APIベースのEPNM通知のトラブルシューティング

内容

概要

このドキュメントでは、REST APIを使用してデバイスの障害情報にアクセスする場合のEPNM通知のトラブルシューティング方法について説明します。

背景説明

実装するクライアントは、通知を送信するためにEvolved Programmable Network Manager(EPNM)で使用される2つのメカニズムのいずれかを処理し、サブスクライブできる必要があります。

EPNM API通知

通知は、ネットワーク管理者とオペレータに、ネットワークに関連する重要なイベントや問題について警告します。 これらの通知により、潜在的な問題を迅速に検出して解決できるため、ダウンタイムが短縮され、ネットワーク全体のパフォーマンスが向上します。

EPNMは、電子メールによる通知、指定されたレシーバへのSimple Network Management Protocol(SNMP)トラップ、外部SyslogサーバへのSyslogメッセージなど、さまざまな方法を処理できます。これらのメソッドに加えて、EPNMはRepresentational State Transfer(REST)アプリケーションプログラミングインターフェイス(REST API)も提供します。REST APIを使用すると、インベントリ、アラーム、サービスアクティベーション、テンプレート実行、および高可用性に関する情報を取得できます。

現在、APIベースの通知は、次の2つの異なるメカニズムを使用してサポートされています。

- コネクション型の通知：クライアントは事前定義されたURLにサブスクライブし、セキュアなHTTPSチャネルを介した基本認証を備えたWebSocketクライアントを使用します。
- コネクションレス型通知：ユーザは、POST要求としてExtensible Markup Language(XML)やJavaScript Object Notation(JSON)ペイロードを受け入れることができるREST Webサービスを持っていることが想定されます。

すべての通知は同じスキーマを共有し、JSONまたはXML形式で取得できます。

基本的なEPNM設定

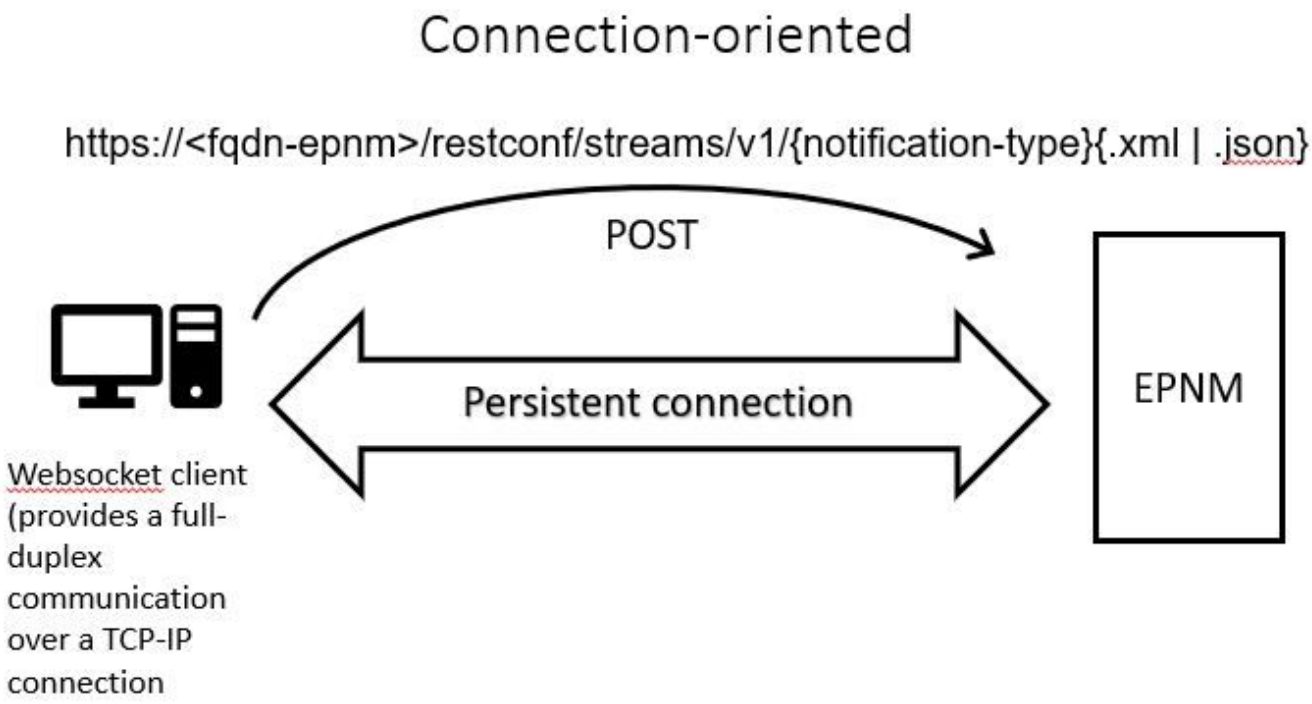
デフォルトでは、アラームとインベントリの通知は無効になっています。これらを有効にするには、 restconf-config.properties ファイルを次のように指定します（EPNMアプリケーションを再起動する必要はありません）。

```
/opt/CSCOlumos/conf/restconf/restconf-config.properties

epnm.restconf.inventory.notifications.enabled=true
epnm.restconf.alarm.notifications.enabled=true
```

