Risoluzione dei problemi relativi alle notifiche EPNM basate su API

Sommario

Introduzione

In questo documento viene descritto come risolvere i problemi relativi alle notifiche EPNM quando si utilizza l'API REST per accedere alle informazioni sugli errori dei dispositivi.

Premesse

Il client implementato deve essere in grado di gestire e sottoscrivere uno qualsiasi dei due meccanismi utilizzati da Evolved Programmable Network Manager (EPNM) per l'invio di notifiche.

Notifiche API EPNM

Le notifiche avvisano gli amministratori e gli operatori della rete in merito a eventi o problemi importanti relativi alla rete. Queste notifiche garantiscono che i potenziali problemi vengano rilevati e risolti rapidamente, riducendo i tempi di inattività e migliorando le prestazioni complessive della rete.

EPNM è in grado di gestire diversi metodi, ad esempio notifiche via e-mail, trap SNMP (Simple Network Management Protocol) verso ricevitori specifici o messaggi Syslog verso server Syslog esterni. Oltre a questi metodi, EPNM fornisce anche un'interfaccia API REST (Representative State Transfer Application Programming Interface) che può essere utilizzata per recuperare informazioni su inventario, allarmi, attivazione dei servizi, esecuzione dei modelli e alta disponibilità.

Le notifiche basate su API sono attualmente supportate con l'utilizzo di due diversi meccanismi:

- Notifiche orientate alla connessione: il client esegue la sottoscrizione a un URL predefinito e utilizza un client WebSocket con autenticazione di base tramite un canale HTTPS protetto.
- Notifiche senza connessione: è previsto che l'utente disponga di un servizio Web REST in grado di accettare payload XML (Extensible Markup Language) e/o JSON (JavaScript Object Notation) come richiesta POST.

Tutte le notifiche condividono lo stesso schema e possono essere recuperate in formato JSON o XML.

Configurazione EPNM di base

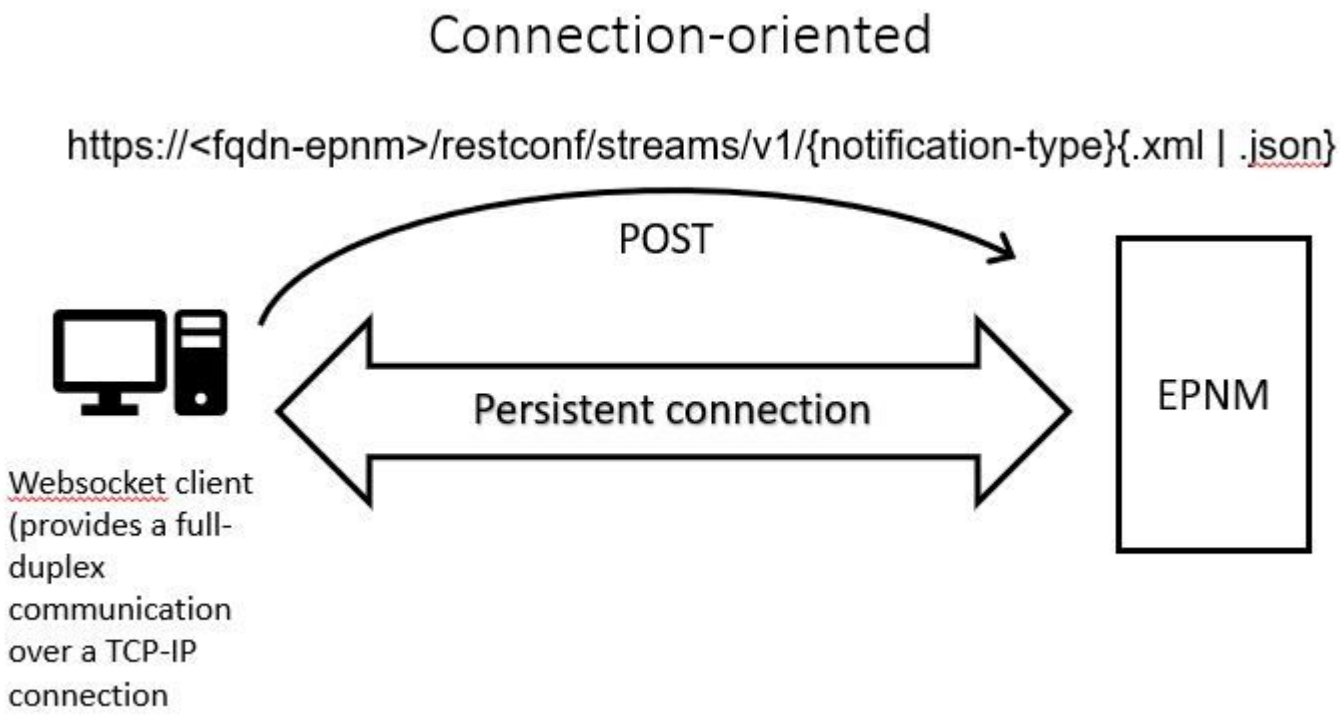
Per impostazione predefinita, le notifiche di allarme e di inventario sono disabilitate. Per abilitarli, modificare il restconf-config.properties file come indicato (non è necessario riavviare l'applicazione EPNM):

