MATHIEU
MOITIE—MATHON

Thomas
Victor

Admin Système

Projet : Sonde

Sommaire

I. Présentation général

L’objectif de ce mini-projet et de réaliser, en binôme, une plate-forme logiciel de monitoring d’un parc informatique.
Cette plate-forme, une fois installée, permettra à l’administrateur système que vous êtes de connaître rapidement à chaque fois que vous le demandez l’état détaillé de l’ensemble des serveurs faisant partis du parc et les dernières alertes du C.E.R.T. (http :// www . cert . ssi . gouv . fr /).

Le logiciel permettra aussi d’être prévenu par e-mail en cas de situation de crise. Une situation de crise étant par exemple : quand un serveur n’a pas donné signe de vie depuis plus de 30 minutes, quand un disque dur atteint 100% d’occupation ou
lorsque la RAM est utilisée à 100% .

Ce mini projet doit être réalisé en BASH et en Python, la proportion de chacun des langages dans le projet reste à votre appréciation. Cependant, certain modules auront un langage imposé. Vous pourrez utilisez l’ensemble de la librairie standard python et tous les binaires installés par défaut sur une distribution Ubuntu serveur.

Les librairies tiers python et bash sont acceptées si vous avez l’accord explicite de votre tuteur. Nous maintiendrons une liste officiel des librairies utilisable sur le forum du cours sur e-uapv.

II. Résolution problème

A. Les sondes

Il nous était demandé de créer des sondes serveur afin de récupérer les données des utilisateurs :
- Nom de la machine,
- %Cpu utilisé,
- %Ram utilisée,
- %Swap utilisé,
- Heure de la sonde

C'est données doivent être récupéré de plusieurs façon, nous avons créé 3 sondes, une en bash, une autre en python et enfin la dernière qui utilise à la fois du python et du bash, avec comme preuve la capture écran de mon code à l'appui.

```python
#!/usr/bin/env python
import os
import time
import Sonde2
import Sonde3
import BDD
import Graph
import module_email


while 1:
    os.system('''sh "Sonde1.sh"''')
    Sonde2.sonde2()
    Sonde3.sonde3()
    BDD.bdd()
    BDD.bdd_pars()
    Graph.graph()



    time.sleep(15)

```

Les sondes qui sont appelé vont récupérer les données des ordinateurs du serveurs et les écrire sur un fichier texte, appelé « resonde1.txt »

```
ge-c026-124

5.60
43.44
0.00
46

1

218

2018-04-30:09:46:31

09


```

B. Base de données

Une fois que les données sont récoltées, nous allons créer une base de données avec toutes ces informations, avec la librairie « sqlite3 », nous allons la créer si elle n'existe pas et si elle existe nous allons la modifier.

Nous allons récupérer les données dans le fichier texte, puis on va se connecter à la base de données qu'on va appeler « sonde.db ».

```
#-------------------------------------------------------------------------------------------------
#Nous allons construire la Base de Données, pour cela nous allons récupérerer les fichier
#dans le fichier texte remplie avec les sonde et les envoyés dans une liste
#-------------------------------------------------------------------------------------------------
liste = [None] * 9
i=0
with open("ressonde1.txt","r") as f:
    for line in f:
        for word in line.split():
            liste[i]=word
            i += 1
    f.close()


#-------------------------------------------------------------------------------------------------
#Création Base de données
#On va créer une base de données avec la librairie "sqlite3", on va indiquer que l'on se
#trouve à la fin du fichier avec le cursor.
#Puis nous allons créer les tables seulement si elle n'existe pas
#-------------------------------------------------------------------------------------------------
connection = sqlite3.connect('sonde.db')
x = connection.cursor()
x.execute('''CREATE TABLE IF NOT EXISTS sonde (
    hostname TEXT,
    cpu INTEGER,
    ram INTEGER,
    swap INTEGER,
    disque INTEGER,
    utilisateur INTEGER,
    processus INTEGER,
    time DATE,
    heure INTEGER
    )''')


#-------------------------------------------------------------------------------------------------
#Supprime les données de plus d'une heure
#-------------------------------------------------------------------------------------------------
x.execute("DELETE FROM sonde WHERE heure != strftime('%H','now')+2")


#-------------------------------------------------------------------------------------------------
#Insert les valeurs de la précédente liste dans la bdd et l'affiche après
#-------------------------------------------------------------------------------------------------
x.execute("INSERT INTO sonde VALUES(?,?,?,?,?,?,?,?,?+2)",liste)
for row in x.execute('SELECT * FROM sonde'):
    print (row)


#-------------------------------------------------------------------------------------------------
#Va récupérer les données du CPU, RAM, Disque et le temps et l'envoie dans le module de
#crise. Pour voir l'état des machines
#-------------------------------------------------------------------------------------------------
x.execute("""SELECT cpu, ram, disque, time FROM sonde""")
data = x.fetchone()

module_crise.al_crise(data[0],data[1],data[2])

connection.commit()
connection.close()
```

Ensuit on va créer une base de données à l'aide d'un parseur, qui va se connecter à un serveur prédéfini pour trouver les nouvelles alertes et les mettre dans une base de données spéciale.

