

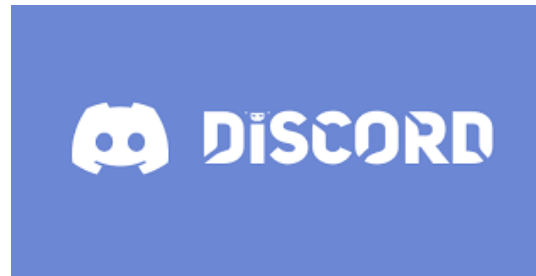
# Trabajo final

## Ingeniería de Servidores

Uso de la API de Zabbix y Discord para notificar eventos.



**UNIVERSIDAD  
DE GRANADA**



**Juan Miguel Acosta Ortega**

# Índice

1) Motivación de la práctica.....	2
2) API de Discord.....	2
3) API de Zabbix.....	2
4) Bot de Discord.....	2

# 1) Motivación de la práctica

En Zabbix podemos configurar de manera nativa los disparadores (Triggers), estos son expresiones lógicas que "evalúan" los datos recopilados por los elementos. y representan el estado actual del sistema.

Para ello creamos "Items" de nuestro interés o definimos / asignamos "Templates" a los Hosts (servidores que queramos monitorizar) que conllevan agregar múltiples Items y Triggers predefinidos.

En Zabbix se representan de manera visual los problemas ( iniciados por los disparadores ), ya que podemos definir su grado de importancia con colores ( Azul → Information, Naranja → Warning ...), y aunque a nivel de usuario es bastante visual se ha de ingresar en la web de zabbix ( < ipZabbixServer >/zabbix ) para visualizarlos.

Para mayor comodidad podemos usar la Api de Zabbix para consultar / eliminar / crear / actualizar ciertos parámetros cuando queramos, y combinar esto con bots de redes sociales y otros métodos de comunicación / notificación.

Es por ello que en esta práctica desarrollaré un bot de Discord que notifique los problemas de nuestro servidor.

**Para la realización de esta práctica se ha de tener como mínimo Zabbix configurado en un servidor y todo lo que ello conlleva (SSH, Puertos habilitados, ficheros de configuración, pila LAMP...).**

## 2) API de Zabbix

En el apartado 19 de la [documentación de Zabbix](#) podemos encontrar información que nos proporciona una descripción general de las funciones proporcionadas por la API de Zabbix y nos ayudará a orientarnos en las clases y métodos disponibles.

El API de Zabbix nos permite recuperar y modificar de forma programada la configuración de Zabbix y provee acceso a los datos históricos. Está ampliamente usado para :

- Crear nuevas aplicaciones para trabajar con Zabbix;
- Integrar Zabbix con software de terceras partes;
- Automatizar tareas rutinarias.

El API de Zabbix es API basado en web y está integrado como parte de la interfaz de la web. El usa protocolo JSON-RPC 2.0 lo cual significa dos cosas:

- El API consiste en un conjunto de métodos separados;

- Solicitudes y respuestas entre los clientes y el API están codificados usando el formato JSON.

Vemos que si por ejemplo nos interesa interactuar de alguna manera con los “Host”, la Api nos proporciona:

- **host.create**
- **host.massad**
- **host.massremove**
- **host.massupdate**
- **host.update**

Estos métodos más allá de ser lo mismo que nos ofrece la interfaz gráfica de Zabbix, pasan a ser una ayuda a la hora de gestionar grandes masas de Hosts, ya que configurar por ejemplo 100 nos llevaría bastante tiempo mientras que con el uso de la Api se podría reducir considerablemente.

Esta filosofía de trabajo se extrapola a eventos , triggers, items ... y demás objetos configurables en Zabbix.

En primera instancia para poder hacer uso de la Api tenemos que asegurarnos que hayamos habilitado su uso, esto podemos hacerlo en “/etc/zabbix/zabbix\_agent.conf”:

```
### Option: EnableRemoteCommands - Deprecated, use AllowKey=system.run[*] or DenyKey=system.run[*] instead
# Internal alias for AllowKey/DenyKey parameters depending on value:
# 0 - DenyKey=system.run[*]
# 1 - AllowKey=system.run[*]
#
# Mandatory: no

AllowKey=system.run[*]
```

Al ser un servicio lo ponemos de nuevo en marcha actualizado con “systemctl restart zabbix-agent”.

