        : 24.4.1.37I\n Location     : /opt/cisco/XR/packages/
Label          : 24.4.1.37I-EFT2LabOnly
cisco NCS-5500 () processor
System uptime is 3 days 22 hours 54 minutes\n\n\n"
        }
      }
    ]
  }
]

```

```

    }
  ]

```

gNMI Bundling of Telemetry Updates

Table 7: Feature History Table

Feature Name	Release Information	Description
gNMI Bundling Size Enhancement	Release 7.8.1	<p>With gRPC Network Management Interface (gNMI) bundling, the router internally bundles multiple gNMI <code>Update</code> messages meant for the same client into a single gNMI <code>Notification</code> message and sends it to the client over the interface.</p> <p>You can now optimize the interface bandwidth utilization by accommodating more gNMI updates in a single notification message to the client. We have now increased the gNMI bundling size from 32768 to 65536 bytes, and enabled gNMI bundling size configuration through Cisco native data model.</p> <p>Prior releases allowed only a maximum bundling size of 32768 bytes, and you could configure only through CLI.</p> <p>The feature introduces new XPaths to the <code>Cisco-IOS-XR-telemetry-model-driven-cfg.yang</code> Cisco native data model to configure gNMI bundling size.</p> <p>To view the specification of gNMI bundling, see Github repository.</p>

To send fewer number of bytes over the gNMI interface, multiple gNMI `Update` messages pertained to the same client are bundled and sent to the client to achieve optimized bandwidth utilization.

The router internally bundles multiple gNMI `Update` messages in a single gNMI `Notification` message of gNMI `SubscribeResponse` message. Cisco IOS XR software Release 7.8.1 supports gNMI bundling size up to 65536 bytes.

Router bundles multiple instances of the same client. For example, a router bundles interfaces `MgmtEth0/RP0/CPU0/0`, `FourHundredGigE0/0/0/0`, `FourHundredGigE0/0/0/1`, and so on, of the following path.

- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`

Router does not bundle messages of different client in a single gNMI `Notification` message. For example,

- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`
- `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/protocols`

Data under the container of the client path cannot be split into different bundles.

The gNMI `Notification` message contains a timestamp at which an event occurred or a sample is taken. The bundling process assigns a single timestamp for all bundled `Update` values. The notification timestamp is the first message of the bundle.

**Note**

- ON-CHANGE subscription mode does not support gNMI bundling.
- Router does not enforce bundling size in the following scenarios:
 - At the end of (N-1) message processing, if the notification message size is less than the configured bundling size, router allows one extra instance which could result in exceeding the bundling size.
 - Data of a single instance exceeding the bundling size.
- The XPath: `network-instances/network-instance/afts` does not support bundling.

Configure gNMI Bundling Size

gNMI bundling is disabled by default and the default bundling size is 32,768 bytes. gNMI bundling size ranges from 1024 to 65536 bytes. Prior to Cisco IOS XR software Release 7.8.1 the range was 1024 to 32768 bytes. You can enable gNMI bundling to all gNMI subscribe sessions and specify the bundling size.

Configuration Example

This example shows how to enable gNMI bundling and configure bundling size.

```
Router# configure
Router(config)# telemetry model-driven
Router(config-model-driven)# gnmi
Router(config-gnmi)# bundling
Router(config-gnmi-bdl)# size 2000
Router(config-gnmi-bdl)# commit
```

Running configuration

This example shows the running configuration of gNMI bundle.

```
Router# show running-config
telemetry model-driven
  gnmi
    bundling
      size 2000
  !
  !
  !
```


Replace Router Configuration at Sub-tree Level Using gNMI

Table 8: Feature History Table

Feature Name	Release Information	Description
Replace Router Configuration at Sub-tree Level Using gNMI	Release 7.8.1	Using the gNMI <code>SetRequest</code> message, you can replace the router's existing configuration with a new set of configurations at the subtree level within the same model. Earlier you could replace router configurations at the data tree root level. To view the specification of gNMI replace, see Github repository .

The gNMI replace feature replaces the existing configuration on the router with the new configuration using a `SetRequest` RPC message. It allows you to specify a `path` (a structured format for path elements, and any associated key values) as the root prompt to perform a `replace` operation. Cisco IOS XR software Release 7.8.1 supports subtree-level replace operation. Prior to this release replace operation was performed at datatree-level.

Replace operation either includes all the path elements which are defined under the root or only few of them. If the omitted path elements are configured with default values, they are reverted to its default values during the replace operation. If the omitted path elements are not configured with default values, they are deleted from the data tree during the replace operation, and returned to its original unconfigured state. Consider the following example:

In the following data tree schema, `b` has a default value of `true` and `c` has no default value. Both `b` and `c` are set as `False`.

```

root +
  |
  + a ---+
  |     |
  |     +-- b
  |     |
  |     +-- c
  |
  + d ---+
  |     |
  |     +-- e
  |     |
  |     +-- f

```

When a `replace` operation is performed with `e` and `f` as set, and all other elements are omitted, `b` is reverted to its default setting `true`, and `c` is deleted from the tree, and returned to its original unconfigured state.

Following example shows the `SetRequest` and `SetResponse` of gNMI replace operation.

gNMI Replace Example

This example shows the gNMI replace request and response messages.

```

Request Message:
replace: <
  path: <
    elem: <
      name: "system"
    >
  >

```

```

    elem: <
      name: "config"
    >
    elem: <
      name: "hostname"
    >
  >
  val: <
    json_ietf_val: "\"testing123\""
  >
>

Response Message:
path: <
  elem: <
    name: "system"
  >
  elem: <
    name: "config"
  >
  elem: <
    name: "hostname"
  >
>
  op: REPLACE
>
message: <
>
timestamp: 1662873319202107537

```

gNMI XPath-Based Authorization

Table 9: Feature History Table

Feature Name	Release Information	Description
gNMI XPath-Based Authorization	Release 24.2.11	<p>We've introduced gNMI authorization through the gNSI pathz policy which is adding authorization of a user or a group to access a specified YANG XPath through gNMI. The policy configurations can be done on the router either when the router boots up or dynamically when the router is up and running. When a user or a group sends a <code>gNMI SetRequest</code> message using a certain XPath, the system validates the request against the permissions specified in the policies associated with that user or the group.</p> <p>To view the specification of gNSI for the OpenConfig XPath-based Authorization, see the Github repository.</p> <p>The feature introduces these changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> • <code>show gnsi path authorization policy</code> • <code>show gnsi path authorization counters</code> • <code>show gnsi trace pathz</code> • <code>show gnsi path authorization statistics</code> • <code>show tech-support gnsi</code> • <code>clear gnsi path authorization counters</code>

How gNSI pathz Policy Works

Upon receiving a `gNMI SetRequest` message for a configuration change, the router applies an XPath-based pathz policy to determine the request's authorization. The pathz policy originates from a gNSI RPC within the router. The policy configurations can be established during the router's boot process or dynamically adjusted while the router is operational.

The router securely receives the initial pathz policy either through Secure Zero Touch Provisioning (sZTP) or a secure bootstrapping protocol like bootz when booting up. The policy includes the user or group name and a list of rules defining XPaths and their associated access permissions. The policy is enforced before processing any gNMI requests.

Authorization by the gNSI pathz policy is granted or denied based on user or group credentials, permitting or declining the `gNMI SetRequest` accordingly.

gNMI Authorization Using gNSI pathz Policy

Starting from Release 24.2.11, you can perform gNMI XPath-based authorization using gNSI pathz policies.

The [gnsi-pathz](#) YANG model defines the following counters and timestamps for each configured rule READ, WRITE, PERMIT, and DENY.

- access-rejects: 64-bit
- last-access-reject: timestamp
- access-accepts: 64-bit
- last-access-accept: timestamp

The counters get incremented per accepted or rejected XPath (Example, per gNMI request).

Define Authorization Policy for a gNSI Pathz

The authorization policy for gNSI Pathz consists of three components.

Table 10: Authorization Policy Components

Authorization Policy Component	Details
Users	Individuals named in rules or group definitions.
Groups of users	<p>A group of users in the administrative domain, such as operators or administrators.</p> <ul style="list-style-type: none"> • The matching policy gives precedence to a specific user over a group. • Match rules enable authorization against either a user or a group, but not both simultaneously.
Policy rules	<p>Each rule defines a single authorization policy.</p> <ul style="list-style-type: none"> • Authorization (how the policy is defined) is performed for a specific user in a predefined group of users on a specific gNMI path and a specific access methodology (example: READ or WRITE). <ul style="list-style-type: none"> • The wildcard character (*): <ul style="list-style-type: none"> • Replaces the missing keys in keyed path elements. Absence of keys implies a wildcard by default. • Masks all the values entirely, it doesn't permit partial value masking (Example: /this/is/a/keyed[name=Ethernet1/*]/things is invalid).

How Authorization Policy Matching Rules Work

Policy Matching Rule	Description
Multiple rules	The authorization process evaluates the rule with the longest match when granting access, rather than defaulting to the first rule encountered.

Policy Matching Rule	Description
A defined KEY and wildcard in a keyed path	The defined KEY in the keyed path is preferred over the wildcard. For example, the router prefers /a/b[key=FOO]/c/d over /a/b[key=*/c/d due to its more precise key match.
A user-specific rule and a corresponding group rule for the same user	The rule that corresponds to a specific user is prioritized over the one that matches with a user's group.
Permission mode	A mode that matches with the request (READ or WRITE) is considered.
DENY or PERMIT	DENY takes priority over PERMIT when other conditions are equal, and multiple matching rules are present.

Policy evaluation results with a single best match rule for the provided {user, path, or mode}. If multiple best matches emerge, an error is logged, and the evaluation fails.

If no matching rule is found, an implicit DENY is applied and detailed in a log entry.

The authorization evaluation process results in a PERMIT or DENY decision, along with the version of the policy and the identifier of the rule applied.

Scenario for Authorization Policy Rules

Rule	User	Group	Path	Action	Mode
1	Bob	—	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	READ
2	Bob	—	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	WRITE
3	Bob	—	/interfaces/interface[FourHundredGigE1/1/1/1]	DENY	WRITE
4	—	Admin	/interfaces/interface[*]	PERMIT	WRITE
5	Bob	—	/interfaces	PERMIT	READ
6	—	Admin	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	WRITE
7	Jim	—	/interfaces/interface[FourHundredGigE0/0/0/0]	DENY	WRITE

For user Bob, the following authorization rules apply:

- READ or WRITE (gNMI request) access to the XPath /interfaces/interface[FourHundredGigE0/0/0/0] is granted under rules 1 and 2.
- READ access to the XPath /interfaces/interface[FourHundredGigE1/1/1/1] is granted under rule 5 due to the longest match criterion, which specifies READ mode. WRITE access to this path is denied by rule 3.
- WRITE access to the XPath /interfaces/interface[FourHundredGigE2/2/2/2] is granted being a member of the Admins group as specified by rule 4. Without the Admin membership, access is denied by the default deny all rule.

- READ access to the XPath `/interfaces/interface[FourHundredGigE2/2/2/2]` is granted under rule 5, independent of group affiliation.

For user Jim, the following authorization rule applies:

- Access to the XPath `/interfaces/interface[FourHundredGigE0/0/0/0]` is controlled by a policy that favors personal user permissions over group permissions. As a result, although the admins group is allowed access, Jim is individually denied access because the policy emphasizes user-specific rules.

gNSI Pathz Authorization Policy Configuration

To set a gNSI pathz authorization policy, you can perform either of the following methods:

- [Load gNSI Pathz Policies at Boot-time, on page 30](#)
- [Rotate, Finalize, and Get the gNSI Pathz Policy, on page 30](#)

Load gNSI Pathz Policies at Boot-time

To load gNSI pathz policies at boot-time into the router, you can use either sZTP or bootstrapping.

For details on loading gNSI pathz policy through sZTP, refer to *Secure Zero Touch Provisioning* section of *Cisco IOS XR Setup and Upgrade Guide for Cisco 8000 Series Routers* guide.

Rotate, Finalize, and Get the gNSI Pathz Policy

When the router is up and running, you can rotate (update), finalize (commit), and get (read) the gNSI pathz policy using the gNSI pathz gRPC operations. To view the specification of gNSI pathz policy rotation, see the [Github](#) repository.

gNSI pathz supports the following policy instances:

- Active policy—Used for authorizing gNMI requests.
- Potential or candidate policy—Used to test a policy before rotation.

Rules for Authorization Policy Rotation

- The node holds on to the candidate policy indefinitely until either:
 - The candidate is committed or again rotated, or
 - The RPC session is closed (this event removes the candidate instance).
- A single policy rotation RPC can be active at any given time. Concurrent RPC requests for policy rotation is rejected with the gRPC error code `UNAVAILABLE`.
 - gNMI allows different encodings, including JSON. IOS XR applies the gNSI pathz policy based on each leaf of the flattened JSON model for authorizing the gNMI request.

Metrics of gNSI Authorization Rules

IOS-XR pathz supports the following statistics, counters, diagnostics, and trace data commands for the gNSI authorization rules:

- [Pathz Policy and Statistics](#)
- [Path Authorization Counters](#)
- [Traces for Pathz](#)
- [gNSI Tech Support](#)

gNSI Path Authorization Counters

The gNSI path authorization counters show the counters for a given gRPC server-name for all XPath, or the specified XPath. Providing the XPath and server-name is optional. To view the gNSI Path Authorization counters, use the **show gnsi path authorization counters** command.

```

Router# show gnsi path authorization counters
Mon Apr  1 08:05:46.297 UTC
-----Pathz Counters Info-----

/system/config/hostname:
Read                               Write
Rejects :                          0                      0
  Last :                          N/A                      N/A
Accepts :                          0                      3
  Last :                          N/A    Mon, 01 Apr 2024 08:05:25 +0000
Total path records received 1

Router# show gnsi path authorization counters server-name 64.103.223.33
Mon Apr  1 08:33:25.194 UTC
-----Pathz Counters Info-----

/:
Read                               Write
Rejects :                          0                      2
  Last :                          N/A    Mon, 01 Apr 2024 08:32:37 +0000
Accepts :                          0                      0
  Last :                          N/A                      N/A

/system/config/hostname:
Read                               Write
Rejects :                          0                      6
  Last :                          N/A    Mon, 01 Apr 2024 08:32:36 +0000
Accepts :                          0                      0
  Last :                          N/A                      N/A
Total path records received 2
Router#

Router# show gnsi path authorization counters path /system/config/hostname
Mon Apr  1 08:32:46.468 UTC
-----Pathz Counters Info-----

/system/config/hostname:
Read                               Write
Rejects :                          0                      6
  Last :                          N/A    Mon, 01 Apr 2024 08:32:36 +0000
Accepts :                          0                      0
  Last :                          N/A                      N/A
Total path records received 1
Router#

• To clear the gNSI path authorization counters, use the clear gnsi path authorization counters command.

Router# clear gnsi path authorization counters
Router#

```

gNSI Pathz Policy and Statistics

To display the configured gNSI policy and statistics, use the are following commands:

- **show gnsi path authorization policy**—Shows the running gNSI path authorization policy.
- **show gnsi path authorization statistics**—Shows gNSI path authorization statistics.

