Cisco Nexus 3600 スイッチ NX-OS プログラマビリティ ガイド、リリース 10.6(x)

最終更新：2026 年 1 月 2 日

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

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

目 次

はじめに

この前書きは、次の項で構成されています。

対象読者

このマニュアルは、Cisco Nexus スイッチの設置、設定、および維持に携わるネットワーク管理者を対象としています。

表記法

コマンドの説明には、次のような表記法が使用されます。

表記法	説明
bold	太字の文字は、表示どおりにユーザが入力するコマンドおよびキーワードです。
italic	イタリック体の文字は、ユーザが値を入力する引数です。
[x]	省略可能な要素（キーワードまたは引数）は、角かっこで囲んで示しています。
[x | y]	いずれか1つを選択できる省略可能なキーワードや引数は、角カッコで囲み、縦棒で区切って示しています。
{x | y}	必ずいずれか1つを選択しなければならない必須キーワードや引数は、波かっこで囲み、縦棒で区切って示しています。

表記法	説明
[x {y | z}]	角かっこまたは波かっこが入れ子になっている箇所は、任意または必須の要素内の任意または必須の選択肢であることを表します。角かっこ内の波かっこと縦棒は、省略可能な要素内で選択すべき必須の要素を示しています。
`variable`	ユーザが値を入力する変数であることを表します。イタリック体が使用できない場合に使用されます。
string	引用符を付けない一組の文字。string の前後には引用符を使用しません。引用符を使用すると、その引用符も含めて string とみなされます。

例では、次の表記法を使用しています。

表記法	説明
`screen フォント`	スイッチが表示する端末セッションおよび情報は、スクリーンフォントで示しています。
太字の `screen` フォント	ユーザが入力しなければならない情報は、太字のスクリーンフォントで示しています。
イタリック体の *screen* フォント	ユーザが値を指定する引数は、イタリック体の screen フォントで示しています。
<>	パスワードのように出力されない文字は、山カッコ（<>）で囲んで示しています。
[ ]	システム プロンプトに対するデフォルトの応答は、角カッコで囲んで示しています。
!、#	コードの先頭に感嘆符（!）またはポンド記号（#）がある場合には、コメント行であることを示します。

Cisco Nexus 3600 プラットフォーム スイッチの関連資料

Cisco Nexus 3600 プラットフォーム スイッチ全体のマニュアル セットは、次の URL にあります。

http://www.cisco.com/c/en/us/support/switches/nexus-3000-series-switches/tsd-products-support-series-home.html

マニュアルに関するフィードバック

このマニュアルに関する技術的なフィードバック、または誤りや記載もれなどお気づきの点がございましたら、HTML ドキュメント内のフィードバック フォームよりご連絡ください。ご協力をよろしくお願いいたします。

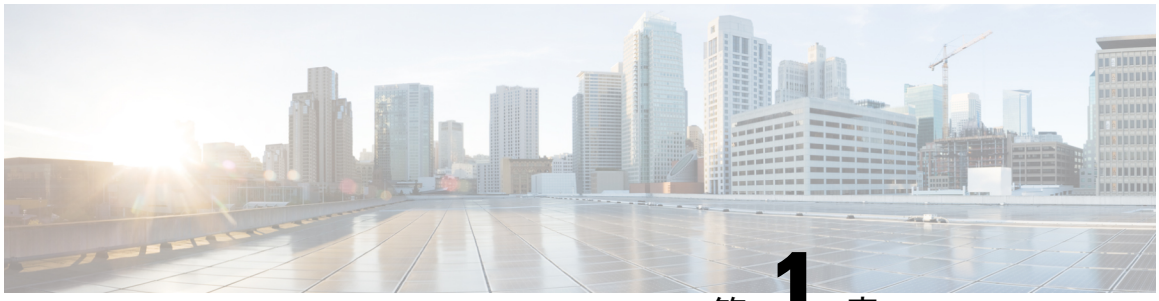
通信、サービス、およびその他の情報

- シスコからタイムリーな関連情報を受け取るには、Cisco Profile Manager でサインアップしてください。
- 重要な技術によって求めるビジネス成果を得るには、Cisco Services [英語] にアクセスしてください。
- サービス リクエストを送信するには、Cisco Support にアクセスしてください。
- 安全で検証済みのエンタープライズクラスのアプリケーション、製品、ソリューション、およびサービスを探して参照するには、Cisco DevNet [英語] にアクセスしてください。
- 一般的なネットワーキング、トレーニング、認定関連の出版物を入手するには、Cisco Press にアクセスしてください。
- 特定の製品または製品ファミリの保証情報を探すには、Cisco Warranty Finder にアクセスしてください。

Cisco バグ検索ツール

Cisco Bug Search Tool（BST）は、シスコ製品とソフトウェアの障害と脆弱性の包括的なリストを管理する Cisco バグ追跡システムへのゲートウェイとして機能する、Web ベースのツールです。BST は、製品とソフトウェアに関する詳細な障害情報を提供します。

通信、サービス、およびその他の情報



第 1 章

新機能および変更された機能に関する情報

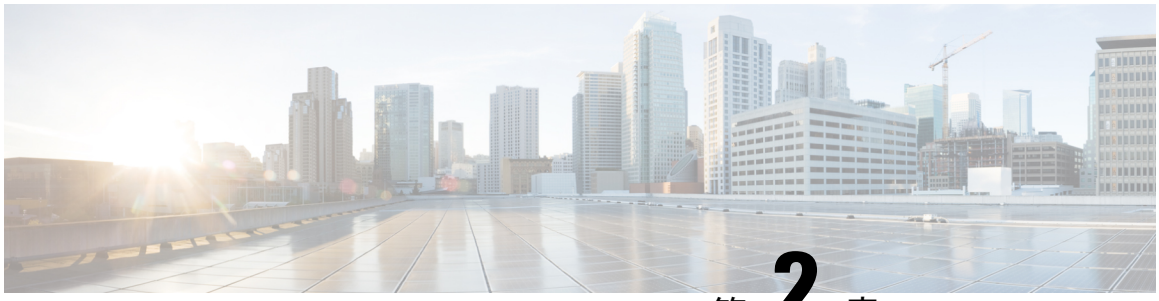
新機能および変更された機能に関する情報

次の表は、『*Cisco Nexus 3600* シリーズ *NX-OS* プログラマビリティ ガイド リリース *10.6(x)*』に記載されている新機能および変更機能を要約したものです。それぞれの説明が記載されている箇所も併記されています。

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
NA	このリリースで追加された新機能はありません。	10.6(1)F	N/A

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 3600 プラットフォーム 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。

- **高度なプログラマティック**

  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。

ライセンス要件

Cisco NX-OS ライセンス方式の推奨の詳細と、ライセンスの取得および適用の方法については、『*Cisco NX-OS* ライセンス ガイド』および『*Cisco NX-OS* ライセンス オプション ガイド』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) 以降では、Nexus スイッチ プラットフォーム サポート マトリクスに基づいて、選択した機能をさまざまな Cisco Nexus 9000 および 3000 スイッチで使用するために、どの Cisco NX-OS リリースが必要かを確認してください。

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

デバイス上の拡張 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、Power On Auto Provisioning（POAP）のサポートが含まれています。

デバイス上の拡張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次の機能が含まれています：

- 電源オン自動プロビジョニング（POAP）サポート

- Chef と Puppetの統合
- OpenStack の統合
- OpenDaylight の統合と OpenFlow のサポート

電源オン自動プロビジョニング サポート

PowerOn Auto Provisioning（POAP）は、ネットワークに初めて導入されたスイッチに対して、ソフトウェア イメージのインストールとアップグレードとコンフィギュレーション ファイルのインストールのプロセスを自動化します。ネットワーク容量を調整するために必要な手動作業が削減されます。

POAP機能を備えたスイッチが起動し、スタートアップ構成が検出されない場合、デバイスはPOAP モードを開始します。DHCP サーバを見つけ、インターフェイス IP アドレス、ゲートウェイ、DNS サーバIPアドレスを使用して自らをブートストラップします。デバイスはTFTPサーバの IP アドレスまたは HTTP サーバの URL を取得し、構成スクリプトをダウンロードします。このスクリプトは、デバイスが適切なソフトウェア イメージと構成ファイルをダウンロードしてインストールできるようにします。

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るスイッチへのアクセスが可能になります。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。

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。

Perl モジュール

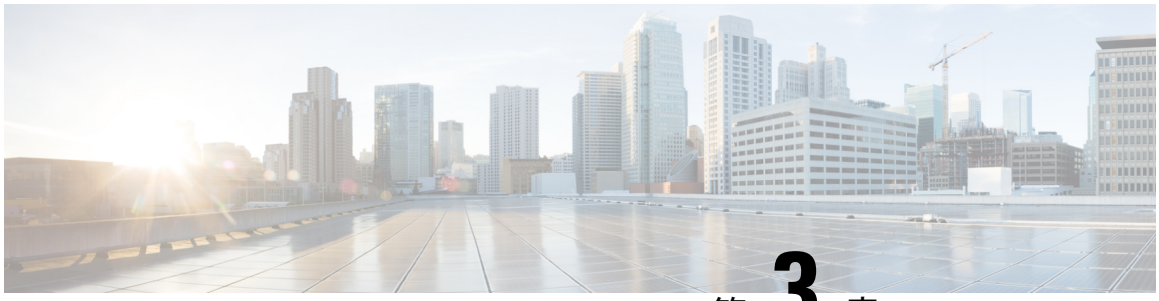
追加のアプリケーションをサポートするために、次の Perl モジュールが追加されました。

- bytes.pm
- feature.pm
- hostname.pl
- lib.pm
- overload.pm
- Carp.pm
- クラス/構造体.pm
- Data/Dumper.pm
- DynaLoader.pm
- エクスポータ/Heavy.pm
- FileHandle.pm
- ファイル/ベース名.pm
- ファイル/Glob.pm
- ファイル/仕様.pm
- ファイル/仕様/Unix.pm
- ファイル/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
- 時刻/ハイ レゾ.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、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。

- `/isan` フォルダにあるバイナリは、 **run bash** コマンドから入力されたシェルとは異なる設定の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。

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** *command* 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、Bash シェルを直接実行できるようになります。

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：

- 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。
- 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。
- 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。
- 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。

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 ユーザはシェルタイプの Bash を使用しており、シェルタイプの Bash を使用してスイッチにログインしています。
- 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。

以下の例は、デフォルトの shelltype 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) --
```

以下の例は、デフォルトの shelltype 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 3600 software ("Nexus 3600 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 3600 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 3600 is strictly limited to use for evaluation, demonstration      *
*  and NX-OS education. Any use or disclosure, in whole or in part of     *
*  the Nexus 3600 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) --
```

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
```

Bash からの Python の実行

次の例は、Python をロードし、Python オブジェクトを使用してスイッチを構成する方法を示しています。

```
switch# run bash
bash-4.2$ python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cisco import *
>>> from cisco.vrf import *
>>> from cisco.interface import *
>>> vrfobj=VRF('myvrf')
>>> vrfobj.get_name()
'myvrf'
```

```
>>> vrfobj.add_interface('Ethernet1/3')
True
>>> intf=Interface('Ethernet1/3')
>>> print intf.config()

!Command: show running-config interface Ethernet1/3
!Time: Mon Nov 4 13:17:56 2013

version 6.1(2)I2(1)

interface Ethernet1/3
  vrf member myvrf

>>>
```

RPM の管理

Bash からの RPM のインストール

手順

	コマンドまたはアクション	目的
ステップ 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。
ステップ 2	**dnf list available**	使用可能な RPM のリストを表示します。
ステップ 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の代わりにコマンドを使用します。**rpmdnf** それ以外の場合、RPM は最初にインストールされたのではなく、リポジトリ名またはファイル名を使用してインストールされるか、または表示されます。**dnf bashinstall cli**

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

手順

	コマンドまたはアクション	目的
ステップ 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
```

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

手順

	コマンドまたはアクション	目的
ステップ 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
```

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/yum/protected.d/protected_pkgs.conf /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

手順

	コマンドまたはアクション	目的
ステップ 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
```

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、次の一般的なフォーマットにする必要があります（このフォーマットの詳細については、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

手順

ステップ 1 上記で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。

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

ステップ 3 **chkconfig --add** *application_name*を入力します

ステップ 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。

ステップ 5 -- 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。**chkconfiglist**

ステップ 6 アプリケーションが /etc/rc3.d にリストされていることを確認します。「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、`../init.d/`*application_name* に 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#
```

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

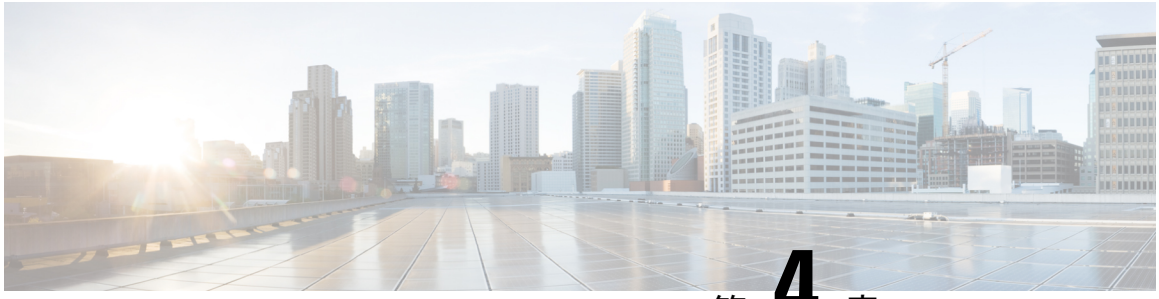
次の例は、ネイティブ Bash シェルのアプリケーションを示しています：

```
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、ゲスト シェルは、有効にすると約 5 MB の RAM と 200 MB のブートフラッシュを占有します。Cisco NX-OS リリース 7.0（3）I2（1）以降、Guest Shell は約 35 MB の RAM 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リソースを再利用します。

（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。

（注） Cisco NX-OS 7.0(3)F3(1)NX-OS 7.0(3)I7(1)以降、Guest Shell は Cisco Nexus 95083500 スイッチでサポートされます。

注意事項と制約事項

すべてのリリースに共通の注意事項

重要 Guest Shell 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 Guest Shell 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 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 Guest Shell 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、Guest Shell にアクセスして Bash プロンプトを取得したり、ゲストシェル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、Guest Shell 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。
  - Guest Shell の再起動後にゲストシェル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 Guest Shell に接続できないようにします。ネットワーク管理者ユーザーは、Guest Shell で sshd の別のインスタンスを開始して、Guest Shell に直接リモート アクセスできるようにすることができますが、Guest Shell にログインするすべてのユーザーには、ネットワーク管理者権限も与えられます。

（注） ゲストシェル 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、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。

2.2（0.2）より前の Guest Shell 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、Guest Shell 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、Guest Shell は自動的にアップグレードされません。
- Guest Shell のリリースでは、配布または配布のバージョンが変更されると、メジャー番号が増分されます。
- Guest Shell のリリースでは、CVE が解決されるとマイナー番号が増分されます。Guest Shell 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、既存のゲスト シェル 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、この新しいゲスト シェル rootfs のコンテキスト内で再度実行する必要があります。

Guest Shell 3.0

Jacksonville リリース 10.1(1) 以降、Guest Shell 2.x にパッケージ化されている CentOS 7 のサポートが終了しているため、Guest Shell 3.0 が CentOS 8 ソフトウェアとともに導入されました。CentOS 8 には Python 3.6 も付属しており、Guestshell 2.x での Python 2.7 サポートに置き換わります。Guest Shell 2.x と Guest Shell 3.0 間の機能は同じままです。ただし、内部実装の相違点として、Guest Shell 3.0 のPython 3.6 ライブラリは、Guest Shell 2.x のPython 2.7 ライブラリを置き換えることになります。これは、 `/usr/lib/python3.6` および `/usr/lib64/python3.6` ライブラリが使用されることを意味します。

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

Guest Shell 1.0 からゲスト シェル 2.x へのアップグレード

Guest Shell 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 Guest Shell 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、Guest Shell 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。

Guest Shell 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

Jacksonville リリース 10.1(1) 以降、Guest Shell 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。Guest Shell 3.0 を以前のリリースにダウングレードする前に、Guest Shell 3.0 を破棄して、Guest Shell 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。

Guest Shell 2.x

Cisco NX-OS は、デフォルトで自動的に Guest Shell のインストールおよび有効化を行います。ただし、Guest Shell 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 Guest Shell 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 Guest Shell が有効になりません。**guestshell enable** コマンドを使用して、Guest Shell 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: Guest shell 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、Guest Shell 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 Guest Shell を削除します。

事前設定された SSHD サービス

Guest Shell 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 Guest Shell 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 Guest Shell 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、Guest Shell 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 Guest Shell 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. ゲスト シェルへの SSH：

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

8. 複数の Guest Shell またはスイッチの再起動にまたが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

Guest Shell 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、ゲスト シェル固有の /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 では、デフォルトで network-admin ユーザのみがゲスト シェル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 %	1/620%
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。

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。

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。

ゲスト シェルでの RPM のインストール

/etc/dnf.repos.d/CentOS-Base.repo ファイルは、デフォルトで CentOS ミラー リストを使用するように設定されています。変更が必要な場合は、そのファイルの指示に従ってください。

dnf は、yumrepo_x86_64.repo ファイルを変更するか、repos.d ディレクトリに new.repo ファイルを追加することによって、いつでも 1 つ以上のリポジトリを指すことができます。

ゲスト シェル 2.x 内にインストールするアプリケーションについては、http://mirror.centos.org/centos/7/os/x86_64/Packages/ にある CentOS 7 リポジトリに移動します。

ゲスト シェル 4.x 内にインストールするアプリケーションについては、https://dl.rockylinux.org/vault/rocky/9.2/BaseOS/x86_64/ にある RockyLinux 9 リポジトリに移動します。ミラーリンクのいずれかを選択し、パッケージを表示します。

Dnf は依存関係を解決し、必要なすべてのパッケージをインストールします。

```
[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!
```

（注） パッケージをインストールまたは実行するためにゲスト シェル ルート ファイル システムにより多くのスペースが必要な場合は、**guestshell resize roofs** *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。

（注） リポジトリからの一部のオープン ソース ソフトウェア パッケージは、ホスト システムの完全性を保護するために設定された制限の結果として、ゲスト シェルで期待どおりにインストールまたは実行されない場合があります。

仮想サービス 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 シェルと仮想サービスの使用は、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うちの 2 つにすぎません。これらのオプションは、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。

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。

[デジタル署名されたアプリケーション パッケージ（Digitally Signed Application Packages）]

デフォルトでは、Cisco ネットワーク要素は、アプリケーションのランタイムに有効な Cisco デジタル署名を提供することを要求します。Cisco のデジタル署名により、Cisco が開発したパッケージとアプリケーションの完全性が保証されます。

Cisco Nexus 30009000 シリーズ スイッチは、署名されていない OVA ソフトウェア パッケージを許可する署名レベル ポリシーの構成をサポートしています。署名されていないパッケージと Cisco 署名されたパッケージで仮想サービスを作成できるようにするために、ネットワーク管理者は次を構成できます。

```
virtual-service
    signing level unsigned
```

（注） ゲストシェル ソフトウェア パッケージには Cisco の署名があり、この構成は必要ありません。

[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。

（注） シスコは、Guestshell 4.x（Rocky Linux 9）環境の脆弱性を追跡しており、将来の修正を、Rocky Linux から入手可能になった時点で含めます。

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

Cisco 3000 9000 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

ホストと仮想サービスは、別々の名前空間に分けられます。これは、仮想サービスの実行スペースをホストから分離する基礎を提供します。名前空間の分離は、信頼境界間の偶発的または意図的なデータの上書きによるデータの損失やデータの破損から保護するのに役立ちます。また、機密データ漏洩を防止することにより、機密データの完全性を確保するのにも役立ちます。ある仮想サービスのアプリケーションは、別の仮想サービスのデータにアクセスできません。

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。

仮想サービス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 が従う仮想サービス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_SYS_BOOT	CAP_MKNOD	CAP_SYS_PACCT
CAP_SYS_MODULE	CAP_MAC_OVERRIDE	CAP_SYS_RESOURCE
CAP_SYS_TIME	CAP_SYS_RAWIO	CAP_AUDIT_WRITE
CAP_AUDIT_CONTROL	CAP_SYS_NICE	CAP_NET_ADMIN
CAP_MAC_ADMIN	CAP_SYS_PTRACE	

- 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

仮想サービス内のルートとして、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。

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すべての仮想サービス間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

仮想サービス内のファイルの完全性を維持するために、仮想サービスのファイル システムには NX-OS CLI からアクセスできません。特定の仮想サービスがファイルの変更を許可している場合、これを実行できる代替手段 (つまり **yum install**、**scp**、**ftp**など) を提供する必要があります。

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 シェルは、ホスト システムの[ブートフラッシュ（bootflash）]を [/ ブートフラッシュ（/bootflash）] にマウントします。ネットワーク管理者は、ゲスト シェル内から NX-OS CLI または Linux コマンドを使用してファイルにアクセスできます。

ホストの [ブートフラッシュ：（bootflash:）] と [揮発性：（volatile:）] は、ゲスト シェル内で [/ ブートフラッシュ（/bootflash）] および [/ 揮発性（/volatile）] としてマウントされます。ネットワーク管理者は、ホストから NX-OS exec コマンドを使用するか、ゲスト シェル内から Linux コマンドを使用して、このメディア上のファイルにアクセスできます。

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
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。

コマンド	説明
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。[（7.0（3）I7（1）以降のリリース）]
guestshell disable	シャット ダウンとゲスト シェルの無効化
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） 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。

圧縮されたイメージを使用した Cisco Nexus 3000 での Guest Shell の有効化

Guest Shell ソフトウェアは、1.6 GB のブートフラッシュと 4 GB の RAM を備えた Cisco Nexus 3000 シリーズ スイッチ用に圧縮された Cisco NX-OS イメージでは使用できません。この場合も引き続き Guest Shell を使用できますが、software.cisco.com から Cisco NX-OS リリース用のソフトウェア パッケージをダウンロードしてから、それを Cisco Nexus 3000 シリーズ スイッチにコピーして有効にする必要があります。それ。

コンパクト イメージの詳細については、『*Cisco Nexus 3000 Series NX-OS Software Upgrade and Downgrade Guide, Release 9.2(1)*』を参照してください。

Guest Shell ソフトウェアは、スイッチのブートフラッシュにインストールされます。できるだけ多くの空きブートフラッシュ スペースを作成するには、ダウンロードした`guestshell.ova`ファイルを `volatile:` ストレージ メディアに置きます。Guest Shell が正常にアクティブ化されたら、`guestshell.ova` ファイルを削除できます。ある時点で Guest Shell が破棄され、再インストールする必要がない限り、再度必要になることはありません。

例：

```
switch# copy scp://admin@1.2.3.4/guestshell.ova volatile: vrf management
guestshell.ova 100% 55MB 10.9MB/s 00:05
Copy complete, now saving to disk (please wait)...
Copy complete.

switch# dir volatile: | inc .ova
57251840 Jun 22 11:56:51 2018 guestshell.ova

switch# guestshell enable package volatile:guestshell.ova
2018 Jun 7 19:13:03 n3x-164 %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual service
 'guestshell+'
2018 Jun 7 19:13:56 n3x-164 %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2018 Jun 7 19:13:56 n3x-164 %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual
service 'guestshell+'
2018 Jun 7 19:15:34 n3x-164 %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
 virtual service 'guestshell+'

switch# del volatile:guestshell.ova
Do you want to delete "/guestshell.ova" ? (yes/no/abort) [y] y

switch# guestshell
[admin@guestshell ~]$
```

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`
`chef Installed`
`chef-0.8.1-n9000-spa-k9.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

手順

ステップ 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

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

ステップ 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

ステップ 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

ステップ 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#
```

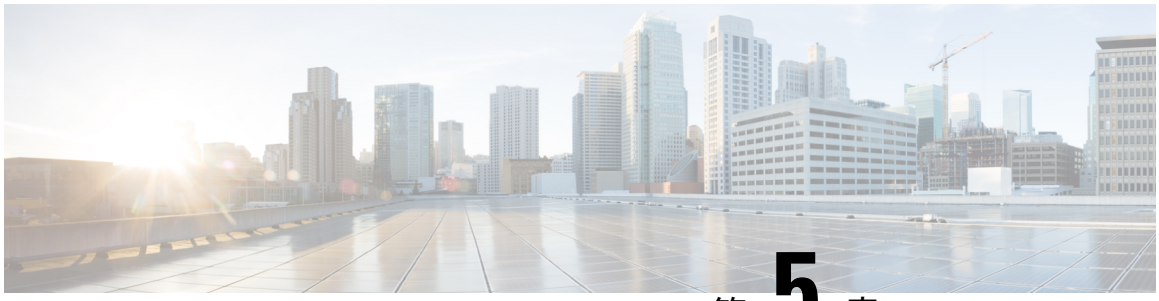
ゲスト シェルから：

```
[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

Python API

[Python API について（About the Python API）]

Python は簡単に習得できる強力なプログラミング言語です。効率的で高水準なデータ構造を持ち、オブジェクト指向プログラミングに対してシンプルで効果的なアプローチを取っています。Python は、簡潔な構文、動的な型指定、インタープリタ型という特長を持っており、ほとんどのプラットフォームのさまざまな分野でスクリプティングと高速アプリケーション開発が可能な理想的な言語です。

Python インタープリタと広範な標準規格ライブラリが Python Web サイトで送信元形式またはバイナリ形式で自由に利用できます：

http://www.python.org/

また、このサイトには、サードパーティが無償で提供している多数の Python モジュール、プログラム、ツールのディストリビューションとそれらへのリンク、さらに追加のドキュメンテーションが掲載されています。

スイッチは、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 Python v2.7.5 をサポートし、ゲスト シェルで使用できます。

Python スクリプト機能は、さまざまなタスクを実行するためにデバイスのコマンドライン インターフェイス（CLI）PowerOn 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。Python は Bash シェルからもアクセスできます。

Python インタプリタは Cisco NX-OS ソフトウェアで利用できます。

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")`	**cli-command** コマンドに XML サポートが存在する場合は、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。

Cisco NX-OS 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

Cisco Nexus 3600 プラットフォーム スイッチ上の組み込みイベント マネージャ（EEM）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 3600
>>>
>>> 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：

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
```

```
[GCC 4.6.3] on linux2
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')
>>> 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# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','r')
Permission denied. Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [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')
'11:28:53.845 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

!Command: show running-config l3vm
!Time: Sun May  8 11:29:40 2011

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue

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
```

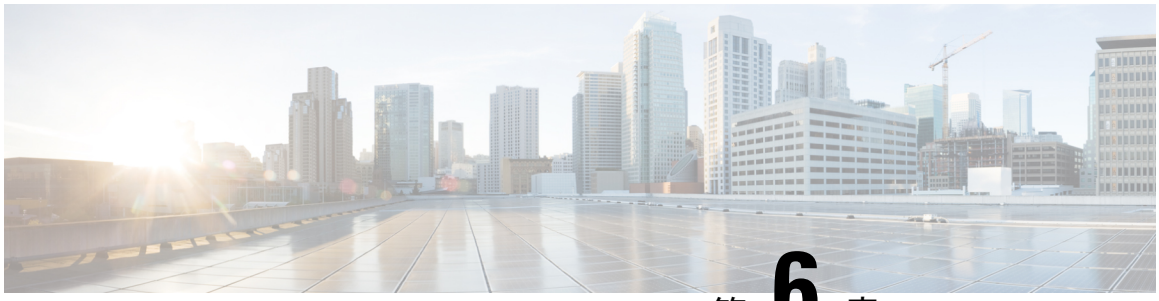
スケジューラでスクリプトを実行する例

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
```

```
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/testplan.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 Mon Mar 14 16:40:03 2011
switch(config-schedule)# end
switch# term mon
2011 Mar 14 16:38:03 switch %VSHD-5-VSHD_SYSLOG_CONFIG_I: Configured from vty by admin
on 10.19.68.246@pts/2
switch# show scheduler schedule
Schedule Name       : testplan
------------------------------
User Name           : admin
Schedule Type       : Run every 0 Days 0 Hrs 4 Mins
Start Time          : Mon Mar 14 16:40:03 2011
```

```
Last Execution Time : Yet to be executed
----------------------------------------------
     Job Name            Last Execution Status
----------------------------------------------
    testplan                              -NA-
==============================================================================
switch#
switch# 2011 Mar 14 16:40:04 switch  %USER-1-SYSTEM_MSG: No user ran
/bootflash/scripts/testplan.py on : switch   - nxpython
2011 Mar 14 16:44:04 switch last message repeated 1 time
switch#
```



第 6 章

tcl によるスクリプティング

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限事項を示します。

- Tcl は、Cisco Nexus スイッチではサポートされません。
- 一部のプロセスおよびコマンドでは、大量の出力が発生する可能性があります。**show** 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

手順

	コマンドまたはアクション	目的

ステップ 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

手順

	コマンドまたはアクション	目的
ステップ 1	**tclsh** 例：	インタラクティブ Tcl シェルを開始します。

	コマンドまたはアクション	目的
	`switch# tclsh` `switch-tcl#`	

ステップ 2	**configure terminal**
	例：
	`switch-tcl# configure terminal`
	`switch(config-tcl)#`
		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。
		（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。

ステップ 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

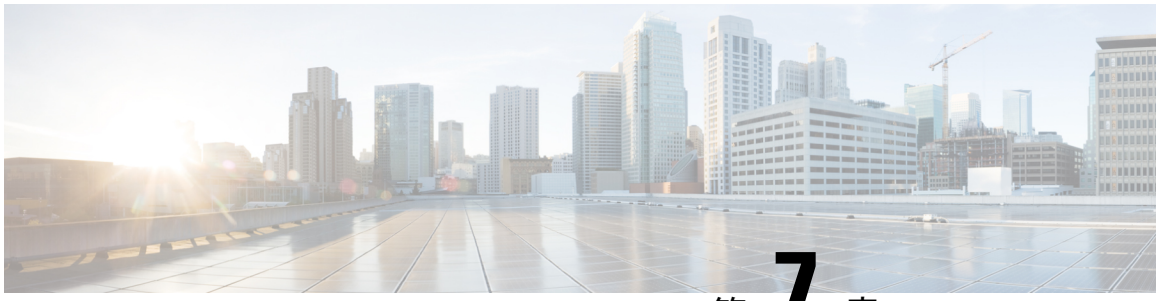
次の例は、対話型 Tcl シェルから Cisco NX-OS モードを変更する方法を示しています：

```
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 7 章

iPXE

この章は、次の内容で構成されています。

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 は追加のプロトコルをサポートします。

標準 PXE よりも iPXE が提供す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、その他によって提供されるスクリプトなどをサポートします。そして、
- DHCPv6 のステートレス アドレス自動構成（SLAAC）およびステートフル IP 自動構成バリアントをサポートします。 iPXE は、DHCPv6 オプションのブート URI とパラメータをサポートします。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、次のようなセキュリティ上の理由から、iPXE の既存の機能の一部が無効になっています。

- bzImage+initramfs/initrd、または ISO 、その他の標準 Linux イメージ形式のブート サポート

- FCoE、iSCSI SAN、ワイヤレス 、その他の未使用のネットワーク ブート オプション
- 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される可能性があるため、サポートされていない NBP（syslinux/pxelinux など）のロード。

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。
- IPv4/IPv6 サポート：導入の柔軟性を向上

注意事項と制約事項

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は、**Ctrl+B** を入力して PXE ブートを終了できる 3 秒間のウィンドウがあります。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/損失をより適切に処理します。そのため、Cisco NX-OS イメージのサイズが 250 Mb 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、Cisco の署名付き NBI イメージのみを許可/ブートします。その他の標準イメージ フォーマットのサポートは、セキュリティ上の理由から無効になっています。

iPXE のメモ

DHCP サーバーのインストール

デフォルトでは、 DHCP はサーバーにインストールされません。 **service dhcpd status** コマンドを使用して、 DHCP サーバーのインストールを確認できます。

```
[switch etc]# service dhcpd status
dhcpd: unrecognized service  /* indicates that dhcp server is not installed */
```

DHCP は **yum install dhcp** コマンドでインストールできます。

（注） DHCP サーバーをインストールするには、ルート ログイン情報が必要です。

```
[switch etc]# yum install dhcp
Repository base is listed more than once in the configuration
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package dhcp.x86_64 12:3.0.5-23.el5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved
================================================================================

 Package         Arch              Version                     Repository
 Size
================================================================================
Installing:
 dhcp            x86_64            12:3.0.5-23.el5             workstation
883 k

Transaction Summary
================================================================================
Install       1 Package(s)
Upgrade       0 Package(s)

Total download size: 883 k
Is this ok [y/N]: y
Downloading Packages:
dhcp-3.0.5-23.el5.x86_64.rpm                                | 883 kB     00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : dhcp
  1/1
Installed:
  dhcp.x86_64 12:3.0.5-23.el5

Complete!
[switch etc]#
```

DHCP サーバーに構成を追加する

DHCP サーバーをインストールすると、構成ファイルは **/etc/dhcpd.conf** に配置されます。

次に、 **dhcpd.conf** ファイルの例を示します。

```
--------------------------------------------------------------------------------

 # Set the amount of time in seconds that  a client may keep the IP address
```

```
default-lease-time 300;
max-lease-time 7200;
one-lease-per-client true;

#Indicate the preferred interface that your DHCP server listens only to that interface
and to no other . Preferred interface should be added to the DHCPDARGS variable
DHCPDARGS=eth0

#A subnet section is generated for each of the interfaces present on your Linux system
subnet 10.0.00.0 netmask 255.255.255.0 {

# The range of IP addresses the server will issue to DHCP enabled PC clients booting up
 on the network
  range 10.0.00.2 10.0.00.100;

#Address of the preferred inteface
  next-server 10.0.00.4;

#The default gateway to be used
  option routers 10.0.00.254;

#The file path where the  ipxe boot looks for the image
  filename = "http://10.0.00.4/pxe/dummy";
#  (http://10.0.00.4 points to the httpd service path mentioned in DocumentRoot variable
#   at /etc/httpd/conf/httpd.conf ) .
# By default it points to "DocumentRoot "/var/www/html"  (Refer the HTTP service section)


  option domain-name "cisco.com";
  option domain-name-servers 100.00.000.183;

host  Nexus {
        hardware ethernet e4:c7:22:bd:c4:f9;
        fixed-address 10.0.00.42;
        filename = "http://10.0.00.4/ipxe/nxos-image.bin";

host Nexus {
          hardware ethernet 64:f6:9d:07:52:f7;
          fixed-address 10.0.00.8;
          filename = "tftp://100.00.000.48/nxos-image.bin";
```

DHCP サービスの管理

（注） DHCP サービスのインストール後、サービスを開始する必要があります。

- DHCP サービスの確認

```
[switch etc]# service dhcpd status
dhcpd is stopped
```

- DHCP サービスの開始

```
[switch etc]#  service dhcpd start
Starting dhcpd:                                            [ok]
```

- DHCP サービスの停止

```
[switch etc]# service dhcpd stop
Stopping dhcpd:                                            [ok]
```

- DHCP サービスの再起動

（注） DHCP 構成ファイル **/etc/dhcpd.conf** が更新されたら、サービスを再起動する必要があります。

