Monitorización mediante snmpv3 usando un bot de Discord

Óscar Fernández Sáchez & Miguel Herráez Sánchez
GRS - GIC

Índice

Escenario

Hoy en dia si hay algo que destaque sobre el resto en el mundo de la informática es la inteligencia artificial y el minado de criptomonedas, ambos son trabajos en los que se requiere hardware específico trabajando al máximo durante largos periodos de tiempo, es común que este hardware acabe inutil con el paso del tiempo ya que no está preparado para trabajar con ese volumen de trabajo y al estar al máximo cualquier pequeño error puede quemar el chip.
Para grandes empresas o estudios existen múltiples herramientas de monitorización de código abierto que permiten tener un control bastante extenso de la red, sin embargo estas herramientas por lo general son complicadas de instalar y configurar alejándose del público sin grandes conocimientos de la informática. Además de requerir servidores web para acceder a ellos de manera remota.
Sin embargo es cada vez más común que gente sin conocimientos previos de informática intente minar criptomonedas o se interese por la inteligencia artificial. Por todo esto hemos decidido crear un sistema de monitorización con una instalación sencilla y sin problemas de compatibilidad entre sus componentes, esto lo va ha hacer más accesible además de usar una aplicación de comunicación tan conocida como Discord.

Ventajas e inconvenientes

Ventajas

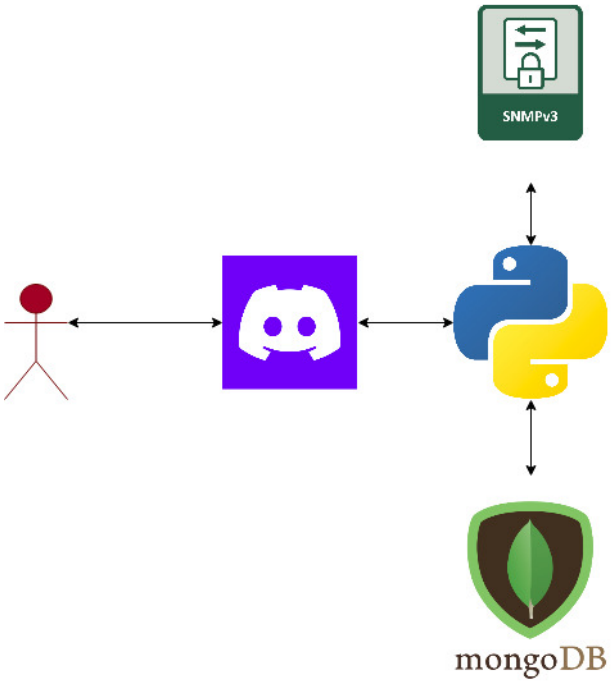
Como hemos expuesto anteriormente la principal ventaja de este servicio de monitorización es su sencillez, tanto de instalación como de uso, también implementa la monitorización de los parámetros de un ordenador de forma remota sin servidor web desde la aplicación de Discord y al usar Python como lenguaje de programación contamos con muchas librerías con las que aumentar las opciones de monitorización de este bot. Además hemos usado software de código abierto por lo que no tiene coste alguno. No consume muchos recursos computacionalmente hablando, y lo único que hay que hacer para comenzar a usarlo es arrancar un fichero de Python para establecer la conexión y la recogida de datos.

Inconvenientes

El principal inconveniente es la etapa de desarrollo en la que se encuentra, las herramientas de monitorización convencionales cuentan con muchas más herramientas, esto se podría solucionar implementando una mayor cantidad de comandos, aunque nunca llegando al nivel que pueden tener nagios por ejemplo.
Otro inconveniente es la dificultad de implementación en windows ya que SNMP no tiene tan facil uso en este SO, por lo que tendremos que instalarlo en una arquitectura Linux, sin embargo esto no supone un gran problema ya que los trabajos para los que hemos pensado usar este software funcionan la mayoría de las veces sobre Linux. Más allá de este inconveniente, que es lo que nos ha alejado de implementar el bot en Windows, otros componentes del proyecto también entran en conflicto con Windows, como puede ser la base de datos MongoDB, que no tiene una implementación tan natural en Windows como en Ubuntu. Si se quisiese usar en Windows habría que crear contenedores con Docker, Kubernetes o desplegar máquinas virtuales con Vagrant para acceder a una versión de Ubuntu en Windows.

Arquitectura

El sistema de monitorización estaría estructurado de la siguiente manera:



Componentes del sistema

SNMPv3

Hemos decidido usar SNMPv3 ya que es la última versión de SNMP y la única que implementa seguridad.

