Cisco Nexus 3500 Series NX-OS Programmability Guide、リリース 10.3 （x）

初版：2022 年 8 月 19 日

最終更新：2022 年 9 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。

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS REFERENCED IN THIS DOCUMENTATION ARE SUBJECT TO CHANGE WITHOUT NOTICE. EXCEPT AS MAY OTHERWISE BE AGREED BY CISCO IN WRITING, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENTATION ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

The Cisco End User License Agreement and any supplemental license terms govern your use of any Cisco software, including this product documentation, and are located at: http://www.cisco.com/go/softwareterms.Cisco product warranty information is available at http://www.cisco.com/go/warranty. US Federal Communications Commission Notices are found here http://www.cisco.com/c/en/us/products/us-fcc-notice.html.

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

Any products and features described herein as in development or available at a future date remain in varying stages of development and will be offered on a when-and if-available basis. Any such product or feature roadmaps are subject to change at the sole discretion of Cisco and Cisco will have no liability for delay in the delivery or failure to deliver any products or feature roadmap items that may be set forth in this document.

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

The documentation set for this product strives to use bias-free language. For the purposes of this documentation set, bias-free is defined as language that does not imply discrimination based on age, disability, gender, racial identity, ethnic identity, sexual orientation, socioeconomic status, and intersectionality. Exceptions may be present in the documentation due to language that is hardcoded in the user interfaces of the product software, language used based on RFP documentation, or language that is used by a referenced third-party product.

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

© 2022 Cisco Systems, Inc. All rights reserved.

目 次

はじめに：

はじめに xi

対象読者 xi

表記法 xi

Cisco Nexus 3000 シリーズ スイッチの関連資料 xii

マニュアルに関するフィードバック xiii

通信、サービス、およびその他の情報 xiii

第 1 章

新規および変更情報 1

新規および変更情報 1

第 2 章

bash 3

Bash について 3

Bash へのアクセス 3

権限をルートにエスカレーションする 4

Bash コマンドの例 5

システム統計情報の表示 5

CLI からの Bash の実行 6

Bash からの Python の実行 6

第 3 章

ゲスト シェル 9

Guest Shell について 9

Guest Shell へのアクセス 10

ゲスト シェルに使用されるリソース 10

ゲストシェルの機能 11

Guest Shell の NX-OS CLI 11
Guest Shell でのネットワーク アクセス 12
ゲスト シェルでのブートフラッシュへのアクセス 15
Guest Shell の Python 15
Installing RPMs in the Guest Shell 15
のセキュリティ ポスチャ 17
[カーネル脆弱性パッチ（Kernel Vulnerability Patches）] 17
[ASLR および X-Space のサポート（ASLR and X-Space Support）] 17
ルートユーザーの制限 17
リソース管理 18
ゲスト ファイル システムのアクセス制限 18
セキュア IPC 18
ゲスト シェルの管理 18
Guest Shell の無効化 23
ゲスト シェルの破棄 23
Guest Shell の有効化 24
仮想サービスと Guest Shell 情報の検証 25
ゲスト シェルからのアプリケーションの永続的な起動 26
Guest Shell からアプリケーションを永続的に起動する手順 27
ゲスト シェルでのサンプル アプリケーション 27

第 4 章 **Python API** 29
Python API の概要 29
Python の使用 29
Cisco Python パッケージ 30
CLI コマンド API の使用 30
CLI からの Python インタープリタの呼び出し 32
表示フォーマット 32
非インタラクティブ Python 34
Embedded Event Manager でのスクリプトの実行 35
Cisco NX-OS ネットワーク インターフェイスとの Python 統合 36

Python による Cisco NX-OS セキュリティ 37
セキュリティとユーザー権限の例 37
Schedular でのスクリプトの実行例 37

第 5 章 tcl によるスクリプティング 39
Tcl について 39
tclsh コマンドのヘルプ 39
tclsh コマンドの履歴 40
tclsh のタブ補完 40
tclsh の CLI コマンド 40
tclsh コマンドの区切り 41
tcl 変数 41
tclquit 41
Tclsh セキュリティ 41
Tclsh コマンドの実行 42
Tclsh コマンドからの Cisco NX-OS モード間の移動 43
tcl の参照 44

第 6 章 Ansible 47
前提条件 47
アンシブルについて 47
Cisco Ansible モジュール 48

第 7 章 Puppet Agent 49
Puppet について 49
前提条件 50
Puppet エージェント NX-OS 環境 50
ciscopuppet モジュール 50

第 8 章 Cisco NX-OS でのシェフ クライアントの使用 53
シェフについて 53

前提条件 54
Chef クライアント NX-OS 環境 54
cisco-cookbook 55

第 9 章 **Cisco NX-OS での Docker の使用 57**
Cisco NX-OS での Docker について 57
注意事項と制約事項 58
Cisco NX-OS 内で Docker コンテナを設定するための前提条件 58
Docker デーモンの開始 59
自動的に起動するように Docker を構成する 59
Docker コンテナの開始: ホスト ネットワーク モデル 60
Docker コンテナの開始: ブリッジ型ネットワーク モデル 61
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 62
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 63
Cisco Nexus Platform Switches Switchover 時に Docker デーモンの永続性を有効にする 63
Docker ストレージ バックエンドのサイズ変更 64
Docker デーモンの停止 66
Docker コンテナ セキュリティ 67
ユーザー[名前空間（namespace）]の分離による Docker コンテナの保護 67
cgroup パーティションの移動 68
Docker のトラブルシューティング 69
Docker の起動が機能不全になる 69
ストレージが不足しているため、Docker が起動に失敗する 69
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） 70
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） 71
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 71
Docker ストレージ バックエンドのサイズ変更が失敗する 72
Docker コンテナがポートで着信トラフィックを受信しない 72
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 72

一般的なトラブルシューティングのヒント 73

第 10 章 NX-API 75

NX-APIについて 75

機能 NX-API 75

転送 76

メッセージ形式 76

セキュリティ 76

NX-API の使用 76

NX-API 管理コマンド 78

NX-API を使用したインタラクティブ コマンドの操作 80

NX-API リクエスト要素 80

NX-API 応答要素 80

JSON の概要（JavaScript オブジェクト表記） 81

CLI の実行 82

XML および JSON でサポートされたコマンド 82

XML および JSON 出力の例 82

第 11 章 NX-API 応答コード 85

NX-API 応答コードの表 85

第 12 章 NX-API 開発者サンドボックス 87

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 87

About the NX-API デベロッパー サンドボックス 87

注意事項と制約事項 88

メッセージ フォーマットとコマンド タイプの構成 88

デベロッパー サンドボックスを使用 90

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 90

第 13 章 N3500 での ABM および LM の XML サポート 95

N3500 での ABM および LM の XML サポート 95

第 14 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 103

XMLIN に関する情報 103

XMLIN のライセンス要件 103

XMLIN ツールのインストールおよび使用 104

show コマンド出力の XML への変換 105

XMLIN の構成例 105

第 15 章 モデル駆動型テレメトリ 109

テレメトリについて 109

テレメトリ コンポーネントとプロセス 109

テレメトリ プロセスの高可用性 111

テレメトリのライセンス要件 111

Telemetry のインストールとアップグレード 111

注意事項と制約事項 112

CLI を使用したテレメトリの構成 116

NX-OS CLI を使用したテレメトリの構成 116

CLI を使用したテレメトリの構成例 119

テレメトリの構成と統計情報の表示 122

テレメトリ ログとトレース情報の表示 127

NX-API を使用したテレメトリの構成 128

Configuring Telemetry Using the NX-API 128

NX-API を使用したテレメトリの構成例 136

DME のテレメトリ モデル 139

その他の参考資料 141

関連資料 141

第 16 章 OpenConfig YANG 143

OpenConfig YANG について 143

OpenConfig YANG のガイドラインと制限事項 143

BGP ルーティング インスタンスの削除について 148

OpenConfig サポートの有効化 150

第 17 章 XML 管理インターフェイス 151

XML 管理インターフェイスについて 151

XML 管理インターフェイスについて 151

NETCONF レイヤ 152

SSH xmlagent 152

XML 管理インターフェイスのライセンス要件 153

XML 管理インターフェイスを使用するための前提条件 153

XML 管理インターフェイスを使用 153

SSH および XML サーバ オプションの構成 153

SSH セッションを開始 154

Hello メッセージを送信 154

XSD ファイルの取得 155

XML ドキュメントを XML サーバに送信する 155

NETCONF XML インスタンスの作成 156

RPC リクエスト タグ rpc 156

NETCONF 動作タグ 157

デバイスタグ 159

拡張された NETCONF の操作 161

NETCONF 応答 165

RPC 応答タグ 165

データタグにカプセル化されたタグの解釈 165

サンプル XML インスタンスに関する情報 166

XML インスタンスの例 166

NETCONF クロース セッション インスタンス 167

NETCONF キルセッション インスタンス 167

NETCONF copy-config インスタンス 167

NETCONF edit-config インスタンス 168

NETCONF get-config インスタンス 169

NETCONF ロック インスタンス 170

NETCONF ロック解除インスタンス 171
NETCONF コミット インスタンス - 候補構成機能 171
NETCONF Confirmed-commit インスタンス 171
NETCONF rollback-on-error インスタンス 172
NETCONF 検証機能インスタンス 172
その他の参考資料 173

付録 A： ストリーミング テレメトリの送信元 175
ストリーミング テレメトリについて 175
テレメトリで利用可能なデータ 175

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 （xi ページ）
- 表記法 （xi ページ）
- Cisco Nexus 3000 シリーズ スイッチの関連資料 （xii ページ）
- マニュアルに関するフィードバック （xiii ページ）
- 通信、サービス、およびその他の情報 （xiii ページ）

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 3000 シリーズ スイッチの関連資料

Cisco Nexus 3000 シリーズ スイッチ全体のマニュアル セットは、次の URL にあります。

https://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。

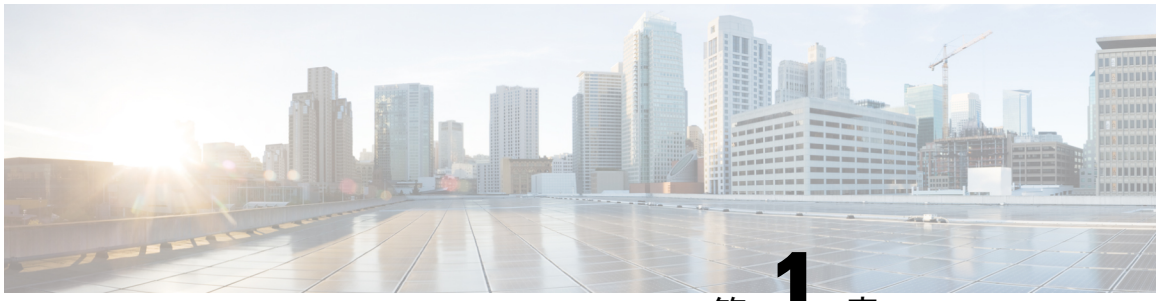
通信、サービス、およびその他の情報

- 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。
- 重要な技術によりビジネスに必要な影響を与えるには、シスコ サービス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、シスコ サポート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 Marketplace にアクセスしてください。
- 一般的なネットワーク、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 Bug Search Tool

Cisco バグ検索ツール（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

新規および変更情報

- 新規および変更情報 （1 ページ）

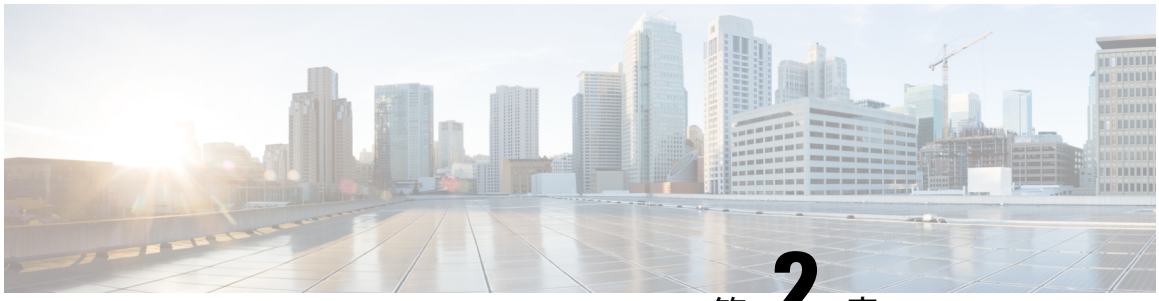
新規および変更情報

次の表は、*[Cisco Nexus 3500* シリーズ *NX-OS* プログラマビリティ ガイド リリース *10.3*（*x*）（*Cisco Nexus 3500 Series NX-OS Programmability Guide, Release 10.3(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。	10.3(1)F	該当なし

新規および変更情報



第 2 章

bash

- Bash について （3 ページ）
- Bash へのアクセス （3 ページ）
- 権限をルートにエスカレーションする （4 ページ）
- Bash コマンドの例 （5 ページ）

Bash について

Cisco NX-OS CLI に加えて、Cisco Nexus 3500 プラットフォーム 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。

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 bash
Linux# whoami
admin
Linux# pwd
/bootflash/home/admin
Linux#
```

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

以下は **run bash** < *[*コマンド（*command*）*]* > コマンドの例です。

```
run bash whoami
```

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。

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
Linux# sudo su root
```

```
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

Password:

Linux# whoami
root
Linux# 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

次に、システム統計情報の表示方法の例を示します：

```
switch# run bash
Linux# cat /proc/meminfo
MemTotal:      3795100 kB
MemFree:       1472680 kB
Buffers:           136 kB
Cached:        1100116 kB
ShmFS:         1100116 kB
Allowed:        948775 Pages
Free:           368170 Pages
Available:      371677 Pages
SwapCached:          0 kB
Active:        1198872 kB
Inactive:       789764 kB
SwapTotal:           0 kB
SwapFree:            0 kB
Dirty:               0 kB
Writeback:           0 kB
AnonPages:      888272 kB
Mapped:         144044 kB
Slab:           148836 kB
SReclaimable:    13892 kB
SUnreclaim:     134944 kB
PageTables:      28724 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
WritebackTmp:        0 kB
CommitLimit:   1897548 kB
Committed_AS: 19984932 kB
VmallocTotal: 34359738367 kB
VmallocUsed:    215620 kB
VmallocChunk: 34359522555 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
```

