Solucionar problemas de notificaciones EPNM basadas en API

Contenido

Introducción

Este documento describe cómo resolver problemas de notificaciones EPNM cuando se utiliza la API REST para acceder a la información de fallas del dispositivo.

Prerequisites

El cliente que implemente debe ser capaz de administrar y suscribirse a cualquiera de los dos mecanismos que utiliza Evolved Programmable Network Manager (EPNM) para enviar notificaciones.

Requirements

No hay requisitos específicos para este documento.

Componentes Utilizados

Este documento no tiene restricciones específicas en cuanto a versiones de software y de hardware.

La información que contiene este documento se creó a partir de los dispositivos en un ambiente de laboratorio específico. Todos los dispositivos que se utilizan en este documento se pusieron en funcionamiento con una configuración verificada (predeterminada). Si tiene una red en vivo, asegúrese de entender el posible impacto de cualquier comando.

Notificaciones de API EPNM

Las notificaciones alertan a los administradores y operadores de red sobre eventos o problemas importantes relacionados con la red. Estas notificaciones ayudan a garantizar que los posibles problemas se detectan y resuelven rápidamente, lo que reduce el tiempo de inactividad y mejora el rendimiento general de la red.

EPNM puede administrar diferentes métodos, como notificaciones por correo electrónico, capturas del protocolo simple de administración de red (SNMP) a receptores especificados o mensajes de Syslog a servidores Syslog externos. Además de estos métodos, EPNM también proporciona una interfaz de programación de aplicaciones de transferencia de estado representacional (API REST) que se puede utilizar para recuperar información sobre el inventario, alarmas, activación de servicios, ejecución de plantillas y alta disponibilidad.

Actualmente, las notificaciones basadas en API son compatibles con el uso de dos mecanismos diferentes:

- **Notificaciones orientadas a la conexión:**El cliente se suscribe a una URL predefinida y utiliza un cliente WebSocket con autenticación básica a través de un canal HTTPS seguro.
- **Notificaciones sin conexión:** se espera que el usuario tenga un servicio web REST que sea capaz de aceptar cargas útiles de lenguaje de marcado extensible (XML) y/o JavaScript Object Notation (JSON) como una solicitud POST.

Todas las notificaciones comparten el mismo esquema y se pueden recuperar en formatos JSON o XML.

Configuración básica de EPNM

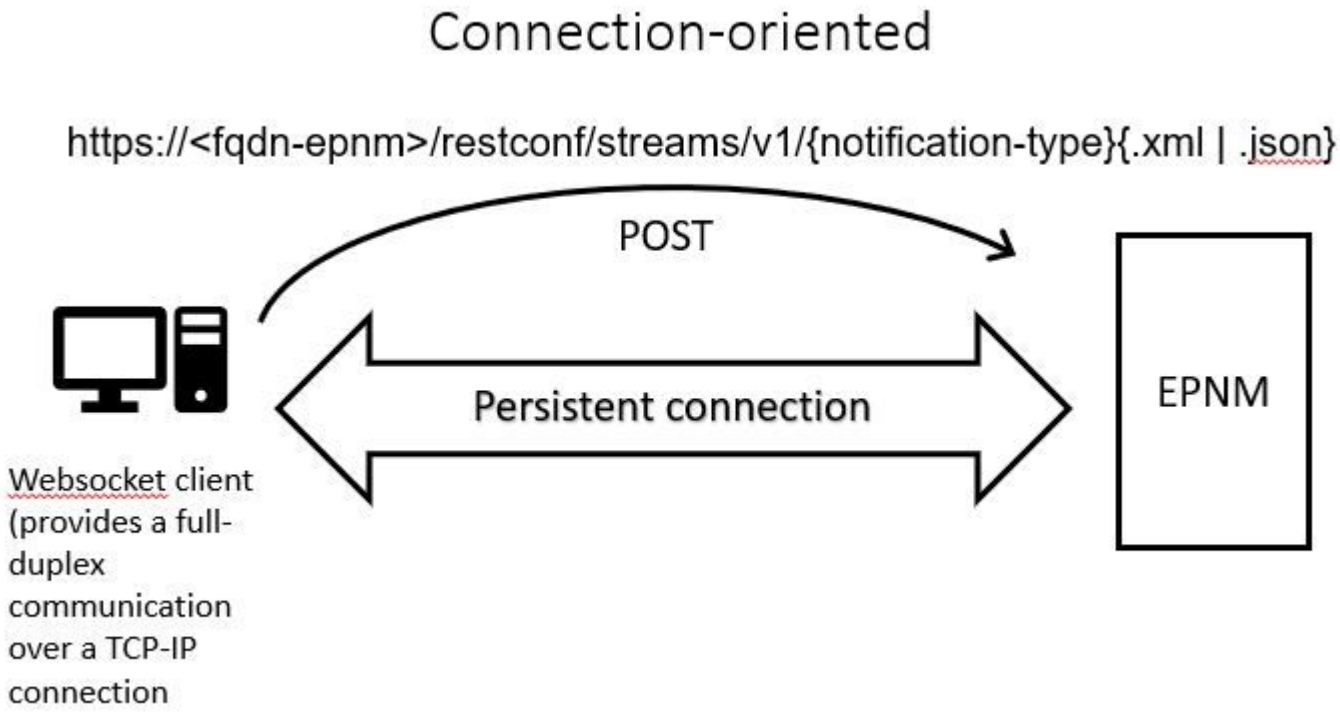
De forma predeterminada, las notificaciones de alarma e inventario están desactivadas. Para habilitarlos, cambie el restconf-config.properties como se indica (no es necesario reiniciar la aplicación EPNM):

