

# Rapport Projet Fil Rouge

## I. Préambule

Ce rapport technique est la restitution de mon travail effectué autour du projet Fil Rouge. A des fins pédagogiques, il regroupe l'ensemble des projets d'évaluation de plusieurs cours dispensés au cours de l'année :

- Concept IPv4, enseigné par M. Laissus,
- Langage Python pour les Data Sciences, enseigné par M. Larroque
- IaaS privé et public, enseigné par M. Joga,
- Design pattern et architecture, enseigné par M. Rognon,
- IPv6 et Cryptographie, enseigné par M. LegrandGérard.

Les livrables du projet autre que ce rapport sont accessibles sur le *repository* GitHub <https://github.com/DavidQuarz/FRG.git>.

## Table des matières

I.	Préambule .....	1
II.	Concept IPv4 & Langage Python pour les Data Sciences .....	3
A.	Structure de fichier de l'application .....	3
B.	Codage de l'API.....	4
C.	Infrastructure & Architecture.....	5
D.	Procédure de déploiement de l'application.....	6
1.	Connexion à la machine virtuelle .....	6
2.	Installation de prérequis .....	6
3.	Téléchargement de l'application.....	6
4.	Déploiement de l'application .....	7
5.	Paramétrage des variables d'environnement .....	7
E.	Requêtage de l'API .....	7
III.	IaaS privé et public .....	8
A.	Procédure de déploiement sur serverless.....	8
1.	Prérequis .....	8
2.	Création du fichier serverless.yml.....	9
3.	Déploiement sur serverless .....	9
B.	Requêtage de l'API serverless .....	10
1.	Requête à l'API .....	10
2.	Retour de l'API.....	10
C.	Archivage dans S3.....	11
D.	Suppression du fichier au bout d'un an.....	11
IV.	Design pattern et architecture .....	13
A.	Mise en place de Swagger .....	13
B.	Architecture.....	13
C.	Accès et utilisation de Swagger .....	15
D.	Déclaration de l'API sur un API manager.....	16
E.	Utilisation de l'API via API Manager .....	17
F.	IPv6 et Cryptographie .....	20
1.	Prérequis .....	20
2.	Création d'un VPC et de sous-réseaux .....	20
3.	Configuration d'un sous-réseau Public.....	20
4.	Lancement d'une instance dans votre sous-réseau public .....	21

## II. Concept IPv4 & Langage Python pour les Data Sciences

**Objectif du projet :** Réaliser une API de type REST en python, accessible sur un serveur distant hébergé dans le cloud (RosettaHUB).

### Contraintes infrastructures :

- Système d'exploitation de la famille des « UNIX libres »,
- Infrastructure hébergée dans le cloud (RosettaHUB),
- API accessible à distance,
- Machine accessible par ssh à des fins de maintenance,
- (Option) Usage du « packet filter » pour protéger et limiter les accès à l'API, notamment limiter le nombre de requêtes par secondes pour les usagers,
- (Option) Usage du protocole https au lieu de http associé à une identification des usagers,
- (Option) Usage de l'OS FreeBSD

### Contraintes de conception et de réalisation logicielle :

L'API devra accepter le dépôt de de tout type de fichier et le restituer au format JSON. L'utilisateur soumet un fichier et récupère sa traduction en JSON associée à des métadonnées établies lors de sa traduction. Il comportera les caractéristiques/fonctionnalités suivantes :

- API de type RESTFULL,
- Retour de l'API séparant les métadonnées des données,
- Fichier minimum à traiter : texte (.txt, .pdf), nombre (.csv) et images,
- En cas de format non supporté, retour d'une erreur à l'utilisateur,
- Service implémenté en Python (3.x) avec Flask comme moteur web,
- (Option) Ajout de formats supplémentaires en plus des minimum requis.

### A. Structure de fichier de l'application

L'application est décomposée en plusieurs modules et fichiers au sein du répertoire racine *FRG/*. Ci-dessous son organisation. Ne sont listés que les fichiers utiles à la conception de l'API. Les autres fichiers relatifs aux autres cours dispensés ne sont pas affichés :

```
FRG/
  app/
    api/
      __init__.py
      api_routes.py
    main/
      routes.py
      __init__.py
      models.py
  venv/
  fichier_test/
  .flaskenv
  config.py
  frg.py
  requirement.txt
```

← environnement virtuel  
 ← ensemble des fichiers à tester  
 ← variables d'environnement de FLASK  
 ← liste des packages et dépendances

Il est à noter que le répertoire *main/* ne sert fonctionnellement pas à l'API. Il est conservé à titre informatif et pourrait être utilisé plus tard pour coder l'interface web du service sans passer par l'API.

## B. Codage de l'API

L'ensemble des scripts python est accessible depuis le *repository* sur GitHub.

### FRG/frg.py : Script principal lancé par FLASK

Ce script python importe l'ensemble du module *app/* qui contient l'application codée en python. Il s'agit du fichier qui est passé en argument lors du lancement de Flask depuis le terminal.

### FRG/config.py : Fichier de configuration de Flask

Ce script python contient la classe *Config* qui stocke les variables de configuration de l'application. Au lancement de Flask, l'application est configurée avec les paramètres définis dans la classe *Config*.

### FRG/app/\_\_init\_\_.py : Instanciation de l'application Flask

Ce script python instancie l'application Flask et appelle sa configuration via le script *config.py*.

### FRG/app/models.py : Définition de la classe *File*

Ce script python contient la classe *File*. L'ensemble des méthodes qui permettent de déterminer les métadonnées et les données d'un fichier sont contenues dans cette classe :

- Une méthode pour déterminer les métadonnées du fichier soumis par l'utilisateur (*filename, mimetype, get\_size()*)
- Des méthodes pour traiter les données contenues dans les fichiers soumis en fonction de leur format (texte, images, csv, json)
- Une méthode pour indiquer que le format n'est pas valide
- Une méthode pour retourner les métadonnées et données au format JSON.

#### Traitement des fichiers texte

Le contenu des fichiers texte est lu au format utf-8 et est restitué sans transformation.

#### Traitement des fichiers image

En l'état, le contenu des images ne peut être inséré dans un fichier JSON. Il est ainsi encodé dans une chaîne de caractères utilisant 64 caractères grâce à la méthode *b64encode()*. Cette chaîne est ensuite insérée en sortie du format JSON.

#### Traitement des fichiers csv

Le contenu des fichiers csv est traité ligne à ligne avec la méthode *DictReader()* de la classe csv.

#### Traitement des fichiers json

Le contenu des fichiers json est traité avec la méthode *loads()* de la classe json.

#### FRG/app/api/\_\_init\_\_.py : Définition d'un *Blueprint*

Ce script python permet d'encapsuler les éléments liés à l'API dans un package python séparé.

#### FRG/app/api/api\_routes.py : Définition des routes

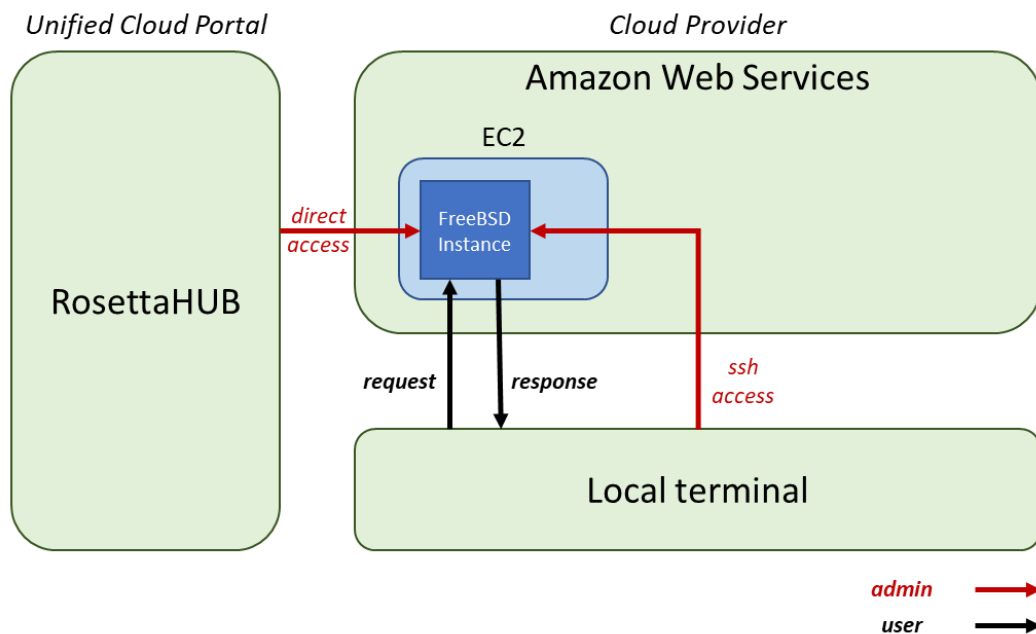
Ce script python permet de définir les routes d'accès à l'API. La soumission des fichiers par l'utilisateur se fait par la méthode POST et admet les formats (*mimetype*) de fichiers suivants :