En mi caso sólo voy a testear con la misma máquina que aloja el server : Ubuntu server.

Para comprobar el correcto funcionamiento de la API, por lo menos en local, probamos un ejemplo sencillo con “curl” para conocer la versión de Zabbix:

```
curl --request POST \
> --url 'http://192.168.56.105/zabbix/api_jsonrpc.php' \
> --header 'Content-Type: application/json-rpc' \
> --data '{"jsonrpc":"2.0","method":"apiinfo.version","params":{},"id":1}'
```

**curl:** Es el comando que se utiliza para realizar solicitudes desde la línea de comandos. En este caso, se está utilizando para enviar una solicitud HTTP POST.

**--request POST:** Indica que la solicitud HTTP debe ser de tipo POST. Esto se usa porque las solicitudes a la API de Zabbix generalmente requieren que se utilice el método POST para enviar datos.

**--url 'http://192.168.56.105/zabbix/api\_jsonrpc.php':** Especifica la URL a la cual se enviará la solicitud. En este caso, la URL es la ubicación de la API de Zabbix (api\_jsonrpc.php) en el servidor con la dirección IP 192.168.56.105. Nota que la URL utiliza el protocolo HTTP.

**--header 'Content-Type: application/json-rpc':** Define el encabezado de la solicitud HTTP. En este caso, se está indicando que el contenido de la solicitud está en formato JSON y que es una solicitud JSON-RPC. Este encabezado es importante para que el servidor Zabbix comprenda el formato de los datos enviados.

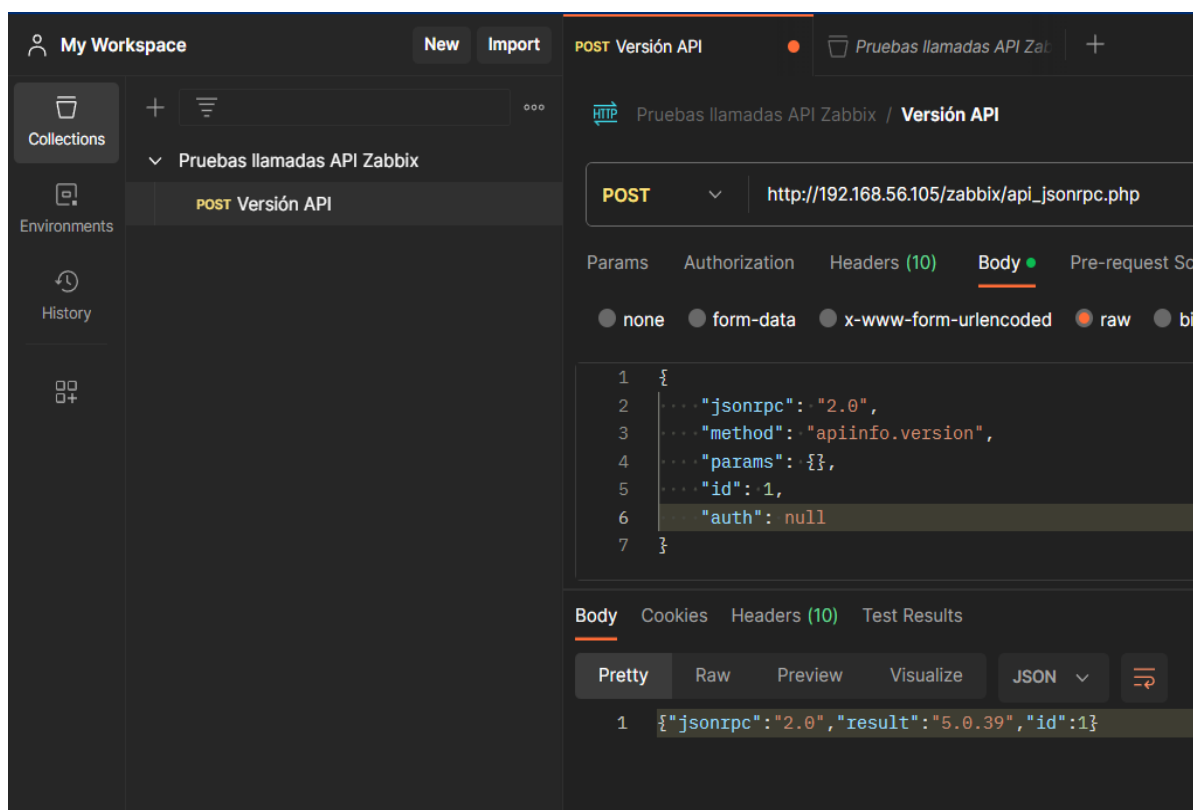
**--data '{"jsonrpc":"2.0","method":"apiinfo.version","params":{},"id":1}':** Especifica los datos que se enviarán en el cuerpo de la solicitud POST. En este caso, se está enviando un objeto JSON que contiene información sobre la solicitud a la API de Zabbix. Los campos clave aquí son:

- **"jsonrpc": "2.0":** Indica que se está utilizando la versión 2.0 del protocolo JSON-RPC.
- 
- **"method": "apiinfo.version":** Especifica el método que se está llamando en la API de Zabbix, en este caso, apiinfo.version.
- 
- **"params": {}:** Indica que no se están pasando parámetros específicos para este método.
- 
- **"id": 1:** Proporciona un identificador único para la solicitud.

**La respuesta:**

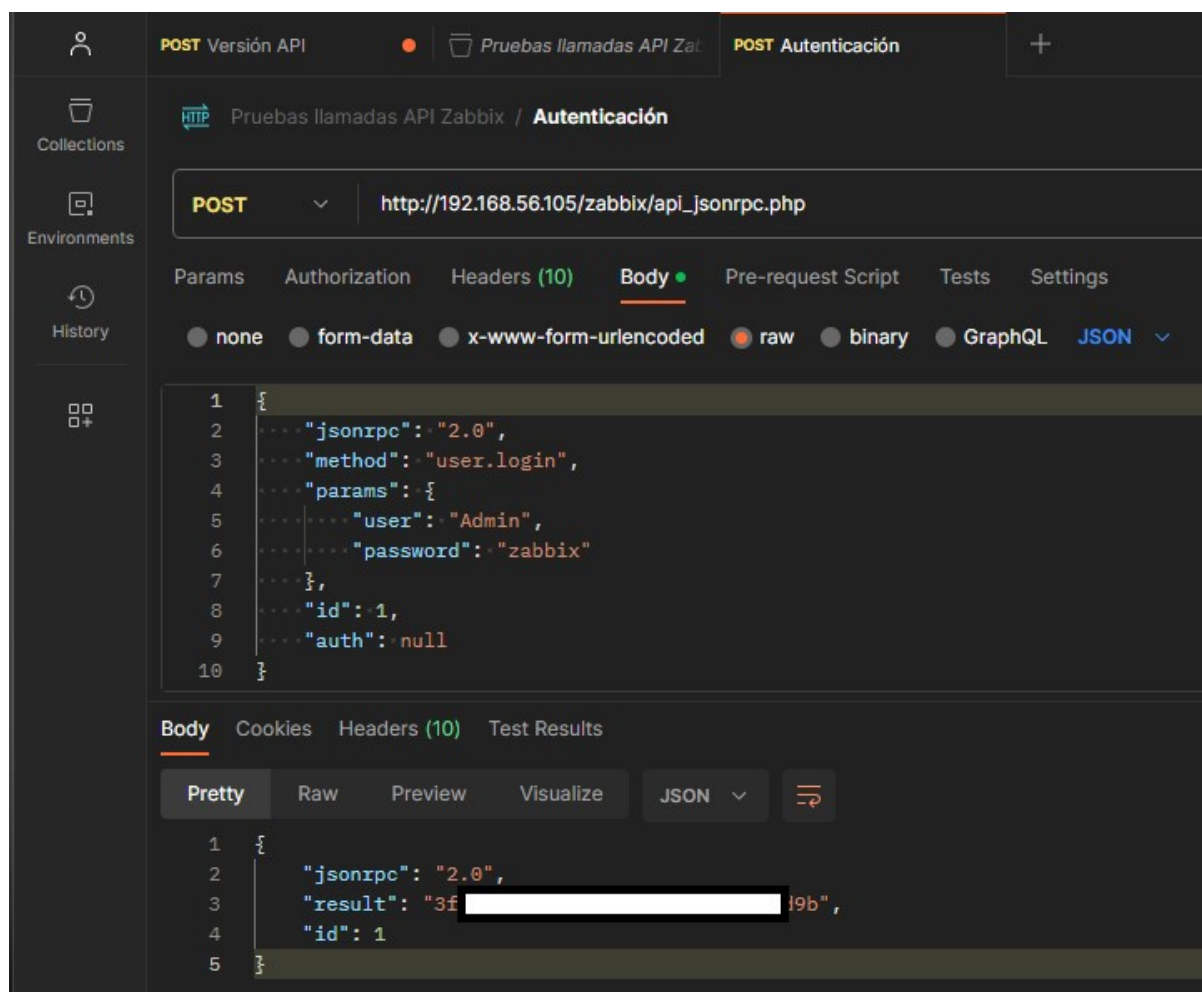
```
{"jsonrpc": "2.0", "result": "5.0.39", "id": 1}juan@ubuntu:~$
```

