Dépannage des notifications EPNM basées sur API

Table des matières

Introduction

Ce document décrit comment dépanner les notifications EPNM lorsque l'API REST est utilisée pour accéder aux informations sur les pannes des périphériques.

Informations générales

Le client que vous implémentez doit être capable de gérer et de s'abonner à l'un des deux mécanismes utilisés par le gestionnaire EPNM (Evolved Programmable Network Manager) pour envoyer des notifications.

Notifications API EPNM

Les notifications alertent les administrateurs réseau et les opérateurs des événements importants ou des problèmes liés au réseau. Ces notifications permettent de s'assurer que les problèmes potentiels sont détectés et résolus rapidement, ce qui réduit les temps d'arrêt et améliore les performances globales du réseau.

EPNM peut gérer différentes méthodes, telles que les notifications par e-mail, les déroutements SNMP (Simple Network Management Protocol) vers des destinataires spécifiés ou les messages Syslog vers des serveurs Syslog externes. Outre ces méthodes, EPNM fournit également une interface de programmation REST (Representational State Transfer Application Programming Interface) qui peut être utilisée afin de récupérer des informations sur l'inventaire, les alarmes,

l'activation de service, l'exécution de modèle et la haute disponibilité.

Les notifications basées sur l'API sont actuellement prises en charge à l'aide de deux mécanismes différents :

- Notifications orientées connexion : le client s'abonne à une URL prédéfinie et utilise un client WebSocket avec authentification de base via un canal HTTPS sécurisé.
- Notifications sans connexion : l'utilisateur doit disposer d'un service Web REST capable d'accepter des charges utiles XML (Extensible Markup Language) et/ou JSON (JavaScript Object Notation) comme demande POST.

Toutes les notifications partagent le même schéma et peuvent être récupérées au format JSON ou XML.

Configuration EPNM de base

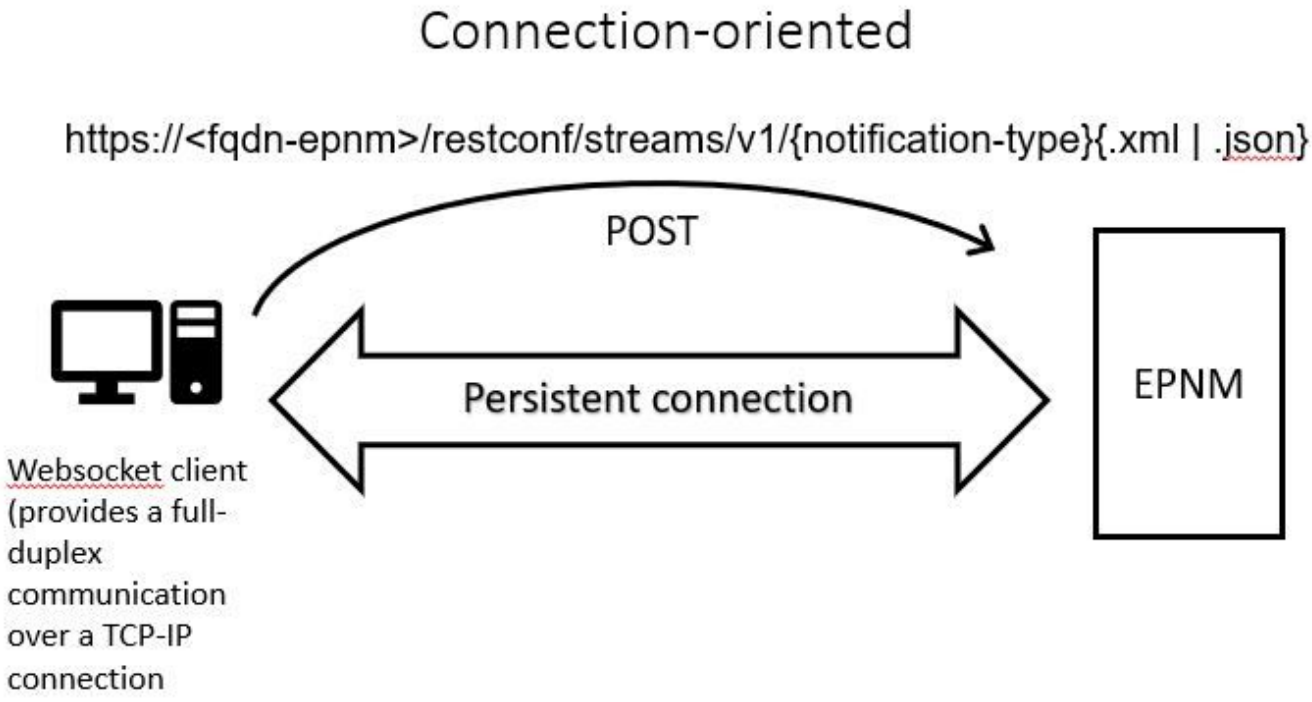
Par défaut, les notifications d'alarme et d'inventaire sont désactivées. Afin de les activer, modifiez la restconf-config.properties comme indiqué (il n'est pas nécessaire de redémarrer l'application EPNM) :

