

Rapport SAE203

[Voir sur GitHub](#)

Table des matières

I/ Introduction.....	3
II/ Plan de travail.....	3
a) Conception des machines.....	3
b) GitHub.....	4
III/ Conception d'aggreg.py.....	4
a) Python « charge_urls ».....	4
b) Python « fusion_flux ».....	5
c) Python « genere_html ».....	6
d) Python « load_config() ».....	7
e) Python « generate_rssurls ».....	7
IV/ Axe de progrets.....	7
V/ Conclusion.....	8

I/ Introduction

L'objectif de cette SAE est de créer une solution informatique déployer sur plusieurs machines avec un cahier des charges. Cette solution est un agrégateur de flux RSS, configurable et déployable.

Pour ce faire, j'ai mis en place un environnement de travail avec trois machines virtuelle qui font office de serveurs pour la génération de flux RSS, une machine virtuelle agrégateur et une machine client avec interface graphique tout en respectant l'attribution d'IP comme indiqué [ICI](#) et en leur attribuant leurs noms d'hôte dans le fichier « `/etc/hosts` ».

II/ Plan de travail

a) Conception des machines

Le flux RSS se transmet via des requêtes HTTP et chaque machine fonctionne avec les mêmes bases. Par conséquent j'ai choisi de me simplifier la vie pour la création de mon espace de test. Pour commencer, j'ai créé ma première machine serveur1.net et effectué toutes ces manipulations :

- Installer l'OS Debian.
- Créé chaque utilisateur (serveur1, serveur2, serveur3 et aggreg).
- Configurer tous les hôtes dans « `/etc/hosts` ».
- *Installer le script de génération de flux RSS.*
- *Paramétrer le masque de sous-réseau, l'IP du serveur DNS et la passerelle.*
- *Installer le packet Apache2.*
- *Configurer Apache2 et créé un fichier de configuration par compte.*
- *Installer Python3 et Pip.*
- *Installer Git.*

Par conséquent, ma machine serveur1.net, peut prendre le rôle de n'importe quel hôte dans mon espace de travail, en simplement changeant son IP, activer le bon fichier de configuration sur Apache. Cette méthode de travail ma permit d'économiser un temps de travail conséquent, et en cas de corruption sur une machine, je peux sans problème cloner une autre machine, et modifier les paramètres énoncés précédemment pour qu'elle prenne son nouveau rôle dans mon espace de travail.

Seul changement, sur ma machine agrégation j'ai mon Git [Python SAE203](#) de cloner.

b) GitHub

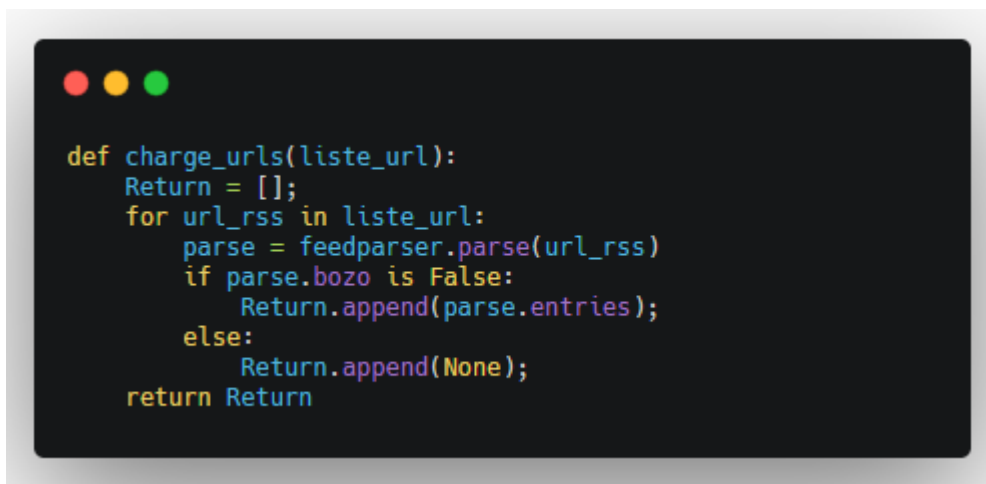
Pour pouvoir travailler depuis partout sans m'embêter à transporter le code avec moi, j'ai choisi la solution GitHub, ayant déjà eu à l'utiliser la prise en main fut facile. J'ai cloné le [Git](#) avec une clef d'authentification générée spécialement pour l'occasion me permettant de commit les changements faient sur la machine directement depuis celle-ci.

Pour la conception du markdown, j'ai utilisé le site web [stackedit.io](#) me permettant d'avoir les sources d'un côté et le résultat en temps réel d'un autre.

III/ Conception d'aggreg.py

[Les programmes sont commentés entièrement sur le GitHub](#)

a) Python « charge_urls »

A screenshot of a code editor with a dark background and light-colored text. The code defines a function `charge_urls` that takes a list `liste_url` as input. It initializes an empty list `Return`. It then iterates over each `url_rss` in `liste_url`. For each URL, it uses `feedparser.parse(url_rss)` to parse the RSS feed. If `parse.bozo` is `False`, it appends `parse.entries` to the `Return` list. Otherwise, it appends `None`. Finally, it returns the `Return` list. The code is as follows:

```
def charge_urls(liste_url):
    Return = []
    for url_rss in liste_url:
        parse = feedparser.parse(url_rss)
        if parse.bozo is False:
            Return.append(parse.entries)
        else:
            Return.append(None)
    return Return
```

Figure 1: `charge_urls()`

Cette fonction consiste à récupérer les documents RSS des URLS *liste_url* qui correspondent à des flux RSS, les range dans une liste et si l'URL est inaccessible il range la valeur *None* et finit par renvoyer la liste.

