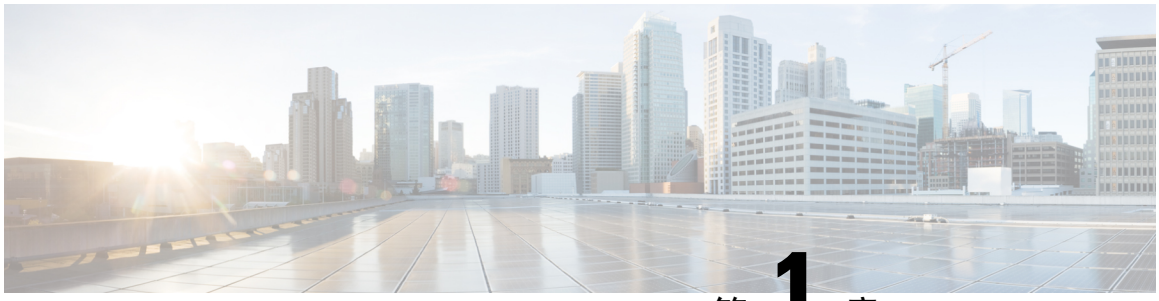
Cisco Nexus 9000 Series NX-OS Programmability Guide、リリース 10.1(x)

First Published: 2021-02-16

Last Modified: 2023-09-11

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS



第 1 章

新機能および変更された機能に関する情報

この章では、*Cisco Nexus 9000* シリーズ *NX-OS* プログラマビリティ ガイド、*10.1(x)* のこのリリースに記載されている、新機能および変更された機能に関するリリース固有の情報について説明します。

新機能および変更された機能に関する情報

次の表では、*Cisco Nexus 9000 Series NX-OS Programmability Guide, Release 10.1* に記載されている新機能および変更機能を要約しており、対応するセクションを示しています。

表 *1:* 新機能および変更された機能

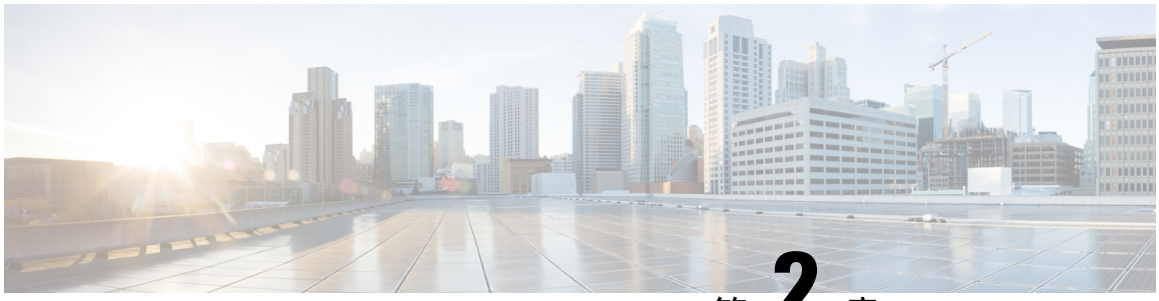
特長	説明	変更が行われたリリース	参照先
テレメトリ	宛先ホスト名のサポートを追加ノードIDのサポートを追加gRPC非同期モード機能を追加 証明書トラストポイント証明書の trustpoint キーワードを追加テレメトリ転送セッションのコマンドを追加エフェメラルイベントをサポートするために、新しいセンサーパスクエリ条件を追加	10.1(1)	モデル駆動型テレメトリ （367 ページ）

新機能および変更された機能に関する情報

特長	説明	変更が行われたリリース	参照先
gRPC エージェント	NX-OS は、10.1(x) 以降の gRPC サービスをサポートしていません。したがって、gRPCエージェントの章は、プログラマビリティ ガイドから削除されます。これまでの gRPC インターフェイスは、NX-OS 9.3(1) から NX-OS とともに出荷されている gNMI インターフェイスに代わられました。	10.1(1)	--
gNMI クライアント ベース証明書	gNMI クライアント証明書認証のサポートが強化された	10.1(1)	gNMI-gRPC ネットワーク管理インターフェイス （307 ページ）
gRPC ネットワーク操作インターフェイス（gNOI）	ネットワークデバイスで運用コマンドを実行するための gNOI のサポートが追加された	10.1(1)	gNOI-gRPC ネットワーク操作インターフェイス （349 ページ）
Guest Shell 3.0	CentOS 8 および Python 3.6 に基づく Guest Shell 3.0 のサポートが追加されたスイッチのDocker のバージョンをアップグレード	10.1(1)	ゲスト シェル （43 ページ）
OpenConfig モデルの追加	インターフェイス参照レベルの OpenConfig ACL の状態コンテナを追加 ドメイン名、ログインバナー、および motd バナーモデルのシステム構成コンテナを追加	10.1(1)	OpenConfig YANG （451 ページ）

特長	説明	変更が行われたリリース	参照先
Linux カーネルのアップグレード	Cisco NX-OS リリース 10.0(1) ソフトウェアは、Yocto 2.6 に基づいています。Yocto 2.6 をダウンロードし、構築する新しいソフトウェアをダウンロードし、ソフトウェアを構築し、ソフトウェアをスイッチにインストールするという方法によって、より多くのアプリケーションをインストールできます。	10.1(1)	bash （13 ページ） ゲスト シェル （43 ページ） サードパーティ製アプリケーション（119ページ） Nexus アプリケーション開発：Yocto （143 ページ） Nexus アプリケーション開発：SDK（147ページ） Cisco NX-OS での Docker の使用 （163 ページ） コンポーネント管理 （361 ページ） モデル駆動型テレメトリ （367 ページ）

新機能および変更された機能に関する情報



第 2 章

概要

プログラマビリティの概要

Cisco Nexus 9000 シリーズ スイッチ上で動作する Cisco NX-OS ソフトウェアには、次のような特徴があります。

- **耐障害性**

  クリティカルなビジネスクラスの可用性を確保します。

- **モジュラ型**

  ビジネス ニーズに対応する拡張機能があります。

- **高度なプログラム性**

  アプリケーション プログラミング インターフェイス（API）を介した迅速な自動化とオーケストレーションを可能にします。

- **セキュア**

  データと運用を保護し維持します。

- **柔軟性**

  新しいテクノロジーを統合して有効にします。

- **優れた拡張性**

  ビジネスと要件に対応して拡大できます。

- **使いやすさ**

必要な学習量が少なく、展開がシンプルで、管理が容易です。

Cisco NX-OS オペレーティング システムでは、デバイスはユニファイド ファブリック モードで機能し、プログラムによる自動化機能を備えたネットワーク接続を提供します。

Cisco NX-OS には、オープン ソース ソフトウェア（OSS）と商用テクノロジーが含まれており、これらは自動化、オーケストレーション、プログラマビリティ、モニタリング、コンプライアンスをサポートします。

オープンな NX-OSの詳細については、https://developer.cisco.com/site/nx-os/を参照してください。

サポートされるプラットフォーム

Cisco NX-OS リリース 7.0(3)I7(1) 以降、「Nexus スイッチ プラットフォーム サポート マトリクス」を使用して、選択した機能をサポートするさまざまな Cisco Nexus 9000 および 3000 スイッチのリリース元である Cisco NX-OS を知ることができます。

標準的なネットワーク管理機能

- SNMP（V1、V2、V3）
- Syslog
- RMON
- NETCONF
- CLI および CLI スクリプト

高度な自動化機能

デバイスでの機能が強化された Cisco NX-OS は、自動化をサポートします。プラットフォームには、電源投入時自動プロビジョニング（POAP）のサポートが含まれています。

デバイスでの機能が強化された Cisco NX-OS は、自動化をサポートします。プラットフォームには、自動化をサポートする機能が含まれています。

プログラマビリティのサポート

スイッチ上の Cisco NX-OS ソフトウェアは、プログラマビリティを支援する複数の機能をサポートしています。

NX-API のサポート

Cisco NX-API を使用すると、HTTP ベースのプログラムによってスイッチにアクセスすることが可能になります。このサポートは、オープンソースの Web サーバーである NX-API によって提供されています。NX-API では、Web ベース API を通じて Cisco NX-OS CLI のすべての構成機能および管理機能を提供しています。デバイスは、XML または JSON フォーマットで API 呼び出しの出力を公開するように設定できます。この API により、スイッチでの迅速な開発が可能になります。

Python スクリプティング

Cisco NX-OS は、Python v2.7.5 を、インタラクティブ モードと非インタラクティブ（スクリプト）モードの両方でサポートしています。

Cisco NX-OS リリース 9.3(5) 以降 、Python 3 もサポートされています。

デバイスの Python スクリプト機能は、さまざまなタスクを実行するためのスイッチの CLI と、Power On Auto Provisioning（POAP）または Embedded Event Manager（EEM）アクションへのプログラムによるアクセスを提供します。Cisco NX-OS CLI を呼び出す Python コールへの応答は、テキストまたは JSON 出力を返します。

Python インタープリタは Cisco NX-OS ソフトウェアに含まれています。

Tcl スクリプティング

Cisco Nexus 9000 シリーズ スイッチは、Tcl（ツール コマンド言語）をサポートします。Tcl は、スイッチで CLI コマンドを柔軟に使用できるようにするスクリプト言語です。Tcl を使用して **show** コマンドの出力の特定の値を抽出したり、スイッチを設定したり、Cisco NX-OS コマンドをループで実行したり、スクリプトで EEM ポリシーを定義したりすることができます。

Broadcom シェル

Cisco Nexus 9000 シリーズ スイッチの前面パネルおよびファブリック モジュールのライン カードには Broadcom のネットワーク転送エンジン（NFE）が搭載されています。これらの NFE から Broadcom コマンドライン シェル（bcm-shell）にアクセスすることができます。

bash

Cisco Nexus スイッチは、Bourne-Again Shell（Bash）への直接アクセスをサポートします。Bash では、デバイス上の基盤となる Linux システムにアクセスし、システムを管理できます。

bash シェルへのアクセスと Linux コンテナのサポート

Cisco Nexus スイッチは、Linux シェルへの直接アクセスと Linux コンテナをサポートしています。Linux シェルへのアクセスでは、スイッチ上の基盤となっている Linux システムにアクセスして、

基盤システムを管理できます。また、Linux コンテナを使用して、独自のソフトウェアを安全にインストールし、Cisco Nexus スイッチの機能を強化することもできます。たとえば、Cobbler などのベアメタル プロビジョニング ツールを Cisco Nexus スイッチにインストールして、トップオブラック スイッチからベアメタル サーバーを自動的にプロビジョニングすることができます。

ゲスト シェル

Cisco Nexus 9000 シリーズ スイッチは、ホスト Cisco Nexus 9000 NX-OS ソフトウェアから分離されたホスト システム上の Linux 実行スペースへの Bash アクセスを提供するゲスト シェルをサポートします。ゲスト シェルを使用すると、ホスト システム ソフトウェアに影響を与えることなく、必要に応じてソフトウェア パッケージを追加し、ライブラリを更新できます。

コンテナ トラッカーのサポート

Cisco NX-OS は、特定のスイッチ ポートの背後にあるコンテナの機能を理解するため、Kubernetes API サーバーと通信するように構成されます。

次のコマンドにより、Kubernetes API サーバーと通信します。

- **show containers kubernetes** コマンドは、HTTP 経由の API コールを使用して *kube-apiserver* からデータを取得します。
- **kubernetes watch** *resource* コマンドは、デーモンを使用して要求されたリソースにサブスクライブし、*kube-apiserver* からのストリーミング データを処理します。
- **watch** コマンドで割り当てられた **action** は、事前定義されたトリガーにより実行されます。（たとえば、ポッドの追加または削除）。

Perl モジュール

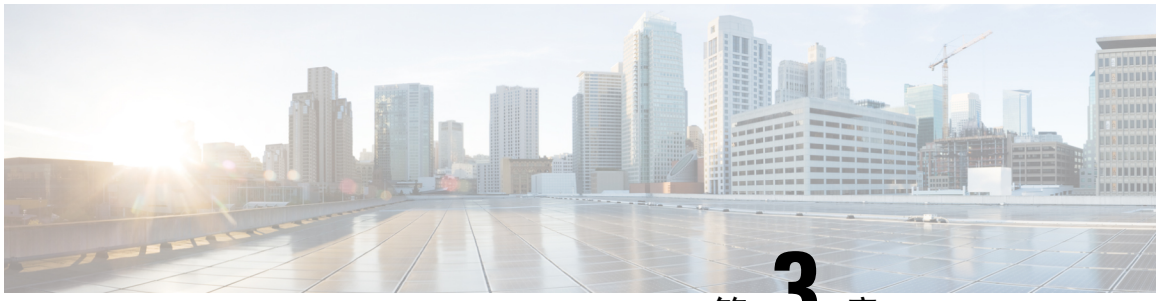
より多くのアプリケーションをサポートするために、次の Perl モジュールが追加されました。

- bytes.pm
- feature.pm
- hostname.pl
- lib.pm
- overload.pm
- Carp.pm
- Class/Struct.pm
- Data/Dumper.pm
- DynaLoader.pm
- Exporter/Heavy.pm

- FileHandle.pm
- File/Basename.pm
- File/Glob.pm
- File/Spec.pm
- File/Spec/Unix.pm
- File/stat.pm
- Getopt/Std.pm
- IO.pm
- IO/File.pm
- IO/Handle.pm
- IO/Seekable.pm
- IO/Select.pm
- List/Util.pm
- MIME/Base64.pm
- SelectSaver.pm
- Socket.pm
- Symbol.pm
- Sys/Hostname.pm
- Time/HiRes.pm
- auto/Data/Dumper/Dumper.so
- auto/File/Glob/Glob.so
- auto/IO/IO.so
- auto/List/Util/Util.so
- auto/MIME/Base64/Base64.so
- auto/Socket/Socket.so
- auto/Sys/Hostname/Hostname.so
- auto/Time/HiRes/HiRes.so

第 I 部

シェルとスクリプト化



第 3 章

bash

Bash について

Cisco NX-OS CLI に加えて、Cisco Nexus シリーズ スイッチは Bourne-Again SHell（Bash）へのアクセスをサポートします。Bashは、ユーザーが入力したコマンドまたはシェルスクリプトから読み取られたコマンドを解釈します。Bash を使用すると、デバイス上の基盤となる Linux システムにアクセスしてシステムを管理できます。

注意事項と制約事項

Bash シェルには、次の注意事項と制約事項があります。

- インターフェイスのリンクローカル アドレスを定義すると、Netstack はカーネルのネット デバイスに /64 プレフィックスをインストールします。

  新しいリンクローカルアドレスがカーネルで構成されると、カーネルはカーネル ルーティング テーブルに /64 ルートをインストールします。

ピア ボックスのインターフェイスに、同じ /64 サブネットに属するリンクローカル アドレスが設定されていない場合、bash プロンプトからの **ping** は成功しません。Cisco NX-OS の**ping** は正常に動作します。

- `/isan` フォルダ内のバイナリは、**run bash** コマンドで入るシェルの環境とは異なるようにセットアップされた環境で実行するためのものです。Bash シェルからこれらのバイナリを使用しないことをお勧めします。その環境内での動作は予測できないからです。
- Cisco Python モジュールをインポートする場合は、Bash シェルから Python を使用しないでください。代わりに NX-OS VSH 内で、より新しい Python を使用します。
- 一部のプロセスおよび**show** コマンドでは、大量の出力が発生する可能性があります。スクリプトを実行していて、実行時間の長い出力を終了する必要がある場合は、Ctrl+C（Ctrl+Z ではなく）を使用してコマンド出力を終了します。Ctrl+Z を使用すると、このキー コマンドによってSIGCONT（信号継続）メッセージが生成され、スクリプトが停止する可能性があります。SIGCONT メッセージによって停止されたスクリプトは、動作を再開するためにユーザーの介入が必要です。
- **show tech support** コマンドが実行中であり、強制終了する必要がある場合は、**clear tech-support lock** コマンドを使用しないでください。Ctrl+C を使用します。

  clear tech-support lock は、テクニカルサポート情報の実際の収集が行われるバックグラウンドVSH セッションを強制終了しないからです。**clear tech-support lock** コマンドは、**show tech support** CLI が呼び出されたフォアグラウンド VSH セッションのみを強制終了します。

  show tech-support セッションを正しく強制終了するには、Ctrl+C を使用します。

  誤って **clear tech-support lock** を使用した場合は、次の手順を実行してバックグラウンドVSH プロセスを強制終了してください。

  1. Bash シェルに入ります。
  2. **show tech support** コマンドの VSH セッション（**ps -l | more**）を見つけます。
  3. **show tech support** セッションの VSH に関連付けられている PID（**kill -9** *PID* など）を強制終了します。

Bash へのアクセス

Cisco NX-OS では、Cisco NX-OS dev-ops ロールまたは Cisco NX-OS network-admin ロールに関連付けられたユーザ アカウントから Bash にアクセスできます。

次の例は、dev-ops ロールと network-admin ロールの権限を示しています。

```
switch# show role name dev-ops

Role: dev-ops
  Description: Predefined system role for devops access. This role
  cannot be modified.
  Vlan policy: permit (default)
  Interface policy: permit (default)
```

```
  Vrf policy: permit (default)
  --------------------------------------------------------------------
  Rule    Perm    Type        Scope               Entity
  --------------------------------------------------------------------
  4       permit  command                         conf t ; username *
  3       permit  command                         bcm module *
  2       permit  command                         run bash *
  1       permit  command                         python *

switch# show role name network-admin

Role: network-admin
  Description: Predefined network admin role has access to all commands
  on the switch
  --------------------------------------------------------------------
  Rule    Perm    Type        Scope               Entity
  --------------------------------------------------------------------
  1       permit  read-write
switch#
```

feature bash-shell コマンドを実行すると、Bash が有効になります。

この **run bash** コマンドは Bash を読み込み、ユーザーのホーム ディレクトリから開始します。

次の例は、Bash シェル機能を有効にする方法と、Bash を実行する方法を示しています。

```
switch# configure terminal
switch(config)# feature bash-shell

switch# run?
  run         Execute/run program
  run-script  Run shell scripts

switch# run bash?
  bash  Linux-bash

switch# run bash
bash-4.2$ whoami
admin
bash-4.2$ pwd
/bootflash/home/admin
bash-4.2$
```

（注） **run bash** コマンドで Bash コマンドを実行することもできます。

たとえば、**run bash** コマンド を使用して **whoami** を実行することもできます。

```
run bash whoami
```

ユーザー **shelltype** を構成して Bash を実行することもできます。

```
username foo shelltype bash
```

このコマンドにより、ログイン時に Bash シェルを直接実行できるようになります。この場合、**feature bash-shell** を有効にする必要はありません。

権限をルートにエスカレーションする

管理者ユーザーの特権は、ルート アクセスの特権をエスカレーションできます。

以下は、権限をエスカレーションするためのガイドラインです：

- 管理者権限ユーザー（network-admin/vdc-admin）は、NX-OS における、Linux の root 権限ユーザーに相当します。
- 認証された管理者ユーザーのみが権限を root に昇格できます。認証された管理者権限ユーザーにパスワードは必要ありません。*
- 権限をエスカレーションする前に、Bash を有効にする必要があります。
- 非管理インターフェイスを介した `root` ユーザー名を使用したスイッチへの SSH では、root ユーザーの Linux Bash シェル タイプ アクセスがデフォルトになります。NX-OS シェル アクセスに戻るために **vsh** を入力します。

* Cisco NX-OS リリース 9.2(3) 以降では、管理者（ネットワーク管理者ロールを持つユーザー）の特権ユーザーであっても、一部の使用例でパスワードプロンプトが必要な場合は、**system security hardening sudo prompt-password** コマンドを入力します。

NX-OS ネットワーク管理者ユーザーは、次の場合に root にエスカレーションして、構成コマンドを NX-OS VSH に渡す必要があります。

- NX-OS ユーザーはシェルタイプの Bash を使用し、シェルタイプの Bash を使用してスイッチにログインします。
- Bash でスイッチにログインした NX-OS ユーザーは、引き続きスイッチで Bash を使用します。

sudo su 'vsh -c "<configuration commands>"' または **sudo bash -c 'vsh -c "<configuration commands>"'** を実行します。

次の例は、デフォルトのシェルタイプが Bash であるネットワーク管理者ユーザー MyUser が、**sudo** を使用して構成コマンドを NX-OS に渡す方法を示しています。

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ sudo vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show
interface eth1/2 brief"

--------------------------------------------------------------------------------
Ethernet      VLAN    Type Mode   Status  Reason                   Speed     Port
Interface                                                                    Ch #
--------------------------------------------------------------------------------
Eth1/2        --      eth  routed down    Administratively down      auto(D) --
```

次の例は、デフォルトのシェルタイプが Bash であるネットワーク管理者ユーザー MyUser が、NX-OS に入り、NX-OS で Bash を実行する方法を示しています。

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ vsh -h
Cisco NX-OS Software
```

```
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
Nexus 9000v software ("Nexus 9000v Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.

Any use or disclosure, in whole or in part, of the Nexus 9000v Software
or Documentation to any third party for any purposes is expressly
prohibited except as otherwise authorized by Cisco in writing.
The copyrights to certain works contained herein are owned by other
third parties and are used and distributed under license. Some parts
of this software may be covered under the GNU Public License or the
GNU Lesser General Public License. A copy of each such license is
available at
http://www.gnu.org/licenses/gpl.html and
http://www.gnu.org/licenses/lgpl.html
*****************************************************************************
*  Nexus 9000v is strictly limited to use for evaluation, demonstration      *
*  and NX-OS education. Any use or disclosure, in whole or in part of     *
*  the Nexus 9000v Software or Documentation to any third party for any      *
*  purposes is expressly prohibited except as otherwise authorized by     *
*  Cisco in writing.                                                        *
*****************************************************************************
switch# run bash
bash-4.2$ vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show interface
 eth1/2 brief"

--------------------------------------------------------------------------------
Ethernet      VLAN    Type Mode   Status  Reason                   Speed     Port
Interface                                                                    Ch #
--------------------------------------------------------------------------------
Eth1/2        --      eth  routed down    Administratively down      auto(D) --
```

（注） **sudo su -** は使用しないでください。使用すると、システムがハングします。

次の例は、特権を root にエスカレーションする方法と、エスカレーションを確認する方法を表示しています。

```
switch# run bash
bash-4.2$ sudo su root
bash-4.2# whoami
root
bash-4.2# exit
exit
```

Bash コマンドの例

このセクションには、Bash コマンドと出力の例が含まれています。

システム統計情報の表示

次の例は、システム統計情報の表示方法を示しています：

```
switch# run bash
bash-4.2$ cat /proc/meminfo
<snip>
MemTotal:       16402560 kB
MemFree:        14098136 kB
Buffers:           11492 kB
Cached:          1287880 kB
SwapCached:            0 kB
Active:          1109448 kB
Inactive:         717036 kB
Active(anon):     817856 kB
Inactive(anon):   702880 kB
Active(file):     291592 kB
Inactive(file):    14156 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                32 kB
Writeback:             0 kB
AnonPages:        527088 kB
Mapped:            97832 kB
<\snip>
```

CLI からの Bash の実行

次に、**run bash**コマンドを使用して Bash から **ps** を実行する例を示します。

```
switch# run bash ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   528 poll_s ?        00:00:03 init
1 S     0     2     0  0  80   0 -     0 kthrea ?        00:00:00 kthreadd
1 S     0     3     2  0  80   0 -     0 run_ks ?        00:00:56 ksoftirqd/0
1 S     0     6     2  0 -40   - -     0 cpu_st ?        00:00:00 migration/0
1 S     0     7     2  0 -40   - -     0 watchd ?        00:00:00 watchdog/0
1 S     0     8     2  0 -40   - -     0 cpu_st ?        00:00:00 migration/1
1 S     0     9     2  0  80   0 -     0 worker ?        00:00:00 kworker/1:0
1 S     0    10     2  0  80   0 -     0 run_ks ?        00:00:00 ksoftirqd/1
```

機能 RPM の管理

RPM インストールの前提条件

RPMをインストールまたは追加する前に、次の手順によりシステムの準備ができていることを確認します。

手順の概要

1. switch# **show logging logfile | grep -i "System ready"**
2. switch# **run bash sudo su**

手順の詳細

	コマンドまたはアクション	目的
Step 1	switch# **show logging logfile | grep -i "System ready"**	Bash を実行する前に、この手順によって、RPM をインストールまたは追加する前のシステムの準備ができていることを確認します。 以下のような出力が表示されれば、続行します。 **2018 Mar 27 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready**

Step 2	switch# **run bash sudo su**
	例:
	`switch# run bash sudo su`
	`bash-4.2#`
		Bash をロードします。

Bash からの機能 RPM のインストール

手順

	コマンドまたはアクション	目的
Step 1	**sudo dnf installed | grep** *platform*	スイッチにインストールされている NX-OS 機能 RPM のリストを表示します。
Step 2	**dnf list available**	使用可能な RPM のリストを表示します。
Step 3	**sudo dnf -y install** *rpm*	使用可能な RPM をインストールします。

例

次に、**bfd** RPM をインストールする例を示します。

```
bash-4.2$ dnf list installed | grep n9000
base-files.n9000                        3.0.14-r74.2                    installed
bfd.lib32_n9000                         1.0.0-r0                        installed
core.lib32_n9000                        1.0.0-r0                        installed
eigrp.lib32_n9000                       1.0.0-r0                        installed
eth.lib32_n9000                         1.0.0-r0                        installed
isis.lib32_n9000                        1.0.0-r0                        installed
lacp.lib32_n9000                        1.0.0-r0                        installed
linecard.lib32_n9000                    1.0.0-r0                        installed
lldp.lib32_n9000                        1.0.0-r0                        installed
ntp.lib32_n9000                         1.0.0-r0                        installed
nxos-ssh.lib32_n9000                    1.0.0-r0                        installed
ospf.lib32_n9000                        1.0.0-r0                        installed
perf-cisco.n9000_gdb                    3.12-r0                         installed
platform.lib32_n9000                    1.0.0-r0                        installed
shadow-securetty.n9000_gdb              4.1.4.3-r1                      installed
snmp.lib32_n9000                        1.0.0-r0                        installed
```

```
svi.lib32_n9000                         1.0.0-r0                        installed
sysvinit-inittab.n9000_gdb              2.88dsf-r14                     installed
tacacs.lib32_n9000                      1.0.0-r0                        installed
task-nxos-base.n9000_gdb                1.0-r0                          installed
tor.lib32_n9000                         1.0.0-r0                        installed
vtp.lib32_n9000                         1.0.0-r0                        installed
bash-4.2$ dnf list available
bgp.lib32_n9000                         1.0.0-r0
bash-4.2$ sudo dnf -y install bfd
```

（注） 起動時のスイッチのリロード時に、永続的な RPM のため、**dnf** の代わりに **rpm** コマンドを使用します。そうしなかった場合、最初に **dnf bash** または **install cli** を使用してインストールされた RPM は、*インストール済み* としてではなく、*リポジトリ名* または *ファイル名* で表示されます。

機能 RPM のアップグレード

始める前に

dnf リポジトリに RPM の上位バージョンが存在する必要があります。

手順の概要

1. **sudo dnf -y upgrade***rpm*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**sudo dnf -y upgrade***rpm*	インストールされている RPM をアップグレードします。

例

次に、**bfd** RPM のアップグレードの例を示します。

```
bash-4.2$ sudo dnf -y upgrade bfd
```

機能 RPM のダウングレード

手順の概要

1. **sudo dnf -y downgrade***rpm*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**sudo dnf -y downgrade***rpm*	いずれかの dnf リポジトリに下位バージョンの RPM がある場合に、RPM をダウングレードします。

例

次に、**bfd** RPM をダウングレードする例を示します。

```
bash-4.2$ sudo dnf -y downgrade bfd
```

機能 RPM の消去

（注） SNMP RPM および NTP RPM は保護されており、消去できません。

これらの RPM をアップグレードまたはダウングレードすることはできます。アップグレードまたはダウングレードを有効にするには、システムのリロードが必要です。

保護された RPM のリストについては、`/etc/dnf/protected.d/protected_pkgs.conf` を参照してください。

手順の概要

1. **sudo dnf -y erase***rpm*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**sudo dnf -y erase***rpm*	RPM を消去します。

例

次の例は、**bfd** RPM を消去する方法を示しています。

```
bash-4.2$ sudo dnf -y erase bfd
```

DME のモジュール性のサポート

NX-OS リリース 9.3(1) 以降、Cisco NX-OS イメージは DME のモジュール性をサポートします。これは、スイッチの RPM マネージャとの相互運用により、DME RPM の非侵入型アップグレードまたはダウングレードを可能にします。非侵入型のアップグレードまたはダウングレードにより、

システムの再起動を実行せずに RPM をインストールできます。また、DME データベースに構成が含まれている他のアプリケーションが妨害されるのを防ぐことができます。DME のモジュール性を使用すると、ISSU やシステムのリロードを行わずに、モデルの変更をスイッチに適用できます。

（注） DME RPM をロードした後、VSH を再起動して新しい MO のクエリを有効にする必要があります。

DME RPM のインストール

デフォルトでは、NX-OS リリース 9.3(1) にアップグレードすると、必須のアップグレード可能 RPM パッケージであるベース DME RPM がインストールされ、アクティブになります。DME RPM は、RPM ファイルのデフォルト インストール ディレクトリ（`/rpms`）にインストールされます。

コードまたはモデルを変更する場合は、DME RPM をインストールする必要があります。インストールするには、**install** コマンドを使用する NX-OS RPM マネージャ、または **dnf** などの標準 RPM ツールを使用します。**dnf** を使用する場合は、スイッチの Bash シェルにアクセスする必要があります。

Step 1 **copy** *path-to-dme-rpm* **bootflash:** [*//sup-#*][*/path*]

例：

```
switch-1# copy scp://test@10.1.1.1/dme-2.0.1.0-9.3.1.lib32_n9000.rpm bootflash://
switch-1#
```

SCP を使用して DME RPM をブートフラッシュにコピーします。

Step 2 DME RPM をインストールまたはアップグレードするには、次のいずれかの方法を選択します。

NX-OS の **install** コマンドを使用するには、次のコマンドを実行します。

- **install add** *path-to-dme-rpm* **activate**

  例：

```
switch-1#install add dme-2.0.1.0-9.3.1.lib32_n9000.rpm activate
Adding the patch (/dme-2.0.1.0-9.3.1.lib32_n9000.rpm)
[####################] 100%
Install operation 90 completed successfully at Fri Jun  7 07:51:58 2019

Activating the patch (/dme-2.0.1.0-9.3.1.lib32_n9000.rpm)
[####################] 100%
Install operation 91 completed successfully at Fri Jun  7 07:52:35 2019
switch-1#
```

- **install add** *path-to-dme-rpm* **activate upgrade**

  例：

```
switch-1#install add dme-2.0.1.0-9.3.1.lib32_n9000.rpm activate upgrade
Adding the patch (/dme-2.0.1.0-9.3.1.lib32_n9000.rpm)
[####################] 100%
Install operation 87 completed successfully at Fri Jun  7 07:18:55 2019
```

```
Activating the patch (/dme-2.0.1.0-9.3.1.lib32_n9000.rpm)
[####################] 100%
Install operation 88 completed successfully at Fri Jun  7 07:19:35 2019
switch-1#
```

- **install add** *path-to-dme-rpm*　それから **install activate** *path-to-dme-rpm*

  例：

```
switch-1#install add bootflash:dme-2.0.1.0-9.3.1.lib32_n9000.rpm
[####################] 100%
Install operation 92 completed successfully at Fri Jun  7 09:31:04 2019
switch-1#install activate dme-2.0.1.0-9.3.1.lib32_n9000.rpm
[####################] 100%
Install operation 93 completed successfully at Fri Jun  7 09:31:55 2019
switch-1#
```

dnf install を使用するには、次のコマンドを実行します。

- **dnf install --add** *path-to-dme-rpm*

```
switch-1# dnf install --add bootflash:///dme-2.0.10.0-9.3.1.lib32_n9000.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
[##################  ] 90%Install operation 96 completed successfully at Fri Jun  7 22:58:50
2019.

[####################] 100%
switch-1#
```

- **dnf install --no-persist --nocommit***path-to-dme-rpm*

  このオプションには、次に示すようにユーザーの操作が必要です。

  例：

```
switch-1# dnf install --no-persist --nocommit dme-2.0.10.0-9.3.1.lib32_n9000
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
localdb/primary                                          | 6.2 kB     00:00 ...
localdb                                                                   2/2
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
wrl-repo                                                 |  951 B     00:00 ...
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package dme.lib32_n9000 0:2.0.1.0-9.3.1 will be updated
---> Package dme.lib32_n9000 0:2.0.10.0-9.3.1 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package     Arch                Version                   Repository       Size
================================================================================
Updating:
 dme         lib32_n9000         2.0.10.0-9.3.1            localdb          45 M

Transaction Summary
================================================================================
```

```
Upgrade       1 Package

Total download size: 45 M
Is this ok [y/N]: y
Retrieving key from file:///etc/pki/rpm-gpg/arm-Nexus9k-dev.gpg
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
/bootflash/.rpmstore/config/etc/pki/rpm-gpg/arm-Nexus9k-dev.gpg
System at HA Standby, running transaction on Standby first
  Updating   : dme-2.0.10.0-9.3.1.lib32_n9000                              1/2
starting pre-install package version mgmt for dme
pre-install for dme complete
ln: failed to create symbolic link /var/run/mgmt/sharedmeta-hash: File exists
ln: failed to create symbolic link /var/run/mgmt/dme-objstores.conf: File exists
ln: failed to create symbolic link /var/run/mgmt/samlog.config: File exists
mgmt/
mgmt/shmetafiles/
mgmt/shmetafiles/sharedmeta-ArgMetaData
mgmt/shmetafiles/sharedmeta-RelsMetaData
mgmt/shmetafiles/sharedmeta-ClassRelMetaData
mgmt/shmetafiles/sharedmeta-ChunkMetaData
mgmt/shmetafiles/sharedmeta-ConstPropMetaData
mgmt/shmetafiles/sharedmeta-ConstIdMetaData
mgmt/shmetafiles/sharedmeta-ClassMetaData
mgmt/shmetafiles/sharedmeta-PropRefsMetaData
mgmt/shmetafiles/sharedmeta-SvcMetaData
mgmt/shmetafiles/sharedmeta-ActionContextMetaData
mgmt/shmetafiles/sharedmeta-ConstDefTypeMetaData
mgmt/shmetafiles/sharedmeta-ConstArgMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
mgmt/shmetafiles/sharedmeta-ConstMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
  Cleanup    : dme-2.0.1.0-9.3.1.lib32_n9000                               2/2

Updated:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Complete!
switch-1#
```

インストールされている RPM の確認

NX-OS **show install** コマンドまたは **dnf list** を使用して、DME RPM がインストールされているかどうかを確認できます。

方法を選択します。

- NX-OS の場合：

  show install active

  例：

```
switch-1# show install active
Boot Image:
        NXOS Image: bootflash:///<boot_image.bin>

Active Packages:
        dme-2.0.1.0-9.3.1.lib32_n9000
switch-1#
```

- **dnf list** では、**dnf** コマンドを発行する前にスイッチの Bash シェル（**run bash**）にログインする必要があります。

  dnf list --patch-only installed | grep dme

  例：

```
switch-1# dnf list --patch-only installed | grep dme
dme.lib32_n9000                          2.0.1.0-9.3.1                   @localdb
```

ローカル リポジトリの RPM のクエリ

スイッチ上の（ローカル）リポジトリを照会して、RPM が存在するかどうかを確認できます。

Step 1 **run bash**

例：

```
switch-1# run bash
bash-4.3$
```

スイッチの Bash シェルにログインします。

Step 2 **ls /bootflash/.rpmstore/patching/localrepo/dme-2.0.1.0-9.3.1.lib32_n9000.rpm**

例：

```
bash-4.3$ ls /bootflash/.rpmstore/patching/localrepo/dme-2.0.1.0-9.3.1.lib32_n9000.rpm
inactive_feature_rpms.inf
repodata

bash-4.3$
```

ベース DME RPM がインストールされている場合は、`/rpms` にあります。

DME RPM のバージョン間ダウングレード

NX-OS コマンド **install** または **dnf** を使用して、DME RPM の上位バージョンから下位バージョンにダウングレードできます。ダウングレードすることで、DMEのモジュラリティ機能が保持されます。

DME RPM は保護されているため、**install deactivate** と **install remove** はサポートされていません。

ダウングレード方法を選択します。

NX-OS の場合：

- **install add** *path-to-dme-rpm* **activate downgrade**

  例：

```
switch-1# install add bootflash:dme-2.0.1.0-9.3.1.lib32_n9000.rpm activate downgrade
Adding the patch (/dme-2.0.1.0-9.3.1.lib32_n9000.rpm)
[####################] 100%
Install operation 94 completed successfully at Fri Jun  7 22:48:34 2019

Activating the patch (/dme-2.0.1.0-9.3.1.lib32_n9000.rpm)
[####################] 100%
Install operation 95 completed successfully at Fri Jun  7 22:49:12 2019
switch-1#
```

- **show install active | include dme**

  例：

```
switch-1# show install active | include dme
        dme-2.0.1.0-9.3.1.lib32_n9000
switch-1#
```

この例では、DME RPM がバージョン 2.0.1.0-9.3.1 にダウングレードされました。

dnf の場合、root ユーザー（**run bash sudo su**）として Bash シェルでコマンドを実行する必要があります。

- Bash で、**dnf downgrade dme** *dme-rpm* を実行します。

  このオプションを使用すると、リポジトリ内の下位バージョンの DME RPM に直接ダウングレードできます。

  次のコマンド出力で強調表示されているように、このオプションを完了するには、ユーザーの介入が必要です。

  例：

```
bash-4.3# dnf downgrade dme 2.0.1.0-9.3.1
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
Setting up Downgrade Process
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
wrl-repo                                                 |  951 B     00:00 ...
Resolving Dependencies
--> Running transaction check
---> Package dme.lib32_n9000 0:2.0.1.0-9.3.1 will be a downgrade
---> Package dme.lib32_n9000 0:2.0.10.0-9.3.1 will be erased
--> Finished Dependency Resolution

Dependencies Resolved
================================================================================
 Package     Arch                Version                   Repository       Size
================================================================================
Downgrading:
 dme         lib32_n9000         2.0.10.0-9.3.1            localdb          45 M
```

```
Transaction Summary
================================================================================
Downgrade     1 Package

Total download size: 45 M
Is this ok [y/N]: y
Retrieving key from file:///etc/pki/rpm-gpg/arm-Nexus9k-dev.gpg
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
/bootflash/.rpmstore/config/etc/pki/rpm-gpg/arm-Nexus9k-dev.gpg
System at HA Standby, running transaction on Standby first
  Installing : dme-2.0.1.0-9.3.1.lib32_n9000                                1/2
starting pre-install package version mgmt for dme
pre-install for dme complete
ln: failed to create symbolic link /var/run/mgmt/sharedmeta-hash: File exists
ln: failed to create symbolic link /var/run/mgmt/dme-objstores.conf: File exists
ln: failed to create symbolic link /var/run/mgmt/samlog.config: File exists
mgmt/
mgmt/shmetafiles/
mgmt/shmetafiles/sharedmeta-ArgMetaData
mgmt/shmetafiles/sharedmeta-RelsMetaData
mgmt/shmetafiles/sharedmeta-ClassRelMetaData
mgmt/shmetafiles/sharedmeta-ChunkMetaData
mgmt/shmetafiles/sharedmeta-ConstPropMetaData
mgmt/shmetafiles/sharedmeta-ConstIdMetaData
mgmt/shmetafiles/sharedmeta-ClassMetaData
mgmt/shmetafiles/sharedmeta-PropRefsMetaData
mgmt/shmetafiles/sharedmeta-SvcMetaData
mgmt/shmetafiles/sharedmeta-ActionContextMetaData
mgmt/shmetafiles/sharedmeta-ConstDefTypeMetaData
mgmt/shmetafiles/sharedmeta-ConstArgMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
mgmt/shmetafiles/sharedmeta-ConstMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
  Cleanup    : dme-2.0.10.0-9.3.1.lib32_n9000                               2/2

Removed:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Installed:
  dme.lib32_n9000 0:2.0.1.0-9.3.1

Complete!
```

DME RPM のあるバージョンから下位のバージョンにダウングレードします。この例では、バージョン 2.0.10.0-9.3.1 がバージョン 2.0.1.0-9.3.1 にダウングレードされます。

- **dnf list --patch-only installed | grep dme**

  例：

```
bash-4.3# dnf list --patch-only installed | grep dme
dme.lib32_n9000                          2.0.1.0-9.3.1                @groups-repo
bash-4.3#
```

インストールされている DME RPM のバージョンを表示します。

ベース RPM へのダウングレード

NX-OS の **install** コマンドを使用してベース DME RPM をインストールするか、または **dnf downgrade** を使用して、上位バージョンの DME RPM からベース DME RPM にダウングレードできます。

ダウングレード方法を選択します。

NX-OS の場合：

- **install activate** *dme-rpm*

  例：

```
switch-1# install activate dme-2.0.0.0-9.2.1.lib32_n9000.rpm
[####################] 100%
Install operation 89 completed successfully at Fri Jun  7 07:21:45 2019
switch-1#
```

- **show install active | dme**

  例：

```
switch-1# show install active | include dme
        dme-2.0.0.0-9.2.1.lib32_n9000
switch-1#
```

dnf の場合、root ユーザー（**run bash sudo su**）として Bash シェルでコマンドを実行する必要があります。

- Bash で、**dnf downgrade dme** *dme-rpm* を実行します。

  このオプションにより、ベース DME RPM に直接ダウングレードできます。

  次のコマンド出力で強調表示されているように、このオプションを完了するには、ユーザーの介入が必要です。

  例：

```
bash-4.3# dnf downgrade dme-2.0.0.0-9.3.1.lib32_n9000
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
Setting up Downgrade Process
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
wrl-repo                                                 |  951 B     00:00 ...
Resolving Dependencies
--> Running transaction check
---> Package dme.lib32_n9000 0:2.0.0.0-9.3.1 will be a downgrade
---> Package dme.lib32_n9000 0:2.0.10.0-9.3.1 will be erased
--> Finished Dependency Resolution

Dependencies Resolved
```

```
================================================================================
Package    Arch               Version                 Repository          Size
================================================================================
Downgrading:
dme        lib32_n9000        2.0.0.0-9.3.1           groups-repo         44 M

Transaction Summary
================================================================================
Downgrade     1 Package

Total download size: 44 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : dme-2.0.0.0-9.3.1.lib32_n9000                               1/2
starting pre-install package version mgmt for dme
pre-install for dme complete
mgmt/
mgmt/shmetafiles/
mgmt/shmetafiles/sharedmeta-ChunkMetaData
mgmt/shmetafiles/sharedmeta-ClassMetaData
mgmt/shmetafiles/sharedmeta-ArgMetaData
mgmt/shmetafiles/sharedmeta-ConstMetaData
mgmt/shmetafiles/sharedmeta-ConstIdMetaData
mgmt/shmetafiles/sharedmeta-ConstDefTypeMetaData
mgmt/shmetafiles/sharedmeta-ConstPropMetaData
mgmt/shmetafiles/sharedmeta-ConstArgMetaData
mgmt/shmetafiles/sharedmeta-ClassRelMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
mgmt/shmetafiles/sharedmeta-PropRefsMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-RelsMetaData
mgmt/shmetafiles/sharedmeta-ActionContextMetaData
mgmt/shmetafiles/sharedmeta-SvcMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
  Cleanup    : dme-2.0.10.0-9.3.1.lib32_n9000                              2/2

Removed:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Installed:
  dme.lib32_n9000 0:2.0.0.0-9.3.1

Complete!
bash-4.3#
```

ベース DME RPM をインストールします。

- **dnf list --patch-only installed | grep dme**

  例:

```
bash-4.3# dnf list --patch-only installed | grep dme
dme.lib32_n9000                         2.0.0.0-9.3.1               @groups-repo
bash-4.3#
```

インストールされているベース DME RPM を表示します。

パッチ RPM の管理

RPM インストールの前提条件

RPMをインストールまたは追加する前に、次の手順によりシステムの準備ができていることを確認します。

手順の概要

1. switch# **show logging logfile | grep -i "System ready"**
2. switch# **run bash sudo su**

手順の詳細

	コマンドまたはアクション	目的
Step 1	switch# **show logging logfile	grep -i "System ready"**	Bash を実行する前に、この手順によって、RPM をインストールまたは追加する前のシステムの準備ができていることを確認します。 以下のような出力が表示されれば、続行します。 **2018 Mar 27 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready**

Step 2	switch# **run bash sudo su**
	例：
	`switch# run bash sudo su`
	`bash-4.2#`
		Bash をロードします。

Bash からのパッチ RPM の追加

手順

	コマンドまたはアクション	目的
Step 1	**dnf list --patch-only**	スイッチに存在するパッチRPMのリストを表示します。
Step 2	**sudo dnf install --add** *URL_of_patch*	リポジトリにパッチを追加します。ここで *URL_of_patch* は、標準的な Linux 形式の **/bootflash/*patch*** などではなく、**bootflash:/*patch*** などの明確に定義された形式です。

	コマンドまたはアクション	目的
Step 3	**dnf list --patch-only available**	リポジトリに追加されているものの非アクティブ状態のパッチのリストを表示します。

例

次に、**nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000** RPM をインストールする例を示します。

```
bash-4.2# dnf list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
bash-4.2#
bash-4.2# sudo dnf install --add
bootflash:/nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
[##############     ] 70%Install operation 135 completed successfully at Tue Mar 27 17:45:34
 2018.

[####################] 100%
bash-4.2#
```

パッチ RPM がインストールされたら、正しくインストールされたことを確認します。次のコマンドは、リポジトリに追加され、非アクティブ状態のパッチのリストを表示します。

```
bash-4.2# dnf list --patch-only available
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000    1.0.0-7.0.3.I7.3    patching
bash-4.2#
```

RPM がバンドルされている tar ファイルを使用して、tar ファイルからリポジトリにパッチを追加することもできます。次に、nxos.CSCab00002_CSCab00003-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 tar ファイルに含まれる 2 つの RPM をパッチ リポジトリに追加する例を示します。

```
bash-4.2# sudo dnf install --add
bootflash:/nxos.CSCab00002_CSCab00003-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000.tar
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
```

```
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
[###############     ] 70%Install operation 146 completed successfully at Tue Mar 27 21:17:39
 2018.

[####################] 100%
bash-4.2#
bash-4.2# dnf list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
patching/primary                                         |  942 B     00:00 ...
patching                                                                   2/2
thirdparty                                               |  951 B     00:00 ...
nxos.CSCab00003-n9k_ALL.lib32_n9000    1.0.0-7.0.3.I7.3    patching
nxos.CSCab00002-n9k_ALL.lib32_n9000    1.0.0-7.0.3.I7.3    patching
bash-4.2#
```

パッチ RPM のアクティブ化

始める前に

Bash からのパッチ RPM の追加（30 ページ） の手順に従って、必要なパッチ RPM がリポジトリに追加されていることを確認します。

手順

	コマンドまたはアクション	目的
Step 1	**sudo dnf install** *patch_RPM* **--nocommit**	パッチ RPM をアクティブにします。ここで、*patch_RPM* はリポジトリにあるパッチです。この手順では、パッチの場所を指定しないでください。 （注） --nocommit フラグをコマンドに追加すると、パッチ RPM がこの手順でアクティブになりますが、コミットされません。パッチ RPM をアクティブ化した後にコミットする手順については、パッチ RPM のコミット（34 ページ） を参照してください。

例

次に、**nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000** パッチ RPM をアクティブにする例を示します。

```
bash-4.2# sudo dnf install nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --nocommit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0-7.0.3.I7.3 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                    Arch           Version               Repository   Size
================================================================================
Installing:
 nxos.CSCab00001-n9k_ALL    lib32_n9000    1.0.0-7.0.3.I7.3      patching     28 k

Transaction Summary
================================================================================
Install       1 Package

Total download size: 28 k
Installed size: 82 k
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000            1/1
[##################  ] 90%error: reading
/var/sysmgr/tmp/patches/CSCab00001-n9k_ALL/isan/bin/sysinfo manifest, non-printable characters
 found

Installed:
  nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0-7.0.3.I7.3

Complete!
Install operation 140 completed successfully at Tue Mar 27 18:07:40 2018.

[####################] 100%
bash-4.2#
```

次のコマンドを入力して、パッチ RPM が正常にアクティブ化されたことを確認します。

```
bash-4.2# dnf list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000    1.0.0-7.0.3.I7.3    installed
bash-4.2#
```

パッチ RPM のコミット

手順

	コマンドまたはアクション	目的
Step 1	**sudo dnf install** *patch_RPM* **--commit**	パッチ RPM をコミットします。パッチ RPM は、リロード後もアクティブな状態を維持するためにコミットする必要があります。

例

次に、**nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000** パッチ RPM をコミットする例を示します。

```
bash-4.2# sudo dnf install nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --commit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
Install operation 142 completed successfully at Tue Mar 27 18:13:16 2018.

[####################] 100%
bash-4.2#
```

次のコマンドを入力して、パッチ RPM が正常にコミットされたことを確認します。

```
bash-4.2# dnf list --patch-only committed
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000    1.0.0-7.0.3.I7.3    installed
bash-4.2#
```

パッチ RPM の非アクティブ化

手順

	コマンドまたはアクション	目的
Step 1	**sudo dnf erase** *patch_RPM* **--nocommit**	パッチ RPM を非アクティブ化します。 （注） コマンドに --nocommit フラグを追加すると、パッチ RPM はこの手順でのみ非アクティブ化されます。

	コマンドまたはアクション	目的
Step 2	**sudo dnf install** *patch_RPM* **--commit**	パッチRPMをコミットします。パッチRPMをコミットしないまま削除しようとすると、エラーメッセージが表示されます。

例

次に、**nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000** パッチ RPM を非アクティブにする例を示します。

```
bash-4.2# sudo dnf erase nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --nocommit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
Setting up Remove Process
Resolving Dependencies
--> Running transaction check
---> Package nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0-7.0.3.I7.3 will be erased
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                    Arch           Version              Repository    Size
================================================================================
Removing:
 nxos.CSCab00001-n9k_ALL    lib32_n9000    1.0.0-7.0.3.I7.3     @patching     82 k

Transaction Summary
================================================================================
Remove        1 Package

Installed size: 82 k
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
[######              ] 30%error: reading
/var/sysmgr/tmp/patches/CSCab00001-n9k_ALL/isan/bin/sysinfo manifest, non-printable characters
 found
  Erasing    : nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000            1/1
[##################  ] 90%
Removed:
  nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0-7.0.3.I7.3

Complete!
Install operation 143 completed successfully at Tue Mar 27 21:03:47 2018.

[####################] 100%
bash-4.2#
```

パッチ RPM は、非アクティブ化した後にコミットする必要があります。パッチ RPM を非アクティブ化した後にコミットしなかった場合に、パッチ RPM の削除 （36 ページ）の手順を使用してパッチ RPM を削除しようとすると、エラー メッセージが表示されます。

```
bash-4.2# sudo dnf install nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --commit
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
Install operation 144 completed successfully at Tue Mar 27 21:09:28 2018.

[####################] 100%
bash-4.2#
```

次のコマンドを入力して、パッチ RPM が正常にコミットされたことを確認します。

```
bash-4.2# dnf list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000    1.0.0-7.0.3.I7.3    patching
bash-4.2#
```

パッチ RPM の削除

手順

	コマンドまたはアクション	目的
Step 1	**sudo dnf install --remove** *patch_RPM*	非アクティブなパッチ RPM を削除します。

例

次に、**nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000** パッチ RPM を削除する例を示します。

```
bash-4.2# sudo dnf install --remove nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                                | 1.1 kB     00:00 ...
localdb                                                    |  951 B     00:00 ...
patching                                                   |  951 B     00:00 ...
thirdparty                                                 |  951 B     00:00 ...
[##########        ] 50%Install operation 145 completed successfully at Tue Mar 27 21:11:05
 2018.

[####################] 100%
bash-4.2#
```

（注） パッチ RPM を削除しようとした後に、次のエラーメッセージが表示されたとします。

Install operation 11 "failed because patch was not committed". at Wed Mar 28 22:14:05 2018

これは、削除を試みる前にパッチ RPM をコミットしていなかったことを意味しています。パッチ RPM を削除する前にコミットする手順については、パッチ RPM の非アクティブ化（34 ページ）を参照してください。

次のコマンドを入力して、非アクティブなパッチ RPM が正常に削除されたことを確認します。

```
bash-4.2# dnf list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
groups-repo                                              | 1.1 kB     00:00 ...
localdb                                                  |  951 B     00:00 ...
patching                                                 |  951 B     00:00 ...
patching/primary                                         |  197 B     00:00 ...
thirdparty                                               |  951 B     00:00 ...
bash-4.2#
```

SDK または ISO で構築されたサードパーティ プロセスの永続的なデーモン化

アプリケーションには、/etc/init.d/*application_name* にインストールされる起動 Bash スクリプトが必要です。この起動 Bash スクリプトは、次の一般的な形式にする必要があります（この形式の詳細については、http://linux.die.net/man/8/chkconfigを参照してください）。

```
#!/bin/bash
#
# <application_name> Short description of your application
#
# chkconfig: 2345 15 85
# description: Short description of your application
#
### BEGIN INIT INFO
# Provides: <application_name>
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Short description of your application
### END INIT INFO
# See how we were called.
case "$1" in
start)
# Put your startup commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
stop)
# Put your stop commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
status)
# Put your status commands here
```

```
# Set RETVAL to 0 for success, non-0 for failure
;;
restart|force-reload|reload)
# Put your restart commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
```

ネイティブBashシェルからのアプリケーションの永続的な起動

Step 1 作成したアプリケーション起動 Bash スクリプトを/etc/init.d/*application_name*にインストールします。

Step 2 /etc/init.d/*application_name* start でアプリケーションを開始します

Step 3 **chkconfig --add** *application_name* を入力します

Step 4 **chkconfig --level 3** *application_name* **on**を入力します

実行レベル 3 は、標準のマルチユーザー実行レベルであり、スイッチが通常実行されるレベルです。

Step 5 **chkconfig --list** *application_name* を実行して、アプリケーションがレベル 3 で実行されるようにスケジュールされていることを確認し、レベル 3 が on に設定されていることを確認します。

Step 6 アプリケーションが /etc/rc3.d に記述されていることを確認します。次のような表示になるはずです。「S」の後に数字が続き、アプリケーション名（この例では tcollector）が続きます。../init.d/*application_name* には Bash 起動スクリプトへのリンクが表示されます。

```
bash-4.2# ls -l /etc/rc3.d/tcollector
lrwxrwxrwx 1 root root 20 Sep 25 22:56 /etc/rc3.d/S15tcollector -> ../init.d/tcollector
bash-4.2#
```

アクティブ ブートフラッシュからスタンバイ ブートフラッシュへのファイルの同期

Cisco Nexus 9500 プラットフォーム スイッチは、通常、高可用性を提供するために 2 つのスーパーバイザ モジュール（1 つのアクティブ スーパーバイザ モジュールと 1 つのスタンバイ スーパーバイザ モジュール）で構成されています。各スーパーバイザモジュールには、ファイル ストレージ用の独自のブートフラッシュ ファイル システムがあり、通常、アクティブ ブートフラッシュ ファイル システムとスタンバイ ブートフラッシュ ファイル システムは互いに独立しています。

アクティブ ブートフラッシュに特定のコンテンツが必要な場合、将来スイッチオーバーが発生した場合に備えて、同じコンテンツがスタンバイ ブートフラッシュにも必要でしょう。

Cisco NX-OS 9.2(2) リリースより前は、アクティブ スーパーバイザ モジュールとスタンバイ スーパーバイザ モジュール間でこのようなコンテンツを手動で管理する必要がありました。Cisco NX-OS 9.2(2) 以降では、スタンバイ スーパーバイザ モジュールが up 状態で使用可能なら、アクティブ スーパーバイザ モジュールまたはアクティブ ブートフラッシュ（/bootflash）上の特定のファイルとディレクトリを、スタンバイ スーパーバイザ モジュールまたはスタンバイ ブートフラッシュ（/bootflash_sup-remote）に自動的に同期できます。同期するファイルとディレクトリを選択するには、スイッチに Bash をロードし、アクティブ ブートフラッシュからスタンバイ ブートフラッシュに同期するファイルとディレクトリを、編集可能ファイル /bootflash/bootflash_sync_list に追加します。

次に例を示します。

```
switch# run bash
bash-4.2# echo "/bootflash/home/admin" | sudo tee --append /bootflash/bootflash_sync_list
bash-4.2# echo "/bootflash/nxos.7.0.3.I7.3.5.bin" | sudo tee --append
/bootflash/bootflash_sync_list
bash-4.2# cat /bootflash/bootflash_sync_list
/bootflash/home/admin
/bootflash/nxos.7.0.3.I7.3.5.bin
```

アクティブ ブートフラッシュのファイルまたはディレクトリに変更が加えられた場合、スタンバイ ブートフラッシュが up 状態で使用可能なら、これらの変更はスタンバイ ブートフラッシュに自動的に同期されます。スタンバイ ブートフラッシュが通常のブート、スイッチオーバー、または手動スタンバイ リロードのいずれかでリブートされた場合、スタンバイ スーパーバイザがオンラインになると、アクティブブートフラッシュへの変更のキャッチアップ同期がスタンバイブートフラッシュにプッシュされます。

次に、編集可能な /bootflash/bootflash_sync_list ファイルの特性と制限事項を示します。

- /bootflash/bootflash_sync_list ファイルは、最初の実行時に自動的に作成されますが、最初の作成状態では空です。
- /bootflash/bootflash_sync_list ファイルのエントリは、次の注意事項に従います。
  - 1 行に 1 エントリ
  - エントリは Linux パスとして指定します（例：/bootflash/img.bin）
  - エントリは /bootflash ファイルシステム内にある必要があります
- /bootflash/bootflash_sync_list ファイル自体は、自動的にスタンバイ ブートフラッシュに同期されます。**copy** virtual shell（VSH）コマンドを使用して、スーパーバイザ モジュールとの間で /bootflash/bootflash_sync_list ファイルを手動でコピーすることもできます。
- 次のコマンドを使用して、スーパーバイザ モジュールで直接 /bootflash/bootflash_sync_list ファイルを編集できます。

```
run bash vi /bootflash/bootflash_sync_list
```

同期イベントからのすべての出力は、ログファイル /var/tmp/bootflash_sync.log にリダイレクトされます。次のいずれかのコマンドを使用して、このログファイルを表示または追跡できます。

```
run bash less /var/tmp/bootflash_sync.log

run bash tail -f /var/tmp/bootflash_sync.log
```

同期スクリプトは、アクティブブートフラッシュディレクトリ上の対応するファイルの削除イベントを明示的に受信しない限り、スタンバイブートフラッシュディレクトリからファイルを削除しません。場合によっては、スタンバイブートフラッシュの使用中のスペースがアクティブブートフラッシュよりも多くなり、アクティブ ブートフラッシュと同期しているときにスタンバイブートフラッシュのスペースが不足することがあります。スタンバイ ブートフラッシュをアクティブブートフラッシュの正確なミラーにする（スタンバイブートフラッシュ上の余分なファイルを削除する）には、次のコマンドを入力します。

```
run bash sudo rsync -a --delete /bootflash/ /bootflash_sup-remote/
```

同期スクリプトは、クラッシュまたは終了することなく、バックグラウンドで実行され続ける必要があります。ただし、何らかの理由で実行が停止した場合は、次のコマンドを使用して手動で再起動できます。

```
run bash sudo /isan/etc/rc.d/rc.isan-start/S98bootflash_sync.sh start
```

Kstack を介してコピー

Cisco NX-OS リリース 9.3(1) 以降では、ファイル コピー操作には、**use-kstack** オプションを使用して別のネットワーク スタックを介して実行するオプションがあります。**use-kstack** を通じてファイルをコピーすると、コピー時間が短縮されます。このオプションは、スイッチから複数のホップにあるリモート サーバーからファイルをコピーする場合に役立ちます。**use-kstack** オプションは、 **scp** や **sftp.** などの標準ファイル コピー機能を通じてスイッチに、またはスイッチからファイルをコピー処理します。

（注） スイッチが FIPS モード機能を実行している場合、**use-kstack** オプションは機能しません。スイッチでFIPSモードが有効になっている場合、コピー操作は引き続き成功しますが、デフォルトのコピー方法が使用されます。

use-kstack を介してコピーするには、NX-OS **copy** コマンドの最後に引数を追加します。たとえば:

```
switch-1# copy scp://test@10.1.1.1/image.bin . vrf management use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin bootflash:// vrf management
```

```
use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin . use-kstack
switch-1#
switch-1# copy scp://test@10.1.1.1/image.bin bootflash:// vrf default
use-kstack
switch-1#
```

use-kstack オプションは、すべての NX-OS **copy** コマンドとファイル システムでサポートされています。オプションは OpenSSL（セキュア コピー）認定済みです。

ネイティブ Bash シェルのアプリケーション例

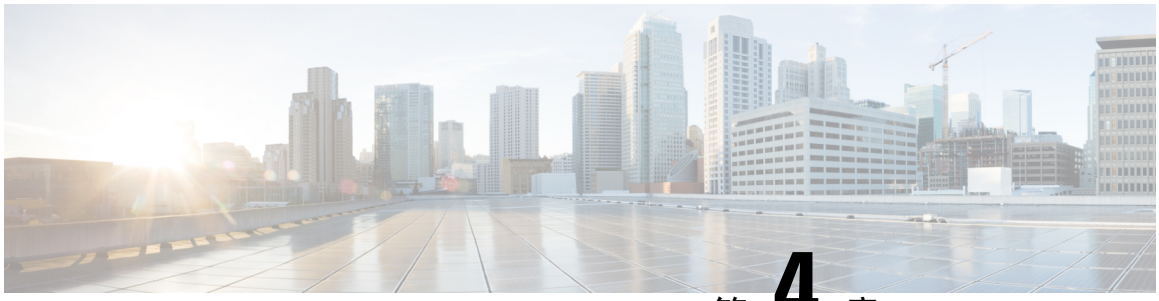
次の例は、ネイティブ Bash シェルのアプリケーションを示しています。

```
bash-4.2# cat /etc/init.d/hello.sh
#!/bin/bash

PIDFILE=/tmp/hello.pid
OUTPUTFILE=/tmp/hello

echo $$ > $PIDFILE
rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
bash-4.2#
bash-4.2#
bash-4.2# cat /etc/init.d/hello
#!/bin/bash
#
# hello Trivial "hello world" example Third Party App
#
# chkconfig: 2345 15 85
# description: Trivial example Third Party App
#
### BEGIN INIT INFO
# Provides: hello
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Trivial example Third Party App
### END INIT INFO

PIDFILE=/tmp/hello.pid

# See how we were called.
case "$1" in
start)
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
stop)
    kill -9 `cat $PIDFILE`
    RETVAL=$?
;;
status)
    ps -p `cat $PIDFILE`
```

```
    RETVAL=$?
;;
restart|force-reload|reload)
    kill -9 `cat $PIDFILE`
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
bash-4.2#
bash-4.2# chkconfig --add hello
bash-4.2# chkconfig --level 3 hello on
bash-4.2# chkconfig --list hello
hello           0:off   1:off   2:on    3:on    4:on    5:on    6:off
bash-4.2# ls -al /etc/rc3.d/*hello*
lrwxrwxrwx 1 root root 15 Sep 27 18:00 /etc/rc3.d/S15hello -> ../init.d/hello
bash-4.2#
bash-4.2# reboot
```

リロード後

```
bash-4.2# ps -ef | grep hello
root      8790     1  0 18:03 ?        00:00:00 /bin/bash /etc/init.d/hello.sh
root      8973  8775  0 18:04 ttyS0    00:00:00 grep hello
bash-4.2#
bash-4.2# ls -al /tmp/hello*
-rw-rw-rw- 1 root root 205 Sep 27 18:04 /tmp/hello
-rw-rw-rw- 1 root root   5 Sep 27 18:03 /tmp/hello.pid
bash-4.2# cat /tmp/hello.pid
8790
bash-4.2# cat /tmp/hello
Sun Sep 27 18:03:49 UTC 2015
Hello World
Sun Sep 27 18:03:59 UTC 2015
Hello World
Sun Sep 27 18:04:09 UTC 2015
Hello World
Sun Sep 27 18:04:19 UTC 2015
Hello World
Sun Sep 27 18:04:29 UTC 2015
Hello World
Sun Sep 27 18:04:39 UTC 2015
Hello World
bash-4.2#
```



第 4 章

ゲスト シェル

Guest Shell について

基盤となる Linux 環境での NX-OS CLI および Bash アクセスに加えて、スイッチは、「ゲスト シェル」と呼ばれる Linux コンテナ（LXC）内で実行される分離された実行スペースへのアクセスをサポートします。

ゲスト シェル内から、network-admin には次の機能があります。

- Linux ネットワーク インターフェイスを介したネットワークへのアクセス。
- スイッチのブートフラッシュへのアクセス。
- スイッチの揮発性 tmpfs へのアクセス。
- スイッチの CLI へのアクセス。
- スイッチのホスト ファイル システムへのアクセス。
- Cisco NX-API REST へのアクセス。

- Python スクリプトをインストールして実行する機能。
- 32 ビットおよび 64 ビットの Linux アプリケーションをインストールして実行する機能。

コンテナ技術によって実行空間を切り離すことで、他の Linux コンテナで実行されているホストシステムやアプリケーションに影響を与えずに、アプリケーションのニーズに合わせて Linux 環境をカスタマイズすることができます。

NX-OS デバイスでは、Linux Containers は virtual-service コマンドでインストールと管理されます。Guest Shell は、virtual-service show コマンドの出力に表示されます。

（注） デフォルトでは、Guest Shell は、有効にすると約 35 MB の RAM と 350 MB のブートフラッシュを占有します。Guest Shell が使用されていない場合は、guestshell destroy コマンドを使用して技術情報を再利用します。

Guestshell に関する注意事項と制限事項

すべてのリリースに共通の注意事項

重要 Guestshell のインストール内でカスタム作業を実行した場合は、`Guestshell のアップグレード`を実行する前に、ブートフラッシュ、オフボックス ストレージ、または Guestshell ルート ファイル システムの外部の他の場所に変更を保存します。

`guestshell upgrade` コマンドは、本質的に、`guestshell destroy`と`guestshell enable`を連続して実行します。

- Guest Shell は、4 GB のメモリを搭載した 3500 モデル（3524、3548、3524-X、3548-X）ではサポートされていません。これは、-XL など、より多くのメモリを備えたプラットフォームでサポートされます。
- Guestshell でサードパーティの DHCPD サーバーを実行している場合、SVI と一緒に使用すると、クライアントに到達するオファーに問題が発生する可能性があります。可能な回避策は、ブロードキャスト応答を使用することです。
- `run guestshell` CLI コマンドを使用して、スイッチの Guestshell にアクセスします。`run guestshell` コマンドは、ホスト シェルへのアクセスに使用される `run bash` コマンドに相当します。このコマンドを使用すると、Guestshell にアクセスして Bash プロンプトを取得したり、Guestshell のコンテキスト内でコマンドを実行したりできます。このコマンドは、パスワードなしの SSH を使用して、デフォルトのネットワーク名前空間にある localhost の使用可能なポートに接続します。

- sshd ユーティリティは、ローカルホストでリッスンして、ネットワークの外部からの接続試行を回避することにより、Guestshell への事前構成された SSH アクセスを保護できます。sshd には次の機能があります。
  - これは、パスワードにフォールバックしないキーベースの認証用に構成されています。
  - Guestshell の再起動後に Guestshell にアクセスするために使用されるキーを読み取ることができるのは root だけです。
  - root だけがホスト上のキーを含むファイルを読み取ることができ、ホスト Bash アクセスを持つ非特権ユーザーがキーを使用して Guestshell に接続できないようにします。ネットワーク管理ユーザーは、Guestshell で sshd の別のインスタンスを開始して、Guestshell ネットワーク管理ユーザーは、Guestshell で sshd の別のインスタンスを開始して、ネットワーク管理ユーザーは、Guestshell で sshd の別のインスタンスを開始して、 アクセスできるようにすることができますが、Guestshell にログインするユーザーにはネットワーク管理者権限も与えられます。

（注） Guestshell 2.2 (0.2) で導入されたキー ファイルは、ユーザー アカウントが作成されたユーザーに対して読み取り可能です。

さらに、Guestshell アカウントは自動的に削除されないため、不要になったときにネットワーク管理者が削除する必要があります。

2.2 (0.2) より前の Guestshell インストールでは、個々のユーザー アカウントが動的に作成されません。

- すぐに使用できる新しいスイッチに Cisco NX-OS ソフトウェア リリースをインストールすると、Guestshell が自動的に有効になります。その後のスイッチ ソフトウェアのアップグレードでは、Guestshell は自動的にアップグレードされません。
- Guestshell リリースでは、配布または配布バージョンが変更されると、メジャー番号が増分します。
- NX-OS の Guestshell は、前面パネルのポートに、ファーストクラスの Linux インターフェイスとしてアクセスできます。
- NX-OS の Guestshell は、NX-API へのローカル Unix ソケットを使用し、dohost を介してコマンド シェルにアクセスできます。
  1. 9.3(8) 以降の NX-OS の Guestshell において、NX-API ソケットへのアクセスは、root/管理者ユーザー権限でのみ許可されます。
  2. 9.3（8）以降の NX-OS の Guestshell において、NX-OS ファイルシステムへのアクセスは、root/管理者ユーザーだけが行います。
- Guestshell リリースでは、CVE が解決されるとマイナー番号が増分します。Guestshell は、CentOS が公開した場合にのみ CVE を更新します。

- **dnf update** を使用して、CentOS リポジトリからサードパーティのセキュリティ脆弱性修正を直接取得することをお勧めします。これにより、Cisco NX-OS ソフトウェアのアップデートを待つことなく、更新が利用可能になったときに入手できる柔軟性が得られます。

  または、**guestshell update** コマンドを使用すると、既存の Guestshell rootfs が置き換えられます。カスタマイズとソフトウェア パッケージのインストールは、この新しい Guestshell rootfs のコンテキスト内で再度実行する必要があります。

CentOS のサポート終了と Guestshell への影響

Guestshell は **CentOS 環境に基づく LXC コンテナ**です。 オープン ソース コミュニティの更新によると、CentOS 8 プロジェクトは 2021 年 12 月までにサポートが終了します。CentOS 7 プロジェクトは継続され、2024 年 6 月までにサポートが終了する予定です。CentOS 7 のこの長期サポートにより、最新の Cisco NX-OS ソフトウェア 10.2.x は Guestshell 2.11（CentOS 7 ベース）にパッケージ化されています。これは、10.1.x リリースのデフォルト環境である Guestshell 3.0（CentOS 8）を置き換えます。

Guestshell 2.11

Cisco NX-OS リリース 10.2(1) 以降、CentOS 7 がデフォルトの Guestshell 環境として再展開されました。理由の詳細については、「*CentOS* のサポート終了」セクションを参照してください。

Guestshell 2.11 には python2 および python3.6 のサポートが付属しています。Guestshell 2.11 と Guestshell 3.0 の間の機能は同じままです。

（注） Guestshell 2.11 の rootfs サイズは約 200 MB に増加しました。

Guestshell 3.0

Guestshell 3.0 は廃止されており、NX-OS 10.2.x からは利用できません。Guestshell 2.11 を使用することをお勧めします。ただし、10.2.x ソフトウェアは、Guestshell 3.0 コンテナおよび 10.1.x で動作している 3.0 Guestshell コンテナとの互換性を維持しています。

（注） Guestshell 3.0 の rootfs サイズは、Guestshell 2.0 の 170 MB に対して 220 MB です。

Guestshell 1.0 から Guestshell 2.x へのアップグレード

Guestshell 2.x は、CentOS 7 ルート ファイル システムに基づいています。コンテンツを Guestshell 1.0 にプルダウンした `.conf` ファイルまたはユーティリティのオフボックス リポジトリがある場合は、Guestshell 2.x で同じ展開手順を繰り返す必要があります。CentOS 7 の違いを考慮して、展開スクリプトを調整する必要がある場合があります。

Guestshell 3.0 を使用した Jacksonville リリースからの NX-OS のダウングレード

Cisco NX-OS リリース 10.1(1) 以降、Guestshell 3.0 サポートのインフラストラクチャ バージョンは 1.11 に引き上げられています（show virtual-service コマンドで確認してください）。したがって、Guestshell 3.0 OVA は以前のリリースでは使用できません。**Install all** コマンドを使用すると、バージョンの不一致が検証され、エラーがスローされます。Guestshell 3.0 を以前のリリースにダウングレードする前に、Guestshell 3.0 を破棄して、Guestshell 3.0 が以前のリリースで起動しないようにすることをお勧めします。

Guestshell 2.x

Cisco NX-OS は、十分なリソースをもつシステムのデフォルトで自動的に Guestshell のインストールおよび有効化を行います。ただし、Guestshell をサポートしない Cisco NX-OS イメージでデバイスがリロードされる場合、既存の Guestshell が自動的に削除され、`%VMAN-2-INVALID_PACKAGE` が発行されます。

（注） 4 GB の RAM を搭載したシステムでは、デフォルトでは Guestshell が有効になりません。**guestshell enable** コマンドを使用して、Guestshell をインストールして有効にします。

install all コマンドは、現在の Cisco NX-OS イメージとターゲットの Cisco NX-OS イメージとの互換性を検証します。

互換性のないイメージをインストールした場合の出力例を次に示します。

```
switch#
Installer will perform compatibility check first. Please wait.
uri is: /
2014 Aug 29 20:08:51 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE:
Successfully activated virtual service 'guestshell+'
Verifying image bootflash:/n9kpregs.bin for boot variable "nxos".
[####################] 100% -- SUCCESS
Verifying image type.
[####################] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[####################] 100% -- SUCCESS
Preparing "bios" version info using image bootflash:/.
[####################] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[####################] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[####################] 100% -- SUCCESS
Preparing "nxos" version info using image bootflash:/.
[####################] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[####################] 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
[####################] 100% -- SUCCESS
"Running-config contains configuration that is incompatible with the new image (strict
incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out which feature
needs to be disabled.".
Performing module support checks.
[####################] 100% -- SUCCESS
Notifying services about system upgrade.
[# ] 0% -- FAIL.
```

```
Return code 0x42DD0006 ((null)).
"Running-config contains configuration that is incompatible with the new image (strict
incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out
which feature needs to be disabled."
Service "vman" in vdc 1: Guestshell not supported, do 'guestshell destroy' to remove
it and then retry ISSU
Pre-upgrade check failed. Return code 0x42DD0006 ((null)).
switch#
```

（注） ベストプラクティスとして、Guestshellをサポートしていない古いCisco NX-OSイメージをリロードする前に、**guestshell destroy** コマンドを使用して Guestshell を削除します。

事前設定された SSHD サービス

Guestshell は、起動時に OpenSSH サーバーを開始します。サーバーは、localhost IP アドレス インターフェイス 127.0.0.1 でランダムに生成されたポートでのみリスンします。これにより、guestshell キーワードが入力されたときに、NX-OS 仮想シェルから Guestshell へのパスワードなしの接続が提供されます。このサーバーが強制終了されるか、その構成 (`/etc/ssh/sshd_config-cisco` にある) が変更された場合、NX-OS CLI からの Guestshellへのアクセスが機能しない可能性があります。

次の手順では、Guestshell 内で root として OpenSSh サーバーをインスタンス化します。

1. SSH 接続を確立するネットワーク名前空間または VRF を決定します。
2. OpenSSHがリッスンするポートを決定します。すでに使用されているポートを表示するには、NX-OS コマンドの **show socket connection** を使用します。

（注） パスワードなしのアクセス用の Guestshell sshd サービスは、17680 から 49150 までのランダム化されたポートを使用します。ポートの競合を避けるには、この範囲外のポートを選択してください。

次の手順では、OpenSSH サーバーを起動します。例では、IP アドレス 10.122.84.34:2222 で管理 netns の OpenSSH サーバーを起動します。

1. 次のファイルを作成します: `/usr/lib/systemd/systm/sshd-mgmt.service` および `/etc/ssh/sshd-mgmt_config`。ファイルには次の構成が必要です。

   ```
   -rw-r--r-- 1 root root 394 Apr 7 14:21 /usr/lib/systemd/system/sshd-mgmt.service
   -rw------- 1 root root 4478 Apr 7 14:22 /etc/ssh/sshd-mgmt_config
   ```

2. Unit と Service の内容を `/usr/lib/systemd/system/ssh.service` ファイルから `sshd-mgmt.service` にコピーします。

3. `sshd-mgmt.service` ファイルを次のように編集します。

   ```
   [Unit]
   Description=OpenSSH server daemon
   After=network.target sshd-keygen.service
   Wants=sshd-keygen.service
   ```

```
[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/sbin/ip netns exec management /usr/sbin/sshd -f /etc/ssh/sshd-mgmt_config
-D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
[Install]
WantedBy=multi-user.target
```

4. /etc/ssh/sshd-config の内容を /etc/ssh/sshd-mgmt_config にコピーします。必要に応じて、ListenAddress IP とポートを変更します。

```
Port 2222
ListenAddress 10.122.84.34
```

5. 次のコマンドを使用して、systemctl デーモンを開始します。

```
sudo systemctl daemon-reload
sudo systemctl start sshd-mgmt.service
sudo systemctl status sshd-mgmt.service -l
```

6. （オプション）構成を確認します。

```
ss -tnldp | grep 2222
```

7. Guestshell への SSH:

```
ssh -p 2222 guestshell@10.122.84.34
```

8. 複数の Guestshell またはスイッチの再起動にわたって構成を保存します。

```
sudo systemctl enable sshd-mgmt.service
```

9. パスワードなしの SSH/SCP およびリモート実行の場合、**ssh-keygen -t dsa** コマンドを使用して、SSH/SCP に使用するユーザー ID の公開鍵と秘密鍵を生成します。

   その後、キーは /.ssh ディレクトリの id_rsa および id_rsa.pub ファイルに保存されます。

```
[root@node01 ~]# cd ~/.ssh
[root@node02 .ssh]# ls -l
total 8
-rw-------. 1 root root 1675 May 5 15:01 id_rsa
-rw-r--r--. 1 root root 406 May 5 15:01 id_rsa.pub
```

10. 公開キーを SSH で接続するマシンにコピーし、アクセス許可を修正します。

```
cat id_rsa.pub >> /root/.ssh/authorized_keys
chmod 700 /root/.ssh
chmod 600 /root/.ssh/*
```

11. パスワードなしでリモート スイッチに SSH または SCP：

```
ssh -p <port#> userid@hostname [<remote command>]
scp -P <port#> userid@hostname/filepath /destination
```

Localtime

Guestshell は、ホスト システムと /etc/localtime を共有します。

（注） ホストと同じ localtime を共有したくない場合は、このシンボリック リンクを切断して、Guestshell 固有の /etc/localtime を作成できます。

```
switch(config)# clock timezone PDT -7 0
switch(config)# clock set 10:00:00 27 Jan 2017
Fri Jan 27 10:00:00 PDT 2017
switch(config)# show clock
10:00:07.554 PDT Fri Jan 27 2017
switch(config)# run guestshell
guestshell:~$ date
Fri Jan 27 10:00:12 PDT 2017
```

Guest Shell へのアクセス

Cisco NX-OS のデフォルトでは、network-admin ユーザーのみが Guest Shell にアクセスできます。これはシステムで自動的に有効になっており、**run guestshell** コマンドを使用してアクセスできます。**run bash** コマンドと一致して、これらのコマンドは、NX-OS CLI コマンドの **run guestshell** コマンド 形式を使用して Guest Shell 内で発行できます。

（注） Guest Shell は、4 GB を超える RAM を搭載したシステムで自動的に有効になります。

```
switch# run guestshell ls -al /bootflash/*.ova
-rw-rw-rw- 1 2002 503 83814400 Aug 21 18:04 /bootflash/pup.ova
-rw-rw-rw- 1 2002 503 40724480 Apr 15 2012 /bootflash/red.ova
```

（注） 2.2(0.2) 以降の Guest Shell は、スイッチにログインしているユーザーと同じユーザー アカウントを動的に作成します。ただし、他のすべての情報は、スイッチと Guest Shell のユーザー アカウント間で共有されません。

さらに、Guest Shell アカウントは自動的に削除されないため、不要になったときにネットワーク管理者が削除する必要があります。

ゲスト シェルに使用されるリソース

デフォルトでは、ゲスト シェルのリソースは、通常のスイッチ操作に使用できるリソースに小さな影響を与えます。ネットワーク管理者がゲスト シェルに追加のリソースを必要とする場合、**guestshell resize** {*cpu* | *memory* | *rootfs*} コマンドは、これらの制限を変更します

リソース	デフォルト	最小/最大
CPU	1 %	1/%

リソース	デフォルト	最小/最大
メモリ	400 MB	256/3840 MB
ストレージ	200 MB	200/2000 MB

CPU制限は、システム内の他のコンピューティング負荷との競合がある場合に、ゲストシェル内で実行されているタスクに与えられるシステムコンピューティングキャパシティのパーセンテージです。CPUリソースの競合がない場合、ゲストシェル内のタスクは制限されません。

（注） リソース割り当てを変更した後は、ゲストシェルの再起動が必要です。そのために、**guestshell reboot** コマンドを使用できます。

ゲストシェルの機能

Guestshell には、デフォルトで利用可能な多くのユーティリティと機能があります。

ゲストシェルは CentOS 7 Linux 環境であり、この流通向けにビルドされたソフトウェア パッケージを、yum インストールすることができます。Guestshell には、**net-tools**、**iproute**、**tcpdump** と OpenSSH などのネットワーキングデバイスで自然に期待される多くの一般的なツールが事前に入力されています。Guestshell 2.x の場合、追加の python パッケージをインストールするための PIP と同様に、python 2.7.5 がデフォルトで含まれています。Guestshell 2.11 では、デフォルトで python 3.6 も含まれています。

デフォルトでは、ゲストシェルは 64 ビットの実行スペースです。32 ビットのサポートが必要な場合は、glibc.i686 パッケージを yum でインストールできます。

Guestshell は、スイッチの管理ポートとデータ ポートを表すために使用される Linux ネットワーク インターフェイスにアクセスできます。**ifconfig** と **ethtool** などの典型的な Linux のメソッドとユーティリティは、カウンターの収集に使用できます。インターフェイスが NX-OS CLI で VRF に配置されると、Linux ネットワーク インターフェイスはその VRF のネットワーク名前空間に配置されます。名前空間は `/var/run/netns` で見ることができ、**ip netns** ユーティリティを使用してさまざまな名前空間のコンテキストで実行できます。いくつかのユーティリティ、**chvrf** と **vrfinfo** は、別の名前空間で実行し、プロセスが実行されている名前空間 /vrf に関する情報を取得するために提供されています。

systemd は、ゲストシェルを含む CentOS 8 環境でサービスを管理するために使用されます。

Guest Shell の NX-OS CLI

ゲスト シェルは、ユーザーがゲスト シェル環境からホスト ネットワーク要素に NX-OS コマンドを発行できるようにするアプリケーションを提供します。**dohost** アプリケーションは、有効な NX-OS 構成または exec コマンドを受け入れ、それらをホスト ネットワーク要素に発行します。

dohost コマンドを呼び出すときは、各NX-OSコマンドを一重引用符または二重引用符で囲むことができます：

```
dohost "<NXOS CLI>"
```

NX-OS CLI は連鎖させることができます：

```
[guestshell@guestshell ~]$ dohost "sh lldp time | in Hold" "show cdp global"
Holdtime in seconds: 120
Global CDP information:
CDP enabled globally
Refresh time is 21 seconds
Hold time is 180 seconds
CDPv2 advertisements is enabled
DeviceID TLV in System-Name(Default) Format
[guestshell@guestshell ~]$
```

NX-OS CLIは、各コマンドの間にセミコロンを追加することにより、NX-OSスタイルのコマンドチェーン技術を使用して一緒にチェーンすることもできます。（セミコロンの両側にスペースが必要です。）：

```
[guestshell@guestshell ~]$ dohost "conf t ; cdp timer 13 ; show run | inc cdp"
Enter configuration commands, one per line. End with CNTL/Z.
cdp timer 13
[guestshell@guestshell ~]$
```

（注） Guest Shell 2.2 (0.2) 以降を使用するリリース 7.0(3)I5(2) の場合、**dohost** コマンドを介してホストで発行されたコマンドは、ゲスト シェル ユーザの有効なロールに基づく特権で実行されます。

以前のバージョンのゲスト シェルは、ネットワーク管理者レベルの権限でコマンドを実行します。

NX-API への UDS 接続の数が最大許容数に達すると、**dohost** コマンドは機能不全になります。

Guest Shell でのネットワーク アクセス

NX-OS スイッチ ポートは、Guest Shell では Linux ネットワーク インターフェイスとして表されます。ifconfig または ethtool を使用して、/proc/net/dev の表示統計などの一般的な Linux メソッドはすべてサポートされています。

Guest Shell には、多くの一般的なネットワーク ユーティリティがデフォルトで含まれており、**chvrf** *vrf command* コマンドを使用してさまざまな VRF で使用できます。

```
[guestshell@guestshell bootflash]$ ifconfig Eth1-47
Eth1-47: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 13.0.0.47 netmask 255.255.255.0 broadcast 13.0.0.255
ether 54:7f:ee:8e:27:bc txqueuelen 100 (Ethernet)
RX packets 311442 bytes 21703008 (20.6 MiB)
RX errors 0 dropped 185 overruns 0 frame 0
TX packets 12967 bytes 3023575 (2.8 MiB)
```

```
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Guest Shell 内では、ネットワーク状態をモニタリングできますが、変更することはできません。ネットワーク状態を変更するには、ホストの bash シェルで NX-OS CLI または適切な Linux ユーティリティを使用します。

この **tcpdump** コマンドはGuest Shell にパッケージ化されており、管理ポートまたはスイッチ ポートでパントされたトラフィックのパケット トレースを可能にします。

この **sudo ip netns exec management ping** ユーティリティは、指定されたネットワーク名前空間のコンテキストでコマンドを実行するための一般的な方法です。これはGuest Shell 内で実行できます。

```
[guestshell@guestshell bootflash]$ sudo ip netns exec management ping 10.28.38.48
PING 10.28.38.48 (10.28.38.48) 56(84) bytes of data.
64 bytes from 10.28.38.48: icmp_seq=1 ttl=48 time=76.5 ms
```

chvrf ユーティリティは便宜のために提供されています。

```
guestshell@guestshell bootflash]$ chvrf management ping 10.28.38.48
PING 10.28.38.48 (10.28.38.48) 56(84) bytes of data.
64 bytes from 10.28.38.48: icmp_seq=1 ttl=48 time=76.5 ms
```

（注） コマンドなしで実行される **chvrf**コマンドは、現在の VRF / ネットワーク名前空間で実行されます。

たとえば、管理 VRF 経由で IP アドレス 10.0.0.1 を ping するには、コマンドは「**chvrf** *management ping 10.0.0.1*」です。 **scp** または **ssh** などの他のユーティリティも同様です。

例：

```
switch# guestshell
[guestshell@guestshell ~]$ cd /bootflash
[guestshell@guestshell bootflash]$ chvrf management scp foo@10.28.38.48:/foo/index.html
index.html
foo@10.28.38.48's password:
index.html 100% 1804 1.8KB/s 00:00
[guestshell@guestshell bootflash]$ ls -al index.html
-rw-r--r-- 1 guestshe users 1804 Sep 13 20:28 index.html
[guestshell@guestshell bootflash]$
[guestshell@guestshell bootflash]$ chvrf management curl cisco.com
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.cisco.com/">here</a>.</p>
</body></html>
[guestshell@guestshell bootflash]$
```

システム上の VRF のリストを取得するには、NX-OS からネイティブに、**show vrf** または **dohost** コマンドを介してコマンドを使用します。

例：

```
[guestshell@guestshell bootflash]$ dohost 'sh vrf'
VRF-Name   VRF-ID  State     Reason
default    1       Up        --
management 2       Up        --
red        6       Up        --
```

Guest Shell 内では、VRF に関連付けられたネットワーク名前空間が実際に使用されます。どのネットワーク名前空間が存在するかを確認する方が便利な場合があります。

```
[guestshell@guestshell bootflash]$ ls /var/run/netns
default management red
[guestshell@guestshell bootflash]$
```

Guest Shell 内からドメイン名を解決するには、リゾルバーを構成する必要があります。Guest Shell で /etc/resolv.conf ファイルを編集して、ネットワークに適した DNS ネームサーバとドメインを含めます。

例：

```
nameserver 10.1.1.1
domain cisco.com
```

ネームサーバーとドメインの情報は、NX-OS 構成で構成されたものと一致する必要があります。

例：

```
switch(config)# ip domain-name cisco.com
switch(config)# ip name-server 10.1.1.1
switch(config)# vrf context management
switch(config-vrf)# ip domain-name cisco.com
switch(config-vrf)# ip name-server 10.1.1.1
```

スイッチが HTTP プロキシ サーバーを使用するネットワーク内にある場合、**http_proxy** および **https_proxy** 環境変数も Guest Shell 内で設定する必要があります。

例：

```
export http_proxy=http://proxy.esl.cisco.com:8080
export https_proxy=http://proxy.esl.cisco.com:8080
```

これらの環境変数は、.bashrc ファイルまたは適切なスクリプトで設定して、永続的であることを確認する必要があります。

ゲスト シェルでのブートフラッシュへのアクセス

ネットワーク管理者は、NX-OS CLI コマンドの使用に加えて、Linux コマンドとユーティリティを使用してファイルを管理できます。ゲスト シェル環境の /bootflash にシステム ブートフラッシュをマウントすることにより、network-admin は Linux コマンドを使用してこれらのファイルを操作できます。

例：

```
find . -name "foo.txt"
rm "/bootflash/junk/foo.txt"
```

（注） ゲストシェル内のユーザーの名前はホストの場合と同じですが、ゲストシェルは別のユーザー名前空間にあり、uidはホスト上のユーザーの名前と一致しません。グループおよびその他のファイルのアクセス許可は、ゲストシェルユーザーがファイルに対して持つアクセスの種類を制御します。

Guest Shell の Python

Python はインタラクティブに使用できますが、python スクリプトをゲストシェルで実行することもできます。

例：

```
guestshell:~$ python
Python 2.7.5 (default, Jun 24 2015, 00:41:19)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
guestshell:~$
```

ネットワーク管理者が新しいPythonパッケージをインストールできるように、ゲストシェルには pip python パッケージ マネージャが含まれています。

例：

```
[guestshell@guestshell ~]$ sudo su
[root@guestshell guestshell]# pip install Markdown
Collecting Markdown
Downloading Markdown-2.6.2-py2.py3-none-any.whl (157kB)
100% |################################| 159kB 1.8MB/s
Installing collected packages: Markdown
Successfully installed Markdown-2.6.2
[root@guestshell guestshell]# pip list | grep Markdown
Markdown (2.6.2)
[root@guestshell guestshell]#
```

（注） **pip install** コマンドを入力する前に、**sudo su** コマンドを入力する必要があります。

Guestshell 2.11 の Python

Guestshell 2.11 には、Python 2 と Python 3.6 の両方がプリインストールされています。Python 2 または 3 をインストールするためにユーザーが必要とするアクションはありません。

```
[admin@guestshell ~]$ python
Python 2.7.5 (default, Nov 16 2020, 22:23:17)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>

[admin@guestshell ~]$ python3
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Guest Shell バージョン 2.10 までの Python 3（CentOS 7）

ゲスト シェル 2.X は、デフォルトで Python 3 がインストールされていない CentOS 7.1 環境を提供します。CentOS 7.1 に Python 3 をインストールするには、サードパーティのリポジトリを使用する、送信元からビルドするなど、複数の方法があります。別のオプションは、同じシステム内に複数のバージョンの Python のインストールをサポートする Red Hat Software Collections を使用することです。

Red Hat Software Collections（SCL）ツールをインストールするには:

1. scl-utils パッケージをインストールします。
2. CentOS SCL リポジトリを有効にして、提供されている Python 3 RPM のいずれかをインストールします。

```
[admin@guestshell ~]$ sudo su
[root@guestshell admin]# dnf install -y scl-utils | tail
Running transaction test
Transaction test succeeded
Running transaction
  Installing : scl-utils-20130529-19.el7.x86_64                            1/1
  Verifying  : scl-utils-20130529-19.el7.x86_64                            1/1

Installed:
  scl-utils.x86_64 0:20130529-19.el7

Complete!

[root@guestshell admin]# dnf install -y centos-release-scl | tail
  Verifying  : centos-release-scl-2-3.el7.centos.noarch                    1/2
  Verifying  : centos-release-scl-rh-2-3.el7.centos.noarch                 2/2

Installed:
  centos-release-scl.noarch 0:2-3.el7.centos

Dependency Installed:
  centos-release-scl-rh.noarch 0:2-3.el7.centos

Complete!

[root@guestshell admin]# dnf install -y rh-python36 | tail
warning: /var/cache/dnf/x86_64/7/centos-sclo-rh/packages/rh-python36-2.0-1.el7.x86_64.rpm:
 Header V4 RSA/SHA1 Signature, key ID f2ee9d55: NOKEY
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm:
[Errno 12] Timeout on
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm: (28,
 'Operation too slow. Less than 1000 bytes/sec transferred the last 30 seconds')
```

```
Trying other mirror.
Importing GPG key 0xF2EE9D55:
 Userid     : "CentOS SoftwareCollections SIG
(https://wiki.centos.org/SpecialInterestGroup/SCLo) <security@centos.org>"
 Fingerprint: c4db d535 b1fb ba14 f8ba 64a8 4eb8 4e71 f2ee 9d55
 Package    : centos-release-scl-rh-2-3.el7.centos.noarch (@extras)
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-SIG-SCLo
  rh-python36-python-libs.x86_64 0:3.6.9-2.el7
  rh-python36-python-pip.noarch 0:9.0.1-2.el7
  rh-python36-python-setuptools.noarch 0:36.5.0-1.el7
  rh-python36-python-virtualenv.noarch 0:15.1.0-2.el7
  rh-python36-runtime.x86_64 0:2.0-1.el7
  scl-utils-build.x86_64 0:20130529-19.el7
  xml-common.noarch 0:0.6.3-39.el7
  zip.x86_64 0:3.0-11.el7

Complete!
```

SCL を使用すると、Python 3 の環境変数を自動的に設定して、インタラクティブな bash セッションを作成できます。

（注） SCL Python インストールを使用するためにルート ユーザーは必要ありません。

```
[admin@guestshell ~]$ scl enable rh-python36 bash
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python SCL のインストールでは、pip ユーティリティも提供されます。

```
[admin@guestshell ~]$ pip3 install requests --user
Collecting requests
  Downloading
https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl
 (57kB)
    100% |################################| 61kB 211kB/s
Collecting idna<2.9,>=2.5 (from requests)
  Downloading
https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl
 (58kB)
    100% |################################| 61kB 279kB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Downloading
https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl
 (133kB)
    100% |################################| 143kB 441kB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading
https://files.pythonhosted.org/packages/b9/63/df50ac98ea0eb006c55a399c3bf1db9da7b5a24de7890bc9cfd5dd9e99/certifi-2019.11.28-py2.py3-none-any.whl
 (156kB)
    100% |################################| 163kB 447kB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests)
  Downloading
https://files.pythonhosted.org/packages/e8/74/6e4f91745020f967d09332bb2b8b9b10090957334692eb88ea4afe91b77f/urllib3-1.25.8-py2.py3-none-any.whl
 (125kB)
    100% |################################| 133kB 656kB/s
Installing collected packages: idna, chardet, certifi, urllib3, requests
Successfully installed certifi-2019.11.28 chardet-3.0.4 idna-2.8 requests-2.22.0
urllib3-1.25.8
```

```
You are using pip version 9.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("https://cisco.com")
<Response [200]>
```

デフォルトの Python 2 インストールは、SCL Python インストールと一緒に使用できます。

```
[admin@guestshell ~]$ which python3
/opt/rh/rh-python36/root/usr/bin/python3
[admin@guestshell ~]$ which python2
/bin/python2
[admin@guestshell ~]$ python2
Python 2.7.5 (default, Aug  7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello world!'
Hello world!
```

Software Collections を使用すると、同じ RPM の複数のバージョンをシステムにインストールできます。この場合、Python 3.6 に加えて Python 3.5 をインストールすることが可能です。

```
[admin@guestshell ~]$ sudo dnf install -y rh-python35 | tail
Dependency Installed:
  rh-python35-python.x86_64 0:3.5.1-13.el7
  rh-python35-python-devel.x86_64 0:3.5.1-13.el7
  rh-python35-python-libs.x86_64 0:3.5.1-13.el7
  rh-python35-python-pip.noarch 0:7.1.0-2.el7
  rh-python35-python-setuptools.noarch 0:18.0.1-2.el7
  rh-python35-python-virtualenv.noarch 0:13.1.2-2.el7
  rh-python35-runtime.x86_64 0:2.0-2.el7

Complete!