- *text/plain* : fichier texte
- *text/csv* : fichier csv
- *application/pdf* : fichier pdf
- *application/json* : fichier json
- *image/jpeg* : fichier image jpeg
- *image/png* : fichier image png
- *image/gif* : fichier image gif

Lorsque l'application reconnaît un de ses formats de fichier, elle oriente la lecture du contenu du fichier vers la bonne méthode (contenu dans FRG/app/models.py) et retourne un code http 201. Dans le cas contraire, un code erreur http 415 est renvoyé.

### C. Infrastructure & Architecture

L'application est hébergée dans le *cloud* sur AWS dont l'accès se fait via RosettaHUB. L'application Flask est déployée sur une machine virtuelle tournant sous FreeBSD 12.1. Celle-ci est créée à partir du service *cloud* EC2 fourni par AWS.



L'image du système d'exploitation FreeBSD 12.1 est accessible dans « AMI de la communauté » lors de la création de l'instance.

Lorsque l'application Flask est lancée sur la machine virtuelle, la soumission des fichiers est faite avec la commande *curl* ou via l'application Postman.

## D. Procédure de déploiement de l'application

### 1. Connexion à la machine virtuelle

#### Connexion via RosettaHub

L'accès à la machine virtuelle se fait par RosettaHUB. Celui-ci donne un accès direct à AWS et à son service EC2. Se connecter via RosettaHUB permet de démarrer la machine virtuelle et de s'y connecter en ssh par le navigateur.

#### Connexion à distance via ssh

Cet accès suppose que la machine virtuelle est déjà démarrée auquel cas il est nécessaire de le faire (voir Connexion via RosettaHUB). Ce type d'accès permet de se connecter à la machine virtuelle depuis n'importe quel terminal de commande à condition d'avoir en local la clé privée *frg2020.pem*.

Pour lancer la connexion en ssh, se positionner dans le dossier où est située la clé privée et lancer la commande :

```
ssh -i "frg2020.pem" ec2-user@ <DNS-public-IPv4>
```

<DNS-public-IPv4> est le nom d'hôte attribué par AWS lors du lancement de la machine virtuelle et change à chaque nouveau démarrage. Il est nécessaire de le connaître pour lancer la connexion à distance. Par ailleurs il est nécessaire d'ouvrir le port 22 au niveau des paramètres de la machine virtuelle.

Remarque : Sur FreeBSD, il n'est pas possible de se connecter en *root* directement. C'est la raison pour laquelle la connexion se fait avec *ec2-user*.

### 2. Installation de prérequis

Il est nécessaire d'installer des prérequis avant de déployer l'application :

```
#Passage en root
su

#Mise à jour des dépôts pkg
pkg update -f

#Installation de python3
pkg install python3

#Installation de pip
pkg install py37-pip
pkg install --upgrade pip

#Installation de git
pkg install git
```

### 3. Téléchargement de l'application

Le code source de l'application est hébergé sur GitHub. Pour le récupérer, il est nécessaire de le télécharger avec la commande :

```
git clone https://github.com/DavidQuarz/FRG.git
```

L'ensemble du code source est contenu dans le dossier FRG.

#### 4. Déploiement de l'application

Il est nécessaire de se positionner dans le dossier FRG puis de créer un environnement virtuel pour y installer tous les packages et dépendances liés à l'application.

```
#Création d'un environnement virtuel
python3 -m venv venv

#Activation de l'environnement virtuel
. venv/bin/activate

#Installation des packages et dépendances
pip install -r requirements.txt
```

#### 5. Paramétrage des variables d'environnement

Le fichier .flaskenv contient les variables d'environnement pour l'application FLASK. Ouvrir le fichier et s'assurer que les variables d'environnement sont définies telles que ci-dessous :

```
FLASK_APP=frg.py
FLASK_RUN_HOST=0.0.0.0
FLASK_RUN_PORT=5000
```

FLASK\_APP permet de définir le script python qui sera exécuté au lancement de Flask. En exécutant *frg.py*, celui-ci importera tous les modules utiles à l'application.

FLASK\_RUN\_HOST permet de définir l'adresse IP sur lequel l'application sera exécutée. Par défaut, Flask se lance en *localhost* (127.0.0.1) et pour le rendre ainsi visible sur le réseau, il est nécessaire de ne pas le lancer en *localhost* mais en 0.0.0.0.

FLASK\_RUN\_PORT permet de définir le port d'exposition de l'application. Par défaut, Flask se lance sur le port 5000, port qui ne sera conservé. En outre, il est nécessaire d'ouvrir le port 5000 au niveau des paramètres de l'instance EC2.

#### E. Requêtage de l'API

L'envoi des fichiers à l'API peut se faire grâce à la commande *curl* ou bien le client Postman (qui ne sera pas détaillé).

A partir d'un terminal de commande (non connecté à la machine virtuelle par ssh), il est possible d'envoyer un fichier avec la méthode POST :

```
curl \
--request POST '<DNS-public-IPv4>:5000/api/files' \
--form 'file=@/PATH/TO/FILE'
```

#### Cas spécifique pour les fichiers .csv et .json

Pour les fichiers .csv et .json, il est nécessaire de spécifier le type de fichier avec le paramètre *--form*. Cette précision supplémentaire indique le type de fichier à la commande *curl*. Dans le cas contraire, le type de fichier ne sera pas reconnu et sera par défaut *application/octet-stream*.

csv	--form 'file=@/PATH/TO/FILE;type=application/json'
json	--form 'file=@/PATH/TO/FILE;type=text/csv'

Le dossier FRG/fichier\_test/ contient plusieurs formats de fichier qui sont acceptés par l'application. A vous de les tester !

### III. IaaS privé et public

**Objectif du projet :** Mettre à disposition l'API en serverless capable de décrire un type de fichier

**Consignes :**

- En plus du dossier fil rouge, fournir un document succinct avec les consignes d'utilisation de l'API. Spécifier le résultat attendu,
- Utiliser *curl* requêter l'API en serverless,
- Fournir un fichier de test idéal à soumettre à l'API,
- Donner un accès à l'API depuis Internet,
- A l'aide de code source, rendre redéployable l'application à l'aide de serverless sur un autre compte AWS moyennant quelques possible changements de paramètres.

#### A. Procédure de déploiement sur serverless

##### 1. Prérequis

##### Installation et mise à jour de nodejs

Avant de déployer l'application sur serverless, il est nécessaire d'installer et de mettre à jour nodejs :

```
#Installation de nodejs
pip install nodejs

#Mise à jour de nodejs
curl -sL https://deb.nodesource.com/setup_13.x | sudo -E bash -
sudo apt-get install -y nodejs
```

La mise à jour de nodejs est essentielle sans quoi les packages suivants ne pourront être installés correctement.

