Solucionar problemas de notificações EPNM baseadas em API

Contents

Introduction

Este documento descreve como solucionar problemas de Notificações de EPNM quando a API REST é usada para acessar informações de falha do dispositivo.

Prerequisites

O cliente que você implementa deve ser capaz de manipular e assinar qualquer um dos dois mecanismos usados pelo Evolved Programmable Network Manager (EPNM) para enviar Notificações.

Requirements

Não existem requisitos específicos para este documento.

Componentes Utilizados

Este documento não se restringe a versões de software e hardware específicas.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. Se a rede estiver ativa, certifique-se de que você entenda o impacto potencial de qualquer comando.

Notificações de API do EPNM

As notificações alertam administradores e operadores de rede sobre eventos ou problemas importantes relacionados à rede. Essas notificações ajudam a garantir que possíveis problemas sejam detectados e

resolvidos rapidamente, o que reduz o tempo de inatividade e melhora o desempenho geral da rede.

O EPNM pode lidar com diferentes métodos, como notificações via e-mail, traps do protocolo de gerenciamento de rede simples (SNMP - Simple Network Management Protocol) para receptores especificados ou mensagens de Syslog para servidores Syslog externos. Além desses métodos, o EPNM também fornece uma interface de programação de aplicativos de transferência de estado representacional (REST API) que pode ser usada para recuperar informações sobre inventário, alarmes, ativação de serviço, execução de modelo e alta disponibilidade.

Notificações baseadas em API são suportadas atualmente com o uso de dois mecanismos diferentes:

- **Notificações orientadas a conexão:**O cliente assina uma URL predefinida e usa um cliente WebSocket com autenticação básica através de um canal HTTPS seguro.
- **Notificações sem conexão:** espera-se que o usuário tenha um serviço Web REST que seja capaz de aceitar payloads de linguagem de marcação extensível (XML) e/ou JSON (JavaScript Object Notation) como uma solicitação POST.

Todas as notificações compartilham o mesmo esquema e podem ser recuperadas nos formatos JSON ou XML.

Configuração básica do EPNM

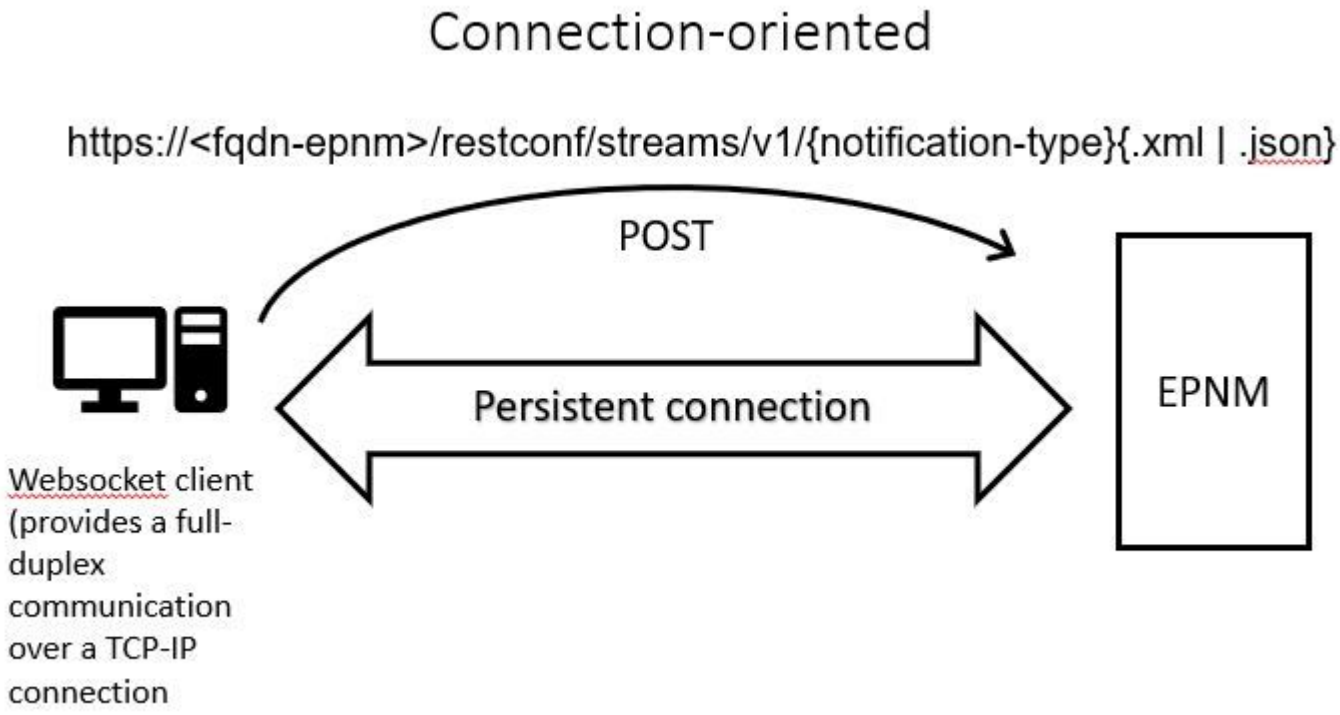
Por padrão, as notificações de alarme e inventário são desativadas. Para ativá-los, altere o restconf-config.properties como indicado (não é necessário reiniciar o aplicativo EPNM):