```
[switch etc]# service dhcpd restart
Starting dhcpd:                                            [ok]
```

HTTP サーバーの管理

- HTTP サーバーのインストール

```
[switch conf]# yum install httpd
```

- HTTP サービスの開始

```
[switch conf]# service httpd start
Starting httpd: httpd: Could not reliably determine the server's fully qualified
domain name,
using 100.00.000.127 for ServerName
                                                           [  OK  ]
```

- HTTP サービスの停止

```
[switch conf]# service httpd stop
Stopping httpd:                                            [  OK  ]
```

- HTTP サービスの再起動

```
[switch conf]# service httpd restart
Stopping httpd:                                            [FAILED]
Starting httpd: httpd: Could not reliably determine the server's fully qualified
domain name,
using 100.00.000.127 for ServerName
                                                           [  OK  ]
```

- HTTP ステータスの確認

```
[switch conf]#  service httpd status
httpd (pid  23032) is running...
```

（注） HTTP の構成ファイルは、**/etc/httpd/conf/httpd.conf** にあります。

（注）

- DocumentRoot：ドキュメントを配信するディレクトリ。デフォルトでは、すべての要求がこのディレクトリから取得されますが、シンボリックリンクとエイリアスを使用して他の場所を指すこともできます。
- DocumentRoot **/var/www/html**

  DocumentRoot 変数には、http://<ip_add>ファイル名変数を使用して **dhcpd.conf** ファイルのフィールドに追加します。

  次に、例を示します。

```
host  Nexus {
        hardware ethernet e4:c7:22:bd:c4:f9;
        fixed-address 10.0.00.42;
       filename =  "http://10.0.00.4/ipxe/nxos-image.bin";
```

  ファイル名のパスは **/var/www/html/ipxe/nxos-image.bin**にリダイレクトされ、ipxeブートアップがイメージを検索します。

- TFTP サーバーのインストール

```
[switch conf]# yum install tftp
```

TFTP 構成ファイル **/etc/xinetd.d/tftp**。

次に、TFTP 構成ファイルの例を示します。

```
[switch xinetd.d]# cat tftp
# default: off
# description: The tftp server serves files using the trivial file transfer \
#       protocol.  The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network-aware printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        disable = no
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot        # Indicates the tftp path
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

- TFTP サービスの停止

```
[switch xinetd.d]# chkconfig tftp off
```

- TFTP サービスの開始

```
[switch xinetd.d]# chkconfig tftp on
```

（注） TFTP 構成ファイルを変更した場合、TFTP サービスを再起動する必要があります。

```
host Nexus {
          hardware ethernet 64:f6:9d:07:52:f7;
          fixed-address 10.0.00.8;
          filename = "tftp://100.00.000.48/nxos-image.bin";
```

（注） 前提条件は、上記の TFTP パス **/tftpboot** の例で示されているように、nxos_image.bin を **/tftpboot** にコピーする必要があることです。

- HTTP プロトコルを使用する iPXE

```
switch# sh int mgmt0
mgmt0 is up
admin state is up,
  Hardware: GigabitEthernet, address: e4c7.22bd.c4a6 (bia e4c7.22bd.c4a6)
  Internet Address is 10.0.00.42/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, medium is broadcast
  full-duplex, 100 Mb/s
  Auto-Negotiation is turned on
  Auto-mdix is turned off
  EtherType is 0x0000
  1 minute input rate 312 bits/sec, 0 packets/sec
  1 minute output rate 24 bits/sec, 0 packets/sec
  Rx
    5433 input packets 10 unicast packets 5368 multicast packets
    55 broadcast packets 405677 bytes
  Tx
    187 output packets 9 unicast packets 175 multicast packets
    3 broadcast packets 45869 bytes
switch#

switch# ping 199.00.000.48 vrf management
PING 199.00.000.48 (199.00.000.48): 56 data bytes
64 bytes from 199.00.000.48: icmp_seq=0 ttl=61 time=82.075 ms
64 bytes from 199.00.000.48: icmp_seq=1 ttl=61 time=0.937 ms
64 bytes from 199.00.000.48: icmp_seq=2 ttl=61 time=0.861 ms
64 bytes from 199.00.000.48: icmp_seq=3 ttl=61 time=0.948 ms
64 bytes from 199.00.000.48: icmp_seq=4 ttl=61 time=0.961 ms

--- 199.00.000.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 0.861/17.156/82.075 ms
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# no boot nxos
switch(config)# boot order pxe bootflash
switch(config)# end
```

```
switch# copy running-config startup-config
[########################################] 100%
Copy complete, now saving to disk (please wait)...
Copy complete.
switch# reload
This command will reboot the system. (y/n)?  [n] y

CISCO SWITCH Ver 8.32

CISCO SWITCH Ver 8.32
Memory Size (Bytes): 0x0000000080000000 + 0x0000000380000000
 Relocated to memory
Time: 9/8/2017  1:3:28
Detected CISCO IOFPGA
Booting from Primary Bios
Code Signing Results: 0x0
Using Upgrade FPGA
FPGA Revison        : 0x20
FPGA ID             : 0x1168153
FPGA Date           : 0x20140317
Reset Cause Register: 0x20
Boot Ctrl Register  : 0x60ff
EventLog  Register1 : 0xc2004000
EventLog  Register2 : 0xfbc77fff
Version 2.16.1240. Copyright (C) 2013 American Megatrends, Inc.
Board type  1
IOFPGA @ 0xe8000000
SLOT_ID @ 0x1b
Standalone chassis
check_bootmode: pxe2grub: Launch pxe
Trying to load ipxe
Loading Application:
/Vendor(429bdb26-48a6-47bd-664c-801204061400)/UnknownMedia(6)/EndEntire
iPXE initialising devices...ok


Cisco iPXE
iPXE 1.0.0+ (3cb3) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP DNS TFTP NBI Menu
net6: e4:c7:22:bd:c4:a6 using dh8900cc on PCI02:00.3 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net6 e4:c7:22:bd:c4:a6)................... ok
net0: fe80::2a0:c9ff:fe00:0/64 (inaccessible)
net1: fe80::2a0:c9ff:fe00:1/64 (inaccessible)
net2: fe80::2a0:c9ff:fe00:2/64 (inaccessible)
net3: fe80::2a0:c9ff:fe00:3/64 (inaccessible)
net4: fe80::200:ff:fe00:5/64 (inaccessible)
net5: fe80::200:ff:fe00:7/64 (inaccessible)
net6: 10.0.00.7/255.255.255.0 gw 10.0.00.254
net6: fe80::e6c7:22ff:febd:c4a5/64
net7: fe80::200:ff:fe00:0/64 (inaccessible)
Next server: 10.0.00.4
Filename: http://10.0.00.4/ipxe/nxos-image.bin
http://10.0.00.4/ipxe/nxos-image.bin... ok
http://10.0.00.4/ipxe/nxos_image.bin... 46%
Further device bootsup fine .
```

- TFTP プロトコルの使用

```
switch# sh int mgmt0
mgmt0 is up
admin state is up,
  Hardware: GigabitEthernet, address: e4c7.22bd.c4a6 (bia e4c7.22bd.c4a6)
```

```
  Internet Address is 10.0.00.8/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, medium is broadcast
  full-duplex, 100 Mb/s
  Auto-Negotiation is turned on
  Auto-mdix is turned off
  EtherType is 0x0000
  1 minute input rate 312 bits/sec, 0 packets/sec
  1 minute output rate 24 bits/sec, 0 packets/sec
  Rx
    5433 input packets 10 unicast packets 5368 multicast packets
    55 broadcast packets 405677 bytes
  Tx
    187 output packets 9 unicast packets 175 multicast packets
    3 broadcast packets 45869 bytes
switch#
switch# ping 199.00.000.48 vrf management
PING 199.00.000.48 (199.00.000.48): 56 data bytes
64 bytes from 199.00.000.48: icmp_seq=0 ttl=61 time=82.075 ms
64 bytes from 199.00.000.48: icmp_seq=1 ttl=61 time=0.937 ms
64 bytes from 199.00.000.48: icmp_seq=2 ttl=61 time=0.861 ms
64 bytes from 199.00.000.48: icmp_seq=3 ttl=61 time=0.948 ms
64 bytes from 199.00.000.48: icmp_seq=4 ttl=61 time=0.961 ms

--- 199.00.000.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 0.861/17.156/82.075 ms

switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# no boot nxos
switch(config)# boot order pxe bootflash
switch(config)# end

switch# copy running-config startup-config
[########################################] 100%
Copy complete, now saving to disk (please wait)...
Copy complete.

switch# reload
This command will reboot the system. (y/n)?  [n] y

CISCO SWITCH Ver 8.32

CISCO SWITCH Ver 8.32
Memory Size (Bytes): 0x0000000080000000 + 0x0000000380000000
 Relocated to memory
Time: 9/8/2017  1:3:28
Detected CISCO IOFPGA
Booting from Primary Bios
Code Signing Results: 0x0
Using Upgrade FPGA
FPGA Revison        : 0x20
FPGA ID             : 0x1168153
FPGA Date           : 0x20140317
Reset Cause Register: 0x20
Boot Ctrl Register  : 0x60ff
EventLog  Register1 : 0xc2004000
EventLog  Register2 : 0xfbc77fff
Version 2.16.1240. Copyright (C) 2013 American Megatrends, Inc.
Board type  1
IOFPGA @ 0xe8000000
SLOT_ID @ 0x1b
```

```
Standalone chassis
check_bootmode: pxe2grub: Launch pxe
Trying to load ipxe
Loading Application:
/Vendor(429bdb26-48a6-47bd-664c-801204061400)/UnknownMedia(6)/EndEntire
iPXE initialising devices...ok


Cisco iPXE
iPXE 1.0.0+ (3cb3) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP DNS TFTP NBI Menu
net6: e4:c7:22:bd:c4:a6 using dh8900cc on PCI02:00.3 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net6 e4:c7:22:bd:c4:a6)................... ok
net0: fe80::2a0:c9ff:fe00:0/64 (inaccessible)
net1: fe80::2a0:c9ff:fe00:1/64 (inaccessible)
net2: fe80::2a0:c9ff:fe00:2/64 (inaccessible)
net3: fe80::2a0:c9ff:fe00:3/64 (inaccessible)
net4: fe80::200:ff:fe00:5/64 (inaccessible)
net5: fe80::200:ff:fe00:7/64 (inaccessible)
net6: 10.0.00.7/255.255.255.0 gw 10.0.00.254
net6: fe80::e6c7:22ff:febd:c4a5/64
net7: fe80::200:ff:fe00:0/64 (inaccessible)
Next server: 10.0.00.4
filename: tftp://199.00.000.48/nxos-image.bin
tftp://199.00.000.48/nxos-image.bin... ok
tftp://199.00.000.48/nxos_image.bin... 26%



********************************************************************
```

- プロセスの中断

  プロセスを中断し、iPXE シェルに到達するには、crtl-B を活用します。

  - 次に、 HTTP プロトコルを使用して PXE サーバーに存在するイメージをブートする例を示します：

    ```
    iPXE> dhcp
    Configuring (net6 e4:c7:22:bd:c4:a6)................... ok
    iPXE>boot  http://10.0.0.4/ipxe/nxos-image.bin
    ```

  - 次に、TFTP プロトコルを使用して PXE サーバー上に存在するイメージをブートする例を示します：

    ```
    iPXE> dhcp
    iPXE> boot tftp://199.00.00.48/nxos-image.bin
    ```

  exit を使用します、iPXE シェルを終了します。

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

たとえば、PXE ブート モードのみを実行する場合、コンフィギュレーション 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 を起動するには、次の手順を実行します。

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

最初に PXE を起動し、次に Grub を起動するには、次の手順を実行します。

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

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。**boot order**

（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、次にPXEが失敗した場合はGrub
-g2p	最初にGrub、次にGrubが失敗した場合はPXE

Grub CLI は、完全な Cisco NX-OS イメージを起動せずにシリアル 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、継続的な PXE ブート状態からボックスを取得するためにも使用できます。

iPXE CLI

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

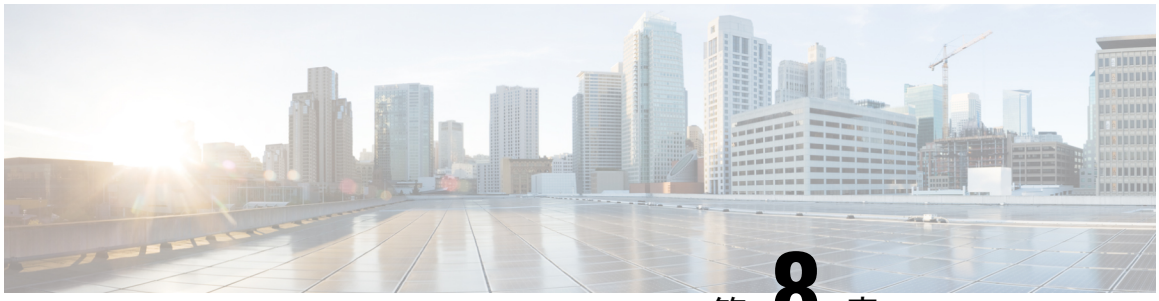
キーワード	機能
– – grub	GRUBのみ
– – pxe	PXE のみ
– – pxe2grub	最初にPXE、次にPXEが失敗した場合はGrub
– – grub2pxe	最初にGrub、次にGrubが失敗した場合はPXE

iPXE CLI は、完全な Cisco NX-OS イメージを起動せずにシリアル 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、継続的な PXE ブート状態からボックスを取得するためにも使用できます。

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	実行コンフィギュレーションからの現在のブート順序と、スタートアップ コンフィギュレーションからの次のリロード時のブート順序値を表示します。



第 8 章

カーネル スタック

この章で説明する内容は、次のとおりです。

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。お客様は、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。

カーネル スタックには次の機能があります：

注意事項と制約事項

カーネル スタックの使用には、次の注意事項と制約事項があります。

- ゲスト シェル、他のオープン コンテナ、およびホスト Bash シェルは、カーネル スタック（kstack）を使用します。
- ホストのデフォルト名前空間で開始されるオープン コンテナ
  - 他のネットワーク名前空間には、**setns** システムコールを使用してアクセスできます。
  - **nsenter** および **ip netns exec** ユーティリティは、異なるネットワーク名前空間のコンテキスト内で実行するために使用できます。
  - **ip netns** コマンドの PID および識別オプションは、ファイル システム デバイス チェックのため、変更なしでは機能しません。ネットワーク管理者に同じ情報を提供する **vrfinfo** ユーティリティが提供されています。

- オープンコンテナは、/proc/net/devからインターフェイス状態を読み取るか、**netstat**または**ifconfig**変更なしで他の通常のLinuxユーティリティを使用できます。これにより、スイッチで開始または終了したパケットのカウンタが提供されます。
- オープンコンテナは、ネットデバイスから拡張統計情報を取得するために**ethtool –S**を使用できます。これには、インターフェイスを介してスイッチングされるパケットが含まれます。
- オープン コンテナは、スイッチで開始または終了したパケットをキャプチャする**tcpdump**などのパケット キャプチャ アプリケーションを実行できます。
- オープン コンテナからのネットワーク状態の変更（インターフェイスの作成/削除、IP アドレスの構成、MTU の変更など）はサポートされていません。
- IPv4 と IPv6 がサポートされます
- Raw PF_PACKET がサポートされています
- ウェルノウンポート（0 ～ 15000）は、ネットワーク名前空間に関係なく、一度に 1 つのスタック（Netstack または kstack）でのみ使用できます。
- Netstack アプリケーションと kstack アプリケーションの間に IP 接続はありません。これは、オープン コンテナにも適用されるホストの制限です。
- オープン コンテナは、ラインカードまたはスタンバイ Sup と通信するために、イーサネットアウトオブバンドチャネル（EOBC）インターフェイスを介してパケットを直接送信することはできません。
- オープン コンテナ内から、ラインカードまたはスタンバイ スーパーバイザとの内部通信に使用されるEOBCインターフェイスに直接アクセスできます。このアクセスが必要な場合は、ホスト bash シェルを使用する必要があります。
- 管理インターフェイス（mgmt0）は、カーネルネットデバイスでeth1として表されます。
- カーネルスタックを使用するアプリケーションでは、VXLANオーバーレイインターフェイス（NVE x）の使用はサポートされていません。CLI コマンドを含む NX-OS 機能は、netstack を介してこのインターフェイスを使用できます。

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。

手順の概要

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

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 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
```

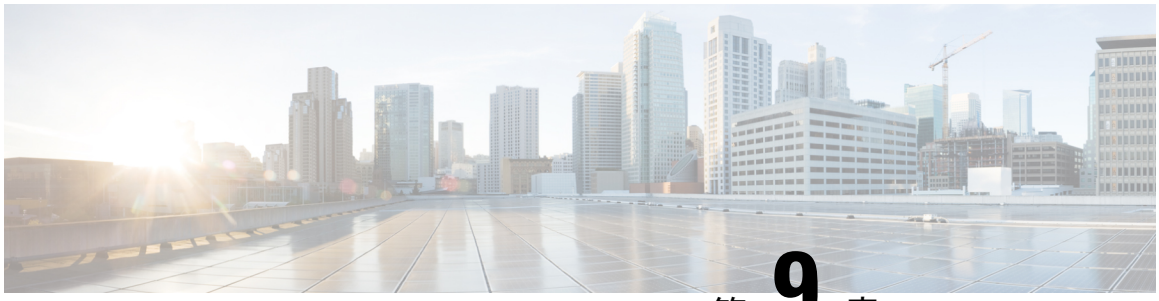
次のタスク

コマンドを入力した後は、次の点に注意する必要があります：

- コマンドを入力した後には、スイッチをリロードする必要があります。
- Netstack で使用される 7000 以上のポートを、未割り当てのままにする必要があります。
- 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 に指定する必要があります。

第 II 部

アプリケーション



第 9 章

サードパーティ製アプリケーション

この章は、次の内容で構成されています。

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/7.0-3-I2-1/x86_64/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。

キーの自動インポートによる署名付きサードパーティ RPM のインストール

キーと RPM を指すように yum リポジトリをセットアップします。

```
root@switch# cat /etc/yum/repos.d/puppet.repo

[puppet]
```

```
name=Puppet RPM

baseurl=file:///bootflash/puppet

enabled=1

gpgcheck=1

gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs

metadata_expire=0

cost=500

bash-4.2# yum install puppet-enterprise

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 ...

puppet                                     |  951 B     00:00 ...

thirdparty                                 |  951 B     00:00 ...

Setting up Install Process

Resolving Dependencies

--> Running transaction check

---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed


--> Finished Dependency Resolution

Dependencies Resolved

================================================================================================


 Package             Arch     Version                              Repository     Size

================================================================================================


Installing:

puppet-enterprise    x86_64    3.7.1.rc2.6.g6cdc186-1.pe.nxos    puppet          14 M

Transaction Summary

================================================================================================


Install       1 Package

Total download size: 14 M
```

```
Installed size: 46 M
Is this ok [y/N]: y
Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs
Importing GPG key 0x4BD6EC30:
 Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"
 From  : /bootflash/RPM-GPG-KEY-puppetlabs
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Warning! Standby is not ready. This can cause RPM database inconsistency.
If you are certain that standby is not booting up right now, you may proceed.
Do you wish to continue?
Is this ok [y/N]: y
Warning: RPMDB altered outside of yum.
Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64
                                                                                1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link
Installed:
puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos

Complete!
```

署名付き RPM のインストール

署名済み RPM の確認

次のコマンドを実行して、特定の RPM が署名されているかどうかを確認します。

Run, ***rpm -K rpm_file_name***

署名付き RPM ではありません

```
bash-4.2# rpm -K bgp-1.0.0-r0.lib32_n3600.rpm

bgp-1.0.0-r0.lib32_n3600.rpm: (sha1) dsa sha1 md5 OK
```

署名付き RPM です

```
bash-4.2#
rpm -K puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm

puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm: RSA sha1 MD5 NOT_OK

bash-4.2#
```

署名されたサードパーティ rpm では、パッケージをインストールする前に公開 GPG キーをインポートする必要があります。そうしないと、**yum** は次のエラーを発生させます：

```
bash-4.2#
yum install puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm -q

Setting up Install Process

warning: rpmts_HdrFromFdno: Header V4 RSA/SHA1 signature: NOKEY, key ID 4bd6ec30

Cannot open: puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm. Skipping.

Error: Nothing to do
```

キーの手動インポートによる署名付き RPM のインストール

- GPG キーを /etc rootfs にコピーして、再起動後も保持されるようにします。

  ```
  bash-4.2# mkdir -p /etc/pki/rpm-gpg

  bash-4.2# cp -f RPM-GPG-KEY-puppetlabs /etc/pki/rpm-gpg/
  ```

- 以下のコマンドを使用して、キーをインポートします。

  ```
  bash-4.2# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs

  bash-4.2#

  bash-4.2# rpm -q gpg-pubkey

  gpg-pubkey-4bd6ec30-4c37bb40
  bash-4.2# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs

  bash-4.2#

  bash-4.2# rpm -q gpg-pubkey

  gpg-pubkey-4bd6ec30-4c37bb40
  ```

- yum コマンドを使用して署名付き RPM をインストールする

```
bash-4.2#
yum install puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.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 puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm:
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64

Marking puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm to be installed


Resolving Dependencies

--> Running transaction check

---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be
installed

--> Finished Dependency ResolutionDependencies Resolved

================================================================================


 Package           Arch     Version                          Repository
  Size

================================================================================


Installing:

puppet-enterprise  x86_64   3.7.1.rc2.6.g6cdc186-1.pe.nxos    /puppet-enterprise-
  46 M
                                                            3.7.1.rc2.6.g6cdc186-1.

                                                              pe.nxos.x86_64


Transaction Summary

================================================================================


Install       1 Package

Total size: 46 M
```

```
Installed size: 46 M

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

  Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64

  1/1

Installed:

  puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos


Complete!

bash-4.2#
```

キーの自動インポートによる署名付きサードパーティ RPM のインストール

キーと RPM を指すように yum リポジトリをセットアップします。

```
root@switch# cat /etc/yum/repos.d/puppet.repo

[puppet]

name=Puppet RPM

baseurl=file:///bootflash/puppet

enabled=1

gpgcheck=1

gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs

metadata_expire=0

cost=500

bash-4.2# yum install puppet-enterprise

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 ...
puppet                                    |  951 B     00:00 ...
thirdparty                                |  951 B     00:00 ...
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed

--> Finished Dependency Resolution
Dependencies Resolved
================================================================================

 Package             Arch     Version                            Repository     Size
================================================================================

Installing:
puppet-enterprise    x86_64    3.7.1.rc2.6.g6cdc186-1.pe.nxos    puppet          14 M
Transaction Summary
================================================================================

Install       1 Package
Total download size: 14 M
Installed size: 46 M
Is this ok [y/N]: y
Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs
Importing GPG key 0x4BD6EC30:
 Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"
 From  : /bootflash/RPM-GPG-KEY-puppetlabs
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
```

```
Warning! Standby is not ready. This can cause RPM database inconsistency.

If you are certain that standby is not booting up right now, you may proceed.

Do you wish to continue?

Is this ok [y/N]: y

Warning: RPMDB altered outside of yum.

Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64
                                                                                1/1

/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link

Installed:

puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos


Complete!
```

リポジトリへの署名済み RPM の追加

手順

ステップ 1 署名済み RPM をリポジトリ ディレクトリにコピーする

ステップ 2 リポジトリの作成を成功させるには、対応するキーをインポートします。

```
bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm  RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# rpm --import RPM-GPG-KEY-puppetlabs
bash-4.2# createrepo .
1/1 - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm
Saving Primary metadata
Saving file lists metadata
Saving other metadata
bash-4.2#
```

キーをインポートしない場合

```
bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm  RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# createrepo .
warning: rpmts_HdrFromFdno: Header V4 RSA/SHA1 signature: NOKEY, key ID 4bd6ec30

Error opening package - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm

Saving Primary metadata
Saving file lists metadata
Saving other metadata
```

ステップ 3 このリポジトリを指す **/etc/yum/repos.d** の下にリポジトリ構成ファイルを作成します。

```
bash-4.2# cat /etc/yum/repos.d/puppet.repo
[puppet]
name=Puppet RPM
baseurl=file:///bootflash/puppet
enabled=1
gpgcheck=1
gpgkey=file:///bootflash/puppet/RPM-GPG-KEY-puppetlabs
#gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
metadata_expire=0
cost=500

bash-4.2# yum list available puppet-enterprise -q
Available Packages
puppet-enterprise.x86_64        3.7.1.rc2.6.g6cdc186-1.pe.nxos
              puppet
bash-4.2#
```

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。

- ローカルリポジトリは、永続的なサードパーティ RPM 専用です。**dnf** /etc/yum/repos.d/ thirdparty.repo は /bootflash/.rpmstore/ thirdparty を指します。
- 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。**dnf install third-party.rpm**
- リブート中に、サードパーティ製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、再起動後は保持されません。

変更した iptables を永続化するには、「」を参照してください。
リロード間で Iptable を永続化する （221 ページ）

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 packages**
2. **install add ?**
3. **install add** *rpm-packagename*

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**show install packages**	すでに存在するパッケージとバージョンを表示します。
ステップ 2	**install add ?**	サポートされている URI を決定します。
ステップ 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 packages
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 アクティベーション（Package Activation）] 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

手順

	コマンドまたはアクション	目的
ステップ 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。
ステップ 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_n3600 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

手順

	コマンドまたはアクション	目的
ステップ 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

手順

	コマンドまたはアクション	目的
ステップ 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

手順

	コマンドまたはアクション	目的
ステップ 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

手順

	コマンドまたはアクション	目的
ステップ 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. インストール 追加 *package-name* アクティベート アップグレード

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	インストール 追加 *package-name* アクティベート アップグレード	パッケージをアップグレードします。

例

次に、パッケージをアップグレードする例を示します：

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.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. インストール 追加 *package-name* アクティベート ダウングレード

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	インストール 追加 *package-name* アクティベート ダウングレード	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_n3600.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.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。

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（パフォーマンス分析）を見つけたり、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非常に低く抑え、同時実行性を高めます。この実装は堅牢であり、

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

Nagios は、ネットワーク サービス（ICMP、 SNMP、 SSH、 FTP、 HTTP などを介して）、ホスト リソース（CPU負荷、ディスク使用率、システム ログなど）、およびサーバー、スイッチ、アプリケーション、およびサービスとのプラグインの接続を確立します。

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効果的に排除するために、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。

スプランク

Splunk は、Web ベースのデータ収集、分析、およびモニタリング ツールであり、ユースケースに合わせて検索、可視化、および事前にパッケージ化されたコンテンツを備えています。raw データは、Splunk Universal Forwarder を使用して Splunk サーバーに送信されます。ユニバーサル フォワーダは、リモート ソースからの信頼性の高いセキュアなデータ収集を可能にし、そのデータをインデックス作成と統合のために Splunk Enterprise に転送します。数万のリモート システムに拡張でき、パフォーマンスへの影響を最小限に抑えながらテラバイト単位のデータを収集できます。

詳細については、http://www.splunk.com/en_us/download/universal-forwarder.html を参照してください。

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、
- 時系列データベース（TSD）への接続を管理し、
- コレクタに TSD コードを埋め込む必要がなくなり、
- 繰り返される値の重複を排除し、
- 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。また、実行時に -w フラグを指定して、パケット データを後で分析するためにファイルに保存することもできます。ネットワーク インターフェイスからではなく保存されたパケット ファイルからデータを読み取るには、-r フラグを指定します。-V フラグで実行することもできます。

それにより、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 は式に一致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。稼働中のネットワークからパケット データをキャプチャしたり、保存済みのキャプチャ ファイルからパケットを読み取ったりできます。読み取ったパケットは、復号された形で標準出力に出力することまたはファイルに書き込むことができます。T-Shark のネイティブ キャプチャ ファイル形式は pcap です。このファイル形式は、**tcpdump** や他のさまざまなツールでも使用されています。TShark は、cap_net_admin ファイル機能を削除した後、ゲスト シェル 2.1 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 10 章

Ansible

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 11 章

Puppet Agent

この章は、次の項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。

- インストールをサポートする スイッチおよびオペレーティング システム ソフトウェア リリースが必要です：
  - Cisco Nexus 3600 プラットフォーム スイッチ。
  - Cisco Nexus 3500 プラットフォーム スイッチ
  - Cisco Nexus 3100 プラットフォーム スイッチ。
  - Cisco Nexus 3000 シリーズ スイッチ。
  - Cisco NX-OS リリース 7.0（3）I5（1）以降。
- 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

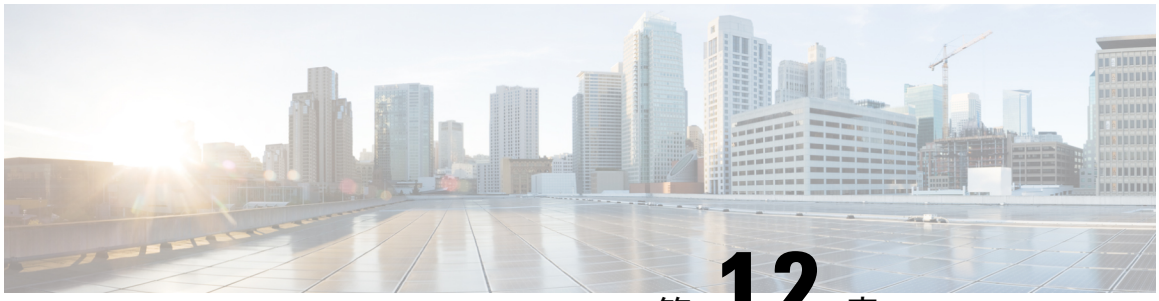
ciscopuppet モジュールは、Cisco が開発したオープン ソース ソフトウェア モジュールです。これは、Puppet マニフェストの抽象技術情報構成と、Cisco NX-OS オペレーティング システム

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 12 章

Cisco NX-OS でのシェフ クライアントの使用

この章は、次の項で構成されています。

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

トピック	リンク
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：

- インストールをサポートするCiscoデバイスおよびオペレーティング システム ソフトウェア リリースが必要です。
  - Cisco Nexus 3600 プラットフォーム スイッチ
  - Cisco Nexus 3500 プラットフォーム スイッチ
  - Cisco Nexus 3100 プラットフォーム スイッチ
  - Cisco Nexus 3000 シリーズ スイッチ
  - Cisco NX-OS リリース 7.0(3)I2(1) またはそれ以降
- 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：スイッチ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

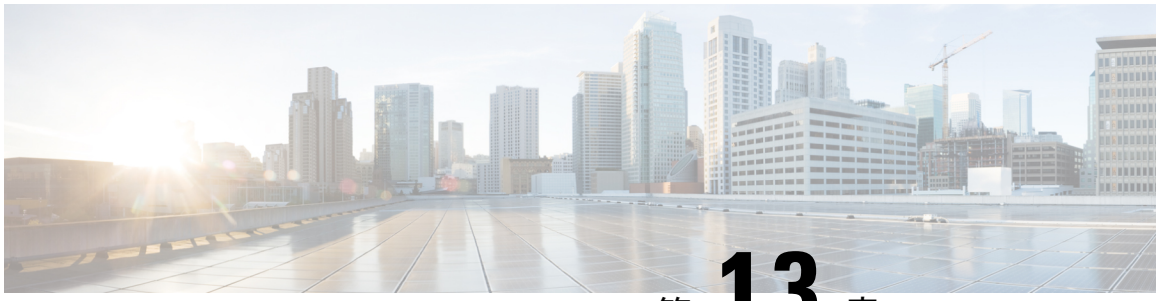
cisco-cookbook

cisco-cookbook は、Chef レシピの抽象情報技術構成と、Cisco Nexus スイッチの特定の実装の詳細との間の、Cisco 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 クックブックの設定と使用方法
Chef Supermarket	Chef Supermarket
Puppet NX-OS マニフェストの例	Cisco Network Chef クックブックのレシピ



第 13 章

Nexus アプリケーション開発：ISO

この章は、次の内容で構成されています。

ISO について

ISO イメージは、サードパーティ製アプリケーションを構築し、スイッチ上でネイティブに実行するために必要なツール、ライブラリ、およびヘッダーを含み、RPM パッケージ化されたブート可能な Wind River 5 環境です。

格納ファイルはすべてを網羅しているわけではなく、特定のアプリケーションに必要な依存関係をユーザーがダウンロードして構築する必要がある場合があります。

（注） 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、構築する必要はありません。

ISO のインストール

ISO イメージは、次の URL からダウンロードできます：http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-I2-1/x86_64/satori-vm-intel-xeon-core.iso。

ISO は、仮想マシンとしてインストールすることを目的としています。仮想化ベンダーの指示に従って、ISO をインストールします。

手順

ステップ 1 （任意） VMware ベースのインストール。

VMWare 仮想マシンに ISO イメージをインストールするには、仮想ディスクを SCSI ではなく SATA として構成する必要があります。

ステップ 2 （任意） QEMU ベースのインストール。

次のコマンドを入力します。

```
bash$ qemu-img create satori.img 10G
bash$ qemu-system-x86_64 -cdrom ./satori-vm-intel-xeon-core.iso -hda ./satori.img -m 8192
```

ISOの起動が開始されると、メニューが表示されます。**[グラフィック コンソールのインストール（Graphics Console Install）]** オプションを選択します。これにより、仮想 HD にインストールされます。インストールが完了したら、仮想マシンを再起動する必要があります。

次のタスク

システムにログインするには、ログインとして **root** を、パスワードとして **root** を入力します。

ISO を使用したアプリケーションの構築 SDK および一般的な Linux で動作するほとんどの構築手順は、ISO 環境にも適用されます。ただし、実行するシェル環境スクリプトはありません。インストールされているツールを使用するには、デフォルトのパスで問題ありません。アプリケーションの送信元コードは、送信元 tar ファイルや git リポジトリなどの通常のメカニズムを使用して取得する必要があります。

送信元コードを構築します。

```
bash$ tar --xvzf 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
```

ISO を使用したアプリケーションの構築

SDK および一般的な Linux で動作するビルド手順のほとんどは、ISO 環境にも適用されます。ただし、実行するシェル環境スクリプトはありません。インストールされているツールを使用するには、デフォルトのパスで問題ありません。アプリケーションの送信元コードは、送信元 tar ファイルや git リポジトリなどの通常のメカニズムを使用して取得する必要があります。

手順

送信元コードを構築ドします。

a) **tar –xvzf example-lib.tgz**
b) **mkdir example-lib-install**
c) **cd example-lib/**
d) ./**configure** –**prefix**=*path_to_example-lib-install*
e) **make**
f) **make install**

手順は通常の Linux です。

例：

次に、送信元コードを構築する例を示します。

