

# Configurer gNMI et implémenter pYANG dans IOS XR

## Table des matières

---

[Introduction](#)

[Conditions préalables](#)

[Exigences](#)

[Composants utilisés](#)

[Informations générales](#)

[Définition gNMI](#)

[Fonctionnalités gNMI](#)

[Configuration de base gNMI dans Cisco IOS XR](#)

[pYANG comme validateur](#)

[Dépannage :](#)

---

## Introduction

Ce document décrit brièvement gNMI dans Cisco IOS® XR et comment utiliser PYANG et vérifier les arborescences de modèles.

## Conditions préalables

### Exigences

Cisco vous recommande de prendre connaissance des rubriques suivantes :

- Plate-forme Cisco IOS XR.
- python.
- Protocoles de gestion du réseau.

### Composants utilisés

Ce document n'est pas limité à des versions matérielles spécifiques, il s'applique à la version 64 bits (eXR).

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. Si votre réseau est en ligne, assurez-vous de bien comprendre l'incidence possible des commandes.

## Informations générales

## Définition gNMI

Dans l'ensemble, il existe différents protocoles de configuration réseau, tels que NETCONF, RESTCONF, gNMI (Google Remote Procedure Calls (gRPC), gRPC Network Management Interface), entre autres. Ces modèles sont utilisés pour configurer la gestion des périphériques réseau et visent toujours à automatiser les processus qui peuvent être mécaniques.

Ces protocoles utilisent différents modèles de données pour permettre aux utilisateurs de comprendre le processus du périphérique réseau, c'est-à-dire une information structurée, un schéma, qui normalise l'information et la façon dont elle est utilisée par le périphérique, dans ce cas, le routeur.

gNMI supervise le traitement des données et fournit RPC (Remote Procedure Calls) pour contrôler les différents périphériques du réseau.

gNMI a quatre fonctions :

- Privilèges : gNMI demande au routeur les modèles qui sont installés dans le routeur, ceci est expliqué plus loin dans ce document.
- GET : Chaque composant leaf dans l'arborescence de données peut être demandé au routeur, cette opération demande les informations demandées.
- Set : Les leafs sont considérés comme des variables, ce qui leur fournit les capacités de changement, Set opération assiste sur ce qui permet à l'utilisateur de mettre à jour une valeur dans le modèle de données.
- Abonnement : Utilisée en télémétrie, cette fonction permet d'extraire des données d'un module particulier du modèle.



Remarque : Cisco a partagé de nombreuses informations à ce sujet. Pour plus d'informations sur gRPC, cliquez sur le lien suivant : [xrdocs blog - OpenConfig gNMI](#)

