
Compte-rendu de projet



Octobre 2019 – Août 2020



Membre du groupe : Buddy-Lee CELDRAN-SCHWARZ

Tuteur de projet : Sébastien Druon

Établissement / Formation : IUT de Béziers / Licence professionnelle Internet des
objets

Sommaire

Sommaire

Sommaire.....	3
1 Introduction.....	4
La réception du signal.....	6
2 Comment réceptionner le signal ?.....	7
2.1 Le signal ADS-B.....	7
2.2 La réception du signal.....	8
3 Idées du matériel à utiliser pour la réception.....	9
4 Le traitement des données.....	9
5 Premières idées de conception du projet.....	9
6 Transmission des données vers le serveur.....	10
7 Répartition des tâches.....	11
Premiers tests.....	12
8 Premier test (de façon chronologique).....	13
Premier test sur dump1090 :.....	13
Test transfert des infos :.....	16
Partie Serveur Prometheus + Grafana :.....	17
Redirection de ports :.....	24
Planefinder (Carte 3D) :.....	24
Développement de scripts et d'applications.....	28
1.1 La base de donnée InfluxDB avec collecte des données.....	29
Script Python :.....	32
Création d'un application Android permettant la consultation de la base de données.....	41
Conclusion de la partie « Développement de scripts et d'applications ».....	49
Nouvelle version.....	49
Conclusion du projet.....	54
Sources :.....	55

1 Introduction

Le but de ce projet est de réceptionner le signal ADS-B/ « AIS » des avions/ « bateaux », à l'aide de modules qui permettront la réception de signaux via des récepteurs RTL-SDR utilisés notamment pour la réception des chaînes de la tnt. Les informations seront envoyées vers un serveur local ou vers l'extérieur, cela nous permettra d'avoir la position en temps réel des avions sur une carte (2D voir 3D voir RV si possible).

Ce rapport se déroulera de la façon suivante :

Premièrement, nous essaierons de comprendre le terme ADS-B, la composition d'un paquet et comment le réceptionner.

Deuxièmement, nous essaierons de mettre en place un premier schéma pour le projet, ainsi qu'une première réflexion sur la façon de comment, les informations seront transmises.

Troisièmement, nous ferons des premiers tests avec la configuration de la clé RTL-SDR et l'utilisation des logiciels comme Dump1090, PlaneFinder, afin de comprendre comment fonctionne ces logiciels.

Quatrièmement, nous essaierons de créer un script permettant la sauvegarde et l'envoi des données vers une base de données, et nous essaierons de créer une application afin d'effectuer des requêtes et d'avoir un retour.

À Noter : il y a eu des difficultés à récupérer les messages des avions à cause de la crise sanitaire du Covid-19. Tous les avions devaient rester au sol. Néanmoins certains avions ont pu décoller (avions de marchandises) ce qui a permis de continuer le projet.

La réception du signal



2 Comment réceptionner le signal ?

2.1 Le signal ADS-B

Automatic Dependent Surveillance-Broadcast (ADS-B) est un protocole de communication utilisant la capacité d'un signal spontané présent dans la couche de transport en Mode S.

- Automatic : il ne nécessite aucune aide.
- Dependent : il dépend de l'altimètre, du GPS et d'autres instruments de l'avion pour obtenir des informations.
- Surveillance : elle fournit des informations actualisées sur l'avion émetteur.
- Broadcast : elle est unidirectionnelle, elle est diffusée vers tous les récepteurs à portée.

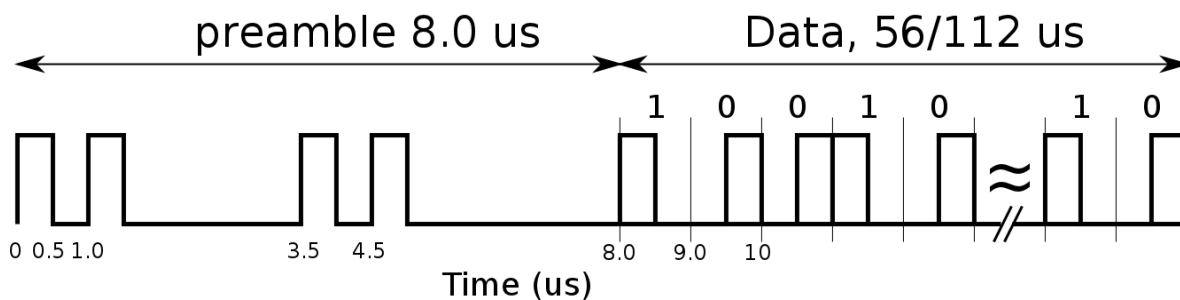
L'ADS-B est donc tout d'abord un moyen de surveillance, c'est-à-dire un moyen pour le contrôle aérien de connaître la position des avions. Les avions peuvent communiquer leurs informations aux tours de contrôle en utilisant deux fréquences en fonction de leurs classes ainsi que l'altitude des avions :

- 978 Mhz UAT = fréquence propre au USA
- 1090 Mhz 1090ES = fréquence utilisé par tous les avions

Entre 0 et 10.000 pieds, les deux fréquences peuvent être utilisés, entre 10.000 pieds et 18.000 pieds, les deux aussi peuvent être utilisés. Mais à partir de 18.000 pieds la fréquence de 1090 Mhz est obligatoire.

La composition d'un paquet se déroule de la façon suivante :

ADS-B mode S Packet



Le paquet est partagé en deux grandes parties, la première constitue la partie qui va initialiser la communication entre les avions et les radars.

Une fois la connexion initialisée, on envoie les données, les données sont envoyées de façon rapide, pour éviter toute collisions entre les avions.

2.2 La réception du signal

Pour réceptionner les signaux émis par les avions, il y a plusieurs possibilités :

La première possibilité serait d'avoir accès à un dispositif prévu à cet effet que l'on appelle transpondeur ou radar, c'est à dire qu'il a été conçu pour réceptionner les informations sur une fréquence pré-déterminée. Les transpondeurs les plus utilisés sont ceux qui utilisent le mode S (S pour selectif) , le radar détermine la position de l'avion et reçoit son altitude ainsi que son immatriculation, un code hexadécimal qui donne des infos sur le type d'avion, sa plage de vitesse, de plus ce mode est destiné à éviter les interrogations multiples lorsqu'il y a plusieurs radars dans la même zone afin d'éviter les collisions, ce qui peut être utile lorsque l'on souhaite envoyer les informations de plusieurs radar vers un serveur.

La deuxième est celle qui nous intéresse le plus, est d'utiliser un récepteur TNT USB portable. Ce dispositif a la capacité d'avoir une plage de fréquence assez large, dont les fréquences utilisés par les avions sont comprises dedans. De plus, on pourra le coupler à un Raspberry Pi afin de traiter les bonnes informations et ainsi de les afficher sur une carte.

3 Idées du matériel à utiliser pour la réception

Au niveau du matériel celui-ci est assez simple :

- Raspberry Pi : Envoie de données (Wi-Fi ou via Ethernet ou modem 4G).
- Carte Micro-SD pour le Raspberry Pi.
- Module rtl-sdr : USB sur le Raspberry Pi.
- Boîtier d'alimentation Raspberry Pi.
- Boîtier étanche si l'on souhaite placer le Raspberry Pi dehors si la station est fixe.

4 Le traitement des données

Une fois les informations reçus, il nous faudra les trier pour récupérer ce qu'il nous intéresse ici la position de l'avion (longitude et latitude), le numéro de vol, l'altitude, sa vitesse,...

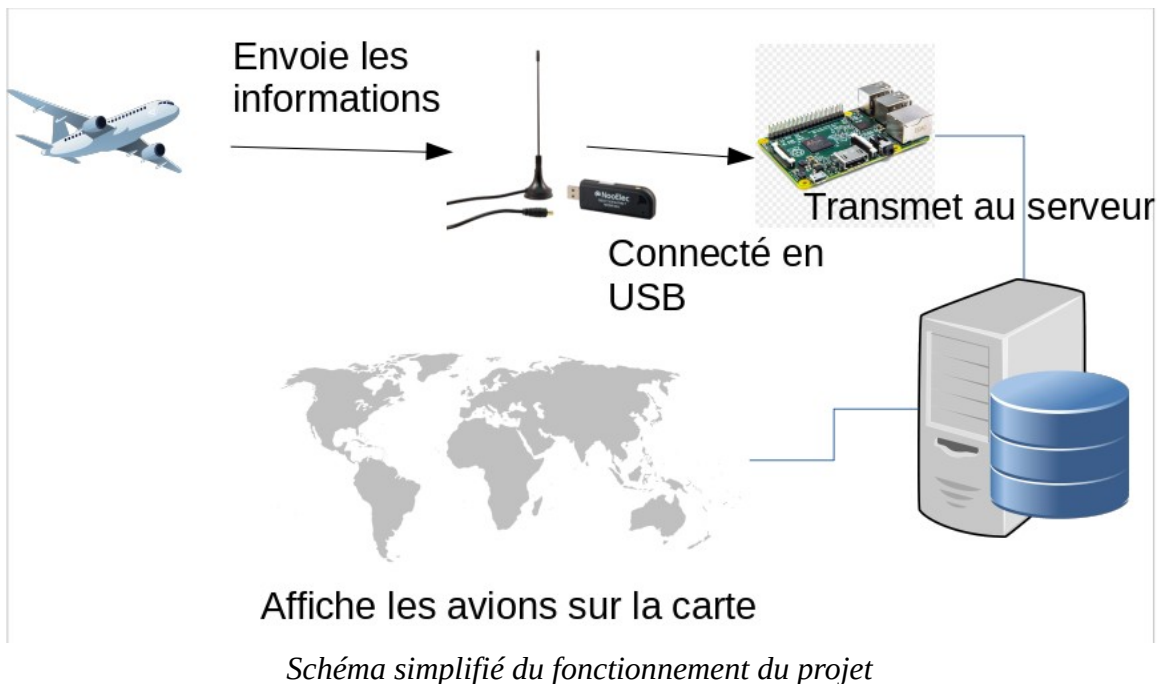
Certains logiciels permettent de trier les données réceptionnées comme dump1090, qui réceptionne les données, et permet d'afficher une carte sur le navigateur web de l'appareil qui l'utilise afin d'avoir la position des avions sur une carte. De plus le but serait de pouvoir envoyer ces informations sur un serveur afin d'avoir une vue globale de la circulation aérienne.