```
HugePages_Surp:       0
Hugepagesize:      2048 kB
DirectMap4k:      40960 kB
DirectMap2M:    4190208 kB
Linux#
```

CLI からの Bash の実行

次の例は、CLI から **run bash** <*command*> コマンドを使用して bash コマンドを実行する方法を示しています。

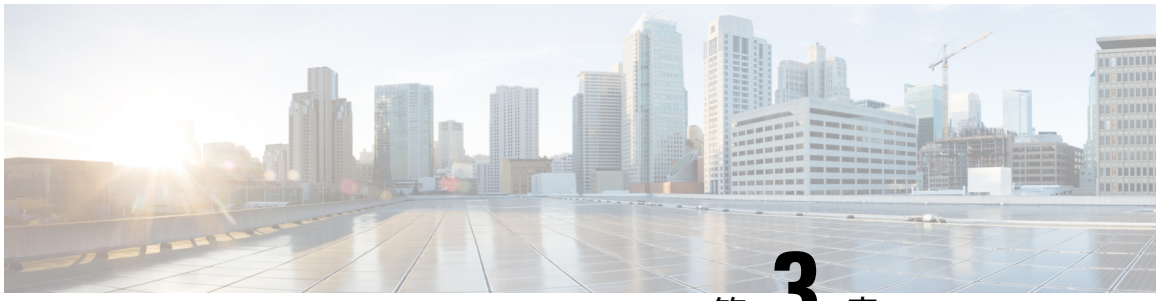
```
switch# run bash ps -el
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   497 select ?        00:00:08 init
5 S     0     2     0  0  75  -5 -     0 kthrea ?        00:00:00 kthreadd
1 S     0     3     2  0 -40   - -     0 migrat ?        00:00:00 migration/0
1 S     0     4     2  0  75  -5 -     0 ksofti ?        00:00:01 ksoftirqd/0
5 S     0     5     2  0  58   - -     0 watchd ?        00:00:00 watchdog/0
1 S     0     6     2  0 -40   - -     0 migrat ?        00:00:00 migration/1
1 S     0     7     2  0  75  -5 -     0 ksofti ?        00:00:00 ksoftirqd/1
5 S     0     8     2  0  58   - -     0 watchd ?        00:00:00 watchdog/1
1 S     0     9     2  0 -40   - -     0 migrat ?        00:00:00 migration/2
1 S     0    10     2  0  75  -5 -     0 ksofti ?        00:00:00 ksoftirqd/2
5 S     0    11     2  0  58   - -     0 watchd ?        00:00:00 watchdog/2
1 S     0    12     2  0 -40   - -     0 migrat ?        00:00:00 migration/3
1 S     0    13     2  0  75  -5 -     0 ksofti ?        00:00:00 ksoftirqd/3
5 S     0    14     2  0  58   - -     0 watchd ?        00:00:00 watchdog/3

...

4 S     0  8864     1  0  80   0 -  2249 wait   ttyS0    00:00:00 login
4 S  2002 28073  8864  0  80   0 - 69158 select ttyS0    00:00:00 vsh
4 R     0 28264  3782  0  80   0 - 54790 select ?        00:00:00 in.dcos-telnet
4 S     0 28265 28264  0  80   0 -  2247 wait   pts/0    00:00:00 login
4 S  2002 28266 28265  0  80   0 - 69175 wait   pts/0    00:00:00 vsh
1 S  2002 28413 28266  0  80   0 - 69175 wait   pts/0    00:00:00 vsh
0 R  2002 28414 28413  0  80   0 -   887 -      pts/0    00:00:00 ps
switch#
```

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
Linux# python
Python 2.7.5 (default, May 16 2014, 10:58:01)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Loaded cisco NxOS lib!
>>>
>>> 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: Thu Aug 21 23:32:25 2014

version 6.0(2)U4(1)

interface Ethernet1/3
  no switchport
  vrf member myvrf


>>>
```



第 3 章

ゲスト シェル

- Guest Shell について （9 ページ）
- Guest Shell へのアクセス （10 ページ）
- ゲスト シェルに使用されるリソース （10 ページ）
- ゲストシェルの機能 （11 ページ）
- のセキュリティ ポスチャ （17 ページ）
- ゲスト ファイル システムのアクセス制限 （18 ページ）
- ゲスト シェルの管理 （18 ページ）
- 仮想サービスと Guest Shell 情報の検証 （25 ページ）
- ゲスト シェルからのアプリケーションの永続的な起動 （26 ページ）
- Guest Shell からアプリケーションを永続的に起動する手順 （27 ページ）
- ゲスト シェルでのサンプル アプリケーション （27 ページ）

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。
- 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。

Guest Shell へのアクセス

Cisco NX-OS では、Guest Shell にはネットワーク管理者がアクセスできます。システムで自動的に有効になり、**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） Guest Shell で実行している場合、ネットワーク管理者レベルの権限があります。

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

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

ゲスト シェルに使用されるリソース

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

リソース	デフォルト	最小/最大
CPU	1 %	1/%

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

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

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

ゲストシェルの機能

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

ゲストシェルは CentOS 7 Linux 環境であり、この流通向けにビルドされたソフトウェア パッケージを、 dnf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 dnf 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） 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"
```

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。

Installing RPMs in the Guest Shell

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

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

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

Yum resolves the dependencies and installs all the required packages.

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

Dependencies Resolved

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

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

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

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

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

```
Complete!
```

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

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

のセキュリティ ポスチャ

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

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

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

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

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。

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。

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。

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、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

セキュア IPC

ゲスト シェルまたは仮想サービス内のアプリケーションは、Cisco onePK サービスを使用してホストとより統合できます。アプリケーションは、TIPC を介してホスト ネットワーク要素と通信します。さまざまなコンテナー内のアプリケーションは、TIPC を介して相互に通信することはできません。ホストとの通話のみが許可されます。これにより、1 つのコンテナの問題が、それが Cisco onePK サービスが実行されている場所であるとスプーフィングを防ぎます。コンテナ内のアプリケーションは、TIPC ポートでリッスンすることもできません。

既知の仮想サービスのみがホストと通信できるようにするために、各仮想サービスの一意の識別子は、有効化時に作成され、onePK 通信チャネルが確立されるときに検証されます。

システムは、個々の仮想サービス内のアプリケーションがホストにメッセージを送信できるレートも制限します。この動作により、不正な動作をするアプリケーションがメッセージを頻繁に送信して、ホストの通常の操作を妨げたり、同じホスト上の他の仮想サービスがホストと通信するのをブロックしたりすることが防止されます。

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

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

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

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

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

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

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

Guest Shell の無効化

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

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

例：

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

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

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

ゲスト シェルの破棄

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

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

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

```
switch# guestshell destroy

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

 'guestshell+'
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Successfully destroyed
virtual service 'guestshell +'

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

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

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

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

Guest Shell の有効化

この **guestshell enable** コマンドは、Guest Shell ソフトウェア パッケージから Guest Shell をインストールします。デフォルトでは、システムイメージに埋め込まれたパッケージがインストールに使用されます。Guest Shell が無効化されている場合は、このコマンドを使用して、Guest Shell を再アクティブ化することもできます。

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

例：

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

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

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

仮想サービスと Guest Shell 情報の検証

次のコマンドを使用して、仮想サービスとゲスト シェルの情報を検証できます。

コマンド	説明

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

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

コマンド	説明

show guestshell detail
 	
`switch# show guestshell detail`
`Virtual service guestshell+ detail`
`  State                 : Activated`
`  Package information`
`    Name                : guestshell.ova`
`    Path                : /isan/bin/guestshell.ova`
`    Application`
`      Name              : GuestShell`
`      Installed version : 2.2(0.2)`
`      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

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
```

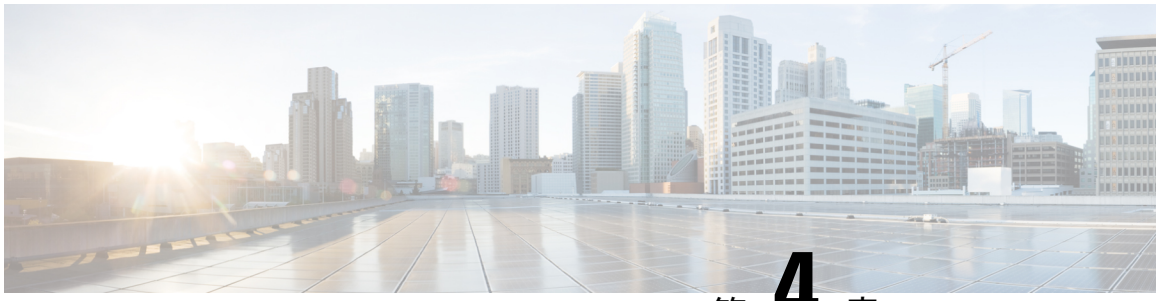
```
[root@guestshell system]# systemctl enable hello
[root@guestshell system]# systemctl status -l hello
hello.service - Trivial "hello world" example daemon
   Loaded: loaded (/usr/lib/systemd/system/hello.service; enabled)
   Active: active (running) since Sun 2015-09-27 18:31:51 UTC; 10s ago
 Main 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]#
```



第 4 章

Python API

- Python API の概要 （29 ページ）
- Python の使用 （29 ページ）

Python API の概要

Cisco Nexus 3500 プラットフォーム 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 v2.7.11 をサポートし、ゲスト シェルで使用できます。

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。

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）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。Python は Bash シェルからアクセスできます。

Python インタープリタは Cisco NX-OS ソフトウェアで使用できます。

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
```

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。キャリッジ リターンは '\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。

```
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
>>>
```

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：

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> r = cli('where detail') ; print r
  mode:
  username:            admin
  vdc:                 EOR-1
  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])
...
                kern_uptm_secs = 21
                kick_file_name = bootflash:///nxos.9.2.1.bin.S246
                    rr_service = None
                     module_id = Supervisor Module
                   kick_tmstmp = 07/11/2018 00:01:44
                bios_cmpl_time = 05/17/2018
                bootflash_size = 20971520
             kickstart_ver_str = 9.2(1)
                kick_cmpl_time = 7/9/2018 9:00:00
                    chassis_id = Nexus9000 C9504 (4 Slot) Chassis
                 proc_board_id = SAL171211LX
                        memory = 16077872
                  manufacturer = Cisco Systems, Inc.
                kern_uptm_mins = 26
                  bios_ver_str = 05.31
                      cpu_name = Intel(R) Xeon(R) CPU D-1528 @ 1.90GHz
                 kern_uptm_hrs = 2
                       rr_usecs = 816550
                   rr_sys_ver = 9.2(1)
                     rr_reason = Reset Requested by CLI command reload
                      rr_ctime = Wed Jul 11 20:44:39 2018
                   header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2018, 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.
                     host_name = switch
                      mem_type = kB
                kern_uptm_days = 0
>>>
```

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 Nexus 3500 プラットフォーム スイッチは、Python スクリプトを実行するための送信元 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:deltaCounters.py
#!/isan/bin/python

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:deltaCounters.py Ethernet1/1 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
============================================================
           0      791        1        0   212739        0
============================================================
1          0        0        0        0       26        0
2          0        0        0        0       27        0
3          0        1        0        0       54        0
4          0        1        0        0       55        0
5          0        1        0        0       81        0
switch#
```

次の例は、**source** コマンドが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、**source** コマンドがパイプ演算子（「|」）の後に続くことも表示しています。

```
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 3500 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 3500 プラットフォーム 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
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 urllib2
>>> from cisco.vrf import *
>>> set_global_vrf('management')
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
>>> print page.read()
Hello Cisco Nexus 9000

>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set global vrf in module cisco.vrf:

set global vrf(vrf)
    Sets the global vrf. Any new sockets that are created (using socket.socket)
    will automatically get set to this vrf (including sockets used by other
```

```
    python libraries).

    Arguments:
        vrf: VRF name (string) or the VRF ID (int).

    Returns: Nothing

>>>
```

Python による Cisco NX-OS セキュリティ

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

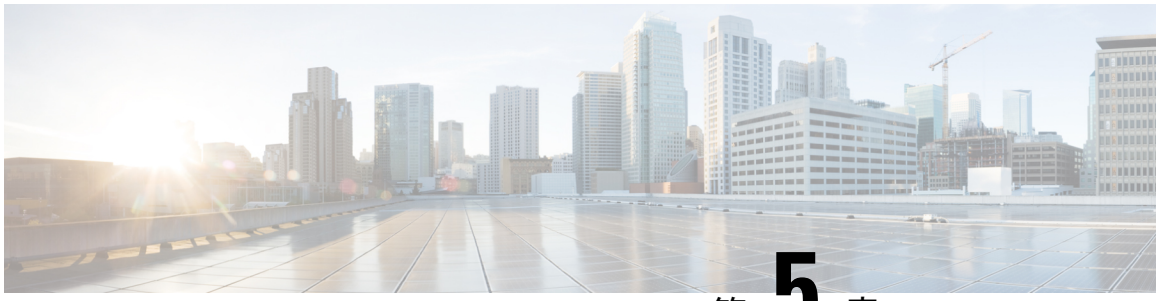
Cisco NX-OS network-admin または dev-ops ロールに関連付けられているすべてのユーザは、特権ユーザです。カスタム ロールで Python へのアクセスが許可されているユーザーは、非特権ユーザーと見なされます。非特権ユーザは、ファイル システム、ゲスト シェル、Bash コマンドなどの Cisco NX-OS 情報技術へのアクセスが制限されています。特権ユーザは、Cisco NX-OS のすべての情報技術へのアクセスが向上します。

セキュリティとユーザー権限の例

-

Schedular でのスクリプトの実行例

-



第 5 章

tcl によるスクリプティング

- Tcl について （39 ページ）
- Tclsh コマンドの実行 （42 ページ）
- Tclsh コマンドからの Cisco NX-OS モード間の移動 （43 ページ）
- tcl の参照 （44 ページ）

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。

tclsh コマンドのヘルプ

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

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

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

switch-tcl#
```

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

tclsh コマンドの履歴

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

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

tclsh のタブ補完

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

tclsh の CLI コマンド

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

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

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

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

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

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

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

tclsh コマンドの区切り

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

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

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

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

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

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

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

tcl 変数

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

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

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

tclquit

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

Tclsh セキュリティ

tcl シェルは、Cisco NX-OS システムの特定の部分への不正アクセスを防止するために、サンドボックスで実行されます。システムは、無限ループや過剰なメモリ使用率などのイベントを検

出するために、tcl シェルによって使用されている CPU、メモリ、ファイルなどのシステム リソースをモニタリングします。

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

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

Tclsh コマンドの実行

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

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

手順の概要

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

手順の詳細

	コマンドまたはアクション	目的

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

	コマンドまたはアクション	目的

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

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。

	コマンドまたはアクション	目的

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：

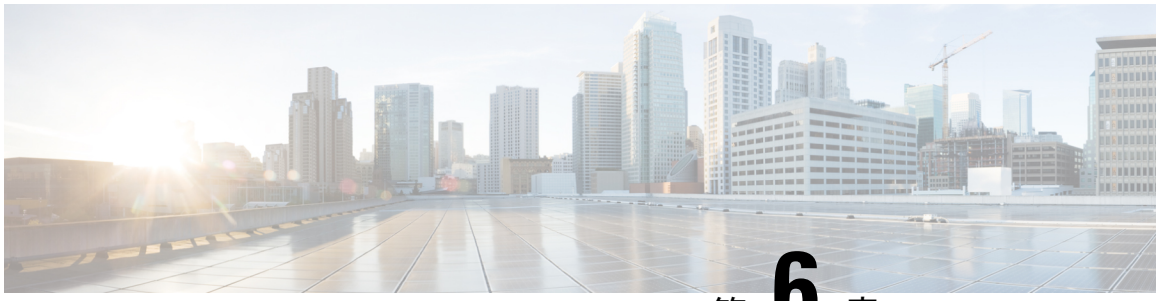
```
switch# tclsh
switch-tcl# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# ?
  description  Enter description of maximum 80 characters
  inherit      Inherit a port-profile
  ip           Configure IP 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 6 章