Además hacemos otra prueba en Postman con la misma petición para ver que todo funciona, y seguir en este entorno que es más visual:



Vemos que obtenemos la misma respuesta, además el “Content-type” es el mismo y se especifica en la cabecera (application/json).

Para empezar a trabajar de verdad con la API hemos de autenticarnos para poder usar el token de la API, ya que en la petición anterior no era necesario ( **“auth”: null** ).



Ahora hemos obtenido el token de autenticación y podemos hacer peticiones reales a la API.

Para más comodidad y no tener que copiar y pegar el token cada vez que se quiera hacer uso de él , podemos hacer un pequeño script en javascript para obtenerlo y guardarlo en una variable de entorno de postman, así haremos referencia al token a través de esta.

```

Params    Authorization    Headers (10)    Body ●    Pre-request Script    Tests ●    Settings
1  var data = JSON.parse(responseBody); //cogemos el cuerpo
2  postman.setEnvironmentVariable("ZTOKEN",data.result);//Hacemos
3  //una variable de entorno
4  pm.sendRequest("https://postman-echo.com/get",
5    function (err, response){
6      console.log(response.json());
7    }
8  );

```

Para dar un paso más en el uso de la API de Zabbix vamos a obtener los hosts configurados:

▼ Pruebas llamadas API Zabbix

- POST Versión API
- POST Autenticación
- POST Obtener Hosts configurados

POST ▼ http://192.168.56.105/zabbix/api\_jsonrpc.php

Params Authorization Headers (10) Body ● Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ b

```

1  {
2    "jsonrpc": "2.0",
3    "method": "host.get",
4    "params": {
5      "output": [
6        "hostid",
7        "host"
8      ],
9      "selectInterfaces": [
10       "interfaceId",
11       "ip"
12     ]
13   },
14   "id": 2,
15   "auth": "{ZTOKEN}"
16 }

```

En el cuerpo de la llamada utilizamos la variable de entorno que referencia al token, y el método es **host.get**. Además especificamos la salida de la respuesta, en este caso nos da los nombres e ip de los host y sus id.

En mi caso tengo dos servidores configurados, un servidor Ubuntu, y otro Rocky.



```
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "jsonrpc": "2.0",
3   "result": [
4     {
5       "hostid": "10084",
6       "host": "Zabbix server",
7       "interfaces": [
8         {
9           "interfaceid": "1",
10          "ip": "127.0.0.1"
11        }
12      ]
13    },
14    {
15      "hostid": "10439",
16      "host": "Rocky agent",
17      "interfaces": [
18        {
19          "interfaceid": "3",
20          "ip": "192.168.56.110"
21        }
22      ]
23    }
24  ],
25  "id": 2
26 }
```

El siguiente paso será utilizar la API para obtener **información de nuestro interés** para esta práctica. Aunque se pueden realizar infinidad de actividades, el enfoque es el de obtener la información de los **problemas del servidor**.

En Zabbix, tanto los eventos como las alertas son conceptos importantes pero tienen significados distintos. A continuación, se describen las diferencias entre eventos y alertas en Zabbix:

### Evento:

Un evento en Zabbix es una situación o cambio que ocurre en el entorno supervisado, como un problema o una condición anormal.

Los eventos pueden ser generados por la evaluación de disparadores (triggers) que están configurados para monitorear ciertos valores o condiciones en los elementos supervisados.

Los eventos pueden tener varios estados, como "No resuelto" y "Resuelto", lo que indica si el problema sigue sin resolverse o si ha sido corregido.

Los eventos pueden ser visualizados en la interfaz de usuario de Zabbix para rastrear problemas y cambios en el sistema.

### Alerta:

Una alerta en Zabbix se refiere a la notificación generada en respuesta a un evento que ha ocurrido.