---

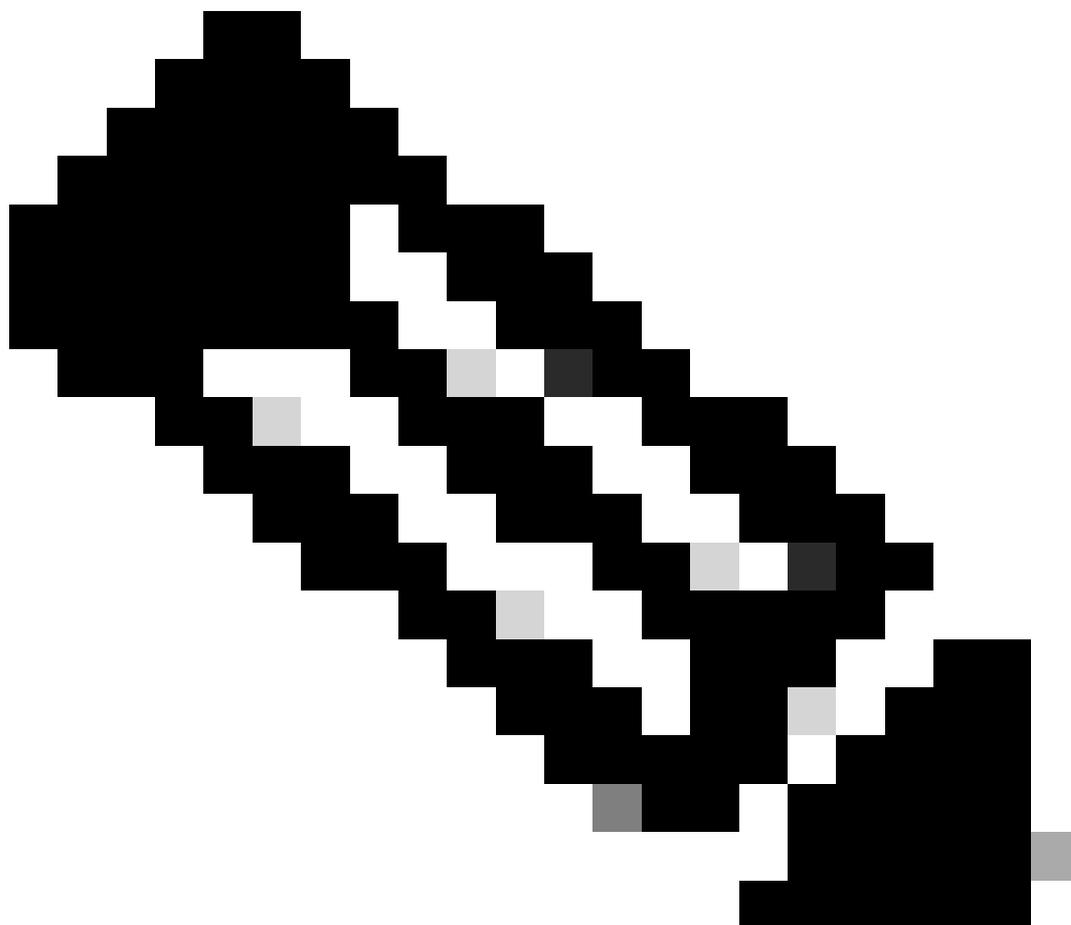
## Fonctionnalités gNMI

Protocole de gestion de réseau	gNMI
Transport utilisé	HTTP/2
Pris en charge par	Neutre fournisseur
Codage	Proto Buff

Proto Buff est la méthode de désérialisation et de sérialisation des données entre deux

périphériques, neutre du point de vue du langage et de la plate-forme, dans laquelle chaque requête a une réponse.

---



Remarque : Pour plus de détails dans gRPC et Proto Buff cliquez sur le lien suivant : [Guide grpc.](#)