b) Python « fusion_flux »

```
def fusion_flux(liste_url, liste_flux, trichrono):
    Return = [];
    for i in range(len(liste_url)):
        if liste_flux[i] is not None:
            for feed in liste_flux[i]:
                dic = {};
                dic["titre"] = feed["title"];
                dic["category"] = feed["tags"][0]["term"];
                dic["serveur"] = liste_url[i];
                dic["date_publi"] = feed["published"];
                dic["lien"] = feed["link"];
                dic["description"] = feed["summary"];

                element = datetime.strptime(dic["date_publi"], "%a, %d %b %Y %H:%M");
                dic["time"] = int(datetime.timestamp(element));

                if(len(Return) > 0 and trichrono):
                    for i2 in range(len(Return)):
                        if Return[i2]["time"] < dic["time"]:
                            Return.insert(i2, dic);
                            break;
                    else:
                        Return.append(dic);
    return Return;
```

Figure 2: *fusion_flux()*

Cette fonction produit en sortie une liste dont chaque élément est un dictionnaire décrivant un événement provenant d'un des sites. Les clés de ce dictionnaire correspondent aux éléments du document RSS que nous voulons récupérer. Si l'option *tri_chrono* est *true*, alors compare le timestamp avec la date de notre document RSS, et on regarde à partir de quel index, ce timestamp est le plus élevé et on l'insert, de ce fait une chronologie se crée au fur et à mesure.

c) Python « `genere_html` »

```
def genere_html(liste_evenements, chemin_html):
    output = "";

    chemin_css = "/".join(chemin_html.split("/")[:-1]) + "/theme.css";

    with open('assets/model.html', "r") as f:
        content = f.read();

    parts = content.split("{boucle}");

    actualTime = time.strftime("%a, %d %b %Y %H:%M", time.localtime());
    output = parts[0].replace("{date-et-heure-actuelle}",actualTime);

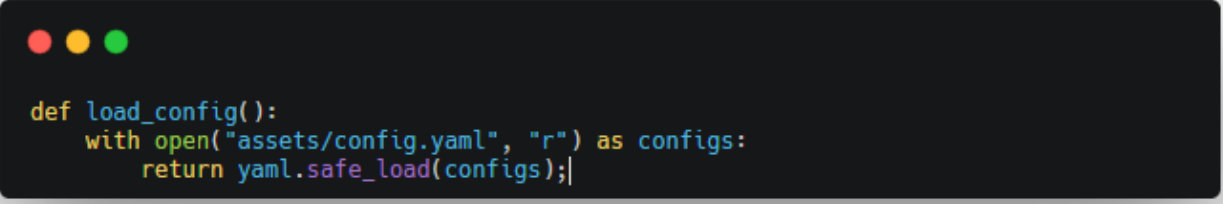
    for event in liste_evenements:
        pre_output = parts[1];
        for key in event:
            pre_output = pre_output.replace("{"+str(key)+"}", str(event[key]));
        output += pre_output;

    output += parts[2];
    with open(chemin_html, "w+") as f:
        f.write(output);
    if not file_exists(chemin_css):
        with open('assets/theme.css', "r") as f:
            with open(chemin_css, "w+") as f:
                f.write(f.read());
```

Figure 3: `genere_html()`

Cette fonction permet de générer la page HTML, elle utilise une mécanique très modulable et simple de modification. Un fichier HTML « model » est chargé, ici `assets/model.html` et découpe la page en trois parties séparées par une balise `{boucle}` qui englobe les balises `<article>` a répété pour chaque flux. Écrit dans la variable `output` l'entête en y ajoutant l'heure actuelle, fait une boucle de chaque flux et écrit pour chacun deux les balises entre `{boucle}` avec les informations du flux dans `output` et finit par rajouter dans la variable le footer. Pour finir, `output` est print dans le fichier de destination et si le fichier CSS n'est pas présent, il le met au côté du fichier de destination.

d) Python « `load_config()` »




```
def load_config():  
    with open("assets/config.yaml", "r") as configs:  
        return yaml.safe_load(configs);|
```

Figure 4: `load_config()`

Cette fonction simple charge avec la librairie Yaml le fichier de configuration et le renvoi.

e) Python « `generate_rssurls` »



```
def generate_rssurls(urls, name):  
    Return = [];  
    for url in urls:  
        Return.append(url+"/"+name);  
    return Return;  
|
```

Figure 5: `generate_rssurls()`

Cette fonction fait la concaténation entre les URLS et le nom du fichier RSS et renvoie une liste contenant les nouvelles URLS.

IV/ Axe de progrès

Lors de tests de grande envergure en dupliquant plusieurs dizaines de fois le même serveur produisant des flux RSS, un problème c'est vite posé : la page HTML devenait trop volumineuse. En effet, il faudrait mettre en place un système de scrolle progressif, pour que les flux se chargent au fur et à mesure que l'on déroule la page WEB.

Mais aussi, mettre en place les requêtes en HTTPS, même si le flux RSS n'a rien de dangereux confidentiellement, HTTPS est toujours préférable à HTTP, et d'expérience il me semble qu'il faut mettre une solution particulière pour le support du ssl.

VI/ Conclusion

Cette SAÉ était d'envergure mais grâce avec mes expériences dans d'autres langages je n'ai pas eu de difficulté particulière à sa réalisation et beaucoup de connaissances requises on été apprises en TP. Cependant, si je les avais pas eus, finir ces travaux aurait beaucoup plus long, cela demandait beaucoup de rechercher personnel. Mais malgré cela, le traitement de flux RSS centralisé était très intéressant et stimulant.. Aucun doute, un jour ou l'autre je serais content d'avoir travaillé dessus quand il faudra que je développe une telle solution.