Guide de développement du plug-in

Guide de développement du plug-in

Introduction

Ce guide de développement des *plugins* contient de la documentation pour les utilisateurs qui souhaitent créer leurs propres *plugins*. Utilisez cette documentation pour apprendre à créer des *plugins* qui répondent aux besoins de votre entreprise.

Qu'est-ce qu'un plugin?

Un plug-in, également connu sous le nom de *plugin*, est une application qui permet d'étendre les fonctions d'une autre application, en l'occurrence Pandora FMS.

Pourquoi les plugins sont-ils utiles?

Les *plugins* permettent d'étendre les fonctionnalités de Pandora FMS, ce qui offre une grande variété de possibilités lors du développement de nouvelles options de supervision.

Ceux-ci peuvent être intégrés avec la possibilité de superviser les services, les applications, les bases de données et bien plus encore, et peuvent même être utilisés pour modifier ou automatiser les tâches et les processus.

L'un des objectifs est de pouvoir superviser autant de systèmes et de services que possible dans le même environnement de travail et l'utilisation de *plugins* aide beaucoup dans cette tâche.

L'utilité de la plupart des *plugins* est de pouvoir afficher dans Pandora FMS les données ou les statistiques de performance qui sont collectées à partir de services externes.

Types de plugins dans le Pandora FMS

Par type d'exécution :

- *Plugin* de l'agent : Les *plugins* de l'agent sont exécutés par l'agent du logiciel Pandora FMS, ils sont généralement locaux et ne fonctionnent pas à distance, l'exécution sera donc effectuée par le *daemon* de l'agent. L'agent *plugins* dans son exécution imprime habituellement dans un XML les modules avec les données.
- *Plugin* du serveur: Les Server *plugins* sont exécutés par le Serveur d'extension, ils obtiennent généralement les données à distance, soit par l'API, soit par une autre méthode, de sorte que, bien qu'ils puissent être configurés en tant qu'agent *plugin* (la plupart du temps), l'option recommandée est de les configurer en tant que server *plugin*. Les plugins serveur *plugin* affichent une seule valeur,

par exemple 1 pour indiquer une exécution réussie.

Comment les utilisateurs trouvent-ils les plugins?

Pandora FMS dispose d'une bibliothèque à partir de laquelle vous pouvez télécharger tous les *plugins* créés pour le système.

Tout le monde peut télécharger des *plugins* dans la bibliothèque, bien qu'il y ait un processus de révision avant qu'ils ne soient acceptés et publiés.

https://pandorafms.com/library/

Actuellement, la bibliothèque dispose d'un grand nombre de *plugins* couvrant tous les domaines, bases de données, applications, *cloud*, services de messagerie...

La bibliothèque dispose d'un moteur de recherche qui vous permet de rechercher n'importe quel plugin téléchargé, ainsi que de différents *tags* et menus qui vous permettent de naviguer confortablement dans la bibliothèque.

Chaque *plugin* est généralement accompagné d'une documentation détaillant l'installation, la configuration du *plugin* et les données qu'il collecte.

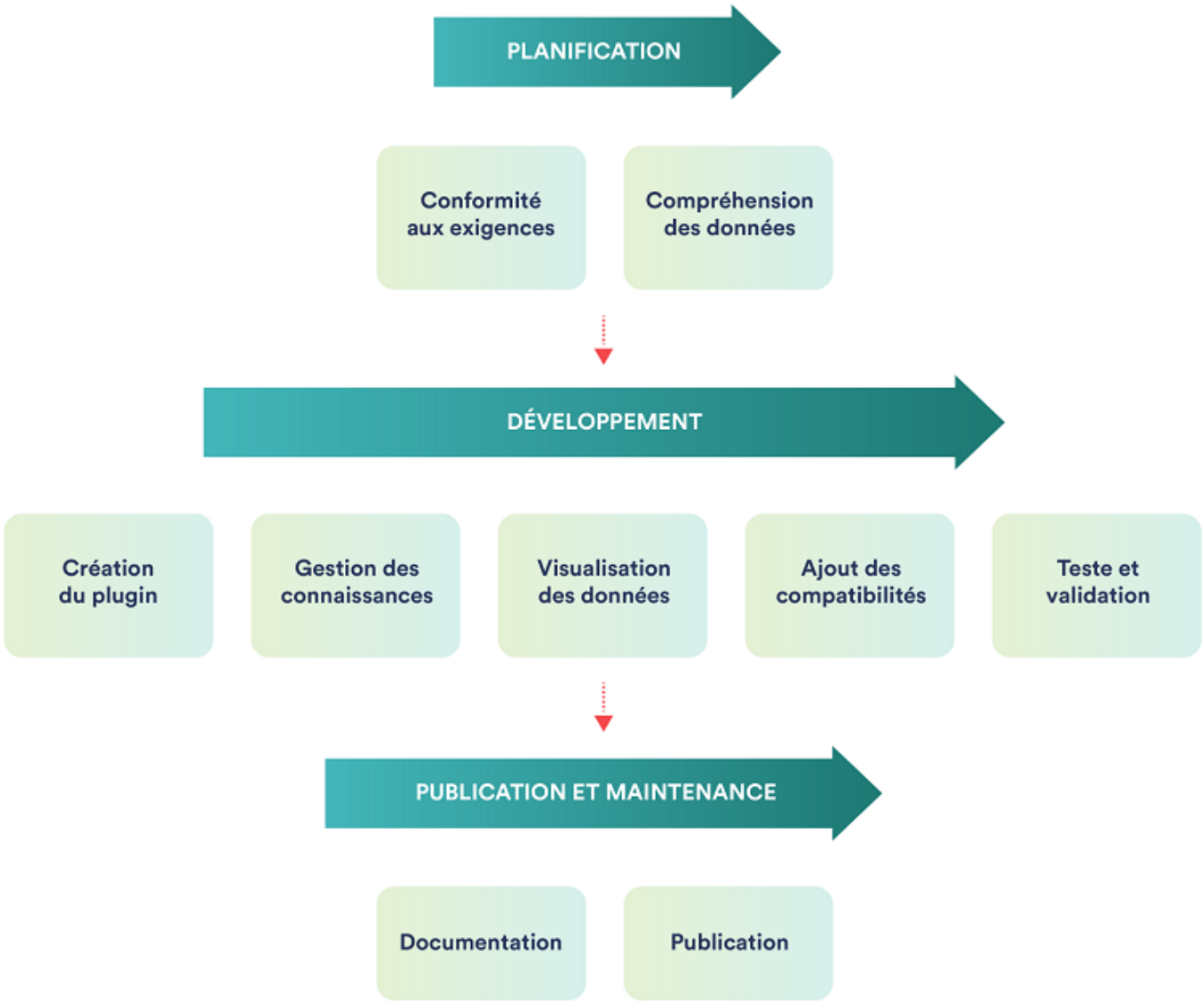
Comment installer des plugins ?