[admin@guestshell ~]$ scl enable rh-python35 python3
Python 3.5.1 (default, May 29 2019, 15:41:33)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

（注） 複数の Python バージョンが SCL にインストールされているときに新しいインタラクティブ bash セッションを作成すると、libpython 共有オブジェクト ファイルをロードできないという問題が発生する可能性があります。**source scl_source enable** *python-installation* コマンドを使用して、現在の bash セッションで環境を適切にセットアップできる回避策があります。

デフォルトの Guest Shell ストレージのキャパシティが、Python 3 をインストールするのに十分ではありません。**guestshell resize rootfs** *size-in-MB* コマンドを使用して、ファイル システムのサイズを増やします。通常、rootfs のサイズを 550 MB に設定すれば十分です。

Installing RPMs in the Guest Shell

The /etc/dnf.repos.d/CentOS-Base.repo file is set up to use the CentOS mirror list by default. Follow instructions in that file if changes are needed.

Dnf can be pointed to one or more repositories at any time by modifying the `yumrepo_x86_64.repo` file or by adding a new `.repo` file in the `repos.d` directory.

For applications to be installed inside Guest Shell 2.x, go to the CentOS 7 repo at http://mirror.centos.org/centos/7/os/x86_64/Packages/.

Dnf resolves the dependencies and installs all the required packages.

```
[guestshell@guestshell ~]$ sudo chvrf management dnf -y install glibc.i686
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: bay.uchicago.edu
* extras: pubmirrors.dal.corespace.com
* updates: mirrors.cmich.edu
Resolving Dependencies
"-->" Running transaction check
"--->" Package glibc.i686 0:2.17-78.el7 will be installed
"-->" Processing Dependency: libfreebl3.so(NSSRAWHASH_3.12.3) for package:
glibc-2.17-78.el7.i686
"-->" Processing Dependency: libfreebl3.so for package: glibc-2.17-78.el7.i686
"-->" Running transaction check
"--->" Package nss-softokn-freebl.i686 0:3.16.2.3-9.el7 will be installed
"-->" Finished Dependency Resolution

Dependencies Resolved

================================================================================
Package Arch Version Repository Size
================================================================================
Installing:
glibc i686 2.17-78.el7 base 4.2 M
Installing for dependencies:
nss-softokn-freebl i686 3.16.2.3-9.el7 base 187 k

Transaction Summary
================================================================================
Install 1 Package (+1 Dependent package)

Total download size: 4.4 M
Installed size: 15 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
(1/2): nss-softokn-freebl-3.16.2.3-9.el7.i686.rpm | 187 kB 00:00:25
(2/2): glibc-2.17-78.el7.i686.rpm | 4.2 MB 00:00:30
--------------------------------------------------------------------------------
Total 145 kB/s | 4.4 MB 00:00:30
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : nss-softokn-freebl-3.16.2.3-9.el7.i686 1/2
Installing : glibc-2.17-78.el7.i686 2/2
error: lua script failed: [string "%triggerin(glibc-common-2.17-78.el7.x86_64)"]:1: attempt
 to compare number with nil
Non-fatal "<"unknown">" scriptlet failure in rpm package glibc-2.17-78.el7.i686
Verifying : glibc-2.17-78.el7.i686 1/2
Verifying : nss-softokn-freebl-3.16.2.3-9.el7.i686 2/2

Installed:
glibc.i686 0:2.17-78.el7

Dependency Installed:
nss-softokn-freebl.i686 0:3.16.2.3-9.el7
```

```
Complete!
```

Note When more space is needed in the Guest Shell root file system for installing or running packages, the **guestshell resize roofs** *size-in-MB* command is used to increase the size of the file system.

Note Some open source software packages from the repository might not install or run as expected in the Guest Shell as a result of restrictions that have been put into place to protect the integrity of the host system.

ゲスト シェルのセキュリティ ポスチャ

スイッチでのゲスト シェルの使用は、ネットワーク管理者がシステムの機能を管理または拡張できる多くの方法の 1 つにすぎません。ゲスト シェルは、ネイティブ ホスト コンテキストから切り離された実行環境を提供することを目的としています。この分離により、ネイティブの実行環境と互換性がない可能性のあるソフトウェアをシステムに導入できます。また、システムの動作、パフォーマンス、またはスケールに影響を与えない環境でソフトウェアを実行することもできます。

[カーネル脆弱性パッチ（Kernel Vulnerability Patches）]

シスコは、既知の脆弱性に対処するプラットフォーム アップデートで、関連する Common Vulnerabilities and Exposures（CVE）に対応します。

[ASLR および X-Space のサポート（ASLR and X-Space Support）]

Cisco NX-OS は、ランタイム ディフェンスのためのアドレス空間 Layout Randomization（ASLR）と Executable Space Protection（X-Space）の使用をサポートしています。Cisco が署名したパッケージのソフトウェアは、この機能を利用します。システムに他のソフトウェアがインストールされている場合は、これらのテクノロジをサポートするホスト OS と開発ツールチェーンを使用して構築することをお勧めします。これにより、ソフトウェアが潜在的な侵入者に提示する潜在的な攻撃対象領域が減少します。

名前空間の分離

Guest Shell 環境は、さまざまな名前空間を使用してGuest Shell の実行スペースをホストの実行スペースから切り離す Linux コンテナー内で実行されます。NX-OS 9.2(1) リリース以降、Guest Shell は別のユーザー名前空間で実行され、Guest Shell 内でルートとして実行されているプロセスはホストのルートではないため、ホスト システムの整合性を保護するのに役立ちます。これらのプロセスは、uid マッピングのためにGuest Shell 内で uid 0 として実行されているように見えますが、

カーネルはこれらのプロセスの実際の uid を認識しており、適切なユーザー名前空間内の POSIX 機能を評価します。

ユーザーがホストからGuest Shell に入ると、Guest Shell 内に同じ名前のユーザーが作成されます。名前は一致しますが、Guest Shell 内のユーザーの uid は、ホストの uid と同じではありません。Guest Shell 内のユーザが共有メディア（たとえば、/bootflash または /volatile ）上のファイルに引き続きアクセスできるようにするために、ホストで使用される一般的な NX-OS gid（たとえば、 network-admin または network-operator）が Guest Shell にマッピングされます。その際に、値は同じになり、ユーザーの Guest Shell インスタンスが ホスト上のグループ メンバーシップに基づく適切なグループに関連付けられています。

例として、ユーザー bob について考えてみましょう。ホスト上で、bob には次の uid および gid メンバーシップがあります。

```
bash-4.3$ id
 uid=2004(bob) gid=503(network-admin) groups=503(network-admin),504(network-operator)
```

ユーザー bob がGuest Shell にある場合、ホストからのグループ メンバーシップがGuest Shell に設定されます。

```
[bob@guestshell ~]$ id
 uid=1002(bob) gid=503(network-admin)
groups=503(network-admin),504(network-operator),10(wheel)
```

ホスト Bash シェルとGuest Shell でユーザー bob によって作成されたファイルの所有者識別子 は異なります。以下の出力例は、Guest Shell 内から作成されたファイルの所有者識別子が、上記の出力例の 1002 ではなく 12002 であることを示しています。これは、ホスト Bash シェルから発行されたコマンドと、Guest Shell の識別子スペースが識別子 11000 で始まるためです。ファイルのグループ識別子は network-admin で、両方の環境で 503 です。

```
bash-4.3$ ls -ln /bootflash/bob_*
-rw-rw-r-- 1 12002 503 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 2004 503 4 Jun 22 15:47 /bootflash/bob_host


bash-4.3$ ls -l /bootflash/bob_*
-rw-rw-r-- 1 12002 network-admin 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 bob network-admin 4 Jun 22 15:47 /bootflash/bob_host
```

network-admin グループのファイル パーミッション設定と、bob がホスト シェルとGuest Shell の両方で network-admin のメンバーであるため、ユーザーはファイルにアクセスできます。

以下の出力例は、Guest Shell 環境内で、bob によってホストから作成されたファイルの所有者識別子が 65534 であることを示しています。これは、実際の識別子が、ユーザーの名前空間にマップされた識別子の範囲外の範囲にあることを示しています。マップされていない識別子は、この値として表示されます。

```
[bob@guestshell ~]$ ls -ln /bootflash/bob_*
-rw-rw-r-- 1 1002 503 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 65534 503 4 Jun 22 15:47 /bootflash/bob_host


[bob@guestshell ~]$ ls -l /bootflash/bob_*
-rw-rw-r-- 1 bob network-admin 4 Jun 22 15:47 /bootflash/bob_guestshell
```

```
-rw-rw-r-- 1 65534 network-admin 4 Jun 22 15:47 /bootflash/bob_host
```

ルートユーザーの制限

安全なコードを開発するためのベスト プラクティスとして、割り当てられたタスクを実行するために必要な最小限の特権でアプリケーションを実行することを推薦します。意図しないアクセスを防ぐために、Guest Shell に追加されたソフトウェアは、このベスト プラクティスに従う必要があります。

内のすべてのプロセスで、Guest Shell は Linux の機能が低下したことによる制限の対象となります。アプリケーションで root 権限を必要とする操作を実行する必要がある場合は、root アカウントの使用を、rootアクセスが絶対に必要な最小限の操作セットに制限し、そのモードでアプリケーションを実行できる時間のハード制限などの他の制御を課します。

Guest Shell が従う内のルートに対してドロップされる一連の Linux 機能は次のとおりです。

- cap_audit_control
- cap_audit_write
- cap_mac_admin
- cap_mac_override
- cap_mknod
- cap_net_broadcast
- cap_sys_boot
- cap_syslog
- cap_sys_module
- cap_sys_nice
- cap_sys_pacct
- cap_sys_ptrace
- cap_sys_rawio
- cap_sys_resource
- cap_sys_time
- cap_wake_alarm

net_admin機能は削除されませんが、ユーザー名前空間とネットワーク名前空間のホスト所有権により、Guest Shell ユーザーはインターフェイスの状態を変更できません。Guest Shell 内の root として、tmpfs と ramfs マウントだけでなくバインドマウントも使用できます。他のマウントは防止されます。

リソース管理

DDoS 攻撃は、攻撃対象のユーザがマシンやネットワーク 技術情報を使用できないようにする試みます。不適切な動作または悪意のあるアプリケーション コードは、接続帯域幅、ディスク容量、メモリ、およびその他のリソースの過剰消費の結果として DoS を引き起こす可能性があります。ホストは、ゲスト シェルとホスト上のサービス間のので技術情報を公平に割り当てる技術情報管理機能を提供します。

ゲスト ファイル システムのアクセス制限

ゲスト シェル内のファイルの完全性を維持するために、ゲスト シェルのファイル システムには NX-OS CLI からアクセスできません。

ゲスト シェルの管理

以下は、ゲスト シェルを管理するためのコマンドです。

表 *2:* ゲスト シェル *CLI* コマンド

コマンド	説明

コマンド	説明
guestshell enable {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• *[ゲストシェルOVAファイル（guest shell OVA file）]* 指定時： システムイメージに組み込まれているOVAを使用して、ゲストシェルをインストールしてアクティブ化します。 指定されたソフトウェアパッケージ（OVAファイル）またはシステムイメージからの組み込みパッケージ（パッケージが指定されていない場合）を使用して、ゲストシェルをインストールしてアクティブ化します。当初、ゲストシェルパッケージは、システムイメージに埋め込むことによってのみ利用できます。 ゲストシェルがすでにインストールされている場合、このコマンドはインストールされているゲストシェルを有効にします。通常、これは **guestshell disable** コマンドの後に使用されます。 • *rootfs-file-URI* が指定されている場合： ゲストシェルが破棄された状態のときに、ゲストシェル **rootfs** をインポートします。このコマンドは、指定されたパッケージでゲスト シェルを起動します。
guestshell export rootfs package *destination-file-URI*	ゲスト シェルの **rootfs** ファイルをローカル URI（ブートフラッシュ、USB1など）にエクスポートします。
guestshell disable	シャット ダウンとゲスト シェルの無効化

コマンド	説明
guestshell upgrade {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• *[*ゲストシェル*OVA*ファイル（*guest shell OVA file*）*]* 指定時： 指定されたソフトウェア パッケージ（OVA ファイル）またはシステム イメージからの組み込みパッケージ（パッケージが指定されていない場合）を使用して、ゲスト シェルを非アクティブ化してアップグレードします。当初、ゲスト シェル パッケージは、システム イメージに埋め込むことによってのみ利用できます。 ゲスト シェルの現在の rootfs は、ソフトウェア パッケージの rootfs に置き換えられます。ゲスト シェルは、アップグレード後も持続するセカンダリ ファイル システムを利用しません。永続的なセカンダリ ファイル システムがない場合、**guestshell destroy** コマンドに続けて**guestshell enable** コマンドを使用して rootfs を置き換えることもできます。アップグレードが成功すると、ゲスト シェルがアクティブ化されます。 アップグレード コマンドを実行する前に、確認を求めるプロンプトが表示されます。 • *rootfs-file-URI* が指定されている場合： ゲスト シェルがすでにインストールされている場合、ゲスト シェルの **rootfs** ファイルをインポートします。このコマンドは、既存のゲスト シェルを削除します。 指定されたパッケージにインストールします。

コマンド	説明
guestshell reboot	ゲストシェルを非アクティブ化してから、再度アクティブ化します。 リブートコマンドを実行する前に、確認を求めるプロンプトが表示されます。 （注） これは、exec モードで **guestshell disable** コマンドの後に **guestshell enable** コマンドが続くのと同じです。 これは、ゲスト シェル内のプロセスが停止しており、再起動する必要がある場合に役立ちます。この **run guestshell** コマンドは、ゲスト シェルで実行されている `sshd` に依存しています。 コマンドが機能しない場合は、`sshd` プロセスが誤って停止した可能性があります。NX-OS CLI からゲストシェルの再起動を実行すると、再起動してコマンドを復元できます。
guestshell destroy	ゲストシェルサービスを非アクティブ化して、アンインストールします。ゲストシェルに関連付けられているすべての技術情報がシステムに返されます。この **show virtual-service global** コマンドは、これらの技術情報がいつ利用可能になるかを示します。 このコマンドを発行すると、destroy コマンドを実行する前に確認を求めるプロンプトが表示されます。
guestshell **run guestshell**	シェルプロンプトですでに実行されているゲスト シェルに接続します。必要なユーザー名 / パスワード
guestshell run *command* **run guestshell** *command*	ゲスト シェル環境のコンテキスト内で Linux / UNIX コマンドを実行します。 コマンドの実行後、スイッチプロンプトに戻ります。

コマンド	説明
guestshell resize[**cpu** |**memory** |**rootfs**]	ゲスト シェルに割り当てられた使用可能な技術情報を変更します。変更は、次にゲスト シェルが有効化または再起動されたときに有効になります。 （注） サイズ変更の値は、**guestshell destroy** コマンドを使用するとクリアされます。
guestshell sync	アクティブ スーパーバイザとスタンバイ スーパーバイザがあるシステムでは、このコマンドはゲスト シェルの格納ファイルをアクティブ スーパーバイザからスタンバイ スーパーバイザに同期します。network-admin は、スタンバイ スーパーバイザが現用系スーパーバイザになったときに同じ rootfs を使用するようにゲスト シェル rootfs が設定されているときに、このコマンドを発行します。このコマンドを使用しない場合、スタンバイ スーパーバイザがそのスーパーバイザで利用可能なゲスト シェル パッケージを使用してアクティブ ロールに移行するときに、ゲスト シェルが新たにインストールされます。
virtual-service reset force	ゲストシェルまたは仮想サービスを管理できない場合は、システムのリロード後でも、reset コマンドを使用してゲスト シェルとすべての仮想サービスを強制的に削除します。クリーンアップを実行するには、システムを再ロードする必要があります。このコマンドを発行した後は、システムがリロードされるまで、ゲスト シェルまたは追加の仮想サービスをインストールまたは有効にすることはできません。 リセットを開始する前に確認を求められます。

（注） ゲスト シェル環境を有効化 / 無効化し、アクセスするには、管理者権限が必要です。

（注） ゲスト シェルは、ホスト システム上の Linux コンテナ（LXC）として導入されます。NX-OS デバイスでは、LXC は virtual-service コマンドでインストールと管理されます。ゲスト シェルは、virtual-service コマンドに `guestshell+` という名前の仮想サービスとして表示されます。

（注） ゲスト シェルに関係のない仮想サービス コマンドは廃止されます。これらのコマンドは NX-OS 9.2（1）リリースでは非表示になっており、将来のリリースでは削除されます。

次の exec キーワードは廃止予定です。

```
# virtual-service ?
connect  Request a virtual service shell
install  Add a virtual service to install database
uninstall  Remove a virtual service from the install database
upgrade  Upgrade a virtual service package to a different version


# show virtual-service ?
detail  Detailed information config)
```

次の構成キーワードは廃止されます。

```
(config) virtual-service ?
WORD  Virtual service name (Max Size 20)


(config-virt-serv)# ?
activate  Activate configured virtual service
description  Virtual service description
```

Guest Shell の無効化

guestshell disable コマンドはシャットダウンして、Guest Shell を無効化します。

Guest Shell が無効化された状態でシステムをリロードすると、Guest Shell は無効化されたままになります。

例：

```
switch# show virtual-service list
Virtual Service List:
Name                   Status          Package Name
------------------------------------------------------------
guestshell+            Activated        guestshe11.ova
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want
to disable the guest shell? (y/n) [n) y

2014 Jul 30 19:47:23 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating virtual
service 'guestshell+'
2014 Jul 30 18:47:29 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated
 virtual service 'guestshell+'
switch# show virtual-service list
Virtual Service List:
Name                   Status                   Package Name
guestshell+            Deactivated              guestshell.ova
```

（注） **guestshell enable** コマンドで Guest Shell が再アクティブ化されます。

ゲスト シェルの破棄

guestshell destroy コマンドは、ゲスト シェルとそのアーティファクトをアンインストールします。このコマンドでは、ゲスト シェル OVA は削除されません。

ゲスト シェルが破棄された状態でシステムをリロードすると、ゲスト シェルは破棄されたままになります。

```
switch# show virtual-service list
Virtual Service List:
Name                 Status          Package Name
------------------------------------------------
guestshell+          Deactivated     guestshell.ova

switch# guestshell destroy

You are about to destroy the guest shell and all of its contents. Be sure to save your work.
 Are you sure you want to continue? (y/n) [n] y
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Destroying virtual service
 'guestshell+'
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Successfully destroyed
virtual service 'guestshell +'

switch# show virtual-service list
Virtual Service List:
```

（注） **guestshell enable** コマンドを使用して、ゲスト シェルを再度有効にすることができます。

（注） Cisco NX-OS ソフトウェアでは、コンテナがインストールされると、**oneP** 機能がローカル アクセスに対して自動的に有効になります。ゲスト シェルはコンテナであるため、**oneP** 機能が自動的に開始されます。

ゲスト シェルを使用しない場合は、**guestshell destroy** コマンドで削除できます。ゲスト シェルが削除されると、その後のリロードのために削除されたままになります。つまり、ゲスト シェル コンテナが削除され、スイッチが再ロードされても、ゲスト シェル コンテナは自動的に開始されません。

Guest Shell の有効化

この **guestshell enable** コマンドは、Guest Shell ソフトウェア パッケージから Guest Shell をインストールします。デフォルトでは、システム イメージに埋め込まれたパッケージがインストールに

使用されます。Guest Shell が無効化されている場合は、このコマンドを使用して、Guest Shell を再アクティブ化することもできます。

Guest Shell が有効化された状態でシステムをリロードすると、Guest Shell は有効化されたままになります。

例：

```
switch# show virtual-service list
Virtual Service List:
switch# guestshell enable
2014 Jul 30 18:50:27 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual service
'guestshell+'
2014 Jul 30 18;50;42 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating

2014 Jul 30 18:50:42 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
 'guestshell+'
2014 Jul 30 18:51:16 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'

switch# show virtual-service list
Virtual Service List:
Name                   Status             Package Name
guestshell+            Activated          guestshell.ova
```

ベース ブート モードでの Guest Shell の有効化

NX-OS 9.2(1) リリース以降、システムを *[*基本ブート モード（*base boot mode*）*]* でブートすることを選択できます。 システムを基本ブート モードで起動すると、Guest Shell はデフォルトでは開始されません。このモードで Guest Shell を使用するには、仮想化インフラストラクチャと Guest Shell イメージを含む RPM をアクティブにする必要があります。これを行うと、Guest Shell と virtual-service コマンドが使用できるようになります。

RPM アクティベーション コマンドが次の順序で実行された場合：

1. `install activate guestshell`
2. `install activate virtualization`

Guest Shell コンテナは、システムがフル モードで起動した場合と同様に自動的にアクティブ化されます。

RPM アクティベーション コマンドを逆の順序で実行した場合：

1. `install activate virtualization`
2. `install activate guestshell`

その後、**[guestshell を有効化（guestshell enable）]** コマンドを実行するまで、Guest Shell は有効になりません。

ゲスト シェルの複製

Cisco NX-OS リリース 7.0（3）I7（1）以降、1 つのスイッチでカスタマイズされたゲスト シェル **rootfs** を複数のスイッチに展開できます。

アプローチは、ゲスト シェル **rootfs** をカスタマイズしてからエクスポートし、ファイル サーバに保存することです。POAP スクリプトは、ゲスト シェル **rootfs** を他のスイッチにダウンロード(インポート) し、特定のゲスト シェルを多数のデバイスに同時にインストールできます。

ゲスト シェル rootfs のエクスポート

ゲスト シェル **rootfs** をエクスポートするには、**guestshell export rootfs package***destination-file-URI* コマンドを使用します。

destination-file-URI パラメータは、ゲスト シェル **rootfs** のコピー先のファイルの名前です。このファイルでは、ローカル URI オプション（ブートフラッシュ、USB1 など）が可能です。

guestshell export rootfs package コマンドでは、次の処理が行われます。

- ゲスト シェルを無効にします（すでに有効になっている場合）。
- ゲスト シェル インポート YAML ファイルを作成し、**rootfs** ext4 ファイルの /cisco ディレクトリに挿入します。
- **rootfs** ext4 ファイルをターゲット URI の場所にコピーします。
- ゲスト シェルが以前に有効になっていた場合は、再度有効にします。

Guest Shell rootfs のインポート

Guest Shell **rootfs** をインポートする場合、考慮すべき 2 つの状況があります。

- Guest Shell が破棄された状態の場合は、 **guestshell enable package** *rootfs-file-URI* コマンドを使用して、Guest Shell **rootfs** をインポートします。このコマンドは、指定されたパッケージで Guest Shell を起動します。
- Guest Shell がすでにインストールされている場合は、 **guestshell upgrade package** *rootfs-file-URI* コマンドを使用して、Guest Shell **rootfs** をインポートします。このコマンドは、既存のGuest Shell を削除し、指定されたパッケージをインストールします。

rootfs-file-URI パラメータは、ローカル ストレージ（ブートフラッシュ、USB など）に保存されている **rootfs** ファイルです。

ブートフラッシュにあるファイルでこのコマンドを実行すると、ファイルはブートフラッシュのストレージ プールに移動されます。

ベスト プラクティスとして、 **guestshell upgrade package** *rootfs-file-URI* コマンドを使用する前に、ファイルをブートフラッシュにコピーし、md5sum を検証する必要があります。

（注） **guestshell upgrade package** *rootfs-file-URI* コマンドは、Guest Shell 内から実行

（注） rootfs ファイルはシスコの署名付きパッケージではありません。例に示すように、有効にする前に、署名されていないパッケージを許可するように設定する必要があります。

```
(config-virt-serv-global)# signing level unsigned
Note: Support for unsigned packages has been user-enabled. Unsigned packages are not endorsed
 by Cisco. User assumes all responsibility.
```

（注） rootfs の組み込みバージョンを復元するには：

- Guest Shell が既にインストールされている場合は、 **guestshell upgrade** コマンドを（追加のパラメーターなしで）使用します。
- Guest Shell が破棄されたときに、 **guestshell enable** コマンドを（追加パラメータなしで）使用します。

（注） Guest Shell 内から、または NX-API を使用してスイッチの外部からこのコマンドを実行する場合は、プロンプトをスキップするように設定する必要があります。**terminal dont-ask**

guestshell enable package *rootfs-file-URI* コマンド：

- **rootfs** ファイルの基本的な検証を実行します。
- **rootfs** をストレージ プールに移動します。
- **rootfs** をマウントして、/cisco ディレクトリから YAML ファイルを抽出します。
- YAML ファイルを解析して VM 定義（リソース要件を含む）を取得します。
- Guest Shell をアクティブにします。

guestshell enable のワークフローの例：

```
switch# copy scp://user@10.1.1.1/my_storage/gs_rootfs.ext4 bootflash: vrf management
switch# guestshell resize cpu 8
Note: System CPU share will be resized on Guest shell enable
switch# guestshell enable package bootflash:gs_rootfs.ext4
Validating the provided rootfs
switch# 2017 Jul 31 14:58:01 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual
service 'guestshell+'
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
```

```
 'guestshell+'
2017 Jul 31 14:58:33 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```

（注） **guestshell upgrade** のワークフローの前に、既存の Guest Shell が破棄されます。

（注） サイズ変更の値は、 **guestshell upgrade** コマンドを使用するとクリアされます。

YAML ファイルのインポート

Guest Shell のユーザーが変更可能ないくつかの特性を定義する YAML ファイルは、エクスポート操作の一部として自動的に作成されます。これは、/cisco ディレクトリの Guest Shell **rootfs** に組み込まれています。これは、Guest Shell コンテナの完全な記述子ではありません。ユーザーが変更できるパラメータの一部のみが含まれています。

Guest Shell インポート YAML ファイルの例：

```
---
import-schema-version: "1.0"
info:
  name: "GuestShell"
  version: "2.2(0.3)"
  description: "Exported GuestShell: 20170216T175137Z"
app:
  apptype: "lxc"
  cpuarch: "x86_64"
  resources:
    cpu: 3
    memory: 307200
    disk:
    - target-dir: "/"
      capacity: 250
...
```

guestshell export rootfs package コマンドを実行すると、YAML ファイルが生成されます。このファイルは、現在実行中の Guest Shell の値をキャプチャします。

情報セクションには、Guest Shell の識別に役立つ非運用データが含まれています。**show guestshell detail** コマンドの出力に一部の情報が表示されます。

説明の値は、YAML ファイルが作成されたときの UTC 時間のエンコーディングです。時刻文字列のフォーマットは、RFC5545（iCal）の DTSTAMP と同じです。

リソース セクションでは、Guest Shell をホストするために必要な情報技術について説明します。この例の target-dir の値「/」は、ディスクを **rootfs** として識別します。

（注） Guest Shell が破棄されたときにサイズ変更された値が指定された場合、 **guestshell enable package** コマンドの使用時にそれらの値がインポート YAML ファイルの値よりも優先されます。

cpuarch 値は、コンテナの実行が予想される CPU アーキテクチャを示します。

エクスポート操作が完了した後、YAML ファイルを変更できます（説明などを変更したり、必要に応じて技術情報パラメータを増やしたりできます）。

Cisco は、JSON スキーマを使用して変更された YAML ファイルを検証するために実行できる Python スクリプトを提供しています。完全なテストではありませんが（たとえば、デバイス固有のリソース制限はチェックされません）、一般的なエラーにフラグを付けることができます。例を含む Python スクリプトは、[Guest Shell インポート エクスポート（Guest Shell Import Export）]https://github.com/datacenter/opennxos/tree/master/guestshell_import_export にあります。 次の JSON ファイルは、Guest Shell インポート YAML のバージョン 1.0 のスキーマを記述しています。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Guest Shell import schema",
  "description": "Schema for Guest Shell import descriptor file - ver 1.0",
  "copyright": "2017 by Cisco systems, Inc. All rights reserved.",
  "id": "",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "import-schema-version": {
      "id": "/import-schema-version",
      "type": "string",
      "minLength": 1,
      "maxLength": 20,
      "enum": [
          "1.0"
      ]
    },
    "info": {
      "id": "/info",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "id": "/info/name",
          "type": "string",
          "minLength": 1,
          "maxLength": 29
        },
        "description": {
          "id": "/info/description",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        },
        "version": {
          "id": "/info/version",
          "type": "string",
          "minLength": 1,
          "maxLength": 63
        },
        "author-name": {
          "id": "/info/author-name",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        },
        "author-link": {
          "id": "/info/author-link",
```

```
      "type": "string",
      "minLength": 1,
      "maxLength": 199
    }
  }
},
"app": {
  "id": "/app",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "apptype": {
      "id": "/app/apptype",
      "type": "string",
      "minLength": 1,
      "maxLength": 63,
      "enum": [
        "lxc"
      ]
    },
    "cpuarch": {
      "id": "/app/cpuarch",
      "type": "string",
      "minLength": 1,
      "maxLength": 63,
      "enum": [
        "x86_64"
      ]
    },
    "resources": {
      "id": "/app/resources",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "cpu": {
          "id": "/app/resources/cpu",
          "type": "integer",
          "multipleOf": 1,
          "maximum": 100,
          "minimum": 1
        },
        "memory": {
          "id": "/app/resources/memory",
          "type": "integer",
          "multipleOf": 1024,
          "minimum": 1024
        },
        "disk": {
          "id": "/app/resources/disk",
          "type": "array",
          "minItems": 1,
          "maxItems": 1,
          "uniqueItems": true,
          "items": {
            "id": "/app/resources/disk/0",
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "target-dir": {
                "id": "/app/resources/disk/0/target-dir",
                "type": "string",
                "minLength": 1,
                "maxLength": 1,
                "enum": [
```

```
                    "/"
                  ]
                },
                "file": {
                  "id": "/app/resources/disk/0/file",
                  "type": "string",
                  "minLength": 1,
                  "maxLength": 63
                },
                "capacity": {
                  "id": "/app/resources/disk/0/capacity",
                  "type": "integer",
                    "multipleOf": 1,
                    "minimum": 1
                }
              }
            }
          }
        },
        "required": [
          "memory",
          "disk"
        ]
      }
    },
    "required": [
      "apptype",
      "cpuarch",
      "resources"
    ]
  }
},
"required": [
  "app"
]
}
```

show guestshell コマンド

show guestshell detail コマンドの出力には、ゲスト シェルがインポートされたか、OVA からインストールされたかを示す情報が含まれます。

rootfsをインポートした後の **show guestshell detail** コマンドの例。

```
switch# show guestshell detail
Virtual service guestshell+ detail
  State                 : Activated
  Package information
    Name                : rootfs_puppet
    Path                : usb2:/rootfs_puppet
    Application
      Name              : GuestShell
      Installed version : 3.0(0.0)
      Description       : Exported GuestShell: 20170613T173648Z
    Signing
      Key type          : Unsigned
      Method            : Unknown
    Licensing
      Name              : None
      Version           : None
```

仮想サービスと Guest Shell 情報の検証

次のコマンドを使用して、仮想サービスとゲスト シェルの情報を検証できます。

コマンド	説明

show virtual-service global
`switch# show virtual-service global`
 	
`Virtual Service Global State and Virtualization Limits:`
 	
`Infrastructure version : 1.11`
`Total virtual services installed : 1`
`Total virtual services activated : 1`
 	
`Machine types supported : LXC`
`Machine types disabled : KVM`
 	
`Maximum VCPUs per virtual service : 1`
 	
`Resource virtualization limits:`
`Name Quota Committed Available`
`---------------------------------------------------------------------`
`system CPU (%) 20 1 19`
`memory (MB) 3840 256 3584`
`bootflash (MB) 8192 200 7992`
`switch#`
	仮想サービスのグローバル状態と制限を表示します。

show virtual-service list
 	
`switch# show virtual-service list *`
 	
`Virtual Service List:`
 	
`Name Status Package Name`
`---------------------------------------------------------------------`
`guestshell+ Activated guestshell.ova`
	仮想サービスの概要、仮想サービスのステータス、およびインストールされているソフトウェアパッケージを表示します。

コマンド	説明

show guestshell detail
 	
`switch# show guestshell detail`
`Virtual service guestshell+ detail`
`  State                 : Activated`
`  Package information`
`    Name                : guestshell.ova`
`    Path                : /isan/bin/guestshell.ova`
`    Application`
`      Name              : GuestShell`
`      Installed version : 3.0(0.0)`
`      Description       : Cisco Systems Guest Shell`
`    Signing`
`      Key type          : Cisco key`
`      Method            : SHA-1`
`    Licensing`
`      Name              : None`
`      Version           : None`
`  Resource reservation`
`    Disk                : 400 MB`
`    Memory              : 256 MB`
`    CPU                 : 1% system CPU`
 	
`  Attached devices`
`    Type              Name        Alias`
`    ---------------------------------------------`
`    Disk              _rootfs`
`    Disk              /cisco/core`
`    Serial/shell`
`    Serial/aux`
`    Serial/Syslog                 serial2`
`    Serial/Trace                  serial3`
	guestshellパッケージに関する詳細（バージョン、署名リソース、デバイスなど）を表示します。

ゲスト シェルからのアプリケーションの永続的な起動

アプリケーションには、 /usr/lib/systemd/system/*application_name*.service にインストールされる systemd / systemctl サービス ファイルが必要です。 このサービス ファイルは、次の一般的なフォーマットにする必要があります。

```
[Unit]
Description=Put a short description of your application here

[Service]
ExecStart=Put the command to start your application here
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

（注） 特定のユーザーとして systemd を実行するには、サービスの [サービス（Service）] セクションに User=<username> を追加します。

Guest Shell からアプリケーションを永続的に起動する手順

Step 1 上記で作成したアプリケーション サービス ファイルを /usr/lib/systemd/system/*application_name* にインストールします。サービス

Step 2 **systemctl start** *application_name* でアプリケーションを開始します

Step 3 アプリケーションが **systemctl status** -l *application_name* で実行されていることを確認します

Step 4 **systemctl enable** *application_name* でリロード時にアプリケーションを再起動できるようにします

Step 5 アプリケーションが **systemctl status** -l *application_name* で実行されていることを確認します

ゲスト シェルでのサンプル アプリケーション

次の例は、ゲスト シェルのアプリケーションを示しています。

```
root@guestshell guestshell]# cat /etc/init.d/hello.sh
#!/bin/bash

OUTPUTFILE=/tmp/hello

rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
[root@guestshell guestshell]#
[root@guestshell guestshell]#
[root@guestshell system]# cat /usr/lib/systemd/system/hello.service
[Unit]
Description=Trivial "hello world" example daemon

[Service]
ExecStart=/etc/init.d/hello.sh &
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
[root@guestshell system]#
[root@guestshell system]# systemctl start hello
```

```
[root@guestshell system]# systemctl enable hello
[root@guestshell system]# systemctl status -l hello
hello.service - Trivial "hello world" example daemon
   Loaded: loaded (/usr/lib/systemd/system/hello.service; enabled)
   Active: active (running) since Sun 2015-09-27 18:31:51 UTC; 10s ago
 Main PID: 355 (hello.sh)
   CGroup: /system.slice/hello.service
           ##355 /bin/bash /etc/init.d/hello.sh &
           ##367 sleep 10

Sep 27 18:31:51 guestshell hello.sh[355]: Executing: /etc/init.d/hello.sh &
[root@guestshell system]#
[root@guestshell guestshell]# exit
exit
[guestshell@guestshell ~]$ exit
logout
switch# reload
This command will reboot the system. (y/n)?  [n] y
```

リロード後

```
[root@guestshell guestshell]# ps -ef | grep hello
root        20     1  0 18:37 ?        00:00:00 /bin/bash /etc/init.d/hello.sh &
root       123   108  0 18:38 pts/4    00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
[root@guestshell guestshell]# cat /tmp/hello
Sun Sep 27 18:38:03 UTC 2015
Hello World
Sun Sep 27 18:38:13 UTC 2015
Hello World
Sun Sep 27 18:38:23 UTC 2015
Hello World
Sun Sep 27 18:38:33 UTC 2015
Hello World
Sun Sep 27 18:38:43 UTC 2015
Hello World
[root@guestshell guestshell]#
```

systemd / systemctl で実行すると、アプリケーションが停止した場合（または強制終了した場合）、アプリケーションは自動的に再起動されます。プロセス 識別子 はもともと 226 です。アプリケーションを強制終了すると、プロセス 識別子 257 で自動的に再起動されます。

```
[root@guestshell guestshell]# ps -ef | grep hello
root       226     1  0 19:02 ?        00:00:00 /bin/bash /etc/init.d/hello.sh &
root       254   116  0 19:03 pts/4    00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
[root@guestshell guestshell]# kill -9 226
[root@guestshell guestshell]#
[root@guestshell guestshell]# ps -ef | grep hello
root       257     1  0 19:03 ?        00:00:00 /bin/bash /etc/init.d/hello.sh &
root       264   116  0 19:03 pts/4    00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
```

Guest Shell に関する問題のトラブルシューティング

7.0（3）I7 へのダウングレード後にゲスト シェルにアクセスできない

ゲスト シェルのアクティブ化または非アクティブ化のプロセス中に、NX-OS 9.2（1）リリースから NX-OS 7.0（3）7 リリース イメージ（ユーザー名前空間のサポートがない）にダウングレード

した場合、次のコマンドを実行できます。ゲストシェルは起動しますが、ゲストシェルにアクセスできない次の状態になります。この問題の理由は、ゲスト シェルの移行中にリロードが発行された場合、ゲスト シェル内のファイルがユーザー名前空間のサポートがない NX-OS リリースで使用可能な識別子範囲に戻されないためです。

```
switch# guestshell
Failed to mkdir .ssh for admin
admin RSA add failed
ERROR: Failed to connect with Virtual-service 'guestshell+'
switch#
switch# sh virt list

Virtual Service List:
Name           Status       Package Name
------------------------------------------------------------------------
guestshell+    Activated    guestshell.ova

switch# run bash ls -al /isan/vdc_1/virtual-instance/guestshell+/rootfs/
drwxr-xr-x  24  11000  11000  1024  Apr 11  10:44  .
drwxrwxrwx   4  root   root     80  Apr 27  20:08  ..
-rw-r--r--   1  11000  11000     0  Mar 21  16:24  .autorelabel
lrwxrwxrwx   1  11000  11000     7  Mar 21  16:24  bin -> usr/bin
```

ゲスト シェルの格納ファイルを失うことなくこの問題から回復するには、以前に実行されていた NX-OS 9.2（x）イメージを使用してシステムをリロードし、NX-OS 7.0（3）I7 イメージでシステムをリロードする前に、ゲスト シェルが[アクティブ化（Activated）]された状態になるようにします。もう 1 つのオプションは、NX-OS 9.2（x）の実行中にゲスト シェルを無効にし、7.0（3）I7 でリロードした後に再度有効にすることです。

ゲスト シェルに保存するものがなく、復元するだけの場合は、イメージを変更せずに破棄して再作成できます。

ゲスト シェルのルートからブートフラッシュのファイルにアクセスできない

ゲスト シェルのルートからブートフラッシュのファイルにアクセスできない場合があります。

ホストから：

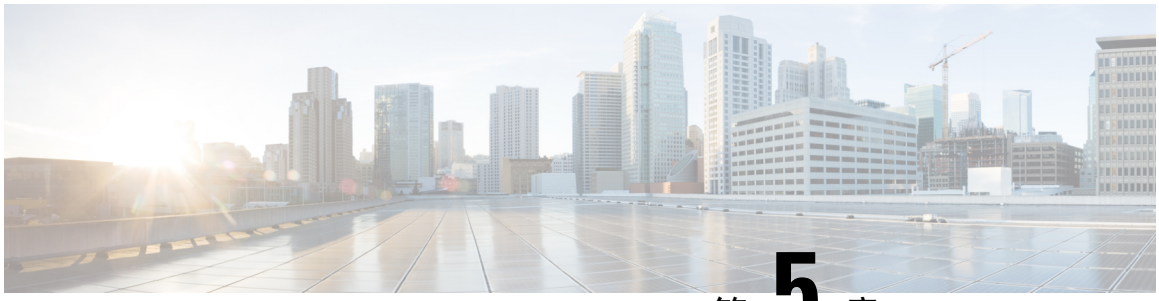
```
root@switch# ls -al /bootflash/try.that
-rw-r--r--  1  root  root  0  Apr 27  20:55  /bootflash/try.that
root@switch#
```

ゲスト シェルから：

```
[root@guestshellbootflash]# ls -al /bootflash/try.that
-rw-r--r--  1  65534  host-root  0  Apr 27  20:55  /bootflash/try.that
[root@guestshellbootflash]# echo "some text" >> /bootflash/try.that
-bash: /bootflash/try.that: Permission denied
[root@guestshellbootflash]#
```

これは、ユーザーの名前空間がホスト システムを保護するために使用されているため、ゲスト シェルのルートが実際にはシステムのルートではないことが原因である可能性があります。

この問題から回復するには、ファイルのアクセス許可とファイルのグループ識別子で、ブートフラッシュ上の共有ファイルに期待どおりにアクセスできることを確認します。ホスト Bash セッションからアクセス許可またはグループ 識別子 を変更する必要がある場合があります。



第 5 章

Broadcom シェル

Broadcom シェルについて

スイッチの前面パネルおよびファブリック モジュール ライン カードには、Broadcom のネットワーク転送エンジン（NFE）が搭載されています。NFE の数は、前面パネル ライン カード（LC）またはファブリック モジュール（FM）のモデルによって異なります。

注意事項と制約事項

T2 ASIC の情報は、制限なくアクセスして読み取ることができます。ただし、シスコは T2 の構成設定の変更を推奨していません。Broadcom シェルにアクセスする場合は注意してください。

Broadcom シェル（bcm-shell）へのアクセス

次のセクションでは、Broadcom シェル（bcm-shell）にアクセスする方法について説明します。

CLI API を使用した bcm-shell へのアクセス

bcm-shell コマンドは、Cisco NX-OS CLI から特定の T2 ASIC インスタンスに直接渡されます。T2 ASIC インスタンスは、ファブリック モジュールまたは前面パネル ライン カード上に配置できます。

コマンドの構文は、次のとおりです。

bcm-shell module *module_number* [*instance_number:command*]

場所

module_number	シャーシのモジュール番号。
instance_number	T2 インスタンス番号 • 指定しなかった場合、T2 インスタンス番号はデフォルトで 0 に設定されます。 • ワイルドカード（「*」）を指定すると、すべての T2 インスタンスが処理されます。
command	Broadcom コマンド

（注） 「pipe include」や「redirect output to file」などの Cisco NX-OS コマンド拡張を使用して、コマンド出力を管理できます。

（注） CLI API を使用してコマンドを入力すると、監査目的でシステム アカウンティング ログに記録されます。bcm-shell から直接入力されたコマンドは、アカウンティング ログに記録されません。

ファブリック モジュールのネイティブ bcm-shell へのアクセス

8 スロット ライン カード（LC）シャーシは、最大 6 つのファブリック モジュール（FM）をホストできます。これらのスロットには 21 ～ 26 の番号が付けられており、bcm-shell にアクセスする FM を指定する必要があります。

次に、スロット 24 の FM の bcm シェルにアクセスし、コンテキスト ヘルプにアクセスし、bcm シェルを終了する例を示します。

- **show module** コマンドを使用して FM を表示します。

```
switch# show module
Mod Ports Module-Type Model Status
--- ----- ----------------------------------- ------------------ ----------
3 36 36p 40G Ethernet Module N9k-X9636PQ ok
4 36 36p 40G Ethernet Module N9k-X9636PQ ok
21 0 Fabric Module Nexus-C9508-FM ok
22 0 Fabric Module Nexus-C9508-FM ok
23 0 Fabric Module Nexus-C9508-FM ok
24 0 Fabric Module Nexus-C9508-FM ok
25 0 Fabric Module Nexus-C9508-FM ok
26 0 Fabric Module Nexus-C9508-FM ok
27 0 Supervisor Module Nexus-SUP-A active *
29 0 System Controller Nexus-SC-A active
```

- モジュール 24 に接続して、スロット 24 の FM のコマンド ラインにアクセスします。

```
switch# attach module 24
Attaching to module 24 ...
To exit type 'exit', to abort type '$.'
```

- コマンドを入力して、ファブリック モジュール ソフトウェアへの root アクセスを取得します。

```
module-24# test hardware internal bcm-usd bcm-diag-shell
Available Unit Numbers: 0 1
bcm-shell.0> 1
```

この時点で、スロット 24、T2 ASIC インスタンス 1 のファブリック モジュールの Broadcom シェルが表示されます。入力するすべてのコマンドは、この特定の ASIC インスタンスに固有のものです。

- bcm シェルを終了し、FM から切り離すには、exit コマンドを使用します。

```
bcm-shell.1> exit
module-24# exit
rlogin: connection closed.
```

ラインカードの bcm シェルへのアクセス

ライン カード（LC）の T2 ASIC に接続する場合は、まずモジュールに接続し、root モードを開始し、shell access exec を実行して、接続する ASIC インスタンスを選択します。使用可能な ASIC の数は、接続しているライン カードのモデルによって異なります。

次に、スロット 2 の LC の ASIC インスタンス 1 の bcm シェルにアクセスし、3 つの T2 インスタンスを含む LC の bcm シェルを終了する例を示します。

- モジュール 2 に接続して、スロット 2 の LC のコマンドラインにアクセスします。

```
switch# attach module 2
Attaching to module 2 ...
To exit type 'exit', to abort type '$.'
Last login: Wed Aug 7 14:13:15 UTC 2013 from sup27 on ttyp0
```

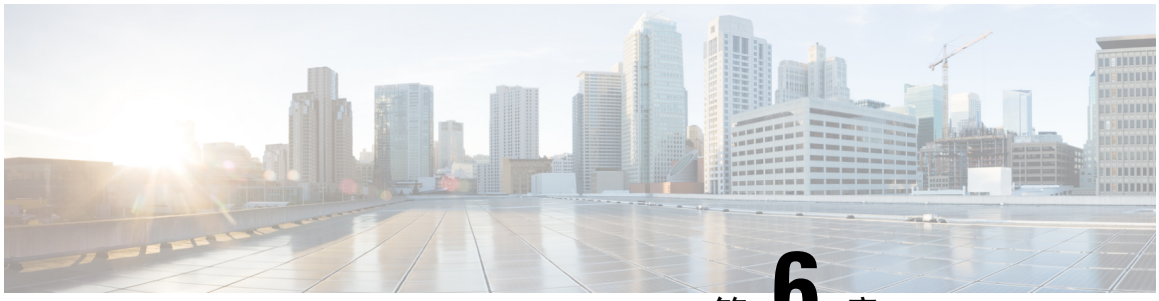
- コマンドを入力して、ラインカード ソフトウェアへの root アクセスを取得します。

```
switch-2# test hardware internal bcm-usd bcm-diag-shell
Available Unit Numbers: 0 1 2
bcm-shell.0> 1
bcm-shell.1>
```

この時点で、スロット 2、T2 ASIC インスタンス 1 のライン カード モジュールの Broadcom シェルを使用している状態になっています。

- **exit** コマンドを使用して bcm シェルを終了し、FM から切り離します。

```
bcm-shell.1> exit
module-2# exit
rlogin: connection closed.
```



第 6 章

Python API

Python の使用

ここでは、Python スクリプトの作成と実行の方法について説明します。

Cisco Python パッケージ

Cisco NX-OS は、インターフェイス、VLAN、VRF、ACL、ルートなど、多くのコア ネットワーク デバイス モジュールへのアクセスを可能にする Cisco Python パッケージを提供します。**help()** コマンドを入力すると、Cisco Python パッケージの詳細を表示できます。モジュール内のクラスとメソッドに関する追加情報を取得するには、特定のモジュールに対して help コマンドを実行します。たとえば、**help**（*cisco.interface*） は、cisco.interface モジュールのプロパティを表示します。

次の例は、Cisco Python パッケージに関する情報を表示する方法を示します。

```
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
    cisco

FILE
    /isan/python/scripts/cisco/__init__.py

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
    ospf
    routemap
    routes
```

```
    section_parser
    ssh
    system
    tacacs
    vrf

CLASSES
     __builtin__.object
        cisco.cisco_secret.CiscoSecret
        cisco.interface.Interface
        cisco.key.Key
```

次に、Python 3 用の Cisco Python パッケージに関する情報を表示する方法の例を示します。

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
cisco

PACKAGE CONTENTS
acl
bgp
buffer_depth_monitor
check_port_discards
cisco_secret
feature
historys
interface
ipaddress
key
line_parser
mac_address_table
md5sum
nxcli
nxos_cli
ospf
routemap
routes
section_parser
ssh
system
tacacs
transfer
vlan
vrf

CLASSES
builtins.dict(builtins.object)
cisco.history.History
builtins.object
cisco.cisco_secret.CiscoSecret
cisco.interface.Interface
cisco.key.Key
```

CLI コマンド API の使用

Python プログラミング言語は、CLI コマンドを実行できる 3 つの API を使用します。API は Python CLI モジュールから利用できます。

これらの API については、次の表で説明します。*** from cli import** コマンドを使用して API を有効にする必要があります。これらの API の引数は CLI コマンドの文字列です。Python インタープリタ経由で CLI コマンドを実行するには、次の API のいずれかの引数文字列として CLI コマンドを入力します。

表 *3 : CLI* コマンド *API*

API	説明
cli() 例： `string = cli ("cli-command")`	制御文字または特殊文字を含む CLI コマンドの未処理の出力を返します。 （注） インタラクティブな Python インタープリタは、制御文字または特殊文字を「エスケープ」して出力します。改行は「\n」として出力され、結果が読みにくい場合があります。**clip()** API は、判読性が高い結果を出力します。
clid() 例： `json_string = clid ("cli-command")`	コマンドに XML サポートが存在する場合は、cli-command の JSON 出力を返します。それ以外の場合は、例外がスローされます。 （注） この API は、show コマンドの出力の検索時に使用すると便利な場合があります。

clip()
例：
`clip ("cli-command")`
	CLI コマンドの出力を直接 stdout に出力し、Python には何も返されません。
	（注） `clip ("cli-command")`
	と同等です（is equivalent to）
	`r=cli("cli-command")`
	`print r`

2 つ以上のコマンドを個別に実行すると、その状態は 1 つのコマンドから後続のコマンドまで持続しません。

次の例では、最初のコマンドの状態が 2 番目のコマンドで持続しないため、2 番目のコマンドが失敗します。

```
>>> cli("conf t")
>>> cli("interface eth4/1")
```

2 つ以上のコマンドを同時に実行すると、その状態は 1 つのコマンドから後続のコマンドまで持続します。

次の例では、2 番目と 3 番目のコマンドの状態が持続するため、2 番目のコマンドは成功しています。

```
>>> cli("conf t ; interface eth4/1 ; shut")
```

（注） 例に示すように、コマンドは「;」で区切られます。セミコロン (;) は、単一のブランク文字で囲む必要があります。

CLI からの Python インタープリタの呼び出し

次に、CLI から Python 2 を呼び出す方法を表示します：

（注） Python インタープリタのプロンプトは「>>>」または「...」で表示されます。

重要 Python 2.7 のサポートは終了し、将来の NX-OS ソフトウェアは Python 2.7 のサポートを廃止します。新しいスクリプトでは、代わりに **python3**' を使用することをお勧めします。新しいシェルを使用するように **python3** を入力します。

```
switch# python
switch# python

Warning: Python 2.7 is End of Support, and future NXOS software will deprecate
python 2.7 support. It is recommended for new scripts to use 'python3' instead.
Type "python3" to use the new shell.

Python 2.7.11 (default, Jun  4 2020, 09:48:24)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print intf['interface']
...
mgmt0
loopback1
```

```
>>>
```

次に、CLI から Python 3 を呼び出す方法を表示します：

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print(intf['interface'])
...
mgmt0
loopback1
>>>
```

表示フォーマット

次に、Python API を使用したさまざまな表示フォーマットを示します：

例 1：

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> clip('where detail')
  mode:
  username:            admin
  vdc:                 switch
  routing-context vrf: default
```

例 2：

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> cli('where detail')
'  mode:                \n  username:            admin\n  vdc:
 switch\n  routing-context vrf: default\n'
>>>
```

例 3：

```
>>> r = cli('where detail')
>>> print(r)
mode:
username: admin
vdc: switch
routing-context vrf: default
```

```
>>>
```

例 4：

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
... print("%30s - %s" % (k,out[k]))
...
header_str - Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2020, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.
bios_ver_str - 07.67
kickstart_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
nxos_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
bios_cmpl_time - 01/29/2020
kick_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
nxos_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
kick_cmpl_time - 5/10/2020 21:00:00
nxos_cmpl_time - 5/10/2020 21:00:00
kick_tmstmp - 05/12/2020 07:08:44
nxos_tmstmp - 05/12/2020 07:08:44
chassis_id - Nexus9000 93180YC-EX chassis
cpu_name - Intel(R) Xeon(R) CPU @ 1.80GHz
memory - 24632252
mem_type - kB
proc_board_id - FDO22280FFK
host_name - switch
bootflash_size - 53298520
kern_uptm_days - 0
kern_uptm_hrs - 0
kern_uptm_mins - 19
kern_uptm_secs - 34
rr_usecs - 641967
rr_ctime - Tue May 12 09:52:28 2020
rr_reason - Reset Requested by CLI command reload
rr_sys_ver - 9.4(1)
rr_service - None
plugins - Core Plugin, Ethernet Plugin
manufacturer - Cisco Systems, Inc.
>>>
```

非インタラクティブ Python

Python スクリプト名を引数として Python CLI コマンドで使用することで、Python スクリプトを非インタラクティブ モードで実行できます。Python スクリプトは、ブートフラッシュまたは揮発性スキームの下に配置する必要があります。Python CLI コマンドでは、Python スクリプトの最大 32 個のコマンド ライン引数を使用できます。

スイッチは、Python スクリプトを実行するためのソース CLI コマンドもサポートしています。`bootflash:scripts` ディレクトリは、ソース CLI コマンドのデフォルトのスクリプト ディレクトリです。

この例では、最初にスクリプトを表示してから実行します。保存は、任意のファイルをブートフラッシュに持ってくるようなものです。

```
switch# show file bootflash:scripts/deltaCounters.py
#!/isan/bin/python3
from cli import *
import sys, time
ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'
out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print ('row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast')
print ('===============================================================')
print (' %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc))
print ('===============================================================')
i = 0
while (i < count):
    time.sleep(delay)
    out = json.loads(clid(cmd))
    rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
    rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
    rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
    txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
    txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
    txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
    i += 1
    print ('%-3d %8d %8d %8d %8d %8d %8d' % (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew -
rxbc, txucNew - txuc, txmcNew - txmc, txbcNew - txbc))
```

```
switch# python bootflash:scripts/deltaCounters.py mgmt0 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
===============================================================
      291     8233     1767      185       57        2
===============================================================
1          1        4        1        1        0        0
2          2        5        1        2        0        0
3          3        9        1        3        0        0
4          4       12        1        4        0        0
5          5       17        1        5        0        0
switch#
```

次の例は、送信元コマンドでコマンドライン引数を指定する方法を表示しています。この例では、*policy-map* は `cgrep python` スクリプトへの引数です。この例は、送信元コマンドがパイプ演算子 (「|」) の後に続くことも示しています。

```
switch# show running-config | source sys/cgrep policy-map

policy-map type network-qos nw-pfc
policy-map type network-qos no-drop-2
policy-map type network-qos wred-policy
policy-map type network-qos pause-policy
policy-map type qos foo
policy-map type qos classify
policy-map type qos cos-based
policy-map type qos no-drop-2
policy-map type qos pfc-tor-port
```

Embedded Event Manager でのスクリプトの実行

Embedded Event Manager（EEM）のポリシーは、Python スクリプトをサポートします。

次の例は、EEM アクションとして Python スクリプトを実行する方法を示しています。

- アクション コマンドを使用することで、EEM アプレットに Python スクリプトを含めることができます。

```
switch# show running-config eem

!Command: show running-config eem
!Running configuration last done at: Thu Jun 25 15:29:38 2020
!Time: Thu Jun 25 15:33:19 2020

version 9.3(5) Bios:version 07.67
event manager applet a1
  event cli match "show clock"
  action 1 cli python bootflash:pydate.py


switch# show file logflash:vdc_1/event_archive_1 | last 33

eem_event_time:06/25/2020,15:34:24 event_type:cli event_id:24 slot:active(1) vdc
:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
stty: standard input: Inappropriate ioctl for device
Executing the following commands succeeded:
          python bootflash:pydate.py
Completed executing policy a1
Event Id:24 event type:10241 handling completed
```

- **show file** *logflash*:*event_archive_1* コマンドを実行して、ログ ファイル内のイベントによってトリガーされたアクションを検索できます。

```
switch# show file logflash:event_archive_1 | last 33

eem_event_time:05/01/2011,19:40:28 event_type:cli event_id:8 slot:active(1)
vdc:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
```

```
Python

2011-05-01 19:40:28.644891
Executing the following commands succeeded:
          python bootflash:pydate.py

PC_VSH_CMD_TLV(7679) with q
```

Cisco NX-OS ネットワーク インターフェイスとの Python 統合

Cisco Nexus スイッチでは、Python が基盤となる Cisco NX-OS ネットワーク インターフェイスと統合されています。cisco.vrf.set_global_vrf（）API を介してコンテキストを設定することにより、ある仮想ルーティング コンテキストから別の仮想ルーティング コンテキストに切り替えることができます。

次の例は、デバイスの管理インターフェイスを介して HTML ドキュメントを取得する方法を示しています。目的の仮想ルーティング コンテキストに切り替えることにより、帯域内インターフェイスを介して外部エンティティへの接続を確立することもできます。

```
switch# python

Warning: Python 2.7 is End of Support, and future NXOS software will deprecate
python 2.7 support. It is recommended for new scripts to use 'python3' instead.
Type "python3" to use the new shell.

Python 2.7.11 (default, Jun  4 2020, 09:48:24)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> from cisco.vrf import *
>>> set_global_vrf('management')
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
>>> print page.read()
Hello Cisco Nexus 9000
>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set global vrf in module cisco.vrf:
set global vrf(vrf)
Sets the global vrf. Any new sockets that are created (using socket.socket)
will automatically get set to this vrf (including sockets used by other
python libraries).
Arguments:
vrf: VRF name (string) or the VRF ID (int).
Returns: Nothing
>>>
```

Python による Cisco NX-OS セキュリティ

Cisco NX-OS 情報技術は、ソフトウェアの Cisco NX-OS サンドボックス レイヤおよび CLI ロールベース アクセス コントロール（RBAC）によって保護されます。

Cisco NX-OS network-admin または dev-ops ロールに関連付けられているすべてのユーザは、特権ユーザです。カスタム ロールで Python へのアクセスが許可されているユーザーは、非特権ユーザーと見なされます。非特権ユーザは、ファイル システム、ゲスト シェル、Bash コマンドなど

の Cisco NX-OS 情報技術へのアクセスが制限されています。特権ユーザは、Cisco NX-OS のすべての情報技術へのアクセスが向上します。

セキュリティとユーザー権限の例

次の例は、特権ユーザーがコマンドを実行する方法を示しています：

Python 3 の例。

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
admin
0
>>> f=open('/tmp/test','w')
>>> f.write('hello from python')
17
>>> f.close()
>>> r=open('/tmp/test','r')
>>> print(r.read())
hello from python
>>> r.close()
>>>
```

次の例は、アクセスを拒否されている非特権ユーザーを示しています：

```
switch# python3
Python 3.7.3 (default, Nov 20 2019, 14:38:01)
[GCC 5.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','w')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
PermissionError: [Errno 13] Permission denied: '/tmp/test'
>>>
```

RBAC は、ログイン ユーザー権限に基づいて CLI アクセスを制御します。ログイン ユーザーの ID は、CLI シェルまたは Bash から呼び出される Python に与えられます。Python は、Python から呼び出されたサブプロセスにログイン ユーザーの ID を渡します。

以下は、特権ユーザーの例です：

```
>>> from cli import *
>>> cli('show clock')
'Warning: No NTP peer/server configured. Time may be out of sync.\n15:39:39.513 UTC Thu Jun
 25 2020\nTime source is NTP\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

!Command: show running-config l3vm
!Running configuration last done at: Thu Jun 25 15:39:49 2020
!Time: Thu Jun 25 15:39:55 2020
```

```
version 9.3(5) Bios:version 07.67

interface mgmt0
  vrf member management
vrf context blue
vrf context management
vrf context myvrf
```

以下は、非特権ユーザーの例です：

```
>>> from cli import *
>>> cli('show clock')
'11:18:47.482 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/isan/python/scripts/cli.py", line 20, in cli
    raise cmd_exec_error(msg)
errors.cmd_exec_error: '% Permission denied for the role\n\nCmd exec error.\n'
```

次の例は、RBAC 構成を示しています：

```
switch# show user-account
user:admin
        this user account has no expiry date
        roles:network-admin
user:pyuser
        this user account has no expiry date
        roles:network-operator python-role
switch# show role name python-role
```

スケジューラでスクリプトを実行する例

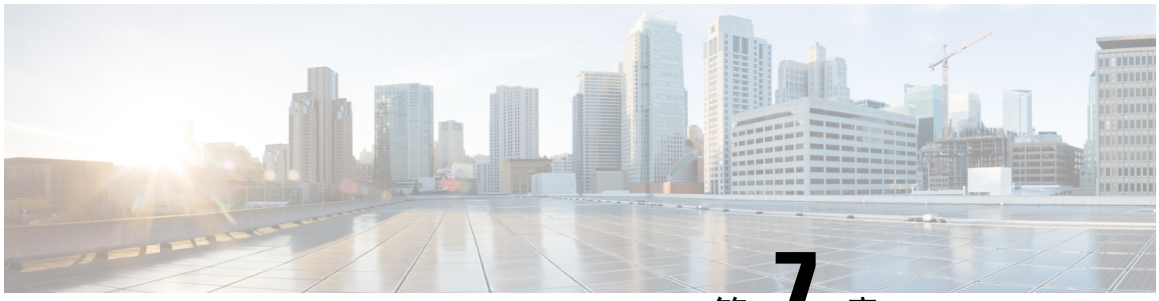
次の例は、スケジューラ機能を使用してスクリプトを実行する Python スクリプトを示しています。

```
#!/bin/env python
from cli import *
from nxos import *
import os

switchname = cli("show switchname")
try:
    user = os.environ['USER']
except:
    user = "No user"
    pass

msg = user + " ran " + __file__ + " on : " + switchname
print msg
py_syslog(1, msg)
# Save this script in bootflash:///scripts
```

Python 3 の例。

```
#!/bin/env python3
from cli import *
from nxos import *
import os
```

```
switchname = cli("show switchname")
try:
    user = os.environ['USER']
except:
    user = "No user"
    pass

msg = user + " ran " + __file__ + " on : " + switchname
print(msg)
py_syslog(1, msg)

# Save this script in bootflash:///scripts

switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# feature scheduler
switch(config)# scheduler job name testplan
switch(config-job)# python bootflash:///scripts/test.py
switch(config-job)# exit
switch(config)# scheduler schedule name testplan
switch(config-schedule)# job name testplan
switch(config-schedule)# time start now repeat 0:0:4
Schedule starts from Sat Jun 13 04:29:38 2020
switch# 2020 Jun 13 04:29:41 switch %USER-1-SYSTEM_MSG: No user ran /bootflash/scripts/test.py
 on : switch - nxpython
switch# show scheduler schedule
Schedule Name : testplan
------------------------------
User Name : admin
Schedule Type : Run every 0 Days 0 Hrs 4 Mins
Start Time : Sat Jun 13 04:29:38 2020
Last Execution Time : Sat Jun 13 04:29:38 2020
Last Completion Time: Sat Jun 13 04:29:41 2020
Execution count : 1
----------------------------------------------
Job Name Last Execution Status
----------------------------------------------
testplan Success (0)
==============================================================================
switch#
```



第 7 章

tcl によるスクリプティング

この章は次のトピックで構成されています。

Tcl について

Tcl (「ティックル」と発音) は、CLI コマンドの柔軟性を高めるスクリプト言語です。Tcl を使用して **show** コマンドの出力の特定の値を抽出したり、スイッチを設定したり、Cisco NX-OS コマンドをループで実行したり、スクリプトで Embedded Event Manager（EEM）ポリシーを定義したりすることができます。

このセクションでは、Tcl スクリプトを実行する方法、またはスイッチで Tcl を対話的に実行する方法について説明します。

注意事項と制約事項

次に、TCL スクリプトに関する注意事項と制約事項を示します。

一部のプロセスおよび **show** コマンドでは、大量の出力が発生する可能性があります。スクリプトを実行していて、実行時間の長い出力を終了する必要がある場合は、Ctrl+C（Ctrl+Z ではなく）を使用してコマンド出力を終了します。Ctrl+Z を使用すると、SIGCONT（信号継続）メッセージが生成され、スクリプトが停止する可能性があります。SIGCONT メッセージによって停止されたスクリプトは、動作を再開するためにユーザーの介入が必要です。

tclsh コマンドのヘルプ

Tcl コマンドでは、コマンドのヘルプは使用できません。インタラクティブ tcl シェル内から Cisco NX-OS コマンドのヘルプ機能に引き続きアクセスできます。

次に、インタラクティブ Tcl シェルで Tcl コマンドのヘルプがない場合の例を示します。

```
switch# tclsh
switch-tcl# set x 1
switch-tcl# puts ?
          ^
% Invalid command at '^' marker.
switch-tcl# configure ?
  <CR>
  session   Configure the system in a session
  terminal  Configure the system from terminal input

switch-tcl#
```

（注） 上の例では、Cisco NX-OS コマンドのヘルプ機能が引き続き使用できますが、Tcl の **puts** コマンドはヘルプ機能からのエラーを返します。

tclsh コマンドの履歴

端末で矢印キーを使用して、以前にインタラクティブ Tcl シェルで入力したコマンドにアクセスできます。

（注） インタラクティブ Tcl シェルを終了すると、 **tclsh** コマンドの履歴は保存されません。

tclsh のタブ補完

インタラクティブ Tcl シェルを実行している場合は、Cisco NX-OS コマンドのタブ補完を使用できます。Tcl コマンドでは、タブ補完は使用できません。

tclsh の CLI コマンド

インタラクティブ tcl シェル内から直接 Cisco NX-OS コマンドにアクセスできますが、 Tcl **cli** コマンドにより付加される場合のみ tcl スクリプト内で Cisco NX-OS コマンドを実行できます。

インタラクティブ Tcl シェルでは、次のコマンドは同じであり、正しく実行されます：

```
switch-tcl# cli show module 1 | incl Mod
switch-tcl# cli "show module 1 | incl Mod"
switch-tcl# show module 1 | incl Mod
```

Tcl スクリプトで、次の例のように、Cisco NX-OS コマンドに Tcl **cli** コマンドを付加する必要があります：

```
set x 1
cli show module $x | incl Mod
cli "show module $x | incl Mod"
```

スクリプトで次のコマンドを使用すると、そのスクリプトは機能不全になり、Tcl シェルにエラーが表示されます：

```
show module $x | incl Mod
"show module $x | incl Mod"
```

tclsh コマンドの区切り

セミコロン（;）は、Cisco NX-OS と Tcl の両方でのコマンド区切りです。Tcl コマンドで複数の Cisco NX-OS コマンドを実行するには、各 Cisco NX-OS コマンドを引用符（""）で囲む必要があります。

双方向性 Tcl シェルでは、次のコマンドは同じであり、正しく実行されます。

```
switch-tcl# cli "configure terminal ; interface loopback 10 ; description loop10"
switch-tcl# cli configure terminal ; cli interface loopback 10 ; cli description loop10
switch-tcl# cli configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.

switch(config-tcl)# cli interface loopback 10
switch(config-if-tcl)# cli description loop10
switch(config-if-tcl)#
```

双方向性 Tcl シェルでは、Tcl **cli** コマンドを付加せずに、直接 Cisco NX-OS コマンドを実行することもできます。

```
switch-tcl# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.

switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# description loop10
switch(config-if-tcl)#
```

tcl 変数

Tcl 変数を Cisco NX-OS コマンドへの引数として使用できます。また、Tcl スクリプトに引数を渡すこともできます。tcl 変数は永続的ではありません。

次の例は、Cisco NX-OS コマンドの引数として Tcl 変数を使用する方法を表示しています。

```
switch# tclsh
switch-tcl# set x loop10
switch-tcl# cli "configure terminal ; interface loopback 10 ; description $x"
switch(config-if-tcl)#
```

tclquit

tclquit コマンドは、どの Cisco NX-OS コマンド モードが現在現用系であるかには関係なく Tcl シェルを終了します。また、**Ctrl+C** を押して Tcl シェルを終了することもできます。**exit** と **end**Cisco NX-OS コマンドは、コマンド モードを変更します。**exit** コマンドは、EXEC コマンド モードからのみ Tcl シェルを終了します。

Tclsh セキュリティ

tcl シェルは、Cisco NX-OS システムの特定の部分への不正アクセスを防止するために、サンドボックスで実行されます。システムは、無限ループや過剰なメモリ使用率などのイベントを検出するために、tcl シェルによって使用されている CPU、メモリ、ファイルなどのシステム リソースをモニタリングします。

初期の tcl 環境は、**scripting tcl init** *init-file* コマンドで設定します。

scripting tcl recursion-limit *iterations* コマンドを使用して、tcl 環境のループ制限を定義できます。デフォルトの再帰制限は 1000 回の繰り返しです。

Tclsh コマンドの実行

tclsh コマンドを使用すると、スクリプトまたはコマンドラインから tcl コマンドを実行できます。

（注） CLI プロンプトの状態で tcl スクリプト ファイルを作成することはできません。スクリプト ファイルをリモート デバイスで作成して、Cisco NX-OS デバイスの bootflash: ディレクトリにコピーすることができます。

手順の概要

1. **tclsh** [**bootflash:***filename* [*argument* **...**]]

手順の詳細

	コマンドまたはアクション	目的

Step 1	**tclsh** [**bootflash:***filename* [*argument* **...**]]
	例：
	`switch# tclsh ?`
	`  <CR>`
	`  bootflash:  The file to run`
		tcl シェルを開始します。
		引数を指定せずに **tclsh** コマンドを実行すると、シェルは対話形式で実行され、標準入力から tcl コマンドを読み込んで、コマンドの結果とエラー メッセージを標準出力に出力します。**tclquit** を入力するか、**Ctrl-C** を押すとインタラクティブ tcl シェルが終了します。
		引数を指定して **tclsh** コマンドを実行すると、最初の引数は、tcl コマンドが記述されたスクリプト ファイルの名前になり、他の引数をスクリプトで変数として使用できます。

例

次の例は、インタラクティブな Tcl シェルを示しています。

```
switch# tclsh
switch-tcl# set x 1
switch-tcl# cli show module $x | incl Mod
Mod  Ports  Module-Type                          Model              Status
1    36     36p 40G Ethernet Module             N9k-X9636PQ        ok
Mod  Sw              Hw
Mod  MAC-Address(es)                           Serial-Num

switch-tcl# exit
switch#
```

次に、Tcl スクリプトを実行する方法の例を示します。

```
switch# show file bootflash:showmodule.tcl
set x 1
while {$x < 19} {
cli show module $x | incl Mod
set x [expr {$x + 1}]
}

switch# tclsh bootflash:showmodule.tcl
Mod  Ports  Module-Type                          Model              Status
1    36     36p 40G Ethernet Module             N9k-X9636PQ        ok
Mod  Sw              Hw
Mod  MAC-Address(es)                           Serial-Num

switch#
```

Tclsh コマンドからの Cisco NX-OS モード間の移動

インタラクティブ Tcl シェルの実行中に Cisco NX-OS のモードを変更できます。

手順の概要

1. **tclsh**
2. **configure terminal**
3. **tclquit**

手順の詳細

	コマンドまたはアクション	目的

Step 1	**tclsh**
	例:
	`switch# tclsh`
	`switch-tcl#`
		インタラクティブ Tcl シェルを開始します。

Step 2	**configure terminal** 例:	Tcl シェルで Cisco NX-OS のコマンドを実行して、モードを変更します。

	コマンドまたはアクション	目的
	`switch-tcl# configure terminal` `switch(config-tcl)#`	（注） Tcl プロンプトが変化して、Cisco NX-OS コマンドモードになったことが示されます。

Step 3	**tclquit**
	例：
	`switch-tcl# tclquit`
	`switch#`
		Tcl シェルを終了し、始めのモードに戻ります。

例

次の例は、対話型 Tcl シェルから Cisco NX-OS モードを変更する方法を示しています：

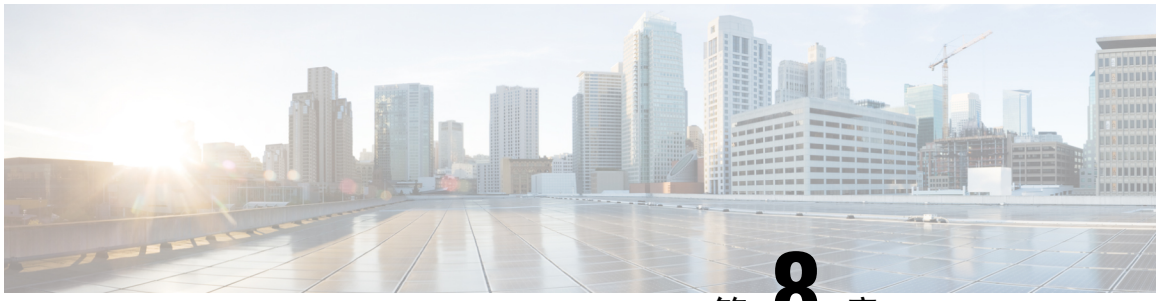
```
switch# tclsh
switch-tcl# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# ?
  description  Enter description of maximum 80 characters
  inherit      Inherit a port-profile
  ip           Configure IP features
  ipv6         Configure IPv6 features
  logging      Configure logging for interface
  no           Negate a command or set its defaults
  rate-limit   Set packet per second rate limit
  shutdown     Enable/disable an interface
  this         Shows info about current object (mode's instance)
  vrf          Configure VRF parameters
  end          Go to exec mode
  exit         Exit from command interpreter
  pop          Pop mode from stack or restore from name
  push         Push current mode to stack or save it under name
  where        Shows the cli context you are in

switch(config-if-tcl)# description loop10
switch(config-if-tcl)# tclquit
Exiting Tcl
switch#
```

tcl の参照

次のタイトルは、参照のために示されています。

- Mark Harrison (ed)、『*Tcl/Tk Tools*』、O'Reilly Media、ISBN 1-56592-218-2、1997 年
- Mark Harrison および Michael McLennan、『*Effective Tcl/Tk Programming*』、Addison-Wesley、Reading、MA、USA、ISBN 0-201-63474-0、1998 年

- Brent B. Ousterhout、『*Tcl and the Tk Toolkit*』、Addison-Wesley、Reading、MA、USA、ISBN 0-201-63337-X、1994 年
- Brent B. Welch、『*Practical Programming in Tcl and Tk*』、Prentice Hall、Upper Saddle River、NJ、USA、ISBN 0-13-038560-3、2003 年
- J Adrian Zimmer、『*Tcl/Tk for Programmers*』、IEEE Computer Society、John Wiley and Sons により出版、ISBN 0-8186-8515-8、1998 年



第 8 章

iPXE

iPXE について

iPXE は、オープンソースのネットワークブート ファームウェアです。 iPXE は、Etherboot から派生したオープンソースの PXE クライアントファームウェアおよびブートローダーである gPXE に基づいています。標準の PXE クライアントは TFTP を使用してデータを転送しますが、gPXE はより多くのプロトコルをサポートします。

標準 PXE の機能以外に iPXE が提供している追加機能のリストを次に示します。

- HTTP、iSCSI SAN、FCoE などを介した Web サーバーからのブート
- IPv4 と IPv6 の両方をサポート
- Netboot は HTTP/TFTP、IPv4、および IPv6 をサポート
- イメージへの埋め込みスクリプトや、HTTP/TFTP によって提供されるスクリプトなどをサポート
- DHCPv6 のステートレス アドレス自動設定（SLAAC）と、ステートフル IP 自動構成バリアントをサポート。 iPXE は、ブート URI と DHCPv6 オプションのパラメータをサポート。これは、IPv6 ルータ アドバタイズメントに依存します。

さらに、セキュリティ上の理由から、次のような iPXE の既存の機能の一部を無効にしています。

- bzImage+initramfs/initrd、または ISO などの標準 Linux イメージ形式のブートのサポート
- FCoE、iSCSI SAN、ワイヤレスなどの未使用のネットワーク ブート オプション
- サポートされていない NBP（syslinux/pxelinux など）のロード。これは適切にコード署名されていないシステム イメージを起動できるためです。

ネットブート要件

主要な要件は次のとおりです。

- 適切に構成された DHCP サーバー。
- TFTP/HTTP サーバー。
- デバイスが PXE ブートされるときに NX-OS がイメージをダウンロードするため、デバイスのブートフラッシュに十分なスペースがあること。
- IPv4/IPv6 サポート：導入の柔軟性を向上

iPXE に関する注意事項と制限事項

PXE に関する注意事項と制限事項は次のとおりです。

- iPXE による自動ブート中には 3 秒間の猶予時間があり、**Ctrl+B** を入力して PXE ブートを終了できます。次のオプションのプロンプトが表示されます。

```
Please choose a bootloader shell:
1). GRUB shell
2). PXE shell
Enter your choice:
```

- HTTP イメージのダウンロードと TFTP：TFTP は UDP ベースのプロトコルであるため、パケット損失が発生し始めた場合に問題が発生する可能性があります。TCP はウィンドウベースのプロトコルであり、帯域幅の共有または損失をより適切に処理できます。そのため、Cisco NX-OS イメージのサイズが 250 M バイトを超える場合には、TCP ベースのプロトコル サポートの方が適しています。
- iPXE は、シスコの署名付き NBI イメージのみを許可し、ブートを実行します。その他の標準イメージ形式は、セキュリティ上の理由からサポートが無効にされています。
- 複数のスーパーバイザがあるスイッチでは、PXE ブートに設定されたスーパーバイザ A+ および B+ の動作は、スーパーバイザ A または B の動作とは異なります。

  スーパーバイザ A+ または B+ が最初に PXE ブートから起動し、次にブートフラッシュから起動するように構成されている場合、スーパーバイザは連続して PXE からの起動を試行し、PXE ブートの再試行が失敗してもブートフラッシュ（GRUB）に切り替えません。ブートフラッシュから起動するには、スーパーバイザを手動でリロードする必要があります。

  PXE ブートは、**Ctrl+C** を入力して中断できます。その後 **Ctrl+B** を入力すると、PXE ブートを停止するためのプロンプトが表示されます。手動でスーパーバイザをリロードすると、ブートフラッシュから起動します。

  この制限は、スーパーバイザ A+ および B+ にのみ適用されます。同様の構成で、スーパーバイザ A と B は、自動的にリブートしてブートフラッシュからロードする前に、PXE ブートを 4 回試行します。

ブート モードの構成

VSH CLI

```
switch# configure terminal
switch(conf)# boot order bootflash|pxe [bootflash|pxe]
switch(conf)# end
```

（注） キーワード **bootflash** は、Grub ベースのブートであることを示します。

たとえば、PXE ブート モードのみを実行する場合、構成コマンドは次のようになります。

```
switch(conf)# boot order pxe
```

最初に Grub を起動し、次に PXE を起動する場合、次のようになります。

```
switch(conf)# boot order bootflash pxe
```

最初に PXE を起動し、次に Grub を起動する場合、次のようになります。