Python

Este lenguaje de programación es ampliamente usado en sistemas que no requieren de una gran velocidad pero sí de muchas funcionalidades, la utilización de este lenguaje nos ha permitido usar una gran cantidad de librerías que agilizan el desarrollo y proporcionan utilidades difícilmente alcanzables sin ellas, alguna de estas librerías son discord.py, pysnmp y numpy. Además python permite una integración rápida con el resto de componentes al disponer de estas librerías.

Mongodb

Mongodb es una base de datos no relacional, la hemos elegido ya que solo necesitamos un modelo para guardar los datos de memoria y cpu que vamos obteniendo, además cuenta con la librería python pymongo que nos permite integrar ambos componentes. Nos es útil ya que no hay que generar modelos antes de insertar los datos, permite borrado por TTL y manejarlo desde la consola con comandos fáciles.

Discord

Discord es una aplicación de comunicación ampliamente conocida y usada, esta aplicación permite un modo developer con el que desarrollar distintos bots y alarmas que trabajaran en el servidor de discord proporcionando funcionalidades adicionales, esto nos va a permitir a nosotros usar su api para implementar nuestra monitorización.

Cada uno de los componentes del sistema realiza las siguientes funciones:

- El usuario a través de **Discord** ejecuta un comando con palabra reservada, como por ejemplo **$sysName**. Para poder programar el bot es necesario tener el modo desarrollador activado en la aplicación.

- Usando la biblioteca discord.py, Discord conecta con el programa bot.py, que ejerce como middleware entre Discord y la parte más cercana a la máquina como mongoDb y snmpv3. Este archivo está dentro de un entorno de virtualización de python y ejecuta los comandos snmpv3 llamando al archivo snmp.py, que es quien hace las distintas llamadas a snmp. Para el uso de claves de snmpv3 usamos variables de entorno guardadas en un archivo .env, donde se guardan las dos claves(SHA y AES), nombre del usuario e información del bot para la integración con discord.

- Snmpv3 está instalado en un sistema operativo Ubuntu 20.04 LTS, y usa las mibs instaladas con el paquete downloader-mibs. Se han creado usuarios ad-hoc para las consultas snmp, con el objetivo proteger el sistema y no permitir al bot acceder a MIBS más allá de aquellas que estén incluidas en la comunidad public. Actualmente permite peticiones exclusivamente desde localhost.

- Con MongoDB guardamos los datos que nos proporciona snmp. Es una base de datos no relacional, uno de los motivos que nos han llevado a elegirla por encima de otras como MySql o Postgres. Hemos escogido esta base de datos en concreto porque es facil de implementar, programar y desplegar y permite borrar datos usando TTLs, ya que dependiendo el tiempo que establezcamos para recoger los datos, podemos sobrecargar la base de datos que métricas que ya no son útiles debido al momento que fueron recogidas.

Desarrollo

Instalación snmpv3 Ubuntu 20.04 LTS

Comprobaremos si snmp está instalado :

snmp –version

Si no está instalado procederemos:

1. sudo apt install snmp
2. sudo apt install snmpd snmp libsnmp-dev
3. Configurar snmp
    a. sudo net-snmp-create-v3-user [-ro] [-A authpass] [-a MD5|SHA] [-X privpass][-x DES|AES] [username]
    b. En nuestro caso: sudo net-snmp-create-v3-user -ro -A STrP@SSWRD -a SHA -X STr0ngP@SSWRD -x AES snmpadmin
4. sudo systemctl start snmpd
5. sudo systemctl enable snmpd
6. Para verificar: sudo systemctl status snmpd

Instalación python

En muchos sistemas Ubuntu, Python3 viene preinstalado, así que para ver la versión:

```
$ python --version
```

Si no está instalado, habría que instalarlo siguiendo los siguientes pasos:

1. sudo apt install software-properties-common
2. sudo add-apt-repository ppa:deadsnakes/ppa
3. sudo apt install python3.8
4. python --version

Instalacion MongoDB

1. `wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key add -`
2. `echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list`
3. `sudo apt-get update`
4. `sudo apt-get install -y mongodb-org`
5. `sudo systemctl start mongod`
6. `sudo systemctl enable mongod`
7. `mongo`

Tras esto, se podrá acceder a mongodb a través de la consola. Configurar una base de datos a través de este gestor se puede hacer a través de comandos sencillos, y no hay que crear un modelo previo a la inserción de los datos, lo que demuestra su flexibilidad
El modelo de datos que estamos guardando es el siguiente:

{_id:<String>, mem:<Number>, cpu:<Number>, createdAt:<Date>}

Usamos createdAt como TTL e indicamos a la base de datos que tomando ese timestamp como referencia, borre los datos creados después de 10 minutos. De esta forma no sobrecargamos la base de datos. Este valor es variable desde dentro de la base de datos.

