# 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 10.6(x)

最終更新：2026 年 2 月 3 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)

© 2025 Cisco Systems, Inc. All rights reserved.

# 目 次


第 1 章 新機能と更新情報 1

新機能と更新情報 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 3

プログラマビリティ機能のプラットフォーム サポート 3

第 3 章 概要 7

プログラマビリティの概要 7

サポートされるプラットフォーム 8

標準的なネットワーク管理機能 8

高度な自動化機能 8

プログラマビリティのサポート 9

NX-API のサポート 9

Python スクリプティング 9

Tcl スクリプティング 9

Broadcom シェル 9

bash 10

bash シェルへのアクセスと Linux コンテナのサポート 10

ゲスト シェル 10

コンテナ トラッカーのサポート 12

Perl モジュール 12

FPM 並列ルックアップ 14

第 I 部 : シェルとスクリプト化 15

第 4 章 シェルとスクリプト化 17

Bash について 17

注意事項と制約事項 17

同意トークンの有効化 19

Bash へのアクセス 19

権限をルートにエスカレーションする 20

Bash コマンドの例 22

システム統計情報の表示 22

CLI からの Bash の実行 23

機能 RPM の管理 23

RPM インストールの前提条件 23

Bash からの機能 RPM のインストール 24

機能 RPM のアップグレード 25

機能 RPM のダウングレード 25

機能 RPM の消去 26

DME のモジュール性のサポート 27

DME RPM のインストール 27

インストールされている RPM の確認 30

ローカル リポジトリの RPM のクエリ 31

DME RPM のバージョン間ダウングレード 31

ベース RPM へのダウングレード 33

パッチ RPM の管理 36

RPM インストールの前提条件 36

Bash からパッチ RPM を追加する 36

パッチ RPM のアクティブ化 38

パッチ RPM のコミット 40

パッチ RPM の非アクティブ化 40

パッチ RPM の削除 42

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 43

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 44

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 45
Kstack を介してコピー 47
ネイティブ Bash シェルのアプリケーション例 48

第 5 章 ゲスト シェル 51
Guest Shell について 51
Guestshell に関する注意事項と制限事項 52
Guest Shell へのアクセス 60
ゲスト シェルに使用されるリソース 60
ゲストシェルの機能 61
Guest Shell の NX-OS CLI 61
Guest Shell でのネットワーク アクセス 62
ゲスト シェルでのブートフラッシュへのアクセス 65
Guest Shell の Python 65
Guestshell 2.11 の Python 66
Guest Shell バージョン 2.10 までの Python 3（CentOS 7） 66
Guestshell 4 での Python 69
ゲスト シェルでの RPM のインストール 69
仮想サービス ゲスト シェルのセキュリティ ポスチャ 71
[カーネル脆弱性パッチ（Kernel Vulnerability Patches）] 71
[ASLR および X-Space のサポート（ASLR and X-Space Support）] 71
名前空間の分離 71
ルートユーザーの制限 73
リソース管理 74
ゲスト ファイル システムのアクセス制限 74
ゲスト シェルの管理 74
Guest Shell の無効化 79
ゲスト シェルの破棄 80
Guest Shell の有効化 81
ゲスト シェルの複製 82
ゲスト シェル rootfs のエクスポート 83

Guest Shell rootfs のインポート 83
YAML ファイルのインポート 85
show guestshell コマンド 88
仮想サービスと Guest Shell 情報の検証 89
ゲスト シェルからのアプリケーションの永続的な起動 90
Guest Shell からアプリケーションを永続的に起動する手順 91
ゲスト シェルでのサンプル アプリケーション 91
Guest Shell に関する問題のトラブルシューティング 93

第 6 章 **Broadcom シェル** 95
Broadcom シェルについて 95
注意事項と制約事項 95
Broadcom シェル（bcm-shell）へのアクセス 95
CLI API を使用した bcm-shell へのアクセス 95
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 97
ラインカードの bcm-shell へのアクセス 99
Broadcom シェル コマンドの例 99
L2 エントリの表示 99
FM および LC ASIC インスタンスからのルーティング情報の表示 100
スパニング ツリー グループ エントリの表示 100
インターフェイス xe0 の T2 カウンタの表示 100
L3 情報の表示 101

第 7 章 **Python API** 103
Python の使用 103
Cisco Python パッケージ 103
CLI コマンド API の使用 105
CLI からの Python インタープリタの呼び出し 106
表示フォーマット 107
非インタラクティブ Python 108
Embedded Event Manager でのスクリプトの実行 109

Cisco NX-OS ネットワーク インターフェイスとの Python 統合 110
Python による Cisco NX-OS セキュリティ 111
セキュリティとユーザー権限の例 111
スケジューラでスクリプトを実行する例 113

第 8 章 **tcl によるスクリプティング 115**
Tcl について 115
注意事項と制約事項 115
tclsh コマンドのヘルプ 115
tclsh コマンドの履歴 116
tclsh のタブ補完 116
tclsh の CLI コマンド 116
tclsh コマンドの区切り 117
tcl 変数 117
tclquit 117
Tclsh セキュリティ 118
Tclsh コマンドの実行 118
Tclsh コマンドからの Cisco NX-OS モード間の移動 119
tcl の参照 120

第 9 章 **iPXE 123**
iPXE について 123
ネットブート要件 124
iPXE に関する注意事項と制限事項 124
ブート モードの構成 125
ブート順の構成の確認 126

第 10 章 **カーネル スタック 127**
カーネル スタックについて 127
注意事項と制約事項 127
ポート範囲の変更 128

kstack を使用した VXLAN について 129
kstack のための VXLAN のセットアップ 129
kstack での VXLAN のトラブルシューティング 130
ネットデバイスのプロパティの変更 130

第 11 部： アプリケーション 133

第 11 章 サードパーティ製アプリケーション 135
サードパーティ製アプリケーションについて 135
注意事項と制約事項 135
Python2 および依存パッケージのインストール 136
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 136
永続的なサードパーティ RPM 138
VSH からの RPM のインストール 139
パッケージの追加 139
パッケージのアクティブ化 140
パッケージの非アクティブ化 141
パッケージの削除 141
インストール済みパッケージの表示 142
詳細ログの表示 142
パッケージのアップグレード 143
パッケージのダウングレード 143
サードパーティ製アプリケーション 144
NX-OS 144
DevOps 構成管理ツール 144
V9K 145
自動化ツールの教育コンテンツ 145
collectd 145
Ganglia 145
Iperf 145
LLDP 146
Nagios 146

OpenSSH 146
Quagga 146
スプランク 146
tcollector 147
tcpdump 147
TShark 148
サードパーティ製アプリケーション（TPA）のホスティング 148
コンテナベースのサードパーティ アプリケーション（TPA）ホスティングの仕組み 149
注意事項と制約事項 150
サードパーティ製アプリケーション（TPA）コンテナ ホスティングの構成 150
サードパーティ製アプリケーション（TPA）コンテナ ホスティングの確認 151

第 12 章 **Cisco NX-OS での Ansible の使用** 153
前提条件 153
アンシブルについて 153
Cisco Ansible モジュール 154

第 13 章 **Puppet Agent** 155
Puppet について 155
前提条件 156
Puppet エージェント NX-OS 環境 156
ciscopuppet モジュール 156

第 14 章 **Cisco NX-OS でのシェフ クライアントの使用** 159
シェフについて 159
前提条件 160
Chef クライアント NX-OS 環境 160
cisco-cookbook 160

第 15 章 **Nexus アプリケーション開発：Yocto** 163
Yocto について 163
Yocto のインストール 163

第 16 章 **Nexus アプリケーション開発：SDK** 167

Cisco SDK について 167

SDK のインストール 167

インストールと環境の初期化の手順 168

SDK を使用したアプリケーションの構築 169

RPM を使用したアプリケーションのパッケージ化 170

RPM ビルド環境の作成 171

一般的な RPM ビルド手順の使用 172

オプションのプラグインを使用しない collectd RPM の構築例 173

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 174

第 17 章 **NX-SDK** 177

NX-SDK について 177

Go バインディングに関する考慮事項 178

オンボックス（ローカル）アプリケーションについて 179

デフォルト Docker イメージ 179

NX-SDK に関する注意事項と制限事項 179

NX-SDK2.0 について 180

NX-SDK2.5 について 180

リモート アプリケーションについて 181

NX-SDK セキュリティ 181

NX SDK 2.0 のセキュリティ プロファイル 182

第 18 章 **Cisco NX-OS での Docker の使用** 185

Cisco NX-OS での Docker について 185

Docker の注意事項と制限事項 186

Cisco NX-OS 内で Docker コンテナを設定するための前提条件 186

Docker デーモンの開始 187

自動的に起動するように Docker を構成する 187

Docker コンテナの開始: ホスト ネットワーク モデル 188

Docker コンテナの開始: ブリッジ型ネットワーク モデル 189
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 190
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 191
Cisco Nexus Platform Switches Switchover 時に Docker デーモンの永続性を有効にする 192
Docker ストレージ バックエンドのサイズ変更 193
Docker デーモンの停止 195
Docker コンテナ セキュリティ 196
ユーザー[名前空間（namespace）]の分離による Docker コンテナの保護 196
cgroup パーティションの移動 197
Docker のトラブルシューティング 198
Docker の起動が機能不全になる 198
ストレージが不足しているため、Docker が起動に失敗する 198
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） 199
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） 200
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 200
Docker ストレージ バックエンドのサイズ変更が失敗する 201
Docker コンテナがポートで着信トラフィックを受信しない 201
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 201
一般的なトラブルシューティングのヒント 202

第 III 部 : アプリケーション ホスティング 203

第 19 章 アプリケーション ホスティング 205
アプリケーション ホスティングの注意事項と制限事項 205
アプリケーション ホスティングに関する情報 206
アプリケーション ホスティングの必要性 206
アプリケーション ホスティングの概要 207
アプリケーション ホスティングの設定方法 207
アプリケーション ホスティング機能の有効化 207
アプリケーション ホスティング ブリッジ接続の設定 208

アプリケーションのライフサイクル 210
アプリケーションのアップグレード 212
Docker ランタイムオプションの設定 213
管理インターフェイスでのアプリケーション ホスティングの構成 214
アプリケーションのリソース設定の上書き 215
高度なアプリケーション ホスティング機能 216
アプリケーション データのコピー 218
アプリケーション データの削除 218
アプリケーション ホスティング設定の確認 218
アプリケーション ホスティングの設定例 221
例：AppHosting 機能の有効化 221
例：アプリケーション ホスティング ブリッジ接続の構成 221
例：Docker ランタイムオプションの設定 222
例：管理インターフェイスでのアプリケーション ホスティングの構成 222
例：アプリケーションのリソース設定の上書き 222
その他の参考資料 223
アプリケーション ホスティングに関する機能情報 223

第 20 章 **ThousandEyes Enterprise Agent** 225
ThousandEyes Enterprise Agent の概要 225
ThousandEyes Enterprise Agent の前提条件 226
ThousandEyes Enterprise Agent に必要なリソース 226
ThousandEyes Enterprise Agent のインストール 226
注意事項と制約事項 226
ThousandEyes Enterprise Agent のアプリケーション ホスティングの構成 227
ThousandEyes Enterprise Agent のインストール 230
ThousandEyes Enterprise Agent の構成例 232
例：ThousandEyes Enterprise Agent のインストール 232
ThousandEyes Enterprise Agent の設定例 233

第 IV 部： **NX-API** 235

第 21 章 NX-API CLI 237

NX-API CLI について 237

注意事項と制約事項 237

転送 238

メッセージ形式 239

セキュリティ 239

NX-API CLI の使用 240

NX-API で権限を root にエスカレーションする 242

NX-API 管理コマンド 243

NX-API を使用したインタラクティブ コマンドの操作 253

NX-API クライアント認証 253

NX-API クライアント基本認証 253

NX-API のクライアント証明書認証 253

注意事項と制約事項 254

NX-API のクライアント証明書認証の前提条件 256

NX-API クライアント証明書認証の構成 256

証明書認証用の Python スクリプトの例 257

cURL 証明書要求の例 258

証明書認証の検証 259

NX-API リクエスト要素 261

NX-API 応答要素 268

NX-API へのアクセスの制限 268

iptable の更新 269

リロード間で Iptable を永続化する 270

カーネル スタック ACL 271

NX-API 応答コードの表 273

JSON および XML 構造化出力 276

JSON の概要（JavaScript オブジェクト表記） 276

XML および JSON 出力の例 277

サンプル NX-API スクリプト 285

第 22 章 NX-API REST 287

NX-API REST について 287

REST による DME 設定の置換 288

REST Put による DME フル構成置換について 288

注意事項と制約事項 288

REST POST によるプロパティレベルの構成置換 289

REST PUT による機能レベルの構成置換 289

REST PUT の構成置換のトラブルシューティング 291

第 23 章 NX-API 開発者サンドボックス 293

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 293

About the NX-API デベロッパー サンドボックス 293

注意事項と制約事項 294

メッセージ フォーマットとコマンド タイプの構成 296

デベロッパー サンドボックスを使用 299

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 299

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 302

NX-API 開発者サンドボックス：NX-OS リリース 9.2（2）以降 307

About the NX-API デベロッパー サンドボックス 307

注意事項と制約事項 308

メッセージ フォーマットと入力タイプの構成 310

デベロッパー サンドボックスを使用 315

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 315

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 318

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 324

第 V 部： モデル駆動型プログラマビリティ 329

第 24 章 概要 331

モデル駆動型プログラマビリティについて 331

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 332

NX-OS プログラマビリティエージェント 333

NX-OS オブザーバビリティエージェント 333

モデル インフラストラクチャ 333

ネイティブ（デバイス）YANG モデル 334

共通（OpenConfig）YANG モデル 334

その他の YANG 関連資料 334

第 25 章 元の YANG と OpenConfig YANG 335

YANG について 335

Cisco デバイス YANG 336

注意事項と制約事項 336

DME からデバイス YANG への移行 337

OpenConfig YANG について 338

注意事項と制約事項 338

OpenConfig パス 338

ハイスケールデータに関するガイドラインと制約事項 362

OpenConfig サポートの構成 364

YANG バージョンのアップグレードまたはダウングレード 364

YANG 向けの RBAC 365

注意事項と制約事項 365

YANG RBAC の構成 366

YANG のトラブルシューティング 367

第 26 章 NETCONF エージェント 369

NETCONF エージェントについて 369

マニュアルの変更履歴 370
NETCONF に関する注意事項と制限事項 371
NETCONF エージェントの構成 373
SSH を介する NETCONF エージェントの構成 373
TLS を介する NETCONF エージェントの構成 374
キー/証明書の生成の例 375
NETCONF セッションの管理 376
セッションの開始 377
セッションの終了 378
NETCONF Get / Set 379
NETCONF 通知 388
NETCONF 通知について 388
機能交換 389
イベント ストリームの検出 389
サブスクリプションの作成 390
通知を受け取る 391
サブスクリプションの終了 392
NETCONF デバイス YANG RPC 393
NETCONF のモデル駆動型 RPC について 393
ネイティブデバイス RPC の例 394
Netconf クライアントの例 397
サンドボックスを使用した NETCONF ペイロードの生成 397
ncclient を使用した Cisco NX-OS の接続 397
構成データの入手 398
実行構成と運用データの取得 399
新しい構成の作成 399
構成の削除 400
NETCONF エージェントのトラブルシューティング 400
機能ステータスの確認 400
接続性の確認 401
TLS を確認 401

NETCONF エージェントのアカウンティング ログ 402
NETCONF 明示モードについて 404
注意事項と制約事項 404
NETCONF 明示モードの Get/Set 405
CLI を使用して構成を追加する 409

第 27 章 **RESTCONF エージェント** 413
RESTCONF エージェントについて 413
注意事項と制約事項 414
RESTCONF エージェントの設定 414
RESTCONF セッションの管理 415
RESTCONT Get / Set 415
RESTCONF デバイス YANG RPC 417
RESTCONF の操作コマンドについて 417
デバイス YANG RPC の例 418
RESTCONF エージェントのトラブルシューティング 420

第 28 章 **gRPC エージェント** 425
gRPC エージェントについて 425
gRPC エージェントに関するガイドラインと制限事項 425
gRPC エージェントの構成 427
gRPC の構成 427
gRPC クライアント証明書認証の構成 430
gRPC クライアント証明書認証の構成 433
GRPC 向けの NGINX プロキシの構成 434
障害対応 436
機能ステータスの確認 436
接続性の確認 436

第 29 章 **gNMI：管理インターフェイス** 437
gNMI について 437

gNMI に関する注意事項と制限事項 439
gNMI の構成 442
gNMI オプションの構成 442
gNMI RPC 444
Get について 445
Set について 447
Set の拡張機能 447
拡張の Confirm-Commit 448
Confirm-Commit のガイドラインと制限事項 448
コマンドの表示 449
登録について 450
カスタム Syslog ストリームの登録について 454
gNMI のトラブルシューティング 458
gRPC コマンドの構成例 461
デバッグ ログの収集 463
gNMI のアカウンティング ログ 463

第 30 章
**gNOI：操作インターフェイス 467**
gNOI について 467
gNOI のガイドラインと制限事項 469
gNOI の構成 469
System .Proto 469
SetPackage 471
SetPackage のガイドラインと制約事項 471
RPC オプション 472
OS .Proto 478
Cert .Proto 479
ファイル proto 479
Put 480
Factory Reset .Proto 481
FactoryReset 482
Containerz .Proto 482

gNOI のトラブルシューティング 499
gNOI のデバッグ 499
show コマンド 500
出力例 500
Bash コマンド 500
デバッグ ログの収集 501

第 31 章 **gRPC トンネル** 503
gRPC トンネルについて 503
注意事項と制約事項 504
トピック 2.1 504
gRPC トンネルの設定 504
認証なしの gRPC トンネルの構成 504
トラストポイントを使用した gRPC トンネルの構成 506
VRF による gRPC トンネルの構成 508
障害対応 513

第 32 章 **テレメトリ** 515
テレメトリについて 515
テレメトリのコンポーネントと用語 516
テレメトリ プロセスの高可用性 516
テレメトリのライセンス要件 517
テレメトリの注意事項と制限事項 517
CLI を使用したテレメトリの構成 524
CLI による構成 524
YANG パスの頻度の設定 530
CLI を使用したテレメトリの構成例 532
Telemetry Merge サブスクリプションの構成例 536
テレメトリの構成と統計情報の表示 537
NX-API を使用したテレメトリの構成 547
DME のテレメトリ モデル 547

NX- API を使用した構成 549
NX-API を使用したテレメトリの構成例 559
マルチキャスト フロー パスの可視性 562
クラウド スケール ソフトウェア テレメトリ 563
クラウド スケール ソフトウェア テレメトリについて 563
Cloud Scale ソフトウェア テレメトリ メッセージの形式 564
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 564
テレメトリ パス ラベル 565
テレメトリ パス ラベルについて 565
データの投票またはイベントの受信 565
パス ラベル注意事項と制約事項 566
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 567
非ゼロ カウンタのインターフェイス パスの構成 569
動作速度のインターフェイス パスの構成 571
複数のクエリによるインターフェイス パスの構成 572
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 574
電力使用量トラッキング機能の有効化 576
電力消費履歴の表示 577
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 579
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 580
パス ラベル 構成 を確認 582
パス ラベル情報の表示 583
ネイティブ データ送信元パス 585
ネイティブ データ送信元パスについて 585
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 586
注意事項と制約事項 588
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 589
MAC 情報のネイティブ データ送信元パスの構成 591
すべての MAC 情報のネイティブデータ送信元パスの構成 593
IP 隣接のネイティブ データ パスの構成 595
ネイティブ データ ソース パス情報の表示 597

ストリーミング Syslog 598
テレメトリ用のストリーミング Syslog について 598
syslog 情報のための YANG データ ソース パスの構成 599
Syslog パスのテレメトリ データ ストリーミング 600
テレメトリのトラブルシューティング 603
テレメトリ ログとトレース情報の表示 603
その他の参考資料 604

第 33 章 診断と有用性 605
診断と有用性について 605
コマンドの表示 605
デバッグ ログ 607
プログラマビリティ エージェント ログ 607
YANG インフラログ 608
DME ログ 608
CLI を使用したログ構成の・更 609
診断の提案 610
マニュアルの変更履歴 612

第 VI 部： XML 管理インターフェイス 613

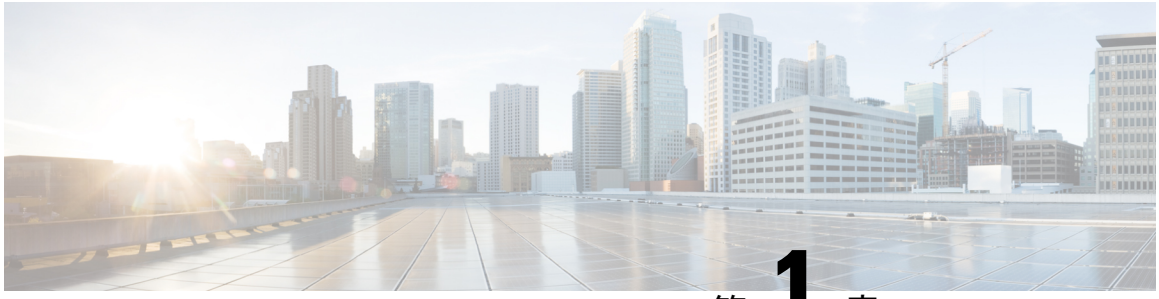
第 34 章 XML 管理インターフェイス 615
XML 管理インターフェイスについて 615
XML 管理インターフェイスについての情報 615
NETCONF レイヤ 615
SSH xmlagent 616
XML 管理インターフェイスのライセンス要件 617
XML 管理インターフェイスを使用するための前提条件 617
XML 管理インターフェイスを使用 617
CLI による SSH および XML サーバー オプションの構成 617
SSHv2 セッションの開始 619

hello メッセージの送信 619
XML スキーマ定義（XSD）ファイルの取得 620
XML ドキュメントを XML サーバに送信する 620
NETCONF XML インスタンスの作成 621
RPC リクエスト タグ 622
NETCONF 動作タグ 623
デバイスタグ 624
拡張された NETCONF の操作 626
NETCONF 応答 630
RPC 応答タグ 630
データ タグにカプセル化されたタグの解釈 631
サンプル XML インスタンスに関する情報 632
XML インスタンスの例 632
NETCONF クロース セッション インスタンス 632
NETCONF 強制終了セッション インスタンス 633
NETCONF での構成インスタンスのコピー 633
NETCONF Edit Config インスタンス 634
NETCONF の Get 構成インスタンス 635
NETCONF ロック インスタンス 636
NETCONF ロック解除インスタンス 637
NETCONF コミット インスタンス：候補構成機能 637
NETCONF 確認済みコミット インスタンス 638
NETCONF Rollback-On-Error インスタンス 638
NETCONF 検証機能インスタンス 639
その他の参考資料 639

付録 A： ストリーミング テレメトリの送信元 641
ストリーミング テレメトリについて 641
注意事項と制約事項 641
テレメトリで利用可能なデータ 641

付録 B： **Websocket サブスクリプション 643**
WebSocket サブスクリプション 643

付録 C： **プログラマビリティに関する RFC 647**
プログラマビリティに関する RFC 647



第 1 章

# 新機能と更新情報

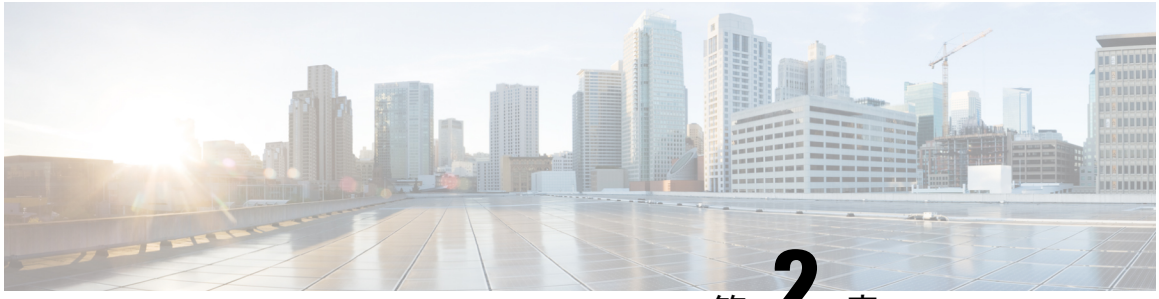

- 新機能と更新情報 （1 ページ）


## 新機能と更新情報

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 |
|---|---|---|---|
| Cisco Live Protect による NX- OS のセキュア化 | テレメトリ機能を使用してエクスポートされる Cisco Live Protect に関連するイベントログのエクスポートのサポートが追加されました。 | 10.6(1)F | テレメトリの注意事項と制限事項 （517 ページ）<br>CLI による構成 （524 ページ） |
| gRPC のパスワードレス UDS | ローカルプロセスが、ユーザー名またはパスワードなしで、UNIX ドメイン ソケット（UDS）を介して gRPC エージェントに接続できるようにします。 | 10.6(1)F | gRPC エージェントに関するガイドラインと制限事項 |

| 特長 | 説明 | 変更が行われたリリース | 参照先 |
|---|---|---|---|
| TPA ライフサイクルオーケストレーション用の gNOI コンテナ | TPA ライフサイクルオーケストレーションの gNOI コンテナのサポートが追加されました<br><br>file.put、system.setpackage、containerz のサポートを追加 | 10.6(1)F | gNOI について （467 ページ）<br><br>gNOI のガイドラインと制限事項 （469 ページ）<br><br>System .Proto （469 ページ）<br><br>ファイル proto （479 ページ）<br><br>Put （480 ページ）<br><br>Containerz .Proto （482 ページ）<br><br>Bash コマンド （500 ページ）<br><br>デバッグ ログの収集 （501 ページ） |
| テレメトリのサポート | N9336C-SE1 のテレメトリのサポートを追加しました。 | 10.6(1)F | テレメトリの注意事項と制限事項 （517 ページ） |
| AuditD をサポート | AuditD のサポートが追加されました | 10.6(1)F | Guestshell に関する注意事項と制限事項 （52 ページ） |



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 （3 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 （17 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 |
|---|---|---|---|
| bash アクセスの同意トークン | Cisco Nexus 9000 シリーズ スイッチ | Cisco NX-OS 10.3(2)F | Cisco Nexus 9808 プラットフォーム スイッチ |
| DME のモジュール方式 | Cisco Nexus 9800 シリーズ スイッチ | Cisco NX-OS 10.3(1)F | |

### Chef クライアント


Cisco NX-OS でのシェフ クライアントの使用 （159 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<br>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<br>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 |
| ソフトウェアテレメトリ | Cisco Nexus 9000 プラットフォーム スイッチ | Cisco NX-OS 10.3(1)F | |

### 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 9800 プラットフォーム スイッチ | Cisco NX-OS 10.3(1)F | |
| 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 （287 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 |

### Python API

Python API （103 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

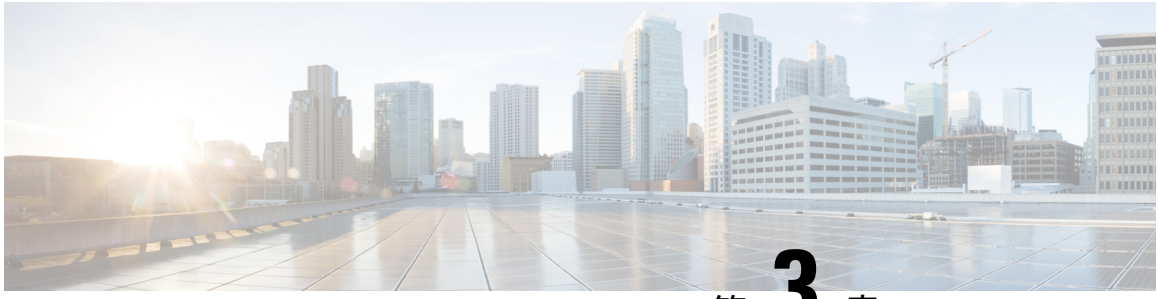
Puppet Agent （155 ページ）に戻ってください。

| 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 |

### TCL スクリプティング

tcl によるスクリプティング （115 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 （7 ページ）
- サポートされるプラットフォーム （8 ページ）
- 標準的なネットワーク管理機能 （8 ページ）
- 高度な自動化機能 （8 ページ）
- プログラマビリティのサポート （9 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

Nexus スイッチ プラットフォーム サポート マトリックスには、次のものがリストされています。

- サポートされている Cisco Nexus 9000 および 3000 スイッチ モデル
- NX- OS ソフトウェア リリース バージョン

プラットフォームと機能の完全なマッピングについては、『Nexusスイッチプラットフォーム サポート マトリックス』を参照してください。

# 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。

# プログラマビリティのサポート

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

## NX-API のサポート

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

## Python スクリプティング

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

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

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

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

## 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 watch***resource* コマンドは、デーモンを使用して要求されたリソース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
- エクスポータ/Heavy.pm
- FileHandle.pm
- ファイル/ベース名.pm
- ファイル/Glob.pm
- ファイル/仕様.pm
- ファイル/仕様/Unix.pm
- ファイル/stat.pm
- Getopt/Std.pm
- IO.pm
- IO/File.pm
- IO/Handle.pm
- IO/Seekable.pm
- IO/Select.pm
- List/Util.pm
- MIME/Base64.pm
- SelectSaver.pm
- Socket.pm
- Symbol.pm
- Sys/Hostname.pm
- 時刻/ハイレゾ.pm
- auto/Data/Dumper/Dumper.so
- auto/File/Glob/Glob.so
- auto/IO/IO.so
- auto/List/Util/Util.so
- auto/MIME/Base64/Base64.so
- auto/Socket/Socket.so
- auto/Sys/Hostname/Hostname.so
- auto/Time/HiRes/HiRes.so

# FPM 並列ルックアップ

Cisco NX-OS リリース 10.4(2)F から、FPM 並列ルックは、Cisco Nexus 93400LD-H1 プラットフォームスイッチでサポートされます。switches.

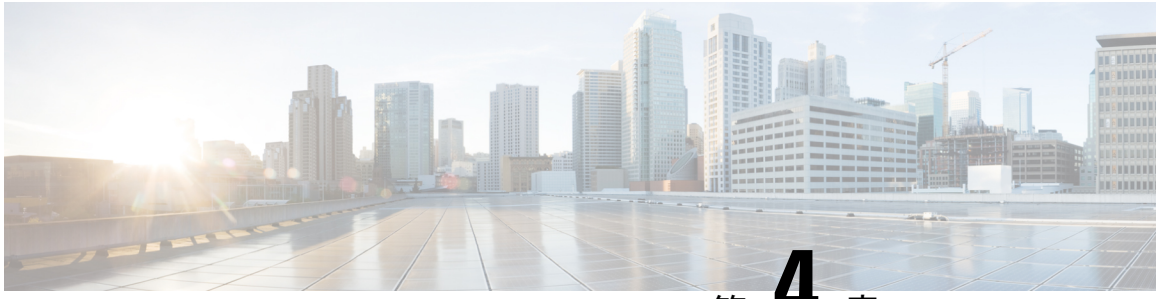
Cisco NX-OS リリース 10.4(2)F から、FPM 並列ルックは、Cisco Nexus N9K-C9364CH1 プラットフォームスイッチでサポートされます。

Nexus 9332D-H2R プラットフォーム スイッチは、ダイ上の 2 つの連続する FPM インスタンス間の並列ルックアップをサポートし、40 個すべての FPM インスタンスをルックアップするためにかかる全体のサイクルを削減します。2 つの FPM インスタンス間の並列ルックアップを有効にするには、2 つのブロック間にルックアップキーまたはインデックスの派生に依存しないようにする必要があります。たとえば、隣接関係インデックスは IP ルートルックアップから取得されるため、一方に IP ルートがあり、もう一方に隣接エントリがある 2 つの FPM インスタンス間では、パラレルルックアップを有効にすることはできません。デフォルトでは、パラレルルックアップ機能はスイッチで有効になっています。この機能を無効にすることはできません。

第 I 部

# シェルとスクリプト化


- シェルとスクリプト化 （17 ページ）
- ゲスト シェル （51 ページ）
- Broadcom シェル （95 ページ）
- Python API （103 ページ）
- tcl によるスクリプティング （115 ページ）
- iPXE （123 ページ）
- カーネル スタック （127 ページ）



第 4 章

# シェルとスクリプト化


- Bash について （17 ページ）
- 注意事項と制約事項 （17 ページ）
- 同意トークンの有効化 （19 ページ）
- Bash へのアクセス （19 ページ）
- 権限をルートにエスカレーションする （20 ページ）
- Bash コマンドの例 （22 ページ）
- 機能 RPM の管理 （23 ページ）
- DME のモジュール性のサポート （27 ページ）
- パッチ RPM の管理 （36 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 （43 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 （44 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 （45 ページ）
- Kstack を介してコピー （47 ページ）
- ネイティブ Bash シェルのアプリケーション例 （48 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**

- Cisco NX-OS リリース 10.3(2)F 以降、bash アクセス機能の同意トークンは、NX-OS でシェルアクセスを有効にするための同意トークンのサポートを提供します。ただし、この機能はトラストアンカーモジュール（TAM）ベースのデバイスでのみ動作します。この機能は、すべての Cisco Nexus 9000 シリーズ プラットフォーム スイッチでサポートされています。ただし、Cisco Nexus 9808 プラットフォーム スイッチを除きます。次の制限が適用されます。

- この設定を無効にするには、Write-Erase リロードが必要です。
- ISSD は、同意トークン機能を備えたリリースでのみサポートされます。
- この設定が有効になっている場合、NX-API/Netconf/Restconf は機能せず、理由を示すエラーが表示されます。
- config-replace は、 コマンドが新しい設定またはファイルに存在する場合にのみ許可されます。
- 同意トークンが有効になっている場合、boot-variable の変更、running-config startup-config のコピー、およびデバイスのリロードは推奨されません。

- Cisco NX-OS リリース 10.5(2)F 以降では、サードパーティ アプリケーション サポート機能を使用して、サードパーティ アプリケーションのクラッシュと終了をモニターできます。クラッシュまたは終了した場合、Cisco NX-OS はアプリケーションを再起動します。

# 同意トークンの有効化

Bash アクセスを制限する同意トークンを有効にするには、次のコマンドを実行します。

**system security consent-token shell-access** [ <*timeout*> ] [**force**]

timeout パラメータに値を指定しなかった場合、デフォルト値の 5 分であると見なされます。timeout パラメータの最大値は 2880 分、つまり 2 日です。

コマンドでこの機能を有効にすると、デバイスは同意トークン セキュアモードになり、ユーザーに与えられるシェルへのアクセス時間は、このコマンドの <*timeout*> パラメータで指定された長さになります。

コマンドは、デフォルトではインタラクティブです。同意トークンモードを強制的に（非インタラクティブ）有効にするには、**force** キーワードを使用します。

（注） セキュリティ上の理由から、このコマンドの **no** 形式を使用してコマンドを無効にすることはできません。したがって、このコマンドを無効にするには、デバイスで write-erase-reload を実行します。

同意トークン機能のステータスを確認するには、**show system security consent-token** コマンドを使用します。

# Bash へのアクセス

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

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

```
switch# show role name 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入力画面が必要な場合、**system security 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 ユーザは、引き続きスイッチで Bash を使用します。

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

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

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

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

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

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

Any use or disclosure, in whole or in part, of the Nexus 9000v Software
or Documentation to any third party for any purposes is expressly
prohibited except as otherwise authorized by Cisco in writing.
```

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

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

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

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

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

# Bash コマンドの例

このセクションには、Bash コマンドと出力の例が含まれています。

## システム統計情報の表示

次の例は、システム統計情報の表示方法を示しています：

```
switch# run bash
bash-4.2$ cat /proc/meminfo
<snip>
MemTotal:       16402560 kB
MemFree:        14098136 kB
Buffers:           11492 kB
Cached:          1287880 kB
SwapCached:            0 kB
Active:          1109448 kB
```

```
Inactive:         717036 kB
Active(anon):     817856 kB
Inactive(anon):   702880 kB
Active(file):     291592 kB
Inactive(file):    14156 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                32 kB
Writeback:             0 kB
AnonPages:        527088 kB
Mapped:            97832 kB
<\snip>
```

## CLI からの Bash の実行

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

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

# 機能 RPM の管理

## RPM インストールの前提条件

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

### 手順の概要

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

### 手順の詳細

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。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | 以下のような出力が表示されれば、続行します。<br>**2018 Mar 27 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready** |
| ステップ 2 | switch# **run bash sudo su**<br>例：<br>`switch# run bash sudo su`<br>`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 dnf installed** \| **grep** *platform* | スイッチにインストールされている NX-OS 機能 RPM のリストを表示します。 |
| ステップ 2 | **dnf list available** | 使用可能な RPM のリストを表示します。 |
| ステップ 3 | **sudo dnf -y install** *rpm* | 使用可能な RPM をインストールします。 |

例

次に、**bfd** RPM をインストールする例を示します。

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

```
bash-4.2$ dnf list available
bgp.lib32_n9000                          1.0.0-r0
bash-4.2$ sudo dnf -y install bfd
```

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

# 機能 RPM のアップグレード

### 始める前に

dnf リポジトリに RPM の上位バージョンが存在する必要があります。

### 手順の概要

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

### 手順の詳細

手順

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

### 例

次に、**bfd** RPM のアップグレードの例を示します。

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

# 機能 RPM のダウングレード

### 手順の概要

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

### 手順の詳細

手順

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

**例**

次に、**bfd** RPM をダウングレードする例を示します。

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

# 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。

### 手順の概要

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

### 手順の詳細

手順

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

**例**

次の例は、**bfd** RPM を消去する方法を示しています：

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

# 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。

Cisco NX-OS リリース 10.3(1)F 以降、DME インフラは、Cisco Nexus 9808 プラットフォーム スイッチでサポートされています。

Cisco NX-OS リリース 10.4(1)F 以降、DME インフラは、Cisco Nexus 9804 プラットフォーム スイッチでサポートされています。

Cisco NX-OS リリース 10.4(1)F 以降、DME は Cisco Nexus 9332D-H2R プラットフォーム スイッチでサポートされます。

Cisco NX-OS リリース 10.4(1)F 以降、DME インフラは、Cisco Nexus 9808 および 9804 スイッチを搭載した N9KX98900CD-A および N9KX9836DM-A ライン カードでサポートされます。

Cisco NX-OS リリース 10.4(2)F 以降、DME は Cisco Nexus 93400LD-H1 プラットフォームスイッチでサポートされます。

Cisco NX-OS リリース 10.4(2)F 以降、DME は Cisco Nexus N9K-C9364C-H1 プラットフォーム スイッチでサポートされます。

## DME RPM のインストール

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

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

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#
```

**dnf install** を使用するには、次のコマンドを実行します。

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

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

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

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

  このオプションには、次に示すようにユーザーの操作が必要です。

  例：

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

Dependencies Resolved

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

Transaction Summary
================================================================================
Upgrade       1 Package

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

```
mgmt/shmetafiles/sharedmeta-ClassRelMetaData
mgmt/shmetafiles/sharedmeta-ChunkMetaData
mgmt/shmetafiles/sharedmeta-ConstPropMetaData
mgmt/shmetafiles/sharedmeta-ConstIdMetaData
mgmt/shmetafiles/sharedmeta-ClassMetaData
mgmt/shmetafiles/sharedmeta-PropRefsMetaData
mgmt/shmetafiles/sharedmeta-SvcMetaData
mgmt/shmetafiles/sharedmeta-ActionContextMetaData
mgmt/shmetafiles/sharedmeta-ConstDefTypeMetaData
mgmt/shmetafiles/sharedmeta-ConstArgMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
mgmt/shmetafiles/sharedmeta-ConstMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
  Cleanup    : dme-2.0.1.0-9.3.1.lib32_n9000                                2/2

Updated:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Complete!
switch-1#
```

# インストールされている RPM の確認

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

## 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#
```

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

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

  例：

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

# ローカル リポジトリの RPM のクエリ

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

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。**installdnf** 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。

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

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

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

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

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

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

Total download size: 45 M
Is this ok [y/N]: y
Retrieving key from file:///etc/pki/rpm-gpg/arm-Nexus9k-dev.gpg
Downloading Packages:
Running Transaction Check
```

```
Running Transaction Test
Transaction Test Succeeded
Running Transaction
/bootflash/.rpmstore/config/etc/pki/rpm-gpg/arm-Nexus9k-dev.gpg
System at HA Standby, running transaction on Standby first
  Installing : dme-2.0.1.0-9.3.1.lib32_n9000                                1/2
starting pre-install package version mgmt for dme
pre-install for dme complete
ln: failed to create symbolic link /var/run/mgmt/sharedmeta-hash: File exists
ln: failed to create symbolic link /var/run/mgmt/dme-objstores.conf: File exists
ln: failed to create symbolic link /var/run/mgmt/samlog.config: File exists
mgmt/
mgmt/shmetafiles/
mgmt/shmetafiles/sharedmeta-ArgMetaData
mgmt/shmetafiles/sharedmeta-RelsMetaData
mgmt/shmetafiles/sharedmeta-ClassRelMetaData
mgmt/shmetafiles/sharedmeta-ChunkMetaData
mgmt/shmetafiles/sharedmeta-ConstPropMetaData
mgmt/shmetafiles/sharedmeta-ConstIdMetaData
mgmt/shmetafiles/sharedmeta-ClassMetaData
mgmt/shmetafiles/sharedmeta-PropRefsMetaData
mgmt/shmetafiles/sharedmeta-SvcMetaData
mgmt/shmetafiles/sharedmeta-ActionContextMetaData
mgmt/shmetafiles/sharedmeta-ConstDefTypeMetaData
mgmt/shmetafiles/sharedmeta-ConstArgMetaData
mgmt/shmetafiles/sharedmeta-ClassNamingMetaData
mgmt/shmetafiles/sharedmeta-ConstMetaData
mgmt/shmetafiles/sharedmeta-PropMetaData
mgmt/shmetafiles/sharedmeta-DnMetaData
  Cleanup    : dme-2.0.10.0-9.3.1.lib32_n9000                               2/2

Removed:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Installed:
  dme.lib32_n9000 0:2.0.1.0-9.3.1

Complete!
```

DME RPM の 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。

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

  例：

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

  インストールされている DME RPM のバージョンを表示します。

# ベース RPM へのダウングレード

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

### 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#
```

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

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

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

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

  例：

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

Dependencies Resolved

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

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

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

Removed:
  dme.lib32_n9000 0:2.0.10.0-9.3.1

Installed:
  dme.lib32_n9000 0:2.0.0.0-9.3.1

Complete!
bash-4.3#
```

ベース DME RPM をインストールします。

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

  例:

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

インストールされているベース DME RPM を表示します。

# パッチ RPM の管理

## RPM インストールの前提条件

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

### 手順の概要

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

### 手順の詳細

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。<br>以下のような出力が表示されれば、続行します。<br>**2018 Mar 27 17:24:22 switch %ASCII-CFG-2-CONF_CONTROL: System ready** |
| ステップ 2 | switch# **run bash sudo su**<br>例：<br>`switch# run bash sudo su`<br>`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 | **dnf list --patch-only** | スイッチに存在するパッチ RPM のリストを表示します。 |
| ステップ 2 | **sudo dnf install --add** *URL_of_patch* | リポジトリにパッチを追加します。ここで、URL_of_patch は、/bootflash/ patch などの標準的な |

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

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

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

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

RPM が 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 dnf install --add
bootflash:/nxos.CSCab00002_CSCab00003-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000.tar
```

```
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
               : protect-packages
groups-repo                                                  | 1.1 kB     00:00 ...
localdb                                                      |  951 B     00:00 ...
patching                                                     |  951 B     00:00 ...
thirdparty                                                   |  951 B     00:00 ...
[###############      ] 70%Install operation 146 completed successfully at Tue Mar 27
21:17:39 2018.

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

# パッチ RPM のアクティブ化

### 始める前に

#unique_64 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 | **sudo dnf install** *patch_RPM* **--nocommit** | パッチ RPM をアクティブにします。ここで、*patch_RPM* はリポジトリにあるパッチです。この手順では、パッチの場所を指定しないでください。<br>（注）<br>--nocommit フラグをコマンドに追加すると、パッチ RPM がこの手順でアクティブになりますが、コミットされません。パッチ RPM をアクティブ化した後にコミットする手順については、パッチ RPM のコミット （40 ページ） を参照してください。 |

### 例

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

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

Dependencies Resolved

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

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

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

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

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

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

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

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

# パッチ RPM のコミット

手順

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

例

次に、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 dnf install nxos.CSCab00001-n9k_ALL-1.0.0-7.0.3.I7.3.lib32_n9000 --commit

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

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

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

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

# パッチ RPM の非アクティブ化

手順

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

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

### 例

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

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

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

Dependencies Resolved

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

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

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

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

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

パッチ RPM は、非アクティブ化した後にコミットする必要があります。パッチ RPM を非アクティブ化した後にコミットしなかった場合に、パッチ RPM の削除 （42 ページ

ジ） の手順を使用してパッチ RPM を削除しようとすると、エラー メッセージが表示されます。

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

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

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

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

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

# パッチ RPM の削除

手順

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

例

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

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

（注） パッチ RPM を削除しようとした後、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、を参照してください。パッチ RPM の非アクティブ化 （40 ページ）

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

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

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

アプリケーションには、/etc/init.d/*application_name* にインストールされる起動 Bash スクリプトが必要です。この起動 Bash スクリプトは、次の一般的な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。

（注）

- Cisco NX-OS リリース 10.5(2)F では、サードパーティ アプリケーションのクラッシュと終了をモニターできます。アプリケーションがクラッシュまたは終了すると、Cisco NX-OS はアプリケーションを再起動します。
- **chkconfig --add***application_name* コマンドを実行した後、 **service** *application_name* **start** コマンドを実行してアプリケーションを起動します。
- アプリケーションを停止する場合、**Linux kill** コマンドを使用しないでください。これは、アプリケーションの再生成を妨げないためです。代わりに、**service** *application_name* **stop** コマンドを使用します。
- アプリケーションが不要な場合は、 **chkconfig --del***application_name* コマンドを使用してアプリケーションを削除します。ただし、このコマンドを使用する前に、必ずアプリケーションを停止してください。

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

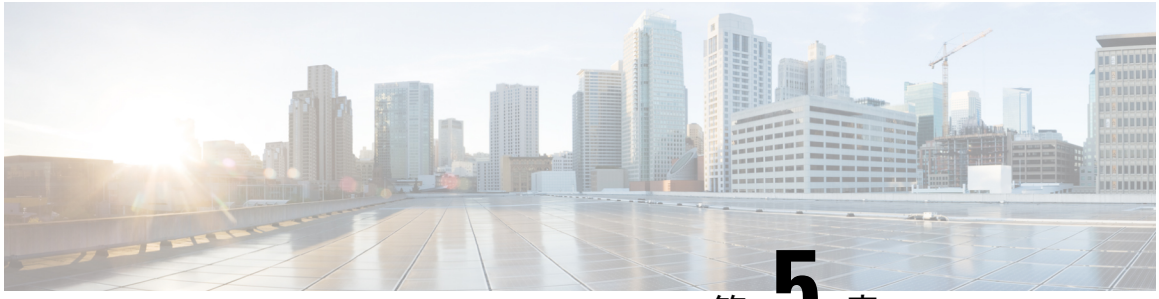
次の例は、ネイティブ Bash シェルのアプリケーションを示しています：

```
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 （51 ページ）
- Guestshell に関する注意事項と制限事項 （52 ページ）
- Guest Shell へのアクセス （60 ページ）
- ゲスト シェルに使用されるリソース （60 ページ）
- ゲストシェルの機能 （61 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 （71 ページ）
- ゲスト ファイル システムのアクセス制限 （74 ページ）
- ゲスト シェルの管理 （74 ページ）
- 仮想サービスと Guest Shell 情報の検証 （89 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 （90 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 （91 ページ）
- ゲスト シェルでのサンプル アプリケーション （91 ページ）
- Guest Shell に関する問題のトラブルシューティング （93 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） デフォルトでは、Guest Shell は、有効にすると約 35 MB の RAM と 350 MB のブートフラッシュを占有します。Guest Shell が使用されていない場合は、guestshell destroy コマンドを使用して技術情報を再利用します。

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

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

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

### AuditD

Cisco NX-OSリリース 10.6(1)F 以降では、**audit monitor guest-shell** を実行して AuditD ログを表示できます。この機能を無効にするには、**no audit monitor guest-shell** コマンドを使用します。

### CentOS のサポート終了と Guestshell への影響

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

### Guestshell 2.11

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

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

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

### Guestshell 3.0

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

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

### Guestshell 4.0

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、RockyLinux 9.2 ベースの lxc コンテナである Guestshell 4.0 が、Guestshell 2.x を置き換える予定です。Guestshell 4.0 は、Cisco NX-OS リリース 10.4.1 およびデフォルト パッケージとしての Guestshell 2.x からダウンロード可能なオプションとして利用できます。Guestshell 4.0 は、次のリリースでデフォルトになります。

（注） Guestshell 4.0 の rootfs サイズは 400 MB です。Guestshell 2.x では 350 MB でした。

### 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 |

| 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 |
|---|---|---|
| 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。

### Jacksonville リリース Guestshell 3.0 からの NX-OS のダウングレード

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

### Guestshell 2.x

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

### Guestshell 2.x から Guestshell 4.x ダウンロード可能 OVA へのアップグレード

Guestshell 4.x は、Cisco の公式ソフトウェア ダウンロード ページからダウンロードでき、command guestshell upgrade コマンドを使用してインストールできます。

次の表に、ゲスト シェルのリリースを示します。

表 *2:* ゲスト シェル リリース

| ゲスト シェル リリース | **NX-OS** がサポートするリリース | サポートされる **Python** のバージョン |
|---|---|---|
| 2.x | 10.4.1 | Python 2.7 および Python 3.6 |
| 3.0 | 10.1.x | python 3.6 |
| 4.x ダウンロード可能 OVA | 10.4.1 | Python 3.9 |

**Guestshell 4.x にアップグレードするには、次のコマンドを使用します**：

- ゲストシェルがインストールされていない場合には、**guestshell enable package <downloaded ova>** コマンドを実行します。
- ゲストシェルがインストールされ、実行されている場合には、**guestshell upgrade package <downloaded ova>** コマンドを実行します。

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

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

互換性のないイメージをインストールした場合の出力例を次に示します。

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

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

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

### 事前設定された SSHD サービス

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

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

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

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

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

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

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

2. Unit と Service の内容を `/usr/lib/systemd/system/ssh.service` ファイルから `sshd-mgmt.service` にコピーします。
3. `sshd-mgmt.service` ファイルを次のように編集します。

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

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

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

```
Port 2222
ListenAddress 10.122.84.34
```

5. 次のコマンドを使用して、systemctl デーモンを開始します。

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

6. （オプション）構成を確認します。

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

7. Guestshell への SSH:

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

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

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

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

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

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

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

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

11. パスワードなしでリモート スイッチに SSH または SCP：

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

### Localtime

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

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

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

# Guest Shell へのアクセス

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

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

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

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

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

# ゲスト シェルに使用されるリソース

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

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

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

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

# ゲストシェルの機能

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

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

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

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

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

## Guest Shell の NX-OS CLI

ゲスト シェルは、ユーザーがゲスト シェル環境からホスト ネットワーク要素に NX-OS コマンドを発行できるようにするアプリケーションを提供します。**dohost** アプリケーションは、有

効な NX-OS 構成または exec コマンドを受け入れ、それらをホスト ネットワーク要素に発行します。

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

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

NX-OS CLI は連鎖させることができます：

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

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

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

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

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

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

# Guest Shell でのネットワーク アクセス

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

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

```
[guestshell@guestshell bootflash]$ ifconfig Eth1-47
Eth1-47: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 13.0.0.47 netmask 255.255.255.0 broadcast 13.0.0.255
```

```
ether 54:7f:ee:8e:27:bc txqueuelen 100 (Ethernet)
RX packets 311442 bytes 21703008 (20.6 MiB)
RX errors 0 dropped 185 overruns 0 frame 0
TX packets 12967 bytes 3023575 (2.8 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

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

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

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

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

chvrf ユーティリティは便宜のために提供されています。

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

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

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

例：

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

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

例：

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

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

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

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

例：

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

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

例：

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

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

例：

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

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

# ゲスト シェルでのブートフラッシュへのアクセス

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

例：

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

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

## Guest Shell の Python

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

例：

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

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

例：

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

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

# Guestshell 2.11 の Python

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

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

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

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

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

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

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

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

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

Complete!

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

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

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

Complete!

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

Complete!
```

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

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

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

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

```
[admin@guestshell ~]$ pip3 install requests --user
Collecting requests
  Downloading
https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl
 (57kB)
    100% |################################| 61kB 211kB/s
Collecting idna<2.9,>=2.5 (from requests)
  Downloading
https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl
 (58kB)
    100% |################################| 61kB 279kB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Downloading
https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl
 (133kB)
    100% |################################| 143kB 441kB/s
```

```
Collecting certifi>=2017.4.17 (from requests)
  Downloading
https://files.pythonhosted.org/packages/b9/63/df50ac98ea0fb006c55a399c3bf1db9da7b5a24de7890bc9cf5bd9e99/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/e8/74/6e4f91745020f967d09332bb2b8b9b10090957334692ab88ea4afe91b77f/urllib3-1.25.8-py2.py3-none-any.whl
 (125kB)
    100% |################################| 133kB 656kB/s
Installing collected packages: idna, chardet, certifi, urllib3, requests
Successfully installed certifi-2019.11.28 chardet-3.0.4 idna-2.8 requests-2.22.0
urllib3-1.25.8
You are using pip version 9.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("https://cisco.com")
<Response [200]>
```

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

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

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

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

Complete!

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

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

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

# Guestshell 4 での Python

Python2 は非推奨になったため、Guestshell 4.0 では使用できません。

Guestshell 4.x は、デフォルトの Python バージョンとして python3.9 をサポートします。

```
[admin@guestshell ~]$ python
Python 3.9.16 (main, Dec  8 2022, 00:00:00)
[GCC 11.3.1 20221121 (Red Hat 11.3.1-4)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
```

# ゲスト シェルでの RPM のインストール

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

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

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

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

Dnf は依存関係を解決し、必要なすべてのパッケージをインストールします。

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

```
"--->" Package nss-softokn-freebl.i686 0:3.16.2.3-9.el7 will be installed
"-->" Finished Dependency Resolution

Dependencies Resolved

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

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

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

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

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

Complete!
```

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

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

# 仮想サービス ゲスト シェルのセキュリティ ポスチャ

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

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

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

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

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

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

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

## 名前空間の分離

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# ルートユーザーの制限

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

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

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

| | | |
|---|---|---|
| CAP_SYS_BOOT | CAP_MKNOD | CAP_SYS_PACCT |
| CAP_SYS_MODULE | CAP_MAC_OVERRIDE | CAP_SYS_RESOURCE |
| CAP_SYS_TIME | CAP_SYS_RAWIO | CAP_AUDIT_WRITE |
| CAP_AUDIT_CONTROL | CAP_SYS_NICE | CAP_NET_ADMIN |
| CAP_MAC_ADMIN | CAP_SYS_PTRACE | |

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

- cap_wake_alarm

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

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

## リソース管理

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

# ゲスト ファイル システムのアクセス制限

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

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

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

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

# ゲスト シェルの管理

以下は、ゲスト シェルを管理するためのコマンドです。

**表 *3*: ゲスト シェル *CLI* コマンド**

| コマンド | 説明 |
|---|---|

| コマンド | 説明 |
| --- | --- |
| **guestshell enable** {**package** [*guest shell OVA file* \| *rootfs-file-URI*]} | • *[ゲストシェルOVAファイル（guest shell OVA file）]* 指定時：<br>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。<br>指定されたソフトウェア パッケージ（OVA ファイル）またはシステム イメージからの組み込みパッケージ（パッケージが指定されていない場合）を使用して、ゲスト シェルをインストールしてアクティブ化します。当初、ゲスト シェル パッケージは、システム イメージに埋め込むことによってのみ利用できます。<br>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。<br>• *rootfs-file-URI* が指定されている場合：<br>ゲスト シェルが破棄された状態のときに、ゲスト シェル **rootfs** をインポートします。このコマンドは、指定されたパッケージでゲスト シェルを起動します。 |
| **guestshell export rootfs package** *destination-file-URI* | ゲスト シェルの **rootfs** ファイルをローカル URI（ブートフラッシュ、USB1 など）にエクスポートします。[（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** [*guest shell OVA file* \| *rootfs-file-URI*]} | • *[ゲストシェルOVAファイル（guest shell OVA file）]* 指定時：<br>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。<br>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。<br>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。<br>• *rootfs-file-URI* が指定されている場合：<br>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。<br>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。<br>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。<br>（注）<br>これは、exec モードで **guestshell disable** コマンドの後に **guestshell enable** コマンドが続くのと同じです。<br>これは、ゲスト シェル内のプロセスが停止しており、再起動する必要がある場合に役立ちます。この **run guestshell** コマンドは、ゲスト シェルで実行されている `sshd` に依存しています。<br>コマンドが機能しない場合は、`sshd` プロセスが誤って停止した可能性があります。NX-OS CLI からゲスト シェルの再起動を実行すると、再起動してコマンドを復元できます。 |
| **guestshell destroy** | ゲスト シェル サービスを非アクティブ化して、アンインストールします。ゲスト シェルに関連付けられているすべての技術情報がシステムに返されます。この **show virtual-service global** コマンドは、これらの技術情報がいつ利用可能になるかを示します。<br>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**<br>**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*<br>**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。<br>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。<br>（注）<br>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。<br>リセットを開始する前に確認を求められます。 |

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

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

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

次の exec キーワードは廃止予定です。

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


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

次の構成キーワードは廃止されます。

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


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

# Guest Shell の無効化

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

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

例：

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

2014 Jul 30 19:47:23 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating virtual
service 'guestshell+'
2014 Jul 30 18:47:29 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated
 virtual service 'guestshell+'
```

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

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

# ゲスト シェルの破棄

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

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

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

switch# guestshell destroy

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

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

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

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

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

# Guest Shell の有効化

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

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

例：

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

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

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

### ベース ブート モードでの Guest Shell の有効化

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

RPM アクティベーション コマンドが次の順序で実行された場合：

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

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

RPM アクティベーション コマンドを逆の順序で実行した場合：

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

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

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

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

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

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

例：

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

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

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

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

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

## ゲスト シェルの複製

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

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

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

## ゲスト シェル rootfs のエクスポート

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

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

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

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

## Guest Shell rootfs のインポート

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

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

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

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

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

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

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

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

（注） rootfs の組み込みバージョンを復元するには：

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

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

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

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

**guestshell enable** のワークフローの例：

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

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

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

## YAML ファイルのインポート

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

Guest Shell インポート YAML ファイルの例：

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

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

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

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

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

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

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

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

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

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

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

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

## show guestshell コマンド

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

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

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

# 仮想サービスと Guest Shell 情報の検証

次のコマンドを使用して、仮想サービスとゲスト シェルの情報を検証できます。

| コマンド | 説明 |
|---|---|
| **show virtual-service global**<br>`switch# show virtual-service global`<br><br>`Virtual Service Global State and Virtualization Limits:`<br><br>`Infrastructure version : 1.11`<br>`Total virtual services installed : 1`<br>`Total virtual services activated : 1`<br><br>`Machine types supported : LXC`<br>`Machine types disabled : KVM`<br><br>`Maximum VCPUs per virtual service : 1`<br><br>`Resource virtualization limits:`<br>`Name Quota Committed Available`<br>`---`<br>`system CPU (%) 20 1 19`<br>`memory (MB) 3840 256 3584`<br>`bootflash (MB) 8192 200 7992`<br>`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**<br><br>`switch# show virtual-service list *`<br><br>`Virtual Service List:`<br><br>`Name Status Package Name`<br>`---`<br>`guestshell+ Activated guestshell.ova`<br>`chef Installed`<br>`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**<br><br>switch# **show guestshell detail**<br>Virtual service guestshell+ detail<br>State : Activated<br>Package information<br>Name : guestshell.ova<br>Path : /isan/bin/guestshell.ova<br>Application<br>Name : GuestShell<br>Installed version : 3.0(0.0)<br>Description : Cisco Systems Guest Shell<br>Signing<br>Key type : Cisco key<br>Method : SHA-1<br>Licensing<br>Name : None<br>Version : None<br>Resource reservation<br>Disk : 400 MB<br>Memory : 256 MB<br>CPU : 1% system CPU<br><br>Attached devices<br>Type Name Alias<br>----------<br>Disk _rootfs<br>Disk /cisco/core<br>Serial/shell<br>Serial/aux<br>Serial/Syslog serial2<br>Serial/Trace serial3 | guestshell パッケージに関する詳細（バージョン、署名リソース、デバイスなど）を表示します。 |

# ゲスト シェルからのアプリケーションの永続的な起動

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

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

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

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

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

# Guest Shell からアプリケーションを永続的に起動する手順

手順

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

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

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

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

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

# ゲスト シェルでのサンプル アプリケーション

次の例は、ゲスト シェルのアプリケーションを示しています。

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

OUTPUTFILE=/tmp/hello

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

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

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

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

リロード後

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

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

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

# Guest Shell に関する問題のトラブルシューティング

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

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

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

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

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

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

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

### ゲスト シェルのルートからブートフラッシュのファイルにアクセスできない

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

ホストから：

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

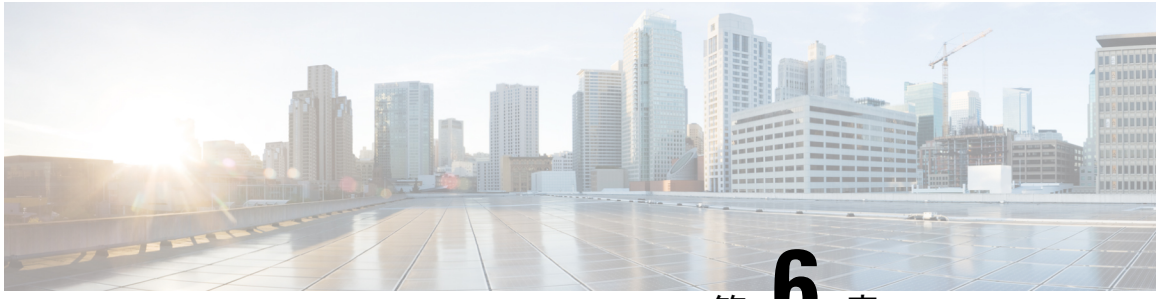
ゲスト シェルから：

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

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

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

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



第 6 章

# Broadcom シェル


- Broadcom シェルについて （95 ページ）
- 注意事項と制約事項 （95 ページ）
- Broadcom シェル（bcm-shell）へのアクセス （95 ページ）
- Broadcom シェル コマンドの例 （99 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

| | |
|---|---|
| *module_number* | シャーシのモジュール番号。 |
| *instance_number* | T2 インスタンス番号<br>• 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。<br>• ワイルドカード（「*」）を指定すると、すべての T2 インスタンスが処理されます。 |
| *command* | Broadcom コマンド |

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

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

次に、T2 インスタンス番号が入力されていない場合の例を示します。

```
switch# bcm-shell module 26 "ports"
Executing ports on bcm shell on module 26
ena/ speed/ link auto STP lrn inter max loop
port link duplex scan neg? state pause discrd ops face frame back
hg0 !ena 42G FD HW No Forward None FA XGMII 16360
hg1 !ena 42G FD HW No Forward None FA XGMII 16360
hg2 !ena 42G FD HW No Forward None FA XGMII 16360
hg3 !ena 42G FD HW No Forward None FA XGMII 16360
hg4 !ena 42G FD HW No Forward None FA XGMII 16360
<snip>
```

次に、モジュール 26、T2 インスタンス 1 の bcm-shell の「ports」表示にアクセスし、パイプを使用して「up」状態のポートのみを表示する例を示します。

```
switch# bcm-shell module 26 "1:ports" | inc up
port link duplex scan neg? state pause discrd ops face frame back
hg9 up 42G FD HW No Forward None FA XGMII 16360
hg10 up 42G FD HW No Forward None FA XGMII 16360
hg11 up 42G FD HW No Forward None FA XGMII 16360
```

次に、T2 インスタンス番号をワイルドカード（「*」）として指定した場合の例を示します。

出力の最初のセットは、T2 インスタンス 0 の結果です。2 番目のセットは、ファブリック モジュールのインスタンス 1 の結果です。

```
switch# bcm-shell module 26 "*:ports" | inc up

port link duplex scan neg? state pause discrd ops face frame back
hg9 up 42G FD HW No Forward None FA XGMII 16360
```

```
hg10 up 42G FD HW No Forward None FA XGMII 16360
hg11 up 42G FD HW No Forward None FA XGMII 16360

port link duplex scan neg? state pause discrd ops face frame back
hg9 up 42G FD HW No Forward None FA XGMII 16360
hg10 up 42G FD HW No Forward None FA XGMII 16360
hg11 up 42G FD HW No Forward None FA XGMII 16360
```

（注） または、スーパーバイザからモジュール（LC または FM）の bcm シェルに直接アクセスできます。次に、スーパーバイザから bcm-shell にアクセスする方法の例を示します。

```
switch# bcm-shell module 3
Entering bcm shell on module 3
Available Unit Numbers: 0 1
bcm-shell.0>
```

# 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。

- 「?」と入力すると、コンテキスト ヘルプが表示され、使用可能なコマンドが表示されます。

```
help: "??" or "help" for summary
Commands common to all modes:
?        ??        ASSert     BackGround  BCM
BCMX     break     BroadSync  CASE        CD
cint     CONFig    CONSole    CoPy        CPUDB
CTEcho   CTInstall CTSetup    DATE        DBDump
DBParse  DeBug     DeBugMod   DELAY       DEVice
```

（注） 大文字はコマンドの省略形を示します。

（注） TAB を使用してコマンド入力を補完する Cisco CLI 機能は使用できません。

- コマンドの省略形を入力します。

```
bcm-shell.1> conf ?
Usage (CONFig): Parameters: show <substring> | refresh |
                            save [filename=<filename>] [pattern=<substring>] |
                            add [<var>=<value>] | delete <var>
      If no parameters are given, displays the current config vars.
      show              display current config vars.
                        Next parameter (optional) maybe a substring to match
      refresh           reload config vars from non-volatile storage
      save              save config vars to non-volatile storage.
                        it can optionally save current config to any given file
                        providing the optional <pattern> will only save variables
matching the pattern
      <var>=<value>     change the value of a config var
      add <var>=<value> create a new config var
      delete <var>      deletes a config var
      clear             deletes all config vars
      =                 prompt for new value for each config var
NOTE: changes are not retained permanently unless saved
```

- 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 ttyp0
```

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

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

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

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

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

# Broadcom シェル コマンドの例

このセクションには、Broadcom シェル コマンドと出力の例が含まれています。

## L2 エントリの表示

FM と LC の L2 エントリを比較するには、T2 インスタンスの L2 エントリを表示します。

- シャーシの LC の T2 インスタンスに接続し、show the l2_mem_entries コマンドを入力します。

```
switch# bcm-shell module 2 “config show l2_mem_entries”
l2_mem_entries=163840
```

- シャーシ内の FM の T2 インスタンスに接続し、show the l2_mem_entries コマンドを入力します。

```
switch# bcm-shell module 24 “config show l2_mem_entries”
l2_mem_entries=32768
```

この例では、T2 ASIC の動作モード設定が LC と FM で異なるため、ASIC には L2 エントリに異なる値が設定されています。

# FM および LC ASIC インスタンスからのルーティング情報の表示

次に、FM インスタンスおよび LC ASIC インスタンスからのルーティング情報を表示する方法を示します。

（注）

- FM モジュール ASIC インスタンスは、最長プレフィックス一致（LPM）エントリを維持します。
- LC モジュール ASIC インスタンスは、ホスト ルートを維持します。

- いずれかの FM ASIC インスタンスの bcm シェルに接続し、**l3 defip show** コマンドを入力します。

```
switch# bcm-shell module 22 "l3 defip show"
Unit 0, Total Number of DEFIP entries: 16385
#  VRF  Net addr        Next Hop Mac       INTF  MODID PORT PRIO CLASS HIT VLAN
0  1    192.168.1.0/24  00:00:00:00:00:00  10000 4     0    0    0     0    n
0  1    192.168.2.0/24  00:00:00:00:00:00  10000 4     0    0    0     0    n
```

- いずれかの LC ASIC インスタンスの bcm シェルに接続し、**l3 defip show** コマンドを入力します。

```
switch# bcm-shell module 2 "l3 defip show"
Unit 2, Total Number of DEFIP entries: 8193
#     VRF       Net addr   Next Hop Mac      INTF  MODID PORT PRIO CLASS HIT VLAN
2048  Override  0.0.0.0/0  00:00:00:00:00:00 10000 4     0    0    0     0   n
```

# スパニング ツリー グループ エントリの表示

次に、スパニングツリーグループ（STG）エントリを表示する方法を示します。

```
switch# bcm-shell module 2 "stg show"
STG 0: contains 7 VLANs (4032-4035,4042,4044,4095)
Forward: xe,hg
STG 1: contains 1 VLAN (4043)
Disable: xe
Forward: hg
STG 4: contains 1 VLAN (1)
Block: xe
Forward: hg
STG 5: contains 1 VLAN (100)
Block: xe
Forward: hg
```

# インターフェイス xe0 の T2 カウンタの表示

次に、前面パネル ポートにマッピングされるインターフェイス xe0 の T2 カウンタを表示する方法を示します。

```
switch# bcm-shell module 2 "show counters xe0"
RDBGC1.xe0 : 38,169  +4
R127.xe0   : 37,994  +4
```

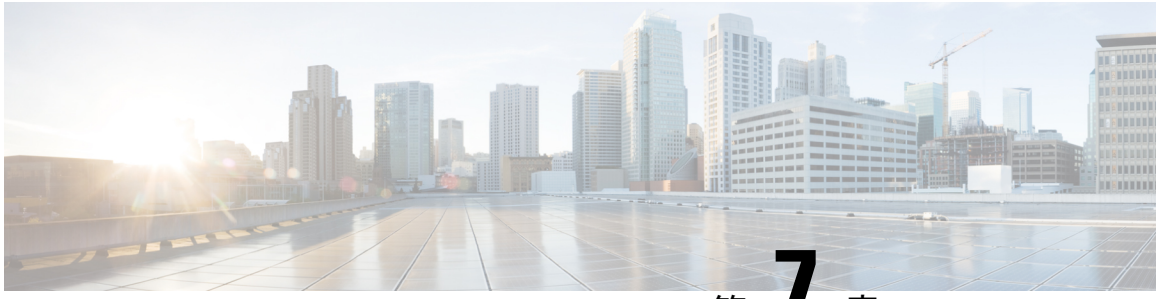
```
R511.xe0   : 16,562  +2
RPKT.xe0   : 55,046  +6
RMCA.xe0   : 54,731  +6
<snip>
```

（注） 出力の右端に「+」で示されている数字は、最後の show コマンドが実行されてからのカウンタ値の変化を示しています。

# L3 情報の表示

次に、T2 から L3 情報を表示する方法を示します。

```
switch# bcm-shell module 2 "l3 l3table show"
Unit 0, free L3 table entries: 147448
Entry  VRF  IP address     Mac Address       INTF    MOD PORT CLASS HIT
1      1    10.100.100.2   00:00:00:00:00:00 100006  0   0    0     n
2      1    10.101.101.2   00:00:00:00:00:00 100006  0   0    0     y
3      1    10.101.101.1   00:00:00:00:00:00 149151  0   0    0     y (LOCAL ROUTE)
4      1    10.100.100.1   00:00:00:00:00:00 149151  0   0    0     n (LOCAL ROUTE)
5      1    10.101.101.0   00:00:00:00:00:00 100000  0   0    0     n (LOCAL ROUTE)
6      1    10.100.100.0   00:00:00:00:00:00 100000  0   0    0     n (LOCAL ROUTE)
7      1    10.101.101.255 00:00:00:00:00:00 149150  0   0    0     n
8      1    10.100.100.255 00:00:00:00:00:00 149150  0   0    0     n
```



第 7 章

# Python API

• Python の使用 （103 ページ）

# 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 *4 : CLI* コマンド *API***

| API | 説明 |
|---|---|
| **cli()**<br>例：<br>`string = cli ("cli-command")` | 制御文字または特殊文字を含む CLI コマンドの未処理の出力を返します。<br>（注）<br>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()**<br>例：<br>`json_string = clid ("cli-command")` | コマンドに XML サポートが存在する場合は、cli-command の JSON 出力を返します。それ以外の場合は、例外がスローされます。<br>（注）<br>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()**<br>例：<br>`clip ("cli-command")` | CLI コマンドの出力を直接 stdout に出力し、Python には何も返されません。<br>（注）<br>`clip ("cli-command")`<br>と同等です（is equivalent to）<br>`r=cli("cli-command")`<br>`print r` |

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

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

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

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

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

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

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

# CLI からの Python インタープリタの呼び出し

次に、CLI から Python 2 を呼び出す方法を表示します：

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

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

次に、CLI から Python 3 を呼び出す方法を表示します：

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

# 表示フォーマット

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

例 1：

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

例 2：

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

例 3：

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

>>>
```

例 4：

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
... print("%30s - %s" % (k,out[k]))
...
header_str - Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright (C) 2002-2020, Cisco and/or its affiliates.
All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under their own
licenses, such as open source. This software is provided "as is," and unless
otherwise stated, there is no warranty, express or implied, including but not
limited to warranties of merchantability and fitness for a particular purpose.
Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or
GNU General Public License (GPL) version 3.0 or the GNU
Lesser General Public License (LGPL) Version 2.1 or
Lesser General Public License (LGPL) Version 2.0.
A copy of each such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://opensource.org/licenses/gpl-3.0.html and
```

```
http://www.opensource.org/licenses/lgpl-2.1.php and
http://www.gnu.org/licenses/old-licenses/library.txt.
bios_ver_str - 07.67
kickstart_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
nxos_ver_str - 9.3(5) [build 9.3(4)IIL9(0.879)]
bios_cmpl_time - 01/29/2020
kick_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
nxos_file_name - bootflash:///nxos.9.3.4.IIL9.0.879.bin
kick_cmpl_time - 5/10/2020 21:00:00
nxos_cmpl_time - 5/10/2020 21:00:00
kick_tmstmp - 05/12/2020 07:08:44
nxos_tmstmp - 05/12/2020 07:08:44
chassis_id - Nexus9000 93180YC-EX chassis
cpu_name - Intel(R) Xeon(R) CPU @ 1.80GHz
memory - 24632252
mem_type - kB
proc_board_id - FDO22280FFK
host_name - switch
bootflash_size - 53298520
kern_uptm_days - 0
kern_uptm_hrs - 0
kern_uptm_mins - 19
kern_uptm_secs - 34
rr_usecs - 641967
rr_ctime - Tue May 12 09:52:28 2020
rr_reason - Reset Requested by CLI command reload
rr_sys_ver - 9.4(1)
rr_service - None
plugins - Core Plugin, Ethernet Plugin
manufacturer - Cisco Systems, Inc.
>>>
```

# 非インタラクティブ Python

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

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

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

```
switch# show file bootflash:scripts/deltaCounters.py
#!/isan/bin/python3
from cli import *
import sys, time
ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'
out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
```

```
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print ('row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast')
print ('===========================================================')
print (' %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc))
print ('===========================================================')
i = 0
while (i < count):
    time.sleep(delay)
    out = json.loads(clid(cmd))
    rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
    rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
    rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
    txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
    txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
    txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
    i += 1
    print ('%-3d %8d %8d %8d %8d %8d %8d' % (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew
 - rxbc, txucNew - txuc, txmcNew - txmc, txbcNew - txbc))
```

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

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

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

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

# Embedded Event Manager でのスクリプトの実行

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

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

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

  ```
  switch# show running-config eem

  !Command: show running-config eem
  ```

```
!Running configuration last done at: Thu Jun 25 15:29:38 2020
!Time: Thu Jun 25 15:33:19 2020

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


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

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

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

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

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

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

PC_VSH_CMD_TLV(7679) with q
```

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

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

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

```
switch# python

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

Python 2.7.11 (default, Jun  4 2020, 09:48:24)
[GCC 4.6.3] on linux2
```

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

# Python による Cisco NX-OS セキュリティ

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

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

## セキュリティとユーザー権限の例

次の例は、特権ユーザーがコマンドを実行する方法を示しています：

Python 3 の例。

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

次の例は、アクセスを拒否されている非特権ユーザーを示しています：

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

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

以下は、特権ユーザーの例です：

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

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

version 9.3(5) Bios:version 07.67

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

以下は、非特権ユーザーの例です：

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

次の例は、RBAC 構成を示しています：

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

```
switch# show role name python-role
```

## スケジューラでスクリプトを実行する例

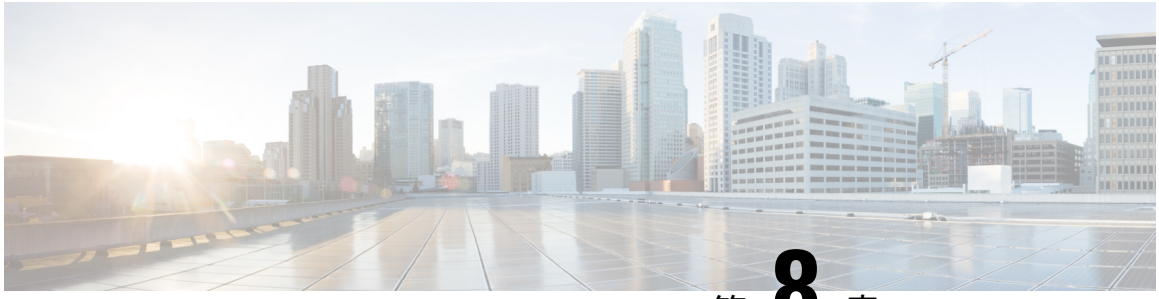
次の例は、スケジューラ機能を使用してスクリプトを実行する Python スクリプトを示しています。

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

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

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

Python 3 の例。

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

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

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

# Save this script in bootflash:///scripts

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

```
Last Execution Time : Sat Jun 13 04:29:38 2020
Last Completion Time: Sat Jun 13 04:29:41 2020
Execution count : 1
----------------------------------------------
Job Name Last Execution Status
----------------------------------------------
testplan Success (0)
==============================================================================
switch#
```



第 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 （115 ページ）
- Tclsh コマンドの実行 （118 ページ）
- Tclsh コマンドからの Cisco NX-OS モード間の移動 （119 ページ）
- tcl の参照 （120 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。

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

### tclsh コマンドのヘルプ

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

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

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

switch-tcl#
```

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

# tclsh コマンドの履歴

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

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

# tclsh のタブ補完

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

# tclsh の CLI コマンド

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

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

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

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

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

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

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

# tclsh コマンドの区切り

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

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

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

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

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

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

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

# tcl 変数

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

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

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

# tclquit

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

# Tclsh セキュリティ

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

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

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

# Tclsh コマンドの実行

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

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

### 手順の概要

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

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **tclsh** ［**bootflash:***filename* ［*argument* **...** ］］<br>例：<br>`switch# tclsh ?`<br>`  <CR>`<br>`  bootflash:  The file to run` | tcl シェルを開始します。<br>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。<br>引数を指定して **tclsh** コマンドを実行すると、最初の引数は、tcl コマンドが記述されたスクリプト ファイルの名前になり、他の引数をスクリプトで変数として使用できます。 |

### 例

次の例は、インタラクティブな Tcl シェルを示しています。

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

switch-tcl# exit
switch#
```

