

gNMI - Management Interface

- About gNMI, on page 1
- Guidelines and Limitations for gNMI, on page 3
- Configuring gNMI, on page 5
- gNMI RPCs, on page 7
- Union-Replace Operations for gNMI Set Requests, on page 8
- About Get, on page 13
- About Set, on page 14
- About Subscribe, on page 19
- About Subscribing Custom Syslog Stream, on page 23
- Troubleshooting gNMI, on page 26
- Configuration Examples for gRPC Commands, on page 30
- Gathering Debug Logs, on page 32
- Accounting Log for gNMI, on page 32

About gNMI

gNMI (gRPC Network Management Interface) is configured on gRPC. Configuring gNMI helps to edit, read the configuration, and operational state of a network device. It also enables the network devices to generate telemetry streams to a designated data collection system.

Note You can configure gNMI as a child service feature of gRPC feature only.

Cisco NX-OS supports all gNMI RPC as mentioned in the following table:

Table 1: Supported gNMI RPCs

gNMI RPC	Supported
Capabilities	Yes
Get	Yes
Set	Yes

gNMI RPC	Supported
Subscribe	Yes

To subscribe RPC, Cisco NX-OS supports the below submodes:

Table 2: Subscribe Options

Type	Sub Type	Supported	Description
Once		Yes	Switch sends current values only once for all specified paths.
Poll		Yes	Whenever the switch receives a Poll message, the switch sends the current values for all specified paths.
Stream	Sample	Yes	Once per stream sample interval, the switch sends the current values for all specified paths. The supported sample interval range is from 1 through 604,800 second. The default sample interval is 10 seconds.
	On_Change	Yes	The switch sends current values as its initial state, but updates values only when there are changes such as create, modify, or delete for the specified paths.

gNMI subscription is commonly referred to as dial-in telemetry. As gNMI requires an external customer to initiate the request in the network device.

Cisco NX-OS supports a separate telemetry feature in which the networking device pushes the telemetry data out of external receivers, it is referred to as dial-out telemetry. For more information, see the Telemetry section.

Beginning with Cisco NX-OS Release 10.4(3)F, OpenConfig path openconfig-system:/system/processes supports On-change gnmi subscriptions. This path supports both gnmi-proto and gnmi-json encodings. The subscription will remain active until the client unsubscribes the same path.

It is recommended to make the most strict possible subscription for better efficiency. For example, to monitor cpu-usage-user, you can subscribe the data directly rather than using On-target gnmi subscription for general path like /system/processes, to avoid event data.

Guidelines and Limitations for gNMI

The following are the guidelines and limitations for gNMI:

- When you subscribe an OpenConfig routing policy with a pre-existing CLI configuration as shown below, it returns an empty value due to current implementation of the OpenConfig model.

  ```
  ip prefix-list bgp_v4_drop seq 5 deny 125.2.0.0/16 le 32
   ipv6 prefix-list bgp_v6_drop seq 5 deny cafe:125:2::/48 le 128
  ```

  use below path:

  ```
  openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v4_drop]/config
  openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v6_drop]/config
  ```

- For gNMI subscriptions, use of origin, use_models or both these commands are optional.
- Starting with Cisco NX-OS release 9.3(x), for information about supported platforms, see Nexus Switch Platform Support Matrix.
- The feature supports JSON and gNMI proto encoding.
- The feature does not support protobuf any encoding.
- The feature does not support a path prefix in the subscription request, but the Subscription contains an empty prefix field.

Wildcard Path

- Multilevel wildcard "..." in path is not allowed.
- wildcard '*' in the top of the path is not allowed
- wildcard '*' in the key name is not allowed
- wildcard and value are not compatible in keys

The **show grpc gnmi** command has the following guidelines and limitations:

- This command does not support xml or json output format.
- The gRPC agent retains gNMI call data for maximum of an hour after the call ends.
- If the total number of calls exceeds 2000, the gRPC agent purges ended calls based on the internal cleanup routine.

Table 3: Wildcard Support for gNMI Requests

Type of Request	Wildcard Support
gNMI GET	Yes
gNMI SET	No
gNMI SUBSCRIBE, ONCE	Yes
gNMI SUBSCRIBE, POLL	Yes

Type of Request	Wildcard Support
gNMI SUBSCRIBE, STREAM, SAMPLE	Yes
gNMI SUBSCRIBE, STREAM, TARGET_DEFINED	Yes
gNMI SUBSCRIBE, STREAM, ON_CHANGE	No

Scale Considerations

- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the maximum size, the collected data drop. This is applicable for gNMI ON_CHANGE mode only.

  You can create focused subscriptions that manage smaller, granular data collection sets to avoid collection data drop. It is recommended to create multiple subscriptions for different, lower-level parts of the path instead of one higher-level path.

  Across all subscriptions, there is support of up to 250K aggregate MOs. Subscribing to more MOs can lead to collection data drops

- For all subscriptions, there is support of up to 250K aggregate MOs. If you subscribe to more MOs this impacts collection data drop.

Guidelines and Limitations for gNMI Subscription

- On-change subscriptions work for both gnmi and telemetry, but only one of these may be active at one time. If a subscription is made from one of these agents it will overwrite any existing subscription information.

  The following is an example for On-change gnmi subscription for openconfig-system:/system/processes:

```
Request:
./gnmi-console_enhanced_plus --host 172.22.244.142 --port 50051 -u admin -p insieme
--tls --cafile /tmp/grpc.pem --hostnameoverride ems.cisco.com  --operation=Subscribe
--submode ON_CHANGE  --xpath "openconfig-system:system/processes/process[pid=1]" -e
JSON

Response:

////// initial snapshot

Received response 1 -------------------------
/system/
{
    "processes": {
        "process": [
            {
                "pid": "1",
                "state": {
                    "pid": "1",
                    "name": "init",
                    "start-time": "1706643350384761800",
                    "cpu-usage-user": "14",
                    "cpu-usage-system": "12",
                    "cpu-utilization": 0,
                    "memory-usage": "180224",
                    "memory-utilization": 0
                }
            }
        ]
```

```
    }
}

///// first event update

/system/
{
    "processes": {
        "process": [
            {
                "pid": "1",
                "state": {
                    "start-time": "1706643350384761800",
                    "cpu-usage-user": "14",
                    "cpu-usage-system": "12"
                }
            }
        ]
    }
}
```

- Beginning with Cisco NX-OS 10.6(2)F, when you pass model instead of origin, Get handles strings that start with the substring "openconfig", for example "openconfig-", "openconfig-system", "openconfig-sys", and so on.

  These strings are not supported on the original device since no model exists, such as "device-xyz".

Configuring gNMI

Configure the gNMI feature through the gRPC gNMI commands.

Configuring gNMI Options

Before you begin

gNMI is a child service feature of gRPC feature only.