Les *plugins* peuvent être installés à partir de la console de Pandora FMS. En fonction de leur type, ils peuvent être exécutés de deux manières différentes, ce qui créera différentes manières de configurer les *plugins*. Les *plugins* de l'agent sont exécutés par l'agent logiciel, tandis que les *plugins* du serveur sont exécutés par le serveur de plugins.

En général, les *plugins* de serveur créent des agents et des modules, tandis que les *plugins* d'agent ne créent des modules qu'à l'intérieur de l'agent logiciel où ils ont été configurés.

L'installation du *plugin* consiste à créer une exécution personnalisée de celui-ci dans Pandora FMS. A chaque intervalle de temps configurable, le *plugin* sera exécuté et affichera les modules mis à jour.

Cycle de vie d'un plugin



Planification d'un plug-in

Lors du développement d'un *plugin*, il est important de bien planifier son développement en tenant compte de certains aspects.

Quel problème le plugin résout-il ?

Il devrait être possible de répondre à ces questions dans une première étape de la planification du *plugin* :

- Quel est l'objectif de *plugin* ?
- Quels sont les cas d'utilisation auxquels votre *plugin* répondra ?

Il est important d'avoir une idée claire de la solution que vous souhaitez obtenir avec le *plugin*. Il peut s'agir de superviser un service d'intérêt, en parvenant à unifier toutes les données dans Pandora FMS, ce qui peut représenter un avantage en termes de temps ou de coûts, ou

d'automatiser un processus, comme un *plugin* qui lit les courriels et qui permet d'activer une alarme à l'arrivée d'un certain message.

Les *plugins* visent à répondre à un besoin ou à faciliter l'interaction, le traitement et la visualisation des données.

Comment le plugin va-t-il extraire les données ?

L'utilité principale d'un *plugin* est d'extraire des données d'un service, d'une application, d'une base de données et de pouvoir visualiser ces données dans Pandora FMS... Mais comment pouvons-nous extraire les données du service requis ?

Cela dépend sans aucun doute de la technologie ou du service à exploiter avec le *plugin*, de sorte qu'une recherche préalable sur le service choisi est nécessaire pour déterminer la faisabilité du *plugin* et, si possible, la création d'un environnement de test pour l'évaluer.

En général, ces services disposent d'un moyen d'obtenir les statistiques de supervision. Ce moyen varie en fonction du service, le plus courant étant qu'ils disposent d'une API ou d'une CLI permettant de traiter les données requises et de les afficher à l'aide de l'extension *plugin*.

Dans certains cas, il existe des bibliothèques que la communauté ou l'entreprise qui crée la technologie ou le service en question a créées pour faciliter l'interaction avec les données de leurs services.

Les utilisateurs ont-ils besoin d'une autorisation pour obtenir les données ?

Il est probable que pour extraire des données d'un service ou d'une technologie, il faille disposer d'un compte qui nécessite certaines autorisations pour interagir avec les données, ou que des configurations préalables doivent être effectuées dans l'environnement avant de pouvoir extraire ces données.

Ces exigences sont généralement spécifiées dans la documentation du service ou de la technologie elle-même.

Il est important de savoir clairement ce qui est nécessaire et quelles sont les exigences pour pouvoir extraire les données de ce service et de les détailler dans la documentation du *plugin*.

Les données sont-elles collectées à distance ou localement ?

Il se peut que les statistiques de performance puissent être extraites à distance par API, soit par une bibliothèque, soit par CLI, soit par API avec des appels HTTP. Il se peut aussi que ces données ne puissent être extraites qu'en interne ou localement par des commandes ou parce qu'une

technologie propriétaire est utilisée. Selon le cas, l'exécution de *plugin* changera et sera différente.

Un *plugin* qui peut obtenir les données à distance peut être exécuté avec le serveur de plugin, car il n'a pas besoin d'être présent sur la machine de service pour obtenir les données.

Certains *plugins* ne peuvent être exécutés que par l'intermédiaire de l'agent logiciel, car ils doivent se trouver sur le même système que le service à partir duquel les données doivent être obtenues.

Comment voulez-vous visualiser les données ?

Lors de la définition de l'affichage des données *plugin* dans la console de Pandora FMS, il est important de préciser comment vous souhaitez afficher ces données.

Normalement, ces données seront visualisées en suivant une structure d'agents et de modules, il est donc important de bien définir cette structure afin de visualiser les informations de la manière la plus confortable possible et sans confusion, ce qui peut être difficile à réaliser dans les *plugins* qui traitent beaucoup de données ou qui supervisent de nombreuses « éléments », comme cela pourrait être le cas avec de nombreuses bases de données, machines virtuelles, etc.

Dans le *plugin*, vous pouvez inclure des options pour personnaliser le nom des agents ou des modules, ou ajouter une sorte de préfixe à leur nom pour les rendre plus visibles et les distinguer.

Exigences et dépendances du plugin

Selon les technologies utilisées dans le *plugin*, il peut être nécessaire d'installer des dépendances pour qu'il fonctionne correctement. Il peut s'agir de bibliothèques du langage utilisé qui ont été importées dans le plugin pour exécuter une fonction ou du langage utilisé lui-même.

Si le *plugin* est écrit en langage Python version 3, il sera nécessaire d'avoir `python3` installé sur la machine sur laquelle vous voulez utiliser le *plugin* ce qui peut être un gros obstacle pour les utilisateurs potentiels de ce *plugin*.

Une option pour les *plugins* créés en langage Python est d'ajouter un fichier appelé `requirements.txt` au *plugin* qui peut inclure toutes les dépendances utilisées par le *plugin*, de sorte que l'installation de ce fichier installera automatiquement toutes les dépendances nécessaires.

Exemple de fichier `requirements.txt` d'un *plugin*, en particulier celui de MongoDB :

```
dnspython==2.1.0
```

```
pymongo==3.12.0
```

Le fichier comprend deux bibliothèques que vous devez avoir installées sur votre système pour pouvoir le lancer (au cas où il ne s'agirait pas d'un *plugin* compilé).

Le fichier peut être écrit à la main, mais vous pouvez aussi générer un fichier `requirements.txt` comprenant toutes les dépendances installées de l'environnement virtuel avec la commande `pip freeze`.

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

Vous pouvez copier le fichier `requeriments.txt` dans l'environnement où vous souhaitez installer les dépendances et lancer la commande suivante pour les installer :

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

Compilation de plugins

Une façon de résoudre les problèmes de dépendance avec les *plugins* est de créer des exécutables qui sont livrés avec toutes les dépendances déjà installées. C'est une solution idéale pour les *plugins* écrits en Perl et les *plugins* écrits en Python.