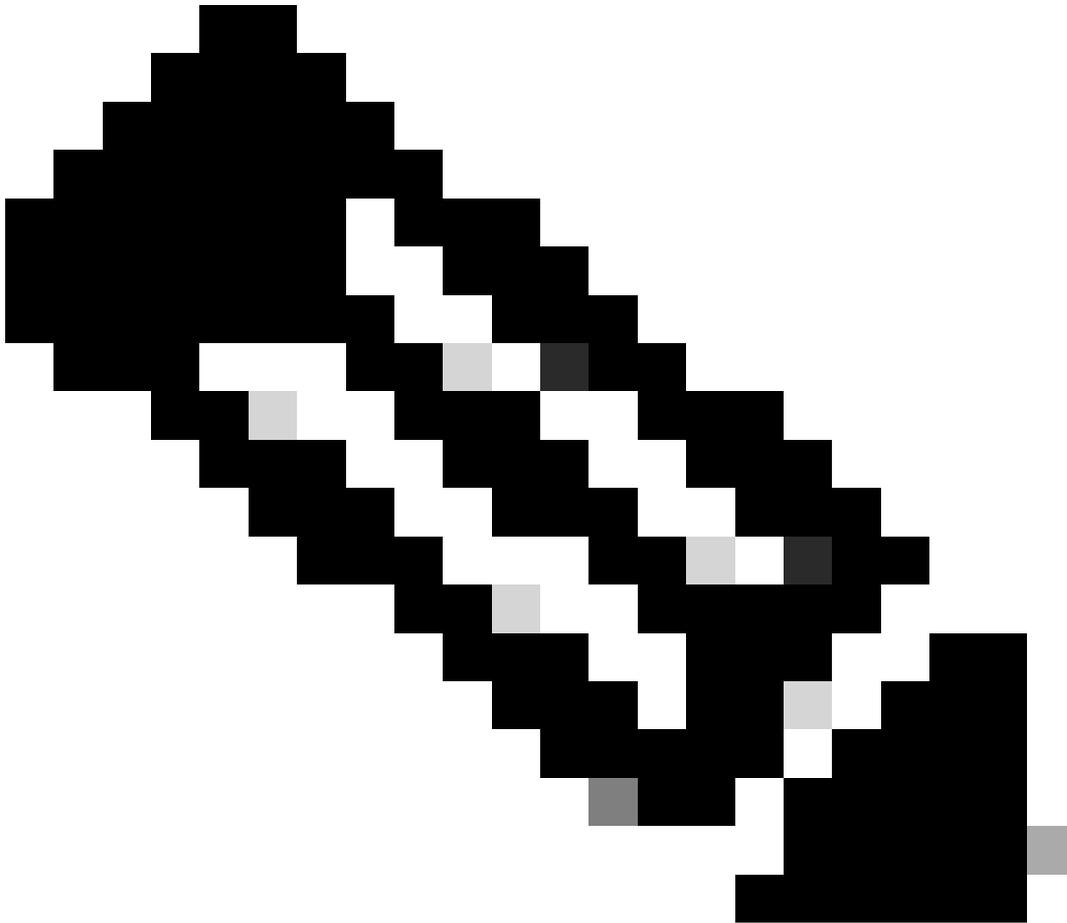
---

## Configuration de base gNMI dans Cisco IOS XR

La configuration de base du routeur est la suivante :

```
RP/0/RSP0/CPU0:XR(config)#grpc
RP/0/RSP0/CPU0:XR(config-grpc)#address-family ipv4
RP/0/RSP0/CPU0:XR(config-grpc)#max-request-total 256
RP/0/RSP0/CPU0:XR(config-grpc)#max-request-per-user 32
```

```
grpc
address-family ipv4
max-request-total 256
```



Remarque : Un port peut être configuré en fonction de la configuration. La valeur par défaut, sans utiliser TLS, est 57400. Pour plus d'informations, cliquez sur : [github - démarrage de grpc](#)

---

## pYANG comme valideur

pYANG est un valideur YANG écrit en python. Cette bibliothèque pour python aide à vérifier les modèles YANG et aussi, les connaître.

Pour que cela fonctionne comme dans la documentation ([documentation pYANG](#)), il est suggéré de créer un environnement virtuel dans l'ordinateur.

Pour que l'environnement virtuel exécute la [documentation venv](#)

Il est nécessaire d'exécuter :

```
python -m venv
```

Par exemple (dans un terminal MacOS) :

```
% mkdir test
% cd test
% python3 -m venv virtual_env
% ls
virtual_env
```

Pour installer pYANG dans cet environnement virtuel, placez le cd dans le répertoire et collez le suivant :

```
% cd virtual_env
% git clone https://github.com/mbj4668/pyang.git
% cd pyang
% pip install -e .
```

Pour cette démonstration, python3 pip a été utilisé, une fois que la commande pip install -e est émise, activez la commande venv: source <virtual environment directory>/bin/activate (pour MacOS).

```
% source virtual_env/bin/activate
```

```
% python3 -m pip install pyang
```

```
Collecting pyang
```

```
  Downloading pyang-2.6.0-py2.py3-none-any.whl (594 kB)
```

```
    |████████████████████████████████████████| 594 kB 819 kB/s
```

```
Collecting lxml
```

```
  Downloading lxml-5.1.0-cp39-cp39-macosx_11_0_arm64.whl (4.5 MB)
```

```
    |████████████████████████████████████████| 4.5 MB 14.2 MB/s
```

```
Installing collected packages: lxml, pyang
```

```
Successfully installed lxml-5.1.0 pyang-2.6.0
```

```
% pyang -h
```

```
Usage: pyang [options] [
```

...]

Validates the YANG module in

(or stdin), and all its dependencies.

Options:

-h, --help	Show this help message and exit
-v, --version	Show version number and exit

Une fois pYANG installé et opérationnel, poursuivez le téléchargement des modèles.

Le lien suivant présente tous les modèles exécutés par Cisco IOS XR : [Modèles Cisco IOS XR](#).