Ansible

- 前提条件 （47 ページ）
- アンシブルについて （47 ページ）
- Cisco Ansible モジュール （48 ページ）

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/intro_installation.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。

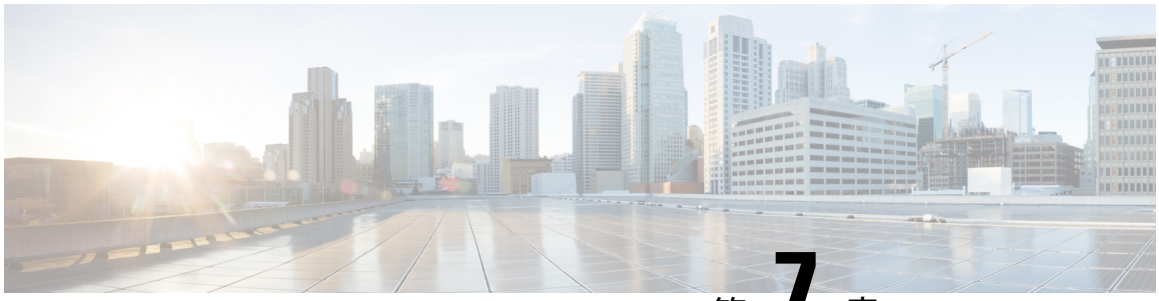
詳細については、Ansible から入手できます。

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 開発者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 7 章

Puppet Agent

この章は、次の項で構成されています。

- Puppet について （49 ページ）
- 前提条件 （50 ページ）
- Puppet エージェント NX-OS 環境 （50 ページ）
- ciscopuppet モジュール （50 ページ）

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/products/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 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

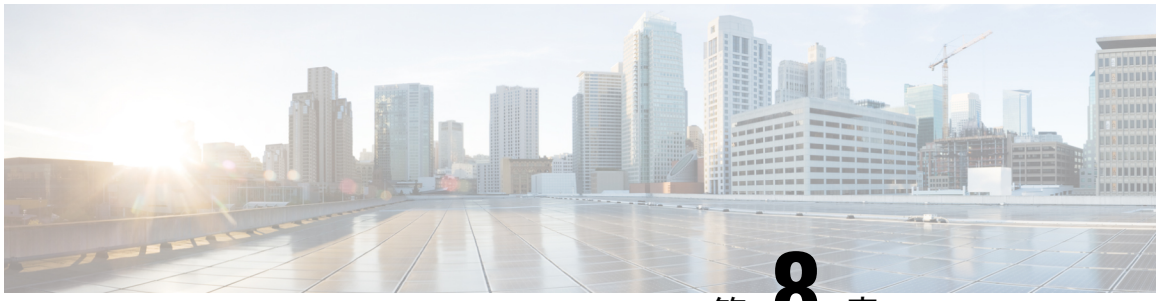
ciscopuppet モジュールは、Cisco が開発したオープン ソース ソフトウェア モジュールです。これは、Puppet マニフェストの抽象技術情報構成と、Cisco NX-OS オペレーティング システムおよびプラットフォームの特定の実装の詳細との間のインターフェイスとなります。このモ

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://docs.puppetlabs.com/puppet/latest/reference/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 8 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 （53 ページ）
- 前提条件 （54 ページ）
- Chef クライアント NX-OS 環境 （54 ページ）
- cisco-cookbook （55 ページ）

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。これは、インフラストラクチャのサイズに関係なく、物理、仮想、またはクラウドの場所にサーバーとアプリケーションを導入する、システムおよびクラウドインフラストラクチャの自動化フレームワークです。各組織は、1 つ以上のワークステーション、単一サーバー、Chef クライアントが設定されていて、維持されているすべてのノードで構成されます。各ノードの設定方法について Chef クライアントに指示するために、クックブックとレシピが使用されます。すべてのノードにインストールされている Chef クライアントが、実際の設定を行います。

Chef クックブックは、設定とポリシーの配布の基本単位です。クックブックではシナリオを定義します。また、そのシナリオをサポートするために必要なすべての内容（ライブラリ、レシピ、ファイルなど）が含まれています。Chef レシピは、デバイスの状態を設定するためのプロパティ定義の集合です。これらのプロパティ状態の確認および設定の詳細は抽象化されているため、レシピは複数のオペレーティングシステムまたはプラットフォームで使用できます。レシピは、通常、構成時の設定を定義するために使用されますが、ソフトウェアパッケージのインストール、ファイルのコピー、およびサービスの開始などにも使用できます。

次のリファレンスは、Chef からの詳細情報を提供します。

トピック	リンク
Chef ホーム	https://www.chef.io
Chef の概要	https://docs.chef.io/chef_overview.html

トピック	リンク
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 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 NX-OS リリース 6.1（2）I3（4）
- 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 クライアント ソフトウェアは、Cisco Nexus スイッチにインストールする必要があります。お客様は、Cisco Nexus スイッチが提供する Linux 環境のいずれかに Chef クライアントをインストールできます。

- Bash Shell — これは、Cisco NX-OS の基礎となるネイティブの WindRiver Linux 環境です。
- Guest Shell — これは、CentOS を実行する安全な Linux コンテナ環境です。その利点は、ホストから切り離された安全でオープンな実行環境です。

両方の使用例のワークフローは似ています。

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、およびセットアップに関する段階的な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（ネイティブ）	クライアント RPM の最新情報は、ここで入手できます。
Chef クライアント (Guest Shell、CentOs7)	クライアント RPM の最新情報は、ここで入手できます。

トピック	リンク
Chef クライアント：Cisco Nexus プラットフォームでのインストールとセットアップ（手動セットアップ）	cisco-cookbook::README-install-agent.md
Chef クライアント：Cisco Nexus プラットフォームでのインストールとセットアップ（Chefプロビジョナを使用した自動インストール）	cisco-cookbook::README-chef-provisioning.md

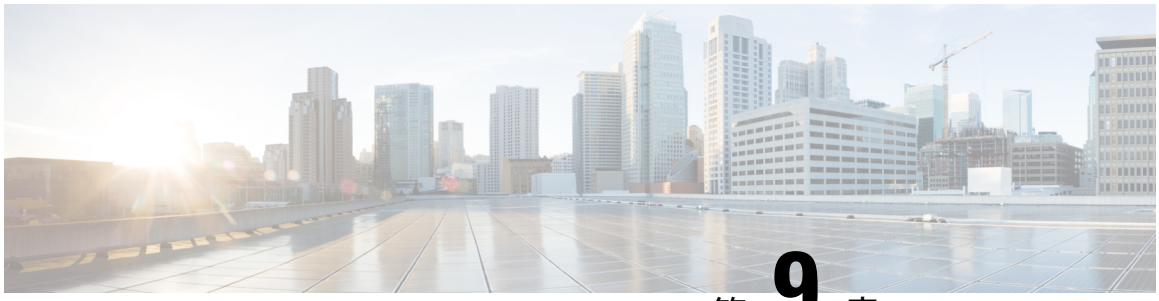
cisco-cookbook

cisco-cookbook は、Chef レシピの抽象リソース構成と、Cisco NX-OS オペレーティング システムおよび Cisco Nexus スイッチの特定の実装の詳細との間の、シスコが開発したオープン ソース インターフェイスです。このクックブックは Chef Server にインストールされ、Cisco Nexus スイッチでの Chef Client の適切な動作に必要です。

cisco-cookbook は、Chef Supermarket にあります。

次のドキュメントには、cisco-cookbook および一般的なクックブックのインストール手順の詳細が記載されています。

トピック	リンク
cisco-cookbook の場所	https://supermarket.chef.io/cookbooks/cisco-cookbook
リソースの種類のカタログ	https://github.com/cisco/cisco-network-chef-cookbook#resource-by-tech
cisco-cookbook: ソース コード リポジトリ	https://github.com/cisco/cisco-network-chef-cookbook
cisco-cookbook: セットアップと使用法	cisco-cookbook::README.md
Chef Supermarket	https://supermarket.chef.io
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 9 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 （57 ページ）
- 注意事項と制約事項 （58 ページ）
- Cisco NX-OS 内で Docker コンテナを設定するための前提条件 （58 ページ）
- Docker デーモンの開始 （59 ページ）
- 自動的に起動するように Docker を構成する （59 ページ）
- Docker コンテナの開始: ホスト ネットワーク モデル （60 ページ）
- Docker コンテナの開始: ブリッジ型ネットワーク モデル （61 ページ）
- 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 （62 ページ）
- 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 （63 ページ）
- Cisco Nexus Platform Switches Switchover 時に Docker デーモンの永続性を有効にする （63 ページ）
- Docker ストレージ バックエンドのサイズ変更 （64 ページ）
- Docker デーモンの停止 （66 ページ）
- Docker コンテナ セキュリティ （67 ページ）
- Docker のトラブルシューティング （69 ページ）

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 1.13.1 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。例：

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

- スイッチのクロックが正しく設定されていることを確認してください。そうしないと、次のエラー メッセージが表示される場合があります：

```
x509: certificate has expired or is not yet valid
```

- ドメイン名とネーム サーバがネットワークに対して適切に構成されていること、および `/etc/resolv.conf` ファイルに反映されていることを確認します：

```
switch# conf t
     Enter configuration commands, one per line. End with CNTL/Z.
     switch(config)# vrf context management
     switch(config-vrf)# ip domain-name ?
     WORD Enter the default domain (Max Size 64)

     switch(config-vrf)# ip name-server ?
     A.B.C.D Enter an IPv4 address
     A:B::C:D Enter an IPv6 address

root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
```

```
nameserver 171.70.168.183 #bleed
root@switch#
```

Docker デーモンの開始

初めて Docker デーモンを開始すると、固定サイズのバックエンド ストレージ スペースがブートフラッシュの dockerpart と呼ばれるファイルに切り出され、次に /var/lib/docker にマウントされます。必要に応じて、Docker デーモンを初めて開始する前に /etc/sysconfig/docker を編集して、この領域のデフォルト サイズを調整できます。後で説明するように、必要に応じてこのストレージ スペースのサイズを変更することもできます。

Docker デーモンを開始するには：

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。

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。

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。

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 （67 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。

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。

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。

Cisco Nexus Platform Switches Switchover 時に Docker デーモンの永続性を有効にする

Docker デーモンと実行中のコンテナの両方を、個別のブートフラッシュ パーティションを持つ 2 つの個別の物理スーパーバイザ間のスイッチオーバーで持続させることができます。ただし、Cisco Nexus スイッチの場合、両方のスーパーバイザのブートフラッシュ パーティションは物理的に分離されています。したがって、スイッチオーバーを実行する前に、`dockerpart` ファイルをスタンバイ スーパーバイザに手動でコピーする必要があります。

ステップ 1 Bash を読み込みしてスーパーユーザーになります。

```
switch# run bash sudo su -
```

ステップ 2 スイッチオーバー後にコンテナが自動的に再起動されるように、`--restart without-stopped` オプションを使用してコンテナを起動します。

次の例では、Alpine コンテナを開始し、明示的に停止するか、Docker を再起動しない限り、常に再起動するように構成します。

```
root@switch# docker run -dit --restart unless-stopped alpine
root@n9k-2#
```

Docker コンテナは切り替え中に中断（再起動）されるため、継続的に実行されないことに注意してください。

ステップ 3 スイッチオーバーを開始する前に、`chkconfig` ユーティリティを使用して Docker サービスを永続化します。

```
root@switch# chkconfig --add docker
root@n9k-2#
```

ステップ 4 Docker バックエンド ストレージ パーティションを現用系からスタンバイ スーパーバイザ ブートフラッシュにコピーします。

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

root@switch# cp /bootflash/dockerpart /bootflash_sup-remote/

root@switch# service docker start
```

Docker ストレージ バックエンドのサイズ変更

Docker デーモンを起動または使用した後、必要に応じて Docker バックエンド ストレージ スペースのサイズを増やすことができます。

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。

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。

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。

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：（Problem :）]システムが Docker ハブからイメージをプルすることに失敗し、次のようなエラー メッセージが表示されます。

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: x509: certificate has
 expired or is not yet valid
```

[考えられる原因：（Possible Cause:）]システム クロックが正しく設定されていない可能性があります。

[解決策：（Solution:）]クロックが正しく設定されているかどうかを確認します。

```
root@n9k-2# sh clock
15:57:48.963 EST Thu Apr 25 2002
Time source is Hardware Calendar
```

必要に応じて、時計をリセットします：

```
root@n9k-2# clock set hh:mm:ss { day month | month day } year
```

例：

```
root@n9k-2# clock set 14:12:00 10 feb 2018
```

Docker Hub からのイメージのプルの失敗（クライアント タイムアウト エラー メッセージ）

問題：システムが Docker ハブからイメージをプルできず、次のようなエラー メッセージが表示されます。

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request
canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

考えられる原因：プロキシまたは DNS 設定が正しく設定されていない可能性があります。

解決策：プロキシ設定を確認し、必要に応じて修正してから、Docker デーモンを再起動します。

```
root@switch# cat /etc/sysconfig/docker | grep proxy
#export http_proxy=http://proxy.esl.cisco.com:8080
#export https_proxy=http://proxy.esl.cisco.com:8080
root@switch# service docker [re]start
```

DNS 設定を確認し、必要に応じて修正してから、Docker デーモンを再起動します。

```
root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch# # conf t
     Enter configuration commands, one per line. End with CNTL/Z.
     switch(config)# vrf context management
     switch(config-vrf)# ip domain-name ?
     WORD Enter the default domain (Max Size 64)

     switch(config-vrf)# ip name-server ?
     A.B.C.D Enter an IPv4 address
     A:B::C:D Enter an IPv6 address
root@switch# service docker [re]start
```

スイッチのリロードまたはスイッチオーバーで Docker デーモンまたはコンテナが実行されない

問題：スイッチのリロードまたはスイッチオーバーを実行した後、Docker デーモンまたはコンテナーが実行されません。

考えられる原因：Docker デーモンが、スイッチのリロードまたはスイッチオーバーで持続するように構成されていない可能性があります。

解決策：Docker デーモンが `chkconfig` コマンドを使用してスイッチのリロードまたはスイッチオーバーで持続するように構成されていることを確認してから、`--restart unless-stopped` オプションを使用して必要な Docker コンテナを開始します。たとえば、Alpine コンテナを開始するには：

```
root@switch# chkconfig --add docker
root@switch#
```

```
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch# docker run -dit --restart unless-stopped alpine
```