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

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

Notificações orientadas a conexão

Na figura, a máquina cliente executa um WebSocket e se inscreve no EPNM com um URL predefinido, com autenticação básica e através de um canal HTTPS seguro.



Executar um cliente WebSockets Python

A biblioteca cliente WebSocket em Python pode ser usada para criar um WebSocket na máquina 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})
```

Assinatura de um cliente orientado a conexão

Este código configura um cliente WebSocket que assina o EPNM em **wss://10.122.28.3/restconf/streams/v1/inventory.json**. Ele usa o Python WebSocketpara estabelecer a conexão e manipular mensagens de entrada e saída. A assinatura também pode ser (com base no tipo de notificação que você deseja assinar):

/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}

O on_message, on_error e on_close funções são funções de retorno de chamada que são chamadas quando a conexão WebSocket recebe uma mensagem, encontra um erro ou é fechada, respectivamente. O **on_open** é um retorno de chamada que é chamado quando a conexão WebSocket é estabelecida e está pronta para uso.

O username e password as variáveis são definidas para as credenciais de login necessárias para acessar o servidor remoto. Essas credenciais são codificadas com o comando base64 e adicionado aos cabeçalhos da solicitação WebSocket.

O run_forever é chamado no objeto WebSocket para iniciar a conexão, mantê-la aberta indefinidamente e ouvir as mensagens que vêm do servidor. O sslopt é usado para configurar as opções SSL/TLS para a conexão. O CERT_NONE flag desabilita a validação de certificação.

Execute o código para ter o WebSocket pronto para receber as notificações:

```
(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: shY1K9SqXTphBqaZFh/iMQ==
Sec-WebSocket-Version: 13
Connection: Upgrade
Authorization: Basic cm9vdDpQYXNzMTIzNA==


-----------------------
--- 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
```

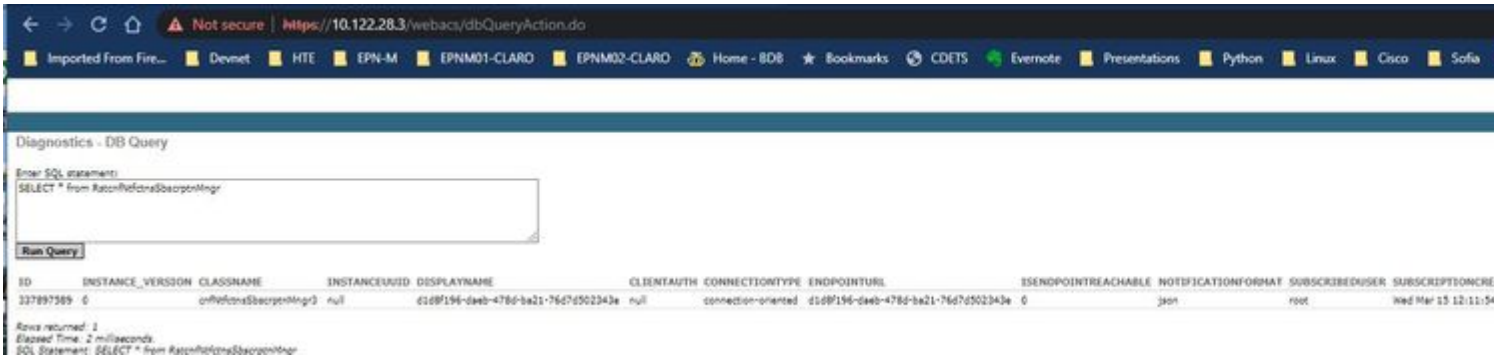
```
-----------------------
Websocket connected
++Sent raw: b'\x81\x8es\x99ry;\xfc\x1e\x15\x1c\xb5R*\x16\xeb\x04\x1c\x01\xb8'
++Sent decoded: fin=1 opcode=1 data=b'Hello, Server!'
++Rcv raw: b'\x81\x0eHello, Server!'
++Rcv decoded: fin=1 opcode=1 data=b'Hello, Server!'
Hello, Server!
```

Você pode verificar as assinaturas de notificação para o servidor com esta consulta DB (navegue até https://

/webacs/dbQueryAction.do

).