```
bash$  tar -xvzf  example-lib.tgz
bash$  mkdir  example-lib-install
bash$  cd example-lib/
bash$  ./configure -prefix=<path_to_example-lib-install>
bash$  make
bach$ make install
```

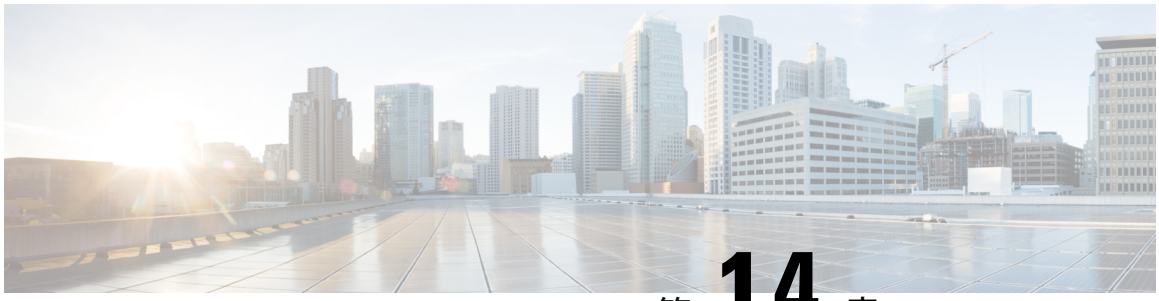
RPM を使用したアプリケーションのパッケージ化

「make」を使用してアプリケーションが正常にビルドされた場合は、RPM にパッケージ化できます。

（注） **RPM および仕様ファイル**

RPMパッケージ形式は、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、設定、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は簡単ではありません。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、tarball にパッケージ化されたソースコードの形式でインターネット上で入手できます。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、ソースコード自体ほど頻繁には更新されません。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、仕様ファイルを編集または最初から作成する必要があります。



第 14 章

Nexus アプリケーション開発：SDK

この章は、次の内容で構成されています。

Cisco SDK について

Cisco SDK は、Yocto 1.2 に基づく開発キットです。Cisco NX-OS リリース 7.0(3)I2(1) 以降を実行するスイッチで実行するアプリケーションを構築するために必要な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。リストはすべてを網羅しているわけではなく、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構築する必要がある場合があります。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使用する準備ができており、構築を必要としないことに注意してください。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。

手順

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_64-wrs-linux コマンドを使用して、SDK 固有のパスをシェル環境に追加します。これは、SDK で使用するシェルごとに実行する必要があります。これは、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ために SDK を設定するためのキーです。

手順

ステップ 1 インストール先ディレクトリを参照します。

ステップ 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_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 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 ファイル形式で、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 を使用してビルドされることを確認します。

```
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。そうでない場合、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構築されます。

```
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場合は、RPM にパッケージ化できます。

（注） **RPM および仕様ファイル**

RPMパッケージ形式は、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、設定、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は簡単ではありません。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、tarball にパッケージ化されたソースコードの形式でインターネット上で入手できます。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、ソースコード自体ほど頻繁には更新されません。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、仕様ファイルを編集または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。

手順

ステップ 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
```

ステップ 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 ファイルを有し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}"
```

ステップ 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、ルート 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、仕様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 仕様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。仕様ファイルで編集された以下を編集します。

```
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。この仕様ファイルは、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コマンドラインに渡す必要があります。**rpmbuild** 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
```

その後、RPMS ディレクトリで collectd の結果の RPM を見つけることができます。

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、現在のユーザーには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 spec 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 15 章

NX-SDK

この章は次のトピックで構成されています。

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。

要件

NX-SDK では次の要件があります。

- Docker
- Linux 環境（Ubuntu 14.04 または Centos 6.7 のいずれか）。提供されている NX-SDK Docker コンテナを使用することをお勧めします。詳細については、「Cisco DevNet NX-SDK」を参照してください。

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 （151 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 （153 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。バージョン .md リンク（）をクリックします。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、Cisco 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 （151 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。https://github.com/CiscoDevNet/NX-SDK/blob/master/versions.md

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 アプリケーションの機能。 • 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。https://wwwin-github.cisco.com/sathsrin/nxsdk/blob/master/include/nx_trace.h

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 （154 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。

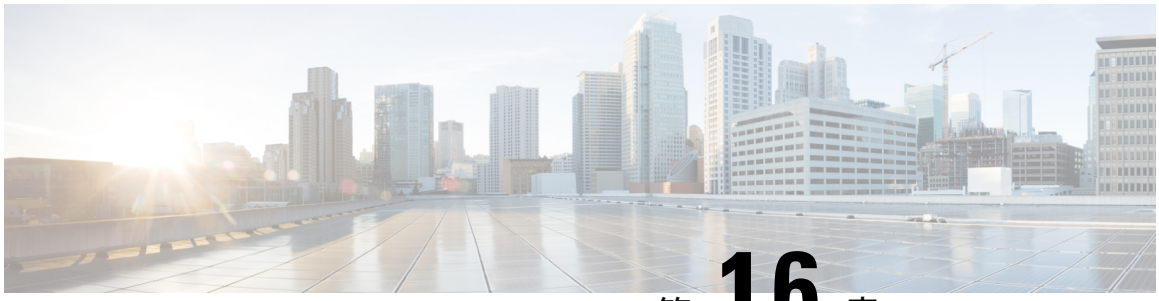
アプリがスイッチを読み書きする機能は、セキュリティプロファイルを介して制御できます。セキュリティプロファイルは、スイッチで実行されているアプリケーションのサービスに付加されるオプションのオブジェクトです。セキュリティプロファイルは、スイッチに書き込むアプリケーションの機能を制御し、スイッチ機能を変更、削除、または構成するアプリケーションの機能を制御します。デフォルトでは、アプリケーションの書き込みは許可されていないため、アプリケーションごとに、スイッチへの書き込みアクセスを有効にするセキュリティプロファイルを作成する必要があります。

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 16 章

Cisco NX-OS での Docker の使用

この章は次のトピックで構成されています。

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。

注意事項と制約事項

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 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。
- 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：

手順

ステップ 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 -
```

ステップ 2 Docker デーモンを起動します。

```
root@switch# service docker start
```

ステップ 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。

手順

ステップ 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 -
```

ステップ 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#
```

ステップ 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#
```

ステップ 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。

手順

ステップ 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 -
```

ステップ 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。

手順

ステップ 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 -
```

ステップ 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
/ #
```

ステップ 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 （167 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。

手順

ステップ 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 -
```

ステップ 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。

手順

ステップ 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 -
```

ステップ 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#
```

ステップ 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。

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。

手順

ステップ 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+'
```

ステップ 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 -
```

ステップ 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#
```

ステップ 4 Docker デーモンを停止します。

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

ステップ 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#
```

ステップ 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#
```

ステップ 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#
```

ステップ 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
```

ステップ 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.
```

ステップ 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
```

ステップ 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':
```

ステップ 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
```

ステップ 13 BASH シェルを終了します。

```
root@switch# exit
logout
switch#
```

ステップ 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。

手順

ステップ 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 -
```

ステップ 2 Docker デーモンを停止します。

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

ステップ 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。

手順

ステップ 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#
```

ステップ 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
```

ステップ 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"
```

ステップ 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。

手順

ステップ 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 -
```

ステップ 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
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

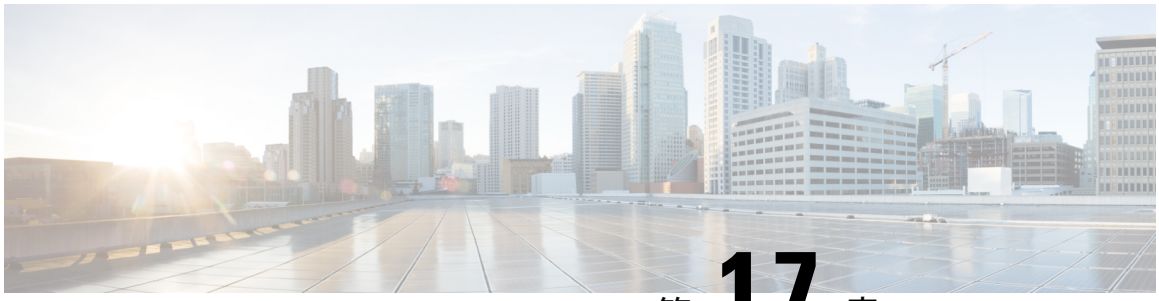
[問題：（Problem:）] 他のトラブルシューティング プロセスを使用しても解決されなかった Docker コンテナに関する他の問題があります。

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

アプリケーション ホスティング



第 17 章

アプリケーション ホスティング

ホステッドアプリケーションは Software as a Service（SaaS）ソリューションであり、コマンドを使用してリモート実行できます。アプリケーションのホスティングによって、管理者には独自のツールやユーティリティを利用するためのプラットフォームが与えられます。

（注） アプリケーション ホスティングは Docker アプリケーションのみをサポートします。

このモジュールでは、アプリケーションホスティング機能とその有効化の方法について説明します。

アプリケーション ホスティングの注意事項と制限事項

この項では、アプリケーション ホスティング機能の注意事項と制限事項について示します。

- コンテナごとに 1 つのインターフェイスのみがサポートされます。
- Cisco NX-OSリリース 10.3(3)F 以降、アプリケーション ホスティング機能は Cisco Nexus 3600 N3K-C36180YC-R、N3K-C3636C-R、および N3K-C36480LD-R2 PID でのみサポートされます。

アプリケーション ホスティングに関する情報

ここでは、アプリケーション ホスティングについて説明します。

アプリケーション ホスティングの必要性

仮想環境への移行により、再利用可能でポータブル、かつスケーラブルなアプリケーションを構築する必要性が高まりました。アプリケーションのホスティングによって、管理者には独自のツールやユーティリティを利用するためのプラットフォームが与えられます。ネットワークデバイスでホストされるアプリケーションは、自動化、設定管理のモニタリング、既存のツールチェーンとの統合など、さまざまな目的で使用できます。

（注） このドキュメントでは、コンテナは Docker アプリケーションを指します。

アプリケーション ホスティングの概要

シスコのアプリケーション ホスティング フレームワークは、デバイス上で実行される仮想化アプリケーションやコンテナアプリケーションを管理する、NX-OS の Python プロセスです。

アプリケーション ホスティングは、次のサービスを提供します。

- コンテナ内の指定されたアプリケーションを起動する。
- 使用可能なリソース（メモリ、CPU、およびストレージ）を確認し、それらを割り当て、管理する。
- REST API を介してサービスへのアクセスを提供する。
- CLI エンドポイントを提供する。
- Cisco Application Framework（CAF）と呼ばれるアプリケーション ホスティング インフラストラクチャを提供する。
- 特別なアプリケーション ブリッジ インターフェイスを介する、プラットフォーム固有のネットワーキング（パケットパス）をセットアップする。

アプリケーション ホスティングのコンテナは、ホスト オペレーティング システムでゲスト アプリケーションを実行するために提供される仮想環境と呼ばれています。Cisco NX-OS アプリケーション ホスティング機能は、ゲストアプリケーションを実行するための管理性とネットワーキングモデルを提供します。仮想化インフラストラクチャにより、管理者はホストとゲスト間の接続を指定する論理インターフェイスを定義できます。Cisco NX-OS は、論理インターフェイスをゲストアプリケーションが使用する仮想ネットワーク インターフェイス カード（vNIC）にマッピングします。

コンテナに展開されるアプリケーションは、.tar または .tar.gz ファイルとしてパッケージ化されています。これらのアプリケーションに固有の構成も、.tar または .tar.gz ファイルの一部としてパッケージ化されています。

アプリケーション ホスティングの設定方法

ここでは、アプリケーションホスティングの設定を構成するさまざまな作業について説明します。

アプリケーション ホスティング機能の有効化

このタスクを実行して、シスコのアプリケーションホスティング機能を有効にします。この機能は、ホストシステム上のアプリケーションの管理、管轄、モニター、トラブルシューティングのためのユーザー インターフェイス コマンドと API インターフェイスを有効にし、関連する様々な活動を実行できるようにします。

手順の概要

1. **configure terminal**
2. **feature app-hosting**
3. **end**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 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。
ステップ 2	**feature app-hosting** 例： `switch(config)# feature app-hosting`	シスコ アプリケーション ホスティング機能を有効にします。
ステップ 3	**end** 例： `switch(config)# end`	グローバル コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。

アプリケーション ホスティング ブリッジ接続の設定

アプリケーション コンテナへのレイヤ 3 接続には、独自のエンドポイント IPv4 アドレスが必要です。NX-OS では、アプリケーション ホスティング ブリッジと呼ばれる仮想ブリッジ メカニズムが、Cisco Nexus スイッチ内のアプリケーション コンテナをホストします。

ブリッジは、アプリケーション コンテナへのゲートウェイとして機能し、アタッチされた VRF ルーティング コンテキストにトラフィックをルーティングするのに役立ちます。ブリッジは、VRF コンテキストごとに、スイッチ インターフェイスを介してアプリケーションのサブネット トラフィックを転送します。

スイッチ インターフェイスを介するネットワーク接続を使用したアプリケーション コンテナのホスティングには、少なくとも 2 つの割り当て可能なアドレスを持つ専用のエンドポイント IP サブネットが必要です。1 つの IP アドレスはアプリケーション コンテナのゲスト インターフェイス用で、もう 1 つの IP アドレスはアプリケーション コンテナのゲートウェイ用です。

内部的には、アプリケーション コンテナのゲスト インターフェイスは、アプリケーション ホスティング仮想ブリッジからは独立した、仮想ネットワーク インターフェイス カード（vNIC）です。

手順の概要

1. **configure terminal**
2. **app-hosting bridge** *bridge-index*
3. **ip address** *ip-address/mask*
4. **vrf member** *name*
5. **exit**
6. **app-hosting appid** *name*
7. **app-vnic gateway bridge** *bridge-index* **guest-interface** *guest-interface-number*
8. **guest-ipaddress** *ip-address/mask*
9. **exit**
10. **app-default-gateway** *ip-address* **guest-interface** *guest-interface*
11. **end**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 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。
ステップ 2	**app-hosting bridge** *bridge-index* 例： `switch(config)# app-hosting bridge 1`	アプリケーションホスティングブリッジを設定し、アプリケーション ホスティング ブリッジ構成モードを開始します。

	コマンドまたはアクション	目的
		• <1-8> ブリッジ インデックス
ステップ 3	**ip address** *ip-address/mask* 例： `switch(config-app-hosting-bridge)# ip address 172.25.44.1/30`	アプリケーション コンテナへのゲートウェイとして機能するアプリケーション ブリッジ IPv4 アドレスを設定します。 （注） IP がインターフェイスまたは仮想 IP のいずれかによって使用されている場合、サブネットは拒否されます。
ステップ 4	**vrf member** *name* 例： `switch(config-app-hosting-bridge)# vrf member overlay-VRF`	VRF コンテキストを設定します。構成していなければ、VRF のデフォルトに属しています。
ステップ 5	**exit** 例： `switch(config-app-hosting-bridge)# exit`	アプリケーションブリッジ構成モードを終了して、グローバル構成モードに戻ります。
ステップ 6	**app-hosting appid** *name* 例： `switch(config)# app-hosting appid te_app`	アプリケーションを構成し、アプリケーション ホスティング構成モードを開始します。
ステップ 7	**app-vnic gateway bridge** *bridge-index* **guest-interface** *guest-interface-number* 例： `switch(config-app-hosting)# app-vnic gateway bridge 1 guest-interface 0`	アプリケーションのゲスト VNIC インターフェイスを設定し、アプリケーション ホスティング vnic インターフェイス モードを開始します。 （注） Cisco NX-OS リリース 10.3(3)F 以降では、1 つの VNIC のみを構成できます。
ステップ 8	**guest-ipaddress** *ip-address/mask* 例： `switch(config-app-hosting-app-vnic)# guest-ipaddress 172.25.44.2/30`	ブリッジ *1* サブネットから使用可能な IPv4 アドレスの 1 つを設定します。
ステップ 9	**exit** 例： `switch(config-app-hosting-app-vnic)# exit`	アプリケーション vnic インターフェイス構成モードを終了して、アプリケーション ホスティング構成モードに戻ります。
ステップ 10	**app-default-gateway** *ip-address* **guest-interface** *guest-interface* 例：	ブリッジ *1* サブネットから使用可能な IPv4 アドレスを構成します。 ステップ 3 で説明されたようにゲートウェイ アドレスを構成します。

	コマンドまたはアクション	目的
	`switch(config-app-hosting-appid)# app-default-gateway 172.25.44.1 guest-interface 0`	
ステップ 11	**end** 例： `switch(config-app-hosting)# end`	アプリケーション ホスティング構成モードを終了し、特権 EXEC モードに戻ります。

アプリケーションのライフサイクル

次の EXEC コマンドは、アプリケーションをアップグレードする方法を示しています。

（注） アプリケーションのインストール後に構成の変更を行った場合、実行状態のアプリケーションにはこれらの変更が反映されません。アプリケーションの起動後に変更を加えるには、変更を行う前にアプリケーションを停止して非アクティブ化し、アプリケーションを再度アクティブ化して起動します。

手順の概要

1. **enable**
2. **app-hosting install appid** *application-name* **package** *package-path*
3. **app-hosting activate appid** *application-name*
4. **app-hosting start appid** *application-name*
5. **app-hosting stop appid** *application-name*
6. **app-hosting deactivate appid***application-name*
7. **app-hosting uninstall appid** *application-name*

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `switch# enable`	特権 EXEC モードを有効にします。 • プロンプトが表示されたら、パスワードを入力します。
ステップ 2	**app-hosting install appid** *application-name* **package** *package-path* 例：	指定した場所からアプリケーションをインストールします。

	コマンドまたはアクション	目的
	`switch# app-hosting install appid te_app package bootflash:my_te_app.tar`	• ローカルの保管場所（つまり、ブートフラッシュ）からアプリケーションをインストールすることができます。
ステップ 3	**app-hosting activate appid** *application-name* 例： `switch# app-hosting activate appid te_app`	アプリケーションをアクティブ化します。 • このコマンドは、すべてのアプリケーションリソース要求を検証します。すべてのリソースが使用可能な場合、コマンドはアプリケーションをアクティブにします。それ以外の場合、アクティブ化は失敗します。
ステップ 4	**app-hosting start appid** *application-name* 例： `switch# app-hosting start appid te_app`	アプリケーションを起動します。 • アプリケーションの起動スクリプトをアクティブにします。
ステップ 5	**app-hosting stop appid** *application-name* 例： `switch# app-hosting stop appid te_app`	（任意）アプリケーションを停止します。
ステップ 6	**app-hosting deactivate appid***application-name* 例： `switch# app-hosting deactivate appid te_app`	（任意）アプリケーションに割り当てられているすべてのリソースを無効にします。
ステップ 7	**app-hosting uninstall appid** *application-name* 例： `switch# app-hosting uninstall appid te_app`	（任意）アプリケーションをアンインストールします。 • 保存されているすべてのパッケージとイメージをアンインストールします。また、アプリケーションに加えられたすべての変更とアップグレードを削除します。

アプリケーションのアップグレード

次の EXEC コマンドは、アプリケーションをアップグレードする方法を示しています。

手順の概要

1. **enable**
2. **switch# app-hosting upgrade appid** *application-name* **package** *package-path*

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `switch# enable`	特権 EXEC モードを有効にします。 • プロンプトが表示されたら、パスワードを入力します。
ステップ 2	**switch# app-hosting upgrade appid** *application-name* **package** *package-path* 例： `switch# app-hosting upgrade appid tea package bootflash:thousandeyes-enterprise-agent-4.1.0.cisco.tar`	既存のアプリケーションを新しいバージョンにアップグレードします。その間、このコマンドはアプリケーションを停止し、アップグレードし、新しいアプリケーションイメージでアップグレード前の状態に戻します。 （注） • STOPPED 状態のアプリケーションをアップグレードした場合、アップグレードが正常に完了すると、新しい app-hosting 状態は ACTIVATED に変わります。 • ローカルの保管場所（つまり、ブートフラッシュ）からアプリケーションをアップグレードすることができます。

Docker ランタイムオプションの設定

最大 30 行のランタイムオプションを追加できます。システムは、1 行目から 30 行目までの連結文字列を生成します。文字列には、複数の Docker ランタイム オプションを指定できます。

（注） 実行時オプションを変更するには、アプリケーションが非アクティブ状態になっている必要があります。

手順の概要

1. **enable**
2. **configure terminal**
3. **app-hosting appid** *application-name*
4. **app-resource docker**
5. **run-opts** *options*
6. **end**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `switch# enable`	特権 EXEC モードを有効にします。 • プロンプトが表示されたら、パスワードを入力します。
ステップ 2	**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。
ステップ 3	**app-hosting appid** *application-name* 例： `switch(config)# app-hosting appid te_app`	アプリケーションを設定し、アプリケーション ホスティング コンフィギュレーション モードを開始します。
ステップ 4	**app-resource docker** 例： `switch(config-app-hosting)# app-resource docker`	アプリケーション ホスティング Docker コンフィギュレーション モードを開始して、アプリケーション リソースの更新を指定します。
ステップ 5	**run-opts** *options* 例： `switch(config-app-hosting-docker)# run-opts 1 "-v $(APP_DATA):/data"`	Docker ランタイム オプションを指定します。
ステップ 6	**end** 例： `switch(config-app-hosting-docker)# end`	アプリケーション ホスティング Docker コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。

管理インターフェイスでのアプリケーション ホスティングの構成

NX-OS を使用すると、アプリケーション コンテナは Cisco NX-OS 管理インターフェイスを介してネットワーク接続を共有できます。仮想 NAT ブリッジを内部的に設定し、ゲスト vNIC インターフェイスにプライベート IP アドレスを割り当てることができます。ゲスト インターフェイスのプライベート IP アドレスは、Apphosting フレームワークによって自動的に割り当てられます。

手順の概要

1. **enable**
2. **configure terminal**
3. **app-hosting appid** *name*
4. **app-vnic management guest-interface** *network-interface*

5. **end**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `switch# enable`	特権 EXEC モードを有効にします。 • プロンプトが表示されたら、パスワードを入力します。
ステップ 2	**configure terminal** 例： `switch# configure terminal`	グローバル構成モードを開始します。
ステップ 3	**app-hosting appid** *name* 例： `switch(config)# app-hosting appid te_app`	アプリケーションを設定し、アプリケーションホスティング コンフィギュレーション モードを開始します。
ステップ 4	**app-vnic management guest-interface** *network-interface* 例： `switch(config-app-hosting)# app-vnic management guest-interface 0`	ゲストインターフェイスを管理ポートに接続し、アプリケーションホスティング管理ゲートウェイ コンフィギュレーション モードを開始します。 • **management** キーワードは、プライベート IPNAT モードでコンテナに接続する Cisco NX-OS インターフェイス *mgmt0* を指定します。 • **guest-interface***network-interface* のキーワード引数ペアは、Cisco NX-OS 管理インターフェイス *mgmt0* に接続されているコンテナの内部イーサネットインターフェイス番号を指定します。この例では、コンテナのイーサネット 0 インターフェイスに対して *guest-interface 0* を使用しています。
ステップ 5	**end** 例： `switch(config-app-hosting-mgmt-gateway)# end`	アプリケーションホスティング管理ゲートウェイ コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。

アプリケーションのリソース設定の上書き

リソースの変更を有効にするには、最初に **app-hosting stop** および **app-hosting deactivate** コマンドを使用してアプリケーションを停止して非アクティブ化し、次に **app-hosting activate** および **app-hosting start** コマンドを使用してアプリケーションを再起動する必要があります。

これらのコマンドを使用して、リソースと app-hosting appid 構成の両方をリセットできます。

手順の概要

1. **enable**
2. **configure terminal**
3. **app-hosting appid** *name*
4. **app-resource profile** *name*
5. **cpu** *unit*
6. **memory** *memory*
7. **end**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `switch# enable`	特権 EXEC モードを有効にします。 • プロンプトが表示されたら、パスワードを入力します。
ステップ 2	**configure terminal** 例： `switch# configure terminal`	グローバル構成モードを開始します。
ステップ 3	**app-hosting appid** *name* 例： `switch(config)# app-hosting appid te_app`	アプリケーション ホスティングをイネーブルにし、アプリケーション ホスティング コンフィギュレーション モードを開始します。
ステップ 4	**app-resource profile** *name* 例： `switch(config-app-hosting)# app-resource profile custom`	カスタム アプリケーション リソース プロファイルを設定し、カスタム アプリケーション リソース プロファイル コンフィギュレーション モードを開始します。 • カスタムプロファイル名のみがサポートされています。
ステップ 5	**cpu** *unit* 例：	アプリケーションのデフォルトの CPU 割り当てを変更します。

	コマンドまたはアクション	目的
	`switch(config-app-resource-profile-custom)# cpu 7400`	• リソース値はアプリケーション固有のため、これらの値を変更した場合、アプリケーションが変更後も確実に稼働できることを確認する必要があります。
ステップ 6	**memory** *memory* 例： `switch(config-app-resource-profile-custom)# memory 2048`	デフォルトのメモリ割り当てを変更します。
ステップ 7	**end** 例： `switch(config-app-resource-profile-custom)# end`	カスタム アプリケーション リソース プロファイル構成モードを終了し、特権 EXEC モードに戻ります。

高度なアプリケーション ホスティング機能

デフォルトでは、アプリケーション ホスティング機能は、シスコがサポートする署名付きアプリケーション パッケージのみを許可します。シスコ以外の署名付きアプリケーション Docker イメージをインストールするには、署名検証機能をオフにする必要があります。これはグローバル設定であり、インストールされているすべてのアプリケーションに影響します。**app-hosting signed-verification** [**disable** | **enable**] コマンドは、署名検証を無効にし、シスコ以外の Docker アプリケーションのインストールを支援します。

アプリケーション ホスティング機能が設定されている場合、ブートフラッシュから 2 GB のファイル スペースがアプリケーション ストレージ スペースとして予約されます。特定のアプリケーションにさらに多くの領域が必要な場合は、パーティションサイズを増やすことができます。または、**app-hosting bootflash backend storage limit size** グローバル構成コマンドを使用して、アプリケーションのスペース要件に基づき、スペースを減らすこともできます。アプリケーションは再起動します。

手順の概要

1. **enable**
2. **configure terminal**
3. **app-hosting signed-verification [disable | enable]**
4. **app-hosting bootflash backend storage limit** *size*
5. **end**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	**enable** 例： `switch# enable`	特権 EXEC モードを有効にします。 • プロンプトが表示されたら、パスワードを入力します。
ステップ 2	**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。
ステップ 3	**app-hosting signed-verification [disable | enable]** 例： `switch(config)# app-hosting signed-verification disable`	パッケージ検証を無効にして、シスコ以外のアプリケーションを許可します。 • 署名による検証はデフォルトで有効になっています。
ステップ 4	**app-hosting bootflash backend storage limit** *size* 例： `switch(config)# app-hosting bootflash backend storage limit 600`	インストールするすべてのアプリケーションを考慮して、必要なアプリケーション ストレージ サイズを設定します。 • デフォルトでは 2048 MB が使用されます。 • サイズは MB 単位で指定します。ブートフラッシュの使用可能な空き領域よりも小さくする必要があります。
ステップ 5	**end** 例： `switch(config-app-resource-profile-custom)# end`	カスタム アプリケーション リソース プロファイル構成モードを終了し、特権 EXEC モードに戻ります。

アプリケーション データのコピー

アプリケーションの永続データマウントからアプリケーションデータを削除するには、特権 EXEC モードで**app-hosting data appid** <*appid*> **copy**コマンドを使用します。

```
app-hosting data appid tea copy bootflash:src dest
```

値は次のとおりです。

`src` はブートフラッシュからの送信元ファイルで、`dest` は接続先ファイル パスです。

アプリケーション データの削除

アプリケーションの永続データマウントからアプリケーションデータを削除するには、特権 EXEC モードで**app-hosting data appid** <*appid*> **delete**コマンドを使用します。

```
app-hosting data appid tea delete file
```

値は次のとおりです。

`file` は、アプリケーションの永続データマウントから削除されるファイルです。

アプリケーション ホスティング設定の確認

show コマンドを使用して設定を確認します。任意の順序でこれらのコマンドを使用できます。

手順の概要

1. **show app-hosting infra**
2. **show app-hosting list**
3. **show app-hosting bridge**
4. **show app-hosting detail**
5. **show app-hosting resource**
6. **show app-hosting app-hosting utilization appid <app-name>**
7. **show-tech app-hosting**

手順の詳細

手順

ステップ 1 **show app-hosting infra**

アプリケーションホスティング インフラの概要を表示します。

（注）
さらに操作を実行する前に、CAF を動作状態に移行します。

例：

```
switch(config)# show app-hosting infra
App signature verification: disabled
Docker partition size: 0 MB
Inband packet rate limit: 0 PPS
Services
-----------------------------------
CAF 1.16.0.0 : Running
HA : Running
App Manager : Running
Libvirtd 4.7.0 : Running
```

```
Dockerd 18.09.0-ce : Running
Linux kernel 5.10.126 : Running
```

ステップ 2 show app-hosting list

動作しているアプリのリストを表示します。

例：

```
switch(config)# show app-hosting list
App id               State
---------------------------------------------------------
nginx_1              started
```

ステップ 3 show app-hosting bridge

アプリケーションホスティング ブリッジのリストを表示します。

例：

```
switch(config)# show app-hosting bridge
Bridge ID   VRF       IP Address       IPv6 Address
------------------------------------------------------------------------------
1           blue       172.10.23.45/24  ::/0
```

ステップ 4 show app-hosting detail

プリケーションホスティングについての詳細情報を表示します。

例：

```
switch(config)# show app-hosting detail
App id : nginx_1
Owner : appmgr
State : started
Application
Type : docker
Name : nginx
Version : latest
Description :
Author :
Path : /bootflash/nginx.tar.gz
URL Path :
Activated profile name : default

Resource reservation
Memory : 64 MB
Disk : 10 MB
CPU : 200 units

Platform resource profiles
Profile Name CPU(unit) Memory(MB) Disk(MB)
--------------------------------------------------------------

Attached devices
Name               Type               Alias
---------------------------------------------
iox_trace          serial/trace       serial3
iox_syslog         serial/syslog      serial2
iox_console_aux    serial/aux         serial1
iox_console_shell serial/shell        serial0

Network interfaces
---------------------------------------
eth0:
```

```
MAC address : 5254.9999.0000
IPv4 address : 192.168.10.130
IPv6 address : fe80::5054:99ff:fe99:0/64
Network name : iox-nat_docker0
Tx Packets : 9
Tx Bytes : 726
Tx Errors : 0
Rx Packets : 0
Rx Bytes : 0
Rx Errors : 0

Docker
------
Run-time information
Command :
Entry-point : /docker-entrypoint.sh nginx -g 'daemon off;'
Run options in use : --publish=40080:80
Package run options :
Application health information
Status : 0
Last probe error :
Last probe output :
```

ステップ 5 show app-hosting resource

アプリケーションホスティングのリソースに関する情報を表示します。

例：

```
switch(config)# show app-hosting resource
CPU:
Total: 7400 units
Available: 7200 units
VCPU:
Application Hosting
Additional References
Application Hosting
46
Count: 1
Memory:
Total: 3840(MB)
Available: 3776(MB)
Storage space:
Total: 110745(MB)
Available: 93273(MB)
vice
```

ステップ 6 show app-hosting app-hosting utilization appid <app-name>

アプリケーションの使用率を表示します。

例：

```
switch(config)# show app-hosting utilization appid nginx_1
Application: nginx_1
CPU Utilization:
CPU Allocation: 200 units
CPU Used: 0 %
Memory Utilization:
Memory Allocation: 64 MB
Memory Used: 7000 KB
Disk Utilization:
Disk Allocation: 10 MB
Disk Used: 0 MB
```

ステップ 7 show-tech app-hosting

すべてのアプリケーションホスティング ログと、関連する依存コンポーネント ログを表示します。

この show-tech コマンドは、次の show コマンドの詳細を収集します。

例：

```
show system internal app-hosting
show system internal app-hosting event-history debug
show system internal app-hosting event-history error
show system internal app-hosting event-history msgs
show app-hosting list
show app-hosting detail
show app-hosting utilization
show app-hosting infra
show app-hosting resource
show app-hosting bridge
show routing appmgr vrf all
show routing ipv6 appmgr vrf all
```

アプリケーション ホスティングの設定例

次に、アプリケーション ホスティング機能の設定に関するさまざまな例を示します。

例：AppHosting 機能の有効化

次の例は、Cisco Apphosting 機能を有効にする方法を示しています。

```
switch# configure terminal
switch(config)# feature app-hosting
switch(config)# end
```

例：アプリケーション ホスティング ブリッジ接続の構成

この例は、アプリケーション ホスティング ブリッジ接続を構成する方法を示しています。

```
switch(config)# configure terminal
switch(config)# app-hosting bridge 1
switch(config-app-hosting-bridge)# ip address 172.25.44.1/30
switch(config-app-hosting-bridge)# vrf member overlay-VRF
switch(config-app-hosting-bridge)# exit
switch(config)# app-hosting appid te_app
switch(config- app-hosting)# app-vnic bridge 1 guest-interface 0
switch(config-app-hosting-app-vnic)# guest-ipaddress 172.25.44.2/30
switch(config-app-hosting-app-vnic)# exit
switch(config-app-hosting-appid)# app-default-gateway 172.25.44.1 guest-interface 0
switch(config-app-hosting)# end
```

例：Docker ランタイムオプションの設定

この例では、Docker ランタイムオプションを設定する方法を示します。

```
switch> enable
switch# configure terminal
switch(config)# app-hosting appid te_app
switch(config-app-hosting)# app-resource docker
switch(config-app-hosting-docker)# run-opts 1 "-v $(APP_DATA):/data"
switch(config-app-hosting-docker)# end
```

例：管理インターフェイスでのアプリケーションホスティングの構成

次に、管理インターフェイスでアプリケーション ホスティングを構成する例を示します。

```
switch> enable
switch# configure terminal
switch(config)# app-hosting appid te_app
switch(config-app-hosting)# app-vnic management guest-interface 0
switch(config-app-hosting)# end
```

例：アプリケーションのリソース設定の上書き

この例では、アプリケーションのリソース設定を上書きする方法を示します。

