# Cisco NCS 560 シリーズ ルータ（Cisco IOS XR リリース 7.0.x）プログラマビリティ コンフィギュレーション ガイド

**初版**：2019 年 8 月 30 日


**シスコシステムズ合同会社**
〒107-6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp
お問い合わせ先：シスコ コンタクトセンター
0120-092-255 （フリーコール、携帯・PHS含む）
電話受付時間：平日 10:00～12:00、13:00～17:00
http://www.cisco.com/jp/go/contactcenter/

【注意】シスコ製品をご使用になる前に、安全上の注意（ **www.cisco.com/jp/go/safety_warning/** ）をご確認ください。本書は、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。また、契約等の記述については、弊社販売パートナー、または、弊社担当者にご確認ください。

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED “AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com go trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2019 Cisco Systems, Inc. All rights reserved.

# 目 次


第 1 章 新機能および変更機能に関する情報 1

新規および変更されたプログラマビリティ機能 1

第 2 章 プログラム可能な YANG データ モデルを使用したネットワーク自動化の促進 3

YANG データ モデル 5

YANG モジュールのコンポーネント 5

YANG データ モデルの構造 6

データ モデルへのアクセス 8

コミュニケーション プロトコル 9

NETCONF プロトコル 9

gRPC プロトコル 10

第 3 章 NETCONF プロトコルを使用したデータ モデルによるネットワーク運用の定義 11

NETCONF の操作 12

NETCONF セッションでのデータ モデルを使用したルータ クロックの設定 16

ルータ クロックの設定 18

ルータ クロックの表示 20

第 4 章 gRPC プロトコルを使用したデータ モデルによるネットワーク運用の定義 21

gRPC の操作 24

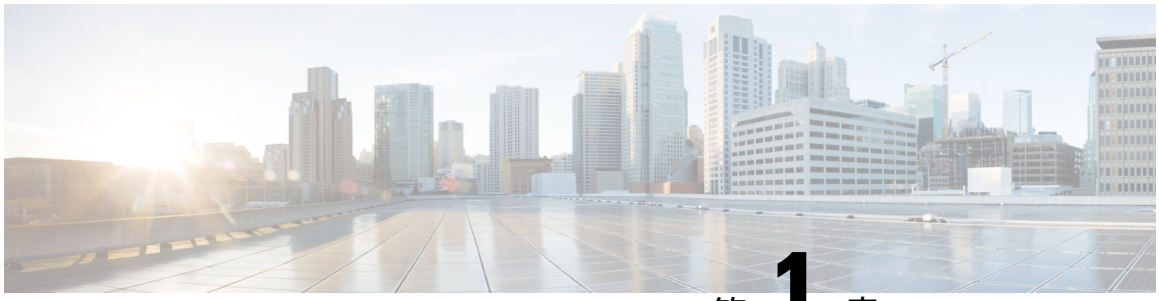
gRPC ネットワーク管理インターフェイス 25

gRPC ネットワーク操作インターフェイス 26

gRPC セッションでのデータ モデルを使用したインターフェイスの設定 28

gRPC プロトコルの有効化 30

インターフェイスの設定　31
インターフェイスの状態の確認　33



第 1 章

# 新機能および変更機能に関する情報

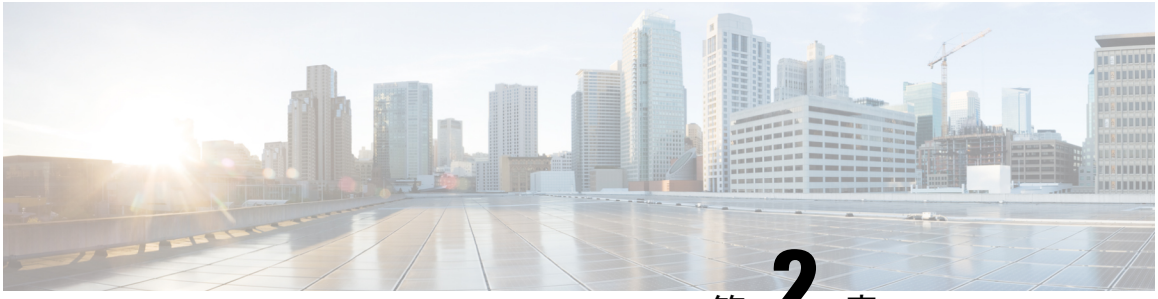
ここでは、『プログラマビリティ コンフィギュレーション ガイド』の新機能および変更された機能をすべて示します。


- 新規および変更されたプログラマビリティ機能 （1 ページ）


## 新規および変更されたプログラマビリティ機能

| 機能 | 説明 | 変更されたリリース | 参照先 |
|---|---|---|---|
| GNOI | gRPC ネットワーク オペレーションインターフェイス（gNOI）は、ネットワークデバイス上で操作コマンドを実行するための gRPC ベースのマイクロサービスセットを定義します。拡張可能な管理サービス（EMS）gNOI は gNOI の Cisco IOS XR 実装です。<br><br>GNOI は、Reboot、RebootStatus、SetPackage、File Get、および File Remove RPC をサポートしています。 | Release 7.0.1 | 「データ モデルを使用するコンポーネント」章<br><br>gRPC ネットワーク操作インターフェイス（26 ページ） |

| 機能 | 説明 | 変更されたリリース | 参照先 |
|---|---|---|---|
| イベント駆動型テレメトリでの `oc-lldp` モデルのサポート | OC コミュニティによって定義された OpenConfig Link Layer Discovery Protocol（oc-lldp）モデルは、LLDP プロトコルの設定と動作状態のデータを定義します。 | Release 7.0.1 | 「データモデルを使用するコンポーネント」章<br>YANG データ モデル（5 ページ）<br>このデータモデルは、Github リポジトリから入手してください。 |
| `oc-if-aggregate` データ モデルのサポート | OC コミュニティによって定義された OpenConfig インターフェイス集約（oc-if-aggregate）モデルは集約（バンドル、リンク アグリゲーション（LAG））インターフェイスを管理します。このモデルは既存の `oc-interfaces` データ モデルを強化します。 | Release 7.0.1 | 「データモデルを使用するコンポーネント」章<br>YANG データ モデル（5 ページ）<br>このデータモデルは、Github リポジトリから入手してください。 |



第 2 章

# プログラム可能な YANG データ モデルを使用したネットワーク自動化の促進

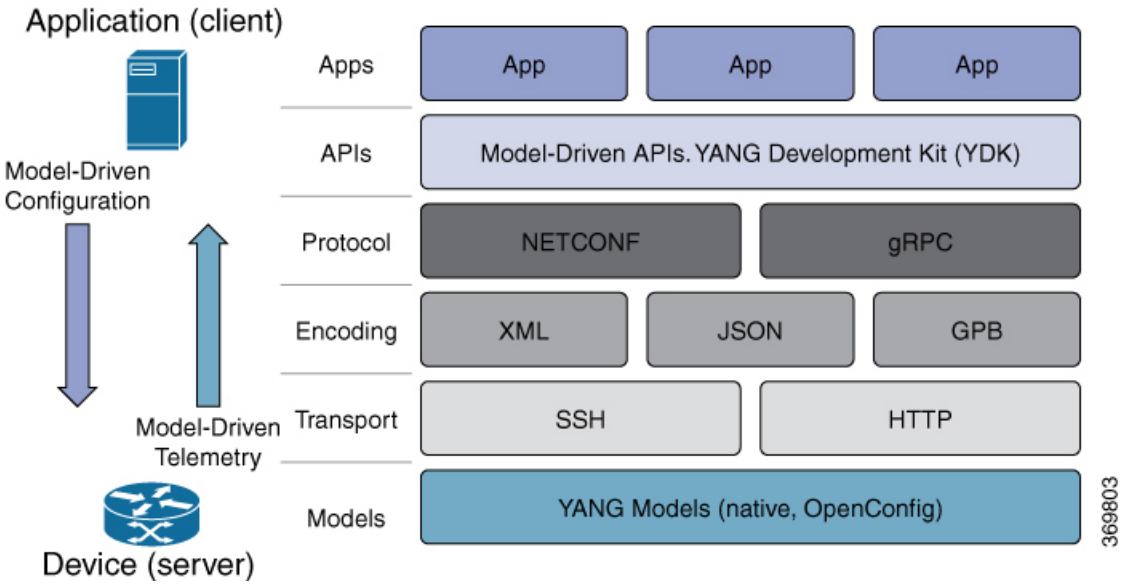
通常、ネットワーク オペレーション センターには、ネットワークの複数のレイヤに多種多様なデバイスが存在しています。このようなネットワーク センターでは、設定が一括で自動化され、シームレスに実行される必要があります。CLI は、ルータの動作の詳細を設定および抽出するために広く使用されています。ただし、CLI スクレイピングの一般的なメカニズムには柔軟性がなく、最適でもありません。設定を少し変更するには、スクリプトを複数回記述する必要があります。CLI を使用して設定を一括で変更すると、複雑になり、エラーが発生しやすくなります。このように自動化や拡張に対する制約があります。これらの制限を克服するには、ネットワークの管理用に自動化されたメカニズムが必要です。