Il est suggéré de cloner git ce modèle dans le répertoire venv avec le lien de code suivant :  
<https://github.com/YangModels/yang.git>



Remarque : Cela n'est pas possible lorsque l'environnement virtuel est activé.

---

```
% git clone https://github.com/YangModels/yang.git
Cloning into 'yang'...
remote: Enumerating objects: 54289, done.
remote: Counting objects: 100% (1910/1910), done.
remote: Compressing objects: 100% (323/323), done.
remote: Total 54289 (delta 1643), reused 1684 (delta 1586), pack-reused 52379
Receiving objects: 100% (54289/54289), 116.64 MiB | 8.98 MiB/s, done.
Resolving deltas: 100% (42908/42908), done.
Updating files: 100% (112197/112197), done.
```

Activez à nouveau l'environnement virtuel et testez la requête suivante : `pyang -f tree yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang`.

```
(virtual_env) % pyang -f tree yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang
yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang:5: error: module "Cisco-IOS-XR-types" not found in
```

yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang:8: error: module "cisco-semver" not found in search  
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 args? enumeration
      | +--rw half-life? uint32
      | +--rw reuse-threshold? uint32
      | +--rw suppress-threshold? uint32
      | +--rw suppress-time? uint32
      | +--rw restart-penalty? uint32
    +--rw mtus
      | +--rw mtu* [owner]
      |   +--rw owner xr:Cisco-ios-xr-string
      |   +--rw mtu uint32
    +--rw encapsulation
      | +--rw encapsulation? string
      | +--rw capsulation-options? uint32
    +--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
```



Remarque : Notez que les leafs ont un format de données comme String, uint32, etc., etc.; tandis que root n'affiche pas cette information. Les opérations telles que GET et SET sont dédiées à l'extraction/la mise à jour de ces valeurs.

---

Une autre remarque est que la plupart des modèles nécessitent des ajouts pour avoir la configuration complète, dans le résultat CLI il y a la configuration de gestion d'interface de base, dans le cas où IPv4 doit être affiché, utilisez la suivante :

```
% pyang -f tree yan2/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang yan2/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-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 args?          enumeration
  | | | +--rw half-life?    uint32
  | | | +--rw reuse-threshold? uint32
```