Pour pousser la chose GNURadio permet la lecture des données fournies via ADS-B et des cartes comme Openflight permettront d'afficher ces informations.

5 Premières idées de conception du projet

Le but de ce projet, c'est d'être capable de réceptionner les données, de les traiter mais aussi de les envoyer sur un serveur afin qu'il puisse lui aussi afficher une carte complète, car on utilisera pas qu'un seul dispositif mais plusieurs. Ça aura pour but d'élargir le champ de réception et ainsi avoir une meilleure triangulation pour la position des avions, mais aussi un champ d'action plus grand.

Le schéma suivant permet de se donner une idée du projet :



6 Transmission des données vers le serveur

Pour transférer les informations vers le serveur, on pourra le faire de la façon suivante :

- Utiliser le logiciel dump1090 ou dump1090-fa afin de récupérer les informations celles-ci sont convertis en json.
- On envoi les données dans un premier temps avec netcat.
- Le serveur lui aussi possède une version de dump1090 en mode réception, ainsi que divers outils permettant un affichage clair et détaillé des informations reçu.

7 Répartition des tâches

La répartition des tâches se fera de la façon suivante :

WordPackage	Tâches
WP1	Recherche sur le sujet
WP2	Recherche solution(s) et début de recherche matériels
WP3	Recherche matériels
WP4	Premiers tests
WP5	Création d'un serveur de réception et envoi des données depuis l'extérieur
WP6	Création de scripts/programmes afin d'éviter l'utilisation de programme existant
WP7	Création d'une application Android pour consulter la base de données

Premiers tests

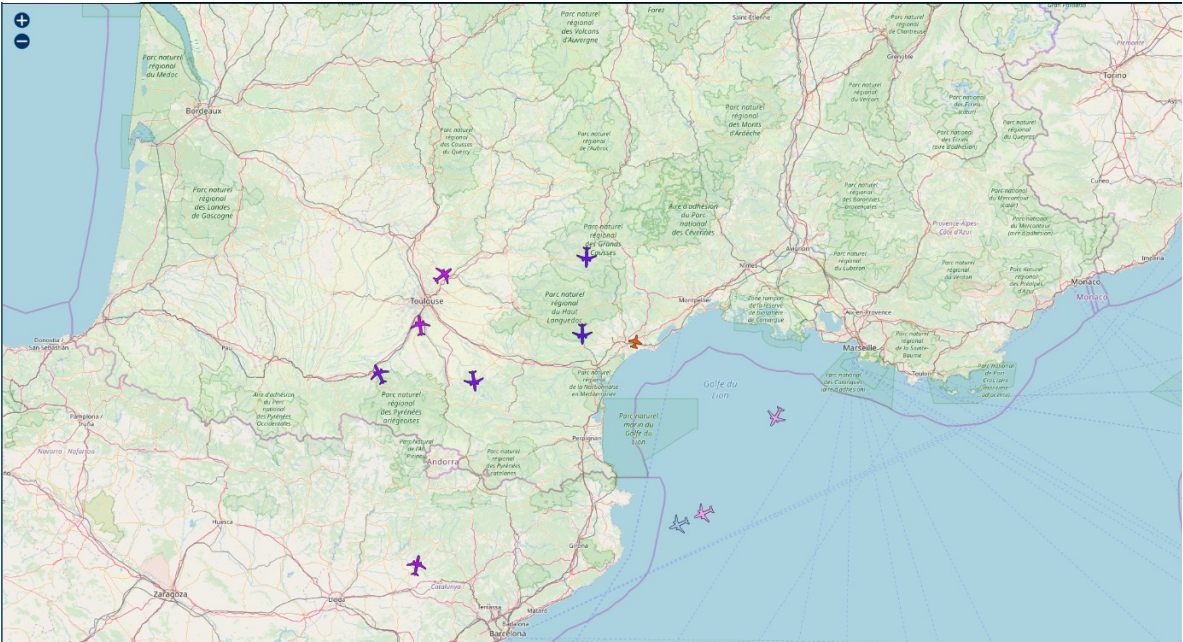


Illustration 1: Carte réception des avions

8 Premier test (de façon chronologique)

Premier test sur dump1090 :

Configuration :

Sur le PC après avoir brancher le dongle usb on utilise la commande dmesg afin de voir les messages venant des périphériques.

Le PC a installé le pilote permettant de regarder la télévision avec le dongle

DVB-T (TNT). Pour qu'il ne vienne pas perturber la réception des signaux

ADS-B, nous allons le placer en liste noire (blacklist) :

On créer un fichier rtl-sdr.conf dans le répertoire /etc/modprobe.d :

```
- cd/etc/modprobe.d  
- sudo nano rtl-sdr.conf
```

Ensuite dans ce fichier on ajoute la ligne suivante :

```
- blacklist dvb-usb-rtl28xxu
```

Cela a pour but de blacklister le driver qui a été installé. On vérifie si le pilote a été chargé avec lsmod si c'est le cas on lance la commande :

```
sudo modprobe -r dvb-usb-rtl28xxu
```

Afin que le pilote soit stoppé.

Ensuite pour fonctionner on aura besoin des pilotes rtl-sdr, pour cela il faut au

préalable installer certain paquet :

```
- sudo apt-get install git cmake build-essential libusb-1.0-0-dev pkg-config
```

Il nous reste plus qu'à récupérer le programme à l'aide de git et de le compiler :

```
- mkdir git  
- cd git
```

```
- git clone git://git.osmocom.org/rtl-sdr.git
- cd rtl-sdr
- mkdir build
- cd build
- cmake .. -DINSTALL-UDEV-RULES=ON
- sudo make install
- sudo ldconfig
- sudo cp ../rtl-sdr.rules /etc/udev/rules.d/
```

Ensuite il nous reste à installer le logiciel dump1090, il existe plusieurs versions, la plus complète est la version dump1090-fa qui comprend la possibilité d'écrire un fichier en json.

Pour ce faire on effectue les commandes suivantes :

```
- mkdir git2
- cd git2
- git clone https://github.com/flightaware/dump1090.git
- cd dump1090/
- cd public-html
- mkdir data
- cd ..
- make
```

On lance le programme avec la commande suivante :

```
- .sudo ./dump1090 --interactive --net --net-ro-port 30002 --write-json public-html/data
```

Cependant il nous faut une carte pour voir les avions sinon on aura qu'un retour en CLI.

Pour ce faire on ouvre un nouveau terminal, on va dans le dossier public_html et on lance la commande :

```
- python -m SimpleHTTPServer 8090
```

Cela permet de lancer un serveur http sur le port 8090.

On peut avoir la carte en tapant dans un navigateur web : l'adresse-machine:8090

Affichage de la carte :

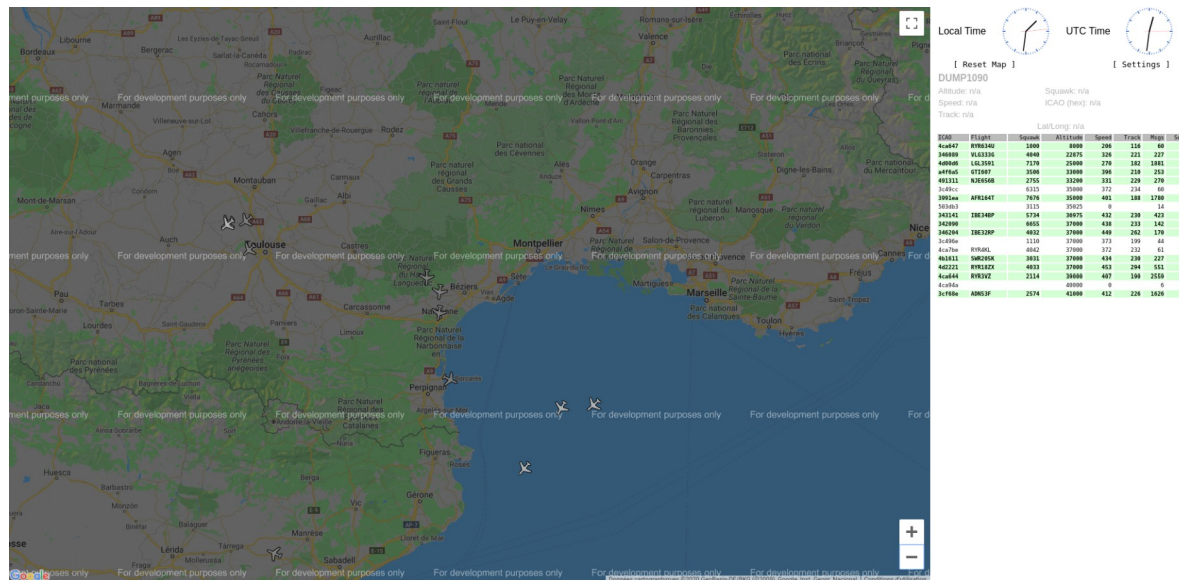


Illustration 2: Carte sous dump1090

En ligne de commande :

[illegible]

Illustration 3: Retour sur le terminal

Sur la partie de gauche, nous avons l’affichage des avions sous forme de tableau, de gauche à droite nous avons :

- La valeur hexadécimal du message
- Le mode de transmission utilisé
- Le squawk qui correspond au numéro d’identification de l’avion
- Le numéro du vol
- L’altitude
- La vitesse
- Le HDG (Mode suivi de cap) correspond au maintient du cap de l’avion.
- La latitude
- La longitude
- La puissance du signal reçu
- Le nombre de messages reçu

Test transfert des infos :

Sur la machine qui possède le dongle faire :

```
sudo ./dump1090 --interactive --net --net-ro-port 30002  
  
ou  
  
sudo ./dump1090 --interactive --net --net-ro-port 30002 --write-json public_html/data
```

Pour tester on peut lancer un netcat :

```
nc localhost 30002
```

Sur la machine qui va recevoir les informations (en local):

```
nc 10.215.0.182 30002 | nc 10.215.0.184 30001
```

Et :

```
sudo ./dump1090 --interactive --net-only --net-ri-port 30001 --write-json public_html/data
```


Hex	Mode	Sqk	Flight	Alt	Spd	Hdg	Lat	Long	Sig	Msgs	Ti
34324E	S			38950					255	3	3
4CA647	S	1000		8000	268	357			255	12	6
4CA644	S								255	3	12
4B169A	S	3043		34950	357	212	43.114	4.512	255	49	0
4CA4F2	S	4047		37000	373	233	42.675	4.562	255	35	1
3451D9	S								255	4	0
4D0806	S	7170	LGL3591	25000	249	182	42.833	2.869	255	154	0
3991EA	S	7676		25700	395	194	42.473	2.835	255	92	1

On a bien une réception des informations de l'instance dump1090 présent sur le pc vers une machine virtuelle.