```
switch> enable
switch# configure terminal
switch(config)# app-hosting appid te_app
switch(config-app-hosting)# app-resource profile custom
switch(config-app-resource-profile-custom)# cpu 7400
switch(config-app-resource-profile-custom)# memory 2048
switch(config-app-resource-profile-custom)# end
```

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
アプリケーション ホスティングの構成	Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference（Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK ユーザ ガイドと API リファレンス）

テクニカル サポート

説明	リンク
Cisco のサポート Web サイトでは、Cisco の製品やテクノロジーに関するトラブルシューティングにお役立ていただけるように、マニュアルやツールをはじめとする豊富なオンライン リソースを提供しています。 お使いの製品のセキュリティ情報や技術情報を入手するために、Cisco Notification Service（Field Notice からアクセス）、Cisco Technical Services Newsletter、Really Simple Syndication（RSS）フィードなどの各種サービスに加入できます。 シスコのサポート Web サイトのツールにアクセスする際は、Cisco.com のユーザ ID およびパスワードが必要です。	http://www.cisco.com/support

アプリケーション ホスティングに関する機能情報

次の表に、このモジュールで説明した機能に関するリリース情報を示します。その機能は、特に断りがない限り、それ以降の一連のソフトウェア リリースでもサポートされます。

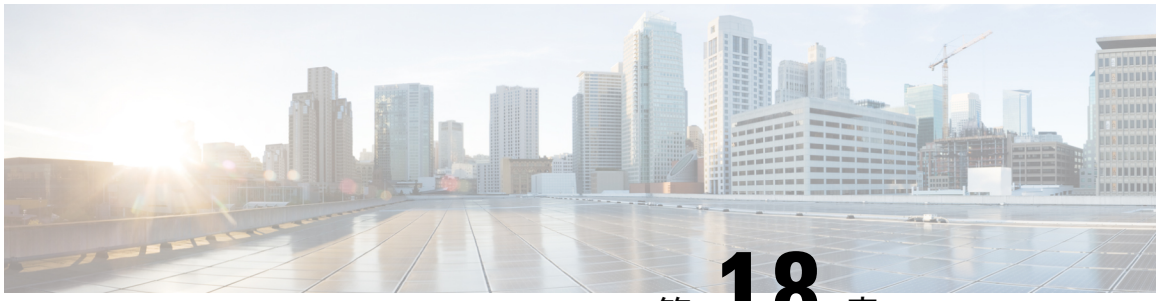
プラットフォームのサポートおよびシスコ ソフトウェアイメージのサポートに関する情報を検索するには、Cisco Feature Navigator を使用します。Cisco Feature Navigator にアクセスするには、www.cisco.com/go/cfn にアクセスしてください。Cisco.com のアカウントは必要ありません。

表 *5:* アプリケーション ホスティングに関する機能情報

機能名	リリース	機能情報
Cisco アプリケーション ホスティング フレームワーク（CAF）	Cisco NX-OS リリース 10.3(3)F	ホスト対象のアプリケーションは Software as a Service（SaaS）ソリューションであり、ユーザーはこのソリューションの実行と運用を完全にクラウドから行うことができます。このモジュールでは、Cisco アプリケーション ホスティング機能とその有効化の方法について説明します。 アプリケーション ホスティング機能は、以下の PID でCisco Nexus 3600 シリーズでサポートされています。 • N3K-C36180YC-R • N3K-C3636C-R • N3K-C36480LD-R2

第 IV 部

NX-API



第 18 章

NX-API CLI

NX-API CLI について

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。

NX- API CLI は、Cisco Nexus スイッチでの JSON/ CLI の実行もサポートしています。

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。

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。Nginx のメモリ使用量が cgroup の制限を超えると、Nginx プロセスが再起動されて復元されます。

（注） 7.x リリースの場合、「no feature NXAPI」コマンドを使用して NX-API を無効にした後でも、Nginx プロセスは引き続き実行されます。これは、他の管理関連プロセスに必要です。6.x リリースでは、「no feature NXAPI」コマンドを実行するとすべてのプロセスが強制終了されていました。これは、7.x リリースでの動作の変更です。

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

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 （223ページ）を参照してください。

デバイスで **feature** manager CLI コマンドを使用して NX-API を有効にする必要があります。デフォルトでは、NX-API は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
switch(config)# interface mgmt 0
switch(config)# ip address 192.0.20.123/24
switch(config)# vrf context managment
switch(config)# ip route 10.0.113.1/0 1.2.3.1
```

- 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"
            }
        }
    }
}
```

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 *6 : 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。

NX-API 管理コマンド	説明
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/設定情報を表示しません。
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**
ファイル名パスフレーズ **nxapi certificatehttpskey keyfile password**	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 trustpoint <trustpoint label>	

NX-API 管理コマンド	説明
	Cisco NX-OS リリース 10.2(3)F 以降では、トラストポイント インフラを使用して NX-API の証明書をインポートするか、CA 証明書を使用できるようになりました。 （注） • 最初に証明書をインポートするように crypto ca import トラストポイントを設定するには、『Cisco Nexus 9000 Security Configuration Guide』を参照してください。 • 現在、この形式では pkcs12 証明書のインポートのみがサポートされています。NX-API 証明書の有効化/NX-API 証明書のトラストポイントと NX-API 証明書の SUDI は相互に排他的であり、各設定によって証明書/キーが上書きされます。 • NX-API 証明書の有効化でサポートされる証明書/キーの最大サイズは 8k です。サイズが 8k を超える場合は、NX-API 証明書トラストポイントを使用して証明書をインポートします。 • トラストポイント インフラを使用して NX-API でカスタム証明書を設定した場合、reload ascii コマンドを入力すると、設定が失われます。デフォルトの day-1 NX-API 証明書に戻ります。reload ascii コマンドを入力すると、スイッチがリロードされます。スイッチが再び起動したら、NX-API 証明書トラストポイントの設定を再設定する必要があります。 • Cisco NX-OS リリース 10.3(1)F 以降では、ASCII トラストポイントリロードのサポートが追加されています。 • 現在の実行コンフィギュレーションにトラストポイントとインポートされた証明書が含まれていないが、ターゲットコンフィギュレーションにトラストポイント「crypto ca trustpoint」の作成が含まれている場合、コンフィギュレーション置換は失敗します。<trustpoint name> " および "nxapi certificate trustpoint<trustpoint-name> "CLI を選択します。トラストポイントが存在しない場合は、最初にトラストポイントを作成し、証明書をインポートする必要があります。<trustpoint-label> "。 • NX-API に関連付けられている証明書またはトラストポイントが削除されると、NX-API の現在のイン

NX-API 管理コマンド	説明
	スタンスは引き続き機能しますが、リンクは切断されます。NX-API 構成の変更または再起動により、NX-API はデフォルトの証明書で実行され、**show nxapi** はこれらの詳細を表示します。 NX-API に関連付けられている証明書とトラストポイントが再度追加されると、NX-API は自動的に再起動され、引き続き機能します。 暗号化証明書を使用した ASCII リロードまたは ISSU の場合、システムの準備後にすべての証明書が復元されるまで、NGINX の再起動が複数回発生する可能性があります。 NX-API を使用するために証明書が復元されるのを待ちます

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 証明書を使用する場合、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。 • 「nxapi certificate sudi」は、設定されている場合にカスタム証明書/キーを上書きし、カスタム証明書/キーを元に戻す方法はありません。 • 「nxapi certificate sudi」と「nxapi certificate trustpoint」と「nxapi certificate enable」は相互に排他的であり、一方を設定するともう一方の設定が削除されます。 • 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。 • NX-API 証明書 CLI は show run の出力に存在しないため、現在、CR/ロールバックの場合は、「nxapi certificate sudi」オプションで上書きされるとカスタム証明書に戻りません。 • Cisco NX-OS リリース 10.5(2)F 以降、「nxapi certificate sudi」は Cisco NX-OS スイッチの FIPS モードでブロックされます。
no nxapi certificate sudi	これにより、SUDI が無効になり、NX-API にはデフォルトの自己署名証明書が付属します。

NX-API 管理コマンド	説明
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、デフォルトに変更されます（デフォルトでは、弱い暗号は無効になります）。

NX-API 管理コマンド	説明

nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2 TLSv1.3}	Cisco NX-OS リリース 10.2(4)M 以降、TLSv1.3 が Cisco Nexus 9000 シリーズプラットフォーム スイッチでサポートされています。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。Cisco NX-OS リリース 9.3(2) 以降では、TLSv1.2 のみがデフォルトで有効になっています。
	
	このコマンドの no 形式を使用すると、TLS バージョンがデフォルト バージョンに変更されます。
	
	• 特定の TLS バージョンを有効にする場合は、それぞれの TLS バージョンのみを指定します。
	
	たとえば、TLSv1.3 が必要な場合は、次のコマンドを使用します。
	
	`switch(config)# nxapi ssl protocols TLSv1.3`
	
	• 後の段階で後方互換性のために複数の TLS バージョンを有効にする場合は、サポートされていて必要なすべての TLS バージョンを指定します。
	
	次に例を示します。
	
	• TLSv1.1 ～ TLSv1.3 が必要な場合は、次のコマンドを使用して、必要なすべての TLS バージョンを有効にします。
	
	`switch(config)# nxapi ssl protocols TLSv1.2 TLSv1.3`
	
	• 後方互換性が必要な場合は、次のコマンドを使用してそのバージョンを有効にします。
	
	`switch(config)# nxapi ssl protocols TLSv1.2`
	
	（注）
	• 下位互換性のために、TLSv1.2 および TLSv1.3 を使用することをお勧めします。
	
	`switch(config)# nxapi ssl protocols TLSv1.2 TLSv1.3`
	
	次の場合を例にします：
	
	• TLSv1.3 を設定する前に、TLSv1.3 をサポートするためにサーバーとクライアントの証明書を検証します。
	
	• NX-API サーバ側の SUDI 証明書は、TLSv1.3 ではサポートされていません。

NX-API 管理コマンド	説明
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。 （注） Cisco NX-OS リリース 7.0(3)I2(1) では、NGINX は 1 つの VRF でのみリッスンします。
system server session cmd-timeout <timeout>	Cisco NX-OS リリース 10.2(3)F 以降、NGINX サーバーでは、コマンドを実行するためのデフォルトのタイムアウトは5分です。ユーザは、必要に応じて、およびコマンドの実行にかかる時間に応じて、タイムアウトを 60 秒（1分）から3600秒（1時間）の任意の値に増やすことができます。
ip netns exec management iptables	アクセス制限を実装し、管理 VRF で実行できます。 （注） 機能 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ブロック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、ほとんどの場合、*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 が bash シェルで変更されたときに、リロード後も iptables を永続化する方法の詳細については、「」を参照してください。リロード間で Iptable を永続化する（221 ページ）
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、このコマンドの no 形式を使用します。 **no nxapi idle-timeout <timeout>**

次に、SUDI の NX-API 出力の例を示します。

```
switch(config)# nxapi certificate sudi
switch# show nxapi
nxapi enabled
NXAPI timeout 10
NXAPI cmd timeout 300
HTTP Listen on port 80
HTTPS Listen on port 443
Certificate Information:
    Issuer:   issuer=CN = High Assurance SUDI CA, O = Cisco
```

```
    Expires:  Aug  9 20:58:26 2099 GMT
switch#
switch#
switch# show run | sec nxapi
feature nxapi
nxapi http port 80
nxapi certificate sudi
switch#
```

次に、トラストポイントの設定例を示します。

```
switch(config)# crypto ca trustpoint ngx
switch(config-trustpoint)# crypto ca import ngx pkcs12 bootflash:server.pfx cisco123
witch(config)# nxapi certificate trustpoint ngx
switch(config)# show nxapi
nxapi enabled
NXAPI timeout 10
NXAPI cmd timeout 300
HTTP Listen on port 80
Trustpoint label ngx
HTTPS Listen on port 443
Certificate Information:
Issuer: issuer=C = IN, ST = KA, L = bang, O = cisco, OU = nxpi, CN = %username%@cisco.com,
 emailAddress = %username%@cisco.com
Expires: Jan 13 06:13:50 2023 GMT
switch(config)#
switch(config)# show run | sec nxapi
feature nxapi
nxapi http port 80
nxapi certificate trustpoint ngx
```

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、証明書とキーを設定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、filename 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。**nxapi certificatehttpskey keyfile password**

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 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。

表 *7: XML* または *JSON* 形式の *NX-API* 要求要素

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_array** 構造化された出力が必要な CLI **show** コマンド。show コマンド専用。**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** 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。

<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 > 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
ロールバック	コンフィギュレーション CLI に対してのみ有効であり、show コマンドに対しては有効ではありません。コンフィギュレーション ロールバック オプションを指定します。次のいずれかのオプションを指定します。 • Stop-on-error：最初に失敗した CLI で停止します。 • Continue-on-error：他の CLI を無視して続行します。 • Rollback-on-error：システム設定を以前の状態にロールバックします。 （注） 入力要求形式が XML または JSON の場合、ロールバック要素は cli_conf モードで使用できます。
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。

<table>
<tr><th>NX-API リクエスト要素</th><th>説明</th></tr>
<tr><td>input</td><td>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。）

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。
<table><tr><td>cli_show</td><td><code>show version ; show interface brief ; show vlan</code></td></tr><tr><td>cli_conf</td><td><code>interface Eth4/1 ; no shut ; switchport</code></td></tr><tr><td>bash</td><td><code>cd /bootflash;mkdir new_dir</code></td></tr></table></td></tr>
<tr><td>output_format</td><td>使用可能な出力メッセージ形式は次のとおりです。
<table><tr><td>xml</td><td>XML 形式を指定します。</td></tr><tr><td>json</td><td>JSON 形式で出力を指定します。</td></tr></table></td></tr>
</table>

JSON-RPC が入力リクエスト形式である場合、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。

表 *8 : JSON-RPC* 形式の *NX-API* 要求要素

NX-API リクエスト要素	説明
jsonrpc	JSON-RPC プロトコルのバージョンを指定する文字列。 バージョンは 2.0 であることが必要です。

NX-API リクエスト要素	説明
method	呼び出されるメソッドの名前を含む文字列。 NX-API は、次のいずれかをサポートします。 • **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 コマンド専用。**cli** に似ていますが、**cli_array** はデータを角括弧（[ ]）で囲まれた 1 つの要素のリスト、つまり配列として返します。
params	メソッドの呼び出し中に使用されるパラメータ値を保持する構造化された値。 以下が含まれている必要があります。 • **cmd**：CLI コマンド • **version**：NX-API リクエストのバージョン識別子
ロールバック	コンフィギュレーション CLI に対してのみ有効であり、show コマンドに対しては有効ではありません。構成ロールバック オプション次のいずれかのオプションを指定できます。 • Stop-on-error：最初に失敗した CLI で停止します。 • Continue-on-error：失敗した CLI を無視し、他の CLI を続行します。 • Rollback-on-error：システム設定を以前の状態にロールバックします。
検証	構成検証設定この要素を使用すると、スイッチに適用する前にコマンドを検証できます。これにより、設定を適用する前に、設定の整合性（必要なハードウェア リソースの可用性など）を確認できます。[検証タイプ（Validation Type）] ドロップダウン リストから検証タイプを選択します。 • Validate-Only：設定を検証しますが、設定は適用しません。 • Validate-and-Set：設定を検証し、検証が成功した場合はスイッチに設定を適用します。

NX-API リクエスト要素	説明
ロック	構成の排他ロックを指定できます。これにより、このロックが保持されている場合、他の管理エージェントまたはプログラミングエージェントは構成を変更できません。
id	クライアントによって確立されるオプションの識別子。指定されている場合は、文字列、数値、または null 値を含む必要があります。値は null にならないはずです。数値には小数部を含めません。ユーザーが id パラメータを指定しなかった場合、サーバーはリクエストが単なる通知であるとみなし、応答はしません。パラメータは*id*: 1 などのように指定します。

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 *9 : 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。
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。

NX-API 応答要素	説明
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

ACL は、VRF が構成されておらず、管理 VRF が NX- API 向けに構成されている場合にも適用されるようになりました。

デフォルトおよびカスタム VRF のアクセス制限は iptable を介して行なわれます。iptable 内では、VRF 名を指定することによって実行されます。

iptable の更新

iptable を使用すると、VRF が NX-API 通信用に設定されている場合に、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 への HTTP および HTTPS アクセスをブロックするルールを追加、確認、および削除する方法を示します。

手順

ステップ 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
```

（注）
この手順に記載されている **management** は VRF 名です。 **management | default | custom vrf name** を使用できます。

ステップ 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
```

ステップ 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
```

ステップ 4 ポート 80 への 10.155.0.0/24 サブネットを持つ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
```

ステップ 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、を参照してください。リロード間で Iptable を永続化する （221 ページ）

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。

手順

ステップ 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
```

ステップ 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
```

ステップ 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
```

ステップ 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
```

ステップ 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

次に、NX-API 応答の考えられる NX-API エラー、エラー コード、およびメッセージを示します。

（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 *10 : 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	入力 Bash コマンド エラー。
CHUNK_ALLOW_ONE_CMD_ERR	400	チャンクは1つのコマンドにのみ許可されます。
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	要求メッセージが無効です。
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。
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。
LIBXML_PARSE_ERR	500	システムの内部 LIBXML 解析エラー。
LIBXML_PATH_CTX_ERR	500	システムの内部 LIBXML パス コンテキスト エラー。
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_OUTPUT_NOT_SUPPORTED_ERR	501	パイプ操作はサポートされていません。
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	入力でパイプ XML 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。
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	応答の出力量が多い。JSON はサポートされません。
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	未定義。

XML および JSON でサポートされたコマンド

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 フォーマット出力の標準規格ブロックを読みやすくする JSON Pretty
- 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。

標準規格の NX-OS 出力を JSON、JSON Pretty、または XML フォーマットに変換することは、出力を JSON または XML インタープリターに「パイプ」することによって、NX-OS CLI で発生します。たとえば、論理パイプ (|) を使用して **show ip access** コマンドを発行し、JSON、JSON Pretty、JSON Native、JSON Native Pretty、または XML を指定すると、NX-OS コマンド出力が適切に構造化され、そのフォーマット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、および XML 出力をサポートしています。

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。JSON プリティ形式、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。

JSON/CLI 実行は現在、N3500 で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 /XML 出力には、サンドボックス経由でアクセスすることもできます。

CLI の実行

```
BLR-VXLAN-NPT-CR-179# show cdp neighbors  | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "
83886080", "device_id": "SW-SPARSHA-SAVBU-F10", "intf_id": "mgmt0", "ttl": "148"
, "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco WS-C2960
S-48TS-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device
_id": "BLR-VXLAN-NPT-CR-178(FOC1745R01W)", "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

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例を示します。

```
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例を示します。

```
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、ACL 統計を XML 形式で表示する方法を示しています。

```
switch-1(config-acl)# show ip access-lists acl-test1 | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:1.0:aclmgr" xmlns:nf="urn:ietf:p
arams:xml:ns:netconf:base:1.0">
 <nf:data>
  <show>
   <__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
    <ip_ipv6_mac>ip</ip_ipv6_mac>
    <access-lists>
     <__XML__OPT_Cmd_show_acl_name>
      <name>acl-test1</name>
```

```
      <__XML__OPT_Cmd_show_acl_capture>
       <__XML__OPT_Cmd_show_acl_expanded>
        <__XML__OPT_Cmd_show_acl___readonly__>
         <__readonly__>
          <TABLE_ip_ipv6_mac>
           <ROW_ip_ipv6_mac>
            <op_ip_ipv6_mac>ip</op_ip_ipv6_mac>
            <show_summary>0</show_summary>
            <acl_name>acl-test1</acl_name>
            <statistics>enable</statistics>
            <frag_opt_permit_deny>permit-all</frag_opt_permit_deny>
            <TABLE_seqno>
             <ROW_seqno>
              <seqno>10</seqno>
              <permitdeny>permit</permitdeny>
              <ip>ip</ip>
              <src_ip_prefix>192.0.2.1/24</src_ip_prefix>
              <dest_any>any</dest_any>
             </ROW_seqno>
            </TABLE_seqno>
           </ROW_ip_ipv6_mac>
          </TABLE_ip_ipv6_mac>
         </__readonly__>
        </__XML__OPT_Cmd_show_acl___readonly__>
       </__XML__OPT_Cmd_show_acl_expanded>
      </__XML__OPT_Cmd_show_acl_capture>
     </__XML__OPT_Cmd_show_acl_name>
    </access-lists>
   </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
switch-1(config-acl)#
```

この例は、ACL 統計を JSON 形式で表示する方法を示しています。

```
switch-1(config-acl)# show ip access-lists acl-test1 | json
{"TABLE_ip_ipv6_mac": {"ROW_ip_ipv6_mac": {"op_ip_ipv6_mac": "ip", "show_summar
y": "0", "acl_name": "acl-test1", "statistics": "enable", "frag_opt_permit_deny
": "permit-all", "TABLE_seqno": {"ROW_seqno": {"seqno": "10", "permitdeny": "pe
rmit", "ip": "ip", "src_ip_prefix": "192.0.2.1/24", "dest_any": "any"}}}}}
switch-1(config-acl)#
```

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、スイッチの冗長ステータスを JSON 形式で表示する方法を示しています。

```
switch-1# show system redundancy status | json
{"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"}
nxosv2#
switch-1#
```

次に、XML 形式で IP ルート要約を表示する例を示します。

```
switch-1# show ip route summary | xml
<?xml version="1.0" encoding="ISO-8859-1"?> <nf:rpc-reply
xmlns="http://www.cisco.com/nxos:1.0:urib"
xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
 <nf:data>
  <show>
   <ip>
    <route>
     <__XML__OPT_Cmd_urib_show_ip_route_command_ip>
      <__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
       <__XML__OPT_Cmd_urib_show_ip_route_command_topology>
        <__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
         <__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
          <__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
           <__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
            <__XML__OPT_Cmd_urib_show_ip_route_command_summary>
             <__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
              <__XML__OPT_Cmd_urib_show_ip_route_command___readonly__>
               <__readonly__>
                <TABLE_vrf>
                 <ROW_vrf>
                  <vrf-name-out>default</vrf-name-out>
                  <TABLE_addrf>
                   <ROW_addrf>
                    <addrf>ipv4</addrf>
                    <TABLE_summary>
                     <ROW_summary>
                      <routes>938</routes>
                      <paths>1453</paths>
                      <TABLE_unicast>
                       <ROW_unicast>
                        <clientnameuni>am</clientnameuni>
                        <best-paths>2</best-paths>
                       </ROW_unicast>
                       <ROW_unicast>
                        <clientnameuni>local</clientnameuni>
                        <best-paths>105</best-paths>
                       </ROW_unicast>
                       <ROW_unicast>
                        <clientnameuni>direct</clientnameuni>
                        <best-paths>105</best-paths>
                       </ROW_unicast>
                       <ROW_unicast>
                        <clientnameuni>broadcast</clientnameuni>
                        <best-paths>203</best-paths>
                       </ROW_unicast>
                       <ROW_unicast>
                        <clientnameuni>ospf-10</clientnameuni>
                        <best-paths>1038</best-paths>
                       </ROW_unicast>
                      </TABLE_unicast>
                      <TABLE_route_count>
                       <ROW_route_count>
                        <mask_len>8</mask_len>
                        <count>1</count>
                       </ROW_route_count>
                       <ROW_route_count>
                        <mask_len>24</mask_len>
                        <count>600</count>
```

```
                        </ROW_route_count>
                        <ROW_route_count>
                         <mask_len>31</mask_len>
                         <count>13</count>
                        </ROW_route_count>
                        <ROW_route_count>
                         <mask_len>32</mask_len>
                         <count>324</count>
                        </ROW_route_count>
                       </TABLE_route_count>
                      </ROW_summary>
                     </TABLE_summary>
                    </ROW_addrf>
                   </TABLE_addrf>
                  </ROW_vrf>
                 </TABLE_vrf>
                </__readonly__>
               </__XML__OPT_Cmd_urib_show_ip_route_command___readonly__>
              </__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
             </__XML__OPT_Cmd_urib_show_ip_route_command_summary>
            </__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
           </__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
          </__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
         </__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
        </__XML__OPT_Cmd_urib_show_ip_route_command_topology>
       </__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
      </__XML__OPT_Cmd_urib_show_ip_route_command_ip>
     </route>
    </ip>
   </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
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#
```

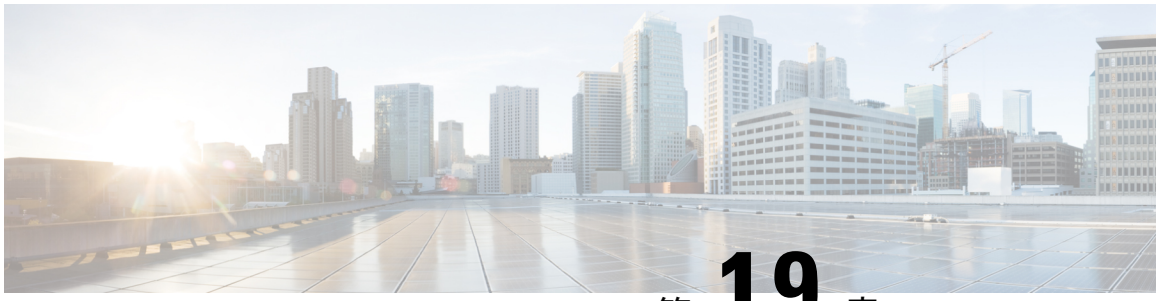
次の例は、JSON 形式で IP ルート要約を表示する例を示します。

```
switch-1# show ip route summary | json
{"TABLE_vrf": {"ROW_vrf": {"vrf-name-out": "default", "TABLE_addrf": {"ROW_addrf":
{"addrf": "ipv4", "TABLE_summary": {"ROW_summary": {"routes": "938", "paths": "
1453", "TABLE_unicast": {"ROW_unicast": [{"clientnameuni": "am", "best-paths": "2"},
{"clientnameuni": "local", "best-paths": "105"}, {"clientnameuni": "direct",
"best-paths": "105"}, {"clientnameuni": "broadcast", "best-paths": "203"},
{"clientnameuni": "ospf-10", "best-paths": "1038"}]}, "TABLE_route_count": {"ROW_route_
count": [{"mask_len": "8", "count": "1"}, {"mask_len": "24", "count": "600"}, {"mask_len":
 "31", "count": "13"}, {"mask_len": "32", "count": "324"}]}}}}}}}}
switch-1#
```

次の例は、JSON Pretty 形式で IP ルート要約を表示する例を示します。

```
switch-1# show ip route summary | json-pretty
  {
      "TABLE_vrf": {
        "ROW_vrf": {
            "vrf-name-out": "default",
            "TABLE_addrf": {
                "ROW_addrf": {
                    "addrf": "ipv4",
                    "TABLE_summary": {
                        "ROW_summary": {
                            "routes": "938",
                            "paths": "1453",
                            "TABLE_unicast": {
                                "ROW_unicast": [
```

```
                                        {
                                            "clientnameuni": "am",
                                            "best-paths": "2"
                                        },
                                        {
                                            "clientnameuni": "local",
                                            "best-paths": "105"
                                        },
                                        {
                                            "clientnameuni": "direct",
                                            "best-paths": "105"
                                        },
                                        {
                                            "clientnameuni": "broadcast",
                                            "best-paths": "203"
                                        },
                                        {
                                            "clientnameuni": "ospf-10",
                                            "best-paths": "1038"
                                        }
                                    ]
                                },
                                "TABLE_route_count": {
                                    "ROW_route_count": [
                                        {
                                            "mask_len": "8",
                                            "count": "1"
                                        },
                                        {
                                            "mask_len": "24",
                                            "count": "600"
                                        },
                                        {
                                            "mask_len": "31",
                                            "count": "13"
                                        },
                                        {
                                            "mask_len": "32",
                                            "count": "324"
                                        }
                                    ]
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
switch-1#
```



第 19 章

NX-API REST

この章は、次の項で構成されています。

NX-API REST について

NX-API REST

リリース 7.0(3)F3(1) 以降、NX- API REST SDK が使用可能になりました。

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、スイッチ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。
- 比較的少数の HTTP/HTTPS 操作で設定アクションを組み合わせることで、多数の CLI コマンドを発行する必要がある設定を有効にします。

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。Nginx の技術情報使用量が cgroup の制限を超えると、Nginx プロセスが再起動されて復元されます。

Cisco NX-API REST SDK の詳細については、https://developer.cisco.com/site/nx-api/documents/n3k-n9k-api-ref/ を参照してください。

REST による DME 設定の置換

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 コンフィギュレーションの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コンフィギュレーションを置換できます。この機能には、次の追加の利点があります。DME の完全な設定置換は、PUT 操作によって行われます。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）は、DME の完全な設定の置換をサポートします。

- スイッチ設定の無停止交換をサポート
- 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。
- 最終的な設定結果を指定できるようにすることで、設定変更を簡素化し、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 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、Cisco NX-OS CLI コマンドを使用して完全な設定置換を実行することもできます。**config replace config-file-name**

注意事項と制約事項

以下は DME フル構成置換機能の注意事項と制約事項です。

- ツリーを確認し、構成置換が適用される場所を確認することが重要です。構成置換操作にサンドボックスを使用する場合、サンドボックスはデフォルトでサブツリーに設定されるため、構成ツリー内の正しいノードをターゲットとするように URI を変更する必要があります。
- NX-OS サンドボックスを使用して（置換のための）変換を行う場合、要求に `status: 'replaced'` 属性が存在するため、POST 操作を使用する必要があります。他の変換オプションを使用する場合は、PUT 操作を使用できます。
- サブツリーノードでこの機能の REST PUT オプションを使用すると、構成置換操作がサブツリー全体に適用されます。ターゲットのサブツリー ノードは PUT の構成置換データで正しく変更されますが、サブツリー ノードのリーフ ノードもデフォルト値に構成されることによって影響を受けることに注意してください。

リーフ ノードに影響が及ばないようにする必要がある場合は、PUT 操作を使用しないでください。代わりに、`status:'replaced'` 属性を指定した POST 操作を使用できます。

リーフ ノードに構成置換を適用する場合、PUT 操作は期待どおりに動作します。

REST PUT によるシステムレベル構成の置換

管理クライアントから REST PUT を送信することで、スイッチの構成全体を置き換えることができます。

次の手順を使用します。

手順

ステップ 1 クライアントから、URL を `/api/mo/sys.json`として、ペイロードをシステムレベルとして REST PUT 操作を発行します。

ペイロードは有効な構成である必要があり、構成は `/api/mo/sys.json?rsp-subtree=full&rsp-prop-include=set-config-only` で GET を発行して、いつでもスイッチから取得可能である必要があります。

ステップ 2 `/api/mo/sys.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 に GET を送信します。

ステップ 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 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設定の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、特定の機能の設定を置き換えることができます。

次の手順を使用します。

手順

ステップ 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 /api/mo/sys/bgp.json の場合

ペイロードは有効な設定である必要があり、機能の DN で GET を発行することで、いつでもスイッチから設定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 です。

