Rapport Projet Fil Rouge

Auteur : JOUBIOUX Alan

Date : 27 mars 2020

Sommaire

Introduction

Prise en main

Dans un premier temps afin de découvrir l'outil conçu, je vous invite à vous rendre sur les liens suivant :

- Lien vers l'application développée en HTTPS avec identification et authentification. https://54.246.242.159/
- Lien vers l'application développée en HTTP sans identification et authentification ainsi que sa documentation SWAGGER http://54.246.242.159:8000/home & http://54.246.242.159:8000/swagger/

Vous trouverez les quatre fonctionnalités suivantes :

- Convertisseur de fichier : convertit un fichier donné en un fichier au format JSON : {métadonnées, fichier_binaire}
- Liste des fichiers : liste des fichiers JSONifiés présents sur le bucket de stockage
- Suppression d'un fichier : supprime un fichier JSONifié donné du bucket de stockage
- Récupération d'un fichier : récupère un fichier JSONifié donné du bucket de stockage

La différence entre la version HTTPS et HTTP, en plus d'être la présence ou non d'un certificat, est que la version HTTPS est plus 'lourde' d'utilisation avec un système de log plus précis et un système d'authentification complet. L'intérêt de la version 'HTTP' a été de permettre de développer certains POC qui ont par la suite été mis en place dans le HTTPS.

Afin d'accéder à l'ensemble de la documentation de ce projet, vous pouvez soit lire le PDF présent à la racine du repository ou vous rendre sur le lien suivant :
https://PFRAlanJBX.readthedocs.io/

Présentation du GIT

Vous trouverez dans ce repository les dossiers suivant :

- Dossier IP : l'ensemble des fichiers et scripts permettant la mise en place de l'instance et la création du certificat HTTPS https://github.com/AlanJBX/FilRouge/tree/master/IP
- Dossier Flask : l'ensemble des fichiers permettant la mise en place des serveurs HTTP, HTTPS en python https://github.com/AlanJBX/FilRouge/tree/master/Flask
- Dossier Serverless : l'ensemble des fichiers permettant le déploiement d'une application serverless. https://github.com/AlanJBX/FilRouge/tree/master/Serverless
- Dossier Sphinx : l'ensemble des fichiers sources permettant la génération de la documentation. https://github.com/AlanJBX/FilRouge/tree/master/Sphinx

Briques développées :

IPV4 :

- Instance EC2 et bucket S3, oui
- OS FreeBSD, oui
- Usage de PacketFilter, oui
- Connexion SSH via clé et id/mdp, oui
- Connexion à l'application, oui
- Protocole HTTPS, oui
- Identification, oui

Python :

- Utilisation de Python et Flask, oui
- Dépôt d'un fichier et retour JSON, oui
- API type RESTFull, en partie
- Gestion des extensions, oui
- Gestion des métadonnées, oui
- Gestion des erreurs d'extension, oui
- Appel d'un autre API, en partie

SSI :

- Connexion SSH via clé et id/mdp, oui
- Protocole HTTPS, oui
- Connexion avec id/mdp à l'application, oui
- Gestion d'un nouvel utilisateur, oui

AWS / IAAS :

- Serverless, en partie
- Gestion des métadonnées, oui
- Utilisation d'un bucket S3, oui
- Fichier test et commande de requête, oui
- Utilisation AWS Rekognition, oui

SOA :

- Interface graphique, oui
- Commande CURL, oui
- API Manager, oui
- API Limitation requête, oui
- API Sécurisation, oui
- API Interconnectée, oui
- API Documentation, oui

Difficultés rencontrées

De petites difficultés ont rencontrées sur l'ensemble du projet et sur l'ensembl des briques développées. Elles ont été de deux natures différentes :

Automatisation : afin d'avoir un code propre et facilement debuggable, j'ai tenté d'automatisé le plus possible de chose. Il reste néanmoins de nombreuses choses à effectuer. Cette difficulté a eu pour principal effet de me faire perdre beaucoup de temps pour des gains marginaux.