Dans le cas de Python, pour créer les exécutables, vous disposez de la bibliothèque `pyinstaller` :

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

Il peut être installé à l'aide de la commande suivante :

```
pip install pyinstaller
```

Une fois installé, la commande suivante peut être utilisée pour créer un binaire du plugin en question, dans un fichier :

Linux® :

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

MS Windows® :

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

Un dossier appelé `dist` sera généré avec le binaire à l'intérieur.

Il peut arriver qu'un *plugin* compilé sur un système d'exploitation ne fonctionne pas sur un autre. Il est recommandé de compiler les *plugins* sur Rocky Linux 7, car les *plugins* compilés sur ce système fonctionnent généralement sur tous les autres systèmes d'exploitation Linux.

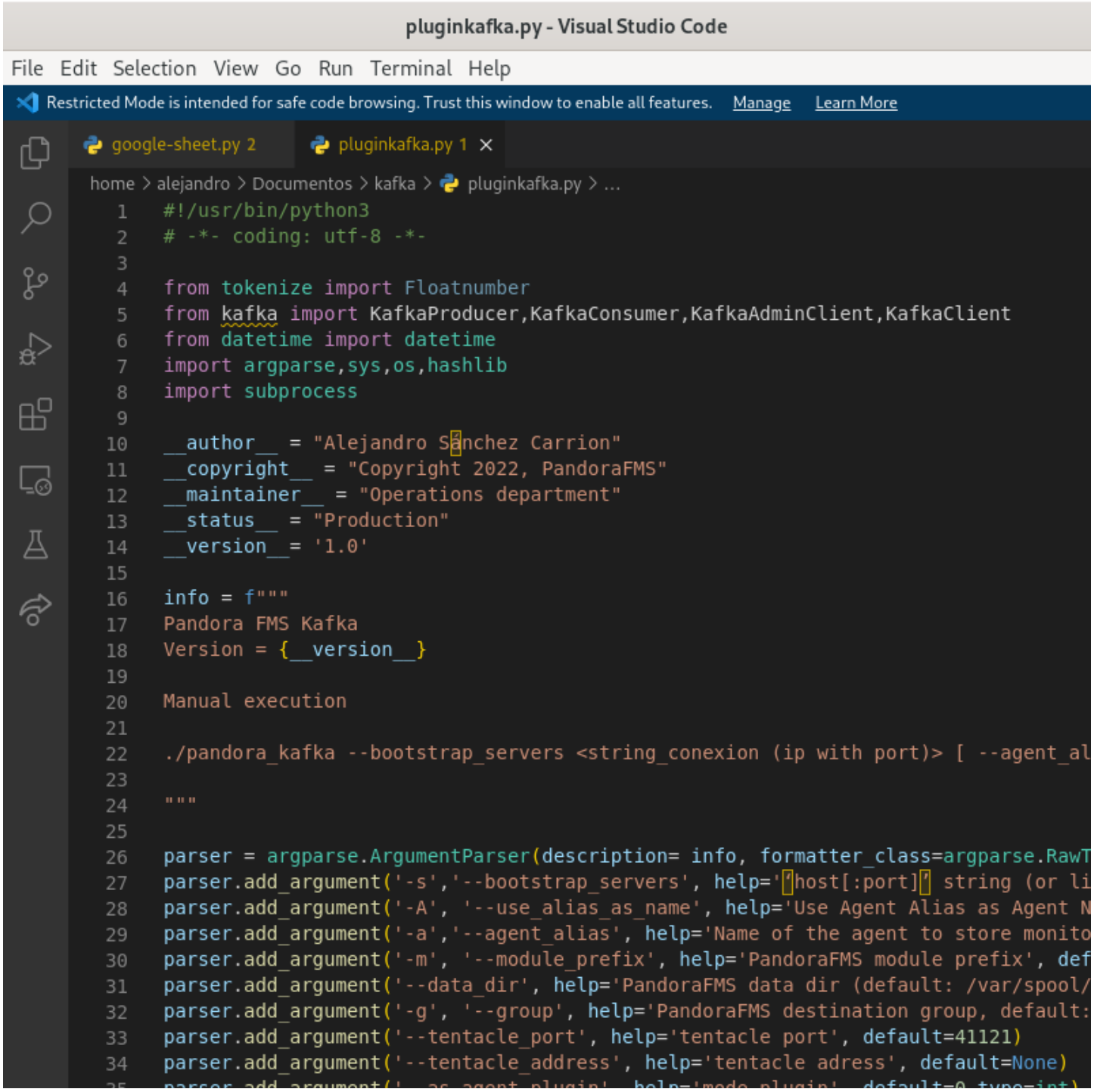
Des binaires créés sur Fedora et Ubuntu génèrent une erreur de dépendance sur d'autres systèmes.

Développement d'un plugin

Il est important d'être clair sur les outils et les technologies à utiliser dans le développement du *plugin*.

Outils de développement

Les *plugins* sont des *scripts* qui intègrent des données ou une fonctionnalité supplémentaire dans Pandora FMS, ce sont encore de petits programmes, donc pour les développer il est nécessaire d'avoir un *framework* ou un éditeur de code tel que Visual Studio Code.



Des outils tels que GitHub ou GitLab peuvent aider à préserver et à maintenir le code des *plugins* créés.

Langages de programmation

Pour le développement des *plugins* il est nécessaire d'utiliser un langage de programmation, Pandora FMS supporte n'importe quel type de langage, tant qu'il présente les données au format XML, Pandora FMS sera capable de les comprendre et de les afficher dans sa console. Pour cela, vous devez suivre une structure de balises avec le XML qui inclut les agents, les modules et leurs attributs.

En général, les langages de *scripting* les plus utilisés dans les *plugins* de Pandora FMS sont Python, Perl, BASH et PowerShell, les deux premiers étant probablement les plus complets.

Python et Perl disposent d'un outil développé par Pandora FMS qui facilite la création de *plugins* appelé Plugin Tools, cet outil dispose de nombreuses fonctions conçues pour faciliter et automatiser la création de *plugins*, il peut donc constituer un avantage décisif lors du choix du bon langage.

XML

Pour que le Pandora FMS puisse consommer les données qu'il reçoit de *plugin* afin de les afficher, il est nécessaire que ces données soient traduites au format XML.

Connaître le format XML des données de Pandora FMS peut vous aider à améliorer les *plugins*, à créer des agents personnalisés avec eux ou simplement à envoyer des fichiers XML personnalisés au serveur de données de Pandora FMS.

Comme tout document XML, le fichier de données doit commencer par une déclaration XML :

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

Cependant, bien que le fonctionnement des *plugins* et de leurs données soit basé sur XML, seul l'agent *plugins* imprimera un XML lors de l'exécution d'un terminal, le serveur *plugins* ne montrera qu'une simple donnée, comme un, qui sera la valeur qui contiendra le module qui l'active, alors le plus normal dans ces cas est qu'il émette un `1` s'il fonctionne et un `0` si au contraire il a donné une quelconque sorte d'erreur.