次に、Tcl スクリプトを実行する方法の例を示します。

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

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

switch#
```

# Tclsh コマンドからの Cisco NX-OS モード間の移動

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

### 手順の概要

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

### 手順の詳細

手順

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

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `switch# tclsh`<br>`switch-tcl#` | |
| ステップ 2 | **configure terminal**<br>例：<br>`switch-tcl# configure terminal`<br>`switch(config-tcl)#` | Tcl シェルで Cisco NX-OS のコマンドを実行して、モードを変更します。<br>（注）<br>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**<br>例：<br>`switch-tcl# tclquit`<br>`switch#` | Tcl シェルを終了し、始めのモードに戻ります。 |

**例**

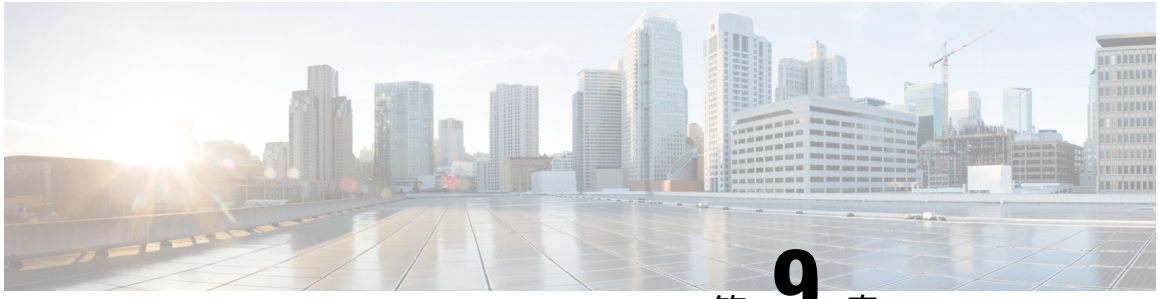
次の例は、対話型 Tcl シェルから Cisco NX-OS モードを変更する方法を示しています：

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

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

# tcl の参照

次のタイトルは、参照のために示されています。

- Mark Harrison (ed)、『*Tcl/Tk Tools*』、O'Reilly Media、ISBN 1-56592-218-2、1997 年
- Mark Harrison および Michael McLennan、『*Effective Tcl/Tk Programming*』、Addison-Wesley、Reading、MA、USA、ISBN 0-201-63474-0、1998 年
- Brent B. Ousterhout、『*Tcl and the Tk Toolkit*』、Addison-Wesley、Reading、MA、USA、ISBN 0-201-63337-X、1994 年
- Brent B. Welch、『*Practical Programming in Tcl and Tk*』、Prentice Hall、Upper Saddle River、NJ、USA、ISBN 0-13-038560-3、2003 年
- J Adrian Zimmer、『*Tcl/Tk for Programmers*』、IEEE Computer Society、John Wiley and Sons により出版、ISBN 0-8186-8515-8、1998 年



第 9 章

# iPXE


- iPXE について （123 ページ）
- ネットブート要件 （124 ページ）
- iPXE に関する注意事項と制限事項 （124 ページ）
- ブート モードの構成 （125 ページ）
- ブート順の構成の確認 （126 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 自動設定バリアントをサポートします。 iPXE は、DHCPv6 オプションのブート URI とパラメータをサポートします。これは、IPv6 ルータ アドバタイズメントに依存します。

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

- bzImage+initramfs/initrd、または ISO 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


- カーネル スタックについて （127 ページ）
- 注意事項と制約事項 （127 ページ）
- ポート範囲の変更 （128 ページ）
- kstack を使用した VXLAN について （129 ページ）
- ネットデバイスのプロパティの変更 （130 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、Nestack 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す。
- 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。

### 手順の概要

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

### 手順の詳細

手順

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

#### 例

次に、kstack ポート範囲を設定する例を示します：

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

#### 次のタスク

コマンドを入力した後は、次の点に注意してください：

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

# kstack を使用した VXLAN について

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

## kstack のための VXLAN のセットアップ

VXLAN EVPN のインターフェイスまたはネットワーク名前空間にサードパーティ アプリケーションからアクセスできるようにするために、追加の構成は必要ありません。VXLAN EVPN ルートは、NX-OS VXLAN EVPN 構成に基づいてカーネルで自動的にプログラムされます。詳

細については、*Cisco Nexus 9000* シリーズ *NX-OS VXLAN* 構成ガイドの「VXLAN BGP EVPN の構成」の章を参照してください。

## kstack での VXLAN のトラブルシューティング

VXLAN の問題をトラブルシューティングするには、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 devveth1-3 lladdr0c:75:bd:07:b4:33 PERMANENT
  ```

# ネットデバイスのプロパティの変更

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

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

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


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

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

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


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

この例では、Eth1-1 は **UP** として示されていますが、**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 （135 ページ）
- Cisco NX-OS での Ansible の使用 （153 ページ）
- Puppet Agent （155 ページ）
- Cisco NX-OS でのシェフ クライアントの使用 （159 ページ）
- Nexus アプリケーション開発：Yocto （163 ページ）
- Nexus アプリケーション開発：SDK （167 ページ）
- NX-SDK （177 ページ）
- Cisco NX-OS での Docker の使用 （185 ページ）

第 11 章

# サードパーティ製アプリケーション


- サードパーティ製アプリケーションについて （135 ページ）
- 注意事項と制約事項 （135 ページ）
- Python2 および依存パッケージのインストール （136 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 （136 ページ）
- 永続的なサードパーティ RPM （138 ページ）
- VSH からの RPM のインストール （139 ページ）
- サードパーティ製アプリケーション （144 ページ）


## サードパーティ製アプリケーションについて

サードパーティのアプリケーションは、Bash シェルで **dnf** コマンドを使用するか、NX-OS CLI を介してネイティブ ホストにインストールされます。

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

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

## 注意事項と制約事項

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

- NX-OS 10.1(1) リリースには、NX-Linux（シスコ独自の Linux ディストリビューション）に基づく新しいオペレーティング システムと rootfs があるため、WRL5/WRL8 を使用して構築されたサードパーティ製RPMはNX-Linux と互換性がない場合がありますそのため、サードパーティ製ソフトウェアが機能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、NX-Linux と互換性のある新しいソフトウェアに置き換えます。 にあります。

- 署名付き RPM のインストールに関するガイドラインと手順については、『Cisco Nexus 9000 Series NX-OS Software Upgrade and Downgrade Guide, Release 9.2(x)』を参照してください。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。
- Cisco NX-OS リリース 10.2(3)F 以降、Python2 および依存 RPM は NX-OS から削除されます。ただし、Python2 および依存する RPM は、devhub サイトからパッケージグループ packagegroup-nxos-64-python-2-deprecated-rpms としてインストールできます。

# Python2 および依存パッケージのインストール

次に、パッケージのインストールの完全なワークフローを示します。

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

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

# サードパーティのネイティブ RPM/パッケージのインストール

パッケージのインストールの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/dnf/repos.d/open-nxos.repo
[open-nxos]
name=open-nxos
```

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

Dependencies Resolved

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

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

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

Installed:
  splunkforwarder.x86_64 0:6.2.3-264376


Complete!
bash-4.2#
```

パッケージが正常にインストールされたかどうかをスイッチに照会し、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# dnf info splunkforwarder
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
Fretta              |  951 B     00:00 ...
```

```
groups-repo          | 1.1 kB     00:00 ...
localdb              |  951 B     00:00 ...
patching             |  951 B     00:00 ...
thirdparty           |  951 B     00:00 ...
Installed Packages
Name        : splunkforwarder
Arch        : x86_64
Version     : 6.2.3
Release     : 264376
Size        : 34 M
Repo        : installed
From repo   : open-nxos
Summary     : SplunkForwarder
License     : Commercial
Description : The platform for machine data.
```

# 永続的なサードパーティ RPM

次に、永続的なサードパーティ RPM の背後にあるロジックを示します。

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

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

変更した iptables を永続化するには、「」を参照してください。
リロード間で Iptable を永続化する （270 ページ）

# VSH からの RPM のインストール

## パッケージの追加

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

### 手順の概要

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

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **show install 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.el5.x86_64.rpm
[####################] 100%
Install operation 314 completed successfully at Thu Aug  6 12:58:22 2015
```

### 次のタスク

パッケージをアクティブ化する準備ができたら、パッケージのアクティブ化（140ページ）に移動します。

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

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

# パッケージのアクティブ化

### 始める前に

RPM は事前に追加しておく必要があります。

## 手順の概要

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

## 手順の詳細

### 手順

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

### 例

次に、パッケージをアクティブ化する例を示します：

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

Inactive Packages:
        sysinfo-1.0.0-7.0.3.x86_64
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
             : protect-packages
Available Packages
chef.x86_64       12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.el5 thirdparty
eigrp.lib32_n9000 1.0.0-r0                                            groups-rep
o
```

```
sysinfo.x86_64    1.0.0-7.0.3                                          patching
switch# install activate chef-12.0-1.el5.x86_64.rpm
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

# パッケージの非アクティブ化

### 手順の概要

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

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **install deactivate** *package-name* | RPM パッケージを非アクティブ化します。 |

例

次に、Chef RPM パッケージを非アクティブ化する例を示します。

```
switch# install deactivate chef
```

# パッケージの削除

始める前に

パッケージを削除する前に非アクティブ化します。非アクティブ化された RPM パッケージのみ削除できます。

### 手順の概要

1. **install remove** *package-name*

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **install remove** *package-name* | RPM パッケージを削除します。 |

例

次に、Chef RPM パッケージを削除する例を示します。

```
switch# install remove chef-12.0-1.el5.x86_64.rpm
```

# インストール済みパッケージの表示

手順の概要

1. **show install packages**

手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **show install packages** | インストールされているパッケージのリストを表示します。 |

例

次に、インストールされているパッケージのリストを表示する例を示します。

```
switch# show install packages
```

# 詳細ログの表示

手順の概要

1. **show tech-support install**

手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **show tech-support install** | 詳細ログを表示します。 |

### 例

次の例は、詳細ログを表示する方法を示しています。

```
switch# show tech-support install
```

# パッケージのアップグレード

### 手順の概要

1. インストール 追加 *package-name* アクティベート アップグレード

### 手順の詳細

手順

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

### 例

次に、パッケージをアップグレードする例を示します：

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate upgrade
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

# パッケージのダウングレード

### 手順の概要

1. インストール 追加 *package-name* アクティベート ダウングレード

### 手順の詳細

手順

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

**例**

次の例は、パッケージをダウングレードする方法を示しています。

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate ?
downgrade  Downgrade package
forced     Non-interactive
upgrade    Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n9000.rpm activate downgrade
[####################] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015
```

# サードパーティ製アプリケーション

## NX-OS

NX-API REST API オブジェクト モデルの仕様の詳細については、https://developer.cisco.com/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

Open NX-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

Nalios は、Nadios Remote Plug-in Executor（NRPE）および SSH または SSL トンネルを介して以下をモニターするオープンソースソフトウェアです。

- ICMP、SNMP、SSH、FTP、HTTP などによるネットワーク サービス
- CPU 負荷、ディスク使用率、システム ログなどのホスト リソース
- サーバー、スイッチ、アプリケーションのアラート サービス
- [サービス（Services）]

詳細については、「https://www.nagios.org/」を参照してください。

# OpenSSH

OpenSSH は、盗聴、接続ハイジャック、およびその他の攻撃を排除するために、すべてのトラフィック（パスワードを含む）を暗号化する SSH 接続ツールのオープンソース バージョンです。OpenSSH は、セキュアなトンネリング機能と複数の認証方式を提供し、すべての SSH プロトコル バージョンをサポートします。

詳細については、「http://www.openssh.com」を参照してください。

# Quagga

Quagga は、さまざまなルーティング プロトコルを実装するネットワーク ルーティング ソフトウェア スイートです。Quagga デーモンは、ネットワーク アクセス可能 CLI（「vty」という）を使用して構成できます。

（注） Quagga BGP のみが検証されています。

詳細については、「http://www.nongnu.org/quagga/」を参照してください。

# スプランク

Splunk は、Web ベースのデータ収集、分析、およびモニタリング ツールであり、ユースケースに合わせて検索、可視化、および事前にパッケージ化されたコンテンツを備えています。raw

データは、Splunk Universal Forwarder を使用して Splunk サーバーに送信されます。ユニバーサルフォワーダは、リモートソースからの信頼性の高いセキュアなデータ収集を可能にし、そのデータをインデックス作成と統合のために Splunk Enterprise に転送します。数万のリモート システムに拡張でき、パフォーマンスへの影響を最小限に抑えながらテラバイト単位のデータを収集できます。

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

# tcollector

tcollector は、ローカル コレクタからデータを収集し、そのデータをオープン時系列データベース（OpenTSDB）にプッシュするクライアント側プロセスです。

tcollector には次の機能があります。

- データ コレクタを実行し、データを照合します。
- 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。

# サードパーティ製アプリケーション（TPA）のホスティング

Cisco NX-OSリリース10.6(2)以降、 Cisco Nexus 9000スイッチは、スイッチ上のLinuxコンテナでのサードパーティアプリケーション（TPA）の直接的なホストをサポートしています。この機能により、管理者および自動化システムは、外部サーバーを必要とせずに、スイッチ上でコンテナ化されたアプリケーションをネイティブに展開、管理、およびモニタできます。

### サードパーティ製アプリケーションのホスティングの利点

この機能は、複数の利点を提供します：

- カスタムおよびベンダーアプリケーションの柔軟なデプロイメント：Linuxベースのアプリケーション（DHCP/TFTPサーバー、syslogコレクタ、モニタリング ツールなど）をスイッチ上で直接コンテナ内で実行でき、ネットワークエッジで幅広い使用例をサポートします。
- 統合管理と自動化：TPAコンテナを構成、監視、管理するためのNX- OS CLIコマンドとプログラム可能なAPI（gNMIやgNOIなど）を提供し、自動化されたワークフローやバックエンド システムとの統合を可能にします。
- 運用の可視性とトラブルシューティング：リアルタイムのステータス、リソース使用率、および実行中のコンテナのログ取得を提供し、トラブルシューティングとパフォーマンス分析を簡素化します。
- 永続的で信頼性の高いアプリケーション運用：重要なアプリケーションをデバイスのアップグレードやリブート後に自動的に再起動するように設定でき、継続的で自動化されたサービス提供をサポートします。

- セキュアでコンプライアンスに準拠した展開のサポート：VRF 対応のデプロイメント、セキュアな通信、設定とログイン情報の永続ストレージなど、シスコのセキュリティと規則遵守のベストプラクティスに準拠しています。
- エッジとクラウドの統合向けに設計：オンボックス アプリケーション ホスティングとクラウドネイティブ管理が必要な Google Distributed Cloud Edge（GDCE）展開などのユースケースに対応します。

## コンテナベースのサードパーティ アプリケーション（TPA）ホスティングの仕組み

Cisco NX-OS リリース 10.6(2) 以降、 Cisco Nexus 9000 スイッチは、統合された Docker エンジンを使用して、サードパーティ アプリケーション（TPA）をスイッチ上で Linux コンテナとして直接ホストできます。

この機能は、NX- OS オペレーティング システム、Docker エンジン、および AppManager 間の統合を活用して、TPA ホスティングのスイッチ上でのコンテナ ライフサイクル管理を提供します。

### process_summary

次のコンポーネントは、コンテナベースの TPA ホスティングで重要な役割を果たします。

- NX- OS：コンテナ化されたアプリケーションの機能の有効化、構成、および監視を管理します。
- Docker エンジン：コンテナ ランタイム環境を提供し、 Linux アプリケーション コンテナのライフサイクルを管理します。
- AppManager：コンテナ制御、ライフサイクル アクション、およびステータス レポートのために NX- OSと Docker 間を調整します。

このプロセスにより、Linux コンテナベースのサードパーティ アプリケーションのセキュアで永続的な自動ホスティングがスイッチ上で直接実行できるようになります。

### process_workflow

コンテナベースのサードパーティ製アプリケーション（TPA）をホストおよび管理するプロセスには、次の段階が含まれます。

1. 機能の有効化：TPA 機能を有効にすると、Docker エンジンが NX- OS 内のマネージド サービスとして起動します。
2. 内部通信：NX- OS と Docker エンジン間のすべての通信に対してループバック インターフェイスが構成され、コンテナのトラフィック分離が保証されます。
3. コンテナのストレージと起動：アプリケーション コンテナ イメージがスイッチに保存され、TPA 構成に従って Docker エンジンによって起動されます。
4. 永続性：永続ストレージ（ホストマウント ボリューム）により、コンテナ データとログイン情報が再起動やアップグレード後も存続します。

5. ライフサイクル管理：AppManagerおよび関連するNX-OSプロセスは、コンテナの開始、停止、再起動、およびステータスの監視を制御します。
6. モニタリングとロギング：NX-OSには、コンテナ ステータス、リソース使用率、およびDockerとコンテナからのログを取得するためのコマンドが用意されています。
7. 自動化とAPI：TPAホスティングは、自動化と外部管理のために、NX-OS CLIおよびプログラム可能なAPI（gNMI/gNOIなど）と統合されています。
8. セキュリティと分離： VRF認識型通信とUNIXドメイン ソケットを使用して、スイッチ内でのセキュアな分離操作を実現します。

## 注意事項と制約事項

- NX-OSとDockerコンテナ間のトランスポートインターフェイスとして使用できるのは、ループバック インターフェイスのみです。このインターフェイスは、TPA通信用に明示的に設定する必要があります。
- 最適なパフォーマンスと信頼性を確保するために、アプリケーションコンテナを実行する場合は、少なくとも6つのCPUコアと64 GBのメモリを備えたスイッチ プラットフォームを使用することをお勧めします。
- 設定ファイル、ログイン情報、および永続的なアプリケーションデータを、専用ボリュームまたはホストマウントディレクトリに保存して、デバイスの再起動やアップグレード後もデータが保持されるようにします。
- 重要なアプリケーションコンテナは、システムのアップグレードまたはリブート後に自動的に再起動するように構成し、重要なサービスの継続的な運用を確保する必要があります。
- コンテナの高度なモニタリングとパフォーマンスの最適化は、現在の実装には含まれていません。ユーザーは、基本的なリソースおよびパフォーマンスの追跡にネイティブのDockerコマンドを使用できます。
- サポートされているDockerのバージョンは20.10.25-ceで、containerd v1.6.19です。

## サードパーティ製アプリケーション（TPA）コンテナ ホスティングの構成

Cisco Nexus 9000スイッチでコンテナベースのサードパーティ アプリケーション（TPA）ホスティングを有効にし、設定するには、次の手順に従います。

**Before you begin**

Dockerサービスの通信に使用するループバック インターフェイスを特定します。

### 手順

**ステップ1** TPA機能を有効化します。

例：

```
feature dockerbox
```

このコマンドは、スイッチで Docker ベースのサードパーティ アプリケーション ホスティングを有効にします。

**ステップ 2** TPA コンフィギュレーション モードに入ります。

例：

```
dockerbox
```

このコマンドにより、スイッチが TPA（Docker Box）構成モードになります。

**ステップ 3** Docker サービス通信のトランスポート インターフェイスを構成します。

例：

```
transport-interface <loopback-interface>
```

Docker サービス通信では、ループバック インターフェイスのみがサポートされます。**loopback-interface** は、選択したループバック インターフェイスで置き換えます。

---

コンテナベースの TPA ホスティングが有効になり、設定されました。スイッチでアプリケーション コンテナを展開および管理できるようになりました。

### 次のタスク

**show dockerbox status** および関連するコマンドで、設定を確認します。コマンド例と予想される出力については、検証のセクションを参照してください。

## サードパーティ製アプリケーション（TPA）コンテナ ホスティングの確認

Cisco Nexus 9000 スイッチでのサードパーティ アプリケーション（TPA）コンテナ ホスティングのステータスと動作を確認するには、次の手順に従います。

これらの検証手順により、NX OS CLIコマンドを使用して、TPA ホスティングとそのコンテナの正常性と動作をチェックできます。

**Before you begin**

TPA ホスティングがスイッチで構成され、有効になっていることを確認します。

### 手順

---

**ステップ 1** TPA ホスティングのステータスを表示します。

例：

```
show dockerbox status
```

このコマンドは、TPA（Docker）サービスの現在のステータスを表示します。

```
<example output placeholder>
```

**ステップ 2** 内部状態およびデバッグ情報を表示します。

例：

```
show system internal dockerbox
```

このコマンドは、TPA ホスティングの詳細な内部状態情報を提供します。

```
<example output placeholder>
```

**ステップ 3** イベント履歴を表示します。

例：

```
show system internal dockerbox event-history [error|debug|msgs]
```

TPA ホスティングに関連するエラーログ、デバッグメッセージ、またはメッセージ履歴を表示するには、このコマンドを使用します。

```
<example output placeholder>
```

**ステップ 4** コンテナ情報を表示します。

例：

```
show system internal dockerbox containers [brief|stats|logs]
```

このコマンドは、概要、リソース統計、ログを含む、実行中のコンテナに関する情報を表示します。

```
<example output placeholder>
```

**ステップ 5** TPA ホスティング ログを表示します。

例：

```
show system internal dockerbox log [errors]
```

このコマンドは、TPA ホスティングの Docker サービス ログを表示します。 **errors** オプションを使用すると、エラー メッセージのみを表示できます。

```
<example output placeholder>
```

第 12 章

# Cisco NX-OS での Ansible の使用


- 前提条件 （153 ページ）
- アンシブルについて （153 ページ）
- Cisco Ansible モジュール （154 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 （155 ページ）
- 前提条件 （156 ページ）
- Puppet エージェント NX-OS 環境 （156 ページ）
- ciscopuppet モジュール （156 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使用できる必要があります。
  - ブートフラッシュに最低 450MB の空きディスク容量。
- Puppet 4.0 以降の Puppet プライマリ サーバーが必要です。
- Puppet エージェント 4.0 以降が必要です。

# Puppet エージェント NX-OS 環境

Puppet Agent ソフトウェアは、ゲスト シェル（CentOS を実行する Linux コンテナ環境）のスイッチにインストールする必要があります。ゲスト シェルは、ホストから切り離された安全でオープンな実行環境を提供します。

Cisco NX-OS リリース 9.2（1）以降、Puppet Agent の Bash-shell（Cisco NX-OS の基盤となるネイティブ WindRiver Linux 環境）インストールはサポートされなくなりました。

次に、エージェント ソフトウェアのダウンロード、インストール、およびセットアップに関する情報を示します：

| | |
|---|---|
| Puppet Agent：Cisco Nexus スイッチでのインストールとセットアップ（手動セットアップ） | https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md |

# ciscopuppet モジュール

ciscopuppet モジュールは、Cisco が開発したオープン ソース ソフトウェア モジュールです。これは、Puppet マニフェストの抽象技術情報構成と、Cisco NX-OS オペレーティング システムおよびプラットフォームの特定の実装の詳細との間のインターフェイスとなります。このモジュールは Puppet プライマリにインストールされ、Cisco Nexus スイッチでの Puppet エージェントの操作に必要です。

ciscopuppet モジュールは、Puppet Forge で利用できます。

ここでは、ciscopuppet モジュール インストール手順についての追加情報を提供します：

| | |
|---|---|
| ciscopuppet モジュールの場所（Puppet Forge） | Puppet Forge |

| | |
|---|---|
| リソースの種類のカタログ | [Cisco Puppet 技術情報の参照先（Cisco Puppet Resource Reference）] |
| ciscopuppet モジュール：送信元コード リポジトリ | [Cisco Network Puppet モジュール（Cisco Network Puppet Module）] |
| ciscopuppet モジュール：セットアップと使用法 | Cisco Puppet モジュール::README.md |
| Puppet Labs: モジュールのインストール | https://puppet.com/docs/puppet/7/modules_installing.html |
| Puppet NX-OS マニフェストの例 | [Cisco Network Puppet モジュールの例（Cisco Network Puppet Module Examples）] |
| NX-OS デベロッパーのランディング ページ。 | 構成管理ツール |

第 14 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 （159 ページ）
- 前提条件 （160 ページ）
- Chef クライアント NX-OS 環境 （160 ページ）
- cisco-cookbook （160 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：Cisco Nexus プラットフォームでのインストールとセットアップ（手動セットアップ） | cisco-cookbook::README-install-agent.md |
| Chef クライアント：スイッチでのインストールとセットアップ（Chef プロビジョナを使用した自動インストール） | cisco-cookbook::README-chef-provisioning.md |

# cisco-cookbook

cisco-cookbook は、Chef レシピの抽象情報技術構成と、Cisco Nexus スイッチの特定の実装の詳細との間の、Cisco が開発したオープン ソース インターフェイスです。このクックブックは

Chef Server にインストールされ、Cisco Nexus スイッチでの Chef Client の適切な動作に必要です。

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

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

| トピック | リンク |
|---|---|
| cisco-cookbook の場所 | Chef Supermarket Cisco クックブック |
| リソースの種類のカタログ | リソースカタログ（テクノロジー別） |
| cisco-cookbook: ソース コード リポジトリ | Cisco Network Chef クックブック |
| cisco-cookbook: セットアップと使用法 | Chef クックブックの設定と使用方法 |
| Chef Supermarket | Chef Supermarket |
| Puppet NX-OS マニフェストの例 | Cisco Network Chef クックブックのレシピ |

第 15 章

# Nexus アプリケーション開発：Yocto


- Yocto について （163 ページ）
- Yocto のインストール （163 ページ）


## Yocto について

Cisco NX-OS リリース 10.1(1) ソフトウェアは、Yocto 2.6 に基づいています。Yocto 2.6 をダウンロードし、構築する新しいソフトウェアをダウンロードし、ソフトウェアを構築して、スイッチにソフトウェアを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 libsdl1.2-dev
 xterm
```

**ステップ 2** Yocto 2.6 をダウンロードします。

```
wget http://downloads.yoctoproject.org/releases/yocto/yocto-2.6/poky-thud-20.0.0.tar.bz2
tar xjfv poky-thud-20.0.0.tar.bz2
cd poky-thud-20.0.0
```

**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# dnf install /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
groups-repo                           | 1.1 kB     00:00 ...
localdb                               |  951 B     00:00 ...
patching                              |  951 B     00:00 ...
thirdparty                            |  951 B     00:00 ...
Setting up Install Process
Examining /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm: libyaml-0-2-0.1.6-r0.x86_64
Marking /bootflash/libyaml-0-2-0.1.6-r0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package libyaml-0-2.x86_64 0:0.1.6-r0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

==========================================================================================
 Package          Arch             Version         Repository                        Size
==========================================================================================
Installing:
 libyaml-0-2      x86_64           0.1.6-r0        /libyaml-0-2-0.1.6-r0.x86_64      119 k

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

Total size: 119 k
Installed size: 119 k
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : libyaml-0-2-0.1.6-r0.x86_64                                     1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link


Installed:
  libyaml-0-2-0.1.6-r0.x86_64
```

```
Complete!
Install operation 2450 completed successfully at Fri Jul 27 18:54:55 2018.

[####################] 100%
```

**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# dnf install /bootflash/ruby-2.2.2-r0.x86_64.rpm
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
groups-repo                            | 1.1 kB     00:00 ...
localdb                                |  951 B     00:00 ...
patching                               |  951 B     00:00 ...
thirdparty                             |  951 B     00:00 ...
thirdparty/primary                     | 1.8 kB     00:00 ...
thirdparty                                 2/2
Setting up Install Process
Examining /bootflash/ruby-2.2.2-r0.x86_64.rpm: ruby-2.2.2-r0.x86_64
Marking /bootflash/ruby-2.2.2-r0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ruby.x86_64 0:2.2.2-r0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=================================================================================================
 Package          Arch             Version         Repository                           Size
=================================================================================================
Installing:
 ruby             x86_64           2.2.2-r0        /ruby-2.2.2-r0.x86_64                32 M

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

Total size: 32 M
Installed size: 32 M
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : ruby-2.2.2-r0.x86_64                                    1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link


Installed:
  ruby.x86_64 0:2.2.2-r0

Complete!
Install operation 2451 completed successfully at Fri Jul 27 18:55:23 2018.

[####################] 100%
```

第 16 章

# Nexus アプリケーション開発：SDK


- Cisco SDK について （167 ページ）
- SDK のインストール （167 ページ）
- インストールと環境の初期化の手順 （168 ページ）
- SDK を使用したアプリケーションの構築 （169 ページ）
- RPM を使用したアプリケーションのパッケージ化 （170 ページ）
- RPM ビルド環境の作成 （171 ページ）
- 一般的な RPM ビルド手順の使用 （172 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 （173 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 （174 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 dnf install glibc.i686
  ```

- Ubuntu 14.04 の場合：

  ```
  bash$ sudo apt-get install gcc-multilib
  ```

# インストールと環境の初期化の手順

SDK ファイルは自己解凍アーカイブで、SDK を任意のディレクトリにインストールできます。SDK のインストール ディレクトリへのパスの入力が求められます。

```
bash$ ./wrlinux-8.0.0.25-glibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Wind River Linux SDK installer version 8.0-n9000
================================================
Enter target directory for SDK (default: /opt/windriver/wrlinux/8.0-n9000):
You are about to install the SDK to "/opt/windriver/wrlinux/8.0-n9000". Proceed[Y/n]? Y
Extracting
SDK.......................................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.

. environment-setup-corei7-64-nxos-linux
. environment-setup-corei7-32-nxosmllib32-linux

source environment-setup-corei7-64-nxos-linux
source environment-setup-corei7-32-nxosmllib32-linux
=============================
```

**source environment-setup-x86-wrsmllib32-linux** および **source environment-setup-x86_64-wrs-linux** コマンドを使用して、シェル環境に SDK 固有のパスを追加します。SDK で使用する予定のシェルごとに、SDK 固有のパスを追加します。SDK 固有のパスを追加することは、ビルド ツールとライブラリの正しいバージョンを使用するように SDK をセットアップする点で重要です。

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-wrsmllib32-linux
bash$ source environment-setup-x86_64-wrs-linux
```

# SDK を使用したアプリケーションの構築

一般的な Linux ビルドプロセスの多くは、このシナリオで機能します。状況に最適な方法を使用してください。

アプリケーションパッケージのソースコードは、さまざまな方法で取得できます。たとえば、tar ファイル形式で、またはパッケージが存在する git リポジトリからダウンロードして、ソースコードを取得できます。

次は最も一般的なケースの一例です。

**（オプション）アプリケーション パッケージが標準の configure/make/make install を使用してビルドされることを確認します。**

```
bash$ tar --xvzf example-app.tgz
bash$ mkdir example-lib-install
bash$ cd example-app/
bash$ ./configure --prefix=/path/to/example-app-install
bash$ make
bash$ make install
```

場合によっては、./configure スクリプトに追加のオプションを渡す必要があります。たとえば、必要なオプションのコンポーネントと依存関係を指定する場合などです。追加オプションを渡すかどうかは、構築するアプリケーションに完全に依存します。

**例：Ganglia とその依存関係の構築**

この例では、ganglia と、必要なサードパーティライブラリ（libexpat、libapr、および libconfuse）を作成します。

**libexpat**

```
bash$ wget 'http://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz'
bash$ mkdir expat-install
bash$ tar xvzf expat-2.1.0.tar.gz
bash$ cd expat-2.1.0
bash$ ./configure --prefix=/home/sdk-user/expat-install
bash$ make
bash$ make install
bash$ cd ..
```

### libapr

```
bash$ wget 'http://www.eu.apache.org/dist/apr/apr-1.5.2.tar.gz'
bash$ mkdir apr-install
bash$ tar xvzf apr-1.5.2.tar.gz
bash$ cd apr-1.5.2
bash$ ./configure --prefix=/home/sdk-user/apr-install
bash$ make
bash$ make install
bash$ cd ..
```

### libconfuse

（注） confuse には、./configure に追加の --enable-shared オプションが必要です。そうでない場合、必要な共有ライブラリの代わりに静的にリンクされたライブラリが構築されます。

```
bash$ wget 'http://savannah.nongnu.org/download/confuse/confuse-2.7.tar.gz'
bash$ mkdir confuse-install
bash$ tar xvzf confuse-2.7.tar.gz
bash$ cd confuse-2.7
bash$ ./configure --prefix=/home/sdk-user/confuse-install --enable-shared
bash$ make
bash$ make install
bash$ cd ..
```

### ganglia

（注） 必要なすべてのライブラリの場所が ./configure に渡されます。

```
bash$ wget
'http://downloads.sourceforge.net/project/ganglia/ganglia%20monitoring%20core/3.7.2/ganglia-3.7.2.tar.gz'
bash$ mkdir ganglia-install
bash$ tar xvzf ganglia-3.7.2.tar.gz
bash$ cd ganglia-3.7.2
bash$ ./configure --with-libexpat=/home/sdk-user/expat-install
--with-libapr=/home/sdk-user/apr-install/bin/apr-1-config
--with-libconfuse=/home/sdk-user/confuse-install --prefix=/home/sdk-user/ganglia-install
bash$ make
bash$ make install
bash$ cd ..
```

# RPM を使用したアプリケーションのパッケージ化

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

（注） **RPM および仕様ファイル**

RPMパッケージ形式は、特定のアプリケーションの完全なインストールに必要なすべてのファイル（バイナリ、ライブラリ、設定、ドキュメントなど）をパッケージ化するように設計されています。したがって、RPM ファイルを作成するプロセスは簡単ではありません。RPM ビルドプロセスを支援するために、ビルドプロセスに関するすべてを制御する .spec ファイルが使用されます。

（注） 多くのサードパーティ製アプリケーションは、tarball にパッケージ化されたソースコードの形式でインターネット上で入手できます。多くの場合、これらの tarball には RPM ビルドプロセスに役立つ .spec ファイルが含まれています。残念ながら、これらの .spec ファイルの多くは、ソースコード自体ほど頻繁には更新されません。さらに悪いことに、spec ファイルがまったくない場合もあります。このような場合、RPM を構築できるように、仕様ファイルを編集または最初から作成する必要があります。

# RPM ビルド環境の作成

SDK を使用して RPM をビルドする前に、RPM ビルド ディレクトリ構造を作成し、いくつかの RPM マクロを設定する必要があります。

**手順**

**ステップ 1** ディレクトリ構造を作成します：

```
bash$ mkdir rpmbuild
bash$ cd rpmbuild
bash$ mkdir BUILD RPMS SOURCES SPECS SRPMS
```

**ステップ 2** 上で作成したディレクトリ構造を指すように topdir マクロを設定します：

```
bash$ echo "_topdir ${PWD}" > ~/.rpmmacros
```

（注）
この手順は、現在のユーザーがすでに設定されている .rpmmacros ファイルを有していないことを前提としています。既存の .rpmmacros ファイルを変更するのが不便な場合は、すべての rpmbuild コマンドラインに次を追加できます。

```
--define "_topdir ${PWD}"
```

**ステップ 3** RPM DB を更新します。

```
bash$ rm /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm/__db.*
bash$ rpm --rebuilddb
```

（注）
SDK の rpm および rpmbuild ツールは、RPM データベースとして通常の /var/lib/rpm の代わりに /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm を使用するように変更されました。この変更により、SDK を使用していない場合にホストの RPM データベースと競合することが回避され、ルート アクセスの必要性がなくなります。SDK のインストール後、この手順に従って SDK RPM データベースを再構築する必要があります。

# 一般的な RPM ビルド手順の使用

一般的な RPM ビルド手順は次のとおりです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app
 tarball>
bash$ # determine location of spec file in tarball:
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

結果は RPMS/ に作成されるバイナリ RPM で、スイッチにコピーしてインストールできます。アプリケーションのインストール方法と構成には様々なバリエーションがあり得ます。これらの手順については、アプリケーションのドキュメントを参照してください。

この rpm ビルドとスイッチへのインストールは、アプリケーションをサポートするために必要なすべてのソフトウェアパッケージで必要です。SDK にまだ含まれていないソフトウェアの依存関係を満たすことが必要な場合は、ソースコードを取得して、依存関係のあるソフトウェアもビルドする必要があります。ビルド用のマシンでは、パッケージを手動でビルドして、依存関係を検証することができます。次に、最も一般的な手順の例を示します。

```
bash$ tar xkzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example-lib-install
bash$ make
bash$ make install
```

これらのコマンドは、ビルドファイル（バイナリ、ヘッダー、ライブラリなど）をインストールディレクトリに配置します。ここから、標準のコンパイラとリンカのフラグを使用して、依存関係を満たすための新しい場所を選択できます。ライブラリなどのランタイム コードがあれば、それらもスイッチにインストールする必要があるため、必要なランタイム コードを RPM にパッケージ化しなければなりません。

（注） Cisco devhub の Web サイトには、すでに RPM 形式にまとめられているサポート ライブラリが多数あります。

# オプションのプラグインを使用しない collectd RPM の構築例

ソース tarball をダウンロードし、仕様ファイルを抽出します。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この tarball には 4 つの spec ファイルがあります。Red Hat 仕様ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

この仕様ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM を設定します。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用します。仕様ファイルで編集された以下を編集します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

collectd には多数のオプションプラグインがあります。この仕様ファイルは、デフォルトで多くのプラグインを有効にします。多くのプラグインには外部依存関係があるため、これらのプラグインを無効にするオプションをコマンドラインに渡す必要があります。**rpmbuild** 1 つの長いコマンドラインを入力する代わりに、次のように Bash 配列でオプションを管理できます。

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>  rpmbuild_opts+=("--without")
>  rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

その後、次のように rpmbuild に渡され、ビルドおよび RPM パッケージ プロセス全体が開始されます。

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

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

これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
```

# オプションの Curl プラグインを使用した collectd の RPM のビルド例

collectd curl プラグインには、依存関係として libcurl があります。

RPM ビルドプロセス中にこのリンクの依存関係を満たすには、SDK で curl をダウンロードしてビルドする必要があります。

```
bash$ wget --no-check-certificate http://curl.haxx.se/download/curl-7.24.0.tar.gz
bash$ tar xkvf curl-7.24.0.tar.gz
bash$ cd curl-7.24.0
bash$ ./configure --without-ssl --prefix /path/to/curl-install
bash$ make
bash$ make install
bash$ cd ..
```

（注） curl バイナリとライブラリは、/path/to/curl-install にインストールされます。このディレクトリが存在しない場合は作成されるため、現在のユーザーには書き込み権限が必要です。次に、ソース tarball をダウンロードし、spec ファイルを抽出します。この手順は、プラグインがない場合の collectd の例とまったく同じです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この spec ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM をセットアップします。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用する必要があるため、spec ファイルで次のように編集します。

（注） この tarball には 4 つの spec ファイルがあります。Red Hat の spec ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

この点は、前の例と違っています。collectd rpmbuild プロセスは、libcurl の場所を認識する必要があります。collectd の spec ファイルを編集して、以下を追加します。

`SPECS/collectd.spec` で文字列 *%configure* を検索します。この行とそれに続く行は、rpmbuild が ./configure スクリプトに渡すオプションを定義します。

次のオプションを追加します：

```
--with-libcurl=/path/to/curl-install/bin/curl-config \
```

次に、rpmbuild コマンド オプションを含む Bash アレイが再度構築されます。次の違いに留意してください。

- *curl* をビルドされないプラグインのリストから削除
- *--with curl=force* の追加

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>  rpmbuild_opts+=("--without")
>  rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=("--with")
bash$ rpmbuild_opts+=("curl=force")bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

それからこれは次のように rpmbuild に渡され、ビルドおよび RPM パッケージ プロセス全体が開始されます：

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

RPMs ディレクトリ内の結果の RPM には、collectd-curl も含まれるようになりました。これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
bash$ rpm --noparentdirs -i /bootflash/collectd-curl-5.5.0-1.ia32e.rpm
```

第 17 章

# NX-SDK


- NX-SDK について （177 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 （179 ページ）
- デフォルト Docker イメージ （179 ページ）
- NX-SDK に関する注意事項と制限事項 （179 ページ）
- NX-SDK2.0 について （180 ページ）
- NX-SDK2.5 について （180 ページ）
- リモート アプリケーションについて （181 ページ）
- NX-SDK セキュリティ （181 ページ）
- NX SDK 2.0 のセキュリティ プロファイル （182 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 （179 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 （181 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 （179 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<br>クロス コンパイル用の Wind River Linux（WRL）ツール チェーン<br>多言語バインディング ツールキット<br>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<br>Go コンパイラ<br>RapidJSON<br>リモート API サポートのための gRPC |

詳細については、https://github.com/CiscoDevNet/NX-SDK#readmeを参照してください。

# NX-SDK に関する注意事項と制限事項

NX-SDK には、アプリケーションをローカル（オンボックス）またはリモート（オフボックス）で実行するための使用上の注意事項と制限事項があります。

注意事項と制限事項については、Cisco DevNet NX-SDK の「役立つメモ」を参照してください。

# NX-SDK2.0 について

NX-SDK バージョン 2.0 は、開発者が必要な場所でアプリケーションを実行できる実行環境の柔軟性を可能にします。このバージョンの NX-SDK では、アプリケーションは引き続きコンテナ内のスイッチをオフにして開発されますが、たとえばクラウドなどで、アプリケーションをスイッチ上またはスイッチ外のいずれかで実行できます。

NX-SDK 2.0 には次のような利点があります。

- スイッチをお客様の環境に簡単に統合できます。
- スイッチがデータセンター、パブリック クラウド、プライベート クラウドでシームレスに動作できるようにする拡張性。
- スイッチ レベルのリソースでの変更がアプリケーションの変更または書き換えを必要としないように、スイッチ リソースから顧客アプリを切り離します。
- アプリケーションがリンクするための使いやすい API を備えた単一のライブラリ。これにより、スイッチの相互作用が簡素化され、アプリケーションをより簡単に記述およびデバッグできる高水準言語で記述できます。
- リモート サービスの実行は、オンボックス アプリケーションよりも安全です。

詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

# NX-SDK2.5 について

Cisco NX-OS Release 9.3(3) 以降では、ストリーミング Syslog 機能が追加されています。

詳細については、『CiscoDevNet』を参照してください。https://github.com/CiscoDevNet/NX-SDK/blob/master/versions.md

**表 *5 : 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。<br>• 詳細については、nx_trace.h の watchSyslog および postSyslogCb API を参照してください。https://wwwin-github.cisco.com/sathsrin/nxsdk/blob/master/include/nx_trace.h |

# リモート アプリケーションについて

リモート アプリケーションは、Cisco Nexus スイッチではない別のスイッチに置くことができます。リモートまたはオフボックスのアプリケーションは、NX-SDK レイヤを介して呼び出し、スイッチと対話して情報の読み取り（取得）または情報の書き込み（設定）をします。

ローカルとリモートの両方の NX-SDK アプリケーションは同じ API を使用するため、NX-SDK アプリケーションをオンボックスまたはオフボックスで柔軟に展開できます。

リモートで実行するには、アプリケーションが特定の要件を満たしている必要があります。詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

### 2.0 より前の NX-SDK アプリケーションの後方互換性

NX-SDK 2.0 には、NX-SDK v1.75 アプリケーションの開発方法に応じて、条件付きの後方互換性があります。

- 通常、NX-SDK は、NX-SDK 2.0 より前に作成したアプリのリモート実行をサポートしており、アプリを完全に書き直す必要はありません。代わりに、API 呼び出しを変更するために変更せずに、同じアプリを再利用できます。新しい NX-SDK 2.0 モデルで古いアプリをサポートするには、API 呼び出しで IP およびポートのパラメータを提供する必要があります。これらのパラメータは NX-SDK 1.75 以前では使用できませんが、アプリが SDK サーバにエクスポートできる環境変数として IP アドレスとポート情報を追加できます。
- ただし、NX-SDK 2.0 より前のアプリの下位互換性がサポートされていない場合があります。古いアプリの一部の API は、リモートでの実行をサポートしていないか、実行できない可能性があります。この場合、API は例外をスローできます。元のアプリケーションに対する例外処理の完全性と堅牢性によっては、アプリケーションが予期しない動作をする可能性があり、最悪の場合、クラッシュする可能性があります。

詳細については、https://github.com/CiscoDevNet/NX-SDK/blob/master/readmes/NXSDK_in_NXOS.md を参照してください。

# NX-SDK セキュリティ

NX-OS 9.3（1）以降、NX-SDK 2.0 は次のセキュリティ機能をサポートしています。

- セッション セキュリティ。リモート アプリケーションは、トランスポート レイヤー サービス （TLS) を介してスイッチ上の NX SDK サーバに接続し、アプリケーションとスイッチの NX SDK サーバ間に暗号化されたセッションを提供できます。
- サーバ証明書のセキュリティ。Cisco NX-OS 9.3（1）を使用した新しいスイッチ展開の場合、NX-SDK サーバは 1 日限りの一時証明書を生成して、カスタム証明書をインストールするのに十分な時間を提供します。

たとえば、以前の NX-SDK バージョンから NX-SDK 2.0 にアップグレードする場合など、NX-SDK サーバにカスタム証明書がすでにインストールされている場合、既存の証明書はアップグレード後も保持され、使用されます。

- API 書き込み呼び出し制御。NX-SDK 2.0 では、セキュリティ プロファイルが導入されています。これにより、アプリケーションが NX-SDK サーバをどの程度制御できるかを制御するための事前定義されたポリシーを選択できます。 セキュリティ プロファイルに関する詳細情報を入手するには、NX SDK 2.0 のセキュリティ プロファイル （182 ページ）を参照します。

# NX SDK 2.0 のセキュリティ プロファイル

以前のリリースでは、SDK バージョン 1.75 の API は、イベントのデータの読み取りと取得のみが許可されていました。Cisco NX-OS リリース 9.3(1) 以降、NX-SDK 2.0 は書き込みコールを含むさまざまなタイプの操作をサポートします。

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

Cisco の NX-SDK は、次のセキュリティ プロファイルを提供します。

| プロファイル | 説明 | 値 |
|---|---|---|
| 拒否 | CLI の追加を除き、API 呼び出しがスイッチに書き込まれないようにします。 | これはデフォルト プロファイルです。 |
| スロットル | スイッチを変更する API を許可しますが、指定された数の呼び出しまでのみ許可します。このセキュリティプロファイルは、スロットリングを適用して API 呼び出しの数を制御します。<br>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 18 章

# Cisco NX-OS での Docker の使用


- Cisco NX-OS での Docker について （185 ページ）
- Docker の注意事項と制限事項 （186 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 （186 ページ）
- Docker デーモンの開始 （187 ページ）
- 自動的に起動するように Docker を構成する （187 ページ）
- Docker コンテナの開始: ホスト ネットワーク モデル （188 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 （189 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 （190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 （191 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 （192 ページ）
- Docker ストレージ バックエンドのサイズ変更 （193 ページ）
- Docker デーモンの停止 （195 ページ）
- Docker コンテナ セキュリティ （196 ページ）
- Docker のトラブルシューティング （198 ページ）


## Cisco NX-OS での Docker について

Dockerは、すべての依存関係とライブラリと共にパッケージ化された、コンテナー内で安全に分離されたアプリケーションを実行する方法を提供します。Dockerの詳細にを表示するためにhttps://docs.docker.com/を参照してください。

Cisco NX-OS リリース 9.2（1）以降、スイッチ上の Cisco NX-OS 内で Docker を使用するためのサポートが追加されました。

スイッチに含まれる Docker のバージョンは CE 18.09.0 です。Docker デーモンはデフォルトでは実行されていません。手動で起動するか、スイッチの起動時に自動的に再起動するように設定する必要があります。

このセクションでは、スイッチ環境の特定のコンテキストで Docker を有効にして使用する方法について説明します。一般的な Docker の使用方法と機能の詳細については、https://docs.docker.com/ にある Docker のドキュメントを参照してください。

# Docker の注意事項と制限事項

次に、スイッチ上の Cisco NX-OS で Docker を使用するためのガイドラインと制限事項を示します。

- Docker でサードパーティの DHCPD サーバーを実行している場合、SVI と一緒に使用すると、クライアントに到達するオファーで問題が発生する可能性があります。可能な回避策は、ブロードキャスト応答を使用することです。
- Docker 機能は、少なくとも 8 GB のシステム RAM を備えた Cisco Nexus 9000Cisco Nexus 3000 スイッチでサポートされています。
- Cisco NX-OS リリース 10.6(2)F 以降、 Cisco Nexus 9000 シリーズ スイッチは、既存の netstack パスに加えて、 Linux カーネル ネットワークスタック（kstack）を介した SSH 接続の受け入れをサポートしています。

# Cisco NX-OS 内で Docker コンテナを設定するための前提条件

スイッチの Cisco NX-OS で Docker を使用するための前提条件は次のとおりです：

- ホスト Bash シェルを有効にします。スイッチの Cisco NX-OS で Docker を使用するには、ホスト Bash シェルのルート ユーザーである必要があります：

```
switch# configure terminal
    Enter configuration commands, one per line. End with CNTL/Z.
    switch(config)# feature bash-shell
```

- スイッチが HTTP プロキシ サーバを使用するネットワーク内にある場合、`http_proxy` と `https_proxy` 環境変数を `/etc/sysconfig/docker` に構成する必要があります。
- スイッチのクロックが正しく設定されていることを確認してください。そうしないと、次のエラー メッセージが表示される場合があります：

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

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

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

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

root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch#
```

# Docker デーモンの開始

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

Docker デーモンを開始するには：

**手順**

**ステップ 1** Bash を読み込み、スーパーユーザーになります。

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

**ステップ 2** Docker デーモンを起動します。

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

**ステップ 3** ステータスをチェックします。

```
root@switch# service docker status
dockerd (pid 3597) is running...
root@switch#
```

（注）
Docker デーモンを起動したら、ブートフラッシュ の `dockerpart` ファイルを削除したり、改ざんしたりしないでください。これは、docker の機能にとって重要であるからです。

```
switch# dir bootflash:dockerpart
2000000000 Mar 14 12:50:14 2018 dockerpart
```

# 自動的に起動するように Docker を構成する

スイッチの起動時に常に自動的に起動するように Docker デーモンを構成できます。

手順

**ステップ 1** Bash をロードしてスーパーユーザーになります。

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

**ステップ 2** chkconfig ユーティリティを使用して、Docker サービスを永続化します。

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

**ステップ 3** chkconfig ユーティリティを使用して、Docker サービスの設定を確認します。

```
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch#
```

**ステップ 4** Docker が自動的に起動しないように構成を削除するには：

```
root@switch# chkconfig --del docker
root@switch# chkconfig --list | grep docker
root@switch#
```

# Docker コンテナの開始: ホスト ネットワーク モデル

Docker コンテナがデータ ポートと管理を含むすべてのホスト ネットワーク インターフェイスにアクセスできるようにする場合は、--network ホスト オプションを使用して Docker コンテナを起動します。コンテナ内のユーザーは、ip netns exec <net_namespace> <cmd> を使用して、/var/run/netns（Cisco NX-OS で設定されたさまざまな VRF に対応）でさまざまなネットワーク名前空間を切り替えることができます。

手順

**ステップ 1** Bash をロードしてスーパーユーザーになります。

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

**ステップ 2** Docker コンテナを開始します。

以下は、スイッチで Alpine Docker コンテナを起動し、すべてのネットワーク インターフェイスを表示する例です。コンテナは、デフォルトで管理ネットワークの名前空間で起動されます。

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm --network
 host --cap-add SYS_ADMIN -it alpine
```

```
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
management
default
/ #
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
3: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default
link/gre 0.0.0.0 brd 0.0.0.0
...
/ #
/ # ip netns exec default ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/16 scope host lo
valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
link/ether 42:0d:9b:3c:d4:62 brd ff:ff:ff:ff:ff:ff
3: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
...
```

# Docker コンテナの開始: ブリッジ型ネットワーク モデル

Dockerコンテナに外部ネットワーク接続（通常は管理インターフェースを介して）のみを許可し、特定のデータ ポートまたは他のスイッチ インターフェースへの可視性を必ずしも気にしない場合は、デフォルトの Docker ブリッジ ネットワーク モデルで Docker コンテナを開始できます。 これは、ネットワーク名前空間の分離も提供するため、前のセクションで説明したホスト ネットワーキング モデルよりも安全です。

## 手順

**ステップ 1** Bash をロードしてスーパーユーザーになります。

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

**ステップ 2** Docker コンテナを開始します。

以下は、スイッチで Alpine Docker コンテナを開始し、`iproute2` パッケージをインストールする例です。

```
root@switch# docker run -it --rm alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
/ #
```

**ステップ 3** ユーザー名前空間の分離を設定するかどうかを決定します。

ブリッジネットワーク モデルを使用するコンテナの場合、ユーザー名前空間の分離を設定して、セキュリティをさらに向上させることもできます。詳細については、「ユーザー[名前空間（namespace）]の分離による Docker コンテナの保護 （196 ページ）」を参照してください。

標準の Docker ポート オプションを使用して、sshd などのコンテナー内からサービスを公開できます。例：

```
root@switch# docker run -d -p 18877:22 --name sshd_container sshd_ubuntu
```

これにより、コンテナ内のポート 22 がスイッチのポート 18877 にマップされます。次の例に示すように、ポート 18877 を介してサービスに外部からアクセスできるようになりました。

```
root@ubuntu-vm# ssh root@ip_address -p 18887
```

# Docker コンテナでのブートフラッシュおよび揮発性パーティションのマウント

Docker コンテナの run コマンドで `-v /bootflash:/bootflash` および `-v /volatile:/volatile` オプションを渡すことで、ブートフラッシュおよび揮発性パーティションを Docker コンテナに表示できます。これは、新しい NX-OS システム イメージをブートフラッシュにコピーするなど、コンテナ内のアプリケーションがホストと共有するファイルにアクセスする必要がある場合に役立ちます。

（注） この **-v** コマンド オプションを使用すると、任意のディレクトリをコンテナにマウントでき、NX-OS システムの動作に影響を与える可能性のある情報漏えいやその他のアクセスが発生する可能性があります。これを、NX-OS CLI を使用してすでにアクセス可能な `/bootflash` や `/volatile` などのリソースに制限します。

手順

**ステップ 1** Bash をロードしてスーパーユーザーになります。

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

**ステップ 2** Docker コンテナの実行コマンドに `-v /bootflash:/bootflash` および `-v /volatile:/volatile` オプションを渡します。

```
root@switch# docker run -v /bootflash:/bootflash -v /volatile:/volatile -it --rm alpine
/# ls /
bin        etc        media      root       srv        usr
bootflash  home       mnt        run        sys        var
dev        lib        proc       sbin       tmp        volatile
/ #
```

# 拡張 ISSU スイッチオーバーでの Docker デーモンの永続性の有効化

Docker デーモンと実行中のコンテナの両方を拡張 ISSU スイッチオーバーで持続させることができます。これが可能なのは、バックエンドの Docker ストレージが存在するブートフラッシュが同じであり、アクティブ スーパーバイザとスタンバイ スーパーバイザの両方で共有されるためです。

Docker コンテナは、切り替え中に中断（再起動）されるため、継続的に実行されません。

手順

**ステップ 1** Bash をロードしてスーパーユーザーになります。

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

**ステップ 2** スイッチオーバーを開始する前に、`chkconfig` ユーティリティを使用して Docker サービスを永続化します。

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

**ステップ 3** スイッチオーバー後にコンテナが自動的に再起動されるように、`--restart without-stopped` オプションを使用してコンテナを起動します。

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

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

Docker コンテナは、切り替え中に中断（再起動）されるため、継続的に実行されません。

# 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/userns-remap/」を参照してください。

### 手順

**ステップ 1** システムに dockremap グループがすでに存在するかどうかを確認します。

dockremap ユーザーは、デフォルトでシステムにすでに設定されている必要があります。dockremap グループがまだ存在しない場合は、次の手順に従って作成します。

a) 次のコマンドを入力して dockremap グループを作成します。

```
root@switch# groupadd dockremap -r
```

b) dockremap ユーザーを作成します（まだ存在していない場合）。

```
root@switch# useradd dockremap -r -g dockremap
```

c) dockremap グループと dockremap ユーザーが正常に作成されたことを確認します。

```
root@switch# id dockremap
uid=999(dockremap) gid=498(dockremap) groups=498(dockremap)
```

```
root@switch#
```

**ステップ 2** 再マップされた必要な ID と範囲を /etc/subuid と /etc/subgid に追加します。

例：

```
root@switch# echo "dockremap:123000:65536" >> /etc/subuid
root@switch# echo "dockremap:123000:65536" >> /etc/subgid
```

**ステップ 3** テキスト エディタを使用して、--userns-remap=default オプションを /etc/sysconfig/docker ファイルの other_args フィールドに追加します。

例：

```
other_args="-debug=true --userns-remap=default"
```

**ステップ 4** [サービス ドッカー [re]start（service docker [re]start）] を使用して、Docker デーモンを再起動するか、まだ実行されていない場合は起動します。

例：

```
root@switch# service docker [re]start
```

ユーザー名前空間の分離によるコンテナの構成と使用の詳細については、https://docs.docker.com/engine/security/userns-remap/で Docker のドキュメントを参照してください。

## cgroup パーティションの移動

サードパーティ サービスの cgroup パーティションは ext_ser で、CPU 使用率をコアあたり 25% に制限します。この ext_ser パーティションで Docker コンテナを実行することをお勧めします。

--cgroup-parent=/ext_ser/ オプションを指定せずに Docker コンテナを実行すると、最大 100% のホスト CPU アクセスが可能になり、Cisco NX-OS の通常の動作を妨げる可能性があります。

### 手順

**ステップ 1** Bash をロードしてスーパーユーザーになります。

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

**ステップ 2** ext_ser パーティションで Docker コンテナを実行します。

例：

```
root@switch# docker run --name=alpinerun -v /var/run/netns:/var/run/netns:ro,rslave --rm --network
 host --cgroup-parent=/ext_ser/ --cap-add SYS_ADMIN -it alpine
```

```
/ #
```

# Docker のトラブルシューティング

これらのトピックでは、Docker コンテナで発生する可能性のある問題について説明し、考えられる解決策を提供します。

## Docker の起動が機能不全になる

**[問題：（Problem:）]** Docker の起動に失敗し、次のようなエラー メッセージが表示されます：

```
switch# run bash
bash-4.3$ service docker start
Free bootflash: 39099 MB, total bootflash: 51771 MB
Carving docker bootflash storage: 2000 MB
2000+0 records in
2000+0 records out
2000000000 bytes (2.0 GB) copied, 22.3039 s, 89.7 MB/s
losetup: /dev/loop18: failed to set up loop device: Permission denied
mke2fs 1.42.9 (28-Dec-2013)
mkfs.ext4: Device size reported to be zero.  Invalid partition specified, or
        partition table wasn't reread after running fdisk, due to
        a modified partition being busy and in use.  You may need to reboot
        to re-read your partition table.

Failed to create docker volume
```

**[考えられる原因：（Possible Cause:）]** root ユーザーではなく、管理ユーザーとして Bash を実行している可能性があります。

**[解決策：（Solution:）]** root ユーザーではなく、管理ユーザーとして Bash を実行しているかどうかを確認します。

```
bash-4.3$ whoami
admin
```

Bash を終了し、ルート ユーザーとして Bash を実行します：

```
bash-4.3$ exit
switch# run bash sudo su -
```

## ストレージが不足しているため、Docker が起動に失敗する

**問題**：ブートフラッシュ ストレージが不足しているため、Docker の起動に失敗し、次のようなエラー メッセージが表示されます。

```
root@switch# service docker start
Free bootflash: 790 MB, total bootflash: 3471 MB
```

```
Need at least 2000 MB free bootflash space for docker storage
```

**考えられる原因**：十分な空きブートフラッシュ ストレージがない可能性があります。

**解決策**：スペースを解放するか、必要に応じて `/etc/sysconfig/docker` の変数 *_dockerstrg* 値を調整してから、Docker デーモンを再起動します。

```
root@switch# cat /etc/sysconfig/docker
# Replace the below with your own docker storage backend boundary value (in MB)
# if desired.
boundary_dockerstrg=5000

# Replace the below with your own docker storage backend values (in MB) if
# desired. The smaller value applies to platforms with less than
# $boundary_dockerstrg total bootflash space, the larger value for more than
# $boundary_dockerstrg of total bootflash space.
small_dockerstrg=300
large_dockerstrg=2000
```

# Docker Hub からのイメージのプルの失敗（509 証明書失効 エラー メッセージ）

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

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

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

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

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

必要に応じて、時計をリセットします：

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

例：

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

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

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

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

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

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

```
root@switch# cat /etc/sysconfig/docker | grep proxy
root@switch# service docker [re]start
```

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

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

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

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

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

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

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

```
root@switch# chkconfig --add docker
root@switch#
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

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

# Dockerストレージ バックエンドのサイズ変更が失敗する

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

**考えられる原因**：ゲスト シェルが無効になっていない可能性があります。

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

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

ゲスト シェルが無効になっている場合、コマンドは出力を表示しません。

必要に応じて、次のコマンドを入力してゲスト シェルを無効にします。

```
switch# guestshell disable
```

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

# Dockerコンテナがポートで着信トラフィックを受信しない

**問題**：Docker コンテナがポートで着信トラフィックを受信しません。

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

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

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

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

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

**解決方法**：

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

# 一般的なトラブルシューティングのヒント

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

**解決方法**：

- `/var/log/docker` で dockerd デバッグ出力を探して、何が問題なのかの手掛かりを見つけてください。
- スイッチに 8 GB 以上の RAM があることを確認します。Docker 機能は、RAM が 8 GB 未満のスイッチではサポートされていません。

# 第 III 部

# アプリケーション ホスティング


- アプリケーション ホスティング （205 ページ）
- ThousandEyes Enterprise Agent （225 ページ）

第 19 章

# アプリケーション ホスティング

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

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

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


- アプリケーション ホスティングの注意事項と制限事項 （205 ページ）
- アプリケーション ホスティングに関する情報 （206 ページ）
- アプリケーション ホスティングの設定方法 （207 ページ）
- アプリケーション データのコピー （218 ページ）
- アプリケーション データの削除 （218 ページ）
- アプリケーション ホスティング設定の確認 （218 ページ）
- アプリケーション ホスティングの設定例 （221 ページ）
- その他の参考資料 （223 ページ）
- アプリケーション ホスティングに関する機能情報 （223 ページ）


## アプリケーション ホスティングの注意事項と制限事項

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

- アプリケーション ホスティング機能は、Cisco NX-OS リリース 10.3(1)F の IPv6 構成をサポートしていません。
- アプリケーションホスティング機能は、Cisco Nexus 9300 シリーズ FX、FX2、FX3、GX、および GX2 プラットフォームと、FX および GX ラインカードを搭載した Cisco Nexus 9500 シリーズモジュラスイッチでサポートされています。
- コンテナごとに 1 つのインターフェイスのみがサポートされます。

- アプリケーション ホスティングは、構成置換機能をサポートしていません。
- Cisco NX-OS リリース 10.3(3)F 以降、Cisco Nexus 9808 プラットフォーム スイッチでアプリケーション ホスティング機能がサポートされています。
- Cisco NX-OS リリース 10.3(3)F 以降、-R および -R2 ライン カードを搭載した Cisco Nexus 9504 および 9508 プラットフォーム スイッチでアプリケーション ホスティング機能はサポートされています。この機能は、Cisco N3K-C36180YC-R、N3K-C3636C-R、および N3K-C36480LD-R2 スイッチでもサポートされています。
- Cisco Nexus 9300-FX2、9300-GX、および 9300-GX2 プラットフォーム スイッチで、アプリケーション ホスティングをサポートしていないリリースから Cisco NX-OS リリース 10.3（3）F への無停止アップグレードを実行する場合、アップグレード後、アプリケーション ホスティング機能を有効にするには、TCAM エントリを有効にするためにシステムをリロードする必要があります。Cisco Nexus 9300-FX スイッチでは、SPAN が有効になっている場合、システムのリロードは必要ありません。
- NX-OS からの ping は netstack プロセスを使用するため、NX-OS からアプリケーションコンテナに ping を行って接続をテストすることはできません。双方向の到達可能性を確認する必要がある場合は、 bash の ping を使用します：**conf t** ; **feature bash** ; **run bash** ; **ping <app-ip-address>**。
- **system vrf-member-change retain-l3-config** コマンドを構成した後、インターフェイスの vrf メンバーシップを、アプリケーションホスティングブリッジと同じ VRF に分類されるように変更し、HSRP VIP をアプリケーションホスティングブリッジ IP のサブネットと一致させると、VIP は **show running-config** または DME から削除されず、障害は発生しません。代わりに、HSRP syslog（シビラティ 3）がスローされ、ユーザーに構成ミスを通知します。

# アプリケーション ホスティングに関する情報

ここでは、アプリケーション ホスティングについて説明します。

## アプリケーション ホスティングの必要性

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

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

# アプリケーション ホスティングの概要

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

アプリケーション ホスティングは、次のサービスを提供します。

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

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

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

# アプリケーション ホスティングの設定方法

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

## アプリケーション ホスティング機能の有効化

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

手順の概要

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

手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル コンフィギュレーション モードを開始します。 |
| ステップ 2 | **feature app-hosting**<br>例：<br>`switch(config)# feature app-hosting` | シスコ アプリケーション ホスティング機能を有効にします。 |
| ステップ 3 | **end**<br>例：<br>`switch(config)# end` | グローバル コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。 |

# アプリケーション ホスティング ブリッジ接続の設定

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

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

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

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

手順の概要

1. **configure terminal**

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

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル コンフィギュレーション モードを開始します。 |
| ステップ 2 | **app-hosting bridge** *bridge-index*<br>例：<br>`switch(config)# app-hosting bridge 1` | アプリケーションホスティングブリッジを設定し、アプリケーション ホスティング ブリッジ構成モードを開始します。<br>• <1-8> ブリッジ インデックス |
| ステップ 3 | **ip address** *ip-address/mask*<br>例：<br>`switch(config-app-hosting-bridge)# ip address 172.25.44.1/30` | アプリケーション コンテナへのゲートウェイとして機能するアプリケーション ブリッジ IPv4 アドレスを設定します。<br>（注）<br>Cisco NX-OS リリース 10.3(2)F 以降では、IP がインターフェイスまたは仮想 IP のいずれかによって使用されている場合、サブネットは拒否されます。 |
| ステップ 4 | **vrf member** *name*<br>例：<br>`switch(config-app-hosting-bridge)# vrf member overlay-VRF` | VRF コンテキストを設定します。構成していなければ、VRF のデフォルトに属しています。<br>（注）<br>管理 VRF の構成はサポートされていません。管理 VRF を構成すると、ブリッジ構成が失敗します。 |
| ステップ 5 | **exit**<br>例：<br>`switch(config-app-hosting-bridge)# exit` | アプリケーションブリッジ構成モードを終了して、グローバル構成モードに戻ります。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 6 | **app-hosting appid** *name*<br>例：<br>`switch(config)# app-hosting appid te_app` | アプリケーションを構成し、アプリケーション ホスティング構成モードを開始します。 |
| ステップ 7 | **app-vnic gateway bridge** *bridge-index* **guest-interface** *guest-interface-number*<br>例：<br>`switch(config-app-hosting)# app-vnic gateway bridge 1 guest-interface 0` | アプリケーションのゲスト VNIC インターフェイスを設定し、アプリケーション ホスティング vnic インターフェイス モードを開始します。<br>（注）<br>Cisco NX-OS リリース 10.3(3)F 以降では、複数の VNIC の構成はサポートされていません。 |
| ステップ 8 | **guest-ipaddress** *ip-address/mask*<br>例：<br>`switch(config-app-hosting-app-vnic)# guest-ipaddress 172.25.44.2/30` | ブリッジ *1* サブネットから使用可能な IPv4 アドレスの 1 つを設定します。 |
| ステップ 9 | **exit**<br>例：<br>`switch(config-app-hosting-app-vnic)# exit` | アプリケーション vnic インターフェイス構成モードを終了して、アプリケーション ホスティング構成モードに戻ります。 |
| ステップ 10 | **app-default-gateway** *ip-address* **guest-interface** *guest-interface*<br>例：<br>`switch(config-app-hosting-appid)# app-default-gateway 172.25.44.1 guest-interface 0` | ブリッジ *1* サブネットから使用可能な IPv4 アドレスを構成します。<br>ステップ 3 で構成したゲートウェイ アドレスを構成します。 |
| ステップ 11 | **end**<br>例：<br>`switch(config-app-hosting)# end` | アプリケーション ホスティング構成モードを終了し、特権 EXEC モードに戻ります。 |

# アプリケーションのライフサイクル

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

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

## 手順の概要

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

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **enable**<br>例：<br>`switch# enable` | 特権 EXEC モードを有効にします。<br>• プロンプトが表示されたら、パスワードを入力します。 |
| ステップ 2 | **app-hosting install appid** *application-name* **package** *package-path*<br>例：<br>`switch# app-hosting install appid te_app package bootflash:my_te_app.tar` | 指定した場所からアプリケーションをインストールします。<br>• ローカルの保管場所（つまり、ブートフラッシュ）からアプリケーションをインストールすることができます。 |
| ステップ 3 | **app-hosting activate appid** *application-name*<br>例：<br>`switch# app-hosting activate appid te_app` | アプリケーションをアクティブ化します。<br>• このコマンドは、すべてのアプリケーション リソース要求を検証します。すべてのリソースが使用可能な場合、コマンドはアプリケーションをアクティブにします。それ以外の場合、アクティブ化は失敗します。 |
| ステップ 4 | **app-hosting start appid** *application-name*<br>例：<br>`switch# app-hosting start appid te_app` | アプリケーションを起動します。<br>• アプリケーションの起動スクリプトをアクティブにします。 |
| ステップ 5 | **app-hosting stop appid** *application-name*<br>例：<br>`switch# app-hosting stop appid te_app` | （任意）アプリケーションを停止します。 |
| ステップ 6 | **app-hosting deactivate appid***application-name*<br>例： | （任意）アプリケーションに割り当てられているすべてのリソースを無効にします。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `switch# app-hosting deactivate appid te_app` | |
| ステップ 7 | **app-hosting uninstall appid** *application-name*<br>例：<br>`switch# app-hosting uninstall appid te_app` | （任意）アプリケーションをアンインストールします。<br>• 保存されているすべてのパッケージとイメージをアンインストールします。また、アプリケーションに加えられたすべての変更とアップグレードを削除します。 |

# アプリケーションのアップグレード

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

## 手順の概要

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

## 手順の詳細

手順

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

# Docker ランタイムオプションの設定

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

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

## 手順の概要

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

## 手順の詳細

手順

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

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 6 | **end**<br>例：<br>`switch(config-app-hosting-docker)# end` | アプリケーション ホスティング Docker コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。 |

# 管理インターフェイスでのアプリケーション ホスティングの構成

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

## 手順の概要

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

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **enable**<br>例：<br>`switch# enable` | 特権 EXEC モードを有効にします。<br>• プロンプトが表示されたら、パスワードを入力します。 |
| ステップ 2 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル構成モードを開始します。 |
| ステップ 3 | **app-hosting appid** *name*<br>例：<br>`switch(config)# app-hosting appid te_app` | アプリケーションを設定し、アプリケーション ホスティング コンフィギュレーション モードを開始します。 |
| ステップ 4 | **app-vnic management guest-interface** *network-interface*<br>例：<br>`switch(config-app-hosting)# app-vnic management guest-interface 0` | ゲストインターフェイスを管理ポートに接続し、アプリケーションホスティング管理ゲートウェイ コンフィギュレーション モードを開始します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | • **management** キーワードは、プライベート IPNAT モードでコンテナに接続する Cisco NX-OS インターフェイス *mgmt0* を指定します。<br>• **guest-interface***network-interface* のキーワード引数ペアは、Cisco NX-OS 管理インターフェイス *mgmt0* に接続されているコンテナの内部イーサネットインターフェイス番号を指定します。この例では、コンテナのイーサネット 0 インターフェイスに対して *guest-interface 0* を使用しています。 |
| ステップ 5 | **end**<br>例：<br>`switch(config-app-hosting-mgmt-gateway)# end` | アプリケーションホスティング管理ゲートウェイ コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。 |

# アプリケーションのリソース設定の上書き

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

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

## 手順の概要

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

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **enable**<br>例：<br>`switch# enable` | 特権 EXEC モードを有効にします。<br>• プロンプトが表示されたら、パスワードを入力します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 2 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル構成モードを開始します。 |
| ステップ 3 | **app-hosting appid** *name*<br>例：<br>`switch(config)# app-hosting appid te_app` | アプリケーション ホスティングをイネーブルにし、アプリケーション ホスティング コンフィギュレーション モードを開始します。 |
| ステップ 4 | **app-resource profile** *name*<br>例：<br>`switch(config-app-hosting)# app-resource profile custom` | カスタム アプリケーション リソース プロファイルを設定し、カスタム アプリケーション リソース プロファイル コンフィギュレーション モードを開始します。<br>• カスタムプロファイル名のみがサポートされています。 |
| ステップ 5 | **cpu** *unit*<br>例：<br>`switch(config-app-resource-profile-custom)# cpu 7400` | アプリケーションのデフォルトのCPU割り当てを変更します。<br>• リソース値はアプリケーション固有のため、これらの値を変更した場合、アプリケーションが変更後も確実に稼働できることを確認する必要があります。 |
| ステップ 6 | **memory** *memory*<br>例：<br>`switch(config-app-resource-profile-custom)# memory 2048` | デフォルトのメモリ割り当てを変更します。 |
| ステップ 7 | **end**<br>例：<br>`switch(config-app-resource-profile-custom)# end` | カスタム アプリケーション リソース プロファイル構成モードを終了し、特権 EXEC モードに戻ります。 |

# 高度なアプリケーション ホスティング機能

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

アプリケーション ホスティング機能が設定されている場合、ブートフラッシュから 2 GB のファイル スペースがアプリケーション ストレージ スペースとして予約されます。特定のアプ

リケーションにさらに多くの領域が必要な場合は、パーティションサイズを増やすことができます。または、**app-hosting bootflash backend storage limit size** グローバル構成コマンドを使用して、アプリケーションのスペース要件に基づき、スペースを減らすこともできます。アプリケーションは再起動します。

## 手順の概要

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

## 手順の詳細

手順

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

# アプリケーション データのコピー

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

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

値は次のとおりです。

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

# アプリケーション データの削除

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

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

値は次のとおりです。

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

# アプリケーション ホスティング設定の確認

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

## 手順の概要

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

## 手順の詳細

手順

**ステップ 1** **show app-hosting infra**

アプリケーションホスティング インフラの概要を表示します。

（注）
さらに操作を実行する前に、CAF を動作状態に移行します。

例：

```
switch(config)# show app-hosting infra
App signature verification: disabled
Docker partition size: 0 MB
Inband packet rate limit: 0 PPS
Services
----------------------------------
CAF 1.16.0.0 : Running
HA : Running
App Manager : Running
Libvirtd 4.7.0 : Running
Dockerd 18.09.0-ce : Running
Linux kernel 5.10.126 : Running
```

**ステップ 2 show app-hosting list**

動作しているアプリのリストを表示します。

例：

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

**ステップ 3 show app-hosting bridge**

アプリケーションホスティング ブリッジのリストを表示します。

例：

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

**ステップ 4 show app-hosting detail**

プリケーションホスティングについての詳細情報を表示します。

例：

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

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

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

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

Network interfaces
---------------------------------------
eth0:
MAC address : 5254.9999.0000
IPv4 address : 192.168.10.130
IPv6 address : fe80::5054:99ff:fe99:0/64
Network name : iox-nat_docker0
Tx Packets : 9
Tx Bytes : 726
Tx Errors : 0
Rx Packets : 0
Rx Bytes : 0
Rx Errors : 0

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

**ステップ 5 show app-hosting resource**

アプリケーションホスティングのリソースに関する情報を表示します。

例：

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

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

アプリケーションの使用率を表示します。

例：

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

**ステップ 7 show-tech app-hosting**

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

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

例：

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

# アプリケーション ホスティングの設定例

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

## 例：AppHosting 機能の有効化

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

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

## 例：アプリケーション ホスティング ブリッジ接続の構成

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

```
switch(config)# configure terminal
switch(config)# app-hosting bridge 1
switch(config-app-hosting-bridge)# ip address 172.25.44.1/30
switch(config-app-hosting-bridge)# vrf member overlay-VRF
```