Docker ストレージ バックエンドのサイズ変更が失敗する

問題：Docker バックエンド ストレージのサイズを変更しようとして失敗しました。

考えられる原因：ゲスト シェルが無効になっていない可能性があります。

解決策：次のコマンドを使用して、ゲスト シェルが無効になっているかどうかを確認します。

```
root@switch# losetup -a | grep dockerpart
root@n9k-2#
```

ゲスト シェルが無効になっている場合、コマンドは出力を表示しません。

必要に応じて、次のコマンドを入力してゲスト シェルを無効にします。

```
switch# guestshell disable
```

それでもDockerバックエンドストレージのサイズを変更できない場合は、`/bootflash/dockerpart` を削除し、`/etc/sysconfig/docker` の `[small_]large_dockerstrg` を調整してから、Docker を再度起動して、必要なサイズの新しい Docker パーティションを取得します。

Docker コンテナがポートで着信トラフィックを受信しない

問題：Docker コンテナがポートで着信トラフィックを受信しません。

考えられる原因：Docker コンテナが kstack ポートではなく netstack ポートを使用している可能性があります。

解決策：Docker コンテナによって使用されるエフェメラル ポートが kstack の範囲内にあることを確認します。そうしないと、着信パケットがサービスのために netstack に送信され、ドロップされる可能性があります。

```
switch# show socket local-port-range
Kstack local port range (15001 - 58000)
Netstack local port range (58001 - 63535) and nat port range (63536 - 65535)
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# sockets local-port-range <start_port> <end_port>
switch# run bash sudo su -
root@switch# cat /proc/sys/net/ipv4/ip_local_port_range
15001   58000
root@switch#
```

Docker コンテナでデータ ポートと / または管理インターフェイスを表示できません

問題： Docker コンテナにデータ ポートまたは管理インターフェースが表示されません。

解決方法：

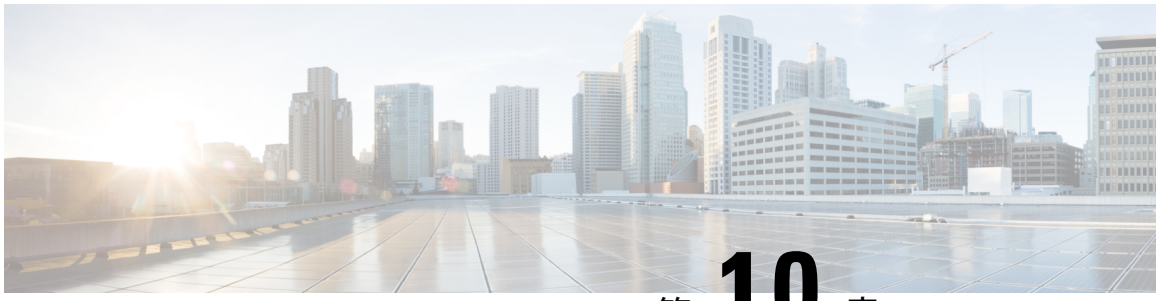
- `-v /var/run/netns:/var/run/netns:ro,rslave --network host` オプションを使用して、すべてのホスト名前空間がマップされたホスト ネットワーク名前空間で Docker コンテナが開始されていることを確認します。
- コンテナに入ると、デフォルトで管理ネットワークの名前空間に入ります。`ip netns` ユーティリティを使用して、データ ポート インターフェイスを持つデフォルト（`init`）ネットワーク名前空間に移動できます。`ip netns` ユーティリティは、yum、apk などを使用してコンテナにインストールする必要がある場合があります。

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 10 章

NX-API

- NX-APIについて （75 ページ）
- NX-API の使用 （76 ページ）
- XML および JSON でサポートされたコマンド （82 ページ）

NX-APIについて

機能 NX-API

- サンドボックスを介してデバイスにアクセスするには、機能 NX-API を有効にする必要があります。
- デバイス上の | json は、内部的に Python スクリプトを使用して出力を生成します。
- NX-API は、ipv4 を介して http / https のいずれかで有効にすることができます。

```
BLR-VXLAN-NPT-CR-179# show nxapi
 nxapi enabled
 HTTP Listen on port 80
 HTTPS Listen on port 443
 BLR-VXLAN-NPT-CR-179#
```

- NX-API は、サードパーティの NGINX プロセスを内部的に生成しています。このプロセスは、ハンドラ受信 / 送信 / http 要求 の処理 / 応答：

```
nxapi certificate {httpscrt |httpskey}
nxapi certificate enable
```

- NX-API 証明書は https で有効にできます
- nginx が動作するデフォルトのポートは、http / https がそれぞれ 80 / 443 です。次の CLI コマンドを使用して変更することもできます：

```
nxapi {http|https} port port-number
```

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。

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またはJSON-RPC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の使用

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サンドボックスを設定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 198.51.100.1/24
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 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 コールの管理インターフェイスの使用

NX-API コールには管理インターフェイスを使用することをお勧めします。

NX-API の非管理インターフェイスとカスタム ポートを使用する場合、NX-API トラフィックが API トラフィックを好ましくない処理する可能性のあるデフォルトの copp エントリにヒットしないように、CoPP ポリシーにエントリを作成する必要があります。

（注） NX-API トラフィックには管理インターフェイスを使用することをお勧めします。それが不可能で、カスタム ポートが使用されている場合は、「copp-http」クラスを更新して、カスタム NX-API ポートを含める必要があります。

次の例のポート 9443 は、NX-API トラフィックに使用されています。

このポートは、copp-system-acl-http ACL に追加され、copp-http クラスの下で一致できるようになり、100 pps ポリシングになります。（特定の環境では、これを増やす必要がある場合があります。）

```
!
ip access-list copp-system-acl-http
  10 permit tcp any any eq www
  20 permit tcp any any eq 443
  30 permit tcp any any eq 9443 <---------
!
class-map type control-plane match-any copp-http
  match access-group name copp-system-acl-http
!
!
policy-map type control-plane copp-system-policy
  class copp-http
    police pps 100
!
```

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 *4 : NX-API* 管理コマンド

NX-API 管理コマンド	説明
feature nxapi	NX-API を有効化します。
no feature nxapi	NX-API を無効化します。
nxapi {**http** ｜ **https**} **port** *port*	ポートを指定します。
no nxapi {**http** ｜ **https**}	HTTP / HTTPS を無効化します。
show nxapi	ポート情報を表示します。
nxapi certificate {httpscrt certfile ｜ **httpskey keyfile**} *filename*	次のアップロードを指定します： • httpscrt が指定されている場合の HTTPS 証明書。 • httpskey が指定されている場合の HTTPS キー。 HTTPS 証明書の例： **nxapi certificate httpscrt certfile bootflash:cert.crt** HTTPS キーの例： **nxapi certificate httpskey keyfile bootflash:privkey.key**
nxapi certificate enable	証明書を有効化します。

以下は、HTTPS 証明書の正常なアップロードの例です：

```
switch(config)# nxapi certificate httpscrt certfile certificate.crt
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

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)#
```

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 httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
Nginx certificate invalid.
switch(config)#
```

これは、キーファイルが暗号化されている場合に発生する可能性があります。その場合、キーファイルをインストールする前に復号化する必要があります。次の例に示すように、Guest Shell に移動してキー ファイルを復号化する必要がある場合があります。

```
switch(config)# guestshell
[b3456@guestshell ~]$
[b3456@guestshell bootflash]$ /bin/openssl rsa -in certfilename.net.pem -out clearkey.pem

Enter pass phrase for certfilename.net.pem:
writing RSA key
[b3456@guestshell bootflash]$
[b3456@guestshell bootflash]$ exit
switch(config)#
```

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
Upload done. Please enable. Note cert and key must match.
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 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 *5: 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。

NX-API 応答要素	説明
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。
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。

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/CLI 実行は現在、Cisco Nexus 3500 プラットフォーム スイッチでサポートされています。

（注） NX-API/JSON 機能は、Cisco Nexus 3500 プラットフォーム スイッチで使用できるようになりました。

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/JSON-RPC/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

Show_Command | json

コード例

```
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 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

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、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。

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)#
```

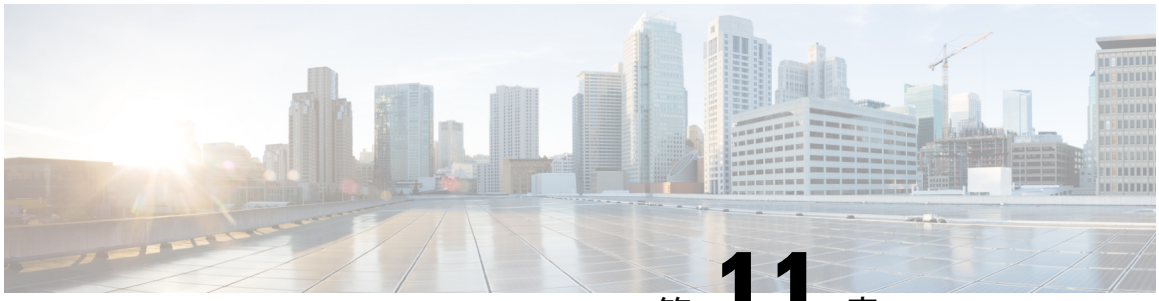
この例では、JSON 形式でスイッチ上に LLDP タイマーを表示する例を示します。