Integración de Componentes

Para integrar todos los componentes y comenzar a usar el bot, está disponible el entorno de programación en el siguiente enlace a nuestro repositorio de Github. Cuando se descargue o se haga git clone, ejecutar el siguiente comando para entrar en el entorno

- `source grs_bot/bin/activate`

Ya dentro del entorno, instalar las dependencias del proyecto

```
1. pip install -u discord.py
2. pip install -u pysnmp
3. pip install -u datetime
4. pip install -u threading6
5. pip install -u time
```

Como paso final antes de utilizar el bot, hay que crear un archivo de variables de entorno donde guardaremos nuestras claves de SNMP y la informacion de nuestro bot, tal que así: (FOTO)

Ya se podria ejecutar el bot con el comando **python3 bot.py**

Echando un ojo al código

Si nos metemos a fondo en el código que ejecuta python, podemos ver que nuestro proyecto se basa en dos archivos, snmp.py y bot.py.

Snmp.py: Archivo el cual realiza peticiones snmp a traves del puerto udp del ordenador, por ahora todo en localhost. Se pueden realizar peticiones snmp get y snmpgetBulk a distintas mibs definidas en el archivo.

Bot.py: Archivo main que realiza las llamadas a snmp.py, y que de forma paralela corre un hilo que se encarga de guardar las métricas en una base de datos de MongoDB. Hace espera activa durante x tiempo(se puede variar en el código) y envia un mapa con los datos de memoria ram libre, cpu y un timestamp en formato de fecha del momento en el que se envia. Discord espera los siguientes comandos para ejecutar snmp:

- `$snmpget`
  - `sysName`
  - `sysLocation`
  - `sysDescr`
  - `ramFree`
  - `freeDiskSpace`
  - `cpuLoad`
- `$snmpGetBulk`
  - `memory`
  - `disk`
  - `cpuTimes`
  - `network`
- `$GR`
  - `Memory`
  - `CPU`

Adicionalmente, a $snmpGetBulk hay que pasarle por parametro el numero de metricas que queremos obtener, por ejemplo, **$snmpGetBulk disk 7**

Posibles mejoras

Este proyecto a fecha que se realiza esta memoria se corre de forma local en un ordenador Ubuntu, pero gracias a la flexibilidad de las herramientas utilizadas sería posible escalarlo para que, por ejemplo, una base de datos centralizada desplegada en MongoDB Cloud recogiese los datos y fuesen tratados en un ordenador central con Nagios y Grafana instalados para interpretar y tratar estos datos. Un ejemplo, la métrica **mem** de nuestra base de datos muestra que durante un periodo de una hora ha alcanzado valores mínimos en uno de los hosts que utilizan el bot. Se podría alertar al administrador del equipo utilizando Nagios e informarle a través de las gráficas que nos ofrece Grafana. Este “ordenador central” podría tratarse de una máquina virtual desplegada en algún servicio cloud como AWS o Kubernetes.

También se podría automatizar el arranque del bot, ya que como hemos descrito en el apartado de **Desarrollo**, actualmente se tiene que arrancar a través de un comando de python. Se podría intentar hacer ver al sistema operativo que se trata de un servicio e indicar que se arranque cuando el ordenador encienda con sudo systemctl enable.

Por supuesto se podrían añadir más llamadas snmp para el bot, incluso cambiar el rol del administrador que accede para que también tenga permiso de escritura(manteniendo protegida la parte más sensible de la información del ordenador).