For more information, see the gPRC Agent documentation to enable the gRPC agent.

SUMMARY STEPS

1. switch# **configure terminal**
2. (Optional) **grpc gnmi max-cocrrent-call**
3. **grpc gnmi subscription target-defined min-interval**<*interval*>
4. **grpc gnmi subscription query-condition keep-data-timestamp**
5. (Optional) **grpc gnmi keepalive-timeout** <*timeout*>

DETAILED STEPS

Procedure

	Command or Action	Purpose

Step 1	switch# **configure terminal**
	Example:
	`switch# config terminal`
	`switch(config)#`
		Enters the global configuration mode.

Step 2	(Optional) **grpc gnmi max-cocrrent-call** **Example:** `switch(config)# grpc gnmi max-concurrent-call 16`	Configuring this command sets the limit of simultaneous dial-in calls to the gNMI server on the switch. The default limit is 8. You can configure value with limit range of 1–16. The maximum value that you configure is on each VRF. If you set the limit value of 16 and if gNMI is configured on both management and default VRFs, each VRF supports 16 simultaneous gNMI calls. This command has no effect for ongoing or in-porgress gNMI calls. gRPC enforces a limit on new calls and the active calls have no impact and you can complete the active call.
Step 3	**grpc gnmi subscription target-defined min-interval**<*interval*> **Example:** `switch(config)# grpc gnmi subscription target-defined min-interval 60`	You can modify the default target-defined sample interval from 30 seconds to other required value.
Step 4	**grpc gnmi subscription query-condition keep-data-timestamp** **Example:** `switch(config)# grpc gnmi subscription query-condition keep-data-timestamp`	You can enable sample or once or poll subscriptions to get a timestamp from the database when the data is last updated. **Note** • This command is supported only for PROTO and this is not supported for JSON code. • It supports for once, poll and sample and not supported for not on_change subscriptions. • It supports for DME, YANG, and OpenConfig data sources. • For the properties which are not supported, the command defaults back to the collection time instead of the last database change time. • This command generates verbose responses for each timestamp. If you are not able to collect the messages on time, the switch drops the collection of messages.

	Command or Action	Purpose
Step 5	(Optional) **grpc gnmi keepalive-timeout** <*timeout*> **Example:** `switch(config)# grpc gnmi keepalive-timeout 1200`	Configuring this command, you can delete inactive or unauthorized connections. The gRPC agent periodically sends an empty response to the client. If this response fails to deliver to the client, then the connection stops. The default interval value is 600 seconds. You can configure to change the keepalive interval with the interval range of 600-86400 seconds.

gNMI RPCs

This section describes each gNMI RPC. You can view use of the reference client *gnmi_cli* in this section.

Capabilities

The capabilities of RPC returns to the list of capabilities of the gNMI service. The response message to an RPC request includes the gNMI service version, version data models, and data encode supported by the server.

Guidelines and Limitations for Capabilities

The following are the guidelines and limitations for capabilities:

Example Client Output

Below mentioned the example of client output for Capabilities. Considering the feature *openconfig* is enabled.

This examples shows the support to YANG model, gNMI version, and the supported encodes.

```
$ ./gnmi_cli -a 172.1.1.1:50051 -ca_crt ./grpc.pem -insecure -capabilities
supported_models: <
  name: "Cisco-NX-OS-device"
  organization: "Cisco Systems, Inc."
  version: "2019-11-13"
>
supported_models: <
  name: "openconfig-acl"
  organization: "OpenConfig working group"
  version: "1.0.0"
>
supported_models: <
  name: "openconfig-bgp-policy"
  organization: "OpenConfig working group"
  version: "4.0.1"
>
.
.
.
supported_models: <
  name: "openconfig-spanning-tree"
  organization: "OpenConfig working group"
  version: "0.2.0"
>
supported_models: <
  name: "openconfig-system"
  organization: "OpenConfig working group"
```

```
  version: "0.3.0"
>
supported_models: <
  name: "openconfig-telemetry"
  organization: "OpenConfig working group"
  version: "0.5.1"
>
supported_models: <
name: "openconfig-vlan"
  organization: "OpenConfig working group"
  version: "3.0.2"
>
supported_models: <
  name: "DME"
  organization: "Cisco Systems, Inc."
>
supported_models: <
  name: "Cisco-NX-OS-Syslog-oper"
  organization: "Cisco Systems, Inc."
  version: "2019-08-15"
>
supported_encodings: JSON
supported_encodings: PROTO
gNMI_version: "0.5.0"
```

Union-Replace Operations for gNMI Set Requests

A union-replace operation is a gNMI Set request type that merges OpenConfig and CLI (or device-native YANG) configurations into one candidate configuration for an atomic commit.

- You can include both OpenConfig and CLI payloads in one gNMI Set request.
- The switch processes and merges both configurations in a defined order. CLI values take precedence if there is overlap.
- The candidate configuration is committed as a single transaction, so all changes are applied together.

Use Cases and Benefits of Union-Replace Operations

Use union-replace operations to manage device configuration with both OpenConfig and device-native CLI in a unified, transaction-based workflow.

Consider union-replace for the following reasons:

- Combine OpenConfig and CLI configuration changes in one operation to streamline updates.
- Apply large-scale configuration changes atomically to reduce the risk of partial updates.
- Support migration from CLI-based management to OpenConfig by allowing both models in the same transaction.

Attributes of Union-Replace Operations

Union-replace lets you apply both OpenConfig YANG and CLI (or device-native YANG) configurations together in one atomic operation using gNMI Set. This ensures configuration consistency and flexibility when you manage device settings.

Table 4: Union-Replace Operation Attributes

Attributes	Description	Benefit	Notes
Union-Replace Operation	Combines OpenConfig and CLI or device-native configurations into one candidate configuration.	Lets you manage both industry-standard and device-native models at the same time.	Applies changes as a single operation.
Candidate Configuration	Temporary configuration database where merged changes are staged.	Lets you review and roll back changes before you commit them to running configuration.	System merges changes in memory before commit.
Precedence	CLI values overwrite OpenConfig values if they conflict.	Prevents unintentional overrides of device-native configurations.	Precedence is set in the union-replace process.
Atomic Transaction	Applies all changes as a single unit.	Prevents partial updates and keeps configuration consistent.	Supports rollback with confirm-commit.
Confirm-Commit Extension	Lets you confirm or cancel the committed configuration within a set period.	Reduces the risk of unwanted changes.	Optional in union-replace.

How Union-Replace Operations Work

Use union-replace operations to merge and commit both OpenConfig and CLI configurations in one atomic transaction on the device.

- Use this process during migration between configuration models or when you need device-specific features that require CLI configuration with OpenConfig.
- Use union-replace for large-scale updates or coordinated changes across different configuration models.