Si ajax call failed :

```
sudo wget -O /etc/udev/rules.d/rtl-sdr.rules "https://raw.githubusercontent.com/osmocom/rtl-sdr/master/rtl-sdr.rules"
```

Partie Serveur Prometheus + Grafana :

Pour regrouper les informations des différentes instances de dump1090, on peut mettre en place un serveur prometheus avec un dashboard grafana afin d'afficher de manière correct les informations reçu.

Prometheus étant un outils de monitoring et d'alerte, il sera utile pour trier les informations reçues.

Afin de transmettre les information à prometheus on récupère un programme qui se nomme dump1090exporter, ce programme a pour but d'extraire les fichiers aircraft.json et receiver.json de dump1090 et de les transférer vers une instance prometheus, le programme s'installe de la façon suivante :

```
pip3 install dump1090exporter
```

Pour prometheus, on récupère les fichiers sur le site prometheus.io :

```
# ls
projet_ido prometheus-2.15.2.linux-amd64.tar.gz snap
# mkdir prometheus
# mv prometheus-2.15.2.linux-amd64.tar.gz prometheus/
```

```
# cd prometheus/  
prometheus# ls  
prometheus-2.15.2.linux-amd64.tar.gz  
# tar xvf prometheus-2.15.2.linux-amd64.tar.gz
```

Une fois installé, il nous suffit de créer un fichier permettant de créer une tâche pour prometheus que l'on nommera dump1090.yml, ce fichier se compose de la façon suivante :

```
scrape_configs:  
  - job_name: 'dump1090'  
    scrape_interval: 10s  
    scrape_timeout: 5s  
    static_configs:  
      - targets: ['localhost:9105']
```

On a le nom de la tâche à effectuer, les intervalles d'actualisation, ainsi que l'IP de la source où prometheus doit récupérer les infos.

On lance le serveur prometheus :

```
root@debian:~/prometheus/prometheus-2.15.2.linux-amd64# ./prometheus --  
config.file=dump1090.yml  
level=info ts=2020-01-27T12:30:09.355Z caller=main.go:294 msg="no time or size  
retention was set so using the default time retention" duration=15d  
[...]  
level=info ts=2020-01-27T12:30:09.367Z caller=web.go:506 component=web msg="Start  
listening for connections" address=0.0.0.0:9090
```

On lance dump1090exporter :

```
dump1090exporter --url=http://localhost:8090 --port=9105 --latitude=43.346813 --  
longitude=3.222171 --debug
```

Pour la longitude et la latitude, on se base sur L'IUT de Béziers.

Pour visualiser les données on pourra utiliser grafana.

Installation grafana :

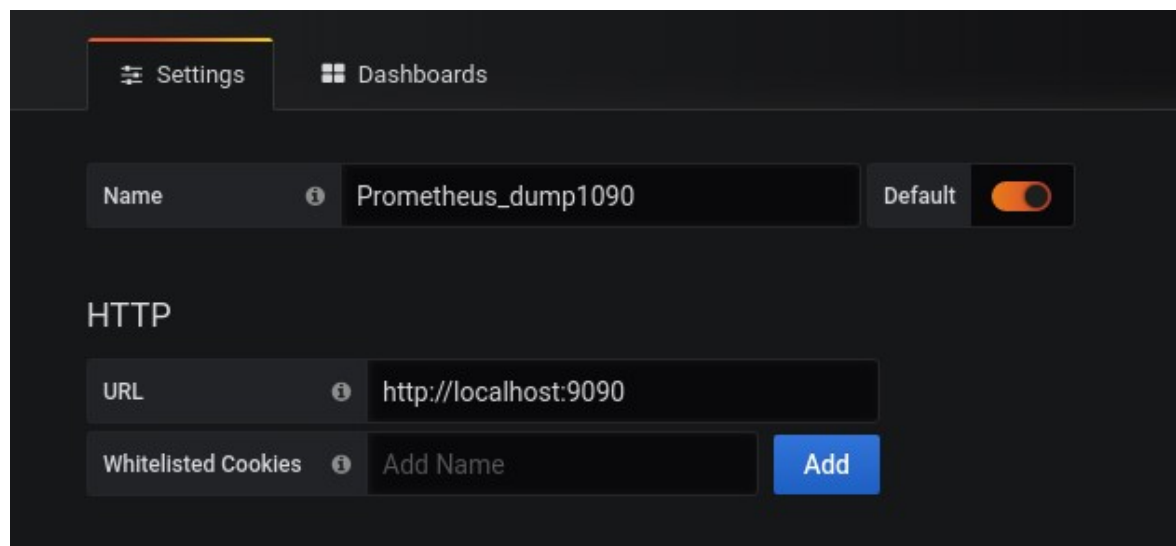
```
sudo apt-get install -y apt-transport-https
sudo apt-get install -y software-properties-common wget
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"
sudo apt-get update
sudo apt-get install grafana
```

On lance le serveur :

```
root@debian:~/prometheus/prometheus-2.15.2.linux-amd64# service grafana-server start
root@debian:~/prometheus/prometheus-2.15.2.linux-amd64# service grafana-server status
• grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-01-27 12:10:32 UTC; 3s ago
```

On se rend sur le site @adresse_serveur:3000 et on modifie le mot de passe ici : Rhodia42

On ajoute la base de donnée prometheus :



The screenshot shows the Grafana Settings page for a new Prometheus database. At the top, there are two tabs: 'Settings' (active) and 'Dashboards'. Below the tabs, there is a form with the following fields:

- Name:** Prometheus_dump1090
- Default:** A toggle switch that is currently turned on.

Below these fields, there is a section titled 'HTTP' with the following fields:

- URL:** http://localhost:9090
- Whitelisted Cookies:** A field with the placeholder text 'Add Name' and a blue 'Add' button next to it.

Illustration 4: Ajout BDD Prometheus

On ajoute un dashboard compatible avec dump1090exporter.

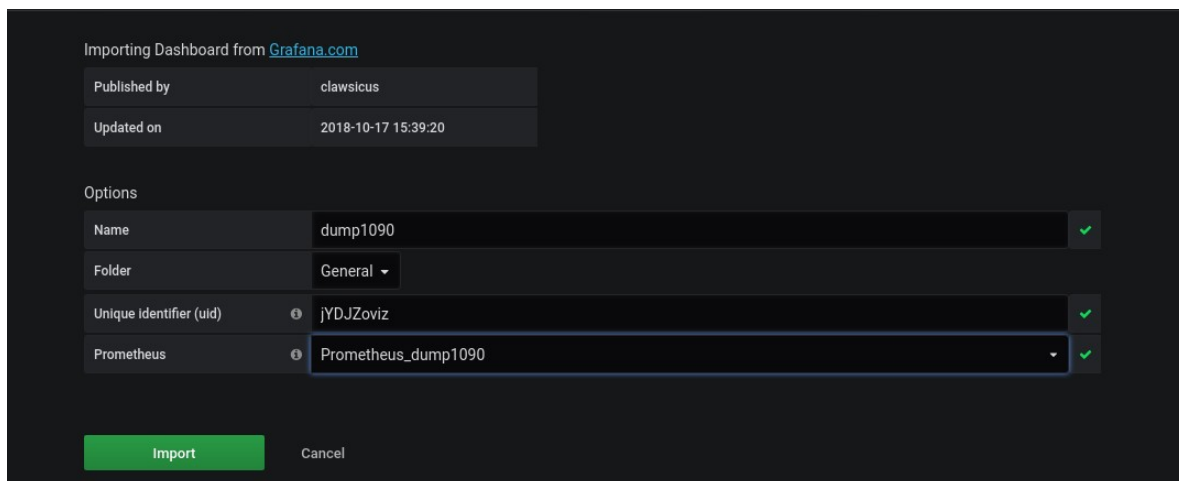


Illustration 5: Ajout du dashboard

On obtient le résultat suivant :