Anexo

Código

bot.py

```
import os

import discord

from dotenv import load_dotenv

from pysnmp import hlapi

from discord.ext import commands

from datetime import datetime

import snmp

import matplotlib.pyplot as plt
import numpy as np
from pymongo import MongoClient
from pymongo import ASCENDING

import threading

import time

load_dotenv()
TOKEN = os.getenv('DISCORD_TOKEN')
GUILD = os.getenv('DISCORD_GUILD')
SHA_PASSWORD = os.getenv('SHA_PASSWORD')
AES_PASSWORD = os.getenv('AES_PASSWORD')
SNMP_USER = os.getenv('SNMP_USER')

bot = commands.Bot(command_prefix="$")


@bot.command()
async def snmpget(ctx, *args):
        order = args[0]
        if(order == 'sysName'):
                await
ctx.channel.send(snmp.get('127.0.0.1',['1.3.6.1.2.1.1.5.0'],hlapi.UsmUserData(SNMP_USER,
authKey=SHA_PASSWORD, privKey=AES_PASSWORD, authProtocol=hlapi.usmHMACSHAAuthProtocol,
privProtocol=hlapi.usmAesCfb128Protocol)))
        elif(order == 'sysLocation'):
                await ctx.channel.send(snmp.get('127.0.0.1',['1.3.6.1.2.1.1.6'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol)))
        elif(order == 'sysDescr'):
                await ctx.channel.send(snmp.get('127.0.0.1',['1.3.6.1.2.1.1.1'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol)))
        elif(order == 'cpuLoad'):
                await ctx.channel.send(snmp.get('127.0.0.1',['1.3.6.1.4.1.2021.10.1.3.3'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol)))
        elif(order == 'ramFree'):
                await ctx.channel.send(snmp.get('127.0.0.1',['1.3.6.1.4.1.2021.4.11.0'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol)))
        elif(order == 'freeDiskSpace'):
                await ctx.channel.send(snmp.get('127.0.0.1',['1.3.6.1.4.1.2021.9.1.7.1'],
```

```
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol)))

@bot.command()
async def snmpGetBulk(ctx, *args):
        order = args[0]
        max_mibs = int(args[1])

        if(order == 'memory'):
                await ctx.channel.send(snmp.getBulk('127.0.0.1', ['1.3.6.1.4.1.2021.4'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol),max_mibs))
        elif(order == 'disk'):
                await ctx.channel.send(snmp.getBulk('127.0.0.1', ['1.3.6.1.4.1.2021.9.1'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol),max_mibs))
        elif(order == 'cpuTimes'):
                await ctx.channel.send(snmp.getBulk('127.0.0.1', ['1.3.6.1.4.1.2021.11'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol),max_mibs))
        elif(order == 'network'):
                await ctx.channel.send(snmp.getBulk('127.0.0.1', ['1.3.6.1.2.1.2.2'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol, privProtocol=hlapi.usmAesCfb128Protocol),max_mibs))

@bot.event
async def on_ready():
    for guild in bot.guilds:
        if guild.name == GUILD:
            break

    print(
        f'{bot.user} is connected to the following guild:\n'
        f'{guild.name}(id: {guild.id})'
    )

    members = '\n - '.join([member.name for member in guild.members])
    print(f'Guild Members:\n - {members}')

@bot.command()
async def GR(ctx,*args):
        print(args)
        order = args [0]
        client = MongoClient('localhost',27017)
        db = client.grs
        find = db.metric.find().sort('created_at', ASCENDING).limit(15)
        if order == "Memory":
                xList = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]
                yList = []#Obtener los 15 valores de memoria
                for i in find:
                        yList.append(i['mem'])
                x = np.array(xList)
                y = np.array(yList)
                plt.plot(x,y)
                plt.savefig(fname='plot')
                await ctx.send(file = discord.File('plot.png'))
                os.remove('plot.png')
                plt.clf()
        elif order == "CPU":
                xList = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]
                yList = []#Obtener los 15 valores de memoria
                for i in find:
                        yList.append(i['cpu'])
                x = np.array(xList)
                y = np.array(yList)
                plt.plot(x,y)
                plt.savefig(fname='plot')
                await ctx.send(file = discord.File('plot.png'))
                os.remove('plot.png')
                plt.clf()

##Crear hilo
```

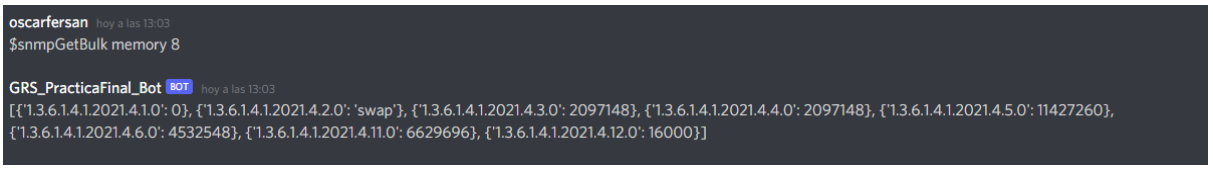
```
def BD ():
        client = MongoClient('localhost',27017)
        db = client.grs
        while(True):
                time.sleep(30)
                memoria = snmp.get('127.0.0.1',['1.3.6.1.4.1.2021.4.11.0'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol,
privProtocol=hlapi.usmAesCfb128Protocol))['1.3.6.1.4.1.2021.4.11.0']
                cpu = snmp.get('127.0.0.1',['1.3.6.1.4.1.2021.10.1.3.3'],
hlapi.UsmUserData(SNMP_USER, authKey=SHA_PASSWORD, privKey=AES_PASSWORD,
authProtocol=hlapi.usmHMACSHAAuthProtocol,
privProtocol=hlapi.usmAesCfb128Protocol))['1.3.6.1.4.1.2021.10.1.3.3']
                date = datetime.now()
                print(date)
                db.metric.insert_one({'createdAt':date,'mem': memoria,'cpu':cpu})

hilo = threading.Thread(target=BD)
hilo.start()

bot.run(TOKEN)
```