Parseur :

```python
#!/usr/bin/python
# -*- coding: latin-1 -*-
import os, sys
from lxml import html
import requests

#-----------------------------------------------------------------------------------------------
#Fonction qui va récupérer la dernier alerte CERT et qui va l'envoyer au moteur de stockage
#-----------------------------------------------------------------------------------------------

def parseur():
    page = requests.get('http://www.cert.ssi.gouv.fr/')
    pageHTML = html.fromstring(page.content)

    alert = pageHTML.xpath('//td[@class="ale"]/text()')
    tmp = ""
    for alerte in alert:
        if alerte != " ":
            tmp = alerte[:-1].split('(')
            break

    return tmp
```

Base de données parseur :

```python
def bdd_pars():

    #-----------------------------------------------------------------------------------------------
    #On va récupérer les info grace à la fonction parseur_web qui va se connecter à une page HTML
    #et qui va renvoyer la dernière alert.
    #Cette dernière alerte va être envoyé à la Bdd créer précédement, si la table est déjà crée
    #on supprime l'ancienne alert et on affiche la BDD
    #-----------------------------------------------------------------------------------------------
    liste = Parseur.parseur()
    connection = sqlite3.connect('Alerte.db')
    c = connection.cursor()

    c.execute('''CREATE TABLE IF NOT EXISTS Alerte (
     Alerte TEXT,
     time DATE
    )''')
    c.execute("DELETE FROM Alerte")

    #c.execute("INSERT INTO Alerte VALUES(?,?)",liste)

    for row in c.execute('SELECT * FROM Alerte'):
        print (row)

    connection.commit()
    connection.close()
```

