Configure Model-based Telemetry

Streaming model-based telemetry data to the intended receiver involves:

Configure Dial-out Mode

In a dial-out mode, the router initiates a session to the destinations based on the subscription.

All 64-bit IOS XR platforms (except for NCS 6000 series routers) support gRPC and TCP protocols. All 32-bit IOS XR platforms support only TCP.

MDT supports sourcing from virtual routing and forwarding (VRF) interface for TCP and gRPC protocols. Source interface and VRF can be configured in dial-out mode. If both VRF and source interface are configured, the source interface must be in the same VRF as the one specified under destination group for the session to be established.

For more information about the dial-out mode, see Dial-out Mode.

The process to configure a dial-out mode involves:

Create a Destination Group

The destination group specifies the destination address, port, encoding and transport that the router uses to send out telemetry data.

A VRF in the destination group implies that the connection to the destination must be created in the specified VRF.

1. Identify the destination address, port, transport, and encoding format.
2. Create a destination group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group <group-name>
Router(config-model-driven-dest)#vrf <vrf-name>
Router(config-model-driven-dest)#address family ipv4 <IP-address> port <port-number>
Router(config-model-driven-dest-addr)#encoding <encoding-format>
Router(config-model-driven-dest-addr)#protocol <transport>
Router(config-model-driven-dest-addr)#commit
```

Example: Destination Group for TCP Dial-out

The following example shows a destination group `DGroup1` created for TCP dial-out configuration with key-value Google Protocol Buffers (also called self-describing-gpb) encoding:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup1
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 5432
Router(config-model-driven-dest-addr)#encoding self-describing-gpb
Router(config-model-driven-dest-addr)#protocol tcp
Router(config-model-driven-dest-addr)#commit
```

Example: Destination Group for UDP Dial-out

The following example shows a destination group `DGroup1` created for UDP dial-out configuration with key-value Google Protocol Buffers (also called self-describing-gpb) encoding:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup1
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 5432
Router(config-model-driven-dest-addr)#encoding self-describing-gpb
Router(config-model-driven-dest-addr)#protocol udp
Router(config-model-driven-dest-addr)#commit
```

The UDP destination is shown as `Active` irrespective of the state of the collector because UDP is connectionless.

Model-driven Telemetry with UDP is not suitable for a busy network. There is no retry if a message is dropped by the network before it reaches the collector.

Example: Destination Group for gRPC Dial-out

Note gRPC is supported in only 64-bit platforms.

gRPC protocol supports TLS and model-driven telemetry uses TLS to dial-out by default. The certificate must be copied to `/misc/config/grpc/dialout/`. To by-pass the TLS option, use `protocol grpc no-tls`.

The following is an example of a certificate to which the server certificate is connected:

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

Wed Aug 24 05:05:46.206 UTC
[xr-vm_node0_RP0_CPU0:~]$ls -l /misc/config/grpc/dialout/
total 4
-rw-r--r-- 1 root root 4017 Aug 19 19:17 dialout.pem
[xr-vm_node0_RP0_CPU0:~]$
```

The CN (CommonName) used in the certificate must be configured as `protocol grpc tls-hostname <>`.

The following example shows a destination group `DGroup2` created for gRPC dial-out configuration with key-value GPB encoding, and with tls disabled:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup2
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 57500
Router(config-model-driven-dest-addr)#encoding self-describing-gpb
```

```
Router(config-model-driven-dest-addr)#protocol grpc no-tls
Router(config-model-driven-dest-addr)#commit
```

The following example shows a destination group `DGroup2` created for gRPC dial-out configuration with key-value GPB encoding, and with tls hostname:

```
Configuration with tls-hostname:
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup2
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 57500
Router(config-model-driven-dest-addr)#encoding self-describing-gpb
Router(config-model-driven-dest-addr)#protocol grpc tls-hostname hostname.com
Router(config-model-driven-dest-addr)#commit
```

Note If only the **protocol grpc** is configured without tls option, tls is enabled by default and tls-hostname defaults to the IP address of the destination.

What to Do Next:

Create a sensor group.

Create a Sensor Group

The sensor-group specifies a list of YANG models that are to be streamed.

1. Identify the sensor path for XR YANG model.
2. Create a sensor group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group <group-name>
Router(config-model-driven-snsr-grp)# sensor-path <XR YANG model>
Router(config-model-driven-snsr-grp)# commit
```

Example: Sensor Group for Dial-out

Note gRPC is supported in only 64-bit platforms.