Spécificités et compatibilités : afin de faire correspondre mon idée de mon programme aux capacités de chaque technologie, il m'a fallu chercher les méthodes spécifiques à chacune d'entre elle. Cette difficulté a eu pour effet principal de rendre le code plus efficace et de me faire montée en compétence.

Améliorations potentielles

IPv4 :

- Mise en place d'un Packet Filter complète et totale d'un PF au sein de l'instance.

Python :

- Augmentation des extensions et métadonnées liées (pour les vidéos notamment)
- Prise en compte du MIMEType
- Gestion plus évoluée des erreurs (afin prendre en comptes les codes status)
- Passage en API RESTfull
- Mise en place de l'AutoDoc en lien avec les DocStrings rédigées
- Résolution du problème de buffer pour l'appel à une autre API

SSI :

- Générateur de mot de passe aléatoire

AWS / IAAS :

- Mise en place complète de ServerLess
- Automatisation de la création et de la gestion du bucket à partir d'un programme/script.

SOA :

- Génération des requêtes CURL prenant en compte la spécificité de mon programme HTTPS avec id/auth.

SAV

Dans le cas d'un problème technique, vous pouvez me joindre à mon adresse mail @student-cs.fr.

Notamment dans le cadre de la gestion de l'API Rekognition qui nécessite une autorisation préalable et temporaire sur RosettaHub depuis mon compte personnel afin d'être utilisée.

Remerciements

À ma fille qui m'a appris à la bercer dans un bras et coder de la main libre. Et à ma conjointe, bien conciliante.

Module IPv4

Introduction

Présentation générale

Le serveur est hébergé via RosettaHub dans le Cloud d'Amazon. Une instance EC2 avec un OS de type FreeBSD relié à un bucket S3 permet le traitement et le stockage des fichiers envoyés par un utilisateur dans le cadre de l'API développée.

Fonctions développées

- Instance EC2 et bucket S3, oui
- OS FreeBSD, oui
- Usage de PacketFilter, oui
- Connexion SSH via clé et id/mdp, oui
- Connexion à l'application, oui
- Protocole HTTPS, oui
- Identification, oui

Accès à l'instance

Afin de vous connecter à l'instance : ssh ec2-user@54.246.242.159

Infrastructure du serveur

Le serveur se décompose en deux entités :

- un bucket S3 AWS 'stockagefbsdpfilrougealan' composé comme suit :

  stockagefbsdpfilrougealan
  |- Sauvegarde/
  |- StockageJSON/

- une instance EC2 AWS sous FreeBSD 11.3-STABLE-amd64-2020-02-20.

Accès au service

Il est possible de se connecter à l'application aux adresses suivantes :

- Lien vers l'application développée en HTTPS avec identification et authentification. https://54.246.242.159/
- Lien vers l'application développée en HTTP sans identification et authentification ainsi que sa documentation SWAGGER.

http://54.246.242.159:8000/home & http://54.246.242.159:8000/swagger

Afin de gérer le serveur, deux connexions SSH sont disponibles :

- avec une clef : ssh -i ~/Desktop/PFR/keyPFilRougeAlan.pem ec2-user@54.246.242.159 , clef que je garde sur mon ordinateur
- avec un identification/authentification : ssh ec2-user@54.246.242.159

Packet Filter

Malgré la présence d'un packet filter disponible via la gestion AWS de l'instance, un PF a néanmoins été développé pour le test.

Néanmoins, par précaution et afin de pouvoir accéder à l'instance depuis différents ordinateurs, il n'a pas été activé de manière permanente. Seul le PF d'AWS est activé.

Un point d'amélioration du service serait la mise en place complète et totale d'un PF au sein de l'instance.

Protocole HTTP & HTTPS

Vous trouverez l'ensemble des informations relatives à la sécurité de l'application sur la page dédiée à la SSI.

