Guía de desarrollo de plugins

Guía de desarrollo de plugins

Introducción

Esta guía para el desarrollo de complementos (*plugins)* contiene documentación para los usuarios que quieran crear *plugins* propios. Utilice esta documentación para aprender a crear *plugins* que solucionen sus necesidades comerciales.

¿Qué es un plugin?

Un complemento informático, también conocido como *plugin*, es una aplicación que permite extender las funciones de otra aplicación, en este caso de Pandora FMS.

¿Por qué son útiles los plugins?

Los *plugins* permiten extender las funcionalidades de Pandora FMS, lo que ofrece gran variedad de posibilidades a la hora de desarrollar nuevas opciones de monitorización.

Con estos se puede integrar la opción de monitorizar servicios, aplicaciones, bases de datos y mucho más, incluso se pueden usar para modificar o automatizar tareas y procesos.

Uno de los objetivos es poder monitorizar el mayor número de sistemas y servicios posibles en un mismo entorno de trabajo y el uso de *plugins* ayuda bastante en este cometido.

La utilidad de la mayoría de *plugins* es poder mostrar en Pandora FMS los datos o estadísticas de rendimiento que se recopilan de servicios externos.

Tipos de plugins en Pandora FMS

Por tipo de ejecución:

- *Plugin* de agente: Los *plugins* de agente son ejecutados por el Agente Software de Pandora FMS, suelen ser locales y no funcionan de forma remota por lo que la ejecución la realizará el *daemon* del agente. Los *plugins* de agente en su ejecución suelen imprimir en un XML los módulos con los datos.
- *Plugin* de servidor: Los *plugins* de servidor son ejecutados por el Servidor de complementos, suelen sacar los datos de forma remota, ya sea por API u otro método, por lo que si bien pueden ser configurados como *plugin* de agente (en su mayoría), la opción recomendada es configurar como *plugin* de servidor. Los *plugins* de servidor imprimen un solo valor, por ejemplo 1 para indicar que la ejecución ha sido correcta.

¿Cómo encuentran los usuarios los plugins?

Pandora FMS cuenta con una biblioteca desde la que se pueden descargar todos los *plugins* creados para el sistema.

Cualquier persona puede subir *plugins* a la biblioteca, aunque hay un proceso de revisión antes de que estos sean aceptados y publicados.

https://pandorafms.com/library/

En la actualidad la biblioteca cuenta con una gran cantidad de *plugins* que abarcan todos los campos, bases de datos, aplicaciones, *cloud*, servicios de mensajería…

La biblioteca cuenta con un buscador con el que se puede buscar cualquier plugin subido a esta, además de distintos *tags* y menús con los que poder navegar por esta cómodamente.

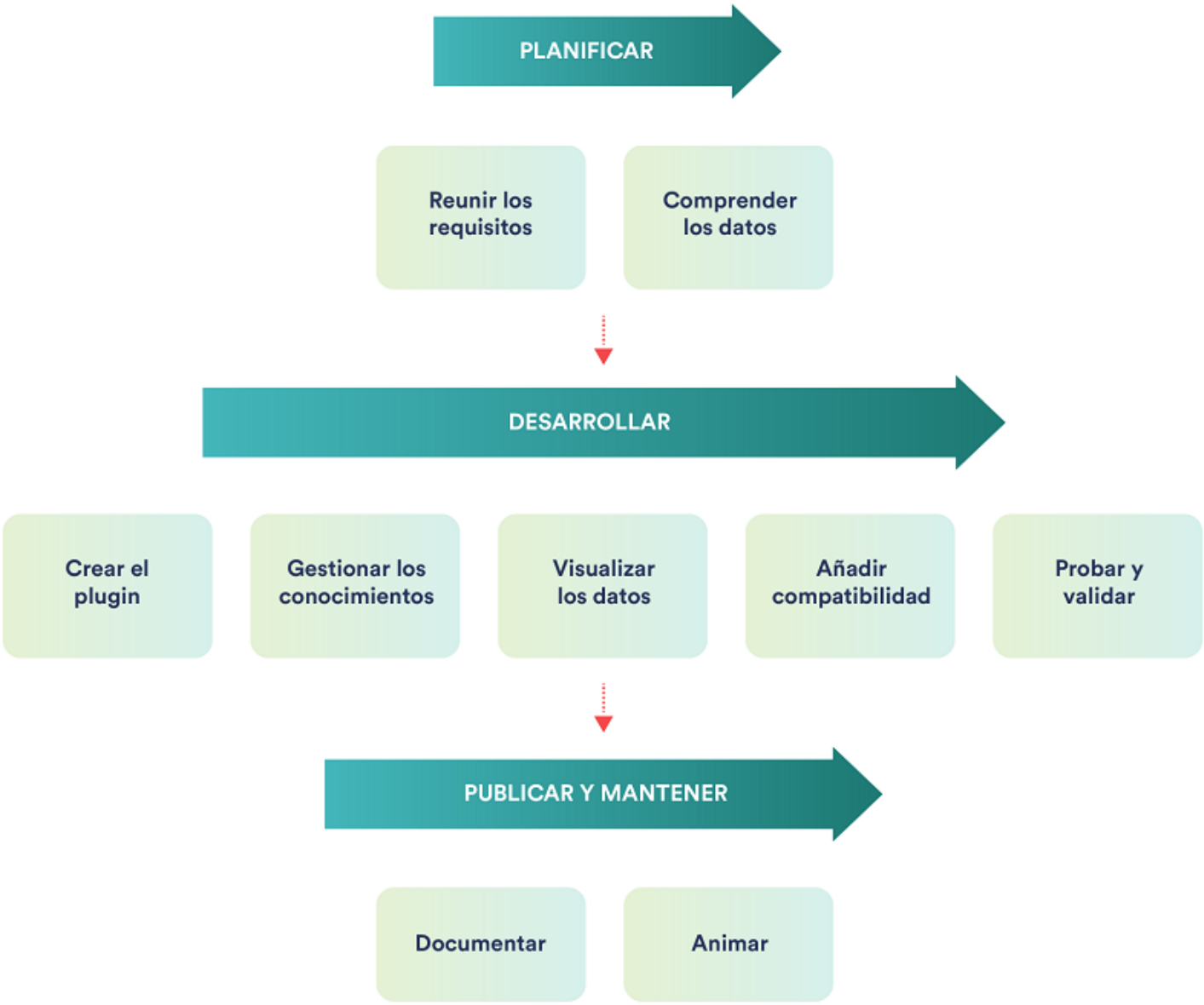
Cada plugin suele venir acompañado de su documentación donde viene detallada la instalación, configuración del plugin y los datos que recoge.

¿Cómo se instalan los plugins ?

Los *plugins* se pueden instalar desde la consola de Pandora FMS. Según su tipo se pueden ejecutar de dos formas diferentes, lo que creará distintas formas de configuración del *plugin*. Los *plugins* de agente los ejecuta el Agente Software en cambio, los *plugins* de servidor los ejecuta el Plugin server.
Por lo general los *plugins* de servidor suelen crear agentes y módulos mientras que los *plugins* de agente sólo crean módulos dentro del Agente software en el que se ha configurado.