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

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

Notifiche orientate alla connessione

Nella figura, il computer client esegue un WebSocket e sottoscrive l'EPNM con un URL predefinito, con autenticazione di base e tramite un canale HTTPS protetto.



Eseguire un client Python WebSockets

La libreria client WebSocket in Python può essere utilizzata per creare un WebSocket nel computer client.

```
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})
```

Sottoscrizione di un client orientato alla connessione

Questo codice imposta un client WebSocket che sottoscrive EPNM in **wss://10.122.28.3/restconf/streams/v1/inventory.json**. Utilizza il Python WebSocketper stabilire la connessione e gestire i messaggi in e out. L'abbonamento può anche essere (in base al tipo di notifica che si desidera sottoscrivere):

- /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}

OSPF (Open Shortest Path First) on_message, on_error e on_close le funzioni di richiamata sono funzioni richiamate rispettivamente quando la connessione WebSocket riceve un messaggio, rileva un errore o è chiusa. OSPF (Open Shortest Path First) **on_open** La funzione è un callback che viene chiamato quando la connessione WebSocket viene stabilita e pronta per l'utilizzo.

OSPF (Open Shortest Path First) username e password le variabili vengono impostate sulle credenziali di accesso necessarie per accedere al server remoto. Queste credenziali vengono quindi codificate con base64 e aggiunto alle intestazioni della richiesta WebSocket.

OSPF (Open Shortest Path First) run_forever viene chiamato sull'oggetto WebSocket per avviare la connessione, mantenerla aperta per un periodo di tempo indefinito e ascoltare i messaggi provenienti dal server. OSPF (Open Shortest Path First) sslopt viene utilizzato per configurare le opzioni SSL/TLS per la connessione. OSPF (Open Shortest Path First) CERT_NONE flag disattiva la convalida della certificazione.

Eseguire il codice Per preparare WebSocket alla ricezione delle notifiche:

```
(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!
```

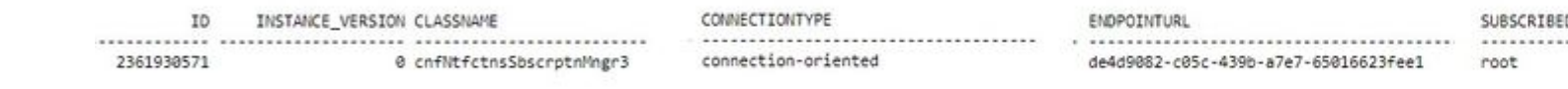
È possibile controllare le sottoscrizioni di notifica al server con questa query del database:

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

Per una migliore visualizzazione conn-oriented.txt (che è il risultato della query DB), è possibile convertirlo in HTML utilizzando uno strumento come aha (qui il suo uso è illustrato in una macchina Ubuntu):

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

Aprire quindi il conn-oriented.html file in un browser:



ID	INSTANCE_VERSION	CLASSNAME	CONNECTIONTYPE	ENDPOINTURL	SUBSCRIBE
2361930571	0	cnfNtfctnsSbscrptnMngr3	connection-oriented	de4d9082-c05c-439b-a7e7-65016623fee1	root

Dalla documentazione in linea di EPNM, una volta stabilita, la stessa connessione rimane attiva per tutto il ciclo di vita dell'applicazione:

- finché il client non si disconnette dal server

- finché il server non si blocca per manutenzione o durante un failover

Se, per qualche motivo, è necessario eliminare una sottoscrizione specifica, è possibile inviare una HTTP DELETE richiesta con il SUBSCRIPTIONID specificato nell'URL https://
. Ad esempio:

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

Verifica dei messaggi, voci DEBUG, show log, Nome file utilizzato, Output SQL

Per risolvere i problemi relativi alla mancata ricezione delle notifiche da parte di un client che utilizza un meccanismo orientato alla connessione, è possibile eseguire la query del database indicata e verificare se la sottoscrizione è presente o meno. Se non è presente, chiedere al proprietario del client di eseguire la sottoscrizione.

