MK Livestatus. Acceso a datos de Nagios mediante la API para Python.

Introducción.

MK Livestatus nos proporciona una API estándar de acceso a los datos de Nagios en varios lenguajes de programación: Python, Perl y C++.

La documentación de uso disponible para la API en dichos lenguajes viene en forma de ejemplos acompañando a los respectivos fuentes. Suficiente para empezar a probarlo. Debemos conocer previamente en que consiste MK Livestatus y su lenguaje de acceso a datos LQL.

La ruta de la documentación y los fuentes / librerías normalmente será, en una instalación de **OMD** en /omd/versions/default/share/doc/check_mk/livestatus/api/. Si tenemos una **instalación directa de check_mk** la ruta será por defecto la siguientes: /usr/share/check_mk/doc/livestatus/api/

API python de livestatus.

En el directorio de Python veremos varios archivos:

- livestatus.py — Módulo Python para acceso a Livestatus. API.
- example.py — Ejemplo de uso de la API para acceder a una instancia de Livestatus
- example_multisite.py — Ejemplo de uso de la API para acceder a varias instancias de Livestatus
- make_nagvis_map.py — Ejemplo para hacer un mapa de Nagvis.

El módulo necesario para importar en nuestro desarrollo en python por tanto será livestatus.py en la forma habitual en python:

```
import livestatus
```

Conexión a MK Livestatus.

Siguiendo el ejemplo principal "example.py" vamos a analizar la forma de conectarnos usando la API que proporciona. La propuesta es para una instalación de OMD como la comentada en el artículo de Introducción a OMD.

```
omd_root = os.getenv("OMD_ROOT")
socket_path = "unix:" + omd_root + "/tmp/run/live"
```

El ejemplo es para una instalación de OMD y debemos estar en una sesión de un site OMD (usuario que se crea para el site) aunque podemos probarlo también en una instalación sencilla de ML Livestatus. Si echamos un ojo a las variables especificas de un usuario de un site OMD vemos entre otras la variable OMD_ROOT.

```
>#  env | grep -i omd
..
OMD_ROOT=/omd/sites/jrm2
OMD_SITE=jrm2
…
```

Por tanto si no estamos en una instalación OMD o queremos ejecutar nuestros scripts Python desde otros usuarios solo tenemos que cambiar la ruta al fichero socket (si estamos en el propio servidor con MKL)

```
    socket_path = "unix:" + "/opt/omd/sites/jrm2/tmp/run/live"
```

Es posible que al acceder nos dé un error de permisos ya que el socket se crea solo con permisos para el usuario del site OMD concreto. Deberemos acceder con el mismo usuario o jugar con la integración de los usuarios en el mismo grupo de usuarios para poder acceder desde cualquiera que pertenezca a este.

Mejor aún, podemos usar un socket de red apuntando al servidor con MK Livestatus ya sea este el mismo (localhost) u otro. Es sencillo y eficiente. Debemos tener configurado el acceso al socket del servidor de Nagios p.e. mediante xinetd como se explica en el artículo "MK Livestatus. Instalación independiente en Nagios" p.e. (cualquier instalación de MK Livestatus integrada o no con OMD / Check_mk nos valdrá configurando adecuadamente el acceso por xinetd)

```
    socket_path = "tcp:" + "localhost:6557"
```

Una vez conectados solo es ir probando el script de ejemplo para ver como funcionan las consultas.

Obtención de datos.

En el artículo anterior "MK Livestatus. Acceso a datos de Nagios con "unixcat" y lenguaje LQL." vimos como obtener los datos necesarios de tablas y campos para usar en las consultas.
Vamos a ir revisando los ejemplos que nos proporcionan en el fichero python **example.py** y revisando las distintas opciones de acceso a datos que nos proporciona.

En dicho ejemplo vemos dos tipos de conexiones, en un caso realiza la conexión pasando la "query" de los datos a buscar en este y especificando el tipo de query (table, row_assoc,)

```
livestatus.SingleSiteConnection(socket_path).query_row_assoc("GET status").items():

hosts = livestatus.SingleSiteConnection(socket_path).query_table("GET hosts\nColumns:
name alias address")
```

El otro tipo de conexión es, primero creo el objeto conexión y luego lo uso para realizar las consultas.

```
conn = livestatus.SingleSiteConnection(socket_path)
num_up = conn.query_value("GET hosts\nStats: hard_state = 0")
```

Vemos que usa distintos tipo de métodos predefinidos para especificar que tipo de datos se quieren obtener y como será el formato de la salida de los datos. Todo ellos están definidos en la API en el fichero livestatus.py. (table, query_row_assoc, query_value, query_row, …)

Ejemplos de querys:
query_row_assoc Devuelve una línea de datos con un diccionario de datos nombre → valor
query_table devuelve una lista de listas representando cada fila de la tabla.
query_value devuelve un único valor (de una línea y una columna = una celda)
query_row devuelve una línea de datos con los elementos como una lista.
query_column devuelve una columna con todos los valores como una lista.