La instalación del *plugin* consiste en crear una ejecución personalizada de este en Pandora FMS. Cada cierto intervalo de tiempo configurable, el *plugin* se ejecutará y mostrará los módulos actualizados.

Ciclo de vida de un plugin



Planificación de un plugin

A la hora de desarrollar un *plugin* es importante planificar bien su desarrollo teniendo en cuenta algunos aspectos.

¿Qué problema resuelve el plugin?

Se debería poder responder a estas preguntas en un primer paso en la planificación del *plugin*:

- ¿Cuál es el propósito del *plugin*?
- ¿Cuáles son los casos de uso que abordará su *plugin*?

Es importante tener claro qué solución se pretende lograr con el *plugin*. Esta puede ser monitorizar un servicio de interés, logrando unificar todos los datos en Pandora FMS lo que puede suponer una ventaja en tiempo o costes, o también puede ser automatizar un proceso, como un *plugin* que lea los correos electrónicos y con el que se pueda activar una alarma cuando llegue

determinado mensaje.

Los *plugins* buscan solucionar una necesidad o facilitar la interacción, tratamiento y visualización de datos.

¿Cómo va a extraer los datos el plugin?

La principal utilidad de un *plugin*, es extraer datos, de un servicio, aplicación, base de datos y poder visualizar estos datos en Pandora FMS... ¿Pero cómo se puede extraer los datos del servicio requerido?

Esto, indudablemente, depende de la tecnología o servicio que se pretende explotar con el *plugin*, por lo que es necesario una investigación previa sobre el servicio elegido, para averiguar la viabilidad del *plugin* y si es posible la creación de un entorno de pruebas para evaluarlo.

Por lo general estos servicios suelen tener alguna forma de sacar las estadísticas de monitorización, esta forma suele variar dependiendo del servicio siendo lo más normal que dispongan de una API o un CLI con el que poder tratar los datos requeridos y poder mostrarlos con el *plugin*.

En algunos casos existen bibliotecas que la comunidad o la empresa creadora de la tecnología o servicio en cuestión ha creado para facilitar mucho la interacción con datos de sus servicios.

¿Los usuarios necesitan permiso para obtener los datos?

Es probable que a la hora de sacar los datos de algún servicio o tecnología se requiera la posesión de una cuenta que necesite unos determinados permisos para interactuar con los datos, o que se necesite realizar algunas configuraciones previas en el entorno antes de poder extraer estos datos.

Por lo general, estos requisitos suelen venir especificados en la documentación del servicio o tecnología en sí.

Es importante tener claro qué es necesario y cuáles son los requisitos para poder extraer los datos de ese servicio y dejarlos detallados en la documentación del *plugin*.

¿Los datos se recogen de forma remota o local?

Puede que las estadísticas de rendimiento se puedan extraer por API de forma remota, ya sea mediante alguna biblioteca, CLI, o por API con llamadas HTTP. O puede ser que estos datos solo se puedan sacar de forma interna o local por comandos o porque se usa una tecnología propia. Dependiendo del caso la ejecución del *plugin* cambiará y será diferente.

Un *plugin* que puede obtener los datos de forma remota, puede ejecutarse con el Plugin server, ya que no necesita estar presente en la máquina del servicio para obtener los datos.

Hay determinados *plugins* que solo pueden ejecutarse a través del Agente Software ya que necesitan estar en el mismo sistema del servicio del que se quiere sacar los datos para poder obtener estos.

¿Cómo desea visualizar los datos?

A la hora de definir cómo mostrar los datos del *plugin* en la consola de Pandora FMS es importante concretar cómo se desea visualizar estos datos.

Normalmente estos datos se van a visualizar siguiendo una estructura de agentes y módulos, por lo que es importante definir bien esta estructura para luego poder visualizar la información de la manera más cómoda posible y que no sea confusa, algo que puede ser difícil de lograr en *plugins* que tratan muchos datos o que monitorizan muchas "cosas", como pudiera ser el caso de tener muchas bases de datos, máquinas virtuales, etcétera.

En el *plugin* se pueden incluir opciones para personalizar el nombre de agentes o de módulos, o añadirles algún tipo de prefijo a su nombre para hacerlos más visibles y distinguibles.

Requisitos y dependencias del plugin

Dependiendo de las tecnologías utilizadas en el *plugin* puede ser necesaria la instalación de dependencias para su correcto funcionamiento. Estas pueden ser bibliotecas del lenguaje utilizado que se haya importado en el plugin para realizar alguna función o el lenguaje utilizado en sí.

Si el *plugin* está escrito en lenguaje Python versión 3, será necesario tener instalado `python3` en la máquina que se quiere usar el *plugin* lo que puede ser un gran obstáculo para posibles usuarios potenciales de ese *plugin*.

Una opción, para los *plugins* creados en lenguaje Python es aportar junto al *plugin* un archivo denominado `requirements.txt` que puede incluir todas las dependencias utilizadas por el *plugin*, así instalando este archivo se instalarán automáticamente todas las dependencias necesarias.

Ejemplo de archivo `requirements.txt` de un *plugin*, concretamente el de MongoDB remoto:

```
dnspython==2.1.0
pymongo==3.12.0
```

El archivo incluye dos bibliotecas que es necesario tener instaladas en el sistema para poder

lanzarlo (en caso de no ser un *plugin* compilado).

El archivo se puede escribir a mano, pero también puede generar un archivo `requirements.txt` que incluya todas las dependencias instaladas del entorno virtual con el comando `pip freeze`.

```
pip freeze> requirements.txt
```

Se puede copiar el archivo `requeriments.txt` en el entorno en el que se quieren instalar las dependencias y lanzar el siguiente comando para instalarlas:

```
pip install -r requirements.txt
```

Compilación de plugins

Una forma de solucionar problemas de dependencias con los *plugins* es la creación de ejecutables que ya vengan con todas las dependencias instaladas, esta es una solución ideal para los *plugins* escritos en lenguaje Perl y para los *plugins* escritos en Python.

En el caso de Python para crear los ejecutables se dispone de la biblioteca `pyinstaller`:

https://pypi.org/project/pyinstaller/

Esta puede instalarse con el siguiente comando:

```
pip install pyinstaller
```

Una vez instalada, se puede usar el siguiente comando para crear un binario del plugin en cuestión, en un archivo:

GNU/Linux®:

```
pyinstaller –onefile <nombre_plugin>.py
```

MS Windows®:

```
python3 -m PyInstaller –onefile <nombre_plugin>.py
```

Se generará una carpeta llamada `dist` con el binario dentro.

Puede darse el caso de que un *plugin* que ha sido compilado en un sistema operativo no funcione en otro, se recomienda compilar los *plugins* en CentOS 7, ya que los *plugins* compilados en este sistema suelen funcionar en todos los demás sistemas operativos GNU/Linux.