Summary

Union-replace operations involve these actors:

- The gNMI client prepares and sends a Set request with both OpenConfig and CLI payloads.
- The switch gRPC/MTX infrastructure receives, processes, and merges the configuration payloads, then commits the combined candidate configuration to the running configuration.

Workflow

These stages describe how union-replace operations work.

1. The gNMI client sends a Set request with both OpenConfig and CLI payloads and specifies the union-replace operation.

- The Set request can include multiple OpenConfig model payloads and one CLI or device-native YANG payload.

The switch receives the union-replace Set request.

2. The switch initializes a candidate configuration database and starts processing the request.
   - The candidate configuration serves as a temporary workspace for merging changes before you commit them to the running configuration.

   The switch prepares to merge both OpenConfig and CLI configurations.
3. The switch processes OpenConfig payloads first and applies their configuration items to the candidate configuration.
   - The system loads all OpenConfig-specified settings into the candidate configuration before CLI processing starts.

   Device-native CLI can overwrite overlapping items as needed.
4. The switch processes the CLI payload, merges its configuration into the candidate configuration, and overwrites any overlapping OpenConfig items.
   - CLI values take precedence if configuration conflicts occur.
   - Process feature dependencies in CLI, such as enabling protocols, before applying dependent OpenConfig settings.

   The candidate configuration now includes all intended configuration.
5. The switch commits the candidate configuration atomically to the running configuration.
   - The system applies all changes as a single transaction. If any part fails, the system makes no changes to the running configuration.
   - If confirm-commit is enabled, confirm or roll back the change within the specified time window.

   The device now runs the configuration with all OpenConfig and CLI changes applied together.

Guidelines and Limitations for Union-Replace Operations

Use union-replace operations when you need to commit both OpenConfig and CLI configurations together. Review overlapping configuration items. CLI values overwrite OpenConfig values when they conflict.

- Union-replace commits all changes atomically as a single transaction. The system does not support partial updates.
- You can use confirm-commit with union-replace. Confirm-commit lets you roll back changes if you do not confirm them within the specified period.
- If the OpenConfig configuration depends on a CLI feature, such as enabling a protocol, the union-replace operation may fail unless the CLI feature command is processed first.
- Do not reference configuration items controlled by gNSI in the candidate configuration. The system generates an error or requires special handling as defined in the platform documentation.

- You can use union-replace only when you provide both OpenConfig and CLI payloads in the same gNMI Set request.
- Union-replace can accept either CLI or CLI with OpenConfig YANG payloads. To support OpenConfig YANG payloads, you need to enable OpenConfig feature on the switch prior.

Configure the Union-Replace Operation

Use this task to combine OpenConfig and CLI configuration changes in a single atomic transaction on your device.

- Maintain configuration consistency across OpenConfig and device-native models.
- Coordinate updates and simplify migration between configuration styles.

Use this task when you need to make configuration changes that include both OpenConfig YANG models and device CLI commands. Apply all updates together in one operation.

- This task helps when device-specific features are not available in OpenConfig models.
- Union-replace operations support large-scale configuration changes.

Before you begin

Before you begin, make sure you meet these requirements:

- The device supports gNMI and union-replace operations.
- You have valid OpenConfig JSON and CLI configuration files.
- You have network connectivity to the device over gNMI and have authentication credentials.

Procedure

Step 1 Send a gNMI Set request with both OpenConfig and CLI payloads. Specify the union-replace operation using the Python client.

Example:

```
python py_gnmicli.py -t <ip> -p <port> -user admin -pass <password> -g -o <origin> -d 0 \
--xpath_union_replace network-instances/network-instance[name=foo]@empty.json \
--xpath_union_replace acl/acl-sets/acl-set[name=acl_1][type=ACL_IPV4]@payload/union/acl-set.json \
--xpath_union_replace nxos_cli:/@payload/union/nxosv104.cfg
```

This table explains each part of the `py_gnmicli.py` command for a union-replace operation.

Table 5: Explanation of Union-Replace Command Arguments

Argument	Description
`python py_gnmicli.py`	Runs the Python-based gNMI client script to send configuration requests to the device.
`-t <ip>`	Specifies the IP address of the target Nexus switch.

Argument	Description
`-p` **<port>**	Specifies the gNMI service port on the device.
`-user` **admin**	Specifies the username for authentication.
`-pass` **<password>**	Specifies the password for authentication.
`-g`	Uses the gNMI protocol for the request.
`-o` **<origin>**	Specifies the origin for the configuration. Replace with the required value for your network, such as `openconfig` or `ems.cisco.com`.
`-d` **0**	Sets the debug level. `0` means minimal debug output.
`--xpath_union_replace` **OpenConfig_xpath@file.json**	Specifies an OpenConfig YANG configuration payload and its target file. You can include multiple OpenConfig payloads as needed.
`--xpath_union_replace` **nxos_cli:/@cli_file.cfg**	Specifies the device-native CLI configuration file. The `nxos_cli:/` origin indicates that the payload is CLI-based.

The device merges and applies both OpenConfig and CLI configurations as a single transaction. If you use confirm-commit, the system stages the changes and you must confirm or cancel the changes within the specified window.

Step 2 (Optional) Send a gNMI Set request with both OpenConfig and CLI payloads using the Go-based `gnmic` client.

This example uses the Go-based `gnmic` client. The `--union-replace-path` and `--union-replace-file` options specify OpenConfig configuration payloads, and `--union-replace-cli-file` specifies the CLI payload.

Example:

```
gnmic -a <ip>:<port> -u admin -p <password> --skip-verify set \
--union-replace-path network-instances/network-instance[name=foo] --union-replace-file
nx/payload/union/empty.json \
--union-replace-path acl/acl-sets/acl-set[name=acl_1][type=ACL_IPV4] --union-replace-file
nx/payload/union/acl-set.json \
--union-replace-cli-file nx/payload/union/nxosv104.cfg
```

Step 3 (Optional) Include confirm-commit parameters to enable rollback or confirmation of configuration changes.

Example:

```
python py_gnmicli.py ... --commit_request 100 --commit_id test
python py_gnmicli.py ... --commit_confirm --commit_id test
python py_gnmicli.py ... --commit_cancel --commit_id test
```

This table explains the confirm-commit parameters you can use with `py_gnmicli.py` to stage, confirm, or roll back configuration changes with a union-replace operation.

Table 6: Explanation of Confirm-Commit Command Arguments

Command/Argument	Description
`--commit_request` **<timeout>**	Commits the configuration and starts a timer in seconds for you to confirm. If you don't confirm, the configuration is reverted back after the configured time is over.
`--commit_id` **<id>**	Specifies a unique identifier for the commit. Use this identifier for both confirmation and cancellation.

Command/Argument	Description
`--commit_confirm --commit_id` **<id>**	Confirms the configuration associated with the specified commit ID before the timeout expires. The configuration is retained.
`--commit_cancel --commit_id` **<id>**	Discards the configuration associated with the specified commit ID before the timeout expires. The configuration is rolled back.