```
switch(config-app-hosting-bridge)# exit
switch(config)# app-hosting appid te_app
switch(config- app-hosting)# app-vnic bridge 1 guest-interface 0
switch(config-app-hosting-app-vnic)# guest-ipaddress 172.25.44.2/30
switch(config-app-hosting-app-vnic)# exit
switch(config-app-hosting-appid)# app-default-gateway 172.25.44.1 guest-interface 0
switch(config-app-hosting)# end
```

# 例：Docker ランタイムオプションの設定

この例では、Docker ランタイムオプションを設定する方法を示します。

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

# 例：管理インターフェイスでのアプリケーションホスティングの構成

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

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

# 例：アプリケーションのリソース設定の上書き

この例では、アプリケーションのリソース設定を上書きする方法を示します。

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

# その他の参考資料

### 関連資料

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

### テクニカル サポート

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

# アプリケーション ホスティングに関する機能情報

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

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

表 *6:* アプリケーション ホスティングに関する機能情報

| 機能名 | リリース | 機能情報 |
|---|---|---|
| Cisco Nexus 9808 プラットフォーム スイッチでの Cisco アプリケーション ホスティング フレームワーク（CAF）のサポート | Cisco NX-OS リリース 10.3(3)F | アプリケーション ホスティング機能が、Cisco Nexus 9808 プラットフォーム スイッチでサポートされるようになりました。 |
| -R および -R2 ライン カードを搭載した Cisco Nexus 9504 および 9508 プラットフォーム スイッチでの Cisco アプリケーション ホスティング フレームワーク（CAF）のサポート | Cisco NX-OS リリース 10.3(3)F | アプリケーション ホスティング機能が、-R および -R2 ライン カードを搭載した Cisco Nexus 9504 および 9508 プラットフォーム スイッチでサポートされるようになりました。この機能は、Cisco N3K-C36180YC-R、N3K-C3636C-R、および N3K-C36480LD-R2 スイッチでもサポートされています。 |
| Cisco アプリケーション ホスティング フレームワーク（CAF） | Cisco NX-OS Release 10.3(1)F | ホスト対象のアプリケーションは Software as a Service（SaaS）ソリューションであり、ユーザーはこのソリューションの実行と運用を完全にクラウドから行うことができます。このモジュールでは、Cisco アプリケーション ホスティング機能とその有効化の方法について説明します。<br><br>アプリケーション ホスティング機能は、Cisco Nexus 9300 シリーズ FX、FX2、GX、および GX2 プラットフォームと、FX および GX ライン カードを搭載した Cisco Nexus 9500 シリーズ モジュラ スイッチでサポートされています。 |

第 20 章

# ThousandEyes Enterprise Agent


- ThousandEyes Enterprise Agent の概要 （225 ページ）
- ThousandEyes Enterprise Agent の前提条件 （226 ページ）
- ThousandEyes Enterprise Agent に必要なリソース （226 ページ）
- ThousandEyes Enterprise Agent のインストール （226 ページ）
- ThousandEyes Enterprise Agent の構成例 （232 ページ）


## ThousandEyes Enterprise Agent の概要

ThousandEyes Enterprise Agent は、エンタープライズ ネットワーク監視ツールであり、ビジネスに影響を与えるネットワークとサービス全体のエンドツーエンドのビューを提供します。内部、外部、キャリア、およびインターネット ネットワーク全体のネットワーク トラフィック パスをリアルタイムでモニタして、ネットワーク パフォーマンスデータを提供します。Enterprise Agent は、WAN やインターネットの接続状態を詳細に把握するために、ブランチ サイトやデータセンターにインストールするのが最も一般的です。

ThousandEyes Enterprise Agent は次の機能を提供します。

- ネットワークとアプリケーションのパフォーマンスベンチマーク。
- 詳細なホップバイホップメトリック。
- ブランチまたはキャンパスからデータセンターまたはクラウドへのエンドツーエンドのパスの可視化。
- 機能停止の検出と解決。
- ユーザーエクスペリエンス分析。
- トラフィックフローパターンの可視化。

# ThousandEyes Enterprise Agent の前提条件

- ThousandEyes Enterprise Agent CAF イメージは、Cisco ルータおよびスイッチのアプリケーション ホスティング機能で使用するために特別に作成された tar ファイルです。
- Enterprise Agent をインストールするには、インターネット接続またはプロキシサーバーが必要です。詳細については、https://docs.thousandeyes.com/product-documentation/global-vantage-points/enterprise-agents/installing/cisco-devices/nexus-switches にある ThousandEyes のドキュメントを参照してください。
- Enterprise Agent アプリケーションは、ユーザーのライセンス権限が検証された後にのみ使用できます。

# ThousandEyes Enterprise Agent に必要なリソース

次の表に、ThousandEyes Enterprise Agent のインストールに必要なリソースを示します。

**表 7 : *ThousandEyes Enterprise Agent* に必要なリソース**

| アプリケーションメディア | 最大リソース | 最小のサポート対象リリース |
|---|---|---|
| ブートフラッシュ | • CPU：0.5<br>• メモリ：500MB RAM<br>• ストレージ：アプリケーションによる永続的なロギング用に 1G。 | Cisco NX-OS リリース 10.4(2)F |

# ThousandEyes Enterprise Agent のインストール

## 注意事項と制約事項

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

- NX-OS は、ThousandEyes Enterprise Agent バージョン5.1.1 以降をサポートしています。
- 10.4(2)F 以降では、コンテナインターフェイスにインターフェイス eth0 を使用することをお勧めします。eth0 以外のインターフェイスを使用した場合、アプリケーションはダウングレード直後には適切なネットワーク構成で起動しますが、アプリケーションを停止、非アクティブ化、または開始すると、デフォルトゲートウェイはインストールされず、アプリケーションへの接続は確立されません。ダウングレードの前に 10.4(2)F で eth1 を使用していた場合、ダウングレードの後、コンテナ内の構成が正しくても、ランタイム構成で

は、VNICがゲートウェイ（前面パネルポート）ではなく管理インターフェイスとして誤って反映されます。この問題を解決するには、 ThousandEyes Enterprise Agent を再構成してください。「ThousandEyes Enterprise Agent のインストール （226 ページ）」を参照してください。

# ThousandEyes Enterprise Agent のアプリケーション ホスティングの構成

Cisco NX-OS 10.4(2)F 以降では、ThousandEye Enterprise Agent 内で最大 4 つの IPv4 および IPv6 インターフェイスを構成できます。

手順に従って、ThousandEyes Enterprise Agent をインストールしてください。

## 手順の概要

1. **configure terminal**
2. **app-hosting bridge** *bridge-index*
3. **ip address** *ip-address/mask*
4. **vrf member** *name*
5. **exit**
6. **app-hosting appid** *name*
7. **app-vnic gateway bridge** *bridge-index* **guest-interface** *guest-interface-number*
8. **guest-ipaddress** *ip-address/mask*
9. **exit**
10. **app-default-gateway** *ip-address* **guest-interface***number*
11. **nameserver#** *ip-address*
12. **app-resource docker**
13. **run-opts** *options*
14. **prepend-pkg-opts**
15. **end**

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル コンフィギュレーション モードを開始します。 |
| ステップ 2 | **app-hosting bridge** *bridge-index*<br>例：<br>`switch(config)# app-hosting bridge 1` | アプリケーションホスティングブリッジを構成し、アプリケーション ホスティング ブリッジ構成モードを開始します。<br><1-8> ブリッジ インデックス |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 3 | **ip address** *ip-address/mask*<br>例：<br>`switch(config-app-hosting-bridge)# ip address 172.25.44.1/30` | アプリケーション コンテナへのゲートウェイとして機能するアプリケーション ブリッジ IPv4 アドレスを構成します。 |
| ステップ 4 | **vrf member** *name*<br>例：<br>`switch(config-app-hosting-bridge)# vrf member overlay-VRF` | VRF コンテキストを設定します。構成されていない場合は、VRF デフォルトの一部になります。 |
| ステップ 5 | **exit**<br>例：<br>`switch(config-app-hosting-bridge)# exit` | アプリケーションブリッジ構成モードを終了して、グローバル構成モードに戻ります。 |
| ステップ 6 | **app-hosting appid** *name*<br>例：<br>`switch(config)# app-hosting appid te_app` | アプリケーションを設定し、アプリケーション ホスティング コンフィギュレーション モードを開始します。 |
| ステップ 7 | **app-vnic gateway bridge** *bridge-index* **guest-interface** *guest-interface-number*<br>例：<br>`switch(config-app-hosting)# app-vnic bridge 1 guest-interface 0` | アプリケーションのゲスト VNIC インターフェイスを構成し、アプリケーション ホスティング vnic インターフェイス モードを開始します。 |
| ステップ 8 | **guest-ipaddress** *ip-address/mask*<br>例：<br>`switch(config-config-app-hosting-app-vnic)# guest-ipaddress 172.25.44.2/30` | ブリッジ 1 サブネットから利用可能な IPv4 アドレスの 1 つを構成します。 |
| ステップ 9 | **exit**<br>例：<br>`switch(config-config-app-hosting-vlan-access-ip)# exit` | アプリケーション vnic インターフェイス構成モードを終了して、アプリケーション ホスティング構成モードに戻ります。 |
| ステップ 10 | **app-default-gateway** *ip-address* **guest-interface***number*<br>例：<br>`switch(config-app-hosting-)# app-default-gateway 172.25.44.1` | bridge1 サブネットから利用可能な IPv4 アドレスを構成します。 |
| ステップ 11 | **nameserver#** *ip-address*<br>例：<br>`Device(config-app-hosting)# name-server0 10.2.2.2` | DNS サーバを設定します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 12 | **app-resource docker**<br>例：<br>`switch(config-app-hosting)# app-resource docker` | アプリケーション ホスティング Docker コンフィギュレーションモードを開始して、アプリケーションリソースの更新を指定します。 |
| ステップ 13 | **run-opts** *options*<br>例：<br>`switch(config-app-hosting-docker)# run-opts 1 "-e TEAGENT_ACCOUNT_TOKEN=[account-token]"`<br>`run-opts 10 "-e TEAGENT_DEF_IPV4_GW_ETH1=172.25.44.65"`<br>`run-opts 11 "-e TEAGENT_DEF_IPV6_GW_ETH1=2001:420:287:2003:7:110:1205:1"`<br>`run-opts 12 "-e TEAGENT_DEF_IPV4_GW_ETH2=172.25.44.73"`<br>`run-opts 13 "-e TEAGENT_DEF_IPV6_GW_ETH2=2001:420:287:2003:7:110:1206:1"`<br>`run-opts 14 "-e TEAGENT_DEF_IPV4_GW_ETH3=172.25.44.81"`<br>`run-opts 15 "-e TEAGENT_DEF_IPV6_GW_ETH3=2001:420:287:2003:7:110:1207:1"` | Docker ランタイムオプションを指定します。<br>**run-opts** オプションを使用して、追加のデフォルトゲートウェイを構成できます。 **TEAGENT_DEF_IPV[4/6]_GW_ETH[0-3]** コマンドは IPv4 および IPv6 を示すために使用され、インデックスは 0 ～ 3 です。 **app-default-gateway** コマンドを使用して設定できるゲートウェイは1つだけです。 |
| ステップ 14 | **prepend-pkg-opts**<br>例：<br>`switch(config-app-hosting-docker)# prepend-pkg-opts` | パッケージオプションを Docker ランタイムオプションとマージします。<br>重複する変数は上書きされます。 |
| ステップ 15 | **end**<br>例：<br>`switch(config-app-hosting-docker)# end` | アプリケーション ホスティング Docker コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。 |

### 例

次に、ThousandEye マルチインターフェイス構成の例を示します。

```
app-hosting bridge 1
ip address 172.25.44.113/29
ipv6 address 2001:420:287:2003:7:110:1210:1/125
app-hosting bridge 2
vrf member apphosting
ip address 172.25.44.97/30
ipv6 address 2001:420:287:2003:7:110:1208:1/126
app-hosting bridge 3
vrf member te
ip address 172.25.44.105/29
ipv6 address 2001:420:287:2003:7:110:1209:1/125
app-hosting bridge 4
vrf member vxlan_blue
ip address 172.25.44.33/30
ipv6 address 2001:420:287:2003:7:110:1203:1/114
app-hosting appid tea
app-vnic gateway bridge 1 guest-interface 0
guest-ipaddress 172.25.44.114/29
```

```
guest-ipv6address 2001:420:287:2003:7:110:1210:2/125
app-vnic gateway bridge 2 guest-interface 1
guest-ipaddress 172.25.44.98/30
guest-ipv6address 2001:420:287:2003:7:110:1208:2/126
app-vnic gateway bridge 3 guest-interface 2
guest-ipaddress 172.25.44.106/29
guest-ipv6address 2001:420:287:2003:7:110:1209:2/125
app-vnic gateway bridge 4 guest-interface 3
guest-ipaddress 172.25.44.34/30
guest-ipv6address 2001:420:287:2003:7:110:1203:2/114
app-default-gateway 172.25.44.97 guest-interface 1
app-default-ipv6-gateway 2001:420:287:2003:7:110:1208:1 guest-interface 1
app-resource docker
prepend-pkg-opts
run-opts 1 "-e TEAGENT_ACCOUNT_TOKEN=[account-token]"
run-opts 2 "--hostname=southlake2-1-Multi-UsrVRF"
run-opts 5 "-e TEAGENT_PROXY_TYPE=STATIC"
run-opts 6 "-e TEAGENT_PROXY_LOCATION=proxy.domainname.com:80"
run-opts 7 "-e TEAGENT_PROXY_BYPASS_LIST=T*.domainname.com"
run-opts 8 "--dns 8:8:8:8"
run-opts 9 "--dns 8::8"
run-opts 10 "-e TEAGENT_DEF_IPV4_GW_ETH0=172.25.44.113"
run-opts 11 "-e TEAGENT_DEF_IPV6_GW_ETH0=2001:420:287:2003:7:110:1210:1"
run-opts 14 "-e TEAGENT_DEF_IPV4_GW_ETH2=172.25.44.105"
run-opts 15 "-e TEAGENT_DEF_IPV6_GW_ETH2=2001:420:287:2003:7:110:1209:1"
run-opts 16 "-e TEAGENT_DEF_IPV4_GW_ETH3=172.25.44.33"
run-opts 17 "-e TEAGENT_DEF_IPV6_GW_ETH3=2001:420:287:2003:7:110:1203:1"
```

# ThousandEyes Enterprise Agent のインストール

### 始める前に

ThousandEyes Enterprise Agent は、以下の URL またはフラッシュ ファイル システムからインストールできます。

ThousandEyes Enterprise Agentは、スイッチのブートフラッシュにあるファイルからインストールできます。

### 手順の概要

1. **enable**
2. **app-hosting install appid** *application-name* **package** *package-path*
3. **app-hosting activate appid** *application-name*
4. **app-hosting start appid** *application-name*
5. **end**

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **enable**<br>例：<br>`switch> enable` | 特権 EXEC モードを開始します。 |
| ステップ 2 | **app-hosting install appid** *application-name* **package** *package-path*<br>例：<br>`switch# app-hosting install 1keyes https://app.thousandeyes.com/install/download/enterprise-agent/thousandeyes-enterprise-agent-x86_64.cisco.tar`<br><br>または<br>`switch# app-hosting install appid 1keyes package flash:/apps/[greenfield-app-tar]`<br>`switch# app-hosting install appid 1keyes package bootflash:[file path]` | 指定した場所からアプリケーションをインストールします。 |
| ステップ 3 | **app-hosting activate appid** *application-name*<br>例：<br>`switch# app-hosting activate appid 1keyes` | アプリケーション ホスティング構成モードをアクティブにします。 |
| ステップ 4 | **app-hosting start appid** *application-name*<br>例：<br>`switch# app-hosting start appid 1keyes` | （オプション）アプリケーションを開始します。 |
| ステップ 5 | **end**<br>例：<br>`switch# end` | アプリケーション ホスティング コンフィギュレーションモードを終了し、特権EXECモードに戻ります。 |

次に、**show app-hosting list** コマンドの出力例を示します。

```
switch# show app-hosting list

App id                                    State
---------------------------------------------------------
1keyes                                    RUNNING
```

# ThousandEyes Enterprise Agent の構成例

## 例：ThousandEyes Enterprise Agent のインストール

次の例は、AppHosting 機能を有効にする方法を示しています。

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

次の例は、AppHosting を設定する例を示しています。

```
switch> enable
switch# configure terminal
switch(config)# app-hosting bridge 2
switch(config)# vrf member red
switch(config)# ip address 172.25.44.1/28
switch(config)# app-hosting appid 1keyes
switch(config-app-hosting)# app-vnic gateway bridge 1 guest-interface 0
switch(config-config-app-hosting)# guest-ipaddress 172.25.44.3/28
switch(config-config-app-hosting)# exit
switch(config-app-hosting)# app-default-gateway 172.25.44.1 guest-interface 0
switch(config-app-hosting)# name-server0 10.2.2.2
switch(config-app-hosting)# app-resource docker
switch(config-app-hosting-docker)# run-opts 1 "-e TEAGENT_ACCOUNT_TOKEN=[account-token]"
switch(config-app-hosting-docker)# prepend-pkg-opts
switch(config-app-hosting-docker)# end
```

次の例は、SVI を構成する方法を示しています。

```
switch(config)# interface Vlan606
switch(config-if)# no shutdown
switch(config-if)# vrf member red
switch(config-if)# no ip redirects
switch(config-if)# ip address 172.30.2.193/26
switch(config-if)# ip proxy-arp
switch(config-if)# ip local-proxy-arp
switch(config-if)# interface Ethernet1/15
switch(config-if)# switchport
switch(config-if)# switchport mode trunk
switch(config-if)# switchport trunk allowed vlan 606
switch(config-if)# no shutdown
```

次の例は、ThousandEyes Enterprise Agent をインストールする方法を示しています。

（注） フラッシュ ファイル システムから事前にパッケージ化された Greenfield アプリケーションをインストールできます。

```
Device> enable
Device# Device# app-hosting install 1keyes https://downloads.thousandeyes.com/
enterprise-agent/thousandeyes-enterprise-agent-3.0.cat9k.tar
```

# ThousandEyes Enterprise Agent の設定例

次に、**show app-hosting detail** コマンドの出力例を示します。

```
switch# show app-hosting detail
App id : 1keyes
Owner : appmgr
State : DEPLOYED
Application
Type : docker
Name : ThousandEyes Enterprise Agent
Version : 4.0.2
Description :
Author : ThousandEyes <support@thousandeyes.com>
Path : /bootflash/thousandeyes-enterprise-agent-4.0.2.cisco.tar.gz
URL Path :
Activated profile name : custom
Resource reservation
Memory : 2048 MB
Disk : 51 MB
CPU : 7400 units
Platform resource profiles
Profile Name CPU(unit) Memory(MB) Disk(MB)
--------------------------------------------------------------
Attached devices
Name Type Alias
---------------------------------------------
iox_trace serial/trace serial3
iox_syslog serial/syslog serial2
iox_console_aux serial/aux serial1
iox_console_shell serial/shell serial0
Network interfaces
---------------------------------------
Docker
------
Run-time information
Command :
Entry-point :
Run options in use :
Package run options :
Application health information
Status : 0
Last probe error :
Last probe output :
```

次に、プロキシ サーバーが存在しない場合の、**show running-configuration** コマンドの出力例を示します。

```
switch# show running-config app-hosting

feature app-hosting
app-hosting signed-verification disable
app-hosting bridge 2
ip address 172.25.44.33/28
app-hosting appid 1keyes
app-vnic gateway bridge 2 guest-interface 0
guest-ipaddress 172.25.44.35/28
app-default-gateway 172.25.44.33 guest-interface 0
name-server0 171.70.168.183
name-server1 173.36.131.10
app-resource docker
prepend-pkg-opts
run-opts 1 "-e TEAGENT_ACCOUNT_TOKEN = [account-token]"
```

```
run-opts 2 "--hostname=southlake2-1"
run-opts 3 "--cap-add=NET_ADMIN"
run-opts 4 "--mount type=tmpfs,destination=/var/log/agent,tmpfs-size=140m"
run-opts 5 "-e TEAGENT_PROXY_TYPE=STATIC"
run-opts 6 "-e TEAGENT_PROXY_LOCATION=proxy.domainname.com:80"
run-opts 7 "-e TEAGENT_PROXY_BYPASS_LIST=T*.domainname.com"
```

# 第 IV 部

# NX-API


- NX-API CLI （237 ページ）
- NX-API REST （287 ページ）
- NX-API 開発者サンドボックス （293 ページ）

第 21 章

# NX-API CLI


- NX-API CLI について （237 ページ）
- NX-API CLI の使用 （240 ページ）
- カーネル スタック ACL （271 ページ）
- NX-API 応答コードの表 （273 ページ）
- JSON および XML 構造化出力 （276 ページ）
- サンプル NX-API スクリプト （285 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。
- Cisco NX-OS リリース 10.2(1)F 以降では、 **system server session cmd-timeout** を使用してタイムアウトを増やすことができます。

- 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（273 ページ）を参照してください。

NX-API CLI は、ローカル アクセスに対してはデフォルトで有効になっています。リモート HTTP アクセスに対してはデフォルトで無効になっています。

次の例は、NX-API CLI を構成して起動する方法を示しています。

- 管理インターフェイスを有効にします。

```
switch# conf t
Enter configuration commands, one per line.
End with CNTL/Z.
switch(config)# interface mgmt 0
switch(config-if)# ip address 10.126.67.53/25
switch(config-if)# vrf context managment
switch(config-vrf)# ip route 0.0.0.0/0 10.126.67.1
switch(config-vrf)# end
switch#
```

- NX-API **nxapi** 機能を有効にします。

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

次の例は、リクエストとそのレスポンスを XML 形式で示しています。

要求:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ins_api>
  <version>0.1</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>session1</sid>
  <input>show switchname</input>
  <output_format>xml</output_format>
</ins_api>
```

応答：

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

次の例は、JSON 形式の要求とその応答を示しています。

要求:

```
{
    "ins_api": {
        "version": "0.1",
        "type": "cli_show",
        "chunk": "0",
        "sid": "session1",
        "input": "show switchname",
        "output_format": "json"
    }
}
```

応答：

```
{
    "ins_api": {
        "type": "cli_show",
        "version": "0.1",
        "sid": "eoc",
        "outputs": {
            "output": {
                "body": {
                    "hostname": "switch"
                },
                "input": "show switchname",
                "msg": "Success",
```

```
                "code": "200"
            }
        }
    }
}
```

（注） ユーザーを削除しようとすると失敗し、次のようなエラー メッセージが約 12 時間ごとに表示されるという既知の問題があります。

`user delete failed for *username*:userdel: user *username* is currently logged in - securityd`

この問題は、NX-API を介してスイッチにログインしているユーザーを削除しようとした場合に発生する可能性があります。この場合、次のコマンドを入力して、最初にユーザーのログアウトを試行します。

`switch(config)# **clear user *username***`

その後、ユーザーの削除を再試行します。回避策を試みても問題が解決しない場合は、Cisco TAC へお問い合わせください。

# NX-API で権限を root にエスカレーションする

NX-API では、管理者ユーザーの権限を root アクセスの権限にエスカレーションできます。

以下は、権限をエスカレーションするためのガイドラインです：

- 特権を root にエスカレーションできるのは管理者ユーザーのみです。
- root へのエスカレーションはパスワードで保護されています。

次の例は、管理者の権限を root にエスカレーションする方法と、エスカレーションを確認する方法を示しています。root になっても、**whoami** コマンドを実行すると admin として表示されることに注意してください。ただし、admin アカウントにはすべての root 権限があります。

最初の例：

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo su root ; whoami</input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>admin </body>
```

```
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

2 番目の例：

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo cat path_to_file </input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>[Contents of file]</body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

# NX-API 管理コマンド

次の表にリストされている CLI コマンドを使用して、NX-API を有効にして管理できます。

**表 *8 : 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。<br>（注）<br>「 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 {httpscrt certfile** \| **httpskey keyfile**} *filename* | 次のアップロードを指定します：<br>• httpscrt が指定されている場合の HTTPS 証明書。<br>• httpskey が指定されている場合の HTTPS キー。<br>HTTPS 証明書の例：<br>`nxapi certificate httpscrt certfile bootflash:cert.crt`<br>HTTPS キーの例：<br>`nxapi certificate httpskey keyfile bootflash:privkey.key` |
| ファイル名パスフレーズ **nxapi certificatehttpskey keyfile password** | 暗号化された秘密キーを使用してNX-API証明書をインストールします。<br>（注）<br>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。<br>例:<br>`nxapi certificate httpskey keyfile bootflash:encr-cc.pem password pass123!` |
| **nxapi certificate enable** | 証明書を有効化します。 |

| NX-API 管理コマンド | 説明 |
| --- | --- |
| **nxapi certificate trustpoint <trustpoint label>** | |

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

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

| NX-API 管理コマンド | 説明 |
| --- | --- |
| **nxapi certificate sudi** | この CLI は、Secure Unique Device Identifier（SUDI）を使用してデバイスを安全に認証する方法を提供します。<br><br>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。<br><br>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。<br><br>（注）<br>• 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。<br>• 「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。<br>• 「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。<br>• 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。<br>• 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。<br>• Cisco NX-OS リリース 10.5(2)F 以降、「nxapi certificate sudi」は Cisco NX-OS スイッチの FIPS モードでブロックされます。 |
| **no nxapi certificate sudi** | これにより、SUDI が無効になり、NX-API にはデフォルトの自己署名証明書が付属します。 |

| NX-API 管理コマンド | 説明 |
| --- | --- |
| **nxapi ssl-ciphers weak** | Cisco NX-OS リリース 9.2(1) 以降、弱い暗号はデフォルトで無効になっています。このコマンドを実行すると、デフォルトの動作が変更され、NGINX の弱い暗号が有効になります。このコマンドの no 形式を使用すると、デフォルトに変更されます（デフォルトでは、弱い暗号は無効になります）。 |

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

| NX-API 管理コマンド | 説明 |
|---|---|
| **nxapi use-vrf** *vrf* | デフォルト VRF、管理 VRF、または名前付き VRF を指定します。<br>（注）<br>Cisco NX-OS リリース 7.0(3)I2(1) では、NGINX は 1 つの VRF でのみリッスンします。 |
| **system server session cmd-timeout <timeout>** | Cisco NX-OS リリース 10.2(3)F 以降、NGINX サーバーでは、コマンドを実行するためのデフォルトのタイムアウトは5分です。ユーザは、必要に応じて、およびコマンドの実行にかかる時間に応じて、タイムアウトを 60 秒（1 分）から 3600 秒（1 時間）の任意の値に増やすことができます。 |
| **ip netns exec management iptables** | アクセス制限を実装し、管理 VRF で実行できます。<br>（注）<br>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。<br>*Iptables* は、ポリシーチェーンを使用してトラフィックを許可またはブロックするコマンドライン ファイアウォールユーティリティであり、ほとんどの場合、*Linux* ディストリビューションにプリインストールされています。<br>（注）<br>iptables が bash シェルで変更されたときに、リロード後も iptables を永続化する方法の詳細については、「」を参照してください。リロード間で Iptable を永続化する（270 ページ） |
| **nxapi idle-timeout <timeout>** | リリース 9.3(5) 以降では、アイドル状態の NX-API セッションが無効になるまでの時間を設定できます。指定できる時間は 1 ～ 1440 分です。デフォルトの時間は 10 分です。デフォルト値に戻すには、このコマンドの no 形式を使用します。 **no nxapi idle-timeout <timeout>** |

次に、SUDI の NX-API 出力の例を示します。

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

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

次に、トラストポイントの設定例を示します。

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

以下は、HTTPS 証明書の正常なアップロードの例です：

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

（注） 証明書を有効にする前に、証明書とキーを設定する必要があります。

以下は、HTTPS キーの正常なアップロードの例です：

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

次に、暗号化された NXAPI サーバー証明書をインストールする方法の例を示します。

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!

switch(config)#nxapi certificate enable
switch(config)#
```

状況によっては、証明書が無効であることを示すエラーメッセージが表示されることがあります。

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
```

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
ERROR: Unable to load private key!
Check keyfile or provide pwd if key is encrypted, using 'nxapi certificate httpskey
keyfile <keyfile> password <passphrase>'.
```

この場合、filename passphrase を使用して暗号化キー ファイルのパスフレーズを指定する必要があります。**nxapi certificatehttpskey keyfile  password**

これが問題の原因である場合、証明書を正常にインストールできるはずです。

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!
switch(config)# nxapi certificate enable
switch(config)#
```

# NX-API を使用したインタラクティブ コマンドの操作

対話型コマンドの確認プロンプトを無効にし、エラーコード500によるタイムアウトを回避するには、対話型コマンドの前に**[端末の dont-ask（terminal dont-ask）]** を追加します。 **を使用。複数の対話型コマンドを区切るには**、それぞれが。は単一のブランク文字で囲まれています。

エラー コード 500 でのタイムアウトを回避するために**端末の dont-ask** を使用する対話型コマンドの例をいくつか次に示します：

```
terminal dont-ask ; reload module 21
```

```
terminal dont-ask ; system mode maintenance
```

# NX-API 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 （239 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 のリストが含められます。クライアントの応答には、NX-API クライアントが使用している特定のユーザー名の証明書など、追加の情報要素が含まれます。

NX-API クライアントは基本認証、証明書認証のいずれかを使用するように構成することができます。または証明書を優先するものの、証明書認証方式が使用できない場合は基本認証にフォールバックするように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。
- 証明書またはトラストポイントが削除されても、NX-API クライアント証明書認証の構成を明示的に変更するか、no feature nxapi コマンドを使用しない限り、その証明書の NX-API は引き続き機能します。証明書が NX-API に関連付けられたトラストポイントに存在することを確認してください。そうでない場合、reload ascii Nnxapi は開始に失敗します。これは、次の**show nxapi**コマンドを使用して確認できます。

  NX-API に関連付けられた証明書とトラストポイントが再度追加されると、NX-API は自動的に再起動し、動作を継続します。

  暗号化証明書を使用して ascii または ISSU をリロードしているときに、システムの準備後にすべての証明書が復元されるまで、NGINX が複数回再起動する場合があります。NX-API を使用するために証明書が復元されるのを待ちます。
- 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 16 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 httpscert 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 {httpscrt certfile | httpskey 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。

### 手順の概要

1. この機能の前提条件が満たされていることを確認します。
2. **config terminal**
3. **nxapi client certificate authentication [{optional | strict | two-step}]**

### 手順の詳細

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（256 ページ）」を参照してください。 |
| ステップ 2 | **config terminal**<br>例：<br>`switch-1# config terminal`<br>`Enter configuration commands, one per line. End with CNTL/Z.`<br>`switch-1(config)#` | 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 3 | **nxapi client certificate authentication [{optional \| strict \| two-step}]**<br>例：<br>`switch-1# nxapi client certificate authentication strict`<br>`switch-1(config)#` | 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。<br>• **optional** はクライアント証明書を要求します。<br>• 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。<br>• 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。<br>• 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。<br>• **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。<br>• **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。<br>（注）<br>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。<br>`No trustpoints configured! Please configure trustpoint using 'crypto ca trustpoint <trustpoint-label>' and associated commands, and then enable this feature.` |

## 証明書認証用の Python スクリプトの例

次の例は、認証用のクライアント証明書を使用した Python スクリプトを示しています。

```
import requests
import json

"""
Modify these please
"""
switchuser='USERID'
switchpassword='PASSWORD'
mgmtip='NXOS MANAGEMENT IP/DOMAIN NAME'

client_cert_file='PATH_TO_CLIENT_CERTIFICATE'
client_key_file='PATH_TO_CLIENT_KEY_FILE'
ca_cert='PATH_TO_CA_CERT_THAT_SIGNED_NXAPI_SERVER_CERT'
```

```
url='https://' + mgmtip + '/ins'
myheaders={'content-type':'application/json-rpc'}
payload=[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 1
  }
]
response = requests.post(url,data=json.dumps(payload),
headers=myheaders,auth=(switchuser,switchpassword),cert=(client_cert_file_path,client_key_file),verify=ca_cert).json()
```

必要に応じて、スクリプトを変更できます。

- クライアント証明書認証モードによっては、スイッチパスワードをヌル値に設定することで（`switchpassword=` ）、スイッチ パスワードを省略できます。
  - **optional** および **strict** モードの場合、`switchpassword=` は空白のままにできます。この場合、NX-API はユーザー名とクライアント証明書のみに基づいてクライアントを認証します。
  - **two-step** モードの場合、パスワードが必要なため、`switchpassword=` の値を指定する必要があります。
- POST コマンドで `verify=False` を設定することで、NX-API サーバの証明書が有効であることの確認をバイパスできます。

## cURL 証明書要求の例

次に、NX-API クライアント認証用の正しく構造化された cURL 証明書要求の例を示します。

```
/usr/bin/curl --user admin: --tlsv1.2 --cacert ./ca.pem --cert ./user.crt:pass123! --key
 ./user.key -v -X POST -H "Accept: application/json" -H "Content-type: application/json"
 --data '{"ins_api":{"version": "1.0", "type": "cli_show", "chunk": "0", "sid": "1",
"input": "show clock","output_format": "json"}}' https://<device-management-ip>:443/ins
```

### 構文要素

次の表は、この要求で使用されるパラメータを示しています。

| パラメータ | 説明 |
|---|---|
| **--user** | ユーザ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。<br>ユーザーのパスワードを指定するには、コロンの後にパスワードを指定します。例：--user username:password |

| パラメータ | 説明 |
|---|---|
| **--cacert** | NX-API サーバー証明書に署名した CA へのパスを使用します。<br>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。<br>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。

### 手順の概要

1. ユーザーまたはクッキーのエラーを確認します。
2. 証明書に誤りがないか確認してください。
3. エラーが発生した場合は、 **no nxapi client certificate authentication** 、それから **nxapi client certificate authentication** を発行して、トラストポイント、CA、CRL、または NX-OS 証明書機能に対する変更をリロードするように機能をフラップします。

### 手順の詳細

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：<br>• 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<br>• 認証ヘッダーで指定されたユーザーが正しくない<br>• 無効なクッキーが提供された |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | • 認証ヘッダーのユーザー名とクライアント証明書の CN フィールドのユーザー名が一致しない<br><br>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：<br><br>• 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。<br>`{{{`<br>`"code": "400",`<br>`"msg": "Authentication failure - user not found."`<br>`}}}`<br><br>• 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。<br>`{{`<br>`"code": -32004,`<br>`"message": "Invalid username or password"`<br>`}}` |
| **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。<br><br>• 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。<br><br>• 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。<br><br>次に例を示します。<br>`<html>`<br>`<head><title>400 The SSL certificate error</title></head>`<br>`<body bgcolor="white">`<br>`<center><h1>400 Bad Request</h1></center>`<br>`<center>The SSL certificate error</center>`<br>`<hr><center>nginx/1.7.10</center>`<br>`</body>`<br>`</html>` |
| **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 *9 : 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。<br>次のタイプのコマンドがサポートされています。<br>• **cli_show**<br>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。<br>• **cli_show_array**<br>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。<br>• **cli_show_ascii**<br>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。<br>• **cli_conf**<br>CLI 構成コマンド<br>• **bash**<br>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。<br>（注）<br>• 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。<br>• 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。<br>• 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。<br>• 対話型コマンドはサポートされていません。 |

<table>
<tr><th>NX-API リクエスト要素</th><th>説明</th></tr>
<tr><td>チャンク</td><td>一部の show コマンドは、大量の出力を返す場合があります。コマンド全体が完了する前に NX-API クライアントが出力の処理を開始するために、NX-API は show コマンドの出力チャンクをサポートしています。<br><br>次の設定を有効または無効にできます。<br><br>（注）<br><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><br>（注）<br>• 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。<br>• 出力メッセージ形式のオプションは、XML または JSON です。<br>• 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）。<br>XML SAX を使用して、チャンクされた出力を解析できます。<br>• 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。<br><br>（注）<br>チャンクが有効になっている場合、現在サポートされている最大メッセージ サイズは、チャンク出力の 200MB です。</td></tr>
</table>

| NX-API リクエスト要素 | 説明 |
| --- | --- |
| ロールバック | コンフィギュレーション CLI に対してのみ有効であり、show コマンドに対しては有効ではありません。コンフィギュレーション ロールバック オプションを指定します。次のいずれかのオプションを指定します。<br>• Stop-on-error：最初に失敗した CLI で停止します。<br>• Continue-on-error：他の CLI を無視して続行します。<br>• 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。<br>（注）<br>入力要求形式が XML または JSON の場合、ロールバック要素は cli_conf モードで使用できます。 |
| *sid* | セッション ID 要素は、応答メッセージがチャンクされている場合にのみ有効です。メッセージの次のチャンクを取得するには、前の応答メッセージの *sid* と一致する *sid* を指定する必要があります。<br>NX-OS リリース 9.3(1) では、*sid* オプション `clear` が導入されています。*sid* を `clear` に設定して新しいチャンク リクエストが開始されると、現在のチャンク リクエストはすべて破棄または破棄されます。<br>応答コード `429` を受け取った場合：同時チャンク リクエストの最大数は `2` です。*sid* `clear` を使用して、現在のチャンク リクエストを破棄します。*sid* `clear` を使用した後、後続の応答コードは、残りのリクエストに対して通常どおり動作します。 |

<table>
<tr><th>NX-API リクエスト要素</th><th>説明</th></tr>
<tr><td>input</td><td>入力は 1 つのコマンドまたは複数のコマンドです。ただし、異なるメッセージ タイプに属するコマンドを混在させてはなりません。たとえば、show コマンドは cli_show メッセージタイプであり、cli_confモードではサポートされません。<br><br>（注）<br>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。）<br><br>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。<br><br><pre>terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport</pre><br>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。）<br><br>以下は、複数のコマンドの例です。<br><br>（注）<table><tr><td>cli_show</td><td><pre>show version ; show interface brief ; show vlan</pre></td></tr><tr><td>cli_conf</td><td><pre>interface Eth4/1 ; no shut ; switchport</pre></td></tr><tr><td>bash</td><td><pre>cd /bootflash;mkdir new_dir</pre></td></tr></table></td></tr>
</table>

| NX-API リクエスト要素 | 説明 |
|---|---|
| *output_format* | 使用可能な出力メッセージ形式は次のとおりです。<br>（注）<br>xml：XML 形式を指定します。<br>json：JSON 形式で出力を指定します。<br>（注）<br>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。<br>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。<br>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。 |

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 *10 : JSON-RPC* 形式の *NX-API* 要求要素**

| NX-API リクエスト要素 | 説明 |
|---|---|
| *jsonrpc* | JSON-RPC プロトコルのバージョンを指定する文字列。<br>バージョンは 2.0 であることが必要です。 |
| *method* | 呼び出されるメソッドの名前を含む文字列。<br>NX-API は、次のいずれかをサポートします。<br>• **cli**：show または構成コマンド<br>• **cli_ascii**：show または構成コマンド。フォーマットせずに出力<br>• **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 |
|---|---|
| *params* | メソッドの呼び出し中に使用されるパラメータ値を保持する構造化された値。<br>以下が含まれている必要があります。<br>• **cmd**：CLI コマンド<br>• **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。<br>• Stop-on-error：最初に失敗した CLI で停止します。<br>• Continue-on-error：失敗した CLI を無視し、他の CLI を続行します。<br>• 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。<br>• Validate-Only：設定を検証しますが、設定は適用しません。<br>• 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。 |
| *id* | クライアントによって確立されるオプションの識別子。指定されている場合は、文字列、数値、または null 値を含む必要があります。値は null にならないはずです。数値には小数部を含めません。ユーザーが id パラメータを指定しなかった場合、サーバーはリクエストが単なる通知であるとみなし、応答はしません。パラメータは*id*: 1 などのように指定します。 |

# NX-API 応答要素

CLI コマンドに応答する NX-API 要素を次の表に示します。

表 *11 : 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。<br>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。<br>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。<br>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。<br>NX-API は、ハイパーテキスト転送プロトコル（HTTP）ステータス コード レジストリ（http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml）で説明されている標準規格の HTTP原因コードを使用します。 |
| msg | 返された原因コードに関連付けられたエラー メッセージ。 |

# NX-API へのアクセスの制限

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

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

## iptable の更新

iptable を使用すると、VRF が NX-API 通信用に設定されている場合に、デバイスへの HTTP または HTTPS アクセスを制限できます。このセクションでは、既存の iptable への HTTP および HTTPS アクセスをブロックするルールを追加、確認、および削除する方法を示します。

### 手順

**ステップ 1** HTTP アクセスをブロックするルールを作成するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp  --dport 80 -j DROP
```

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

**ステップ 2** HTTPS アクセスをブロックするルールを作成するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp  --dport 443 -j DROP
```

**ステップ 3** 適用されたルールを確認するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -L


Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:http
DROP       tcp  --  anywhere             anywhere             tcp dpt:https

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

**ステップ 4** ポート 80 への 10.155.0.0/24 サブネットを持つすべてのトラフィックをブロックするルールを作成して確認するには、次の手順を実行します。

```
bash-4.3# ip netns exec management iptables -A INPUT -s 10.155.0.0/24 -p tcp --dport 80 -j DROP
bash-4.3# ip netns exec management iptables -L


Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  10.155.0.0/24        anywhere             tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

**ステップ 5** 以前に適用したルールを削除して確認するには、次の手順を実行します。

この例では、最初のルールを INPUT から削除します。

```
bash-4.3# ip netns exec management iptables -D INPUT 1
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

### 次のタスク

iptables のルールを bash シェルで変更した場合、リロード後は保持されません。ルールを永続的にするには、を参照してください。リロード間で Iptable を永続化する （270 ページ）

## リロード間で Iptable を永続化する

iptable のルールを bash シェルで変更した場合、リロード後は保持されません。このセクションでは、リロード後も変更された iptable を永続化する方法について説明します。

### 始める前に

iptable を変更したとします。

### 手順

**ステップ 1** iptables_init.log という名前のファイルを /etc ディレクトリに作成します：

```
bash-4.3# touch /etc/iptables_init.log; chmod 777 /etc/iptables_init.log
```

**ステップ 2** iptable の変更を保存する /etc/sys/iptables ファイルを作成します：

```
bash-4.3# ip netns exec management iptables-save > /etc/sysconfig/iptables
```

**ステップ 3** 次の一連のコマンドを使用して、/etc/init.d ディレクトリに「iptables_init」という起動スクリプトを作成します：

```
#!/bin/sh

### BEGIN INIT INFO

# Provides:          iptables_init

# Required-Start:
```

```
# Required-Stop:

# Default-Start:     2 3 4 5

# Default-Stop:

# Short-Description: init for iptables

# Description:       sets config for iptables

#                    during boot time

### END INIT INFO


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
  start)
    start_script
   ;;
  stop)
    ;;
  restart)
    sleep 1
    $0 start
    ;;
  *)
    echo "Usage: $0 {start|stop|status|restart}"
    exit 1
esac
exit 0
```

**ステップ 4** 起動スクリプトに適切な権限を設定します：

```
bash-4.3# chmod 777 /etc/init.d/iptables_int
```

**ステップ 5** chkconfig ユーティリティを使用して、「iptables_int」起動スクリプトを「オン」に設定します：

```
bash-4.3# chkconfig iptables_init on
```

「iptables_init」起動スクリプトは、リロードを実行するたびに実行されます。これで iptable ルールを永続的にすることができました。

# カーネル スタック ACL

カーネル スタック ACL は、インバンドコンポーネントとアウトバンドコンポーネントを管理するための ACL を構成するための一般的な CLI インフラストラクチャです。

カーネル スタック ACL は、NX-OS ACL CLI を使用して、管理およびフロント パネル ポート上の管理アプリケーションを保護します。単一の ACL を設定することで、NX-OS 上のすべての管理アプリケーションを保護できる必要があります。

カーネル スタック ACL は、ユーザーの手動介入を修正し、ACL が mgmt0 インターフェイスに適用されるときに iptable エントリを自動的にプログラムするコンポーネントです。

以下は、カーネル スタック ACL を構成する例です。

```
swtich# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# ip access-list kacl1
switch(config-acl)# statistics per-entry
switch(config-acl)# 10 deny tcp any any eq 443
switch(config-acl)# 20 permit ip any any
switch(config-acl)# end
switch#
```

```
switch(config-if)# interface mgmt0
switch(config-if)#   ip access-group acl1 in
switch(config-if)#   ipv6 traffic-filter acl6 in
switch(config-if)#
```

```
switch# sh ip access-lists kacl1
IP access list kacl1
statistics per-entry
10 deny tcp any any eq 443 [match=136]
20 permit ip any any [match=44952]
switch(config)#
```

以下は、構成に基づいた iptables エントリのカーネル スタック フィルタリングです。

```
bash-4.4# ip netns exec management iptables -L -n -v --line-numbers
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num pkts bytes target prot opt in out source destination
1 9 576 DROP tcp -- * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:443
2 0 0 ACCEPT all -- * * 0.0.0.0/0 0.0.0.0/0
3 0 0 DROP all -- * * 0.0.0.0/0 0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num pkts bytes target prot opt in out source destination
bash-4.4#
```

カーネル スタック ACL サポートの制限は次のとおりです。

- この機能は、mgmt0 インターフェイスでのみサポートされ、他のインバンドインターフェイスではサポートされません。
- ACL エントリの 5 つのタプル（protocol、source-ip、destination-ip、source-port、および destination-port）は、iptables にプログラムされています。ACL エントリで提供される残りのオプションは iptables でプログラムされておらず、そのような場合に警告の syslog をスローします。

  たとえば、「警告: 一部の ACL オプションは kstack ではサポートされていません。部分的なルールのみがインストールされます。」

- デバイス ユーザーがホスト bash アクセス権を持っている場合、ユーザーは手動で iptables を更新できます。この更新により、プログラムされている iptable ルールが破損する可能性があります。
- 検証される ACE の最大数は、IPv4 トラフィックの場合は 100、IPv6 トラフィックの場合は加えてさらに 100 です。このスケール以上を適用すると、スループットに影響を与える可能性があります。

# 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 *12 : NX-API* 応答コード

| [NX-API 応答（NX-API Response）] | コード | メッセージ |
|---|---|---|
| 成功 | 200 | 成功。 |
| CUST_OUTPUT_PIPED | 204 | 要求により、出力は別の場所にパイプされます。 |
| BASH_CMD_ERR | 400 | Bash コマンド エラー。 |
| CHUNK_ALLOW_ONE_CMD_ERR | 400 | チャンクは、1 つのコマンドだけを受け入れます。 |
| CLI_CLIENT_ERR | 400 | CLI の実行エラー |
| CLI_CMD_ERR | 400 | CLI コマンド エラーの入力。 |
| EOC_NOT_ALLOWED_ERR | 400 | eoc 値は、リクエストのセッション ID として許可されていません。 |
| IN_MSG_ERR | 400 | 着信メッセージが無効です。 |
| INVALID_REMOTE_IP_ERR | 400 | 要求のリモートIPを取得できません。 |

| | | |
|---|---|---|
| MSG_VER_MISMATCH | 400 | メッセージ バージョンの不一致 |
| NO_INPUT_CMD_ERR | 400 | 入力コマンドがありません。 |
| SID_NOT_ALLOWED_ERR | 400 | セッション ID として入力された文字が無効です。 |
| PERM_DENY_ERR | 401 | 権限が拒否されました。 |
| CONF_NOT_ALLOW_SHOW_ERR | 405 | 構成モードは **[表示（show）]** を許可しません。 |
| SHOW_NOT_ALLOW_CONF_ERR | 405 | 表示モードでは構成できません。 |
| EXCEED_MAX_SHOW_ERR | 413 | 連続する show コマンドの最大数を超えました。最大値は 10 です。 |
| MSG_SIZE_LARGE_ERR | 413 | 応答サイズが大きすぎます。 |
| RESP_SIZE_LARGE_ERR | 413 | 応答サイズが最大メッセージ サイズを超えたため、処理を停止しました。最大サイズは 200 MB です。 |
| EXCEED_MAX_INFLIGHT_CHUNK_REQ_ERR | 429 | 同時チャンク リクエストの最大数は超えています。最大は 2 です。 |
| MAX_SESSIONS_ERR | 429 | 最大セッション数に到達しました。新しいユーザー/クライアントの場合は、しばらくしてからもう一度お試しください。 |
| OBJ_NOT_EXIST | 432 | 要求したオブジェクトが存在しません。 |
| BACKEND_ERR | 500 | バックエンド処理エラー。 |
| CREATE_CHECKPOINT_ERR | 500 | チェックポイントの作成をするエラー。 |
| DELETE_CHECKPOINT_ERR | 500 | チェックポイントの削除中にエラーが発生しました。 |
| FILE_OPER_ERR | 500 | システム内部ファイル操作エラー。 |
| LIBXML_NS_ERR | 500 | システムの内部 LIBXML NS エラー。これは要求フォーマットのエラーです。 |
| LIBXML_PARSE_ERR | 500 | システムの内部 LIBXML 解析エラー。これは要求フォーマットのエラーです。 |
| LIBXML_PATH_CTX_ERR | 500 | システムの内部 LIBXML パス コンテキスト エラー。これは要求フォーマットのエラーです。 |

| | | |
|---|---|---|
| MEM_ALLOC_ERR | 500 | システムの内部メモリ割り当てエラー。 |
| ROLLBACK_ERR | 500 | ロールバックの実行中にエラーが発生しました。 |
| SERVER_BUSY_ERR | 500 | サーバーがビジー状態のため、リクエストは拒否されました。 |
| USER_NOT_FOUND_ERR | 500 | 入力またはキャッシュからユーザーが見つかりません。 |
| VOLATILE_FULL | 500 | 揮発性メモリは一杯です。メモリ スペースを解放して、再試行してください。 |
| XML_TO_JSON_CONVERT_ERR | 500 | XML から JSON への変換エラー。 |
| BASH_CMD_NOT_SUPPORTED_ERR | 501 | Bash コマンドはサポートされていません。 |
| CHUNK_ALLOW_XML_ONLY_ERR | 501 | チャンクは XML 出力のみを許可します。 |
| CHUNK_ONLY_ALLOWED_IN_SHOW_ERR | 501 | 応答のチャンクは、`show` コマンドでのみ許可されます。 |
| CHUNK_TIMEOUT | 501 | チャンク応答の生成中にタイムアウトしました。 |
| CLI_CMD_NOT_SUPPORTED_ERR | 501 | CLI コマンドはサポートされていません。 |
| JSON_NOT_SUPPORTED_ERR | 501 | 大量の出力の可能性があるため、JSON はサポートされていません。 |
| MALFORMED_XML | 501 | 不正な XML 出力。 |
| MSG_TYPE_UNSUPPORTED_ERR | 501 | メッセージ タイプはサポートされていません |
| OUTPUT_REDIRECT_NOT_SUPPORTED_ERR | 501 | 出力リダイレクトはサポートされていません。 |
| PIPE_XML_NOT_ALLOWED_IN_INPUT | 501 | このコマンドへのパイプ XML は入力では許可されていません。 |
| PIPE_NOT_ALLOWED_IN_INPUT | 501 | この入力タイプにはパイプを使用できません。 |
| RESP_BIG_USE_CHUNK_ERR | 501 | 応答が許容最大値を超えています。最大は 10 MB です。チャンクを有効にして XML または JSON 出力を使用します。 |
| STRUCT_NOT_SUPPORTED_ERR | 501 | 構造化出力はサポートされていません。 |

| | | |
|---|---|---|
| ERR_UNDEFINED | 600 | 不明なエラー。 |

# JSON および XML 構造化出力

NX-OS は、次の構造化された出力フォーマットで、さまざまな **show** コマンドの標準規格出力のリダイレクトをサポートしています。

- XML
- JSON. JSON 出力の上限は 60 MB です。
- JSON フォーマット出力の標準規格ブロックを読みやすくした JSON Pretty もあります。JSON 出力の上限は 60 MB です。
- NX-OS リリース 9.3（1）で導入された JSON Native と JSON Pretty Native は、追加のコマンド解釈レイヤーをバイパスすることにより、JSON 出力をより高速かつ効率的に表示します。JSON Native および JSON Pretty Native は、出力のデータ型を保持します。出力用の文字列に変換する代わりに、整数を整数として表示します。

NX-OS CLI で、標準の NX-OS 出力を JSONまたは XML インタープリターに「パイプ接続」すると、これらのフォーマットへの変換が行われます。たとえば、**show ip access** コマンドを発行する際、論理パイプ（|）を続けて、その後に出力形式を指定できます。こうすると、NX-OS コマンドの出力が適切に構造化され、その形式でエンコードされます。この機能により、プログラムによるデータの解析が可能になり、ソフトウェア ストリーミング テレメトリを介したスイッチからのストリーミングデータがサポートされます。Cisco NX-OS のほとんどのコマンドは、JSON、JSON Pretty、JSON ネイティブ、JSON ネイティブ Pretty、および XML 出力をサポートしています。整合性チェッカー コマンドなど、一部のコマンドは、すべての形式をサポートしてはいません。整合性チェッカー コマンドは XML をサポートしていますが、JSON のバリアントはどれもサポートしていません。

（注） 検証エラーを回避するには、ファイル リダイレクトを使用して JSON 出力をファイルにリダイレクトし、そのファイル出力を使用します。

例:

```
Switch#show version | json > json_output ; run bash cat /bootflash/json_output
```

この機能の選択された例を以下に表示します。

## JSON の概要（JavaScript オブジェクト表記）

JSON は、判読可能なデータのために設計された軽量テキストベースのオープン スタンダードで、XML の代替になります。JSON はもともと JavaScript から設計されましたが、言語に依存しないデータ形式です。コマンド出力では、JSON および JSON プリティ形式、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-Disput
e"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
BLR-VXLAN-NPT-CR-179#
```

# XML および JSON 出力の例

このセクションでは、XML および JSON 出力として表示される NX-OS コマンドの例について説明します。

次の例は、ハードウェア テーブルのユニキャストおよびマルチキャスト ルーティング エントリを JSON 形式で表示する方法を示しています。

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096",
 "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "u
sed_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_tot
al": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "
used_mcast_oifl": "2", "used_host_in_host_total": "13", "used_host4_in_host": "1
2", "used_host6_in_host": "1", "max_ecmp_table_limit": "64", "used_ecmp_table":
"0", "mfib_fd_status": "Disabled", "mfib_fd_maxroute": "0", "mfib_fd_count": "0"
}
switch(config)#
```

次に、ハードウェア テーブルのユニキャストおよびマルチキャスト ルーティング エントリを XML 形式で表示する例を示します。

```
switch(config)# show hardware profile status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:fib">
 <nf:data>
  <show>
   <hardware>
    <profile>
     <status>
      <__XML__OPT_Cmd_dynamic_tcam_status>
       <__XML__OPT_Cmd_dynamic_tcam_status___readonly__>
        <__readonly__>
         <total_lpm>8191</total_lpm>
         <total_host>8192</total_host>
         <total_lpm>1024</total_lpm>
```

```
          <max_host4_limit>4096</max_host4_limit>
          <max_host6_limit>2048</max_host6_limit>
          <max_mcast_limit>2048</max_mcast_limit>
          <used_lpm_total>9</used_lpm_total>
          <used_v4_lpm>6</used_v4_lpm>
          <used_v6_lpm>3</used_v6_lpm>
          <used_v6_lpm_128>1</used_v6_lpm_128>
          <used_host_lpm_total>0</used_host_lpm_total>
          <used_host_v4_lpm>0</used_host_v4_lpm>
          <used_host_v6_lpm>0</used_host_v6_lpm>
          <used_mcast>0</used_mcast>
          <used_mcast_oifl>2</used_mcast_oifl>
          <used_host_in_host_total>13</used_host_in_host_total>
          <used_host4_in_host>12</used_host4_in_host>
          <used_host6_in_host>1</used_host6_in_host>
          <max_ecmp_table_limit>64</max_ecmp_table_limit>
          <used_ecmp_table>0</used_ecmp_table>
          <mfib_fd_status>Disabled</mfib_fd_status>
          <mfib_fd_maxroute>0</mfib_fd_maxroute>
          <mfib_fd_count>0</mfib_fd_count>
        </__readonly__>
      </__XML__OPT_Cmd_dynamic_tcam_status___readonly__>
     </__XML__OPT_Cmd_dynamic_tcam_status>
    </status>
   </profile>
  </hardware>
 </show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

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

```
switch(config)# show lldp timers | json
{"ttl": "120", "reinit": "2", "tx_interval": "30", "tx_delay": "2", "hold_mplier
": "4", "notification_interval": "5"}
switch(config)#
```

この例では、XML 形式でスイッチ上に LLDP タイマーを表示する方法を示します。

```
switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:lldp">
 <nf:data>
  <show>
   <lldp>
    <timers>
     <__XML__OPT_Cmd_lldp_show_timers___readonly__>
      <__readonly__>
       <ttl>120</ttl>
       <reinit>2</reinit>
       <tx_interval>30</tx_interval>
       <tx_delay>2</tx_delay>
       <hold_mplier>4</hold_mplier>
       <notification_interval>5</notification_interval>
      </__readonly__>
     </__XML__OPT_Cmd_lldp_show_timers___readonly__>
    </timers>
   </lldp>
  </show>
 </nf:data>
```

```
</nf:rpc-reply>
]]>]]>
switch(config)#
```

この例は、ACL 統計を XML 形式で表示する方法を示しています。

```
switch-1(config-acl)# show ip access-lists acl-test1 | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:1.0:aclmgr" xmlns:nf="urn:ietf:p
arams:xml:ns:netconf:base:1.0">
 <nf:data>
  <show>
   <__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
    <ip_ipv6_mac>ip</ip_ipv6_mac>
    <access-lists>
     <__XML__OPT_Cmd_show_acl_name>
      <name>acl-test1</name>
      <__XML__OPT_Cmd_show_acl_capture>
       <__XML__OPT_Cmd_show_acl_expanded>
        <__XML__OPT_Cmd_show_acl___readonly__>
         <__readonly__>
          <TABLE_ip_ipv6_mac>
           <ROW_ip_ipv6_mac>
            <op_ip_ipv6_mac>ip</op_ip_ipv6_mac>
            <show_summary>0</show_summary>
            <acl_name>acl-test1</acl_name>
            <statistics>enable</statistics>
            <frag_opt_permit_deny>permit-all</frag_opt_permit_deny>
            <TABLE_seqno>
             <ROW_seqno>
              <seqno>10</seqno>
              <permitdeny>permit</permitdeny>
              <ip>ip</ip>
              <src_ip_prefix>192.0.2.1/24</src_ip_prefix>
              <dest_any>any</dest_any>
             </ROW_seqno>
            </TABLE_seqno>
           </ROW_ip_ipv6_mac>
          </TABLE_ip_ipv6_mac>
         </__readonly__>
        </__XML__OPT_Cmd_show_acl___readonly__>
       </__XML__OPT_Cmd_show_acl_expanded>
      </__XML__OPT_Cmd_show_acl_capture>
     </__XML__OPT_Cmd_show_acl_name>
    </access-lists>
   </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
switch-1(config-acl)#
```

この例は、ACL 統計を JSON 形式で表示する方法を示しています。

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

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_t
os0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false"
,"admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S"
,"spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_
time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"m
ax_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cn
t":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_
nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa
":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"100","ins
tance_number":3,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":
"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive"
,"gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true"
,"is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_
time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0
S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_
aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"
asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"
area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal"
:0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_
rt_int":"false"},{"ptag":"111","instance_number":1,"cname":"default","rid":"0.0
.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_pe
riod":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only"
:"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_d
ist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max
_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT
5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric
_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"aso
paque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"
act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_d
iscard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_num
ber":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","g
r_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last
_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr"
:"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT
0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_h
```

```
old_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pac
e":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa
_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_norm
al":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_a
rea_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"
false"}]}}
switch-1#
```

次の例は、OSPF ルーティング パラメータを JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show ip ospf | json-pretty native
{
   "TABLE_ctx": {
          "ROW_ctx": [{
                         "ptag": "Blah",
                         "instance_number":      4,
                         "cname":        "default",
                         "rid":  "0.0.0.0",
                         "stateful_ha":  "true",
                         "gr_ha":        "true",
                         "gr_planned_only":      "true",
                         "gr_grace_period":      "PT60S",
                         "gr_state":     "inactive",
                         "gr_last_status":       "None",
                         "support_tos0_only":    "true",
                         "support_opaque_lsa":   "true",
                         "is_abr":       "false",
                         "is_asbr":      "false",
                         "admin_dist":   110,
                         "ref_bw":       40000,
                         "spf_start_time":       "PT0S",
                         "spf_hold_time":        "PT1S",
                         "spf_max_time": "PT5S",
                         "lsa_start_time":       "PT0S",
                         "lsa_hold_time":        "PT5S",
                         "lsa_max_time": "PT5S",
                         "min_lsa_arr_time":     "PT1S",
                         "lsa_aging_pace":       10,
                         "spf_max_paths":        8,
                         "max_metric_adver":     "false",
                         "asext_lsa_cnt":        0,
                         "asext_lsa_crc":        "0",
                         "asopaque_lsa_cnt":     0,
                         "asopaque_lsa_crc":     "0",
                         "area_total":   0,
                         "area_normal":  0,
                         "area_stub":    0,
                         "area_nssa":    0,
                         "act_area_total":       0,
                         "act_area_normal":      0,
                         "act_area_stub":        0,
                         "act_area_nssa":        0,
                         "no_discard_rt_ext":    "false",
                         "no_discard_rt_int":    "false"
                      }, {
                         "ptag": "100",
                         "instance_number":      3,
                         "cname":        "default",
                         "rid":  "0.0.0.0",
                         "stateful_ha":  "true",
                         "gr_ha":        "true",
                         "gr_planned_only":      "true",
                         "gr_grace_period":      "PT60S",
```

```
                        "gr_state":     "inactive",

                        ... content deleted for brevity ...

                        "max_metric_adver":     "false",
                        "asext_lsa_cnt":        0,
                        "asext_lsa_crc":        "0",
                        "asopaque_lsa_cnt":     0,
                        "asopaque_lsa_crc":     "0",
                        "area_total":   0,
                        "area_normal":  0,
                        "area_stub":    0,
                        "area_nssa":    0,
                        "act_area_total":       0,
                        "act_area_normal":      0,
                        "act_area_stub":        0,
                        "act_area_nssa":        0,
                        "no_discard_rt_ext":    "false",
                        "no_discard_rt_int":    "false"
                }]
        }
}
switch-1#
```

次に、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,"p
aths":3,"TABLE_unicast":{"ROW_unicast":[{"clientnameuni":"broadcast","best-paths":3}]},"TABLE_route_count":{"ROW_route_count":[{"mask_len":8,
"count":1},{"mask_len":32,"count":2}]}}]}}]}}]}}
switch-1(config)#
```

JSON ネイティブ（および JSON プリティネイティブ）では、整数が真の整数として表されることに注意してください。たとえば、「mask len:」は実際の値 32 として表示されます。

次の例は、JSON プリティ ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json-pretty native
{
  "TABLE_vrf": {
    "ROW_vrf": [{
        "vrf-name-out": "default",
        "TABLE_addrf": {
               "ROW_addrf": [{
                    "addrf": "ipv4",
                    "TABLE_summary": {
                           "ROW_summary": [{
                                "routes": 3,
                                "paths": 3,
                              "TABLE_unicast": {
                                    "ROW_unicast": [{
                                          "clientnameuni": "broadcast",
                                          "best-paths": 3
                                                    }]
                                              },
                                               "TABLE_route_count": {
                                                     "ROW_route_count":[{
                                                           "mask_len": 8,
                                                           "count": 1
                                                                 }, {
                                                           "mask_len": 32,
                                                           "count": 2
                                                      }]
                                                 }
                                        }]
                              }
                       }]
                }
           }]
     }
}
switch-1(config)#
```

# サンプル NX-API スクリプト

ユーザーはNX-APIでスクリプトを使用する方法を示すサンプルスクリプトにアクセスできます。サンプルスクリプトにアクセスするには、次のリンクをクリックして、必要なソフトウェア リリースに対応するディレクトリを選択します：Cisco Nexus 9000 NX-OS NX-API

第 22 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 （287 ページ）
- REST による DME 設定の置換 （288 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 （3 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 でサポートされます。
- ツリーを確認し、構成置換が適用される場所を確認することが重要です。構成置換操作にサンドボックスを使用する場合、サンドボックスはデフォルトでサブツリーに設定されるため、構成ツリー内の正しいノードをターゲットとするように URI を変更する必要があります。

- NX-OS サンドボックスを使用して（置換のための）変換を行う場合、要求に `status:'replaced'` 属性が存在するため、POST 操作を使用する必要があります。他の変換オプションを使用する場合は、PUT 操作を使用できます。
- サブツリーノードでこの機能の REST PUT オプションを使用すると、構成置換操作がサブツリー全体に適用されます。ターゲットのサブツリー ノードは PUT の構成置換データで正しく変更されますが、サブツリー ノードのリーフ ノードもデフォルト値に構成されることによって影響を受けることに注意してください。

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

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

# REST POST によるプロパティレベルの構成置換

シスコの DME モデルは、REST POST 操作による CLI ベースの機能のプロパティレベルの構成置換をサポートしています。要求ペイロードを生成し、REST POST 操作を介してスイッチに送信することにより、NX-OS サンドボックスを介して機能のプロパティの構成を置き換えることができます。NX-OS サンドボックスの詳細については、NX-API 開発者サンドボックスを参照してください。

**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 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 PUT の構成置換のトラブルシューティング

以下は、REST Put 操作による設定の置換が成功しない場合のトラブルシューティングに役立つ手順です。

**手順**

**ステップ 1** 要求が有効かどうかを確認します。

URL、操作、およびペイロードが有効である必要があります。たとえば、URL が api/mo/sys/foo.json の場合、ペイロードは foo で始まる必要があります。

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

- 正常に設定されました
- 有効なデバイス設定から取得

設定プロパティのみを取得するには、rsp-subtree=full&rsp-prop-include=set-config-only をフィルタリングする GET を使用します。

**ステップ 3** ペイロードを検証するには、DME POST 操作を使用してペイロードをスイッチに送信します。

**ステップ 4** エラーをチェックして、MO の名前とプロパティがあることを確認します。

**ステップ 5** ペイロードに MO の名前とプロパティも含まれていることを確認します。

第 23 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 （293 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 （307 ページ）

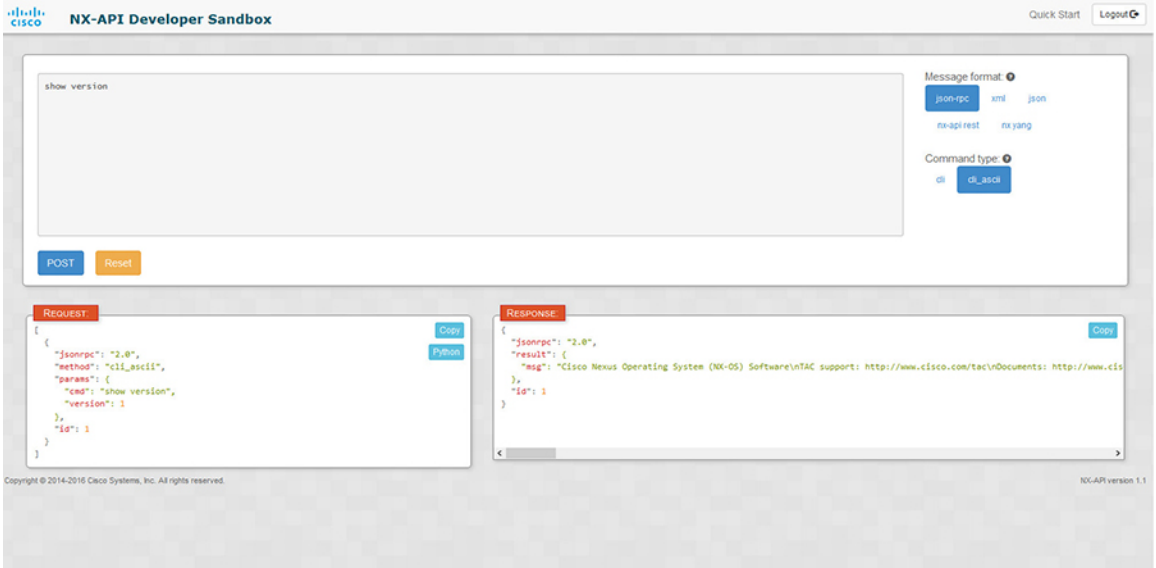

# NX-API 開発者サンドボックス:9.2（2）より前の NX-OS リリース

## 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。

# 注意事項と制約事項

デベロッパー サンドボックスのガイドラインと制限は次のとおりです：

- サンドボックスで **[送信（Send）]** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。
- 一部の機能構成コマンドは、関連する機能が有効になるまで使用できません。 たとえば、BGP ルータを構成するには、最初に **[機能 bgp（feature bgp）]** コマンドを使用して BGP を有効にする必要があります。同様に、OSPF ルータを構成するには、最初に **[機能 ospf（feature ospf）]** コマンドを使用して OSPF を有効にする必要があります。これは、**[evpn マルチホーミング コア トラッキング（evpn multihoming core-tracking）]** などの依存コマンドを有効にする **[evpn esi マルチホーミング（evpn esi multihoming）]** にも適用されます。機能が機能依存コマンドにアクセスできるようにする方法の詳細については、Cisco Nexus 9000 Configuration Guides Cisco Nexus 3000 Configuration Guides を参照してください。

- サンドボックスを使用した DN での変換は、CLI 構成の DN を検索する場合にのみサポートされます。DME を使用して CLI 構成コマンドの DN を変換するなど、他のワークフローはサポートされていません。
- OSPFv2 インターフェイス コマンドのための、CLI からモデルまたは xml への変換は、**[no] ip router ospf <tag> area {<area-id-ip> | <area-id-int>} [secondaries none]** コマンドを使用してルータ インスタンスとエリアを構成し、インターフェイスで OSPF を明示的に有効にするまでは行われません。
- コマンド ペイン（上部のペイン）は、最大 10,000 行の入力をサポートします。
- CLI 入力のメッセージ タイプとして XML または JSON を使用する場合、セミコロンを使用して同じ行の複数のコマンドを区切ることができます。ただし、CLI 入力のメッセージ タイプとして JSON RPC を使用する場合、同じ行に複数のコマンドを入力し、セミコロン ( ; ) で区切ることはできません。

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

  Sandbox で、次のように CLI を入力します。

  ```
  show hostname ; show clock
  ```

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

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

  リクエストを送信すると、レスポンスで次のエラーが返されます。

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

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

  同じ例を続けて、NX-API CLI で次のようにコマンドを入力します。

```
show hostname
show clock
```

リクエストでは、入力は次のようにフォーマットされます。

```
[
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show hostname",
      "version": 1
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 2
  }
]
```

応答は正常に完了します。

```
[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]
```

# メッセージ フォーマットとコマンド 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 *13 : 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 *14 :* 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<br>• cli_ascii — show または構成コマンド、フォーマットせずに出力 |

| メッセージ形式 | コマンド タイプ |
|---|---|
| xml | • cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。<br>• cli_show_ascii — コマンドを表示、フォーマットせずに出力<br>• cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。<br>• bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。<br>（注）<br>スイッチで bash シェルを有効にする必要があります。 |
| json | • cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。<br>• cli_show_ascii — コマンドを表示、フォーマットせずに出力<br>• cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。<br>• bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。<br>（注）<br>スイッチで bash シェルを有効にする必要があります。 |
| nx-api rest | • cli — 構成コマンド<br>• モデル — DN および対応するペイロード。 |
| nx yang | • json — ペイロードに JSON 構造が使用されます<br>• xml — XML 構造がペイロードに使用されます |

### 出力チャンク

大量の show コマンド出力を処理するために、一部の NX-API メッセージ フォーマットでは、show コマンドの出力チャンクがサポートされています。この場合、**[チャンク モードを有効にする（Enable chunk mode）]**チェックボックスが、セッション ID (**SID** ) 入力ボックスとともに **[コマンド タイプ（Command Type）]** コントロールの下に表示されます。

チャンクが有効な場合、応答は複数の「チャンク」で送信され、最初のチャンクが即時のコマンド応答で送信されます。応答メッセージの次のチャンクを取得するには、前の応答メッセージのセッション ID に設定された **SID** を使用して NX-API 要求を送信する必要があります。

# デベロッパー サンドボックスを使用

## デベロッパー サンドボックスを使用して CLI コマンドを REST ペイロードに変換する

ヒント　オンライン ヘルプは、サンドボックス ウィンドウの右上隅にある **[クイック スタート（Quick Start）]** をクリックすると利用できます。

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。

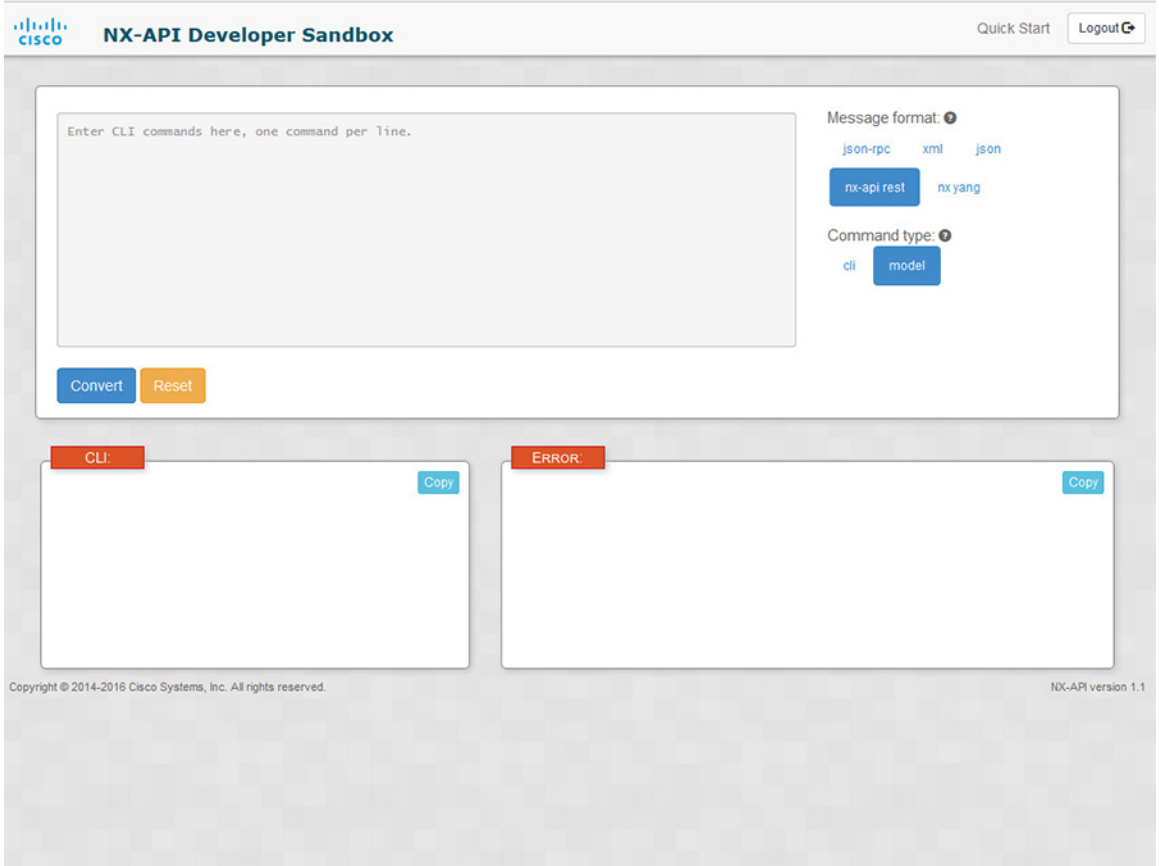
手順

**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（296 ページ）を参照してください。

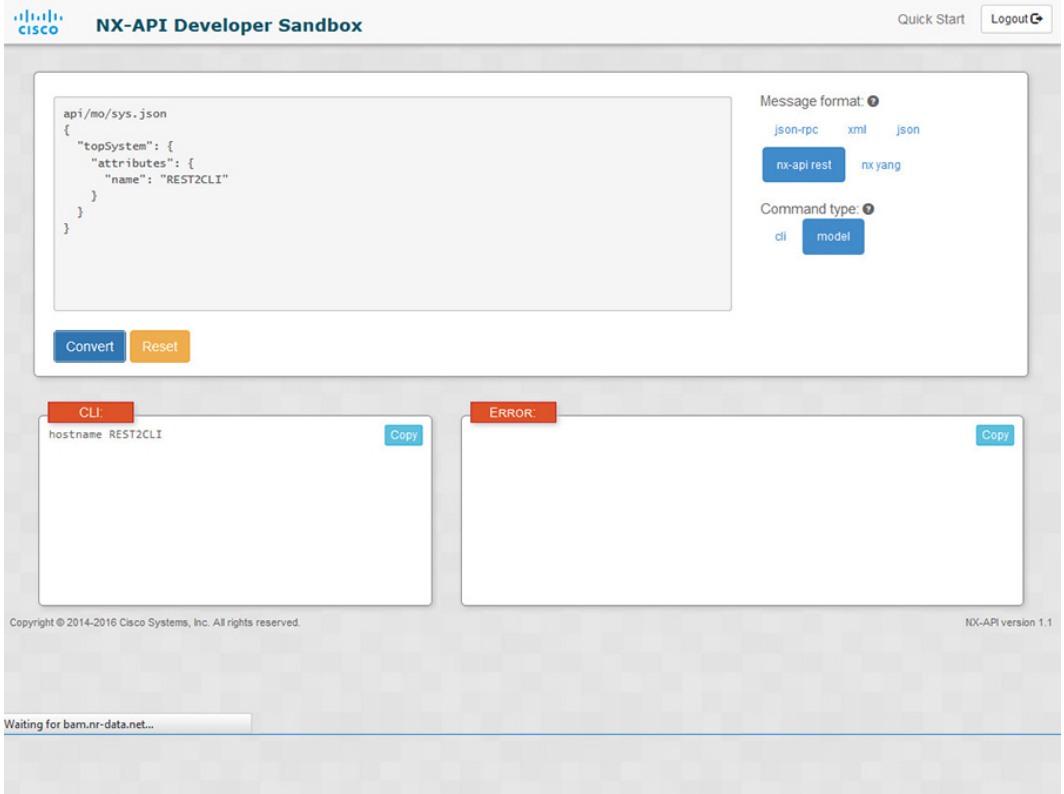
**ステップ 2**　上部ペインのテキスト エントリ ボックスに、NX-OS CLI 構成コマンドを 1 行に 1 つずつ入力するか貼り付けます。

上部ペインの下部にある **[リセット（Reset）]** をクリックすると、テキスト エントリ ボックス (および **[要求（Request）]** ペインと **[応答（Response）]** ペイン) の内容を消去できます。




**ステップ 3** トップ ペインの最下部にある **[変換（Convert）]** をクリックします。

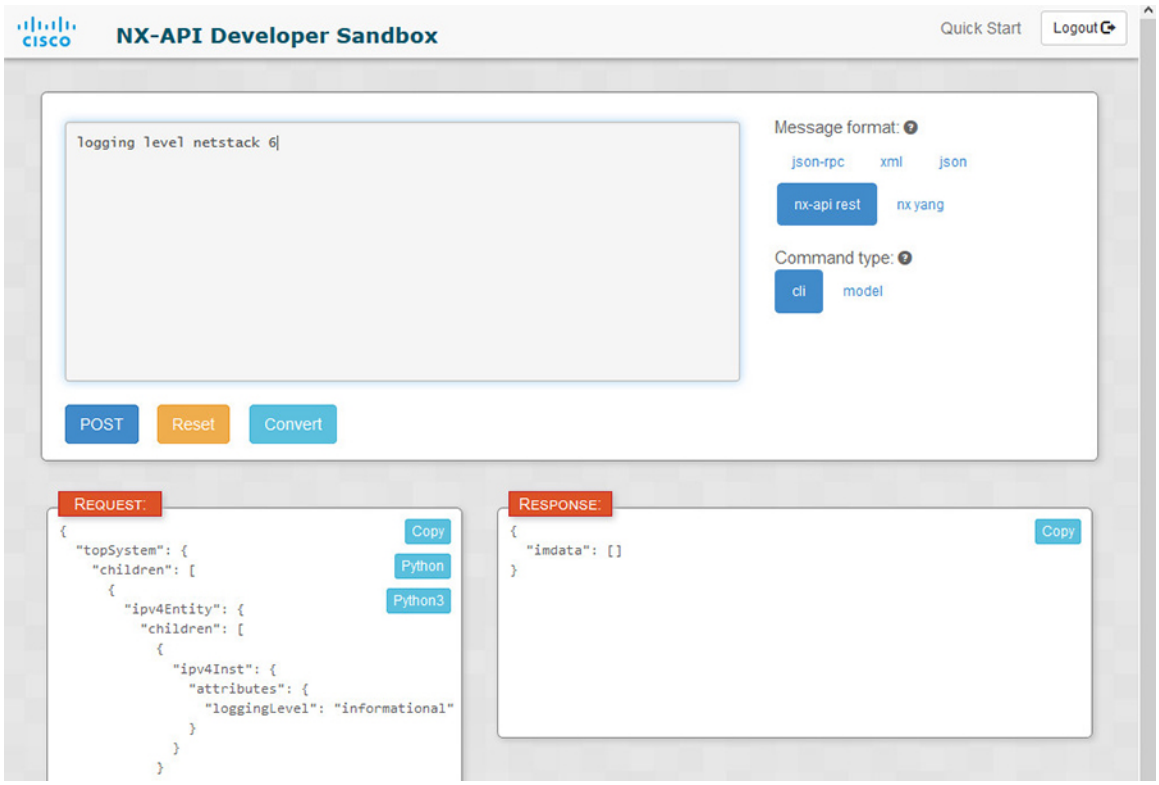
CLI コマンドに構成エラーが含まれていない場合、ペイロードは **[要求（Request）]** ペインに表示されます。エラーが存在する場合は、説明のエラー メッセージが **[応答（Response）]** ペインに表示されます。




**ステップ 4** **[リクエスト（Request）]** ペインに有効なペイロードが表示されている場合は、**POST** をクリックして、ペイロードを API 呼び出しとしてスイッチに送信できます。

スイッチからのレスポンスは **[Response（応答）]** ペインに表示されます。

警告
**POST** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。




**ステップ 5** ペインで **[コピー（Copy）]** をクリックすると、**[要求（Request）]** ペインまたは **[応答（Response）]** ペインの格納ファイルをクリップボードにコピーできます。

**ステップ 6** **[リクエスト（Request）]** ペインで **Python** をクリックすると、クリップボード上のリクエストの Python 導入を取得できます。

## 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。

### 手順の概要

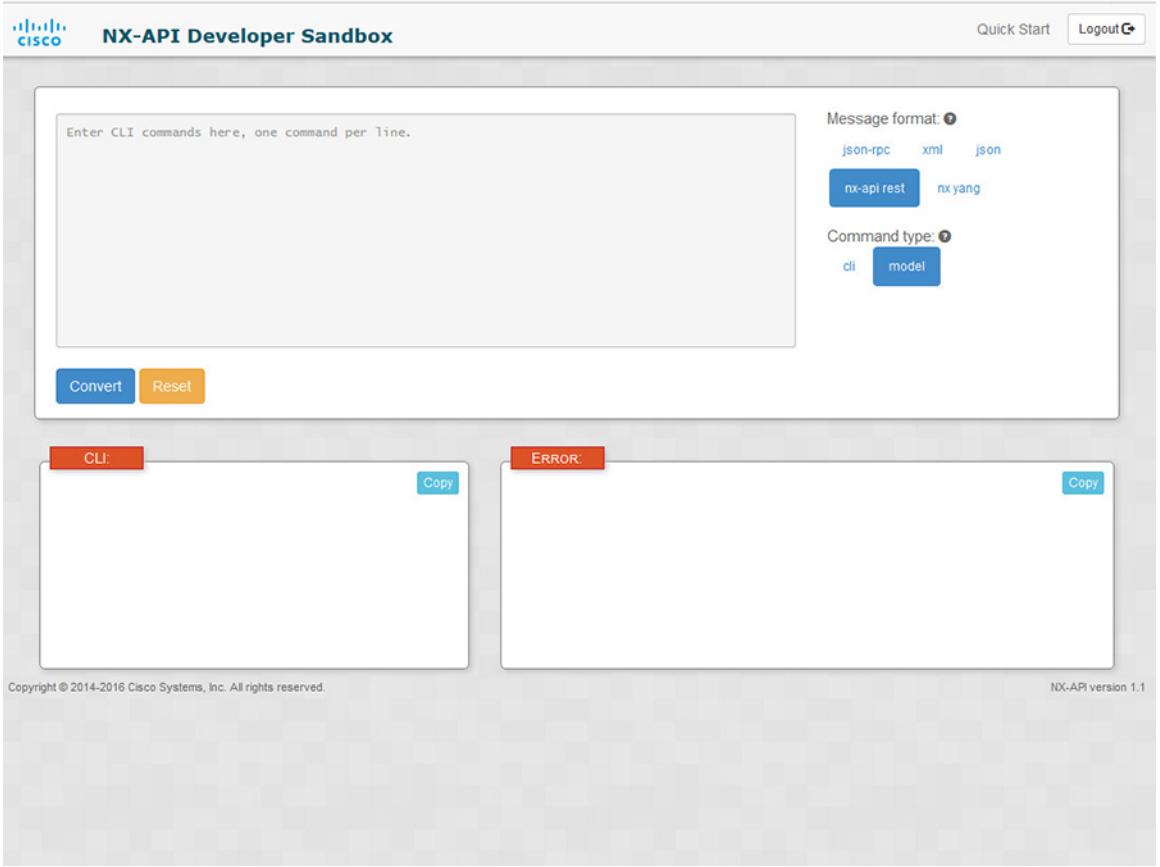
1. **メッセージ フォーマット**として **`nx-api rest`** を選択し、**コマンド タイプ**として **`model`** を選択します。
2. 上部ペインのテキスト入力ボックスに DN とペイロードを入力します。次に、上部ペインの下にある **[変換（Convert）]** ボタンをクリックします。

## 手順の詳細

### 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：**

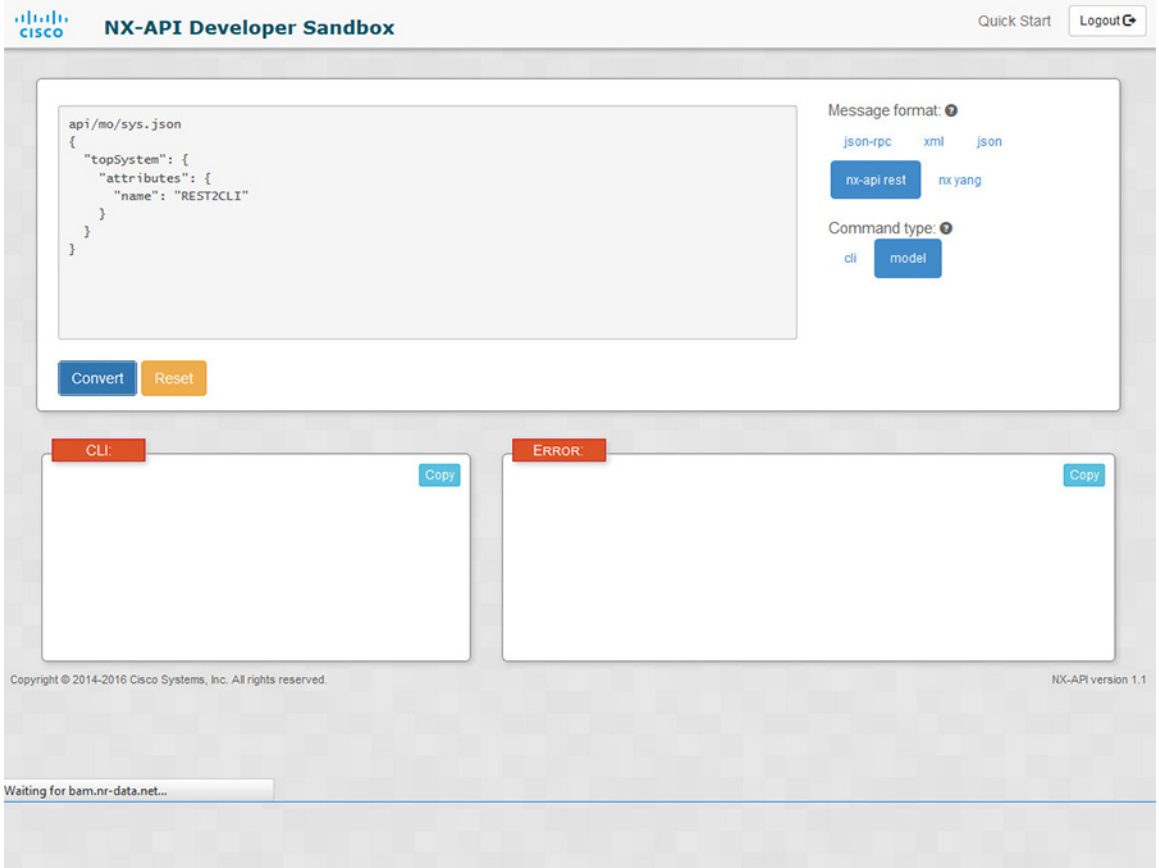



**ステップ 2** 上部ペインのテキスト入力ボックスにDNとペイロードを入力します。次に、上部ペインの下にある **[変換（Convert）]** ボタンをクリックします。

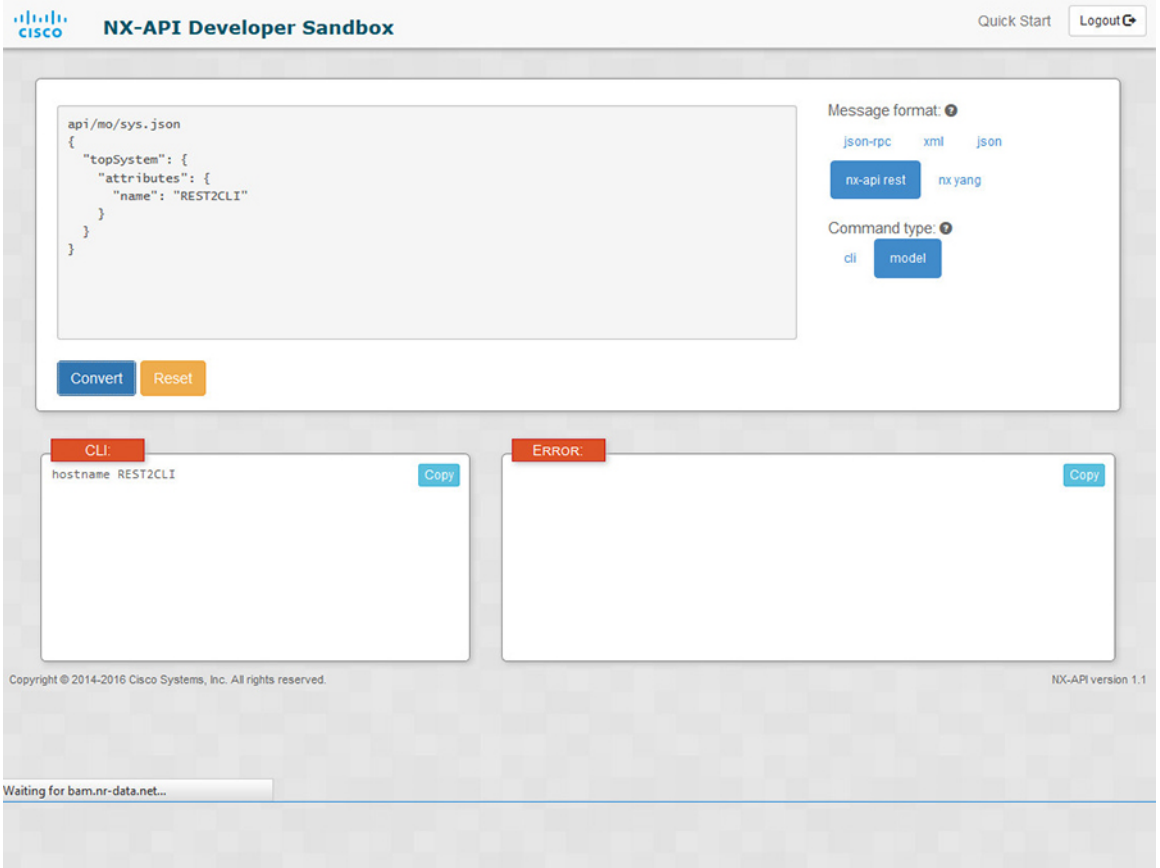
**例：**

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 *15 : REST2CLI* エラーの原因

| ペイロードの問題 | 結果 |
|---|---|
| ペイロードに、MO に存在しない属性が含まれています。<br>例：<br>`api/mo/sys.json`<br>`{ "topSystem": { "children": [ { "interfaceEntity": { "children": [ { "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": "totallyFake" } } } ] } } ] } }` | **[エラー（Error）]**ペインは、属性に関連するエラーを返します。<br>例：<br>**CLI**<br>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。<br>例：<br>`api/mo/sys.json`<br>`{ "topSystem": { "children": [ { "dhcpEntity": { "children": [ { "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" } } } ] } } ] } }` | **[エラー（Error）]**ペインは、サポートされていない MO に関連するエラーを返します。<br>例：<br>**CLI**<br>**[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

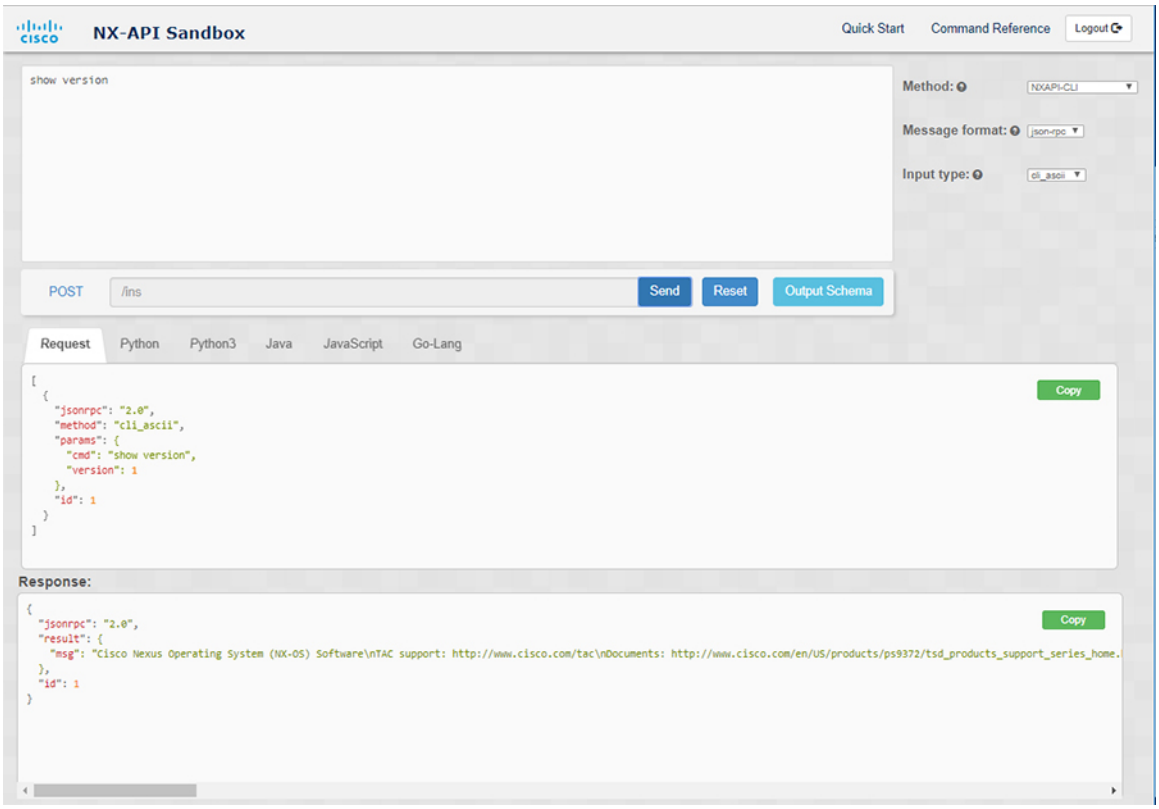
## About the NX-API デベロッパー サンドボックス

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**



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 （315 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 （318 ページ）」を参照してください。

# 注意事項と制約事項

デベロッパー サンドボックスのガイドラインと制限は次のとおりです：

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

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

Sandbox で、次のように CLI を入力します。