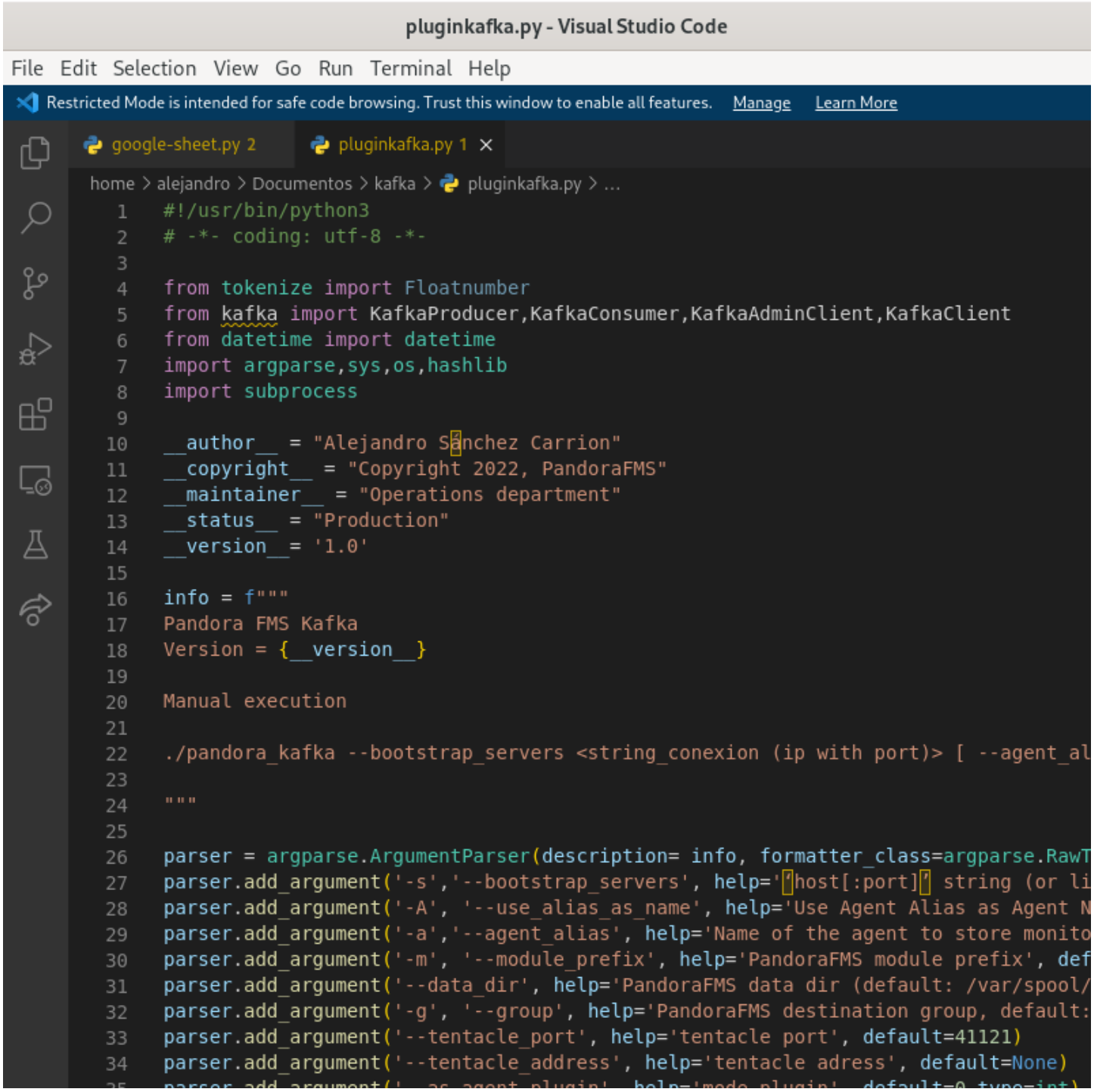
Con binarios creados en Fedora y Ubuntu se ha comprobado que en otros sistemas se genera un error de dependencias.

Desarrollo de un plugin

Es importante tener claro qué herramientas y tecnologías se van a utilizar en el desarrollo del *plugin*.

Herramientas de desarrollo

Los *plugins* son *scripts* que integran datos o una funcionalidad extra en Pandora FMS, no dejan de ser pequeños programas, por lo que para desarrollarlos es necesario disponer de un *framework* o un editor de código como puede ser Visual Studio Code.



A su vez herramientas como GitHub o como GitLab pueden ayudar a conservar y mantener el código de los *plugins* creados.

Lenguajes de programación

Para el desarrollo de los *plugins* es necesario usar un lenguaje de programación, Pandora FMS soporta cualquier tipo de lenguaje, mientras este muestre los datos en formato XML, Pandora FMS podrá entenderlos y mostrarlos en su consola. Para ello se debe seguir una estructura de etiquetas con el XML que incluye agentes, módulos y los atributos de estos.

Por lo general los lenguajes de *scripting* más utilizados en los *plugins* de Pandora FMS son Python,

Perl, BASH y Powershell, siendo los dos primeros probablemente los más completos.

Python y Perl cuentan con una herramienta desarrollada por Pandora FMS que facilita la creación de *plugins* llamada Plugin Tools, esta herramienta tiene un montón de funciones diseñadas para facilitar y automatizar la creación de *plugins*, por lo que puede ser una ventaja decisiva a la hora de elegir el lenguaje adecuado.

Herramientas útiles para montar entornos de pruebas

Para el desarrollo de un *plugin* suele ser necesario un entorno de pruebas, ya sea una instalación del servicio a monitorizar, una cuenta, etc. Dos tecnologías útiles para la creación de entornos debido a su rapidez y sencillez son Docker y Vagrant, ya que permiten levantar entornos con solo unos comandos o unas pequeñas configuraciones, aunque claro, es necesario que exista una solución en estos proyectos del servicio que se quiere comprobar.

XML

Para que Pandora FMS pueda consumir los datos que recibe del *plugin* para poder mostrarlos, es necesario que estos datos se traduzcan al formato XML.
Conocer el formato de los XML de datos de Pandora FMS le puede ayudar a mejorar los *plugins*, crear agentes personalizados con estos o simplemente enviar ficheros XML personalizados al servidor de datos de Pandora FMS.

Como cualquier documento XML, el fichero de datos debería comenzar con una declaración XML:

```
<?xml version='1.0' encoding='UTF-8'?>
```

No obstante, aunque el funcionamiento de los *plugins* y sus datos se basa en XML, solo los *plugins* de agente imprimirán un XML en una ejecución por terminal, los *plugins* de servidor solo muestran un dato simple, como puede ser un uno, que será el valor que contendrá el módulo que lo active, entonces lo más normal en estos casos es que emita un `1` si funciona y un `0` si por el contrario ha dado algún tipo de error.