Module Python

Introduction

Présentation générale

L'application permet accepte le dépôt des fichiers dont les extensions sont les suivantes : pdf, jpeg, png, jpg, gif, bmp, txt, py, csv, ods, odt, odg, ipynb, json, docx, doc, xls, tex avec un stockage local sur le serveur et dans le cloud et de restituer un fichier JSON des fichiers envoyés enrichi des métadonnées de ces derniers.

Fonctions développées

- Utilisation de Python et Flask, oui
- Dépôt d'un fichier et retour JSON, oui
- API type RESTFull, en partie
- Gestion des extensions, oui
- Gestion des métadonnées, oui
- Gestion des erreurs d'extension, oui
- Appel d'un autre API, en partie

Mise en oeuvre

Afin d'utiliser le programme, deux méthodes sont possibles :

- Depuis l'interface graphique :
  https://54.246.242.159/ ou http://54.246.242.159/home

- Avec la méthode 'curl' (uniquement pour la version HTTP) :
  http://54.246.242.159:8000/swagger/

Il est également possible de lancer l'application sur votre ordinateur :
Version HTTPS :
gunicorn-3.7 --certfile=certificat.crt --keyfile=certificat.key --bind 0.0.0.0:443 __init__:application

Version HTTP :
gunicorn-3.7 --bind 0.0.0.0:8000 __init__http:application

Il suffit alors de remplacer l'adresse IP de l'instance AWS par l'adresse IP de votre machine (adresse IP externe et non le localhost)

Améliorations possibles

Il serait envisageable d'effectuer les améliorations suivantes :

- Augmentation des extensions et métadonnées liées (pour les vidéos notamment)
- Prise en compte du MIMEType
- Gestion plus évoluée des erreurs (afin prendre en comptes les codes status)
- Passage en API RESTfull
- Mise en place de l'AutoDoc en lien avec les DocStrings rédigées
- Résolution du problème de buffer pour l'appel à une autre API

Programmation sous Python et Flask

Le programme est développé en Flask-1.1.1 et Python-3.7.3.

Python est en charge du traitement du fichier tandis que Flask est en charge du moteur web tandis que Gunicorn se charge de la mise en place du serveur.

Modules développés

- Module Auth : permet de gérer les authentifications/identifications, l'ajout d'un nouvel utilisateur
- Module AWS : permet de gérer les appels vers et depuis le Bucket S3 ainsi que l'API Rekognition
- Module Extensions : permet de tester l'extension du fichier pris en compte et de retourner les métadonnées particulères liées
- Module FlaskApp : permet de gérer les pages WEB du serveur HTTP(S) de l'application et de faire du micro traitement de fichier
- Module Hash : mini programme permettant de traiter de hasher des mots de passe

- Module Logger : permet de traiter les LO
- Module Swagger : permet d'appeler le module SWAGGER pour la génération de la documentation
- Module Traitement : permet de traiter la conversion du fichier d'origine en version JSONifié
- Module Serverless : permet de gérer l'application en version serverless
- Module ViaCURL : permet d'appeler une autre API pour la conversion

Application au format RESTFull

L'application est développée afin de correspondre qu'en partie aux propriétés RESTfull. Notamment la partie 'authentification' et la partie 'liens entre les ressources'.

- **URI comme identifiant** : chaque ressource de l'API est défini par une URI propre et hiérarchisée
- **Verbes HTTP** en identification des opérations : utilisation des opérations POST et GET
- **Réponses HTTP** en représentation des ressources : utilisation de la réponse GET
- **Liens entre les ressources** : non mis en œuvre
- **Paramètre comme jeton d'authentification** : non mis en œuvre

Restitution d'un fichier sous format JSON

La restitution du fichier s'effectue en trois phases :

Phase 1, récupération des métadonnées

* Analyse de l'extension du fichier et stockage en local si extension pris en compte
* Récupération des métadonnées générales et particulières, en fonction de l'extension, sous forme d'un dictionnaire :ref: `PythonMETA`