```
show hostname ; show clock
```

JSON RPC リクエストでは、入力は次のようにフォーマットされます。

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

リクエストを送信すると、レスポンスで次のエラーが返されます。

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

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

同じ例を続けて、NX-API CLI で次のようにコマンドを入力します。

```
show hostname
show clock
```

リクエストでは、入力は次のようにフォーマットされます。

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

```
    "id": 2
  }
]
```

応答は正常に完了します。

```
[
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "hostname": "switch-1"
      }
    },
    "id": 1
  },
  {
    "jsonrpc": "2.0",
    "result": {
      "body": {
        "simple_time": "12:31:02.686 UTC Wed Jul 10 2019\n",
        "time_source": "NTP"
      }
    },
    "id": 2
  }
]
```

# メッセージ フォーマットと入力タイプの構成

**メソッド**、**メッセージ形式**、および **入力タイプ** は、コマンド ペイン（上部のペイン）の右上隅で構成されます。**[メソッド]** で、使用する API プロトコルの形式を選択します。Cisco NX-API Developer Sandbox は、次の API プロトコルをサポートしています。

表 ***16 : NX-OS API*** プロトコル

| プロトコル | 説明 |
|---|---|
| NXAPI-CLI | XML または JSON ペイロードで NX-OS CLI または bash コマンドを配信するための Cisco NX-API 独自のプロトコル。 |

| プロトコル | 説明 |
|---|---|
| NXAPI-REST（DME） | 内部 NX-OS データ管理エンジン（DME）モデルで管理対象オブジェクト（MO）とそのプロパティを操作および読み取るための Cisco NX-API 独自のプロトコル。NXAPI-REST（DME）プロトコルは、次の方法から選択できるドロップダウン リストを表示します。<br>• **POST**<br>• **GET**<br>• **PUT**<br>• **DELETE**<br><br>Cisco Nexus 3000 および 9000 シリーズ NX-API REST SDK の詳細については、https://developer.cisco.com/site/cisco-nexus-nx-api-references/ を参照してください。 |
| RESTCONF（Yang） | 構成および状態データ用の YANG（「Yet Another Next Generation」）データ モデリング言語。<br><br>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。<br>• **POST**<br>• **GET**<br>• **PUT**<br>• **PATCH**<br>• **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 *17*: 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<br>• cli_ascii — show または構成コマンド、フォーマットせずに出力<br>• cli-array — show コマンド。cli に似ていますが、cli_array を使用すると、データは 1 つの要素のリスト、または角括弧 [ ] で囲まれたアレイとして返されます。 |
| NXAPI-CLI | xml | • cli_show — コマンドを表示します。コマンドが XML 出力をサポートしていない場合、エラー メッセージが返されます。<br>• cli_show_ascii — コマンドを表示、フォーマットせずに出力<br>• cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。<br>• bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。<br>（注）<br>スイッチで bash シェルを有効にする必要があります。 |

| 方法 | メッセージ形式 | 入力/コマンド タイプ |
|---|---|---|
| NXAPI-CLI | json | • cli_show — コマンドを表示します。コマンドがXML出力をサポートしていない場合、エラー メッセージが返されます。<br>（注）<br>Cisco NX-OS リリース 9.3(3) 以降では、cli_show コマンドよりも cli_show_array コマンドが推奨されます。<br>• cli_show_array — show コマンドcli_show に似ていますが、cli_show_array を使用すると、データは角括弧 [ ] で囲まれた 1 つの要素のリストまたは配列として返されます。<br>• cli_show_ascii — コマンドを表示、フォーマットせずに出力<br>• cli_conf — 構成 コマンド。対話型の構成コマンドはサポートされていません。<br>• bash — bash コマンド。ほとんどの非対話型 bash コマンドがサポートされています。<br>（注）<br>スイッチでbashシェルを有効にする必要があります。 |
| NXAPI-REST（DME） | | • cli — CLI からモデルへの変換<br>• model — モデルから CLI への変換。 |
| RESTCONF（Yang） | • json — ペイロードに JSON 構造が使用されます<br>• xml — XML 構造がペイロードに使用されます | |

### 出力チャンク

JSON および XML NX-API メッセージ形式を使用すると、10 MB のチャンクで大きな show コマンド応答を受信できます。受信すると、チャンクが連結されて、有効な JSON オブジェクトまたはXML 構造が作成されます。出力チャンクを示すサンプル スクリプトを表示するには、

次のリンクをクリックし、リリース 9.3x に対応するディレクトリを選択します：Cisco NX-OS NXAPI。

（注） チャンク JSON モードの場合、ブラウザーまたは Python スクリプト パーツは有効な JSON 出力を提供しません（終了タグはありません）。チャンク モードを使用して有効な JSON を取得するには、ディレクトリで提供されるスクリプトを使用します。

即時のコマンド応答で最初のチャンクを受け取ります。これには、セッション ID を含む **sid** フィールドも含まれます。次のチャンクを取得するには、前のチャンクのセッション ID を **[SID]** テキスト ボックスに入力します。**sid** フィールドの **eoc**（コンテンツの終わり）値で示される最後の応答に到達するまで、プロセスを繰り返します。

チャンク モードは、**JSON** または **XML** フォーマット タイプおよび **cli_show**、**cli_show_array**,、または **cli_show_ascii** コマンド タイプで **NXAPI-CLI** メソッドを使用する場合に使用できます。チャンク モードの設定の詳細については、チャンク モード フィールドの表を参照してください。

（注） NX-API は、最大 2 つのチャンク セッションをサポートします。

表 *18*：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。<br>（注）<br>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

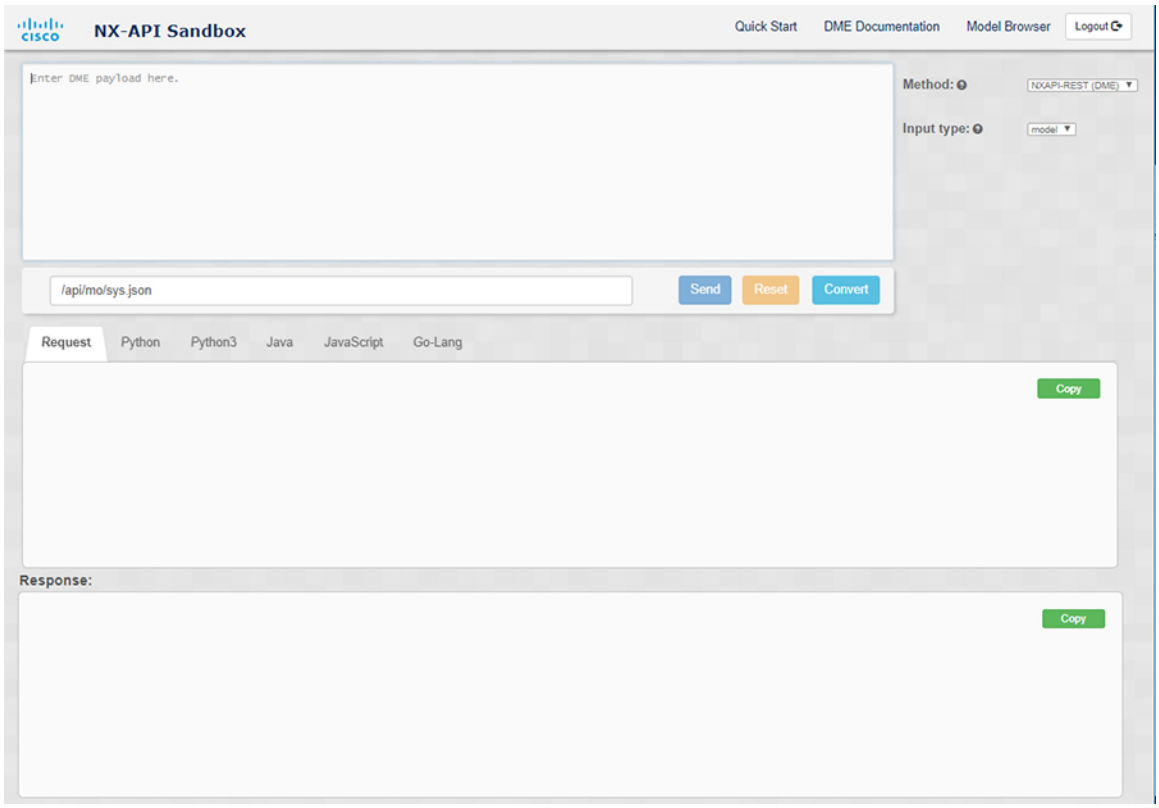
**ステップ 1** **[方法（Method）]** ドロップダウン リストをクリックし、**NXAPI-REST (DME)** を選択します。

**[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。



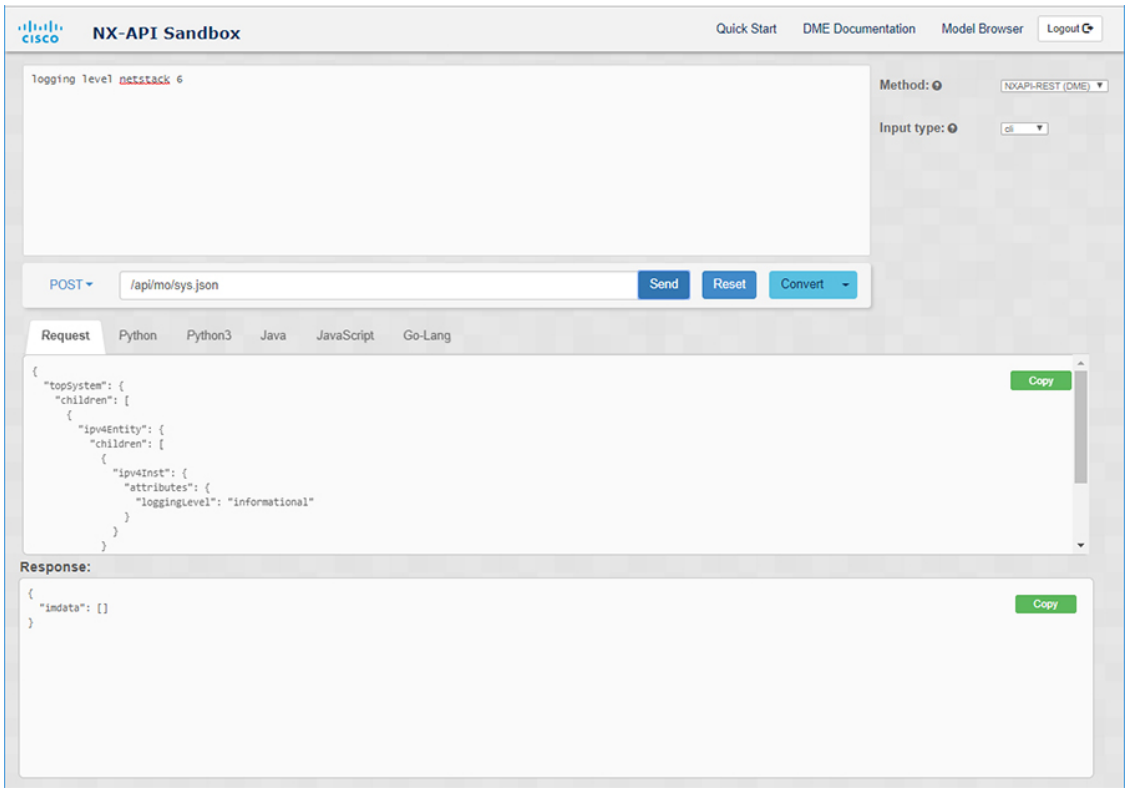

**ステップ 4** [変換（Convert）] をクリックします。

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

**ステップ 5** (オプション) 有効なペイロードを API 呼び出しとしてスイッチに送信するには、**[送信（Send）]** をクリックします。

スイッチからのレスポンスは **[Response（応答）]** ペインに表示されます。

警告
**[送信（Send）]** をクリックすると、コマンドがスイッチにコミットされ、構成または状態が変更される可能性があります。




**ステップ 6** (オプション) ペイロード内の MO の DN を取得するには：

1. **[リクエスト（Request）]** ペインから、**POST** を選択します。
2. **[変換（Convert）]** ドロップダウン リストをクリックし、**[変換 (DN を使用)（Convert (with DN)）]** を選択します。

ペイロードは、ペイロード内の各 MO に対応する DN を含む **dn** フィールドとともに表示されます。

**ステップ 7** (オプション) 新しい構成で現在の構成を上書きする場合：

1. **[変換（Convert）]** ドロップダウン リストをクリックし、**[変換（置換用）（Convert（for Replace））]** を選択します。**[リクエスト（Request）]** ペインには、**[ステータス（status）]** フィールドが**[置換（replace）]**ように設定されたペイロードが表示されます。
2. **[リクエスト（Request）]** ペインから、**POST** を選択します。
3. [送信（Send）] をクリックします。

現在の構成は、投稿された構成に置き換えられます。たとえば、次の構成で開始するとします：

```
interface eth1/2
  description test
  mtu 1501
```

次に、**[変換（置換用）（Convert（for Replace））]**を使用して、次の構成を POST します。

```
interface eth1/2
  description testForcr
```

`mtu` 構成が削除され、新しい説明 (`testForcr`) のみがインターフェイスの下に表示されます。この変更は、**show running-config** と入力すると確認されます。

**ステップ 8** (オプション) **[リクエスト（Request）]** ペインや **[応答（Response）]** ペインなどのペインの内容をコピーするには、**[コピー（Copy）]** をクリックします。それぞれのペインの内容がクリップボードにコピーされます。

**ステップ 9** (オプション) リクエストを以下のいずれかのフォーマットに変換するには、**[リクエスト（Request）]** ペインの適切なタブをクリックします。

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

## デベロッパー サンドボックスを使用した REST ペイロードから CLI コマンドへの変換

Cisco NX-API Developer Sandbox を使用すると、REST ペイロードを対応する CLI コマンドに変換できます。このオプションは、NXAPI-REST (DME) メソッドでのみ使用できます。

ヒント

- Cisco NX-API Developer Sandbox のフィールド名の横にあるヘルプ アイコン（**?**）をクリックすると、オンライン ヘルプを利用できます。ヘルプ アイコンをクリックして、それぞれのフィールドに関する情報を取得します。

  応答コードやセキュリティ メソッドなどの詳細については、*NX-API CLI* の章を参照してください。

- Cisco NX-API Developer Sandbox の右上隅には、追加情報へのリンクが含まれています。表示されるリンクは、選択した**[方法（Method）]**によって異なります。NXAPI-REST（DME）メソッドに表示されるリンク：

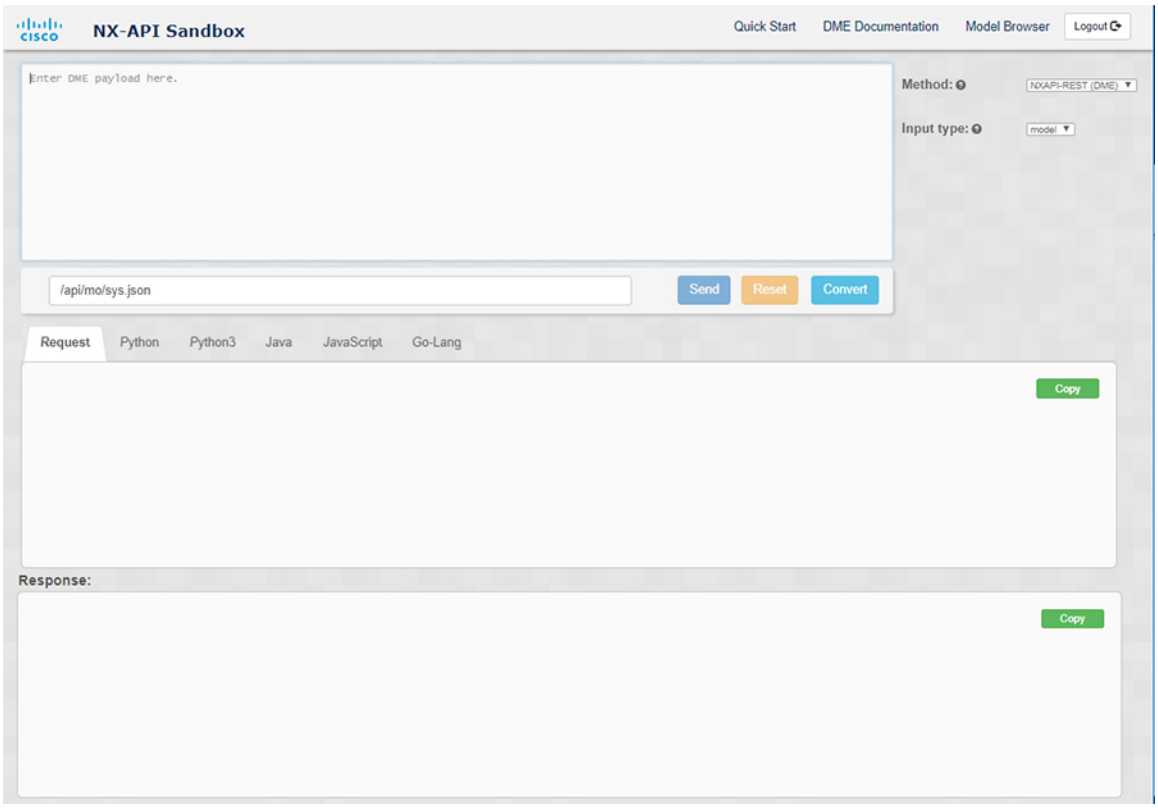
  - **[NX-API リファレンス（NX-API References）]**— 追加の NX-API ドキュメントにアクセスできます。
  - **[DME ドキュメント（DME Documentation）]** — NX-API DME モデル リファレンス ページにアクセスできます。
  - **[モデル ブラウザ（Model Browser）]** — モデル ブラウザである Visore にアクセスできます。Visore ページにアクセスするには、スイッチの IP アドレスを手動で入力する必要がある場合があることに注意してください。

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

手順

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

例：




**ステップ 2** **[タイプを入力（Input Type）]** タイプドロップダウン リストをクリックし、**[モデル（model）]** を選択します。

**ステップ 3** 要求ペインの上にあるフィールドに、ペイロードに対応する指定名（DN）を入力します。

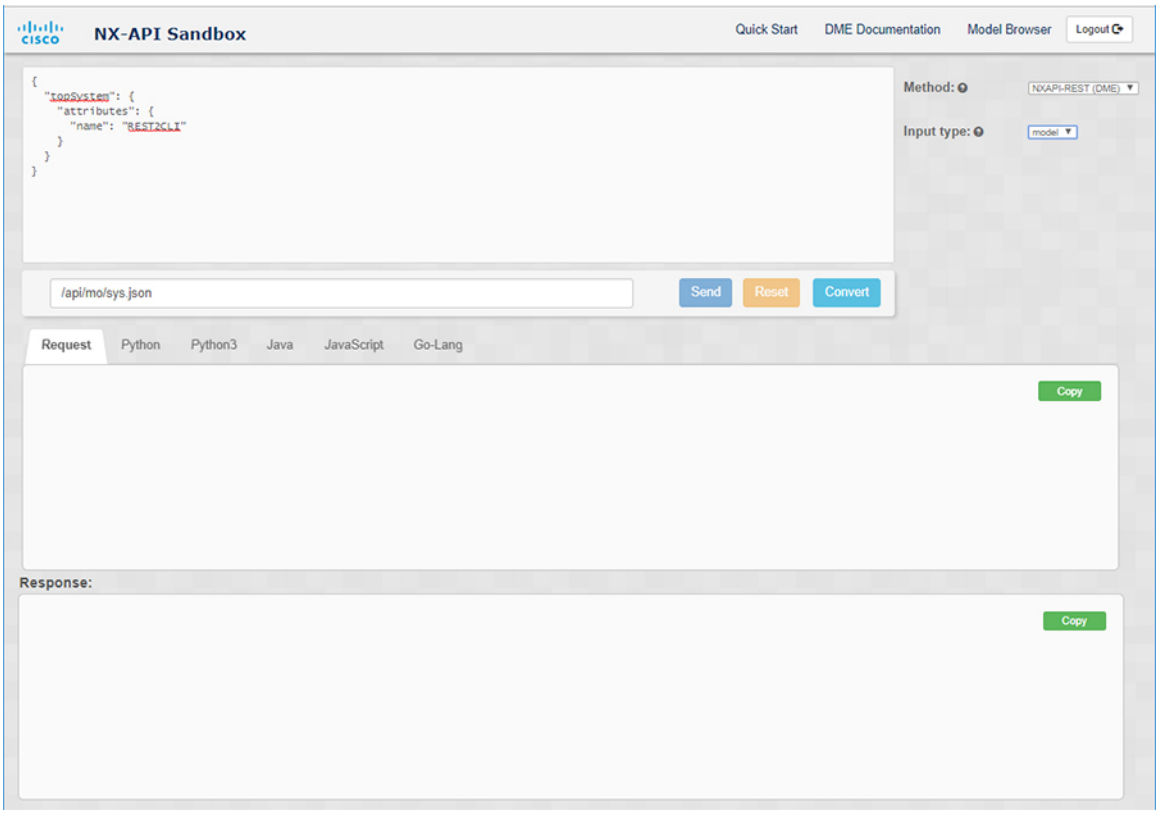
**ステップ 4** コマンド ペインにペイロードを入力します。

**ステップ 5** [変換（Convert）] をクリックします。

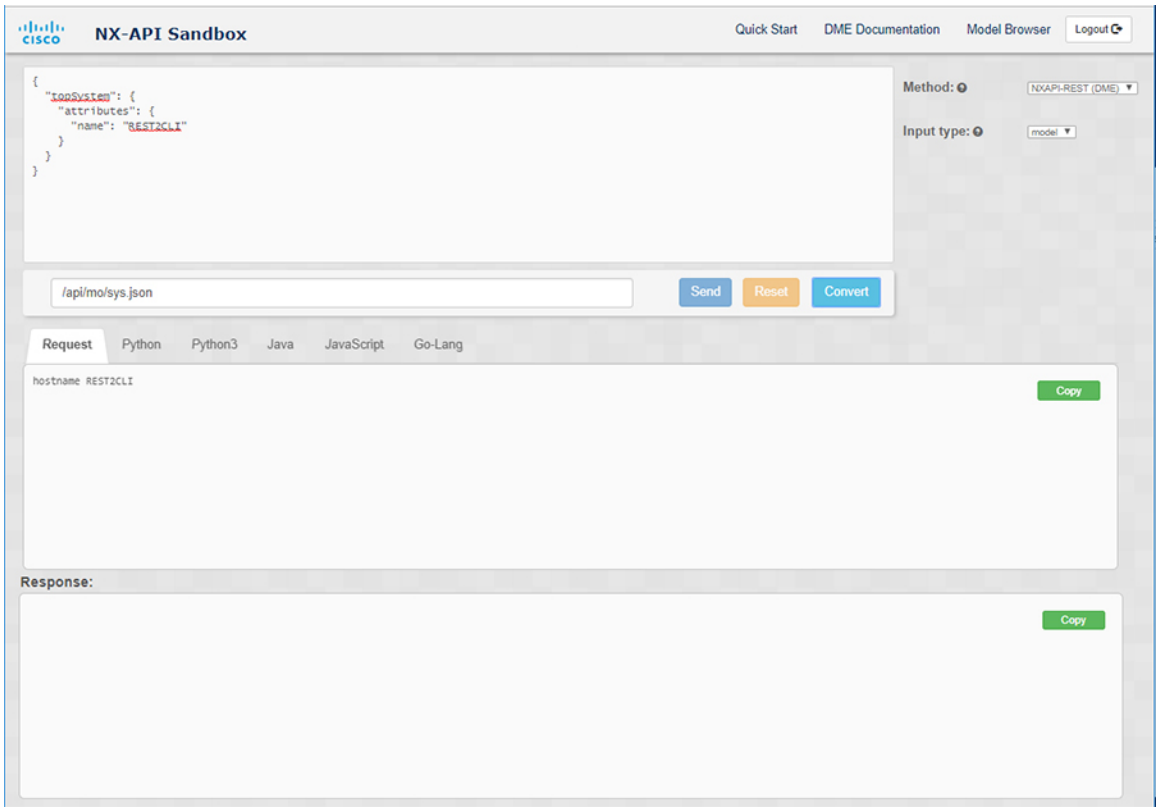
**例：**

この例では、DN は **/api/mo/sys.json** であり、NX-API REST ペイロードは次のとおりです。

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```




**[変換（Convert）]** ボタンをクリックすると、次の図に示すように、同等の CLI が **CLI** ペインに表示されます。




（注）
Cisco NX-API Developer Sandbox は、サンドボックスが CLI を NX-API REST ペイロードに変換した場合でも、すべてのペイロードを同等の CLI に変換できません。以下は、ペイロードが CLI コマンドに完全に変換するのを妨げる可能性のあるエラーの原因のリストです。

表 *19 : 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。<br>例：<br>`api/mo/sys.json`<br>`{ "topSystem": { "children": [ { "interfaceEntity": { "children": [ { "l1PhysIf": { "attributes": { "id": "eth1/1", "fakeattribute": "totallyFake" } } } ] } } ] } }` | **[エラー（Error）]**ペインは、属性に関連するエラーを返します。<br>例：<br>**CLI**<br>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。<br>例：<br>`api/mo/sys.json`<br>`{ "topSystem": { "children": [ { "dhcpEntity": { "children": [ { "dhcpInst": { "attributes": { "SnoopingEnabled": "yes" } } } ] } } ] } }` | **[エラー（Error）]**ペインは、サポートされていない MO に関連するエラーを返します。<br>例：<br>**CLI**<br>**[エラー（Error）]** [「`sys/dhcp`」のサブツリー全体が変換されていません。（`The entire subtree of "sys/dhcp" is not converted.`）] |

# デベロッパー サンドボックスを使用して RESTCONF から json または XML に変換する

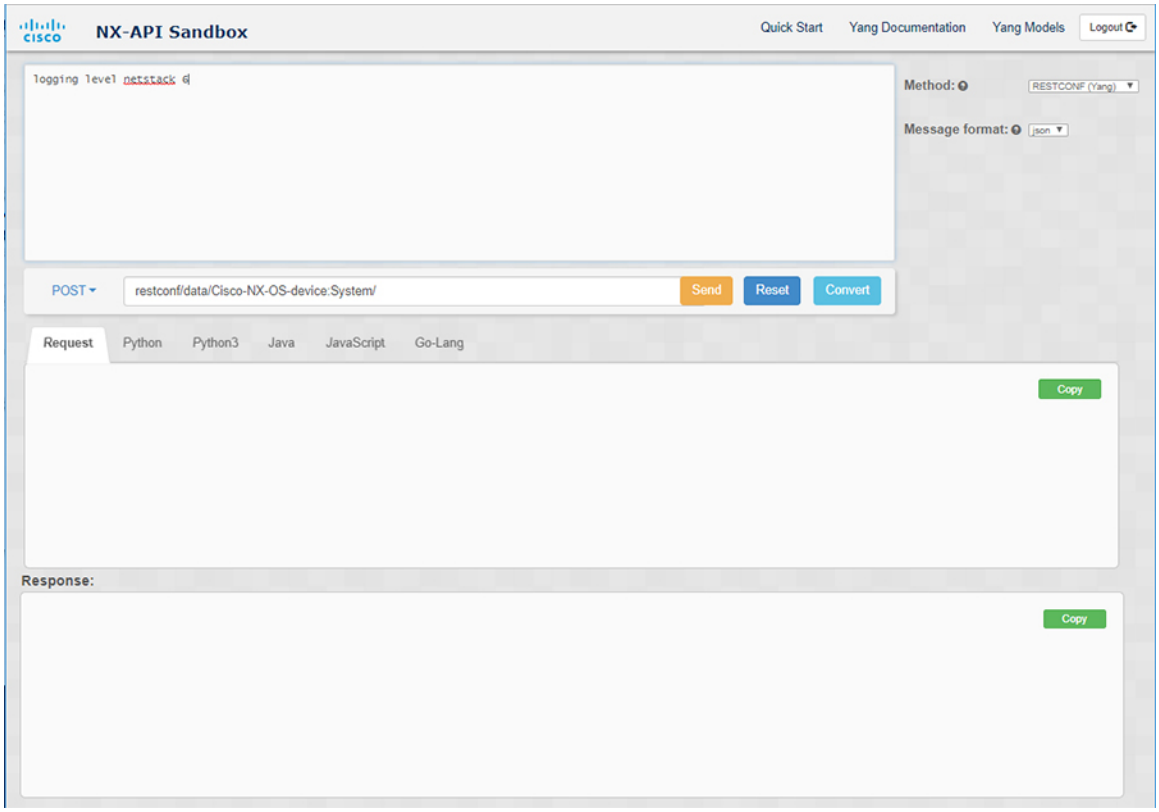
ヒント

- Cisco NX-API Developer Sandbox ウィンドウの右上隅にあるヘルプ アイコン（?）をクリックすると、オンライン ヘルプを利用できます。
- [サンドボックス] ウィンドウの右上隅にある **Yang Documentation** リンクをクリックして、Model Driven Programmability with Yang ページに移動します。
- [サンドボックス] ウィンドウの右上隅にある **Yang Models** リンクをクリックして、YangModels GitHub サイトにアクセスします。

手順

**ステップ 1** **[メソッド]** ドロップダウン リストをクリックし、**[RESTCONF (Yang)]** を選択します。

例：




**ステップ 2** **[メッセージ形式]** をクリックし、**json** または **xml** を選択します。

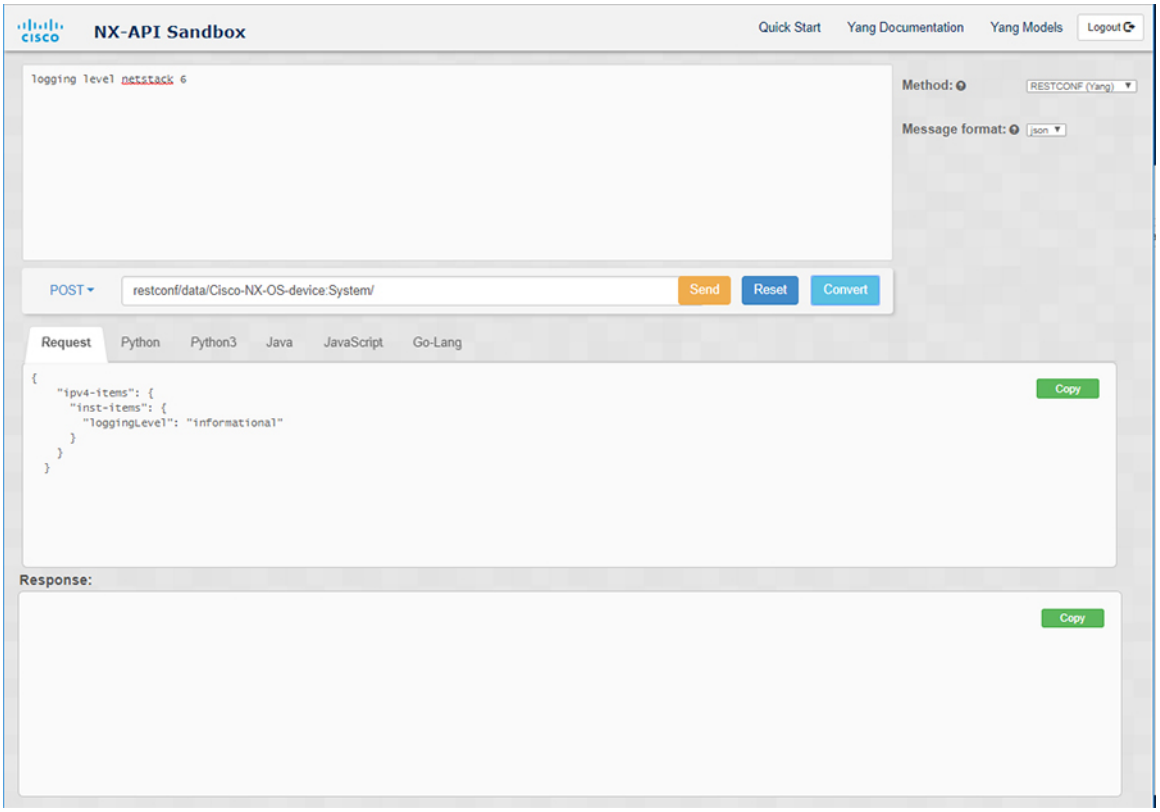
**ステップ 3** 上部ペインのテキスト入力ボックスにコマンドを入力します。

**ステップ 4** メッセージ形式を選択します。

**ステップ 5** [変換（Convert）] をクリックします。

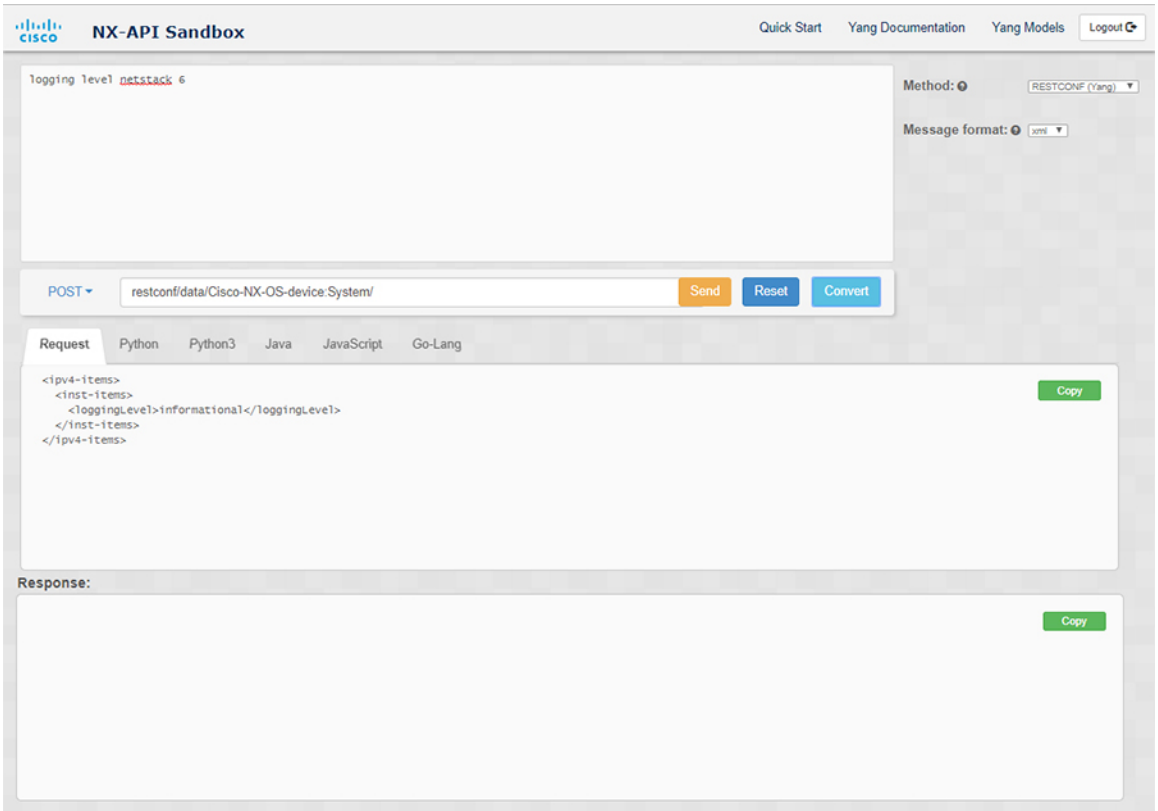
**例：**

この例では、コマンドは**ログ レベル netstack 6** で、メッセージ形式は json です。




**例：**

この例では、コマンドは**ログ レベル netstack 6** で、メッセージ形式は xml です。




（注）
XML または JSON メッセージ形式を使用して、否定された CLI を Yang ペイロードに変換すると、サンドボックスは警告をスローし、**[送信]** オプションを無効にします。表示される警告メッセージは、メッセージの形式によって異なります。

- XML メッセージ形式の場合 — 「これは Netconf ペイロードであり、DELETE 操作用に生成されているため、Restconf では SEND オプションが無効になっています!」
- JSON メッセージ形式の場合 - 「これは、DELETE 操作用に生成される gRPC ペイロードであるため、Restconf では SEND オプションが無効になっています!」

**ステップ 6** **[リクエスト]** ペインの適切なタブをクリックして、リクエストを次の形式に変換することもできます。

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

（注）

[リクエスト] タブの上の領域にあるドロップダウン メニューから [PATCH] オプションを選択した場合、Java で生成されたスクリプトは機能しません。これは Java の既知の制限であり、予期される動作です。

# 第 V 部

# モデル駆動型プログラマビリティ


- 概要 （331 ページ）
- 元の YANG と OpenConfig YANG （335 ページ）
- NETCONF エージェント （369 ページ）
- RESTCONF エージェント （413 ページ）
- gRPC エージェント （425 ページ）
- gNMI：管理インターフェイス （437 ページ）
- gNOI：操作インターフェイス （467 ページ）
- gRPC トンネル （503 ページ）
- テレメトリ （515 ページ）
- 診断と有用性 （605 ページ）

第 24 章

# 概要


- モデル駆動型プログラマビリティについて （331 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 （332 ページ）


## モデル駆動型プログラマビリティについて

Cisco NX-OS デバイスのモデル駆動型プログラマビリティ（model-driven programmability、MDP）により、デバイスの構成と制御を自動化できます。また、デバイスの状態変化をモニターすることもできます。

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。

次に、NX-OS でサポートされる標準ベースのプログラマブルインターフェイスを示します。

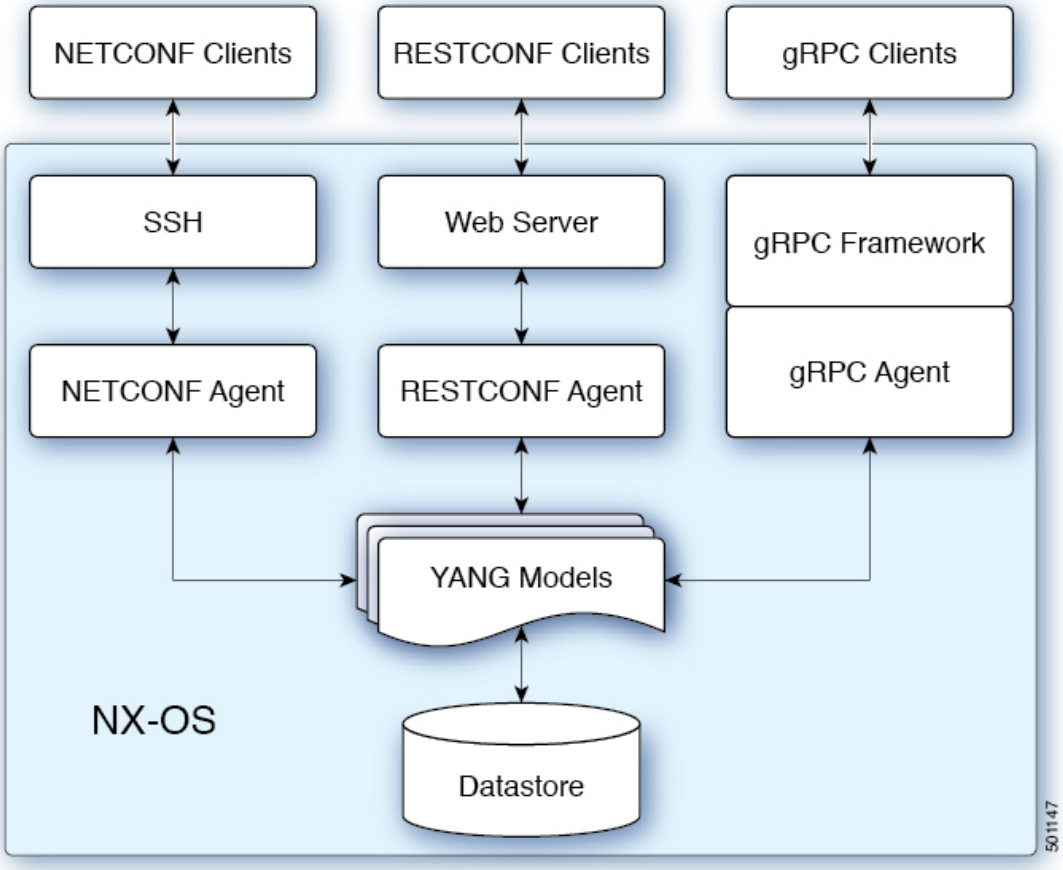
- NETCONF
- RESTCONF
- gRPC

NX-OS でサポートされるオブザーバビリティインターフェイスは次のとおりです。

- NETCONF
- gRPC
- テレメトリ

# 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 のどの方法で受信した場合でも、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、要求内のモデル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て、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、基礎となる NX-OS のデバイス 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 プログラマビリティエージェント

エージェントは、NX-OS デバイスと外部クライアント間のインターフェイスを提供します。ネットワーキングデバイスとの通信の特定のトランスポート、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 モデルを介する構成管理に異なるインターフェイスを提供します。

**表 *20 : 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 | トランスポート | プロトコル（Protocol） | エンコーディング（Encoding） |
|---|---|---|---|
| NETCONF | SSH | 該当なし | XML |
| RESTCONF | HTTP | draft-ietf-netconf-restconf-10[1] | XML または JSON |
| gRPC | HTTP | gRPC プロトコル [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。

- RESTCONF エージェント （413 ページ）
- gRPC エージェント （425 ページ）

# NX-OS オブザーバビリティエージェント

スイッチイベントをモニターするための、読み取り専用ストリーミングインターフェイスが存在することに注意してください。NX-OS は、標準ベースの NETCONF と gRPC、および独自のソフトウェアテレメトリをサポートします。

**表 *21 : NX-OS* オブザーバビリティインターフェイスエージェント**

| エージェント | トランスポート | プロトコル（Protocol） |
|---|---|---|
| NETCONF | SSH | 該当なし |
| gRPC | HTTP | gRPC プロトコル |
| テレメトリ | HTTP、UDP、GRPC | NA |

# 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。

# ネイティブ（デバイス）YANG モデル

デバイス構成を記述している YANG モデルは、デバイス YANG モデルと呼ばれます。デバイスモデルは、デバイス名前空間の他のモデルコンポーネントのように、モデルインフラストラクチャに明示されます。

# 共通（OpenConfig）YANG モデル

共通モデルは、パブリックコミュニティによって定義される別の種類のモデルコンポーネントです。共通モデルの YANG パスは、同等のデバイスモデル要素に変換できます。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。

（注） Cisco NX-OSは、中央集中型の DME データベースを維持します。デバイスと OpenConfig YANG は、DME データベースのラッピング表現です。これにより、すべてのプログラマビリティインターフェイスで一貫性が保証されます。ユーザーが異なるインターフェイスを使用していても、同じスイッチの構成/状態が矛盾なく表示されます。

# その他の YANG 関連資料

YANG の詳細については、https://handwiki.org/wiki/YANGを参照してください。

第 25 章

# 元の YANG と OpenConfig YANG


- YANG について （335 ページ）
- Cisco デバイス YANG （336 ページ）
- 注意事項と制約事項 （336 ページ）
- DME からデバイス YANG への移行 （337 ページ）
- OpenConfig YANG について （338 ページ）
- YANG バージョンのアップグレードまたはダウングレード （364 ページ）
- YANG 向けの RBAC （365 ページ）
- 注意事項と制約事項 （365 ページ）
- YANG RBAC の構成 （366 ページ）
- YANG のトラブルシューティング （367 ページ）


## YANG について

Yet Another Next Generation（YANG）は、ネットワーク管理プロトコルを介して転送されるデータを定義するためのデータモデリング言語です。YANGは、ツリーデータ構造を定義する主要な構造と、デバイス構成と動作ステータスを表すために一般的に使用されるさまざまなタイプを定義しています。

YANG はテキスト言語であり、モデルの定義に使用できます。Cisco NX-OSは、次の YANG モデルをサポートします。

- **Cisco NX-OS-device**

  これは、DME REST 定義を反映した Cisco NX-OS 独自の YANG モデルです。このドキュメントでは、このモデルを「デバイス」または「オリジナル」モデルと呼びます。NX-OS デバイス上でのフルアクセスが必要な場合は、「オリジナル」モデルを使用することをお勧めします。詳細については、『NX- API REST』ガイドを参照してください。

- **Open Config**

  このモデルの主な目的は、ネットワーク管理全体の共通機能を抽象化することです。出力は YANG モデルのコレクションであり、Open Config モデルと呼ばれます。詳細については、「OpenConfig YANG について」を参照してください。

NX-OS でサポートされているモデルを確認するには、『Yang NX リポジトリ』ガイドを参照してください。Open Config YANG モデルは、 Cisco NX-OS のリリースに基づいてグループ化されます。

この章では、オリジナルモデルと Open Config モデルの使用方法と制限事項について説明します。

# Cisco デバイス YANG

Cisco NX-OSは、構成と状態の一元化されたビューを表す単一の DME データベースを保持します。セキュアな NX- API RESTインターフェイスを介して DME にアクセスできます。標準規格の YANG セマンティクスに基づいて動作させる場合は、以下のオリジナル YANG の名前空間からデバイスにアクセスできます。

*Cisco-NX-OS-device - http://cisco.com/ns/yang/cisco-nx-os-device*.

（注） デバイス YANG の URL は、一意の名前空間識別子を定義するために使用されます。上記の URL は例であり、実際にアクセス可能な HTTP URL ではありません。

# 注意事項と制約事項

元の YANG は DME データベースを表しており、標準規格の YANG 動作に従っていない動作を継承します。YANG モデルのガイドラインと制限事項は次のとおりです。

- プロパティが数値の場合、DME は特定の値の文字列エイリアスの定義をサポートします。プロパティは、エイリアスまたは数値を使用して構成できます。DME はエイリアスを数値に変換します。YANG 標準を厳守する場合は、エイリアスの使用を避けてください。
- プロパティがビットマスクの場合、DME は特別なキーワード（+、-）をサポートします。ビットマスク範囲の数値を徐々に変更するためのものです。このシンタックスはデバイス YANG で表示されますが、YANG に準拠してはいません。YANG 標準を厳守する場合は、特別なキーワードの使用を避けてください。
- デバイス YANG では、エフェメラルパスはコメント **Ephemeral data** で識別されます。これらのパスは、NX-OS 固有の非永続的な大容量データであり、他のモデルとは異なる方法で管理されます。`<GET>` クエリの `<FILTER>` パラメータに、コメントでマークされた要素に固有のマークがある場合にのみ、取得できます。詳細については、「エフェメラルデータのサポート」を参照してください。
- Cisco NX-OS リリース 10.4(3)F 以降、YANG は 92348GC-X でサポートされます。
- Cisco NX-OS リリース 10.5（2）F 以降、次のプロパティは CLI からの対応するレベルでサポートされていません。
  - **Bgp/neighbors/neighbor/config/send-community**

CLI を介したネイバーでの send-community はサポートされていません。

- **Bgp/neighbors/neighbor/timers/config/minimum-advertisement-interval**

  minimum-advertisement-interval は、CLI を介したネイバーではサポートされていません

- **bgp/neighbors/neighbor/route-reflector/config/route-reflector-client**

  route-reflector-client が CLI を介したネイバーでサポートされていない

- **bgp/peer-groups/peer-group/config/send-community**

  CLI を介したテンプレート ピアでは、send-community はサポートされていません

- **bgp/neighbors/neighbor/as-path-options/config/allow-own-as**
- **gp/neighbors/neighbor/as-path-options/config/disable-peer-as-filter**
- **bgp/peer-groups/peer-group/route-reflector/config/route-reflector-client**

  route-reflector-client は、CLI のテンプレート ピアではサポートされていません。

- **bgp/peer-groups/peer-group/as-path-options/config/allow-own-as**
- **bgp/peer-groups/peer-group/as-path-options/config/disable-peer-as-filter**

# DME からデバイス YANG への移行

Cisco NX-OS DME インターフェイスをよく理解している場合は、既存の DME をデバイス YANG に移行できます。スイッチは、DME モデルとデバイス YANG モデルの間で 1 対 1 の対応を維持します。

次の表に、DME およびデバイス YANG の VRF の表現を示します。

| DME | デバイス YANG |
|---|---|
| `sys`<br>`  +- name`<br>`  +- inst`<br>`  \|  +- name = default`<br>`  +- inst`<br>`  +- name = management` | `System`<br>`  +- name`<br>`  +- inst-items`<br>`  +- Inst-list`<br>`  \|  +- name = default`<br>`  +- Inst-list`<br>`  +- name = management` |

DME モデルをデバイス YANG モデルに移行する場合は、以下のガイドラインに従ってください。

- ルート要素「sys」MO は、YANG モデルでは「System」ルート要素として表されます。
- YANG モデルでは、DME MO はサフィックス「-items」で表されます。
- DME MO に同じタイプの子がある場合、デバイス YANG モデルは子ごとに中間親ノードを追加し、サフィックス「-list」を使用してリストを表します。この動作は OpenConfig と同じです。

- DME プロパティ名は、デバイス YANG モデルでも同じままです。

# OpenConfig YANG について

OpenConfig YANG モデルは、宣言型の構成やモデル駆動型の管理と操作など、高度なネットワーキングの標準をサポートしています。OpenConfig は、ネットワークを構成およびモニタリングするためのビジネスデータモデルです。このデータモデルは、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 は、BGP、OSPF、インターフェイス L2 および L3、VRF、VLAN、TACAC などの幅広い機能領域をサポートしています。

## 注意事項と制約事項

OpenConfig YANG モデルのガイドラインと制限事項は次のとおりです。

### 引き続き必要なガイドラインと制限事項

- ネットワーキングプロトコルでは、別の方法で L2 MACを追加できます。MAC リーフスイッチプロパティ値を入力として `NETCONF GET` 操作を実行する場合は、MAC アドレスの文字を大文字フォーマット（AA:AA:AA:AA:AA:AA）で入力することをお勧めします。たとえば、source-mac: 0A:0B:0C:0D:0E:0F です。

# OpenConfig パス

*表 22 : OpenConfig IP*

| パス | 説明 |
|---|---|
| `…/oc-ip:ip`<br>`…/oc-ip:prefix_length` | 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**）の削除で同じ操作を提供する必要があります。<br>次に例を示します。<br>`oc-ip:ip: remove`<br>` oc-ip:prefix_length: remove` |

表 *23 : OpenConfig* ネットワークインスタンス

| パス | 説明 |
| --- | --- |
| bgp | |

| パス | 説明 |
| --- | --- |
| | 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しても、プロトコルまたは BGP レベルで BGP 情報を削除することはできません。ペイロードに自律システム番号を入力し、デフォルトVRF の設定のみを削除することはできますが、BGP ルーティングインスタンスを削除することはできません。<br><br>以下は、BGP のデフォルト VRF で構成を削除するために使用するペイロードの例です。<br><br>`<rpc`<br><br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br><br>`  message-id="101">`<br>`  <edit-config>`<br>`    <target>`<br>`      <running />`<br>`    </target>`<br>`    <config>`<br>`      <network-instances`<br><br>`xmlns="http://openconfig.net/yang/network-instance">`<br><br>`        <network-instance>`<br>`          <name>default</name>`<br>`          <protocols>`<br>`            <protocol>`<br>`              <identifier>BGP</identifier>`<br>`              <name>bgp</name>`<br>`              <bgp`<br><br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br><br>`                operation="delete">`<br>`                <global>`<br>`                  <config>`<br>`                    <as>100</as>`<br>`                  </config>`<br>`                </global>`<br>`              </bgp>`<br>`            </protocol>`<br>`          </protocols>`<br>`        </network-instance>`<br>`      </network-instances>`<br>`    </config>`<br>`  </edit-config>`<br>`</rpc>`<br><br>**予期される動作**：BG ルーティングインスタンスが削除されることを予期します。これは、`no rout bgp 100` の構成と同様です。 |

| パス | 説明 |
| --- | --- |
| | **実際の動作**：デフォルト VRF の BGP 構成のみが削除されます。同等の結果になる単一の CLI 構成はありません。<br>以下で言及されているのは、削除操作前の実行構成です。<br>`router bgp 100`<br>`  router-id 1.2.3.4`<br>`  address-family ipv4 unicast`<br>`  vrf abc`<br>`  address-family ipv4 unicast`<br>`  maximum-paths 2`<br>以下で言及されているのは、削除操作後の実行構成です。<br>`router bgp 100`<br>`  vrf abc`<br>`  address-family ipv4 unicast`<br>`  maximum-paths 2` |
| `.../set-med` | OpenCofig NETCONF を経由し、同じルートマップ内のメトリックを使用して、 `set-med` コマンドと OSPF コマンドにより BGP アクションを構成しないでください。OSPF コマンドメトリックは、BGP `set-med` プロパティを介した場合、成功するからです。<br>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する必要があります。個別のルートマップを使用して、BGP アクションで `set-med` を使用します。<br>Cisco は、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。 |
| `…/is-reachability` | システム ID をキーとして IS-IS の`is` 到達可能性を検索すると、オリジナルの DME がすべてのエントリを返します。しかし、OpenConfig はメトリック値が最大のエントリを 1 つだけ返します。 |

| パス | 説明 |
| --- | --- |
| …/bgp/global/config/as | |

| パス | 説明 |
| --- | --- |
| | 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 番号にはデフォルト値が存在しないため、BGP インスタンスを削除しないで NETCONF/OPENCONFIG<asn> を削除しようとすると、次の**強調表示された**エラーメッセージが表示されます。<br>764<br><rpc<br><br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"<br><br>message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d"><br><br><edit-config><br><target><br><running /><br></target><br><config><br><network-instances<br><br>xmlns="http://openconfig.net/yang/network-instance"><br><br><network-instance><br><name>default</name><br><protocols><br><protocol><br><identifier>BGP</identifier><br><name>bgp</name><br><bgp><br><global><br><config operation="delete"><br><br><as>100</as><br></config><br></global><br><neighbors><br><neighbor><br><br><neighbor-address>1.1.1.1</neighbor-address><br><br><enable-bfd<br><br>xmlns="http://openconfig.net/yang/bfd"><br><config><br><br><enabled>true</enabled><br></config><br></enable-bfd><br></neighbor><br></neighbors><br></bgp><br></protocol><br></protocols><br></network-instance><br></network-instances><br></config><br></edit-config> |

| パス | 説明 |
| --- | --- |
| | `</rpc>`<br><br>`## Received:`<br><br>`<rpc-reply`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br><br>`message-id="urn:uuid:1ea09de2-605e-46aa-984b-9dfdad03354d">`<br><br>`  <rpc-error>`<br>`  <error-type>protocol</error-type>`<br>`  <error-tag>operation-failed</error-tag>`<br><br>`  <error-severity>error</error-severity>`<br>`  <error-message xml:lang="en">invalid property value , for`<br>`  property asn, class bgpInst</error-message>`<br><br>`<error-path>/config/network-instances</error-path>`<br><br>`  </rpc-error>`<br>`  <rpc-error>`<br>`  <error-type>protocol</error-type>`<br>`  <error-tag>operation-failed</error-tag>`<br><br>`  <error-severity>error</error-severity>`<br>`  <error-message xml:lang="en">invalid property value , for`<br>`  property asn, class bgpInst Commit Failed</error-message>`<br><br>`<error-path>/config/network-instances</error-path>`<br><br>`  </rpc-error>`<br>`</rpc-reply>` |

| パス | 説明 |
| --- | --- |
| …/protocol/ospf | |

| パス | 説明 |
| --- | --- |
| | • OSPF でエリア構成を構成してから削除した場合、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。これらエリアは、OpenConfig YANG の `GETCONFIG/GET` 出力に表示されます。<br>• ユーザーは、OSPF ポリシー `match ospf-area` 構成で、OpenConfig YANG の 1 つのエリアを構成できます。`match ospf-area 100 101` などのように、複数のエリアを構成することもできます。ただし、OpenConfig YANG で構成できるのは、1 つのエリアのみです。<br>たとえば、 `match ospf-area 100` と入力します。<br>• 同じエリアリストで、エリア仮想リンクと、エリアインターフェイス構成ペイロードの両方を構成することはできません。同じペイロード内で、エリアコンテナペイロードを、仮想リンクエリアと、インターフェイスエリアに分割したことを確認してください。<br>• OSPF OpenConfig YANG では MD5 認証文字列は構成できません。<br>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。<br>`leaf authentication-type {`<br>`type string;`<br>`description`<br>`"The type of authentication that should be used on this interface";`<br>`}`<br>• 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。<br>• 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。<br>• デフォルトVRF にある OSPF または BGP インスタンス構成は削除できません。たとえば、**router ospf 1/router bgp 1** で、デ |

| パス | 説明 |
|---|---|
| | フォルトのネットワークインスタンスを使用してプロトコルコンテナを削除することはできません。<br>• 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 がエラー応答を送信することがあります。問題がある場合、インターフェイスを追加することはできず、RPC 応答には次のようなリストマージ失敗エラーが表示されます。<br>`<rpc-reply`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br>`message-id="urn:uuid:39507023-8569-4cf8-869c-e19aaf76a260">`<br>`<rpc-error>`<br>`<error-type>protocol</error-type>`<br>`<error-tag>operation-failed</error-tag>`<br>`<error-severity>error</error-severity>`<br>`<error-message xml:lang="en">List Merge Failed: operation-failed</error-message>`<br>`<error-path/>`<br>`</rpc-error>`<br>`</rpc-reply>` |

表 *24 : OpenConfig* ルーティングポリシー

| パス | 説明 |
|---|---|
| /bgp-defined-sets/community-sets/community-set/<br>/bgp-defined-sets/as-path-sets/as-path-set/ | community-set および as-path-set のアクションタイプは常に permit です。<br>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。 |

| パス | 説明 |
| --- | --- |
| /bgp-defined-sets/community-sets/community-set/<br>/bgp-defined-sets/as-path-sets/as-path-set/ | community-set および as-path-set のアクションタイプは常に permit です。<br>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。 |
| /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。OpenConfig YANG モデルでは、community-set-name に区別はありません。<br>OpenConfig YANG を使用して community-set-name を作成すると、次の変更が構成されます。<br>• 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。<br>• community-member が拡張形式（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。<br>`<community-set>`<br>`<community-set-name>oc_commset1d</community-set-name>`<br>`<config>`<br>`<community-set-name>oc_commset1d</community-set-name>`<br>`<community-member>0:1</community-member>`<br>`<community-member>_1_</community-member>`<br>`</config>`<br>`</community-set>`<br>上記の OpenConfig YANG 構成は、次の CLI にマップされます。<br>`ip community-list expanded oc_commset1d_exp`<br>` seq 5 permit "_1_"`<br>`ip community-list standard oc_commset1d_std`<br>` seq 5 permit 0:1` |

| パス | 説明 |
| --- | --- |
| /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。OpenConfig YANG モデルでは、community-set-name に区別はありません。<br><br>OpenConfig YANG を使用して community-set-name を作成すると、次の変更が構成されます。<br><br>• 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。<br><br>• community-member が拡張形式（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。<br><br><community-set><br><community-set-name>oc_commset1d</community-set-name><br><config><br><community-set-name>oc_commset1d</community-set-name><br><community-member>0:1</community-member><br><community-member>_1_</community-member><br></config><br></community-set><br><br>上記の OpenConfig YANG 構成は、次の CLI にマップされます。<br><br>ip community-list expanded oc_commset1d_exp<br>seq 5 permit "_1_"<br>ip community-list standard oc_commset1d_std<br>seq 5 permit 0:1 |

| パス | 説明 |
| --- | --- |
| /bgp-conditions/match-community-set/config/community-set/ | |

| パス | 説明 |
| --- | --- |
| | 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。<br>• CLI の場合：<br>`ip community-list standard 1-1 seq 1 permit 1:1`<br>`ip community-list standard 1-2 seq 1 permit 1:2`<br>`ip community-list standard 1-3 seq 1 permit 1:3`<br>`route-map To_LC permit 10`<br>`match community 1-1 1-2 1-3`<br>• 対応する OpenConfig YANG ペイロードは次のとおりです。<br>`<config>`<br>`<routing-policy`<br>`xmlns="http://openconfig.net/yang/routing-policy">`<br>`<defined-sets>`<br>`<bgp-defined-sets`<br>`xmlns="http://openconfig.net/yang/bgp-policy">`<br>`<community-sets>`<br>`<community-set>`<br>`<community-set-name>cs</community-set-name>`<br>`<config>`<br>`<community-set-name>cs</community-set-name>`<br>`<community-member>1:1</community-member>`<br>`<community-member>1:2</community-member>`<br>`<community-member>1:3</community-member>`<br>`</config>`<br>`</community-set>`<br>`</community-sets>`<br>`</bgp-defined-sets>`<br>`</defined-sets>`<br>`<policy-definitions>`<br>`<policy-definition>`<br>`<name>To_LC</name>`<br>`<statements>`<br>`<statement>`<br>`<name>10</name>`<br>`<conditions>`<br>`<bgp-conditions` |

| パス | 説明 |
| --- | --- |
| | `xmlns="http://openconfig.net/yang/bgp-policy">`<br><br>`<match-community-set>`<br>`<config>`<br><br>`<community-set>cs</community-set>`<br>`</config>`<br>`</match-community-set>`<br>`</bgp-conditions>`<br>`</conditions>`<br>`</statement>`<br>`</statements>`<br>`</policy-definition>`<br>`</policy-definitions>`<br>`</routing-policy>`<br>`</config>`<br><br>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。<br><br>`ip community-list standard cs_std seq 5 permit 1:1`<br>`ip community-list standard cs_std seq 10 permit 1:2`<br>`ip community-list standard cs_std seq 15 permit 1:3`<br>`route-map To_LC permit 10 match community cs_std` |

| パス | 説明 |
| --- | --- |
| /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。以下は、OpenConfig YANG ペイロードの next-hop-in の例です。<br>`<policy-definition>`<br>`  <name>sc0</name>`<br>`  <statements>`<br>`  <statement>`<br>`  <name>5</name>`<br>`  <conditions>`<br>`  <bgp-conditions`<br>`xmlns="http://openconfig.net/yang/bgp-policy">`<br>`  <config>`<br>**`<next-hop-in>2.3.4.5</next-hop-in>`**<br>`  </config>`<br>`  </bgp-conditions>`<br>`  </conditions>`<br>`  </statement>`<br>`  </statements>`<br>`</policy-definition>`<br>以下は、CLI での同じ情報の例です。<br>`ip prefix-list IPV4_PFX_LIST_OPENCONFIG_sc0_5 seq 5 permit 2.3.4.5/32`<br>`route-map sc0 permit 5 match ip next-hop prefix-list IPV4_PFX_LIST_OPENCONFIG_sc0_5` |
| /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 | enum "SELF" はサポートされていません |
| /bgp-conditions/match-community-set/config/community-set | match community <community-set>_std にのみマップした場合、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 セットへのマッチングはサポートされていません。 |

| パス | 説明 |
| --- | --- |
| …/match-tag-set | tag-sets 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。<br>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。 |

表 *25 : OpenConfig* インターフェイス

| パス | 説明 |
| --- | --- |
| *interfaces/interface/ethernet/switched-vlan/config/interface-mode* | |

| パス | 説明 |
| --- | --- |
| | トランク モードインターフェイスとトランクに関連する構成を同じ OpenConfig ペイロードで同時に構成しようとすると、構成が正常に完了しません。<br><br>これは、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** であるためです。トランク固有の構成を実装するには、最初にインターフェイスモードを **trunk** に変更します。そして、トランク関連の構成のみ有効にできます。NX-OS は、同じ要求内でそれらを変更することをサポートしていません。<br><br>リクエストは、次の 2 段階で送信することを推奨します。<br><br>• インターフェイスをトランク モードに変更するための最初の手順<br><br>• トランク固有の構成を変更するための 2 番目のステップ<br><br>**1.** インターフェイスをトランクモードに構成します。<br><br><code><rpc<br><br>xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"<br><br>message-id="101"><br><edit-config><br><config><br><interfaces<br><br>xmlns="http://openconfig.net/yang/interfaces"><br><br><interface><br><name>eth1/2</name><br><ethernet<br><br>xmlns="http://openconfig.net/yang/interfaces/ethernet"><br><br><switched-vlan<br><br>xmlns="http://openconfig.net/yang/vlan"><br><br><config><br><br><interface-mode>TRUNK</interface-mode><br></config><br></switched-vlan><br></ethernet><br></interface><br></interfaces></code> |

| パス | 説明 |
| --- | --- |
| | `</config>`<br>`</edit-config>`<br>`</rpc>` |

| パス | 説明 |
| --- | --- |
| | **2.** 2番目のステップでは、トランク固有の構成を指定します。次に、説明のために 2 つの個別の例を示します。この手順では、必要に応じて、同じペイロードにこれら 2 つの例を組み合わせることができます。<br>• 例：change "native-vlan"<br>`<rpc`<br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br>`message-id="101">`<br>`<edit-config>`<br>`<config>`<br>`<interfaces`<br>`xmlns="http://openconfig.net/yang/interfaces">`<br>`<interface>`<br>`<name>eth1/2</name>`<br>`<ethernet`<br>`xmlns="http://openconfig.net/yang/interfaces/ethernet">`<br>`<switched-vlan`<br>`xmlns="http://openconfig.net/yang/vlan">`<br>`<config>`<br>`<native-vlan>999</native-vlan>`<br>`</config>`<br>`</switched-vlan>`<br>`</ethernet>`<br>`</interface>`<br>`</interfaces>`<br>`</config>`<br>`</edit-config>`<br>`</rpc>`<br>• 例：change "trunk-vlans"<br>「trunk-vlans」はリーフリストであり、各範囲は一意のリストエントリであることに注意してください。Cisco NX-OS では、インターフェイスが最初にトランク モードに移されると、NX-OS は便宜上、範囲全体を許可するためにデフォルトのリスト エントリ「1..4094」を自動的に作成します。範囲をカスタマイズするには、ユーザーはデフォルトエントリを削除し、カスタマイズされた重複しない範囲を追加する必要があります。 |

| パス | 説明 |
|---|---|
| | `<rpc`<br><br>`xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br><br>`message-id="101">`<br>`<edit-config>`<br>`<config>`<br>`<interfaces`<br><br>`xmlns="http://openconfig.net/yang/interfaces">`<br><br>`<interface>`<br>`<name>eth1/2</name>`<br>`<ethernet`<br><br>`xmlns="http://openconfig.net/yang/interfaces/ethernet">`<br><br>`<switched-vlan`<br><br>`xmlns="http://openconfig.net/yang/vlan">`<br><br>`<config>`<br>`<trunk-vlans`<br><br>`<xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"`<br><br>`operation="delete">1..4094</trunk-vlans>`<br><br>`<trunk-vlans>401</trunk-vlans>`<br><br>`<trunk-vlans>999</trunk-vlans>`<br>`</config>`<br>`</switched-vlan>`<br>`</ethernet>`<br>`</interface>`<br>`</interfaces>`<br>`</config>`<br>`</edit-config>`<br>`</rpc>` |
| *interfaces/interface/ethernet/switched-vlan/config/trunk-vlans* | OpenConfig YANG の制限により、ペイロード内の VLAN とインターフェイスで構成された VLAN との間で VLAN の構成が重複しないようにしてください。重複した場合、OpenConfig による構成は失敗します。<br><br>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 と異なり、VLAN 範囲に関して適切であることを確認してください。 |

| パス | 説明 |
| --- | --- |
| | スイッチポート、shut/no shut、MTU、および mac-address の場合は次のとおりです。<br><br>スイッチポート、shut/no shut、MTU、および mac-address を設定する場合は、ASCII リロードが必要です。バイナリ結果をリロードすると、構成は失われます。 |

表 *26 : OpenConfig LACP*

| パス | 説明 |
| --- | --- |
| *lacp/interfaces/interface/config/lacp-mode* | 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 の場合、ポートチャネルモードは、チャネルグループモード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構成されます。<br><br>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構成しますが、ポートチャネルインターフェイスで NX-OS の **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構成は表示されません。<br><br>メンバーをポートチャネルに追加すると、**show running interface ethernet <>** は、ポートチャネル モードの構成を、チャネルグループモードアクティブまたはパッシブとして表示します。<br><br>（注）<br>OpenConfig を介して作成されたすべてのポートチャネルは、その後も OpenConfig によって管理する必要があります。 |

| パス | 説明 |
|---|---|
| *lacp/interfaces/interface/config/interval* | 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。<br>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。<br>• 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。<br>OpenConfig を使用してポートチャネル間隔を構成し、メンバーを追加します。新しいメンバーに構成を適用するために、OpenConfig を使用して間隔を構成してください。 |
| *lacp/interfaces/interface/config/system-id-mac* | ポートチャネルごとの `system-id-mac` はサポートしていません。 |
| LACP | 次のメンバー状態データは、ポートが管理上アップ状態の場合にのみに限られます。<br>• LACP<br>• インターフェイス<br>• インターフェイス<br>• メンバー<br>• 状態 |

次の状態コンテナは、OpenConfig ACL にインターフェイス参照レベルで実装されています。

**表 *27 : OpenConfig ACL***

| パス | 説明 |
|---|---|
| *acl/interfaces/interface/interface-ref/stateforacl/interfaces/state* | 該当なし |
| *acl/interfaces/interface/interface-ref/state/interface* | read-only |
| *acl/interfaces/interface/interface-ref/state/subinterface* | read-only |

表 *28 : OpenConfig QoS*

| パス | 説明 |
|---|---|
| QoS | • 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。<br>• 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。<br>• 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。<br>• 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。 |

次のシステム構成コンテナは、ドメイン名、ログインバナー、および motd-バナーモデルに実装されています。

表 *29 : OpenConfig* システム

| パス | 説明 |
|---|---|
| *system/config/domain-name* | System/dns-items/<br>prof-items/Prof-list/dom-items/name |
| *system/config/login-banner* | System/userext-items/postloginbanner-items/message |
| *system/config/motd-banner* | System/userext-items/preloginbanner-items/message |

# ハイスケールデータに関するガイドラインと制約事項

Cisco NX-OSは、ハイスケールデータを持つ新しい一連の動作状態として、OpenConfig パスをサポートします。ガイドラインとと制約事項は次のとおりです。

- 最適なパフォーマンスを得るには、正確なパスを指定してデータを取得する必要があります。親レベルのパスクエリでは、同じパフォーマンスは得られません。
- ハイスケールパスはgNMIでのみサポートされ、RESTCONFまたはNETCONFではサポートされません。
- ハイスケールパスは、suppress-redundant をサポートしていません。
- ハイスケールパスは、gNMI ON_CHANGEサブスクリプションをサポートしていません。

| パス | 説明 |
|---|---|
| */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* | 親レベルクエリの場合、インフラストラクチャはリスト項目のすべてのキーを取得し、これらのリスト項目ごとに、残りのデータを入力する要求が送信されます。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。<br><br>インフラストラクチャに静的エントリのトラックがあり、バックエンドに静的エントリとダイナミックエントリがある場合を考えてみましょう。リストについては、インフラストラクチャをたどって静的エントリごとに要求が送信されるため、不完全なデータが生成されます。現在のリリースでこの制限を受けるパスは次のとおりです。<br><br>「`/interfaces/interface/routed-vlan/ipv6/neighbors/neighbor/state`」および「`/interfaces/interface/routed-vlan/ipv4/neighbors/neighbor/state`」です。<br><br>データには、ダイナミックと静的の両方の ARP および ND エントリが含まれています。正確な経路が指定されている場合、親経路が与えられていれば、静的エントリが含まれることになります。 |
| */network-instances/network-instance/protocols/protocol/bgp/rib/afi-safi/ipv4-mvpn-mdt/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/overlay/points/overlay-point/endpoints/endpoint/vlans/endpoint-vlans* | |
| */network-instances/network-instance/overlay/points/overlay-point/endpoints/endpoint/vlans/endpoint-vlan-vis* | |

# OpenConfig サポートの構成

NETCONF、RESTCONF、および gRPC などのプログラマビリティエージェントで、OpenConfig サポートを有効または無効にします。

### 手順の概要

1. **configureterminal**
2. **featureopenconfig**

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configureterminal**<br>例：<br>`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 | **featureopenconfig**<br>例： | 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。<br>`switch(config)# feature netconf` |

# YANG バージョンのアップグレードまたはダウングレード

YANG モデル は、論理的にサポートされている構成または状態の集合です。このモデルは、以前または将来の互換性リリースをサポートしていません。

Cisco NX-OSは、1 つのリリースで複数の YANG バージョンをサポートしていません。各 NX-OS リリースは、特定の認可された YANG コレクションをサポートします。それぞれの NX-OS リリースについては、「Yang NX リポジトリ」を参照してください。

NX-OS リリースをアップグレードまたはダウングレードすると、サポートされる YANG モデルにいくらかの変更が生じます。YANG モデルを確認し、適切なアクションを実行する必要があります。

| 変更 | 推奨 |
|---|---|
| 追加されたパス | 追加の YANG パスを処理する方法を評価できます。 |
| 変更されたパス | これは、プロパティタイプが整数から浮動小数点数に変更されたた<br>ト側のロジックを更新する必要があります。 |

| 変更 | 推奨 |
| --- | --- |
| パスが削除された | OpenConfig では、コミュニティが、ベンダー間で適用されない場合に対応する機能を使用するには、デバイス YANG や CLI なクセスします。 |

# YANG 向けの RBAC

Cisco NX-OSは、管理者以外のユーザーロールによる YANG パスへの書き込み権限をサポートします。

# 注意事項と制約事項

次に、YANG の RBAC のガイドラインと制限事項を示します：

- ロール network-admin を持つユーザーには、書き込みアクセス権があります。このユーザーは、管理者以外のグループに他の編集権限を付与できます。
- ユーザーロール network-admin にのみ、読み取りアクセス権があります。
- RBAC は、管理者以外のユーザーロールへ、ルートシステムアクセス権を提供しません。
- RBAC は、管理者以外のユーザーロールへ、パス *System/yangrbacdb-items* または *System/rbacdb-items* へのアクセス権を提供しません。これらのユーザーロールは、自分のロールのアクセス権を変更できないよう制限されています。
- RBAC の承認は、候補ではなく実行構成でサポートされます。候補構成は NETCONF でサポートされ、RESTCONF または gNMI ではサポートされません。Cisco では、単一トランザクションの実行構成で RBAC を構成することを推奨しています。候補構成を使用して RBAC を変更する場合は、RBAC 自体を変更してコミットすることをお勧めします。RBAC 構成と通常の構成を同時に変更しないでください。このような場合、候補構成での RBAC の変更は、実際には変更されません。
- パスの最後の要素とそのサブツリーへのアクセスを提供していることを確認します。たとえば、パス「x/y/z」の場合、最後のパス要素と子ツリーを変更するためのアクセスをユーザーに提供します。
- 同じパス内の特定の親または子ノードへの書き込みアクセス権を提供する場合、それらのルールを単一のルールにまとめる構文はありません。個別に設定する必要があります。

  たとえば、次のルールは、priv-9 へのアクセスを提供して、すべてのインターフェイスの説明を変更します。ただし、特定の SVI インターフェイスの 2 つのプロパティを変更することはできます。これらのルールをまとめることはできません。

  - interfaces/interface/description (priv-9)
  - interfaces/interface[id=vlan100]/type (priv-9)

- interfaces/interface[id=vlan100]/enabled (priv-9)

- RBAC は最大 512 のルールをサポートします。
- RBAC ユーザーロールは前方参照をサポートします。
  - ユーザーロールは、存在しないロールを指定できます。
  - ユーザーロールは、存在しない yang パスを指定できます。
- 次の構文を確認してください。
  - 特定のワイルドカード文字を追加しないでください。
  - 名前空間の文字列は、最初の要素にのみ追加してください。
  - パスの先頭に「/」を追加しないでください。
  - 「system」文字列のみのパスは使用できません。
- RBAC は、NETCONF の既存の制限、none 動作には影響しません。

# YANG RBAC の構成

特定のユーザーロールへの yang パスを構成します。

## 手順の概要

1. **cofnigureterminal**
2. **rbacyang-rule***yang-pathrole*
3. **allow-writes**

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **cofnigureterminal**<br>例：<br>`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 | **rbacyang-rule***yang-pathrole*<br>例：<br>`switch(config)# rbac yang-rule System/intf-items/aggr-items/AggrIf-list[id=po101] network-operator`<br>例： | ユーザーロールへの yang パスを構成します。パスはデバイスまたは OpenConfig YANG パスのいずれかです。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `switch(config)# rbac yang-rule openconfig-interface:interfaces/interface[name=vlan100]/desc priv-9` | |
| ステップ 3 | **allow-writes**<br>例：<br>`switch(config)# allow-writes` | 書き込み権限を有効にします。 |

# YANG のトラブルシューティング

次のコマンドを使用して、YANG の構成を確認できます。具体的には、「機能 OpenConfig」の有効化を確認します。

特定のエージェントの読み取りまたは書き込み操作に関連する問題については、それぞれのエージェントのトラブルシューティングガイドを参照してください。

| コマンド | 説明 |
|---|---|
| `show running-config openconfig` | OpenConfig が有効になっているかどうかを確認します。 |
| `show openconfig nxsdk event-history { event \| errors }` | OpenConfig 機能のデバッグを確認します。 |
| `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


- NETCONF エージェントについて （369 ページ）
- マニュアルの変更履歴 （370 ページ）
- NETCONF に関する注意事項と制限事項 （371 ページ）
- NETCONF エージェントの構成 （373 ページ）
- NETCONF セッションの管理 （376 ページ）
- NETCONF Get / Set （379 ページ）
- NETCONF 通知 （388 ページ）
- NETCONF デバイス YANG RPC （393 ページ）
- Netconf クライアントの例 （397 ページ）
- NETCONF エージェントのトラブルシューティング （400 ページ）
- NETCONF 明示モードについて （404 ページ）
- 注意事項と制約事項 （404 ページ）
- NETCONF 明示モードの Get/Set （405 ページ）
- CLI を使用して構成を追加する （409 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の応答のため、SSH または TLS 上のセキュアな転送を可能にします。

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 *30*：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。 |
| 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 |
| 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 |
| 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。 |
| create-subscription | イベント通知ストリームへのサブスクライブ |

# マニュアルの変更履歴

| リリース | 説明（**Description**） |
|---|---|
| 9.3(1) | 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 |
| 9.3(3) | RFC 6241 準拠 |
| 9.3(5) | NETCONF 通知での OpenConfig モデルをサポートします |
| 10.2(1) | チェックポイント、ロールバック、インストール、CA 証明書のインポート、モジュールのリロード、個々のモジュールのリロード、およびファイルのコピーといった RPC 操作をサポートします |
| 10.3(1) | Cisco Nexus 9800 プラットフォームスイッチ |
| 10.3 (2) | edit-config の RBAC のサポート |

| リリース | 説明（Description） |
|---|---|
| 10.3(3) | TLS を介した Netconf トランスポートのサポート |
| 10.4（3） | Netconf 明示モードをサポート |
| 10.5（3） | Netconf 並列実行のサポート |

# 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：

**Netconf YANG 操作**

- 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。
- Cisco NX-OS の 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）を完全にはサポートしていません。代わりに、**network-admin** 以外のロールにパスごとの EDIT 権限を付与できます。
- **network-admin** アクセス権のないユーザーは、DME 構造にアクセスできません。
- 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。次のようなエラー応答が表示された場合、要求は名前空間で伝送されていません。

  名前空間とフィルタのない要求は、サポートされていない操作です。

  次に、NX-OS リリース 9.3(1) 以降での要求と応答の正しい動作例を示します。