Cisco IOS XR は、ネットワーク デバイスの運用データを設定し、収集するプログラマチックな手法をサポートしています。手動による設定プロセスに代わるもので、独自仕様であり、高度なテキストベースです。データ モデルは、業界で定義された言語で記述されており、ネットワーク内の異種デバイス間で設定タスクを自動化し、運用データを取得するために使用されます。CLI を使用した設定のほうが簡単で判別しやすいですが、モデル駆動型プログラマビリティを使用して設定を自動化すると拡張性が得られます。

モデル駆動型プログラマビリティは、デバイス プログラマビリティにシンプルで柔軟性のあるフレームワークを豊富に提供します。このプログラマビリティ フレームワークは、トランスポート、プロトコル、およびエンコーディングの観点から、IOS XR デバイスとのやり取りに複数のインターフェイスを提供します。このインターフェイスは、柔軟性向上のためにモデルから分離されています。

次の図は、モデル駆動型プログラマビリティの層を示しています。

図 *1:* モデル駆動型プログラマビリティの層




データ モデルでは、ネットワーク設定プロトコル（NETCONF プロトコル）または google で定義されたリモートプロシージャ コール（gRPC プロトコル）を使用してネットワーク内のデバイスの機能にアクセスできます。ルータ上の操作は YANG モデルを使用してプロトコルによって実行され、ネットワーク内の操作を自動化およびプログラムします。

### データ モデルの利点

データ モデルを使用したルータ設定では、従来のルータの管理がもたらす欠点が解消されます。これはデータ モデルで次のことが行われるためです。

- 設定データと運用状態データに共通するモデルを提供し、NETCONF アクションを実行します。
- プロトコルを使用してルータと通信し、ネットワーク内の設定を取得、操作、および削除します。
- ネットワーク全体の複数のルータの設定と操作を自動化します。

ここでは、データ モデルを使用してネットワーク運用をプログラム的に管理する際の利点について説明します。


- YANG データ モデル （5 ページ）
- データ モデルへのアクセス （8 ページ）
- コミュニケーション プロトコル （9 ページ）

# YANG データ モデル

YANG モジュールは、ルータのデータを介してデータ モデルを定義し、そのデータに対する階層的な組織と制約を定義します。各モジュールは、名前空間 URL によって一意に識別されます。YANG モデルはネットワーク デバイスの設定データと運用データ、実行アクション、リモート プロシージャ コール、および通知を記述します。

YANGモデルはルータから取得する必要があります。モデルは、ルータとクライアントの間で交換されるデータの有効な構造を定義します。モデルは NETCONF および gRPC 対応アプリケーションで使用されます。

YANG モデルは次のとおりです。

- **シスコ固有のモデル**：サポート対象モデルおよびその表記のリストについては、「ネイティブ モデル」を参照してください。
- **共通モデル**：これらのモデルは、IETF や IEEE などの標準化機関の業界全体の標準 YANG モデルです。また、これらのモデルは Open Config（OC）モデルとも呼ばれます。合成モデルの場合と同様に、OC モデルには、設定データ、運用データ、アクションに対して定義された個別の YANG モデルがあります。

サポートされている OC モデルとその表記のリストについては、「OC モデル」を参照してください。

YANG の詳細については、RFC 6020 および 6087 を参照してください。

## YANG モジュールのコンポーネント

YANG モジュールでは、単一のデータ モデルを定義します。ただし、モジュールは、次のいずれかのステートメントを使用して他のモジュールおよびサブモジュールで定義を参照できます。

- **import** は外部モジュールをインポートします
- **include** には 1 つ以上のサブモジュールが含まれます
- **augment** は別のモジュールを拡張し、データ モデル階層で新しいノードの配置を定義します
- **when** は新しいノードが有効な条件を定義します
- **prefix** はインポートされたモジュールの定義を参照します

YANGモデルでは、機能の設定、ルータの運用状態の取得、およびアクションの実行が行われます。

（注） gRPC YANG パスまたは JSON データは、YANG 名前空間ではなく、YANG モジュール名に基づいています。

# YANG データ モデルの構造

データ モデルは、ルータ（RFC 6244）で次のタイプの要件を処理します。

- **設定データ**：システムを初期のデフォルト状態から現在の状態に変えるために必要とされる、書き込み可能データのセットです。たとえば、IP ルーティング テーブルのエントリを設定したり、特定の値を使用するようにインターフェイス MTU を設定したり、イーサネット インターフェイスを特定の速度で実行するように設定したりできます。
- **運用状態データ**：実行時にシステムによって取得され、設定データと同様の方法でシステムの動作に影響を与えるデータのセットです。ただし、設定データとは対照的に、運用状態データは一時的なものです。データは、内部コンポーネントまたは特殊なプロトコルを使用する他のシステムとの相互作用によって変更されます。たとえば、OSPF のようなルーティング プロトコル、およびネットワーク インターフェイスの属性などから取得したエントリです。
- **アクション**：堅牢なネットワーク全体の設定トランザクションをサポートする NETCONF アクションのセットです。複数のデバイスに影響を与える変更が試行されると、NETCONF アクションによって障害シナリオの管理が簡略化されます。その結果、確実に成功するか、完全に失敗するトランザクションを利用できるようになります。

データ モデルの詳細については、RFC 6244 を参照してください。

YANG データ モデルはノードのある階層型のツリーベース構造で表現できます。この表現により、モデルを簡単に理解できるようになります。

各機能には、スキーマから合成された定義済みの YANG モデルがあります。ツリー形式のモデルは次のとおりです。

- トップ レベル ノードおよびサブツリー
- 他の YANG モデル内でノードを拡張するサブツリー
- カスタム RPC

YANG は 4 つのノード タイプを定義します。各ノードには名前があります。ノード タイプに応じて、ノードは値を定義するか、一連の子ノードを含めます。データ モデリングの場合、次のノード タイプがあります。

- リーフ ノード：特定のタイプの単一の値が含まれています。
- リーフリスト ノード：一連のリーフ ノードが含まれています。
- リスト ノード：一連のリーフリスト エントリが含まれています。リーフリスト エントリのそれぞれは 1 つ以上のキー リーフによって一意に識別されます。

- コンテナ ノード：子ノードのみを含む関連ノードのグループが含まれます。子ノードは4つのノード タイプのいずれかです。

### CDP データ モデルの構造

Cisco Discovery Protocol（CDP）の設定には、固有の拡張モデル（インターフェイス設定）があります。この拡張は、グローバル設定レベルとインターフェイス設定レベルの両方で CDP を設定できることを示しています。ツリー構造の CDP インターフェイス マネージャのデータ モデルは次のとおりです。

```
module: Cisco-IOS-XR-cdp-cfg
   +--rw cdp
      +--rw timer?               uint32
      +--rw advertise-v1-only?   empty
      +--rw enable?              boolean
      +--rw hold-time?           uint32
      +--rw log-adjacency?       empty
augment /a1:interface-configurations/a1:interface-configuration:
   +--rw cdp
      +--rw enable?   empty
```

CDP YANG モデルでは、拡張は次のように表現されます。

```
augment "/a1:interface-configurations/a1:interface-configuration" {
    container cdp {
      description "Interface specific CDP configuration";
      leaf enable {
        type empty;
        description "Enable or disable CDP on an interface";
      }
    }
    description
      "This augment extends the configuration data of
      'Cisco-IOS-XR-ifmgr-cfg'";
  }
```