```
Router# show gnsi path authorization policy
Mon Apr  1 04:29:37.905 UTC
version:"1" created_on:1711946719670313 policy:{rules:{user:"cafyauto"
path:{origin:"openconfig" elem:{name:"system"} elem:{name:"config"} elem:{name:"hostname"}}
  action:ACTION_PERMIT mode:MODE_WRITE}}
Router#

Router# show gnsi path authorization statistics
Mon Apr  1 04:29:23.259 UTC
-----Pathz Info-----
Engine:

State:
  Active Policy:
    Version                : 1
    Created On (UTC)        : Wed, 09 Dec 54251401 07:58:33 +0000
  Sandbox Policy:
    Version                : N/A
    Created On (UTC)        : N/A
  Policy Rotation in Progress: False

Stats:
  Rotations in Progress Count: 0
  Policy Rotations             : 0
  Policy Rotation Errors       : 0
  Policy Upload Requests       : 0
  Policy Upload Errors         : 0
  Policy Finalize              : 0
  Policy Finalize Errors       : 0
  Probe Requests               : 0
  Probe Errors                 : 0
  Get Requests                 : 0
  Get Errors                   : 0
  Policy Unmarshall Errors     : 0
  Sandbox Policy Errors        : 0

Counters:
  No Policy Auth Requests      : 0
  gNMI Path Leaves             : 0
  gNMI Authorizations          : 0
  gNMI Set Path Permit         : 0
  gNMI Set Path Deny          : 0
  gNMI Get Path Permit         : 0
  gNMI Get Path Deny          : 0

Errors:
  Path To String               : 0
  Origin Type                  : 0
  Bad Mode                     : 0
  Bad Action                   : 0
  JSON Flatten                 : 0
  String To Path               : 0
  Join Paths                   : 0
  Nil Path                     : 0
  Nil SetRequest               : 0
  Empty User                   : 0
```



```

Probe Internal          : 0
Path Counters:
  Increment             : 0
  Find                  : 0
  Clear                 : 0
  Walk                  : 0

```

gNSI Pathz Trace Data

To trace the configured gNSI policy, use the **show gnsi trace pathz** command.

```

Router# show gnsi trace pathz all
Mon Apr  1 04:31:26.689 UTC
61 wrapping entries (21760 possible, 512 allocated, 0 filtered, 61 total)
Apr  1 04:07:09.681 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(178) 'Trying to load policy'
'/mnt/rdsfs/ems/gnsi/pathz_policy.txt'
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(173) 'Set Sandbox policy'
'1(54251382-02-18 11:34:58 +0000 UTC)'
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(179) 'Set Policy from'
'/mnt/rdsfs/ems/gnsi/pathz_policy.txt'
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(249) 'Pathz Policy Clearing
Counters' ' '
Apr  1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code (79): 'Engine Initialized'
Apr  1 04:08:05.761 gnsi/pathz 0/RP0/CPU0 t11794 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:05.761 gnsi/pathz_err 0/RP0/CPU0 t11794 Pathz ERROR: Code (65): 'Nil Policy'
Apr  1 04:08:05.788 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:05.788 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(176) 'Get'
'POLICY_INSTANCE_ACTIVE 1(1711946094752098)'
Apr  1 04:08:05.791 gnsi/pathz_deny 0/RP0/CPU0 t11481 Pathz DENY: Code(235) 'Upd/Rep Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:05.808 gnsi/pathz_deny 0/RP0/CPU0 t11383 Pathz DENY: Code(234) 'Del Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:05.821 gnsi/pathz_deny 0/RP0/CPU0 t11480 Pathz DENY: Code(235) 'Upd/Rep Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:07.348 gnsi/pathz_deny 0/RP0/CPU0 t11383 Pathz DENY: Code(235) 'Upd/Rep Denied
path' 'cafyauto@/lldp/config/enabled,|1,1711946094752098'
Apr  1 04:08:08.205 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:08.205 gnsi/pathz_err 0/RP0/CPU0 t11383 Pathz ERROR: Code (65): 'Nil Policy'
Apr  1 04:08:08.221 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(63) 'Pathz.Get()'
'5.38.4.111:52126'
Apr  1 04:08:08.221 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(176) 'Get'
'POLICY_INSTANCE_ACTIVE 1(1711946094752098)'
Apr  1 04:08:08.238 gnsi/pathz_deny 0/RP0/CPU0 t11481 Pathz DENY: Code(235) 'Upd/Rep Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Apr  1 04:08:08.281 gnsi/pathz_deny 0/RP0/CPU0 t11480 Pathz DENY: Code(234) 'Del Denied
path' 'cafyauto@/system/config/hostname,|1,1711946094752098'
Router#

```

gNSI State Details

To collect diagnostic information of gNSI, use the **show tech-support gnsi** command.

```

Router# show tech-support gnsi
Mon Apr  1 06:55:51.482 UTC
++ Show tech start time: 2024-Apr-01.065551.UTC ++
Mon Apr  1 06:55:52 UTC 2024 Waiting for gathering to complete
...
Mon Apr  1 06:56:01 UTC 2024 Compressing show tech output
Show tech output available at Router#:
/harddisk:/showtech/showtech-mtb_sf2-gnsi-2024-Apr-01.065551.UTC.tgz
++ Show tech end time: 2024-Apr-01.065601.UTC ++

```

show tech-support gnsi command places the collected diagnostic information in a file, example **Router#:/harddisk:/showtech/showtech-mtb_sf2-gnsi-2024-Apr-01.065551**.

gRPC Network Operations Interface

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based microservices for executing operational commands on network devices. These services are to be used in conjunction with gRPC network management interface (gNMI) for all target state and operational state of a network. gNOI uses gRPC as the transport protocol and the configuration is same as that of gRPC. For more information about gNOI, see the [Github](#) repository.

gNOI RPCs

To send gNOI RPC requests, you need a client that implements the gNOI client interface for each RPC.

All messages within the gRPC service definition are defined as protocol buffer (.proto) files. gNOI OpenConfig proto files are located in the [Github](#) repository.

Table 11: Feature History Table

Feature Name	Release Information	Description
gNOI MPLS Proto	Release 7.5.4	The RPCs defined in the proto file can be used to perform Multiprotocol Label Switching (MPLS) operations on the router.
gNOI OS Proto	Release 7.9.1	The RPCs defined in the proto file can be used to install the software, activate the software version and verify that the installation is successful.
gNOI System Proto	Release 7.8.1	You can now avail the services of <code>CancelReboot</code> to terminate outstanding reboot request, and <code>KillProcess</code> RPCs to restart the process on device.

gNOI supports the following remote procedure calls (RPCs):

System RPCs

The RPCs are used to perform key operations at the system level such as upgrading the software, rebooting the device, and troubleshooting the network. The **system.proto** file is available in the [Github](#) repository.

RPC	Description
Reboot	Reboots the target. The router supports the following reboot options: <ul style="list-style-type: none"> • COLD = 1; Shutdown and restart OS and all hardware • POWERDOWN = 2; Halt and power down • HALT = 3; Halt • POWERUP = 7; Apply power
RebootStatus	Returns the status of the target reboot.
SetPackage	Places a software package including bootable images on the target device.
Ping	Pings the target device and streams the results of the ping operation.
Traceroute	Runs the traceroute command on the target device and streams the result. The default hop count is 30.
Time	Returns the current time on the target device.
SwitchControlProcessor	Switches from the current route processor to the specified route processor. If the target does not exist, the RPC returns an error message.
CancelReboot	Cancels any pending reboot request.
KillProcess	Stops an OS process and optionally restarts it.

File RPCs

The RPCs are used to perform key operations at the file level such as reading the contents of a file and its metadata. The **file.proto** file is available in the [Github](#) repository.

RPC	Description
Get	Reads and streams the contents of a file from the target device. The RPC streams the file as sequential messages with 64 KB of data.
Remove	Removes the specified file from the target device. The RPC returns an error if the file does not exist or permission is denied to remove the file.
Stat	Returns metadata about a file on the target device.
Put	Streams data into a file on the target device.

RPC	Description
TransferToRemote	Transfers the contents of a file from the target device to a specified remote location. The response contains the hash of the transferred data. The RPC returns an error if the file does not exist, the file transfer fails or an error when reading the file. This is a blocking call until the file transfer is complete.

Certificate Management (Cert) RPCs

The RPCs are used to perform operations on the certificate in the target device. The **cert.proto** file is available in the [Github](#) repository.

RPC	Description
Rotate	Replaces an existing certificate on the target device by creating a new CSR request and placing the new certificate on the target device. If the process fails, the target rolls back to the original certificate.
Install	Installs a new certificate on the target by creating a new CSR request and placing the new certificate on the target based on the CSR.
GetCertificates	Gets the certificates on the target.
RevokeCertificates	Revokes specific certificates.
CanGenerateCSR	Asks a target if the certificate can be generated.
LoadCertificateAuthorityBundle	Loads a bundle of CA certificates on the target. This CA certificate bundle is used to verify the client certificate when mutual TLS is enabled.

Interface RPCs

The RPCs are used to perform operations on the interfaces. The **interface.proto** file is available in the [Github](#) repository.

RPC	Description
SetLoopbackMode	Sets the loopback mode on an interface.
GetLoopbackMode	Gets the loopback mode on an interface.
ClearInterfaceCounters	Resets the counters for the specified interface.

Layer2 RPCs

The RPCs are used to perform operations on the Link Layer Discovery Protocol (LLDP) layer 2 neighbor discovery protocol. The **layer2.proto** file is available in the [Github](#) repository.

Feature Name	Description
ClearLLDPInterface	Clears all the LLDP adjacencies on the specified interface.

BGP RPCs

The RPCs are used to perform operations on the Link Layer Discovery Protocol (LLDP) layer 2 neighbor discovery protocol. The **bgp.proto** file is available in the [Github](#) repository.

Feature Name	Description
ClearBGPNeighbor	Clears a BGP session.

Diagnostic (Diag) RPCs

The RPCs are used to perform diagnostic operations on the target device. You assign each bit error rate test (BERT) operation a unique ID and use this ID to manage the BERT operations. The **diag.proto** file is available in the [Github](#) repository.

Feature Name	Description
StartBERT	Starts BERT on a pair of connected ports between devices in the network.
StopBERT	Stops an already in-progress BERT on a set of ports.
GetBERTResult	Gets the BERT results during the BERT or after the operation is complete.

MPLS RPCs

The RPCs are used to perform MPLS operations on the target device. The **mpls.proto** file is available in the [Github](#) repository.

Feature Name	Description
MPLSPing	Checks basic connectivity using MPLS ping operation. See RFC 4379. In Cisco IOS XR Release 7.5.4, the RPC supports <code>ldp_fec</code> and <code>rsvpte_lsp_name</code> destination types. The destination types <code>fec129_pwe</code> and <code>rsvpte_lsp</code> are not supported.
ClearLSP	Clears a single tunnel.
ClearLSPCounters	Clears the MPLS counters for the specified Label Switched Path (LSP).

Operating System (OS) RPCs

The OS service provides an interface for the OS installation on a target device. The RPCs replace the router software to upgrade the system. No concurrent installation is allowed on the same target. The **os.proto** file is available in the [Github](#) repository.

Feature Name	Description
Install	Transfers an OS package onto the target. Note Only Golden ISO installation is supported; RPM installation is not supported.
Activate	Sets the requested OS version as the version that is used at the next reboot. If booting up the requested OS version fails, the system recovers by rolling back to the previously running OS package.
Verify	Verifies the running OS version. The following gNOI OS verify information returns based on the install state: <ul style="list-style-type: none"> • If <code>success</code>, verify returns the installed version. • If <code>failure</code>, verify the version returned by install and set the <code>activation_fail_message</code> to the error returned by the install. • If <code>in-progress</code>, verify returns version returned by install and set the <code>activation_fail_message</code> to <code>in-progress</code>. • If the install state was not retrieved, verify that the version returned is <code>unknown</code> and set the <code>activation_fail_message</code> to <code>Failed to verify the current version.</code>

gNOI RPCs

The following examples show the representation of few gNOI RPCs:

Get RPC

Streams the contents of a file from the target.

```
RPC to 10.105.57.106:57900
RPC start time: 20:58:27.513638
-----File Get Request-----
RPC start time: 20:58:27.513668
remote_file: "harddisk:/giso_image_repo/test.log"

-----File Get Response-----
RPC end time: 20:58:27.518413
contents: "GNOI \n\n"

hash {
  method: MD5
  hash: "D\002\375h\237\322\024\341\370\3619k\310\333\016\343"
}
```

Remove RPC

Remove the specified file from the target.

```
RPC to 10.105.57.106:57900
RPC start time: 21:07:57.089554
-----File Remove Request-----
remote_file: "harddisk:/sample.txt"

-----File Remove Response-----
RPC end time: 21:09:27.796217
File removal harddisk:/sample.txt successful
```

Reboot RPC

Reloads a requested target.

```
RPC to 10.105.57.106:57900
RPC start time: 21:12:49.811536
-----Reboot Request-----
RPC start time: 21:12:49.811561
method: COLD
message: "Test Reboot"
subcomponents {
  origin: "openconfig-platform"
  elem {
    name: "components"
  }
  elem {
    name: "component"
    key {
      key: "name"
      value: "0/RP0"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "location"
  }
}
-----Reboot Request-----
RPC end time: 21:12:50.023604
```

Set Package RPC

Places software package on the target.

```
RPC to 10.105.57.106:57900
RPC start time: 21:12:49.811536
-----Set Package Request-----
RPC start time: 15:33:34.378745
Sending SetPackage RPC
package {
  filename: "harddisk:/giso_image_repo/<platform-version>-giso.iso"
  activate: true
}
method: MD5
hash: "C\314\207\354\217\270=\021\341y\355\240\274\003\034\334"
RPC end time: 15:47:00.928361
```

Reboot Status RPC

Returns the status of reboot for the target.

```

RPC to 10.105.57.106:57900
RPC start time: 22:27:34.209473
-----Reboot Status Request-----
subcomponents {
  origin: "openconfig-platform"
  elem {
    name: "components"
  }
  elem {
    name: "component"
    key {
      key: "name"
      value: "0/RP0"
    }
  }
  elem {
    name: "state"
  }
  elem
    name: "location"
  }
}

RPC end time: 22:27:34.319618

-----Reboot Status Response-----
Active : False
Wait : 0
When : 0
Reason : Test Reboot
Count : 0

```

CancelReboot RPC

Cancels any outstanding reboot

```

Request :
CancelRebootRequest
subcomponents {
  origin: "openconfig-platform"
  elem {
    name: "components"
  }
  elem {
    name: "component"
    key {
      key: "name"
      value: "0/RP0/CPU0"
    }
  }
  elem {
    name: "state"
  }
  elem {
    name: "location"
  }
}

CancelRebootResponse

(rhel7-22.24.10) -bash-4.2$

```

KillProcess RPC

Kills the executing process. Either a PID or process name must be specified, and a termination signal must be specified.