##### Initialisation du fichier `package.json`

Se positionner dans le répertoire racine de l'application et entrer la commande :

```
npm init -f
```

##### Installation de dépendances pour serverless

Pour utiliser Flask avec serverless, nous utilisons deux plugins *serverless-wsgi* et *serverless-python-requirements* :

- *serverless-wsgi* permet de déployer les applications Python WSGI avec serverless. Il fonctionne avec les *framework* Flask, Django et Pyramid,
- *serverless-python-requirements* permet de déployer les packages contenus dans le fichier *requirements.txt* qui sont nécessaires au lancement de l'application avec serverless.

```
npm install --save-dev serverless-wsgi serverless-python-requirements
```



## 2. Création du fichier `serverless.yml`

Dans le répertoire racine de l'application, créer un fichier `serverless.yml` qui contiendra les informations ci-dessous. Pour ne pas alourdir le déploiement sur serverless, il est nécessaire d'exclure les packages `node_modules/` et `venv/`.

```
# serverless.yml
service: serverless-FRG

plugins:
  - serverless-python-requirements
  - serverless-wsgi

custom:
  wsgi:
    app: app.app
    packRequirements: false
    pythonRequirements:
      dockerizePip: non-linux

package:
  exclude:
    - node_modules/**
    - venv/**

provider:
  name: aws
  runtime: python3.7
  stage: dev
  region: eu-west-1

functions:
  app:
    handler: wsgi.handler
    events:
      - http: ANY /
      - http: 'ANY {proxy+}'
```

## 3. Déploiement sur serverless

L'application est prête à être déployée sur serverless avec la commande :

```
sls deploy
```

Le résultat attendu figure ci-dessous. En particulier, le lien d'accès à serverless est communiqué au niveau de la ligne « endpoints ». L'API sera requêtée à partir de cette URL en y ajoutant les routes nécessaires.

```
Service Information
service: serverless-FRG
stage: dev
region: eu-west-1
stack: serverless-FRG-dev
resources: 12
api keys:
  None
endpoints:
  ANY - https://y4dq8zt776.execute-api.eu-west-1.amazonaws.com/dev
  ANY - https://y4dq8zt776.execute-api.eu-west-1.amazonaws.com/dev/{proxy+}
functions:
  app: serverless-FRG-dev-app
layers:
  None
Serverless: Removing old service artifacts from S3...
Serverless: Run the "serverless" command to setup monitoring, troubleshooting and testing.
```

## B. Requêtage de l'API serverless

### 1. Requête à l'API

L'envoi d'un fichier à serverless se fait avec la commande *curl*. L'application *Postman* peut également être utilisé pour la requête. Ci-dessous la route d'accès à l'API

Méthode	Routes
POST	/api/files

La commande *curl* à utiliser doit avoir la structure ci-dessous :

```
curl --request POST '<URL-Endpoint>/api/files' --form 'file=@<PATH/TO/FILE>
```

Lors du redéploiement de serverless sur un autre compte AWS, l'*endpoint* sera modifié et devra être adapté en fonction du retour du déploiement de serverless (ligne *endpoints*).

#### Lien de test

Pour le besoin de cette évaluation le lien de test est le suivant :

<https://0h94qcood3.execute-api.eu-west-1.amazonaws.com/dev/api/files>

#### Fichier de test

Le fichier à tester se trouve dans le dossier FRG/fichier\_test/ et est nommé « application\_json.json ». Le cas particulier des fichiers json et csv est qu'il est nécessaire d'ajouter une précision supplémentaire avec le paramètre *--form*. Cette précision supplémentaire indique le type de fichier à la commande *curl*. Dans le cas contraire, le type de fichier ne sera pas reconnu et sera par défaut *application/octet-stream*.

csv	<code>--form 'file=@/PATH/TO/FILE;type=application/json'</code>
json	<code>--form 'file=@/PATH/TO/FILE;type=text/csv'</code>

### 2. Retour de l'API

Le retour attendu par l'API une chaîne de caractère au format JSON qui spécifie :

- Le contenu du fichier : *content*
- Le nom du fichier : *name*
- La taille du fichier : *size*
- Le type de fichier : *type*

Ci-dessous le retour attendu du fichier test « application\_json.json ».

```
quarz@quarz-ubuntu:~/Documents/FRG$ curl --request POST 'https://y4dq8zt76.execute-api.eu-west-1.amazonaws.com/dev/api/files' -H "accept: application/json" -H "Content-Type: multipart/form-data" --form 'file=@/home/quarz/Documents/FRG/fichiers_test/application_json.json;type=application/json'
{"content":{"active":true,"formed":2016,"homeTown":"Metro City","members":[{"age":29,"name":"Molecule Man","powers":["Radiation resistance","Turning tiny","Radiation blast"],"secretIdentity":"Dan Jukes"}, {"age":39,"name":"Madame Uppercut","powers":["Million tonne punch","Damage resistance","Superhuman reflexes"],"secretIdentity":"Jane Wilson"}, {"age":1000000,"name":"Eternal Flame","powers":["Immortality","Heat Immunity","Inferno","Teleportation","Interdimensional travel"],"secretIdentity":"Unknown"}],"secretBase":"Super tower","squadName":"Super hero squad"},"name":"application_json.json","size":827,"type":"application/json"}
```

### C. Archivage dans S3

Le stockage dans s3 se fait grâce au package *boto3* qui fournit une API orientée objet facile à utiliser, ainsi qu'un accès de bas niveau aux services AWS. La méthode *upload\_fileobj()* permet de déposer sur un *bucket* s3 un objet de type fichier. Celui-ci doit être lu en mode *binary* pour l'être.

Le script python vérifie si le nom de la *bucket* existe dans le cas contraire, il crée la *bucket* au nom spécifié. Dans le cadre du projet, la *bucket* est nommée « bucket-frg ». Attention, ce nom peut être changé mais ne doit contenir que :

- Des lettres en minuscule sans accent,
- Aucun caractère spécial hormis « - »,
- Aucun espace.

```
#Stockage du fichier json dans S3
s3 = boto3.client('s3')
buckets = s3.list_buckets()
bucket_FRG = 'bucket-frg'

existing_bucket = False
for bucket in buckets['Buckets']:
    if bucket_FRG == bucket["Name"]:
        existing_bucket = True

if not existing_bucket:
    s3.create_bucket(Bucket=bucket_FRG, CreateBucketConfiguration={'LocationConstraint':'eu-west-1'})

file_to_s3 = BytesIO(json.dumps(Response.get_json(reponse), sort_keys=True, indent=4).encode('utf-8'))
s3.upload_fileobj(file_to_s3, bucket_FRG, file.name+'.json')
```

Il est également nécessaire d'ajouter dans le fichier *serverless.yml* un accès à s3. Dans la section *provider*, il faut ajouter les lignes suivantes :

```
#...

provider:

    #...

    iamRoleStatements:
        - Effect: Allow
          Action:
            - s3:PutObject
            - s3:GetObject
            - s3>CreateBucket
            - s3:ListAllMyBuckets
          Resource: 'arn:aws:s3:::*'

    #...
```

### D. Suppression du fichier au bout d'un an

La suppression des fichiers au bout d'un an se configure au niveau du *bucket* dans l'onglet « Gestion », sous-onglet « Cycle de vie », bouton « Ajouter une règle de cycle de vie ».

#### Nom de portée

- Saisir un nom de règle
- Cocher : "S'appliquer à tous les objets du compartiment"

## Transitions

- Ne rien cocher

## Expiration

- Cocher "Version actuelle" : faire expirer la version actuelle de l'objet après 365 jours suivant la création de l'objet.
- Cocher "Versions précédentes" : Supprimer définitivement les versions précédentes 1 jour après être devenue une version précédente

Avec cette règle de cycle de vie, tous les fichiers déjà présents dans la *bucket* expireront après 365 jours et seront définitivement supprimés 1 jour après leur expiration. Les nouveaux fichiers déposés dans la *bucket* suivront la même règle mais à compter de leur date de dépôt.

bucket-frg

Présentation	Propriétés	Autorisations	Gestion	Points d'accès
Cycle de vie	Réplication	Analyse	Métriques	
Inventaire				
<a href="#">+ Ajouter une règle de cycle de vie</a> <a href="#">Modifier</a> <a href="#">Supprimer</a> <a href="#">Actions</a>				

Règle de cycle de vie	Appliquée à	Transitions pour la version actuelle	Transitions pour la ou les versions précédentes
<input checked="" type="checkbox"/> delete_after_365_days	Compartiment entier	Expirer	Permanently Delete

Sur la bucket « bucket-frg », cette règle a été mise en place sous le nom de "delete\_after\_365\_days".