Agentes y módulos

Los datos que recolecta el *plugin* pueden ser visualizados en Pandora FMS mediante agentes y módulos, por lo que es importante esquematizar la forma en la que se quieren mostrar los datos. Por ejemplo, cada agente podría ser una base de datos a monitorizar, con módulos que representan la información de esta.

La estructura XML de los agentes y de los módulos tienen distintos atributos para configurar estos.

El elemento `agent_data` que define al agente que envía los datos, soporta los siguientes atributos:

- `description`: Descripción del agente.
- `group`: Nombre del grupo al que pertenece el agente (debe existir en la base de datos de Pandora FMS). Si se deja vacío y no hay un grupo configurado por defecto en el servidor, el agente no se creará.
- `os_name`: Nombre del sistema operativo en el que corre el agente (debe existir en la base de datos de Pandora FMS).
- `os_version`: Cadena libre que describe la versión del sistema operativo.
- `interval`: Intervalo del agente (en segundos).
- `version`: Cadena con la versión del agente.
- `timestamp`: Marca de tiempo que indica cuándo se generó el XML (`YYYY/MM/DD HH:MM:SS`).
- `agent_name`: Nombre del agente.
- `timezone_offset`: Desplazamiento que se suma a la marca de tiempo (en horas). Útil si se trabaja con diferentes husos horarios (UTC).
- `address`: Dirección IP del agente (o FQN).
- `parent_agent_name`: Nombre del padre del agente.
- `agent_alias`: Alias del agente.
- `agent_mode`: Modo de trabajo del agente (`0`: Normal mode, `1`: Learning mode, `2`: Autodisable mode) .
- `secondary_groups`: Grupos secundarios añadidos al agente.
- `custom_id`: ID personalizado del agente.
- `url_address`: URL de acceso al agente.

Ejemplo de cabecera XML:

```
<agent_data description= group= os_name='linux' os_version='Ubuntu 10.10'
interval='30' version='3.2(Build 101227)' timestamp='2011/04/20 12:24:03'
agent_name='foo' timezone_offset='0' parent_agent_name='too'
address='192.168.1.51' custom_id='BS4884'
url_address='http://mylocalhost:8080'>
```

Se necesita un elemento `module` por cada módulo y los siguientes items definen cada módulo:

- name: Nombre del módulo.
- description: Descripción del módulo.
- tags: Etiquetas asociadas al módulo.
- type: Tipo del módulo (debe existir en la base de datos de Pandora FMS).
- data: Dato del módulo.
- max: Valor máximo del módulo.
- min: Valor mínimo del módulo.
- post_process: Valor de postprocesado.
- module_interval: Intervalo del módulo (intervalo en segundos / intervalo del agente).
- min_critical: Valor mínimo para el estado crítico.
- max_critical: Valor máximo para el estado crítico.

- min_warning: Valor mínimo para el estado de alerta.
- max_warning: Valor máximo para el estado de alerta.
- disabled: Deshabilita (0) o habilita el módulo. Los módulos deshabilitados no se procesan.
- min_ff_event: Umbral de Flip-Flop.
- status: Estado del módulo (NORMAL, WARNING or CRITICAL). Los límites de los estados crítico y de alerta se ignoran si se especifica el estado.
- datalist: Envía el dato del módulo en formato datalist (una entrada en base de datos por cada uno de los valores recibidos) [0/1].
- unit: Unidad del módulo. Soporta la macro `_timeticks_` para transformar un dato en formato *timeticks* a `dd/hh/mm/ss`.
- timestamp: Establece un estampado de fecha y hora en el dato recibido del módulo.
- module_group: Grupo de módulos al que será añadido el módulo.
- custom_id: Custom ID del módulo.
- str_warning: Umbral de warning para módulos cadena (*string*).
- str_critical: Umbral de crítico para módulos cadena (*string*).
- critical_instructions: Critical instructions del módulo.

- warning_instructions: Warning instructions del módulo.
- unknown_instructions: Unknown instructions del módulo.
- critical_inverse: Activa el Inverse interval en el umbral crítico del módulo. [0/1].
- warning_inverse: Activa el Inverse interval en el umbral warning del módulo. [0/1].
- quiet: Activa el modo Quiet del módulo [0/1].
- module_ff_interval: Especifica un valor de FF Interval del módulo.
- alert_template: Asocia una plantilla de alerta al módulo.
- crontab: Especifica un crontab en el módulo.
- min_ff_event_normal: Valor de FF threshold en el cambio de estado a NORMAL.
- min_ff_event_warning: Valor de FF threshold en el cambio de estado a WARNING.
- min_ff_event_critical: Valor de FF threshold en el cambio de estado a CRITICAL.
- ff_timeout: Valor de *FlipFlop* timeout.
- each_ff: Habilita la opción “Change each status”.
- module_parent: Nombre del módulo en el mismo agente que será el padre de este módulo.
- ff_type: Activa el Keep counters del FF threshold. [0/1].
- min_warning_forced: Fuerza min_warning al nuevo valor indicado aunque el módulo exista, tiene prioridad sobre min_warning.
- max_warning_forced: Fuerza max_warning al nuevo valor indicado aunque el módulo exista, tiene prioridad sobre max_warning.
- min_critical_forced: Fuerza min_critical al nuevo valor indicado aunque el módulo exista, tiene prioridad sobre min_critical.
- max_critical_forced: Fuerza max_critical al nuevo valor indicado aunque el módulo exista, tiene prioridad sobre max_critical.
- str_warning_forced: Fuerza str_warning al nuevo valor indicado aunque el módulo exista, tiene prioridad sobre str_warning.
- str_critical_forced: Fuerza str_critical al nuevo valor indicado aunque el módulo exista, tiene prioridad sobre str_critical.

- Un módulo deberá tener al menos un elemento name, type y data.

Por ejemplo:

```
<module>
<name>CPU</name>
<description>CPU usage percentage</description>
<type>generic_data</type>
<data>21</data>
</module>
```

Puede haber cualquier número de elementos en un fichero de datos XML.

¡Recuerde cerrar la etiqueta `agent_data`!