```
KillProcessRequest
pid: 3451
signal: SIGNAL_TERM
```

```
KillProcessResponse
-bash-4.2$
```

gNOI Packet Link Qualification

Table 12: Feature History Table

Feature Name	Release Information	Feature Description
gNOI Packet Link Qualification	Release 24.2.11	<p>You can now check and assess the reliability of the link speed and packet drops between the two network devices (generator and the reflector) by performing the gNOI packet-based link qualification service.</p> <p>This can be achieved by sending the packets from the generator to the reflector, and receiving the looped back packets from the reflector within a certain tolerance limit.</p> <p>The link transmission rate and the link's capacity range for that interface can be obtained from the following gNSI Packet Link Qualification RPC messages:</p> <ul style="list-style-type: none"> • Capabilities—Minimum and maximum rate of the transmission link • Get—Expected rate and actual rate of link transmission

The gRPC Network Operations Interface (gNOI) Packet Link Qualification service provides a way to certify link quality between a generator and a reflector device. The generator device generates test traffic and sends it out of the requested interface, maintaining counters of the sent, received, errored, and dropped packets. The reflector device loops back the traffic on the requested interface. The Packet-Based Link Qualification service verifies that the packets are sent and received on the requested interface. You can obtain the transmission rate and the link's capacity range for that interface from the gNSI Packet Link Qualification RPC messages:

Capabilities and **Get**.

To view the packet link qualification specification, see the [Github](#) repository.

Table 13: Packet Link Qualification (PLQ) RPCs

RPC	Description
Capabilities	<p>Fetches the capabilities of the device as a link qualification service. The capabilities result includes:</p> <ul style="list-style-type: none"> • The roles supported on the device (Packet generator, Physical Medium Dependent (PMD) loopback reflector) • Information on whether the NTP synchronization is supported or not. • Information on whether the current device time is synchronized through NTP or not. • The Maximum number of results stored per interface
Create	<p>Creates a set of link qualifications on the device.</p> <p>Each element in a <code>Create</code> message specifies the following parameters:</p> <ul style="list-style-type: none"> • A unique qualification ID • The interface on which to run the qualification • The endpoint type (the role of the device) • Role-specific configuration • Timing information in the form of either NTP-based or RPC-based timing For more information, see Link Qualifications Based on Timing table. <p>Note Packet generator and PMD loopback roles are supported The packet injector and ASIC loopback roles are not supported.</p>
Delete	<p>Deletes a set of qualifications by their IDs.</p> <p>Stops all the running qualification tests listed and deletes their records from the device.</p> <p>The qualifications are automatically deleted from the device 24 hours either after successful completion or in the event of any error.</p>
Get	<p>Gets the status of each of the unique qualification IDs that you specify. For generator qualifications, it returns the number of packets sent, received, errored, dropped, and the expected and achieved rate in bytes per second. This data isn't present for reflector qualifications.</p>
List	<p>This RPC lists all the qualifications on the device.</p>

Link Qualifications Based on Timing

When you run the `Create` RPC (see table [Packet Link Qualification \(PLQ\) RPCs](#)), it creates a set of link qualifications based on either it's NTP-based or RPC-based timing.

For both NTP-based and RPC-based timings, the qualification start time must be set no earlier than the minimum setup duration from the current time, as specified in the `Capabilities` RPC (see table [Packet Link Qualification \(PLQ\) RPCs](#)) response message.

NTP-based timing specifies:

- Specific start time
- Specific end time
- Teardown time

RPC-based timing specifies:

- Presync duration (duration from the current time to when the setup should start)
- Setup duration
- Qualification duration
- Postsync duration (duration from the end of the qualification to when the teardown should start)
- Teardown duration

gNOI Healthz

Table 14: Feature History Table

Feature Name	Release Information	Feature Description
gNOI Healthz	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100]) This feature is now supported on: <ul style="list-style-type: none"> • 8712-MOD-M

Feature Name	Release Information	Feature Description
gNOI Healthz	Release 24.4.1	<p>Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100]); Centralized Systems (8600 [ASIC:Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100])</p> <p>With gNOI Healthz, you can monitor and troubleshoot device health by collecting logs and conducting root-cause analysis on detected issues. This proactive approach enables early identification and resolution of system health problems, thereby reducing downtime and enhancing reliability.</p> <p>For the specification on gNOI.healthz, see the GitHub repository.</p>

The gRPC Network Operations Interface (gNOI) Healthz is gRPC service that focuses on the health check, and monitoring of the network devices. It determines whether all the nodes of a network are fully functional, degraded, or must be replaced. The gNOI Healthz process involves

- waiting for the health status data from various subsystem components
- inspecting and analyzing health status data to identify any unhealthy entities, and
- collecting logs.

gNOI Healthz, in conjunction with gNMI telemetry, monitors the health of network components.

When a component becomes `HEALTHY` or `UNHEALTHY`, a telemetry update is sent for that health event. For more details about the health event, see [gNOI Healthz RPCs](#). When a system component changes its state to `UNHEALTHY`, the intended artifacts (debug logs, core file, and so on) are generated automatically at the time of the health event.

Router health status updates workflow

1. The client subscribes to the component's OpenConfig path with an `ON_CHANGE` request and waits for a health event to occur. When a health event is detected in the router for that component, the client receives a notification. The client monitors these parameters:
 - **status:** Health, Unhealthy, or Unknown
 - **last-unhealthy time:** Timestamp of last known healthy state
 - **unhealthy-count:** Number of times the particular component is reported unhealthy

- When the router receives gNOI Healthz RPCs from gNOI client, it performs these actions and responds to the gNOI client.

Table 15: gNOI healthz RPCs

When the gNOI client sends...	The Router...
Get RPC	retrieves the latest set of health statuses that are associated with a specific component and its subcomponents.
List RPC	returns all events that are associated with a device.
Artifact RPC	retrieves specific artifacts that are listed by the target system in the List() or Get() RPC.
Acknowledge RPC	acknowledges a series of artifacts that are listed by the Acknowledge() RPC.
Check RPC	performs intensive health checks that may impact the service, ensuring they are done intentionally to avoid disruptions.

Verify router health using gNOI RPCs

Monitor health status telemetry of a router using gNOI healthz RPC.

Procedure

- Step 1** Monitor health state of the router.

Example:

```
Router# show health status
SNo   Component name                               Health status
-----
1      0_RP0_CPU0-appmgr                           healthy
2      0_RP0_CPU0-ownershipd                       healthy
```

- Step 2** Monitor router health with gNOI List RPC by tracking all the events.

Example:

```
Router# /auto/appmgr/xrhealth/bin/gnoic -a ${MGMT_IP} --port 57400 --insecure -u
cisco -p <password> healthz
list --path "openconfig:/components/component[name=${OC_COMP}]"
WARN[0000] "192.0.1.0" could not lookup hostname: lookup 198.51.100.0.in-addr.arpa.
on
171.70.168.183:53: no such host
target "192.0.1.0:57400":
```

Target Name	ID	Status	Path

```

+-----+-----+-----+-----+
| 192.0.1.0:57400 | 1721815320614225976 |
openconfig:components/component[name=0_RP0_CPU0-appmgr] | STATUS_UNHEALTHY |
| 192.0.1.0:57400 | 1721815320614225976 |
openconfig:components/component[name=0_RP0_CPU0-appmgr] | STATUS_UNHEALTHY |
| 192.0.1.0:57400 | 1721815321290718105 |
openconfig:components/component[name=0_RP0_CPU0-appmgr] | STATUS_HEALTHY |
+-----+-----+-----+-----+

+-----+-----+-----+-----+
| Created At |
+-----+-----+-----+-----+
| Artifact ID |
+-----+-----+-----+-----+

2024-07-24 10:02:00.614225976 +0000 UTC |
0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bflaf290fade112ad0ece654e285568|
2024-07-24 10:02:00.614225976 +0000 UTC |
0_RP0_CPU0-appmgr-1721815320614225976-85f9ab33eccf4e48373865f00d8fd24f4e8e4901b49b7809297694f7b57864ea|
2024-07-24 10:02:01.290718105 +0000 UTC |
+-----+-----+-----+-----+

```

Step 3 Monitor router health with gNOI **Get RPC** for specific components.

Example:

```

Router# /auto/appmgr/xrhealth/bin/gnoic -a ${MGMT_IP} --port 57400 --insecure -u
cisco -p <password>
healthz get --path "openconfig:/components/component[name=${OC_COMP}]"

WARN[0000] "192.0.1.0" could not lookup hostname: lookup 198.51.100.0.in-addr.arpa.
on 171.70.168.183:53: no such host
target "192.0.1.0:57400":
  path      : openconfig:components/component[name=0_RP0_CPU0-appmgr]
  status    : STATUS_HEALTHY
  id        : 1721815321290718105
  acked     : false
  created   : 2024-07-24 10:02:01.290718105 +0000 UTC
  expires   : 2024-07-31 10:02:01.000290718 +0000 UTC
Router# cd /tmp/
Router/tmp#
Router/tmp# /auto/appmgr/xrhealth/bin/gnoic -a ${MGMT_IP} --port 57400 --insecure
-u cisco -p <password>
healthz artifact --id 0_RP0_CPU0-appmgr-1721815320614225976-
58c4d59caf2e8bd971715eea491048673bflaf290fade112ad0ece654e285568
WARN[0000] "192.0.1.0" could not lookup hostname: lookup 198.51.100.0.in-addr.arpa.
on 171.70.168.183:53: no such host

```

```

INFO[0000] 192.0.1.0:57400: received file header for artifactID:
0_RP0_CPU0-appmgr-1721815320614225976-
58c4d59caf2e8bd971715eea491048673bflaf290fade112ad0ece654e285568
id:
"0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bflaf290fade112ad0ece654e285568"
file: {
  name: "procmgr_event_20240724100205.tar.gz"
  path:
"/harddisk:/eem_ac_logs/xrhealth/artifacts/procmgr_event_20240724100205.tar.gz"
  mimetype: "application/gzip"
  size: 3825
  hash: {
    method: SHA256
    hash:
"\xf5\xa5\xfe]\xc1~Y\xbc-\xe4\xfcJ\xe9r\xb4\xe\x2\x6\x0fvk\x90\xf52\r\xe6J\x83\x80\xc9\xff"

  }
}
INFO[0000] received 3825 bytes for artifactID:
0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bflaf290fade112ad0ece654e285568

INFO[0000] 192.0.1.0:57400: received trailer for artifactID:
0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bflaf290fade112ad0ece654e285568

INFO[0000] 192.0.1.0:57400: received 3825 bytes in total
INFO[0000] 192.0.1.0:57400: comparing file HASH
INFO[0000] 192.0.1.0:57400: HASH OK

```

gRPC Network Security Interface

Table 16: Feature History Table

Feature Name	Release Information	Feature Description
gRPC Network Security Interface	Release 24.4.1	<p>Introduced in this release on: Fixed Systems (8200, 8700)(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*).</p> <p>*This feature is now supported on:</p> <ul style="list-style-type: none"> • 8212-32FH-M • 8711-32FH-M • 88-LC1-12TH24FH-E

Feature Name	Release Information	Feature Description
gRPC Network Security Interface	Release 7.11.1	<p>This release implements authorization mechanisms to restrict access to gRPC applications and services based on client permissions. This is made possible by introducing an authorization protocol buffer service for gRPC Network Security Interface (gNSI).</p> <p>Prior to this release, the gRPC services in the gNSI systems could be accessed by unauthorized users.</p> <p>This feature introduces the following change:</p> <p>CLI:</p> <p>To view the specification of gNSI, see Github repository.</p>

gRPC Network Security Interface (gNSI) is a repository which contains security infrastructure services necessary for safe operations of an OpenConfig platform. The services such as authorization protocol buffer manage a network device's certificates and authorization policies.

This feature introduces a new authorization protocol buffer under gRPC gNSI. It contains gNSI.authz policies which prevent unauthorized users to access sensitive information. It defines an API that allows the configuration of the RPC service on a router. It also controls the user access and restricts authorization to update specific RPCs.

By default, gRPC-level authorization policy is provisioned using [Secure ZTP](#). If the router is in zero-policy mode that is, in the absence of any policy, you can use gRPC authorization policy configuration to restrict access to specific users. The default authorization policy at the gRPC level can permit access to all RPCs except for the gNSI.authz RPCs.

If there is no policy specified or the policy is invalid, the router will fall back to zero-policy mode, in which the default behavior allows access to all gRPC services to all the users if their profiles are configured. If an invalid policy is configured, you can revert it by loading a valid policy using exec command **gnsi load service authorization policy**. For more information on how to create user profiles and update authorization policy for these user profiles, see [How to Update gRPC-Level Authorization Policy, on page 49](#). Using **show gnsi service authorization policy** command, you can see the active policy in a router.

We have introduced the following commands in this release :

- **gnsi load service authorization policy**: To load and update the gRPC-level authorization policy in a router.
- **show gnsi service authorization policy**: To see the active policy applied in a router.



Note When both gNSI and gNOI are configured, gNSI takes precedence over gNOI. If neither gNSI nor gNOI is configured, then tls trsutpoint's data is considered for certificate management.

The following RPCs are used to perform key operations at the system level such as updating and displaying the current status of the authorization policy in a router.

Table 17: Operations

RPC	Description
gNSI.authz.Rotate()	Updates the gRPC-level authorization policy.
gNSI.authz.Probe()	Verifies the authenticity of a user based on the defined policy of the gRPC-level authorization policy engine.
gNSI.authz.Get()	Shows the current instance of the gRPC-level authorization policy, including the version and date of creation of the policy.

How to Update gRPC-Level Authorization Policy

gRPC-level authorization policy is configured by default at the time of router deployment using secure ZTP. You can update the same gRPC-level authorization policy using any of two the following methods:

- Using gNSI Client.
- Using exec command.

Updating the gRPC-Level Authorization Policy in the Router Using gNSI Client

Before you start

When a router boots for the first time, it should have the following prerequisites:

- The gNSI.authz service is up and running.
- The default gRPC-level authorization policy is added for all gRPC services.
- The default gRPC-level authorization policy allows access to all RPCs.

The following steps are used to update the gRPC-level authorization policy:

1. Initiate the **gNSI.authz.Rotate()** streaming RPC. This step creates a streaming connection between the router and management application (client).



Note Only one `gNSI.authz.Rotate()` must be in progress at a time. Any other RPC request is rejected by the server.

2. The client uploads new gRPC-level authorization policy using the **UploadRequest** message.

**Note**

- There must be only one gRPC-level authorization policy in the router. All the policies must be defined in the same gRPC-level authorization policy which is being updated. As `gNSI.authz.Rotate()` method replaces all previously defined or used policies once the **finalize** message is sent.
- The upgrade information is passed to the version and the created_on fields. These information are not used by the gNSI.authz service. It is designed to help you to track the active gRPC-level authorization policy on a particular router.

3. The router activates the gRPC-level authorization policy.
4. The router sends the `UploadResponse` message back to the client after activating the new policy.
5. The client verifies the new gRPC-level authorization policy using separate `gNSI.authz.Probe()` RPCs.
6. The client sends the **FinalizeRequest** message, indicating the previous gRPC-level authorization policy is replaced.

**Note**

It is not recommended to close the stream without sending the **finalize** message. It results in the abandoning of the uploaded policy and rollback to the one that was active before the `gNSI.authz.Rotate()` RPC started.

Below is an example of a gRPC-level authorization policy that allows admins, V1,V2,V3 and V4, access to all RPCs that are defined by the gNSI.ssh interface. All the other users won't have access to call any of the gNSI.ssh RPCs:

```
{
  "version": "version-1",
  "created_on": "1632779276520673693",
  "policy": {
    "name": "gNSI.ssh policy",
    "allow_rules": [{
      "name": "admin-access",
      "source": {
        "principals": [
          "spiffe://company.com/sa/V1",
          "spiffe://company.com/sa/V2"
        ]
      },
      "request": {
        "paths": [
          "/gnsi.ssh.Ssh/*"
        ]
      }
    }],
    "deny_rules": [{
      "name": "sales-access",
      "source": {
        "principals": [
          "spiffe://company.com/sa/V3",
          "spiffe://company.com/sa/V4"
        ]
      },
      "request": {
        "paths": [
          "/gnsi.ssh.Ssh/MutateAccountCredentials",

```

```

        "/gnsi.ssh.Ssh/MutateHostCredentials"
    ]
}
}}
}
}

```

Updating the gRPC-Level Authorization Policy file Using Exec Command

Use the following steps to update the authorization policy in the router.

1. Create the users profiles for the users who need to be added in the authorization policy. You can skip this step if you have already defined the user profiles.

The following example creates three users who are added in the authorization policy.