## IV. Design pattern et architecture

### Objectif du projet :

- Mettre à disposition l'API respectant le standard OpenAPI.
- Publier l'API sur un API manager

#### A. Mise en place de Swagger

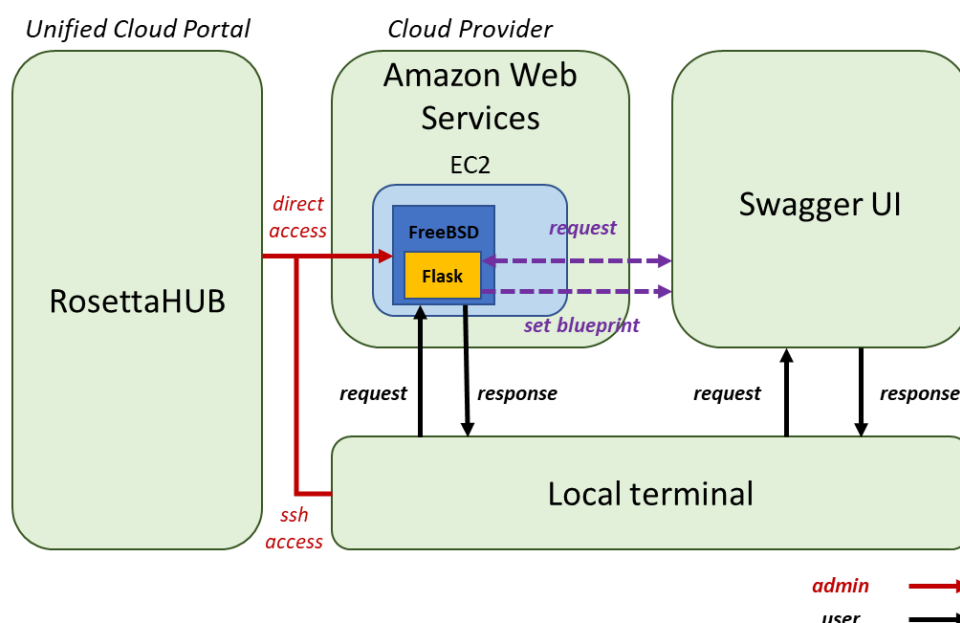
Pour lier Swagger à notre API et qu'il permette de le documenter, nous utilisons le package flask-swagger-ui accessible au lien suivant <https://github.com/sveint/flask-swagger-ui.git>.

Pour permettre de connecter Swagger à Flask, les ajouts suivants sont à faire dans les scripts python de l'application.

FRG/app/___init___.py : Instanciation de l'application Flask	
<pre> ### swagger specific ### from app.api import SWAGGERUI BLUEPRINT, SWAGGER_URL app.register_blueprint(SWAGGERUI_BLUEPRINT, url_prefix=SWAGGER_URL) ### end swagger specific ### </pre>	<ul style="list-style-type: none"> <li>▪ Enregistrement du <i>Blueprint</i> de Swagger</li> </ul>
FRG/app/api/___init___.py : Définition d'un <i>Blueprint</i>	
<pre> SWAGGER_URL = '/swagger' API_URL = './static/swagger.yaml'  swagger_yaml = load(open(API_URL, 'r'), Loader=Loader)  SWAGGERUI_BLUEPRINT = get_swaggerui_blueprint(     SWAGGER_URL,     API_URL,     config={         'app_name': "Quarz API REST FRG",         'spec': swagger_yaml     } ) </pre>	<ul style="list-style-type: none"> <li>▪ Configuration de la route d'accès</li> <li>▪ Configuration de l'URL de l'API</li> <li>▪ Mise en place du Swagger</li> </ul>

#### B. Architecture

Le package flask-swagger-ui construire l'interface graphique de Swagger UI à partir des informations contenues dans le fichier `/static/swagger.yaml`. Ce fichier définit les informations à afficher sur l'interface graphique mais aussi les différentes méthodes http liées à l'application Flask.



## Contenu du fichier swagger.yaml

```
swagger: "2.0"
info:
  description: "API documentation. Including FRG, SOA, AWS"
  version: "1.0.0"
  title: "File JSON Convertor"
  contact:
    email: "david.tia@student-cs.fr.com"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
tags:
- name: "Convert to JSON"
  description: "Supported format : .txt, .csv, .json, .pdf, .png, .jpeg, .gif"
  externalDocs:
    description: "Find out more"
    url: "http://swagger.io"
schemes:
- "http"
paths:
  /api/files:
    post:
      tags:
      - "Convert to JSON"
      summary: "uploads a file and convert it to JSON"
      description: ""
      operationId: "uploadFile"
      consumes:
      - "multipart/form-data"
      produces:
      - "application/json"
      parameters:
      - name: "file"
        in: "formData"
        description: "file to upload"
        required: false
        type: "file"
      responses:
        201:
          description: "successful operation"
        415:
          description: "supported media type"
  externalDocs:
    description: "Find out more about Swagger"
    url: "http://swagger.io"
```

Le fichier définit entre autres la méthode POST qui permet de soumettre un fichier à l'application et la route qui la lie à l'application.

### C. Accès et utilisation de Swagger

L'accès à Swagger suppose que l'application Flask est lancée (voir §II.D). Elle est accessible à l'adresse suivante : <DNS-public-IPv4>:5000/swagger/, où <DNS-public-IPv4> est le nom du serveur donnée au moment du lancement de la machine virtuelle.

L'interface graphique de swagger permet au même titre que l'application Flask de soumettre un fichier à convertir en json. Il propose également la commande *curl* associée au requêtage de l'API.

Schemes

HTTP ▼

**Convert to JSON** Supported format : .txt, .csv, .json, .pdf, .png, .jpeg, .gif

Find out more: <http://swagger.io> ▼

**POST** `/api/files` uploads a file and convert it to JSON

Cancel

**Parameters**

Name	Description
file	file to upload
file (formData)	<div style="border: 1px solid #ccc; padding: 5px; display: inline-block;"> Choisir un fichier application_json.json </div>

Execute

Clear

**Responses**

Response content type

application/json ▼

**Curl**

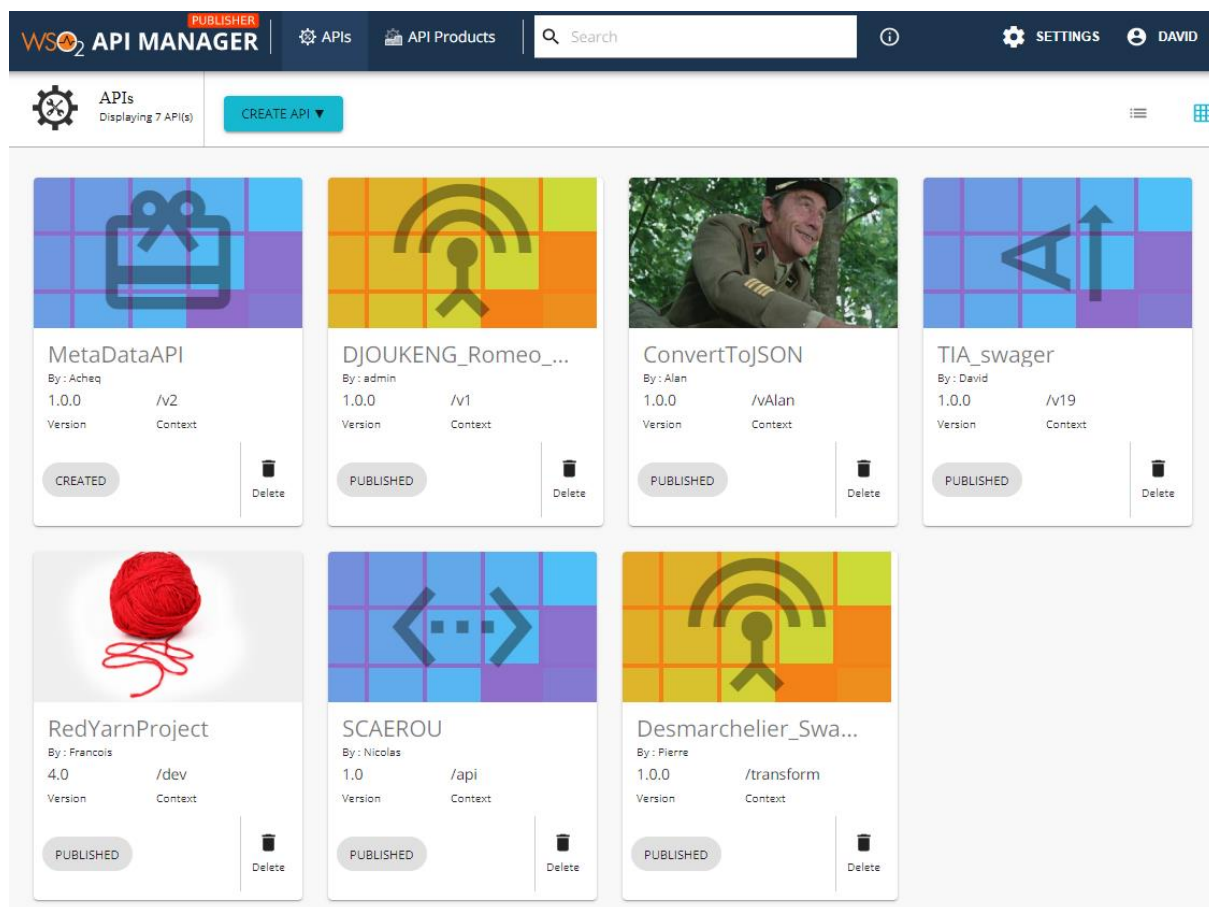
```
curl -X POST "http://ec2-54-77-173-59.eu-west-1.compute.amazonaws.com:5000/api/files" -H "accept: application/json" -H "Content-Type: multipart/form-data" -F "file=@application_json.json;type=application/json"
```