Agents et modules

Les données collectées par le *plugin* peuvent être visualisées dans Pandora FMS par le biais d'agents et de modules, il est donc important de schématiser la manière dont vous souhaitez présenter les données. Par exemple, chaque agent peut être une base de données à surveiller, avec des modules qui représentent les informations de celle-ci.

La Structure XML des agents et des modules possède différents attributs pour les configurer.

Agents

L'élément `agent_data`, qui définit l'agent qui envoie les données, prend en charge les attributs suivants:

- `description`: Description de l'agent.
- `group`: Nom du groupe auquel appartient l'agent (il doit exister dans la base de données de Pandora FMS). S'il est laissé vide et qu'aucun groupe n'est configuré par défaut dans le serveur, l'agent ne sera pas créé.
- `os_name`: Nom du système d'exploitation sur lequel l'agent fonctionne (doit exister dans la base de données de Pandora FMS).
- `os_version`: Chaîne libre décrivant la version du système d'exploitation.
- `interval`: Intervalle agent (en secondes).
- `version`: Chaîne avec la version de l'agent.
- `timestamp`: Horodatage indiquant quand le XML a été généré (`YYYY/MM/DD HH:MM:SS`).
- `agent_name`: Nom de l'agent.
- `timezone_offset`: Décalage ajouté à l'horodatage (en heures). Utile lorsque l'on travaille avec différents fuseaux horaires (UTC).
- `address`: Adresse IP de l'agent (ou FQN).
- `parent_agent_name`: Nom du père de l'agent.
- `agent_alias`: Alias de l'agent.
- `agent_mode`: Fonctionnement de l'agent (`0`: Mode normal, `1`: Mode d'apprentissage, `2`: Mode Auto disponible) .
- `secondary_groups`: Groupes secondaires ajoutés à l'agent.
- `custom_id`: Identifiant personnalisé de l'agent.
- `url_address`: URL d'accès à l'agent.

Exemple d'en-tête 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'>
```

Modules

Un élément `module` est nécessaire pour chaque module et les éléments suivants définissent chaque module:

- name: Nom du module.
- description: Description du module.
- tags: Tags associés au module.
- type: Type de module (doit exister dans la base de données de Pandora FMS).
- data: Données du module.
- max: Valeur maximale du module.

- min: Valeur minimale du module.
- post_process: Valeur de post-traitement.
- module_interval: Intervalle du module (intervalle en secondes / intervalle de l'agent).
- min_critical: Valeur minimale pour l'état critique.
- max_critical: Valeur maximale pour l'état critique.
- min_warning: Valeur minimale pour la vigilance.
- max_warning: Valeur maximale pour la vigilance.
- disabled: Désactive (0) ou active le module. Les modules désactivés ne sont pas traités.
- min_ff_event: Seuil Flip-Flop.
- status: Statut du module (NORMAL, WARNING ou CRITICAL). Les limites des états critique et d'alerte sont ignorées si l'état est spécifié.
- datalist: Envoie les données du module sous forme de datalist (une entrée dans la base de données pour chacune des valeurs reçues) [0/1].
- unit: Unité de module. Supporte la macro `_timeticks_` pour transformer une donnée au format *timeticks* en `dd/hh/mm/ss`.
- timestamp: Fixe un timestamp sur les données reçues du module (pour utiliser timestamp dans les modules, ils doivent être dans un bloc datalist, comme montré dans les exemples ci-dessous).
- module_group: Groupe de modules auquel le module sera ajouté.
- custom_id: ID personnalisé du module.
- str_warning: Seuil d'alerte pour les modules de chaînes (*string*).
- str_critical: Seuil de criticité pour les modules de chaînes (*string*).
- critical_instructions: Instructions critiques du module.

- warning_instructions: Instructions d'avertissement du module.
- unknown_instructions: Instructions inconnues du module.
- critical_inverse: Active l'intervalle inverse au seuil critique du module [0/1].
- warning_inverse: Active l'intervalle inverse au seuil d'alerte du module. [0/1].
- quiet: Active le mode silencieux du module [0/1].
- module_ff_interval: Spécifie une valeur pour l'intervalle FF du module.
- alert_template: Il associe un modèle d'alerte au module.
- crontab: Spécifie une crontab dans le module.
- min_ff_event_normal: Valeur du seuil FF au changement d'état a NORMAL.
- min_ff_event_warning: Valeur du seuil FF au changement d'état a WARNING.
- min_ff_event_critical: Valeur du seuil FF au changement d'état a CRITICAL.
- ff_timeout: Valeur de *FlipFlop* timeout.
- each_ff: Activez l'option “Modifier chaque statut”.
- module_parent: Nom du module du même agent qui sera le parent de ce module.
- ff_type: Activer les Keep counters du seuil FF [0/1].
- min_warning_forced: Force min_warning à la nouvelle valeur spécifiée même si le module existe, a priorité sur min_warning.
- max_warning_forced: Force max_warning à la nouvelle valeur spécifiée même si le module existe, a priorité sur max_warning.
- min_critical_forced: Force min_critical à la nouvelle valeur spécifiée même si le module existe, a priorité sur min_critical.
- max_critical_forced: Force max_critical à la nouvelle valeur spécifiée même si le module existe, a priorité sur max_critical.
- str_warning_forced: Force str_warning à la nouvelle valeur spécifiée même si le module existe, a priorité sur str_warning.
- str_critical_forced: Force str_critical à la nouvelle valeur spécifiée même si le module existe, a la priorité sur str_critical.

Éléments du module

Un module doit avoir au moins un nom d'élément, un type et des données, par exemple:

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

Il peut y avoir un nombre quelconque d'éléments dans un fichier de données XML.

¡N'oubliez pas de fermer la balise `agent_data`!

Modules de type liste

Il existe également des modules de type liste qui, au lieu d'avoir une seule valeur, peuvent en contenir plusieurs; ils sont utiles pour les valeurs de type *string*. La balise datalist doit être utilisée dans ce type de modules, comme le montre l'exemple suivant:

```
<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>
```

Horodatage des modules

Un horodatage peut être spécifié pour chaque valeur:

```
<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>
```

Seuils et unités

Quelques autres exemples, dont l'utilisation de seuils et d'unités:

```
<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 dispose d'un outil appelé *Plugintools*, disponible pour les langages de programmation Perl et Python, qui facilite le développement des *plugins*, car il a incorporé des fonctions qui facilitent la plupart des processus nécessaires à la création d'un *plugin* :

- Création d'agents.

- Création de modules.
- Envoyer des fichiers par Tentacle.
- Lire les fichiers de configuration.

Plugintools version Python

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

Plugintools version Perl

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

Paramètres et fichiers de configuration

Les paramètres et les fichiers de configuration sont un utilitaire à prendre en compte et qui est généralement très utile lors de l'interaction avec un plugin, car ils permettent de personnaliser l'exécution du plugin en donnant la possibilité de créer différentes exécutions, avec lesquelles vous pouvez définir des données qui seront toujours des variables telles que l'adresse IP ou les informations d'identification du service à pointer ou de personnaliser le nom des modules ou des agents, entre autres choses.