```
<get>
  <filter>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    </System>
  </filter>
</get>
```

- `<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を、置換を行うための基礎の構成とする方が適切です。

- 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。

  デバイスモデル全体をクエリするにはオブジェクトが多すぎる（`459134 > 250000`）

### ネイティブデバイス RPC

- Cisco NX-OS では、チェックポイント、ロールバック、インストール、CA 証明書のインポート、モジュールのリロード、個々のモジュールのリロード、およびファイルのコピーといった、デバイス YANG RPC の操作がサポートされています。

### サブスクリプション/通知

- Cisco NX-OS は、最大 5 つのサブスクリプションをサポートします（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）。
- 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。
- **Netconf 並列実行**の実行
  - Cisco NX-OS は、最大 10 の並列 Netconf 要求（クライアント セッションごとに 1 つの要求）をサポートします。
- Cisco NX-OS リリース 10.4(3)F 以降、NETCONF は 92348GC-X でサポートされます。

### トランスポート層セキュリティ（TLS）

- Cisco NX-OS リリース 10.4（3）F 以降、TLS は、v1.3 がサポートされています。最低限サポートされるバージョンは v1.2 です。

### Netconf 並列実行の実行

- Cisco NX-OS リリース 10.5(3)F 以降、最大 10 の並列 Netconf 要求（クライアントセッションごとに 1 つの要求）がサポートされます。

# NETCONF エージェントの構成

## SSH を介する NETCONF エージェントの構成

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。

### 始める前に

NETCONF を使用してスイッチと通信する前に、NETCONF エージェントを有効にする必要があります。NETCONF エージェントを有効または無効にするには、**[no] feature netconf** コマンドを入力します。

### 手順の概要

1. **configure terminal**
2. **feature netconf**
3. （任意） **netconf idle-timeout** *it-num*

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`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**<br>例：<br>`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*<br>例：<br>`switch(config)# netconf idle-timeout 5` | アイドル状態のクライアントセッションが切断されるまでのタイムアウトを分単位で指定します。*it-num* の範囲は 0 ～ 1440 分です。デフォルトのタイムアウトは 5 分です。値を 0 に設定するとタイムアウトが無効になります。 |

# TLS を介する NETCONF エージェントの構成

この手順では、NETCONF エージェント over TLS を有効にして設定する方法について説明します。これにより、追加のTLS トランスポートチャンネルを SSH トランスポートと並行して実行できます。

### 始める前に

サーバ側とクライアント側の両方の認証に必要な証明書ファイルを準備して署名します。

（注） これは Netconf に固有のものではないため、既存のトラストポイントファイルを再利用できます。

## 手順の概要

1. **configure terminal**
2. （任意） **crypto ca trustpoint** *server-trustpoint*
3. （任意） **crypto ca trustpoint** *server-trustpoint* **pkcs12 bootflash:** *server-ca-file pkcs-password*
4. （任意） **crypto ca trustpoint** *client-root-trustpoint*
5. （任意） **rsakeypair** *client-key*
6. （任意） **crypto ca authenticate** *client-root-trustpoint*
7. **netconf tls certificate** *server-trustpoint*
8. **netconf tls client root certificate** *client-root-trustpoint*
9. **netconf tls port** *port*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`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 | （任意） **crypto ca trustpoint** *server-trustpoint*<br>例：<br>`switch(config)# crypto ca trustpoint tls_server_trustpoint` | サーバ認証用のトラストポイントを作成します。<br>（注）<br>使用可能なサーバとクライアントのトラストポイントがすでにある場合、ステップ 2 ～ 6 はオプションです。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 3 | （任意） **crypto ca trustpoint** *server-trustpoint* **pkcs12 bootflash:** *server-ca-file pkcs-password*<br>例：<br>`switch(config)# crypto ca import tls_server_trustpoint pkcs12 bootflash:server.pfx test` | サーバの pkcs12 ファイルをトラストポイントにインポートします。 |
| ステップ 4 | （任意） **crypto ca trustpoint** *client-root-trustpoint*<br>例：<br>`switch(config)# crypto ca trustpoint tls_client_trustpoint` | クライアント認証用のトラストポイントを作成します。 |
| ステップ 5 | （任意） **rsakeypair** *client-key*<br>例：<br>`switch(config)# rsakeypair client-key` | クライアントトラストポイントの rsa キーペアを生成します。 |
| ステップ 6 | （任意） **crypto ca authenticate** *client-root-trustpoint*<br>例：<br>`switch(config)# crypto ca authenticate tls_client_trustpoint` | クライアント証明書をインポートします。この手順では、手動でコピーして貼り付ける必要があります。手順に従ってください。 |
| ステップ 7 | **netconf tls certificate** *server-trustpoint*<br>例：<br>`switch(config)# netconf tls certificate tls_server_trustpoint` | サーバアイデンティティ証明書をホストするトラストポイントを入力します。 |
| ステップ 8 | **netconf tls client root certificate** *client-root-trustpoint*<br>例：<br>`switch(config)# netconf tls client root certificate tls_client_trustpoint` | クライアント CA ルート証明書をホストするトラストポイントを入力します。 |
| ステップ 9 | **netconf tls port** *port*<br>例：<br>`switch(config)# netconf tls port 7000` | TLS ポートを指定します。デフォルトポートは 6513 です。 |

## キー/証明書の生成の例

次に、スイッチの bash シェルで自己署名キー/証明書を生成する例を示します。

これは実験的な使用のみであることに注意してください。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* シリーズ *NX-OS* セキュリティ構成ガイド』の「**アイデンティティ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。

（注） このタスクは、NX-OS 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。

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
```

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
```

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

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

4. セットアップを確認します。

```
Verify the setup.
switch(config)# show crypto ca certificates Trustpoint: mytrustpoint certificate:
subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San
Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R serial=0413
notBefore=Nov 5 16:48:58 2015 GMT notAfter=Nov 5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
CA certificate 0: subject= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San
Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R
issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San
Jose/ST=CA/street=3700 Cisco
Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R serial=0413
notBefore=Nov 5 16:48:58 2015 GMT notAfter=Nov 5 16:48:58 2035 GMT
SHA1 Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E
purposes: sslserver sslclient
```

# NETCONF セッションの管理

NETCONF は、クライアントとサーバー間の永続的な接続を必要とする接続指向のプロトコルです。スイッチ上の NETCONF エージェントは、管理ポート IP アドレスのポート 830 でリッスンします。NETCONF エージェントは、ポート 22 で実行されている XML サーバーと混同されることが多いことに注意してください。ポート 22 で実行されている XML サーバーについて

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 *10.5*（*x*）』の「XML Management Interface」の章を参照してください。

# セッションの開始

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、クライアントも<hello>メッセージをサーバに送信します。<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。NETCONFクライアントとサーバの両方は、相手の<hello>メッセージで、共通したプロトコルバージョンがアドバタイズされていることを確認する必要があります。また、サーバの<hello> メッセージには<session-id> を含める必要がありますが、クライアントの<hello> メッセージに<session-id>を含めてはなりません。

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>メッセージはサーバから受信され、2番目のメッセージはクライアントから送信されたものです。サーバの<hello>メッセージは、プロトコルバージョンの「urn:ietf:params:netconf:base:1.1」と、サポートされているデータモデルを示しています。次の例は、現在のCisco NX-OSリリースを反映していない可能性があります。

（注） サーバの<hello> メッセージには<session-id>が含まれていますが、クライアントのメッセージには含まれていません。

```
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>
...
    </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>
]]>]]>
```

素の **ssh** コマンドで NETCONF を使用することは便利ではなく、エラーが発生しやすくなります。RFC 6242（SSH 上で NETCONF プロトコルを使用する）から分かるとおり、メッセージの枠組みが複雑だからです。上の **ssh** コマンドは、説明のみを目的として使用しています。**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存在します。ncclient はそのような例の 1 つです。例のセクションを参照してください。

# セッションの終了

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>` と `<kill-session>` です。サーバは `<close-session>` 要求を受け取ると、セッションに関連付けられているロックとリソースを解放し、クライアントとの接続を閉じることにより、問題の生じない方法でセッションを終了します。

次の例は、`<close-session>` 要求と、それが正しく実行されたとの応答を示しています。

```
<rpc
  message-id="1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>

<rpc-reply
  message-id="1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

`<kill-session>` 要求は別のセッションを強制的に終了します。要求メッセージには `<session-id>` が必要です。`<kill-session>` 要求を受信すると、サーバ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。

次の例は、`<kill-session>` 要求と、それが正しく実行されたとの応答を示しています。

```
<rpc
  message-id="2"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>296324181</session-id>
  </kill-session>
</rpc>

<rpc-reply
  message-id="2"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <ok/>
</rpc-reply>
```

`<close-session>` と `<kill-session>` 要求以外にも、クライアントが一定時間何も要求を送信しなかった場合、セッションは自動的に終了されます。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 Get / Set

このセクションでは、データストアの操作とクエリのためにサポートされているNETCONF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。これらの操作の基本的な使用方法について説明しています。詳細については、RFC 6242 を参照してください。

### <get-config>

この操作により、指定したデータストアの構成データを取得します。サポートされるパラメータは `<source>` と `<filter>` です。`<source>` は、`<running/>` のように、クエリの対象となるデータストアを指定します。これは現在アクティブな構成を保持しています。`<filter>`は、指定されたデータストアのうち、どの部分を取得するかを指定します。

次に、`<get-config>` を使用する要求および応答メッセージの例を示します。

- 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、指定された構成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/>。<running/> データストアを操作すると、スイッチの構成に直ちに変更が加えられます。一方、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。

<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。意図するモデルは「xmlns」属性で指定します。<config> サブツリーの任意の数の要素に、それぞれ 1 つの「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。「operation」属性が指定されていない場合は、「merge」操作がデフォルトと見なされます。デフォルトの動作は、オプションの<default-operation>パラメータによって上書きできます。パラメータには「merge」、「replace」、「none」などがあります。

- 「merge」操作は、構成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「create」とは異なります。
- 「remove」操作は、構成データが存在しない場合にエラーが返されないという点で「delete」とは異なります。

次に、<edit-config> を使用する要求および応答メッセージの例を示します。

- 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、それぞれ `<source>` と `<target>` です。

次に、`<copy-config>` を使用する要求および応答メッセージの例を示します。

- 実行構成をスタートアップ構成にコピーします。

```
<rpc
  message-id="106"
```

```
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <target>
      <startup/>
    </target>
    <source>
      <running/>
    </source>
  </copy-config>
</rpc>

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="106">
  <ok/>
</rpc-reply>
```

- 実行構成を候補構成にコピーします。

```
<rpc
  message-id="107"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <target>
      <candidate/>
    </target>
    <source>
      <running/>
    </source>
  </copy-config>
</rpc>

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="107">
  <ok/>
</rpc-reply>
```

### <lock>

`<lock>`操作を使用すると、クライアントは構成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>`セッションの操作時、またはセッションの終了時にリリースされます。`<target>`パラメータは、ロックするデータストアを指定します。

次に、`<lock>` を使用する要求および応答メッセージの例を示します。

- ロックの取得に成功した場合：

```
<rpc
  message-id="108"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target>
      <running/>
    </target>
  </lock>
</rpc>

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="108">
```

```
  <ok/>
</rpc-reply>
```

- 別のセッションですでに使用されてい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、<lock> 操作によって取得した構成のロックをリリースします。`<lock>` 操作を発行したのと同じセッションでのみ、`<unlock>` 操作を使用できます。`<target>` パラメータは、ロックを解除するデータストアを指定するために使用します。

次に、`<unlock>` を使用する要求および応答メッセージの例を示します。

- ロック解除

```
<rpc
  message-id="110"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target>
      <candidate/>
    </target>
  </unlock>
</rpc>

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="110">
  <ok/>
</rpc-reply>
```

### <get>

<get> 操作は、実行中の構成とデバイスの状態情報を取得します。サポートされているパラメータは `<filter>` です。`<filter>` パラメータは、実行構成の動作状態データのうち、どの部分を取得するかを指定します。

次に、`<get>` を使用する要求および応答メッセージの例を示します。

- 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>
              <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。<source> パラメータは <candidate/> をサポートします。

次に、<validate> を使用する要求および応答メッセージの例を示します。

- 候補データストアの内容を検証します：

```
<rpc
  message-id="112"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
</rpc>

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="112">
  <ok/>
</rpc-reply>
```

### <commit>

候補構成を実行構成にコミットします。パラメータ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。

<commit> は、<confirmed/> パラメータとともに発行されます。これは確認を求めるコミットと見なされ、<confirmed/> パラメータを持たない別の <commit> 操作が続く場合にのみ、実際に確定します。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>。<confirm-timeout> は、確認を求めるコミットが発行されてから、そのコミットが元に戻されて、実行構成が復元されるまでの秒数です。その間に確認を行うコミットが発行されると、コミットは確定します。または、確認を求める別のコミットにより、タイムアウトがリセットされる場合もあります。<confirm-timeout> が指定されていない場合の、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。<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を求めるコミットは保持されます。<persist> パラメータの値は、任意のセッションからの確認を求めるパラメータを識別するために使用されるもので、その後の確認を求めるコミットまたは確認を行うコミットの <persist-id> パラメータの値でも使用する必要があります。

次に、<commit> を使用する要求および応答メッセージの例を示します。

- 候補データストアの内容をコミットします。

```
<rpc
  message-id="113"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit/>
</rpc>

<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="113">
  <ok/>
</rpc-reply>
```

- タイムアウト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を求めるコミットをキャンセルする必要がある場合、確認を求めるそのコミットの `<persist>` パラメータで与えたのと同じ値を、`<persist-id>` パラメータで使用する必要があります。

- 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。パラメータは必要ありません。

次に、`<discard-changes>` を使用する要求および応答メッセージの例を示します。

- 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。https://datatracker.ietf.org/doc/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、デバイス YANG 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。

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ライブ**

  各サブスクリプションは、NETCONF クライアントからのセッションを介した 1 回限りの要求です。Cisco NX-OS

  NETCONF エージェントが応答し、NETCONF クライアントによってセッションが明示的に閉じられるまで、サブスクリプションはアクティブです。サブスクリプションは、スイッチの再起動やスイッチの NETCONF 機能の無効化などの管理アクションによって閉じることもできます。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ればアクティブです。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、構成または動作の変更イベントの通知にサブスクライブできます。そのうちのいくつかとして、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、機能の有効化などがあります。

- **イベント通知の受信**

  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。

- **他の動作とのイベント通知のインターリーブ**

  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 サーバが `<ok/>` 要素で応答した場合、サブスクリプションはアクティブです。

同期の 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。

サーバはサブスクリプションを開始しないため、ユーザーが <create-subscription> の送信操作を含むクライアントプログラムを書く必要があります。

次に、NETCONF クライアントによる <create-subscription> の送信例を示します。

```
<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>
```

<create-subscription> 操作は、次のオプションのどれでもサポートします。

- <stream>クライアントがサブスクライブするイベントのストリームを指定します。ストリームを指定しなかった場合、NETCONF ストリーム内のイベントがデフォルトでクライアントに送信されます。
- <filter>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 サーバは、サブスクリプションを正常に作成できた場合、応答で <ok> メッセージを送り返します。

次に、<create-subscription> を送信した後、クライアントが受信した成功の応答の例を示します。

```
<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） リプレイを使用するサブスクリプションはサポートされていないため、[開始時間（Start Time）] および [終了時間（Stop Time）] オプションは使用できません。

# 通知を受け取る

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、リクエストされたフィルタにマッチするスイッチのすべてのイベントについて、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、インターフェイス「eth1/54/1」がダウンしたときの通知の例を示します。この場合、クライアントは上記の例のインターフェイス operSt に登録しています。

```
<?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>`、イベントが発生した日時を示すタイムスタンプです。
- `<operation>` 、モデルノードのイベントのタイプです。
- `<event>`、クライアントが登録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 クライアントが次のいずれかの方法で Cisco NX-OS NETCONF サーバに特定の要求を送信すると終了します。

- 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 デバイス YANG RPC

## NETCONF のモデル駆動型 RPC について

YANGは、スイッチ構成の操作に加えて、YANGモデルを介してスイッチ内の特定のアクションをトリガーする拡張性を提供します。これは、 CLIを介したEXECモードコマンドの呼び出しに相当します。Cisco NX-OSでは、次のネイティブデバイス RPC が定義されています。

| 操作 | デバイス YANG RPC | CLI | |
|---|---|---|---|
| チェックポイント | checkpoint | **checkpoint** <*name*><br>**checkpoint** <*file*> | 180 |
| ロールバック | ロールバック | **rollback running-config checkpoint** <*name*><br>**rollback running-config checkpoint** <*file*> | 6900 |
| インストールするもの | install_all_nxos | **install all nxos** <*image*> | 1800 |
| | install_add<br>install_activate<br>install_deactivate<br>install_commit<br>install_remove | **install** {**add** \| **activate** \| **deactivate** \| **commit** \| **remove**} <*rpm*> | 180 |
| 暗号化証明書のインポート | import_ca_certificate | **crypto ca import** <*trustpoint*> **pkcs12** <*file*> <*passphrase*> | 5 |
| スイッチのリロードまたはモジュールのリロード | reload | **reload** [**timer** <*seconds*>]<br>**reload module** <*module number*> | 180 |
| ファイルのコピー | copy | **copy** <*source*> <*destination*> | 7200 |
| ファイル/ディレクトリ情報 | dir | dir <*file*> | 180 |
| cli を実行 | cli | <cli command> | 1800 |

# ネイティブデバイス RPC の例

- ファイル名オプションを使用したチェックポイントの作成

```
<rpc
  message-id="checkpoint-3"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">

    <file>bootflash:my_checkpoint2</file>
  </checkpoint>
</rpc>
```

- チェックポイント名、説明を使用したチェックポイントの作成

```
<rpc
  message-id="checkpoint-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

  <checkpoint
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <action>create</action>
    <name>my_checkpoint1</name>
    <description>test checkpoint one</description>
  </checkpoint>
</rpc>
```

- チェックポイント名を使用したチェックポイントの削除

```
<rpc
  message-id="delatecheckpoint-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <action>delete</action>
    <name>my_checkpoint1</name>
  </checkpoint>
</rpc>
```

- ロールバック（**Rollback**）

（注） 次のオプションタグは、atomic（アトミック）、stop-at-first-failure（最初の失敗時に停止）、best-effort（ベストエフォート）として使用できます。

```
<rpc
  message-id="rollback-cfg-option1"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <rollback
   xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <name>my_checkpoint1</name>
      <option>atomic</option>
   </rollback>
</rpc>
```

- ファイルオプションを使用したロールバック

```
<rpc
  message-id="rollback-cfg1"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <rollback
   xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <file>bootflash:my_checkpoint2</file>
   </rollback>
</rpc>
```

- **ファイルのコピー**

  リモートサーバからスイッチストレージ（ブートフラッシュなど）に任意のファイルをコピーします。

```
<rpc
  message-id="copy-file-1"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <copy
   xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <source>tftp://172.27.xxx.xxx//file_location?/tls1-server.pfx
    </source>
      <destination>bootflash:</destination>
      <vrf>management</vrf>
   </copy>
</rpc>
```

- **CA 証明書のインポート**

  前提条件：スイッチで my_truspoint がすでに作成されている必要があります。

```
<rpc
  message-id="import_ca_certificate-1"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <import_ca_certificate
      xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <trustpoint>my_trustpoint</trustpoint>
      <pkcs12>tls1-server.pfx</pkcs12>
      <passphrase>xxxxxx</passphrase>
   </import_ca_certificate>
</rpc>
```

- **RPM パッケージ EXEC RPC コマンドのインストール**

  **Install <add>**

```
<rpc
  message-id="install-add-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <install_add
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <add>rpm_packagenamehere_from_bootflash</add>
  </install_add>
</rpc>
```

  **Install <activate>**

```
<rpc
  message-id="install-activate-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <install_activate
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <activate> rpm_packagenamehere_from_bootflash</activate>
  </install_activate>
</rpc>
```

- **Install <deactivate>**

```
<rpc
  message-id="install-deactivate-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <install_deactivate
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <deactivate>rpm_packagenamehere_from_bootflash </deactivate>
  </install_deactivate>
</rpc>
```

**Install <remove>**

```
<rpc
  message-id="rpc-install_remove-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <install_remove
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <remove>rpm_packagenamehere_from_bootflash </remove>
  </install_remove>
</rpc>
```

**すべての nx-os イメージのインストール**

```
<rpc
  message-id="rpc-install_all_nxos-1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

  <install_all_nxos
    xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <nxos>nxos.image.bin.upg</nxos>

  </install_all_nxos>

</rpc>
```

**モジュール番号のリロード**

```
<rpc
  message-id="reload-module-pyld1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

  <reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">

    <module>29</module>
  </reload>
</rpc>
```

**再読み込み**

（注） クライアントが次の RPC を要求または送信すると、exec コマンドはスイッチのリロードを実行し、それ以上 Netconf クライアントは受信しません。<ok>応答を返します。

```
<rpc
  message-id="563"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
</rpc>
```

**ディレクトリまたはファイル情報**

```
<rpc
  message-id="563"
  xmlns="urn:ietf:params:xml: ns:netconf:base:1.0">
  <dir xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"> bootflash:
  </dir>
</rpc>
```

**スイッチで CLI コマンドを実行します**

```
<rpc
  message-id="563"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <cli xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <mode>CONFIG</mode>
    <cmdline>dir bootflash:</cmdline>
  </cli>
</rpc>
```

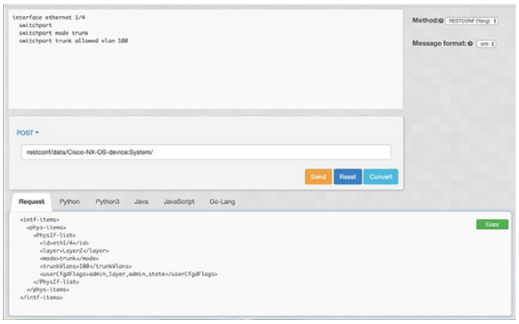
# Netconf クライアントの例

## 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の必要なコマンドを入力し、**[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 : NX-OS* 開発者サンドボックス**




## ncclient を使用した Cisco NX-OS の接続

（注） さまざまな NETCONF クライアントが存在します。このセクションのすべての例では、特定の ncclient python ライブラリを使用しています。

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
```

# 構成データの入手

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())
```

# 実行構成と運用データの取得

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())
```

# 新しい構成の作成

次に、ncclient の edit config を使用して、名前付きの VLAN 100 を作成する方法の例を示します。

```
from ncclient import manager import sys from lxml import etree
device = {
 "address": "nexus",
 "netconf_port": 830, "username": "admin",
 "password": "cisco"
}
def main(): add_vlan = """
 <config>
  <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
   <bd-items>
    <bd-items>
     <BD-list>
       <fabEncap>vlan-100</fabEncap>
      <name>inb_mgmt</name>
     </BD-list>
    </bd-items>
   </bd-items>
  </System>
 </config>
 """
 with manager.connect(host=device["address"], port=device["netconf_port"],
```

```
      username=device["username"],
      password=device["password"],hostkey_verify=False) as m:
  # create vlan with edit_config
  netconf_response = m.edit_config(target="running", config=add_vlan)
  print(netconf_response)
if __name__ == '__main__': sys.exit(main())
```

## 構成の削除

次に、Cisco NX-OS からループバック インターフェイスを削除する例を示します。

```
from ncclient import manager import sys from lxml import etree
device = {
 "address": "nexus",
 "netconf_port": 830, "username": "admin",
 "password": "cisco"
}
def main(): remove_loopback = """
 <config>
  <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
   <intf-items>
    <lb-items>
     <LbRtdIf-list operation="delete"> <id>lo10</id>
     </LbRtdIf-list>
    </lb-items>
   </intf-items>
  </System>
 </config>"""
 with manager.connect(host=device["address"], port=device["netconf_port"],
      username=device["username"], password=device["password"],
      hostkey_verify=False) as m:
  # create vlan with edit_config
  netconf_response = m.edit_config(target="running", config=remove_loopback)
  print(netconf_response)
if __name__ == '__main__':
 sys.exit(main())
```

# NETCONF エージェントのトラブルシューティング

## 機能ステータスの確認

- Cisco NX-OS で、**show feature | inc netconf** コマンドを入力してエージェントの構成を確認します。
- NETCONF エージェントのステータスを表示するには、**show feature** コマンドを使用します。

```
switch-1# show feature | grep netconf
restconf 1 enabled
switch-1#
```

# 接続性の確認

- 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。
- XML 管理インターフェイス（xmlagent とも呼ばれる）というものがあります。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。

# TLS を確認

TLS が使用されている場合、ユーザーは **show netconf internal tls service statistics** コマンドを使用して、TLS のステータスを確認できます。

最初に、サーバの開始時刻を確認します。TLS サーバが実行されているかどうかがわかります。

サーバが実行されていない場合は、次に証明書とクライアントルート証明書をチェックして、証明書が有効かどうかを確認します。

```
# show netconf internal tls service statistics
=============
TLS Service
=============
Port           : 6513
Cert notBefore : Nov  5 16:48:58 2015 GMT
Cert notAfter  : Nov  5 16:48:58 2035 GMT
Client Root Cert notBefore : Oct  1 18:00:24 2020 GMT
Client Root Cert notAfter  : Sep 26 18:00:24 2040 GMT
Server restarts            : 108
Created sessions           : 54
Start                      : 02/23 01:13:10
Run time                   : 142 hour(s) 15 min(s)sec(s)
```

サーバが正常に動作しているように思われる場合は、**show netconf internal tls session all summary** コマンドを使用して、セッションのステータスをさらに確認します。

- サーバが TLC 接続をまったく受信しない場合は、クライアント側の接続をデバッグすることをお勧めします。
- サーバが実際にセッションを受信しているものの、すぐにセッションを拒否する場合は、ユーザー認証を確認することを推奨します。証明書が適切に構成されているか確認します。

```
# show netconf internal tls session all summary
N9k (config)# show netconf internal tls session all summary
=============
TLS Session
=============
* - history
Client                                          Read (KB)   Write (KB)  Status
----------------------------------------------- ----------- ----------- -------------
```

```
*10.28.23.116:35490                                    0.4         1.1 End
*10.28.23.116:35494                                    0.4         1.1 End
--                                                     0.0         0.0 Listening
```

# NETCONF エージェントのアカウンティング ログ

<edit-config>、<commit> または <abort> などの書き込み操作の場合、NETCONF は対応するアカウンティングログを出力します。これには、受信した元の要求と、スイッチに適用された最終的な変更の両方が含まれます。これは、Netconf を経由した構成変更の履歴を確認するのに役立ちます。

アカウンティングログは、**show accounting log** コマンドを使用して表示できます。

例として、次の NETCONF 要求を参照してください。

```
---
<edit-config>
<config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
<System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
 <intf-items>
  <lb-items>
   <LbRtdIf-list>
    <id>lo10</id>
    <descr nc:operation="create">test</descr>
   </LbRtdIf-list>
  </lb-items>
 </intf-items>
</System>
</config>
</edit-config>
---
```

アカウンティング ログには、次の項目が含まれます。

- スイッチに適用された変更

| 項目 | 説明 |
|---|---|
| コンテキスト | セッション ID とユーザー |
| オペレーション | コミット/中止 |
| データベース | 実行または候補 |
| ConfigMO | MO ツリーのテキスト表現。最大 3,000 文字。 |
| ステータス | 成功/失敗 |

例:

```
Wed Jun 29 13:48:03
2022:type=update:id=2496515744:user=admin:cmd=(COMMIT),database=[running],configMo=[
 <topSystem childAction="" dn="sys" status="created,modified"><interfaceEntity
childAction="" rn="intf"
status="created,modified"><l3LbRtdIf childAction="" id="lo10" rn="lb-[lo10]"
status="created,modified"/></interfaceEntity></topSystem>] (SUCCESS)
```

- 受信した元の要求

| 項目 | 説明 |
|---|---|
| コンテキスト | セッション ID とユーザー |
| オペレーション | NETCONF:EDIT-CONFIG、NETCONF:COMMIT、N |
| 送信元 IP（Source IP） | NETCONF クライアント IP |
| ペイロード | 受信したXML要求。最大 3,000 文字。 |
| ステータス | 成功/失敗 |
| tem | 説明 |
| コンテキスト | セッション ID とユーザー |

例:

```
Wed Jun 29 13:48:03
2022:type=update:id=2496515744:user=admin:cmd=(NETCONF:EDITCONFIG),sourceIp=[192.168.1.2],
payload=[<edit-config><config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
<System
xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"><intf-items><lb-items><LbRtdIflist><id>lo10</id>
<descr
nc:operation="remove">test</descr></LbRtdIf-list></lb-items></intfitems></System></config></edit-config>]

(SUCCESS)
```

失敗した要求の場合、失敗のシナリオによっては、ユーザーは両方のログを確認できない場合があります。

- 無効な要求：

  無効な要求は、構成の変更なしに拒否されるため、元の要求のみがログに記録されます。

  例:

  ```
  Wed Jun 29 20:08:36
  2022:type=update:id=2517274784:user=admin:cmd=(NETCONF:EDITCONFIG),
  sourceIp=[192.168.1.2],payload=[<edit-config><config
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <System
  xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"><intf-items><lb-items><LbRtdIflist
   nc:operation="create"><id>lo10</id>
  </LbRtdIf-list></lb-items></intfitems></System></config></edit-config>] (FAILED)
  ```

- さまざまな構成制限による失敗：

  この場合、失敗した構成試行と元の要求の両方がログに記録されます。

  例：

  ```
  Wed Jun 29 20:11:04
  2022:type=update:id=2517274784:user=admin:cmd=(COMMIT),database=[running],
  configMo=[<topSystem childAction="" dn="sys"
  status="created,modified"><telemetryEntity
  childAction=""rn="tm" status="created,modified"><telemetryCertificate
  childAction=""
  filename="foo" hostname="foo" rn="certificate" status="created,modified"
  trustpoint="test"/>
  ```

```
</telemetryEntity></topSystem>] (FAILED)
Wed Jun 29 20:11:04
2022:type=update:id=2517274784:user=admin:cmd=(NETCONF:EDITCONFIG),sourceIp=[192.168.1.2],
payload=[<edit-config><config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
<tm-items><certificateitems><trustpoint>test</trustpoint><hostname>foo</hostname>
<filename>foo</filename></certificateitems></tm-items></System></config></edit-config>]
 (FAILED
```

# NETCONF 明示モードについて

Network Configuration（ネットワーク構成、NETCONF）プロトコルの明示モードは、RFC 6243 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。このプロトコルは、NETCONF サーバからデフォルト構成を読み取るための 3 つの標準モードを定義します。標準モードは、**report-all**、 **trim**、および **explicit** です。

Cisco NX-OS は、**report-all** モードをすでにサポートしています。**report-all** モードを使用して構成を読み取ると、デフォルト構成を含むすべての構成がユーザーに表示されます。この機能で NETCONF クライアントは、**explicit**（明示）モードで構成を読み取ることが付加的に可能です。明示モードでは、 Cisco NX-OSは、ユーザーが明示的に行った構成のみを公開します。

Cisco NX-OS が NETCONF の明示モードでサポートするのは、**<get-config>** と **<edit-config>** RPC だけです。

- **<get-config>**

  明示モードで **<get-config>** は、その値に関係なく、ユーザーによって明示的に設定されたデータを取得します。

- **<edit-config>**

  明示モードでの作成、削除、マージ、置換、移動などの操作は、 **report-all** モードの場合とは少し異なります。

  - クライアントによってスキーマのデフォルト値に設定されたデータノードの **create** 操作属性は、有効であっても、**data-exists** エラータグで失敗します。サーバによってスキーマのデフォルト値に設定されたデータノードの有効な **create** 操作属性は、有効であれば成功します。

  - クライアントによってスキーマのデフォルト値に設定されたデータノードの **delete** 操作属性は、有効であれば成功します。サーバによってスキーマのデフォルト値に設定されたデータノードの **delete** 操作属性は、有効であっても、**data-missing** エラータグで失敗します。

# 注意事項と制約事項

- Cisco NX-OSは、 `get-config` および `edit-config` 以外の操作をサポートしていません。

- この機能は RFC 6243 の部分的なサポートであるため、スイッチは with-default の明示的な機能をアドバタイズしません。
- クリーンに起動されたスイッチからの明示的な get-config 応答が空になることはありません。
- この機能は、複数のスイッチが同時にスイッチにアクセスすることを許可されている場合、予期される応答を保証しません。
- 以前の 10.3(4) からのアップグレードには、次の制限が課されます。
  - インストール ... non-xx
  - インストール ..
- 任意のリリースからのリロードでは、次の制限が課されます。
- reload ascii
- 以前のリリースからのアップグレード
- 以前のリリースからのダウングレード

# NETCONF 明示モードの Get/Set

明示モードを説明するために、次のテレメトリ構成モデルを考えてみましょう。

```
+--rw tm-items
  |   +--rw dest-items
  |   |  +--rw DestGroup-list* [id]
  |   |       +--rw id                   telemetry_IDType
  |   |       +--rw addr-items
  |   |       |  +--rw Dest-list* [addr port]
  |   |       |     +--rw addr      address_Ip
  |   |       |     +--rw port      uint16
  |   |       |     +--rw proto?    telemetry_Protocol      <===== Default Config
  |   |       |     +--rw enc?       telemetry_Encoding    <===== Default Config
  |   |       |     +--rw nodeid?   telemetry_NodeIDorZero
```

### 設定を追加

- 次の edit-config 要求は、id と、宛先アドレスおよびポートを構成します。

```
<edit-config>
    <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

    <target>
        <running/>
    </target>
    <config>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <tm-items>
                <dest-items>
                    <DestGroup-list>
```

```
            <id
              xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
              nc:operation="create">1</id>
                        <addr-items>
                            <Dest-list>
                                <addr>2.2.2.2</addr>
                                <port>2</port>
                            </Dest-list>
                        </addr-items>
                    </DestGroup-list>
                </dest-items>
            </tm-items>
        </System>
    </config>
</edit-config>
-----

Response:

-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <ok/>
</rpc-reply>
```

- 次の report-all get-config 要求は、単にすべてのデータを返します。

```
<get-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">report-all</with-defaults>

  <filter>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
      </tm-items>
    </System>
  </filter>
</get-config>

 Response:

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <adminSt>enabled</adminSt>
        <batchDmeEvt>true</batchDmeEvt>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc>GPB</enc>
                <proto>gRPC</proto>
```

```
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

- 次の明示的な get-config 要求では、構成されたデータのみが返されます。

```
Explicit <get-config> request:

<get-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

  <filter>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items/>
    </System>
  </filter>
</get-config>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

### CLI を使用して構成を追加する

次に、明示モードで NETCONF 要求/応答を構成する手順の例を示します。

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 2.2.2.2 port 2 protocol UDP encoding JSON
```

- 上記の明示的な get-config 要求は、次の応答を返します。

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
```

```
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc>JSON</enc>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

### 構成の削除

```
<edit-config>
  <with-defaults

xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

  <target>
    <running/>
  </target>
  <config>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc
                  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
                  nc:operation="delete">JSON</enc>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </config>
</edit-config>


Response:

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <ok/>
</rpc-reply>
```

- 上記の明示的な get-config 要求は、次の応答を返します。

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>
```

# CLI を使用して構成を追加する

明示モードを構成するには、次の手順を実行します。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **destination-group** *dgrp_id*
4. **ip address** *ip_address* port *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル構成モードを開始します。 |
| ステップ 2 | **telemetry**<br>例：<br>`switch# telemetry` | ストリーミング テレメトリの構成モードに入ります。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 3 | **destination-group** *dgrp_id*<br>例：<br>`switch# destination-group 1` | 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。 |
| ステップ 4 | **ip address** *ip_address* port *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*<br>例：<br>`switch# ip address 2.2.2.2 port 2 protocol UDP encoding JSON` | エンコードされたテレメトリ データを受信する IPv4 IP アドレスとポートを指定します。 |

例

- 上記の明示的な get-config 要求は、次の応答を返します。

```
-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <enc>JSON</enc>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>

-----
```

構成の削除

```
Request:

-----
<edit-config>
    <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults">explicit</with-defaults>

    <target>
        <running/>
    </target>
    <config>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <tm-items>
```

```
                <dest-items>
                    <DestGroup-list>
                        <id>1</id>
                        <addr-items>
                            <Dest-list>
                                <addr>2.2.2.2</addr>
                                <port>2</port>
                            <enc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
 nc:operation="delete">JSON</enc>
                            </Dest-list>
                        </addr-items>
                    </DestGroup-list>
                </dest-items>
            </tm-items>
        </System>
    </config>
</edit-config>

-----

Response:

-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>

-----
```

- 上記の明示的な get-config 要求は、次の応答を返します。

```
-----

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <data>
    <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <tm-items>
        <dest-items>
          <DestGroup-list>
            <id>1</id>
            <addr-items>
              <Dest-list>
                <addr>2.2.2.2</addr>
                <port>2</port>
                <proto>UDP</proto>
              </Dest-list>
            </addr-items>
          </DestGroup-list>
        </dest-items>
      </tm-items>
    </System>
  </data>
</rpc-reply>

-----
```

第 27 章

# RESTCONF エージェント


- RESTCONF エージェントについて （413 ページ）
- 注意事項と制約事項 （414 ページ）
- RESTCONF エージェントの設定 （414 ページ）
- RESTCONF セッションの管理 （415 ページ）
- RESTCONT Get / Set （415 ページ）
- RESTCONF デバイス YANG RPC （417 ページ）
- RESTCONF エージェントのトラブルシューティング （420 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構成するための 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。

**表 *31* : サポートされる操作**

| 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") |
| 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 の RESTCONF は、RESTCONF Protocol というタイトルの RFC ドラフトに基づいています。詳細については、 RESTCONF Protocol draft-ietf-netconf-restconf-10 を参照します。
- Cisco NX-OS リリース 10.4(3)F 以降では、RFC 8040（https://datatracker.ietf.org/doc/html/rfc8040）の次の項目がサポートされています。
  - 3.2 RESTCONF のメディアタイプ
  - 4.2 ヘッド
- TLS については、v1.3 がサポートされています。最低限サポートされるバージョンは v1.2 です。
- Cisco NX-OS リリース 10.4(3)F 以降、RESTCONF は 92348GC-X でサポートされます。

# RESTCONF エージェントの設定

この手順では、RESTCONF を有効にして構成する方法について説明します。

### 始める前に

RESTCONF を使用してスイッチと通信する前に、RESTCONF エージェントを有効にする必要があります。RESTCONF エージェントを有効または無効にするには、**[no] feature restconf** コマンドを入力します。http サーバを実行するには、**feature nxapi** が必要です。

### 手順の概要

1. **configure terminal**
2. （任意） **feature nxapi**
3. （任意） **nxapi http port port-num**
4. （任意） **nxapi https port port-num**
5. **feature restconf**

手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル構成モードを開始します。 |
| ステップ 2 | （任意） **feature nxapi**<br>例：<br>`switch(config)# feature nxapi` | HTTP サーバ用の nxapi を有効にします。 |
| ステップ 3 | （任意） **nxapi http port port-num**<br>例：<br>`switch(conf-tm-sub)# nxapi http port 80` | 使用するポートを指定します。デフォルトのポートは 80 です。 |
| ステップ 4 | （任意） **nxapi https port port-num**<br>例：<br>`switch(conf-tm-sub)# nxapi https port 443` | https に使用するポートを指定します。デフォルトのポートは 443 です。 |
| ステップ 5 | **feature restconf**<br>例：<br>`switch(conf-tm-sensor)# feature restconf` | restconf を有効にします。 |

# RESTCONF セッションの管理

RESTCONF は HTTP 上で実行されます。スイッチ上の NX- API サーバは、管理ポート IP アドレスのポート 80 / 443（または設定されたポート）でリッスンします。クライアントは HTTP サーバとの接続を確立でき、RESTCONF サブシステムは「/restconf」URL にフックします。

クライアントは、RESTCONF エージェントと対話するために継続期間の長い HTTP 接続を維持できますが、通常は単一の要求を送信して終了する方が簡単です。説明については、以下のセクションを参照してください。

# RESTCONT Get / Set

次に、Linux の **curl** コマンドを使用した要求の例を示します。

**<GET>**

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。

次に、 HTTP の Get 要求と応答メッセージの例を示します。

```
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 %
```

### <POST>

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。

次に、 HTTP の POST 要求と応答メッセージの例を示します。

```
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:14750085316974134
Status: 201 Created
Location: /System/bgp-items/inst-items/dom-items/Dom-list=default/always/rtrId/
```

### エフェメラルデータの取得

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。

要求内の RESTCONF 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。

**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
```

サーバは次のように応答します。

```
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", ...
```

# RESTCONF デバイス YANG RPC

## RESTCONF の操作コマンドについて

この機能は、スイッチでモデル駆動型操作コマンドを実行する方法を提供します。

次に、サポートされている実行 RPC のリストを示します。RPC に関する情報は、公開されている Cisco-NX-OS-device.yang ファイルで確認できます。

**表 *32 :* デバイス *YANG RPC***

| 操作 | デバイス YANG RPC | CLI |
|---|---|---|
| チェックポイント | checkpoint | checkpoint <name><br>checkpoint <file> |
| ロールバック | ロールバック | rollback running-config checkpoi<br>rollback running-config checkpoi |

| | | |
|---|---|---|
| インストールするもの | install_all_nxos<br>install_add<br>install_activate<br>install_deactivate<br>install_commit<br>install_remove | すべての nxos をインストールす<br>install {add \|アクティブ化 \|非アク |
| 暗号化証明書のインポート | import_ca_certificate | crypto ca import <trustpoint> pkcs12 |
| スイッチのリロードまたはモジュールのリロード | reload | reload [タイマー<seconds>]<br>モジュールのリロード<module nu |
| ファイルのコピー | copy | コピー<destination><source> |

# デバイス YANG RPC の例

### チェックポイントの作成

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:checkpoint
Accept: application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
"name": "checkpoint-1",
"description": "testing checkpoint through Restconf" }
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

### ロールバック（Rollback）

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:rollback
Accept: application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
"name": "checkpoint-1",
"action": "create”
}
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

### インストールするもの

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:install_all_nxos
Accept: application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
"nxos": "bootflash:nxos.10.1.1-jcco.bin" }
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

### ca 証明書 RPC のインポート

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:import_ca_certificate Accept:
application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
"trustpoint": "mytrustpoint",
"pkcs12": "bootflash:server.pfx",
"passphrase": "mypassphrase"
}
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

### スイッチのリロード

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:reload
Accept: application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
}
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

### モジュールのリロード

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
The client might send the following POST request message:
POST /restconf/operations/Cisco-NX-OS-device:reload
Accept: application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
"module": "31"
```

```
}
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

### ファイル RPC のコピー

クライアントは、次のような POST 要求メッセージを送信する場合があります。

```
POST /restconf/operations/Cisco-NX-OS-device:reload
Accept: application/yang.operation+json,application/yang.errors+json
Content-type: application/yang.operation+json
Body: {
"input": {
"source": "tftp://10.1.1.1/users/myname/config1.txt",
"destination": "bootflash:",
"vrf": "management"
}
}
```

サーバは次のように応答します。

```
HTTP/1.1 204 No content
```

# RESTCONF エージェントのトラブルシューティング

### 機能ステータスの確認

- Cisco NX-OS で、show feature | inc restconf コマンドを入力してエージェントの構成を確認します。
- RESTCONF エージェントのステータスを表示するには、**show feature** コマンドを使用し、式 restconf を含めます。

```
switch-1# show feature | grep restconf
restconf 1 enabled
switch-1#
```

### NX- API 接続の確認

- **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。

### Restconf エラーの確認

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。

- 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：

  **Error Message:** *Sorry, the page you are looking for is currently unavailable*

  - 「接続のトラブルシューティング」に記載されているように、NXAPI 機能が有効になっていること。
  - RESTCONF 機能が有効になっていること（**show feature | grep restconf**）。RESTCONF が有効になっていない場合は、有効にします（**feature restconf**）。
  - ポートは、NX-API によって HTTP または HTTPS 用に構成されます。show nxapi を使用して、ポートが構成されていることを確認します。

    ```
    switch-1# show nxapi
    nxapi enabled
    HTTP Listen on port 80
    HTTPS Listen on port 443
    ```

    ポートが HTTP または HTTPS 用に構成されていない場合は、 **nxapi http port 80** または **nxapi https port 443**を発行して構成します。

- 要求の送信後にこのメッセージを受信した場合（数分後など）、スイッチの最上位レベルからの過剰な同時クエリー要求でシステムが過負荷になっていないことを確認します。過剰な最上位レベルのクエリは、かなりのリソース負荷を生じる可能性があります。

  次のいずれかによって、スイッチが過負荷になっていないことを確認できます。

  - クライアントが送信する要求の数をスロットルバックします。
  - スイッチで **no feature restconf** そして **feature restconf** を発行して RESTCONF エージェントを再起動します。

### RESTCONF エージェントのアカウンティング ログ

POST、PUT、PATCH、またはDELETE などの書き込み操作の場合、RESTCONF は関連するアカウンティングログを出力します。これには、受信した元の要求と、スイッチに適用された最終的な変更の両方が含まれます。

アカウンティングログは、show accounting log コマンドを使用して表示できます。

次の要求例を考えてみます。

```
---
curl -s -L --request POST --user admin: --header 'Content-Type: application/yang.data+json'
 --url “restconf/data/Cisco-NX-OS-device:System/intf-items/lb-items/LbRtdIf-list=lo10”
--data-raw @request.txt
Payload:
<descr>test</descr>
Or
```

```
{"descr":"test"}
---
```

アカウンティング ログには、次の項目が含まれます。

表 *33*：スイッチに適用される変更

| 項目 | 説明 |
|---|---|
| コンテキスト | セッション ID とユーザー |
| オペレーション | コミット/中止 |
| データベース | 実行または候補 |
| ConfigMO | MO ツリーのテキスト表現。最大 3,000 文字。 |
| ステータス | 成功/失敗 |

例:

```
Wed Jun 29 13:53:37
2022:type=update:id=3180018864:user=admin:cmd=(COMMIT),database=[running],configMo=[
<topSystem childAction="" dn="sys" status="created,modified"><interfaceEntity
childAction=""
rn="intf" status="created,modified"><l3LbRtdIf childAction="" descr="test" id="lo10"
rn="lb-
[lo10]" status="created,modified"/></interfaceEntity></topSystem>] (SUCCESS)
```

表 *34*：受信した元の要求

| 項目 | 説明 |
|---|---|
| コンテキスト | セッション ID とユーザー |
| オペレーション | RESTCONF:POST、RESTCONF:PUT、RESTCONF:PATCH、REST |
| 送信元 IP（Source IP） | RESTCONF クライアント IP |
| URL | HTTP URL |
| ペイロード | 受信したXML/JSONリクエスト。最大 3,000 文字。 |
| ステータス | 成功/失敗 |

例:

```
Wed Jun 29 13:53:37
2022:type=update:id=3180018864:user=admin:cmd=(RESTCONF:POST),sourceIp=[192.168.1.2],
url=[/restconf/data/Cisco-NX-OS-device:System/intf-items/lb-items/LbRtdIflist=lo10],payload=[<descr>test</descr>]
 (SUCCESS)
```

失敗した要求の場合、失敗のシナリオによっては、ユーザーは両方のログを確認できない場合があります。

**無効な要求**：

無効な要求は、構成の変更なしに拒否されるため、元の要求のみがログに記録されます。

例:

```
Wed Jun 29 20:16:26
2022:type=update:id=3180018864:user=admin:cmd=(RESTCONF:POST),
sourceIp=[192.168.1.2],url=[/restconf/data/Cisco-NX-OS-device:System/intf-items/lb-items/LbRtdIflist=lo10],payload=[<descr>test</descr>]
 (FAILED)
```

さまざまな構成制限による要求の失敗：

この場合、失敗した構成試行と元の要求の両方がログに記録されます。

例：

```
Wed Jun 29 20:32:01
2022:type=update:id=3180018864:user=admin:cmd=(COMMIT),database=[running],
configMo=[<topSystem childAction="" dn="sys" status="created,modified"><telemetryEntity
childAction="" rn="tm" status="created,modified"><telemetryCertificate childAction=""
filename="foo" hostname="foo" rn="certificate" status="created,modified"
trustpoint="test"/></telemetryEntity></topSystem>] (FAILED)
Wed Jun 29 20:32:01
2022:type=update:id=3180018864:user=admin:cmd=(RESTCONF:PATCH),
sourceIp=[192.168.1.2],url=[/restconf/data/Cisco-NX-OS-device:System/tm-items/certificateitems],payload=[<trustpoint>test</trustpoint><hostname>foo</hostname><filename>
 foo</filename>] (FAILED)
```

第 28 章

# gRPC エージェント


- gRPC エージェントについて （425 ページ）
- gRPC エージェントに関するガイドラインと制限事項 （425 ページ）
- gRPC エージェントの構成 （427 ページ）
- 障害対応 （436 ページ）


## gRPC エージェントについて

gRPC は、最新のオープンソースの高性能なリモートプロシージャコール（Remote Procedure Call、RPC）フレームワークです。Cisco NX-OS は、gNMI や gNOI などの gRPC 関連サービスをサポートする gRPC エージェントを提供します。

## gRPC エージェントに関するガイドラインと制限事項

以下は、gRPC エージェントに関するガイドラインと制限事項です。

- 管理 VRF とデフォルト VRF の両方で gRPC を有効にし、後でデフォルト VRF で無効にすると、管理 VRF の gNMI 操作は機能しなくなります。

  回避策として、**no feature grpc** コマンドを入力して gRPC を完全に無効にし、**feature grpc** コマンド、または **grpc certificate** や **grpc port** のような任意の既存 gRPC 構成コマンドを入力して、再プロビジョニングします。また、管理 VRF の既存の通知に再登録する必要もあります。

- 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。

  **show grpc gnmi service statistics** コマンドを入力して確認します。次のステータスエラーメッセージが表示されます。

  `Status: Not running - Initializing...Port not available or certificate invalid.`（ステータス: 実行していません - 初期化中...ポートが使用できないか、証明書が無効です。）

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。

- 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** コマンドを入力した後には、スイッチをリロードします。スイッチが再び起動したら、gRPC カスタム証明書を再設定する必要があります。

（注） これは、grpc 証明書コマンドを入力した場合に適用されます。

- gRPC のデフォルト以外の VRF の到達可能性は、L3VNI/EVPN および IP 経由でのみサポートされます。ただし、デフォルト以外の VRF および VXLAN フラッドおよびラーニングでの MPLS を介した到達可能性はサポートされていません。
- 9.3(x) より前の Cisco NX-OS リリース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) 以降でサポートされているプラットフォームについては、『Nexus Switch Platform Matrix』を参照してください。
- gRPC プロセスは、CPU 使用率を CPU の 75% に、メモリを 4 GB に制限する HIGH_PRIO 制御グループを使用します。
- gRPC エージェントは、各スイッチ上で、合計で 2 台の gRPC サーバに対し、管理 VRF と 1 台のユーザー指定 VRF をサポートします。ユーザー指定 VRF（たとえばデフォルト VRF）で gRPC をサポートすれば、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 呼び出しの処理を、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。
- Cisco NX-OS リリース 10.4(3)F 以降、gRPC は 92348GC-X でサポートされます。
- Cisco NX-OS リリース 10.4（3）F 以降、TLS は、v1.3 がサポートされています。最低限サポートされるバージョンは v1.2 です。
- Cisco NX-OSリリース 10.5(1)F 以降、 **fips mode enable** または **fips mode disable**を実行するたびに NX- API が再起動します。
- Cisco NX-OS リリース 10.6（1）F 以降、UDS を介した gRPC のパスワードレス認証がサポートされています。この機能は、顧客が外部通信セキュリティを管理し、信頼されたエージェント接続に対する AAA ログイン情報を管理する必要がない場合に役立ちます。

- パスワードレス UDS へのアクセスは、許可されたローカルプロセスに制限されます。ルートとして実行されているか、必要なグループ権限（Unix ファイルのアクセス許可によって決定）を持つプロセスのみが、UDS を介して gRPC エージェントに接続できます。
- パスワードレス UDS での接続のために認証はバイパスされますが、認可とアカウンティングでは依然として有効なユーザ名が必要です。
- gRPC メタデータでユーザー名が指定されていない場合、デフォルトの「管理者」ユーザーが承認とアカウンティングに使用されます。
- 「admin」ユーザーに十分な権限がない場合は、必要なロールを持つ特定のユーザー（network-admin など）を構成します。

# gRPC エージェントの構成

## gRPC の構成

gNMI 機能は、grpc コマンドを使用して構成します。

**grpc certificate**コマンドで使用される証明書をスイッチにインポートするには、『Cisco Nexus 9000 シリーズ NX-OS セキュリティ構成ガイド』の「アイデンティティ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。

（注） インストールされている ID 証明書または と の値を変更すると、gRPC サーバーが再起動して変更が適用される場合があります。**grpc portgrpc certificate** gRPC サーバが再起動すると、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。

### 始める前に

サーバ認証に必要な証明書ファイルを準備し、署名します。

これは gRPC に固有ではないため、既存のトラストポイントファイルを再利用できます。

### 手順の概要

1. **configure terminal**
2. （任意） **crypto ca trustpoint** *<server-trustpoint>*
3. **crypto ca import** *<server-trustpoint>* **pkcs12 bootflash:** *:<server-ca-file> <pkcs-password>*
4. **feature grpc**
5. （任意） **grpc port** *port-id*
6. **grpc certificate** *certificate-id*

7. （任意） **use-vrf default**

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`switch(config)#` | 構成モードに入ります。 |
| ステップ 2 | （任意） **crypto ca trustpoint** *<server-trustpoint>*<br>例：<br>`switch# crypto ca trustpoint tls_server_trustpoint` | サーバ認証用のトラストポイントを作成します。<br>使用可能なサーバトラストポイントがすでに存在する場合、ステップ 2 ～ 3 はオプションです。 |
| ステップ 3 | **crypto ca import** *<server-trustpoint>* **pkcs12 bootflash:** *:<server-ca-file> <pkcs-password>*<br>例：<br>`switch# crypto ca import tls_server_trustpoint pkcs12 bootflash:server.pfx test` | サーバの pkcs12 ファイルをトラストポイントにインポートします。 |
| ステップ 4 | **feature grpc**<br>例：<br>`switch# feature grpc`<br>`switch(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 5 | （任意） **grpc port** *port-id*<br>例：<br>`switch(config)# grpc port 50051` | ポート番号を構成します。port-id の範囲は 1024 ～ 65535 です。50051 がデフォルトです。 |
| ステップ 6 | **grpc certificate** *certificate-id*<br>例：<br>`switch(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 シリーズ NX-OS セキュリティ構成ガイド』の「アイデンティティ証明書のインストール」セクションで、証明書のインポートについて確認してください。 |
| ステップ 7 | （任意） **use-vrf default**<br>例：<br>`switch(config)# grpc use-vrf default` | 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。 |

## キー/証明書の生成

次に、スイッチの bash シェルで自己署名キー/証明書を生成する例を示します。これは実験のみを目的としています。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 シリーズ NX-OS セキュリティ構成ガイド』の「アイデンティティ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。

（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。

### 手順の概要

1. 自己署名キーと pem ファイルを生成します。
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。
3. pkcs12 バンドルをトラストポイントに入力して、トラストポイント CA アソシエーションを設定します。
4. セットアップを確認します。

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | 自己署名キーと pem ファイルを生成します。 | `switch# run bash sudo su`<br>`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。 | `After generating the key and pem files, you must bundle the key and pem files for use in the trustpoint CA Association.`<br>`switch# run bash sudo su bash-4.3# cd /bootflash/`<br>`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!`<br>`bash-4.3# exit` |
| ステップ 3 | pkcs12 バンドルをトラストポイントに入力して、トラストポイント CA アソシエーションを設定します。 | `switch(config)# crypto ca trustpoint mytrustpoint`<br>`switch(config-trustpoint)# crypto ca import mytrustpoint pkcs12 self_sign2048.pfx Ciscolab123!` |
| ステップ 4 | セットアップを確認します。 | `switch(config)# show crypto ca certificates`<br>`Trustpoint: mytrustpoint certificate:`<br>`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`<br>`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` |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | serial=0413<br>notBefore=Nov 5 16:48:58 2015 GMT<br>notAfter=Nov 5 16:48:58 2035 GMT<br>SHA1<br>Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E<br>purposes: sslserver sslclient<br>CA certificate 0: subject= /C=US/O=Cisco Systems,<br>Inc./OU=CSG/L=San Jose/ST=CA/street=3700 Cisco<br>Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R<br>issuer= /C=US/O=Cisco Systems, Inc./OU=CSG/L=San<br>Jose/ST=CA/street=3700 Cisco<br>Way/postalCode=95134/CN=ems.cisco.com/serialNumber=FGE18420K0R<br>serial=0413<br>notBefore=Nov 5 16:48:58 2015 GMT notAfter=Nov 5<br>16:48:58 2035 GMT<br>SHA1<br>Fingerprint=2E:99:2C:CE:2F:C3:B4:EC:C7:E2:52:3A:19:A2:10:D0:54:CA:79:3E<br>purposes: sslserver sslclient |

# gRPC クライアント証明書認証の構成

gRPC では、証明書ファイル（公開キー）に基づいてクライアントを認証することもできます。これにより、パスワードベースの認証よりも安全であると考えられるパスワードレス認証が提供されます。

### 始める前に

サーバ認証に必要な証明書ファイルを準備し、署名します。

これは gRPC に固有ではないため、既存のトラストポイントファイルを再利用できます。

### 手順の概要

1. **configure terminal**
2. （任意） **crypto ca trustpoint** <*server-trustpoint*>
3. **rsakeypair** <*client-key*>
4. （任意） **crypto ca authenticate** <*client-root-trustpoint*>
5. **grpc client root certificate** <*client-root-trustpoint*>

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`switch(config)#` | 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 2 | （任意） **crypto ca trustpoint** <*server-trustpoint*><br>例：<br>`switch# crypto ca trustpoint tls_server_trustpoint` | サーバ認証用のトラストポイントを作成します。<br>使用可能なサーバ トラストポイントがすでに存在する場合、ステップ 2 ～ 3 はオプションです。 |
| ステップ 3 | **rsakeypair** <*client-key*><br>例：<br>`switch# rsakeypar client-key` | クライアント トラストポイントの rsa キーペアを生成します。 |
| ステップ 4 | （任意） **crypto ca authenticate** <*client-root-trustpoint*><br>例：<br>`switch# crypto ca authenticate client_trustpoint` | クライアント証明書をインポートします。この手順では、手動でコピーして貼り付ける必要があります。指示に従ってください。 |
| ステップ 5 | **grpc client root certificate** <*client-root-trustpoint*><br>例：<br>`switch(config)# grpc client root certificate client_trustpoint` | クライアント CA ルート証明書をホストするトラストポイントを入力します。 |

### 例

#### 構成例

このセクションでは、説明のために構成シーケンスの例を示します。

1. クライアント ルートCA証明書を準備します。
2. 証明書のインポート

   クライアント root に対する新しい証明書が正常に生成されたときの、スイッチで証明書を構成するためのコマンド例とその出力を次に示します。

```
switch(config)# crypto ca trustpoint my_client_trustpoint
switch(config-trustpoint)# crypto ca authenticate my_client_trustpoint
input (cut & paste) CA certificate (chain) in PEM format; end the input with a line
 containing only END OF INPUT :
-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJAJLisBKCGjQOMA0GCSqGSIb3DQEBCwUAMD0xCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJDQTERMA8GA1UEBwwIU2FuIEpvc2UxDjAMBgNVBAoM
BUNpc2NvMB4XDTIwMTAxNDIwNTYyN1oXDTQwMTAwOTIwNTYyN1owPTELMAkGA1UE
BhMCVVMxCzAJBgNVBAgMAkNBMREwDwYDVQQHDAhTYW4gSm9zZTEOMAwGA1UECgwF
Q2lzY28wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDEX7qZ2EdogZU4
EW0NSpB3EjY0nSlFLOw/iLKSXfIiQJD0Qhaw16fDnnYZj6vzWEa0ls8canqHCXQl
gUyxFOdGDXa6neQFTqLowSA6UCSQA+eenN2PIpMOjfdFpaPiHu3mmcTI1xP39Ti3
/y548NNORSepApBNkZ1rJSB6Cu9AIFMZgrZXFqDKBGSUOf/CPnvIDZeLcun+zpUu
CxJLA76Et4buPMysuRqMGHIX8CYw8MtjmuCuCTHXNN31ghhgpFxfrW/69pykjU3R
YOrwlSUkvYQhtefHuTHBmqym7MFoBEchwrlC5YTduDzmOvtkhsmpogRe3BiIBx45
AnZdtdi1AgMBAAGjUzBRMB0GA1UdDgQWBBSh3IqRrm+mtB5GNsoLXFb3bAVg5TAf
BgNVHSMEGDAWgBSh3IqRrm+mtB5GNsoLXFb3bAVg5TAPBgNVHRMBAf8EBTADAQH/
MA0GCSqGSIb3DQEBCwUAA4IBAQAZ4Fpc6lRKzBGJQ/7oK1FNcTX/YXkneXDk7Zrj
8W0RS0Khxgke97d2Cwl5P5reXO27kvXsnsz/VZn7JYGUvGSlxTlcCb6x6wNBr4Qr
t9qDBu+LykwqNOFe4VCAv6e4cMXNbH2wHBVS/NSoWnM2FGZ10VppjEGFm6OM+N6z
8n4/rWslfWFbn7T7xHH+Nl0Ffc+8q8h37opyCnb0ILj+a4rnyus8xXJPQb05DfJe
ahPNfdEsXKDOWkrSDtmKwtWDqdtjSQC4xioKHoshnNgWBJbovPlMQ64UrajBycwV
```

```
z9snWBm6p9SdTsV92YwF1tRGUqpcI9olsBgH7FUVU1hmHDWE
-----END CERTIFICATE----END OF INPUT
Fingerprint(s): SHA1
Fingerprint=0A:61:F8:40:A0:1A:C7:AF:F2:F7:D9:C7:12:AE:29:15:52:9D:D2:AE
Do you accept this certificate? [yes/no]:yes switch(config)#
NOTE: Use the CA Certificate from the .pem file content.
switch# show crypto ca certificates Trustpoint: my_client_trustpoint CA certificate
 0:
subject=C = US, ST = CA, L = San Jose, O = Cisco
issuer=C = US, ST = CA, L = San Jose, O = Cisco
serial=B7E30B8F4168FB87 notBefore=Oct 1 17:29:47 2020 GMT notAfter=Sep 26 17:29:47
2040 GMT
SHA1 Fingerprint=E4:91:4E:D4:41:D2:7D:C0:5A:E8:F7:2D:32:81:B3:37:94:68:89:10 purposes:
 sslserver sslclient
```

3. gRPC へのトラストポイントの関連付け

   クライアントルートに新しい証明書を正常に構成した後、スイッチ上でトラストポイントを gRPC サーバに関連付ける出力例を次に示します。

```
switch(config)# feature grpc
switch(config)# grpc client root certificate my_client_trustpoint switch(config)#
show run grpc
!Command: show running-config grpc
!Running configuration last done at: Wed Dec 16 20:18:35 2020
!Time: Wed Dec 16 20:18:40 2020
version 10.1(1) Bios:version N/A feature grpc
grpc gnmi max-concurrent-calls 14 grpc use-vrf default grpc certificate my_trustpoint
grpc client root certificate my_client_trustpoint grpc port 50003
```

4. 証明書の詳細の検証

   スイッチの gRPC にトラストポイントを正常に関連付けられた場合の、証明書の詳細を検証するための出力例を次に示します。

```
switch# show grpc gnmi service statistics
============= gRPC Endpoint =============
Vrf : management
Server address : [::]:50003
Cert notBefore : Mar 13 19:05:24 2020 GMT
Cert notAfter : Nov 20 19:05:24 2033 GMT
Client Root Cert notBefore : Oct 1 17:29:47 2020 GMT
Client Root Cert notAfter : Sep 26 17:29:47 2040 GMT
...
```

5. 任意の gNMI クライアントのクライアント証明書認証を使用した接続の確認。

   クライアント証明書は、秘密キー（pkey）と CA チェーン（cchain）を使用して要求を行います。現在では、パスワードはオプションです。クライアントがルート CAからクライアント証明書への完全なチェーンを提供する必要があることを確認してください。

6. gRPC からトラストポイント参照を削除するには（no コマンド）、次のコマンドを使用します。

```
switch(config)# no grpc client root certificate my_client_trustpoint
```

   コマンドは、gRPCエージェントのトラストポイント参照だけを削除します。トラストポイント CA 証明書は削除されません。スイッチ上の gRPC サーバーへのクラ

イアント証明書認証を使用する接続は確立されませんが、ユーザー名とパスワードによる基本認証は通過します。

# gRPC クライアント証明書認証の構成

これには「pkcs7」ファイルをトラストポイントにインポートする必要があります。そのようなファイルには、関連するすべてのルート CA、中間 CA、サブ CA が含まれること。

（注） クライアント認証用のルート証明書を構成または削除すると、gRPC がサーバーを再起動します。

### 始める前に

クライアント認証に必要な証明書ファイルを準備して署名します。これは gRPC に固有のものではないため、既存のトラストポイントファイルを再利用できます。

### 手順の概要

1. **configure terminal**
2. （オプション）**crypto ca trustpoint** *<client-root-trustpoint>*
3. （オプション）**crypto ca import** *<client-root-trustpoint>* **pkcs7** *bootflash:<p7b file>*
4. **grpc client root certificate** *<client-root-trustpoint>*

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`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 | （オプション）**crypto ca trustpoint** *<client-root-trustpoint>*<br>例：<br>`switch# crypto ca trustpoint client_trustpoint` | クライアント認証用のトラストポイントを作成します。 |
| ステップ 3 | （オプション）**crypto ca import** *<client-root-trustpoint>* **pkcs7** *bootflash:<p7b file>*<br>例：<br>`switch# crypto ca import client_trustpoint bootflash:my_ca_bundle.p7b` | CA バンドルのインポートp7b ファイルが入力で、これらは証明書を含みます。これにより、自動的に複数の子トラストポイントが作成され、それぞれが一意の証明書チェーンを表します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 4 | **grpc client root certificate** <*client-root-trustpoint*><br>例：<br>`switch(config)# grpc client root certificate client_trustpoint` | メイン トラストポイントを gRPC に関連付けます。 |

# GRPC 向けの NGINX プロキシの構成

NetconfやRestconfと同様に、gRPCエージェントは専用のサーバ/ポートで実行されます。gRPCクライアントは、gRPC エージェント/サーバに直接接続する必要があります。

リリース10.3(3)F 以降、NX-OS NGINX は gRPCトラフィックをリレーすることで GRPCプロキシとしても機能できます。これは特定のユースケースに役立ちます。

- GRPC ポートがブロックされました：GRPC エージェントはポート 50051 でリッスンします。このポートがファイアウォールによってブロックされている場合、GRPC クライアントは NGINX HTTPS ポート 443 を介して gRPC サービスに間接的にアクセスできます。
- VRF サポートの強化：現在、GRPC サービスには、管理 VRF または 1 つのユーザー指定 VRF を経由してのみアクセスできます。NGINXプロキシは、任意の VRF からの gRPC 要求を転送できます。

この新しいサポートは、既存の動作には影響しません。GRPCクライアントは、引き続き GRPC エージェントに直接接続できます。代わりにNGINXサーバに接続することもできます。NGINX サーバは、プロキシとして、GRPC 要求を GRPCエージェントに送ります。このようなリダイレクトを行うと、追加の要求応答遅延が発生したと見なされることに注意してください。

すべてのサーバーとクライアントの認証は、NGINX によって処理されます。GRPC を有効にして、NGINX サーバー証明書やクライアント証明書を構成するだけで十分です。

### 始める前に

grpc 機能を有効にします。

NX- API 証明書を準備します。詳細については、「NX-API CLI の使用」を参照してください。

### 手順の概要

1. **configure terminal**
2. **feature nxapi**
3. **nxapi certificate httpscrt certfile** *cert-file*
4. **nxapi certificate httpscrt keyfile** *key-file* **password** <*password*>
5. **nxapi certificate enable**
6. （任意） **crypto ca trustpoint** <*trustpoint*>
7. （任意） **crypto ca authenticate** <*trustpoint*>
8. （任意） **nxapi client certificate authentication**

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`switch(config)#` | 構成モードに入ります。 |
| ステップ 2 | **feature nxapi**<br>例：<br>`switch# feature nxapi`<br>`switch(config)#` | nxapi 機能を有効にします。 |
| ステップ 3 | **nxapi certificate httpscrt certfile** *cert-file*<br>例：<br>`switch# nxapi certificate httpscrt certfile bootflash:nxapi.crt` | 証明書ファイルを構成します。 |
| ステップ 4 | **nxapi certificate httpscrt keyfile** *key-file* **password** *<password>*<br>例：<br>`switch# nxapi certificate httpskey keyfile bootflash:nxapi.key password cisco123` | キーファイルを構成します。 |
| ステップ 5 | **nxapi certificate enable**<br>例：<br>`switch# nxapi certificate enable` | 証明書認証を有効にします。 |
| ステップ 6 | （任意） **crypto ca trustpoint** *<trustpoint>*<br>例：<br>`switch# crypto ca trustpoint grpcClientCA` | サーバ認証用のトラストポイントを作成します。 |
| ステップ 7 | （任意） **crypto ca authenticate** *<trustpoint>*<br>例：<br>`switch# crypto ca authenticate grpcClientCA` | クライアントルート証明書をトラストポイントにインポートします。 |
| ステップ 8 | （任意） **nxapi client certificate authentication**<br>例：<br>`switch# nxapi client certificate authentication` | クライアント証明書認証を有効にします。 |

# 障害対応

## 機能ステータスの確認

- Cisco NX-OS デバイスで、**show feature grpc** コマンドを入力してエージェントの構成を確認します。
- gRPCエージェントのステータスを表示するには、 **show feature** コマンドを使用します。

```
switch-1# show feature | grep grpc
restconf 1 enabled
switch-1#
```

## 接続性の確認

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。

第 29 章

# gNMI：管理インターフェイス


- gNMI について （437 ページ）
- gNMI に関する注意事項と制限事項 （439 ページ）
- gNMI の構成 （442 ページ）
- gNMI RPC （444 ページ）
- Get について （445 ページ）
- Set について （447 ページ）
- 登録について （450 ページ）
- カスタム Syslog ストリームの登録について （454 ページ）
- gNMI のトラブルシューティング （458 ページ）
- gRPC コマンドの構成例 （461 ページ）
- デバッグ ログの収集 （463 ページ）
- gNMI のアカウンティング ログ （463 ページ）