### CDP の運用上の YANG：

データ モデルの構造は、pyang などの YANG バリデータ ツールを使用して検証され、データ モデルはツリー構造でフォーマットされます。次に、CDP運用モデルをツリー形式で表示する例を示します。

```
module: Cisco-IOS-XR-cdp-oper
   +--ro cdp
      +--ro nodes
         +--ro node* [node-name]
            +--ro neighbors
            |  +--ro details
            |  |  +--ro detail*
            |  |     +--ro interface-name?   xr:Interface-name
            |  |     +--ro device-id?        string
            |  |     +--ro cdp-neighbor*
            |  |        +--ro detail
            |  |        |  +--ro network-addresses
            |  |        |  |  +--ro cdp-addr-entry*
            |  |        |  |     +--ro address
            |  |        |  |        +--ro address-type?   Cdp-l3-addr-protocol
            |  |        |  |        +--ro ipv4-address?   inet:ipv4-address
            |  |        |  |        +--ro ipv6-address?   In6-addr
```

```
|  |        |  +--ro protocol-hello-list
|  |        |  |  +--ro cdp-prot-hello-entry*
|  |        |  |     +--ro hello-message?   yang:hex-string
|  |        |  +--ro version?               string
|  |        |  +--ro vtp-domain?            string
|  |        |  +--ro native-vlan?           uint32
|  |        |  +--ro duplex?                Cdp-duplex
|  |        |  +--ro system-name?           string
|  |        +--ro receiving-interface-name?   xr:Interface-name
|  |        +--ro device-id?                  string
|  |        +--ro port-id?                    string
|  |        +--ro header-version?             uint8
|  |        +--ro hold-time?                  uint16
|  |        +--ro capabilities?               string
|  |        +--ro platform?                   string

..... (snipped) .....
```

# データ モデルへのアクセス

バージョン管理用のホスティング サービスを提供するソフトウェア開発プラットフォームである GitHub から Cisco IOS XR native および OpenConfig データ モデルにアクセスできます。

また、ルータからサポートされているデータ モデルにアクセスすることもできます。ルータには、データ モデルを定義する YANG ファイルが付属しています。`ietf-netconf-monitoring` 要求を使用してルータで利用可能なデータ モデルを表示するには、NETCONF プロトコルを使用します。

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
 <get>
   <filter type="subtree">
     <netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
     <schemas/>
     </netconf-state>
   </filter>
 </get>
</rpc>
```

サポートされているすべての YANG モデルが RPC 要求に対する応答として表示されます。

```
<rpc-reply message-id="16a79f87-1d47-4f7a-a16a-9405e6d865b9"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<netconf-state xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
<schemas>
<schema>
    <identifier>Cisco-IOS-XR-crypto-sam-oper</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>Cisco-IOS-XR-crypto-sam-oper-sub1</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-crypto-sam-oper</namespace>
    <location>NETCONF</location>
```

```
</schema>
<schema>
    <identifier>Cisco-IOS-XR-snmp-agent-oper</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://cisco.com/ns/yang/Cisco-IOS-XR-snmp-agent-oper</namespace>
    <location>NETCONF</location>
</schema>

------------<snipped>--------------
<schema>
    <identifier>openconfig-aft-types</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/fib-types</namespace>
    <location>NETCONF</location>
</schema>
<schema>
    <identifier>openconfig-mpls-ldp</identifier>
    <version>1.0.0</version>
    <format>yang</format>
    <namespace>http://openconfig.net/yang/ldp</namespace>
    <location>NETCONF</location>
</schema>
</schemas>
</netconf-state>
------------<truncated>--------------
```

# コミュニケーション プロトコル

通信プロトコルはルータとクライアント間の接続を確立します。プロトコルにより、クライアントは YANG データ モデルを使用し、ネットワーク操作を自動化およびプログラムすることができます。

YANG は、次のいずれかのプロトコルを使用します。

- ネットワーク設定プロトコル（NETCONF）
- gRPC（google 定義リモート プロシージャ コール）

次の表に、これら 2 つのプロトコルのトランスポートおよびエンコード メカニズムを示します。

| プロトコル | トランスポート | 符号化/復号化 |
|---|---|---|
| NETCONF | ssh | xml |
| gRPC | http/2 | json |

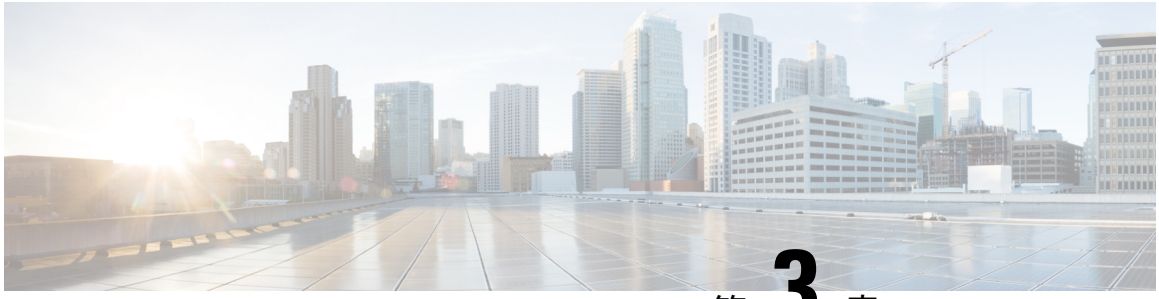
## NETCONF プロトコル

NETCONF は、ネットワーク デバイスの設定をインストール、操作、または削除するためのメカニズムです。コンフィギュレーション データとプロトコル メッセージに Extensible Markup Language（XML）ベースのデータ符号化を使用します。シンプルな NETCONF RPC（リモート

プロシージャ コール）ベースのメカニズムを使用してクライアントとサーバ間の通信を促進できます。NETCONF RPC の発行を開始してデータ モデルを使用してネットワーク機能を設定するには、NETCONF プロトコルを使用したデータ モデルによるネットワーク運用の定義 （11 ページ）を参照してください。

# gRPC プロトコル

gRPC はオープンソースの RPC フレームワークです。これはプロトコル バッファ（Protobuf）に基づいたオープンソースのバイナリ シリアル化プロトコルです。gRPC は、XML などの構造化されたデータをシリアル化するための柔軟で効率的な自動メカニズムですが、小型で使いやすくなっています。`.proto` ファイルでプロトコル バッファ メッセージ タイプを定義することで、構造を定義することができます。各プロトコル バッファ メッセージは、一連の名前と値のペアを含む情報の小型の論理レコードです。NETCONF RPC の発行を開始してデータ モデルを使用してネットワーク機能を設定するには、gRPC プロトコルを使用したデータ モデルによるネットワーク運用の定義 （21 ページ）を参照してください。



第 3 章

# NETCONF プロトコルを使用したデータ モデルによるネットワーク運用の定義

XR デバイスには、サポート対象のデータ モデルを定義する YANG ファイルが付属しています。NETCONF や gRPC などの管理プロトコルを使用して、デバイスに対してサポートされているモデルのリストをプログラムで検索し、モデル ファイルを取得することができます。

ネットワーク設定プロトコル（NETCONF）は、ネットワーク デバイスと通信する標準的なトランスポート プロトコルです。NETCONF は、設定データを編集し、ネットワーク デバイスから運用データを取得するためのメカニズムを提供します。設定データは、インターフェイス、ルーティング プロトコル、およびその他のネットワーク機能をプロビジョニングする方法を表します。運用データは、インターフェイスの統計情報、メモリ使用率、エラーなどを表します。

NETCONF は設定データとプロトコル メッセージに Extensible Markup Language（XML）ベースのデータ符号化を使用します。シンプルな RPC（リモート プロシージャ コール）ベースのメカニズムを使用してクライアントとサーバ間の通信を促進します。クライアントはネットワーク マネージャの一部として実行されているスクリプトやアプリケーションです。サーバは、ルータなどのネットワーク デバイスです。NETCONF はデバイスとの通信方法を定義しますが、クライアントとサーバ間で交換するデータを処理することはありません。

NETCONF を有効にするには、**ssh server capability netconf-xml** コマンドを使用してポート 22 の XML サブシステムにアクセスします。

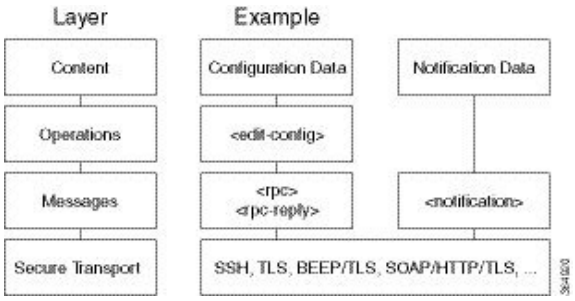
### NETCONF セッション

NETCONF セッションは、ネットワーク コンフィギュレーション アプリケーション（クライアント）とネットワーク デバイス（ルータ）の間の論理接続です。設定の属性は承認されたセッション時に変更でき、その影響はすべてのセッションに反映されます。NETCONF はコネクション型で、基盤となる転送に SSH を使用しています。NETCONF セッションは、機能が公開されている場合は「hello」メッセージで確立され、*close* メッセージまたは *kill* メッセージを使用して終了します。

### NETCONF レイヤ

NETCONF プロトコルは 4 つのレイヤに分割できます。

図 *2 : NETCONF* レイヤ




- **コンテンツ レイヤ**：設定および通知データが含まれます
- **動作レイヤ**：XML で符号化されたパラメータによる RPC メソッドとして起動される一連の基本プロトコル動作を定義します
- **メッセージ レイヤ**：RPC と通知を符号化するためのシンプルな、トランスポート非依存のフレーミング メカニズムを提供します
- **セキュア トランスポート レイヤ**：クライアントとサーバ間の通信パスを提供します

NETCONF の詳細については、RFC 6241 を参照してください。

ここでは、ルータのローカル時刻を設定する使用例を用いて、CLI と比較し、データ モデルが高速プログラム設定でどのように役立つかについて説明します。

- NETCONF の操作 （12 ページ）
- NETCONF セッションでのデータ モデルを使用したルータ クロックの設定 （16 ページ）

# NETCONF の操作

NETCONF は 1 つ以上のコンフィギュレーション データストアの存在を定義し、それらでの設定操作を可能にします。設定データストアは、初期のデフォルト状態から必要な動作状態にデバイスを移行するために必要な一連の完全なコンフィギュレーション データです。コンフィギュレーション データストアには状態データやエグゼクティブ コマンドは含まれません。

基本プロトコルには、次の NETCONF 操作が含まれています。