```
switch(conf)# boot order pxe bootflash
```

（注） スーパーバイザ A+ または B+ で **boot order pxe bootflash** を設定すると、スーパーバイザは継続的に PXE ブートを試行します。スーパーバイザ A+ または B+ は、手動による介入なしでは GRUB からのブートに切り替わりません。

boot order コマンドを使用しない場合、デフォルトのブート順序は Grub です。

（注） 次のセクションでは、Grub と iPXE を切り替える方法について説明します。

Grub CLI

bootmode [**-g**|**-p**|**-p2g**|**-g2p**]

キーワード	機能
-g	Grub のみ
-p	PXE のみ
-p2g	最初に PXE、PXE が失敗した場合には Grub
-g2p	最初に Grub、Grub が失敗した場合には PXE

Grub CLI は、フルバージョンの Cisco NX-OS イメージを起動せず、シリアル コンソールからブートモードを切り替える場合に役立ちます。また、すぐに継続的なPXEブート状態に移行する場合にも使用できます。

iPXE CLI

bootmode [**-g**|**--grub**] [**-p**|**--pxe**] [**-a**|**--pxe2grub**] [**-b**|**--grub2pxe**]

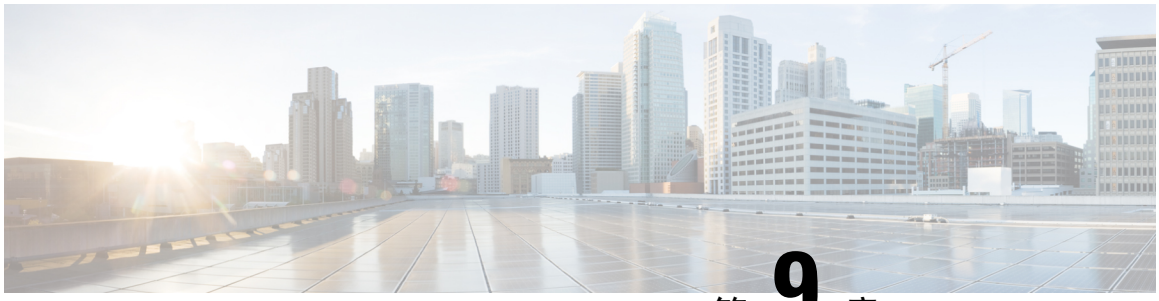
キーワード	機能
– – grub	Grub のみ
– – pxe	PXE のみ
– – pxe2grub	最初に PXE、PXE が失敗した場合には Grub
– – grub2pxe	最初に Grub、Grub が失敗した場合には PXE

iPXE CLI は、フルバージョンの Cisco NX-OS イメージを起動せず、シリアル コンソールからブートモードを切り替える場合に役立ちます。また、すぐに継続的なPXEブート状態に移行する場合にも使用できます。

ブート順の構成の確認

ブート順の構成情報を表示するには、次のコマンドを入力します。

コマンド	目的
show boot order	実行構成からの現在のブート順と、スタートアップ構成から取得した、次のリロード時のブート順の値を表示します。



第 9 章

カーネル スタック

カーネル スタックについて

カーネル スタック（kstack）は、既知の Linux API を使用してルートとフロント パネル ポートを管理します。

オープン コンテナは、ゲスト シェルと同様に、ホスト ソフトウェアから分離された Linux 環境です。ホスト ソフトウェア パッケージに影響を与えることなく、その環境内でソフトウェアをインストールまたは変更できます。

Guidelines and Limitations

- Guest shell, Docker containers, and the host Bash Shell use Kernel Stack (kstack).
- The Guest Shell and the host Bash Shell start in the default network namespace. Docker containers start in the management network namespace by default.
  - Other network namespaces may be accessed by using the **setns** system call
  - The **nsenter** and **ip netns exec** utilities can be used to execute within the context of a different network namespace.
- The interface state may be read from `/proc/net/dev` or retrieved using other typical Linux utilities such as **ip**, **ifconfig**, or **netstat**. The counters are for packets that have initiated or terminated on the switch.
- **ethtool –S** may be used to get extended statistics from the net devices, which includes packets that are switched through the interface.

- Packet capture applications like **tcpdump** may be run to capture packets that are initiated from or terminated on the switch.
- There is no support for networking state changes (interface creation or deletion, IP address configuration, MTU change, and so on) from the Guest Shell.
- IPv4 and IPv6 are supported.
- Raw PF_PACKET is supported.
- Only on stack (Netstack or kstack) at a time can use well-known ports (0-15000), regardless of the network namespace.
- There is no IP connectivity between applications using Nestack and applications running kstack on the same switch. This limitation holds true regardless of whether the kstack applications are being run from the host Bash Shell or within a container.
- Applications within the Guest Shell are not allowed to send packets directly over an Ethernet out-of-band channel (EOBC) interface to communicate with the line cards or standby Sup.
- The management interface (mgmt0) is represented as eth1 in the kernel netdevices.
- Use of the VXLAN overlay interface (NVE x) is not supported for applications utilizing the kernel stack. NX-OS features, including CLI commands, are able to use this interface via netstack.

  For more information about the NVE interface, see the Cisco Nexus 9000 Series NX-OS VXLAN Configuration Guide.

ポート範囲の変更

Netstack と kstack は、それらの間のポート範囲を分割します。デフォルトのポート範囲は次のとおりです：

- Kstack：15001 ～ 58000
- Netstack：58001 ～ 65535

（注） この範囲内で、63536 ～ 65535 は NAT 用に予約されています。

（注） **nxapi use-vrf management** で構成されたポートは kstack を使用し、アクセス可能です。

手順の概要

1. [**no**] **sockets local-port-range** *start-port end-port*

手順の詳細

	コマンドまたはアクション	目的
Step 1	[**no**] **sockets local-port-range** *start-port end-port*	このコマンドは、kstack のポート範囲を変更します。このコマンドは、Netstack の範囲を変更しません。

例

次に、kstack ポート範囲を設定する例を示します：

```
switch# sockets local-port-range 15001 25000
```

次のタスク

コマンドを入力した後は、次の点に注意してください：

- コマンドを入力した後には、スイッチをリロードします。
- Netstack で使用される 7000 以上のポートを、未割り当てのままにします。
- ポート範囲に抜けが生じるのを回避するには、*start-port* を 15001 に指定するか、*end-port* を 65535 に指定します。

kstack を使用した VXLAN について

NX-OS 9.2(1) 以降、VXLAN EVPN は kstack でサポートされ、サードパーティ製アプリケーションで活用できます。この機能は、Cisco Nexus 9000 ToR スイッチでサポートされています。

kstack のための VXLAN のセットアップ

VXLAN EVPN のインターフェイスまたはネットワーク名前空間にサードパーティ アプリケーションからアクセスできるようにするために、追加の構成は必要ありません。VXLAN EVPN ルートは、NX-OS VXLAN EVPN 構成に基づいてカーネルで自動的にプログラムされます。詳細については、『*Cisco Nexus 9000* シリーズ *NX-OS VXLAN* 構成ガイド』の「VXLAN BGP EVPN の構成」の章を参照してください。

kstack での VXLAN のトラブルシューティング

VXLAN の問題をトラブルシューティングする際には、次のコマンドを入力して、収集するべきいくつかの重要な情報を一覧表示してください。

```
switch(config)# show tech-support kstack
```

- **ip route show** コマンドを実行します：

```
root@switch(config)# run bash sudo su-
root@switch# ip netns exec evpn-tenant-kk1 ip route show
```

次のような出力が表示されます。

```
10.160.1.0/24 dev Vlan1601 proto kernel scope link src 10.160.1.254
10.160.1.1 dev veth1-3 proto static scope link metric 51
10.160.2.0/24 dev Vlan1602 proto kernel scope link src 10.160.2.253
127.250.250.1 dev veth1-3 proto static scope link metric 51
```

対応する VRF のすべての EVPN ルートがカーネルに存在することを確認します。

- **ip neigh show** コマンドを実行します：

```
root@switch(config)# run bash sudo su-
root@switch# ip netns exec evpn-tenant-kk1 ip neigh show
```

次のような出力が表示されます。

```
10.160.1.1 dev veth1-3 lladdr 0c:75:bd:07:b4:33 PERMANENT
127.250.250.1 devveth1-3 lladdr0c:75:bd:07:b4:33 PERMANENT
```

ネットデバイスのプロパティの変更

NX-OS 9.2(2) リリース以降、フロント チャネル ポート インターフェイスを表すネットデバイスは常に ADMIN UP 状態です。最終的に有効な状態は、リンク キャリアの状態によって決まります。

次に、NX-OS の以下のインターフェイスの例を示します。eth1/17 は **up** として表示され、eth1/1 は **down** として表示されます。

```
root@kstack-switch# sh int ethernet 1/17 brief
Eth1/17       --      eth  routed up      none                      1000(D) -


root@kstack-switch# sh int ethernet 1/1 brief
Eth1/1        --      eth  routed down    Link not connected        auto(D) -
```

次の例は、これらと同じインターフェイスを示していますが、今回は**ip link show** コマンドを使用して Bash シェルに表示しています。

```
bash-4.3# ip link show Eth1-17
49: Eth1-17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT group default qlen 100
    link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff


bash-4.3# ip link show Eth1-1
33: Eth1-1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode
DEFAULT group default qlen 100
    link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff
```

この例では、Eth1-1 は **UP** として表示されていますが、**NO-CARRIER** および **state DOWN** として表示されています。

次の例は、これらと同じインターフェイスを示していますが、今回は **ifconfig** コマンドを使用して Bash シェルに表示しています。

```
bash-4.3# ifconfig Eth1-17
Eth1-17   Link encap:Ethernet  HWaddr 00:42:68:58:f8:eb
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7388 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:1869164 (1.7 MiB)


bash-4.3# ifconfig Eth1-1
Eth1-1    Link encap:Ethernet  HWaddr 00:42:68:58:f8:eb
          inet addr:99.1.1.1  Bcast:99.1.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

ifconfig コマンドの出力には、さまざまな情報が表示されますが、**RUNNING** キーワードを使用して最終的な状態を示すことができます。デフォルトでは、すべてのネットデバイスにキーワード **UP** が表示されます。これは、カーネル内のネットデバイスの ADMIN 状態を表しています。

NX-OS 9.2(2) リリースの変更点の一部に、次のものがあります。

- **ネットデバイスの IPv4 アドレス：**NX-OS 9.2(2) リリースより前は、NX-OS の対応するインターフェイスが **DOWN** 状態であっても、IPv4 アドレスはカーネルのネットデバイスに組み込まれていました。NX-OS 9.2(2) リリース以降、IPv4 アドレスは、インターフェイスが **UP** 状態の場合にのみカーネル空間に組み込まれます。いったん組み込まれると、インターフェイスが **DOWN** になっても、IPv4 アドレスはカーネル内のネットデバイスに残ります。次の CLI コマンドを入力して、NX-OS インターフェイスから IP アドレスを明示的に削除した後にのみ削除されます。

  ```
  Interface Eth1/1
              no ip address IP-address
  ```

- **ネットデバイスの IPv6 アドレス：**NX-OS 9.2(2) リリースより前は、インターフェイスが **DOWN** になると、IPv6 アドレスはカーネルのネットデバイスからフラッシュされていました。NX-OS 9.2(2) リリース以降、ネットデバイスは常に管理 **UP** 状態であるため、インターフェイスがダウンしても、IPv6 アドレスはカーネルからフラッシュされません。

第 II 部

アプリケーション

第 10 章

サードパーティ製アプリケーション

サードパーティ製アプリケーションについて

サードパーティ製アプリケーションの RPM は、https://devhub.cisco.com/artifactory/open-nxos/9.2.1/ のリポジトリで入手できます。これらのアプリケーションは、Bash シェルで **dnf** コマンドを使用するか、NX-OS CLI を介してネイティブホストにインストールされます。

dnf install *rpm* コマンドを入力すると、Cisco **DNF** プラグインが実行されます。このプラグインは、RPM を表示されない場所にコピーします。スイッチのリロード時に、システムは RPM を再インストールします。

構成が `/etc` に置かれている場合、Linux プロセス、**incrond** は、ディレクトリで作成されたアーティファクトをモニターし、それらを表示されない場所にコピーし、この場所から `/etc` にコピーし直します。

注意事項と制約事項

サードパーティ製アプリケーションの RPM には、次の注意事項と制約事項があります。

- Cisco NX-OS リリース 9.2(1) 以降では、エージェントが保存されている Cisco リポジトリは https://devhub.cisco.com/artifactory/open-nxos/9.2.1/ にあります。このリポジトリでホストされているすべての RPM は、リリース キーで署名されています。
- NX-OS 10.1(1) リリースには、NX-Linux（シスコ独自の Linux ディストリビューション）に基づく新しいオペレーティング システムと rootfs があります。WRL5/WRL8 を使用してビルド

されたサードパーティ製 RPM は NX-Linux と互換性がなく、動作しない可能性があります。この場合、以前のリリースで使用されていたアプリケーションの古いバージョンを削除し、https://devhub.cisco.com/artifactory/open-nxos/10.1.1/ のリポジトリで入手可能な NX-Linux と互換性のある新しいソフトウェアに置き換えます。

- 署名付き RPM をインストールするための注意事項と手順は、*Cisco Nexus 9000* シリーズ *NX-OS* ソフトウェア アップグレードおよびダウングレード ガイド、リリース *9.2(x)*に記載されています。RPM 管理のための DNF および VSH の CLI オプション、署名付きおよび署名なしの RPM のインストール、リポジトリのクリーンアップなどが含まれています。
- サードパーティ製アプリケーションは、スイッチの起動時に開始されます。サードパーティ製アプリケーションは、その通信インターフェイスが起動する前、またはスイッチと通信ピアまたはサーバー間のルーティングが確立される前に起動される可能性があります。したがって、サードパーティ製アプリケーションは、通信障害が発生した場合でも堅牢に動作するように、そして接続の確立を再試行するように作成する必要があります。通信障害が発生した場合の復元力がアプリケーションにない場合は、「ラッパー」アプリケーションを使用して、目的のアプリケーションを起動する前に通信ピアが到達可能であるかどうか確認し、必要に応じて目的のアプリケーションを再起動できるようにする必要があります。

Python2 および依存パッケージのインストール

次に、パッケージのインストールの詳しいワークフローを示します。

```
switch# cat /etc/dnf/repos.d/open-nxos.repo
[open-nxos]
name=open-nxos
baseurl=https://devhub.cisco.com/artifactory/open-nxos/10.2.3/
enabled=1
gpgcheck=0
sslverify=0

dnf info packagegroup-nxos-64-python-2-deprecated-rpms
dnf install packagegroup-nxos-64-python-2-deprecated-rpms
The output of these cmds will be available post KR3F CCO.
```

サードパーティのネイティブ RPM/パッケージのインストール

パッケージのインストールの詳しいワークフローは次のとおりです。

エージェントが保存されているシスコのリポジトリを指すように、スイッチのリポジトリを構成します。

```
bash-4.2# cat /etc/dnf/repos.d/open-nxos.repo
[open-nxos]
name=open-nxos

baseurl=https://devhub.cisco.com/artifactory/open-nxos/9.2.1/
baseurl=https://devhub.cisco.com/artifactory/open-nxos/10.1.1/
```

```
enabled=1
gpgcheck=0
sslverify=0
```

CLI を使用してデジタル署名をインポートする手順については、*Cisco Nexus 9000* シリーズ *NX-OS* ソフトウェア アップグレードおよびダウングレード ガイド、リリース *9.2(x)* の「デジタル署名サポートのためのインストール CLI の使用」を参照してください。

dnf を使用した RPM のインストールの例と、詳しいインストール ログです。

例：

```
bash-4.2# dnf install splunkforwarder
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package splunkforwarder.x86_64 0:6.2.3-264376 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================================
 Package              Arch                 Version           Repository              Size
================================================================================================
Installing:
splunkforwarder       x86_64               6.2.3-264376      open-nxos               13 M

Transaction Summary
================================================================================================
Install       1 Package

Total size: 13 M
Installed size: 34 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : splunkforwarder-6.2.3-264376.x86_64
                                                                                1/1
complete

Installed:
  splunkforwarder.x86_64 0:6.2.3-264376


Complete!
bash-4.2#
```

パッケージが正常にインストールされたかどうかをスイッチに問い合わせ、そのプロセスまたはサービスが稼働していることを確認する例です。

例：

```
bash-4.2# dnf info splunkforwarder
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
Fretta              |  951 B     00:00 ...
groups-repo         | 1.1 kB     00:00 ...
localdb             |  951 B     00:00 ...
patching            |  951 B     00:00 ...
```

```
thirdparty          |  951 B     00:00 ...
Installed Packages
Name        : splunkforwarder
Arch        : x86_64
Version     : 6.2.3
Release     : 264376
Size        : 34 M
Repo        : installed
From repo   : open-nxos
Summary     : SplunkForwarder
License     : Commercial
Description : The platform for machine data.
```

永続的なサードパーティ RPM

次に、永続的なサードパーティ RPM の背後にあるロジックを示します。

- ローカル **dnf** リポジトリは、永続的なサードパーティ RPM 専用です。`/etc/yum/repos.d/thirdparty.repo` は `/bootflash/.rpmstore/ thirdparty` を指しています。
- **dnf install third-party.rpm** コマンドを入力するたびに、RPM のコピーが `//bootflash/.rpmstore/ thirdparty` に保存されます。
- リブート中に、サードパーティ リポジトリ内のすべての RPM がスイッチに再インストールされます。
- `/etc` 構成ファイルの変更は、`/bootflash/.rpmstore/config/etc` の下に保持され、`/etc` での起動時に再生されます。
- `/etc` ディレクトリに作成されたスクリプトは、リロード後も保持されます。たとえば、`/etc/init.d/` の下に作成されたサードパーティのサービス スクリプトは、リロード中にアプリケーションを起動します。

（注） iptables のルールは、bash シェルで変更された場合、再起動後に保持されません。

変更した iptables を永続化するには、リロード間で Iptable を永続化する（206 ページ）を参照してください。

VSH からの RPM のインストール

パッケージの追加

NX-OS 機能 RPM は、VSH CLI を使用してインストールすることもできます。

手順の概要

1. **show install package**
2. **install add ?**
3. **install add** *rpm-packagename*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**show install package**	すでに存在するパッケージとバージョンを表示します。
Step 2	**install add ?**	サポートされている URI を確認します。
Step 3	**install add** *rpm-packagename*	The **install add** コマンドは、ローカル ストレージ デバイスまたは、ネットワーク サーバーへパッケージ ファイルをコピーします。

例

次に、Chef RPM をアクティブにする例を示します:

```
switch# show install package
switch# install add ?
WORD         Package name
bootflash:   Enter package uri
ftp:         Enter package uri
http:        Enter package uri
modflash:    Enter package uri
scp:         Enter package uri
sftp:        Enter package uri
tftp:        Enter package uri
usb1:        Enter package uri
usb2:        Enter package uri
volatile:    Enter package uri
switch# install add
bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5.x86_64.rpm
[####################] 100%
Install operation 314 completed successfully at Thu Aug  6 12:58:22 2015
```

次のタスク

パッケージをアクティブ化する準備ができたら、パッケージのアクティブ化（124 ページ）に移動します。

（注） RPM パッケージの追加とアクティブ化は、次の 1 つのコマンドで実行できます。

```
switch#
install add bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5.x86_64.rpm
activate
```

パッケージのアクティブ化

始める前に

RPM は事前に追加しておく必要があります。

手順の概要

1. **show install inactive**
2. **install activate** *rpm-packagename*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**show install inactive**	追加されていても、アクティブ化されていないパッケージのリストを表示します。
Step 2	**install activate** *rpm-packagename*	パッケージをアクティブ化します。

例

次に、パッケージをアクティブ化する例を示します。

```
switch# show install inactive
Boot image:
        NXOS Image: bootflash:///yumcli6.bin

Inactive Packages:
        sysinfo-1.0.0-7.0.3.x86_64
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
              : protect-packages
Available Packages
chef.x86_64       12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5 thirdparty
eigrp.lib32_n9000 1.0.0-r0                                            groups-rep
o
sysinfo.x86_64    1.0.0-7.0.3                                         patching
switch# install activate chef-12.0-1.el5.x86_64.rpm
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

パッケージの非アクティブ化

手順の概要

1. **install deactivate** *package-name*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**install deactivate** *package-name*	RPM パッケージを非アクティブ化します。

例

次に、Chef RPM パッケージを非アクティブ化する例を示します：

```
switch# install deactivate chef
```

パッケージの削除

始める前に

パッケージを削除する前に非アクティブ化します。非アクティブ化されたRPMパッケージのみ削除できます。

手順の概要

1. **install remove** *package-name*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**install remove** *package-name*	RPM パッケージを削除します。

例

次に、Chef RPM パッケージを削除する例を示します。

```
switch# install remove chef-12.0-1.el5.x86_64.rpm
```

インストール済みパッケージの表示

手順の概要

1. **show install packages**

手順の詳細

	コマンドまたはアクション	目的
Step 1	**show install packages**	インストールされているパッケージのリストを表示します。

例

次に、インストールされているパッケージのリストを表示する例を示します。

```
switch# show install packages
```

詳細ログの表示

手順の概要

1. **show tech-support install**

手順の詳細

	コマンドまたはアクション	目的
Step 1	**show tech-support install**	詳細ログを表示します。

例

次の例は、詳細ログを表示する方法を示しています。

```
switch# show tech-support install
```

パッケージのアップグレード

手順の概要

1. install add *package-name* activate upgrade

手順の詳細

	コマンドまたはアクション	目的
Step 1	install add *package-name* activate upgrade	パッケージをアップグレードします。

例

次の例は、パッケージをアップグレードする方法を示しています。

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate upgrade
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

パッケージのダウングレード

手順の概要

1. install add *package-name* activate downgrade

手順の詳細

	コマンドまたはアクション	目的
Step 1	install add *package-name* activate downgrade	パッケージをダウングレードします。

例

次の例は、パッケージをダウングレードする方法を示しています。

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate downgrade
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

サードパーティ製アプリケーション

NX-OS

NX-API REST API オブジェクト モデルの仕様の詳細については、https://developer.cisco.com/media/dme/index.html を参照してください。

DevOps 構成管理ツール

DevOps 構成管理ツールについては、次のリンクを参照してください。

- Ansible 2.0 リリース（Nexus サポート）、Ansible リリース インデックス
- Ansible NX-OS サンプル モジュール、Ansible NX-OS サンプル モジュール
- Puppet、Puppet Forge Cisco Puppet
- Cisco Puppet モジュール（Git）Cisco Network Puppet モジュール
- Chef、Chef Supermarket Cisco クックブック
- Cisco Chef クックブック（Git）Cisco Network Chef クックブック

V9K

ESX5.1/5.5、VirtualBox、Fusion、および KVM の場合、仮想 Nexus 9000 スイッチをダウンロードするには、https://software.cisco.com/portal/pub/download/portal/select.html?&mdfid=286312239&flowid=81422&softwareid=282088129に移動します。

Automation Tool Educational Content

For a free book on Open NX-OS architecture and automation, see http://www.cisco.com/c/dam/en/us/td/docs/switches/datacenter/nexus9000/sw/open_nxos/programmability/guide/Programmability_Open_NX-OS.pdf

collectd

collectd は、システム パフォーマンスの統計情報を定期的に収集し、RRD ファイルなどの値を保存する複数の手段を提供するデーモンです。これらの統計情報を使用して、現在のパフォーマンスのボトルネックを見つけたり（パフォーマンス分析など）、将来のシステム負荷を予測したりできます（つまり、キャパシティ プランニング）。

詳細については、https://collectd.org を参照してください。

Ganglia

Ganglia は、クラスタやグリッドなどのハイパフォーマンス コンピューティング システム向けのスケーラブルな分散モニタリング システムです。これは、クラスタのフェデレーションを対象とした階層設計に基づいています。データ表現のための XML、コンパクトでポータブルなデータ転送のための XDR、データストレージと可視化のための RRDtool など、広く使用されているテクノロジーを活用しています。設計されたデータ構造とアルゴリズムを使用して、ノードあたりのオーバーヘッドを低く抑え、同時実行性を高めます。この実装は堅牢で、広範なオペレーティングシステムとプロセッサアーキテクチャに移植されており、現在、世界中の何千ものクラスタで使用されています。世界中の大学キャンパス間でクラスタをリンクするために使用されており、2000 ノードのクラスタを処理するように拡張できます。

詳細については、http://ganglia.info を参照してください。

Iperf

Iperfは、TCPおよびUDPの最大帯域幅パフォーマンスを測定するためにNLANR/DASTによって開発されました。Iperfを使用すると、さまざまなパラメータとUDP特性を調整できます。Iperfは、帯域幅、遅延ジッターとデータグラム損失を報告します。

詳細については、http://sourceforge.net/projects/iperf/またはhttp://iperf.sourceforge.netを参照してください。

LLDP

リンク層検出プロトコル（LLDP）は、EDPやCDPなどの独自のリンク層プロトコルに代わるように設計された業界標準プロトコルです。LLDPの目的は、隣接するネットワーク デバイスにリンク層通知を配信するための、ベンダー間互換性のあるメカニズムを提供することです。

詳細については、https://vincentbernat.github.io/lldpd/index.html を参照してください。

Nagios

Nalios は、Nadios Remote Plug-in Executor（NRPE）および SSH または SSL トンネルを介して以下をモニターするオープンソースソフトウェアです。

- ICMP、SNMP、SSH、FTP、HTTP などによるネットワーク サービス
- CPU 負荷、ディスク使用率、システム ログなどのホスト リソース
- サーバー、スイッチ、アプリケーションのアラート サービス
- [サービス（Services）]

詳細については、https://www.nagios.org/ を参照してください。

OpenSSH

OpenSSH は、盗聴、接続ハイジャック、およびその他の攻撃を排除するために、すべてのトラフィック（パスワードを含む）を暗号化するSSH接続ツールのオープンソースバージョンです。OpenSSHは、セキュアなトンネリング機能と複数の認証方式を提供し、すべてのSSHプロトコルバージョンをサポートします。

詳細については、http://www.openssh.com を参照してください。

Quagga

Quaggaは、さまざまなルーティングプロトコルを実装するネットワークルーティングソフトウェア スイートです。Quaggaデーモンは、ネットワーク アクセス可能CLI（「vty」という）を使用して構成できます。

（注） Quagga BGP のみが検証されています。

詳細については、http://www.nongnu.org/quagga/ を参照してください。

Splunk

Splunk is a web-based data collection, analysis, and monitoring tool that has search, visualization, and prepackaged content for use-cases. The raw data is sent to the Splunk server using the Splunk Universal Forwarder. Universal Forwarders provide reliable, secure data collection from remote sources and forward that data into the Splunk Enterprise for indexing and consolidation. They can scale to tens of thousands of remote systems, collecting terabytes of data with a minimal impact on performance.

For additional information, see http://www.splunk.com/en_us/download/universal-forwarder.html.

tcollector

tcollector は、ローカル コレクタからデータを収集し、そのデータをオープン時系列データベース（OpenTSDB）にプッシュするクライアント側プロセスです。

tcollector には次の機能があります。

- データ コレクタを実行し、データを照合します。
- 時系列データベース（TSD）への接続を管理します。
- コレクタに TSD コードを埋め込む必要がなくなります。
- 繰り返される値の重複を排除します。
- ワイヤ プロトコル作業を処理します。

詳細については、http://opentsdb.net/docs/build/html/user_guide/utilities/tcollector.html を参照してください。

tcpdump

tcpdump は、ネットワーク インターフェイス上の Boolean 式に一致するパケットの内容に関する説明を出力する CLI アプリケーションです。説明の前にタイムスタンプが表示されます。デフォルトでは、午前 0 時からの時間、分、秒、および小数点以下の秒として出力されます。

tcpdump は、次のフラグを使用して実行できます。

- -w：後で分析するためにパケット データをファイルに保存します。
- -r：ネットワーク インターフェイスからパケットを読み取るのではなく、保存されたパケット ファイルから読み取ります。
- -V：保存されたパケット ファイルのリストを読み取ります。

いずれの場合も、tcpdump は式にマッチするパケットだけを処理します。

詳細については、http://www.tcpdump.org/manpages/tcpdump.1.html を参照してください。

TShark

TShark は、CLI のネットワーク プロトコル アナライザです。Tshark を使用すると、ライブネットワークからパケットデータをキャプチャしたり、以前に保存したキャプチャファイルからパケットを読み取ったりできます。これらのパケットのデコードされた形式を標準出力に出力するか、パケットをファイルに書き込むことができます。TShark のネイティブ キャプチャ ファイル フォーマットは、pcap です。このフォーマットは、**tcpdump** と他のツールに使用されています。TShark は、cap_net_admin ファイル機能を削除した後、ゲストシェル内で使用できます。

```
setcap
 cap_net_raw=ep /sbin/dumpcap
```

（注） このコマンドは、ゲストシェル内で実行する必要があります。

詳細については、https://www.wireshark.org/docs/man-pages/tshark.html を参照してください。

第 11 章

Cisco NX-OS での Ansible の使用

前提条件

サポートされている制御環境のインストール要件については、https://docs.ansible.com/ansible/latest/getting_started/index.html を参照してください。

アンシブルについて

Ansibleは、クラウドプロビジョニング、構成管理、アプリケーションの展開、サービス内オーケストレーション、およびその他の IT ニーズを自動化するオープンソースの IT 自動化エンジンです。

Ansible は、Ansible モジュールと呼ばれる小さなプログラムを使用してノードへの API 呼び出しを行い、Playbook で定義された構成を適用します。

デフォルトでは、Ansibleは、すべての管理対象マシンを独自に選択したグループに入れる単純な INI ファイルを使用して、管理対象のマシンを表します。

詳細については、Ansible から入手できます。

Ansible	https://www.ansible.com/
Ansible 自動化ソリューションインストール手順、プレイブックの手順と例、モジュールリストなどが含まれます。	https://docs.ansible.com/

Cisco Ansible モジュール

次のリンクの表に示すように、Ansible には複数の Cisco NX-OS でサポートされるモジュールとプレイブックがあります。

NX-OS デベロッパーのランディング ページ。	構成管理ツール
Ansible NX-OS プレイブックの例	Ansible nxos Playbook のリポジトリ
Ansible NX-OS ネットワーク モジュール	nxos ネットワーク モジュール

第 12 章

Puppet Agent

この章は次のトピックで構成されています。

Puppet について

Puppet Labs によって開発された Puppet ソフトウェア パッケージは、サーバやその他の技術情報を管理するためのオープン ソースの自動化ツールセットです。Puppet ソフトウェアは、構成設定などのデバイス状態を適用することにより、サーバとリソースの管理を実現します。

Puppet コンポーネントには、管理対象デバイス（ノード）および Puppet Primary（サーバ）上で動作する Puppet エージェントが含まれます。通常、Puppet Primary は個別の専用サーバ上で実行され、複数のデバイスにサービスを提供します。Puppet エージェントの操作では、Puppet Primary に定期的に接続する必要があります。そして、Puppet Primary は構成マニフェストをコンパイルしてエージェントに送信します。エージェントは、ノードの現在の状態でこのマニフェストを調整し、相違点に基づいて状態を更新します。

Puppet マニフェストは、デバイスの状態を設定するためのプロパティ定義の集合です。これらのプロパティ状態の確認および設定の詳細は抽象化されているため、マニフェストは複数のオペレーティングシステムまたはプラットフォームで使用できます。マニフェストは、通常、構成時の設定を定義するために使用されますが、ソフトウェアパッケージのインストール、ファイルのコピー、およびサービスの開始にも使用できます。

詳細については、Puppet Labs を参照してください。

Puppet Labs	https://puppetlabs.com
Puppet Labs FAQ	https://puppet.com/blog/how-get-started-puppet-enterprise-faq/
Puppet Labs ドキュメント	https://puppet.com/docs

前提条件

Puppet エージェントの前提条件は次のとおりです。

- 仮想サービスのインストールと Puppet Agent の展開に必要なディスク ストレージをデバイスで使用できる必要があります。
  - ブートフラッシュに最低 450MB の空きディスク容量。
- Puppet 4.0 以降の Puppet プライマリ サーバーが必要です。
- Puppet エージェント 4.0 以降が必要です。

Puppet エージェント NX-OS 環境

Puppet Agent ソフトウェアは、ゲスト シェル（CentOS を実行する Linux コンテナ環境）のスイッチにインストールする必要があります。ゲスト シェルは、ホストから切り離された安全でオープンな実行環境を提供します。

Cisco NX-OS リリース 9.2（1）以降、Puppet Agent の Bash-shell（Cisco NX-OS の基盤となるネイティブ WindRiver Linux 環境）インストールはサポートされなくなりました。

次に、エージェント ソフトウェアのダウンロード、インストール、およびセットアップに関する情報を示します：

Puppet Agent：Cisco Nexus スイッチでのインストールとセットアップ （手動セットアップ）	https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md

ciscopuppet モジュール

ciscopuppet モジュールは、Cisco が開発したオープン ソース ソフトウェア モジュールです。これは、Puppet マニフェストの抽象技術情報構成と、Cisco NX-OS オペレーティング システムおよびプラットフォームの特定の実装の詳細との間のインターフェイスとなります。このモジュールは Puppet プライマリにインストールされ、Cisco Nexus スイッチでの Puppet エージェントの操作に必要です。

ciscopuppet モジュールは、Puppet Forge で利用できます。

ここでは、ciscopuppet モジュール インストール手順についての追加情報を提供します：

ciscopuppet モジュールの場所 （Puppet Forge）	Puppet Forge

リソースの種類のカタログ	[Cisco Puppet 技術情報の参照先（Cisco Puppet Resource Reference）]
ciscopuppet モジュール：送信元コード リポジトリ	[Cisco Network Puppet モジュール（Cisco Network Puppet Module）]
ciscopuppet モジュール：セットアップと使用法	Cisco Puppet モジュール::README.md
Puppet Labs: モジュールのインストール	https://puppet.com/docs/puppet/7/modules_installing.html
Puppet NX-OS マニフェストの例	[Cisco Network Puppet モジュールの例（Cisco Network Puppet Module Examples）]
NX-OS デベロッパーのランディング ページ。	構成管理ツール

第 13 章

Cisco NX-OS でのシェフクライアントの使用

シェフについて

Chef は、Chef Software、 Inc. によって開発されたオープンソース ソフトウェア パッケージです。ソフトウェアパッケージは、インフラストラクチャのサイズに関係なく、物理、仮想、またはクラウドの場所にサーバーとアプリケーションを導入する、システムおよびクラウドインフラストラクチャの自動化フレームワークです。各組織は、1 つ以上のワークステーション、単一サーバー、Chef クライアントが設定されていて、維持されているすべてのノードで構成されます。各ノードの設定方法について Chef クライアントに指示するために、クックブックとレシピが使用されます。すべてのノードにインストールされている Chef クライアントが、実際の設定を行います。

Chef クックブックは、設定とポリシーの配布の基本単位です。クックブックではシナリオを定義します。また、そのシナリオをサポートするために必要なすべての内容（ライブラリ、レシピ、ファイルなど）が含まれています。Chef レシピは、デバイスの状態を設定するためのプロパティ定義の集合です。これらのプロパティ状態の確認および設定の詳細は抽象化されているため、レシピは複数のオペレーティングシステムまたはプラットフォームで使用できます。レシピは、通常、構成時の設定を定義するために使用されますが、ソフトウェアパッケージのインストール、ファイルのコピー、およびサービスの開始などにも使用できます。

次のリファレンスは、Chef からの詳細情報を提供します。

トピック	リンク
Chef ホーム	https://www.chef.io
Chef の概要	https://docs.chef.io/chef_overview.html
Chef のドキュメント（すべて）	https://docs.chef.io/

前提条件

シェフの前提条件は次のとおりです：

- シェフの展開に必要なディスク ストレージがデバイス上に用意されている必要があります：
  - ブートフラッシュに最低 500 MB の空きディスク容量
- シェフ 12.4.1 以降のシェフ サーバが必要です。
- シェフ クライアント 12.4.1 以降が必要です。

Chef クライアント NX-OS 環境

chef-client ソフトウェアは、ゲスト シェル（CentOS を実行する Linux コンテナ環境）のスイッチにインストールする必要があります。このソフトウェアは、ホストから切り離された安全でオープンな実行環境を提供します。

Cisco NX-OS リリース 9.2(1) 以降、chef-client の Bash-shell（NX-OS の基盤となるネイティブ WindRiver Linux 環境）インストールはサポートされなくなりました。

次のドキュメントには、エージェント ソフトウェアのダウンロード、インストール、および手順ごとのガイダンスが記載されています。

トピック	リンク
Chefクライアント：Cisco Nexusプラットフォームでのインストールとセットアップ（手動セットアップ）	cisco-cookbook::README-install-agent.md
Chefクライアント：Cisco Nexusプラットフォームでのインストールとセットアップ（Chefプロビジョナを使用した自動インストール）	cisco-cookbook::README-chef-provisioning.md

cisco-cookbook

cisco-cookbook は、Chef レシピの抽象リソース構成と、Cisco Nexus スイッチの特定の実装の詳細との間の、シスコが開発したオープン ソース インターフェイスです。このクックブックは Chef Server にインストールされ、Cisco Nexus スイッチでの Chef Client の適切な動作に必要です。

cisco-cookbook は、Chef Supermarket にあります。

次のドキュメントには、cisco-cookbook および一般的なクックブックのインストール手順の詳細が記載されています。

トピック	リンク
cisco-cookbook の場所	Chef Supermarket Cisco クックブック
リソースの種類のカタログ	リソース カタログ（テクノロジー別）
cisco-cookbook: ソース コード リポジトリ	Cisco Network Chef クックブック
cisco-cookbook: セットアップと使用法	Chef クックブックのセットアップと使用方法
Chef Supermarket	Chef Supermarket
Chef NX-OS マニフェストの例	Cisco Network Chef クックブックのレシピ

第 14 章

Nexus アプリケーション開発：Yocto

Yocto について

Cisco NX-OS リリース 10.1(1) ソフトウェアは、Yocto 2.6 に基づいています。Yocto 2.6 をダウンロードし、構築する新しいソフトウェアをダウンロードし、ソフトウェアを構築し、ソフトウェアをスイッチにインストールするという方法によって、より多くのアプリケーションをインストールできます。

Yocto のインストール

次の例では、Ubuntu 16.04 仮想マシンで Ruby バージョン 2.2.2 をビルドします。

Step 1 Ubuntu 16.04 仮想マシンにすべての必須パッケージをインストールします。

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath
 socat cpio python python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping libsdl1.2-dev
 xterm
```

Step 2 Yocto 2.6 をダウンロードします。

```
wget http://downloads.yoctoproject.org/releases/yocto/yocto-2.6/poky-thud-20.0.0.tar.bz2
tar xjfv poky-thud-20.0.0.tar.bz2
cd poky-thud-20.0.0
```

Step 3 `oe-init-build-env` ファイルを取得します。

```
source oe-init-build-env
```

Step 4 テキスト エディタを使用して `conf/local.conf` を編集し、次の行を追加します。

```
MACHINE = "genericx86-64"
DEFAULTTUNE  = "x86-64"
```

Step 5 次のコマンドを入力します。

```
bitbake ruby
```

ビルドが完了すると、RPM は tmp/deploy/rpm/x86_64/*.rpm に生成されています。

Step 6 RPM をスイッチにコピーします。

```
Switch# copy scp://<username>@<IP_address>/ruby-2.2.2-r0.x86_64.rpm  bootflash: vrf management
use-kstack
Switch# copy scp://<username>@<IP_address>/libyaml-0-2-0.1.6-r0.x86_64.rpm  bootflash: vrf management
 use-kstack
```

Step 7 Bash シェルから、次のコマンドを入力します。

インストール プロセスのある時点で求められたら、**y** を入力します。

```
bash-4.3# dnf install /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
groups-repo                              | 1.1 kB     00:00 ...
localdb                                  |  951 B     00:00 ...
patching                                 |  951 B     00:00 ...
thirdparty                               |  951 B     00:00 ...
Setting up Install Process
Examining /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm: libyaml-0-2-0.1.6-r0.x86_64
Marking /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package libyaml-0-2.x86_64 0:0.1.6-r0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================================
 Package          Arch             Version         Repository                           Size
================================================================================================
Installing:
 libyaml-0-2      x86_64           0.1.6-r0        /libyaml-0-2-0.1.6-r0.x86_64      119 k

Transaction Summary
================================================================================================
Install       1 Package

Total size: 119 k
Installed size: 119 k
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : libyaml-0-2-0.1.6-r0.x86_64                                  1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link


Installed:
  libyaml-0-2-0.1.6-r0.x86_64
```

```
Complete!
Install operation 2450 completed successfully at Fri Jul 27 18:54:55 2018.

[####################] 100%
```

Step 8 次のコマンドは、Ubuntu 16.04 仮想マシンで Ruby バージョン 2.2.2 をビルドする例を示しています。

インストール プロセスのある時点で求められたら、**y** を入力します。

```
bash-4.3# dnf install /bootflash/ruby-2.2.2-r0.x86_64.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
groups-repo                              | 1.1 kB     00:00 ...
localdb                                  |  951 B     00:00 ...
patching                                 |  951 B     00:00 ...
thirdparty                               |  951 B     00:00 ...
thirdparty/primary                       | 1.8 kB     00:00 ...
thirdparty                                  2/2
Setting up Install Process
Examining /bootflash/ruby-2.2.2-r0.x86_64.rpm: ruby-2.2.2-r0.x86_64
Marking /bootflash/ruby-2.2.2-r0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ruby.x86_64 0:2.2.2-r0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================================
 Package          Arch             Version         Repository                          Size
================================================================================================
Installing:
 ruby             x86_64           2.2.2-r0        /ruby-2.2.2-r0.x86_64               32 M

Transaction Summary
================================================================================================
Install       1 Package

Total size: 32 M
Installed size: 32 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : ruby-2.2.2-r0.x86_64                                      1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link


Installed:
  ruby.x86_64 0:2.2.2-r0

Complete!
Install operation 2451 completed successfully at Fri Jul 27 18:55:23 2018.

[####################] 100%
```

第 15 章

Nexus アプリケーション開発：SDK

Cisco SDK について

Cisco SDK は、Yocto 2.0 に基づく開発キットです。Cisco NX-OS リリース 9.2(1) が動作する Cisco Nexus スイッチで実行するアプリケーションをビルドするためのすべてのツールが含まれています。基本コンポーネントは、多くのアプリケーションで一般的に使用されるCクロスコンパイラ、リンカ、ライブラリ、およびヘッダー ファイルです。リストは網羅的ではないため、特定のアプリケーションに必要な依存関係をダウンロードしてビルドすることが必要となる場合があります。一部のアプリケーションは、Cisco devhub Web サイトからダウンロードしてすぐに使用できるようになっていて、ビルドの必要はありません。SDKは、スイッチに直接インストールできるRPMパッケージをビルドするために使用できます。

SDK のインストール

以下にシステム要件を示します。

- SDK は、ほとんどの最新の 64 ビット x86_64 Linux システムで実行できます。CentOS 7 および Ubuntu 14.04 で検証済みです。Bash シェルで SDK をインストールして実行します。
- SDK には、32 ビット アーキテクチャと 64 ビット アーキテクチャの両方のバイナリが含まれているため、32 ビット ライブラリもインストールされている x86_64 Linux システムで実行する必要があります。

32 ビットライブラリがインストールされているかどうかを確認します。

例：

```
bash$ ls /lib/ld-linux.so.2
```

このファイルが存在する場合は、32 ビットライブラリがすでにインストールされています。それ以外の場合は、次のように 32 ビット ライブラリをインストールします。

- CentOS 7 の場合：

  ```
  bash$ sudo dnf install glibc.i686
  ```

- Ubuntu 14.04 の場合：

  ```
  bash$ sudo apt-get install gcc-multilib
  ```

インストールと環境の初期化の手順

SDK は https://devhub.cisco.com/artifactory/open-nxos/10.0.1/nx-linux-x86_64-nxos-rootfs-n9k-sup-toolchain-1.1.0.sh からダウンロードできます。

このファイルは自己解凍アーカイブで、SDK を任意のディレクトリにインストールできます。SDK のインストール ディレクトリへのパスの入力が求められます。

```
bash$ ./wrlinux-8.0.0.25-glibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Wind River Linux SDK installer version 8.0-n9000
================================================
Enter target directory for SDK (default: /opt/windriver/wrlinux/8.0-n9000):
You are about to install the SDK to "/opt/windriver/wrlinux/8.0-n9000". Proceed[Y/n]? Y
Extracting
SDK.....................................................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.

. environment-setup-corei7-64-nxos-linux
. environment-setup-corei7-32-nxosmllib32-linux

source environment-setup-corei7-64-nxos-linux
source environment-setup-corei7-32-nxosmllib32-linux
==============================
```

source environment-setup-x86-wrsmllib32-linux および **source environment-setup-x86_64-wrs-linux** コマンドを使用して、シェル環境に SDK 固有のパスを追加します。SDK で使用する予定のシェルごとに、SDK 固有のパスを追加します。SDK 固有のパスを追加することは、ビルド ツールとライブラリの正しいバージョンを使用するように SDK をセットアップする点で重要です。

Step 1 インストール先ディレクトリを参照します。

Step 2 Bash プロンプトで次のコマンドを入力します。

```
bash$ source environment-setup-x86-wrsmllib32-linux
bash$ source environment-setup-x86_64-wrs-linux
```

SDK を使用したアプリケーションの構築

一般的な Linux でのビルド プロセスの多くは、このシナリオで機能します。状況に最適な方法を使用してください。

アプリケーション パッケージのソースコードは、さまざまな方法で取得できます。たとえば、tar ファイル フォーマットで、またはパッケージが存在する git リポジトリからダウンロードして、ソースコードを取得できます。

次は最も一般的なケースの一例です。

（オプション）アプリケーション パッケージが標準の configure/make/make install を使用してビルドできるかどうかを確認します。

```
bash$ tar --xvzf example-app.tgz
bash$ mkdir example-lib-install
bash$ cd example-app/
bash$ ./configure --prefix=/path/to/example-app-install
bash$ make
bash$ make install
```

場合によっては、`./configure` スクリプトに追加のオプションを渡す必要があります。たとえば、必要なオプションのコンポーネントと依存関係を指定する場合などです。追加オプションを渡すかどうかは、構築するアプリケーションに完全に依存します。

例：Ganglia とその依存関係の構築

この例では、ganglia と、必要なサードパーティ ライブラリ（libexpat、libapr、および libconfuse）を作成します。

libexpat

```
bash$ wget 'http://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz'
bash$ mkdir expat-install
bash$ tar xvzf expat-2.1.0.tar.gz
bash$ cd expat-2.1.0
bash$ ./configure --prefix=/home/sdk-user/expat-install
bash$ make
bash$ make install
bash$ cd ..
```

libapr

```
bash$ wget 'http://www.eu.apache.org/dist/apr/apr-1.5.2.tar.gz'
bash$ mkdir apr-install
bash$ tar xvzf apr-1.5.2.tar.gz
bash$ cd apr-1.5.2
bash$ ./configure --prefix=/home/sdk-user/apr-install
bash$ make
bash$ make install
```

```
bash$ cd ..
```

libconfuse

（注） confuse には、./configure に追加の --enable-shared オプションが必要です。そうしないと、必要な共有ライブラリの代わりに静的にリンクされたライブラリがビルドされます。

```
bash$ wget 'http://savannah.nongnu.org/download/confuse/confuse-2.7.tar.gz'
bash$ mkdir confuse-install
bash$ tar xvzf confuse-2.7.tar.gz
bash$ cd confuse-2.7
bash$ ./configure --prefix=/home/sdk-user/confuse-install --enable-shared
bash$ make
bash$ make install
bash$ cd ..
```

ganglia

（注） 必要なすべてのライブラリの場所が ./configure に渡されます。

```
bash$ wget
'http://downloads.sourceforge.net/project/ganglia/ganglia%20monitoring%20core/3.7.2/ganglia-3.7.2.tar.gz'
bash$ mkdir ganglia-install
bash$ tar xvzf ganglia-3.7.2.tar.gz
bash$ cd ganglia-3.7.2
bash$ ./configure --with-libexpat=/home/sdk-user/expat-install
--with-libapr=/home/sdk-user/apr-install/bin/apr-1-config
--with-libconfuse=/home/sdk-user/confuse-install --prefix=/home/sdk-user/ganglia-install
bash$ make
bash$ make install
bash$ cd ..
```

RPM を使用したアプリケーションのパッケージ化

「make」を使用してアプリケーションが正常にビルドされたら、RPM にパッケージ化できます。

（注） **RPM および spec ファイル**

RPM パッケージフォーマットは、特定のアプリケーションの完全なインストールに必要なすべてのファイル（バイナリ、ライブラリ、構成、ドキュメントなど）をパッケージ化するように設計されています。したがって、RPM ファイルを作成するプロセスにはいくらかの手間がかかります。RPM ビルドプロセスを支援するために、ビルドプロセスに関するすべてを制御する .spec ファイルが使用されます。

（注） 多くのサードパーティ製アプリケーションは、パッケージ化されたソース コードを tarball にした形式で、インターネットから入手できます。多くの場合、これらの tarball には RPM ビルドプロセスに役立つ .spec ファイルが含まれています。残念ながら、これらの .spec ファイルの多くは、送信元コード自体ほど頻繁には更新されていません。さらに悪いことに、spec ファイルがまったくない場合もあります。このような場合、RPM を構築できるように、spec ファイルを編集するか、最初から作成する必要があります。

RPM ビルド環境の作成

SDK を使用して RPM をビルドする前に、RPM ビルド ディレクトリ構造を作成し、いくつかの RPM マクロを設定する必要があります。

Step 1 ディレクトリ構造を作成します：

```
bash$ mkdir rpmbuild
bash$ cd rpmbuild
bash$ mkdir BUILD RPMS SOURCES SPECS SRPMS
```

Step 2 上で作成したディレクトリ構造を指すように topdir マクロを設定します：

```
bash$ echo "_topdir ${PWD}" > ~/.rpmmacros
```

（注） この手順は、現在のユーザーがすでに設定されている .rpmmacros ファイルを持っていないことを前提としています。既存の .rpmmacros ファイルを変更するのが不便な場合は、すべての rpmbuild コマンドラインに次を追加できます：

```
--define "_topdir ${PWD}"
```

Step 3 RPM DB を更新します：

```
bash$ rm /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm/__db.*
bash$ rpm --rebuilddb
```

（注） SDK の rpm および rpmbuild ツールは、RPM データベースとして通常の `/var/lib/rpm` の代わりに `/path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm` を使用するように変更されました。この変更により、SDK を使用していない場合にホストの RPM データベースと競合することが回避され、root アクセスの必要性がなくなります。SDK のインストール後、この手順に従って SDK RPM データベースを再構築する必要があります。

一般的な RPM ビルド手順の使用

一般的な RPM ビルド手順は次のとおりです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app
tarball>
bash$ # determine location of spec file in tarball:
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

結果は RPMS/ に作成されるバイナリ RPM で、スイッチにコピーしてインストールできます。アプリケーションのインストール方法と構成には様々なバリエーションがあり得ます。これらの手順については、アプリケーションのドキュメントを参照してください。

この rpm ビルドとスイッチへのインストールは、アプリケーションをサポートするために必要なすべてのソフトウェアパッケージで必要です。SDK にまだ含まれていないソフトウェアの依存関係を満たすことが必要な場合は、ソースコードを取得して、依存関係のあるソフトウェアもビルドする必要があります。ビルド用のマシンでは、パッケージを手動でビルドして、依存関係を検証することができます。次に、最も一般的な手順の例を示します。

```
bash$ tar xkzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example-lib-install
bash$ make
bash$ make install
```

これらのコマンドは、ビルドファイル（バイナリ、ヘッダー、ライブラリなど）をインストールディレクトリに配置します。ここから、標準のコンパイラとリンカのフラグを使用して、依存関係を満たすための新しい場所を選択できます。ライブラリなどのランタイム コードがあれば、それらもスイッチにインストールする必要があるため、必要なランタイム コードを RPM にパッケージ化しなければなりません。

（注） Cisco devhub の Web サイトには、すでに RPM 形式にまとめられているサポート ライブラリが多数あります。

オプションのプラグインを使用しない collectd RPM の構築例

ソース tarball をダウンロードし、spec ファイルを抽出します。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
```

```
bash$ rm -rf collectd-5.5.0
```

この tarball には 4 つの spec ファイルがあります。Red Hat の spec ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

この仕様ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM を設定します。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用します。spec ファイルで以下のように編集します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

collectd には多数のオプション プラグインがあります。この spec ファイルは、デフォルトで多くのプラグインを有効にします。多くのプラグインには外部依存関係があるため、これらのプラグインを無効にするオプションを **rpmbuild** コマンド ラインに渡す必要があります。1 つの長いコマンド ラインを入力する代わりに、次のように Bash 配列でオプションを管理できます。

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>  rpmbuild_opts+=("--without")
>  rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

その後、次のように rpmbuild に渡され、ビルドおよび RPM パッケージ プロセス全体が開始されます。

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

結果として、RPMS ディレクトリに、collectd 用に作成された RPM ができています。

これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます:

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
```

オプションの Curl プラグインを使用した collectd の RPM のビルド例

collectd curl プラグインには、依存関係として libcurl があります。

RPM ビルド プロセス中にこのリンクの依存関係を満たすには、SDK で curl をダウンロードしてビルドする必要があります。

```
bash$ wget --no-check-certificate http://curl.haxx.se/download/curl-7.24.0.tar.gz
bash$ tar xkvf curl-7.24.0.tar.gz
```

```
bash$ cd curl-7.24.0
bash$ ./configure --without-ssl --prefix /path/to/curl-install
bash$ make
bash$ make install
bash$ cd ..
```

（注） curl バイナリとライブラリは、/path/to/curl-install にインストールされます。このディレクトリが存在しない場合は作成されるため、現在のユーザーの書き込み権限が必要です。次に、ソース tarball をダウンロードし、spec ファイルを抽出します。この手順は、プラグインがない場合の collectd の例とまったく同じです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この仕様ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM をセットアップします。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用する必要があるため、spec ファイルで次のように編集します。

（注） この tarball には 4 つの spec ファイルがあります。Red Hat の spec ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

この点は、前の例と違っています。collectd rpmbuild プロセスは、libcurl の場所を認識する必要があります。collectd の spec ファイルを編集して、以下を追加します。

SPECS/collectd.spec で文字列 *%configure* を検索します。この行とそれに続く行は、rpmbuild が ./configure スクリプトに渡すオプションを定義します。

次のオプションを追加します。

```
--with-libcurl=/path/to/curl-install/bin/curl-config \
```

次に、rpmbuild コマンドオプションを含む Bash アレイが再度構築されます。次の違いに留意してください。

- *curl* をビルドされないプラグインのリストから削除
- *--with curl=force* の追加

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
```

```
> amqp apache ascent bind curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>  rpmbuild_opts+=("--without")
>  rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=("--with")
bash$ rpmbuild_opts+=("curl=force")bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

それからこれは次のように rpmbuild に渡され、ビルドおよび RPM パッケージ プロセス全体が開始されます：

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

RPMs ディレクトリ内の結果の RPM には、collectd-curl も含まれるようになりました。これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
bash$ rpm --noparentdirs -i /bootflash/collectd-curl-5.5.0-1.ia32e.rpm
```

第 16 章

NX-SDK

NX-SDK について

Cisco NX-OS SDK（NX-SDK）は、自動化およびカスタム アプリケーションの作成（カスタムの生成など）のためのインフラストラクチャへのアクセスを合理化する C++ 抽象化およびプラグイン ライブラリ レイヤです。

- CLI
- Syslog
- イベント マネージャとエラー マネージャ
- アプリケーション間通信
- ハイ アベイラビリティ（HA）
- ルート マネージャ

NX-SDK を使用したアプリケーション開発には、C++、Python、または Go を使用できます。

ローカル（オン スイッチ）およびリモート（オフ スイッチ）アプリケーションのサポート

NX-SDK を使用して開発するアプリケーションは、Cisco Nexus スイッチではなく、NX-SDK が提供する Docker コンテナで作成または開発することになります（オフ スイッチ）。アプリケーションの作成後、アプリケーションを展開できる場所を柔軟に選択できます。

- ローカル（オンボックス）アプリケーションはスイッチ上で実行されます。詳細については、オンボックス（ローカル）アプリケーションについて （158 ページ）を参照してください。
- リモート（オフボックス）アプリケーションは、スイッチをオフにして実行されます。このオプションは NX-SDK 2.0 以降でサポートされており、アプリケーションをスイッチ以外の場所で実行するようにデプロイできます。詳細については、リモート アプリケーションについて （160 ページ）を参照してください。

関連情報

Cisco NX-SDK の詳細については、次にアクセスしてください。

- Cisco DevNet NX-SDK。サポートされるリリースごとの機能と詳細については、`versions.md` リンク（https://github.com/CiscoDevNet/NX-SDK/blob/master/versions.md） を参照してください。
- NX-SDK Readme

必要に応じて、シスコは NX-SDK の情報を GitHub に追加します。

Go バインディングに関する考慮事項

Goバインディングは、NX-SDKのリリースと、アプリがローカルで実行されているかリモートで実行されているかに応じて、さまざまなレベルでサポートされます。

- NX-SDK リモート アプリケーションのすべてのバージョンの Go バインディングは、EFT 前の品質です。
- ローカル NX-SDK 2.0 アプリケーションの Go バインディングは、EFT 前です。
- ローカル NX-SDK 1.7.5 以前のアプリケーションの Go バインディングがサポートされています。

詳細については、「NX-SDK アプリケーションの GO バインディング」を参照してください。

オンボックス（ローカル）アプリケーションについて

オンボックス（ローカル）アプリケーションでは、NX-SDK をインストールし、選択したサポート対象言語でアプリケーションをビルドし、スイッチにインストールできる `.rpm` ファイルとしてアプリをパッケージ化し、スイッチにアプリケーションをインストールして実行します。 。 `.rpm` ファイルは手動で生成することも、自動生成することもできます。

アプリケーション開発は、NX-SDK が提供するコンテナで行われます。ローカル アプリケーションには、リモート アプリケーションとは異なるコンテナとツールを使用します。詳細については、デフォルト Docker イメージ （159 ページ）を参照してください。

ローカルアプリケーションのビルド、インストール、および実行については、Cisco DevNet NX-SDK を参照してください。

デフォルト Docker イメージ

NX-SDK には、ローカルまたはリモートで使用するための次の Docker イメージとツールがデフォルトで含まれています。

使用方法	目次
スイッチの場合	Cisco ENXOS SDK クロス コンパイル用の Wind River Linux（WRL）ツール チェーン 多言語バインディング ツールキット NX-SDK 1.75 以降、Go コンパイラ
オフ スイッチ（リモート）	事前にビルドされた libnxsdk.so を備えた NX-SDK 多言語バインディング ツールキット Go コンパイラ RapidJSON リモート API サポートのための gRPC

詳細については、https://github.com/CiscoDevNet/NX-SDK#readmeを参照してください。

NX-SDK に関する注意事項と制限事項

NX-SDK には、アプリケーションをローカル（オンボックス）またはリモート（オフボックス）で実行するための使用上の注意事項と制限事項があります。

注意事項と制限事項については、Cisco DevNet NX-SDK の「役立つメモ」を参照してください。

NX-SDK2.0 について

NX-SDK バージョン 2.0 は、開発者が必要な場所でアプリケーションを実行できる実行環境の柔軟性を可能にします。このバージョンの NX-SDK では、アプリケーションは引き続きコンテナ内のスイッチをオフにして開発されますが、たとえばクラウドなどで、アプリケーションをスイッチ上またはスイッチ外のいずれかで実行できます。

NX-SDK 2.0 には次のような利点があります。

- スイッチをお客様の環境に簡単に統合できます。
- スイッチがデータセンター、パブリック クラウド、プライベート クラウドでシームレスに動作できるようにする拡張性。

- スイッチ レベルのリソースでの変更がアプリケーションの変更または書き換えを必要としないように、スイッチ リソースから顧客アプリを切り離します。
- アプリケーションがリンクするための使いやすい API を備えた単一のライブラリ。これにより、スイッチの相互作用が簡素化され、アプリケーションをより簡単に記述およびデバッグできる高水準言語で記述できます。
- リモート サービスの実行は、オンボックス アプリケーションよりも安全です。

詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

NX-SDK2.5 について

Cisco NX-OS Release 9.3(3) 以降では、ストリーミング Syslog 機能が追加されています。

詳細については、CiscoDevNetを参照してください。

表 *4 : syslog* イベント

機能	詳細
syslog イベント	• Cisco NX-OS syslog イベントに登録するカスタム アプリケーション向けの機能です。 • 詳細については、nx_trace.h の watchSyslog および postSyslogCb API を参照してください。

リモート アプリケーションについて

リモート アプリケーションは、Cisco Nexus スイッチではない別のスイッチに置くことができます。リモートまたはオフボックスのアプリケーションは、NX-SDK レイヤを介して呼び出し、スイッチと対話して情報の読み取り（取得）または情報の書き込み（設定）をします。

ローカルとリモートの両方の NX-SDK アプリケーションは同じ API を使用するため、NX-SDK アプリケーションをオンボックスまたはオフボックスで柔軟に展開できます。

リモートで実行するには、アプリケーションが特定の要件を満たしている必要があります。詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.mdを参照してください。

2.0 より前の NX-SDK アプリケーションの後方互換性

NX-SDK 2.0 には、NX-SDK v1.75 アプリケーションの開発方法に応じて、条件付きの後方互換性があります。

- 通常、NX-SDK は、NX-SDK 2.0 より前に作成したアプリのリモート実行をサポートしており、アプリを完全に書き直す必要はありません。代わりに、API 呼び出しを変更するために変更せずに、同じアプリを再利用できます。新しいNX-SDK 2.0 モデルで古いアプリをサポートするには、API 呼び出しで IP およびポートのパラメータを提供する必要があります。これらのパラメータは NX-SDK 1.75 以前では使用できませんが、アプリが SDK サーバにエクスポートできる環境変数として IP アドレスとポート情報を追加できます。
- ただし、NX-SDK 2.0 より前のアプリの下位互換性がサポートされていない場合があります。古いアプリの一部の API は、リモートでの実行をサポートしていないか、実行できない可能性があります。この場合、API は例外をスローできます。元のアプリケーションに対する例外処理の完全性と堅牢性によっては、アプリケーションが予期しない動作をする可能性があり、最悪の場合、クラッシュする可能性があります。

詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

NX-SDK セキュリティ

NX-OS 9.3（1）以降、NX-SDK 2.0 は次のセキュリティ機能をサポートしています。

- セッション セキュリティ。リモート アプリケーションは、トランスポート レイヤー サービス （TLS) を介してスイッチ上の NX SDK サーバに接続し、アプリケーションとスイッチの NX SDK サーバ間に暗号化されたセッションを提供できます。
- サーバ証明書のセキュリティ。Cisco NX-OS 9.3（1）を使用した新しいスイッチ展開の場合、NX-SDK サーバは 1 日限りの一時証明書を生成して、カスタム証明書をインストールするのに十分な時間を提供します。

  たとえば、以前の NX-SDK バージョンから NX-SDK 2.0 にアップグレードする場合など、NX-SDK サーバにカスタム証明書がすでにインストールされている場合、既存の証明書はアップグレード後も保持され、使用されます。
- API 書き込み呼び出し制御。NX-SDK 2.0 では、セキュリティ プロファイルが導入されています。これにより、アプリケーションが NX-SDK サーバをどの程度制御できるかを制御するための事前定義されたポリシーを選択できます。セキュリティ プロファイルに関する詳細情報を入手するには、NX SDK 2.0 のセキュリティ プロファイル（161 ページ） を参照します。

NX SDK 2.0 のセキュリティ プロファイル

以前のリリースでは、SDK バージョン 1.75 の API は、イベントのデータの読み取りと取得のみが許可されていました。Cisco NX-OS リリース 9.3(1) 以降、NX-SDK 2.0 は書き込みコールを含むさまざまなタイプの操作をサポートします。

アプリがスイッチを読み書きする機能は、セキュリティ プロファイルを介して制御できます。セキュリティ プロファイルは、スイッチで実行されているアプリケーションのサービスに付加され

るオプションのオブジェクトです。セキュリティ プロファイルは、スイッチに書き込むアプリケーションの機能を制御し、スイッチ機能を変更、削除、または構成するアプリケーションの機能を制御します。デフォルトでは、アプリケーションの書き込みは許可されていないため、アプリケーションごとに、スイッチへの書き込みアクセスを有効にするセキュリティ プロファイルを作成する必要があります。

Cisco の NX-SDK は、次のセキュリティ プロファイルを提供します。

プロファイル	説明	値
拒否	CLI の追加を除き、API 呼び出しがスイッチに書き込まれないようにします。	これはデフォルトプロファイルです。
スロットル	スイッチを変更する API を許可しますが、指定された数の呼び出しまでのみ許可します。このセキュリティ プロファイルは、スロットリングを適用して API 呼び出しの数を制御します。 アプリケーションは制限まで書き込むことができますが、制限を超えると書き込みが停止し、応答でエラー メッセージが送信されます。	スロットルは 50 回の API 呼び出しであり、スロットルは 5 秒後にリセットされます。
許可	スイッチを変更する API は制限なしで許可されます	

NX-SDK のセキュリティ プロファイルの詳細については、[NX-SDK アプリケーションのセキュリティ プロファイル（Security Profiles for NX-SDK Applications）] を参照してください。

アプリケーションの構築、インストール、および実行の詳細については、CiscoDevNet NX-SDK にアクセスしてください。

第 17 章

Cisco NX-OS での Docker の使用

Cisco NX-OS での Docker について

Docker は、すべての依存関係とライブラリと共にパッケージ化された、コンテナー内で安全に分離されたアプリケーションを実行する方法を提供します。Docker の詳細にを表示するために https://docs.docker.com/を参照してください。

Cisco NX-OS リリース 9.2（1）以降、スイッチ上の Cisco NX-OS 内で Docker を使用するためのサポートが追加されました。

スイッチに含まれる Docker のバージョンは CE 18.09.0 です。Docker デーモンはデフォルトでは実行されていません。手動で起動するか、スイッチの起動時に自動的に再起動するように設定する必要があります。

このセクションでは、スイッチ環境の特定のコンテキストで Docker を有効にして使用する方法について説明します。一般的な Docker の使用方法と機能の詳細については、https://docs.docker.com/にある Docker のドキュメントを参照してください。

Docker の注意事項と制限事項

次に、スイッチ上の Cisco NX-OS で Docker を使用するためのガイドラインと制限事項を示します。

- Docker でサードパーティの DHCPD サーバーを実行している場合、SVI と一緒に使用すると、クライアントに到達するオファーで問題が発生する可能性があります。可能な回避策は、ブロードキャスト応答を使用することです。
- Docker 機能は、少なくとも 8 GB のシステム RAM を備えた スイッチでサポートされています。

Cisco NX-OS 内で Docker コンテナを設定するための前提条件

スイッチの Cisco NX-OS で Docker を使用するための前提条件は次のとおりです：

- ホスト Bash シェルを有効にします。スイッチの Cisco NX-OS で Docker を使用するには、ホスト Bash シェルのルート ユーザーである必要があります：

```
switch# configure terminal
    Enter configuration commands, one per line. End with CNTL/Z.
    switch(config)# feature bash-shell
```

- スイッチが HTTP プロキシ サーバを使用するネットワーク内にある場合、`http_proxy` と `https_proxy` 環境変数を `/etc/sysconfig/docker` に構成する必要があります。例：

```
export http_proxy=http://proxy.esl.cisco.com:8080
export https_proxy=http://proxy.esl.cisco.com:8080
```

- スイッチのクロックが正しく設定されていることを確認してください。そうしないと、次のエラー メッセージが表示される場合があります：

```
x509: certificate has expired or is not yet valid
```

- ドメイン名とネーム サーバがネットワークに対して適切に構成されていること、および `/etc/resolv.conf` ファイルに反映されていることを確認します：

```
switch# conf t
     Enter configuration commands, one per line. End with CNTL/Z.
     switch(config)# vrf context management
     switch(config-vrf)# ip domain-name ?
     WORD Enter the default domain (Max Size 64)

     switch(config-vrf)# ip name-server ?
     A.B.C.D Enter an IPv4 address
     A:B::C:D Enter an IPv6 address

root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
```

```
nameserver 171.70.168.183 #bleed
root@switch#
```

Docker デーモンの開始

初めて Docker デーモンを開始すると、固定サイズのバックエンド ストレージ スペースがブートフラッシュの dockerpart と呼ばれるファイルに切り出され、次に /var/lib/docker にマウントされます。必要に応じて、Docker デーモンを初めて開始する前に /etc/sysconfig/docker を編集して、この領域のデフォルト サイズを調整できます。後で説明するように、必要に応じてこのストレージ スペースのサイズを変更することもできます。

Docker デーモンを開始するには：

Step 1 Bash を読み込み、スーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 Docker デーモンを起動します。

```
root@switch# service docker start
```

Step 3 ステータスをチェックします。

```
root@switch# service docker status
dockerd (pid 3597) is running...
root@switch#
```

（注） Docker デーモンを起動したら、ブートフラッシュ の dockerpart ファイルを削除したり、改ざんしたりしないでください。これは、docker の機能にとって重要であるからです。

```
switch# dir bootflash:dockerpart
2000000000 Mar 14 12:50:14 2018 dockerpart
```

自動的に起動するように Docker を構成する

スイッチの起動時に常に自動的に起動するように Docker デーモンを構成できます。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 chkconfig ユーティリティを使用して、Docker サービスを永続化します。

```
root@switch# chkconfig --add docker
root@n9k-2#
```

Step 3 chkconfig ユーティリティを使用して、Docker サービスの設定を確認します。

```
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch#
```

Step 4 Docker が自動的に起動しないように構成を削除するには:

```
root@switch# chkconfig --del docker
root@switch# chkconfig --list | grep docker
root@switch#
```

Docker コンテナの開始: ホスト ネットワーク モデル

Docker コンテナがデータ ポートと管理を含むすべてのホスト ネットワーク インターフェイスにアクセスできるようにする場合は、--network ホスト オプションを使用して Docker コンテナを起動します。コンテナ内のユーザーは、ip netns exec <net_namespace> <cmd> を使用して、/var/run/netns（Cisco NX-OS で設定されたさまざまな VRF に対応）でさまざまなネットワーク名前空間を切り替えることができます。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 Docker コンテナを開始します。

以下は、スイッチで Alpine Docker コンテナを起動し、すべてのネットワーク インターフェイスを表示する例です。コンテナは、デフォルトで管理ネットワークの名前空間で起動されます。

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm --network
host --cap-add SYS_ADMIN -it alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
management
default
/ #
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
```

```
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
3: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default
link/gre 0.0.0.0 brd 0.0.0.0
...
/ #
/ # ip netns exec default ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/16 scope host lo
valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
link/ether 42:0d:9b:3c:d4:62 brd ff:ff:ff:ff:ff:ff
3: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
...
```

Docker コンテナの開始: ブリッジ型ネットワーク モデル

Dockerコンテナに外部ネットワーク接続（通常は管理インターフェースを介して）のみを許可し、特定のデータ ポートまたは他のスイッチインターフェースへの可視性を必ずしも気にしない場合は、デフォルトのDockerブリッジ ネットワーク モデルでDockerコンテナを開始できます。 これは、ネットワーク名前空間の分離も提供するため、前のセクションで説明したホスト ネットワーキング モデルよりも安全です。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 Docker コンテナを開始します。

以下は、スイッチで Alpine Docker コンテナを開始し、`iproute2` パッケージをインストールする例です。

```
root@switch# docker run -it --rm alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
/ #
```

Step 3 ユーザー名前空間の分離を設定するかどうかを決定します。

ブリッジ ネットワーク モデルを使用するコンテナの場合、ユーザー名前空間の分離を設定して、セキュリティをさらに向上させることもできます。詳細については、「ユーザー[名前空間（namespace）]の分離による Docker コンテナの保護 （173 ページ）」を参照してください。

標準の Docker ポート オプションを使用して、sshd などのコンテナー内からサービスを公開できます。例：

```
root@switch# docker run -d -p 18877:22 --name sshd_container sshd_ubuntu
```

これにより、コンテナ内のポート 22 がスイッチのポート 18877 にマップされます。次の例に示すように、ポート 18877 を介してサービスに外部からアクセスできるようになりました。

```
root@ubuntu-vm# ssh root@ip_address -p 18887
```

Dockerコンテナでのブートフラッシュおよび揮発性パーティションのマウント

Docker コンテナの run コマンドで `-v /bootflash:/bootflash` および `-v /volatile:/volatile` オプションを渡すことで、ブートフラッシュおよび揮発性パーティションを Docker コンテナに表示できます。これは、新しい NX-OS システム イメージをブートフラッシュにコピーするなど、コンテナ内のアプリケーションがホストと共有するファイルにアクセスする必要がある場合に役立ちます。

（注） この **-v** コマンドオプションを使用すると、任意のディレクトリをコンテナにマウントでき、NX-OS システムの動作に影響を与える可能性のある情報漏えいやその他のアクセスが発生する可能性があります。これを、NX-OS CLI を使用してすでにアクセス可能な `/bootflash` や `/volatile` などのリソースに制限します。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 Docker コンテナの実行コマンドに `-v /bootflash:/bootflash` および `-v /volatile:/volatile` オプションを渡します。

```
root@switch# docker run -v /bootflash:/bootflash -v /volatile:/volatile -it --rm alpine
/# ls /
bin         etc         media       root        srv         usr
bootflash   home        mnt         run         sys         var
dev         lib         proc        sbin        tmp         volatile
/ #
```

拡張 ISSU スイッチオーバーでの Docker デーモンの永続性の有効化

Docker デーモンと実行中のコンテナの両方を拡張 ISSU スイッチオーバーで持続させることができます。これが可能なのは、バックエンドの Docker ストレージが存在するブートフラッシュが同じであり、アクティブ スーパーバイザとスタンバイ スーパーバイザの両方で共有されるためです。

Docker コンテナは、切り替え中に中断（再起動）されるため、継続的に実行されません。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 スイッチオーバーを開始する前に、`chkconfig` ユーティリティを使用して Docker サービスを永続化します。

```
root@switch# chkconfig --add docker
root@n9k-2#
```

Step 3 スイッチオーバー後にコンテナが自動的に再起動されるように、`--restart without-stopped` オプションを使用してコンテナを起動します。

次の例では、Alpine コンテナを開始し、明示的に停止するか、Docker を再起動しない限り、常に再起動するように構成します。

```
root@switch# docker run -dit --restart unless-stopped alpine
root@n9k-2#
```

Docker コンテナは、切り替え中に中断（再起動）されるため、継続的に実行されません。

Cisco Nexus Platform Switches Switchover 時に Docker デーモンの永続性を有効にする

Docker デーモンと実行中のコンテナの両方を、個別のブートフラッシュ パーティションを持つ 2 つの個別の物理スーパーバイザ間のスイッチオーバーで持続させることができます。ただし、Cisco Nexus スイッチの場合、両方のスーパーバイザのブートフラッシュ パーティションは物理的に分離されています。したがって、スイッチオーバーを実行する前に、`dockerpart` ファイルをスタンバイ スーパーバイザに手動でコピーする必要があります。

Step 1 Bash を読み込みしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 スイッチオーバー後にコンテナが自動的に再起動されるように、`--restart without-stopped`オプションを使用してコンテナを起動します。

次の例では、Alpineコンテナを開始し、明示的に停止するか、Dockerを再起動しない限り、常に再起動するように構成します。

```
root@switch# docker run -dit --restart unless-stopped alpine
root@n9k-2#
```

Dockerコンテナは切り替え中に中断（再起動）されるため、継続的に実行されないことに注意してください。

Step 3 スイッチオーバーを開始する前に、`chkconfig`ユーティリティを使用してDockerサービスを永続化します。

```
root@switch# chkconfig --add docker
root@n9k-2#
```

Step 4 Dockerバックエンドストレージパーティションを現用系からスタンバイ スーパーバイザ ブートフラッシュにコピーします。

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown

root@switch# cp /bootflash/dockerpart /bootflash_sup-remote/

root@switch# service docker start
```

Dockerストレージ バックエンドのサイズ変更

Dockerデーモンを起動または使用した後、必要に応じてDockerバックエンド ストレージ スペースのサイズを増やすことができます。

Step 1 Guest Shell を無効にします。

ゲスト シェルを無効にしないと、サイズ変更が妨げられる可能性があります。

```
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want to disable
 the guest shell? (y/n) [n] y
switch# 2018 Mar 15 17:16:55 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating virtual
service 'guestshell+'
2018 Mar 15 17:16:57 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated virtual
 service 'guestshell+'
```

Step 2 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 3 現在利用可能なストレージ容量に関する情報を取得します。

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 1.9G 7.6M 1.8G 1% /var/lib/docker
root@n9k-2#
```

Step 4 Dockerデーモンを停止します。

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

Step 5 Dockerバックエンド ストレージ スペース（/bootflash/dockerpart）の現在のサイズに関する情報を取得します。

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2000000000 Mar 15 16:53 /bootflash/dockerpart
root@n9k-2#
```

Step 6 Dockerバックエンド ストレージ スペースのサイズを変更します。

たとえば、次のコマンドはサイズを500メガバイト増やします。

```
root@switch# truncate -s +500MB /bootflash/dockerpart
root@n9k-2#
```

Step 7 Dockerバックエンド ストレージ スペースのサイズに関する最新情報を取得して、サイズ変更プロセスが正常に完了したことを確認します。

たとえば、次の出力は、Dockerバックエンドストレージのサイズが500メガバイト増加したことを確認します。

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2500000000 Mar 15 16:54 /bootflash/dockerpart
root@n9k-2#
```

Step 8 /bootflash/dockerpart のファイル システムのサイズを確認します。

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/122160 files (0.6% non-contiguous), 17794/488281 blocks
```

Step 9 /bootflash/dockerpart のファイル システムのサイズを変更します。

```
root@switch# /sbin/resize2fs /bootflash/dockerpart
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /bootflash/dockerpart to 610351 (4k) blocks.
The filesystem on /bootflash/dockerpart is now 610351 blocks long.
```

Step 10 /bootflash/dockerpart のファイル システムのサイズを再度チェックして、ファイル システムのサイズが正常に変更されたことを確認します。

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
```

```
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/154736 files (0.6% non-contiguous), 19838/610351 blocks
```

Step 11 Daemon デーモンを再起動します。

```
root@switch# service docker start
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Starting dockerd with args '--debug=true':
```

Step 12 使用可能なストレージ領域の大きさを確認します。

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 2.3G 7.6M 2.3G 1% /var/lib/docker
```

Step 13 BASH シェルを終了します。

```
root@switch# exit
logout
switch#
```

Step 14 Guest Shell を有効にします。

```
switch# guestshell enable

switch# 2018 Mar 15 17:12:53 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual service
 'guestshell+'
switch# 2018 Mar 15 17:13:18 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```

Docker デーモンの停止

Docker を今後使用しない場合は、このトピックの手順に従って Docker デーモンを停止します。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 Docker デーモンを停止します。

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

Step 3 Docker デーモンが停止していることを確認します。

```
root@switch# service docker status
dockerd is stopped
root@switch#
```

（注） 必要に応じて、この時点でブートフラッシュの dockerpart ファイルを削除することもできます。

```
switch# delete bootflash:dockerpart
Do you want to delete "/dockerpart" ? (yes/no/abort) y
switch#
```

Docker コンテナ セキュリティ

Docker コンテナのセキュリティに関する推奨事項は次のとおりです。

- 可能であれば、別のユーザー［名前空間（namespace）］で実行します。
- 可能であれば、別のネットワーク［名前空間（namespace）］で実行します。
- cgroup を使用して技術情報を制限します。既存の cgroup（ ext_ser ）が作成され、ホストされているアプリケーションを、プラットフォーム チームがスイッチで実行される追加のソフトウェアに対して妥当と見なしたものに制限します。Docker では、これを使用して、コンテナごとの技術情報を制限できます。
- 不要な POSIX 機能を追加しないでください。

ユーザー[名前空間（namespace）]の分離による Docker コンテナの保護

ブリッジネットワークモデルを使用するコンテナの場合、ユーザー名前空間の分離を設定して、セキュリティをさらに向上させることもできます。詳細については、「https://docs.docker.com/engine/security/userns-remap/」を参照してください。

Step 1 システムに dockremap グループがすでに存在するかどうかを確認します。

dockremap ユーザーは、デフォルトでシステムにすでに設定されている必要があります。dockremap グループがまだ存在しない場合は、次の手順に従って作成します。

a) 次のコマンドを入力して dockremap グループを作成します。

```
root@switch# groupadd dockremap -r
```

b) dockremap ユーザーを作成します（まだ存在していない場合）。

```
root@switch# useradd dockremap -r -g dockremap
```

c) dockremap グループと dockremap ユーザーが正常に作成されたことを確認します。

```
root@switch# id dockremap
uid=999(dockremap) gid=498(dockremap) groups=498(dockremap)
root@switch#
```

Step 2 再マップされた必要な ID と範囲を /etc/subuid と /etc/subgid に追加します。

例：

```
root@switch# echo "dockremap:123000:65536" >> /etc/subuid
root@switch# echo "dockremap:123000:65536" >> /etc/subgid
```

Step 3 テキスト エディタを使用して、--userns-remap=default オプションを /etc/sysconfig/docker ファイルの other_args フィールドに追加します。

例：

```
other_args="-debug=true --userns-remap=default"
```

Step 4 [サービス ドッカー [re]start（service docker [re]start）] を使用して、Docker デーモンを再起動するか、まだ実行されていない場合は起動します。

例：

```
root@switch# service docker [re]start
```

ユーザー名前空間の分離によるコンテナの構成と使用の詳細については、https://docs.docker.com/engine/security/userns-remap/で Docker のドキュメントを参照してください。

cgroup パーティションの移動

サードパーティ サービスの cgroup パーティションは ext_ser で、CPU 使用率をコアあたり 25% に制限します。この ext_ser パーティションで Docker コンテナを実行することをお勧めします。

--cgroup-parent=/ext_ser/ オプションを指定せずに Docker コンテナを実行すると、最大 100% のホスト CPU アクセスが可能になり、Cisco NX-OS の通常の動作を妨げる可能性があります。

Step 1 Bash をロードしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

Step 2 ext_ser パーティションで Docker コンテナを実行します。

例：

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm --network
host --cgroup-parent=/ext_ser/ --cap-add SYS_ADMIN -it alpine
/ #
```

Docker のトラブルシューティング

これらのトピックでは、Docker コンテナで発生する可能性のある問題について説明し、考えられる解決策を提供します。

Docker の起動が機能不全になる

[問題：（Problem:）] Docker の起動に失敗し、次のようなエラー メッセージが表示されます：

```
switch# run bash
bash-4.3$ service docker start
Free bootflash: 39099 MB, total bootflash: 51771 MB
Carving docker bootflash storage: 2000 MB
2000+0 records in
2000+0 records out
2000000000 bytes (2.0 GB) copied, 22.3039 s, 89.7 MB/s
losetup: /dev/loop18: failed to set up loop device: Permission denied
mke2fs 1.42.9 (28-Dec-2013)
mkfs.ext4: Device size reported to be zero.  Invalid partition specified, or
        partition table wasn't reread after running fdisk, due to
        a modified partition being busy and in use.  You may need to reboot
        to re-read your partition table.

