



Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド、 リリース 9.3(x)

最終更新：2025 年 7 月 4 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: <https://www.cisco.com/c/en/us/about/legal/cloud-and-software/software-terms.html>. Cisco product warranty information is available at <https://www.cisco.com/c/en/us/products/warranty-listing.html>. US Federal Communications Commission Notices are found here <https://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: <https://www.cisco.com/c/en/us/about/legal/trademarks.html>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2019–2020 Cisco Systems, Inc. All rights reserved.



目次

はじめに :

はじめに **xxi**

対象読者 **xxi**

表記法 **xxi**

Cisco Nexus 9000 シリーズ スイッチの関連資料 **xxii**

マニュアルに関するフィードバック **xxii**

Communications, services, and additional information **xxiii**

Cisco Bug Search Tool **xxiii**

Documentation feedback **xxiii**

第 1 章

新機能および変更された機能に関する情報 **1**

新機能および変更された機能に関する情報 **1**

第 2 章

プログラマビリティ機能のプラットフォーム サポート **7**

プログラマビリティ機能のプラットフォーム サポート **7**

第 3 章

概要 **11**

プログラマビリティの概要 **11**

サポートされるプラットフォーム **12**

標準的なネットワーク管理機能 **12**

高度な自動化機能 **12**

電源オン自動プロビジョニング サポート **12**

XMPP のサポート **13**

Chef と Puppet の統合 **13**

OpenDaylight の統合と OpenFlow のサポート **14**

プログラマビリティのサポート	14
NX-API のサポート	14
Python スクリプティング	14
Tcl スクリプティング	15
Broadcom シェル	15
bash	15
bash シェルへのアクセスと Linux コンテナのサポート	15
ゲスト シェル	15
コンテナ トラッカーのサポート	17
Perl モジュール	18

第 1 部 :	シェルとスクリプト化	21

第 4 章	bash	23
	Bash について	23
	注意事項と制約事項	23
	Bash へのアクセス	24
	権限をルートにエスカレーションする	26
	Bash コマンドの例	28
	システム統計情報の表示	28
	CLI からの Bash の実行	28
	機能 RPM の管理	29
	RPM インストールの前提条件	29
	Bash からの機能 RPM のインストール	29
	機能 RPM のアップグレード	30
	機能 RPM のダウングレード	31
	機能 RPM の消去	31
	DME のモジュール性のサポート	32
	DME RPM のインストール	32
	インストールされている RPM の確認	35
	ローカル リポジトリの RPM のクエリ	35

DME RPM のバージョン間ダウングレード	36
ベース RPM へのダウングレード	38
パッチ RPM の管理	40
RPM インストールの前提条件	40
Bash からパッチ RPM を追加する	41
パッチ RPM のアクティブ化	43
パッチ RPM のコミット	44
パッチ RPM の非アクティブ化	45
パッチ RPM の削除	47
SDK または ISO で構築されたサードパーティ プロセスの永続的なデーモン化	48
ネイティブ Bash シェルからのアプリケーションの永続的な起動	49
現用系ブートフラッシュからスタンバイ ブートフラッシュへのファイルの同期	49
Kstack を介してコピー	51
ネイティブ Bash シェルのアプリケーション例	52

第 5 章

ゲスト シェル 55

Guest Shell について	55
Guestshell に関する注意事項と制限事項	56
Guest Shell へのアクセス	62
ゲスト シェルに使用されるリソース	63
ゲスト シェルの機能	63
Guest Shell の NX-OS CLI	64
Guest Shell でのネットワーク アクセス	65
ゲスト シェルでのブートフラッシュへのアクセス	67
Guest Shell の Python	67
Guest Shell バージョン 2.10 までの Python 3 (CentOS 7)	68
ゲスト シェルでの RPM のインストール	71
仮想サービス ゲスト シェルのセキュリティ ポスチャ	72
[カーネル脆弱性パッチ (Kernel Vulnerability Patches)]	73
[ASLR および X-Space のサポート (ASLR and X-Space Support)]	73
名前空間の分離	73

ルータユーザの制限	74
リソース管理	75
ゲスト ファイル システムのアクセス制限	76
ゲスト シェルの管理	76
Guest Shell の無効化	81
ゲスト シェルの破棄	82
Guest Shell の有効化	83
ゲスト シェルの複製	84
ゲスト シェル rootfs のエクスポート	85
Guest Shell rootfs のインポート	85
YAML ファイルのインポート	87
show guestshell コマンド	90
仮想サービスと Guest Shell 情報の検証	91
ゲスト シェルからのアプリケーションの永続的な起動	92
Guest Shell からアプリケーションを永続的に起動する手順	93
ゲスト シェルでのサンプルアプリケーション	93
Guest Shell に関する問題のトラブルシューティング	95

第 6 章

Broadcom シェル	97
Broadcom シェルについて	97
注意事項と制約事項	97
Broadcom シェル (bcm-shell) へのアクセス	97
CLI API を使用した bcm-shell へのアクセス	97
ファブリック モジュールのネイティブ bcm-shell へのアクセス	98
ラインカードの bcm-shell へのアクセス	99

第 7 章

Python API	101
[Python API について (About the Python API)]	101
Python の使用	102
Cisco Python パッケージ	102
CLI コマンド API の使用	103

CLI からの Python インタープリタの呼び出し	105
表示フォーマット	106
非インタラクティブ Python	106
Embedded Event Manager でのスクリプトの実行	108
Cisco NX-OS ネットワーク インターフェイスとの Python 統合	109
Python による Cisco NX-OS セキュリティ	109
セキュリティとユーザー権限の例	109
スケジューラでスクリプトを実行する例	111

第 8 章

tcl によるスクリプティング	113
Tcl について	113
注意事項と制約事項	113
tclsh コマンドのヘルプ	114
tclsh コマンドの履歴	114
tclsh のタブ補完	114
tclsh の CLI コマンド	114
tclsh コマンドの区切り	115
tcl 変数	115
tclquit	116
Tclsh セキュリティ	116
Tclsh コマンドの実行	116
Tclsh コマンドからの Cisco NX-OS モード間の移動	117
tcl の参照	118

第 9 章

iPXE	121
iPXE について	121
ネットブート要件	122
iPXE に関する注意事項と制限事項	122
ブート モードの構成	123
ブート順の構成の確認	124

第 10 章

カーネル スタック 125

カーネル スタックについて 125

注意事項と制約事項 125

ポート範囲の変更 126

kstack を使用した VXLAN について 127

kstack のための VXLAN のセットアップ 127

kstack での VXLAN のトラブルシューティング 128

ネットデバイスのプロパティの変更 128

第 II 部 :

アプリケーション 131

第 11 章

サードパーティ製アプリケーション 133

サードパーティ製アプリケーションについて 133

注意事項と制約事項 133

サードパーティのネイティブ RPM/パッケージのインストール 134

署名付き RPM のインストール 136

署名済み RPM の確認 136

キーの手動インポートによる署名付き RPM のインストール 136

キーの自動インポートによる署名付きサードパーティ RPM のインストール 138

リポジトリへの署名済み RPM の追加 140

永続的なサードパーティ RPM 141

VSH からの RPM のインストール 142

パッケージの追加 142

パッケージのアクティブ化 143

パッケージの非アクティブ化 144

パッケージの削除 144

インストール済みパッケージの表示 145

詳細ログの表示 145

パッケージのアップグレード 145

パッケージのダウングレード 146

サードパーティ製アプリケーション 146**NX-OS 146****DevOps 構成管理ツール 146****V9K 147****自動化ツールの教育コンテンツ 147****collectd 147****Ganglia 147****Iperf 148****LLDP 148****Nagios 148****OpenSSH 148****Quagga 148****スプラנק 149****tcollector 149****tcpdump 149****TShark 150**

第 12 章**Ansible 151****前提条件 151****アンシブルについて 151****Cisco Ansible モジュール 152**

第 13 章**Puppet Agent 153****Puppet について 153****前提条件 154****Puppet エージェント NX-OS 環境 154****ciscopuppet モジュール 154**

第 14 章**SaltStack 157****SaltStack について 157****NX-OS と SaltStack について 158****注意事項と制約事項 158**

SaltStack の Cisco NX-OS 環境	159
SaltStack の NX-API の有効化	159
NX-OS 用の SaltStack のインストール	160

第 15 章	Cisco NX-OS でのシェフ クライアントの使用	161
	シェフについて	161
	前提条件	162
	Chef クライアント NX-OS 環境	162
	cisco-cookbook	163

第 16 章	Nexus アプリケーション開発 : Yocto	165
	Yocto について	165
	Yocto のインストール	165

第 17 章	Nexus アプリケーション開発 : SDK	169
	Cisco SDK について	169
	SDK のインストール	169
	インストールと環境の初期化の手順	170
	SDK を使用したアプリケーションの構築	171
	RPM を使用したアプリケーションのパッケージ化	173
	RPM ビルド環境の作成	173
	一般的な RPM ビルド手順の使用	174
	オプションのプラグインを使用しない collectd RPM の構築例	175
	オプションの Curl プラグインを使用した collectd の RPM のビルド例	176

第 18 章	NX-SDK	179
	NX-SDK について	179
	Go バインディングに関する考慮事項	180
	オンボックス (ローカル) アプリケーションについて	181
	デフォルト Docker イメージ	181
	NX-SDK に関する注意事項と制限事項	181

NX-SDK2.0 について	182
NX-SDK2.5 について	183
リモートアプリケーションについて	183
NX-SDK セキュリティ	184
NX SDK 2.0 のセキュリティ プロファイル	185

第 19 章

Cisco NX-OS での Docker の使用	187
Cisco NX-OS での Docker について	187
Docker の注意事項と制限事項	188
Cisco NX-OS 内で Docker コンテナを設定するための前提条件	188
Docker デーモンの開始	189
自動的に起動するように Docker を構成する	189
Docker コンテナの開始: ホスト ネットワーク モデル	190
Docker コンテナの開始: ブリッジ型ネットワーク モデル	191
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	192
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	193
Cisco Nexus Platform Switches Switchover 時に Docker デーモンの永続性を有効にする	194
Docker ストレージ バックエンドのサイズ変更	195
Docker デーモンの停止	197
Docker コンテナ セキュリティ	198
ユーザー[名前空間 (namespace)]の分離による Docker コンテナの保護	198
cgroup パーティションの移動	199
Kubernetes クラスタにノードを追加	200
Docker のトラブルシューティング	202
Docker の起動が機能不全になる	202
ストレージが不足しているため、Docker が起動に失敗する	203
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)	204
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)	204
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	205

Docker ストレージ バックエンドのサイズ変更が失敗する	205
Docker コンテナがポートで着信トラフィックを受信しない	206
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	206
一般的なトラブルシューティングのヒント	206

第 III 部 : **NX-API** 209

第 20 章 **NX-API CLI** 211

NX-API CLI について	211
注意事項と制約事項	211
転送	212
メッセージ形式	213
セキュリティ	213
NX-API CLI の使用	214
NX-API で権限を root にエスカレーションする	216
NX-API 管理コマンド	217
NX-API を使用したインタラクティブ コマンドの操作	221
NX-API クライアント認証	222
NX-API クライアント基本認証	222
NX-API のクライアント証明書認証	222
注意事項と制約事項	222
NX-API のクライアント証明書認証の前提条件	224
NX-API クライアント証明書認証の構成	224
証明書認証用の Python スクリプトの例	226
cURL 証明書要求の例	226
証明書認証の検証	227
NX-API リクエスト要素	229
NX-API 応答要素	236
NX-API へのアクセスの制限	236
iptables の更新	237
リロード間で Iptable を永続化する	238

NX-API 応答コードの表	240
JSON および XML 構造化出力	243
JSON の概要 (JavaScript オブジェクト表記)	243
XML および JSON 出力の例	244
サンプル NX-API スクリプト	252

第 21 章

NX-API REST 253

NX-API REST について	253
REST による DME 設定の置換	254
REST Put による DME フル構成置換について	254
注意事項と制約事項	254
REST PUT によるシステムレベル構成の置換	255
REST PUT による機能レベルの構成置換	255
REST POST によるプロパティレベルの構成の置換	257
REST PUT の構成置換のトラブルシューティング	257

第 22 章

NX-API 開発者サンドボックス 259

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	259
About the NX-API デベロッパー サンドボックス	259
デベロッパー サンドボックスの注意事項と制約事項	260
注意事項と制約事項	260
メッセージフォーマットとコマンドタイプの構成	262
デベロッパー サンドボックスを使用	265
デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する	265
デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換	268
NX-API 開発者サンドボックス : NX-OS リリース 9.2 (2) 以降	273
About the NX-API デベロッパー サンドボックス	273
注意事項と制約事項	274
メッセージフォーマットと入力タイプの構成	276
デベロッパー サンドボックスを使用	280

デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する	280
デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換	283
デベロッパー サンドボックスを使用して RESTCONF から json または XML に変換する	289

第 IV 部 : モデル駆動型プログラマビリティ 293

第 23 章 インフラストラクチャの概要 295

モデル駆動型プログラマビリティについて	295
プログラム可能なインターフェイス インフラストラクチャについて	295

第 24 章 コンポーネント管理 299

コンポーネント RPM パッケージについて	299
インストールの準備	302
Cisco Artifactory からのコンポーネントのダウンロード	303
RPM パッケージのインストール	303
プログラマブルインターフェイスベースおよび共通モデルコンポーネント RPM パッケージのインストール	303

第 25 章 OpenConfig YANG 305

OpenConfig YANG について	305
OpenConfig YANG のガイドラインと制限事項	305
BGP ルーティング インスタンスの削除について	314
YANG の検証	315

第 26 章 NETCONF エージェント 317

NETCONF エージェントについて	317
NETCONF に関する注意事項と制限事項	318
NETCONF エージェントの構成	320
Cisco NX-OS 9.3(5) 以降の SSH を介した NETCONF エージェントの設定	320

Cisco NX-OS 9.3(4) 以前の NETCONF エージェントの構成	321
NETCONF セッションの確立	322
NETCONF の読み取りおよび書き込み構成	324
NETCONF 通知	332
NETCONF 通知について	332
機能交換	333
イベント ストリームの検出	333
サブスクリプションの作成	334
受信通知	335
サブスクリプションの終了	336
NETCONF の例	337
NETCONF エージェントのトラブルシューティング	341

第 27 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	343
XMLIN に関する情報	343
XMLIN のライセンス要件	343
XMLIN ツールのインストールおよび使用	344
show コマンド出力の XML への変換	345
XMLIN の構成例	345

第 28 章

RESTConf エージェント	349
RESTCONF エージェントについて	349
注意事項と制約事項	350
RESTCONF エージェントの使用方法	350
RESTCONF エージェントのトラブルシューティング	352
エフェメラルデータ	353
RESTCONF のエフェメラル データについて	353
RESTCONF のエフェメラル データの例	353

第 29 章

gRPC エージェント	355
gRPC エージェント	355

gRPC エージェントについて	355
gRPC に関する注意事項と制限事項	357
Cisco NX-OS リリース 9.3(3) 以降の gRPC エージェントの構成	358
gRPC の構成	360
Cisco NX-OS リリース 9.3 (2) 以前の gRPC エージェントの構成	361
gRPC エージェントの使用	364
gRPC エージェントのトラブルシューティング	365
gRPC Protobuf ファイル	366
エフェメラル データ	372
gRPC のエフェメラル データについて	372
gRPC のエフェメラル データの例	373

第 30 章

gNMI : gRPC ネットワーク管理インターフェイス	379
gNMI について	379
gNMI の VRF コンテキスト	380
gNMI サブスクライブ RPC	380
gNMI に関する注意事項と制限事項	382
gNMI の構成	384
gNMI : gRPC ネットワーク管理インターフェイス	386
サーバー証明書の構成	387
キー/証明書の生成の例	388
Cisco NX-OS リリース 9.3(2) 以前のキー/証明書の生成と構成の例	388
Cisco NX-OS リリース 9.3(3) 以降のキー/証明書の生成と構成の例	389
gNMI の確認	391
クライアント	398
DME サブスクリプションの例 : PROTO エンコーディング	398
機能	400
機能について	400
機能に関する注意事項と制限事項	400
機能のクライアント出力の例	401
結果	404

Get について	404
Get に関する注意事項と制限事項	404
設定	405
Set について	405
Set に関する注意事項と制限事項	406
登録	406
サブスクリプションに関する注意事項と制限事項	406
gNMI ペイロード	408
ストリーミング Syslog	411
gNMI のストリーミング Syslog について	411
ストリーミング Syslog に関する注意事項と制限事項 : gNMI	411
Syslog ネイティブ YANG モデル	412
サブスクリプション要求の例	412
PROTO 出力の例	413
JSON 出力の例	416
トラブルシューティング	417
TM トレース ログの収集	417
MTX 内部ログの収集	417

第 31 章

ダイナミック ロガー 421

前提条件 421

参照先 421

第 32 章

モデル駆動型テレメトリ 429

テレメトリについて 429

 テレメトリ コンポーネントとプロセス 430

 テレメトリ プロセスの高可用性 431

テレメトリのライセンス要件 432

注意事項と制約事項 432

CLI を使用したテレメトリの構成 438

 NX-OS CLI を使用したテレメトリの構成 438

YANG パスの頻度の設定	446
CLI を使用したテレメトリの構成例	448
テレメトリの構成と統計情報の表示	452
テレメトリ ログとトレース情報の表示	462
NX-API を使用したテレメトリの構成	464
NX-API を使用したテレメトリの構成	464
NX-API を使用したテレメトリの構成例	475
DME のテレメトリ モデル	478
クラウド スケール ソフトウェア テレメトリ	480
クラウド スケール ソフトウェア テレメトリについて	480
Cloud Scale ソフトウェア テレメトリ メッセージの形式	480
クラウド スケール ソフトウェア テレメトリの注意事項と制限事項	481
テレメトリ パス ラベル	481
テレメトリ パス ラベルについて	481
データの投票またはイベントの受信	482
パス ラベル注意事項と制約事項	482
データまたはイベントをポーリングするためのインターフェイス パスの構成	483
非ゼロ カウンタのインターフェイス パスの構成	485
動作速度のインターフェイス パスの構成	487
複数のクエリによるインターフェイス パスの構成	489
データまたはイベントをポーリングするための環境パスの構成	490
イベントまたはデータをポーリングするためのリソース パスの構成	492
イベントまたはデータをポーリングするための VXLAN パスの構成	493
パス ラベル 構成 を確認	495
パス ラベル情報の表示	496
ネイティブ データ送信元パス	498
ネイティブ データ送信元パスについて	498
ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ	499
注意事項と制約事項	503
ルーティング情報のネイティブ データ送信元パスの構成	503
MAC 情報のネイティブ データ送信元パスの構成	505

すべての MAC 情報のネイティブ データ送信元パスの構成	507
IP 隣接のネイティブ データ パスの構成	509
ネイティブ データ ソース パス情報の表示	511
ストリーミング Syslog	512
テレメトリ用のストリーミング Syslog について	512
syslog 情報のための YANG データ ソース パスの構成	513
Syslog パスのテレメトリ データ ストリーミング	514
JSON 出力の例	516
KVGPB の出力例	516
その他の参考資料	519
関連資料	519

第 V 部 :

XML 管理インターフェイス 521

第 33 章

XML 管理インターフェイス 523

XML 管理インターフェイスについて	523
XML 管理インターフェイスについての情報	523
NETCONF レイヤ	523
SSH xmlagent	524
XML 管理インターフェイスのライセンス要件	525
XML 管理インターフェイスを使用するための前提条件	525
XML 管理インターフェイスを使用	525
CLI による SSH および XML サーバー オプションの構成	525
SSHv2 セッションの開始	527
hello メッセージの送信	527
XML スキーマ定義 (XSD) ファイルの取得	528
XML ドキュメントを XML サーバに送信する	528
NETCONF XML インスタンスの作成	529
RPC リクエスト タグ	530
NETCONF 動作タグ	531
デバイスタグ	532

拡張された NETCONF の操作	534
NETCONF 応答	538
RPC 応答タグ	538
データ タグにカプセル化されたタグの解釈	539
サンプル XML インスタンスに関する情報	540
XML インスタンスの例	540
NETCONF クローズ セッション インスタンス	540
NETCONF 強制終了セッション インスタンス	541
NETCONF での構成インスタンスのコピー	541
NETCONF Edit Config インスタンス	542
NETCONF の Get 構成インスタンス	543
NETCONF ロック インスタンス	544
NETCONF ロック解除インスタンス	545
NETCONF コミット インスタンス : 候補構成機能	545
NETCONF 確認済みコミット インスタンス	546
NETCONF Rollback-On-Error インスタンス	546
NETCONF 検証機能インスタンス	547
その他の参考資料	547

付録 A :	ストリーミング テレメトリの送信元	549
	ストリーミング テレメトリについて	549
	注意事項と制約事項	549
	テレメトリで利用可能なデータ	549

付録 B :	WebSocket サブスクリプション	551
	WebSocket サブスクリプション	551

付録 C :	プログラマビリティに関する RFC	555
	プログラマビリティに関する RFC	555



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](#) (xxi ページ)
- [表記法](#) (xxi ページ)
- [Cisco Nexus 9000 シリーズ スイッチの関連資料](#) (xxii ページ)
- [マニュアルに関するフィードバック](#) (xxii ページ)
- [Communications, services, and additional information](#) (xxiii 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。
<i>italic</i>	イタリック体の文字は、ユーザが値を指定する引数です。
[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、角かっこで囲み、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ないでください。引用符を使用すると、その引用符も含めて string と見なされます。

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 screen 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 9000 シリーズ スイッチの関連資料

Cisco Nexus 9000 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/en/US/products/ps13386/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。

Communications, services, and additional information

- To receive timely, relevant information from Cisco, sign up at [Cisco Profile Manager](#).
- To get the business impact you're looking for with the technologies that matter, visit [Cisco Services](#).
- To submit a service request, visit [Cisco Support](#).
- To discover and browse secure, validated enterprise-class apps, products, solutions, and services, visit [Cisco DevNet](#).
- To obtain general networking, training, and certification titles, visit [Cisco Press](#).
- To find warranty information for a specific product or product family, access [Cisco Warranty Finder](#).

Cisco Bug Search Tool

[Cisco Bug Search Tool](#) (BST) is a gateway to the Cisco bug-tracking system, which maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. The BST provides you with detailed defect information about your products and software.

Documentation feedback

To provide feedback about Cisco technical documentation, use the feedback form available in the right pane of every online document.



第 1 章

新機能および変更された機能に関する情報

この章では、『Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド、9.3(x)』のこのリリースに記載されている、新機能および変更された機能に関するリリース固有の情報について説明します。

- ・ [新機能および変更された機能に関する情報（1 ページ）](#)

新機能および変更された機能に関する情報

次の表では、『Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド、リリース 9.3(x)』に記載されている新機能および変更機能を要約しており、対応するセクションを示しています。

表 1: 新機能および変更された機能

特長	説明	変更が行われたリリース	参照先
モデル駆動型テレメトリのセンサーパスのエイリアスオプション	path CLI コマンドに alias キーワードが追加されました。	9.3(5)	NX-OS CLI を使用したテレメトリの構成（438 ページ）
gNMI Get/Set	Get および Set RPC が追加されました。	9.3(5)	gNMI : gRPC ネットワーク管理インターフェイス（379 ページ）
NETCONF/gRPC	Cisco NX-OS リリース 9.3(5) 以降の NETCONF 構成手順をドキュメント化します。	9.3(5)	NETCONF エージェントの構成（320 ページ）

特長	説明	変更が行われたリリース	参照先
NETCONF OpenConfig 通知	NETCONF 通知での Cisco デバイス YANG と OpenConfig モデルのサポートが追加されました。	9.3(5)	NETCONF に関する注意事項と制限事項 (318 ページ)
Python 3	Python 3 のサポートが追加されました。	9.3(5)	[Python API について (About the Python API)] (101 ページ)
複数キーに対する YANG サポート	単一のテレメトリ DME ストリームの構成が追加されました。	9.3(5)	NX-OS CLI を使用したテレメトリの構成 (438 ページ)
nxapi idle-timeout <timeout>	アイドル状態の NX-API セッションが無効になるまでの時間を構成できます。	9.3(5)	NX-API 管理コマンド (217 ページ)
2 つのエラー応答コードが追加されました。	次の応答コードが追加されました。 • <code>INVALID_REMOTE_IP_ERR</code> • <code>MAX_SESSIONS_ERR</code>	9.3(5)	NX-API 応答コードの表 (240 ページ)
NX-API 開発者サンドボックス	機能がコマンドにアクセスできるようにすることに関する詳細は、「開発者サンドボックスの注意事項と制限事項」セクションに追加されました。	9.3(5)	注意事項と制約事項 (260 ページ)
OC YANG	新しい制限が追加されました。	9.3(5)	OpenConfig YANG のガイドラインと制限事項 (305 ページ)
RFC	サポートされている RFC のリストが追加されました。	9.3(x)	プログラマビリティに関する RFC (555 ページ)
NETCONF RFC 6241	NETCONF が RFC 6241 に準拠していることを文書化しました。	9.3(3)	NETCONF に関する注意事項と制限事項 (318 ページ)

特長	説明	変更が行われたリリース	参照先
NX-API クライアント認証	クライアントが開始する証明書ベースの認証のサポートが追加されました。これにより、NX-API クライアントとサーバの両方が、有効な認証局（CA）を介して割り当てられた X509 SSL 証明書を介して認証されます。	9.3(3)	NX-API のクライアント証明書認証（222 ページ）
ソフトウェアテレメトリ	Cisco Nexus 9300-GX プラットフォームスイッチのサポートが追加されました。	9.3(3)	モデル駆動型テレメトリ（429 ページ）
ストリーミング syslog とフィルタリング syslog	NX-SDK、gNMI - gRPC ネットワーク管理インターフェイス、およびモデル駆動型テレメトリの章で、ストリーミング Syslog およびフィルタ済み Syslog のサポートが追加されました。	9.3(3)	gNMI のストリーミング Syslog について（411 ページ）
テレメトリ/gNMI OpenConfig	Subscribe は OpenConfig モデルをサポートします	9.3(3)	
テレメトリ/gNMI OC ファストパス	Subscribe は OpenConfig モデルをサポートします。	9.3(3)	
ネイティブ データ送信元	path mac-all コマンドによる手順が追加されました。	9.3(2)	すべての MAC 情報のネイティブデータ送信元パスの構成（507 ページ）
テレメトリ マルチスレッディング	show コマンドの出力を更新しました。	9.3(2)	gNMI : gRPC ネットワーク管理インターフェイス（379 ページ）

特長	説明	変更が行われたリリース	参照先
NX-SDK	NX-SDK 2.0 のサポートが追加されました。これには、リモート（オフボックス）アプリケーションのサポートと強化されたセキュリティが含まれています。	9.3(1)	NX-SDK （179 ページ）
SaltStack	SaltStack の自動化と統合のサポートが追加されました。	9.3(1)	SaltStack （157 ページ）
CloudScale テレメトリ	Cisco の CloudScale スイッチファミリのデータテレメトリのパフォーマンスが向上しました。	9.3(1)	クラウドスケールソフトウェアテレメトリについて （480 ページ）
NX-OS copy コマンド	スイッチの copy コマンドの速度と操作性が向上しました。	9.3(1)	Kstack を介してコピー （51 ページ）
NX-OS show コマンド	NX-OS show コマンドの JSON ネイティブおよび JSON ネイティブ プリティサポートが追加されました。	9.3(1)	JSON および XML 構造化出力 （243 ページ）
NX-API チャンク	メッセージおよびチャンク機能の強化	9.3(1)	NX-API CLI （211 ページ）
REST API	DME のツリー、サブツリー、およびリーフレベルでスイッチ構成を置き換えるための REST サポートが追加されました。	9.3(1)	REST Put による DME フル構成置換について （254 ページ）
モジュラ DME インフラストラクチャ	DME RPM の非侵入型アップグレードまたはダウングレードをサポートするためのモジュラ性が追加されました。	9.3(1)	DME のモジュール性のサポート （32 ページ）

特長	説明	変更が行われたリリース	参照先
モデル駆動型テレメトリ、使いやすさの向上	テレメトリ パス データの複数のクエリを統合するパス ラベルが追加されました テレメトリ メッセージの ノード ID 文字列を設定または変更するためのサポートが追加されました	9.3(1)	テレメトリ パス ラベルについて (481 ページ) モデル駆動型テレメトリ (429 ページ)
モデル駆動型テレメトリ、ネイティブ データ送信元パス	特定のタイプのインフラストラクチャやデータベースの制限なしに、テレメトリ データをストリーミングするためのアプリケーションのサポートが追加されました。	9.3(1)	ネイティブ データ送信元パスについて (498 ページ)
Cisco NX-OS リリース 9.2(x) からアップデートなし)	最初の 9.3(x) リリース	N/A	N/A



第 2 章

プログラマビリティ機能のプラットフォーム サポート

この章では、Cisco Nexus プラットフォームスイート全体でサポートされていない機能のプラットフォーム サポートについて定義します。

- [プログラマビリティ機能のプラットフォーム サポート \(7 ページ\)](#)

プログラマビリティ機能のプラットフォーム サポート

次の表に、各機能でサポートされるプラットフォームと、それらが最初に導入されたリリースを示します。最初の製品リリースでサポートされるプラットフォームの詳細については、リリース ノートを参照してください。

bash シェル

[Bash について \(23 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
DME のモジュール方式	Cisco Nexus 9000 シリーズ スイッチ	Cisco NX-OS 9.3(1)	

Chef クライアント

[Cisco NX-OS でのシェフ クライアントの使用 \(161 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
Chef エージェント	Cisco Nexus 9300 プラットフォームスイッチ Cisco Nexus 9500 プラットフォームスイッチとラインカード	Cisco NX-OS 7.0(3)I2(5)	N9K-C92348GC

モデル駆動型テレメトリ

モデル駆動型テレメトリに戻ります。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
センサーパスのエイリアスオプション	Cisco Nexus 9200、9300-EX、9300-FX/FX2/FXP プラットフォームスイッチ EX/FX ラインカード搭載の Cisco Nexus 9500 プラットフォームスイッチ	Cisco NX-OS 9.3(5)	N9K-C92348GC
ソフトウェアテレメトリ（ダイヤルアウト）	Cisco Nexus 9000 プラットフォームスイッチ	Cisco NX-OS 7.0(3)I5(1)	N9K-92348GC

NETCONF エージェント

NETCONF エージェントに戻ります。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
NETCONF のサポート	Cisco Nexus 9000 プラットフォームスイッチ		N9K-C92348GC

NX-API REST

[NX-API REST \(253 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
DME 構成置換	Cisco Nexus 9000 プラットフォームスイッチ	Cisco NX-OS 9.3(1)	N9K-C92348GC
DME のサポート	Cisco Nexus 9000 プラットフォームスイッチ	Cisco NX-OS 9.3(1)	N9K-C92348GC

Python API

[Python API \(101 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
Python 3	Cisco Nexus 9000 シリーズ スイッチ	Cisco NX-OS 9.3(5)	

Puppet Agent

[Puppet Agent \(153 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
Puppet Agent	Cisco Nexus 9300 および 9500 プラットフォーム スイッチ	Cisco NX-OS 7.0(3)I2(5)	N9K-C92348GC

SaltStack

[SaltStack \(157 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
SaltStack	Cisco Nexus 9000 シリーズ スイッチ	Cisco NX-OS 9.3(1)	

TCL スクリプティング

[tcl によるスクリプティング \(113 ページ\)](#) に戻ってください。

機能	サポートされるプラットフォームまたはラインカード	サポートされるようになった最初のリリース	プラットフォームの例外
TCL シェル	Cisco Nexus 9000 シリーズ スイッチ	-	



第 3 章

概要

- プログラマビリティの概要 (11 ページ)
- サポートされるプラットフォーム (12 ページ)
- 標準的なネットワーク管理機能 (12 ページ)
- 高度な自動化機能 (12 ページ)
- プログラマビリティのサポート (14 ページ)

プログラマビリティの概要

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

- **耐障害性**
クリティカルなビジネスクラスの可用性を確保します。
- **モジュラ型**
ビジネスニーズに対応する拡張機能があります。
- **高度なプログラマティック**
アプリケーションプログラミングインターフェイス (API) を介した迅速な自動化とオーケストレーションを可能にします。
- **セキュア**
データと運用を保護し維持します。
- **柔軟性**
新しいテクノロジーを統合して有効にします。
- **優れた拡張性**
ビジネスと要件に対応して拡大できます。
- **使いやすさ**

必要な学習量が少なく、展開がシンプルで、管理が容易です。

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

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

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

サポートされるプラットフォーム

Cisco NX-OS リリース 7.0(3)I7(1) 以降では、[Nexus スイッチ プラットフォーム サポート マトリクス](#)に基づいて、選択した機能をさまざまな Cisco Nexus 9000 および 3000 スイッチで使用するために、どの Cisco NX-OS リリースが必要かを確認してください。

標準的なネットワーク管理機能

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

高度な自動化機能

デバイス上の拡張 Cisco NX-OS は、自動化をサポートします。プラットフォームには、Power On Auto Provisioning (POAP) のサポートが含まれています。

デバイス上の拡張 Cisco NX-OS は、自動化をサポートします。プラットフォームには、自動化をサポートする機能が含まれています。

電源オン自動プロビジョニング サポート

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

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

XMPP のサポート

Nexus 9000 ファミリ スイッチ上の拡張 NX-OS では、XMPP クライアントがオペレーティングシステムに統合されています。これにより、XMPP 対応チャット クライアントで Nexus 9000 スイッチを管理および構成できます。（チャット クライアントはユーザー間のコミュニケーションに一般的に使用されます）。XMPP のサポートにより、特定の有益な機能を利用できます。

- **グループ コンフィギュレーション**

チャット グループに Nexus 9000 デバイスのセットを追加すると、Nexus 9000 スイッチのセットをグループとして管理できます。デバイスを個別に構成するのではなく、一連の Nexus 9000 シリーズ デバイスに共通の特定の構成をプッシュするのに便利です。

- **1 箇所での管理**

XMPP サーバが単一の管理ポイントとして機能します。1 つの XMPP サーバで認証を受ければ、サーバに登録されているすべてのデバイスにアクセスできます。

- **セキュリティ**

XMPP インターフェイスは、ロールベースのアクセス制御をサポートしています。これにより、ユーザーは自分が許可されているコマンドのみを実行できるようになります。

- **自動化**

XMPP は標準ベースのインターフェイスです。インターフェイスにより、スクリプトや管理ツールによる Nexus 9000 デバイスの管理を自動化できます。

Chef と Puppet の統合

Chef と Puppet は、2 つもインテントベースのインフラストラクチャ自動化フレームワークです。

Chef では、レシピで意図を定義できます。レシピは、再利用可能な一連の構成または管理タスクです。Chef では、レシピを多数のデバイスに展開できます。スイッチで展開される場合、レシピは統計情報と分析情報の収集に使用する、ネットワーク構成とコマンドのセットに変換されます。レシピは、スイッチの自動構成と管理を実現する方法を提供します。

Puppet では、マニフェストと呼ばれる、同様のインテント定義構造を使用します。スイッチで展開される場合、マニフェストはスイッチからの情報の収集に使用する、ネットワーク構成とコマンドのセットに変換されます。

スイッチは、Puppet と Chef の両方のフレームワークをサポートします。Puppet クライアントと Chef クライアントの両方がスイッチの拡張 Cisco NX-OS に統合されています。

OpenDaylight の統合と OpenFlow のサポート

Cisco Nexus スイッチは、オープンソースの OpenDaylight プロジェクトとの統合をサポートしています。OpenDaylight は、オペレータやアプリケーション開発者のインフラストラクチャに対する要件の一部を満たすのに役立ちます。

- 統合された仮想コンピューティング、アプリケーション、ネットワークのリアルタイムオーケストレーションと運用。
- ネットワークへのシンプルなインターフェイス。ルータ、スイッチ、トポロジなどの基礎となる詳細を抽象化して、よりシンプルにすることができます。

Cisco Nexus スイッチの OpenDayLight オーケストレーションでは、OpenDaylight がサウスパウンドフローで利用できる他のプログラムインターフェイス（NETCONF など）もサポートされます。

Cisco Nexus スイッチでは、ネットワーク タップ集約などの使用例に対応するために、OpenFlow もサポートする予定です。

プログラマビリティのサポート

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

NX-API のサポート

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

Python スクリプティング

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

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

デバイスの Python スクリプト機能は、さまざまなタスクを実行するためのスイッチの CLI と、Power On Auto Provisioning (POAP) または Embedded Event Manager (EEM) アクションへのプ

プログラムによるアクセスを提供します。Cisco NX-OS CLI を呼び出す Python コールへの応答は、テキストまたは JSON 出力を返します。

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

Tcl スクリプティング

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

Broadcom シェル

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

bash

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

bash シェルへのアクセスと Linux コンテナのサポート

Cisco Nexus スイッチは、Linux シェルへの直接アクセスと Linux コンテナをサポートしています。Linux シェルへのアクセスでは、スイッチ上の基盤となっている Linux システムにアクセスして、基盤システムを管理できます。Linux コンテナを使用して、独自のソフトウェアを安全にインストールし、Cisco Nexus スイッチの機能を強化することもできます。たとえば、Cobbler などのベアメタル プロビジョニング ツールを Cisco Nexus スイッチにインストールして、トップオブブラック スイッチからベアメタル サーバーを自動的にプロビジョニングすることができます。

ゲスト シェル

プロセスホスト型展開では、仮想サービス実行環境（VSEE）が、NOS が提供するコアルーティングおよびスイッチングアプリケーションから onePK アプリケーションを分離します。また、この環境は、そのコンテンツを同じホスト上の他の仮想サービスから分離します。アプリケーション開発者は、指定された量の CPU 時間、メモリ、ディスク領域、およびその他のリソースを特定の VSEE に割り当てることができます。

シスコは、Cisco Secure Development Lifecycle (SDLC) を利用して、クローズド シスコ システム上の VSEE で実行される onePK アプリケーションを開発します。SDLC は、シスコが提供するアプリケーション、VSEE、およびプラットフォームの脆弱性のフットプリントを削減する、業界で認められている一連のベストプラクティスとツール（シスコ独自のツールを含む）で構成されています。これらのプラクティスには、ロードする前にシスコが署名したバイナリの整合性を確保するランタイム チェックの使用と、シスコが提供する onePK アプリケーションが信頼できるプロセスとして実行できる信頼ドメインの確立が含まれます。

VSEE の onePK アプリケーションは、onePK SDK が提供する onePK API を使用してネットワーク要素と通信します。セキュア通信チャネルが、VSEE と onePK サーバーの間でメッセージを伝送します。信頼ドメインを使用することで、シスコが提供する VSEE 内のプロセス間通信の整合性が保証され、追加のセキュリティ対策により VSEE とシスコ ホスト間の通信が保護されます。

ネットワーク管理者は、ネットワーク要素と対話する VSEE およびアプリケーションの展開を直接制御します。ネットワーク管理者の職務には、アプリケーションパッケージの完全性と有効性を検証する責任が含まれます。シスコは、ネットワーク管理者が標準のソフトウェア開発ツールを使用して、アプリケーションパッケージを展開する前にその完全性を検証できるようにする情報（デジタル署名やMD5チェックサムなど）を提供します。さらに、onePK VSEE セキュリティ インフラストラクチャのデフォルト設定では、シスコが署名したアプリケーション パッケージのみがプロセス ホスト モードで実行できます。署名されていないコンテナまたはサード パーティによって署名されたコンテナの展開を許可するには、ネットワーク管理者が明示的なアクションを実行する必要があります。

onePK ではネットワーク要素への低レベルのアクセスが許可されるため、ネットワーク管理者は、onePK へのアクセスを許可するユーザー プロファイルとアプリケーションについて慎重に選択する必要があります。

Cisco Nexus 9000 シリーズ スイッチは、セキュア Linux コンテナ (LXC) 内で実行される仮想サービス環境をサポートします。仮想サービス環境のゲストシェルで実行されているアプリケーションは、ホスト Cisco NX-OS が提供する他のルーティングおよびスイッチングアプリケーションから分離されます。また、この環境は、その格納ファイルを同じホスト上の他の仮想サービスから分離します。指定した量の CPU 時間、メモリ、ディスク容量、およびその他の情報技術を特定の仮想サービス環境に割り当てることができます。

シスコは、Cisco Secure Development Lifecycle (SDLC) を利用して、Cisco Nexus 9000 シリーズ デバイスの仮想サービス環境で実行されるアプリケーションを開発します。SDLC は、シスコが提供するアプリケーション、仮想サービス環境、およびプラットフォームの脆弱性フットプリントを削減する、業界で認められている一連のベストプラクティスとツール（シスコ独自のツールを含む）で構成されています。これらのプラクティスには、ロードする前にシスコが署名したバイナリの整合性を確保するランタイムチェックの使用と、シスコが提供するアプリケーションが信頼できるプロセスとして実行できる信頼ドメインの確立が含まれます。

ゲストシェルのアプリケーションは、API を使用してネットワークと通信します。セキュアな通信チャネルは、ゲストシェルとデバイス間でメッセージを送ります。信頼ドメインを使用することで、シスコが提供するゲストシェル内のプロセス間通信の整合性が保証され、追加のセキュリティ対策によりゲストシェルとデバイス間の通信が保護されます。

ネットワーク管理者は、ネットワークと対話する仮想サービス環境およびアプリケーションの展開を制御します。ネットワーク管理者の職務には、アプリケーションパッケージの整合性と有効性を検証する責任が含まれます。シスコは、ネットワーク管理者が標準のソフトウェア開発ツールを使用して、アプリケーションパッケージを展開する前にその整合性を検証できるようにする情報（デジタル署名やMD5チェックサムなど）を提供します。さらに、ゲストシェルセキュリティインフラストラクチャのデフォルト設定では、シスコが署名したアプリケーションパッケージのみがホスト上で実行できます。署名されていないコンテナまたはサードパーティによって署名されたコンテナの展開を許可するには、ネットワーク管理者が明示的なアクションを実行する必要があります。

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

ゲストシェルで実行されているアプリケーションは、ソケットAPIを介してホストシステムと外部ネットワークにIP接続します。

ゲストシェルはセキュア Linux コンテナ (LXC) として実装され、ホストシステムの起動時に自動的に開始されます。これにより、システムの起動時にゲストシェルのアプリケーションを自動的に起動できます。ゲスト シェルに使用される CPU、メモリ、およびブートフラッシュ領域の量は、ゲスト シェルとホスト システム間のリソース使用率を調整するように調整できます。ゲスト シェルはシステムのブートフラッシュをマウントし、Linux コマンドを使用してブートフラッシュ上のファイルにアクセスできるようにします。

コンテナトラッカーのサポート

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

次のコマンドは、Kubernetes API サーバーと通信します。

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

Perl モジュール



(注) Cisco NX-OS リリース9.2 (2) 以降では、-R ライン カードを備えた Cisco Nexus 9504 および 9508 スイッチのパール モジュールのサポートが追加されています。

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

- bytes.pm
- feature.pm
- hostname.pl
- lib.pm
- overload.pm
- Carp.pm
- クラス/構造体.pm
- Data/Dumper.pm
- DynaLoader.pm
- エクスポート/Heavy.pm
- FileHandle.pm
- ファイル/ベース名.pm
- ファイル/Glob.pm
- ファイル/仕様.pm
- ファイル/仕様/Unix.pm
- ファイル/stat.pm
- Getopt/Std.pm
- IO.pm
- IO/File.pm
- IO/Handle.pm
- IO/Seekable.pm
- IO/Select.pm
- List/Util.pm
- MIME/Base64.pm
- SelectSaver.pm

- Socket.pm
- Symbol.pm
- Sys/Hostname.pm
- 時刻/ハイ レゾ .pm
- auto/Data/Dumper/Dumper.so
- auto/File/Glob/Glob.so
- auto/IO/IO.so
- auto/List/Util/Util.so
- auto/MIME/Base64/Base64.so
- auto/Socket/Socket.so
- auto/Sys/Hostname/Hostname.so
- auto/Time/HiRes/HiRes.so



第 Ⅰ 部

シェルとスクリプト化

- [bash](#) (23 ページ)
- [ゲスト シェル](#) (55 ページ)
- [Broadcom シェル](#) (97 ページ)
- [Python API](#) (101 ページ)
- [tcl によるスクリプティング](#) (113 ページ)
- [iPXE](#) (121 ページ)
- [カーネル スタック](#) (125 ページ)



第 4 章

bash

この章は次のトピックで構成されています。

- [Bash について \(23 ページ\)](#)
- [注意事項と制約事項 \(23 ページ\)](#)
- [Bash へのアクセス \(24 ページ\)](#)
- [権限をルートにエスカレーションする \(26 ページ\)](#)
- [Bash コマンドの例 \(28 ページ\)](#)
- [機能 RPM の管理 \(29 ページ\)](#)
- [DME のモジュール性のサポート \(32 ページ\)](#)
- [パッチ RPM の管理 \(40 ページ\)](#)
- [SDK または ISO で構築されたサードパーティ プロセスの永続的なデーモン化 \(48 ページ\)](#)
- [ネイティブ Bash シェルからのアプリケーションの永続的な起動 \(49 ページ\)](#)
- [現用系ブートフラッシュからスタンバイ ブートフラッシュへのファイルの同期 \(49 ページ\)](#)
- [Kstack を介してコピー \(51 ページ\)](#)
- [ネイティブ Bash シェルのアプリケーション例 \(52 ページ\)](#)

Bash について

Cisco NX-OS CLI に加えて、Cisco Nexus 30009000 シリーズ スイッチは Bourne-Again SHell (Bash) へのアクセスをサポートします。Bash は、ユーザーが入力したコマンドまたはシェルスクリプトから読み取られたコマンドを解釈します。Bash を使用すると、デバイス上の基盤となる Linux システムにアクセスしてシステムを管理できます。

注意事項と制約事項

Bash シェルには、次の注意事項と制約事項があります。

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

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

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

- **/isan** フォルダ内のバイナリは、コマンドで入力するシェルの環境とは異なるように設定された環境で実行するためのものです。 **run bash** この環境内の動作は予測できないため、Bash シェルからこれらのバイナリを使用しないことをお勧めします。
- デフォルトでは、Bash シェルは自動的にログアウトしません。ユーザーは、標準の Linux 環境変数 **TMOUT** を使用して、目的とするアイドルタイムアウトを設定できます。値は秒単位です。たとえば、「**export TMOUT=300**」です。
- Cisco Python モジュールをインポートする場合は、Bash シェルから Python を使用しないでください。代わりに、NX-OS VSH で最新の Python を使用します。
- 一部のプロセスおよびコマンドでは、大量の出力が発生する可能性があります。 **show** スクリプトを実行していて、実行時間の長い出力を終了する必要がある場合は、**Ctrl+C** (**Ctrl+Z** ではなく) を使用してコマンド出力を終了します。 **Ctrl+Z** を使用すると、このキー コマンドによって **SIGCONT** (信号継続) メッセージが生成され、スクリプトが停止する可能性があります。 **SIGCONT** メッセージによって停止されたスクリプトは、動作を再開するためにユーザの介入が必要です。
- コマンドが実行中であり、強制終了する必要がある場合は、コマンドを使用しないでください。 **show tech supportclear tech-support lock Ctrl+C** を使用します。

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

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

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

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

Bash へのアクセス

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

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

```
switch# show role name dev-ops

Role: dev-ops
Description: Predefined system role for devops access. This role
cannot be modified.
Vlan policy: permit (default)
Interface policy: permit (default)
Vrf policy: permit (default)
-----
Rule      Perm      Type      Scope      Entity
-----
4          permit   command               conf t ; username *
3          permit   command               bcm module *
2          permit   command               run bash *
1          permit   command               python *

switch# show role name network-admin

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

switch#
```

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

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

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

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

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

switch# run bash?
      bash         Linux-bash

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



(注) **run bash command** コマンドで Bash コマンドを実行することもできます。
たとえば、**run bash** コマンドを使用して **whoami** を実行することもできます。

```
run bash whoami
```

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

```
username foo shelltype bash
```

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

権限をルートにエスカレーションする

管理者ユーザーの特権は、ルート アクセスの特権をエスカレーションできます。

以下は、権限をエスカレーションするためのガイドラインです：

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

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

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

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

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

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

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

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

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

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ vsh -h
Cisco NX-OS Software
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
Nexus 9000v software ("Nexus 9000v Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.

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

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



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

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

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

```
bash-4.2# whoami
root
bash-4.2# exit
exit
```

Bash コマンドの例

このセクションには、Bash コマンドと出力の例が含まれています。

システム統計情報の表示

次の例は、システム統計情報の表示方法を示しています：

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

CLI からの Bash の実行

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

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

機能 RPM の管理

RPM インストールの前提条件

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

手順

	コマンドまたはアクション	目的
ステップ 1	switch# show logging logfile grep -i "System ready"	Bash を実行する前に、この手順によって、RPM をインストールまたは追加する前のシステムの準備ができていることを確認します。 以下のような出力が表示されれば、続行します。 2018 Mar 27 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready
ステップ 2	switch# run bash sudo su 例 : switch# run bash sudo su bash-4.2#	Bash をロードします。

Bash からの機能 RPM のインストール

手順

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

例

次に、**bfd** RPM をインストールする例を示します。

```
bash-4.2$ yum list installed | grep n9000
base-files.n9000                3.0.14-r74.2                installed
bfd.lib32_n9000                 1.0.0-r0                    installed
core.lib32_n9000                1.0.0-r0                    installed
eigrp.lib32_n9000               1.0.0-r0                    installed
eth.lib32_n9000                 1.0.0-r0                    installed
isis.lib32_n9000                1.0.0-r0                    installed
lACP.lib32_n9000                1.0.0-r0                    installed
linecard.lib32_n9000            1.0.0-r0                    installed
lldp.lib32_n9000                1.0.0-r0                    installed
ntp.lib32_n9000                 1.0.0-r0                    installed
nxos-ssh.lib32_n9000            1.0.0-r0                    installed
ospf.lib32_n9000                1.0.0-r0                    installed
perf-cisco.n9000_gdb            3.12-r0                     installed
platform.lib32_n9000            1.0.0-r0                    installed
shadow-securetty.n9000_gdb      4.1.4.3-r1                  installed
snmp.lib32_n9000                1.0.0-r0                    installed
svi.lib32_n9000                 1.0.0-r0                    installed
sysvinit-inittab.n9000_gdb      2.88dsf-r14                 installed
tacacs.lib32_n9000              1.0.0-r0                    installed
task-nxos-base.n9000_gdb        1.0-r0                       installed
tor.lib32_n9000                 1.0.0-r0                    installed
vtp.lib32_n9000                 1.0.0-r0                    installed
bash-4.2$ yum list available
bgp.lib32_n9000                  1.0.0-r0
bash-4.2$ sudo yum -y install bfd
```



- (注) 起動時のスイッチのリロード時に、永続的な RPM の代わりにコマンドを使用します。**rpm** **rpm** それ以外の場合、RPM は最初にインストールされたのではなく、リポジトリ名またはファイル名を使用してインストールされるか、または表示されます。**rpm** **rpm** **rpm**

機能 RPM のアップグレード

始める前に

yum リポジトリに RPM の上位バージョンが存在する必要があります。

手順

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

例

次に、**bfd** RPM のアップグレードの例を示します。

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

機能 RPM のダウングレード

手順

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

例

次に、**bfd** RPM をダウングレードする例を示します。

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

機能 RPM の消去



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

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

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

手順

	コマンドまたはアクション	目的
ステップ 1	sudo yum -y erase rpm	RPM を消去します。

例

次の例は、**bfd** RPM を消去する方法を示しています：

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

DME のモジュール性のサポート

NX-OS リリース 9.3(1) 以降、Cisco NX-OS イメージは DME のモジュール性をサポートします。これは、スイッチの RPM マネージャと相互運用して、DME RPM の非侵入型アップグレードまたはダウングレードを可能にします。非侵入型のアップグレードまたはダウングレードにより、システムの再起動を実行せずに RPM をインストールでき、DME データベースに設定がある他のアプリケーションの妨害を防ぐことができます。DME のモジュール性を使用すると、ISSU やシステムのリロードを行わずに、モデルの変更をスイッチに適用できます。



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

DME RPM のインストール

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

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

手順

ステップ 1 copy path-to-dme-rpm bootflash: [/sup-#][/path]

例：

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

SCP を介して DME RPM をブートフラッシュにコピーします。

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

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

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

例：

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

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

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

例：

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

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

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

例：

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

yum install を使用するには、次のコマンドを実行します。

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

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

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

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

このオプションには、次に示すようにユーザーの操作が必要です。

例：

```
switch-1# yum install --no-persist --nocommit dme-2.0.10.0-9.3.1.lib32_n9000
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo | 1.1 kB 00:00 ...
localdb | 951 B 00:00 ...
localdb/primary | 6.2 kB 00:00 ...
localdb 2/2
```

```

patching | 951 B 00:00 ...
thirdparty | 951 B 00:00 ...
wrl-repo | 951 B 00:00 ...
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package dme.lib32_n9000 0:2.0.1.0-9.3.1 will be updated
---> Package dme.lib32_n9000 0:2.0.10.0-9.3.1 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

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

Transaction Summary
=====
Upgrade 1 Package

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

Updated:
dme.lib32_n9000 0:2.0.10.0-9.3.1

Complete!
switch-1#

```

インストールされている RPM の確認

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

手順

方法を選択します。

- NX-OS の場合：

show install active

例：

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

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

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

yum list --patch-only installed | grep dme

例：

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

ローカル リポジトリの RPM のクエリ

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

手順

ステップ 1 run bash

例：

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

スイッチの Bash シェルにログインします。

ステップ2 `ls /bootflash/.rpmstore/patching/localrepo/dme-2.0.1.0-9.3.1.lib32_n9000.rpm`

例 :

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

bash-4.3\$

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

DME RPM のバージョン間ダウングレード

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

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

手順

ダウングレード方法を選択します。

NX-OS の場合 :

- **install add dme-rpm** へのパス **activate downgrade**

例 :

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

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

- **show install active | include dme**

例 :

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

この例では、DME RPM がバージョン 2.0.1.0 ~ 9.3.1 にダウングレードされています。

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

- Bash で、**yum downgrade dmedme-rpm** を実行します。

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

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

例：

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

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

Transaction Summary
=====
Downgrade      1 Package

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

```

mgmt/shmetafiles/sharedmeta-ConstArgMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
mgmt/shmetafiles/sharedmeta-ConstMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
Cleanup      : dme-2.0.10.0-9.3.1.lib32_n9000                2/2

Removed:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Installed:
  dme.lib32_n9000 0:2.0.1.0-9.3.1

Complete!

```

DME RPM の 1 つのバージョンから下位のバージョンにダウングレードします。この例では、バージョン 2.0.10.0 ~ 9.3.1 がバージョン 2.0.1.0 ~ 9.3.1 にダウングレードされます。

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

例 :

```

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

```

インストールされている DME RPM のバージョンを表示します。

ベース RPM へのダウングレード

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

手順

ダウングレード方法を選択します。

NX-OS の場合 :

- **install activate dme-rpm**

例 :

```

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

```

- **show install active | dme**

例 :

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

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

- Bash で、**yum downgrade dmedme-rpm** を実行します。

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

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

例：

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

Dependencies Resolved

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

Transaction Summary
=====
Downgrade      1 Package

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

```

mgmt/shmetafiles/sharedmeta-ClassRelMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
mgmt/shmetafiles/sharedmeta-PropRefsMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-RelsMetaData
mgmt/shmetafiles/sharedmeta-ActionContextMetaData
mgmt/shmetafiles/sharedmeta-SvcMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
Cleanup      : dme-2.0.10.0-9.3.1.lib32_n9000

```

2/2

Removed:

```
dme.lib32_n9000 0:2.0.10.0-9.3.1
```

Installed:

```
dme.lib32_n9000 0:2.0.0.0-9.3.1
```

Complete!
bash-4.3#

ベース DME RPM をインストールします。

• `yum list --patch-only installed | grep dme`

例:

```

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

```

インストールされているベース DME RPM を表示します。

パッチ RPM の管理

RPM インストールの前提条件

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

手順

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

	コマンドまたはアクション	目的
		2018 Mar 27 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready
ステップ 2	switch# run bash sudo su 例 : switch# run bash sudo su bash-4.2#	Bash をロードします。

Bash からパッチ RPM を追加する

手順

	コマンドまたはアクション	目的
ステップ 1	yum list --patch-only	スイッチに存在するパッチ RPM のリストを表示します。
ステップ 2	sudo yum install --add URL_of_patch	リポジトリにパッチを追加します。ここで、URL_of_patch は、/bootflash/ patch などの標準的な Linux 形式ではなく、bootflash:/patch などの明確に定義された形式です。
ステップ 3	yum list --patch-only available	リポジトリに追加されているが非アクティブ状態のパッチのリストを表示します。

例

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

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

```

patching | 951 B 00:00 ...
thirdparty | 951 B 00:00 ...
[#####] 70%Install operation 135 completed successfully at Tue Mar 27
17:45:34 2018.

```

```

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

```

パッチ RPM がインストールされたら、正しくインストールされたことを確認します。
次のコマンドは、リポジトリに追加され、非アクティブ状態のパッチを一覧表示します。

```

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

```

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

```

bash-4.2# sudo yum install --add
bootflash:/nxos.CSCab00002_CSCab00003-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000.tar
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo | 1.1 kB 00:00 ...
localdb | 951 B 00:00 ...
patching | 951 B 00:00 ...
thirdparty | 951 B 00:00 ...
[#####] 70%Install operation 146 completed successfully at Tue Mar 27
21:17:39 2018.

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

```

パッチ RPM のアクティブ化

始める前に

[#unique_86](#) の手順に従って、必要なパッチ RPM がリポジトリに追加されていることを確認します。

手順

	コマンドまたはアクション	目的
ステップ 1	<code>sudo yum install patch_RPM --nocommit</code>	パッチ RPM をアクティブにします。ここで、<code>patch_RPM</code> はリポジトリにあるパッチです。この手順では、パッチの場所を指定しないでください。 (注) <code>--nocommit</code> フラグをコマンドに追加すると、パッチ RPM がこの手順でアクティブになりますが、コミットされません。パッチ RPM をアクティブ化した後にコミットする手順については、パッチ RPM のコミット (44 ページ) を参照してください。

例

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

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

Dependencies Resolved

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

Transaction Summary
```

```

=====
Install          1 Package

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

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

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

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

```

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

```

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

```

パッチ RPM のコミット

手順

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

例

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

```
bash-4.2# sudo yum install nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --commit

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                                | 1.1 kB      00:00 ...
localdb                                    | 951 B       00:00 ...
patching                                  | 951 B       00:00 ...
thirdparty                                | 951 B       00:00 ...
Install operation 142 completed successfully at Tue Mar 27 18:13:16 2018.

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

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

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

パッチ RPM の非アクティブ化

手順

	コマンドまたはアクション	目的
ステップ 1	sudo yum erase patch_RPM --nocommit	パッチ RPM を非アクティブ化します。 (注) コマンドに --nocommit フラグを追加すると、パッチ RPM はこの手順でのみ非アクティブ化されます。
ステップ 2	sudo yum install patch_RPM --commit	パッチ RPM をコミットします。パッチ RPM をコミットしないまま削除しようとする、エラーメッセージが表示されます。

例

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

```
bash-4.2# sudo yum erase nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --nocommit

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
```

```

Setting up Remove Process
Resolving Dependencies
--> Running transaction check
----> Package nxos.CSCab00001-n9k_ALL.lib32_n9000 0:1.0.0-7.0.3.I7.3 will be erased
--> Finished Dependency Resolution

Dependencies Resolved

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

Transaction Summary
=====
Remove          1 Package

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

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

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

```

パッチ RPM は、非アクティブ化した後にコミットする必要があります。パッチ RPM を非アクティブ化した後にコミットしなかった場合に、[パッチ RPM の削除 \(47 ページ\)](#) の手順を使用してパッチ RPM を削除しようとすると、エラーメッセージが表示されます。

```

bash-4.2# sudo yum install nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --commit

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
                : protect-packages
groups-repo                                | 1.1 kB    00:00 ...
localdb                                    | 951 B     00:00 ...
patching                                  | 951 B     00:00 ...
thirdparty                                | 951 B     00:00 ...
Install operation 144 completed successfully at Tue Mar 27 21:09:28 2018.

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

```

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

```

bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
                : protect-packages

```

```
groups-repo | 1.1 kB 00:00 ...
localdb | 951 B 00:00 ...
patching | 951 B 00:00 ...
thirdparty | 951 B 00:00 ...
nxos.CSCab00001-n9k_ALL.lib32_n9000 1.0.0-7.0.3.I7.3 patching
bash-4.2#
```

パッチ RPM の削除

手順

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

例

次に、パッチ RPM を削除する例を示します。

nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000

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

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



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

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

その後、削除を試みる前にパッチ RPM をコミットしませんでした。パッチ RPM を削除する前にコミットする手順については、[パッチ RPM の非アクティブ化 \(45 ページ\)](#) を参照してください。

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

```
bash-4.2# yum list --patch-only
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo | 1.1 kB 00:00 ...
```

```

localdb          | 951 B      00:00 ...
patching         | 951 B      00:00 ...
patching/primary | 197 B      00:00 ...
thirdparty       | 951 B      00:00 ...
bash-4.2#

```

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

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

```

#!/bin/bash
#
# <application_name> Short description of your application
#
# chkconfig: 2345 15 85
# description: Short description of your application
#
### BEGIN INIT INFO
# Provides: <application_name>
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Short description of your application
### END INIT INFO
# See how we were called.
case "$1" in
start)
# Put your startup commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
stop)
# Put your stop commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
status)
# Put your status commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
restart|force-reload|reload)
# Put your restart commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL

```


ネイティブ Bash シェルからのアプリケーションの永続的な起動

手順

-
- ステップ 1** 作成したアプリケーション起動 Bash スクリプトを `/etc/init.d/application_name` にインストールします。
- ステップ 2** `/etc/init.d/application_name start` でアプリケーションを開始します
- ステップ 3** `chkconfig --add application_name` を入力します
- ステップ 4** `chkconfig --level 3 application_name on` を入力します
- 実行レベル 3 は、標準のマルチユーザ実行レベルであり、スイッチが通常実行されるレベルです。
- ステップ 5** `-- application_name` を実行して、アプリケーションがレベル 3 で実行されるようにスケジュールされていることを確認し、レベル 3 が `on` に設定されていることを確認します。 `chkconfiglist`
- ステップ 6** アプリケーションが `/etc/rc3.d` にリストされていることを確認します。「S」の後に数字が続き、アプリケーション名（この例では `tcollector`）が続き、`../init.d/application_name` に Bash 起動スクリプトへのリンクが表示されます。
-

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

現用系ブートフラッシュからスタンバイ ブートフラッシュへのファイルの同期

Cisco Nexus 9500 プラットフォーム スイッチは、通常、高可用性を提供するために 2 つのスーパーバイザ モジュール（1 つの現用系スーパーバイザ モジュールと 1 つのスタンバイ スーパーバイザ モジュール）で構成されています。各スーパーバイザ モジュールには、ファイルストレージ用の独自のブートフラッシュ ファイルシステムがあり、通常、現用系ブートフラッシュ ファイルシステムとスタンバイ ブートフラッシュ ファイルシステムは互いに独立しています。現用系ブートフラッシュに特定のコンテンツが必要な場合、将来スイッチオーバーが発生した場合に備えて、同じコンテンツがスタンバイ ブートフラッシュにも必要でしょう。

Cisco NX-OS 9.2(2) リリースより前は、現用系スーパーバイザ モジュールとスタンバイ スーパーバイザ モジュール間でこのようなコンテンツを手動で管理する必要がありました。Cisco NX-OS 9.2(2) 以降では、スタンバイ スーパーバイザ モジュールが `up` 状態で使用可能なら、現

用系スーパーバイザ モジュールまたは現用系ブートフラッシュ（/bootflash）上の特定のファイルとディレクトリを、スタンバイ スーパーバイザ モジュールまたはスタンバイ ブートフラッシュ（/bootflash_sup-remote）に自動的に同期できます。同期するファイルとディレクトリを選択するには、スイッチに Bash をロードし、現用系ブートフラッシュからスタンバイ ブートフラッシュに同期するファイルとディレクトリを、編集可能ファイル /bootflash/bootflash_sync_list に追加します。

次に例を示します。

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

```
bash-4.2# echo /bootflash/home/admin >> /bootflash/bootflash_sync_list
```

```
bash-4.2# echo /bootflash/nxos.7.0.3.I7.3.5.bin >>
/bootflash/bootflash_sync_list
```

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

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

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

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

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

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

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

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

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

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

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

Kstack を介してコピー

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



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

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

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

```

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

```

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

ネイティブ Bash シェルのアプリケーション例

次の例は、ネイティブ Bash シェルのアプリケーションを示しています：

```

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

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

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

PIDFILE=/tmp/hello.pid

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

```

```

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

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

```

リロード後

```

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

```




第 5 章

ゲスト シェル

- [Guest Shell について](#) (55 ページ)
- [Guestshell に関する注意事項と制限事項](#) (56 ページ)
- [Guest Shell へのアクセス](#) (62 ページ)
- [ゲスト シェルに使用されるリソース](#) (63 ページ)
- [ゲストシェルの機能](#) (63 ページ)
- [仮想サービス ゲスト シェルのセキュリティ ポスチャ](#) (72 ページ)
- [ゲスト ファイル システムのアクセス制限](#) (76 ページ)
- [ゲスト シェルの管理](#) (76 ページ)
- [仮想サービスと Guest Shell 情報の検証](#) (91 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](#) (92 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](#) (93 ページ)
- [ゲスト シェルでのサンプル アプリケーション](#) (93 ページ)
- [Guest Shell に関する問題のトラブルシューティング](#) (95 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。
- スイッチのホスト ファイル システムへのアクセス。
- Cisco NX-API REST へのアクセス。

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

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

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



- (注) デフォルトでは、ゲスト シェルは、有効にすると約 5 MB の RAM と 200 MB のブートフラッシュを占有します。Cisco NX-OS リリース 7.0 (3) I2 (1) 以降、Guest Shell は約 35 MB の RAM を占有します。Guest Shell が使用されていない場合は、**guestshell destroy** コマンドを使用してリソースを再利用します。



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

Guestshell に関する注意事項と制限事項

すべてのリリースに共通の注意事項



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

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

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

したり、Guestshell のコンテキスト内でコマンドを実行したりできます。このコマンドは、パスワードなしの SSH を使用して、デフォルトのネットワーク名前空間にある localhost の使用可能なポートに接続します。

- NXOS の 2 つの異なる VRF 間で（静的または動的に）ルートが交換されている場合、対応する VRF/名前空間のルーティングテーブルに、ゲストシェルコンテナの「共有ルート」が入力されることはありません。
- sshd ユーティリティは、ローカルホストでリッスンして、ネットワークの外部からの接続試行を回避することにより、Guestshell への事前構成された SSH アクセスを保護できます。sshd には次の機能があります。
 - これは、パスワードにフォールバックしないキーベースの認証用に構成されています。
 - Guestshell の再起動後に Guestshell にアクセスするために使用されるキーを読み取ることができるのは root だけです。
 - root だけがホスト上のキーを含むファイルを読み取ることができ、ホスト Bash アクセスを持つ非特権ユーザーがキーを使用して Guestshell に接続できないようにします。ネットワーク管理ユーザーは、Guestshell で sshd の別のインスタンスを開始して、Guestshell ネットワーク管理ユーザーは、Guestshell で sshd の別のインスタンスを開始して、ネットワーク管理ユーザーは、Guestshell で sshd の別のインスタンスを開始して、アクセスできるようにすることができますが、Guestshell にログインするユーザーにはネットワーク管理者権限も与えられます。



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

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

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

- すぐに使用できる新しいスイッチに Cisco NX-OS ソフトウェア リリースをインストールすると、Guestshell が自動的に有効になります。その後のスイッチ ソフトウェアのアップグレードでは、Guestshell は自動的にアップグレードされません。
- Guestshell リリースでは、配布または配布バージョンが変更されると、メジャー番号が増分します。
- NX-OS の Guestshell は、前面パネルのポートに、ファーストクラスの Linux インターフェイスとしてアクセスできます。
- NX-OS の Guestshell は、NX-API へのローカル Unix ソケットを使用し、dohost を介してコマンドシェルにアクセスできます。

1. 9.3(8)以降のNX-OSのGuestshellにおいて、NX-APIソケットへのアクセスは、root/管理者ユーザー権限でのみ許可されます。
 2. 9.3 (8) 以降のNX-OSのGuestshellにおいて、NX-OSファイルシステムへのアクセスは、root/管理者ユーザーだけが行います。
- Guestshellリリースでは、CVEが解決されるとマイナー番号が増分します。Guestshellは、CentOSが公開した場合にのみCVEを更新します。
 - **yum update** を使用して、CentOSリポジトリからサードパーティのセキュリティ脆弱性修正を直接取得することをお勧めします。これにより、Cisco NX-OSソフトウェアのアップデートを待つことなく、更新が利用可能になったときに入手できる柔軟性が得られます。

または、**guestshell update** コマンドを使用すると、既存のGuestshell rootfs が置き換えられます。カスタマイズとソフトウェアパッケージのインストールは、この新しいGuestshell rootfs のコンテキスト内で再度実行する必要があります。
 - bash シェルからのNexusクロックの設定はサポートされていません。

CentOSのサポート終了とGuestshellへの影響

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

Guestshell 2.11

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

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



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

Guestshell 4.x

Guestshell 2.x には Centos 7 が含まれています。Centos 7 のサポート終了は 2024 年初めです。したがって、Guestshell 4.x は RockyLinux 9 ベースとなり、Guestshell 2.x を置き換えることになります。Guestshell 4.x は、次の NX-OS リリースでダウンロード可能になり、デフォルトのオプションとして使用できるようになります。

NX-OS リリース	Guestshell のデフォルトパッケージバージョン	Guestshell のダウンロード可能なオプションが適用可能
9.3(14) 以降	4.1 以降	不要
10.2(6)	2.15	4.0
10.2(7)	2.15	4.1
10.2(8) 以降	4.1 以降	不要
10.3(4)	2.15	4.0
10.3(5) 以降	4.1 以降	不要
10.4(1) および 10.4(2)	2.15	4.0
10.4(3) 以降	4.1 以降	不要
10.5(1) 以降	4.1 以降	不要



- (注) Guestshell 4.x の rootfs サイズは 400 MB です。Guestshell 2.x では 350 MB でした。Guestshell 4.x のダウンロード可能な OVA は、デフォルトとして Guestshell 2.x を実行しているすべてのリリースに後方互換性を持ちます。

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

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



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

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

互換性のないイメージをインストールした場合の出力例を次に示します。

```
switch#
Installer will perform compatibility check first. Please wait.
uri is: /
2014 Aug 29 20:08:51 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE:
Successfully activated virtual service 'guestshell+'
Verifying image bootflash:/n9kpregs.bin for boot variable "nxos".
[#####] 100% -- SUCCESS
Verifying image type.
```

```

#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/
#####] 100% -- SUCCESS
Preparing "bios" version info using image bootflash:/
#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/
#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/
#####] 100% -- SUCCESS
Preparing "nxos" version info using image bootflash:/
#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/
#####] 100% -- SUCCESS
Preparing "" version info using image bootflash:/
#####] 100% -- SUCCESS
"Running-config contains configuration that is incompatible with the new image (strict
incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out which feature
needs to be disabled.".
Performing module support checks.
#####] 100% -- SUCCESS
Notifying services about system upgrade.
[# ] 0% -- FAIL.
Return code 0x42DD0006 ((null)).
"Running-config contains configuration that is incompatible with the new image (strict
incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out
which feature needs to be disabled."
Service "vman" in vdc 1: Guestshell not supported, do 'guestshell destroy' to remove
it and then retry ISSU
Pre-upgrade check failed. Return code 0x42DD0006 ((null)).
switch#

```



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

事前設定された SSHD サービス

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

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

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



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

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

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

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

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

3. ssh-mgmt.service ファイルを次のように編集します。

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

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

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

```
Port 2222
ListenAddress 10.122.84.34
```

5. 次のコマンドを使用して、systemctl デーモンを開始します。

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

6. (オプション) 構成を確認します。

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

7. Guestshell への SSH:

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

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

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

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

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

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

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

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

11. パスワードなしでリモート スイッチに SSH または SCP :

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

Localtime

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



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

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

Guest Shell へのアクセス

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



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

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

```
-rw-rw-rw- 1 2002 503 40724480 Apr 15 2012 /bootflash/red.ova
```



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

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

ゲスト シェルに使用されるリソース

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

リソース	デフォルト	最小/最大
CPU	1 %	1/620%
メモリ	400 MB	256/3840 MB
ストレージ	200 MB	200/2000 MB

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



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

ゲストシェルの機能

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

ゲストシェルは CentOS 7 Linux 環境であり、この流通向けにビルドされたソフトウェア パッケージを、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 などのネットワークング 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 `vrinfo` は、別の名前空間で実行し、プロセスが実行されている名前空間 `/vrf` に関する情報を取得するために提供されています。

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

Guest Shell の NX-OS CLI

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

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

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

NX-OS CLI は連鎖させることができます：

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

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

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




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

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

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

Guest Shell でのネットワーク アクセス

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

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

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

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

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

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

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

chvrf ユーティリティは便宜のために提供されています。

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



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

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

例：

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

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

例：

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

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

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

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

例：

```
nameserver 10.1.1.1
```

```
domain cisco.com
```

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

例：

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

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

例：

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

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

ゲスト シェルでのブートフラッシュへのアクセス

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

例：

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



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

Guest Shell の Python

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

例：

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

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

例：

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



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

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

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

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

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

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

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

Complete!

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

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

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

Complete!

```
[root@guestshell admin]# yum install -y rh-python36 | tail
warning: /var/cache/yum/x86_64/7/centos-scl-rh/packages/rh-python36-2.0-1.el7.x86_64.rpm:
Header V4 RSA/SHA1 Signature, key ID f2ee9d55: NOKEY
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm:
[Errno 12] Timeout on
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm:
(28, 'Operation too slow. Less than 1000 bytes/sec transferred the last 30 seconds')
Trying other mirror.
Importing GPG key 0xF2EE9D55:
  Userid      : "CentOS SoftwareCollections SIG
  (https://wiki.centos.org/SpecialInterestGroup/SCLO) <security@centos.org>"
  Fingerprint: c4db d535 b1fb ba14 f8ba 64a8 4eb8 4e71 f2ee 9d55
  Package     : centos-release-scl-rh-2-3.el7.centos.noarch (@extras)
  From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-SIG-SCLO
  rh-python36-python-libs.x86_64 0:3.6.9-2.el7
  rh-python36-python-pip.noarch 0:9.0.1-2.el7
  rh-python36-python-setuptools.noarch 0:36.5.0-1.el7
  rh-python36-python-virtualenv.noarch 0:15.1.0-2.el7
  rh-python36-runtime.x86_64 0:2.0-1.el7
  scl-utils-build.x86_64 0:20130529-19.el7
  xml-common.noarch 0:0.6.3-39.el7
  zip.x86_64 0:3.0-11.el7
```

Complete!