コネクション型の通知

この図では、クライアントマシンがWebSocketを実行し、事前定義済みのURL、基本認証、およびセキュアHTTPSチャネルを使用してEPNMに登録しています。



WebSockets Pythonクライアントの実行

PythonのWebSocket-clientライブラリを使用して、クライアントマシンにWebSocketを作成できます。

```
import websocket
import time
import ssl
import base64

def on_message(ws, message):
    print(message)

def on_error(ws, error):
    print(error)

def on_close(ws, close_status_code, close_msg):
    print("### closed \###")

def on_open(ws):
    ws.send("Hello, Server!")

if __name__ == "__main__":
    username = "username"
    password = "password"
    credentials = base64.b64encode(f"{username}:{password}".encode("utf-8")).decode("utf-8")
    headers = {"Authorization": f"Basic {credentials}"}
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://10.122.28.3/restconf/streams/v1/inventory.json",
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close,
                                header=headers)
    ws.on_open = on_open
    ws.run_forever(sslopt={"cert_reqs": ssl.CERT_NONE})
```

コネクション型クライアントのサブスクリプション

このコードは、EPNMにサブスクライブするWebSocketクライアントを **wss://10.122.28.3/restconf/streams/v1/inventory.json**. Pythonを使用する WebSocketライブラリを作成して、送受信メッセージの接続と処理を行います。サブスクリプションは、次の場合もあります（登録する通知の種類に基づいて）。

- /restconf/streams/v1/alarm{.xml | .json}
- /restconf/streams/v1/service-activation{.xml | .json}
- /restconf/streams/v1/template-execution{.xml | .json}
- /restconf/streams/v1/all{.xml | .json}

「 on_message、 on_error と on_close 関数は、WebSocket接続がメッセージを受信したとき、エラーに遭遇したとき、または閉じられたときに呼び出されるコールバック関数です。「 **on_open** 関数は、WebSocket接続が確立され、使用可能な状態になったときに呼び出されるコールバックである。

「 username と password 変数には、リモートサーバへのアクセスに必要なログインクレデンシャルが設定されます。これらのクレデンシャルは、 base64 モジュールを追加し、WebSocket要求のヘッダーに追加します。

「 run_forever メソッドは、接続を開始し、無制限に開き続け、サーバから送信されるメッセージをリッスンするためにWebSocketオブジェクトで呼び出されます。「 sslopt パラメータは、接続のSSL/TLSオプションを設定するために使用されます。「 CERT_NONE フラグは証明書の検証を無効にします。

コードを実行します。WebSocketで通知を受信する準備を整えるには、次の手順を実行します。

```
(env) devasc@labvm:~/epnm$ python conn-oriented.py
--- request header ---
GET /restconf/streams/v1/inventory.json HTTP/1.1
Upgrade: websocket
Host: 10.122.28.3
Origin: https://10.122.28.3
Sec-WebSocket-Key: YYYYYYYYYYY
Sec-WebSocket-Version: 13
Connection: Upgrade
Authorization: Basic XXXXXXXXXXXX


-----------------------
--- response header ---
HTTP/1.1 101
Set-Cookie: JSESSIONID=5BFB68B0126226A0A13ABE595DC63AC9; Path=/restconf; Secure; HttpOnly
Strict-Transport-Security: max-age=31536000;includeSubDomains
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Accept: Ozns7PGgHjrXj0nAgnlhbyVKPjc=
Date: Thu, 30 Mar 2023 16:18:19 GMT
Server: Prime
-----------------------
Websocket connected
++Sent raw: b'\x81\x8es\x99ry;\xfc\x1e\x15\x1c\xb5R*\x16\xeb\x04\x1c\x01\xb8'
++Sent decoded: fin=1 opcode=1 data=b'Hello, Server!'
++Rcv raw: b'\x81\x0eHello, Server!'
++Rcv decoded: fin=1 opcode=1 data=b'Hello, Server!'
Hello, Server!
```