Failed to create docker volume
```

[考えられる原因：（Possible Cause:）] root ユーザーではなく、管理ユーザーとして Bash を実行している可能性があります。

[解決策：（Solution:）] root ユーザーではなく、管理ユーザーとして Bash を実行しているかどうかを確認します。

```
bash-4.3$ whoami
admin
```

Bash を終了し、ルート ユーザーとして Bash を実行します：

```
bash-4.3$ exit
switch# run bash sudo su -
```

ストレージが不足しているため、Docker が起動に失敗する

問題：ブートフラッシュ ストレージが不足しているため、Docker の起動に失敗し、次のようなエラー メッセージが表示されます。

```
root@switch# service docker start
Free bootflash: 790 MB, total bootflash: 3471 MB
Need at least 2000 MB free bootflash space for docker storage
```

考えられる原因：十分な空きブートフラッシュ ストレージがない可能性があります。

解決策：スペースを解放するか、必要に応じて /etc/sysconfig/docker の変数 *_dockerstrg* 値を調整してから、Docker デーモンを再起動します。

```
root@switch# cat /etc/sysconfig/docker
# Replace the below with your own docker storage backend boundary value (in MB)
# if desired.
boundary_dockerstrg=5000

# Replace the below with your own docker storage backend values (in MB) if
# desired. The smaller value applies to platforms with less than
# $boundary_dockerstrg total bootflash space, the larger value for more than
# $boundary_dockerstrg of total bootflash space.
small_dockerstrg=300
large_dockerstrg=2000
```

Docker Hub からのイメージのプルの失敗（509 証明書失効 エラー メッセージ）

問題：システムがDockerハブからイメージをプルできず、次のようなエラーメッセージが表示されます。

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: x509: certificate has
expired or is not yet valid
```

[考えられる原因：（Possible Cause:）]システム クロックが正しく設定されていない可能性があります。

[解決策：（Solution:）]クロックが正しく設定されているかどうかを確認します。

```
root@n9k-2# sh clock
15:57:48.963 EST Thu Apr 25 2002
Time source is Hardware Calendar
```

必要に応じて、時計をリセットします：

```
root@n9k-2# clock set hh:mm:ss { day month | month day } year
```

例：

```
root@n9k-2# clock set 14:12:00 10 feb 2018
```

Docker Hubからのイメージのプルの失敗（クライアントタイムアウト エラー メッセージ）

問題：システムがDockerハブからイメージをプルできず、次のようなエラー メッセージが表示されます。

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled
 while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

考えられる原因：プロキシまたは DNS 設定が正しく設定されていない可能性があります。

解決策：プロキシ設定を確認し、必要に応じて修正してから、Docker デーモンを再起動します。

```
root@switch# cat /etc/sysconfig/docker | grep proxy
#export http_proxy=http://proxy.esl.cisco.com:8080
#export https_proxy=http://proxy.esl.cisco.com:8080
root@switch# service docker [re]start
```

DNS 設定を確認し、必要に応じて修正してから、Docker デーモンを再起動します。

```
root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch# # conf t
     Enter configuration commands, one per line. End with CNTL/Z.
     switch(config)# vrf context management
     switch(config-vrf)# ip domain-name ?
     WORD Enter the default domain (Max Size 64)

     switch(config-vrf)# ip name-server ?
     A.B.C.D Enter an IPv4 address
     A:B::C:D Enter an IPv6 address
root@switch# service docker [re]start
```

スイッチのリロードまたはスイッチオーバーでDockerデーモンまたはコンテナが実行されない

問題：スイッチのリロードまたはスイッチオーバーを実行した後、Docker デーモンまたはコンテナーが実行されません。

考えられる原因：Docker デーモンが、スイッチのリロードまたはスイッチオーバーで持続するように構成されていない可能性があります。

解決策：Dockerデーモンが `chkconfig` コマンドを使用してスイッチのリロードまたはスイッチオーバーで持続するように構成されていることを確認してから、`--restart unless-stopped` オプションを使用して必要なDockerコンテナを開始します。たとえば、Alpine コンテナを開始するには：

```
root@switch# chkconfig --add docker
root@switch#
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

```
root@switch# docker run -dit --restart unless-stopped alpine
```

Dockerストレージ バックエンドのサイズ変更が失敗する

問題：Dockerバックエンド ストレージのサイズを変更しようとして失敗しました。

考えられる原因：ゲスト シェルが無効になっていない可能性があります。

解決策：次のコマンドを使用して、ゲスト シェルが無効になっているかどうかを確認します。

```
root@switch# losetup -a | grep dockerpart
root@n9k-2#
```

ゲスト シェルが無効になっている場合、コマンドは出力を表示しません。

必要に応じて、次のコマンドを入力してゲスト シェルを無効にします。

```
switch# guestshell disable
```

それでもDockerバックエンド ストレージのサイズを変更できない場合は、`/bootflash/dockerpart`を削除し、`/etc/sysconfig/docker`の`[small_]large_dockerstrg`を調整してから、Dockerを再度起動して、必要なサイズの新しいDockerパーティションを取得します。

Dockerコンテナがポートで着信トラフィックを受信しない

問題：Dockerコンテナがポートで着信トラフィックを受信しません。

考えられる原因：Dockerコンテナがkstackポートではなくnetstackポートを使用している可能性があります。

解決策：Dockerコンテナによって使用されるエフェメラル ポートがkstackの範囲内にあることを確認します。そうしないと、着信パケットがサービスのためにnetstackに送信され、ドロップされる可能性があります。

```
switch# show socket local-port-range
Kstack local port range (15001 - 58000)
Netstack local port range (58001 - 63535) and nat port range (63536 - 65535)
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# sockets local-port-range <start_port> <end_port>
switch# run bash sudo su -
root@switch# cat /proc/sys/net/ipv4/ip_local_port_range
15001   58000
root@switch#
```

Dockerコンテナでデータ ポートと/または管理インターフェイスを表示できません

問題： Dockerコンテナにデータ ポートまたは管理インターフェースが表示されません。

解決方法：

- `-v /var/run/netns:/var/run/netns:ro,rslave --network host` オプションを使用して、すべてのホスト名前空間がマップされたホストネットワーク名前空間でDockerコンテナが開始されていることを確認します。
- コンテナに入ると、デフォルトで管理ネットワークの名前空間に入ります。`ip netns` ユーティリティを使用して、データポートインターフェイスを持つデフォルト（`init`）ネットワーク名前空間に移動できます。`ip netns` ユーティリティは、dnf、apk などを使用してコンテナにインストールする必要がある場合があります。

一般的なトラブルシューティングのヒント

[問題：（Problem:）]他のトラブルシューティングプロセスを使用しても解決されなかったDockerコンテナに関する他の問題があります。

解決方法：

- `/var/log/docker` で dockerd デバッグ出力を探して、何が問題なのかの手掛かりを見つけてください。
- スイッチに 8 GB 以上の RAM があることを確認します。Docker 機能は、RAM が 8 GB 未満のスイッチではサポートされていません。

第 III 部

NX-API

第 18 章

NX-API CLI

NX-API CLI について

NX-API CLI は、XML 出力をサポートする Cisco NX-OS CLI システムの拡張機能です。NX-API CLI は、特定のコマンドの JSON 出力フォーマットもサポートしています。

Cisco Nexus スイッチでは、コマンドライン インターフェイス（CLI）はスイッチ上でのみ実行されます。NX-API CLI は HTTP / HTTPS を使ってスイッチの外部で CLI を使用できるようにすることで、これらの CLI のユーザー補助を改善します。この拡張機能をスイッチの既存の Cisco NX-OS CLI システムに使用できます。NX-API CLI は **show** コマンド、構成と Linux Bash をサポートします。

NX-API CLI は JSON-RPC をサポートしています。

注意事項と制約事項

NX-API CLI は、スイッチで Cisco NX-OS CLI を実行するために VSH を生成します。VSH のタイムアウトは 5 分です。Cisco NX-OS CLI の実行に 5 分以上かかると、コマンドは失敗し、「Back-end processing error.」というメッセージが表示されます。これは、NX-API コマンドのタイムアウトによって制御されます。これは、NX-API を介して要求されたコマンドを実行できる時間を制御します。この値は 300 秒に固定されており、変更できません。

転送

NX-APIは、転送のように HTTP または HTTPS を使用します。CLI は、HTTP / HTTPS POST 本文にエンコードされます。

Cisco NX-OS リリース 9.2(1) 以降、NX-API 機能は HTTPS ポート 443 でデフォルトで有効になっています。HTTP ポート 80 は無効です。

NX-APIは、ホスト上でネイティブに、またはゲストシェル内で実行されるアプリケーションの、UNIX ドメイン ソケットを介してサポートされます。

NX-API バックエンドは Nginx HTTP サーバを使用します。Nginx プロセスとそのすべての子プロセスは、CPU とメモリの使用量が制限されている Linux cgroup 保護下にあります。NX-API プロセスは、cgroup ext_ser_nginx の一部であり、2,147,483,648 バイトのメモリに制限されています。Nginx のメモリ使用量が cgroup の制限を超えると、Nginx プロセスは再起動されて、NX-API 構成（VRF、ポート、証明書構成）が復元されます。

メッセージ形式

（注）
- NX-API XML 出力は、情報を使いやすいフォーマットで表示します。
- NX-API XML は、Cisco NX-OS NETCONF 導入に直接マッピングされません。
- NX-API XML 出力は、JSON に変換できます。

セキュリティ

- NX-API は HTTPS をサポートします。HTTPS を使用すると、デバイスへのすべての通信が暗号化されます。
- NX-API は、デフォルトでは非セキュア HTTP をサポートしていません。
- NX-API は、デフォルトでは弱い TLSv1 プロトコルをサポートしていません。

NX-API は、デバイスの認証システムに統合されています。ユーザーは、NX-API を介してデバイスにアクセスするための適切なアカウントを持っている必要があります。NX-API では HTTP basic 認証が使用されます。すべてのリクエストには、HTTP ヘッダーにユーザー名とパスワードが含まれている必要があります。

（注） ユーザーのログイン資格情報を保護するには、HTTPS の使用を検討する必要があります。

[機能（feature）] マネージャ CLI コマンドを使用して、NX-API を有効にすることができます。NX-API はデフォルトで無効になっています。

NX-API は、ユーザーが最初に認証に成功したときに、セッション ベースの Cookie、**nxapi_auth** を提供します。セッション Cookie を使用すると、デバイスに送信される後続のすべての NX-API 要求にユーザー名とパスワードが含まれます。ユーザー名とパスワードは、完全な認証プロセスの再実行をバイパスするために、セッション Cookie で使用されます。セッション Cookie が後続の要求に含まれていない場合は、別のセッション Cookie が必要であり、認証プロセスによって提

供されます。認証プロセスの不必要な使用を避けることで、デバイスのワークロードを軽減できます。

（注） **nxapi_auth** cookie は 600 秒（10 分）で期限切れになります。この値は固定されており、調整できません。

（注） NX-API は、スイッチ上の Programmable Authentication Module（PAM）を使用して認証を行います。cookie を使用して PAM の認証数を減らし、PAM の負荷を減らします。

NX-API CLI の使用

Cisco Nexus 9000 シリーズ スイッチのコマンド、コマンド タイプ、および出力タイプは、CLI を HTTP/HTTPS POST の本文にエンコードすることにより、NX-API を使用して入力されます。要求に対する応答は、XML または JSON 出力形式で返されます。

（注） NX-API 応答コードの詳細については、NX-API 応答コードの表（208 ページ）を参照してください。

NX-API CLI は、ローカル アクセスに対してはデフォルトで有効になっています。リモート HTTP アクセスに対してはデフォルトで無効になっています。

次の例は、NX-API CLI を構成して起動する方法を示しています。

- 管理インターフェイスを有効にします。

```
switch# conf t
Enter configuration commands, one per line.
End with CNTL/Z.
switch(config)# interface mgmt 0
switch(config-if)# ip address 10.126.67.53/25
switch(config-if)# vrf context managment
switch(config-vrf)# ip route 0.0.0.0/0 10.126.67.1
switch(config-vrf)# end
switch#
```

- NX-API **nxapi** 機能を有効にします。

```
switch# conf t
switch(config)# feature nxapi
```

次の例は、リクエストとそのレスポンスを XML 形式で示しています。

要求:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ins_api>
```

```
  <version>0.1</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>session1</sid>
  <input>show switchname</input>
  <output_format>xml</output_format>
</ins_api>
```

応答：

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

次の例は、JSON 形式の要求とその応答を示しています。

要求:

```
{
    "ins_api": {
        "version": "0.1",
        "type": "cli_show",
        "chunk": "0",
        "sid": "session1",
        "input": "show switchname",
        "output_format": "json"
    }
}
```

応答：

```
{
    "ins_api": {
        "type": "cli_show",
        "version": "0.1",
        "sid": "eoc",
        "outputs": {
            "output": {
                "body": {
                    "hostname": "switch"
                },
                "input": "show switchname",
                "msg": "Success",
                "code": "200"
            }
        }
    }
}
```

（注） ユーザーを削除しようとすると失敗し、次のようなエラー メッセージが約 12 時間ごとに表示されるという既知の問題があります。

```
user delete failed for username:userdel: user username is currently logged in - securityd
```

この問題は、NX-APIを介してスイッチにログインしているユーザーを削除しようとした場合に発生する可能性があります。この場合、次のコマンドを入力して、最初にユーザーのログアウトを試行します。

```
switch(config)# clear user username
```

その後、ユーザーの削除を再試行します。回避策を試みても問題が解決しない場合は、Cisco TAC へお問い合わせください。

NX-API で権限を root にエスカレーションする

NX-API では、管理者ユーザーの権限を root アクセスの権限にエスカレーションできます。

以下は、権限をエスカレーションするためのガイドラインです：

- 特権を root にエスカレーションできるのは管理者ユーザーのみです。
- root へのエスカレーションはパスワードで保護されています。

次の例は、管理者の権限を root にエスカレーションする方法と、エスカレーションを確認する方法を示しています。root になっても、**whoami** コマンドを実行すると admin として表示されることに注意してください。ただし、admin アカウントにはすべての root 権限があります。

最初の例：

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo su root ; whoami</input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>admin </body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

2 番目の例：

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo cat path_to_file </input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>[Contents of file]</body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

NX-API 管理コマンド

次の表にリストされている CLI コマンドを使用して、NX-API を有効にして管理できます。

表 *5 : NX-API* 管理コマンド

NX-API 管理コマンド	説明
feature nxapi	NX-API を有効化します。
no feature nxapi	NX-API を無効化します。
nxapi {**http** | **https**} **port** *port*	ポートを指定します。
no nxapi {**http** | **https**}	HTTP / HTTPS を無効化します。
show nxapi	ポートと証明書情報を表示します。 （注） 「**show nxapi**」コマンドは、network-operator ロールの証明書/構成情報を表示しません。

NX-API 管理コマンド	説明

nxapi certificate {httpscrt certfile | **httpskey keyfile**} *filename*	次のアップロードを指定します：
	• httpscrt が指定されている場合の HTTPS 証明書。
	• httpskey が指定されている場合の HTTPS キー。
	HTTPS 証明書の例：
	`nxapi certificate httpscrt certfile bootflash:cert.crt`
	HTTPS キーの例：
	`nxapi certificate httpskey keyfile bootflash:privkey.key`

nxapi certificatehttpskey keyfile *filename* **password** *passphrase*	暗号化された秘密キーを使用してNX-API証明書をインストールします。 （注） 暗号化された秘密キーを復号するためのパスフレーズは **pass123!** です。 例: `nxapi certificate httpskey keyfile bootflash:encr-cc.pem password pass123!`
nxapi certificate enable	証明書を有効化します。

NX-API 管理コマンド	説明
nxapi certificate sudi	この CLI は、Secure Unique Device Identifier（SUDI）を使用してデバイスを安全に認証する方法を提供します。 nginx の SUDI ベースの認証は、CISCO SUDI 準拠のコントローラによって使用されます。 SUDI は、X.509v3 証明書に含まれる IEEE 802.1AR 準拠のセキュアデバイス識別子で、Cisco デバイスの製品識別子とシリアル番号を維持します。ID は製造時に実装され、公的に識別可能なルート認証局につながれます。 （注） NX-API が SUDI 証明書を使用している場合、ブラウザ、curl などのサードパーティ アプリケーションからはアクセスできません。 （注） 「nxapi certificate sudi」は、構成されるとカスタム証明書/キーを上書きします。カスタム証明書/キーを元に戻す方法はありません。 （注） 「nxapi certificate sudi」と「nxapi certificate trustpoint」と「nxapi certificate enable」は排他的であり、いずれかを構成すると他の構成は削除されます。 （注） NX-API は、SUDI 証明書ベースのクライアント証明書認証をサポートしていません。クライアント証明書認証が必要な場合は、アイデンティティ証明書を使用する必要があります。 （注） NX-API 証明書 CLI は show run の出力に存在しないため、現在の CR/ロールバックでは、「nxapi certificate sudi」オプションで上書きした場合、カスタム証明書に戻せません。
nxapi ssl-ciphers weak	Cisco NX-OS リリース 9.2(1) 以降、弱い暗号はデフォルトで無効になっています。このコマンドを実行すると、デフォルトの動作が変更され、NGINX の弱い暗号が有効になります。このコマンドの **no** 形式を使用すると、デフォルトに戻ります（デフォルトでは弱い暗号は無効です）。

NX-API 管理コマンド	説明
nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2}	Cisco NX-OS リリース 9.2(1) 以降、TLS1.0 はデフォルトで無効になっています。このコマンドを実行すると、文字列で指定された TLS バージョンが有効になります。デフォルトで無効になっている TLS1.0 も、必要であれば指定できます。このコマンドの **no** 形式を使用すると、デフォルトに戻ります（デフォルトでは、TLS1.1 と TLS1.2 のみが有効です）。
nxapi use-vrf *vrf*	デフォルト VRF、管理 VRF、または名前付き VRF を指定します。
ip netns exec management iptables	アクセス制限を実装したうえで、管理 VRF で実行できます。 （注） **feature bash-shell** を有効にしてから、Bash シェルからコマンドを実行する必要があります。Bash シェルの詳細については、Bash の章を参照してください。 *Iptables* は、ポリシーチェーンを使用してトラフィックを許可または阻止するコマンドライン ファイアウォール ユーティリティであり、ほとんどの *Linux* ディストリビューションにプリインストールされています。 （注） iptables を bash シェルで変更した後、リロード後も iptables を永続化する方法の詳細については、リロード間で Iptable を永続化する（206 ページ）を参照してください。
nxapi idle-timeout <timeout>	リリース 9.3(5) 以降では、アイドル状態の NX-API セッションが無効になるまでの時間を設定できます。指定できる時間は 1 ～ 1440 分です。デフォルトの時間は 10 分です。デフォルト値に戻すには、**no nxapi idle-timeout <timeout>** コマンドの no 形式を使用します。

以下は、HTTPS 証明書の正常なアップロードの例です：

```
switch(config)# nxapi certificate httpscrt certfile certificate.crt
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

（注） 証明書を有効にする前に、証明書とキーを構成する必要があります。

以下は、HTTPS キーの正常なアップロードの例です：

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

次に、暗号化された NXAPI サーバー証明書をインストールする方法の例を示します。

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!

switch(config)#nxapi certificate enable
switch(config)#
```

状況によっては、証明書が無効であることを示すエラー メッセージが表示されることがあります。

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
ERROR: Unable to load private key!
Check keyfile or provide pwd if key is encrypted, using 'nxapi certificate httpskey keyfile
 <keyfile> password <passphrase>'.
```

この場合、**nxapi certificatehttpskey keyfile** *filename* **password** *passphrase*を使用して暗号化キー ファイルのパスフレーズを指定する必要があります。

これが問題の原因である場合、証明書を正常にインストールできるはずです。

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!
switch(config)# nxapi certificate enable
switch(config)#
```

NX-API を使用したインタラクティブ コマンドの操作

対話型コマンドの確認プロンプトを無効にし、エラー コード 500 によるタイムアウトを回避するには、対話型コマンドの前に**[端末の dont-ask（terminal dont-ask）]** を追加します。 **を使用。複数の対話型コマンドを区切るには**、それぞれが **。**は単一のブランク文字で囲まれています。

エラー コード 500 でのタイムアウトを回避するために**端末の dont-ask** を使用する対話型コマンドの例をいくつか次に示します**:**

```
terminal dont-ask ; reload module 21
```

```
terminal dont-ask ; system mode maintenance
```

NX-API クライアント認証

NX-API Client Basic Authentication

NX-API clients can authenticate with the NGINX server on the switch through basic authentication over SSL/TLS. This authentication method is supported by configuring a username and password that is saved to a database on the switch. When the NX-API client initiates a connection request, it sends the Hello message which contains the username and password. Assuming the username and password exist in the database, the switch responds by sending the Hello response, which contains a cookie. After this initial handshake is complete, the communication session is open, and the client can begin sending API calls to the switch. For additional information, see セキュリティ, on page 184.

For additional information about basic authentication, including how to configure the username and password on the switch, refer to the Cisco Nexus 9000 Series NX-OS Security Configuration Guide.

NX-API のクライアント証明書認証

NX-OS 9.3(3) 以降、NX-API はクライアントが開始する証明書ベースの認証をサポートしています。証明書ベースの認証では、TLS ハンドシェイク時に信頼できる関係者、つまり認証局（CA）を使用してクライアントとサーバーの両方を相互に認証することで、セキュリティを強化します。証明書ベースの認証では、NX-OS スイッチにアクセスするためのマシン認証だけでなく、人間による認証も可能です。

クライアント証明書認証は、有効な CA（認証局）を介して割り当てられ、NX-API クライアントに保存されている X509 SSL 証明書を使用してサポートされます。証明書は、各 NX-API ユーザー名に割り当てられます。

NX-API クライアントが Hello メッセージを使用して接続要求を開始すると、サーバーの Hello 応答に有効な CA のリストが含められます。クライアントの応答には、NX-API クライアントが使用している特定のユーザー名の証明書など、追加の情報要素が含まれます。

NX-API クライアントは基本認証、証明書認証のいずれかを使用するように構成することができます。または証明書を優先するものの、証明書認証方式が使用できない場合は基本認証にフォールバックするように構成することもできます。

注意事項と制約事項

証明書認証には次の注意事項と制約事項があります。

- NX-API クライアントには、ユーザー名とパスワードを設定する必要があります。
- NX-API クライアントとスイッチは、デフォルトでウェルノウン ポートで HTTP を介して通信します。柔軟性を高めるために、HTTP はウェルノウン ポートでもサポートされます。ただし、追加のポートを設定することもできます。
- クライアント証明書認証の Python スクリプティングがサポートされています。クライアント証明書がパスフレーズで暗号化されている場合、python はパスフレーズの入力を通常どおり要求します。ただし、Python 要求ライブラリの現在の制限により、パスフレーズをスクリプトに渡すことはできません。
- NX-API クライアントとスイッチは、同じトラストポイントを使用する必要があります。
- サポートされるトラストポイントの最大数は、スイッチごとに 26 です。
- 信頼できる CA のリストは、すべての NX-API クライアントとスイッチで同じである必要があります。信頼できる CA の個別のリストはサポートされていません。
- 証明書認証は、NX-API サンドボックスではサポートされていません。
- 次の条件によって、NX-API サンドボックスがスイッチにロードされるかどうかが決まります。
  - NX-API サンドボックスは、**nxapi client certificate authentication optional** または **no nxapi client certificate authentication** が構成されている場合にのみロードされます。

- NX-APIサンドボックスは、接続の確立時に有効なクライアント証明書がブラウザに提示されない限り、**strict** および **two-step** 認証モードをロードしません。

- スイッチにはNGINXサーバーが組み込まれています。複数のトラストポイントが設定されていても、証明書失効リスト（CRL）が1つのトラストポイントだけにインストールされている場合、NGINXの制限によりNX-APIクライアント証明書認証は失敗します。この制限を回避するには、すべてのトラストポイントにCRLを構成します。
- 証明書は期限切れになったり、期限切れになったりする可能性があり、CA（トラストポイント）によって設定されたCRLの有効性に影響を与える可能性があります。スイッチが有効なCRLを使用できるように、設定されているすべてのトラストポイントに必ずCRLをインストールしてください。トラストポイントによって失効した証明書が存在しない場合は、空のCRLを生成、インストールし、更新する必要があります。これはたとえば週に1回行います。

  暗号化CLIを使用してCRLを更新した後、**nxapi client cert authentication** を発行して、新しく更新されたCRLを再適用します。
- NX-APIクライアント証明書認証が有効になっているときにASCIIリロードを使用する場合は、リロードの完了後に **nxapi client certificate authentication** を発行する必要があります。
- 証明書パスは信頼済みCA証明書で終了している必要があります。
- TLS用に提示されるサーバー証明書には、extendedKeyUsageフィールドにサーバー認証目的（OID 1.3.6.1.5.5.7.3.1のid-kp 1）が必要です。
- TLS用に提示されるクライアント証明書には、extendedKeyUsageフィールドにサーバー認証目的（OID 1.3.6.1.5.5.7.3.2のid-kp 1）が必要です。
- この機能はCRL（証明書失効リスト）をサポートします。オンライン証明書ステータスプロトコル（OCSP）はサポートされていません。
- 『NX-OSセキュリティ ガイド』の追加の注意事項と制約事項に従ってください。
  - 証明書と基本認証の両方を使用します。そうすることで、証明書が何らかの理由で侵害された場合でも、正しいユーザーとパスワードが要求されます。
  - 秘密キーは秘密にしておきます。サーバーの公開キーには接続を試みるすべてのユーザーがアクセスできるからです。
  - CRLは中央CAからダウンロードし、最新の状態に保つ必要があります。古いCRLはセキュリティリスクにつながる可能性があります。
  - トラストポイントは最新の状態に保ちます。トラストポイントまたは構成の変更が証明書認証機能に加えられた場合は、更新された情報をリロードするために、この機能を明示的に無効にしてから再度有効にします。

- **nxapi certificate httpscert certfile bootflash:<>**「CLI」でNX-APIに関連付けられるクライアント証明書アイデンティティファイルには、最大8Kというファイルサイズの制限があります。これはDay-1の制限です。

- NX-API 管理コマンドの表 1 において、コマンド nxapi certificate {httpscrt certfile | httpskey keyfile} filename に関連付けられる行で、サポートされる証明書ファイルの最大サイズは 8K 未満です。

NX-API Client Certificate Authentication Prerequisites

Before configuring certificate authentication, make sure the following are present on the switch:

1. Configure the client with a username and password. For information see Configuring User Accounts and RBAC.
2. Configure the CA(s) (trustpoint) and CRL(s) (if any).

If no certificates were revoked by a trustpoint, create a blank CRL for each trustpoint.

For information, see the Cisco Nexus 9000 Series NX-OS Security Configuration Guide.

NX-API クライアント証明書認証の構成

nxapi client certificate authentication コマンドを使用して、NX-API 証明書認証を設定できます。コマンドは、認証方法を制御する制限オプションをサポートします。

この機能は、**no nxapi client certificate authentication** を使用して無効にすることができます。

NX-API クライアントの証明書認証を構成するには、次の手順を実行します。

手順の概要

1. この機能の前提条件が満たされていることを確認します。
2. **config terminal**
3. **nxapi client certificate authentication [{optional | strict | two-step}]**

手順の詳細

	コマンドまたはアクション	目的
Step 1	この機能の前提条件が満たされていることを確認します。	「NX-API Client Certificate Authentication Prerequisites（195 ページ）」を参照してください。

Step 2	**config terminal**
	例：
	`switch-1# config terminal`
	`Enter configuration commands, one per line. End with CNTL/Z.`
	`switch-1(config)#`
		コンフィギュレーション モードに入ります。

Step 3	**nxapi client certificate authentication [{optional | strict | two-step}]** 例：	次のいずれかのモードで証明書認証を有効にします。 • **optional** はクライアント証明書を要求します。

	コマンドまたはアクション	目的
	switch-1# **nxapi client certificate authentication strict** switch-1(config)#	• クライアントが証明書を提供すると、クライアントとサーバーの間で相互検証が行われます。 • クライアントが無効な証明書を提供した場合、認証は失敗し、基本認証へのフォールバックは行われません。 • クライアントが証明書を提供しなかった場合、認証は基本認証（ユーザー名とパスワード）にフォールバックします。 • **strict** は、クライアント証明書の検証を有効にし、認証のために有効なクライアント証明書の提示を要求します。 • **two-step** は、基本認証方式と証明書認証方式の両方が必要な 2 段階検証を有効にします。 （注） スイッチにトラストポイントが構成されていない場合は、この機能を有効にすることはできず、スイッチの画面にエラーメッセージが表示されます。 `No trustpoints configured! Please configure trustpoint using 'crypto ca trustpoint <trustpoint-label>' and associated commands, and then enable this feature.`

証明書認証用の Python スクリプトの例

次の例は、認証用のクライアント証明書を使用した Python スクリプトを示しています。

```
import requests
import json

"""
Modify these please
"""
switchuser='USERID'
switchpassword='PASSWORD'
mgmtip='NXOS MANAGEMENT IP/DOMAIN NAME'

client_cert_file='PATH_TO_CLIENT_CERTIFICATE'
client_key_file='PATH_TO_CLIENT_KEY_FILE'
ca_cert='PATH_TO_CA_CERT_THAT_SIGNED_NXAPI_SERVER_CERT'

url='https://' + mgmtip + '/ins'
myheaders={'content-type':'application/json-rpc'}
payload=[
  {
    "jsonrpc": "2.0",
```

```
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 1
  }
]
response = requests.post(url,data=json.dumps(payload),
headers=myheaders,auth=(switchuser,switchpassword),cert=(client_cert_file_path,client_key_file),verify=ca_cert).json()
```

必要に応じて、スクリプトを変更できます。

- クライアント証明書認証モードによっては、スイッチ パスワードをヌル値に設定することで（`switchpassword=` ）、スイッチ パスワードを省略できます。
  - **optional** および **strict** モードの場合、`switchpassword=` は空白のままにできます。この場合、NX-APIはユーザー名とクライアント証明書のみに基づいてクライアントを認証します。
  - **two-step** モードの場合、パスワードが必要なため、`switchpassword=` の値を指定する必要があります。
- POST コマンドで `verify=False` を設定することで、NX-API サーバの証明書が有効であることの確認をバイパスできます。

cURL 証明書要求の例

次に、NX-API クライアント認証用の正しく構造化された cURL 証明書要求の例を示します。

```
/usr/bin/curl --user admin: --tlsv1.2 --cacert ./ca.pem --cert ./user.crt:pass123! --key
./user.key -v -X POST -H "Accept: application/json" -H "Content-type: application/json"
--data '{"ins_api":{"version": "1.0", "type": "cli_show", "chunk": "0", "sid": "1", "input":
 "show clock","output_format": "json"}}' https://<device-management-ip>:443/ins
```

構文要素

次の表は、この要求で使用されるパラメータを示しています。

パラメータ	説明
--user	ユーザーがログインするユーザー名を取得します。これは、user.crt の共通名と同じである必要があります。 ユーザーのパスワードを指定するには、コロンの後にパスワードを指定します。例：**--user username:password**

パラメータ	説明
--cacert	NX-API サーバー証明書に署名した CA へのパスを使用します。 サーバー証明書を検証する必要がない場合は、**-k**（insecure）オプションを使用して cURL を指定します。例：**`/usr/bin/curl -k`**
--cert	クライアント証明書へのパスを指定します。 クライアント証明書が暗号化されている場合は、コロンの後にパスワードを指定します。例：**`--cert user.crt:pass123!`**
--key	クライアント証明書の秘密キーへのパスを指定します。

証明書認証の検証

正しく構成されている場合、証明書認証が行われ、NX-API クライアントはスイッチにアクセスできます。

NX-API クライアントがスイッチにアクセスできない場合は、次のガイドラインに従ってトラブルシューティングを行うことができます。

手順の概要

1. ユーザーまたはクッキーのエラーを確認します。
2. 証明書に誤りがないか確認してください。
3. エラーが発生した場合は、 **no nxapi client certificate authentication** 、それから **nxapi client certificate authentication** を発行して、トラストポイント、CA、CRL、または NX-OS 証明書機能に対する変更をリロードするように機能をフラップします。

手順の詳細

	コマンドまたはアクション	目的
Step 1	ユーザーまたはクッキーのエラーを確認します。	次のいずれかのエラーが発生していないか、確認します： • 認証ヘッダーにユーザー名が指定されておらず、有効なクッキーが指定されていない • 認証ヘッダーで指定されたユーザーが正しくない • 無効なクッキーが提供された

	コマンドまたはアクション	目的

		• 認証ヘッダーのユーザー名とクライアント証明書の CN フィールドのユーザー名が一致しない
		
		使用されている NX-API 方式に応じて、特定のエラーが表示されます：
		• JSON/XML の場合、401 認証エラー：ユーザーが見つからないエラーが発生します。次に例を示します。
		`{{{`
		`"code": "400",`
		`"msg": "Authentication failure - user not found."`
		`}}}`
		• JSON RPC 2.0 の場合、-32004 無効なユーザー名またはパスワードエラーが発生します。次に例を示します。
		`{{`
		`"code": -32004,`
		`"message": "Invalid username or password"`
		`}}`

Step 2	証明書に誤りがないか確認してください。	次の内容を示す HTTPs 400 エラーを探します。
		• 無効または失効したクライアント証明書が提供された場合。
		• スイッチに構成されている CRL の有効期限が切れている場合。
		次に例を示します。
		`<html>`
		`<head><title>400 The SSL certificate error</title></head>`
		`<body bgcolor="white">`
		`<center><h1>400 Bad Request</h1></center>`
		`<center>The SSL certificate error</center>`
		`<hr><center>nginx/1.7.10</center>`
		`</body>`
		`</html>`

Step 3	エラーが発生した場合は、 **no nxapi client certificate authentication** 、それから **nxapi client certificate authentication** を発行して、トラストポイント、CA、CRL、または NX-OS 証明書機能に対する変更をリロードするように機能をフラップします。	証明書認証を無効にしてから、再度有効にします。

NX-API リクエスト要素

NX-API リクエスト要素は、XML フォーマットまたは JSON フォーマットでデバイスに送信されます。リクエストの HTTP ヘッダーは、リクエストのコンテンツ タイプを識別する必要があります。

次の表にリストされている NX-API 要素を使用して、CLI コマンドを指定します。

（注） ユーザーには、「configure terminal」コマンドを実行する権限が必要です。JSON-RPC が入力リクエスト フォーマットの場合、「configure terminal」コマンドは、常に、ペイロード内のコマンドが実行される前に実行されます。

表 *6 : XML* または *JSON* フォーマットの *NX-API* リクエスト要素

NX-API リクエスト要素	説明
version	NX-API バージョンを指定します。

NX-API リクエスト要素	説明
type	実行するコマンドのタイプを指定します。 次のタイプのコマンドがサポートされています。 • **cli_show** 構造化された出力が必要なCLI **show** コマンド。コマンドが XML 出力をサポートしていない場合は、エラーメッセージが返されます。 • **cli_show_ascii** ASCII出力が必要なCLI **show** コマンド。これは、ASCII出力を解析する既存のスクリプトと一致します。ユーザーは、最小限の変更で既存のスクリプトを使用できます。 • **cli_conf** CLI 構成コマンド • **bash** Bashコマンド。ほとんどの非対話型Bashコマンドは、NX-API でサポートされています。 （注） • 各コマンドは、現在のユーザーの権限でのみ実行可能です。 • メッセージタイプがASCIIの場合、出力でパイプ操作がサポートされます。出力がXML 形式の場合、パイプ操作はサポートされていません。 • 最大10の連続する **show** コマンドがサポートされています。**show** コマンドの数が 10 を超える場合、11番目以降のコマンドは無視されます。 • 対話型コマンドはサポートされていません。

<table>
<tr><th>NX-API リクエスト要素</th><th>説明</th></tr>
<tr><td>チャンク</td><td>一部の show コマンドは、大量の出力を返す場合があります。コマンド全体が完了する前に NX-API クライアントが出力の処理を開始するために、NX-API は show コマンドの出力チャンクをサポートしています。

次の設定を有効または無効にできます。

（注）
<table><tr><td>0</td><td>チャンク出力しません。</td></tr><tr><td>1</td><td>チャンク出力。</td></tr></table>
（注）
• チャンクをサポートするのは show コマンドだけです。一連の show コマンドが入力されると、最初のコマンドだけがチャンクされて返されます。
• 出力メッセージ形式のオプションは、XML または JSON です。
• XML 出力メッセージ形式の場合< または> などの特殊文字は、有効な XML メッセージを形成するために変換されます（< は < に変換されます > は &gt に変換されます）。
XML SAX を使用して、チャンクされた出力を解析できます。
• 出力メッセージ形式が JSON の場合、チャンクが連結されて有効な JSON オブジェクトが作成されます。

（注）
チャンクが有効になっている場合、現在サポートされている最大メッセージサイズは、チャンク出力の 200MB です。</td></tr>
</table>

NX-API リクエスト要素	説明
sid	セッションID要素は、応答メッセージがチャンクされている場合にのみ有効です。メッセージの次のチャンクを取得するには、前の応答メッセージの *sid* と一致する *sid* を指定する必要があります。 NX-OS リリース 9.3(1) では、*sid* オプション `clear` が導入されています。*sid* を `clear` に設定して新しいチャンク リクエストが開始されると、現在のチャンク リクエストはすべて破棄または破棄されます。 応答コード `429` を受け取った場合：同時チャンク リクエストの最大数は `2` です。*sid* `clear` を使用して、現在のチャンク リクエストを破棄します。*sid* `clear` を使用した後、後続の応答コードは、残りのリクエストに対して通常どおり動作します。
input	入力は1つのコマンドまたは複数のコマンドです。ただし、異なるメッセージ タイプに属するコマンドを混在させてはなりません。たとえば、**show** コマンドは cli_show メッセージ タイプであり、cli_conf モードではサポートされません。 （注） **bash** を除き、複数のコマンドは「;」で区切ります。（; は、単一のブランク文字で囲む必要があります。） エラーコード 500 でタイムアウトしないように、コマンドの前に `terminal dont-ask` を付加します。次に例を示します。 `terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport` **bash** の場合、複数のコマンドは「;」で区切ります。（; は単一のブランク文字で囲まれて**いません**。） 以下は、複数のコマンドの例です。 （注） cli_show: `show version ; show interface brief ; show vlan` cli_conf: `interface Eth4/1 ; no shut ; switchport` bash: `cd /bootflash;mkdir new_dir`

NX-API リクエスト要素	説明
output_format	使用可能な出力メッセージ形式は次のとおりです。 （注） xml：XML 形式を指定します。 json：JSON 形式で出力を指定します。 （注） Cisco NX-OS CLI は XML 出力をサポートしています。つまり、JSON 出力は XML から変換されます。変換はスイッチで処理されます。 計算のオーバーヘッドを管理するために、JSON 出力は出力の量によって決定されます。出力が 1 MB を超える場合、出力は XML 形式で返されます。出力がチャンクされている場合、XML 出力のみがサポートされます。 HTTP/HTTPS ヘッダーの content-type ヘッダーは、応答形式（XML または JSON）のタイプを示します。

NX-API 応答要素

CLI コマンドに応答する NX-API 要素を次の表に示します。

表 *7: NX-API* 応答要素

NX-API 応答要素	説明
version	NX-API バージョン。
type	実行するコマンドのタイプ。
sid	応答のセッション識別子。この要素は、応答メッセージがチャンクされている場合にのみ有効です。
outputs	すべてのコマンド出力を囲むタグ。 複数のコマンドが cli_show または cli_show_ascii にある場合、各コマンド出力は単一の出力タグで囲まれます。 メッセージ タイプが cli_conf または bash の場合、cli_conf および bash コマンドにはコンテキストが必要なため、すべてのコマンドに単一の出力タグがあります。

NX-API 応答要素	説明
出力	単一のコマンド出力の出力を囲むタグ。 cli_conf と bash メッセージ タイプの場合、この要素にはすべてのコマンドの出力が含まれます。
input	リクエストで指定された1つのコマンドを囲むタグ。この要素は、要求入力要素を適切な応答出力要素に関連付けるのに役立ちます。
本文	コマンド応答の本文。
コード	コマンドの実行から返された原因コード。 NX-API は、ハイパーテキスト転送プロトコル（HTTP）ステータス コード レジストリ（http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml）で説明されている標準規格の HTTP原因コードを使用します。
msg	返された原因コードに関連付けられたエラー メッセージ。

NX-API へのアクセスの制限

デバイスへの HTTP および HTTPS アクセスを制限するには、ACL と iptable の 2 つの方法があります。使用する方法は、`nxapi use-vrf<vrf-name>` CLI コマンドを使用して、NX-API 通信の VRF を構成していたかどうかに応じて決まります。

特定の VRF を使用するように NXAPI を構成していない場合にのみ、ACL を使用してデバイスへの HTTP または HTTPS アクセスを制限します。ACL の構成の詳細については、使用しているスイッチ ファミリの *Cisco Nexus* シリーズ *NX-OS* セキュリティ構成ガイドを参照してください。

ただし、NX-API 通信用に VRF を設定した場合、ACL は HTTP または HTTPS アクセスを制限しません。代わりに、iptable のルールを作成します。ルールの作成の詳細については、iptable の更新 （205 ページ）を参照してください。

iptable の更新

iptable を使用すると、VRF が NX-API 通信用に構成されている場合に、デバイスへの HTTP または HTTPS アクセスを制限できます。このセクションでは、既存の iptable に HTTP および HTTPS アクセスをブロックするルールを追加、確認、削除する方法を示します。

Step 1 HTTP アクセスをブロックするルールを作成するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp  --dport 80 -j DROP
```

Step 2 HTTPS アクセスをブロックするルールを作成するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp  --dport 443 -j DROP
```

Step 3 適用されたルールを確認するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:http
DROP       tcp  --  anywhere             anywhere             tcp dpt:https

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Step 4 10.155.0.0/24 サブネットのポート 80 へのすべてのトラフィックをブロックするルールを作成して確認するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -s 10.155.0.0/24 -p tcp --dport 80 -j DROP
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  10.155.0.0/24        anywhere             tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Step 5 以前に適用したルールを削除して確認するには、次の手順を実行します。

この例では、最初のルールを INPUT から削除します。

```
bash-4.3# ip netns exec management iptables -D INPUT 1
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

次のタスク

iptables のルールを bash シェルで変更した場合、リロード後は保持されません。ルールを永続的にするには、リロード間で Iptable を永続化する （206 ページ）を参照してください。

リロード間で Iptable を永続化する

iptable のルールを bash シェルで変更した場合、リロード後は保持されません。このセクションでは、リロード後も変更された iptable を永続化する方法について説明します。

始める前に

iptable を変更したとします。

Step 1 iptables_init.log という名前のファイルを /etc ディレクトリに作成します。

```
bash-4.3# touch /etc/iptables_init.log; chmod 777 /etc/iptables_init.log
```

Step 2 iptable の変更を保存する /etc/sys/iptables ファイルを作成します。

```
bash-4.3# ip netns exec management iptables-save > /etc/sysconfig/iptables
```

Step 3 次の一連のコマンドを使用して、/etc/init.dディレクトリに「iptables_init」という起動スクリプトを作成します。

```
#!/bin/sh

### BEGIN INIT INFO

# Provides:          iptables_init

# Required-Start:

# Required-Stop:

# Default-Start:     2 3 4 5

# Default-Stop:

# Short-Description: init for iptables

# Description:       sets config for iptables

#                    during boot time

### END INIT INFO


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
  start)
    start_script
   ;;
  stop)
    ;;
  restart)
    sleep 1
    $0 start
    ;;
  *)
    echo "Usage: $0 {start|stop|status|restart}"
    exit 1
esac
exit 0
```

Step 4 起動スクリプトに適切な権限を設定します。

```
bash-4.3# chmod 777 /etc/init.d/iptables_int
```

Step 5 chkconfig ユーティリティを使用して、「iptables_int」起動スクリプトを「オン」に設定します。

```
bash-4.3# chkconfig iptables_init on
```

「iptables_init」起動スクリプトは、リロードを実行するたびに実行されます。これで iptable ルールを永続的にすることができました。

NX-API 応答コードの表

（注） 標準の HTTP エラー コードは、ハイパーテキスト転送プロトコル (HTTP) ステータス コード レジストリ (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml) にあります。

表 *8 : NX-API* 応答コード

[NX-API 応答（NX-API Response）]	コード	メッセージ
成功	200	成功。
CUST_OUTPUT_PIPED	204	要求により、出力は別の場所にパイプされます。
BASH_CMD_ERR	400	Bash コマンド エラー
CHUNK_ALLOW_ONE_CMD_ERR	400	チャンクは、1 つのコマンドだけを受け入れます。
CLI_CLIENT_ERR	400	CLI の実行エラー
CLI_CMD_ERR	400	CLI コマンド エラーの入力。
EOC_NOT_ALLOWED_ERR	400	eoc 値は、リクエストのセッション ID として許可されていません。
IN_MSG_ERR	400	着信メッセージが無効です。
INVALID_REMOTE_IP_ERR	400	要求のリモートIPを取得できません。
MSG_VER_MISMATCH	400	メッセージ バージョンの不一致
NO_INPUT_CMD_ERR	400	入力コマンドがありません。

SID_NOT_ALLOWED_ERR	400	セッション ID として入力された文字が無効です。
PERM_DENY_ERR	401	権限が拒否されました。
CONF_NOT_ALLOW_SHOW_ERR	405	構成モードは **[表示（show）]** を許可しません。
SHOW_NOT_ALLOW_CONF_ERR	405	表示モードでは構成できません。
EXCEED_MAX_SHOW_ERR	413	連続する show コマンドの最大数を超えました。最大値は 10 です。
MSG_SIZE_LARGE_ERR	413	応答サイズが大きすぎます。
RESP_SIZE_LARGE_ERR	413	応答サイズが最大メッセージ サイズを超えたため、処理を停止しました。最大サイズは 200 MB です。
EXCEED_MAX_INFLIGHT_CHUNK_REQ_ERR	429	同時チャンク リクエストの最大数は超えています。最大は 2 です。
MAX_SESSIONS_ERR	429	最大セッション数に到達しました。新しいユーザー/クライアントの場合は、しばらくしてからもう一度お試しください。
OBJ_NOT_EXIST	432	要求したオブジェクトが存在しません。
BACKEND_ERR	500	バックエンド処理エラー。
CREATE_CHECKPOINT_ERR	500	チェックポイントの作成をするエラー。
DELETE_CHECKPOINT_ERR	500	チェックポイントの削除中にエラーが発生しました。
FILE_OPER_ERR	500	システム内部ファイル操作エラー。
LIBXML_NS_ERR	500	システムの内部 LIBXML NS エラー。これは要求フォーマットのエラーです。
LIBXML_PARSE_ERR	500	システムの内部 LIBXML 解析エラー。これは要求フォーマットのエラーです。
LIBXML_PATH_CTX_ERR	500	システムの内部 LIBXML パス コンテキスト エラー。これは要求フォーマットのエラーです。
MEM_ALLOC_ERR	500	システムの内部メモリ割り当てエラー。

ROLLBACK_ERR	500	ロールバックの実行中にエラーが発生しました。
SERVER_BUSY_ERR	500	サーバーがビジー状態のため、リクエストは拒否されました。
USER_NOT_FOUND_ERR	500	入力またはキャッシュからユーザーが見つかりません。
VOLATILE_FULL	500	揮発性メモリは一杯です。メモリ スペースを解放して、再試行してください。
XML_TO_JSON_CONVERT_ERR	500	XML から JSON への変換エラー。
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash コマンドはサポートされていません。
CHUNK_ALLOW_XML_ONLY_ERR	501	チャンクは XML 出力のみを許可します。
CHUNK_ONLY_ALLOWED_IN_SHOW_ERR	501	応答のチャンクは、`show` コマンドでのみ許可されます。
CHUNK_TIMEOUT	501	チャンク応答の生成中にタイムアウトしました。
CLI_CMD_NOT_SUPPORTED_ERR	501	CLI コマンドはサポートされていません。
JSON_NOT_SUPPORTED_ERR	501	大量の出力の可能性があるため、JSONはサポートされていません。
MALFORMED_XML	501	不正な XML 出力。
MSG_TYPE_UNSUPPORTED_ERR	501	メッセージ タイプはサポートされていません
OUTPUT_REDIRECT_NOT_SUPPORTED_ERR	501	出力リダイレクトはサポートされていません。
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	このコマンドへのパイプXMLは入力では許可されていません。
PIPE_NOT_ALLOWED_IN_INPUT	501	この入力タイプにはパイプを使用できません。
RESP_BIG_USE_CHUNK_ERR	501	応答が許容最大値を超えています。最大は 10 MB です。チャンクを有効にして XML または JSON 出力を使用します。
STRUCT_NOT_SUPPORTED_ERR	501	構造化出力はサポートされていません。
ERR_UNDEFINED	600	不明なエラー。

JSON および XML 構造化出力

NX-OS は、次の構造化された出力フォーマットで、さまざまな **show** コマンドの標準規格出力のリダイレクトをサポートしています。

- XML
- JSON。JSON 出力の上限は 60 MB です。
- JSON フォーマット出力の標準規格ブロックを読みやすくした JSON Pretty もあります。JSON 出力の上限は 60 MB です。
- NX-OS リリース 9.3（1）で導入された JSON Native と JSON Pretty Native は、追加のコマンド解釈レイヤーをバイパスすることにより、JSON 出力をより高速かつ効率的に表示します。JSON Native および JSON Pretty Native は、出力のデータ型を保持します。出力用の文字列に変換する代わりに、整数を整数として表示します。

NX-OS CLI で、標準の NX-OS 出力を JSONまたは XML インタープリターに「パイプ接続」すると、これらのフォーマットへの変換が行われます。たとえば、**show ip access** コマンドを発行する際、論理パイプ（|）を続けて、その後に出力形式を指定できます。こうすると、NX-OS コマンドの出力が適切に構造化され、その形式でエンコードされます。この機能により、プログラムによるデータの解析が可能になり、ソフトウェア ストリーミング テレメトリを介したスイッチからのストリーミング データがサポートされます。Cisco NX-OS のほとんどのコマンドは、JSON、JSON Pretty、JSON ネイティブ、JSON ネイティブ Pretty、および XML 出力をサポートしています。整合性チェッカー コマンドなど、一部のコマンドは、すべての形式をサポートしてはいません。整合性チェッカー コマンドは XML をサポートしていますが、JSON のバリアントはどれもサポートしていません。

（注） 検証エラーを回避するには、ファイル リダイレクトを使用して JSON 出力をファイルにリダイレクトし、そのファイル出力を使用します。

例:

```
Switch#show version | json > json_output ; run bash cat /bootflash/json_output
```

この機能の選択された例を以下に表示します。

JSON の概要（JavaScript オブジェクト表記）

JSON は、判読可能なデータのために設計された軽量テキスト ベースのオープン スタンダードで、XML の代替になります。JSON はもともと JavaScript から設計されましたが、言語に依存しないデータ形式です。コマンド出力では、JSON および JSON プリティ フォーマット、および JSON ネイティブおよび JSON プリティ ネイティブがサポートされています。

ほぼすべての最新のプログラミング言語で何らかの方法でサポートされている 2 つの主要なデータ構造は次のとおりです。

- 順序付きリスト :: 配列

- 順序付けられていないリスト（名前/値のペア）:: オブジェクト

show コマンドの JSON または XML 出力には、NX-API サンドボックスからもアクセスできます。

CLI の実行

```
switch-1-vxlan-1# show cdp neighbors  | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "
83886080", "device_id": "SW-SWITCH-1", "intf_id": "mgmt0", "ttl": "148"
, "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco AA-C0000
S-29-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device
_id": "SWITCH-1-VXLAN-1(FOC1234A01B)", "intf_id": "Ethernet1/1", "ttl": "166
", "capability": ["router", "switch", "IGMP_cnd_filtering", "Supports-STP-Disput
e"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
BLR-VXLAN-NPT-CR-179#
```

XML および JSON 出力の例

このセクションでは、XML および JSON 出力として表示される NX-OS コマンドの例について説明します。

次の例は、ハードウェア テーブルのユニキャストおよびマルチキャスト ルーティング エントリを JSON 形式で表示する方法を示しています。

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096",
 "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "u
sed_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_tot
al": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "
used_mcast_oifl": "2", "used_host_in_host_total": "13", "used_host4_in_host": "1
2", "used_host6_in_host": "1", "max_ecmp_table_limit": "64", "used_ecmp_table":
"0", "mfib_fd_status": "Disabled", "mfib_fd_maxroute": "0", "mfib_fd_count": "0"
}
switch(config)#
```

次に、ハードウェアテーブルのユニキャストおよびマルチキャストルーティングエントリをXML形式で表示する例を示します。

```
switch(config)# show hardware profile status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:fib">
 <nf:data>
  <show>
   <hardware>
    <profile>
     <status>
      <__XML__OPT_Cmd_dynamic_tcam_status>
       <__XML__OPT_Cmd_dynamic_tcam_status___readonly__>
        <__readonly__>
         <total_lpm>8191</total_lpm>
         <total_host>8192</total_host>
         <total_lpm>1024</total_lpm>
         <max_host4_limit>4096</max_host4_limit>
         <max_host6_limit>2048</max_host6_limit>
         <max_mcast_limit>2048</max_mcast_limit>
         <used_lpm_total>9</used_lpm_total>
         <used_v4_lpm>6</used_v4_lpm>
         <used_v6_lpm>3</used_v6_lpm>
```

```
          <used_v6_lpm_128>1</used_v6_lpm_128>
          <used_host_lpm_total>0</used_host_lpm_total>
          <used_host_v4_lpm>0</used_host_v4_lpm>
          <used_host_v6_lpm>0</used_host_v6_lpm>
          <used_mcast>0</used_mcast>
          <used_mcast_oifl>2</used_mcast_oifl>
          <used_host_in_host_total>13</used_host_in_host_total>
          <used_host4_in_host>12</used_host4_in_host>
          <used_host6_in_host>1</used_host6_in_host>
          <max_ecmp_table_limit>64</max_ecmp_table_limit>
          <used_ecmp_table>0</used_ecmp_table>
          <mfib_fd_status>Disabled</mfib_fd_status>
          <mfib_fd_maxroute>0</mfib_fd_maxroute>
          <mfib_fd_count>0</mfib_fd_count>
         </__readonly__>
        </__XML__OPT_Cmd_dynamic_tcam_status___readonly__>
       </__XML__OPT_Cmd_dynamic_tcam_status>
      </status>
     </profile>
    </hardware>
   </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

この例では、JSON 形式でスイッチ上に LLDP タイマーを表示する方法を示します。

```
switch(config)# show lldp timers | json
{"ttl": "120", "reinit": "2", "tx_interval": "30", "tx_delay": "2", "hold_mplier
": "4", "notification_interval": "5"}
switch(config)#
```

この例では、XML 形式でスイッチ上に LLDP タイマーを表示する方法を示します。

```
switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:lldp">
 <nf:data>
  <show>
   <lldp>
    <timers>
     <__XML__OPT_Cmd_lldp_show_timers___readonly__>
      <__readonly__>
       <ttl>120</ttl>
       <reinit>2</reinit>
       <tx_interval>30</tx_interval>
       <tx_delay>2</tx_delay>
       <hold_mplier>4</hold_mplier>
       <notification_interval>5</notification_interval>
      </__readonly__>
     </__XML__OPT_Cmd_lldp_show_timers___readonly__>
    </timers>
   </lldp>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

この例は、スイッチの冗長性情報を JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show system redundancy status | json-pretty native
{
        "rdn_mode_admin":       "HA",
        "rdn_mode_oper":        "None",
        "this_sup":     "(sup-1)",
        "this_sup_rdn_state":   "Active, SC not present",
        "this_sup_sup_state":   "Active",
        "this_sup_internal_state":      "Active with no standby",
        "other_sup":    "(sup-1)",
        "other_sup_rdn_state":  "Not present"
}
switch-1#
```

次の例は、スイッチの OSPF ルーティング パラメータを JSON ネイティブ形式で表示する方法を示しています。

```
switch-1# show ip ospf | json native
{"TABLE_ctx":{"ROW_ctx":[{"ptag":"Blah","instance_number":4,"cname":"default","
rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr
_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_t
os0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false"
,"admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S"
,"spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_
time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"m
ax_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cn
t":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_
nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa
":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"100","ins
tance_number":3,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":
"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive"
,"gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true"
,"is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_
time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0
S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_
aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"
asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"
area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal"
:0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_
rt_int":"false"},{"ptag":"111","instance_number":1,"cname":"default","rid":"0.0
.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_pe
riod":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only"
:"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_d
ist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max
_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT
5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric
_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"aso
paque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"
act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_d
iscard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_num
ber":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","g
r_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last
_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr"
:"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT
0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_h
old_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pac
e":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa
_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_norm
al":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_a
rea_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"
false"}]}}
switch-1#
```

次の例は、OSPF ルーティング パラメータを JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show ip ospf | json-pretty native
{
   "TABLE_ctx": {
          "ROW_ctx": [{
                        "ptag": "Blah",
                        "instance_number":      4,
                        "cname":        "default",
                        "rid":  "0.0.0.0",
                        "stateful_ha":  "true",
                        "gr_ha":        "true",
                        "gr_planned_only":      "true",
                        "gr_grace_period":      "PT60S",
                        "gr_state":     "inactive",
                        "gr_last_status":       "None",
                        "support_tos0_only":    "true",
                        "support_opaque_lsa":   "true",
                        "is_abr":       "false",
                        "is_asbr":      "false",
                        "admin_dist":   110,
                        "ref_bw":       40000,
                        "spf_start_time":       "PT0S",
                        "spf_hold_time":        "PT1S",
                        "spf_max_time": "PT5S",
                        "lsa_start_time":       "PT0S",
                        "lsa_hold_time":        "PT5S",
                        "lsa_max_time": "PT5S",
                        "min_lsa_arr_time":     "PT1S",
                        "lsa_aging_pace":       10,
                        "spf_max_paths":        8,
                        "max_metric_adver":     "false",
                        "asext_lsa_cnt":        0,
                        "asext_lsa_crc":        "0",
                        "asopaque_lsa_cnt":     0,
                        "asopaque_lsa_crc":     "0",
                        "area_total":   0,
                        "area_normal":  0,
                        "area_stub":    0,
                        "area_nssa":    0,
                        "act_area_total":       0,
                        "act_area_normal":      0,
                        "act_area_stub":        0,
                        "act_area_nssa":        0,
                        "no_discard_rt_ext":    "false",
                        "no_discard_rt_int":    "false"
                      }, {
                        "ptag": "100",
                        "instance_number":      3,
                        "cname":        "default",
                        "rid":  "0.0.0.0",
                        "stateful_ha":  "true",
                        "gr_ha":        "true",
                        "gr_planned_only":      "true",
                        "gr_grace_period":      "PT60S",
                        "gr_state":     "inactive",

                        ... content deleted for brevity ...

                        "max_metric_adver":     "false",
                        "asext_lsa_cnt":        0,
                        "asext_lsa_crc":        "0",
                        "asopaque_lsa_cnt":     0,
                        "asopaque_lsa_crc":     "0",
                        "area_total":   0,
                        "area_normal":  0,
```

```
                        "area_stub":    0,
                        "area_nssa":    0,
                        "act_area_total":       0,
                        "act_area_normal":      0,
                        "act_area_stub":        0,
                        "act_area_nssa":        0,
                        "no_discard_rt_ext":    "false",
                        "no_discard_rt_int":    "false"
                }]
        }
}
switch-1#
```

次の例は、JSON ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json native
{"TABLE_vrf":{"ROW_vrf":[{"vrf-name-out":"default","TABLE_addrf":{"ROW_addrf":[{"addrf":"ipv4","TABLE_summary":{"ROW_summary":[{"routes":3,"p
aths":3,"TABLE_unicast":{"ROW_unicast":[{"clientnameuni":"broadcast","best-paths":3}]},"TABLE_route_count":{"ROW_route_count":[{"mask_len":8,
"count":1},{"mask_len":32,"count":2}]}}]}}]}}]}}
switch-1(config)#
```

JSON ネイティブ（および JSON プリティ ネイティブ）では、整数が真の整数として表されることに注意してください。たとえば、「mask len:」は実際の値 32 として表示されます。

次の例は、JSON プリティ ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json-pretty native
{
  "TABLE_vrf": {
    "ROW_vrf": [{
        "vrf-name-out": "default",
        "TABLE_addrf": {
               "ROW_addrf": [{
                    "addrf": "ipv4",
                    "TABLE_summary": {
                           "ROW_summary": [{
                                "routes": 3,
                                "paths": 3,
                              "TABLE_unicast": {
                                    "ROW_unicast": [{
                                          "clientnameuni": "broadcast",
                                          "best-paths": 3
                                                    }]
                                                },
                                                 "TABLE_route_count": {
                                                       "ROW_route_count":[{
                                                             "mask_len": 8,
                                                             "count": 1
                                                                   }, {
                                                             "mask_len": 32,
                                                             "count": 2
                                                       }]
                                                }
                                          }]
                                    }
                           }]
                    }
               }]
        }
}
switch-1(config)#
```

サンプル NX-API スクリプト

ユーザーはNX-APIでスクリプトを使用する方法を示すサンプル スクリプトにアクセスできます。サンプル スクリプトにアクセスするには、次のリンクをクリックして、必要なソフトウェア リリースに対応するディレクトリを選択します：Cisco Nexus 9000 NX-OS NX-API

第 19 章

NX-API REST

この章は次のトピックで構成されています。

NX-API REST について

NX-API REST

Cisco Nexus スイッチでは、構成はコマンドライン インターフェイス（CLI）を使用して実施します。CLI は、当該スイッチ上でしか実行できません。NX-API REST は、HTTP/HTTPS API を提供することにより、Cisco Nexus 構成のアクセシビリティを向上させます。

- 特定の CLI コマンドをスイッチの外部から実行可能です。
- 多数の CLI コマンドを発行する必要がある構成を、比較的少数の HTTP/HTTPS 操作の構成アクションを組み合わせることで実現します。

NX-API REST は、**show** コマンド、基本および詳細スイッチ構成と Linux Bash をサポートします。

NX-API REST は HTTP/HTTPS をトランスポートとして使用します。CLI は、HTTP / HTTPS POST 本文にエンコードされます。NX-API REST バックエンドは Nginx HTTP サーバーを使用します。Nginx プロセスとそのすべての子プロセスは、CPU とメモリの使用量の上限が定められている Linux cgroup の保護下に置かれます。NX-API プロセスは、cgroup ext_ser_nginx の一部であり、2,147,483,648 バイトのメモリに制限されています。Nginx のメモリ使用量が cgroup の制限を超えると、Nginx プロセスは再起動されて、NX-API 構成（VRF、ポート、証明書構成）が復元されます。

Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、https://developer.cisco.com/docs/nx-os-n3k-n9k-api-ref/ を参照してください。

REST による DME 構成置換

REST Put による DME フル構成置換について

Cisco NX-OS リリース 9.3(1) 以降、Cisco NX-OS は REST PUT 操作によるモデルベースの完全な設定置換をサポートします。設定を置き換えるこの方法では、Cisco DME モデルを使用します。

DME フル構成置換機能を使用すると、REST プログラム インターフェイスを使用してスイッチの実行構成を置き換えられます。DME フル構成置換機能には、置換を PUT 操作によって行えるという追加のメリットがあります。設定ツリーのすべての部分（システムレベル、サブツリー、およびリーフ）が、DME フル構成置換をサポートします。

- スイッチ構成の無停止交換のサポート
- 自動化のサポート
- 他の機能やその構成に影響を与えることなく、機能を選択的に変更する機能を提供します。
- 最終的な構成結果を指定できるようにすることで、構成変更を簡素化し、人的エラーを排除します。スイッチは差分を計算し、構成ツリーの影響を受ける部分にプッシュします。

（注） プログラム的なインターフェイスを使用して実行することはできませんが、**config replace config-file-name** Cisco NX-OS CLI コマンドにより、フル構成置換を実行することもできます。

Guidelines and Limitations

The following are the guidelines and limitations for the DME full config replace feature:

- For information about supported platforms, see the Nexus Switch Platform Matrix.
- It is important for you to know the tree and know where you are applying the config replace. If you are using the Sandbox for the config replace operation, the Sandbox defaults to the subtree, so you might need to change the URI to target the correct node in the config tree.
- If you use the NX-OS Sandbox to Convert (for Replace), you must use the POST operation because of the presence of the `status: 'replaced'` attribute in the request. If you are using any other conversion option, you can use the PUT operation.
- If you use the REST PUT option for this feature on a subtree node, config replace operation is applied to the entire subtree. The target subtree node is correctly changed with the config replace data in the PUT, but be aware that leaf nodes of the subtree node are also affected by being set to default values.

  If you do not want the leaf nodes to be affected, do not use a PUT operation. Instead, you can use a POST operation with the `status:'replaced'` attribute.

  If you are applying the config replace to a leaf node, the PUT operation operates predictably.

REST POST によるプロパティレベルの構成置換

シスコの DME モデルは、REST POST 操作による CLI ベースの機能のプロパティレベルの構成置換をサポートしています。要求ペイロードを生成し、REST POST 操作を介してスイッチに送信することにより、NX-OS サンドボックスを介して機能のプロパティの構成を置き換えることができます。NX-OS サンドボックスの詳細については、NX-API 開発者サンドボックスを参照してください。

Step 1 HTTPS を介し、NX-OS サンドボックスを介してスイッチに接続し、ログイン情報を入力します。

Step 2 作業エリアで、変更する機能の CLI を入力します。

Step 3 作業エリアの下のフィールドで、構成する機能に対するツリー内の MO への URI を設定します。この MO レベルは Put 要求の送信先です。

Step 4 [方法（Method）] で、`NX-API (DME)` を選択します。

Step 5 [入力タイプ（Input Type）] で、[CLI] を選択します。

Step 6 [変換（Convert）] ドロップダウンリストから `Convert (for replace)` を選択して、[要求（Request）] ペインでペイロードを生成します。

Step 7 スイッチへの **POST** 操作を使用する要求をクリックします。

（注） プロパティレベルの構成置換は、構成がデフォルト構成の場合に失敗する可能性があります。これは、置換操作はすべての子 MOを削除し、すべてのプロパティをデフォルトにリセットしようと試みるからです。

REST PUT による機能レベルの構成置換

Cisco DME は、REST PUT 操作による機能レベルの構成置換をサポートしています。モデルの機能レベルで PUT を送信することで、特定の機能の構成を置き換えることができます。

次の手順を使用します。

Step 1 クライアントから、機能のモデル オブジェクト（MO）で REST PUT 操作を発行します。

a) Put は、最上位システムレベルから機能の MO への URL を指定する必要があります。

たとえば、BGPの場合、`/api/mo/sys/bgp.json` のようになります。

ペイロードは有効な構成である必要があり、機能の DN で GET を発行することで、いつでもスイッチから構成を取得できる必要があります。たとえば、BGP の場合、`/api/mo/sys/bgp.json?rsp-subtree=full&rsp-prop-include=set-config-only` のようになります。

b) 機能のペイロードは、置き換える MO（たとえば `bgp` ）で始まる必要があります。

次に例を示します。

```
{
            "bgpInst": {
              "attributes": {
```

```
      "asn": "100",
      "rn": "inst"
    },
    "children": [

    ... content removed for brevity ...

      {
        "bgpDom": {
          "attributes": {
            "name": "vrf1",
            "rn": "dom-vrf1"
          },
          "children": [
            {
              "bgpPeer": {
                "attributes": {
                  "addr": "10.1.1.1",
                  "inheritContPeerCtrl": "",
                  "rn": "peer-[10.1.1.1]"
                }
              }
            }
          ]
        }
      },
      {
        "bgpDom": {
          "attributes": {
            "name": "default",
            "rn": "dom-default",
            "rtrId": "1.1.1.1"
          }
        }
      }
    ]
  }
}
```

Step 2 `/api/mo/sys/bgp.json?rsp-subtree=full&rsp-prop-include=set-config-only` を使用して、構成の置換に使用した DN で GET を送信します。

Step 3 （オプション）送信したペイロードを、置き換えた DN の GET と比較します。GET のペイロードは、送信したペイロードと同じである必要があります。

REST PUT の構成置換のトラブルシューティング

以下は、REST Put 操作による構成置換が成功しない場合のトラブルシューティングに役立つ手順です。

Step 1 要求が有効かどうかを確認します。

URL、操作、およびペイロードが有効である必要があります。たとえば、URL が `api/mo/sys/foo.json` の場合、ペイロードは `foo` で始まる必要があります。

Step 2 ペイロードが有効であり、次のような構成プロパティのみが含まれていることを確認します。

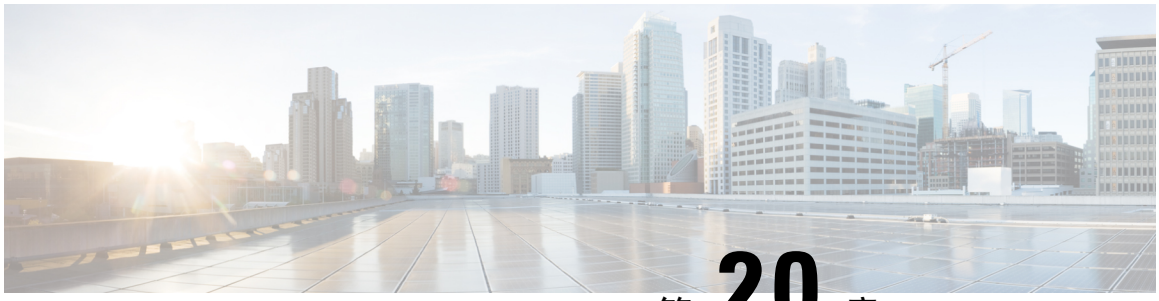
- 正常に設定されたもの
- 有効なデバイス構成から取得したもの

構成プロパティのみを取得するには、`rsp-subtree=full&rsp-prop-include=set-config-only` をフィルタリングする GET を使用します。

Step 3 ペイロードを検証するには、DME POST 操作を使用してペイロードをスイッチに送信します。

Step 4 エラーをチェックして、MO の名前とプロパティがあることを確認します。

Step 5 ペイロードにも MO の名前とプロパティが含まれていることを確認します。



第 20 章

NX-API 開発者サンドボックス

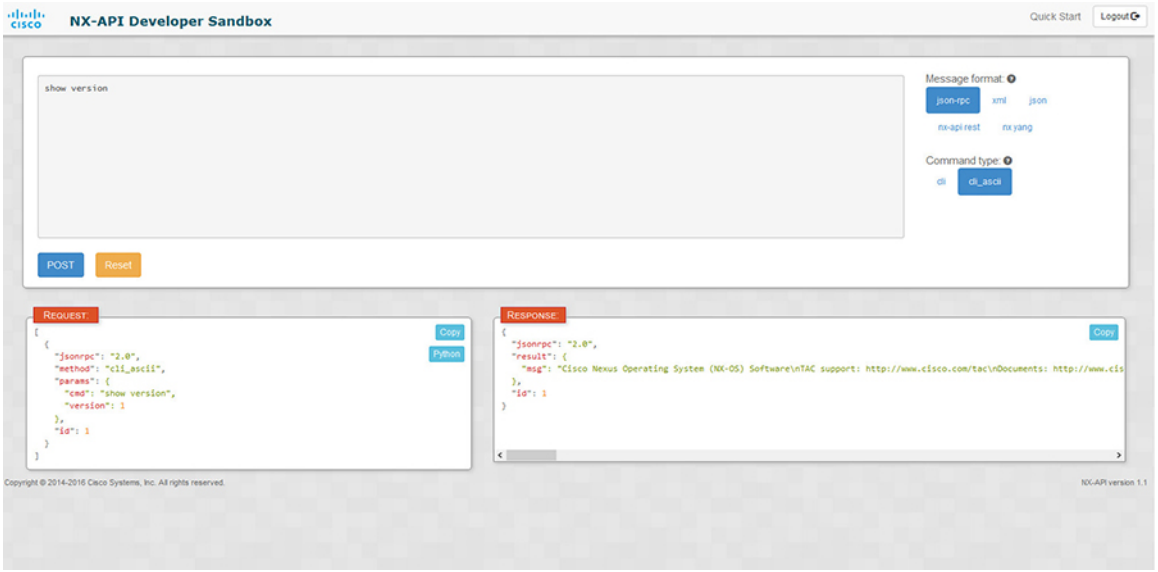
NX-API 開発者サンドボックス:9.2（2）より前の NX-OS リリース

About the NX-API デベロッパー サンドボックス

NX-API Developer Sandbox は、スイッチでホストされる Web フォームです。NX-OS CLI コマンドを同等の XML または JSON ペイロード。

図に示すように、Web フォームは 3 つのペイン（コマンド（上部ペイン）、要求、および応答）を持つ 1 つの画面です。

図 *1* : リクエストと出力応答の例を含む ***NX-API*** デベロッパー サンドボックス



コマンド ペインのコントロールを使用すると、サポートされている API のメッセージフォーマット（NX-API REST など）とコマンド タイプ（XML や JSON など）を選択できます。使用可能なコマンド タイプ オプションは、選択したメッセージ フォーマットによって異なります。

コマンド ペインに 1 つ以上の CLI コマンドを入力するか貼り付けると、Web フォームはコマンドを API ペイロードに変換し、構成エラーをチェックし、結果のペイロードを要求ペインに表示します。次に、コマンドペインの POST ボタンを使用して、ペイロードをサンドボックスからスイッチに直接送信することを選択した場合、応答ペインに API 応答が表示されます。

Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **Send** in the Sandbox commits the command to the switch, which can result in a configuration or state change.
- Some feature configuration commands are not available until their associated feature has been enabled. For example, configuring a BGP router requires first enabling BGP with the **feature bgp** command. Similarly, configuring an OSPF router requires first enabling OSPF with the **feature ospf** command. This also applies to **evpn esi multihoming**, which enables its dependent commands such as **evpn multihoming core-tracking**. For more information about enabling features to access feature dependent commands, see the .
- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.
- When you use XML or JSON as the Message Type for CLI input, you can use semicolon to separate multiple commands on the same line. However, when you use JSON RPC as the Message Type for CLI input, you cannot enter multiple commands on the same line and separate them with a semicolon (;).

For example, assume that you want to send **show hostname** and **show clock** commands through JSON RPC as the following.

In the Sandbox, you enter the CLIs as follows.

```
show hostname ; show clock
```

In the JSON RPC request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname ; show clock",
      "version": 1
    },
    "id": 1
  }
]
```

When you send the request, the response returns the following error.

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32602,
    "message": "Invalid params",
    "data": {
      "msg": "Request contains invalid special characters"
    }
  },
  "id": 1
}
```

This situation occurs because the Sandbox parses each command in a JSON RPC request as individual items and assigns an ID to each. When using JSON RPC requests, you cannot use internal punctuation to separate multiple commands on the same line. Instead, enter each command on a separate line and the request completes sucessfully.

Continuing with the same example, enter the commands as follows in the NX-API CLI.

```
show hostname
show clock
```

In the request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
```

```
  }
]
```

The response completes successfully.

```
[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]
```

メッセージ フォーマットとコマンド タイプの構成

[メッセージ フォーマット（Message Format）]と**[コマンド タイプ（Command Type）]**は、コマンドペイン（上部ペイン）の右上隅で構成されます。**[メッセージ フォーマット（Message Format）]**で、使用する API プロトコルのフォーマットを選択します。開発者サンドボックスは、次の API プロトコルをサポートしています。

表 9 : NX-OS API プロトコル

プロトコル	説明
json-rpc	JSON ペイロードで NX-OS CLI コマンドを配信するために使用できる標準の軽量リモート プロシージャ コール（RPC）プロトコル。JSON-RPC 2.0 仕様は、jsonrpc.org によって概説されています。
xml	XML ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。
json	JSON ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。
nx-api rest	内部 NX-OS データ管理エンジン（DME）モデルで管理対象オブジェクト（MO）とそのプロパティを操作および読み取るための Cisco NX-API 独自のプロトコル。Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、https://developer.cisco.com/site/cisco-nexus-nx-api-references/ を参照してください。

プロトコル	説明
nx yang	構成および状態データ用の YANG（「Yet Another Next Generation」）データモデリング言語。

[メッセージ フォーマット（Message Format）] を選択すると、**[コマンド タイプ（Command Type）]** オプションのセットが**[メッセージ フォーマット（Message Format）]** コントロールのすぐ下に表示されます。**[コマンド タイプ（Command Type）]** の設定は、入力 CLI を制限でき、**[要求（Request）]**と**[応答（Response）]**のフォーマットを決定できます。オプションは、選択した**[メッセージ フォーマット（Message Format）]** によって異なります。各**[メッセージ フォーマット（Message Format）]** について、次の表で **[コマンド タイプ（Command Type）]** オプションについて説明します。

表 *10 :* コマンド タイプ

メッセージ形式	コマンド タイプ
json-rpc	• cli — show または構成コマンド • cli_ascii — show または構成コマンド、フォーマットせずに出力
xml	• cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。 • cli_show_ascii — コマンドを表示、フォーマットせずに出力 • cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。 • bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。 （注） スイッチで bash シェルを有効にする必要があります。

メッセージ形式	コマンド タイプ
json	• cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。 • cli_show_ascii—コマンドを表示、フォーマットせずに出力 • cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。 • bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。 （注） スイッチで bash シェルを有効にする必要があります。
nx-api rest	• cli — 構成コマンド
nx yang	• json — ペイロードに JSON 構造が使用されます • xml — XML 構造がペイロードに使用されます

出力チャンク

大量の show コマンド出力を処理するために、一部の NX-API メッセージフォーマットでは、show コマンドの出力チャンクがサポートされています。この場合、**[チャンク モードを有効にする（Enable chunk mode）]**チェックボックスが、セッション ID (**SID**) 入力ボックスとともに**[コマンド タイプ（Command Type）]** コントロールの下に表示されます。

チャンクが有効な場合、応答は複数の「チャンク」で送信され、最初のチャンクが即時のコマンド応答で送信されます。応答メッセージの次のチャンクを取得するには、前の応答メッセージのセッション ID に設定された **SID** を使用して NX-API 要求を送信する必要があります。

デベロッパー サンドボックスを使用

デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する

ヒント オンライン ヘルプは、サンドボックス ウィンドウの右上隅にある **[クイック スタート（Quick Start）]** をクリックすると利用できます。

レスポンス コードやセキュリティ メソッドなどの詳細については、「NX-API CLI」の章を参照してください。

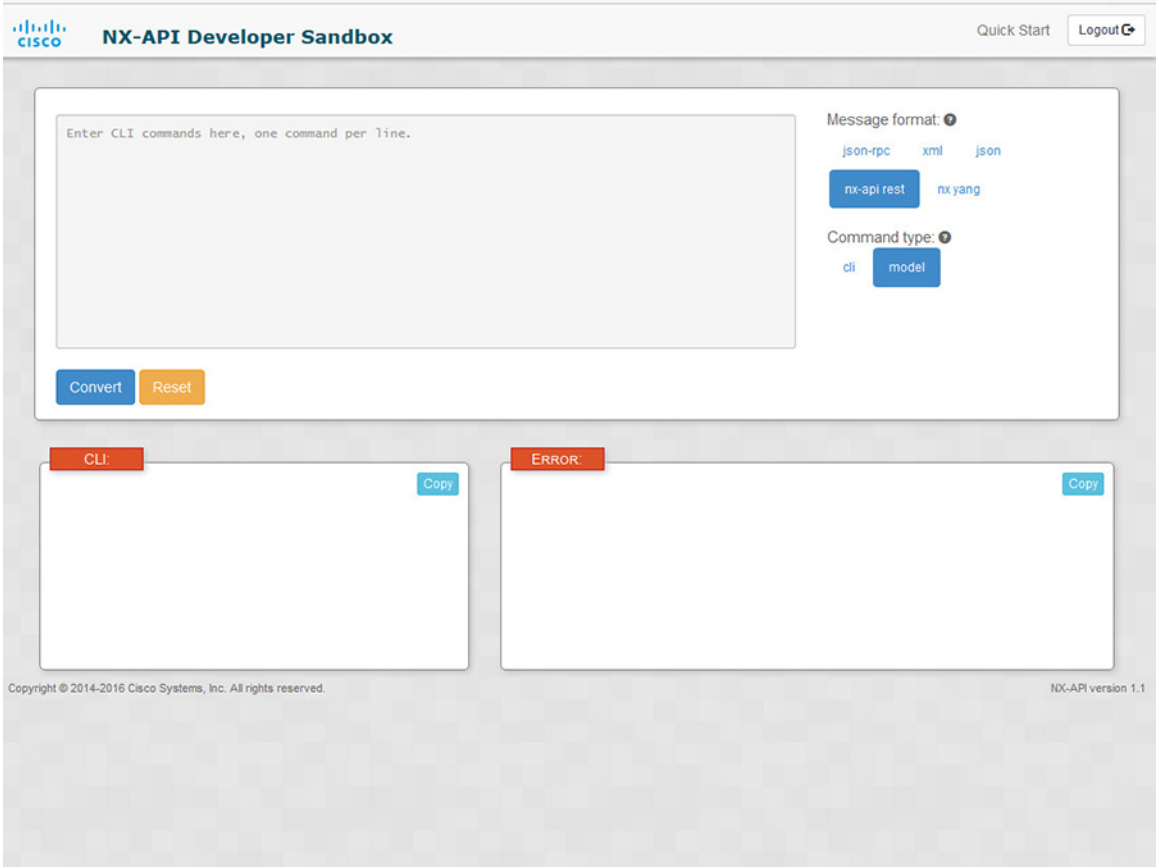
構成コマンドはサポートされていません。

Step 1 使用する API プロトコルの**[メッセージ形式（Message Format）]**と**[コマンド タイプ（Command Type）]**を構成します。

詳細な手順については、メッセージフォーマットとコマンドタイプの構成（228 ページ）を参照してください。

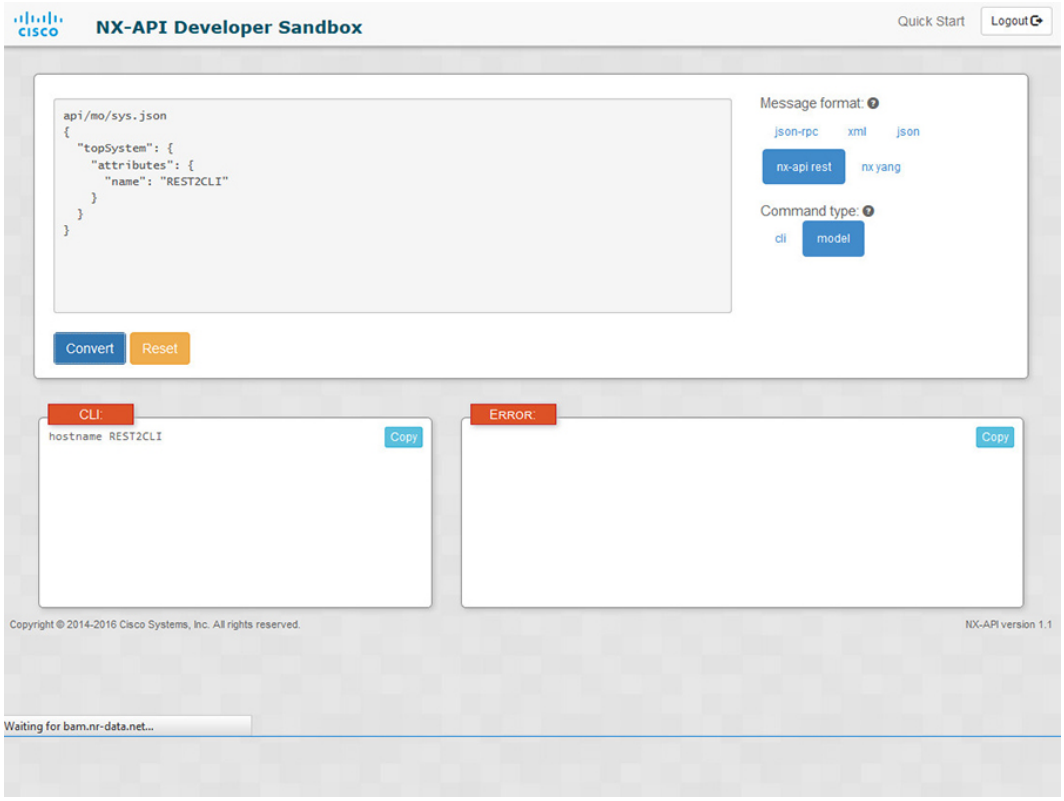
Step 2 上部ペインのテキスト エントリ ボックスに、NX-OS CLI 構成コマンドを 1 行に 1 つずつ入力するか貼り付けます。

上部ペインの下部にある **[リセット（Reset）]** をクリックすると、テキスト エントリ ボックス (および **[要求（Request）]** ペインと **[応答（Response）]** ペイン) の内容を消去できます。



Step 3 トップ ペインの最下部にある **[変換（Convert）]** をクリックします。

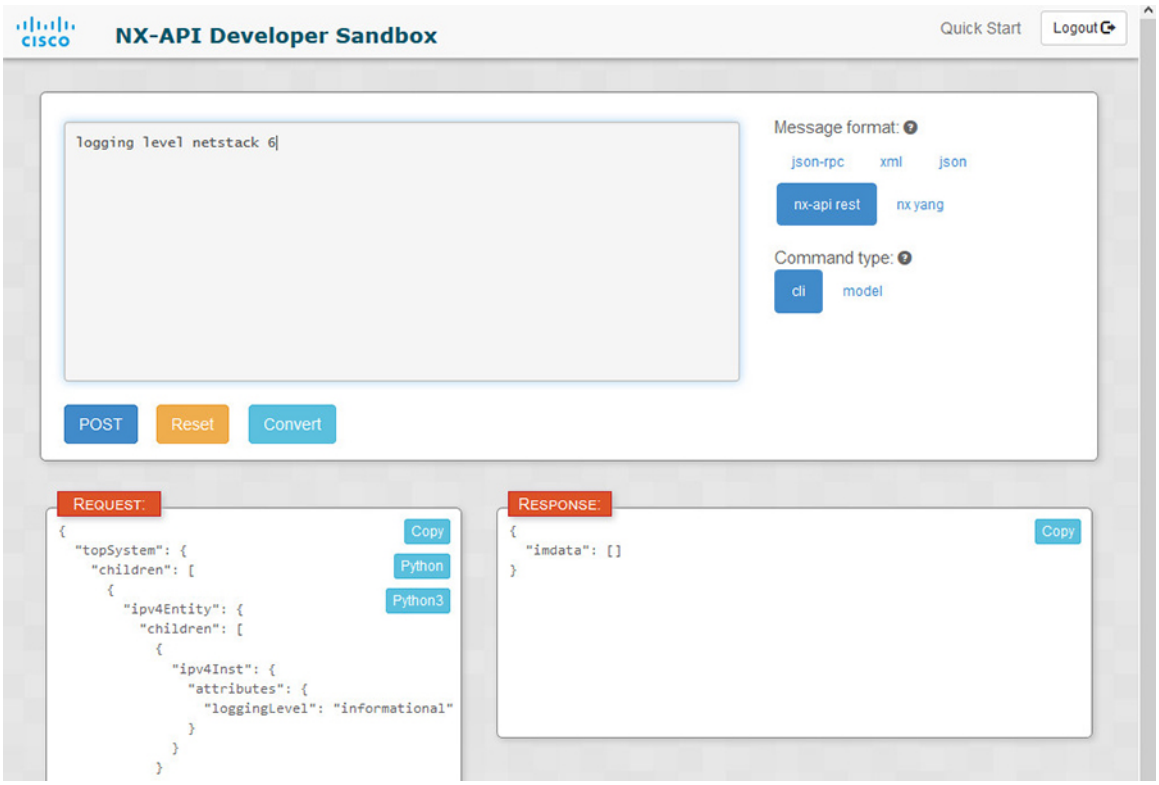
CLI コマンドに構成エラーが含まれていない場合、ペイロードは**[要求（Request）]**ペインに表示されます。エラーが存在する場合は、説明のエラー メッセージが **[応答（Response）]** ペインに表示されます。



Step 4 **[リクエスト（Request）]**ペインに有効なペイロードが表示されている場合は、**POST** をクリックして、ペイロードを API 呼び出しとしてスイッチに送信できます。

スイッチからのレスポンスは **[Response（応答）]** ペインに表示されます。

警告 **POST** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。



Step 5 ペインで **[コピー（Copy）]** をクリックすると、**[要求（Request）]** ペインまたは **[応答（Response）]** ペインの格納ファイルをクリップボードにコピーできます。

Step 6 **[リクエスト（Request）]** ペインで **Python** をクリックすると、クリップボード上のリクエストの Python 導入を取得できます。

デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換

ヒント オンライン ヘルプは、サンドボックス ウィンドウの右上隅にある **[クイック スタート（Quick Start）]** をクリックすると利用できます。

レスポンス コードやセキュリティ メソッドなどの詳細については、「NX-API CLI」の章を参照してください。

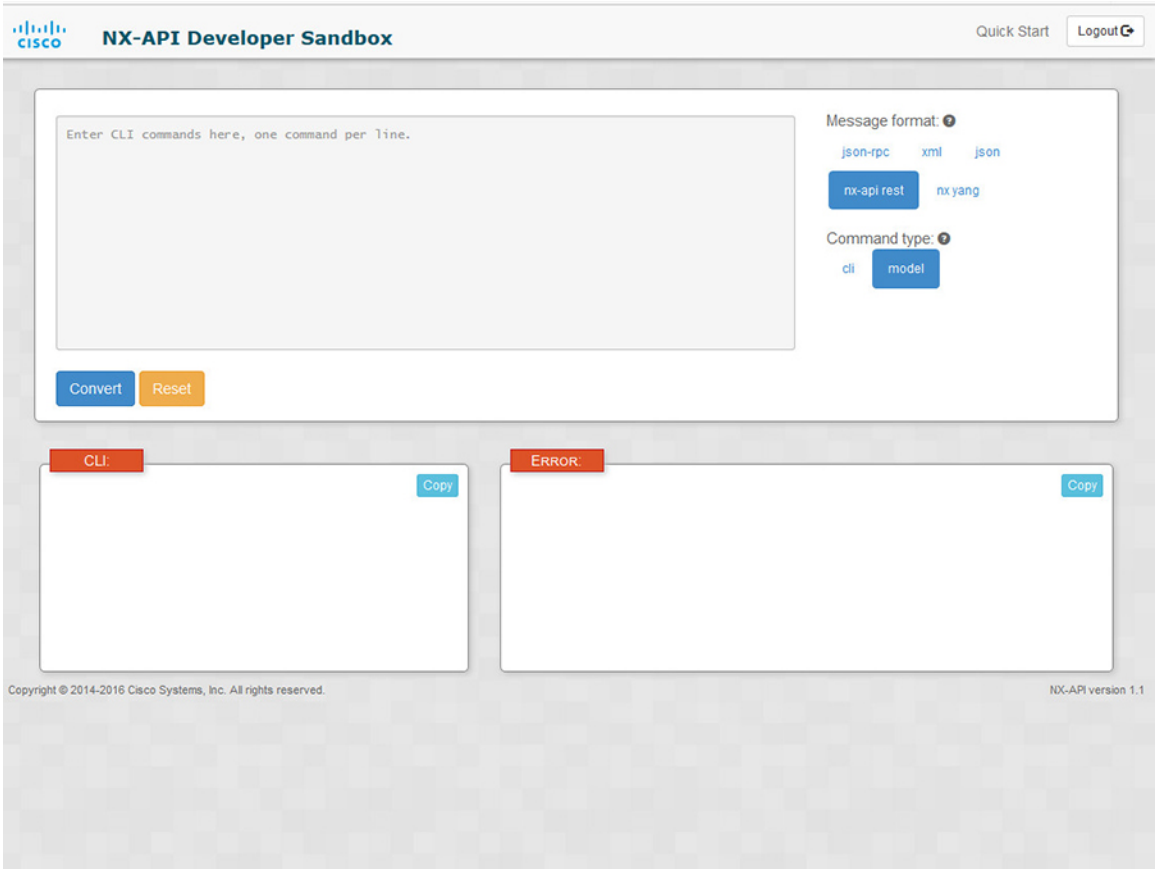
手順の概要

1. **メッセージ フォーマット**として **nx-api rest** を選択し、**コマンド タイプ**として **model** を選択します。
2. 上部ペインのテキスト入力ボックスに DN とペイロードを入力します。次に、上部ペインの下にある **[変換（Convert）]** ボタンをクリックします。

手順の詳細

Step 1 メッセージ フォーマットとして **nx-api rest** を選択し、コマンド タイプとして **model** を選択します。

例：

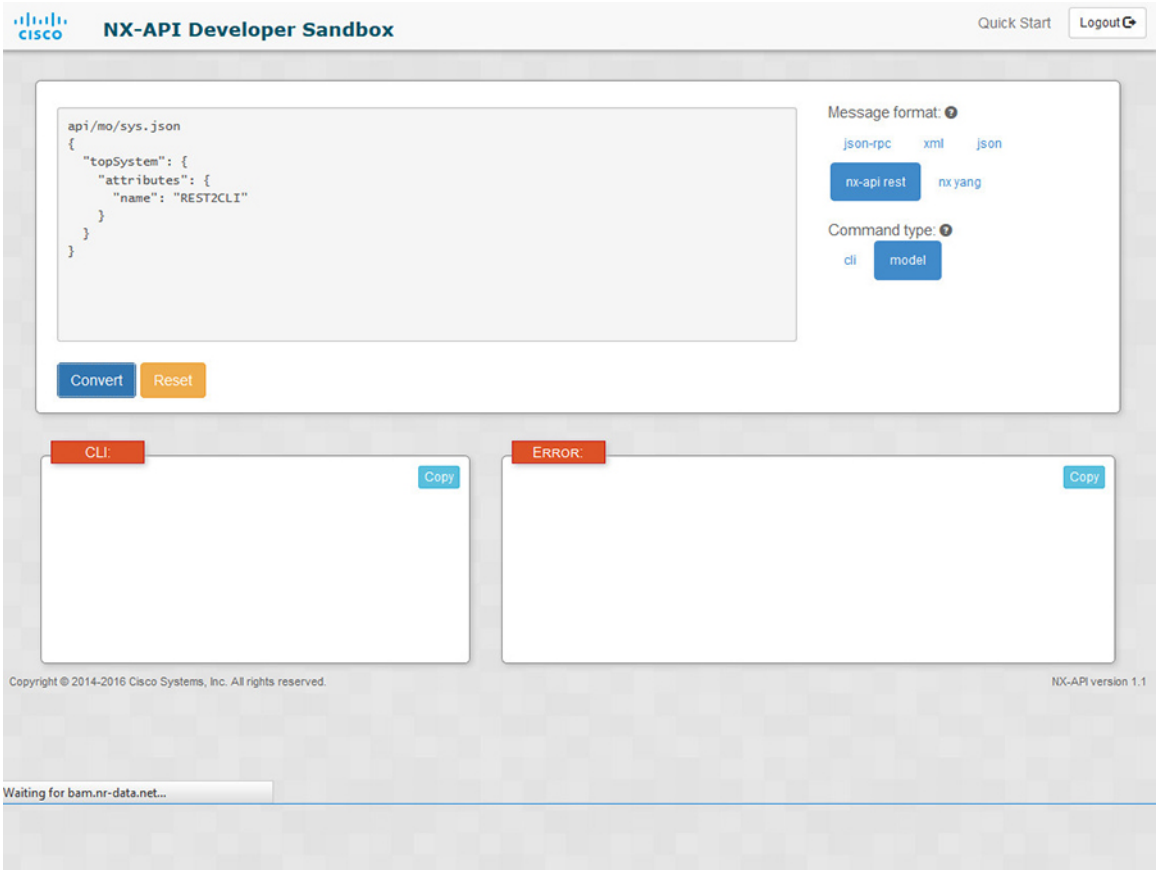


Step 2 上部ペインのテキスト入力ボックスに DN とペイロードを入力します。次に、上部ペインの下にある **[変換（Convert）]** ボタンをクリックします。

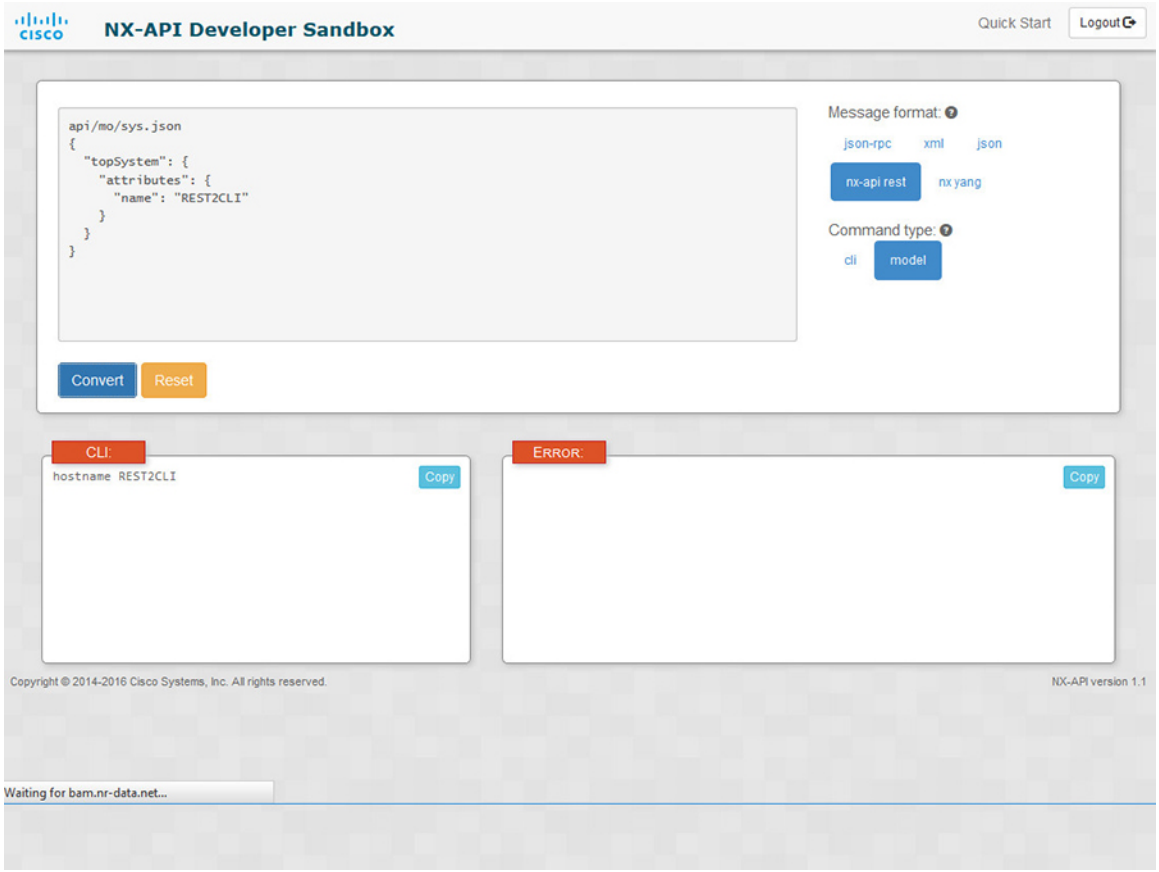
例：

この例では、DN は **/api/mo/sys.json** であり、NX-API REST ペイロードは次のとおりです：

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```



[変換（Convert）] ボタンをクリックすると、次の図に示すように、同等の CLI が **CLI** ペインに表示されます。


NX-API Developer Sandbox
Quick Start
Logout
api/mo/sys.json
{
"topSystem": {
"attributes": {
"name": "REST2CLI"
}
}
}
Message format:
json-rpc
xml
json
nx-api rest
nx yang
Command type:
cli
model
Convert
Reset
CLI:
hostname REST2CLI
Copy
ERROR:
Copy
Copyright © 2014-2016 Cisco Systems, Inc. All rights reserved.
NX-API version 1.1
Waiting for bam.nr-data.net...

（注）

デベロッパー サンドボックスは、サンドボックスが CLI を NX-API REST ペイロードに変換した場合でも、すべてのペイロードを同等の CLI に変換することはできません。以下は、ペイロードが CLI コマンドに完全に変換するのを妨げる可能性のあるエラーの原因のリストです。

表 *11 : REST2CLI* エラーの原因

ペイロードの問題	結果

ペイロードに、MO に存在しない属性が含まれています。
例：
`api/mo/sys.json`
`{ "topSystem": { "children": [ { "interfaceEntity": { "children": [ { "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": "totallyFake" } } } ] } } ] } }`
	[エラー（Error）] ペインは、属性に関連するエラーを返します。
	例：
	CLI
	要素「l1PhysIf」の不明な属性「fakeattribute」の**[エラー（Error）]**

ペイロードには、変換がまだサポートされていない MO が含まれています。
例：
`api/mo/sys.json`
`{ "topSystem": { "children": [ { "dhcpEntity": { "children": [ { "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" } } } ] } } ] } }`
	[エラー（Error）] ペインは、サポートされていない MO に関連するエラーを返します。
	例：
	CLI
	[エラー（Error）] ［「sys/dhcp」のサブツリー全体が変換されていません。（The entire subtree of "sys/dhcp" is not converted.）］

ペイロードの問題	結果

NX-API 開発者サンドボックス：NX-OS リリース 9.2（2）以降

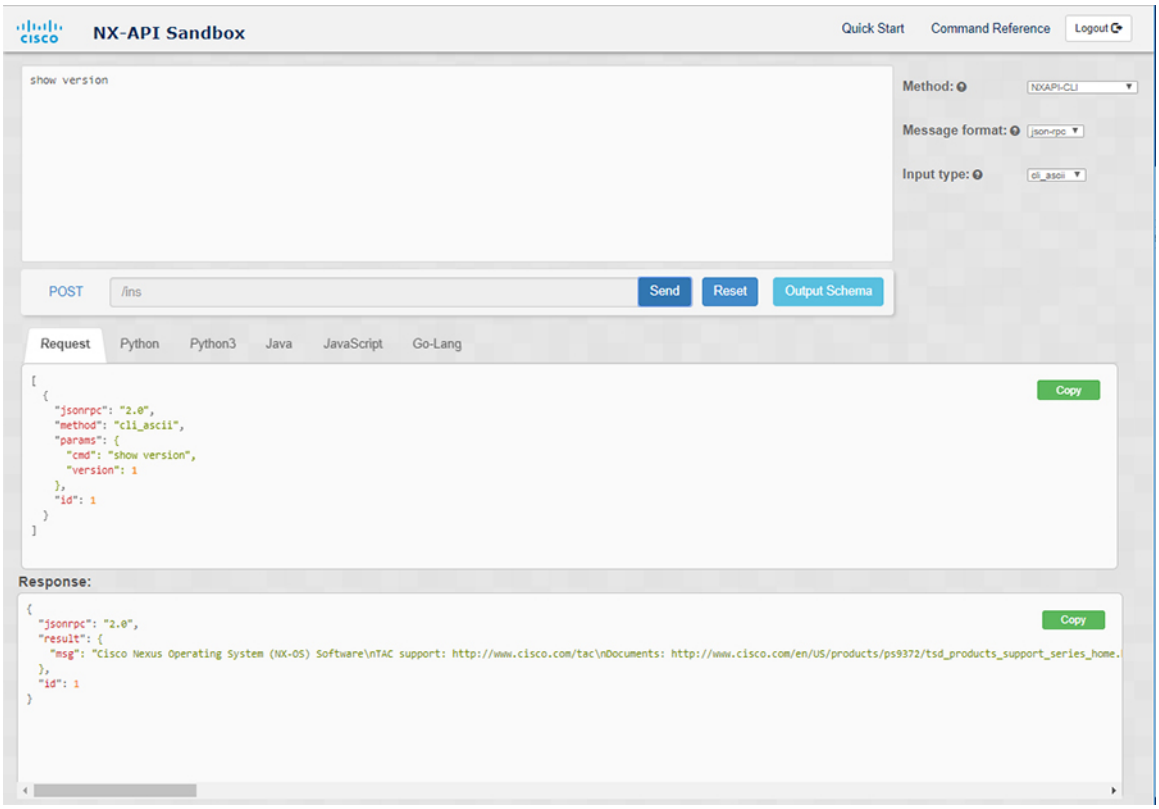
About the NX-API デベロッパー サンドボックス

Cisco NX-API Developer Sandbox は、スイッチでホストされる Web フォームです。NX-OS CLI コマンドを同等の XML または JSON ペイロードに変換し、NX-API REST ペイロードを同等の CLI に変換します。

Web フォームは、次の図に示すように、コマンド（上部のペイン）、要求（中央のペイン）、および応答（下部のペイン）の 3 つのペインを持つ 1 つの画面です。指定名（DN）フィールドは、コマンドペインとリクエストペインの間にあります（下図の **POST** と**送信**オプションの間にあります）。

リクエストペインにも一連のタブがあります。各タブは、**Python**、**Python3**、**Java**、**JavaScript**、**Go-Lang** の異なる言語を表します。各タブでは、それぞれの言語でリクエストを表示できます。たとえば、CLI コマンドを XML または JSON ペイロードに変換した後、**[Python]** タブをクリックして、スクリプトの作成に使用できる Python でのリクエストを表示します。

図 *2:* リクエストと出力応答の例を含む ***NX-API*** デベロッパー サンドボックス



コマンドペインのコントロールを使用すると、NX-API REST などのサポートされている API、モデル（ペイロード）や CLI などの入力タイプ、および XML や JSON などのメッセージ形式を選択できます。使用可能なオプションは、選択した方法によって異なります。

NXAPI-REST（DME）メソッドを選択し、1 つ以上の CLI コマンドをコマンド ペインに入力するか貼り付けて、**[変換]** をクリックすると、Web フォームはコマンドを REST API ペイロードに変換し、構成エラーをチェックし、要求ペインに結果のペイロードを表示します。次に、ペイロードをサンドボックスからスイッチに直接送信することを選択した場合（**POST** オプションを選択して **[SEND]** をクリック）、[応答] ペインに API 応答が表示されます。詳細については、デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する （247 ページ）を参照してください。

逆に、Cisco NX-API Developer Sandbox はペイロードの設定エラーをチェックし、対応する CLI を [応答] ペインに表示します。詳細については、「デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換 （250 ページ）」を参照してください。

Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **Send** in the Sandbox commits the command to the switch, which can result in a configuration or state change.

- Some feature configuration commands are not available until their associated feature has been enabled. For example, configuring a BGP router requires first enabling BGP with the **feature bgp** command. Similarly, configuring an OSPF router requires first enabling OSPF with the **feature ospf** command. This also applies to **evpn esi multihoming**, which enables its dependent commands such as **evpn multihoming core-tracking**. For more information about enabling features to access feature dependent commands, see the .
- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.
- When you use XML or JSON as the Message Type for CLI input, you can use semicolon to separate multiple commands on the same line. However, when you use JSON RPC as the Message Type for CLI input, you cannot enter multiple commands on the same line and separate them with a semicolon (;).

  For example, assume that you want to send **show hostname** and **show clock** commands through JSON RPC as the following.

  In the Sandbox, you enter the CLIs as follows.