```

Router(config)#username V1
Router(config-un)#group root-lr
Router(config-un)#group cisco-support
Router(config-un)#secret x
Router(config-un)#exit
Router(config)#username V2
Router(config-un)#group root-lr
Router(config-un)#password x
Router(config-un)#exit
Router(config)#username V3
Router(config-un)#group root-lr
Router(config-un)#password x
Router(config-un)#commit

```

2. Enable **tls-mutual** to establish the secure mutual between the client and the router.

```

Router(config)#grpc
Router(config-grpc)#port 0
Router(config-grpc)#tls-mutual
Router(config-grpc)#certificate-authentication
Router(config-grpc)#commit

```

3. Define the gRPC-level authorization policy.

The following sample gRPC-level authorization policy defines authorization policy for the users V1, V2 and V3.

```

{
  "name": "authz",
  "allow_rules": [
    {
      "name": "allow all gNMI for all users",
      "source": {
        "principals": [
          "*"
        ]
      },
      "request": {
        "paths": [
          "*"
        ]
      }
    }
  ],
  "deny_rules": [

```

```

    {
      "name": "deny gNMI set for oper users",
      "source": {
        "principals": [
          "v1"
        ]
      },
      "request": {
        "paths": [
          "/gnmi.gNMI/Get".
        ]
      }
    },
    {
      "name": "deny gNMI set for oper users",
      "source": {
        "principals": [
          "v2"
        ]
      },
      "request": {
        "paths": [
          "/gnmi.gNMI/Get"
        ]
      }
    },
    {
      "name": "deny gNMI set for oper users",
      "source": {
        "principals": [
          "v3"
        ]
      },
      "request": {
        "paths": [
          "/gnmi.gNMI/Set"
        ]
      }
    }
  ]
}

```

4. Copy the gRPC-level authorization policy to the router.

The following example copies the gNSI Authz policy to the router:

```

-bash-4.2$ scp test.json v1@192.0.2.255:/disk0:/
Password:
test.json
100% 993 161.4KB/s 00:00
-bash-4.2$

```

5. Activate the gRPC-level authorization policy to the router.

The following example loads the policy to the router.

```

Router(config)#gnsi load service authorization policy /disk0:/test.json
Successfully loaded policy

```

Verification

Use the **show gnsi service authorization policy** to verify if the policy is active in the router.

```
Router#show gnsi service authorization policy
Wed Jul 19 10:56:14.509 UTC{
  "version": "1.0",
  "created_on": 1700816204,
  "policy": {
    "name": "authz",
    "allow_rules": [
      {
        "name": "allow all gNMI for all users",
        "request": {
          "paths": [
            "*"
          ]
        },
        "source": {
          "principals": [
            "*"
          ]
        }
      }
    ],
    "deny_rules": [
      {
        "name": "deny gNMI set for oper users",
        "request": {
          "paths": [
            "/gnmi.gNMI/*"
          ]
        },
        "source": {
          "principals": [
            "User1"
          ]
        }
      }
    ]
  }
}
```

In the following example, User1 user tries to access the **get** RPC request for which the permission is denied in the above authorization policy.

```
bash-4.2$ ./gnmi_cli -address 198.51.100.255 -ca_cert
certs/certs/ca.cert -client_cert certs/certs/User1.pem -client_key
certs/certs/User1.key -server_name ems.cisco.com -get -proto get-oper.proto
```

Output

```
E0720 14:49:42.277504 26473 gnmi_cli.go:195]
target returned RPC error for Get("path:{origin:"openconfig-interfaces"
elem:{name:"interfaces"}
  elem:{name:"interface" key:{key:"name" value:"HundredGigE0/0/0/0"}}}
  type:OPERATIONAL encoding:JSON_IETF"):
rpc error: code = PermissionDenied desc = unauthorized RPC request rejected
```

gNSI Acctz Logging

Table 18: Feature History Table

Feature Name	Release Information	Feature Description
gNSI Acctz Logging	Release 25.1.1	<p>Introduced in this release on: Fixed Systems (8700 [ASIC: K100])</p> <p>This feature is now supported on:</p> <ul style="list-style-type: none"> 8712-MOD-M
gNSI Acctz Logging	Release 24.3.1	<p>Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100], Centralized Systems (8600 [ASIC:Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100])</p> <p>You can now log and monitor AAA (Authentication, Authorization, and Accounting) accounting of gRPC operations and CLI accounting data through gNSI Acctz for effective management of network for better performance and resource utilization. You can also configure the number of gNSI accounting records that can be streamed.</p> <p>Previously, you could monitor the AAA accounting data through syslog only.</p> <p>The feature introduces these changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> grpc aaa accounting queue-size show gnsi acctz statistics <p>To view the specification of gNSI Accounting (Acctz) RPCs and messages, see the Github repository.</p>

gNSI Acctz Data Logging

The gNSI accounting (Acctz) is a gNSI accounting protocol that collects and transfers accounting records from a router to a remote collection service over a gRPC transport connection.

Starting from Release 24.3.1, you can log gRPC AAA accounting data through gNSI accounting (Acctz). The gNSI Acctz data is logged, stored in accounting records, and sent to gNSI client for monitoring purposes. These gNSI Acctz accounting records contain

- users' login or logout times,
- network access resources such as interface IP and port, and
- duration of each session.

The gNSI Acctz logging can be done using the RecordSubscribe() gRPC request to a router. For more information on the RecordSubscribe() RPC, see the [GitHub](#) repository.

gNSI Acctz Logging Stream Capacity

The gNSI Acctz logs are recorded in a queue, maintaining a history of the 10 most recent records. When the accounting queue is full and no gNSI Acctz collectors are connected, the stream drops the records. Besides the 10 records stored for streaming, up to 512 additional records are stored during processing. As new records arrive, the data stream continues until the gNSI session ends or an error occurs, such as a client disconnection due to network issues or the server going down. If the server's output buffer remains full for an extended period, new records are dropped until the collector starts receiving them.

When the queue reaches its full capacity, the system automatically replaces the oldest records with the newest ones. The router then transmits this logged information through gNSI to gNSI client for real-time monitoring purposes. You can configure the queue size using the **grpc aaa accounting queue-size** command.

Supported Records for gNSI Acctz Logging

gNSI Acctz logging system supports Command and gRPC service records.

Table 19: CLI and gRPC Accounting Records

Command Services Accounting Records	gRPC Services Accounting Records
<p>The command accounting records are generated for the commands executed in CLI mode and sent to gNSI Acctz collectors. The details logged include:</p> <ul style="list-style-type: none"> • Session Info: remote/local IP addresses, remote/local ports, and channel ID. • Authentication details: Identity, privilege level, authentication status (PERMIT/DENY), and the cause of denial (if applicable). • Command and Command status: authentication status (PERMIT/DENY). • Timestamp: The time when the event was generated. 	<p>The gRPC accounting records are generated for the RPCs executed by gRPC services and sent to gNSI Acctz collectors. The details logged include:</p> <ul style="list-style-type: none"> • Session Info: remote/local IP addresses, remote/local ports, and channel ID. • Authentication details: Identity and privilege level. • RPC Service Request: Service type, RPC name, payload, and configuration metadata. • gRPC Service Status: PERMIT/DENY. • Timestamp: The time at which the event was generated.

Default Behavior and Verification of gNSI Acctz Logging

By default, gNSI Acctz records are logged when the [configuration](#) is enabled. You can verify the gNSI Acctz using **show gnsi state**, **show gnsi acctz statistics**, and **show aaa accounting statistics** commands.

Configure gNSI Acctz Logging

Monitor AAA information through gNSI Acctz logs.

Procedure

Step 1 Monitor gNSI state in the router.

Example:

Configure gNSI Acctz Logging

```

Router# show gnsi state
Wed Jun 26 09:26:39.035 UTC
-----GNSI state-----
Global:
  Main Thread cerrno      : Success
  Acctz Thread cerrno     : Success
  State                   : Active
  RDSFS State              : Active

```

Step 2 Obtain gRPC port number.**Example:**

```

show grpc
Tue Aug 13 14:21:50.995 IST

Server name                : DEFAULT
Address family             : dual
Port                       : 57400

Service ports
  gNMI                     : none
  P4RT                     : none
  gRIBI                    : none

DSCP                       : Default
TTL                        : 64
VRF                        :
Server                     : enabled
TLS                        : disabled
TLS mutual                 : disabled
Trustpoint                 : none
Certificate Authentication : disabled
Certificate common name    : ems.cisco.com
TLS v1.0                   : disabled
Maximum requests           : 128
Maximum requests per user  : 10
Maximum streams            : 32
Maximum streams per user   : 32
Maximum concurrent streams : 32
Memory limit (MB)          : 1024
Keepalive time             : 30
Keepalive timeout          : 20
Keepalive enforcement minimum time : 300

TLS cipher suites
  Default                  : none
  Default TLS1.3           : aes_128_gcm_sha256
                           : aes_256_gcm_sha384
                           : chacha20_poly1305_sha256

  Enable                   : none
  Disable                  : none

  Operational enable       : none
  Operational disable      : none
Listen addresses           : ANY

```

Step 3 Configure gNSI queue size.**Example:**

```

Router# configure
Router(config)# grpc aaa accounting queue-size 30
Router(config)# end

```


Step 4 Monitor gNSI Acctz statistics in the router.**Example:**

```
Router# show gnsi acctz statistics
Tue Aug 13 05:57:24.210 UTC
SentToAAA Queue:
  Grpc services:
    GNMI:    4998 sent, 0 dropped
    GNOI:    0 sent, 0 dropped
    GNSI:    2 sent, 0 dropped
    GRIBI:   0 sent, 0 dropped
    P4RT:    0 sent, 0 dropped
    UNSPECIFIED: 0 sent, 0 dropped
  Stats:
    Total Sent: 5000
    Total Drops: 0

Streams:
  Grpc services:
    GNMI:    4996 sent, 2 dropped
    GNOI:    0 sent, 0 dropped
    GNSI:    1 sent, 0 dropped
    GRIBI:   0 sent, 0 dropped
    P4RT:    0 sent, 0 dropped
    UNSPECIFIED: 0 sent, 0 dropped
  Stats:
    Total Sent: 4997
    Total Drops: 2
  Cmd services:
    CLI:      3 sent, 0 dropped
  Stats:
    Total Sent: 3
    Total Drops: 0
Router#
```

Step 5 Provide port and IP address to the Acctz gNSI client.**Example:**

```
acctz_collector -server_addr 192.0.2.111:57400 -username <user name> -password <password> -dieafter 600

----- gNSI Remote Collector -----
2024/08/25 22:59:13 Connecting to gNSI Server.
2024/08/25 22:59:13 gNSI Server connected.
2024/08/25 22:59:13 Started new acctz client.
2024/08/25 22:59:13 Initiate Acctz RecordSubscribe with server .
2024/08/25 22:59:13 Stream started
2024/08/25 22:59:13 Waiting for response from server.
```

Step 6 Verify the accounting record from the router.**Example:****gNSI Acctz RPC RecordSubscribe() response to the Acctz gRPC client**

```
session_info:
{
  local_address:"192.0.2.111"
  local_port:57400
  remote_address:"192.0.2.1"
  remote_port:44374
  ip_proto:6
  user:
```

```

    {
      identity:"lab"
    }
  }
  timestamp:
  {
    seconds:1718971022 nanos:105825300
  }
  grpc_service:
  {
    service_type:GRPC_SERVICE_TYPE_GNSI
    rpc_name:"/gnsi.acctz.v1.AcctzStream/RecordSubscribe" payload_istruncated:true
    authz:
    {
      status:AUTHZ_STATUS_PERMIT
    }
  }
}

```

AAA Accounting Statistics

```

Router# show aaa accounting statistics
Sat Aug 17 17:10:43.055 UTC
Successfully logged events:
Total events: 0
XR CLI: 0
XR SHELL: 0
GRPC:
GNMI: 0
GNSI: 2
GNOI: 0
GRIBI: 0
P4RT: 0
SLAPI: 0
NETCONF: 0
SysAdmin:
CLI: 0
SHELL: 0
Host:
SHELL: 0

Errors:
Invalid requests: 0

Max. records in buffer: 100
Total records in buffer: 0
Router#

```

Data logging with gNSI AcctzStream service

Table 20: Feature History Table

Feature Name	Release Information	Feature Description
Data logging with gNSI AcctzStream service	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100]) This feature is now supported on: <ul style="list-style-type: none"> 8712-MOD-M

Feature Name	Release Information	Feature Description
Data logging with gNSI AcctzStream service	Release 24.4.1	<p>Introduced in this release on: Fixed Systems(8200, 8700); Centralized Systems (8600); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]).</p> <p>This feature replaces the existing bi-directional data streaming service, Acctz, with the new server-streaming service, AcctzStream. In this unidirectional data streaming service, while the router continues to send accounting records to the collector (until the connection is terminated), the collector, on the other hand, sends only the timestamp on its initial connection with the router. With this feature, you can configure the maximum memory allocated for cached accounting history records thus ensuring effective network optimization and resource utilization.</p> <p>The feature introduces these changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> • grpc aaa accounting history-memory <p>For specifications on the gNSI Accounting (AcctzStream) RPCs and messages, refer to the Github repository.</p>

Starting from Cisco IOS XR Release 24.4.1, the **gNSI AcctzStream** server-streaming service is used to collect and transfer accounting records from a router to a remote collection service over a gRPC transport connection, similar to the deprecated **gNSI Acctz** protocol.

The collectors request for logs using the [RecordSubscribe\(\)](#) gRPC from the gNSI AcctzStream service running on the router. The logs are sent to the collectors through the [RecordResponse\(\)](#) gRPC.

This feature has introduced the new **grpc aaa accounting history-memory** command used to configure the maximum memory allocated for cached accounting history records. Use this command with the **grpc aaa accounting queue-size** configuration to effectively limit the EMSD memory used by cached accounting history records.

Configure gNSI AcctzStream logging

Monitor AAA information through gNSI AcctzStream logs.

Procedure

Step 1 Monitor gNSI state in the router.

Example:

```
Router# show gnsi state
Thu Sep 12 12:06:44.035 UTC
-----GNSI state-----
Global:
  Main Thread cerrno      : Success
  Acctz Thread cerrno     : Success
  State                   : Active
  RDSFS State             : Active
```

Step 2 Obtain gRPC port number.**Example:**

```

show grpc
Thu Sep 12 13:23:06.022 UTC

Server name           : DEFAULT
Address family        : dual
Port                  : 57400

Service ports
  gNMI                 : none
  P4RT                 : none
  gRIBI                : none

