Partie 2 : Pratiquer

Chapitre 2-1
Récupérer des infos sur le système

1. Introduction

La découverte du langage Python est terminée. Déjà dans votre cerveau de nombreuses nouvelles connexions neuronales se sont créées. Mais celles-ci n'ont pas encore un poids suffisant pour produire du code Python sans forcer. Il est nécessaire de passer en phase d'apprentissage et de pratiquer ce langage, et en premier lieu de regarder et comprendre comment font les autres scripteurs Python.

C'est le but de la deuxième partie de cet ouvrage : pratiquer.

Maintenant, il est possible d'utiliser Python et quelques-uns de ses modules pour en faire des outils du quotidien.

Voici ce que nous allons aborder dans cette deuxième partie :

– Acquérir des informations sur le système
– Utiliser des formats populaires de fichiers
– Manipuler des données
– Générer des rapports et des sites statiques
– Simuler de l'activité sur une base de données

2. psutil : récupérer des informations sur le système

Les premières versions du langage Python ont été inspirées par d'autres langages, tels ABC ou Modula-3, mais surtout le langage C et les outils Unix. Python n'a donc jamais été très loin du système d'exploitation ; de plus, il offre toutes les possibilités du langage C et du système d'exploitation Unix, à portée de script.

Mais au fil du temps, Python est devenu un composant essentiel de nombreuses distributions Linux, a été porté et est compilable sur de très nombreux systèmes d'exploitation. Cela a permis la réalisation de librairies multiplateformes comme `psutil`.

`psutil` (*Python System and process UTILities*) permet de récupérer des informations sur l'utilisation du système et sur la gestion des processus en cours d'exécution.

Ce module regroupe les informations initialement données par de nombreuses commandes Unix telles que : `ps`, `top`, `lsof`, `netstat`, `ifconfig`, `who`, `df`, `kill`, `free`, `nice`, `ionice`, `iostat`, `iotop`, `uptime`, `pidof`, `tty`, `taskset`, `pmap`.

Multiplateforme, il est utilisable sur les plateformes suivantes : Linux, Windows, MacOS, FreeBSD, OpenBSD, NetBSD, Sun Solaris et AIX.

Ce module est essentiel pour superviser, monitorer ou analyser l'utilisation de vos systèmes. Il permet de récupérer des informations sur les processeurs, la mémoire, les disques, les interfaces réseau, les différents capteurs présents et surtout sur les processus en cours d'exécution.