## gNMI について

gNMI（gRPC ネットワーク管理インターフェイス、gRPC Network Management Interface）は gRPC 上で構成されます。gNMI を構成すると、ネットワークデバイスの構成および動作状態の編集、読み取りに役立ちます。また、ネットワークデバイスは、指定されたデータ収集システムへのテレメトリストリームを生成できます。

（注） gNMI は、gRPC 機能の子サービス機能としてのみ構成できます。

Cisco NX-OSは、次の表に示すすべての gNMI RPC をサポートします。

表 *35*：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 |

RPC を登録するために、 Cisco NX-OSは次のサブモードをサポートしています。

表 *36*：サブスクライブオプション

| タイプ | サブタイプ | サポート対象 | 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。<br>デフォルトのサンプル間隔は 10 秒です。 |

| タイプ | サブタイプ | サポート対象 | 説明 |
|---|---|---|---|
| | [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、値の更新は、指定されたパスで作成、変更、または削除などの変更が発生した場合にのみ行われます。 |

gNMIサブスクリプションは、一般にダイヤルインテレメトリと呼ばれます。gNMI では、ネットワークデバイスで要求を開始する外部の顧客を必要とします。

Cisco NX-OSは、ネットワーキングデバイスが外部受信者にテレメトリデータをプッシュする、別のテレメトリ機能をサポートしています。これはダイヤルアウトテレメトリと呼ばれます。詳細については、「テレメトリ」のセクションを参照してください。

Cisco NX-OS リリース 10.4(3)F 以降、OpenConfig パスの openconfig-system:/system/processes は、On-change gnmi サブスクリプションをサポートします。このパスは、gnmi-proto と gnmi-json の両方のエンコーディングをサポートします。サブスクリプションは、クライアントが同じパスのサブスクリプションを解除するまでアクティブなままです。

効率を高めるために、可能な限り厳密なサブスクリプションを作成することをお勧めします。たとえば、cpu-usage-user をモニターにする場合、/system/processes などの一般的なパスに On-target gnmi サブスクリプションを使用するのではなく、データを直接登録すれば、イベントデータを回避できます。

# gNMI に関する注意事項と制限事項

gNMI に関するガイドラインと制限事項は次のとおりです。

- 次のような既存の CLI 構成を使用して OpenConfig ルーティングポリシーをサブスクライブしようとすると、OpenConfig モデルの現在の実装により、空の値が返されます。

  ```
  ip prefix-list bgp_v4_drop seq 5 deny 125.2.0.0/16 le 32
   ipv6 prefix-list bgp_v6_drop seq 5 deny cafe:125:2::/48 le 128
  ```

  以下のパスを使用します。

  ```
  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
  ```

- 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。

- Cisco NX-OS 9.3(x) リリースより前にサポートされるプラットフォームについては、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) 以降でサポートされているプラットフォームについては、『Nexus Switch Platform Support Matrix』を参照してください。

- この機能は、JSON および gNMI proto エンコーディングをサポートします。
- この機能は、protobuf any エンコーディングをサポートしていません。
- この機能はサブスクリプション要求の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。

### ワイルドカードパス

- パスではマルチレベルワイルドカード「...」は使用できません。
- パスの先頭にワイルドカード「*」を使用することはできません
- キー名でワイルドカード「*」を使用することはできません
- キーでのワイルドカードと値は互換性がありません

**show grpc gnmi** コマンドには、次のガイドラインと制限事項があります。

- このコマンドは、xml または json 出力フォーマットをサポートしていません。
- 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。
- 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。

表 *37 : gNMI* 要求のワイルドカード サポート

| リクエストの種類 | ワイルドカード サポート |
|---|---|
| gNMI GET | はい |
| gNMI 設定 | いいえ |
| gNMI サブスクリプション（1 回） | はい |
| gNMI SUBSCRIBE、POLL | はい |
| gNMI SUBSCRIBE、STREAM、SAMPLE | はい |
| gNMI SUBSCRIBE、STREAM、TARGET_DEFINED | はい |
| gNMI SUBSCRIBE、STREAM、ON_CHANGE | いいえ |

### スケールの考慮事項

- 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を超えると、収集されたデータはドロップされます。これは、gNMI ON_CHANGE モードにのみ適用されます。

  収集データのドロップを回避するには、小規模できめ細かいデータ収集セットを管理する、対象を絞ったサブスクリプションを作成してください。したがって、1 つの上位レベ

ルのパスに対するサブスクリプションの代わりに、パスの異なる下位レベルの部分に対する複数のサブスクリプションを作成します。

すべてのサブスクリプションで、最大 250K 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。

- すべてのサブスクリプションに対し、最大 250K 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、収集データのドロップに影響が出ます。

### gNMI サブスクリプションに関するガイドラインと制限事項

- 変更時サブスクリプションは gnmi とテレメトリの両方で機能しますが、一度にアクティブにできるのはいずれか 1 つだけです。これらのエージェントのいずれかからサブスクリプションが作成された場合、既存のサブスクリプション情報は上書きされます。

  次に、openconfig-system:/system/processes の変更時 gnmiサブスクリプションの例を示します。

```
Request:
./gnmi-console_enhanced_plus --host 172.22.244.142 --port 50051 -u admin -p insieme
 --tls --cafile /tmp/grpc.pem --hostnameoverride ems.cisco.com  --operation=Subscribe
 --submode ON_CHANGE  --xpath "openconfig-system:system/processes/process[pid=1]"
-e JSON

Response:

////// initial snapshot

Received response 1 --------------------------
/system/
{
    "processes": {
        "process": [
            {
                "pid": "1",
                "state": {
                    "pid": "1",
                    "name": "init",
                    "start-time": "1706643350384761800",
                    "cpu-usage-user": "14",
                    "cpu-usage-system": "12",
                    "cpu-utilization": 0,
                    "memory-usage": "180224",
                    "memory-utilization": 0
                }
            }
        ]
    }
}

///// first event update

/system/
{
    "processes": {
        "process": [
            {
                "pid": "1",
                "state": {
                    "start-time": "1706643350384761800",
```

```
                    "cpu-usage-user": "14",
                    "cpu-usage-system": "12"
                }
            }
        ]
    }
}
```

# gNMI の構成

gNMI 機能は、gRPC gNMI コマンド を使用して構成します。

# gNMI オプションの構成

### 始める前に

gNMI は、gRPC 機能のみの子サービス機能です。

gRPC エージェントを有効にする方法については、gPRC エージェントのドキュメントを参照してください。

### 手順の概要

1. switch# **configure terminal**
2. （任意） **grpc gnmi max-cocrrent-call**
3. **grpc gnmi subscription target-defined min-interval**<*interval*>
4. **grpc gnmi subscription query-condition keep-data-timestamp**
5. （任意） **grpc gnmi keepalive-timeout** <*timeout*>

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | switch# **configure terminal**<br>例：<br>`switch# config terminal`<br>`switch(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 | （任意） **grpc gnmi max-cocrrent-call**<br>例：<br>`switch(config)# grpc gnmi max-concurrent-call 16` | このコマンドは、スイッチ上の gNMI サーバーに対する同時ダイヤルイン呼び出しの制限を設定します。デフォルトの制限は 8 です。値は、1 ～ 16 の制限範囲で設定できます。<br>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 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | の両方に構成されている場合、各 VRF は 16 の同時 gNMI 呼び出しをサポートします。<br>このコマンドは、進行中または入力中の gNMI 呼び出しには影響しません。 gRPC は新しい呼び出しに制限を適用し、アクティブな呼び出しにはは影響を与えないため、アクティブな呼び出しは完了することができます。 |
| ステップ 3 | **grpc gnmi subscription target-defined min-interval**<*interval*><br>例：<br>`switch(config)# grpc gnmi subscription target-defined min-interval 60` | ターゲット定義のデフォルトのサンプル間隔を、30 秒から、必要な他の値に変更できます。 |
| ステップ 4 | **grpc gnmi subscription query-condition keep-data-timestamp**<br>例：<br>`switch(config)# grpc gnmi subscription query-condition keep-data-timestamp` | このコマンドは、sample/once/poll サブスクリプションが、データが最後に更新されたときにデータベースからタイムスタンプを取得できるようにします。<br>（注）<br>• このコマンドは PROTO でのみサポートされ、JSON コードではサポートされません。<br>• once、poll、sample をサポートしますが、on_change サブスクリプションはサポートしません。<br>• DME、YANG、および OpenConfig データソースにおいてサポートします。<br>• サポートされていないプロパティの場合、コマンドのデフォルトは、直前のデータベース変更時刻ではなく、収集時刻に戻ります。<br>• このコマンドは、タイムスタンプごとに冗長応答を生成します。時間どおりメッセージを収集できない場合、スイッチはメッセージの収集をドロップします。 |
| ステップ 5 | （任意） **grpc gnmi keepalive-timeout** <*timeout*><br>例：<br>`switch(config)# grpc gnmi keepalive-timeout 1200` | このコマンドを構成すると、非アクティブまたは無許可の接続を削除できます。gRPCエージェントは、クライアントに空の応答を定期的に送信します。この応答がクライアントに届かなかった場合、接続は停止します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | デフォルトの間隔値は 600 秒です。キープアライブインターバルを 600 ～ 86400 秒の間隔範囲で変更するように構成できます。 |

# gNMI RPC

このセクションでは、それぞれの gNMI RPC について説明します。このセクションでは、参照クライアント *gnmi_cli* の使用方法を確認できます。

### 機能

RPC の機能は、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、バージョンデータモデル、およびサーバでサポートされているデータエンコードが含まれます。

### 機能に関する注意事項と制限事項

次は機能に関するガイドラインと制限事項です。

### クライアント出力の例

次に言及している例は、機能のクライアント出力を示しています。機能 *openconfig* が有効になっていることを確認します。

この例は、YANG モデル、gNMIバージョン、およびサポートされているエンコードのサポートを示しています。

```
$ ./gnmi_cli -a 172.1.1.1:50051 -ca_crt ./grpc.pem -insecure -capabilities
supported_models: <
  name: "Cisco-NX-OS-device"
  organization: "Cisco Systems, Inc."
  version: "2019-11-13"
>
supported_models: <
  name: "openconfig-acl"
  organization: "OpenConfig working group"
  version: "1.0.0"
>
supported_models: <
  name: "openconfig-bgp-policy"
  organization: "OpenConfig working group"
  version: "4.0.1"
>
.
.
.
supported_models: <
  name: "openconfig-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"
```

# Get について

`Get RPC` を使用すれば、スイッチからデータツリーのスナップショットを取得できます。1 つの要求で複数のパスを要求できます。gNMI パスの規則に従って、単純な形式の XPATH を使用できます。「gNMI のスキーマパスエンコーディング規則」を参照してください。`GET` 操作の詳細については、『gRPC ネットワーク管理インターフェイス』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。

### Get に関する注意事項と制限事項

次に、Get のガイドラインと制限事項を示します。

- `GetRequest` エンコードは JSONフォーマットのみをサポートします。
- `GetRequest`.type の場合、DataType CONFIG と STATE のみが YANG 形式との直接の関係と式を持ちます。操作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。1 つの要求には 1 つのパスのみが含まれます。
- ルートパス（**すべて**のモデルからのすべてで「/」）に対する `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。Report-all モード RFC 6243 [4]を参照してください。
- `Get` は `Cisco-NX-OS-syslog-oper` モデルをサポートしていません。

- openconfig-procmon データを取得するには、クエリをパス /system/processes または /system に送信します。

（注） Cisco NX-OSリリース 10.3(x) より前では、 /system パスからデータを取得することはできません。

- 以下のオプション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 GetReqeuest で 10 のパスを要求することができます。
- GetResponse のサイズ返り値が 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。
- gRPC の最大受信バッファサイズは 8 MB です。
- より多くの構成があるスイッチに対して Get 操作を実行すると、gRPC プロセスは使用可能なすべてのメモリを消費します。メモリが使い果たされると、次の syslog が生成されます。

```
MTX-API: The memory usage is
        reaching the max memory resource limit (3072) MB
```

  メモリが連続して使い果たされると、次の syslog が生成されます。

```
The process has become unstable and
        the feature should be restarted.
```

  Cisco では、gNMI トランザクションを正常に機能させるため、gRPC 機能を再起動することをお勧めします。
- Get の合計同時セッションの最大数は、構成されている最大同時呼び出し数の 75% です。たとえば、MTX 同時呼び出しが 16 に構成されている場合、Get の合計同時セッションの最大数は 12 になります。
- Get と Set の同時セッションの合計数は、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 同時呼び出しが 16 に構成されている場合、Get と Set の合計同時セッションの最大数は 15 になります。

# Set について

Set RPC を使用して、スイッチの構成を変更できます。スイッチの削除、交換、更新が行えます。1 つの Set 要求に含まれるこれらすべての操作は、1 つのトランザクションと見なされて、すべて正常に完了するか、またはスイッチが元の状態にとどまるかのいずれかになります。

Set 操作は、Set Request 構成で指定された順序で実行されます。パスが複数回要求されている場合、パスが互いを上書きする場合でも、変更は実行されます。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。フィールドの削除、置換、更新など、構成データパスであり、Set 要求で指定されているパスは編集できます。

Set 操作の詳細については、gNMI の仕様の「状態の変更」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 エンコードは JSONフォーマット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。1 つの要求には 1 つのパスのみが含まれます。
- 以下のオプション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 SetReqeuest で 20 のパスを要求できます。
- gRPC の最大受信バッファサイズは 8 MB です。
- Get と Set の合計同時セッションの最大数は、構成されている最大 gNMI 同時呼び出し数です。たとえば、gNMI 同時呼び出しが 16 に構成されている場合、Get と Set の合計同時セッションの最大数は 15 になります。
- スイッチに対して Set::Delete RPC 操作を実行した場合、構成が大規模だと、下に示すような MTX 警告メッセージが生成されます。

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

         restart.
```

# Set の拡張機能

Set RPC では、次の拡張機能がサポートされています。

## 拡張の Confirm-Commit

この拡張機能により、新しい設定によってデバイスへの接続が失われた場合に、適用された設定を自動的にロールバックできます。すべての protobuf メッセージの定義を含む拡張の詳細については、次を参照してください：https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-commit-confirmed.md

gnmi confirm-commit 機能は、既存の netconf confirm-commit の簡素化されたサブセットです。gnmi confirm-commit トランザクションの場合、拡張は単一の **Set** 操作に適用されます。この **Set** 操作は、指定された時間内に確認される必要があります。そうしないと、 設定リクエストの操作が以前の状態にロールバックされます。**Set** 操作を確認するのではなく、明示的にキャンセルすることもできます。

この拡張機能の一般的な使用例は簡単です。gnmi クライアントは通常のように **Set** 操作を使用して構成変更を適用しますが、**Commit** 拡張メッセージも含まれています。**Commit** メッセージには、文字列識別子と、ロールバック期間を設定する **CommitRequest** メッセージが含まれます。このロールバック期間が期限切れになる前に、指定された id 文字列で識別されるこの **CommitRequest** メッセージを確認する必要があります。**CommitRequest** は、データを含めず、拡張機能で **CommitConfirm** メッセージを伝送する別の **Set** 要求を送信することによって確認されます。構成の変更によってネットワーク接続が中断された場合、この確認メッセージはデバイスで受信されず、構成は以前の構成にロールバックされます。クライアントが希望する場合は、構成をすぐにキャンセルしてロールバックするために **CommitCancel** メッセージを送信することもできます。ロールバック期間タイマーを延長するメッセージもあります。CommitConfirm メッセージが正常に送信されて **CommitRequest Set** 操作が確認されると、適用された設定が保持されます。メッセージの詳細については、上記のリファレンスを参照してください。

## Confirm-Commit のガイドラインと制限事項

gnmiconfirmed-commit 機能を使用して設定をプッシュする場合、ユーザーがいくつかの点に留意する必要があります。

- 構成変更の意図と効果を理解する必要がある。たとえば、ユーザーはインターフェイスをシャットダウンしたり、grpc 自体を無効にしたりすることができます。そのような場合、CommitConfirm メッセージを受信できないため、このような場合、confirm-commit 拡張は意味を持ちません。
- 場合によっては、構成の変更により一時的にネットワーク接続が中断され、回復に時間がかかることがあります。この一時的な中断を予測し、確認が行われるときに構成が安定した状態に達するように、適切なタイムアウト期間を選択する必要があります。構成変更に不適切なタイムアウトを選択すると、不要な構成ロールバックが発生する可能性があります。このタイムアウトの選択は、ユーザーの責任です。
- 管理者とオペレータは、構成のロールバックの可能性を認識している必要があります。これは、ログ収集やその他のデバッグ/モニタリングツールに干渉する可能性があるためです。

- CommitRequest メッセージを使用して Set 操作が適用され、「id」が確立されると、他の Set 操作は受け入れられません。アクティブなセット（「id」と一致）は、確認、キャンセル、またはタイムアウトを許可する必要があります。
- CommitConfirm、CommitCancel、または CommitSetRollbackDuration メッセージを伝送する Set オペレーションは、更新/置換/削除オペレーションのデータを伝送することはできません。操作対象のパスを含むセット（トランザクションの初期セット）とともに伝送できるのは、CommitRequestメッセージだけです。
- 1 つの CommitConfirm トランザクションはいつでもアクティブになる可能性があります。
- タイムアウト値の最小単位は、Duration.seconds で指定された秒レベルのみです。指定された期間のナノ秒以外の値がある場合、その期間は拒否されます。

## コマンドの表示

次のコマンドは、gnmi の確認済みコミットトランザクションに関する情報を表示するように拡張されています。以下の例は、Set 要求の後で、Lo1 インターフェイスの説明を変更し、600 秒のタイムアウトで commit-id を「commitId-123」に設定した後です。

- `show grpc gnmi transactions`

  このコマンドは、コミット「id」、タイムアウト期間、メッセージタイプ（CommitRequest、CommitConfirm、CommitCancel など）などの confirm-commit 拡張の gnmi メッセージレベルの詳細を表示します。

```
RPC          DataType   Session         Time In              Duration(ms) Status

------------ ---------- --------------- -------------------- ------------ ------

Set          -          3               06/16 20:02:51       5            0

Commit Type: CommitConfirm   Persist: commitId-123         Timeout :600


Set          -          2               06/16 20:01:41       75           0

subtype: dtx:  st: path:

Replace  -     OK  /System/intf-items/lb-items/LbRtdIf-list[id=lo1]/descr

Commit Type: CommitRequest   Persist: commitId-123         Timeout :600


Set          -          1               06/16 20:00:55       13           0
```

- `show grpc internal mtx commit history`

  このコマンドは、セッション ID やタイムスタンプなどの内部状態情報を表示します。

```
MTX Commit History
```

```
Session    Op      Persist          PersistId        Timeout      Ret    Timestamp

3          COMMIT                   commitId-123                  1      2025-06-16
  20:02:51

2          COMMIT  commitId-123                      600          1      2025-06-16
  20:01:41
```

- `show grpc internal mtx db`

  このコマンドは、DB レベルの内部状態情報を表示します。

```
MTX DME DB state variables
Running Config locked : false
Session that locked Running : 18446744073709551615
Session doing confirmed commit : 18446744073709551615
Is a confirmed commit active : false
Thread Local Data instances : 1
Confirmed Commit start time : 2025-06-16 20:01:41
Confirmed Commit timeout : 600
Confirmed Commit time remaining : 0
Persistent Session id : 3
Commit Candidate to Running :
    Transaction Token : commitId-123
    Commit Type : COMMIT_CONFIRM_END
```

# 登録について

登録 RPC を使用すれば、特定のパスのテレメトリストリームを登録できます。登録されたパスに変更があると、スイッチは設定またはパスの状態の変更に関する通知をにプッシュします。

登録 RPCで使用可能な、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のみ適用されます。このオプションのフラグが設定されている場合、スイッチは現在の状態を抑制し、最初の応答とともに通知が送信されます。

これらのフラグを使用して、次の表に示すオプションへの応答を変更できます。

**表 *38 : SUBSCRIBE* フラグのサポートマトリクス**

| サブスクリプションタイプ | **heartbeat_interval** | **[冗長抑制（suppress_redundant）]** |
|---|---|---|
| ON_CHANGE | 発信元：デバイス YANG、OpenConfig YANG、DME | 該当なし |
| SAMPLE | 発信元：デバイス YANG、OpenConfig YANG、DME | 発信元：デバイス YANG、OpenConfig YANG |

Cisco NX-OS リリース 10.2(3)F 以降、次のオプションフラグがサポートされています。

- heartbeat_interval
- [冗長抑制（suppress_redundant）]

サンプルサブスクリプションでの suppress_redundant の動作を変更するために、heartbeat_interval を指定できます。この場合、ターゲットは、suppress_redundant フラグが true に設定されているかどうかに関係なく、hearbeat_interval ごとに 1 つのテレメトリ更新を生成する必要があります。この値は、ナノ秒単位の符号なし 64 ビット整数として指定します。

sample サブスクリプションの場合は、suppress_redundant フラグを使用できます。この場合、設定ステータスは true です。ターゲットは、レポートされたパスの値が、最後の更新が生成されてから変更されていた場合、テレメトリ更新メッセージを生成します。更新は、サブスクリプションが変更された個々のリーフノードに対して生成されます。

たとえば、サブスクリプション名が「A」と「B」で、「B」ノードからブランチ「C」と「D」が出ていたとします。「D」の値ではなく、「C」の値が変更されたとします。更新は「C」に対してのみ生成され、「D」に対しては生成されません。

### 注意事項と制約事項

次に、Set サブスクリプションのガイドラインと制限事項を示します。

- 次のフラグはサポートされていません。
  - エイリアス
  - allow_aggregation
  - 内線番号
  - prefix
  - QoS（Dest. QoS）
- 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が同じになっていることを確認します。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。
- メモリが不足している状態では、OC サブスクリプション要求はメモリが使用可能になるのを待たずに、すぐに失敗する場合があります。
- システムによって過剰なイベントが生成された場合、OC イベント通知がドロップされる場合があります。
- サブスクリプションスナップショット中はOCイベント通知が無効になり、スナップショットが完了するとフラッシュされます。
- OC スナップショットの完了時間は 30 分に制限されています。その時間内に完了しない場合、サブスクリプションはキャンセルされます。

### gNMI サブスクライブオプション

次の表に、サブスクライブオプションのモードを示します。

表 *39*：

| Type | サブスクリプション タイプ | 説明 |
|---|---|---|
| ONCE | | データを受信するように登録し、セッションを閉じます。 |
| 投票 | | 登録すると、アクティブセッションを保持します。<br>ポーリング要求を発生させる場合は、要求ごとにデータを入力してください。 |
| STREAM | SAMPLE | 特定の頻度でデータを登録し、受信します。<br>ペイロード値の単位はナノ秒です。1 秒 = 1000000000 になります。 |
| | ON_CHANGE | スナップショットを受信し、ツリーに変更がある場合にのみデータを受信するように登録します。 |
| TARGET_DEFINED | | 作成できる最適なサブスクリプションを見つけるように登録します。 |

### モードの設定

それぞれのモードは、サブスクリプション内部とサブスクリプション外部の2つの設定を必要とします。次に、サブスクリプションの組み合わせについて説明します。

- ONCE：SAMPLE、ONCE
- POLL：SAMPLE、POLL
- STREAM：SAMPLE、STREAM
- ON_CHANGE：ON_CHANGE、STREAM
- TARGET_DEFINED：TARGET_DEFINED、STREAM

### Origin

- DME：DME モデルへの登録
- DEVICE：YANG モデルへの登録
- OPENCONFIG：OpenConfig モデルへの登録

### 名前（Name）

- DME：DME モデルへの登録
- Cisco-NX-OS-device：YANG モデルへの登録

### エンコーディング（Encoding）

- JSON：ストリームは JSON 形式で送信される。
- PROTO：ストリームは元のプロトコルフォーマットで送信される。

## ペイロードの構成例

次のセクションでは、ペイロードのクライアントの例を示します。

### DME ストリーム/サンプルへの登録

```
{
    "SubscribeRequest":
    [
        {
            "subscribe":
            {
                "subscription":
                [
                    {
                        "path":
                        {
                            "origin": "DME",
                            "elem":
                            [
                                {
                                    "name": "sys"
                                },
                                {
                                    "name": "bgp"
                                 }
                             ]
                        },
                        "mode": "SAMPLE"
                    }
                ],
                "mode": "ONCE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "name": "DME",
                        "organization": "Cisco Systems, Inc.",
                        "version": "1.0.0"
                    }
                ],
                "encoding": "JSON"
            }
        }
    ]
}
```

### OpenConfig YANG/サンプルへの登録

```
{
    "SubscribeRequest":
    [
        {
            "subscribe":
            {
                "subscription":
                [
                    {
                        "path":
                        {
                            "origin": "openconfig",
                            "elem":
                            [
                                {
                                    "name": "interfaces"
                                }
                            ]
                        },
                        "mode": "SAMPLE",
                        "Sample_interval": 10000000000
                    }
                ],
                "mode": "ONCE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "name": "openconfig-interfaces",
                        "organization": "OpenConfig working group",
                        "version": "0.8.1"
                    }
                ],
                "encoding": "JSON"
            }
        }
    ]
}
```

# カスタム Syslog ストリームの登録について

Cisco NX-OSは、オリジナルのモデルと OpenConfig モデルをサポートします。gNMI の場合、または ON_CHANGE を登録する場合、カスタム YANG モデルは syslog イベントをストリーミングするように設計されています。この機能は、8 GB 以上のメモリを搭載した Cisco Nexus 9000 シリーズスイッチで構成できます。

### 注意事項と制約事項

Syslog ストリームに関するガイドラインと制限事項は次のとおりです。

- 無効な syslog はサポートされません。たとえば、フィルタまたはクエリ条件を含む syslog はサポートされません。
- Syslog ストリームは以下のパスのみをサポートします。

- Cisco-NX-OS-Syslog-oper: syslog
- Cisco-NX-OS-Syslog-oper: syslog messages

- ストリームサンプルモードと POLL モードのみがサポートされます。
- サポートされるエンコーディングフォーマットは JSON と PROTO です。

### オリジナルの YANG Syslog モデル

次に、YANG Syslog モデルの構成例を示します。

（注） デフォルトでは、タイムゾーンフィールドは空です。タイムゾーンフィールドは、`clock format show-timezone syslog` を構成するときに設定されます。

```
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

次に、サブスクライブ要求の例を示します。

```
{
    "SubscribeRequest":
    [
        {
            "subscribe":
            {
                "subscription":
                [
                    {
                        "path":
                        {
                            "origin": "syslog-oper",
                            "elem":
                              [
                                {
                                    "name": "syslog"
                                },
                                {
                                    "name":"messages"
                                }
                            ]
                        },
                        "mode": "ON_CHANGE"
```

```
                    }
                ],
                "mode": "ON_CHANGE",
                "allow_aggregation" : false,
                "use_models":
                [
                    {
                        "name": "Cisco-NX-OS-Syslog-oper",
                        "organization": "Cisco Systems, Inc.",
                        "version": "0.0.0"
                    }
                ],
                "encoding":"JSON"
            }
        }
    ]
}
```

### 応答の例

次に、PROTO エンコードでの出力応答の例を示します。

```
[Subscribe]-------------------------------
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: "time-stamp"
}
}
val {
uint_val: 1574375200000
}
}
update {
path {
elem {
name: "severity"
}
}
val {
uint_val: 5
}
}
}

/Received -------------------------------------
```

次に、JSON エンコードでの出力応答の例を示します。

```
[Subscribe]-------------------------------
### Reading from file ' testing_bl/stream_on_change/OC_SYSLOG.json '

Tue Nov 26 11:47:00 2019
### Generating request : 1 -----------
### Comment : STREAM request
### Delay : 2 sec(s) ...
### Delay : 2 sec(s) DONE
subscribe {
subscription {
path {
origin: "syslog-oper"
elem {
name: "syslog"
}
elem {
name: "messages"
}
}
mode: ON_CHANGE
}
use_models {
name: "Cisco-NX-OS-Syslog-oper"
organization: "Cisco Systems, Inc."
version: "0.0.0"
}
}

Tue Nov 26 11:47:15 2019
Received response 5 --------------------------
update {
timestamp: 1574797636002053000
prefix {
}
update {
path {
origin: "Syslog-oper"
elem {
name: "syslog"
```

```
}
}
val {
json_val: "[ { \"messages\" : [[
{\"message-id\":657},{\"node-name\":\"task-n9k-1\",\"time-stamp\":\"1574797635000\",\"time-of-day\":\"Nov
 26 2019
11:47:15\",\"severity\":3,\"message-name\":\"HDR_L2LEN_ERR\",\"category\":\"ARP\",\"group\":\"ARP\",\"text\":\"arp
 [30318] Received packet with incorrect layer 2 address length (8 bytes), Normal pkt
with S/D MAC: 003a.7d21.d55e ffff.ffff.ffff eff_ifc mgmt0(9), log_ifc mgmt0(9), phy_ifc
 mgmt0(9)\",\"time-zone\":\"\"} ]] } ]"
}
}
}
```


# gNMI のトラブルシューティング

show コマンドを実行して、gNMI のステータスを表示できます。特定の gNMI 構成を確認するには、次の表に記載されているコマンドを入力します。

**表 *40 : show* コマンド**

| コマンド | 説明 |
|---|---|
| `show grpc gnmi service statistics` | それぞれの管理 VRF、またはデフォルト構成 VRF のエージェント実行ステータスのサマリーを表示します。また、次の項目も表示されます。<br>• 基本の全般的なカウンタ<br>• 証明書の有効期限日時 |
| `show grpc gnmi rpc summary` | 次のステータスが表示されます。<br>• 受信した機能 RPC の数。<br>• RPC の機能のエラー。<br>• 受信した Get RPC の数。<br>• Get RPC エラー。<br>• 受信した Set RPC の数。<br>• Set RPC エラー。 |

| コマンド | 説明 |
| --- | --- |
| `show grpc gnmi transactions` | |

| コマンド | 説明 |
| --- | --- |
| | show grpc gnmi transaction コマンドは最も密度が高く、かなり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。<br><br>このコマンドは、詳細情報を表示します。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を表示します。新しい RPC が入力されると、最も古い入力が履歴から順に削除されます。次は表示される情報を示しています。<br><br>• RPC：受信した RPC のタイプ（Get、Set、機能）を示します。<br>• 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。<br>• 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。<br>• 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。<br>• 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）。<br>• 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 コードを示します（0 は成功、0 以外はエラー）。<br><br>次は、単一の gNMI トランザクション内のパスごとのデータを示します。たとえば、単一の Get または Set です。<br><br>• **サブタイプ（subtype）**：Set RPC の場合に、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。 |

| コマンド | 説明 |
| --- | --- |
| | • **Dtx**：このパスが DTX ファストパスで処理されるかどうかを示します。ダッシュ（-）はいいえを示し、アスタリスク（*）ははいを示します。<br>• **st**：このパスのステータスを表示します。表示されるステータスには次のものがあります。<br>  • **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。<br>  • **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<br>  • **--**：パスはまだ処理されていません。または無効なため、インフラに送信されていません。 |

# gRPC コマンドの構成例

### gRPC gNMI サービス統計

次に、`show grpc gnmi service statistics` コマンドの出力例を示します。

```
=============
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
```

### gRPC gNMI rPC のサマリー

次に、show grpc gnmi service statistics コマンドの出力例を示します。

```
=============
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
```

### gRPC gNMI トランザクション

次に、show grpc gnmi transaction コマンドの出力例を示します。

```
=============
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          -          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=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          -          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]
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=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

...

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
```

# デバッグ ログの収集

gNOI は、gRPC エージェントの子サービスです。詳細については、「診断と有用性（605 ページ）」を参照してください。

# gNMI のアカウンティング ログ

gNMI では、`Set RPC` はスイッチの構成を変更します。`SET` で構成の更新（`UPDATE`）、置き換え（`REPLACE`）、または削除（`DELETE`）をリクエストした場合、gNMI は対応するアカウンティングログを生成します。これらのログには、元のリクエストとスイッチで行われた変更の両方が含まれます。

アカウンティングログは、`show accouting log` コマンドを使用して表示できます。

次は、次の gNMI パスを使用する、`SetRequest`, encoding = JSON to localhost の例を示しています。

```
---
<<<<<<< set_delete >>>>>>>
[]
<<<<<<< set_replace >>>>>>>
[] []
<<<<<<< set_update >>>>>>>
[elem { name: "System"
} elem {
name: "tm-items"
} elem {
```

```
name: "certificate-items"
}
] [json_val: "{\"hostname\": \"test\", \"trustpoint\": \"foo\"}"
]
The SetRequest response is below ------------------------response { path {
elem {
name: "System"
} elem {
name: "tm-items"
} elem {
name: "certificate-items"
}
} op: UPDATE
} timestamp: 1656512303065384369
---
```

次の表に、スイッチで構成できるオプションを示します。アカウンティングログには次の項目が含まれます。

| 項目 | 説明 |
|---|---|
| コンテキスト | セッション ID とユーザー |
| オペレーション | コミットまたは中止 |
| データベース | 実行または候補 |
| ConfigMO | MO ツリーのテキスト表現。最大 3000 文字。 |
| ステータス | 成功または失敗 |

次にアカウンティングログのサンプルを示します。

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(COMMIT),database=[candidate],
configMo=[<topSystem childAction="" dn="sys" status="created,modified"><telemetryEntity

childAction="" rn="tm" status="created,modified"><telemetryCertificate childAction=""
hostname="test" rn="certificate" status="created,modified"
trustpoint="foo"/></telemetryEntity></topSystem>] (SUCCESS)
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(COMMIT:CANDIDATE-TO-RUNNING),
database=[running] (SUCCESS)
```

次の表に、スイッチで受信した元のリクエストを示します。

| 項目 | 説明 |
|---|---|
| コンテキスト | セッション ID とユーザー |
| オペレーション | gNMI:SET:UPDATE,<br>gNMI:SET:REPLACE,<br>gNMI:SET:DELETE,<br>COMMIT:CANDIDATE-TO-RUNNING |
| 送信元 IP（Source IP） | gNMI クライアント IP |

| 項目 | 説明 |
|---|---|
| パス | テキスト フォーマットの gNMI パス |
| ペイロード | 受け取った JSON 要求。最大 3000 文字。 |
| ステータス | 成功または失敗 |

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(GNMI:SET:UPDATE),sourceIp=[192.168.1.2],

path=[/System/tm-items/certificate-items],payload=[{"hostname":"test","trustpoint":"foo"}]

(SUCCESS)
```

要求が失敗した場合、および失敗したシナリオに基づく場合、両方のログを表示することはできません。

### 無効な要求

無効な要求を送ると、この要求は構成を変更することなく拒否され、元の要求のみがログに記録されます。次に、無効な要求のログの例を示します。

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(GNMI:SET:UPDATE),
sourceIp=[192.168.1.2],path=[/System/tm-items/certificateitems],
payload=[{"hostname":"test","trustpoint":"foo"}] (FAILED)
```

### 失敗した要求

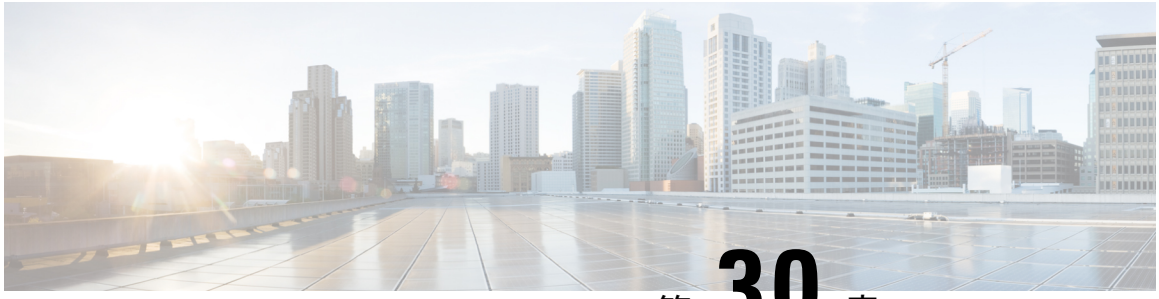
要求を送り、それが何らかの構成上の制限で失敗した場合、元の要求と失敗した構成要求の両方がログに記録されます。次にログの例を示します。

```
Wed Jun 29 20:52:15
2022:type=update:id=1429663200:user=admin:cmd=(COMMIT),database=[candidate],
configMo=[<topSystem childAction="" dn="sys"
status="created,modified"><telemetryEntity childAction="" rn="tm"
status="created,modified"><telemetryCertificate childAction="" filename="foo"
hostname="test" rn="certificate" status="created,modified,replaced"
trustpoint="foo"/></telemetryEntity></topSystem>] (FAILED)

Wed Jun 29 20:52:15
2022:type=update:id=1429663200:user=admin:cmd=(GNMI:SET:REPLACE),
sourceIp=[192.168.1.2],path=[/System/tm-items/certificate-items],
payload=[{"hostname":"test","trustpoint":"foo","filename":"foo"}] (FAILED)
```

要求を送り、コミットに失敗した場合、元の要求と失敗した要求がログに記録されます。次にログの例を示します。

```
Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd
(COMMIT),database=[candidate],configMo=[<topSystem childAction="" dn="sys"
status="created,modified"><telemetryEntity childAction="" rn="tm"
status="created,modified"><telemetryCertificate childAction="" hostname="test"
rn="certificate" status="created,modified"
trustpoint="foo"/></telemetryEntity></topSystem>] (SUCCESS)

Wed Jun 29 14:18:23
2022:type=update:id=1430425712:user=admin:cmd=(GNMI:SET:UPDATE),
sourceIp=[192.168.1.2],path=[/System/tm-items/certificate-items],
payload=[{"hostname":"test","trustpoint":"foo"}] (SUCCESS)
```

```
Wed Jun 29 20:34:06
2022:type=update:id=1429665744:user=admin:cmd=(COMMIT:CANDIDATE-TO-RUNNING),
database=[running] (FAILED)
```



第 30 章

# gNOI：操作インターフェイス


- gNOI について （467 ページ）
- gNOI のガイドラインと制限事項 （469 ページ）
- gNOI の構成 （469 ページ）
- System .Proto （469 ページ）
- OS .Proto （478 ページ）
- Cert .Proto （479 ページ）
- ファイル proto （479 ページ）
- Put （480 ページ）
- Factory Reset .Proto （481 ページ）
- Containerz .Proto （482 ページ）
- gNOI のトラブルシューティング （499 ページ）


## gNOI について

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

gNOI は Google リモートプロシジャコール（gRPC）をトランスポートプロトコルとして使用します。構成は gNMI と同じです。gNMI 構成の詳細については、gRPC エージェント （425 ページ）を参照してください。gNOI RPC 要求を送信するには、各 RPC に gNOI クライアントインターフェイスを実装するクライアントが必要です。Cisco NX-OS リリース 10.1(1) では、gNOI は限られた数のコンポーネントに対してリモートプロシージャコール（RPC）を定義しており、その一部はハードウェア（光インターフェイスなど）に関連しています。

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

表 *41 :* サポートされる *gNOI RPC*

| プロトコル | gNOI RPC | サポートあり |
|---|---|---|
| System | ping | ○ |
| | トレースルート | はい |
| | 時間 | はい |
| | SetPackage | はい |
| | SwitchControl プロセッサ | はい |
| | リブート | はい |
| | RebootStatus | はい |
| | CancelReboot | はい |
| OS | アクティブ化 | はい |
| | インターフェイス | はい |
| Cert | LoadCertificate | はい |
| File | get | はい |
| | Put | はい |
| | Stat | はい |
| | 削除 | はい |
| FactoryReset | 開始 | はい |
| Containerz | [展開（Deploy）] | 対応 |
| | ListImage | はい |
| | RemoveImage | はい |
| | RemoveContainer | はい |
| | ListContainer | はい |
| | StartContainer | はい |
| | StopContainer | はい |
| | UpdateContainer | はい |
| | Log | はい |
| | CreateVolume | はい |
| | RemoveVolume | はい |
| | ListVolume | はい |

# gNOI のガイドラインと制限事項

gNOI 機能には、次の注意事項と制約事項があります。

- 最大 16 のアクティブな gNOI RPC がサポートされます。
- Cisco Nexus 9000 シリーズ スイッチは、1 つの gNMI サービスと 2 つの gNOI マイクロサービスを持つ 1 つのエンドポイントを実行します。

（注） このドキュメントは、gNOI クライアントの例を提供します。参照されるクライアントは、gNOI 要求と応答の raw 交換を示す Python スクリプトです。ユーザーは、関心がある自分のクライアントを使用するものとします。

# gNOI の構成

gNMI は、gRPC エージェントの子機能です。gRPC エージェントを有効にするには、 gRPC エージェント （425 ページ） を参照してください。現在、gNOI の個別の構成はありません。

現在、gNOI の個別の構成はありません。

# System .Proto

システム proto サービスは、設定およびテレメトリ パイプラインの外部でターゲットを管理できるようにする操作可能な RPC のコレクションです。

次に、システム proto の RPC サポートの詳細を示します。

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| ping | ping/ping6 cli コマンド | ターゲットで ping コマンドを実行し、結果をストリームバックします。一部のターゲットでは、すべての結果が使用可能になるまで結果がストリーミングされない場合があります。パケット数が明示的に指定されていない場合は、ping5 が使用されます。 | do_not_resolve オプションはサポートされていません。 |

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| トレースルート | traceroute/traceroute6 cli コマンド | ターゲットで traceroute コマンドを実行し、結果をストリームバックします。一部のターゲットでは、すべての結果が使用可能になるまで結果がストリーミングされない場合があります。最大ホップカウント 30 が使用されます。 | itial_ttl、marx_ttl、wait、do_not_fragment、do_not_resolve、および l4protocol オプションはサポートされていません。 |
| 時間 | ローカル時刻 | ターゲットの現在の時刻を返します。通常、ターゲットが応答しているかどうかをテストするために使用されます。 | - |
| SetPackage | install{ add \| activate } install all nxos | ターゲットのインストールコマンドを実行します。これは、OS 画像または RPM のいずれかのインストールをサポートします。 | bootflash: ファイルシステム上の RPM または OS イメージのみをサポートします。 |
| SwitchControl プロセッサ | system switchover cli コマンド | 現在のルートプロセッサから指定されたルートプロセッサに切り替えます。スイッチオーバーは即座に発生します。応答がクライアントに返されることが保証されない場合があります。 | スイッチオーバーは即座に発生します。その結果、応答がクライアントに返されることが保証されない場合があります。 |
| リブート | reload module | ターゲットをリブートします。 | message オプションはサポートされません。delay オプションはスイッチのリロードでサポートされます。path オプションは 1 つのモジュール番号を受け入れます。 |

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| RebootStatus | show version [module] cli コマンド | ターゲットのリブートのステータスを返します。 | - |
| CancelReboot | reload cancel | 保留中の再起動要求をキャンセルします。 | - |

# SetPackage

gNOI setpackage RPC は、ソフトウェア パッケージをスイッチにコピーし、必要に応じてアクティブ化できるメカニズムを提供します。ソフトウェアパッケージは、RPC発信者から直接コピーすることも、リモート ダウンロードを指定することもできます。いずれの場合も、パッケージをインストールする前に、ハッシュを使用してパッケージの転送が確認されます。

setpackage RPC と関連するすべてのメッセージについての完全な定義は、次を参照してください：

https://github.com/openconfig/gnoi/blob/main/system/system.proto#L61

この RPC の目的は、ソフトウェア パッケージをスイッチにインストールすることです。ソフトウェア パッケージは、RPM またはブート可能 NXOS システム イメージのいずれかです。RPMパッケージは、シスコの署名付き RPM またはサードパーティ RPM のいずれかです。サードパーティ RPM の場合、インストール前に次の構成が必要です。

- `system software allow third-party`

## SetPackage のガイドラインと制約事項

次に、SetPackage に関するガイドラインと制限事項を示します：

- 標準規格の NXOS システム イメージと RPM パッケージのみがサポートされています。その他のファイル タイプは拒否され、RPC に失敗します。
- パッケージは「bootflash:」ファイルシステムにのみコピーできます。
- NXOS システム イメージがサード パーティのアプリケーション パッケージと組み合わされているバンドル イメージはサポートされません。
- RemoteDownload.source_address オプションのサポートなし
- Package.filename は標準規格のNXOS ファイル命名規則に準拠する必要があり、「bootflash:」にのみ存在する必要があります。
- Package.version は空であるか、バージョン文字列に設定されます。空の場合、チェックは行われません。設定されている場合、このバージョンの文字列はインストールされているパッケージの文字列に一致する必要があります。

- SSH/SFTP を使用するリモートコピー操作では、パスワードなしの ssh を使用する必要があります。これは管理者が構成する必要があります。これは、RPC 操作の範囲外です。RemoteDownload で指定されたパスワード。ログイン情報は拒否され、RPC に失敗します。
- HTTP(S) を使用するリモートコピー操作では、プロキシの使用が必要になる場合があります。これには、/etc/.curlrc での手動構成が必要です。
- リモートコピーは、RemoteDownload.source_vrf に特に指定されていない限り、VRF が「デフォルト」であると想定します。

## RPC オプション

次の表に、パッケージメッセージのフィールドと値を示します。この情報はクライアントの実装に依存しないことに注意してください。これは protobuf メッセージとそれらに含まれるデータの形式にすぎません。各 protobuf メッセージを適切に構築するのはクライアントの責任です。

| オプション | 説明 | 値 |
|---|---|---|
| string filename | スイッチ上のパッケージの宛先ファイル名 | bootflash: 上の有効な NXOS ファイル名。 |
| string version | パッケージ バージョンの文字列。 | 空にするか、パッケージバージョンに設定する可能性があります |
| bool activate | パッケージをインストールのみ、またはインストールパッケージを追加でアクティブ化するかどうかを指定します。デフォルトは ‘‘false’ です。 | **activate = false および OS image**<br>スイッチをリロードせずにイメージをインストール<br>**activate = false および RPM**<br>RPM を有効にせずにリポジトリにインストールする<br>**activate = true および OS image**<br>イメージをインストールし、スイッチをリロードする<br>**activate = true および RPM**<br>RPM をインストールして有効にする<br>NX OS SMU RPM の場合、スイッチをリロードする場合とリロードしない場合があります。これは、SMU の影響範囲によって決定されます。SMU RPM のリリース情報を参照してください。 |

| | | |
|---|---|---|
| remote_download | パッケージを別のサーバーの場所から取得するように指定します。 | 非インタラクティブなリモートコピーをのみサポートします：パスワードなしの SSH が設定されている必要があります。RemoteDownload メッセージで提供されるリモート情報。 |

次の表は、RemoteDownload メッセージを示しています。

| オプション | 説明 | 値 |
|---|---|---|
| path | 取得元のホストとパスの情報を指定します。ホスト名または IP を使用できます。 | http(s) :<br>example.com/path/to/package.rpm<br>sftp/scp:<br>a.b.c.d:/path/to/package.rpm |
| protocol | リモート コピー プロトコル | SFTP、SCP、HTTP、HTTPS。パスワードレスである必要があります（scp/sftp）。 |
| source_address | サポート対象外 | |
| source_vrf | リモート コピーを実行する vrf。 | 指定されていない場合、「デフォルト」が使用されます |
| credentials | コピーのためにリモート リソースにアクセスするためのクレデンシャル | 必要な場合、ユーザー名とパスワード |
| • ユーザ名 | リモート サーバーのコピー操作に使用するユーザー名 | リモート コピー操作には必須です。 |
| • パスワード | 必要な場合のパスワード。クリアテキストまたはハッシュ | パスワードが必要な場合（http（s））。scp/sftp では、パスワードレスの ssh キーが必要なため、必要ありません。 |

（注） **プロキシ**：http/https を介したリモートダウンロードの場合、プロキシを介してのみダウンロードが可能なシナリオが存在します。このような場合、ユーザーは curl プロキシ情報を更新する必要があります。

次の例を参照してください。

これにより、リモートダウンロードが指定されたプロキシを通過できるようになり、そのような構成はスイッチのリロード後も保持されます。

```
# feature bash
# run bash sudo su -
# <edit> /etc/.curlrc
--proxy http://<proxy>:<proxy-port>
# ln -s /etc/.curlrc /etc/curlrc
```

（注） **応答とスイッチのリロードのタイミング** ：SetPackage がスイッチをリロードすると、場合によってはスイッチがすぐにリロードされ、ネットワーク接続が中断されることがあります。クライアントが gNOI 応答を受信できない可能性があることが予想されます。

次に、setpackage RPC を使用してスイッチにソフトウェアをインストールするクライアントのインタラクションの例を示します。これらは、さまざまなクライアントを使用して RPC の使用方法を説明する例です。他のクライアントは同様のオプションを使用しますが、詳細は異なります。

### 例：**RPM** のインストール、アクティブ化なし

```
./gnxi-console --host <ip> --port 50051 --cafile /tmp/grpc.pem --hostnameoverride
ems.cisco.com -u admin -p <passwd> --operation gnoi.system.setpackage --arg
local_file=/backup/xbarfile/nxos64-cs.10.6.1.IQD9.0.29.F.bin,package_filename=bootflash:nxos64-cs.10.6.1.IQD9.0.29.F.bin,chunk_size=4000,hash_method=3,checksum=0ad2bf5c3b0be5b7363ac1e18fcc5f8f,package_activate=false


[gnoi.system.setpackage]-------------------------------

SetPackageRequest package {

    filename: "bootflash:nxos64-cs.10.6.1.IQD9.0.29.F.bin"

}

Sent 50777 content RPC messages

End>>

hash {

  method: MD5

  hash: "\n\322\277\\;\013\345\2676:\301\341\217\314_\217"

}


Hex coded checksum: 0ad2bf5c3b0be5b7363ac1e18fcc5f8f

[RESP] : 0





n9k_pi2(config)# show boot

Current Boot Variables:

sup-1

NXOS variable = bootflash:/nxos64-cs.10.6.1.IQD9.0.29.F.bin
```

### 例： **OS** イメージのインストール、アクティブ化

```
./gnxi-console --host <ip> --port 50051 --cafile /tmp/grpc.pem --hostnameoverride
ems.cisco.com -u admin -p <passwd> --operation gnoi.system.setpackage --arg
local_file=/tmp/xyz/nxos64-cs.10.6.1.IQD9.0.29.F.bin,package_file=bootflash:nxos64-cs.10.6.1.IQD9.0.29.F.bin,chunk_size=4000,hash=method:MD5,hash=0ad2bf5c3b0be5b7363ac1e18fcc5f8f,activate=true
```

```
[gnoi.system.setpackage]-------------------------------

SetPackageRequest package { filename: "bootflash:nxos64-cs.10.6.1.IQD9.0.29.F.bin"
activate: true }

Sent 50777 content RPC messages

End>>

hash {

  method: MD5

  hash: "\n\322\277\\;\013\345\2676:\301\341\217\314_\217"

}


Hex coded checksum: 0ad2bf5c3b0be5b7363ac1e18fcc5f8f

[RESP] : 0


// switch reloaded into new image
```

### 例： OS イメージのインストール、リモート ダウンロード

```
./gnxi-console --host <ip> --port 50051 --cafile /tmp/grpc.pem --hostnameoverride
ems.cisco.com -u admin -p <passwd> --operation gnoi.system.setpackage --arg
package_path=/nobackup/xyz/nxos64-cs.10.6.1.IQD9.0.29.F.bin,chunk_size=4000,hash=method:MD5,hash=0ad2bf5c3b0be5b7363ac1e18fcc5f8f,remote_download=protocol:SCP,path:10.30.216.231:/nobackup/xyz/nxos64-cs.10.6.1.IQD9.0.29.F.bin,credentials=username:<user_id>,source_vrf:management,activate=true
```

```
[gnoi.system.setpackage]-------------------------------

SetPackageRequest package {

    filename: "bootflash:nxos64-cs.10.6.1.IQD9.0.29.F.bin"

    remote_download {

        path: "10.30.216.231:/nobackup/xyz/nxos64-cs.10.6.1.IQD9.0.29.F.bin"

        protocol: SCP

        credentials {

            username: "<user_id>"

        }

        source_vrf: "management" } }

End>>
```

```
hash {

    method: MD5

    hash: "\n\322\277\;\013\345\2676:\301\341\217\314_\217"

}

Hex coded checksum: 0ad2bf5c3b0be5b7363ac1e18fcc5f8f

[RESP] : 0

n9k_pi2(config)# show boot

Current Boot Variables:

sup-1

NXOS variable = bootflash:/nxos64-cs.10.6.1.IQD9.0.29.F.bin

Boot Variables on next reload:

sup-1

NXOS variable = bootflash:/nxos64-cs.10.6.1.IQD9.0.29.F.bin
```

### 例：RPM のインストール、アクティブ化なし

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> system.setpackage
  --arg local_file=valgrind-3.14.0-r0.corei7_64.rpm,

        package.filename=bootflash:valgrind-3.14.0-r0.corei7_64.rpm,
        package.version=3.14.0,
        hash.method=3,
        package.activate=false


[REQ]

SetPackageRequest

package {

  filename: "bootflash:valgrind-3.14.0-r0.corei7_64.rpm"

}



Sent 16010 content RPC messages

End>>

hash {

  method: MD5

  hash: "\002\313\016j\233A\"\235\261\325`\333\350>\314\346"

}



Hex coded checksum: 02cb0e6a9b41229db1d560dbe83ecce6
```

```
[RESP] : 0
```

### 例：RPM のインストール、アクティブ化

```
n9k_pi2# show install active


Active Packages:




gnxi-console --host <ip> --port 50051 --cafile /tmp/grpc.pem --hostnameoverride
ems.cisco.com -u admin -p <passwd> --operation gnoi.system.setpackage     --arg
local_file=/tmp/valgrind-3.14.0-r0.corei7_64.rpm,package_file=bootflash:valgrind-3.14.0-r0.corei7_64.rpm,chunk_size=400,hash_method=3,hash=02cb0e6a9b41229db1d560dbe83ecce6,activate=true



### [gnoi.system.setpackage]-------------------------------

SetPackageRequest

package {

  filename: "bootflash:valgrind-3.14.0-r0.corei7_64.rpm"

  activate: true

}


Sent 65 content RPC messages

End>>

hash {

  method: MD5

  hash: "\002\313\016j\233A\"\235\261\325`\333\350>\314\346"

}


Hex coded checksum: 02cb0e6a9b41229db1d560dbe83ecce6

[RESP] : 0




n9k_pi2# show install active
```

```
Active Packages:

        valgrind-3.14.0-r0.corei7_64
```

# OS .Proto

OS サービスは、ターゲット上の OS インストールに対するインターフェイスを提供します。OS パッケージのファイル形式は、プラットフォームによって異なります。プラットフォームは、提供された OS パッケージが有効でブート可能であることを検証する必要があります。これには、既知の良好なハッシュに対するハッシュチェックを含める必要があります。ハッシュは OS パッケージに埋め込むことをお勧めします。

ターゲットは、独自の永続ストレージと OS インストールプロセスを管理します。一連の個別の OS パッケージを保存し、着信する新しい OS パッケージ用に常にプロアクティブにスペースを解放します。ターゲットには、有効な着信 OS パッケージ用の十分なスペースが常にあることが保証されます。現在実行中の OS パッケージは削除しないでください。クライアントは、最後に正常にインストールされたパッケージが使用可能であることを期待する必要があります。

次に、OS プロトコルの RPC サポートの詳細を示します。

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| アクティブ化 | install all nxos bootflash:///img_name | 要求された OS バージョンを、次回のリブート時に使用されるバージョンとして設定します。この RPC は、ターゲットを再起動します。 | 再起動に失敗した場合は、ロールバックまたは回復できません。 |
| 検証 | show version | [検証（Verify）] は、実行中の OS バージョンを確認します。この RPC は、ターゲットの起動中に成功するまで複数回呼び出される場合があります。 | - |

（注） インストール RPC はサポートされていません。

# Cert .Proto

証明書管理サービスは、ターゲットによってエクスポートされます。ローテーション、インストール、およびその他の証明書プロトコル RPC はサポートされていません。

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| LoadCertificate | crypto ca import <trustpoint><br>pkcs12 <file> <passphrase> | CA 証明書のバンドルをロードします。 | - |

# ファイル proto

ファイル proto は、file.proto RPC の機能に基づいてメッセージをストリーミングします。

Get、Stat、および Remove RPC は、bootflash、bootflash://sup-remote、logflash、logflash://sup-remote、usb、volatile、volatile://sup-remote、および debug のファイル システムをサポートします。Put RPC は bootflash のみをサポートしています。

次に、ファイル proto の RPC サポートの詳細を示します。

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| 結果 | | Get はターゲットからファイルの内容を読み取り、ストリーミングします。ファイルは連続したメッセージによってストリーミングされます。各メッセージには最大 64 KB のデータが含まれます。最後のメッセージが送信された後、送信されたデータのハッシュが送信され、ストリームが閉じられます。ファイルが存在しない場合、またはファイルの読み取り中にエラーが発生した場合は、エラーが返されます。 | ファイルサイズの上限は 32 MB です。 |

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| Put | | ターゲットスイッチにファイルをアップロードします。ファイルの接続先は、NXOS 命名構文に従います。ファイルタイプに制限はありません。 | 最大ファイル サイズ（Maximum file size (MB)）<br>bootflash のみ：サポートされています |
| Stat | | Stat は、ターゲット上のファイルに関するメタデータを返します。ファイルが存在しない場合、またはファイルのメタデータへのアクセス中にエラーが発生した場合は、エラーが返されます。 | - |
| 削除 | | Remove は、ターゲットから指定されたファイルを削除します。ファイルが存在しない場合、ディレクトリである場合、または削除操作でエラーが発生した場合は、エラーが返されます。 | - |

# Put

Put 操作により、ユーザはファイルをスイッチにアップロードできます。「remote_file」フィールドは、NX-OS の「copy」コマンドで必要な命名規則に従う必要があります。たとえば、「bootflash:example.txt」などです。スイッチにコピーされるファイルには、次の制限が適用されます。

- 3.5GB の最大ファイル サイズ
- ターゲット ファイルシステムはローカル bootflash: である必要があります。
- RPC ユーザーには、ターゲットファイルへの書き込み権限が必要です。

PutRequest メッセージの形式は、次の URL で詳しく説明されています：
https://github.com/openconfig/gnoi/blob/main/file/file.proto#L78

**例**

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> file.put
  --arg local_file=valgrind-3.14.0-r0.corei7_64.rpm,

        local_file=bootflash:valgrind-3.14.0-r0.corei7_64.rpm,
        remote_file=bootflash:test.rpm,
        hash.method=3



[REQ]

Open>>

open {

  remote_file: "bootflash:test.rpm"

  permissions: 493

}



...


End>>

hash {

  method: MD5

  hash: "\002\313\016j\233A\"\235\261\325`\333\350>\314\346"

}


Hex coded checksum: 02cb0e6a9b41229db1d560dbe83ecce6




[RESP] : 0
```

# Factory Reset .Proto

この .proto は現在、1 つの RPC のみを定義しています。「https://github.com/openconfig/gnoi/blob/master/factory_reset/factory_reset.proto」を参照してください。

| RPC | サポート | 説明（Description） | 制限事項 |
|---|---|---|---|
| FactoryReset | factory-reset module all [bypass-secure-erase] preserve-image force | ターゲットで factory-rest コマンドを実行します。 | 詳細については、後述の内容を参照してください。 |

# FactoryReset

gNOI の初期設定へのリセット操作を行うと、指定されたモジュールのすべての永続ストレージが消去されます。これには、構成、すべてのログデータ、およびフラッシュと SSD（ソリッドステートドライブ）のすべての内容が含まれます。リセットは直前のブートイメージでブートし、ライセンスを含むすべてのストレージを消去します。 gNOI の初期設定へのリセットは、次の 2 つのモードをサポートしています。

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

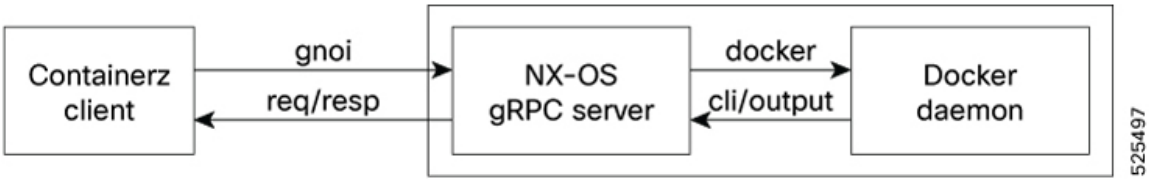
| オプション | Description | 値 |
|---|---|---|
| factory_os | 工場出荷時の OS バージョンにロールバックするかどうかを指定します。 | NX-OS では **true** に設定することはサポートされていません。現在のブートイメージを保持する必要があります。 |
| zero_fill | 時間のかかる、全面的なセキュア消去を実行するかどうかを指定します。 | **zero_fill = true**：factory-reset module all preserve-image force を指定します。<br>**zero_fill =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

# Containerz .Proto

NX-OS は、特権 bash シェルを介したネイティブの Docker アクセスをサポートしています。詳細については、『Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド』の「Cisco NX-OS での Docker を使用」の章 を参照してください。

この gNOI コンテナは、既存の NX-OS Docker 機能に関する GRPC-ish ラッパー インターフェイスをさらに提供します。containerz API インターフェイスを利用するには、ユーザーは最初に上記のガイドラインに従って Docker サービスを起動してプロビジョニングする必要があります。ネイティブの Docker サービスが実行され、NX-OS bash シェル経由でアクセスできるようになると、ユーザーはコンテナを使用して Docker サービスを管理できます。NX-OS は、containerz 要求を受信すると、与えられた gnoi 要求を対応する docker コマンドに変換して呼び出します。docker コマンド出力は、それぞれの containerz 応答形式に変換されてクライアントに配信されます。

図 *4 :*




次に、Containerz proto の RPC サポートの詳細を示します：

| **RPC** | **サポート** | **説明（Description）** |
|---|---|---|
| 展開 | docker image load | tar 形式の docker イメージをアップロードし<br>ポジトリにインストールする |
| ListImage | docker image ls | デプロイされたイメージの一覧表示 |
| RemoveImage | docker image rm | イメージの削除 |
| RemoveContainer | docker container rm | コンテナの削除 |
| ListContainer | docker container ls | コンテナの一覧表示（List containers） |
| StartContainer | docker container run<br>docker container start | 新しいコンテナまたは停止したコンテナを開<br>る |
| StopContainer | docker container stop<br>docker container restart | コンテナの停止または再起動 |
| UpdateContainer | docker container stop<br>docker container start | コンテナの更新 |
| Log | docker container logs | コンテナ ログを取得する |
| CreateVolume | docker volume create | docker ボリュームの作成 |
| RemoveVolume | docker volume rm | docker ボリュームの削除 |
| ListVolume | docker volume ls | Docker ボリュームのリスト |

（注） StartPlugin、StopPlugin、ListPlugin、および RemovePlugin RPC はサポートされていません。

### <Deploy>

展開操作により、ユーザーはDockerイメージ(「tar」または「tar.gz」形式)をスイッチにアップロードし、スイッチのリポジトリに登録できます。

展開 RPC については、次の場所に記載されています：

https://github.com/openconfig/gnoi/blob/main/containerz/containerz.proto#L54

展開 RPC の目的は、イメージアーカイブを Docker イメージレジストリにロードすることです。展開 RPC は、「docker load」コマンドラインと同等の機能を提供します。イメージアーカイブファイルは、GZIP または TAR ファイルで、有効な Docker イメージが含まれている必要があります。

### 遅延展開に関するガイドラインおよび制限事項

展開操作に関するガイドラインと制限事項は次のとおりです。

- 展開するファイルは、Gzip または TAR 形式の有効な Docker イメージである必要があります。
- プラグインは、サポートされません。
- 非インタラクティブなリモートコピーのみを実行できます。つまり、 SCP/SFTP の場合、管理者は RPC を実行する前にパスワードレス SSH をセットアップ必要があります。
- リモートダウンロードの場合、スイッチはImageTransferReadyを送信せず、クライアントは ImageTransferEnd を送信しないと見なされます。

### RPC オプション

RPC の実行中に次の protobuf メッセージが交換されます（詳細については、RPC プロトコルの定義を参照してください）。これらのメッセージは、クライアントによってスイッチに送信されます。

ImageTransfer メッセージは、展開操作を開始するためにクライアントからスイッチに送信される最初のメッセージです。

| メッセージオプション | 説明 | 値 |
|---|---|---|
| 文字列の名前 | インストールしたイメージに付ける名前 | 有効な Doc |
| 文字列のタグ | インストール後に画像に適用されるタグ | 有効な dock |
| Unit64 image_size | インストールされるイメージのサイズ（バイト）。これは、イメージをロードするのに十分な空き領域があることを確認するためにチェックされます。 | ユーザーは<br>ます。この<br>Size」で確<br>Dockerパー<br>なりません |
| Bool is_plugin | サポート対象外 | いいえ（Fa |

| メッセージ オプション | 説明 | 値 |
|---|---|---|
| RemoteDownload remote_download | | スイッチ<br>る場合に<br>ブなリモ<br>パスワー<br>がありま<br>供される |

RemoteDownload メッセージを示す表

| オプション | 説明 | 値 |
|---|---|---|
| path | 取得元のホストとパスの情報を指定します。ホスト名または IP を使用できます。 | http(s) :<br>example.<br>sftp/scp:<br>a.b.c.d:/p |
| protocol | リモート コピー プロトコル | SFTP、S<br>である必 |
| source_address | サポート対象外 | |
| source_vrf | リモート コピーを実行する vrf。 | 指定され<br>されます |
| credentials | コピーのためにリモートリソースにアクセスするためのクレデンシャル | 必要な場 |
| • ユーザ名 | リモート サーバーのコピー操作に使用するユーザー名 | リモート |
| • パスワード | 必要な場合のパスワード。クリアテキストまたはハッシュ | パスワー<br>では、パ<br>必要あり |

バイトメッセージ

| オプション | 説明 | 値 |
|---|---|---|
| content | このメッセージは、スイッチに直接コピーを行う場合にデータを伝送します。 | コピーさ<br>raw バイ |

ImageTransferEnd メッセージ

| オプション | 説明 | 値 |
|---|---|---|
| <none> | これは、スイッチに転送されたバイトの終了を通知する空のメッセージです。 | このメッ |

次の protobuf メッセージは、DeployResponse メッセージの形式でスイッチからクライアントに送信されます。

- ImageTransferReady：スイッチはデータを受信する準備が整っています。
- ImageTransferProgress：これまでに受信したバイト数をクライアントに示します。
- ImageTransferSuccess -- クライアントへの転送が成功したことを示します
- Google.rpc.Status：RPC ステータス コード

ImageTransferReady メッセージ

| オプション | 説明 | 値 |
|---|---|---|
| chunk_size | 各コンテンツ メッセージで伝送するデータ量をクライアントに示します。 | 常に 64KB |

ImageTransferProgress メッセージ

| オプション | 説明 | 値 |
|---|---|---|
| Bytes_received | データ コピーでこれまでにスイッチに正常に転送されたバイト数 | 2 MB の受信れた合計バト コピーの |

ImageTransferSuccess メッセージ

| オプション | 説明 | 値 |
|---|---|---|
| 名前 | レジストリにロードされたイメージの名前 | 文字列の D |
| tag | このイメージに使用されたタグ | 文字列とし |
| image_size | 読み込まれた画像サイズ | バイト単位 |

### クライアント RPC インタラクションの例

次の例は、特定のクライアントを使用した展開 RPC の使用方法を示しています。詳細は他のクライアントによって異なりますが、これは基本的な相互作用を示しています。

#### 例：直接アップロード

```
./gnxi-console --host 172.22.244.142 --port 50051 --cafile /tmp/grpc.pem --hostnameoverride
 ems.cisco.com -u admin -p <passwd>
--operation gnoi.containerz.deploy --arg
image_transfer.name=alpine_dev,image_transfer.tag="DEV-1.0.1",image_transfer.image_size=3309581,
image_transfer.is_plugin=false,local_file=/auto/mtx-dev/sanity/docker/docker_alpine_latest.tar.gz
```

```
### [gnoi.containerz.deploy]--------------------------------
Transfer>>
image_transfer {
  name: "alpine_dev"
  tag: "DEV-1.0.1"
```

```
  image_size: 3309581
}

[RESP] : 0
 image_transfer_ready {
  chunk_size: 64000
}

Sent 52 chunks
End>>
image_transfer_end {
}

[RESP] : 0
 image_transfer_progress {
  bytes_received: 2112000
}

[RESP] : 0
 image_transfer_success {
  name: "alpine_dev"
  tag: "DEV-1.0.1"
  image_size: 3309581
}

The state of docker on n9k:

bash-5.1# docker image ls
REPOSITORY                                TAG         IMAGE ID       CREATED
SIZE
busybox                                   latest      af4709625109   5 months ago
4.27MB
alpine_dev                                DEV-1.0.1   2c64cf60f6f0   6 months ago
7.35MB
```

## 例：リモート展開

```
(pyats) [sjc-ads-7111]$ ./gnxi-console --host <ip> --port 50051 --cafile /tmp/grpc.pem
--hostnameoverride ems.cisco.com -u admin -p <passwd> --operation gnoi.containerz.deploy
 --arg
[illegible]

[gnoi.containerz.deploy]-------------------------------
Transfer>> image_transfer {
     name: "alpine"
     tag: "new_image"
     image_size: 7349436
     remote_download {
         path: "10.0.0.1:/auto/mtx-dev/sanity/docker/docker_alpine_latest.tar.gz"
        protocol: SCP
        credentials {
                   username: "<user>"
           }
     source_vrf: "management" } }
[RESP] : 0 image_transfer_success {
       name: "alpine"
       tag: "new_image"
       image_size: 7349436
  }
bash-5.1# docker image ls
REPOSITORY    TAG         IMAGE ID       CREATED        SIZE
busybox       latest      af4709625109   8 months ago   4.27MB
alpine        new_image   2c64cf60f6f0   9 months ago   7.35MB
```

### <ListImage>

ListImage操作は、スイッチのローカルリポジトリ内のイメージのリストを返します。各イメージには、イメージ ID、名前、タグなどの情報が含まれます。

ListImage 操作は、次のオプションのどれでもサポートします。

| オプション | Description | 値 |
|---|---|---|
| Limit | 「limit」は、「docker image list」コマンドと同じ表示順序に従って、応答で返されるイメージエントリの数を制限します。 | int32 として |
| Filter<br>{key:value} | ListImage操作は、フィルタに一致するすべての画像を返します | キーと値の |

次の例は、ListImage 操作の使用方法を示しています。

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.listimage
  --arg limit=2


[REQ]
 limit: 2

[RESP] : 1
 id: "f7a401783329"
 image_name: "ghcr.io/openconfig/gnmic"
 tag: "latest"

[RESP] : 2
 id: "2c64cf60f6f0"
 image_name: "alpine"
 tag: "latest"
```

### <RemoveImage>

RemoveImage 操作は、名前で識別される Docker コンテナを削除します。

デフォルトでは、イメージが一部のコンテナによって使用されている場合、操作は失敗します。「force」オプションを使用すると、イメージを強制的に停止して削除できます。ただし、この場合、docker はイメージのタグを解除するだけです。

RemoveImage 操作は、次のオプションをサポートします。

| オプション | 説明 | 値 |
|---|---|---|
| Name | 削除するイメージ名 | 文字列とし |
| タグ | 削除するタグ | 有効な文字 |
| 強制 | 「force=true」の場合、イメージは強制的に削除/タグなしにされます | True/False。<br>います。 |

次の例は、RemoveImage 操作の使用方法を示しています。

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.removeimage
```

```
  --arg name=alpine

[REQ]
 name: "alpine"

[RESP]
 code: UNKNOWN
detail: "Error response from daemon: conflict: unable to remove repository reference
\"alpine\" (must force) - container 21e718a3bf52 is using its referenced image
2c64cf60f6f0\n"
```

次の例は、強制削除（タグ解除のみ）を示しています。

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.removeimage
  --arg name=alpine,force=True

[REQ]
 name: "alpine"
force: true

[RESP]
 code: SUCCESS
 detail: "Untagged: alpine:latest\n"
```

次の例は、コンテナが停止した後に再度強制削除することを示しています。

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.removeimage
  --arg name= 2c64cf60f6f0,force=True

[REQ]
 name: "foo"
force: true

[RESP]
 code: UNKNOWN
detail: "Deleted:
sha256:2c64cf60f6f05256c049f58403d0c3b33f2145a70830cb8e24efa826854c1a46\n"
```

### <StartContainer>

StartContainer 操作は、新しいコンテナを開始するか、以前に停止したコンテナを再起動するために使用できます。判断は、次の条件に基づきます。

- コンテナが存在しない場合は、新しいコンテナを開始します
- otherwise
  - コンテナがすでに実行されている場合は、エラーを返します。
  - コンテナが停止しているが、リクエストにインスタンス名だけではない多くのフィールドが含まれている場合は、エラーを返します。
  - コンテナが停止し、リクエストにインスタンス名のみが含まれている場合は、コンテナを開始します。

StartContainer は次のオプションをサポートしています

| オプション | Description | 値 |
|---|---|---|
| image_name | コンテナを起動するイメージ名 | 文字列と |

| | | |
|---|---|---|
| tag | コンテナを開始するイメージ タグ | 有効な文字 |
| cmd | 新しいコンテナで実行するコマンド | 有効なコマ |
| instance_name | 新しいコンテナの識別子 | 文字列とし |
| port<br>-内部<br>-外部 | コンテナの外部に公開される内部ポートのリスト | 有効な内部<br>例：内部ポ<br>外部ポート |
| 環境 =<VAR1> ：<value> | コンテナでの環境変数の設定 | 文字列とし<br>ペアだけに<br>例：VAR1 |
| ネットワーク（network） | このコンテナを接続するネットワーク | 有効なネッ<br>これは、「<br>ランタイム<br>クです。 |
| Cap.ADD | 追加する機能 | 有効な機能<br>https://man7.o<br>で確認でき<br>トです。<br>AUDIT_WR<br>FOWNER、<br>NET_BIND<br>SETFCAP、<br>SYS_CHRO |
| Cap.REMOVE | 削除される機能 | |
| Restart.policy | 再起動ポリシーの設定 | 有効な再起<br>NONE、AL<br>ON_FAILU |
| Restart.attempts | 再起動の試行を設定 | unit32とし |
| RunAs.user<br>RunAs.group | このコンテナを実行するユーザーとグループを設定 | 文字列とし |
| ラベル（Label）<br><key>:<val> | メタデータをコンテナに設定するためのキーと値のペア | 文字列ラベ<br>にすること<br>例: my-label |
| Limits.max_cpu | このコンテナで使用できる最大CPUの数を設定。 | 有効な CPU<br>例：値 0.5<br>CPU の半分 |

| | | |
|---|---|---|
| Limits.soft_mem_bytes | メモリのソフト制限を設定 | 有効な<br>は、使<br>必要があ<br>例：soft |
| Limits.hard_mem_bytes | メモリのハード制限を設定 | 有効な<br>例：limi |

### ガイドラインと制約事項

- StartContainer 操作は、「docker run」 CLI を介して実現されます。この CLIでサポートされる最大長は 2500 バイトです。
- RunAs の場合、現在の Docker は UID と GID のみをサポートします。コンテナは UID と GID の任意の値で開始し、ユーザー名とグループ名が指定されると失敗します。
- コンテナの起動中に警告が発生した場合、NXOS は警告を返さずにコンテナを起動します。

次の例は、StartContainer 操作を示しています。

例：新しいコンテナの開始

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.startcontainer
  --arg instance_name=foo,
        image_name=busybox,
        cmd="sh -c \"sleep 300\"",
        volume=my_volume:/docker

[REQ]
 image_name: "busybox"
 cmd: "sh -c \"sleep 300\""
 instance_name: "foo"
 volumes {
   name: "my_volume"
   mount_point: "/docker"
 }

[RESP]
 start_ok {
  instance_name: "foo"
}
```

例：新しいコンテナの開始

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.startcontainer
--arg image_name=alpine,
                        cmd="sh -c \"while true; do $(echo date); sleep 1; done\"",
                        instance_name=foo,
                        volume=my_volume:/docker:True,
                        env=VAR1:value1,env=VAR2:value2,
                        cap.add=FOWNER,
                        cap.add=AUDIT_WRITE,
                        cap.remove=SETFCAP,
                        cap.remove=NET_RAW,
                        network=host,
                        restart.policy=3,restart.attempts=2,
                        run_as.user=1001,run_as.group=2003,
```

```
                        label=env:prod,label=team:neteng,
                        limits.max_cpu=1.5,
                        limits.soft_mem_bytes=7000000,
                        limits.hard_mem_bytes=90000000,
                        port=1001:2001
[REQ]
image_name: "alpine2"
cmd: "sh -c \"while true; do date; sleep 1; done\""
instance_name: "foo"
ports {
  internal: 2001
  external: 1001
}
environment {
  key: "VAR1"
  value: "value1"
}
environment {
  key: "VAR2"
  value: "value2"
}
volumes {
  name: "my_volume"
  mount_point: "/docker"
  read_only: true
}
network: "host"
cap {
  add: "FOWNER"
  add: "AUDIT_WRITE"
  remove: "SETFCAP"
  remove: "NET_RAW"
}
restart {
  policy: ON_FAILURE
  attempts: 2
}
run_as {
  user: "1001"
  group: "2003"
}
labels {
  key: "env"
  value: "prod"
}
labels {
  key: "team"
  value: "neteng"
}
limits {
  max_cpu: 1.5
  soft_mem_bytes: 7000000
  hard_mem_bytes: 90000000
}

[RESP]
start_ok {
  instance_name: "foo"
}
```

### 例：既存の実行中のコンテナの開始

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.startcontainer
  --arg instance_name=foo,
        image_name=busybox
```

```
[REQ]
 image_name: "busybox"
 instance_name: "foo "

[RESP]
 start_error {
  error_code: UNKNOWN
  details: "container is already running"
}
```

例：既存の停止したコンテナの起動

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.startcontainer
  --arg instance_name=foo

[REQ]
 instance_name: "foo "

[RESP]
 start_ok {
  instance_name: "foo "
}
```

### <RemoveContainer>

RemoveContainer 操作は、名前で識別される Docker コンテナを削除します。

デフォルトでは、コンテナがまだ実行中の場合、操作は失敗します。**force** オプションを使用すると、コンテナを強制的に停止して削除できます。

RemoveContainer 操作では、次のオプションがサポートされています。

| オプション | Description | 値 |
|---|---|---|
| Name | 削除するコンテナ名 | 文字列と |
| 強制 | 「force=True」の場合、コンテナは停止し、強制的に削除されます | True/Fals います。 |

Dockerイメージとコンテナの前には一定の独立性があるため、基礎となるイメージが削除された場合でも、コンテナは存在し続ける可能性があることを認識することが重要です。ユーザーはイメージとコンテナを慎重に削除する必要があります。

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.removecontainer

[REQ]
 name: "foo"

[RESP]
 code: RUNNING
detail: "Error response from daemon: You cannot remove a running container
706723d4ae83e3fa7b5dcfe4edbb8edd570ea8af690c3eef2526f9c603bfba97. Stop the container
before attempting removal or force remove\n"
```

例：強制削除

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.removecontainer
  --arg force=True
```

```
[REQ]
 name: "foo"
force: true

[RESP]
 code: SUCCESS
detail: "Container removed successfully"
```

### <ListContainer>

ListContainer 操作は、スイッチのローカル リポジトリ内のコンテナのリストを返します。

各コンテナには、イメージ名、コンテナ ID、名前、実行ステータス、コンテナランタイムによって計算されたラベルとイメージハッシュを含むメタデータなどの情報が含まれます。

ListContainer 操作では、次のオプションがサポートされます

| オプション | Description | 値 |
|---|---|---|
| すべて | 「all」オプションを使用すると、実行中のコンテナと実行されていないコンテナをすべて返すことができます。デフォルトでは、ListContainer 操作は実行中のコンテナのみを返します。 | **True/False。ています。** |
| Limit | 「limit」は、「docker container list」コマンドと同じ表示順序に従って、応答で返されるコンテナ エントリの数を制限します。 | int32 として |
| Filter<br>{key:value} | ListImage 操作は、フィルタに一致するすべての画像を返します | キーと値の |

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.listcontainer
  --arg name=my_volume

[REQ]

[RESP] : 1
 id: "706723d4ae83e3fa7b5dcfe4edbb8edd570ea8af690c3eef2526f9c603bfba97"
 name: "my_volume"
 image_name: "alpine"
 status: RUNNING
 hash {
   method: SHA256
   hash: "sha256:2c64cf60f6f05256c049f58403d0c3b33f2145a70830cb8e24efa826854c1a46"
 }
```