**Request URL**

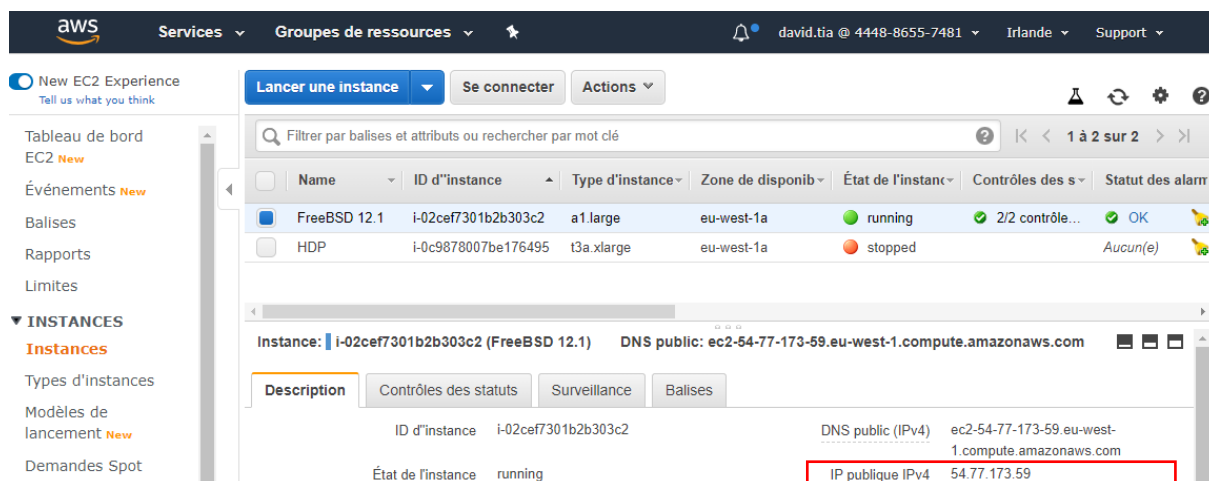
```
http://ec2-54-77-173-59.eu-west-1.compute.amazonaws.com:5000/api/files
```

## D. Déclaration de l'API sur un API manager

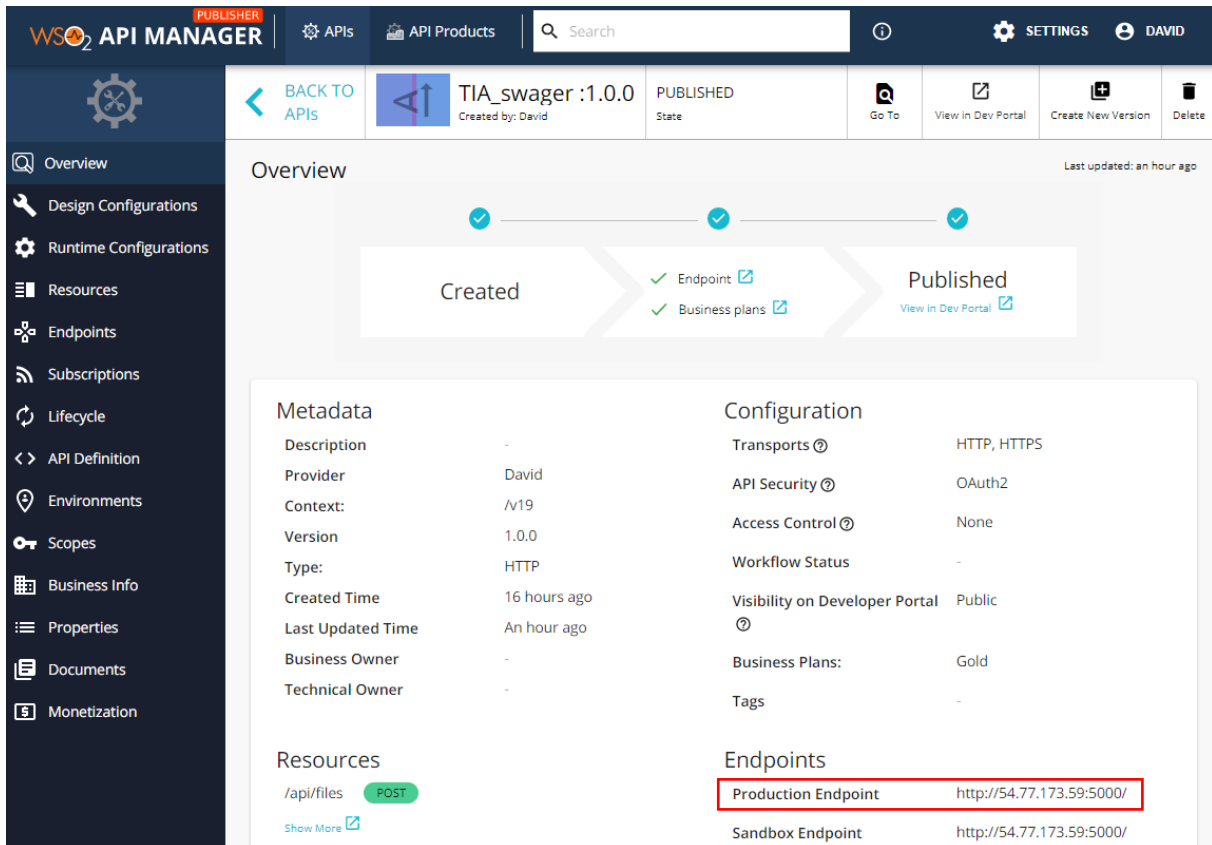
L'API est publiée sur WSO2 à l'adresse suivante : <https://52.51.220.151:9443/publisher/apis> au nom de « TIA\_swagger ».



Pour pouvoir l'utiliser, il est impératif que le *endpoint* de l'API fasse référence à l'IP publique IPv4 de la machine virtuelle lancée sur AWS et au port 5000. L'application doit également y être lancée.