The union-replace operation combines and commits both OpenConfig and CLI configuration changes together. This ensures consistency and reduces the risk of partial updates.

About Get

You can use `Get RPC` to retrieve a snapshot of the data tree from the switch. You can request multiple paths in a single request. According to gNMI Path conventions, you can use simple form of XPATH. See Schema path encoding conventions for gNMI. For more information about `GET` operation, see the *Retrieving Snapshots of State Information* section in the gRPC Network Management Interface.

Guidelines and Limitations for Get

The following are guidelines and limitations for Get:

- `GetRequest` encode supports only the JSON format.
- For the `GetRequest`.type, only DataType CONFIG and STATE have correlation and the expression in YANG format. The operation is not supported.
- You cannot have both OpenConfig (OC) YANG and device YANG paths in a single request. A single request has only one path.
- `GetRequest` for root path ("/": everything from **all** models) is not permitted.
- gNMI Get returns all default value. See Report-all mode RFC 6243 [4].
- `Get` does not support the `Cisco-NX-OS-syslog-oper` model.
- To retrieve `openconfig-procmon` data, you can send a query to the path `/system/processes` or `/system`.

Note Before Cisco NX-OS Release 10.3(x), you cannot retrieve data from the `/system` path.

- The following optionals are not supported:
  - Path prefix
  - Path alias
  - Wildcards in the path
- You can request 10 paths in a single `GetReqeuest`.

- If the return value of size in `GetResponse` is more than 12 MB, the system displays the error status `grpc::RESOURCE_EXHAUSTED`.
- The maximum receive buffer size of gRPC is 8 MB.
- If you perform an `Get` operation for a switch which has more configuration, the gRPC process consumes all available memory. If the memory exhausts, the below syslog is generated:

```
MTX-API: The memory usage is
        reaching the max memory resource limit (3072) MB
```

  If the memory exhausts consecutively, the below syslog is generated:

```
The process has become unstable and
        the feature should be restarted.
```

  Cisco recommends you to restart the gRPC feature for normal functioning of gNMI transactions.
- The maximum number of total concurrent sessions for `Get` is 75% of maximum configured concurrent calls. For an example, if the MTX concurrent calls are configured to 16, the maximum number of total concurrent sessions for `Get` is 12.
- The total number of concurrent sessions for `Get` and `Set` is configured max gNMI concurrent-1.

  For an example, if gNMI concurrent calls are configured to 16, the maximum number of total concurrent sessions for `Get` and `Set` is 15.

About Set

You can use Set RPC to change the configuration of the switch. You can delete, replace, and update the switch. All these operations in a single Set request that is considered as a transaction which is a successful operation or the switch is set to original state.

The Set operations are performed in the order that is specified in the Set Request configuration. If a path is requested multiple times, the changes are performed even if the paths overwrite each other path. The final state of the data is achieved with the final operation in the transaction. You can edit the paths that are specified in the Set Request such as delete, replace, update fields are configuration data paths.

For more information on Set operations, see *Modifying State* section of gNMI specification.

<Set> Replace with CLI origin

The Set RPC generally supports the YANG-based origin : **openconfig** and **device** origin. Specifically for the **replace** operation, it supports the extra CLI based origin : **nxos_cli** and **cli** origin.

These two keywords provide identical functionality. The standard and preferred keyword is **nxos_cli**, while Nexus also supports the **cli** keyword in case the gNMI clients are not able to send the **nxos_cli** keyword.

The gNMI **Set Replace with CLI origin** can be viewed as the drop-in replacement of the **cli config replace**.

Instead of executing **config replace** in the Nexus CLI shell, the user can trigger the config replace by sending the gNMI request with **nxos_cli** origin keyword, using the same payload as the CLI replace. Then gNMI would trigger the CLI config replace accordingly.

In the below example, the user would prepare the target config in the ‘'show run” format, then supply the file content to the gNMI client to send as replace PRC with the cli origin. With the opensource gnmic tool, this is done via the “--replace-cli-file" option.

```
> cat switch_run.cfg
!Running configuration last done at: Thu Nov  6 22:49:47 2025
!Time: Fri Nov  7 16:22:45 2025

version 10.6(3) Bios:version
hostname switch
...
```

<Set> Union-Replace with CLI and Openconfig

Beginning with Cisco NX-OS Release 10.6(3)F, gNMISet RPC supports message type **union-replace** that allows a single gNMI Set RPC to carry both the CLI text configuration as well as the Opeconfig YANG JSON payloads. The intent is to replace the switch configuration with the **union** of the CLI and Openconfig. If the CLI and Openconfig overlaps, CLI configuration takes the higher precedence.

To process the **union-replace**, NX-OS executes the following steps.

- Reset the gNMI candidate database to the default status, which includes the mandatory baseline NX-OS configuration.
- If **feature** configurations exist in the CLI text, enable the feature configuration in the candidate database.
- Apply the OpenconfigYANG payloads to the candidate database based on the receiving order.
- Apply the CLI text configuration to the candidate database.
- Commit the candidate database to the running configuration.

In the following example, the user prepares a base CLI configuration and a collection of Openconfig YANG payloads. Then the user supplies the CLI and Openconfig files to the gNMI client to send as **union-replace** request. With the open source gnmic tool. This is done through the **--union-replace-path**, **--union-replace-file** and the **--union-replace-cli-file** options.

```
> > gnmiclient
> --union-replace-path network-instances/network-instance[name=example] --unionreplace-
file vrf.json
> --union-replace-path acl/acl-sets/acl-set[name=acl_1][type=ACL_IPV4] --union-replacefile
acl.json
> --union-replace-cli-file switch_run.cfg

> cat switch_run.cfg
!Running configuration last done at: Thu Nov 6 22:49:47 2025
!Time: Fri Nov 7 16:22:45 2025
version 10.6(3) Bios:version
hostname switch
...
```

Guidelines and Limitations for Set

The following are guidelines and limitations for Set:

- `SetRequest` encode supports JSON format and ASCII only for the replace operation.
- You cannot have both OpenConfig (OC) YANG and device YANG paths in a single request. A single request has only one path.
- The following optionals are not supported:
  - Path prefix

- Path alias
- Wildcards in the path

- You can request 20 paths in a single `SetReqeuest`.

  For Set::Replace with CLI, only 1 path is allowed.

- The maximum receive buffer size of gRPC is 8 MB.
- The maximum number of total concurrent sessions for `Get` and `Set` is configured maximum gNMI concurrent calls. For an example, if the gNMI concurrent calls are configured to 16, the maximum number of total concurrent sessions for `Get` and `Set` is 15.
- If you perform a Set::Delete RPC operation for a switch, if a configuration is bulky, an MTX warning message is generated as shown below:

```
Configuration size for this
        namespace exceeds operational limit. Feature may become unstable and require
        restart.
```

Guidelines and limitations for union replace

Follow these guidelines and note these limitations when you use union-replace operations to combine OpenConfig and CLI configurations in a single gNMI Set request.