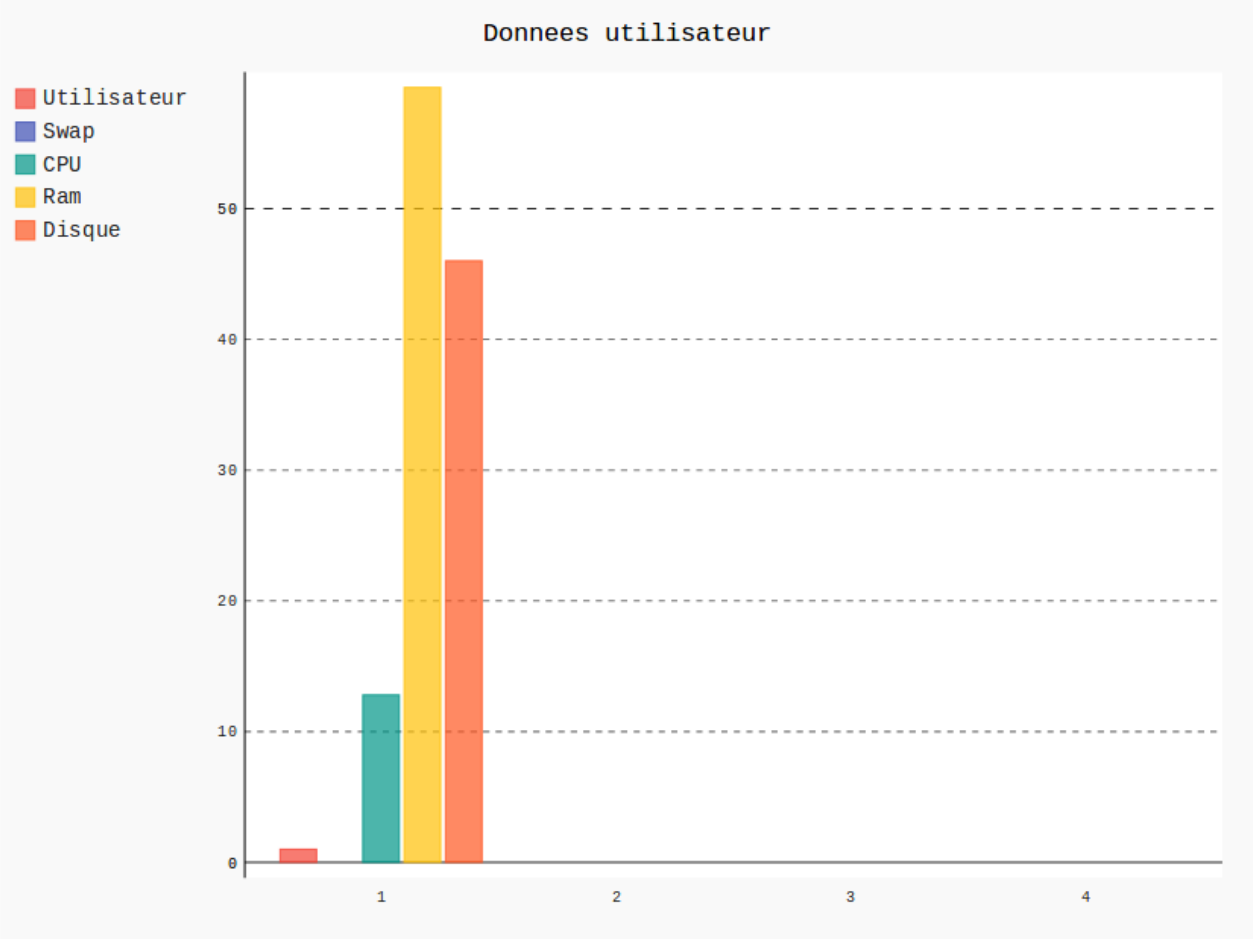
C. Graphique

Pour créer les graphiques de chaque utilisateur, nous allons nous connecter à la base de données précédemment créé et afficher les utilisateurs avec toutes leurs informations. Nous avons utilisé la librairie « Pygal » pour ça.

Nous nous connectons à la base de données, nous lançons une recherche SQL dans la base de données.

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
import os, sys
import sqlite3
import pygal
from datetime import datetime, timedelta

def graph():
        connection = sqlite3.connect('sonde.db')

        c = connection.cursor()

        #declaration des list dont on a besoin
        hostnames = list()
        n=5

        #-----------------------------------------------------------------------------------------
        #Les fonctions qui vont suivre vont permettre de créer un graphique des données suivantes :
        #    - %Disque utilisé,
        #    - %Ram utilisé,
        #    - %CPU utilisé,
        #    - SWAP total,
        #    - Nombre utilisateur connectés
        #Les graphiques seront crées sous forme de page HTML, elles vont se mettre à jour a chaque
        #fois que les sondes vont être relancés
        #-----------------------------------------------------------------------------------------

        for row in c.execute('SELECT DISTINCT hostname FROM sonde'):
            hostnames.append(row)

        for name in hostnames:
            for row in c.execute("""SELECT utilisateur, ram, disque, cpu, swap time FROM sonde"""):
                    data = c.fetchone()
                    bar_chart = pygal.Bar()
                    bar_chart.title = 'Donnees utilisateur'
                    bar_chart.x_labels = map(str, range(1,n))
                    bar_chart.add('Utilisateur', data[0])
                    bar_chart.add('Swap', data[4])
                    bar_chart.add('CPU',  data[3])
                    bar_chart.add('Ram', data[1])
                    bar_chart.add('Disque', data[2])
                    bar_chart.render_in_browser()
```

La fonction va écrire dans une liste tous les utilisateurs enregistrés dans la base de données créée précédemment. Ensuite on va parcourir la liste de tous les utilisateurs enregistrés dans la liste, on va effectuer une recherche SQL dans la base de données pour récupérer les données qui nous intéressent pour chaque utilisateur et les envoyer dans le graphe avec PYGAL, via un site internet.


Donnees utilisateur
Utilisateur
Swap
CPU
Ram
Disque
50
40
30
20
10
0
1
2
3
4

D. Mail

Nous allons écrire dans un fichier texte l'adresse de l'envoyeur, ensuite l'adresse du destinataire et enfin le mot de passe de la personne qui enverra le mail. Si le fichier texte est vide, il va demander à l'utilisateur de rentrer les informations .

Quand la capacité maximum d'un outil va être atteinte, en l'occurrence :
- %CPU,
- %Ram,
- %Disque.

Le serveur envoi un mail afin de prévenir que la capacité maximale à été atteinte et qu'il y a une erreur, nous allons vous montrer un exemple, avec une valeur de 100 au moment ou on appel le module de crise avec la crise qui est en train de s'effectuer :

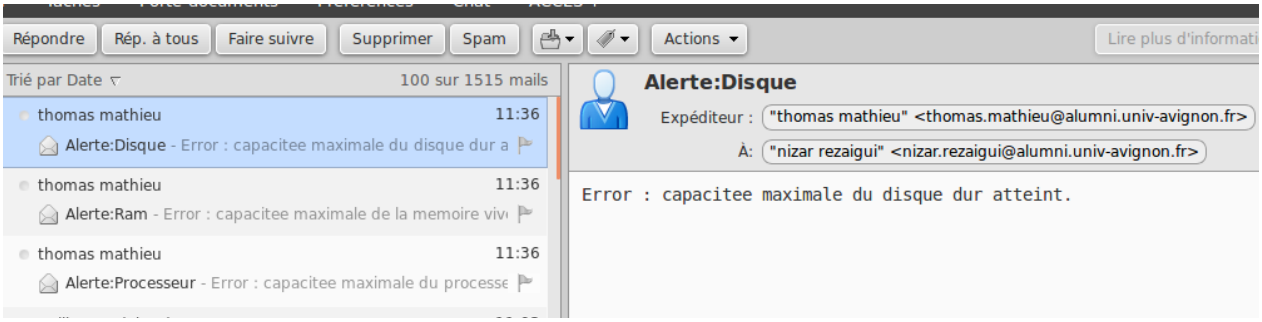
```
module_crise.al_crise(100,100,100)
```