Todos los ejemplos son muy sencillos y las funciones usadas están suficientemente detalladas en el módulo livestatus.py. En el otro fichero de ejemplo "example_multisite.py" podemos ver como conectar simultaneamente con varios servidores Livestatus para ejecutar la misma consulta en todos y obtenerla agregada.

A continuación vamos a un ejemplo de uso de la API python.

Ejemplo. Plugin para verificar la conexión con un servidor MK Livestatus.

Vamos a ver un sencillo y practico ejemplo de uso de la API, un plugin para Nagios / CMK que podremos usar para asegurarnos que otros servidores con MK Livestatus están respondiendo adecuadamente. En una configuración de consola única con CMK Multisite es importante asegurarnos que los servidores con MK Livestatus respondan ya que nos podemos encontrar con un escenario en el que no vemos los errores de objetos Hosts / Servicios de otros servidores MK Livestatus porque no conectamos a ellos por cualquier razón. Aunque en la definición del Site remoto con MK Multisite puedes asociarle un host propio a chequear para validad la conexión remota, no lo asocia a una conexión Livestatus que sería lo más adecuado. Que mejor forma que un "plugin" clásico para avisarnos ante problemas de conexión con el socket o el acceso a los objetos.

Para usar el plugin y evitar errores mejor bajarlo en modo texto de este enlace. Se deja comentarios útiles de interés o para realizar pruebas de debug.
El plugin es muy sencillo y se entiende bien. Definimos las variables del servidor y puerto del MK Livestatus al que nos vamos a conectar y además un objeto host a buscar en este (definido en dicho Nagios). No solo nos aseguramos de conectar con ML Livestatus si no de que es capaz de retornarnos adecuadamente una busqueda de un objeto host concreto.

```
#!/usr/bin/python
# -*- coding: iso-8859-15 -*-
# OBJETIVO, Plugin Nagios para chequear estado de socket cmk livestatus
# El script se conecta por livestatus al servidor dado y busca un host (definición) concreto

import livestatus
import sys

cmk_livestatus_nagios_server = "localhost"
cmk_livestatus_tcp_port = 6557
host_to_find = "srv0001a"

try:
    # Creamos el socket
    #socket_path = "tcp:" + "localhost:6557"
    socket_path = "tcp:%s:%s" % (cmk_livestatus_nagios_server,cmk_livestatus_tcp_port)
except:
    sys.exit(1)

try:
    # Creamos la conexión con MK Livestatus usando el socket
    conn = livestatus.SingleSiteConnection(socket_path)
    # Consulta LQL usada para buscar un host
    # GET hosts
    # Columns: name
    # Filter: name = srv0001
    host = conn.query_value("GET hosts\nColumns: name\nFilter: name = %s" % host_to_find)
    print "OK - Host '%s' found in livestatus connect to '%s:%s'" \
          % (host_to_find, cmk_livestatus_nagios_server,cmk_livestatus_tcp_port)
    exit (0)
except Exception, e: # livestatus.MKLivestatusException, e:
   #print "Livestatus error: %s" % str(e)
   print "CRITICAL - Host '%s' not found or livestatus connect to '%s:%s'" \
          % (host_to_find, cmk_livestatus_nagios_server,cmk_livestatus_tcp_port)
   exit (2)
```

Usamos una conexión contra MK Livestatus para lanzar un query de tipo “valor” (función / clase “query_value”) para buscar unico host. Si lo encuentra continuará la ejecución y devolvemos la cadena de éxito maś el valor de retorno 0 (OK). Si no lo encuentra ejecutará la excepción y obtendremos una caden con el error y un valor de retono de 2 (Critical para Nagios)

Probando el plugin y el valor de retorno para Nagios (0 = OK, 2 = Critical)

```
$ python ./test_livestatus
OK - Host 'srv0001' found in livestatus connect to 'localhost:6557'
$ echo $?
0
# Cambiando nombre de host a uno inexistente
$ python ./test_livestatus
CRITICAL - Host 'srv0001a' not found or livestatus connect to 'localhost:6557'
$ echo $?
2
```

Idealmente habría que definir el paso de parámetros que estos no estuvieran definidos en el plugin y poder usarlo para chequear varios servidores. Para ver las posibilidades es suficiente :)

Otros posibles usos.

El potencial de la API es muy interesante. Se puede usar tanto para hacer nuevos plugins que usen datos existentes, para extraer datos con la finalidad de tratarlos con programas externos (estadísticas, slas,...), para otros programas que muestren online el estado de nuestros objetos nagios como ya hacen muchos programas (nagvis, nagstamon,...).

Un uso muy importante que le dí recientemente fue un plugin que obtenia datos de otros plugins existentes. Puede ser muy práctico para realizar chequeos que dependan de valores obtenidos de uno o más chequeos existentes o sencillamente que sumarizen / operen con valores obtenidos por otros plugins.
En artículos posteriores intentaremos profundizar con otros ejemplos.