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



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

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

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

```
[admin@guestshell ~]$ pip3 install requests --user
Collecting requests
  Downloading
https://files.pythonhosted.org/packages/51/1d/23c26cd341e667dd02a00ba99e0f823e89d72c2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl
(57kB)
100% |#####| 61kB 211kB/s
Collecting idna<2.9,>=2.5 (from requests)
```

```

Downloading
https://files.pythonhosted.org/packages/14/2c/cd51d81dce15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl
(58kB)
100% |#####| 61kB 279kB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
Downloading
https://files.pythonhosted.org/packages/bc/a9/01ff8b8562e42746487b1d1bdc7ca55ec7510b22e4c51f14098443c8/chardet-3.0.4-py2.py3-none-any.whl
(133kB)
100% |#####| 143kB 441kB/s
Collecting certifi>=2017.4.17 (from requests)
Downloading
https://files.pythonhosted.org/packages/19/63/d50cac98a0800655a399c3f1d18a7f5a24e7890c9cf50b8e9/certifi-2019.11.28-py2.py3-none-any.whl
(156kB)
100% |#####| 163kB 447kB/s
Collecting urllib3!=1.25.0,!>=1.25.1,<1.26,>=1.21.1 (from requests)
Downloading
https://files.pythonhosted.org/packages/e8/74/6e4f91745020f967d09332b2889d1009057334692d88e4afe91b77f/urllib3-1.25.8-py2.py3-none-any.whl
(125kB)
100% |#####| 133kB 656kB/s
Installing collected packages: idna, chardet, certifi, urllib3, requests
Successfully installed certifi-2019.11.28 chardet-3.0.4 idna-2.8 requests-2.22.0
urllib3-1.25.8
You are using pip version 9.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("https://cisco.com")
<Response [200]>

```

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

```

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

```

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

```

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

```

Complete!

```

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

```



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

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

ゲスト シェルでの RPM のインストール

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

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

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

Yum は依存関係を解決し、必要なすべてのパッケージをインストールします。

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

Dependencies Resolved

Package	Arch	Version	Repository	Size

Installing:

glibc i686 2.17-78.el7 base 4.2 M

Installing for dependencies:

nss-softokn-freebl i686 3.16.2.3-9.el7 base 187 k

Transaction Summary

Install 1 Package (+1 Dependent package)

Total download size: 4.4 M

Installed size: 15 M

Downloading packages:

```
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
(1/2): nss-softokn-freebl-3.16.2.3-9.el7.i686.rpm | 187 kB 00:00:25
(2/2): glibc-2.17-78.el7.i686.rpm | 4.2 MB 00:00:30

Total 145 kB/s | 4.4 MB 00:00:30
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : nss-softokn-freebl-3.16.2.3-9.el7.i686 1/2
Installing : glibc-2.17-78.el7.i686 2/2
error: lua script failed: [string "%triggerin(glibc-common-2.17-78.el7.x86_64)"]:1:
attempt to compare number with nil
Non-fatal "<"unknown">" scriptlet failure in rpm package glibc-2.17-78.el7.i686
Verifying : glibc-2.17-78.el7.i686 1/2
Verifying : nss-softokn-freebl-3.16.2.3-9.el7.i686 2/2

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

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

Complete!
```



(注) パッケージをインストールまたは実行するためにゲストシェルルートファイルシステムにより多くのスペースが必要な場合は、**guestshell resize roofs size-in-MB** コマンドを使用してファイルシステムのサイズを増やします。



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

仮想サービス ゲスト シェルのセキュリティ ポスチャ

スイッチでのゲストシェルと仮想サービスの使用は、ネットワーク管理者がシステムの機能を管理または拡張できる多くの方法のうちの2つにすぎません。これらのオプションは、ネイティブホストコンテキストから切り離された実行環境を提供することを目的としています。この分離により、ネイティブの実行環境と互換性がない可能性のあるソフトウェアをシステムに導入できます。また、システムの動作、パフォーマンス、またはスケールに影響を与えない環境でソフトウェアを実行することもできます。

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

ムの動作、パフォーマンス、またはスケールに影響を与えない環境でソフトウェアを実行することもできます。

[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 3000 9000 NX-OS は、ランタイムディフェンスのためのアドレス空間 Layout Randomization (ASLR) と Executable Space Protection (X-Space) の使用をサポートしています。Cisco が署名したパッケージのソフトウェアは、この機能を利用します。システムに他のソフトウェアがインストールされている場合は、これらのテクノロジーをサポートするホスト OS と開発ツールチェーンを使用して構築することをお勧めします。これにより、ソフトウェアが潜在的な侵入者に提示する潜在的な攻撃対象領域が減少します。

名前空間の分離

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

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

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

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

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

```
[bob@guestshell ~]$ id
uid=1002(bob) gid=503(network-admin)
```

```
groups=503(network-admin),504(network-operator),10(wheel)
```

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

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

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

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

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

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

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

ルートユーザーの制限

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

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

Guest Shell が従う仮想サービス内のルートに対してドロップされる一連の **Linux** 機能は次のとおりです。

CAP_SYS_BOOT	CAP_MKNOD	CAP_SYS_PACCT
CAP_SYS_MODULE	CAP_MAC_OVERRIDE	CAP_SYS_RESOURCE

CAP_SYS_TIME	CAP_SYS_RAWIO	CAP_AUDIT_WRITE
CAP_AUDIT_CONTROL	CAP_SYS_NICE	CAP_NET_ADMIN
CAP_MAC_ADMIN	CAP_SYS_PTRACE	

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

仮想サービス内のルートとして、`tmpfs` と `ramfs` マウントだけでなくバインドマウントも使用できます。他のマウントは防止されます。

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

リソース管理

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

ゲスト ファイル システムのアクセス制限

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

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

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

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

ゲスト シェルの管理

以下は、ゲスト シェルを管理するためのコマンドです。

表 2: ゲスト シェル **CLI** コマンド

コマンド	説明

コマンド	説明
guestshell enable { package [<i>guest shell OVA file</i> <i>rootfs-file-URI</i>]}	• [ゲストシェルOVAファイル (<i>guest shell OVA file</i>)] 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つ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。 • <i>rootfs-file-URI</i> 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 <i>destination-file-URI</i>	ゲスト シェルの rootfs ファイルをローカル URI (ブートフラッシュ、USB1 など) にエクスポートします。[(7.0 (3) I7 (1) 以降のリリース)]
guestshell disable	シャット ダウンとゲスト シェルの無効化

コマンド	説明
guestshell upgrade {package [<i>guest shell OVA file</i> <i>rootfs-file-URI</i>]}	• [<i>ゲストシェルOVAファイル (guest shell OVA file)</i>] 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。 • <i>rootfs-file-URI</i> 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 <code>sshd</code> に依存しています。 コマンドが機能しない場合は、<code>sshd</code> プロセスが誤って停止した可能性があります。NX-OS CLI からゲストシェルの再起動を実行すると、再起動してコマンドを復元できます。
guestshell destroy	ゲスト シェル サービスを非アクティブ化して、アンインストールします。ゲストシェルに関連付けられているすべての技術情報がシステムに返されます。この show virtual-service global コマンドは、これらの技術情報がいつ利用可能になるかを示します。 このコマンドを発行すると、destroy コマンドを実行する前に確認を求めるプロンプトが表示されます。
guestshell run guestshell	シェル プロンプトですでに実行されているゲストシェルに接続します。必要なユーザー名 / パスワード
guestshell run command run guestshell command	ゲスト シェル環境のコンテキスト内で Linux / UNIX コマンドを実行します。 コマンドの実行後、スイッチ プロンプトに戻ります。

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



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



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



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

次の `exec` キーワードは廃止予定です。

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

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

次の構成キーワードは廃止されます。

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

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

Guest Shell の無効化

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

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

例：

```
switch# show virtual-service list
Virtual Service List:
Name                Status              Package Name
-----
guestshell+         Activated           guestshell.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、ゲストシェルの再度有効にすることができます。



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

Guest Shell の有効化

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

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

例：

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

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

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

ベース ブート モードでの Guest Shell の有効化

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

RPM アクティベーション コマンドが次の順序で実行された場合：

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

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

RPM アクティベーション コマンドを逆の順序で実行した場合：

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

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

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

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

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

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

例：

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

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

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

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

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

ゲストシェルの複製

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

アプローチは、ゲストシェル **rootfs** をカスタマイズしてからエクスポートし、ファイルサーバに保存することです。POAP スクリプトは、ゲストシェル **rootfs** を他のスイッチにダウン

ロード (インポート) し、特定のゲスト シェルを多数のデバイスに同時にインストールできます。

ゲスト シェル **rootfs** のエクスポート

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

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

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

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

Guest Shell **rootfs** のインポート

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

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

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

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

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



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



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

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



- (注) rootfs の組み込みバージョンを復元するには：
- Guest Shell が既にインストールされている場合は、**guestshell upgrade** コマンドを（追加のパラメーターなしで）使用します。
 - Guest Shell が破棄されたときに、**guestshell enable** コマンドを（追加パラメータなしで）使用します。



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

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

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

guestshell enable のワークフローの例：

```
switch# copy scp://user@10.1.1.1/my_storage/gs_rootfs.ext4 bootflash: vrf management
switch# guestshell resize cpu 8
Note: System CPU share will be resized on Guest shell enable
switch# guestshell enable package bootflash:gs_rootfs.ext4
Validating the provided rootfs
switch# 2017 Jul 31 14:58:01 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual
service 'guestshell+'
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual
service 'guestshell+'
2017 Jul 31 14:58:33 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```



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



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

YAML ファイルのインポート

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

Guest Shell インポート YAML ファイルの例：

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

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

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

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

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



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

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

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

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

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



```

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

```

```

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

```

show guestshell コマンド

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

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

```

switch# show guestshell detail
Virtual service guestshell+ detail
  State      : Activated
  Package information
    Name      : rootfs_puppet
    Path      : usb2:/rootfs_puppet
  Application
    Name      : GuestShell
    Installed version : 2.3(0.0)
    Description  : Exported GuestShell: 20170613T173648Z
  Signing
    Key type   : Unsigned
    Method    : Unknown
  Licensing
    Name      : None
    Version   : None

```

仮想サービスと Guest Shell 情報の検証

次のコマンドを使用して、仮想サービスとゲストシェルの情報を検証できます。

コマンド	説明
show virtual-service global switch# show virtual-service global Virtual Service Global State and Virtualization Limits: Infrastructure version : 1.11 Total virtual services installed : 1 Total virtual services activated : 1 Machine types supported : LXC Machine types disabled : KVM Maximum VCPUs per virtual service : 1 Resource virtualization limits: Name Quota Committed Available ----- system CPU (%) 20 1 19 memory (MB) 3840 256 3584 bootflash (MB) 8192 200 7992 switch#	仮想サービスのグローバル状態と制限を表示します。
show virtual-service list switch# show virtual-service list * Virtual Service List: Name Status Package Name ----- guestshell+ Activated guestshell.ova chef Installed chef-0.8.1-n9000-spa-k9.ova	仮想サービスの概要、仮想サービスのステータス、およびインストールされているソフトウェア パッケージを表示します。

コマンド	説明
show guestshell detail <pre>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</pre>	guestshell パッケージに関する詳細（バージョン、署名リソース、デバイスなど）を表示します。

ゲスト シェルからのアプリケーションの永続的な起動

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

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

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

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



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

Guest Shell からアプリケーションを永続的に起動する手順

手順

- ステップ1 上記で作成したアプリケーション サービス ファイルを
`/usr/lib/systemd/system/application_name` にインストールします。サービス
- ステップ2 `systemctl start application_name` でアプリケーションを開始します
- ステップ3 アプリケーションが `systemctl status -l application_name` で実行されていることを確認します
- ステップ4 `systemctl enable application_name` でリロード時にアプリケーションを再起動できるようにします
- ステップ5 アプリケーションが `systemctl status -l application_name` で実行されていることを確認します

ゲスト シェルでのサンプル アプリケーション

次の例は、ゲスト シェルのアプリケーションを示しています。

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

OUTPUTFILE=/tmp/hello

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

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

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

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

リロード後

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

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

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

Guest Shell に関する問題のトラブルシューティング

7.0 (3) I7 へのダウングレード後にゲスト シェルにアクセスできない

ゲスト シェルのアクティブ化または非アクティブ化のプロセス中に、NX-OS 9.2 (1) リリースから NX-OS 7.0 (3) 7 リリース イメージ (ユーザー名前空間のサポートがない) にダウングレードした場合、次のコマンドを実行できます。ゲスト シェルは起動しますが、ゲスト シェルにアクセスできない次の状態になります。この問題の理由は、ゲスト シェルの移行中にリロードが発行された場合、ゲスト シェル内のファイルがユーザー名前空間のサポートがない NX-OS リリースで使用可能な識別子範囲に戻されないためです。

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

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

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

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

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

ゲスト シェルのルートからブートフラッシュのファイルにアクセスできない

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

ホストから：

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

ゲスト シェルから：

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

```
-bash: /bootflash/try.that: Permission denied  
[root@guestshellbootflash]#
```

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

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



第 6 章

Broadcom シェル

- [Broadcom シェルについて](#) (97 ページ)
- [注意事項と制約事項](#) (97 ページ)
- [Broadcom シェル \(bcm-shell\) へのアクセス](#) (97 ページ)

Broadcom シェルについて

スイッチの前面パネルおよびファブリック モジュール ライン カードには、Broadcom Network Forwarding Engine (NFE) が搭載されています。NFE の数は、前面パネルラインカード (LC) またはファブリック モジュール (FM) の特定のモデルによって異なります。

注意事項と制約事項

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

Broadcom シェル (bcm-shell) へのアクセス

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

CLI API を使用した bcm-shell へのアクセス

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

コマンドの構文は、次のとおりです。

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

場所

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



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



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

ファブリック モジュールのネイティブ **bcm-shell** へのアクセス

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

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

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

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

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

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

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

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

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

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

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

ラインカードの **bcm-shell** へのアクセス

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

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

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

```
switch# attach module 2
Attaching to module 2 ...
To exit type 'exit', to abort type '$.'
Last login: Wed Aug 7 14:13:15 UTC 2013 from sup27 on ttty0
```

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

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

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

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

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




第 7 章

Python API

- [\[Python API について \(About the Python API\)\]](#) (101 ページ)
- [Python の使用](#) (102 ページ)

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

Cisco NX-OS リリース 9.3(5) 以降、Python 3 はサポート対象です：Python 2.7 は引き続きサポートされます。新しいスクリプトには、**python3** コマンドを使用することをお勧めします。

Cisco Nexus 3000 9000 シリーズ 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 と v3.7.3 をサポートし、ゲストシェルで使用できます。Python スクリプトをサポートする Cisco Nexus スイッチのリストについては、[プログラマビリティ機能のプラットフォーム サポート \(7 ページ\)](#) を参照してください。

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 インタプリタは Cisco NX-OS ソフトウェアで使用できます。



(注) Python をサポートするプラットフォームの場合、ユーザーは低レベルの OS システム コールにアクセスできます。

Python の使用

ここでは、Python スクリプトの作成と実行の方法について説明します。

Cisco Python パッケージ

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

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

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

NAME
    cisco

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

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    vrf

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

```
cisco.key.Key
```

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

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

NAME
cisco

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

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

CLI コマンド API の使用

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

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

表 3: CLI コマンド API

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

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

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

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

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

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

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




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

CLI からの Python インタープリタの呼び出し

次に、CLI から Python 2 を呼び出す方法を表示します：



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



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

```
switch# python
switch# python
```

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

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

次に、CLI から Python 3 を呼び出す方法を表示します：

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

```
>>> cli('configure terminal ; interface loopback 1 ; no shut')
''
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print(intf['interface'])
...
mgmt0
loopback1
>>>
```

表示フォーマット

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

例 1：

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

例 2：

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

例 3：

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

>>>
```

例 4：

非インタラクティブ Python

Python スクリプト名を引数として Python CLI コマンドで使用することで、Python スクリプトを非インタラクティブ モードで実行できます。Python スクリプトは、ブートフラッシュまた

は揮発性スキームの下に配置する必要があります。Python CLI コマンドでは、Python スクリプトの最大 32 個のコマンドライン引数を使用できます。

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

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

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

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

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

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

```

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

```

Embedded Event Manager でのスクリプトの実行

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

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

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

```

switch# show running-config eem

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

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

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

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

```

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

```

switch# show file logflash:event_archive_1 | last 33

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

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

PC_VSH_CMD_TLV(7679) with q

```

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

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

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

```
switch# python
```

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

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

Python による Cisco NX-OS セキュリティ

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

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

セキュリティとユーザー権限の例

次の例は、特権ユーザがコマンドを実行する方法を示しています：

Python 3 の例。

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

次の例は、アクセスを拒否されている非特権ユーザーを示しています：

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

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

以下は、特権ユーザーの例です：

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

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

version 9.3(5) Bios:version 07.67

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

以下は、非特権ユーザーの例です：

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

次の例は、RBAC 構成を示しています：

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

スケジューラでスクリプトを実行する例

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

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

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

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

Python 3 の例。

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

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

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

# Save this script in bootflash:///scripts
```

```

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

```




第 8 章

tcl によるスクリプティング

この章は次のトピックで構成されています。

- [Tcl について](#) (113 ページ)
- [Tclsh コマンドの実行](#) (116 ページ)
- [Tclsh コマンドからの Cisco NX-OS モード間の移動](#) (117 ページ)
- [tcl の参照](#) (118 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。

注意事項と制約事項

次に、TCL スクリプトに関する注意事項と制限事項を示します。

- プラットフォームのサポートに関する注意については、[プログラマビリティ機能のプラットフォーム サポート](#) (7 ページ) を参照してください。
- 一部のプロセスおよびコマンドでは、大量の出力が発生する可能性があります。**show** スクリプトを実行していて、実行時間の長い出力を終了する必要がある場合は、Ctrl+C (Ctrl+Z ではなく) を使用してコマンド出力を終了します。Ctrl+Z を使用すると、SIGCONT (信号継続) メッセージが生成され、スクリプトが停止する可能性があります。SIGCONT メッセージによって停止されたスクリプトは、動作を再開するためにユーザーの介入が必要です。

tclsh コマンドのヘルプ

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

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

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

switch-tcl#
```



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

tclsh コマンドの履歴

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



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

tclsh のタブ補完

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

tclsh の CLI コマンド

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

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

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

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

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

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

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

tclsh コマンドの区切り

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

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

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

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

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

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

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

tcl 変数

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

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

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

tclquit

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

Tclsh セキュリティ

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

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

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

Tclsh コマンドの実行

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



(注) CLI プロンプトの状態では 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

	コマンドまたはアクション	目的
ステップ 1	tclsh [bootflash:filename [<i>argument</i> ...]] 例 : <pre>switch# tclsh ? <CR> bootflash: The file to run</pre>	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

	コマンドまたはアクション	目的
		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

	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。

	コマンドまたはアクション	目的
ステップ 2	configure terminal 例 : <pre>switch-tcl# configure terminal switch(config-tcl)#</pre>	Tcl シェルで Cisco NX-OS のコマンドを実行して、モードを変更します。 (注) Tcl プロンプトが変化して、Cisco NX-OS コマンドモードになったことが示されます。
ステップ 3	tclquit 例 : <pre>switch-tcl# tclquit switch#</pre>	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 9 章

iPXE

この章は、次の内容で構成されています。

- [iPXE について \(121 ページ\)](#)
- [ネットブート要件 \(122 ページ\)](#)
- [iPXE に関する注意事項と制限事項 \(122 ページ\)](#)
- [ブート モードの構成 \(123 ページ\)](#)
- [ブート順の構成の確認 \(124 ページ\)](#)

iPXE について

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

標準 PXE よりも iPXE が提供する追加機能のリストを次に示します。

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

さらに、次のようなセキュリティ上の理由から、iPXE の既存の機能の一部が無効になっています。

- bzImage+initramfs/initrd、または ISO などの標準 Linux イメージ形式のブート サポート
- FCoE、iSCSI SAN、ワイヤレスなどの未使用のネットワークブートオプション

- 適切にコード署名されていないシステムイメージを起動できるため、サポートされていない NBP (syslinux/pxelinux など) のロード。

ネットブート要件

主要な要件は次のとおりです。

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

iPXE に関する注意事項と制限事項

PXE に関する注意事項と制限事項は次のとおりです。

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

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

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

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

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

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

ブートモードの構成

VSH CLI

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



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

たとえば、PXE ブートモードのみを実行する場合、コンフィギュレーション コマンドは次のようになります。

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

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

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

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

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



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

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



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

GRUB CLI

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

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

Grub CLI は、完全な Cisco NX-OS イメージを起動せずにシリアル コンソールからブート モードを切り替える場合に役立ちます。また、継続的な PXE ブート状態からボックスを取得するためにも使用できます。

iPXE CLI

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

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

iPXE CLI は、完全な Cisco NX-OS イメージを起動せずにシリアル コンソールからブート モードを切り替える場合に役立ちます。また、継続的な PXE ブート状態からボックスを取得するためにも使用できます。

ブート順の構成の確認

ブート順の構成情報を表示するには、次のコマンドを入力します。

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



第 10 章

カーネル スタック

この章は、次の内容で構成されています。

- [カーネル スタックについて \(125 ページ\)](#)
- [注意事項と制約事項 \(125 ページ\)](#)
- [ポート範囲の変更 \(126 ページ\)](#)
- [kstack を使用した VXLAN について \(127 ページ\)](#)
- [ネットデバイスのプロパティの変更 \(128 ページ\)](#)

カーネル スタックについて

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

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

カーネル スタックには次の機能があります：

注意事項と制約事項

- ゲストシェル、Docker コンテナ、およびホスト Bash シェルは、カーネルスタック (kstack) を使用します。
- ゲストシェルとホスト Bash シェルは、デフォルトのネットワーク名前空間で開始されます。Docker 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開始されます。
 - 他のネットワーク名前空間には、**setns** システムコールを使用してアクセスできます。
- **nsenter** および **ip netns exec** ユーティリティは、異なるネットワーク名前空間のコンテキスト内で実行するために使用できます。

- インターフェイスの状態は、`/proc/net/dev` から読み取るか、**ip**、**ifconfig**、または **netstat** などの他の一般的な Linux ユーティリティを使用して取得できます。カウンタは、スイッチで開始または終了したパケットに関するものです。
 - **ethtool -S** は、インターフェイスを介してスイッチングされるパケットを含む、ネットデバイスから拡張統計情報を取得するために使用できます。
 - **tcpdump** などのパケット キャプチャ アプリケーションは、スイッチで開始または終了するパケットをキャプチャできます。
 - ゲストシェルからのネットワーク状態の変更（インターフェイスの作成または削除、IP アドレスの構成、MTU の変更など）はサポートされていません。
 - IPv4 と IPv6 がサポートされます。
 - Raw PF_PACKET がサポートされています。
 - ネットワーク名前空間に関係なく、一度に 1 つのスタック（Netstack または kstack）でのみ、ウェルノウンポート（0 ～ 15000）を使用できます。
 - 同じスイッチ上でも、Netstack を使用するアプリケーションと kstack を実行するアプリケーションの間に IP 接続はありません。この制限は、kstack アプリケーションがホスト Bash シェルから実行されているか、コンテナ内で実行されているかに関係なく当てはまりません。
 - ゲスト シェル内のアプリケーションは、ライン カードまたはスタンバイ Sup との通信のために、イーサネットアウトオブバンドチャネル（EOBC）インターフェイスを介してパケットを直接送信することはできません。
 - 管理インターフェイス（mgmt0）は、カーネルネットデバイスで eth1 として表されます。
 - カーネルスタックを使用するアプリケーションでは、VXLAN オーバーレイ インターフェイス（NVE x）の使用はサポートされていません。CLI コマンドを含む NX-OS 機能は、netstack を介してこのインターフェイスを使用できます。
- NVE インターフェイスの詳細については、[Cisco Nexus 9000 シリーズ NX-OS VXLAN 構成ガイド](#)を参照してください。

ポート範囲の変更

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

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



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



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

手順

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

例

次に、**kstack** ポート範囲を設定する例を示します：

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

次のタスク

コマンドを入力した後は、次の点に注意してください：

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

kstack を使用した VXLAN について

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

kstack のための VXLAN のセットアップ

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

kstack での VXLAN のトラブルシューティング

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

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

- **ip route show** コマンドを実行します：

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

次のような出力が表示されます。

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

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

- **ip neigh show** コマンドを実行します：

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

次のような出力が表示されます。

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

ネットデバイスのプロパティの変更

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

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

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

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

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

```
bash-4.3# ip link show Eth1-17
49: Eth1-17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT group default qlen 100
```



```
link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff

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

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

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

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

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

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

NX-OS 9.2(2) リリースの一部である変更点は次のとおりです。

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

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

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



第 II 部

アプリケーション

- [サードパーティ製アプリケーション](#) (133 ページ)
- [Ansible](#) (151 ページ)
- [Puppet Agent](#) (153 ページ)
- [SaltStack](#) (157 ページ)
- [Cisco NX-OS でのシェフ クライアントの使用](#) (161 ページ)
- [Nexus アプリケーション開発：Yocto](#) (165 ページ)
- [Nexus アプリケーション開発：SDK](#) (169 ページ)
- [NX-SDK](#) (179 ページ)
- [Cisco NX-OS での Docker の使用](#) (187 ページ)



第 11 章

サードパーティ製アプリケーション

この章は、次の内容で構成されています。

- サードパーティ製アプリケーションについて (133 ページ)
- 注意事項と制約事項 (133 ページ)
- サードパーティのネイティブ RPM/パッケージのインストール (134 ページ)
- 署名付き RPM のインストール (136 ページ)
- 永続的なサードパーティ RPM (141 ページ)
- VSH からの RPM のインストール (142 ページ)
- サードパーティ製アプリケーション (146 ページ)

サードパーティ製アプリケーションについて

サードパーティ製アプリケーションの RPM は、https://devhub.cisco.com/artifactory/open-nxos/7.0-3-I2-1/x86_64/https://devhub.cisco.com/artifactory/open-nxos/9.2.1/ のリポジトリで入手できます。これらのアプリケーションは、Bash シェルで **yum** コマンドを使用するか、NX-OS CLI を介してネイティブホストにインストールされます。

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

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

注意事項と制約事項

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

- Cisco NX-OS リリース 9.2(1) 以降では、エージェントが保存されている Cisco リポジトリにはありません。<https://devhub.cisco.com/artifactory/open-nxos/9.2.1/> のリポジトリでホストされているすべての RPM は、リリースキーで署名されています。

- NX-OS 9.2(1) リリースには、WindRiver Linux 8 (WRL8) に基づく新しいオペレーティングシステムと rootfs があるため、WRL5 を使用して構築されたサードパーティ製 RPM は WRL8 と互換性がない可能性があります。動作しない可能性があります。この場合、以前のリリースで使用されていたアプリケーションの古いバージョンを削除し、次のリポジトリで入手可能な WRL8 と互換性のある新しいソフトウェアに置き換えます。
<https://devhub.cisco.com/artifactory/open-nxos/9.2.1/>
- 署名付き RPM をインストールするためのガイドラインと手順は、『Cisco Nexus 9000 Series NX-OS Software Upgrade and Downgrade Guide, Release 9.2(x)』に記載されています。リポジトリの数などです。
- サードパーティ製アプリケーションは、スイッチの起動時に開始されます。サードパーティ製アプリケーションは、その通信インターフェイスが起動する前、またはスイッチと通信ピアまたはサーバー間のルーティングが確立される前に起動される可能性があります。したがって、すべてのサードパーティ製アプリケーションは、通信障害が発生した場合に堅牢になるように作成し、アプリケーションは接続の確立を再試行する必要があります。通信障害が発生してもアプリケーションに復元力がない場合は、「ラッパー」アプリケーションを使用して、目的のアプリケーションを起動する前に通信ピアが到達可能であることを確認するか、必要に応じて目的のアプリケーションを再起動する必要があります。

サードパーティのネイティブ RPM/パッケージのインストール

パッケージのインストールの完全なワークフローは次のとおりです。

手順

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

```
bash-4.2# cat /etc/yum/repos.d/open-nxos.repo
[open-nxos]
name=open-nxos
baseurl=https://devhub.cisco.com/artifactory/open-nxos/7.0-3-I2-1/x86\_64/
baseurl=https://devhub.cisco.com/artifactory/open-nxos/9.2.1/

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

CLI を使用してデジタル署名をインポートする手順については、『Cisco Nexus 9000 Series NX-OS Software Upgrade and Downgrade Guide, Release 9.2(x)』の「Using Install CLIs for Digital Signature Support」を参照してください。

フルインストールログを使用した yum dnf を使用した RPM のインストールの例。

例：

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

Dependencies Resolved

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

Transaction Summary

```
=====
Install                1 Package
```

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

1/1

complete

```
Installed:
  splunkforwarder.x86_64 0:6.2.3-264376
```

Complete!

bash-4.2#

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

例：

```
bash-4.2# yum info splunkforwarder
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
Fretta                | 951 B      00:00 ...
groups-repo           | 1.1 kB     00:00 ...
localdb               | 951 B      00:00 ...
patching              | 951 B      00:00 ...
thirdparty            | 951 B      00:00 ...
Installed Packages
Name      : splunkforwarder
Arch      : x86_64
Version   : 6.2.3
Release   : 264376
```

```

Size           : 34 M
Repo           : installed
From repo      : open-nxos
Summary        : SplunkForwarder
License        : Commercial
Description    : The platform for machine data.

```

署名付き RPM のインストール

署名済み RPM の確認

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

Run, **rpm -K rpm_file_name**

署名付き RPM ではありません

```

bash-4.2# rpm -K bgp-1.0.0-r0.lib32_n9000.rpm

bgp-1.0.0-r0.lib32_n9000.rpm: (sha1) dsa sha1 md5 OK

```

署名付き RPM です

```

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

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

bash-4.2#

```

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

```

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

Setting up Install Process

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

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

Error: Nothing to do

```

キーの手動インポートによる署名付き RPM のインストール

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


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

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

- 次のコマンドを使用して、キーをインポートします。

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

```
bash-4.2#
```

```
bash-4.2# rpm -q gpg-pubkey
```

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

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

```
bash-4.2#
```

```
bash-4.2# rpm -q gpg-pubkey
```

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

- yum コマンドを使用して署名付き RPM をインストールする

```
bash-4.2#
```

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

```
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
```

```
groups-repo          | 1.1 kB      00:00 ...
```

```
localdb              | 951 B       00:00 ...
```

```
patching             | 951 B       00:00 ...
```

```
thirdparty           | 951 B       00:00 ...
```

```
Setting up Install Process
```

```
Examining puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm:
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64
```

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

```
Resolving Dependencies
```

```
--> Running transaction check
```

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

```
--> Finished Dependency ResolutionDependencies Resolved
```

```
=====
```

```

Package Arch Version Repository
Size

=====

Installing:

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

Transaction Summary

=====

Install 1 Package

Total size: 46 M

Installed size: 46 M

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

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

Installed:

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

Complete!

bash-4.2#
```

キーの自動インポートによる署名付きサードパーティ RPM のインストール

キーと RPM を指すように yum リポジトリをセットアップします。

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

```
[puppet]
```

```

name=Puppet RPM

baseurl=file:///bootflash/puppet

enabled=1

gpgcheck=1

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

metadata_expire=0

cost=500

bash-4.2# yum install puppet-enterprise

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages

groups-repo                               | 1.1 kB      00:00 ...
localdb                                   |   951 B     00:00 ...
patching                                  |   951 B     00:00 ...
puppet                                    |   951 B     00:00 ...
thirdparty                               |   951 B     00:00 ...

Setting up Install Process

Resolving Dependencies

--> Running transaction check

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

--> Finished Dependency Resolution

Dependencies Resolved

=====

Package                Arch      Version                               Repository      Size
=====

Installing:

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

Transaction Summary

=====

Install      1 Package

Total download size: 14 M

```

```

Installed size: 46 M

Is this ok [y/N]: y

Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs

Importing GPG key 0x4BD6EC30:

  Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"
  From   : /bootflash/RPM-GPG-KEY-puppetlabs

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction

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

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

Do you wish to continue?

Is this ok [y/N]: y

Warning: RPMDB altered outside of yum.

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

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

Installed:

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

Complete!

```

リポジトリへの署名済み RPM の追加

手順

ステップ 1 署名済み RPM をリポジトリディレクトリにコピーする

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

```

bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm  RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# rpm --import RPM-GPG-KEY-puppetlabs
bash-4.2# createrepo .

```

```
1/1 - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm
Saving Primary metadata
Saving file lists metadata
Saving other metadata
bash-4.2#
```

キーをインポートしない場合

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

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

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

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

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

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

永続的なサードパーティ RPM

次に、永続的なサードパーティ RPM の背後にあるロジックを示します。

- ローカルリポジトリは、永続的なサードパーティ RPM 専用です。**yum /etc/yum/repos.d/ thirdparty.repo** は `/bootflash/.rpmstore/ thirdparty` を指します。
- コマンドを入力するたびに、RPM のコピーが `/bootflash/.rpmstore/ thirdparty` に保存されます。**yum install third-party.rpm**
- リブート中に、サードパーティ製リポジトリ内のすべてのRPMがスイッチに再インストールされます。
- `/etc` 構成ファイルの変更は、`/bootflash/.rpmstore/config/etc` の下に保持され、`/etc` での起動時に再生されます。

- /etc ディレクトリに作成されたスクリプトは、リロード後も保持されます。たとえば、/etc/init.d/ の下に作成されたサードパーティのサービススクリプトは、リロード中にアプリケーションを起動します。



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

変更した iptables を永続化するには、「」を参照してください。
[リロード間で Iptable を永続化する \(238 ページ\)](#)

VSH からの RPM のインストール

パッケージの追加

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

手順

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

例

次に、Chef RPM をアクティブにする例を示します：

```
switch# show install package
switch# install add ?
WORD          Package name
bootflash:    Enter package uri
ftp:          Enter package uri
http:         Enter package uri
modflash:     Enter package uri
scp:          Enter package uri
sftp:         Enter package uri
tftp:         Enter package uri
usb1:         Enter package uri
```

```
usb2:      Enter package uri
volatile:  Enter package uri
switch# install add
bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.e15.x86_64.rpm
[#####] 100%
Install operation 314 completed successfully at Thu Aug 6 12:58:22 2015
```

次のタスク

パッケージをアクティブ化する準備ができたなら、[パッケージのアクティブ化（143 ページ）](#) に移動します。



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

```
switch#
install add bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.e15.x86_64.rpm
activate
```

パッケージのアクティブ化

始める前に

RPM は事前に追加しておく必要があります。

手順

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

例

次に、パッケージをアクティブ化する例を示します：

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

Inactive Packages:
  sysinfo-1.0.0-7.0.3.x86_64
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
Available Packages
chef.x86_64          12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.e15  thirdparty
eigrp.lib32_n9000  1.0.0-r0                                              groups-rep
```

```

○
sysinfo.x86_64      1.0.0-7.0.3
switch# install activate chef-12.0-1.el5.x86_64.rpm
[#####] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015

```

patching

パッケージの非アクティブ化

手順

	コマンドまたはアクション	目的
ステップ 1	install deactivate <i>package-name</i>	RPM パッケージを非アクティブ化します。

例

次に、Chef RPM パッケージを非アクティブ化する例を示します。

```
switch# install deactivate chef
```

パッケージの削除

始める前に

パッケージを削除する前に非アクティブ化します。非アクティブ化された RPM パッケージのみ削除できます。

手順

	コマンドまたはアクション	目的
ステップ 1	install remove <i>package-name</i>	RPM パッケージを削除します。

例

次に、Chef RPM パッケージを削除する例を示します。

```
switch# install remove chef-12.0-1.el5.x86_64.rpm
```


インストール済みパッケージの表示

手順

	コマンドまたはアクション	目的
ステップ 1	show install packages	インストールされているパッケージのリストを表示します。

例

次に、インストールされているパッケージのリストを表示する例を示します。

```
switch# show install packages
```

詳細ログの表示

手順

	コマンドまたはアクション	目的
ステップ 1	show tech-support install	詳細ログを表示します。

例

次の例は、詳細ログを表示する方法を示しています。

```
switch# show tech-support install
```

パッケージのアップグレード

手順

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

例

次に、パッケージをアップグレードする例を示します：

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate ?
downgrade Downgrade package
forced      Non-interactive
upgrade     Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate upgrade
[#####] 100%
Install operation completed successfully at Thu Aug 6 12:46:53 2015
```

パッケージのダウングレード

手順

	コマンドまたはアクション	目的
ステップ 1	インストール 追加 <i>package-name</i> アクティベート ダウングレード	パッケージをダウングレードします。

例

次の例は、パッケージをダウングレードする方法を示しています。

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate ?
downgrade Downgrade package
forced      Non-interactive
upgrade     Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate downgrade
[#####] 100%
Install operation completed successfully at Thu Aug 6 12:46:53 2015
```

サードパーティ製アプリケーション

NX-OS

他のサードパーティ製アプリケーションの Cisco NX-OS リポジトリの詳細については、https://devhub.cisco.com/artifactory/open-nxos/7.0-3-I2-1/x86_64/ を参照してください。

NX-API REST API オブジェクトモデルの仕様の詳細については、<https://developer.cisco.com/docs/nx-api-dme-model-9-3-1-reference/> を参照してください。

DevOps 構成管理ツール

DevOps 構成管理ツールについては、次のリンクを参照してください。

- Ansible 2.0 リリース (Nexus サポート)、[Ansible リリース インデックス](#)

- Ansible NX-OS サンプル モジュール、[Ansible NX-OS サンプル モジュール](#)
- Puppet、[Puppet Forge Cisco Puppet](#)
- Cisco Puppet モジュール (Git) [Cisco Network Puppet モジュール](#)
- Chef、[Chef Supermarket Cisco クックブック](#)
- Cisco Chef クックブック (Git) [Cisco Network Chef クックブック](#)

V9K

ESX5.1/5.5、VirtualBox、Fusion、および KVM の場合、仮想 Nexus 9000 スイッチをダウンロードするには、<https://software.cisco.com/portal/pub/download/portal/select.html?&mdfid=286312239&flowid=81422&softwareid=282088129>に移動します。

自動化ツールの教育コンテンツ

OpenNX-OS アーキテクチャと自動化に関する無料の書籍については、次を参照してください。
http://www.cisco.com/c/dam/en/us/td/docs/switches/datacenter/nexus9000/sw/open_nxos/programmability/guide/Programmability_Open_NX-OS.pdf

collectd

collectd は、システムパフォーマンスの統計情報を定期的に収集し、RRD ファイルなどの値を保存する複数の手段を提供するデーモンです。これらの統計情報を使用して、現在のパフォーマンスのボトルネック（パフォーマンス分析など）を見つけたり、将来のシステム負荷を予測したりできます（つまり、キャパシティ プランニング）。

詳細については、<https://collectd.org> を参照してください。

Ganglia

Ganglia は、クラスタやグリッドなどのハイパフォーマンス コンピューティング システム向けのスケーラブルな分散モニタリングシステムです。これは、クラスタのフェデレーションを対象とした階層設計に基づいています。データ表現のための XML、コンパクトでポータブルなデータ転送のための XDR、データストレージと可視化のための RRDtool など、広く使用されているテクノロジーを活用します。設計されたデータ構造とアルゴリズムを使用して、ノードあたりのオーバーヘッドを低く抑え、同時実行性を高めます。この実装は堅牢であり、広範なオペレーティングシステムとプロセッサアーキテクチャに移植されており、現在、世界中の何千ものクラスタで使用されています。世界中の大学キャンパス間でクラスタをリンクするために使用されており、2000 ノードのクラスタを処理するように拡張できます。

詳細については、<http://ganglia.info> を参照してください。

Iperf

Iperfは、TCP およびUDP の最大帯域幅パフォーマンスを測定するためにNLANR/DASTによって開発されました。Iperfを使用すると、さまざまなパラメータとUDP 特性を調整できます。Iperfは、帯域幅、遅延ジッターとデータグラム損失を報告します。

詳細については、<http://sourceforge.net/projects/iperf/>または<http://iperf.sourceforge.net>を参照してください。

LLDP

リンク層検出プロトコル（LLDP）は、EDPやCDPなどの独自のリンク層プロトコルに代わるように設計された業界標準プロトコルです。LLDPの目的は、隣接するネットワークデバイスにリンク層通知を配信するための、ベンダー間互換性のあるメカニズムを提供することです。

詳細については、「<https://vincentbernat.github.io/lldpd/index.html>」を参照してください。

Nagios

Nagiosは、Nagios Remote Plug-in Executor（NRPE）およびSSHまたはSSLトンネルを介して以下をモニターするオープンソースソフトウェアです。

- ICMP、SNMP、SSH、FTP、HTTP などによるネットワーク サービス
- CPU 負荷、ディスク使用率、システム ログなどのホスト リソース
- サーバー、スイッチ、アプリケーションのアラート サービス
- [サービス（Services）]

詳細については、「<https://www.nagios.org/>」を参照してください。

OpenSSH

OpenSSHは、盗聴、接続ハイジャック、およびその他の攻撃を排除するために、すべてのトラフィック（パスワードを含む）を暗号化するSSH 接続ツールのオープンソースバージョンです。OpenSSHは、セキュアなトンネリング機能と複数の認証方式を提供し、すべてのSSH プロトコルバージョンをサポートします。

詳細については、「<http://www.openssh.com>」を参照してください。

Quagga

Quaggaは、さまざまなルーティングプロトコルを実装するネットワークルーティングソフトウェアスイートです。Quaggaデーモンは、ネットワークアクセス可能CLI（「vty」という）を使用して構成できます。



(注) Quagga BGP のみが検証されています。

詳細については、「<http://www.nongnu.org/quagga/>」を参照してください。

スプラunk

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

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

tccollector

tccollector は、ローカルコレクタからデータを収集し、そのデータをオープン時系列データベース (OpenTSDB) にプッシュするクライアント側プロセスです。

tccollector には次の機能があります。

- データ コレクタを実行し、データを照合します。
- 時系列データベース (TSD) への接続を管理します。
- コレクタに TSD コードを埋め込む必要がなくなります。
- 繰り返される値の重複を排除します。
- ワイヤプロトコル作業を処理します。

詳細については、http://opentsdb.net/docs/build/html/user_guide/utilities/tcollector.html を参照してください。

tcpdump

tcpdump は、ネットワーク インターフェイス上の Boolean 式に一致するパケットの内容に関する説明を出力する CLI アプリケーションです。説明の前にタイムスタンプが表示されます。デフォルトでは、午前 0 時からの時間、分、秒、および小数点以下の秒として出力されます。

tcpdump は、次のフラグを使用して実行できます。

- **-w** : 後で分析するためにパケット データをファイルに保存します。

- **-r** : ネットワーク インターフェイスからパケットを読み取るのではなく、保存されたパケット ファイルから読み取ります。
- **-V** : 保存されたパケット ファイルのリストを読み取ります。

いずれの場合も、**tcpdump** は式にマッチするパケットだけを処理します。

詳細については、「<http://www.tcpdump.org/manpages/tcpdump.1.html>」を参照してください。

TShark

TShark は、CLI のネットワーク プロトコル アナライザです。Tshark を使用すると、ライブ ネットワークからパケット データをキャプチャしたり、以前に保存したキャプチャ ファイルからパケットを読み取ったりできます。これらのパケットのデコードされた形式を標準出力に出力するか、パケットをファイルに書き込むことができます。TShark のネイティブ キャプチャ ファイル フォーマットは、**pcap** です。このフォーマットは、**tcpdump** と他のツールに使用されています。TShark は、**cap_net_admin** ファイル機能を削除した後、ゲストシェル内で使用できます。

```
setcap
cap_net_raw=ep /sbin/dumpcap
```



(注) このコマンドは、ゲストシェル内で実行する必要があります。

詳細については、「<https://www.wireshark.org/docs/man-pages/tshark.html>」を参照してください。



第 12 章

Ansible

- 前提条件 (151 ページ)
- アンシブルについて (151 ページ)
- Cisco Ansible モジュール (152 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/latest/getting_started/index.html を参照してください。

アンシブルについて

Ansible は、クラウドプロビジョニング、構成管理、アプリケーションの展開、サービス内オーケストレーション、およびその他の IT ニーズを自動化するオープンソースの IT 自動化エンジンです。

Ansible は、Ansible モジュールと呼ばれる小さなプログラムを使用してノードへの API 呼び出しを行い、Playbook で定義された構成を適用します。

デフォルトでは、Ansible は、すべての管理対象マシンを独自に選択したグループに入れる単純な INI ファイルを使用して、管理対象のマシンを表します。

詳細については、Ansible から入手できます。

Ansible	https://www.ansible.com/
Ansible 自動化ソリューションインストール手順、プレイブックの手順と例、モジュールリストなどが含まれます。	https://docs.ansible.com/

Cisco Ansible モジュール

次のリンクの表に示すように、Ansible には複数の Cisco NX-OS でサポートされるモジュールとプレイブックがあります。

NX-OS デベロッパーのランディング ページ。	構成管理ツール
Ansible NX-OS プレイブックの例	Ansible nxos Playbook のリポジトリ
Ansible NX-OS ネットワーク モジュール	nxos ネットワーク モジュール



第 13 章

Puppet Agent

この章は次のトピックで構成されています。

- [Puppet について \(153 ページ\)](#)
- [前提条件 \(154 ページ\)](#)
- [Puppet エージェント NX-OS 環境 \(154 ページ\)](#)
- [ciscopuppet モジュール \(154 ページ\)](#)

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/blog/how-get-started-puppet-enterprise-faq/
Puppet Labs ドキュメント	https://puppet.com/docs

前提条件

Puppet エージェントの前提条件は次のとおりです。

- 仮想サービスのインストールと Puppet Agent の展開に必要なディスクストレージをデバイスで利用できる必要があります。
- ブートフラッシュに最低 450MB の空きディスク容量。
- Puppet 4.0 以降の Puppet プライマリ サーバーが必要です。
- Puppet エージェント 4.0 以降が必要です。

Puppet エージェント NX-OS 環境

Puppet Agent ソフトウェアは、ゲストシェル（CentOS を実行する Linux コンテナ環境）のスイッチにインストールする必要があります。ゲストシェルは、ホストから切り離された安全でオープンな実行環境を提供します。

Cisco NX-OS リリース 9.2（1）以降、Puppet Agent の Bash-shell（Cisco NX-OS の基盤となるネイティブ WindRiver Linux 環境）インストールはサポートされなくなりました。

次に、エージェントソフトウェアのダウンロード、インストール、およびセットアップに関する情報を示します：

Puppet Agent : Cisco Nexus スイッチでのインストールとセットアップ（手動セットアップ）	https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md

ciscopuppet モジュール

ciscopuppet モジュールは、Cisco が開発したオープン ソース ソフトウェア モジュールです。これは、Puppet マニフェストの抽象技術情報構成と、Cisco NX-OS オペレーティングシステムおよびプラットフォームの特定の実装の詳細との間のインターフェイスとなります。このモジュールは Puppet プライマリにインストールされ、Cisco Nexus スイッチでの Puppet エージェントの操作に必要です。

ciscopuppet モジュールは、Puppet Forge で利用できます。

ここでは、ciscopuppet モジュール インストール手順についての追加情報を提供します：

ciscopuppet モジュールの場所 (Puppet Forge)	Puppet Forge

リソースの種類のカatalog	[Cisco Puppet 技術情報の参照先 (Cisco Puppet Resource Reference)]
ciscopuppet モジュール：送信元コードリポジトリ	[Cisco Network Puppet モジュール (Cisco Network Puppet Module)]
ciscopuppet モジュール：セットアップと使用法	Cisco Puppet モジュール::README.md
Puppet Labs: モジュールのインストール	https://puppet.com/docs/puppet/7/modules_installing.html
Puppet NX-OS マニフェストの例	[Cisco Network Puppet モジュールの例 (Cisco Network Puppet Module Examples)]
NX-OS デベロッパーのランディング ページ。	構成管理ツール



第 14 章

SaltStack

この章は次のトピックで構成されています。

- [SaltStack について](#) (157 ページ)
- [注意事項と制約事項](#) (158 ページ)
- [SaltStack の Cisco NX-OS 環境](#) (159 ページ)
- [SaltStack の NX-API の有効化](#) (159 ページ)
- [NX-OS 用の SaltStack のインストール](#) (160 ページ)

SaltStack について

Cisco Nexus スイッチは、NX-OS を介して SaltStack をサポートします。Cisco NX-OS のサポートされているリリースの詳細については、<https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst#step-1-verify-platform-and-software-version-support> を参照してください。

SaltStack は、サーバーやその他のネットワーク デバイスの構成、管理、およびリモート実行のための無料のオープンソース自動化フレームワークです。SaltStack フレームワークは、Salt プライマリと呼ばれるサーバーと、ミニオンと呼ばれるクライアントプログラムを実行する Salt ノードで構成されます。Cisco Nexus スイッチ (スイッチ) は、Salt ノードであり、Salt プライマリではありません。

SaltStack ミニオンは、スイッチに応じてオン ボックスまたはオフ ボックスのいずれかで実行して、構成または管理操作を実行できます。

- オンボックスでは、ミニオンはスイッチの Bash シェルで実行されます。これらのネイティブ ミニオンは、プライマリからリモート コマンドを受信して実行し、コマンドの結果をプライマリに中継します。オンボックス展開では、ミニオンはスイッチの Guest Shell で有効になっています。
- オフボックスでは、別のタイプのミニオンであるプロキシミニオンが、スイッチへの SSH 接続または NX-API を介して実行されます。プロキシミニオン (SSH プロキシミニオンまたは NX-API プロキシミニオン) は、コマンドを受信して実行します。その後、プロキシはコマンドの結果をプライマリに中継します。

鍵は、Salt プライマリと Cisco Nexus スイッチで実行されているミニオンとの間のセキュリティを確保するために使用されます。Salt のプライマリは、Cisco Nexus スイッチで実行されているミニオンとの接続を開始すると、最初にキーを渡します。ミニオンはキーを受け取り、正しい応答を計算し、キーをプライマリに送信します。プライマリは、キーの正しい応答値も計算しました。プライマリがミニオンからキーを受け取ると、キーが一致すると、セッションが開かれます。これにより、Salt プライマリはコマンドを送信できます。セッションは、電源を入れ直したり再起動したりしても持続しません。

SaltStack は、スイッチの CLI、プロパティ、および機能に影響を与える実行モジュールとソルトの状態を通じて、スイッチを管理および構成します。たとえば、モジュールを通じて、SaltStack を使用して Cisco Nexus スイッチをアップグレードできます。Salt プライマリは、自動化とスケーラビリティを活用するためにプログラムでコマンドを送信します。

詳細については、次のドキュメントを参照してください。

SaltStack	https://www.saltstack.com/
SaltStack のドキュメント	https://docs.saltstack.com/en/latest/
Cisco Nexus Salt Minion のインストールおよび構成ガイド	https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst

NX-OS と SaltStack について

Salt Open は、Salt 構成管理および分散リモート実行システムのオープンソースのコミュニティ版です。Cisco NX-OS は、物理スイッチと Salt Open ソフトウェアの間に中間層を提供します。Cisco NX-OS と Salt Open は相互運用して、Salt ミニオンと Cisco Nexus スイッチの間に API とコマンド実行レイヤを提供します。Cisco NX-OS はミニオンをホストし、次のように実行できるようにします。

- スイッチでは、Cisco NX-OS ゲストシェルが SaltStack ミニオンをホストし、統合インターフェイスを介して 1 つ以上のスイッチの自動オーケストレーションを提供します。ゲストシェルで実行されているミニオンはネイティブ ミニオンであり、NX-API を介して UNIX ドメイン ソケット (UDS) に接続します。
- スイッチをオフにすると、Salt プライマリはネットワーク デバイス上で Salt Open ソフトウェアを実行し、SSH (SSH プロキシ ミニオン) または NX-API over HTTPS (NX-API プロキシ ミニオン) を介して NX-OS と通信します。Cisco NX-OS はコマンドを解釈し、必要な構成タスクを実行し、成功または失敗を適切なプロキシミニオンに報告します。プロキシミニオンは、このデータを Salt プライマリに送信します。

注意事項と制約事項

Cisco Nexus スイッチに SaltStack の導入の注意事項と制約事項は次のとおりです：

- プラットフォーム サポートについての注意は、[Nexus Switch Platform Support Matrix](#) を参照してください。
- SSH または NX-API HTTPS を介して SaltStack を実行している場合は、Salt を実行する前に NX-API 機能 (**feature nxapi**) を有効にします。
- Salt のプライマリは、ポート 4506 でミニオンをリスンします。このポートが開いていて (ブロックされていない) 、別のサービスで使われていないことを確認してください。

SaltStack の Cisco NX-OS 環境

Cisco NX-OS 環境は、Salt をオンボックスで実行しているかオフボックスで実行しているかによって異なります。

- スイッチをオンボックスで管理するには、ミニオンのホスティング環境であるゲストシェルに SaltStack ミニオン RPM をインストールする必要があります。
- スイッチをオフボックスで管理するには、NX-OS で SSH または NX-API を有効にする必要があります。

SaltStack をサポートする Cisco Nexus スイッチなどの詳細については、<https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst#step-1-verify-platform-and-software-version-support> にアクセスしてください。

SaltStack の NX-API の有効化

始める前に

SSH または NX-API HTTPS で実行されているプロキシ ミニオンの場合、SaltStack が機能するには NX-API 機能を有効にする必要があります。デフォルトでは、NX-API が有効になっています。再度有効にする必要がある場合に備えて、次の手順を示します。

手順

	コマンドまたはアクション	目的
ステップ 1	config terminal 例 : <pre>switch-1# config terminal Enter configuration commands, one per line. End with CNTL/Z. switch-1(config)#</pre>	コンフィギュレーション モードに入ります。

	コマンドまたはアクション	目的
ステップ 2	feature nxapi 例 : switch-1# feature nxapi switch-1#(config)#	プロキシ ミニオンの NX-API を有効にします。

次のタスク

ソルトスタックをインストールします。

NX-OS 用の SaltStack のインストール

次の設置ガイドを使用して、Cisco Nexus スイッチに SaltStack をインストールして起動します。

<https://github.com/saltstack/salt/blob/develop/doc/topics/installation/nxos.rst#cisco-nexus-salt-minion-installation-and-configuration-guide>



第 15 章

Cisco NX-OS でのシェフクライアントの使用

- [シェフについて](#) (161 ページ)
- [前提条件](#) (162 ページ)
- [Chef クライアント NX-OS 環境](#) (162 ページ)
- [cisco-cookbook](#) (163 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。ソフトウェアパッケージは、インフラストラクチャのサイズに関係なく、物理、仮想、またはクラウドの場所にサーバーとアプリケーションを導入する、システムおよびクラウドインフラストラクチャの自動化フレームワークです。各組織は、1 つ以上のワークステーション、単一サーバー、Chef クライアントが設定されていて、維持されているすべてのノードで構成されます。各ノードの設定方法について Chef クライアントに指示するために、クックブックとレシピが使用されます。すべてのノードにインストールされている Chef クライアントが、実際の設定を行います。

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

次のリファレンスは、Chef からの詳細情報を提供します。

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

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：

- シェフの展開に必要なディスク ストレージがデバイス上に用意されている必要があります：
- ブートフラッシュに最低 500 MB の空きディスク容量
- シェフ 12.4.1 以降のシェフ サーバが必要です。
- シェフ クライアント 12.4.1 以降が必要です。

Chef クライアント NX-OS 環境

chef-client ソフトウェアは、ゲストシェル（CentOS を実行する Linux コンテナ環境）のスイッチにインストールする必要があります。このソフトウェアは、ホストから切り離された安全でオープンな実行環境を提供します。

Cisco NX-OS リリース 9.2(1) 以降、chef-client の Bash-shell（NX-OS の基盤となるネイティブ WindRiver Linux 環境）インストールはサポートされなくなりました。

次のドキュメントには、エージェントソフトウェアのダウンロード、インストール、および手順ごとのガイダンスが記載されています。

トピック	リンク
Chef クライアント（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。
Chef クライアント：Cisco Nexus プラットフォームでのインストールとセットアップ（手動セットアップ）	cisco-cookbook::README-install-agent.md
Chef クライアント：スイッチでのインストールとセットアップ（Chef プロビジョナを使用した自動インストール）	cisco-cookbook::README-chef-provisioning.md
NX-OS デベロッパーのランディング ページ	https://www.in-github.cisco.com/agents/DevNet-Config-Management

cisco-cookbook

cisco-cookbook は、Chef レシピの抽象情報技術構成と、Cisco Nexus スイッチの特定の実装の詳細との間の、Cisco が開発したオープン ソース インターフェイスです。このクックブックは Chef Server にインストールされ、Cisco Nexus スイッチでの Chef Client の適切な動作に必要です。

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

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

トピック	リンク
cisco-cookbook の場所	Chef Supermarket Cisco クックブック
リソースの種類のカatalog	リソースカatalog (テクノロジー別)
cisco-cookbook: ソース コード リポジトリ	Cisco Network Chef クックブック
cisco-cookbook: セットアップと使用法	Chef クックブックの設定と使用方法
Chef Supermarket	Chef Supermarket
Puppet NX-OS マニフェストの例	Cisco Network Chef クックブックのレシピ



第 16 章

Nexus アプリケーション開発 : Yocto

この章は、次の項で構成されています。

- [Yocto について](#) (165 ページ)
- [Yocto のインストール](#) (165 ページ)

Yocto について

Cisco NX-OS リリース 9.2(1) ソフトウェアは、Yocto 2.0 に基づいています。Yocto 2.0 をダウンロードし、構築する新しいソフトウェアをダウンロードし、ソフトウェアを構築し、ソフトウェアをスイッチにインストールすることで、より多くのアプリケーションをインストールできます。

Yocto のインストール

次の例では、Ubuntu 16.04 仮想マシンで Ruby バージョン 2.2.2 をビルドします。

手順

ステップ 1 Ubuntu 16.04 仮想マシンにすべての必須パッケージをインストールします。

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential  
chrpath socat cpio python python3 python3-pip python3-pexpect xz-utils debianutils  
iputils-ping libssl1.2-dev xterm
```

ステップ 2 Yocto 2.0 をダウンロードします。

```
wget http://downloads.yoctoproject.org/releases/yocto/yocto-2.0/poky-jethro-14.0.0.tar.bz2  
tar -xjfv poky-jethro-14.0.0.tar.bz2  
cd poky-jethro-14.0.0
```

ステップ 3 oe-init-build-env ファイルを取得します。

```
source oe-init-build-env
```

ステップ 4 テキスト エディタを使用して conf/local.conf を編集し、次の行を追加します。

```
MACHINE = "genericx86-64"
DEFAULTTUNE = "x86-64"
```

ステップ 5 次のコマンドを入力します。

```
bitbake ruby
```

ビルドが完了すると、RPM は tmp/deploy/rpm/x86_64/*.rpm に生成されています。

ステップ 6 RPM をスイッチにコピーします。

```
Switch# copy scp://<username>@<IP_address>/ruby-2.2.2-r0.x86_64.rpm bootflash: vrf
management use-kstack
Switch# copy scp://<username>@<IP_address>/libyaml-0-2-0.1.6-r0.x86_64.rpm bootflash:
vrf management use-kstack
```

ステップ 7 Bash シェルから、次のコマンドを入力します。

インストール プロセスのある時点で求められたら、**y** を入力します。

```
bash-4.3# yum install /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
groups-repo                | 1.1 kB      00:00 ...
localdb                    | 951 B       00:00 ...
patching                   | 951 B       00:00 ...
thirdparty                 | 951 B       00:00 ...
Setting up Install Process
Examining /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm: libyaml-0-2-0.1.6-r0.x86_64
Marking /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package libyaml-0-2.x86_64 0:0.1.6-r0 will be installed
--> Finished Dependency Resolution
```

Dependencies Resolved

```
=====
Package Arch Version Repository
Size
=====
Installing:
libyaml-0-2 x86_64 0.1.6-r0 /libyaml-0-2-0.1.6-r0.x86_64 119
k
```

Transaction Summary

```
=====
Install 1 Package
```

Total size: 119 k

Installed size: 119 k

Is this ok [y/N]: **y**

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

Installing : libyaml-0-2-0.1.6-r0.x86_64

1/1

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

```

Installed:
  libyaml-0-2-0.1.6-r0.x86_64

Complete!
Install operation 2450 completed successfully at Fri Jul 27 18:54:55 2018.

[#####] 100%

```

ステップ 8 次のコマンドは、Ubuntu 16.04 仮想マシンで Ruby バージョン 2.2.2 をビルドする例を示しています。

インストールプロセスのある時点で求められたら、**y** を入力します。

```

bash-4.3# yum install /bootflash/ruby-2.2.2-r0.x86_64.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
groups-repo           | 1.1 kB      00:00 ...
localdb               | 951 B       00:00 ...
patching              | 951 B       00:00 ...
thirdparty            | 951 B       00:00 ...
thirdparty/primary    | 1.8 kB      00:00 ...
thirdparty            2/2
Setting up Install Process
Examining /bootflash/ruby-2.2.2-r0.x86_64.rpm: ruby-2.2.2-r0.x86_64
Marking /bootflash/ruby-2.2.2-r0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ruby.x86_64 0:2.2.2-r0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====

Package Arch Version Repository
Size
=====
Installing:
ruby x86_64 2.2.2-r0 /ruby-2.2.2-r0.x86_64 32
M

Transaction Summary
=====
Install 1 Package

Total size: 32 M
Installed size: 32 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Installing : ruby-2.2.2-r0.x86_64 1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link

Installed:
  ruby.x86_64 0:2.2.2-r0

```

```
Complete!  
Install operation 2451 completed successfully at Fri Jul 27 18:55:23 2018.  
  
[#####] 100%
```



第 17 章

Nexus アプリケーション開発：SDK

この章は、次の内容で構成されています。

- [Cisco SDK について \(169 ページ\)](#)
- [SDK のインストール \(169 ページ\)](#)
- [インストールと環境の初期化の手順 \(170 ページ\)](#)
- [SDK を使用したアプリケーションの構築 \(171 ページ\)](#)
- [RPM を使用したアプリケーションのパッケージ化 \(173 ページ\)](#)
- [RPM ビルド環境の作成 \(173 ページ\)](#)
- [一般的な RPM ビルド手順の使用 \(174 ページ\)](#)
- [オプションのプラグインを使用しない collectd RPM の構築例 \(175 ページ\)](#)
- [オプションの Curl プラグインを使用した collectd の RPM のビルド例 \(176 ページ\)](#)

Cisco SDK について

Cisco SDK は、Yocto 2.0 に基づく開発キットです。Cisco NX-OS リリース 9.2(1) が動作する Cisco Nexus スイッチで実行するアプリケーションをビルドするためのすべてのツールが含まれています。基本コンポーネントは、多くのアプリケーションで一般的に使用される C クロスコンパイラ、リンカ、ライブラリ、およびヘッダーファイルです。リストは網羅的ではないため、特定のアプリケーションに必要な依存関係をダウンロードしてビルドすることが必要となる場合があります。一部のアプリケーションは、Cisco devhub Web サイトからダウンロードしてすぐに使用できるようになっていて、ビルドの必要はありません。SDK は、スイッチに直接インストールできる RPM パッケージをビルドするために使用できます。

SDK のインストール

以下にシステム要件を示します。

- SDK は、ほとんどの最新の 64 ビット x86_64 Linux システムで実行できます。CentOS 7 および Ubuntu 14.04 で検証済みです。Bash シェルで SDK をインストールして実行します。

- SDK には、32 ビットアーキテクチャと 64 ビットアーキテクチャの両方のバイナリが含まれているため、32 ビットライブラリもインストールされている x86_64 Linux システムで実行する必要があります。

手順

32 ビットライブラリがインストールされているかどうかを確認します。

例：

```
bash$ ls /lib/ld-linux.so.2
```

このファイルが存在する場合は、32 ビットライブラリがすでにインストールされています。それ以外の場合は、次のように 32 ビットライブラリをインストールします。

- CentOS 7 の場合：

```
bash$ sudo yum install glibc.i686
```

- Ubuntu 14.04 の場合：

```
bash$ sudo apt-get install gcc-multilib
```

インストールと環境の初期化の手順

SDK は https://devhub.cisco.com/artifactory/open-nxos/9.2.1/wrlinux-8.0.0.25-glibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh からダウンロードできます。

このファイルは自己解凍アーカイブで、SDK を任意のディレクトリにインストールできます。SDK のインストールディレクトリへのパスの入力が求められます。

```
bash$ ./wrlinux-8.0.0.25-glibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Wind River Linux SDK installer version 8.0-n9000
=====
Enter target directory for SDK (default: /opt/windriver/wrlinux/8.0-n9000):
You are about to install the SDK to "/opt/windriver/wrlinux/8.0-n9000". Proceed[Y/n]? Y
Extracting
SDK.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

新しいシェルセッションで SDK を使用するたびに、source コマンドで環境セットアップスクリプトを読み込む必要があります。次に例を示します。

```
$ . /opt/windriver/wrlinux/8.0-n9000/environment-setup-x86-wrsmlib32-linux
$ . /opt/windriver/wrlinux/8.0-n9000/environment-setup-x86_64-wrs-linux
bash$
```

`source environment-setup-x86-wrsmlib32-linux` および `source environment-setup-x86_64-wrs-linux` コマンドを使用して、シェル環境に SDK 固有のパスを追加します。SDK で使用する予定のシェルごとに、SDK 固有のパスを追加します。SDK 固有のパスを追加することは、ビルドツールとライブラリの正しいバージョンを使用するように SDK をセットアップする点で重要です。

手順

ステップ 1 インストール先ディレクトリを参照します。

ステップ 2 Bash プロンプトで次のコマンドを入力します。

```
bash$ source environment-setup-x86-wrsmlib32-linux
bash$ source environment-setup-x86_64-wrs-linux
```

SDK を使用したアプリケーションの構築

一般的な Linux ビルドプロセスの多くは、このシナリオで機能します。状況に最適な方法を使用してください。

アプリケーションパッケージのソースコードは、さまざまな方法で取得できます。たとえば、tar ファイル形式で、またはパッケージが存在する git リポジトリからダウンロードして、ソースコードを取得できます。

次は最も一般的なケースの一例です。

(オプション) アプリケーションパッケージが標準の `configure/make/make install` を使用してビルドされることを確認します。

```
bash$ tar --xvzf example-app.tgz
bash$ mkdir example-lib-install
bash$ cd example-app/
bash$ ./configure --prefix=/path/to/example-app-install
bash$ make
bash$ make install
```

場合によっては、`./configure` スクリプトに追加のオプションを渡す必要があります。たとえば、必要なオプションのコンポーネントと依存関係を指定する場合などです。追加オプションを渡すかどうかは、構築するアプリケーションに完全に依存します。

例 : Ganglia とその依存関係の構築

この例では、ganglia と、必要なサードパーティライブラリ (libexpat、libapr、および libconfuse) を作成します。

libexpat

```
bash$ wget 'http://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz'
bash$ mkdir expat-install
bash$ tar xvzf expat-2.1.0.tar.gz
```

```
bash$ cd expat-2.1.0
bash$ ./configure --prefix=/home/sdk-user/expat-install
bash$ make
bash$ make install
bash$ cd ..
```

libapr

```
bash$ wget 'http://www.eu.apache.org/dist/apr/apr-1.5.2.tar.gz'
bash$ mkdir apr-install
bash$ tar xvzf apr-1.5.2.tar.gz
bash$ cd apr-1.5.2
bash$ ./configure --prefix=/home/sdk-user/apr-install
bash$ make
bash$ make install
bash$ cd ..
```

libconfuse



(注) confuse には、./configure に追加の --enable-shared オプションが必要です。そうでない場合、必要な共有ライブラリの代わりに静的にリンクされたライブラリが構築されます。

```
bash$ wget 'http://savannah.nongnu.org/download/confuse/confuse-2.7.tar.gz'
bash$ mkdir confuse-install
bash$ tar xvzf confuse-2.7.tar.gz
bash$ cd confuse-2.7
bash$ ./configure --prefix=/home/sdk-user/confuse-install --enable-shared
bash$ make
bash$ make install
bash$ cd ..
```

ganglia



(注) 必要なすべてのライブラリの場所が ./configure に渡されます。

```
bash$ wget
'http://downloads.sourceforge.net/project/ganglia/ganglia%20monitoring%20core/3.7.2/ganglia-3.7.2.tar.gz'
bash$ mkdir ganglia-install
bash$ tar xvzf ganglia-3.7.2.tar.gz
bash$ cd ganglia-3.7.2
bash$ ./configure --with-libexpat=/home/sdk-user/expat-install
--with-libapr=/home/sdk-user/apr-install/bin/apr-1-config
--with-libconfuse=/home/sdk-user/confuse-install --prefix=/home/sdk-user/ganglia-install
bash$ make
bash$ make install
bash$ cd ..
```

RPM を使用したアプリケーションのパッケージ化

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



(注) RPM および仕様ファイル

RPM パッケージ形式は、特定のアプリケーションの完全なインストールに必要なすべてのファイル（バイナリ、ライブラリ、設定、ドキュメントなど）をパッケージ化するように設計されています。したがって、RPM ファイルを作成するプロセスは簡単ではありません。RPM ビルドプロセスを支援するために、ビルドプロセスに関するすべてを制御する .spec ファイルが使用されます。



(注) 多くのサードパーティ製アプリケーションは、tarball にパッケージ化されたソースコードの形式でインターネット上で入手できます。多くの場合、これらの tarball には RPM ビルドプロセスに役立つ .spec ファイルが含まれています。残念ながら、これらの .spec ファイルの多くは、ソースコード自体ほど頻繁には更新されません。さらに悪いことに、spec ファイルがまったくない場合もあります。このような場合、RPM を構築できるように、仕様ファイルを編集または最初から作成する必要があります。

RPM ビルド環境の作成

SDK を使用して RPM をビルドする前に、RPM ビルドディレクトリ構造を作成し、いくつかの RPM マクロを設定する必要があります。

手順

ステップ1 ディレクトリ構造を作成します：

```
bash$ mkdir rpmbuild
bash$ cd rpmbuild
bash$ mkdir BUILD RPMS SOURCES SPECS SRPMS
```

ステップ2 上で作成したディレクトリ構造を指すように topdir マクロを設定します：

```
bash$ echo "_topdir ${PWD}" > ~/.rpmmacros
```

(注)

この手順は、現在のユーザーがすでに設定されている .rpmmacros ファイルを有していないことを前提としています。既存の .rpmmacros ファイルを変更するのが不便な場合は、すべての rpmbuild コマンドラインに次を追加できます。

```
--define "_topdir ${PWD}"
```

ステップ3 RPM DB を更新します。

```
bash$ rm /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm/__db.*
bash$ rpm --rebuilddb
```

(注)

SDK の rpm および rpmbuild ツールは、RPM データベースとして通常の /var/lib/rpm の代わりに /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm を使用するように変更されました。この変更により、SDK を使用していない場合にホストの RPM データベースと競合することが回避され、ルートアクセスの必要性がなくなります。SDK のインストール後、この手順に従って SDK RPM データベースを再構築する必要があります。

一般的な RPM ビルド手順の使用

一般的な RPM ビルド手順は次のとおりです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app tarball>
bash$ # determine location of spec file in tarball:
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

結果は RPMS/ に作成されるバイナリ RPM で、スイッチにコピーしてインストールできます。アプリケーションのインストール方法と構成には様々なバリエーションがあり得ます。これらの手順については、アプリケーションのドキュメントを参照してください。

この rpm ビルドとスイッチへのインストールは、アプリケーションをサポートするために必要なすべてのソフトウェアパッケージで必要です。SDK にまだ含まれていないソフトウェアの依存関係を満たすことが必要な場合は、ソースコードを取得して、依存関係のあるソフトウェアもビルドする必要があります。ビルド用のマシンでは、パッケージを手動でビルドして、依存関係を検証することができます。次に、最も一般的な手順の例を示します。

```
bash$ tar xkzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example-lib-install
bash$ make
bash$ make install
```

これらのコマンドは、ビルドファイル（バイナリ、ヘッダー、ライブラリなど）をインストールディレクトリに配置します。ここから、標準のコンパイラとリンカのフラグを使用して、依存関係を満たすための新しい場所を選択できます。ライブラリなどのランタイムコードがあれば、それらもスイッチにインストールする必要があるため、必要なランタイムコードを RPM にパッケージ化しなければなりません。



(注) Cisco devhub の Web サイトには、すでに RPM 形式にまとめられているサポート ライブラリが多数あります。

オプションのプラグインを使用しない collectd RPM の構築例

ソース tarball をダウンロードし、仕様ファイルを抽出します。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この tarball には 4 つの spec ファイルがあります。Red Hat 仕様ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

この仕様ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM を設定します。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用します。仕様ファイルで編集された以下を編集します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

collectd には多数のオプションプラグインがあります。この仕様ファイルは、デフォルトで多くのプラグインを有効にします。多くのプラグインには外部依存関係があるため、これらのプラグインを無効にするオプションをコマンドラインに渡す必要があります。rpmbuild 1 つの長いコマンドラインを入力する代わりに、次のように Bash 配列でオプションを管理できます。

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java memcached mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>   rpmbuild_opts+=("--without")
>   rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

その後、次のように rpmbuild に渡され、ビルドおよび RPM パッケージプロセス全体が開始されます。

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

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

これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
```

オプションの Curl プラグインを使用した collectd の RPM のビルド例

collectd curl プラグインには、依存関係として libcurl があります。

RPM ビルドプロセス中にこのリンクの依存関係を満たすには、SDK で curl をダウンロードしてビルドする必要があります。

```
bash$ wget --no-check-certificate http://curl.haxx.se/download/curl-7.24.0.tar.gz
bash$ tar xkvf curl-7.24.0.tar.gz
bash$ cd curl-7.24.0
bash$ ./configure --without-ssl --prefix /path/to/curl-install
bash$ make
bash$ make install
bash$ cd ..
```



(注) curl バイナリとライブラリは、/path/to/curl-install にインストールされます。このディレクトリが存在しない場合は作成されるため、現在のユーザーには書き込み権限が必要です。次に、ソース tarball をダウンロードし、spec ファイルを抽出します。この手順は、プラグインがない場合の collectd の例とまったく同じです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この spec ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM をセットアップします。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用する必要があるため、spec ファイルで次のように編集します。



- (注) この tarball には 4 つの spec ファイルがあります。Red Hat の spec ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

この点は、前の例と違ってしています。collectdrpmbuild プロセスは、libcurl の場所を認識する必要があります。collectd の spec ファイルを編集して、以下を追加します。

SPECS/collectd.spec で文字列 `%configure` を検索します。この行とそれに続く行は、rpmbuild が `./configure` スクリプトに渡すオプションを定義します。

次のオプションを追加します：

```
--with-libcurl=/path/to/curl-install/bin/curl-config \
```

次に、rpmbuild コマンド オプションを含む Bash アレイが再度構築されます。次の違いに留意してください。

- `curl` をビルドされないプラグインのリストから削除
- `--with curl=force` の追加

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>   rpmbuild_opts+=("--without")
>   rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=("--with")
bash$ rpmbuild_opts+=("curl=force")bash$ rpmbuild_opts+=("--nodeps)
bash$ rpmbuild_opts+=("--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

それからこれは次のように rpmbuild に渡され、ビルドおよび RPM パッケージプロセス全体が開始されます：

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

RPMs ディレクトリ内の結果の RPM には、collectd-curl も含まれるようになりました。これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
bash$ rpm --noparentdirs -i /bootflash/collectd-curl-5.5.0-1.ia32e.rpm
```




第 18 章

NX-SDK

この章は次のトピックで構成されています。

- [NX-SDK について](#) (179 ページ)
- [オンボックス \(ローカル\) アプリケーションについて](#) (181 ページ)
- [デフォルト Docker イメージ](#) (181 ページ)
- [NX-SDK に関する注意事項と制限事項](#) (181 ページ)
- [NX-SDK2.0 について](#) (182 ページ)
- [NX-SDK2.5 について](#) (183 ページ)
- [リモート アプリケーションについて](#) (183 ページ)
- [NX-SDK セキュリティ](#) (184 ページ)
- [NX SDK 2.0 のセキュリティ プロファイル](#) (185 ページ)

NX-SDK について

Cisco NX-OS SDK (NX-SDK) は、自動化およびカスタム アプリケーションの作成 (カスタムの生成など) のためのインフラストラクチャへのアクセスを合理化する C++ 抽象化およびプラグイン ライブラリ レイヤです。

- CLI
- Syslog
- イベント マネージャとエラー マネージャ
- アプリケーション間通信
- ハイ アベイラビリティ (HA)
- ルート マネージャ

NX-SDK を使用したアプリケーション開発には、C++、Python、または Go を使用できます。

要件

NX-SDK では次の要件があります。

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

ローカル（オン スイッチ）およびリモート（オフ スイッチ）アプリケーションのサポート

NX-SDK を使用して開発するアプリケーションは、Cisco Nexus スイッチではなく、NX-SDK が提供する Docker コンテナで作成または開発することになります（オフ スイッチ）。アプリケーションの作成後、アプリケーションを展開できる場所を柔軟に選択できます。

- ローカル（オンボックス）アプリケーションはスイッチ上で実行されます。詳細については、[オンボックス（ローカル）アプリケーションについて（181 ページ）](#)を参照してください。
- リモート（オフボックス）アプリケーションは、スイッチをオフにして実行されます。このオプションは NX-SDK 2.0 以降でサポートされており、アプリケーションをスイッチ以外の場所で実行するようにデプロイできます。詳細については、[リモートアプリケーションについて（183 ページ）](#)を参照してください。

関連情報

Cisco NX-SDK の詳細については、次にアクセスしてください。

- [Cisco DevNet NX-SDK](#)。バージョン .md リンク（）をクリックします。<https://github.com/CiscoDevNet/NX-SDK/blob/master/versions.md> を参照してください。
- [NX-SDK Readme](#)

必要に応じて、Cisco は NX-SDK の情報を GitHub に追加します。

Go バインディングに関する考慮事項

Go バインディングは、NX-SDK のリリースと、アプリがローカルで実行されているかリモートで実行されているかに応じて、さまざまなレベルでサポートされます。

- NX-SDK リモート アプリケーションのすべてのバージョンの Go バインディングは、EFT 前の品質です。
- ローカル NX-SDK 2.0 アプリケーションの Go バインディングは、EFT 前です。
- ローカル NX-SDK 1.7.5 以前のアプリケーションの Go バインディングがサポートされています。

詳細については、「[NX-SDK アプリケーションの GO バインディング](#)」を参照してください。

オンボックス（ローカル）アプリケーションについて

オンボックス（ローカル）アプリケーションでは、NX-SDK をインストールし、選択したサポート対象言語でアプリケーションをビルドし、スイッチにインストールできる .rpm ファイルとしてアプリをパッケージ化し、スイッチにアプリケーションをインストールして実行します。。 .rpm ファイルは手動で生成することも、自動生成することもできます。

アプリケーション開発は、NX-SDK が提供するコンテナで行われます。ローカルアプリケーションには、リモートアプリケーションとは異なるコンテナとツールを使用します。詳細については、[デフォルト Docker イメージ（181 ページ）](#) を参照してください。

ローカルアプリケーションのビルド、インストール、および実行については、[Cisco DevNet NX-SDK](#) を参照してください。

デフォルト Docker イメージ

NX-SDK には、ローカルまたはリモートで使用するための次の Docker イメージとツールがデフォルトで含まれています。

使用方法	目次
スイッチの場合	Cisco ENXOS SDK クロス コンパイル用の Wind River Linux (WRL) ツール チェーン 多言語バインディング ツールキット NX-SDK 1.75 以降、Go コンパイラ
オフ スイッチ（リモート）	事前にビルドされた libnxsdk.so を備えた NX-SDK 多言語バインディング ツールキット Go コンパイラ RapidJSON リモート API サポートのための gRPC

詳細については、<https://github.com/CiscoDevNet/NX-SDK#readme> を参照してください。

NX-SDK に関する注意事項と制限事項

NX-SDK には、アプリケーションをローカル（オンボックス）またはリモート（オフボックス）で実行するための使用上の注意事項と制限事項があります。

注意事項と制限事項については、[Cisco DevNet NX-SDK](#) の「役立つメモ」を参照してください。

- リモートアプリケーションの場合、SDK サーバーに接続するポートは 50002 ~ 50100 です。これらのポートが開いていて、他のサービスによって使用されていないことを確認してください。ポート 50051 は、内部アプリケーションによる使用がブロックされており、リモートアプリケーションでは使用できません。
- NX-SDK を使用したカスタム アプリケーションの開発には、次の制限が適用されます。
 - GNUC/C++ ツールチェーンと標準ライブラリを備えた現在の Linux 配布を使用してアプリケーションを開発する必要があります。
 - 提供されている Docker イメージでアプリケーションを開発することをお勧めします。Docker イメージと関連ツールを実行するには、最低 8 MB の空きメモリと 64 ビットホストが必要です。「[デフォルト Docker イメージ \(181 ページ\)](#)」を参照してください。
- Cisco は、Bash でアプリケーションをビルドしてテストすることを推奨しています。それらを本番環境に展開する場合は、アプリケーションを RPM としてビルドし、NX-OS VSH (Vshell) で実行します。
- Cisco Nexus スイッチは、Bash および VSH で最大 32 のアプリケーションをサポートします。
- NX-SDK 2.0 の場合、Cisco Nexus スイッチで実行されているリモート NX-SDK アプリケーションおよび NX-SDK サーバーには 1 対 1 の制限があります。
- 最大 10 個のリモート NX-SDK アプリケーションをスイッチで同時に実行できます。
- NX-SDK サーバーは、CLI で構成されたアプリケーションからのリモート要求のみを受け入れ、ランダムなアプリケーションからは受け入れません。
- エラーが発生すると、SDK サーバーは例外をスローします。
- 一部の API は、リモートアプリケーションでは実行されません。詳細については、API ドキュメントを参照してください。
- リモートクライアント ライブラリ コードはオープン ソースではないため、NX-SDK リモートアプリは NX-SDK リモート Docker コンテナで実行する必要があります。自分の OS でリモート NX-SDK を使用するには、使用している OS とコンパイラのバージョンを github でリクエストします。Cisco は、バージョンに基づいてリモート libnxsdk.so を生成できます。

NX-SDK2.0 について

NX-SDK バージョン 2.0 は、開発者が必要な場所でアプリケーションを実行できる実行環境の柔軟性を可能にします。このバージョンの NX-SDK では、アプリケーションは引き続きコンテナ内のスイッチをオフにして開発されますが、たとえばクラウドなどで、アプリケーションをスイッチ上またはスイッチ外のいずれかで実行できます。

NX-SDK 2.0 には次のような利点があります。

- スイッチをお客様の環境に簡単に統合できます。
- スイッチがデータセンター、パブリック クラウド、プライベート クラウドでシームレスに動作できるようにする拡張性。
- スイッチレベルのリソースでの変更がアプリケーションの変更または書き換えを必要としないように、スイッチ リソースから顧客アプリを切り離します。
- アプリケーションがリンクするための使いやすいAPIを備えた単一のライブラリ。これにより、スイッチの相互作用が簡素化され、アプリケーションをより簡単に記述およびデバッグできる高水準言語で記述できます。
- リモート サービスの実行は、オンボックス アプリケーションよりも安全です。

詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.mdを参照してください。

NX-SDK2.5 について

Cisco NX-OS Release 9.3(3) 以降では、ストリーミング Syslog 機能が追加されています。

詳細については、『CiscoDevNet』を参照してください。<https://github.com/CiscoDevNet/NX-SDK/blob/master/versions.md>

表 4: *syslog* イベント

機能	詳細
syslog イベント	• Cisco NX-OS syslog イベントに登録するカスタム アプリケーションの機能。 • 詳細については、<code>nx_trace.h</code> の <code>watchSyslog</code> および <code>postSyslogCb</code> API を参照してください。https://www.win-github.com/sathsrin/nxsdk/blob/master/include/nx_trace.h

リモート アプリケーションについて

リモート アプリケーションは、Cisco Nexus スイッチではない別のスイッチに置くことができます。リモートまたはオフボックスのアプリケーションは、NX-SDK レイヤを介して呼び出し、スイッチと対話して情報の読み取り（取得）または情報の書き込み（設定）をします。

ローカルとリモートの両方の NX-SDK アプリケーションは同じ API を使用するため、NX-SDK アプリケーションをオンボックスまたはオフボックスで柔軟に展開できます。

リモートで実行するには、アプリケーションが特定の要件を満たしている必要があります。詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

2.0 より前の NX-SDK アプリケーションの後方互換性

NX-SDK 2.0 には、NX-SDK v1.75 アプリケーションの開発方法に応じて、条件付きの後方互換性があります。

- 通常、NX-SDK は、NX-SDK 2.0 より前に作成したアプリのリモート実行をサポートしており、アプリを完全に書き直す必要はありません。代わりに、API 呼び出しを変更するために変更せずに、同じアプリを再利用できます。新しい NX-SDK 2.0 モデルで古いアプリをサポートするには、API 呼び出しで IP およびポートのパラメータを提供する必要があります。これらのパラメータは NX-SDK 1.75 以前では使用できませんが、アプリが SDK サーバにエクスポートできる環境変数として IP アドレスとポート情報を追加できます。
- ただし、NX-SDK 2.0 より前のアプリの下位互換性がサポートされていない場合があります。古いアプリの一部の API は、リモートでの実行をサポートしていないか、実行できない可能性があります。この場合、API は例外をスローできます。元のアプリケーションに対する例外処理の完全性と堅牢性によっては、アプリケーションが予期しない動作をする可能性があります、最悪の場合、クラッシュする可能性があります。

詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

NX-SDK セキュリティ

NX-OS 9.3 (1) 以降、NX-SDK 2.0 は次のセキュリティ機能をサポートしています。

- セッションセキュリティ。リモートアプリケーションは、トランスポートレイヤーサービス (TLS) を介してスイッチ上の NX SDK サーバに接続し、アプリケーションとスイッチの NX SDK サーバ間に暗号化されたセッションを提供できます。
- サーバ証明書のセキュリティ。Cisco NX-OS 9.3 (1) を使用した新しいスイッチ展開の場合、NX-SDK サーバは 1 日限りの一時証明書を生成して、カスタム証明書をインストールするのに十分な時間を提供します。
たとえば、以前の NX-SDK バージョンから NX-SDK 2.0 にアップグレードする場合など、NX-SDK サーバにカスタム証明書がすでにインストールされている場合、既存の証明書はアップグレード後も保持され、使用されます。
- API 書き込み呼び出し制御。NX-SDK 2.0 では、セキュリティ プロファイルが導入されています。これにより、アプリケーションが NX-SDK サーバをどの程度制御できるかを制御するための事前定義されたポリシーを選択できます。セキュリティ プロファイルに関する詳細情報を入手するには、[NX SDK 2.0 のセキュリティ プロファイル \(185 ページ\)](#) を参照します。

NX SDK 2.0 のセキュリティ プロファイル

以前のリリースでは、SDK バージョン 1.75 の API は、イベントのデータの読み取りと取得のみが許可されていました。Cisco NX-OS リリース 9.3(1) 以降、NX-SDK 2.0 は書き込みコールを含むさまざまなタイプの操作をサポートします。

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

Cisco の NX-SDK は、次のセキュリティ プロファイルを提供します。

プロファイル	説明	値
拒否	CLI の追加を除き、API 呼び出しがスイッチに書き込まないようにします。	これはデフォルト プロファイルです。
スロットル	スイッチを変更する API を許可しますが、指定された数の呼び出しまでのみ許可します。このセキュリティプロファイルは、スロットリングを適用して API 呼び出しの数を制御します。 アプリケーションは制限まで書き込むことができますが、制限を超えると書き込みが停止し、応答でエラーメッセージが送信されます。	スロットルは 50 回の API 呼び出しであり、スロットルは 5 秒後にリセットされます。
許可	スイッチを変更する API は制限なしで許可されます	

NX-SDK のセキュリティプロファイルの詳細については、[\[NX-SDK アプリケーションのセキュリティ プロファイル \(Security Profiles for NX-SDK Applications\)\]](#) を参照してください。

アプリケーションの構築、インストール、および実行の詳細については、[CiscoDevNet NX-SDK](#) にアクセスしてください。



第 19 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 (187 ページ)
- Docker の注意事項と制限事項 (188 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 (188 ページ)
- Docker デーモンの開始 (189 ページ)
- 自動的に起動するように Docker を構成する (189 ページ)
- Docker コンテナの開始: ホスト ネットワーク モデル (190 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 (191 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 (192 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 (193 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 (194 ページ)
- Docker ストレージバックエンドのサイズ変更 (195 ページ)
- Docker デーモンの停止 (197 ページ)
- Docker コンテナ セキュリティ (198 ページ)
- Kubernetes クラスタにノードを追加 (200 ページ)
- Docker のトラブルシューティング (202 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。

Docker の注意事項と制限事項

次に、スイッチ上の Cisco NX-OS で Docker を使用するためのガイドラインと制限事項を示します。

- Docker でサードパーティの DHCPD サーバーを実行している場合、SVI と一緒に使用すると、クライアントに到達するオフアーで問題が発生する可能性があります。可能な回避策は、ブロードキャスト応答を使用することです。
- Docker 機能は、少なくとも 8 GB のシステム RAM を備えた Cisco Nexus 9000Cisco Nexus 3000 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：

手順

ステップ 1 Bash を読み込み、スーパーユーザーになります。

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

ステップ 2 Docker デーモンを起動します。

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

ステップ 3 ステータスをチェックします。

```
root@switch# service docker status
dockerd (pid 3597) is running...
root@switch#
```

(注)

Docker デーモンを起動したら、ブートフラッシュの `dockerpart` ファイルを削除したり、改ざんしたりしないでください。これは、docker の機能にとって重要であるからです。

```
switch# dir bootflash:dockerpart
2000000000 Mar 14 12:50:14 2018 dockerpart
```

自動的に起動するように Docker を構成する

スイッチの起動時に常に自動的に起動するように Docker デーモンを構成できます。

手順

ステップ 1 Bash をロードしてスーパーユーザーになります。

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

ステップ 2 chkconfig ユーティリティを使用して、Docker サービスを永続化します。

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

ステップ 3 chkconfig ユーティリティを使用して、Docker サービスの設定を確認します。

```
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch#
```

ステップ 4 Docker が自動的に起動しないように構成を削除するには：

```
root@switch# chkconfig --del docker
root@switch# chkconfig --list | grep docker
root@switch#
```

Docker コンテナの開始: ホスト ネットワーク モデル

Docker コンテナがデータ ポートと管理を含むすべてのホスト ネットワーク インターフェイスにアクセスできるようにする場合は、`--network` ホスト オプションを使用して Docker コンテナを起動します。コンテナ内のユーザーは、`ip netns exec <net_namespace> <cmd>` を使用して、`/var/run/netns`（Cisco NX-OS で設定されたさまざまな VRF に対応）でさまざまなネットワーク名前空間を切り替えることができます。

手順

ステップ 1 Bash をロードしてスーパーユーザーになります。

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

ステップ 2 Docker コンテナを開始します。

以下は、スイッチで Alpine Docker コンテナを起動し、すべてのネットワーク インターフェイスを表示する例です。コンテナは、デフォルトで管理ネットワークの名前空間で起動されます。

```

root@switch# docker run --name=alpinerrun -v /var/run/netns:/var/run/netns:ro,rslave --rm
--network host --cap-add SYS_ADMIN -it alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
management
default
/ #
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
3: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default
link/gre 0.0.0.0 brd 0.0.0.0
...
/ #
/ # ip netns exec default ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/16 scope host lo
valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
link/ether 42:0d:9b:3c:d4:62 brd ff:ff:ff:ff:ff:ff
3: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
...