DSCP                  : Default
TTL                   : 64
VRF                   :
Server                : enabled
TLS                   : disabled
TLS mutual            : disabled
Trustpoint            : none
Certificate Authentication : disabled
Certificate common name : ems.cisco.com
TLS v1.0              : disabled
Maximum requests      : 128
Maximum requests per user : 10
Maximum streams       : 32
Maximum streams per user : 32
Maximum concurrent streams : 32
Memory limit (MB)     : 1024
Keepalive time        : 30
Keepalive timeout     : 20
Keepalive enforcement minimum time : 300

TLS cipher suites
  Default              : none
  Default TLS1.3       : aes_128_gcm_sha256
                       : aes_256_gcm_sha384
                       : chacha20_poly1305_sha256

  Enable               : none
  Disable              : none

  Operational enable   : none
  Operational disable  : none
Listen addresses      : ANY

```

Step 3 Configure gNSI history memory.**Example:**

```

Router# configure
Router(config)# grpc aaa accounting history-memory 20
Router(config)# end

```

Step 4 Configure gNSI queue size.**Example:**

```

Router# configure
Router(config)# grpc aaa accounting queue-size 30
Router(config)# end

```

Step 5 Monitor gNSI Acctz statistics in the router.**Example:**

```
Router# show gnsi acctz statistics
Thu Sep 12 13:56:18.043 UTC
SentToAAA Queue:
  Grpc services:
    GNMI:      4998 sent, 0 dropped
    GNOI:      0 sent, 0 dropped
    GNSI:      2 sent, 0 dropped
    GRIBI:     0 sent, 0 dropped
    P4RT:      0 sent, 0 dropped
    UNSPECIFIED: 0 sent, 0 dropped
  Stats:
    Total Sent: 5000
    Total Drops: 0

Streams:
  Grpc services:
    GNMI:      4996 sent, 2 dropped
    GNOI:      0 sent, 0 dropped
    GNSI:      1 sent, 0 dropped
    GRIBI:     0 sent, 0 dropped
    P4RT:      0 sent, 0 dropped
    UNSPECIFIED: 0 sent, 0 dropped
  Stats:
    Total Sent: 4997
    Total Drops: 2
  Cmd services:
    CLI:       3 sent, 0 dropped
  Stats:
    Total Sent: 3
    Total Drops: 0
Router#
```

Step 6 Provide port and IP address to the Acctz gNSI client.**Example:**

```
acctz_collector -server_addr 192.0.2.111:57400 -username <user name> -password <password> -dieafter 600

----- gNSI Remote Collector -----
2024/08/25 22:59:13 Connecting to gNSI Server.
2024/08/25 22:59:13 gNSI Server connected.
2024/08/25 22:59:13 Started new acctz client.
2024/08/25 22:59:13 Initiate Acctz RecordSubscribe with server .
2024/08/25 22:59:13 Stream started
2024/08/25 22:59:13 Waiting for response from server.
```

Step 7 Verify the accounting record from the router.**Example:****gNSI AcctzStream RPC RecordSubscribe() response to the Acctz gRPC client**

```
session_info:
{
  local_address:"192.0.2.111"
  local_port:57400
  remote_address:"192.0.2.1"
  remote_port:44374
  ip_proto:6
  user:
```

```

    {
      identity:"lab"
    }
  }
timestamp:
{
  seconds:1718971022 nanos:105825300
}
grpc_service:
{
  service_type:GRPC_SERVICE_TYPE_GNSI
  rpc_name:"/gnsi.acctz.v1.AcctzStream/RecordSubscribe" payload_istruncated:true
  authz:
  {
    status:AUTHZ_STATUS_PERMIT
  }
}

```

AAA Accounting Statistics

Router# **show gnsi accounting statistics**

Acctz History Buffer:

Total record: 200

Total history truncation: 10

Cmd service records:

Shell: 0

Cli: 0

Netconf: 0

Grpc service records:

GNMI: 0

GNOI: 0

GNSI: 0

GRIBI: 0

P4RT: 0

History Snapshot:

Max Memory size: 200 MB

Memory used: 8 MB

Max number of records: 100

Total number of records present: 16

gRPC Accounting Queue:

Grpc services:

GNMI: 0 sent, 0 dropped, 0 truncated

GNOI: 0 sent, 0 dropped, 0 truncated

GNSI: 0 sent, 0 dropped, 0 truncated

GRIBI: 0 sent, 0 dropped, 0 truncated

P4RT: 0 sent, 0 dropped, 0 truncated

Stats:

Queue buffer size: 100 MB

Queue buffer used: 0 MB

Queue size: 1

Queue used: 0

Queue enqueue: 0

Queue dequeue: 0

Queue drops: 0

Queue max time: 0 usec

Queue min time: 0 usec

Queue avg time: 0 usec

Errors:

Queue init failure: 0

Queue update failure: 0

Queue dequeue failure: 0

Queue invalid parameters: 0

Queue memory limit: 0

```
Queue size limit: 0
SendtoAAA Accounting Queue:
  Grpc services:
    GNMI: 0 sent, 0 dropped, 0 truncated
    GNOI: 0 sent, 0 dropped, 0 truncated
    GNSI: 0 sent, 0 dropped, 0 truncated
    GRIBI: 0 sent, 0 dropped, 0 truncated
    P4RT: 0 sent, 0 dropped, 0 truncated
  Stats:
    Queue buffer size: 100 MB
    Queue buffer used: 0 MB
    Queue size: 1
    Queue used: 0
    Queue enqueue: 0
    Queue dequeue: 0
    Queue drops: 0
    Queue max time: 0 usec
    Queue min time: 0 usec
    Queue avg time: 0 usec
  Errors:
    Queue init failure: 0
    Queue update failure: 0
    Queue dequeue failure: 0
    Queue invalid parameters: 0
    Queue memory limit: 0
    Queue size limit: 0
Cmd Accounting Queue:
  Cmd services:
    Shell: 0 sent, 0 dropped, 0 truncated
    Cli: 0 sent, 0 dropped, 0 truncated
    Netconf: 0 sent, 0 dropped, 0 truncated
  Stats:
    Queue buffer size: 100 MB
    Queue buffer used: 0 MB
    Queue size: 1
    Queue used: 0
    Queue enqueue: 0
    Queue dequeue: 0
    Queue drops: 0
    Queue max time: 0 usec
    Queue min time: 0 usec
    Queue avg time: 0 usec
  Errors:
    Queue init failure: 0
    Queue update failure: 0
    Queue dequeue failure: 0
    Queue invalid parameters: 0
    Queue memory limit: 0
    Queue size limit: 0
```

gNSI Credentialz Update

Table 21: Feature History Table

Feature Name	Release Information	Description
gNSI Credentialz Update	Release 24.2.11	<p>To improve communication confidentiality and security, you can now update or rotate account-specific and host-specific SSH credentials on a router. You can access the latest SSH credentials through the gNMI credentialz RPC. The updated SSH credentials encompass passwords, host keys, and certificates.</p> <p>To view the specification of gNSI credentialz RPCs and messages, see the Github repository.</p>

Rotation is the process of changing or updating SSH credentials such as passwords, keys, or certificates in a network. You can now update the account-related and host-related SSH credentials through the gNSI credentialz RPC when the router is up and running.

gNSI Rotate Credentialz RPC

Starting from Release 24.2.1, Cisco IOS XR supports four RPCs to change the existing SSH credentials.

gNSI Rotate Credentialz RPC	Run This When	For More Information
RotateAccountCredentials	<p>You want to specify an SSH authentication service policy for the network element.</p> <p>If the policy is valid, it replaces the existing policy.</p>	See, RotateAccountCredentials
RotateHostParameters	<p>You want to change both the Certificate Authority (CA) public key and the key and certificate used by the SSH server.</p>	See, RotateHostParameters
CanGenerateKey	<p>You want to check whether the target can generate a public or private key pair.</p>	See, CanGenerateKey
GetPublicKeys	<p>You want to get the current public keys from the host. It returns each configured key in the provided list.</p>	See, GetPublicKey

Rotate Account Credentials

This RPC automates secure credential rotation on routers, updating passwords and SSH keys to enforce security and prevent unauthorized access. It updates the user-specific authorized keys, authorized principles, invalidates old credentials, logs activities, and notifies stakeholders, enhancing overall network security.

Prerequisites

- Configure a user account on your router.
- Configure SSH Version 2.

The following table outlines the messages that `Rotate Account Credentials` RPC supports, along with their descriptions.

Message	Description
AuthorizedKeysRequest	<p>This message defines the authorized key list for password-less SSH accepted by the router's SSH service.</p> <p>The gNSI client dispatches an <code>AuthorizedKeysRequest</code> to the router to update or replace credentials on the SSH service. The router responds with a <code>AuthorizedKeysResponse</code> message to the gNSI client.</p> <p>It supports the following keys:</p> <ul style="list-style-type: none"> • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519
AuthorizedUsersRequest	<p>This message performs a user authorization check. User authorization can be done using both static and dynamic methods.</p> <p><u>Static Authorization:</u> You can perform static authorization based on a principal name (unique identifier for a user) using Cisco SSH. For static authorization, use the <code>AuthorizedUsersRequest</code> message.</p> <p><u>Dynamic authorization:</u> For dynamic authorization, use the <code>AuthorizedPrincipalCheckRequest</code> message. For details, see Rotate Host Parameters, on page 65</p> <p>CiscoSSH supports the user authorization using <code>AuthorizedPrincipalsFile</code>. <code>AuthorizedPrincipalsFile</code> contains pairs of account names and their corresponding principal names that the router recognizes for certificate-based authentication. For more details, see AuthorizedPrincipalsFile</p>

Rotate Host Parameters

The `RotateHostParameters` RPC updates and verifies host account credentials on network devices to enhance security and ensure stable SSH access. If updates fail, the system either adopts new credentials after successful validation or reverts to the old ones to maintain uninterrupted access. The router automatically falls back to prevent lockouts and preserve network integrity.

Prerequisites

- Configure a user account on your router.
- Configure SSH Version 2.

The following table outlines the messages that `Rotate Host Parameters` RPC supports, along with their descriptions.

Message	Description
CA public key	<p>The <code>CA public key</code> message to verify the gNSI client certificates presented during connection establishment.</p> <p>Without Host Identity Based Authorization (HIBA), the following keys are supported:</p> <ul style="list-style-type: none"> • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519
Server keys	<p>The <code>Server keys</code> message includes host keys and router certificates that serve as credentialz for the gNSI client.</p> <p>If the host keys are generated externally, they must be specified in the <code>Server keys</code> request.</p> <p>It supports the following keys:</p> <ul style="list-style-type: none"> • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519 <p>It supports the following router certificates:</p> <ul style="list-style-type: none"> • Router certificates with HIBA Support <ul style="list-style-type: none"> • <code>ssh-rsa-cert-v01@openssh.com</code> • Router certificates without HIBA support: <ul style="list-style-type: none"> • <code>ecdsa-sha2-nistp256-cert-v01@openssh.com</code> • <code>ecdsa-sha2-nistp521-cert-v01@openssh.com</code> • <code>ssh-ed25519-cert-v01@openssh.com</code> • <code>rsa-sha2-256-cert-v01@openssh.com</code> • <code>rsa-sha2-512-cert-v01@openssh.com</code>
Generate key	<p>The <code>Generate Key</code> message is used for host key management in SSH. When the host keys are generated by the router, this message triggers the creation of new host keys for SSH host key management. The <code>Generate key</code> message supports the following keys:</p> <p>It supports the following keys:</p> <ul style="list-style-type: none"> • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519

Message	Description
AllowedAuthenticationRequest	<p>The <code>AllowedAuthenticationRequest</code> message specifies the permissible authentication methods for the gNSI client authentication.</p> <p>The supported authentication methods are as follows:</p> <ul style="list-style-type: none"> • Keyboard interactive • Password-based • Pubkey-based <ul style="list-style-type: none"> • OpenSSH certificate-based • Public key-based <p>By default, the SSH server allows all authentication methods.</p>
AuthorizedPrincipalCheckRequest	<p>The <code>AuthorizedPrincipalCheckRequest</code> message supports the dynamic authorization of the user against the principal name using the OpenSSH or CiscoSSH.</p> <p>Setting the <code>TOOL_HIBA_DEFAULT</code> flag prompts the router to use the HIBA binary for dynamic authorization. Un setting the <code>HIBA_DEFAULT</code> flag switches the router to use a static authorization.</p> <p><u>Dynamic Authorization:</u> You can enforce the user for authorization check using HIBA.</p> <p>Note The support is only for ssh-rsa-cert-v01@openssh.com</p> <p>CiscoSSH supports <code>AuthorizedPrincipalCheck</code> using <code>AuthorizedPrincipalsCommand</code> and <code>AuthorizedPrincipalsCommandUser</code></p> <p><u>AuthorizedPrincipalsCommand:</u></p> <p>This command generates the list of allowed certificate principals by executing a HIBA binary (By setting the <code>TOOL_HIBA_DEFAULT</code> flag).</p> <p><u>AuthorizedPrincipalsCommandUser:</u></p> <p>This command specifies the user account under which the system executes the <code>AuthorizedPrincipalsCommand</code>. For more details on the specification, see AuthorizedPrincipalsCommandUser</p>

CanGenerateKey

This RPC checks if the router can generate a public or private key pair.

It supports the following key pairs:

- RSA 2048, RSA 4096 bits
- ECDSA-p-256, ECDSA-p-521
- Ed25519

GetPublicKey

This RPC gets the available public keys from the router and displays them. It supports the following keys:

- RSA 2048, RSA 4096 bits
- ECDSA-p-256, ECDSA-p-521
- Ed25519

Manage certificates using Certz.proto

Table 22: Feature History Table

Feature Name	Release Information	Feature Description
Manage certificates using Certz.proto	Release 24.1.1	<p>Instead of using multiple RPCs, Certz.proto provides a bidirectional Rotate RPC to replace, revoke, or load a certificate. It also provides additional APIs to install Public Key Infrastructure (PKI) entities such as like identity certificates, trust-bundles, and Certificate Revocation Lists (CRLs) for a gRPC Server.</p> <p>This feature introduces the following changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> • grpc gnsi service certz ssl-profile-id • show grpc certificate <p>Yang Data Models:</p> <ul style="list-style-type: none"> • Cisco-IOS-XR-man-ems-cfg.yang (see Github, YANG Data Models Navigator)

Certz RPCs

The Certz RPCs are specific methods used for executing operations on the certificate that resides in the target device. The **certz.proto** file is available in the [Github](#) repository.

In cert.proto, a certificate identifier differentiates between leaf certificates. However, the CA bundle lacks an identifier, meaning a new request to load a bundle could overwrite the existing one. On the other hand, in certz.proto, entities like Certificate, CA bundle, key, CRL, and authentication policy are tied to a unique SSL profile.

In cert.proto, a certificate identifier differentiates between leaf certificates. However, the CA bundle lacks an identifier, meaning a new request to load a bundle could overwrite the existing one. On the other hand, in certz.proto, entities like Certificate, CA bundle, key, CRL, and authentication policy are tied to a unique SSL profile.

The certz.proto differs from the cert.proto in the way that it handles the upload of all entities. While in cert.proto, separate RPCs are used to replace, load, and revoke a certificate, in certz.proto, a single Rotate() RPC is used to upload all entities at once. This includes the certificate, the key, the CA bundle, and the CRL.

In addition to these features, certz.proto also provides support for different cryptographic algorithms, including Rivest-Shamir-Adleman (RSA), Elliptic Curve Digital Signature Algorithm (ECDSA), and ED25519, a public-key signature system.

These functionalities make certz.proto a comprehensive solution for managing SSL profiles, providing a streamlined process for handling cryptographic entities and algorithms.



Note If neither cert.proto nor certz.proto is configured, then tls trustpoint data is considered for certificate management.

The following table describes the RPCs supported under Certz.proto.

Table 23: Certz RPCs

RPC	Description
AddProfile	<p>AddProfile is part of SSL profile management. It allows adding a new SSL profile. When an SSL profile is added, all its elements, that is, certificate, CA trusted bundle and a set of certificate revocation lists are NULL/Empty. So, before an SSL profile can be used these entities have to be 'rotated' using the 'Rotate()' RPC.</p> <p>Note An attempt to add an already existing profile is rejected with an error.</p>
Rotate	<p>Rotate replaces/adds an existing device certificate and/or CA certificates (trust bundle) or/and a certificate revocation list bundle on the target. The new device certificate can be created from a target-generated or client-generated CSR (Certificate Signing Request). In the latter case, the client must provide the corresponding private key with the signed certificate.</p>
DeleteProfile	<p>DeleteProfile is part of SSL profile management. It allows for removing an existing SSL profile.</p> <p>Note An attempt to delete a not existing profile results in an error. The profile used by the gRPC server can't be deleted and an attempt to remove it will be rejected with an error.</p>
GetProfileList	<p>GetProfileList is part of SSL profile management. It allows for retrieving a list of IDs of SSL profiles present on the target.</p>
CanGenerateCSR	<p>An RPC to ask a target if it can generate a CSR.</p>

SSL Profile

An SSL profile is a named set of SSL settings that determine how end-user systems connect to or from SSL-based applications or interfaces. The settings in an SSL profile include information about the version of SSL/TLS to be used, certificates, keys, and other parameters related to SSL/TLS communication. By using profiles, administrators can manage and apply these settings more easily across multiple applications or connections.

Here are some key-points regarding SSL profile:

- SSL profiles logically groups certificate, private key, Certificate Authority chain of certificates (a.k.a. a CA trust bundle) and a list of Certificate Revocation Lists into a single set that then can be assigned to a gRPC server.
- There's at least one profile present on a target - the one that is used by the gRPC server. Its ID is gNxI but when the `ssl_profile_id` field in the `RotateCertificateRequest` message isn't set (or set to an empty string) it also refers to this SSL profile by default.
- You can't remove the gRPC SSL profile (gNxI).

Configure gNSI Certz

Before you begin

- Ensure you've created and stored SSL-Profile at `cd/misc/config/grpc/gnsi/certz/ssl_profiles/`

Procedure

Step 1 Create SSL-Profile using `AddProfile` RPC.

Step 2 Rotate SSL-profile using `Rotate` RPC. You can't rotate SSL-profile using a command line interface.

Step 3 Activate the profile using `grpc gnsi service certz ssl-profile-id`.

Example:

Router (config-grpc) `#gnsi service certz profile ssl-profile id <ssl-profile-name>`

Step 4 Verify that `certz.proto` is configured using the `show grpc certificate`.

Example:

```
Router#show grpc certificate
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 32 (0x20)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=localhost,O=OpenConfig,C=US
    Validity
      Not Before: Nov  8 08:49:38 2023 GMT
      Not After : Mar 22 08:49:38 2025 GMT
    Subject: CN=ems,O=OpenConfig,C=US
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (4096 bit)
      Modulus:
        00:ea:6a:6c:25:be:9f:15:71:ce:74:89:03:ec:ef:
        0b:3b:de:58:a8:7e:28:b8:cf:b3:82:91:b4:5c:42:
```

```

e7:d8:28:98:35:bd:35:60:a7:4e:f8:77:02:46:5f:
27:a4:16:cf:3c:e3:24:28:69:9c:22:1e:e3:52:96:
71:87:7c:40:0c:1f:dd:30:ea:dc:40:ca:93:00:54:
5e:de:20:54:5b:f4:2f:9f:19:6f:71:61:28:69:3d:
97:26:ab:e1:5f:53:3c:f1:a2:c3:14:f4:01:90:1a:
e3:08:7b:51:c9:5d:aa:6d:eb:99:a4:08:97:d3:72:
8c:86:a3:f3:b3:77:10:72:e7:a9:3b:fc:38:65:3d:
41:1a:f5:cf:3e:a0:d8:17:d6:d5:53:86:49:a3:dc:
cc:3a:d9:6d:46:25:b0:f9:3b:98:fa:2f:98:09:08:
51:ac:2c:b1:43:c4:b7:96:3e:4e:4e:a6:a5:36:1f:
1f:0f:6a:6a:1a:ea:72:6e:74:90:21:05:fb:26:df:
81:0d:96:e7:13:94:62:2b:ce:3c:7c:de:32:f4:d9:
fa:24:ce:f5:b2:0f:d3:f7:4b:6b:ee:bd:cf:ac:a6:
ed:69:37:fc:d3:4f:3b:46:8b:1b:62:4d:3b:60:30:
74:68:50:4e:48:35:5f:15:66:9a:01:7c:37:1f:e1:
5a:8a:d9:c0:2c:3e:12:fd:71:30:13:b8:b7:16:98:
03:27:6d:45:c4:0f:34:fd:f1:aa:29:8e:c1:63:ac:
57:04:f6:a7:83:83:06:45:dc:0f:f9:de:f9:1e:b6:
d8:5a:bc:3a:98:f8:ac:b0:be:3f:87:df:8c:5e:47:
12:ca:77:70:26:14:02:14:79:fa:6f:1f:ab:ee:06:
2c:83:93:e4:22:db:37:83:90:c1:72:5b:36:78:1b:
6d:0a:06:72:76:dc:89:df:86:89:43:54:03:55:bd:
fc:a0:9a:d6:8e:5d:22:87:a2:32:19:35:c8:17:4e:
1c:1b:5e:81:9d:a5:67:9e:a7:ed:06:e8:e2:91:f1:
ae:f9:19:b1:ae:a8:e6:66:14:2c:6d:a6:c3:0f:8b:
7f:ef:c0:60:cb:c2:52:a5:46:1e:a4:20:52:f8:93:
93:2b:02:23:98:90:81:b3:e6:c4:4e:8f:85:a6:ff:
4e:8e:dd:6c:12:ea:db:58:7f:3c:66:c4:38:96:44:
d1:5b:da:c2:66:6a:4e:97:4d:99:59:9f:24:a0:4a:
57:b6:9d:69:22:f7:5a:10:cb:96:bc:58:ca:96:0e:
ab:b0:4d:14:da:03:e1:d3:24:c1:f2:bd:40:32:20:
82:66:4d:78:4b:13:c6:bd:66:a9:83:2f:15:29:7e:
11:95:37
Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Key Usage: critical
    Digital Signature
  X509v3 Extended Key Usage:
    TLS Web Client Authentication, TLS Web Server Authentication
  X509v3 Authority Key Identifier:
    keyid:0A:A8:9A:6A:23:34:AE:CA:96:00:2C:F3:04:38:14:E3:D4:8D:77:BD

  X509v3 Subject Alternative Name:
    DNS, IP Address:64.103.223.56
Signature Algorithm: sha256WithRSAEncryption
b9:89:ec:60:3d:8d:7d:9c:dc:08:56:89:99:44:92:98:45:b6:
97:ba:e3:e5:f2:48:b2:44:8d:db:23:bb:a1:c0:62:79:78:18:
d7:55:f6:4a:67:5b:75:e0:c0:0b:52:51:07:36:d5:6c:c7:67:
48:86:8d:dd:70:1c:9f:7c:a1:7b:aa:a5:4e:e1:ad:cf:4c:e5:
81:db:92:cf:88:70:5a:1c:8d:de:0d:e8:b3:05:de:b9:04:4d:
23:e1:de:66:e5:08:bd:2e:31:0a:07:a6:c0:00:3a:38:2f:00:
cd:cf:be:e2:1f:12:9f:8a:44:8d:2d:24:d5:d3:bb:9e:db:70:
bf:89:ea:0c:31:b4:b2:fc:3d:73:f5:17:09:07:54:ab:2f:23:
cb:66:0e:0e:7a:9e:21:bf:1e:bf:07:f1:fc:09:88:23:4e:2d:
5d:08:35:16:cd:07:df:25:34:7f:42:0a:dc:6f:d0:ec:9d:99:
72:d8:5f:d6:7e:6f:cc:67:4d:d7:b9:b8:c8:56:75:db:56:1e:
03:1b:6d:37:21:4d:e0:f1:e2:80:99:40:24:24:f2:e4:9b:7e:
6c:bc:f7:f9:3a:b6:fc:8e:dd:9a:cd:dd:88:15:d7:46:71:d2:
11:20:86:8f:ea:c5:a8:e8:4e:b6:ef:9b:06:5b:b1:c4:11:36:
38:7a:63:8e:1a:a6:a8:f8:bb:7d:0b:a6:f2:89:49:94:ac:0c:
8b:c4:fc:02:e8:b2:b8:27:bc:70:95:32:83:09:f5:de:68:34:
3f:a4:5a:73:dc:92:15:2c:0e:ab:46:dd:13:06:98:aa:08:2d:
b8:37:a0:52:4b:ba:f7:be:ed:68:cd:fb:67:3b:66:ea:16:85:
61:75:cf:06:85:a0:06:e8:4a:3e:63:72:c1:79:c7:fd:d4:85:

```

```

74:d8:ea:66:d3:42:74:e2:fb:7c:9e:93:4b:24:2f:ad:c5:13:
bc:eb:83:f7:6d:3e:53:9a:ec:16:85:b7:b5:6c:77:48:53:7e:
19:2e:48:2d:83:35:7b:b9:66:5e:12:b4:f3:ee:e8:b2:3b:ba:
18:46:91:b0:f9:6f:b0:d5:17:a8:de:5c:a0:0e:35:85:7b:c0:
e3:79:06:fa:ad:8e:f2:28:ab:09:19:b7:f0:f3:9e:cb:94:93:
b7:04:63:74:82:c3:71:3b:16:8b:58:c7:fa:ff:ff:2a:97:91:
e7:1d:06:ab:0a:6c:cc:a0:41:31:54:f2:e7:db:a3:b5:22:c4:
ab:ec:e2:5d:86:e6:ac:a5:c6:e2:0e:15:44:a2:32:42:3d:07:
65:0a:0d:58:2e:22:3c:7b:e3:e8:8e:2e:60:47:f0:60:04:89:
64:65:fc:fc:74:dd:4d:7f

```

gNSI EnrollZ and AttestZ

Table 24: Feature History Table

Feature Name	Release Information	Description
TPM Enrollment and Attestation	Release 25.2.1	Introduced in this release on: Centralized Systems (8400 [ASIC: K100]) (select variants only*) *This feature is now supported on the Cisco 8404-SYS-D routers.
TPM Enrollment and Attestation	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100]) This feature is now supported on: <ul style="list-style-type: none">• 8712-MOD-M
TPM Enrollment and Attestation	Release 24.3.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100], Centralized Systems (8600 [ASIC: Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) You can now use the new gNSI service for enrollment and attestation, EnrollZ and AttestZ, to enhance security of networking devices. The EnrollZ has been added to meet open-source requirements, thereby providing advantages such as the verification of device identity and integrity during boot-up, and the provisioning of owner-specific certificates. This bypasses the need for router vendor certificate authorities, offering a user-friendly and secure system. Sensitive credentials are only available to devices that have completed the EnrollZ and AttestZ processes.

Secure TPM Enrollment and Attestation Workflow for Network Devices

The EnrollZ and AttestZ gNSI services provide a secure method for verifying the identity and integrity of network devices. The EnrollZ service handles the TPM 2.0 enrollment workflow, involving cryptographic

verification of the device's TPM-rooted identity and provisioning of attestation and Transport Layer Security (TLS) certificates by the device owner. This ensures that the device is under the control of the owner and not dependent on external vendor Certificate Authorities (CAs) during the attestation process. The AttestZ service manages the TPM 2.0 attestation workflow, confirming the device's integrity throughout the boot process by comparing observed Platform Configuration Register (PCR) values against expected ones to verify the device's boot state. This approach simplifies the TPM enrollment process for device owners, enhances control over certificate management, and eliminates external dependencies, while aligning with Trusted Computing Group (TCG) specifications.

Enroll a TPM 2.0 on Network Devices

The Trusted Platform Module (TPM) 2.0 enrollment workflow is a secure process for network devices to obtain the necessary credentials and configurations for TPM management. This workflow is initiated after the device boot process and involves interaction with various gRPC API endpoints.

Before you begin

- Device has completed the Bootz workflow.
- Device is equipped with a default SSL profile using the Secure Unique Device Identifier (SUDI) key pair and certificate.
- EnrollZ service is available and ready to enroll the TPM on the control card.
- Router owner has access to the trust bundle/anchor from the router vendor.

Procedure

-
- Step 1** Prepare Device for TPM Enrollment: Ensure the device has completed the Bootz workflow and is ready to serve TPM enrollment gRPC API endpoints on the required port.
- Step 2** Trigger EnrollZ Service: Use the `GetIakCert` API to retrieve the Initial Attestation Key (IAK) and IDevID certificates.
- Step 3** Verify and Validate Certificates:
- Verify the signature over the IAK certificate using the trust bundle/anchor from the router vendor.
 - Confirm that the device identity fields in the IAK and IDevID certificates meet the expected criteria.
- Step 4** Request and Install Owner Certificates:
- Request the router owner CA to issue the Owner IAK (oIAK) and Owner IDevID (oIDevID) certificates based on the public keys.
 - Use the `RotateOIakCert` API to install the oIAK and oIDevID certificates on the control card.
- Step 5** Verify and Store Certificates:
- Verify that the public keys in the oIAK and oIDevID certificates match with respective IAK and SUDI public key.
 - Store the oIAK and oIDevID certificates in non-volatile memory for presentation during the TPM attestation (`attestz`) workflow.

- Step 6** Update SSL Profile: Update the SSL profile to use the trust bundle and rotate the certificates to the Owner IDevID certificate.
- Step 7** Enroll Secondary Control Card: Repeat the enrollment workflow for the secondary control card, if present.
-

TPM 2.0 Attestation

The TPM 2.0 attestation workflow ensures the integrity and identity of network devices by verifying their configurations and credentials. This process involves interaction with gRPC TPM 2.0 attestation endpoints and requires the device to be booted with the correct OS image and configurations.

Before you begin

- Device must be booted with the correct OS image.
- Correct configurations and credentials must be applied.
- Primary/active control card is responsible for all RPCs directed to the secondary/standby control card.

Procedure

- Step 1** Serve gRPC TPM 2.0 Attestation Endpoints: Ensure the device serves gRPC TPM 2.0 attestation endpoints on port 9339, the same port as gNOI/gNSI/gNMI.
- The device must be booted with the correct OS image and configurations.
- Step 2** Authenticate Standby Control Card: Perform an authentication handshake between the active and standby control cards using the IDevID key pair/cert.
- The active control card is responsible for this handshake as the router owner cannot directly TLS authenticate the standby card.
- Step 3** Secure Initial Attestation RPCs: Use the active control card's IDevID private key and oIDevID cert to secure TLS for the initial attestation RPCs.
- Step 4** Call AttestZ Service: AttestZ service calls the device's Attest endpoint for a given control card (and a random nonce) to get back:
- An oIAK cert signed by the router owner's CA.
 - Final observed PCR hashes/values.
 - PCR Quote structure and signature over it signed by IAK private key.
 - (Optional) oIDevID cert of the standby control card.
- Step 5** Verify Certificates and Signatures:
- AttestZ service uses the trust bundle/anchor from the router owner CA to verify the oIAK cert and its validity/revocation status.
 - Ensure that the control card serial number in the oIAK cert and oIDevID cert is the same.

- Step 6** Compare PCR Values: The AttestZ service compares the PCR values against the known PCR values provided by the OEM vendor specific to a release.
- Step 7** Compare PCR Values and Record Attestation Status: AttestZ service fetches expected final PCR values from its database and compares them to the observed ones reported by the device.
- AttestZ service records a successful attestation status for the given control card and repeats the workflow for the secondary/standby control card if one is available.

gNOI Containerz

The Containerz - gNOI Container Service on Cisco 8000 series routers introduces a workflow to onboard and manage third-party applications using gNOI RPCs.

Table 25: Feature History Table

Feature Name	Release Information	Feature Description
	Release 25.1.1	<p>Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100]); Centralized Systems (8600 [ASIC:Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100])</p> <p>The Containerz - gNOI service enables the onboarding and management of third-party applications via gNOI RPCs. Applications can be deployed using Docker images, ensuring easy integration and management. The service offers operations such as starting, stopping, updating containers, and managing docker volumes. This feature simplifies application deployment and management, enhancing operational efficiency.</p> <p>For the specification on gNOI.containerz, see the GitHub repository.</p>

The Containerz - gNOI Container Service on Cisco 8000 series routers introduces a workflow to onboard and manage third-party applications using gNOI RPCs.

Containerz offers an interface that facilitates the deployment of containers onto network devices from an external client. Through this interface, you can perform essential operations such as deploying, starting,

stopping, and listing containers. Additionally, it provides functionality to retrieve logs for containers running on the target systems, ensuring seamless monitoring and management of the containerized environments.

For more information regarding application manager, refer [Application Hosting Configuration Guide for Cisco 8000 Series Routers, Cisco IOS XR Releases](#).

Prerequisites

Before utilising the Containerz service, ensure that the router supports gNOI and has the capability to handle gNOI requests.

gNOI Containerz APIs

1. **Deploy Applications:** Use the **Deploy()** RPC to place docker images on the target and load them into the registry.
2. **Manage Containers:** Use **StartContainer()**, **StopContainer()**, and **UpdateContainer()** RPCs to handle the lifecycle of containers.
3. **Volume Management:** Create, list, and remove docker volumes with **CreateVolume()**, **ListVolume()**, and **RemoveVolume()** RPCs.

Table 26: gNOI Containerz RPCs

When the gNOI client sends...	The Router...
Deploy RPC	deploys a container image on the target device, loading it into the registry upon successful reception.
ListImage RPC	returns all container images that match the request specification.
RemoveImage RPC	deletes specified container images, returning an error if associated with a running container.
ListContainer RPC	lists all containers that match the request specifications.
StartContainer RPC	starts a container, returning an error if the image does not exist or if the instance name is already running.
StopContainer RPC	stops a running container, optionally forcing a kill, and returns an error if the container does not exist or is not running.
UpdateContainer RPC	updates a running container to a specified image with synchronous or asynchronous modes.
Log RPC	streams logs from a running container, returning the latest logs if it is not running.
CreateVolume RPC	creates a volume, returning an error if it already exists.
RemoveVolume RPC	removes a specified volume with no operation if the volume does not exist.
ListVolume RPC	lists all volumes that match the specification in the filter.

Deploy and Manage Containers Using ContainerZ RPCs

Use ContainerZ RPCs to deploy images, start containers, and manage volumes.

- Deploy an Image: Streaming