L'utilisation de paramètres pour la configuration du plugin présente des avantages et des inconvénients par rapport à l'utilisation de fichiers de configuration, car vous évitez d'utiliser plus de fichiers pour l'utilisation du plugin, mais la configuration du plugin peut probablement être effectuée plus rapidement avec un fichier de configuration, il est important d'évaluer quelle option est la meilleure pour chaque plugin.

Aide sur le plugin

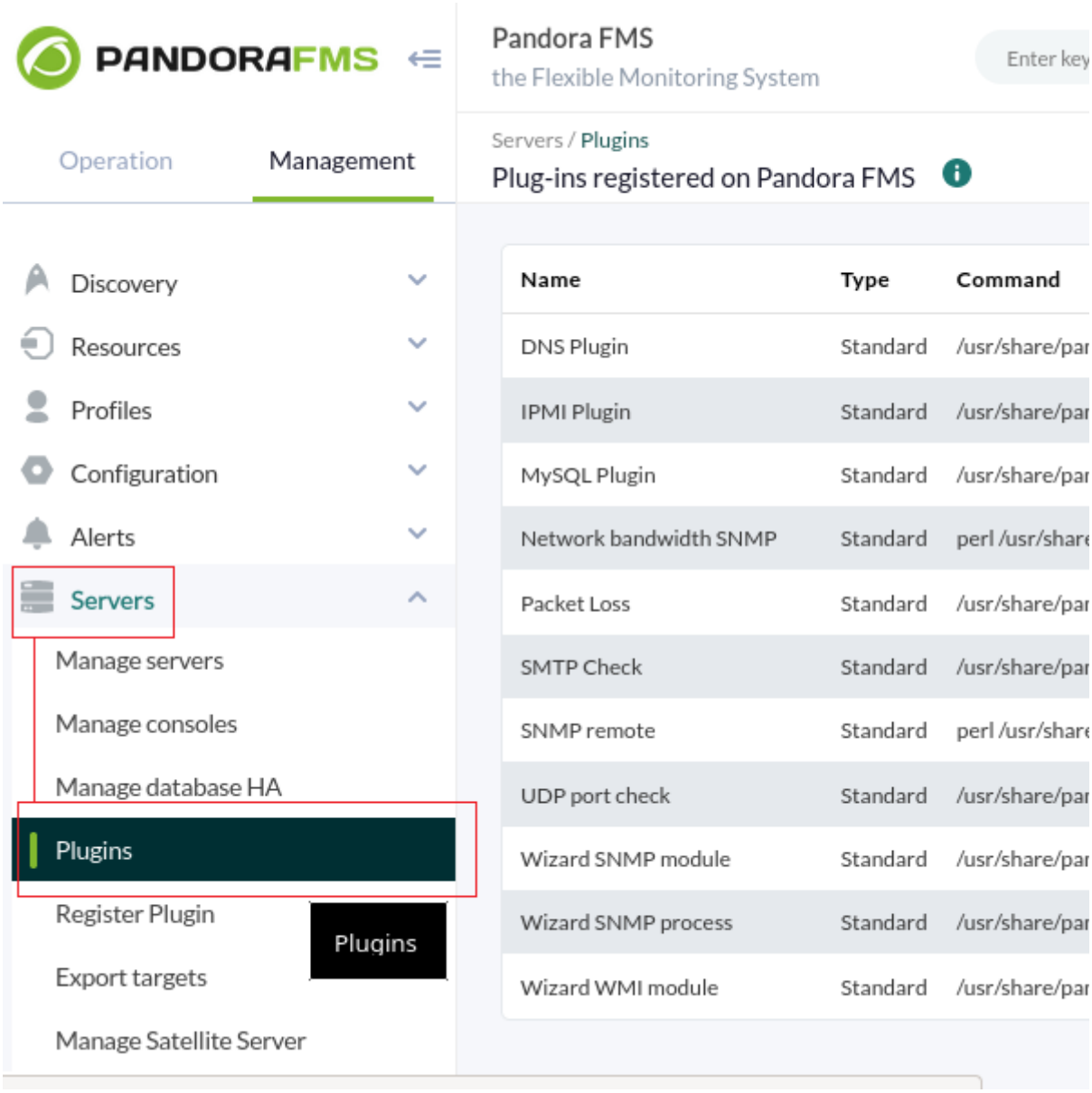
Une autre option qui est généralement très utile dans un *plugin*, est d'inclure un paramètre comme option d'aide dans celui-ci, pour montrer toutes les options, les paramètres du *plugin* et comment le configurer, c'est comme une petite documentation numérique sur le *plugin* inclus dans le logiciel du *plugin*.

Configuration du plugin dans Pandora FMS

La configuration du *plugin* peut varier selon qu'il s'agit d'un agent ou d'un serveur *plugin*.

Plugins serveur

Pour pouvoir surperviser depuis Pandora FMS avec un *plugin* d'agent serveur, vous devez aller dans la section « plugins » à l'intérieur du menu des serveurs ; exemple avec le *plugin* d'IMAP :



Cliquez sur « ajouter ». Vous devez lui donner le nom et la description de votre choix.

En tant que commande, l'exécution doit être entrée par le chemin *plugin*.

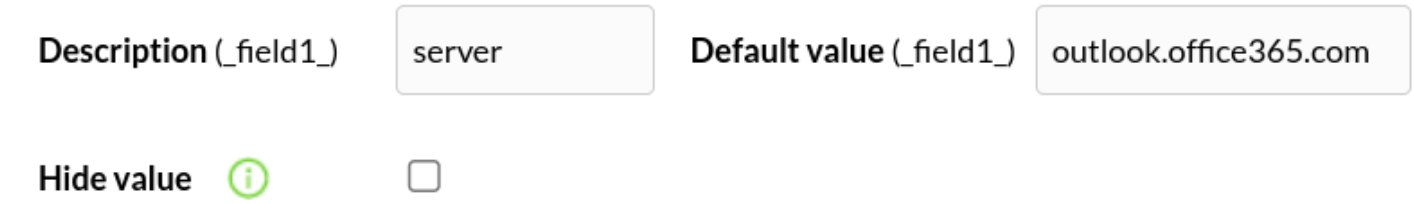
Le chemin recommandé pour l'utilisation des plugins de serveur est le suivant :