Las alertas se envían a través de varios medios, como correo electrónico, mensajes de texto, Slack u otros métodos de notificación configurados en Zabbix.

Las alertas son activadas cuando un evento se encuentra en estado "No resuelto". Es decir, cuando Zabbix detecta un problema pero aún no ha sido resuelto.

Puedes configurar diferentes tipos de acciones en Zabbix que definan cómo se generan y envían las alertas cuando se produce un evento específico.

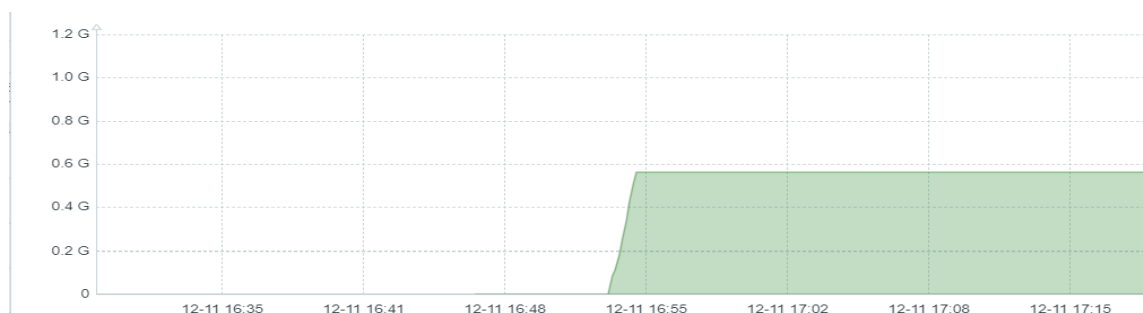
En resumen, un evento es una representación interna de un cambio o problema detectado en el sistema supervisado, mientras que una alerta es una notificación externa generada en respuesta a un evento no resuelto. Los eventos pueden tener diferentes estados a lo largo del tiempo, mientras que las alertas están relacionadas con el estado "No resuelto" de un evento. En esta práctica **serán los dos "objetos" de zabbix que notificaremos en el bot.**

Estableceremos como ejemplo la monitorización del volumen de peticiones entrantes y salientes a una interfaz de red en concreto "**enp0s8**", y estableceremos un trigger que notifique un problema al sobrepasar un cierto umbral.

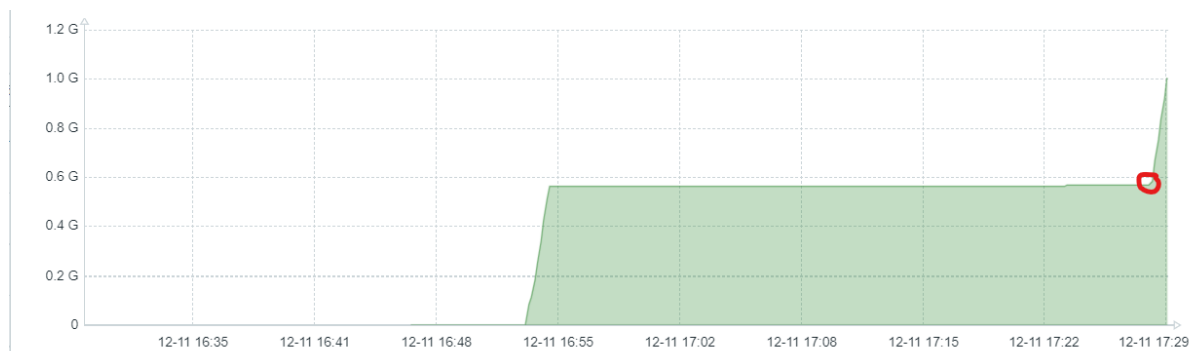
Para monitorizar este aspecto creamos un ítem (**predefined key** → **net.if.total[enp0s8,bytes]**):

Key	Interval	History	Trends	Type
net.if.total[enp0s8,bytes]	10s	90d	365d	Zabbix agent

Comprobamos en un gráfico que podemos visualizar de manera correcta el tráfico en la interfaz de red:



Además podemos simular varias peticiones al servidor ( caso inusual ) para activar el trigger que más adelante configuraremos. Así se verían 1000000 peticiones hechas con Apache Benchmark desde otro servidor diferente al Ubuntu:



Se ve claramente que el tráfico aumenta drásticamente.