```

Docker コンテナの開始: ブリッジ型ネットワーク モデル

Docker コンテナに外部ネットワーク接続（通常は管理インターフェースを介して）のみを許可し、特定のデータ ポートまたは他のスイッチ インターフェースへの可視性を必ずしも気にしない場合は、デフォルトの Docker ブリッジ ネットワーク モデルで Docker コンテナを開始できます。これは、ネットワーク名前空間の分離も提供するため、前のセクションで説明したホスト ネットワーキング モデルよりも安全です。

手順

ステップ 1 Bash をロードしてスーパーユーザーになります。

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

ステップ 2 Docker コンテナを開始します。

以下は、スイッチで Alpine Docker コンテナを開始し、iproute2 パッケージをインストールする例です。

```
root@switch# docker run -it --rm alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
/ #
```

ステップ 3 ユーザー名前空間の分離を設定するかどうかを決定します。

ブリッジ ネットワーク モデルを使用するコンテナの場合、ユーザー名前空間の分離を設定して、セキュリティをさらに向上させることもできます。詳細については、「[ユーザー\[名前空間 \(namespace\)\]の分離による Docker コンテナの保護 \(198 ページ\)](#)」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 などのコンテナ内からサービスを公開できます。例：

```
root@switch# docker run -d -p 18877:22 --name sshd_container sshd_ubuntu
```

これにより、コンテナ内のポート 22 がスイッチのポート 18877 にマップされます。次の例に示すように、ポート 18877 を介してサービスに外部からアクセスできるようになりました。

```
root@ubuntu-vm# ssh root@ip_address -p 18887
```

Docker コンテナでのブートフラッシュおよび揮発性パーティションのマウント

Docker コンテナの run コマンドで `-v /bootflash:/bootflash` および `-v /volatile:/volatile` オプションを渡すことで、ブートフラッシュおよび揮発性パーティションを Docker コンテナに表示で

きます。これは、新しい NX-OS システム イメージをブートフラッシュにコピーするなど、コンテナ内のアプリケーションがホストと共有するファイルにアクセスする必要がある場合に役立ちます。



- (注) この **-v** コマンド オプションを使用すると、任意のディレクトリをコンテナにマウントでき、NX-OS システムの動作に影響を与える可能性のある情報漏えいやその他のアクセスが発生する可能性があります。これを、NX-OS CLI を使用してすでにアクセス可能な `/bootflash` や `/volatile` などのリソースに制限します。

手順

ステップ 1 Bash をロードしてスーパーユーザーになります。

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

ステップ 2 Docker コンテナの実行コマンドに `-v /bootflash:/bootflash` および `-v /volatile:/volatile` オプションを渡します。

```
root@switch# docker run -v /bootflash:/bootflash -v /volatile:/volatile -it --rm alpine
/# ls /
bin          etc          media        root         srv          usr
bootflash    home        mnt          run          sys          var
dev          lib         proc         sbin         tmp          volatile
/ #
```

拡張 ISSU スイッチオーバーでの Docker デーモンの永続性の有効化

Docker デーモンと実行中のコンテナの両方を拡張 ISSU スイッチオーバーで持続させることができます。これが可能なのは、バックエンドの Docker ストレージが存在するブートフラッシュが同じであり、アクティブ スーパーバイザとスタンバイ スーパーバイザの両方で共有されるためです。

Docker コンテナは、切り替え中に中断（再起動）されるため、継続的に実行されません。

手順

ステップ 1 Bash をロードしてスーパーユーザーになります。

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

ステップ2 スイッチオーバーを開始する前に、`chkconfig` ユーティリティを使用して Docker サービスを永続化します。

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

ステップ3 スイッチオーバー後にコンテナが自動的に再起動されるように、`--restart without-stopped` オプションを使用してコンテナを起動します。

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

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

Docker コンテナは、切り替え中に中断（再起動）されるため、継続的に実行されません。

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。

手順

ステップ1 Bash を読み込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。

手順

- ステップ1** Guest Shell を無効にします。

ゲスト シェルを無効にしないと、サイズ変更が妨げられる可能性があります。

```
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want
to disable the guest shell? (y/n) [n] y
switch# 2018 Mar 15 17:16:55 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating
virtual service 'guestshell+'
2018 Mar 15 17:16:57 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated
virtual service 'guestshell+'
```

- ステップ2** Bash をロードしてスーパーユーザーになります。

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

- ステップ3** 現在利用可能なストレージ容量に関する情報を取得します。

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 1.9G 7.6M 1.8G 1% /var/lib/docker
root@n9k-2#
```

- ステップ4** Docker デーモンを停止します。

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

- ステップ 5** Docker バックエンド ストレージ スペース（/bootflash/dockerpart）の現在のサイズに関する情報を取得します。

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2000000000 Mar 15 16:53 /bootflash/dockerpart
root@n9k-2#
```

- ステップ 6** Docker バックエンド ストレージ スペースのサイズを変更します。

たとえば、次のコマンドはサイズを 500 メガバイト増やします。

```
root@switch# truncate -s +500MB /bootflash/dockerpart
root@n9k-2#
```

- ステップ 7** Docker バックエンド ストレージ スペースのサイズに関する最新情報を取得して、サイズ変更プロセスが正常に完了したことを確認します。

たとえば、次の出力は、Docker バックエンド ストレージのサイズが 500 メガバイト増加したことを確認します。

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2500000000 Mar 15 16:54 /bootflash/dockerpart
root@n9k-2#
```

- ステップ 8** /bootflash/dockerpart のファイル システムのサイズを確認します。

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/122160 files (0.6% non-contiguous), 17794/488281 blocks
```

- ステップ 9** /bootflash/dockerpart のファイル システムのサイズを変更します。

```
root@switch# /sbin/resize2fs /bootflash/dockerpart
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /bootflash/dockerpart to 610351 (4k) blocks.
The filesystem on /bootflash/dockerpart is now 610351 blocks long.
```

- ステップ 10** /bootflash/dockerpart のファイル システムのサイズを再度チェックして、ファイル システムのサイズが正常に変更されたことを確認します。

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/154736 files (0.6% non-contiguous), 19838/610351 blocks
```

- ステップ 11** Daemon デーモンを再起動します。

```
root@switch# service docker start
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Starting dockerd with args '--debug=true':
```

ステップ 12 使用可能なストレージ領域の大きさを確認します。

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 2.3G 7.6M 2.3G 1% /var/lib/docker
```

ステップ 13 BASH シェルを終了します。

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

ステップ 14 Guest Shell を有効にします。

```
switch# guestshell enable

switch# 2018 Mar 15 17:12:53 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating
virtual service 'guestshell+'
switch# 2018 Mar 15 17:13:18 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully
activated virtual service 'guestshell+'
```

Docker デーモンの停止

Docker を今後使用しない場合は、このトピックの手順に従って Docker デーモンを停止します。

手順

ステップ 1 Bash をロードしてスーパーユーザーになります。

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

ステップ 2 Docker デーモンを停止します。

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

ステップ 3 Docker デーモンが停止していることを確認します。

```
root@switch# service docker status
dockerd is stopped
root@switch#
```

(注)

必要に応じて、この時点でブートフラッシュの `dockerpart` ファイルを削除することもできます。

```
switch# delete bootflash:dockerpart
Do you want to delete "/dockerpart" ? (yes/no/abort) y
switch#
```

Docker コンテナ セキュリティ

Docker コンテナのセキュリティに関する推奨事項は次のとおりです。

- 可能であれば、別のユーザー [名前空間 (namespace)] で実行します。
- 可能であれば、別のネットワーク [名前空間 (namespace)] で実行します。
- `cgroup` を使用して技術情報を制限します。既存の `cgroup` (`ext_ser`) が作成され、ホストされているアプリケーションを、プラットフォームチームがスイッチで実行される追加のソフトウェアに対して妥当と見なしたものに制限します。Docker では、これを使用して、コンテナごとの技術情報を制限できます。
- 不要な POSIX 機能を追加しないでください。

ユーザー[名前空間 (namespace)]の分離による Docker コンテナの保護

ブリッジ ネットワーク モデルを使用するコンテナの場合、ユーザー名前空間の分離を設定して、セキュリティをさらに向上させることもできます。詳細については、「<https://docs.docker.com/engine/security/usersns-remap/>」を参照してください。

手順

ステップ 1 システムに `dockremap` グループがすでに存在するかどうかを確認します。

`dockremap` ユーザーは、デフォルトでシステムにすでに設定されている必要があります。`dockremap` グループがまだ存在しない場合は、次の手順に従って作成します。

a) 次のコマンドを入力して `dockremap` グループを作成します。

```
root@switch# groupadd dockremap -r
```

b) `dockremap` ユーザーを作成します (まだ存在していない場合)。

```
root@switch# useradd dockremap -r -g dockremap
```

- c) dockremap グループと dockremap ユーザーが正常に作成されたことを確認します。

```
root@switch# id dockremap
uid=999(dockremap) gid=498(dockremap) groups=498(dockremap)
root@switch#
```

- ステップ2** 再マップされた必要な ID と範囲を /etc/subuid と /etc/subgid に追加します。

例：

```
root@switch# echo "dockremap:123000:65536" >> /etc/subuid
root@switch# echo "dockremap:123000:65536" >> /etc/subgid
```

- ステップ3** テキスト エディタを使用して、--userns-remap=default オプションを /etc/sysconfig/docker ファイルの other_args フィールドに追加します。

例：

```
other_args="--debug=true --userns-remap=default"
```

- ステップ4** [サービス ドッカー [re]start (service docker [re]start)] を使用して、Docker デーモンを再起動するか、まだ実行されていない場合は起動します。

例：

```
root@switch# service docker [re]start
```

ユーザー名前空間の分離によるコンテナの構成と使用の詳細については、<https://docs.docker.com/engine/security/userns-remap/>で Docker のドキュメントを参照してください。

cgroup パーティションの移動

サードパーティ サービスの cgroup パーティションは ext_ser で、CPU 使用率をコアあたり 25% に制限します。この ext_ser パーティションで Docker コンテナを実行することをお勧めします。

--cgroup-parent=/ext_ser/ オプションを指定せずに Docker コンテナを実行すると、最大 100% のホスト CPU アクセスが可能になり、Cisco NX-OS の通常の動作を妨げる可能性があります。

手順

-
- ステップ1** Bash をロードしてスーパーユーザーになります。

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

ステップ2 ext_ser パーティションで Docker コンテナを実行します。

例：

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm
--network host --cgroup-parent=/ext_ser/ --cap-add SYS_ADMIN -it alpine
/ #
```

Kubernetes クラスタにノードを追加

このトピックでは、Kubernetes クラスタにノードを追加する方法について説明します。この例では、次のようになります。

- Kubernetes (Ubuntu) プライマリの IP アドレスは 10.122.197.246 です。
- Docker コンテナとして実行されているスイッチソフトウェアの IP アドレスは 10.122.84.24 です。



(注) 次の例では、読みやすくなるように、長い行を \ 文字で分割しています。

手順

ステップ1 Kubernetes (Ubuntu) プライマリに対して次のコマンドを (on? for?) 実行します。

a) コマンド

```
root@switch# docker run -d --net=host gcr.io/google_containers/etcd:2.2.1
/usr/local/bin/etcd --listen-client-urls=http://0.0.0.0:4001
--advertise-client-urls=http://0.0.0.0:4001 --data-dir=/var/etcd/data
```

b) コマンド

```
root@switch# docker run -d --name=api --net=host --pid=host --privileged=true
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube apiserver
--insecure-bind-address=0.0.0.0 --allow-privileged=true
--service-cluster-ip-range=10.0.0.1/24 --etcd_servers=http://127.0.0.1:4001 --v=2
```

c) コマンド

```
root@switch# docker run -d --name=kubs --volume=/:/rootfs:ro --volume=/sys:/sys:ro
--volume=/dev:/dev --volume=/var/lib/docker:/var/lib/docker:rw
--volume=/var/lib/kubelet:/var/lib/kubelet:rw --volume=/var/run:/var/run:rw
--net=host --pid=host --privileged=true gcr.io/google_containers/hyperkube:v1.2.2
/hyperkube kubelet --allow-privileged=true --hostname-override="127.0.0.1"
--address="0.0.0.0" --api-servers=http://0.0.0.0:8080 --cluster_dns=10.0.0.10
--cluster_domain=cluster.local --config=/etc/kubernetes/manifests-multi
```


d) コマンド

```
root@switch# docker run -d --name=proxy --net=host --privileged
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube proxy
--master=http://0.0.0.0:8080 --v=2
```

e) コマンド

```
root@switch# export KUBERNETES_MASTER=http://10.122.197.246:8080
```

f) コマンド

```
root@switch# curl -o /usr/bin/kubect1
http://storage.googleapis.com/kubernetes-release/release/v1.2.2/bin/linux/amd64/kubect1
```

g) コマンド

```
root@switch# kubect1 -s $KUBERNETES_MASTER create -f kube-system.json
```

h) コマンド

```
root@switch# kubect1 -s $KUBERNETES_MASTER create -f skydns-rc.yaml
```

i) コマンド

```
root@switch# kubect1 -s $KUBERNETES_MASTER create -f skydns-svc.yaml
```

j) コマンド

```
root@switch# kubect1 -s $KUBERNETES_MASTER create -f dashboard.yaml
kubect1 -s $KUBERNETES_MASTER cluster-info
```

k) コマンド

```
root@switch# kubect1 -s $KUBERNETES_MASTER cluster-info
```

ステップ2 スイッチに対して次の手順 (on? for?) を実行します。

a) コマンド

```
root@switch# docker run -d --name=kubs --net=host --pid=host --privileged=true
--volume=/:/rootfs:ro --volume=/sys:/sys:ro --volume=/dev:/dev
--volume=/var/lib/docker/:/var/lib/docker:rw
--volume=/var/lib/kubelet/:/var/lib/kubelet:rw --volume=/var/run:/var/run:rw
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube kubelet --allow-privileged=true
--containerized --enable-server --cluster_dns=10.0.0.10 --cluster_domain=cluster.local
--config=/etc/kubernetes/manifests-multi --hostname-override="10.122.84.34"
--address=0.0.0.0 --api-servers=http://10.122.197.246:8080
```

b) コマンド

```
root@switch# docker run -d --name=proxy --net=host --privileged=true
gcr.io/google_containers/hyperkube:v1.2.2 /hyperkube proxy
```

```
--master=http://10.122.197.246:8080 --v=2
```

ステップ3 Kubernetes (Ubuntu) プライマリで次のコマンド (on? for?) を実行して、レプリケーションコントローラー オブジェクトに `nginx` アプリをデプロイします。

a) コマンド

```
lab@rnimbalk-ubuntu1:~$ kubectl get node
NAME                STATUS    AGE
10.122.84.34        Ready     16m
127.0.0.1           Ready     22m
```

b) コマンド

```
lab@rnimbalk-ubuntu1:~$ kubectl apply -f replication.yaml
replicationcontroller "nginx" created
```

c) コマンド

```
lab@rnimbalk-ubuntu1:~$ kubectl describe -f replication.yaml
Name:                nginx
Namespace:           default
Image(s):            nginx
Selector:            app=nginx
Labels:              app=nginx
Replicas:            3 current / 3 desired
Pods Status:        3 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen    LastSeen    Count   From              SubobjectPath
  Type         Reason
  -----
  17s          17s         1       {replication-controller }
    Normal      SuccessfulCreate Created pod: nginx-zqfpz
  17s          17s         1       {replication-controller }
    Normal      SuccessfulCreate Created pod: nginx-jij40
  17s          17s         1       {replication-controller }
    Normal      SuccessfulCreate Created pod: nginx-loa0g
```

Docker のトラブルシューティング

これらのトピックでは、Docker コンテナで発生する可能性のある問題について説明し、考えられる解決策を提供します。

Docker の起動が機能不全になる

[問題： (Problem:)] Docker の起動に失敗し、次のようなエラーメッセージが表示されます：

```
switch# run bash
bash-4.3$ service docker start
Free bootflash: 39099 MB, total bootflash: 51771 MB
```

```

Carving docker bootflash storage: 2000 MB
2000+0 records in
2000+0 records out
2000000000 bytes (2.0 GB) copied, 22.3039 s, 89.7 MB/s
losetup: /dev/loop18: failed to set up loop device: Permission denied
mke2fs 1.42.9 (28-Dec-2013)
mkfs.ext4: Device size reported to be zero. Invalid partition specified, or
partition table wasn't reread after running fdisk, due to
a modified partition being busy and in use. You may need to reboot
to re-read your partition table.

Failed to create docker volume

```

[考えられる原因：（Possible Cause:）] root ユーザーではなく、管理ユーザーとして Bash を実行している可能性があります。

[解決策：（Solution:）] root ユーザーではなく、管理ユーザーとして Bash を実行しているかどうかを確認します。

```

bash-4.3$ whoami
admin

```

Bash を終了し、ルート ユーザーとして Bash を実行します：

```

bash-4.3$ exit
switch# run bash sudo su -

```

ストレージが不足しているため、**Docker** が起動に失敗する

問題： ブートフラッシュ ストレージが不足しているため、Docker の起動に失敗し、次のようなエラー メッセージが表示されます。

```

root@switch# service docker start
Free bootflash: 790 MB, total bootflash: 3471 MB
Need at least 2000 MB free bootflash space for docker storage

```

考えられる原因： 十分な空きブートフラッシュ ストレージがない可能性があります。

解決策： スペースを解放するか、必要に応じて /etc/sysconfig/docker の変数 `_dockerstrg` 値を調整してから、Docker デーモンを再起動します。

```

root@switch# cat /etc/sysconfig/docker
# Replace the below with your own docker storage backend boundary value (in MB)
# if desired.
boundary_dockerstrg=5000

# Replace the below with your own docker storage backend values (in MB) if
# desired. The smaller value applies to platforms with less than
# $boundary_dockerstrg total bootflash space, the larger value for more than
# $boundary_dockerstrg of total bootflash space.
small_dockerstrg=300
large_dockerstrg=2000

```

Docker Hub からのイメージのプルの失敗 (509 証明書失効 エラー メッセージ)

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

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

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

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

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

必要に応じて、時計をリセットします：

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

例：

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

DockerHub 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イ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 III 部

NX-API

- [NX-API CLI](#) (211 ページ)
- [NX-API REST](#) (253 ページ)
- [NX-API 開発者サンドボックス](#) (259 ページ)



第 20 章

NX-API CLI

- [NX-API CLI について \(211 ページ\)](#)
- [NX-API CLI の使用 \(214 ページ\)](#)
- [NX-API 応答コードの表 \(240 ページ\)](#)
- [JSON および XML 構造化出力 \(243 ページ\)](#)
- [サンプル NX-API スクリプト \(252 ページ\)](#)

NX-API CLI について

NX-API CLI は、XML 出力をサポートする Cisco NX-OS CLI システムの拡張機能です。NX-API CLI は、特定のコマンドの JSON 出力フォーマットもサポートしています。

Cisco Nexus スイッチでは、コマンドラインインターフェイス (CLI) はスイッチ上でのみ実行されます。NX-API CLI は HTTP / HTTPS を使ってスイッチの外部で CLI を使用できるようにすることで、これらの CLI のユーザー補助を改善します。この拡張機能をスイッチの既存の Cisco NX-OS CLI システムに使用できます。NX-API CLI は **show** コマンド、構成と Linux Bash をサポートします。

NX-API CLI は JSON-RPC をサポートしています。

注意事項と制約事項

- NX-API CLI は、スイッチで Cisco NX-OS CLI を実行するために VSH を生成します。VSH のタイムアウトは 5 分です。Cisco NX-OS CLI の実行に 5 分以上かかると、コマンドは失敗し、「Back-end processing error.」というメッセージが表示されます。これは、NX-API コマンドのタイムアウトによって制御されます。これは、NX-API を介して要求されたコマンドを実行できる時間を制御します。この値は 300 秒に固定されており、変更できません。
-
- NX-API はワーカープロセスを生成し、複数のワーカープロセスがリクエストを均等に負担するようにします。
 - nginx のバックエンドワーカープロセスの数は 4 です。

- N3k および低メモリベースプラットフォームの **nginx** バックエンドワーカープロセスの数は 2 です。
- 各ワーカープロセスは、5 つの永続的な **VSH** セッションのプールを維持します。各 **VSH** セッションは、着信リクエストからのユーザー名とリモート IP の組み合わせで一意に識別されます。新しいリクエストが来るたびに、ワーカープロセスは一致するユーザー名とリモート IP エントリがすでに存在するかどうかを確認します。存在する場合は、対応する **VSH** セッションを使用します。存在しない場合は、プール内の可用性に基づいて新しい **VSH** セッションが作成され、新しいエントリがプールに追加されます。ワーカープロセスがすでに最大許容 **VSH** セッションを実行している場合、新しいリクエストは拒否され、適切なエラーメッセージが応答で返されます。
- ワーカープロセスごとの **VSH** セッション数はハードコードされた値であり、構成することはできません。任意の時点で存在できるセッションの合計数は 20 です。
- **NX-API** に関連付けられているトラストポイント、証明書、またはキーが削除されても、**NX-API** は **NX-API** 証明書、トラストポイント、または **NX-API** クライアント証明書の認証設定を保持します。そのため、**NX-API** 機能には影響が及びます。**NX-API** の現在のインスタンスは正常に動作しますが、**NX-API** コマンドの再構成によってインスタンスが破損する可能性があります。これを防ぐには、**no crypto ca trustpoint** コマンドを使用してトラストポイントまたは証明書を削除するときに、**NX-API** 設定も削除または更新することが重要です。

チャンクモード

- チャンクモードは、2 つの同時セッションのみをサポートします。チャンクオプションが選択されている場合、一度に 2 つの並列セッションでのみ指定できます。
- リリース 10.3(1)F リリースまで、チャンク モードでサポートされる応答の最大サイズは 200 MB です。
- 10.3(1)F リリース以降、チャンクモードは、スペースが揮発性領域（約 2.0GB）で使用可能である限り、応答サイズをサポートします。チャンクモード応答がサポートするサイズは、揮発性領域のスペースによって異なります。揮発性領域の 90% がいっぱいになると、その後最初に **show** 出力がファイルに収集されたとき、チャンクモードは失敗を返します。各応答でサポートされるチャンクサイズは 10 MB です。

転送

NX-APIは、転送のように HTTP または HTTPS を使用します。CLI は、HTTP/HTTPS POST 本文にエンコードされます。

Cisco NX-OS リリース 9.2(1) 以降、**NX-API** 機能は HTTPS ポート 443 でデフォルトで有効になっています。HTTP ポート 80 は無効化されています。

NX-API は、ホスト上でネイティブに、またはゲストシェル内で実行されるアプリケーションの UNIX ドメイン ソケットを介してサポートされます。

NX-API バックエンドは Nginx HTTP サーバを使用します。Nginx プロセスとそのすべての子プロセスは、CPU とメモリの使用量が制限されている Linux cgroup 保護下にあります。NX-API プロセスは、cgroup ext_ser_nginx の一部であり、2,147,483,648 バイトのメモリに制限されています。Nginx のメモリ使用量が cgroup の制限を超えると、Nginx プロセスは再起動されて、NX-API 構成（VRF、ポート、証明書構成）が復元されます。

メッセージ形式

NX-API は、XML 出力をサポートする Cisco Nexus 7000 シリーズ CLI システムの拡張機能です。NX-API は、特定のコマンドの JSON 出力フォーマットもサポートしています。

NX-API は、XML 出力をサポートする Cisco NX-OS CLI システムの拡張機能です。NX-API は、特定のコマンドの JSON 出力フォーマットもサポートしています。



- (注)
- NX-API XML 出力は、情報を使いやすいフォーマットで表示します。
 - NX-API XML は、Cisco NX-OS NETCONF 導入に直接マッピングされません。
 - NX-API XML 出力は、JSON に変換できます。

セキュリティ

- NX-API は HTTPS をサポートします。HTTPS を使用すると、デバイスへのすべての通信が暗号化されます。
- NX-API は、デフォルトでは非セキュア HTTP をサポートしていません。
- NX-API は、デフォルトでは弱い TLSv1 プロトコルをサポートしていません。

NX-API は、デバイスの認証システムに統合されています。ユーザーは、NX-API を介してデバイスにアクセスするための適切なアカウントを持っている必要があります。NX-API では HTTP basic 認証が使用されます。すべてのリクエストには、HTTP ヘッダーにユーザー名とパスワードが含まれている必要があります。



- (注) ユーザーのログイン資格情報を保護するには、HTTPS の使用を検討する必要があります。

[機能 (feature)] マネージャ CLI コマンドを使用して、NX-API を有効にすることができます。NX-API はデフォルトで無効になっています。

NX-API は、ユーザーが最初に認証に成功したときに、セッションベースの Cookie、**nxapi_auth** を提供します。セッション Cookie を使用すると、デバイスに送信される後続のすべての NX-API 要求にユーザー名とパスワードが含まれます。ユーザー名とパスワードは、完全な認証プロセスの再実行をバイパスするために、セッション Cookie で使用されます。セッション Cookie が後続の要求に含まれていない場合は、別のセッション Cookie が必要であり、認証プロセスに

よって提供されます。認証プロセスの不必要な使用を避けることで、デバイスのワークロードを軽減できます。



(注) **nxapi_auth** cookie は 600 秒（10 分）で期限切れになります。この値は固定されており、調整できません。



(注) NX-API は、スイッチ上の Programmable Authentication Module（PAM）を使用して認証を行います。**cookie** を使用して PAM の認証数を減らし、PAM の負荷を減らします。

NX-API CLI の使用

Cisco Nexus 9000 シリーズ スイッチのコマンド、コマンドタイプ、および出力タイプは、CLI を HTTP/HTTPS POST の本文にエンコードすることにより、NX-API を使用して入力されます。要求に対する応答は、XML または JSON 出力形式で返されます。



(注) NX-API 応答コードの詳細については、[NX-API 応答コードの表（240 ページ）](#) を参照してください。

NX-API CLI は、ローカル アクセスに対してはデフォルトで有効になっています。リモート HTTP アクセスに対してはデフォルトで無効になっています。

次の例は、NX-API CLI を構成して起動する方法を示しています。

- 管理インターフェイスを有効にします。

```
switch# conf t
Enter configuration commands, one per line.
End with CNTL/Z.
switch(config)# interface mgmt 0
switch(config-if)# ip address 10.126.67.53/25
switch(config-if)# vrf context managment
switch(config-vrf)# ip route 0.0.0.0/0 10.126.67.1
switch(config-vrf)# end
switch#
```

- NX-API **nxapi** 機能を有効にします。

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

次の例は、リクエストとそのレスポンスを XML 形式で示しています。

要求:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ins_api>
```

```

<version>0.1</version>
<type>cli_show</type>
<chunk>0</chunk>
<sid>session1</sid>
<input>show switchname</input>
<output_format>xml</output_format>
</ins_api>

```

応答 :

```

<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>

```

次の例は、JSON 形式の要求とその応答を示しています。

要求:

```

{
  "ins_api": {
    "version": "0.1",
    "type": "cli_show",
    "chunk": "0",
    "sid": "session1",
    "input": "show switchname",
    "output_format": "json"
  }
}

```

応答 :

```

{
  "ins_api": {
    "type": "cli_show",
    "version": "0.1",
    "sid": "eoc",
    "outputs": {
      "output": {
        "body": {
          "hostname": "switch"
        },
        "input": "show switchname",
        "msg": "Success",
        "code": "200"
      }
    }
  }
}

```



(注) ユーザーを削除しようとするとう失敗し、次のようなエラーメッセージが約 12 時間ごとに表示されるという既知の問題があります。

```
user delete failed for username:userdel: user username is currently logged in - securityd
```

この問題は、NX-API を介してスイッチにログインしているユーザーを削除しようとした場合に発生する可能性があります。この場合、次のコマンドを入力して、最初にユーザーのログアウトを試行します。

```
switch(config)# clear user username
```

その後、ユーザーの削除を再試行します。回避策を試みても問題が解決しない場合は、Cisco TAC へお問い合わせください。

NX-API で権限を root にエスカレーションする

NX-API では、管理者ユーザーの権限を root アクセスの権限にエスカレーションできます。

以下は、権限をエスカレーションするためのガイドラインです：

- 特権を root にエスカレーションできるのは管理者ユーザーのみです。
- root へのエスカレーションはパスワードで保護されています。

次の例は、管理者の権限を root にエスカレーションする方法と、エスカレーションを確認する方法を示しています。root になっても、**whoami** コマンドを実行すると **admin** として表示されることに注意してください。ただし、**admin** アカунトにはすべての root 権限があります。

最初の例：

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo su root ; whoami</input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>admin </body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```


2 番目の例 :

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo cat path_to_file </input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>[Contents of file]</body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

NX-API 管理コマンド

次の表にリストされている CLI コマンドを使用して、NX-API を有効にして管理できます。

表 5: NX-API 管理コマンド

NX-API 管理コマンド	説明
feature nxapi	NX-API を有効化します。
no feature nxapi	NX-API を無効化します。
nxapi {http https} port port	ポートを指定します。
no nxapi {http https}	HTTP / HTTPS を無効化します。
show nxapi	ポートと証明書情報を表示します。 (注) 「show nxapi」コマンドは、network-operator ロールの証明書/設定情報を表示しません。

NX-API 管理コマンド	説明
nxapi certificate {httpsrpt certfile httpskey keyfile} filename	次のアップロードを指定します： • httpsrpt 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： <pre>nxapi certificate httpsrpt certfile bootflash:cert.crt</pre> HTTPS キーの例： <pre>nxapi certificate httpskey keyfile bootflash:privkey.key</pre>
ファイル名パスフレーズ nxapi certificatehttpskey keyfile password	暗号化された秘密キーを使用して NX-API 証明書をインストールします。 (注) 暗号化された秘密キーを復号するためのパスフレーズは pass123! です。 例: <pre>nxapi certificate httpskey keyfile bootflash:encr-cc.pem password pass123!</pre>
nxapi certificate enable	証明書を有効化します。

NX-API 管理コマンド	説明
nxapi certificate sudi	この CLI は、Secure Unique Device Identifier (SUDI) を使用してデバイスを安全に認証する方法を提供します。 nginx の SUDI ベースの認証は、CISCO SUDI 準拠のコントローラによって使用されます。 SUDI は、X.509v3 証明書に含まれる IEEE 802.1AR 準拠のセキュアデバイス識別子で、Cisco デバイスの製品識別子とシリアル番号を維持します。ID は製造時に実装され、公的に識別可能なルート認証局につながれます。 (注) • NX-API が SUDI 証明書を使用する場合、ブラウザ、curl などのサードパーティ アプリケーションからはアクセスできません。 • 「nxapi certificate sudi」は、設定されている場合にカスタム証明書/キーを上書きし、カスタム証明書/キーを元に戻す方法はありません。 • 「nxapi certificate sudi」と「nxapi certificate trustpoint」と「nxapi certificate enable」は相互に排他的であり、一方を設定するともう一方の設定が削除されます。 • NX-API は、SUDI 証明書ベースのクライアント証明書認証をサポートしていません。クライアント証明書認証が必要な場合は、アイデンティティ証明書を使用する必要があります。 • NX-API 証明書 CLI は show run の出力に存在しないため、現在、CR/ロールバックの場合は、「nxapi certificate sudi」オプションで上書きされるとカスタム証明書に戻りません。
nxapi ssl-ciphers weak	Cisco NX-OS リリース 9.2(1) 以降、弱い暗号はデフォルトで無効になっています。このコマンドを実行すると、デフォルトの動作が変更され、NGINX の弱い暗号が有効になります。このコマンドの no 形式を使用すると、デフォルトに変更されます（デフォルトでは、弱い暗号は無効になります）。

NX-API 管理コマンド	説明
nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2}	Cisco NX-OS リリース 9.2(1)以降、TLS1.0はデフォルトで無効になっています。このコマンドを実行すると、文字列で指定された TLS バージョンが有効になります。必要に応じて、デフォルトで無効になっている TLS1.0 も含まれます。このコマンドの no 形式を使用すると、デフォルトに変更されます（デフォルトでは、TLS1.1 と TLS1.2 のみが有効になります）。
nxapi use-vrf vrf	デフォルト VRF、管理 VRF、または名前付き VRF を指定します。 (注) Cisco NX-OS リリース 7.0(3)I2(1) では、NGINX は 1 つの VRF でのみリッスンします。
ip netns exec management iptables	アクセス制限を実装し、管理 VRF で実行できます。 (注) 機能 bash-shell を有効にしてから、Bash シェルから コマンドを実行する必要があります。Bash シェルの詳細については、Bash の章を参照してください。 <i>Iptables</i> は、ポリシーチェーンを使用してトラフィックを許可またはブロックするコマンドライン ファイアウォールユーティリティであり、ほとんどの場合、Linux ディストリビューションにプリインストールされています。 (注) <i>iptables</i> が bash シェルで変更されたときに、リロード後も <i>iptables</i> を永続化する方法の詳細については、「」を参照してください。 リロード間で Iptable を永続化する (238 ページ)
nxapi idle-timeout <timeout>	リリース 9.3(5)以降では、アイドル状態の NX-API セッションが無効になるまでの時間を設定できます。指定できる時間は 1 ～ 1440 分です。デフォルトの時間は 10 分です。デフォルト値に戻すには、このコマンドの no 形式を使用します。 no nxapi idle-timeout <timeout>

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



(注) 証明書を有効にする前に、証明書とキーを設定する必要があります。

以下は、HTTPS キーの正常なアップロードの例です：

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

次に、暗号化された NXAPI サーバー証明書をインストールする方法の例を示します。

```
switch(config)# nxapi certificate httpskey certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!

switch(config)#nxapi certificate enable
switch(config)#
```

状況によっては、証明書が無効であることを示すエラーメッセージが表示されることがあります。

```
switch(config)# nxapi certificate httpskey certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
ERROR: Unable to load private key!
Check keyfile or provide pwd if key is encrypted, using 'nxapi certificate httpskey
keyfile <keyfile> password <passphrase>'.
```

この場合、filename passphrase を使用して暗号化キー ファイルのパスフレーズを指定する必要があります。 **nxapi certificatehttpskey keyfile password**

これが問題の原因である場合、証明書を正常にインストールできるはずです。

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!
switch(config)# nxapi certificate enable
switch(config)#
```

NX-API を使用したインタラクティブ コマンドの操作

対話型コマンドの確認プロンプトを無効にし、エラーコード 500 によるタイムアウトを回避するには、対話型コマンドの前に[端末の dont-ask (terminal dont-ask)]を追加します。 を使用。複数の対話型コマンドを区切るには、それぞれが。は単一のブランク文字で囲まれています。

エラー コード 500 でのタイムアウトを回避するために端末の dont-ask を使用する対話型コマンドの例をいくつか次に示します：

```
terminal dont-ask ; reload module 21
terminal dont-ask ; system mode maintenance
```

NX-API クライアント認証

NX-API クライアント基本認証

NX-API クライアントは、SSL/TLS を介した基本認証を介してスイッチ上の NGINX サーバで認証できます。この認証方式は、スイッチのデータベースに保存されるユーザー名とパスワードを構成することでサポートされます。NX-API クライアントは、接続要求を開始するときに、ユーザー名とパスワードを含む Hello メッセージを送信します。ユーザー名とパスワードがデータベースに存在する場合、スイッチはクッキーを含む Hello 応答を送信することで応答します。この最初のハンドシェイクが完了すると、通信セッションが開き、クライアントはスイッチへの API コールの送信を開始できます。詳細については、[セキュリティ \(213 ページ\)](#) を参照してください。

スイッチでのユーザー名とパスワードの設定方法など、基本認証の詳細については、[Cisco Nexus 9000 シリーズ NX-OS セキュリティ構成ガイド](#)を参照してください。

NX-API のクライアント証明書認証

NX-OS 9.3(3) 以降、NX-API はクライアントが開始する証明書ベースの認証をサポートしています。証明書ベースの認証では、TLS ハンドシェイク時に信頼できる関係者、つまり認証局 (CA) を使用してクライアントとサーバーの両方を相互に認証することで、セキュリティを強化します。証明書ベースの認証では、NX-OS スイッチにアクセスするためのマシン認証だけでなく、人間による認証も可能です。

クライアント証明書認証は、有効な CA (認証局) を介して割り当てられ、NX-API クライアントに保存されている X509 SSL 証明書を使用してサポートされます。証明書は、各 NX-API ユーザー名に割り当てられます。

NX-API クライアントが Hello メッセージを使用して接続要求を開始すると、サーバーの Hello 応答に有効な CA のリストが含まれます。クライアントの応答には、NX-API クライアントが使用している特定のユーザー名の証明書など、追加の情報要素が含まれます。

NX-API クライアントは基本認証、証明書認証のいずれかを使用するように構成することができます。または証明書を優先するものの、証明書認証方式が使用できない場合は基本認証にフォールバックするように設定することもできます。

注意事項と制約事項

証明書認証には次の注意事項と制約事項があります。

- NX-API クライアントには、ユーザー名とパスワードを設定する必要があります。
- NX-API クライアントとスイッチは、デフォルトでウェルノウンポートで HTTP を介して通信します。柔軟性を高めるために、HTTP は既知のポートでもサポートされます。ただし、追加のポートを設定できます。
- クライアント証明書認証の Python スクリプティングがサポートされています。クライアント証明書がパスフレーズで暗号化されている場合、python はパスフレーズの入力を正常

に要求します。ただし、Python 要求ライブラリの現在の制限により、パスフレーズをスクリプトに渡すことはできません。

- NX-API クライアントとスイッチは、同じトラストポイントを使用する必要があります。
 - サポートされるトラストポイントの最大数は、スイッチごとに 26 です。
 - 信頼できる CA のリストは、すべての NX-API クライアントとスイッチで同じである必要があります。信頼できる CA の個別のリストはサポートされていません。
 - 証明書認証は、NX-API サンドボックスではサポートされていません。
 - 次の条件によって、NX-API サンドボックスがスイッチにロードされるかどうかが決まります。
 - NX-API サンドボックスは、または が設定されている場合にのみロードされます。
nxapi client certificate authentication optionalno nxapi client certificate authentication
 - NX-API サンドボックスは、接続の確立時に有効なクライアント証明書がブラウザに提示されない限り、および認証モードをロードしません。 **stricttwo-step**
 - スイッチには NGINX サーバーが組み込まれています。複数のトラストポイントが設定されているが、証明書失効リスト (CRL) が 1 つのトラストポイントのみにインストールされている場合、NGINX の制限により NX-API クライアント証明書認証は失敗します。この制限を回避するには、すべてのトラストポイントに CRL を設定します。
 - 証明書は期限切れになったり、期限切れになったりする可能性があり、CA (トラストポイント) によって設定された CRL の有効性に影響を与える可能性があります。スイッチが有効な CRL を使用するようにするには、設定されているすべてのトラストポイントに必ず CRL をインストールしてください。トラストポイントによって証明書が失効しなかった場合は、空の CRL を生成、インストール、および更新する必要があります。たとえば、週に 1 回などです。
- 暗号化 CLI を使用して CRL を更新した後、を発行して、新しく更新された CRL を再適用します。 **nxapi client cert authentication**
- NX-API クライアント証明書認証が有効になっているときに ASCII リロードを使用する場合は、リロードの完了後にを発行する必要があります。 **nxapi client certificate authentication**
 - 証明書パスは信頼済み CA 証明書で終了している必要があります。
 - TLS 用に提示されるサーバー証明書には、extendedKeyUsage フィールドにサーバー認証目的 (OID 1.3.6.1.5.5.7.3.1 の id-kp 1) が必要です。
 - TLS 用に提示されるクライアント証明書には、extendedKeyUsage フィールドにサーバー認証目的 (OID 1.3.6.1.5.5.7.3.2 の id-kp 1) が必要です。
 - この機能は、CRL (証明書失効リスト) をサポートします。オンライン証明書ステータスプロトコル (OCSP) はサポートされていません。
 - 『NX-OS Security Guide』の追加のガイドラインと制限事項に従ってください。

- 証明書と基本認証の両方を使用します。そうすることで、証明書が何らかの理由で侵害された場合でも、正しいユーザーとパスワードが必要になります。
- サーバーの公開キーには接続を試みるすべてのユーザーがアクセスできるため、秘密キーは秘密にしておきます。
- CRL は中央 CA からダウンロードし、最新の状態に保つ必要があります。古い CRL はセキュリティリスクにつながる可能性があります。
- トラストポイントを最新の状態に保つ。トラストポイントまたは設定変更が証明書認証機能に加えられた場合は、更新された情報をリロードするために、この機能を明示的にディセーブルにしてから再度イネーブルにします。
- `nxapi certificate https-cert certfile bootflash:` これは Day-1 の制限です。
- [NX-API Management Commands] 表 1 で、コマンド `nxapi certificate {https-cert certfile | https-key keyfile}` ファイル名の場合、サポートされる certfile の最大サイズは 8K 未満です。

NX-API のクライアント証明書認証の前提条件

証明書認証を設定する前に、スイッチで次の操作を実行済みであることを確認してください。

1. クライアントでユーザー名とパスワードを構成します。詳細については、「ユーザーアカウントおよび RBAC の構成」を参照してください。
2. CA（トラストポイント）と CRL（存在する場合）を構成します。

トラストポイントによって失効した証明書がない場合は、トラストポイントごとに空白の CRL を作成します。

詳細については、『Cisco Nexus 9000 シリーズ NX-OS セキュリティ設定ガイド』を参照してください。

NX-API クライアント証明書認証の構成

コマンドを使用して、NX-API 証明書認証を設定できます。 `nxapi client certificate authentication` コマンドは、認証方法を制御する制限オプションをサポートします。

この機能は、`no nxapi client certificate authentication` を使用して無効にすることができます。

NX-API クライアントの証明書認証を設定するには、次の手順を実行します。

手順

	コマンドまたはアクション	目的
ステップ 1	この機能の前提条件が満たされていることを確認します。	「 NX-API のクライアント証明書認証の前提条件（224 ページ） 」を参照してください。

	コマンドまたはアクション	目的
ステップ 2	config terminal 例 : <pre>switch-1# config terminal Enter configuration commands, one per line. End with CNTL/Z. switch-1(config)#</pre>	コンフィギュレーション モードに入ります。
ステップ 3	nxapi client certificate authentication [{optional strict two-step}] 例 : <pre>switch-1# nxapi client certificate authentication strict switch-1(config)#</pre>	次のいずれかのモードで証明書認証をイネーブルにします。 • optional はクライアント証明書を要求します。 • クライアントが証明書を提供すると、クライアントとサーバーの間で相互検証が行われます。 • クライアントが無効な証明書を提供した場合、認証は失敗し、基本認証へのフォールバックは行われません。 • クライアントが証明書を提供しない場合、認証は基本認証（ユーザー名とパスワード）にフォールバックします。 • strict クライアント証明書の検証を有効にし、認証のために有効なクライアント証明書を提示する必要があります。 • two-step は、基本認証方式と証明書認証方式の両方が必要な2段階検証を有効にします。 (注) スイッチにトラストポイントが設定されていない場合は、この機能をイネーブルにできず、スイッチの画面にエラーメッセージが表示されます。 <pre>No trustpoints configured! Please configure trustpoint using 'crypto ca trustpoint <trustpoint-label>' and associated commands, and then enable this feature.</pre>

証明書認証用の Python スクリプトの例

次の例は、認証用のクライアント証明書を使用した Python スクリプトを示しています。

```
import requests
import json

"""
Modify these please
"""
switchuser='USERID'
switchpassword='PASSWORD'
mgmtip='NXOS MANAGEMENT IP/DOMAIN NAME'

client_cert_file='PATH_TO_CLIENT_CERTIFICATE'
client_key_file='PATH_TO_CLIENT_KEY_FILE'
ca_cert='PATH_TO_CA_CERT_THAT_SIGNED_NXAPI_SERVER_CERT'

url='https://' + mgmtip + '/ins'
myheaders={'content-type':'application/json-rpc'}
payload=[
    {
        "jsonrpc": "2.0",
        "method": "cli",
        "params": {
            "cmd": "show clock",
            "version": 1
        },
        "id": 1
    }
]
response = requests.post(url,data=json.dumps(payload),
headers=myheaders,auth=(switchuser,switchpassword),cert=(client_cert_file_path,client_key_file),verify=ca_cert).json()
```

必要に応じて、スクリプトを変更できます。

- クライアント証明書認証モードによっては、スイッチパスワードをヌル値に設定することで (switchpassword=) 、スイッチパスワードを省略できます。
- **optional** および **strict** モードの場合、switchpassword= は空白のままにできます。この場合、NX-API はユーザー名とクライアント証明書のみに基づいてクライアントを認証します。
- **two-step** モードの場合、パスワードが必要なため、switchpassword= の値を指定する必要があります。
- POST コマンドで verify=False を設定することで、NX-API サーバの証明書が有効であることの確認をバイパスできます。

cURL 証明書要求の例

次に、NX-API クライアント認証用の正しく構造化された cURL 証明書要求の例を示します。

```
/usr/bin/curl --user admin: --tlsv1.2 --cacert ./ca.pem --cert ./user.crt:pass123! --key ./user.key -v -X POST -H "Accept: application/json" -H "Content-type: application/json" --data '{"ins_api":{"version": "1.0", "type": "cli_show", "chunk": "0", "sid": "1", "input": "show clock","output_format": "json"}}' https://<device-management-ip>:443/ins
```

構文要素

次の表は、この要求で使用されるパラメータを示しています。

パラメータ	説明
--user	ユーザがログインするユーザ名を取得します。 これは、 user.crt の共通名と同じである必要があります。 ユーザーのパスワードを指定するには、コロンの後にパスワードを指定します。例： --user username:password
--cacert	NX-API サーバー証明書に署名した CA へのパスを使用します。 サーバー証明書を検証する必要がない場合は、 (insecure) オプションを使用して cURL を指定します。例： /usr/bin/curl -k -k
--cert	クライアント証明書へのパスを使用します。 クライアント証明書が暗号化されている場合は、コロンの後にパスワードを指定します。 例： --cert user.crt:pass123!
--key	クライアント証明書の秘密キーへのパスを使用します。

証明書認証の検証

正しく構成されている場合、証明書認証が行われ、NX-API クライアントはスイッチにアクセスできます。

NX-API クライアントがスイッチにアクセスできない場合は、次のガイドラインに従ってトラブルシューティングを行うことができます。

手順

	コマンドまたはアクション	目的
ステップ 1	ユーザーまたはクッキーのエラーを確認します。	次のいずれかのエラーが発生していた場合： - 認証ヘッダーにユーザー名が指定されておらず、有効なクッキーが指定されていない - 認証ヘッダーで指定されたユーザーが正しくない

	コマンドまたはアクション	目的
		- 無効なクッキーが提供された - 認証ヘッダーのユーザー名とクライアント証明書の CN フィールドのユーザー名が一致しない 使用されている NX-API 方式に応じて、特定のエラーが表示されます： - JSON/XML の場合、401 認証エラー：ユーザーが見つからないエラーが発生します。次に例を示します。 <pre> {{{ "code": "400", "msg": "Authentication failure - user not found." }}}</pre> - JSON RPC 2.0 の場合、-32004 無効なユーザー名またはパスワードエラーが発生します。次に例を示します。 <pre> {{ "code": -32004, "message": "Invalid username or password" }}</pre>
ステップ 2	証明書に誤りがないか確認してください。	次の内容を示す HTTPs 400 エラーを探します。 - 無効または失効したクライアント証明書が提供されていないか確認します。 - スイッチに設定されている CRL の有効期限が切れていないか確認します。 次に例を示します。 <pre> <html> <head><title>400 The SSL certificate error</title></head> <body bgcolor="white"> <center><h1>400 Bad Request</h1></center> <center>The SSL certificate error</center> <hr><center>nginx/1.7.10</center> </body> </html></pre>

	コマンドまたはアクション	目的
ステップ 3	エラーが発生した場合は、 no nxapi client certificate authentication 、それから nxapi client certificate authentication を発行して、トラストポイント、CA、CRL、または NX-OS 証明書機能に対する変更をリロードするように機能をフラップします。	証明書認証を無効にしてから、再度有効にします。

NX-API リクエスト要素

NX-API リクエスト要素は、XML フォーマットまたは JSON フォーマットでデバイスに送信されます。リクエストの HTTP ヘッダーは、リクエストのコンテンツ タイプを識別する必要があります。

次の表にリストされている NX-API 要素を使用して、CLI コマンドを指定します。



- (注) ユーザには、「configure terminal」コマンドを実行する権限が必要です。JSON-RPC が入力要求形式の場合、「configure terminal」コマンドは、ペイロード内のコマンドが実行される前に常に実行されます。

表 6: XML または JSON 形式の NX-API 要求要素

NX-API リクエスト要素	説明
version	NX-API バージョンを指定します。

NX-API リクエスト要素	説明
type	実行するコマンドのタイプを指定します。 次のタイプのコマンドがサポートされています。 • cli_show 構造化された出力が必要な CLI show コマンド。コマンドが XML 出力をサポートしていない場合は、エラーメッセージが返されます。 • cli_show_array 構造化された出力が必要な CLI show コマンド。show コマンド専用。cli_show に似ていますが、cli_show_array を使用すると、データは角括弧 [] で囲まれた 1 つの要素のリストまたは配列として返されます。 • cli_show_ascii ASCII 出力が必要な CLI show コマンド。これは、ASCII 出力を解析する既存のスクリプトと一致します。ユーザーは、最小限の変更で既存のスクリプトを使用できます。 • cli_conf CLI 構成コマンド • bash Bash コマンド。ほとんどの非対話型 Bash コマンドは、NX-API でサポートされています。 (注) • 各コマンドは、現在のユーザーの権限でのみ実行可能です。 • メッセージタイプが ASCII の場合、出力でパイプ操作がサポートされます。出力が XML 形式の場合、パイプ操作はサポートされていません。 • 最大 10 の連続する show コマンドがサポートされています。show コマンドの数が 10 を超える場合、11 番目以降のコマンドは無視されます。 • 対話型コマンドはサポートされていません。

NX-API リクエスト要素	説明				
チャンク	一部の show コマンドは、大量の出力を返す場合があります。コマンド全体が完了する前に NX-API クライアントが出力の処理を開始するために、NX-API は show コマンドの出力チャンクをサポートしています。 次の設定を有効または無効にできます。 (注) <table><tr><td>0</td><td>チャンク出力しません。</td></tr><tr><td>1</td><td>チャンク出力。</td></tr></table> (注) • チャンクをサポートするのは show コマンドだけです。一連の show コマンドが入力されると、最初のコマンドだけがチャンクされて返されます。 • 出力メッセージ形式のオプションは、XML または JSON です。 • XML 出力メッセージ形式の場合<または>などの特殊文字は、有効な XML メッセージを形成するために変換されます (<は<に変換されます>は>に変換されます)。 XML SAX を使用して、チャンクされた出力を解析できます。 • 出力メッセージ形式が JSON の場合、チャンクが連結されて有効な JSON オブジェクトが作成されます。 (注) チャンクが有効になっている場合、現在サポートされている最大メッセージサイズは、チャンク出力の 200MB です。	0	チャンク出力しません。	1	チャンク出力。
0	チャンク出力しません。				
1	チャンク出力。				

NX-API リクエスト要素	説明
ロールバック	コンフィギュレーション CLI に対してのみ有効であり、<code>show</code> コマンドに対しては有効ではありません。コンフィギュレーションロールバック オプションを指定します。次のいずれかのオプションを指定します。 • <code>Stop-on-error</code> : 最初に失敗した CLI で停止します。 • <code>Continue-on-error</code> : 他の CLI を無視して続行します。 • <code>Rollback-on-error</code> : システム設定を以前の状態にロールバックします。 (注) 入力要求形式が XML または JSON の場合、ロールバック要素は <code>cli_conf</code> モードで使用できます。
<i>sid</i>	セッションID要素は、応答メッセージがチャンクされている場合にのみ有効です。メッセージの次のチャンクを取得するには、前の応答メッセージの <i>sid</i> と一致する <i>sid</i> を指定する必要があります。 NX-OS リリース 9.3(1) では、<i>sid</i> オプション <code>clear</code> が導入されています。<i>sid</i> を <code>clear</code> に設定して新しいチャンク リクエストが開始されると、現在のチャンク リクエストはすべて破棄または破棄されます。 応答コード 429 を受け取った場合：同時チャンク リクエストの最大数は 2 です。<i>sid clear</i> を使用して、現在のチャンク リクエストを破棄します。<i>sid clear</i> を使用した後、後続の応答コードは、残りのリクエストに対して通常どおり動作します。

NX-API リクエスト要素	説明						
<i>input</i>	入力は 1 つのコマンドまたは複数のコマンドです。ただし、異なるメッセージ タイプに属するコマンドを混在させてはなりません。たとえば、show コマンドは cli_show メッセージタイプであり、cli_conf モードではサポートされません。 (注) bash を除き、複数のコマンドは「;」で区切ります。(; は、単一のブランク文字で囲む必要があります。) エラーコード 500 でタイムアウトしないように、コマンドの前に端末の dont-ask を付加します。次に例を示します。 <pre>terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport</pre> bash の場合、複数のコマンドは「;」で区切ります。(; は単一のブランク文字で囲まれていません。) 以下は、複数のコマンドの例です。 (注) <table><tr><td>cli_show</td><td>show version ; show interface brief ; show vlan</td></tr><tr><td>cli_conf</td><td>interface Eth4/1 ; no shut ; switchport</td></tr><tr><td>bash</td><td>cd /bootflash;mkdir new_dir</td></tr></table>	cli_show	show version ; show interface brief ; show vlan	cli_conf	interface Eth4/1 ; no shut ; switchport	bash	cd /bootflash;mkdir new_dir
cli_show	show version ; show interface brief ; show vlan						
cli_conf	interface Eth4/1 ; no shut ; switchport						
bash	cd /bootflash;mkdir new_dir						

NX-API リクエスト要素	説明				
<i>output_format</i>	使用可能な出力メッセージ形式は次のとおりです。 (注) <table border="1"> <tr> <td>xml</td><td>XML 形式を指定します。</td></tr> <tr> <td>json</td><td>JSON 形式で出力を指定します。</td></tr> </table> (注) Cisco NX-OS CLI は XML 出力をサポートしています。つまり、JSON 出力は XML から変換されます。変換はスイッチで処理されます。 計算のオーバーヘッドを管理するために、JSON 出力は出力の量によって決定されます。出力が 1 MB を超える場合、出力は XML 形式で返されます。出力がチャンクされている場合、XML 出力のみがサポートされます。 HTTP/HTTPS ヘッダーの content-type ヘッダーは、応答形式 (XML または JSON) のタイプを示します。	xml	XML 形式を指定します。	json	JSON 形式で出力を指定します。
xml	XML 形式を指定します。				
json	JSON 形式で出力を指定します。				

JSON-RPC が入力リクエスト形式である場合、次の表にリストされている NX-API 要素を使用して、CLI コマンドを指定します。

表 7: JSON-RPC 形式の NX-API 要求要素

NX-API リクエスト要素	説明
<i>jsonrpc</i>	JSON-RPC プロトコルのバージョンを指定する文字列。 バージョンは 2.0 である必要があります。
<i>method</i>	呼び出されるメソッドの名前を含む文字列。 NX-API は、次のいずれかをサポートします。 • cli : show または構成コマンド • cli_ascii : show または構成コマンド。フォーマットせずに出力 • cli_array : show コマンド専用。cli に似ていますが、cli_array はデータを角括弧 ([]) で囲まれた 1 つの要素のリスト、つまり配列として返します。

NX-API リクエスト要素	説明
<i>params</i>	メソッドの呼び出し中に使用されるパラメータ値を保持する構造化された値。 以下が含まれている必要があります。 • cmd : CLI コマンド • version : NX-API リクエストのバージョン識別子
ロールバック	コンフィギュレーション CLI に対してのみ有効であり、show コマンドに対しては有効ではありません。構成ロールバック オプション次のいずれかのオプションを指定できます。 • Stop-on-error : 最初に失敗した CLI で停止します。 • Continue-on-error : 失敗した CLI を無視し、他の CLI を続行します。 • Rollback-on-error : システム設定を以前の状態にロールバックします。
検証	構成検証設定この要素を使用すると、スイッチに適用する前にコマンドを検証できます。これにより、設定を適用する前に、設定の整合性（必要なハードウェア リソースの可用性など）を確認できます。[検証タイプ (Validation Type)] ドロップダウン リストから検証タイプを選択します。 • Validate-Only : 設定を検証しますが、設定は適用しません。 • Validate-and-Set : 設定を検証し、検証が成功した場合はスイッチに設定を適用します。
ロック	構成の排他ロックを指定できます。これにより、このロックが保持されている場合、他の管理エージェントまたはプログラミングエージェントは構成を変更できません。
<i>id</i>	クライアントによって確立されるオプションの識別子。指定されている場合は、文字列、数値、または null 値を含む必要があります。値は null にならないはずです。数値には小数部を含めません。ユーザーが id パラメータを指定しなかった場合、サーバーはリクエストが単なる通知であるとみなし、応答はしません。パラメータは <i>id</i>: 1 などのように指定します。

NX-API 応答要素

CLI コマンドに応答する NX-API 要素を次の表に示します。

表 8: NX-API 応答要素

NX-API 応答要素	説明
version	NX-API バージョン。
type	実行するコマンドのタイプ。
sid	応答のセッション識別子。この要素は、応答メッセージがチャ ンクされている場合にのみ有効です。
outputs	すべてのコマンド出力を囲むタグ。 複数のコマンドが cli_show または cli_show_ascii にある場合、各 コマンド出力は単一の出力タグで囲まれます。 メッセージタイプが cli_conf または bash の場合、cli_conf およ び bash コマンドにはコンテキストが必要なため、すべてのコマ ンドに単一の出力タグがあります。
出力	単一のコマンド出力の出力を囲むタグ。 cli_conf と bash メッセージタイプの場合、この要素にはすべて のコマンドの出力が含まれます。
input	リクエストで指定された 1 つのコマンドを囲むタグ。この要素 は、要求入力要素を適切な応答出力要素に関連付けるのに役立 ちます。
本文	コマンド応答の本文。
コード	コマンドの実行から返された原因コード。 NX-API は、ハイパーテキスト転送プロトコル (HTTP) ステータ ス コード レジストリ (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml) で説明されている標準規格の HTTP 原因コードを使用します。
msg	返された原因コードに関連付けられたエラー メッセージ。

NX-API へのアクセスの制限

デバイスへの HTTP および HTTPS アクセスを制限するには、ACL と iptable の 2 つの方法があ
ります。使用方法は、nxapi use-vrf<vrf-name> CLI コマンドを使用して、NX-API 通信の
VRF を構成していたかどうかに応じて決まります。

特定の VRF を使用するように NXAPI を構成していない場合にのみ、ACL を使用してデバイスへの HTTP または HTTPS アクセスを制限します。ACL の構成の詳細については、使用しているスイッチファミリの *Cisco Nexus* シリーズ NX-OS セキュリティ構成ガイドを参照してください。

ただし、NX-API 通信用に VRF を設定した場合、ACL は HTTP または HTTPS アクセスを制限しません。代わりに、iptables のルールを作成します。ルールの作成の詳細については、[iptables の更新 \(237 ページ\)](#) を参照してください。

iptables の更新

iptables を使用すると、VRF が NX-API 通信用に設定されている場合に、デバイスへの HTTP または HTTPS アクセスを制限できます。このセクションでは、既存の iptable への HTTP および HTTPS アクセスをブロックするルールを追加、確認、および削除する方法を示します。

手順

ステップ 1 HTTP アクセスをブロックするルールを作成するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp --dport 80 -j DROP
```

(注)

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

ステップ 2 HTTPS アクセスをブロックするルールを作成するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp --dport 443 -j DROP
```

ステップ 3 適用されたルールを確認するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -L
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            tcp dpt:http
DROP       tcp  --  anywhere              anywhere               tcp dpt:https
DROP       tcp  --  anywhere              anywhere               tcp dpt:https
```

```
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
```

```
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

ステップ 4 ポート 80 への 10.155.0.0/24 サブネットを持つすべてのトラフィックをブロックするルールを作成して確認するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -s 10.155.0.0/24 -p tcp --dport 80 -j DROP
```

```
bash-4.3# ip netns exec management iptables -L
```

```
Chain INPUT (policy ACCEPT)
```

```

target      prot opt source      destination
DROP        tcp  --  10.155.0.0/24 anywhere      tcp dpt:http

Chain FORWARD (policy ACCEPT)
target      prot opt source      destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source      destination

```

ステップ 5 以前に適用したルールを削除して確認するには、次の手順を実行します。

この例では、最初のルールを INPUT から削除します。

```

bash-4.3# ip netns exec management iptables -D INPUT 1
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target      prot opt source      destination

Chain FORWARD (policy ACCEPT)
target      prot opt source      destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source      destination

```

次のタスク

iptables のルールを bash シェルで変更した場合、リロード後は保持されません。ルールを永続的にするには、を参照してください。[リロード間で Iptable を永続化する \(238 ページ\)](#)

リロード間で Iptable を永続化する

iptables のルールを bash シェルで変更した場合、リロード後は保持されません。このセクションでは、リロード後も変更された iptable を永続化する方法について説明します。

始める前に

iptables を変更したとします。

手順

ステップ 1 iptables_init.log という名前のファイルを /etc ディレクトリに作成します：

```
bash-4.3# touch /etc/iptables_init.log; chmod 777 /etc/iptables_init.log
```

ステップ 2 iptable の変更を保存する /etc/sys/iptables ファイルを作成します：

```
bash-4.3# ip netns exec management iptables-save > /etc/sysconfig/iptables
```

ステップ 3 次の一連のコマンドを使用して、/etc/init.d ディレクトリに「iptables_init」という起動スクリプトを作成します：

```
#!/bin/sh

### BEGIN INIT INFO

# Provides:          iptables_init

# Required-Start:

# Required-Stop:

# Default-Start:     2 3 4 5

# Default-Stop:

# Short-Description: init for iptables

# Description:       sets config for iptables

#                    during boot time

### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
    start)
        start_script
        ;;
    stop)
        ;;
    restart)
        sleep 1
        $0 start
        ;;
    *)
        echo "Usage: $0 {start|stop|status|restart}"
        exit 1
esac
exit 0
```

ステップ 4 起動スクリプトに適切な権限を設定します：

```
bash-4.3# chmod 777 /etc/init.d/iptables_init
```

ステップ 5 chkconfig ユーティリティを使用して、「iptables_init」起動スクリプトを「オン」に設定します：

```
bash-4.3# chkconfig iptables_init on
```

「iptables_init」起動スクリプトは、リロードを実行するたびに実行されます。これで iptable ルールを永続的にすることができました。

NX-API 応答コードの表

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

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。

リクエスト形式が XML または JSON フォーマットの場合、NX-API エラー、エラー コード、および NX-API 応答のメッセージは次のとおりです。



(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 9: NX-API 応答コード

[NX-API 応答 (NX-API Response)]	コード	メッセージ
成功	200	成功。
CUST_OUTPUT_PIPED	204	要求により、出力は別の場所にパイプされます。
BASH_CMD_ERR	400	Bash コマンドエラー。
CHUNK_ALLOW_ONE_CMD_ERR	400	チャンクは、1つのコマンドだけを受け入れます。
CLI_CLIENT_ERR	400	CLI の実行エラー
CLI_CMD_ERR	400	CLI コマンドエラーの入力。
EOC_NOT_ALLOWED_ERR	400	eoc 値は、リクエストのセッション ID として許可されていません。
IN_MSG_ERR	400	着信メッセージが無効です。
INVALID_REMOTE_IP_ERR	400	要求のリモートIPを取得できません。
MSG_VER_MISMATCH	400	メッセージバージョンの不一致
NO_INPUT_CMD_ERR	400	入力コマンドがありません。
SID_NOT_ALLOWED_ERR	400	セッション ID として入力された文字が無効です。
PERM_DENY_ERR	401	権限が拒否されました。

CONF_NOT_ALLOW_SHOW_ERR	405	構成モードは [表示 (show)] を許可しません。
SHOW_NOT_ALLOW_CONF_ERR	405	表示モードでは構成できません。
EXCEED_MAX_SHOW_ERR	413	連続する show コマンドの最大数を超過しました。最大値は 10 です。
MSG_SIZE_LARGE_ERR	413	応答サイズが大きすぎます。
RESP_SIZE_LARGE_ERR	413	応答サイズが最大メッセージサイズを超えたため、処理を停止しました。最大サイズは 200 MB です。
EXCEED_MAX_INFLIGHT_CHUNK_REQ_ERR	429	同時チャンク リクエストの最大数を超過しています。最大は 2 です。
MAX_SESSIONS_ERR	429	最大セッション数に到達しました。新しいユーザー/クライアントの場合は、しばらくしてからもう一度お試しください。
OBJ_NOT_EXIST	432	要求したオブジェクトが存在しません。
BACKEND_ERR	500	バックエンド処理エラー。
CREATE_CHECKPOINT_ERR	500	チェックポイントの作成をするエラー。
DELETE_CHECKPOINT_ERR	500	チェックポイントの削除中にエラーが発生しました。
FILE_OPER_ERR	500	システム内部ファイル操作エラー。
LIBXML_NS_ERR	500	システムの内部 LIBXML NS エラー。これは要求フォーマットのエラーです。
LIBXML_PARSE_ERR	500	システムの内部 LIBXML 解析エラー。これは要求フォーマットのエラーです。
LIBXML_PATH_CTX_ERR	500	システムの内部 LIBXML パス コンテキスト エラー。これは要求フォーマットのエラーです。
MEM_ALLOC_ERR	500	システムの内部メモリ割り当てエラー。
ROLLBACK_ERR	500	ロールバックの実行中にエラーが発生しました。
SERVER_BUSY_ERR	500	サーバーがビジー状態のため、リクエストは拒否されました。

USER_NOT_FOUND_ERR	500	入力またはキャッシュからユーザーが見つかりません。
VOLATILE_FULL	500	揮発性メモリは一杯です。メモリ スペースを解放して、再試行してください。
XML_TO_JSON_CONVERT_ERR	500	XML から JSON への変換エラー。
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash コマンドはサポートされていません。
CHUNK_ALLOW_XML_ONLY_ERR	501	チャンクは XML 出力のみを許可します。
CHUNK_ONLY_ALLOWED_IN_SHOW_ERR	501	応答のチャンクは、show コマンドでのみ許可されます。
CHUNK_TIMEOUT	501	チャンク応答の生成中にタイムアウトしました。
CLI_CMD_NOT_SUPPORTED_ERR	501	CLI コマンドはサポートされていません。
JSON_NOT_SUPPORTED_ERR	501	大量の出力の可能性があるため、JSON はサポートされていません。
MALFORMED_XML	501	不正な XML 出力。
MSG_TYPE_UNSUPPORTED_ERR	501	メッセージ タイプはサポートされていません
OUTPUT_REDIRECT_NOT_SUPPORTED_ERR	501	出力リダイレクトはサポートされていません。
PIPE_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	このコマンドへのパイプ XML は入力では許可されていません。
PIPE_NOT_ALLOWED_IN_INPUT	501	この入力タイプにはパイプを使用できません。
RESP_BIG_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。
RESP_BIG_USE_CHUNK_ERR	501	応答が許容最大値を超えています。最大は 10 MB です。チャンクを有効にして XML または JSON 出力を使用します。
STRUCT_NOT_SUPPORTED_ERR	501	構造化出力はサポートされていません。
ERR_UNDEFINED	600	不明なエラー。

JSON および XML 構造化出力

NX-OS は、次の構造化された出力フォーマットで、さまざまな **show** コマンドの標準規格出力のリダイレクトをサポートしています。

- XML
- JSON. JSON 出力の上限は 60 MB です。
- JSON フォーマット出力の標準規格ブロックを読みやすくした JSON Pretty もあります。JSON 出力の上限は 60 MB です。
- NX-OS リリース 9.3 (1) で導入された JSON Native と JSON Pretty Native は、追加のコマンド解釈レイヤーをバイパスすることにより、JSON 出力をより高速かつ効率的に表示します。JSON Native および JSON Pretty Native は、出力のデータ型を保持します。出力用の文字列に変換する代わりに、整数を整数として表示します。

NX-OS CLI で、標準の NX-OS 出力を JSON または XML インタープリターに「パイプ接続」すると、これらのフォーマットへの変換が行われます。たとえば、**show ip access** コマンドを発行する際、論理パイプ (|) を続けて、その後に出力形式を指定できます。こうすると、NX-OS コマンドの出力が適切に構造化され、その形式でエンコードされます。この機能により、プログラムによるデータの解析が可能になり、ソフトウェア ストリーミング テレメトリを介したスイッチからのストリーミングデータがサポートされます。Cisco NX-OS のほとんどのコマンドは、JSON、JSON Pretty、JSON ネイティブ、JSON ネイティブ Pretty、および XML 出力をサポートしています。整合性チェッカー コマンドなど、一部のコマンドは、すべての形式をサポートしていません。整合性チェッカー コマンドは XML をサポートしていますが、JSON のバリエーションはどれもサポートしていません。



- (注) 検証エラーを回避するには、ファイルリダイレクトを使用して JSON 出力をファイルにリダイレクトし、そのファイル出力を使用します。

例:

```
Switch#show version | json > json_output ; run bash cat /bootflash/json_output
```

この機能の選択された例を以下に表示します。

JSON の概要 (JavaScript オブジェクト表記)

JSON は、判読可能なデータのために設計された軽量テキストベースのオープンスタンダードで、XML の代替になります。JSON はもともと JavaScript から設計されましたが、言語に依存しないデータ形式です。コマンド出力では、JSON および JSON プリティ形式、および JSON ネイティブおよび JSON プリティネイティブがサポートされています。

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

コマンドの JSON または XML 出力には、NX-API サンドボックスからもアクセスできます。

show

CLI の実行

```
switch-1-vxlan-1# show cdp neighbors | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "83886080", "device_id": "SW-SWITCH-1", "intf_id": "mgmt0", "ttl": "148", "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco AA-C0000 S-29-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device_id": "SWITCH-1-VXLAN-1(FOC1234A01B)", "intf_id": "Ethernet1/1", "ttl": "166", "capability": ["router", "switch", "IGMP_cnd_filtering", "Supports-STP-Dispute"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
```