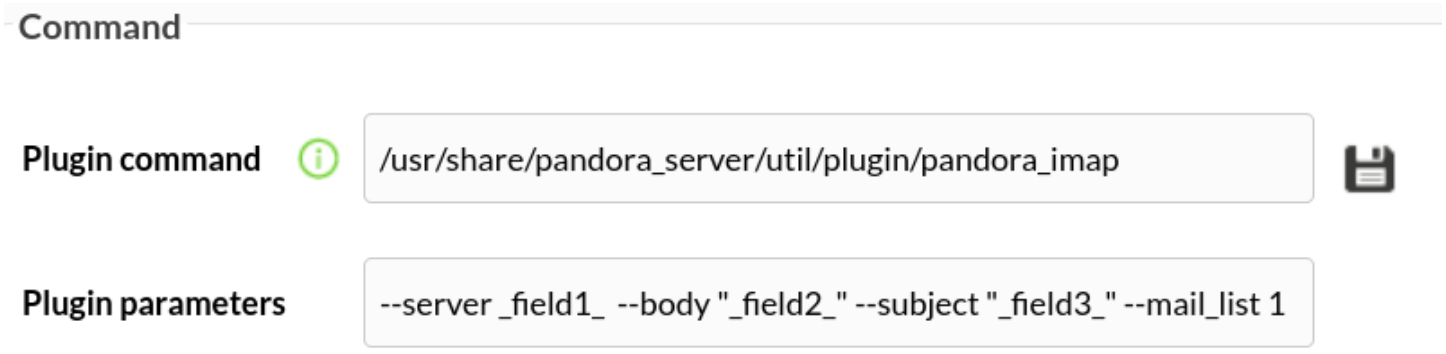
```
/usr/share/pandora_server/util/plugin/
```

L'exécution doit être saisie comme une commande avec le chemin d'accès *plugin* :

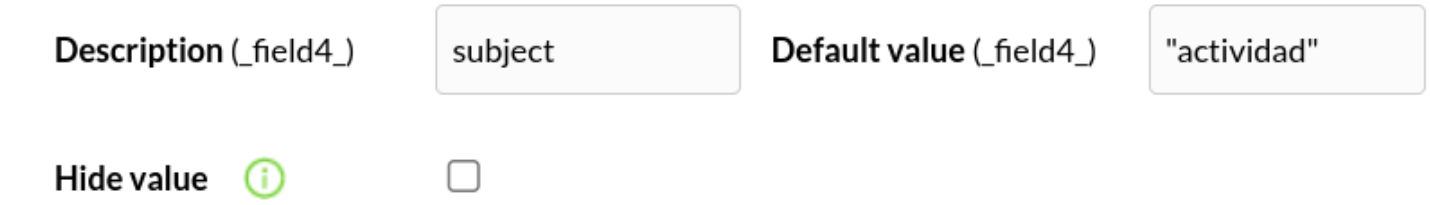
Dans les paramètres *plugin*, ils doivent être introduits suivis de la macro `_field_`.

–SERVER

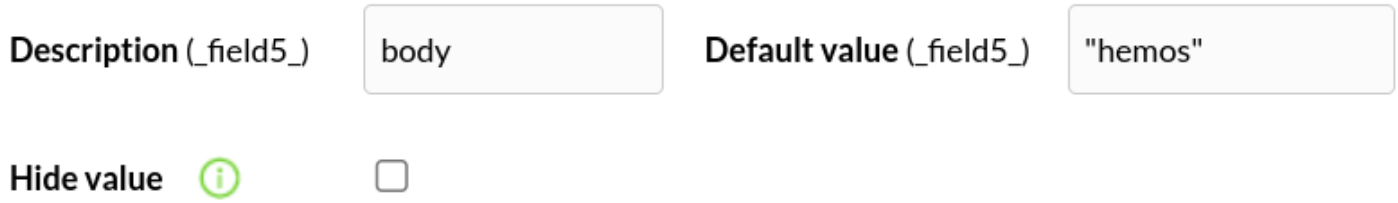




–SUBJECT



–BODY



Une fois cela fait, cliquez sur « créer ».

Une fois les étapes précédentes terminées, la seule chose qui reste à faire est d'appeler le plugin, vous devez donc vous rendre dans la vue d'un agent et créer un module complémentaire. Vous devez le nommer et dans la section *plugin*, ajouter celui que vous venez de configurer.

Une fois cela fait, cliquez sur « créer ».

Si le module est affiché avec 1, cela signifie qu'il fonctionne correctement et qu'un agent contenant les modules a été créé.

Plugins d'agent

Pour pouvoir superviser à partir de Pandora FMS avec un agent plugin, appelez-le à partir du fichier `.conf` de l'agent logiciel qui se trouve dans le chemin suivant, sous Linux:

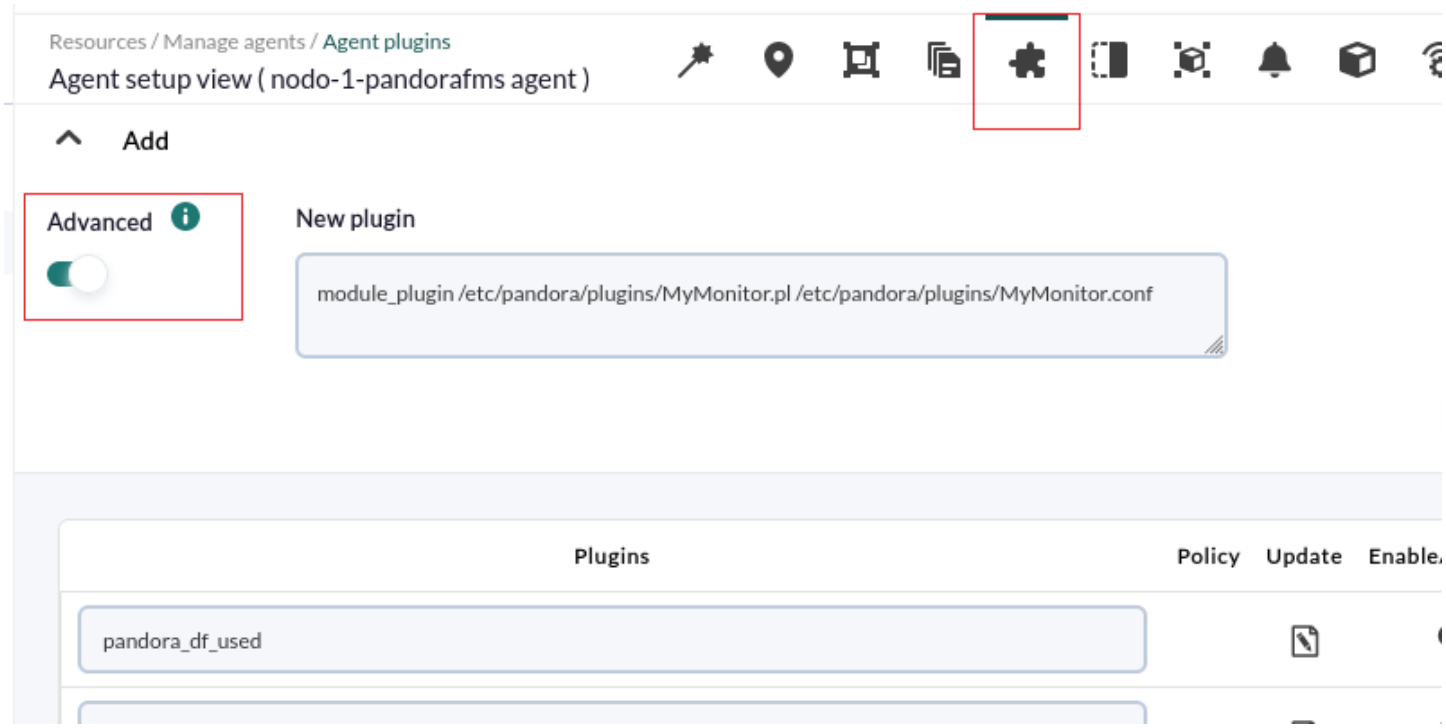
```
/etc/pandora/pandora_agent.conf
```

Vous pouvez créer le run dans la dernière ligne de `.conf` avec la commande `module_plugin`, suivie de la commande run.