```
SELECT * from RstcnfNtfctnsSbscrptnMngr;
```



A partir da documentação on-line do EPNM, uma vez estabelecida, a mesma conexão é mantida ativa durante todo o ciclo de vida do aplicativo:

- até o cliente se desconectar do servidor
- até que o servidor fique inativo para manutenção ou durante um failover

Se, por algum motivo, for necessário excluir uma assinatura específica, você poderá enviar uma HTTP DELETE com a SUBSCRIPTIONID especificado na URL https://

. Por exemplo:

```
devasc@labvm:~/epnm$ curl --location --insecure --request DELETE 'https://10.122.28.3/restconf/data/v1/c
> --header 'Accept: application/json' \
> --header 'Content-Type: application-json' \
> --header 'Authorization: Basic cm9vdDpQYXNzMTIzNA==' \
> --header 'Cookie: JSESSIONID=2CB4F43E3D4BCE5F42411114065F6292'
```

Verificação de mensagens, entradas de DEPURAÇÃO, show log**, Nome de arquivo usado, Saídas SQL**

Para solucionar o motivo pelo qual um cliente que usa um mecanismo orientado a conexão não recebe notificações corretamente, você pode

executar a consulta DB indicada e verificar se a assinatura está presente ou não. Se ela não estiver presente, peça ao proprietário do cliente para certificar-se de emitir a assinatura.

Enquanto isso, você pode ativar o nível de DEPURAÇÃO em com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter para que você possa capturá-la sempre que a assinatura for enviada:

```
[root@epnm-spo-lab-host SCRIPTS]# sudo /opt/CSCOlumos/bin/setLogLevel.sh com.cisco.nms.nbi.epnm.restconf
Loglevel set to DEBUG for com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdap
```

Após o envio da assinatura, você poderá verificar se uma entrada com o endereço IP do cliente WebSocket aparece na localhost_access_log.txt:

```
[root@epnm-spo-lab-host SCRIPTS]# zgrep -h '"GET /restconf/streams/.* HTTP/1.1" 101' $(ls -1t /opt/CSCO
```

```
10.134.4.35 - - [30/Mar/2023:15:33:43 -0300] "GET /restconf/streams/v1/all.json HTTP/1.1" 101 -
```

Finalmente, verifique novamente o DB (observe que o timestamp corresponde à entrada em localhost_access_log.txt.

Diagnostics - DB Query

Enter SQL statement:

```
SELECT * from RstcnfNtfctnsSbscrptnMngr;
```

Run Query

ID	INSTANCE_VERSION	CLASSNAME	INSTANCEUUID	DISPLAYNAME	CLIENTAUTH	CONNECTIONTYPE	ENDPOINTURL	ISENDPOINTREACHABLE	NOTIFICATIONFORMAT	SUBSCRIBEDUSER	SUBSCR
337897623	0	cnfNtfctnsSbscrptnMngr3	null	cd90fa14-9d5c-4847-b180-539767c77fee	null	connection-oriented	cd90fa14-9d5c-4847-b180-539767c77fee	0	json	root	Thu Mar

Rows returned: 1
Elapsed Time: 2 milliseconds.
*SQL Statement: SELECT * from RstcnfNtfctnsSbscrptnMngr*

O próximo registro mostra quando as solicitações POST para assinaturas são enviadas:

```
[root@epnm-spo-lab-host SCRIPTS]# grep -Eh 'DEBUG com.cisco.nms.nbi.epnm.restconf.notifications.handler.
```

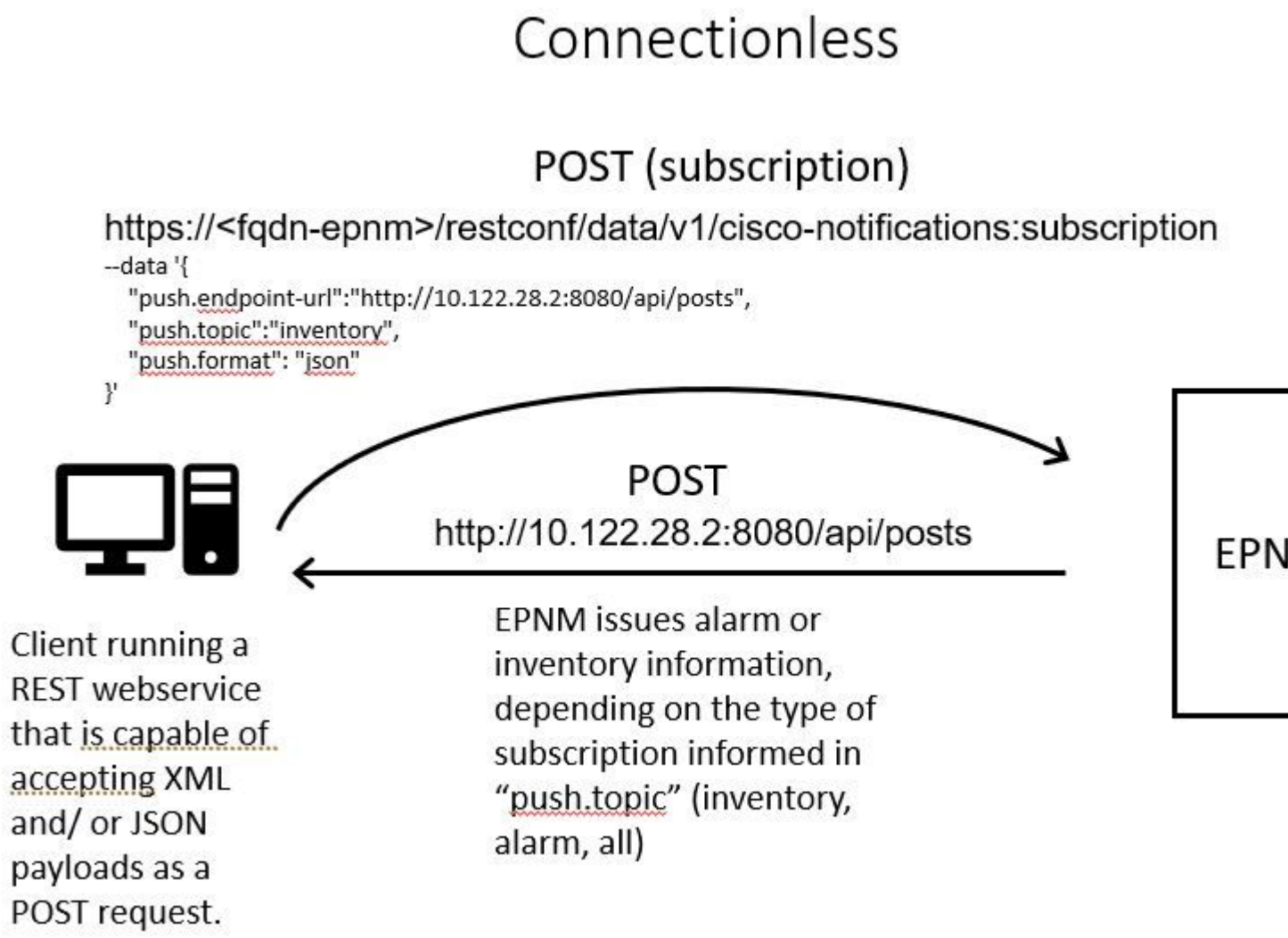
```
2023-03-30 15:33:43,399: DEBUG com.cisco.nms.nbi.epnm.restconf.notifications.handler.Notificatior
```

Enquanto a conexão for mantida ativa, uma notificação do tipo push-change-update será enviada do servidor EPN-M para todos os clientes que assinaram as notificações. O exemplo mostra uma das notificações enviadas pelo EPNM quando o nome de host de um NCS2k é alterado:


```
{
  "push.push-change-update":{
    "push.notification-id":2052931975556780123,
    "push.topic":"inventory",
    "push.time-of-update":"2023-03-31 13:50:36.608",
    "push.time-of-update-iso8601":"2023-03-31T13:50:39.681-03:00",
    "push.operation":"push:modify",
    "push.update-data":{
      "nd.node":{
        "nd.description":"SOFTWARE=ONS,IPADDR=10.10.1.222,IPMASK=255.255.255.0,DEFRTR=255.255.255.255,IP
        "nd.equipment-list":"",
        "nd.fdn":"MD=CISCO_EPNM!ND=tcc222c",
        "nd.sys-up-time":"217 days, 14:40:170.00"
      }
    }
  }
}
```


Notificações sem conexão

O próximo é o fluxo de trabalho no caso de connectionless notificações:



Executar um cliente Python de serviço Web REST

Espera-se que o usuário tenha um serviço Web REST que seja capaz de aceitar payloads XML e/ou JSON como uma solicitação POST. Esse serviço REST é o ponto final para o qual o EPNM da Ciscoa estrutura de notificações restconf publica notificações. Este é um exemplo de um serviço Web REST a ser instalado na máquina remota:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@ app.route('/api/posts', methods=['POST'])
def create_post():
    post_data = request.get_json()
    response = {'message': 'Post created successfully'}
    print(post_data)
    return jsonify(response), 201

if __name__ == '__main__':
    app.run(debug=True, host='10.122.28.2', port=8080)
```

Esta é uma aplicação Web Python Flask que define um único ponto final /api/posts que aceita **HTTP POST** solicitações. O create_post() função é chamada sempre que um **HTTP POST** pedido é feito a /api/posts. Dentro do create_post() função, os dados da solicitação recebidos são recuperados com o uso de **request.get_json()**, que retorna um dicionário do payload JSON. A carga útil é então impressa com **print(post_data)** para depuração. Depois disso, uma mensagem de resposta é criada com a tecla message e valor **Post created successfully** (em formato de dicionário). Essa mensagem de resposta é retornada ao cliente com um código de status HTTP 201 (criado).

O if __name__ == '__main__': block é uma construção padrão do Python que verifica se o script é executado como o programa principal, em vez de importado como um módulo. Se o script for executado como o programa principal, ele iniciará o aplicativo Flask e o executará no endereço IP e na porta especificados. O **debug=True** O argumento ativa o modo de depuração, que fornece mensagens de erro detalhadas e o recarregamento automático do servidor quando são feitas alterações no código.

Execute o programa para iniciar o REST serviço Web:

```
(venv) [apinelli@centos8_cxlabs_spo app]$ python connectionless.py
 * Serving Flask app 'connectionless' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://10.122.28.2:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 117-025-064
```

Assinatura de um cliente sem conexão

O usuário assina as notificações: o RESTo ponto final de serviço é enviado junto com o tópico no qual se inscrever. Neste caso, o tópico é all.

```
curl --location -X POST --insecure 'https://10.122.28.3/restconf/data/v1/cisco-notifications:subscriptio
--header 'Accept: application/json' \
--header 'Content-Type: application-json' \
--header 'Authorization: Basic cm9vdDpQYXNzMTIzNA==' \
--data '{
    "push.endpoint-url":"http://10.122.28.2:8080/api/posts",
    "push.topic":"all",
    "push.format": "json"
}'
```