BLR-VXLAN-NPT-CR-179#

XML および JSON 出力の例

このセクションでは、XML および JSON 出力として表示される NX-OS コマンドの例について説明します。

次の例は、ハードウェア テーブルのユニキャストおよびマルチキャスト ルーティング エントリを JSON 形式で表示する方法を示しています。

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096", "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "used_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_total": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "used_mcast_oif1": "2", "used_host_in_host_total": "13", "used_host4_in_host": "12", "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://www.cisco.com/nxos:1.0:fib">
  <nf:data>
    <show>
      <hardware>
        <profile>
          <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_oif1>2</used_mcast_oif1>
    <used_host_in_host_total>13</used_host_in_host_total>
    <used_host4_in_host>12</used_host4_in_host>
    <used_host6_in_host>1</used_host6_in_host>
    <max_ecmp_table_limit>64</max_ecmp_table_limit>
    <used_ecmp_table>0</used_ecmp_table>
    <mfib_fd_status>Disabled</mfib_fd_status>
    <mfib_fd_maxroute>0</mfib_fd_maxroute>
    <mfib_fd_count>0</mfib_fd_count>
  </__readonly__>
</__XML__OPT_Cmd_dynamic_tcam_status__readonly__>
</__XML__OPT_Cmd_dynamic_tcam_status>
</status>
</profile>
</hardware>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#

```

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

```

switch(config)# show lldp timers | json
{"ttl": "120", "reinit": "2", "tx_interval": "30", "tx_delay": "2", "hold_mplier": "4", "notification_interval": "5"}
switch(config)#

```

この例では、XML 形式でスイッチ上に LLDP タイマーを表示する方法を示します。

```

switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:lldp">
  <nf:data>
    <show>
      <lldp>
        <timers>
          <__XML__OPT_Cmd_lldp_show_timers__readonly__>
            <__readonly__>
              <ttl>120</ttl>
              <reinit>2</reinit>
              <tx_interval>30</tx_interval>
              <tx_delay>2</tx_delay>
              <hold_mplier>4</hold_mplier>
              <notification_interval>5</notification_interval>
            </__readonly__>
          </__XML__OPT_Cmd_lldp_show_timers__readonly__>
        </timers>
      </lldp>
    </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#

```

この例は、ACL 統計を XML 形式で表示する方法を示しています。

```
switch-1(config-acl)# show ip access-lists acl-test1 | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:1.0:aclmgr" xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nf:data>
    <show>
      <__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
        <ip_ipv6_mac>ip</ip_ipv6_mac>
        <access-lists>
          <__XML__OPT_Cmd_show_acl_name>
            <name>acl-test1</name>
            <__XML__OPT_Cmd_show_acl_capture>
              <__XML__OPT_Cmd_show_acl_expanded>
                <__XML__OPT_Cmd_show_acl__readonly__>
                  <__readonly__>
                    <TABLE_ip_ipv6_mac>
                      <ROW_ip_ipv6_mac>
                        <op_ip_ipv6_mac>ip</op_ip_ipv6_mac>
                        <show_summary>0</show_summary>
                        <acl_name>acl-test1</acl_name>
                        <statistics>enable</statistics>
                        <frag_opt_permit_deny>permit-all</frag_opt_permit_deny>
                      <TABLE_seqno>
                        <ROW_seqno>
                          <seqno>10</seqno>
                          <permitdeny>permit</permitdeny>
                          <ip>ip</ip>
                          <src_ip_prefix>192.0.2.1/24</src_ip_prefix>
                          <dest_any>any</dest_any>
                        </ROW_seqno>
                      </TABLE_seqno>
                    </ROW_ip_ipv6_mac>
                  </TABLE_ip_ipv6_mac>
                </__readonly__>
              </__XML__OPT_Cmd_show_acl__readonly__>
            </__XML__OPT_Cmd_show_acl_expanded>
          </__XML__OPT_Cmd_show_acl_capture>
        </__XML__OPT_Cmd_show_acl_name>
      </access-lists>
    </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
  </show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1(config-acl)#
```

この例は、ACL 統計を JSON 形式で表示する方法を示しています。

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

次の例は、スイッチの冗長ステータスを JSON 形式で表示する方法を示しています。

```
switch-1# show system redundancy status | json
{"rdn_mode_admin": "HA", "rdn_mode_oper": "None", "this_sup": "(sup-1)", "this_sup_rdn_state": "Active, SC not present", "this_sup_sup_state": "Active", "this_sup_internal_state": "Active with no standby", "other_sup": "(sup-1)", "other_sup_rdn_state": "Not present"}
```

```
nxosv2#
switch-1#
```

この例は、スイッチの冗長性情報を JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show system redundancy status | json-pretty native
{
  "rdn_mode_admin":      "HA",
  "rdn_mode_oper":       "None",
  "this_sup":            "(sup-1)",
  "this_sup_rdn_state":  "Active, SC not present",
  "this_sup_sup_state":   "Active",
  "this_sup_internal_state": "Active with no standby",
  "other_sup":           "(sup-1)",
  "other_sup_rdn_state":  "Not present"
}
switch-1#
```

次の例は、スイッチの OSPF ルーティング パラメータを JSON ネイティブ形式で表示する方法を示しています。

```
switch-1# show ip ospf | json native
{"TABLE_ctx":{"ROW_ctx":[{"ptag":"Blah","instance_number":4,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":0,"asopaque_lsa_cnt":0,"asopaque_lsa_crc":0,"area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"100","instance_number":3,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":0,"asopaque_lsa_cnt":0,"asopaque_lsa_crc":0,"area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false"},{"ptag":"111","instance_number":1,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":0,"asopaque_lsa_cnt":0,"asopaque_lsa_crc":0,"area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_number":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":0,"asopaque_lsa_cnt":0,"asopaque_lsa_crc":0,"area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"}]}
```

```
false}}}}
switch-1#
```

次の例は、OSPF ルーティング パラメータを JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show ip ospf | json-pretty native
{
  "TABLE_ctx": {
    "ROW_ctx": [{
      "ptag": "Blah",
      "instance_number": 4,
      "cname": "default",
      "rid": "0.0.0.0",
      "stateful_ha": "true",
      "gr_ha": "true",
      "gr_planned_only": "true",
      "gr_grace_period": "PT60S",
      "gr_state": "inactive",
      "gr_last_status": "None",
      "support_tos0_only": "true",
      "support_opaque_lsa": "true",
      "is_abr": "false",
      "is_asbr": "false",
      "admin_dist": 110,
      "ref_bw": 40000,
      "spf_start_time": "PT0S",
      "spf_hold_time": "PT1S",
      "spf_max_time": "PT5S",
      "lsa_start_time": "PT0S",
      "lsa_hold_time": "PT5S",
      "lsa_max_time": "PT5S",
      "min_lsa_arr_time": "PT1S",
      "lsa_aging_pace": 10,
      "spf_max_paths": 8,
      "max_metric_adver": "false",
      "asext_lsa_cnt": 0,
      "asext_lsa_crc": "0",
      "asopaque_lsa_cnt": 0,
      "asopaque_lsa_crc": "0",
      "area_total": 0,
      "area_normal": 0,
      "area_stub": 0,
      "area_nssa": 0,
      "act_area_total": 0,
      "act_area_normal": 0,
      "act_area_stub": 0,
      "act_area_nssa": 0,
      "no_discard_rt_ext": "false",
      "no_discard_rt_int": "false"
    }, {
      "ptag": "100",
      "instance_number": 3,
      "cname": "default",
      "rid": "0.0.0.0",
      "stateful_ha": "true",
      "gr_ha": "true",
      "gr_planned_only": "true",
      "gr_grace_period": "PT60S",
      "gr_state": "inactive",
      ... content deleted for brevity ...
      "max_metric_adver": "false",
```



```

        "asext_lsa_cnt":      0,
        "asext_lsa_crc":     "0",
        "asopaque_lsa_cnt":  0,
        "asopaque_lsa_crc":  "0",
        "area_total":       0,
        "area_normal":      0,
        "area_stub":        0,
        "area_nssa":        0,
        "act_area_total":    0,
        "act_area_normal":   0,
        "act_area_stub":     0,
        "act_area_nssa":     0,
        "no_discard_rt_ext": "false",
        "no_discard_rt_int": "false"
    }
}
}
switch-1#

```

次に、XML 形式で IP ルート要約を表示する例を示します。

```

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

```

```

        <best-paths>203</best-paths>
      </ROW_unicast>
    </ROW_unicast>
    <clientnameuni>ospf-10</clientnameuni>
    <best-paths>1038</best-paths>
  </ROW_unicast>
</TABLE_unicast>
<TABLE_route_count>
  <ROW_route_count>
    <mask_len>8</mask_len>
    <count>1</count>
  </ROW_route_count>
  <ROW_route_count>
    <mask_len>24</mask_len>
    <count>600</count>
  </ROW_route_count>
  <ROW_route_count>
    <mask_len>31</mask_len>
    <count>13</count>
  </ROW_route_count>
  <ROW_route_count>
    <mask_len>32</mask_len>
    <count>324</count>
  </ROW_route_count>
</TABLE_route_count>
</ROW_summary>
</TABLE_summary>
</ROW_addrf>
</TABLE_addrf>
</ROW_vrf>
</TABLE_vrf>
</__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
</__XML__OPT_Cmd_urib_show_ip_route_command_summary>
</__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
</__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
</__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
</__XML__OPT_Cmd_urib_show_ip_route_command_topology>
</__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip>
</route>
</ip>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1#

```

次の例は、JSON 形式で IP ルート要約を表示する例を示します。

```

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

```

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

```

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

```

次の例は、JSON ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json native
{"TABLE_vrf":{"ROW_vrf":[{"vrf-name-out":"default","TABLE_addrf":{"ROW_addrf":{"addrf":"ipv4","TABLE_summary":{"ROW_summary":[{"routes":3,"paths":3,"TABLE_unicast":{"ROW_unicast":[{"clientnameuni":"broadcast","best-paths":3}],TABLE_route_count":{"ROW_route_count":{"mask_len":8,"count":1},{"mask_len":32,"count":2}}}}}}}}]}
```

JSON ネイティブ（および JSON プリティネイティブ）では、整数が真の整数として表されることに注意してください。たとえば、「mask len:」は実際の値 32 として表示されます。

次の例は、JSON プリティ ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json-pretty native
{
  "TABLE_vrf": {
    "ROW_vrf": [{
      "vrf-name-out": "default",
      "TABLE_addrf": {
        "ROW_addrf": [{
          "addrf": "ipv4",
          "TABLE_summary": {
            "ROW_summary": [{
              "routes": 3,
              "paths": 3,
              "TABLE_unicast": {
                "ROW_unicast": [{
                  "clientnameuni": "broadcast",
                  "best-paths": 3
                }]
              },
              "TABLE_route_count": {
                "ROW_route_count": [{
                  "mask_len": 8,
                  "count": 1
                }, {
                  "mask_len": 32,
                  "count": 2
                }]
              }
            }]
          }
        }]
      }
    ]
  }
}
```

サンプル NX-API スクリプト

ユーザーはNX-APIでスクリプトを使用する方法を示すサンプルスクリプトにアクセスできます。サンプルスクリプトにアクセスするには、次のリンクをクリックして、必要なソフトウェア リリースに対応するディレクトリを選択します：[Cisco Nexus 9000 NX-OS NX-API](#)



第 21 章

NX-API REST

この章は次のトピックで構成されています。

- [NX-API REST について \(253 ページ\)](#)
- [REST による DME 設定の置換 \(254 ページ\)](#)

NX-API REST について

NX-API REST

リリース 7.0(3)I2(1) では、NX-API REST SDK が追加されました。

Cisco Nexus スイッチでは、構成はコマンドラインインターフェイス (CLI) を使用して実施します。CLI は、当該スイッチ上でしか実行できません。NX-API REST は、HTTP/HTTPS API を提供することにより、Cisco Nexus 構成のアクセシビリティを向上させます。

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

NX-API REST は、**show** コマンド、基本および詳細スイッチ構成と Linux Bash をサポートします。

NX-API REST は HTTP/HTTPS をトランスポートとして使用します。CLI は、HTTP / HTTPS POST 本文にエンコードされます。NX-API REST バックエンドは Nginx HTTP サーバーを使用します。Nginx プロセスとそのすべての子プロセスは、CPU とメモリの使用量の上限が定められている Linux cgroup の保護下に置かれます。NX-API プロセスは、cgroup ext_ser_nginx の一部であり、2,147,483,648 バイトのメモリに制限されています。Nginx のメモリ使用量が cgroup の制限を超えると、Nginx プロセスは再起動されて、NX-API 構成 (VRF、ポート、証明書構成) が復元されます。

Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、<https://developer.cisco.com/docs/nx-os-n3k-n9k-api-ref/> を参照してください。

REST による DME 設定の置換

REST Put による DME フル構成置換について

Cisco NX-OS リリース 9.3(1) 以降、Cisco NX-OS は REST PUT 操作によるモデルベースの完全な設定置換をサポートします。設定を置き換えるこの方法では、Cisco DME モデルを使用します。

DME フル コンフィギュレーションの置換機能を使用すると、REST プログラム インターフェイスを使用してスイッチの実行コンフィギュレーションを置換できます。この機能には、次の追加の利点があります。DME の完全な設定置換は、PUT 操作によって行われます。設定ツリーのすべての部分（システムレベル、サブツリー、およびリーフ）は、DME の完全な設定の置換をサポートします。

- スイッチ設定の無停止交換をサポート
- 自動化のサポート
- 他の機能やその構成に影響を与えることなく、機能を選択的に変更する機能を提供します。
- 最終的な設定結果を指定できるようにすることで、設定変更を簡素化し、人的エラーを排除します。スイッチは差分を計算し、構成ツリーの影響を受ける部分にプッシュします。



(注) プログラム インターフェイスを使用して実行することはできませんが、Cisco NX-OS CLI コマンドを使用して完全な設定置換を実行することもできます。 **config replace config-file-name**

注意事項と制約事項

以下は DME フル構成置換機能の注意事項と制約事項です。

- リリース 9.3(x) より前の Cisco NX-OS でサポートされるプラットフォームについては、[プログラマビリティ機能のプラットフォーム サポート \(7 ページ\)](#) を参照してください。Cisco NX-OS リリース 9.3(x) 以降、サポートされるプラットフォームの詳細については、[Nexus Switch Platform Matrix](#) を参照してください。
- DME は N9K-92348GC-X ではサポートされていません。
- ツリーを確認し、構成置換が適用される場所を確認することが重要です。構成置換操作にサンドボックスを使用する場合、サンドボックスはデフォルトでサブツリーに設定されるため、構成ツリー内の正しいノードをターゲットとするように URI を変更する必要があります。

- NX-OS サンドボックスを使用して（置換のための）変換を行う場合、要求に `status: 'replaced'` 属性が存在するため、POST 操作を使用する必要があります。他の変換オプションを使用する場合は、PUT 操作を使用できます。
- サブツリーノードでこの機能の REST PUT オプションを使用すると、構成置換操作がサブツリー全体に適用されます。ターゲットのサブツリー ノードは PUT の構成置換データで正しく変更されますが、サブツリー ノードのリーフ ノードもデフォルト値に構成されることによって影響を受けることに注意してください。

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

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

REST PUT によるシステムレベル構成の置換

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

次の手順を使用します。

手順

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

ペイロードは有効な構成である必要があり、構成は

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

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

ステップ 3 （オプション）送信したペイロードを、置き換えた DN の GET と比較します。GET のペイロードは、送信したペイロードと同じである必要があります。

REST PUT による機能レベルの構成置換

Cisco DME は、REST PUT 操作による機能レベルの設定の置換をサポートしています。モデルの機能レベルで PUT を送信することで、特定の機能の設定を置き換えることができます。

次の手順を使用します。

手順

ステップ 1 クライアントから、機能のモデルオブジェクト（MO）で REST PUT 操作を発行します。

- a) Put は、最上位システムレベルから機能の MO への URL を指定する必要があります。

たとえば、BGP /api/mo/sys/bgp.json の場合

ペイロードは有効な設定である必要があり、機能の DN で GET を発行することで、いつでもスイッチから設定を取得できる必要があります。たとえば、BGP の場合、/api/mo/sys/bgp.json?rsp-subtree=full&rsp-prop-include=set-config-only です。

- b) 機能のペイロードは、置き換える MO（たとえば、bgp）で始まる必要があります。

次に例を示します。

```
{
  "bgpInst": {
    "attributes": {
      "asn": "100",
      "rn": "inst"
    },
    "children": [
      ... content removed for brevity ...
      {
        "bgpDom": {
          "attributes": {
            "name": "vrf1",
            "rn": "dom-vrf1"
          },
          "children": [
            {
              "bgpPeer": {
                "attributes": {
                  "addr": "10.1.1.1",
                  "inheritContPeerCtrl": "",
                  "rn": "peer-[10.1.1.1]"
                }
              }
            }
          ]
        }
      }
    ]
  },
  {
    "bgpDom": {
      "attributes": {
        "name": "default",
        "rn": "dom-default",
        "rtrId": "1.1.1.1"
      }
    }
  }
]
```

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

- ステップ3** (オプション) 送信したペイロードを、置き換えたDNのGETと比較します。GETのペイロードは、送信したペイロードと同じである必要があります。

REST POST によるプロパティレベルの構成の置換

シスコのDMEモデルは、REST POST 操作による CLI ベースの機能のプロパティレベルの構成置換をサポートしています。要求ペイロードを生成し、REST POST 操作を介してスイッチに送信することにより、NX-OS サンドボックスを介して機能のプロパティの構成を置き換えることができます。NX-OS サンドボックスの詳細については、[NX-API 開発者サンドボックス \(259 ページ\)](#) を参照してください。

手順

- ステップ1** HTTPS を介し、NX-OS サンドボックスを介してスイッチに接続し、ログイン情報を入力します。
- ステップ2** 作業エリアで、変更する機能の CLI を入力します。
- ステップ3** 作業エリアの下のフィールドで、構成する機能に対するツリー内の MO への URI を設定します。この MO レベルは Put 要求の送信先です。
- ステップ4** [方法 (Method)] で、NX-API (DME) を選択します。
- ステップ5** [入力タイプ (Input Type)] で、[CLI] を選択します。
- ステップ6** [変換 (Convert)] ドロップダウンリストから Convert (for replace) を選択して、[要求 (Request)] ペインでペイロードを生成します。
- ステップ7** スイッチへの **POST** 操作を使用する要求をクリックします。

(注)

プロパティレベルの構成置換は、構成がデフォルト構成の場合に失敗する可能性があります。これは、置換操作はすべての子MOを削除し、すべてのプロパティをデフォルトにリセットしようと試みるからです。

REST PUT の構成置換のトラブルシューティング

以下は、REST Put 操作による設定の置換が成功しない場合のトラブルシューティングに役立つ手順です。

手順

- ステップ1** 要求が有効かどうかを確認します。

URL、操作、およびペイロードが有効である必要があります。たとえば、URL が `api/mo/sys/foo.json` の場合、ペイロードは `foo` で始まる必要があります。

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

- 正常に設定されました
- 有効なデバイス設定から取得

設定プロパティのみを取得するには、`rsp-subtree=full&rsp-prop-include=set-config-only` をフィルタリングする GET を使用します。

ステップ 3 ペイロードを検証するには、DMEPOST 操作を使用してペイロードをスイッチに送信します。

ステップ 4 エラーをチェックして、MO の名前とプロパティがあることを確認します。

ステップ 5 ペイロードに MO の名前とプロパティも含まれていることを確認します。



第 22 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 \(259 ページ\)](#)
- [NX-API 開発者サンドボックス : NX-OS リリース 9.2 \(2\) 以降 \(273 ページ\)](#)

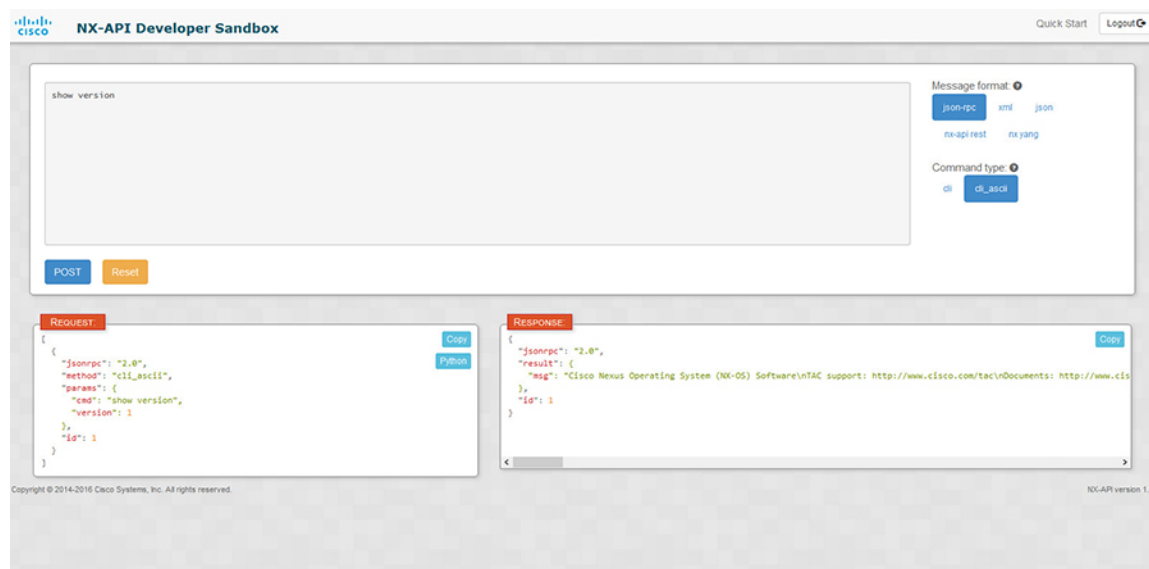
NX-API 開発者サンドボックス: 9.2 (2) より前の NX-OS リリース

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

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

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



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

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

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

デベロッパー サンドボックスの注意事項と制約事項

- 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。

注意事項と制約事項

デベロッパー サンドボックスのガイドラインと制限は次のとおりです：

- サンドボックスで **[送信 (Send)]** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。

- 一部の機能構成コマンドは、関連する機能が有効になるまで使用できません。たとえば、BGP ルータを構成するには、最初に **[機能 bgp (feature bgp)]** コマンドを使用して BGP を有効にする必要があります。同様に、OSPF ルータを構成するには、最初に **[機能 ospf (feature ospf)]** コマンドを使用して OSPF を有効にする必要があります。これは、**[evpn マルチホーミング コア トラッキング (evpn multihoming core-tracking)]** などの依存コマンドを有効にする **[evpn esi マルチホーミング (evpn esi multihoming)]** にも適用されます。機能が機能依存コマンドにアクセスできるようにする方法の詳細については、[Cisco Nexus 9000 Configuration Guides](#) [Cisco Nexus 3000 Configuration Guides](#) を参照してください。
- サンドボックスを使用した DN での変換は、CLI 構成の DN を検索する場合にのみサポートされます。DME を使用して CLI 構成コマンドの DN を変換するなど、他のワークフローはサポートされていません。
- コマンド ペイン（上部のペイン）は、最大 10,000 行の入力をサポートします。
- CLI 入力のメッセージタイプとして XML または JSON を使用する場合、セミコロンを使用して同じ行の複数のコマンドを区切ることができます。ただし、CLI 入力のメッセージタイプとして JSON RPC を使用する場合、同じ行に複数のコマンドを入力し、セミコロン (;) で区切ることはできません。

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

Sandbox で、次のように CLI を入力します。

```
show hostname ; show clock
```

JSON RPC リクエストでは、入力は次のようにフォーマットされます。

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

リクエストを送信すると、レスポンスで次のエラーが返されます。

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

この状況は、Sandbox が JSON RPC リクエストの各コマンドを個別のアイテムとして解析し、それぞれに識別子を割り当てるために発生します。JSON RPC リクエストを使用す

る場合、同じ回線で複数のコマンドを区切るために内部句読点を使用することはできません。代わりに、各コマンドを個別の回線に入力すると、リクエストが正常に完了します。

同じ例を続けて、NX-API CLI で次のようにコマンドを入力します。

```
show hostname
show clock
```

リクエストでは、入力は次のようにフォーマットされます。

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
  }
]
```

応答は正常に完了します。

```
[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]
```

メッセージフォーマットと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 10: NX-OS API プロトコル

プロトコル	説明
json-rpc	JSON ペイロードで NX-OS CLI コマンドを配信するために使用できる標準の軽量リモート プロシージャ コール (RPC) プロトコル。JSON-RPC 2.0 仕様は、 jsonrpc.org によって概説されています。
xml	XML ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。
json	JSON ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。
nx-api rest	内部 NX-OS データ管理エンジン (DME) モデルで管理対象オブジェクト (MO) とそのプロパティを操作および読み取るための Cisco NX-API 独自のプロトコル。Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、 https://developer.cisco.com/site/cisco-nexus-nx-api-references/ を参照してください。
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 11: コマンドタイプ

メッセージ形式	コマンドタイプ
json-rpc	- cli — show または構成コマンド - cli_ascii — show または構成コマンド、フォーマットせずに出力

メッセージ形式	コマンドタイプ
xml	• <code>cli_show</code> — 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。 • <code>cli_show_ascii</code> — コマンドを表示、フォーマットせずに出力 • <code>cli_conf</code> — 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。 • <code>bash</code> — <code>bash</code> コマンド。ほとんどの非対話型 <code>bash</code> コマンドがサポートされています。 (注) スイッチで <code>bash</code> シェルを有効にする必要があります。
json	• <code>cli_show</code> — 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。 • <code>cli_show_ascii</code> — コマンドを表示、フォーマットせずに出力 • <code>cli_conf</code> — 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。 • <code>bash</code> — <code>bash</code> コマンド。ほとんどの非対話型 <code>bash</code> コマンドがサポートされています。 (注) スイッチで <code>bash</code> シェルを有効にする必要があります。
nx-api rest	• <code>cli</code> — 構成コマンド • モデル — DN および対応するペイロード。
nx yang	• <code>json</code> — ペイロードに JSON 構造が使用されます • <code>xml</code> — XML 構造がペイロードに使用されます

出力チャンク

大量の `show` コマンド出力を処理するために、一部の NX-API メッセージフォーマットでは、`show` コマンドの出力チャンクがサポートされています。この場合、**[チャンクモードを有効にする (Enable chunk mode)]** チェックボックスが、セッション ID (SID) 入力ボックスとともに **[コマンドタイプ (Command Type)]** コントロールの下に表示されます。

チャンクが有効な場合、応答は複数の「チャンク」で送信され、最初のチャンクが即時のコマンド応答で送信されます。応答メッセージの次のチャンクを取得するには、前の応答メッセージのセッション ID に設定された **SID** を使用して NX-API 要求を送信する必要があります。

デベロッパー サンドボックスを使用

デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する



ヒント オンライン ヘルプは、サンドボックス ウィンドウの右上隅にある **[クイック スタート (Quick Start)]** をクリックすると利用できます。

応答コードやセキュリティ メソッドなどの詳細については、「NX-API CLI」の章を参照してください。

構成コマンドはサポートされていません。

手順

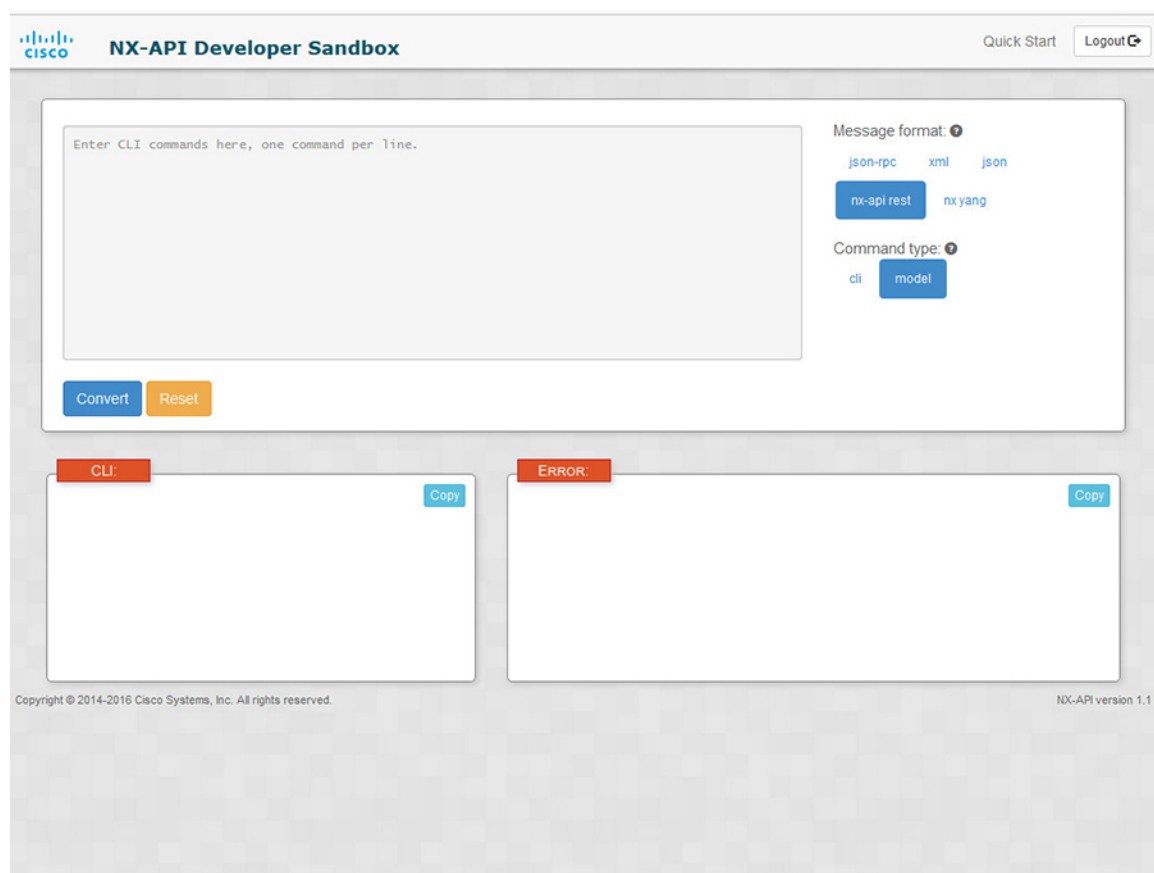
ステップ 1 使用する API プロトコルの **[メッセージ形式 (Message Format)]** と **[コマンドタイプ (Command Type)]** を構成します。

詳細な手順については、[メッセージフォーマットとコマンドタイプの構成 \(262 ページ\)](#) 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。

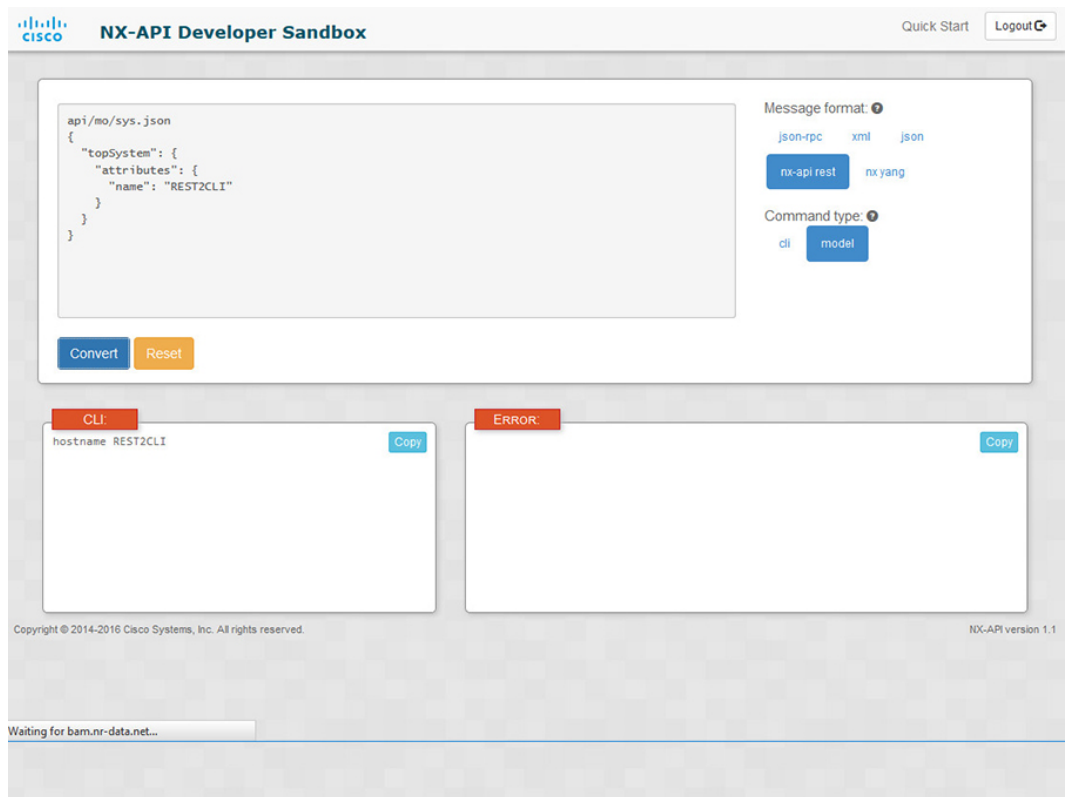
上部ペインの下部にある **[リセット (Reset)]** をクリックすると、テキスト エントリ ボックス (および **[要求 (Request)]** ペインと **[応答 (Response)]** ペイン) の内容を消去できます。

デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する



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。

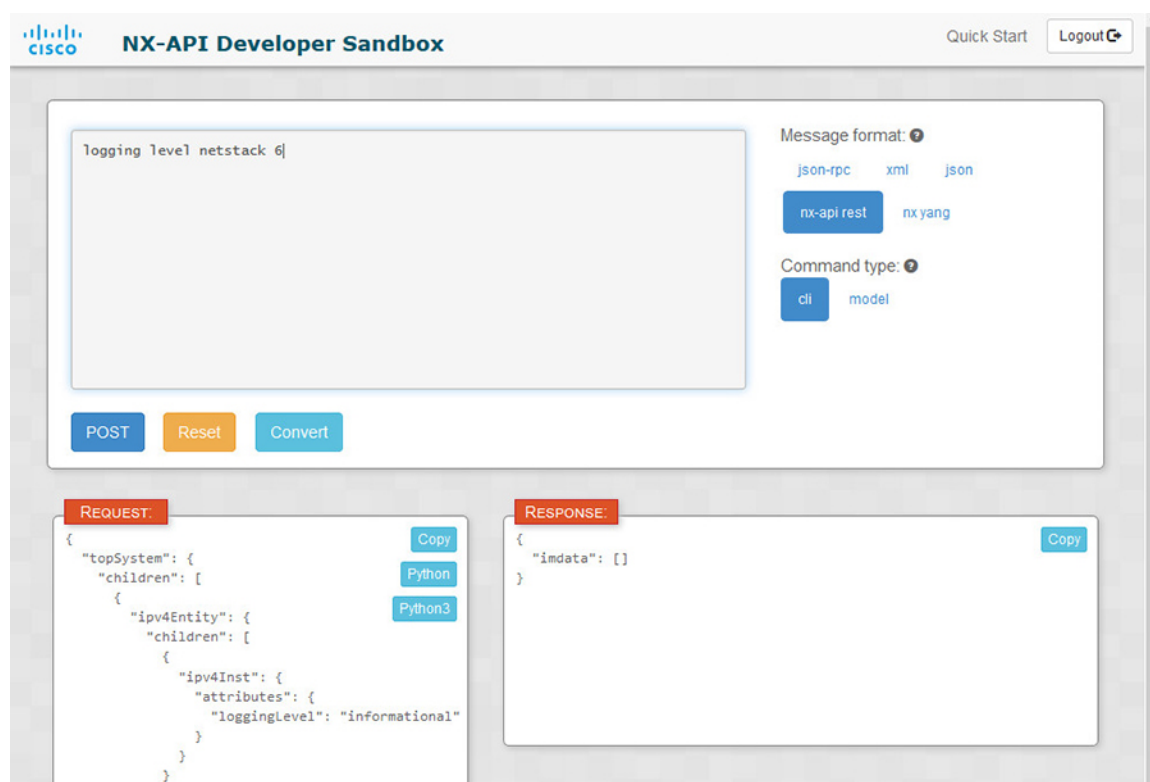


ステップ 4 [リクエスト (Request)] ペインに有効なペイロードが表示されている場合は、**POST** をクリックして、ペイロードを API 呼び出しとしてスイッチに送信できます。

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。

デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換



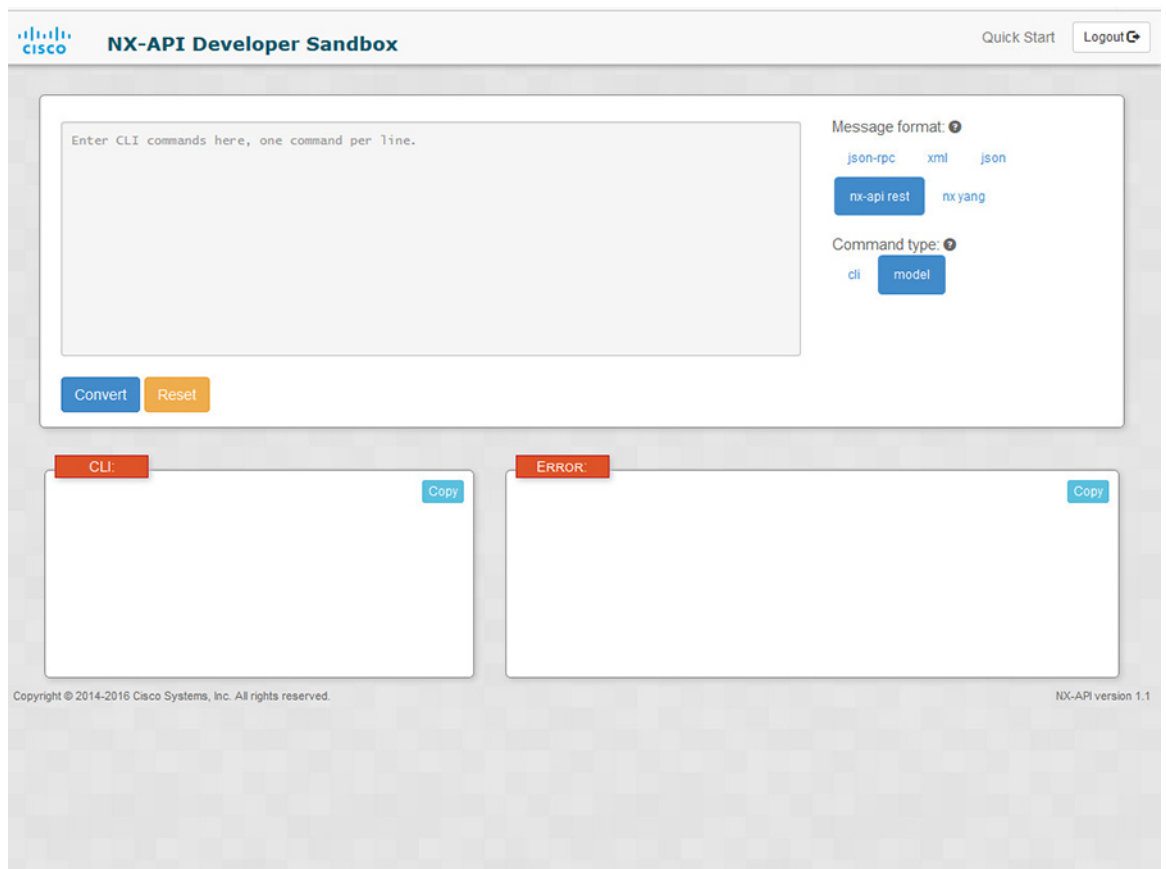
ヒント オンライン ヘルプは、サンドボックス ウィンドウの右上隅にある[クイック スタート (Quick Start)]をクリックすると利用できます。

応答コードやセキュリティ メソッドなどの詳細については、「NX-API CLI」の章を参照してください。

手順

ステップ1 メッセージフォーマットとして **nx-api rest** を選択し、コマンドタイプとして **model** を選択します。

例：

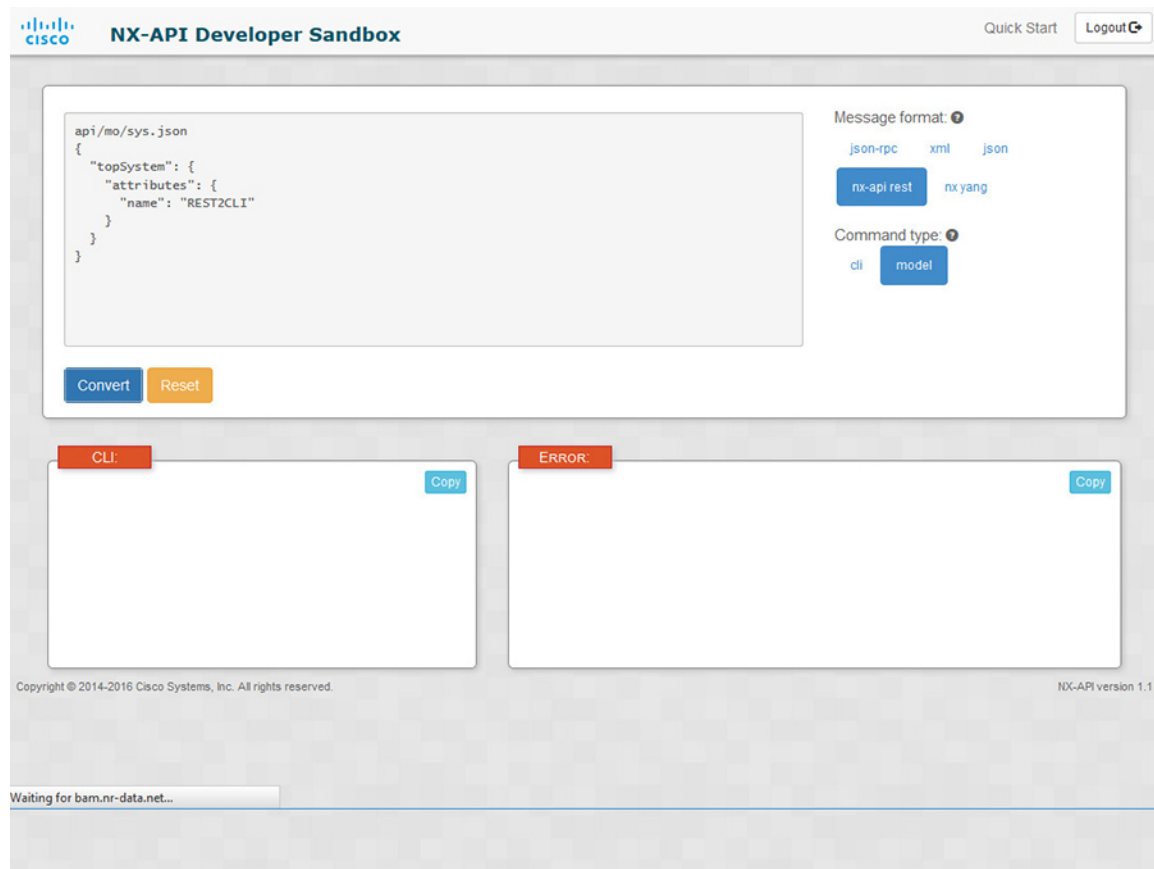


ステップ 2 上部ペインのテキスト入力ボックスに DN とペイロードを入力します。次に、上部ペインの下にある **[変換 (Convert)]** ボタンをクリックします。

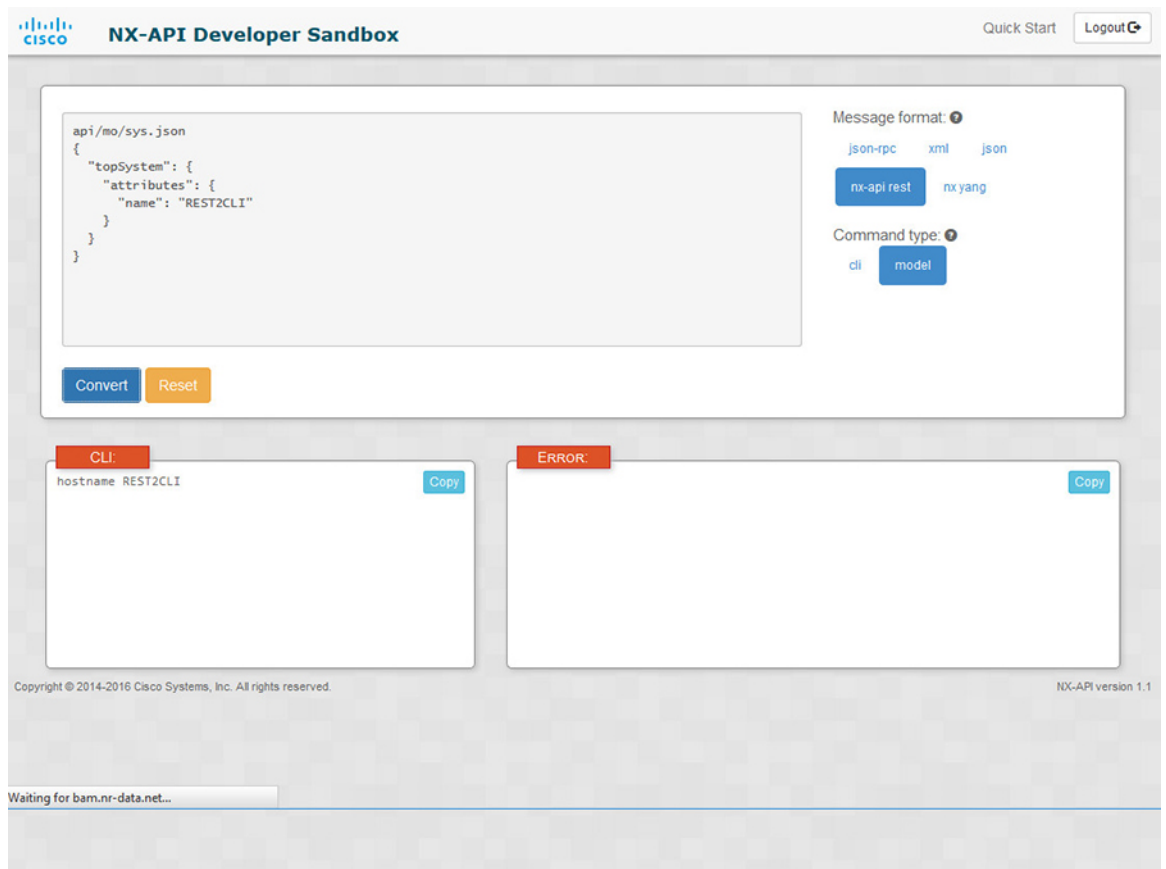
例：

この例では、DN は `/api/mo/sys.json` であり、NX-API REST ペイロードは次のとおりです：

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```



[変換 (Convert)] ボタンをクリックすると、次の図に示すように、同等の CLI が **CLI** ペインに表示されます。



(注)

デベロッパー サンドボックスは、サンドボックスが CLI を NX-API REST ペイロードに変換した場合でも、すべてのペイロードを同等の CLI に変換することはできません。以下は、ペイロードが CLI コマンドに完全に変換するのを妨げる可能性のあるエラーの原因のリストです。

表 12: REST2CLI エラーの原因

ペイロードの問題	結果
ペイロードに、MO に存在しない属性が含まれています。 例 : <pre>api/mo/sys.json { "topSystem": { "children": [{ "interfaceEntity": { "children": [{ "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": </pre>	[エラー (Error)] ペインは、属性に関連するエラーを返します。 例 : CLI 要素「l1PhysIf」の不明な属性「fakeattribute」の[エラー (Error)]
ペイロードには、変換がまだサポートされていない MO が含まれています。 例 : <pre>api/mo/sys.json { "topSystem": { "children": [{ "dhcpEntity": { "children": [{ "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" </pre>	[エラー (Error)] ペインは、サポートされていない MO に関連するエラーを返します。 例 : CLI [エラー (Error)] [「sys/dhcp」のサブツリー全体が変換されていません。(The entire subtree of "sys/dhcp" is not converted.)]

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

About the NX-API デベロッパー サンドボックス

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

Web フォームは、次の図に示すように、コマンド（上部のペイン）、要求（中央のペイン）、および応答（下部のペイン）の 3 つのペインを持つ 1 つの画面です。指定名（DN）フィールドは、コマンドペインとリクエストペインの間にあります（下図の **POST** と送信オプションの間にあります）。

リクエストペインにも一連のタブがあります。各タブは、**Python**、**Python3**、**Java**、**JavaScript**、**Go-Lang** の異なる言語を表します。各タブでは、それぞれの言語でリクエストを表示できます。たとえば、CLI コマンドを XML または JSON ペイロードに変換した後、**[Python]** タブをクリックして、スクリプトの作成に使用できる Python でのリクエストを表示します。

図 2: リクエストと出力応答の例を含む NX-API デベロッパー サンドボックス

The screenshot displays the NX-API Developer Sandbox web interface. At the top, there's a header with the Cisco logo, 'NX-API Sandbox', and links for 'Quick Start', 'Command Reference', and 'Logout'. The main area is divided into three horizontal sections. The top section contains a text input field with 'show version' and a 'POST' button. Below the input field are buttons for 'Send', 'Reset', and 'Output Schema'. The middle section shows a 'Request' tab selected, displaying a JSON payload for a 'show version' command. The bottom section shows the 'Response' tab, displaying the JSON response from the switch. The response includes the 'msg' field with the version information and a link to the support page.

```
{
  "jsonrpc": "2.0",
  "method": "cli_escal",
  "params": {
    "cmd": "show version",
    "version": 1
  },
  "id": 1
}
```

Response:

```
{
  "jsonrpc": "2.0",
  "result": {
    "msg": "Cisco Nexus Operating System (NX-OS) Software\nTAC support: http://www.cisco.com/tac\nDocuments: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home..."
  },
  "id": 1
}
```

コマンドペインのコントロールを使用すると、NX-API REST などのサポートされている API、モデル（ペイロード）や CLI などの入力タイプ、および XML や JSON などのメッセージ形式を選択できます。使用可能なオプションは、選択した方法によって異なります。

NXAPI-REST（DME）メソッドを選択し、1つ以上の CLI コマンドをコマンドペインに入力するか貼り付けて、**[変換]** をクリックすると、Web フォームはコマンドを REST API ペイロードに変換し、構成エラーをチェックし、要求ペインに結果のペイロードを表示します。次に、ペイロードをサンドボックスからスイッチに直接送信することを選択した場合（**POST** オプションを選択して **[SEND]** をクリック）、**[応答]** ペインに API 応答が表示されます。詳細については、[デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する（280 ページ）](#) を参照してください。

逆に、Cisco NX-API Developer Sandbox はペイロードの設定エラーをチェックし、対応する CLI を **[応答]** ペインに表示します。詳細については、「[デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換（283 ページ）](#)」を参照してください。

注意事項と制約事項

デベロッパー サンドボックスのガイドラインと制限は次のとおりです：

- サンドボックスで **[送信（Send）]** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。
- 一部の機能構成コマンドは、関連する機能が有効になるまで使用できません。たとえば、BGP ルータを構成するには、最初に **[機能 bgp（feature bgp）]** コマンドを使用して BGP を有効にする必要があります。同様に、OSPF ルータを構成するには、最初に **[機能 ospf（feature ospf）]** コマンドを使用して OSPF を有効にする必要があります。これは、**[evpn マルチホーミング コア トラッキング（evpn multihoming core-tracking）]** などの依存コマンドを有効にする **[evpn esi マルチホーミング（evpn esi multihoming）]** にも適用されます。機能が機能依存コマンドにアクセスできるようにする方法の詳細については、[Cisco Nexus 9000 Configuration Guides](#) [Cisco Nexus 3000 Configuration Guides](#) を参照してください。
- サンドボックスを使用した DN での変換は、CLI 構成の DN を検索する場合にのみサポートされます。DME を使用して CLI 構成コマンドの DN を変換するなど、他のワークフローはサポートされていません。
- コマンド ペイン（上部のペイン）は、最大 10,000 行の入力をサポートします。
- CLI 入力メッセージタイプとして XML または JSON を使用する場合、セミコロンを使用して同じ行の複数のコマンドを区切ることができます。ただし、CLI 入力メッセージタイプとして JSON RPC を使用する場合、同じ行に複数のコマンドを入力し、セミコロン（;）で区切ることはできません。

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

Sandbox で、次のように CLI を入力します。

```
show hostname ; show clock
```

JSON RPC リクエストでは、入力は次のようにフォーマットされます。

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

リクエストを送信すると、レスポンスで次のエラーが返されます。

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

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

同じ例を続けて、NX-API CLI で次のようにコマンドを入力します。

```
show hostname
show clock
```

リクエストでは、入力には次のようにフォーマットされます。

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
  }
]
```

応答は正常に完了します。

```
[
  {
    "jsonrpc": "2.0",
```

```

    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]

```

メッセージフォーマットと入力タイプの構成

メソッド、メッセージ形式、および入力タイプは、コマンドペイン（上部のペイン）の右上隅で構成されます。**[メソッド]**で、使用するAPIプロトコルの形式を選択します。Cisco NX-API Developer Sandbox は、次の API プロトコルをサポートしています。

表 13: **NX-OS API** プロトコル

プロトコル	説明
NXAPI-CLI	XML または JSON ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。
NXAPI-REST (DME)	内部 NX-OS データ管理エンジン（DME）モデルで管理対象オブジェクト（MO）とそのプロパティを操作および読み取るための Cisco NX-API 独自のプロトコル。NXAPI-REST（DME）プロトコルは、次の方法から選択できるドロップダウンリストを表示します。 • POST • GET • PUT • DELETE Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、https://developer.cisco.com/site/cisco-nexus-nx-api-references/ を参照してください。

プロトコル	説明
RESTCONF (Yang)	構成および状態データ用の YANG（「Yet Another Next Generation」）データモデリング言語。 RESTCONF（Yang）プロトコルは、次の方法から選択できるドロップダウンリストを表示します。 • POST • GET • PUT • PATCH • DELETE

メソッドを選択すると、メッセージ形式または入力タイプのオプションのセットがドロップダウンリストに表示されます。メッセージ形式は、入力 CLI を制約し、要求と応答の形式を決定できます。オプションは、選択したメソッドによって異なります。

次の表では、各メッセージ形式の入力/コマンドタイプオプションについて説明します。

表 14: コマンドタイプ

方法	メッセージ形式	入力/コマンドタイプ
NXAPI-CLI	json-rpc	• cli — show または構成コマンド • cli_ascii — show または構成コマンド、フォーマットせずに出力 • cli-array — show コマンド。cli に似ていますが、cli_array を使用すると、データは 1 つの要素のリスト、または角括弧 [] で囲まれたアレイとして返されます。

方法	メッセージ形式	入力/コマンドタイプ
NXAPI-CLI	xml	• <code>cli_show</code> — 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。 • <code>cli_show_ascii</code> — コマンドを表示、フォーマットせずに出力 • <code>cli_conf</code> — 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。 • <code>bash</code> — <code>bash</code> コマンド。ほとんどの非対話型 <code>bash</code> コマンドがサポートされています。 (注) スイッチで <code>bash</code> シェルを有効にする必要があります。
NXAPI-CLI	json	• <code>cli_show</code> — コマンドを表示します。コマンドがXML出力をサポートしていない場合、エラーメッセージが返されます。 (注) Cisco NX-OS リリース 9.3(3) 以降では、<code>cli_show</code> コマンドよりも <code>cli_show_array</code> コマンドが推奨されます。 • <code>cli_show_array</code> — <code>show</code> コマンド <code>cli_show</code> に似ていますが、<code>cli_show_array</code> を使用すると、データは角括弧 <code>[]</code> で囲まれた1つの要素のリストまたは配列として返されます。 • <code>cli_show_ascii</code> — コマンドを表示、フォーマットせずに出力 • <code>cli_conf</code> — 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。 • <code>bash</code> — <code>bash</code> コマンド。ほとんどの非対話型 <code>bash</code> コマンドがサポートされています。 (注) スイッチで <code>bash</code> シェルを有効にする必要があります。

方法	メッセージ形式	入力/コマンドタイプ
NXAPI-REST (DME)		• <code>cli</code> — CLI からモデルへの変換 • <code>model</code> — モデルから CLI への変換。
RESTCONF (Yang)	• <code>json</code> — ペイロードに JSON 構造が使用されます • <code>xml</code> — XML 構造がペイロードに使用されます	

出力チャンク

JSON および XML NX-API メッセージ形式を使用すると、10 MB のチャンクで大きな `show` コマンド応答を受信できます。受信すると、チャンクが連結されて、有効な JSON オブジェクトまたは XML 構造が作成されます。出力チャンクを示すサンプルスクリプトを表示するには、次のリンクをクリックし、リリース 9.3x に対応するディレクトリを選択します：[Cisco NX-OS NXAPI](#)。



- (注) チャンク JSON モードの場合、ブラウザーまたは Python スクリプトパーツは有効な JSON 出力を提供しません（終了タグはありません）。チャンクモードを使用して有効な JSON を取得するには、ディレクトリで提供されるスクリプトを使用します。

即時のコマンド応答で最初のチャンクを受け取ります。これには、セッション ID を含む `sid` フィールドも含まれます。次のチャンクを取得するには、前のチャンクのセッション ID を **[SID]** テキストボックスに入力します。`sid` フィールドの `eoc`（コンテンツの終わり）値で示される最後の応答に到達するまで、プロセスを繰り返します。

チャンクモードは、**JSON** または **XML** フォーマットタイプおよび `cli_show`、`cli_show_array`、または `cli_show_ascii` コマンドタイプで **NXAPI-CLI** メソッドを使用する場合に使用できます。チャンクモードの設定の詳細については、チャンクモードフィールドの表を参照してください。



- (注) NX-API は、最大 2 つのチャンクセッションをサポートします。

表 15: チャンク モード フィールド

フィールド名	説明
チャンク モードを有効にする	[チャンク モードを有効にする (Enable Chunk Mode)] チェックボックスをクリックしてチェックマークを付けると、チャンクが有効になります。チャンク モードを有効にすると、10 MB を超える応答は、最大 10 MB のサイズの複数のチャンクで送信されます。
SID	応答メッセージの次のチャンクを取得するには、SID テキスト ボックスに前の応答のセッション ID を入力します。 (注) 使用できる文字は英数字と「_」のみです。無効な文字はエラーを受け取ります。

デベロッパー サンドボックスを使用

デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する



ヒント

- Cisco NX-API デベロッパー サンドボックス ウィンドウの右上隅にあるフィールド名の横にあるヘルプアイコン (?) をクリックすると、オンライン ヘルプを利用できます。
- 応答コードやセキュリティ メソッドなどの詳細については、*NX-API CLI* の章を参照してください。
- 構成コマンドはサポートされていません。

Cisco NX-API Developer Sandbox を使用すると、CLI コマンドを REST ペイロードに変換できます。

手順

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

[入力タイプ] ドロップダウン リストが表示されます。

ステップ 2 [入力 (Input)] タイプドロップダウン リストをクリックし、**cli** を選択します。

ステップ 3 上部ペインのテキスト エントリ ボックスに、NX-OS CLI 構成コマンドを 1 行に 1 つずつ入力するか貼り付けます。

上部ペインの下部にある **[リセット (Reset)]** をクリックすると、テキスト エントリ ボックス (および **[要求 (Request)]** ペインと **[応答 (Response)]** ペイン) の内容を消去できます。

The screenshot shows the Cisco NX-API Sandbox web interface. At the top, there's a navigation bar with links for 'Quick Start', 'DME Documentation', 'Model Browser', and a 'Logout' button. The main area is divided into several sections. At the top, there's a large text box labeled 'Enter DME payload here.' with a 'Method:' dropdown set to 'NXAPI-REST (DME)' and an 'Input type:' dropdown set to 'model'. Below this is a text input field containing the CLI command '/api/mo/sys.json', with 'Send', 'Reset', and 'Convert' buttons to its right. Underneath the input field is a tabbed interface with tabs for 'Request', 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab is selected, showing a large empty text area with a 'Copy' button. Below the 'Request' tab is a 'Response:' section, also with a large empty text area and a 'Copy' button.

ステップ 4 [変換 (Convert)] をクリックします。

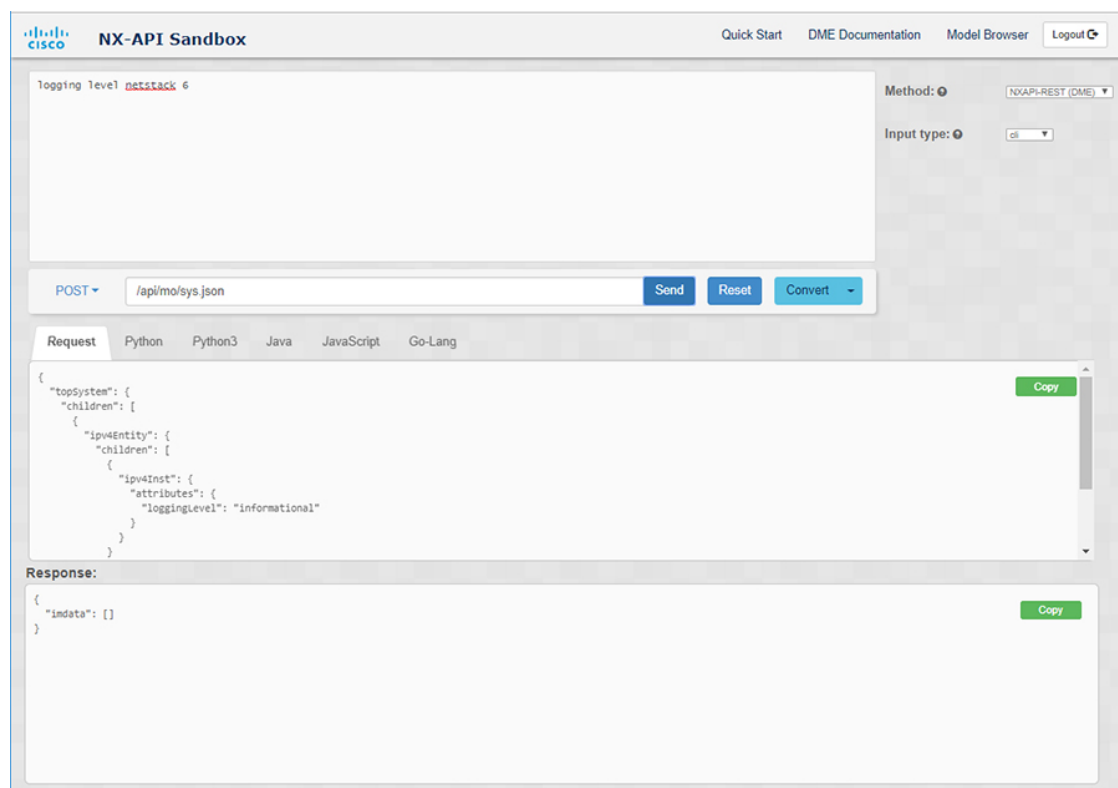
CLI コマンドに構成エラーが含まれていない場合、ペイロードは **[要求 (Request)]** ペインに表示されます。エラーが存在する場合は、説明のエラー メッセージが **[応答 (Response)]** ペインに表示されます。

ステップ 5 (オプション)有効なペイロードを API 呼び出しとしてスイッチに送信するには、**[送信 (Send)]** をクリックします。

スイッチからのレスポンスは **[Response (応答)]** ペインに表示されます。

警告

[送信 (Send)] をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。



ステップ 6 (オプション) ペイロード内の MO の DN を取得するには：

1. **[リクエスト (Request)]** ペインから、**POST** を選択します。
2. **[変換 (Convert)]** ドロップダウンリストをクリックし、**[変換(DNを使用) (Convert (with DN))]** を選択します。

ペイロードは、ペイロード内の各 MO に対応する DN を含む **dn** フィールドとともに表示されます。

ステップ 7 (オプション) 新しい構成で現在の構成を上書きする場合：

1. **[変換 (Convert)]** ドロップダウン リストをクリックし、**[変換 (置換用) (Convert (for Replace))]** を選択します。**[リクエスト (Request)]** ペインには、**[ステータス (status)]** フィールドが**[置換 (replace)]** ように設定されたペイロードが表示されます。
2. **[リクエスト (Request)]** ペインから、**POST** を選択します。
3. **[送信 (Send)]** をクリックします。

現在の構成は、投稿された構成に置き換えられます。たとえば、次の構成で開始するとします：

```
interface eth1/2
  description test
  mtu 1501
```

次に、[変換（置換用）（Convert（for Replace））]を使用して、次の構成を POST します。

```
interface eth1/2
  description testForcr
```

mtu 構成が削除され、新しい説明 (testForcr) のみがインターフェイスの下に表示されます。この変更は、**show running-config** と入力すると確認されます。

ステップ 8 (オプション) [リクエスト (Request)] ペインや [応答 (Response)] ペインなどのペインの内容をコピーするには、[コピー (Copy)] をクリックします。それぞれのペインの内容がクリップボードにコピーされます。

ステップ 9 (オプション) リクエストを以下のいずれかのフォーマットに変換するには、[リクエスト (Request)] ペインの適切なタブをクリックします。

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

デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換

Cisco NX-API Developer Sandbox を使用すると、REST ペイロードを対応する CLI コマンドに変換できます。このオプションは、NXAPI-REST (DME) メソッドでのみ使用できます。



ヒント

- Cisco NX-API Developer Sandbox のフィールド名の横にあるヘルプアイコン (?) をクリックすると、オンライン ヘルプを利用できます。ヘルプ アイコンをクリックして、それぞれのフィールドに関する情報を取得します。

応答コードやセキュリティ メソッドなどの詳細については、*NX-API CLI* の章を参照してください。

- Cisco NX-API Developer Sandbox の右上隅には、追加情報へのリンクが含まれています。表示されるリンクは、選択した**[方法 (Method)]**によって異なります。NXAPI-REST (DME) メソッドに表示されるリンク：
 - **[NX-API リファレンス (NX-API References)]** — 追加の NX-API ドキュメントにアクセスできます。
 - **[DME ドキュメント (DME Documentation)]** — NX-API DME モデル リファレンス ページにアクセスできます。
 - **[モデル ブラウザ (Model Browser)]** — モデル ブラウザである Visore にアクセスできます。Visore ページにアクセスするには、スイッチの IP アドレスを手動で入力する必要がある場合があることに注意してください。

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

手順

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

例：

ステップ 2 [タイプを入力 (Input Type)] タイプドロップダウンリストをクリックし、[モデル (model)] を選択します。

ステップ 3 要求ペインの上にあるフィールドに、ペイロードに対応する指定名 (DN) を入力します。

ステップ 4 コマンド ペインにペイロードを入力します。

ステップ 5 [変換 (Convert)] をクリックします。

例：

この例では、DN は **/api/mo/sys.json** であり、NX-API REST ペイロードは次のとおりです。

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```

The screenshot shows the Cisco NX-API Sandbox web interface. At the top, there's a header with the Cisco logo and 'NX-API Sandbox' title, along with links for 'Quick Start', 'DME Documentation', 'Model Browser', and a 'Logout' button. The main area is divided into two sections. The top section contains a JSON payload in a text area:

```
{  "topSystem": {    "attributes": {      "name": "RESTAPI"    }  }}
```

 To the right of the text area, there are dropdown menus for 'Method' (set to 'NX-API-REST (DME)') and 'Input type' (set to 'model'). Below the text area is a URL input field containing '/api/mo/sys.json', and three buttons: 'Send' (blue), 'Reset' (orange), and 'Convert' (blue). Below the URL field is a tabbed interface with 'Request' selected, and other tabs for 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab shows a large empty text area with a 'Copy' button in the top right corner. Below the 'Request' tab is a 'Response' section, also with a large empty text area and a 'Copy' button in the top right corner.

[変換 (Convert)] ボタンをクリックすると、次の図に示すように、同等の CLI が CLI ペインに表示されます。

The screenshot shows the Cisco NX-API Sandbox web interface. At the top, there's a header with the Cisco logo and 'NX-API Sandbox' title, along with links for 'Quick Start', 'DME Documentation', 'Model Browser', and a 'Logout' button. The main area is divided into two sections. The top section contains a JSON payload in a text area:

```
{  "configSystem": {    "attributes": {      "name": "REST2CLI"    }  }}
```

 To the right of the text area are dropdown menus for 'Method' (set to 'NX-API REST (DME)') and 'Input type' (set to 'model'). Below the text area is a URL input field containing '/api/mo/sys.json', and three buttons: 'Send' (blue), 'Reset' (orange), and 'Convert' (blue). The bottom section is titled 'Request' and has tabs for 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab is active, showing the converted CLI command: 'hostname REST2CLI'. To the right of this text is a green 'Copy' button. Below the 'Request' section is a 'Response' section, which is currently empty, with a green 'Copy' button to its right.

(注)

Cisco NX-API Developer Sandbox は、サンドボックスが CLI を NX-API REST ペイロードに変換した場合でも、すべてのペイロードを同等の CLI に変換できません。以下は、ペイロードが CLI コマンドに完全に変換するのを妨げる可能性のあるエラーの原因のリストです。

表 16: REST2CLI エラーの原因

ペイロードの問題	結果
ペイロードに、MO に存在しない属性が含まれています。 例 : <pre>api/mo/sys.json { "topSystem": { "children": [{ "interfaceEntity": { "children": [{ "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": </pre>	[エラー (Error)] ペインは、属性に関連するエラーを返します。 例 : CLI 要素「l1PhysIf」の不明な属性「fakeattribute」の[エラー (Error)]
ペイロードには、変換がまだサポートされていない MO が含まれています。 例 : <pre>api/mo/sys.json { "topSystem": { "children": [{ "dhcpEntity": { "children": [{ "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" </pre>	[エラー (Error)] ペインは、サポートされていない MO に関連するエラーを返します。 例 : CLI [エラー (Error)] [「sys/dhcp」のサブツリー全体が変換されていません。(The entire subtree of "sys/dhcp" is not converted.)]

デベロッパー サンドボックスを使用して RESTCONF から json または XML に変換する



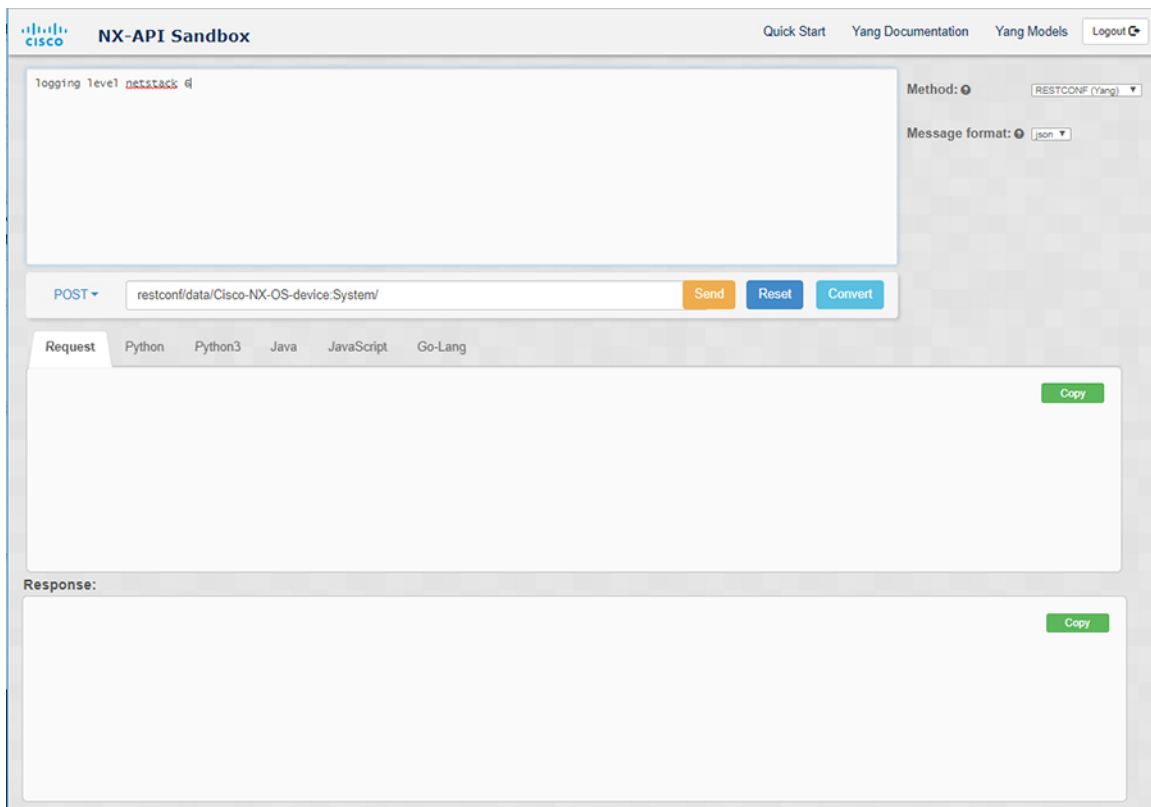
ヒント

- Cisco NX-API Developer Sandbox ウィンドウの右上隅にあるヘルプアイコン (?) をクリックすると、オンライン ヘルプを利用できます。
- [サンドボックス] ウィンドウの右上隅にある **Yang Documentation** リンクをクリックして、Model Driven Programmability with Yang ページに移動します。
- [サンドボックス] ウィンドウの右上隅にある **Yang Models** リンクをクリックして、YangModels GitHub サイトにアクセスします。

手順

ステップ 1 [メソッド] ドロップダウン リストをクリックし、[RESTCONF (Yang)] を選択します。

例：



ステップ 2 [メッセージ形式] をクリックし、**json** または **xml** を選択します。

ステップ 3 上部ペインのテキスト入力ボックスにコマンドを入力します。

ステップ 4 メッセージ形式を選択します。

ステップ 5 [変換 (Convert)] をクリックします。

例 :

この例では、コマンドはログ レベル **netstack 6** で、メッセージ形式は json です。

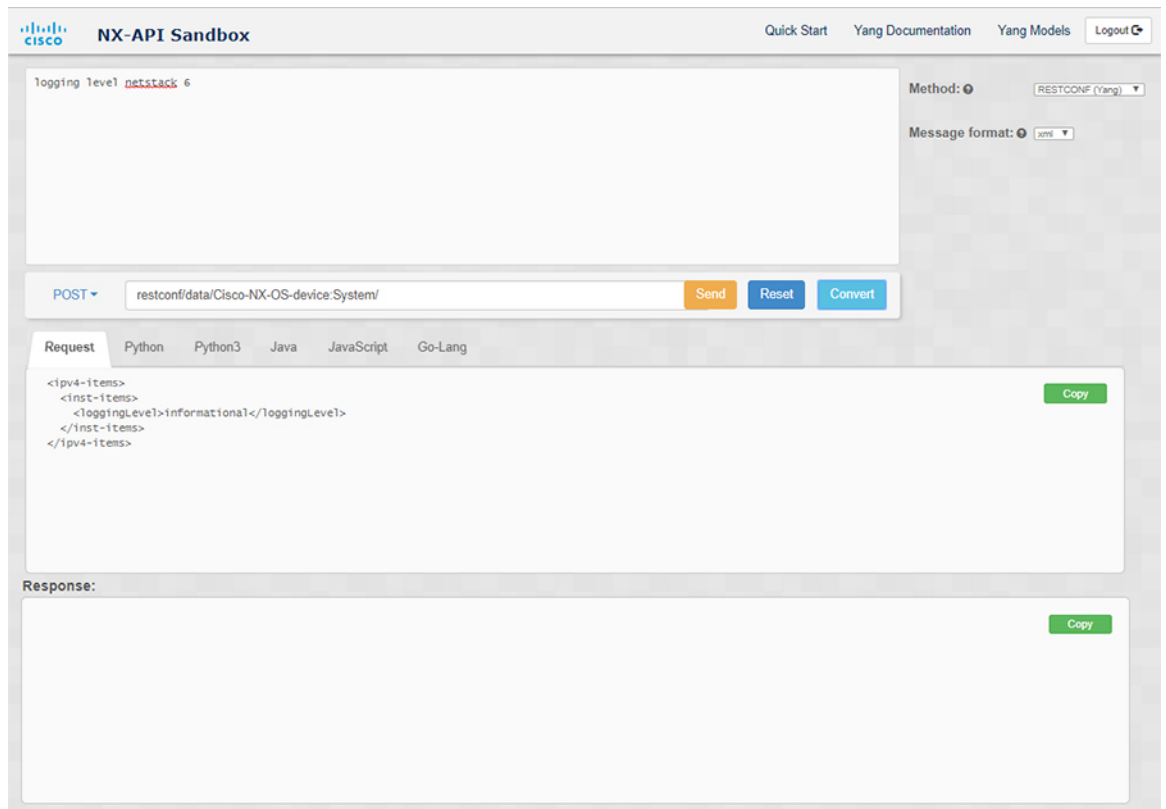
The screenshot shows the NX-API Sandbox interface. At the top, there's a header with the Cisco logo, 'NX-API Sandbox', and links for 'Quick Start', 'Yang Documentation', 'Yang Models', and 'Logout'. Below the header, there's a text area containing 'logging level netstack 6'. To the right of this area, there are two dropdown menus: 'Method:' set to 'RESTCONF (Yang)' and 'Message format:' set to 'json'. Below these, there's a 'POST' dropdown and a text input field containing 'restconf/data/Cisco-NX-OS-device:System/'. To the right of the input field are three buttons: 'Send' (orange), 'Reset' (blue), and 'Convert' (blue). Below the input field, there's a 'Request' tab selected, showing a JSON object:

```
{  "ipv4-items": {    "inst-items": {      "loggingLevel": "informational"    }  }}
```

 To the right of the JSON is a green 'Copy' button. Below the 'Request' tab is a 'Response' section, which is currently empty, with a green 'Copy' button to its right.

例 :

この例では、コマンドはログ レベル **netstack 6** で、メッセージ形式は xml です。



(注)

XML または JSON メッセージ形式を使用して、否定された CLI を Yang ペイロードに変換すると、サンドボックスは警告をスローし、**[送信]** オプションを無効にします。表示される警告メッセージは、メッセージの形式によって異なります。

- XML メッセージ形式の場合 — 「これは Netconf ペイロードであり、DELETE 操作用に生成されているため、Restconf では SEND オプションが無効になっています!」
- JSON メッセージ形式の場合 - 「これは、DELETE 操作用に生成される gRPC ペイロードであるため、Restconf では SEND オプションが無効になっています!」

ステップ 6 [リクエスト] ペインの適切なタブをクリックして、リクエストを次の形式に変換することもできます。

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

(注)

[リクエスト] タブの上の領域にあるドロップダウン メニューから [PATCH] オプションを選択した場合、Java で生成されたスクリプトは機能しません。これは Java の既知の制限であり、予期される動作です。



第 **IV** 部

モデル駆動型プログラマビリティ

- [インフラストラクチャの概要 \(295 ページ\)](#)
- [コンポーネント管理 \(299 ページ\)](#)
- [OpenConfig YANG \(305 ページ\)](#)
- [NETCONF エージェント \(317 ページ\)](#)
- [CLI コマンドのネットワーク構成フォーマットへの変換 \(343 ページ\)](#)
- [RESTConf エージェント \(349 ページ\)](#)
- [gRPC エージェント \(355 ページ\)](#)
- [gNMI : gRPC ネットワーク管理インターフェイス \(379 ページ\)](#)
- [ダイナミック ロガー \(421 ページ\)](#)
- [モデル駆動型テレメトリ \(429 ページ\)](#)



第 23 章

インフラストラクチャの概要

- [モデル駆動型プログラマビリティについて \(295 ページ\)](#)
- [プログラム可能なインターフェイス インフラストラクチャについて \(295 ページ\)](#)

モデル駆動型プログラマビリティについて

NX-OS デバイスのモデル駆動型プログラマビリティにより、デバイスの構成と制御を自動化できます。

データモデリング

データモデリングでは、手作業の構成プロセスの代わりに、プログラムを使用した標準ベースの方法でネットワーク デバイスの構成を記述します。データ モデルは、業界で定義された標準言語で記述されます。CLI を使用した設定のほうがわかりやすいですが、データモデルを使用して設定を自動化すると拡張性が向上します。

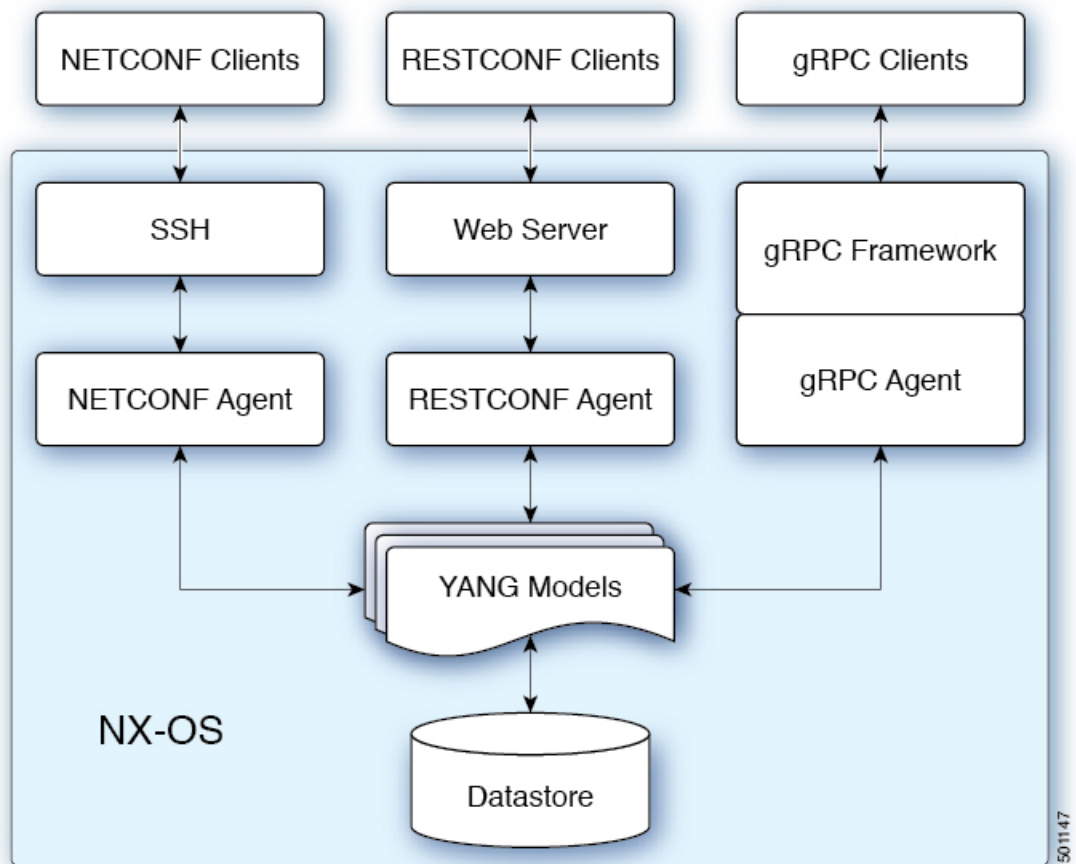
Cisco NX-OS デバイスは YANG データ モデリング言語をサポートしています。YANG は、ネットワーク デバイスの構成データと運用データ、リモート プロシージャ呼び出し、および通知を記述するために用いられるデータ モデリング 言語です。

プログラム可能なインターフェイス

NX-OS では、データ モデルでの操作のために 3 つの標準ベースのプログラマブル インターフェイス (NETCONF、RESTConf、および gRPC) がサポートされています。

プログラム可能なインターフェイス インフラストラクチャについて

ここでは、NX-OS プログラマブル インターフェイス インフラストラクチャの概要について説明します。



NETCONF、RESTConf または、gRPC 経由でのいずれかの方法で受信されても要求は、抽象メッセージオブジェクトに変換されます。そのメッセージオブジェクトは、要求内の名前空間に基づいて基盤となるモデルインフラストラクチャに配布されます。名前空間を使用すると、適切なモデルが選択され、要求は処理のために渡されます。モデルインフラストラクチャは、デバイスデータストアで要求（読み取りまたは書き込み）を実行します。結果は要求元のエージェントに返され、要求元のクライアントに応答が送信されます。

NX-OS プログラマブル インターフェイス エージェント

エージェントは、デバイスとクライアント間のインターフェイスを提供します。デバイスとの通信のトランスポート、プロトコル、およびエンコーディングを指定します。NX-OS プログラマブルインターフェイスは、NETCONF、RESTConf、および gRPC の 3 つのエージェントをサポートします。それぞれが YANG モデルを介してデバイスの構成管理に異なるインターフェイスを提供します。



(注) 各 Cisco NX-OS リリースでサポートされている YANG モデルについては、<https://devhub.cisco.com/artifactory/open-nxos-agents> を参照してください。

表 17: NX-OS プログラマブルインターフェイス エージェント

Agent	トランスポート	Protocol	エンコーディング
NETCONF	SSH	RFC 6241 ^[1]	XML
RESTCONF	HTTP	draft-ietf-netconf-restconf-10 ^[1]	XML または JSON
gRPC	HTTP	gRPC Protocol Spec ^[2]	Google Protobuf

プロトコルの仕様については、次のドキュメントを参照してください。

- [1] R. Enns, Ed., RFC6241: [ネットワーク構成プロトコル (Network Configuration Protocol (NETCONF))] <http://www.rfc-editor.org/rfc/rfc6241.txt>
- [1] RESTCONF プロトコル draft-ietf-netconf-restconf-10 <https://tools.ietf.org/html/draft-ietf-netconf-restconf-10>
- [2] Cisco NX-OS gRPC プロトコル仕様

モデル インフラストラクチャ

モデルインフラストラクチャは、エージェントから受信した要求を受け取り、要求内の YANG モデルに関連付けられている名前空間を決定し、名前空間に一致するモデルコンポーネントを選択して要求を処理します。選択したモデルコンポーネントが要求処理を完了すると、処理結果が要求元のエージェントに送信され、クライアントに送り返されます。モデルインフラストラクチャは、エージェントプロトコルの指定に従って、認証、ハンドシェイクなどを含むプロトコル開始要求の処理も行います。

デバイス YANG モデル

デバイス構成は、デバイスモデルと呼ばれる YANG モデルで記述されます。デバイス モデルは、デバイス名前空間を持つ別のモデル コンポーネントとしてモデル インフラストラクチャに明示されます。

共通 YANG モデル

共通モデルは、同等のデバイスモデル要素への YANG パスを要素内に含む、別の種類のモデル コンポーネントです。これら同等のデバイス モデル要素は、デバイス YANG コンテキストでデバイス モデルデータの読み取りと書き込みに使用されます。

その他の YANG 関連資料

YANG モデリング言語については、次のドキュメントで説明されています。M. Bjorklund, Ed., RFC6020: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) <http://www.rfc-editor.org/rfc/rfc6020.txt>

YANG の詳細については、YANG Central Wiki <https://handwiki.org/wiki/YANG> (M. Bjorklund, Ed.) を参照してください。



第 24 章

コンポーネント管理

- [コンポーネント RPM パッケージについて \(299 ページ\)](#)
- [インストールの準備 \(302 ページ\)](#)
- [Cisco Artifactory からのコンポーネントのダウンロード \(303 ページ\)](#)
- [RPM パッケージのインストール \(303 ページ\)](#)

コンポーネント RPM パッケージについて



(注) Cisco NX-OS リリース 7.0(3)I6(2) 以降では、Cisco NX-OS イメージに、NX-OS プログラマブルインターフェイスコンポーネント RPM パッケージ（エージェント、Cisco ネイティブモデル、その他の必要なほとんどのモデル、およびインフラストラクチャ）が含まれています。そのため、イメージがロードされると、必要なソフトウェアのほぼすべてが自動的にインストールされます。この状況は、Cisco Artifactory から大量のソフトウェアをダウンロードしてインストールする必要がないことを意味します。例外は、必須の OpenConfig モデルです。Cisco Artifactory から OpenConfig モデルを明示的にダウンロードする必要があります。

一方、Cisco NX-OS リリース 7.0(3)I6(1) 以前のリリースでは、アップグレードが必要な場合、パッケージのダウンロードとインストールについて説明する次のセクションに従うことが必要です。

NX-OS プログラマブルインターフェイスコンポーネント RPM パッケージは、Cisco Artifactory からダウンロードできます。必要なコンポーネント RPM パッケージには、次の 2 つのタイプがあります。

- 基本コンポーネント（必須）
- 共通モデルコンポーネント（OpenConfig モデルは明示的にダウンロードしてインストールする必要があります）

基本コンポーネント

基本コンポーネントは、次の必要な RPM パッケージで構成されます。

- **mtx-infra** : インフラストラクチャ
- **mtx-device** : Cisco ネイティブ モデル

モデル化された NX-OS インターフェイスにアクセスするには、次のエージェント パッケージの少なくとも 1 つをインストールする必要があります。

- **mtx-netconf-agent** : NETCONF エージェント
- **mtx-restconf-agent** : RESTCONF エージェント
- **mtx-grpc-agent** : gRPC エージェント

共通モデル コンポーネント

共通モデル コンポーネント RPM は、OpenConfig モデルをサポートします。OpenConfig モデルを使用するには、OpenConfig RPM をダウンロードしてインストールする必要があります。利便性のため、サポートされているすべての OpenConfig モデルの単一の結合パッケージ、mtx-openconfig-all があります。

単一の結合パッケージを推奨していますが、代わりに、選択したモデルと、次の表に記載されているサポート モデル間の依存関係を満たすための RPM をダウンロードしてインストールすることもできます。mtx-openconfig-all RPM は、個々のモデル RPM と互換性がありません。後者をインストールする場合はその前に前者をアンインストールし、また逆に、前者をインストールする場合はその前に後者をアンインストールする必要があります。

モデル名	モデル リビジョン	モデル Ver	パッケージ名	依存関係
openconfig-acl	2017-05-26	1.0.0	mtx-openconfig-acl	mtx-openconfig-interfaces
openconfig-bgp-policy	2017-07-30	4.0.1	mtx-openconfig-bgp-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-if-aggregate	2017-07-14	2.0.0	mtx-openconfig-if-aggregate	mtx-openconfig-if-ethernet mtx-openconfig-interfaces
openconfig-if-ethernet	2017-07-14	2.0.0	mtx-openconfig-if-ethernet	mtx-openconfig-interfaces
openconfig-if-ip	2016-05-26	1.0.2	mtx-openconfig-if-ip	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-vlan

モデル名	モデル リビジョン	モデル Ver	パッケージ名	依存関係
openconfig-if-ip-ext	2018-01-05	2.3.0	mtx-openconfig-if-ip-ext	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-if-ip mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-interfaces	2017-07-14	2.0.0	mtx-openconfig-interfaces	-
openconfig-network-instance	2017-08-24	0.8.1	mtx-openconfig-network-instance	mtx-openconfig-bgp-policy mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-routing-policy mtx-openconfig-vlan
openconfig-network-instance-policy	2017-02-15	0.1.0	mtx-openconfig-network-instance-policy	mtx-openconfig-routing-policy
openconfig-ospf-policy	2017-08-24	0.1.1	mtx-openconfig-ospf-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-platform	2018-01-16	0.8.0	mtx-openconfig-platform	-
openconfig-platform-linecard	2017-08-03	0.1.0	mtx-openconfig-platform-linecard	mtx-openconfig-platform
openconfig-platform-port	2018-01-20	0.3.0	mtx-openconfig-platform-port	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-platform-transceiver	2018-01-22	0.4.1	mtx-openconfig-platform-transceiver	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-relay-agent	2016-05-16	0.1.0	mtx-openconfig-relay-agent	mtx-openconfig-interfaces
openconfig-routing-policy	2016-05-12	2.0(1)	mtx-openconfig-routing-policy	-
openconfig-spanning-tree	2017-07-14	0.2.0	mtx-openconfig-spanning-tree	mtx-openconfig-interfaces
openconfig-system	2017-09-18	0.3.0	mtx-openconfig-system	-

モデル名	モデル リビジョン	モデル Ver	パッケージ名	依存関係
openconfig-vlan	2017-07-14	2.0.0	mtx-openconfig-vlan	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces

インストールの準備

ここでは、インストールの準備と、NX-OS プログラマブル インターフェイス コンポーネントを管理するためのその他の有用な情報について説明します。

デバイスで Bash シェルを開く

スイッチへの RPM のインストールは、Bash シェルで実行されます。デバイスで機能 **feature bash** が構成されていることを確認します。

```
Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# feature bash-shell
Switch(config)# end
Switch# run bash sudo su
bash-4.2#
```

Bash からデバイスの CLI プロンプトに戻るには、**exit** または **Ctrl-D** を入力します。

デバイスの準備状況の確認

RPM をインストールする前に、次の CLI **show** コマンドを使用してデバイスの準備状況を確認できます。

- **show module** : すべてのモジュールが稼働しているかどうかを示します。

```
Switch# show module
```

- **show system redundancy status** : スタンバイ デバイスが稼働しており、HA モードになっているかどうかを示します。スタンバイ同期が進行中の場合、RPM のインストールが失敗する可能性があります。

```
Switch# show system redundancy status
```

ラインカードが起動しない場合は、Bash シェルで **createrepo /rpms** コマンドを入力します。

```
bash-4.2# createrepo /rpms
```

デバイスへのファイルのコピー

SCP を使用すると、次の形式のコマンドを使用してファイルをデバイスにコピーできます。

```
copy scp://username@source_ip/path_to_agent_rpm bootflash: vrf management
```

例:

```
Switch# copy scp://jdoe@192.0.20.123//myrpms/mtx-infra.1.0.0.r082616.x86_64.rpm bootflash:  
vrf management
```

インストールされている **NX-OS** プログラマブル インターフェイス RPM の表示

インストールされているすべての NXOS プログラマブル インターフェイス RPM を表示するには、デバイスで次のコマンドを実行します。

```
bash-4.2# yum list installed | grep mtx
```

Cisco Artifactory からのコンポーネントのダウンロード

NX-OS プログラマブル インターフェイス コンポーネント RPM は、次の URL の Cisco Artifactory からダウンロードできます。RPM は、NX-OS リリースごとのディレクトリに整理されています。正しい NX-OS リリース ディレクトリから RPM をダウンロードしていることを確認してください。

<https://devhub.cisco.com/artifactory/open-nxos-agents>

NX-OS プログラマブル インターフェイス コンポーネント RPM は、次の命名規則に従っています。

`<package>-<version>-<NX-OS release>.<architecture>.rpm`

次のセクションの説明に従って、目的のインストールする NX-OS プログラマブル インターフェイス コンポーネント RPM パッケージを選択して、デバイスにダウンロードします。

RPM パッケージのインストール

プログラマブル インターフェイス ベースおよび共通モデル コンポーネント RPM パッケージのインストール

始める前に

- Cisco Artifactory から、次のパッケージをダウンロードします。
 - mtx-infra
 - mtx-device
 - mtx-netconf-agent/mtx-restconf-agent/mtx-grpc-agent (少なくとも 1 つ)
 - mtx-openconfig-all (または、選択した個々のモデル)

- [デバイスの準備状況の確認 \(302 ページ\)](#) の CLI コマンドを使用して、アクティブ デバイスとスタンバイ デバイスのすべてのライン カードが稼働しており、準備ができていることを確認します。

手順

ステップ 1 ダウンロードした RPM をデバイスにコピーします。

例 :

```
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy
scp://jdoe@192.0.20.123/myrpms/mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm bootflash:
vrf management
```

ステップ 2 Bash シェルから、RPM をインストールします。

例 :

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtx-infra-2.0.0.0-9.2.1.lib32_n9000.rpm
mtx-device-2.0.0.0-9.2.1.lib32_n9000.rpm mtx-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
mtx-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
```

ステップ 3 Bash シェルから、インストールを確認します。

例 :

```
bash-4.2# yum list installed | grep mtx
```




第 25 章

OpenConfig YANG

ここでは、次の内容について説明します。

- [OpenConfig YANG について](#) (305 ページ)
- [OpenConfig YANG のガイドラインと制限事項](#) (305 ページ)
- [BGP ルーティング インスタンスの削除について](#) (314 ページ)
- [YANG の検証](#) (315 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。

- IPv4 および IPv6 アドレスの場合、IP アドレス フィールド (**oc-ip:ip** および **oc-ip:prefix_length**) の削除と削除に同じ操作を提供する必要があります。

例：

```
oc-ip:ip: remove
oc-ip:prefix_length: remove
```

- OSPF アクションメトリックが BGP **set med** プロパティよりも優先されるため、OpenConfig NETCONF を介して同じルートマップ内のメトリックを使用した **set med** と一緒に BGP アクションおよび OSPF アクションを設定することは推奨されません。

2 つの異なるルートマップを使用して、OSPF アクションでメトリックを設定します。個別のルートマップを使用して BGP アクションの下で **set-med** を使用します。

単一のペイロードで、BGP アクションのメトリックを OSPF アクションに変更したり、OSPF アクションをルートマップの BGP アクションに変更したりしないことをお勧めします。

- 有効な BGP インスタンスを使用するには、自律システム (AS) 番号を指定する必要があります。AS 番号にデフォルト値が存在しないため、NETCONF / OPENCONFIG で削除しようとする `<asn>` BGP インスタンスを削除しないと、次の強調表示されたエラーメッセージが表示されます。

```

764
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:config>
      <network-instances xmlns="http://openconfig.net/yang/network-instance">
        <network-instance>
          <name>default</name>
          <protocols>
            <protocol>
              <identifier>BGP</identifier>
              <name>bgp</name>
              <bgp>
                <global>
                  <config nc:operation="delete">
                    <as>100</as>
                  </config>
                </global>
                <neighbors>
                  <neighbor>
                    <neighbor-address>1.1.1.1</neighbor-address>
                    <enable-bfd xmlns="http://openconfig.net/yang/bfd">
                      <config>
                        <enabled>true</enabled>
                      </config>
                    </enable-bfd>
                  </neighbor>
                </neighbors>
              </bgp>
            </protocol>
          </protocols>
        </network-instance>
      </network-instances>
    </nc:config>
  </nc:edit-config>
</nc:rpc>

##
Received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">
  <rpc-error>

```

```

        <error-type>protocol</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">invalid property value , for property asn, class
bgpInst</error-message>
        <error-path>/config/network-instances</error-path>
    </rpc-error>
    <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>operation-failed</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">invalid property value , for property asn, class
bgpInst Commit Failed</error-message>
        <error-path>/config/network-instances</error-path>
    </rpc-error>
</rpc-reply>

```

• OC-BGP-POLICY には、次の OpenConfig YANG 制限があります：

- アクションタイプは、community-set および as-path-set に対して常に [許可 (permit)] され、次のコンテナに適用されます。
 - /bgp-defined-sets/community-sets/community-set/
 - /bgp-defined-sets/as-path-sets/as-path-set/

OpenConfig YANG には、community-set および as-path-set の CLI にあるようなアクションタイプの概念はありません。したがって、community-set および as-path-set のアクションタイプは常に permit です。

- このコンテナには、次の OpenConfig YANG 制限が適用されます。
/bgp-defined-sets/community-sets/community-set/

CLI では、community-list には、標準と拡張の 2 つの異なるタイプがあります。ただし、OpenConfig YANG モデルでは、community-set-name にそのような区別はありません。

OpenConfig YANG を使用して community-set-name を作成すると、次のことが内部で発生します。

- community-member が標準形式 (AS:NN) の場合、community-set-name の後に _std サフィックスが追加されます。
- community-member が展開形式 (正規表現) の場合、community-set-name の後に _exp サフィックスが追加されます。

```

<community-set>
  <community-set-name>oc_commsetld</community-set-name>
  <config>
    <community-set-name>oc_commsetld</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_commsetld_exp seq 5 permit "_1_"
ip community-list standard oc_commsetld_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、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>
              </config>
            </switched-vlan>
          </ethernet>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```

```

        <trunk-vlans>999</trunk-vlans>
    </config>
</switched-vlan>
</ethernet>
</interface>
</interfaces>
</config>
</edit-config>
</rpc>

```

- OpenConfig YANG の設計により、VLAN を設定する場合、ペイロード内の VLAN とインターフェイスですでに設定されている VLAN との間に重複があってはなりません。オーバーラップが存在する場合、OpenConfig による構成は失敗します。インターフェイスに設定されている VLAN が、OpenConfig ペイロードの VLAN と異なることを確認してください。範囲内の開始 VLAN と終了 VLAN に特に注意してください。
- 次の注意事項および制約事項が OC-LACP に適用されます。
 - ポートチャネルモード：
 - OC-LACP を使用すると、ポートチャネル インターフェイスでポートチャネルモードを設定できます。ただし、NXOS-CLI を通じて、ポートチャネルモードは、チャンネル グループ モードのアクティブまたはパッシブを使用してメンバー インターフェイスで設定されます。
 - OC-LACP はポートチャネル インターフェイスでポートチャネルモードを明示的に設定しますが、ポートチャネル インターフェイスで NX-OS **show running-config** コマンドを発行しても、空または空でないポートチャネルのポートチャネルモード設定は表示されません。
 - メンバーがポートチャネルに追加されると、**show running interface ethernet <>** はポート チャネルモードの構成をチャンネル グループ モードのアクティブまたはパッシブとして表示されます。



(注) OpenConfig を介して作成されたすべてのポートチャネルは、引き続き OpenConfig によって管理される必要があります。

- ポートチャネルの間隔：
 - ポートチャネルの間隔は、メンバーがシャット状態の場合にのみ変更できます。
 - OC-LACP 間隔はポートチャネルごとです。NX-OS LACP 間隔は、ポートチャネル メンバーごとです。この違いにより、次の動作が予想されます。
 - OpenConfig を使用してポートチャネル間隔を設定すると、ポートチャネルのすべてのメンバーに同じ設定が適用されます。
 - OpenConfig を使用してポートチャネル間隔を構成し、後でメンバーがポートチャネルに追加された場合、設定を新しいメンバーに適用するには、OpenConfig を使用して間隔を再度設定する必要があります。

- システム MAC ID :

- このリリースでは、Cisco NX-OS はポートチャネルごとの system-id-mac をサポートしていません。

- 次のメンバー状態データは、ポートが管理 up state:状態の場合にのみ存在します。

- LACP
- インターフェイス
- インターフェイス
- メンバー
- 状態

- OpenConfig YANG を介してインターフェイスを追加しようとすると、OSPFv2 はエラー応答を送信できます。問題が発生すると、インターフェイスは追加されず、RPC 応答には次のように「リストのマージに失敗しました (list merge failed)」というエラーが含まれます。

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:39507023-8569-4cf8-869c-e19aaf76a260">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">List Merge Failed:
operation-failed</error-message>

    <error-path>/network-instances/network-instance/protocols/protocol/ospfv2/areas/area/interfaces/interface/ik</error-path>

  </rpc-error>
</rpc-reply>
```

- Hig (ii) ポートの統計のキューイングはサポートされていません。
- ユニキャスト、マルチキャスト、またはブロードキャスト キューごとの tx パケット、またはバイト、およびドロップ パケットは表示されません。OC 応答に表示される統計は、qos-group ごとの ucast、mcast、および bcast キューの合計です。
- OpenConfig YANG は、VLAN レベルで適用される QoS ポリシーの統計をサポートしていません。
- OC を介して取得できる入力キュー ドロップ数は、プラットフォームに応じてスライス/ポート/キュー レベルで表示できます。
- 以下は、switchport、shut/no shut、MTU、および MAC アドレスの OpenConfig 設定のガイドラインと制限です。
 - スイッチポート、shut/no shut、MTU、および MAC アドレスを設定する場合は、ASCII リロードが必要です。バイナリ リロードを使用すると、構成が失われます。

- 次の新しい動作状態 OpenConfig パスがサポートされています。一部のパスには、次に示す追加のガイドラインと制限事項があります。

- `/network-instances/network-instance/fdb/l2rib/mac-table`
 - l2rib の親レベルのクエリは、l2rib レベルでサポートされています。たとえば、`network-instances/network-instance/fdb/l2rib` まではクエリできますが、`fdb` レベルの `network-instances/network-instance/fdb` ではクエリできません。

- `/interfaces/interface/routed-vlan/ipv4/neighbors/neighbor/state`

- `/interfaces/interface/routed-vlan/ipv6/neighbors/neighbor/state`

- 親クエリの場合、インフラストラクチャはすべてのリスト項目のすべてのキーを取得し、これらの各リスト項目の残りのデータを入力する要求がバックエンドに送信されます。つまり、インフラストラクチャにはバックエンドと同じツリービューが必要です。

たとえば、インフラストラクチャに静的エントリのみが表示され、バックエンドに静的エントリと動的エントリがある場合、リストウォークでは、インフラストラクチャは各静的エントリの要求のみを送信し、不完全なデータになります。現在のリリースでこの制限があるパス

は、`/interfaces/interface/routed-vlan/ipv6/neighbors/neighbor/state` および `/interfaces/interface/routed-vlan/ipv4/neighbors/neighbor/state` です。正確なパスが指定されている場合、データにはダイナミックとスタティックの両方の ARP および ND エントリが含まれますが、親パスが指定されている場合はスタティック エントリのみが含まれます。

- `/network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/l2vpn-expr/loc-rib/routes`
- `/network-instances/network-instance/protocols/protocol/bgp/rib/attr-sets`
- `/network-instances/network-instance/protocols/protocol/bgp/rib/communities`
- `/network-instances/network-instance/protocols/protocol/bgp/rib/ext-communities`
- `/network-instances/network-instance/connection-points/connection-point/endpoints/endpoint/vlan/endpoint-peers`
- `/network-instances/network-instance/connection-points/connection-point/endpoints/endpoint/vlan/endpoint-vnis`

BGP ルーティング インスタンスの削除について

OpenConfig YANG ネットワーク インスタンス (OCNI) を使用して、BGP ルーティング インスタンス全体を削除するのではなく、デフォルトの VRF の BGP 構成のみを削除しようとする、プロトコル/BGP レベルで BGP 情報が削除されないことがあります。この状況では、ペイロードに自律システム番号を含むプロトコルまたは BGP レベルで削除が行われると、BGP ルーティング インスタンス全体が削除されるのではなく、デフォルトの VRF の設定のみが削除されます。

以下は、BGP のデフォルト VRF で設定を削除するために使用されるペイロードの例です。

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <network-instances xmlns="http://openconfig.net/yang/network-instance">
        <network-instance>
          <name>default</name>
          <protocols>
            <protocol>
              <identifier>BGP</identifier>
              <name>bgp</name>
              <bgp xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
nc:operation="delete">
                <global>
                  <config>
                    <as>100</as>
                  </config>
                </global>
              </bgp>
            </protocol>
          </protocols>
        </network-instance>
      </network-instances>
    </config>
  </edit-config>
</rpc>
```

予期される動作：BGP ルーティング インスタンス自体を削除する必要があります。これは、**no router bgp 100** と同等です。

実際の動作：デフォルト VRF の BGP 構成のみが削除され、同等の単一の CLI 構成はありません。

削除操作前の実行構成は次のとおりです。

```
router bgp 100
  router-id 1.2.3.4
  address-family ipv4 unicast
  vrf abc
    address-family ipv4 unicast
      maximum-paths 2
```

削除操作後の実行構成は次のとおりです。

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

YANG の検証

YANG 設定を検証するには、次のコマンドを使用します。

表 18: YANG 検証

コマンド	説明
show telemetry yang direct-path cisco-nxos-device	サポートされているパスを表示します。



第 26 章

NETCONF エージェント

この章は次のトピックで構成されています。

- [NETCONF エージェントについて \(317 ページ\)](#)
- [NETCONF に関する注意事項と制限事項 \(318 ページ\)](#)
- [NETCONF エージェントの構成 \(320 ページ\)](#)
- [NETCONF セッションの確立 \(322 ページ\)](#)
- [NETCONF の読み取りおよび書き込み構成 \(324 ページ\)](#)
- [NETCONF 通知 \(332 ページ\)](#)
- [NETCONF の例 \(337 ページ\)](#)
- [NETCONF エージェントのトラブルシューティング \(341 ページ\)](#)

NETCONF エージェントについて

NETCONF (Network Configuration Protocol、ネットワーク構成プロトコル) は、[RFC 6241](#) によって定義されているネットワーク管理プロトコルです。Cisco NX-OS は、クライアント側のインターフェイスである NETCONF エージェントを提供しており、XML でエンコードされた YANG モデルの形式で、クライアントの要求とサーバーの応答のため、SSH 上のセキュアな転送を提供します。

NETCONF は、構成データストアと、これらのデータストアでの操作とクエリを可能にする一連の作成、読み取り、更新、および削除 (CRUD) 操作を定義しています。NX-OS では、実行、起動、候補の 3 つのデータストアがサポートされています。サポートされている操作の簡単な説明を次に示します。

表 19: サポートされる操作

動作	説明
get	実行構成と動作状態を取得します。
get-config	指定されたデータストアから構成を取得します。

動作	説明
edit-config	指定されたターゲット データストアに指定された構成をロードします
close-session	セッションの適切な終了を要求します。
kill-session	セッションを強制終了します。
copy-config	別のデータストアの内容でデータストアを作成するか、置き換えます。
ロック	データストアをロックします。
unlock	データストアのロックを解除します。
検証	指定された構成の内容を検証します。
commit	候補構成を新しい現行の実行構成としてコミットします。
cancel-commit	進行中の要確認コミットをキャンセルします。
discard-changes	候補構成を現行の実行構成に戻します。

NETCONF に関する注意事項と制限事項

NETCONF エージェントには、次の注意事項と制限事項があります：

- Cisco NX-OS は、NETCONF 通知で Cisco デバイス YANG モデルと OpenConfig モデルの両方をサポートします。
- デバイス YANG モデルはエフェメラルデータを定義します。これらは「`//Ephemeral data`」というコメントでマークされます。これらの非永続的な大容量データは、モデルの残りの部分とは異なる方法で処理されます。これらは、`<get>` クエリの `<filter>` パラメータが、コメントでマークされた特定の要素を具体的に指している場合にのみ返されます。使用方法の詳細については、エフェメラルデータサポートのドキュメントを参照してください。
- Cisco NX-OS リリース 9.3 (3) 以降、NETCONF は [RFC 6241](#) に準拠していますが、次の例外があります：
 - 兄弟格納ファイルマッチノードは、「AND」式ではなく「OR」式で論理的に結合されます。（セクション 6.2.5）
 - 候補データストアを編集した後は、同じプロパティの実行構成を編集しないでください。
- 1 つの Get 要求でサポートされるオブジェクトの数は 250,000 です。次のエラーが表示された場合は、要求されたデータが 250,000 を超えていることを意味します。このエラーを

回避するには、より狭い範囲のデータをクエリするフィルタを使用してリクエストを送信します。

too many objects(459134 > 250000) to query the entire device model.

- NETCONF は、RFC 6536 で指定されている拡張ロールベース アクセス コントロール (RBAC) をサポートしていません。「network-admin」ロールを持つユーザーのみが NETCONF エージェントへのアクセスを許可されます。
- NX-OS 9.3(1) 以降、NETCONF クライアントからスイッチへの NETCONF `get` および `get-config` 要求には、明示的な名前空間とフィルタが含まれている必要があります。この要件は、OpenConfig YANG および NETCONF デバイス モデルへの要求に影響します。次のようなメッセージが表示された場合、要求は名前空間を伝送していません。

Request without namespace and filter is an unsupported operation

次の例は、この変更前の動作を使用した `get` 要求と応答を示しています。この例は、サポートされなくなった動作が原因で発生するエラーメッセージを示しています。

要求:

```
<get>
</get>
```

応答:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>operation-not-supported</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">Request without filtering is an unsupported
operation</error-message>
  </rpc-error>
</rpc-reply>
```

次に、NX-OS リリース 9.3(1) 以降での正しい動作の `get` 要求と応答の例を示します。

要求:

```
<get>
  <filter>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    </System>
  </filter>
</get>
```

応答:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <System> ...
  </data>
</rpc-reply>
```

- `<edit-config>` の「置換」操作は、影響を受けるシステム コンポーネントによって実装されている実行時デフォルト値と動作が原因で、機能しない場合があります。したがって、NX-API 開発者サンドボックスの代わりに、`<get-config>` クエリによって取得した構成上で、置換を行うための構成を基礎とする方が適切です。
- Cisco NX-OS NETCONF サーバーは、最大 5 つのサブスクリプション（クライアントセッションごとに 1 つのサブスクリプション）をサポートします。
- [RFC 5277](#) に従って、自律通知は、イベント送信元の NETCONF、SYSLOG、および SNMP ストリームをサポートします。このリリースでは、Cisco NX-OS は NETCONF ストリームのみをサポートします。
- Cisco NX-OS は、サブスクリプションの [再生 (Replay)] オプションをサポートしていません。[開始時刻 (Start Time)] オプションと [終了時刻 (Stop Time)] オプションは再生の一部であるため、サポートされていません。
- ストリーム サブスクリプションとフィルタリングでは、サブツリー フィルタリングのみがサポートされます。XPath フィルタリングはサポートされていません。
- Cisco NX-OS NETCONF エージェントが高負荷で動作している場合、一部のイベント通知がドロップされる可能性があります。
- Cisco NX-OS は、Cisco NX-OS リリース 9.3(1) 以降で NETCONF 通知をサポートします。Cisco NX-OS は、Cisco デバイス YANG モデルのみをサポートします。
- NATがすでに構成されている場合は、スイッチに NETCONF を構成できません。NETCONF と NAT の構成には互換性がなく、同じスイッチ上で共存できません。
- Cisco NX-OS は、Cisco Device YANG モデルと OpenConfig モデルの両方をサポートします。NETCONF 通知で OpenConfig モデルをサポートするのは、Cisco NX-OS 9.3(5) リリース以降です。

NETCONF エージェントの構成

Cisco NX-OS 9.3(5) 以降の SSH を介した NETCONF エージェントの設定

この手順では、SSH を介して NETCONF エージェントを有効にして構成する方法について説明します。



(注) この手順は、Cisco NX-OS リリース 9.3(5) 以降で使います。

始める前に

NETCONF を使用してスイッチと通信する前に、NETCONF エージェントを有効にする必要があります。NETCONF エージェントを有効または無効にするには、**[no] feature netconf** コマンドを入力します。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : switch# configure terminal	グローバル コンフィギュレーション モードを開始します。
ステップ 2	feature netconf 例 : switch(config)# feature netconf	NETCONF サービスを有効にします。
ステップ 3	(任意) netconf idle-timeout it-num 例 : switch(config)# netconf idle-timeout 5	(オプション) アイドル状態のクライアント セッションが切断されるまでのタイムアウトを分単位で指定します。 <i>it-num</i> の範囲は 0 ～ 1440 分です。デフォルトのタイムアウトは 5 分です。値を 0 に設定するとタイムアウトが無効になります。
ステップ 4	(任意) netconf sessions num-sessions 例 : switch(config)# netconf sessions 5	同時クライアント セッションの最大数を指定します。 <i>num-sessions</i> の範囲は 1 ～ 10 です。デフォルト値は 5 セッションです。

Cisco NX-OS 9.3(4) 以前の NETCONF エージェントの構成



(注) Cisco NX-OS リリース 9.3(4) 以前の場合は、次の手順に従ってください。

NETCONF エージェントは、構成ファイル (/etc/mtx.conf) の [netconf] セクションで、次のオプションの構成パラメータをサポートします。

パラメータ	説明
idle_timeout	(オプション) アイドル状態のクライアントセッションが切断されるまでのタイムアウトを分単位で指定します。 デフォルト値は 5 分です。 値を 0 に設定するとタイムアウトが無効になります。
limit	(オプション) 同時クライアントセッションの最大数を指定します。 デフォルト値は 5 セッションです。 指定できる範囲は、1 ～ 10 です。

次に、構成ファイルの [netconf] セクションの例を示します。

```
[netconf]
mtxadapter=/opt/mtx/lib/libmtxadapternetconf.1.0.1.so
idle_timeout=10
limit=1
```

変更した構成ファイルを有効にするには、CLI コマンド **[no] feature netconf** を使用して NETCONF エージェントを無効にしてから再度有効にして、再起動する必要があります。

NETCONF セッションの確立

NETCONF は、クライアントとサーバー間の永続的な接続を必要とする接続指向のプロトコルです。スイッチ上の NETCONF エージェントは、管理ポート IP アドレスのポート 830 でリスンします。クライアントは、SSH を介して NETCONF サブシステムとの接続を確立できます。クライアントが NETCONF エージェントとのセッションを確立すると、サーバーは<hello>メッセージをクライアントに送信します。同様に、クライアントは<hello>メッセージをサーバーに送信します。は、<hello>メッセージは、接続が開くとすぐに同時に交換されます。各<hello>メッセージには、送信側ピアのプロトコルバージョンと機能のリストが含まれています。これらのメッセージは、プロトコルの互換性と機能を判断するために使用されます。両方の NETCONF ピアは、共通のプロトコルバージョンが他のピアの<hello>メッセージを表示します。また、サーバーの<hello>メッセージには<session-id>一方、クライアントのメッセージを送信することはできません。

次に、ssh コマンドを使用したセッション確立の例を示します。最初の<hello>メッセージがサーバーから受信され、2 番目のメッセージがクライアントから送信されます。サーバーの<hello>メッセージに、プロトコルバージョン「urn:ietf:params:netconf:base:1.1」と、Cisco NX-OS リリース 9.3(4) でサポートされている NETCONF 基本機能が表示されます。また、サーバーの<hello>メッセージには、サポートされているデータモデルが含まれています。現在の Cisco NX-OS リリースでサポートされているモデルと一致しない場合があります。



- (注) サーバーの<hello>メッセージには、<session-id>ですが、クライアントのメッセージはそうではありません。

```
client-host % ssh admin@172.19.193.166 -p 830 -s netconf
User Access Verification
Password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>

    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>

    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>

    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>

    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=report-all</capability>

    <capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2020-04-20&module=Cisco-NX-OS-device</capability>

    <capability>http://openconfig.net/yang/acl?revision=2019-11-27&module=openconfig-acl&deviations=cisco-nx-openconfig-acl-deviations</capability>

    <capability>http://openconfig.net/yang/bfd?revision=2019-10-25&module=openconfig-bfd&deviations=cisco-nx-openconfig-bfd-deviations</capability>

    ...
  </capabilities>
  <session-id>1286775422</session-id>
</hello>
]]>]]><hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
]]>]]>
```

コマンドで NETCONF を使用することは便利ではなく、エラーが発生しやすくなります。**ssh** 上記の例では、説明のみを目的としてコマンドが使用されています。**ssh** コマンドよりも推奨される NETCONF 用に作成されたさまざまなクライアントがあります。**ssh ncclient** はそのような例の 1 つであり、「使用例」セクションで使用されています。

NETCONF は、セッションを終了するための 2 つの操作をサポートしています。<close-session> および <kill-session>。サーバーが <close-session> 要求があった場合、セッションに関連付けられているロックとリソースを解放し、クライアントとの接続を閉じることで、セッションを正常に終了します。次に、<close-session> 成功の要求と応答：

```
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>
```

```
<rpc-reply message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

は、**<kill-session>**要求は別のセッションの終了を強制し、**<session-id>**要求メッセージで。を受信すると、**<kill-session>**要求がある場合、サーバーは現在の操作を終了し、ロックとリソースを解放し、指定されたセッション ID に関連付けられている接続を閉じます。次に、**<kill-session>**成功の要求と応答：

```
<rpc message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>296324181</session-id>
  </kill-session>
</rpc>

<rpc-reply message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

に加えて、**<close-session>**および**<kill-session>**クライアントが一定時間要求を送信しない場合、セッションは自動的に終了します。デフォルトは5分です。アイドルタイムアウトの設定については、「NETCONF エージェントの設定」を参照してください。

NETCONF の読み取りおよび書き込み構成

このセクションでは、データストアを操作およびクエリするためにサポートされている基本プロトコル操作について説明します。クライアントは、NETCONF エージェントとのセッションを確立した後、これらの操作の RPC メッセージを送信できます。これらの操作の詳細については、基本的な使用方法の説明が記載されており、RFC 6242 を参照できます。

<get-config>

この操作により、指定したデータストアの構成データを取得します。サポートされるパラメータは次のとおりです。**<filter>**。**<source>**は、クエリ対象のデータストアを指定します。**<running/>**。現在アクティブな設定を保持します。**<source>**は、**<filter>**取得する指定されたデータストアの部分指定します。

次に例を示します。要求および応答メッセージ。

- 全体を取得します。**<System>**サブツリー：

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
    </filter>
  </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
```

```

    <data>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        ...
      </System>
    </data>
  </rpc-reply>

```

- 特定のリスト項目を取得します。

```

<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bgp-items>
          <inst-items>
            <dom-items>
              <Dom-list>
                <name>default</name>
              </Dom-list>
            </dom-items>
          </inst-items>
        </bgp-items>
      </System>
    </filter>
  </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <bgp-items>
        <inst-items>
          <dom-items>
            <Dom-list>
              <name>default</name>
              ...
              <rtctrl-items>
                <enforceFirstAs>enabled</enforceFirstAs>
                <fibAccelerate>disabled</fibAccelerate>
                <logNeighborChanges>enabled</logNeighborChanges>
                <supprRt>enabled</supprRt>
              </rtctrl-items>
              <rtrId>1.2.3.4</rtrId>
            </Dom-list>
          </dom-items>
        </inst-items>
      </bgp-items>
    </System>
  </data>
</rpc-reply>

```

<edit-config>

この操作は、指定された設定をターゲットデータストアに書き込みます。は、<target>パラメータは、編集するデータストアを指定します。<running>または<candidate/>。候補データストアは、変更がコミットされるまで、実行中のデータストアに影響を与えることなく操作できます。詳細については、を参照してください。<commit>セクションを参照してください。は、

<config>パラメータは、ターゲットデータストアに書き込まれるモデル化されたデータを指定します。モデルは「xmlns」属性で指定されます。任意の数の要素<config>サブツリーには「operation」属性を含めることができます。要素の操作は、新しい「操作」属性によってオーバーライドされるまで、その子孫要素に継承されます。サポートされている操作は、「merge」、「replace」、「create」、「delete」、および「remove」です。「削除」操作は、設定データが存在しない場合にエラーが返されないという点で「削除」とは異なります。「operation」属性が指定されていない場合は、マージ操作がデフォルトと見なされます。デフォルトの動作は、オプションの<default-operation>パラメータには、「merge」、「replace」、または「none」があります。

次に例を示します。<edit-config>要求および応答メッセージ。

- MTU 9216 と実行コンフィギュレーションの説明を使用して、「po5」という名前のポートチャネルを作成します。

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <aggr-items>
            <AggrIf-list xc:operation="create">
              <id>po5</id>
              <mtu>9216</mtu>
              <descr>port-channel 5</descr>
            </AggrIf-list>
          </aggr-items>
        </intf-items>
      </System>
    </config>
  </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
  <ok/>
</rpc-reply>
```

- ポートチャネルのすべての設定を新しい設定に置き換えます。

```
<rpc message-id="104" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <aggr-items>
            <AggrIf-list xc:operation="replace">
              <id>po5</id>
              <mtu>1500</mtu>
              <adminSt>down</adminSt>
            </AggrIf-list>
          </aggr-items>
        </intf-items>
      </System>
    </config>
```

```

    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="104">
  <ok/>
</rpc-reply>

```

- ポートチャネルを削除します：

```

<rpc message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <aggr-items>
            <AggrIf-list xc:operation="delete">
              <id>po5</id>
            </AggrIf-list>
          </aggr-items>
        </intf-items>
      </System>
    </config>
  </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="105">
  <ok/>
</rpc-reply>

```

<copy-config>

この操作は、ターゲットの構成データストアを、ソース構成データストア全体のコンテンツによって置き換えます。ソースデータストアとターゲットデータストアのパラメータは次のとおりです。<target>に設定します。<source>

次に例を示します。<copy-config>要求および応答メッセージ。

- 実行構成をスタートアップ構成にコピーします。

```

<rpc message-id="106" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <target>
      <startup/>
    </target>
    <source>
      <running/>
    </source>
  </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="106">
  <ok/>
</rpc-reply>

```

- 実行構成を候補構成にコピーします。

```

<rpc message-id="107" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>

```

```

        <target>
          <candidate/>
        </target>
        <source>
          <running/>
        </source>
      </copy-config>
    </rpc>

    <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="107">
      <ok/>
    </rpc-reply>

```

<lock>

は、<lock>操作を使用すると、クライアントは設定データストアをロックし、他のクライアントがデータストアをロックまたは変更するのを防ぐことができます。クライアントが保持しているロックは、<unlock>セッションの操作または終了。は、<target>パラメータを使用して、ロックするデータストアを指定します。

次に例を示します。<lock>要求および応答メッセージ。

- ロックの取得に成功した場合：

```

    <rpc message-id="108" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <lock>
        <target>
          <running/>
        </target>
      </lock>
    </rpc>

    <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="108">
      <ok/>
    </rpc-reply>

```

- 別のセッションですでに使用されているロックの取得に失敗しました。

```

    <rpc message-id="109" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <lock>
        <target>
          <candidate/>
        </target>
      </lock>
    </rpc>

    <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="109">
      <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>lock-denied</error-tag>
        <error-severity>error</error-severity>
        <error-message xml:lang="en">Lock failed, lock is already held</error-message>

        <error-info>
          <session-id>1553704357</session-id>
        </error-info>
      </rpc-error>
    </rpc-reply>

```


<unlock>

<unlock> 操作は、以前に <lock> 操作によって取得した構成のロックを解除します。を発行したのと同じセッションのみ<lock>操作では、<unlock>操作を実行します。は、<target>パラメータは、ロック解除するデータストアを指定するために使用されます。

次に例を示します。<unlock>要求および応答メッセージ。

- ロック解除

```
<rpc message-id="110" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target>
      <candidate/>
    </target>
  </unlock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="110">
  <ok/>
</rpc-reply>
```

<get>

<get> 操作は、実行中の構成とデバイスの状態情報を取得します。サポートされているパラメータは <filter> です。<filter> パラメータは、実行構成の動作状態データのうち、どの部分を取得するかを指定します。

次に例を示します。要求および応答メッセージ。

- リスト項目の実行コンフィギュレーションおよび動作状態データを取得します。

```
<rpc message-id="111" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bgp-items>
          <inst-items>
            <dom-items>
              <Dom-list>
                <name>default</name>
              </Dom-list>
            </dom-items>
          </inst-items>
        </bgp-items>
      </System>
    </filter>
  </get>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="111">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <bgp-items>
        <inst-items>
          <dom-items>
            <Dom-list>
              <name>default</name>
              <always>disabled</always>
              <bestPathIntvl>300</bestPathIntvl>
            </Dom-list>
          </dom-items>
        </inst-items>
      </bgp-items>
    </System>
  </data>
</rpc-reply>
```

```

<clusterId>120</clusterId>

<firstPeerUpTs>2020-04-20T16:19:03.784+00:00</firstPeerUpTs>
  <holdIntvl>180</holdIntvl>
  <id>1</id>
  <kaIntvl>60</kaIntvl>
  <mode>fabric</mode>
  <numEstPeers>0</numEstPeers>
  <numPeers>0</numPeers>
  <numPeersPending>0</numPeersPending>
  <operRtrId>1.2.3.4</operRtrId>
  <operSt>up</operSt>
  <pfxPeerTimeout>90</pfxPeerTimeout>
  <pfxPeerWaitTime>90</pfxPeerWaitTime>
  <reConnIntvl>60</reConnIntvl>
  <rtrId>1.2.3.4</rtrId>
  <vnid>0</vnid>
  ...
</Dom-list>
</dom-items>
</inst-items>
</bgp-items>
</System>
</data>
</rpc-reply>

```

<validate>

この操作では、候補データストアの設定内容を検証します。これは、実行中のデータストアにコミットする前に、候補データストアで行われた設定変更を検証するのに役立ちます。パラメータは、<candidate/>。<source>

次に例を示します。<validate>要求および応答メッセージ。

- 候補データストアの内容を検証します：

```

<rpc message-id="112" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="112">
  <ok/>
</rpc-reply>

```

<commit>

候補構成を実行構成にコミットします。パラメータのない操作は最終と見なされ、元に戻すことはできません。次の場合<commit>が発行されます。<confirmed/>パラメータを使用すると、確定されたコミットと見なされ、別のを使用しない操作パラメータを設定します。つまり、コミットの確認です。確認済みコミットでは、次の2つのパラメータを使用できます。

<confirm-timeout>および<persist>。は、<confirm-timeout>は、確認されたコミットが元に戻されるまでの秒数です。これにより、確認コミットが発行される前の状態に実行コンフィギュレー

セッションが復元されます。次の場合、`<confirm-timeout>`が指定されていない場合、デフォルトのタイムアウトは600秒です。また、セッションが終了すると、確認されたコミットは元に戻ります。は、`<persist>`パラメータを使用すると、セッションが終了しても確認されたコミットが保持されます。の値は、`<persist>`パラメータは、任意のセッションから確認されたコミットを識別するために使用され、`<persist-id>`後続の確認済みコミットまたは確認コミットのパラメータ。

次に例を示します。`<commit>`要求および応答メッセージ。

- 候補データストアの内容をコミットします。

```
<rpc message-id="113" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="113">
  <ok/>
</rpc-reply>
```

- タイムアウトで確定されたコミット：

```
<rpc message-id="114" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
    <confirmed/>
    <confirm-timeout>120</confirm-timeout>
  </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="114">
  <ok/>
</rpc-reply>
```

- 永続的な確認済みコミットを開始し、永続的な確認済みコミットを確認します。

```
<rpc message-id="115" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
    <confirmed/>
    <persist>ID1234</persist>
  </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="115">
  <ok/>
</rpc-reply>

<!-- confirm the persistent confirmed-commit, from the same session or another session -->
<rpc message-id="116" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
    <persist-id>ID1234</persist-id>
  </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="116">
  <ok/>
</rpc-reply>
```

<cancel-commit>

この操作は、進行中の確認済みコミットをキャンセルします。別のセッションからの確認済みコミットをキャンセルする必要がある場合、<persist-id>パラメータは、<persist>パラメータを設定します。

- 同じセッションから確認されたコミットをキャンセルします。

```
<rpc message-id="117" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <cancel-commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="117">
  <ok/>
</rpc-reply>
```

<discard-changes>

この操作では、実行コンフィギュレーションの内容にリセットすることによって、候補コンフィギュレーションで行われたコミットされていない変更が破棄されます。パラメータはありません。

次に例を示します。<discard-changes>要求および応答メッセージ。

- 候補データストアで行われた変更を破棄します。

```
<rpc message-id="118" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <discard-changes/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="118">
  <ok/>
</rpc-reply>
```

NETCONF 通知

NETCONF 通知について

NETCONF 通知は、NETCONF クライアントがシステムイベントに登録し、NETCONF エージェントからこれらのイベントに関する通知を受信できるメカニズムです。これらの機能は、RFC 5277 で定義されています。<http://tools.ietf.org/html/rfc5277> Cisco NX-OS リリース 9.3(1) 以降、RFC 5277 で説明されているように、NETCONF 通知のサポートが開始されました。<http://tools.ietf.org/html/rfc5277> これは、NETCONF hello メッセージでアドバタイズされるオプションの機能です。

NETCONF クライアントは、Deviceyang または OpenConfig モデルを使用して通知をサブスクライブできます。NETCONF 通知での OpenConfig モデルのサポートは、Cisco NX-OS リリース 9.3(5) から開始されます。

このサポートにより、NETCONF クライアントは次のことができます。

- イベント通知へのサブスクリプション

各サブスクリプションは、NETCONF クライアントからのセッションを介した 1 回限りの要求です。Cisco NX-OS NETCONF エージェントが応答し、NETCONF クライアントによってセッションが明示的に閉じられるまでサブスクリプションはアクティブです。サブスクリプションは、スイッチの再起動やスイッチの NETCONF 機能の無効化などの管理アクションによって閉じることもできます。サブスクリプションは、基盤となる NETCONF セッションがアクティブである限りアクティブです。これらの登録済みフィルタに対して生成されたイベントは、通知としてクライアントに送信されます。クライアントは、システムイベントの通知に登録できます。たとえば、ポート状態の変更、ファン速度の変更、プロセスメモリの変更などです。また、有効になっている新機能などの設定イベント。

- 障害イベント通知を受信します。

イベント通知は、スイッチの設定イベントまたは動作イベントに関する情報を含む、整形式の XML ドキュメントです。NETCONF クライアントは、サブスクリプション要求でフィルタリング基準を送信して、すべてのイベントではなくイベントのサブセットを指定できます。

- 他の操作でイベント通知をインターリーブします。

Cisco NX-OS NETCONF エージェントは、アクティブな通知サブスクリプションを持つセッションで NETCONF 要求を受信し、処理し、応答できます。

機能交換

NETCONF ハンドシェイク中に、Cisco NX-OS NETCONF サーバーは<capabilities> 要素を接続している NETCONF クライアントに送信して、サーバーが処理できる要求を示します。交換の一部として、サーバーは次の識別子を含めます。これらの識別子は、Cisco NX-OS NETCONF サーバーが通知とインターリーブの両方をサポートしていることをクライアントに通知します。

通知の機能識別子：

```
urn:ietf:params:netconf:capability:notification:1.0
```

インターリーブの機能識別子：

```
urn:ietf:params:netconf:capability:interleave:1.0
```

イベント ストリームの検出

クライアントは、NETCONF を使用して、Cisco NX-OS NETCONF サーバーのサポートされているストリームを検出できます。使用可能なすべての<streams>。Cisco NX-OS は NETCONF ストリームのみをサポートします。イベント ストリームの検出は、要求と応答のシーケンスによって行われます。

使用可能なストリームを取得する要求：

すべてのNETCONFクライアントがNETCONFを送信できます。次のフィルタを使用した要求<streams>サポートされているすべてのストリームを識別します。次の例は、クライアント要求メッセージのペイロードを示しています。

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
        <streams/>
      </netconf>
    </filter>
  </get>
</rpc>
```

応答：

Cisco NX-OS NETCONF サーバーは、クライアントがサブスクライブできる使用可能なすべてのイベントストリームで応答します。Cisco NX-OS は NETCONF ストリームのみをサポートします。

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
      <streams>
        <stream>
          <name>NETCONF</name>
          <description>default NETCONF event stream </description>
        </stream>
      </streams>
    </netconf>
  </data>
</rpc-reply>
```

サブスクリプションの作成

NETCONF クライアントは、<create-subscription>プロトコル動作。Cisco NX-OS NETCONF サーバーが<ok/>要素の場合、サブスクリプションはアクティブです。

同期の Get および Set 操作とは異なり、サブスクリプションは永続的な非同期操作です。サブスクリプションは、クライアントが明示的にサブスクリプションを閉じるか、セッションがオフラインになるまでアクティブなままです。たとえば、スイッチの再起動によって。

クライアントがイベント通知をサブスクライブしていたものの、オフラインになった場合には、サーバーはサブスクリプションを終了し、セッションを閉じます。

サブスクリプションが閉じられている場合、すべてのイベント通知を受信するには、NETCONF クライアントが再接続してサブスクリプションを再度作成する必要があります。

サーバーはサブスクリプションを開始しないため、<create-subscription>操作を実行します。次に、<create-subscription> NETCONF クライアントによって送信されます。

```
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <stream>NETCONF</stream>
  <filter xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0" type="subtree">
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <intf-items>
        <phys-items>
          <PhysIf-list>
            <id>eth1/54/1</id>
            <phys-items>
              <operSt/>
            </phys-items>
          </PhysIf-list>
        </phys-items>
      </intf-items>
    </System>
  </filter>
</create-subscription>
```

は、`<create-subscription>`操作は、次のオプションのいずれかをサポートします。

- `<stream>` クライアントがサブスクライブするイベントのストリームを指定します。ストリームを指定しなかった場合、NETCONF ストリーム内のイベントがデフォルトでクライアントに送信されます。
- `<filter>` これにより、イベントをフィルタリングして、ストリームで伝送されるイベントのサブセットを提供できます。

Cisco NX-OS NETCONF サーバーは、`<ok>`サーバーがサブスクリプションを正常に作成できる場合は、メッセージが表示されます。

次に、クライアントで受信した成功応答の例を示します。`<create-subscription>`サーバーに送信するように要求します。

の応答`<create-subscription>`がクライアントで受信されます。

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:6ff0bda6-d3f1-4288-9a7e-0f30581e4bab">
  <ok/>
</rpc-reply>
```



- (注) リプレイを使用したサブスクリプションはサポートされていないため、[開始時間 (StartTime)] および [終了時間 (Stop Time)] オプションは使用できません。

受信通知

NETCONF クライアントがサブスクリプションを正常に作成すると、Cisco NX-OS NETCONF サーバは、スイッチ内のすべてのイベントについて、使用されたフィルタに関連するイベント通知の送信を開始します。イベント通知は、notification 要素を含む独自の XML フォーマットのドキュメントです。

次に、クライアントが **DeviceYang** モデルからインターフェイス **operSt** にサブスクライブして、イーサネットインターフェイスがダウンした場合の通知の例を示します。

<create-subscription> は、[サブスクリプションの作成 (Creating Subscriptions)] セクションにあります。

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2020-05-05T10:22:52.260+00:00</eventTime>
  <operation>modified</operation>
  <event>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <intf-items>
        <phys-items>
          <PhysIf-list>
            <id>eth1/54/1</id>
            <phys-items>
              <operSt>down</operSt>
            </phys-items>
          </PhysIf-list>
        </phys-items>
      </intf-items>
    </System>
  </event>
</notification>
```

<notification> メッセージには次のフィールドが含まれます。

- **<eventTime>** はイベントが発生した日時を示すタイムスタンプです。
- **<operation>** はモデル ノードのイベントのタイプです。
- **<event>** はクライアントがサブスクライブしているモデル データです。

サブスクリプションの終了

サブスクリプションは、NETCONF クライアントが NETCONF メッセージのペイロードで **Cisco NX-OS NETCONF** サーバーに特定の操作を送信すると終了します。サブスクリプションの終了は、次のいずれかの方法で発生します。

- サブスクリプションセッションの終了。**<close-session>**操作は、特定のサブスクリプションセッションの NETCONF サーバーに送信されます。
- NETCONF セッションの終了。**<kill-session>**操作が NETCONF サーバーに送信されます。

すべてのサブスクリプションは、1 つの NETCONF セッションに関連付けられます。これは 1 対 1 の関係です。

NETCONF の例



(注) このセクションのすべての例では、ncclient python ライブラリを使用します。

ncclient を使用した Cisco NX-OS の接続

ncclient は、NETCONF クライアント用の Python ライブラリです。次に、ncclient Manager API から Cisco NX-OS への接続を確立する方法の例を示します。

```
device = {
    "address": "10.10.10.10",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}
with manager.connect(host = device["address"],
                    port = device["netconf_port"],
                    username = device["username"],
                    password = device["password"],
                    hostkey_verify = False) as m:

    # do your stuff
```

サンドボックスを使用した NETCONF ペイロードの生成

有効にするには、「NXAPI 開発者サンドボックス」セクションを参照してください。NETCONF のペイロードを生成するには、メソッドを RESTCONF (Yang) に変更し、メッセージ形式を XML に変更します。テキストウィンドウに変換する必要があるコマンドを入力し、[変換 (Convert)] をクリックすると、同等のペイロードが[要求 (Request)] テキストボックスに表示されます。

図 3: NCCLIENT

The screenshot shows the ncclient web interface. At the top, a text area contains the following configuration:

```
interface ethernet 1/4
  switchport
  switchport mode trunk
  switchport trunk allowed vlan 100
```

On the right, the 'Method' is set to 'RESTCONF (Yang)' and 'Message format' is set to 'xml'. Below this, a 'POST' dropdown is followed by a URL input field containing 'restconf/data/Cisco-NX-OS-device:System/'. To the right of the input are 'Send', 'Reset', and 'Convert' buttons.

Below the input field, there are tabs for 'Request', 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab is selected, showing the following XML response:

```
<intf-items>
  <phys-items>
    <PhysIf-list>
      <id>eth1/4</id>
      <layer>Layer2</layer>
      <mode>trunk</mode>
      <trunkVlans>100</trunkVlans>
      <userCfgFlags>admin_layer,admin_state</userCfgFlags>
    </PhysIf-list>
  </phys-items>
</intf-items>
```

A 'Copy' button is located to the right of the XML response.

Cisco NX-OS からの設定データの取得

次に、ncclient を使用して Cisco NX-OS から BGP 設定を取得する方法の例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco!"
}

# create a main() method
def main():
    bgp_dom = """
    <filter type="subtree">
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bgp-items>
          <inst-items>
            <dom-items>
              <Dom-list/>
            </dom-items>
          </inst-items>
        </bgp-items>
      </System>
    </filter>
    """
```

```

with manager.connect(host=device["address"],
                    port=device["netconf_port"],
                    username=device["username"],
                    password=device["password"],
                    hostkey_verify=False) as m:

    # Collect the NETCONF response
    netconf_response = m.get_config(source='running', filter=bgp_dom)
    # Parse the XML and print the data
    xml_data = netconf_response.data_ele
    print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))

if __name__ == '__main__':
    sys.exit(main())

```

Cisco NX-OS からの実行コンフィギュレーションおよび運用データの取得

次に、Cisco NX-OS 上のすべての物理インターフェイスのインターフェイスカウンタを取得する例を示します。

```

from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}

def main():

    intf_ctr_filter = """
    <filter>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <phys-items>
            <PhysIf-list>
              <dbgIfIn-items/>
              <dbgIfOut-items/>
            </PhysIf-list>
          </phys-items>
        </intf-items>
      </System>
    </filter>"""

    with manager.connect(host=device["address"],
                        port=device["netconf_port"],
                        username=device["username"],
                        password=device["password"],
                        hostkey_verify=False) as m:

        # Collect the NETCONF response
        netconf_response = m.get(filter=intf_ctr_filter)
        # Parse the XML and print the data
        xml_data = netconf_response.data_ele
        print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))

```

```
if __name__ == '__main__':
    sys.exit(main())
```

NETCONF を使用した新しい設定の作成

次に、ncclient の edit config を使用して、名前付きの VLAN 100 を作成する方法の例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}

def main():
    add_vlan = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bd-items>
          <bd-items>
            <BD-list>
              <fabEncap>vlan-100</fabEncap>
              <name>inb_mgmt</name>
            </BD-list>
          </bd-items>
        </bd-items>
      </System>
    </config>
    """

    with manager.connect(host=device["address"],
                        port=device["netconf_port"],
                        username=device["username"],
                        password=device["password"],
                        hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=add_vlan)
        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF を使用した設定の削除

次に、Cisco NX-OS からループバック インターフェイスを削除する例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
```

```

    "username": "admin",
    "password": "cisco"
}

def main():
    remove_loopback = ""
    <config>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <lb-items>
                    <LbRtdIf-list operation="delete">
                        <id>lo10</id>
                    </LbRtdIf-list>
                </lb-items>
            </intf-items>
        </System>
    </config>"""

    with manager.connect(host=device["address"],
                        port=device["netconf_port"],
                        username=device["username"],
                        password=device["password"],
                        hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=remove_loopback)

        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())

```

NETCONF エージェントのトラブルシューティング

接続のトラブルシューティング

- クライアント システムから、スイッチの管理ポートに **ping** を実行して、スイッチが到達可能であることを確認します。
- Cisco NX-OS で、**show feature | inc netconf** コマンドを入力してエージェントのステータスを確認します。
- XML 管理インターフェイス (xmlagent と呼ばれる) というものがあります。よく混同されますが、NETCONF エージェントとはまったく異なります。サーバーが正しい NETCONF メッセージで応答しない場合は、正しいポート 830 に接続していて、サーバーから正しい <hello> メッセージ (「NETCONF セッションの確立」セクションに示されているものと同様) を受信していることを確認します。
- `/volatile/netconf-internal-log` ファイルの内容を表示することで、Bash シェルからの NETCONF エージェントのデバッグ情報を表示できます。Bash シェルを有効にするには、**feature bash** コマンドを使用します。Bash シェルを有効にした後、**run bash** コマンドを使用して Bash

シェルを開始します。詳細については、このドキュメントの *Bash* の章を参照してください。

注： `debug netconf` コマンドは、NETCONF エージェント動作のデバッグには使用できません。これらのデバッグコマンドは、NETCONF エージェント関連のログを出力しません。



第 27 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](#) (343 ページ)
- [XMLIN のライセンス要件](#) (343 ページ)
- [XMLIN ツールのインストールおよび使用](#) (344 ページ)
- [show コマンド出力の XML への変換](#) (345 ページ)
- [XMLIN の構成例](#) (345 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 20: 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、『 <i>Cisco NX-OS Licensing Guide</i> 』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

	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。
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

	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:ml="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>
              <ml:cdp>
                <ml:enable/>
              </ml: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>

```

```

        </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 28 章

RESTCONF エージェント

- [RESTCONF エージェントについて \(349 ページ\)](#)
- [注意事項と制約事項 \(350 ページ\)](#)
- [RESTCONF エージェントの使用方法 \(350 ページ\)](#)
- [RESTCONF エージェントのトラブルシューティング \(352 ページ\)](#)
- [エフェメラル データ \(353 ページ\)](#)

RESTCONF エージェントについて

Cisco NX-OS RESTCONF は、NETCONF で定義されたデータストアを使用して、YANG バージョン 1 で定義されたデータを設定するための HTTP ベースのプロトコルです。

NETCONF は、設定データストアと、これらのデータストアへのアクセスに使用できる一連の作成、取得、更新、および削除 (CRUD) 操作を定義します。YANG 言語は、データストア格納ファイル、運用データ、プロトコル操作、イベント通知の構文とセマンティクスを定義します。

Cisco NX-OS RESTCONF は、HTTP 操作を使用して、YANG 定義のデータを含む概念データストアで CRUD 操作を提供します。このデータは、NETCONF データストアを実装するサーバーと互換性があります。

RESTCONF プロトコルは、XML と JSON の両方のペイロードエンコーディングをサポートします。ユーザー認証は、HTTP 基本認証によって行われます。

次の表に、Cisco NX-OS RESTCONF エージェントがサポートするプロトコル操作を示します。

RESTCONF	同等の NETCONF
オプション	NETCONF : なし
HEAD	NETCONF : なし
GET	NETCONF: <get-config>, <get>
POST	NETCONF: <edit-config> (operation="create")
PUT	NETCONF: <edit-config> (operation="create/replace")

RESTCONF	同等の NETCONF
PATCH	NETCONF: <edit-config> (operation="merge")
DELETE	NETCONF: <edit-config> (operation="delete")

注意事項と制約事項

RESTCONF エージェントには、次の注意事項と制約事項があります。

- Cisco NX-OS RESTCONF は、RESTCONF Protocol draft-ietf-netconf-restconf-10 というタイトルの RFC ドラフトに基づいています。<https://tools.ietf.org/html/draft-ietf-netconf-restconf-10> を参照してください。
- RESTCONF は、RFC 6536 で指定されている拡張ロールベース アクセス コントロール (RBAC) をサポートしていません。「network-admin」ロールを持つユーザのみに、RESTCONF エージェントへのアクセス権が付与されます。

RESTCONF エージェントの使用方法

一般的なコマンド

- HTTP または HTTPS アクセスを有効にするには、次のコマンドを設定します。
 - **feature nxapi**
 - **nxapi http port 80**
 - **nxapi https port 443**

一般的な制御コマンド

RESTCONF エージェントの **[no] feature restconf** コマンドを有効または無効にできます。

RESTCONF エージェントで使用可能な制御コマンドは次のとおりです。

restconfctl { status | start | restart | reload | stop }

エージェントステータスの表示

RESTCONF エージェントのステータスを表示するには、コマンドを使用し、式 **restconf** を含めます。 **show feature**

```
switch-1# show feature | grep restconf
restconf          1          enabled
switch-1#
```

```
bash-4.2# restconfctl status
```

```
xosdsd is stopped
restconfctl is stopped
```

エージェントの起動

```
bash-4.2# restconfctl start
Starting Restconf Agent: [OK]
```

Curl を使用したサーバーへの POST 要求の送信

```
client-host % curl -X POST -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Content-Type:
application/yang.data+xml" -d '<always>enabled</always><rtrId>2.2.2.2</rtrId>'
'http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list=default'
-i

HTTP/1.1 201 Created
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:25:31 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500853169574134
Status: 201 Created
Location: /System/bgp-items/inst-items/dom-items/Dom-list=default/always/rtrId/
```

Curl を使用したサーバーへの GET 要求の送信

```
client-host % curl -X GET -H "Authorization: Basic YWRtaW46Y2lzY28=" -H "Accept:
application/yang.data+xml"
'http://192.0.20.123/restconf/data/Cisco-NX-OS-device:System/bgp-items/inst-items/dom-items/Dom-list?content=config'
-i

HTTP/1.1 200 OK
Server: nginx/1.7.10
Date: Tue, 27 Sep 2016 20:26:03 GMT
Content-Type: application/yang.data+xml
Content-Length: 395
Connection: keep-alive
Set-Cookie: nxapi_auth=admin:147500856185650327
Status: 200 OK

<Dom-list>
  <name>default</name>
  <always>enabled</always>
  <bestPathIntvl>300</bestPathIntvl>
  <holdIntvl>180</holdIntvl>
  <kaIntvl>60</kaIntvl>
  <maxAsLimit>0</maxAsLimit>
  <pfxPeerTimeout>30</pfxPeerTimeout>
  <pfxPeerWaitTime>90</pfxPeerWaitTime>
  <reConnIntvl>60</reConnIntvl>
  <rtrId>2.2.2.2</rtrId>
</Dom-list>
client-host %
```

RESTCONF エージェントのトラブルシューティング

接続のトラブルシューティング

- **feature nxapi** コマンドを発行して、Web サーバーを有効にします。
- HTTP 用にポートを開くように **nxapi http port 80** コマンドが構成されていることを確認します。
- HTTPS 用にポートを開くように **nxapi https port 443** コマンドが構成されていることを確認します。
- スイッチの管理ポートに **ping** を実行して、スイッチが到達可能であることを確認します。

トラブルシューティングのエラー

次に、一般的なエラーメッセージと、それを解決するためのガイドラインを示します。

エラーメッセージ：申し訳ありませんが、検索しているページは現在使用できません (Sorry, the page you are looking for is currently unavailable)

- 要求を送信した直後（数秒後など）にこのメッセージを受信した場合は、次の点を確認してください：
 - 「接続のトラブルシューティング」に記載されているように、NXAPI 機能が有効になっています。
 - RESTCONF 機能が有効になっています (**show feature | grep restconf**)。RESTCONF が有効になっていない場合は、有効にします (**feature restconf**)。
 - ポートは、NX-API によって HTTP または HTTPS 用に構成されます。 **show nxapi** を使用して、ポートが設定されていることを確認します。

```
switch-1# show nxapi
nxapi enabled
HTTP Listen on port 80
HTTPS Listen on port 443
...
switch-1#
```

ポートが HTTP または HTTPS 用に構成されていない場合は、**nxapi http port 80** または **nxapi https port 443** を発行して構成します。

- 要求の送信後にこのメッセージを受信した場合（数分後など）、スイッチの最上位レベルからの過剰な同時クエリ要求でシステムが過負荷になっていないことを確認します。過剰な最上位レベルのクエリは、かなりのリソース負荷を生じる可能性があります。

次のいずれかによって、スイッチが過負荷になっていないことを確認できます。

- クライアントが送信する要求の数をスロットルバックします。

- スイッチで **no feature restconf** そして **feature restconf**を発行して RESTCONF エージェントを再起動します。

エフェメラル データ

RESTCONF のエフェメラル データについて

この機能は、エフェメラル データへのアクセスを提供します。エフェメラル データは、大量のデータです。DME は、各バッチがメモリ使用量の観点から管理可能なサイズになるように、データを取得するバッチ メカニズムを提供します。バッチのサイズは、取得する MO の数です。

パブリッシュされた Cisco-NX-OS-device.yang ファイルの「Ephemeral data」というコメントで、どのデータがエフェメラルであるかに関する情報を確認できます。

リクエスト内の URI が以下を指している場合にのみ、エフェメラル データからの出力が返されます。

- エフェメラル データのリーフ
- エフェメラル データの子を持つコンテナまたはリスト
- 直接のエフェメラルデータの子を持つリストをラップするために使用される空のコンテナ

システム レベルの GET クエリは、エフェメラル データを返しません。

RESTCONF のエフェメラル データの例

これは、エフェメラル データを取得する例です。

クライアントは、次の GET 要求メッセージを送信する場合があります。

```
GET
/restconf/data/Cisco-NX-OS-device:System//urib-items/table4-items/Table4-list=management/route4-items
HTTP/1.1
```

```
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Fri, 06 Mar 2020 11:10:30 GMT
Server: nginx/1.7.10
Content-Type: application/yang.data+json
```

```
{
  "route4-items": {
    "Route4-list": [{
      "prefix": "172.23.167.255/32",
      "flags": "0",
      ...
    ]
  }
}
```




第 29 章

gRPC エージェント

- [gRPC エージェント \(355 ページ\)](#)
- [エフェメラル データ \(372 ページ\)](#)

gRPC エージェント

gRPC エージェントについて

Cisco NX-OS gRPC プロトコルは、ネットワーク デバイスを管理し、その構成データを取得してインストールするためのメカニズムを定義します。プロトコルは、クライアントがデバイス構成を管理するために使用できる完全に正式なアプリケーションプログラミングインターフェイス (API) を公開します。

Cisco NX-OS gRPC プロトコルは、リモート プロシージャ コール (RPC) パラダイムを使用します。このパラダイムでは、外部クライアントが、Google プロトコル バッファ (GPB) で定義された API コールとそのサービス固有の引数を使用してデバイス設定を操作します。これらの GPB 定義 API は、同じ GPB 定義 API コンテキストで応答を返すデバイスへの RPC コールを透過的に引き起こします。

gRPC エージェントは、TLS によるセキュアな転送と、AAA によるユーザー認証と認可を提供します。

Cisco NX-OS gRPC プロトコルの機能上の目的は、特にステートレスとステートフルの両方の構成操作に関して、NETCONF によって提供されるプロトコルをミラーリングして、運用の柔軟性を最大限に高めることです。

Cisco NX-OS gRPC エージェントは、次のプロトコル操作をサポートしています。

- 結果
- GetConfig
- GetOper
- edit-config
- start-session

- CloseSession
- kill-session

gRPC エージェントは、次の 2 種類の操作をサポートしています：

- **ステートレス操作**は、セッションを作成せずに単一のメッセージ内で完全に実行されます。
- **ステートフル操作**は、複数のメッセージを使用して実行されます。実行される動作のシーケンスは、次のとおりです：
 1. セッションを開始します。このアクションは、一意のセッション ID を取得します。
 2. セッション ID を使用してセッションタスクを実行します。
 3. セッションを終了します。このアクションにより、セッション ID が無効になります。

次にあるのは、サポートされている操作です。gRPC エージェントによってエクスポートされる **.proto** ファイルの RPC 定義については、付録を参照してください。

動作	説明
start-session	クライアントとサーバー間の新しいセッションを開始し、一意のセッション ID を取得します。
edit-config	指定された YANG データ サブセットをターゲット データストアに書き込みます。
GetConfig	送信元データストアから指定された YANG 構成データのサブセットを取得します。
GetOper	送信元データストアから指定された YANG 動作データを取得します。
結果	指定された YANG 構成および動作データを送信元データストアから取得します。
KillSession	セッションを強制終了します。
CloseSession	セッションの適切な終了を要求します。

GetConfig、GetOper、および Get はステートレス操作であるため、セッション ID は必要ありません。

EditConfig は、ステートレスまたはステートフルのいずれかです。ステートレス操作の場合は、SessionID を 0 に指定します。ステートフル操作の場合、有効な（ゼロ以外の）SessionID が必要です。

gRPC エージェントは、セッションのタイムアウトをサポートします。セッションのアイドルタイムアウトはデバイスで構成できます。アイドルセッションは終了して削除されます。

gRPCに関する注意事項と制限事項

gRPC エージェントには、次のガイドラインと制限があります。

- Cisco NX-OS リリース 9.3 (3) 以降では、カスタム gRPC 証明書を構成している場合、**reload ascii** コマンドを入力すると構成が失われます。デフォルトの day-1 証明書に戻ります。**reload ascii** コマンドを入力すると、スイッチがリロードされます。スイッチが再び起動したら、gRPC カスタム証明書を再設定する必要があります。
- gRPC は、RFC 6536 で指定されている拡張ロールベース アクセス コントロール (RBAC) をサポートしていません。「network-admin」ロールを持つユーザーのみに、gRPC エージェントへのアクセス権が付与されます。
- Nexus デバイス向けの gRPC トラフィックは、デフォルト クラスのコントロールプレーン ポリサー (CoPP) にヒットします。gRPC ドロップの可能性を抑えるには、管理クラス of gRPC 構成ポートを使用して、カスタム CoPP ポリシーを構成してください。
- NX-OS 9.3 (1) 以降、gRPC クライアントからスイッチへの gRPCGet および GetConfig 要求には、明示的な名前空間とフィルタが含まれている必要があります。この要件は、OpenConfig YANG およびデバイス モデルへの要求に影響します。次のようなメッセージが表示された場合、要求は名前空間を伝送していません。

```
Request without namespace and filter is an unsupported operation
```

次の例は、この変更前の動作を使用した Get リクエストと応答を示しています。この例は、サポートされなくなった動作が原因で発生するエラーメッセージを示しています。

要求:

```
client-host % cat payload.json {
}
```

応答:

```
{ "rpc-reply":
  { "rpc-error": { ... Request without filtering is an unsupported operation ... } }
}
```

次に、NX-OS リリース 9.3 (1) 以降での正しい動作の Get リクエストと応答の例を示します。

要求:

```
client-host % cat payload.json {
  "namespace": "http://cisco.com/ns/yang/cisco-nx-os-device",
  "System": {}
}
```

応答:

```
{ "rpc-reply":
  { "data": { "System": ... } }
}
```

Cisco NX-OS リリース 9.3(3) 以降の gRPC エージェントの構成

gRPC エージェントは、次の構成コマンドをサポートしています。

パラメータ	説明
grpc port	ポート番号を設定します。 <i>port-id</i> の範囲は 1024 ~ 65535 です。50051 がデフォルトです。
grpc 証明書	証明書トラストポイント ID を指定します。詳細については、『Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。 https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/security/configuration/guide/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x_chapter_011010.html#task_2088148 (注) このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。 (注) 証明書が無効で、試行に 5 回連続して失敗した場合、サーバは次の syslog で保留状態になります。 <pre>2020 May 17 00:35:44 n9k %NXSDK-4-WARNING_MSG: grpc (5459) Reached maximal mgt server retry. Verify certificate and port config are valid.</pre>

gRPC エージェントは、構成ファイル（`/etc/mtx.conf`）の **[grpc]** セクションで次の構成パラメータをサポートします。

パラメータ	説明
idle_timeout	(オプション) アイドル状態のクライアントセッションが切断されるまでのタイムアウトを分単位で指定します。 デフォルトのタイムアウトは 5 分です。 値を 0 に設定するとタイムアウトが無効になります。

パラメータ	説明
limit	(オプション) 同時クライアントセッションの最大数を指定します。 デフォルト制限値は 5 セッションです。 範囲は 1 ～ 50 です。
security	セキュアな接続のタイプを指定します。 有効な選択肢は、次のとおりです。 • TLS の TLS • セキュアでない接続の場合は NONE

変更した構成ファイルを有効にするには、次のコマンドを使用して gRPC エージェントを再起動またはリロードする必要があります：

操作	コマンド
エージェントを再起動します	grpcctl restart または grpcctl stop grpcctl start
構成をリロードします	grpcctl reload

証明書について

Cisco NX-OS 9.3 (1) を使用した新しいスイッチ展開の場合、スイッチは 1 日限りの一時証明書を生成して、カスタム証明書をインストールするのに十分な時間を許可します。この一時的な 1 日証明書は、Cisco NX-OS の以前のバージョンで生成されたハードコードされたデフォルト証明書を置き換えます。スイッチのリロードまたはシステム スイッチオーバー イベントが発生すると、一時的な 1 日証明書が再生成されます。

たとえば、以前の NX-OS バージョンから NX-OS 9.3 (1) にアップグレードする場合など、スイッチにカスタム証明書がすでにインストールされている場合、既存の証明書はアップグレード後も保持され、使用されます。

次に、構成ファイルの **[grpc]** セクションの例を示します：

```
[grpc]
mtxadapter=/opt/mtx/lib/libmtxadaptergrpc.1.0.1.so
idle_timeout=10
limit=1
lport=50051
security=TLS
cert=/etc/grpc.pem
```

```
key=/etc/grpc.key
```

Cisco NX-OS リリース 9.3(3) 以降では、**lport=50051** は **grpc port** コマンドに置き換えられました。**cert=/etc/grpc.pem** と **key=/etc/grpc.key** は不要になり、**grpc certificate** コマンドに置き換えられました。

gRPC の構成

gRPC 機能は、**grpc** コマンドを使用して構成します。

コマンドで使用される証明書をスイッチにインポートするには、『Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。**grpc certificate**https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/security/configuration/guide/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x_chapter_011010.html#task_2088148



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

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch-1(config)#</pre>	グローバル コンフィギュレーション モードを開始します。
ステップ 2	feature grpc 例 : <pre>switch# feature grpc switch-1(config)#</pre>	ダイヤルイン用の gNMI インターフェイスをサポートする gRPC エージェントを有効にします。
ステップ 3	grpc port port-id 例 : <pre>switch(config)# grpc port 50051</pre>	ポート番号を設定します。 <i>port-id</i> の範囲は 1024 ~ 65535 です。50051 がデフォルトです。 (注) このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。
ステップ 4	grpc certificate 証明書 ID 例 :	証明書トラストポイント ID を指定します。詳細については、『Cisco Nexus 9000

	コマンドまたはアクション	目的
	<code>switch-1(config)# grpc certificate cert-1</code>	Series NX-OS Security Configuration Guide, Release 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/security/configuration/guide/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x_chapter_011010.html#task_2088148 (注) このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。

Cisco NX-OS リリース 9.3 (2) 以前の gRPC エージェントの構成

gRPC エージェントは、構成ファイル（`/etc/mtx.conf`）の `[grpc]` セクションで次の構成パラメータをサポートします。

パラメータ	説明
idle_timeout	（オプション）アイドル状態のクライアントセッションが切断されるまでのタイムアウトを分単位で指定します。 デフォルトのタイムアウトは 5 分です。 値を 0 に設定するとタイムアウトが無効になります。
limit	（オプション）同時クライアントセッションの最大数を指定します。 デフォルト制限値は 5 セッションです。 範囲は 1 ～ 50 です。
lport	（任意）gRPC エージェントがリスンするポート番号を指定します。 デフォルトのポートは 50051 です。
key	TLS 認証のキーファイルの場所を指定します。 デフォルトの場所は <code>/opt/mtx/etc/grpc.key</code> です。

パラメータ	説明
cert	TLS 認証の認定ファイルの場所を指定します。 デフォルトの場所は <code>/opt/mtx/etc/grpc.pem</code> です。 NX-OS リリース 9.3 (1) 以降では、gRPC エージェントの証明書にいくつかの変更が加えられています。以下の「証明書について」を参照してください。
security	セキュアな接続のタイプを指定します。 有効な選択肢は、次のとおりです。 • TLS の TLS • セキュアでない接続の場合は NONE

変更した構成ファイルを有効にするには、次のコマンドを使用して gRPC エージェントを再起動またはリロードする必要があります：

操作	コマンド
エージェントを再起動します	grpcctl restart または grpcctl stop grpcctl start
構成をリロードします	grpcctl reload

証明書について

Cisco NX-OS 9.3 (1) を使用した新しいスイッチ展開の場合、スイッチは 1 日限りの一時証明書を生成して、カスタム証明書をインストールするのに十分な時間を許可します。この一時的な 1 日証明書は、Cisco NX-OS の以前のバージョンで生成されたハードコードされたデフォルト証明書を置き換えます。スイッチのリロードまたはシステム スイッチオーバー イベントが発生すると、一時的な 1 日証明書が再生成されます。

たとえば、以前の NX-OS バージョンから NX-OS 9.3 (1) にアップグレードする場合など、スイッチにカスタム証明書がすでにインストールされている場合、既存の証明書はアップグレード後も保持され、使用されます。

次に、構成ファイルの **[grpc]** セクションの例を示します：

```
[grpc]
mtxadapter=/opt/mtx/lib/libmtxadaptergrpc.1.0.1.so
idle_timeout=10
limit=1
```

```
lport=50051
security=TLS
cert=/etc/grpc.pem
key=/etc/grpc.key
```



- (注) 以下の場合、ハードコードされたキーがある場合は、それを削除し、永続的な場所にカスタムキーをインストールする必要があります。

[ハードコードされた証明書の削除 (Delete the Hard-Coded Certificate)]

9.3 (1) より前のバージョンの NX-OS を使用してスイッチを導入した場合は、ハードコードされた証明書を削除する必要があります。

[証明書の場所を変更します (Change the Certificate Location)]

証明書の以前のデフォルトの場所 (/etc/mtx.conf file) は永続的ではなくなりました。

デフォルトでは、/etc/mtx.conf.user ファイルは存在しないため、ファイルを作成し、キーと証明書の新しい場所を指定して、変更を永続的にする必要があります。次に例を示します。

```
[grpc]
cert="new location"
key="new location"
```



- (注) デュアルスーパーバイザモジュールを搭載したスイッチの場合、証明書ファイルをアップロードし、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 /etc/mtx.conf.user ファイルを変更する必要があります。両方のスーパーバイザでこれを行わないと、証明書ファイルはスイッチオーバー後に永続的ではありません。

[期限切れの証明書の交換 (Replace an Expired Certificate)]

一時的な 1 日証明書の有効期限が切れている場合は、それを削除して新しい証明書に置き換えます。

[構成をリロードします (Reload the Configuration File)]

スイッチをリロードするかどうかに応じて、適切なオプションを使用します：

- スイッチをリロードせずに新しい証明書を使用するために gRPC エージェントを再起動する場合は、次の手順を実行します：
 1. 証明書とキーを /etc/mtx.conf に追加します。
 2. 変更した構成ファイルを有効にするには、**no feature grpc**を発行して gRPC エージェントを無効にしてから、再度有効にするために**feature grpc**を発行します。
- スイッチをリロードする場合は、/etc/mtx.conf.user を変更するだけで十分です。/etc/mtx.conf を変更する必要はありません。

gRPC エージェントの使用

一般的なコマンド

gRPC エージェントを有効または無効にするには、**[no] feature grpc** コマンドを発行します。

一般的なコマンド

gRPC エージェントで使用可能な制御コマンドは次のとおりです。

grpcctl { status | start | restart | reload | stop }

エージェントステータスの表示

```
bash-4.2# grpcctl status
xosdsd is stopped
grpcctl is stopped
```

エージェントの起動

```
bash-4.2# grpcctl start
Starting Grpc Agent: [OK]
```

例：JSON フォーマットの基本的な YANG パス

```
client-host % cat payload.json
{
  "namespace": "http://cisco.com/ns/yang/cisco-nx-os-device",
  "System": {
    "bgp-items": {
      "inst-items": {
        "dom-items": {
          "Dom-list": {
            "name": "default",
            "rtrId": "7.7.7.7",
            "holdIntvl": "100"
          }
        }
      }
    }
  }
}
```



(注) JSON 構造は、読みやすいように整形されています。

サーバーへの EditConfig 要求の送信

```
client-host % ./grpc_client -username=admin -password=cisco -operation=EditConfig
```

```
-e_oper=Merge -def_op=Merge -err_op=stop-on-error -infile=payload.json -reqid=1
-source=running -tls=true -serverAdd=192.0.20.123 -lport=50051
```

```
#####
Starting the client service
#####
TLS set true for client requestslems.cisco.com
TLS FLAG:1
192.0.20.123:50051
All the client connections are secured
Sending EditConfig request to the server
sessionid is
0
reqid:1
{"rpc-reply":{"ok":""}}
```

サーバーへの GetConfig 要求の送信

```
client-host % ./grpc_client -username=admin -password=cisco -operation=GetConfig
-infile=payload.json -reqid=1 -source=running -tls=true -serverAdd=192.0.20.123
-lport=50051
```

```
#####
Starting the client service
#####
TLS set true for client requestslems.cisco.com
TLS FLAG:1
192.0.20.123:50051
All the client connections are secured
Sending GetConfig request to the server
in get config
Got the response from the server
#####
Yang Json is:
#####
{"rpc-reply":{"data":{"System":{"top-items":{"inst-items":{"don-items":{"don-list":{"name":"default","trid":"7.7.7.7","holdintvl":"100"}}}}}}}}
#####
client-host %
```

gRPC エージェントのトラブルシューティング

接続のトラブルシューティング

- クライアントシステムから、エージェントがポートでリッスンしていることを確認します。次に例を示します。

```
client-host % nc -z 192.0.20.222 50051
Connection to 192.0.20.222 50051 port [tcp/*] succeeded!
client-host % echo $?
0
client-host %
```

- NX-OS で、を発行して gRPC エージェントのステータスを確認します **show feature | grep grpc**。

gRPC Protobuf ファイル

gRPC エージェントは、/opt/mtx/etc/nxos_grpc.proto にある proto 定義ファイルでサポートされている操作とデータ構造をエクスポートします。ファイルは、gRPC エージェント RPM に含まれています。次に、定義を示します。

```
// Copyright 2016, Cisco Systems Inc.
// All rights reserved.

syntax = "proto3";

package NXOSExtensibleManagabilityService;

// Service provided by Cisco NX-OS gRPC Agent
service gRPCConfigOper {

    // Retrieves the specified YANG configuration data subset from the
    // source datastore
    rpc GetConfig(GetConfigArgs) returns(stream GetConfigReply) {};

    // Retrieves the specified YANG operational data from the source datastore
    rpc GetOper(GetOperArgs) returns(stream GetOperReply) {};

    // Retrieves the specified YANG configuration and operational data
    // subset from the source datastore
    rpc Get(GetArgs) returns(stream GetReply) {};

    // Writes the specified YANG data subset to the target datastore
    rpc EditConfig(EditConfigArgs) returns(EditConfigReply) {};

    // Starts a new session between the client and server and acquires a
    // unique session ID
    rpc StartSession(SessionArgs) returns(SessionReply) {};

    // Requests graceful termination of a session
    rpc CloseSession(CloseSessionArgs) returns (CloseSessionReply) {};

    // Forces the termination of a session
    rpc KillSession(KillArgs) returns(KillReply) {};

    // Unsupported; reserved for future
    rpc DeleteConfig(DeleteConfigArgs) returns(DeleteConfigReply) {};

    // Unsupported; reserved for future
    rpc CopyConfig(CopyConfigArgs) returns(CopyConfigReply) {};

    // Unsupported; reserved for future
    rpc Lock(LockArgs) returns(LockReply) {};

    // Unsupported; reserved for future
    rpc UnLock(UnLockArgs) returns(UnLockReply) {};

    // Unsupported; reserved for future
    rpc Commit(CommitArgs) returns(CommitReply) {};

    // Unsupported; reserved for future
    rpc Validate(ValidateArgs) returns(ValidateReply) {};

    // Unsupported; reserved for future
    rpc Abort(AbortArgs) returns(AbortReply) {};
```

```
}

message GetConfigArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the source datastore; only "running" is supported.
    // Default is "running".
    string Source = 3;
}

message GetConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message GetOperArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message GetOperReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message GetArgs
{
    // JSON-encoded YANG data to be retrieved
    string YangPath=1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message GetReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // JSON-encoded YANG data that was retrieved
    string YangData = 2;
}
```

```

        // JSON-encoded error information when request fails
        string Errors = 3;
    }

    message EditConfigArgs
    {
        // JSON-encoded YANG data to be edited
        string YangPath = 1;

        // Specifies the operation to perform on the configuration datastore with
        // the YangPath data. Possible values are:
        //   create
        //   merge
        //   replace
        //   delete
        //   remove
        // If not specified, default value is "merge".
        string Operation = 2;

        // A unique session ID acquired from a call to StartSession().
        // For stateless operation, this value should be set to 0.
        int64 SessionID = 3;

        // (Optional) Specifies the request ID. Default value is 0.
        int64 ReqID = 4;

        // (Optional) Specifies the target datastore; only "running" is supported.
        // Default is "running".
        string Target = 5;

        // Specifies the default operation on the given object while traversing
        // the configuration tree.
        // The following operations are possible:
        //   merge: merges the configuration data with the target datastore;
        //           this is the default.
        //   replace: replaces the configuration data with the target datastore.
        //   none: target datastore is unaffected during the traversal until
        //           the specified object is reached.
        string DefOp = 6;

        // Specifies the action to be performed in the event of an error during
        // configuration. Possible values are:
        //   stop
        //   roll-back
        //   continue
        // Default is "roll-back".
        string ErrorOp = 7;
    }

    message EditConfigReply
    {
        // The request ID specified in the request.
        int64 ReqID = 1;

        // If EditConfig is successful, YangData contains a JSON-encoded "ok" response.
        string YangData = 2;

        // JSON-encoded error information when request fails
        string Errors = 3;
    }

    message DeleteConfigArgs
    {

```



```
// A unique session ID acquired from a call to StartSession().
// For stateless operation, this value should be set to 0.
int64 SessionID = 1;

// (Optional) Specifies the request ID. Default value is 0.
int64 ReqID = 2;

// (Optional) Specifies the target datastore; only "running" is supported.
// Default is "running".
string Target = 3;
}

message DeleteConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If DeleteConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CopyConfigArgs
{
    // A unique session ID acquired from a call to StartSession().
    // For stateless operation, this value should be set to 0.
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the source datastore; only "running" is supported.
    // Default is "running".
    string Source = 3;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 4;
}

message CopyConfigReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If CopyConfig is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message LockArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID=2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
```

```

        string Target = 3;
    }

message LockReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Lock is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message UnlockArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;

    // (Optional) Specifies the target datastore; only "running" is supported.
    // Default is "running".
    string Target = 3;
}

message UnlockReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Unlock is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message SessionArgs
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;
}

message SessionReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;
    int64 SessionID = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CloseSessionArgs
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;

    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 2;
}

```

```
message CloseSessionReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If CloseSession is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message KillArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    int64 SessionIDToKill = 2;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 3;
}

message KillReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Kill is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message ValidateArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message ValidateReply
{
    // The request ID specified in the request.
    int64 ReqID = 1;

    // If Validate is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message CommitArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}
```

```

}

message CommitReply
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;

    // If Commit is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

message AbortArgs
{
    // A unique session ID acquired from a call to StartSession().
    int64 SessionID = 1;

    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 2;
}

message AbortReply
{
    // (Optional) Specifies the request ID. Default value is 0.
    int64 ReqID = 1;

    // If Abort is successful, YangData contains a JSON-encoded "ok" response.
    string YangData = 2;

    // JSON-encoded error information when request fails
    string Errors = 3;
}

```

エフェメラル データ

gRPC のエフェメラル データについて

Cisco NX-OS リリース 9.3(3) 以降、この機能はエフェメラル データへのアクセスを提供します。エフェメラル データは、大量のデータです。DME は、各バッチがメモリ使用量の観点から管理可能なサイズになるように、データを取得するバッチメカニズムを提供します。バッチのサイズは、取得する MO の数です。

バッチ処理メカニズムは gRPC を介して公開されません。バッチ処理は内部的に処理されます。gRPC バッチ処理は明示的に使用しません。REST API を介して DME から直接アクセスしようとするときには、バッチ処理を使用できます。

パブリッシュされた YANG ファイルの「Ephemeral data」というコメントで、どのデータがエフェメラルであるかに関する情報を確認できます。

エフェメラル データからの出力は、要求のフィルタが以下を指している場合にのみ返されます。

- エフェメラル データのリーフ

- エフェメラル データの子を持つコンテナまたはリスト
- 直接のエフェメラルデータの子を持っていたリストをラップするために使用される空のコンテナ

システム レベルの GET クエリは、エフェメラル データを返しません。

gRPC のエフェメラル データの例

これは、gRPC のエフェメラル データを取得する例です。

```

ayyim-lnx.cisco.com:200> cat urib.mgmt.json
{"namespace":"http://cisco.com/ns/yang/cisco-nx-os-device",
  "System": {
    "urib-items":{
      "table4-items":{
        "Table4-list":{
          "vrfName":"management",
          "route4-items":{
            "Route4-list":{
              }
            }
          }
        }
      }
    }
  }
}

ayyim-lnx.cisco.com:395> ./grpc_client -username=admin -password=C\!sco123 -operation=Get
-reqid=1 -serverAdd=172.23.167.216 -lport=50051 -tls=true -infile=payloads/urib.mgmt.json
-cafile=grpc.pem.n9kv-blade2

#####
Starting the client service
#####
TLS set true for client requestslems.cisco.com
TLS FLAG:1
172.23.167.216:50051
All the client connections are secured
Sending Get request to the server
sending get request
Got the response from the server
#####
Yang Json is:
#####
{
  "rpc-reply" : {
    "data" : {
      "System" : {
        "xmlns" : "http://cisco.com/ns/yang/cisco-nx-os-device",
        "urib-items" : {
          "table4-items" : {
            "Table4-list" : [
              {
                "vrfName" : "management",
                "route4-items" : {
                  "Route4-list" : [
                    {
                      "prefix" : "172.23.167.255/32",
                      "flags" : "0",
                      "mBestNextHopCount" : "0",

```

```

"nh4-items" : {
  "NextHop4-list" : [
    {
      "id" : "0",
      "address" : "172.23.167.255",
      "bindinglabel" : "0",
      "encapType" : "none",
      "interfaceName" : "mgmt0",
      "isBest" : "true",
      "metric" : "0",
      "owner" : "broadcast",
      "preference" : "0",
      "routeType" : "unknown",
      "segidType" : "null",
      "segmentId" : "0",
      "tag" : "0",
      "tunnelId" : "0",
      "uptime" : "00:18:18",
      "vrf" : "management"
    }
  ]
},
"pendingHw" : "false",
"pendingUfdm" : "false",
"sortKey" : "0",
"uBestNextHopCount" : "1"
},
{
  "prefix" : "172.23.167.216/32",
  "flags" : "0",
  "mBestNextHopCount" : "0",
  "nh4-items" : {
    "NextHop4-list" : [
      {
        "id" : "0",
        "address" : "172.23.167.216",
        "bindinglabel" : "0",
        "encapType" : "none",
        "interfaceName" : "mgmt0",
        "isBest" : "true",
        "metric" : "0",
        "owner" : "local",
        "preference" : "0",
        "routeType" : "unknown",
        "segidType" : "null",
        "segmentId" : "0",
        "tag" : "0",
        "tunnelId" : "0",
        "uptime" : "00:18:18",
        "vrf" : "management"
      }
    ]
  },
  "pendingHw" : "false",
  "pendingUfdm" : "false",
  "sortKey" : "0",
  "uBestNextHopCount" : "1"
},
{
  "prefix" : "172.23.167.20/32",
  "flags" : "0",
  "mBestNextHopCount" : "0",
  "nh4-items" : {
    "NextHop4-list" : [

```

```

    {
      "id" : "0",
      "address" : "172.23.167.20",
      "bindinglabel" : "0",
      "encapType" : "none",
      "interfaceName" : "mgmt0",
      "isBest" : "true",
      "metric" : "0",
      "owner" : "am",
      "preference" : "250",
      "routeType" : "unknown",
      "segidType" : "null",
      "segmentId" : "0",
      "tag" : "0",
      "tunnelId" : "0",
      "uptime" : "00:03:06",
      "vrf" : "management"
    }
  ],
  "pendingHw" : "false",
  "pendingUfdm" : "false",
  "sortKey" : "0",
  "uBestNextHopCount" : "1"
},
{
  "prefix" : "172.23.167.8/32",
  "flags" : "0",
  "mBestNextHopCount" : "0",
  "nh4-items" : {
    "NextHop4-list" : [
      {
        "id" : "0",
        "address" : "172.23.167.8",
        "bindinglabel" : "0",
        "encapType" : "none",
        "interfaceName" : "mgmt0",
        "isBest" : "true",
        "metric" : "0",
        "owner" : "am",
        "preference" : "250",
        "routeType" : "unknown",
        "segidType" : "null",
        "segmentId" : "0",
        "tag" : "0",
        "tunnelId" : "0",
        "uptime" : "00:02:23",
        "vrf" : "management"
      }
    ]
  },
  "pendingHw" : "false",
  "pendingUfdm" : "false",
  "sortKey" : "0",
  "uBestNextHopCount" : "1"
},
{
  "prefix" : "0.0.0.0/0",
  "flags" : "0",
  "mBestNextHopCount" : "0",
  "nh4-items" : {
    "NextHop4-list" : [
      {
        "id" : "0",

```

```

        "address" : "172.23.167.1",
        "bindinglabel" : "0",
        "encapType" : "none",
        "interfaceName" : "N/A",
        "isBest" : "true",
        "metric" : "0",
        "owner" : "static",
        "preference" : "1",
        "routeType" : "",
        "segidType" : "null",
        "segmentId" : "0",
        "tag" : "0",
        "tunnelId" : "0",
        "uptime" : "00:18:18",
        "vrf" : "management"
    }
}
},
"pendingHw" : "false",
"pendingUfdm" : "false",
"sortKey" : "0",
"uBestNextHopCount" : "1"
},
{
    "prefix" : "172.23.167.0/24",
    "flags" : "0",
    "mBestNextHopCount" : "0",
    "nh4-items" : {
        "NextHop4-list" : [
            {
                "id" : "0",
                "address" : "172.23.167.216",
                "bindinglabel" : "0",
                "encapType" : "none",
                "interfaceName" : "mgmt0",
                "isBest" : "true",
                "metric" : "0",
                "owner" : "direct",
                "preference" : "0",
                "routeType" : "",
                "segidType" : "null",
                "segmentId" : "0",
                "tag" : "0",
                "tunnelId" : "0",
                "uptime" : "00:18:18",
                "vrf" : "management"
            }
        ]
    },
    "pendingHw" : "false",
    "pendingUfdm" : "false",
    "sortKey" : "0",
    "uBestNextHopCount" : "1"
},
{
    "prefix" : "172.23.167.1/32",
    "flags" : "0",
    "mBestNextHopCount" : "0",
    "nh4-items" : {
        "NextHop4-list" : [
            {
                "id" : "0",
                "address" : "172.23.167.1",
                "bindinglabel" : "0",

```


Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド、リリース 9.3(x)



第 30 章

gNMI : gRPC ネットワーク管理インターフェイス

この章は次のトピックで構成されています。

- [gNMI について \(379 ページ\)](#)
- [gNMI の VRF コンテキスト \(380 ページ\)](#)
- [gNMI サブスクリプション RPC \(380 ページ\)](#)
- [gNMI に関する注意事項と制限事項 \(382 ページ\)](#)
- [gNMI の構成 \(384 ページ\)](#)
- [gNMI : gRPC ネットワーク管理インターフェイス \(386 ページ\)](#)
- [サーバー証明書の構成 \(387 ページ\)](#)
- [キー/証明書の生成の例 \(388 ページ\)](#)
- [Cisco NX-OS リリース 9.3\(2\) 以前のキー/証明書の生成と構成の例 \(388 ページ\)](#)
- [Cisco NX-OS リリース 9.3\(3\) 以降のキー/証明書の生成と構成の例 \(389 ページ\)](#)
- [gNMI の確認 \(391 ページ\)](#)
- [クライアント \(398 ページ\)](#)
- [DME サブスクリプションの例 : PROTO エンコーディング \(398 ページ\)](#)
- [機能 \(400 ページ\)](#)
- [結果 \(404 ページ\)](#)
- [設定 \(405 ページ\)](#)
- [登録 \(406 ページ\)](#)
- [ストリーミング Syslog \(411 ページ\)](#)
- [トラブルシューティング \(417 ページ\)](#)

gNMI について

gNMI は、トランスポートプロトコルとして gRPC (Google リモート プロシージャ コール) を使用します。

Cisco NX-OS は、Cisco Nexus 9000 シリーズ スイッチで実行されるテレメトリ アプリケーションへのダイヤルイン サブスクリプション用に gNMI をサポートします。過去のリリースでは

gRPC を介したテレメトリイベントがサポートされていましたが、スイッチはテレメトリデータをテレメトリレシーバにプッシュしていました。この方法はダイヤルアウトと呼ばれていました。

gNMI を使用すると、アプリケーションはスイッチから情報をプルできます。サポートされているテレメトリ機能を学習し、必要なテレメトリサービスのみをサブスクライブすることで、特定のテレメトリサービスにサブスクライブします。

表 21: サポートされる gNMI RPC

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

gNMI の VRF コンテキスト

以前のリリースでは、gRPC サーバーは管理 VRF でのみ実行されていました。その結果、gRPC プロセスはこの VRF でのみ通信でき、管理インターフェイスはすべての gRPC 呼び出しをサポートする必要がありました。この構成は、リリース 9.3(1) でもサポートされています。

リリース 9.3(1) 以降、gRPC 機能には、Cisco Nexus 9000 スイッチごとに合計 2 つの gRPC サーバー用のデフォルト VRF が含まれています。VRF ごとに 1 つの gRPC サーバーを実行することも、管理 VRF で 1 つの gRPC サーバーのみを実行することもできます。デフォルト VRF で gRPC をサポートすると、かなり部分のトラフィック負荷が望ましくないものである、管理 VRF からの gRPC コールの処理を柔軟にオフロードできます。

2 台の gRPC サーバーを構成する場合は、次の点に注意してください。

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

gNMI サブスクライブ RPC

Cisco NX-OS 9.3(1) リリース以降では、次の gNMI サブスクリプション機能がサポートされています。

表 22: サブスクライブ オプション

タイプ	サブタイプ	サポートの有無	説明
[1 回 (Once)]		はい	スイッチは、指定されたすべてのパスに対して現在の値を 1 回だけ送信します。
ポーリング (Poll)		はい	スイッチは、ポーリングメッセージを受信するたびに、指定されたすべてのパスの現在の値を送信します。
ストリーム	サンプル	はい	ストリームサンプル間隔ごとに 1 回、スイッチは指定されたすべてのパスの現在の値を送信します。サポートされるサンプル間隔の範囲は 1 ~ 604800 秒です。 デフォルトのサンプル間隔は 10 秒です。
	[変更時 (On_Change)]	はい	スイッチは初期状態として現在の値を送信しますが、指定されたパスのいずれかに作成、変更、または削除などの変更が発生した場合にのみ値を更新します。
	[ターゲット定義 (Target_Defined)]	いいえ	

オプションの SUBSCRIBE フラグ

SUBSCRIBE オプションでは、表にリストされているオプションへの応答を変更するオプションのフラグを使用できます。Cisco NX-OS リリース 9.3(1) 以降では、updates_only オプションフラグがサポートされています。これは、ON_CHANGE サブスクリプションに適用されます。このフラグが設定されている場合、スイッチは通常最初の応答で送信される初期スナップショットデータ（現在の状態）を抑制します。

次のフラグはサポートされていません。

- [エイリアス (aliases)]
- [集約許可 (allow_aggregation)]
- [拡張 (extensions)]
- prefix
- [qos]

サブスクリプションフラグのサポートメトリックは次のとおりです。

表 23: **SUBSCRIBE** フラグのサポートメトリック

サブスクリプションタイプ	[ハートビート間隔 (heartbeat_interval)]	[冗長抑制 (suppress_redundant)]
[変更時 (On_Change)]	発信元: デバイス YANG、 OpenConfig YANG、DME	該当なし
サンプル	発信元: デバイス YANG、 OpenConfig YANG、DME	発信元: デバイス YANG、 OpenConfig YANG

gNMI に関する注意事項と制限事項

gNMI に関する注意事項と制限事項は次のとおりです。

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

回避策として、コマンドを入力して gRPC を完全に無効にし、コマンドと既存の gRPC コンフィギュレーション コマンドを入力して再プロビジョニングします。 **no feature grpcfeature grpc** たとえば、**grpc certificate** または **grpc port**。また、管理 VRF の既存の通知に再登録する必要もあります。

- 次のような既存の CLI 設定を使用して OpenConfig ルーティングポリシーをサブスクリプションしようすると、OpenConfig モデルの現在の実装により空の値が返されます。

```
ip prefix-list bgp_v4_drop seq 5 deny 125.2.0.0/16 le 32
ipv6 prefix-list bgp_v6_drop seq 5 deny cafe:125:2::/48 le 128
```

using the xpath

```
openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v4_drop]/config
```

```
openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v6_drop]/config
```

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

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

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



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

- origin、use_models、またはその両方の使用は、gNMI サブスクリプションではオプションです。
- gNMI サブスクリプションは、Cisco DME およびデバイス YANG データモデルをサポートします。Cisco NX-OS リリース 9.3(3)以降、サブスクライブは OpenConfig モデルをサポートします。
- 9.3(x) より前の Cisco NX-OS においてサポートされるプラットフォームの詳細については、そのリリース向けガイドのプログラマビリティ機能のプラットフォームサポートを参照してください。Cisco NX-OS リリース 9.3(x) 以降、サポートされるプラットフォームの詳細については、[Nexus Switch Platform Matrix](#)を参照してください。
- gNMI 機能は、Subscribe and Capability gNMI RPC をサポートします。
- この機能は、JSON および gnmi.proto エンコーディングをサポートします。この機能は、protobuf.any エンコーディングをサポートしていません。
- 各 gNMI メッセージの最大サイズは 12 MB です。収集されたデータの量が最大値 12 MB を超えると、収集されたデータはドロップされます。gNMION_CHANGEモードにのみ適用されます。

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

- すべてのサブスクリプションで、最大 150K の集約 MO がサポートされます。より多くの MO に登録すると、収集データがドロップする可能性があります。
- この機能はサブスクリプション要求の単一パス プレフィックスをサポートしていませんが、サブスクリプションには空のプレフィックス フィールドを含めることができます。
- gNMI をサポートする gRPC プロセスは、CPU 使用率を CPU の 75% に、メモリを 1.5 GB に制限する HIGH_PRIO 制御グループを使用します。
- **show grpc gnmi** コマンドには、次の考慮事項があります。
 - gRPC エージェントは、コールが終了した後、最大 1 時間 gNMI コールを保持します。
 - コールの合計数が 2000 を超えると、gRPC エージェントは、内部クリーンアップルーチンに基づいて終了したコールを消去します。

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

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

2 つの gRPC サーバーを構成する場合は、次の点に注意してください。

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

gNMI の構成

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

コマンドで使用される証明書をスイッチにインポートするには、『Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。 **grpc certificate**https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/security/configuration/guide/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x_chapter_011010.html#task_2088148



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

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : switch-1# configure terminal switch-1(config)#	グローバル コンフィギュレーション モードを開始します。
ステップ 2	feature grpc 例 : switch-1# feature grpc switch-1(config)#	ダイヤルイン用の gNMI インターフェイスをサポートする gRPC エージェントを有効にします。
ステップ 3	(任意) grpc port port-id 例 : switch-1(config)# grpc port 50051	ポート番号を設定します。 <i>port-id</i> の範囲は 1024 ~ 65535 です。50051 がデフォルトです。 (注) このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。
ステップ 4	grpc certificate 証明書 ID 例 : switch-1(config)# grpc certificate cert-1	証明書トラストポイント ID を指定します。詳細については、『Cisco Nexus 9000 Series NX-OS Security Configuration Guide, Release 9.3(x)』の「Installing Identity Certificates」セクションを参照してください。 https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/93x/security/configuration/guide/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x/b-cisco-nexus-9000-nx-os-security-configuration-guide-93x_chapter_011010.html#task_2088148 (注) このコマンドは、Cisco NX-OS リリース 9.3(3) 以降で使用できます。
ステップ 5	grpc gnmi max-concurrent-call number 例 :	スイッチ上の gNMI サーバーに対する同時ダイヤルイン呼び出しの制限を設定し

	コマンドまたはアクション	目的
	<pre>switch-1(config)# grpc gnmi max-concurrent-call 16 switch-1(config)#</pre>	ます。1 ～ 16 の範囲で制限を構成します。デフォルトの制限は 8 です。 構成する最大値は、各 VRF に対するものです。制限を 16 に設定し、gNMI が管理 VRF とデフォルト VRF の両方に設定されている場合、各 VRF は 16 の同時 gNMI コールをサポートします。 このコマンドは、進行中または進行中の gNMI コールには影響しません。代わりに、gRPC は新しいコールに制限を適用するため、進行中のコールは影響を受けず、完了できます。 (注) 設定された制限は、gRPCConfigOper サービスには影響しません。
ステップ 6	(任意) grpc use-vrf default 例 : <pre>switch(config)# grpc use-vrf default</pre>	gRPC エージェントがデフォルト VRF からの着信（ダイヤルイン）RPC 要求を受け入れられるようにします。この手順により、デフォルト VRF が着信 RPC 要求を処理できるようになります。デフォルトでは、gRPC 機能が有効になっている場合、管理 VRF が着信 RPC 要求を処理します。 (注) 要求が VRF 間を通過しないように、両方の VRF が要求を個別に処理します。

gNMI : gRPC ネットワーク管理インターフェイス

このセクションでは、ターゲットデバイスから構成を変更および取得するための gRPC ベースのプロトコルを定義します。また、ターゲットデバイスからデータ収集システムへのテレメトリストリームの制御と生成を定義します。その目的は、単一の gRPC サービス定義で構成とテレメトリの両方をカバーできるようにすることです。これにより、ターゲット上の単一の実装と、単一の NMS 要素が、テレメトリと構成 RPC を介してデバイスと対話できるようになります。

サーバー証明書の構成

TLS 証明書を設定し、スイッチに正常にインポートした場合の **show grpc gnmi service statistics** コマンドの出力例を次に示します。

```
#show grpc gnmi service statistics

=====
gRPC Endpoint
=====

Vrf          : management
Server address : [::]:50051

Cert notBefore : Mon Jan 27 15:34:08 PDT 2020
Cert notAfter  : Tue Jan 26 15:34:08 PDT 2021

Max concurrent calls      : 8
Listen calls              : 1
Active calls              : 0

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

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

gNMIはgRPCを介して通信し、TLSを使用してスイッチとクライアント間のチャネルをセキュアにします。デフォルトのハードコードされたgRPC証明書は、スイッチに同梱されなくなりました。デフォルトの動作は、次に示すように、スイッチで生成される有効期限が1日の自己署名キーと証明書です。

証明書の有効期限が切れているか、正常にインストールできなかった場合は、1-D デフォルト証明書が表示されます。次に、**show grpc gnmi service statistics** コマンドの出力を示します。

```
#show grpc gnmi service statistics

=====
gRPC Endpoint
=====

Vrf          : management
Server address : [::]:50051

Cert notBefore : Wed Mar 11 19:43:01 PDT 2020
Cert notAfter  : Thu Mar 12 19:43:01 PDT 2020

Max concurrent calls      : 8
Listen calls              : 1
Active calls              : 0

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

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

有効期限は1日ですが、この一時証明書を使用してテストを簡単に行えます。長期的には、新しいキー/証明書を生成する必要があります。



- (注) 証明書の有効期限が切れた後、キー/証明書を再生成するには、次の2つの方法があります。
- スイッチをリロードします。
 - /opt/mtx/etc フォルダのキー/証明書を手動で削除し、**no feature grpc** および **feature grpc** コマンドを入力します。

キー/証明書の生成の例

キー/証明書を生成するには、次の例に従います。

- [Cisco NX-OS リリース 9.3\(2\) 以前のキー/証明書の生成と構成の例 \(388 ページ\)](#)

Cisco NX-OS リリース 9.3(2) 以前のキー/証明書の生成と構成の例

次に、キー/証明書を生成する例を示します。

ID 証明書の生成の詳細については、『Cisco Nexus 9000 シリーズ NX-OS セキュリティ コンフィギュレーション ガイド リリース 9.3(x)』の「[ID 証明書のインストール](#)」のセクションを参照してください。

手順

ステップ 1 自己署名キーと pem ファイルを生成します。

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

ステップ 2 キーと pem ファイルを生成した後、gRPC サービスが証明書を取得するように Bash シェルの mtz.conf.user ファイルを変更します。

```
[grpc]
key = /bootflash/self-sign2048.key
cert = /bootflash/self-sign2048.pem
```

ステップ 3 ボックスをリロードして、gRPC サービスが証明書を取得するようにします。

ステップ 4 gRPC が証明書を使用していることを確認します。

```
switch# show grpc gnmi service statistics

=====
```

```
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051

Cert notBefore : Nov 5 16:48:58 2015 GMT
Cert notAfter : Nov 5 16:48:58 2035 GMT

Max concurrent calls : 16
Listen calls : 1
Active calls : 0

Number of created calls : 953
Number of bad calls : 0

Subscription stream/once/poll : 476/238/238

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 10
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0
```

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

次に、キー/証明書を生成する例を示します。



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

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

手順

ステップ 1 自己署名キーと pem ファイルを生成します。

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

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

```
switch# run bash sudo su
bash-4.3# cd /bootflash/
bash-4.3# openssl pkcs12 -export -out self_sign2048.pfx -inkey self_sign2048.key -in
self_sign2048.pem -certfile self_sign2048.pem -password pass:Ciscolab123!
bash-4.3# exit
```

ステップ 3 セットアップを確認します。

```
switch(config)# show crypto ca certificates
Trustpoint: mytrustpoint
certificate:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
serial=0413
notBefore=Nov  5 16:48:58 2015 GMT
notAfter=Nov  5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient

CA certificate 0:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
serial=0413
notBefore=Nov  5 16:48:58 2015 GMT
notAfter=Nov  5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
```

ステップ 4 トラストポイントを使用するように gRPC を構成します。

```
switch(config)# grpc certificate mytrustpoint
switch(config)# show run grpc

!Command: show running-config grpc
!Running configuration last done at: Thu Jul  2 12:24:02 2020
!Time: Thu Jul  2 12:24:05 2020

version 9.3(5) Bios:version 05.38
feature grpc

grpc gnmi max-concurrent-calls 16
grpc use-vrf default
grpc certificate mytrustpoint
```

ステップ 5 gRPC が証明書を使用していることを確認します。

```
switch# show grpc gnmi service statistics

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051
```

```
Cert notBefore : Nov 5 16:48:58 2015 GMT
Cert notAfter  : Nov 5 16:48:58 2035 GMT

Max concurrent calls : 16
Listen calls : 1
Active calls : 0

Number of created calls : 953
Number of bad calls : 0

Subscription stream/once/poll : 476/238/238

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 10
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0
```

gNMI の確認

gNMI 構成を確認するには、次のコマンドを入力します。

コマンド	説明
show grpc gnmi service statistics	管理 VRF またはデフォルト VRF（設定されている場合）のエージェント実行ステータスの概要を表示します。また、次の項目も表示されます。 • 基本の全般的なカウンタ • 証明書の有効期限日時 （注） 証明書の有効期限が切れている場合、エージェントは要求を受け入れることができません。

コマンド	説明
show grpc gnmi rpc summary	次のステータスが表示されます。 • 受信した機能 RPC の数。 • 機能 RPC エラー。 • 受信した Get RPC の数。 • Get RPC エラー。 • 受信した Set RPC の数。 • Set RPC エラー。 • 詳細なエラー タイプとエラー数。

コマンド	説明
show grpc gnmi transactions	

コマンド	説明
	show grpc gnmi transactions コマンドは最も密度が高く、多くの情報が含まれています。これは、スイッチが受信した最新 50 の gNMI トランザクションの履歴バッファです。新しい RPC が着信すると、末尾から最も古い履歴エントリが削除されます。次に、表示内容について説明します。 • [RPC] : 受信した RPC のタイプ (Get、Set、機能) を示します。 • [データタイプ (DataType)] : Get の場合のみです。値は ALL、CONFIG、および STATE です。 • [セッション (Session)] : このトランザクションに割り当てられている一意のセッション ID を示します。他のログファイルで見つかったデータを関連付けるために使用できます。 • [入力時間 (Time In)] : gNMI ハンドラが RPC を受信したときのタイムスタンプを示します。 • [期間 (Duration)] : 要求を受信してから応答を返すまでの時間差です (ミリ秒単位)。 • [ステータス (Status)] : クライアントに返された操作のステータスコード (0 は成功、0 以外はエラー)。 このセクションは、単一の gNMI トランザクション内のパスごとに保持されるデータです。たとえば、単一の Get または Set です。 • [サブタイプ (subtype)] : Set RPC の場合、パスごとに要求される特定の操作 (削除、更新、置換) を示します。Get の場合、サブタイプはありません。 • [dtx] : このパスが DTX の「高速」パスで処理されたかどうかを示します。ダッシュ「-」は「いいえ」を意味し、アスタリスク「*」は「はい」を意味します。 • [st] : このパスのステータス。意味は次の

コマンド	説明
	とおりです。 • OK : パスは有効で、インフラによって正常に処理されました。 • ERR: パスが無効であるか、インフラによってエラーが生成されました • -- : パスはまだ処理されていません。有効な場合と無効な場合がありますが、まだインフラに送信されていません。 • [path] : パス

show grpc gnmi service statistics の例

```

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051

Cert notBefore : Mar 13 19:05:24 2020 GMT
Cert notAfter  : Nov 20 19:05:24 2033 GMT

Max concurrent calls : 8
Listen calls : 1
Active calls : 0

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

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

Max gNMI::Get concurrent : 5
Max grpc message size : 8388608
gNMI Synchronous calls : 74
gNMI Synchronous errors : 0
gNMI Adapter errors : 0
gNMI Dtx errors : 0

```

show grpc gnmi rpc summary の例

```

=====
gRPC Endpoint
=====

Vrf : management
Server address : [::]:50051

Cert notBefore : Mar 31 20:55:02 2020 GMT
Cert notAfter  : Apr  1 20:55:02 2020 GMT

```

```

Capability rpcs      : 1
Capability errors    : 0
Get rpcs             : 53
Get errors           : 19
Set rpcs             : 23
Set errors           : 8
Resource Exhausted   : 0
Option Unsupported   : 6
Invalid Argument     : 18
Operation Aborted    : 1
Internal Error       : 2
Unknown Error        : 0

```

RPC Type	State	Last Activity	Cnt Req	Cnt Resp	Client
Subscribe	Listen	04/01 07:39:21	0	0	

show grpc gnmi transactions の例

```

=====
gRPC Endpoint
=====

```

```

Vrf          : management
Server address : [::]:50051

```

```

Cert notBefore : Mar 31 20:55:02 2020 GMT
Cert notAfter  : Apr  1 20:55:02 2020 GMT

```

RPC	DataType	Session	Time In	Duration(ms)	Status
Set	-	2361443608	04/01 07:43:49	173	0
subtype: dtx:	st: path:				
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo789]		
Set	-	2293989720	04/01 07:43:45	183	0
subtype: dtx:	st: path:				
Replace	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo6]		
Set	-	2297110560	04/01 07:43:41	184	0
subtype: dtx:	st: path:				
Update	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo7]		
Set	-	0	04/01 07:43:39	0	10
Set	-	3445444384	04/01 07:43:33	3259	0
subtype: dtx:	st: path:				
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo789]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo790]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo791]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo792]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo793]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo794]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo795]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo796]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo797]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo798]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo799]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo800]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo801]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo802]		
Delete	-	OK	/System/intf-items/lb-items/LbRtdIf-list[id=lo803]		

```

Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo804]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo805]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo806]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo807]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo808]

Set - 2297474560 04/01 07:43:26 186 0
subtype: dtx: st: path:
Update - OK /System/ipv4-items/inst-items/dom-items/Dom-list[name=foo]/rt-
items/Route-list[prefix=0.0.0.0/0]/nh-items/Nexthop-list[nhAddr=192.168.1.1/32][n
hVrf=foo][nhIf=unspecified]/tag

Set - 2294408864 04/01 07:43:17 176 13
subtype: dtx: st: path:
Delete - ERR /System/intf-items/lb-items/LbRtdIf-list/descr

Set - 0 04/01 07:43:11 0 3
subtype: dtx: st: path:
Update - -- /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Update - ERR /system/processes

Set - 2464255200 04/01 07:43:05 708 0
subtype: dtx: st: path:
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo2]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo777]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo778]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo779]
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo780]
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo5]/descr
Update - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr
Update - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo4]/descr
Update - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo5]/descr

Set - 3491213208 04/01 07:42:58 14 0
subtype: dtx: st: path:
Replace - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/descr

Set - 3551604840 04/01 07:42:54 35 0
subtype: dtx: st: path:
Delete - OK /System/intf-items/lb-items/LbRtdIf-list[id=lo1]

Set - 2362201592 04/01 07:42:52 13 13
subtype: dtx: st: path:
Delete - ERR /System/intf-items/lb-items/LbRtdIf-list[id=lo3]/lbrtdif-items
/operSt

Set - 0 04/01 07:42:47 0 3
subtype: dtx: st: path:
Delete - ERR /System/*

Set - 2464158360 04/01 07:42:46 172 3
subtype: dtx: st: path:
Delete - ERR /system/processes/shabang

Set - 2295440864 04/01 07:42:46 139 3
subtype: dtx: st: path:
Delete - ERR /System/invalid/path

```

```

Set          -          3495739048      04/01 07:42:44      10          0

Get          ALL          3444580832      04/01 07:42:40      3          0
subtype: dtx: st: path:
-          -          OK /System/bgp-items/inst-items/disPolBatch

Get          ALL          0              04/01 07:42:36      0          3
subtype: dtx: st: path:
-          -          -- /system/processes/process[pid=1]

Get          ALL          3495870472      04/01 07:42:36      2          0
subtype: dtx: st: path:
-          *          OK /system/processes/process[pid=1]

Get          ALL          2304485008      04/01 07:42:36      33         0
subtype: dtx: st: path:
-          *          OK /system/processes

Get          ALL          2464159088      04/01 07:42:36      251        0
subtype: dtx: st: path:
-          -          OK /system

Get          ALL          2293232352      04/01 07:42:35      258        0
subtype: dtx: st: path:
-          -          OK /system

Get          ALL          0              04/01 07:42:33      0          12
subtype: dtx: st: path:
-          -          -- /intf-items

```

クライアント

gNMIサブスクリプションには、使用可能なクライアントがいくつかあります。このようなクライアントの1つはhttps://github.com/influxdata/telegraf/tree/master/plugins/inputs/cisco_telemetry_gnmiにあります。

DME サブスクリプションの例：PROTO エンコーディング

```

gnmi-console --host >iip> --port 50051 -u <user> -p <pass> --tls --
operation=Subscribe --rpc /root/gnmi-console/testing_bl/once/61_subscribe_bgp_dme_gpb.json

[Subscribe]-----
### Reading from file ' /root/gnmi-console/testing_bl/once/61_subscribe_bgp_dme_gpb.json
,
Wed Jun 26 11:49:17 2019
### Generating request : 1 -----
### Comment : ONCE request
### Delay : 2 sec(s) ...
### Delay : 2 sec(s) DONE
subscribe {
  subscription {
    path {
      origin: "DME"
      elem {
        name: "sys"

```

```

    }
    elem {
      name: "bgp"
    }
  }
  mode: SAMPLE
}
mode: ONCE
use_models {
  name: "DME"
  organization: "Cisco Systems, Inc."
  version: "1.0.0"
}
encoding: PROTO
}
Wed Jun 26 11:49:19 2019
Received response 1 -----
update {
  timestamp: 1561574967761
  prefix {
    elem {
      name: "sys"
    }
    elem {
      name: "bgp"
    }
  }
  update {
    path {
      elem {
      }
      elem {
        name: "version_str"
      }
    }
    val {
      string_val: "1.0.0"
    }
  }
  update {
    path {
      elem {
      }
      elem {
        name: "node_id_str"
      }
    }
    val {
      string_val: "n9k-tm2"
    }
  }
  update {
    path {
      elem {
      }
      elem {
        name: "encoding_path"
      }
    }
    val {
      string_val: "sys/bgp"
    }
  }
  update {

```

```

path {
  elem {
  }
  elem {
    /Received -----
Wed Jun 26 11:49:19 2019
Received response 2 -----
sync_response: true
/Received -----
(_gnmi) [root@tm-ucs-1 gnmi-console]#

```

機能

機能について

Capabilities RPC は、gNMI サービスの機能のリストを返します。RPC 要求に対する応答メッセージには、gNMI サービスのバージョン、バージョン管理されたデータモデル、およびサーバーでサポートされているデータ エンコーディングが含まれます。

機能に関する注意事項と制限事項

次は機能に関する注意事項と制限事項です。

- Cisco NX-OS リリース 9.3(3) 以降、機能は OpenConfig モデルをサポートします。
- サポートされるプラットフォームの詳細については、『Nexus Switch Platform Matrix』を参照してください。 <https://www.cisco.com/c/dam/en/us/td/docs/Website/datacenter/platform/platform.html>
- gNMI 機能は、gNMI サービスのオプションとしてサブスクライブと機能をサポートします。
- この機能は、JSON および gnmi.proto エンコーディングをサポートします。この機能は、protobuf.any エンコーディングをサポートしていません。
- 各 gNMI メッセージの最大サイズは 12 MB です。収集されたデータの量が最大値 12 MB を超えると、収集されたデータはドロップされます。

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

- 同じサブスクリプション要求内のすべてのパスのサンプル間隔は同じである必要があります。同じパスで異なるサンプル間隔が必要な場合は、複数のサブスクリプションを作成します。
- この機能はサブスクリプション要求の単一パス プレフィックスをサポートしていませんが、サブスクリプションには空のプレフィックス フィールドを含めることができます。

- この機能は、Cisco DME およびデバイス YANG データモデルをサポートします。
- gNMI をサポートする gRPC プロセスは HIGH_PRIO cgroup を使用します。これにより、CPU 使用率が CPU の 75% に、メモリが 1.5 GB に制限されます。
- **show grpc gnmi** コマンドには、次の考慮事項があります。
 - このリリースでは、コマンドは XML 化されていません。
 - gRPC エージェントは、呼び出しが終了した後、最大 1 時間 gNMI 呼び出しを保持します。
 - 呼び出しの合計数が 2000 を超えると、gRPC エージェントは内部クリーンアップルーチンに基づいて終了した呼び出しを消去します。

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

gRPC 機能には、Cisco Nexus 9000 スイッチごとに合計 2 つの gRPC サーバー用のデフォルト VRF が含まれています。VRF ごとに 1 つの gRPC サーバーを実行することも、管理 VRF で 1 つの gRPC サーバーのみを実行することもできます。デフォルト VRF で gRPC をサポートすると、かなり部分のトラフィック負荷が望ましくないものである、管理 VRF からの gRPC コールの処理を柔軟にオフロードできます。

2 台の gRPC サーバーを構成する場合は、次の点に注意してください。

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

機能のクライアント出力の例

この例では、すべての OpenConfig モデル RPM がスイッチにインストールされています。

次に、機能のクライアント出力の例を示します。

```
hostname user$ ./gnmi_cli -a 172.19.193.166:50051 -ca_cert ./grpc.pem -insecure
-capabilities
supported_models: <
  name: "Cisco-NX-OS-device"
  organization: "Cisco Systems, Inc."
  version: "2019-11-13"
>
supported_models: <
  name: "openconfig-acl"
  organization: "OpenConfig working group"
  version: "1.0.0"
>
supported_models: <
```

```

    name: "openconfig-bgp-policy"
    organization: "OpenConfig working group"
    version: "4.0.1"
  >
  supported_models: <
    name: "openconfig-interfaces"
    organization: "OpenConfig working group"
    version: "2.0.0"
  >
  supported_models: <
    name: "openconfig-if-aggregate"
    organization: "OpenConfig working group"
    version: "2.0.0"
  >
  supported_models: <
    name: "openconfig-if-ethernet"
    organization: "OpenConfig working group"
    version: "2.0.0"
  >
  supported_models: <
    name: "openconfig-if-ip"
    organization: "OpenConfig working group"
    version: "2.3.0"
  >
  supported_models: <
    name: "openconfig-if-ip-ext"
    organization: "OpenConfig working group"
    version: "2.3.0"
  >
  supported_models: <
    name: "openconfig-lacp"
    organization: "OpenConfig working group"
    version: "1.0.2"
  >
  supported_models: <
    name: "openconfig-lldp"
    organization: "OpenConfig working group"
    version: "0.2.1"
  >
  supported_models: <
    name: "openconfig-network-instance"
    organization: "OpenConfig working group"
    version: "0.11.1"
  >
  supported_models: <
    name: "openconfig-network-instance-policy"
    organization: "OpenConfig working group"
    version: "0.1.1"
  >
  supported_models: <
    name: "openconfig-ospf-policy"
    organization: "OpenConfig working group"
    version: "0.1.1"
  >
  supported_models: <
    name: "openconfig-platform"
    organization: "OpenConfig working group"
    version: "0.12.2"
  >
  supported_models: <
    name: "openconfig-platform-cpu"
    organization: "OpenConfig working group"
    version: "0.1.1"
  >

```

```
supported_models: <
  name: "openconfig-platform-fan"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-platform-linecard"
  organization: "OpenConfig working group"
  version: "0.1.1"
>
supported_models: <
  name: "openconfig-platform-port"
  organization: "OpenConfig working group"
  version: "0.3.2"
>
supported_models: <
  name: "openconfig-platform-psu"
  organization: "OpenConfig working group"
  version: "0.2.1"
>
supported_models: <
  name: "openconfig-platform-transceiver"
  organization: "OpenConfig working group"
  version: "0.7.0"
>
supported_models: <
  name: "openconfig-relay-agent"
  organization: "OpenConfig working group"
  version: "0.1.0"
>
supported_models: <
  name: "openconfig-routing-policy"
  organization: "OpenConfig working group"
  version: "2.0.1"
>
supported_models: <
  name: "openconfig-spanning-tree"
  organization: "OpenConfig working group"
  version: "0.2.0"
>
supported_models: <
  name: "openconfig-system"
  organization: "OpenConfig working group"
  version: "0.3.0"
>
supported_models: <
  name: "openconfig-telemetry"
  organization: "OpenConfig working group"
  version: "0.5.1"
>
supported_models: <
  name: "openconfig-vlan"
  organization: "OpenConfig working group"
  version: "3.0.2"
>
supported_models: <
  name: "DME"
  organization: "Cisco Systems, Inc."
>
supported_models: <
  name: "Cisco-NX-OS-Syslog-oper"
  organization: "Cisco Systems, Inc."
  version: "2019-08-15"
>
```

```
supported_encodings: JSON
supported_encodings: PROTO
gNMI_version: "0.5.0"

hostname user$
```

結果

Get について

Get RPC の目的は、クライアントがデバイスからデータツリーのスナップショットを取得できるようにすることです。1 つの要求で複数のパスを要求できます。gNMI パス規則に従って、XPATH の簡易形式である gNMI のスキーマ パス エンコーディング規則がパスに使用されます。<https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-path-conventions.md>

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

Get に関する注意事項と制限事項

次に、Get および Set に関する注意事項と制限事項を示します。

- `GetRequest.encoding` は JSON のみをサポートします。
- `GetRequest.type` の場合、`DataType CONFIG` と `STATE` のみが YANG で直接の相関関係と式を持ちます。OPERATIONAL はサポートされていません。
- 1 つの要求に OpenConfig (OC) YANG パスとデバイス YANG パスの両方を含めることはできません。要求には、OC YANG パスまたはデバイス YANG パスのみを含める必要があります。両方を含めることはできません。
- ルートパス（「/」：すべてのモデルのすべて）の `GetRequest` は許可されていません。
- デバイスモデルの最上位レベル（「/System」）の `GetRequest` は許可されていません。
- gNMI Get はすべてのデフォルト値を返します（RFC 6243 [4] の report-all モードを参照）。
- `Subscribe` は、モデル `Cisco-NX-OS-syslog-oper` をサポートします。
- `Get` はモデル `Cisco-NX-OS-syslog-oper` をサポートしていません。
- パス `/system` からのクエリは、パス `/system/processes` からのデータを返しません。
openconfig-procmon データのクエリには、特定のパス `/system/processes` を使用する必要があります。
- 次のオプション項目はサポートされていません。
 - パスのプレフィックス

- パスのエイリアス
 - パス内のワイルドカード
- 1 つの `GetRequest` には最大 10 のパスを含めることができます。
 - `GetResponse` で返される値フィールドのサイズが 12 MB を超える場合、システムはエラーステータス `grpc::RESOURCE_EXHAUSTED` を返します。
 - 最大 gRPC 受信バッファサイズは 8 MB に設定されます。
 - `Get` の同時セッションの合計数は 5 に制限されています。
 - 大規模な構成がスイッチに適用されているときに `Get` 操作を実行すると、gRPC プロセスが使用可能なすべてのメモリを消費する可能性があります。メモリ枯渇状態が発生すると、次の syslog が生成されます。

MTX-API: The memory usage is reaching the max memory resource limit (3072) MB

この条件が複数回連続して発生すると、次の syslog が生成されます。

The process has become unstable and the feature should be restarted.

この時点で gRPC 機能を再起動して、gNMI トランザクションの通常の処理を続行することをお勧めします。

- `Get` と `Set` の同時セッションの合計数は、現在構成されている gNMI の同時最大数から 1 を引いたものです。たとえば、`gnmi` の同時呼び出しが 16 に構成されている場合、`Get` および `Set` の合計同時セッションの最大数は 15 になります。
- 大規模な構成がスイッチに適用されているときに `Get` 操作を実行すると、gRPC プロセスが要求を処理できなくなる可能性があります。その時点で、次のエラーが返されます。

There is insufficient memory available on the device to process the subscription.

設定

Set について

Set RPC は、デバイスの構成を変更するためにクライアントによって使用されます。デバイスデータに適用できる操作は削除、置換、更新で、順番を付けて行われます。単一の Set 要求のすべての操作はトランザクションとして扱われます。つまり、すべての操作が成功しなかった場合は、デバイスが元の状態にロールバックされます。Set 操作は、`SetRequest` で指定された順序で適用されます。パスが複数回指定されている場合、互いを上書きすることになったとしても、変更が適用されます。データの最終状態は、トランザクションの最終操作によって実現されます。`SetRequest::delete`、`replace`、`update` フィールドで指定されたすべてのパスは CONFIG データパスであり、クライアントによって書き込み可能であると想定されています。

Set 操作の詳細については、gNMI 仕様、
<https://github.com/openconfig/reference/blob/1cf43d2146f9ba70abb7f04f6b0f6eaa504ceff05/tpc/gnmi/gnmi-specification.md>
 の「Modifying State」のセクションを参照してください。

Set に関する注意事項と制限事項

次に、Set のガイドラインと制限事項を示します。

- SetRequest.encoding は JSON のみをサポートします。
- 1 つの要求に OpenConfig (OC) YANG パスとデバイス YANG パスの両方を含めることはできません。要求には、OC YANG パスまたはデバイス YANG パスのみを含める必要があります。両方を含めることはできません。
- Subscribe は、モデル Cisco-NX-OS-syslog-oper をサポートします。
- パス /system からのクエリは、パス /system/processes からのデータを返しません。
openconfig-procmon データのクエリには、特定のパス /system/processes を使用する必要があります。
- 次のオプション項目はサポートされていません。
 - パスのプレフィックス
 - パスのエイリアス
 - パス内のワイルドカード
- 1 つの SetRequest には最大 20 のパスを含めることができます。
- 最大 gRPC 受信バッファサイズは 8 MB に設定されます。
- Get と Set の同時セッションの合計数は、現在構成されている gNMI の同時最大数から 1 を引いたものです。たとえば、gNMI 同時コールが 16 に設定されている場合、Get および Set の合計同時セッションの最大数は 15 になります。
- Set::Delete RPC の場合、操作対象の設定が大きすぎる可能性がある場合、MTX ログメッセージが警告します。

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

登録

サブスクリプションに関する注意事項と制限事項

サブスクリプションに関する注意事項と制限事項は次のとおりです。

- CLIを使用してルーティングポリシー **prefix-list** を構成し、ルーティングポリシー OpenConfig モデルの gNMI サブスクリプションを要求しても、サポートされません。たとえば、次のような既存の CLI 設定を使用して OpenConfig ルーティングポリシーをサブスクリプトしようすると、OpenConfig モデルの現在の実装のため空の値が返されます。

```
ip prefix-list bgp_v4_drop seq 5 deny 125.2.0.0/16 le 32
ipv6 prefix-list bgp_v6_drop seq 5 deny cafe:125:2::/48 le 128
Using the example paths,
openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v4_drop]/config
openconfig-routing-policy:/routing-policy/defined-sets/prefix-sets/prefix-set[name=bgp_v6_drop]/config
```

- Cisco NX-OS リリース 9.3(3)以降、サブスクリプトは OpenConfig モデルをサポートします。
- サポートされるプラットフォームの詳細については、[Nexus Switch Platform Matrix](#) を参照してください。
- gNMI 機能は、サブスクリプトと機能の両 RPC をサポートします。
- この機能は、JSON および gnmi.proto エンコーディングをサポートします。この機能は、protobuf.any エンコーディングをサポートしていません。
- 各 gNMI メッセージの最大サイズは 12 MB です。収集されたデータの量が最大値 12 MB を超えると、収集されたデータはドロップされます。

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

- 同じサブスクリプション要求内のすべてのパスのサンプル間隔は同じである必要があります。同じパスで異なるサンプル間隔が必要な場合は、複数のサブスクリプションを作成します。
- この機能はサブスクリプション要求の単一パス プレフィックスをサポートしていませんが、サブスクリプションには空のプレフィックス フィールドを含めることができます。
- この機能は、Cisco DME およびデバイス YANG データモデルをサポートします。
- gNMI をサポートする gRPC プロセスは HIGH_PRIO cgroup を使用します。これにより、CPU 使用率が CPU の 75% に、メモリが 1.5 GB に制限されます。
- **show grpc gnmi** コマンドには、次の考慮事項があります。
 - このリリースでは、コマンドは XML 化されていません。
 - gRPC エージェントは、呼び出しが終了した後、最大 1 時間 gNMI 呼び出しを保持します。
 - 呼び出しの合計数が 2000 を超えると、gRPC エージェントは内部クリーンアップルーチンに基づいて終了した呼び出しを消去します。

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

gRPC 機能には、Cisco Nexus 9000 スイッチごとに合計 2 つの gRPC サーバー用のデフォルト VRF が含まれています。VRF ごとに 1 つの gRPC サーバーを実行することも、管理 VRF で 1 つの gRPC サーバーのみを実行することもできます。デフォルト VRF で gRPC をサポートすると、かなり部分のトラフィック負荷が望ましくないものである、管理 VRF からの gRPC コールの処理を柔軟にオフロードできます。

2 台の gRPC サーバーを構成する場合は、次の点に注意してください。

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

gNMI ペイロード

gNMI は、特定のペイロード形式を使用してサブスクライブします。

- DME ストリーム
- YANG ストリーム

サブスクライブ操作は、次のモードでサポートされています。

- ONCE : データを 1 回サブスクライブして受信し、セッションを閉じます。
- POLL : サブスクライブしてセッションを開いたままにし、クライアントはデータが必要になるたびにポーリング要求を送信します。
- STREAM : 特定の頻度でデータを登録および受信します。ペイロードは、ナノ秒（1 秒 = 1000000000）の値を受け入れます。
- ON_CHANGE : サブスクライブしてスナップショットを受信し、ツリーで何かに変更された場合にのみデータを受信します。

設定モード :

- 各モードには、内部サブと外部サブの 2 つの設定が必要です。
- ONCE : サンプル、1 回
- POLL : SAMPLE、POLL
- ストリーム : サンプル、ストリーム
- ON_CHANGE : ON_CHANGE、STREAM

Origin

- DME : DME モデルへの登録
- デバイス : YANG モデルへの登録

名前

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

エンコーディング

- JSON = ストリームは JSON 形式で送信されます。
- PROTO = ストリームは protobuf.any 形式で送信されます。

DME ストリームの gNMI ペイロードの例



(注) クライアントごとに独自の入力形式があります。

```
{
  "SubscribeRequest":
  [
    {
      "_comment" : "ONCE request",
      "_delay" : 2,
      "subscribe":
      {
        "subscription":
        [
          {
            "_comment" : "1st subscription path",
            "path":
            {
              "origin": "DME",
              "elem":
              [
                {
                  "name": "sys"
                },
                {
                  "name": "bgp"
                }
              ]
            },
            "mode": "SAMPLE"
          }
        ],
        "mode": "ONCE",
        "allow_aggregation" : false,
        "use_models":
        [
          {
            "_comment" : "1st module",
            "name": "DME",
```

```

        "organization": "Cisco Systems, Inc.",
        "version": "1.0.0"
    },
    "encoding": "JSON"
}
]
}

```

gNMI ペイロード YANG ストリームの例

```

{
  "SubscribeRequest":
  [
    {
      "_comment" : "ONCE request",
      "_delay" : 2,
      "subscribe":
      {
        "subscription":
        [
          {
            "_comment" : "1st subscription path",
            "path":
            {
              "origin": "device",
              "elem":
              [
                {
                  "name": "System"
                },
                {
                  "name": "bgp-items"
                }
              ]
            },
            "mode": "SAMPLE"
          }
        ],
        "mode": "ONCE",
        "allow_aggregation" : false,
        "use_models":
        [
          {
            "_comment" : "1st module",
            "name": "Cisco-NX-OS-device",
            "organization": "Cisco Systems, Inc.",
            "version": "0.0.0"
          }
        ],
        "encoding": "JSON"
      }
    }
  ]
}

```

ストリーミング Syslog

gNMI のストリーミング Syslog について

gNMI Subscribe は、gNMI Subscribe 要求に従って構造化データをプッシュすることで、システムで何が起きているのかをリアルタイムで表示する、ネットワークをモニターする新しい方法です。

Cisco NX-OS リリース 9.3(3) 以降では、gNMI サブスクライブ機能の垂ポートが追加されました。

gNMI Subscribe サポートの詳細

- Syslog-oper モデル ストリーミング
 - stream_on_change

この機能は、8 GB 以上のメモリを搭載した Cisco Nexus 9000 シリーズ スイッチに適用されます。

ストリーミング Syslog に関する注意事項と制限事項：gNMI

ストリーミング Syslog に関する注意事項と制限事項は次のとおりです。

- 無効な syslog はサポートされていません。たとえば、フィルタまたはクエリ条件を持つ syslog です。
- 次のパスだけがサポートされます：
 - Cisco-NX-OS-Syslog-oper:syslog
 - Cisco-NX-OS-Syslog-oper:syslog/messages
- 次のモードはサポートされていません。
 - ストリーム サンプル
 - 投票
- 要求は YANG モデル フォーマットである必要があります。
- 内部アプリケーションを使用することも、独自のアプリケーションを作成することもできます。
- ペイロードはコントローラから送信され、gNMI は応答を送信します。
- エンコーディング フォーマットは JSON と PROTO です。

Syslog ネイティブ YANG モデル

YangModel はここにあります。 <https://github.com/YangModels/yang/tree/master/vendor/cisco/nx/9.3-3>



(注) タイムゾーンフィールドは、が入力された場合にのみ設定されます。 **clock format show-timezone syslog** デフォルトでは設定されていないため、タイムゾーンフィールドは空です。

```
PYANG Tree for Syslog Native Yang Model:
>>> pyang -f tree Cisco-NX-OS-infra-syslog-oper.yang
module: Cisco-NX-OS-syslog-oper
+--ro syslog
+--ro messages
+--ro message* [message-id]
+--ro message-id int32
+--ro node-name? string
+--ro time-stamp? uint64
+--ro time-of-day? string
+--ro time-zone? string
+--ro category? string
+--ro group? string
+--ro message-name? string
+--ro severity? System-message-severity
+--ro text? string
```

サブスクライブ要求の例

次に、サブスクライブ要求の例を示します。

```
{
  "SubscribeRequest":
  [
    {
      "_comment" : "STREAM request",
      "_delay" : 2,
      "subscribe":
      {
        "subscription":
        [
          {
            "_comment" : "1st subscription path",
            "path":
            {
              "origin": "syslog-oper",
              "elem":
              [
                {
                  "name": "syslog"
                },
                {
                  "name": "messages"
                }
              ]
            },
            "mode": "ON_CHANGE"
          }
        ]
      }
    }
  ],
```

```

        "mode": "ON_CHANGE",
        "allow_aggregation" : false,
        "use_models":
        [
            {
                "_comment" : "1st module",
                "name": "Cisco-NX-OS-Syslog-oper",
                "organization": "Cisco Systems, Inc.",
                "version": "0.0.0"
            }
        ],
        "encoding": "JSON"
    }
}
]
}

```

PROTO 出力の例

これは PROTO 出力のサンプルです。

```
#####
```

```
[Subscribe]-----
```

```
### Reading from file ' /root/gnmi-console/testing_b1/stream_on_change/OC_SYSLOG.json '
```

```
Sat Aug 24 14:38:06 2019
```

```
### Generating request : 1 -----
```

```
### Comment : STREAM request
```

```
### Delay : 2 sec(s) ...
```

```
### Delay : 2 sec(s) DONE
```

```
subscribe {
```

```
subscription {
```

```
path {
```

```
origin: "syslog-oper"
```

```
elem {
```

```
name: "syslog"
```

```
}
```

```
elem {
```

```
name: "messages"
```

```
}
```

```
}
```

```
mode: ON_CHANGE
```

```
}
```

```

use_models {
  name: "Cisco-NX-OS-Syslog-oper"
  organization: "Cisco Systems, Inc."
  version: "0.0.0"
}

encoding: PROTO

Thu Nov 21 14:26:41 2019
Received response 3 -----
update {
  timestamp: 1574375201665688000
  prefix {
    origin: "Syslog-oper"
    elem {
      name: "syslog"
    }
    elem {
      name: "messages"
    }
  }
  update {
    path {
      elem {
        name: "message-id"
      }
    }
  }
  val {
    uint_val: 529
  }
  update {
    path {
      elem {
        name: "node-name"
      }
    }
  }
  val {
    string_val: "task-n9k-1"
  }
  update {
    path {
      elem {
        name: "message-name"
      }
    }
  }
  val {
    string_val: "VSHD_SYSLOG_CONFIG_I"
  }
  update {
    path {
      elem {
        name: "text"
      }
    }
  }
}

```

```
val {
  string_val: "Configured from vty by admin on console0"
}
update {
  path {
    elem {
      name: "group"
    }
  }
}
val {
  string_val: "VSHD"
}
update {
  path {
    elem {
      name: "category"
    }
  }
}
val {
  string_val: "VSHD"
}
update {
  path {
    elem {
      name: "time-of-day"
    }
  }
}
val {
  string_val: "Nov 21 2019 14:26:40"
}
update {
  path {
    elem {
      name: "time-zone"
    }
  }
}
val {
  string_val: ""
}
update {
  path {
    elem {
      name: "time-stamp"
    }
  }
}
val {
  uint_val: 1574375200000
}
update {
  path {
    elem {
      name: "severity"
    }
  }
}
val {
  uint_val: 5
}
}
```

```

}

/Received -----
.

```

JSON 出力の例

これは JSON 出力の例です。

```

[Subscribe]-----
### Reading from file ' testing_bl/stream_on_change/OC_SYSLOG.json '

Tue Nov 26 11:47:00 2019
### Generating request : 1 -----
### Comment : STREAM request
### Delay : 2 sec(s) ...
### Delay : 2 sec(s) DONE
subscribe {
  subscription {
    path {
      origin: "syslog-oper"
      elem {
        name: "syslog"
      }
      elem {
        name: "messages"
      }
    }
    mode: ON_CHANGE
  }
  use_models {
    name: "Cisco-NX-OS-Syslog-oper"
    organization: "Cisco Systems, Inc."
    version: "0.0.0"
  }
}

Tue Nov 26 11:47:15 2019
Received response 5 -----
update {
  timestamp: 1574797636002053000
  prefix {
  }
  update {
    path {
      origin: "Syslog-oper"
      elem {
        name: "syslog"
      }
    }
    val {
      json_val: "[ { \"messages\" : [[
        { \"message-id\":657},{\"node-name\": \"task-n9k-1\", \"time-stamp\": \"1574797635000\", \"time-of-day\": \"Nov
        26 2019
        11:47:15\", \"severity\":3, \"message-name\": \"HDR_I2LEN_ERR\", \"category\": \"ARP\", \"group\": \"ARP\", \"text\": \"arp
        [30318] Received packet with incorrect layer 2 address length (8 bytes), Normal pkt
        with S/D MAC: 003a.7d21.d55e ffff.ffff.ffff eff_ifc mgmt0(9), log_ifc mgmt0(9), phy_ifc
        mgmt0(9)\", \"time-zone\": \"\" } ] ] } ] ]"
    }
  }
}

```



```
/Received -----
```

トラブルシューティング

TM トレース ログの収集

```
1. tmtrace.bin -f gnmi-logs gnmi-events gnmi-errors following are available
2. Usage:

bash-4.3# tmtrace.bin -d gnmi-events | tail -30 Gives the last 30
}
}
}
[06/21/19 15:58:38.969 PDT f8f 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 124, sync_response:1
[06/21/19 15:58:43.210 PDT f90 3133] [3621780288][tm_ec_yang_data_processor.c:93] TM_EC:
[Y] Data received for 2799743488: 49
{
"cdp-items" : {
"inst-items" : {
"if-items" : {
"If-list" : [
{
"id" : "mgmt0",
"ifstats-items" : {
"v2Sent" : "74",
"validV2Rcvd" : "79"
}
}
}
}
}
}
}
[06/21/19 15:58:43.210 PDT f91 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id: 0,
sub_id_str: 2329, dc_start_time: 0, length: 141, sync_response:1
[06/21/19 15:59:01.341 PDT f92 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/intf-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157935518, length: 3063619, sync_response:0
[06/21/19 15:59:03.933 PDT f93 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/cdp-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157940881, length: 6756, sync_response:0
[06/21/19 15:59:03.940 PDT f94 3133] [3981658944][tm_transport_internal.c:43] dn:
Cisco-NX-OS-device:System/lldp-items, sub_id:
4091, sub_id_str: , dc_start_time: 1561157940912, length: 8466, sync_response:1
bash-4.3#
```

MTX 内部ログの収集

1. Modify the following file with below /opt/mtx/conf/mtxlogger.cfg

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
```

```

    <container name="logging">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="allActive" type="boolean" default="false">true<
/leaf>
    <container name="format">
      <leaf name="content" type="string" default="$DATETIME$
$COMPONENTID$ $TYPE$: $MSG$">$DATETIME$ $COMPONENTID$ $TYPE$
$SRCFID$ @ $SRCLINE$ $FCNINFO$: $MSG$</leaf>
      <container name="componentID">
        <leaf name="enabled" type="boolean" default="true"></leaf>
      </container>
      <container name="dateTime">
        <leaf name="enabled" type="boolean" default="true"></leaf>
        <leaf name="format" type="string" default="%Y%m%d.%H%M%S"><
/leaf>
      </container>
      <container name="fcn">
        <leaf name="enabled" type="boolean" default="true"></leaf>
        <leaf name="format" type="string"
default="$CLASS$: $FCNNAME$ ($ARGS$) @$LINE$"></leaf>
      </container>
    </container>
    <container name="facility">
      <leaf name="info" type="boolean" default="true">true</leaf>
      <leaf name="warning" type="boolean" default="true">true<
/leaf>
      <leaf name="error" type="boolean" default="true">true</leaf>

```

Note: Beginning with Cisco NX-OS Release 9.3(4), the following default configuration is changed from

default true to false. To investigate an issue which requires the debug messages, edit the following configuration and toggle it to true.

```

    <leaf name="debug" type="boolean" default="false">true<
/leaf>
  </container>
  <container name="dest">
    <container name="console">
      <leaf name="enabled" type="boolean" default="false">true<
/leaf>
    </container>
    <container name="file">
      <leaf name="enabled" type="boolean" default="false">true<
/leaf>
    <leaf name="name" type="string" default="mtx-internal.log"><
/leaf>

    <leaf name="location" type="string" default="./mtxlogs">
/volatile</leaf>
    <leaf name="mbytes-rollover" type="uint32" default="10"
>50</leaf>
    <leaf name="hours-rollover" type="uint32" default="24"
>24</leaf>
    <leaf name="startup-rollover" type="boolean" default="
false">true</leaf>
    <leaf name="max-rollover-files" type="uint32" default="10"
>10</leaf>
  </container>
</container>
<list name="logitems" key="id">
  <listitem>
    <leaf name="id" type="string">*</leaf>

```

```

        <leaf name="active" type="boolean" default="false"
>false</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">MTX-EvtMgr</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">TM-ADPT</leaf>
        <leaf name="active" type="boolean" default="true"
>false</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">TM-ADPT-JSON</leaf>
        <leaf name="active" type="boolean" default="true"
>false</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">SYSTEM</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">LIBUTILS</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">MTX-API</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">Model-*</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">Model-Cisco-NX-OS-
device</leaf>
        <leaf name="active" type="boolean" default="true"
>false</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">Model-openconfig-bgp<
/leaf>
        <leaf name="active" type="boolean" default="true"
>false</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">INST-MTX-API</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
    <listitem>
        <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
        <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>

```

```

        </listitem>
      </listitem>
      <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
      <leaf name="active" type="boolean" default="true"
>true</leaf>
    </listitem>
  </list>
</container>
</container>
</config>

```

2. Run "no feature grpc" / "feature grpc"

3. The /volatile directory houses the mtx-internal.log, the log rolls over time so be sure to grab what you need before then.

```

bash-4.3# cd /volatile/
bash-4.3# cd /volatile -al
total 148
drwxrwxrwx 4 root root 340 Jun 21 15:47 .
drwxrwxr-t 64 root network-admin 1600 Jun 21 14:45 ..
-rw-rw-rw- 1 root root 103412 Jun 21 16:14 grpc-internal-log
-rw-r--r-- 1 root root 24 Jun 21 14:44 mtx-internal-19-06-21-14-46-21.log
-rw-r--r-- 1 root root 24 Jun 21 14:46 mtx-internal-19-06-21-14-46-46.log
-rw-r--r-- 1 root root 175 Jun 21 15:11 mtx-internal-19-06-21-15-11-57.log
-rw-r--r-- 1 root root 175 Jun 21 15:12 mtx-internal-19-06-21-15-12-28.log
-rw-r--r-- 1 root root 175 Jun 21 15:13 mtx-internal-19-06-21-15-13-17.log
-rw-r--r-- 1 root root 175 Jun 21 15:13 mtx-internal-19-06-21-15-13-42.log
-rw-r--r-- 1 root root 24 Jun 21 15:13 mtx-internal-19-06-21-15-14-22.log
-rw-r--r-- 1 root root 24 Jun 21 15:14 mtx-internal-19-06-21-15-19-05.log
-rw-r--r-- 1 root root 24 Jun 21 15:19 mtx-internal-19-06-21-15-47-09.log
-rw-r--r-- 1 root root 24 Jun 21 15:47 mtx-internal.log
-rw-rw-rw- 1 root root 355 Jun 21 14:44 netconf-internal-log
-rw-rw-rw- 1 root root 0 Jun 21 14:45 nginx_logflag
drwxrwxrwx 3 root root 60 Jun 21 14:45 uwsgipy
drwxrwxrwx 2 root root 40 Jun 21 14:43 virtual-instance
bash-4.3#.