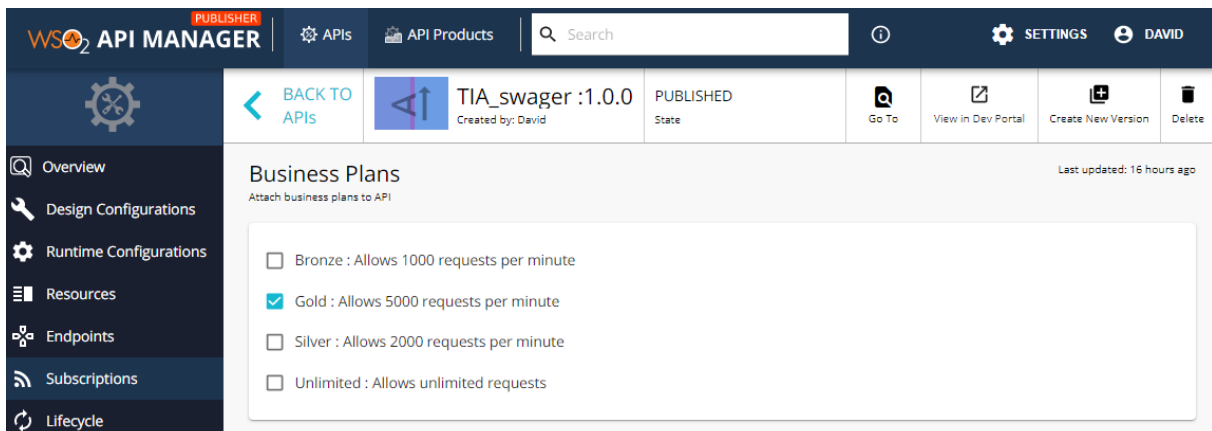






The screenshot shows the WSO2 API Manager interface. The left sidebar contains navigation options: Overview, Design Configurations, Runtime Configurations, Resources, Endpoints, Subscriptions, Lifecycle, API Definition, Environments, Scopes, Business Info, Properties, Documents, and Monetization. The main content area displays the 'Overview' for the API 'TIA\_swagger :1.0.0' (PUBLISHED State). A progress bar shows the lifecycle: Created → Endpoint/Business plans → Published. Below this, there are two sections: 'Metadata' and 'Configuration'. The 'Metadata' section lists details like Description, Provider (David), Context (/v19), Version (1.0.0), Type (HTTP), Created Time (16 hours ago), Last Updated Time (An hour ago), Business Owner, and Technical Owner. The 'Configuration' section lists Transports (HTTP, HTTPS), API Security (OAuth2), Access Control (None), Workflow Status, Visibility on Developer Portal (Public), Business Plans (Gold), and Tags. At the bottom, the 'Resources' section shows a POST endpoint for /api/files. The 'Endpoints' section lists the Production Endpoint (http://54.77.173.59:5000/) and the Sandbox Endpoint (http://54.77.173.59:5000/).

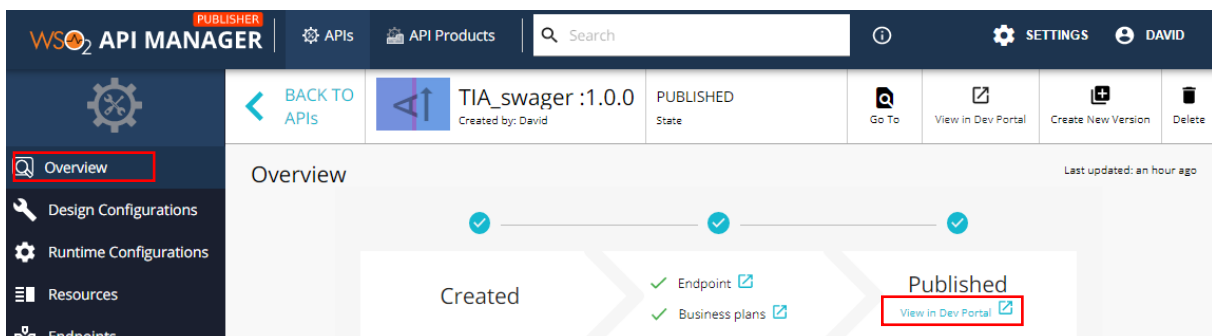
L'API est configurée pour permettre un maximum de 5000 requêtes par minute.



The screenshot shows the 'Business Plans' page for the API 'TIA\_swagger :1.0.0'. The page title is 'Business Plans' with a subtitle 'Attach business plans to API'. Below this, there is a list of business plans with checkboxes: Bronze (Allows 1000 requests per minute), Gold (Allows 5000 requests per minute, checked), Silver (Allows 2000 requests per minute), and Unlimited (Allows unlimited requests). The last updated time is 16 hours ago.

## E. Utilisation de l'API via API Manager

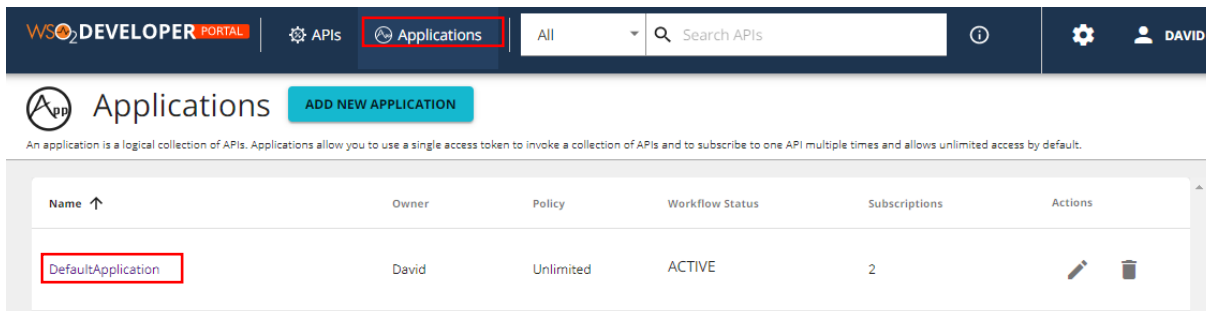
- Accéder au portail développeur



The screenshot shows the 'Overview' page for the API 'TIA\_swagger :1.0.0'. The 'Endpoints' section is visible, showing the Production Endpoint (http://54.77.173.59:5000/) and the Sandbox Endpoint (http://54.77.173.59:5000/). The 'View in Dev Portal' link is highlighted with a red box.

Attention, s'assurer que l'adresse IP, est bien sur 52.51.220.151:9443 et non sur localhost:9443.

### ▪ Accéder à Applications puis DefaultApplication



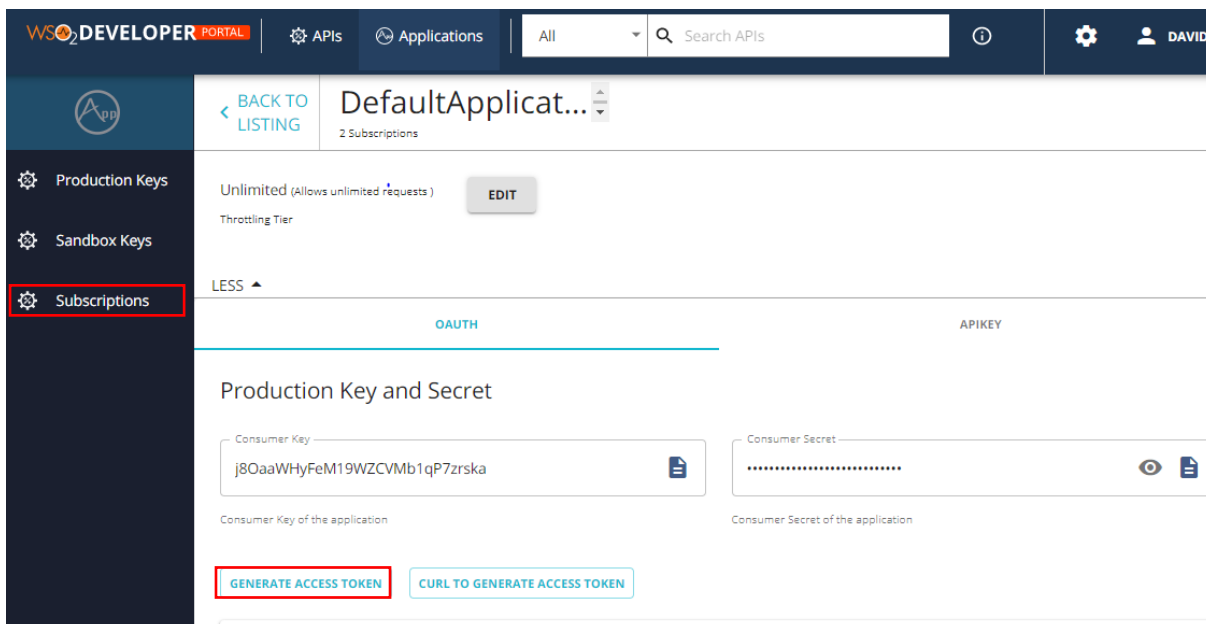
WSO2 DEVELOPER PORTAL | APIs Applications | All | Search APIs | DAVID

**Applications** ADD NEW APPLICATION

An application is a logical collection of APIs. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe to one API multiple times and allows unlimited access by default.