```
|   +--get-config
|   +--edit-Config
|      +--merge
|      +--replace
|      +--create
|      +--delete
|      +--remove
|      +--default-operations
|         +--merge
|         +--replace
|         +--none
|   +--get
|   +--lock
|   +--unLock
```

```
|   +--close-session
|   +--kill-session
```

これらの NETCONF 操作については、次の表で説明します。

| NETCONF動作 | 説明 | 例 |
|---|---|---|
| <get-config> | 指定した設定の全部または一部を名前付きのデータストアから取得します | フィルタ オプションを使用して実行中の設定から特定のインターフェイス設定の詳細を取得します<br><br>`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get-config><source><running/></source><filter><interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"\><interface-configuration><active>act</active><interface-name>TenGigE0/0/0/2</interface-name></interface-configuration></interface-configurations></filter></get-config></rpc>` |
| <get> | 実行中の設定およびデバイスの状態情報を取得します | すべての ACL 設定およびデバイスの状態情報を取得します<br><br>`Request:<get><filter><ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-oper"/></filter></get>` |

| NETCONF 動作 | 説明 | 例 |
|---|---|---|
| <edit-config> | 指定した設定の全部または一部を指定したターゲット設定にロードします | **Merge** 操作で ACL 設定を設定します<br><br>`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <edit-config> <target><candidate/></target> <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"> <ipv4-acl-and-prefix-list xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-acl-cfg" xc:operation=”merge”> <accesses> <access> <access-list-name>aclv4-1</access-list-name> <access-list-entries> <access-list-entry> <sequence-number>10</sequence-number> <remark>GUEST</remark> </access-list-entry> <access-list-entry> <sequence-number>20</sequence-number> <grant>permit</grant> <source-network> <source-address>172.0.0.0</source-address> <source-wild-card-bits>0.0.255.255</source-wild-card-bits> </source-network> </access-list-entry> </access-list-entries> </access> </accesses> </ipv4-acl-and-prefix-list> </config> </edit-config> </rpc>`<br><br>`Commit: <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <commit/> </rpc>` |
| <lock> | クライアントがデバイスの設定データストア システム全体をロックすることができます | 実行中の設定をロックします。<br>`Request: <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <lock> <target> <running/> </target> </lock> </rpc>`<br><br>`Response : <rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <ok/> </rpc-reply>` |

| NETCONF 動作 | 説明 | 例 |
|---|---|---|
| <Unlock> | 以前にロックされた設定をリリースします。<br><br>次のいずれかの条件が true の場合、<unlock> 操作は失敗します。<br>• 指定されたロックが現在アクティブではない。<br>• <unlock> 操作を発行するセッションが、ロックを取得したセッションと同じではない。 | 同一セッションの実行中の設定をロックおよびロック解除します。<br><br>Request:<br>rpc message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><unlock><br><target><br><running/><br></target><br></unlock><br></rpc><br><br>Response -<br><rpc-reply message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><ok/><br></rpc-reply> |
| <close-session> | セッションを終了します。サーバは、セッションに関連付けられているロックとリソースをリリースし、関連付けられた接続を終了します。 | NETCONF セッションを終了します。<br><br>Request :<br><rpc message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><close-session/><br></rpc><br><br>Response:<br><rpc-reply message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><ok/><br></rpc-reply> |
| <kill-session> | 現在処理中の操作を中止し、そのセッションに関連付けられているロックとリソースをリリースして、関連付けられた接続を終了します。 | ID が他のセッション ID の場合は、セッションを中止します。<br><br>Request:<br><rpc message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><kill-session><br><session-id>4</session-id><br></kill-session><br></rpc><br><br>Response:<br><rpc-reply message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><ok/><br></rpc-reply> |

### 設定を取得する NETCONF 操作

次に、CDP 機能に対する NETCONF <get-config> 要求の動作の例を示します。

クライアントはメッセージを送信し、ルータ上で実行している CDP の現在の設定を取得します。ルータは現在の CDP 設定で応答します。

| NETCONF 要求（クライアントからルータへ） | Netconf 応答（ルータからクライアントへ） |
|---|---|