```



第 31 章

ダイナミック ロガー

- [前提条件](#) (421 ページ)
- [参照先](#) (421 ページ)

前提条件

ダイナミック ロギングを使用する前に、スイッチに次のものが存在することを確認します。

- libmtxlogmgr*.so ライブラリが /opt/mtx/lib/ にインストールされていること。
libmtxlogmgr*.so ライブラリは、mtx_infra RPM の一部です。
- mtx.conf ファイルが /etc/ にあること。

```
[mtxlogger]
config=/opt/mtx/conf/mtxlogger.cfg
```
- mtxlogger.cfg ファイルが /opt/mtx/conf/ にあること。

参照先

コンフィギュレーション ファイルの構造は次のとおりです。

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false"></leaf>
      <leaf name="allActive" type="boolean" default="false"></leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$"></leaf>
      <container name="componentID">
        <leaf name="enabled" type="boolean" default="true"></leaf>
      </container>
      <container name="dateTime">
        <leaf name="enabled" type="boolean" default="true"></leaf>
        <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
      </container>
      <container name="fcn">
        <leaf name="enabled" type="boolean" default="true"></leaf>
      </container>
    </container>
  </container>
</config>
```

```

        <leaf name="format" type="string"
default="$CLASS::$FCNNAME$ ($ARGS$) @$LINE$"></leaf>
    </container>
</container>
    <container name="dest">
        <container name="console">
            <leaf name="enabled" type="boolean" default="false"></leaf>
        </container>
        <container name="file">
            <leaf name="enabled" type="boolean" default="false"></leaf>
            <leaf name="name" type="string" default="mtx-internal.log"></leaf>
            <leaf name="location" type="string" default="./mtxlogs"></leaf>
        <leaf name="mbytes-rollover" type="uint32" default="10"></leaf>
        <leaf name="hours-rollover" type="uint32" default="24"></leaf>
        <leaf name="startup-rollover" type="boolean" default="false"></leaf>
            <leaf name="max-rollover-files" type="uint32" default="10"></leaf>
        </container>
    </container>
    <list name="logitems" key="id">
        <listitem>
            <leaf name="id" type="string"></leaf>
        <leaf name="active" type="boolean" default="true"></leaf>
    </listitem>
    </list>
</container>
</container>
</config>

```

タグは、ログフィルタを定義します。<list><componentID>

次の表では、コンテナとそのリーフの一部について説明します。

表 24: コンテナとリーフの説明

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
logging	すべてのロギングデータタイプが含まれます。	形式 dest ファイル (注) リストタグ 「logitems」も含まれます	enabled : ロギングがオンかオフかを決定するブール値。デフォルトはオフです。 allActive : ロギング用に定義されたすべてのロギング項目をアクティブにするブール値。デフォルトはオフ

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
形式	ログメッセージの形式情報を格納します。	componentID dateTime type fcn	content : ログメッセージに含まれるデータ型をリストする文字列。 内容 : • \$DATETIMES\$: ログメッセージに日付または時刻を含める • \$COMPONENTIDS\$: ログメッセージにコンポーネント名を含めます。 • \$TYPES\$: メッセージタイプ ("","INFO、WARNING、ERROR) を含みます。 • \$SRCFILE\$: ソースファイルの名前を含みます。 • \$SRCLINE\$: ソースファイルの行番号を含める • \$FCNINFO\$ ソースファイルから class::関数名を含めます。 • \$MSG\$: 実際のログメッセージテキストを含めます。

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
componentID	ログが記録されたコンポーネントの名前です。	該当なし	enabled : ログメッセージにコンポーネントIDを含めるかどうかを決定するブール値。デフォルトは「true」です。「false」の値は、ログメッセージに""文字列を返します。
dateTime	ログメッセージの日付または時刻	該当なし	enabled : ログメッセージに日付または時刻の情報を含めるかどうかのブール値。デフォルトはイネーブルです。 format : ログメッセージに含める値の文字列。%y%am%d%H%M%Sの形式。
dest	宛先ロガーの構成設定を保持します。	console : 宛先コンソール。許可されるのは1つだけです。 file : 接続先ファイルです。複数ファイルが許可されます。	該当なし
コンソール	宛先コンソール	該当なし	enabled : コンソールのロギングが有効かどうかを決定するブール値。デフォルトは「false」です。

コンテナ	コンテナの説明	含まれるコンテナ	含まれるリーフと説明
ファイル	宛先ファイルの設定を決定します。	該当なし	enabled : 宛先が有効かどうかを決定するブール値。デフォルトは「false」です。 name : 宛先ログファイルの文字列。デフォルトは「mtx-internal.log」 location : 宛先ファイルパスの文字列。デフォルトは「./mtxlogs」です。 mbytes-rollover : システムが最も古いデータを上書きする前のログファイルの長さを決定する uint32。デフォルトは 10 M バイトです。 hours-rollover : 時間単位のログファイルの長さを決定する uint32。デフォルトは 24 時間です。 startup-rollover : エージェントの起動時または再起動時にログファイルをロールオーバーするかどうかを決定するブール値。デフォルト値は「false」です。 max-rollover-files : ロールオーバーファイルの最大数を決定する uint32。 max-rollover-files の値を超えると、最も古いファイルが削除されます。デフォルト値は 10 です。

例

次に、デフォルトでインストールされている設定を含む設定ファイルを示します。

```
<config name="nxos-device-mgmt">
  <container name="mgmtConf">
    <container name="logging">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="allActive" type="boolean" default="false">false</leaf>
      <container name="format">
        <leaf name="content" type="string" default="$DATETIME$ $COMPONENTID$ $TYPE$:
$MSG$">$DATETIME$ $COMPONENTID$ $TYPE$ $SRCFILE$ @ $SRCLINE$ $FCNINFO$: $MSG$</leaf>
      <container name="componentID">
        <leaf name="enabled" type="boolean" default="true"></leaf>
      </container>
    <container name="dateTime">
      <leaf name="enabled" type="boolean" default="true"></leaf>
      <leaf name="format" type="string" default="%y%m%d.%H%M%S"></leaf>
    </container>
    <container name="fcn">
      <leaf name="enabled" type="boolean" default="true"></leaf>
      <leaf name="format" type="string"
default="$CLASS$: $FCNNAME$ ($ARGS$) @$LINE$"></leaf>
    </container>
  </container>
  <container name="dest">
    <container name="console">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
    </container>
    <container name="file">
      <leaf name="enabled" type="boolean" default="false">true</leaf>
      <leaf name="name" type="string" default="mtx-internal.log"></leaf>
      <leaf name="location" type="string" default="./mtxlogs">/volatile</leaf>
    <leaf name="mbytes-rollover" type="uint32" default="10">50</leaf>
    <leaf name="hours-rollover" type="uint32" default="24">24</leaf>
    <leaf name="startup-rollover" type="boolean" default="false">true</leaf>
    <leaf name="max-rollover-files" type="uint32" default="10">10</leaf>
  </container>
  <list name="logitems" key="id">
    <listitem>
      <leaf name="id" type="string">*</leaf>
    <leaf name="active" type="boolean" default="false">false</leaf>
  </listitem>
    <listitem>
      <leaf name="id" type="string">SYSTEM</leaf>
    <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
    <listitem>
      <leaf name="id" type="string">LIBUTILS</leaf>
    <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
    <listitem>
      <leaf name="id" type="string">MTX-API</leaf>
    <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
    <listitem>
      <leaf name="id" type="string">Model-*</leaf>
    <leaf name="active" type="boolean" default="true">true</leaf>
  </listitem>
    <listitem>
      <leaf name="id" type="string">Model-Cisco-NX-OS-device</leaf>
    <leaf name="active" type="boolean" default="true">false</leaf>
  </listitem>
  </list>
</config>
```

```
<listitem>
  <leaf name="id" type="string">Model-openconfig-bgp</leaf>
</listitem>
<listitem>
  <leaf name="id" type="string">INST-MTX-API</leaf>
</listitem>
<listitem>
  <leaf name="id" type="string">INST-ADAPTER-NC</leaf>
</listitem>
<listitem>
  <leaf name="id" type="string">INST-ADAPTER-RC</leaf>
</listitem>
<listitem>
  <leaf name="id" type="string">INST-ADAPTER-GRPC</leaf>
</listitem>
</list>
</container>
</container>
</config>
```




第 32 章

モデル駆動型テレメトリ

- [テレメトリについて \(429 ページ\)](#)
- [テレメトリのライセンス要件 \(432 ページ\)](#)
- [注意事項と制約事項 \(432 ページ\)](#)
- [CLI を使用したテレメトリの構成 \(438 ページ\)](#)
- [NX-API を使用したテレメトリの構成 \(464 ページ\)](#)
- [クラウドスケール ソフトウェア テレメトリ \(480 ページ\)](#)
- [テレメトリ パス ラベル \(481 ページ\)](#)
- [ネイティブ データ送信元パス \(498 ページ\)](#)
- [ストリーミング Syslog \(512 ページ\)](#)
- [その他の参考資料 \(519 ページ\)](#)

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。)

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

Cisco NX-OS リリース 9.2(1) 以降、テレメトリは IPv6 接続先および IPv4 接続先へのストリーミングをサポートするようになりました。

次のコマンドを使用して、JSON または GPB のデータグラム ソケットを使用してデータをストリーミングするように UDP トランスポートを構成します。

```
destination-group num
  ip address xxx.xxx.xxx.xxx port xxxx protocol UDP encoding {JSON | GPB }
```

IPv4 接続先の例:

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

IPv6 接続先の例:

```
destination-group 100
  ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB
```

UDP テレメトリには次のヘッダーがあります。

```
typedef enum tm_encode_ {
    TM_ENCODE_DUMMY,
    TM_ENCODE_GPB,
    TM_ENCODE_JSON,
    TM_ENCODE_XML,
```

```

    TM_ENCODE_MAX,
} tm_encode_type_t;

typedef struct tm_pak_hdr {
    uint8_t version; /* 1 */
    uint8_t encoding;
    uint16_t msg_size;
    uint8_t secure;
    uint8_t padding;
} __attribute__((packed, aligned(1))) tm_pak_hdr_t;

```

次のいずれかの方法で、ペイロードの最初の6バイトを使用して、UDPを使用してテレメトリ データを処理します。

- 受信側が複数のエンドポイントから異なるタイプのデータを受信することになっている場合は、ヘッダーの情報を読んで、データのデコードに使用するデコーダー（JSON または GPB）を決定します。
- 1 つのデコーダー（JSON または GPB）が必要で、もう 1 つのデコーダーは必要ない場合は、ヘッダーを削除します。



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

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

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

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

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

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、『 <i>Cisco NX-OS Licensing Guide</i> 』を参照してください。

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。

- サポートされるプラットフォームの詳細については、[Nexus Switch Platform Matrix](#) を参照してください。
- データ管理エンジン (DME) ネイティブモデルをサポートする Cisco NX-OS リリースは、テレメトリをサポートします。
- 以下のサポートが実施されています。
 - DME データ収集
 - NX-API データ ソース
 - Google リモート プロシージャ コール (gRPC) トランスポートを介した Google プロトコル バッファ (GPB) エンコーディング
 - HTTP 経由の JSON エンコーディング
- サポートされている最小の送信間隔 (ケイデンス) は、深さが 0 の場合の 5 秒です。0 より大きい深度値の最小ケイデンス値は、ストリーミングされるデータのサイズによって異なります。最小値未満のどのケイデンスでもを構成すると、望ましくないシステム動作が発生する可能性があります。
- テレメトリは、最大 5 つの遠隔管理受信者 (接続先) をサポートします。5 つ以上の遠隔受信者を構成すると、システムが望ましくない動作をする可能性があります。
- テレメトリは、CPU 技術情報の最大 20% を消費する可能性があります。
- SSL 証明書ベースの認証とストリーミングデータの暗号化を構成するには、`certificateSSL cert path hostname"CN"` コマンドで自己署名 SSL 証明書を提供します。

- YANG パスにテレメトリ ケイデンスを設定するためのガイドラインは次のとおりです：
 - YANG ストリーミング コレクションには 1 つのスレッドが必要です。テレメトリに複数の YANG パスが存在する場合は、それぞれが異なる周期で実行して、同時スケジューリングと結果として生じる遅延を防ぐ必要があります。
 - YANG パスのテレメトリ ケイデンスを構成する前に、合計ストリーミング時間を決定し、合計ストリーミング時間よりも大きい値にケイデンスを構成します。「[YANG パスの頻度の設定 \(446 ページ\)](#)」を参照してください。

古いリリースにダウングレードした後の構成コマンド

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

次の例は、この手順を表示しています。

- テレメトリ構成をファイルにコピーします。

```
switch# show running-config | section telemetry
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch# show running-config | section telemetry > telemetry_running_config
switch# show file bootflash:telemetry_running_config
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch#
```

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

```
switch# copy telemetry_running_config running-config echo-commands
switch# config terminal
switch(config)# feature telemetry
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB`
```

```
`switch(conf-tm-dest)# sensor-group 100`
`switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
`switch(conf-tm-sensor)# subscription 600`
`switch(conf-tm-sub)# dst-grp 100`
`switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000`
`switch(conf-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete.
switch#
```

gRPC エラーの動作

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

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

gRPC トランスポートのテレメトリ圧縮

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

次の例では、圧縮を有効にします。

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

次の例は、圧縮が有効になっていることを表示しています。

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

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

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

Session Id: 0
```

```

Connection Stats
Connection Count 0
Last Connected: Never
Disconnect Count 0
Last Disconnected: Never
Transmission Stats
Compression: disabled
Source Interface: loopback1 (1.1.3.4)
Transmit Count: 0
Last TX time: None
Min Tx Time: 0 ms
Max Tx Time: 0 ms
Avg Tx Time: 0 ms
Cur Tx Time: 0 ms
switch2(config-if)#

```

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

```

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

```

gRPC チャンキングのサポート

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

gRPC ユーザーは、gRPC チャンクを行う必要があります。gRPC クライアント側は断片化を行い、gRPC サーバ側はリアセンブルを行います。テレメトリは引き続きメモリにバインドされており、メモリ サイズがテレメトリに許可されている制限である 12 MB を超えると、データが削除される可能性があります。チャンクをサポートするには、[テレメトリ コンポーネントとプロセス \(430 ページ\)](#) で説明されているように、gRPC チャンク用に更新された Cisco の GibLab で入手可能なテレメトリ .proto ファイルを使用します。

チャンク サイズは 64 ～ 4096 バイトです。

次に、NX-API CLI による構成例を表示します。

```

feature telemetry
!
telemetry
 destination-group 1
   ip address 171.68.197.40 port 50051 protocol gRPC encoding GPB
   use-chunking size 4096
 destination-group 2
   ip address 10.155.0.15 port 50001 protocol gRPC encoding GPB

```

```

use-chunking size 64
sensor-group 1
  path sys/intf depth unbounded
sensor-group 2
  path sys/intf depth unbounded
subscription 1
  dst-grp 1
  snsr-grp 1 sample-interval 10000
subscription 2
  dst-grp 2
  snsr-grp 2 sample-interval 15000

```

次に、NX-API REST による構成例を表示します。

```

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

```

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

```

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

```

NX-API センサー パスの制限

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

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

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

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



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

2. フィルタまたはオプションを含めるように **show** コマンドを調整します。

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

サマリーまたはカウンタのみが必要な場合は、**show** コマンド出力全体をダンプすることは避け、データ収集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。

テレメトリの VRF サポート

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

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

次の例では、トランスポート VRF を指定しています。

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

以下は、POST ペイロードとしての **use-vrf** の例です。

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

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、ノード識別子のサポートにより、テレメトリ受信者データの `node_id_str` の識別子を設定または変更できます。

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

次の例は、ノード識別子の構成を表示しています。

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

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

```
Telemetry receiver:
=====
node_id_str: "test-srvr-10"
subscription_id_str: "1"
encoding_path: "sys/ch/psuslot-1/psu"
collection_id: 3896
msg_timestamp: 1559669946501
```

YANG モデルのストリーミングのサポート

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

YANG データモデリング言語の詳細については、[インフラストラクチャの概要（295 ページ）](#)と [RESTConf エージェント（349 ページ）](#) を参照してください。

CLI を使用したテレメトリの構成

NX-OS CLI を使用したテレメトリの構成

次の手順では、ストリーミングテレメトリを有効にし、データストリームの送信元と接続先を構成します。これらの手順には、SSL/TLS 証明書と GPB エンコーディングを有効にして構成するオプションの手順も含まれています。

始める前に

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

手順

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

	コマンドまたはアクション	目的
ステップ 2	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	グローバル構成モードを開始します。
ステップ 3	feature telemetry	ストリーミングテレメトリ機能を有効にします。
ステップ 4	feature nxapi	NX-API を有効にします。
ステップ 5	nxapi use-vrf management 例 : <pre>switch(config)# switch(config)# nxapi use-vrf management switch(config)#</pre>	NX-API 通信に使用する VRF 管理を有効にします。 (注) ACL はネットスタックパケットのみをフィルタリングできるため、10.2(3)F より前のリリースでは次の警告が表示されます。 "警告: 設定された管理 ACL は、HTTP サービスには有効になりません。 iptablesを使用してアクセスを制限してください。" (注) 10.2(3)F 以降、ACL は、管理 vrf に着信する netstack パケットと kstack パケットの両方をフィルタリングできます。次の意味の警告が表示されます: 「警告: 非管理 VRF で設定された ACL は、その VRF の HTTP サービスには有効ではありません。」
ステップ 6	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	ストリーミングテレメトリの構成モードに入ります。
ステップ 7	(任意) certificate certificate_path host_URL 例 : <pre>switch(config-telemetry)# certificate /bootflash/server.key localhost</pre>	既存の SSL/TLS 証明書を使用します。 EOR デバイスの場合、証明書もスタンバイ SUP にコピーする必要があります。

	コマンドまたはアクション	目的
ステップ 8	(任意) トランスポート VRF を指定するか、gRPC トランスポートのテレメトリ圧縮を有効にします。 例 : <pre>switch(config-telemetry) # destination-profile switch(conf-tm-dest-profile) # use-vrf default switch(conf-tm-dest-profile) # use-compression gzip switch(conf-tm-dest-profile) # use-retry size 10 switch(conf-tm-dest-profile) # source-interface loopback1</pre>	• destination-profile コマンドを入力して、デフォルトの接続先プロファイルを指定します。 • 次のコマンドを任意で入力します。 • use-vrf vrf で接続先 VRF を指定します。 • use-compression gzip を使用して、接続先の圧縮方法を指定します。 • use-retry size size を使用して、送信再試行の詳細を指定します。再試行バッファサイズは 10～1500 メガバイトです。 • source-interface interface-name は、構成されたインターフェイスから送信元 IP アドレスを持つ接続先にデータをストリーミングします。 (注) use-vrf コマンドを構成した後、新しい VRF 内に新しい接続先 IP アドレスを構成する必要があります。ただし、接続先を構成解除して再構成することにより、同じ接続先 IP アドレスを再利用できます。このアクションにより、テレメトリ データは新しい VRF でも同じ接続先 IP アドレスにストリーミングされます。
ステップ 9	sensor-group sgrp_id 例 : <pre>switch(config-telemetry) # sensor-group 100 switch(conf-tm-sensor) #</pre>	ID <i>srgp_id</i> 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。

	コマンドまたはアクション	目的
ステップ 10	(任意) data-source <i>data-source-type</i> 例 : <pre>switch(config-telemetry) # data-source NX-API</pre>	データ ソースを選択します。データ ソースとして YANG、DME または NX-API のいずれかを選択します。 (注) DME はデフォルトのデータ ソースです。
ステップ 11	path <i>sensor_path</i> depth <i>unbounded</i> [filter-condition <i>filter</i>] [alias <i>path_alias</i>] 例 : - 次のコマンドは、NX-API ではなく、DME または YANG に適用されます : <pre>switch(conf-tm-sensor) # path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(l2BD.operSt, "down")</pre> 以下の構文を使用し、状態ベースのフィルタリングを使用して、operSt が up から down に変化したときにのみトリガーするようにします。MO が変化しても通知しません。 <pre>switch(conf-tm-sensor) # path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))</pre> UTR 側のパスを区別するには、次の構文を使用します。 <pre>switch(conf-tm-sensor) # path sys/ch/ftslot-1/ft alias ft_1</pre> - 次のコマンドは、DME ではなく、NX-API または YANG に適用されます : <pre>switch(conf-tm-sensor) # path "show interface" depth 0</pre> - 次のコマンドは、デバイス YANG に適用されます。 <pre>switch(conf-tm-sensor) # path Cisco-NX-OS-device:System/top-items/inst-items</pre> - 次のコマンドは、OpenConfig YANG に適用されます。	ここでの無制限とは、出力に子管理対象オブジェクト (MO) を含めることを意味します。したがって、POLL テレメトリストリームの場合、そのパスと EVENT のすべての子 MO が子 MO で行われた変更を取得します。 (注) これは、データ ソース DME パスにのみ適用されます。 センサー グループにセンサーパスを追加します。 - Cisco NX-OS 9.3(5) リリース以降では、キーワードが導入されています。 alias depth 設定では、センサー パスの取得レベルを指定します。 0-32、unbounded の深さ設定がサポートされています。 (注) depth 0 デフォルトの深さです。 NX-API ベースのセンサー パスは、depth 0 のみを使用できます。 イベント収集のパスがサブスクライブされている場合、深さは 0 とバウンドなしのみをサポートします。その他の値は 0 として扱われます。 - オプションの filter-condition パラメータを指定して、イベントベースのサブスクリプション用の特定のフィルタを作成できます。

	コマンドまたはアクション	目的
	<pre>switch(conf-tm-sensor)# path openconfig-bgp:bgp switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items alias bgp_alias</pre> - 次のコマンドは、NX-API に適用されます： <pre>switch(conf-tm-sensor)# path "show interface" depth 0 alias sh_int_alias</pre> - 次のコマンドは、OpenConfig に適用されます。 <pre>switch(conf-tm-sensor)# path openconfig-bgp:bgp alias oc_bgp_alias</pre>	状態ベースのフィルタ処理の場合、フィルタ処理は、状態が変化したときと、指定された状態でイベントが発生したときの両方を返します。つまり、eq(l2Bd.operSt, "down") の DN sys/bd/bd-[vlan] のフィルタ条件は、operSt が変更されたとき、および operSt が down である間に DN のプロパティが変更されたとき（VLAN が動作上 down である間に no shutdown コマンドが発行された場合など）にトリガーされます。 (注) query-condition パラメータ — DME の場合、DN に基づいて、次の構文で MOTL および一時データをフェッチするために query-condition パラメータを指定できます。クエリ条件 「rsp-foreign-subtree=ephemeral」。 - YANG モデルの場合、センサーパスの形式は module_name : YANG_path です。module_name は YANG モデル ファイルの名前です。次に例を示します。 - デバイス YANG の場合： <pre>Cisco-NX-OS-device:System/bgp-items/inst-items</pre> - OpenConfig YANG の場合： <pre>openconfig-bgp:bgp</pre> (注) 、およびパラメータは、現在 YANG ではサポートされていません。 depthfilter-conditionquery-condition openconfig YANG モデルの場合は、に移動して、最新リリースの適切なフォルダに移動します。

	コマンドまたはアクション	目的
		https://github.com/YangModels/yang/tree/master/vendor/cisco/nx 特定のモデルをインストールする代わりに、すべての OpenConfig モデルを含む openconfig-all RPM をインストールできます。パッチ RPM のインストールの詳細については、を参照してください。 #unique_86 次に例を示します。 <pre>install add nx-openconfig-10000-7031-HD81b2_r8000.rpm activate</pre>
ステップ 12	destination-group <i>dgrp_id</i> 例 : <pre>switch(conf-tm-sensor) # destination-group 100 switch(conf-tm-dest) #</pre>	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、 <i>dgrp_id</i> は、数字の ID 値のみをサポートしています。
ステップ 13	(任意) ip address <i>ip_address</i> port <i>port</i> protocol <i>procedural-protocol</i> encoding <i>encoding-protocol</i> 例 : <pre>switch(conf-tm-sensor) # ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB switch(conf-tm-sensor) # ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON switch(conf-tm-sensor) # ip address 171.70.55.69 port 50009 protocol UDP encoding JSON</pre>	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 14	(任意) ipv6 address <i>ipv6_address</i> port <i>port</i> protocol <i>procedural-protocol</i> encoding <i>encoding-protocol</i> 例 : <pre>switch(conf-tm-sensor) # ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB switch(conf-tm-sensor) # ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON switch(conf-tm-sensor) # ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON</pre>	エンコードされたテレメトリデータを受信する IPv6 IP アドレスとポートを指定します。 (注) gRPC はデフォルトのトランスポートプロトコルです。 GPB がデフォルトのエンコーディングです。

	コマンドまたはアクション	目的
ステップ 15	ip_version address ip_address port portnum 例 : - IPv4 の場合 : <pre>switch(conf-tm-dest) # ip address 1.2.3.4 port 50003</pre> - IPv6 の場合 : <pre>switch(conf-tm-dest) # ipv6 address 10:10::1 port 8000</pre>	発信データの宛先プロファイルを作成します。ip_version は、ip (IPv4 の場合) または ipv6 (IPv6 の場合) です。 接続先グループがサブスクリプションにリンクされている場合、テレメトリデータは、このプロファイルで指定されている IP アドレスとポートに送信されます。
ステップ 16	(任意) use-chunking size chunking_size 例 : <pre>switch(conf-tm-dest) # use-chunking size 64</pre>	gRPC チャンクを有効にして、チャンク サイズを 64~4096 バイトに設定します。詳細については、「gRPC チャンクのサポート」セクションを参照してください。
ステップ 17	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 100 switch(conf-tm-sub) #</pre>	ID を持つサブスクリプションノードを作成し、サブスクリプション構成モードを開始します。 現在、sub_id は、数字の ID 値のみをサポートしています。 (注) DN にサブスクライブする場合は、イベントが確実にストリーミングされるように、その DN が REST を使用して DME でサポートされているかどうかを確認します。
ステップ 18	snsr-grp sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 100 sample-interval 15000</pre>	ID sgrp_id のセンサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔 (ミリ秒単位) を設定します。 間隔の値が 0 の場合、イベントベースのサブスクリプションが作成され、テレメトリデータは、指定された MO での変更時にのみ送信されます。0 より大きい間隔値の場合、テレメトリデータが指定された間隔で定期的に送信される頻度に基づ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

	コマンドまたはアクション	目的
		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 19	dst-grp dgrp_id 例 : switch(conf-tm-sub) # dst-grp 100	ID <i>dgrp_id</i> を持つ接続先グループをこのサブスクリプションにリンクします。

YANG パスの頻度の設定

YANG パスの頻度は、合計ストリーミング時間よりも長くする必要があります。合計ストリーミング時間と頻度が正しく構成されていない場合、テレメトリデータの収集にストリーミング間隔よりも長くかかることがあります。この状況では、次のことがわかります。

- テレメトリ データが受信側へのストリーミングよりも速く蓄積されるため、徐々に満たされるキュー。
- 現在の間隔からではない古いテレメトリ データ。

合計ストリーミング時間よりも大きい値に頻度を構成します。

手順

	コマンドまたはアクション	目的																		
ステップ 1	show telemetry control database sensor-groups 例 : switch# show telemetry control database sensor-groups Sensor Group Database size = 2 <table><thead><tr><th>Row ID</th><th>Sensor Group ID</th><th>Sensor Group type</th><th>Sampling interval(ms)</th><th>Linked subscriptions</th><th>SubID</th></tr></thead><tbody><tr><td>1</td><td>2</td><td>Timer</td><td></td><td></td><td></td></tr><tr><td>/YANG</td><td>5000</td><td>/Running</td><td>1</td><td></td><td></td></tr></tbody></table> Collection Time in ms (Cur/Min/Max): 2444/2294/2460 Encoding Time in ms (Cur/Min/Max): 56/55/57 Transport Time in ms (Cur/Min/Max): 0/0/1 Streaming Time in ms (Cur/Min/Max): 2515/2356/28403 Collection Statistics: collection_id_dropped = 0 last_collection_id_dropped = 0 drop count = 0	Row ID	Sensor Group ID	Sensor Group type	Sampling interval(ms)	Linked subscriptions	SubID	1	2	Timer				/YANG	5000	/Running	1			合計ストリーミング時間を計算します。 合計ストリーミング時間は、各センサーグループの個々の現在のストリーミング時間の合計です。個々のストリーミング時間は、ミリ秒単位のストリーミング時間（Cur）に表示されます。この例では、合計ストリーミング時間は2.664 秒（2515 ミリ秒+ 149 ミリ秒）です。 構成された頻度をセンサー グループの合計ストリーミング時間と比較します。 頻度はサンプル間隔で表示されます。この例では、合計ストリーミング時間（2.664 秒）がケイデンス（デフォルトの 5.000 秒）よりも短いため、頻度は正しく構成されています。
Row ID	Sensor Group ID	Sensor Group type	Sampling interval(ms)	Linked subscriptions	SubID															
1	2	Timer																		
/YANG	5000	/Running	1																	

	コマンドまたはアクション	目的
	<pre> 2 1 Timer /YANG 5000 /Running 1 1 Collection Time in ms (Cur/Min/Max): 144/142/1471 Encoding Time in ms (Cur/Min/Max): 0/0/1 Transport Time in ms (Cur/Min/Max): 0/0/0 Streaming Time in ms (Cur/Min/Max): 149/147/23548 Collection Statistics: collection_id_dropped = 0 last_collection_id_dropped = 0 drop_count = 0 switch# telemetry destination-group 1 ip address 192.0.2.1 port 9000 protocol HTTP encoding JSON sensor-group 1 data-source YANG path /Cisco-NX-OS-device:System/procsys-items depth unbounded sensor-group 2 data-source YANG path /Cisco-NX-OS-device:System/intf-items/phys-items depth unbounded subscription 1 dst-grp 1 snsr-grp 1 sample-interval 5000 snsr-grp 2 sample-interval 5000 </pre>	
ステップ 2	sensor group number 例 : switch(config-telemetry) # sensor group 1	合計ストリーミング時間がその頻度以上の 場合、間隔を設定したいセンサーグ ループを入力します。
ステップ 3	subscription number 例 : switch(conf-tm-sensor) # subscription 100	センサー グループのサブスクリプショ ンを編集します。
ステップ 4	snsr-grp number sample-interval milliseconds 例 : switch(conf-tm-sub) # snsr-grp number sample-interval 5000	適切なセンサー グループについて、サ ンプル間隔を合計ストリーミング時間よ りも大きい値に設定します。 この例では、サンプル間隔は 5.000 秒に 設定されています。これは、2.664 秒の 合計ストリーミング時間よりも長いた め、有効です。

	コマンドまたはアクション	目的
ステップ 5	show system resources 例 : <pre>switch# show system resources Load average: 1 minute: 0.38 5 minutes: 0.43 15 minutes: 0.43 Processes: 555 total, 3 running CPU states : 24.17% user, 4.32% kernel, 71.50% idle CPU0 states: 0.00% user, 2.12% kernel, 97.87% idle CPU1 states: 86.00% user, 11.00% kernel, 3.00% idle CPU2 states: 8.08% user, 3.03% kernel, 88.88% idle CPU3 states: 0.00% user, 1.02% kernel, 98.97% idle Memory usage: 16400084K total, 5861652K used, 10538432K free Current memory status: OK</pre>	CPUの使用状況を確認してください。 この例に示すように、CPU ユーザー状態が高い使用率を示している場合、頻度とストリーミング値が正しく構成されていません。この手順を繰り返して、頻度を正しく設定します。

CLI を使用したテレメトリの構成例

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

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

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

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


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

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(config-dest)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

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

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

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

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

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

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
```

```

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

switch(config-telemetry)# sensor-group 200
switch(config-tm-sensor)# path sys/cdp depth 0
switch(config-tm-sensor)# path sys/ipv4 depth 0

switch(config-telemetry)# sensor-group 300
switch(config-tm-sensor)# path sys/fm depth 0
switch(config-tm-sensor)# path sys/bgp depth 0

switch(config-tm-sensor)# destination-group 100
switch(config-tm-dest)# ip address 1.2.3.4 port 50004
switch(config-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)# snsrg 100 sample-interval 7000
switch(conf-tm-sub)# snsrg 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)# snsrg 200 sample-interval 7000
switch(conf-tm-sub)# snsrg 300 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 300
```

この例に示すように、**show running-config telemetry** コマンドを使用してテレメトリ構成を確認できます。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# end
switch# show run telemetry

!Command: show running-config telemetry
!Time: Thu Oct 13 21:10:12 2016

version 7.0(3)I5(1)
feature telemetry

telemetry
destination-group 100
ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
```

この例に示すように、**use-vrf** コマンドと **use-compression gzip** コマンドを使用して、gRPC のトランスポート VRF とテレメトリ データ圧縮を指定できます。

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(conf-tm-dest-profile)# use-vrf default
switch(conf-tm-dest-profile)# use-compression gzip
switch(conf-tm-dest-profile)# sensor-group 1
switch(conf-tm-sensor)# path sys/bgp depth unbounded
switch(conf-tm-sensor)# destination-group 1
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-sub)# snsrg 1 sample-interval 10000
```

テレメトリの構成と統計情報の表示

次の NX-OS CLI **show** コマンドを使用して、テレメトリの構成、統計情報、エラー、およびセッション情報を表示します。

show telemetry yang direct-path cisco-nxos-device

このコマンドは、他のパスよりもパフォーマンスが向上するように直接エンコードされた YANG パスを表示します。

```
switch# show telemetry yang direct-path cisco-nxos-device
1) Cisco-NX-OS-device:System/lldp-items
2) Cisco-NX-OS-device:System/acl-items
3) Cisco-NX-OS-device:System/mac-items
4) Cisco-NX-OS-device:System/intf-items
5) Cisco-NX-OS-device:System/procsys-items/sysload-items
6) Cisco-NX-OS-device:System/ospf-items
7) Cisco-NX-OS-device:System/procsys-items
8) Cisco-NX-OS-device:System/ipqos-items/queuing-items/policy-items/out-items
9) Cisco-NX-OS-device:System/mac-items/static-items
10) Cisco-NX-OS-device:System/ch-items
11) Cisco-NX-OS-device:System/cdp-items
12) Cisco-NX-OS-device:System/bd-items
13) Cisco-NX-OS-device:System/eps-items
14) Cisco-NX-OS-device:System/ipv6-items
```

show telemetry control database

次に、テレメトリの構成を反映している内部データベースのコマンドを表示します。

```
switch# show telemetry control database ?
<CR>
>                                Redirect it to a file
>>                             Redirect it to a file in append mode
destination-groups             Show destination-groups
destinations                   Show destinations
sensor-groups                  Show sensor-groups
sensor-paths                   Show sensor-paths
subscriptions                  Show subscriptions
|                               Pipe command output to filter
```

```
switch# show telemetry control database
```

```
Subscription Database size = 1
```

```
-----
Subscription ID      Data Collector Type
-----
100                  DME NX-API
```

```
Sensor Group Database size = 1
```

```
-----
Sensor Group ID      Sensor Group type      Sampling interval(ms)  Linked subscriptions
-----
100                  Timer                  10000 (Running)       1
```

```
Sensor Path Database size = 1
-----
```

Subscribed Query Filter	Linked Groups	Sec Groups	Retrieve level	Sensor Path
No	1	0	Full	sys/fm

Destination group Database size = 2

Destination Group ID	Refcount
100	1

Destination Database size = 2

Dst IP Addr	Dst Port	Encoding	Transport	Count
192.168.20.111	12345	JSON	HTTP	1
192.168.20.123	50001	GPB	gRPC	1

show telemetry control database sensor-paths

このコマンドは、テレメトリ設定のセンサーパスの詳細を表示します。これには、エンコーディング、収集、トランスポート、およびストリーミングのカウンタが含まれます。

```
switch(conf-tm-sub)# show telemetry control database sensor-paths
```

Sensor Path Database size = 4

Row ID	Subscribed	Linked Groups	Sec Groups	Retrieve level	Path(GroupId) : Query : Filter

1	No	1	0	Full	sys/cdp(1) : NA : NA

GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0

JSON Encoded Data size in bytes (Cur/Min/Max): 65785/65785/65785

Collection Time in ms (Cur/Min/Max): 10/10/55

Encoding Time in ms (Cur/Min/Max): 8/8/9

Transport Time in ms (Cur/Min/Max): 0/0/0

Streaming Time in ms (Cur/Min/Max): 18/18/65

2	No	1	0	Self	show module(2) : NA : NA

GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0

JSON Encoded Data size in bytes (Cur/Min/Max): 1107/1106/1107

Collection Time in ms (Cur/Min/Max): 603/603/802

Encoding Time in ms (Cur/Min/Max): 0/0/0

Transport Time in ms (Cur/Min/Max): 0/0/1

Streaming Time in ms (Cur/Min/Max): 605/605/803

3	No	1	0	Full	sys/bgp(1) : NA : NA

GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0

JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0

Collection Time in ms (Cur/Min/Max): 0/0/44

Encoding Time in ms (Cur/Min/Max): 0/0/0

Transport Time in ms (Cur/Min/Max): 0/0/0

Streaming Time in ms (Cur/Min/Max): 1/1/44

4	No	1	0	Self	show version(2) : NA : NA

GPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0

JSON Encoded Data size in bytes (Cur/Min/Max): 2442/2441/2442

Collection Time in ms (Cur/Min/Max): 1703/1703/1903

Encoding Time in ms (Cur/Min/Max): 0/0/0

```
Transport Time in ms (Cur/Min/Max): 0/0/0
Streaming Time in ms (Cur/Min/Max): 1703/1703/1904
```

```
switch(conf-tm-sub)#
```

show telemetry control stats

このコマンドは、テレメトリの構成についての内部データベースの統計を表示します。

```
switch# show telemetry control stats
show telemetry control stats entered
```

Error Description	Error Count
Chunk allocation failures	0
Sensor path Database chunk creation failures	0
Sensor Group Database chunk creation failures	0
Destination Database chunk creation failures	0
Destination Group Database chunk creation failures	0
Subscription Database chunk creation failures	0
Sensor path Database creation failures	0
Sensor Group Database creation failures	0
Destination Database creation failures	0
Destination Group Database creation failures	0
Subscription Database creation failures	0
Sensor path Database insert failures	0
Sensor Group Database insert failures	0
Destination Database insert failures	0
Destination Group Database insert failures	0
Subscription Database insert failures	0
Sensor path Database delete failures	0
Sensor Group Database delete failures	0
Destination Database delete failures	0
Destination Group Database delete failures	0
Delete Subscription from Subscription Database failures	0
Sensor path delete in use	0
Sensor Group delete in use	0
Destination delete in use	0
Destination Group delete in use	0
Delete destination(in use) failure count	0
Failed to get encode callback	0
Sensor path Sensor Group list creation failures	0
Sensor path prop list creation failures	0
Sensor path sec Sensor path list creation failures	0
Sensor path sec Sensor Group list creation failures	0
Sensor Group Sensor path list creation failures	0
Sensor Group Sensor subs list creation failures	0
Destination Group subs list creation failures	0
Destination Group Destinations list creation failures	0
Destination Destination Groups list creation failures	0
Subscription Sensor Group list creation failures	0
Subscription Destination Groups list creation failures	0
Sensor Group Sensor path list delete failures	0
Sensor Group Subscriptions list delete failures	0
Destination Group Subscriptions list delete failures	0
Destination Group Destinations list delete failures	0
Subscription Sensor Groups list delete failures	0
Subscription Destination Groups list delete failures	0
Destination Destination Groups list delete failures	0
Failed to delete Destination from Destination Group	0
Failed to delete Destination Group from Subscription	0
Failed to delete Sensor Group from Subscription	0

```

Failed to delete Sensor path from Sensor Group          0
Failed to get encode callback                          0
Failed to get transport callback                       0
switch# Destination Database size = 1

```

```

-----
Dst IP Addr      Dst Port   Encoding   Transport   Count
-----
192.168.20.123  50001      GPB        gRPC         1

```

show telemetry data collector brief

このコマンドは、データ収集に関する簡単な統計情報を表示します。

```
switch# show telemetry data collector brief
```

```

-----
Collector Type      Successful Collections   Failed Collections
-----
DME                  143                      0

```

show telemetry data collector details

このコマンドは、すべてのセンサパスの詳細を含む、データ収集に関する詳細な統計情報を表示します。

```
switch# show telemetry data collector details
```

```

-----
Succ Collections    Failed Collections        Sensor Path
-----
150                  0                          sys/fm

```

show telemetry event collector errors

このコマンドは、イベントコレクションに関するエラー統計情報を表示します。

```
switch# show telemetry event collector errors
```

```

-----
Error Description                                         Error Count
-----
APIC-Cookie Generation Failures                         - 0
Authentication Failures                                 - 0
Authentication Refresh Failures                         - 0
Authentication Refresh Timer Start Failures             - 0
Connection Timer Start Failures                         - 0
Connection Attempts                                     - 3
Dme Event Subscription Init Failures                    - 0
Event Data Enqueue Failures                             - 0
Event Subscription Failures                             - 0
Event Subscription Refresh Failures                    - 0
Pending Subscription List Create Failures               - 0
Subscription Hash Table Create Failures                 - 0
Subscription Hash Table Destroy Failures                - 0
Subscription Hash Table Insert Failures                 - 0

```

```
Subscription Hash Table Remove Failures      - 0
Subscription Refresh Timer Start Failures    - 0
Websocket Connect Failures                   - 0
```

show telemetry event collector stats

このコマンドは、すべてのセンサー パスの内訳を含むイベント コレクションに関する統計情報を表示します。

```
switch# show telemetry event collector stats
```

```
-----
Collection Count  Latest Collection Time    Sensor Path
-----
```

show telemetry control pipeline stats

このコマンドは、テレメトリ パイプラインの統計情報を表示します。

```
switch# show telemetry pipeline stats
```

```
Main Statistics:
```

```
Timers:
```

```
Errors:
```

```
Start Fail      =      0
```

```
Data Collector:
```

```
Errors:
```

```
Node Create Fail =      0
```

```
Event Collector:
```

```
Errors:
```

```
Node Create Fail =      0    Node Add Fail      =      0
```

```
Invalid Data     =      0
```

```
Memory:
```

```
Allowed Memory Limit      = 1181116006 bytes
```

```
Occupied Memory           = 93265920 bytes
```

```
Queue Statistics:
```

```
Request Queue:
```

```
High Priority Queue:
```

```
Info:
```

```
Actual Size      =      50    Current Size      =      0
```

```
Max Size         =      0    Full Count       =      0
```

```
Errors:
```

```
Enqueue Error    =      0    Dequeue Error    =      0
```

```
Low Priority Queue:
```

```
Info:
```

```
Actual Size      =      50    Current Size      =      0
```

```
Max Size         =      0    Full Count       =      0
```

```
Errors:
```

```
Enqueue Error    =      0    Dequeue Error    =      0
```

```
Data Queue:
```

```
High Priority Queue:
```

```
Info:
```



```

        Actual Size      =    50    Current Size      =    0
        Max Size         =    0     Full Count         =    0

Errors:
  Enqueue Error         =    0     Dequeue Error        =    0

Low Priority Queue:
Info:
  Actual Size           =    50    Current Size       =    0
  Max Size              =    0     Full Count         =    0

Errors:
  Enqueue Error         =    0     Dequeue Error        =    0

```

show telemetry transport

次に、構成されているすべての転送セッションの例を表示します。

```
switch# show telemetry transport
```

Session Id	IP Address	Port	Encoding	Transport	Status
0	192.168.20.123	50001	GPB	gRPC	Connected

表 25 : show telemetry transport の構文の説明

構文	説明
show	実行中のシステム情報を表示します
telemetry	テレメトリ情報を表示します
トランスポート	テレメトリのトランスポート情報を表示します
session_id	(オプション) セッション ID
stats	(オプション) すべてのテレメトリ統計情報を表示します
errors	(オプション) すべてのテレメトリエラー情報を表示します。
読み取り専用	(オプション)
TABLE_transport_info	(オプション) トランスポート情報
session_idx	(オプション) セッション ID
ip_address	(オプション) トランスポート IP アドレス
port	(オプション) トランスポート ポート
dest_info	(オプション) 接続先情報

構文	説明
<i>encoding_type</i>	(オプション) エンコーディング タイプ
<i>transport_type</i>	(オプション) トランスポート タイプ
<i>transport_status</i>	(オプション) トランスポート ステータス
<i>transport_security_cert_fname</i>	(オプション) トランスポートセキュリティ ファイル名
<i>transport_last_connected</i>	(オプション) 最後に接続されたトランスポート
<i>transport_last_disconnected</i>	(オプション) この接続先構成が最後に削除 された時刻
<i>transport_errors_count</i>	(オプション) トランスポート エラー数
<i>transport_last_tx_error</i>	(オプション) トランスポートの最後の tx エ ラー
<i>transport_statistics</i>	(オプション) トランスポート統計情報
<i>t_session_id</i>	(オプション) トランスポートセッションID
<i>connect_statistics</i>	(オプション) 接続統計情報
<i>connect_count</i>	(オプション) 接続数
<i>last_connected</i>	(オプション) 最終接続のタイムスタンプ
<i>Disconnect_count</i>	(オプション) 切断数
<i>last_disconnected</i>	(オプション) この接続先構成が最後に削除 された時刻
<i>trans_statistics</i>	(オプション) トランスポート統計情報
<i>compression</i>	(オプション) 圧縮ステータス
<i>source_interface_name</i>	(オプション) 送信元インターフェイス名
<i>source_interface_ip</i>	(オプション) 送信元インターフェイス IP
<i>transmit_count</i>	(オプション) 送信数
<i>last_tx_time</i>	(オプション) 最終送信時刻
<i>min_tx_time</i>	(オプション) 最小送信時間
<i>max_tx_time</i>	(オプション) 最大送信時間

構文	説明
<i>avg_tx_time</i>	(オプション) 平均送信時間
<i>cur_tx_time</i>	(オプション) 現在の送信時間
<i>transport_errors</i>	(オプション) トランスポート エラー
<i>connect_errors</i>	(オプション) 接続エラー
<i>connect_errors_count</i>	(オプション) 接続エラー数
<i>trans_errors</i>	(オプション) トランスポート エラー
<i>trans_errors_count</i>	(オプション) トランスポート エラー数
<i>last_tx_error</i>	(オプション) 最後のトランスポートエラー
<i>last_tx_return_code</i>	(オプション) 最後のトランスポート戻りコード
<i>transport_retry_stats</i>	(オプション) 再試行統計情報
<i>ts_event_retry_bytes</i>	(オプション) イベント再試行バッファサイズ
<i>ts_timer_retry_bytes</i>	(オプション) タイマー再試行バッファサイズ
<i>ts_event_retry_size</i>	(オプション) メッセージのイベント再試行回数
<i>ts_timer_retry_size</i>	(オプション) タイマー再試行メッセージ回数
<i>ts_retries_sent</i>	(オプション) 送信された再試行回数
<i>ts_retries_dropped</i>	(オプション) ドロップされた再試行回数
<i>event_retry_bytes</i>	(オプション) イベント再試行バッファサイズ
<i>timer_retry_bytes</i>	(オプション) タイマー再試行バッファサイズ
<i>retries_sent</i>	(オプション) 送信された再試行回数
<i>retries_dropped</i>	(オプション) ドロップされた再試行回数
<i>retry_buffer_size</i>	(オプション) 再試行バッファ サイズ

show telemetry transport <session-id>

次のコマンドでは、特定の転送セッションの詳細なセッション情報が表示されます。

```
switch# show telemetry transport 0

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Fri Sep 02 11:45:57.505 UTC
Last Disconnected:   Never
Tx Error Count:      224
Last Tx Error:       Fri Sep 02 12:23:49.555 UTC
```

```
switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
```

次に、IPv6 エントリの出力例を示します。

```
switch# show telemetry transport 0

Session Id: 0
IP Address:Port [10:10::1]:8000
Transport: GRPC
Status: Idle
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
Event Retry Queue Bytes: 0
Event Retry Queue Size: 0
Timer Retry Queue Bytes: 0
Timer Retry Queue Size: 0
Sent Retry Messages: 0
Dropped Retry Messages: 0
```

show telemetry transport <session-id> stats

次に、特定の転送セッションの詳細のコマンドを示します。

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

show telemetry transport <session-id> stats

次に、特定の転送セッションの詳細のコマンドを示します。

```
Session Id: 0
Transmission Stats
  Compression: disabled
  Source Interface: not set()
  Transmit Count: 319297
  Last TX time: Fri Aug 02 03:51:15.287 UTC
  Min Tx Time: 1 ms
  Max Tx Time: 3117 ms
  Avg Tx Time: 3 ms
  Cur Tx Time: 1 ms
```

show telemetry transport <session-id> errors

次のコマンドでは、特定の転送セッションの詳細なエラーの統計情報が表示されます。

```
switch# show telemetry transport 0 errors
Session Id: 0
Connection Errors
Connection Error Count: 0
Transmission Errors
Tx Error Count: 30
Last Tx Error: Thu Aug 01 04:39:47.083 UTC
Last Tx Return Code: No error
```

show telemetry control databases sensor-paths

これらの次の構成手順により、次の **show telemetry control databases sensor-paths** コマンド出力が得られます。

```
feature telemetry

telemetry
  destination-group 1
    ip address 172.25.238.13 port 50600 protocol gRPC encoding GPB
  sensor-group 1
    path sys/cdp depth unbounded
    path sys/intf depth unbounded
    path sys/mac depth 0
  subscription 1
    dst-grp 1
    snsrr-grp 1 sample-interval 1000
```

コマンド出力。

```
switch# show telemetry control databases sensor-paths

Sensor Path Database size = 3
-----
Row ID      Subscribed Linked Groups  Sec Groups  Retrieve level  Path(GroupID) :
Query : Filter
-----
1           No              1            0             Full          sys/cdp(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 30489/30489/30489
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 6/5/54
```

```

Encoding Time in ms (Cur/Min/Max): 5/5/6
Transport Time in ms (Cur/Min/Max): 1027/55/1045
Streaming Time in ms (Cur/Min/Max): 48402/5/48402

2          No          1          0          Full          sys/intf(1) : N
A : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 539466/539466/539466
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 66/64/114
Encoding Time in ms (Cur/Min/Max): 91/90/92
Transport Time in ms (Cur/Min/Max): 4065/4014/5334
Streaming Time in ms (Cur/Min/Max): 48365/64/48365

3          No          1          0          Self          sys/mac(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 247/247/247
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 1/1/47
Encoding Time in ms (Cur/Min/Max): 1/1/1
Transport Time in ms (Cur/Min/Max): 4/1/6
Streaming Time in ms (Cur/Min/Max): 47369/1/47369

```

テレメトリ ログとトレース情報の表示

ログとトレース情報を表示するには、次の NX-OS CLI コマンドを使用します。

テクニカル サポート テレメトリを表示

この NX-OS CLI コマンドは、テクニカルサポート ログからテレメトリ ログの内容を収集します。この例では、コマンド出力がブートフラッシュのファイルにリダイレクトされます。

```
switch# show tech-support telemetry > bootflash:tmst.log
```

tmtrace.bin

この BASH シェル コマンドは、テレメトリ トレースを収集して出力します。

```

switch# configure terminal
switch(config)# feature bash
switch(config)# run bash
bash-4.2$ tmtrace.bin -d tm-errors
bash-4.2$ tmtrace.bin -d tm-logs
bash-4.2$ tmtrace.bin -d tm-events

```

例：

```

bash-4.2$ tmtrace.bin -d tm-logs
[01/25/17 22:52:24.563 UTC 1 29130] [3944724224][tm_ec_dme_auth.c:59] TM_EC: Authentication
refresh url http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.565 UTC 2 29130] [3944724224][tm_ec_dme_rest_util.c:382] TM_EC:
Performed POST request on http://127.0.0.1/api/aaaRefresh.json
[01/25/17 22:52:24.566 UTC 3 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER:
Starting leaf timer for leaf:0x11e17ea4 time_in_ms:540000
[01/25/17 22:52:45.317 UTC 4 29130] [3944724224][tm_ec_dme_event_subsc.c:790] TM_EC:
Event subscription database size 0

```

```
[01/25/17 22:52:45.317 UTC 5 29130] [3944724224][tm_mgd_timers.c:114] TM_MGD_TIMER:
Starting leaf timer for leaf:0x11e17e3c time_in_ms:50000
bash-4.2#
```



(注) **tm-logs** オプションは冗長であるため、デフォルトでは有効になっていません。

`tmtrace.bin -LD tm-logs` コマンドで **tm-logs** を有効にします。

`tmtrace.bin -LW tm-logs` コマンドを使用して **tm-logs** を無効にします。

show system internal telemetry trace

show system internal telemetry trace [tm-events | tm-errors | tm-logs | all] コマンドは、システムの内部テレメトリ トレース情報を表示します。

```
switch# show system internal telemetry trace all
Telemetry All Traces:
Telemetry Error Traces:
[07/26/17 15:22:29.156 UTC 1 28577] [3960399872][tm_cfg_api.c:367] Not able to destroy
dest profile list for config node rc:-1610612714 reason:Invalid argument
[07/26/17 15:22:44.972 UTC 2 28577] [3960399872][tm_stream.c:248] No subscriptions for
destination group 1
[07/26/17 15:22:49.463 UTC 3 28577] [3960399872][tm_stream.c:576] TM_STREAM: Subscriptoin
1 does not have any sensor groups

3 entries printed
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
switch#

switch# show system internal telemetry trace tm-logs
Telemetry Log Traces:
0 entries printed
switch#

switch# show system internal telemetry trace tm-events
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
[07/26/17 15:19:40.610 UTC 4 28577] [3960399872][tm_init_n9k.c:207] Adding telemetry to
cgroup
[07/26/17 15:19:40.670 UTC 5 28577] [3960399872][tm_init_n9k.c:215] Added telemetry to
cgroup successfully!

switch# show system internal telemetry trace tm-errors
Telemetry Error Traces:
0 entries printed
switch#
```

NX-API を使用したテレメトリの構成

NX-API を使用したテレメトリの構成

スイッチ DME のオブジェクト モデルでは、「DME のテレメトリ モデル」のセクションで説明されているように、テレメトリ機能の構成がオブジェクトの階層構造で定義されています。構成する主なオブジェクトは次のとおりです。

- **fmEntity** — NX-API およびテレメトリ機能の状態が含まれています。
 - **fmNxapi** — NX-API の状態が含まれています。
 - **fmTelemetry** — テレメトリ機能の状態が含まれています。
- **telemetryEntity** — テレメトリ機能の構成が含まれています。
 - **telemetrySensorGroup** — テレメトリのために監視される 1 つ以上のセンサー パスまたはノードの定義が含まれています。テレメトリ エンティティには、1 つ以上のセンサー グループを含めることができます。
 - **telemetryRtSensorGroupRel** — センサー グループをテレメトリ サブスクリプションに関連付けます。
 - **telemetrySensorPath** — モニタリングされるパス。センサー グループには、このタイプのオブジェクトを複数含めることができます。
 - **telemetryDestGroup** — テレメトリ データを受信する 1 つ以上の接続先の定義が含まれています。テレメトリ エンティティには、1 つ以上の接続先グループを含めることができます。
 - **telemetryRtDestGroupRel** — 接続先グループをテレメトリ サブスクリプションに関連付けます。
 - **telemetryDest** — 接続先アドレス接続先グループには、このタイプのオブジェクトを複数含めることができます。
 - **telemetrySubscription** — 1 つ以上のセンサー グループからのテレメトリ データを 1 つ以上の接続先グループに送信する方法とタイミングを指定します。
 - **telemetryRsDestGroupRel** — テレメトリ サブスクリプションを接続先グループに関連付けます。
 - **telemetryRsSensorGroupRel** — テレメトリ サブスクリプションをセンサー グループに関連付けます。
 - **telemetryCertificate** — テレメトリ サブスクリプションを証明書とホスト名に関連付けます。

NX-API を使用してテレメトリ機能を設定するには、テレメトリ オブジェクト構造の JSON 表現を構築し、HTTP または HTTPS POST 操作で DME にプッシュする必要があります。



- (注) NX-API の使用に関する詳細な手順は、『Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference (Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK ユーザー ガイドと API リファレンス)』を参照してください。

始める前に

スイッチは、Cisco NX-OS リリース 7.3 (0) I5 (1) 以降のリリースを実行している必要があります。

CLI から NX-API を実行するようにスイッチを構成する必要があります。

```
switch(config)# feature nxapi
```

NX-API は、管理 VRF を介してテレメトリ データを送信します。

```
switch(config)# nxapi use-vrf management
```

```
nxapi use-vrf vrf_name
nxapi http port port_number
```

手順

	コマンドまたはアクション	目的
ステップ 1	テレメトリ機能を有効にします。 例 : <pre>{ "fmEntity" : { "children" : [{ "fmTelemetry" : { "attributes" : { "adminSt" : "enabled" } }] } }</pre>	ルート要素は fmTelemetry であり、この要素のベース パスは <code>sys/fm</code> です。 adminSt 属性を有効に構成します。
ステップ 2	テレメトリ構成を記述するために、JSON ペイロードのルート レベルを作成します。 例 : <pre>{</pre>	ルート要素は telemetryEntity であり、この要素のベース パスは <code>sys/tm</code> です。 dn 属性を <code>sys/tm</code> として構成します。

	コマンドまたはアクション	目的
	<pre> "telemetryEntity": { "attributes": { "dn": "sys/tm" }, } </pre>	
ステップ 3	定義されたセンサー パスを含むセンサーグループを作成します。 例 : <pre> "telemetrySensorGroup": { "attributes": { "id": "10", "rn": "sensor-10" "dataSrc": "NX-API" }, "children": [{ }] </pre>	テレメトリセンサーグループは、クラス telemetrySensorGroup のオブジェクトで定義されます。以下のオブジェクト属性を構成します。 • id—センサーグループの識別子。現在は、数字の ID 値のみサポートされています。 • rn — センサーグループオブジェクトの相対名（形式: sensor-id）。 • dataSrc : DEFAULT、DME、YANG、または NX-API からデータ送信元を選択します。 センサーグループオブジェクトの子には、センサーパスと1つ以上の関係オブジェクト（telemetryRtSensorGroupRel）が含まれ、センサーグループをテレメトリサブスクリプションに関連付けます。
ステップ 4	（任意）SSL/TLS 証明書とホストを追加します。 例 : <pre> { "telemetryCertificate": { "attributes": { "filename": "root.pem" "hostname": "c.com" } } } </pre>	telemetryCertificate は、テレメトリサブスクリプション/接続先でSSL/TLS証明書の場所を定義します。
ステップ 5	テレメトリの接続先グループを定義します。 例 : <pre> { "telemetryDestGroup": { "attributes": { "id": "20" } } } </pre>	テレメトリ接続先グループが telemetryEntity で定義されています。id 属性を構成します。

	コマンドまたはアクション	目的
	<pre> } } } </pre>	
ステップ 6	テレメトリの接続先プロファイルを定義します。 例 : <pre> { "telemetryDestProfile": { "attributes": { "adminSt": "enabled" }, "children": [{ "telemetryDestOptSourceInterface": { "attributes": { "name": "lo0" } } }] } } </pre>	テレメトリの接続先プロファイルは、telemetryDestProfile で定義されています。 • adminSt 属性を有効に設定します。 • telemetryDestOptSourceInterface の下で、構成されたインターフェイスからソース IP アドレスを持つ接続先にデータをストリーミングするためのインターフェイス名を使用して name 属性を構成します。
ステップ 7	テレメトリ データの送信先となる IP アドレスとポート番号で構成される、1 つ以上のテレメトリの接続先を定義します。 例 : <pre> { "telemetryDest": { "attributes": { "addr": "1.2.3.4", "enc": "GPB", "port": "50001", "proto": "gRPC", "rn": "addr-[1.2.3.4]-port-50001" } } } </pre>	テレメトリの接続先は、クラス telemetryDest のオブジェクトで定義されます。以下のオブジェクト属性を構成します。 • addr — 接続先の IP アドレス。 • port — 接続先のポート番号。 • rn — path-[path] 形式の接続先オブジェクトの相対名。 • enc — 送信されるテレメトリ データのエンコーディング タイプ。NX-OS は以下をサポートします。 • gRPC の Google Protocol Buffer (GPB) 。 • C の JSON。 • UDP およびセキュア UDP (DTLS) の場合は GPB または JSON。

	コマンドまたはアクション	目的
		• proto — 送信されるテレメトリデータのトランスポートプロトコルタイプ。NX-OS は以下をサポートします。 • gRPC • HTTP • VUDP とセキュア UDP (DTLS) • サポートされているエンコードタイプは次のとおりです。 • HTTP/JSON はい • HTTP/Form-data はい Bin Logging でのみサポートされます。 • GRPC/GPB-Compact はい ネイティブデータソースのみ。 • GRPC/GPB はい • UDP/GPB はい • UDP/JSON はい
ステップ 8	gRPC チャンクを有効にして、チャンク サイズを 64 ～ 4096 バイトに設定します。 例 : <pre>{ "telemetryDestGrpOptChunking": { "attributes": { "chunkSize": "2048", "dn": "sys/tm/dest-1/chunking" } } }</pre>	詳細については、[gRPC チャンクのサポート (Support for gRPC Chunking)] を参照してください。
ステップ 9	テレメトリサブスクリプションを作成して、テレメトリの動作を構成します。 例 :	テレメトリサブスクリプションは、クラス telemetrySubscription のオブジェクトで定義されます。以下のオブジェクト属性を構成します。

	コマンドまたはアクション	目的
	<pre> "telemetrySubscription": { "attributes": { "id": "30", "rn": "subs-30" }, "children": [{ }] } </pre>	• id — サブスクリプションの識別子。現在は、数字の ID 値のみサポートされています。 • rn — subs-id という形式のサブスクリプションオブジェクトの相対名。 サブスクリプションオブジェクトの子には、センサー グループ (telemetryRsSensorGroupRel) および 接続先グループ (telemetryRsDestGroupRel) の関係オブジェクトが含まれます。
ステップ 10	ルート要素の下の telemetrySubscription 要素に子オブジェクトとしてセンサーグループ オブジェクトを追加します (telemetryEntity)。 例 : <pre> { "telemetrySubscription": { "attributes": { "id": "30" } "children": [{ "telemetryRsSensorGroupRel": { "attributes": { "sampleIntvl": "5000", "tDn": "sys/tm/sensor-10" } }] } </pre>	
ステップ 11	サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ センサーグループに関連付け、データ サンプリング動作を指定します。 例 : <pre> "telemetryRsSensorGroupRel": { "attributes": { "rType": "mo", "rn": </pre>	関係オブジェクトはクラス telemetryRsSensorGroupRel であり、telemetrySubscription の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • rn — rsensorGroupRel-[sys/tm/sensor-group-id] 形式の関係オブジェクトの相対名。

	コマンドまたはアクション	目的
	<pre> "rSensorGroupRel-[sys/tm/sensor-10]", "sampleIntvl": "5000", "tCl": "telemetrySensorGroup", "tDn": "sys/tm/sensor-10", "tType": "mo" } } </pre>	• sampleIntvl— ミリ秒単位のデータサンプリング期間。間隔の値が 0 の場合、イベントベースのサブスクリプションが作成され、テレメトリ データは、指定された MO の変更時にのみ送信されます。0 より大きい間隔値の場合、テレメトリ データが指定された間隔で定期的に送信される頻度に基づいたサブスクリプションが作成されます。たとえば、間隔値が 15000 の場合、テレメトリ データは 15 秒ごとに送信されます。 • tCl— telemetrySensorGroup であるターゲット（センサー グループ）オブジェクトのクラス。 • tDn— sys/tm/sensor-group-id であるターゲット（センサー グループ）オブジェクトの識別名。 • rType— 管理対象オブジェクト用 mo の関係タイプ。 • tType— 管理対象オブジェクト用 mo のターゲット タイプ。
ステップ 12	テレメトリをモニタリングする 1 つ以上のセンサーパスまたはノードを定義します。 例： 単一センサー パス <pre> { "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0", "alias": "cdp_alias", } } } </pre>	センサー パスは、クラス telemetrySensorPath のオブジェクトで定義されます。以下のオブジェクト属性を構成します。 • path— モニタリングされるパス。 • rn— path-[path] 形式のパス オブジェクトの相対名： • depth— センサー パスの取得レベル。0 の深さ設定は、ルート MO プロパティのみを取得します。 • filterCondition— （オプション）イベントベースのサブスクリプション用の特定のフィルタを作成します。DME はフィルター式を提供します。フィルタリングの詳細

	コマンドまたはアクション	目的
	<pre> }</pre> 例 : NX-API の単一センサー パス <pre> { "telemetrySensorPath": { "attributes": { "path": "show interface", "path": "show bgp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }</pre> 例 : 複数のセンサー パス <pre> { "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }, { "telemetrySensorPath": { "attributes": { "excludeFilter": "", "filterCondition": "", "path": "sys/fm/dhcp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }</pre> 例 : BGP 無効化イベントの単一センサー パス フィルタリング :	については、クエリの作成に関する Cisco APIC REST API の使用注意事項を参照してください。 https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635 • alias : このパスのエイリアスを指定します。

	コマンドまたはアクション	目的
	<pre>{ "telemetrySensorPath": { "attributes": { "path": "sys/cdp", "rn": "path-[sys/cdp]", "excludeFilter": "", "filterCondition": "eq(fmBgp.operSt.\"disabled\")", "path": "sys/fm/bgp", "secondaryGroup": "0", "secondaryPath": "", "depth": "0" } } }</pre>	
ステップ 13	センサーパスを子オブジェクトとしてセンサー グループ オブジェクト (telemetrySensorGroup) に追加します。	
ステップ 14	接続先を子オブジェクトとして接続先グループ オブジェクト (telemetryDestGroup) に追加します。	
ステップ 15	接続先グループオブジェクトを子オブジェクトとしてルート要素に追加します (telemetryEntity)。	
ステップ 16	センサー グループをサブスクリプションに関連付けるために、テレメトリ センサー グループの子オブジェクトとして関係オブジェクトを作成します。 例 : <pre>"telemetryRtSensorGroupRel": { "attributes": { "rn": "rtsensorGroupRel-[sys/tm/subs-30]", "tCl": "telemetrySubscription", "tDn": "sys/tm/subs-30" } }</pre>	関係オブジェクトはクラス telemetryRtSensorGroupRel であり、telemetrySensorGroup の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • rn — rtsensorGroupRel-[sys/tm/subscription-id] の形式の関係オブジェクトの相対名。 • tCl — サブスクリプション オブジェクト telemetrySubscription のターゲット クラス。 • tDn — sys/tm/subscription-id である、サブスクリプションオブジェクトのターゲット識別名。

	コマンドまたはアクション	目的
ステップ 17	テレメトリ接続先グループの子オブジェクトとして関係オブジェクトを作成して、接続先グループをサブスクリプションに関連付けます。 例 : <pre>"telemetryRtDestGroupRel": { "attributes": { "rn": "rtdestGroupRel-[sys/tm/subs-30]", "tCl": "telemetrySubscription", "tDn": "sys/tm/subs-30" } }</pre>	関係オブジェクトはクラス telemetryRtDestGroupRel であり、telemetryDestGroup の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • rn — rtdestGroupRel-[sys/tm/subscription-id] の形式の関係オブジェクトの相対名。 • tCl — サブスクリプション オブジェクト telemetrySubscription のターゲット クラス。 • tDn — sys/tm/subscription-id である、サブスクリプションオブジェクトのターゲット識別名。
ステップ 18	サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ接続先グループに関連付けます。 例 : <pre>"telemetryRsDestGroupRel": { "attributes": { "rType": "mo", "rn": "rsdestGroupRel-[sys/tm/dest-20]", "tCl": "telemetryDestGroup", "tDn": "sys/tm/dest-20", "tType": "mo" } }</pre>	関係オブジェクトはクラス telemetryRsDestGroupRel であり、telemetrySubscription の子オブジェクトです。関係オブジェクトの以下のオブジェクト属性を構成します。 • rn — rsdestGroupRel-[sys/tm/destination-group-id] の形式の関係オブジェクトの相対名。 • tCl — ターゲット (接続先グループ) オブジェクトのクラス telemetryDestGroup。 • tDn — 接続先グループ ID であるターゲット (接続先グループ) オブジェクト sys/tm/destination-group-id の識別名。 • rType — 管理対象オブジェクト用 mo の関係タイプ。 • tType — 管理対象オブジェクト用 mo のターゲット タイプ。

	コマンドまたはアクション	目的
ステップ 19	テレメトリ構成のために、結果のJSON 構造を HTTP/HTTPS POST ペイロードとして NX-API エンドポイントに送信します。	テレメトリ エンティティのベースパスは <code>sys/tm</code> で、NX-API エンドポイントは次のとおりです。 <code>{{URL}}/api/node/mo/sys/tm.json</code>

例

以下は、1 つの POST ペイロードに収集された、その前のすべてのステップの例です（一部の属性が一致しない場合があることに注意してください）。

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
      },
      "children": [{
        "telemetrySensorPath": {
          "attributes": {
            "excludeFilter": "",
            "filterCondition": "",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
          }
        }
      ]
    }
  ],
  {
    "telemetryDestGroup": {
      "attributes": {
        "id": "20"
      }
    },
    "children": [{
      "telemetryDest": {
        "attributes": {
          "addr": "10.30.217.80",
          "port": "50051",
          "enc": "GPB",
          "proto": "gRPC"
        }
      }
    }
  ],
  {
    "telemetrySubscription": {
      "attributes": {
        "id": "30"
      }
    },
    "children": [{
      "telemetryRsSensorGroupRel": {
```

NX-API を使用したテレメトリの構成例

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。

Payload:

Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド、リリース 9.3(x)

```

    }
  }, {
    "telemetrySensorPath": {
      "attributes": {
        "path": "sys/ipv4",
        "rn": "path-[sys/ipv4]",
        "excludeFilter": "",
        "filterCondition": "",
        "secondaryGroup": "0",
        "secondaryPath": "",
        "depth": "0"
      }
    }
  }
]
}, {
  "telemetryDestGroup": {
    "attributes": {
      "id": "20",
      "rn": "dest-20"
    },
    "children": [{
      "telemetryRtDestGroupRel": {
        "attributes": {
          "rn": "rtdestGroupRel-[sys/tm/subs-30]",
          "tCl": "telemetrySubscription",
          "tDn": "sys/tm/subs-30"
        }
      }
    }
  }, {
    "telemetryDest": {
      "attributes": {
        "addr": "1.2.3.4",
        "enc": "GPB",
        "port": "50001",
        "proto": "gRPC",
        "rn": "addr-[1.2.3.4]-port-50001"
      }
    }
  }
]
}, {
  "telemetrySubscription": {
    "attributes": {
      "id": "30",
      "rn": "subs-30"
    },
    "children": [{
      "telemetryRsDestGroupRel": {
        "attributes": {
          "rType": "mo",
          "rn": "rsdestGroupRel-[sys/tm/dest-20]",
          "tCl": "telemetryDestGroup",
          "tDn": "sys/tm/dest-20",
          "tType": "mo"
        }
      }
    }
  }, {
    "telemetryRsSensorGroupRel": {
      "attributes": {
        "rType": "mo",
        "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
        "sampleIntvl": "5000",
        "tCl": "telemetrySensorGroup",

```

```

        "tDn": "sys/tm/sensor-10",
        "tType": "mo"
    }
}
    }
}
    }
}
}

```

BGP 通知のフィルタ条件

次のペイロードの例では、telemetrySensorPath MO の filterCondition 属性に従って BFP 機能が無効になっているときにトリガーされる通知を有効にします。データは 10.30.217.80 ポート 50055 にストリーミングされます。

POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:

```

{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        ]
      }
    ]
  },
  {
    "telemetryDestGroup": {
      "attributes": {
        "id": "20"
      }
      "children": [{
        "telemetryDest": {
          "attributes": {
            "addr": "10.30.217.80",
            "port": "50055",
            "enc": "GPB",
            "proto": "gRPC"
          }
        }
      ]
    }
  },
  {
    "telemetrySubscription": {
      "attributes": {

```

```

        "id": "30"
      }
      "children": [{
        "telemetryRsSensorGroupRel": {
          "attributes": {
            "sampleIntvl": "0",
            "tDn": "sys/tm/sensor-10"
          }
        }
      },
      {
        "telemetryRsDestGroupRel": {
          "attributes": {
            "tDn": "sys/tm/dest-20"
          }
        }
      }
    ]
  }
}
]
}
}
}

```

テレメトリ構成のための Postman コレクションの使用

[Postman コレクションの例](#)は、テレメトリ機能の構成を開始する簡単な方法であり、1 つのペイロードですべてのテレメトリ CLI に相当するものを実行できます。好みのテキストエディターを使用して前述のリンクのファイルを変更し、ペイロードをニーズに合わせて更新してから、Postman でコレクションを開いてコレクションを実行します。

DME のテレメトリ モデル

テレメトリ アプリケーションは、次の構造を持つ DME でモデル化されます。

```

model
|----package [name:telemetry]
|  @name:telemetry
|----objects
|  |----mo [name:Entity]
|  |  @name:Entity
|  |  @label:Telemetry System
|  |--property
|  |  @name:adminSt
|  |  @type:AdminState
|  |
|  |----mo [name:SensorGroup]
|  |  |  @name:SensorGroup
|  |  |  @label:Sensor Group
|  |  |--property
|  |  |  @name:id [key]
|  |  |  @type:string:Basic
|  |  |  @name:dataSrc
|  |  |  @type:DataSource
|  |  |
|  |  |----mo [name:SensorPath]
|  |  |  @name:SensorPath
|  |  |  @label:Sensor Path
|  |  |--property

```

```

|         | @name:path [key]
|         | @type:string:Basic
|         | @name:filterCondition
|         | @type:string:Basic
|         | @name:excludeFilter
|         | @type:string:Basic
|         | @name:depth
|         | @type:RetrieveDepth
|
|----mo [name:DestGroup]
|   | @name:DestGroup
|   | @label:Destination Group
|   |--property
|   | @name:id
|   | @type:string:Basic
|   |
|   |----mo [name:Dest]
|   |   | @name:Dest
|   |   | @label:Destination
|   |   |--property
|   |   | @name:addr [key]
|   |   | @type:address:Ip
|   |   | @name:port [key]
|   |   | @type:scalar:Uint16
|   |   | @name:proto
|   |   | @type:Protocol
|   |   | @name:enc
|   |   | @type:Encoding
|   |
|   |----mo [name:Subscription]
|   |   | @name:Subscription
|   |   | @label:Subscription
|   |   |--property
|   |   | @name:id
|   |   | @type:scalar:Uint64
|   |   |----reldef
|   |   |   | @name:SensorGroupRel
|   |   |   | @to:SensorGroup
|   |   |   | @cardinality:ntom
|   |   |   | @label:Link to sensorGroup entry
|   |   |   |--property
|   |   |   | @name:sampleIntvl
|   |   |   | @type:scalar:Uint64
|   |   |
|   |   |----reldef
|   |   |   | @name:DestGroupRel
|   |   |   | @to:DestGroup
|   |   |   | @cardinality:ntom
|   |   |   | @label:Link to destGroup entry

```

テレメトリ機能で利用できるDNのリストについては、「[テレメトリソースをストリーミングする](#)」を参照してください。

クラウドスケール ソフトウェア テレメトリ

クラウドスケール ソフトウェア テレメトリについて

NX-OS リリース 9.3(1) 以降、ソフトウェア テレメトリは、Tahoe ASIC を使用する Cisco Nexus クラウドスケールスイッチでサポートされます。このリリースで、サポートされているクラウドスケールスイッチは、ASIC と緊密に統合された TCP/IP サーバーをホストします。これにより、スイッチからのテレメトリ データのレポートをすばやく処理できます。サーバーは TCP ポート 7891 を使用します。テレメトリ クライアントはこのポートでサーバーに接続して、最大 10 ミリ秒でハードウェア カウンタ データを取得できます。

クラウドスケール ソフトウェア テレメトリには、独自のクライアントプログラムを作成したり、NX-OS リリース 9.3.1 以降にバンドルされているデフォルトのクライアントプログラムを使用したりする柔軟性があります。クライアントプログラムは、Python 2.7 以降、C、PHP など、TCP/IP をサポートする任意のプログラミング言語で作成できます。クライアントプログラムは、正しいメッセージフォーマットで作成する必要があります。

NX-OS リリース 9.3(1) 以降、クラウドスケール ソフトウェア テレメトリ機能は NX-OS で使用できます。この機能はデフォルトで有効になっているため、NX-OS 9.3(1) 以降を実行しているサポート対象のスイッチでは、この機能を使用できます。

Cloud Scale ソフトウェア テレメトリ メッセージの形式

Cloud Scale テレメトリは、クライアントとスイッチ上の TCP/IP サーバー間のハンドシェイクで始まります。その間にクライアントは TCP ソケットを介して接続を開始します。クライアントメッセージは、32 ビット整数での 0 です。スイッチは、特定の形式のカウンタ データを含むメッセージで応答します。

NX-OS リリース 9.3(1) では、次のメッセージフォーマットがサポートされています。独自のクライアントプログラムを作成する場合は、クライアントが開始するメッセージがこの形式に準拠していることを確認してください。

長さ	指定します。
4 バイト	ポート数、N

長さ	指定します。
56 バイト	各ポートのデータ、合計 $56 * N$ バイト。 データの各 56 バイトチャンクは、次のもので構成されます。 • 24 バイトのインターフェイス名 • 8 バイトの送信 (TX) パケット • 8 バイトの送信 (TX) バイト • 8 バイトの受信 (RX) パケット • 8 バイトの受信 (RX) バイト