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

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

Notifications orientées connexion

Dans l'image, la machine cliente exécute un WebSocket et s'abonne à l'EPNM avec une URL prédéfinie, avec une authentification de base et via un canal HTTPS sécurisé.



Exécuter un client Python WebSockets

La bibliothèque WebSocket-client en Python peut être utilisée pour créer un WebSocket dans la machine cliente.

```
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})
```

Abonnement à un client orienté connexion

Ce code configure un client WebSocket qui s'abonne à EPNM à **wss://10.122.28.3/restconf/streams/v1/inventory.json**. Il utilise le Python `WebSocket`afin d'établir la connexion et de gérer les messages entrants et sortants. L'abonnement peut également être (en fonction du type de notification auquel vous souhaitez vous abonner) :

- /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}

Les `on_message`, `on_error` et `on_close` Les fonctions sont des fonctions de rappel qui sont appelées lorsque la connexion WebSocket reçoit un message, rencontre une erreur ou est fermée, respectivement. Les **`on_open`** est un rappel qui est appelé lorsque la connexion WebSocket est établie et prête à l'emploi.

Les `username` et `password` sont définies sur les identifiants de connexion requis pour accéder au serveur distant. Ces informations d'identification sont ensuite codées avec le `base64` et ajouté aux en-têtes de la requête WebSocket.

Les `run_forever` est appelée sur l'objet WebSocket afin de démarrer la connexion, de la maintenir ouverte indéfiniment et d'écouter les messages provenant du serveur. Les `sslopt` est utilisé pour configurer les options SSL/TLS pour la connexion. Les `CERT_NONE` désactive la validation de certification.

Exécutez le code Pour que WebSocket soit prêt à recevoir les notifications :

```
(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: YYYYYYYYYY
Sec-WebSocket-Version: 13
Connection: Upgrade
Authorization: Basic XXXXXXXXXXX


-----------------------
--- 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!
```

Vous pouvez vérifier les abonnements de notification au serveur avec cette requête de base de données :

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

Afin de mieux visualiser le conn-oriented.txt (qui est le résultat de la requête DB), vous pouvez le convertir en HTML en utilisant un outil comme aha (son utilisation est illustrée dans une machine Ubuntu) :

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

Ensuite, ouvrez le conn-oriented.html fichier dans un navigateur :

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

Dans la documentation en ligne d'EPNM, une fois établie, la même connexion reste active tout au long du cycle de vie de l'application :

- jusqu'à ce que le client se déconnecte du serveur
- jusqu'à ce que le serveur soit arrêté pour maintenance ou pendant un basculement

Si, pour une raison quelconque, vous devez supprimer un abonnement spécifique, vous pouvez envoyer un HTTP DELETE demande auprès du SUBSCRIPTIONID spécifié dans l'URL https://
. Exemple :

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

Vérification des messages, entrées DEBUG, show log, Nom de fichier utilisé, Résultats SQL

Afin de dépanner pourquoi un client qui utilise un mécanisme orienté connexion ne reçoit pas correctement les notifications, vous pouvez exécuter la requête de base de données indiquée et vérifier si l'abonnement est présent ou non. S'il n'est pas présent, demandez au propriétaire du client de s'assurer d'émettre l'abonnement.