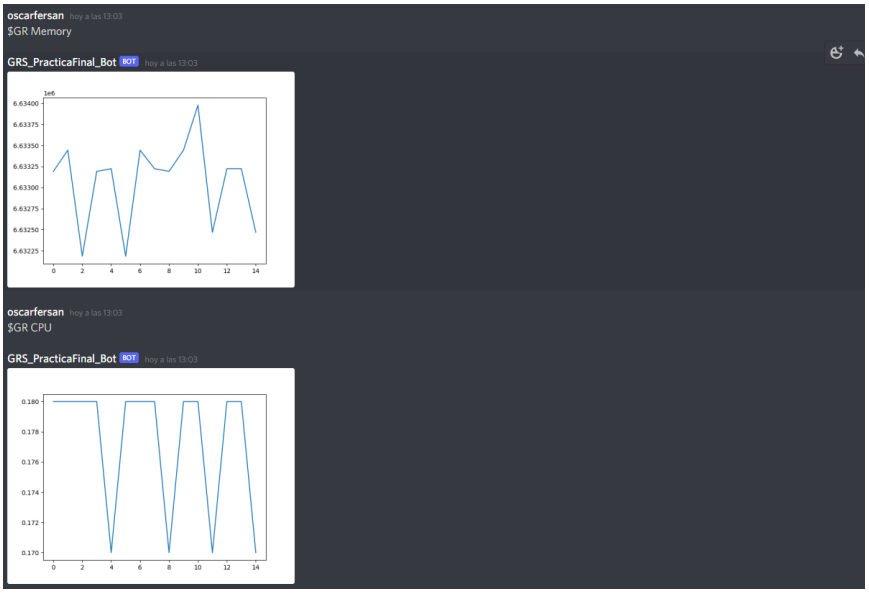
snmp.py

```
from pysnmp import hlapi

def cast(value):
    try:
        return int(value)
    except (ValueError, TypeError):
        try:
            return float(value)
        except (ValueError, TypeError):
            try:
                return str(value)
            except (ValueError, TypeError):
                pass
    return value

def construct_object_types(list_of_oids):
    object_types = []
    for oid in list_of_oids:
        object_types.append(hlapi.ObjectType(hlapi.ObjectIdentity(oid)))
    return object_types

def fetch(handler, count):
    result = []
    for i in range(count):
        try:
            error_indication, error_status, error_index, var_binds =
next(handler)
            if not error_indication and not error_status:
                items = {}
                for var_bind in var_binds:
                    items[str(var_bind[0])] = cast(var_bind[1])
```

```
                result.append(items)
            else:
                raise RuntimeError('Got SNMP error:
{0}'.format(error_indication))
        except StopIteration:
            break
    return result

def get(target, oids, credentials, port=161, engine=hlapi.SnmpEngine(),
context=hlapi.ContextData()):
    handler = hlapi.getCmd(
        engine,
        credentials,
        hlapi.UdpTransportTarget((target, port)),
        context,
        *construct_object_types(oids)
    )
    return fetch(handler, 1)[0]

def getBulk(target, oids, credentials, count, start_from=0, port=161,
            engine=hlapi.SnmpEngine(), context=hlapi.ContextData()):
    handler = hlapi.bulkCmd(
        engine,
        credentials,
        hlapi.UdpTransportTarget((target, port)),
        context,
        start_from, count,
        *construct_object_types(oids)
    )
    return fetch(handler, count)
```

Ejemplos de uso





Bibliografía

Repositorio de Github: [oscarfersan/GestionDeRedes: Practica final de gestion de redes y seguridad (github.com)]

Instalacion SNMP: [Quick Way to Install and Configure SNMP on Ubuntu 20.04 - kifarunix.com]

Instalacion MongoDB: [Install MongoDB Community Edition on Ubuntu — MongoDB Manual]