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"
                    }
                  }
                }
              ]
            }
          }
```

ステップ 2 /api/mo/sys/bgp.json?rsp-subtree=full&rsp-prop-include=set-config-only を使用して、設定の置換に使用した DN で GET を送信します。

ステップ 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 POST によるプロパティレベルの構成の置換

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（241 ページ） を参照してください。

手順

ステップ 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。

ステップ 2 作業エリアで、変更する機能の CLI を入力します。

ステップ 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。

ステップ 4 [方法（Method）] で、`NX-API (DME)` を選択します。

ステップ 5 [入力タイプ（Input Type）] で、[CLI] を選択します。

ステップ 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。

ステップ 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 の構成置換のトラブルシューティング

以下は、REST Put 操作による設定の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。

手順

ステップ 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。

ステップ 2 ペイロードが有効であり、次の構成プロパティのみが含まれていることを確認します。

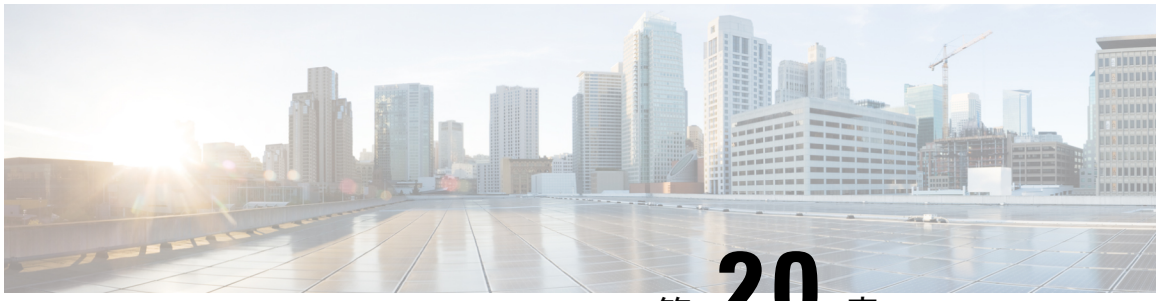
- 正常に設定されました
- 有効なデバイス設定から取得

設定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。

ステップ 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。

ステップ 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。

ステップ 5 ペイロードに MO の名前とプロパティも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

この章は、次の項で構成されています。

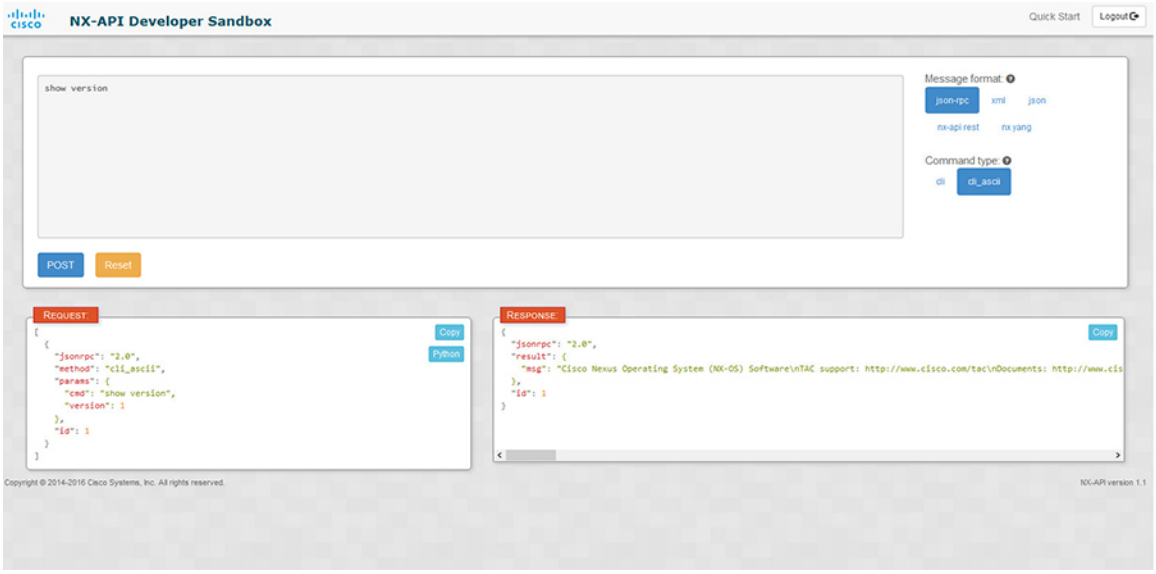
NX-API 開発者サンドボックス: 9.2（2）より前の NX-OS リリース

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 ペイロードに変換し、NX-API REST ペイロードを同等の CLI に変換します。

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。

逆に、コマンド ペインに NX-API REST 指定名 (DN) とペイロードを入力し、**`nx-api rest`** メッセージ フォーマットと **[モデル（`model`）]** コマンド タイプを選択すると、デベロッパー サンドボックスはペイロードの構成エラーをチェックし、応答ペインに同等の CLI が表示されます。

注意事項と制約事項

デベロッパー サンドボックスのガイドラインと制限は次のとおりです：

- サンドボックスで **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。
- 一部の機能構成コマンドは、関連する機能が有効になるまで使用できません。

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 *11 : 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。詳細については、[Cisco Nexus NX-API リファレンス（Cisco Nexus NX-API References）]を参照してください。
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 *12 :* 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

メッセージ形式	コマンド タイプ
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。
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 • モデル — DN および対応するペイロード。
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。

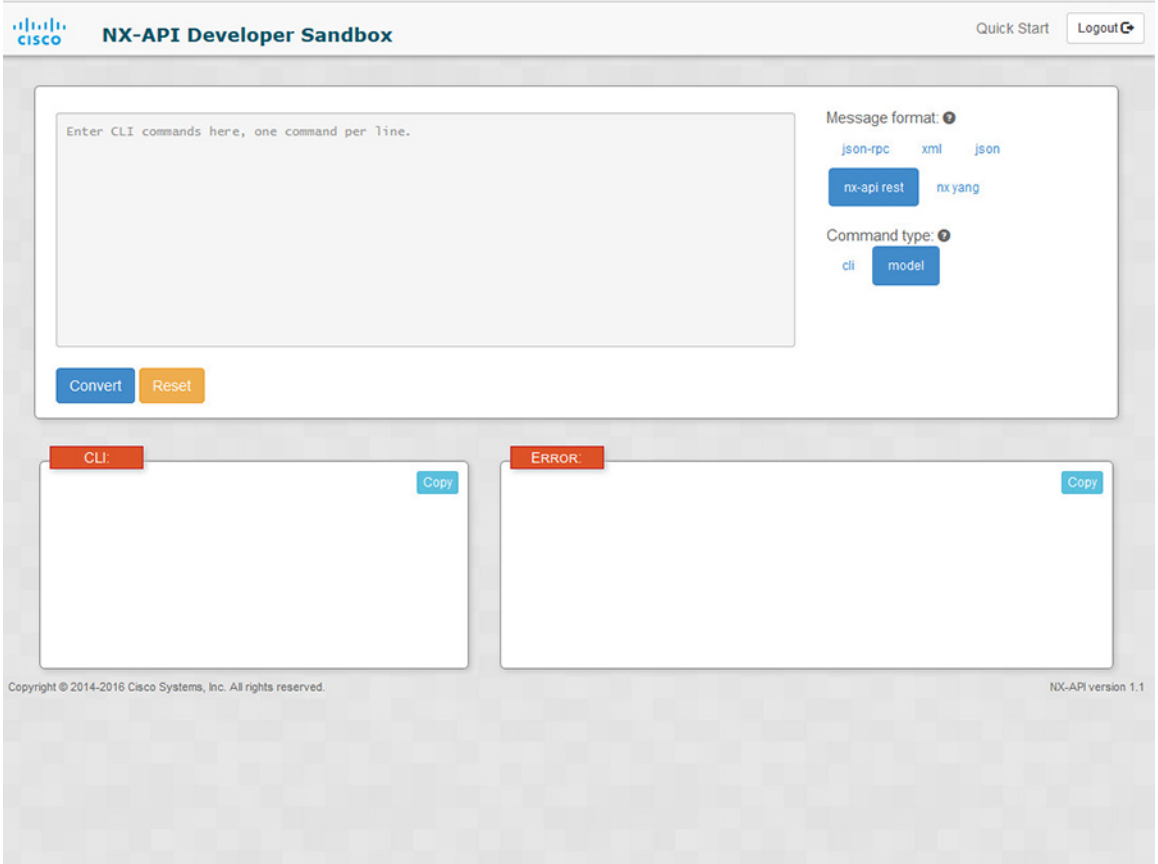
手順

ステップ 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（242 ページ）を参照してください。

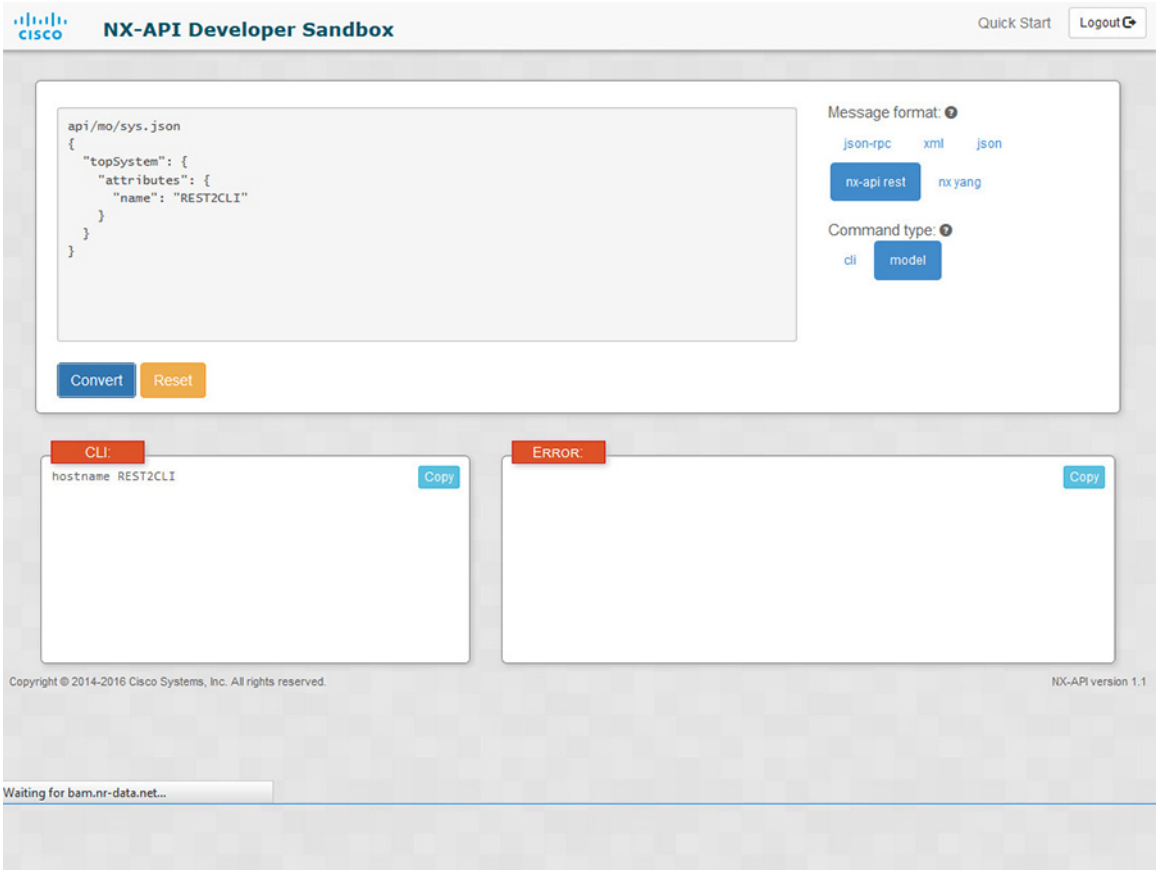
ステップ 2　上部ペインのテキスト エントリ ボックスに、NX-OS CLI 構成コマンドを 1 行に 1 つずつ入力するか貼り付けます。

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。



ステップ 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。

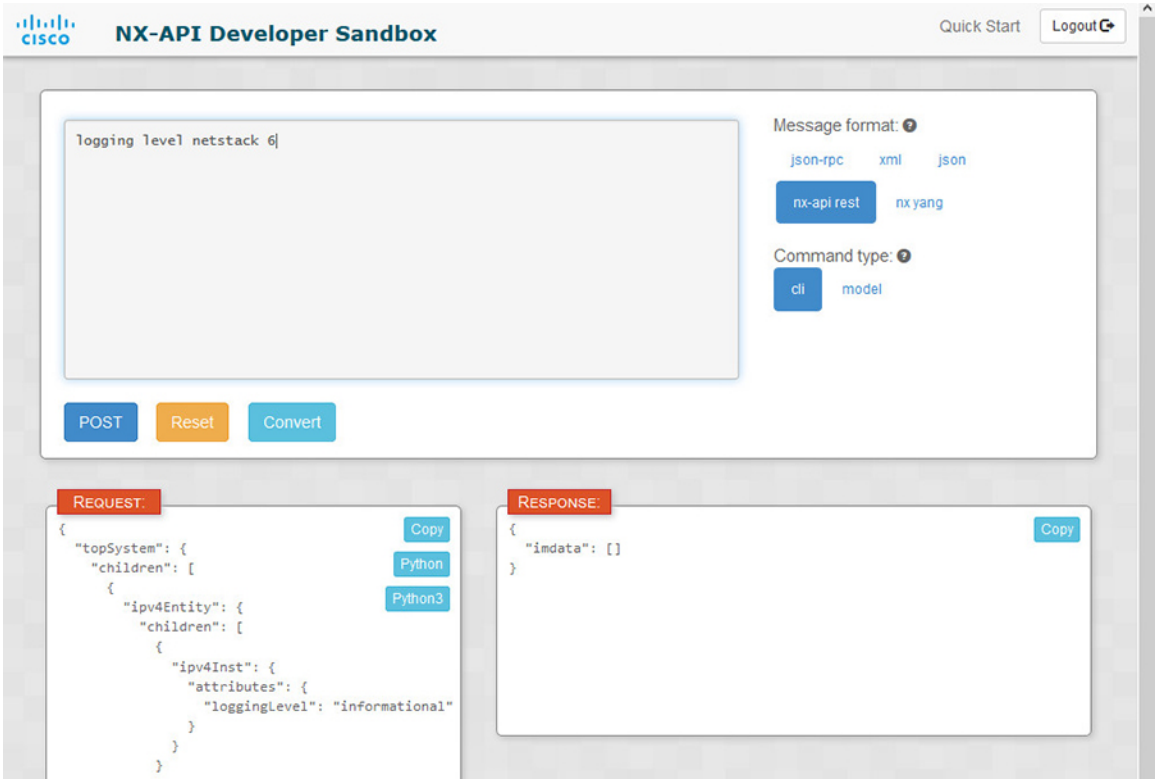
CLI コマンドに構成エラーが含まれていない場合、ペイロードは **[要求（Request）]** ペインに表示されます。エラーが存在する場合は、説明のエラー メッセージが **[応答（Response）]** ペインに表示されます。



ステップ 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。



ステップ 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。

ステップ 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。

NX-API 開発者サンドボックス：NX-OS リリース 9.2（2）以降

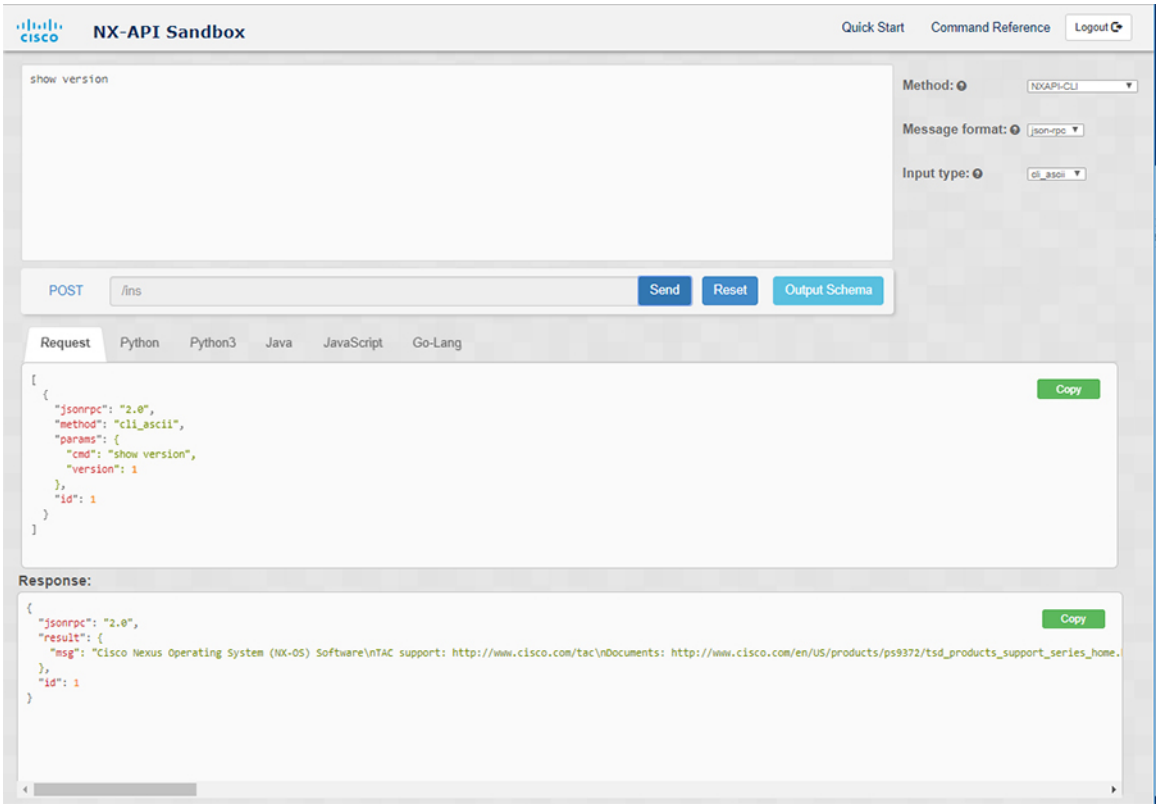
About the NX-API デベロッパー サンドボックス

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 （256 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 （259 ページ）」を参照してください。

注意事項と制約事項

デベロッパー サンドボックスのガイドラインと制限は次のとおりです：

- サンドボックスで **[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。
- 一部の機能構成コマンドは、関連する機能が有効になるまで使用できません。たとえば、BGP ルータを構成するには、最初に **[機能 bgp（feature bgp）]** コマンドを使用して BGP を有効にする必要があります。同様に、OSPF ルータを構成するには、最初に **[機能 ospf（feature ospf）]** コマンドを使用して OSPF を有効にする必要があります。これは、**[evpn マルチホーミング コア トラッキング（evpn multihoming core-tracking）]** などの依存コマンドを有効にする **[evpn esi マルチホーミング（evpn esi multihoming）]** にも適用されます。機能が機能依存コマンドにアクセスできるようにする方法の詳細については、Cisco Nexus 9000 Configuration Guides Cisco Nexus 3000 Configuration Guides を参照してください。
- サンドボックスを使用した DN での変換は、CLI 構成の DN を検索する場合にのみサポートされます。DME を使用して CLI 構成コマンドの DN を変換するなど、他のワークフローはサポートされていません。
- OSPFv2 インターフェイス コマンドのための、CLI からモデルまたは xml への変換は、**[no] ip router ospf <tag> area {<area-id-ip> | <area-id-int>} [secondaries none]** コマンドを使用してルータ インスタンスとエリアを構成し、インターフェイスで OSPF を明示的に有効にするまでは行われません。
- コマンド ペイン（上部のペイン）は、最大 10,000 行の入力をサポートします。
- CLI 入力のメッセージ タイプとして XML または JSON を使用する場合、セミコロンを使用して同じ行の複数のコマンドを区切ることができます。ただし、CLI 入力のメッセージ タイプとして JSON RPC を使用する場合、同じ行に複数のコマンドを入力し、セミコロン (;) で区切ることはできません。

  たとえば、次のように JSON RPC を介して**show hostname** コマンドと**show clock** コマンドをを送信したいとします。

  Sandbox で、次のように CLI を入力します。

  ```
  show hostname ; show clock
  ```

  JSON RPC リクエストでは、入力は次のようにフォーマットされます。

  ```
  [
    {
      "jsonrpc": "2.0",
      "method": "cli",
      "params": {
        "cmd": "show hostname ; show clock",
        "version": 1
      },
      "id": 1
    }
  ]
  ```

  リクエストを送信すると、レスポンスで次のエラーが返されます。

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32602,
    "message": "Invalid params",
    "data": {
      "msg": "Request contains invalid special characters"
    }
  },
  "id": 1
}
```

この状況は、Sandbox が JSON RPC リクエストの各コマンドを個別のアイテムとして解析し、それぞれに 識別子 を割り当てるために発生します。JSON RPC リクエストを使用する場合、同じ回線で複数のコマンドを区切るために内部句読点を使用することはできません。代わりに、各コマンドを個別の回線に入力すると、リクエストが正常に完了します。

同じ例を続けて、NX-API CLI で次のようにコマンドを入力します。

```
show hostname
show clock
```

リクエストでは、入力は次のようにフォーマットされます。

```
[
  {
    "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
  }
]
```

応答は正常に完了します。

```
[
  {
    "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 *13 : 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。
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 *14*: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 *15*：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

Cisco NX-API デベロッパー サンドボックス を使用して、次のような複数の変換を行うことができます。

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。

手順

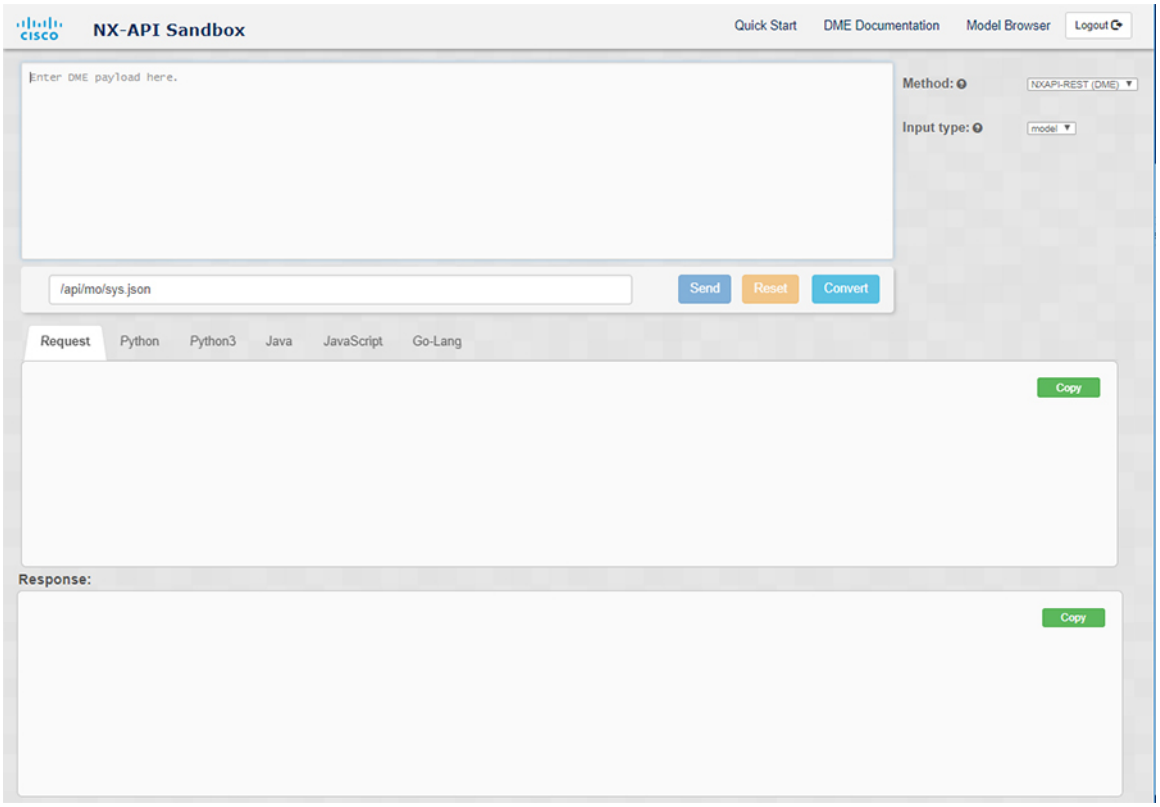
ステップ 1 **[方法（Method）]** ドロップダウン リストをクリックし、**NXAPI-REST (DME)** を選択します。

[入力タイプ] ドロップダウン リストが表示されます。

ステップ 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。

ステップ 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。



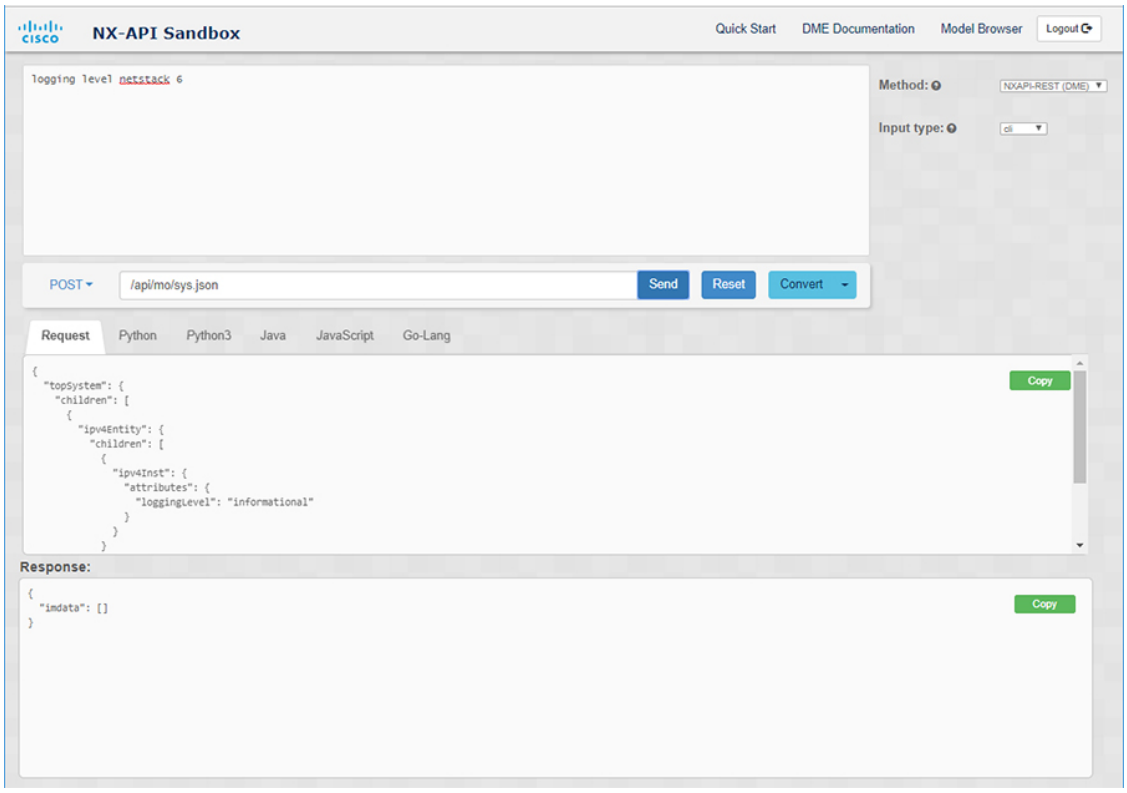
ステップ 4 [変換（Convert）] をクリックします。

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。

ステップ 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。



ステップ 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。

ステップ 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。

ステップ 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。

ステップ 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：

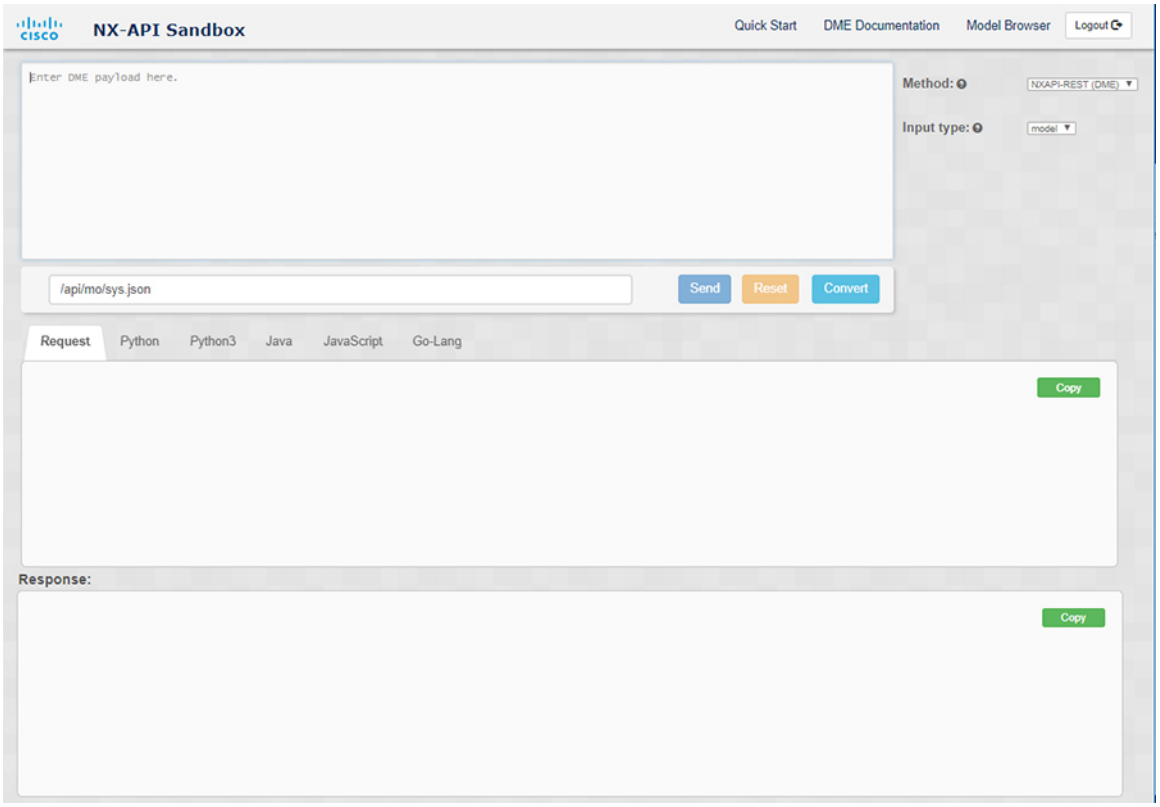
  - **[NX-API リファレンス（NX-API References）]**— 追加の NX-API ドキュメントにアクセスできます。
  - **[DME ドキュメント（DME Documentation）]** — NX-API DME モデル リファレンス ページにアクセスできます。
  - **[モデル ブラウザ（Model Browser）]** — モデル ブラウザである Visore にアクセスできます。Visore ページにアクセスするには、スイッチの IP アドレスを手動で入力する必要がある場合があることに注意してください。

    `https://`*`management-ip-address`*`/visore.html`。

手順

ステップ 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：



ステップ 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。

ステップ 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。

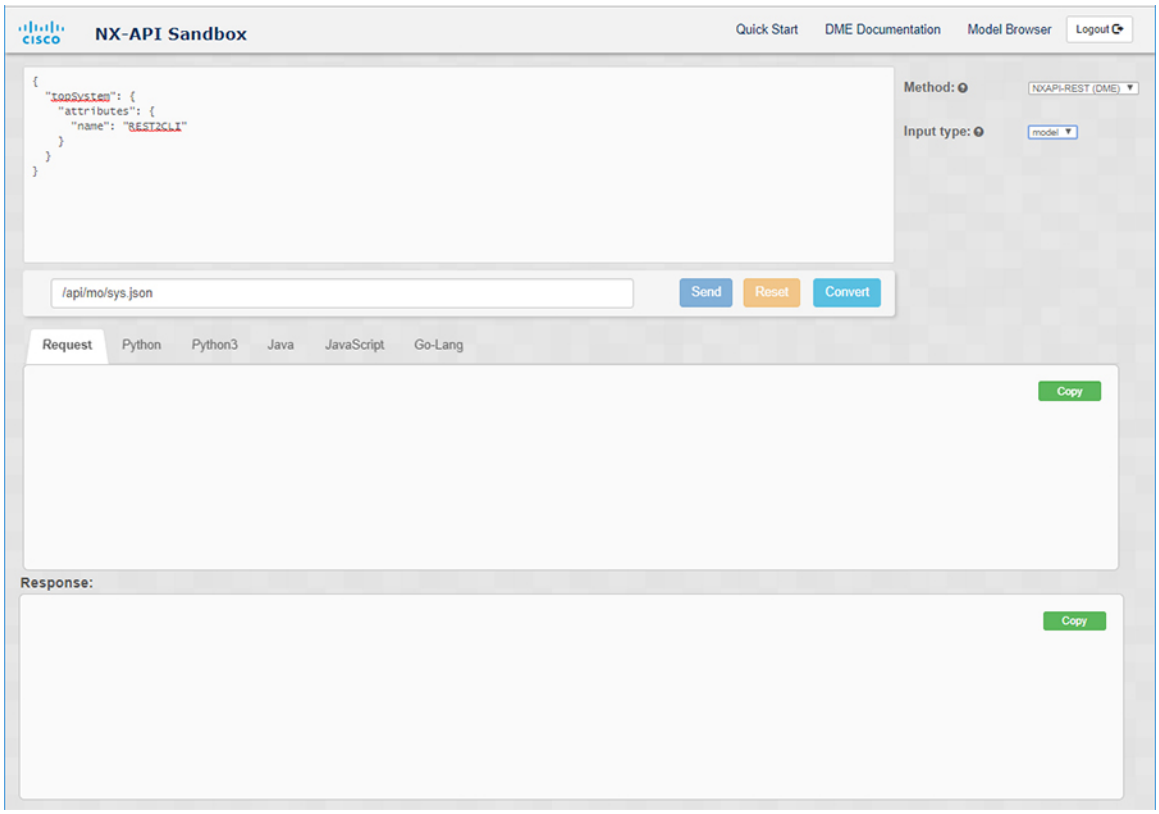
ステップ 4 コマンド ペインにペイロードを入力します。

ステップ 5 [変換（Convert）] をクリックします。

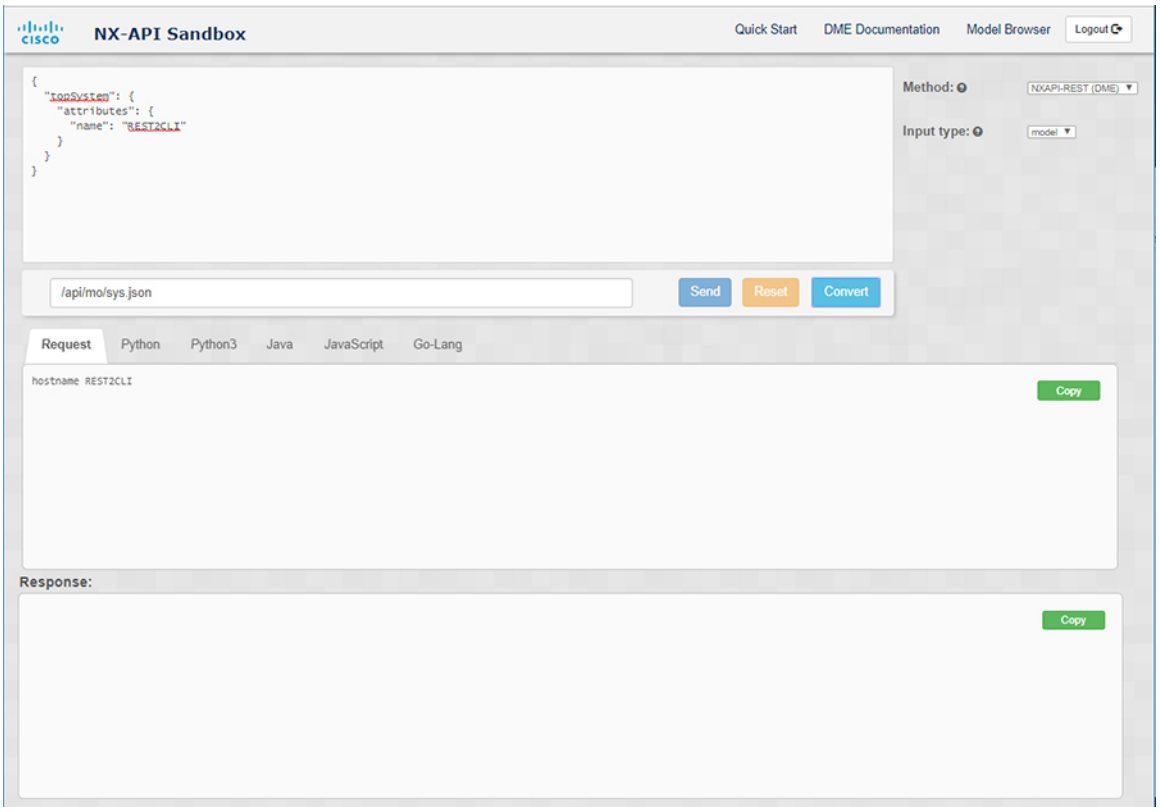
例：

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。



（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 *16 : 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.）]

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

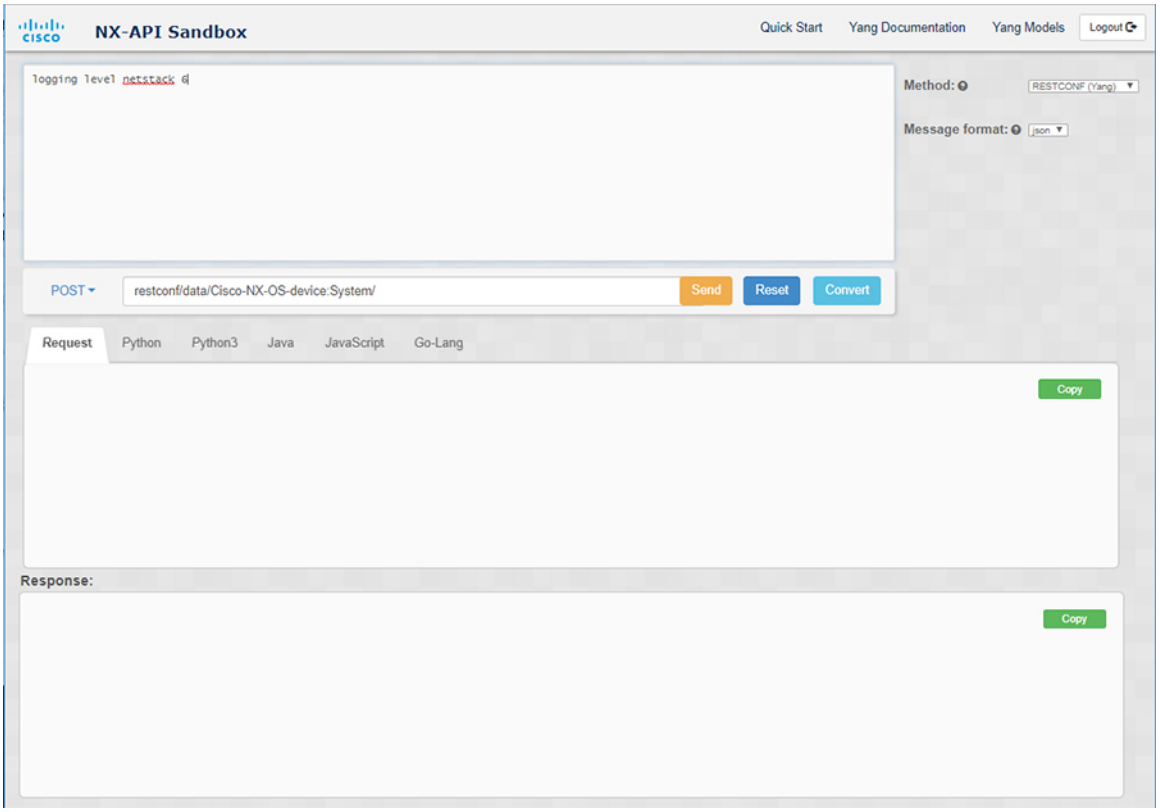
ヒント

- 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。

手順

ステップ 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：



ステップ 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。

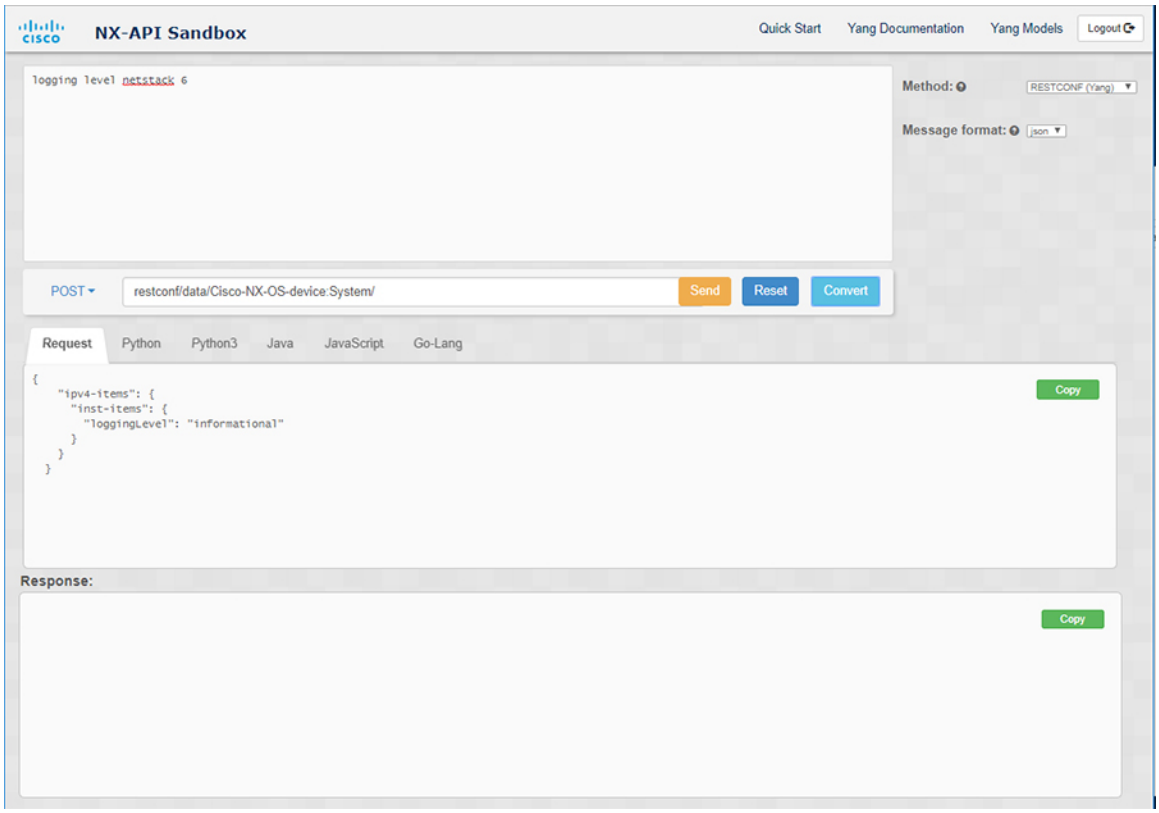
ステップ 3 上部ペインのテキスト入力ボックスにコマンドを入力します。

ステップ 4 メッセージ形式を選択します。

ステップ 5 [変換（Convert）] をクリックします。

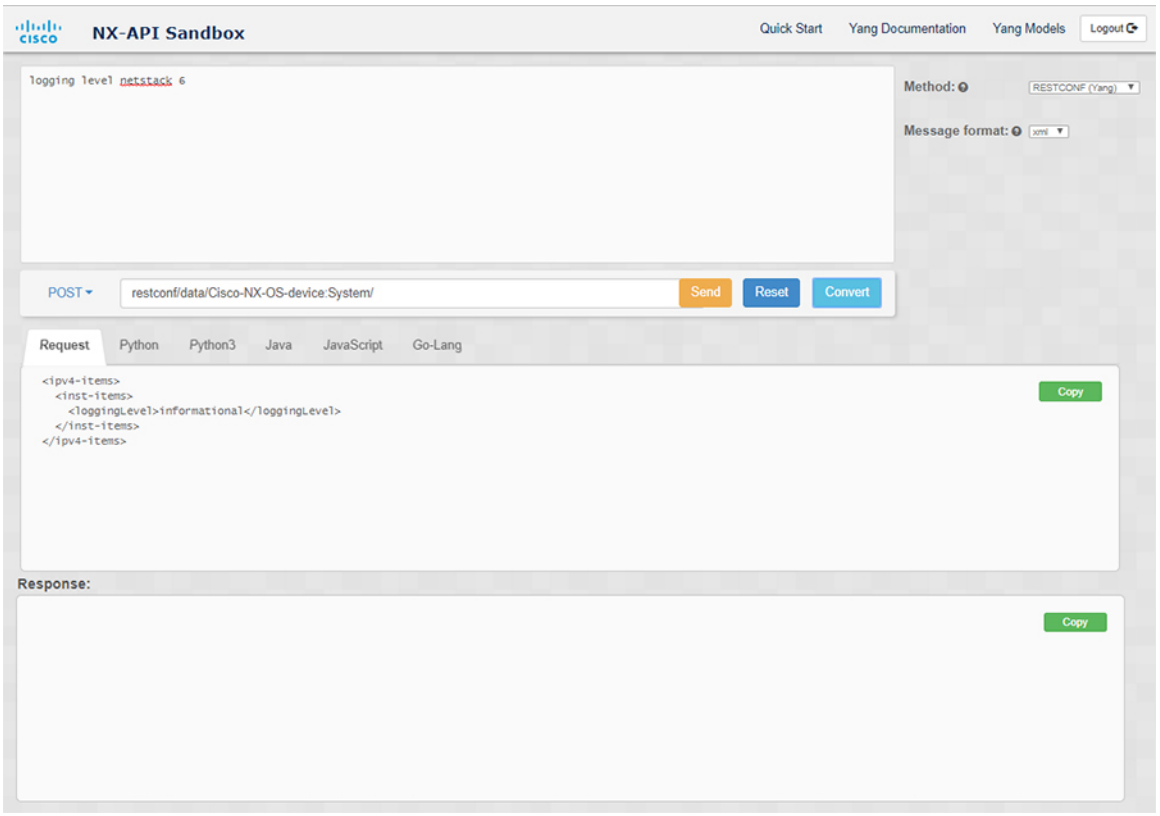
例：

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!」

ステップ 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。

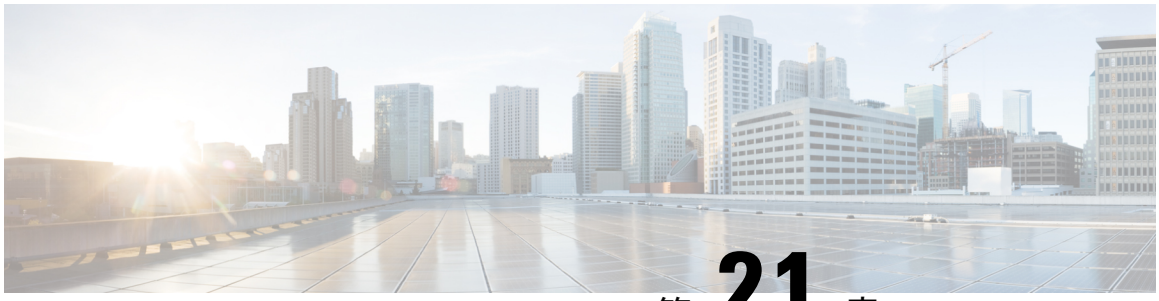
- Python
- python3
- Java
- JavaScript
- Go-Lang

（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 V 部

モデル駆動型プログラマビリティ



第 21 章

CLI コマンドのネットワーク構成フォーマットへの変換

XMLIN に関する情報

XMLIN ツールは、CLI コマンドをネットワーク構成（NETCONF）プロトコル形式に変換します。NETCONF は、ネットワーク デバイスの構成をインストール、処理、削除する機能を提供するネットワーク管理プロトコルです。これは、構成データとプロトコル メッセージに XML ベースのエンコーディングを使用します。NETCONF プロトコルの NX-OS 実装は、<get>、<edit-config>、<close-session>、<kill-session>、および <exec-command> のプロトコル操作をサポートします。

XMLIN ツールは、show、EXEC、および構成コマンドを対応する NETCONF <get>、<exec-command>、および<edit-config> リクエストに変換します。複数の構成コマンドを単一の NETCONF <edit-config> インスタンスにまとめることができます。

XMLIN ツールはまた、show コマンドの出力を XML 形式に変換します。

XMLIN のライセンス要件

表 *17 : XMLIN* ライセンス要件

製品	ライセンス要件

Cisco NX-OS	XMLIN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すべて Cisco NX-OS システム イメージにバンドルされており、追加費用は一切発生しません。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。

XMLIN ツールのインストールおよび使用

XMLIN ツールをインストールして、構成 コマンドを NETCONF フォーマットに変換するために使用できます。

始める前に

XMLIN ツールは、対応する機能セットまたは必要なハードウェア機能がデバイスで利用できない場合でも、コマンドの NETCONF インスタンスを生成できます。ただし、**xmlin** コマンドを入力する前に、いくつかの機能セットをインストールする必要がある場合があります。

手順の概要

1. switch# **xmlin**
2. switch(xmlin)# **configure terminal**
3. コンフィギュレーション コマンド
4. （任意） switch(config)(xmlin)# **end**
5. （任意） switch(config-if-verify)(xmlin)# **show** *commands*
6. （任意） switch(config-if-verify)(xmlin)# **exit**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	switch# **xmlin**	
ステップ 2	switch(xmlin)# **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
ステップ 3	コンフィギュレーション コマンド	構成コマンドをNETCONFフォーマットに変換します。
ステップ 4	（任意） switch(config)(xmlin)# **end**	対応する <edit-config> 要求を生成します。 （注） **show** コマンドに対してXMLインスタンスを生成する前に、**end** コマンドを入力して現在の XML 構成を終了する必要があります。

	コマンドまたはアクション	目的
ステップ 5	（任意） switch(config-if-verify)(xmlin)# **show** *commands*	**show** コマンドを NETCONF フォーマットに変換します。
ステップ 6	（任意） switch(config-if-verify)(xmlin)# **exit**	EXEC モードに戻ります。

show コマンド出力の XML への変換

show コマンドの出力を XML に変換できます。

始める前に

変換するコマンドのすべての機能がインストールされ、デバイス上で有効になっていることを確認します。そうしない場合、コマンドは機能不全になります。

terminal verify-only コマンドを使用すると、デバイスに入力しなくても機能が有効になっていることを確認できます。

コマンドに対するすべての必須ハードウェアがデバイス上に存在することを確認します。そうしない場合、コマンドは機能不全になります。

XMLIN ツールがインストールされていることを確認します。

手順の概要

1. switch# *show-command* | **xmlin**

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	switch# *show-command*	**xmlin**	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 （注） 構成 コマンドと一緒にこのコマンドを使用することはできません。

XMLIN の構成例

次の例は、XMLIN ツールがデバイス上にどのようにインストールされ、一連の構成コマンドを <edit-config> インスタンスに変換するためにどのように使用されるかを示しています。