- Enable Openconfig feature on switch beforehand.
- Use union-replace operations when you need to commit both OpenConfig and CLI configurations together. Review overlapping configuration items. CLI values overwrite OpenConfig values when they conflict.
- Union-replace commits all changes atomically as a single transaction. The system does not support partial updates.
- You can use confirm-commit with union-replace. Confirm-commit lets you roll back changes if you do not confirm them within the specified period.
- If the OpenConfig configuration depends on a CLI feature, such as enabling a protocol, the unionreplace operation may fail unless the CLI feature command is processed first.
- Do not reference configuration items controlled by gNSI in the candidate configuration. The system generates an error or requires special handling as defined in the platform documentation.
- You can use union-replace only when you provide both OpenConfig and CLI payloads in the same gNMI Set request.
- Union-replace can accept either CLI or CLI with OpenConfig YANG payloads. To support OpenConfig YANG payloads, you need to enable OpenConfig feature on the switch prior.
- Openconfig and GRPC features must be enabled on switch before running union-replace with cli and openconfig.

Extensions for Set

The following extensions are supported for the Set RPC.

Extension Confirm-Commit

This extension provides an automatic way to rollback an applied configuration in case the new configuration results in a loss of connectivity to the device. The details of the extension, including all protobuf message definitions, can be found here: https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-commit-confirmed.md

The gnmi confirm-commit feature is a simplified subset of the existing netconf confirm-commit. For gnmi confirm-commit transaction, the extension applies to a single **Set** operation. This **Set** operation must be confirmed within a specified amount of time, otherwise the operations in the Set request will be rolled back to the previous state. It is also possible to explicitly cancel a **Set** operation rather than confirming it.

A typical use case for this extension is straightforward. A gnmi client applies a configuration change using a **Set** operation as usual, but also includes the **Commit** extension message. This **Commit** message will contain a string identifier and a **CommitRequest** message which sets the rollback duration. This **CommitRequest** message, identified by the given id string, must be confirmed before this rollback duration expires. The **CommitRequest** is confirmed by sending another **Set** request, with no data included, that carries the **CommitConfirm** message in the extension. If a configuration change has disrupted network connectivity, this confirmation message will not be received by the device, and the configuration will be rolled back to the previous configuration. If the client desires, it may also send a **CommitCancel** message to immediately cancel and rollback the configuration. There is also a message to extend the rollback duration timer. If a **CommitRequest Set** operation is confirmed by successfully sending the CommitConfirm message, the applied configuration is retained. Refer to the reference above for a complete explanation of the messages.

Guidelines and Limitations for Confirm-Commit

There are a few things the user should keep in mind when using the gnmi confirmed-commit feature to push configurations.

- The intent and effect of the configuration change must be understood. For example in some cases a user may wish to shutdown the interface or disable grpc itself. In such cases it is not possible to receive the CommitConfirm message, thus the confirm-commit extension does not make sense in such cases.
- In some cases a configuration change may temporarily disrupt network connectivity and require time to recover. This transient disruption should be anticipated and an appropriate timeout period chosen such that the configuration has reached steady state when a confirmation may be made. Choosing an inappropriate timeout for the configuration change may result in unnecessary configuration rollback. Choosing this timeout is the responsibility of the user.
- Administrators and operators should be aware of the possibility of configuration rollback, as this may interfere with log collection and other debug/monitoring tools.
- Once a Set operation is applied using the CommitRequest message and the ‘id’ established, no other Set operation will be accepted. The active Set (matching the ‘id’) must be confirmed, canceled, or allowed to time out.
- Set operations carrying the CommitConfirm, CommitCancel, or CommitSetRollbackDuration messages are not allowed to also carry data for update/replace/delete operations. Only the CommitRequest message may be carried with a Set that contains paths to operate on (the initial Set of the transaction).
- A single CommitConfirm transaction may be active at any time considering both the YANG-based and CLI-based commit.
- If a supervisor switchover occurs during an active gNMI Set operation that uses the confirm-commit extension, the newly active supervisor cannot roll back the configuration. The operation cannot be

canceled, and expiration of the confirm-commit timeout does not trigger a rollback. The new configuration remains in effect. This limitation applies to both Device YANG and OpenConfig paths because they use the shared data store.

- The timeout value has only seconds level granularity given in Duration.seconds. If the given Duration has non-zero value for nanoseconds it is rejected.
- For the Set::Replace with CLI origin, it shares the same limitations as the CLI config replace.
  - The cli-based confirmed commit only support timer between 30 and 3600 seconds.
  - Does not support to extend the timeout.
  - The CLI config replace does not support the commit id. Therefore, the commit id is processed localin in Nexus gRPC, and not visible in the show output of the CLI config-replace. If the switch resetsdisruptively The Nexus gRPCmay become out-of-sync with the CLI config-replace. In this case, the user is recommanded to clean up the CLI config-replace manually, then proceed the normal operation.

Show Commands

The following commands are enhanced to display information about gnmi confirmed commit transactions. The examples below are after a Set request to modify the Lo1 interface description and setting commit-id to "commitId-123" with timeout of 600 seconds.

- `show grpc gnmi transactions`

  This command shows gnmi message level details of the confirm-commit extension, such as the commit ‘id’, the timeout duration, and the message type (CommitRequest, CommitConfirm, CommitCancel and so on).

```
RPC          DataType   Session         Time In              Duration(ms) Status
------------ ---------- --------------- -------------------- ------------ ------
Set          -          3               06/16 20:02:51       5            0
Commit Type: CommitConfirm   Persist: commitId-123         Timeout :600


Set          -          2               06/16 20:01:41       75           0
subtype: dtx:  st: path:
Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo1]/descr
Commit Type: CommitRequest   Persist: commitId-123         Timeout :600


Set          -          1               06/16 20:00:55       13           0
```

- `show grpc internal mtx commit history`

  This command shows internal state information such as the session-id and timestamp.

```
MTX Commit History

Session    Op      Persist          PersistId        Timeout      Ret    Timestamp

3          COMMIT                   commitId-123                  1      2025-06-16
20:02:51

2          COMMIT  commitId-123                      600          1      2025-06-16
20:01:41
```

- `show grpc internal mtx db`

  This command shows internal state information at the DB level.

```
MTX DME DB state variables
Running Config locked : false
Session that locked Running : 18446744073709551615
Session doing confirmed commit : 18446744073709551615
Is a confirmed commit active : false
Thread Local Data instances : 1
Confirmed Commit start time : 2025-06-16 20:01:41
Confirmed Commit timeout : 600
Confirmed Commit time remaining : 0
Persistent Session id : 3
Commit Candidate to Running :
    Transaction Token : commitId-123
    Commit Type : COMMIT_CONFIRM_END
```

About Subscribe

You can use `Subscribe RPC` to register a telemetry stream for specific paths. When there is change in the registered paths, the switch pushes the notification to for the change in configuration or the path state.