```
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)#
```



第 11 章

NX-API 応答コード

• NX-API 応答コードの表 （85 ページ）

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 *6 : 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。
IN_MSG_ERR	400	要求メッセージが無効です。
NO_INPUT_CMD_ERR	400	入力コマンドがありません。

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。
BACKEND_ERR	500	バックエンド処理エラー。
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。
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。
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。
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。
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
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。
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。

第 12 章

NX-API 開発者サンドボックス

• NX-API 開発者サンドボックス: 9.2（2）より前の NX-OS リリース （87 ページ）

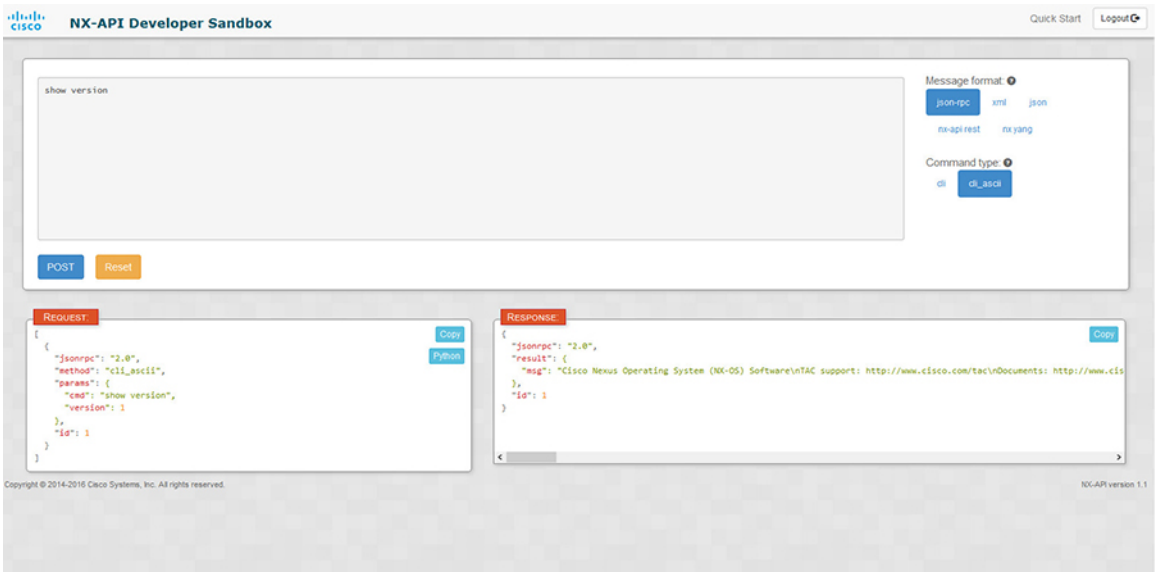
NX-API 開発者サンドボックス:9.2（2）より前の NX-OS リリース

About the NX-API デベロッパー サンドボックス

NX-API Developer Sandbox は、スイッチでホストされる Web フォームです。NX-OS CLI コマンドを同等の XML または JSON ペイロード。

図に示すように、Web フォームは 3 つのペイン（コマンド（上部ペイン）、要求、および応答）を持つ 1 つの画面です。

図 *1*:リクエストと出力応答の例を含む *NX-API* デベロッパー サンドボックス



コマンド ペインのコントロールを使用すると、サポートされている API のメッセージフォーマット（NX-API REST など）とコマンド タイプ（XML や JSON など）を選択できます。使用可能なコマンドタイプオプションは、選択したメッセージフォーマットによって異なります。

コマンド ペインに 1 つ以上の CLI コマンドを入力するか貼り付けると、Web フォームはコマンドを API ペイロードに変換し、構成エラーをチェックし、結果のペイロードを要求ペインに表示します。次に、コマンドペインの POST ボタンを使用して、ペイロードをサンドボックスからスイッチに直接送信することを選択した場合、応答ペインに API 応答が表示されます。

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 *7: 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 *8:* コマンド タイプ

メッセージ形式	コマンド タイプ
json-rpc	• cli — show または構成コマンド • cli_ascii — show または構成コマンド、フォーマットせずに出力
xml	• cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。 • cli_show_ascii — コマンドを表示、フォーマットせずに出力 • cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。 • bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。 （注） スイッチで bash シェルを有効にする必要があります。
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

メッセージ形式	コマンド タイプ
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。

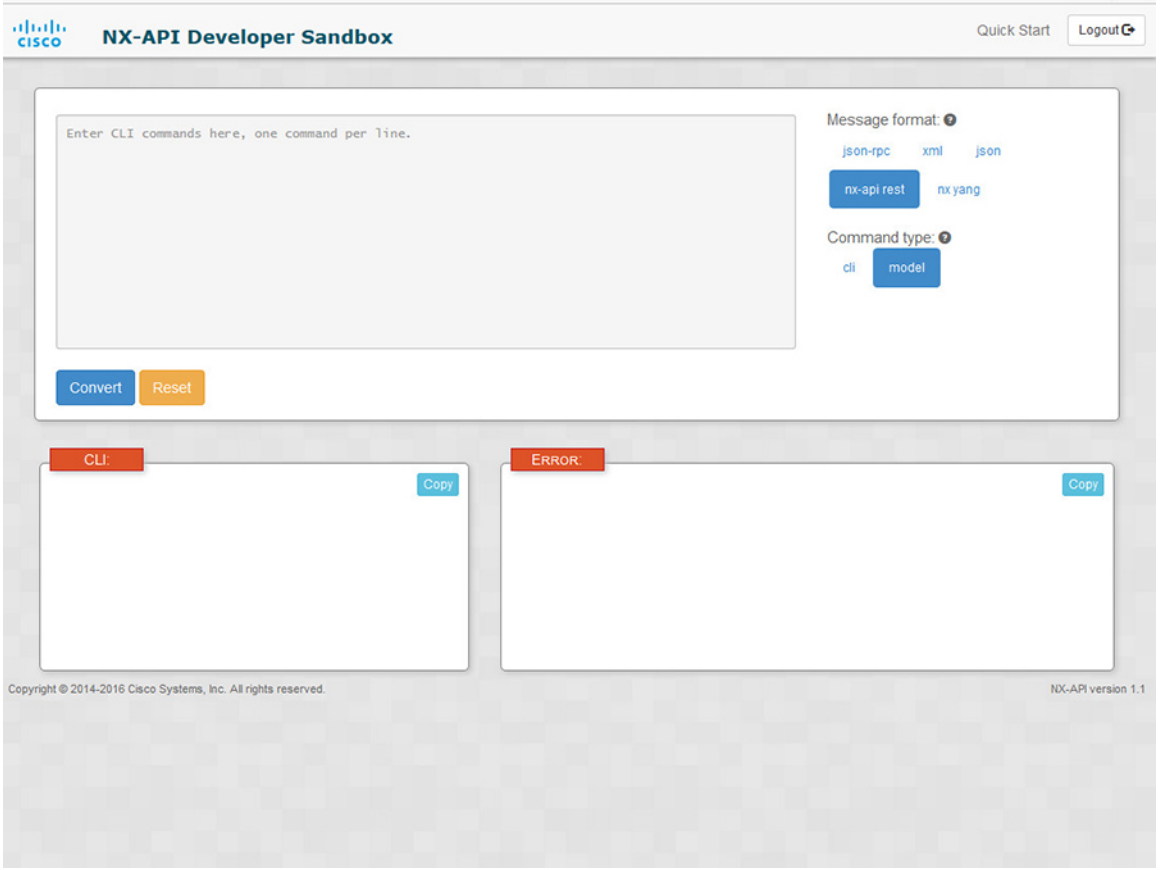
構成コマンドはサポートされていません。

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 （88 ページ）を参照してください。

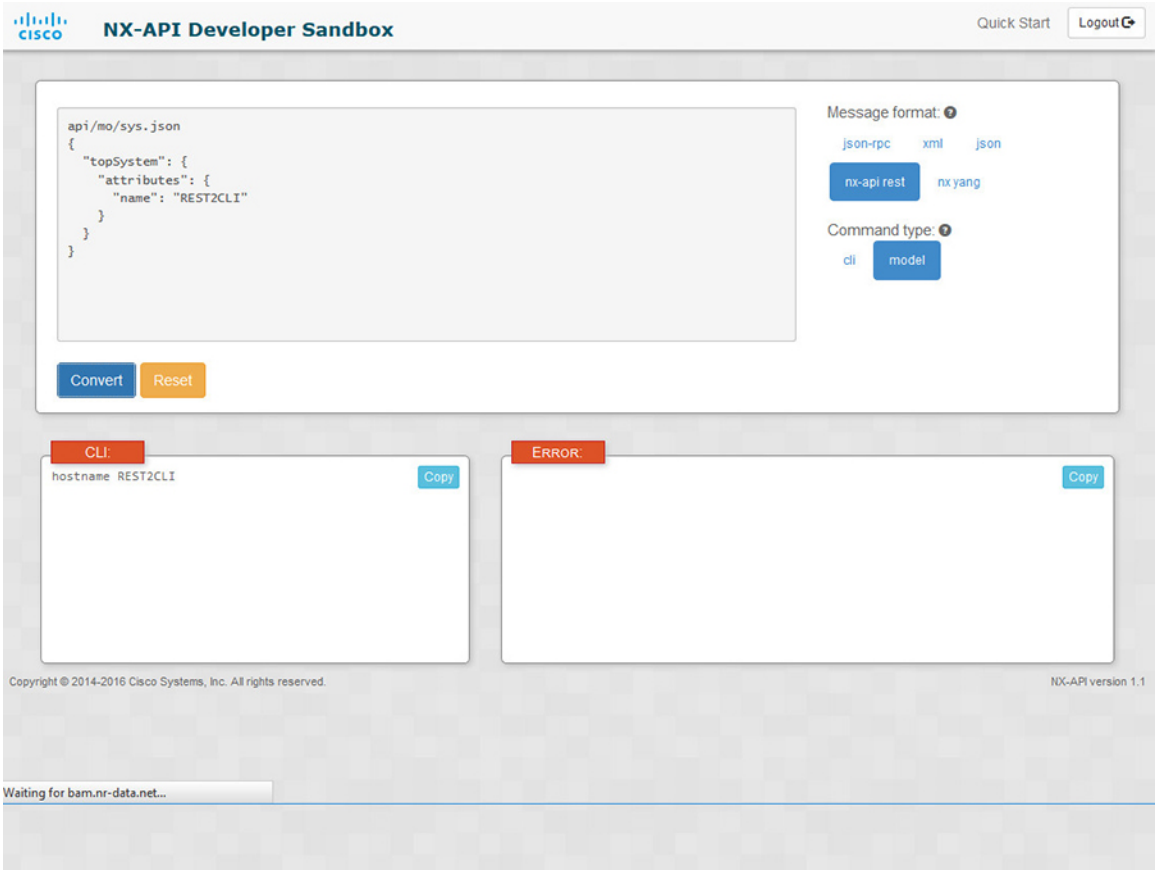
ステップ 2　上部ペインのテキスト エントリ ボックスに、NX-OS CLI 構成コマンドを 1 行に 1 つずつ入力するか貼り付けます。

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。

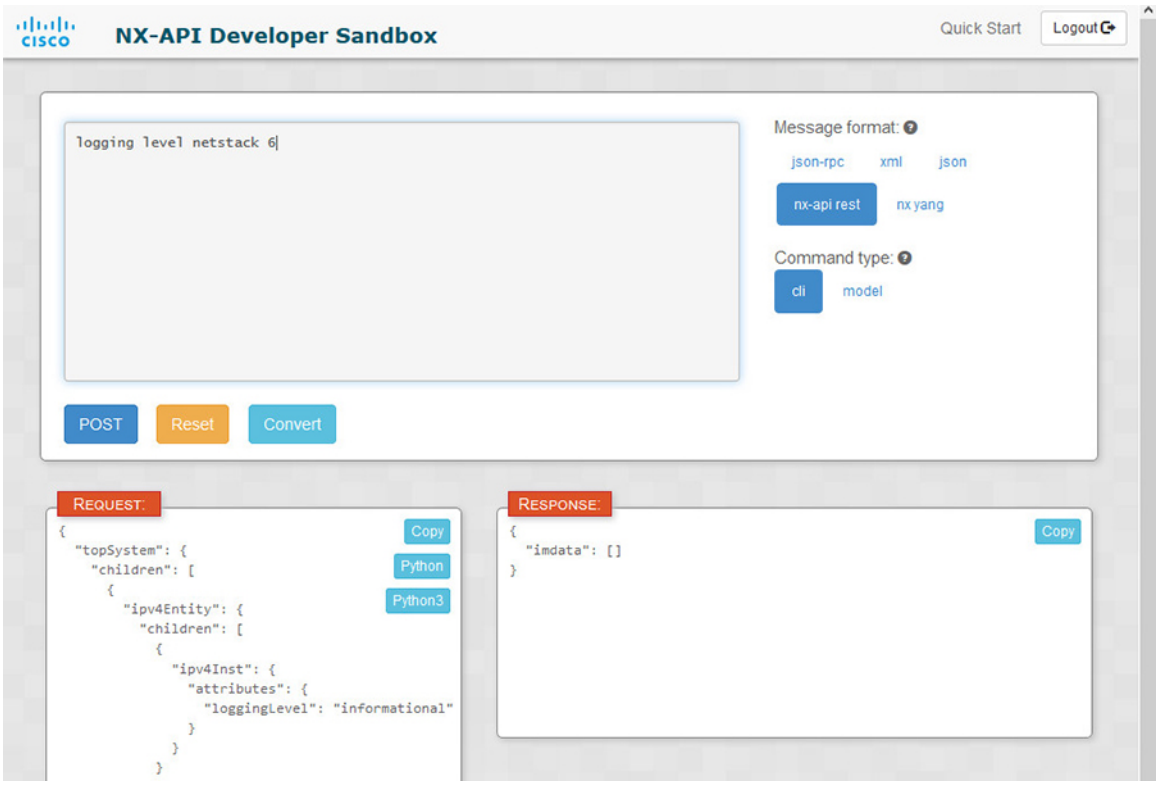
CLI コマンドに構成エラーが含まれていない場合、ペイロードは **[要求（Request）]** ペインに表示されます。エラーが存在する場合は、説明のエラー メッセージが **[応答（Response）]** ペインに表示されます。



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。

第 13 章

N3500 での ABM および LM の XML サポート

• N3500 での ABM および LM の XML サポート （95 ページ）

N3500 での ABM および LM の XML サポート

次のコマンドは、ABM および LM の XML 出力を表示します：

[ハードウェア プロファイル バッファ モニタ サンプリングを表示します（show hardware profile buffer monitor sampling）]

CLI ：

```
MTC-8(config)# show hardware profile buffer monitor sampling

Sampling CLI issued at: 05/25/2016 04:18:56

Sampling interval: 200
```

XML ：

```
MTC-8(config)# show hardware profile buffer monitor sampling | 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:mtc_usd_cli">

 <nf:data>

  <show>

   <hardware>

    <profile>

     <buffer>

      <monitor>

       <__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
```

N3500 での ABM および LM の XML サポート

```
          <__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
           <__readonly__>
            <cmd_name>Sampling CLI</cmd_name>
            <cmd_issue_time>05/25/2016 04:19:12</cmd_issue_time>
            <TABLE_sampling>
             <ROW_sampling>
              <sampling_interval>200</sampling_interval>
             </ROW_sampling>
            </TABLE_sampling>
           </__readonly__>
          </__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
         </__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
        </monitor>
       </buffer>
      </profile>
     </hardware>
    </show>
   </nf:data>
</nf:rpc-reply>
]]>]]>
```

[ハードウェア プロファイル バッファ モニタの詳細を表示します | xml（show hardware profile buffer monitor detail | xml）]

XML :

```
<show>
   <hardware>
    <profile>
     <buffer>
      <monitor>
       <__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
        <__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
         <__readonly__>
          <cmd_name>Detail CLI</cmd_name>
          <cmd_issue_time>10/02/2001 10:58:58</cmd_issue_time>
          <TABLE_detail_entry>
           <ROW_detail_entry>
            <detail_util_name>Ethernet1/1</detail_util_name>
            <detail_util_state>Active</detail_util_state>
           </ROW_detail_entry>
           <ROW_detail_entry>
            <time_stamp>10/02/2001 10:58:58</time_stamp>
            <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
            <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
```

N3500 での ABM および LM の XML サポート

```
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:57</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:56</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:55</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
```

N3500 での ABM および LM の XML サポート

```
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
<ROW_detail_entry>
 <time_stamp>10/02/2001 10:58:54</time_stamp>
 <__XML__DIGIT384k_util>0</__XML__DIGIT384k_util>
 <__XML__DIGIT768k_util>0</__XML__DIGIT768k_util>
 <__XML__DIGIT1152k_util>0</__XML__DIGIT1152k_util>
 <__XML__DIGIT1536k_util>0</__XML__DIGIT1536k_util>
 <__XML__DIGIT1920k_util>0</__XML__DIGIT1920k_util>
 <__XML__DIGIT2304k_util>0</__XML__DIGIT2304k_util>
 <__XML__DIGIT2688k_util>0</__XML__DIGIT2688k_util>
 <__XML__DIGIT3072k_util>0</__XML__DIGIT3072k_util>
 <__XML__DIGIT3456k_util>0</__XML__DIGIT3456k_util>
 <__XML__DIGIT3840k_util>0</__XML__DIGIT3840k_util>
 <__XML__DIGIT4224k_util>0</__XML__DIGIT4224k_util>
 <__XML__DIGIT4608k_util>0</__XML__DIGIT4608k_util>
 <__XML__DIGIT4992k_util>0</__XML__DIGIT4992k_util>
 <__XML__DIGIT5376k_util>0</__XML__DIGIT5376k_util>
 <__XML__DIGIT5760k_util>0</__XML__DIGIT5760k_util>
 <__XML__DIGIT6144k_util>0</__XML__DIGIT6144k_util>
</ROW_detail_entry>
```

[ハードウェア プロファイル バッファ モニタの概要を表示します（show hardware profile buffer monitor brief）]

XML :

```
show hardware profile buffer monitor brief | 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:mtc_usd_cli">
<nf:data>
<show>
<hardware>
<profile>
<buffer>
<monitor>
<__XML__BLK_Cmd_show_hardware_profile_buffer_monitor_summary>
<__XML__OPT_Cmd_show_hardware_profile_buffer_monitor___readonly__>
<__readonly__>
<cmd_name>Brief CLI</cmd_name>
<cmd_issue_time>03/21/2016 09:06:38</cmd_issue_time>
<TABLE_ucst_hdr>
<ROW_ucst_hdr>
<ucst_hdr_util_name>Buffer Block 1</ucst_hdr_util_name>
<ucst_hdr_1sec_util>0KB</ucst_hdr_1sec_util>
<ucst_hdr_5sec_util>0KB</ucst_hdr_5sec_util>
<ucst_hdr_60sec_util>N/A</ucst_hdr_60sec_util>
<ucst_hdr_5min_util>N/A</ucst_hdr_5min_util>
<ucst_hdr_1hr_util>N/A</ucst_hdr_1hr_util>
<ucst_hdr_total_buffer>Total Shared Buffer Available = 5397 Kbytes
</ucst_hdr_total_buffer>
<ucst_hdr_class_threshold>Class Threshold Limit = 5130 Kbytes
</ucst_hdr_class_threshold>
</ROW_ucst_hdr>
</TABLE_ucst_hdr>
<TABLE_brief_entry>
<ROW_brief_entry>
```

N3500 での ABM および LM の XML サポート

```
<brief_util_name>Ethernet1/45</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>

<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/46</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/47</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/48</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/21</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/22</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/23</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>

<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/24</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/9</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/10</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/11</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
```

N3500 での ABM および LM の XML サポート

```
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
<brief_util_name>Ethernet1/12</brief_util_name>
<brief_1sec_util>0KB</brief_1sec_util>
<brief_5sec_util>0KB</brief_5sec_util>
<brief_60sec_util>N/A</brief_60sec_util>
<brief_5min_util>N/A</brief_5min_util>
<brief_1hr_util>N/A</brief_1hr_util>
```

[ハードウェア プロファイルの遅延モニタ サンプリングを表示します（show hardware profile latency monitor sampling）]

CLI