```
switch# xmlin
**************************************
Loading the xmlin tool. Please be patient.
**************************************
Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright ©) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php

switch(xmlin)# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)(xmlin)# interface ethernet 2/1
% Success
switch(config-if-verify)(xmlin)# cdp enable
% Success
switch(config-if-verify)(xmlin)# end
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:configure_"
xmlns:m="http://www.cisco.com/nxos:6.2.2.:_exec"
xmlns:m1="http://www.cisco.com/nxos:6.2.2.:configure__if-eth-base" message-id="1">
  <nf:edit-config>
     <nf:target>
      <nf:running/>
  </nf:target>
  <nf:config>
    <m:configure>
      <m:terminal>
        <interface>
           <__XML__PARAM__interface>
              <__XML__value>Ethernet2/1</__XML__value>
              <m1:cdp>
                <m1:enable/>
              </m1:cdp>
            </__XML__PARAM__interface>
          </interface>
         </m:terminal>
        </m:configure>
       </nf:config>
      </nf:edit-config>
     </nf:rpc>
    ]]>]]>
```

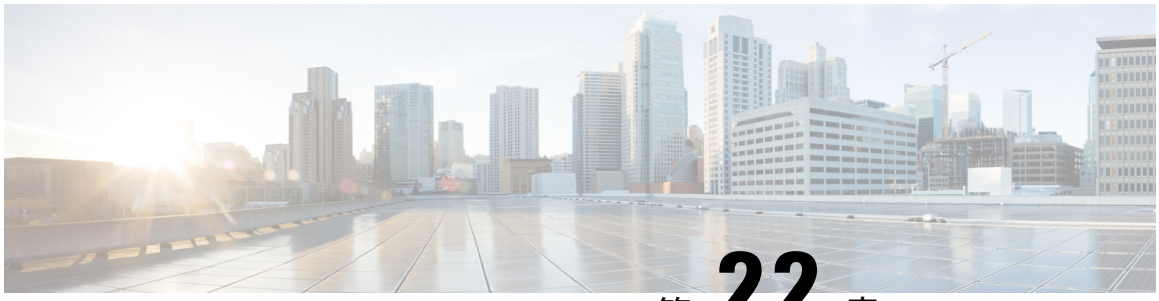
次の例は、**show** コマンドに対して XML インスタンスを生成する前に現在の XML 構成を終了するために **end** コマンドを入力する方法を表示しています。

```
switch(xmlin)# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)(xmlin)# interface ethernet 2/1
switch(config-if-verify)(xmlin)# show interface ethernet 2/1
******************************************************
Please type "end" to finish and output the current XML document before building a new
one.
******************************************************
% Command not successful
```

```
switch(config-if-verify)(xmlin)# end
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:configure_"
xmlns:m="http://www.cisco.com/nxos:6.2.2.:_exec" message-id="1">
    <nf:edit-config>
      <nf:target>
         <nf:running/>
      </nf:target>
      <nf:config>
         <m:configure>
           <m:terminal>
              <interface>
                 <__XML__PARAM__interface>
                    <__XML__value>Ethernet2/1</__XML__value>
                 </__XML__PARAM__interface>
              </interface>
            </m:terminal>
           </m:configure>
         </nf:config>
      </nf:edit-config>
    </nf:rpc>
   ]]>]]>

switch(xmlin)# show interface ethernet 2/1
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:if_manager" message-id="1">
  <nf:get>
    <nf:filter type="subtree">
      <show>
      <interface>
        <__XML__PARAM__ifeth>
          <__XML__value>Ethernet2/1</__XML__value>
        </__XML__PARAM__ifeth>
      </interface>
     </show>
   </nf:filter>
  </nf:get>
</nf:rpc>
]]>]]>
switch(xmlin)# exit
switch#
```

次の例は、**show interface brief** コマンドの出力を XML に変換する方法を示しています。

```
switch# show interface brief | xmlin
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:if_manager"

message-id="1">
  <nf:get>
    <nf:filter type="subtree">
       <show>
          <interface>
              <brief/>
          </interface>
       </show>
     </nf:filter>
   </nf:get>
</nf:rpc>
]]>]]>
```



第 22 章

gNMI：gRPC ネットワーク管理インターフェイス

この章は次のトピックで構成されています。

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、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 *18*：サポートされる *gNMI RPC*

gNMI RPC	サポート対象
機能	はい
get	はい
設定	はい
登録	はい

gNMI RPC および SUBSCRIBE

NX-OS リリース 9.3(1) は、gNMI バージョン 0.5.0 がサポートされています。Cisco NX-OS リリース 9.3(1) は、gNMI バージョン 0.5.0 の次の部分がサポートされています。

表 *19*：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。リリース 9.3(1) では、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）]
- heart-beat interval
- prefix
- [qos]
- [冗長抑制（suppress_redundant）]

gNMI に関する注意事項と制限事項

gNMI に関する注意事項と制限事項は次のとおりです。

- Cisco NX-OS リリース 9.3(5) 以降、Get および Set がサポートされています。
- gNMI クエリは、パス内のワイルドカードをサポートしていません。
- Nexus デバイス向けの gRPCトラフィックは、デフォルト クラスのコントロール プレーン ポリサー（CoPP）にヒットします。gRPC ドロップの可能性を抑えるには、管理クラスの gRPC 構成ポートを使用して、カスタム CoPP ポリシーを構成してください。
- 管理 VRF とデフォルト VRF の両方で gRPC をイネーブルにし、後でデフォルト VRF でディセーブルにすると、管理 VRF の gNMI 通知が機能しなくなります。

  回避策として、 コマンドを入力して gRPC を完全に無効にし、 コマンドと既存の gRPC コンフィギュレーション コマンドを入力して再プロビジョニングします。**no feature grpcfeature grpc** たとえば、**grpc certificate** または **grpc port**。また、管理 VRF の既存の通知に再登録する必要もあります。
- 次のような既存の CLI 設定を使用して OpenConfig ルーティングポリシーをサブスクライブしようとすると、OpenConfig モデルの現在の実装により空の値が返されます。

  ```
  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
  ```

- サーバー証明書認証のみが実行されます。クライアント証明書がサーバーによって認証されません。
- gRPC 証明書が明示的に設定されている場合、保存されたスタートアップ コンフィギュレーションを使用して以前の Cisco NX-OS 9.3(x) イメージにリロードした後、gRPC 機能は接続を受け入れません。この問題を確認するには、 コマンドを入力します。ステータス行に次のようなエラーが表示されます。**show grpc gnmi service statistics**

  ```
  Status: Not running - Initializing...Port not available or certificate invalid.
  ```

  サービスを復元するには、適切な証明書コマンドを設定解除して設定します。
- Cisco NX-OS リリース 9.3(3) 以降では、カスタム gRPC 証明書を設定している場合、 コマンドを入力すると設定が失われます。**reload ascii** デフォルトの day-1 証明書に戻ります。**reload ascii** コマンドを入力した後には、スイッチをリロードします。スイッチが再び起動したら、gRPC カスタム証明書を再設定する必要があります。

（注） これは、 コマンドを入力した場合に適用されます。**grpc certificate**

- gRPC のデフォルト以外の VRF の到達可能性は、L3VNI/EVPN および IP 経由でのみサポートされます。ただし、デフォルト以外の VRF および VXLAN フラッドおよびラーニングでの MPLS を介した到達可能性はサポートされていません。
- origin、use_models、またはその両方の使用は、gNMI サブスクリプションではオプションです。
- gNMI サブスクリプションは、Cisco DME およびデバイス YANG データモデルをサポートします。Cisco NX-OS リリース 9.3(3)以降、サブスクライブは OpenConfig モデルをサポートします。
- 9.3(x) より前の Cisco NX-OS においてサポートされるプラットフォームの詳細については、そのリリース向けガイドのプログラマビリティ機能のプラットフォーム サポートを参照してください。Cisco NX-OS リリース 9.3(x) 以降、サポートされるプラットフォームの詳細については、Nexus Switch Platform Matrix を参照してください。
- この機能は、JSON および gnmi.proto エンコーディングをサポートします。この機能は、protobuf.any エンコーディングをサポートしていません。
- 各 gNMI メッセージの最大サイズは 12 MB です。収集されたデータの量が最大値 12 MB を超えると、収集されたデータはドロップされます。gNMI ON_CHANGE モードにのみ適用されます。

  この状況は、より小規模で詳細なデータ収集セットを処理する、焦点を絞ったサブスクリプションを作成することで回避できます。したがって、1 つの上位レベルのパスにサブスクライブする代わりに、パスの異なる下位レベルの部分に対して複数のサブスクリプションを作成してください。
- すべてのサブスクリプションで、最大 150K の集約 MO がサポートされます。より多くの MOに登録すると、収集データがドロップする可能性があります。
- この機能はサブスクリプション要求の単一パス プレフィックスをサポートしていませんが、サブスクリプションには空のプレフィックス フィールドを含めることができます。
- gNMI をサポートする gRPC プロセスは、CPU 使用率を CPU の 75% に、メモリを 1.5 GB に制限する HIGH_PRIO 制御グループを使用します。
- **show grpc gnmi** コマンドには、次の考慮事項があります。
  - gRPC エージェントは、コールが終了した後、最大 1 時間 gNMI コールを保持します。
  - コールの合計数が 2000 を超えると、gRPC エージェントは、内部クリーンアップルーチンに基づいて終了したコールを消去します。
- Cisco NX-OS リリース 10.2(3)F 以降では、デバイス YANG エフェメラルデータ（アカウンティングログおよびマルチキャスト）のサブスクリプションの変更がサポートされています。

gRPC サーバーは管理 VRF で実行されます。その結果、gRPC プロセスはこの VRF でのみ通信し、管理インターフェイスはすべての gRPC 呼び出しをサポートする必要があります。

gRPC 機能には、各スイッチの合計 2 つの gRPC サーバーのデフォルト VRF が含まれるようになりました。VRF ごとに 1 つの gRPC サーバーを実行することも、管理 VRF で 1 つの gRPC サーバーのみを実行することもできます。デフォルト VRF で gRPC をサポートすると、大量のトラフィック負荷が望ましくない管理 VRF からの gRPC コールの処理を柔軟にオフロードできます。

2 つの gRPC サーバーを構成する場合は、次の点に注意してください。

- VRF 境界は厳密に適用されるため、各 gRPC サーバーは相互に独立して要求を処理します。要求は VRF 間を通過しません。
- 2 台のサーバーは HA またはフォールト トレラントではありません。一方の gRPC サーバーは他方をバックアップせず、それらの間でスイッチオーバーまたはスイッチバックはありません。
- gRPC サーバーの制限は VRF 単位です。

gNMI の制限事項は次のとおりです。

- パスでは複数レベルのワイルドカード「...」は使用できません
- パスの先頭にワイルドカード「*」を使用することはできません
- キー名でワイルドカード「*」を使用することはできません
- キーにワイルドカードと値を混在させることはできません

次の表に、gNMI のワイルドカードサポートの詳細を示します。

表 *20 : gNMI* 要求のワイルドカード サポート

リクエストの種類	ワイルドカード サポート
gNMI GET	YES
gNMI 設定	NO
gNMI サブスクリプション（1 回）	YES
gNMI SUBSCRIBE、POLL	YES
gNMI SUBSCRIBE、STREAM、SAMPLE	YES
gNMI SUBSCRIBE、STREAM、TARGET_DEFINED	YES
gNMI SUBSCRIBE、STREAM、ON_CHANGE	NO

gNMI の構成

コマンドを使用して gNMI 機能を設定します。 **grpc gnmi**

grpc certificate コマンドで使用される証明書をスイッチにインポートするには、『Cisco Nexus 3500 シリーズ NX-OS セキュリティ構成ガイド、リリース 9.3(x)』の「識別証明書をインストールする」セクションを参照してください。

（注） インストールされている ID 証明書または と の値を変更すると、gRPC サーバーが再起動して変更が適用される場合があります。**grpc portgrpc certificate** gRPC サーバーが再起動すると、アクティブなサブスクリプションはすべてドロップされるため、再サブスクライブする必要があります。

手順の概要

1. **configure terminal**
2. **feature grpc**
3. （任意） **grpc port** *port-id*
4. （任意） **grpc certificate** 証明書 *ID*
5. **grpc gnmi max-concurrent-call** *number*

手順の詳細

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		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。

ステップ 2	**feature grpc**
	例：
	`switch-1# feature grpc`
	`switch-1(config)#`
		ダイヤルイン用の gNMI インターフェイスをサポートする gRPC エージェントを有効にします。

ステップ 3	（任意） **grpc port** *port-id* 例： `switch-1(config)# grpc port 50051`	ポート番号を設定します。*port-id* の範囲は 1024 ～ 65535 です。50051 がデフォルトです。 （注） このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。
ステップ 4	（任意） **grpc certificate** 証明書 *ID* 例： `switch-1(config)# grpc certificate cert-1`	証明書トラストポイント ID を指定します。詳細については、『Cisco Nexus 30009000 シリーズ NX-OS セキュリティ構成ガイド、リリース 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。

	コマンドまたはアクション	目的
		（注） このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。

ステップ 5	**grpc gnmi max-concurrent-call** *number*
	例：
	`switch-1(config)# grpc gnmi max-concurrent-call 16`
	`switch-1(config)#`
		スイッチ上の gNMI サーバーに対する同時ダイヤルイン呼び出しの制限を設定します。1 ～ 16 の範囲で制限を構成します。デフォルトの制限は 8 です。
		構成する最大値は、各 VRF に対するものです。制限を 16 に設定し、gNMI が管理 VRF とデフォルト VRF の両方に設定されている場合、各 VRF は 16 の同時 gNMI コールをサポートします。
		このコマンドは、進行中または進行中の gNMI コールには影響しません。代わりに、gRPC は新しいコールに制限を適用するため、進行中のコールは影響を受けず、完了できます。
		（注）
		設定された制限は、gRPCConfigOper サービスには影響しません。

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。

```
switch(config)# sh 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
Client Root Cert notBefore : n/a
Client Root Cert notAfter : n/a

Max concurrent calls : 8
Listen calls : 1
Active calls : 0
KeepAlive Timeout : 120

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

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

Max gNMI::Get concurrent : 6
Max grpc message size : 25165824
gNMI Synchronous calls : 3
gNMI Synchronous errors : 3
```

```
gNMI Adapter errors : 3
gNMI Dtx errors : 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-D 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。

• Cisco NX-OS リリース 9.3(3) 以降のキー/証明書の生成と構成の例 （285 ページ）

Cisco NX-OS リリース 9.3(3) 以降のキー/証明書の生成と構成の例

次に、キー/証明書を生成する例を示します。

（注） このタスクは、スイッチで証明書を生成する方法の例です。任意の Linux 環境で証明書を生成することもできます。実稼働環境では、CA 署名付き証明書の使用を検討する必要があります。

アイデンティティ証明書の生成の詳細については、『Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/security/configuration/guide/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x_chapter_011010.html#task_2088148

手順

ステップ 1 自己署名キーと pem ファイルを生成します。

```
switch# run bash sudo su
bash-4.3# openssl req -x509 -newkey rsa:2048 -keyout self_sign2048.key -out self_sign2048.pem -days
 365 -nodes
```

ステップ 2 キー ファイルと pem ファイルを生成した後、トラストポイント CA アソシエーションで使用するためにキー ファイルと pem ファイルをバンドルする必要があります。

```
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
```

ステップ 3 pkcs12 バンドルをトラストポイントに入力して、トラストポイント CA アソシエーションを設定します。

```
switch(config)# crypto ca trustpoint mytrustpoint
switch(config-trustpoint)# crypto ca import mytrustpoint pkcs12 self_sign2048.pfx Ciscolab123!
```

ステップ 4 セットアップを確認します。

```
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
```

ステップ 5 トラストポイントを使用するように gRPC を構成します。

```
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
```

ステップ 6 gRPC が証明書を使用していることを確認します。

```
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（設定されている場合）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。
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 50 の 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、機能）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 =` 成功、`!0 ==` 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

コマンド	説明
	とおりです。 • 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。

新しいクライアント ルート CA 証明書の生成

次に、クライアント ルートに新しい証明書を生成する例を示します。

- 信頼できる認証局（CA）

DigiCert などの信頼できる CA を使用する場合は、次の手順を実行します。

手順の概要

1. CA 証明書ファイルをダウンロードします。
2. Cisco NX-OS セキュリティ構成ガイドの手順に従って、NX-OS にインポートします。

手順の詳細

手順

	コマンドまたはアクション	目的
ステップ 1	CA 証明書ファイルをダウンロードします。	
ステップ 2	Cisco NX-OS セキュリティ構成ガイドの手順に従って、NX-OS にインポートします。	• トラストポイント CA のアソシエーションの作成の手順に従って、トラストポイントラベルを作成します。 • CA の認証の手順に従って、信頼できる CA 証明書を使用してトラストポイントを認証します。 （注） cat [CA_cert_file] からの CA 証明書を使用します。

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 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。

機能に関する注意事項と制限事項

次は機能に関する注意事項と制限事項です。

- Cisco NX-OS リリース 9.3(3) 以降、機能は OpenConfig モデルをサポートします。
- gNMI 機能は、gNMI サービスのオプションとしてサブスクライブと機能をサポートします。
- この機能は、JSON および gnmi.proto エンコーディングをサポートします。この機能は、protobuf.any エンコーディングをサポートしていません。
- 各 gNMI メッセージの最大サイズは 12 MB です。収集されたデータの量が最大値 12 MB を超えると、収集されたデータはドロップされます。

  この状況は、より小規模で詳細なデータ収集セットを処理する、焦点を絞ったサブスクリプションを作成することで回避できます。したがって、1 つの上位レベルのパスにサブスクライブする代わりに、パスの異なる下位レベルの部分に対して複数のサブスクリプションを作成してください。
- 同じサブスクリプション要求内のすべてのパスのサンプル間隔は同じである必要があります。同じパスで異なるサンプル間隔が必要な場合は、複数のサブスクリプションを作成します。
- この機能はサブスクリプション要求の単一パス プレフィックスをサポートしていませんが、サブスクリプションには空のプレフィックス フィールドを含めることができます。
- この機能は、Cisco DME およびデバイス YANG データモデルをサポートします。Openconfig YANG はサポートされていません。
- gNMI をサポートする gRPC プロセスは HIGH_PRIO cgroup を使用します。これにより、CPU 使用率が CPU の 75% に、メモリが 1.5 GB に制限されます。
- **show grpc gnmi** コマンドには、次の考慮事項があります。
  - このリリースでは、コマンドは XML 化されていません。
  - gRPC エージェントは、呼び出しが終了した後、最大 1 時間 gNMI 呼び出しを保持します。

- 呼び出しの合計数が 2000 を超えると、gRPC エージェントは内部クリーンアップ ルーチンに基づいて終了した呼び出しを消去します。

gRPC サーバーは管理 VRF で実行されます。その結果、gRPC プロセスはこの VRF でのみ通信し、管理インターフェイスはすべての gRPC 呼び出しをサポートする必要があります。

gRPC 機能には、各スイッチの合計 2 つの gRPC サーバのデフォルト VRF が含まれるようになりました。VRF ごとに 1 つの gRPC サーバーを実行することも、管理 VRF で 1 つの gRPC サーバーのみを実行することもできます。デフォルト VRF で gRPC をサポートすると、かなり部分のトラフィック負荷が望ましくないものである、管理 VRF からの gRPC コールの処理を柔軟にオフロードできます。

2 台の gRPC サーバーを構成する場合は、次の点に注意してください。

- VRF 境界は厳密に適用されるため、各 gRPC サーバーは相互に独立して要求を処理し、要求は VRF 間を通過しません。
- 2 台のサーバーは HA またはフォールト トレラントではありません。一方の gRPC サーバーは他方をバックアップせず、それらの間でスイッチオーバーまたはスイッチバックはありません。
- gRPC サーバーの制限は VRF 単位です。

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 パス規則に従って、XPATH の簡易形式である gNMI のスキーマ パス エンコーディング規則がパスに使用されます。https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-path-conventions.md

Get 操作の詳細については、gNMI 仕様の「状態情報のスナップショットの取得」セクションを参照してください。gRPC Network Management Interface (gNMI) https://github.com/openconfig/reference/blob/1cf43d2146f9ba70abb7f04f6b0f6eaa504cef05/rpc/gnmi/gnmi-specification.md

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。

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 のガイドライン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 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 Set 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。
- Set::Delete RPC の場合、操作対象の設定が大きすぎる可能性がある場合、MTX ログメッセージが警告します。

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

登録

サブスクライブに関する注意事項と制限事項

サブスクライブに関する注意事項と制限事項は次のとおりです。

- Cisco NX-OS リリース 9.3(3)以降、サブスクライブは OpenConfig モデルをサポートします。
- gNMI 機能は、gNMI サービスのオプションとしてサブスクライブと機能をサポートします。
- この機能は、JSON および gnmi.proto エンコーディングをサポートします。この機能は、protobuf.any エンコーディングをサポートしていません。

- 各 gNMI メッセージの最大サイズは 12 MB です。収集されたデータの量が最大値 12 MB を超えると、収集されたデータはドロップされます。

  この状況は、より小規模で詳細なデータ収集セットを処理する、焦点を絞ったサブスクリプションを作成することで回避できます。したがって、1 つの上位レベルのパスにサブスクライブする代わりに、パスの異なる下位レベルの部分に対して複数のサブスクリプションを作成してください。

- 同じサブスクリプション要求内のすべてのパスのサンプル間隔は同じである必要があります。同じパスで異なるサンプル間隔が必要な場合は、複数のサブスクリプションを作成します。

- この機能はサブスクリプション要求の単一パス プレフィックスをサポートしていませんが、サブスクリプションには空のプレフィックス フィールドを含めることができます。

- この機能は、Cisco DME およびデバイス YANG データモデルをサポートします。Openconfig YANG はサポートされていません。

- gNMI をサポートする gRPC プロセスは HIGH_PRIO cgroup を使用します。これにより、CPU 使用率が CPU の 75% に、メモリが 1.5 GB に制限されます。

- **show grpc gnmi** コマンドには、次の考慮事項があります。

  - このリリースでは、コマンドは XML 化されていません。

  - gRPC エージェントは、呼び出しが終了した後、最大 1 時間 gNMI 呼び出しを保持します。

  - 呼び出しの合計数が 2000 を超えると、gRPC エージェントは内部クリーンアップ ルーチンに基づいて終了した呼び出しを消去します。

gRPC サーバーは管理 VRF で実行されます。その結果、gRPC プロセスはこの VRF でのみ通信し、管理インターフェイスはすべての gRPC 呼び出しをサポートする必要があります。

gRPC 機能には、各スイッチの合計 2 つの gRPC サーバのデフォルト VRF が含まれるようになりました。VRF ごとに 1 つの gRPC サーバーを実行することも、管理 VRF で 1 つの gRPC サーバーのみを実行することもできます。デフォルト VRF で gRPC をサポートすると、かなり部分のトラフィック負荷が望ましくないものである、管理 VRF からの gRPC コールの処理を柔軟にオフロードできます。

2 台の gRPC サーバーを構成する場合は、次の点に注意してください。

- VRF 境界は厳密に適用されるため、各 gRPC サーバーは相互に独立して要求を処理し、要求は VRF 間を通過しません。

- 2 台のサーバーは HA またはフォールト トレラントではありません。一方の gRPC サーバーは他方をバックアップせず、それらの間でスイッチオーバーまたはスイッチバックはありません。

- gRPC サーバーの制限は VRF 単位です。

gNMI ペイロード

gNMI は、特定のペイロード形式を使用して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、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登録および受信します。ペイロードは、ナノ秒（1 秒 = 1000000000）の値を受け入れます。
- ON_CHANGE：サブスクライブしてスナップショットを受信し、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：サンプル、1 回
- POLL：SAMPLE、POLL
- ストリーム：サンプル、ストリーム
- ON_CHANGE：ON_CHANGE、STREAM

Origin

- DME：DME モデルへの登録
- デバイス：YANG モデルへの登録

名前

- DME = DME モデルに登録
- Cisco-NX-OS-device = YANG モデルに登録

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 Subscribe は、gNMI Subscribe 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 Subscribe サポートの詳細

- Syslog-oper モデル 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 3500 プラットフォーム 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。https://github.com/YangModels/yang/tree/master/vendor/cisco/nx/9.3-3

（注） タイムゾーン フィールドは、が入力された場合にのみ設定されます。**clock format show-timezone syslog** 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" : [
{
```