```
show hostname ; show clock
```

  In the JSON RPC request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname ; show clock",
      "version": 1
    },
    "id": 1
  }
]
```

  When you send the request, the response returns the following error.

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32602,
    "message": "Invalid params",
    "data": {
      "msg": "Request contains invalid special characters"
    }
  },
  "id": 1
}
```

  This situation occurs because the Sandbox parses each command in a JSON RPC request as individual items and assigns an ID to each. When using JSON RPC requests, you cannot use internal punctuation to separate multiple commands on the same line. Instead, enter each command on a separate line and the request completes sucessfully.

  Continuing with the same example, enter the commands as follows in the NX-API CLI.

```
show hostname
show clock
```

  In the request, the input is formatted as follows.

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
  }
]
```

The response completes successfully.

```
[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]
```

メッセージ フォーマットと入力タイプの構成

メソッド、**メッセージ形式**、および **入力タイプ** は、コマンド ペイン（上部のペイン）の右上隅で構成されます。**[メソッド]** で、使用する API プロトコルの形式を選択します。Cisco NX-API Developer Sandbox は、次の API プロトコルをサポートしています。

表 *12 : NX-OS API* プロトコル

プロトコル	説明
NXAPI-CLI	XML または JSON ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。

プロトコル	説明
NXAPI-REST（DME）	内部 NX-OS データ管理エンジン（DME）モデルで管理対象オブジェクト（MO）とそのプロパティを操作および読み取るための Cisco NX-API 独自のプロトコル。NXAPI-REST（DME）プロトコルは、次の方法から選択できるドロップダウン リストを表示します。 • **POST** • **GET** • **PUT** • **DELETE** Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、https://developer.cisco.com/site/cisco-nexus-nx-api-references/ を参照してください。
RESTCONF（Yang）	構成および状態データ用の YANG（「Yet Another Next Generation」）データモデリング言語。 RESTCONF（Yang）プロトコルは、次の方法から選択できるドロップダウン リストを表示します。 • **POST** • **GET** • **PUT** • **PATCH** • **DELETE**

メソッドを選択すると、**メッセージ形式**または**入力タイプ**のオプションのセットがドロップダウン リストに表示されます。**メッセージ形式**は、入力 CLI を制約し、**要求**と**応答**の形式を決定できます。オプションは、選択した**メソッド**によって異なります。

次の表では、各**メッセージ形式**の**入力/コマンド タイプ** オプションについて説明します。

表 *13*: コマンド タイプ

方法	メッセージ形式	入力/コマンド タイプ
NXAPI-CLI	json-rpc	• cli — show または構成コマンド • cli_ascii — show または構成コマンド、フォーマットせずに出力 • cli-array — show コマンド。cli に似ていますが、cli_array を使用すると、データは 1 つの要素のリスト、または角括弧 [] で囲まれたアレイとして返されます。
NXAPI-CLI	xml	• cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。 • cli_show_ascii — コマンドを表示、フォーマットせずに出力 • cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。 • bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。 （注） スイッチで bash シェルを有効にする必要があります。

方法	メッセージ形式	入力/コマンド タイプ
NXAPI-CLI	json	• cli_show — コマンドを表示します。コマンドがXML出力をサポートしていない場合、エラー メッセージが返されます。 （注） Cisco NX-OS リリース 9.3(3) 以降では、cli_show コマンドよりも cli_show_array コマンドが推奨されます。 • cli_show_array — show コマンドcli_show に似ていますが、cli_show_array を使用すると、データは角括弧 [ ] で囲まれた 1 つの要素のリストまたは配列として返されます。 • cli_show_ascii — コマンドを表示、フォーマットせずに出力 • cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。 • bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。 （注） スイッチで bash シェルを有効にする必要があります。
NXAPI-REST（DME）		• cli — CLI からモデルへの変換 • model — モデルから CLI への変換。
RESTCONF（Yang）	• json — ペイロードに JSON 構造が使用されます • xml — XML 構造がペイロードに使用されます	

出力チャンク

JSON および XML NX-API メッセージ形式を使用すると、10 MB のチャンクで大きな show コマンド応答を受信できます。受信すると、チャンクが連結されて、有効な JSON オブジェクトまたは XML 構造が作成されます。出力チャンクを示すサンプル スクリプトを表示するには、次のリンクをクリックし、リリース 9.3x に対応するディレクトリを選択します：Cisco NX-OS NXAPI。

（注） チャンク JSON モードの場合、ブラウザーまたは Python スクリプト パーツは有効な JSON 出力を提供しません（終了タグはありません）。チャンク モードを使用して有効な JSON を取得するには、ディレクトリで提供されるスクリプトを使用します。

即時のコマンド応答で最初のチャンクを受け取ります。これには、セッション ID を含む **sid** フィールドも含まれます。次のチャンクを取得するには、前のチャンクのセッション ID を **[SID]** テキスト ボックスに入力します。**sid** フィールドの **eoc**（コンテンツの終わり）値で示される最後の応答に到達するまで、プロセスを繰り返します。

チャンク モードは、**JSON** または **XML** フォーマット タイプおよび **cli_show**、 **cli_show_array**,、または **cli_show_ascii** コマンド タイプで **NXAPI-CLI** メソッドを使用する場合に使用できます。チャンク モードの設定の詳細については、チャンク モード フィールドの表を参照してください。

（注） NX-API は、最大 2 つのチャンク セッションをサポートします。

表 *14:* チャンク モード フィールド

フィールド名	説明
チャンク モードを有効にする	**[チャンク モードを有効にする（Enable Chunk Mode）]** チェック ボックスをクリックしてチェック マークを付けると、チャンクが有効になります。チャンク モードを有効にすると、10 MB を超える応答は、最大 10 MB のサイズの複数のチャンクで送信されます。
SID	応答メッセージの次のチャンクを取得するには、**SID** テキスト ボックスに前の応答のセッション ID を入力します。 （注） 使用できる文字は英数字と「_」のみです。無効な文字はエラーを受け取ります。

デベロッパー サンドボックスを使用

デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する

ヒント
- Cisco NX-API デベロッパー サンドボックス ウィンドウの右上隅にあるフィールド名の横にあるヘルプ アイコン（ **?** ）をクリックすると、オンライン ヘルプを利用できます。
- 応答コードやセキュリティ メソッドなどの詳細については、*NX-API CLI* の章を参照してください。
- 構成コマンドはサポートされていません。

Cisco NX-API Developer Sandbox を使用すると、CLI コマンドを REST ペイロードに変換できます。

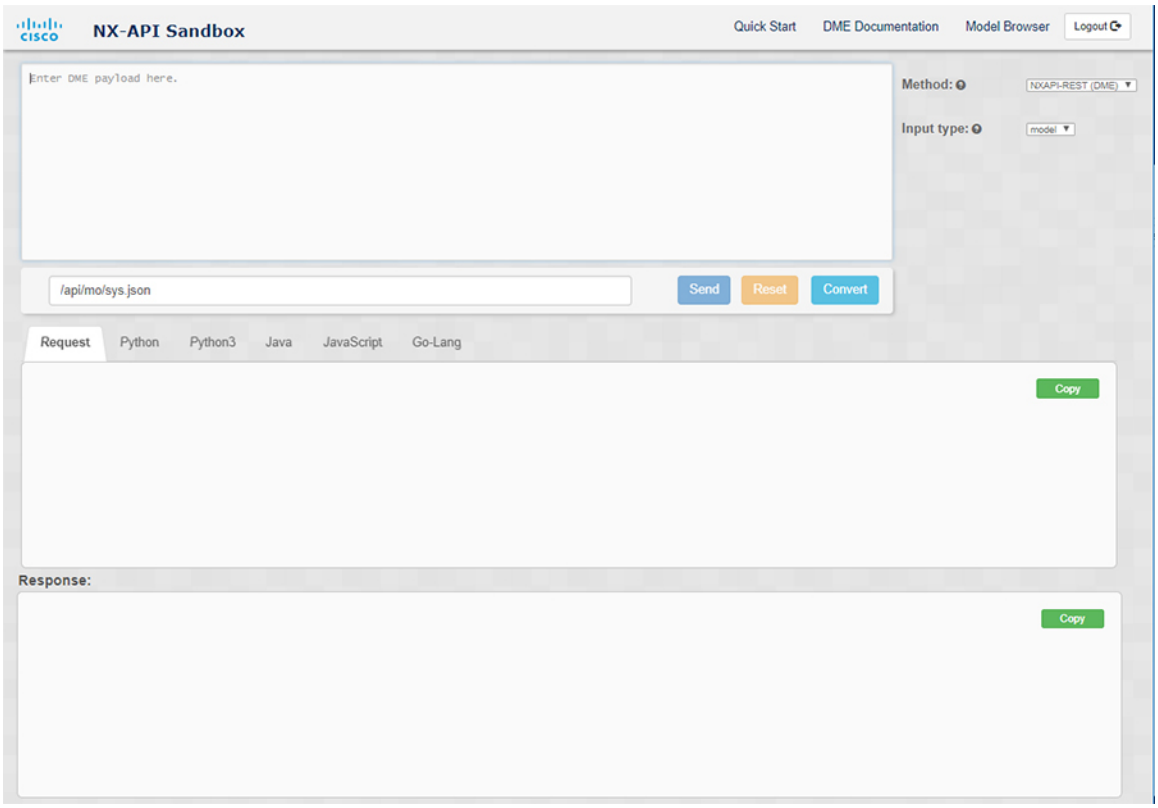
Step 1 **[方法（Method）]** ドロップダウン リストをクリックし、**NXAPI-REST (DME)** を選択します。

[入力タイプ] ドロップダウン リストが表示されます。

Step 2 **[入力（Input）]** タイプドロップダウン リストをクリックし、**cli** を選択します。

Step 3 上部ペインのテキスト エントリ ボックスに、NX-OS CLI 構成コマンドを 1 行に 1 つずつ入力するか貼り付けます。

上部ペインの下部にある **[リセット（Reset）]** をクリックすると、テキスト エントリ ボックス (および **[要求（Request）]** ペインと **[応答（Response）]** ペイン) の内容を消去できます。



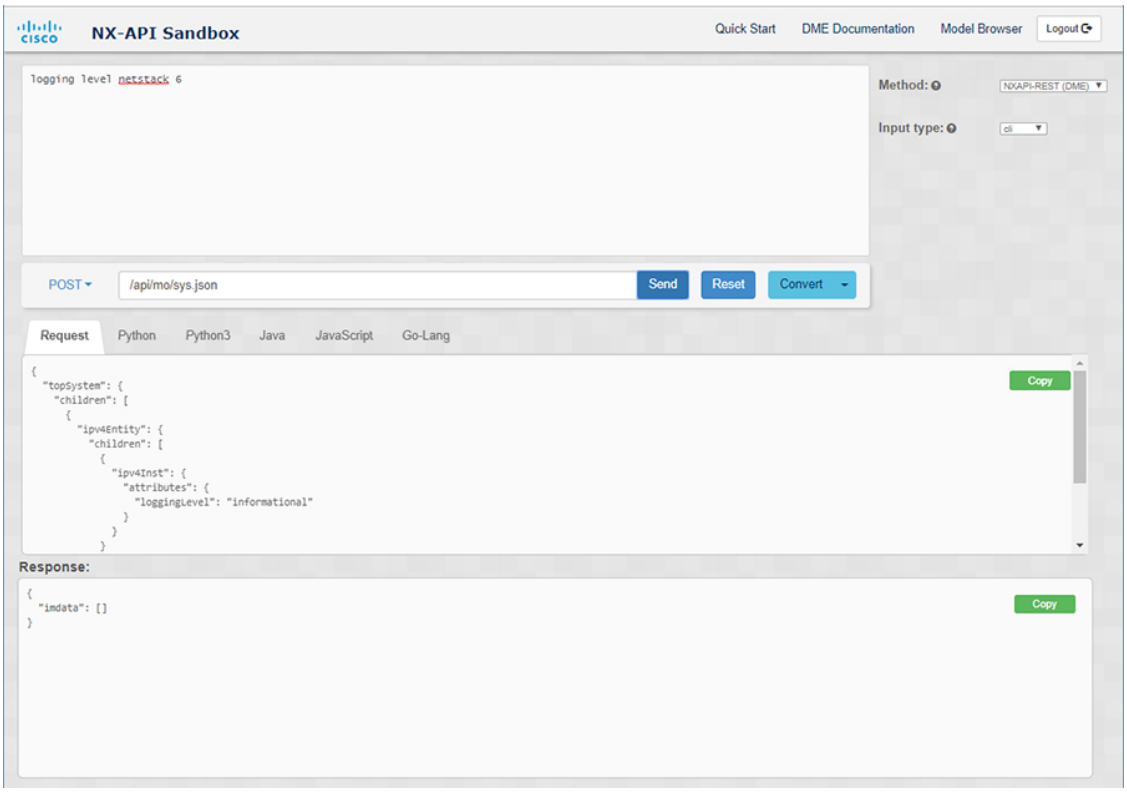
Step 4 [変換（Convert）] をクリックします。

CLI コマンドに構成エラーが含まれていない場合、ペイロードは**[要求（Request）]**ペインに表示されます。エラーが存在する場合は、説明のエラー メッセージが **[応答（Response）]** ペインに表示されます。

Step 5 (オプション) 有効なペイロードを API 呼び出しとしてスイッチに送信するには、**[送信（Send）]** をクリックします。

スイッチからのレスポンスは **[Response（応答）]** ペインに表示されます。

警告 **[送信（Send）]** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。



Step 6 (オプション) ペイロード内の MO の DN を取得するには：

1. **[リクエスト（Request）]** ペインから、**POST** を選択します。
2. **[変換（Convert）]** ドロップダウン リストをクリックし、**[変換 (DN を使用)（Convert (with DN)）]** を選択します。

ペイロードは、ペイロード内の各 MO に対応する DN を含む **dn** フィールドとともに表示されます。

Step 7 (オプション) 新しい構成で現在の構成を上書きする場合：

1. **[変換（Convert）]** ドロップダウン リストをクリックし、**[変換（置換用）（Convert（for Replace））]** を選択します。**[リクエスト（Request）]** ペインには、**[ステータス（status）]** フィールドが**[置換（replace）]**ように設定されたペイロードが表示されます。
2. **[リクエスト（Request）]** ペインから、**POST** を選択します。
3. [送信（Send）] をクリックします。

現在の構成は、投稿された構成に置き換えられます。たとえば、次の構成で開始するとします：

```
interface eth1/2
  description test
  mtu 1501
```

次に、**[変換（置換用）（Convert（for Replace））]**を使用して、次の構成を POST します。

```
interface eth1/2
  description testForcr
```

`mtu` 構成が削除され、新しい説明 (`testForcr`) のみがインターフェイスの下に表示されます。この変更は、**show running-config** と入力すると確認されます。

Step 8 (オプション) **[リクエスト（Request）]** ペインや **[応答（Response）]** ペインなどのペインの内容をコピーするには、**[コピー（Copy）]** をクリックします。それぞれのペインの内容がクリップボードにコピーされます。

Step 9 (オプション) リクエストを以下のいずれかのフォーマットに変換するには、**[リクエスト（Request）]** ペインの適切なタブをクリックします。

- **Python**
- **python3**
- **Java**
- **JavaScript**
- **Go-Lang**

デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換

Cisco NX-API Developer Sandbox を使用すると、REST ペイロードを対応する CLI コマンドに変換できます。このオプションは、NXAPI-REST (DME) メソッドでのみ使用できます。

ヒント

- Cisco NX-API Developer Sandbox のフィールド名の横にあるヘルプ アイコン（**?**）をクリックすると、オンライン ヘルプを利用できます。ヘルプ アイコンをクリックして、それぞれのフィールドに関する情報を取得します。

  応答コードやセキュリティ メソッドなどの詳細については、*NX-API CLI* の章を参照してください。

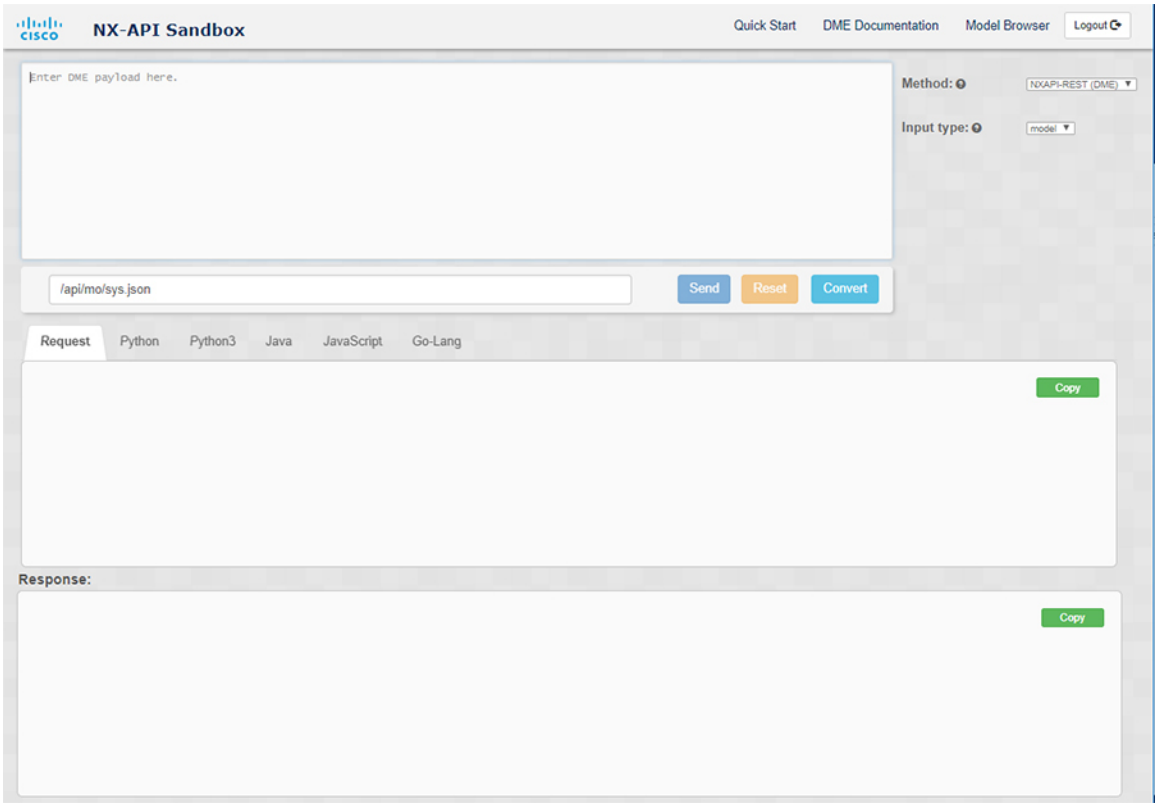
- Cisco NX-API Developer Sandbox の右上隅には、追加情報へのリンクが含まれています。表示されるリンクは、選択した**[方法（Method）]**によって異なります。NXAPI-REST（DME）メソッドに表示されるリンク：

  - **[NX-API リファレンス（NX-API References）]**— 追加の NX-API ドキュメントにアクセスできます。
  - **[DME ドキュメント（DME Documentation）]** — NX-API DME モデル リファレンス ページにアクセスできます。
  - **[モデル ブラウザ（Model Browser）]** — モデル ブラウザである Visore にアクセスできます。Visore ページにアクセスするには、スイッチの IP アドレスを手動で入力する必要がある場合があることに注意してください。

    `https://management-ip-address/visore.html`。

Step 1 **[方法（Method）]** ドロップダウン リストをクリックし、**NXAPI-REST (DME)** を選択します。

例：



Step 2 **[タイプを入力（Input Type）]** タイプドロップダウン リストをクリックし、**[モデル（model）]** を選択します。

Step 3 要求ペインの上にあるフィールドに、ペイロードに対応する指定名（DN）を入力します。

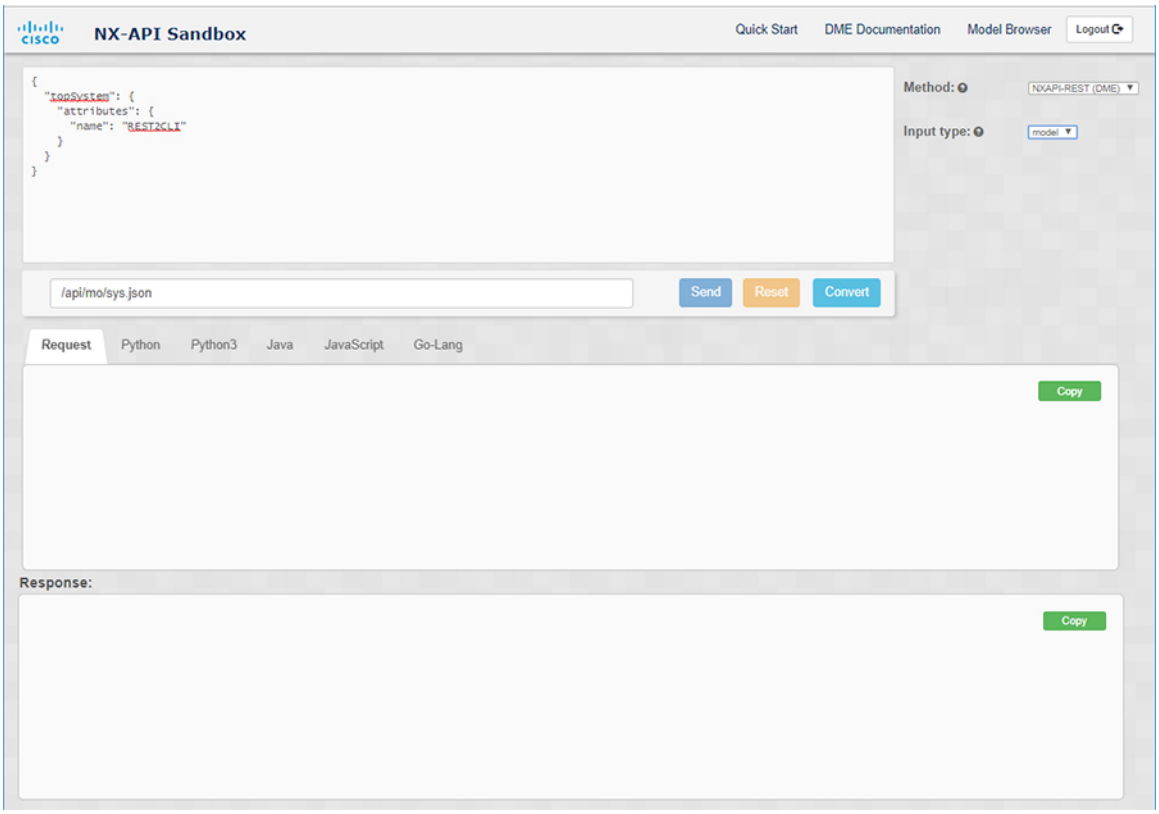
Step 4 コマンド ペインにペイロードを入力します。

Step 5 [変換（Convert）] をクリックします。

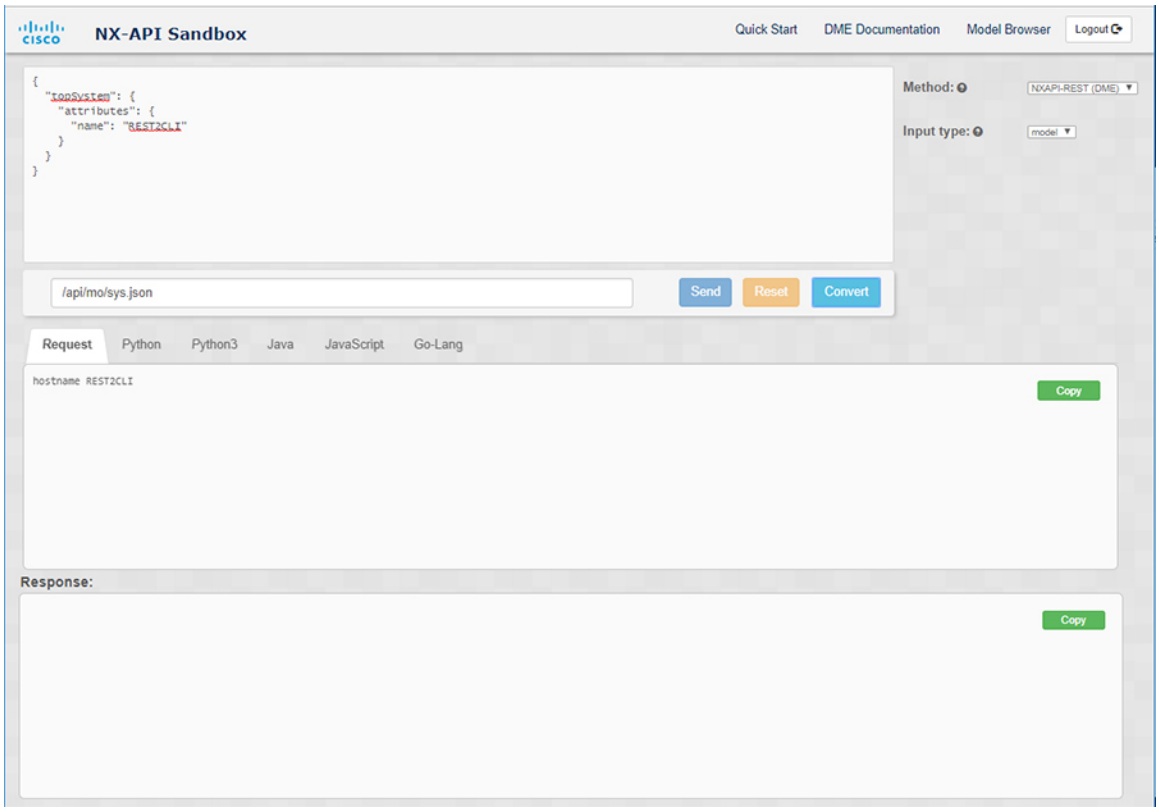
例：

この例では、DN は **`/api/mo/sys.json`** であり、NX-API REST ペイロードは次のとおりです。

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```



[変換（Convert）] ボタンをクリックすると、次の図に示すように、同等の CLI が **CLI** ペインに表示されます。


NX-API Sandbox
Quick Start
DME Documentation
Model Browser
Logout
{
"topSystem": {
"attributes": {
"name": "REST2CLI"
}
}
}
Method:
NXAPI-REST (DME)
Input type:
model
/api/mo/sys.json
Send
Reset
Convert
Request
Python
Python3
Java
JavaScript
Go-Lang
hostname REST2CLI
Copy
Response:
Copy

（注）

Cisco NX-API Developer Sandbox は、サンドボックスが CLI を NX-API REST ペイロードに変換した場合でも、すべてのペイロードを同等の CLI に変換できません。以下は、ペイロードが CLI コマンドに完全に変換するのを妨げる可能性のあるエラーの原因のリストです。

表 *15 : REST2CLI* エラーの原因

ペイロードの問題	結果

ペイロードに、MO に存在しない属性が含まれています。
例：
`api/mo/sys.json`
`{ "topSystem": { "children": [ { "interfaceEntity": { "children": [ { "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": "totallyFake" } } } ] } } ] } }`
	[エラー（Error）] ペインは、属性に関連するエラーを返します。
	例：
	CLI
	要素「l1PhysIf」の不明な属性「fakeattribute」の**[エラー（Error）]**

ペイロードには、変換がまだサポートされていない MO が含まれています。
例：
`api/mo/sys.json`
`{ "topSystem": { "children": [ { "dhcpEntity": { "children": [ { "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" } } } ] } } ] } }`
	[エラー（Error）] ペインは、サポートされていない MO に関連するエラーを返します。
	例：
	CLI
	[エラー（Error）] ［「sys/dhcp」のサブツリー全体が変換されていません。（The entire subtree of "sys/dhcp" is not converted.）］

ペイロードの問題	結果

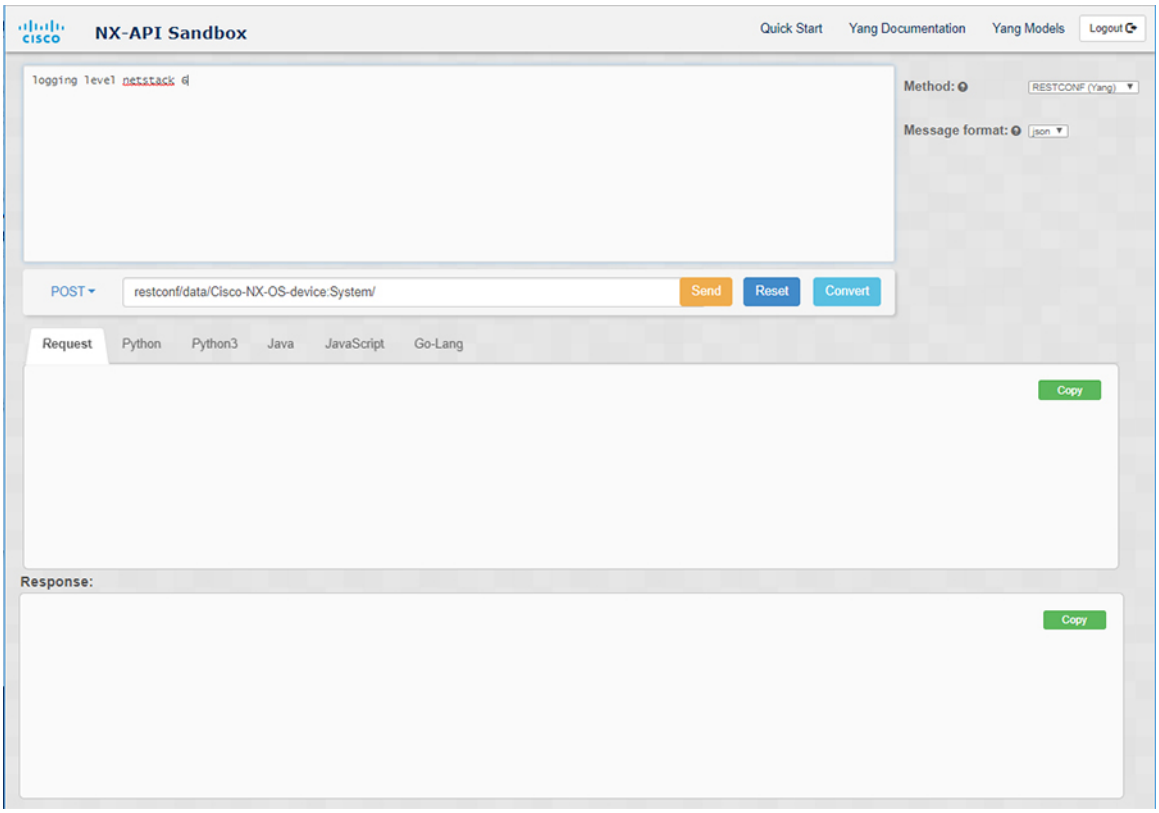
デベロッパー サンドボックスを使用して RESTCONF から json または XML に変換する

ヒント

- Cisco NX-API Developer Sandbox ウィンドウの右上隅にあるヘルプ アイコン（？）をクリックすると、オンライン ヘルプを利用できます。
- [サンドボックス] ウィンドウの右上隅にある **Yang Documentation** リンクをクリックして、Model Driven Programmability with Yang ページに移動します。
- [サンドボックス] ウィンドウの右上隅にある **Yang Models** リンクをクリックして、YangModels GitHub サイトにアクセスします。

Step 1 **[メソッド]** ドロップダウン リストをクリックし、**[RESTCONF (Yang)]** を選択します。

例：



Step 2 **[メッセージ形式]** をクリックし、**json** または **xml** を選択します。

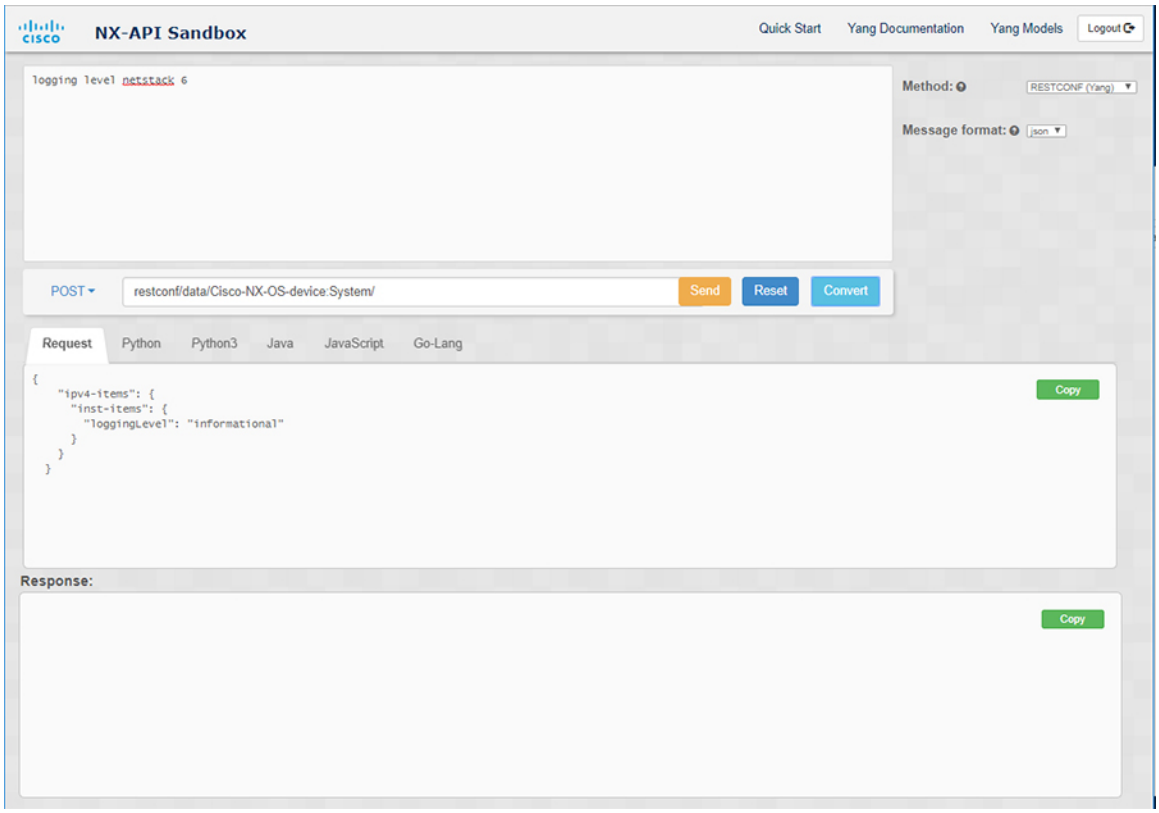
Step 3 上部ペインのテキスト入力ボックスにコマンドを入力します。

Step 4 メッセージ形式を選択します。

Step 5 [変換（Convert）] をクリックします。

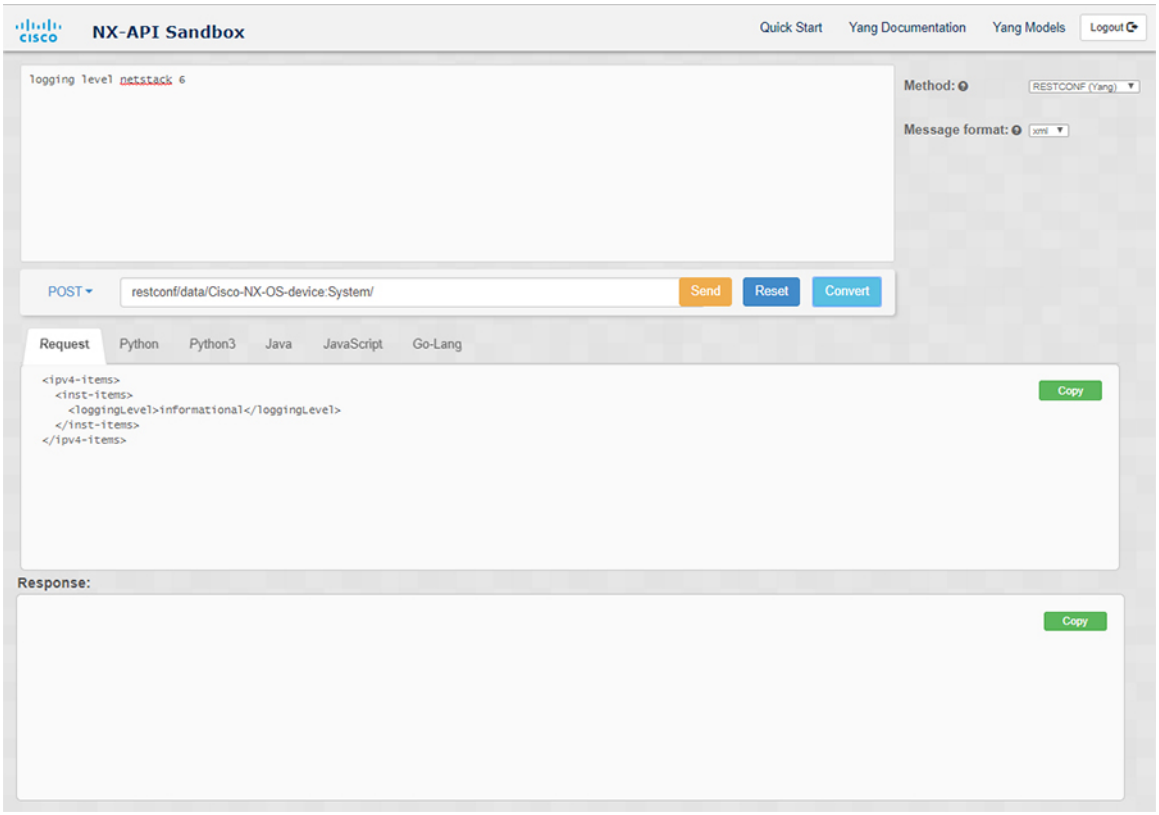
例：

この例では、コマンドは**ログ レベル netstack 6** で、メッセージ形式は json です。



例：

この例では、コマンドは**ログ レベル netstack 6** で、メッセージ形式は xml です。



（注） XML または JSON メッセージ形式を使用して、否定された CLI を Yang ペイロードに変換すると、サンドボックスは警告をスローし、**[送信]** オプションを無効にします。表示される警告メッセージは、メッセージの形式によって異なります。

- XML メッセージ形式の場合 — 「これは Netconf ペイロードであり、DELETE 操作用に生成されているため、Restconf では SEND オプションが無効になっています!」
- JSON メッセージ形式の場合 - 「これは、DELETE 操作用に生成される gRPC ペイロードであるため、Restconf では SEND オプションが無効になっています!」

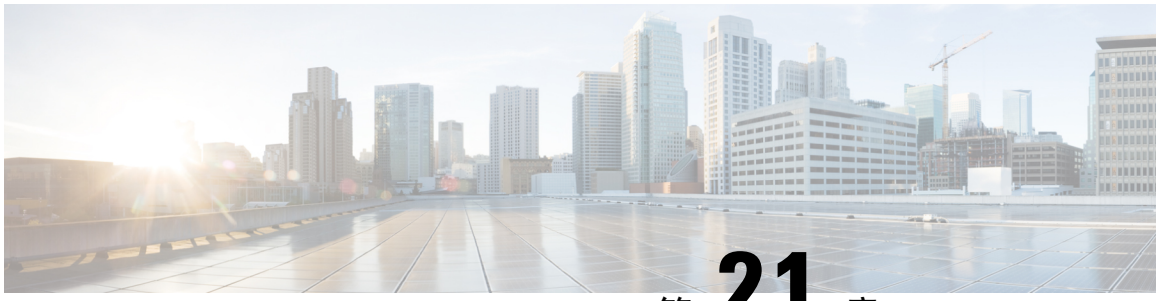
Step 6 **[リクエスト]** ペインの適切なタブをクリックして、リクエストを次の形式に変換することもできます。

- Python
- python3
- Java
- JavaScript
- Go-Lang

（注） [リクエスト] タブの上の領域にあるドロップダウン メニューから [PATCH] オプションを選択した場合、Java で生成されたスクリプトは機能しません。これは Java の既知の制限であり、予期される動作です。

第 IV 部

モデル駆動型プログラマビリティ



第 21 章

NETCONF エージェント

この章は次のトピックで構成されています。

NETCONF エージェントについて

NETCONF（Network Configuration Protocol、ネットワーク構成プロトコル）は、RFC 6241 によって定義されているネットワーク管理プロトコルです。Cisco NX-OS は、クライアント側のインターフェイスである NETCONF エージェントを提供しており、XML でエンコードされた YANG モデルの形式で、クライアントの要求とサーバーの応答のため、SSH 上のセキュアな転送を提供します。

NETCONF は、構成データストアと、これらのデータストアでの操作とクエリを可能にする一連の作成、読み取り、更新、および削除（CRUD）操作を定義しています。NX-OS では、実行、起動、候補の 3 つのデータストアがサポートされています。サポートされている操作の簡単な説明を次に示します。

表 *16* : サポートされる操作

動作	説明
get	実行構成と動作状態を取得します。
get-config	指定されたデータストアから構成を取得します。

動作	説明
edit-config	指定されたターゲットデータストアに、指定された構成をロードします
close-session	セッションの適切な終了を要求します。
kill-session	セッションを強制終了します。
copy-config	別のデータストアの内容でデータストアを作成するか、置き換えます。
ロック	データストアをロックします。
unlock	データストアのロックを解除します。
検証	指定された構成の内容を検証します。
commit	候補構成を新しい現行の実行構成としてコミットします。
cancel-commit	進行中の要確認コミットをキャンセルします。
discard-changes	候補構成を現行の実行構成に戻します。

NETCONF に関する注意事項と制限事項

NETCONF エージェントには、次の注意事項と制限事項があります。

- Cisco NX-OS は、NETCONF 通知で Cisco デバイス YANG モデルと OpenConfig モデルの両方をサポートします。
- デバイス YANG モデルはエフェメラルデータを定義します。これらは「// Ephemeral data」というコメントでマークされます。これらの非永続的な大容量データは、モデルの残りの部分とは異なる方法で処理されます。これらは、<get> クエリの <filter> パラメータが、コメントでマークされた特定の要素を具体的に指している場合にのみ返されます。使用方法の詳細については、エフェメラル データ サポートのドキュメントを参照してください。
- Cisco NX-OS リリース 9.3(3) 以降、NETCONF は RFC 6241 に準拠していますが、次の例外があります。
  - 兄弟コンテンツ マッチ ノードは、「AND」式ではなく「OR」式で論理的に結合されます。（セクション 6.2.5）
  - 候補データストアを編集した後は、同じプロパティの実行構成を編集しないでください。
- 1 つの Get 要求でサポートされるオブジェクトの数は 250,000 です。次のエラーが表示された場合は、要求されたデータが 250,000 を超えていることを意味します。このエラーを回避するには、データをさらに絞り込んでクエリするためのフィルタを使用して要求を送信します。

```
too many objects(459134 > 250000) to query the entire device model.
```

- NETCONF は、RFC 6536 で指定されている拡張ロールベース アクセス コントロール（RBAC）をサポートしていません。「network-admin」ロールを持つユーザーのみが NETCONF エージェントへのアクセスを許可されます。

- NX-OS 9.3(1) 以降、NETCONF クライアントからスイッチへの NETCONF `get` および `get-config` 要求には、明示的な名前空間とフィルタが含まれている必要があります。この要件は、OpenConfig YANG および NETCONF デバイス モデルへの要求に影響します。次のようなメッセージが表示された場合、要求は名前空間を伝送していません。

  ```
  Request without namespace and filter is an unsupported operation
  ```

  次の例は、この変更前の動作を使用した `get` 要求と応答を示しています。この例は、サポートされなくなった動作が原因で発生するエラーメッセージを示しています。

  要求:

  ```
  <get>
  </get>
  ```

  応答：

  ```
  <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>operation-not-supported</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">Request without filtering is an unsupported
  operation</error-message>
    </rpc-error>
  </rpc-reply>
  ```

  次に、NX-OS リリース 9.3(1) 以降での正しい動作の `get` 要求と応答の例を示します。

  要求:

  ```
  <get>
    <filter>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      </System>
    </filter>
  </get>
  ```

  応答：

  ```
  <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <data>
          <System> …
      </data>
  </rpc-reply>
  ```

- `<edit-config>` の「置換」操作は、影響を受けるシステム コンポーネントによって実装されている実行時デフォルト値と動作が原因で、機能しない場合があります。したがって、NX-API

開発者サンドボックスの代わりに、<get-config> クエリによって取得した構成上で、置換を行うための構成を基礎とする方が適切です。

- Cisco NX-OS NETCONF サーバは、最大 5 つのサブスクリプション（クライアント セッションごとに 1 つのサブスクリプション）をサポートします。
- RFC 5277 によれば、自律通知は、イベント ソースの NETCONF、SYSLOG、および SNMP ストリームをサポートします。このリリースでは、Cisco NX-OS は NETCONF ストリームのみをサポートします。
- Cisco NX-OS は、サブスクリプションの [再生（Replay）] オプションをサポートしていません。[開始時刻（Start Time）] オプションと [終了時刻（Stop Time）] オプションは再生の一部であるため、サポートされていません。
- ストリーム サブスクリプションとフィルタリングでは、サブツリー フィルタリングのみがサポートされます。XPath フィルタリングはサポートされていません。
- Cisco NX-OS NETCONF エージェントが高負荷で動作している場合、一部のイベント通知がドロップされる可能性があります。
- Cisco NX-OS は、Cisco NX-OS リリース 9.3(1) 以降で NETCONF 通知をサポートします。Cisco NX-OS は、Cisco デバイス YANG モデルのみをサポートします。
- Cisco NX-OS は、Cisco デバイス YANG モデルと OpenConfig モデルの両方をサポートします。NETCONF 通知で OpenConfig モデルをサポートするのは、Cisco NX-OS 9.3(5) リリース以降です。

NETCONF エージェントの構成

Cisco NX-OS 9.3(5) 以降の SSH を介した NETCONF エージェントの設定

この手順では、SSH を介して NETCONF エージェントを有効にして構成する方法について説明します。

（注） この手順は、Cisco NX-OS リリース 9.3(5) 以降で使用します。

始める前に

NETCONF を使用してスイッチと通信する前に、NETCONF エージェントを有効にする必要があります。NETCONF エージェントを有効または無効にするには、**[no] feature netconf** コマンドを入力します。

手順の概要

1. **configure terminal**

2. **feature netconf**
3. （任意） **netconf idle-timeout** *it-num*
4. （任意） **netconf sessions** *num-sessions*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**configure terminal** 例： `switch# configure terminal`	グローバル コンフィギュレーション モードを開始します。
Step 2	**feature netconf** 例： `switch(config)# feature netconf`	NETCONF サービスを有効にします。
Step 3	（任意） **netconf idle-timeout** *it-num* 例： `switch(config)# netconf idle-timeout 5`	（オプション）アイドル状態のクライアントセッションが切断されるまでのタイムアウトを分単位で指定します。*it-num* の範囲は 0 ～ 1440 分です。デフォルトのタイムアウトは 5 分です。値を 0 に設定するとタイムアウトが無効になります。
Step 4	（任意） **netconf sessions** *num-sessions* 例： `switch(config)# netconf sessions 5`	同時クライアントセッションの最大数を指定します。*num-sessions* の範囲は 1 ～ 10 です。デフォルト値は 5 セッションです。

Cisco NX-OS 9.3(4) 以前の NETCONF エージェントの構成

（注） Cisco NX-OS リリース9.3(4) 以前の場合は、次の手順に従ってください。

NETCONFエージェントは、構成ファイル（`/etc/mtx.conf`）の `[netconf]` セクションで、次のオプションの構成パラメータをサポートします。

パラメータ	説明
idle_timeout	（オプション）アイドル状態のクライアントセッションが切断されるまでのタイムアウトを分単位で指定します。 デフォルト値は 5 分です。 値を 0 に設定するとタイムアウトが無効になります。

パラメータ	説明
limit	（オプション）同時クライアントセッションの最大数を指定します。 デフォルト値は 5 セッションです。 指定できる範囲は、1 ～ 10 です。

次に、構成ファイルの [netconf] セクションの例を示します。

```
[netconf]
mtxadapter=/opt/mtx/lib/libmtxadapternetconf.1.0.1.so
idle_timeout=10
limit=1
```

変更した構成ファイルを有効にするには、CLI コマンド [**no**] **feature netconf** を使用して NETCONF エージェントを無効にしてから再度有効にして、再起動する必要があります。

NETCONF セッションの確立

NETCONF は、クライアントとサーバー間の永続的な接続を必要とする接続指向のプロトコルです。スイッチ上の NETCONF エージェントは、管理ポート IP アドレスのポート 830 でリッスンします。クライアントは、SSH を介して NETCONF サブシステムとの接続を確立できます。クライアントが NETCONF エージェントとのセッションを確立すると、サーバーは <hello> メッセージをクライアントに送信します。同様に、クライアントは <hello> メッセージをサーバーに送信します。<hello> メッセージは、接続が開くと同時に交換されます。各 <hello> メッセージには、送信側ピアのプロトコル バージョンと機能のリストが含まれています。これらのメッセージは、プロトコルの互換性と機能を判断するために使用されます。両方の NETCONF ピアは、相手の <hello> メッセージで、共通のプロトコル バージョンがアドバタイズされたかどうかを確認する必要があります。また、サーバーの <hello> メッセージには <session-id> を含める必要がありますが、クライアントの <hello> メッセージには含めることができません。

次に、**ssh** コマンドを使用したセッション確立の例を示します。最初の <hello> メッセージがサーバーから受信され、2 番目のメッセージがクライアントから送信されます。サーバーの <hello> メッセージには、プロトコル バージョンである「urn:ietf:params:netconf:base:1.1」と、Cisco NX-OS リリース 9.3(4) でサポートされている NETCONF の基本機能が含められます。また、サーバーの <hello> メッセージには、サポートされているデータ モデルが含められます。これらは、現在の Cisco NX-OS リリースでサポートされているモデルとは一致しない場合があります。

（注） サーバーの <hello> メッセージには <session-id> が含められますが、クライアントのメッセージには含められません。

```
client-host % ssh admin@172.19.193.166 -p 830 -s netconf
User Access Verification
Password:
```

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.0</capability>
        <capability>urn:ietf:params:netconf:base:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>
        <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>

<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=report-all</capability>


<capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2020-04-20&module=Cisco-NX-OS-device</capability>


<capability>http://openconfig.net/yang/acl?revision=2019-11-27&module=openconfig-acl&deviations=cisco-nx-openconfig-acl-deviations</capability>


<capability>http://openconfig.net/yang/bfd?revision=2019-10-25&module=openconfig-bfd&deviations=cisco-nx-openconfig-bfd-deviations</capability>

        ...
    </capabilities>
    <session-id>1286775422</session-id>
</hello>
]]>]]><hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
]]>]]>
```

RFC 6242（SSH 上の NETCONF プロトコルの使用）からわかるとおり、メッセージフレーミングが複雑であるため、**ssh** コマンドで NETCONF を使用することは便利ではなく、エラーが発生しやすくなります。上記の例では、説明のみを目的として **ssh** コマンドを使用しています。NETCONF 用にはさまざまなクライアントが作成されており、**ssh** コマンドを使用するよりも推奨されています。ncclient はそのような例の 1 つであり、「使用例」のセクションで使用方法を説明します。

NETCONF は、セッションを終了するために、`<close-session>` および `<kill-session>` という 2 つの操作をサポートしています。サーバーは、`<close-session>` 要求を受信すると、セッションに関連付けられているロックとリソースを解放し、クライアントとの接続を閉じて、セッションを適切な方法で終了します。`<close-session>` 要求と応答が成功した例を次に示します。

```
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>

<rpc-reply message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

`<kill-session>` 要求は別のセッションを強制的に終了するもので、要求メッセージ内で `<session-id>` を指定する必要があります。サーバーは、`<kill-session>` 要求を受信すると、現在の操作を終了し、指定されたセッション ID に関連したロックとリソースを解放し、接続を閉じます。`<kill-session>` 要求と応答が成功した例を次に示します。

```
<rpc message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>296324181</session-id>
  </kill-session>
</rpc>

<rpc-reply message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

<close-session> および <kill-session> 要求とは別に、クライアントが一定時間要求を送信しなかった場合、セッションは自動的に終了します。デフォルトは 5 分です。アイドル タイムアウトの構成については、「NETCONF エージェントの構成」を参照してください。

NETCONF の読み取りおよび書き込み構成

このセクションでは、データストアの操作とクエリのためにサポートされている基本プロトコル操作について説明します。クライアントは、NETCONF エージェントとのセッションを確立した後、これらの操作のための RPC メッセージを送信できます。ここでは基本的な使用方法について説明します。詳細については RFC 6242 を参照してください。

<get-config>

この操作により、指定したデータストアの構成データを取得します。サポートされるパラメータは <source> と <filter> です。<source> は、（現在アクティブな構成を保持している）<running/> のような、クエリ対象のデータストアを指定します。<filter> は、指定されたデータストアのうちどの部分を取得するかを指定します。

次に示すのは、<get-config> 要求と応答メッセージの例です。

- <System> サブツリー全体を取得します：

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <running/>
        </source>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
        </filter>
    </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        ...
        </System>
    </data>
</rpc-reply>
```

- 特定のリスト項目を取得します：

```
<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
```

```
        <source>
            <running/>
        </source>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list>
                                <name>default</name>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
    </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <bgp-items>
                <inst-items>
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            ...
                            <rtctrl-items>
                                <enforceFirstAs>enabled</enforceFirstAs>
                                <fibAccelerate>disabled</fibAccelerate>
                                <logNeighborChanges>enabled</logNeighborChanges>
                                <supprRt>enabled</supprRt>
                            </rtctrl-items>
                            <rtrId>1.2.3.4</rtrId>
                        </Dom-list>
                    </dom-items>
                </inst-items>
            </bgp-items>
        </System>
    </data>
</rpc-reply>
```

<edit-config>

この操作は、指定された構成をターゲットデータストアに書き込みます。<target>パラメータは、<running/>または<candidate/> など、編集するデータストアを指定します。候補データストアは、変更がコミットされるまで、実行中のデータストアに影響を与えることなく操作できます。詳細については、<commit> のセクションを参照してください。<config> パラメータは、ターゲットデータストアに書き込まれるモデル化されたデータを指定します。モデルは「xmlns」属性で指定されます。「operation」属性は、<config>サブツリーの任意の数の要素に含めることができます。要素の操作は、新しい「operation」属性によってオーバーライドされるまで、その子孫要素に継承されます。サポートされている操作は、「merge」、「replace」、「create」、「delete」、および「remove」です。「remove」操作は、設定データが存在しない場合にエラーが返されないという点で「delete」とは異なります。「operation」属性が指定されていない場合は、「merge」操作がデフォルトと見なされます。デフォルトの動作は、オプションの<default-operation>パラメータ（「merge」、「replace」、または「none」があります）によりオーバーライドできます。

次に示すのは、<edit-config> 要求と応答メッセージの例です。

- MTU が 9216 である「po5」という名前のポートチャネルを、実行構成の説明に基づいて作成します。

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="create">
                            <id>po5</id>
                            <mtu>9216</mtu>
                            <descr>port-channel 5</descr>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
    <ok/>
</rpc-reply>
```

- ポートチャネルのすべての構成を新しい構成で置き換えます。

```
<rpc message-id="104" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
               <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="replace">
                            <id>po5</id>
                            <mtu>1500</mtu>
                            <adminSt>down</adminSt>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="104">
    <ok/>
</rpc-reply>
```

- ポートチャネルを削除します:

```
<rpc message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
```

```
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
           <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="delete">
                            <id>po5</id>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="105">
    <ok/>
</rpc-reply>
```

<copy-config>

この操作は、ターゲットの構成データストアを、ソース構成データストア全体のコンテンツによって置き換えます。ソースデータストアとターゲットデータストアのパラメータはそれぞれ<source>と <target> です。

次に示すのは、<copy-config> 要求と応答メッセージの例です。

- 実行構成をスタートアップ構成にコピーします。

```
<rpc message-id="106" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <target>
            <startup/>
        </target>
        <source>
            <running/>
        </source>
    </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="106">
    <ok/>
</rpc-reply>
```

- 実行構成を候補構成にコピーします。

```
<rpc message-id="107" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <target>
            <candidate/>
        </target>
        <source>
            <running/>
        </source>
    </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="107">
    <ok/>
```

```
</rpc-reply>
```

<lock>

<lock>操作を使用すると、クライアントは設定データストアをロックし、他のクライアントがデータストアをロックまたは変更するのを防ぐことができます。クライアントが保持しているロックは、<unlock> 操作を実行するか、セッションが終了すると、解除されます。<target>パラメータは、ロックするデータストアを指定します。

次に示すのは、<lock> 要求と応答メッセージの例です。

- ロックの取得に成功した場合：

```
<rpc message-id="108" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
        <target>
            <running/>
        </target>
    </lock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="108">
    <ok/>
</rpc-reply>
```

- 別のセッションですでに使用されていたため、ロックの取得に失敗した場合：

```
<rpc message-id="109" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
        <target>
            <candidate/>
        </target>
    </lock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="109">
    <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>lock-denied</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">Lock failed, lock is already held</error-message>

        <error-info>
            <session-id>1553704357</session-id>
        </error-info>
    </rpc-error>
</rpc-reply>
```

<unlock>

<unlock>操作は、以前に<lock>操作によって取得した構成のロックを解除します。<lock>操作を発行したのと同じセッションでのみ、<unlock>操作を使用できます。<target>パラメータは、ロック解除するデータストアを指定します。

次に示すのは、<unlock> 要求と応答メッセージの例です。

- ロック解除

```
<rpc message-id="110" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <unlock>
        <target>
            <candidate/>
        </target>
    </unlock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="110">
    <ok/>
</rpc-reply>
```

<get>

<get>操作は、実行中の構成とデバイスの状態情報を取得します。サポートされているパラメータは<filter>です。<filter>パラメータは、実行構成の動作状態データのうち、どの部分を取得するかを指定します。

次に示すのは、<get>要求と応答メッセージの例です。

- リスト項目の実行構成と動作状態データを取得します。

```
<rpc message-id="111" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list>
                                <name>default</name>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
    </get>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="111">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <bgp-items>
                <inst-items>
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            <always>disabled</always>
                            <bestPathIntvl>300</bestPathIntvl>
                            <clusterId>120</clusterId>
                          <firstPeerUpTs>2020-04-20T16:19:03.784+00:00</firstPeerUpTs>

                            <holdIntvl>180</holdIntvl>
                            <id>1</id>
                            <kaIntvl>60</kaIntvl>
                            <mode>fabric</mode>
                            <numEstPeers>0</numEstPeers>
                            <numPeers>0</numPeers>
                            <numPeersPending>0</numPeersPending>
                            <operRtrId>1.2.3.4</operRtrId>
```

```
                                <operSt>up</operSt>
                                <pfxPeerTimeout>90</pfxPeerTimeout>
                                <pfxPeerWaitTime>90</pfxPeerWaitTime>
                                <reConnIntvl>60</reConnIntvl>
                                <rtrId>1.2.3.4</rtrId>
                                <vnid>0</vnid>
                                ...
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </data>
    </rpc-reply>
```

<validate>

この操作は、候補データストアの構成内容を検証します。これは、実行データストアにコミットする前に、候補データストアで行われた構成変更を検証するのに役立ちます。<source> パラメータは <candidate/> をサポートします。

次に示すのは、<validate> 要求と応答メッセージの例です。

- 候補データストアの内容を検証します：

```
<rpc message-id="112" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <validate>
        <source>
            <candidate/>
        </source>
    </validate>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="112">
    <ok/>
</rpc-reply>
```

<commit>

候補構成を実行構成にコミットします。パラメータを付けずに操作すると、最終的なものと見なされ、元に戻すことはできません。<commit> を <confirmed/> パラメータを付けて発行すると、要確認コミットであると見なされます。別の、<confirmed/> パラメータを付けない<commit>操作を続けた場合にのみ、コミットはファイナライズされます。このようにして、確認した上でコミットできます。要確認コミットでは、<confirm-timeout> と <persist> という 2 つのパラメータを使用できます。<confirm-timeout> は、要確認コミットが元に戻されるまでの秒数です。この時間が経過すると、実行構成は要確認コミットが発行される前の状態に復元されます。<confirm-timeout> が指定されていない場合、デフォルトのタイムアウトは 600 秒です。セッションが終了すると、要確認コミットは元に戻ります。<persist>パラメータを使用すると、セッションが終了しても要確認コミットが保持されます。<persist> パラメータの値は、任意のセッションからの要確認コミットを識別するために使用されます。そして後続の要確認コミットまたはこれから確認するコミットの <persist-id> の値として使用する必要があります。

次に示すのは、<commit> 要求と応答メッセージの例です。

- 候補データストアの内容をコミットします:

```
<rpc message-id="113" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="113">
    <ok/>
</rpc-reply>
```

- タイムアウトが経過した要確認コミット:

```
<rpc message-id="114" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
        <confirm-timeout>120</confirm-timeout>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="114">
    <ok/>
</rpc-reply>
```

- 永続的な要確認コミットを開始し、その後、永続的な要確認コミットを実際に確認します:

```
<rpc message-id="115" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
        <persist>ID1234</persist>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="115">
    <ok/>
</rpc-reply>

<!-- confirm the persistent confirmed-commit, from the same session or another session
 -->
<rpc message-id="116" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <persist-id>ID1234</persist-id>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="116">
    <ok/>
</rpc-reply>
```

<cancel-commit>

この操作は、進行中の要確認コミットをキャンセルします。別のセッションからの要確認コミットをキャンセルする必要がある場合、要確認コミットの <persist> パラメータで指定したのと同じ値を指定して、<persist-id> パラメータを使用する必要があります。

- 同じセッションの要確認コミットをキャンセルします:

```
<rpc message-id="117" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <cancel-commit/>
</rpc>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="117">
    <ok/>
</rpc-reply>
```

<discard-changes>

この操作は、実行構成の内容にリセットすることによって、候補構成に加えられたコミットされていない変更を破棄します。パラメータは必要ありません。

次に示すのは、<discard-changes> 要求と応答メッセージの例です。

- 候補データストアに加えられた変更を破棄します。

```
<rpc message-id="118" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <discard-changes/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="118">
    <ok/>
</rpc-reply>
```

NETCONF の実行

NETCONF でのモデル駆動型操作について

表 *17 : NETCONF* でのモデル駆動型操作について

オペレーション	NETCONF RPC	CLI
チェックポイント	checkpoint	checkpoint <name> checkpoint <file>
ロールバック	ロールバック	rollback running-config checkpoint <name> rollback running-config checkpoint <file>
インストールするもの	install_all_nxos install_add install_activate install_deactivate install_commit install_remove	install all nxos <image> install {add | activate | deactivate | commit | remove} <rpm>
暗号化証明書のインポート	import_ca_certificate	crypto ca import <trustpoint> pkcs12 <file> <passphrase>

オペレーション	NETCONF RPC	CLI
スイッチのリロードまたはモジュールのリロード	reload	reload [timer <seconds>] reload module <module number>
ファイルのコピー	copy	copy <source> <destination>

モデル駆動型操作の例

モデル駆動型操作の例

ファイル名オプションを使用したチェックポイントの作成：

```
RPC:
<rpc message-id="checkpoint-3" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <file>bootflash:my_checkpoint2</file>
  </checkpoint>
</rpc>
```

チェックポイント名、説明を使用したチェックポイントの作成：

```
RPC:
<rpc message-id="checkpoint-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <action>create</action>
      <name>my_checkpoint1</name>
      <description>test checkpoint one</description>
  </checkpoint>
</rpc>
```

チェックポイント名を使用したチェックポイントの削除：

```
RPC:
<rpc message-id="delatecheckpoint-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <action>delete</action>
      <name>my_checkpoint1</name>
  </checkpoint>
</rpc>
```

ロールバック：

（注） 次のオプションタグは、アトミック、最初のエラーで停止、ベストエフォートとして使用できます。

```
<rpc message-id="rollback-cfg-option1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rollback xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <name>my_checkpoint1</name>
    <option>atomic</option>
</rollback>
</rpc>
```

ファイル オプションを使用したロールバック

```
<rpc message-id="rollback-cfg1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><
<rollback xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
```

```
    <file>bootflash:my_checkpoint2</file>
</rollback>
</rpc>
```

ファイルのコピー

リモートサーバーからスイッチストレージに任意のファイルをコピーします（例：ブートフラッシュ）。

Kerry tftp の場合：プロトコルがファイル転送をサポートします。

```
<rpc message-id="copy-file-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <source>tftp://172.27.xxx.xxx//<file_location?/tls1-server.pfx</source>
    <destination>bootflash:</destination>
      <vrf>management</vrf>
</copy>
</rpc>
```

CA 証明書のインポート

前提条件：スイッチで my_truspoint がすでに作成されている必要があります。

```
<rpc message-id="import_ca_certificate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<import_ca_certificate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <trustpoint>my_trustpoint</trustpoint>
    <pkcs12>tls1-server.pfx</pkcs12>
    <passphrase>xxxxxx</passphrase>
</import_ca_certificate>
</rpc>
```

RPM パッケージ インストール EXEC RPC コマンド.

Install <add>

```
<rpc message-id="install-add-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_add xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
 <add>rpm_packagenamehere_from_bootflash</add>
</install_add>
</rpc>
```

Install <activate>

```
<rpc message-id="install-activate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_activate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <activate> rpm_packagenamehere_from_bootflash</activate>
</install_activate>
</rpc>
```

Install <deactivate>

```
<rpc message-id="install-deactivate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <install_deactivate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <deactivate>rpm_packagenamehere_from_bootflash </deactivate>
 </install_deactivate>
</rpc>
```

Install <remove>

```
<rpc message-id="rpc-install_remove-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_remove xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
<remove>rpm_packagenamehere_from_bootflash </remove>
</install_remove>
</rpc>
```

すべての nx-os イメージのインストール

```
<rpc message-id="rpc-install_all_nxos-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<install_all_nxos xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <nxos>nxos.image.bin.upg</nxos>
</install_all_nxos>
</rpc>
```

モジュール番号のリロード

```
<rpc message-id="reload-module-pyld1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <module>29</module>
</reload>
</rpc>
```

再読み込み（Reload）

（注） クライアントが次の RPC を要求または送信すると、exec コマンドはスイッチのリロードを実行します。それ以上、Netconf クライアントは <ok> 応答を受信しません。

```
<rpc message-id="563" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
</rpc>
```

NETCONF 通知

NETCONF 通知について

NETCONF 通知は、NETCONF クライアントがシステムイベントにサブスクライブして、NETCONF エージェントからこれらのイベントに関する通知を受信できるようにするメカニズムです。これらの機能は、RFC 5277 で定義されています。Cisco NX-OS リリース 9.3(1) 以降、RFC 5277 で説明されているように、NETCONF 通知のサポートが開始されました。これは、NETCONF hello メッセージでアドバタイズされるオプションの機能です。

NETCONF クライアントは、Deviceyang または OpenConfig モデルを使用して、通知にサブスクライブできます。NETCONF 通知での OpenConfig モデルのサポートは、Cisco NX-OS リリース 9.3(5) から開始されました。

このサポートにより、NETCONF クライアントは次のことが行えます。

- イベント通知へのサブスクライブ

  各サブスクリプションは、NETCONF クライアントからのセッションを介した 1 回限りの要求です。Cisco NX-OS NETCONF エージェントが応答し、NETCONF クライアントによってセッションが明示的に閉じられるまで、サブスクリプションはアクティブです。サブスクリプションは、スイッチの再起動やスイッチの NETCONF 機能の無効化などの管理アクションによって閉じられることもあります。サブスクリプションは、基盤となる NETCONF セッションがアクティブである限りアクティブです。これらの登録済みフィルタに対して生成されたイベントは、通知としてクライアントに送信されます。クライアントは、システム イベントの通知にサブスクライブできます。たとえば、一部だけ挙げてみても、ポート状態の変更、

ファン速度の変更、プロセスメモリの変更などがあります。また、有効になっている新機能などの構成イベントもあります。

- 障害イベント通知を受信します。

  イベント通知は、スイッチの設定イベントまたは動作イベントに関する情報を含む、形式の整ったXMLドキュメントです。NETCONFクライアントは、サブスクリプション要求でフィルタリング基準を送信して、すべてのイベントではなくイベントのサブセットを指定できます。

- 他の操作でイベント通知をインターリーブします。

  Cisco NX-OS NETCONF エージェントは、アクティブな通知サブスクリプションを持つセッションで、NETCONF 要求を受信し、処理し、応答できます。

機能交換

NETCONF ハンドシェイク中に、Cisco NX-OS NETCONF サーバーは<capabilities> 要素を接続している NETCONF クライアントに送信して、サーバーが処理できる要求を示します。交換の一部として、サーバーは次の識別子を含めます。これらの識別子は、Cisco NX-OS NETCONF サーバーが通知とインターリーブの両方をサポートしていることをクライアントに通知します。

通知の機能識別子：

```
urn:ietf:params:netconf:capability:notification:1.0
```

インターリーブの機能識別子：

```
urn:ietf:params:netconf:capability:interleave:1.0
```

イベント ストリームの検出

クライアントは、使用可能なすべての<streams>に対しNETCONFの<get>操作を使用して、Cisco NX-OS NETCONF サーバーがサポートしているストリームを検出できます。Cisco NX-OS は NETCONF ストリームのみをサポートします。イベント ストリームの検出は、要求と応答のシーケンスによって行われます。

使用可能なストリームを取得するための要求：

任意の NETCONF クライアントは、すべてのサポートされているストリームを識別するため、<streams> に対するフィルタとともに、NETCONF <get> 要求を送信できます。次の例は、クライアント要求メッセージのペイロードを示しています。

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get>
      <filter type="subtree">
        <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
           <streams/>
        </netconf>
      </filter>
```

```
    </get>
</rpc>
```

応答：

Cisco NX-OS NETCONF サーバーは、クライアントがサブスクライブできる、使用可能なすべてのイベント ストリームで応答します。Cisco NX-OS は NETCONF ストリームのみをサポートします。

```
<rpc-reply message-id="101"
                   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <data>
       <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
        <streams>
           <stream>
              <name>NETCONF</name>
              <description>default NETCONF event stream </description>
           </stream>
        </streams>
       </netconf>
     </data>
</rpc-reply>
```

サブスクリプションの作成

NETCONF クライアントは、<create-subscription> プロトコル動作により、RPC を通してスイッチ上でイベントのサブスクリプションを作成できます。Cisco NX-OS NETCONF サーバーが <ok/> 要素でのみ応答した場合、サブスクリプションはアクティブです。

同期式の Get および Set 操作とは異なり、サブスクリプションは永続的な非同期操作です。サブスクリプションは、クライアントが明示的にサブスクリプションを閉じるか、セッションがオフラインになるまでアクティブなままです。たとえば、スイッチが再起動すると、セッションはオフラインになります。

クライアントがイベント通知をサブスクライブしていたものの、オフラインになった場合には、サーバーはサブスクリプションを終了し、セッションを閉じます。

サブスクリプションが閉じられた場合、すべてのイベント通知を受信するには、NETCONF クライアントで再接続してサブスクリプションを再度作成する必要があります。

サーバーはサブスクリプションを開始しないため、ユーザーが <create-subscription> 操作を含むクライアント プログラムを書く必要があります。次は、NETCONF クライアントが送信する <create-subscription> の例です。

```
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>NETCONF</stream>
    <filter xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0" type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                    <PhysIf-list>
                        <id>eth1/54/1</id>
                        <phys-items>
                          <operSt/>
                        </phys-items>
```

```
                    </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
    </filter>
</create-subscription>
```

<create-subscription> 操作は、次のオプションのどれでもサポートします。

- <stream>：クライアントがサブスクライブするイベントのストリームを指定します。ストリームを指定しなかった場合、NETCONF ストリーム内のイベントがデフォルトでクライアントに送信されます。
- <filter>：イベントをフィルタリングして、ストリームでイベントのサブセットだけが伝送されるようにできます。

Cisco NX-OS NETCONF サーバーは、サブスクリプションを正常に作成できた場合、応答で <ok> メッセージを返します。

次に、クライアントが <create-subscription> 要求をサーバーに送信し、成功応答を受信した例を示します。

クライアントが受信した、<create-subscription> への応答：

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:6ff0bda6-d3f1-4288-9a7e-0f30581e4bab">
    <ok/>
</rpc-reply>
```

（注） リプレイを使用したサブスクリプションはサポートされていないため、[開始時間（Start Time）] および [終了時間（Stop Time）] オプションは使用できません。

受信通知

NETCONF クライアントがサブスクリプションを正常に作成すると、Cisco NX-OS NETCONF サーバは、スイッチ内のすべてのイベントについて、使用されたフィルタに関連するイベント通知の送信を開始します。イベント通知は、notification 要素を含む独自の XML フォーマットのドキュメントです。

次に、クライアントが DeviceYang モデルからインターフェイス operSt にサブスクライブしていて、イーサネットインターフェイスがダウンした場合の通知の例を示します。<create-subscription> は、[サブスクリプションの作成（Creating Subscriptions）] セクションにあります。

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2020-05-05T10:22:52.260+00:00</eventTime>
    <operation>modified</operation>
    <event>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
```

```
                <phys-items>
                    <PhysIf-list>
                        <id>eth1/54/1</id>
                        <phys-items>
                            <operSt>down</operSt>
                        </phys-items>
                    </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
    </event>
</notification>
```

<notification> メッセージには次のフィールドが含まれます。

- <eventTime> はイベントが発生した日時を示すタイムスタンプです。
- <operation> はモデル ノードのイベントのタイプです。
- <event> はクライアントがサブスクライブしているモデル データです。

サブスクリプションの終了

サブスクリプションは、NETCONF クライアントが NETCONF メッセージのペイロードで Cisco NX-OS NETCONF サーバーに特定の操作を送信すると終了します。サブスクリプションの終了は、次のいずれかの方法で発生します。

- サブスクリプション セッションを閉じる。<close-session> 操作は、特定のサブスクリプション セッションに関連して NETCONF サーバーに送信されます。
- NETCONF セッションを終了する。<kill-session> 操作が NETCONF サーバーに送信されます。

すべてのサブスクリプションは、1 つの NETCONF セッションに関連付けられます。これは 1 対 1 の関係です。

NETCONF の例

（注） このセクションのすべての例では、ncclient python ライブラリを使用します。

ncclient を使用した Cisco NX-OS の接続

ncclient は、NETCONF クライアント用の Python ライブラリです。次に、ncclient Manager API から Cisco NX-OS への接続を確立する方法の例を示します。

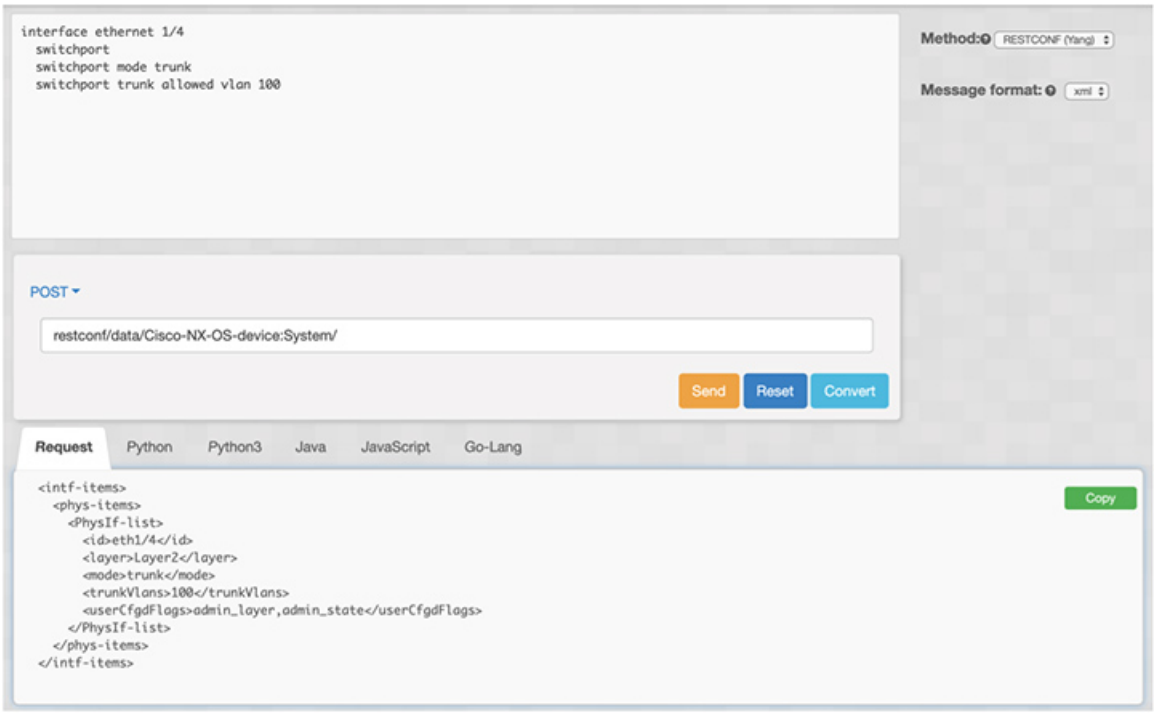
```
device = {
    "address": "10.10.10.10",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
```

```
}
with manager.connect(host = device["address"],
                         port = device["netconf_port"],
                         username = device["username"],
                         password = device["password"],
                         hostkey_verify = False) as m:
    # do your stuff
```

サンドボックスを使用した NETCONF ペイロードの生成

有効にするには、「NXAPI 開発者サンドボックス」のセクションを参照してください。NETCONF のペイロードを生成するには、メソッドを RESTCONF（Yang）に変更し、メッセージ フォーマットを XML に変更します。変換する必要があるコマンドをテキスト ウィンドウに入力し、**[変換（Convert）]** をクリックすると、同等のペイロードが [要求（Request）] テキストボックスに表示されます。

図 *3 : NCCLIENT*



Cisco NX-OS からの構成データの取得

次に、ncclient を使用して Cisco NX-OS から BGP 設定を取得する方法の例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco!"
```

```
}


# create a main() method
def main():
    bgp_dom = """
    <filter type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list/>
                        </dom-items>
                    </inst-items>
                </bgp-items>
        </System>
    </filter>
    """