También existen módulos de tipo lista, que en vez de tener solo un valor pueden contener varios, estos son útiles para valores de tipo *string*. Se tendría que usar la etiqueta datalist en este tipo de módulos tal como vemos en el siguiente ejemplo:

```
<module>
<type>async_string</type>
<datalist>
<data><value><![CDATA[xxxxx]]></value></data>
<data><value><![CDATA[yyyyy]]></value></data>

<data><value><![CDATA[zzzzz]]></value></data>
</datalist>
</module>
Se puede especificar una marca de tiempo para cada valor:
<module>
<type>async_string</type>
<datalist>
<data>
<value><![CDATA[xxxxx]]></value>
<timestamp>1970-01-01 00:00:00</timestamp>
</data>
<data>
<value><![CDATA[yyyyy]]></value>
<timestamp>1970-01-01 00:00:01</timestamp>
</data>
<data>
<value><![CDATA[zzzzz]]></value>
<timestamp>1970-01-01 00:00:02</timestamp>
</data>
</datalist>
</module>
```

Algunos ejemplos más que incluyen el uso de umbrales y de unidades:

```
<module>
<name><![CDATA[Cache mem free]]></name>
<description><![CDATA[Free cache memory in MB]]></description>
```

```
<tags>tag</tags>
<type>generic_data</type>
<module_interval>1</module_interval>
<min_critical>100</min_critical>
<max_critical>499</max_critical>
<min_warning>500</min_warning>
<max_warning>600</max_warning>
<unit><![CDATA[MB]]></unit>
<data><![CDATA[3866]]></data>
</module>
<module>
<name><![CDATA[Load Average]]></name>
<description><![CDATA[Average process in CPU (Last minute)
]]></description>
<tags>tag</tags>
<type>generic_data</type>
<module_interval>1</module_interval>
<data><![CDATA[1.89]]></data>
</module>
```

Plugintools

Pandora FMS dispone de una herramienta llamada *plugintools*, disponible para los lenguajes de programación Perl y Python que facilita el desarrollo de los *plugins*, ya que tiene incorporadas funciones que facilitan la mayoría de procesos necesarios en la creación de un *plugin*:

- Creación de agentes.
- Creación de módulos.
- Mandar archivos por Tentacle.
- Leer archivos de configuración.

Plugintools versión Python

https://pypi.org/project/pandoraPlugintools/

Plugintools versión Perl

https://github.com/pandorafms/pandorafms/blob/develop/pandora_server/lib/PandoraFMS/PluginTools.pm

Parámetros y ficheros de configuración

Una utilidad a considerar que suele ser bastante útil a la hora de interactuar con un plugin son los parámetros y los ficheros de configuración, ya que estos dan la opción de poder personalizar la

ejecución del plugin dando la posibilidad de crear ejecuciones diferentes, con estos se pueden definir datos que siempre van a ser variables como la ip o las credenciales del servicio a apuntar o poder personalizar el nombre de los módulos o agentes entre otras cosas.
El uso de parámetros para la configuración del plugin tiene ventajas y desventajas respecto al uso de ficheros de configuración, ya que evitas usar más archivos para el uso del plugin, pero la configuración del plugin seguramente se pueda realizar más rápido con un archivo de configuración, es importante evaluar qué opción es mejor para cada plugin.

Ayuda en el plugin

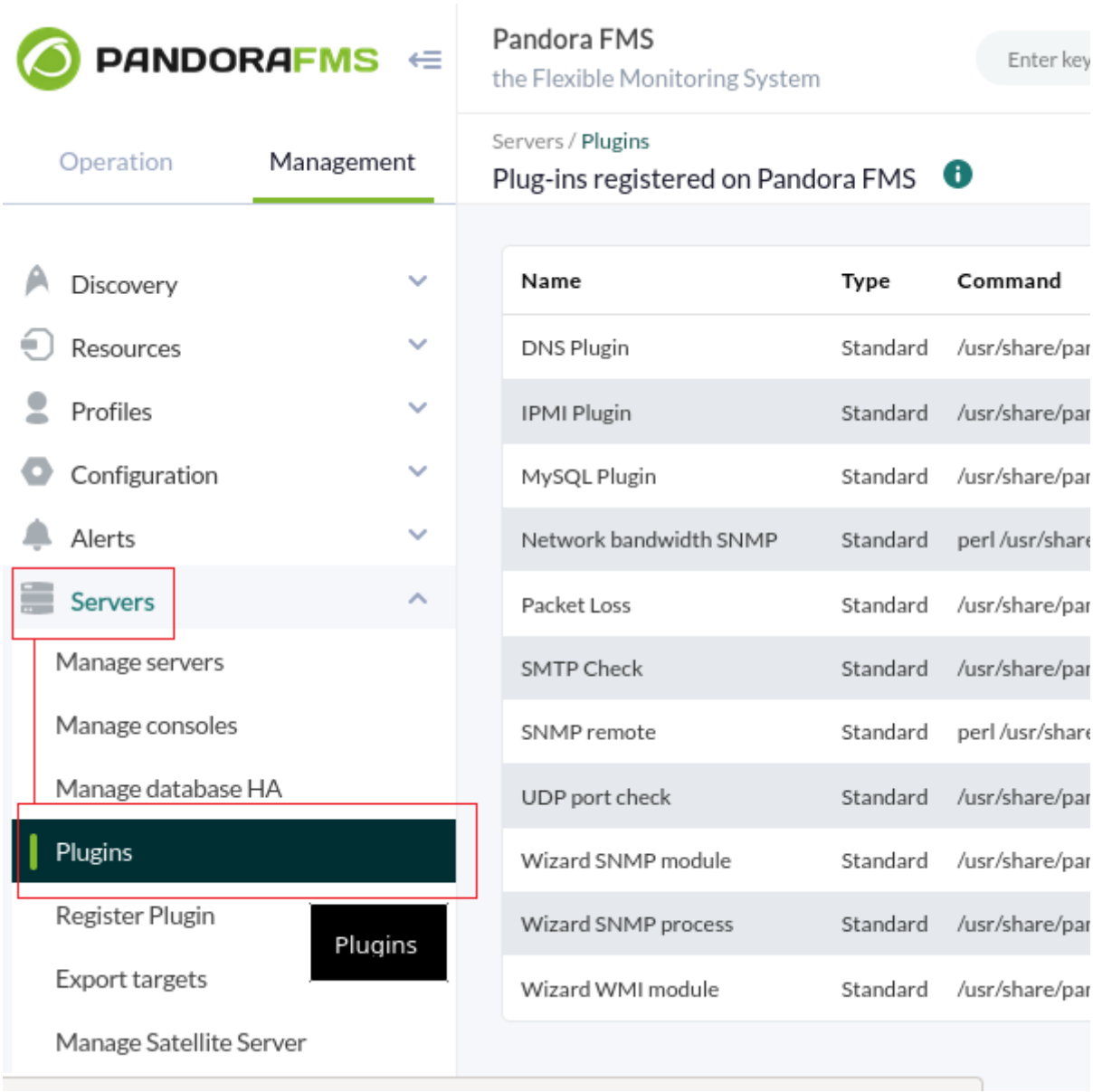
Otra opción que suele ser bastante útil en un *plugin*, es incluir un parámetro como opción de ayuda en este, para mostrar todas las opciones, parámetros del *plugin* y como poder configurar, es como una pequeña documentación digital sobre el *plugin* incluida en el software de este.

Configuración del plugin en Pandora FMS

La configuración del *plugin* puede variar dependiendo de si este es un *plugin* de agente o de servidor.

Plugins de servidor

Para poder monitorizar desde Pandora FMS con un *plugin* de agente de servidor se debe ir al apartado “plugins” dentro del menú de servidores; caso de ejemplo con el *plugin* de IMAP:



Se hace clic en “añadir”. Se le debe poner el nombre y la descripción que se prefiera.

Como comando se le debe introducir la ejecución con la ruta del *plugin*.