You can use an optional flag available for `Subscribe RPC`. Beginning with Cisco NX-OS Release 9.3(1), the `UPDATES_ONLY` optional flag is supported and applicable only for `ON_CHANGE` subscriptions. If this optional flag is configured, the switch suppresses the current state and a notification is sent with the first response.

You can use these flags to modify the response to options listed in the following table.

Table 7: Support Metrics for `SUBSCRIBE` Flags

Subscription Type	hearbeat_interval	suppress_redundant
`ON_CHANGE`	Origin: Device YANG, OpenConfig YANG, DME	N/A
`SAMPLE`	Origin: Device YANG, OpenConfig YANG, DME	Origin: Device YANG, OpenConfig YANG

Beginning with Cisco NX-OS Release 10.2(3)F, the following optional flags are supported:

- `hearbeat_interval`
- `suppress_redundant`

You can use a `hearbeat_interval` flag to modify the behavior of `suppress_redundant` in a `sample` subscription. In this case, the target generates one telemetry update per `hearbeat_interval`, despite of

`suppress_redundant` flag status is set to true. The value is specified as an unsigned 64-bit integer in nano seconds.

You can use the `suppress_redundant` flag for a `sample` subscription. In this case, the set status is true. The target generates a telemetry update message when the value of the path reported has changed the last update is generated. Updates are generated for the individual leaf nodes for which the subscription has changed.

For an example, subscriptions name 'A' and 'B' which has leaf nodes 'C' and 'D' branch from 'B' node. If the value of 'C' is changed, and not the value of 'D'. An update is generated only for 'C' and not for 'D'.

Guidelines and Limitations

The following are guidelines and limitations for Set subscription:

- The following flags are not supported:
  - Aliases
  - allow_aggregation
  - extensions
  - prefix
  - QoS
- Make sure that all paths within the same subscription request must have the same `sample` interval. If the same path requires different sample intervals, you must create multiple subscriptions.
- In low memory condition, an OC subscription requests may fail immediately rather than wait for memory to become available.
- OC event notifications may be dropped if excessive events are generated by the system.
- OC event notifications are disabled during subscription snapshot, and will be flushed when snapshot completes.
- OC snapshot completion time limit is 30 minutes. If not complete in that time, the subscription is canceled.

gNMI Subscribe Options

The following table lists the subscribe option modes:

Table 8:

Type	Subscription Type	Description
`ONCE`		Subscribe to receive data when and close the session.
`POLL`		Subscribe to retain an active session. When you raise a poll request, ensure that you enter data for each request.

Type	Subscription Type	Description
STREAM	SAMPLE	Subscribe to receive data at a specific cadence. The payload value is in nano seconds. 1 second =1000000000.
	ON_CHANGE	Subscribe to receive a snapshot and receive data only when there is change in tree.
TARGET_DEFINED		Subscribe to find the best subscription which can be created.

To Set Modes

Each mode requires 2 settings, such as inside subscription and outside subscription. The following mentioned the combination of subscriptions:

- ONCE: SAMPLE, ONCE
- POLL: SAMPLE, POLL
- STREAM: SAMPLE, STREAM
- ON_CHANGE: ON_CHANGE, STREAM
- TARGET_DEFINED: TARGET_DEFINED, STREAM

Origin

- DME: Subscribing to DME model
- DEVICE: Subscribing to YANG model
- OPENCONFIG: Subscribing to OpenConfig model

Name

- DME: Subscribing to DME model
- Cisco-NX-OS-device: Subscribing to YANG model

Encoding

- JSON: Stream is sent in JSON format.
- PROTO: Stream is sent in original proto format.

Configuring Example of Payload

The following section lists client examples for a payload:

Subscribe to DME Stream/Sample

```
{
    "SubscribeRequest":
    [
```

```
        {
            "subscribe":
            {
                "subscription":
                [
                    {
                        "path":
                        {
                            "origin": "DME",
                            "elem":
                            [
                                {
                                    "name": "sys"
                                },
                                {
                                    "name": "bgp"
                                 }
                             ]
                        },
                        "mode": "SAMPLE"
                    }
                ],
                "mode": "ONCE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "name": "DME",
                        "organization": "Cisco Systems, Inc.",
                        "version": "1.0.0"
                    }
                ],
                "encoding": "JSON"
            }
        }
    ]
}
```

Subscribe to OpenConfig YANG/Sample

```
{
    "SubscribeRequest":
    [
        {
            "subscribe":
            {
                "subscription":
                [
                    {
                        "path":
                        {
                            "origin": "openconfig",
                            "elem":
                            [
                                {
                                    "name": "interfaces"
                                }
                            ]
                        },
                        "mode": "SAMPLE",
                        "Sample_interval": 10000000000
                    }
                ],
```

```
                    "mode": "ONCE",
                    "allow_aggregation" : false,
                    "use_models":
                    [
                        {
                            "name": "openconfig-interfaces",
                            "organization": "OpenConfig working group",
                            "version": "0.8.1"
                        }
                    ],
                    "encoding": "JSON"
                }
            }
        ]
}
```

About Subscribing Custom Syslog Stream

Cisco NX-OS supports the original and OpenConfig model. For gNMI or, subscribe `ON_CHANGE`, a custom YANG model is designed to stream the syslog events. You can configure this feature for the Cisco Nexus 9000 Series switches with minimum memory of 8 GB and above.

Guidelines and Limitations

The following are guidelines and limitations for Syslog Stream:

- An invalid syslog is not supported, such as a syslog with a filter or query condition is not supported.
- Syslog stream supports only below paths:
  - Cisco-NX-OS-Syslog-oper: syslog
  - Cisco-NX-OS-Syslog-oper: syslog messages
- Only stream sample and POLL modes are supported.
- Encoding formats that are supported are JSON and PROTO.