Exemple d'exécution possible d'un *plugin* :

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

Cette opération peut également être effectuée à partir de la console Web si la configuration à distance est activée. Exemple de gestion des plugins avancés de l'agent logiciel depuis la console :



Visualisation du plugin dans Pandora FMS

La visualisation des données du *plugin* se fait généralement à partir de la vue de l'agent. La structure des données est généralement définie par des agents et des modules, de sorte que le *plugin* crée normalement des agents et des modules à l'intérieur de ceux-ci, de sorte que ces données peuvent être visualisées à partir du menu de visualisation de l'agent en question.

Dans de nombreux *plugins* il peut arriver qu'ils créent de nombreux agents, comme par exemple le *plugin* de Xenserver qui crée un agent pour chaque machine virtuelle du serveur. Heureusement, nous pouvons définir un préfixe pour les agents, dans ce cas par exemple, le préfixe `xen-` pourrait être utilisé, de sorte que lorsque nous visualisons les agents, lorsque nous voyons que certains d'entre eux ont cette structure dans le nom, nous saurons que ces agents proviennent de ce *plugin* spécifique.

PSPZ2

Un fichier pspz2 est une archive zip de deux fichiers, le *plugin* et un fichier `plugin_definition.ini` contenant la spécification du *plugin* et du module. Ce format de fichier est très utile lors de l'empaquetage des *plugins*, car il simplifie l'installation des plugins.

En utilisant ce format, les *plugins* peuvent être installés plus rapidement dans la console de Pandora FMS à partir de la section « enregistrement des plugins ».

Exemple de `plugins_definition.ini` d'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 : Il doit porter le même nom que le *script* inclus dans le fichier .pspz, précédemment nommé <fichier_script>. Dans cet exemple, il s'agit d'un *script shell* (format `.sh`) nommé `ssh_pandoraplugin.sh`.

plugin_type : `0` pour un plugin standard *de Pandora FMS*, et `1` pour un plugin de type Nagios.

total_modules_provided : Il pécifie combien de modules sont définis dans les sections suivantes du fichier `.ini`. Vous devez en définir un au minimum (pour l'utiliser dans un exemple au minimum).

execution_command : S'il est utilisé, il doit être placé devant *script*. Il peut s'agir d'un interpréteur, tel que `java -jar`. Ainsi, le *plugin* sera appelé à l'exécution, depuis le serveur de plugins de Pandora FMS, avec le code suivant : `java -jar <chemin_du_plugin/nom_du_fichier_du_plugin`.

execution_postcommand : S'il est utilisé, il définit les paramètres supplémentaires transmis à *plugin* après la commande < plugin_filename >, qui est invisible pour l'utilisateur.

total_macros_provided : Il définit le nombre de macros dynamiques que possède le *plugin*.

macro__value : Qui définit la valeur de ce module à l'aide de cette macro dynamique ; si elle n'existe pas, c'est la valeur par défaut qui est retenue.

Et vous devez créer une section pour chaque macro dynamique, par exemple :

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

Compatibilité

Il est important que le *plugin* soit aussi compatible que possible. Certains *plugins* sont propres à des services qui ne peuvent être exploités que sur un système d'exploitation particulier, par exemple MS Windows®. Mais dans le cadre des limitations inévitables, il est important de donner au *plugin* une compatibilité qui couvre autant de scénarios que possible.

Certains systèmes d'exploitation ont des paquets qui n'existent pas sur d'autres systèmes d'exploitation, ce qui peut entraîner des erreurs dans l'exécution d'un *plugin* compilé. Il est donc recommandé de compiler les *plugins* sur une machine du système d'exploitation Rocky Linux 7 avec toutes les dépendances *plugin* nécessaires installées.

Contrôle des erreurs

Un bon suivi des bogues peut faciliter la maintenance de *plugin* au fil du temps, car il permet de détecter les futurs bogues éventuels du plugin et de les localiser plus rapidement.

Il peut également aider à détecter la nature exacte de l'erreur, car sans contrôle d'erreur, dans certains cas, la sortie d'erreur peut être trop générique et il peut être difficile de trouver la nature du problème.

Test et validation des plugins

Une fois le *plugin* développé, il convient de le tester dans des environnements de test afin de vérifier son fonctionnement.

Un bon processus de test peut aider à corriger des bogues potentiels et à rendre *plugin* plus robuste au fil du temps, comme on dit, mieux vaut prévenir que guérir.

Lancement d'un plugin

Lors de la publication du *plugin*, il est important de tenir compte de certains aspects.

Documentation du plugin

L'utilisation de certains *plugins* peut être difficile à comprendre au début, certains peuvent avoir plusieurs options et peuvent être fastidieux à comprendre.

Une bonne documentation peut faciliter la compréhension et peut même être utile au créateur du *plugin* lui-même à l'avenir, pour l'aider à se souvenir de certaines choses sur le *plugin* qu'il a peut-être oubliées au fil du temps.