```
MTC-8(config)# show hardware profile latency monitor sampling

Sampling CLI issued at: 05/25/2016 04:19:54

Sampling interval: 20
```

XML

```
MTC-8(config)# show hardware profile latency monitor sampling | 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:mtc_usd_cli">

 <nf:data>

  <show>

   <hardware>

    <profile>

     <latency>

      <monitor>

       <__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>

        <__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>

         <__readonly__>

          <cmd_issue_time>05/25/2016 04:20:06</cmd_issue_time>

          <device_instance>0</device_instance>

          <TABLE_sampling>

           <ROW_sampling>

            <sampling_interval>20</sampling_interval>

           </ROW_sampling>

          </TABLE_sampling>

         </__readonly__>
```

N3500 での ABM および LM の XML サポート

```
        </__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>
       </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
      </monitor>
     </latency>
    </profile>
   </hardware>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
```

[ハードウェア プロファイルの遅延モニタのしきい値を表示します（how hardware profile latency monitor threshold）]

CLI

```
MTC-8(config)# show hardware profile latency monitor threshold

Sampling CLI issued at: 05/25/2016 04:20:53

Threshold Avg: 3000

Threshold Max: 300000
```

XML

```
MTC-8(config)# show hardware profile latency monitor threshold | 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:mtc_usd_cli">

 <nf:data>

  <show>

   <hardware>

    <profile>

     <latency>

      <monitor>

       <__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>

        <__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>

         <__readonly__>

          <cmd_issue_time>05/25/2016 04:21:04</cmd_issue_time>

          <device_instance>0</device_instance>
```

N3500 での ABM および LM の XML サポート

```
            <TABLE_threshold>
             <ROW_threshold>
              <threshold_avg>3000</threshold_avg>
              <threshold_max>300000</threshold_max>
             </ROW_threshold>
            </TABLE_threshold>
           </__readonly__>
          </__XML__OPT_Cmd_show_hardware_profile_latency_monitor___readonly__>
         </__XML__BLK_Cmd_show_hardware_profile_latency_monitor_summary>
        </monitor>
       </latency>
      </profile>
     </hardware>
    </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
```

第 14 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 （103 ページ）
- XMLIN のライセンス要件 （103 ページ）
- XMLIN ツールのインストールおよび使用 （104 ページ）
- show コマンド出力の XML への変換 （105 ページ）
- XMLIN の構成例 （105 ページ）

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 *9 : 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

	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。
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。

	コマンドまたはアクション	目的
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

	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>
     ]]>]]>
```

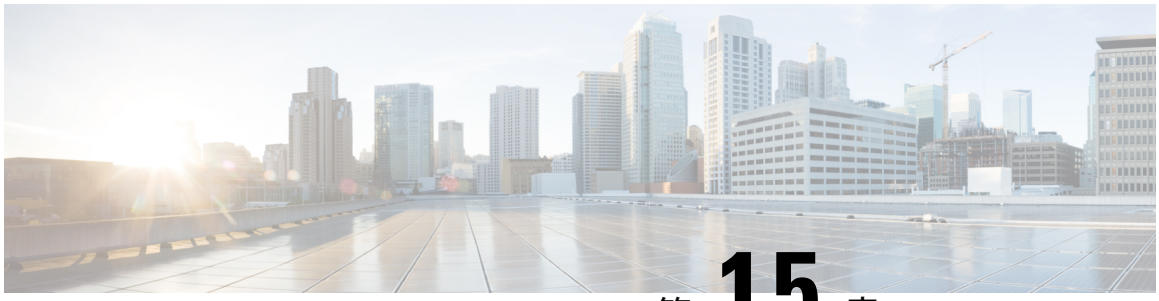
次の例は、**show** コマンドに対して XML インスタンスを生成する前に現在の XML 構成を終了するために **end** コマンドを入力する方法を表示しています。

```
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 15 章

モデル駆動型テレメトリ

- テレメトリについて （109 ページ）
- テレメトリのライセンス要件 （111 ページ）
- Telemetry のインストールとアップグレード （111 ページ）
- 注意事項と制約事項 （112 ページ）
- CLI を使用したテレメトリの構成 （116 ページ）
- NX-API を使用したテレメトリの構成 （128 ページ）
- その他の参考資料 （141 ページ）

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、4MB 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。）

  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 }
```

  num は 1 ～ 4095 の数値です。

  UDP テレメトリは、次のヘッダーと共に送信されます。

```
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 データを正常に処理します。

  - 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、データの復号化に使用するデコーダー（JSON または GPB）を決定します。または、
  - 1 つのデコーダー（JSON または GPB）が必要で、もう 1 つは必要ない場合は、ヘッダーを削除します。

- **テレメトリ レシーバー**：テレメトリ レシーバーは、テレメトリ データを保存するリモート管理システムです。

GPB エンコーダーは、汎用キーと値の形式でデータを格納します。また、データを GPB 形式に変換するには、コンパイルされた .proto ファイル形式のメタデータが GPB エンコーダに必要です。

データ ストリームを正しく受信して復号化するには、受信側で暗号化とトランスポート サービスを記述した .proto ファイルが必要です。エンコードは、バイナリ ストリームをキー値の文字列のペアにデコードします。

GPB エンコーディングと gRPC トランスポートを記述する telemetry .proto ファイルは、Cisco の GitLab で入手できます。 https://github.com/CiscoDevNet/nx-telemetry-proto

テレメトリ プロセスの高可用性

テレメトリ プロセスの高可用性は、次の動作でサポートされています。

- **[システムのリロード（System Reload）]** — システムのリロード中に、テレメトリ構成とストリーミング サービスが復元されます。
- **[スーパーバイザ フェールオーバー（Supervisor Failover）]** — テレメトリはホット スタンバイではありませんが、テレメトリ構成とストリーミング サービスは、新しい現用系スーパーバイザが実行されているときに復元されます。
- **[プロセスの再起動（Process Restart）]** — なんらかの理由でテレメトリ プロセスがフリーズまたは再起動した場合、テレメトリが再開されると、構成およびストリーミング サービスが復元されます。

テレメトリのライセンス要件

製品	ライセンス要件
Cisco NX-OS	テレメトリにはライセンスは必要ありません。ライセンス パッケージに含まれていない機能は Cisco NX-OS イメージにバンドルされており、無料で提供されます。NX-OS ライセンス方式の詳細については、『*Cisco NX-OS Licensing Guide*』を参照してください。

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。

次の例のように：

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：

yum 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。

```
yum 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。

注意事項と制約事項

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。

- テレメトリは、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 リリースでサポートされています。
- DME データ収集、NX-API データ 送信元、Google リモート プロシージャ コール（gRPC）トランスポート経由の Google プロトコル バッファ（GPB）エンコーディング、HTTP 経由の JSON エンコーディングがサポートされています。
- 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。

- 最大 5 つの遠隔管理受信者（接続先）がサポートされます。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 受信者がダウンした場合、その他の受信者にはデータ フローが中断したことが表示されます。障害が発生したレシーバーは再起動する必要があります。次に、接続先グループの下にある障害受信者の IP アドレスを構成解除してから再構成することにより、スイッチとの新しい接続を開始します。
- 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。

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ときのベストプラクティスとして、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するために、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。

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 レシーバーが 20 のエラーを送信した場合、スイッチ クライアントは gRPC レシーバーへの接続を無効化します。gRPC レシーバーを有効にするには、接続先グループの下のレシーバーの 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している。
- 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。

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。つまり、show vlan 識別子 100、show vlan 識別子 101 などです。代わりに、CLI 範囲オプションを使用してください。つまり、パフォーマンスを向上させるために可能な限り、VLAN 識別子 100-110、204 を表示します。

サマリー / カウンターのみが必要な場合は、show コマンド出力全体をダンプして、データ収集に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。

ノード識別子のサポート

NX-OS リリース 9.3.1 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構成できます。デフォルトではホスト名が使用されますが、ノード受信者のサポートにより、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 ID 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表示しています。

```
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
```

テレメトリの VRF サポート

テレメトリ VRF 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 パネルのポートを介して送信され、SSH / 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。

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"
            }
          }
        }
      ]
    }
  }
```

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。

始める前に

スイッチは、Cisco NX-OS リリース 9.2(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. **configure terminal**
2. **feature telemetry**
3. **feature nxapi**
4. **nxapi use-vrf management**
5. **telemetry**
6. （任意） **certificate** *certificate_path host_URL*
7. **sensor-group** *sgrp_id*
8. **path** *sensor_path* **depth 0** [**filter-condition** *filter*]
9. **destination-group** *dgrp_id*
10. （任意） **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
11. *ip_version* **address** *ip_address* **port** *portnum*
12. **subscription** *sub_id*
13. **snsr-grp** *sgrp_id* **sample-interval** *interval*
14. **dst-grp** *dgrp_id*

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**configure terminal** 例：	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
	switch# **configure terminal** switch(config)#	
ステップ 2	**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 3	**feature nxapi**	nxapi を有効にします。
ステップ 4	**nxapi use-vrf management**	nxapi 通信に使用する VRF 管理を有効にします。
ステップ 5	**telemetry** 例： switch(config)# **telemetry** switch(config-telemetry)#	ストリーミング テレメトリの構成モードに入ります。
ステップ 6	（任意） **certificate** *certificate_path host_URL* 例： switch(config-telemetry)# **certificate /bootflash/server.key localhost**	既存の SSL/TLS 証明書を使用します。
ステップ 7	**sensor-group** *sgrp_id* 例： switch(config-telemetry)# **sensor-group 100** switch(conf-tm-sensor)#	ID *srgp_id* を持つセンサー グループを作成し、センサー グループ構成モードを開始します。 現在は、数字の ID 値のみサポートされています。センサー グループでは、テレメトリ レポートのモニタリング対象ノードを定義します。
ステップ 8	**path** *sensor_path* **depth 0** [**filter-condition** *filter*] 例： • 次のコマンドは、NX-API ではなく DME 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て、状態ベースのフィルタリングを使用して、**operSt**が**上**から**下**に変化したときにのみトリガーし、MOがいつ変化したかを通知しません。 switch(conf-tm-sensor)# **path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))** • 次のコマンドは、DME ではなく NX-API に適用されます。 switch(conf-tm-sensor)# **path "show interface" depth 0**	センサー グループにセンサー パスを追加します。 • **depth** 設定では、センサー パスの取得レベルを指定します。**0 - 32**、**unbounded** の深さ設定がサポートされています。 （注） **depth 0** デフォルトの深さです。 NX-API ベースのセンサー パスは、**depth 0** のみを使用できます。 イベント収集のパスがサブスクライブされている場合、深さは 0 とバウンドなしのみをサポートします。その他の値は 0 として扱われます。 • オプションの **filter-condition** パラメータを指定して、イベントベースのサブスクリプション用の特定のフィルタを作成できます。 状態ベースのフィルタリングの場合、フィルタは、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

	コマンドまたはアクション	目的
		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 が残っている間に DN のプロパティが変更されたとき（vlan **down** が **down** で動作している間に **no shutdown** コマンドが発行されるなど）にトリガーされます。

ステップ 9	**destination-group** *dgrp_id*
	例：
	`switch(conf-tm-sensor)# destination-group 100`
	`switch(conf-tm-dest)#`
		接続先グループを作成して、接続先グループ構成モードを開始します。
		現在、*dgrp_id* は、数字の ID 値のみをサポートしています。

ステップ 10	（任意）**ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	例：
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON`
		エンコードされたテレメトリ データを受信する IPv4 IP アドレスとポートを指定します。
		（注） gRPC はデフォルトのトランスポート プロトコルです。
		GPB がデフォルトのエンコーディングです。

ステップ 11	*ip_version* **address** *ip_address* **port** *portnum* 例： `switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`	発信データの接続先プロファイルを作成します。 接続先グループがサブスクリプション ノード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 12	**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ときは、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。

ステップ 13	**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

	コマンドまたはアクション	目的
		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 14	**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。

CLI を使用したテレメトリの構成例

次の手順では、GPB エンコーディングを使用して 10 秒のリズムで単一のテレメトリ DME ストリームを構成する方法について説明します。

```
switch# configure terminal
switch(config)# feature telemetry
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(config-tm-dest)# ip address 171.70.59.62 port 50051 protocol gRPC encoding GPB
switch(config-tm-dest)# exit
switch(config-telemetry)# sensor group sg1
switch(config-tm-sensor)# data-source DME
switch(config-tm-dest)# path interface depth unbounded query-condition keep-data-type
switch(config-tm-dest)# subscription 1
switch(config-tm-dest)# dst-grp 1
switch(config-tm-dest)# snsr grp 1 sample interval 10000
```

この例では、sys/bgp ルート MO のデータを宛先 IP 1.2.3.4 ポート 50003 に 5 秒ごとにストリーミングするサブスクリプションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

次に、sys/intf のデータを 5 秒ごとに、宛先 IP 1.2.3.4 ポート 50003 にストリーミングし、test.pem を使用して検証された GPB エンコーディングを使用してストリームを暗号化するサブスクリプションの作成例を示します。

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(config-dest)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
```

```
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

この例では、sys/cdp のデータを接続先 IP 1.2.3.4 ポート 50004 に 15 秒ごとにストリーミングするサブスクリプションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000
switch(conf-tm-sub)# dst-grp 100
```

この例では、750 秒ごとに **show** コマンド データのケイデンス ベースのコレクションを作成します。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 172.27.247.72 port 60001 protocol gRPC encoding GPB
switch(conf-tm-dest)# sensor-group 1
switch(conf-tm-sensor# data-source NX-API
switch(conf-tm-sensor)# path "show system resources" depth 0
switch(conf-tm-sensor)# path "show version" depth 0
switch(conf-tm-sensor)# path "show environment power" depth 0
switch(conf-tm-sensor)# path "show environment fan" depth 0
switch(conf-tm-sensor)# path "show environment temperature" depth 0
switch(conf-tm-sensor)# path "show process cpu" depth 0
switch(conf-tm-sensor)# path "show nve peers" depth 0
switch(conf-tm-sensor)# path "show nve vni" depth 0
switch(conf-tm-sensor)# path "show nve vni 4002 counters" depth 0
switch(conf-tm-sensor)# path "show int nve 1 counters" depth 0
switch(conf-tm-sensor)# path "show policy-map vlan" depth 0
switch(conf-tm-sensor)# path "show ip access-list test" depth 0
switch(conf-tm-sensor)# path "show system internal access-list resource utilization"
depth 0
switch(conf-tm-sensor)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-dest)# snsr-grp 1 sample-interval 750000
```

この例では、sys/fm のイベント ベースのサブスクリプションを作成します。sys/fm MO に変更がある場合にのみ、データは接続先にストリーミングされます。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
```