クラウドスケールソフトウェアテレメトリの注意事項と制限事項

次に、クラウドスケールソフトウェアテレメトリ機能の注意事項と制限事項を示します。

- リリース 9.3(x) より前の Cisco NX-OS でサポートされるプラットフォームについては、同ガイドのプログラマビリティに対するプラットフォームのサポートの項を参照してください。Cisco NX-OS リリース 9.3(x) 以降でサポートされているプラットフォームのについては、[Nexus Switch Platform Matrix](#)を参照してください。
- カスタムクライアントテレメトリプログラムでは、サポートされているメッセージフォーマットは 1 つです。クライアントプログラムは、この形式に準拠している必要があります。

テレメトリ パス ラベル

テレメトリ パス ラベルについて

NX-OS リリース 9.3(1) 以降、モデル駆動型テレメトリはパス ラベルをサポートします。パス ラベルを使用すると、複数のソースからテレメトリデータを一度に簡単に収集できます。この機能では、収集するテレメトリデータのタイプを指定すると、テレメトリ機能によって複数のパスからそのデータが収集されます。次に、機能は情報を 1 つの統合された場所（パス ラベル）に返します。この機能により、次の作業が不要になるため、テレメトリの使用が簡素化されます。

- Cisco DME モデルに関する深く包括的な知識を持っています。
- 収集されるイベントの数と頻度のバランスを取りながら、複数のクエリを作成し、サブスクリプションに複数のパスを追加します。
- スイッチからテレメトリ情報の複数のチャンクを収集し、有用性を簡素化します。

パス ラベルは、モデル内の同じオブジェクト タイプの複数のインスタンスにわたり、カウンタまたはイベントを収集して返します。パス ラベルは、次のテレメトリ グループをサポートします。

- ファン、温度、電力、ストレージ、スーパーバイザ、ラインカードなどのシャーシ情報をモニタリングする環境。
- すべてのインターフェイス カウンターとステータスの変更をモニタリングするインターフェイス。
このラベルは、**query-condition** コマンドを使用して返されるデータを絞り込むための定義済みのキーワード フィルタをサポートします。
- リソース。CPU 使用率やメモリ使用率などのシステム リソースをモニタリングします。
- VXLAN: VXLAN ピア、VXLAN カウンタ、VLAN カウンター、および BGP ピア データを含む VXLAN EVPN をモニタリングします。

データの投票またはイベントの受信

センサー グループのサンプル間隔によって、テレメトリ データがパス ラベルに送信される方法とタイミングが決まります。サンプル間隔は、テレメトリ データを定期的に投票するか、イベントが発生したときにテレメトリ データを収集するように構成できます。

- テレメトリのサンプル間隔がゼロ以外の値に設定されている場合、テレメトリは各サンプル間隔中に環境、インターフェイス、情報技術、および vxlan ラベルのデータを定期的に送信します。
- サンプル間隔がゼロに設定されている場合、環境、インターフェイス、情報技術、vxlan ラベルで動作状態の更新、および MO の作成と削除が発生するとテレメトリはイベント通知を送信します。

データの投票または受信イベントは相互に排他的です。パス ラベルごとに投票またはイベント駆動型テレメトリを構成できます。

パス ラベル注意事項と制約事項

テレメトリ パス ラベル機能には、次の注意事項と制約事項があります。

- この機能は、Cisco DME データ 送信元のみをサポートします。
- 同じセンサー グループ内の通常の DME パスとユーザビリティ パスを混在させて一致させることはできません。たとえば、sys/intf と [インターフェイス (interface)] を同じセンサー グループに構成することはできません。また、sys/intf と [interface (インターフェイス)] で同じセンサー グループを構成することはできません。この状況が発生した場合、NX-OS は構成を拒否します。
- oper-speed や counters=[detailed] などのユーザー フィルター キーワードは、[インターフェイス (interface)] パスに対してのみサポートされます。

- この機能は、[深度 (depth)] や [フィルター条件 (filter-condition)] などの他のセンサー パス オプションをサポートしていません。
- テレメトリ パス ラベルには、パス ラベルの使用に関する次の制限があります。

- 大文字と小文字が区別されるため、小文字のプレフィックス `show` で開始する必要があります。

例：`show version` は許可されます。ただし、`show version` または `version` は使用できません。

- 次の文字を含めることはできません。

- ;
- |
- " または '

- 次の単語を含めることはできません。

- `telemetry`
- `conf t`
- 設定

データまたはイベントをポーリングするためのインターフェイスパスの構成

インターフェイス パス ラベルは、すべてのインターフェイス カウンタとステータスの変更をモニタリングします。次のインターフェイス タイプをサポートします。

- 物理
- サブインターフェイス
- 管理
- ループバック
- VLAN
- ポート チャネル

インターフェイス パス ラベルを構成して、定期的にデータをポーリングするか、イベントを受信することができます。「[データの投票またはイベントの受信 \(482 ページ\)](#)」を参照してください。



(注) このモデルは、サブインターフェイス、ループバック、または VLAN のカウンタをサポートしていないため、ストリームアウトされません。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	コンフィギュレーション モードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(config-telemetry)# sensor-group 6 switch(conf-tm-sensor)#</pre>	テレメトリ データのセンサー グループを作成します。
ステップ 4	path interface 例 : <pre>switch(conf-tm-sensor)# path interface switch(conf-tm-sensor)#</pre>	インターフェイス パス ラベルを構成して、複数の個々のインターフェイスに対して 1 つのテレメトリ データ クエリを送信できるようにします。ラベルは、複数のインターフェイスのクエリを 1 つに統合します。次に、テレメトリはデータを収集し、ラベルに返します。 ポーリング間隔の設定方法に応じて、インターフェイス データは定期的に、またはインターフェイスの状態が変化するたびに送信されます。
ステップ 5	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。

	コマンドまたはアクション	目的
ステップ 6	ip address ip_addr port port 例 : <pre>switch(conf-tm-dest) # ip address 1.2.3.4 port 50004 switch(conf-tm-dest) #</pre>	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 8	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的送信するか、インターフェイス イベントが発生したときに送信するかを決定します。
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

非ゼロ カウンタのインターフェイス パスの構成

ゼロ以外の値を持つカウンタのみを返す事前定義されたキーワードフィルタを使用して、インターフェイス パス ラベルを構成できます。フィルタは `counters=[detailed]` です。

このフィルタを使用することにより、インターフェイス パスは使用可能なすべてのインターフェイスカウンタを収集し、収集したデータをフィルタ処理してから、結果を受信側に転送します。フィルタはオプションであり、使用しない場合、ゼロ値カウンタを含むすべてのカウンタがインターフェイス パスに表示されます。



(注) フィルタの使用は、概念的には `show interface mgmt0 counters detailed` と類似しています。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : switch# configure terminal switch(config)#	コンフィギュレーション モードを入力します。
ステップ 2	telemetry 例 : switch(config)# telemetry switch(config-telemetry)#	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : switch(config-telemetry)# sensor-group 6 switch(conf-tm-sensor)#	テレメトリ データのセンサー グループを作成します。
ステップ 4	path interface query-condition counters=[detailed] 例 : switch(conf-tm-sensor)# path interface query-condition counters=[detailed] switch(conf-tm-sensor)#	インターフェイス パス ラベルを構成し、すべてのインターフェイスからのゼロ以外のカウンタのみを照会します。
ステップ 5	destination-group grp_id 例 : switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
ステップ 6	ip address ip_addr port port 例 : switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch(conf-tm-dest)#	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : switch(conf-tm-dest)# subscription 33 switch(conf-tm-sub)#	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 8	snsr-group sgrp_id sample-interval interval 例 :	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定しま

	コマンドまたはアクション	目的
	<pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	す。サンプリング間隔は、スイッチがテレメトリデータを定期的送信するか、インターフェイス イベントが発生したときに送信するかを決定します。
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

動作速度のインターフェイス パスの構成

指定された動作速度のインターフェイスのカウンタを返す定義済みのキーワードフィルタを使用して、インターフェイスパスラベルを構成できます。フィルタは `oper-speed=[]` です。次の動作速度がサポートされています: auto、10M、100M、1G、10G、40G、200G、および 400G。

このフィルタを使用することにより、インターフェイスパスは指定された速度のインターフェイスのテレメトリデータを収集し、その結果を受信側に転送します。フィルタはオプションです。使用しない場合、動作速度に関係なく、すべてのインターフェイスのカウンタが表示されます。

フィルタは、複数の速度をコンマ区切りのリストとして受け入れることができます。たとえば、`oper-speed=[1G,10G]` は、1 および 10 Gbps で動作するインターフェイスのカウンタを取得します。区切り文字として空白を使用しないでください。



(注) インターフェイス タイプ サブインターフェイス、ループバック、および VLAN には動作速度プロパティがないため、フィルタはこれらのインターフェイス タイプをサポートしません。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config) #</pre>	コンフィギュレーション モードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config) # telemetry switch(config-telemetry) #</pre>	テレメトリ機能の構成モードに入ります。

	コマンドまたはアクション	目的
ステップ 3	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。
ステップ 4	path interface query-condition oper-speed=[speed] 例 : <pre>switch(conf-tm-sensor) # path interface query-condition oper-speed=[1G,40G] switch(conf-tm-sensor) #</pre>	インターフェイス パス ラベルを設定し、指定された速度（この例では 1 Gbps と 40 Gbps のみ）を実行しているインターフェイスからのカウンターを照会します。
ステップ 5	destination-group grp_id 例 : <pre>switch(conf-tm-sensor) # destination-group 33 switch(conf-tm-dest) #</pre>	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
ステップ 6	ip address ip_addr port port 例 : <pre>switch(conf-tm-dest) # ip address 1.2.3.4 port 50004 switch(conf-tm-dest) #</pre>	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 8	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

複数のクエリによるインターフェイスパスの構成

インターフェイスパスラベルの同じクエリ条件に対して複数のフィルタを構成できます。その場合、使用する個々のフィルタはANDで結合されます。

クエリ条件の各フィルタは、コンマを使用して区切ります。query-condition には、任意の数のフィルタを指定できますが、追加するフィルタが多いほど、結果の焦点が絞られることに注意してください。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : switch# configure terminal switch(config)#	コンフィギュレーション モードを入力します。
ステップ 2	telemetry 例 : switch(config)# telemetry switch(config-telemetry)#	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group grp_id 例 : switch(config-telemetry)# sensor-group 6 switch(conf-tm-sensor)#	テレメトリ データのセンサー グループを作成します。
ステップ 4	path interface query-condition counters=[detailed],oper-speed=[1G,40G] 例 : switch(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G] switch(conf-tm-sensor)#	同じクエリで複数の条件を構成します。 この例では、クエリは次の両方を実行します。 • 1 Gbps で実行されているインターフェイスでゼロ以外のカウンターを収集して返します。 • 40 Gbps で実行されているインターフェイスでゼロ以外のカウンターを収集して返します。
ステップ 5	destination-group grp_id 例 : switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#	テレメトリ接続先グループサブモードに入り、接続先グループを構成します。

	コマンドまたはアクション	目的
ステップ 6	ip address ip_addr port port 例 : <pre>switch(conf-tm-dest) # ip address 1.2.3.4 port 50004 switch(conf-tm-dest) #</pre>	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 8	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

データまたはイベントをポーリングするための環境パスの構成

環境パスラベルは、ファン、温度、電源、ストレージ、スーパーバイザ、ラインカードなどのシャシ情報をモニタリングします。テレメトリデータを定期的にポーリングするか、イベントが発生したときにデータを取得するように環境パスを構成できます。詳細については、[データの投票またはイベントの受信（482 ページ）](#) を参照してください。

定期的なポーリングまたはイベントに基づいてシステム リソース情報を返すようにリソースパスを設定できます。このパスはフィルタリングをサポートしていません。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config) #</pre>	コンフィギュレーション モードを入力します。

	コマンドまたはアクション	目的
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(config-telemetry)# sensor-group 6 switch(conf-tm-sensor)#</pre>	テレメトリ データのセンサー グループを作成します。
ステップ 4	path environment 例 : <pre>switch(conf-tm-sensor)# path environment switch(conf-tm-sensor)#</pre>	複数の個々の環境オブジェクトのテレメトリ データをラベルに送信できるようにする環境パス ラベルを構成します。ラベルは、複数のデータ入力を1つの出力に統合します。 サンプル間隔に応じて、環境データはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。
ステップ 5	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
ステップ 6	ip address ip_addr port port 例 : <pre>switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch(conf-tm-dest)#</pre>	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : <pre>switch(conf-tm-dest)# subscription 33 switch(conf-tm-sub)#</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 8	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch(conf-tm-sub)#</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、環境イベントが発生したときに送信するかを決定します。

	コマンドまたはアクション	目的
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

イベントまたはデータをポーリングするためのリソース パスの構成

リソースパスは、CPU使用率やメモリ使用率などのシステムリソースをモニタリングします。このパスを構成して、テレメトリデータを定期的に収集するか、イベントが発生したときに収集できます。「[データの投票またはイベントの受信 \(482 ページ\)](#)」を参照してください。

このパスはフィルタリングをサポートしていません。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config) #</pre>	コンフィギュレーションモードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config) # telemetry switch(config-telemetry) #</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(config-telemetry) # sensor-group 6 switch(conf-tm-sensor) #</pre>	テレメトリ データのセンサー グループを作成します。
ステップ 4	path resources 例 : <pre>switch(conf-tm-sensor) # path resources switch(conf-tm-sensor) #</pre>	複数の個々のシステム リソースのテレメトリ データをラベルに送信できるようにするリソース パス ラベルを構成します。ラベルは、複数のデータ入力を1つの出力に統合します。 サンプル間隔に応じて、リソース データはポーリング間隔に基づいてストリーミングされるか、システム メモリが「Not OK」に変更されたときに送信されます。

	コマンドまたはアクション	目的
ステップ 5	destination-group grp_id 例 : <pre>switch(conf-tm-sensor) # destination-group 33 switch(conf-tm-dest) #</pre>	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
ステップ 6	ip address ip_addr port port 例 : <pre>switch(conf-tm-dest) # ip address 1.2.3.4 port 50004 switch(conf-tm-dest) #</pre>	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 8	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、リソース イベントが発生したときに送信するかを決定します。
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

イベントまたはデータをポーリングするための VXLAN パスの構成

vxlan パス ラベルは、VXLAN ピア、VXLAN カウンター、VLAN カウンター、BGP ピア データなど、スイッチの仮想拡張 LAN EVPN に関する情報を提供します。このパス ラベルを構成して、定期的に、またはイベントが発生したときにテレメトリ情報を収集できます。「[データの投票またはイベントの受信（482 ページ）](#)」を参照してください。

このパスはフィルタリングをサポートしていません。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	コンフィギュレーション モードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(config-telemetry)# sensor-group 6 switch(conf-tm-sensor)#</pre>	テレメトリ データのセンサー グループを作成します。
ステップ 4	vxlan environment 例 : <pre>switch(conf-tm-sensor)# vxlan environment switch(conf-tm-sensor)#</pre>	複数の個々の VXLAN オブジェクトのテレメトリ データをラベルに送信できるようにする vxlan パス ラベルを構成します。ラベルは、複数のデータ入力を1つの出力に統合します。サンプル間隔に応じて、VXLAN データはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。
ステップ 5	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。
ステップ 6	ip address ip_addr port port 例 : <pre>switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 switch(conf-tm-dest)#</pre>	サブスクリプションのテレメトリ データを構成して、指定された IP アドレスとポートにストリーミングします。
ステップ 7	subscription sub_id 例 : <pre>switch(conf-tm-dest)# subscription 33 switch(conf-tm-sub)#</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。

	コマンドまたはアクション	目的
ステップ 8	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000 switch(conf-tm-sub)#</pre>	センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、VXLAN イベントが発生したときに送信するかを決定します。
ステップ 9	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub)# dst-grp 33 switch(conf-tm-sub)#</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

パス ラベル 構成 を確認

いつでも、パス ラベルが構成されていることを確認し、実行中のテレメトリ構成を表示してその値を確認できます。

手順

	コマンドまたはアクション	目的
ステップ 1	show running-config-telemetry 例 : <pre>switch(conf-tm-sensor)# show running-config telemetry !Command: show running-config telemetry !Running configuration last done at: Mon Jun 10 08:10:17 2019 !Time: Mon Jun 10 08:10:17 2019 version 9.3(1) Bios:version feature telemetry telemetry destination-profile use-nodeid tester sensor-group 4 path interface query-condition and(counters=[detailed],oper-speed=[1G,10G]) sensor-group 6 path interface query-condition oper-speed=[1G,40G] subscription 6 snsr-grp 6 sample-interval 6000 nxosv2(conf-tm-sensor)#</pre>	テレメトリの現在の実行構成を表示します。 この例では、センサー グループ 4 は、1 および 10 Gbps で実行されているインターフェイスからゼロ以外のカウンターを収集するように構成されています。センサー グループ 6 は、1 と 40 Gbps で実行されているインターフェイスからすべてのカウンターを収集するように構成されています。

パス ラベル情報の表示

パス ラベル表示コマンド

show telemetry usability コマンドを使用すると、クエリを発行したときにパス ラベルがたどる個々のパスを表示できます。

コマンド	表示内容
show telemetry usability {all environment interface resources vxlan}	すべてのパス ラベルのすべてのテレメトリパス、または指定されたパス ラベルのすべてのテレメトリパス。また、出力には、各パスが定期的なポーリングまたはイベントに基づいてテレメトリ データを報告するかどうかが表示されます。 インターフェイスパス ラベルには、設定したキーワードフィルタまたはクエリ条件も含まれます。
show running-config telemetry	テレメトリと選択されたパス情報の実行構成。

コマンドの例



(注) **show telemetry usability all** コマンドは、このセクションに示されている個々のコマンドをすべて連結したものです。

show telemetry usability environment コマンドの例を次に示します。

```
switch# show telemetry usability environment
  1) label_name      : environment

      path_name      : sys/ch
      query_type     : poll
      query_condition :
rsp-subtree-full&query-target-subtree=targets-subtree-class=eqPtPssSlot,eqPtFtSlot,eqPtSupCSlot,eqPtPss,eqPtFt,eqPtSensor,eqPtICSlot

  2) label_name      : environment

      path_name      : sys/ch
      query_type     : event
      query_condition :
rsp-subtree-full&query-target-subtree=targets-subtree-class=eqPtPssSlot,eqPtFtSlot,eqPtSupCSlot,eqPtPss,eqPtFt,eqPtSensor,eqPtICSlot
switch#
```

show telemetry usability interface コマンドの出力を次に示します。

```
switch# show telemetry usability interface
  1) label_name      : interface

      path_name      : sys/intf
```



```

        query_type          : poll
        query_condition      :
query-target=children&query-target-filter=eq(l2PhysIf.adminSt,"up")&rsp-subtree=children&rsp-subtree-class=notBdrStats,monIfIn,monIfOut,monIfClr,monIfOut

2) label_name              : interface

        path_name           : sys/mgmt-[mgmt0]
        query_type          : poll
        query_condition      :
query-target=stats&query-target-filter=eq(mgmtIf.adminSt,"up")&rsp-subtree=full&rsp-subtree-class=notBdrStats,monIfIn,monIfOut,monIfClr,monIfOut

3) label_name              : interface

        path_name           : sys/intf
        query_type          : event
        query_condition      :
query-target=stats&query-target-filter=eq(ethpmEncRtdIf.operSt,"down"),and(updated(ethpmEncRtdIf.operSt),eq(ethpmEncRtdIf.operSt,"up"))

4) label_name              : interface

        path_name           : sys/mgmt-[mgmt0]
        query_type          : event
        query_condition      :
query-target=stats&query-target-filter=or(delete0,create0),or(and(update(mgmtIf.qeSt),eq(mgmtIf.qeSt,"down")),and(update(mgmtIf.qeSt),eq(mgmtIf.qeSt,"up"))))
switch#

```

show telemetry usability resources コマンドの例を次に示します。

```

switch# show telemetry usability resources
1) label_name              : resources

        path_name           : sys/proc
        query_type          : poll
        query_condition      : rsp-subtree=full&rsp-foreign-subtree=ephemeral

2) label_name              : resources

        path_name           : sys/procsys
        query_type          : poll
        query_condition      :
query-target=stats&query-target-filter=eq(procSysMem.memstatus,"OK"),ne(procSysMem.memstatus,"OK")

3) label_name              : resources

        path_name           : sys/procsys/systemem
        query_type          : event
        query_condition      :
query-target-filter=and(updated(procSysMem.memstatus),ne(procSysMem.memstatus,"OK"))

switch#

```

show telemetry usability vxlan コマンドの例を次に示します。

```

switch# show telemetry usability vxlan
1) label_name              : vxlan

        path_name           : sys/bd
        query_type          : poll
        query_condition      : query-target=subtree&target-subtree-class=l2VlanStats

```

```

2) label_name      : vxlan

   path_name       : sys/eps
   query_type      : poll
   query_condition  : rsp-subtree=full&rsp-foreign-subtree=ephemeral

3) label_name      : vxlan

   path_name       : sys/eps
   query_type      : event
   query_condition  : query-target=subtree&target-subtree-class=nvoDyPeer

4) label_name      : vxlan

   path_name       : sys/bgp
   query_type      : event
   query_condition  : query-target=subtree&query-target-filter=or(deleted(),created())

5) label_name      : vxlan

   path_name       : sys/bgp
   query_type      : event
   query_condition  :
query-target=subtree&target-subtree-class=logDom,logPeer,logPeerAf,logDomAf,logPeerAfEntry,logQoSctrlL3,logQoSctrlP,logQoSctrlEntry,logQoSctrl
switch#

```

ネイティブ データ送信元パス

ネイティブ データ送信元パスについて

NX-OS テレメトリは、特定のインフラストラクチャまたはデータベースに限定されないニュートラルデータ送信元であるネイティブ データ ソースをサポートします。代わりに、ネイティブ データ送信元を使用すると、コンポーネントまたはアプリケーションをフックして、関連情報を発信テレメトリ ストリームに挿入できます。ネイティブ データ送信元のパスはインフラストラクチャに属さないため、この機能は柔軟性を提供し、ネイティブ アプリケーションは NX-OS テレメトリと対話できます。

ネイティブ データ送信元パスを使用すると、特定のセンサーパスに登録して、セレクトしたテレメトリ データを受信できます。この機能は NX-SDK と連携して、次のパスからのテレメトリ データのストリーミングをサポートします。

- IP ルートのテレメトリ データを送信する RIB パス。
- 静的および動的 MAC エントリのテレメトリ データを送信する MAC パス。
- IPv4 と IPv6 隣接のテレメトリ データを送信する隣接関係パス。

サブスクリプションを作成すると、選択したパスのすべてのテレメトリ データが基準値として受信者にストリーミングされます。基準値の後、イベント通知のみが受信者にストリーミングされます。

ネイティブ データ 送信元 パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf (GPB)
- JavaScript Object Notation (JSON)
- コンパクト Google Protobuf (コンパクト GPB)

ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ

次の表は、各ソースパスについて、サブスクリプションが最初に作成されたとき（ベースライン）とイベント通知が発生したときにストリーミングされる情報を示しています。

Path Type	サブスクリプション ベースライン	イベント通知 (Event Notifications)
RIB	全てのルートの送信	

Path Type	サブスクリプション ベースライン	イベント通知 (Event Notifications)
		イベントの作成、更新、および削除に関するイベント通知を送信します。次の値は、RIB パスのテレメトリを介してエクスポートされます： • ネクスト ホップ ルーティング情報： • ネクスト ホップ のアドレス • ネクスト ホップ の発信インターフェイス • ネクスト ホップ の VRF 名 • ネクスト ホップ の所有者 • ネクスト ホップ の優先度 • ネクスト ホップ のメトリック • ネクスト ホップ のタグ • ネクスト ホップ のセグメント 識別子 • ネクスト ホップ のトンネル 識別子 • ネクスト ホップ のカプセル化タイプ • ネクスト ホップ タイプのフラグのビットごとの OR • レイヤ 3 のルーティング情報を検証する： • ルートの VRF 名 • ルート プレフィックス

Path Type	サブスクリプション ベースライン	イベント通知 (Event Notifications)
		ス アドレス • ルートのマスク長 • ルートのネクスト ホップ数 • イベントの種類 • ネクスト ホップ
MAC	静的およびダイナミック MAC エントリに対して DME から GETALL を実行します。	イベントの追加、更新および削除に関するイベント通知を送信します。次の値は、MAC パスのテレメトリを通じてエクスポートされます： • MAC アドレス (MAC address) • MAC アドレス タイプ • VLAN番号 • インターフェイス名 • イベント タイプ イベント通知では、静的エントリとダイナミック エントリの両方がサポートされています。

Path Type	サブスクリプション ベースライン	イベント通知 (Event Notifications)
隣接	IPv4 および IPv6 隣接関係 (アジャセンシー) を送信します。	イベントの追加、更新および削除に関するイベント通知を送信します。次の値は、隣接関係 (アジャセンシー) パスのテレメトリを通じてエクスポートされます： • IP アドレス • MAC アドレス • インターフェイス名 • 物理インターフェイス名 • VRF 名 • プリファレンス • 隣接の送信元 • 隣接関係 (アジャセンシー) のアドレス ファミリ • 隣接関係 (アジャセンシー) のイベント タイプ

詳細については、Github <https://github.com/CiscoDevNet/nx-telemetry-proto> を参照してください。

注意事項と制約事項

ネイティブ データ 送信元 パス機能には、次の注意事項と制約事項があります。

- RIB、MAC、および隣接関係 (アジャセンシー) のネイティブ データ 送信元パスからのストリーミングの場合、センサー パス プロパティの更新は、**depth**、**query-condition**あるいは、**filter-condition**などのカスタム基準をサポートしません。

ルーティング情報のネイティブ データ 送信元パスの構成

URIB に含まれるすべてのルートに関する情報を送信するルーティング情報のネイティブ データ 送信元パスを構成できます。登録すると、基準値はすべてのルート情報を送信します。ベースラインの後、スイッチがサポートするルーティングプロトコルのルート更新と削除操作について通知が送信されます。RIB 通知で送信されるデータについては、[ネイティブ データ 送信元パス用にストリーミングされるテレメトリ データ \(499 ページ\)](#) を参照してください。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	コンフィギュレーションモードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(conf-tm-sub)# sensor-grp 6 switch(conf-tm-sub)#</pre>	センサー グループを作成します。
ステップ 4	data-source native 例 : <pre>switch(conf-tm-sensor)# data-source native switch(conf-tm-sensor)#</pre>	特定のモデルやデータベースを必要とせずに、ネイティブアプリケーションがストリームデータを使用できるように、データ送信元をネイティブに設定します。
ステップ 5	path rib 例 : <pre>nxosv2(conf-tm-sensor)# path rib nxosv2(conf-tm-sensor)#</pre>	ルートとルートアップデート情報をストリーミングする RIB パスを構成します。
ステップ 6	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループサブモードに入り、接続先グループを構成します。
ステップ 7	ip address ip_addr port port protocol { HTTP gRPC } encoding { JSON GPB GPB-compact } 例 : <pre>switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch(conf-tm-dest)#</pre>	サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データストリームのプロトコルとエンコードを設定します。

	コマンドまたはアクション	目的
	例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch(conf-tm-dest) #</pre> 例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch(conf-tm-dest) #</pre>	
ステップ 8	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 9	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサーグループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。
ステップ 10	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

MAC 情報のネイティブ データ送信元パスの構成

MAC テーブルのすべてのエントリに関する情報を送信する MAC 情報のネイティブ データ送信元パスを構成できます。登録すると、基準値はすべての MAC 情報を送信します。基準値の後、MAC アドレスの追加、更新、および削除操作の通知が送信されます。MAC 通知で送信されるデータについては、[ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ（499 ページ）](#) を参照してください。



(注) 更新または削除イベントの場合、MAC 通知は、IP 隣接関係を持つ MAC アドレスに対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	コンフィギュレーションモードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(conf-tm-sub)# sensor-grp 6 switch(conf-tm-sub)#</pre>	センサー グループを作成します。
ステップ 4	data-source native 例 : <pre>switch(conf-tm-sensor)# data-source native switch(conf-tm-sensor)#</pre>	特定のモデルやデータベースを必要とせずに、ネイティブアプリケーションがストリームデータを使用できるように、データ送信元をネイティブに設定します。
ステップ 5	path mac 例 : <pre>nxosv2(conf-tm-sensor)# path mac nxosv2(conf-tm-sensor)#</pre>	MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。
ステップ 6	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループサブモードに入り、接続先グループを構成します。
ステップ 7	ip address ip_addr port port protocol { HTTP gRPC } encoding { JSON GPB GPB-compact } 例 : <pre>switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch(conf-tm-dest)#</pre>	サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データストリームのプロトコルとエンコードを設定します。

	コマンドまたはアクション	目的
	例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch(conf-tm-dest) #</pre> 例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch(conf-tm-dest) #</pre>	
ステップ 8	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 9	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサーグループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。
ステップ 10	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

すべての MAC 情報のネイティブ データ送信元パスの構成

レイヤ 3 およびレイヤ 2 から、MAC テーブルのすべてのエントリに関する情報を送信する MAC 情報のネイティブ データ 送信元 パスを構成できます。登録すると、基準値はすべての MAC 情報を送信します。基準値の後、MAC アドレスの追加、更新、および削除操作の通知が送信されます。MAC 通知で送信されるデータについては、[ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ（499 ページ）](#) を参照してください。



(注) 更新または削除イベントの場合、MAC 通知は、IP 隣接関係を持つ MAC アドレスに対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	コンフィギュレーションモードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(conf-tm-sub)# sensor-grp 6 switch(conf-tm-sub)#</pre>	センサー グループを作成します。
ステップ 4	data-source native 例 : <pre>switch(conf-tm-sensor)# data-source native switch(conf-tm-sensor)#</pre>	特定のモデルやデータベースを必要とせずに、ネイティブアプリケーションがストリームデータを使用できるように、データ送信元をネイティブに設定します。
ステップ 5	path mac-all 例 : <pre>nxosv2(conf-tm-sensor)# path mac-all nxosv2(conf-tm-sensor)#</pre>	すべての MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。
ステップ 6	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループサブモードに入り、接続先グループを構成します。
ステップ 7	ip address ip_addr port port protocol { HTTP gRPC } encoding { JSON GPB GPB-compact } 例 : <pre>switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch(conf-tm-dest)#</pre>	サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データストリームのプロトコルとエンコードを設定します。

	コマンドまたはアクション	目的
	例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch(conf-tm-dest) #</pre> 例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch(conf-tm-dest) #</pre>	
ステップ 8	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリ サブスクリプションを構成します。
ステップ 9	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサーグループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。
ステップ 10	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

IP 隣接のネイティブ データ パスの構成

スイッチのすべての IPv4 と IPv6 隣接に関する情報を送信する IP 隣接情報のネイティブ データ送信元パスを構成できます。登録すると、基準値はすべての隣接情報を送信します。基準値の後、隣接操作の追加、更新、および削除に関する通知が送信されます。隣接関係通知で送信されるデータについては、[ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ（499 ページ）](#) を参照してください。

始める前に

テレメトリ機能を有効にしていない場合は、ここで有効にします（**feature telemetry**）。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例 : <pre>switch# configure terminal switch(config)#</pre>	コンフィギュレーションモードを入力します。
ステップ 2	telemetry 例 : <pre>switch(config)# telemetry switch(config-telemetry)#</pre>	テレメトリ機能の構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例 : <pre>switch(conf-tm-sub)# sensor-grp 6 switch(conf-tm-sub)#</pre>	センサー グループを作成します。
ステップ 4	data-source native 例 : <pre>switch(conf-tm-sensor)# data-source native switch(conf-tm-sensor)#</pre>	ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。
ステップ 5	path adjacency 例 : <pre>nxosv2(conf-tm-sensor)# path adjacency nxosv2(conf-tm-sensor)#</pre>	IPv4 と IPv6 隣接に関する情報をストリームする隣接パスを構成します。
ステップ 6	destination-group grp_id 例 : <pre>switch(conf-tm-sensor)# destination-group 33 switch(conf-tm-dest)#</pre>	テレメトリ接続先グループサブモードに入り、接続先グループを構成します。
ステップ 7	ip address ip_addr port port protocol { HTTP gRPC } encoding { JSON GPB GPB-compact } 例 : <pre>switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json switch(conf-tm-dest)#</pre> 例 :	サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データストリームのプロトコルとエンコードを設定します。

	コマンドまたはアクション	目的
	<pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb switch(conf-tm-dest) #</pre> 例 : <pre>switch(conf-tm-dest) # ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact switch(conf-tm-dest) #</pre>	
ステップ 8	subscription sub_id 例 : <pre>switch(conf-tm-dest) # subscription 33 switch(conf-tm-sub) #</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリサブスクリプションを構成します。
ステップ 9	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(conf-tm-sub) # snsr-grp 6 sample-interval 5000 switch(conf-tm-sub) #</pre>	センサーグループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。
ステップ 10	dst-group dgrp_id 例 : <pre>switch(conf-tm-sub) # dst-grp 33 switch(conf-tm-sub) #</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで設定した接続先グループと一致する必要があります。

ネイティブ データ ソース パス情報の表示

NX-OS の **show telemetry event collector** コマンドを使用して、ネイティブ データ ソース パスの統計情報とカウンタ、またはエラーを表示できます。

統計情報の表示

show telemetry event collector stats コマンドを発行して、各ネイティブ データ ソース パスの統計情報とカウンタを表示できます。

RIB パスの統計情報の例 :

```
switch# show telemetry event collector stats
```

```
-----
Row ID      Collection Count  Latest Collection Time  Sensor Path(GroupId)
-----
1           4                Mon Jul 01 13:53:42.384 PST rib(1)
switch#
```

MAC パスの統計情報の例：

```
switch# show telemetry event collector stats
```

```
-----
Row ID           Collection Count   Latest Collection Time   Sensor Path(GroupID)
-----
1                 3                   Mon Jul 01 14:01:32.161 PST mac(1)
switch#
```

隣接パスの統計情報の例：

```
switch# show telemetry event collector stats
```

```
-----
Row ID           Collection Count   Latest Collection Time   Sensor Path(GroupID)
-----
1                 7                   Mon Jul 01 14:47:32.260 PST adjacency(1)
switch#
```

エラー カウンタの表示

show telemetry event collector stats コマンドを使用して、すべてのネイティブ データ ソース パスのエラーの合計を表示できます。

```
switch# show telemetry event collector errors
```

```
-----
-
Error Description                               Error Count
-----
-
Dme Event Subscription Init Failures             - 0
Event Data Enqueue Failures                      - 0
Event Subscription Failures                      - 0
Pending Subscription List Create Failures         - 0
Subscription Hash Table Create Failures           - 0
Subscription Hash Table Destroy Failures          - 0
Subscription Hash Table Insert Failures           - 0
Subscription Hash Table Remove Failures           - 0
switch#
```

ストリーミング Syslog

テレメトリ用のストリーミング Syslog について

Cisco NX-OS リリース 9.3(3) 以降、モデル駆動型テレメトリは、YANG をデータソースとして使用する syslog のストリーミングをサポートします。サブスクリプションを作成すると、すべての syslog が基準値として受信者にストリーミングされます。この機能は NX-SDK と連携して、次の syslog パスからのストリーミング syslog データをサポートします。

- Cisco-NX-OS-Syslog-oper:syslog
- Cisco-NX-OS-Syslog-oper:syslog/messages

基準値の後は、syslog イベント通知のみが受信者にストリーミングされます。syslog パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf (GPB)
- JavaScript Object Notation (JSON)

syslog 情報のための YANG データ ソース パスの構成

スイッチで生成されたすべての syslog に関する情報を送信する syslog の syslog パスを構成できます。サブスクライブすると、ベースラインはすべての既存の syslog 情報を送信します。ベースラインの後、通知は、スイッチで生成された新しい syslog に対してのみ送信されます。

始める前に

テレメトリ機能を有効にしていない場合は、**feature telemetry** コマンドで有効にします。

手順

	コマンドまたはアクション	目的
ステップ 1	configure terminal 例： switch# configure terminal	グローバル コンフィギュレーション モードを開始します。
ステップ 2	telemetry 例： switch(config)# telemetry	テレメトリの構成モードに入ります。
ステップ 3	sensor-group sgrp_id 例： switch(config-telemetry)# sensor-group 6	センサー グループを作成します。
ステップ 4	data source data-source-type 例： switch(config-tm-sensor)# data source YANG	データソースを YANG に設定し、ネイティブ YANG ストリーミング モデルを使用して syslog をストリーミングできるようにします。
ステップ 5	path Cisco-NX-OS-Syslog-oper:syslog/messages 例： switch(config-tm-sensor)# path Cisco-NX-OS-Syslog-oper:syslog/messages	スイッチで生成された syslog をストリーミングする syslog パスを設定します。

	コマンドまたはアクション	目的
ステップ 6	destination-group grp_id 例 : <pre>switch(config-tm-sensor)# destination-group 33</pre>	テレメトリ接続先グループサブモードに入り、接続先グループを構成します。
ステップ 7	ip address ip_addr port port protocol {HTTP gRPC} encoding {JSON GPB GPB-compact} 例 : <pre>switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json</pre> 例 : <pre>switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb</pre>	サブスクリプションのテレメトリデータを、指定された IP アドレスとポートにストリーミングするように構成し、データストリームのプロトコルとエンコードを設定します。
ステップ 8	subscription sub-id 例 : <pre>switch(config-tm-dest)# subscription 33</pre>	テレメトリ サブスクリプション サブモードに入り、テレメトリサブスクリプションを構成します。
ステップ 9	snsr-group sgrp_id sample-interval interval 例 : <pre>switch(config-tm-sub)# snsr-group 6 sample-interval 0</pre>	センサーグループを現在のサブスクリプションにリンクし、データサンプリングを 0 に設定して、syslog イベントが発生したときにスイッチがテレメトリ データを送信するようにします。 <i>interval</i> については、0 のみが受け入れ可能な値です。
ステップ 10	dst-group dgrp_id 例 : <pre>switch(config-tm-sub)# dst-grp 33</pre>	接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、 destination-group コマンドで構成した接続先グループとマッチする必要があります。

Syslog パスのテレメトリ データ ストリーミング

送信元パスごとに、次のテーブルは、サブスクリプションが最初に作成されるときの「ベースライン」で、そしてイベントの通知が発生するときに、どんな情報がストリーミングされるかを示しています。

パス	サブスクリプション ベースライン	イベント通知
Cisco-NX-OS-Syslog-oper:syslog/messages	スイッチから既存のすべての syslog をストリーミングします。	スイッチで発生した syslog のイベント通知を送信します。 • message-id • node-name • time-stamp • time-of-day • time-zone • category • message-name • severity • text

syslog パス情報の表示

syslog パスの統計情報とカウンタ、またはエラーを表示するには、Cisco NX-OS の **show telemetry event collector** コマンドを使用します。

統計情報の表示

show telemetry event collector stats コマンドを入力すると、syslog パスごとの統計情報とカウンタを表示できます。

次に、syslog パスの統計情報の例を示します。

```
switch# show telemetry event collector stats
```

```
-----
Row ID           Collection Count  Latest Collection Time  Sensor Path (GroupId)
-----
1                 138                Tue Dec 03 11:20:08.200 PST
Cisco-NX-OS-Syslog-oper:syslog(1)
2                 138                Tue Dec 03 11:20:08.200 PST
Cisco-NX-OS-Syslog-oper:syslog/messages(1)
```

エラー カウンタの表示

show telemetry event collector errors コマンドを使用すると、すべての syslog パスのエラーの合計を表示できます。

```
switch(config-if)# show telemetry event collector errors
```

```
-----
Error Description                               Error Count
-----
Dme Event Subscription Init Failures             - 0
```

```

Event Data Enqueue Failures           - 0
Event Subscription Failures           - 0
Pending Subscription List Create Failures - 0
Subscription Hash Table Create Failures - 0
Subscription Hash Table Destroy Failures - 0
Subscription Hash Table Insert Failures - 0
Subscription Hash Table Remove Failures - 0

```

JSON 出力の例

次に、JSON 出力のサンプルを示します。

```

172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
172.19.216.13 - - [03/Dec/2019 19:38:50] "POST
/network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages HTTP/1.0" 200 -
>>> URL           : /network/Cisco-NX-OS-Syslog-oper%3Asyslog%2Fmessages
>>> TM-HTTP-VER    : 1.0.0
>>> TM-HTTP-CNT    : 1
>>> Content-Type   : application/json
>>> Content-Length : 578
Path => Cisco-NX-OS-Syslog-oper:syslog/messages
      node_id_str   : task-n9k-1
      collection_id : 40
      data_source   : YANG
      data          :
[
  [
    {
      "message-id": 420
    },
    {
      "category": "ETHPORT",
      "group": "ETHPORT",
      "message-name": "IF_UP",
      "node-name": "task-n9k-1",
      "severity": 5,
      "text": "Interface loopback10 is up ",
      "time-of-day": "Dec 3 2019 11:38:51",
      "time-stamp": "1575401931000",
      "time-zone": ""
    }
  ]
]

```

.

KVGPB の出力例

次に KVGPB の出力例を示します。

```

KVGPB Output:
---Telemetry msg received @ 18:22:04 UTC

```

```
Read frag:1 size:339 continue to block on read..
All the fragments:1 read successfully total size read:339
node_id_str: "task-n9k-1"
subscription_id_str: "1"
collection_id: 374
data_gpbkv {
  fields {
    name: "keys"
    fields {
      name: "message-id"
      uint32_value: 374
    }
  }
  fields {
    name: "content"
    fields {
      fields {
        name: "node-name"
        string_value: "task-n9k-1"
      }
      fields {
        name: "time-of-day"
        string_value: "Jun 26 2019 18:20:21"
      }
      fields {
        name: "time-stamp"
        uint64_value: 1574293838000
      }
      fields {
        name: "time-zone"
        string_value: "UTC"
      }
    }
  }
}
```

```

    fields {
      name: "process-name"
      string_value: ""
    }
    fields {
      name: "category"
      string_value: "VSHD"
    }
    fields {
      name: "group"
      string_value: "VSHD"
    }
    fields {
      name: "message-name"
      string_value: "VSHD_SYSLOG_CONFIG_I"
    }
    fields {
      name: "severity"
      uint32_value: 5
    }
    fields {
      name: "text"
      string_value: "Configured from vty by admin on console0"
    }
  }
}

```

•

その他の参考資料

関連資料

関連項目	マニュアル タイトル
VXLAN EVPN のテレメトリ展開の構成例。	[VXLAN EVPN ソリューションのテレメトリ展開 (Telemetry Deployment for VXLAN EVPN Solution)]



第 **V** 部

XML 管理インターフェイス

- [XML 管理インターフェイス](#) (523 ページ)



第 33 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 \(523 ページ\)](#)
- [XML 管理インターフェイスのライセンス要件, on page 525](#)
- [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 525](#)
- [XML 管理インターフェイスを使用, on page 525](#)
- [サンプル XML インスタンスに関する情報 \(540 ページ\)](#)
- [その他の参考資料, on page 547](#)

XML 管理インターフェイスについて

XML 管理インターフェイスについての情報

XML 管理インターフェイスを使用してデバイスを構成できます。インターフェイスは XML ベースのネットワーク構成プロトコル (NETCONF) を使用します。これにより、デバイスを管理し、インターフェイスを介して XML 管理ツールまたはプログラムと通信できます。NETCONF の Cisco NX-OS 導入では、デバイスとの通信にセキュアシェル (SSH) セッションを使用する必要があります。

NETCONF は、リモートプロシージャコール (RPC) メッセージ内にデバイス構成要素を含めることができる XML Schema (XSD) を使用して導入されます。RPC メッセージ内から、デバイスに実行させたいコマンドのタイプに一致する NETCONF 操作の 1 つを選択します。NETCONF を使用して、デバイスで CLI コマンドのセット全体を設定できます。NETCONF の使用については、[NETCONF XML インスタンスの作成, on page 529](#) と [RFC 4741](#) を参照してください。

SSH を介した NETCONF の使用の詳細については、[RFC 4742](#) を参照してください。

このセクションは、次のトピックで構成されています。

NETCONF レイヤ

次のテーブルは NETCONF レイヤのリストです：

Table 26: NETCONF レイヤ

レイヤ	例
トランスポート プロトコル	SSHv2
RPC	RPC、RPC-reply
操作	get-config、edit-config
コンテンツ	show または configuration コマンド

以下は、4 つの NETCONF レイヤの説明です。

- SSH トランスポートプロトコル：クライアントとサーバー間の暗号化接続を提供します。
- RPC タグ：リクエストからの構成コマンドと、それに対応する XML サーバからの応答を導入します。
- NETCONF 操作タグ：構成コマンドのタイプを示します。
- 格納ファイル — 構成する機能の XML 表現を示します。

SSH xmlagent

デバイス ソフトウェアは、SSH バージョン 2 を介した NETCONF をサポートする xmlagent と呼ばれる SSH サービスを提供します。



Note xmlagent サービスは、Cisco NX-OS ソフトウェアでは XML サーバと呼ばれます。

NETCONF over SSH は、クライアントと XML サーバ間の hello メッセージの交換から始まります。最初の交換の後、クライアントは XML 要求を送信し、サーバは XML 応答で応答します。クライアントとサーバは、文字シーケンス>で要求と応答を終了します。この文字シーケンスは XML では有効ではないため、クライアントとサーバはメッセージがいつ終了するかを解釈でき、通信の同期が維持されます。

使用できる XML 構成インスタンスを定義する XML スキーマについては、[NETCONF XML インスタンスの作成](#), on page 529 で説明されています。

XML 管理インターフェイスのライセンス要件

製品	ライセンス要件
Cisco NX-OS	XML 管理インターフェイ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 のイメージにバンドルされており、無料で提供されます。NX-OS ライセンス方式の詳細については、『Cisco NX-OS Licensing Guide』を参照してください。

XML 管理インターフェイスを使用するための前提条件

XML 管理インターフェイスを使用するには、次の前提条件があります。

- クライアント PC に SSHv2 をインストールする必要があります。
- クライアント PC に NETCONF over SSH をサポートする XML 管理ツールをインストールする必要があります。
- デバイスの XML サーバに適切なオプションを設定する必要があります。

XML 管理インターフェイスを使用

このセクションでは、XML 管理インターフェイスを手動で構成して使用する方法について説明します。

**Note**

デバイスのデフォルト設定で XML 管理インターフェイスを使用します。

CLI による SSH および XML サーバー オプションの構成

デフォルトで SSH サーバーが有効になっています。SSH を無効にする場合は、クライアント PC で SSH セッションを開始する前に有効にする必要があります。

XML サーバー オプションを構成して、同時セッションの数とアクティブセッションのタイムアウトを制御できます。XML ドキュメントの検証を有効にして、XML セッションを終了することもできます。



Note XML サーバ タイムアウトはアクティブ セッションだけに適用できます。

SSH の構成の詳細については、ご使用のプラットフォームの Cisco NX-OS セキュリティ構成ガイドを参照してください。

XML コマンドの詳細については、ご使用のプラットフォームの Cisco NX-OS システム マネジメント 構成ガイドを参照してください。

Procedure

- ステップ 1** グローバル コンフィギュレーション モードを開始します。
configure terminal
- ステップ 2** (オプション) XML サーバー構成およびアクティブ XML サーバー セッションに関する情報を表示します。コマンド出力でセッション番号を確認できます。
show xml server status
- ステップ 3** 指定されたサーバー セッションの XML ドキュメントを検証します。
xml server validate all
- ステップ 4** 指定した XML サーバセッションを終了します。
xml server terminate session
- ステップ 5** (オプション) キーを生成できるように SSH サーバーを無効にします。
no feature ssh
- ステップ 6** SSH サーバをイネーブルにします。(デフォルトでは有効になっています。)
feature ssh
- ステップ 7** (オプション) SSH サーバーのステータスを表示します。
show ssh server
- ステップ 8** 許容される XML サーバーのセッション数を設定します。
xml server max-session sessions
 有効な範囲は 1 ～ 8 です。デフォルト値は 8 です。
- ステップ 9** XML サーバー セッションを終了するまでに待機する秒数を設定します。
xml server timeout seconds
 範囲は 1～1200 です。デフォルトは 1200 秒です。
- ステップ 10** (オプション) XML サーバー設定およびアクティブ XML サーバー セッションに関する情報を表示します。

show xml server status

ステップ 11 (任意) 実行コンフィギュレーションをスタートアップ コンフィギュレーションに保存します。

copy running-config startup-config**Example**

次の例は、CLI を通して SSH と XML サーバー オプションを構成する方法の例を示しています：

```
switch# configure terminal
switch(config)# xml server validate all
switch(config)# xml server terminate 8665
switch(config)# no feature ssh
switch(config)# feature ssh server
switch(config)# xml server max-session 6
switch(config)# xml server timeout 1200
switch(config)# copy running-config startup-config
```

SSHv2 セッションの開始

クライアント PC で SSHv2 セッションを開始するには、次のような **ssh2** コマンドを使用します。

```
ssh2 username@ip-address -s xmlagent
```

ログインユーザー名、デバイスの IP アドレス、接続するサービスを入力します。xmlagent サービスは、デバイス ソフトウェアでは XML サーバと呼ばれます。



Note SSH コマンドの構文は、クライアント PC の SSH ソフトウェアによって異なることがあります。

XML サーバから hello メッセージを受信しなかった場合は、次の条件を確認してください。

- デバイスで SSH サーバがイネーブルになっています。
- XML サーバの *max-sessions* オプションは、デバイスへの SSH 接続の数をサポートするのに十分です。
- デバイス上の現用系 XML サーバセッションの一部が使用されていません。

hello メッセージの送信

サーバーが他の要求を処理する前に、hello メッセージを使用して、サーバーに自分の機能をアドバタイズする必要があります。XML サーバへの SSH セッションを開始すると、サーバー

はすぐに **hello** メッセージで応答し、サーバーの機能をクライアントに通知します。このメッセージは、サーバーの機能をクライアントに通知します。XML サーバーは基本機能のみをサポートしているので、クライアントに対しても、基本機能のみのサポートを想定しています。

以下は、サーバーとクライアントからのサンプルの **hello** メッセージです。



Note すべての XML ドキュメントは、`]]>]]>` で終了して、SSH 経由の NETCONF で同期がサポートされるようにする必要があります。

サーバーからの hello メッセージ

```
<?xml version="1.0"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  </capabilities>
  <session-id>25241</session-id>
</hello>]]>]]>
```

クライアントからの hello メッセージ

```
<?xml version="1.0"?>
<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
    <nc:capability>urn:ietf:params:xml:ns:netconf:base:1.0</nc:capability>
  </nc:capabilities>
</nc:hello>]]>]]>
```

XML スキーマ定義 (XSD) ファイルの取得

Procedure

- ステップ 1 switch# feature bash shell
- ステップ 2 switch# run bash
- ステップ 3 bash-3.2\$ cd /isan/etc/schema
- ステップ 4 必要なスキーマを取得します。

XML ドキュメントを XML サーバに送信する

コマンドシェルで開いた SSH セッションを介して XML ドキュメントを XML サーバに送信するには、エディターから XML テキストをコピーして、SSH セッションに貼り付けます。通

常、XML ドキュメントを XML サーバに送信するには自動化されたメソッドを使用しますが、このコピーと貼り付けによる方法を使えば、XML サーバへの SSH 接続を確認できます。

XML ドキュメントを XML サーバに送信する場合のガイドラインは次のとおりです。

- コマンド シェル出力で Hello メッセージテキストを検索して、SSH セッションを開始した直後に XML サーバが Hello メッセージを送信したことを確認します。
- XML 要求を送信する前に、クライアントの Hello メッセージを送信します。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ないことに注意してください。
- XML ドキュメントは常に文字シーケンス `]]>]]>` で終了します。

NETCONF XML インスタンスの作成

NETCONF XML インスタンスは、RPC タグおよび NETCONF 操作タグで XML デバイス要素を囲むことにより作成できます。XML デバイス要素は、使用可能な CLI コマンドを XML フォーマットで囲む機能ベースの XML スキーマ定義 (XSD) ファイルで定義されます。

以下は、フレームワーク コンテキストの NETCONF XML リクエストで使用されるタグです。タグ ラインは次のレター コードでマークキングされています：

- X — XML 宣言
- R — RPC リクエスト タグ
- N — NETCONF 操作タグ
- D — デバイス タグ

NETCONF XML フレームワークのコンテキスト

```
X <?xml version="1.0"?>
R <nc:rpc message-id="1" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
R xmlns="http://www.cisco.com/nxos:1.0:nfcli">
N <nc:get>
N <nc:filter type="subtree">
D <show>
D <xml>
D <server>
D <status/>
D </server>
D </xml>
D </show>
N </nc:filter>
N </nc:get>
R </nc:rpc>]]>]]>
```



Note

任意の XML エディタまたは XML 管理インターフェイス ツールを使用して、XML インスタンスを作成する必要があります。

RPC リクエスト タグ

すべての NETCONF XML インスタンスは、RPC リクエスト タグ `<rpc>` で開始する必要があります。は、`<rpc>` 要素には、メッセージ ID (message-id) 属性があります。message-id 属性は、`<rpc-reply>` 内に複製され、リクエストと応答を関連付けるために使用できます。`<rpc>` ノードもまた、次の XML 名前空間宣言も含まれています。

- NETCONF 名前空間宣言 : `urn:ietf:params:xml:ns:netconf:base:1.0` 名前空間で定義されている `<rpc>` と NETCONF タグは、`netconf.xsd` スキーマ ファイルに存在します。
- デバイスの名前空間宣言 : `<rpc>` と NETCONF タグによってカプセル化されたデバイス タグは、他の名前空間で定義されています。デバイスの名前空間は機能指向です。Cisco NX-OS 機能タグは、さまざまな名前空間で定義されています。RPC リクエスト タグ `<rpc>` は、NFCLI 機能の使用例です。これは、デバイスの名前空間が `xmlns=http://www.cisco.com/nxos:1.0:nfcli` であることを宣言しています。`nfcli.xsd` には、この名前空間の定義が含まれています。詳細については、[XML スキーマ定義 \(XSD\) ファイルの取得, on page 528](#)を参照してください。

例

RPC リクエスト タグ `<rpc>`

```
<nc:rpc message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
...
</nc:rpc>]]>]]>
```

構成リクエスト

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:config>
      <configure>
        <__XML__MODE__exec_configure>
          <interface>
            <ethernet>
              <interface>2/30</interface>
              <__XML__MODE_if-ethernet>
                <__XML__MODE_if-eth-base>
                  <description>
                    <desc_line>Marketing Network</desc_line>
                  </description>
                </__XML__MODE_if-eth-base>
              </__XML__MODE_if-ethernet>
            </ethernet>
          </interface>
        </__XML__MODE__exec_configure>
      </configure>
```

```

    </nc:config>
  </nc:edit-config>
</nc:rpc>]]>]]>

```



Note `_XML_MODE` タグは、NETCONF エージェントによって内部的に使用されます。一部のタグは、特定の `_XML_MODE` の子としてのみ存在します。スキーマ ファイルを調べると、XML で CLI コマンドを表すタグにつながる正しいモード タグを見つけることができます。

NETCONF 動作タグ

NETCONF は、次の構成動作を提供します。

Table 27: 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 クローズ セッション インスタンス, on page 540
commit	候補構成の現在の内容を実行構成として設定します。	NETCONF コミット インスタンス：候補構成機能, on page 545
confirmed-commit	指定された時間に構成を確定するためのパラメータを指定します。この操作の後の、確認タイムアウト期間内に確定操作が行われなかった場合、構成は確定確認操作の前の状態に戻ります。	NETCONF 確認済みコミット インスタンス, on page 546
copy-config	送信元構成データストアの内容をターゲット データストアにコピーします。	NETCONF での構成インスタンスのコピー, on page 541
delete-config	操作がサポートされていません。	—
edit-config	デバイスの実行構成の機能を構成します。この操作は configuration コマンドに使用します。	NETCONF Edit Config インスタンス, on page 542 NETCONF Rollback-On-Error インスタンス, on page 546

NETCONF 動作	説明	例
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 529
get-config	構成の全体または一部を取得します。	NETCONF XML インスタンスの作成, on page 529
kill-session	指定した XML サーバー セッションを閉じます。自分のセッションを閉じることはできません。	NETCONF 強制終了セッションインスタンス, on page 541
ロック	クライアントがデバイスの構成システムをロックできるようにします。	NETCONF ロック インスタンス, on page 544
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 545
検証	構成をデバイスに適用する前に、構成候補の構文エラーおよびセマンティクスエラーをチェックします。	NETCONF 検証機能インスタンス, on page 547

デバイスタグ

XML デバイス要素は、使用可能な CLI コマンドを XML フォーマットで表します。機能固有のスキーマファイルには、その特定の機能の CLI コマンドの XML タグが含まれています。

「[XML スキーマ定義 \(XSD\) ファイルの取得, on page 528](#)」を参照してください。

このスキーマを使用して、XML インスタンスを構築することができます。NETCONF インスタンスのビルドに使用された `nfcli.xsd` スキーマファイルの関連部分です。（[NETCONF XML インスタンスの作成, on page 529](#) を参照）。

xml デバイス タグを表示します。

```
<xs:element name="show" type="show_type_Cmd_show_xml"/>
<xs:complexType name="show_type_Cmd_show_xml">
  <xs:annotation>
    <xs:documentation>to display xml agent information</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice maxOccurs="1">
      <xs:element name="xml" minOccurs="1" type="xml_type_Cmd_show_xml"/>
      <xs:element name="debug" minOccurs="1" type="debug_type_Cmd_show_debug"/>
    </xs:choice>
  </xs:sequence>
```

```
<xs:attribute name="xpath-filter" type="xs:string"/>
<xs:attribute name="uses-namespace" type="nxos:bool_true"/>
</xs:complexType>
```

サーバステータス デバイス タグ

```
<xs:complexType name="xml_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="server" minOccurs="1" type="server_type_Cmd_show_xml"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="server_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>xml agent server</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:choice maxOccurs="1">
<xs:element name="status" minOccurs="1" type="status_type_Cmd_show_xml"/>
<xs:element name="logging" minOccurs="1" type="logging_type_Cmd_show_logging_facility"/>
</xs:choice>
</xs:sequence>
</xs:complexType>
```

デバイスタグの応答

```
<xs:complexType name="status_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="__XML__OPT_Cmd_show_xml__readonly__" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:group ref="og_Cmd_show_xml__readonly__" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:group name="og_Cmd_show_xml__readonly__">
<xs:sequence>
<xs:element name="__readonly__" minOccurs="1" type="__readonly__type_Cmd_show_xml"/>
</xs:sequence>
</xs:group>
<xs:complexType name="__readonly__type_Cmd_show_xml">
<xs:sequence>
<xs:group ref="bg_Cmd_show_xml_operational_status" maxOccurs="1"/>
<xs:group ref="bg_Cmd_show_xml_maximum_sessions_configured" maxOccurs="1"/>
<xs:group ref="og_Cmd_show_xml_TABLE_sessions" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
```



Note

「__XML__OPT_Cmd_show_xml__readonly__」タグはオプションです。このタグは応答を表します。応答の詳細については、[RPC 応答タグ, on page 538](#) を参照してください。

<get> 操作を実行するために使用できるタグを見つけるための |XML オプションを使用できます。以下は |XML オプションの例です。この例は、このデバイスで操作を実行するための名前空間定義タグが `http://www.cisco.com/nxos:1.0:nfcli` であり、要求のビルドのために `nfcli.xsd` ファイルを使用できることがわかります。

NETCONF 操作タグとデバイス タグを RPC タグで囲むことができます。</rpc> 終了タグの後に XML 終了文字シーケンスが続きます。

XML の例

```
Switch#> show xml server status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:nfcli">
<nf:data>
<show>
<xml>
<server>
<status>
<__XML__OPT_Cmd_show_xml__readonly__>
<__readonly__>
<operational_status>
<o_status>enabled</o_status>
</operational_status>
<maximum_sessions_configured>
<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML__OPT_Cmd_show_xml__readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

拡張された NETCONF の操作

Cisco NX-OS は、<exec-command> という名前の <rpc> 操作をサポートします。この操作により、クライアントアプリケーションは CLI の **configuration** コマンドと **show** コマンドを送信し、それらのコマンドへの応答を XML タグとして受信できます。

以下は、インターフェイスの構成に使用されるタグの例です。タグラインは次のレターコードでマークキングされています：

- X — XML 宣言
- R — RPC リクエスト タグ
- EO — 拡張操作

次の表に、操作タグの詳細な説明を示します。

Table 28: 操作タグ

タグ	説明
<exec-command>	CLI コマンドの実行
<cmd>	CLI コマンドが含まれています。コマンドは、 show または configuration コマンドです。複数の構成コマンドは、セミコロン (;) を使用して区切ります。複数の show コマンドはサポートされていませんが、複数の構成コマンドを異なる <cmd> タグに含めて、同じリクエストの一部として送信することはできます。詳細については、<exec-command> を通して送信される構成 CLI コマンドの例を参照してください。

を介して送信される構成コマンドへの応答 <cmd> タグは次のとおりです。

- <nf:ok> : すべての **configuration** コマンドが正常に実行されました。
- <nf:rpc-error> : 一部のコマンドが失敗しました。操作は最初のエラーで停止します。
<nf:rpc-error> サブツリーは、失敗した構成について詳細情報を提供します。失敗したコマンドの前に実行された構成は、実行構成に適用されていることに注意してください。

<exec-command> を通して送信される構成 CLI コマンド

show コマンドは、次の例に示すように、それ自体の <exec-command> インスタンスに含めて送信する必要があります :

```
X <?xml version="1.0"?>
R <nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
EO <nxos:exec-command>
EO <nxos:cmd>conf t ; interface ethernet 2/1 </nxos:cmd>
EO <nxos:cmd>channel-group 2000 ; no shut; </nxos:cmd>
EO </nxos:exec-command>
R </nf:rpc>]]>]]>
```

<exec-command> を通して送信された CLI コマンドへの応答

送信操作に対する応答は次のとおりです :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]>]]>
```

<exec-command> を通して送信された表示 CLI コマンド

次の例は、<exec-command> を通して送信される **show** CLI コマンドによってデータを取得する方法を示しています：

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show interface brief</nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
```

<exec-command> を通して送信された表示 CLI コマンドへの応答

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0"
xmlns:mod="http://www.cisco.com/nxos:1.0:if_manager" message-id="110">
<nf:data>
<mod:show>
<mod:interface>
<mod:__XML__OPT_Cmd_show_interface_brief__readonly__>
<mod:__readonly__>
<mod:TABLE_interface>
<mod:ROW_interface>
<mod:interface>mgmt0</mod:interface>
<mod:state>up</mod:state>
<mod:ip_addr>192.0.2.20</mod:ip_addr>
<mod:speed>1000</mod:speed>
<mod:mtu>1500</mod:mtu>
</mod:ROW_interface>
<mod:ROW_interface>
<mod:interface>Ethernet2/1</mod:interface>
<mod:vlan>--</mod:vlan>
<mod:type>eth</mod:type>
<mod:portmode>routed</mod:portmode>
<mod:state>down</mod:state>
<mod:state_rsn_desc>Administratively down</mod:state_rsn_desc>
<mod:speed>auto</mod:speed>
<mod:ratemode>D</mod:ratemode>
</mod:ROW_interface>
</mod:TABLE_interface>
</mod:__readonly__>
</mod:__XML__OPT_Cmd_show_interface_brief__readonly__>
</mod:interface>
</mod:show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

機能不全の構成

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
```



```

xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 192.0.2.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>

```

コマンドの実行により、インターフェイスの IP アドレスは設定されますが、管理状態は変更されません (**no shut** コマンドは実行されません)。管理状態が変更されない理由は、**no port-channel 2000** コマンドがエラーになったためです。

この `<rpc-reply>` は、ある **show** コマンドが、その **show** コマンドの XML 出力を含む `<cmd>` タグによって送信された結果です。

構成コマンドと表示コマンドを同じに組み合わせることはできません `<exec-command>` インスタンス。次は、同じインスタンスで **config** コマンドと **show** コマンドが組み合わせられた例を示しています。

構成コマンドと表示コマンドの組み合わせ

```

<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>conf t ; interface ethernet 2/1 ; ip address 1.1.1.4/24 ; show xml
server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: cannot mix config and show in exec-command. Config cmds
before the show were executed.
Cmd:show xml server status</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>

```

<exec-command> を通して送信された表示 CLI コマンド

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
  <nxos:exec-command>
    <nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
  </nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
  <nf:rpc-error>
    <nf:error-type>application</nf:error-type>
    <nf:error-tag>invalid-value</nf:error-tag>
    <nf:error-severity>error</nf:error-severity>
    <nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
    </nf:error-message>
    <nf:error-info>
      <nf:bad-element><cmd></nf:bad-element>
    </nf:error-info>
  </nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

NETCONF 応答

クライアントによって送信されるすべての XML 要求に対して、XML サーバーは RPC 応答タグ **<rpc-reply>** で囲まれた XML 応答を送信します。

RPC 応答タグ

次の例は、RPC 応答タグ **<rpc-reply>** を示しています。

RPC 応答タグ<rpc-reply>

```
<nc:rpc-reply message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
  <ok/>
</nc:rpc-reply>]]>]]>
```

RPC 応答エレメント

<ok>、**<data>**、そして **<rpc-error>** の要素は RPC 応答に表示される可能性があります。次の表は、**<rpc-reply>** タグに現れる可能性のある RPC 応答要素について説明しています。

Table 29: 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。

要素	説明
<data>	RPC 要求は正常に完了しました。RPC 要求に関連するデータは、<data> 要素を伴っています。
<rpc-error>	RPC 要求が失敗しました。エラー情報は、<rpc-error> 要素。

データ タグにカプセル化されたタグの解釈

<data> タグ内にカプセル化されたデバイス タグには、要求とそれに続く応答が含まれます。次の例に表示されているように、クライアントアプリケーションは、<readonly> タグ前のタグを全て安全に無視できます：

RPC 応答データ

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
  <nf:data>
    <show>
      <interface>
        <__XML__OPT_Cmd_show_interface_brief__readonly__>
          <__readonly__>
            <TABLE_interface>
              <ROW_interface>
                <interface>mgmt0</interface>
                <state>up</state>
                <ip_addr>xx.xx.xx.xx</ip_addr>
                <speed>1000</speed>
                <mtu>1500</mtu>
              </ROW_interface>
              <ROW_interface>
                <interface>Ethernet2/1</interface>
                <vlan>--</vlan>
                <type>eth</type>
                <portmode>routed</portmode>
                <state>down</state>
                <state_rsn_desc>Administratively down</state_rsn_desc>
                <speed>auto</speed>
                <ratemode>D</ratemode>
              </ROW_interface>
            </TABLE_interface>
          </__readonly__>
        </__XML__OPT_Cmd_show_interface_brief__readonly__>
      </interface>
    </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
```

**Note**

<_XML__OPT.*> と <_XML__BLK.*> はレスポンスに表示され、リクエストで使用されることもあります。これらのタグは NETCONF エージェントによって使用され、<_readonly__> タグの後の応答に存在します。これらは要求内で必須であり、スキーマファイルに従って追加した後、CLI コマンドを表す XML タグを続ける必要があります。

サンプル XML インスタンスに関する情報

XML インスタンスの例

このセクションでは、次の XML インスタンスの例を示します：

- [NETCONF クローズ セッション インスタンス, on page 540](#)
- [NETCONF 強制終了セッション インスタンス, on page 541](#)
- [NETCONF での構成インスタンスのコピー, on page 541](#)
- [NETCONF Edit Config インスタンス, on page 542](#)
- [NETCONF の Get 構成インスタンス, on page 543](#)
- [NETCONF ロック インスタンス, on page 544](#)
- [NETCONF ロック解除インスタンス, on page 545](#)
- [NETCONF コミット インスタンス：候補構成機能, on page 545](#)
- [NETCONF 確認済みコミット インスタンス, on page 546](#)
- [NETCONF Rollback-On-Error インスタンス, on page 546](#)
- [NETCONF 検証機能インスタンス, on page 547](#)

NETCONF クローズ セッション インスタンス

次の例は、close-session 要求とそれに続く close-session 応答を表示しています。

セッション要求の終了

```
<?xml version="1.0"?>
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:close-session/>
</nc:rpc>]]>]]>
```

クローズセッション応答

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF 強制終了セッション インスタンス

次の例は、セッションの強制終了要求と、セッションの強制終了の応答を示しています。

セッションの強制終了要求

```
<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>]]>]]>
```

セッションの終了応答

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

NETCONF での構成インスタンスのコピー



Note

`<startup/>`は、ソースまたはターゲットのデータストアとしてサポートされていません。コマンドを入力するなどしてコピー操作を実行するには、にフォールバックする必要があります。`<exec-command>`メソッドを使用します。 **startup-config copy running-config startup-config**

次に、設定のコピー要求と、それに続く設定のコピー応答の例を示します。

構成コピーの要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
<target>
<running/>
</target>
<source>
<url>https://user@example.com:passphrase/cfg/new.txt</url>
</source>
</copy-config>
</rpc>
```

設定応答のコピー

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF Edit Config インスタンス



Note 候補データストアを使用した XML edit-config は、バージョン 1.0 の XML 要求ではサポートされていません。ツールで xml を使用して生成できる新しいバージョンでのみサポートされます。

次の例は、NETCONF edit config の使用方法を示しています。

設定リクエストの編集

```
<?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>]]>]]>
```

設定応答の編集

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager" message-id="16">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

`edit-config` の `operation` 属性は、指定された操作が実行される構成のポイントを指定します。操作属性が指定されていない場合、構成は既存の構成データストアにマージされます。操作属性には、次の値を指定できます。

- create
- merge
- delete

Edit Config : 削除操作要求

次の例は、実行構成からインターフェイス Ethernet 0/0 の構成を削除する方法を示しています。

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<default-operation>none</default-operation>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<top xmlns="http://example.com/schema/1.2/config">
<interface xc:operation="delete">
<name>Ethernet0/0</name>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>
```

Edit Config 応答 : 削除操作

次の例は、実行構成のインターフェイス Ethernet 0/0 の構成を編集する方法を示しています。

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

NETCONF の Get 構成インスタンス

次の例は、NETCONF の `get` 構成 の使用方法を示しています :

サブツリー全体を取得するための Get 構成要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
```

```
<filter type="subtree">
<top xmlns="http://example.com/schema/1.2/config">
<users/>
</top>
</filter>
</get-config>
</rpc>]]>]]>
```

クエリの結果を含む Get 構成応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
<type>superuser</type>
<full-name>Charlie Root</full-name>
<company-info>
<dept>1</dept>
<id>1</id>
</company-info>
</user>
<!-- additional <user> elements appear here... -->
</users>
</top>
</data>
</rpc-reply>]]>]]>
```

NETCONF ロック インスタンス

次の例は、ロック要求、成功の応答、および失敗した試行への応答を示しています。

ロック要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<lock>
<target>
<running/>
</target>
</lock>
</rpc>]]>]]>
```

ロックの取得に成功した場合の応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/> <!-- lock succeeded -->
</rpc-reply>]]>]]>
```


ロックの取得に失敗した場合の応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error> <!-- lock failed -->
<error-type>protocol</error-type>
<error-tag>lock-denied</error-tag>
<error-severity>error</error-severity>
<error-message>
Lock failed, lock is already held
</error-message>
<error-info>
<session-id>454</session-id>
<!-- lock is held by NETCONF session 454 -->
</error-info>
</rpc-error>
</rpc-reply>]]>]]>
```

NETCONF ロック解除インスタンス

次の例は、NETCONF ロック解除の使用方法を示しています。

ロック解除要求

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<unlock>
<target>
<running/>
</target>
</unlock>
</rpc>
```

ロック解除要求への応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

NETCONF コミット インスタンス：候補構成機能

次の例は、コミット操作とコミット返信を示しています。

操作をコミット

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit/>
</rpc>
```

返信をコミット

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

NETCONF 確認済みコミット インスタンス

次の例は、確認済みのコミット操作と確認済みのコミット返答を示しています。

確認されたコミット リクエスト

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit>
<confirmed/>
<confirm-timeout>120</confirm-timeout>
</commit>
</rpc>]]>]]>
```

確認されたコミット 応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>]]>]]>
```

NETCONF Rollback-On-Error インスタンス

次の例は、エラー時のロールバックと、この要求への応答を構成する方法を示しています。

Rollback-On-Error の機能

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <error-option>rollback-on-error</error-option>
  </edit-config>
  <config>
    <top xmlns="http://example.com/schema/1.2/config">
      <interface>
        <name>Ethernet0/0</name>
        <mtu>100000</mtu>
      </interface>
    </top>
  </config>
</rpc>]]>]]>
```

Rollback-On-Error への応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

NETCONF 検証機能インスタンス

次の例は、NETCONF 検証機能の使用方法を示しています。文字列
urn:ietf:params:netconf:capability:rollback-on-error:1.0 は、NETCONF 検証機能を指定します。

検証要求

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
<candidate/>
</source>
</validate>
</rpc>]]>]]>
```

検証要求への応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

その他の参考資料

ここでは、XML 管理インターフェイスの実装に関する追加情報について説明します。

RFC

RFC	タイトル
RFC 4741	NETCONF Configuration Protocol
RFC 4742	セキュアシェル（SSH）経由の NETCONF 構成プロトコルの使用



付録 **A**

ストリーミング テレメトリの送信元

この章は次のトピックで構成されています。

- [ストリーミング テレメトリについて](#) (549 ページ)
- [注意事項と制約事項](#) (549 ページ)
- [テレメトリで利用可能なデータ](#) (549 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。

注意事項と制約事項

ストリーミング テレメトリの注意事項と制約事項は次のとおりです。

- サポートされるプラットフォームの詳細については、[Nexus Switch Platform Matrix](#) を参照してください。
- メモリが 8 GB 未満の Cisco Nexus スイッチは、テレメトリをサポートしていません。

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-APIDME モデルリファレンス](#)の付録にある識別名 (DN) は、一覧表示されたプロパティをテレメトリのデータとして提供できます。



付録 B

WebSocket サブスクリプション

この付録は、次の内容で構成されています。

- [WebSocket サブスクリプション](#) (551 ページ)

WebSocket サブスクリプション

Cisco NX-OS は、スイッチが関係するサブスクリバに通知をプッシュできるようにするインターフェイス機能を提供します。NX-API WebSocket インターフェイスを介して、プログラムとエンドユーザはスイッチのさまざまな状態変化に関する通知を受信できるため、定期的なポーリングの必要がなくなります。

Cisco NX-API REST インターフェイスを使用して API クエリを実行する場合、特定のクエリの結果における将来の変更に対するサブスクリプションを作成するオプションがあります。ユーザーまたはシステムにより開始されたアクションによって、管理オブジェクト (MO) が作成、変更、または削除されると、イベントが生成されます。受信したイベントがサブスクライブ済みのクエリの結果を変更すると、スイッチはサブスクリプションを作成した API クライアントへのプッシュ通知を生成します。

•

WebSocket を開く

API サブスクリプション機能は、WebSocket プロトコル (RFC 6455) を使用して、API クライアントとの双方向接続を実装します。このようにして、API は非送信請求通知メッセージをクライアント自体に送信できます。通知チャネルを確立するには、まずそれぞれの API との WebSocket 接続をオープンする必要があります。各スイッチ内で複数のクエリ サブスクリプションをサポートするために必要な WebSocket 接続は 1 つだけです。WebSocket 接続は (トークン検証によって) API セッション接続に依存しており、API セッションが終了すると閉じます。

WebSocket 接続を開く方法は多数あります。Python クライアントは次のように記述できます。

```
from websocket import create_connection

connection_string = "ws:// 10.1.2.3/socket{0}".format(token)

ws = create_connection(connection_string, sslopt={"check_hostname": False})
```

URI では、トークンは現在の API セッション トークン（クッキー）です。次に、トークン付きの URI の例を示します。

```
ws://10.1.2.3/socketGkZl5NLRZJl5+jqChouaZ9CYjgE58W/pMccR+LeXmdO0obG9NB
Iwo1VBo7+YC1oiJL9mS6I9qh62BkX+Xddhe0JYrTmSG4JcKZ4t3bcP2Mxy3VBmgoJjwZ76ZOuf9V9AD6X
l83lyoR4bLBzqbSSU1R2NIgUotCGWjZt5JX6CJF0=
```

サブスクリプションの作成

クエリにサブスクリプションを作成するには、オプション「?subscription=yes」でクエリを実行します。この例では、JSON 形式の sys/intf/phys-[eth1/1] のクエリへのサブスクリプションを作成します。

```
GET http://10.1.1.1/api/mo/sys/intf/phys-[eth1/1].json?subscription=yes
```

クエリ応答には、サブスクリプションの識別子である subscriptionId が含まれています。サブスクリプションを更新し、特定のサブスクリプションからの今後の通知を識別するために使用できます。

```
{"totalCount": "0", "subscriptionId": "18374686685813276673", "imdata": []}
```

受信通知

サブスクリプションからのイベント通知では、サブスクリプション ID と MO の説明を含むデータ構造が提供されます。この JSON の例では、sys/intf/phys-[eth1/1] の説明が「test」に変更されています。

```
{"subscriptionId":["18374686685813276673"],"imdata":[{"l1PhysIf": {"attributes": {"childAction":
"","descr": "test","dn": "sys/intf/phys-[eth1/1]","modTs": "2019-10-18T19:42:29.446+00:00","rn":
"","status": " modified"}}}]}
```

特定のクエリに対して複数のアクティブなサブスクリプションが存在する可能性があるため、通知には複数のサブスクリプション ID を含めることができます。上記の例と同様です。通知は、JSON または XML フォーマットでサポートされています。

サブスクリプションの更新

イベント通知を受信し続けるには、API セッション中に各サブスクリプションを定期的に更新する必要があります。サブスクリプションを更新するには、例のように、subscriptionId に等しいパラメータ id を使用して、HTTP GET メッセージを API メソッド subscriptionRefresh に送信します。

```
GET http://10.1.1.1/api/subscriptonRefresh.json?id=18374686685813276673
```

サブスクリプションが期限切れになっていなければ、API はリフレッシュメッセージに空の応答を返します。



-
- (注) WebSocket サブスクリプションのタイムアウト期間は、デフォルトで 90 秒です。通知が失われるようにするには、サブスクリプションのリフレッシュメッセージを少なくとも 90 秒ごとに送信する必要があります。
-

要約すると、WebSocket は、NX-OS REST API 内でイベント サブスクリプションのパブリッシャとサブスクライバ間の通信を可能にする強力なツールを提供します。



付録 C

プログラマビリティに関する RFC

この付録は、プログラマビリティのためサポートされている RFC の一覧です。

- [プログラマビリティに関する RFC](#) (555 ページ)

プログラマビリティに関する RFC

次の表に、RFC 準拠標準を示します。各 RFC の詳細については、www.ietf.org を参照してください。

表 30: RFC 準拠標準

RFC	タイトル
RFC 5277	NETCONF イベント通知
RFC 6241	ネットワーク設定プロトコル (NETCONF)
RFC 6243	NETCONF の With-defaults 機能 (report-all でのみサポート)

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。