The following example shows a sensor group `SGroup1` created for dial-out configuration with the YANG model for interface statistics:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group SGroup1
Router(config-model-driven-snsr-grp)# sensor-path
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters
Router(config-model-driven-snsr-grp)# commit
```

What to Do Next:

Create a subscription.

Create a Subscription

The subscription associates a destination-group with a sensor-group and sets the streaming method - cadence-based or event-based telemetry.

A source interface in the subscription group specifies the interface that will be used for establishing the session to stream data to the destination. If both VRF and source interface are configured, the source interface must be in the same VRF as the one specified under destination group for the session to be established.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription <subscription-name>
Router(config-model-driven-subs)#sensor-group-id <sensor-group> sample-interval <interval>

Router(config-model-driven-subs)#destination-id <destination-group>
Router(config-model-driven-subs)#source-interface <source-interface>
Router(config-mdt-subscription)#commit
```

Example: Subscription for Cadence-based Dial-out Configuration

The following example shows a subscription `Sub1` that is created to associate the sensor-group and destination-group, and configure an interval of 30 seconds to stream data:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription Sub1
Router(config-model-driven-subs)#sensor-group-id SGroup1 sample-interval 30000
Router(config-model-driven-subs)#destination-id DGroup1
Router(config-mdt-subscription)# commit
```

Example: Configure Event-driven Telemetry for Interface Path

```
telemetry model-driven
 destination-group 1
  address family ipv4 <ip-address> port <port-number>
   encoding self-describing-gpb
   protocol grpc no-tls
  !
 !
 sensor-group 1
  sensor-path
Cisco-IOS-XR-ipv6-ma-oper:ipv6-network/nodes/node/interface-data/vrfs/vrf/global-briefs/global-brief

 !
 sensor-group 2
  sensor-path Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface
 !
 subscription 1
  sensor-group-id 1 sample-interval 0
  sensor-group-id 2 sample-interval 0
  destination-id 1
!
```

Example: Subscription for Event-based Dial-out Configuration

The following example shows a subscription `Sub1` that is created to associate the sensor-group and destination-group, and configure event-based method to stream data:

```
Router(config)#telemetry model-driven
```

```
Router(config-model-driven)#subscription Sub1
Router(config-model-driven-subs)#sensor-group-id SGroup1 sample-interval 0
Router(config-model-driven-subs)#destination-id DGroup1
Router(config-mdt-subscription)# commit
```

What to Do Next:

Validate the configuration.

Validate Dial-out Configuration

Use the following command to verify that you have correctly configured the router for dial-out.

```
Router#show telemetry model-driven subscription <subscription-group-name>
```

Example: Validation for TCP Dial-out

```
Router#show telemetry model-driven subscription Sub1
Thu Jul 21 15:42:27.751 UTC
Subscription:  Sub1                    State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  SGroup1           30000               Resolved

  Destination Groups:
  Id                Encoding            Transport   State   Port    IP
  DGroup1           self-describing-gpb tcp         Active  5432    172.0.0.0
```

Example: Validation for gRPC Dial-out

Note gRPC is supported in only 64-bit platforms.

```
Router#show telemetry model-driven subscription Sub2
Thu Jul 21 21:14:08.636 UTC
Subscription:  Sub2                    State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  SGroup2           30000               Resolved

  Destination Groups:
  Id                Encoding            Transport   State   Port    IP
  DGroup2           self-describing-gpb grpc        ACTIVE      57500   172.0.0.0
```

The telemetry data starts steaming out of the router to the destination.

Example: Configure model-driven telemetry with different sensor groups

```
RP/0/RP0/CPU0:ios#sh run telemetry model-driven

Wed Aug 24 04:49:19.309 UTC

telemetry model-driven
 destination-group 1
```

```
 address family ipv4 1.1.1.1 port 1111
  protocol grpc
 !
!