| <rpc message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><br><get-config><br><source><running/></source><br><filter><br><cdp<br>xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"/><br></filter><br></get-config><br></rpc> | <?xml version="1.0"?><br><rpc-reply message-id="101"<br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><br><br><data><br><cdp<br>xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-cdp-cfg"><br><br><timer>10</timer><br><enable>true</enable><br><log-adjacency></log-adjacency><br><hold-time>200</hold-time><br><advertise-v1-only></advertise-v1-only><br></cdp><br>#22<br></data><br></rpc-reply> |

要求および応答メッセージ内の <rpc> 要素は、クライアントとルータ間で送信される NETCONF 要求を囲みます。<rpc> 要素内の `message-id` 属性は必須です。この属性は送信者によって選択された文字列で、整数をエンコードします。<rpc> 要素の受信者はこの文字列を復号化したり解釈したりしせず、<rpc-reply> メッセージ内で使用するために保存するだけです。送信者は `message-id` 値が正規化されていることを確認する必要があります。クライアントがサーバから情報を受信した場合、<rpc-reply> メッセージには同じ `message-id` が含まれています。

# NETCONFセッションでのデータモデルを使用したルータクロックの設定

NETCONFプロトコルは、ネットワークを設定するためにセキュアシェル（SSH）転送で使用される XML ベースのプロトコルです。クライアントアプリケーションはこのプロトコルを使用してルータの情報を要求し、ルータの設定を変更します。

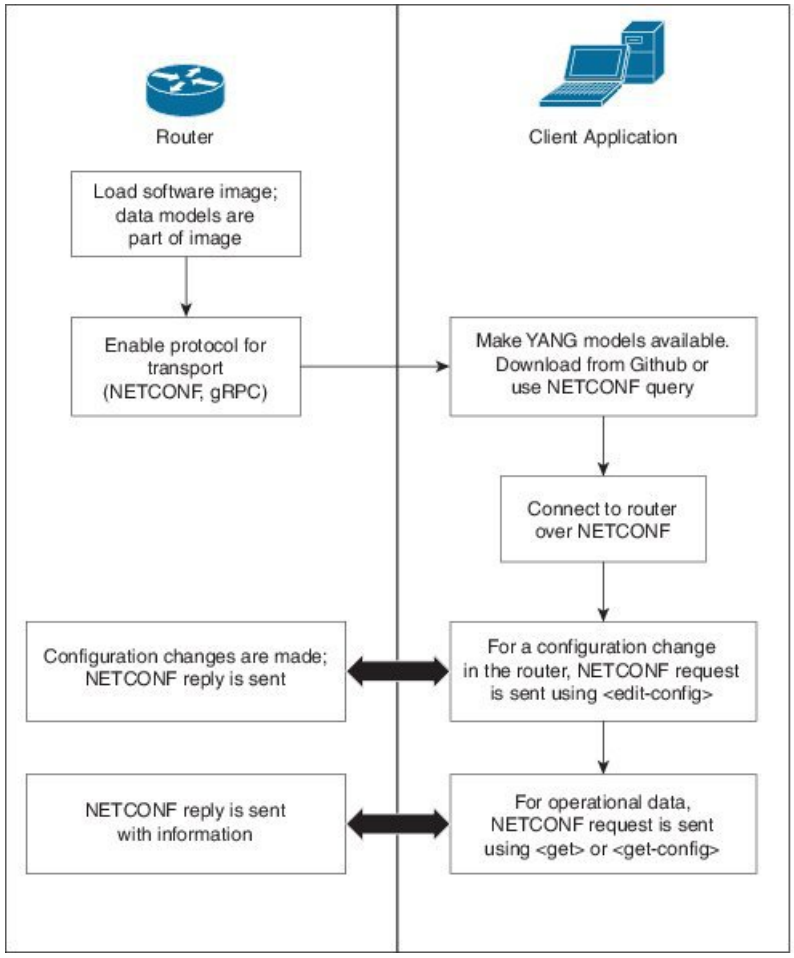
データ モデルを使用するプロセス：

- データ モデルを取得します。
- NETCONF 通信プロトコルを使用して、ルータとクライアント間の接続を確立します。
- データ モデルを使用してクライアントからルータの設定を管理します。

（注） ユーザが制御されていないアクセスを行うのを制限するために AAA 認証を設定します。AAA 認証が設定されていない場合、ユーザに割り当てられたグループに関連付けられたコマンドおよびデータ ルールはバイパスされます。IOS-XR ユーザは、ネットワーク設定プロトコル（NETCONF）、Google 定義のリモート プロシージャ コール（gRPC）または任意の YANG ベースのエージェントを介して、IOS-XR 設定への完全な読み取り/書き込みアクセス権を持つことができます。制御されていないアクセスを許可しないようにするには、いずれかの設定を行う前に AAA 認証を有効にします。

次の図は、データ モデルの使用に関連するタスクを示しています。

**図 *3:* データ モデルを使用するプロセス**



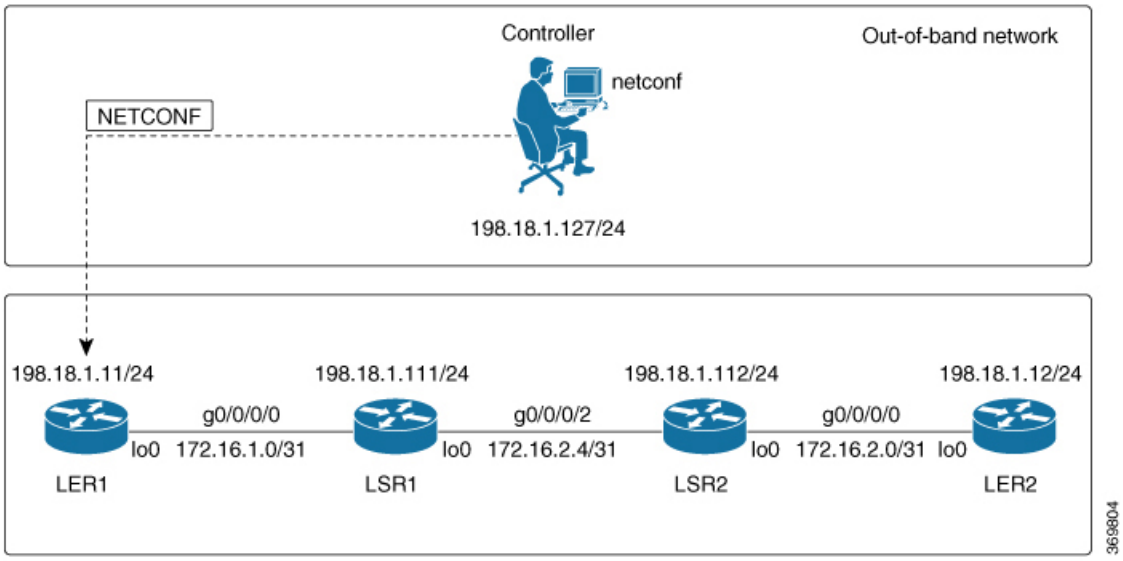

このセクションでは、ネイティブデータ モデルを使用してルータ クロックを設定し、NETCONF セッションを使用してクロック状態を確認します。

4 台のルータと 1 台のコントローラで構成されるネットワーク トポロジを考えてみます。ネットワークは、ラベル エッジ ルータ（ルータ）とラベル スイッチング ルータ（LSR）で構成されています。LER1 および LER2 の 2 つのルータはラベル エッジ ルータであり、2 つのルータ

LSR1 と LSR2 はラベル スイッチング ルータです。ホストは gRPC クライアントを使用するコントローラです。コントローラはアウトオブバンドネットワークを介してすべてのルータと通信します。LER1 を除くすべてのルータは、適切な IP アドレッシングとルーティングの動作によって事前に設定されています。ルータ間のインターフェイスには、`/31` アドレッシングを使用したポイントツーポイント設定があります。ループバック プレフィックスは`172.16.255.x/32` 形式を使用します。

次の図に、ネットワーク トポロジを示します。

**図 *4: gRPC* セッションのネットワーク トポロジ**




Cisco IOS XR ネイティブ モデルの `Cisco-IOS-XR-infra-clock-linux-cfg.yang` および `Cisco-IOX-XR-shellutil-oper` を使用すると、ルータ クロックをプログラムで設定できます。pyang などの YANG バリデータ ツールを使用して、データ モデルの構造を調べることができます。

### 始める前に

NETCONF モニタリング RPC を使用して、ルータ上の YANG モジュールのリストを取得します。詳細については、データ モデルへのアクセス （8 ページ）を参照してください。

# ルータ クロックの設定

### 手順

**ステップ 1** システムのローカル タイムゾーンのネイティブ設定モデルを確認します。

**例：**

```
controller:netconf$ pyang --format tree Cisco-IOS-XR-infra-infra-clock-linux-cfg.yang
```

```
module: Cisco-IOS-XR-infra-infra-clock-linux-cfg
    +--rw clock
        +--rw time-zone!
        +--rw time-zone-name string
        +--rw area-name string
```

**ステップ 2** システム時刻のネイティブ動作状態モデルを確認します。

**例**：

```
controller:netconf$ pyang --format tree Cisco-IOS-XR-shellutil-oper.yang
module: Cisco-IOS-XR-shellutil-oper
    +--ro system-time
        +--ro clock
        | +--ro year? uint16
        | +--ro month? uint8
        | +--ro day? uint8
        | +--ro hour? uint8
        | +--ro minute? uint8
        | +--ro second? uint8
        | +--ro millisecond? uint16
        | +--ro wday? uint16
        | +--ro time-zone? string
        | +--ro time-source? Time-source
        +--ro uptime
            +--ro host-name? string
            +--ro uptime? uint32
```

**ステップ 3** ルータ LER1 の現在の時刻を取得します。

**例**：

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ netconf get --filter xr-system-time-oper.xml
198.18.1.11:830
<?xml version="1.0" ?>
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>17</hour>
        <minute>30</minute>
        <second>37</second>
        <millisecond>690</millisecond>
        <wday>1</wday>
        <time-zone>UTC</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>851237</uptime>
    </uptime>
</system-time>
```