```
Router# ContZ_Client --addr 192.168.122.174:57400 image push
--file /auto/tftp-blr-users4/user/vrf-relay.tar.gz --image vrf-relay --tag 3.10

Pushed vrf-relay/3.10
```

- Deploy an Image: Download

```
Router# ContZ_Client --addr 10.77.132.103:57400 image pull
--path 64.104.151.152:8080/nginx.tar.gz --image nginx --tag 1.4

Pulled nginx /1.4
```

- Remove an Image:

```
Router# ContZ_Client --addr 192.168.122.174:57400 image remove
--image vrf-relay --tag 3.10

Image vrf-relay:3.10 has been removed.
```

- Start a Container:

```
Router# ContZ_Client --addr 192.168.122.174:57400 container
start --command sh --env --net=host --instance vrf-one --image vrf-relay --tag 3.10

Container started with id - vrf-one
```

- Update Container:

```
Router# ContZ_Client --addr 192.168.122.174:57400 container
update --instance vrf-one --image vrf-relay --tag 3.20

Container updated with id - vrf-one
```

- Create a Volume:

```
Router# ContZ_Client --addr 192.168.122.174:57400 volume create
--name User1

Volume "User1" created!
```

- List Images:

```
Router# ContZ_Client --addr 192.168.122.174:57400 image list

Images:
- Name: vrf-relay, Tag: 3.10
- Name: vrf-helper, Tag: 2.5
```

- List Containers:

```
Router# /auto/tftp-blr-users4/user/ContZ_Client --addr
192.168.122.174:57400 container list --all

Containers:
- ID: abc123, Name: vrf-one, Status: RUNNING
- ID: def456, Name: vrf-two, Status: STOPPED
```

- List Volumes:

```
Router# /auto/tftp-blr-users4/user/ContZ_Client --addr
192.168.122.174:57400 volume list

Volumes:
- Name: User1, Driver: DS_LOCAL
- Name: Data_Store_2, Driver: DS_CUSTOM
```

- Stop a Container:

```
Router# /auto/tftp-blr-users4/user/ContZ_Client --addr
192.168.122.174:57400 container stop --instance vrf-one

Container "vrf-one" stopped.
```

- Stream Logs:

```
Router# /auto/tftp-blr-users4/user/ContZ_Client --addr
192.168.122.174:57400 container log --instance vrf-one --follow

Logs:
- [2023-10-01 10:00:00] Starting process...
- [2023-10-01 10:01:00] Process running smoothly.
```

- Remove a Volume:

```
Router# /auto/tftp-blr-users4/user/ContZ_Client --addr
192.168.122.174:57400 volume remove --name User1

Volume "User1" removed.
```



Note All RPCs responses are based on containerz client implementation, this can differ based on your specific containerz client implementation

Show Commands for Containerz

Show App Manager Source Table

```
Router#show appmgr source-table
Sno Name File Installed By
-----
```

```

1  alpine-3.20      Streaming      CONTAINERZ
2  nginx-1.4        http://64.104.151.152:8080/nginx.tar.gz  CONTAINERZ
Router#

```

Show App Manager Application Table

```

Router#show appmgr application-table
Name      Type      Config State Status      Workflow
-----
alpine-cz Docker    Activated Up 19 seconds containerz
nginx-cz   Docker    Activated Up 10 seconds containerz
Router#

```

GISO Workflow for Migration from Legacy Systems

The GISO workflow is designed to facilitate the seamless migration of third-party applications from legacy systems to Containerz environment. This process ensures that applications are onboarded efficiently and managed effectively, leveraging both config-driven and containerz workflows. Here are the key aspects of the GISO workflow for migration:

Migrating TPA with Service Files

You can migrate your existing TPAs from legacy environment to containerz environment using these steps

1. **Post GISO Upgrade:** Post GISO Upgrade: After the GISO upgrade, the app manager will bring the container up based on the set of parameters specified in the service file part of TPA RPM.

Sample Service File:

```

type: docker
source: bonnet
application:
- name: "bnt-one"
  run-opts: "-it"
  run-cmd: "sh"
- name: "bnt-two"
  run-opts: "-it --restart=always"
  run-cmd: "sh"

```

2. **Container Image Runtime Operations:** Further runtime operations on the container or image can be performed by the containerz workflow

Migrating TPA without Service Files

The life cycle of such third-party applications will continue to be supported via the legacy app manager config workflow.

P4Runtime

Table 27: Feature History Table

Feature Name	Release Information	Description
P4Runtime to Manage Traffic Operations	Release 7.10.1	<p>With this release, the router supports Programming Protocol-Independent Packet Processors Runtime (P4), a gRPC-based service, to program the data plane elements for network operations such as sending and receiving packets between the router and the P4Runtime controller using packet I/O messages.</p> <p>This feature introduces the following commands:</p> <p>CLI:</p> <ul style="list-style-type: none"> • grpc p4rt • grpc p4rt interface • grpc p4rt location • show p4rt devices • show p4rt interfaces • show p4rt state • show p4rt stats • show p4rt trace <p>YANG Data Model:</p> <p><code>openconfig-p4rt.yang</code> OpenConfig data model (see GitHub, YANG Data Models Navigator)</p>

P4Runtime is a control plane specification to manage the data plane elements of a device. It defines the navigation and management of packets through data plane blocks using P4Runtime APIs. These blocks can be managed to perform the following set of traffic operations between the P4Runtime controller and the router:

- Send or receive packets using PacketOut and PacketIn I/O messages—`StreamMessageRequest`, `StreamMessageResponse` and `StreamError` messages.
- Elect the primary controller using the `MasterArbitrationUpdate` message.
- Read and write forwarding table entries, protocol headers, counters, and other P4 entities.

For more information about how controllers can connect to the router and program P4-defined functionalities, see [P4RT specification](#).

Configure P4RT to Manage Packets

Configure P4RT to send or receive packets between one or more controllers and the router.

Procedure

Step 1 Enable P4Runtime.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#p4rt
Router(config-grpc-p4rt)#commit
```

Step 2 Assign a unique P4 numeric identifier to the required physical port on the router. The controller uses this port ID as an alias to identify the interface through which the packets are sent or received with ingress or egress metadata.

Example:

```
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/24 port-id 3
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/25 port-id 6
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/26 port-id 7
```

The `port-id` is a unique 32-bit identifier. The range is 1 to 4294967039.

Step 3 Assign a unique P4 device identifier to each Network Processing Unit (NPU) in the system.

Example:

```
Router(config-grpc-p4rt)#location 0/0/CPU0 npu-id 0 device-id 1000000
Router(config-grpc-p4rt)#location 0/0/CPU0 npu-id 1 device-id 1000001
Router(config-grpc-p4rt)#location 0/1/CPU0 npu-id 0 device-id 1000002
Router(config-grpc-p4rt)#location 0/1/CPU0 npu-id 1 device-id 1000011
Router(config-grpc-p4rt)#commit
Router(config-grpc-p4rt)#end
```

The `device-id` is a unique 64-bit identifier. The range is 1 to 18446744073709551615. The `npu-id` represents a NPU identifier within a line card and the value ranges from 0 to 7.

The controller or the P4Runtime agent, which can be external or internal to the router, can use the `port-id` and `device-id` to inject packets and request to send certain packet types. For example, P4Runtime supports the ability to configure Access Control Lists (ACLs) in order to redirect packets with TTL value 1 to the controller. When the router receives a packet with that TTL value, the packet is sent to the controller with the details such as packet received from `device-id x`, `port-id y` and the packet is being sent to `port-id z`.

For more information about programming the router using P4Runtime, see [P4RT specification](#).

IANA Port Numbers For gRPC Services

Table 28: Feature History Table

Feature Name	Release Information	Description
IANA Port Numbers For gRPC Services	Release 24.1.1	<p>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.</p> <p>This feature introduces the following changes:</p> <p>CLI:</p> <ul style="list-style-type: none"> • port (gRPC) • gnmi

IANA (Internet Assigned Numbers Authority) 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 assigned in various ways, based on three ranges: System Ports (0-1023), User Ports (1024-49151), and the Dynamic and/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.

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 will enable 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 will cause the service to use the default or IANA port.

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 will 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) will be accepted. This flexibility allows for seamless communication across different service ports and the general gRPC port.

Starting from Cisco IOS XR Release 24.2.1, the allowed port range is 1024-65535.

Configure gRPC Service-Level Port

To configure a default listening port for the gRPC services such as gNMI, gRIBI, and P4RT, use the respective service command (**gnmi**, **gribi**, or **p4rt**) under the gRPC configuration mode.

To specify a port number for gRPC, gNMI, gRIBI, and P4RT services within the defined range, use the **port** command under respective submodes.



Note

The IANA port ranges are:

- System ports (Reserved): 0—1023
- Registered ports: 1024—49151
- Dynamic or Private or Ephemeral ports: 49152—65535

XR Ephemeral port range: 15232–57343

If the configured port is in the range of IANA registered ports (1024-49151) or XR ephemeral ports (15232-57343), a syslog is generated with a NOTICE to warn the user for a possible application conflict.

Resetting the port reverts to the default service port, and disabling the service stops listening on that port.

Procedure

Configure the port number for a service.

The following examples display the service-level port configurations.

• For gRPCservice:

This configuration creates a gRPC listener with the default or IANA ratified port of 57400.

The allowed range is 1024-65535.

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

Verify the listening port created for gRPC service.

```
Router#show running-config grpc
grpc
!
```

The **port** command under gRPC submode allows the port to be modified in the port range or IANA ratified port.

```
Router# config
Router(config)# grpc port 2000
Router(config)# commit
```

Verify the port number.

```
Router#show running-config grpc
grpc
```

```
port 2000
!
```

- **For gNMI service:**

This configuration creates a gRPC listener with the default or IANA ratified gNMI port of 9339.

The allowed range is 1024-65535.

```
Router(config-grpc) #gnmi
Router(config-grpc-gnmi) #commit
```

Verify the listening port created for gNMI service.

```
Router#show running-config grpc
grpc
  gnmi
!
```

The **port** command under gNMI submode allows the port to be modified in the port range or IANA ratified port.

```
Router(config-grpc) #gnmi
Router(config-grpc-gnmi) #port 9339
Router(config-grpc-gnmi) #commit
```

Verify the port number.

```
Router#show running-config grpc
grpc
  gnmi
    port 9339
!
```

- **For P4RT service:**

This configuration creates a gRPC listener with the default or IANA ratified P4RT port of 9559.

The allowed range is 1024-65535.

```
Router(config-grpc) #p4rt
Router(config-grpc-p4rt) #commit
```

Verify the listening port created for P4RT service.

```
Router#show running-config grpc
grpc
  p4rt
!
```

The **port** command under P4RT submode allows the port to be modified in the port range or IANA ratified port.

```
Router(config-grpc) #p4rt
Router(config-grpc-p4rt) #port 9559
Router(config-grpc-p4rt) #commit
```

Verify the port number.

```
Router#show running-config grpc
grpc
  p4rt
    port 9559
!
```

- **For gRIBI service:**

This configuration creates a gRPC listener with the default or IANA ratified gRIBI port of 9340.

The allowed range is 1024-65535.

```
Router(config-grpc) #gribi
Router(config-grpc-gribi) #commit
```

Verify the listening port created for gRIBI service.

```
Router#show running-config grpc
grpc
  gribi
!
```

The **port** command under gRIBI submode allows the port to be modified in the port range or IANA ratified port.

```
Router(config-grpc) #gribi
Router(config-grpc-gribi) #port 9340
Router(config-grpc-gribi) #commit
```

Verify the port number.

```
Router#show running-config grpc
grpc
  gribi
    port 9340
!
```

Unconfiguring the port command in a service and

Unconfiguring a service under gRPC

- Unconfiguring the **port** command results in using the default port for the respective service.