次のDBクエリを使用して、サーバへの通知サブスクリプションを確認できます。

```
ade # ./sql_execution.sh "SELECT * from RstcnfNtfctnsSbscrptnMngr WHERE CONNECTIONTYPE = 'connection-or
```

ネットワークを可視化し、 conn-oriented.txt ファイル（DBクエリの結果）をHTMLに変換するには、次のようなツールを使用します aha （ここではUbuntuマシンでの使用例を示します）。

```
devasc@labvm:~/tmp$ sudo apt-get install aha
devasc@labvm:~/tmp$ cat conn-oriented.txt | aha > conn-oriented.html
```

次に、 conn-oriented.html ブラウザ内のファイル：

ID	INSTANCE_VERSION	CLASSNAME	CONNECTIONTYPE	ENDPOINTURL	SUBSCRIBEDUSER	SUBSCRIPTIONCREATIONTIME	SUBSCRIPTIONID	SUBSCRIPTIONTOPIC	SUBSCRIPTIONUPDATETIME
2361930571	0	cnfNtfctnsSbscrptnMngr3	connection-oriented	de4d9082-c05c-439b-a7e7-65016623fee1	root	Mon Aug 28 16:13:04 BRT 2023	3648313822269611499	inventory	Mon Aug 28 16:13:04 BRT 2023

EPNMのオンラインドキュメントでは、一度確立されると、アプリケーションのライフサイクルを通じて同じ接続が維持されます。

- クライアントがサーバから切断されるまで
- メンテナンスまたはフェールオーバー中にサーバがダウンするまで

何らかの理由で特定のサブスクリプションを削除する必要がある場合は、 HTTP DELETE 要求を行います。 SUBSCRIPTIONID URLで指定します。 https://
を参照。例：

```
devasc@labvm:~/tmp$ curl --location --insecure --request DELETE 'https://10.122.28.3/restconf/data/v1/c
```

メッセージ、DEBUGエントリ、 show log、使用するファイル名、SQL出力

コネクション型メカニズムを使用するクライアントが通知を適切に受信しない原因をトラブルシューティングするには、表示されたDBクエリを実行して、サブスクリプションが存在するかどうかを確認します。これが存在しない場合は、クライアントの所有者にサブスクリプションを発行するように依頼します。

それまでの間、DEBUGレベルは com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter そのため、サブスクリプションが送信されるたびに受信できます。

```
ade # sudo /opt/CSCOlumos/bin/setLogLevel.sh com.cisco.nms.nbi.epnm.restconf.notifications.handler.Noti
```

サブスクリプションの送信後、WebSocketクライアントのIPアドレスを持つエントリが localhost_access_log.txt：を入力します。

```
ade # zgrep -h '"GET /restconf/streams/.* HTTP/1.1" 101' $(ls -1t /opt/CSCOlumos/logs/localhost_access_
```

最後に、再度DBを確認します(タイムスタンプが localhost_access_log.txt).

H	CONNECTIONTYPE	ENDPOINTURL	ISENDPOINTREACHABLE	NOTIFICATIONFORMAT	SUBSCRIBEDUSER	SUBSCRIPTIONCREATIONTIME	SUBSCRIPTIONID	SUBSCRIPTIOI
	connection-oriented	852a674a-e3d0-4ecc-8ea0-787af30f1305	0	json	root	Mon Aug 28 22:17:06 BRT 2023	8743327441517764088	inventory