destination-group 2
 address family ipv4 2.2.2.2 port 2222
 !
!

destination-group test
 address family ipv4 172.0.0.0 port 8801
  encoding self-describing-gpb
  protocol grpc no-tls
 !
 address family ipv4 172.0.0.0 port 8901
  encoding self-describing-gpb
  protocol grpc tls-hostname chkpt1.com
 !
!

sensor-group 1
 sensor-path Cisco-IOS-XR-plat-chas-invmgr-oper:platform-inventory/racks/rack
!

sensor-group mdt
 sensor-path Cisco-IOS-XR-telemetry-model-driven-oper:telemetry-model-driven
!

sensor-group generic
 sensor-path
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters

!

sensor-group if-oper
 sensor-path Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface
!

subscription mdt
 sensor-group-id mdt sample-interval 10000
!

subscription generic
 sensor-group-id generic sample-interval 10000
!

subscription if-oper
 sensor-group-id if-oper sample-interval 10000
 destination-id test
!
!
```

A sample output from the destination with TLS certificate `chkpt1.com`:

```
RP/0/RP0/CPU0:ios#sh telemetry model-driven dest

Wed Aug 24 04:49:25.030 UTC
  Group Id        IP              Port    Encoding            Transport   State

  ------------------------------------------------------------------------------
  1               1.1.1.1         1111    none                grpc        ACTIVE
      TLS:1.1.1.1
  2               2.2.2.2         2222    none                grpc        ACTIVE
      TLS:2.2.2.2
```

```
  test            172.0.0.0   8801    self-describing-gpb grpc        Active
  test            172.0.0.0   8901    self-describing-gpb grpc        Active
      TLS:chkpt1.com
```

A sample output from the subscription:

```
RP/0/RP0/CPU0:ios#sh telemetry model-driven subscription

Wed Aug 24 04:49:48.002 UTC
Subscription:  mdt                     State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  mdt               10000               Resolved

Subscription:  generic                 State: ACTIVE

-------------
  Sensor groups:
  Id                Interval(ms)        State
  generic           10000               Resolved

Subscription:  if-oper                 State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  if-oper           10000               Resolved

  Destination Groups:
  Id                Encoding            Transport   State   Port    IP
  test              self-describing-gpb grpc        ACTIVE  8801    172.0.0.0

    No TLS :

  test              self-describing-gpb grpc        Active  8901    172.0.0.0
    TLS :             chkpt1.com


RP/0/RP0/CPU0:ios#sh telemetry model-driven subscription if-oper

Wed Aug 24 04:50:02.295 UTC
Subscription:  if-oper
-------------
  State:       ACTIVE
  Sensor groups:
  Id: if-oper
    Sample Interval:      10000 ms
    Sensor Path:          Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: test
    Destination IP:       172.0.0.0
    Destination Port:     8801
    Encoding:             self-describing-gpb
    Transport:            grpc
    State:                ACTIVE
    No TLS
    Destination IP:       172.0.0.0
    Destination Port:     8901
    Encoding:             self-describing-gpb
    Transport:            grpc
    State:                ACTIVE
    TLS :                 chkpt1.com
```

```
  Total bytes sent:     120703
  Total packets sent:   11
  Last Sent time:       2016-08-24 04:49:53.52169253 +0000

Collection Groups:
------------------
  Id: 1
  Sample Interval:      10000 ms
  Encoding:             self-describing-gpb
  Num of collection:    11
  Collection time:      Min:    69 ms Max:    82 ms
  Total time:           Min:    69 ms Avg:    76 ms Max:    83 ms
  Total Deferred:       0
  Total Send Errors:    0
  Total Send Drops:     0
  Total Other Errors:   0
  Last Collection Start:2016-08-24 04:49:53.52086253 +0000
  Last Collection End:  2016-08-24 04:49:53.52169253 +0000
  Sensor Path:          Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface
```

Example: Configure Event-driven Telemetry for LLDP

Telemetry supports NETCONF event notifications where the NETCONF client is configured to receive event notifications from a NETCONF server through a subscription. The NETCONF client must subscribe using a `create-subscription` request. Currently, only the events from Link Layer Discovery Protocol (LLDP) is supported. These event notifications are sent until either the NETCONF session or the subscription is terminated.

Note Configuring a sensor group and a subscription is not required for receiving NETCONF notifications. While sensor path and subscription configurations are required for receiving telemetry events, NETCONF `create-subscription` is required for receiving NETCONF notifications.

To generate NETCONF notifications:

1. Enable NETCONF agent and SSH sub system.