    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # Collect the NETCONF response
        netconf_response = m.get_config(source='running', filter=bgp_dom)
        # Parse the XML and print the data
        xml_data = netconf_response.data_ele
        print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))


if __name__ == '__main__':
    sys.exit(main())
```

Cisco NX-OS からの実行構成および運用データの取得

次に、Cisco NX-OS 上のすべての物理インターフェイスのインターフェイス カウンタを取得する例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():

    intf_ctr_filter = """
    <filter>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                <PhysIf-list>
                    <dbgIfIn-items/>
```

```
                    <dbgIfOut-items/>
                </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
    </filter>"""

    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # Collect the NETCONF response
        netconf_response = m.get(filter=intf_ctr_filter)
        # Parse the XML and print the data
        xml_data = netconf_response.data_ele
        print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF を使用した新しい構成の作成

次に、ncclient の edit config を使用して、名前付きの VLAN 100 を作成する方法の例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():
    add_vlan = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bd-items>
          <bd-items>
            <BD-list>
              <fabEncap>vlan-100</fabEncap>
              <name>inb_mgmt</name>
            </BD-list>
          </bd-items>
        </bd-items>
      </System>
    </config>
    """


    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=add_vlan)
```

```
        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF を使用した構成の削除

次に、Cisco NX-OS からループバック インターフェイスを削除する例を示します。

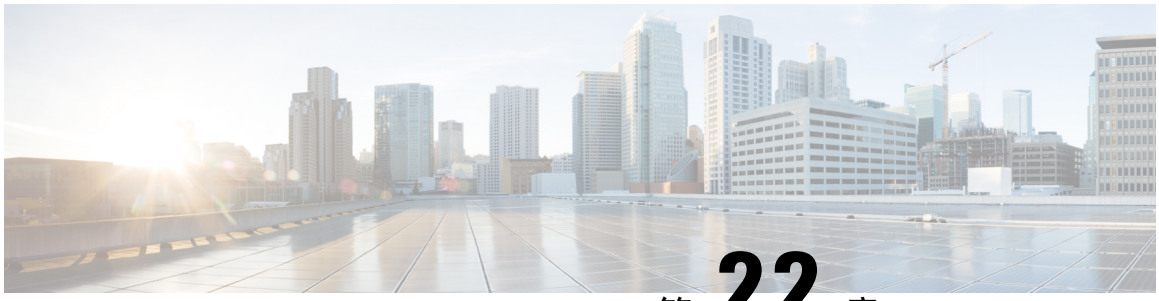
```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():
    remove_loopback = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <lb-items>
            <LbRtdIf-list operation="delete">
              <id>lo10</id>
            </LbRtdIf-list>
          </lb-items>
        </intf-items>
      </System>
    </config>"""


    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=remove_loopback)

        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF エージェントのトラブルシューティング

接続のトラブルシューティング

- クライアント システムから、スイッチの管理ポートに ping を実行して、スイッチが到達可能であることを確認します。

- Cisco NX-OS で、**show feature | inc netconf** コマンドを入力してエージェントのステータスを確認します。
- XML 管理インターフェイス（xmlagent とも呼ばれる）というものがあります。これは NETCONF エージェントとはまったく異なりますが、よく混同されます。サーバーが正しい NETCONF メッセージで応答しない場合は、正しいポート 830 に接続していて、サーバーから正しい <hello> メッセージ（「NETCONF セッションの確立」セクションに示されているものと同様）を受信していることを確認します。



第 22 章

RESTCONF エージェント

RESTCONF エージェントについて

Cisco NX-OS RESTCONF は、NETCONF で定義されたデータストアを使用して、YANG バージョン 1 で定義されたデータを構成するための HTTP ベースのプロトコルです。

NETCONF は、構成データストアと、これらのデータストアへのアクセスに使用できる一連の作成、取得、更新、および削除（CRUD）操作を定義します。YANG 言語は、データストア格納ファイル、運用データ、プロトコル操作、イベント通知の構文とセマンティクスを定義します。

Cisco NX-OS RESTCONF は、HTTP 操作を使用して、YANG 定義のデータを含む概念データストアでの CRUD 操作を可能にします。このデータは、NETCONF データストアを実装するサーバーと互換性があります。

RESTCONF プロトコルは、XML と JSON の両方のペイロード エンコーディングをサポートします。ユーザー認証は、HTTP 基本認証によって行われます。

次の表に、Cisco NX-OS RESTCONF エージェントがサポートするプロトコル操作を示します。

RESTCONF	同等の NETCONF
オプション	NETCONF：なし
HEAD	NETCONF：なし
GET	NETCONF: <get-config>, <get>
POST	NETCONF: <edit-config> (operation="create")
PUT	NETCONF: <edit-config> (operation="create/replace")

RESTCONF	同等の NETCONF
PATCH	NETCONF: <edit-config> (operation="merge")
DELETE	NETCONF: <edit-config> (operation="delete")

注意事項と制約事項

RESTCONF エージェントには、次の注意事項と制約事項があります。

- Cisco NX-OS RESTCONF は、RESTCONF Protocoldraft-ietf-netconf-restconf-10 と題する RFC ドラフトに基づいています。https://tools.ietf.org/html/draft-ietf-netconf-restconf-10 を参照してください。
- RESTCONF は、RFC 6536 で指定されている拡張ロールベース アクセス コントロール（RBAC）をサポートしていません。「network-admin」ロールを持つユーザのみに、RESTCONF エージェントへのアクセス権が付与されます。

RESTCONF エージェントの使用方法

一般的なコマンド

- HTTP または HTTPS アクセスを有効にするには、次のコマンドを構成します。
  - **feature nxapi**
  - **nxapi http port 80**
  - **nxapi https port 443**

一般的な制御コマンド

RESTCONF エージェントの [**no**] **feature restconf** コマンドを有効または無効にできます。

エージェント ステータスの表示

RESTCONF エージェントのステータスを表示するには、**show feature** コマンドを使用し、式 restconf を含めます。

```
switch-1# show feature | grep restconf
restconf              1          enabled
switch-1#
```

Curl を使用したサーバーへの POST 要求の送信

```
client-host % curl -X POST -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Content-Type:
application/yang.data+xml" -d '<always>enabled</always><rtrId>2.2.2.2</rtrId>'
```

```
"http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list=default"
 -i

HTTP/1.1 201 Created
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:25:31 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500853169574134
Status: 201 Created
Location: /System/bgp-items/inst-items/dom-items/Dom-list=default/always/rtrId/
```

Curl を使用したサーバーへの GET 要求の送信

```
client-host % curl -X GET -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Accept:
application/yang.data+xml"
"http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list?content=config"
 -i

HTTP/1.1 200 OK
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:26:03 GMT
Content-Type: application/yang.data+xml
Content-Length: 395
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500856185650327
Status: 200 OK

    <Dom-list>
        <name>default</name>
        <always>enabled</always>
        <bestPathIntvl>300</bestPathIntvl>
        <holdIntvl>180</holdIntvl>
        <kaIntvl>60</kaIntvl>
        <maxAsLimit>0</maxAsLimit>
        <pfxPeerTimeout>30</pfxPeerTimeout>
        <pfxPeerWaitTime>90</pfxPeerWaitTime>
        <reConnIntvl>60</reConnIntvl>
        <rtrId>2.2.2.2</rtrId>
    </Dom-list>
client-host %
```

RESTCONF エージェントのトラブルシューティング

接続のトラブルシューティング

- **feature nxapi** コマンドを発行して、Web サーバーを有効にします。
- HTTP 用にポートを開くように **nxapi http port 80** コマンドが構成されていることを確認します。
- HTTPS 用にポートを開くように **nxapi https port 443** コマンドが構成されていることを確認します。
- スイッチの管理ポートに ping を実行して、スイッチが到達可能であることを確認します。

トラブルシューティングのエラー

次に、一般的なエラーメッセージと、それを解決するためのガイドラインを示します。

エラーメッセージ：申し訳ありませんが、検索しているページは現在使用できません（Sorry, the page you are looking for is currently unavailable）

- 要求を送信した直後（数秒後など）にこのメッセージを受信した場合は、次の点を確認してください：
  - 「接続のトラブルシューティング」に記載されているように、NXAPI 機能が有効になっていること。
  - RESTCONF 機能が有効になっていること（**show feature | grep restconf**）。RESTCONF が有効になっていない場合は、有効にします（**feature restconf**）。
  - ポートは、NX-API によって HTTP または HTTPS 用に構成されます。**show nxapi** を使用して、ポートが構成されていることを確認します。

    ```
    switch-1# show nxapi
    nxapi enabled
    HTTP Listen on port 80
    HTTPS Listen on port 443
    ...
    switch-1#
    ```

    ポートが HTTP または HTTPs 用に構成されていない場合は、**nxapi http port 80** または **nxapi https port 443** を発行して構成します。

- 要求の送信後にこのメッセージを受信した場合（数分後など）、スイッチの最上位レベルからの過剰な同時クエリー要求でシステムが過負荷になっていないことを確認します。過剰な最上位レベルのクエリは、かなりのリソース負荷を生じる可能性があります。

  次のいずれかによって、スイッチが過負荷になっていないことを確認できます。
  - クライアントが送信する要求の数をスロットルバックします。
  - スイッチで **no feature restconf** そして **feature restconf**を発行して RESTCONF エージェントを再起動します。

エフェメラル データ

RESTCONF のエフェメラル データについて

この機能は、エフェメラル データへのアクセスを提供します。エフェメラル データは、大量のデータです。DME は、各バッチがメモリ使用量の観点から管理可能なサイズになるように、データを取得するバッチ メカニズムを提供します。バッチのサイズは、取得する MO の数です。

パブリッシュされた Cisco-NX-OS-device.yang ファイルの「Ephemeral data」というコメントで、どのデータがエフェメラルであるかに関する情報を確認できます。

リクエスト内の URI が以下を指している場合にのみ、エフェメラル データからの出力が返されます。

- エフェメラル データのリーフ
- エフェメラル データの子を持つコンテナまたはリスト
- 直接のエフェメラル データの子を持つリストをラップするために使用される空のコンテナ

システム レベルの GET クエリは、エフェメラル データを返しません。

RESTCONF のエフェメラル データの例

これは、エフェメラル データを取得する例です。

クライアントは、次の GET 要求メッセージを送信する場合があります。

```
GET
/restconf/data/Cisco-NX-OS-device:System//urib-items/table4-items/Table4-list=management/route4-items
 HTTP/1.1
   Host: example.com
   Accept: application/yang.data+json

   The server might respond:

      HTTP/1.1 200 OK
      Date: Fri, 06 Mar 2020 11:10:30 GMT
      Server: nginx/1.7.10
      Content-Type: application/yang.data+json

      {
          "route4-items": {
               "Route4-list": [{
                   "prefix": "172.23.167.255/32",
                   "flags": "0",
                   ...
```

実行操作

RESTCONF の操作コマンドについて

この機能は、スイッチでモデル駆動型操作コマンドを実行する方法を提供します。

次に、サポートされている実行 RPC のリストを示します。RPC に関する情報は、公開されている Cisco-NX-OS-device.yang ファイルで確認できます。

表 *18 : RESTCONF* でのモデル駆動型操作について

オペレーション	RESTCONF RPC	CLI
チェックポイント	checkpoint	checkpoint <name> checkpoint <file>

オペレーション	RESTCONF RPC	CLI
ロールバック	ロールバック	rollback running-config checkpoint <name> rollback running-config checkpoint <file>
インストールするもの	install_all_nxos install_add install_activate install_deactivate install_commit install_remove	install all nxos <image> install {add | activate | deactivate | commit | remove} <rpm>
暗号化証明書のインポート	import_ca_certificate	crypto ca import <trustpoint> pkcs12 <file> <passphrase>
スイッチのリロードまたはモジュールのリロード	reload	reload [timer <seconds>] reload module <module number>
ファイルのコピー	copy	copy <source> <destination>

RESTCONF 操作コマンドの例

チェックポイント RPC の例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:checkpoint
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
                    "name": "checkpoint-1",
                    "description": "testing checkpoint through Restconf"
                }
          }

The server might respond:
      HTTP/1.1 204 No content
```

ロールバック RPC の例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:rollback
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
                    "name": "checkpoint-1",
                    "action": "create”
                }
```

```
          }

The server might respond:
      HTTP/1.1 204 No content
```

インストール RPC の例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:install_all_nxos
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
                    "nxos": "bootflash:nxos.10.1.1-jcco.bin"
                }
          }

The server might respond:
      HTTP/1.1 204 No content
```

CA 証明書 RPC のインポートの例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
The client might send the following POST request message:
POST /restconf/operations/Cisco-NX-OS-device:import_ca_certificate
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
                    "trustpoint": "mytrustpoint",
                    "pkcs12": "bootflash:server.pfx",
                    "passphrase": "mypassphrase"
                }
          }

The server might respond:
      HTTP/1.1 204 No content
```

スイッチリロード RPC の例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:reload
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
                }
          }

The server might respond:
      HTTP/1.1 204 No content
```

モジュールリロード RPC の例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

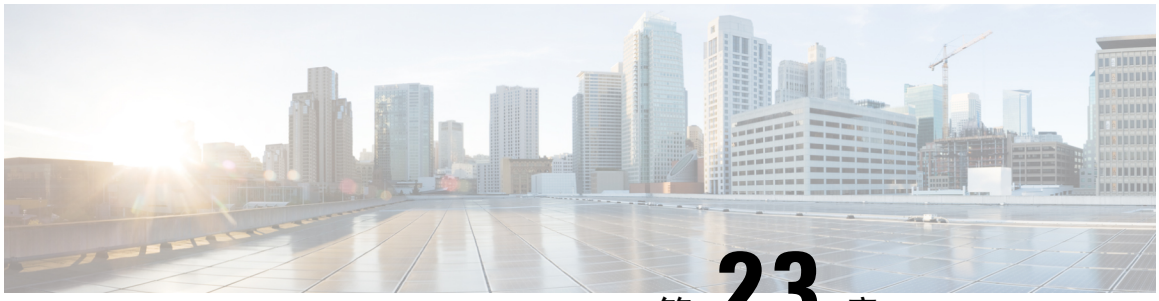
```
The client might send the following POST request message:
POST /restconf/operations/Cisco-NX-OS-device:reload
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
```

```
                    "module": "31"
                }
            }

The server might respond:
      HTTP/1.1 204 No content
```

コピー ファイル RPC の例

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:reload
    Accept: application/yang.operation+json,application/yang.errors+json
    Content-type: application/yang.operation+json
    Body: {
                "input": {
                    "source": "tftp://10.1.1.1/users/myname/config1.txt",
                    "destination": "bootflash:",
                    "vrf": "management"
                }
            }

The server might respond:
      HTTP/1.1 204 No content
```



第 23 章

ダイナミック ロガー

前提条件

ダイナミック ロギングを使用する前に、スイッチに次のものが存在することを確認します。

- libmtxlogmgr*.so ライブラリが /opt/mtx/lib/ にインストールされていること。libmtxlogmgr*.so ライブラリは、mtx_infra RPM の一部です。
- mtx.conf ファイルが /etc/ にあること。

```
[mtxlogger]
config=/opt/mtx/conf/mtxlogger.cfg
```

- mtxlogger.cfg ファイルが /opt/mtx/conf/ にあること。

参照先

構成ファイルの構造は次のとおりです。

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false"></leaf>
      <leaf name="allActive" type="boolean" default="false"></leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$"></leaf>
     <container name="componentID">
            <leaf name="enabled" type="boolean" default="true"></leaf>
     </container>
     <container name="dateTime">
            <leaf name="enabled" type="boolean" default="true"></leaf>
            <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
     </container>
     <container name="fcn">
            <leaf name="enabled" type="boolean" default="true"></leaf>
```

```
            <leaf name="format" type="string"
default="$CLASS$::$FCNNAME$($ARGS$)@$LINE$"></leaf>
     </container>
      </container>
      <container name="dest">
        <container name="console">
          <leaf name="enabled" type="boolean" default="false"></leaf>
        </container>
        <container name="file">
          <leaf name="enabled" type="boolean" default="false"></leaf>
          <leaf name="name" type="string" default="mtx-internal.log"></leaf>
          <leaf name="location" type="string" default="./mtxlogs"></leaf>
    <leaf name="mbytes-rollover" type="uint32" default="10"></leaf>
    <leaf name="hours-rollover" type="uint32" default="24"></leaf>
    <leaf name="startup-rollover" type="boolean" default="false"></leaf>
          <leaf name="max-rollover-files" type="uint32" default="10"></leaf>
        </container>
      </container>
      <list name="logitems" key="id">
        <listitem>
         <leaf name="id" type="string"></leaf>
   <leaf name="active" type="boolean" default="true"></leaf>
        </listitem>
      </list>
    </container>
  </container>
</config>
```

<list> タグは、**<componentID>** によるログ フィルタを定義します。

次の表では、コンテナとそのリーフの一部について説明します。

表 *19:* コンテナとリーフの説明

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
logging	すべてのロギング データ タイプが含まれます。	形式 dest ファイル （注） リストタグ「logitems」も含まれます	enabled：ロギングがオンかオフかを決定するブール値です。デフォルトはオフです。 allActive：ロギング用に定義されたすべてのロギング項目をアクティブにするブール値です。デフォルトはオフです。

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
形式	ログ メッセージの形式情報を格納します。	componentID dateTime type fcn	content：ログ メッセージに含まれるデータ型をリストする文字列です。内容： • $DATETIME$：ログ メッセージに日付または時刻を含めます。 • $COMPONENTID$：ログ メッセージにコンポーネント名を含めます。 • $TYPE$：メッセージタイプ（""、INFO、WARNING、ERROR）を含めます。 • $SRCFILE$：ソース ファイルの名前を含めます。 • $SRCLINE$：ソース ファイルの行番号を含めます。 • $FCNINFO$ ソース ファイルからの class::関数名を含めます。 • MSG：実際のログ メッセージテキストを含めます。

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
componentID	ログが記録されたコンポーネントの名前です。	該当なし	enabled：ログメッセージにコンポーネントIDを含めるかどうかを決定するブール値です。デフォルトは「true」です。値を「false」にすると、ログメッセージに "" 文字列を返します。
dateTime	ログメッセージの日付または時刻です	該当なし	enabled：ログメッセージに日付または時刻の情報を含めるかどうかのブール値です。デフォルトはイネーブルです。 format：ログメッセージに含める値を示す文字列です。フォーマットは%y%m%d.%H%M%Sです。
dest	接続先ロガーの構成設定を保持します。	console：接続先コンソールです。許可されるのは 1 つだけです。 file：接続先ファイルです。複数ファイルが許可されます。	該当なし
コンソール	接続先コンソール	該当なし	enabled：コンソールのロギングが有効かどうかを決定するブール値です。デフォルトは「false」です。

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
ファイル	接続先ファイルの設定を決定します。	該当なし	

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
			enabled：接続先が有効にされているかどうかを決定するブール値です。デフォルトは「false」です。 name：接続先ログファイルを示す文字列です。デフォルトは「mtx-internal.log」です。 location：接続先ファイル パスを示す文字列です。デフォルトは「./mtxlogs」です。 mbytes-rollover：システムが最も古いデータを上書きするまでのログファイルの長さを決定する uint32 値です。デフォルトは10Mバイトです。 hours-rollover：時間単位でのログファイルの長さを決定する uint32 値です。デフォルトは24 時間です。 startup-rollover：エージェントの起動時または再起動時にログファイルをロールオーバーするかどうかを決定するブール値です。デフォルト値は「false」です。 max-rollover-files：ロールオーバー ファイルの最大数を決定する uint32 値です。max-rollover-files の値を超えると、最も古いファイルが削除されま

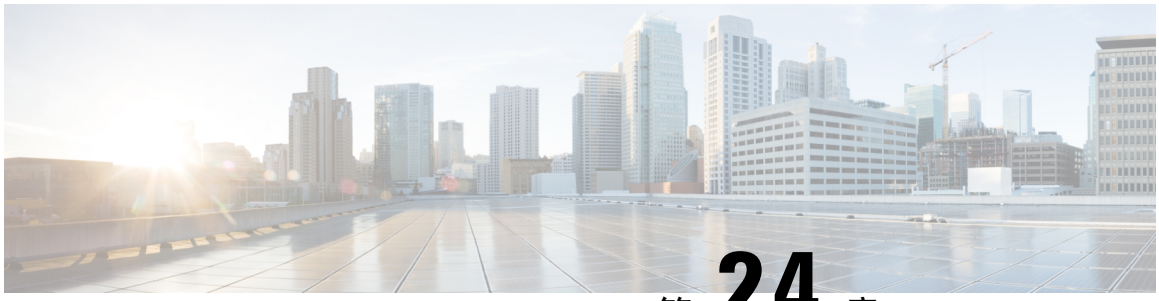
コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
			す。デフォルト値は 10 です。

例

次に、デフォルトでインストールされている設定を含む設定ファイルを示します。

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="allActive" type="boolean" default="false">false</leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$">$DATETIME$ $COMPONENTID$ $TYPE$ $SRCFILE$ @ $SRCLINE$ $FCNINFO$:$MSG$</leaf>
     <container name="componentID">
            <leaf name="enabled" type="boolean" default="true"></leaf>
     </container>
     <container name="dateTime">
            <leaf name="enabled" type="boolean" default="true"></leaf>
            <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
     </container>
     <container name="fcn">
            <leaf name="enabled" type="boolean" default="true"></leaf>
            <leaf name="format" type="string"
default="$CLASS$::$FCNNAME$($ARGS$)@$LINE$"></leaf>
     </container>
      </container>
      <container name="dest">
        <container name="console">
          <leaf name="enabled" type="boolean" default="false">true</leaf>
        </container>
        <container name="file">
          <leaf name="enabled" type="boolean" default="false">true</leaf>
          <leaf name="name" type="string" default="mtx-internal.log"></leaf>
          <leaf name="location" type="string" default="./mtxlogs">/volatile</leaf>
    <leaf name="mbytes-rollover" type="uint32" default="10">50</leaf>
    <leaf name="hours-rollover" type="uint32" default="24">24</leaf>
    <leaf name="startup-rollover" type="boolean" default="false">true</leaf>
          <leaf name="max-rollover-files" type="uint32" default="10">10</leaf>
        </container>
      </container>
      <list name="logitems" key="id">
        <listitem>
         <leaf name="id" type="string">*</leaf>
   <leaf name="active" type="boolean" default="false">false</leaf>
        </listitem>
        <listitem>
         <leaf name="id" type="string">SYSTEM</leaf>
   <leaf name="active" type="boolean" default="true">true</leaf>
        </listitem>
        <listitem>
         <leaf name="id" type="string">LIBUTILS</leaf>
   <leaf name="active" type="boolean" default="true">true</leaf>
        </listitem>
        <listitem>
         <leaf name="id" type="string">MTX-API</leaf>
   <leaf name="active" type="boolean" default="true">true</leaf>
        </listitem>
        <listitem>
```

```
          <leaf name="id" type="string">Model-*</leaf>
      <leaf name="active" type="boolean" default="true">true</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">Model-Cisco-NX-OS-device</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">Model-openconfig-bgp</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-MTX-API</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
           <listitem>
            <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
      <leaf name="active" type="boolean" default="true">false</leaf>
           </listitem>
         </list>
       </container>
     </container>
   </config>
```



第 24 章

gNMI-gRPC ネットワーク管理インターフェイス

gNMI について

gNMI は、トランスポート プロトコルとして gRPC（Google リモート プロシージャ コール）を使用します。

Cisco NX-OS は、Cisco Nexus 9000 シリーズ スイッチで実行されるテレメトリ アプリケーションへのダイヤルイン サブスクリプション用に gNMI をサポートします。過去のリリースでは gRPC を介したテレメトリ イベントがサポートされていましたが、スイッチはテレメトリ データをテレメトリ レシーバにプッシュしていました。この方法はダイヤル アウトと呼ばれていました。

gNMI を使用すると、アプリケーションはスイッチから情報をプルできます。サポートされているテレメトリ機能を学習し、必要なテレメトリ サービスのみをサブスクライブすることで、特定のテレメトリ サービスにサブスクライブします。

表 *20 :* サポートされる *gNMI RPC*

gNMI RPC	サポート対象
機能	はい
結果	はい
設定	はい
登録	はい

gNMI サブスクライブ RPC

Cisco NX-OS 9.3(1) リリース以降では、次の gNMI サブスクリプション機能がサポートされています。

表 *21 :* サブスクライブ オプション

タイプ	サブタイプ	サポートの有無	説明
[1 回（Once）]		はい	スイッチは、指定されたすべてのパスに対して現在の値を 1 回だけ送信します。
ポーリング（Poll）		はい	スイッチは、ポーリング メッセージを受信するたびに、指定されたすべてのパスの現在の値を送信します。

タイプ	サブタイプ	サポートの有無	説明
ストリーム	サンプル	はい	ストリーム サンプル間隔ごとに 1 回、スイッチは指定されたすべてのパスの現在の値を送信します。サポートされるサンプル間隔の範囲は 1 ～ 604800 秒です。 デフォルトのサンプル間隔は 10 秒です。
	[変更時（On_Change）]	はい	スイッチは初期状態として現在の値を送信しますが、指定されたパスのいずれかに作成、変更、または削除などの変更が発生した場合にのみ値を更新します。
	[ターゲット定義（Target_Defined）]	いいえ	

オプションの SUBSCRIBE フラグ

SUBSCRIBE オプションでは、表にリストされているオプションへの応答を変更するオプションのフラグを使用できます。Cisco NX-OS リリース 9.3(1) 以降では、[更新のみ（updates_only）] オプション フラグがサポートされています。これは、ON_CHANGE サブスクリプションに適用されます。このフラグが設定されている場合、スイッチは通常最初の応答で送信される初期スナップショット データ（現在の状態）を抑制します。

次のフラグはサポートされていません。

- [エイリアス（aliases）]
- [集約許可（allow_aggregation）]
- [拡張（extensions）]
- prefix
- [qos]

サブスクライブ フラグのサポート メトリクスは次のとおりです。

表 *22 : SUBSCRIBE* フラグのサポート メトリクス

サブスクリプションタイプ	[ハートビート間隔（**heartbeat_interval**）]	[冗長抑制（**suppress_redundant**）]
[変更時（On_Change）]	発信元：デバイス YANG、OpenConfig YANG、DME	該当なし
サンプル	発信元：デバイス YANG、OpenConfig YANG、DME	発信元：デバイス YANG、OpenConfig YANG

Guidelines and Limitations for gNMI

Following are the guidelines and limitations for gNMI:

- Beginning with Cisco NX-OS Release 9.3(5), Get and Set are supported.
- gNMI queries do not support wildcards in paths.
- When you enable gRPC on both the management VRF and default VRF and later disable on the default VRF, the gNMI notifications on the management VRF stop working.

  As a workaround, disable gRPC completely by entering the **no feature grpc** command and reprovision it by entering the **feature grpc** command and any existing gRPC configuration commands. For example, **grpc certificate** or **grpc port**. You must also resubscribe to any existing notifications on the management VRF.
- When you attempt to subscribe an OpenConfig routing policy with a preexisting CLI configuration like the following, it returns empty values due to the 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

  using the xpath

  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
  ```

- Only server certificate authentication takes place. The client certificate is not authenticated by the server.
- If the gRPC certificate is explicitly configured, after a reload with the saved startup configuration to a prior Cisco NX-OS 9.3(x) image, the gRPC feature does not accept connections. To confirm this issue, enter the **show grpc gnmi service statistics** command and the status line displays an error like the following:

  ```
  Status: Not running - Initializing...Port not available or certificate invalid.
  ```

  Unconfigure and configure the proper certificate command to restore the service.
- Beginning with Cisco NX-OS Release 9.3(3), if you have configured a custom gRPC certificate, upon entering the **reload ascii** command the configuration is lost. It reverts to the default day-1 certificate. After entering the **reload ascii** command, the switch reloads. Once the switch is up again, you must reconfigure the gRPC custom certificate.

Note This applies when entering the **grpc certificate** command.

- Use of origin, use_models, or both, is optional for gNMI subscriptions.
- gNMI Subscription supports Cisco DME and Device YANG data models. Beginning with Cisco NX-OS Release 9.3(3), Subscribe supports the OpenConfig model.
- For Cisco NX-OS prior to 9.3(x), information about supported platforms, see *Platform Support for Programmability Features* in the guide for that release. Starting with Cisco NX-OS release 9.3(x), for information about supported platforms, see the Nexus Switch Platform Matrix.
- The feature supports JSON and gnmi.proto encoding. The feature does not support protobuf.any encoding.
- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the 12 MB maximum, the collected data is dropped. Applies to gNMI ON_CHANGE mode only.

  You can avoid this situation by creating more focused subscriptions that handle smaller, more granular data-collection sets. So, instead of subscribing to one higher-level path, create multiple subscriptions for different, lower-level parts of the path.
- Across all subscriptions, there is support of up to 150K aggregate MOs. Subscribing to more MOs can lead to collection data drops.
- The feature does not support a path prefix in the Subscription request, but the Subscription can contain an empty prefix field.
- The gRPC process that supports gNMI uses the HIGH_PRIO control group, which limits the CPU usage to 75% of CPU and memory to 1.5 GB.
- The **show grpc gnmi** command has the following considerations:
  - The gRPC agent retains gNMI calls for a maximum of one hour after the call has ended.
  - If the total number of calls exceeds 2000, the gRPC agent purges ended calls based on the internal cleanup routine.

The gRPC server runs in the management VRF. As a result, the gRPC process communicates only in this VRF forcing the management interface to support all gRPC calls.

gRPC functionality now includes the default VRF for a total of two gRPC servers on each switch. You can run one gRPC server in each VRF, or run only one gRPC server in the management VRF. Supporting a gRPC in the default VRF adds flexibility to offload processing gRPC calls from the management VRF, where significant traffic load is not desirable.

If two gRPC servers are configured, be aware of the following:

- VRF boundaries are strictly enforced, so each gRPC server process requests independent of the other. Requests do not cross between VRFs.
- The two servers are not HA or fault tolerant. One gRPC server does not back up the other, and there is no switchover or switchback between them.
- Any limits for the gRPC server are per VRF.

Configuring gNMI

Configure the gNMI feature through the **grpc gnmi** commands.

To import certificates used by the **grpc certificate** command onto the switch, see the Installing Identity Certificates section of the Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x).

Note When modifying the installed identity certificates or **grpc port** and **grpc certificate** values, the gRPC server might restart to apply the changes. When the gRPC server restarts, any active subscription is dropped and you must resubscribe.

SUMMARY STEPS

1. **configure terminal**
2. **feature grpc**
3. (Optional) **grpc port** *port-id*
4. **grpc certificate** *certificate-id*
5. **grpc gnmi max-concurrent-call** *number*
6. (Optional) **grpc use-vrf default**

DETAILED STEPS

	Command or Action	Purpose

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

Step 2	**feature grpc**
	Example:
	`switch-1# feature grpc`
	`switch-1(config)#`
		Enables the gRPC agent, which supports the gNMI interface for dial-in.

Step 3	(Optional) **grpc port** *port-id* **Example:** `switch-1(config)# grpc port 50051`	Configure the port number. The range of *port-id* is from 1024 to 65535. 50051 is the default. **Note** This command is available beginning with Cisco NX-OS Release 9.3(3).
Step 4	**grpc certificate** *certificate-id* **Example:** `switch-1(config)# grpc certificate cert-1`	Specify the certificate trustpoint ID. For more information, see the Installing Identity Certificates section of the Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x) for importing the certificate to the switch. **Note** This command is available beginning with Cisco NX-OS Release 9.3(3).

	Command or Action	Purpose

Step 5	**grpc gnmi max-concurrent-call** *number*
	Example:
	`switch-1(config)# grpc gnmi max-concurrent-call 16`
	`switch-1(config)#`
		Sets the limit of simultaneous dial-in calls to the gNMI server on the switch. Configure a limit from 1 through 16. The default limit is 8.
		The maximum value that you configure is for each VRF. If you set a limit of 16 and gNMI is configured for both management and default VRFs, each VRF supports 16 simultaneous gNMI calls.
		This command does not affect and ongoing or in-progress gNMI calls. Instead, gRPC enforces the limit on new calls, so any in-progress calls are unaffected and allowed to complete.
		Note The configured limit does not affect the gRPCConfigOper service.

Step 6	(Optional) **grpc use-vrf default** **Example:** `switch(config)# grpc use-vrf default`	Enables the gRPC agent to accept incoming (dial-in) RPC requests from the default VRF. This step enables the default VRF to process incoming RPC requests. By default, the management VRF processes incoming RPC requests when the gRPC feature is enabled. **Note** Both VRFs process requests individually, so that requests do not cross between VRFs.

サーバー証明書の構成

TLS 証明書を設定し、スイッチに正常にインポートした場合の **show grpc gnmi service statistics** コマンドの出力例を次に示します。

```
#show grpc gnmi service statistics

=============
gRPC Endpoint
=============

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

Cert notBefore : Mon Jan 27 15:34:08 PDT 2020
Cert notAfter  : Tue Jan 26 15:34:08 PDT 2021

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
```

gNMI は gRPC を介して通信し、TLS を使用してスイッチとクライアント間のチャネルをセキュアにします。デフォルトのハードコードされた gRPC 証明書は、スイッチに同梱されなくなりました。デフォルトの動作は、次に示すように、スイッチで生成される有効期限が 1 日の自己署名キーと証明書です。

証明書の有効期限が切れているか、正常にインストールできなかった場合は、1 日限りのデフォルト証明書が表示されます。次に、**show grpc gnmi service statistics** コマンドの出力を示します。

```
#show grpc gnmi service statistics

==============
gRPC Endpoint
==============

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

Cert notBefore : Wed Mar 11 19:43:01 PDT 2020
Cert notAfter  : Thu Mar 12 19:43:01 PDT 2020

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
```

有効期限は 1 日ですが、この一時証明書を使用してテストを簡単に行えます。長期的には、新しいキー/証明書を生成する必要があります。

キー/証明書の生成の例

キー/証明書を生成するには、次の例に従います。

- Generating and Configuring Key/Certificate Examples for Cisco NX-OS Release 9.3(3) and Later （314 ページ）

Generating and Configuring Key/Certificate Examples for Cisco NX-OS Release 9.3(3) and Later

The following is an example for generating key/certificate.

Note This task is an example of how a certificate can be generated on a switch. You can also generate a certificate in any Linux environment. In a production environment, you should consider using a CA signed certificate.

For more information on generating identity certificates, see the Installing Identity Certificates section of the *Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)*.

Step 1 Generate the selfsigned key and pem files.

a) switch# openssl req -x509 -newkey rsa:2048 -keyout self_sign2048.key -out self_sign2048.pem -days 365 -nodes

Step 2 After generating the key and pem files, you must bundle the key and pem files for use in the trustpoint CA Association.

```
switch# run bash sudo su
bash-4.3# cd /bootflash/
bash-4.3# openssl pkcs12 -export -out self_sign2048.pfx -inkey self_sign2048.key -in self_sign2048.pem
 -certfile self_sign2048.pem -password pass:Ciscolab123!
bash-4.3# exit
```

Step 3 Verify the setup.

```
switch(config)# show crypto ca certificates
Trustpoint: mytrustpoint
certificate:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
serial=0413
notBefore=Nov  5 16:48:58 2015 GMT
notAfter=Nov  5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient

CA certificate 0:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
serial=0413
notBefore=Nov  5 16:48:58 2015 GMT
notAfter=Nov  5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
```

Step 4 Configure gRPC to use the trustpoint.

```
switch(config)# grpc certificate mytrustpoint
switch(config)# show run grpc

!Command: show running-config grpc
!Running configuration last done at: Thu Jul  2 12:24:02 2020
!Time: Thu Jul  2 12:24:05 2020

version 9.3(5) Bios:version 05.38
feature grpc

grpc gnmi max-concurrent-calls 16
grpc use-vrf default
grpc certificate mytrustpoint
```

Step 5 Verify gRPC is now using the certificate.

```
switch# show grpc gnmi service statistics

=============
gRPC Endpoint
=============

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

Cert notBefore : Nov 5 16:48:58 2015 GMT
Cert notAfter : Nov 5 16:48:58 2035 GMT

Max concurrent calls : 16
Listen calls : 1
Active calls : 0

Number of created calls : 953
Number of bad calls : 0

Subscription stream/once/poll : 476/238/238

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 10
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0
```

gNMI の確認

gNMI 構成を確認するには、次のコマンドを入力します。

コマンド	説明
show grpc gnmi service statistics	管理 VRF またはデフォルト VRF（構成されている場合）のエージェント実行ステータスの概要を表示します。また、次の項目も表示されます。 • 基本の全般的なカウンタ • 証明書の有効期限日時 （注） 証明書の有効期限が切れている場合、エージェントは要求を受け入れることができません。

コマンド	説明
show grpc gnmi rpc summary	次のステータスが表示されます。 • 受信した機能 RPC の数。 • 機能 RPC エラー。 • 受信した Get RPC の数。 • Get RPC エラー。 • 受信した Set RPC の数。 • Set RPC エラー。 • 詳細なエラー タイプとエラー数。

コマンド	説明
show grpc gnmi transactions	

コマンド	説明
	show grpc gnmi transactions コマンドは最も密度が高く、多くの情報が含まれています。これは、スイッチが受信した最新の 50 gNMI トランザクションの履歴バッファです。新しい RPC が着信すると、末尾から最も古い履歴エントリが削除されます。次に、表示内容について説明します。 • [RPC]：受信した RPC のタイプ（Get、Set、Capabilities）を示します。 • [データタイプ（DataType）]：Get の場合のみです。値は ALL、CONFIG、および STATE です。 • [セッション（Session）]：このトランザクションに割り当てられている一意のセッション ID を示します。他のログ ファイルで見つかったデータを関連付けるために使用できます。 • [入力時間（Time In）]：gNMI ハンドラが RPC を受信したときのタイムスタンプを示します。 • [期間（Duration）]：要求を受信してから応答を返すまでの時間差です（ミリ秒単位）。 • [ステータス（Status）]：クライアントに返された操作のステータス コード（0 は成功、0 以外はエラー） 。 このセクションは、単一の gNMI トランザクション内のパスごとに保持されるデータです。たとえば、単一の Get または Set です。 • [サブタイプ（subtype）]：Set RPC の場合、パスごとに要求される特定の操作（削除、更新、置換）を示します。Get の場合、サブタイプはありません。 • [dtx]：このパスが DTX の「高速」パスで処理されたかどうかを示します。ダッシュ「-」は「いいえ」を意味し、アスタリスク「*」は「はい」を意味します。 • [st]：このパスのステータス。意味は次のと

コマンド	説明
	おりです。 • OK：パスは有効で、インフラによって正常に処理されました。 • ERR:パスが無効であるか、インフラによってエラーが生成されました。 • --：パスはまだ処理されていません。有効な場合と無効な場合がありますが、まだインフラに送信されていません。 • [path]：パス

show grpc gnmi service statistics の例

```
=============
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
```

show grpc gnmi rpc summary の例

```
=============
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
```

show grpc gnmi transactions の例

```
=============
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          -          2293989720      04/01 07:43:45       183          0
subtype: dtx:  st: path:
Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo6]

Set          -          2297110560      04/01 07:43:41       184          0
subtype: dtx:  st: path:
Update   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo7]

Set          -          0               04/01 07:43:39       0            10

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=lo791]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo792]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo793]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo794]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo795]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo796]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo797]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo798]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo799]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo800]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo801]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo802]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo803]
```

```
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo804]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo805]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo806]
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          -          2294408864      04/01 07:43:17       176          13
subtype: dtx:  st: path:
Delete   -     ERR /System/intf-items/lb-items/LbRtdIf-list/descr


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]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo777]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo778]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo779]
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo780]
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=lo4]/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

Set          -          3551604840      04/01 07:42:54       35           0
subtype: dtx:  st: path:
Delete   -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo1]

Set          -          2362201592      04/01 07:42:52       13           13
subtype: dtx:  st: path:
Delete   -     ERR /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/lbrtdif-items
/operSt


Set          -          0               04/01 07:42:47       0            3
subtype: dtx:  st: path:
Delete   -     ERR /System/*

Set          -          2464158360      04/01 07:42:46       172          3
subtype: dtx:  st: path:
Delete   -     ERR /system/processes/shabang

Set          -          2295440864      04/01 07:42:46       139          3
subtype: dtx:  st: path:
Delete   -     ERR /System/invalid/path
```

```
Set           -           3495739048      04/01 07:42:44       10           0


Get           ALL         3444580832      04/01 07:42:40       3            0
subtype: dtx:  st: path:
-         -     OK  /System/bgp-items/inst-items/disPolBatch

Get           ALL         0               04/01 07:42:36       0            3
subtype: dtx:  st: path:
-         -     --  /system/processes/process[pid=1]

Get           ALL         3495870472      04/01 07:42:36       2            0
subtype: dtx:  st: path:
-         *     OK  /system/processes/process[pid=1]

Get           ALL         2304485008      04/01 07:42:36       33           0
subtype: dtx:  st: path:
-         *     OK  /system/processes

Get           ALL         2464159088      04/01 07:42:36       251          0
subtype: dtx:  st: path:
-         -     OK  /system

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
```

gRPC クライアント証明書認証

10.1(1) リリース以降、gRPC に追加の認証方式が提供されます。 10.1(1) リリースより前の gRPC サービスは、サーバー証明書のみをサポートしていました。10.1(1) 以降では、クライアント証明書のサポートも追加するように認証が拡張され、gRPC でサーバー証明書とクライアント証明書の両方を検証できるようになっています。この機能拡張により、さまざまなクライアントにパスワードなしの認証が提供されます。

Generating New Client Root CA Certificates

The following is the example for generating a new certificate to the client root:

- Trusted Certificate Authorities (CA)

Perform the following steps when you use a trusted CA such as a DigiCert:

SUMMARY STEPS

1. Download the CA certificate file.
2. Import to NX-OS using the steps in Cisco NX-OS Security Configuration Guide.

DETAILED STEPS

	Command or Action	Purpose
Step 1	Download the CA certificate file.	
Step 2	Import to NX-OS using the steps in Cisco NX-OS Security Configuration Guide.	• To create a trustpoint label, use steps in Creating a Trustpoint CA Association • To authenticate the trustpoint using the trusted CA certificates, use steps in Authenticating the CA. **Note** Use the CA Certificate from cat [CA_cert_file].

NX-OS デバイスでの生成されたルート CA 証明書の構成

クライアント root に対する新しい証明書が正常に生成されたときの、スイッチで証明書を構成するためのコマンド例とその出力を次に示します。

```
switch(config)# crypto ca trustpoint my_client_trustpoint
enticate my_client_trustpoint
switch(config-trustpoint)# crypto ca authenticate my_client_trustpoint
input (cut & paste) CA certificate (chain) in PEM format;
end the input with a line containing only END OF INPUT :
-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJAJLisBKCGjQOMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJDQTERMA8GA1UEBwwIU2FuIEpvc2UxDjAMBgNVBAoM
BUNpc2NvMB4XDTIwMTAxNDIwNTYyN1oXDTQwMTAwOTIwNTYyN1owPTELMAkGA1UE
BhMCVVMxCzAJBgNVBAgMAkNBMREwDwYDVQQHDAhTYW4gSm9zZTEOMAwGA1UECgwF
Q2lzY28wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDEX7qZ2EdogZU4
EW0NSpB3EjY0nSlFLOw/iLKSXfIiQJD0Qhaw16fDnnYZj6vzWEa0ls8canqHCXQl
gUyxFOdGDXa6neQFTqLowSA6UCSQA+eenN2PIpMOjfdFpaPiHu3mmcTI1xP39Ti3
/y548NNORSepApBNkZ1rJSB6Cu9AIFMZgrZXFqDKBGSUOf/CPnvIDZeLcun+zpUu
CxJLA76Et4buPMysuRqMGHIX8CYw8MtjmuCuCTHXNN31ghhgpFxfrW/69pykjU3R
YOrwlSUkvYQhtefHuTHBmqym7MFoBEchwrlC5YTduDzmOvtkhsmpogRe3BiIBx45
AnZdtdi1AgMBAAGjUzBRMB0GA1UdDgQWBBSh3IqRrm+mtB5GNsoLXFb3bAVg5TAf
BgNVHSMEGDAWgBSh3IqRrm+mtB5GNsoLXFb3bAVg5TAPBgNVHRMBAf8EBTADAQH/
MA0GCSqGSIb3DQEBCwUAA4IBAQAZ4Fpc6lRKzBGJQ/7oK1FNcTX/YXkneXDk7Zrj
8W0RS0Khxgke97d2Cwl5P5reXO27kvXsnsz/VZn7JYGUvGSlxTlcCb6x6wNBr4Qr
t9qDBu+LykwqNOFe4VCAv6e4cMXNbH2wHBVS/NSoWnM2FGZ10VppjEGFm6OM+N6z
8n4/rWslfWFbn7T7xHH+Nl0Ffc+8q8h37opyCnb0ILj+a4rnyus8xXJPQb05DfJe
ahPNfdEsXKDOWkrSDtmKwtWDqdtjSQC4xioKHoshnNgWBJbovPlMQ64UrajBycwV
z9snWBm6p9SdTsV92YwF1tRGUqpcI9olsBgH7FUVU1hmHDWE
-----END CERTIFICATE-----
END OF INPUT
Fingerprint(s): SHA1 Fingerprint=0A:61:F8:40:A0:1A:C7:AF:F2:F7:D9:C7:12:AE:29:15:52:9D:D2:AE


Do you accept this certificate? [yes/no]:yes
switch(config)#

NOTE: Use the CA Certificate from the .pem file content.

switch# show crypto ca certificates
Trustpoint: my_client_trustpoint
CA certificate 0:
subject=C = US, ST = CA, L = San Jose, O = Cisco
```

```
issuer=C = US, ST = CA, L = San Jose, O = Cisco
serial=B7E30B8F4168FB87
notBefore=Oct  1 17:29:47 2020 GMT
notAfter=Sep 26 17:29:47 2040 GMT
SHA1 Fingerprint=E4:91:4E:D4:41:D2:7D:C0:5A:E8:F7:2D:32:81:B3:37:94:68:89:10
purposes: sslserver sslclient
```

gRPC へのトラストポイントの関連付け

クライアント ルートに新しい証明書を正常に構成した場合に、スイッチ上でトラストポイントを gRPC に関連付ける出力例を次に示します。

（注） クライアント認証用のルート証明書を構成または削除すると、gRPCプロセスが再起動します。

```
# switch(config)# feature grpc

switch(config)# grpc client root certificate my_client_trustpoint
switch(config)# show run grpc

!Command: show running-config grpc
!Running configuration last done at: Wed Dec 16 20:18:35 2020
!Time: Wed Dec 16 20:18:40 2020

version 10.1(1) Bios:version N/A
feature grpc

grpc gnmi max-concurrent-calls 14
grpc use-vrf default
grpc certificate my_trustpoint
grpc client root certificate my_client_trustpoint
grpc port 50003
```

証明書の詳細の検証

スイッチの gRPC にトラストポイントを正常に関連付けられた場合の、証明書の詳細を検証するための出力例を次に示します。

```
switch# show grpc gnmi service statistics

==============
gRPC Endpoint
==============

Vrf : management
Server address : [::]:50003

Cert notBefore : Mar 13 19:05:24 2020 GMT
Cert notAfter : Nov 20 19:05:24 2033 GMT
Client Root Cert notBefore : Oct 1 17:29:47 2020 GMT
Client Root Cert notAfter : Sep 26 17:29:47 2040 GMT

Max concurrent calls : 14
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 : 0
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0
```

任意の gNMI クライアントのクライアント証明書認証を使用した接続の確認

クライアント証明書は、秘密キー（pkey）とCAチェーン（cchain）を使用して要求を行います。現在では、パスワードはオプションです。

```
Performing GetRequest, encoding = JSON to 172.19.199.xxx with the following gNMI Path
 -------------------------
 [elem {
  name: "System"
}
elem {
  name: "bgp-items"
}
]
The GetResponse is below
-------------------------

notification {
  timestamp: 1608071208072199559
  update {
    path {
      elem {
        name: "System"
      }
      elem {
        name: "bgp-items"
      }
    }
    val {
      json_val: ""
    }
  }
}
```

gRPC からトラストポイント参照を削除するには（no コマンド）、次のコマンドを使用します。

```
 [no] grpc client root certificate <my_client_trustpoints>
switch(config)# no grpc client root certificate my_client_trustpoint
```

コマンドは、gRPC エージェントのトラストポイント参照だけを削除します。トラストポイントCA証明書は削除されません。スイッチ上のgRPCサーバーへのクライアント証明書認証を使用する接続は確立されませんが、ユーザー名とパスワードによる基本認証は通過します。

（注） クライアントの証明書が中間 CA によって署名されているが、上記の構成からインポートされたルート CA によって直接署名されていない場合、grpc クライアントは、ユーザー、中間 CA 証明書、およびルート CA 証明書を含む完全な証明書チェーンを提供する必要があります。

クライアント

gNMIサブスクリプションには、使用可能なクライアントがいくつかあります。このようなクライアントの 1 つは https://github.com/influxdata/telegraf/tree/master/plugins/inputs/cisco_telemetry_gnmi にあります。

DMEサブスクリプションの例：PROTOエンコーディング

```
gnmi-console --host >iip> --port 50051 -u <user> -p <pass> --tls --
operation=Subscribe --rpc /root/gnmi-console/testing_bl/once/61_subscribe_bgp_dme_gpb.json

[Subscribe]--------------------------------
### Reading from file ' /root/gnmi-console/testing_bl/once/61_subscribe_bgp_dme_gpb.json '
Wed Jun 26 11:49:17 2019
### Generating request : 1 -----------
### Comment : ONCE request
### Delay : 2 sec(s) ...
### Delay : 2 sec(s) DONE
subscribe {
subscription {
path {
origin: "DME"
elem {
name: "sys"
}
elem {
name: "bgp"
}
}
mode: SAMPLE
}
mode: ONCE
use_models {
name: "DME"
organization: "Cisco Systems, Inc."
version: "1.0.0"
}
encoding: PROTO
}
Wed Jun 26 11:49:19 2019
Received response 1 --------------------------
update {
timestamp: 1561574967761
prefix {
elem {
name: "sys"
}
elem {
```

```
name: "bgp"
}
}
update {
path {
elem {
}
elem {
name: "version_str"
}
}
val {
string_val: "1.0.0"
}
}
update {
path {
elem {
}
elem {
name: "node_id_str"
}
}
val {
string_val: "n9k-tm2"
}
}
update {
path {
elem {
}
elem {
name: "encoding_path"
}
}
val {
string_val: "sys/bgp"
}
}
update {
path {
elem {
}
elem {
/Received --------------------------------------
Wed Jun 26 11:49:19 2019
Received response 2 -------------------------
sync_response: true
/Received --------------------------------------
(_gnmi) [root@tm-ucs-1 gnmi-console]#
```

機能

機能について

Capabilities RPC は、gNMI サービスの機能のリストを返します。RPC 要求に対する応答メッセージには、gNMI サービスのバージョン、バージョン管理されたデータ モデル、およびサーバーでサポートされているデータ エンコーディングが含まれます。

Guidelines and Limitations for Capabilities

Following are the guidelines and limitations for Capabilities:

- Beginning with Cisco NX-OS Release 9.3(3), Capabilities supports the OpenConfig model.
- For information about supported platforms, see Nexus Switch Platform Matrix.
- The gNMI feature supports Subscribe and Capability as options of the gNMI service.
- The feature supports JSON and gnmi.proto encoding. The feature does not support protobuf.any encoding.
- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the 12-MB maximum, the collected data is dropped.

  You can avoid this situation by creating more focused subscriptions that handle smaller, more granular data-collection sets. So, instead of subscribing to one higher-level path, create multiple subscriptions for different, lower-level parts of the path.
- All paths within the same subscription request must have the same sample interval. If the same path requires different sample intervals, create multiple subscriptions.
- The feature does not support a path prefix in the Subscription request, but the Subscription can contain an empty prefix field.
- The feature supports Cisco DME and Device YANG data models.
- The gRPC process that supports gNMI uses the HIGH_PRIO cgroup, which limits the CPU usage to 75% of CPU and memory to 1.5 GB.
- The **show grpc gnmi** command has the following considerations:
  - The commands are not XMLized in this release.
  - The gRPC agent retains gNMI calls for a maximum of 1 hour after the call has ended.
  - If the total number of calls exceeds 2000, the gRPC agent purges ended calls based an internal cleanup routine.

The gRPC server runs in the management VRF. As a result, the gRPC process communicates only in this VRF forcing the management interface to support all gRPC calls.

gRPC functionality now includes the default VRF for a total of 2 gRPC servers on each Cisco Nexus 9000 switch. You can run one gRPC server in each VRF, or run only one gRPC server in the management VRF. Supporting a gRPC in the default VRF adds flexibility to offload processing gRPC calls from the management VRF, where significant traffic load might not be desirable.

If two gRPC servers are configured, be aware of the following:

- VRF boundaries are strictly enforced, so each gRPC server processes requests independent of the other, and requests do not cross between VRFs.
- The two servers are not HA or fault tolerant. One gRPC server does not back up the other, and there is no switchover or switchback between them.
- Any limits for the gRPC server are per VRF.

機能のクライアント出力の例

この例では、すべての OpenConfig モデル RPM がスイッチにインストールされています。

次に、機能のクライアント出力の例を示します。

```
hostname user$ ./gnmi_cli -a 172.19.193.166:50051 -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-interfaces"
  organization: "OpenConfig working group"
  version: "2.0.0"
>
supported_models: <
  name: "openconfig-if-aggregate"
  organization: "OpenConfig working group"
  version: "2.0.0"
>
supported_models: <
  name: "openconfig-if-ethernet"
  organization: "OpenConfig working group"
  version: "2.0.0"
>
supported_models: <
  name: "openconfig-if-ip"
  organization: "OpenConfig working group"
  version: "2.3.0"
>
supported_models: <
  name: "openconfig-if-ip-ext"
  organization: "OpenConfig working group"
  version: "2.3.0"
>
supported_models: <
  name: "openconfig-lacp"
  organization: "OpenConfig working group"
  version: "1.0.2"
>
supported_models: <
  name: "openconfig-lldp"
  organization: "OpenConfig working group"
  version: "0.2.1"
>
supported_models: <
  name: "openconfig-network-instance"
  organization: "OpenConfig working group"
  version: "0.11.1"
>
supported_models: <
```

```
  name: "openconfig-network-instance-policy"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-ospf-policy"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-platform"
  organization: "OpenConfig working group"
  version: "0.12.2"
>
supported_models: <
  name: "openconfig-platform-cpu"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-platform-fan"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-platform-linecard"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-platform-port"
  organization: "OpenConfig working group"
  version: "0.3.2"
>
supported_models: <
  name: "openconfig-platform-psu"
  organization: "OpenConfig working group"
  version: "0.2.1"
>
supported_models: <
  name: "openconfig-platform-transceiver"
  organization: "OpenConfig working group"
  version: "0.7.0"
>
supported_models: <
  name: "openconfig-relay-agent"
  organization: "OpenConfig working group"
  version: "0.1.0"
>
supported_models: <
  name: "openconfig-routing-policy"
  organization: "OpenConfig working group"
  version: "2.0.1"
>
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"

hostname user$
```

結果

Get について

Get RPC の目的は、クライアントがデバイスからデータ ツリーのスナップショットを取得できるようにすることです。1 つの要求で複数のパスを要求できます。gNMI パス規約に従って、XPATH の簡易形式である gNMI スキーマ パス エンコーディング規約がパスに使用されます。

Get 操作の詳細については、gNMI の仕様である gRPC ネットワーク管理インターフェイス（gNMI）の「状態情報のスナップショット取得」セクションを参照してください。

Get に関する注意事項と制限事項

次に、Get および Set に関する注意事項と制限事項を示します。

- GetRequest.encoding は JSON のみをサポートします。
- GetRequest.type の場合、DataType CONFIG と STATE のみが YANG で直接の相関関係と式を持ちます。OPERATIONAL はサポートされていません。
- 1 つの要求に OpenConfig（OC）YANG パスとデバイス YANG パスの両方を含めることはできません。要求には、OC YANG パスまたはデバイス YANG パスのみを含める必要があります。両方を含めることはできません。
- ルートパス（「/」：**すべて**のモデルのすべて）の GetRequest は許可されていません。
- デバイスモデルの最上位レベル（「/System」）の GetRequest は許可されていません。

- gNMI Get はすべてのデフォルト値を返します（RFC 6243 [4] の report-all モードを参照）。
- Subscribe は、モデル `Cisco-NX-OS-syslog-oper` をサポートします。
- Get はモデル `Cisco-NX-OS-syslog-oper` をサポートしていません。
- パス `/system` からのクエリは、パス `/system/processes` からのデータを返しません。`openconfig-procmon` データのクエリには、特定のパス `/system/processes` を使用する必要があります。
- 次のオプション項目はサポートされていません。
  - パスのプレフィックス
  - パスのエイリアス
  - パス内のワイルドカード
- 1 つの GetRequest には最大 10 のパスを含めることができます。
- GetResponse で返される値フィールドのサイズが 12 MB を超える場合、システムはエラー ステータス `grpc::RESOURCE_EXHAUSTED` を返します。
- 最大 gRPC 受信バッファサイズは 8 MB に設定されています。
- Get の同時セッションの合計数は 5 に制限されています。
- 大規模な構成がスイッチに適用されているときに Get 操作を実行すると、gRPC プロセスが使用可能なすべてのメモリを消費する可能性があります。メモリ枯渇状態が発生すると、次の syslog が生成されます。

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

  この条件が複数回連続して発生すると、次の syslog が生成されます。

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

  この時点で gRPC 機能を再起動して、gNMI トランザクションの通常の処理を続行することをお勧めします。
- Get と Set の同時セッションの合計数は、現在構成されている gNMI の同時最大数から 1 を引いたものです。たとえば、gnmi の同時呼び出しが 16 に構成されている場合、Get および Set の合計同時セッションの最大数は 15 になります。
- 大規模な構成がスイッチに適用されているときに Get 操作を実行すると、gRPC プロセスが要求を処理できなくなる可能性があります。その時点で、次のエラーが返されます。

  There is insufficient memory available on the device to process the subscription.

設定

Set について

Set RPC は、デバイスの構成を変更するためにクライアントによって使用されます。デバイスデータに適用できる操作は削除、置換、更新で、順番を付けて行われます。単一の Set 要求のすべての操作はトランザクションとして扱われます。つまり、すべての操作が成功しなかった場合は、デバイスが元の状態にロールバックされます。Set 操作は、SetRequest で指定された順序で適用されます。パスが複数回指定されている場合、互いを上書きすることになったとしても、変更が適用されます。データの最終状態は、トランザクションの最終操作によって実現されます。SetRequest::delete、replace、update フィールドで指定されたすべてのパスは CONFIG データパスであり、クライアントによって書き込み可能であると想定されています。

Set 操作の詳細については、gNMI 仕様、https://github.com/openconfig/reference/blob/1cf43d2146f9ba70abb7f04f6b0f6eaa504cef05/rpc/gnmi/gnmi-specification.md の「Modifying State」のセクションを参照してください。

Set に関する注意事項と制限事項

次に、Set に関する注意事項と制限事項を示します。

- SetRequest.encoding は JSON のみをサポートします。
- 1 つの要求に OpenConfig（OC）YANG パスとデバイス YANG パスの両方を含めることはできません。要求には、OC YANG パスまたはデバイス YANG パスのみを含める必要があります。両方を含めることはできません。
- Subscribe は、モデル `Cisco-NX-OS-syslog-oper` をサポートします。
- パス `/system` からのクエリは、パス `/system/processes` からのデータを返しません。`openconfig-procmon` データのクエリには、特定のパス `/system/processes` を使用する必要があります。
- 次のオプション項目はサポートされていません。
  - パスのプレフィックス
  - パスのエイリアス
  - パス内のワイルドカード
- 1 つの SetRequest には最大 20 のパスを含めることができます。
- 最大 gRPC 受信バッファサイズは 8 MB に設定されます。
- Get と Set の同時セッションの合計数は、現在構成されている gNMI の同時最大数から 1 を引いたものです。たとえば、gNMI の同時呼び出しが 16 に設定されている場合、Get および Set の合計同時セッションの最大数は 15 になります。

- Set::Delete RPC で、操作対象の構成が大きすぎる可能性がある場合、MTX ログ メッセージに警告が記録されます。

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

登録

Guidelines and Limitations for Subscribe

Following are the guidelines and limitations for Subscribe:

- If you configure a routing-policy **prefix-list** using the CLI and request gNMI Subscription for the routing-policy OpenConfig model, it is not supported. For example, when you attempt to subscribe an OpenConfig routing policy with a preexisting CLI configuration like the following, it returns empty values due to the 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
Using the example paths,
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
```

- Beginning with Cisco NX-OS Release 9.3(3), Subscribe supports the OpenConfig model.
- For information about supported platforms, see the Nexus Switch Platform Matrix.
- The gNMI feature supports Subscribe and Capability RPCs.
- The feature supports JSON and gnmi.proto encoding. The feature does not support protobuf.any encoding.
- Each gNMI message has a maximum size of 12 MB. If the amount of collected data exceeds the 12 MB maximum, the collected data is dropped.

  You can avoid this situation by creating more focused subscriptions that handle smaller, more granular data-collection sets. So, instead of subscribing to one higher-level path, create multiple subscriptions for different, lower-level parts of the path.
- All paths within the same subscription request must have the same sample interval. If the same path requires different sample intervals, create multiple subscriptions.
- The feature does not support a path prefix in the Subscription request, but the Subscription can contain an empty prefix field.
- The feature supports Cisco DME and Device YANG data models.
- The gRPC process that supports gNMI uses the HIGH_PRIO cgroup, which limits the CPU usage to 75% of CPU and memory to 1.5 GB.
- The **show grpc gnmi** command has the following considerations:
  - The commands are not XMLized in this release.
  - The gRPC agent retains gNMI calls for a maximum of 1 hour after the call has ended.

- If the total number of calls exceeds 2000, the gRPC agent purges ended calls based an internal cleanup routine.

The gRPC server runs in the management VRF. As a result, the gRPC process communicates only in this VRF forcing the management interface to support all gRPC calls.

gRPC functionality now includes the default VRF for a total of 2 gRPC servers on each Cisco Nexus 9000 switch. You can run one gRPC server in each VRF, or run only one gRPC server in the management VRF. Supporting a gRPC in the default VRF adds flexibility to offload processing gRPC calls from the management VRF, where significant traffic load might not be desirable.

If two gRPC servers are configured, be aware of the following:

- VRF boundaries are strictly enforced, so each gRPC server processes requests independent of the other, and requests do not cross between VRFs.
- The two servers are not HA or fault tolerant. One gRPC server does not back up the other, and there is no switchover or switchback between them.
- Any limits for the gRPC server are per VRF.

gNMI ペイロード

gNMI は、特定のペイロード形式を使用して次のものにサブスクライブします：

- DME ストリーム
- YANG ストリーム

サブスクライブ操作は、次のモードでサポートされています：

- ONCE：データを 1 回サブスクライブして受信し、セッションを閉じます。
- POLL：サブスクライブしてセッションを開いたままにします。クライアントはデータが必要になるたびにポーリング要求を送信します。
- STREAM：特定の頻度でデータをサブスクライブし、受信します。ペイロードはナノ秒単位で値を受け入れます。1 秒 = 1000000000 ナノ秒です。
- ON_CHANGE：サブスクライブしてスナップショットを受信します。ツリーで何かが変更された場合にのみデータを受信します。

設定モード：

- 各モードには、内部サブと外部サブの 2 つの設定が必要です。
- ONCE：SAMPLE、ONCE
- POLL：SAMPLE、POLL
- STREAM：SAMPLE、STREAM
- ON_CHANGE：ON_CHANGE、STREAM

Origin

- DME：DME モデルへのサブスクライブ
- device：YANG モデルへのサブスクライブ

名前

- DME = DME モデルへのサブスクライブ
- Cisco-NX-OS-device = YANG モデルへのサブスクライブ

エンコーディング

- JSON = ストリームは JSON 形式で送信されます。
- PROTO = ストリームは protobuf.any 形式で送信されます。

DME ストリームの gNMI ペイロードの例

（注） クライアントごとに独自の入力形式があります。

```
{
    "SubscribeRequest":
    [
        {
            "_comment" : "ONCE request",
            "_delay" : 2,
            "subscribe":
            {
                "subscription":
                [
                    {
                        "_comment" : "1st subscription path",
                        "path":
                        {
                            "origin": "DME",
                            "elem":
                            [
                                {
                                    "name": "sys"
                                },
                                {
                                    "name": "bgp"
                                 }
                             ]
                        },
                        "mode": "SAMPLE"
                    }
                ],
                "mode": "ONCE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "_comment" : "1st module",
                        "name": "DME",
```

```
                        "organization": "Cisco Systems, Inc.",
                        "version": "1.0.0"
                    }
                ],
                "encoding": "JSON"
            }
        }
    ]
}
```

gNMI ペイロード YANG ストリームの例

```
{
    "SubscribeRequest":
    [
        {
            "_comment" : "ONCE request",
            "_delay" : 2,
            "subscribe":
            {
                "subscription":
                [
                    {
                        "_comment" : "1st subscription path",
                        "path":
                        {
                            "origin": "device",
                            "elem":
                            [
                                {
                                    "name": "System"
                                },
                                {
                                    "name": "bgp-items"
                                }
                            ]
                        },
                                                    "mode": "SAMPLE"
                    }
                ],
                "mode": "ONCE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "_comment" : "1st module",
                        "name": "Cisco-NX-OS-device",
                        "organization": "Cisco Systems, Inc.",
                        "version": "0.0.0"
                    }
                ],
                "encoding": "JSON"
            }
        }
    ]
}
```

ストリーミング Syslog

gNMI のストリーミング Syslog について

gNMI サブスクライブは、gNMI サブスクライブ要求に従って構造化データをプッシュすることで、システムで何が起こっているのかをリアルタイムで表示する、ネットワークをモニターする新しい方法です。

Cisco NX-OS リリース 9.3(3) 以降では、gNMI サブスクライブ機能の亜ポートが追加されました。

gNMI サブスクライブ サポートの詳細

- Syslog-oper モデルのストリーミング
  - stream_on_change

この機能は、8 GB 以上のメモリを搭載した Cisco Nexus 9000 シリーズ スイッチに適用されます。

ストリーミング Syslog に関する注意事項と制限事項：gNMI

ストリーミング Syslog に関する注意事項と制限事項は次のとおりです。

- 無効な syslog はサポートされていません。たとえば、フィルタまたはクエリ条件を持つ syslog です。
- 次のパスだけがサポートされます：
  - Cisco-NX-OS-Syslog-oper:syslog
  - Cisco-NX-OS-Syslog-oper:syslog/messages
- 次のモードはサポートされていません。
  - ストリーム サンプル
  - 投票
- 要求は YANG モデル フォーマットである必要があります。
- 内部アプリケーションを使用することも、独自のアプリケーションを作成することもできます。
- ペイロードはコントローラから送信され、gNMI は応答を送信します。
- エンコーディング フォーマットは JSON と PROTO です。

Syslog ネイティブ YANG モデル

YangModel はここにあります。

（注） タイムゾーン フィールドは、**clock format show-timezone syslog** が入力された場合にのみ設定されます。デフォルトでは設定されていないため、タイムゾーン フィールドは空です。

```
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
```

サブスクライブ要求の例

次に、サブスクライブ要求の例を示します。

```
{
    "SubscribeRequest":
    [
        {
            "_comment" : "STREAM request",
            "_delay"   : 2,
            "subscribe":
            {
                "subscription":
                [
                    {
                        "_comment" : "1st subscription path",
                        "path":
                        {
                            "origin": "syslog-oper",
                            "elem":
                              [
                                {
                                    "name": "syslog"
                                },
                                {
                                    "name":"messages"
                                }
                            ]
                        },
                        "mode": "ON_CHANGE"
                    }
                ],
```

```
                "mode": "ON_CHANGE",
              "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "_comment" : "1st module",
                        "name": "Cisco-NX-OS-Syslog-oper",
                        "organization": "Cisco Systems, Inc.",
                        "version": "0.0.0"
                    }
                ],
                "encoding":"JSON"
            }
        }
    ]
}
```

PROTO 出力の例

これは PROTO 出力のサンプルです。

```
############################

[Subscribe]--------------------------------

### Reading from file ' /root/gnmi-console/testing_bl/stream_on_change/OC_SYSLOG.json '

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: "message-id"
}
}
val {
uint_val: 529
}
}
update {
path {
elem {
name: "node-name"
}
}
val {
string_val: "task-n9k-1"
}
}
update {
path {
elem {
name: "message-name"
}
}
val {
string_val: "VSHD_SYSLOG_CONFIG_I"
}
}
update {
path {
elem {
name: "text"
}
}
```

```
val {
string_val: "Configured from vty by admin on console0"
}
}
update {
path {
elem {
name: "group"
}
}
val {
string_val: "VSHD"
}
}
update {
path {
elem {
name: "category"
}
}
val {
string_val: "VSHD"
}
}
update {
path {
elem {
name: "time-of-day"
}
}
val {
string_val: "Nov 21 2019 14:26:40"
}
}
update {
path {
elem {
name: "time-zone"
}
}
val {
string_val: ""
}
}
update {
path {
elem {
name: "time-stamp"
}
}
val {
uint_val: 1574375200000
}
}
update {
path {
elem {
name: "severity"
}
}
val {
uint_val: 5
}
}
```

```
}

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

-

JSON 出力の例

これは JSON 出力の例です。

```
[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\":\"\"} ]] } ]"
}
}
}
```

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

トラブルシューティング

TM トレース ログの収集

```
1. tmtrace.bin -f gnmi-logs gnmi-events gnmi-errors following are available
2. Usage:

bash-4.3# tmtrace.bin -d gnmi-events | tail -30 Gives the last 30
}
}
}
[06/21/19 15:58:38.969 PDT f8f 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 124, sync_response:1
[06/21/19 15:58:43.210 PDT f90 3133] [3621780288][tm_ec_yang_data_processor.c:93] TM_EC:
[Y] Data received for 2799743488: 49
{
"cdp-items" : {
"inst-items" : {
"if-items" : {
"If-list" : [
{
"id" : "mgmt0",
"ifstats-items" : {
"v2Sent" : "74",
"validV2Rcvd" : "79"
}
}
]
}
}
}
}
[06/21/19 15:58:43.210 PDT f91 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 141, sync_response:1
[06/21/19 15:59:01.341 PDT f92 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/intf-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157935518, length: 3063619, sync_response:0
[06/21/19 15:59:03.933 PDT f93 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157940881, length: 6756, sync_response:0
[06/21/19 15:59:03.940 PDT f94 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/lldp-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157940912, length: 8466, sync_response:1
bash-4.3#
```

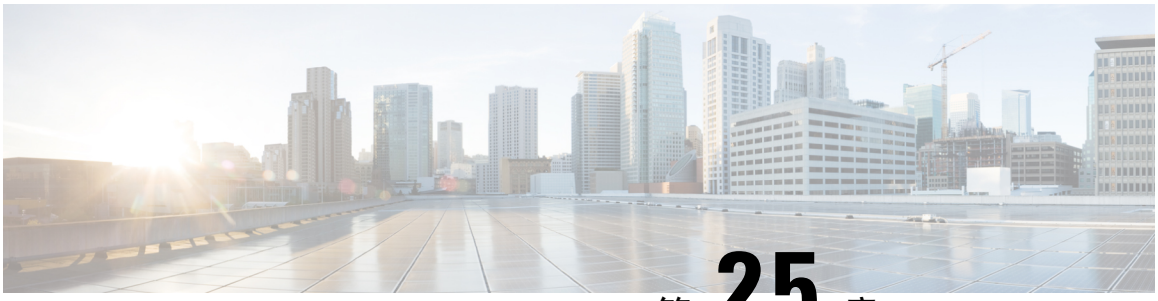
MTX 内部ログの収集

```
1. Modify the following file with below /opt/mtx/conf/mtxlogger.cfg

<config name="nxos-device-mgmt">
  <container name="mgmtConf">
```

```
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="allActive" type="boolean" default="false">true<
/leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$
$COMPONENTID$ $TYPE$: $MSG$">$DATETIME$ $COMPONENTID$ $TYPE$
$SRCFILE$ @ $SRCLINE$ $FCNINFO$:$MSG$</leaf>
          <container name="componentID">
          <leaf name="enabled" type="boolean" default="true"></leaf>
          </container>
          <container name="dateTime">
          <leaf name="enabled" type="boolean" default="true"></leaf>
          <leaf name="format" type="string" default="%y%m%d.%H%M%S"><
/leaf>
           </container>
           <container name="fcn">
         <leaf name="enabled" type="boolean" default="true"></leaf>
         <leaf name="format" type="string"
default="$CLASS$::$FCNNAME$($ARGS$)@$LINE$"></leaf>
           </container>
      </container>
      <container name="facility">
          <leaf name="info" type="boolean" default="true">true</leaf>
          <leaf name="warning" type="boolean" default="true">true<
/leaf>
          <leaf name="error" type="boolean" default="true">true</leaf>

Note: Beginning with Cisco NX-OS Release 9.3(4), the following default configuration is
changed from
      default true to false. To investigate an issue which requires the debug messages,
edit
      the following configuration and toggle it to true.

          <leaf name="debug" type="boolean" default="false">true<
/leaf>
        </container>
        <container name="dest">
          <container name="console">
            <leaf name="enabled" type="boolean" default="false">true<
/leaf>
          </container>
          <container name="file">
         <leaf name="enabled" type="boolean" default="false">true<
/leaf>
    <leaf name="name" type="string" default="mtx-internal.log">
/leaf>


        <leaf name="location" type="string" default="./mtxlogs">
/volatile</leaf>
              <leaf name="mbytes-rollover" type="uint32" default="10"
>50</leaf>
              <leaf name="hours-rollover" type="uint32" default="24"
>24</leaf>
              <leaf name="startup-rollover" type="boolean" default="
false">true</leaf>
            <leaf name="max-rollover-files" type="uint32" default="10"
>10</leaf>
        </container>
      </container>
      <list name="logitems" key="id">
        <listitem>
                <leaf name="id" type="string">*</leaf>
```

```
                    <leaf name="active" type="boolean" default="false"
>false</leaf>
          </listitem>
           <listitem>
                  <leaf name="id" type="string">MTX-EvtMgr</leaf>
                      <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
        <listitem>
               <leaf name="id" type="string">TM-ADPT</leaf>
                   <leaf name="active" type="boolean" default="true"
>false</leaf>
        </listitem>
        <listitem>
              <leaf name="id" type="string">TM-ADPT-JSON</leaf>
                  <leaf name="active" type="boolean" default="true"
>false</leaf>
        </listitem  >
        <listitem>
                <leaf name="id" type="string">SYSTEM</leaf>
                    <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
        <listitem>
                <leaf name="id" type="string">LIBUTILS</leaf>
                     <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
        <listitem>
                <leaf name="id" type="string">MTX-API</leaf>
                    <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
         <listitem>
                 <leaf name="id" type="string">Model-*</leaf>
                     <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
        <listitem>
                <leaf name="id" type="string">Model-Cisco-NX-OS-
device</leaf>
                 <leaf name="active" type="boolean" default="true"
>false</leaf>
        </listitem>
        <listitem>
                 <leaf name="id" type="string">Model-openconfig-bgp<
/leaf>
                     <leaf name="active" type="boolean" default="true"
>false</leaf>
        </listitem>
        <listitem>
               <leaf name="id" type="string">INST-MTX-API</leaf>
                   <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
        <listitem>
                <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
                    <leaf name="active" type="boolean" default="true"
>true</leaf>
        </listitem>
        <listitem>
               <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
                   <leaf name="active" type="boolean" default="true"
>true</leaf>
```

```
        </listitem>
        <listitem>
               <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
                   <leaf name="active" type="boolean" default="true"
>true</leaf>
         </listitem>
      </list>
    </container>
  </container>
</config>

2. Run "no feature grpc" / "feature grpc"
3. The /volatile directory houses the mtx-internal.log, the log rolls over time so be sure
 to grab what you need before then.

bash-4.3# cd /volatile/
bash-4.3# cd /volatile -al
total 148
drwxrwxrwx 4 root root 340 Jun 21 15:47 .
drwxrwxr-t 64 root network-admin 1600 Jun 21 14:45 ..
-rw-rw-rw- 1 root root 103412 Jun 21 16:14 grpc-internal-log
-rw-r--r-- 1 root root 24 Jun 21 14:44 mtx-internal-19-06-21-14-46-21.log
-rw-r--r-- 1 root root 24 Jun 21 14:46 mtx-internal-19-06-21-14-46-46.log
-rw-r--r-- 1 root root 175 Jun 21 15:11 mtx-internal-19-06-21-15-11-57.log
-rw-r--r-- 1 root root 175 Jun 21 15:12 mtx-internal-19-06-21-15-12-28.log
-rw-r--r-- 1 root root 175 Jun 21 15:13 mtx-internal-19-06-21-15-13-17.log
-rw-r--r-- 1 root root 175 Jun 21 15:13 mtx-internal-19-06-21-15-13-42.log
-rw-r--r-- 1 root root 24 Jun 21 15:13 mtx-internal-19-06-21-15-14-22.log
-rw-r--r-- 1 root root 24 Jun 21 15:14 mtx-internal-19-06-21-15-19-05.log
-rw-r--r-- 1 root root 24 Jun 21 15:19 mtx-internal-19-06-21-15-47-09.log
-rw-r--r-- 1 root root 24 Jun 21 15:47 mtx-internal.log
-rw-rw-rw- 1 root root 355 Jun 21 14:44 netconf-internal-log
-rw-rw-rw- 1 root root 0 Jun 21 14:45 nginx_logflag
drwxrwxrwx 3 root root 60 Jun 21 14:45 uwsgipy
drwxrwxrwx 2 root root 40 Jun 21 14:43 virtual-instance
bash-4.3#.
```



第 25 章

gNOI-gRPC ネットワーク操作インターフェイス

About gNOI

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based micro-services for executing operational commands on network devices. The operational commands supported are Ping, Traceroute, Time, SwitchControlProcessor, Reboot, RebootStatus, CancelReboot, Activate and Verify.

gNOI uses gRPC as the transport protocol and the configuration is same as that of gNMI. For details on configuration, please refer to Configuring gNMI.

To send gNOI RPC requests, user needs a client that implements the gNOI client interface for each RPC.

In Cisco NX-OS Release 10.1(1) the gNOI defines Remote Procedure Calls (RPCs) for a limited number of components and some of them related to hardware (like optical interfaces).

Proto files are defined for the gRPC micro-services and are available at GitHub.

サポートされる gNOI RPC

サポートされている gNOI RPC は次のとおりです。

表 *23 :*

プロトコル	gNOI RPC	サポート対象
System	ping	○
	トレースルート	はい
	時間	はい
	SwitchControl プロセッサ	はい
	リブート	はい
	RebootStatus	はい
	CancelReboot	はい
OS	アクティブ化	はい
	インターフェイス	はい
Cert	LoadCertificate	はい
File	get	はい
	Stat	はい
	削除	はい

システム proto

システム proto サービスは、設定およびテレメトリ パイプラインの外部でターゲットを管理できるようにする操作可能な RPC のコレクションです。

次に、システム proto の RPC サポートの詳細を示します。

RPC	サポート	説明	制限事項
ping	ping/ping6 cli コマンド	ターゲットで ping コマンドを実行し、結果をストリームで返します。一部のターゲットでは、すべての結果が利用可能になるまで結果がストリーミングされない場合があります。パケット数が明示的に指定されていない場合は、5 が使用されます。	do_not_resolve オプションはサポートされていません。
トレースルート	traceroute/traceroute6 cli コマンド	ターゲットで traceroute コマンドを実行し、結果をストリームで返します。一部のターゲットでは、すべての結果が使用可能になるまで結果がストリーミングされない場合があります。最大ホップ カウント 30 が使用されます。	itial_ttl、marx_ttl、wait、do_not_fragment、do_not_resolve、および l4protocol オプションはサポートされていません。
時間	ローカル時刻	ターゲットの現在の時刻を返します。通常、ターゲットが応答しているかどうかをテストするために使用されます。	-
SwitchControl プロセッサ	system switchover cli コマンド	現在のルート プロセッサから指定されたルート プロセッサに切り替えます。スイッチオーバーは即座に発生しますが、応答がクライアントに返されるとは保証されない場合があります。	スイッチオーバーは即座に発生します。その結果、応答がクライアントに返されるとは保証されない場合があります。

RPC	サポート	説明	制限事項
リブート	reload module	ターゲットをリブートします。	message オプションはサポートされません。delay オプションはスイッチのリロードでサポートされます。path オプションは 1 つのモジュール番号を受け入れます。
RebootStatus	show version [module] cli コマンド	ターゲットのリブートのステータスを返します。	-
CancelReboot	reload cancel	保留中のリブート要求をキャンセルします。	-

（注） SetPackage RPC はサポートされていません。

OS プロトコル

OS サービスは、ターゲット上の OS インストールに対するインターフェイスを提供します。OS パッケージのファイル フォーマットは、プラットフォームによって異なります。プラットフォームは、提供された OS パッケージが有効でブート可能であることを検証する必要があります。これには、既知の正しいハッシュに対するハッシュ チェックを含める必要があります。ハッシュは OS パッケージに埋め込むことをお勧めします。

ターゲットは、独自の永続ストレージと OS インストール プロセスを管理します。一連の個別の OS パッケージを保存し、新しい OS パッケージが届く場合に備えて、事前にスペースを解放します。ターゲットには、有効な新しい OS パッケージ用の十分なスペースが常にあることが保証されます。現在実行中の OS パッケージは削除しないでください。クライアントは、最後に正常にインストールされたパッケージが使用可能であることを想定して動作するからです。

次に、OS proto の RPC サポートの詳細を示します。

RPC	サポート	説明	制限事項
アクティブ化	install all nxos bootflash:///img_name	要求された OS バージョンを、次回のリブート時に使用されるバージョンとして設定します。この RPC は、ターゲットをリブートします。	リブートに失敗した場合、ロールバックまたは回復はできません。
検証	show version	[検証（Verify）] は、実行中の OS バージョンを確認します。この RPC は、ターゲットの起動中に成功するまで複数回呼び出される場合があります。	-

（注） インストール RPC はサポートされていません。

証明書 Proto

証明書管理サービスは、ターゲットによってエクスポートされます。ローテーション、インストール、およびその他の証明書 Proto RPC はサポートされていません。

次に、証明書 proto の RPC サポートの詳細を示します。

RPC	サポート	説明	制限事項
LoadCertificate	crypto ca import <trustpoint> pkcs12 <file> <passphrase>	CA 証明書のバンドルをロードします。	-

ファイル Proto

ファイル proto は、file.proto RPC の機能に基づいてメッセージをストリーミングします。ここに記載されていない Put およびその他の RPC は、ファイル Proto ではサポートされていません。

Get、Stat、および Remove RPC は、bootflash、bootflash://sup-remote、logflash、logflash://sup-remote、usb、volatile、volatile://sup-remote、および debug のファイル システムをサポートします。

次に、ファイル proto の RPC サポートの詳細を示します。

RPC	説明	制限事項
結果	Get はターゲットからファイルの内容を読み取り、ストリーミングします。ファイルは連続したメッセージによってストリーミングされます。各メッセージには最大 64 KB のデータが含まれます。最後のメッセージが送信された後、送信されたデータのハッシュが送信され、ストリームが閉じられます。ファイルが存在しない場合、またはファイルの読み取り中にエラーが発生した場合は、エラーが返されます。	ファイルサイズの上限は 32 MB です。
Stat	Stat は、ターゲット上のファイルに関するメタデータを返します。ファイルが存在しない場合、またはファイルのメタデータへのアクセス中にエラーが発生した場合は、エラーが返されます。	-
削除	Remove は、ターゲットから指定されたファイルを削除します。ファイルが存在しない場合、ディレクトリである場合、または削除操作でエラーが発生した場合は、エラーが返されます。	-

注意事項と制約事項

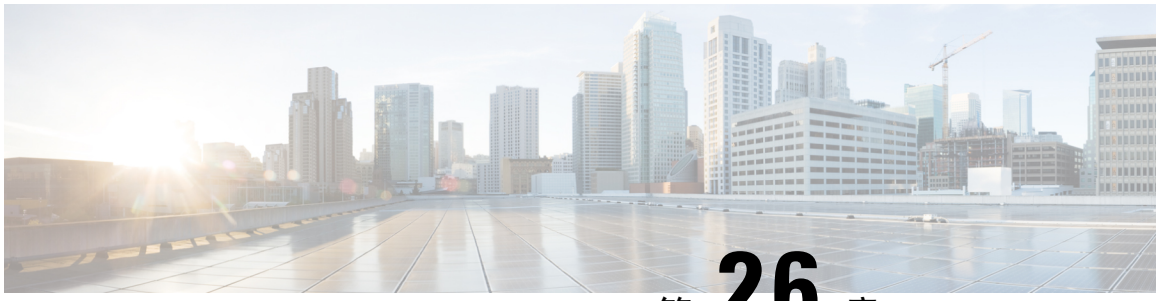
gNOI 機能には、次の注意事項と制約事項があります。

- 最大 16 のアクティブな gNOI RPC がサポートされます。
- Cisco Nexus 9000 シリーズ スイッチは、1 つの gNMI サービスと 2 つの gNOI マイクロサービスを持つ 1 つのエンドポイントを実行します。
- 10.1(1) リリースでは、gNOI RPC は同等の CLI を使用して実装されます。既存の CLI 制限または有効なオプションはそのまま適用されます。

gNOI の確認

gNOI の構成を確認するには、次のコマンドを入力します。

コマンド	説明
clear grpc gnoi rpc	カウンタまたは呼び出しをクリーンアップするために使用されます。
debug grpc events {events|errors} show grpc nxsdk event-history {events|errors}	イベント履歴からイベントとエラーをデバッグします。
show grpc internal gnoi rpc {summary|detail}	有用性のために「internal 」キーワード コマンドが追加されました。



第 26 章

インフラストラクチャの概要

モデル駆動型プログラマビリティについて

NX-OS デバイスのモデル駆動型プログラマビリティにより、デバイスの構成と制御を自動化できます。

データモデリング

データ モデリングでは、手作業の構成プロセスの代わりに、プログラムを使用した標準ベースの方法でネットワーク デバイスの構成を記述します。データ モデルは、業界で定義された標準言語で記述されます。CLI を使用した設定のほうがわかりやすいですが、データモデルを使用して設定を自動化すると拡張性が向上します。

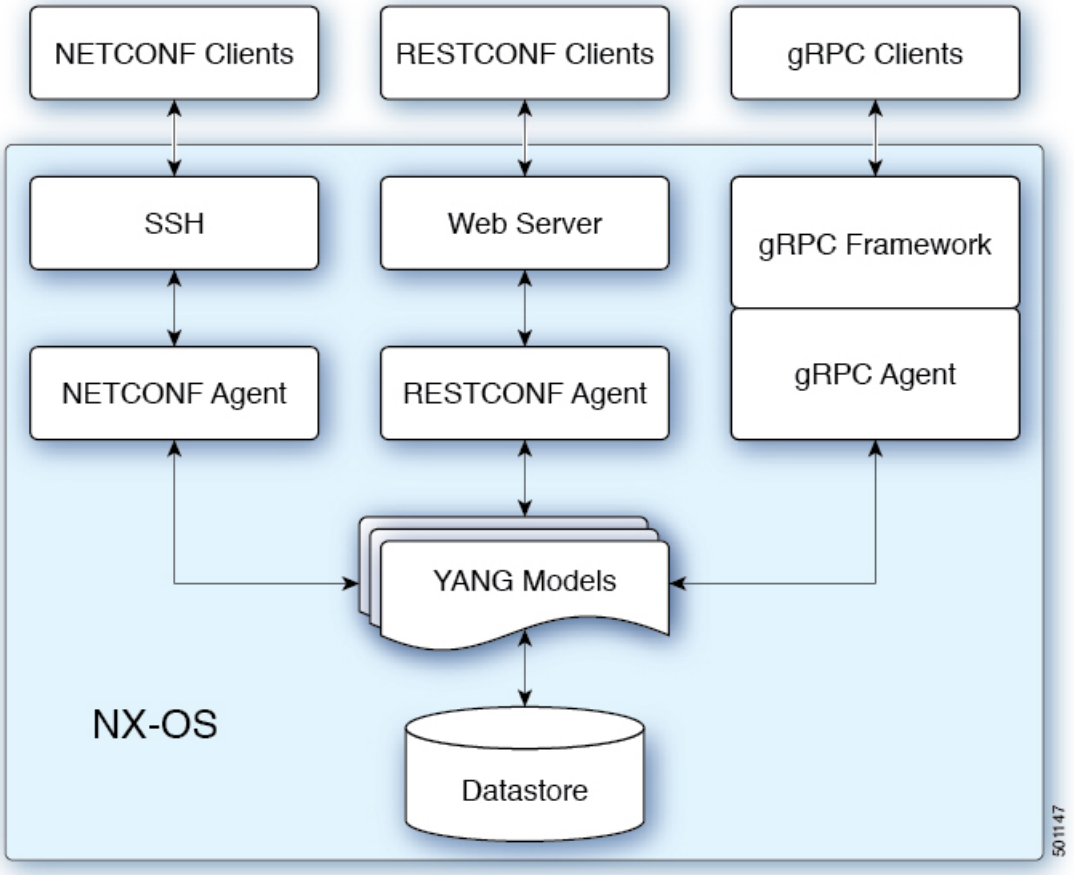
Cisco NX-OS デバイスは YANG データ モデリング言語をサポートしています。YANG は、ネットワーク デバイスの構成データと運用データ、リモート プロシージャ呼び出し、および通知を記述するために用いられるデータ モデリング 言語です。

プログラム可能なインターフェイス

NX-OS では、データ モデルでの操作のために 3 つの標準ベースのプログラマブル インターフェイス（NETCONF、RESTConf、および gRPC）がサポートされています。

プログラム可能なインターフェイス インフラストラクチャについて

ここでは、NX-OS プログラマブル インターフェイス インフラストラクチャの概要について説明します。



NETCONF、RESTConf または、 gRPC 経由でのいずれかの方法で受信されても要求は、抽象メッセージ オブジェクトに変換されます。そのメッセージオブジェクトは、要求内の名前空間に基づいて基盤となるモデルインフラストラクチャに配布されます。名前空間を使用すると、適切なモデルが選択され、要求は処理のために渡されます。モデルインフラストラクチャは、デバイスデータストアで要求（読み取りまたは書き込み）を実行します。結果は要求元のエージェントに返され、要求元のクライアントに応答が送信されます。

NX-OS プログラマブル インターフェイス エージェント

エージェントは、デバイスとクライアント間のインターフェイスを提供します。デバイスとの通信のトランスポート、プロトコル、およびエンコーディングを指定します。NX-OS プログラマブル インターフェイスは、NETCONF、RESTConf、および gRPC の 3 つのエージェントをサポートします。それぞれが YANG モデルを介してデバイスの構成管理に異なるインターフェイスを提供します。

（注） 各 Cisco NX-OS リリースでサポートされている YANG モデルについては、https://devhub.cisco.com/artifactory/open-nxos-agents を参照してください。

表 *24 : NX-OS* プログラマブル インターフェイス エージェント

Agent	トランスポート	Protocol	エンコーディング
NETCONF	SSH		XML
RESTConf	HTTP	draft-ietf-netconf-restconf-10[1]	XML または JSON
gRPC	HTTP	gRPC Protocol Spec[2]	Google Protobuf

プロトコルの仕様については、次のドキュメントを参照してください。

- [1] RESTCONF Protocol draft-ietf-netconf-restconf-10 https://tools.ietf.org/html/draft-ietf-netconf-restconf-10
- [2] Cisco NX-OS gRPC Protocol Specification

モデル インフラストラクチャ

モデルインフラストラクチャは、エージェントから受信した要求を受け取り、要求内のYANGモデルに関連付けられている名前空間を決定し、名前空間に一致するモデル コンポーネントを選択して要求を処理します。選択したモデル コンポーネントが要求処理を完了すると、処理結果が要求元のエージェントに送信され、クライアントに送り返されます。モデル インフラストラクチャは、エージェント プロトコルの指定に従って、認証、ハンドシェイクなどを含むプロトコル開始要求の処理も行います。

デバイス YANG モデル

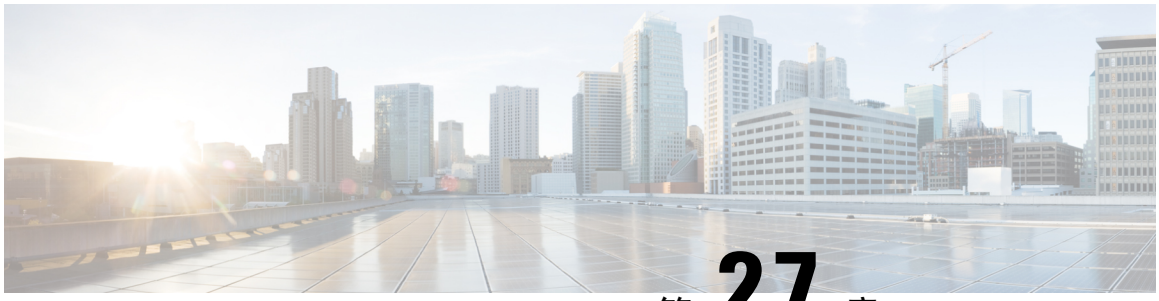
デバイス構成は、デバイスモデルと呼ばれるYANGモデルで記述されます。デバイス モデルは、デバイス名前空間を持つ別のモデルコンポーネントとしてモデルインフラストラクチャに明示されます。

共通 YANG モデル

共通モデルは、同等のデバイスモデル要素へのYANGパスを要素内に含む、別の種類のモデル コンポーネントです。これら同等のデバイス モデル要素は、デバイスYANGコンテキストでデバイス モデル データの読み取りと書き込みに使用されます。

その他の YANG 関連資料

YANG の詳細については、*YANG Central Wiki* https://handwiki.org/wiki/YANG（M. Bjorklund, Ed.）を参照してください。



第 27 章

コンポーネント管理

コンポーネント RPM パッケージについて

NX-OS プログラマブル インターフェイス コンポーネント RPM パッケージは、Cisco Artifactory からダウンロードできます。必要なコンポーネント RPM パッケージには、次の 2 つのタイプがあります。

- 基本コンポーネント（必須）
- 共通モデル コンポーネント（OpenConfig モデルは明示的にダウンロードしてインストールする必要があります）

基本コンポーネント

基本コンポーネントは、次の必要な RPM パッケージで構成されます。

- **mtx-infra**：インフラストラクチャ
- **mtx-device**：Cisco ネイティブ モデル

モデル化された NX-OS インターフェイスにアクセスするには、次のエージェント パッケージの少なくとも 1 つをインストールする必要があります。

- **mtx-netconf-agent**：NETCONF エージェント
- **mtx-restconf-agent**：RESTCONF エージェント
- **mtx-grpc-agent**：gRPC エージェント

共通モデル コンポーネント

共通モデル コンポーネント RPM は、OpenConfig モデルをサポートします。OpenConfig モデルを使用するには、OpenConfig RPM をダウンロードしてインストールする必要があります。利便性のため、サポートされているすべての OpenConfig モデルの単一の結合パッケージ、`mtx-openconfig-all` があります。

単一の結合パッケージを推奨していますが、代わりに、選択したモデルと、次の表に記載されているサポート モデル間の依存関係を満たすための RPM をダウンロードしてインストールすることもできます。`mtx-openconfig-all` RPM は、個々のモデル RPM と互換性がありません。後者をインストールする場合はその前に前者をアンインストールし、また逆に、前者をインストールする場合はその前に後者をアンインストールする必要があります。

モデル名	モデル リビジョン	モデル Ver	パッケージ名	依存関係
openconfig-acl	2017-05-26	1.0.0	mtx-openconfig-acl	mtx-openconfig-interfaces
openconfig-bgp-policy	2017-07-30	4.0.1	mtx-openconfig-bgp-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-if-aggregate	2017-07-14	2.0.0	mtx-openconfig-if-aggregate	mtx-openconfig-if-ethernet mtx-openconfig-interfaces
openconfig-if-ethernet	2017-07-14	2.0.0	mtx-openconfig-if-ethernet	mtx-openconfig-interfaces
openconfig-if-ip	2016-05-26	1.0.2	mtx-openconfig-if-ip	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-if-ip-ext	2018-01-05	2.3.0	mtx-openconfig-if-ip-ext	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-if-ip mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-interfaces	2017-07-14	2.0.0	mtx-openconfig-interfaces	-
openconfig-network-instance	2017-08-24	0.8.1	mtx-openconfig-network-instance	mtx-openconfig-bgp-policy mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-routing-policy mtx-openconfig-vlan

モデル名	モデル リビジョン	モデル Ver	パッケージ名	依存関係
openconfig-network-instance-policy	2017-02-15	0.1.0	mtx-openconfig-network-instance-policy	mtx-openconfig-routing-policy
openconfig-ospf-policy	2017-08-24	0.1.1	mtx-openconfig-ospf-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-platform	2018-01-16	0.8.0	mtx-openconfig-platform	-
openconfig-platform-linecard	2017-08-03	0.1.0	mtx-openconfig-platform-linecard	mtx-openconfig-platform
openconfig-platform-port	2018-01-20	0.3.0	mtx-openconfig-platform-port	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-platform-transceiver	2018-01-22	0.4.1	mtx-openconfig-platform-transceiver	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-relay-agent	2016-05-16	0.1.0	mtx-openconfig-relay-agent	mtx-openconfig-interfaces
openconfig-routing-policy	2016-05-12	2.0(1)	mtx-openconfig-routing-policy	-
openconfig-spanning-tree	2017-07-14	0.2.0	mtx-openconfig-spanning-tree	mtx-openconfig-interfaces
openconfig-system	2017-09-18	0.3.0	mtx-openconfig-system	-
openconfig-vlan	2017-07-14	2.0.0	mtx-openconfig-vlan	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces

インストールの準備

ここでは、インストールの準備と、NX-OS プログラマブル インターフェイス コンポーネントを管理するためのその他の有用な情報について説明します。

デバイスで Bash シェルを開く

スイッチへの RPM のインストールは、Bash シェルで実行されます。デバイスで機能 **feature bash** が構成されていることを確認します。