Ahora pasamos a definir un trigger asociado al ítem que monitoriza el tráfico en la interfaz “enp0s8”. Para ello y aunque quizás no sea lo más adecuado porque también hay que tener en cuenta el tiempo que pasa entre ciertas muestras, escogemos la función “**last()**” para comparar el valor de la muestra con un umbral (1G).

Simulamos de nuevo un tráfico inusual ( `ab -n 1000000 -c 100 http://192.168.56.105/index.html` ), y vemos que el trigger se activa. Comprobemos que a través de la API podemos obtener el nombre del evento:

Problems			
Time ▼	Info	Host	Problem • Severity
00:37:35		Ubuntu	Tráfico inusual en la interfaz de red enp0s8

En caso de sólo querer recibir unos eventos en concreto podemos sólo recoger los eventos con un “triggerid” en concreto:

```
192.168.56.105/zabbix/triggers.php?form=update&triggerid=20174
```

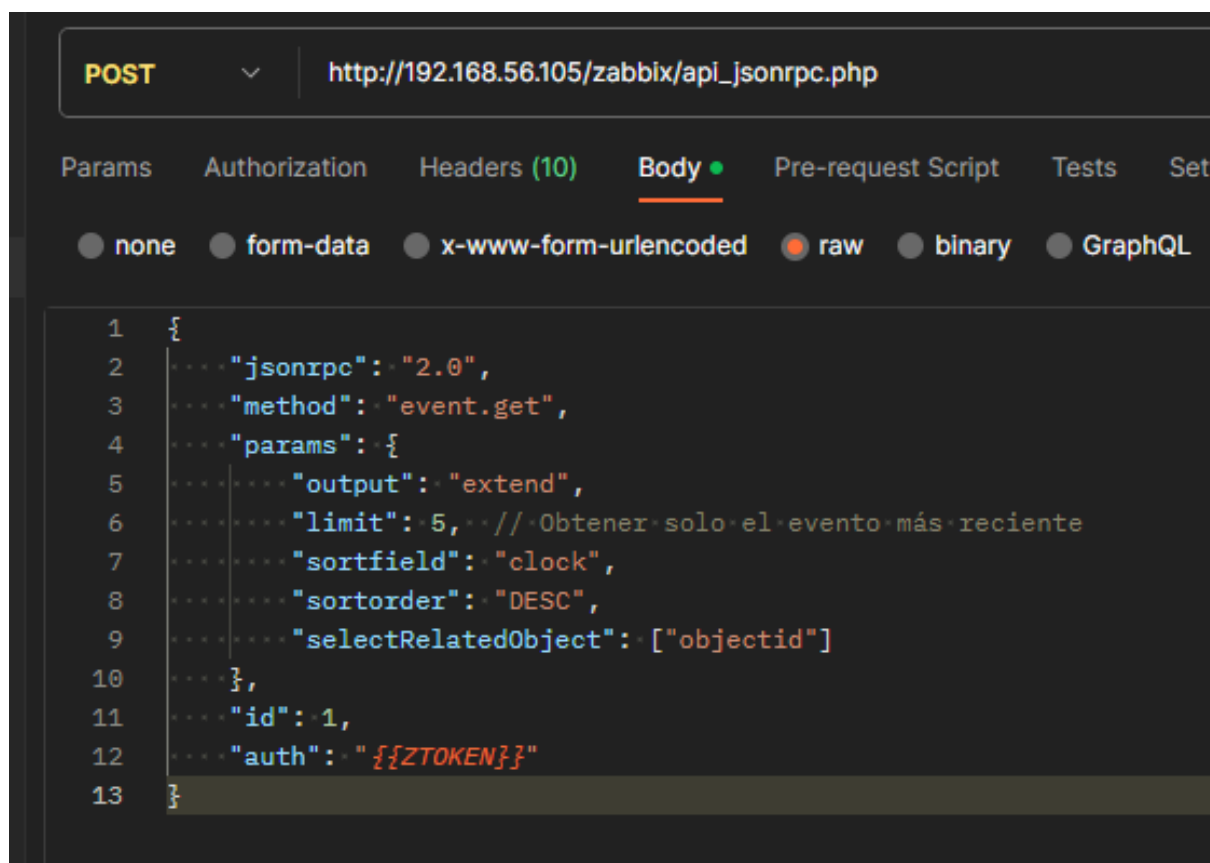
ers

ts / Ubuntu Enabled ZBX SNMP JMX IPMI Applications 22 Items 168 Triggers 76 Graphs 38 Discove

r Tags Dependencies

\* Name

Pero para abarcar los últimos eventos independientemente de cuáles sean cogemos los cinco problemas más recientes:



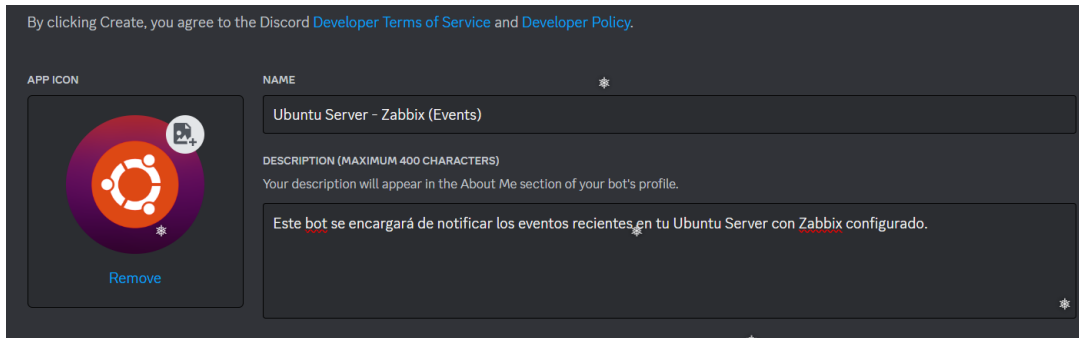
Y vemos que uno de estos es el que queríamos recoger, el que detectaba el tráfico inusual en una interfaz de red en concreto :

```
Body Cookies Headers (10) Test Results
Pretty Raw Preview Visualize JSON ↗
{
  "urls": [],
  },
  {
    "eventid": "1114",
    "source": "0",
    "object": "0",
    "objectid": "20174",
    "clock": "1702420295",
    "value": "1",
    "acknowledged": "0",
    "ns": "408578517",
    "name": "Tráfico inusual en la interfaz de red enp0s8",
    "severity": "2",
    "r_eventid": "1115",
    "c_eventid": "0",
    "correlationid": "0",
    "userid": "0",
    "opdata": "",
    "relatedObject": {
      "triggerid": "20174"
    },
    },
    "suppressed": "0",
    "urls": []
  },
  },
```

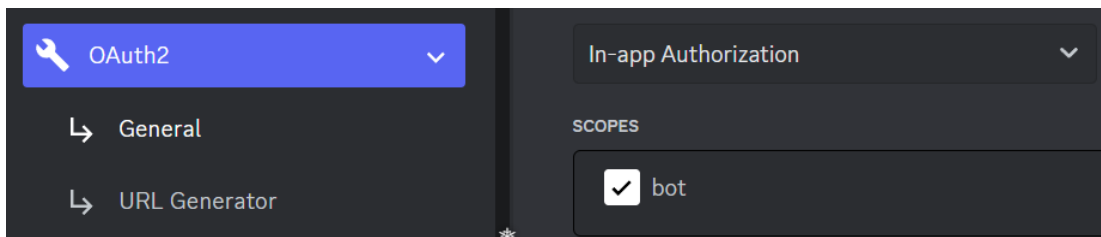
Con este primer acercamiento al uso de la API de Zabbix hemos visto cómo recoger información con ella. En el siguiente apartado veremos cómo crear un servidor de Discord que nos notifique los eventos del servidor.

### 3) Bot de Discord

Lo primero que debemos hacer para empezar a crear nuestro bot es acceder, con nuestro usuario de Discord a [Discord Developer Portal](https://discord.com/developers) y crear una nueva Aplicación:

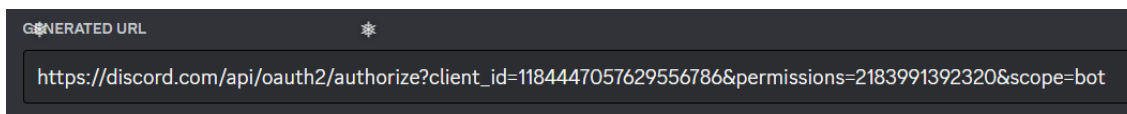


Seguido especificamos el tipo de autorización y los tipos de permisos, en este caso de texto, que nos serán necesarios:



Además creamos la URL del bot y volvemos a seleccionar los permisos necesarios y en la sección de BOT se ha de seleccionar la opción de MESSAGE CONTENT INTENT.