```
ssh server netconf
netconf-yang agent ssh
```

2. Enable model-driven telemetry.

```
telemetry model-driven
```

3. Enable LLDP.

```
lldp
```

This example shows event-driven telemetry fo LLDP configuration data.

1. Create a destination group.

```
grpc
port 56782
address-family ipv4
!
telemetry model-driven
destination-group <destination-udp>
  address-family ipv4 <client-ip>1 port <udp port num>
```

```
   encoding self-describing-gpb
   protocol udp
  !
!
destination-group <destination-tcp>
  address-family ipv4 <client-ip> port <tcp port num>
   encoding gpb
   protocol tcp
  !
destination-group <destination-grpc>
  address-family ipv4 <grpc client ip>port <grpc port num>
   encoding self-describing-gpb
   protocol grpc no-tls
```

2. Create a sensor group.

```
sensor-group <sensor-group-name>
  sensor-path Cisco-IOS-XR-ethernet-lldp-oper:lldp/global-lldp/lldp-info
  sensor-path Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/interfaces/interface
  sensor-path Cisco-IOS-XR-ethernet-lldp-oper:lldp/nodes/node/neighbors/details/detail

 !
```

3. Create a subscription.

```
subscription udp-out
  sensor-group-id <sensor-group-name> sample-interval 0
  destination-id <destination-udp>
!

subscription <subscription-name>
  sensor-group-id <sensor-group-name> sample-interval 0
  destination-id <destination-tcp>

subscription <subscription-name>
  sensor-group-id <sensor-group-name> sample-interval 0
!
netconf-yang agent
ssh
!
```

4. Set the notification to stream data when an event occurs.

```
Router(config-lldp)#timer 12
Router(config-lldp)#commit

Router(config-lldp)#holdtime 150
Router (config-lldp)#commit

Router (config-lldp)#exit
#506
<?xml version="1.0"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>Date-and-Time</eventTime>
 <lldp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ethernet-lldp-oper">
  <global-lldp>
   <lldp-info>
    <chassis-id>000b.1bc9.e700</chassis-id>
    <chassis-id-sub-type>4</chassis-id-sub-type>
    <system-name>ios</system-name>
    <timer>12</timer>
    <hold-time>120</hold-time>
    <re-init>2</re-init>
```

```
   </lldp-info>
  </global-lldp>
 </lldp>
</notification>
Ready to send a request.
Paste your request or enter 'get', 'get-config', 'create-sub', or 'bye' to quit):
```

5. Validate response received from NETCONF agent.

```
#506
<?xml version="1.0"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>Date-and-Time</eventTime>
 <lldp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ethernet-lldp-oper">
  <global-lldp>
   <lldp-info>
    <chassis-id>000b.1bc9.e700</chassis-id>
    <chassis-id-sub-type>4</chassis-id-sub-type>
    <system-name>ios</system-name>
    <timer>12</timer>
    <hold-time>150</hold-time>
    <re-init>2</re-init>
   </lldp-info>
  </global-lldp>
 </lldp>
</notification>
```

Configure Dial-in Mode

In a dial-in mode, the destination initiates a session to the router and subscribes to data to be streamed.

Note Dial-in mode is supported over gRPC in only 64-bit platforms.

For more information about dial-in mode, see Dial-in Mode.

The process to configure a dial-in mode involves these tasks:

- Enable gRPC
- Create a sensor group
- Create a subscription
- Validate the configuration

Enable gRPC

Configure the gRPC server on the router to accept incoming connections from the collector.

1. Enable gRPC over an HTTP/2 connection.

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

2. Enable access to a specified port number.

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

The <port-number> range is from 57344 to 57999. If a port number is unavailable, an error is displayed.

3. In the configuration mode, set the session parameters.

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

where:

- **address-family:** set the address family identifier type
- **dscp:** set QoS marking DSCP on transmitted gRPC
- **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.
- **service-layer:** enable the grpc service layer configuration
- **tls-cipher:** enable the gRPC TLS cipher suites
- **tls-mutual:** set the mutual authentication
- **tls-trustpoint:** configure trustpoint
- **server-vrf:** enable server vrf

4. Commit the configuration.

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

The following example shows the output of `show grpc` command. The sample output displays the gRPC configuration when TLS is enabled on the router.