```

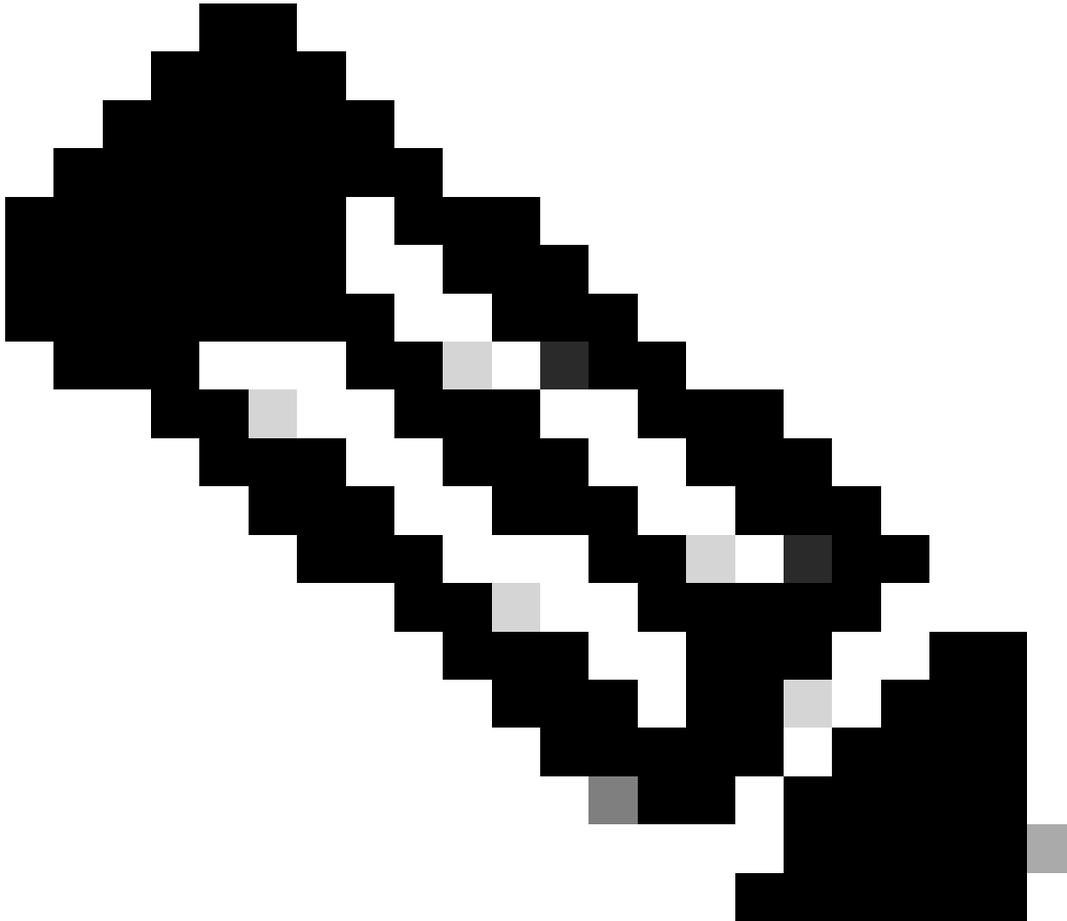
| +--rw suppress-threshold? uint32
| +--rw suppress-time?      uint32
| +--rw restart-penalty?    uint32
+--rw mtus
| +--rw mtu* [owner]
|   +--rw owner      xr:Cisco-ios-xr-string
|   +--rw mtu        uint32
+--rw encapsulation
| +--rw encapsulation?      string
| +--rw capsulation-options? uint32
+--rw 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:bgp-pa
| | +--rw ipv4-io-cfg:input
| | | +--rw ipv4-io-cfg:source-accounting? boolean
| | | +--rw ipv4-io-cfg:destination-accounting? boolean
| | +--rw ipv4-io-cfg:output
| |   +--rw ipv4-io-cfg:source-accounting? boolean
| |   +--rw ipv4-io-cfg:destination-accounting? boolean
| +--rw ipv4-io-cfg:verify
| | +--rw ipv4-io-cfg:reachable? Ipv4-reachable
| | +--rw ipv4-io-cfg:self-ping? Ipv4-self-ping
| | +--rw ipv4-io-cfg:default-ping? Ipv4-default-ping
| +--rw ipv4-io-cfg:bgp
| | +--rw ipv4-io-cfg:qppb
| | | +--rw ipv4-io-cfg:input
| | |   +--rw ipv4-io-cfg:source? Ipv4-interface-qppb
| | |   +--rw ipv4-io-cfg:destination? Ipv4-interface-qppb
| | +--rw ipv4-io-cfg:flow-tag
| |   +--rw ipv4-io-cfg:flow-tag-input
| |   +--rw ipv4-io-cfg:source? boolean
| |   +--rw ipv4-io-cfg:destination? boolean
| +--rw ipv4-io-cfg:addresses
| | +--rw ipv4-io-cfg:secondaries
| | | +--rw ipv4-io-cfg:secondary* [address]
| | |   +--rw ipv4-io-cfg:address      inet:ipv4-address-no-zone
| | |   +--rw ipv4-io-cfg:netmask      inet:ipv4-address-no-zone
| | |   +--rw ipv4-io-cfg:route-tag?   uint32
| | +--rw ipv4-io-cfg:primary!
| | | +--rw ipv4-io-cfg:address      inet:ipv4-address-no-zone
| | | +--rw ipv4-io-cfg:netmask      inet:ipv4-address-no-zone
| | | +--rw ipv4-io-cfg:route-tag?   uint32
| | +--rw ipv4-io-cfg:unnumbered?    xr:Interface-name
| | +--rw ipv4-io-cfg:dhcp?          empty
| +--rw ipv4-io-cfg:helper-addresses
| | +--rw ipv4-io-cfg:helper-address* [address vrf-name]
| |   +--rw ipv4-io-cfg:address      inet:ipv4-address-no-zone
| |   +--rw ipv4-io-cfg:vrf-name     xr:Cisco-ios-xr-string
| +--rw ipv4-io-cfg:forwarding-enable? empty
| +--rw ipv4-io-cfg:icmp-mask-reply? empty
| +--rw ipv4-io-cfg:tcp-mss-adjust-enable? empty
| +--rw ipv4-io-cfg:ttl-propagate-disable? empty
| +--rw ipv4-io-cfg:point-to-point? empty
| +--rw ipv4-io-cfg:mtu?            uint32

```