タイムゾーン UTC は、ローカル タイムゾーンが設定されていないことを示していることに注意してください。

**ステップ 4** 太平洋標準時（PST）を LER1 のローカル タイムゾーンとして設定します。

**例**：

```
controller:netconf$ more xr-system-time-oper.xml <system-time
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>
controller:netconf$ get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
  <system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2019</year>
        <month>8</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

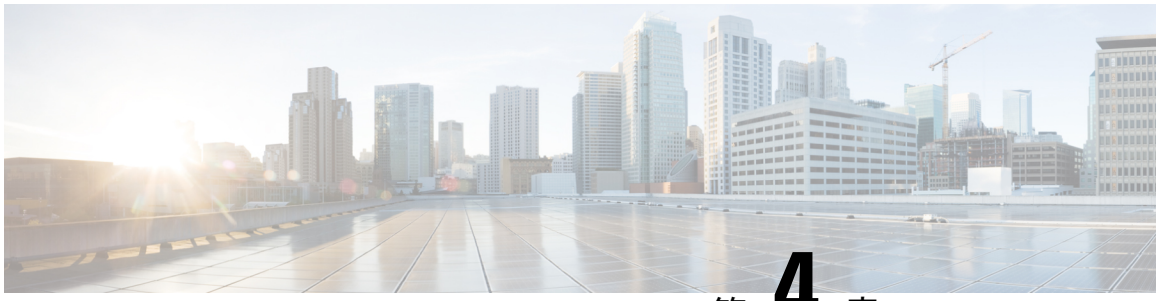
# ルータ クロックの表示

ルータ クロックが PST タイムゾーンに設定されていることを確認します。

```
controller:netconf$ more xr-system-time-oper.xml
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper"/>

controller:netconf$ netconf get --filter xr-system-time-oper.xml
<username>:<password>@198.18.1.11:830
<?xml version="1.0" ?>
<system-time xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-shellutil-oper">
    <clock>
        <year>2018</year>
        <month>12</month>
        <day>22</day>
        <hour>9</hour>
        <minute>52</minute>
        <second>10</second>
        <millisecond>134</millisecond>
        <wday>1</wday>
        <time-zone>PST</time-zone>
        <time-source>calendar</time-source>
    </clock>
    <uptime>
        <host-name>ler1</host-name>
        <uptime>852530</uptime>
    </uptime>
</system-time>
```

つまり、ローカル タイムゾーンが設定されていないルータ LER1 は、データ モデルを使用してプログラムで設定されます。



第 4 章

# gRPC プロトコルを使用したデータ モデルによるネットワーク運用の定義

XR デバイスには、サポート対象のデータ モデルを定義する YANG ファイルが付属しています。NETCONF や gRPC などの管理プロトコルを使用して、デバイスに対してサポートされているモデルのリストをプログラムで検索し、モデル ファイルを取得することができます。

gRPC はオープンソースの RPC フレームワークです。これはプロトコル バッファ（Protobuf）に基づいたオープンソースのバイナリ シリアル化プロトコルです。gRPC は、XML などの構造化されたデータをシリアル化するための柔軟で効率的な自動メカニズムですが、小型で使いやすくなっています。`.proto` ファイルでプロトコル バッファ メッセージ タイプを使用し、構造を定義することができます。各プロトコル バッファ メッセージは、一連の名前と値のペアを含む情報の小型の論理レコードです。

gRPC は要求および応答をバイナリでエンコードします。gRPC は、Protobuf とともに他のコンテンツ タイプに拡張可能です。gRPC の Protobuf バイナリ データ オブジェクトは HTTP/2 を介して転送されます。

gRPC はクライアントとサーバ間の分散型のアプリケーションやサービスをサポートします。gRPC はサーバとクライアント間の設定データと運用データを交換するためにデバイス管理サービスを構築するインフラストラクチャを提供します。そのデータの構造は YANG モデルによって定義されます。

Cisco gRPC IDL はプロトコル バッファ インターフェイス定義言語（IDL）を使用して、サービス方式を定義し、パラメータを定義し、プロトコル バッファ メッセージ タイプとしてタイプを返します。gRPC 要求はエンコードされ、JSON を使用してルータに送信されます。クライアントは、IDL で定義される RPC コールを起動してルータをプログラミングできます。

次に、gRPC 設定の proto ファイルの構文例を示します。

```
syntax = "proto3";

package IOSXRExtensibleManagabilityService;

service gRPCConfigOper {

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

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

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

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

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

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

    rpc CommitReplace(CommitReplaceArgs) returns(CommitReplaceReply) {};
}
message ConfigGetArgs {
     int64 ReqId = 1;
     string yangpathjson = 2;
}

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

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

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

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

}

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

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

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

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

message CommitReplaceReply {
     int64 ResReqId = 1;
     string errors = 2;
}
```

gRPCExec 設定の例：

```
service gRPCExec {
    rpc ShowCmdTextOutput(ShowCmdArgs) returns(stream ShowCmdTextReply) {};
    rpc ShowCmdJSONOutput(ShowCmdArgs) returns(stream ShowCmdJSONReply) {};
    rpc ActionJSON(ActionJSONArgs) returns(stream ActionJSONReply) {};
}

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

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

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

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

OpenConfiggRPC 設定の例：

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

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

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

ここでは、ルータのインターフェイスを設定する使用例を用いて、CLI と比較し、データ モデルが高速プログラム設定および標準ベースの設定でどのように役立つかについて説明します。


- gRPC の操作 （24 ページ）
- gRPC ネットワーク管理インターフェイス （25 ページ）
- gRPC ネットワーク操作インターフェイス （26 ページ）
- gRPC セッションでのデータ モデルを使用したインターフェイスの設定 （28 ページ）


# gRPC の操作

次の gRPC 操作を発行できます。

| gRPC の操作 | 説明 |
|---|---|
| GetConfig | 設定を取得します |
| GetModels | ルータで、サポートされている Yang モデルを取得します |
| MergeConfig | 既存の設定に追加します |
| DeleteConfig | 設定を削除します |
| ReplaceConfig | 既存の設定の一部を変更します |
| CommitReplace | 既存の設定を指定された新しい設定ファイルに置き換えます |
| GetOper | JSON を使用して運用データを取得します |
| CliConfig | CLI 設定を起動します |
| ShowCmdTextOutput | show コマンドの出力を表示します |
| ShowCmdJSONOutput | show コマンドの JSON 出力を表示します |

### 設定を取得する gRPC 操作

次に、CDP 機能に対する gRPC GetConfig 要求の動作例を示します。

クライアントはメッセージを送信し、ルータ上で実行している CDP の現在の設定を取得します。ルータは現在の CDP 設定で応答します。

| gRPC 要求（クライアントからルータへ） | gRPC 応答（ルータからクライアントへ） |
|---|---|
| rpc GetConfig<br>{<br>"Cisco-IOS-XR-cdp-cfg:cdp": [<br>"cdp": "running-configuration"<br>]<br>} | {<br>"Cisco-IOS-XR-cdp-cfg:cdp": {<br>"timer": 50,<br>"enable": true,<br>"log-adjacency": [<br>null<br>],<br>"hold-time": 180,<br>"advertise-v1-only": [<br>null<br>]<br>}<br>} |