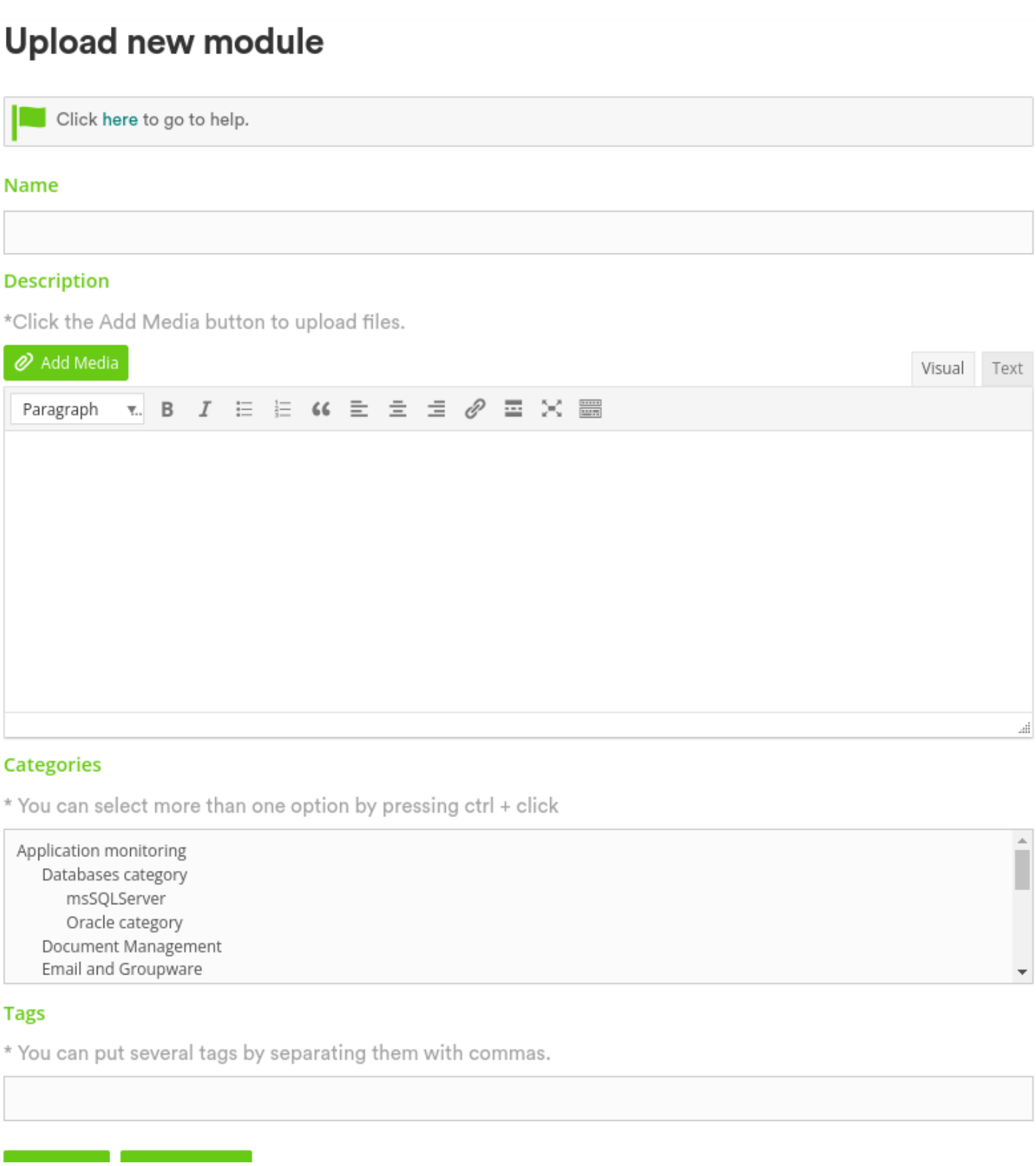
Pandora FMS dispose d'un système de guides rapides *en ligne* / qui facilite la mise à jour de la documentation, accessible à partir du lien suivant :

https://pandorafms.com/guides/public/shelves/fr-guides-rapides

L'alternative à la documentation sous forme de guides rapides est le format PDF.

Télécharger un plugin dans la bibliothèque

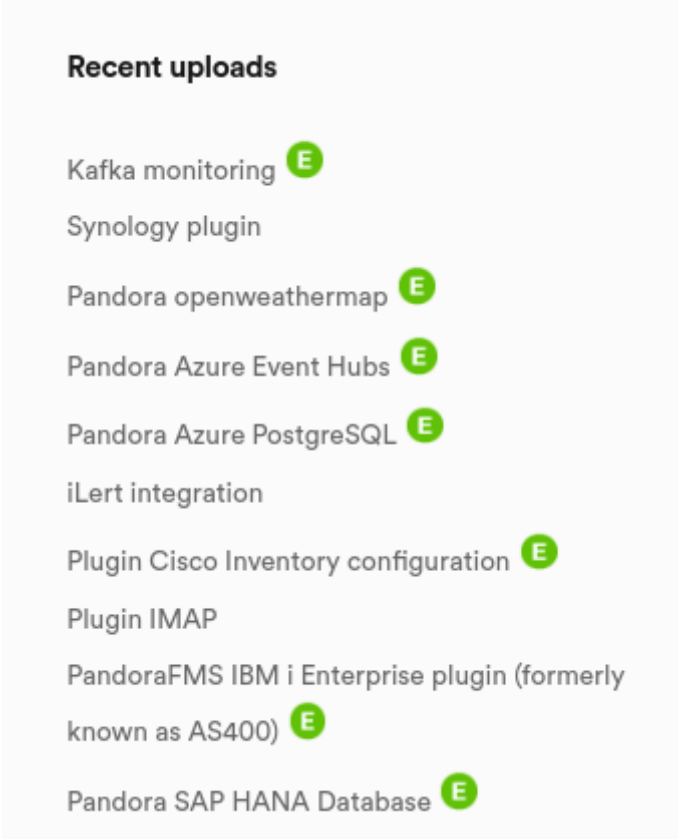
Le processus de téléchargement d'un *plugin* dans la bibliothèque de Pandora FMS est simple. Il suffit d'accéder à la section « upload new module » dans la section « my plugins » de la bibliothèque.



Une fois que vous avez configuré toutes les sections de l'entrée, telles que le titre, une description du *plugin*, le *plugin* compressé au format zip, la documentation, les catégories et les *tags*, vous pouvez publier le plugin en cliquant sur la section submit.

Tous les plugins téléchargés dans la bibliothèque sont examinés et nécessitent un certain temps d'examen avant leur publication finale.

Une fois le plugin publié, il apparaîtra dans la section « derniers téléchargements » :



Meilleures pratiques pour la révision et la maintenance d'un plugin

Certaines bonnes pratiques peuvent contribuer à réduire le temps consacré aux tâches, à prévenir et à contourner les erreurs courantes au cours des différentes étapes de la réalisation du *plugin*, à maintenir le *plugin* dans le temps et à faciliter sa révision. Il peut s'agir :

- Laissez une bonne préconfiguration documentée au cas où il serait nécessaire d'effectuer une préconfiguration du système dans le *plugin* en question.
- Documenter un exemple d'exécution réelle de *plugin* pour aider l'utilisateur à visualiser l'utilisation de *plugin*.
- Écrire des commentaires dans le code du *plugin* peut vous aider à le comprendre mieux ou plus rapidement.
- Donner la priorité à la lisibilité du code. Plus il est complexe, plus il faudra de temps et de ressources pour le traiter.
- Testez tout le code. Il est important de tester le *plugin* pour vérifier que tout va bien. Trouver un bogue à temps et le corriger permet d'éviter des problèmes à l'avenir.

Retour à l'index de la documentation de Pandora FMS