Nel frattempo, è possibile abilitare il livello DEBUG in com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter in modo da poterlo intercettare ogni volta che viene inviata la sottoscrizione:

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

Dopo l'invio della sottoscrizione, è possibile verificare se una voce con l'indirizzo IP del client WebSocket viene visualizzata in localhost_access_log.txt:

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

Infine, controllare nuovamente il database (si noti che il timestamp corrisponde alla voce in localhost_access_log.txt).

H	CONNECTIONTYPE	ENDPOINTURL	ISENDPOINTREACHABLE	NOTIFICATIONFORMAT	SUBSCR
	connection-oriented	852a674a-e3d0-4ecc-8ea0-787af30f1305	0	json	root

Il log successivo mostra quando vengono inviate le richieste POST per le sottoscrizioni:

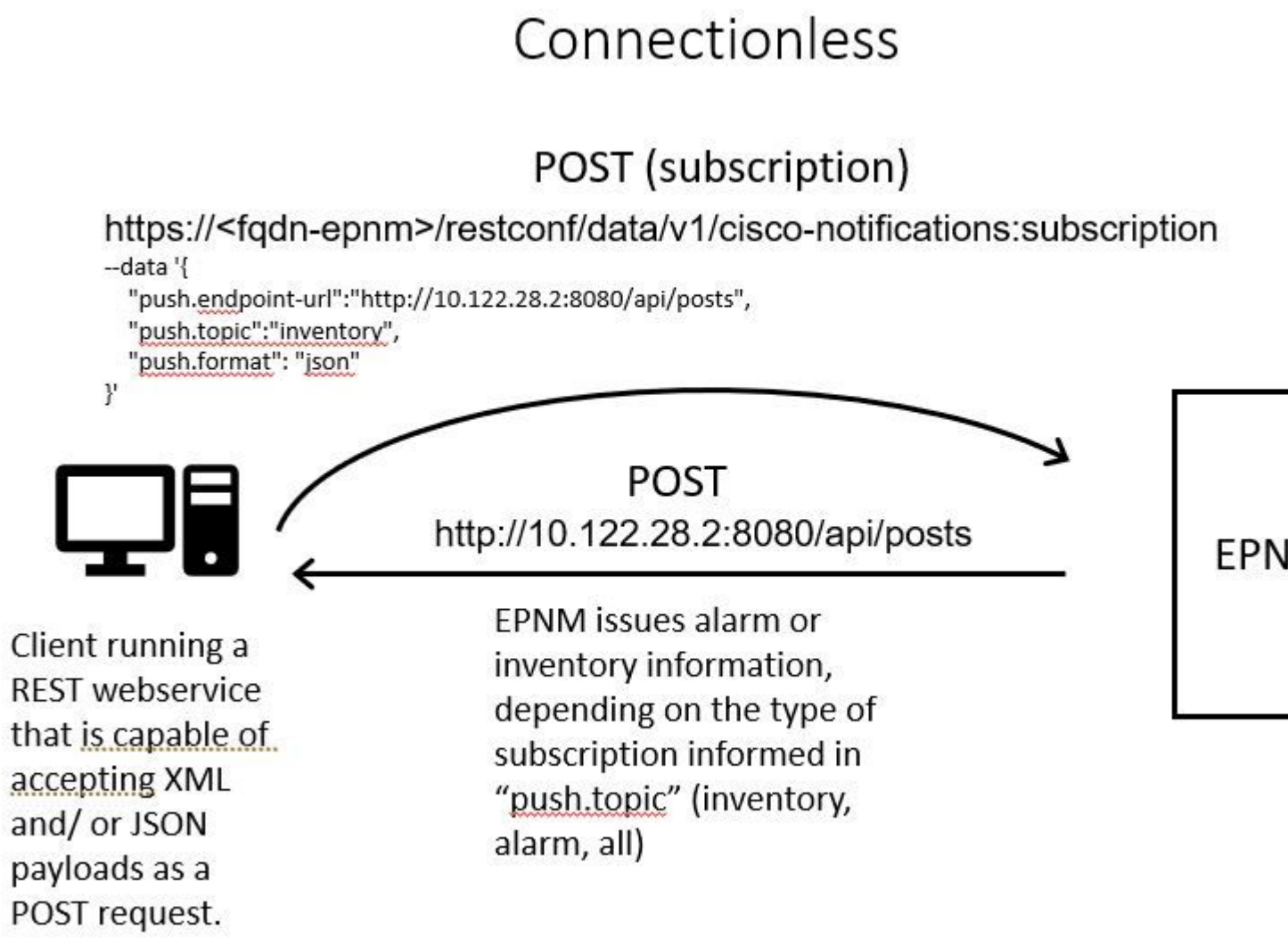
```
ade # grep -Eh 'DEBUG com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter
```