```
Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# feature bash-shell
Switch(config)# end
Switch# run bash sudo su
```

```
bash-4.2#
```

Bash からデバイスの CLI プロンプトに戻るには、**exit** または **Ctrl-D** を入力します。

デバイスの準備状況の確認

RPM をインストールする前に、次の CLI **show** コマンドを使用してデバイスの準備状況を確認できます。

- `show module`: すべてのモジュールが稼働しているかどうかを示します。

  ```
  Switch# show module
  ```

- `show system redundancy status`: スタンバイ デバイスが稼働しており、HA モードになっているかどうかを示します。スタンバイ同期が進行中の場合、RPMのインストールが失敗する可能性があります。

  ```
  Switch# show system redundancy status
  ```

ラインカードが起動しない場合は、Bash シェルで **createrepo /rpms** コマンドを入力します。

```
bash-4.2# createrepo /rpms
```

Cisco Artifactory からのコンポーネントのダウンロード

NX-OS プログラマブル インターフェイス コンポーネント RPM は、次の URL の Cisco Artifactory からダウンロードできます。RPMは、NX-OS リリースごとのディレクトリに整理されています。正しい NX-OS リリース ディレクトリから RPM をダウンロードしていることを確認してください。

https://devhub.cisco.com/artifactory/open-nxos-agents

NX-OS プログラマブル インターフェイス コンポーネント RPM は、次の命名規則に従っています。

<package>-<version>-<NX-OS release>.<architecture>.**rpm**

次のセクションの説明に従って、目的のインストールする NX-OS プログラマブル インターフェイス コンポーネント RPM パッケージを選択して、デバイスにダウンロードします。

RPM パッケージのインストール

プログラマブル インターフェイス ベースおよび共通モデル コンポーネント RPM パッケージのインストール

始める前に

- Cisco Artifactory から、次のパッケージをダウンロードします。

- mtx-infra
- mtx-device
- mtx-netconf-agent/mtx-restconf-agent/mtx-grpc-agent（少なくとも 1 つ）
- mtx-openconfig-all（または、選択した個々のモデル）

- デバイスの準備状況の確認（364 ページ）の CLI コマンドを使用して、アクティブ デバイスとスタンバイ デバイスのすべてのライン カードが稼働しており、準備ができていることを確認します。

Step 1 ダウンロードした RPM をデバイスにコピーします。

例：

```
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm bootflash: vrf
management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm bootflash: vrf
 management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm bootflash:
 vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm bootflash:
 vrf management
```

Step 2 Bash シェルから、RPM をインストールします。

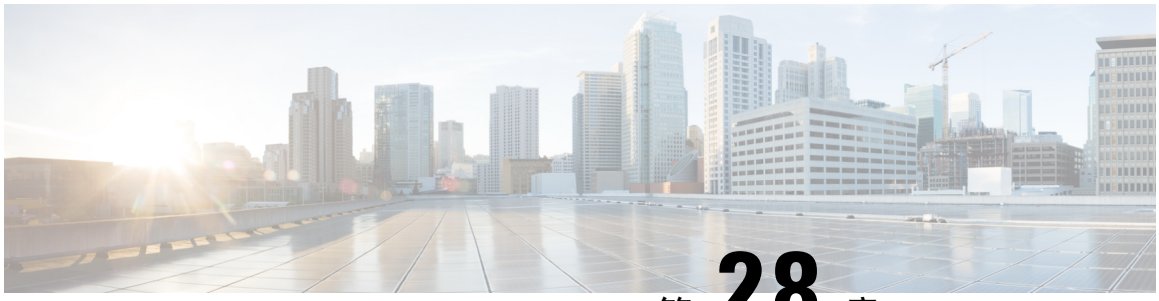
例：

```
bash-4.2# cd /bootflash
bash-4.2# dnf install mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm
 mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
```

Step 3 Bash シェルから、インストールを確認します。

例：

```
bash-4.2# dnf list installed | grep mtx
```



第 28 章

モデル駆動型テレメトリ

テレメトリについて

分析やトラブルシューティングのためのデータ収集は、ネットワークの健全性をモニタリングする上で常に重要な要素であり続けています。

Cisco NX-OS は、ネットワークからデータを収集するための、SNMP、CLI や Syslog といった複数のメカニズムを提供します。これらのメカニズムには、自動化や拡張に対する制約があります。ネットワーク要素からのデータの最初の要求がクライアントから出された場合、プル モデルの使用が制限されることもその制約の 1 つです。プル モデルは、ネットワーク内に複数のネットワーク管理ステーション（NMS）がある場合は拡張しません。このモデルを使用すると、クライアントが要求した場合に限り、サーバーがデータを送信します。このような要求を開始するには、手動による介入を続けて行う必要があります。このような手動による介入を続けると、プル モデルの効率が失われます。

プッシュ モデルは、ネットワークからデータを継続的にストリーミングし、クライアントに通知します。テレメトリはプッシュ モデルをイネーブルにし、モニタリング データにほぼリアルタイムでアクセスできるようにします。

テレメトリ コンポーネントとプロセス

テレメトリは、次の 4 つの主要な要素で構成されます。

- **データ収集**：テレメトリ データは、識別名（DN）パスを使用して指定されたオブジェクト モデルのブランチにあるデータ管理エンジン（DME）データベースから収集されます。データは定期的に取得されるか（頻度ベース）、指定したパスのオブジェクトで変更があった場合にのみ取得できます（イベント ベース）。NX-API を使用して、頻度ベースのデータを収集できます。

- **データ エンコーディング**：テレメトリ エンコーダが、収集されたデータを目的の形式で転送できるようにカプセル化します。

  NX-OS は、テレメトリ データを Google Protocol Buffers（GPB）および JSON 形式でエンコードします。

- **データ トランスポート**：NX-OS は、JSON エンコードに HTTP を使用してテレメトリ データを転送し、GPB エンコードに Google リモート プロシージャ コール (gRPC) プロトコルを使用します。gRPC レシーバーは、4 MB を超えるメッセージ サイズをサポートします。（証明書が構成されている場合は、HTTPS を使用したテレメトリ データもサポートされます。）

  Cisco NX-OS リリース 9.2(1) 以降、テレメトリは IPv6 接続先および IPv4 接続先へのストリーミングをサポートするようになりました。

  次のコマンドを使用して、JSON または GPB のデータグラム ソケットを使用してデータをストリーミングするように UDP トランスポートを構成します。

```
destination-group num
  ip address xxx.xxx.xxx.xxx port xxxx protocol UDP encoding {JSON | GPB }
```

  IPv6 接続先の例:

```
destination-group 100
  ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB
```

  UDP テレメトリには次のヘッダーがあります。

```
typedef enum tm_encode_ {
  TM_ENCODE_DUMMY,
  TM_ENCODE_GPB,
  TM_ENCODE_JSON,
  TM_ENCODE_XML,
  TM_ENCODE_MAX,
} tm_encode_type_t;

typedef struct tm_pak_hdr_ {
  uint8_t version; /* 1 */
  uint8_t encoding;
  uint16_t msg_size;
  uint8_t secure;
  uint8_t padding;
}__attribute__ ((packed, aligned (1))) tm_pak_hdr_t;
```

  次のいずれかの方法で、ペイロードの最初の 6 バイトを使用して、UDP を使用してテレメトリ データを処理します。

  - 受信側が複数のエンドポイントから異なるタイプのデータを受信することになっている場合は、ヘッダーの情報を読んで、データのデコードに使用するデコーダー（JSON または GPB）を決定します。

- 1 つのデコーダー（JSON または GPB）が必要で、もう 1 つのデコーダーは必要ない場合は、ヘッダーを削除します。

- **テレメトリ レシーバー**：テレメトリ レシーバーは、テレメトリ データを保存するリモート管理システムです。

GPB エンコーダーは、汎用キーと値の形式でデータを格納します。また、データを GPB 形式に変換するには、コンパイルされた `.proto` ファイル形式のメタデータが GPB エンコーダに必要です。

データ ストリームを正しく受信してデコードするには、受信側でエンコードとトランスポート サービスを記述した `.proto` ファイルが必要です。エンコードは、バイナリ ストリームをキー値の文字列のペアにデコードします。

GPB エンコーディングと gRPC トランスポートを記述する telemetry `.proto` ファイルは、Cisco の GitLab で入手できます。 https://github.com/CiscoDevNet/nx-telemetry-proto

テレメトリ プロセスの高可用性

テレメトリ プロセスの高可用性は、次の動作でサポートされています。

- **[システムのリロード（System Reload）]** — システムのリロード中に、テレメトリ構成とストリーミング サービスが復元されます。
- **[スーパーバイザ フェールオーバー（Supervisor Failover）]** — テレメトリはホット スタンバイではありませんが、テレメトリ構成とストリーミング サービスは、新しい現用系スーパーバイザが実行されているときに復元されます。
- **[プロセスの再起動（Process Restart）]** — なんらかの理由でテレメトリ プロセスがフリーズまたは再起動した場合、テレメトリが再開されると、構成およびストリーミング サービスが復元されます。

テレメトリのライセンス要件

製品	ライセンス要件
Cisco NX-OS	テレメトリにはライセンスは必要ありません。ライセンス パッケージに含まれていない機能は Cisco NX-OS イメージにバンドルされており、無料で提供されます。NX-OS ライセンス方式の詳細については、『*Cisco NX-OS Licensing Guide*』を参照してください。

Guidelines and Limitations

Telemetry has the following configuration guidelines and limitations:

- For information about supported platforms, see the Nexus Switch Platform Matrix.

- Cisco NX-OS releases that support the data management engine (DME) Native Model support Telemetry.
- Support is in place for the following:
  - DME data collection
  - NX-API data sources
  - Google protocol buffer (GPB) encoding over Google Remote Procedure Call (gRPC) transport
  - JSON encoding over HTTP
- The smallest sending interval (cadence) supported is five seconds for a depth of 0. The minimum cadence values for depth values greater than 0 depends on the size of the data being streamed out. Configuring any cadences below the minimum value may result in undesirable system behavior.
- Telemetry supports up to five remote management receivers (destinations). Configuring more than five remote receivers may result in undesirable system behavior.
- Telemetry can consume up to 20% of the CPU resource.

Configuration Commands After Downgrading to an Older Release

After a downgrade to an older release, some configuration commands or command options can fail because the older release may not support them. When downgrading to an older release, unconfigure and reconfigure the telemetry feature after the new image comes up. This sequence avoids the failure of unsupported commands or command options.

The following example shows this procedure:

- Copy the telemetry configuration to a file:

```
switch# show running-config | section telemetry
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch# show running-config | section telemetry > telemetry_running_config
switch# show file bootflash:telemetry_running_config
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch#
```

- Execute the downgrade operation. When the image comes up and the switch is ready, copy the telemetry configurations back to the switch.

```
switch# copy telemetry_running_config running-config echo-commands
`switch# config terminal`
`switch(config)# feature telemetry`
`switch(config)# telemetry`
`switch(config-telemetry)# destination-group 100`
`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB `
`switch(conf-tm-dest)# sensor-group 100`
`switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
`switch(conf-tm-sensor)# subscription 600`
`switch(conf-tm-sub)# dst-grp 100`
`switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000`
`switch(conf-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete.
switch#
```

gRPC Error Behavior

The switch client disables the connection to the gRPC receiver if the gRPC receiver sends 20 errors. Unconfigure then reconfigure the receiver's IP address under the destination group to enable the gRPC receiver. Errors include:

- The gRPC client sends the wrong certificate for secure connections.
- The gRPC receiver takes too long to handle client messages and incurs a timeout. Avoid timeouts by processing messages using a separate message processing thread.

Support for gRPC Chunking

Starting with Release 9.2(1), support for gRPC chunking has been added. For streaming to occur successfully, you must enable chunking if gRPC has to send an amount of data greater than 12 MB to the receiver.

The gRPC user must do the gRPC chunking. The gRPC client side does the fragmentation, and the gRPC server side does the reassembly. Telemetry is still bound to memory and data can be dropped if the memory size is more than the allowed limit of 12 MB for telemetry. In order to support chunking, use the telemetry `.proto` file that is available at Cisco's GibLab, which has been updated for gRPC chunking, as described in テレメトリ コンポーネントとプロセス, on page 367.

The chunking size is from 64 through 4096 bytes.

Following shows a configuration example through the NX-API CLI:

```
feature telemetry
!
telemetry
  destination-group 1
    ip address 171.68.197.40 port 50051 protocol gRPC encoding GPB
    use-chunking size 4096
  destination-group 2
    ip address 10.155.0.15 port 50001 protocol gRPC encoding GPB
    use-chunking size 64
  sensor-group 1
    path sys/intf depth unbounded
  sensor-group 2
    path sys/intf depth unbounded
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 10000
  subscription 2
```

```
    dst-grp 2
    snsr-grp 2 sample-interval 15000
```

Following shows a configuration example through the NX-API REST:

```
{
    "telemetryDestGrpOptChunking": {
        "attributes": {
            "chunkSize": "2048",
            "dn": "sys/tm/dest-1/chunking"
        }
    }
}
```

The following error message appears on systems that do not support gRPC chunking, such as the Cisco MDS series switches:

```
MDS-9706-86(conf-tm-dest)# use-chunking size 200
ERROR: Operation failed: [chunking support not available]
```

NX-API Sensor Path Limitations

NX-API can collect and stream switch information not yet in the DME using **show** commands. However, using the NX-API instead of streaming data from the DME has inherent scale limitations as outlined:

- The switch backend dynamically processes NX-API calls such as **show** commands,
- NX-API spawns several processes that can consume up to a maximum of 20% of the CPU.
- NX-API data translates from the CLI to XML to JSON.

The following is a suggested user flow to help limit excessive NX-API sensor path bandwidth consumption:

1. Check whether the **show** command has NX-API support. You can confirm whether NX-API supports the command from the VSH with the pipe option: `show <command> | json` or `show <command> | json pretty`.

Note Avoid commands that take the switch more than 30 seconds to return JSON output.

2. Refine the **show** command to include any filters or options.
   - Avoid enumerating the same command for individual outputs; for example, **show vlan id 100** , **show vlan id 101** , and so on. Instead, use the CLI range options; for example, **show vlan id 100-110,204** , whenever possible to improve performance.

     If only the summary or counter is needed, then avoid dumping a whole show command output to limit the bandwidth and data storage that is required for data collection.

3. Configure telemetry with sensor groups that use NX-API as their data sources. Add the **show** commands as sensor paths
4. Configure telemetry with a cadence of five times the processing time of the respective **show** command to limit CPI usage.
5. Receive and process the streamed NX-API output as part of the existing DME collection.

Telemetry VRF Support

Telemetry VRF support allows you to specify a transport VRF, which means that the telemetry data stream can egress through front-panel ports and avoid possible competition between SSH or NGINX control sessions.

You can use the **use-vrf** *vrf-name* command to specify the transport VRF.

The following example specifies the transport VRF:

The following is an example of use-vrf as a POST payload:

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptVrf": {
                      "attributes": {
                        "name": "default"
                      }
                    }
                  }
                ]
              }
            }
```

Certificate Trustpoint Support

Beginning in NX-OS release 10.1(1), the **trustpoint** keyword is added in the existing global level command.

The following is the command syntax:

```
switch(config-telemetry)# certificate ?
trustpoint    specify trustpoint label
WORD          .pem certificate filename (Max Size 256)
switch(config-telemetry)# certificate trustpoint
WORD          trustpoint label name (Max Size 256)
switch(config-telemetry)# certificate trustpoint trustpoint1 ?
WORD  Hostname associated with certificate (Max Size 256)
switch(config-telemetry)#certificate trustpoint trustpoint1 foo.test.google.fr
```

Destination Hostname Support

Beginning in NX-OS release 10.1(1), the **host** keyword is added in destination-group command.

The following is the example for the destination hostname support:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ?
certificate Specify certificate
host Specify destination host
ip Set destination IPv4 address
ipv6 Set destination IPv6 address
...
switch(conf-tm-dest)# host ?
A.B.C.D|A:B::C:D|WORD  IPv4 or IPv6 address or DNS name of destination
switch(conf-tm-dest)#
```

```
switch(conf-tm-dest)# host abc port 11111 ?
protocol  Set transport protocol
switch(conf-tm-dest)# host abc port 11111 protocol ?
HTTP
```

```
UDP
gRPC
switch(conf-tm-dest)# host abc port 11111 protocol gRPC ?
encoding  Set encoding format
switch(conf-tm-dest)# host abc port 11111 protocol gRPC encoding ?
Form-data    Set encoding to Form-data only
GPB          Set encoding to GPB only
GPB-compact  Set encoding to Compact-GPB only
JSON         Set encoding to JSON
XML          Set encoding to XML
switch(conf-tm-dest)# host ip address 1.1.1.1 port 2222 protocol HTTP encoding JSON
<CR>
```

Support for Node ID

Beginning in NX-OS release 10.1(1), you can configure a custom Node ID string for a telemetry receiver through the **use-nodeid** command. By default, the host name is used, but support for a node ID enables you to set or change the identifier for the `node_id_str` of the telemetry receiver data.

You can assign the node ID through the telemetry destination profile, by using the **usenode-id** command. This command is optional.

The following example shows configuring the node ID.

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(conf-tm-dest-profile)# use-nodeid test-srvr-10
switch(conf-tm-dest-profile)#
```

The following example shows a telemetry notification on the receiver after the node ID is configured.

```
Telemetry receiver:
==================================
node_id_str: "test-srvr-10"
subscription_id_str: "1"
encoding_path: "sys/ch/psuslot-1/psu"
collection_id: 3896
msg_timestamp: 1559669946501
```

Use the **use-nodeid** sub-command under the **host** command. The destination level **use-nodeid** configuration preceeds the global level configuration.

The following example shows the command syntax:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# use-nodeid ?
WORD  Node ID (Max Size 128)
switch(conf-tm-dest-host)# use-nodeid session_1:18112
```

The following example shows the output from the Telemetry receiver:

```
>> Message size 923
Telemetry msg received @  23:41:38 UTC
    Msg Size: 11
    node_id_str   : session_1:18112
    collection_id : 3118
    data_source   : DME
    encoding_path : sys/ch/psuslot-1/psu
    collection_start_time   : 1598485314721
    collection_end_time   : 1598485314721
    data          :
```

Support for Streaming of YANG Models

Beginning in NX-OS release 9.2(1), telemetry supports the YANG ("Yet Another Next Generation") data modeling language. Telemetry supports data streaming for both device YANG and OpenConfig YANG.

For more information on the YANG data modeling language, see インフラストラクチャの概要 and RESTCONF エージェント.

Support for Proxy

Beginning in NX-OS release 10.1(1), the **proxy** command is included in the host command. The following is the command syntax:

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# proxy ?
    A.B.C.D|A:B::C:D|WORD  IPv4 or IPv6 address or DNS name of proxy server
    <1-65535>  Proxy port number, Default value is 8080
username  Set proxy authentication username
password  Set proxy authentication password
```

gRPC Asynchronous Mode

The gRPC asynchronous mode is available only under the **host** command. In normal stream condition, this mode allows the receivers to stream data in **mdtDialout** call without exiting or receiving **WriteDone()** call.

The following is the command syntax:

```
nxosv-1(config-telemetry)# destination-group 1
nxosv-1(conf-tm-dest)# host 172.22.244.130 port 50007 ?
nxosv-1(conf-tm-dest-host)# grpc-async ?
```

CLI を使用したテレメトリの構成

NX-OS CLI を使用したテレメトリの構成

次の手順では、ストリーミングテレメトリを有効にし、データストリームの送信元と接続先を構成します。

手順の概要

1. **configure terminal**
2. **feature telemetry**
3. **feature nxapi**
4. **nxapi use-vrf management**
5. **telemetry**
6. （任意） **certificate** *certificate_path* *host_URL*
7. **sensor-group** *sgrp_id*
8. **path** *sensor_path* **depth** **unbounded** [**filter-condition** *filter*] [**alias** *path_alias*]
9. **destination-group** *dgrp_id*

10. （任意）**ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
11. （任意）**ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
12. *ip_version* **address** *ip_address* **port** *portnum*
13. （任意）**use-chunking size** *chunking_size*
14. **subscription** *sub_id*
15. **snsr-grp** *sgrp_id* **sample-interval** *interval*
16. **dst-grp** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		グローバル構成モードを開始します。

Step 2	**feature telemetry**	ストリーミング テレメトリ機能を有効にします。
Step 3	**feature nxapi**	NX-API を有効にします。

Step 4	**nxapi use-vrf management**
	例：
	`switch(config)#`
	`switch(config)# nxapi use-vrf management`
	`switch(config)#`
		NX-API 通信に使用する VRF 管理を有効にします。
		（注） ACL はネットスタックパケットのみをフィルタ処理できるため、10.2（3）F より前のリリースでは次の意味の警告が表示されます：
		「警告：構成された管理 ACL は、HTTP サービスでは有効になりません。iptables を使用してアクセスを制限してください」。
		（注） 10.2(3)F 以降、ACL は、管理 vrf に着信する netstack パケットと kstack パケットの両方をフィルタリングできます。次の意味の警告が表示されます：
		「警告：非管理 VRF で構成された ACL は、その VRF の HTTP サービスでは有効になりません」。

Step 5	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		ストリーミング テレメトリの構成モードに入ります。

	コマンドまたはアクション	目的
Step 6	（任意） **certificate** *certificate_path host_URL* 例： `switch(config-telemetry)# certificate /bootflash/server.key localhost`	既存の SSL/TLS 証明書を使用します。 EOR デバイスの場合、証明書もスタンバイ SUP にコピーする必要があります。

Step 7	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 100`
	`switch(conf-tm-sensor)#`
		ID *srgp_id* を持つセンサー グループを作成し、センサー グループ構成モードを開始します。
		現在は、数字の ID 値のみサポートされています。センサー グループでは、テレメトリ レポートのモニタリング対象ノードを定義します。

Step 8	**path** *sensor_path* **depth unbounded** [**filter-condition** *filter*] [**alias** *path_alias*]
	例：
	• 次のコマンドは、NX-API ではなく、DME または YANG に適用されます：
	`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(l2BD.operSt, "down")`
	以下の構文を使用し、状態ベースのフィルタリングを使用して、**operSt** が **up** から **down** に変化したときにのみトリガーするようにします。MO が変化しても通知しません。
	`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))`
	UTR 側のパスを区別するには、次の構文を使用します。
	`switch(conf-tm-sensor)# path sys/ch/ftslot-1/ft alias ft_1`
	• 次のコマンドは、DME ではなく、NX-API または YANG に適用されます：
	`switch(conf-tm-sensor)# path "show interface" depth 0`
	• 次のコマンドは、デバイス YANG に適用されます：
	`switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items`
	• 次のコマンドは、OpenConfig YANG に適用されます：
		ここでの制限なしとは、出力に子管理対象オブジェクト（MO）を含めることを意味します。したがって、POLL テレメトリ ストリームの場合、そのパスと EVENT のすべての子 MO が子 MO で行われた変更を取得します。
		（注） これは、データ送信元 DME パスにのみ適用されます。
		センサー グループにセンサー パスを追加します。
		• Cisco NX-OS 9.3(5) リリース以降では、**alias** キーワードが導入されています。
		• **depth** 設定では、センサー パスの取得レベルを指定します。**0 - 32**、**unbounded** の深さ設定がサポートされています。
		（注） **depth 0** デフォルトの深さです。
		NX-API ベースのセンサー パスは、**depth 0** のみを使用できます。
		イベント収集のパスがサブスクライブされている場合、深さは 0 とバウンドなしのみをサポートします。その他の値は 0 として扱われます。
		• オプションの **filter-condition** パラメータを指定して、イベントベースのサブスクリプション用の特定のフィルタを作成できます。
		状態ベースのフィルタ処理の場合、フィルタ処理は、状態が変化したときと、指定された状態

	コマンドまたはアクション	目的
	switch(conf-tm-sensor)# **path openconfig-bgp:bgp** switch(conf-tm-sensor)# **path Cisco-NX-OS-device:System/bgp-items/inst-items alias bgp_alias** • 次のコマンドは、NX-API に適用されます： switch(conf-tm-sensor)# **path "show interface" depth 0 alias sh_int_alias** • 次のコマンドは、OpenConfig に適用されます： switch(conf-tm-sensor)# **path openconfig-bgp:bgp alias oc_bgp_alias**	でイベントが発生したときの両方を返します。つまり、**eq(l2Bd.operSt, "down")** の DN **sys/bd/bd-[vlan]** のフィルタ条件は、operSt が変更されたとき、および operSt が **down** である間に DN のプロパティが変更されたとき（VLAN が動作上 **down** である間に **no shutdown** コマンドが発行された場合など）にトリガーされます。 • YANG モデルの場合、センサー パスのフォーマットは *module_name* : *YANG_path* です。*module_name* は YANG モデル ファイルの名前です。次に例を示します。 • デバイス YANG の場合： **Cisco-NX-OS-device:System/bgp-items/inst-items** • OpenConfig YANG の場合： **openconfig-bgp:bgp** （注） **depth**、**filter-condition**、および **query-condition** パラメータは、現在 YANG ではサポートされていません。 openconfig YANG モデルの場合は、https://github.com/YangModels/yang/tree/master/vendor/cisco/nx に移動して、最新リリースの適切なフォルダに移動します。 特定のモデルをインストールする代わりに、すべての OpenConfig モデルを含む openconfig-all RPM をインストールできます。 次に例を示します。 **install add mtx-openconfig-bgp-1.0.0.0.0-7.0.3.IHD8.1.lib32_n9000.rpm activate**
Step 9	**destination-group** *dgrp_id* 例： switch(conf-tm-sensor)# **destination-group 100** switch(conf-tm-dest)#	接続先グループを作成して、接続先グループ構成モードを開始します。 現在、*dgrp_id* は、数字の ID 値のみをサポートしています。

	コマンドまたはアクション	目的

Step 10	（任意） **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	例：
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON`
		エンコードされたテレメトリ データを受信する IPv4 IP アドレスとポートを指定します。
		（注） gRPC はデフォルトのトランスポート プロトコルです。
		GPB がデフォルトのエンコーディングです。

Step 11	（任意） **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	例：
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON`
		エンコードされたテレメトリ データを受信する IPv6 IP アドレスとポートを指定します。
		（注） gRPC はデフォルトのトランスポート プロトコルです。
		GPB がデフォルトのエンコーディングです。

Step 12	*ip_version* **address** *ip_address* **port** *portnum*
	例：
	• IPv4 の場合：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`
	• IPv6 の場合：
	`switch(conf-tm-dest)# ipv6 address 10:10::1 port 8000`
		発信データの接続先プロファイルを作成します。ここで *ip_version* は、ip（IPv4 の場合）または ipv6（IPv6 の場合）のいずれかです。
		接続先グループがサブスクリプションにリンクされている場合、テレメトリ データは、このプロファイルで指定されている IP アドレスとポートに送信されます。

Step 13	（任意） **use-chunking size** *chunking_size* 例： `switch(conf-tm-dest)# use-chunking size 64`	gRPC チャンクを有効にして、チャンク サイズを 64～4096 バイトに設定します。詳細については、「gRPC チャンクのサポート」セクションを参照してください。

Step 14	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 100`
	`switch(conf-tm-sub)#`
		ID を持つサブスクリプション ノードを作成し、サブスクリプション構成モードを開始します。
		現在、*sub_id* は、数字の ID 値のみをサポートしています。
		（注） DN にサブスクライブする場合は、イベントが確実にストリーミングされるように、その DN が REST を使用して DME でサポートされているかどうかを確認します。

	コマンドまたはアクション	目的
Step 15	**snsr-grp** *sgrp_id* **sample-interval** *interval* 例： `switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000`	ID *sgrp_id* のセンサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。 間隔の値が 0 の場合、イベントベースのサブスクリプションが作成され、テレメトリ データは、指定された MO での変更時にのみ送信されます。0 より大きい間隔値の場合、テレメトリ データが指定された間隔で定期的に送信される頻度に基いたサブスクリプションが作成されます。たとえば、間隔値が 15000 の場合、テレメトリ データは 15 秒ごとに送信されます。
Step 16	**dst-grp** *dgrp_id* 例： `switch(conf-tm-sub)# dst-grp 100`	ID *dgrp_id* を持つ接続先グループをこのサブスクリプションにリンクします。

YANG パスの頻度の設定

YANG パスの頻度は、合計ストリーミング時間よりも長くする必要があります。合計ストリーミング時間と頻度が正しく構成されていない場合、テレメトリ データの収集にストリーミング間隔よりも長くかかることがあります。この状況では、次のことがわかります。

- テレメトリ データが受信側へのストリーミングよりも速く蓄積されるため、徐々に満たされるキュー。
- 現在の間隔からではない古いテレメトリ データ。

合計ストリーミング時間よりも大きい値に頻度を構成します。

手順の概要

1. **show telemetry control database sensor-groups**
2. **sensor group** *number*
3. **subscription** *number*
4. **snsr-grp** *number* **sample-interval** *milliseconds*
5. **show system resources**

手順の詳細

	コマンドまたはアクション	目的
Step 1	**show telemetry control database sensor-groups** 例：	合計ストリーミング時間を計算します。 合計ストリーミング時間は、各センサー グループの個々の現在のストリーミング時間の合計です。個々

	コマンドまたはアクション	目的
	switch# **show telemetry control database sensor-groups** Sensor Group Database size = 2 --- Row ID Sensor Group ID Sensor Group type Sampling interval(ms) Linked subscriptions SubID --- 1 2 Timer /YANG 5000 /Running 1 1 Collection Time in ms (Cur/Min/Max): 2444/2294/2460 Encoding Time in ms (Cur/Min/Max): 56/55/57 Transport Time in ms (Cur/Min/Max): 0/0/1 **Streaming Time in ms (Cur/Min/Max): 2515/2356/28403** Collection Statistics: collection_id_dropped = 0 last_collection_id_dropped = 0 drop_count = 0 2 1 Timer /YANG 5000 /Running 1 1 Collection Time in ms (Cur/Min/Max): 144/142/1471 Encoding Time in ms (Cur/Min/Max): 0/0/1 Transport Time in ms (Cur/Min/Max): 0/0/0 **Streaming Time in ms (Cur/Min/Max): 149/147/23548** Collection Statistics: collection_id_dropped = 0 last_collection_id_dropped = 0 drop_count = 0 switch# telemetry destination-group 1 ip address 192.0.2.1 port 9000 protocol HTTP encoding JSON sensor-group 1 data-source YANG path /Cisco-NX-OS-device:System/procsys-items depth unbounded sensor-group 2 data-source YANG path /Cisco-NX-OS-device:System/intf-items/phys-items depth unbounded subscription 1 dst-grp 1 snsr-grp 1 sample-interval 5000 **snsr-grp 2 sample-interval 5000**	のストリーミング時間は、ミリ秒単位のストリーミング時間（Cur）に表示されます。この例では、合計ストリーミング時間は 2.664 秒（2515 ミリ秒 + 149 ミリ秒）です。 構成された頻度をセンサー グループの合計ストリーミング時間と比較します。 頻度はサンプル間隔で表示されます。この例では、合計ストリーミング時間（2.664 秒）がケイデンス（デフォルトの 5.000 秒）よりも短いため、頻度は正しく構成されています。
Step 2	**sensor group** *number* 例： switch(config-telemetry)# **sensor group*1***	合計ストリーミング時間がその頻度以上の場合、間隔を設定したいセンサーグループを入力します。
Step 3	**subscription** *number* 例： switch(conf-tm-sensor)# **subscription *100***	センサー グループのサブスクリプションを編集します。

	コマンドまたはアクション	目的
Step 4	**snsr-grp** *number* **sample-interval** *milliseconds* 例： `switch(conf-tm-sub)# snsr-grp number sample-interval 5000`	適切なセンサー グループについて、サンプル間隔を合計ストリーミング時間よりも大きい値に設定します。 この例では、サンプル間隔は5.000秒に設定されています。これは、2.664秒の合計ストリーミング時間よりも長いため、有効です。

Step 5	**show system resources**
	例：
	`switch# show system resources`
	`Load average: 1 minute: 0.38 5 minutes: 0.43 15 minutes: 0.43`
	`Processes: 555 total, 3 running`
	`CPU states : 24.17% user, 4.32% kernel, 71.50% idle`
	`CPU0 states: 0.00% user, 2.12% kernel, 97.87% idle`
	`CPU1 states: 86.00% user, 11.00% kernel, 3.00% idle`
	`CPU2 states: 8.08% user, 3.03% kernel, 88.88% idle`
	`CPU3 states: 0.00% user, 1.02% kernel, 98.97% idle`
	`Memory usage: 16400084K total, 5861652K used, 10538432K free`
	`Current memory status: OK`
		CPUの使用状況を確認してください。
		この例に示すように、CPU ユーザー状態が高い使用率を示している場合、頻度とストリーミング値が正しく構成されていません。この手順を繰り返して、頻度を正しく設定します。

CLI を使用したテレメトリの構成例

次の手順では、GPB エンコーディングを使用して 10 秒のリズムで単一のテレメトリ DME ストリームを構成する方法について説明します。

```
switch# configure terminal
switch(config)# feature telemetry
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(config-tm-dest)# ip address 171.70.59.62 port 50051 protocol gRPC encoding GPB
switch(config-tm-dest)# exit
switch(config-telemetry)# sensor group sg1
switch(config-tm-sensor)# data-source DME
switch(config-tm-dest)# path interface depth unbounded query-condition keep-data-type
switch(config-tm-dest)# subscription 1
switch(config-tm-dest)# dst-grp 1
switch(config-tm-dest)# snsr grp 1 sample interval 10000
```

この例では、sys/bgp ルート MO のデータを宛先 IP 1.2.3.4 ポート 50003 に 5 秒ごとにストリーミングするサブスクリプションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
```

```
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

次に、sys/intf のデータを 5 秒ごとに、宛先 IP 1.2.3.4 ポート 50003 にストリーミングし、test.pem を使用して検証された GPB エンコーディングを使用してストリームを暗号化するサブスクリプションの作成例を示します。

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(config-dest)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

この例では、sys/cdp のデータを接続先 IP 1.2.3.4 ポート 50004 に 15 秒ごとにストリーミングするサブスクリプションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000
switch(conf-tm-sub)# dst-grp 100
```

この例では、750 秒ごとに **show** コマンド データのケイデンス ベースのコレクションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 172.27.247.72 port 60001 protocol gRPC encoding GPB
switch(conf-tm-dest)# sensor-group 1
switch(conf-tm-sensor# data-source NX-API
switch(conf-tm-sensor)# path "show system resources" depth 0
switch(conf-tm-sensor)# path "show version" depth 0
switch(conf-tm-sensor)# path "show environment power" depth 0
switch(conf-tm-sensor)# path "show environment fan" depth 0
switch(conf-tm-sensor)# path "show environment temperature" depth 0
switch(conf-tm-sensor)# path "show process cpu" depth 0
switch(conf-tm-sensor)# path "show nve peers" depth 0
switch(conf-tm-sensor)# path "show nve vni" depth 0
switch(conf-tm-sensor)# path "show nve vni 4002 counters" depth 0
switch(conf-tm-sensor)# path "show int nve 1 counters" depth 0
switch(conf-tm-sensor)# path "show policy-map vlan" depth 0
switch(conf-tm-sensor)# path "show ip access-list test" depth 0
switch(conf-tm-sensor)# path "show system internal access-list resource utilization" depth
 0
switch(conf-tm-sensor)# subscription 1
switch(conf-tm-sub)# dst-grp 1
```

```
switch(conf-tm-dest)# snsr-grp 1 sample-interval 750000
```

この例では、sys/fm のイベント ベースのサブスクリプションを作成します。sys/fm MO に変更がある場合にのみ、データは接続先にストリーミングされます。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
```

動作中に、サンプル間隔を変更することで、センサー グループを周波数ベースからイベント ベースに変更したり、イベント ベースから周波数ベースに変更したりできます。この例では、センサー グループを前の例から頻度ベースに変更します。次のコマンドの後、テレメトリ アプリケーションは 7 秒ごとに sys/fm データの接続先へのストリーミングを開始します。

```
switch(config)# telemetry
switch(config-telemetry)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
```

複数のセンサー グループと接続先を 1 つのサブスクリプションにリンクできます。この例のサブスクリプションは、イーサネット ポート 1 / 1 のデータを 4 つの異なる接続先に 10 秒ごとにストリーミングします。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol HTTP encoding JSON
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

次に、センサー グループに複数のパスを含め、接続先グループに複数の接続先プロファイルを含め、サブスクリプションを複数のセンサー グループと宛先グループにリンクできる例を表示します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# path sys/epId-1 depth 0
switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0

switch(config-telemetry)# sensor-group 200
switch(conf-tm-sensor)# path sys/cdp depth 0
```

```
switch(conf-tm-sensor)# path sys/ipv4 depth 0

switch(config-telemetry)# sensor-group 300
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# path sys/bgp depth 0

switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 4.3.2.5 port 50005

switch(conf-tm-dest)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001

switch(conf-tm-dest)# destination-group 300
switch(conf-tm-dest)# ip address 1.2.3.4 port 60003

switch(conf-tm-dest)# subscription 600
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 200 sample-interval 20000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200

switch(conf-tm-dest)# subscription 900
switch(conf-tm-sub)# snsr-grp 200 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 300 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 300
```

この例に示すように、**show running-config telemetry** コマンドを使用してテレメトリ構成を確認できます。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# end
switch# show run telemetry

!Command: show running-config telemetry
!Time: Thu Oct 13 21:10:12 2016

version 7.0(3)I5(1)
feature telemetry

telemetry
destination-group 100
ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
```

テレメトリの構成と統計情報の表示

次の NX-OS CLI **show** コマンドを使用して、テレメトリの構成、統計情報、エラー、およびセッション情報を表示します。

show telemetry yang direct-path cisco-nxos-device

このコマンドは、他のパスよりもパフォーマンスが向上するように直接エンコードされた YANG パスを表示します。

```
switch# show telemetry yang direct-path cisco-nxos-device
) Cisco-NX-OS-device:System/lldp-items
2) Cisco-NX-OS-device:System/acl-items
3) Cisco-NX-OS-device:System/mac-items
4) Cisco-NX-OS-device:System/intf-items
5) Cisco-NX-OS-device:System/procsys-items/sysload-items
6) Cisco-NX-OS-device:System/ospf-items
7) Cisco-NX-OS-device:System/procsys-items
8) Cisco-NX-OS-device:System/ipqos-items/queuing-items/policy-items/out-items
9) Cisco-NX-OS-device:System/mac-items/static-items
10) Cisco-NX-OS-device:System/ch-items
11) Cisco-NX-OS-device:System/cdp-items
12) Cisco-NX-OS-device:System/bd-items
13) Cisco-NX-OS-device:System/eps-items
14) Cisco-NX-OS-device:System/ipv6-items
```

show telemetry control database

次に、テレメトリの構成を反映している内部データベースのコマンドを表示します。

```
switch# show telemetry control database ?
  <CR>
  >                   Redirect it to a file
  >>                  Redirect it to a file in append mode
  destination-groups  Show destination-groups
  destinations        Show destinations
  sensor-groups       Show sensor-groups
  sensor-paths        Show sensor-paths
  subscriptions       Show subscriptions
  |                   Pipe command output to filter

switch# show telemetry control database

Subscription Database size = 1

--------------------------------------------------------------------------------
Subscription ID      Data Collector Type
--------------------------------------------------------------------------------
100                  DME NX-API

Sensor Group Database size = 1

--------------------------------------------------------------------------------
Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
--------------------------------------------------------------------------------
100              Timer              10000(Running)         1

Sensor Path Database size = 1

--------------------------------------------------------------------------------
Subscribed Query Filter  Linked Groups  Sec Groups  Retrieve level  Sensor Path
--------------------------------------------------------------------------------
No                       1              0           Full            sys/fm

Destination group Database size = 2

--------------------------------------------------------------------------------
```

```
Destination Group ID  Refcount
------------------------------------------------------------------------------
100                   1

Destination Database size = 2

------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
------------------------------------------------------------------------------
192.168.20.111       12345      JSON       HTTP       1
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry control database sensor-paths

このコマンドは、テレメトリ設定のセンサーパスの詳細を表示します。これには、エンコーディング、収集、トランスポート、およびストリーミングのカウンタが含まれます。

```
switch(conf-tm-sub)# show telemetry control database sensor-paths
Sensor Path Database size = 4
------------------------------------------------------------------------------------------------------------
Row ID     Subscribed Linked Groups  Sec Groups  Retrieve level  Path(GroupId) : Query :
Filter
------------------------------------------------------------------------------------------------------------
1          No         1              0           Full            sys/cdp(1) : NA : NA

GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 65785/65785/65785
Collection Time in ms (Cur/Min/Max): 10/10/55
Encoding Time in ms (Cur/Min/Max): 8/8/9
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 18/18/65

2          No         1              0           Self            show module(2) : NA : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 1107/1106/1107
Collection Time in ms (Cur/Min/Max): 603/603/802
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/1
Streaming Time in ms (Cur/Min/Max): 605/605/803

3          No         1              0           Full            sys/bgp(1) : NA : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 0/0/44
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1/1/44

4          No         1              0           Self            show version(2) : NA : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
JSON Encoded Data size in bytes (Cur/Min/Max): 2442/2441/2442
Collection Time in ms (Cur/Min/Max): 1703/1703/1903
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1703/1703/1904

switch(conf-tm-sub)#
```

show telemetry control stats

このコマンドは、テレメトリの構成についての内部データベースの統計を表示します。

```
switch# show telemetry control stats
show telemetry control stats entered

-------------------------------------------------------------------------------
Error Description                                                  Error Count
-------------------------------------------------------------------------------
Chunk allocation failures                                          0
Sensor path Database chunk creation failures                       0
Sensor Group Database chunk creation failures                      0
Destination Database chunk creation failures                       0
Destination Group Database chunk creation failures                 0
Subscription Database chunk creation failures                      0
Sensor path Database creation failures                             0
Sensor Group Database creation failures                            0
Destination Database creation failures                             0
Destination Group Database creation failures                       0
Subscription Database creation failures                            0
Sensor path Database insert failures                               0
Sensor Group Database insert failures                              0
Destination Database insert failures                               0
Destination Group Database insert failures                         0
Subscription insert to Subscription Database failures              0
Sensor path Database delete failures                               0
Sensor Group Database delete failures                              0
Destination Database delete failures                               0
Destination Group Database delete failures                         0
Delete Subscription from Subscription Database failures            0
Sensor path delete in use                                          0
Sensor Group delete in use                                         0
Destination delete in use                                          0
Destination Group delete in use                                    0
Delete destination(in use) failure count                           0
Failed to get encode callback                                      0
Sensor path Sensor Group list creation failures                    0
Sensor path prop list creation failures                            0
Sensor path sec Sensor path list creation failures                 0
Sensor path sec Sensor Group list creation failures                0
Sensor Group Sensor path list creation failures                    0
Sensor Group Sensor subs list creation failures                    0
Destination Group subs list creation failures                      0
Destination Group Destinations list creation failures              0
Destination Destination Groups list creation failures              0
Subscription Sensor Group list creation failures                   0
Subscription Destination Groups list creation failures             0
Sensor Group Sensor path list delete failures                      0
Sensor Group Subscriptions list delete failures                    0
Destination Group Subscriptions list delete failures               0
Destination Group Destinations list delete failures                0
Subscription Sensor Groups list delete failures                    0
Subscription Destination Groups list delete failures               0
Destination Destination Groups list delete failures                0
Failed to delete Destination from Destination Group                0
Failed to delete Destination Group from Subscription               0
Failed to delete Sensor Group from Subscription                    0
Failed to delete Sensor path from Sensor Group                     0
Failed to get encode callback                                      0
Failed to get transport callback                                   0
switch#  Destination Database size = 1

-------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
-------------------------------------------------------------------------------
```

```
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry data collector brief

このコマンドは、データ収集に関する簡略化した統計情報を表示します。

```
switch# show telemetry data collector brief

------------------------------------------------------------------------
Collector Type       Successful Collections     Failed Collections
------------------------------------------------------------------------
DME                   143                        0
```

show telemetry data collector details

このコマンドは、すべてのセンサー パスの詳細を含む、データ収集に関する詳細な統計情報を表示します。

```
switch# show telemetry data collector details

------------------------------------------------------------------------------
Succ Collections      Failed Collections     Sensor Path
------------------------------------------------------------------------------
150                   0                      sys/fm
```

show telemetry event collector errors

このコマンドは、イベント収集に関するエラー統計情報を表示します。

```
switch# show telemetry event collector errors

------------------------------------------------------------------------------
Error Description                                 Error Count
------------------------------------------------------------------------------
APIC-Cookie Generation Failures                    - 0
Authentication Failures                            - 0
Authentication Refresh Failures                    - 0
Authentication Refresh Timer Start Failures        - 0
Connection Timer Start Failures                    - 0
Connection Attempts                                - 3
Dme Event Subscription Init Failures               - 0
Event Data Enqueue Failures                        - 0
Event Subscription Failures                        - 0
Event Subscription Refresh Failures                - 0
Pending Subscription List Create Failures          - 0
Subscription Hash Table Create Failures            - 0
Subscription Hash Table Destroy Failures           - 0
Subscription Hash Table Insert Failures            - 0
Subscription Hash Table Remove Failures            - 0
Subscription Refresh Timer Start Failures          - 0
Websocket Connect Failures                         - 0
```

show telemetry event collector stats

このコマンドは、すべてのセンサー パスの内訳を含むイベント収集に関する統計情報を表示します。

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Collection Count  Latest Collection Time    Sensor Path
------------------------------------------------------------------------------
```

show telemetry control pipeline stats

このコマンドは、テレメトリ パイプラインの統計情報を表示します。

```
switch# show telemetry pipeline stats
Main Statistics:
    Timers:
        Errors:
            Start Fail        =     0

    Data Collector:
        Errors:
            Node Create Fail  =     0

    Event Collector:
        Errors:
            Node Create Fail  =     0    Node Add Fail     =     0
            Invalid Data      =     0



Queue Statistics:
    Request Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

    Data Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
```

```
Info:
    Actual Size       =    50    Current Size      =     0
    Max Size          =     0    Full Count        =     0

Errors:
    Enqueue Error     =     0    Dequeue Error     =     0
```

show telemetry transport

次に、構成されているすべての転送セッションの例を表示します。

```
switch# show telemetry transport

Session Id      IP Address      Port       Encoding   Transport  Status
--------------------------------------------------------------------------------
0               192.168.20.123   50001      GPB        gRPC       Connected
```

表 *25 : show telemetry transport* の構文の説明

構文	説明
show	実行中のシステム情報を表示します。
telemetry	テレメトリ情報を表示します
トランスポート	テレメトリのトランスポート情報を表示します。
session_id	（オプション）セッション ID
stats	（オプション）すべてのテレメトリ統計情報を表示します。
errors	（オプション）すべてのテレメトリ エラー情報を表示します。
読み取り専用	（オプション）
TABLE_transport_info	（オプション）トランスポート情報
session_idx	（オプション）セッション ID
ip_address	（オプション）トランスポート IP アドレス
port	（任意）トランスポート ポート
dest_info	（オプション）接続先情報
encoding_type	（オプション）エンコーディング タイプ
transport_type	（オプション）トランスポート タイプ

構文	説明
transport_status	（オプション）トランスポート ステータス
transport_security_cert_fname	（オプション）トランスポート セキュリティ ファイル名
transport_last_connected	（オプション）最後に接続されたトランスポート
transport_last_disconnected	（オプション）この宛先構成が最後に削除された時刻
transport_errors_count	（オプション）トランスポート エラー数
transport_last_tx_error	（オプション）トランスポートの最後の送信エラー
transport_statistics	（オプション）トランスポート統計情報
t_session_id	（オプション）トランスポート セッション ID
connect_statistics	（オプション）接続統計情報
connect_count	（オプション）接続数
last_connected	（オプション）最終接続のタイムスタンプ
Disconnect_count	（オプション）切断数
last_disconnected	（オプション）この宛先構成が最後に削除された時刻
trans_statistics	（オプション）トランスポート統計情報
compression	（オプション）圧縮ステータス
source_interface_name	（オプション）送信元インターフェイス名
source_interface_ip	（オプション）送信元インターフェイス IP
transmit_count	（オプション）送信数
last_tx_time	（オプション）最終送信時刻
min_tx_time	（オプション）最小送信時間
max_tx_time	（オプション）最大送信時間
avg_tx_time	（オプション）平均送信時間
cur_tx_time	（オプション）現在の送信時間

構文	説明
transport_errors	（オプション）トランスポート エラー
connect_errors	（オプション）接続エラー
connect_errors_count	（オプション）接続エラー数
trans_errors	（オプション）トランスポート エラー
trans_errors_count	（オプション）トランスポート エラー数
last_tx_error	（オプション）最後のトランスポート エラー
last_tx_return_code	（オプション）最後の送信戻りコード
transport_retry_stats	（オプション）再試行統計情報
ts_event_retry_bytes	（オプション）イベント再試行バッファ サイズ
ts_timer_retry_bytes	（オプション）タイマー再試行バッファ サイズ
ts_event_retry_size	（オプション）イベント再試行メッセージ数
ts_timer_retry_size	（オプション）タイマー再試行メッセージ数
ts_retries_sent	（オプション）送信された再試行回数
ts_retries_dropped	（オプション）ドロップされた再試行回数
event_retry_bytes	（オプション）イベント再試行バッファ サイズ
timer_retry_bytes	（オプション）タイマー再試行バッファ サイズ
retries_sent	（オプション）送信された再試行回数
retries_dropped	（オプション）ドロップされた再試行回数
retry_buffer_size	（オプション）再試行バッファ サイズ

show telemetry transport <session-id>

次のコマンドでは、特定の転送セッションの詳細なセッション情報が表示されます。

```
switch# show telemetry transport 0

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Fri Sep 02 11:45:57.505 UTC

Tx Error Count:      224
```

```
Last Tx Error:       Fri Sep 02 12:23:49.555 UTC

switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235 Encoding:           JSON
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never

Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
```

次に、IPv6 エントリの出力例を示します。

```
switch# show telemetry transport 0
Session Id: 0
IP Address:Port [10:10::1]:8000
Transport: GRPC
Status: Idle
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
Event Retry Queue Bytes: 0
Event Retry Queue Size: 0
Timer Retry Queue Bytes: 0
Timer Retry Queue Size: 0
Sent Retry Messages: 0
Dropped Retry Messages: 0
```

show telemetry transport <session-id> stats

次に、特定の転送セッションの詳細のコマンドを示します。

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

show telemetry transport <session-id> errors

次のコマンドでは、特定の転送セッションの詳細なエラーの統計情報が表示されます。

```
switch# show telemetry transport 0 errors

Session Id:                    0
Connection Stats
   Connection Count            1
   Last Connected:             Mon May 01 11:29:46.912 PST
   Disconnect Count            0
   Last Disconnected:          Never
Transmission Stats
```

```
   Transmit Count:            1225
   Last TX time:              Tue May 02 11:40:03.531 PST
   Min Tx Time:               7                  ms
   Max Tx Time:               1760               ms
   Avg Tx Time:               500                ms
```

show telemetry control databases sensor-paths

次の構成手順により、次の **show telemetry control databases sensor-paths** コマンド出力が得られます。

```
feature telemetry

telemetry
  destination-group 1
    ip address 172.25.238.13 port 50600 protocol gRPC encoding GPB
  sensor-group 1
    path sys/cdp depth unbounded
    path sys/intf depth unbounded
    path sys/mac depth 0
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 1000
```

コマンド出力。

```
switch# show telemetry control databases sensor-paths

Sensor Path Database size = 3
------------------------------------------------------------------------------
--------------------
Row ID     Subscribed Linked Groups  Sec Groups  Retrieve level  Path(GroupId) :
Query : Filter
------------------------------------------------------------------------------
--------------------
1          No         1              0           Full            sys/cdp(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 30489/30489/30489
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 6/5/54
Encoding Time in ms (Cur/Min/Max): 5/5/6
Transport Time in ms (Cur/Min/Max): 1027/55/1045
Streaming Time in ms (Cur/Min/Max): 48402/5/48402

2          No         1              0           Full            sys/intf(1) : N
A : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 539466/539466/539466
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 66/64/114
Encoding Time in ms (Cur/Min/Max): 91/90/92
Transport Time in ms (Cur/Min/Max): 4065/4014/5334
Streaming Time in ms (Cur/Min/Max): 48365/64/48365

3          No         1              0           Self            sys/mac(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 247/247/247
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 1/1/47
Encoding Time in ms (Cur/Min/Max): 1/1/1
```

```
Transport Time in ms (Cur/Min/Max): 4/1/6
Streaming Time in ms (Cur/Min/Max): 47369/1/47369
```

show telemetry transport sessions

次のコマンドは、すべてのトランスポート セッションをループし、1 つのコマンドで情報を出力します。

```
switch# show telemetry transport sessions
switch# show telemetry transport stats
switch# show telemetry transport errors
switch# show telemetry transport all
```

次に、テレメトリ トランスポート セッションの例を示します。

```
switch# show telemetry transport sessions
Session Id:          0
IP Address:Port      172.27.254.13:50004
Transport:           GRPC
Status:              Transmit Error
SSL Certificate:     trustpoint1
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      2
Last Tx Error:       Wed Aug 19 23:32:21.749 UTC
…
Session Id:          4
IP Address:Port      172.27.254.13:50006
Transport:           UDP
```

テレメトリ エフェメラル イベント

エフェメラル イベントをサポートするために、新しいセンサー パス クエリ条件が追加されました。アカウンティング ログの外部イベント ストリーミングを有効にするには、次のクエリ条件を使用します。

```
sensor-group 1
path sys/accounting/log query-condition query-target=subtree&complete-mo=yes&notify-interval=1
```

エフェメラル イベントをサポートするその他のセンサー パスは次のとおりです。

```
sys/pim/inst/routedb-route, sys/pim/pimifdb-adj, sys/pim/pimifdb-prop
sys/igmp/igmpifdb-prop, sys/igmp/inst/routedb, sys/igmpsnoop/inst/dom/db-exptrack,
sys/igmpsnoop/inst/dom/db-group, sys/igmpsnoop/inst/dom/db-mrouter
sys/igmpsnoop/inst/dom/db-querier, sys/igmpsnoop/inst/dom/db-snoop
```

テレメトリ ログとトレース情報の表示

ログとトレース情報を表示するには、次の NX-OS CLI コマンドを使用します。

テクニカル サポート テレメトリを表示

この NX-OS CLI コマンドは、テクニカル サポート ログからテレメトリ ログの内容を収集します。この例では、コマンド出力がブートフラッシュのファイルにリダイレクトされます。

```
switch# show tech-support telemetry > bootflash:tmst.log
```

NX-API を使用したテレメトリの構成

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in the section "Telemetry Model in the DME." Following are the main objects to be configured:

- **fmEntity** — Contains the NX-API and Telemetry feature states.
  - **fmNxapi** — Contains the NX-API state.
  - **fmTelemetry** — Contains the Telemetry feature state.
- **telemetryEntity** — Contains the telemetry feature configuration.
  - **telemetrySensorGroup** — Contains the definitions of one or more sensor paths or nodes to be monitored for telemetry. The telemetry entity can contain one or more sensor groups.
    - **telemetryRtSensorGroupRel** — Associates the sensor group with a telemetry subscription.
    - **telemetrySensorPath** — A path to be monitored. The sensor group can contain multiple objects of this type.
  - **telemetryDestGroup** — Contains the definitions of one or more destinations to receive telemetry data. The telemetry entity can contain one or more destination groups.
    - **telemetryRtDestGroupRel** — Associates the destination group with a telemetry subscription.
    - **telemetryDest** — A destination address. The destination group can contain multiple objects of this type.
  - **telemetrySubscription** — Specifies how and when the telemetry data from one or more sensor groups is sent to one or more destination groups.
    - **telemetryRsDestGroupRel** — Associates the telemetry subscription with a destination group.
    - **telemetryRsSensorGroupRel** — Associates the telemetry subscription with a sensor group.
  - **telemetryCertificate** — Associates the telemetry subscription with a certificate and hostname.

To configure the telemetry feature using the NX-API, you must construct a JSON representation of the telemetry object structure and push it to the DME with an HTTP or HTTPS POST operation.

Note For detailed instructions on using the NX-API, see the *Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*.

Before you begin

Your switch must be configured to run the NX-API from the CLI:

```
switch(config)# feature nxapi

nxapi use-vrf vrf_name
nxapi http port port_number
```

Procedure

	Command or Action	Purpose
Step 1	Enable the telemetry feature. **Example:** `{ "fmEntity" : { "children" : [{ "fmTelemetry" : { "attributes" : { "adminSt" : "enabled" } } } } ] } }`	The root element is **fmTelemetry** and the base path for this element is `sys/fm`. Configure the **adminSt** attribute as `enabled`.
Step 2	Create the root level of the JSON payload to describe the telemetry configuration. **Example:** `{ "telemetryEntity": { "attributes": { "dn": "sys/tm" }, } }`	The root element is **telemetryEntity** and the base path for this element is `sys/tm`. Configure the **dn** attribute as `sys/tm`.
Step 3	Create a sensor group to contain the defined sensor paths. **Example:** `"telemetrySensorGroup": { "attributes": { "id": "10", "rn": "sensor-10" }, "children": [{ }] }`	A telemetry sensor group is defined in an object of class **telemetrySensorGroup**. Configure the following attributes of the object: • **id** — An identifier for the sensor group. Currently only numeric ID values are supported. • **rn** — The relative name of the sensor group object in the format: **sensor-***id*. • **dataSrc** — Selects the data source from **DEFAULT**, **DME**, **YANG**, or **NX-API**. Children of the sensor group object include sensor paths and one or more relation objects (**telemetryRtSensorGroupRel**) to associate the sensor group with a telemetry subscription.

	Command or Action	Purpose
Step 4	(Optional) Add an SSL/TLS certificate and a host. **Example:** `{ "telemetryCertificate": { "attributes": { "filename": "root.pem" "hostname": "c.com" } } }`	The **telemetryCertificate** defines the location of the SSL/TLS certificate with the telemetry subscription/destination.
Step 5	Define a telemetry destination group. **Example:** `{ "telemetryDestGroup": { "attributes": { "id": "20" } } }`	A telemetry destination group is defined in **telemetryEntity**. Configure the id attribute.
Step 6	Define a telemetry destination profile. **Example:** `{ "telemetryDestProfile": { "attributes": { "adminSt": "enabled" }, "children": [ { "telemetryDestOptSourceInterface": { "attributes": { "name": "lo0" } } } ] } }`	A telemetry destination profile is defined in **telemetryDestProfile**. • Configure the **adminSt** attribute as `enabled`. • Under **telemetryDestOptSourceInterface**, configure the **name** attribute with an interface name to stream data from the configured interface to a destination with the source IP address.
Step 7	Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent. **Example:** `{ "telemetryDest": { "attributes": { "addr": "1.2.3.4", "enc": "GPB", "port": "50001", "proto": "gRPC", "rn": "addr-[1.2.3.4]-port-50001"`	A telemetry destination is defined in an object of class **telemetryDest**. Configure the following attributes of the object: • **addr** — The IP address of the destination. • **port** — The port number of the destination. • **rn** — The relative name of the destination object in the format: **path-[***path***]**. • **enc** — The encoding type of the telemetry data to be sent. NX-OS supports:

	Command or Action	Purpose
	<pre> } } }</pre>	• Google protocol buffers (GPB) for gRPC. • JSON for C. • **proto** — The transport protocol type of the telemetry data to be sent. NX-OS supports: • gRPC • HTTP • Supported encoded types are: • HTTP/JSON YES • HTTP/Form-data YES Only supported for Bin Logging. • GRPC/GPB-Compact YES Native Data Source Only. • GRPC/GPB YES • UDP/GPB YES • UDP/JSON YES

Step 8	Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.
	Example:
	<pre>{
	    "telemetryDestGrpOptChunking": {
	        "attributes": {
	            "chunkSize": "2048",
	            "dn": "sys/tm/dest-1/chunking"
	        }
	    }
	}</pre>
		See Guidelines and Limitations section for more information.

Step 9	Create a telemetry subscription to configure the telemetry behavior.
	Example:
	<pre>"telemetrySubscription": {
	    "attributes": {
	        "id": "30",
	        "rn": "subs-30"
	    }, "children": [{
	    }]
	}</pre>
		A telemetry subscription is defined in an object of class **telemetrySubscription**. Configure the following attributes of the object:
		• **id** — An identifier for the subscription. Currently only numeric ID values are supported.
		• **rn** — The relative name of the subscription object in the format: **subs-***id*.
		Children of the subscription object include relation objects for sensor groups (**telemetryRsSensorGroupRel**) and destination groups (**telemetryRsDestGroupRel**).

	Command or Action	Purpose
Step 10	Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**). **Example:** `{ "telemetrySubscription": { "attributes": { "id": "30" } "children": [{ "telemetryRsSensorGroupRel": { "attributes": { "sampleIntvl": "5000", "tDn": "sys/tm/sensor-10" } } } ] } }`	
Step 11	Create a relation object as a child object of the subscription to associate the subscription to the telemetry sensor group and to specify the data sampling behavior. **Example:** `"telemetryRsSensorGroupRel": { "attributes": { "rType": "mo", "rn": "rssensorGroupRel-[sys/tm/sensor-10]", "sampleIntvl": "5000", "tCl": "telemetrySensorGroup", "tDn": "sys/tm/sensor-10", "tType": "mo" } }`	The relation object is of class **telemetryRsSensorGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object: • **rn** — The relative name of the relation object in the format: **rssensorGroupRel-[sys/tm/***sensor-group-id***]**. • **sampleIntvl** — The data sampling period in milliseconds. An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds. • **tCl** — The class of the target (sensor group) object, which is **telemetrySensorGroup**. • **tDn** — The distinguished name of the target (sensor group) object, which is **sys/tm/***sensor-group-id*. • **rType** — The relation type, which is **mo** for managed object. • **tType** — The target type, which is **mo** for managed object.
Step 12	Define one or more sensor paths or nodes to be monitored for telemetry. **Example:**	A sensor path is defined in an object of class **telemetrySensorPath**. Configure the following attributes of the object:

	Command or Action	Purpose
	Single sensor path `{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }` **Example:** Multiple sensor paths `{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }, { "telemetrySensorPath": { "attributes": { "excludeFilter": "", "filterCondition": "", "path": "sys/fm/dhcp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }`  **Example:** Single sensor path filtering for BGP disable events:  `{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition":`	• **path** — The path to be monitored. • **rn** — The relative name of the path object in the format: **path-[***path***]** • **depth** — The retrieval level for the sensor path. A depth setting of **0** retrieves only the root MO properties. • **filterCondition** — (Optional) Creates a specific filter for event-based subscriptions. The DME provides the filter expressions. For more information about filtering, see the Cisco APIC REST API Usage Guidelines on composing queries. You can find it at the following Cisco APIC documents landing page:

	Command or Action	Purpose

	<pre>"eq(fmBgp.operSt.\"disabled\")",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre>	

Step 13	Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).	
Step 14	Add destinations as child objects to the destination group object (**telemetryDestGroup**).	
Step 15	Add the destination group object as a child object to the root element (**telemetryEntity**).	

Step 16	Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
	Example:
	<pre>"telemetryRtSensorGroupRel": {
	    "attributes": {
	       "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
	
	        "tCl": "telemetrySubscription",
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		The relation object is of class **telemetryRtSensorGroupRel** and is a child object of **telemetrySensorGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtsensorGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

Step 17	Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
	Example:
	<pre>"telemetryRtDestGroupRel": {
	    "attributes": {
	        "rn": "rtdestGroupRel-[sys/tm/subs-30]",
	        "tCl": "telemetrySubscription",
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		The relation object is of class **telemetryRtDestGroupRel** and is a child object of **telemetryDestGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtdestGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

Step 18	Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.
	Example:
	<pre>"telemetryRsDestGroupRel": {
	    "attributes": {
	        "rType": "mo",
	        "rn": "rsdestGroupRel-[sys/tm/dest-20]",</pre>
		The relation object is of class **telemetryRsDestGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rsdestGroupRel-[sys/tm/***destination-group-id***]**.
		• **tCl** — The class of the target (destination group) object, which is **telemetryDestGroup**.

	Command or Action	Purpose

	<pre>"tCl": "telemetryDestGroup",
	"tDn": "sys/tm/dest-20",
	"tType": "mo"
	}
	}</pre>
		• **tDn** — The distinguished name of the target (destination group) object, which is **sys/tm/***destination-group-id*.
		• **rType** — The relation type, which is **mo** for managed object.
		• **tType** — The target type, which is **mo** for managed object.

Step 19	Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.	The base path for the telemetry entity is `sys/tm` and the NX-API endpoint is: `{{URL}}/api/node/mo/sys/tm.json`

Example

The following is an example of all the previous steps that are collected into one POST payload (note that some attributes may not match):

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50051",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
```

```
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "5000",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

NX-API を使用したテレメトリの構成例

宛先へのストリーミング パス

この例では、パス `sys/cdp` および `sys/ipv4` を接続先 `1.2.3.4` ポート `50001` に 5 秒ごとにストリーミングするサブスクリプションを作成します。

```
POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
    "telemetryEntity": {
        "attributes": {
            "dn": "sys/tm"
        },
        "children": [{
            "telemetrySensorGroup": {
                "attributes": {
                    "id": "10",
                    "rn": "sensor-10"
                },  "children": [{
                    "telemetryRtSensorGroupRel": {
                        "attributes": {
                            "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
```

```
                    "attributes": {
                        "path": "sys/cdp",
                        "rn": "path-[sys/cdp]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/ipv4",
                        "rn": "path-[sys/ipv4]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }]
        }
    }, {
        "telemetryDestGroup": {
            "attributes": {
                "id": "20",
                "rn": "dest-20"
            },
            "children": [{
                "telemetryRtDestGroupRel": {
                    "attributes": {
                        "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                        "tCl": "telemetrySubscription",
                        "tDn": "sys/tm/subs-30"
                    }
                }
            }, {
                "telemetryDest": {
                    "attributes": {
                        "addr": "1.2.3.4",
                        "enc": "GPB",
                        "port": "50001",
                        "proto": "gRPC",
                        "rn": "addr-[1.2.3.4]-port-50001"
                    }
                }
            }]
        }
    }, {
        "telemetrySubscription": {
            "attributes": {
                "id": "30",
                "rn": "subs-30"
            },
            "children": [{
                "telemetryRsDestGroupRel": {
                    "attributes": {
                        "rType": "mo",
                        "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                        "tCl": "telemetryDestGroup",
                        "tDn": "sys/tm/dest-20",
                        "tType": "mo"
```

```
                        }
                    }
                }, {
                    "telemetryRsSensorGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                            "sampleIntvl": "5000",
                            "tCl": "telemetrySensorGroup",
                            "tDn": "sys/tm/sensor-10",
                            "tType": "mo"
                        }
                    }
                }]
            }
        }]
    }
}
```

BGP 通知のフィルタ条件

次のペイロードの例では、`telemetrySensorPath` MO の `filterCondition` 属性に従って BFP 機能が無効になっているときにトリガーされる通知を有効にします。データは`10.30.217.80` ポート `50055` にストリーミングされます。

```
POST  https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50055",
              "enc": "GPB",
              "proto": "gRPC"
```

```
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "0",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

テレメトリ構成のための Postman コレクションの使用

Postman コレクションの例は、テレメトリ機能の構成を開始する簡単な方法であり、1 つのペイロードですべてのテレメトリ CLI に相当するものを実行できます。好みのテキストエディターを使用して前述のリンクのファイルを変更し、ペイロードをニーズに合わせて更新してから、Postman でコレクションを開いてコレクションを実行します。

DME のテレメトリ モデル

テレメトリ アプリケーションは、次の構造を持つ DME でモデル化されます。

```
model
|----package [name:telemetry]
    |  @name:telemetry
    |----objects
         |----mo [name:Entity]
              |    @name:Entity
              |      @label:Telemetry System
              |--property
              |    @name:adminSt
              |      @type:AdminState
              |
              |----mo [name:SensorGroup]
              |    |    @name:SensorGroup
              |    |      @label:Sensor Group
              |    |--property
```

```
|     |    @name:id [key]
|     |      @type:string:Basic
|     |
|     |----mo [name:SensorPath]
|          |   @name:SensorPath
|          |     @label:Sensor Path
|          |--property
|          |    @name:path [key]
|          |      @type:string:Basic
|          |    @name:filterCondition
|          |      @type:string:Basic
|          |    @name:excludeFilter
|          |      @type:string:Basic
|          |    @name:depth
|          |      @type:RetrieveDepth
|
|----mo [name:DestGroup]
|     |    @name:DestGroup
|     |      @label:Destination Group
|     |--property
|     |    @name:id
|     |      @type:string:Basic
|     |
|     |----mo [name:Dest]
|          |   @name:Dest
|          |     @label:Destination
|          |--property
|          |    @name:addr [key]
|          |      @type:address:Ip
|          |    @name:port [key]
|          |      @type:scalar:Uint16
|          |    @name:proto
|          |      @type:Protocol
|          |    @name:enc
|          |      @type:Encoding
|
|----mo [name:Subscription]
     |    @name:Subscription
     |      @label:Subscription
     |--property
     |    @name:id
     |      @type:scalar:Uint64
     |----reldef
     |     |  @name:SensorGroupRel
     |     |    @to:SensorGroup
     |     |    @cardinality:ntom
     |     |    @label:Link to sensorGroup entry
     |     |--property
     |          @name:sampleIntvl
     |            @type:scalar:Uint64
     |
     |----reldef
          |  @name:DestGroupRel
          |    @to:DestGroup
          |    @cardinality:ntom
          |    @label:Link to destGroup entry
```

クラウド スケール ソフトウェア テレメトリ

クラウド スケール ソフトウェア テレメトリについて

NX-OS リリース 9.3(1) 以降、ソフトウェア テレメトリは、Tahoe ASIC を使用する Cisco Nexus クラウド スケール スイッチでサポートされます。このリリースで、サポートされているクラウド スケール スイッチは、ASIC と緊密に統合された TCP/IP サーバーをホストします。これにより、スイッチからのテレメトリ データのレポートをすばやく処理できます。サーバーは TCP ポート 7891 を使用します。テレメトリ クライアントはこのポートでサーバーに接続して、最大 10 ミリ秒でハードウェア カウンタ データを取得できます。

クラウドスケール ソフトウェアテレメトリには、独自のクライアントプログラムを作成したり、NX-OS リリース 9.3.1 以降にバンドルされているデフォルトのクライアント プログラムを使用したりする柔軟性があります。クライアント プログラムは、Python 2.7 以降、C、PHP など、TCP/IP をサポートする任意のプログラミング言語で作成できます。クライアント プログラムは、正しいメッセージ フォーマットで作成する必要があります。

NX-OS リリース 9.3(1) 以降、クラウド スケール ソフトウェア テレメトリ機能は NX-OS で使用できます。この機能はデフォルトで有効になっているため、NX-OS 9.3(1) 以降を実行しているサポート対象のスイッチでは、この機能を使用できます。

Cloud Scale ソフトウェア テレメトリ メッセージの形式

Cloud Scale テレメトリは、クライアントとスイッチ上の TCP/IP サーバー間のハンドシェイクで始まります。その間にクライアントは TCP ソケットを介して接続を開始します。クライアント メッセージは、32 ビット整数での 0 です。スイッチは、特定のファオーマットのカウンタ データを含むメッセージで応答します。

NX-OS リリース 9.3(1) では、次のメッセージ フォーマットがサポートされています。独自のクライアント プログラムを作成する場合は、クライアントが開始するメッセージがこのフォーマットに準拠していることを確認してください。

長さ	指定します。
4 バイト	ポート数、*N*

長さ	指定します。
56 バイト	各ポートのデータ、合計 56 * *N* バイト。 データの各 56 バイト チャンクは、次のもので構成されます。 • 24 バイトのインターフェイス名 • 8 バイトの送信（TX）パケット • 8 バイトの送信（TX）バイト • 8 バイトの受信（RX）パケット • 8 バイトの受信（RX）バイト

Guidelines and Limitations for Cloud Scale Software Telemetry

The following are the guidelines and limitations for the Cloud Scale software telemetry feature:

- For information about supported platforms for Cisco NX-OS prior to release 9.3(x), see the section for *Platform Support for Programmability Features* in that guide. Starting with Cisco NX-OS release 9.3(x) for information about supported platforms, see the Nexus Switch Platform Matrix.
- For custom client telemetry programs, one message format is supported. Your client programs must comply with this format.

テレメトリ パス ラベル

テレメトリ パス ラベルについて

NX-OS リリース 9.3(1) 以降、モデル駆動型テレメトリはパス ラベルをサポートします。パス ラベルを使用すると、複数のソースからテレメトリ データを一度に簡単に収集できます。この機能では、収集するテレメトリ データのタイプを指定すると、テレメトリ機能によって複数のパスからそのデータが収集されます。次に、機能は情報を 1 つの統合された場所（パス ラベル）に返します。この機能により、次の作業が不要になるため、テレメトリの使用が簡素化されます。

- Cisco DME モデルに関する深く包括的な知識を持っています。
- 収集されるイベントの数と頻度のバランスを取りながら、複数のクエリを作成し、サブスクリプションに複数のパスを追加します。
- スイッチからテレメトリ情報の複数のチャンクを収集し、有用性を簡素化します。

パス ラベルは、モデル内の同じオブジェクト タイプの複数のインスタンスにわたり、カウンタまたはイベントを収集して返します。パス ラベルは、次のテレメトリ グループをサポートします。

- ファン、温度、電力、ストレージ、スーパーバイザ、ライン カードなどのシャーシ情報をモニタリングする環境。
- すべてのインターフェイス カウンターとステータスの変更をモニタリングするインターフェイス。

  このラベルは、**query-condition** コマンドを使用して返されるデータを絞り込むための定義済みのキーワード フィルタをサポートします。
- リソース。CPU 使用率やメモリ使用率などのシステム リソースをモニタリングします。
- VXLAN: VXLAN ピア、VXLAN カウンタ、VLAN カウンター、および BGP ピア データを含む VXLAN EVPN をモニタリングします。

データの投票またはイベントの受信

センサー グループのサンプル間隔によって、テレメトリ データがパス ラベルに送信される方法とタイミングが決まります。サンプル間隔は、テレメトリ データを定期的に投票するか、イベントが発生したときにテレメトリ データを収集するように構成できます。

- テレメトリのサンプル間隔がゼロ以外の値に設定されている場合、テレメトリは各サンプル間隔中に環境、インターフェイス、情報技術、および vxlan ラベルのデータを定期的に送信します。
- サンプル間隔がゼロに設定されている場合、環境、インターフェイス、情報技術、vxlan ラベルで動作状態の更新、および MO の作成と削除が発生するとテレメトリはイベント通知を送信します。

データの投票または受信イベントは相互に排他的です。パス ラベルごとに投票またはイベント駆動型テレメトリを構成できます。

パス ラベル注意事項と制約事項

テレメトリ パス ラベル機能には、次の注意事項と制約事項があります。

- この機能は、Cisco DME データ 送信元のみをサポートします。
- 同じセンサー グループ内の通常の DME パスとユーザビリティ パスを混在させて一致させることはできません。たとえば、`sys/intf` と `[インターフェイス（interface）]` を同じセンサー グループに構成することはできません。また、`sys/intf` と `[interface（インターフェイス）]` で同じセンサー グループを構成することはできません。この状況が発生した場合、NX-OS は構成を拒否します。
- `oper-speed` や `counters=[detailed]` などのユーザー フィルター キーワードは、`[インターフェイス（interface）]` パスに対してのみサポートされます。
- この機能は、`[深度（depth）]`や`[フィルター条件（filter-condition）]`などの他のセンサー パス オプションをサポートしていません。

- テレメトリ パス ラベルには、パス ラベルの使用に関する次の制限があります。
  - 大文字と小文字が区別されるため、小文字のプレフィックス **show** で開始する必要があります。

    例：**show version** は許可されます。ただし、**SHOW version** または version は使用できません。
  - 次の文字を含めることはできません。
    - ;
    - |
    - " " または ' '
  - 次の単語を含めることはできません。
    - telemetry
    - conf t
    - 設定

データまたはイベントをポーリングするためのインターフェイスパスの構成

インターフェイス パス ラベルは、すべてのインターフェイス カウンタとステータスの変更をモニタリングします。次のインターフェイス タイプをサポートします。

- 物理
- サブインターフェイス
- 管理
- ループバック
- VLAN
- ポート チャネル

インターフェイスパスラベルを構成して、定期的にデータをポーリングするか、イベントを受信することができます。「データの投票またはイベントの受信（412 ページ）」を参照してください。

（注） このモデルは、サブインターフェイス、ループバック、または VLAN のカウンタをサポートしていないため、ストリームアウトされません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

Step 4	**path interface**
	例：
	`switch(conf-tm-sensor)# path interface`
	`switch(conf-tm-sensor)#`
		インターフェイス パス ラベルを構成して、複数の個々のインターフェイスに対して 1 つのテレメトリ データ クエリを送信できるようにします。ラベルは、複数のインターフェイスのクエリを 1 つに統合します。次に、テレメトリはデータを収集し、ラベルに返します。
		ポーリング間隔の設定方法に応じて、インターフェイス データは定期的に、またはインターフェイスの状態が変化するたびに送信されます。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

	コマンドまたはアクション	目的

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリデータを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

Step 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

非ゼロ カウンタのインターフェイス パスの構成

ゼロ以外の値を持つカウンターのみを返す事前定義されたキーワード フィルタを使用して、インターフェイス パス ラベルを構成できます。フィルタは `counters=[detailed]` です。

このフィルタを使用することにより、インターフェイス パスは使用可能なすべてのインターフェイス カウンターを収集し、収集したデータをフィルタ処理してから、結果を受信側に転送します。フィルタはオプションであり、使用しない場合、ゼロ値カウンターを含むすべてのカウンターがインターフェイス パスに表示されます。

（注） フィルタの使用は、概念的には **show interface mgmt0 counters detailed** と類似しています。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface query-condition counters=[detailed]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*

7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

Step 4	**path interface query-condition counters=[detailed]**
	例：
	`switch(conf-tm-sensor)# path interface query-condition counters=[detailed]`
	`switch(conf-tm-sensor)#`
		インターフェイス パス ラベルを構成し、すべてのインターフェイスからのゼロ以外のカウンターのみを照会します。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval* 例：	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテ

	コマンドまたはアクション	目的
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000` `switch(conf-tm-sub)#`	レメトリデータを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

Step 9	**dst-group** *dgrp_id*
	例:
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

動作速度のインターフェイス パスの構成

指定された動作速度のインターフェイスのカウンタを返す定義済みのキーワード フィルタを使用して、インターフェイス パス ラベルを構成できます。フィルタは `oper-speed=[]` です。次の動作速度がサポートされています: auto、10M、100M、1G、10G、40G、200G、および 400G。

このフィルタを使用することにより、インターフェース パスは指定された速度のインターフェースのテレメトリ データを収集し、その結果を受信側に転送します。フィルタはオプションです。使用しない場合、動作速度に関係なく、すべてのインターフェイスのカウンタが表示されます。

フィルタは、複数の速度をコンマ区切りのリストとして受け入れることができます。たとえば、`oper-speed=[1G,10G]` は、1 および 10 Gbps で動作するインターフェイスのカウンタを取得します。区切り文字として空白を使用しないでください。

（注） インターフェイス タイプ サブインターフェイス、ループバック、および VLAN には動作速度プロパティがないため、フィルタはこれらのインターフェイス タイプをサポートしません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **snsr-group** *sgrp_id* **sample-interval** *interval*
4. **path interface query-condition oper-speed=[***speed***]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

Step 4	**path interface query-condition oper-speed=[***speed***]**
	例：
	`switch(conf-tm-sensor)# path interface query-condition oper-speed=[1G,40G]`
	`switch(conf-tm-sensor)#`
		インターフェイス パス ラベルを設定し、指定された速度 (この例では 1 Gbps と 40 Gbps のみ) を実行しているインターフェイスからのカウンターを照会します。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

	コマンドまたはアクション	目的

Step 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

複数のクエリによるインターフェイス パスの構成

インターフェイスパスラベルの同じクエリ条件に対して複数のフィルタを構成できます。その場合、使用する個々のフィルタは AND で結合されます。

クエリ条件の各フィルタは、コンマを使用して区切ります。query-condition には、任意の数のフィルタを指定できますが、追加するフィルタが多いほど、結果の焦点が絞られることに注意してください。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

	コマンドまたはアクション	目的

Step 4	**path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
	例：
	`switch(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G]`
	`switch(conf-tm-sensor)#`
		同じクエリで複数の条件を構成します。この例では、クエリは次の両方を実行します。
		• 1 Gbps で実行されているインターフェイスでゼロ以外のカウンターを収集して返します。
		• 40 Gbps で実行されているインターフェイスでゼロ以外のカウンターを収集して返します。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリデータを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

Step 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

データまたはイベントをポーリングするための環境パスの構成

環境パス ラベルは、ファン、温度、電源、ストレージ、スーパーバイザ、ラインカードなどのシャーシ情報をモニタリングします。テレメトリ データを定期的にポーリングするか、イベントが発生したときにデータを取得するように環境パスを構成できます。詳細については、データの投票またはイベントの受信（412 ページ）を参照してください。

定期的なポーリングまたはイベントに基づいてシステム リソース情報を返すようにリソース パスを設定できます。このパスはフィルタリングをサポートしていません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path environment**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

Step 4	**path environment**
	例：
	`switch(conf-tm-sensor)# path environment`
	`switch(conf-tm-sensor)#`
		複数の個々の環境オブジェクトのテレメトリ データをラベルに送信できるようにする環境パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。
		サンプル間隔に応じて、環境データはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

	コマンドまたはアクション	目的

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリデータを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、環境イベントが発生したときに送信するかを決定します。

Step 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

イベントまたはデータをポーリングするためのリソース パスの構成

リソースパスは、CPU使用率やメモリ使用率などのシステムリソースをモニタリングします。このパスを構成して、テレメトリ データを定期的に収集するか、イベントが発生したときに収集できます。「データの投票またはイベントの受信（412 ページ）」を参照してください。

このパスはフィルタリングをサポートしていません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path resources**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(config-telemetry)# sensor-group 6`
	`switch(conf-tm-sensor)#`
		テレメトリ データのセンサー グループを作成します。

Step 4	**path resources**
	例：
	`switch(conf-tm-sensor)# path resources`
	`switch(conf-tm-sensor)#`
		複数の個々のシステム リソースのテレメトリ データをラベルに送信できるようにするリソース パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。
		サンプル間隔に応じて、リソース データはポーリング間隔に基づいてストリーミングされるか、システム メモリが「Not OK」に変更されたときに送信されます。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテ

	コマンドまたはアクション	目的
		レメトリ データを定期的に送信するか、リソースイベントが発生したときに送信するかを決定します。

Step 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

イベントまたはデータをポーリングするための VXLAN パスの構成

vxlan パス ラベルは、VXLAN ピア、VXLAN カウンター、VLAN カウンター、BGP ピア データなど、スイッチの仮想拡張 LAN EVPN に関する情報を提供します。このパス ラベルを構成して、定期的に、またはイベントが発生したときにテレメトリ情報を収集できます。「データの投票またはイベントの受信 （412 ページ）」を参照してください。

このパスはフィルタリングをサポートしていません。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **vxlan environment**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id* 例：	テレメトリ データのセンサー グループを作成します。

	コマンドまたはアクション	目的
	`switch(config-telemetry)# sensor-group 6` `switch(conf-tm-sensor)#`	

Step 4	**vxlan environment**
	例：
	`switch(conf-tm-sensor)# vxlan environment`
	`switch(conf-tm-sensor)#`
		複数の個々のVXLANオブジェクトのテレメトリデータをラベルに送信できるようにする vxlan パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。サンプル間隔に応じて、VXLANデータはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。

Step 5	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

Step 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリデータを構成して、指定された IP アドレスとポートにストリーミングします。

Step 7	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、VXLAN イベントが発生したときに送信するかを決定します。

Step 9	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

パス ラベル 構成 を確認

いつでも、パス ラベルが構成されていることを確認し、実行中のテレメトリ構成を表示してその値を確認できます。

手順の概要

1. **show running-config-telemetry**

手順の詳細

	コマンドまたはアクション	目的

Step 1	**show running-config-telemetry**
	例：
	`switch(conf-tm-sensor)# show running-config telemetry`
	
	`!Command: show running-config telemetry`
	`!Running configuration last done at: Mon Jun 10 08:10:17 2019`
	`!Time: Mon Jun 10 08:10:17 2019`
	
	`version 9.3(1) Bios:version`
	`feature telemetry`
	
	`telemetry`
	`  destination-profile`
	`    use-nodeid tester`
	`    sensor-group 4`
	`      path interface query-condition and(counters=[detailed],oper-speed=[1G,10G])`
	`    sensor-group 6`
	`      path interface query-condition oper-speed=[1G,40G]`
	`    subscription 6`
	`      snsr-grp 6 sample-interval 6000`
	`nxosv2(conf-tm-sensor)#`
		テレメトリの現在の実行構成を表示します。
		この例では、センサー グループ 4 は、1 および 10 Gbps で実行されているインターフェイスからゼロ以外のカウンターを収集するように構成されています。センサー グループ 6 は、1 と 40 Gbps で実行されているインターフェイスからすべてのカウンターを収集するように構成されています。

パス ラベル情報の表示

パス ラベル表示コマンド

show telemetry usability コマンドを使用すると、クエリを発行したときにパス ラベルがたどる個々のパスを表示できます。

コマンド	表示内容
show telemetry usability {all | environment | interface | resources | vxlan}	すべてのパス ラベルのすべてのテレメトリ パス、または指定されたパス ラベルのすべてのテレメトリ パス。また、出力には、各パスが定期的なポーリングまたはイベントに基づいてテレメトリ データを報告するかどうかが示されます。 インターフェイス パス ラベルには、設定したキーワード フィルタまたはクエリ条件も含まれます。
show running-config telemetry	テレメトリと選択されたパス情報の実行構成。

コマンドの例

（注） **show telemetry usability all** コマンドは、このセクションに示されている個々のコマンドをすべて連結したものです。

show telemetry usability environment コマンドの例を次に示します。