`psutil` a permis la réalisation d'un outil comme glances, dont voici un exemple de session :

```
XXXXXXXXXX (LinuxMint 18.3 64bit / Linux 4.4.0-170-generic)               Uptime: 0:32:51

CPU       1.9%  nice:     0.0%   LOAD    4-core   MEM     10.6%   SWAP      0.0%
user:     1.0%  irq:      0.0%   1 min:    0.24   total:  15.4G   total:   15.7G
system:   0.7%  iowait:   0.2%   5 min:    0.25   used:   1.63G   used:        0
idle:    98.1%  steal:    0.0%   15 min:   0.19   free:   13.7G   free:    15.7G

NETWORK     Rx/s   Tx/s   TASKS 231 (657 thr), 1 run, 230 slp, 0 oth sorted automatically
eth0          0b     0b
lo          520b   520b     CPU%  MEM%   PID USER        NI S Command
wlan0         0b     0b      4.6   0.1  4144 xxxxx      -11 S /usr/bin/pulseaudio --start --
                             2.6   0.2 11372 xxxxx        0 R /usr/bin/python3 /usr/bin/glan
DISK I/O     R/s    W/s      0.7   1.8  9717 xxxxx        0 S /opt/libreoffice6.3/program/so
loop0          0      0      0.7   1.2  4376 xxxxx        0 S cinnamon --replace
loop1          0      0      0.7   1.1  1354 root         0 S /usr/lib/xorg/Xorg -core :0 -s
loop2          0      0      0.3   1.1 10963 xxxxx        0 S /usr/lib/firefox/firefox -cont
loop3          0      0      0.3   0.3  4567 xxxxx        2 S java -Xmx512M -Dsun.net.inetad
loop4          0      0      0.3   0.5  1000 mongodb      0 S /usr/bin/mongod --config /etc/
loop5          0      0      0.0   0.0  1334 root         0 S /usr/sbin/sshd -D
sda1           0      0      0.0   0.1  1319 colord       0 S /usr/lib/colord/colord
sda2           0      0      0.0   0.0 11271 root         0 S kworker/2:0
sda3           0      0      0.0   0.0    46 root       -20 S vmstat
sda4           0      0      0.0   0.0  3821 root         0 S /sbin/agetty --noclear tty1 li
sda5           0      0      0.0   0.1  4222 root         0 S /usr/sbin/smbd -D
sda6           0      0      0.0   0.0  1099 root         0 S /sbin/cgmanager -m name=system
                             0.0   0.0  1377 www-data     0 S /usr/sbin/apache2 -k start
FILE SYS    Used  Total      0.0   0.0  1109 root         0 S /usr/lib/bluetooth/bluetoothd
/ (sda5)    215G   240G      0.0   0.0  1094 root         0 S /usr/sbin/acpid
_ore/8213  89.1M  89.1M      0.0   0.0  4184 root       -10 S krfcommd
_ore/8268  89.1M  89.1M      0.0   0.0    15 root       -20 S kworker/1:0H
_shell/39  56.5M  56.5M
_shell/77  56.5M  56.5M
2020-01-04 08:53:52       No warning or critical alert detected
```

glances

Comme top sur Linux ou topas sur AIX, il se rafraîchit à intervalles réguliers.

De nombreux exemples se trouvent dans le répertoire scripts sur GitHub à l'adresse suivante : https://github.com/giampaolo/psutil

Même si quelques exemples sont donnés dans cet ouvrage, le site est bien fourni en scripts utiles et significatifs, suffisamment pour que cela soit notifié.

Normalement, `psutil` est livré avec les versions de Python 3.4 et supérieures, sinon faites `'pip install psutil'`.

3. Des informations sur les composants

Un ordinateur, finalement, n'est pas quelque chose de complexe, il n'y a que 4 composants principaux.

Et le module psutil permet de tous les examiner en détail.

3.1 Les processeurs

Très simplement, `psutil` permet d'obtenir des informations sur les processeurs avec les méthodes suivantes :

`cpu_times`	Retourne un tuple nommé avec le temps passé en secondes par CPU en mode user / system / idle. Le détail dépend du système. `percpu=True` permet d'avoir le détail par CPU.
`cpu_percent`	Retourne l'utilisation du CPU en pourcentage. `Interval=1` temporise la première utilisation en secondes. `percpu=True` permet d'avoir le détail par CPU.
`cpu_times_percent`	Retourne les pourcentages de l'utilisation des CPU comme le fait la commande `sar -u` (%user %system …%iowait …%idle).
`cpu_count`	Retourne le nombre de CPU logiques. Si l'argument `logical = False`, retourne le nombre de CPU physiques.
`cpu_stats`	Retourne différentes informations comme le nombre de changements de contexte depuis le boot, le nombre d'interruptions...
`cpu_freq`	Retourne la fréquence du/des CPU. `percpu=True` permet d'avoir le détail par CPU.

`getloadavg` Retourne la charge du système il y a 1, 5 et 15 minutes.

Voici un petit exemple pour vérifier si cela se tient avec la commande `sar -u` :

```
# fichier : psutil/cpu1.py

import psutil
import datetime
import time

temps = 1
max_temps=5

count=1

while count < max_temps:
   print(datetime.datetime.now().time(),
                psutil.cpu_times_percent(interval=1))
   time.sleep(temps)
   count += 1
```

On met tout cela dans un script shell :

```
python cpu1.py > cpu1.txt &
sar -u 1 5 >sar.txt &
```

On le lance et il est ensuite possible de consulter les fichiers.