Phase 2, transformation json du fichier

* Ouverture du fichier d'origine au format binaire
* Stockage du fichier binaire dans un dictionnaire
* Fusion du dictionnaire des métadonnées et du binaire

Phase 3, restitution

* Le fichier original est stocké sur le cloud
* Le fichier JSON est stocké en local et sur le cloud
* Restitution du fichier JSON à l'utilisateur via une fenêtre graphique de téléchargement (si utilisation d'un navigateur)

Le fichier JSON retourné a alors la structure suivante :

```
{"META": {
      "fichiernom": "string",
      "type": "MIMEType",
      "taille": "int",
      "extension": ".string",
      "metaparticulière": "{string}"
      },
   "fichier_bytes": "binaire"}
```

Gestion des extensions

La gestion des extensions s'est révélée relativement basique. Le stockage du fichier en JSON se faisant sur la base d'une lecture binaire de ce dernier, la limite d'utilisation du programme est sa capacité à gérer les métadonnées générales.

Le choix a été fait de traiter les extensions depuis leur nom que depuis leur MIMIType pour une plus grande flexibilité de traitement.

Les améliorations possibles du programme seraient d'augmenter la liste des extensions disponibles et un traitement à partir du MIMEType.

Le choix arbitraire de ne pas traiter les formats vidéos a été fait car il correspond de manière similaire à la gestion des images.

Gestion des métadonnées

La gestion des métadonnées va dépendre principalement de l'extension du fichier. On distingue trois catégories principales :

- Les images
- Les PDF
- Les autres format

Gestion des métadonnées générales

Les métadonnées suivantes sont générées pour l'ensemble des extensions prises en compte.

- nom du fichier
- MIMEType du fichier
- taille du fichier
- nom de l'extension

Gestion des images (.jpeg, .png, .jpg, .gif, .bmp)

Les librairies utilisées pour extraire les métadonnées des images sont :

- Pillow : permet d'ouvrir l'image en tant qu'une image et non comme un fichier *lambda*
- Exif : permet d'extraire les métadonnées si elles sont présentes. La nature principale de ces métadonnées coorespond au caractéristique de l'appareil photo ayant pris la photo.

AWS Rekognition

Si l'image correspond à une extension donnée et une taille minimum, elle est envoyée à l'API Amazon Rekognition qui est chargée de déterminer les éléments présents dans l'image. Les métadonnées déterminées sont alors ajoutées au fichier JSON.

Gestion des pdf

La lecture des métadonnées des PDF s'appuie sur la librairie PyPDF2. Cette librairie permet d'obtenir les informations de quatre natures différentes :

- DocumentInformation, pour obtenir les informations générales du PDF
- XMPInformation, pour obtenir les information XMP disponible
- getFields, pour obtenir les champs présents dans le PDF
- getNumPages, pour obtenir le nombre de page

Vous trouverez toutes les informations disponibles au lien suivant : https://pythonhosted.org/PyPDF2/Other%20Classes.html

Appel vers une autre API

Un module_ permettant d'appeller une autre API a été développé. Il permet d'appeller l'API sur le serveur HTTP pour convertir un fiche "text/plain".

Si l'appel et la conversion fonctionnent parfaitement quand l'API est en route, cela n'est plus le cas lorsque l'API HTTP est éteinte. En effet, un problème sur le buffer du 'request.files' n'a pas pu être géré.

Le buffer lit entièrement le fichier à convertir lors de l'appel à l'API et se retrouve en bout de fichier lorsqu'il passe à l'API HTTPS en cas d'échec de la première.