Name ↑	Owner	Policy	Workflow Status	Subscriptions	Actions
DefaultApplication	David	Unlimited	ACTIVE	2	

### ▪ Générer un Access Token, le copier puis accéder à Subscriptions



WSO2 DEVELOPER PORTAL | APIs Applications | All | Search APIs | DAVID

**DefaultApplication** 2 Subscriptions

Unlimited (Allows unlimited requests) EDIT

Throttling Tier

LESS ▲

OAuth APIKEY

Production Key and Secret

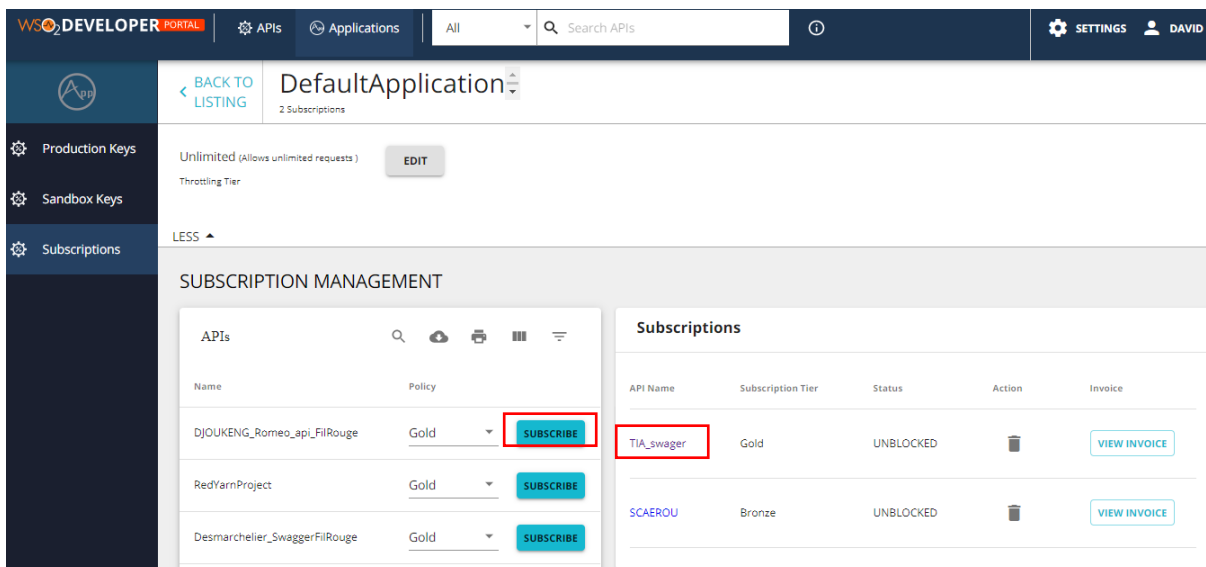
Consumer Key  
j8OaaWHyFeM19WZCVmb1qP7zrska

Consumer Secret  
.....