動作中に、サンプル間隔を変更することで、センサー グループを周波数ベースからイベント ベースに変更したり、イベント ベースから周波数ベースに変更したりできます。この例では、

センサー グループを前の例から頻度ベースに変更します。次のコマンドの後、テレメトリ アプリケーションは 7 秒ごとに sys/fm データの接続先へのストリーミングを開始します。

```
switch(config)# telemetry
switch(config-telemetry)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
```

複数のセンサー グループと接続先を 1 つのサブスクリプションにリンクできます。この例のサブスクリプションは、イーサネット ポート 1 / 1 のデータを 4 つの異なる接続先に 10 秒ごとにストリーミングします。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol HTTP encoding JSON
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

次に、センサー グループに複数のパスを含め、接続先グループに複数の接続先プロファイルを含め、サブスクリプションを複数のセンサー グループと宛先グループにリンクできる例を表示します。

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# path sys/epId-1 depth 0
switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0

switch(config-telemetry)# sensor-group 200
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# path sys/ipv4 depth 0

switch(config-telemetry)# sensor-group 300
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# path sys/bgp depth 0

switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 4.3.2.5 port 50005

switch(conf-tm-dest)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001

switch(conf-tm-dest)# destination-group 300
switch(conf-tm-dest)# ip address 1.2.3.4 port 60003

switch(conf-tm-dest)# subscription 600
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 200 sample-interval 20000
```

```
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200

switch(conf-tm-dest)# subscription 900
switch(conf-tm-sub)# snsr-grp 200 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 300 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 300
```

この例に示すように、**show running-config telemetry** コマンドを使用してテレメトリ構成を確認できます。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# end
switch# show run telemetry

!Command: show running-config telemetry
!Time: Thu Oct 13 21:10:12 2016

version 7.0(3)I5(1)
feature telemetry

telemetry
destination-group 100
ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
```

テレメトリの構成と統計情報の表示

次の NX-OS CLI **show** コマンドを使用して、テレメトリの構成、統計情報、エラー、およびセッション情報を表示します。

show telemetry 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 stats

次に、テレメトリの構成に関する内部データベース関連の統計情報のコマンドを表示します。

```
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

次に、データ収集に関する簡単な統計情報のコマンドを表示します。

```
switch# show telemetry data collector brief

-----------------------------------------------------------------------
Collector Type       Successful Collections     Failed Collections
-----------------------------------------------------------------------
DME                   143                        0
```

show telemetry data collector details

次に、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のコマンドを表示します。

```
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を表示します。

```
switch# show telemetry pipeline stats
Main Statistics:
    Timers:
        Errors:
            Start Fail        =     0

    Data Collector:
        Errors:
            Node Create Fail  =     0

    Event Collector:
        Errors:
```

```
            Node Create Fail  =     0    Node Add Fail     =     0
            Invalid Data      =     0

Queue Statistics:
    Request Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

    Data Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0
```

show telemetry transport

次に、構成されているすべての転送セッションの例を表示します。

```
switch# show telemetry transport

Session Id      IP Address      Port       Encoding   Transport  Status
--------------------------------------------------------------------------------
0               192.168.20.123   50001      GPB        gRPC       Connected
```

show telemetry transport <session-id>

次のコマンドでは、特定の転送セッションの詳細なセッション情報が表示されます。

```
switch# show telemetry transport 0

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
```

```
Last Connected:      Fri Sep 02 11:45:57.505 UTC
Tx Error Count:      224
Last Tx Error:       Fri Sep 02 12:23:49.555 UTC

switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235
Encoding:            JSON
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
```

show telemetry transport <session-id> stats

次に、特定の転送セッションの詳細のコマンドを示します。

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

show telemetry transport <session-id> errors

次のコマンドでは、特定の転送セッションの詳細なエラーの統計情報が表示されます。

```
switch# show telemetry transport 0 errors

Session Id:                    0
Connection Stats
   Connection Count            1
   Last Connected:             Mon May 01 11:29:46.912 PST
   Disconnect Count            0
   Last Disconnected:          Never
Transmission Stats
   Transmit Count:             1225
   Last TX time:               Tue May 02 11:40:03.531 PST
   Min Tx Time:                7                  ms
   Max Tx Time:                1760               ms
   Avg Tx Time:                500                ms
```

テレメトリ ログとトレース情報の表示

ログとトレース情報を表示するには、次の NX-OS CLI コマンドを使用します。

テクニカル サポート テレメトリを表示

このNX-OS CLIコマンドは、テクニカル サポート ログからテレメトリ ログの内容を収集します。この例では、コマンド出力がブートフラッシュのファイルにリダイレクトされます。

```
switch# show tech-support telemetry > bootflash:tmst.log
```

NX-API を使用したテレメトリの構成

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in DME のテレメトリ モデル, on page 139. Following are the main objects to be configured:

- **fmEntity** — Contains the NX-API and Telemetry feature states.
  - **fmNxapi** — Contains the NX-API state.
  - **fmTelemetry** — Contains the Telemetry feature state.
- **telemetryEntity** — Contains the telemetry feature configuration.
  - **telemetrySensorGroup** — Contains the definitions of one or more sensor paths or nodes to be monitored for telemetry. The telemetry entity can contain one or more sensor groups.
    - **telemetryRtSensorGroupRel** — Associates the sensor group with a telemetry subscription.
    - **telemetrySensorPath** — A path to be monitored. The sensor group can contain multiple objects of this type.
  - **telemetryDestGroup** — Contains the definitions of one or more destinations to receive telemetry data. The telemetry entity can contain one or more destination groups.
    - **telemetryRtDestGroupRel** — Associates the destination group with a telemetry subscription.
    - **telemetryDest** — A destination address. The destination group can contain multiple objects of this type.
  - **telemetrySubscription** — Specifies how and when the telemetry data from one or more sensor groups is sent to one or more destination groups.
    - **telemetryRsDestGroupRel** — Associates the telemetry subscription with a destination group.
    - **telemetryRsSensorGroupRel** — Associates the telemetry subscription with a sensor group.

To configure the telemetry feature using the NX-API, you must construct a JSON representation of the telemetry object structure and push it to the DME with an HTTP or HTTPS POST operation.

Note For detailed instructions on using the NX-API, see the *Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*.

Before you begin

Your switch must be configured to run the NX-API from the CLI:

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

SUMMARY STEPS

1. Enable the telemetry feature.
2. Create the root level of the JSON payload to describe the telemetry configuration.
3. Create a sensor group to contain the defined sensor paths.
4. Define a telemetry destination group.
5. Define a telemetry destination profile.
6. Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
7. Create a telemetry subscription to configure the telemetry behavior.
8. Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**).
9. Create a relation object as a child object of the subscription to associate the subscription to the telemetry sensor group and to specify the data sampling behavior.
10. Define one or more sensor paths or nodes to be monitored for telemetry.
11. Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).
12. Add destinations as child objects to the destination group object (**telemetryDestGroup**).
13. Add the destination group object as a child object to the root element (**telemetryEntity**).
14. Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
15. Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
16. Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.
17. Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.

DETAILED STEPS

	Command or Action	Purpose

ステップ 1	Enable the telemetry feature.
	Example:
	`{`
	`  "fmEntity" : {`
	`  "children" : [{`
	`  "fmTelemetry" : {`
		The root element is **fmTelemetry** and the base path for this element is `sys/fm`. Configure the **adminSt** attribute as `enabled`.

	Command or Action	Purpose

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

ステップ 2	Create the root level of the JSON payload to describe the telemetry configuration.
	Example:
	<pre>{
	"telemetryEntity": {
	"attributes": {
	"dn": "sys/tm"
	},
	}
	}</pre>
		The root element is **telemetryEntity** and the base path for this element is `sys/tm`. Configure the **dn** attribute as `sys/tm`.

ステップ 3	Create a sensor group to contain the defined sensor paths.
	Example:
	<pre>"telemetrySensorGroup": {
	"attributes": {
	"id": "10",
	"rn": "sensor-10"
	
	}, "children": [{
	}]
	}</pre>
		A telemetry sensor group is defined in an object of class **telemetrySensorGroup**. Configure the following attributes of the object:
		• **id** — An identifier for the sensor group. Currently only numeric ID values are supported.
		• **rn** — The relative name of the sensor group object in the format: **sensor-***id*.
		Children of the sensor group object will include sensor paths and one or more relation objects (**telemetryRtSensorGroupRel**) to associate the sensor group with a telemetry subscription.

ステップ 4	Define a telemetry destination group.
	Example:
	<pre>{
	"telemetryDestGroup": {
	"attributes": {
	"id": "20"
	}
	}
	}</pre>
		A telemetry destination group is defined in **telemetryEntity**. Configure the id attribute.

ステップ 5	Define a telemetry destination profile.
	Example:
	<pre>{
	"telemetryDestProfile": {
	"attributes": {
	"adminSt": "enabled"</pre>
		A telemetry destination profile is defined in **telemetryDestProfile**.
		• Configure the **adminSt** attribute as `enabled`.
		• Under **telemetryDestOptSourceInterface**, configure the **name** attribute with an interface name to stream

	Command or Action	Purpose

	<pre> },
	      "children": [
	         {
	            "telemetryDestOptSourceInterface":
	{
	               "attributes": {
	                  "name": "lo0"
	               }
	            }
	         }
	      ]
	   }
	}</pre>
		data from the configured interface to a destination with the source IP address.

ステップ 6	Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
	Example:
	<pre>{
	    "telemetryDest": {
	        "attributes": {
	             "addr": "1.2.3.4",
	             "enc": "GPB",
	             "port": "50001",
	             "proto": "gRPC",
	             "rn": "addr-[1.2.3.4]-port-50001"
	        }
	    }
	}</pre>
		A telemetry destination is defined in an object of class **telemetryDest**. Configure the following attributes of the object:
		• **addr** — The IP address of the destination.
		• **port** — The port number of the destination.
		• **rn** — The relative name of the destination object in the format: **path-[***path***]**.
		• **enc** — The encoding type of the telemetry data to be sent. NX-OS supports:
		  • Google protocol buffers (GPB) for gRPC.
		  • JSON for C.
		• **proto** — The transport protocol type of the telemetry data to be sent. NX-OS supports:
		  • gRPC
		  • HTTP

ステップ 7	Create a telemetry subscription to configure the telemetry behavior.
	Example:
	<pre>"telemetrySubscription": {
	    "attributes": {
	        "id": "30",
	        "rn": "subs-30"
	    }, "children": [{
	    }]
	}</pre>
		A telemetry subscription is defined in an object of class **telemetrySubscription**. Configure the following attributes of the object:
		• **id** — An identifier for the subscription. Currently only numeric ID values are supported.
		• **rn** — The relative name of the subscription object in the format: **subs-***id*.
		Children of the subscription object will include relation objects for sensor groups (**telemetryRsSensorGroupRel**) and destination groups (**telemetryRsDestGroupRel**).

	Command or Action	Purpose
ステップ 8	Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**). **Example:** `{ "telemetrySubscription": { "attributes": { "id": "30" } "children": [{ "telemetryRsSensorGroupRel": { "attributes": { "sampleIntvl": "5000", "tDn": "sys/tm/sensor-10" } } } ] } }`	
ステップ 9	Create a relation object as a child object of the subscription to associate the subscription to the telemetry sensor group and to specify the data sampling behavior. **Example:** `"telemetryRsSensorGroupRel": { "attributes": { "rType": "mo", "rn": "rssensorGroupRel-[sys/tm/sensor-10]", "sampleIntvl": "5000", "tCl": "telemetrySensorGroup", "tDn": "sys/tm/sensor-10", "tType": "mo" } }`	The relation object is of class **telemetryRsSensorGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object: • **rn** — The relative name of the relation object in the format: **rssensorGroupRel-[sys/tm/***sensor-group-id***]**. • **sampleIntvl** — The data sampling period in milliseconds. An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds. • **tCl** — The class of the target (sensor group) object, which is **telemetrySensorGroup**. • **tDn** — The distinguished name of the target (sensor group) object, which is **sys/tm/***sensor-group-id*. • **rType** — The relation type, which is **mo** for managed object. • **tType** — The target type, which is **mo** for managed object.

	Command or Action	Purpose
ステップ 10	Define one or more sensor paths or nodes to be monitored for telemetry. **Example:** Single sensor path `{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }` **Example:** Multiple sensor paths `{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }, { "telemetrySensorPath": { "attributes": { "excludeFilter": "", "filterCondition": "", "path": "sys/fm/dhcp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }`  **Example:** Single sensor path filtering for BGP disable events:  `{ "telemetrySensorPath": {`	A sensor path is defined in an object of class **telemetrySensorPath**. Configure the following attributes of the object: • **path** — The path to be monitored. • **rn** — The relative name of the path object in the format: **path-[***path***]** • **depth** — The retrieval level for the sensor path. A depth setting of **0** retrieves only the root MO properties. • **filterCondition** — (Optional) Creates a specific filter for event-based subscriptions. The DME provides the filter expressions. For more information regarding filtering, see the Cisco APIC REST API Usage Guidelines on composing queries: 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

	Command or Action	Purpose

	<pre>"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 11	Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).	
ステップ 12	Add destinations as child objects to the destination group object (**telemetryDestGroup**).	
ステップ 13	Add the destination group object as a child object to the root element (**telemetryEntity**).	

ステップ 14	Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
	Example:
	<pre>"telemetryRtSensorGroupRel": {
	"attributes": {
	"rn":
	"rtsensorGroupRel-[sys/tm/subs-30]",
	"tCl": "telemetrySubscription",
	"tDn": "sys/tm/subs-30"
	}
	}</pre>
		The relation object is of class **telemetryRtSensorGroupRel** and is a child object of **telemetrySensorGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtsensorGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

ステップ 15	Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
	Example:
	<pre>"telemetryRtDestGroupRel": {
	"attributes": {
	"rn": "rtdestGroupRel-[sys/tm/subs-30]",
	
	"tCl": "telemetrySubscription",
	"tDn": "sys/tm/subs-30"
	}
	}</pre>
		The relation object is of class **telemetryRtDestGroupRel** and is a child object of **telemetryDestGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtdestGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

ステップ 16	Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group. **Example:**	The relation object is of class **telemetryRsDestGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:

	Command or Action	Purpose

	<pre>"telemetryRsDestGroupRel": {
	    "attributes": {
	        "rType": "mo",
	       "rn": "rsdestGroupRel-[sys/tm/dest-20]",
	
	        "tCl": "telemetryDestGroup",
	        "tDn": "sys/tm/dest-20",
	        "tType": "mo"
	    }
	}</pre>
		• **rn** — The relative name of the relation object in the format: **rsdestGroupRel-[sys/tm/***destination-group-id***]**.
		• **tCl** — The class of the target (destination group) object, which is **telemetryDestGroup**.
		• **tDn** — The distinguished name of the target (destination group) object, which is **sys/tm/***destination-group-id*.
		• **rType** — The relation type, which is **mo** for managed object.
		• **tType** — The target type, which is **mo** for managed object.