```
#-------------------------------------------------------------------------------------------
#Fonction qui va permettre de vérifier que
#-------------------------------------------------------------------------------------------


def al_crise(process,ram,disque):

    #-------------------------------------------------------------------------------------------
    #Dans cette fonction, on va vérifier si les arguments passer dans la fonction
    #    - process
    #    - ram
    #    - disque
    #    - absence
    #ne sont pas supérieur à 100, si ils sont supérieur on envoi un mail
    #avec le fonction send_mail()
    #-------------------------------------------------------------------------------------------

    if process > 99 :
        module_email.send_mail("Processeur","Error : capacitee maximale du processeur atteinte.")

    if ram > 99 :
        module_email.send_mail("Ram","Error : capacitee maximale de la memoire vive atteinte.")

    if disque > 99 :
        module_email.send_mail("Disque","Error : capacitee maximale du disque dur atteint.")
```

La fonction que vous voyez, va effectuer des vérifications pour les arguments, si elles sont supérieures on envoi un mail avec le module suivant :

```python
#!/usr/bin/env python
import smtplib
import sys
import os
from getpass import getpass
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

#parametre mail
def param_mail():

    print("Enregistrer les nouveaux parametres emails:")

    print("Votre adresse:")
    From = sys.stdin.readline()
    print("Destinataire")
    To = sys.stdin.readline()
    print("Votre mot de passe:")
    Mdp = sys.stdin.readline()
    fichier = open("paramail.txt", "w")
    fichier.write(From)
    fichier.write(To)
    fichier.write(Mdp)
    print("Parametre sauvegarde")
    fichier.close()


#Creation serveur mail
def send_mail(typ,mes):
    fichier = open("paramail.txt", "r")
    user = fichier.read().split('\n')

    if os.path.getsize("paramail.txt") == 0:
        print("Vos parametre sont vides")
        param_mail()

    msg = MIMEMultipart()
    msg['From'] = user[0]
    msg['To'] = user[1]
    msg['Subject'] = 'Alerte:' + typ


    mailserver = smtplib.SMTP_SSL('smtpz.univ-avignon.fr:465')
    message = mes
    msg.attach(MIMEText(message))

    mailserver.ehlo()
    mailserver.login(user[0], user[2])

    #envoie mail
    mailserver.sendmail(user[0], user[1], msg.as_string())
    mailserver.close()
    print "Un mail a ete envoye %s" %(msg['To'])
    fichier.close()
```

Capture d’écran des mails envoyé :



III. Conclusion

Ce projet nous à appris a utiliser plusieurs librairie utiles, cela nous a permis de mieux connaître le rôle d’administrateur système qui est présent dans la plupart des entreprise.

Nous avons appris à travailler en duo malgré les difficultés que nous avons put trouver (manque de connaissance en bash, ne pas connaître les librairie utilisé) en nous partageant les tâches et en planifiant des réunion entre nous afin de pouvoir mieux travailler et se partager la charge de travail qui était colossal.

Ce projet a été pour nous un vrai apport professionnel.