次のログは、サブスクリプションのPOST要求がいつ送信されるかを示しています。

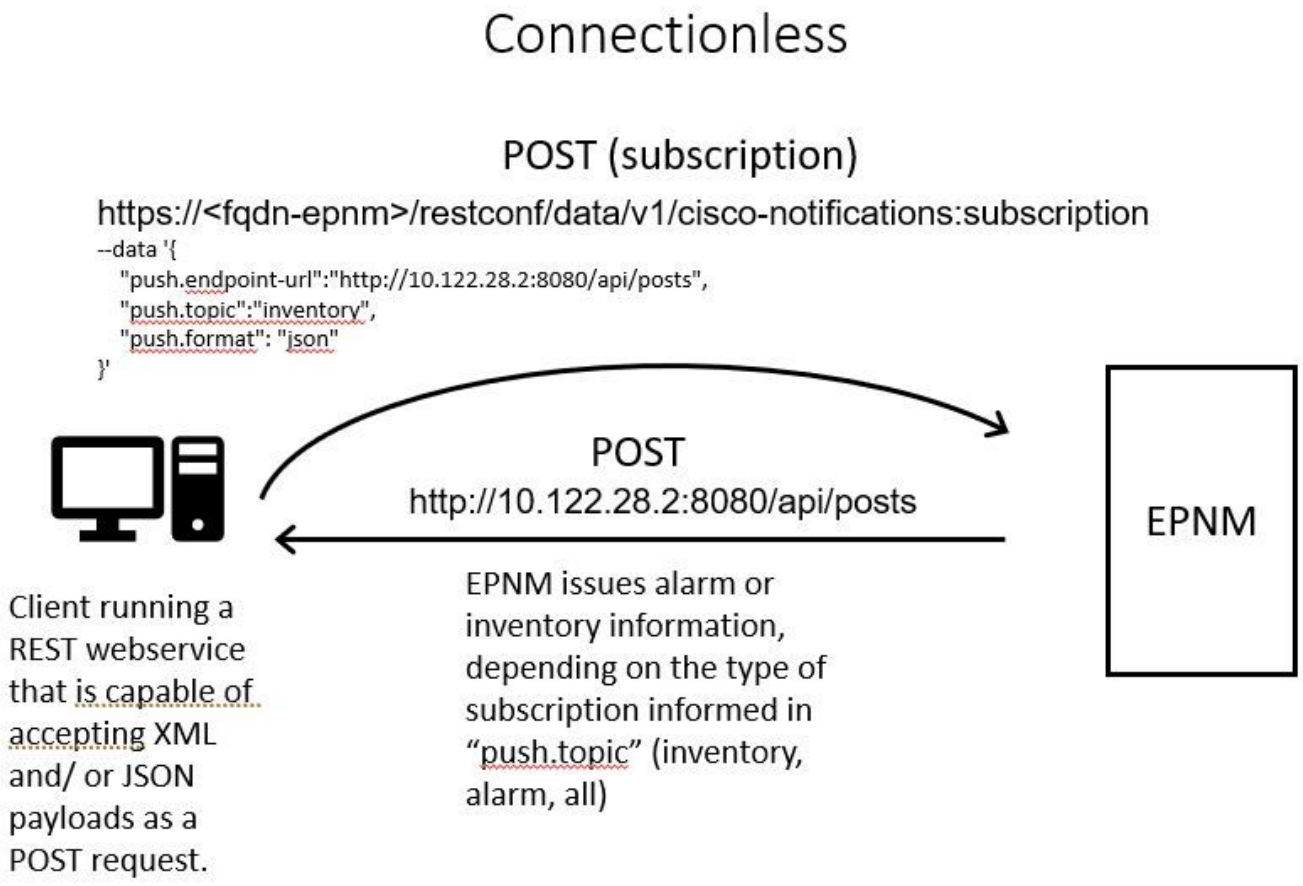
```
ade # grep -Eh 'DEBUG com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter
```

接続が維持されている限り、push-change-updateタイプの通知がEPN-Mサーバから通知をサブスクライブしたすべてのクライアントに送信されます。この例は、NCS2kのホスト名が変更されたときにEPNMによって送信される通知の1つを示しています。

```
{ "push.push-change-update":{ "push.notification-id":2052931975556780123, "push.topic":"inventory", "pu
```

コネクションレス型通知

次に、次の場合のワークフローを示します connectionless 通知：



REST WebサービスPythonクライアントの実行

ユーザは、XMLペイロードやJSONペイロードをPOST要求として受け入れることができるREST Webサービスを持っていることが

想定されます。このRESTサービスは、 Cisco EPNMrestconf notifications frameworkは通知を発行します。これは リモートマシンにインストールされるREST Webサービスの例：

```
from flask import Flask, request, jsonify app = Flask(__name__) @ app.route('/api/posts', methods=['POS
```

これは、単一のエンドポイントを定義するPython Flask Webアプリケーションです /api/posts 受け入れる **HTTP POST** 保持されます。「 create_post() 関数は常に呼び出されます。 **HTTP POST** 要求が行われます /api/posts. 内部 create_post() 関数を呼び出した場合、要求からデータを取得するには、 **request.get_json()**JSONペイロードのディクショナリを返します。次に、ペイロードに次の文字列が出力されます **print(post_data)** デバッグに使用します。その後、キーを含む応答メッセージが作成されます message および値 **Post created successfully**（辞書形式）。この応答メッセージは、HTTPステータスコード201（作成済み）でクライアントに返されます。