ステップ **17**	Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.	The base path for the telemetry entity is `sys/tm` and the NX-API endpoint is: `{{URL}}/api/node/mo/sys/tm.json`

Example

The following is an example of all the previous steps collected into one POST payload (note that some attributes may not match):

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
```

```
                "addr": "10.30.217.80",
                "port": "50051",
                "enc": "GPB",
                "proto": "gRPC"
              }
            }
          }
          ]
        }
      },
      {
        "telemetrySubscription": {
          "attributes": {
            "id": "30"
          }
          "children": [{
            "telemetryRsSensorGroupRel": {
              "attributes": {
                "sampleIntvl": "5000",
                "tDn": "sys/tm/sensor-10"
              }
            }
          },
          {
            "telemetryRsDestGroupRel": {
              "attributes": {
                "tDn": "sys/tm/dest-20"
              }
            }
          }
          ]
        }
      }
      ]
    }
}
```

NX-API を使用したテレメトリの構成例

宛先へのストリーミング パス

この例では、パス `sys/cdp` および `sys/ipv4` を接続先 `1.2.3.4` ポート `50001` に 5 秒ごとにストリーミングするサブスクリプションを作成します。

```
POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
    "telemetryEntity": {
        "attributes": {
            "dn": "sys/tm"
        },
        "children": [{
            "telemetrySensorGroup": {
                "attributes": {
                    "id": "10",
                    "rn": "sensor-10"
                },  "children": [{
                    "telemetryRtSensorGroupRel": {
```

```
                    "attributes": {
                        "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                        "tCl": "telemetrySubscription",
                        "tDn": "sys/tm/subs-30"
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/cdp",
                        "rn": "path-[sys/cdp]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/ipv4",
                        "rn": "path-[sys/ipv4]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }]
        }
    }, {
        "telemetryDestGroup": {
            "attributes": {
                "id": "20",
                "rn": "dest-20"
            },
            "children": [{
                "telemetryRtDestGroupRel": {
                    "attributes": {
                        "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                        "tCl": "telemetrySubscription",
                        "tDn": "sys/tm/subs-30"
                    }
                }
            }, {
                "telemetryDest": {
                    "attributes": {
                        "addr": "1.2.3.4",
                        "enc": "GPB",
                        "port": "50001",
                        "proto": "gRPC",
                        "rn": "addr-[1.2.3.4]-port-50001"
                    }
                }
            }]
        }
    }, {
        "telemetrySubscription": {
            "attributes": {
                "id": "30",
                "rn": "subs-30"
            },
```

```
                "children": [{
                    "telemetryRsDestGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                            "tCl": "telemetryDestGroup",
                            "tDn": "sys/tm/dest-20",
                            "tType": "mo"
                        }
                    }
                }, {
                    "telemetryRsSensorGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                            "sampleIntvl": "5000",
                            "tCl": "telemetrySensorGroup",
                            "tDn": "sys/tm/sensor-10",
                            "tType": "mo"
                        }
                    }
                }]
            }
        }]
    }
}
```

BGP 通知のフィルタ条件

次のペイロードの例では、`telemetrySensorPath` MO の `filterCondition` 属性に従って BFP 機能が無効になっているときにトリガーされる通知を有効にします。データは `10.30.217.80` ポート `50055` にストリーミングされます。

```
POST  https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
```

```
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50055",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "0",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

テレメトリ構成のための Postman コレクションの使用

Postman コレクションの例は、テレメトリ機能の構成を開始する簡単な方法であり、1 つのペイロードですべてのテレメトリ CLI に相当するものを実行できます。好みのテキスト エディターを使用して前述のリンクのファイルを変更し、ペイロードをニーズに合わせて更新してから、Postman でコレクションを開いてコレクションを実行します。

DME のテレメトリ モデル

テレメトリ アプリケーションは、次の構造を持つ DME でモデル化されます。

```
model
|----package [name:telemetry]
     |  @name:telemetry
     |----objects
          |----mo [name:Entity]
               |    @name:Entity
               |      @label:Telemetry System
```

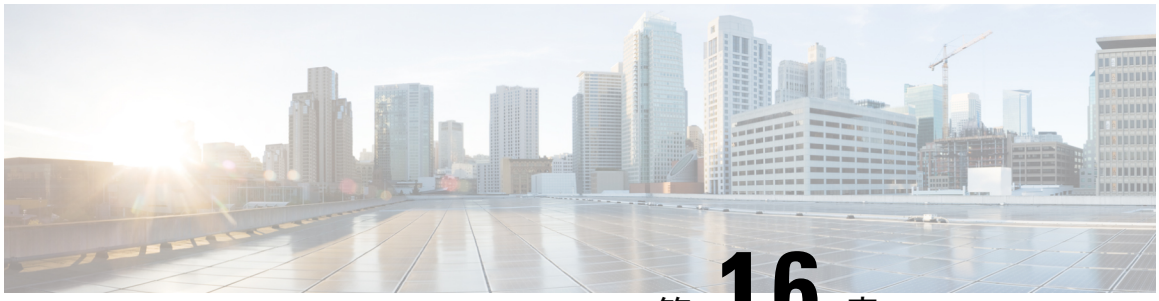
```
|--property
|    @name:adminSt
|      @type:AdminState
|
|----mo [name:SensorGroup]
|    |    @name:SensorGroup
|    |      @label:Sensor Group
|    |--property
|    |    @name:id [key]
|    |      @type:string:Basic
|    |
|    |----mo [name:SensorPath]
|         |   @name:SensorPath
|         |     @label:Sensor Path
|         |--property
|         |    @name:path [key]
|         |      @type:string:Basic
|         |    @name:filterCondition
|         |      @type:string:Basic
|         |    @name:excludeFilter
|         |      @type:string:Basic
|         |    @name:depth
|         |      @type:RetrieveDepth
|
|----mo [name:DestGroup]
|    |    @name:DestGroup
|    |      @label:Destination Group
|    |--property
|    |    @name:id
|    |      @type:string:Basic
|    |
|    |----mo [name:Dest]
|         |   @name:Dest
|         |     @label:Destination
|         |--property
|         |    @name:addr [key]
|         |      @type:address:Ip
|         |    @name:port [key]
|         |      @type:scalar:Uint16
|         |    @name:proto
|         |      @type:Protocol
|         |    @name:enc
|         |      @type:Encoding
|
|----mo [name:Subscription]
     |    @name:Subscription
     |      @label:Subscription
     |--property
     |    @name:id
     |      @type:scalar:Uint64
     |----reldef
     |    |  @name:SensorGroupRel
     |    |    @to:SensorGroup
     |    |    @cardinality:ntom
     |    |    @label:Link to sensorGroup entry
     |    |--property
     |         @name:sampleIntvl
     |           @type:scalar:Uint64
     |
     |----reldef
          |  @name:DestGroupRel
          |    @to:DestGroup
          |    @cardinality:ntom
```

```
|     @label:Link to destGroup entry
```

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 16 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 （143 ページ）
- OpenConfig YANG のガイドラインと制限事項 （143 ページ）
- BGP ルーティング インスタンスの削除について （148 ページ）
- OpenConfig サポートの有効化 （150 ページ）

OpenConfig YANG について

OpenConfig YANG は、宣言型の構成やモデル駆動型の管理と操作など、最新のネットワーキングの原則をサポートしています。OpenConfig は、ネットワークの構成とモニタリングのためにベンダーに依存しないデータ モデルを提供します。また、サブスクリプションとイベント更新ストリーミングにより、プル モデルからプッシュ モデルへの移行を支援します。

Cisco NX-OS リリース 9.2(1) 以降、幅広い機能エリアにわたってサポートが追加されています。これらには、BGP、OSPF、インターフェイス L2 と L3、VRF、VLAN、TACAC が含まれます。

OpenConfig YANG の詳細については、「OpenConfig YANG について」を参照してください。

Cisco NX-OS 9.2（1）の OpenConfig モデルについては、「YANG モデル 9.2(1)」を参照してください。OpenConfig YANG モデルは Cisco NX-OS リリースごとにグループ化されているため、Cisco NX-OS リリース番号が変更されると、URL の最後の桁が変更されます。

OpenConfig YANG のガイドラインと制限事項

OpenConfig YANG には、次のガイドラインと制限事項があります。

- 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。

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。

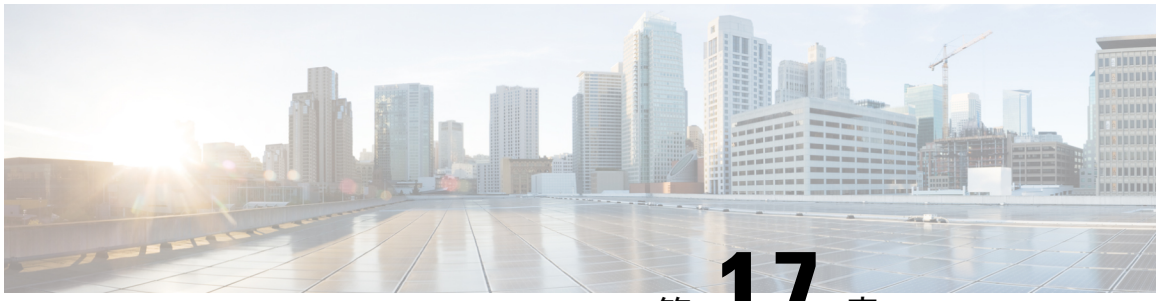
```
router bgp 100
  vrf abc
    address-family ipv4 unicast
      maximum-paths 2
```

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) リリースでは廃止されています。



第 17 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 （151 ページ）
- XML 管理インターフェイスのライセンス要件, on page 153
- XML 管理インターフェイスを使用するための前提条件, on page 153
- XML 管理インターフェイスを使用, on page 153
- サンプル XML インスタンスに関する情報 （166 ページ）
- その他の参考資料, on page 173

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 156 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, on page 152
- SSH xmlagent, on page 152

NETCONF レイヤ

NETCONF レイヤは次のとおりです：

***Table 10: 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 156 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

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 11: 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 167
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 171
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 171
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 167
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 168 NETCONF rollback-on-error インスタンス , on page 172
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 156
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 169
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 167

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 170
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 171
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 172

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 155」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 156 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 165 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。タグ回線は、次の文字コードでマークされます。

- 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 12: *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 165
- データタグにカプセル化されたタグの解釈, on page 165

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 13: 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, on page 167
- NETCONF キルセッション インスタンス, on page 167
- NETCONF copy-config インスタンス, on page 167
- NETCONF edit-config インスタンス, on page 168
- NETCONF get-config インスタンス, on page 169
- NETCONF ロック インスタンス, on page 170

• NETCONF ロック解除インスタンス, on page 171
• NETCONF コミット インスタンス - 候補構成機能, on page 171
• NETCONF Confirmed-commit インスタンス , on page 171
• NETCONF rollback-on-error インスタンス , on page 172
• NETCONF 検証機能インスタンス , on page 172

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

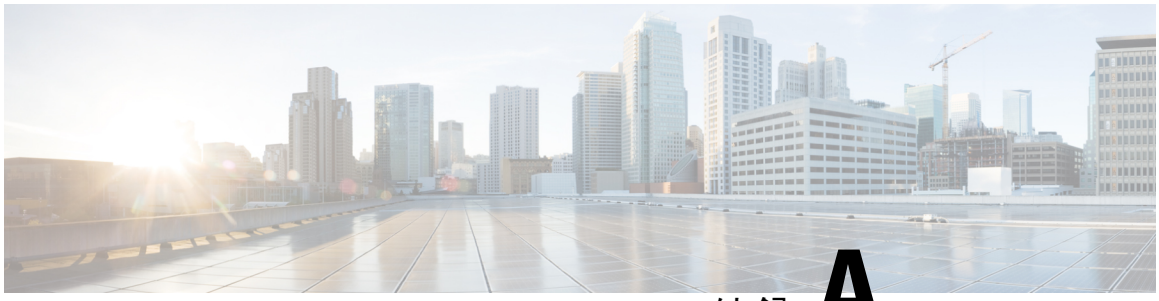
ここでは、XML 管理インターフェイスの実装に関する追加情報について説明します。

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



付録 A

ストリーミング テレメトリの送信元

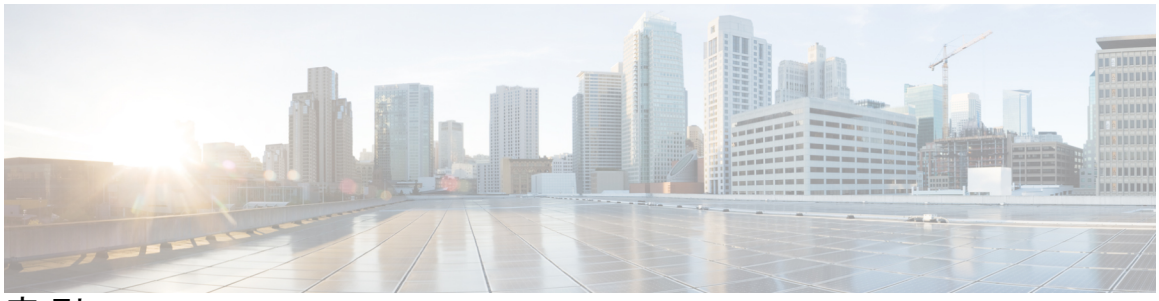
- ストリーミング テレメトリについて （175 ページ）
- テレメトリで利用可能なデータ （175 ページ）

ストリーミング テレメトリについて

Cisco Nexus スイッチのストリーミング テレメトリ機能は、ネットワークからデータを継続的にストリーミングしてクライアントに通知し、モニタリング データへのほぼリアルタイムのアクセスを提供します。

テレメトリで利用可能なデータ

コンポーネント グループごとに、NX-API DME モデル リファレンスの付録にある識別名（DN）は、一覧表示されたプロパティをテレメトリのデータとして提供できます。



索 引

B

bash 3, 5
  アクセス 3
  機能 bash シェル 3
  例 5
Bourne-Again シェル（）。参照先： Bash

N

NX-API 75–76, 78, 80, 85, 87
  CLI 76
  Cookie 76
  応答コード 85
  応答要素 80
  セキュリティ 76
  トランスポート層 76
  管理コマンド 78
  メッセージ形式 76
  ユーザ インターフェイス 87
  リクエスト要素 80

T

tcl 39–41, 44
  cli コマンド 40
  tclquit コマンド 41
  オプション 41
  コマンドの区切り 41
  references 44
  サンドボックス 41
  セキュリティ 41
  対話型ヘルプがない 39
  タブ補完 40
  変数 41
  履歴 40
Tool Command Language。参照先： tcl

て

テクニカル サポート テレメトリを表示 127
telemetry 111
  取り付け 111
  ハイ アベイラビリティ 111

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。