# gRPC ネットワーク管理インターフェイス

gRPC ネットワーク管理インターフェイス（gNMI）は、ネットワーク デバイスの設定を変更、インストール、または削除するために使用される gRPC ベースのネットワーク管理プロトコルです。また、運用データの表示、ターゲットデバイスからデータ収集システムへのテレメトリストリームの制御および生成にも使用されます。単一のプロトコルを使用して、ネットワークデバイスから設定およびストリーム テレメトリ データを管理します。

gNNMI は gRPC をトランスポートプロトコルとして使用し、設定は gRPC の設定と同じです。次の gNMI RPC がサポートされています。

| gNMI RPC | gNMI RPC 要求 | 説明 |
|---|---|---|
| 機能 | | サポート対象のデータ モデルなどの機能情報を交換するためのネットワーク デバイス（サーバ）とクライアント間の初期ハンドシェイク |
| セット | `message SetRequest {}` | クライアントからネットワーク デバイスのモデルに関連付けられているデータを変更します |
| Get | `message GetRequest {}` | ネットワーク デバイスからデータを取得します |
| Subscribe | `message SubscribeRequest {}` | サーバでデータ サブスクリプションを制御します |

gNMI の詳細については、Github を参照してください。

# gRPC ネットワーク操作インターフェイス

gRPC ネットワーク オペレーション インターフェイス（gNOI）は、ネットワーク デバイス上で操作コマンドを実行するための gRPC ベースのマイクロサービス セットを定義します。拡張可能な管理サービス（EMS）gNOI は gNOI の Cisco IOS XR 実装です。

gNOI は gRPC をトランスポート プロトコルとして使用し、設定は gRPC の設定と同じです。次の gNOI がサポートされています。

**Get RPC**

ターゲットからファイルの内容をストリーミングします。

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

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

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

**Remove RPC**

ターゲットから指定されたファイルを削除します。

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

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

**Reboot RPC**

要求されたターゲットをリロードします。

```
RPC to 10.105.57.106:57900
RPC start time: 21:12:49.811536
---------------------Reboot Request----------------------
RPC start time: 21:12:49.811561
method: COLD
message: "Test Reboot"
subcomponents {
origin: "openconfig-platform"
elem {
name: "components"
}
elem {
name: "component"
```

```
key {
key: "name"
value: "0/RP0"
}
}
elem {
name: "state"
}
elem {
name: "location"
}
}
---------------------Reboot Request---------------------
RPC end time: 21:12:50.023604
```

### Set Package RPC

ターゲットにソフトウェア パッケージを配置します。

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

### Reboot Status RPC

ターゲットのリブートのステータスを返します。

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

RPC end time: 22:27:34.319618

----------------------Reboot Status Response----------------------
Active : False
```

```
Wait : 0
When : 0
Reason : Test Reboot
Count : 0
```

gNOI RPC 要求を送信するには、各 RPC に gNOI クライアント インターフェイスを実装するクライアントが必要です。

gRPC サービス定義内のすべてのメッセージは、プロトコル バッファ（proto ファイル）として定義されます。gNOI OpenConfig proto ファイルは Github にあります。

# gRPC セッションでのデータ モデルを使用したインターフェイスの設定

Google 定義されたリモート プロシージャ コール（gRPC プロトコル）は、オープンソースの RPC フレームワークです。gRPC は IPv4 および IPv6 アドレス ファミリをサポートしています。クライアントアプリケーションはこのプロトコルを使用してルータの情報を要求し、ルータの設定を変更します。

データ モデルを使用するプロセス：

- データ モデルを取得します。
- gRPC 通信プロトコルを使用して、ルータとクライアント間の接続を確立します。
- データ モデルを使用してクライアントからルータの設定を管理します。

（注） ユーザが制御されていないアクセスを行うのを制限するために AAA 認証を設定します。AAA 認証が設定されていない場合、ユーザに割り当てられたグループに関連付けられたコマンドおよびデータ ルールはバイパスされます。IOS-XR ユーザは、ネットワーク設定プロトコル（NETCONF）、Google 定義のリモート プロシージャ コール（gRPC）または任意の YANG ベースのエージェントを介して、IOS-XR 設定への完全な読み取り/書き込みアクセス権を持つことができます。制御されていないアクセスを許可しないようにするには、いずれかの設定を行う前に AAA 認証を有効にします。

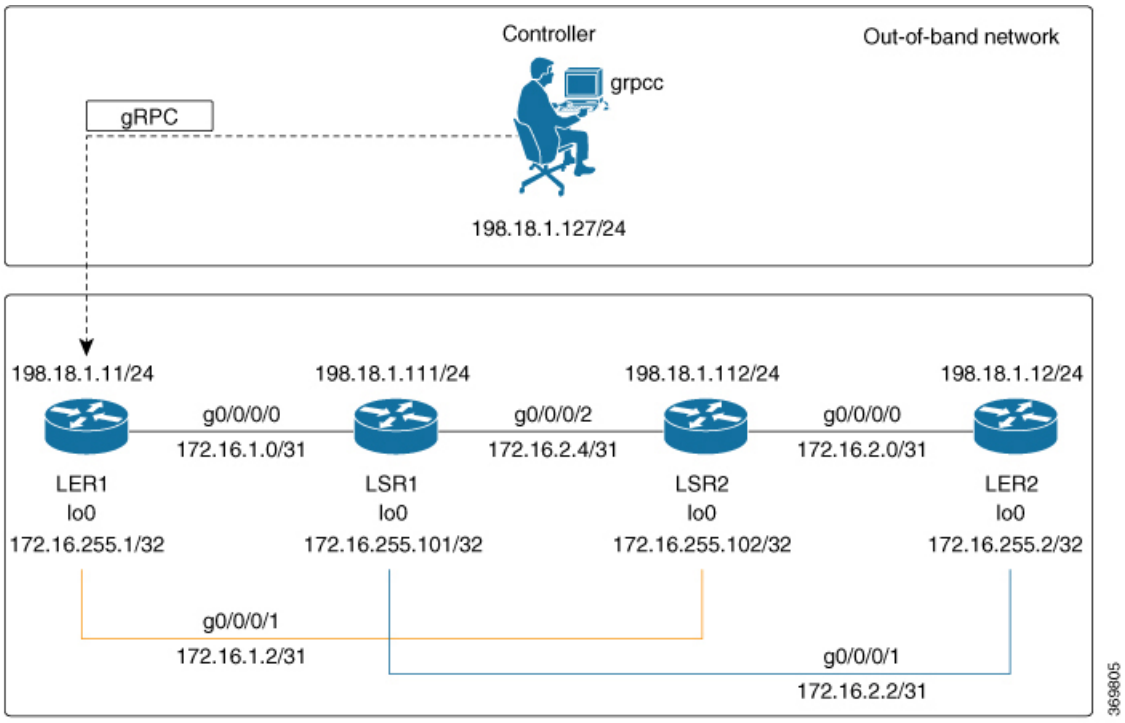
このセクションでは、ネイティブデータ モデルを使用し、gRPC セッションを使用してルータ上のループバック インターフェイスとイーサネット インターフェイスを設定します。

4 台のルータと 1 台のコントローラで構成されるネットワーク トポロジを考えてみます。ネットワークは、ラベル エッジ ルータ（ルータ）とラベル スイッチング ルータ（LSR）で構成されています。LER1 および LER2 の 2 つのルータはラベル エッジ ルータであり、2 つのルータ LSR1 と LSR2 はラベル スイッチング ルータです。ホストは gRPC クライアントを使用するコントローラです。コントローラはアウトオブバンドネットワークを介してすべてのルータと通信します。LER1 を除くすべてのルータは、適切な IP アドレッシングとルーティングの動作によって事前に設定されています。ルータ間のインターフェイスには、/31 アドレッシングを使

用したポイントツーポイント設定があります。ループバック プレフィックスは172.16.255.x/32形式を使用します。

次の図に、ネットワーク トポロジを示します。

**図 5 : gRPC セッションのネットワーク トポロジ**




Cisco IOS XR ネイティブ モデル Cisco-IOS-XR-ifmgr-cfg.yang を使用して、ルータ LER1 をプログラムで設定します。

## 始める前に

- NETCONF モニタリング RPC を使用して、ルータ上の YANG モジュールのリストを取得します。詳細については、データ モデルへのアクセス （8 ページ）を参照してください。
- トランスポート層セキュリティ（TLS）の設定gRPC プロトコルを有効にすると、TLS を使用しないデフォルトの HTTP/2 トランスポートが使用されます。gRPC では、すべての gRPC 要求に対して AAA 認証および認可が義務付けられています。TLS が設定されていない場合、認証クレデンシャルはネットワーク上で暗号化されずに転送されます。TLS を有効にすると、クレデンシャルがセキュアで暗号化されることが保証されます。非 TLS モードは、セキュアな内部ネットワークでのみ使用できます。

# gRPC プロトコルの有効化

ネットワークデバイスを設定して運用データを表示するには、サーバでgRPCプロトコルを有効にする必要があります。次の例では、LER1（サーバ）の gRPC プロトコルを有効にしています。

**手順**

**ステップ 1** HTTP/2 接続で gRPC を有効にします。

**例**：

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

ポート番号の範囲は 57344 ～ 57999 です。ポート番号が使用できない場合は、エラーが表示されます。

**ステップ 2** セッション パラメータを設定します。

**例**：