A resposta esperada é uma resposta de 201, junto com os detalhes da assinatura no corpo da resposta:

```
{
   "push.notification-subscription":{
      "push.subscription-id":6243853653106271664,
      "push.subscribed-user":"root",
      "push.endpoint-url":"http:\/\/10.122.28.2:8080\/api\/posts",
      "push.topic":"all",
      "push.creation-time":"Fri Mar 31 17:07:48 BRT 2023",
      "push.creation-time-iso8601":"2023-03-31T17:07:48.159-03:00",
      "push.time-of-update":"Fri Mar 31 17:07:48 BRT 2023",
      "push.time-of-update-iso8601":"2023-03-31T17:07:48.159-03:00",
      "push.format":"json",
      "push.connection-type":"connection-less"
   }
}
```


É possível obter a lista de notificações em que o usuário está inscrito com uma solicitação GET:

```
curl --location --insecure 'https://10.122.28.3/restconf/data/v1/cisco-notifications:subscription' \
--header 'Accept: application/json' \
--header 'Content-Type: application-json' \
--header 'Authorization: Basic cm9vdDpQYXNzMTIzNA=='
```

A resposta obtida é:


```
{
   "com.response-message":{
      "com.header":{
         "com.firstIndex":0,
         "com.lastIndex":0
      },
      "com.data":{
         "push.notification-subscription":{
            "push.subscription-id":6243853653106271664,
            "push.subscribed-user":"root",
            "push.endpoint-url":"http:\/\/10.122.28.2:8080\/api\/posts",
            "push.session-id":0,
            "push.topic":"all",
            "push.creation-time":"Fri Mar 31 17:07:48 BRT 2023",
            "push.time-of-update":"Fri Mar 31 17:07:48 BRT 2023",
            "push.format":"json",
            "push.connection-type":"connection-less"
         }
      }
   }
}
#2
```

Verificação de mensagens, entradas de DEPURAÇÃO, show log, **Nome de arquivo usado, Saídas SQL**

Observe na resposta que há uma assinatura para all **("push.topic": "all")**. Você pode confirmá-la com uma consulta ao banco de dados (observe que o tipo de assinatura é 'sem conexão' e o SUBSCRIPTIONID corresponde à saída do comando GET como destacado em amarelo):

Se precisar excluir uma assinatura sem conexão, você poderá enviar uma solicitação HTTP DELETE, com a ID da assinatura que deseja excluir. Suponha que você queira excluir **subscription-id** 6243853653106271664:

```
curl --location --insecure --request DELETE 'https://10.122.28.3/restconf/data/v1/cisco-notifications:su
--header 'Accept: application/json' \
--header 'Content-Type: application-json' \
--header 'Authorization: Basic cm9vdDpQYXNzMTIzNA=='
```

Agora, se você consultar o BD novamente, não verá nenhuma entrada:

Quando ocorre uma alteração no inventário, o cliente imprime as notificações (que são do mesmo tipo que o connection-oriented as notificações vistas na seção sobre connected-oriented clientes), seguido da resposta de 201:

```
(venv) [apinelli@centos8_cxlabs_spo app]$ python connectionless.py
 * Serving Flask app 'connectionless' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://10.122.28.2:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 117-025-064
{'push.push-change-update': {'push.notification-id': -2185938612268228828, 'push.topic': 'inventory', 'p
10.122.28.3 - - [31/Mar/2023 16:47:23] "POST /api/posts HTTP/1.1" 201 -
{'push.push-change-update': {'push.notification-id': -1634959052215805274, 'push.topic': 'inventory', 'p
10.122.28.3 - - [31/Mar/2023 16:47:27] "POST /api/posts HTTP/1.1" 201 -
```

Conclusão

Neste documento, os dois tipos de notificações baseadas em API que podem ser configurados no EPNM (connectionlesse connection-oriented) são explicados e são dados exemplos dos respectivos clientes que podem ser utilizados como base para efeitos de simulação.

Referências

- https://<fqdn-epnm>/nbi_help/component.html?comp_id=Notification%20Subscriptions%20Retrieval&api=restconf
- https://www.cisco.com/c/dam/en/us/td/docs/net_mgmt/epn_manager/RESTConf/Cisco_EPN_Manager_RESTConf_NBI_Guide_5_1_2.zip
- https://www.cisco.com/c/dam/en/us/td/docs/net_mgmt/epn_manager/RESTConf/Cisco_Evolved_Programmable_Network_Manager_5_1_2_REST
- Suporte Técnico e Documentação - Cisco Systems

Sobre esta tradução
A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês (link fornecido) seja sempre consultado.