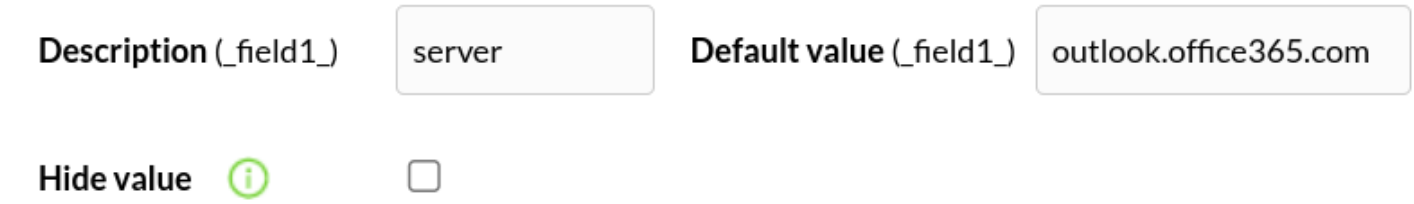
La ruta recomendada para el uso de los plugins de servidor es:

```
/usr/share/pandora_server/util/plugin/
```

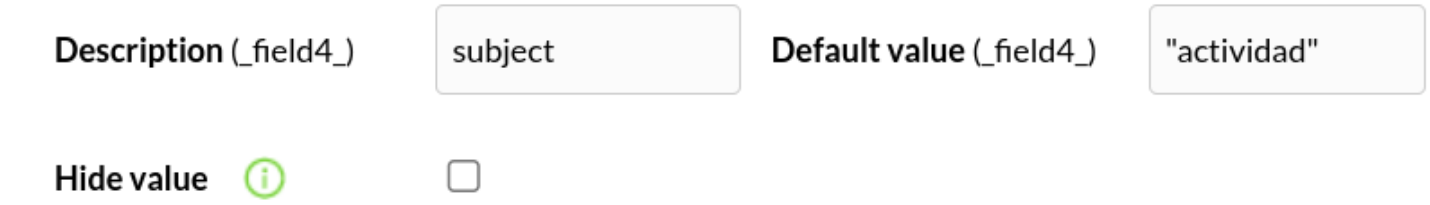
Se debe introducir como comando la ejecución con la ruta del plugin:

En parámetros del *plugin* se deben introducir estos seguidos de la macro `_field_`.

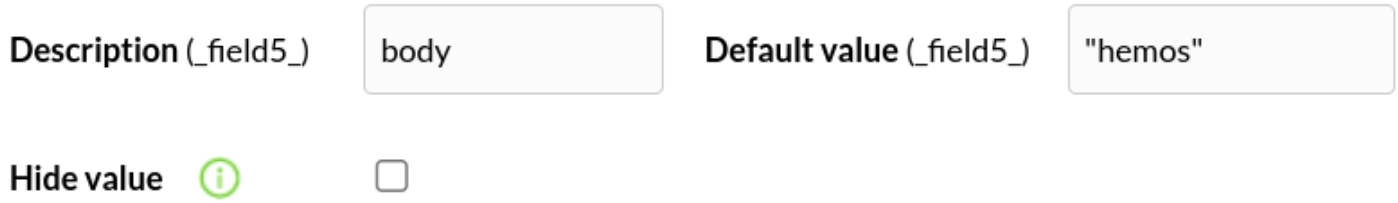
–SERVER



–SUBJECT



–BODY



Una vez hecho esto, se debe pulsa en “crear”.

Finalizados los anteriores pasos, solo quedaría llamar al plugin por lo que se debe ir a la vista de algún agente y crear un módulo de complementos. Se le debe poner un nombre y en el apartado *plugin* se añade el que acabamos de configurar.

Una vez hecho esto, se debe oprimir en “crear”.

Si el módulo se muestra con 1, quiere decir que se está ejecutando correctamente y se habrá creado un agente que contendrá los módulos.

Plugins de agente

Para poder monitorizar desde Pandora FMS con un plugin de agente, deberemos llamarlo desde el fichero `.conf` del agente software que está en la siguiente ruta, en GNU/Linux :

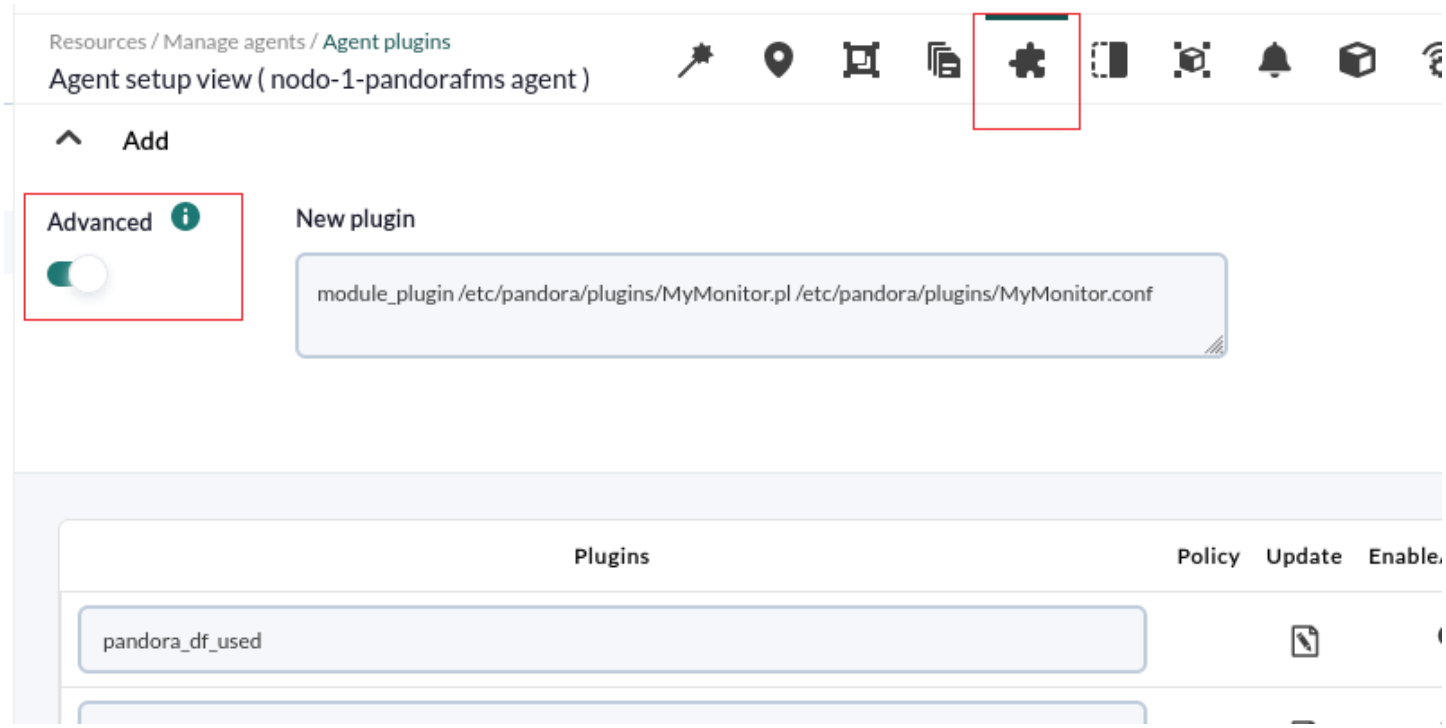
```
/etc/pandora/pandora_agent.conf
```

Se puede crear la ejecución en la última línea del `.conf` con el comando `module_plugin`, seguido del comando de ejecución.