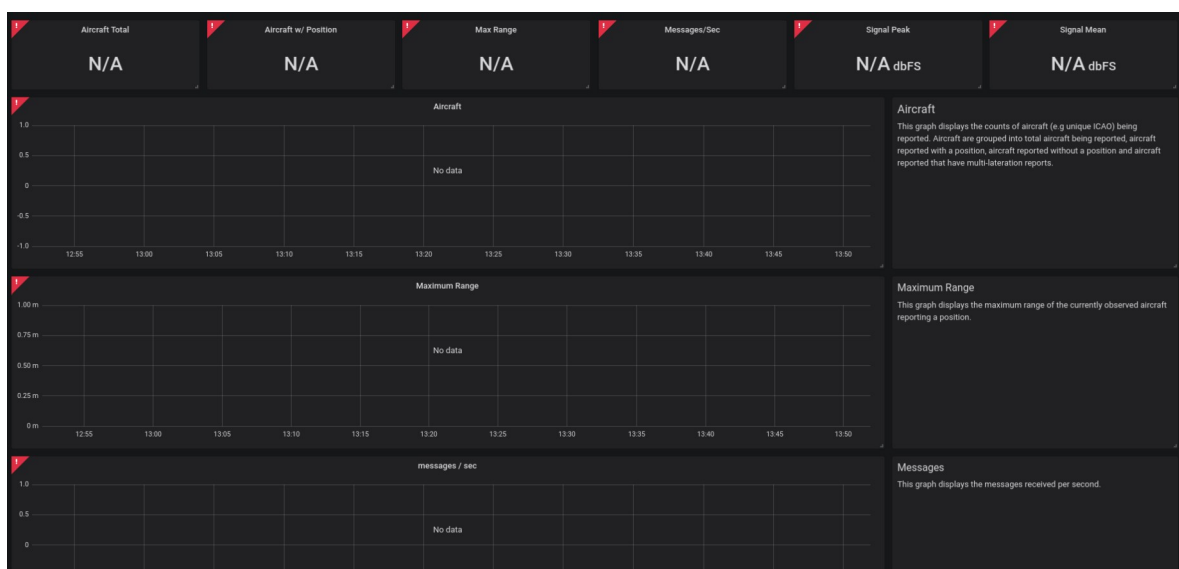


Illustration 6: Dashboard

Une fois que tout est initialisé, il nous suffit de lancer les instances de dump1090 sur la machine qui possède la clé et le serveur et de mettre en place netcat :

Sur la machine qui possède la clé :

```
sudo ./dump1090 --interactive --net --net-ro-port 30002 --write-json public_html/data  
python -m SimpleHTTPServer 8090
```

Ou

```
sudo ./dump1090 --interactive --net --net-ro-port 32772 --write-json public_html/data
```

On peut définir un port de sortie, le port par défaut est 30002 mais rien n'empêche de le changer.

Sur le serveur :

```
sudo ./dump1090 --interactive --net-only --net-ri-port 30001 --write-json public_html/data
```

```
root@debian:~/projet_ido/p2/dump1090/public_html# python -m SimpleHTTPServer 8090
```

```
root@debian:~/prometheus/prometheus-2.15.2.linux-amd64# ./prometheus --config.file=dump1090.yml
```

```
root@debian:~# dump1090exporter --url=http://localhost:8090 --port=9105 --latitude=43.346813 --longitude=3.222171 --debug
```

```
root@debian:~# nc 192.168.43.179 30002 | nc localhost 30001
```

Sur grafana on obtient le résultat suivant :



On a le nombre d'avions, suivi du nombre d'avions auquel on a reçu leurs positions, le Max range correspond à la distance maximale entre le point définit (ici l'IUT) et l'avion le plus loin, on reçoit alors les informations, la distance varie par rapport à la longitude et latitude définies dans le fichier dump1090.yml de prometheus.

On a aussi le nombre de messages reçu par seconde.

Afin de faciliter la mise en place du serveur on peut créer un script :

Sur la machine avec la clé :

```
#!/bin/bash

cd /home/buddy/projet_ido/git2/dump1090
sudo ./dump1090 --interactive --net --net-ro-port 30002 --write-json public_html/data
cd public_html
python -m SimpleHTTPServer 8090
```

Sur le serveur :

```
root@debian:~# cat serveur.sh
#!/bin/bash
# Lance Serveur Grafana
service grafana-server start
# Instance dump1090
cd /root/projet_ido/p2/dump1090
sudo ./dump1090 --interactive --net-only --net-ri-port 30001 --net-bo-port 30005 --write-json
public_html/data
cd /root/projet_ido/p2/dump1090/public_html
python -m SimpleHTTPServer 8090
# Dump1090 exporter
cd
dump1090exporter --url=http://localhost:8090 --port=9105 --latitude=43.346813 --
longitude=3.222171 --debug &
# Prometheus
cd /root/prometheus/prometheus-2.15.2.linux-amd64
./prometheus --config.file=dump1090.yml &
```

Pour la partie transfert depuis l'extérieur, on peut utiliser netcat, c'est un outil permettant d'ouvrir des connexions réseau TCP et UDP.

Pour netcat, il suffit de rentrer la commande `nc localhost <port> | nc @serveur <port>` dans le script qui permet d'envoyer les informations vers le serveur.

Pour ce qui est de la partie serveur afin de récupérer les informations provenant des différents modules, il faut configurer le serveur de la façon suivante :

- Tout d'abord si le serveur est à l'extérieur nous devons faire de l'ip forwarding c'est à dire que l'on va se servir de l'adresse IP Publique avec un port spécifique, qui redirigera vers la machine visé auquel on a configuré le port et une adresse IP fixe. Ici on souhaite que le port 30001 (port de réceptions des informations avec dump1090), dans ce cas on configure sur le routeur un port quelconque qui redirigera vers la machine et le port 30001.

Cependant avec netcat, l'outil n'accepte qu'une connexion par port à la fois, pour ce faire on peut configurer deux ports voir plus qui redirigerons vers le port 30001, comme ça chaque instance netcat des différents modules pourra envoyer ses informations sans avoir de conflits.

- Sur les machines il nous suffit juste de lancer les instances de dump1090 et de lancer la commande netcat avec la bonne adresse et le port attribué à chaque module :

Machine 1 :

```
nc localhost 30002 | nc -q 1 @IP_Publique 32772
```

Machine 2 :

```
nc localhost 30002 | nc -q 1 @IP_Publique 32774
```

Les machines vont envoyer les données vers le serveurs avec des ports spécifiques qui redirigeront ensuite vers le port 30001 du serveur.

Les données :

Les données envoyées par le port 30002 vers le port 30001 sont sous cette forme :

```
*8F4401E458B5001E98F482A1E801;
```

Le début de chaque information commence par « * » et finis par « ; ».

Une fois reçu par le serveur, les données sont retranscrite et affichées pour quelles soit lisible par l'utilisateur.

Redirection de ports :

Afin d'avoir accès au serveur à distance, j'ai dû faire une ouverture et une redirection des ports

Port 32822	SSH
Port 32770	Port 8090 (Carte dump1090)
Port 32771	3000 (Grafana)
Port 32772	Port 30001 dump1090 entrée
Port 32774	Port 30001 dump1090 entrée
Port 32773	Port 30002 dump1090 sortie
Port 1883	Port 1883 mqtt

Planefinder (Carte 3D) :

Le programme planefinder est un programme de suivi d'avions semblable à dump1090, dans notre cas, il nous servira à afficher une carte en 3D, car il s'agit de l'un des seuls programmes permettant d'obtenir une carte 3D du monde de manière libre et gratuite.

Pour pouvoir utiliser cette carte, il nous faut télécharger le client celui-ci est disponible à cette adresse :

<https://planefinder.net/sharing/client>

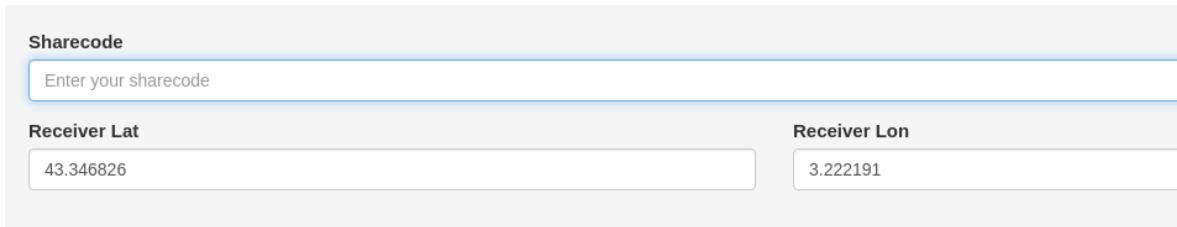
Puis on télécharge le paquet pour machine i386, car on utilise un pc et non un raspberry pi, une fois téléchargé, il nous faut l'installer :

```
sudo dpkg -i pfclient_4.1.1_i386.deb
```

Une fois installé, il est conseillé de redémarrer la machine afin que certains paramètres soient pris en compte.

Par la suite il faut lancer le client avec la commande pfclient, au premier démarrage il demandera une configuration. Pour ce faire il faudra utiliser un navigateur web et entrer cette adresse : @Ipdelamachine:30053.

Une fois sur le site on commence la configuration :

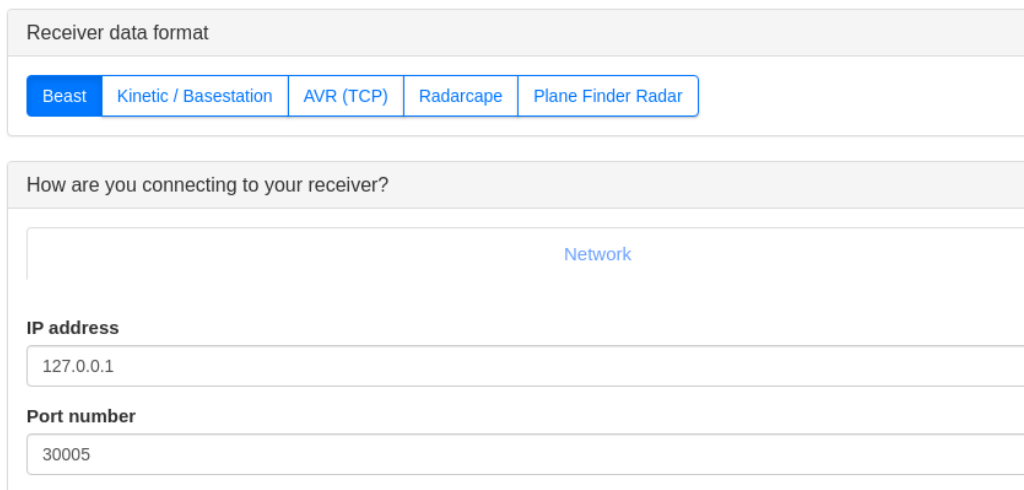


The screenshot shows a web form for PlaneFinder configuration. At the top is a 'Sharecode' section with a text input field labeled 'Enter your sharecode'. Below this are two input fields: 'Receiver Lat' with the value '43.346826' and 'Receiver Lon' with the value '3.222191'.