Example:

Unconfiguring the **port** command will result in a gNMI service using the default gNMI port.

```
Router(config-grpc) #gnmi
Router(config-grpc-gnmi) #no port
Router(config-grpc-gnmi) #commit
```

Verify the service port configuration.

```
Router#show running-config grpc
grpc
  gnmi
!
```

- Unconfiguring a service removes the listener for the respective port and no requests will be accepted on that port.

Example:

Unconfiguring gNMI disables the requests on port 9339.

```
Router(config-grpc) #no gnmi
Router(config-grpc-gnmi) #commit
```

Verify the port configuration.

```
Router#show running-config grpc
grpc
!
```

Configure Interfaces Using Data Models in a gRPC Session

Table 29: Feature History Table

Feature Name	Release Information	Description
Set Limit on Concurrent Streams for gRPC Server	Release 24.1.1	<p>You can prevent potential security attacks by disallowing any single gRPC server client on Cisco IOS XR from consuming excessive resources and monopolizing connection resources, both of which can be potential attack vectors. Such prevention is possible because you now have the option to configure the gRPC server to limit the number of concurrent streams per gRPC connection.</p> <p>The feature introduces the grpc max-concurrent-streams command.</p> <p>YANG Data Models:</p> <ul style="list-style-type: none"> • <code>Cisco-IOS-XR-man-ems-oper.yang</code> • <code>Cisco-IOS-XR-man-ems-cfg.yang</code> <p>(see GitHub, YANG Data Models Navigator)</p>

Google-defined remote procedure call () is an open-source RPC framework. gRPC supports IPv4 and IPv6 address families. The client applications use this protocol to request information from the router, and make configuration changes to the router.

The process for using data models involves:

- Obtain the data models.
- Establish a connection between the router and the client using gRPC communication protocol.
- Manage the configuration of the router from the client using data models.



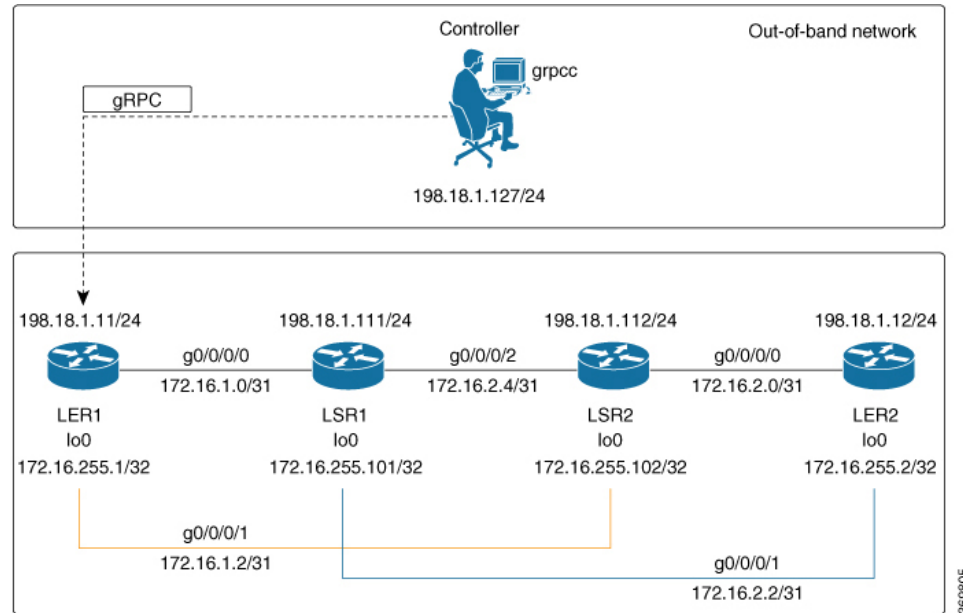
Note Configure AAA authorization to restrict users from uncontrolled access. If AAA authorization is not configured, the command and data rules associated to the groups that are assigned to the user are bypassed. An IOS-XR user can have full read-write access to the IOS-XR configuration through Network Configuration Protocol (NETCONF), google-defined Remote Procedure Calls (gRPC) or any YANG-based agents. In order to avoid granting uncontrolled access, enable AAA authorization using **aaa authorization exec** command before setting up any configuration. For more information about configuring AAA authorization, see the *System Security Configuration Guide*.

In this section, you use native data models to configure loopback and ethernet interfaces on a router using a gRPC session.

Consider a network topology with four routers and one controller. The network consists of label edge routers (LER) and label switching routers (LSR). Two routers LER1 and LER2 are label edge routers, and two routers LSR1 and LSR2 are label switching routers. A host is the controller with a gRPC client. The controller communicates with all routers through an out-of-band network. All routers except LER1 are pre-configured with proper IP addressing and routing behavior. Interfaces between routers have a point-to-point configuration with /31 addressing. Loopback prefixes use the format 172.16.255.x/32.

The following image illustrates the network topology:

Figure 1: Network Topology for gRPC session



You use Cisco IOS XR native model `Cisco-IOS-XR-ifmgr-cfg.yang` to programmatically configure router LER1.

Before you begin

- Retrieve the list of YANG modules on the router using NETCONF monitoring RPC. For more information
- Configure Transport Layer Security (TLS). Enabling gRPC protocol uses the default HTTP/2 transport with no TLS. gRPC mandates AAA authentication and authorization for all gRPC requests. If TLS is not configured, the authentication credentials are transferred over the network unencrypted. Enabling TLS ensures that the credentials are secure and encrypted. Non-TLS mode can only be used in secure internal network.

Procedure

Step 1 Enable gRPC Protocol

To configure network devices and view operational data, gRPC protocol must be enabled on the server. In this example, you enable gRPC protocol on LER1, the server.

Note

Cisco IOS XR 64-bit platforms support gRPC protocol. The 32-bit platforms do not support gRPC protocol.

- a) Enable gRPC over an HTTP/2 connection.

Example:

```
Router#configure
Router(config)#grpc
Router(config-grpc)#port <port-number>
```

The port number ranges from 57344 to 57999. If a port number is unavailable, an error is displayed.

Starting Release 24.1.1, you can now configure IANA port numbers for specified gRPC services. To see the port numbers for the various gRPC services, see *Support IANA Port Numbers*.

- b) Set the session parameters.

Example:

```
Router(config)#grpc {address-family | certificate-authentication | dscp | max-concurrent-streams
| max-request-per-user | max-request-total | max-streams |
max-streams-per-user | no-tls | tlsv1-disable | tls-cipher | tls-mutual | tls-trustpoint |
service-layer | vrf}
```

where:

- **address-family**: set the address family identifier type.
- **certificate-authentication**: enables certificate based authentication
- **dscp**: set QoS marking DSCP on transmitted gRPC.
- **max-concurrent-streams**: set the limit on the maximum concurrent streams per gRPC connection to be applied on the server.
- **max-request-per-user**: set the maximum concurrent requests per user.
- **max-request-total**: set the maximum concurrent requests in total.
- **max-streams**: set the maximum number of concurrent gRPC requests. The maximum subscription limit is 128 requests. The default is 32 requests.
- **max-streams-per-user**: set the maximum concurrent gRPC requests for each user. The maximum subscription limit is 128 requests. The default is 32 requests.
- **no-tls**: disable transport layer security (TLS). The TLS is enabled by default
- **tlsv1-disable**: disable TLS version 1.0
- **service-layer**: enable the grpc service layer configuration.
This parameter is not supported in Cisco ASR 9000 Series Routers, Cisco NCS560 Series Routers, , and Cisco NCS540 Series Routers.
- **tls-cipher**: enable the gRPC TLS cipher suites.
- **tls-mutual**: set the mutual authentication.
- **tls-trustpoint**: configure trustpoint.
- **server-vrf**: enable server vrf.

After gRPC is enabled, use the YANG data models to manage network configurations.

Step 2 Configure the interfaces.

In this example, you configure interfaces using Cisco IOS XR native model `Cisco-IOS-XR-ifmgr-cfg.yang`. You gain an understanding about the various gRPC operations while you configure the interface. For the complete list of operations, see [gRPC Operations, on page 4](#). In this example, you merge configurations with `merge-config` RPC, retrieve operational statistics using `get-oper` RPC, and delete a configuration using `delete-config` RPC. You can explore the structure of the data model using YANG validator tools such as [pyang](#).

LER1 is the gRPC server, and a command line utility `grpc` is used as a client on the controller. This utility does not support YANG and, therefore, does not validate the data model. The server, LER1, validates the data model.

Note

The OC interface maps all IP configurations for parent interface under a VLAN with index 0. Hence, do not configure a sub interface with tag 0.

- a) Explore the XR configuration model for interfaces and its IPv4 augmentation.

Example:

```
controller:grpc$ pyang --format tree --tree-depth 3 Cisco-IOS-XR-ifmgr-cfg.yang
Cisco-IOS-XR-ipv4-io-cfg.yang
module: Cisco-IOS-XR-ifmgr-cfg
  +--rw global-interface-configuration
  | +--rw link-status? Link-status-enum
  +--rw interface-configurations
    +--rw interface-configuration* [active interface-name]
      +--rw dampening
      | ...
      +--rw mtus
      | ...
      +--rw encapsulation
      | ...
      +--rw shutdown? empty
      +--rw interface-virtual? empty
      +--rw secondary-admin-state? Secondary-admin-state-enum
      +--rw interface-mode-non-physical? Interface-mode-enum
      +--rw bandwidth? uint32
      +--rw link-status? empty
      +--rw description? string
      +--rw active Interface-active
      +--rw interface-name xr:Interface-name
      +--rw ipv4-io-cfg:ipv4-network
      | ...
      +--rw ipv4-io-cfg:ipv4-network-forwarding ...
```

- b) Configure a loopback0 interface on LER1.

Example:

```
controller:grpc$ more xr-interfaces-lo0-cfg.json
{
  "Cisco-IOS-XR-ifmgr-cfg:interface-configurations":
  { "interface-configuration": [
    {
      "active": "act",
      "interface-name": "Loopback0",
      "description": "LOCAL TERMINATION ADDRESS",
      "interface-virtual": [
        null
      ],
      "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
        "addresses": {
          "primary": {
```

```

        "address": "172.16.255.1",
        "netmask": "255.255.255.255"
    }
}
}
}
]
}
}

```

- c) Merge the configuration.

Example:

```

controller:grpc$ grpc -username admin -password admin -oper merge-config
-server_addr 198.18.1.11:57400 -json_in_file xr-interfaces-gi0-cfg.json
emsMergeConfig: Sending ReqId 1
emsMergeConfig: Received ReqId 1, Response '
'

```

- d) Configure the ethernet interface on LER1.

Example:

```

controller:grpc$ more xr-interfaces-gi0-cfg.json
{
  "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
    "interface-configuration": [
      {
        "active": "act",
        "interface-name": "GigabitEthernet0/0/0/0",
        "description": "CONNECTS TO LSR1 (g0/0/0/0)",
        "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
          "addresses": {
            "primary": {
              "address": "172.16.1.0",
              "netmask": "255.255.255.254"
            }
          }
        }
      }
    ]
  }
}

```

- e) Merge the configuration.

Example:

```

controller:grpc$ grpc -username admin -password admin -oper merge-config
-server_addr 198.18.1.11:57400 -json_in_file xr-interfaces-gi0-cfg.json
emsMergeConfig: Sending ReqId 1
emsMergeConfig: Received ReqId 1, Response '
'

```

- f) Enable the ethernet interface GigabitEthernet 0/0/0/0 on LER1 to bring up the interface. To do this, delete shutdown configuration for the interface.

Example:

```

controller:grpc$ grpc -username admin -password admin -oper delete-config
-server_addr 198.18.1.11:57400 -yang_path "$(< xr-interfaces-gi0-shutdown-cfg.json )"
emsDeleteConfig: Sending ReqId 1, yangJson {

```

```

"Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
  "interface-configuration": [
    {
      "active": "act",
      "interface-name": "GigabitEthernet0/0/0/0",
      "shutdown": [
        null
      ]
    }
  ]
}
}
}
emsDeleteConfig: Received ReqId 1, Response ''

```

Step 3 Verify that the loopback interface and the ethernet interface on router LER1 are operational.

Example:

```

controller:grpc$ grpc -username admin -password admin -oper get-oper
-server_addr 198.18.1.11:57400 -oper_yang_path "$(< xr-interfaces-briefs-oper-filter.json )"
emsGetOper: Sending ReqId 1, yangPath {
  "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces": {
    "interface-briefs": [
      null
    ]
  }
}
{ "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces": {
  "interface-briefs": {
    "interface-brief": [
      {
        "interface-name": "GigabitEthernet0/0/0/0",
        "interface": "GigabitEthernet0/0/0/0",
        "type": "IFT_ETHERNET",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "ether",
        "encapsulation-type-string": "ARPA",
        "mtu": 1514,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 1000000
      },
      {
        "interface-name": "GigabitEthernet0/0/0/1",
        "interface": "GigabitEthernet0/0/0/1",
        "type": "IFT_ETHERNET",
        "state": "im-state-up",
        "actual-state": "im-state-up",
        "line-state": "im-state-up",
        "actual-line-state": "im-state-up",
        "encapsulation": "ether",
        "encapsulation-type-string": "ARPA",
        "mtu": 1514,
        "sub-interface-mtu-overhead": 0,
        "l2-transport": false,
        "bandwidth": 1000000
      },
      {
        "interface-name": "Loopback0",
        "interface": "Loopback0",

```

```

    "type": "IFT_LOOPBACK",
    "state": "im-state-up",
    "actual-state": "im-state-up",
    "line-state": "im-state-up",
    "actual-line-state": "im-state-up",
    "encapsulation": "loopback",
    "encapsulation-type-string": "Loopback",
    "mtu": 1500,
    "sub-interface-mtu-overhead": 0,
    "l2-transport": false,
    "bandwidth": 0
  },
  {
    "interface-name": "MgmtEth0/RP0/CPU0/0",
    "interface": "MgmtEth0/RP0/CPU0/0",
    "type": "IFT_ETHERNET",
    "state": "im-state-up",
    "actual-state": "im-state-up",
    "line-state": "im-state-up",
    "actual-line-state": "im-state-up",
    "encapsulation": "ether",
    "encapsulation-type-string": "ARPA",
    "mtu": 1514,
    "sub-interface-mtu-overhead": 0,
    "l2-transport": false,
    "bandwidth": 1000000
  },
  {
    "interface-name": "Null0",
    "interface": "Null0",
    "type": "IFT_NULL",
    "state": "im-state-up",
    "actual-state": "im-state-up",
    "line-state": "im-state-up",
    "actual-line-state": "im-state-up",
    "encapsulation": "null",
    "encapsulation-type-string": "Null",
    "mtu": 1500,
    "sub-interface-mtu-overhead": 0,
    "l2-transport": false,
    "bandwidth": 0
  }
]
}
}
}
emsGetOper: ReqId 1, byteRecv: 2325

```

In summary, router LER1, which had minimal configuration, is now programmatically configured using data models with an ethernet interface and is assigned a loopback address. Both these interfaces are operational and ready for network provisioning operations.