例：すべてのコンテナ

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.listcontainer
  --arg all=True

[REQ]
 all: true

[RESP] : 1
```

```
 id: "706723d4ae83e3fa7b5dcfe4edbb8edd570ea8af690c3eef2526f9c603bfba97"
 name: "foo"
 image_name: "alpine"
 status: RUNNING
 hash {
   method: SHA256
   hash: "sha256:2c64cf60f6f05256c049f58403d0c3b33f2145a70830cb8e24efa826854c1a46"
 }

[RESP] : 2
 id: "21e718a3bf525635b606b6a121bf8a79c323da4edc044f2e6c3811cd3219de94"
 name: "test"
 image_name: "alpine"
 status: STOPPED
 hash {
   method: SHA256
   hash: "sha256:2c64cf60f6f05256c049f58403d0c3b33f2145a70830cb8e24efa826854c1a46"
 }
```

### <StopContainer>

StopContainer 操作は、名前で識別される Docker コンテナを停止します。これは、「docker container stop」コマンドを介して実行されます。「restart」オプションを指定した場合、「docker chain restart」コマンドを使用して docker コンテナも再起動します。

StopContainer 操作は次のオプションをサポートしています

| オプション | Description | 値 |
|---|---|---|
| Instance_name | 停止するコンテナ インスタンス | 文字列と |
| force | このオプションが true の場合、コンテナは強制的に停止されます | True/Fals います。 |
| 再起動 | このオプションが true の場合、コンテナは「docker container restart」を使用して再起動されます。 | True/Fals います。 |

制限と不具合

- 実行中のコンテナが StopContainer 操作を使用して停止されると、エラーを返さずにコンテナを停止するのに 10 秒かかります。
- 実行中のコンテナを「force」オプションで停止すると、コンテナは5秒後に停止します。

次の例は、StopContainer 操作を示しています。

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.stopcontainer
  --arg instance_name=foo


[REQ]
 Instance_name: "foo"

[RESP]
 code: SUCCESS
details: "Container stopped/re-started successfully"
```

### <UpdateContainer>

UpdateContainer 操作は、新しいイメージと引数でコンテナ インスタンスを更新します。更新が失敗すると、コンテナは以前の状態に復元されます。

現在、ネイティブのDockerはこの操作をサポートしていないため、NX-OSはシーケンスDocker操作を介してこの機能を実現することに注意してください。

1. 現在実行中の場合は、指定されたコンテナを停止する
2. コンテナの名前をバックアップとしての一時的な名前に変更します
3. 指定された新しいイメージと引数を使用して、同じコンテナ名で新しいコンテナを開始します。
4. 新しいコンテナの実行に成功した場合は、バックアップ コンテナを削除します。
5. 新しいコンテナの実行に失敗した場合は、復元を試みます
   - 新しいコンテナを停止して削除します。
   - バックアップを元のコンテナ名に戻します
   - 実行していた場合、コンテナを開始します。

UpdateContainer 操作では、次のオプションがサポートされています。

| オプション | Description | 値 |
|---|---|---|
| instance_name | 更新する実行コンテナの名前。 | 文字列とし |
| image_name | コンテナを更新する イメージ。 | 文字列とし |
| image_tag | コンテナを更新するためのタグ。 | 文字列とし |
| async | この操作を非同期で実行するかどうか | True/False。<br>います。 |
| コンテナを実行するための startContainer 操作でサポートされているすべてのオプション。 | | |

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.updatecontainer
  --arg instance_name=foo,image_name=alpine,params=<same as StartContainer>

[REQ]
 instance_name: "foo"
 image_name: ”alpine”
 params {
  instance_name: "foo"
  cap {
  }
  ...
 }

[RESP]
 updateOk: {
  instanceName: “foo”
 }
```

### <Log>

ログ操作により、「docker container logs」コマンドと同様に、クライアントはコンテナ ログを取得できます。操作で、ログを「フォローする」かどうかを選択できます。

- Follow=False の場合、スイッチは現在のログをフェッチして返します。
- Follow=True の場合、スイッチは現在のログを返し、RPC がクライアントによってキャンセルされるか、コンテナが停止するまで、新しいログをストリーミングします。

ログ操作は、次のオプションのどれでもサポートします

| オプション | Description | 値 |
|---|---|---|
| instance_name | コンテナ名 | 文字列と |
| follow | 設定されている場合、クライアントがキャンセルするまで、ストリームは開いたままになります。 | True/Fals ていまず |

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.log
  --arg instance_name=test

[REQ]
 instance_name: "test"

[RESP] : 1
 msg: "Wed Jun 11 20:21:05 UTC 2025\nWed Jun 11 20:21:06 UTC 2025\nWed Jun 11 20:21:07
UTC 2025\nWed Jun 11 20:21:08 UTC 2025\nWed Jun 11 20:21:09 UTC 2025\nWed Jun 11 20:21:10
 UTC 2025\nWed Jun 11 20:21:11 UTC 2025\nWed Jun 11 20:21:12 UTC 2025\n
Wed Jun 11 20:21:13 UTC 2025\nWed Jun 11 20:21:14 UTC 2025\nWed Jun 11 20:21:15 UTC
2025\nWed Jun 11 20:21:16 UTC 2025\nWed Jun 11 20:21:17 UTC 2025\nWed Jun 11 20:21:18
UTC 2025\nWed Jun 11 20:21:19 UTC 2025\nWed Jun 11 20:21:20 UTC 2025\nWed Jun 11 20:21:21
 UTC 2025\nWed Jun 11 20:21:22 UTC 2025\nWed Jun 11 20:21:23 UTC 2025\nWed Jun 11 20:21:24
 UTC 2025\n"

[RESP] : 2
 msg: "Wed Jun 11 20:21:25 UTC 2025\n
```

例：ログをたどる

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.log
  --arg instance_name=test,follow=True

[REQ]
 instance_name: "test"

…

[RESP] : 16
 msg: "Wed Jun 11 20:26:06 UTC 2025\nWed Jun 11 20:26:07 UTC 2025\nWed Jun 11 20:26:08
UTC 2025\nWed Jun 11 20:26:09 UTC 2025\nWed Jun 11 20:26:10 UTC 2025\nWed Jun 11 20:26:11
 UTC 2025\nWed Jun 11 20:26:12 UTC 2025\nWed Jun 11 20:26:13 UTC 2025\n"

[RESP] : 17
 msg: "Wed Jun 11 20:26:14 UTC 2025\n"

[RESP] : 18
 msg: "Wed Jun 11 20:26:15 UTC 2025\n"
```

…

### <CreateVolume>

CreateVolume 操作では、名前で識別される Docker ボリュームが作成されます。

ユーザーは、このボリュームを複数のコンテナにマウントできます。

CreateVolume 操作は、次のオプションのどれでもサポートします

| オプション | Description | 値 |
|---|---|---|
| 名前 | ボリュームの名前 | 文字列とし |
| ドライバ | ボリューム ドライバ | NXOS は「<br>します。 |
| オプション | ドライバのオプション。実際のオプションキーと値はドライバ固有です。 | NXOS はこ<br>せん。 |
| ラベル | ボリュームに適用するラベル。ラベルはボリュームのメタデータです。 | 文字列ラベ<br>にすること<br>例: my-label |

#### 注意事項と制限事項

- NXOS は、「LOCAL」のドライバ ro create ボリュームのみをサポートします。
- NXOS は「ローカル マウント オプション」をサポートしていません。

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.createvolume
  --arg name=my_volume

[REQ]
 name: "my_volume"

[RESP] : 1
 name: “my_volume”
```

\

### <RemoveVolume>

RemoveVolume 操作は、名前で識別される Docker ボリュームを削除します。

RemoveVolume 操作は次のオプションをサポートしています

| オプション | Description | 値 |
|---|---|---|
| 名前 | 削除するボリューム名 | 有効なボリ |
| force | ボリュームを強制的に削除する | NXOSでサ |

#### ガイドラインと制約事項

- 「force」オプションはdockerではサポートされていないため、ボリュームを正常に削除できるように、すべてのコンテナからボリュームを切り離してください

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.removevolume
  --arg name=my_volume

[REQ]
 name: "my_volume"

[RESP] : 1
 name: “my_volume”
```

#### <ListVolume>

ListVolume 操作は Docker ボリュームのリストを返します。

これには、ボリューム名、作成されたタイムスタンプ、ドライバ、およびボリュームのオプションと該当する場合、ラベルが含まれます。

ListVolume 操作は次のオプションをサポートします

| オプション | Description | 値 |
|---|---|---|
| Filter<br>{key:value} | ListVolume 操作では、フィルタに一致するすべてのボリュームが返されます | キーと値 |

例

```
> gnoi-client --host <ip> --port <port> -u admin -p <pass> containerz.listvolume

[REQ]

[RESP] : 1
 name: "92945dc4e9ffdf571c85994738562b1d1f54158f784cac3eadc080c558e034ee"
 created {
   seconds: 1748490463
 }
 driver: "local"
```

# gNOI のトラブルシューティング

## gNOI のデバッグ

gNOI のステータスを確認するには、次のコマンドを入力します。

# show コマンド

| コマンド | 説明 |
|---|---|
| **clear grpc gnoi rpc** | カウンタまたは呼び出しをクリーンアップするために使用されます。 |
| **debug grpc events** {**events**\|**errors**}<br>**show grpc nxsdk event-history** {**events**\|**errors**} | イベント履歴からイベントとエラーをデバッグします。 |
| **clear grpc gnoi rpc** | カウンタまたは呼び出しをクリーンアップするために使用されます。 |

# 出力例

**show grpc gnmi service statistics**

```
=============
gRPC Endpoint
=============

Vrf            : management
Server address : [::]:50051

Status         : Running - certificate expired
Cert notBefore : Jun 20 16:43:49 2023 GMT
Cert notAfter  : Jun 21 16:43:49 2023 GMT
Client Root Cert notBefore : n/a
Client Root Cert notAfter  : n/a

Max concurrent calls            :  16
Active calls                    :  0
```

# Bash コマンド

特に、containerz については、「Cisco Nexus 9000 シリーズ NX-OS プログラマビリティ ガイド」の「Cisco NX-OS で Docker を使用」の章のデバッグ手順を参照してください。基本的に、ユーザーは Docker ユーティリティに直接アクセスして、デバッグ目的で各イメージとコンテナの詳細を調べることができるものとします。

| コマンド | 説明 |
|---|---|
| docker image list | docker イメージをリスト |
| docker container list | Docker コンテナをリスト |
| docker volume list | Docker ボリュームをリスト |

## 出力例

**Docker image list**

```
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE

busybox      latest    af4709625109   8 months ago   4.27MB

alpine       latest    2c64cf60f6f0   9 months ago   7.35MB
```

**docker container list**

```
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

root@nxosv-104#docker container list -a

CONTAINER ID   IMAGE            COMMAND                  CREATED       STATUS
          PORTS     NAMES

9bd2156dd341   busybox:latest   "sh"                     5 days ago    Exited (255) 3
days ago             stupefied_euclid

c8d945cfc5f6   busybox:latest   "/bin/bash"              5 days ago    Created
                    determined_lehmann

7463c28fd039   busybox          "sh -c 'while true; …"   5 days ago    Exited (255) 3
days ago             busy_test

26dc68f0e4d2   alpine:latest    "/bin/bash"              5 days ago    Created
                    admiring_raman

0f5de1376925   alpine:latest    "/bin/bash"              5 days ago    Created
                    vigilant_keller

4ac6c84be9f1   alpine:latest    "/bin/bash"              5 days ago    Created
                    optimistic_dirac

e5b43676fc98   alpine           "sh -c 'sleep 100'"      3 weeks ago   Exited (0) 3 weeks
 ago              gnoi_docker_container_20250519_183038_7

d912c55525ae   alpine           "sh -c 'sleep 100'"      3 weeks ago   Exited (0) 3 weeks
 ago              gnoi_docker_container_20250519_183028_6

0360fcde5af4   alpine           "sh -c 'sleep 100'"      3 weeks ago   Exited (0) 3 weeks
 ago              gnoi_docker_container_20250519_183019_5

53f2e94399ca   alpine           "sh -c 'sleep 100'"      3 weeks ago   Exited (0) 3 weeks
 ago              gnoi_docker_container_20250519_182049_4

35130a5e3f86   alpine           "sh -c 'sleep 100'"      3 weeks ago   Exited (0) 3 weeks
 ago              gnoi_docker_container_20250519_182039_3

5d6dbe0dc904   alpine           "sh -c 'sleep 100'"      3 weeks ago   Exited (0) 3 weeks
 ago              gnoi_docker_container_20250519_181102_2
```

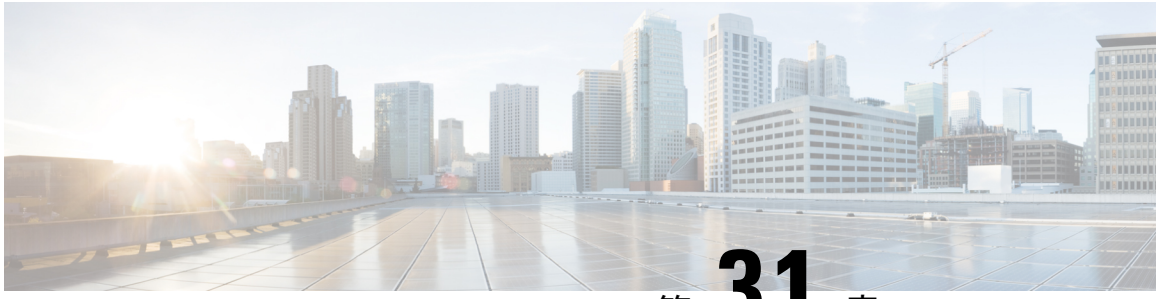
# デバッグ ログの収集

gNOI は、gRPC エージェントの子サービスです。詳細については、「gRPC エージェント（425 ページ）」の章を参照してください。

### Docker ログ

特に、containerzについては、次の場所のログを調査してください。このファイルには、Docker デーモンのデバッグ メッセージが維持されます。詳細については、https://docs.docker.com/engine/daemon/logs/ を参照してください。

```
/var/log/docker
```



第 31 章

# gRPC トンネル


- gRPC トンネルについて （503 ページ）
- 注意事項と制約事項 （504 ページ）
- gRPC トンネルの設定 （504 ページ）
- 障害対応 （513 ページ）


## gRPC トンネルについて

Cisco NX-OSは、「gPRC トンネル」をサポートします。これは、アプリケーションレイヤトンネルの特定の実装で、ファイアウォールなどで分離されたネットワーク間での外部通信を可能にします。

一般的なネットワーク展開では、ファイアウォールを使用して、ネットワークをファイアウォールの「外部」と「内部」に大まかに分離します。そうした場合、たとえばgNMIでは、「gnmi クライアント（コントローラ）」は通常ファイアウォールの外部にあり、「gnmi サーバ（ネットワーキングスイッチ）」は内部にあります。したがって、gNMI 接続はスイッチへのインバウンド RPC と呼ばれ、通常は「ダイヤルイン」と呼ばれます。

このような「ダイヤルイン」には、次の 2 つの制限があります。

- **ファイアウォールのプロビジョニング**：

  「ダイヤルイン」では、gNMI 接続が通過できるように、特定のホスト、アドレス、ポートなどに関連したファイアウォールルールを指定して、ファイアウォールに穴をあける必要があります。

- **前提条件のネットワークインベントリ**：

  特定の接続先に「ダイヤルイン」するには、最初に勧誘手順が必要です。ユーザーは手動またはプログラムで、gNMI サーバーのホスト情報を収集します。

  そうして初めて、クライアントはそれらのサーバーに「ダイヤルイン」するためのアドレス/ポートを知ることができます。

grpc-tunnel は、これら 2 つの制限を回避することを目的として、逆の「ダイヤルアウト」アプローチによりトンネルを確立します。grpc トンネルの詳細については、 gRPC Tunnel の外部ドキュメントを参照してください。

# 注意事項と制約事項

gRPC トンネルには、次の注意事項と制約事項があります。

- トンネルのターゲット識別子を割り当てるときの命名規則は、完全にユーザーに任されています。
- ユーザーは、ターゲット識別子の命名規則が一意であることを確認する必要があります。自動展開ワークフローでターゲット識別子の一意性を扱うようにすることをお勧めします。
- Cisco NX-OS は、最大 8 つのトンネル構成をサポートします。

## トピック 2.1

# gRPC トンネルの設定

## 認証なしの gRPC トンネルの構成

この手順では、サーバまたはクライアント認証なしに、gRPC トンネルを有効にして構成する方法について説明します。これは主に実験用です。

### 始める前に

gRPC トンネルは、さまざまなネットワークトラフィックを転送できることが想定されている、透明性のないトンネルです。ただし、 Cisco NX-OS における主なユースケースは、gNMI/gNOI 要求のプロキシを行うことです。その場合は、grpcエージェントを適切に設定する必要があります。gRPC エージェントのプログラミングガイドを参照してください。

### 手順の概要

1. **configure terminal**
2. **[no] feature grpctunnel**
3. **[no] grpctunnel destination**

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch-1# 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 | **[no] feature grpctunnel**<br>例：<br>`switch-1# feature grpctunnel` | grpc-tunnel 機能を有効または無効にします。 |
| ステップ 3 | **[no] grpctunnel destination**<br>例：<br>`switch-1# grpctunnel destination 1.1.1.1 port 8000 target test1 type GNMI_GNOI use-vrf management` | トンネル接続を構成します。トンネル接続を削除するには、このコマンドの no 形式を使用します。<br>• **destination**：（タイプ：IPv4/IPv6 アドレスまたはホスト名文字列）トンネル サーバーの IP アドレスまたはホスト名。ホスト名が指定されている場合は、有効なネームサーバー構成が必要です。<br>• **port**：（タイプ：tcp port）トンネル サーバーのポート番号。<br>• **target**：（タイプ：文字列、最大 64 バイト）ターゲット ID は文字列です。ユーザーが ID を予約済みキーワード「HOSTNAME」に設定すると、スイッチはスイッチのホスト名でターゲットを置き換えます。<br>• **type**：（タイプ：文字列、制限 64 バイト）タイプは、10.3.2F リリースの GNMI_GNOI のみをサポートします。<br>• **use-vrf**：（タイプ：文字列）スイッチが grpc トンネル セッションのダイヤル アウトに使用する VRF 名文字列。<br>• **（オプション）source-interface**：（タイプ：インターフェイス名文字列）source-interface は、トンネル確立の際の出力送信元 IP アドレスを決定するために使用されます。設定すると、スイッチは、インターフェイスの最初の ipv6 グローバルユニキャストアドレスを選択します。それ以外の場合は、インターフェイスの ipv4 ユニキャスト アドレスを選択します。この設定 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | は、ループバックおよび svi インターフェイスのみをサポートします。インターフェイスは、Lo10、Vlan100 などの短縮名形式で指定する必要があります。<br>• **（オプション）target-vrf**：（タイプ：文字列）ローカル grpc サーバー ターゲットに到達するため、指定した VRF 名を使用します。指定しない場合は、vrf パラメータと同じ名前を使用します。たとえば、grpctunnel を use-vrf foo のように指定したとすると、target-vrf bar は、スイッチが vrf foo 内の外部トンネルサーバへの接続を確立するものの、着信 grpc 要求は vrf bar 内に存在するローカルスイッチ grpc サーバに転送することを意味します。 |

# トラストポイントを使用した gRPC トンネルの構成

gRPC トンネルの場合、 Cisco NX-OS デバイスが指定された外部 gRPC トンネル接続先への接続を開始することに注意してください。ユーザーは、このようなアウトバウンド接続を保護するためにトラストポイント/証明書を構成できます。

- 「cert」オプションを使用したサーバ認証：これにより、外部接続先の証明書がスイッチに構成されます。構成された証明書がリモートトンネルと一致しない場合、スイッチは接続を拒否します。
- 「client-cert」オプションを使用したクライアント認証：これにより、スイッチのアイデンティティ証明書が構成されます。スイッチが一致する証明書を提示できない場合、リモートトンネルサーバはスイッチからの接続を拒否します。
- 相互認証は、「cert」認証と「client-cert」認証の両方を組み合わせたものです。

以下の手順では、手順 1 ～ 3 は「サーバ認証」をインポートすることを意図し、手順 4 ～ 5 は「クライアント認証」をインポートすることを意図しています。ユーザーは、いずれかまたは両方を有効にする適切な組み合わせを決定できます。

（注） クライアント認証用のルート証明書を構成または削除すると、gRPC トンネルは、リモート接続先への接続を再起動します。

（注） クライアントの証明書が中間 CA によって署名されているが、上記の構成からインポートされたルート CA によって直接署名されていない場合、grpc トンネル証明書（ステップ 5）は、ユーザー、中間 CA 証明書、およびルート CA 証明書を含む完全な証明書チェーンを提供する必要があります。

### 始める前に

サーバ認証に必要な証明書ファイルを準備して署名します。これは gRPC トンネルに固有のものではないため、既存のトラストポイントファイルを再利用できます。

このセクションは、特に「cert」および「client-cert」オプションの使用方法を明確にすることを意図しています。前のセクションで説明したオプションは、これら 2 つのオプションと自由に組み合わせることができます。

## 手順の概要

1. **configure terminal**
2. （任意） **crypto ca trustpoint** <*tunnel-trustpoint*>
3. （任意） **rsakeypair** <*client-key*>
4. （任意） **crypto ca authenticate** <*tunnel-trustpoint*>
5. （任意） **crypto ca trustpoint** <tunnel-client-trustpoint>
6. （任意） **crypto ca import** <*tunnel-client-trustpoint*> **pkcs12 bootflash** *:*<*ca-file*> <*pkcs-password*>
7. **[no] grpctunnel destination …**

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`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 | （任意） **crypto ca trustpoint** <*tunnel-trustpoint*><br>例：<br>`switch# crypto ca trustpoint tunnel_trustpoint` | トンネルサーバ認証用のトラストポイントを作成します。 |
| ステップ 3 | （任意） **rsakeypair** <*client-key*><br>例：<br>`switch# rsakeypar key` | クライアントトラストポイントの rsa キーペアを生成します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 4 | （任意） **crypto ca authenticate** <*tunnel-trustpoint*><br>例：<br>`switch# crypto ca authenticate tunnel_trustpoint` | トンネルサーバ証明書をインポートします。この手順では、手動でコピーして貼り付ける必要があります。指示に従ってください。 |
| ステップ 5 | （任意） **crypto ca trustpoint** <tunnel-client-trustpoint><br>例：<br>`switch# crypto ca trustpoint tunnel_client_trustpoint` | トンネルクライアント認証用のトラストポイントを作成します。 |
| ステップ 6 | （任意） **crypto ca import** <*tunnel-client-trustpoint*> **pkcs12 bootflash** *:<ca-file> <pkcs-password>*<br>例：<br>`switch# crypto ca import tunnel_client_trustpoint pkcs12 bootflash:ca.pfx test` | pkcs12 ファイルをトラストポイントにインポートします。 |
| ステップ 7 | **[no] grpctunnel destination …**<br>例：<br>`switch(config)# grpctunnel destination 1.1.1.1 port 8000 target test1 type GNMI_GNOI cert tunnel_trustpoint client-cert tunnel_client_trustpoint …` | トンネル接続を構成します。トンネル接続を削除するには、このコマンドの no 形式を使用します。<br>• **[オプション] cert** -（タイプ：文字列）トンネルサーバ証明書を保持するトラストポイント。指定しない場合、サーバーの検証はスキップされます。<br>• （**オプション**）**client-cert**：（タイプ：文字列）クライアント証明書を保持するトラストポイント。指定した場合、 スイッチはトンネル サーバーとの相互認証を実行します。 |

# VRF による gRPC トンネルの構成

以下の手順では、手順 1 ～ 3 は「サーバ認証」をインポートすることを意図し、手順 4 ～ 5 は「クライアント認証」をインポートすることを意図しています。ユーザーは、いずれかまたは両方を有効にする適切な組み合わせを決定できます。

### 始める前に

このセクションでは、特に「use-vrf」オプションと「target-vrf」オプションの使用方法を明確にします。前のセクションで説明したオプションは、これら 2 つのオプションと自由に組み合わせることができます。

### 手順の概要

1. **configure terminal**
2. **[no] feature grpctunnel**

3. **[no] grpctunnel destination …**

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch-1# 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 | **[no] feature grpctunnel**<br>例：<br>`switch(config)# feature grpctunnel` | grpc-tunnel 機能を有効または無効にします。 |
| ステップ 3 | **[no] grpctunnel destination …**<br>例：<br>`switch(config)# switch(config)# grpctunnel destination 1.1.1.1 port 8000 target test1 type GNMI_GNOI use-vrf management target-vrf default` | トンネル接続を構成します。トンネル接続を削除するには、このコマンドの no 形式を使用します。<br>• **use-vrf**：（タイプ：文字列）スイッチが grpc トンネル セッションのダイヤル アウトに使用する VRF 名文字列。<br>• **（オプション）target-vrf**：（タイプ：文字列）ローカル grpc サーバー ターゲットに到達するため、指定した VRF 名を使用します。指定しない場合は、vrf パラメータと同じ名前を使用します。たとえば、grpctunnel を use-vrf foo のように指定したとすると、target-vrf bar は、スイッチが vrf foo 内の外部トンネルサーバへの接続を確立するものの、着信 grpc 要求は vrf bar 内に存在するローカルスイッチ grpc サーバに転送することを意味します。 |

### 例

このセクションでは、トンネルの使用方法を示すいくつかの構成例を示します。

#### 認証なし

次の手順では、サーバ検証を行わずにトンネルの接続先を構成する方法について説明します。

```
switch(config)# grpctunnel destination 1.1.1.1 port 8000 target test1 type GNMI_GNOI
use-vrf management
switch(config)# grpctunnel destination server.foo.com port 8000 target test2 type GNMI_GNOI
 use-vrf management
```

この例では、ユーザーは2つのトンネル接続先「1.1.1.1:8000」と「server.foo.com:8000」を、それぞれターゲット「test1」と「test2」として構成します。接続は、管理名前空間を介して開始されます。

### サーバ認証あり

次の手順では、サーバ検証を行ってトンネルの接続先を構成する方法について説明します。

次のコマンドを実行して、トラストポイントにサーバ証明書をインポートします。

```
switch(config)# crypto ca trustpoint tunnel_server_trustpoint switch(config-trustpoint)#
 crypto ca authenticate tunnel_server_trustpoint
input (cut & paste) CA certificate (chain) in PEM format; end the input with a line
containing only END OF INPUT :
-----BEGIN CERTIFICATE-----
MIIC3TCCAcWgAwIBAgIJAO4xEeL+IrpuMA0GCSqGSIb3DQEBCwUAMBcxFTATBgNV
BAMMDHNqYy1hZHMtNjAxNDAeFw0yMjA1MjYwMDE4MzBaFw0zMjA1MjMwMDE4MzBa
MBcxFTATBgNVBAMMDHNqYy1hZHMtNjAxNDCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBALudrG824XmW/4+BNd632CT3x47akV0QfjwAU1xBDScpAw9brERO
YTLP9BxInbA+WAS+zGql6nmBoZxbqZZL/NVD8ltLKYJJxtDQHJkqdX2lURnMUFr2
9pyJQtuh/udq9hp8zGcEpbPayfIdHCnZqraWMLvk1W0mqAa7ek0iizIZNwKmU3oR
7CGQOxi8aMsAfH5iBsRTNURFdaXdJYTOjry0il+jBKT21F2Z3vGcB7ddTt+I7qrd
GjJs4BI4a22Y3usYb/dnsEa0ZCFTFIq6Y2Pwc3DOuKalUhujSqisqfMDuqC34ATw
kWwLnHDWVu0iVaWndy3uvQZKDNv/bIIuoo8CAwEAAaMsMCowFwYDVR0RBBAwDoIM
c2pjLWFkcy02MDE0MA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEB
AIjNgq/paYfPtHDe9PlZKzrmGz+UlUAx8saj2WHtrKgBj48J6fYvz1yTPWLKMPct
/5y+nhia6gRlV/navFcpiUUpQGpoZQnaa40/nkBMDvVxnTu6l9UC0WUAyTh2l7ec
BriY8yq3elpQWHZS4KRNmBH8fuviAv4f0fzOAuNGeIuv7UGnfA8Ed/q/Z3frQxOI
qNXr3vBBTptYTLwdrRM0axagL6waZgZyTFfFHpIXBPEtsXKb/5GuP4+nqXvtfkfe
d6P9jA4BKA/e6Gu6NAR0JMOdmJeEFjMbg+uu8jghcRTcwRsGeb9DqPUL+5IsVg3a dKMaZxyQFiRz0LyTqQtZmE0=
 -----END CERTIFICATE-----
END OF INPUT
Fingerprint(s): SHA1 Fingerprint=D4:9D:79:5B:8B:38:D6:50:6D:46:89:A8:C4:41:AB:
C9:D9:9F:D1:66
Do you accept this certificate? [yes/no]:yes
```

その後、次のコマンドを実行して、トンネルの接続先を構成します。

また、show コマンドを使用して設定をディスプレイします。

```
switch(config)# grpctunnel destination 1.1.1.1 port 8000 target test1 type GNMI_GNOI
use-vrf
management cert tunnel_server_trustpoint

switch(config)# show system internal dme running-config all dn sys/grpctunnel {
"grpctunnelInst": {
"attributes": {
"childAction": "",
"dn": "sys/grpctunnel",
"modTs": "2022-12-02T12:57:37.891+00:00",
"status": ""
},
"children": [
{
"grpctunnelTunnelMgr": {
"attributes": {
"childAction": "",
"dn": "sys/grpctunnel/tunnelmgr",
"modTs": "2022-12-02T12:57:37.891+00:00",
"status": ""
},
"children": [
```

```
{
"grpctunnelTunnel": {
"attributes": {
"cert": "tunnel_server_trustpoint",
"certClient": "",
"childAction": "", "dest": "1.1.1.1", "dn":
"sys/grpctunnel/tunnelmgr/tunnel-[1.1.1.1]-port-[8000]-target-[test1]-type-[GNMI_GNOI]-vrf-[management]",
"modTs": "2022-12-05T10:09:45.163+00:00",
"port": "8000",
"srcIf": "unspecified",
"status": "",
"targetId": "test1",
"targetType": "GNMI_GNOI",
"targetVrf": "",
"vrf": "management"
}
}
}
]
}
}
]
}
}
```


### クライアント認証あり

次の手順では、クライアント検証を行ってトンネルの接続先を構成する方法について説明します。

次の手順では、サーバ検証を行わずにトンネルの接続先を構成する方法について説明します。

```
switch(config)# crypto ca trustpoint tunnel_client_trustpoint
switch(config)# crypto ca import tunnel_client_trustpoint pkcs12 bootflash://ca.pfx test
```

その後、次のコマンドを実行して、トンネルの接続先を構成します。

また、show コマンドを使用して設定をディスプレイします。

```
switch(config)# grpctunnel destination 1.1.1.1 port 8000 target test1 type GNMI_GNOI
use-vrf
management client-cert tunnel_client_trustpoint

switch(config)# show system internal dme running-config all dn sys/grpctunnel {
"grpctunnelInst": {
"attributes": {
"childAction": "",
"dn": "sys/grpctunnel",
"modTs": "2022-12-02T12:57:37.891+00:00",
"status": ""
},
"children": [
{
"grpctunnelTunnelMgr": {
"attributes": {
"childAction": "",
"dn": "sys/grpctunnel/tunnelmgr",
"modTs": "2022-12-02T12:57:37.891+00:00",
"status": ""
},
"children": [
{
```

```
"grpctunnelTunnel": {
"attributes": {
"cert": "",
"certClient": "tunnel_client_trustpoint ",
"childAction": "", "dest": "1.1.1.1", "dn":
"sys/grpctunnel/tunnelmgr/tunnel-[1.1.1.1]-port-[8000]-target-[test1]-type-[GNMI_GNOI]-vrf-[management]",
"modTs": "2022-12-05T10:09:45.163+00:00",
"port": "8000",
"srcIf": "unspecified",
"status": "",
"targetId": "test1",
"targetType": "GNMI_GNOI",
"targetVrf": "",
"vrf": "management"
}
}
}
]
}
}
]
}
}
```


### VRF あり

「use-vrf」と「target-vrf」構成を組み合わせると、導入の柔軟性は高くなりますが、混乱が生じる可能性もあります。

次の違いに留意してください。

- use-vrf：リモートトンネルの接続先に到達する方法。
- target-vrf：スイッチの内部サービスにトンネル トラフィックを転送/リレーする方法。

次のシナリオの例を参照してください。

- リモートトンネルサーバは、「管理」vrfを経由して到達可能です。スイッチがトンネル内の gNMI 接続を受信すると、スイッチは gnmi「管理」サーバに転送します。

  ```
  grpctunnel destination server1 port 9000 target target2 type GNMI_GNOI vrf management
  ```

- ローカル grpcエージェントがデフォルトの vrf で実行されている間、リモートトンネルサーバは「管理」vrfを経由して到達可能です。次の構成では、スイッチがトンネル内の gNMI 接続を受信すると、スイッチは gnmi 要求をデフォルト vrf に送信します。

  ```
  grpc use-vrf default
  grpctunnel destination server1 port 9000 target target2 type GNMI_GNOI use-vrf
  management target-vrf default
  ```

- リモートトンネルサーバとローカル grpc エージェントの両方が、デフォルトの vrf で実行されます。

  ```
  grpc use-vrf default
  grpctunnel destination server1 port 9000 target target2 type GNMI_GNOI use-vrf default
  ```

- リモートトンネルサーバは「デフォルト」vrfを経由して到達可能ですが、ローカル grpc エージェントは「test」vrf で実行されています。次の構成では、スイッチがトンネル内の gNMI 接続を受信すると、スイッチは gnmi 要求を test vrf に送信します。

```
grpc use-vrf test
grpctunnel destination server1 port 9000 target target2 type GNMI_GNOI vrf default
local-vrf test
```

- この場合、ローカル grpc エージェントが「abc」vrfで実行されている間、リモートトンネルサーバは「デフォルト」vrfを経由して到達可能です。次の構成では、スイッチがトンネル内の gNMI 接続を受信すると、スイッチは gnmi 要求を test vrf に送信するため、接続は機能しません。これは前方参照として扱うことができます。grpc config を「grpc use-vrf test」に変更すると、接続は機能するようになります。

```
grpc use-vrf abc
grpctunnel destination server1 port 9000 target target2 type GNMI_GNOI vrf default
local-vrf test
```

# 障害対応

### 機能ステータスの確認

- Cisco NX-OS で、**show feature grpctunnel** コマンドを入力してエージェントの構成を確認します。
- gRPC トンネルのステータスを表示するには、**show feature** コマンドを使用します。

```
switch-1# show feature | grep grpctunnel
restconf 1 enabled
switch-1#
```

### gRPC トンネルのデバッグ

トンネルのステータスを表示する一連の show コマンドがあります。

### コマンドの表示

トンネルの構成/ステータスを確認するには、次のコマンドを入力します。

| コマンド | 説明 |
|---|---|

| | |
|---|---|
| show grpctunnel internal sessions [all] { summary \| detail } } | トンネルのステータスを表示しま<br>• 「sessions」オプションは、ト<br>を表示します。<br>「all」オプションを使用すると、終<br>終了したセッションについては、<br>を保持します。 |
| debug grpctunnel events all | CLI EXEC モードで実行します。<br>これにより、コンソールにデバッ |

### 出力例

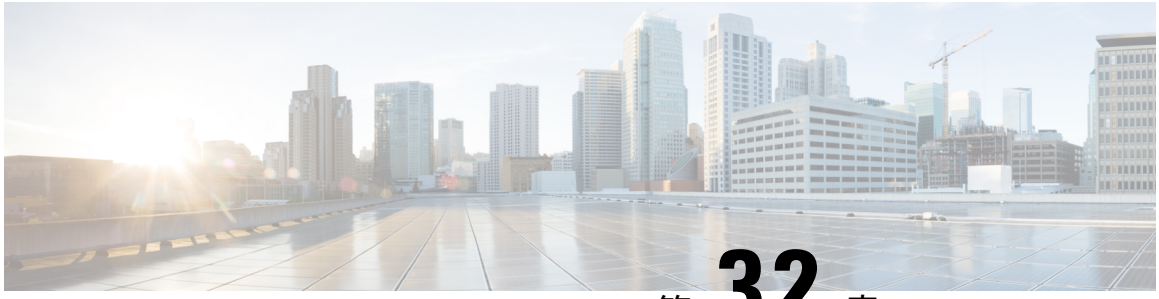
**show grpctunnel internal sessions summary**

```
=============
gRPC Tunnel
=============
Restart Count : 1
* - history
Destination                                                  Target/Type
 Cnt Retry  Cnt Sess   Status/Error
------------------------------------------------------------ ------------------------------
 ---------- ---------- ------------------------------
1.1.1.1:8080 (management)                                   test/GNMI_GNOI
         0          0 NOT CONNECTED - Dialing [1.1.1.1]:8080
```

### gRPC トンネルログの収集

/volatile ディレクトリには、grpcトンネルログが格納されます。

```
bash-4.3# cd /volatile/ bash-4.3# ls /volatile -al
…
-rw-rw-rw- 1 root root 103412 Jun 21 16:14 grpc-internal-tunnel-log
…
```



第 32 章

# テレメトリ


- テレメトリについて （515 ページ）
- テレメトリのコンポーネントと用語 （516 ページ）
- テレメトリ プロセスの高可用性 （516 ページ）
- テレメトリのライセンス要件 （517 ページ）
- テレメトリの注意事項と制限事項 （517 ページ）
- CLI を使用したテレメトリの構成 （524 ページ）
- NX-API を使用したテレメトリの構成 （547 ページ）
- クラウド スケール ソフトウェア テレメトリ （563 ページ）
- テレメトリ パス ラベル （565 ページ）
- ネイティブ データ送信元パス （585 ページ）
- ストリーミング Syslog （598 ページ）
- テレメトリのトラブルシューティング （603 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。

# テレメトリのコンポーネントと用語

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、またはパスのブランチにある YANG インフラから収集されます。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 プロトコルバッファ（GPB）と JSON 形式でエンコードします。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エンコーダに必要です。
- **データトランスポート**：NX-OS は、JSON エンコードには HTTP を使用し、GPB エンコードには Google リモートプロシージャコール（gRPC）プロトコルを使用して、テレメトリデータを転送します。gRPC レシーバーは、4 MB を超えるメッセージ サイズをサポートします。（証明書が構成されている場合は、HTTPS を使用したテレメトリ データもサポートされます。）
- **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。データストリームを正しく受信してデコードするには、受信側に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テレメトリの `.proto` ファイルは、Cisco の GitLab で入手できます：https://github.com/CiscoDevNet/nx-telemetry-proto

（注） テレメトリのコンテキストでは、gRPC は特定の独自仕様 .proto を指します。標準規格の gNMI/gNOI サービスをホストする「gRPC エージェント」と混同しないでください。

# 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**：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を復元します。
- **スーパーバイザフェールオーバー**：テレメトリがホットスタンバイではなくても、新しいアクティブなスーパーバイザが実行されているときに、テレメトリ構成とストリーミングサービスを復元します。

- **プロセスの再起動**：なんらかの理由でテレメトリプロセスがフリーズまたは再起動した場合、再起動時に、構成およびストリーミング サービスを復元します。

# テレメトリのライセンス要件

表 *42*：テレメトリのライセンス要件

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

# テレメトリの注意事項と制限事項

テレメトリ 構成時の注意事項および制約事項は、次のとおりです。

- 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。

- NX-OS リリース 10.4(1)F より前のリリースでは、DME 構成 MO の時間プロパティは現地時間で印刷されますが、タイム ゾーンのオフセットは常に 00.00 です。たとえば、時刻は 2024-05-21T14:01:03.012+00:00 のように表示されます。

  NX-OS リリース 10.5(1)F 以降、currentTime、modTs などの DME 構成 MO のタイムスタンプは、localtime-timezone_offset 形式で表示されます。たとえば、時刻は 2024-05-21T14:01:03.012-04:00 のように表示されます。

- Cisco NX-OS リリース 10.4(2)F 以降、テレメトリは Cisco Nexus 93400LD-H1 プラットフォームスイッチでサポートされます。
- Cisco NX-OS リリース 10.4(3)F 以降、テレメトリは 92348GC-X でサポートされます。
- Cisco NX-OS リリース 10.4(2)F 以降、テレメトリは Cisco Nexus N9KC9364C-H1 プラットフォームスイッチでサポートされます。
- Cisco NX-OS リリース 10.6(1)F 以降、テレメトリは Cisco Nexus N9336C-SE1 プラットフォーム スイッチでサポートされます。
- Cisco NX-OSリリース 10.6(1)F 以降、Cisco Live Protect 機能による NX- OSのセキュアリングでは、nxsecure の新しいセンサーパスを使用してテレメトリ機能を使用してエクスポートできるイベント ログがサポートされています。

### 古いリリースにダウングレードした後の構成コマンド

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

次の例は、この手順を表示しています。

- テレメトリ構成をファイルにコピーします。

```
switch# show running-config | section telemetry
feature telemetry
telemetry
destination-group 100
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB use-chunking size 4096
sensor-group 100 path sys/bgp/inst/dom-default depth 0
subscription 600
dst-grp 100
snsr-grp 100 sample-interval 7000
switch# show running-config | section telemetry > telemetry_running_config switch#
show file bootflash:telemetry_running_config
feature telemetry
telemetry
destination-group 100
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB use-chunking size 4096
sensor-group 100 path sys/bgp/inst/dom-default depth 0
subscription 600
dst-grp 100
snsr-grp 100 sample-interval 7000
switch#
```

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

```
switch# copy telemetry_running_config running-config echo-commands
`switch# config terminal`
`switch(config)# feature telemetry`
`switch(config)# telemetry`
`switch(config-telemetry)# destination-group 100`
`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB `
`switch(conf-tm-dest)# sensor-group 100`
`switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
`switch(conf-tm-sensor)# subscription 600`
`switch(conf-tm-sub)# dst-grp 100`
`switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000`
`switch(conf-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete. switch#
```

### gRPC エラーの動作

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

一部のエラーの内容は、次のとおりです。

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

### gRPC チャンキングのサポート

Cisco NX-OSは、gRPC チャンクをサポートします。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、次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。

```
switch# 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します。

   ```
   show
           <command> | json
   ```

   つまり `show <command> | json pretty` のようにします。

（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. show コマンドを調整して、フィルタまたはオプションを含めます。

   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。show vlan id 100、show vlan id 101 などです。パフォーマンスを改善するため、可能であれば、代わりに CLI 範囲オプションを使用してください。show vlan id 100-110,204 などです。

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

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

### テレメトリの VRF サポート

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

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

次の例では、トランスポート VRF を指定しています。

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"
}
}
}
]
}
}
```

与えられた構成で、特定の IP アドレスとポートの組み合わせでデータのルーティングを行うために使用できる VRF は 1 つだけです。使用される VRF は、最後に行われた vrf 構成によって決定されます。

次に例を示します。

```
telemetry
  destination-group 1
    ip address 91.1.1.1 port 50007
    use-vrf default
  destination-group 2
    ip address 91.1.1.1 port 50007
    use-vrf test
  sensor-group 1
    data-source DME
    path sys/fm
  subscription 1
    dst-grp 1
    dst-grp 2
    snsr-grp 1 sample-interval 1000
```

上記の構成では、destination-group 1 と destination-group 2 の両方に同じ宛先 IP アドレスとポート（91.1.1.1 50007）が設定されていますが、異なる VRF が構成されています（default と test）。このシナリオでは、destination-group 2 が後から構成されていた場合、91.1.1.1 50007 に送信されるデータは、**test** VRF を使用してルーティングされます。destination-group に VRF が構成されていない場合は、暗黙的に**管理** VRF が使用されます。

### 証明書トラストポイント サポート

テレメトリは trustpoint キーワードをサポートします。

次にあるのは、コマンド シンタックスです。

```
switch(config-telemetry)# certificate ?
trustpoint    specify trustpoint label
WORD          .pem certificate filename (Max Size 256)
switch(config-telemetry)# certificate trustpoint
WORD          trustpoint label name (Max Size 256)
switch(config-telemetry)# certificate trustpoint trustpoint1 ?
WORD  Hostname associated with certificate (Max Size 256)
switch(config-telemetry)#certificate trustpoint trustpoint1 foo.test.google.fr
```

### 接続先ホスト名サポート

テレメトリは、destination-group コマンドで **host** キーワードをサポートしています。

次に、接続先ホスト名のサポートの例を示します。

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ?
certificate Specify certificate
host Specify destination host
ip Set destination IPv4 address
ipv6 Set destination IPv6 address
...
switch(conf-tm-dest)# host ?
A.B.C.D|A:B::C:D|WORD  IPv4 or IPv6 address or DNS name of destination
switch(conf-tm-dest)#
switch(conf-tm-dest)# host abc port 11111 ?
protocol  Set transport protocol
switch(conf-tm-dest)# host abc port 11111 protocol ?
HTTP
UDP
gRPC
switch(conf-tm-dest)# host abc port 11111 protocol gRPC ?
encoding  Set encoding format
switch(conf-tm-dest)# host abc port 11111 protocol gRPC encoding ?
Form-data    Set encoding to Form-data only
GPB          Set encoding to GPB only
GPB-compact  Set encoding to Compact-GPB only
JSON         Set encoding to JSON
XML          Set encoding to XML
switch(conf-tm-dest)# host ip address 1.1.1.1 port 2222 protocol HTTP encoding JSON
<CR>
```

### ノード識別子のサポート

テレメトリは、 **use-nodeid** コマンドを使用して、テレメトリ受信者のカスタム Node ID 文字列をサポートします。デフォルトではホスト名が使用されますが、ノード ID のサポートにより、テレメトリ受信者データの `node_id_str` 識別子を設定または変更できます。

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

次の例は、ノード識別子の構成を表示しています。

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

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

```
Telemetry receiver:
==================================
node_id_str: "test-srvr-10"
subscription_id_str: "1" encoding_path:
"sys/ch/psuslot-1/psu" collection_id:
3896 msg_timestamp: 1559669946501
```

**host** コマンドの下の **use-nodeid** サブコマンドを使用します。接続先レベルの **use-nodeid** 構成は、グローバルレベルの構成よりも優先されます。次の例はコマンドシンタックスを表示します。

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# use-nodeid ?
WORD  Node ID (Max Size 128)
switch(conf-tm-dest-host)# use-nodeid session_1:18112
```

テレメトリ 受信者の出力の例を表示します：

```
>> Message size 923
Telemetry msg received @ 23:41:38 UTC Msg Size: 11
node_id_str : session_1:18112 collection_id : 3118
data_source : DME
encoding_path : sys/ch/psuslot-1/psu collection_
start_time : 1598485314721
collection_end_time : 1598485314721
data :
```

### YANG モデルのストリーミングのサポート

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。

### プロキシのサポート

テレメトリは、ホストコマンドで **proxy** キーワードをサポートします。次にあるのは、コマンド シンタックスです。

```
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# host 172.19.216.78 port 18112 protocol http enc json
switch(conf-tm-dest-host)# proxy ?
A.B.C.D|A:B::C:D|WORD IPv4 or IPv6 address or DNS name of proxy server
<1-65535> Proxy port number, Default value is 8080 username Set proxy authentication
username
password Set proxy authentication password
```

### gRPC 非同期モード

gRPC 非同期モードは、**host** コマンドでのみ使用できます。通常のストリーム状態では、このモードにおいて、受信者は **WriteDone()** 呼び出しを終了したり、受信したりしなくても、**mdtDialout** 呼び出しでデータをストリーミングできます。

次にあるのは、コマンド シンタックスです。

```
nxosv-1(config-telemetry)# destination-group 1
nxosv-1(conf-tm-dest)# host 172.22.244.130 port 50007 ?
nxosv-1(conf-tm-dest-host)# grpc-async ?
```

### センサ グループの単一テレメトリ収集

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

- この設定はイベント サブスクリプションには使用できません。
- 接続先グループとマージサブスクリプションのフィルタファイル設定は、相互に互換性がありません。NX-OS ではブロックされません。その場合、収集はすべてのセンサーグループに対して個別に行われます。これは以前のリリースと同じです。
- サンプル間隔が大きいサブスクリプションのデータ送信のサンプル間隔は、min_sample_interval*floor(cur_sample_interval /min_sample_interval) の式を使用して決定されます。
  - min_sample_interval は、すべてのサブスクリプションのセンサー グループの最小サンプル間隔です。
  - cur_sample 間隔は、そのセンサー グループの特定のサブスクリプションのサンプル間隔です。
- このオプションは以前のリリースでは使用できません。互換性チェックでの失敗を回避するには、ダウングレードする前にこの設定を削除してください。

# CLI を使用したテレメトリの構成

## CLI による構成

次の手順では、ストリーミング テレメトリを有効にし、データ ストリームの送信元と接続先を構成します。

### 手順の概要

1. **configure terminal**
2. **feature telemetry**

3. **feature nxapi**
4. **nxapi use-vrf management**
5. **telemetry**
6. **[no] merge-subscriptions**
7. （任意） **certificate** *certificate_path host_URL*
8. **sensor-group** *sgrp_id*
9. **path** *sensor_path* **depth unbounded [filter-condition** *filter]* **[alias** *path_alias]*
10. （任意） **data-source native**
11. （任意） **path event-nxsecure**
12. **destination-group** *dgrp_id*
13. （任意） **ip address** *ip_address* port *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
14. （任意） **ipv6 address** *ipv6_address port* **port protocol** *procedural-protocol* **encoding** *encoding-protocol*
15. （任意） **source-interface** *interface*
16. *ip_version* **address ip_address port** *portnum*
17. （任意） **use-chunking size** *chunking_size*
18. **subscription sub_id**
19. **snsr-grp** *sgrp_id* **sample-interval** *interval*
20. **dst-grp** *dgrp_id*

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal` | グローバル構成モードを開始します。 |
| ステップ 2 | **feature telemetry** | ストリーミング テレメトリ機能を有効にします。 |
| ステップ 3 | **feature nxapi** | NX-API を有効にします。 |
| ステップ 4 | **nxapi use-vrf management**<br>例：<br>`switch(config)#`<br>`switch(config)# nxapi use-vrf management`<br>`switch(config)` | NX-API 通信に使用する VRF 管理を有効にします。<br>（注）<br>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。<br><br>警告<br>警告: 構成された管理 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。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | （注）<br>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：<br><br>警告<br>警告：非管理 VRF で構成された 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 5 | **telemetry**<br>例：<br>`switch# telemetry`<br>`switch(config-telemetry)#` | ストリーミング テレメトリの構成モードに入ります。 |
| ステップ 6 | **[no] merge-subscriptions**<br>例：<br>`switch# merge-subscriptions` | 複数のサブスクリプションに含まれるすべてのセンサー グループに対して 1 つのコレクションを作成します。 |
| ステップ 7 | （任意） **certificate** *certificate_path host_URL*<br>例：<br>`switch# certificate /bootflash/server.key localhost` | 既存の SSL/TLS 証明書を使用します。<br>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。 |
| ステップ 8 | **sensor-group** *sgrp_id*<br>例：<br>`switch# sensor-group 100`<br>`switch(config-telemetry)#` | ID srgp_id を持つセンサー グループを作成し、センサー グループ構成モードを開始します。<br>英数字の識別子がサポートされています。センサーグループでは、テレメトリ レポートのモニタリング対象ノードを定義します。 |
| ステップ 9 | **path** *sensor_path* **depth unbounded [filter-condition** *filter]* **[alias** *path_alias]*<br>例：<br>• 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：<br>`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(l2BD.operSt, "down")`<br>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。<br>`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition` | 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。<br>（注）<br>これは、データ ソース DME パスにのみ適用されます。<br><br>センサー グループにセンサー パスを追加します。<br>• Cisco NX-OS 9.3(5) リリース以降では、alias キーワードが導入されています。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))`<br>UTR 側のパスを区別するには、次の構文を使用します。<br>`switch(conf-tm-sensor)# path sys/ch/ftslot-1/ft alias ft_1`<br>• 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：<br>`switch(conf-tm-sensor)# path "show interface" depth 0`<br>• 次のコマンドは、デバイス YANG に適用されます。<br>`switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items`<br>• 次のコマンドは、OpenConfig YANG に適用されます。<br>`switch(conf-tm-sensor)# path openconfig-bgp:bgp`<br>`switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items alias bgp_alias`<br>• 次のコマンドは、NX-API に適用されます：<br>`switch(conf-tm-sensor)# path "show interface" depth 0 alias sh_int_alias`<br>• 次のコマンドは、OpenConfig に適用されます。<br>`switch(conf-tm-sensor)# path openconfig-bgp:bgp alias oc_bgp_alias` | • [深さ（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、および無制限の深さ設定がサポートされています。<br>（注）<br>深さ 0 がデフォルトの深さです。NX-API ベースのセンサーパスは、深さ 0 のみを使用できます。<br>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。<br>• 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に特定のフィルタを作成できます。<br>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] フィルタ条件は、operSt が変更されたとき、および operSt が down のままである間に 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。<br>• 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。<br>• デバイスYANG の場合：Cisco-NX-OS-device:System/bgp-items/inst-items<br>• OpenConfig YANG の場合：openconfig-bgp:bgp<br>（注）<br>depth、filter-condition、および and query-condition パラメータは、現在 YANG ではサポートされていません。<br>openconfig YANG モデルの場合は、https://github.com/YangModels/yang/tree/master/vendor/cisco/nx に移動して、最新リリースの適切なフォルダに移動します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | 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。<br>次に例を示します。<br>install add mtx-openconfig-bgp-1.0.0.0.0-7.0.3.IHD8.1.lib32_n9000.rpm activate |
| ステップ 10 | （任意） **data-source native**<br>例：<br>`switch(conf-tm-sensor)# data-source native` | 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 11 | （任意） **path event-nxsecure**<br>例：<br>`switch(conf-tm-sensor)# path event-nxsecure`<br>`switch(conf-tm-sensor)# path event-history`<br>`switch(conf-tm-sensor)# path event-monitor` | テレメトリ パス センサー タイプにより、ログ ファイルを外部の受信者にエクスポートできます。このコマンドは、テレメトリ パス センサー タイプを、次の 3 つのイベント ベースのパスのいずれかとして設定します。<br>• `event-nxsecure` ：nxsecure ログ ファイルをリモート https サーバに転送できるようにします。ファイルは https を介して転送されます。<br>• `event-history` ：履歴イベント ログを維持および設定します<br>• `event-monitor` ：リアルタイムのイベント データ ストリームを統合します。 |
| ステップ 12 | **destination-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 100` | 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。<br>英数字の識別子がサポートされています。 |
| ステップ 13 | （任意） **ip address** *ip_address* port *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*<br>例：<br>`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`<br>`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON` | エンコードされたテレメトリ データを受信する IPv4 IP アドレスとポートを指定します。<br>（注）<br>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** *ipv6_address port* **port protocol** *procedural-protocol* **encoding** *encoding-protocol*<br>例： | 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。<br>（注） |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB`<br>`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON`<br>`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON` | gRPC はデフォルトのトランスポート プロトコルです。GPB がデフォルトのエンコーディングです。 |
| **ステップ 15** | （任意） **source-interface** *interface*<br>**例**：<br>`switch(conf-tm-sensor)# source-interface vlan1500` | 接続先 IP アドレスのテレメトリ送信元インターフェイスを指定します。<br>• 接続先 IP：ポートごとに、1 つの送信元インターフェイスのみがサポートされます。このような接続先 IP：ポートに対しては、最後に構成された送信元インターフェイスが使用されます。<br>• 送信元インターフェイス構成には、接続先グループで use-vrf CLI を使用して、そのインターフェイスごとの vrf を指定する必要があります。use-vrf が指定されていない場合は、管理 vrf が使用されます。 |
| **ステップ 16** | *ip_version* **address ip_address port** *portnum*<br>**例**：<br>IPv4 の場合：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`<br>IPv6 の場合：<br>`switch(conf-tm-dest)# ipv6 address 10:10::1 port 8000` | 発信データの宛先プロファイルを作成します。ip_version は、ip（IPv4 の場合）または ipv6（IPv6 の場合）です。<br>接続先グループがサブスクリプションにリンクされている場合、テレメトリ データは、このプロファイルで指定されている IP アドレスとポートに送信されます。 |
| **ステップ 17** | （任意） **use-chunking size** *chunking_size*<br>**例**：<br>`switch(conf-tm-dest)# use-chunking size 64` | gRPC チャンクを有効にして、チャンク サイズを 64～4096 バイトに設定します。詳細については、「gRPC チャンクのサポート」セクションを参照してください。 |
| **ステップ 18** | **subscription sub_id**<br>**例**：<br>`switch(conf-tm-dest)# subscription 100`<br>`switch(conf-tm-sub)#` | ID を持つサブスクリプション ノードを作成し、サブスクリプション構成モードを開始します。<br>現在、sub_id は、数字の ID 値のみをサポートしています。<br>**（注）**<br>DN にサブスクライブする場合は、イベントが確実にストリーミングされるように、その DN が REST を使用して DME でサポートされているかどうかを確認します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 19 | **snsr-grp** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000` | ID sgrp_id のセンサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。<br>間隔の値が 0 の場合、イベント ベースのサブスクリプションが作成され、テレメトリ データは、指定された MO での変更時にのみ送信されます。0 より大きい間隔値の場合、テレメトリ データが指定された間隔で定期的に送信される頻度に基いたサブスクリプションが作成されます。たとえば、間隔値が 15000 の場合、テレメトリ データは 15 秒ごとに送信されます。 |
| ステップ 20 | **dst-grp** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 100` | ID dgrp_id を持つ接続先グループをこのサブスクリプションにリンクします。 |

# YANG パスの頻度の設定

YANG パスの頻度は、合計ストリーミング時間よりも長くする必要があります。合計ストリーミング時間と頻度が正しく構成されていない場合、テレメトリ データの収集にストリーミング間隔よりも長くかかることがあります。この状況では、次のことがわかります。

- テレメトリ データが受信側へのストリーミングよりも速く蓄積されるため、徐々に満たされるキュー。
- 現在の間隔からではない古いテレメトリ データ。

合計ストリーミング時間よりも大きい値に頻度を構成します。

## 手順の概要

1. **show telemetry control database sensor-groups**
2. **sensor group** *number*
3. **subscription** *number*
4. **snsr-grp** *number* **sample-interval** *milliseconds*
5. **show system resources**

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **show telemetry control database sensor-groups**<br><br>例：<br>`switch# show telemetry control database sensor-groups`<br>`Sensor Group Database size = 2`<br>`----------`<br>`Row ID  Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions  SubID`<br>`----------`<br>`1  2  Timer  /YANG  5000  /Running  1  1`<br><br>`Collection Time in ms (Cur/Min/Max): 2444/2294/2460`<br>`Encoding Time in ms (Cur/Min/Max): 56/55/57`<br>`Transport Time in ms (Cur/Min/Max): 0/0/1`<br>`Streaming Time in ms (Cur/Min/Max): 2515/2356/28403`<br><br>`Collection Statistics:`<br>`  collection_id_dropped = 0`<br>`  last_collection_id_dropped = 0`<br>`  drop_count = 0`<br><br>`2  1  Timer  /YANG  5000  /Running  1  1`<br><br>`Collection Time in ms (Cur/Min/Max): 144/142/1471`<br>`Encoding Time in ms (Cur/Min/Max): 0/0/1`<br>`Transport Time in ms (Cur/Min/Max): 0/0/0`<br>`Streaming Time in ms (Cur/Min/Max): 149/147/23548`<br>`Collection Statistics:`<br>`  collection_id_dropped = 0`<br>`  last_collection_id_dropped = 0`<br>`  drop_count = 0`<br><br>`switch#`<br>`telemetry`<br>`  destination-group 1`<br>`  ip address 192.0.2.1 port 9000 protocol HTTP encoding JSON`<br>`  sensor-group 1`<br>`  data-source YANG`<br>`  path /Cisco-NX-OS-device:System/procsys-items depth unbounded`<br>`  sensor-group 2`<br>`  data-source YANG`<br>`  path /Cisco-NX-OS-device:System/intf-items/phys-items depth unbounded`<br>`  subscription 1`<br>`  dst-grp 1` | 合計ストリーミング時間を計算します。<br><br>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。<br><br>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。<br><br>頻度はサンプル間隔で表示されます。この例では、合計ストリーミング時間（2.664秒）がケイデンス（デフォルトの5.000秒）よりも短いため、頻度は正しく構成されています。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `snsr-grp 1 sample-interval 5000`<br>`snsr-grp 2 sample-interval 5000` | |
| ステップ 2 | **sensor group** *number*<br>例：<br>`switch(config-telemetry)# sensor group1` | 合計ストリーミング時間がその頻度以上の場合、間隔を設定したいセンサーグループを入力します。 |
| ステップ 3 | **subscription** *number*<br>例：<br>`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*<br>例：<br>`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。<br>この例では、サンプル間隔は 5.000 秒に設定されています。これは、2.664 秒の合計ストリーミング時間よりも長いため、有効です。 |
| ステップ 5 | **show system resources**<br>例：<br>`switch# show system resources`<br>`Load average:  1 minute: 0.38  5 minutes: 0.43  15 minutes: 0.43`<br>`Processes:  555 total, 3 running`<br>`CPU states  :  24.17% user,  4.32% kernel, 71.50% idle`<br>`CPU0 states:  0.00% user,  2.12% kernel, 97.87% idle`<br>`CPU1 states:  86.00% user,  11.00% kernel,  3.00% idle`<br>`CPU2 states:  8.08% user,  3.03% kernel, 88.88% idle`<br>`CPU3 states:  0.00% user,  1.02% kernel, 98.97% idle`<br>`Memory usage:  16400084K total,  5861652K used, 10538432K free`<br>`Current memory status: OK` | CPUの使用状況を確認してください。この例に示すように、CPUユーザー状態が高い使用率を示している場合、頻度とストリーミング値が正しく構成されていません。この手順を繰り返して、頻度を正しく設定します。 |

# CLI を使用したテレメトリの構成例

### 単一のテレメトリ DME ストリームの構成

ケイデンス 10 秒で GPB エンコーディングを使用します。

```
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)# source-interface vlan1500
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
```

### 実行構成の表示

```
switch# show running-config telemetry
telemetry
destination-group 1
ip address 171.70.59.62 port 50051 protocol gRPC encoding GPB
source-interface Vlan1500
sensor-group sg1
data-source DME
subscription 1
dst-grp 1
snsr-grp sg1 sample-interval 10000
```

### sys/bgp ルート MO の登録

5 秒ごとに接続先 IP 1.2.3.4、ポート50003 に送信します。

```
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
```

### sys/intf MO の登録

- 5 秒ごとに接続先 IP 1.2.3.4、ポート50003 に送信します。
- 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
```

### sys/cdp MOの登録

15 秒ごとに接続先 IP 1.2.3.4、ポート50004 へ

```
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
```

### show コマンドのケイデンスベースの収集の登録

750 秒ごと。

```
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
```

### sys/fm へのイベントベースサブスクリプションの登録

sys/fm MO に変更がある場合にのみストリーミング

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

動作中に、サンプル間隔を変更することで、センサー グループを周波数ベースからイベントベースに変更したり、イベントベースから周波数ベースに変更したりできます。この例では、センサー グループを前の例から頻度ベースに変更します。次のコマンドの後、テレメトリ アプリケーションは 7 秒ごとに sys/fm データの接続先へのストリーミングを開始します。

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

### 複数のセンサーおよび接続先グループへの登録

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

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

```
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol HTTP encoding JSON
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

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

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

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

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

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

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

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

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

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

### UDP トランスポートへの登録

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 }
Example for an IPv6 destination:
destination-group 100 ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB
```

UDP テレメトリには次のヘッダーがあります。

```
typedef enum tm_encode_ {
TM_ENCODE_DUMMY,
TM_ENCODE_GPB,
TM_ENCODE_JSON,
TM_ENCODE_XML,
TM_ENCODE_MAX, } tm_encode_type_t;
typedef struct tm_pak_hdr_ {
uint8_t version; /* 1 */ uint8_t encoding; uint16_t msg_size; uint8_t secure; uint8_t
padding;
}__attribute__ ((packed, aligned (1))) tm_pak_hdr_t;
```

次のいずれかの方法で、ペイロードの最初の6バイトを使用して、UDPを使用してテレメトリデータを処理します。

- 受信側が複数のエンドポイントから異なるタイプのデータを受信することになっている場合は、ヘッダーの情報を読んで、データのデコードに使用するデコーダー（JSONまたはGPB）を決定します。
- 1つのデコーダー（JSONまたはGPB）が必要で、もう1つのデコーダーは必要ない場合は、ヘッダーを削除します。

### テレメトリ構成の確認

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
```

## Telemetry Merge サブスクリプションの構成例

次に、merge-subscriptionを構成する例を示します。

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

```
  subscription 2
    dst-grp 2
    snsr-grp 1 sample-interval 25000
```

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

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

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

### 構成の確認

次のコマンドを使用して、IOA 構成を確認します。

```
show telemetry control database sensor-groups
Sensor Group Database size = 1
Row ID     Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
  SubID
1          1                Timer   /DME            10000/Running      2
      1
Collection Time in ms (Cur/Min/Max): 1/1/2
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 10003/10003/10003
Streaming Time in ms (Cur/Min/Max): 10004/10004/10006
Collection Statistics:
collection_id_dropped      = 0
last_collection_id_dropped = 0
drop_count                 = 0
Configuration method: CONFIG_DME-ADMIN
2          1                Timer   /DME            20000*/Running      2
       2
Collection Time in ms (Cur/Min/Max): 1/1/1
Encoding Time in ms (Cur/Min/Max): 0/0/0
Transport Time in ms (Cur/Min/Max): 4004/4004/4004
Streaming Time in ms (Cur/Min/Max): 4005/4005/4005
Collection Statistics:
collection_id_dropped      = 0
last_collection_id_dropped = 0
drop_count                 = 0
Configuration method: CONFIG_DME-ADMIN
*Calculated sample interval for merge-subscriptions
```

# テレメトリの構成と統計情報の表示

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

### show telemetry yang direct-path cisco-nxos-device

このコマンドは、他のパスよりもパフォーマンスが向上するように直接エンコードされた YANG パスを表示します。

```
switch# show telemetry yang direct-path cisco-nxos-device
) Cisco-NX-OS-device:System/lldp-items
```

```
2) Cisco-NX-OS-device:System/acl-items
3) Cisco-NX-OS-device:System/mac-items
4) Cisco-NX-OS-device:System/intf-items
5) Cisco-NX-OS-device:System/procsys-items/sysload-items
6) Cisco-NX-OS-device:System/ospf-items
7) Cisco-NX-OS-device:System/procsys-items
8) Cisco-NX-OS-device:System/ipqos-items/queuing-items/policy-items/out-items
9) Cisco-NX-OS-device:System/mac-items/static-items
10) Cisco-NX-OS-device:System/ch-items
11) Cisco-NX-OS-device:System/cdp-items
12) Cisco-NX-OS-device:System/bd-items
13) Cisco-NX-OS-device:System/eps-items
14) Cisco-NX-OS-device:System/ipv6-items
```

### show telemetry control database

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 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
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 sys/bgp(1) : NA : NA
4 No 1 0 Self
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 version(2) : NA : NA
```

### show telemetry control stats

このコマンドは、テレメトリの構成についての内部データベースの統計を表示します。

```
switch# show telemetry control stats
show telemetry control stats entered

-------------------------------------------------------------------------------
Error Description                                          Error Count
-------------------------------------------------------------------------------
Chunk allocation failures                                  0
Sensor path Database chunk creation failures               0
Sensor Group Database chunk creation failures              0
Destination Database chunk creation failures               0
```

```
Destination Group Database chunk creation failures          0
Subscription Database chunk creation failures               0
Sensor path Database creation failures                      0
Sensor Group Database creation failures                     0
Destination Database creation failures                      0
Destination Group Database creation failures                0
Subscription Database creation failures                     0
Sensor path Database insert failures                        0
Sensor Group Database insert failures                       0
Destination Database insert failures                        0
Destination Group Database insert failures                  0
Subscription insert to Subscription Database failures       0
Sensor path Database delete failures                        0
Sensor Group Database delete failures                       0
Destination Database delete failures                        0
Destination Group Database delete failures                  0
Delete Subscription from Subscription Database failures     0
Sensor path delete in use                                   0
Sensor Group delete in use                                  0
Destination delete in use                                   0
Destination Group delete in use                             0
Delete destination(in use) failure count                    0
Failed to get encode callback                               0
Sensor path Sensor Group list creation failures             0
Sensor path prop list creation failures                     0
Sensor path sec Sensor path list creation failures          0
Sensor path sec Sensor Group list creation failures         0
Sensor Group Sensor path list creation failures             0
Sensor Group Sensor subs list creation failures             0
Destination Group subs list creation failures               0
Destination Group Destinations list creation failures       0
Destination Destination Groups list creation failures       0
Subscription Sensor Group list creation failures            0
Subscription Destination Groups list creation failures      0
Sensor Group Sensor path list delete failures               0
Sensor Group Subscriptions list delete failures             0
Destination Group Subscriptions list delete failures        0
Destination Group Destinations list delete failures         0
Subscription Sensor Groups list delete failures             0
Subscription Destination Groups list delete failures        0
Destination Destination Groups list delete failures         0
Failed to delete Destination from Destination Group         0
Failed to delete Destination Group from Subscription        0
Failed to delete Sensor Group from Subscription             0
Failed to delete Sensor path from Sensor Group              0
Failed to get encode callback                               0
Failed to get transport callback                            0
switch#  Destination Database size = 1

-------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
-------------------------------------------------------------------------------
192.168.20.123 50001      GPB        gRPC       1
```

### show telemetry data collector brief

このコマンドは、データ収集に関する簡単な統計情報を表示します。

```
switch# show telemetry data collector brief
-----------------------------------------------------------------------
 Collector Type Successful Collections Failed Collections
-----------------------------------------------------------------------
 DME 143 0
```

### show telemetry data collector details

このコマンドは、すべてのセンサーパスの詳細を含む、データ収集に関する詳細な統計情報を表示します。

- **[成功（Successful）]**：完了した収集の数を反映します。成功数には、成功した収集の数のみが含まれます。試行回数は含まれません。収集が成功しても、必ずしもデータが返されるわけではありません。無効なパス、 OC モデルがインストールされていない、データ収集のエラーなどの場合、失敗とされます。データがないことは、失敗とは見なされません。
- **[ペイロード（Payload）]**：収集ごとのペイロードの数を反映します。
- **[失敗（Failed）]**：失敗した収集またはペイロードの数を反映します。
- **[スキップ（Skipped）]**：スキップされた収集の数を反映します。収集は、アグレッシブなサンプル間隔やシステムリソースの不足が原因でスキップされる場合があります。
- **[ドロップ済み（Dropped）]**：ドロップされたペイロードの数を反映します。ペイロードは、システムリソースまたはトランスポートの状態が原因でドロップされる可能性があります。
- **[センサーパス（Sensor Path）]**：センサーパスを表示します。

```
switch# show telemetry data collector details
-----------------------------------------------------------------------------------------------------
Row ID         Successful     Payloads   Failed      Skipped        Dropped        Sensor
 Path(GroupId)
-----------------------------------------------------------------------------------------------------
1              3              6120       0           0              0
Cisco-NX-OS-device:System/intf-items(1)
```

### 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



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 *43 : show telemetry transport* の構文の説明

| 構文 | 説明（Description） |
|---|---|
| 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 |
| *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 |
| errors | （オプション）すべてのテレメトリ エラー |
| 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 |
| *encoding_type* | （オプション）エンコーディング タイプ |
| *transport_type* | （オプション）トランスポート タイプ |
| *transport_status* | （オプション）トランスポート ステータス |
| *transport_security_cert_fname* | （オプション）トランスポート セキュリテ |
| *transport_last_connected* | （オプション）最後に接続されたトランス |
| *transport_last_disconnected* | （オプション）この接続先構成が最後に削 |
| *transport_errors_count* | （オプション）トランスポート エラー数 |
| *transport_last_tx_error* | （オプション）トランスポートの最後の tx |
| transport_statistics | （オプション）トランスポート統計情報 |
| *t_session_id* | （オプション）トランスポート セッション |

| | |
|---|---|
| connect_statistics | （オプション）接続統計情報 |
| *connect_count* | （オプション）接続数 |
| *last_connected* | （オプション）最終接続のタイムスタンプ |
| *Disconnect_count* | （オプション）切断数 |
| *last_disconnected* | （オプション）この接続先構成が最後に削除さ |
| trans_statistics | （オプション）トランスポート統計情報 |
| *compression* | （オプション）圧縮ステータス |
| *source_interface_name* | （オプション）送信元インターフェイス名 |
| *source_interface_ip* | （オプション）送信元インターフェイス IP |
| *transmit_count* | （オプション）送信数 |
| *last_tx_time* | （オプション）最終送信時刻 |
| *min_tx_time* | （オプション）最小送信時間 |
| *max_tx_time* | （オプション）最大送信時間 |
| *avg_tx_time* | （オプション）平均送信時間 |
| *cur_tx_time* | （オプション）現在の送信時間 |
| transport_errors | （オプション）トランスポート エラー |
| connect_errors | （オプション）接続エラー |
| *connect_errors_count* | （オプション）接続エラー数 |
| trans_errors | （オプション）トランスポート エラー |
| *trans_errors_count* | （オプション）トランスポート エラー数 |
| *last_tx_error* | （オプション）最後のトランスポート エラー |
| *last_tx_return_code* | （オプション）最後のトランスポート戻りコー |
| transport_retry_stats | （オプション）再試行統計情報 |
| *ts_event_retry_bytes* | （オプション）イベント再試行バッファ サイ |
| *ts_timer_retry_bytes* | （オプション）タイマー再試行バッファ サイ |
| *ts_event_retry_size* | （オプション）メッセージのイベント再試行回 |
| *ts_timer_retry_size* | （オプション）タイマー再試行メッセージ回数 |
| *ts_retries_sent* | （オプション）送信された再試行回数 |
| *ts_retries_dropped* | （オプション）ドロップされた再試行回数 |
| *event_retry_bytes* | （オプション）イベント再試行バッファ サイ |

| | |
|---|---|
| *timer_retry_bytes* | （オプション）タイマー再試行バッファ サ |
| *retries_sent* | （オプション）送信された再試行回数 |
| *retries_dropped* | （オプション）ドロップされた再試行回数 |
| *retry_buffer_size* | （オプション）再試行バッファ サイズ |

### show telemetry transport <session-id>

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

```
switch# show telemetry transport 0

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Fri Sep 02 11:45:57.505 UTC

Tx Error Count:      224
Last Tx Error:       Fri Sep 02 12:23:49.555 UTC

switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235
Encoding:            JSON
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never

Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
The following example shows output from an IPv6 entry.
switch# show telemetry transport 0
Session Id: 0
IP Address:Port [10:10::1]:8000
Transport: GRPC
Status: Idle
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
Event Retry Queue Bytes: 0
Event Retry Queue Size: 0
Timer Retry Queue Bytes: 0
Timer Retry Queue Size: 0
Sent Retry Messages: 0
Dropped Retry Messages: 0
```

### show telemetry transport <session-id> stats

次に、特定の転送セッションの詳細のコマンドを示します。

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
```

```
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

### show telemetry transport <session-id> errors

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

```
switch# show telemetry transport 0 errors

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

### show telemetry control databases sensor-paths

これらの構成手順が、**show telemetry control databases sensor-paths** コマンドの出力に表示されます。

```
below. feature telemetry
telemetry
destination-group 1
ip address 172.25.238.13 port 50600 protocol gRPC encoding GPB
sensor-group 1
path sys/cdp depth unbounded path sys/intf depth unbounded path sys/mac depth 0
subscription 1
dst-grp 1 snsr-grp 1 sample-interval 1000 Command output. switch# show telemetry control
 databases sensor-paths
Sensor Path Database size = 3
--------------------------------------------------------------------------------
--------------------
Row ID Subscribed Linked Groups Sec Groups Retrieve level Path(GroupId) : Query : Filter
--------------------------------------------------------------------------------
--------------------
1 No 1 0 Full sys/cdp(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 30489/30489/30489
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 6/5/54
Encoding Time in ms (Cur/Min/Max): 5/5/6
Transport Time in ms (Cur/Min/Max): 1027/55/1045
Streaming Time in ms (Cur/Min/Max): 48402/5/48402
2 No 1 0 Full sys/intf(1) : N
A : NA
GPB Encoded Data size in bytes (Cur/Min/Max): 539466/539466/539466
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 66/64/114
Encoding Time in ms (Cur/Min/Max): 91/90/92
Transport Time in ms (Cur/Min/Max): 4065/4014/5334
Streaming Time in ms (Cur/Min/Max): 48365/64/48365
```

```
 3 No 1 0 Self sys/mac(1) : NA
: NA
GPB Encoded Data size in bytes (Cur/Min/Max): 247/247/247
JSON Encoded Data size in bytes (Cur/Min/Max): 0/0/0
CGPB Encoded Data size in bytes (Cur/Min/Max): 0/0/0
Collection Time in ms (Cur/Min/Max): 1/1/47
Encoding Time in ms (Cur/Min/Max): 1/1/1
Transport Time in ms (Cur/Min/Max): 4/1/6
Streaming Time in ms (Cur/Min/Max): 47369/1/47369
```

#### show telemetry transport sessions

次のコマンドは、すべてのトランスポートセッションをループし、1つのコマンドで情報を出力します。

```
switch# show telemetry transport sessions
switch# show telemetry transport stats
switch# show telemetry transport errors
switch# show telemetry transport all
The following is an example for telemetry transport session:
switch# show telemetry transport sessions
Session Id:          0
IP Address:Port      172.27.254.13:50004
Transport:           GRPC
Status:              Transmit Error
SSL Certificate:     trustpoint1
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      2
Last Tx Error:       Wed Aug 19 23:32:21.749 UTC
…
Session Id:          4
IP Address:Port      172.27.2
```

#### テレメトリ エフェメラル イベント

エフェメラル イベントをサポートするために、新しいセンサー パス クエリ条件が追加されました。アカウンティング ログの外部イベント ストリーミングを有効にするには、次のクエリ条件を使用します。

```
sensor-group 1 path sys/accounting/log query-condition
query-target=subtree&complete-mo=yes&notify-interval=1

The following are the other sensor paths that support ephemeral event:
sys/pim/inst/routedb-route, sys/pim/pimifdb-adj, sys/pim/pimifdb-prop
sys/igmp/igmpifdb-prop, sys/igmp/inst/routedb, sys/igmpsnoop/inst/dom/db-exptrack,
sys/igmpsnoop/inst/dom/db-group, sys/igmpsnoop/inst/dom/db-mrouter
sys/igmpsnoop/inst/dom/db-querier, sys/igmpsnoop/inst/dom/db-snoop
```

# NX-API を使用したテレメトリの構成

## DME のテレメトリ モデル

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

```
model
|----package [name:telemetry]
```

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