```
switch# show telemetry usability environment
  1) label_name          : environment

     path_name           : sys/ch
     query_type          : poll
     query_condition     :
rsp-subtree=full&query-target=subtree&target-subtree-class=eqptPsuSlot,eqptFtSlot,eqptSupCSlot,eqptPsu,eqptFt,eqptSensor,eqptLCSlot


  2) label_name          : environment

     path_name           : sys/ch
     query_type          : event
     query_condition     :
[illegible]
switch#
```

show telemetry usability interface コマンドの出力を次に示します。

```
switch# show telemetry usability interface
  1) label_name          : interface

     path_name           : sys/intf
     query_type          : poll
     query_condition     :
query-target=children&query-target-filter=eq(l1PhysIf.adminSt,"up")&rsp-subtree=children&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut


  2) label_name          : interface

     path_name           : sys/mgmt-[mgmt0]
     query_type          : poll
     query_condition     :
query-target=subtree&query-target-filter=eq(mgmtMgmtIf.adminSt,"up")&rsp-subtree=full&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut


  3) label_name          : interface

     path_name           : sys/intf
     query_type          : event
     query_condition     :
[illegible]
ethpmEncRtdIf.operSt,"down")),and(updated(ethpmEncRtdIf.operSt),eq(ethpmEncRtdIf.operSt,"up"))))


  4) label_name          : interface

     path_name           : sys/mgmt-[mgmt0]
     query_type          : event
     query_condition     :
query-target=subtree&query-target-filter=or(or(deleted(),created()),or(and(updated(mgmtIf.operSt),eq(mgmtIf.operSt,"down")),and(updated(mgmtIf.operSt),eq(mgmtIf.operSt,"up"))))
switch#
```

show telemetry usability resources コマンドの例を次に示します。

```
switch# show telemetry usability resources
  1) label_name          : resources

     path_name           : sys/proc
     query_type          : poll
     query_condition     : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  2) label_name          : resources

     path_name           : sys/procsys
     query_type          : poll
     query_condition     :
query-target=subtree&target-subtree-class=procSystem,procSysCpuSummary,procSysProcess,procSysCpuHistory,procSysLoad,procSysMem,procSysMemHistory,procSysVm,procSysVmHistory

  3) label_name          : resources

     path_name           : sys/procsys/sysmem
     query_type          : event
     query_condition     :
query-target-filter=and(updated(procSysMem.memstatus),ne(procSysMem.memstatus,"OK"))

switch#
```

show telemetry usability vxlan コマンドの例を次に示します。

```
switch# show telemetry usability vxlan
  1) label_name          : vxlan

     path_name           : sys/bd
     query_type          : poll
     query_condition     : query-target=subtree&target-subtree-class=l2VlanStats

  2) label_name          : vxlan

     path_name           : sys/eps
     query_type          : poll
     query_condition     : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  3) label_name          : vxlan

     path_name           : sys/eps
     query_type          : event
     query_condition     : query-target=subtree&target-subtree-class=nvoDyPeer

  4) label_name          : vxlan

     path_name           : sys/bgp
     query_type          : event
    query_condition     : query-target=subtree&query-target-filter=or(deleted(),created())

  5) label_name          : vxlan

     path_name           : sys/bgp
     query_type          : event
     query_condition     :
query-target=subtree&target-subtree-class=bgpDom,bgpPeer,bgpPeerAf,bgpDomAf,bgpPeerAfEntry,bgpOperRtctrlL3,bgpOperRtP,bgpOperRtEntry,bgpOperAfCtrl

switch#
```

ネイティブ データ送信元パス

ネイティブ データ送信元パスについて

NX-OSテレメトリは、特定のインフラストラクチャまたはデータベースに限定されないニュートラル データ 送信元であるネイティブ データ ソースをサポートします。代わりに、ネイティブ データ 送信元を使用すると、コンポーネントまたはアプリケーションをフックして、関連情報を発信テレメトリ ストリームに挿入できます。ネイティブ データ 送信元のパスはインフラストラクチャに属さないため、この機能は柔軟性を提供し、ネイティブ アプリケーションは NX-OS テレメトリと対話できます。

ネイティブ データ 送信元 パスを使用すると、特定のセンサー パスに登録して、セレクトしたテレメトリ データを受信できます。この機能は NX-SDK と連携して、次のパスからのテレメトリ データのストリーミングをサポートします。

- IP ルートのテレメトリ データを送信する RIB パス。
- 静的および動的 MAC エントリのテレメトリ データを送信する MAC パス。
- IPv4 と IPv6 隣接のテレメトリ データを送信する隣接関係パス。

サブスクリプションを作成すると、選択したパスのすべてのテレメトリ データが基準値として受信者にストリーミングされます。基準値の後、イベント通知のみが受信者にストリーミングされます。

ネイティブ データ 送信元 パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf（GPB）
- JavaScript Object Notation（JSON）
- コンパクト Google Protobuf (コンパクト GPB)

ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ

次の表は、各ソース パスについて、サブスクリプションが最初に作成されたとき（ベースライン）とイベント通知が発生したときにストリーミングされる情報を示しています。

Path Type	サブスクリプションベースライン	イベント通知（**Event Notifications**）
RIB	全てのルートの送信	

Path Type	サブスクリプションベースライン	イベント通知（Event Notifications）

		イベントの作成、更新、および削除に関するイベント通知を送信します。次の値は、RIBパスのテレメトリを介してエクスポートされます：
		• ネクスト ホップ ルーティング情報：
		  • ネクストホップのアドレス
		  • ネクストホップの発信インターフェイス
		  • ネクスト ホップの VRF 名
		  • ネクストホップの所有者
		  • ネクストホップの優先度
		  • ネクストホップのメトリック
		  • ネクストホップのタグ
		  • ネクストホップのセグメント 識別子
		  • ネクストホップのトンネル 識別子
		  • ネクストホップのカプセル化タイプ
		  • ネクスト ホップ タイプのフラグのビットごとの OR
		• レイヤ3のルーティング情報を検証する：
		  • ルートの VRF 名
		  • ルートプレフィックス アドレス

Path Type	サブスクリプションベースライン	イベント通知（Event Notifications）
		• ルートのマスク長 • ルートのネクストホップ数 • イベントの種類 • ネクスト ホップ
MAC	静的およびダイナミック MAC エントリに対して DME から GETALL を実行します。	イベントの追加、更新および削除に関するイベント通知を送信します。次の値は、MAC パスのテレメトリを通じてエクスポートされます： • MAC アドレス（MAC address） • MAC アドレス タイプ • VLAN番号 • インターフェイス名 • イベント タイプ イベント通知では、静的エントリとダイナミックエントリの両方がサポートされています。

Path Type	サブスクリプションベースライン	イベント通知（Event Notifications）
隣接	IPv4 および IPv6 隣接関係（アジャセンシー）を送信します。	イベントの追加、更新および削除に関するイベント通知を送信します。次の値は、隣接関係（アジャセンシー）パスのテレメトリを通じてエクスポートされます： • IP アドレス • MAC アドレス • インターフェイス名 • 物理インターフェイス名 • VRF 名 • プリファレンス • 隣接の送信元 • 隣接関係（アジャセンシー）のアドレスファミリ • 隣接関係（アジャセンシー）のイベント タイプ

詳細については、Github https://github.com/CiscoDevNet/nx-telemetry-proto を参照してください。

注意事項と制約事項

ネイティブ データ 送信元 パス機能には、次の注意事項と制約事項があります。

- RIB、MAC、および隣接関係（アジャセンシー）のネイティブ データ送信元パスからのストリーミングの場合、センサー パス プロパティの更新は、**depth**、**query-condition**あるいは、**filter-condition**などのカスタム基準をサポートしません。

ルーティング情報のネイティブ データ送信元パスの構成

URIB に含まれるすべてのルートに関する情報を送信するルーティング情報のネイティブ データ送信元 パスを構成できます。登録すると、基準値はすべてのルート情報を送信します。ベースラインの後、スイッチがサポートするルーティング プロトコルのルート更新と削除操作について通知が送信されます。RIB 通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ （429 ページ） を参照してください。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path rib**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(conf-tm-sub)# sensor-grp 6`
	`switch(conf-tm-sub)#`
		センサー グループを作成します。

Step 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

Step 5	**path rib**
	例：
	`nxosv2(conf-tm-sensor)# path rib`
	`nxosv2(conf-tm-sensor)#`
		ルートとルートアップデート情報をストリーミングする RIB パスを構成します。

	コマンドまたはアクション	目的

Step 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループサブモードに入り、接続先グループを構成します。

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データストリームのプロトコルとエンコードを設定します。

Step 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサーグループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。

Step 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

MAC 情報のネイティブ データ送信元パスの構成

MAC テーブルのすべてのエントリに関する情報を送信する MAC 情報のネイティブ データ 送信元 パスを構成できます。登録すると、基準値はすべての MAC 情報を送信します。基準値の後、MACアドレスの追加、更新、および削除操作の通知が送信されます。MAC通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ （429 ページ） を参照してください。

（注） 更新または削除イベントの場合、MAC 通知は、IP 隣接関係を持つ MAC アドレスに対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path mac**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(conf-tm-sub)# sensor-grp 6`
	`switch(conf-tm-sub)#`
		センサー グループを作成します。

Step 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

	コマンドまたはアクション	目的

Step 5	**path mac**
	例：
	`nxosv2(conf-tm-sensor)# path mac`
	`nxosv2(conf-tm-sensor)#`
		MACエントリおよびMAC通知に関する情報をストリームする MAC パスを構成します。

Step 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループサブモードに入り、接続先グループを構成します。

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

Step 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。

Step 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

すべての MAC 情報のネイティブ データ送信元パスの構成

レイヤ 3 およびレイヤ 2 から、MAC テーブルのすべてのエントリに関する情報を送信する MAC 情報のネイティブ データ 送信元 パスを構成できます。登録すると、基準値はすべての MAC 情報を送信します。基準値の後、MAC アドレスの追加、更新、および削除操作の通知が送信されます。MAC 通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ （429 ページ） を参照してください。

（注） 更新または削除イベントの場合、MAC 通知は、IP 隣接関係を持つ MAC アドレスに対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path mac-all**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

Step 3	**sensor-group** *sgrp_id* 例：	センサー グループを作成します。

	コマンドまたはアクション	目的
	switch(conf-tm-sub)# **sensor-grp 6** switch(conf-tm-sub)#	
Step 4	**data-source native** 例： switch(conf-tm-sensor)# **data-source native** switch(conf-tm-sensor)#	特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。
Step 5	**path mac-all** 例： nxosv2(conf-tm-sensor)# **path mac-all** nxosv2(conf-tm-sensor)#	すべての MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。
Step 6	**destination-group** *grp_id* 例： switch(conf-tm-sensor)# **destination-group 33** switch(conf-tm-dest)#	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** } 例： switch(conf-tm-dest)# **ip address 192.0.2.11 port 50001 protocol http encoding json** switch(conf-tm-dest)# 例： switch(conf-tm-dest)# **ip address 192.0.2.11 port 50001 protocol grpc encoding gpb** switch(conf-tm-dest)# 例： switch(conf-tm-dest)# **ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact** switch(conf-tm-dest)#	サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。
Step 8	**subscription** *sub_id* 例： switch(conf-tm-dest)# **subscription 33** switch(conf-tm-sub)#	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval* 例： switch(conf-tm-sub)# **snsr-grp 6 sample-interval 5000** switch(conf-tm-sub)#	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。

	コマンドまたはアクション	目的

Step 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

IP 隣接のネイティブ データ パスの構成

スイッチのすべての IPv4 と IPv6 隣接に関する情報を送信する IP 隣接情報のネイティブ データ送信元パスを構成できます。登録すると、基準値はすべての隣接情報を送信します。基準値の後、隣接操作の追加、更新、および削除に関する通知が送信されます。隣接関係通知で送信されるデータについては、ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ（429 ページ） を参照してください。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path adjacency**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的

Step 1	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		コンフィギュレーション モードを入力します。

Step 2	**telemetry**
	例：
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

	コマンドまたはアクション	目的

Step 3	**sensor-group** *sgrp_id*
	例：
	`switch(conf-tm-sub)# sensor-grp 6`
	`switch(conf-tm-sub)#`
		センサー グループを作成します。

Step 4	**data-source native**
	例：
	`switch(conf-tm-sensor)# data-source native`
	`switch(conf-tm-sensor)#`
		ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。

Step 5	**path adjacency**
	例：
	`nxosv2(conf-tm-sensor)# path adjacency`
	`nxosv2(conf-tm-sensor)#`
		IPv4 と IPv6 隣接に関する情報をストリームする隣接パスを構成します。

Step 6	**destination-group** *grp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 33`
	`switch(conf-tm-dest)#`
		テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

Step 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch(conf-tm-dest)#`
	例：
	`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch(conf-tm-dest)#`
		サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

Step 8	**subscription** *sub_id*
	例：
	`switch(conf-tm-dest)# subscription 33`
	`switch(conf-tm-sub)#`
		テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch(conf-tm-sub)#`
		センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、イン

	コマンドまたはアクション	目的
		ターフェイスイベントが発生したときに送信するかを決定します。

Step 10	**dst-group** *dgrp_id*
	例：
	`switch(conf-tm-sub)# dst-grp 33`
	`switch(conf-tm-sub)#`
		接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで設定した接続先グループと一致する必要があります。

ネイティブ データ ソース パス情報の表示

NX-OS の **show telemetry event collector** コマンドを使用して、ネイティブ データ ソース パスの統計情報とカウンタ、またはエラーを表示できます。

統計情報の表示

show telemetry event collector stats コマンドを発行して、各ネイティブ データ ソース パスの統計情報とカウンタを表示できます。

RIB パスの統計情報の例：

```
switch# show telemetry event collector stats

--------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
--------------------------------------------------------------------------------
1              4                 Mon Jul 01 13:53:42.384 PST rib(1)
switch#
```

MAC パスの統計情報の例：

```
switch# show telemetry event collector stats

--------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
--------------------------------------------------------------------------------
1              3                 Mon Jul 01 14:01:32.161 PST mac(1)
switch#
```

隣接パスの統計情報の例：

```
switch# show telemetry event collector stats

--------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
--------------------------------------------------------------------------------
1              7                 Mon Jul 01 14:47:32.260 PST adjacency(1)
switch#
```

エラー カウンタの表示

show telemetry event collector stats コマンドを使用して、すべてのネイティブ データ ソース パスのエラーの合計を表示できます。

```
switch# show telemetry event collector errors

-------------------------------------------------------------------------------
-
Error Description                                   Error Count
-------------------------------------------------------------------------------
-
Dme Event Subscription Init Failures                - 0
Event Data Enqueue Failures                         - 0
Event Subscription Failures                         - 0
Pending Subscription List Create Failures           - 0
Subscription Hash Table Create Failures             - 0
Subscription Hash Table Destroy Failures            - 0
Subscription Hash Table Insert Failures             - 0
Subscription Hash Table Remove Failures             - 0
switch#
```

ストリーミング Syslog

テレメトリ用のストリーミング Syslog について

Cisco NX-OS リリース 9.3(3) 以降、モデル駆動型テレメトリは、YANG をデータ ソースとして使用する syslog のストリーミングをサポートします。サブスクリプションを作成すると、すべての syslog が基準値として受信者にストリーミングされます。この機能は NX-SDK と連携して、次の syslog パスからのストリーミング syslog データをサポートします。

- Cisco-NX-OS-Syslog-oper:syslog
- Cisco-NX-OS-Syslog-oper:syslog/messages

基準値の後は、syslog イベント通知のみが受信者にストリーミングされます。syslog パスのストリーミングは、次のエンコーディング タイプをサポートします:

- Google Protobuf（GPB）
- JavaScript Object Notation（JSON）

Syslog 情報のための YANG データ ソース パスの構成

スイッチで生成されたすべての syslog に関する情報を送信する syslog の syslog パスを構成できます。サブスクライブすると、ベースラインはすべての既存の syslog 情報を送信します。ベースラインの後、通知は、スイッチで生成された新しい syslog に対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、**feature telemetry** コマンドで有効にします。

手順の概要

1. **configure terminal**

2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data source** *data-source-type*
5. **path Cisco-NX-OS-Syslog-oper:syslog/messages**
6. **destination-group** *grp_id*
7. ip address *ip_addr* **port** *port* **protocol** {**HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
8. **subscription** *sub-id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的
Step 1	**configure terminal** 例： `switch# configure terminal`	グローバル コンフィギュレーション モードを開始します。
Step 2	**telemetry** 例： switch(config)# **telemetry**	テレメトリの構成モードに入ります。
Step 3	**sensor-group** *sgrp_id* 例： `switch(config-telemetry)# sensor-group 6`	センサー グループを作成します。
Step 4	**data source** *data-source-type* 例： `switch(config-tm-sensor)# data source YANG`	データ ソースを YANG に設定し、ネイティブ YANG ストリーミング モデルを使用して syslog をストリーミングできるようにします。
Step 5	**path Cisco-NX-OS-Syslog-oper:syslog/messages** 例： `switch(config-tm-sensor)# path Cisco-NX-OS-Syslog-oper:syslog/messages`	スイッチで生成された syslog をストリーミングする syslog パスを設定します。
Step 6	**destination-group** *grp_id* 例： `switch(config-tm-sensor)# destination-group 33`	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
Step 7	ip address *ip_addr* **port** *port* **protocol** {**HTTP**	**gRPC** } **encoding** { **JSON**	**GPB**	**GPB-compact** } 例： `switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json` 例：	サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。

	コマンドまたはアクション	目的
	`switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`	
Step 8	**subscription** *sub-id* 例： `switch(config-tm-dest)# subscription 33`	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
Step 9	**snsr-group** *sgrp_id* **sample-interval** *interval* 例： `switch(config-tm-sub)# snsr-group 6 sample-interval 0`	センサーグループを現在のサブスクリプションにリンクし、データ サンプリングを 0 に設定して、syslog イベントが発生したときにスイッチがテレメトリ データを送信するようにします。*interval* については、0 のみが受け入れ可能な値です。
Step 10	**dst-group** *dgrp_id* 例： `switch(config-tm-sub)# dst-grp 33`	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、**destination-group** コマンドで構成した接続先グループとマッチする必要があります。

Syslog パスのテレメトリ データ ストリーミング

送信元パスごとに、次のテーブルは、サブスクリプションが最初に作成されるときの「ベースライン」において、そしてイベントの通知が発生するときに、どんな情報がストリーミングされるかを示しています。

パス	サブスクリプションベースライン	イベント通知
Cisco-NX-OS-Syslog-oper:syslog/messages	スイッチから既存のすべての syslog をストリーミングします。	スイッチで発生した syslog のイベント通知を送信します。 • message-id • node-name • time-stamp • time-of-day • time-zone • category • message-name • severity • text

syslog パス情報の表示

syslog パスの統計情報とカウンタ、またはエラーを表示するには、Cisco NX-OS の **show telemetry event collector** コマンドを使用します。

統計情報の表示

show telemetry event collector stats コマンドを入力すると、syslog パスごとの統計情報とカウンタを表示できます。

次に、syslog パスの統計情報の例を示します。

```
switch# show telemetry event collector stats

------------------------------------------------------------------------------
Row ID         Collection Count  Latest Collection Time     Sensor Path(GroupId)
------------------------------------------------------------------------------
1             138             Tue Dec 03 11:20:08.200 PST Cisco-NX-OS-Syslog-oper:syslog(1)

2              138               Tue Dec 03 11:20:08.200 PST
Cisco-NX-OS-Syslog-oper:syslog/messages(1)
```

エラー カウンタの表示

show telemetry event collector errors コマンドを使用すると、すべての syslog パスのエラーの合計を表示できます。

```
switch(config-if)# show telemetry event collector errors

------------------------------------------------------------------------------
Error Description                                 Error Count
------------------------------------------------------------------------------
Dme Event Subscription Init Failures                    - 0
Event Data Enqueue Failures                             - 0
Event Subscription Failures                             - 0
Pending Subscription List Create Failures               - 0
Subscription Hash Table Create Failures                 - 0
Subscription Hash Table Destroy Failures                - 0
Subscription Hash Table Insert Failures                 - 0
Subscription Hash Table Remove Failures                 - 0
```

JSON 出力の例

次に、JSON 出力のサンプルを示します。

```
172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
>>> URL            : /network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages
>>> TM-HTTP-VER    : 1.0.0
>>> TM-HTTP-CNT    : 1
>>> Content-Type   : application/json
>>> Content-Length : 578
    Path => Cisco-NX-OS-Syslog-oper:syslog/messages
            node_id_str   : task-n9k-1
            collection_id : 40
```

```
            data_source   : YANG
            data          :
[
  [
    {
      "message-id": 420
    },
    {
      "category": "ETHPORT",
      "group": "ETHPORT",
      "message-name": "IF_UP",
      "node-name": "task-n9k-1",
      "severity": 5,
      "text": "Interface loopback10 is up ",
      "time-of-day": "Dec 3 2019 11:38:51",
      "time-stamp": "1575401931000",
      "time-zone": ""
    }
  ]
]
```

-

KVGPB の出力例

次に KVGPB の出力例を示します。

```
KVGPB Output:
---Telemetry msg received @ 18:22:04 UTC

Read frag:1 size:339 continue to block on read..

All the fragments:1 read successfully total size read:339

node_id_str: "task-n9k-1"

subscription_id_str: "1"

collection_id: 374

data_gpbkv {

  fields {

    name: "keys"

    fields {

      name: "message-id"

      uint32_value: 374

    }

  }

  fields {
```

```
name: "content"
fields {
  fields {
    name: "node-name"
    string_value: "task-n9k-1"
  }
  fields {
    name: "time-of-day"
    string_value: "Jun 26 2019 18:20:21"
  }
  fields {
    name: "time-stamp"
    uint64_value: 1574293838000
  }
  fields {
    name: "time-zone"
    string_value: "UTC"
  }
  fields {
    name: "process-name"
    string_value: ""
  }
  fields {
    name: "category"
    string_value: "VSHD"
  }
  fields {
    name: "group"
    string_value: "VSHD"
  }
  fields {
    name: "message-name"
```

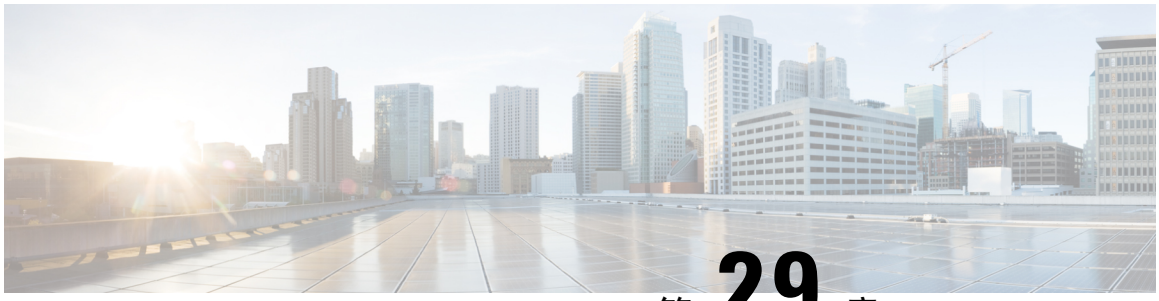
```
        string_value: "VSHD_SYSLOG_CONFIG_I"
      }
      fields {
        name: "severity"
        uint32_value: 5
      }
      fields {
        name: "text"
        string_value: "Configured from vty by admin on console0"
      }
    }
  }
}
```

-

その他の参考資料

関連資料

関連項目	マニュアル タイトル
VXLAN EVPN のテレメトリ展開の構成例。	*[VXLAN EVPN ソリューションのテレメトリ展開（Telemetry Deployment for VXLAN EVPN Solution）]*



第 29 章

OpenConfig YANG

OpenConfig YANG について

OpenConfig YANG は、宣言型の構成やモデル駆動型の管理と操作など、最新のネットワーキングの原則をサポートしています。OpenConfig は、ネットワークの構成とモニタリングのためにベンダーに依存しないデータ モデルを提供します。また、サブスクリプションとイベント更新ストリーミングにより、プル モデルからプッシュ モデルへの移行を支援します。

Cisco NX-OS リリース 9.2(1) 以降、幅広い機能エリアにわたってサポートが追加されています。これらには、BGP、OSPF、インターフェイス L2 と L3、VRF、VLAN、TACAC が含まれます。

OpenConfig YANG の詳細については、「OpenConfig YANG について」を参照してください。

Cisco NX-OS 9.2（1）の OpenConfig モデルについては、「YANG モデル 9.2(1)」を参照してください。OpenConfig YANG モデルは Cisco NX-OS リリースごとにグループ化されているため、Cisco NX-OS リリース番号が変更されると、URL の最後の桁が変更されます。

OpenConfig YANG のガイドラインと制限事項

OpenConfig YANG には、次のガイドラインと制限事項があります。

- IPv4 および IPv6 アドレスの場合、IP アドレス フィールド（**oc-ip:ip** および **oc-ip:prefix_length**）の削除と削除に同じ操作を提供する必要があります。

  例：

  ```
  oc-ip:ip: remove
  oc-ip:prefix_length: remove
  ```

- OSPF アクション メトリックが BGP **set med** プロパティよりも優先されるため、OpenConfig NETCONF を介して同じルート マップ内のメトリックを使用した**set med**と一緒のBGP アクションおよび OSPF アクションを設定することは推奨されません。

  2つの異なるルートマップを使用して、OSPFアクションでメトリックを設定します。個別のルート マップを使用して BGP アクションの下で**set-med** を使用します。

  単一のペイロードで、BGPアクションのメトリックをOSPFアクションに変更したり、OSPFアクションをルート マップの BGP アクションに変更したりしないことをお勧めします。

- 有効な BGP インスタンスを使用するには、自律システム (AS) 番号を指定する必要があります。AS 番号にデフォルト値が存在しないため、NETCONF / OPENCONFIG で削除しようとすると<asn> BGP インスタンスを削除しないと、次の強調表示されたエラー メッセージが表示されます。

```
764
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:config>
      <network-instances xmlns="http://openconfig.net/yang/network-instance">
        <network-instance>
          <name>default</name>
          <protocols>
            <protocol>
              <identifier>BGP</identifier>
              <name>bgp</name>
              <bgp>
                <global>
                  <config nc:operation="delete">
                    <as>100</as>
                  </config>
                </global>
                <neighbors>
                  <neighbor>
                    <neighbor-address>1.1.1.1</neighbor-address>
                    <enable-bfd xmlns="http://openconfig.net/yang/bfd">
                      <config>
                        <enabled>true</enabled>
                      </config>
                    </enable-bfd>
                  </neighbor>
                </neighbors>
              </bgp>
            </protocol>
          </protocols>
        </network-instance>
      </network-instances>
    </nc:config>
  </nc:edit-config>
</nc:rpc>

##
Received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">
    <rpc-error>
        <error-type>protocol</error-type>
```

```
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">invalid property value , for property asn, class
bgpInst</error-message>
        <error-path>/config/network-instances</error-path>
    </rpc-error>
    <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">invalid property value , for property asn, class
bgpInst Commit Failed</error-message>
        <error-path>/config/network-instances</error-path>
    </rpc-error>
</rpc-reply>
```

- OC-BGP-POLICY には、次の OpenConfig YANG 制限があります：
  - アクション タイプは、community-set および as-path-set に対して常に[許可（permit）]され、次のコンテナに適用されます。
    - /bgp-defined-sets/community-sets/community-set/
    - /bgp-defined-sets/as-path-sets/as-path-set/

    OpenConfig YANG には、community-set および as-path-set の CLI にあるようなアクション タイプの概念はありません。したがって、community-set および as-path-set のアクション タイプは常に permit です。
  - このコンテナには、次の OpenConfig YANG 制限が適用されます。
    /bgp-defined-sets/community-sets/community-set/

    CLI では、community-list には、標準と拡張の 2 つの異なるタイプがあります。ただし、OpenConfig YANG モデルでは、community-set-name にそのような区別はありません。

    OpenConfig YANG を使用して community-set-name を作成すると、次のことが内部で発生します。
    - community-member が標準形式（AS:NN）の場合、community-set-name の後に _std サフィックスが追加されます。
    - community-member が展開形式（正規表現）の場合、community-set-name の後に _exp サフィックスが追加されます。

```
<community-set>
    <community-set-name>oc_commset1d</community-set-name>
    <config>
        <community-set-name>oc_commset1d</community-set-name>
        <community-member>0:1</community-member>
        <community-member>_1_</community-member>
    </config>
</community-set>
```

    上記の OpenConfig YANG 構成は、次の CLI にマップされます。

```
ip community-list expanded oc_commset1d_exp seq 5 permit "_1_"
ip community-list standard oc_commset1d_std seq 5 permit 0:1
```

- このコンテナには、次の OpenConfig YANG 制限が適用されます。
  /bgp-conditions/match-community-set/config/community-set/

  OpenConfig YANG は 1 つのコミュニティ セットにのみマッピングできますが、CLI はコミュニティ セットの複数のインスタンスに一致できます。

  - CLI の場合：

```
ip community-list standard 1-1 seq 1 permit 1:1
  ip community-list standard 1-2 seq 1 permit 1:2
  ip community-list standard 1-3 seq 1 permit 1:3
route-map To_LC permit 10
  match community 1-1 1-2 1-3
```

  - 対応する OpenConfig YANG ペイロードは次のとおりです。

```
<config>
  <routing-policy xmlns="http://openconfig.net/yang/routing-policy">
    <defined-sets>
      <bgp-defined-sets xmlns="http://openconfig.net/yang/bgp-policy">
        <community-sets>
          <community-set>
            <community-set-name>cs</community-set-name>
            <config>
              <community-set-name>cs</community-set-name>
              <community-member>1:1</community-member>
              <community-member>1:2</community-member>
              <community-member>1:3</community-member>
            </config>
          </community-set>
        </community-sets>
      </bgp-defined-sets>
    </defined-sets>
    <policy-definitions>
      <policy-definition>
        <name>To_LC</name>
        <statements>
          <statement>
            <name>10</name>
            <conditions>
              <bgp-conditions xmlns="http://openconfig.net/yang/bgp-policy">
                <match-community-set>
                  <config>
                    <community-set>cs</community-set>
                  </config>
                </match-community-set>
              </bgp-conditions>
            </conditions>
          </statement>
        </statements>
      </policy-definition>
    </policy-definitions>
  </routing-policy>
</config>
```

回避策として、OpenConfig YANG を介して複数のステートメントを持つ 1 つのコミュニティを作成します。

```
ip community-list standard cs_std seq 5 permit 1:1
  ip community-list standard cs_std seq 10 permit 1:2
  ip community-list standard cs_std seq 15 permit 1:3
route-map To_LC permit 10
  match community cs_std
```

- 次の OpenConfig YANG 制限がこのコンテナに適用されます。
  `/bgp-conditions/state/next-hop-in`

  OpenConfig YANG では、`next-hop-in` タイプは IP アドレスですが、CLI では IP プレフィックスです。

  OpenConfig YANG を介して `next-hop-in` を作成する際、IP アドレスは CLI 設定で「/32」マスク プレフィックスに変換されます。例：

  - 以下は、OpenConfig YANG ペイロードの `next-hop-in` の例です。

```
<policy-definition>
    <name>sc0</name>
    <statements>
        <statement>
            <name>5</name>
            <conditions>
                <bgp-conditions xmlns="http://openconfig.net/yang/bgp-policy">

                    <config>
                        <next-hop-in>2.3.4.5</next-hop-in>
                    </config>
                </bgp-conditions>
            </conditions>
        </statement>
    </statements>
</policy-definition>
```

  - 以下は、CLI での同じ情報の例です。

```
ip prefix-list IPV4_PFX_LIST_OPENCONFIG_sc0_5 seq 5 permit 2.3.4.5/32
route-map sc0 permit 5
  match ip next-hop prefix-list IPV4_PFX_LIST_OPENCONFIG_sc0_5
```

- OC-BGP-POLICY には、次の NX-OS 制限があります。
  - `/bgp-actions/set-community/config/method` enum "REFERENCE" はサポートされていません。
  - `/bgp-actions/config/set-next-hop` の OpenConfig YANG モデルでサポートされている enum "SELF" はサポートされていません。
- OC-BGP-POLICY の場合、`/bgp-conditions/match-community-set/config/community-set` は、`match community <community-set>_std` にのみマップされるので、標準コミュニティのみがサポートされます。拡張コミュニティ セットへの一致はサポートされていません。

- タグセットの定義済みセットは現在実装されていないため、match-tag-set の置換には制限があります。

  現在、match-tag-set を置き換えると、値が追加されます。match-tag-set を置き換えるには、それを削除してから、もう一度作成します。

- FIPS には、OSPF OpenConfig YANG の注意事項および制約事項が適用されます。
  - OSPF でエリア構成を構成して削除すると、削除されたエリア (古いエントリ) が引き続き DME に表示されます。これらの古いエリア エントリは、OpenConfig YANG の GETCONFIG/GET 出力に表示されます。
  - OSPF ポリシー match ospf-area 構成の OpenConfig YANG でサポートされるエリアは 1 つだけです。CLI では、match ospf-area 100 101 など、複数のエリアに一致するように設定できます。ただし、OpenConfig YANG では、1 つのエリアのみを設定できます（たとえば、match ospf-area 100）。
  - エリア仮想リンクおよびエリアインターフェイス構成ペイロードは、同じエリア リストの下に置くことはできません。エリア コンテナ ペイロードを同じペイロード内の仮想リンク エリアとインターフェイス エリアとして分割します。
  - MD5 認証文字列は、OSPF OpenConfig YANG では構成できません。

    OSPF モデルでは、認証に対して認証タイプが定義されています。

    ```
    leaf authentication-type {
      type string;
      description
        "The type of authentication that should be used on this
        interface";
    }
    ```

    OSPF OpenConfig YANG は、認証パスワードのオプションをサポートしていません。
  - OSPF エリア認証構成はサポートされていません。たとえば、area 0.0.0.200 authentication message-digest は、OpenConfig YANG から設定できません。
  - デフォルトのネットワーク インスタンスでプロトコル コンテナを削除しても、デフォルトの VRF（たとえば、**router ospf 1/router bgp 1**）に該当する OSPF/BGP インスタンス設定は削除されません。

- 次に、OpenConfig ペイロードと Cisco Nexus 9000 インターフェイス間の VLAN 設定に関する注意事項と制限事項を示します。
  - トランク モード インターフェイスとトランク VLAN を同じ OpenConfig ペイロードで同時に構成しようとすると、構成が正常に完了しません。ただし、ペイロードを分割してトランク モード インターフェイスが最初に送信され、次にトランク VLAN が送信されると、構成は正常に完了します。

    Cisco NX-OS インターフェイスでは、インターフェイス モードのデフォルト値は **access** です。トランク関連の設定を実装するには、最初にインターフェイス モードを **trunk** に変更してから、トランク VLAN 範囲を設定する必要があります。これらの構成は、個別のペイロードで行います。

次の例は、トランク モードと VLAN 範囲を設定するための個別のペイロードを示しています。

例 1、インターフェイスをトランク モードに設定するペイロード。

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces xmlns="http://openconfig.net/yang/interfaces">
        <interface>
          <name>eth1/47</name>
          <subinterfaces>
            <subinterface>
              <index>0</index>
              <config>
                <index>0</index>
              </config>
            </subinterface>
          </subinterfaces>
          <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
            <switched-vlan xmlns="http://openconfig.net/yang/vlan">
              <config>
                <interface-mode>TRUNK</interface-mode>
              </config>
            </switched-vlan>
          </ethernet>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

例 2、VLAN 範囲を構成するペイロード。

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interfaces xmlns="http://openconfig.net/yang/interfaces">
        <interface>
          <name>eth1/47</name>
          <subinterfaces>
            <subinterface>
              <index>0</index>
              <config>
                <index>0</index>
              </config>
            </subinterface>
          </subinterfaces>
          <ethernet xmlns="http://openconfig.net/yang/interfaces/ethernet">
            <switched-vlan xmlns="http://openconfig.net/yang/vlan">
              <config>
                <native-vlan>999</native-vlan>
                <trunk-vlans xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
nc:operation="delete">1..4094</trunk-vlans>
                <trunk-vlans>401</trunk-vlans>
                <trunk-vlans>999</trunk-vlans>
              </config>
            </switched-vlan>
```

```
          </ethernet>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

- OpenConfig YANG の設計により、VLAN を設定する場合、ペイロード内の VLAN とインターフェイスですでに設定されている VLAN との間に重複があってはなりません。オーバーラップが存在する場合、OpenConfig による構成は失敗します。インターフェイスに設定されている VLAN が、OpenConfig ペイロードの VLAN と異なることを確認してください。範囲内の開始 VLAN と終了 VLAN に特に注意してください。

- 次の注意事項および制約事項が OC-LACP に適用されます。
  - ポートチャネル モード：
    - OC-LACP を使用すると、ポートチャネル インターフェイスでポートチャネル モードを設定できます。ただし、NXOS-CLI を通じて、ポートチャネル モードは、チャネル グループ モードのアクティブまたはパッシブを使用してメンバーインターフェイスで設定されます。
    - OC-LACP はポートチャネル インターフェイスでポート チャネル モードを明示的に設定しますが、ポートチャネル インターフェイスで NX-OS **show running-config** コマンドを発行しても、空または空でないポートチャネルのポートチャネル モード設定は表示されません。
    - メンバーがポートチャネルに追加されると、**show running interface ethernet <>** はポート チャネル モードの構成をチャネル グループ モードのアクティブまたはパッシブとして表示されます。

（注） OpenConfig を介して作成されたすべてのポートチャネルは、引き続き OpenConfig によって管理される必要があります。

  - ポートチャネルの間隔：
    - ポートチャネルの間隔は、メンバーがシャット状態の場合にのみ変更できます。
    - OC-LACP 間隔はポートチャネルごとです。NX-OS LACP 間隔は、ポートチャネル メンバーごとです。この違いにより、次の動作が予想されます。
      - OpenConfig を使用してポートチャネル間隔を設定すると、ポートチャネルのすべてのメンバーに同じ設定が適用されます。
      - OpenConfig を使用してポートチャネル間隔を構成し、後でメンバーがポートチャネルに追加された場合、設定を新しいメンバーに適用するには、OpenConfig を使用して間隔を再度設定する必要があります。
  - システム MAC ID：

- このリリースでは、Cisco NX-OS はポートチャネルごとの `system-id-mac` をサポートしていません。

- 次のメンバー状態データは、ポートが管理 `up` state:状態の場合にのみ存在します。
  - LACP
  - インターフェイス
  - インターフェイス
  - メンバー
  - 状態
- OpenConfig YANG を介してインターフェイスを追加しようとすると、OSPFv2 はエラー応答を送信できます。問題が発生すると、インターフェイスは追加されず、RPC 応答には次のように「リストのマージに失敗しました（list merge failed）」というエラーが含まれます。

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:39507023-8569-4cf8-869c-e19aaf76a260">
    <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
       <error-message xml:lang="en">List Merge Failed: operation-failed</error-message>


<error-path>/network-instances/network-instance/protocols/protocol/ospfv2/areas/area/interfaces/interface/id</error-path>

    </rpc-error>
</rpc-reply>
```

- Hig (ii) ポートの統計のキューイングはサポートされていません。
- ユニキャスト、マルチキャスト、またはブロードキャストキューごとの tx パケット、またはバイト、およびドロップパケットは表示されません。OC 応答に表示される統計は、qos-group ごとの ucast、mcast、および bcast キューの合計です。
- OpenConfig YANG は、VLAN レベルで適用される QoS ポリシーの統計をサポートしていません。
- OC を介して取得できる入力キュー ドロップ数は、プラットフォームに応じてスライス/ポート/キュー レベルで表示できます。
- 以下は、switchport、shut/no shut、MTU、および MAC アドレスの OpenConfig 設定のガイドラインと制限です。
  - スイッチポート、shut/no shut、MTU、および MAC アドレスを設定する場合は、ASCII リロードが必要です。バイナリ リロードを使用すると、構成が失われます。
- 次の状態コンテナは、インターフェイス参照レベルの OpenConfig ACL に実装されています。

- acl/interfaces/state コンテナの /acl/interfaces/interface/interface-ref/state。
- read-onlyoc-if:interface リーフの acl/interfaces/interface/interface-ref/state/interface。
- read-onlyoc-if:subinterface リーフの acl/interfaces/interface/interface-ref/state/subinterface。

- 次のシステム構成コンテナは、ドメイン名、ログイン バナー、および motd-バナー モデルに実装されています。
  - /system/config/domain-name for /top:System/top:dns-items/top:prof-items/top:Prof-list/top:dom-items/top:name container
  - system/config/login-banner for /top:System/top:userext-items/top:postloginbanner-items/top:message container
  - system/config/login-banner for /top:System/top:userext-items/top:postloginbanner-items/top:message container

- 次の新しい動作状態 OpenConfig パスがサポートされています。一部のパスには、次に示す追加の注意事項と制限事項があります。
  - /network-instances/network-instance/fdb/l2rib/mac-table
    - l2rib の親レベルのクエリは、l2rib レベルでサポートされています。たとえば、network-instances/network-instance/fdb/l2rib まではクエリできますが、fdb レベルの network-instances/network-instance/fdb ではクエリできません。
  - /interfaces/interface/routed-vlan/ipv4/neighbors/neighbor/state
  - /interfaces/interface/routed-vlan/ipv6/neighbors/neighbor/state
    - 親クエリの場合、インフラストラクチャはすべてのリスト項目のすべてのキーを取得し、これらの各リスト項目の残りのデータを入力する要求がバックエンドに送信されます。つまり、インフラストラクチャにはバックエンドと同じツリービューが必要です。

      たとえば、インフラストラクチャには静的エントリのみがあり、バックエンドには静的エントリと動的エントリがある場合、リスト ウォークでは、インフラストラクチャは静的エントリごとの要求のみを送信するので、不完全なデータになります。現在のリリースでこの制限があるパスは、/interfaces/interface/routed-vlan/ipv6/neighbors/neighbor/state および /interfaces/interface/routed-vlan/ipv4/neighbors/neighbor/state です。正確なパスが指定されている場合、データには動的と静的の両方の ARP およ

び ND エントリが含まれますが、親パスが指定されている場合は静的エントリのみが含まれます。

- /network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/l2vpn-evpn/loc-rib/routes
- /network-instances/network-instance/protocols/protocol/bgp/rib/attr-sets
- /network-instances/network-instance/protocols/protocol/bgp/rib/communities
- /network-instances/network-instance/protocols/protocol/bgp/rib/ext-communities
- /network-instances/network-instance/connection-points/connection-point/endpoints/endpoint/vxlan/endpoint-peers
- /network-instances/network-instance/connection-points/connection-point/endpoints/endpoint/vxlan/endpoint-vnis

BGP ルーティング インスタンスの削除について

OpenConfig YANG ネットワーク インスタンス（OCNI）を使用して、BGP ルーティング インスタンス全体を削除するのではなく、デフォルトの VRF の BGP 構成のみを削除しようとすると、プロトコル/BGP レベルで BGP 情報が削除されないことがあります。この状況では、ペイロードに自律システム番号を含むプロトコルまたは BGP レベルで削除が行われると、BGP ルーティング インスタンス全体が削除されるのではなく、デフォルトの VRF の設定のみが削除されます。

以下は、BGP のデフォルト VRF で設定を削除するために使用されるペイロードの例です。

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <network-instances xmlns="http://openconfig.net/yang/network-instance">
        <network-instance>
          <name>default</name>
          <protocols>
            <protocol>
              <identifier>BGP</identifier>
              <name>bgp</name>
             <bgp xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" nc:operation="delete">

                 <global>
                   <config>
                     <as>100</as>
                   </config>
                 </global>
               </bgp>
             </protocol>
           </protocols>
         </network-instance>
       </network-instances>
     </config>
  </edit-config>
</rpc>
```

予期される動作：BGP ルーティング インスタンス自体を削除する必要があります。これは、**no router bgp 100** と同等です。

実際の動作：デフォルト VRF の BGP 構成のみが削除され、同等の単一の CLI 構成はありません。

削除操作前の実行構成は次のとおりです。

```
router bgp 100
  router-id 1.2.3.4
  address-family ipv4 unicast
  vrf abc
    address-family ipv4 unicast
      maximum-paths 2
```

削除操作後の実行構成は次のとおりです。

```
router bgp 100
  vrf abc
    address-family ipv4 unicast
      maximum-paths 2
```

YANG の検証

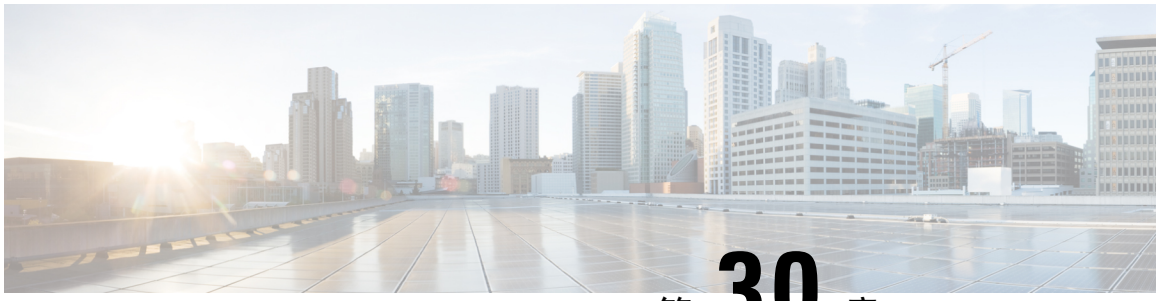
YANG 設定を検証するには、次のコマンドを使用します。

表 *26 : YANG* 検証

コマンド	説明
show telemetry yang direct-path cisco-nxos-device	サポートされているパスを表示します。

第 V 部

XML 管理インターフェイス



第 30 章

XML 管理インターフェイス

XML 管理インターフェイスについて

XML 管理インターフェイスについての情報

XML 管理インターフェイスを使用してデバイスを構成できます。インターフェイスは XML ベースのネットワーク構成プロトコル（NETCONF）を使用します。これにより、デバイスを管理し、インターフェイスを介して XML 管理ツールまたはプログラムと通信できます。NETCONF の Cisco NX-OS 導入では、デバイスとの通信にセキュア シェル（SSH）セッションを使用する必要があります。

NETCONF は、リモート プロシージャ コール（RPC）メッセージ内にデバイス構成要素を含めることができる XML Schema（XSD）を使用して導入されます。RPC メッセージ内から、デバイスに実行させたいコマンドのタイプに一致する NETCONF 操作の 1 つを選択します。NETCONF を使用して、デバイスで CLI コマンドのセット全体を設定できます。NETCONF の使用については、NETCONF XML インスタンスの作成, on page 471 と RFC 4741 を参照してください。

SSH を介した NETCONF の使用の詳細については、RFC 4742 を参照してください。

このセクションは、次のトピックで構成されています。

NETCONF レイヤ

次のテーブルは NETCONF レイヤのリストです：

Table 27: NETCONF レイヤ

レイヤ	例
トランスポート プロトコル	SSHv2
RPC	RPC、RPC-reply
操作	get-config、edit-config
コンテンツ	**show** または **configuration** コマンド

以下は、4 つの NETCONF レイヤの説明です。

- SSH トランスポート プロトコル：クライアントとサーバー間の暗号化接続を提供します。
- RPC タグ：リクエスタからの構成コマンドと、それに対応する XML サーバからの応答を導入します。
- NETCONF 操作タグ：構成コマンドのタイプを示します。
- 格納ファイル — 構成する機能の XML 表現を示します。

SSH xmlagent

デバイス ソフトウェアは、SSH バージョン 2 を介した NETCONF をサポートする xmlagent と呼ばれる SSH サービスを提供します。

Note xmlagent サービスは、Cisco NX-OS ソフトウェアでは XML サーバと呼ばれます。

NETCONF over SSH は、クライアントと XML サーバ間の hello メッセージの交換から始まります。最初の交換の後、クライアントは XML 要求を送信し、サーバは XML 応答で応答します。クライアントとサーバは、文字シーケンス > で要求と応答を終了します。この文字シーケンスは XML では有効ではないため、クライアントとサーバはメッセージがいつ終了するかを解釈でき、通信の同期が維持されます。

使用できる XML 構成インスタンスを定義する XML スキーマについては、NETCONF XML インスタンスの作成, on page 471 で説明されています。

XML 管理インターフェイスのライセンス要件

製品	ライセンス要件
Cisco NX-OS	XML管理インターフェースにはライセンスは必要ありません。ライセンス パッケージに含まれていない機能は Cisxo NX-OS のイメージにバンドルされており、無料で提供されます。NX-OS ライセンス方式の詳細については、『*Cisco NX-OS Licensing Guide*』を参照してください。

XML 管理インターフェイスを使用するための前提条件

XML 管理インターフェイスを使用するには、次の前提条件があります。

- クライアント PC に SSHv2 をインストールする必要があります。
- クライアント PC に NETCONF over SSH をサポートする XML 管理ツールをインストールする必要があります。
- デバイスの XML サーバに適切なオプションを設定する必要があります。

XML 管理インターフェイスを使用

このセクションでは、XML管理インターフェイスを手動で構成して使用する方法について説明します。

Note デバイスのデフォルト設定で XML 管理インターフェイスを使用します。

CLI による SSH および XML サーバー オプションの構成

デフォルトで SSH サーバーが有効になっています。SSH を無効にする場合は、クライアント PC で SSH セッションを開始する前に有効にする必要があります。

XML サーバー オプションを構成して、同時セッションの数とアクティブ セッションのタイムアウトを制御できます。XML ドキュメントの検証を有効にして、XML セッションを終了することもできます。

Note XML サーバ タイムアウトはアクティブ セッションだけに適用できます。

SSH の構成の詳細については、ご使用のプラットフォームの Cisco NX-OS セキュリティ構成ガイドを参照してください。

XML コマンドの詳細については、ご使用のプラットフォームの Cisco NX-OS システム マネジメント 構成ガイドを参照してください。

Step 1 グローバル コンフィギュレーション モードを開始します。

configure terminal

Step 2 （オプション）XML サーバー構成およびアクティブ XML サーバー セッションに関する情報を表示します。コマンド出力でセッション番号を確認できます。

show xml server status

Step 3 指定されたサーバー セッションの XML ドキュメントを検証します。

xml server validate all

Step 4 指定した XML サーバ セッションを終了します。

xml server terminate *session*

Step 5 （オプション）キーを生成できるように SSH サーバーを無効にします。

no feature ssh

Step 6 SSH サーバをイネーブルにします。（デフォルトでは有効になっています。）

feature ssh

Step 7 （オプション）SSH サーバーのステータスを表示します。

show ssh server

Step 8 許容される XML サーバーのセッション数を設定します。

xml server max-session *sessions*

有効な範囲は 1 ～ 8 です。デフォルト値は 8 です。

Step 9 XML サーバー セッションを終了するまでに待機する秒数を設定します。

xml server timeout *seconds*

範囲は 1～1200 です。デフォルトは 1200 秒です。

Step 10 （オプション）XML サーバー設定およびアクティブ XML サーバー セッションに関する情報を表示します。

show xml server status

Step 11 （任意）実行コンフィギュレーションをスタートアップ コンフィギュレーションに保存します。

copy running-config startup-config

Example

次の例は、CLI を通して SSH と XML サーバー オプションを構成する方法の例を示しています：

```
switch# configure terminal
switch(config)# xml server validate all
switch(config)# xml server terminate 8665
switch(config)# no feature ssh
switch(config)# feature ssh server
switch(config)# xml server max-session 6
switch(config)# xml server timeout 2400
switch(config)# copy running-config startup-config
```

SSHv2 セッションの開始

クライアント PC で SSHv2 セッションを開始するには、次のような **ssh2** コマンドを使用します。

```
ssh2 username@ip-address -s xmlagent
```

ログインユーザー名、デバイスの IP アドレス、接続するサービスを入力します。xmlagent サービスは、デバイス ソフトウェアでは XML サーバと呼ばれます。

Note SSH コマンドの構文は、クライアント PC の SSH ソフトウェアによって異なることがあります。

XML サーバから hello メッセージを受信しなかった場合は、次の条件を確認してください。

- デバイスで SSH サーバがイネーブルになっています。
- XML サーバの *max-sessions* オプションは、デバイスへの SSH 接続の数をサポートするのに十分です。
- デバイス上の現用系 XML サーバ セッションの一部が使用されていません。

hello メッセージの送信

サーバーが他の要求を処理する前に、hello メッセージを使用して、サーバーに自分の機能をアドバタイズする必要があります。XML サーバーへの SSH セッションを開始すると、サーバはすぐに hello メッセージで応答し、サーバーの機能をクライアントに通知しますこのメッセージは、サーバーの機能をクライアントに通知します。XML サーバーは基本機能のみをサポートしているので、クライアントに対しても、基本機能のみのサポートを想定しています。

以下は、サーバーとクライアントからのサンプルの hello メッセージです。

Note すべての XML ドキュメントは、]]>]]> で終了して、SSH 経由の NETCONF で同期がサポートされるようにする必要があります。

サーバーからの hello メッセージ

```
<?xml version="1.0"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
  <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  </capabilities>
  <session-id>25241</session-id>
</hello>]]>]]>
```

クライアントからの hello メッセージ

```
<?xml version="1.0"?>
<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
  <nc:capability>urn:ietf:params:xml:ns:netconf:base:1.0</nc:capability>
  </nc:capabilities>
</nc:hello>]]>]]>
```

XML スキーマ定義（XSD）ファイルの取得

Step 1 switch# feature bash shell

Step 2 switch# run bash

Step 3 bash-3.2$ cd /isan/etc/schema

Step 4 必要なスキーマを取得します。

XML ドキュメントを XML サーバに送信する

コマンド シェルで開いた SSH セッションを介して XML ドキュメントを XML サーバに送信するには、エディターからXMLテキストをコピーして、SSHセッションに貼り付けます。通常、XML ドキュメントを XML サーバに送信するには自動化されたメソッドを使用しますが、このコピーと貼り付けによる方法を使えば、XML サーバへの SSH 接続を確認できます。

XML ドキュメントを XML サーバに送信する場合のガイドラインは次のとおりです。

- コマンド シェル出力で Hello メッセージ テキストを検索して、SSH セッションを開始した直後に XML サーバが Hello メッセージを送信したことを確認します。
- XML 要求を送信する前に、クライアントの Hello メッセージを送信します。クライアントが Hello メッセージを送信すると、XML サーバは Hello 応答を直ちに送信し、追加の応答は送信しないことに注意してください。
- XML ドキュメントは常に文字シーケンス]]>]]> で終了します。

NETCONF XML インスタンスの作成

NETCONF XML インスタンスは、RPC タグおよび NETCONF 操作タグで XML デバイス要素を囲むことにより作成できます。XML デバイス要素は、使用可能な CLI コマンドを XML フォーマットで囲む機能ベースの XML スキーマ定義（XSD）ファイルで定義されます。

以下は、フレームワーク コンテキストの NETCONF XML リクエストで使用されるタグです。タグ ラインは次のレター コードでマークキングされています：

- X — XML 宣言
- R — RPC リクエスト タグ
- N — NETCONF 操作タグ
- D — デバイス タグ

NETCONF XML フレームワークのコンテキスト

```
X <?xml version="1.0"?>
R <nc:rpc message-id="1" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
R xmlns="http://www.cisco.com/nxos:1.0:nfcli”>
N <nc:get>
N <nc:filter type="subtree">
D <show>
D <xml>
D <server>
D <status/>
D </server>
D </xml>
D </show>
N </nc:filter>
N </nc:get>
R </nc:rpc>]]>]]>
```

Note 任意の XML エディタまたは XML 管理インターフェイス ツールを使用して、XML インスタンスを作成する必要があります。

RPC リクエスト タグ

すべての NETCONF XML インスタンスは、RPC リクエスト タグ <rpc> で開始する必要があります。<rpc> 要素には、メッセージ ID（message-id）属性があります。message-id 属性は、<rpc-reply> 内に複製され、リクエストと応答を関連付けるために使用できます。<rpc> ノードもまた、次の XML 名前空間宣言も含まれています。

- NETCONF 名前空間宣言：`urn:ietf:params:xml:ns:netconf:base:1.0` 名前空間で定義されている <rpc> と NETCONF タグは、netconf.xsd スキーマ ファイルに存在します。
- デバイスの名前空間宣言：<rpc> と NETCONF タグによってカプセル化されたデバイス タグは、他の名前空間で定義されています。デバイスの名前空間は機能指向です。Cisco NX-OS 機能タグは、さまざまな名前空間で定義されています。RPC リクエスト タグ <rpc> は、NFCLI

機能の使用例です。これは、デバイスの名前空間が xmlns=http://www.cisco.com/nxos:1.0:nfcli であることを宣言しています。nfcli.xsd には、この名前空間の定義が含まれています。詳細については、XML スキーマ定義（XSD）ファイルの取得, on page 470を参照してください。

例

RPC リクエスト タグ <rpc>

```
<nc:rpc message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
...
</nc:rpc>]]>]]>
```

構成リクエスト

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns="http://www.cisco.com/nxos:1.0:if_manager">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:config>
      <configure>
        <__XML__MODE__exec_configure>
          <interface>
            <ethernet>
              <interface>2/30</interface>
              <__XML__MODE_if-ethernet>
                <__XML__MODE_if-eth-base>
                  <description>
                    <desc_line>Marketing Network</desc_line>
                  </description>
                </__XML__MODE_if-eth-base>
              </__XML__MODE_if-ethernet>
            </ethernet>
          </interface>
        </__XML__MODE__exec_configure>
      </configure>
    </nc:config>
  </nc:edit-config>
</nc:rpc>]]>]]>
```

Note __XML__MODE タグは、NETCONF エージェントによって内部的に使用されます。一部のタグは、特定の __XML__MODE の子としてのみ存在します。スキーマ ファイルを調べると、XML で CLI コマンドを表すタグにつながる正しいモード タグを見つけることができます。

NETCONF 動作タグ

NETCONF は、次の構成動作を提供します。

Table 28: Cisco NX-OS の NETCONF 動作

NETCONF 動作	説明	例
close-session	XML サーバー セッションを閉じます。	NETCONF クロース セッション インスタンス, on page 482
commit	候補構成の現在の内容を実行構成として設定します。	NETCONF コミット インスタンス：候補構成機能, on page 487
confirmed-commit	指定された時間に構成を確定するためのパラメータを指定します。この操作の後の、確認タイムアウト期間内にコミットが行われなかった場合、構成はコミット確認操作の前の状態に戻ります。	NETCONF 確認済みコミット インスタンス, on page 487
copy-config	送信元構成データストアの内容をターゲットデータストアにコピーします。	NETCONF での構成インスタンスのコピー, on page 483
delete-config	操作がサポートされていません。	—
edit-config	デバイスの実行構成の機能を構成します。この操作は **configuration** コマンドに使用します。	NETCONF Edit Config インスタンス, on page 483 NETCONF Rollback-On-Error インスタンス, on page 488
get	デバイスから構成情報を受信します。この操作は **show** コマンドに使用します。データのソースは実行コンフィギュレーションです。	NETCONF XML インスタンスの作成, on page 471
get-config	構成の全体または一部を取得します。	NETCONF XML インスタンスの作成, on page 471
kill-session	指定した XML サーバー セッションを閉じます。自分のセッションを閉じることはできません。	NETCONF 強制終了セッション インスタンス, on page 482

NETCONF 動作	説明	例
ロック	クライアントがデバイスの構成システムをロックできるようにします。	NETCONF ロック インスタンス, on page 486
unlock	セッションが発行した構成ロックを解放します。	NETCONF ロック解除インスタンス, on page 486
検証	構成をデバイスに適用する前に、構成候補の構文エラーおよびセマンティクス エラーをチェックします。	NETCONF 検証機能インスタンス, on page 488

デバイスタグ

XML デバイス要素は、使用可能な CLI コマンドを XML フォーマットで表します。機能固有のスキーマ ファイルには、その特定の機能の CLI コマンドの XML タグが含まれています。「XML スキーマ定義（XSD）ファイルの取得, on page 470」を参照してください。

このスキーマを使用して、XML インスタンスを構築することができます。NETCONF インスタンスのビルドに使用された nfcli.xsd スキーマ ファイルの関連部分です。（NETCONF XML インスタンスの作成, on page 471 を参照）。

xml デバイス タグを表示します。

```
<xs:element name="show" type="show_type_Cmd_show_xml"/>
<xs:complexType name="show_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>to display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:choice maxOccurs="1">
<xs:element name="xml" minOccurs="1" type="xml_type_Cmd_show_xml"/>
<xs:element name="debug" minOccurs="1" type="debug_type_Cmd_show_debug"/>
</xs:choice>
</xs:sequence>
<xs:attribute name="xpath-filter" type="xs:string"/>
<xs:attribute name="uses-namespace" type="nxos:bool_true"/>
</xs:complexType>
```

サーバー ステータス デバイス タグ

```
<xs:complexType name="xml_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="server" minOccurs="1" type="server_type_Cmd_show_xml"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="server_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent server</xs:documentation>
</xs:annotation>
```

```
<xs:sequence>
<xs:choice maxOccurs="1">
<xs:element name="status" minOccurs="1" type="status_type_Cmd_show_xml"/>
<xs:element name="logging" minOccurs="1" type="logging_type_Cmd_show_logging_facility"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
```

デバイスタグの応答

```
<xs:complexType name="status_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="__XML__OPT_Cmd_show_xml___readonly__" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:group ref="og_Cmd_show_xml___readonly__" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:group name="og_Cmd_show_xml___readonly__">
<xs:sequence>
<xs:element name="__readonly__" minOccurs="1" type="__readonly___type_Cmd_show_xml"/>
</xs:sequence>
</xs:group>
<xs:complexType name="__readonly___type_Cmd_show_xml">
<xs:sequence>
<xs:group ref="bg_Cmd_show_xml_operational_status" maxOccurs="1"/>
<xs:group ref="bg_Cmd_show_xml_maximum_sessions_configured" maxOccurs="1"/>
<xs:group ref="og_Cmd_show_xml_TABLE_sessions" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
```

Note 「__XML__OPT_Cmd_show_xml___readonly__」タグはオプションです。このタグは応答を表します。応答の詳細については、RPC 応答タグ, on page 480 を参照してください。

<get>操作を実行するために使用できるタグを見つけるための|XMLオプションを使用できます。以下は|XMLオプションの例です。この例では、このデバイスで操作を実行するための名前空間定義タグが`http://www.cisco.com/nxos:1.0:nfcli`であり、要求のビルドのためにnfcli.xsdファイルを使用できることがわかります。

NETCONF 操作タグとデバイス タグを RPC タグで囲むことができます。</rpc> 終了タグの後に XML 終了文字シーケンスが続きます。

XML の例

```
Switch#> show xml server status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:nfcli">
<nf:data>
<show>
<xml>
```

```
<server>
<status>
<__XML__OPT_Cmd_show_xml___readonly__>
<__readonly__>
<operational_status>
<o_status>enabled</o_status>
</operational_status>
<maximum_sessions_configured>
<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML__OPT_Cmd_show_xml___readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

拡張された NETCONF の操作

Cisco NX-OS は、<exec-command> という名前の<rpc> 操作をサポートします。この操作により、クライアントアプリケーションはCLIの **構成**コマンドと**表示**コマンドを送信し、それらのコマンドへの応答を XML タグとして受信できます。

以下は、インターフェイスの構成に使用されるタグの例です。タグラインは次のレターコードでマークキングされています：

- X — XML 宣言
- R — RPC リクエスト タグ
- EO — 拡張操作

次の表に、操作タグの詳細な説明を示します。

Table 29: 操作タグ

タグ	説明
<exec-command>	CLI コマンドの実行
<cmd>	CLI コマンドが含まれています。コマンドは、**表示** または **構成** コマンドです。複数の構成コマンドは、セミコロン（;）を使用して区切ります。複数の **show** コマンドはサポートされていませんが、複数の構成コマンドを異なる <cmd> タグに含めて、同じ要求の一部として送信することはできます。詳細については、*<exec-command>* を通して送信される構成 *CLI* コマンドの例を参照してください。

を介して送信される構成コマンドへの応答 <cmd> タグは次のとおりです。

- <nf:ok>：すべての **configuration** コマンドが正常に実行されました。
- <nf:rpc-error>：一部のコマンドが失敗しました。操作は最初のエラーで停止します。<nf:rpc-error> サブツリーは、失敗した構成について詳細情報を提供します。失敗したコマンドの前に実行された構成は、実行構成に適用されていることに注意してください。

<exec-command> を通して送信される構成 CLI コマンド

show コマンドは、次の例に示すように、それ自体の <exec-command> インスタンスに含めて送信する必要があります。

```
X <?xml version="1.0"?>
R <nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
EO <nxos:exec-command>
EO <nxos:cmd>conf t ; interface ethernet 2/1 </nxos:cmd>
EO <nxos:cmd>channel-group 2000 ; no shut; </nxos:cmd>
EO </nxos:exec-command>
R </nf:rpc>]]>]]>
```

<exec-command> を通して送信された CLI コマンドへの応答

送信操作に対する応答は次のとおりです。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]>]]>
```

<exec-command> を通して送信された表示 CLI コマンド

次の例は、<exec-command> を通して送信される **show** CLI コマンドによってデータを取得する方法を示しています。

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show interface brief</nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
```

<exec-command> を通して送信された表示 CLI コマンドへの応答

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0"
xmlns:mod="http://www.cisco.com/nxos:1.0:if_manager" message-id="110">
```

```
<nf:data>
<mod:show>
<mod:interface>
<mod:__XML__OPT_Cmd_show_interface_brief___readonly__>
<mod:__readonly__>
<mod:TABLE_interface>
<mod:ROW_interface>
<mod:interface>mgmt0</mod:interface>
<mod:state>up</mod:state>
<mod:ip_addr>192.0.2.20</mod:ip_addr>
<mod:speed>1000</mod:speed>
<mod:mtu>1500</mod:mtu>
</mod:ROW_interface>
<mod:ROW_interface>
<mod:interface>Ethernet2/1</mod:interface>
<mod:vlan>--</mod:vlan>
<mod:type>eth</mod:type>
<mod:portmode>routed</mod:portmode>
<mod:state>down</mod:state>
<mod:state_rsn_desc>Administratively down</mod:state_rsn_desc>
<mod:speed>auto</mod:speed>
<mod:ratemode>D</mod:ratemode>
</mod:ROW_interface>
</mod:TABLE_interface>
</mod:__readonly__>
</mod:__XML__OPT_Cmd_show_interface_brief___readonly__>
</mod:interface>
</mod:show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

機能不全の構成

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 192.0.2.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

コマンドの実行により、インターフェイスの IP アドレスは設定されますが、管理状態は変更されません（**no shut** コマンドは実行されません）。管理状態が変更されない理由は、**no port-channel 2000** コマンドがエラーになるためです。

この <rpc-reply> は、ある **show** コマンドが、その **show** コマンドの XML 出力を含む <cmd> タグによって送信された結果です。

構成コマンドと表示コマンドを同じに組み合わせることはできません<exec-command> インスタンス。次は、同じインスタンスで **config** コマンドと **show** コマンドが組み合わされた例を示しています。

構成コマンドと表示コマンドの組み合わせ

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>conf t ; interface ethernet 2/1 ; ip address 1.1.1.4/24 ; show xml
server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: cannot mix config and show in exec-command. Config cmds
before the show were executed.
Cmd:show xml server status</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

<exec-command> を通して送信された表示 CLI コマンド

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element><cmd></nf:bad-element>
```

```
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

NETCONF 応答

クライアントによって送信されるすべての XML 要求に対して、XML サーバーは RPC 応答タグ <rpc-reply> で囲まれた XML 応答を送信します。

RPC 応答タグ

次の例は、RPC 応答タグ <rpc-reply> を示しています。

RPC 応答タグ<rpc-reply>

```
<nc:rpc-reply message-id=”315” xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
<ok/>
</nc:rpc-reply>]]>]]>
```

RPC 応答エレメント

<ok>、<data>、そして <rpc-error> の要素は RPC 応答に表示される可能性があります。次の表は、<rpc-reply> タグに現れる可能性のある RPC 応答要素について説明しています。

***Table 30: RPC* 応答エレメント**

要素	説明
<ok>	RPC 要求は正常に完了しました。この要素は、応答でデータが返されない場合に使用されます。
<data>	RPC 要求は正常に完了しました。RPC 要求に関連するデータは、<data> 要素を伴っています。
<rpc-error>	RPC 要求が失敗しました。エラー情報は、<rpc-error> 要素。

データ タグにカプセル化されたタグの解釈

<data> タグ内にカプセル化されたデバイス タグには、要求とそれに続く応答が含まれます。次の例に表示されているように、クライアント アプリケーションは、<readonly> タグ前のタグを全て安全に無視できます：

RPC 応答データ

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nf:data>
<show>
<interface>
<__XML__OPT_Cmd_show_interface_brief___readonly__>
<__readonly__>
<TABLE_interface>
<ROW_interface>
<interface>mgmt0</interface>
<state>up</state>
<ip_addr>xx.xx.xx.xx</ip_addr>
<speed>1000</speed>
<mtu>1500</mtu>
</ROW_interface>
<ROW_interface>
<interface>Ethernet2/1</interface>
<vlan>--</vlan>
<type>eth</type>
<portmode>routed</portmode>
<state>down</state>
<state_rsn_desc>Administratively down</state_rsn_desc>
<speed>auto</speed>
<ratemode>D</ratemode>
</ROW_interface>
</TABLE_interface>
</__readonly__>
</__XML__OPT_Cmd_show_interface_brief___readonly__>
</interface>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

Note <__XML__OPT.*> と <__XML__BLK.*> はレスポンスに表示され、リクエストで使用されることもあります。これらのタグは NETCONF エージェントによって使用され、<__readonly__> タグの後の応答に存在します。これらは要求内で必須であり、スキーマファイルに従って追加した後、CLI コマンドを表す XML タグを続ける必要があります。

サンプル XML インスタンスに関する情報

XML インスタンスの例

このセクションでは、次の XML インスタンスの例を示します：

- NETCONF クロース セッション インスタンス, on page 482
- NETCONF 強制終了セッション インスタンス, on page 482

NETCONF クロース セッション インスタンス

次の例は、close-session 要求とそれに続く close-session 応答を表示しています。

クローズ セッション要求

```
<?xml version="1.0"?>
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:close-session/>
</nc:rpc>]]>]]>
```

クローズ セッション応答

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF 強制終了セッション インスタンス

次の例は、強制終了セッション要求と、 強制終了セッション応答を示しています。

強制終了セッション要求

```
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:kill-session>
<nc:session-id>25241</nc:session-id>
</nc:kill-session>
</nc:rpc>]]>]]>
```

強制終了セッション応答

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF での構成インスタンスのコピー

Note <startup/>は、ソースまたはターゲットのデータストアとしてサポートされていません。**copy running-config startup-config** コマンドを入力するなどして **startup-config** でコピー操作を実行するには、<exec-command>メソッドにフォールバックする必要があります。

次に、構成コピーの要求と、それに続く構成コピーの応答の例を示します。

構成コピーの要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
<target>
<running/>
</target>
<source>
<url>https://user@example.com:passphrase/cfg/new.txt</url>
</source>
</copy-config>
</rpc>
```

構成コピーの応答

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF Edit Config インスタンス

Note 候補データストアを使用する XML edit-config は、バージョン 1.0 の XML 要求ではサポートされていません。ツールで xml を使用して生成できる新しいバージョンでのみサポートされます。

次の例は、NETCONF edit config の使用方法を示しています。

Edit Config 要求

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nc:edit-config>
<nc:target>
<nc:running/>
</nc:target>
<nc:config>
<configure>
<__XML__MODE__exec_configure>
<interface>
<ethernet>
<interface>2/30</interface>
<__XML__MODE_if-ethernet>
<__XML__MODE_if-eth-base>
<description>
<desc_line>Marketing Network</desc_line>
</description>
</__XML__MODE_if-eth-base>
</__XML__MODE_if-ethernet>
</ethernet>
</interface>
</__XML__MODE__exec_configure>
</configure>
</nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>
```

Edit Config 応答

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager" message-id="16">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

edit-config の operation 属性は、指定された操作が実行される構成のポイントを指定します。操作属性が指定されていない場合、構成は既存の構成データ ストアにマージされます。操作属性には、次の値を指定できます。

- create
- merge
- delete

Edit Config：削除操作要求

次の例は、実行構成からインターフェイス Ethernet 0/0 の構成を削除する方法を示しています。

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
```

```
<target>
<running/>
</target>
<default-operation>none</default-operation>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<top xmlns="http://example.com/schema/1.2/config">
<interface xc:operation="delete">
<name>Ethernet0/0</name>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>
```

Edit Config 応答：削除操作

次の例は、実行構成のインターフェイス Ethernet 0/0 の構成を編集する方法を示しています。

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

NETCONF の Get Config インスタンス

次の例は、NETCONF の get config の使用方法を示しています。

サブツリー全体を取得するための Get Config 要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
<filter type="subtree">
<top xmlns="http://example.com/schema/1.2/config">
<users/>
</top>
</filter>
</get-config>
</rpc>]]>]]>
```

クエリの結果を含む Get Config 応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
<type>superuser</type>
<full-name>Charlie Root</full-name>
<company-info>
<dept>1</dept>
```

```
<id>1</id>
</company-info>
</user>
<!-- additional <user> elements appear here... -->
</users>
</top>
</data>
</rpc-reply>]]>]]>
```

NETCONF ロック インスタンス

次の例は、ロック要求、成功の応答、および失敗した試行への応答を示しています。

ロック要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<lock>
<target>
<running/>
</target>
</lock>
</rpc>]]>]]>
```

ロックの取得に成功した場合の応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/> <!-- lock succeeded -->
</rpc-reply>]]>]]>
```

ロックの取得に失敗した場合の応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error> <!-- lock failed -->
<error-type>protocol</error-type>
<error-tag>lock-denied</error-tag>
<error-severity>error</error-severity>
<error-message>
Lock failed, lock is already held
</error-message>
<error-info>
<session-id>454</session-id>
<!-- lock is held by NETCONF session 454 -->
</error-info>
</rpc-error>
</rpc-reply>]]>]]>
```

NETCONF ロック解除インスタンス

次の例は、NETCONF ロック解除の使用方法を示しています。

ロック解除要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<unlock>
<target>
<running/>
</target>
</unlock>
</rpc>
```

ロック解除要求への応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF コミット インスタンス：候補構成機能

次の例は、コミット操作とコミット応答を示しています。

操作をコミット

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit/>
</rpc>
```

返信をコミット

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF 確認済みコミット インスタンス

次の例は、確認済みのコミット操作と確認済みのコミット返答を示しています。

確認されたコミット リクエスト

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit>
<confirmed/>
<confirm-timeout>120</confirm-timeout>
</commit>
</rpc>]]>]]>
```

確認されたコミット応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

NETCONF Rollback-On-Error インスタンス

次の例は、エラー時のロールバックと、この要求への応答を構成する方法を示しています。

Rollback-On-Error の機能

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<error-option>rollback-on-error</error-option>
<config>
<top xmlns="http://example.com/schema/1.2/config">
<interface>
<name>Ethernet0/0</name>
<mtu>100000</mtu>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>
```

Rollback-On-Error への応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

NETCONF 検証機能インスタンス

次の例は、NETCONF 検証機能の使用方法を示しています。文字列 urn:ietf:params:netconf:capability:rollback-on-error:1.0 は、NETCONF 検証機能を指定します。

検証要求

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
<candidate/>
</source>
</validate>
</rpc>]]>]]>
```

検証要求への応答

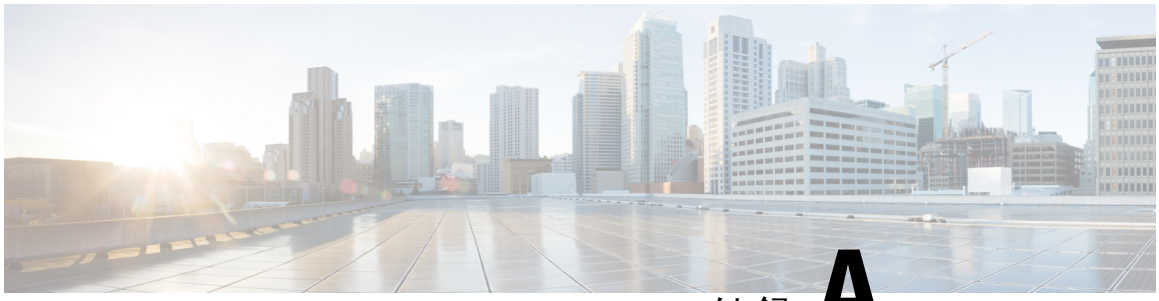
```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

その他の参考資料

ここでは、XML 管理インターフェイスの実装に関する追加情報について説明します。

RFC

RFC	タイトル
RFC 4741	NETCONF Configuration Protocol
RFC 4742	セキュアシェル（SSH）経由の NETCONF 構成プロトコルの使用



付録 A

ストリーミング テレメトリの送信元

ストリーミング テレメトリについて

Cisco Nexus スイッチのストリーミングテレメトリ機能は、ネットワークからデータを継続的にストリーミングしてクライアントに通知し、モニタリング データへのほぼリアルタイムのアクセスを提供します。

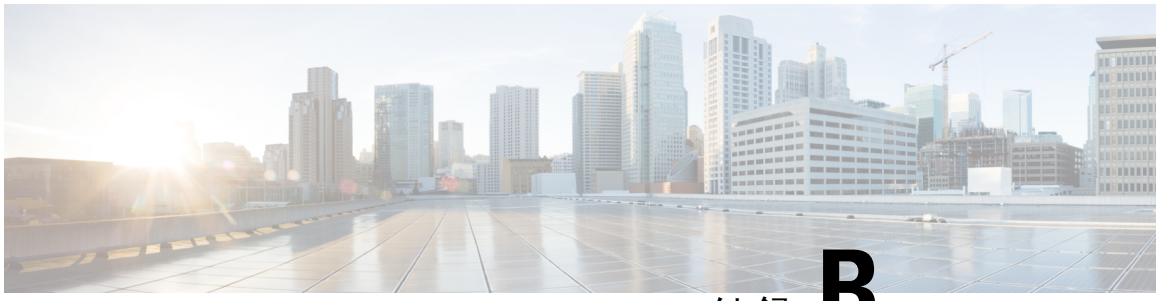
Guidelines and Limitations

Following are the guideline and limitations for the streaming telemetry:

- For information about supported platforms, see the Nexus Switch Platform Matrix.
- Cisco Nexus switches with less than 8 GB of memory do not support telemetry.

テレメトリで利用可能なデータ

コンポーネント グループごとに、NX-API DME モデル リファレンスの付録にある識別名（DN）は、一覧表示されたプロパティをテレメトリのデータとして提供できます。



付録 B

Websocket サブスクリプション

WebSocket サブスクリプション

Cisco NX-OS は、関係するサブスクライバにスイッチが通知をプッシュできるようにするインターフェイス機能を提供します。NX-API WebSocket インターフェイスを介して、プログラムとエンドユーザーはスイッチのさまざまな状態変化に関する通知を受信できるため、定期的なポーリングの必要がなくなります。

Cisco NX-API REST インターフェイスを使用して API クエリを実行する場合、特定のクエリの結果における将来の変更に対するサブスクリプションを作成するオプションがあります。ユーザーまたはシステムにより開始されたアクションによって、管理オブジェクト（MO）が作成、変更、または削除されると、イベントが生成されます。受信したイベントがサブスクライブ済みのクエリの結果を変更すると、スイッチはサブスクリプションを作成した API クライアントへのプッシュ通知を生成します。

•

WebSocket を開く

API サブスクリプション機能は、WebSocket プロトコル（RFC 6455）を使用して、API クライアントとの双方向接続を実装します。このようにして、API は非送信請求の通知メッセージをクライアント自体に送信できます。通知チャネルを確立するには、まずそれぞれの API との WebSocket 接続をオープンする必要があります。各スイッチ内で複数のクエリ サブスクリプションをサポートするために必要な WebSocket 接続は 1 つだけです。WebSocket 接続は（トークン検証によって）API セッション接続に依存しており、API セッションが終了すると閉じます。

WebSocket 接続を開く方法はいくつかあります。Python クライアントは次のように記述できます。

```
from websocket import create_connection

connection_string = "ws:// 10.1.2.3/socket{0}".format(token)

ws = create_connection(connection_string, sslopt={"check_hostname": False})
```

URI では、トークンは現在の API セッション トークン（クッキー）です。次に、トークン付きの URI の例を示します。

Websocket サブスクリプション

```
ws://10.1.2.3/socketGkZl5NLRZJl5+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB
Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3bcP2Mxy3VBmgoJjwZ76ZOuf9V9AD6X
l83lyoR4bLBzqbSSU1R2NIgUotCGWjZt5JX6CJF0=
```

サブスクリプションの作成

クエリにサブスクリプションを作成するには、オプション「?subscription=yes」でクエリを実行します。この例では、JSON 形式の sys/intf/phys-[eth1/1] のクエリへのサブスクリプションを作成します。

```
GET http://10.1.1.1/api/mo/sys/intf/phys-[eth1/1].json?subscription=yes
```

クエリ応答には、サブスクリプションの識別子である subscriptionId が含まれています。サブスクリプションを更新し、特定のサブスクリプションからの今後の通知を識別するために使用できます。

```
{"totalCount":"0","subscriptionId":"18374686685813276673","imdata":[]}
```

受信通知

サブスクリプションからのイベント通知では、サブスクリプション ID と MO の説明を含むデータ構造が提供されます。この JSON の例では、sys/intf/phys-[eth1/1] の説明が「test」に変更されています。

{"subscriptionId":["18374686685813276673"],"imdata":[{"l1PhysIf": {"attributes": {"childAction": "","descr": "test","dn": "sys/intf/phys-[eth1/1]","modTs": "2019-10-18T19:42:29.446+00:00","rn": "","status": "modified"}}}]}

特定のクエリに対して複数のアクティブなサブスクリプションが存在する可能性があるため、通知には上記の例と同様なサブスクリプション ID を複数の含めることができます。通知は、JSON または XML フォーマットでサポートされています。

サブスクリプションの更新

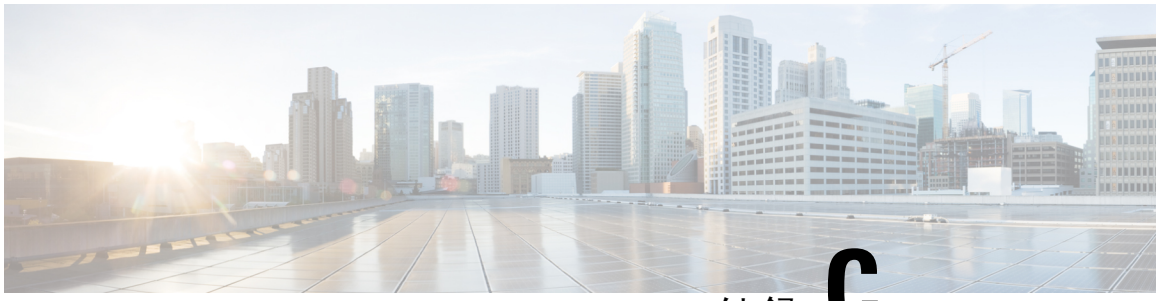
イベント通知を受信し続けるには、API セッション中に各サブスクリプションを定期的に更新する必要があります。サブスクリプションを更新するには、例のように、`subscriptionId` に等しいパラメータ `id` を使用して、HTTP GET メッセージを API メソッド `subscriptionRefresh` に送信します。

```
GET http://10.1.1.1/api/subscriptonRefresh.json?id=18374686685813276673
```

サブスクリプションが期限切れになっていなければ、API はリフレッシュ メッセージに空の応答を返します。

（注） WebSocket サブスクリプションのタイムアウト期間は、デフォルトで 90 秒です。通知が失われないようにするには、サブスクリプションのリフレッシュ メッセージを少なくとも 90 秒ごとに送信する必要があります。

要約すると、WebSocket は、NX-OS REST API 内でイベント サブスクリプションのパブリッシャーとサブスクライバ間の通信を可能にする強力なツールとなり得ます。



付録 C

プログラマビリティに関する RFC

プログラマビリティに関する RFC

次の表に、RFC 準拠標準を示します。各 RFC の詳細については、www.ietf.org を参照してください。

表 *31 : RFC* 準拠標準

RFC	タイトル
RFC 5277	NETCONF イベント通知
RFC 6241	ネットワーク設定プロトコル（NETCONF）
RFC 6243	NETCONF の With-defaults 機能（**report-all** でのみサポート）

プログラマビリティに関する RFC

翻訳について
このドキュメントは、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。