# 夜莺插件机制

## 设计初衷

collector虽然内置了很多监控指标的采集，但无法解决所有场景，所以提供了插件机制，以便扩展collector的功能。

查看collector的配置文件，里边配置了plugin的目录，我们只需把插件脚本放到这个目录下，collector就会自动探测到，然后周期性运行。对于业务系统的监控指标采集，可以把采集脚本放到业务程序发布包中，随着业务代码上线而上线（上线的时候把脚本放到collector的plugin目录下），随着业务代码升级而升级，随着业务代码的下线而下线（下线的时候把脚本从plugin目录下移除），这样会比较容易管理。

## 插件规范

对于插件，有如下几个规范要求：

- 插件脚本必须具有可执行权限，部署完了脚本记得chmod +x一下
- 插件脚本可以是sh、py、pl、rb，甚至可以是二进制，只要机器上有runtime环境
- 插件脚本的命名：${step}_xx.xx，比如60_uptime.sh，${step}是在告诉collector多久运行一次插件
- plugin目录下非${step}_xx.xx命名格式的文件或者目录可以存在没关系，不会被识别为插件

- 插件执行之后要在stdout输出一个json array，collector会截获这个输出，解析为监控指标上报
- 如果插件执行报错了，报错消息要打印到stderr，不要打印到stdout

# 插件样例

下面给一个shell编写的插件例子60_uptime.sh：

```
#!/bin/bash

duration=$(cat /proc/uptime | awk '{print $1}')
localip=$(/usr/sbin/ifconfig `/usr/sbin/route|grep '^default'|awk '{print $NF}'`|grep inet|awk '{print $2}'
|head -n 1)
step=$(basename $0|awk -F'_' '{print $1}')
echo '[
    {
        "endpoint": "'${localip}'",
        "tags": "",
        "timestamp": '$(date +%s)',
        "metric": "sys.uptime.duration",
        "value": '${duration}',
        "counterType": "GAUGE",
        "step": '${step}'
    }
]'
```

下面给一个python编写的插件例子60_plugin_status.py：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time
import commands
import json
```

```
import sys
import os

items = []


def collect_myself_status():
    item = {}
    item["metric"] = "plugin.myself.status"
    item["value"] = 1
    item["tags"] = ""
    item["counterType"] = "GAUGE"
    items.append(item)


def main():
    code, endpoint = commands.getstatusoutput(
        "timeout 1 /usr/sbin/ifconfig `/usr/sbin/route|grep '^default'|awk '{print $NF}'`|grep inet|awk '{pri
nt $2}'|head -n 1")
    if code != 0:
        sys.stderr.write('cannot get local ip')
        return

    timestamp = int("%d" % time.time())
    plugin_name = os.path.basename(sys.argv[0])
    step = int(plugin_name.split("_", 1)[0])

    collect_myself_status()

    for item in items:
        item["endpoint"] = endpoint
        item["timestamp"] = timestamp
        item["step"] = step

    print json.dumps(items)


if __name__ == "__main__":
    main()
```

# 直推数据

除了插件机制，夜莺也直接提供了HTTP接口接收监控数据的推送，实际上，transfer和collector都有HTTP接口暴露，实践上来看，推荐将监控数据推送给collector，不要直接推送给transfer，除非transfer前面做了负载均衡，否则直接将监控数据推送到某一个transfer，容易导致单transfer压力过大，负载不均。

这里我们只是给出collector的推送接口规范：

```
Method: POST
Path: /api/push
Body:
[
    {
        "metric": "disk.bytes.free.percent",
        "endpoint": "10.5.5.5",
        "timestamp": 1564449874,
        "step": 60,
        "value": 32.4,
        "counterType": "GAUGE",
        "tags": "mount=/data12,fstype=xfs"
    }
]

上面只是个例子，各个字段解释：

     metric：指标名称
   endpoint：监控对象
  timestamp：UNIX时间戳
       step：采集上报周期
      value：监控数据当前时刻的值
counterType：当前只支持GAUGE
       tags：监控指标打的标签
```

其实，log_collector的监控数据就是推送给了本地的collector，collector再借助与transfer的长连接将数据转发给transfer。一般collector监听的端口是2018，所以推送数据的地址就是：http://127.0.0.1:2018/api/push。