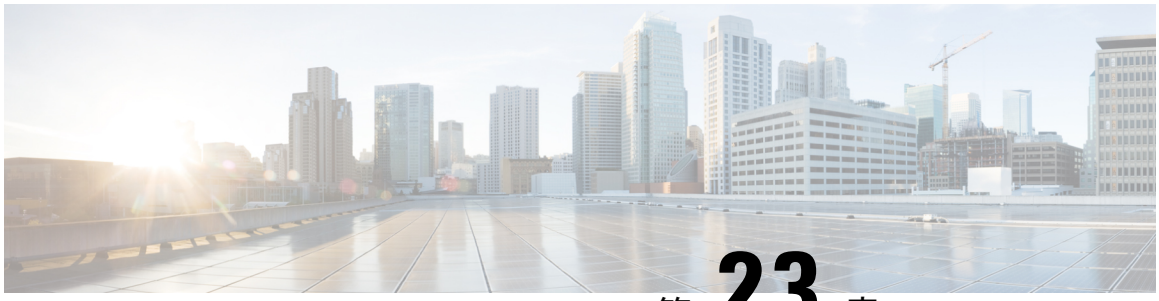
```
"id" : "mgmt0",
"ifstats-items" : {
"v2Sent" : "74",
"validV2Rcvd" : "79"
}
}
]
}
}
}
}
[06/21/19 15:58:43.210 PDT f91 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 141, sync_response:1
[06/21/19 15:59:01.341 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>
          <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 /volataile directory houses the mtx-internal.log, the log rolls over over time
so be sure to grab what  you need before thenbash-4.3# cd /volatile/

bash-4.3# cd /volaiflels -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 23 章

gNOI-gRPC ネットワーク操作インターフェイス

gNOI について

gRPC ネットワーク オペレーション インターフェイス（gNOI）は、ネットワーク デバイス上で操作コマンドを実行するための gRPC ベースのマイクロサービス セットを定義します。サポートされている操作コマンドは、Ping、Traceroute、Time、SwitchControlProcessor、Reboot、RebootStatus、CancelReboot、Activate、および Verify です。

gNOI は gRPC をトランスポート プロトコルとして使用し、設定は gNMI の設定と同じです。設定の詳細については、「gNMI の設定」を参照してください。https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/progammability/guide/b-cisco-nexus-9000-series-nx-os-programmability-guide-93x/b-cisco-nexus-9000-series-nx-os-programmability-guide-93x_chapter_0110001.html#id_107728

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

Cisco NX-OS リリース 10.1(1) では、gNOI は限られた数のコンポーネントに対してリモート プロシージャ コール（RPC）を定義し、その一部はハードウェア（光インターフェイスなど）に関連しています。

Proto ファイルは gRPC マイクロサービス用に定義されており、GitHub で入手できます。
https://github.com/openconfig/gnoi

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 *21 :*

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	対応

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、結果をストリームバックします。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。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、結果をストリームバックします。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。応答がクライアントに返されることが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ことが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	保留中の再起動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形式は、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、既知の良好な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、着信する新しい OS パッケージ用に常にプロアクティブにスペースを解放します。ターゲットには、有効な着信 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期待する必要があります。

次に、OS プロトコルの 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、ターゲットを再起動します。	再起動に失敗した場合は、ロールバックまたは回復できません。
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プロトコル RPC はサポートされていません。

次に、Cert 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。	-

gNOI 工場リセット

gNOI の初期設定へのリセット操作を行うと、指定されたモジュールのすべての永続ストレージが消去されます。これには、構成、すべてのログ データ、およびフラッシュと SSD のすべての内容が含まれます。リセットは直前のブートイメージでブートし、ライセンスを含むすべ

てのストレージを消去します。 gNOI の初期設定へのリセットは、次の 2 つのモードをサポートしています。

- 再フォーマットと再パーティションのみが可能な高速消去。
- データをセキュアに消去してワイプし、回復不可能にする、セキュア消去。

factory_reset.proto で定義されている gNOI の初期設定へのリセット操作は、デバイス上のすべての永続ストレージを消去します。こちらの factory_reset.proto リンクを参照してください：https://github.com/openconfig/gnoi/blob/master/factory_reset/factory_reset.proto

次に、gNOI FactoryReset サービスの例を示します。

```
/ The FactoryReset service exported by Targets.
service FactoryReset {
  // The Start RPC allows the Client to instruct the Target to immediately
  // clean all existing state and boot the Target in the same condition as it is
  // shipped from factory. State includes storage, configuration, logs,
  // certificates and licenses.
  //
  // Optionally allows rolling back the OS to the same version shipped from
  // factory.
  //
  // Optionally allows for the Target to zero-fill permanent storage where state
  // data is stored.
  //
  // If any of the optional flags is set but not supported, a gRPC Status with
  // code INVALID_ARGUMENT must be returned with the details value set to a
  // properly populated ResetError message.
  rpc Start(StartRequest) returns (StartResponse);
}

message StartRequest {
  // Instructs the Target to rollback the OS to the same version as it shipped
  // from factory.
  bool factory_os = 1;
  // Instructs the Target to zero fill persistent storage state data.
  bool zero_fill = 2;
}
```

次に、gNOI の工場リセットで使用される引数の詳細を示します。

- **factory_os = false**：工場出荷時の OS バージョンにロールバックするかどうかを指定します。NX-OS では **true** に設定することはサポートされていません。現在のブート イメージを保持する必要があります。
- **zero_fill**：時間のかかる包括的なセキュア消去を実行するかどうかを指定します。
  - **zero_fill = true**：factory-reset module all preserve-image force を指定します。
  - **zero_fill = false**：factory-reset module all bypass-secure-erase preserve-image force を指定します。

注意事項と制約事項

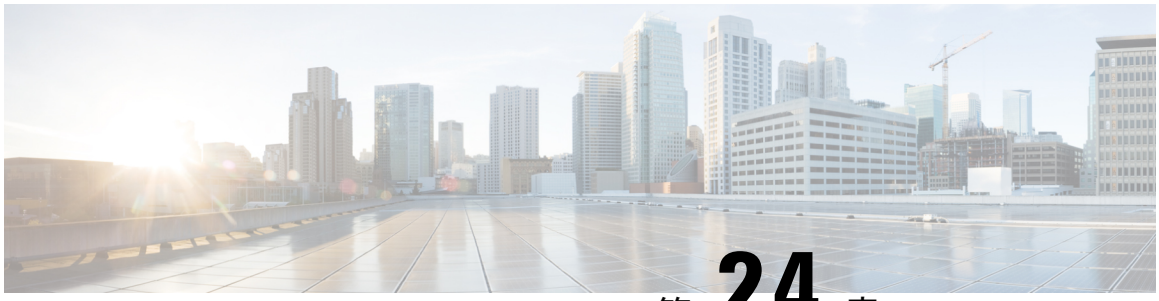
gNOI 機能には、次の注意事項と制約事項があります。

- 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。
- 10.2(1)F リリース以降、file.proto および cert.proto RPC がサポートされています。
- Nexus デバイス向けの gRPC トラフィックは、デフォルト クラスのコントロール プレーン ポリサー（CoPP）にヒットします。gRPC ドロップの可能性を抑えるには、管理クラスの gRPC 構成ポートを使用して、カスタム CoPP ポリシーを構成してください。

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 24 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。

  Cisco NX-OS リリース 7.0(3)I7(1) 以降、UDP およびセキュア UDP（DTLS）がテレメトリ トランスポート プロトコルとしてサポートされています。UDP を受信する接続先を追加できます。UDP およびセキュア UDP のエンコーディングは、GPB または JSON にすることができます。

  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 }
```

  IPv4 接続先の例:

```
destination-group 100
  ip address 171.70.55.69 port 50001 protocol UDP encoding 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。

（注） UDPプロトコルを使用した場合、受信側のOSやネットワークの負荷によってはパケットドロップが発生する場合があります。

- **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。
- **[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、再起動時に、構成およびストリーミング サービスを復元します。

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。

Telemetry のインストールとアップグレード

アプリケーションのインストール

テレメトリ アプリケーションは機能 RPM としてパッケージ化されており、NX-OS リリースに含まれています。RPM は、イメージ ブートアップの一部としてデフォルトでインストールされます。**feature telemetry** コマンドを使用して、アプリケーションを起動します。RPM ファイルは `/rpms` ディレクトリにあり、次のような名前が付けられています。

telemetry-*version-build_ID***.libn32_n9000.rpm**

telemetry-*version-build_ID***.libn32_n3000.rpm**

次の例のように：

```
telemetry-2.0.0-7.0.3.I5.1.lib32_n9000.rpm
telemetry-2.0.0-7.0.3.I5.1.lib32_n3000.rpm
```

増分更新と修正のインストール

RPM をデバイスのブートフラッシュにコピーし、`bash` プロンプトから次のコマンドを使用します：

```
feature bash
run bash sudo su
```

そして、デバイス ブートフラッシュに RPM のコピーをします。`bash` プロンプトから次のコマンドを使用します：

dnf upgrade *telemetry_new_version*.**rpm**

アプリケーションがアップグレードされ、アプリケーションを再起動すると変更が表示されます。

以前のバージョンにダウングレードします

テレメトリ アプリケーションを以前のバージョンにダウングレードするには、`bash` プロンプトから次のコマンドを使用します。

```
dnf downgrade telemetry
```

アクティブなバージョンの確認

現用系なバージョンを確認するには、スイッチの exec プロンプトから次のコマンドを実行します。

```
show install active
```

（注） [現用系のインストールを表示します（show install active）] コマンドは、アップグレードが実行された後に、インストールされている現用系な RPM のみを表示します。NX-OS にバンドルされているデフォルトの RPM は表示されません。

モデル動作テレメトリの注意事項と制限事項

テレメトリ 構成時の注意事項および制約事項は、次のとおりです。

- データ管理エンジン（DME）ネイティブ モデルをサポートする Cisco NX-OS リリースは、テレメトリをサポートします。
- 以下のサポートが実施されています。
  - DME データ収集
  - NX-API データ ソース
  - Google リモート プロシージャ コール（gRPC）トランスポートを介した Google プロトコル バッファ（GPB）エンコーディング
  - HTTP 経由の JSON エンコーディング
- サポートされている最小の送信間隔（ケイデンス）は、深さが 0 の場合の 5 秒です。0 より大きい深度値の最小ケイデンス値は、ストリーミングされるデータのサイズによって異なります。最小値未満のどのケイデンスでもを構成すると、望ましくないシステム動作が発生する可能性があります。
- テレメトリは、最大 5 つの遠隔管理受信者（接続先）をサポートします。5 つ以上の遠隔受信者を構成すると、システムが望ましくない動作をする可能性があります。
- テレメトリは、CPU 技術情報の最大 20% を消費する可能性があります。
- SSL 証明書ベースの認証とストリーミング データの暗号化を構成するには、**certificate***SSL cert path* **hostname"CN"** コマンドで自己署名 SSL 証明書を提供します。
- YANG パスにテレメトリ ケイデンスを設定するためのガイドラインは次のとおりです：
  - YANG ストリーミング コレクションには 1 つのスレッドが必要です。テレメトリに複数の YANG パスが存在する場合は、それぞれが異なる周期で実行して、同時スケジューリングと結果として生じる遅延を防ぐ必要があります。

- YANG パスのテレメトリ ケイデンスを構成する前に、合計ストリーミング時間を決定し、合計ストリーミング時間よりも大きい値にケイデンスを構成します。「*YANG* パスの頻度の構成」を参照してください。

古いリリースにダウングレードした後の構成コマンド

古いリリースにダウングレードした後、古いリリースではサポートされていない可能性があるため、一部の構成コマンドまたはコマンドオプションが機能不全になる可能性があります。古いリリースにダウングレードする場合は、新しいイメージが起動した後にテレメトリ機能を構成解除して再構成します。このシーケンスにより、サポートされていないコマンドまたはコマンド オプションの失敗を回避できます。

次の例は、この手順を表示しています。

- テレメトリ構成をファイルにコピーします。

```
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#
```

- ダウングレード操作を実行します。イメージが表示され、スイッチの準備ができたら、テレメトリ構成をスイッチにコピーして戻します。

```
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 エラーの動作

gRPC 受信者が 20 のエラーを送信した場合、スイッチ クライアントは gRPC 受信者への接続を無効化します。gRPC 受信者を有効にするには、接続先グループの下の接続先 IP アドレスの構成を解除して再構成する必要があります。一部のエラーの内容は、次のとおりです。

- gRPC クライアントがセキュアな接続に対して誤った証明書を送信する。
- gRPC レシーバでのクライアント メッセージの処理に時間がかかりすぎて、タイムアウトが発生する。別のメッセージ処理スレッドを使用してメッセージを処理することで、タイムアウトを回避している。

gRPC トランスポートのテレメトリ圧縮

gRPC トランスポートでは、テレメトリ圧縮のサポートが利用できます。**use-compression gzip** コマンドを使用して、圧縮を有効にすることができます。（**no use-compression gzip** コマンドで圧縮を無効にします。）

次の例では、圧縮を有効にします。

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-compression gzip
```

次の例は、圧縮が有効になっていることを表示しています。

```
switch(conf-tm-dest)# show telemetry transport 0 stats

Session Id:                     0
Connection Stats
   Connection Count             0
   Last Connected:              Never
   Disconnect Count             0
   Last Disconnected:           Never
Transmission Stats
   Compression:                 gzip
   Source Interface:            loopback1(1.1.3.4)
   Transmit Count:              0
   Last TX time:                None
   Min Tx Time:                 0                    ms
   Max Tx Time:                 0                    ms
   Avg Tx Time:                 0                    ms
   Cur Tx Time:                 0                    ms


switch2(config-if)# show telemetry transport 0 stats

Session Id: 0
Connection Stats
Connection Count 0
Last Connected: Never
Disconnect Count 0
Last Disconnected: Never
Transmission Stats
Compression: disabled
```

```
Source Interface: loopback1(1.1.3.4)
Transmit Count: 0
Last TX time: None
Min Tx Time: 0 ms
Max Tx Time: 0 ms
Avg Tx Time: 0 ms
Cur Tx Time: 0 ms
switch2(config-if)#
```

以下は、POST ペイロードとしての use-compression の例です。

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptCompression": {
                      "attributes": {
                        "name": "gzip"
                      }
                    }
                  }
                ]
              }
            }
```

gRPC チャンキングのサポート

リリース 9.2(1) 以降、gRPC チャンクのサポートが追加されました。ストリーミングを正常に行うには、gRPC が 12 MB を超えるデータ量を受信者に送信する必要がある場合、チャンクを有効にする必要があります。

gRPC ユーザーは、gRPC チャンクを行う必要があります。gRPC クライアント側は断片化を行い、gRPC サーバ側はリアセンブルを行います。テレメトリは引き続きメモリにバインドされており、メモリ サイズがテレメトリに許可されている制限である 12 MB を超えると、データが削除される可能性があります。チャンクをサポートするには、「テレメトリ コンポーネントおよびプロセス」で説明されているように、gRPC チャンク用に更新された、Cisco の GibLab で入手可能なテレメトリ `.proto` ファイルを使用します。

チャンク サイズは 64 ～ 4096 バイトです。

次に、NX-API CLI による構成例を表示します。

```
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
```

次に、NX-API REST による構成例を表示します。

```
{
    "telemetryDestGrpOptChunking": {
        "attributes": {
            "chunkSize": "2048",
            "dn": "sys/tm/dest-1/chunking"
        }
    }
}
```

Cisco MDS シリーズ スイッチなど、gRPC チャンクをサポートしていないシステムでは、次のエラー メッセージが表示されます。

```
MDS-9706-86(conf-tm-dest)# use-chunking size 200
ERROR: Operation failed: [chunking support not available]
```

NX-API センサー パスの制限

NX-API は、**show** コマンドを使用して、DME にまだ存在しないスイッチ情報を収集してストリーミングできます。ただし、DME からデータをストリーミングする代わりに NX-API を使用すると、次に示すように、固有の拡張制限があります。

- スイッチ バックエンドは、**show** コマンドなどの NX-API 呼び出しを動的に処理します。
- NX-API は、CPU の最大 20% を消費する可能性のあるいくつかのプロセスを生成します。
- NX-API データは、CLI から XML、JSON に変換されます。

以下は、過度の NX-API センサー パス帯域幅消費を制限するのに役立つ推奨ユーザー フローです。

1. **show** コマンドが NX-API をサポートしているかどうかを確認します。パイプ オプションを使用して、NX-API が VSH からのコマンドをサポートしているかどうかを確認できます：`<command> | json` または`<command> | json pretty`。

（注） スイッチが JSON 出力を返すまでに 30 秒以上かかるコマンドは避けてください。

2. フィルタまたはオプションを含めるように **show** コマンドを調整します。
   - 個々の出力に対して同じコマンドを列挙することは避けてください。たとえば **show vlan id 100**、**show vlan id 101** などです。代わりに、パフォーマンスを向上させるため、可能な場合は常に CLI 範囲オプションを使用してください。たとえば **show vlan id 100-110,204** です。

サマリーまたはカウンタのみが必要な場合は、showコマンド出力全体をダンプすることは避け、データ収集で必要な帯域幅とデータ ストレージを制限しないようにします。

3. NX-API をデータ送信元として使用するセンサー グループでテレメトリを構成します。**show** コマンドをセンサー パスとして追加する
4. CPI の使用を制限するために、それぞれの **show** コマンドの処理時間の 5 倍の周期でテレメトリを構成します。
5. ストリーミングされた NX-API 出力を既存の DME コレクションの一部として受信して処理します。

テレメトリの VRF サポート

テレメトリ VRF のサポートにより、トランスポート VRF を指定できます。これは、テレメトリ データ ストリームがフロント パネル ポートを介して出力され、SSH または NGINX 制御セッション間の競合の可能性を回避できることを意味します。

use-vrf *vrf-name* コマンドを使用して、トランスポート VRF を指定できます。

次の例では、トランスポート VRF を指定しています。

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-vrf test_vrf
```

以下は、POST ペイロードとしての use-vrf の例です。

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptVrf": {
                      "attributes": {
                        "name": "default"
                      }
                    }
                  }
                ]
              }
            }
```

証明書トラストポイント サポート

NX-OS リリース 10.1（1）以降、既存のグローバル レベル コマンドに**trustpoint** キーワードが追加されました。

次にあるのは、コマンド シンタックスです。

```
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
```

接続先ホスト名サポート

NX-OS リリース 10.1（1）以降、destination-group コマンドに **host** キーワードが追加されました。

次に、接続先ホスト名のサポートの例を示します。

```
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>
```

ノード識別子のサポート

NX-OS リリース 10.1（1）以降、**use-nodeid** コマンドを使用してテレメトリ受信者のカスタムノード識別子文字列を設定できます。デフォルトではホスト名が使用されますが、ノード識別子のサポートにより、テレメトリ受信者データの `node_id_str` の識別子を設定または変更できます。

usenode-id コマンドを使用して、テレメトリ接続先プロファイルを介してノード識別子を割り当てることができます。このコマンドはオプションです。

次の例は、ノード識別子の構成を表示しています。

```
switch-1(config)# telemetry
switch-1(config-telemetry)# destination-profile
switch-1(conf-tm-dest-profile)# use-nodeid test-srvr-10
switch-1(conf-tm-dest-profile)#
```

次の例は、ノード識別子が構成された後の受信側でのテレメトリ通知を示しています。

```
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
```

host コマンドの下の **use-nodeid** サブコマンドを使用します。接続先レベルの**use-nodeid** 構成は、グローバル レベルの構成よりも優先されます。

次の例はコマンド シンタックスを表示します。

```
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
```

テレメトリ 受信者の出力の例を表示します：

```
>> 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          :
```

YANG モデルのストリーミングのサポート

リリース 9.2(1) 以降、テレメトリは YANG（「Yet Another Next Generation」）データ モデリング言語をサポートします。テレメトリは、デバイス YANG と OpenConfig YANG の両方のデータ ストリーミングをサポートします。

センサ グループの単一テレメトリ収集

Cisco NX-OS 10.5(2)F 以降、ユーザーは **telemetry** CLI で新しい CLI オプション **merge-subscriptions** を使用して、センサー グループが複数のサブスクリプションに含まれており、接続先グループで解析ファイルが構成されていない場合に、センサー グループの単一のテレメトリ コレクションを作成できます。

- この設定はイベント サブスクリプションには使用できません。
- 接続先グループとマージサブスクリプションのフィルタファイル設定は、相互に互換性がありません。NX-OS ではブロックされません。その場合、収集はすべてのセンサーグループに対して個別に行われます。これは以前のリリースと同じです。
- サンプル間隔が大きいサブスクリプションのデータ送信のサンプル間隔は、min_sample_interval*floor(cur_sample_interval /min_sample_interval) の式を使用して決定されます。
  - min_sample_interval は、すべてのサブスクリプションのセンサー グループの最小サンプル間隔です。

- cur_sample 間隔は、そのセンサー グループの特定のサブスクリプションのサンプル間隔です。

- このオプションは以前のリリースでは使用できません。互換性チェックでの失敗を回避するには、ダウングレードする前にこの設定を削除してください。

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。これらの手順には、SSL/TLS 証明書と GPB エンコーディングを有効にして構成するオプションの手順も含まれています。

始める前に

スイッチは、Cisco NX-OS リリース 7.3(0)I5(1) 以降のリリースを実行している必要があります。

手順の概要

1. （任意） **openssl** *argument*
2. **configure terminal**
3. **feature telemetry**
4. **feature nxapi**
5. **nxapi use-vrf management**
6. **telemetry**
7. **[no] merge-subscriptions**
8. （任意） **certificate** *certificate_path host_URL*
9. （任意） トランスポート VRF を指定するか、gRPC トランスポートのテレメトリ圧縮を有効にします。
10. **sensor-group** *sgrp_id*
11. （任意） **data-source** *data-source-type*
12. **path** *sensor_path* **depth 0** [**filter-condition** *filter*] [**alias** *path_alias*]
13. **destination-group** *dgrp_id*
14. （任意） **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
15. （任意） **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
16. *ip_version* **address** *ip_address* **port** *portnum*
17. （任意） **use-chunking size** *chunking_size*
18. **subscription** *sub_id*
19. **snsr-grp** *sgrp_id* **sample-interval** *interval*
20. **dst-grp** *dgrp_id*

手順の詳細

手順

	コマンドまたはアクション	目的

ステップ 1	（任意） **openssl** *argument*
	例：
	次のような特定の引数を使用して、SSL/TLS 証明書を生成します。
	• RSA 秘密キーを生成するには： **openssl genrsa** *-cipher* **-out** *filename.key cipher-bit-length*
	例：
	`switch# openssl genrsa -des3 -out server.key 2048`
	• RSA キーを作成するには：**openssl rsa -in** *filename.key* **-out** *filename.key*
	例：
	`switch# openssl rsa -in server.key -out server.key`
	• 公開キーまたは秘密キーを含む証明書を作成するには、次の手順を実行します。 **openssl req**
	-encoding-standard filename.key filename.csr **-new -new -out -subj '/CN=localhost'**
	例：
	`switch# openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'`
	• 公開キーを作成するには：**openssl x509 -req** *-encoding-standard* **-days** *timeframe* **-in** *filename.csr* **-signkey** *filename.key* **-out** *filename.csr*
	例：
	`switch# openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt`
		データを受信するサーバー上に SSL または TLS 証明書を作成します。ここで、*private*.key ファイルは秘密キーであり、*public*.crt は公開キーです。

ステップ 2	**configure terminal**
	例：
	`switch# configure terminal`
	`switch(config)#`
		グローバル構成モードを開始します。

	コマンドまたはアクション	目的
ステップ 3	**feature telemetry**	ストリーミング テレメトリ機能を有効にします。
ステップ 4	**feature nxapi**	NX-API を有効にします。
ステップ 5	**nxapi use-vrf management**	NX-API 通信に使用する VRF 管理を有効にします。

ステップ 6	**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。

ステップ 7	**[no] merge-subscriptions** 例： `switch# merge-subscriptions`	複数のサブスクリプションに含まれるすべてのセンサー グループに対して 1 つのコレクションを作成します。
ステップ 8	（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。

ステップ 9	（任意）トランスポート VRF を指定するか、gRPC トランスポートのテレメトリ圧縮を有効にします。
	例：
	`switch(config-telemetry)# destination-profile`
	`switch(conf-tm-dest-profile)# use-vrf default`
	`switch(conf-tm-dest-profile)# use-compression gzip`
	`switch(conf-tm-dest-profile)# use-retry size 10`
	`switch(conf-tm-dest-profile)# source-interface loopback1`
		• **destination-profile** コマンドを入力して、デフォルトの接続先プロファイルを指定します。
		• 次のコマンドを任意で入力します。
		• **use-vrf** *vrf* で接続先 VRF を指定します。
		• **use-compression gzip** を使用して、接続先の圧縮方法を指定します。
		• **use-retry size** *size* を使用して、送信再試行の詳細を指定します。再試行バッファサイズは 10～1500 メガバイトです。
		• **source-interface** *interface-name* は、構成されたインターフェイスから送信元 IP アドレスを持つ接続先にデータをストリーミングします。
		（注）
		use-vrf コマンドを構成した後、新しい VRF 内に新しい接続先 IP アドレスを構成する必要があります。ただし、接続先を構成解除して再構成することにより、同じ接続先 IP アドレスを再利用できます。このアクションにより、テレメトリ データは新しい VRF でも同じ接続先 IP アドレスにストリーミングされます。

	コマンドまたはアクション	目的

ステップ 10	**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。

ステップ 11	（任意） **data-source** *data-source-type* 例： `switch(config-telemetry)# data-source NX-API`	データ ソースを選択します。データ ソースとして YANG、DME または NX-API のいずれかを選択します。 （注） DME はデフォルトのデータ ソースです。

ステップ 12	**path** *sensor_path* **depth 0** [**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`
		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、 キーワードが導入されています。**alias**
		• **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。

	コマンドまたはアクション	目的

	• 次のコマンドは、OpenConfig YANG に適用されます。
	`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`
		（注）
		query-condition パラメータ — DME の場合、DN に基づいて、次の構文で MOTL および一時データをフェッチするために query-condition パラメータを指定できます。クエリ条件「rsp-foreign-subtree=ephemeral」。
		• YANG モデルの場合、センサー パスの形式は 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）
		、、およびパラメータは、現在 YANG ではサポートされていません。
		depthfilter-conditionquery-condition
		openconfig YANG モデルの場合は、 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。https://github.com/YangModels/yang/tree/master/vendor/cisco/nx
		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。パッチ RPM のインストールの詳細については、「バッシュからパッチ *RPM* を追加する」を参照してください。
		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

ステップ 13	**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

ステップ 14	（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`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50009 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 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。

ステップ 15	（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。

ステップ 16	*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`
		発信データの宛先プロファイルを作成します。ip_version は、ip（IPv4 の場合）または ipv6（IPv6 の場合）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。

ステップ 17	（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。

ステップ 18	**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
ステップ 19	**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。
ステップ 20	**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

手順

	コマンドまたはアクション	目的

ステップ 1	**show telemetry control database sensor-groups**
	例：
	`switch-1# 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-1#`
	`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`
		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。

	コマンドまたはアクション	目的
	dst-grp 1 snsr-grp 1 sample-interval 5000 **snsr-grp 2 sample-interval 5000**	
ステップ 2	**sensor group** *number* 例： `switch-1(config-telemetry)# sensor group1`	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。
ステップ 3	**subscription** *number* 例： `switch-1(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。
ステップ 4	**snsr-grp** *number* **sample-interval** *milliseconds* 例： `switch-1(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。

ステップ 5	**show system resources**
	例：
	`switch-1# 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
```

この例に示すように、**use-vrf** コマンドと **use-compression gzip** コマンドを使用して、gRPC のトランスポート VRF とテレメトリ データ圧縮を指定できます。

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(conf-tm-dest-profile)# use-vrf default
switch(conf-tm-dest-profile)# use-compression gzip
switch(conf-tm-dest-profile)# sensor-group 1
switch(conf-tm-sensor)# path sys/bgp depth unbounded
switch(conf-tm-sensor)# destination-group 1
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-sub)# snsr-grp 1 sample-interval 10000
```

Telemetry Merge サブスクリプションの構成例

次に、merge-subscription を構成する例を示します。

```
Telemetry
  merge-subscriptions
  destination-group 1
    ip address 192.186.1.2 port 1 protocol HTTP encoding JSON
  Destination-group 2
    ip address 192.168.1.3 port 2 protocol gRPC encoding GPB
  sensor-group 1
    path sys/fm
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 10000
  subscription 2
    dst-grp 2
    snsr-grp 1 sample-interval 25000
```

センサー グループ 1 のデータは 10 秒ごとに収集されます。収集されたデータは、10 秒ごとにサブスクリプション 1 の接続先に送信され、20 秒ごとにサブスクリプション 2 の接続先に送信されます。

サブスクリプション 2 の snsr-grp 1 の例では、min_sample 間隔は 10 秒、cur_sample_interval は 25 秒です。したがって、そのデータは 20 秒ごとに送信されます。

```
Sample_interval = 10*floor(25/10)
            = 10*2
                          = 20s
```

構成の確認

次のコマンドを使用して、IOA 構成を確認します。

```
show telemetry control database sensor-groups
Sensor Group Database size = 1
Row ID     Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
  SubID
1          1                Timer   /DME            10000/Running      2
      1
Collection Time in ms (Cur/Min/Max): 1/1/2
Encoding Time in ms (Cur/Min/Max): 0/0/0
```

```
Transport Time in ms (Cur/Min/Max): 10003/10003/10003
Streaming Time in ms (Cur/Min/Max): 10004/10004/10006
Collection Statistics:
collection_id_dropped      = 0
last_collection_id_dropped = 0
drop_count                 = 0
Configuration method: CONFIG_DME-ADMIN
2          1               Timer   /DME            20000*/Running      2
       2
Collection Time in ms (Cur/Min/Max): 1/1/1
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 4004/4004/4004
Streaming Time in ms (Cur/Min/Max): 4005/4005/4005
Collection Statistics:
collection_id_dropped      = 0
last_collection_id_dropped = 0
drop_count                 = 0
Configuration method: CONFIG_DME-ADMIN
*Calculated sample interval for merge-subscriptions
```

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-1(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-1(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単な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 コレクション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 コレクション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

    Memory:
            Allowed Memory Limit               = 1181116006 bytes
            Occupied Memory                    = 93265920 bytes

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
```

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
Last Disconnected:   Never
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
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never
Last Disconnected:   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> 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。

```
Session Id:                   0
Transmission Stats
   Compression:               disabled
   Source Interface:          not set()
   Transmit Count:            319297
   Last TX time:              Fri Aug 02 03:51:15.287 UTC
   Min Tx Time:               1                  ms
   Max Tx Time:               3117               ms
   Avg Tx Time:               3                  ms
   Cur Tx Time:               1                  ms
```

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 Errors
Connection Error Count:     0
Transmission Errors
Tx Error Count:             30
Last Tx Error:              Thu Aug 01 04:39:47.083 UTC
Last Tx Return Code:        No error
```

show telemetry control databases sensor-paths

これらの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
```

tmtrace.bin

この BASH シェル コマンドは、テレメトリ トレースを収集して出力します。

```
switch# configure terminal
switch(config)# feature bash
switch(config)# run bash
bash-4.2$ tmtrace.bin -d tm-errors
bash-4.2$ tmtrace.bin -d tm-logs
bash-4.2$ tmtrace.bin -d tm-events
```

例：

```
bash-4.2$ tmtrace.bin -d tm-logs
[01/25/17 22:52:24.563 UTC 1 29130] [3944724224][tm_ec_dme_auth.c:59] TM_EC: Authentication
 refresh url http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.565 UTC 2 29130] [3944724224][tm_ec_dme_rest_util.c:382] TM_EC:
Performed POST request on http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.566 UTC 3 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER:
Starting leaf timer for leaf:0x11e17ea4 time_in_ms:540000
[01/25/17 22:52:45.317 UTC 4 29130] [3944724224][tm_ec_dme_event_subsc.c:790] TM_EC:
Event subscription database size 0
[01/25/17 22:52:45.317 UTC 5 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER:
Starting leaf timer for leaf:0x11e17e3c time_in_ms:50000
bash-4.2#
```

（注） **tm-logs** オプションは冗長であるため、デフォルトでは有効になっていません。

`tmtrace.bin -LD tm-logs` コマンドで **tm-logs** を有効にします。

`tmtrace.bin -LW tm-logs` コマンドを使用して **tm-logs** を無効にします。

show system internal telemetry trace

show system internal telemetry trace [**tm-events** | **tm-errors** |**tm-logs** | **all**] コマンドは、システムの内部テレメトリ トレース情報を表示します。

```
switch# show system internal telemetry trace all
Telemetry All Traces:
Telemetry Error Traces:
[07/26/17 15:22:29.156 UTC 1 28577] [3960399872][tm_cfg_api.c:367] Not able to destroy
dest profile list for config node rc:-1610612714 reason:Invalid argument
[07/26/17 15:22:44.972 UTC 2 28577] [3960399872][tm_stream.c:248] No subscriptions for
destination group 1
[07/26/17 15:22:49.463 UTC 3 28577] [3960399872][tm_stream.c:576] TM_STREAM: Subscriptoin
 1 does not have any sensor groups
```

```
3 entries printed
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
 successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
switch#

switch# show system internal telemetry trace tm-logs
Telemetry Log Traces:
0 entries printed
switch#
switch# show system internal telemetry trace tm-events
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
 successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
[07/26/17 15:19:40.610 UTC 4 28577] [3960399872][tm_init_n9k.c:207] Adding telemetry to
 cgroup
[07/26/17 15:19:40.670 UTC 5 28577] [3960399872][tm_init_n9k.c:215] Added telemetry to
cgroup successfully!

switch# show system internal telemetry trace tm-errors
Telemetry Error Traces:
0 entries printed
switch#
```

NX-API を使用したテレメトリの構成

NX-API を使用したテレメトリの構成

スイッチ DME のオブジェクト モデルでは、「DME のテレメトリ モデル」のセクションで説明されているように、テレメトリ機能の構成がオブジェクトの階層構造で定義されています。構成する主なオブジェクトは次のとおりです。

- **fmEntity** — NX-API およびテレメトリ機能の状態が含まれています。
  - **fmNxapi** — NX-API の状態が含まれています。
  - **fmTelemetry** — テレメトリ機能の状態が含まれています。
- **telemetryEntity** — テレメトリ機能の構成が含まれています。
  - **telemetrySensorGroup** — テレメトリのために監視される 1 つ以上のセンサー パスまたはノードの定義が含まれています。テレメトリ エンティティには、1 つ以上のセンサー グループを含めることができます。
    - **telemetryRtSensorGroupRel** — センサー グループをテレメトリ サブスクリプションに関連付けます。

- **telemetrySensorPath** — モニタリングされるパス。センサー グループには、このタイプのオブジェクトを複数含めることができます。

- **telemetryDestGroup** — テレメトリ データを受信する 1 つ以上の接続先の定義が含まれています。テレメトリ エンティティには、1 つ以上の接続先グループを含めることができます。
  - **telemetryRtDestGroupRel** — 接続先グループをテレメトリ サブスクリプションに関連付けます。
  - **telemetryDest** — 接続先アドレス接続先グループには、このタイプのオブジェクトを複数含めることができます。

- **telemetrySubscription** — 1 つ以上のセンサー グループからのテレメトリ データを 1 つ以上の接続先グループに送信する方法とタイミングを指定します。
  - **telemetryRsDestGroupRel** — テレメトリ サブスクリプションを接続先グループに関連付けます。
  - **telemetryRsSensorGroupRel** — テレメトリ サブスクリプションをセンサー グループに関連付けます。

- **telemetryCertificate** — テレメトリ サブスクリプションを証明書とホスト名に関連付けます。

NX-API を使用してテレメトリ機能を設定するには、テレメトリ オブジェクト構造の JSON 表現を構築し、HTTP または HTTPS POST 操作で DME にプッシュする必要があります。

（注） NX-API の使用に関する詳細な手順は、『*Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*（Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK ユーザー ガイドと API リファレンス）』を参照してください。

始める前に

スイッチは、Cisco NX-OS リリース 7.3（0）I5（1）以降のリリースを実行している必要があります。

CLI から NX-API を実行するようにスイッチを構成する必要があります。