「 if __name__ == '__main__': blockは、スクリプトがモジュールとしてインポートされるのではなく、メインプログラムとして実行されているかどうかをチェックする標準のPythonコンストラクトです。スクリプトがメインプログラムとして実行される場合、Flaskアプリケーションが起動され、指定されたIPアドレスとポートで実行されます。「 **debug=True** この引数はデバッグモードを有効にします。デバッグモードでは、詳細なエラーメッセージが表示され、コードに変更が加えられた際にサーバが自動的にリロードされます。

プログラムを実行して、 REST webサービス：

```
(venv) [apinelli@centos8_cxlabs_spo app]$ python connectionless.py * Serving Flask app 'connectionless'
```

コネクションレス型クライアントのサブスクリプション

ユーザは通知をサブスクライブします。 REST購読するトピックとともにサービスエンドポイントが送信されます。この場合、トピックは次のとおりです all を参照。

```
[apinelli@centos8_cxlabs_spo ~]$ curl --location -X POST --insecure 'https://10.122.28.3/restconf/data/
```

予想される応答は201応答で、応答の本文のサブスクリプションからの詳細も含まれています。

```
{ "push.notification-subscription": { "push.subscription-id": 7969974728822328535, "push.subscribed-use
```

ユーザがGET要求でサブスクライブしている通知のリストを取得できます。

```
curl --location --insecure 'https://10.122.28.3/restconf/data/v1/cisco-notifications:subscription' \ --
```

得られた応答は次のとおりです。

```
{ "com.response-message": { "com.header": { "com.firstIndex": 0, "com.lastIndex": 1 }, "com.data": { "p
```

メッセージ、DEBUGエントリ、 show log, 使用するファイル名、SQL出力

応答から、サブスクリプションが2つあることに注目してください。1つは all ("**push.topic": "all**") インベントリ用に1つ ("**push.topic": "inventory**")を参照。データベースに対するクエリを使用して確認できます(サブスクリプションのタイプが「コネクションレス型」であり、 SUBSCRIPTIONID フィールドは出力に一致します。 GET コマンドを入力します (黄色で強調表示) 。

```
ade # ./sql_execution.sh "SELECT * from RstcnfNtfctnsSbscrptnMngr WHERE CONNECTIONTYPE = 'connection-le
```

```
        ID     INSTANCE_VERSION CLASSNAME                CONNECTIONTYPE     ENDPOINTURL                        SUBSCRIBEDUSER  SUBSCRIPTIONCREATIONTIME       SUBSCRIPTIONID        SUBSCRIPTIONTOPIC
---------- -------------------- ---------------------- ------------------ ---------------------------------- --------------- ------------------------------ --------------------- ---------------------
2361930573                    0 cnfNtfctnsSbscrptnMngr3 connection-less    http://10.122.28.2:8080/api/posts  root            Tue Aug 29 10:02:05 BRT 2023   7969974728822328535   all
 337897630                    0 cnfNtfctnsSbscrptnMngr3 connection-less    http://10.122.28.2:8080/api/posts  root            Fri Mar 31 17:45:47 BRT 2023   2985507860170167151   inventory
```

コネクションレス型サブスクリプションを削除する必要がある場合は、 HTTP DELETE 削除するサブスクリプションIDを指定して要求します。削除すると仮定します **subscription-id 2985507860170167151**：を入力します。

```
curl --location --insecure --request DELETE 'https://10.122.28.3/restconf/data/v1/cisco-notifications:s
```

DBに再度クエリを実行すると、 SUBSCRIPTIONID 等しい 7969974728822328535.

インベントリの変更が発生すると、クライアントは通知を印刷します(これは connection-oriented に関するセクションに表示される通知 connected-oriented 201応答が続きます。

```
(venv) [apinelli@centos8_cxlabs_spo app]$ python connectionless.py * Serving Flask app 'connectionless'
```

結論

このドキュメントでは、EPNM(connectionlessと connection-oriented)について説明し、シミュレーションのベースとして使用できるそれぞれのクライアントの例を示します。

関連情報

- https://www.cisco.com/c/dam/en/us/td/docs/net_mgmt/epn_manager/RESTConf/Cisco_Evolved_Programmable_Network_Manager_5_1_2_F
- テクニカル サポートとドキュメント - Cisco Systems

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。