```
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 }
```

値は次のとおりです。

- `address-family` : アドレス ファミリ識別子タイプを設定します
- `dscp` : 送信された gRPC で QoS マーキング DSCP を設定します
- `max-request-per-user` : ユーザあたりの同時要求の最大数を設定します
- `max-request-total` : 合計同時要求の最大数を設定します
- `max-streams` : 同時 gRPC 要求の最大数を設定します。サブスクリプションの上限は 128 要求です。デフォルトは 32 要求です
- `max-streams-per-user` : ユーザあたりの同時 gRPC 要求の最大数を設定します。サブスクリプションの上限は 128 要求です。デフォルトは 32 要求です
- `no-tls` : トランスポート レイヤ セキュリティ（TLS）を無効化します。TLS はデフォルトで有効になっています。
- `service-layer` : gRPC サービス レイヤの設定を有効にします
- `tls-cipher` : gRPC TLS 暗号スイートを有効にします
- `tls-mutual` : 相互認証を設定します
- `tls-trustpoint` : トラストポイントを設定します

- `vrf` : サーバ VRF を有効にします

gRPC を有効にした後、YANG データ モデルを使用してネットワーク設定を管理します。

# インターフェイスの設定

次の例では、Cisco IOS XR ネイティブ モデル `Cisco-IOS-XR-ifmgr-cfg.yang` を使用してインターフェイスを設定しています。インターフェイスの設定中は、さまざまな gRPC の操作を理解することになります。操作の完全なリストについては、gRPC の操作 （24 ページ）を参照してください。次の例では、設定と `merge-config` RPC のマージ、`get-oper` RPC を使用した運用統計情報の取得、`delete-config` RPC を使用した設定の削除を示します。pyang などの YANG バリデータ ツールを使用して、データ モデルの構造を調べることができます。

LER1 は gRPC サーバであり、コマンド ライン ユーティリティ `grpcc` はコントローラ上のクライアントとして使用されます。このユーティリティは YANG をサポートしていないため、データ モデルは検証されません。サーバ LER1 はデータ モードを検証します。

**手順**

**ステップ 1** インターフェイスとその IPv4 拡張の XR 設定モデルを確認します。

**例：**

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

**ステップ 2** LER1 で loopback0 インターフェイスを設定します。

a) ループバック インターフェイス `Loopback0` を設定し、IP アドレスを割り当てます。

例：

```
controller:grpc$ more xr-interfaces-lo0-cfg.json
{
 "Cisco-IOS-XR-ifmgr-cfg:interface-configurations":
  { "interface-configuration": [
    {
      "active": "act",
      "interface-name": "Loopback0",
      "description": "LOCAL TERMINATION ADDRESS",
      "interface-virtual": [
        null
       ],
       "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
        "addresses": {
            "primary": {
              "address": "172.16.255.1",
              "netmask": "255.255.255.255"
       }
      }
     }
    }
   ]
 }
}
```

b) 設定をマージします。

例：

```
controller:grpc$ grpcc -username admin -password admin -oper merge-config
-server_addr 198.18.1.11:57400 -json_in_file xr-interfaces-lo0-cfg.json
emsMergeConfig: Sending ReqId 1
emsMergeConfig: Received ReqId 1, Response '
'
```

**ステップ 3** LER1 でイーサネット インターフェイスを設定します。

a) LER1 でイーサネット `GigabitEthernet0/0/0/0` を設定します。

例：

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

b) 設定をマージします。

**例**：

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

**ステップ 4** LER1 でイーサネット インターフェイス `GigabitEthernet 0/0/0/0` を有効にして、インターフェイスを起動します。このためには、インターフェイスで `shutdown` 設定を削除します。

**例**：

```
controller:grpc$ grpcc -username admin -password admin -oper delete-config
-server_addr 198.18.1.11:57400 -yang_path "$(< xr-interfaces-gi0-shutdown-cfg.json )"
emsDeleteConfig: Sending ReqId 1, yangJson {
 "Cisco-IOS-XR-ifmgr-cfg:interface-configurations": {
   "interface-configuration": [
   {
    "active": "act",
    "interface-name": "GigabitEthernet0/0/0/0",
    "shutdown": [
      null
    ]
   }
  ]
 }
}
emsDeleteConfig: Received ReqId 1, Response ''
```

# インターフェイスの状態の確認

ルータ LER1 上のループバック インターフェイスとイーサネット インターフェイスが動作していることを確認します。

```
controller:grpc$ grpcc -username admin -password admin -oper get-oper
-server_addr 198.18.1.11:57400 -oper_yang_path "$(< xr-interfaces-briefs-oper-filter.json
 )"
emsGetOper: Sending ReqId 1, yangPath {
  "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces": {
    "interface-briefs": [
      null
     ]
  }
}
{ "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces": {
  "interface-briefs": {
    "interface-brief": [
    {
      "interface-name": "GigabitEthernet0/0/0/0",
      "interface": "GigabitEthernet0/0/0/0",
      "type": "IFT_GETHERNET",
      "state": "im-state-up",
      "actual-state": "im-state-up",
```

```
    "line-state": "im-state-up",
    "actual-line-state": "im-state-up",
    "encapsulation": "ether",
    "encapsulation-type-string": "ARPA",
    "mtu": 1514,
    "sub-interface-mtu-overhead": 0,
    "l2-transport": false,
    "bandwidth": 1000000
   },
   {
    "interface-name": "GigabitEthernet0/0/0/1",
    "interface": "GigabitEthernet0/0/0/1",
    "type": "IFT_GETHERNET",
    "state": "im-state-up",
    "actual-state": "im-state-up",
    "line-state": "im-state-up",
    "actual-line-state": "im-state-up",
    "encapsulation": "ether",
    "encapsulation-type-string": "ARPA",
    "mtu": 1514,
    "sub-interface-mtu-overhead": 0,
    "l2-transport": false,
    "bandwidth": 1000000
   },
   {
    "interface-name": "Loopback0",
    "interface": "Loopback0",
    "type": "IFT_LOOPBACK",
    "state": "im-state-up",
    "actual-state": "im-state-up",
    "line-state": "im-state-up",
    "actual-line-state": "im-state-up",
    "encapsulation": "loopback",
    "encapsulation-type-string": "Loopback",
    "mtu": 1500,
    "sub-interface-mtu-overhead": 0,
    "l2-transport": false,
    "bandwidth": 0
},
{
   "interface-name": "MgmtEth0/RP0/CPU0/0",
   "interface": "MgmtEth0/RP0/CPU0/0",
   "type": "IFT_ETHERNET",
   "state": "im-state-up",
   "actual-state": "im-state-up",
   "line-state": "im-state-up",
   "actual-line-state": "im-state-up",
   "encapsulation": "ether",
   "encapsulation-type-string": "ARPA",
   "mtu": 1514,
   "sub-interface-mtu-overhead": 0,
   "l2-transport": false,
   "bandwidth": 1000000
},
{
   "interface-name": "Null0",
   "interface": "Null0",
   "type": "IFT_NULL",
   "state": "im-state-up",
   "actual-state": "im-state-up",
   "line-state": "im-state-up",
   "actual-line-state": "im-state-up",
   "encapsulation": "null",
   "encapsulation-type-string": "Null",
```

```
      "mtu": 1500,
      "sub-interface-mtu-overhead": 0,
      "l2-transport": false,
      "bandwidth": 0
    }
   ]
  }
 }
}
emsGetOper: ReqId 1, byteRecv: 2325
```

つまり、最小限に設定したルータ LER1 は、イーサネット インターフェイスを備えたデータ モデルを使用してプログラムで設定され、ループバック アドレスが割り当てられるようになりました。これらのインターフェイスは両方とも動作可能で、ネットワーク プロビジョニング操作の準備が完了しています。