```
switch(config)# feature nxapi
```

NX-API は、管理 VRF を介してテレメトリ データを送信します。

```
switch(config)# nxapi use-vrf management
```

```
nxapi use-vrf vrf_name
nxapi http port port_number
```

手順の概要

1. テレメトリ機能を有効にします。
2. テレメトリ構成を記述するために、JSON ペイロードのルート レベルを作成します。
3. 定義されたセンサー パスを含むセンサーグループを作成します。
4. （任意） SSL/TLS 証明書とホストを追加します。
5. テレメトリの接続先グループを定義します。
6. テレメトリの接続先プロファイルを定義します。
7. テレメトリ データの送信先となる IP アドレスとポート番号で構成される、1 つ以上のテレメトリの接続先を定義します。
8. 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。
9. テレメトリ サブスクリプションを作成して、テレメトリの動作を構成します。
10. ルート要素の下の **telemetrySubscription** 要素に子オブジェクトとしてセンサー グループ オブジェクトを追加します（**telemetryEntity**）。
11. サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ センサー グループに関連付け、データ サンプリング動作を指定します。
12. テレメトリをモニタリングする 1 つ以上のセンサー パスまたはノードを定義します。
13. センサー パスを子オブジェクトとしてセンサー グループ オブジェクト（**telemetrySensorGroup**）に追加します。
14. 接続先を子オブジェクトとして接続先グループ オブジェクト（**telemetryDestGroup**）に追加します。
15. 接続先グループ オブジェクトを子オブジェクトとしてルート要素に追加します（**telemetryEntity**）。
16. センサー グループをサブスクリプションに関連付けるために、テレメトリ センサー グループの子オブジェクトとして関係オブジェクトを作成します。
17. テレメトリ接続先グループの子オブジェクトとして関係オブジェクトを作成して、接続先グループをサブスクリプションに関連付けます。
18. サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ接続先グループに関連付けます。
19. テレメトリ構成のために、結果の JSON 構造を HTTP/HTTPS POST ペイロードとして NX-API エンドポイントに送信します。

手順の詳細

手順

	コマンドまたはアクション	目的

ステップ 1	テレメトリ機能を有効にします。
	例：
	`{`
	`  "fmEntity" : {`
		ルート要素は **fmTelemetry** であり、この要素のベース パスは `sys/fm` です。**adminSt** 属性を有効に構成します。

	コマンドまたはアクション	目的

	<pre>"children" : [{
	"fmTelemetry" : {
	"attributes" : {
	"adminSt" : "enabled"
	}
	}
	}
	]
	}
	}</pre>	

ステップ 2	テレメトリ構成を記述するために、JSONペイロードのルート レベルを作成します。
	例：
	<pre>{
	"telemetryEntity": {
	"attributes": {
	"dn": "sys/tm"
	},
	}
	}</pre>
		ルート要素は **telemetryEntity** であり、この要素のベース パスは `sys/tm` です。**dn** 属性を `sys/tm` として構成します。

ステップ 3	定義されたセンサー パスを含むセンサーグループを作成します。
	例：
	<pre>"telemetrySensorGroup": {
	"attributes": {
	"id": "10",
	"rn": "sensor-10"
	"dataSrc": "NX-API"
	}, "children": [{
	}]
	}</pre>
		テレメトリ センサー グループは、クラス **telemetrySensorGroup** のオブジェクトで定義されます。以下のオブジェクト属性を構成します。
		• **id** — センサー グループの識別子。現在は、数字の ID 値のみサポートされています。
		• **rn**—センサー グループ オブジェクトの相対名（形式: **sensor-***id*）。
		• **dataSrc** ：**DEFAULT**、**DME**、**YANG**、 または **NX-API** からデータ送信元を選択します。
		センサーグループオブジェクトの子には、センサーパスと 1 つ以上の関係オブジェクト（**telemetryRtSensorGroupRel**）が含まれ、センサーグループをテレメトリ サブスクリプションに関連付けます。

ステップ 4	（任意） SSL/TLS 証明書とホストを追加します。
	例：
	<pre>{
	"telemetryCertificate": {
	"attributes": {
	"filename": "root.pem"
	"hostname": "c.com"
	}
	}
	}</pre>
		telemetryCertificate は、テレメトリ サブスクリプション/接続先で SSL/TLS 証明書の場所を定義します。

	コマンドまたはアクション	目的
ステップ 5	テレメトリの接続先グループを定義します。 例： `{ "telemetryDestGroup": { "attributes": { "id": "20" } } }`	テレメトリ接続先グループが **telemetryEntity** で定義されています。id 属性を構成します。
ステップ 6	テレメトリの接続先プロファイルを定義します。 例： `{ "telemetryDestProfile": { "attributes": { "adminSt": "enabled" }, "children": [ { "telemetryDestOptSourceInterface": { "attributes": { "name": "lo0" } } } ] } }`	テレメトリの接続先プロファイルは、**telemetryDestProfile** で定義されています。 • **adminSt** 属性を有効に設定します。 • **telemetryDestOptSourceInterface** の下で、構成されたインターフェイスからソース IP アドレスを持つ接続先にデータをストリーミングするためのインターフェイス名を使用して **name** 属性を構成します。

ステップ 7	テレメトリ データの送信先となる IP アドレスとポート番号で構成される、1 つ以上のテレメトリの接続先を定義します。
	例：
	`{ "telemetryDest": { "attributes": { "addr": "1.2.3.4", "enc": "GPB", "port": "50001", "proto": "gRPC", "rn": "addr-[1.2.3.4]-port-50001" } } }`
		テレメトリの接続先は、クラス **telemetryDest** のオブジェクトで定義されます。以下のオブジェクト属性を構成します。
		• **addr** — 接続先の IP アドレス。
		• **port** — 接続先のポート番号。
		• **rn — path-[***path***]** 形式の接続先オブジェクトの相対名。
		• **enc**—送信されるテレメトリ データのエンコーディング タイプ。NX-OS は以下をサポートします。
		  • gRPC の Google Protocol Buffer（GBP）。
		  • C の JSON。
		  • UDP およびセキュア UDP（DTLS）の場合は GPB または JSON。

	コマンドまたはアクション	目的

		• **proto** — 送信されるテレメトリ データのトランスポート プロトコル タイプ。NX-OS は以下をサポートします。
		  • gRPC
		  • HTTP
		  • VUDP とセキュア UDP（DTLS）
		• サポートされているエンコード タイプは次のとおりです。
		  • HTTP/JSON はい
		  • HTTP/Form-data はい Bin Logging でのみサポートされます。
		  • GRPC/GPB-Compact はい ネイティブデータ ソースのみ。
		  • GRPC/GPB はい
		  • UDP/GPB はい
		  • UDP/JSON はい

ステップ 8	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。 **例**： `{ "telemetryDestGrpOptChunking": { "attributes": { "chunkSize": "2048", "dn": "sys/tm/dest-1/chunking" } } }`	詳細については、「gRPC チャンキングのサポート（336 ページ）」を参照してください。
ステップ 9	テレメトリ サブスクリプションを作成して、テレメトリの動作を構成します。 **例**： `"telemetrySubscription": { "attributes": { "id": "30", "rn": "subs-30" }, "children": [{ }] }`	テレメトリ サブスクリプションは、クラス **telemetrySubscription** のオブジェクトで定義されます。以下のオブジェクト属性を構成します。 • **id** — サブスクリプションの識別子。現在は、数字の ID 値のみサポートされています。 • **rn** — **subs-***id* という形式のサブスクリプション オブジェクトの相対名。

	コマンドまたはアクション	目的
		サブスクリプション オブジェクトの子には、センサー グループ（**telemetryRsSensorGroupRel**）および接続先グループ（**telemetryRsDestGroupRel**）の関係オブジェクトが含まれます。
ステップ 10	ルート要素の下の **telemetrySubscription** 要素に子オブジェクトとしてセンサー グループ オブジェクトを追加します（**telemetryEntity**）。 **例**： `{ "telemetrySubscription": { "attributes": { "id": "30" } "children": [{ "telemetryRsSensorGroupRel": { "attributes": { "sampleIntvl": "5000", "tDn": "sys/tm/sensor-10" } } } ] } }`	
ステップ 11	サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ センサー グループに関連付け、データ サンプリング動作を指定します。 **例**： `"telemetryRsSensorGroupRel": { "attributes": { "rType": "mo", "rn": "rssensorGroupRel-[sys/tm/sensor-10]", "sampleIntvl": "5000", "tCl": "telemetrySensorGroup", "tDn": "sys/tm/sensor-10", "tType": "mo" } }`	関係オブジェクトはクラス **telemetryRsSensorGroupRel** であり、**telemetrySubscription** の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • **rn** — **rssensorGroupRel-[sys/tm/***sensor-group-id***]** 形式の関係オブジェクトの相対名。 • **sampleIntvl** — ミリ秒単位のデータ サンプリング期間。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。 • **tCl** — **telemetrySensorGroup** であるターゲット（センサー グループ）オブジェクトのクラス。

	コマンドまたはアクション	目的
		• **tDn** — **sys/tm/***sensor-group-id* であるターゲット（センサーグループ）オブジェクトの識別名。 • **rType** — 管理対象オブジェクト用**mo** の関係タイプ。 • **tType** — 管理対象オブジェクト用 **mo** のターゲット タイプ。

ステップ 12	テレメトリをモニタリングする 1 つ以上のセンサー パスまたはノードを定義します。
	例：
	単一センサー パス
	
	`{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0", "alias": "cdp_alias", } } }`
	
	例：
	NX-API の単一センサー パス
	
	`{ "telemetrySensorPath": { "attributes": { "path": "show interface", "path": "show bgp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }`
	
	例：
	複数のセンサー パス
		センサー パスは、クラス **telemetrySensorPath** のオブジェクトで定義されます。以下のオブジェクト属性を構成します。
		• **path** — モニタリングされるパス。
		• **rn** — **path-[***path***]** 形式のパス オブジェクトの相対名：
		• **depth** — センサー パスの取得レベル。**0** の深さ設定は、ルート MO プロパティのみを取得します。
		• **filterCondition** — （オプション）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します。DME はフィルター式を提供します。フィルタリングの詳細については、クエリの作成に関する Cisco APIC REST API の使用注意事項を参照してください。 https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635
		• **alias**：このパスのエイリアスを指定します。

	コマンドまたはアクション	目的

	<pre>{
	    "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"
	        }
	    }
	}</pre>
	例：
	BGP 無効化イベントの単一センサー パス フィルタリング：
	<pre>{
	    "telemetrySensorPath": {
	        "attributes": {
	            "path": "sys/cdp",
	            "rn": "path-[sys/cdp]",
	            "excludeFilter": "",
	            "filterCondition":
	"eq(fmBgp.operSt.\"disabled\")",
	            "path": "sys/fm/bgp",
	            "secondaryGroup": "0",
	            "secondaryPath": "",
	            "depth": "0"
	        }
	    }
	}</pre>	

ステップ 13	センサー パスを子オブジェクトとしてセンサー グループ オブジェクト（**telemetrySensorGroup**）に追加します。	
ステップ 14	接続先を子オブジェクトとして接続先グループ オブジェクト（**telemetryDestGroup**）に追加します。	

	コマンドまたはアクション	目的
ステップ 15	接続先グループ オブジェクトを子オブジェクトとしてルート要素に追加します（**telemetryEntity**）。	
ステップ 16	センサー グループをサブスクリプションに関連付けるために、テレメトリ センサー グループの子オブジェクトとして関係オブジェクトを作成します。 例： `"telemetryRtSensorGroupRel": { "attributes": { "rn": "rtsensorGroupRel-[sys/tm/subs-30]", "tCl": "telemetrySubscription", "tDn": "sys/tm/subs-30" } }`	関係オブジェクトはクラス **telemetryRtSensorGroupRel** であり、**telemetrySensorGroup** の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • **rn** —**rtsensorGroupRel-[sys/tm/***subscription-id***]** の形式の関係オブジェクトの相対名。 • **tCl** — サブスクリプション オブジェクト **telemetrySubscription** のターゲット クラス。 • **tDn** — **sys/tm/***subscription-id* である、サブスクリプションオブジェクトのターゲット識別名。
ステップ 17	テレメトリ接続先グループの子オブジェクトとして関係オブジェクトを作成して、接続先グループをサブスクリプションに関連付けます。 例： `"telemetryRtDestGroupRel": { "attributes": { "rn": "rtdestGroupRel-[sys/tm/subs-30]", "tCl": "telemetrySubscription", "tDn": "sys/tm/subs-30" } }`	関係オブジェクトはクラス **telemetryRtDestGroupRel** であり、**telemetryDestGroup** の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • **rn** —**rtdestGroupRel-[sys/tm/***subscription-id***]** の形式の関係オブジェクトの相対名。 • **tCl** — サブスクリプション オブジェクト **telemetrySubscription** のターゲット クラス。 • **tDn** — **sys/tm/***subscription-id* である、サブスクリプションオブジェクトのターゲット識別名。
ステップ 18	サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ接続先グループに関連付けます。 例： `"telemetryRsDestGroupRel": { "attributes": { "rType": "mo", "rn": "rsdestGroupRel-[sys/tm/dest-20]", "tCl": "telemetryDestGroup", "tDn": "sys/tm/dest-20", "tType": "mo" } }`	関係オブジェクトはクラス **telemetryRsDestGroupRel** であり、**telemetrySubscription** の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • **rn** — **rsdestGroupRel-[sys/tm/***destination-group-id***]** の形式の関係オブジェクトの相対名。 • **tCl**— ターゲット (接続先グループ) オブジェクトのクラス **telemetryDestGroup**。 • **tDn** — 接続先グループ ID であるターゲット（接続先グループ）オブジェクト **sys/tm/***destination-group-id* の識別名。

	コマンドまたはアクション	目的
		• **rType** — 管理対象オブジェクト用**mo** の関係タイプ。 • **tType** — 管理対象オブジェクト用 **mo** のターゲット タイプ。
ステップ 19	テレメトリ構成のために、結果の JSON 構造を HTTP/HTTPS POST ペイロードとして NX-API エンドポイントに送信します。	テレメトリ エンティティのベース パスは `sys/tm` で、NX-API エンドポイントは次のとおりです。 `{{URL}}/api/node/mo/sys/tm.json`

例

以下は、1 つの POST ペイロードに収集された、その前のすべてのステップの例です（一部の属性が一致しない場合があることに注意してください）。

```
{
  "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
               |    |    @name:dataSrc
```

```
|    |      @type:DataSource
|    |
|    |----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
```

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。

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（380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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(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。

ステップ 4	**path interface**
	例：
	`switch-1(conf-tm-sensor)# path interface`
	`switch-1(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。

	コマンドまたはアクション	目的
		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。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(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。

ステップ 4	**path interface query-condition counters=[detailed]**
	例：
	`switch-1(conf-tm-sensor)# path interface query-condition counters=[detailed]`
	`switch-1(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のみを照会します。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 4	**path interface query-condition oper-speed=[***speed***]**
	例：
	`switch-1(conf-tm-sensor)# path interface query-condition oper-speed=[1G,40G]`
	`switch-1(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。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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ます。

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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(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。

ステップ 4	**path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
	例：
	`switch-1(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G]`
	`switch-1(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。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 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

	コマンドまたはアクション	目的
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000` `switch-1(conf-tm-sub)#`	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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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 （380 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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

	コマンドまたはアクション	目的

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(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。

ステップ 4	**path environment**
	例：
	`switch-1(conf-tm-sensor)# path environment`
	`switch-1(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。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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 （380 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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(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。

ステップ 4	**path resources**
	例：
	`switch-1(conf-tm-sensor)# path resources`
	`switch-1(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

	コマンドまたはアクション	目的
		ムメモリが「Not OK」に変更されたときに送信されます。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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、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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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、VLAN カウンタ、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 （380 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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(config-telemetry)# sensor-group 6`
	`switch-1(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。

ステップ 4	**vxlan environment**
	例：
	`switch-1(conf-tm-sensor)# vxlan environment`
	`switch-1(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。

ステップ 5	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 6	**ip address** *ip_addr* **port** *port*
	例：
	`switch-1(conf-tm-dest)# ip address 1.2.3.4 port 50004`
	`switch-1(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。

	コマンドまたはアクション	目的

ステップ 7	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 8	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 9	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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

手順

	コマンドまたはアクション	目的

ステップ 1	**show running-config-telemetry**
	例：
	`switch-1(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`
		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。

	コマンドまたはアクション	目的
	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)#	

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-1# 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-1#
```

show telemetry usability interface コマンドの出力を次に示します。

```
switch-1# 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-1#
```

show telemetry usability resources コマンドの例を次に示します。

```
switch-1# 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     :
[illegible]


  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-1#
```

show telemetry usability vxlan コマンドの例を次に示します。

```
switch-1# 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-1#
```

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

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）
		ス アドレス • 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 エントリーとダイナミック エントリー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。

ネイティブ データ ソース パスの注意事項と制限事項

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。
- Cisco NX-OSリリース 10.4(3)F 以降では、RIB ネイティブ パスのサンプル ベースのサブスクリプションまたは更新のみをサポートする、新しいクエリ条件が導入されています。

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 （398 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 query-condition** [**data=ephemeral** | **updates_only**]
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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(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

ステップ 5	**path rib query-condition** [**data=ephemeral** | **updates_only**]
	例：
	`nxosv2(conf-tm-sensor)# path rib`
	`nxosv2(conf-tm-sensor)#`
	例：
	`nxosv2(conf-tm-sensor)# path rib query condition data=ephemeral`
	`nxosv2(conf-tm-sensor)#`
	例：
	`nxosv2(conf-tm-sensor)# path rib query condition updates_only`
	`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。
		query condition data=ephemeral（オプション）：サンプル間隔 0 または 0 以外を設定できます。このサンプル間隔によって、接続先にルート情報が定期的に送信される頻度が決まります（構成されたサンプル間隔で）。
		query condition updates-only（オプション）：サンプル間隔 0 でのみサポートされます。このクエリ条件では、最初のスナップショットデータは送信されず、ルート情報の更新のみが接続先に送信されます。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(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。

ステップ 8	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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、rib イベントが発生したときに送信するかを決定します。
		（注）

	コマンドまたはアクション	目的
		サンプリング間隔に応じて、rib センサーパスはポーリング間隔に基づいてストリーミングします。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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 （398 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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(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。

ステップ 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。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	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-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact` `switch-1(conf-tm-dest)#`	

ステップ 8	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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 （398 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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(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。

ステップ 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。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP**	**gRPC** } **encoding** { **JSON**	**GPB**	**GPB-compact** } 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-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(conf-tm-dest)#`	

ステップ 8	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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 （398 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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(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。

ステップ 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。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`
	`switch-1(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。

ステップ 8	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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-1# 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-1#
```

MAC パスの統計情報の例：

```
switch-1# 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-1#
```

隣接パスの統計情報の例：

```
switch-1# 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-1#
```

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-1# 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-1#
```

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）

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 （398 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 query-condition** [**data=ephemeral** | **updates_only**]
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

手順

	コマンドまたはアクション	目的

ステップ 1	**configure terminal**
	例：
	`switch-1# configure terminal`
	`switch-1(config)#`
		コンフィギュレーション モードを入力します。

	コマンドまたはアクション	目的

ステップ 2	**telemetry**
	例：
	`switch-1(config)# telemetry`
	`switch-1(config-telemetry)#`
		テレメトリ機能の構成モードに入ります。

ステップ 3	**sensor-group** *sgrp_id*
	例：
	`switch-1(conf-tm-sub)# sensor-grp 6`
	`switch-1(conf-tm-sub)#`
		センサー グループを作成します。

ステップ 4	**data-source native**
	例：
	`switch-1(conf-tm-sensor)# data-source native`
	`switch-1(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。

ステップ 5	**path rib query-condition** [**data=ephemeral** | **updates_only**]
	例：
	`nxosv2(conf-tm-sensor)# path rib`
	`nxosv2(conf-tm-sensor)#`
	例：
	`nxosv2(conf-tm-sensor)# path rib query condition data=ephemeral`
	`nxosv2(conf-tm-sensor)#`
	例：
	`nxosv2(conf-tm-sensor)# path rib query condition updates_only`
	`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。
		query condition data=ephemeral（オプション）：サンプル間隔 0 または 0 以外を設定できます。このサンプル間隔によって、接続先にルート情報が定期的に送信される頻度が決まります（構成されたサンプル間隔で）。
		query condition updates-only（オプション）：サンプル間隔 0 でのみサポートされます。このクエリ条件では、最初のスナップショットデータは送信されず、ルート情報の更新のみが接続先に送信されます。

ステップ 6	**destination-group** *grp_id*
	例：
	`switch-1(conf-tm-sensor)# destination-group 33`
	`switch-1(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。

ステップ 7	**ip address** *ip_addr* **port** *port* **protocol** { **HTTP** | **gRPC** } **encoding** { **JSON** | **GPB** | **GPB-compact** }
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`
	`switch-1(conf-tm-dest)#`
	例：
	`switch-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`
	`switch-1(conf-tm-dest)#`
	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-1(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact` `switch-1(conf-tm-dest)#`	

ステップ 8	**subscription** *sub_id*
	例：
	`switch-1(conf-tm-dest)# subscription 33`
	`switch-1(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。

ステップ 9	**snsr-group** *sgrp_id* **sample-interval** *interval*
	例：
	`switch-1(conf-tm-sub)# snsr-grp 6 sample-interval 5000`
	`switch-1(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、rib イベントが発生したときに送信するかを決定します。
		（注）
		サンプリング間隔に応じて、rib センサーパスはポーリング間隔に基づいてストリーミングします。

ステップ 10	**dst-group** *dgrp_id*
	例：
	`switch-1(conf-tm-sub)# dst-grp 33`
	`switch-1(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。

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」で、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 • 重大度 • 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 25 章

OpenConfig YANG

ここでは、次の内容について説明します。

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。

CPU 使用率は、ユーザーとカーネルの使用率の値の合計です。この値は、「システム技術情報を表示」の出力に表示されるユーザーとカーネルの割合の合計とほぼ同じです。

最小、最大、平均、最小時間、および最大時間の値は、30 秒の移動ウィンドウで計算されます。この 30 秒のウィンドウでは、CPU 使用率が 5 秒ごとに計算されます。最小値、最大値、および平均値は、これらの 6 つのサンプルで計算されます。

過去 30 秒のウィンドウ内で最小値と最大値が発生する時刻は、エポック時間フォーマットで表示されます。

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-only oc-if:interface` リーフの `acl/interfaces/interface/interface-ref/state/interface`。
  - `read-only oc-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`

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 *22 : 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。

OpenConfig サポートの有効化

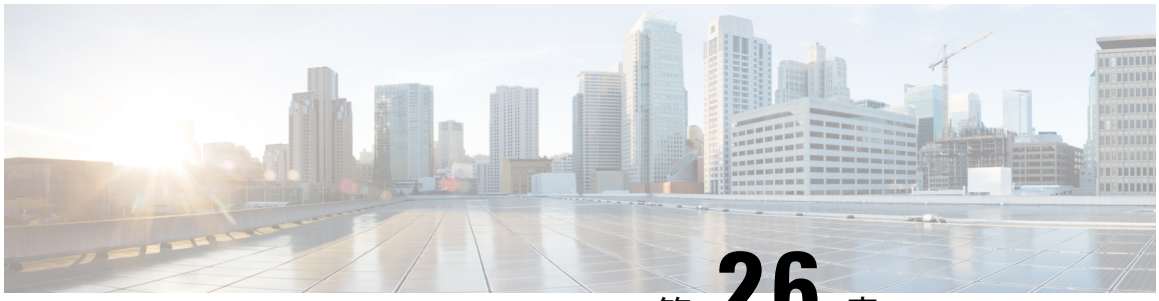
プログラマビリティ エージェント（NETCONF、RESTCONF、および gRPC）で OpenConfig サポートを有効または無効にするには、「[no] feature openconfig」を設定します。例：

```
switch(config)# feature netconf
switch(config)# feature restconf
switch(config)# feature grpc
switch(config)# feature openconfig
```

（注） 以前のリリースでは、mtx-openconfig-all RPM は個別にダウンロードしてインストールしていました。このメソッドは、10.2(2)F リリースでは廃止されています。

第 VI 部

XML 管理インターフェイス



第 26 章

XML 管理インターフェイス

ここでは、次の内容について説明します。

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

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 Secure Shell（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 440 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

NETCONF レイヤは次のとおりです：

***Table 23: 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>
運用者	<get-config>、<edit-config>
コンテンツ	show または 構成 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間の安全な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。

この NETCONF XML インスタンスの作成, on page 440 セクションでは、使用できる XML 構成インスタンスを定義する XML スキーマについて説明します。

XML 管理インターフェイスのライセンス要件

製品	製品
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 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。

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。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。

SSH および XML サーバ オプションの構成

デバイス上デフォルトで SSH サーバがイネーブル化されています。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 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。

SSH セッションを開始

次のようなコマンドを使用して、クライアント PC で SSHv2 セッションを開始できます。

```
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い場合は、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 メッセージを送信

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機能をアドバタイズする必要があります。XML サーバーは基本機能のみをサポートし、クライアントからの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>]]>]]>
```

XSD ファイルの取得

Procedure

ステップ 1 ブラウザから、次の URL にあるシスコ ソフトウェア ダウンロード サイトに移動します。

http://www.cisco.com/cisco/web/support/JP/loc/download/index.html

ソフトウェア ダウンロード ページが開きます。

ステップ 2 [製品の選択] リストで、**Switches > Data Center Switches >** *[プラットフォーム（platform）] > [モデル（model）]* を選択します。

ステップ 3 登録済みの Cisco ユーザーとしてまだログインしていない場合は、ここでログインするようにプロンプトします。

ステップ 4 ソフトウェア タイプの選択リストから、 **NX-OS XML Schema Definition.** を選択します。

ステップ 5 目的のリリースを見つけて **Download.** をクリックします

ステップ 6 リクエストをされたら、強力な暗号化ソフトウェアイメージをダウンロードするための資格を適用するには、次の手順を実行します。

Cisco エンド ユーザー使用許諾契約書が開きます。

ステップ 7 次の手順に従って、**Agree** をクリックしてファイルを PC にダウンロードします。

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、自動化されたメソッドを使用して XML ドキュメントを XML サーバに送信しますが、この方法で XML サーバへの SSH 接続を確認できます。

このメソッドの注意事項に従ってください：

- 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。XML サーバは hello 応答をすぐに送信するため、クライアント hello メッセージを送信した後、追加の応答は送信されません。
- 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

RPC タグおよび NETCONF 操作タグ内に XML デバイス要素を囲むことにより、NETCONF XML インスタンスを作成できます。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 rpc

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* リクエスト タグ*<rpc>* の例は、<rpc> 要素を必須の **message-id** 属性と共に示しています。message-id 属性は、<rpc-reply>リクエストと返信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 機能を使用する例です。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、「*XSD* ファイルの取得」に関するセクションを参照してください。

RPC タグ リクエスト

```
<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

以下は、構成リクエストの例です。

```
<?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>]]>]]>
```

__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 24: 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 451
commit	候補構成の現在のコンテンツに、実行構成を設定します。	NETCONF コミット インスタンス - 候補構成機能, on page 455
confirmed-commit	指定された時間に構成をコミットするためのパラメータを提供します。確認タイムアウト期間内にこの操作の後にコミット操作が行われない場合、構成は確認済みコミット操作の前の状態に戻ります。	NETCONF Confirmed-commit インスタンス , on page 455
copy-config	送信元構成データストアのコンテンツをターゲット データストアにコピーします。	NETCONF copy-config インスタンス, on page 451
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。この動作は、構成コマンドに使用します。	NETCONF edit-config インスタンス, on page 452 NETCONF rollback-on-error インスタンス , on page 456
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 440
get-config	構成の全体または一部を取得する	NETCONF get-config インスタンス, on page 453
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。close-session NETCONF 動作を参照してください。	NETCONF キルセッション インスタンス, on page 451

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することができます	NETCONF ロック インスタンス, on page 454
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 455
検証	構成をデバイスに適用する前に、構成候補のシンタックス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 456

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。「XSD ファイルの取得, on page 439」の項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。次の例では、ビルド NETCONF XML インスタンスの作成, on page 440 に使用された nfcli.xsd スキーマ ファイルの関連部分が示されています。

次の例は、XML デバイス タグを示しています。

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>
```

次の例は、サーバ ステータス デバイス タグを示しています。

サーバ 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>
```

次の例は、デバイス タグの応答を示しています。

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__」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 449 のセクションを参照してください。

<get> 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。

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>
]]>]]>
```

この応答から、このコンポーネントで操作を実行するタグを定義する名前空間は http://www.cisco.com/nxos:1.0:nfcli であり、nfcli.xsd ファイルを使用してこの機能の要求を作成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>end-tag の後に XML 終了文字シーケンスが続きます。

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 構成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

<exec-command> を通して送信される構成 CLI コマンド

```
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>]]>]]>
```

操作に対する応答は次のとおりです。

<exec-command>を通して送信された 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" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]>]]>
```

次の例は、<exec-command> データの取得に使用できます。

<exec-command> を通して送信される表示 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 interface brief</nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
```

以下は操作に対する反応です。

<exec-command> を通して送信された show 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>172.23.152.20</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>
]]>]]>
```

次の表に、操作タグの詳細な説明を示します。

Table 25: 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、show 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。複数の構成コマンドを異なる <cmd> タグを同じリクエストの一部として送信できます。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（Configuration CLI Commands Sent Through <exec-command>）]* 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>：すべての構成コマンドが正常に実行されました。
- <nf:rpc-error>：一部のコマンドが機能不全になりました。操作は最初のエラーで停止し、<nf:rpc-error> サブツリー は、機能不全になった構成に関する詳細情報を提供します。機能不全になったコマンドの前に実行された構成は、実行中の構成に適用されていることに注意してください。

次の例は、機能不全になった構成を示しています：

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 1.1.1.2/24 </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。

は <rpc-reply> を介して送信される show コマンドの結果 <cmd> show コマンドの XML 出力を含むタグ。

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。次の例は、同じインスタンスで組み合わされた構成と **show** コマンドを示しています。

構成コマンドと show 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>
]]>]]>
```

show コマンドは、それ自体で送信する必要があります<exec-command> 次の例に示すようなインスタンス。

送信された CLI コマンドを表示 <exec-command>

```
<?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。

ここでは、次の内容について説明します。

- RPC 応答タグ, on page 449
- データタグにカプセル化されたタグの解釈, on page 449

RPC 応答タグ

次の例は、RPC 応答タグ <rpc-reply> を表示しています。

RPC 応答エレメント

```
<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>]]>]]>
```

<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。

***Table 26: 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 リクエストに関連するデータは、<data> 要素。
<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

によってカプセル化されたデバイス タグ <data> タグには、リクエストとそれに続くレスポンスが含まれます。クライアント アプリケーションは、<readonly> タグ。次に、例を示します。

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>
]]>]]>
```

<__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。これらは要求で必要であり、CLI コマンドを表す XML タグに到達するためにスキーマ ファイルに従って追加する必要があります。

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

次の例は、セッション終了要求とそれに続くセッション終了応答を表示しています。

クローズ セッション リクエスト

```
<?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 セッションの応答

```
<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 キルセッション インスタンス

次の例は、キルセッション要求とそれに続く kill-session レスポンスを示しています。

キルセッション要求

```
<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>]]>]]>
```

キルセッション要求

```
<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>]]>]]>
```

NETCONF copy-config インスタンス

次の例は、copy-config 要求とそれに続く copy-config 応答を示しています。

copy-config リクエスト

```
<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>
```

copy-config 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

次の例は、NETCONF edit-config の使用を示しています。

edit-config 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>]]>]]>
```

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識別します。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

次の例は、実行中の構成からインターフェイス Ethernet0/0 の構成を削除する方法を示しています。

edit-config: 削除操作の要求

```
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 への応答: 削除操作

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

NETCONF get-config インスタンス

次の例は、NETCONF get-config の使用を示しています。

サブツリー全体を取得するための Get-config リクエスト

```
<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、NETCONF ロック操作の使用を示しています。

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>]]>]]>
```

ロック取得成功時の応答

```
<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 ロック解除操作の使用を示しています。

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、操作をコミットと返信をコミットを示しています。

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 Confirmed-commit インスタンス

次の例は、confirmed-commit 操作とconfirmed-commit 応答を表示しています。

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、エラー機能での NETCONF ロールバックの使用を示しています。文字列 urn:ietf:params:netconf:capability:rollback-on-error:1.0 は、機能を識別します。

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 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 リスポンス

```
<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 検証機能の使用を示しています。文字列 **urn:ietf:params:netconf:capability:validate:1.0** は機能を識別します。

リクエストの検証

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
```

```
<candidate/>
</source>
</validate>
</rpc>]]>]]>
```

リクエストの検証への応答

```
<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

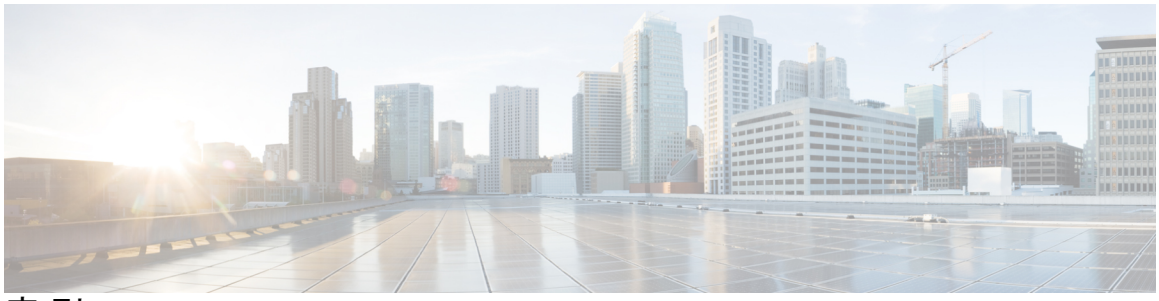
ここでは、XML 管理インターフェイスの実装に関する追加情報について説明します。

標準

標準	タイトル
この機能がサポートする新しい規格または変更された規格はありません。既存の規格のサポートは、この機能によって変更されていません。	—

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



索 引

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。