```
Router#show grpc

Address family              : ipv4
Port                        : 57300
VRF                         : global-vrf
TLS                         : enabled
TLS mutual                  : disabled
Trustpoint                  : none
Maximum requests            : 128
Maximum requests per user   : 10
Maximum streams             : 32
Maximum streams per user    : 32

TLS cipher suites
  Default                   : none
  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
```

What to Do Next:

Create a sensor group.

Create a Sensor Group

The sensor group specifies a list of YANG models that are to be streamed.

1. Identify the sensor path for XR YANG model.
2. Create a sensor group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group <group-name>
Router(config-model-driven-snsr-grp)# sensor-path <XR YANG model>
Router(config-model-driven-snsr-grp)# commit
```

Example: Sensor Group for gRPC Dial-in

The following example shows a sensor group `SGroup3` created for gRPC dial-in configuration with the YANG model for interfaces:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group SGroup3
Router(config-model-driven-snsr-grp)# sensor-path openconfig-interfaces:interfaces/interface

Router(config-model-driven-snsr-grp)# commit
```

What to Do Next:

Create a subscription.

Create a Subscription

The subscription associates a sensor-group with a streaming interval. The collector requests the subscription to the sensor paths when it establishes a connection with the router.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription <subscription-name>
Router(config-model-driven-subs)#sensor-group-id <sensor-group> sample-interval <interval>
```

```
Router(config-model-driven-subs)#destination-id <destination-group>
Router(config-mdt-subscription)#commit
```

Example: Subscription for gRPC Dial-in

The following example shows a subscription `Sub3` that is created to associate the sensor-group with an interval of 30 seconds to stream data:

```
Router(config)telemetry model-driven
Router(config-model-driven)#subscription Sub3
Router(config-model-driven-subs)#sensor-group-id SGroup3 sample-interval 30000
Router(config-mdt-subscription)#commit
```

What to Do Next:

Validate the configuration.

Validate Dial-in Configuration

Use the following command to verify that you have correctly configured the router for gRPC dial-in.

```
Router#show telemetry model-driven subscription
```

Example: Validation for gRPC Dial-in

```
RP/0/RP0/CPU0:SunC#show telemetry model-driven subscription Sub3
Thu Jul 21 21:32:45.365 UTC
Subscription:  Sub3
-------------
  State:       ACTIVE
  Sensor groups:
  Id: SGroup3
    Sample Interval:      30000 ms
    Sensor Path:          openconfig-interfaces:interfaces/interface
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: DialIn_1002
    Destination IP:       172.30.8.4
    Destination Port:     44841
    Encoding:             self-describing-gpb
    Transport:            dialin
    State:                Active
    Total bytes sent:     13909
    Total packets sent:   14
    Last Sent time:       2016-07-21 21:32:25.231964501 +0000

  Collection Groups:
  ------------------
    Id: 2
    Sample Interval:      30000 ms
    Encoding:             self-describing-gpb
    Num of collection:    7
    Collection time:      Min:    32 ms Max:    39 ms
    Total time:           Min:    34 ms Avg:    37 ms Max:    40 ms
    Total Deferred:       0
    Total Send Errors:    0
    Total Send Drops:     0
    Total Other Errors:   0
```

```
Last Collection Start:2016-07-21 21:32:25.231930501 +0000
Last Collection End:  2016-07-21 21:32:25.231969501 +0000
Sensor Path:          openconfig-interfaces:interfaces/interface
```