Illustration 7: Configuration PlaneFinder

On renseigne les coordonnées GPS qui servira de repère (ici l'IUT), le sharecode est un code personnel qui ne peut être partagé.

À l'étape suivante on configure la machine qui va lui envoyer les informations, dans notre cas comme le serveur récupère les informations des différentes instances de dump1090, c'est lui qui va transférer les informations au client, l'adresse sera alors 127.0.0.1 :



The screenshot shows the 'Receiver data format' section with five buttons: 'Beast', 'Kinetic / Basestation', 'AVR (TCP)', 'Radarcapex', and 'Plane Finder Radar'. Below this is the 'How are you connecting to your receiver?' section, which has a 'Network' radio button selected. Under 'Network', there are two input fields: 'IP address' with the value '127.0.0.1' and 'Port number' with the value '30005'.

Illustration 8: Réception des données PlaneFinder

Pour le format des données on le laisse en brute, car dump1090 a la possibilité d'envoyer les données sous ce format. En ce qui concerne le port il s'agit du port par défaut pour ce type de format de donnée avec dump1090.

Une fois le tout configuré, il nous reste à modifier les arguments pour dump1090 afin qu'il prenne en compte le port 30005.

```
sudo ./dump1090 --interactive --net-only --net-ri-port 30001 --net-bo-port 30005 --lat 43.346915 --lon 3.222259 --write-json public_html/data
```

On rajoute l'argument `--net-bo-port` qui correspond à la sortie brute des données sur le port 30005.

Plane finder va alors écouter sur ce port et récupérer les informations.

Une fois les programmes lancés, on se connecte sur l'adresse du serveur:30053, et on affiche la carte en 3D :

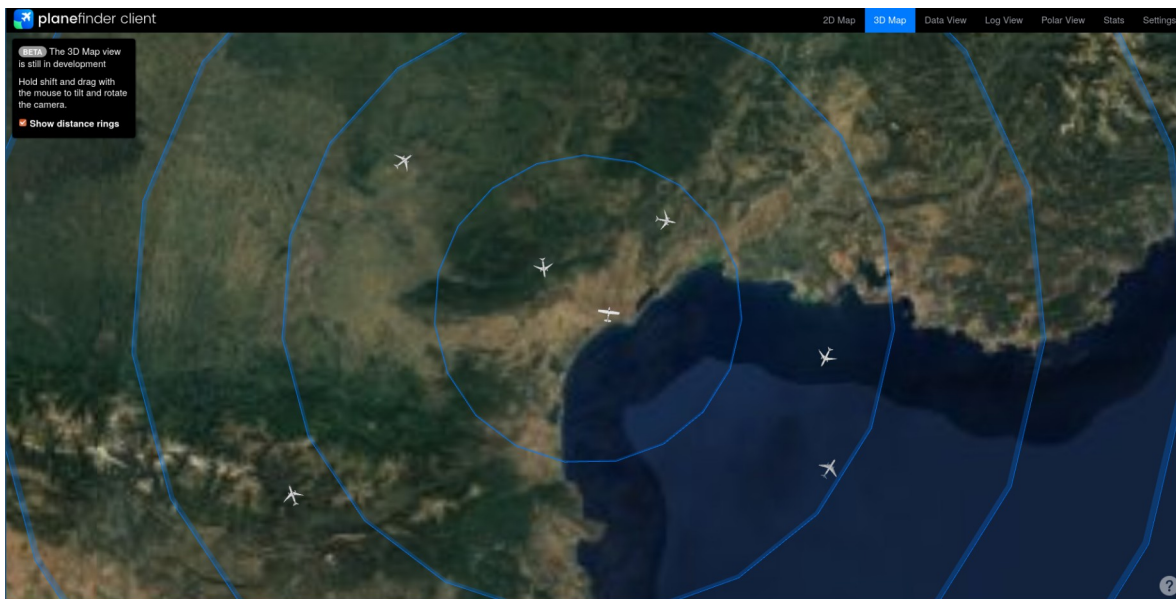


Illustration 9: Affichage 3D de la carte

Les avions présentés ici sont les avions qui ont partagés leurs positions.

GNU Radio :

Une fois que l'on a étudié le fonctionnement de dump1090 et ses résultats on peut essayer de recevoir les informations d'une autre façon, pour cela on utilisera GNU Radio, C'est une suite de logiciels dédiée à l'implémentation de radios logicielles et de systèmes de traitement du signal.

Installation module rtl-sdr gnuradio :

<https://doc.ubuntu-fr.org/gqrx-sdr>

```
sudo apt install gqrx-sdr
```

<https://github.com/mhostetter/gr-adsb>

Installation du module ads-b sur GNU-Radio :

```
buddy@buddy-G5-5590:~/projet_ido$ git clone https://github.com/mhostetter/gr-adsb.git
Clonage dans 'gr-adsb'...
remote: Enumerating objects: 1132, done.
remote: Total 1132 (delta 0), reused 0 (delta 0), pack-reused 1132
Réception d'objets: 100% (1132/1132), 1.40 MiB | 1.74 MiB/s, fait.
Résolution des deltas: 100% (718/718), fait.
buddy@buddy-G5-5590:~/projet_ido$ cd gr-adsb/
```

```

buddy@buddy-G5-5590:~/projet_ido/gr-adsb$ mkdir build
buddy@buddy-G5-5590:~/projet_ido/gr-adsb$ cd build/
buddy@buddy-G5-5590:~/projet_ido/gr-adsb/build$ cmake ../
buddy@buddy-G5-5590:~/projet_ido/gr-adsb/build$ make
buddy@buddy-G5-5590:~/projet_ido/gr-adsb/build$ sudo make install
buddy@buddy-G5-5590:~/projet_ido/gr-adsb/build$ sudo ldconfig

```

Le schéma suivant permet la réception du signal brute, puis avec les modules ads-b nous pouvons le démoduler et d'afficher un retour.

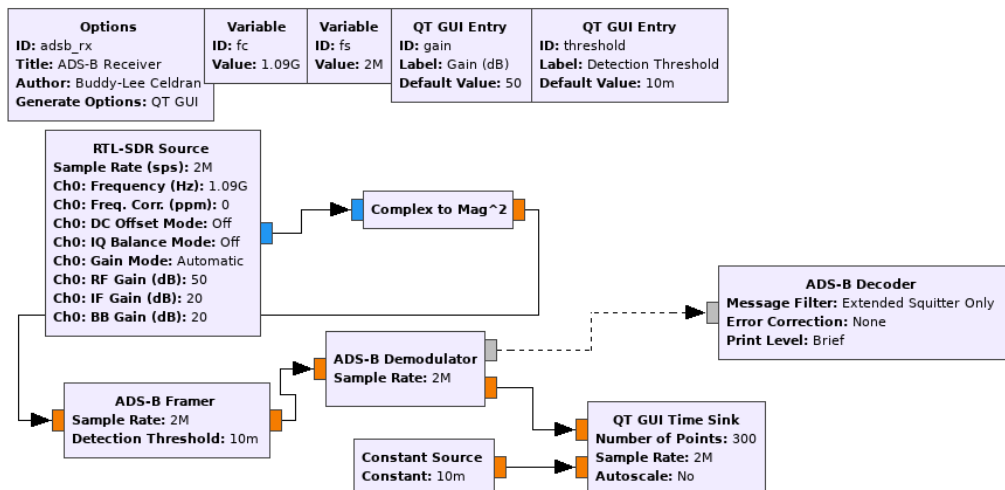


Illustration 10: Schéma GNU-Radio

ICAO	Callsign	Alt (ft) (ft/m)	Climb (kt)	Speed (deg)	Hdng	Latitude	Longitude	Msgs	Age
0a0090		35000	-64	533	-90		6 0		
4cabbf	RYR63QN	23525					3 8		
44a832		39000					1 49		
406a3e		38000	64	359	89	42.3678434	0.8403902	6	2
4249b2		64	420	-143			1 2		
3c56f6		38025					1 98		
020132		33775			44.1013184	2.4970902	2 71		
440452		36975					2 21		
4cab99		20250	-2240	473	-14		3 38		
346111		36000	0	360	97		7 17		
3b77f3		19975	0	223	-78	43.4375610	3.8537740	5	10
3c6664		0	425	35			1 71		
44035c		34000	64	425	36		4 5		
394c19		29625	192	366	98		2 64		
0a008c		35000	64	544	-90		4 30		
40097d		30850	960	353	108		4 34		
502d16		34025	0	419	37	43.8934999	1.7885481	11	42
4d2133		40975	-64	424	-141		3 11		

Illustration 11: Retour GNU-Radio

Développement de scripts et d'applications

```
177         ),
178         default="v",
179     )
180     global_scale_setting = FloatProperty(
181         name="Scale",
182         min=0.01, max=1000.0,
183         default=1.0,
184     )
185
186     def execute(self, context):
187         # get the folder
188         folder_path = (os.path.dirname(self.filepath))
189
190         # get objects selected in the viewport
191         viewport_selection = bpy.context.selected_objects
192
193         # get export objects
194         obj_export_list = viewport_selection
195         if self.use_selection_setting == False:
196             obj_export_list = [i for i in bpy.context.scene.objects]
197
198         # deselect all objects
199         bpy.ops.object.select_all(action='DESELECT')
200
201         for item in obj_export_list:
202             item.select = True
203             if item.type == 'MESH':
204                 file_path = os.path.join(folder_path, "{}.obj".format(item.name))
205                 bpy.ops.export_scene.obj(
206                     filepath=file_path, use_selection=True,
207                     axis_forward=self.axis_forward_setting,
208                     axis_up=self.axis_up_setting,
209                     use_animation=self.use_animation_setting,
210                     use_mesh_modifiers=self.use_mesh_modifiers_setting,
211                     use_edges=self.use_edges_setting,
212                     use_smooth_groups=self.use_smooth_groups_setting,
213                     use_smooth_groups_bitflags=self.use_smooth_groups_bitflags_setting,
214                     use_normals=self.use_normals_setting,
215                     use_uvs=self.use_uvs_setting,
216                     use_materials=self.use_materials_setting,
```

Une fois les différents logiciels mis à disposition utilisés, on peut dans un premier temps essayer de développer une solution de collecte de données.