Ejemplo de una posible ejecución de un *plugin*:

```
module_plugin /etc/pandora/plugins/MyMonitor.pl
/etc/pandora/plugins/MyMonitor.conf
```

Esto también se puede hacer desde la Consola web si está la configuración remota habilitada. Ejemplo de gestión de plugins avanzados de Agente Software desde la Consola:



Visualización del plugin en Pandora FMS

La visualización de los datos del *plugin* normalmente suele realizarse desde la vista de agentes. La estructura de datos suele estar definida por agentes y módulos por lo que lo normal es que el *plugin* cree agentes y módulos dentro de estos, por lo que estos datos pueden verse desde el menú de vista del agente en cuestión.

En muchos *plugins* se puede dar el caso de que creen muchos agentes, como puede ser por ejemplo el *plugin* de Xenserver que crea un agente por cada máquina virtual del servidor. Por suerte podemos definir un prefijo para los agentes, en este caso por ejemplo podría usarse el prefijo `xen-`, por lo que al visualizar los agentes, al ver que algunos tienen esa estructura en el nombre sabremos que esos agentes provienen de ese determinado *plugin*.

PSPZ2

Un archivo pspz2 es un comprimido en zip de dos archivos, el *plugin* y un fichero `plugin_definition.ini` que contiene la especificación del *plugin* y de los módulos. Este formato de archivo es muy útil a la hora de empaquetar *plugins*, ya que simplifica la instalación de estos.

Usando este formato, los *plugins* pueden instalarse de una manera más rápida en la consola de Pandora FMS desde el apartado “registro de plugins”.

Ejemplo de `plugins_definition.ini` de un *plugin*:

```
[plugin_definition]
name = Pandora Azure Storage
description = "Take data from azure storage"
timeout = 20
filename = pandora_azure
execution_command =
execution_postcommand =
parameters = -c _field1_ –agent_name _field2_ -g _field3_
plugin_type = 0
total_modules_provided = 0
total_macros_provided = 8
[macro_1]
hide = 0
description = "Conf path (obligatory)"
help = "Path conf"
value =
[macro_2]
hide = 0
description = "Agent name (optional)"
help = "Name of the agent"
value = _agentname_
[macro_3]
hide = 0
description = "group (optional)"
help = "Pandora FMS Target Group "
value =
```

filename: Debería tener el mismo nombre que el *script* incluido en el fichero .pspz, nombrado antes como < script_file >. En este ejemplo es un *shell script* (formato `.sh`) llamado `ssh_pandoraplugin.sh`.

plugin_type: `0` para un *plugin* estándar de Pandora FMS, y `1` para un plugin tipo Nagios.

total_modules_provided: Especifica cuántos módulos se definen en las secciones siguientes del fichero `.ini`. Debe establecer uno como mínimo (para usar en un ejemplo como mínimo).

execution_command: Si se emplea, hay que ponerlo delante del *script*. Podrá ser un intérprete, como por ejemplo `java -jar`. Así pues, el *plugin* se llamará a ejecución, desde el Plugin Server de Pandora FMS, con el siguiente código: `java -jar < plugin_path/ plugin_filename`.

execution_postcommand: Si se emplea, define los parámetros adicionales transmitidos al *plugin* después del < plugin_filename >, que es invisible para el usuario.

total_macros_provided: define el número de macros dinámicas que tiene el *plugin*.

macro__value: que define el valor para ese Módulo usando esa macro dinámica, si noexiste se toma el valor por defecto.

Y debe crear una sección por cada macro dinámica, por ejemplo:

```
[macro_<N>]
hide = 0
description = <your_description>
help = <text_help>
value = <your_value>
```

Compatibilidad

Es importante que el *plugin* tenga la mayor compatibilidad posible. Determinados *plugins* son exclusivos de servicios que solo se pueden explotar en un sistema operativo determinado, como por ejemplo, MS Windows®. Pero dentro de las limitaciones inevitables es importante dar al *plugin* una compatibilidad que abarque el mayor número de escenarios posible.

Determinados sistemas operativos tienen paquetes que no existen en otros lo que puede llevar a errores en la ejecución de un *plugin* compilado, por lo que es recomendable compilar los *plugins* en una máquina con sistema operativo centOS7 que cuente con todas las dependencias necesarias del *plugin* instaladas.

Control de errores

Tener un buen control de errores puede ayudar al mantenimiento del *plugin* a través del tiempo, ya que puede ayudar a detectar posibles futuros errores en el plugin y ubicar más rápidamente.

También puede ayudar a detectar cual es el error con exactitud, ya que sin control de errores en determinados casos puede que la salida del error sea demasiado genérica y sea difícil averiguar cuál es el problema.

Pruebas y validación del plugin

Una vez desarrollado el *plugin*, es adecuado realizar pruebas en entornos para comprobar el funcionamiento del mismo, de hecho sería genial el poder probarlo en más de un entorno de pruebas.

Un buen proceso de comprobación puede ayudar a solventar posibles errores y a hacer más sólido el funcionamiento del *plugin* en el tiempo, como se suele decir, mejor prevenir que curar.

Lanzamiento de un plugin

A la hora de publicar el *plugin*, es importante tener en cuenta ciertos aspectos.

Documentación del plugin

El uso de algunos *plugins*, de primeras, puede ser difícil de entender, puede que algunos tengan varias opciones y pueda ser algo tediosa su comprensión.

Una buena documentación puede ayudar a facilitar su entendimiento, incluso puede ser útil para el propio creador del *plugin* en un futuro, para ayudarle a recordar ciertas cosas del *plugin* que con el tiempo puede haber olvidado.