Finché la connessione rimane attiva, una notifica di tipo push-change-update viene inviata dal server EPN-M a tutti i client che hanno effettuato la sottoscrizione per le notifiche. Nell'esempio viene illustrata una delle notifiche inviate da EPNM quando il nome host di un NCS2k viene modificato:

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

Notifiche senza connessione

Il passaggio successivo è il workflow nel caso di connectionless notifiche:



Esegui un client Python servizio Web REST

È previsto che l'utente disponga di un servizio Web REST in grado di accettare payload XML e/o JSON come richiesta POST. Questo servizio REST è l'endpoint a cui Cisco EPNMrestconf notifications framework pubblica notifiche. Si tratta di un esempio di un servizio Web REST da installare nel computer remoto:

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

Questa è un'applicazione Web Python Flask che definisce un singolo endpoint /api/posts che accetta **HTTP POST** richieste. OSPF (Open Shortest Path First) create_post() viene chiamata ogni volta che un **HTTP POST** richiesta inviata a /api/posts. All'interno del create_post() funzione, i dati della richiesta in ingresso vengono recuperati utilizzando **request.get_json()**, che restituisce un dizionario del payload JSON. Il payload viene quindi stampato con **print(post_data)** per scopi di debug. In seguito, viene creato un messaggio di risposta con la chiave

message e valore **Post created successfully** (in formato dizionario). Questo messaggio di risposta viene quindi restituito al client con un codice di stato HTTP 201 (creato).

OSPF (Open Shortest Path First) if __name__ == '__main__': è un costrutto Python standard che verifica se lo script viene eseguito come programma principale, anziché importato come modulo. Se lo script viene eseguito come programma principale, avvia l'applicazione Flash e lo esegue sulla porta e sull'indirizzo IP specificati. OSPF (Open Shortest Path First) **debug=True** abilita la modalità di debug, che fornisce messaggi di errore dettagliati e il ricaricamento automatico del server quando vengono apportate modifiche al codice.

Eseguire il programma per avviare il REST servizio Web:

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

Sottoscrizione di un client senza connessione

L'utente sottoscrive le notifiche: RESTl'endpoint del servizio viene inviato insieme all'argomento per la sottoscrizione a. In questo caso, l'argomento è all.

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

La risposta prevista è una risposta del 2011, insieme ai dettagli della sottoscrizione nel corpo della risposta:

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

È possibile ottenere l'elenco delle notifiche a cui l'utente è iscritto con una richiesta GET:

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

La risposta ottenuta è la seguente:

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

Verifica dei messaggi, voci DEBUG, show log, Nome file utilizzato, Output SQL

Si noti dalla risposta che sono presenti due sottoscrizioni: una per all (**"push.topic": "all"**) e uno per il magazzino (**"push.topic": "inventory"**). È possibile confermarla con una query sul database. Si noti che il tipo di sottoscrizione è "senza connessione" e SUBSCRIPTIONID corrispondono all'output del GET come evidenziato in giallo):

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

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

Se è necessario eliminare una sottoscrizione senza connessione, è possibile inviare una HTTP DELETE con l'ID sottoscrizione che si desidera eliminare. Si supponga di voler eliminare **subscription-id 2985507860170167151**:

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

Se si esegue nuovamente una query sul database, verrà visualizzata solo la sottoscrizione con SUBSCRIPTIONID uguale a 7969974728822328535.

Quando si verifica una modifica nell'inventario, il client stampa le notifiche (che sono dello stesso tipo del connection-oriented notifiche visualizzate nella sezione informazioni su connected-oriented seguito dalla risposta del 201:

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

Conclusioni

In questo documento, i due tipi di notifiche basate su API che possono essere configurati in EPNM (connectionlesse connection-oriented) e vengono forniti esempi dei rispettivi clienti che possono essere utilizzati come base per la simulazione.

Informazioni correlate

- https://www.cisco.com/c/dam/en/us/td/docs/net_mgmt/epn_manager/RESTConf/Cisco_Evolved_Programmable_Network_Manager_5_1_2_F
- Documentazione e supporto tecnico â€“ Cisco Systems

Informazioni su questa traduzione

Cisco ha tradotto questo documento utilizzando una combinazione di tecnologie automatiche e umane per offrire ai nostri utenti in tutto il mondo contenuti di supporto nella propria lingua. Si noti che anche la migliore traduzione automatica non sarà mai accurata come quella fornita da un traduttore professionista. Cisco Systems, Inc. non si assume alcuna responsabilità per l’accuratezza di queste traduzioni e consiglia di consultare sempre il documento originale in inglese (disponibile al link fornito).