Module SSI

Introduction

Deux modules ont été partiellement ou complètement implémentés :

- Module HTTPS : un certificat temporaire a été mis en place pour une connexion certifiée
- Module authentification : mise en place d'un traitement des utilisateurs avec connexion par mot de passe à l'application

Fonctions développées

- Connexion SSH via clé et id/mdp, oui
- Protocole HTTPS, oui
- Connexion avec id/mdp à l'application, oui
- Gestion d'un nouvel utilisateur, oui

HTTPS

Dans un premier temps, il a été essayé d'obtenir un certificat à l'aide de *Letsencrypt*, cependant, le nom de domaine du serveur, lié à AWS a été refusé.

Dans un second temps, l'outil *openssl* a permi rapidement de fabriquer un certificat temporaire permettant alors le déploiement du serveur en version HTTPS.

Authentification

Dans le cadre de l'identification/authentification permettant un accès à l'application, plusieurs briques ont été développées :

- **Gestion admin** : l'application catégorise l'utilisateur en admin ou utilisateur *lambda*. L'admin a les droits (décidés arbitrairement) de récupérer des fichiers JSONifiés et/ou de les supprimer. Les droits sont testés en permanence lors de l'utilisation de l'application.

- **Gestion des mots de passe** : la présence en clair des mots de passe est réduite au minimum. Ainsi le mot de passe n'est utilisé en clair par l'application uniquement lors du test permettant de déterminer si il est ou non valide pour un nouvel utilisateur. Sinon, uniquement le hash du mot de passe est utilisé. Même l'admin n'a pas accès au mot de passe en clair.

- **Gestion des nouveaux utilisateurs** : il est possible de rentrer un nouvel utilisateur à condition que ces identifiant, mot de passe et adresse mail soient conformes. Un mail est alors envoyé à l'administrateur.

Il aurait été envisageable de proposer un générateur de mot de passe aléatoire que l'utilisateur aurait reçu par mail une fois son inscription terminée.

Module IAAS

Introduction

Présentation générale

Le module IAAS s'oriente autour deux modules :
- utilisation d'AWS avec le déploiement d'une instance EC2 et un bucket S3
- déploiement d'une application 'serverless' et de l'API Rekognition d'AWS.

Fonctions développées

- Serverless, en partie
- Gestion des métadonnées, oui
- Utilisation d'un bucket S3, oui
- Fichier test et commande de requête, oui
- Utilisation AWS Rekognition, oui

Serveless

La mise en place de ServerLess s'effectue via un environnement virtuel décrit dans la partie Serverless du Git.

https://github.com/AlanJBX/FilRouge/tree/master/Serverless

Si les deux parties fonctionnent indépendamment, un problème n'a pu être entièrement résolu. En effet, lors du traitement du fichier envoyé lors de la requête *CURL*, ce dernier n'est pas pris en compte par l'application directement. Il est nécessaire de sauvegarder localement le fichier avant de pouvoir le traiter correctement. Cependant, le serverless n'accepte pas cette sauvegarde locale.

J'ai donc implémenté le code différemment de l'application Flask pur afin de travailler avec un objet stocké puis récupéré sur le bucket S3. Si ce code fonctionne indépendamment, je n'ai pas pu le tester via la version Serverless. Des essais ont été fait sur l'ordinateur MAC de l'école et le serveur se lançait correctement, néanmoins, lors des nouvelles tentatives sur une machine virtuelle à domicile, il n'a pas été possible de relancer un serveur et d'effectuer les nouveaux tests de l'API.

Fichier de test

curl -X POST "http://54.246.242.159:8000/rekognition"
-F "data_file=@~/FilRouge/Testeurs/Ville.jpg"
> Ville_from_JPEG_to_JSON.json

Afin de tester le serverless, je vous propose de tester cette commande vous permettant d'obtenir l'ensemble des informations que peut traiter le programme. (mise à jour de l'adresse IP avant exécution)

Il sera nécessaire de corriger l'adresse IP et peut-être nécessaire de corriger à la marge le chemin d'accès au fichier de test.