Este será el link necesario para invitar al bot a uno de nuestros servidores de Discord:

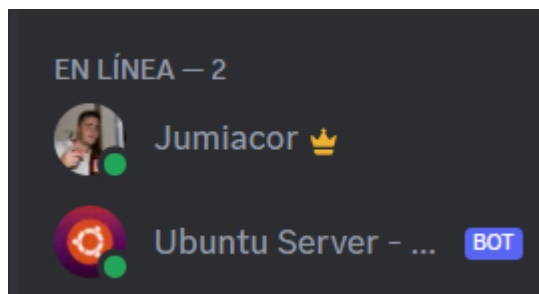


Una vez guardado el link procedemos a crear el Bot, guardaremos el Token ya que es único e intransferible. En mi caso como el Bot lo programaré el Python lo guardaré en un fichero "private.py", aunque también se podría guardar en una variable de entorno como hicimos anteriormente.

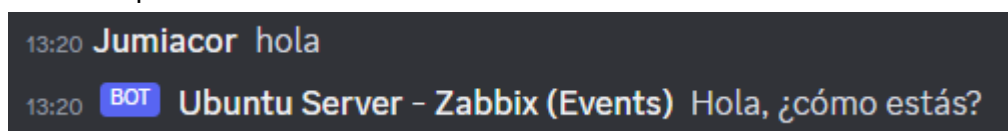
Instalamos con "pip" **discord** y **py-zabbix**.

Una vez configurado el entorno comenzamos a hacer el bot. Un módulo se encargará de las respuestas, otro en iniciar el bot y el main que será el que inicie todo.

Accedemos al link guardado anteriormente y esto nos permite meter al Bot en uno de nuestros Servidores, pero aparecerá como desconectado. Si ejecutamos el código nos aparecerá “En Línea”:



Ahora mismo con mensajes sencillos el bot funciona ya que sólo estamos manejando dos o tres mensajes por parte del usuario, en caso de recibir un mensaje inválido pediría un “!events” que es realmente la función del bot.



Ahora vamos a adentrarnos en la librería “py-zabbix” y ver cómo se hacen peticiones a la API de Zabbix, para ello podemos ver tanto la documentación oficial como el github:

<https://github.com/adubkov/py-zabbix>

Para hacer la misma petición que recoge los eventos en python lo hacemos de la siguiente manera:

```
zapi = ZabbixAPI(url='http://192.168.56.105/zabbix/', user='Admin', password='zabbix')
num_events = 5
result = zapi.do_request('event.get', {
    'output': 'extend',
    'limit': 5, # Obtener solo el evento más reciente
    'sortfield': 'clock',
    'sortorder': 'DESC',
    'selectRelatedObject': ['objectid']
})
zapi.user.logout()
```

En esta muestra nos logueamos y recibimos el Token en “zapi”, y así podemos hacer la petición de los eventos con los mismos parámetros que anteriormente.

En la misma función transformamos el resultado a texto haciendo ciertas acciones y este sería el resultado:

```
15:53 Jumiacor leventos
15:53 BOT Ubuntu Server - Zabbix (Events) Eventos del servidor más recientes:
Ubuntu has been restarted (uptime < 10m): Tipo de evento:Not clasified relacionado con el trigger 16054
Zabbix agent is not available (for 3m): Tipo de evento:Average relacionado con el trigger 20136
Tráfico inusual en la interfaz de red enp0s8: Tipo de evento:Not clasified relacionado con el trigger 20174
Load average is too high (per CPU load over 1.5 for 5m): Tipo de evento:Not clasified relacionado con el trigger 16565
```

## 4) Conclusión

Con este resultado damos por terminada esta pequeña práctica sobre la API de Zabbix y la API de Discord.

Tras lo visto con las dos APIs vemos que las posibilidades son infinitas y que realmente se pueden desarrollar aplicaciones muy interesantes con estas dos herramientas.

Una de estas ideas podrían ser:

- Creación masiva de objetos para aligerar trabajo
- Gestionar alertas masivas tras el arreglo de problemas
- Obtención de información de manera automática
- Crear un timer que vaya notificando eventos y alertas en ciertos periodos de tiempo
- etc