Le résultat de la commande `sar` :

```
Linux 4.4.0-170-generic (W550-BR-007)     04/01/2020     _x86_64_    (4 CPU)

11:51:38        CPU     %user     %nice   %system   %iowait    %steal     %idle
11:51:39        all      3,99      0,00      0,50      0,75      0,00     94,76
11:51:40        all      1,01      0,00      0,75      0,00      0,00     98,24
11:51:41        all      1,74      0,00      0,75      1,00      0,00     96,52
11:51:42        all      2,49      0,00      0,75      1,00      0,00     95,76
11:51:43        all      0,76      0,00      0,25      0,25      0,00     98,74
Moyenne :       all      2,00      0,00      0,60      0,60      0,00     96,80
```

Le résultat de la commande Python `cpu1.py` :

```
11:51:38.720886 scputimes(user=1.0, nice=0.0, system=0.5, idle=98.3,
11:51:40.723420 scputimes(user=1.5, nice=0.0, system=0.7, idle=97.0,
11:51:42.726065 scputimes(user=0.8, nice=0.0, system=0.3, idle=98.7,
11:51:44.728709 scputimes(user=0.5, nice=0.0, system=0.5, idle=98.7,
```

Les valeurs sont quand même assez différentes, sans être non plus foncièrement très éloignées. Mais finalement, une précision absolue n'est pas nécessaire, surtout dans les cas qui nous intéressent.

Généralement, dans notre cas, l'utilisation du CPU est à 10 % près.

Si, sur de grandes périodes de plus de 10-15 minutes en exploitation normale, les valeurs sont :

entre 1 et 75 %	Ok.
à partir de 75 % et au-delà de 80 %	Il faut mettre en œuvre une solution de remplacement ou d'extension.
à partir de 90 %	Il faut planifier l'évolution.

Mais pour être tout à fait honnête, cela n'arrive quasiment plus, au regard de la puissance des processeurs.

3.2 La mémoire

Le module propose deux méthodes pour interroger l'état de la mémoire :

`virtual_memory` Retourne un tuple nommé avec des informations dépendantes du système.

`total` : mémoire totale (sauf la mémoire swap)

`available` : mémoire disponible (sans la mémoire swap)

Pour Linux :

`active` : en cours d'utilisation

`inactive` : marquée comme inutilisée

`buffers` : mémoire cache

`cached` : mémoire cache aussi

`shared` : mémoire partagée

`slab` : cache du noyau

`swap_memory`	Retourne des statistiques sur la mémoire swap.
	`total` : mémoire swap totale
	`used` : mémoire swap utilisée
	`free` : mémoire swap libre
	`percent` : usage en pourcentage
	`sin` : cumul des octets écrits dans la swap
	`sout` : cumul des octets sortis de la swap

Voici un petit script pour illustrer ces informations.

Celui-ci crée deux listes (`mem`, `swp`) à partir des deux méthodes fournies par `psutil`. Ensuite, il comble les listes avec des blancs pour faciliter l'affichage en colonne, vu que les deux listes n'auront pas le même nombre d'éléments. Il suffit ensuite de faire un `print()` par rang.

Ce script est une réécriture du script fourni en exemple meminfo.py sur le site de psutil.

```
#fichier : psutil/ps_meminfo.py

import psutil
from psutil._common import bytes2human


def fmt_ntuple( nt ):
   r = []
   for name in nt._fields:
       valeur = getattr(nt, name)
       if name != 'percent':
           valeur = bytes2human(valeur)
       else:
           valeur = "{:5.2f}%".format(valeur)
       r.append( "{:10s} : {:>7s}".
                format( name.capitalize(), valeur ))
   return r

def main():
   mem = fmt_ntuple( psutil.virtual_memory())
```