En attendant, vous pouvez activer le niveau DEBUG
dans com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter vous pouvez donc l'intercepter à chaque envoi de l'abonnement :

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

Une fois l'abonnement envoyé, vous pouvez vérifier si une entrée avec l'adresse IP du client WebSocket apparaît dans localhost_access_log.txt:

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

Enfin, vérifiez à nouveau la base de données (notez que l'horodatage correspond à l'entrée dans 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	8743274415177640 88	inventory

Le journal suivant indique quand les demandes POST pour les abonnements sont envoyées :

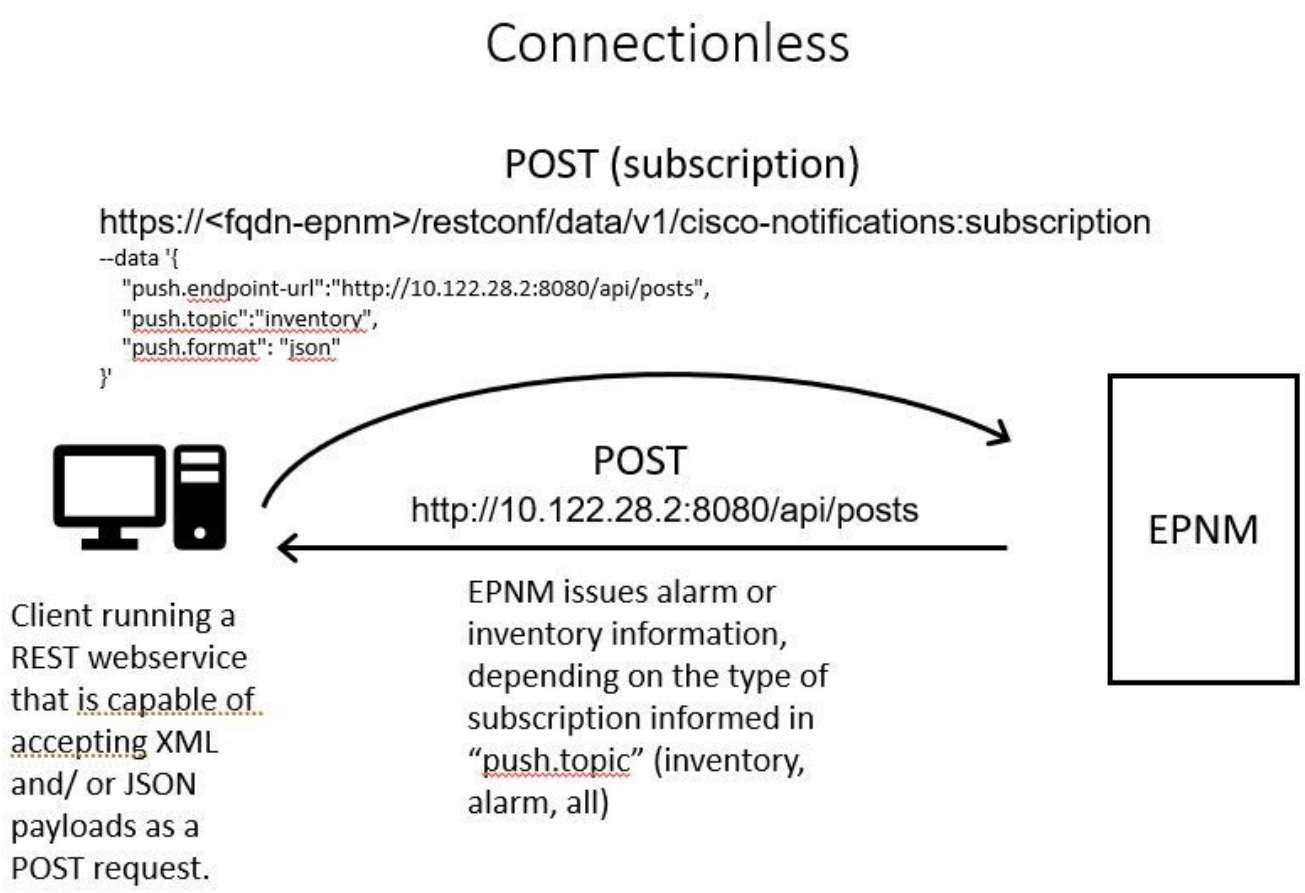
```
ade # grep -Eh 'DEBUG com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter
```