1.1 La base de donnée InfluxDB avec collecte des données

Pour sauvegarder les messages envoyés, on configure une base de donnée. Pour cela on utilisera InfluxDB comme base de donnée. Ce choix est fait car nous avons pu nous en servir a plusieurs reprise pendant les cours.

On installe le paquet influxDB :

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
echo "deb https://repos.influxdata.com/ubuntu stretch stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
apt update
apt install influxdb
E: Sub-process /usr/bin/dpkg returned an error code (1)
systemctl unmask influxdb.service
Removed /etc/systemd/system/influxdb.service.
service influxdb start
influx
```

Puis on va dans / etc/ influxDB / influxdb.conf

On cherche la connexion via HTTP :

```
[http]
enabled = true
bind-address = ":8086"
auth-enabled = true
write-tracing = false
log-enabled = true
pprof-enabled = true
debug-pprof-enabled = false
https-enabled = true
https-certificate = "/etc/ssl/influxdb.pem"
```

On active le support HTTP, avec le port 8086, avec authentification.

On vérifie si le service est lancé :

```
service influxd status
```

- influxdb.service - InfluxDB is an open-source, distributed, time series database

Loaded: loaded (/lib/systemd/system/influxdb.service; enabled; vendor preset: enabled)

Active: active (running) since Thu 2019-11-28 11:52:14 UTC; 2min 14s ago

On lance influx pour afficher le CLI afin de créer un user admin :

```
influx
Connected to http://localhost:8086 version 1.7.9
InfluxDB shell version: 1.7.9
> CREATE USER admin WITH PASSWORD 'influx' WITH ALL PRIVILEGES
> SHOW USERS
ERR: unable to parse authentication credentials
Warning: It is possible this error is due to not setting a database.
Please set a database with the command "use <database>".
```

L'utilisateur admin a été créé avec tous les privilèges, hors si on fait un show users on a une erreur pour corriger cela il suffit de quitter et de se reconnecter de nouveau :

```
influx -username admin -password influx
Connected to http://localhost:8086 version 1.7.9
InfluxDB shell version: 1.7.9
> SHOW USERS
user  admin
----  ----
admin true
>
```

L'utilisateur est présent et possède les droits.

On crée la base de donnée dump1090 :

```
CREATE DATABASE dump1090
```

Par la suite, on installera Telegraf sur le serveur, on utilise Telegraf afin que les données soit acheminées vers la base de données, car Telegraf est un collecteur c'est-à-dire qu'il va récupérer les données à intervalles réguliers afin de les envoyer par la suite vers InfluxDB. :

L'installation se passe de la manière suivante :

```
sudo apt install -y gnupg2 curl wget
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -echo "deb
https://repos.influxdata.com/debian buster stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
sudo apt update
sudo apt install telegraf
```

Une fois le telegraf installé, il faut par la suite, le démarrer :

```
service telegraf start
```

Ensuite il faut faire en sorte que Telegraf puisse communiquer avec InfluxDB, pour se faire on modifie le fichier de conf de Telegraf, qui est présent dans /etc / telegraf :

```
# Configuration for sending metrics to InfluxDB
[[outputs.influxdb]]
  ## The full HTTP or UDP URL for your InfluxDB instance.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  urls = ["http://127.0.0.1:8086"]
  ## The target database for metrics; will be created as needed.
  ## For UDP url endpoint database needs to be configured on server side.
  database = "dump1090"
  ## If true, no CREATE DATABASE queries will be sent. Set to true when using
  ## Telegraf with a user without permissions to create databases or when the
  ## database already exists.
  skip_database_creation = false
  ## HTTP Basic Auth
  username = "admin"
  password = "influx"

  ## HTTP User-Agent
  user_agent = "telegraf"
```

On renseigne l'URL utilisé comme on utilise le même serveur l'adresse reste en 127.0.0.1 et on lui renseigne le port de influxDB ici le port 8086. On indique la base de données à utiliser ainsi que les identifiants et mot de passe pour y accéder. Dans l'avenir on pourra créer plusieurs utilisateurs.

Script Python :

Suite à la réunion de l'avancement des projets qui a eu lieu le 29 Juin 2020, et suite aux recommandations des professeurs.

Nous souhaitons par la suite, envoyer les informations vers le serveur de manière automatique, pour cela, nous souhaitons créer un script en python qui va récupérer les données de dump1090 sur le port 30003 qui correspond au port basestation et les acheminer ensuite vers la base de données, via le protocole HTTP. InfluxDB possède une API permettant de le faire.

Pour l'écriture du script nous nous sommes basé sur un modèle pour la partie réception du signal afin d'avoir une base qui fonctionne pour la suite du projet. Nous avons donc créé notre version modifiée afin d'avoir plus d'informations stockées sur la base de données. Nous avons supprimé les parties qui nous paraissaient « non nécessaires », car nous voulons garder un maximum d'informations et non les supprimer.

Pour pouvoir utiliser influxDB et Dump1090, il nous faut définir les imports :

```
import signal
import socket
import time
import calendar
import argparse
import re
import logging
import requests
```

Le module socket permet la lecture des informations de dump1090 sur le port 30003 et l'envoi des données vers l'adresse du serveur. Les modules time et calendar nous sont utiles pour l'horodatage des avions sur la base de données. Le module request nous sert avec influxDB pour écrire des requêtes sur la base de données.

La partie suivante permet la recherche des termes souhaiter sur dump1090 afin de pouvoir les exporter par la suite, pour ce faire on utilise RegEx présent dans « import re » :

```
class AdsbProcessor(object):

    REGEXP_MSG = r'^MSG,' \
        r'(?P<transmission>\d),' \
        r'(?P<session>\d+),' \
        r'(?P<aircraft>\d+),' \
        r'(?P<hexident>[0-9A-F]+),' \
        r'(?P<flight>\d+),' \
        r'(?P<gen_date>[0-9/]+),' \
        r'(?P<gen_time>[0-9:\.]+),' \
        r'(?P<log_date>[0-9/]+),' \
        r'(?P<log_time>[0-9:\.]+),' \
        r'(?P<callsign>[\w\s]*),' \
        r'(?P<altitude>\d*),' \
        r'(?P<speed>\d*),' \
        r'(?P<track>[\d\-*]*),' \
        r'(?P<latitude>[\d\-\.]*),' \
        r'(?P<longitude>[\d\-\.]*),' \
        r'(?P<verticalrate>[\d\-*]*),' \
        r'(?P<squawk>\d*),' \
        r'(?P<alert>[\d\-*]*),' \
        r'(?P<emergency>[\d\-*]*),' \
        r'(?P<spi>[\d\-*]*),' \
        r'(?P<onground>[\d\-*]*)$'

    NORMALIZE_MSG = {
        'transmission': (lambda v: int(v)),
        'session': (lambda v: int(v)),
        'aircraft': (lambda v: int(v)),
        'flight': (lambda v: int(v)),
        'callsign': (lambda v: v.strip()),
        'altitude': (lambda v: int(v)),
        'speed': (lambda v: int(v)),
        'track': (lambda v: int(v)),
        'latitude': (lambda v: float(v)),
        'longitude': (lambda v: float(v)),
        'verticalrate': (lambda v: int(v)),
        'alert': (lambda v: True if v == '-1' else False),
        'emergency': (lambda v: True if v == '-1' else False),
        'spi': (lambda v: True if v == '-1' else False),
        'onground': (lambda v: True if v == '-1' else False),
    }
    [...]
```

Dans la partie qui concerne influxDB, on peut aussi indiquer un utilisateur et son mot de passe par défaut (s'ils ont été créés) :

```
class InfluxDB(object):
    def __init__(self, url, database='dump1090', username="admin", password="influx"):
        self.url = url
        self.params = '/write?precision=s&db={}'.format(database)
    def write(self, measurement, data, timestamp=None):
        lines = []
        for d in data:
            fields = []
            for k, v in d['fields'].items():
                if v is None:
                    continue
                elif type(v) is bool:
                    fields.append('{}={}'.format(k, 't' if v else 'f'))
                elif type(v) is int:
                    fields.append('{}={}'.format(k, v))
                elif type(v) is float:
                    fields.append('{}={}'.format(k, v))
                elif type(v) is str:
                    fields.append('{}="{}".format(k, v))
                else:
                    log.warning("Le type {} n'est supporté par InfluxDB. {}={}".format(
                        type(v), k, v
                    ))

            lines.append('{measurement},{tags} {fields} {timestamp}'.format(
                measurement = measurement,
                tags = ','.join('{}={}'.format(k, v) for k, v in d['tags'].items()),
                fields = ','.join(x for x in fields),
                timestamp = d['timestamp'] if 'timestamp' in d else int(time.time()),
            ))
        [...]

```

On cherche à vérifier si on a des informations et dans quel format on les reçoit. Si ce sont des valeurs comme l'altitude et la longitude, on utilisera float, pour l'altitude, on utilisera int,...

Pour `lines.append` dès que nous avons les valeurs nécessaires, on construit la requête à envoyer. Le `measurement` est défini dans le main.

Pour Dump1090, on cherche à écouter sur le port 30003 du serveur qui accueille dump1090, le programme étant sur le même appareil l'adresse sera localhost.