```
|----reldef
     |  @name:DestGroupRel
     |    @to:DestGroup
     |    @cardinality:ntom
     |    @label:Link to destGroup entry
```

# NX- API を使用した構成

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**：モニター対象のパス。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*』を参照してください。

### 始める前に

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

## 手順の概要

1. テレメトリ機能を有効にします。
2. テレメトリ構成を記述するために、JSON ペイロードのルート レベルを作成します。
3. 定義されたセンサー パスを含むセンサーグループを作成します。
4. （任意） SSL/TLS 証明書とホストを追加します。
5. テレメトリの接続先グループを定義します。
6. テレメトリの接続先プロファイルを定義します。
7. テレメトリ データの送信先となる IP アドレスとポート番号で構成される、1 つ以上のテレメトリの接続先を定義します。
8. gRPC チャンクを有効にして、チャンク サイズを 64 ～ 4096 バイトに設定します。
9. テレメトリサブスクリプションを作成して、テレメトリの動作を構成します。
10. ルート要素の下の telemetrySubscription 要素に子オブジェクトとしてセンサー グループ オブジェクトを追加します（telemetryEntity）。
11. サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ センサー グループに関連付け、データ サンプリング動作を指定します。
12. テレメトリをモニタリングする 1 つ以上のセンサー パスまたはノードを定義します。
13. センサーパスを子オブジェクトとしてセンサーグループオブジェクト（telemetrySensorGroup）に追加します。
14. 接続先を子オブジェクトとして接続先グループオブジェクト（telemetryDestGroup）に追加します。
15. 接続先グループオブジェクトを子オブジェクトとしてルート要素に追加します（telemetryEntity）。
16. センサー グループをサブスクリプションに関連付けるために、テレメトリ センサー グループの子オブジェクトとして関係オブジェクトを作成します。
17. テレメトリ接続先グループの子オブジェクトとして関係オブジェクトを作成して、接続先グループをサブスクリプションに関連付けます。
18. サブスクリプションの子オブジェクトとして関係オブジェクトを作成して、サブスクリプションをテレメトリ接続先グループに関連付けます。

19. テレメトリ構成のために、結果の JSON 構造を HTTP/HTTPS POST ペイロードとして NX-API エンドポイントに送信します。

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| **ステップ 1** | テレメトリ機能を有効にします。<br>**例：**<br>`{ "fmEntity" : { "children" : [{ "fmTelemetry" : { "attributes" : { "adminSt" : "enabled" } } } ] } }` | ルート要素は fmTelemetry であり、この要素のベース パスは `sys/fm` です。adminSt属性を `enabled` 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。<br>**例：**<br>`{ "telemetryEntity": { "attributes": { "dn": "sys/tm" }, } }` | ルート要素は telemetryEntity で、この要素のベースパスは sys/tm です。dn 属性を sys/tm として構成します。 |
| **ステップ 3** | 定義されたセンサー パスを含むセンサーグループを作成します。<br>**例：**<br>`"telemetrySensorGroup": { "attributes": { "id": "10", "rn": "sensor-10" }, "children": [{ }] }` | 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。<br>• 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。<br>• rm：センサーグループオブジェクトの相対名（形式：sensor-id）。<br>• dataSrc：DEFAULT、DME、YANG、または NX- API の中からデータ送信元を選択します。<br>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 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | 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。<br>**例**：<br>{<br>"telemetryCertificate": {<br>"attributes": {<br>"filename": "root.pem"<br>"hostname": "c.com"<br>}<br>}<br>} | telemetryCertificate は、テレメトリの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。<br>**例**：<br>{<br>"telemetryDestGroup": {<br>"attributes": {<br>"id": "20"<br>}<br>}<br>} | テレメトリ接続先グループは telemetryEntity で定義されます。id 属性を構成します。 |
| **ステップ 6** | テレメトリの接続先プロファイルを定義します。<br>**例**：<br>{<br>"telemetryDestProfile": {<br>"attributes": {<br>"adminSt": "enabled"<br>},<br>"children": [<br>{<br>"telemetryDestOptSourceInterface":<br>{<br>"attributes": {<br>"name": "lo0"<br>}<br>}<br>}<br>]<br>}<br>} | 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。<br>• adminSt属性を `enabled` に構成します。<br><br>telemetryDestOptSourceInterface の下で、構成されたインターフェイスから接続先に、ソース IP アドレスととも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。<br>**例**：<br><br>{<br>"telemetryDest": {<br>"attributes": {<br>"addr": "1.2.3.4", | 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。<br>• addr：接続先の IPアドレス。<br>• port：接続先のポート番号。<br>• rn：path-[path] 形式の接続先オブジェクトの相対名。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `"enc": "GPB",`<br>`"port": "50001",`<br>`"proto": "gRPC",`<br>`"rn": "addr-[1.2.3.4]-port-50001"`<br>`}`<br>`}`<br>`}` | • 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。<br>• gRPC の Google Protocol Buffer（GBP）。<br>• C の JSON。<br>• 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。<br>• gRPC<br>• HTTP<br>• 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。<br>• HTTP/JSON はい<br>• HTTP/Form-data はい Bin Logging でのみサポートされます。<br>• GRPC/GPB-Compact はい ネイティブデータ ソースのみ。<br>• GRPC/GPB はい<br>• UDP/GPB はい<br>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。<br>**例**：<br>`{`<br>`"telemetryDestGrpOptChunking": {`<br>`"attributes": {`<br>`"chunkSize": "2048",`<br>`"dn": "sys/tm/dest-1/chunking"`<br>`}`<br>`}`<br>`}` | 詳細については、「gRPC チャンク」セクションを参照してください。https://www.cisco.com/c/en/us/td/docs/dcn/nx-os/nexus9000/103x/programmability/cisco-nexus-9000-series-nx-os-programmability-guide-release-103x/m-n9k-model-driven-telemetry-101x.html#Cisco_Reference.dita_a97d6406-43b4-4c8d-bb24-f33bb187f126 |
| **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。<br>**例**：<br>`"telemetrySubscription": {` | 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。<br>• 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。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `"attributes": {`<br>`"id": "30",`<br>`"rn": "subs-30"`<br>`},  "children": [{`<br>`}]`<br>`}` | • 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。<br><br>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）。<br><br>**例**：<br><br>`{`<br>`"telemetrySubscription": {`<br>`"attributes": {`<br>`"id": "30"`<br>`}`<br>`"children": [{`<br>`"telemetryRsSensorGroupRel": {`<br>`"attributes": {`<br>`"sampleIntvl": "5000",`<br>`"tDn": "sys/tm/sensor-10"`<br>`}`<br>`}`<br>`}`<br>`]`<br>`}`<br>`}` | |
| **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。<br><br>**例**：<br><br>`"telemetryRsSensorGroupRel": {`<br>`"attributes": {`<br>`"rType": "mo",`<br>`"rn": "rssensorGroupRel-[sys/tm/sensor-10]",`<br>`"sampleIntvl": "5000",`<br>`"tCl": "telemetrySensorGroup",`<br>`"tDn": "sys/tm/sensor-10",`<br>`"tType": "mo"`<br>`}`<br>`}` | 関係オブジェクトはクラス telemetryRsSensorGroupRel に属しており、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。<br><br>• rn：rssensorGroupRel-[sys/tm/sensor-group-id] 形式の関係オブジェクトの相対名。<br><br>• sampleIntvl — ミリ秒単位のデータサンプリング期間。間隔の値が 0 の場合、イベント ベースのサブスクリプションが作成され、テレメトリ データは、指定された MO での変更時にのみ送信されます。0 より大きい間隔値の場合、テレメトリ データが指定された間隔で定期的に送信される頻度に基いたサブスクリプションが作成されます。たとえば、間隔値が 15000 の場合、テレメトリ データは 15 秒ごとに送信されます。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | | • tCl：ターゲット（センサーグループ）オブジェクトのクラス。telemetrySensorGroup です。<br>• tDn：ターゲット（センサーグループ）オブジェクトの識別名。sensor-group-id です。<br>• rType：関係タイプ。管理対象オブジェクトの mo です。<br>tType: ターゲットタイプ。管理対象オブジェクトの mo です。 |
| **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。<br>**例：**<br>単一センサー パス<br>{<br>"telemetrySensorPath": {<br>"attributes": {<br>"path": "sys/cdp",<br>"rn": "path-[sys/cdp]",<br>"excludeFilter": "",<br>"filterCondition": "",<br>"path": "sys/fm/bgp",<br>"secondaryGroup": "0",<br>"secondaryPath": "",<br>"depth": "0"<br>}<br>}<br>}<br>**例：**<br>複数のセンサー パス<br>{<br>"telemetrySensorPath": {<br>"attributes": {<br>"path": "sys/cdp",<br>"rn": "path-[sys/cdp]",<br>"excludeFilter": "",<br>"filterCondition": "",<br>"path": "sys/fm/bgp",<br>"secondaryGroup": "0",<br>"secondaryPath": "",<br>"depth": "0"<br>}<br>}<br>},<br>{<br>"telemetrySensorPath": {<br>"attributes": { | 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。<br>• path：モニタリングされるパス。<br>• rn：path-[path] 形式のパスオブジェクトの相対名：<br>• depth：センサーパスの取得レベル。depth 設定が 0 の場合、ルート MO プロパティのみを取得します。<br>filterCondition：（オプション）イベントベースのサブスクリプション用に特定のフィルタを作成します。DME はフィルター式を提供します。フィルタリングの詳細については、クエリの作成に関する Cisco APIC REST API の使用ガイドラインを参照してください。次の Cisco APIC ドキュメントのランディングページで確認できます。<br>https://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/tsd-products-support-series-home.html |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | <pre>"excludeFilter": "",<br>"filterCondition": "",<br>"path": "sys/fm/dhcp",<br>"secondaryGroup": "0",<br>"secondaryPath": "",<br>"depth": "0"<br>}<br>}<br>}</pre><br>**例**：<br>BGP 無効化イベントの単一センサー パス フィルタリング：<br><pre>{<br>"telemetrySensorPath": {<br>"attributes": {<br>"path": "sys/cdp",<br>"rn": "path-[sys/cdp]",<br>"excludeFilter": "",<br>"filterCondition":<br>"eq(fmBgp.operSt.\"disabled\")",<br>"path": "sys/fm/bgp",<br>"secondaryGroup": "0",<br>"secondaryPath": "",<br>"depth": "0"<br>}<br>}<br>}</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。<br>**例**：<br><pre>"telemetryRtSensorGroupRel": {<br>"attributes": {<br>"rn":<br>"rtsensorGroupRel-[sys/tm/subs-30]",<br>"tCl": "telemetrySubscription",<br>"tDn": "sys/tm/subs-30"<br>}<br>}</pre> | 関係オブジェクトはクラス telemetryRtSensorGroupRel に属しており、 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。<br>• rn：rtsensorGroupRel-[sys/tm/subscription-id] 形式の関係オブジェクトの相対名。<br>• tCl：サブスクリプションオブジェクトのターゲットクラス。telemetrySubscription です。<br>tDn：サブスクリプションオブジェクトのターゲット識別名。sys/tm/subscription-id です。 |

| | 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。<br>**例**：<br>`"telemetryRtDestGroupRel": {`<br>`"attributes": {`<br>`"rn": "rtdestGroupRel-[sys/tm/subs-30]",`<br>`"tCl": "telemetrySubscription",`<br>`"tDn": "sys/tm/subs-30"`<br>`}`<br>`}` | 関係オブジェクトはクラス telemetryRtDestGroupRel に属しており、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。<br>• rn：rtdestGroupRel-[sys/tm/subscription-id] 形式の関係オブジェクトの相対名。<br>• tCl：サブスクリプションオブジェクトのターゲットクラス。telemetrySubscription です。<br>tDn：サブスクリプションオブジェクトのターゲット識別名。sys/tm/subscription-id です。 |
| **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。<br>**例**：<br>`"telemetryRsDestGroupRel": {`<br>`"attributes": {`<br>`"rType": "mo",`<br>`"rn": "rsdestGroupRel-[sys/tm/dest-20]",`<br>`"tCl": "telemetryDestGroup",`<br>`"tDn": "sys/tm/dest-20",`<br>`"tType": "mo"`<br>`}`<br>`}` | 関係オブジェクトはクラス telemetryRsDestGroupRel に属しており、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。<br>• rn：rsdestGroupRel-[sys/tm/destination-group-id] 形式の関係オブジェクトの相対名。<br>• tCl：ターゲット（接続先グループ）オブジェクトのクラス。telemetryDestGroup です。<br>• tDn：ターゲット（接続先グループ）オブジェクトの識別名。sys/tm/destination-group-id です。<br>• rType：関係タイプ。管理対象オブジェクトの mo です。<br>tType: ターゲットタイプ。管理対象オブジェクトの mo です。 |
| **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。<br>**例**： | テレメトリ エンティティのベース パスは `sys/tm` で、NX-API エンドポイントは次のとおりです。<br>`{{URL}}/api/node/mo/sys/tm.json` |

**例**

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

```
{
    "telemetryEntity": {
```

```
                "children": [{
                    "telemetrySensorGroup": {
                        "attributes": {
                            "id": "10"
                           }
                           "children": [{
                            "telemetrySensorPath": { "attributes": {

                            "excludeFilter": "",
                            "filterCondition": "", "path": "sys/fm/bgp",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                                                                "depth": "0"
                           }
                          }
                         }
                         ]
                     }
                    },
                    {
                        "telemetryDestGroup": {
                            "attributes": {
                               "id": "20"
                             }
                             "children": [{
                                "telemetryDest": {
                                                                "attributes": {
                                    "addr": "10.30.217.80",
                                                                  "port": "50051",
                                                                  "enc": "GPB",
                                                                  "proto": "gRPC"
                                  }
                                 }
                               }
                             ]
                           }
                        },
                        {
                           "telemetrySubscription": {
                               "attributes": {
                                   "id": "30"
                                }
                                "children": [{
                                  "telemetryRsSensorGroupRel": {
                                                                 "attributes": {
                                      "sampleIntvl": "5000",
                                      "tDn": "sys/tm/sensor-10"
                                    }
                                   }
                                 },
                                 {
                                   "telemetryRsDestGroupRel": {
                                                                "attributes": {
                                       "tDn": "sys/tm/dest-20"
                               }
                             }
                           }
                           ]
                        }
                    }
                    ]
            }
        }
```

# NX-API を使用したテレメトリの構成例

### 宛先へのストリーミング パス

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

```
POST https://192.168.20.123/api/node/mo/sys/tm.json
Payload:
{
    "telemetryEntity": {
        "attributes": {
            "dn": "sys/tm"
        },
        "children": [{
            "telemetrySensorGroup": {
                "attributes": {
                    "id": "10",
                    "rn": "sensor-10"
                },  "children": [{
                    "telemetryRtSensorGroupRel": {
                        "attributes": {
                            "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
                        "attributes": {
                            "path": "sys/cdp",
                            "rn": "path-[sys/cdp]",
                            "excludeFilter": "",
                            "filterCondition": "",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                            "depth": "0"
                        }
                    }
                }, {
                    "telemetrySensorPath": {
                        "attributes": {
                            "path": "sys/ipv4",
                            "rn": "path-[sys/ipv4]",
                            "excludeFilter": "",
                            "filterCondition": "",
                            "secondaryGroup": "0",
                            "secondaryPath": "",
                            "depth": "0"
                        }
                    }
                }]
            }
        }, {
            "telemetryDestGroup": {
                "attributes": {
                    "id": "20",
```

```
                        "rn": "dest-20"
                    },
                    "children": [{
                        "telemetryRtDestGroupRel": {
                            "attributes": {
                                "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                                "tCl": "telemetrySubscription",
                                "tDn": "sys/tm/subs-30"
                            }
                        }
                    }, {
                        "telemetryDest": {
                            "attributes": {
                                "addr": "1.2.3.4",
                                "enc": "GPB",
                                "port": "50001",
                                "proto": "gRPC",
                                "rn": "addr-[1.2.3.4]-port-50001"
                            }
                        }
                    }]
                }
            }, {
                "telemetrySubscription": {
                    "attributes": {
                        "id": "30",
                        "rn": "subs-30"
                    },
                    "children": [{
                        "telemetryRsDestGroupRel": {
                            "attributes": {
                                "rType": "mo",
                                "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                                "tCl": "telemetryDestGroup",
                                "tDn": "sys/tm/dest-20",
                                "tType": "mo"
                            }
                        }
                    }, {
                        "telemetryRsSensorGroupRel": {
                            "attributes": {
                                "rType": "mo",
                                "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                                "sampleIntvl": "5000",
                                "tCl": "telemetrySensorGroup",
                                "tDn": "sys/tm/sensor-10",
                                "tType": "mo"
                            }
                        }
                    }]
                }
            }]
        }
}
```

### BGP 通知のフィルタ条件

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

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

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50055",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "0",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
```

```
  }
}
```

### テレメトリ構成のための Postman コレクションの使用

テレメトリ機能の構成を開始し、単一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する例については、https://github.com/CiscoDevNet/nx-telemetry-proto/tree/master/postman_collections を参照してください。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。

## マルチキャスト フロー パスの可視性

この機能は、Nexus 3548-XL スイッチで使用可能な必要なすべてのマルチキャスト ステートをエクスポートする手段を提供します。エクスポートにより、各フローが送信元から各受信者までたどるパスの完全で信頼性の高いトレーサビリティが確保されます。

この機能は、DME ですべての適切な情報を公開することを目的としており、プッシュモデル（ソフトウェアテレメトリ）またはプルモデル（DME REST クエリ）のいずれかを介してコンシューマ/コントローラにアクセスできるようにします。

この機能の利点は次のとおりです。

- フロー パスの可視化
- 障害検出のためにフローの統計と状態のエクスポート
- ユーザがフローパス上のスイッチで適切なデバッグコマンドを実行できるようにすることによる根本原因の分析

MFDM は、上位レベルのコンポーネントからの情報を消費し、各マルチキャスト機能のインテリジェンスを構築してから、情報をコンシューマに伝達するマルチキャスト FIB 分散管理です。これは、機能が DME とともに実装されるコアコンポーネントです。MRIB によって提供される情報と MFIB によって収集された統計情報に基づいて、すべてのマルチキャスト ステートを DME にパブリッシュします。

DME は、コンシューマ/コントローラが使用できるようにする必要があるすべての情報を保存するために使用されます。また、イベントベースの通知をサポートするためにオブジェクトが作成、削除、または変更されるたびに、テレメトリへの適切な通知を生成します。

テレメトリプロセスは、DME に保存されているすべてのデータをコンシューマにストリーミングし、データを適切な形式でフォーマットします。

### マルチキャスト フロー パスの可視性のための CLI

次に、マルチキャスト フロー パスの可視性の正確な機能を確認するために導入された CLI を示します。

- DME への情報のエクスポートを有効にするコンフィギュレーション コマンド。この CLI は、システムに存在するすべてのルートに対してこの機能を有効にします。

```
switch(config)# multicast flow-path export

        switch(config)# sh system internal dme run all dn sys/mca/config
```

- MFDM と DME に存在する状態間の整合性チェックを実行する整合性チェッカーの show コマンド。このコマンドを使用すると、特に大規模なセットアップで不整合をすばやく検出できます。

```
switch# show forwarding distribution internal multicast
        consistency-checker flow-path route

Starting flow-path DME consistency-check for VRF:
        default

(0.0.0.0/0, 230.0.0.1/32). Result:
      PASS

(10.0.0.10/32, 230.0.0.1/32). Result:
      PASS

(0.0.0.0/0, 232.0.0.0/8) 結果：パス
```

- グローバル show コマンドを使用して、この機能がシステムで有効になっているかどうかを確認します。

```
switch(config)# show forwarding distribution internal
        multicast global_state

**** MFDM Flow PATH VISIBILITY INFO
      ****

Multicast flow-path info export enabled:
      Y

BE DME ハンドラ：0x117c3e6c

PE DME ハンドラ：0x117b955c

switch(config)# show forwarding distribution internal
        multicast fpv CC

PASS/FAIL (In case of fail, it will highlight the
        inconsistencies)
```

# 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

Cisco NX-OSは、Tahoe ASIC を使用するCisco Nexus クラウドスケールスイッチをサポートします。このプラットフォームで、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* |
| 56 バイト | 各ポートのデータ、合計 56 * *N* バイト。<br>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。<br>• 24 バイトのインターフェイス名<br>• 8 バイトの送信（TX）パケット<br>• 8 バイトの送信（TX）バイト<br>• 8 バイトの受信（RX）パケット<br>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。

- Cisco NX-OS リリース 9.3(x) より前でサポートされるプラットフォームについては、そのガイドの https://www.cisco.com/c/dam/en/us/td/docs/Website/datacenter/platform/platform.html のセクション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るプラットフォームについては、https://www.cisco.com/c/dam/en/us/td/docs/Website/datacenter/platform/platform.html 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。

- Cisco NX-OS リリース 10.3(1)F 以降、ソフトウェアテレメトリは Cisco Nexus 9800 プラットフォームスイッチでサポートされています。

# 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。

- **環境**、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。
- **インターフェイス**、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。

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。
- **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。

## 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 `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` で同じセンサーグループを構成することもできません。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、`interface` パスに対してのみサポートされます。
- この機能は、`depth` や `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。
  - 小文字のプレフィックス **show** で開始する必要があります。大文字と小文字は区別されるからです。

たとえば、**show version** は許可されます。一方、**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。45ページの「データのポーリングまたはイベントの受信」を参照してください。

（注） このモデルは、サブインターフェイス、ループバック、または VLAN のカウンタをサポートしていないため、ストリームアウトされません。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例： | 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 |
|---|---|---|
| | `switch# configure terminal`<br>`switch(config)#` | |
| ステップ 2 | **telemetry**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(config-telemetry)# sensor-group 6`<br>`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**<br>例：<br>`switch(conf-tm-sensor)# path interface`<br>`switch(conf-tm-sensor)#` | インターフェイス パス ラベルを構成して、複数の個々のインターフェイスに対して 1 つのテレメトリ データ クエリを送信できるようにします。ラベルは、複数のインターフェイスのクエリを 1 つに統合します。次に、テレメトリはデータを収集し、ラベルに返します。<br>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*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 9 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 非ゼロ カウンタのインターフェイス パスの構成

ゼロ以外の値を持つカウンターのみを返す事前定義されたキーワードフィルタを使用して、インターフェイス パス ラベルを構成できます。フィルタは `counters=[detailed]` です。

このフィルタを使用することにより、インターフェイス パスは使用可能なすべてのインターフェイスカウンターを収集し、収集したデータをフィルタ処理してから、結果を受信側に転送します。フィルタはオプションであり、使用しない場合、ゼロ値カウンターを含むすべてのカウンターがインターフェイス パスに表示されます。

（注） フィルタの使用は、show interface mgmt0 counters detailed の発行と概念的に似ています。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group***sgrp_id*
4. **path interface query-condition counters=[detailed]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例： | 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 |
|---|---|---|
| | `switch(config)# telemetry`<br>`switch(config-telemetry)#` | |
| ステップ 3 | **sensor-group***sgrp_id*<br>例：<br>`switch(config-telemetry)# sensor-group 6`<br>`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]**<br>例：<br>`switch(conf-tm-sensor)# path interface query-condition counters=[detailed]`<br>`switch(conf-tm-sensor)#` | インターフェイス パス ラベルを構成し、すべてのインターフェイスからのゼロ以外のカウンターのみを照会します。 |
| ステップ 5 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。 |
| ステップ 9 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 動作速度のインターフェイス パスの構成

指定された動作速度のインターフェイスのカウンタを返す定義済みのキーワードフィルタを使用して、インターフェイスパス ラベルを構成できます。フィルタは `oper-speed=[]` です。次の動作速度がサポートされています: auto、10M、100M、1G、10G、40G、200G、および 400G。

このフィルタを使用することにより、インターフェースパスは指定された速度のインターフェースのテレメトリ データを収集し、その結果を受信側に転送します。フィルタはオプションです。使用しない場合、動作速度に関係なく、すべてのインターフェイスのカウンタが表示されます。

フィルタは、複数の速度をコンマ区切りのリストとして受け入れることができます。たとえば、`oper-speed=[1G,10G]` は、1 および 10 Gbps で動作するインターフェイスのカウンタを取得します。区切り文字として空白を使用しないでください。

（注） インターフェイス タイプ サブインターフェイス、ループバック、および VLAN には動作速度プロパティがないため、フィルタはこれらのインターフェイス タイプをサポートしません。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **snsr-group** *sgrp_id* **sample-interval** *interval*
4. **path interface query-condition oper-speed=[speed]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`switch(config-telemetry)#` | テレメトリ機能の構成モードに入ります。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 3 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。 |
| ステップ 4 | **path interface query-condition oper-speed=[speed]**<br>例：<br>`switch(conf-tm-sensor)# path interface query-condition oper-speed=[1G,40G]`<br>`switch(conf-tm-sensor)#` | インターフェイスパスラベルを構成し、指定された速度（この例では 1 Gbps と 40 Gbps のみ）で動作しているインターフェイスからのカウンタを照会します。 |
| ステップ 5 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。 |
| ステップ 9 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 複数のクエリによるインターフェイス パスの構成

インターフェイス パス ラベルの同じクエリ条件に対して複数のフィルタを構成できます。その場合、使用する個々のフィルタは AND で結合されます。

クエリ条件の各フィルタは、コンマを使用して区切ります。query-condition には、任意の数のフィルタを指定できますが、追加するフィルタが多いほど、結果の焦点が絞られることに注意してください。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path interface query-condition counters=[detailed],oper-speed=[1G,40G]**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(config-telemetry)# sensor-group 6`<br>`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]**<br>例：<br>`switch(conf-tm-sensor)# path interface query-condition counters=[detailed],oper-speed=[1G,40G]`<br>`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。<br>• 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。<br>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。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 5 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイスイベントが発生したときに送信するかを決定します。 |
| ステップ 9 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# データまたはイベントをポーリングするための環境パスの構成

環境パス ラベルは、ファン、温度、電源、ストレージ、スーパーバイザ、ラインカードなどのシャーシ情報をモニタリングします。テレメトリ データを定期的にポーリングするか、イベントが発生したときにデータを取得するように環境パスを構成できます。詳細については、45 ページの「データのポーリングまたはイベントの受信」を参照してください。

定期的なポーリングまたはイベントに基づいてシステム リソース情報を返すようにリソース パスを設定できます。このパスはフィルタリングをサポートしていません。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **path environment**
5. **destination-group** *grp_id*

6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(config-telemetry)# sensor-group 6`<br>`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 environment**<br>例：<br>`switch(conf-tm-sensor)# path environment`<br>`switch(conf-tm-sensor)#` | 複数の個々の環境オブジェクトのテレメトリ データをラベルに送信できるようにする環境パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。<br>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*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例： | 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 |
|---|---|---|
| | `switch(conf-tm-dest)# subscription 33`<br>`switch(conf-tm-sub)#` | |
| ステップ 8 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサーグループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、環境イベントが発生したときに送信するかを決定します。 |
| ステップ 9 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 電力使用量トラッキング機能の有効化

NX-OS リリース 10.4.1(F) 以降、Cisco Nexus 9336C-FX2 および 9332D-GX2B スイッチでは、電力消費を追跡するために **power usage-history** コマンドがサポートされています。デフォルトでは、この機能は無効になっています。

Cisco NX-OS リリース 10.4(2)F 以降、Cisco Nexus 9000 シリーズ プラットフォーム スイッチでは、 **power usage-history** コマンドがサポートされています。

この機能を有効にするには、次の手順を実行します。

## 手順の概要

1. **configure terminal**
2. **[no] power usage-history**

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`switch(config)#` | 構成モードに入ります。 |
| ステップ 2 | **[no] power usage-history**<br>例：<br>`switch# power usage-history`<br>`switch(config)#` | 電力使用量トラッキング機能を有効にします。この機能を無効化するには、このコマンドの **no** 形式を使用します。 |

# 電力消費履歴の表示

## 電力使用量追跡の表示コマンド

電力使用量追跡機能を有効にするには、電力使用量トラッキング機能の有効化（576ページ）を参照してください。有効にした後、 **show environment power history** を使用してさまざまなターゲットの電力使用量の統計情報を表示します。

| コマンド | 表示内容 |
|---|---|
| **show environment power history { peak \| target 1min \| target 1hr \| target 14 days \| target 14days day _day_no }** | さまざまなターゲットの電力使用量情報。 |

## コマンドの例

**show environment power history peak** コマンドの例を次に示します。

```
switch# show environment power history peak
 Power                                    Output Power       Peak  Time           Input
 Power      Peak  Time
Supply    Model              Status         (peak)         (Output Power)          (peak)
        (Input Power)
-----------------------------------------------------------------------------------------------

1      N9K-PAC-3000W-B       Ok            334.30 W      06/07/2023 10:54:29     362.45
 W  06/07/2023 10:54:59
2      N9K-PAC-3000W-B       Ok            362.45 W      06/07/2023 10:53:29     425.80
 W  06/07/2023 10:51:44
switch#



Last 1min usage data would contain average usage in last 15secs, 30secs and 60 secs.

module-4#  show environment power history target 1min

Power                                    Output/Input             Output/Input
    Output/Input
Supply    Model              Status          (15 sec)                (30 sec)
      (60 sec)
-----------------------------------------------------------------------------------------------

1      N9K-PAC-3000W-B       Ok          330.78 W / 362.45  W     330.00 W / 362.00  W
   330.00 W / 362.00  W
2      N9K-PAC-3000W-B       Ok          358.94 W / 415.24  W     358.00 W / 415.00  W
   358.00 W / 415.00  W
switch#
```

**show environment power history target 1hr** コマンドの出力を次に示します。

```
switch# show environment power history target 1hr
   1 min avg data for 1 Hr for
 slot: 1 Product Name: N9K-PAC-3000W-B status: Ok
   Output Power      Input Power             Time
-------------------------------------------------------
    331.00 W          362.00 W           06/07/2023 11:34:44
    330.00 W          362.00 W           06/07/2023 11:33:44
    333.00 W          362.00 W           06/07/2023 11:32:44
```

```
    333.00 W          362.00 W           06/07/2023 11:31:44
    331.00 W          362.00 W           06/07/2023 11:30:44


 1 min avg data for 1 Hr for
 slot: 2 Product Name: N9K-PAC-3000W-B status: Ok
   Output Power      Input Power             Time
-------------------------------------------------------
    358.00 W          417.00 W           06/07/2023 11:34:44
    358.00 W          417.00 W           06/07/2023 11:33:44
    358.00 W          417.00 W           06/07/2023 11:32:44
    358.00 W          417.00 W           06/07/2023 11:31:44
    357.00 W          415.00 W           06/07/2023 11:30:44
```

**show environment power history target 24hr** コマンドの例を次に示します。

```
switch# show environment power history target 24hr
   1HR  avg data for 24 Hr for
 slot: 1 Product Name: N9K-PAC-3000W-B status: Ok
   Output Power      Input Power             Time
-------------------------------------------------------
    332.15 W          363.56 W           06/07/2023 12:50:44
    332.13 W          363.66 W           06/07/2023 11:50:44

 1HR  avg data for 24 Hr for
 slot: 2 Product Name: N9K-PAC-3000W-B status: Ok
   Output Power      Input Power             Time
-------------------------------------------------------
    358.23 W          416.68 W           06/07/2023 12:50:44
    358.35 W          417.05 W           06/07/2023 11:50:44
switch#
```

**show environment power history target 14days** コマンドの例を次に示します。

```
switch# show environment power history target 14days
   1 Day avg data  over a period of 14 days
 slot: 1 Product Name: N9K-PAC-3000W-B status: Ok
Day   Output Power        Input Power            Date
------------------------------------------------------------
  1      332.17 W          363.61 W             06/07/23

 1 Day avg data  over a period of 14 days
 slot: 2 Product Name: N9K-PAC-3000W-B status: Ok
Day   Output Power        Input Power            Date
------------------------------------------------------------
  1      358.23 W          416.81 W             06/07/23
switch#

This CLI displays the average usage throughout the day for each day in last 14days. For
 each PSU 14 days average usage is displayed. A detailed per hour usage for each day is
 displayed when day number is given. Output for that is given in next slide.
```

**show environment power history target 14days day 1** コマンドの例を次に示します。

```
switch# show environment power history target 14days day 1
 1 HR  avg data for 1 Day
 slot: 1 Product Name: N9K-PAC-3000W-B status: Ok
Day 1
   Output Power      Input Power             Time
-------------------------------------------------------
    332.23 W          363.61 W           06/07/2023 13:50:44
    332.15 W          363.56 W           06/07/2023 12:50:44
    332.13 W          363.66 W           06/07/2023 11:50:44
```

```
 1 HR  avg data for 1 Day
 slot: 2 Product Name: N9K-PAC-3000W-B status: Ok
Day 1
   Output Power      Input Power             Time
-------------------------------------------------------
    358.11 W          416.71 W           06/07/2023 13:50:44
    358.23 W          416.68 W           06/07/2023 12:50:44
    358.35 W          417.05 W           06/07/2023 11:50:44
switch#
switch#
```

# 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。45ページの「データのポーリングまたはイベントの受信」を参照してください。

このパスはフィルタリングをサポートしていません。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group***sgrp_id*
4. **path resources**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例： | テレメトリ データのセンサー グループを作成します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `switch(config-telemetry)# sensor-group 6`<br>`switch(conf-tm-sensor)#` | |
| ステップ 4 | **path resources**<br>例：<br>`switch(conf-tm-sensor)# path resources`<br>`switch(conf-tm-sensor)#` | 複数の個々のシステム リソースのテレメトリ データをラベルに送信できるようにするリソース パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。<br>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。 |
| ステップ 5 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、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*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 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。「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 （586ページ）」を参照してください。

このパスはフィルタリングをサポートしていません。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **vxlan environment**
5. **destination-group** *grp_id*
6. **ip address** *ip_addr* **port** *port*
7. **subscription** *sub_id*
8. **snsr-group** *sgrp_id* **sample-interval** *interval*
9. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(config-telemetry)# sensor-group 6`<br>`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 | **vxlan environment**<br>例：<br>`switch(conf-tm-sensor)# vxlan environment`<br>`switch(conf-tm-sensor)#` | 複数の個々の VXLAN オブジェクトのテレメトリ データをラベルに送信できるようにする vxlan パス ラベルを構成します。ラベルは、複数のデータ入力を 1 つの出力に統合します。サンプル間隔に応じて、VXLAN データはポーリング間隔に基づいてストリーミングされるか、イベントが発生したときに送信されます。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 5 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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*<br>例：<br>`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004`<br>`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*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、VXLAN イベントが発生したときに送信するかを決定します。 |
| ステップ 9 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# パス ラベル 構成 を確認

いつでも、パス ラベルが構成されていることを確認し、実行中のテレメトリ構成を表示してその値を確認できます。

## 手順の概要

1. **show running-config-telemetry**

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **show running-config-telemetry**<br>例：<br>`switch(conf-tm-sensor)# show running-config telemetry`<br><br>`!Command: show running-config telemetry`<br>`!Running configuration last done at: Mon Jun 10 08:10:17 2019`<br>`!Time: Mon Jun 10 08:10:17 2019`<br><br>`version 9.3(1) Bios:version`<br>`feature telemetry`<br><br>`telemetry`<br>`  destination-profile`<br>`  use-nodeid tester`<br>`  sensor-group 4`<br>`  path interface query-condition and(counters=[detailed],oper-speed=[1G,10G])`<br>`  sensor-group 6`<br>`  path interface query-condition oper-speed=[1G,40G]`<br>`  subscription 6`<br>`  snsr-grp 6 sample-interval 6000`<br>`nxosv2(conf-tm-sensor)#` | 現在のテレメトリ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 で動作し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<br>パス ラベルのすべてのテレメト<br>定期的なポーリングまたはイベ<br>報告するかどうかが示されます<br><br>インターフェイス パス ラベルに<br>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 |

### 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として働きます。

次に、**show telemetry usability environment** コマンドの例を示します。

```
switch# show telemetry usability environment
1) label_name : environment
path_name : sys/ch query_type : poll query_condition :
rsp-subtree=full&query-target=subtree&target-subtree-class=eqptPsuSlot,eqptFtSlot,eqptSupCSlot,eqptPsu,eqptFt,eqptSensor,eqptLCSlot
2) label_name : environment
path_name : sys/ch query_type : event query_condition :
[illegible]
 switch#
```

次に、**show telemetry usability interface** コマンドの出力を示します。

```
switch# show telemetry usability interface
1) label_name : interface
path_name : sys/intf query_type : poll query_condition :
query-target=children&query-target-filter=eq(l1PhysIf.adminSt,"up")&rsp-subtree=children&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut
2) label_name : interface
path_name : sys/mgmt-[mgmt0] query_type : poll query_condition :
query-target=subtree&query-target-filter=eq(mgmtMgmtIf.adminSt,"up")&rsp-subtree=full&rsp-subtree-class=rmonEtherStats,rmonIfIn,rmonIfOut,rmonIfHCIn,rmonIfHCOut
3) label_name : interface
path_name : sys/intf query_type : event query_condition :
[illegible]
ethpmEncRtdIf.operSt,"down")),and(updated(ethpmEncRtdIf.operSt),eq(ethpmEncRtdIf.operSt,"up"))))
4) label_name : interface
path_name : sys/mgmt-[mgmt0] query_type : event query_condition :
query-target=subtree&query-target-filter=or(deleted(),created()),or(and(updated(mgmtIf.operSt),eq(mgmtIf.operSt,"down")),and(updated(mgmtIf.operSt),eq(mgmtIf.operSt,"up"))))
 switch#
```

次に、**show telemetry usability resources** コマンドの例を示します。

```
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     :
[illegible]


  3) label_name          : resources

     path_name           : sys/procsys/sysmem
     query_type          : event
     query_condition     :
query-target-filter=and(updated(procSysMem.memstatus),ne(procSysMem.memstatus,"OK"))

switch#
```

次に、**show telemetry usability vxlan** コマンドの例を示します。

```
switch# show telemetry usability vxlan
  1) label_name          : vxlan

     path_name           : sys/bd
     query_type          : poll
     query_condition     : query-target=subtree&target-subtree-class=l2VlanStats

  2) label_name          : vxlan

     path_name           : sys/eps
     query_type          : poll
     query_condition     : rsp-subtree=full&rsp-foreign-subtree=ephemeral

  3) label_name          : vxlan

     path_name           : sys/eps
     query_type          : event
     query_condition     : query-target=subtree&target-subtree-class=nvoDyPeer

  4) label_name          : vxlan

     path_name           : sys/bgp
     query_type          : event
    query_condition    : query-target=subtree&query-target-filter=or(deleted(),created())


  5) label_name          : vxlan

     path_name           : sys/bgp
     query_type          : event
     query_condition     :
query-target=subtree&target-subtree-class=bgpDom,bgpPeer,bgpPeerAf,bgpDomAf,bgpPeerAfEntry,bgpOperRtctrlL3,bgpOperRttP,bgpOperRttEntry,bgpOperAfCtrl

switch#
```

# ネイティブ データ送信元パス

## ネイティブ データ送信元パスについて

NX-OSテレメトリは、特定のインフラストラクチャまたはデータベースに限定されないニュートラル データ 送信元であるネイティブ データ ソースをサポートします。代わりに、ネイティブ データ 送信元を使用すると、コンポーネントまたはアプリケーションをフックして、関連情報を発信テレメトリ ストリームに挿入できます。ネイティブ データ 送信元のパスはインフラストラクチャに属さないため、この機能は柔軟性を提供し、ネイティブアプリケーションはNX-OS テレメトリと対話できます。

ネイティブ データ 送信元 パスを使用すると、特定のセンサー パスに登録して、セレクトしたテレメトリ データを受信できます。この機能は NX-SDK と連携して、次のパスからのテレメトリ データのストリーミングをサポートします。

- IP ルートのテレメトリ データを送信する RIB パス。
- 静的および動的 MAC エントリのテレメトリ データを送信する MAC パス。
- IPv4 と IPv6 隣接のテレメトリ データを送信する隣接関係パス。

サブスクリプションを作成すると、選択したパスのすべてのテレメトリデータが基準値として受信者にストリーミングされます。基準値の後、イベント通知のみが受信者にストリーミングされます。

ネイティブ データ 送信元 パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf（GPB）
- JavaScript Object Notation（JSON）
- コンパクト Google Protobuf (コンパクト GPB)

# ネイティブ データ送信元パス用にストリーミングされるテレメトリ データ

次の表は、各ソースパスについて、サブスクリプションが最初に作成されたとき（ベースライン）とイベント通知が発生したときにストリーミングされる情報を示しています。

| Path Type | サブスクリプション ベースライン | イベント通 |
|---|---|---|

| | | |
|---|---|---|
| RIB | 全てのルートの送信 | イベント<br>イベント<br>パスのラ<br>ます：<br>• ネク<br>•<br>•<br>•<br>•<br>•<br>•<br>•<br>•<br>•<br>•<br>• レイ<br>•<br>•<br>•<br>•<br>•<br>ネクスト |

| | | |
|---|---|---|
| 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<br>ベント通知<br>スのテレメ<br>す：<br>• MAC フ<br>• MAC フ<br>• VLAN<br>• インタ<br>• イベン<br>イベント通<br>ク エントリ |
| 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<br>ベント通知<br>係（アジャ<br>じてエクス<br>• IP アド<br>• MAC フ<br>• インタ<br>• 物理イ<br>• VRF 名<br>• プリフ<br>• 隣接の<br>• 隣接関<br>ファミ<br>隣接関係（<br>プ |

詳細については、Github の 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**、または **filter-condition** などのカスタム基準をサポートしません。
- Cisco NX-OSリリース 10.4(3)F 以降では、RIB ネイティブ パスのサンプル ベースのサブスクリプションまたは更新のみをサポートする、新しいクエリ条件が導入されています。

# ルーティング情報のネイティブ データ送信元パスの構成

URIB に含まれるすべてのルートに関する情報を送信するルーティング情報のネイティブ データ送信元パスを構成できます。登録すると、基準値はすべてのルート情報を送信します。ベースラインの後、スイッチがサポートするルーティングプロトコルのルート更新と削除操作について通知が送信されます。RIB 通知で送信されるデータについては、61 ページの「ネイティブデータソースパス用にストリーミングされるテレメトリデータ」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（**機能テレメトリ**）。

### 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path rib query-condition** [**data=ephemeral** | **updates_only**]
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol { HTTP | gRPC } encoding { JSON | GPB | GPB-compact }**
8. subscription sub_id
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例： | 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 |
|---|---|---|
| | `switch(config)# telemetry`<br>`switch(config-telemetry)#` | |
| ステップ 3 | **sensor-group** *sgrp_id*<br>例：<br>`switch(conf-tm-sub)# sensor-grp 6`<br>`switch(conf-tm-sub)#` | センサー グループを作成します。 |
| ステップ 4 | **data-source native**<br>例：<br>`switch(conf-tm-sensor)# data-source native`<br>`switch(conf-tm-sensor)#` | 特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。 |
| ステップ 5 | **path rib query-condition** [**data=ephemeral** \| **updates_only**]<br>例：<br>`nxosv2(conf-tm-sensor)# path rib`<br>`nxosv2(conf-tm-sensor)#`<br>例：<br>`nxosv2(conf-tm-sensor)# path rib query condition data=ephemeral`<br>`nxosv2(conf-tm-sensor)#`<br>例：<br>`nxosv2(conf-tm-sensor)# path rib query condition updates_only`<br>`nxosv2(conf-tm-sensor)#` | ルートとルート アップデート情報をストリーミングする RIB パスを構成します。<br>**query condition data=ephemeral**（オプション）：サンプル間隔 0 または 0 以外を設定できます。このサンプル間隔によって、接続先にルート情報が定期的に送信される頻度が決まります（構成されたサンプル間隔で）。<br>**query condition updates-only**（オプション）：サンプル間隔 0 でのみサポートされます。このクエリ条件では、最初のスナップショットデータは送信されず、ルート情報の更新のみが接続先に送信されます。 |
| ステップ 6 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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 7 | **ip address** *ip_addr* **port** *port* **protocol { HTTP \| gRPC } encoding { JSON \| GPB \| GPB-compact }**<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`<br>`switch(conf-tm-dest)#`<br>`Example:`<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`<br>`switch(conf-tm-dest)#`<br>`Example:`<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`<br>`switch(conf-tm-dest)#` | サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。 |
| ステップ 8 | subscription sub_id<br>例： | 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 |
|---|---|---|
| | `switch(conf-tm-dest)# subscription 33`<br>`switch(conf-tm-sub)#` | |
| ステップ 9 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリデータを定期的に送信するか、rib イベントが発生したときに送信するかを決定します。<br>（注）<br>サンプリング間隔に応じて、rib センサーパスはポーリング間隔に基づいてストリーミングします。 |
| ステップ 10 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 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 （586 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（**機能テレメトリ**）。

### 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source native**
5. **path mac**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol { HTTP | gRPC } encoding { JSON | GPB | GPB-compact }**

8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group***dgrp_id*

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(conf-tm-sub)# sensor-grp 6`<br>`switch(conf-tm-sub)#` | センサー グループを作成します。 |
| ステップ 4 | **data-source native**<br>例：<br>`switch(conf-tm-sensor)# data-source native`<br>`switch(conf-tm-sensor)#` | 特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。 |
| ステップ 5 | **path mac**<br>例：<br>`nxosv2(conf-tm-sensor)# path mac`<br>`nxosv2(conf-tm-sensor)#` | MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。 |
| ステップ 6 | **destination-group** *grp_id*<br>例：<br>`switch(conf-tm-sensor)# destination-group 33`<br>`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 7 | **ip address** *ip_addr* **port** *port* **protocol { HTTP \| gRPC } encoding { JSON \| GPB \| GPB-compact }**<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`<br>`switch(conf-tm-dest)#`<br>例： | サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| | `switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`<br>`switch(conf-tm-dest)#`<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`<br>`switch(conf-tm-dest)#` | |
| ステップ 8 | **subscription** *sub_id*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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 9 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。 |
| ステップ 10 | **dst-group***dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 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 （586 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（**機能テレメトリ**）。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source** *native*
5. **path mac-all**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol { HTTP | gRPC } encoding { JSON | GPB | GPB-compact }**
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(conf-tm-sub)# sensor-grp 6`<br>`switch(conf-tm-sub)#` | センサー グループを作成します。 |
| ステップ 4 | **data-source** *native*<br>例：<br>`switch(conf-tm-sensor)# data-source native`<br>`switch(conf-tm-sensor)#` | 特定のモデルやデータベースを必要とせずに、ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。 |
| ステップ 5 | **path mac-all**<br>例：<br>`nxosv2(conf-tm-sensor)# path mac-all`<br>`nxosv2(conf-tm-sensor)#` | すべての MAC エントリおよび MAC 通知に関する情報をストリームする MAC パスを構成します。 |
| ステップ 6 | **destination-group** *grp_id*<br>例： | 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 |
|---|---|---|
| | `switch(conf-tm-sensor)# destination-group 33`<br>`switch(conf-tm-dest)#` | |
| ステップ 7 | **ip address** *ip_addr* **port** *port* **protocol { HTTP \| gRPC } encoding { JSON \| GPB \| GPB-compact }**<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`<br>`switch(conf-tm-dest)#`<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`<br>`switch(conf-tm-dest)#`<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`<br>`switch(conf-tm-dest)#` | サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。 |
| ステップ 8 | **subscription** *sub_id*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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 9 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。 |
| ステップ 10 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 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、61 ページの「ネイティブデータソースパス用にストリーミングされるテレメトリデータ」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（**機能テレメトリ**）。

## 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data-source** *native*
5. **path adjacency**
6. **destination-group** *grp_id*
7. **ip address ip_addr port port protocol { HTTP | gRPC } encoding { JSON | GPB | GPB-compact }**
8. **subscription** *sub_id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

## 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(conf-tm-sub)# sensor-grp 6`<br>`switch(conf-tm-sub)#` | センサー グループを作成します。 |
| ステップ 4 | **data-source** *native*<br>例：<br>`switch(conf-tm-sensor)# data-source native`<br>`switch(conf-tm-sensor)#` | ネイティブ アプリケーションがストリーム データを使用できるように、データ送信元をネイティブに設定します。 |
| ステップ 5 | **path adjacency**<br>例：<br>`nxosv2(conf-tm-sensor)# path adjacency`<br>`nxosv2(conf-tm-sensor)#` | IPv4 と IPv6 隣接に関する情報をストリームする隣接パスを構成します。 |
| ステップ 6 | **destination-group** *grp_id*<br>例： | 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 |
|---|---|---|
| | `switch(conf-tm-sensor)# destination-group 33`<br>`switch(conf-tm-dest)#` | |
| ステップ 7 | **ip address ip_addr port port protocol { HTTP \| gRPC } encoding { JSON \| GPB \| GPB-compact }**<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`<br>`switch(conf-tm-dest)#`<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb`<br>`switch(conf-tm-dest)#`<br>例：<br>`switch(conf-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb-compact`<br>`switch(conf-tm-dest)#` | サブスクリプションのテレメトリ データを、指定された IP アドレスとポートにストリーミングするように構成し、データ ストリームのプロトコルとエンコードを設定します。 |
| ステップ 8 | **subscription** *sub_id*<br>例：<br>`switch(conf-tm-dest)# subscription 33`<br>`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 9 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(conf-tm-sub)# snsr-grp 6 sample-interval 5000`<br>`switch(conf-tm-sub)#` | センサー グループを現在のサブスクリプションにリンクして、データのサンプリング間隔（ミリ秒単位）を設定します。サンプリング間隔は、スイッチがテレメトリ データを定期的に送信するか、インターフェイス イベントが発生したときに送信するかを決定します。 |
| ステップ 10 | **dst-group** *dgrp_id*<br>例：<br>`switch(conf-tm-sub)# dst-grp 33`<br>`switch(conf-tm-sub)#` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 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#
An example of the statistics for the MAC path:
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#
An example of the statistics for the Adjacency path:
 switch# show telemetry event collector stats
-------------------------------------------------------------------------------
 Row ID Collection Count Latest Collection Time Sensor Path(GroupId)
-------------------------------------------------------------------------------
 1 7 Mon Jul 01 14:47:32.260 PST adjacency(1)
switch#
```

### エラー カウンタの表示

**show telemetry event collector stats** コマンドを使用して、すべてのネイティブデータソースパスのエラーの合計を表示できます。

```
switch# show telemetry event collector errors
-------------------------------------------------------------------------------
 Error Description Error Count
-------------------------------------------------------------------------------
 Dme Event Subscription Init Failures - 0
 Event Data Enqueue Failures - 0
 Event Subscription Failures - 0
 Pending Subscription List Create Failures - 0
 Subscription Hash Table Create Failures - 0
 Subscription Hash Table Destroy Failures - 0
 Subscription Hash Table Insert Failures - 0
 Subscription Hash Table Remove Failures - 0
switch#
```

# ストリーミング Syslog

## テレメトリ用のストリーミング Syslog について

Cisco NX-OS リリース 9.3(3) 以降、モデル駆動型テレメトリは、YANG をデータソースとして使用する syslog のストリーミングをサポートします。サブスクリプションを作成すると、すべての syslog が基準値として受信者にストリーミングされます。この機能は NX-SDK と連携して、次の syslog パスからのストリーミング syslog データをサポートします。

- Cisco-NX-OS-Syslog-oper:syslog
- Cisco-NX-OS-Syslog-oper:syslog/messages

基準値の後は、syslog イベント通知のみが受信者にストリーミングされます。syslog パスのストリーミングは、次のエンコーディング タイプをサポートします：

- Google Protobuf（GPB）

• JavaScript Object Notation（JSON）

# syslog 情報のための YANG データ ソース パスの構成

スイッチで生成されたすべての syslog に関する情報を送信する syslog の syslog パスを構成できます。サブスクライブすると、ベースラインはすべての既存の syslog 情報を送信します。ベースラインの後、通知は、スイッチで生成された新しい syslog に対してのみ送信されます。

### 始める前に

テレメトリ機能を有効にしていない場合は、**feature telemetry** コマンドで有効にします。

### 手順の概要

1. **configure terminal**
2. **telemetry**
3. **sensor-group** *sgrp_id*
4. **data source** *data-source-type*
5. **path Cisco-NX-OS-Syslog-oper:syslog/messages**
6. **destination-group** *grp_id*
7. **ip address** *ip_addr* **port** *port* **protocol { HTTP | gRPC } encoding { JSON | GPB | GPB-compact }**
8. **subscription** *sub-id*
9. **snsr-group** *sgrp_id* **sample-interval** *interval*
10. **dst-group** *dgrp_id*

### 手順の詳細

手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **configure terminal**<br>例：<br>`switch# configure terminal`<br>`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**<br>例：<br>`switch(config)# telemetry`<br>`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*<br>例：<br>`switch(conf-tm-sub)# sensor-grp 6`<br>`switch(conf-tm-sub)#` | センサー グループを作成します。 |

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 4 | **data source** *data-source-type*<br>例：<br>`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**<br>例：<br>`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。 |
| ステップ 6 | **destination-group** *grp_id*<br>例：<br>`switch(config-tm-sensor)# destination-group 33` | テレメトリ接続先グループ サブモードに入り、接続先グループを構成します。 |
| ステップ 7 | **ip address** *ip_addr* **port** *port* **protocol { HTTP \| gRPC } encoding { JSON \| GPB \| GPB-compact }**<br>例：<br>`switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol http encoding json`<br>例：<br>`switch(config-tm-dest)# ip address 192.0.2.11 port 50001 protocol grpc encoding gpb` | 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*<br>例：<br>`switch(config-tm-dest)# subscription 33` | テレメトリ サブスクリプションサブモードに入り、テレメトリ サブスクリプションを構成します。 |
| ステップ 9 | **snsr-group** *sgrp_id* **sample-interval** *interval*<br>例：<br>`switch(config-tm-sub)# snsr-group 6 sample-interval 0` | センサーグループを現在のサブスクリプションにリンクし、データ サンプリングを 0 に設定して、syslog イベントが発生したときにスイッチがテレメトリ データを送信するようにします。interval については、0 のみが受け入れ可能な値です。 |
| ステップ 10 | **dst-group** *dgrp_id*<br>例：<br>`switch(config-tm-sub)# dst-grp 33` | 接続先グループをこのサブスクリプションにリンクします。指定する接続先グループは、destination-group コマンドで構成した接続先グループと一致している必要があります。 |

# 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 |
|---|---|---|

| | | |
|---|---|---|
| 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<br>送信し<br>• mes<br>• nod<br>• time<br>• time<br>• time<br>• cate<br>• mes<br>• seve<br>• text |

### syslog パス情報の表示

Cisco NX-OS の **show telemetry event collector** コマンドを使用して、syslog 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

### 統計の表示

**show telemetry event collector stats** コマンドを入力して、各 syslog パス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。

```
--------------------------------------------------------------------------------
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** コマンドを使用して、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。

```
---Telemetry msg received @ 18:22:04 UTC


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"
      }
    }
  }
}
```


# テレメトリのトラブルシューティング

- 

## 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

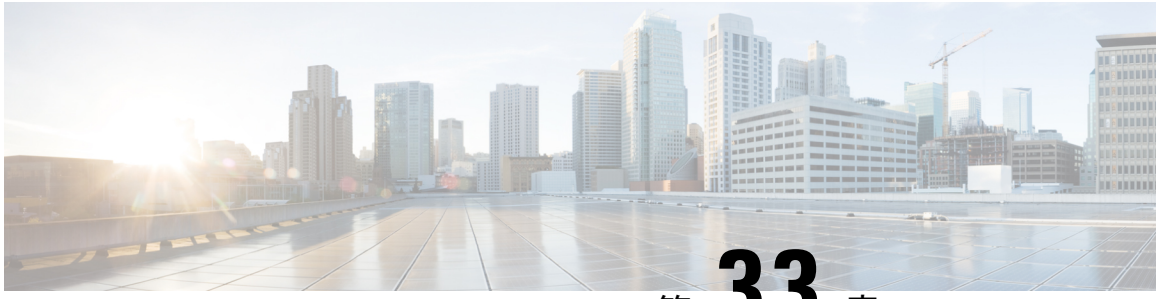
このNX-OS CLIコマンドは、テクニカル サポート ログからテレメトリ ログの内容を収集します。この例では、コマンド出力がブートフラッシュのファイルにリダイレクトされます。

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

# 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* |



第 33 章

# 診断と有用性

この章は次のトピックで構成されています。


- 診断と有用性について （605 ページ）
- コマンドの表示 （605 ページ）
- デバッグ ログ （607 ページ）
- 診断の提案 （610 ページ）
- マニュアルの変更履歴 （612 ページ）


## 診断と有用性について

Cisco NX-OSは、Netconf、Restconf、gNMI/gNOI、テレメトリなどの幅広いプロトコルインターフェイスを介して、モデル駆動型プログラマビリティ（MDP）をサポートします。実際、これらのインターフェイスは、共通の基盤となる YANG および DME/ CLIインフラストラクチャを中心に動作します。ユーザーは、一般的に使用される一群のユーティリティにより、動作を診断できます。

## コマンドの表示

このセクションでは、一般的に使用される show コマンドについて説明します。スイッチの実行状態を確認するために使用できます。

表 *44 :* 表示コマンド：診断と有用性

| 項目 | コマンド | 使用法 |
|---|---|---|
| netconf | **show netconf nxsdk event-history { msgs \| cli \| internal libnxsdk \| all }** | 詳細イベント履歴の表示 |
| | **show netconf nxsdk event-history component { agent \| mtx }** | イベント履歴の表示 |
| | **show netconf internal mtx** | MTXインフラ情報を表示します。 |
| | **show running-config netconf** | netconf 構成の表示 |
| | **show netconf nxsdk event-history** {**events** \| **errors**} | イベント履歴の表示 |
| | **show tech-support netconf** | netconf テクニカルサポートの収集 |
| restconf | **show running-config** \| **grep restconf** | restconf 構成の表示 |
| | **show retconf nxsdk event-history** {**events** \| **errors**} | イベント履歴の表示 |
| | **show restconf nxsdk event-history { msgs \| cli \| internal libnxsdk \| all }** | 詳細イベント履歴の表示 |
| | **show restconf nxsdk event-history component { agent \| mtx }** | イベント履歴の表示 |
| | **show restconf internal mtx** | MTXインフラ情報を表示します。 |
| | **show tech-support restconf** | restconfテクニカル サポートの収集 |

| 項目 | コマンド | 使用法 |
|---|---|---|
| grpc | **show running-config grpc** | grpc 構成の表示 |
| | **show grpc nxsdk event-history** {**events** \| **errors**} | イベント履歴の表示 |
| | **show tech-support grpc** | grpc テクニカルサポートの収集 |
| | **show grpc nxsdk event-history { msgs \| cli \| internal libnxsdk \| all }** | 詳細イベント履歴の表示 |
| | **show grpc nxsdk event-history component { agent \| mtx \| telemetry }** | イベント履歴の表示 |
| | **show grpc internal mtx** | MTXインフラ情報を表示します。 |
| gnmi | **show grpc gnmi service statistics** | grpc サーバの状態の確認 |
| | **show grpc gnmi rpc** [**all**] { **summary** \| **detail** } | 現在/過去の gNMIサブスクリプションのリスト表示 |
| | **show grpc gnmi transactions** | List gNMI Get/Set |
| gnoi | **show grpc gnoi service statistics** | grpc サーバの状態の確認 |
| openconfig | **show running-config openconfig** | openconfig 構成の表示 |
| | **show openconfig nxsdk event-history** {**events** \| **errors**} | イベント履歴の表示 |
| dme | **show system internal dme transaction history** | DME トランザクションの確認 |
| | **show tech-support dme** | DMEテクニカルサポートの収集 |

# デバッグ ログ

このセクションでは、デバッグログを有効にして収集する方法について説明します。

## プログラマビリティ エージェント ログ

Netconf、Restconf、および gRPC エージェントの場合、次の方法でログを収集できます。

- **show コマンド**

これは、エージェント イベントを表示/確認する簡単な方法です。これらのコマンドは、エージェントがクライアント接続とどのように対話するかを確認するのに役立ちます。

```
show netconf nxsdk event-history { events | errors }
show netconf nxsdk event-history component { agent events | mtx events }
show restconf nxsdk event-history { events | errors }
show restconf nxsdk event-history component { agent events | mtx events }
show grpc nxsdk event-history { events | errors }
show grpc nxsdk event-history component { agent events | mtx events | telemetry
events }
```

- **ログファイル**

  より長い履歴、またはエージェントを無効にした後のログを確認する場合は、 **/volatile** ディレクトリに保存されているログファイルを確認します。ユーザーには、スイッチの bash シェルにアクセスする権限が必要です。

```
/volatile/netconf-internal-log
              grpc-internal-log
              restconf-internal-log
```

# YANG インフラログ

YANG インフラのデバッグ/トレース レベルのログとファイルのロギングは、デフォルトで無効になっています。ユーザーはファイル ロギングを有効にできます。ファイル ロギングが有効になった後、YANG インフラ ログは、 /volatile ディレクトリに保存されます。

ユーザーには、スイッチの bash シェルにアクセスする権限が必要です。

```
debug netconf mtx file
debug restconf mtx file
debug grpc mtx file

/volatile/mtx-internal.netconf.log
              mtx-internal.grpc.log
              mtx-internal.restconf.log
```

# DME ログ

DME インフラログは、**/nxos/dme_logs** ディレクトリに保存されます。ユーザーには、スイッチの bash シェルにアクセスする権限が必要です。「https://developer.cisco.com/site/cisco-nexus-nx-api-references/」を参照してください。

```
/nxos/dme_logs/svc_ifc_policyelem.<pid>.log
```

### XOS ログ

tmtrace.bin コマンドによって公開される XOS ログ。

ユーザーには、スイッチの bash シェルにアクセスする権限が必要です。

```
tmtrace.bin -d cmi-errors
tmtrace.bin -d cmi-events
tmtrace.bin -d cmi-debug
tmtrace.bin -d cmi-msg
```

# CLI を使用したログ構成の・更

CLI を使用して、プロセスを再起動せずに上記のロギング構成をダイナミックに変更することができます。これらのコマンドは、エージェント EXEC コマンドごとであり、構成ファイルへの変更ではないため、現在の操作に影響を与えることなく変更できます。

## 手順の概要

1. **[no] debug grpc mtx enable-all**
2. **[no] debug grpc mtx level <level>**
3. **[no] debug grpc mtx item <item>**

## 手順の詳細

### 手順

| | コマンドまたはアクション | 目的 |
|---|---|---|
| ステップ 1 | **[no] debug grpc mtx enable-all** | これは、すべてのログを有効にする便利な cli です。 |
| ステップ 2 | **[no] debug grpc mtx level <level>**<br>例：<br>`switch# debug grpc mtx level info` | ロギングレベルを切り替えます：エラー（error）、警告（warning）、情報（info）、デバッグ（debug）から選択できます。<br>デフォルトのレベルは **info** です。 |
| ステップ 3 | **[no] debug grpc mtx item <item>**<br>例：<br>`switch# debug grpc mtx item MTX-EvtMgr` | 特定の項目のロギングを切り替えます。これは自由形式の文字列で、 **show grpc internal mtx debug** を使用して使用可能な項目を表示します。 |

### 例

次の show cli は、現在のロギング構成を表示します。

```
show grpc internal mtx debug

Example:
  switch# show grpc internal mtx debug
  Log enabled  : 1
  File enabled : 0
  All active   : 0
  Log Level    : Info
  Log items    :
    *                              : 0
    DB                             : 0
    DtxUserFunc                    : 0
    INST-ADAPTER                   : 0
    INST-ADAPTER-GNMI              : 0
    INST-ADAPTER-GNOI              : 0
    INST-ADAPTER-GRPC              : 1
    INST-ADAPTER-NC                : 1
```

```
INST-ADAPTER-RC                : 1
INST-ADAPTER-TM                : 0
INST-MTX-API                   : 1
LIBUTILS                       : 1
MTX-API                        : 1
MTX-ActionMgr                  : 0
MTX-Coder                      : 0
MTX-Dy-EvtMgr                  : 0
MTX-EvtMgr                     : 0
MTX-RbacMgr                    : 0
MTXEXPR                        : 0
MTXItem                        : 0
MTXNetConfMessage              : 0
MTXOperation                   : 0
MTXRestConfMessage             : 0
MTXgNMIMessage                 : 0
Model-*                        : 1
Model-Cisco-NX-OS-device       : 1
Model-openconfig-bgp           : 0
PVSH                           : 0
RPC                            : 0
SYSTEM                         : 1
TM-ADPT                        : 0
TM-ADPT-JSON                   : 0
```

# 診断の提案

このセクションでは、発生しやすい問題をトリアージするためのいくつかの手順を示します。

### 接続の問題

ユーザーのプログラミングクライアントがスイッチに接続できない場合は、次の点を確認します。

- 実行構成をチェックして、機能が有効になっているかどうかを確認します。
- 個々のエージェントの show コマンドをチェックして、サーバが実行されていることを確認します。
- IP/ポートをチェックして、接続がファイアウォールなどによって制限されていないことを確認します。
- クライアントが正しいユーザー名/パスワードを送信することを確認します。
- 証明書ベースの認証を使用する場合は、トラストポイントがスイッチに正しく構成されていること、およびクライアント証明書が一致し、期限切れになっていないことを確認します。

### ネイティブデバイス YANG

読み取り/書き込み操作に関連するネイティブ openconfig YANG に問題がある場合は、次の点を確認してください。

- 「書き込み」操作の場合は、DME トランザクションを確認して障害の詳細を確認します。
- 同等の DME REST 要求を送信し、同じ問題があるかどうかを確認します。

### OpenConfig Yang

ネイティブ openconfig YANG の読み取り/書き込みに問題がある場合は、次のことを確認します。

- **feature openconfig** が有効になっているかどうかを確認します。
- 公開されている YANG との相違点をチェックして、サポートステータスを確認します。
- **write** 操作の場合は、DME トランザクションを確認して障害の詳細を確認します。

### テレメトリ

テレメトリは、「feature telemetry」構成を介して YANG およびその他のデータソースを収集するために使用されます。テレメトリは、「feature grpc」を介した gNMI 登録にも使用されます。トラブルシューティングの手順は、使用シナリオによって異なります。

#### デバッグ ログ

デバッグログは、次の方法で表示できます。

- **show telemetry internal event history { errors | events }**
- **show grpc nxsdk event-history { events | errors }**

#### データ/イベント収集の問題：

show コマンドで、失敗した、またはスキップされた収集を確認します。

- **show telemetry data collector detail**
- **show telemetry event collector {errors | stats}**
- **show grpc internal gnmi subscription statistics**

#### 収集時間またはサイズの問題：

次の show コマンドを使用して、収集のサイズと時間を確認します。

- **show telemetry control database**
- **show grpc internal gnmi rpc subscription-data**

#### トランスポートの問題：

次の show コマンドを使用して、トランスポートの問題を確認します。トランスポートの問題は、 **feature telemetry** シナリオにのみ影響することに注意してください。

- **show telemetry transport <num> stats | errros**

# マニュアルの変更履歴

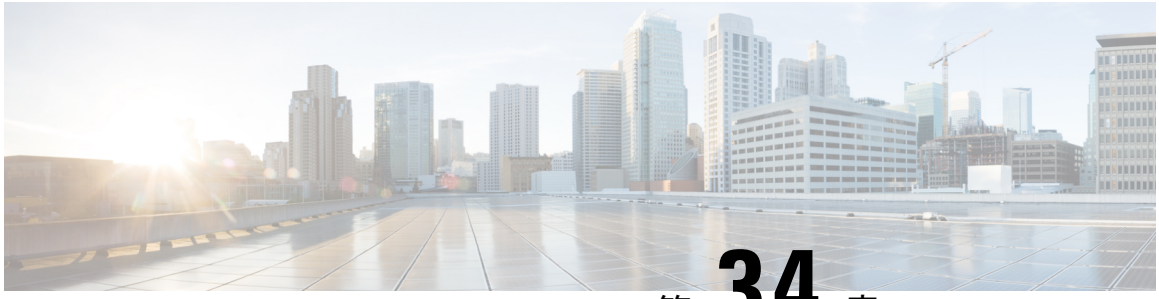
表 *45 :* 改訂履歴：診断と有用性

| リリース | 説明 |
|---|---|
| 10.4 (2) | ロギング パラメータを変更するための新しい CLI コマンド |
| 10.5（3） | Logging 2.0 のサポート。新しいコンポーネント event-history コマンドを追加します。feature dynamic event-history コマンドのサポート |

# 第 VI 部

# XML 管理インターフェイス

• XML 管理インターフェイス （615 ページ）



第 34 章

# XML 管理インターフェイス


- XML 管理インターフェイスについて （615 ページ）
- XML 管理インターフェイスのライセンス要件, on page 617
- 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 617
- XML 管理インターフェイスを使用, on page 617
- サンプル XML インスタンスに関する情報 （632 ページ）
- その他の参考資料, on page 639


# 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 621 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 46: NETCONF レイヤ*

| レイヤ | 例 |
|---|---|
| トランスポート プロトコル | SSHv2 |
| RPC | RPC、RPC-reply |
| 操作 | get-config、edit-config |
| コンテンツ | **show** または **configuration** コマンド |

以下は、4 つの NETCONF レイヤの説明です。

- SSH トランスポートプロトコル：クライアントとサーバー間の暗号化接続を提供します。
- RPC タグ：リクエスタからの構成コマンドと、それに対応する XML サーバからの応答を導入します。
- NETCONF 操作タグ：構成コマンドのタイプを示します。
- 格納ファイル — 構成する機能の XML 表現を示します。

## SSH xmlagent

デバイス ソフトウェアは、SSH バージョン 2 を介した NETCONF をサポートする xmlagent と呼ばれる SSH サービスを提供します。

**Note** xmlagent サービスは、Cisco NX-OS ソフトウェアでは XML サーバと呼ばれます。

NETCONF over SSH は、クライアントと XML サーバ間の hello メッセージの交換から始まります。最初の交換の後、クライアントはXML要求を送信し、サーバはXML応答で応答します。クライアントとサーバは、文字シーケンス>で要求と応答を終了します。この文字シーケンスは XML では有効ではないため、クライアントとサーバはメッセージがいつ終了するかを解釈でき、通信の同期が維持されます。

使用できる XML 構成インスタンスを定義する XML スキーマについては、NETCONF XML インスタンスの作成, on page 621 で説明されています。

# XML 管理インターフェイスのライセンス要件

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

# XML 管理インターフェイスを使用するための前提条件

XML 管理インターフェイスを使用するには、次の前提条件があります。

- クライアント PC に SSHv2 をインストールする必要があります。
- クライアント PC に NETCONF over SSH をサポートする XML 管理ツールをインストールする必要があります。
- デバイスの XML サーバに適切なオプションを設定する必要があります。

# XML 管理インターフェイスを使用

このセクションでは、XML 管理インターフェイスを手動で構成して使用する方法について説明します。

**Note** デバイスのデフォルト設定で XML 管理インターフェイスを使用します。

## CLI による SSH および XML サーバー オプションの構成

デフォルトで SSH サーバーが有効になっています。SSH を無効にする場合は、クライアント PC で SSH セッションを開始する前に有効にする必要があります。

XML サーバー オプションを構成して、同時セッションの数とアクティブ セッションのタイムアウトを制御できます。XML ドキュメントの検証を有効にして、XML セッションを終了することもできます。

**Note** XML サーバ タイムアウトはアクティブ セッションだけに適用できます。

SSH の構成の詳細については、ご使用のプラットフォームの Cisco NX-OS セキュリティ構成ガイドを参照してください。

XML コマンドの詳細については、ご使用のプラットフォームの Cisco NX-OS システム マネジメント 構成ガイドを参照してください。

### 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 サーバーへの 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 サーバー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 サーバーが 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 サーバーは 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 620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 47: Cisco NX-OS の NETCONF 動作*

| NETCONF 動作 | 説明 | 例 |
|---|---|---|
| close-session | XML サーバーセッションを閉じます。 | NETCONF クロース セッション インスタンス, on page 632 |
| 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 637 |
| 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 638 |
| 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 633 |
| 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 634<br>NETCONF Rollback-On-Error インスタンス, on page 638 |

| 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 621 |
| 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 621 |
| 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 633 |
| 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 636 |
| 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 637 |
| 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 639 |

## 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 620」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 621 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 630 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 48:* 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された例を示しています。

### 構成コマンドと表示コマンドの組み合わせ

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>conf t ; interface ethernet 2/1 ; ip address 1.1.1.4/24 ; show xml
server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: cannot mix config and show in exec-command. Config cmds
before the show were executed.
Cmd:show xml server status</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

### <exec-command> を通して送信された表示 CLI コマンド

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element><cmd></nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

# NETCONF 応答

クライアントによって送信されるすべての XML 要求に対して、XML サーバーは RPC 応答タグ <rpc-reply> で囲まれた XML 応答を送信します。

## RPC 応答タグ

次の例は、RPC 応答タグ <rpc-reply> を示しています。

### RPC 応答タグ<rpc-reply>

```
<nc:rpc-reply message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
<ok/>
</nc:rpc-reply>]]>]]>
```

### RPC 応答エレメント

<ok> 、<data> 、そして <rpc-error> の要素は RPC 応答に表示される可能性があります。次の表は、<rpc-reply> タグに現れる可能性のある RPC 応答要素について説明しています。

*Table 49: 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 クロース セッション インスタンス, on page 632
- NETCONF 強制終了セッション インスタンス, on page 633
- NETCONF での構成インスタンスのコピー, on page 633
- NETCONF Edit Config インスタンス, on page 634
- NETCONF の Get 構成インスタンス, on page 635
- NETCONF ロック インスタンス, on page 636
- NETCONF ロック解除インスタンス, on page 637
- NETCONF コミット インスタンス：候補構成機能, on page 637
- NETCONF 確認済みコミット インスタンス, on page 638
- NETCONF Rollback-On-Error インスタンス, on page 638
- NETCONF 検証機能インスタンス, on page 639


## NETCONF クロース セッション インスタンス

次の例は、close-session 要求とそれに続く close-session 応答を表示しています。

### 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-configcopy 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>
<config>
<top xmlns="http://example.com/schema/1.2/config">
<interface>
<name>Ethernet0/0</name>
<mtu>100000</mtu>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>
```

### Rollback-On-Error への応答

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

## NETCONF 検証機能インスタンス

次の例は、NETCONF 検証機能の使用方法を示しています。文字列 urn:ietf:params:netconf:capability:rollback-on-error:1.0 は、NETCONF 検証機能を指定します。

### 検証要求

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
<candidate/>
</source>
</validate>
</rpc>]]>]]>
```

### 検証要求への応答

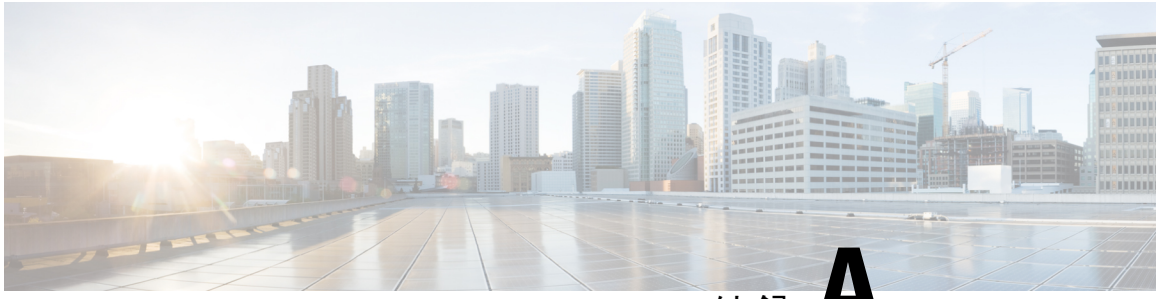
```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

# その他の参考資料

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

### RFC

| RFC | タイトル |
|---|---|
| RFC 4741 | NETCONF Configuration Protocol |
| RFC 4742 | セキュアシェル（SSH）経由の NETCONF 構成プロトコルの使用 |



付録 A

# ストリーミング テレメトリの送信元


- ストリーミング テレメトリについて （641 ページ）
- 注意事項と制約事項 （641 ページ）
- テレメトリで利用可能なデータ （641 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。

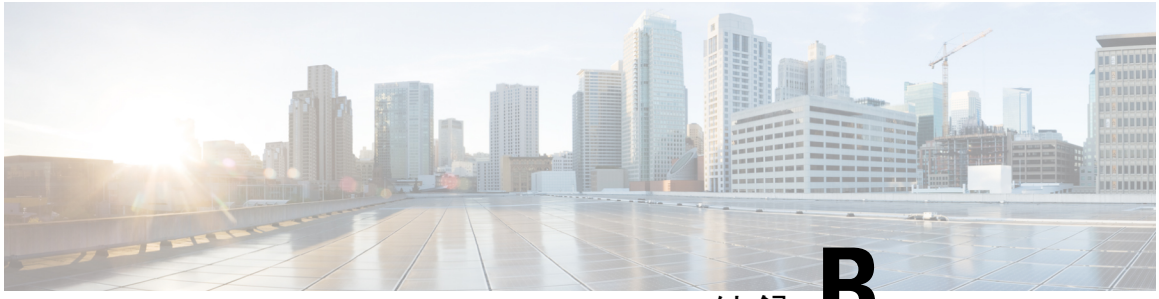
## 注意事項と制約事項

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



付録 B

# Websocket サブスクリプション


• WebSocket サブスクリプション （643 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ライ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。

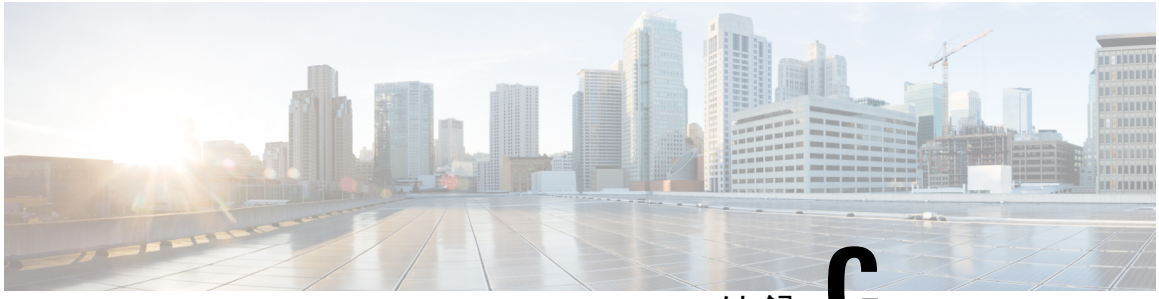
```
GET http://10.1.1.1/api/subscriptonRefresh.json?id=18374686685813276673
```

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。通知が失われないようにするには、サブスクリプションのリフレッシュ メッセージを少なくとも 90 秒ごとに送信する必要があります。

要約すると、WebSocket は、NX-OS REST API 内でイベント サブスクリプションのパブリッシャ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。

Websocket サブスクリプション



付録 C

# プログラマビリティに関する RFC


- プログラマビリティに関する RFC （647 ページ）

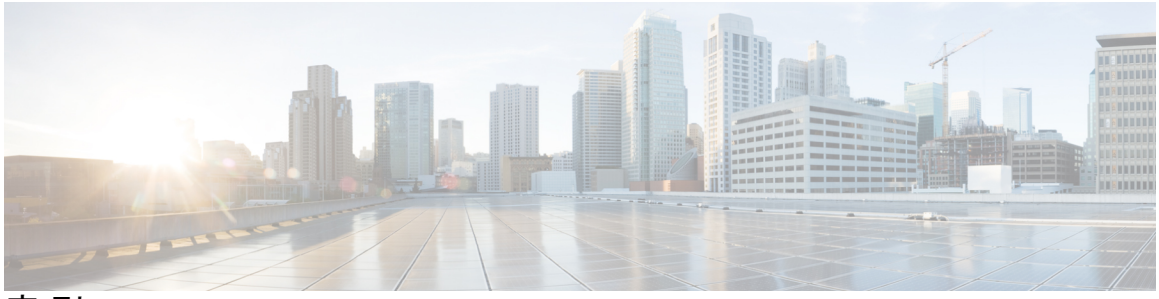

## プログラマビリティに関する RFC

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 *50 : RFC* 準拠標準**

| RFC | タイトル |
|---|---|
| RFC 5277 | NETCONF イベント通知 |
| RFC 6241 | ネットワーク設定プロトコル（NETCONF） |
| RFC 6243 | NETCONF の With-defaults 機能（**report-all** でのみサポート） |

プログラマビリティに関する RFC



# 索 引


## B

bash 19, 22
- アクセス 19
- 機能 bash シェル 19
- 例 22

bcm-shell 95, 97, 99
- CLI 95
- ファブリック モジュール 97
- ライン カード 99
- 例 99

## N

NX-API 237–240, 243, 261, 268, 273, 285, 293, 307
- CLI 240
- Cookie 239
- サンプル スクリプト 285
- セキュリティ 239
- トランスポート層 238
- メッセージ形式 239
- ユーザ インターフェイス 293, 307
- リクエスト要素 261
- 応答コード 273
- 応答要素 268
- 管理コマンド 243

## P

Python 103, 105–111
- Cisco パッケージ 103
- CLI コマンド API 105
- embedded event manager（EEM） 109
- non-interactive 108
- NX-OS セキュリティ 111
- NX-OS ネットワーク インターフェイス 110
- スクリプト 103
- 呼び出し 106
- 表示フォーマットの例 107

## T

tcl 115–118, 120
- cli コマンド 116
- history 116
- references 120
- tclquit コマンド 117
- オプション 117
- コマンドの区切り 117
- サンドボックス 118
- セキュリティ 118
- タブ補完 116
- 対話型ヘルプがない 115
- 変数 117

Tool Command Language。参照先： tcl

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。