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

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

Notificaciones orientadas a la conexión

En la imagen, el equipo cliente ejecuta un WebSocket y se suscribe al EPNM con una URL predefinida, con autenticación básica y a través de un canal HTTPS seguro.



Ejecutar un cliente de WebSockets Python

La biblioteca de cliente WebSocket en Python se puede utilizar para crear un WebSocket en el equipo 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})
```

Suscripción de un cliente orientado a la conexión

Este código configura un cliente WebSocket que se suscribe a EPNM en **wss://10.122.28.3/restconf/streams/v1/inventory.json**. Utiliza el Python WebSocketpara establecer la conexión y gestionar los mensajes de entrada y salida. La suscripción también puede ser (en función del tipo de notificación a la que desea suscribirse):

/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 y on_close Las funciones son funciones de devolución de llamada a las que se llama cuando la conexión WebSocket recibe un mensaje, encuentra un error o está cerrada, respectivamente.
on_open función es una devolución de llamada que se llama cuando la conexión WebSocket está establecida y lista para utilizarse.

username y password se establecen en las credenciales de inicio de sesión necesarias para acceder al servidor remoto. Estas credenciales se codifican con el base64 y se agrega a los encabezados de la solicitud WebSocket.

run_forever se llama a este método en el objeto WebSocket para iniciar la conexión y mantenerla abierta indefinidamente, y escucha los mensajes que provienen del servidor. sslopt se utiliza para configurar las opciones SSL/TLS para la conexión. CERT_NONE marca deshabilita la validación de certificación.

Ejecute el código para que WebSocket esté listo para recibir las notificaciones:

```
(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=
```

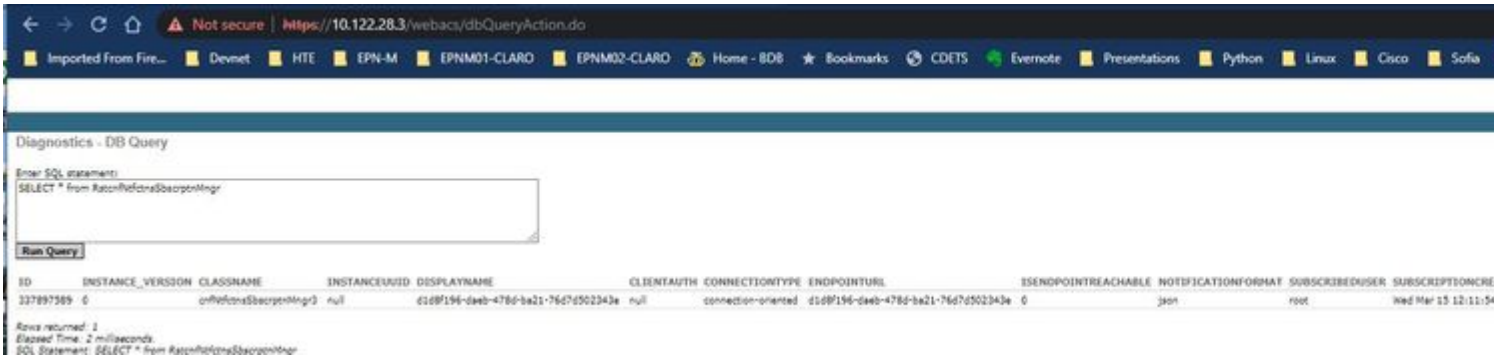
```
Date: Thu, 30 Mar 2023 16:18:19 GMT
Server: Prime
-----------------------
Websocket connected
++Sent raw: b'\x81\x8es\x99ry;\xfc\x1e\x15\x1c\xb5R*\x16\xeb\x04\x1c\x01\xb8'
++Sent decoded: fin=1 opcode=1 data=b'Hello, Server!'
++Rcv raw: b'\x81\x0eHello, Server!'
++Rcv decoded: fin=1 opcode=1 data=b'Hello, Server!'
Hello, Server!
```

Puede comprobar las suscripciones de notificación al servidor con esta consulta de base de datos (navegue hasta https://

/webacs/dbQueryAction.do

).