Tant que la connexion est maintenue active, une notification de type push-change-update est envoyée depuis le serveur EPN-M à tous les clients qui se sont abonnés aux notifications. L'exemple montre l'une des notifications envoyées par le gestionnaire EPNM lorsque le nom d'hôte d'un NCS2k est modifié :

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

Notifications sans connexion

La suivante est le workflow dans le cas de connectionless notifications :



Exécuter un client Python REST Webservice

L'utilisateur doit disposer d'un service Web REST capable d'accepter des données utiles XML et/ou JSON en tant que requête POST. Ce service REST est le point de terminaison auquel le Cisco EPNMrestconf notifications framework publie des notifications. C'est un exemple d'un service Web REST à installer sur l'ordinateur distant :

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

Il s'agit d'une application Web Python Flask qui définit un point de terminaison unique /api/posts qui accepte **HTTP POST** demandes. Les create_post() est appelée chaque fois qu'une **HTTP POST** demande est adressée à /api/posts. À l'intérieur create_post() , les données de la demande qui arrive sont récupérées à l'aide de **request.get_json()**, qui renvoie un dictionnaire de la charge utile JSON. La charge utile est ensuite imprimée avec **print(post_data)** à des fins de débogage. Ensuite, un message de réponse est créé avec la clé message et valeur **Post created successfully** (au format dictionnaire). Ce message de réponse est ensuite renvoyé au client avec un code d'état HTTP de 201 (créé).

Les if __name__ == '__main__': block est une construction Python standard qui vérifie si le script s'exécute en tant que programme principal, plutôt qu'importé en tant que module. Si le script s'exécute en tant que programme principal, il démarre l'application Flask et l'exécute sur l'adresse IP et le port spécifiés. Les **debug=True** active le mode de débogage, qui fournit des messages d'erreur détaillés et le rechargement automatique du serveur lorsque des modifications sont apportées au code.

Exécutez le programme pour démarrer le REST service web :

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

Abonnement d'un client sans connexion

L'utilisateur s'abonne aux notifications : RESTservice endpoint est envoyé avec le sujet afin de s'abonner à. Dans ce cas, le sujet est all.

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

La réponse attendue est une réponse 201, avec les détails de l'abonnement dans le corps de la réponse :

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

Il est possible d'obtenir la liste des notifications auxquelles l'utilisateur est abonné avec une requête GET :

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

La réponse obtenue est la suivante :

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

Vérification des messages, entrées DEBUG, show log, Nom de fichier utilisé, résultats SQL

Notez dans la réponse qu'il existe deux abonnements : un pour all (**"push.topic": "all"**) et un pour l'inventaire (**"push.topic": "inventory"**).
Vous pouvez le confirmer par une requête à la base de données (notez que le type d'abonnement est 'sans connexion' et
que SUBSCRIPTIONID correspondent à la sortie du GET comme surligné en jaune) :

```
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

Si vous devez supprimer un abonnement non orienté connexion, vous pouvez envoyer un HTTP DELETE avec l'ID d'abonnement que vous souhaitez supprimer. Supposons que vous souhaitez supprimer **subscription-id 2985507860170167151**:

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

Si vous interrogez à nouveau la base de données, vous ne voyez que l'abonnement avec SUBSCRIPTIONID égal à 7969974728822328535.

Lorsqu'un changement de stock se produit, le client imprime les notifications (qui sont du même type que le connection-oriented notifications vues dans la section à propos de connected-oriented clients), suivie de la réponse de 2010 :

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

Conclusion

Dans ce document, les deux types de notifications basées sur l'API qui peuvent être configurées dans EPNM (connectionlesset connection-oriented) sont expliquées et les exemples des clients respectifs qui peuvent être utilisés comme base à des fins de simulation sont donnés.

Informations connexes

- https://www.cisco.com/c/dam/en/us/td/docs/net_mgmt/epn_manager/RESTConf/Cisco_Evolved_Programmable_Network_Manager_5_1_2_F
- Assistance et documentation techniques - Cisco Systems

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.