```
+-rw ipv4-io-cfg:ipv4-network-forwarding
+-rw ipv4-io-cfg:directed-broadcast? empty
+-rw ipv4-io-cfg:unreachables? empty
+-rw ipv4-io-cfg:redirects? empty
```

Dans cette requête, deux modèles sont utilisés : Cisco-IOS-XR-ifmgr-cfg.yang et Cisco-IOS-XR-ipv4-io-cfg.yang. L'adresse IPv4 s'affiche désormais sous la forme d'un leaf.

---



Remarque : au cas où vous verriez une erreur comme : "yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang:5: erreur : module "Cisco-IOS-XR-types" not found in search path" ajoutez la commande `—path=` dans la commande.

---

Ceci fait et coché, tout utilisateur peut demander des informations avec les opérations gNMI et changer la date, pour plus d'exemples cliquez sur le lien suivant : [Guide de configuration de programmabilité](#)

Si l'utilisateur souhaite exécuter une API simple, il existe des outils tels que : [grpcc](#).

Cette API est installée via NPM, c'est l'outil utilisé dans le lien Guide de configuration de la programmabilité, ce lien partage plus d'exemples pour les utilisateurs pour tester les requêtes et les réponses.

## Dépannage :

Pour gNMI, il est nécessaire de vérifier la requête avant de collecter une entrée, la plupart des API comme :

- gnmique
- grpcc
- gRPC

Tous, affiche l'erreur générée par le routeur.

## Exemple :

```
"cisco-grpc:errors": {
  "error": [
    {
      "error-type": "application",
      "error-tag": "operation-failed",
      "error-severity": "error",
      "error-message": "'YANG framework' detected the 'fatal' condition 'Operation failed'"
    }
  ]
}
```

OU

```
"error": [
  {
    "error-type": "application",
    "error-tag": "operation-failed",
    "error-severity": "error",
    "error-path":
      ,
    "error-message": "'sysdb' detected the 'warning' condition 'A verifier or EDM callback function returned"
  }
]
```

Il s'agit d'erreurs dépendant de la plate-forme qui doivent être vérifiées sur le routeur. Il est

conseillé de vérifier que les commandes de la requête peuvent également être exécutées sur le routeur via l'interface de ligne de commande.

Pour ce type d'erreurs, ou toute autre erreur liée à la plate-forme Cisco IOS XR, partagez les informations suivantes avec le TAC :

- Requête utilisée et opération :

```
{
  "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"
          }
        }
      }
    }
  ]
}
```

- Erreur qui s'affiche (l'une des erreurs indiquées ci-dessus).
- Exécutez la commande suivante :

```
show grpc trace all
```

Testez la requête plusieurs fois et répétez la commande « show grpc trace all ».

Les erreurs sont des variantes, mais elles indiquent également le composant qui peut générer un problème :

Exemple :

- "'sysdb' a détecté la condition 'warning' 'Un vérificateur ou une fonction de rappel EDEDM a renvoyé : 'introuvable'" : Cette erreur décrit sysdb. Il est nécessaire de collecter les commandes show tech pour ce processus dans le routeur.

L'exemple suivant montre le processus show tech pour sysdb montrant l'erreur.

```
show tech-support sysdb
```

Pour cette sortie, l'erreur affiche un composant et l'erreur, recueillez tout show tech-support qui peut être lié à l'erreur affichée.

- "'Cadre YANG' a détecté la condition 'fatale' 'Échec de l'opération'" : Cette erreur n'affiche pas de processus dans le routeur, ce qui signifie que la requête échoue dans le modèle. Partagez ces informations avec le TAC pour examiner ce qui peut échouer.

Une fois ces informations collectées, ajoutez également l'ensemble de commandes suivant :

Dans la machine virtuelle XR :

```
show tech-support tctcsr
```

```
show tech-support grpcc
```

```
show tech-support gsp
```

```
show tech-support
```

À propos de cette traduction

Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.