Consumer Key of the application Consumer Secret of the application

GENERATE ACCESS TOKEN CURL TO GENERATE ACCESS TOKEN

### ▪ Souscrire à l'API TIA\_swager puis accéder à l'API



WSO2 DEVELOPER PORTAL | APIs Applications | All | Search APIs | SETTINGS DAVID

**DefaultApplication** 2 Subscriptions

Unlimited (Allows unlimited requests) EDIT

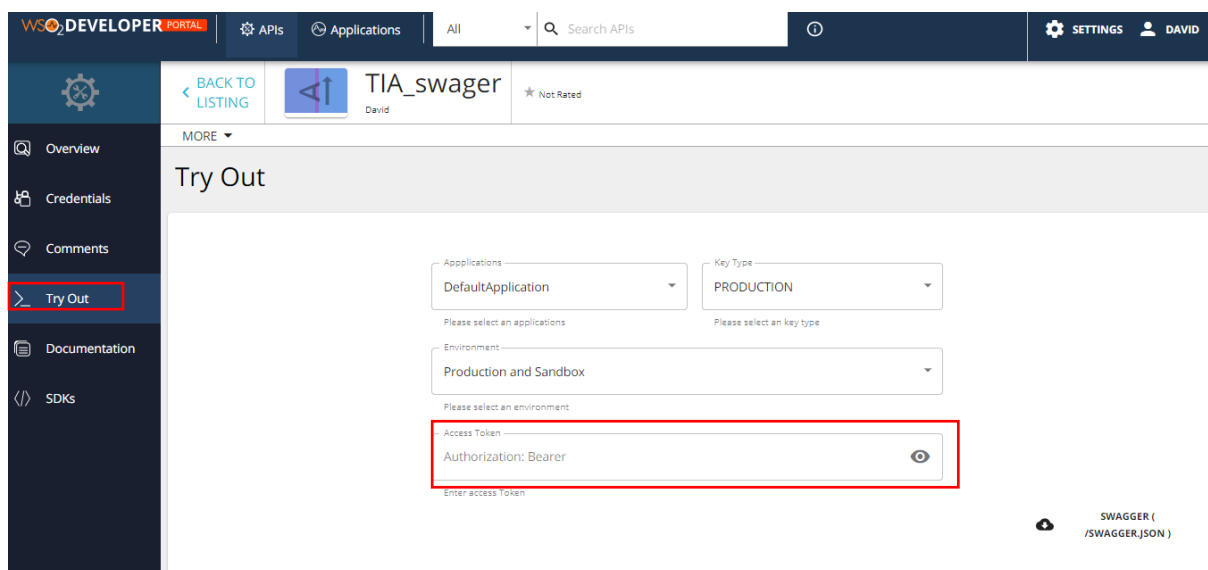
Throttling Tier

LESS ▲

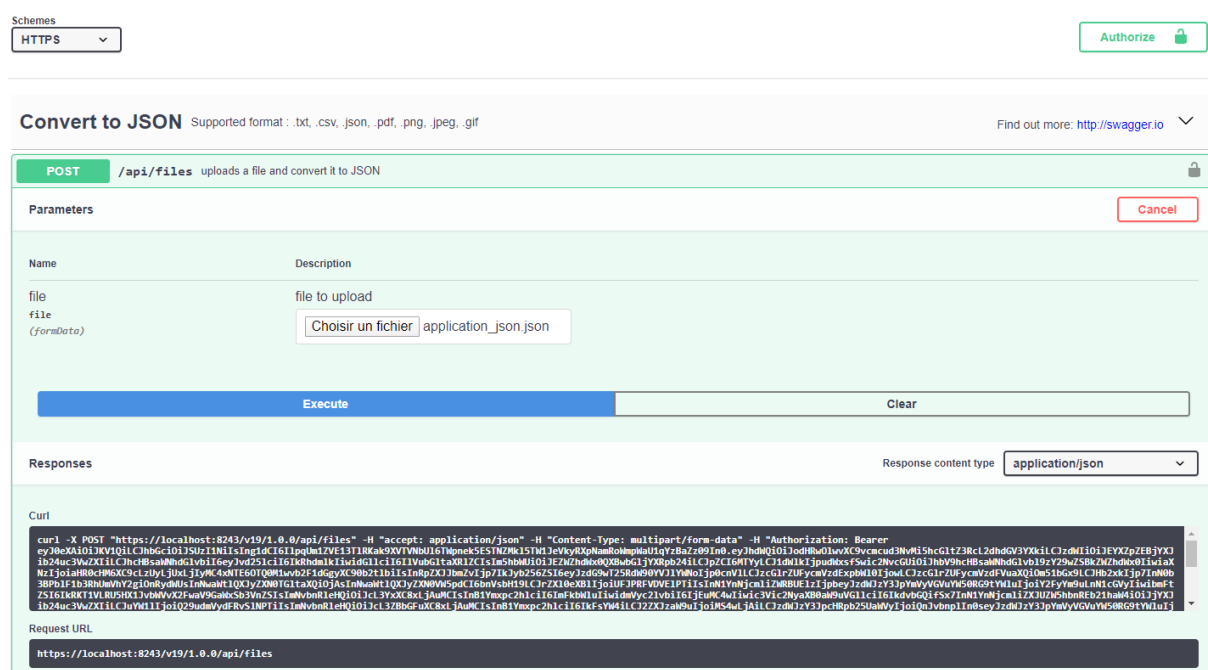
SUBSCRIPTION MANAGEMENT

APIs	Subscriptions																											
<table border="1"> <thead> <tr> <th>Name</th> <th>Policy</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>DJOUKENG_Romeo_api_FilRouge</td> <td>Gold</td> <td>SUBSCRIBE</td> </tr> <tr> <td>RedYarnProject</td> <td>Gold</td> <td>SUBSCRIBE</td> </tr> <tr> <td>Desmarchelier_SwaggerFilRouge</td> <td>Gold</td> <td>SUBSCRIBE</td> </tr> </tbody> </table>	Name	Policy	Action	DJOUKENG_Romeo_api_FilRouge	Gold	SUBSCRIBE	RedYarnProject	Gold	SUBSCRIBE	Desmarchelier_SwaggerFilRouge	Gold	SUBSCRIBE	<table border="1"> <thead> <tr> <th>API Name</th> <th>Subscription Tier</th> <th>Status</th> <th>Action</th> <th>Invoice</th> </tr> </thead> <tbody> <tr> <td>TIA_swager</td> <td>Gold</td> <td>UNBLOCKED</td> <td></td> <td><a href="#">VIEW INVOICE</a></td> </tr> <tr> <td>SCAEROU</td> <td>Bronze</td> <td>UNBLOCKED</td> <td></td> <td><a href="#">VIEW INVOICE</a></td> </tr> </tbody> </table>	API Name	Subscription Tier	Status	Action	Invoice	TIA_swager	Gold	UNBLOCKED		<a href="#">VIEW INVOICE</a>	SCAEROU	Bronze	UNBLOCKED		<a href="#">VIEW INVOICE</a>
Name	Policy	Action																										
DJOUKENG_Romeo_api_FilRouge	Gold	SUBSCRIBE																										
RedYarnProject	Gold	SUBSCRIBE																										
Desmarchelier_SwaggerFilRouge	Gold	SUBSCRIBE																										
API Name	Subscription Tier	Status	Action	Invoice																								
TIA_swager	Gold	UNBLOCKED		<a href="#">VIEW INVOICE</a>																								
SCAEROU	Bronze	UNBLOCKED		<a href="#">VIEW INVOICE</a>																								

▪ Accéder à Try Out puis coller l'*access token*



▪ Tester l'API proposée plus bas sur la même page



L'API pourra être testé à travers l'API Manager avec *curl* ou l'application Postman à l'URL communiquée par la réponse du Swagger. Il faudra remplacer *localhost* par 52.51.220.151.

Pour la commande *curl*, il faudra en plus préciser dans la commande le paramètre « *-H "Authorization : Bearer access\_token"* », *access\_token* étant le token communiqué précédemment.

## F. IPv6 et Cryptographie

**Objectif du projet :** Créer un Virtual Private Cloud (VPC) et des sous-réseaux IPv6

### 1. Prérequis

La création du VPC et des sous-réseaux se fait avec AWS CLI (*Command Line Interface*) et nécessite qu'il soit installé auparavant. Le bloc d'adresse utilisé sera un CIDR 10.0.0.0/4

### 2. Création d'un VPC et de sous-réseaux

1	Créer un VPC avec un bloc d'adresse CIDR 10.0.0.0/16 et associer un bloc d'adresse CIDR IPv6 avec le VPC. Noter l'ID du VPC <vpc-id>.
	<code>aws ec2 create-vpc --cidr-block 10.0.0.0/16 --amazon-provided-ipv6-cidr-block</code>
2	Décrire le VPC avec <vpc-id>. Noter la valeur de <Ipv6CidrBlock> retourné.
	<code>aws ec2 describe-vpcs --vpc-id &lt;vpc-id&gt;</code>
3	Créer un sous-réseau avec un bloc d'adresse CIDR IPv4 10.0.0.0/24 et un bloc d'adresse CIDR IPv6 avec la valeur <Ipv6CidrBlock>.
	<code>aws ec2 create-subnet --vpc-id &lt;vpc-id&gt; --cidr-block 10.0.0.0/24 --ipv6-cidr-block &lt;Ipv6CidrBlock&gt;</code>
4	Créer un second sous-réseau dans le VPC avec un bloc d'adresse CIDR IPv4 10.0.1.0/24 et un bloc d'adresse CIDR IPv6 avec la valeur <Ipv6CidrBlock>.
	<code>aws ec2 create-subnet --vpc-id &lt;vpc-id&gt; --cidr-block 10.0.1.0/24 --ipv6-cidr-block &lt;Ipv6CidrBlock&gt;</code>

### 3. Configuration d'un sous-réseau Public

1	Créer une passerelle Internet. Noter l'ID de la passerelle <gateway-id>.
	<code>aws ec2 create-internet-gateway</code>
2	Attacher la passerelle Internet à votre VPC.
	<code>aws ec2 attach-internet-gateway --vpc-id &lt;vpc-id&gt; --internet-gateway-id &lt;gateway-id&gt;</code>
3	Créer une table de routage personnalisée pour le VPC. Noter l'ID de la table de routage <routetable-id>
	<code>aws ec2 create-route-table --vpc-id &lt;vpc-id&gt;</code>
4	Créer une route dans la table de routage qui dirige tout le trafic IPv6 (::/0) vers la passerelle Internet.
	<code>aws ec2 create-route --route-table-id &lt;routetable-id&gt; --destination-ipv6-cidr-block ::/0 --gateway-id &lt;gateway-id&gt;</code>
5	Associer la table de routage à un sous-réseau du VPC pour que le trafic de ce sous-réseau soit acheminé vers la passerelle Internet. Noter l'ID des sous-réseaux afficher <subnet-id>
	<code>aws ec2 describe-subnets --filters "Name=vpc-id,Values=&lt;vpc-id&gt;" --query 'Subnets[*].{ID:SubnetId,IPv4CIDR:CidrBlock,IPv6CIDR:Ipv6CidrBlockAssociationSet[*].Ipv6CidrBlock}'</code>
6	Choisir un sous-réseau à associer à la table de routage personnalisée
	<code>aws ec2 associate-route-table --subnet-id &lt;subnet-id&gt; --route-table-id &lt;routetable-id&gt;</code>

#### 4. Lancement d'une instance dans votre sous-réseau public

1	Créer une paire de clés et utilisez l'option --query et l'option de texte --output pour diriger la clé privée directement dans un fichier avec l'extension .pem.
	<code>aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial' --output text &gt; MyKeyPair.pem</code>
2	Modifier les droits d'accès à la clé privée
	<code>chmod 400 MyKeyPair.pem</code>
3	Créer un groupe de sécurité <securitygroup-id> pour le VPC et ajoutez une règle qui autorise l'accès SSH à partir de n'importe quelle adresse IPv6.
	<code>aws ec2 create-security-group --group-name SSHAccess --description "Security group for SSH access" --vpc-id &lt;vpc-id&gt;</code>
4	Lancer une instance <instance-id> dans le sous-réseau public, à l'aide du groupe de sécurité et de la paire de clés créés. Choisir une AMI Amazon Linux par exemple <ami-id>.
	<code>aws ec2 run-instances --image-id &lt;ami-id&gt; --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids &lt;securitygroup-id&gt; --subnet-id &lt;subnet-id&gt;</code>
5	Décrire l'instance et confirmer son état. Noter son adresse IPv6 publique <ipv6-address>
	<code>aws ec2 describe-instances --instance-id &lt;instance-id&gt;</code>
6	S'y connecter en ssh.
	<code>ssh -i "MyKeyPair.pem" ec2-user@&lt;ipv6-address&gt;</code>