Original YANG Syslog Model

The following example shows the configuration of the YANG Syslog Model:



Note By default, the time-zone field is empty. The time zone field is set when you configure `clock format show-timezone syslog`.

```
PYANG Tree for Syslog Native Yang Model:
>>> pyang -f tree Cisco-NX-OS-infra-syslog-oper.yang module: Cisco-NX-OS-syslog-oper
+--ro syslog
+--ro messages
+--ro message* [message-id]
+--ro message-id int32
+--ro node-name? string
+--ro time-stamp? uint64
+--ro time-of-day? string
```

```
+--ro time-zone? string
+--ro category? string
+--ro group? string
+--ro message-name? string
+--ro severity? System-message-severity
+--ro text? string
```

Subscribe Request

The following is an example for the subscribe request:

```
{
    "SubscribeRequest":
    [
        {
            "subscribe":
            {
                "subscription":
                [
                    {
                        "path":
                        {
                            "origin": "syslog-oper",
                            "elem":
                              [
                                {
                                    "name": "syslog"
                                },
                                {
                                    "name":"messages"
                                }
                              ]
                        },
                        "mode": "ON_CHANGE"
                    }
                ],
                "mode": "ON_CHANGE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "name": "Cisco-NX-OS-Syslog-oper",
                        "organization": "Cisco Systems, Inc.",
                        "version": "0.0.0"
                    }
                ],
                "encoding":"JSON"
            }
        }
    ]
}
```

Example Response

The following is an output response example for PROTO encode:

```
[Subscribe]------------------------------
Sat Aug 24 14:38:06 2019
### Generating request : 1 -----------
### Comment : STREAM request
### Delay : 2 sec(s) ...
### Delay : 2 sec(s) DONE

subscribe {
subscription {
path {
```

```
origin: "syslog-oper"
elem {
name: "syslog"
}
elem {
name: "messages"
}
}
mode: ON_CHANGE
}

use_models {
name: "Cisco-NX-OS-Syslog-oper"
organization: "Cisco Systems, Inc."
version: "0.0.0"
}
encoding: PROTO
}

Thu Nov 21 14:26:41 2019
Received response 3 --------------------------
update {
timestamp: 1574375201665688000
prefix {
origin: "Syslog-oper"
elem {
name: "syslog"
}
elem {
name: "messages"
}
}
…
update {
path {
elem {
name: "time-stamp"
}
}
val {
uint_val: 1574375200000
}
}
update {
path {
elem {
name: "severity"
}
}
val {
uint_val: 5
}
}
}

/Received -------------------------------------
```

The following is an example of output response for JSON encode:

```
[Subscribe]-------------------------------
### Reading from file ' testing_bl/stream_on_change/OC_SYSLOG.json '

Tue Nov 26 11:47:00 2019
### Generating request : 1 -----------
### Comment : STREAM request
```

```
### Delay : 2 sec(s) ...
### Delay : 2 sec(s) DONE
subscribe {
subscription {
path {
origin: "syslog-oper"
elem {
name: "syslog"
}
elem {
name: "messages"
}
}
mode: ON_CHANGE
}
use_models {
name: "Cisco-NX-OS-Syslog-oper"
organization: "Cisco Systems, Inc."
version: "0.0.0"
}
}

Tue Nov 26 11:47:15 2019
Received response 5 --------------------------
update {
timestamp: 1574797636002053000
prefix {
}
update {
path {
origin: "Syslog-oper"
elem {
name: "syslog"
}
}
val {
json_val: "[ { \"messages\" : [[
{\"message-id\":657},{\"node-name\":\"task-n9k-1\",\"time-stamp\":\"1574797635000\",\"time-of-day\":\"Nov
 26 2019
11:47:15\",\"severity\":3,\"message-name\":\"HDR_L2LEN_ERR\",\"category\":\"ARP\",\"group\":\"ARP\",\"text\":\"arp
 [30318] Received packet with incorrect layer 2 address length (8 bytes), Normal pkt with
S/D MAC: 003a.7d21.d55e ffff.ffff.ffff eff_ifc mgmt0(9), log_ifc mgmt0(9), phy_ifc
mgmt0(9)\",\"time-zone\":\"\"} ]] } ]"
}
}
}
```

Troubleshooting gNMI

You can run show commands to view gNMI status. To verify specific gNMI configurations, enter the following commands that are listed in the below table:

Table 9: Show Commands

Command	Description
`show grpc gnmi service statistics`	Displays the summary of the agent running status, respectively for the management VRF, or the default configured VRF. It also displays: • Basic overall counters • Certificate expiration time
`show grpc gnmi rpc summary`	Displays the following: • Number of capability RPCs received. • Capability of RPC errors. • Number of Get RPCs received. • Get RPC errors. • Number of Set RPCs received. • Set RPC errors.

Command	Description
`show grpc gnmi transactions`	

Command	Description
	The show grpc gnmi transactions command is the densest and contains considerable information. It is history buffer of the most recent 50 gNMI transactions that are received by the switch. As new RPCs come in, the oldest history entry is removed from the end. The following explains what is displayed: This command displays the detailed information. It shows the history of the latest 50 gNMI transactions that are received by the switch. An entry of new RPCs, the history of the oldest entry of RPCs is removed. The following shows the information which is displayed: • RPC - This shows the type of RPC that was received (Get, Set, Capabilities) • DataType - For a Get only. It has the values ALL, CONFIG, and STATE. • Session - Displays the unique session-id that is assigned to this transaction. It can be used to correlate data that is found in other log files. • Time In - Displays the timestamp when the RPC was received by the gNMI handler. • Duration - Time delta in milliseconds from receiving the request to giving a response. • Status – Displays the status code of the operation returned to the client `(0 = Success, !0 == error)` The following shows the data per path within a single gNMI transaction. For a single `Get` or `Set`: • **Subtype** – For a `Set RPC`, it displays the specific operation that is requested per path (Delete, Update, Replace). For `Get`, there is no subtype. • **Dtx** – Displays that this path is processed in DTX fast path or not. A dash (-) indicates no, an asterisk (*) indicates yes. • **St** – Displays the status for this path. The different status that is displayed as shown below: • **OK**: The path is valid and processed by infra successfully. • **ERR**: The path is either invalid or generated error by infra. • **--**: The path is not processed yet, or not a

Command	Description
	valid and not sent to infra yet.

Configuration Examples for gRPC Commands

gRPC gNMI Service Statistics

The following shows the sample output for the command `show grpc gnmi service statistics`:

```
=============
gRPC Endpoint
=============

Vrf : management
Server address : [::]:50051

Cert notBefore : Mar 13 19:05:24 2020 GMT
Cert notAfter : Nov 20 19:05:24 2033 GMT

Max concurrent calls : 8
Listen calls : 1
Active calls : 0

Number of created calls : 1
Number of bad calls : 0

Subscription stream/once/poll : 0/0/0

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 74
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0
```

gRPC gNMI rPC Summary

The following shows the sample output for the command `show grpc gnmi service statistics`:

```
=============
gRPC Endpoint
=============

Vrf            : management
Server address : [::]:50051

Cert notBefore : Mar 31 20:55:02 2020 GMT
Cert notAfter  : Apr  1 20:55:02 2020 GMT

Capability rpcs    : 1
Capability errors  : 0
Get rpcs           : 53
Get errors         : 19
Set rpcs           : 23
Set errors         : 8
Resource Exhausted : 0
Option Unsupported : 6
Invalid Argument   : 18
Operation Aborted  : 1
```

```
Internal Error     : 2
Unknown Error      : 0

RPC Type        State      Last Activity  Cnt Req    Cnt Resp   Client
--------------- ---------- -------------- ---------- ---------- -----------------
-----------------------
Subscribe       Listen     04/01 07:39:21          0          0
```