```
SELECT * from RstcnfNtfctnsSbscrptnMngr;
```



A partir de la documentación en línea de EPNM, una vez establecida, la misma conexión se mantiene activa durante todo el ciclo de vida de la aplicación:

- hasta que el cliente se desconecte del servidor
- hasta que el servidor deje de funcionar por motivos de mantenimiento o durante una recuperación ante fallos

Si, por alguna razón, necesita eliminar una suscripción específica, puede enviar una HTTP DELETE solicitud con el SUBSCRIPTIONID especificado en la URL https://

. Por ejemplo:

```
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'
```

Verificación de mensajes, entradas de DEBUG, show log**, Nombre de archivo utilizado, Salidas de SQL**

Para resolver problemas por qué un cliente que utiliza un mecanismo orientado a la conexión no recibe notificaciones correctamente, puede ejecutar la consulta de base de datos indicada y verificar si la suscripción está presente o no. Si no está presente, pida al propietario del cliente que se asegure de emitir la suscripción.

Mientras tanto, puede habilitar el nivel DEBUG en com.cisco.nms.nbi.epnm.restconf.notifications.handler.NotificationsHandlerAdapter para poder captarla siempre que se envíe la suscripción:

```
[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
```

Después de enviar la suscripción, puede comprobar si aparece una entrada con la dirección IP del cliente WebSocket en localhost_access_log.txt:

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

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

Finalmente, verifique nuevamente la BD (observe que la marca de tiempo coincide con la entrada en localhost_access_log.txt.

El siguiente registro muestra cuándo se envían las solicitudes POST de suscripciones:

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

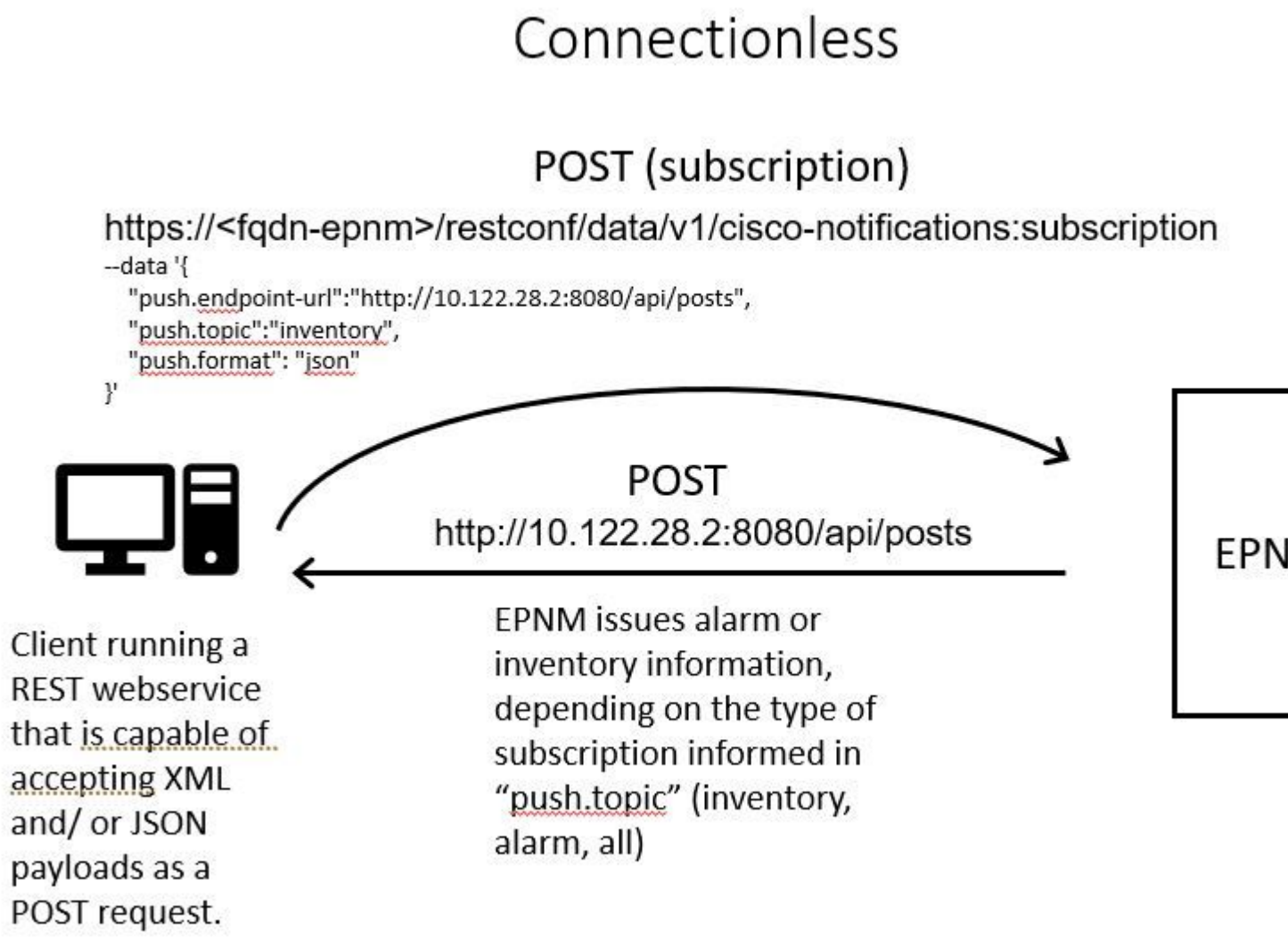
```
2023-03-30 15:33:43,399: DEBUG com.cisco.nms.nbi.epnm.restconf.notifications.handler.Notificatior
```

Mientras la conexión se mantenga activa, se envía una notificación de tipo push-change-update desde el servidor EPN-M a todos los clientes que se suscribieron para recibir notificaciones. El ejemplo muestra una de las notificaciones que envía el EPNM cuando se cambia el nombre de host de un NCS2k:


```
{
  "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"
      }
    }
  }
}
```


Notificaciones sin conexión

El siguiente es el flujo de trabajo en el caso de connectionless notificaciones:



Ejecutar un cliente de Python del servicio web REST

Se espera que el usuario tenga un servicio web REST que sea capaz de aceptar cargas XML y/o JSON como una solicitud POST. Este servicio REST es el terminal al que se conecta el Cisco EPNMel marco de notificaciones restconf publica notificaciones. Se trata de un ejemplo de un servicio web REST que se va a instalar en el equipo remoto:

```
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)
```

Se trata de una aplicación web de Python Flask que define un único terminal /api/posts que acepta **HTTP POST** solicitudes. create_post() función se llama siempre que un **HTTP POST** la solicitud se realiza a /api/posts. Dentro de la create_post() función, los datos de la solicitud que entra se recuperan con el uso de **request.get_json()**, que devuelve un diccionario de la carga de JSON. La carga útil se imprime con **print(post_data)** con fines de depuración. Después de esto, se crea un mensaje de respuesta con la clave message y valor **Post created successfully** (en formato de diccionario). Este mensaje de respuesta se devuelve al cliente con un código de estado HTTP 201 (creado).

if __name__ == '__main__': es una construcción estándar de Python que verifica si el script se ejecuta como el programa principal, en lugar de importarse como un módulo. Si la secuencia de comandos se ejecuta como programa principal, inicia la aplicación Flask y la ejecuta en la dirección IP y el puerto especificados. **debug=True** habilita el modo de depuración, que proporciona mensajes de error detallados y recarga automática del servidor cuando se realizan cambios en el código.

Ejecute el programa para iniciar el REST servicio 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
```

Suscripción de un cliente sin conexión

El usuario se suscribe a las notificaciones: RESTse envía el extremo del servicio junto con el tema al que se suscribe. En este caso, el tema es 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"
}'
```

La respuesta esperada es una respuesta 201, junto con los detalles de la suscripción en el cuerpo de la respuesta:

```
{
   "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"
   }
}
```


Es posible obtener la lista de notificaciones a las que está suscrito el usuario con una solicitud 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=='
```

La respuesta obtenida fue la siguiente:


```
{
   "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
```

Verificación de mensajes, entradas de DEBUG, show log, **Nombre de archivo utilizado, salidas de SQL**

Aviso de la respuesta de que hay una suscripción para all **("push.topic": "all")**. Puede confirmarlo con una consulta a la base de datos (observe que el tipo de suscripción es 'sin conexión' y el SUBSCRIPTIONID coincide con el resultado del GET (resaltado en amarillo):

Si necesita eliminar una suscripción sin conexión, puede enviar una solicitud HTTP DELETE con la ID de suscripción que desea eliminar. Suponga que desea eliminar el **id de suscripción** 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=='
```

Ahora, si vuelve a consultar la base de datos, no verá entradas:

Cuando se produce un cambio en el inventario, el cliente imprime las notificaciones (que son del mismo tipo que el connection-oriented las notificaciones que aparecen en la sección acerca de connected-oriented clientes), seguido de la respuesta de 2011:

```
(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 -
```

Conclusión

En este documento, se describen los dos tipos de notificaciones basadas en API que se pueden configurar en EPNM (connectionlessy connection-oriented) y se proporcionan ejemplos de los clientes respectivos que pueden utilizarse como base para fines de simulación.

Referencias

- https://<fqdn-epnm>/nbi_help/component.html?comp_id=Notification%20Suscripciones%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
- Soporte Técnico y Documentación - Cisco Systems

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).