```
class Dump1090(object):
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.s = None
        self.data = "

```

```

def connect(self):
    log.info('Connexion à dump1090 en TCP sur {}:{}'.format(self.host, self.port))
    self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connected = False

    while not connected:
        try:
            self.s.connect((self.host, self.port))
            log.info('Connexion OK, reception en cours')
        except:
            connected = False
            log.warning('Connexion impossible')
            time.sleep(1)
        else:
            connected = True

    self.s.setblocking(False)
    self.s.settimeout(1)

```

Dans le main, on lance les classes que nous avons vu précédemment :

```

ap = AdsbProcessor()

dump1090 = Dump1090(args.dump1090_server, int(args.dump1090_port))
dump1090.connect()

influx = InfluxDB(args.influx_url, database=args.influx_database)

```

De plus on peut aussi définir les informations que l'on souhaite recevoir :

```

'tags': {
    'hexident': hexident,
    'callsign': msg['callsign'],
    'squawk': msg['squawk'],
},
'fields': {
    'hexident': hexident,
    'callsign': msg['callsign'],
    'generated': timestamp,
    'altitude': msg.get('altitude'),
    'speed': msg.get('speed'),
    'track': msg.get('track'),
    'latitude': msg.get('latitude'),
    'longitude': msg.get('longitude'),
    'verticalrate': msg.get('verticalrate'),
    'alert': msg.get('alert'),
    'emergency': msg.get('emergency'),
    'spi': msg.get('spi'),
}

```

```

        'onground': msg.get('onground'),
        'count': msg.get('count', 0),
    }
})

```

Les « fields » définis ici sont les champs que le script va récupérer afin de les transférer vers la base de données.

La valeur avions est le measurement où les informations vont être stockées. On a aussi le nombre d'avions sauvegardés entre les intervalles qui sont définis dans les arguments.

```

if len(to_send) > 0:
    if influx.write('avions', to_send):
        log.info('Sauvegarde de {} avions vers InfluxDB.'.format(len(to_send)))
    else:
        log.info('Aucun avions sauvegardés.')

```

Pour lancer le programme on le lance donc de la manière suivante :

```
python3 adsbdatabase.py -s 127.0.0.1 -p 30003 -si 5 -u http://@IP_serveur:8186 -db dump1090
```

Le script doit être exécuté sur la machine qui envoie les données.

-s : instance dump1090

-p : port utilisé

-si : intervalle d'écriture

-u : @ serveur de la base de données

-db : nom de la base de données auquel le programme se rattache

Résultat :

```

buddy@buddy-G5-5590:~$ python3 adsbdatabase.py -s 127.0.0.1 -p 30003 -si 5 -u http://192.168.1.172:8186 -db dump1090
2020-07-28 11:06:23,480 - adsbdatabase - INFO - Namespace(dump1090_port='30003', dump1090_server='127.0.0.1', influx_database='dump1090', influx_url='http://192.168.1.172:8186', send_interval='5')
2020-07-28 11:06:23,481 - adsbdatabase - INFO - Connexion à dump1090 en TCP sur 127.0.0.1:30003.
2020-07-28 11:06:23,481 - adsbdatabase - INFO - Connexion OK, reception en cours
2020-07-28 11:06:29,417 - adsbdatabase - INFO - Sauvegarde de 1 avions vers InfluxDB.
2020-07-28 11:06:35,324 - adsbdatabase - INFO - Sauvegarde de 1 avions vers InfluxDB.
2020-07-28 11:06:40,330 - adsbdatabase - INFO - l'Avion 484133 n'a pas été vu depuis longtemps.
2020-07-28 11:06:40,330 - adsbdatabase - INFO - Aucun avions sauvegardés.
2020-07-28 11:06:45,336 - adsbdatabase - INFO - l'Avion 484133 n'a pas été vu depuis

```

```

longtemps.
2020-07-28 11:06:45,336 - adsbdatabase - INFO - Aucun avions sauvegardés.
2020-07-28 11:06:50,417 - adsbdatabase - INFO - Sauvegarde de 1 avions vers InfluxDB.
2020-07-28 11:06:55,435 - adsbdatabase - INFO - callsign ou squawk manquant pour
484163
2020-07-28 11:06:55,439 - adsbdatabase - INFO - Sauvegarde de 1 avions vers InfluxDB.

```

Partie Graphique :

Sur grafana on définit la base de donnée et on défini un dashboard qui contiendra différentes visualisations, comme un tableau avec les coordonnées (latitude et longitude), la vitesse, ainsi que l'altitude. On aura aussi un compteur pour le nombre de fois que l'avion a été enregistré. On aura une carte nous permettant de suivre le trajet de l'avion souhaité.

Sur l'exemple ci-dessous on a une requête sur la base de données qui nous montre les différentes variables qui ont été inscrit.

```

> select * from avions where time > now() - 4d
name: avions
time          alert altitude callsign callsign_1 count emergency generated hexident hexident_1 host latitude longitude onground speed spi
squawk track verticalrate
-----
1595927189000000000 false 35000 KLM1495 KLM1495 8 false 1595934388 484133 484133 debian false 452
false 2111 175 -64
1595927195000000000 false 35000 KLM1495 KLM1495 4 false 1595934393 484133 484133 debian false 452
false 2111 175 -64
1595927210000000000 false 35000 KLM1495 KLM1495 2 false 1595934410 484133 484133 debian false 452
false 2111 175 -64
1595927215000000000 false 35000 KLM1495 KLM1495 1 false 1595934410 484133 484133 debian false 452
false 2111 175 -64
1595927220000000000 false 35000 KLM1495 KLM1495 1 false 1595934416 484133 484133 debian false 452
false 2111 175 -96
1595927305000000000 false 35000 KLM1495 KLM1495 2 false 1595934503 484133 484133 debian false 454
false 2111 175 96
1595927311000000000 false 35000 KLM1495 KLM1495 2 false 1595934510 484133 484133 debian false 454
false 2111 175 -96
1595927322000000000 false 35000 KLM1495 KLM1495 1 false 1595934521 484133 484133 debian false 454
false 2111 175 -96
1595927338000000000 false 35000 KLM1495 KLM1495 2 false 1595934534 484133 484133 debian false 454
false 2111 175 -96
1595927343000000000 false 28225 NJE772C NJE772C 5 false 1595934542 493288 493288 debian false 349
false 6763 281 1376

```

Pour la partie visualisation on pourra partir sur cette requête :

```

SELECT latitude, longitude , speed, altitude FROM "avions" WHERE ("host" = 'debian')
AND time > now() - 4d GROUP BY "callsign"

```

Sur la base de données on obtient ce résultat :

```
name: avions
tags: callsign=NJE772C
time      latitude longitude speed altitude
-----
1595927343000000000 349 28225
1595927348000000000 351 28375
1595927354000000000 43.78513 3.06165 352 28575
1595927360000000000 43.78513 3.06165 352 28625
1595927365000000000 43.78513 3.06165 354 28825
1595927371000000000 43.78513 3.06165 354 28950
1595927377000000000 43.79196 3.0137 354 29125
```

On cherche la latitude, la longitude, la vitesse et l'altitude en fonction des Callsign, c'est à dire du nom de l'avions, et pour éviter d'avoir trop de données à gérer, on définira un temps pour afficher que les dernières données reçu.

Sur grafana on obtient ce résultat :

Time	avions.Callsign	avions.Latitude	avions.Longitude	avions.Speed	avions.Altitude
2020-07-28 11:08:30	KLM1495	-	-	454.00	35.00 K
2020-07-28 11:08:40	KLM1495	-	-	454.00	35.00 K
2020-07-28 11:08:50	KLM1495	-	-	454.00	35.00 K
2020-07-28 11:09:00	NJE772C	-	-	454.00	35.00 K
2020-07-28 11:09:10	NJE772C	43.79	3.06	454.00	35.00 K
2020-07-28 11:09:20	NJE772C	43.79	3.06	452.00	35.00 K
2020-07-28 11:09:30	NJE772C	43.79	3.06	452.00	35.00 K
2020-07-28 11:09:40	KLM1495	-	-	452.00	35.00 K

Illustration 12: Tableaux des valeurs vu par grafana

On a bien la latitude, la longitude, la vitesse et l'altitude en fonction des Callsign.

Pour ce qui est de la carte on doit installer un plugin, ici on utilisera le plugin TrackMap, c'est un plugin qui permet de suivre en temps réel (en fonction des paramètres de la BDD) le trajet d'un système qui envoi ses données de positions.

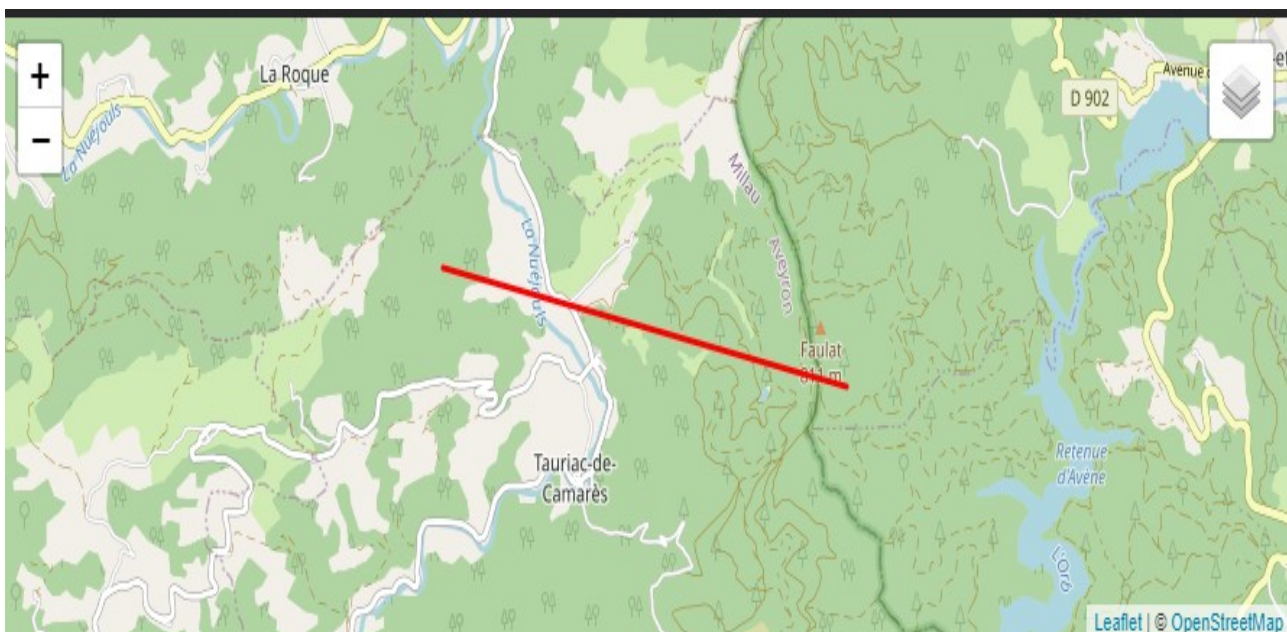
Pour l'installer, on fait de la manière suivante :