gRPC gNMI Transactions

The following shows the sample output for the command `show grpc gnmi transactions`:

```
=============
gRPC Endpoint
=============

Vrf            : management
Server address : [::]:50051

Cert notBefore : Mar 31 20:55:02 2020 GMT
Cert notAfter  : Apr  1 20:55:02 2020 GMT

RPC          DataType   Session         Time In              Duration(ms) Status
------------ ---------- --------------- -------------------- ------------ ------
Set          -          2361443608      04/01 07:43:49       173          0
subtype: dtx:  st: path:
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo789]

Set          -          3445444384      04/01 07:43:33       3259         0
subtype: dtx:  st: path:
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo789]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo790]
…
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo807]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo808]

Set          -          2297474560      04/01 07:43:26       186          0
subtype: dtx:  st: path:
Update   -     OK  /System/ipv4-items/inst-items/dom-items/Dom-list[name=foo]/rt-
items/Route-list[prefix=0.0.0.0/0]/nh-items/Nexthop-list[nhAddr=192.168.1.1/32][n
hVrf=foo][nhIf=unspecified]/tag


Set          -          0               04/01 07:43:11       0            3
subtype: dtx:  st: path:
Update   -     --  /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Update   -     ERR /system/processes

Set          -          2464255200      04/01 07:43:05       708          0
subtype: dtx:  st: path:
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo2]
Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr
Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo5]/descr
Update   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr
Update   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo5]/descr


Set          -          3491213208      04/01 07:42:58       14           0
subtype: dtx:  st: path:
Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr

...

Get          ALL        2293232352      04/01 07:42:35       258          0
```

```
subtype: dtx:  st: path:
-        -     OK  /system

Get          ALL        0               04/01 07:42:33       0            12
subtype: dtx:  st: path:
-        -     --  /intf-items
```

Gathering Debug Logs

gNOI is a child service of the gRPC agent. For more information, see Diagnosis and Serviceability.

Accounting Log for gNMI

In gNMI, `Set RPC` changes the configuration on the switch. For the `SET` requests such as `UPDATE`, `REPLACE`, or `DELETE` configuration, gNMI generates the correspoding accouting logs. These logs include both original request and the changes made on the switch.

You can use `show accouting log` command to view the accouting logs.

The following example shows `SetRequest`, encoding = JSON to localhost with the following gNMI paths:

```
---
<<<<<<< set_delete >>>>>>>
[]
<<<<<<< set_replace >>>>>>>
[] []
<<<<<<< set_update >>>>>>>
[elem { name: "System"
} elem {
name: "tm-items"
} elem {
name: "certificate-items"
}
] [json_val: "{\"hostname\": \"test\", \"trustpoint\": \"foo\"}"
]
The SetRequest response is below ------------------------response { path {
elem {
name: "System"
} elem {
name: "tm-items"
} elem {
name: "certificate-items"
}
} op: UPDATE
} timestamp: 1656512303065384369
---
```

The below table shows the options can be configured on the switches. The accouting logs include the following items.

Item	Description
Context	Session ID and user
Operation	Commit or Abort
Database	Running or Candidate

Item	Description
ConfigMO	MO tree's text representation. Maximum up to 3000 characters.
Status	Success or Failed

The following shows the sample of accouting logs.

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(COMMIT),database=[candidate],
configMo=[<topSystem childAction="" dn="sys" status="created,modified"><telemetryEntity
childAction="" rn="tm" status="created,modified"><telemetryCertificate childAction=""
hostname="test" rn="certificate" status="created,modified"
trustpoint="foo"/></telemetryEntity></topSystem>] (SUCCESS)
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(COMMIT:CANDIDATE-TO-RUNNING),
database=[running] (SUCCESS)
```

The below table shows the original received request on the switch.

Item	Description
Context	Session ID and user
Operation	gNMI:SET:UPDATE, gNMI:SET:REPLACE, gNMI:SET:DELETE, COMMIT:CANDIDATE-TO-RUNNING
Source IP	gNMI Client IP
Path	gNMI path in the text format
Payload	Received JSON Request. Maximum up to 3000 characters.
ConfigCli	Received ASCII Request. Up to 3000 characters
Status	Success or Failed

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(GNMI:SET:UPDATE),sourceIp=[192.168.1.2],
path=[/System/tm-items/certificate-items],payload=[{"hostname":"test","trustpoint":"foo"}]

(SUCCESS)
```

In case of failed request and based on failed scenario, you cannot view both the logs.

Invalid Requests

If you raise a request which is invalid, this request will be rejected without any confiugartion changes and only the initial request will be logged. The following shows the example of invalid request logs.

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(GNMI:SET:UPDATE),
sourceIp=[192.168.1.2],path=[/System/tm-items/certificateitems],
payload=[{"hostname":"test","trustpoint":"foo"}] (FAILED)
```

Failed Requests

If you raise a request and if it fails due to various configuration restrictions, in such case both the original and failed configuration request is logged. The following example shows the logs.

```
Wed Jun 29 20:52:15
2022:type=update:id=1429663200:user=admin:cmd=(COMMIT),database=[candidate],
configMo=[<topSystem childAction="" dn="sys"
status="created,modified"><telemetryEntity childAction="" rn="tm"
status="created,modified"><telemetryCertificate childAction="" filename="foo"
hostname="test" rn="certificate" status="created,modified,replaced"
trustpoint="foo"/></telemetryEntity></topSystem>] (FAILED)

Wed Jun 29 20:52:15
2022:type=update:id=1429663200:user=admin:cmd=(GNMI:SET:REPLACE),
sourceIp=[192.168.1.2],path=[/System/tm-items/certificate-items],
payload=[{"hostname":"test","trustpoint":"foo","filename":"foo"}] (FAILED)
```

If you raise a request and if it fails to commit, in such case the original request is logged with the failed request. The following example shows the logs.

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd
(COMMIT),database=[candidate],configMo=[<topSystem childAction="" dn="sys"
status="created,modified"><telemetryEntity childAction="" rn="tm"
status="created,modified"><telemetryCertificate childAction="" hostname="test"
rn="certificate" status="created,modified" trustpoint="foo"/></telemetryEntity></topSystem>]
 (SUCCESS)

Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(GNMI:SET:UPDATE),
sourceIp=[192.168.1.2],path=[/System/tm-items/certificate-items],
payload=[{"hostname":"test","trustpoint":"foo"}] (SUCCESS)

Wed Jun 29 20:34:06
2022:type=update:id=1429665744:user=admin:cmd=(COMMIT:CANDIDATE-TO-RUNNING),
database=[running] (FAILED)
```