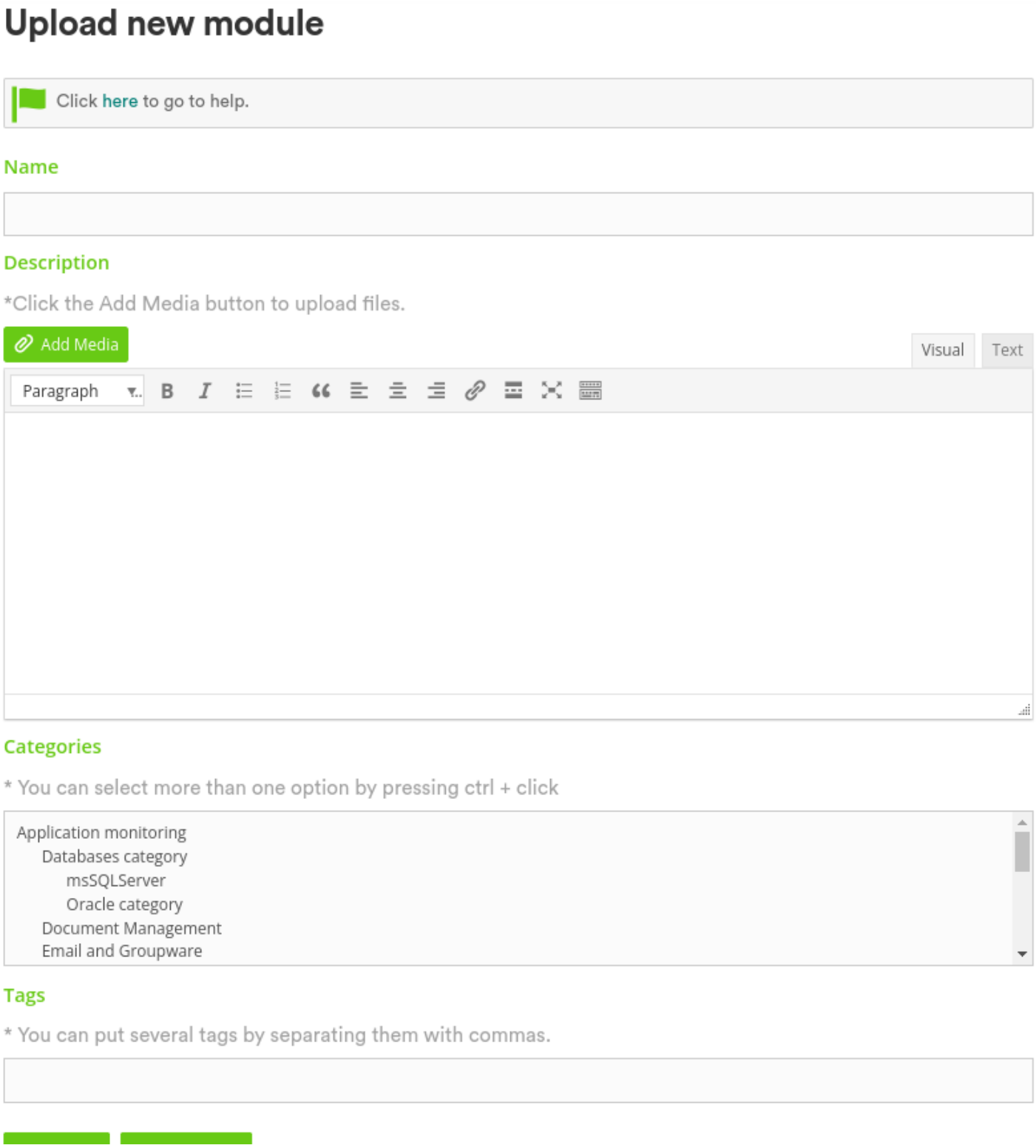
Pandora FMS cuenta con un sistema de guías rápidas *online* que facilita el mantenimiento de la documentación, al que se puede acceder desde el siguiente enlace:

https://pandorafms.com/guides/public/shelves/es-guias-rapidas

La alternativa a la documentación en formato de guías rápidas es el formato PDF.

Subir un plugin a la biblioteca

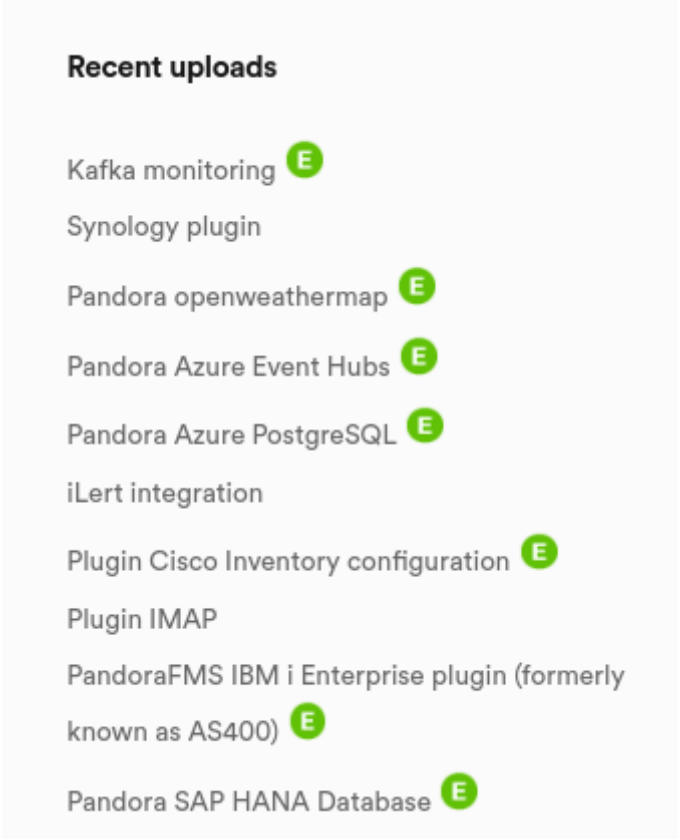
El proceso a la hora de subir un *plugin* a la biblioteca de Pandora FMS es simple. Tan solo se debe acceder al apartado “upload new module” dentro de la sección “my plugins” de la biblioteca.



Una vez configurados todos los apartados de la entrada, como son el título, una descripción del *plugin*, el *plugin* comprimido en formato zip, la documentación, categorías y *tags*, se puede publicar el plugin pinchando en el apartado submit.

Todos los plugins subidos a la biblioteca son revisados y necesitan un tiempo de revisión, antes de su publicación definitiva.

Una vez el plugin sea publicado aparecerá en el apartado de “ultimas subidas”:



Buenas prácticas para ayudar a la revisión y mantenimiento de un plugin

Algunas buenas prácticas pueden ayudar a disminuir el tiempo de dedicación en las tareas, a prevenir y sortear errores comunes durante las diferentes etapas de realización del *plugin* y al mantenimiento del *plugin* con el tiempo, y facilitar su revisión. Estas pueden ser:

- Dejar bien documentada una buena configuración previa en caso de que fuera necesario realizar algunas configuraciones previas en el sistema en el *plugin* en cuestión.
- Documentar un ejemplo de ejecución real del *plugin* que ayude a visualizar al usuario el uso del mismo.
- Escribir comentarios en el código del *plugin* puede ayudar a entender mejor o más rápido este.
- Prioriza la legibilidad del código. Cuanto más complejo sea, más tiempo y recursos serán necesarios para tratarlo.
- Probar todo el código. Es importante probar el funcionamiento del *plugin* para comprobar que todo está bien. Encontrar un error a tiempo y solucionarlo evitará problemas en el futuro.