Métadonnées

Les métadonnées récoltées sont de deux origines :

- métadonnées générales : nom du fichier, MIMEType du fichier et taille du fichier
- métadonnées particulières : objets détectés par l'API rekognition, voir la partie AWS Rekognition

AWS Bucket S3

Un bucket S3 a été déployé manuellement via le navigateur Web pour le stockage des fichiers d'origine et les fichiers JSONifiés. La gestion des suppressions périodiques a été effectuée de manière manuelle via le navigateur Web et sur le programme de création du ServerLess.

Un point d'amélioration serait l'automatisation de la création et de la gestion du bucket à partir d'un programme/script.

En théorie, la suppression des données est fixée à 365 jours dans le code de description du serverless.

De manière pratique, une suppression des données a été fixée à 365 jours via le navigateur.

AWS Rekognition

Dans le cadre du développement initial de l'API sur une instance EC2, un appel de l'API Rekognition d'Amazon a été mis en place. Cette application permet d'enrichir les métadonnées de l'image considérée. Vous trouverez son implémentation dans le code source de l'API.

Elle permet de renvoyer les objets détectés sur l'image considérée.

L'autorisation de connexion s'effectue en deux temps :

- Vérification des AWS Credentials Keys
- Vérification de l'autorisation temporaire délivrée par Rosetta

Module SOA

Introduction

Présentation générale

La gestion de l'API se fait de trois manières différentes :

- une documentation open-source de l'application : permettant à chaque de se l'approprier et de l'améliorer.
- une gestion API : via un API manager pour accéder à l'application
- un document swagger : permettant de rapidement obtenir les requêtes nécessaires à l'utilisation de l'API.

Fonctions développées

- Interface graphique, oui
- Commande CURL, oui
- API Manager, oui
- API Limitation requête, oui
- API Sécurisation, oui
- API Interconnectée, oui
- API Documentation, oui

Interface graphique

Avant la mise en place des requêtes via CURL, il m'a semblé plus facile d'utiliser une interface graphique qui me permettrait d'accéder rapidement aux fichiers que je souhaitais tester dans le cadre de la JSONification. J'ai, par conséquent développé, une première interface graphique permettant le chargement du fichier à convertir. Par la suite et afin d'améliorer l'expérience utilisateur, j'ai mis en place les différentes pages disponibles.

Vous trouverez ci-dessous les interfaces relatives au choix de la fonction désirée ainsi que la page du convertisseur.

Commandes CURL

L'ensemble des commandes CURL disponibles pour l'application sont disponibles à l'adresse suivante : 'http://54.246.242.159:8000/swagger/'

La génération des commandes CURL en HTTPS ne se sont pas montrées concluantes et je n'ai pas eu le temps de débugger le programme. Il semblerait que cela soit lié aux cookies qu'utilise l'application dans la gestion de l'authentification de l'utilisateur et des LOGs (la gestion de l'HTTPS se fait facilement avec CURL avec le rajout de l'option *-k* à la ligne de commande)

API Manager

L'API Manager suivant a été mis en place pour regrouper nos APIs et les gérer : https://52.51.220.151:9443/devportal/apis

L'API relative à l'application developpée ici est : https://52.51.220.151:9443/devportal/apis/e28ff3c8-9755-4d38-ab6e-8c3117bf18b8/overview

Attention néanmoins à la partie de test du CURL, en effet, l'adresse fournie pour le site ne correspond pas à l'adresse IP du serveur.

API Limitation requête

À l'aide de l'API Manager, une limitation à 1000 requêtes par minute est choisie.

API Sécurisation

À l'aide de l'API Manager et d'un compte WSO2, il est possible d'accéder à l'API via un Token préalablement demandé.

API interconnectée

L'application développée est connectée au Bucket S3 d'Amazon et appelle l'API Rekognition AWS.

Une tentative de connexion à une autre API de conversion de fichier en JSON a été également implémenté.

Documentation

La documentation de l'application est constituée du présent document (principalement la partie Python) ainsi que la page swagger de l'application http://54.246.242.159:8000/swagger/