```
root@debian:/etc/telegraf# grafana-cli plugins install pr0ps-trackmap-panel
installing pr0ps-trackmap-panel @ 2.0.4
from: https://grafana.com/api/plugins/pr0ps-trackmap-panel/versions/2.0.4/download
into: /var/lib/grafana/plugins
✓ Installed pr0ps-trackmap-panel successfully
```

On redémarre le service pour grafana. Puis on définit les paramètres de test pour la visualisation.

FROM	default	avions	WHERE	callsign	=	NJE772C	+
SELECT	field (latitude)	alias (Latitude)	+				
	field (longitude)	alias (Longitude)	+				
GROUP BY	tag (callsign)	+					
FORMAT AS	Time series	▼					
ALIAS BY	Naming pattern						

La requête indiquée ci-dessus, va chercher dans la table avions, la latitude et la longitude de l'avion NJE772C, si les paramètres sont correctes et que l'avion nous a envoyé sa position on obtiendra alors son trajet du moment où on a commencé à réceptionner ses informations :



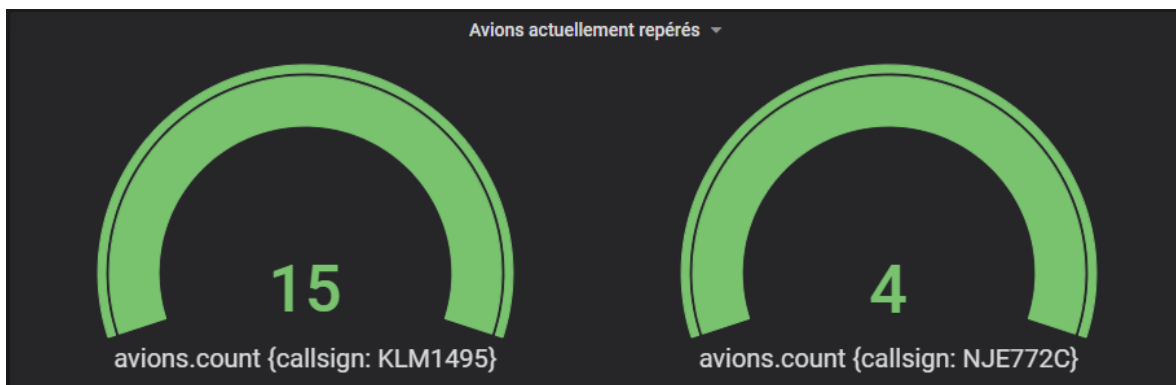
La dernière visualisation permet de savoir combien de fois un avions a donné ses informations c'est-à-dire, le nom de l'appareil, le vol, l'altitude, longitude, altitude, vitesse.

Si toutes ces conditions sont remplies alors l'avion apparaîtra sous la forme d'un compteur :



Ici on a qu'un seul avion pour le moment, dès qu'il y a plus de deux avions, le graphique fait apparaître quel avion envoie ses informations.

Avec plusieurs avions, on a un aperçu comme ceci :



Création d'une application Android permettant la consultation de la base de données

Pour se recentrer sur le thème de l'internet des objets et donc de l'objet connecté, nous souhaitons créer une application qui permettra de consulter la base de données au travers de requêtes.

Cette application se connectera à la base de données et effectuera des requêtes au travers de l'API HTTP fourni par InfluxDB.

Les requêtes se font via le navigateur web du smartphone ou de l'ordinateur et ressemble à cela :

```
http://192.168.1.172:8086/query?db=dump1090&u=admin&p=influx&q=SELECT *  
FROM avions LIMIT 2
```

Pour ce faire nous utiliserons un logiciel de création d'application que je me suis déjà servi pour mon projet en RT1, il s'agit de MIT App Inventor 2.

MIT App Inventor 2 permet de créer des applications sous Android, de manière simple et intuitive, la programmation se fait sous forme de blocks. Nous avons aussi un éditeur pour la partie graphique de l'application.

L'application dans son développement ressemble à cela :

On commencera par la partie graphique de la première page :

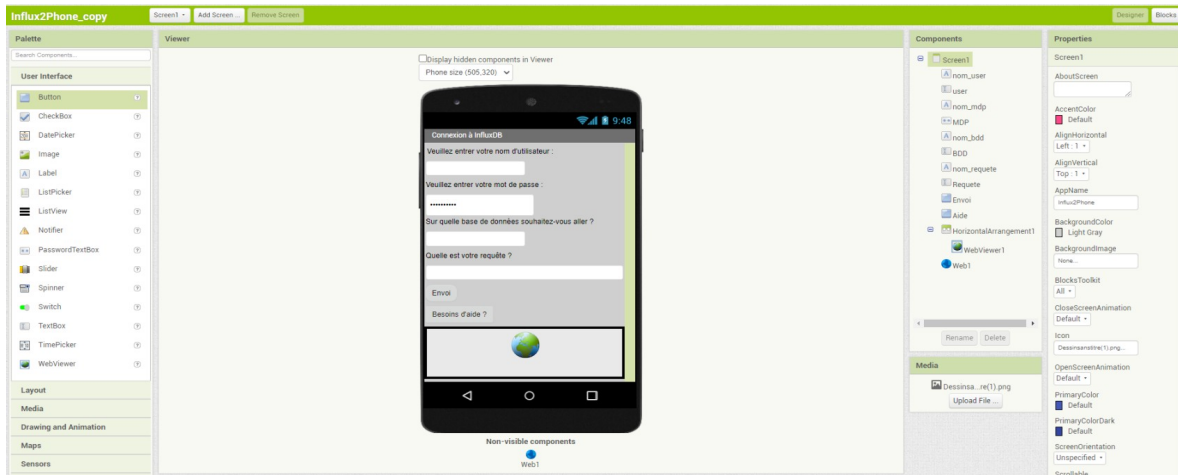


Illustration 13: Première page de l'application Influx2Phone

Nous avons ici 4 cases permettant l'écriture, on définit ici le nom d'utilisateur, le mot de passe, la base de données auquel on souhaite se connecter, ainsi qu'une case afin d'écrire une requête. Il y a un bouton « Envoi » qui comme son nom l'indique, envoie la requête.

Il y a aussi un bouton aide qui redirige vers une deuxième page, qui permet de comprendre comment se connecter et comment faire une requête et quelles commandes utilisées.

La partie avec le globe terrestre en dessous est le module Webviewer permettant l'affichage de la réponse de la requête.

La partie code ressemble se constitue de la manière suivante :

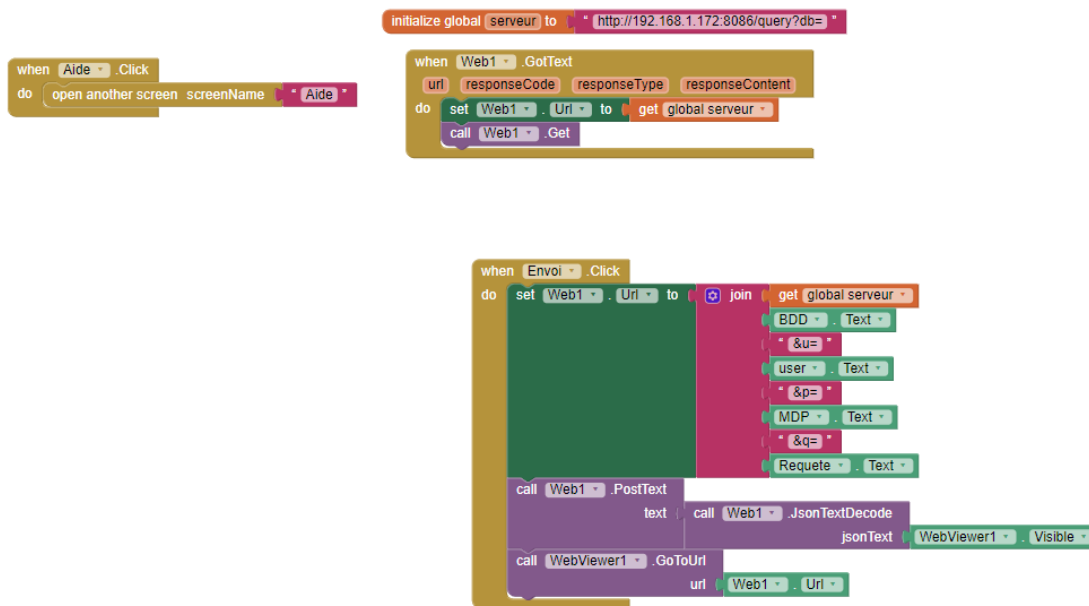


Illustration 14: Code de l'application

Si on appuie sur le bouton « Aide » il ouvre la deuxième page. Ici, on initialise le début avec l'adresse du serveur (partie orange), on est sur un serveur en local. On lui attribue le port auquel le téléphone doit se connecter, et le début de la requête. Lorsque l'on appuie sur le bouton « Envoi », on crée la requête en reprenant les valeurs de la base de données, de l'utilisateur, son mot de passe et enfin la requête. On ajoute simplement les valeurs ensemble et on ajoute entre elles les valeurs qui définissent l'utilisateur (&u=), le mot de passe (&p=) et la requête (&q=).

Une fois la requête envoyée, on appelle la fonction Web1 afin qu'il fasse apparaître le module Webviewer et que celui-ci nous affiche le résultat de la requête.

Comme il s'agit de la connexion vers une base de données au travers du protocole HTTP, aucun utilisateur et mot de passe ne sont enregistrés en local sur le téléphone.

Les identifiants sont enregistrés sur la base de données.

On va maintenant tester l'application :

Connexion à InfluxDB

Veuillez entrer votre nom d'utilisateur :

Veuillez entrer votre mot de passe :

Sur quelle base de données souhaitez-vous aller ?

Quelle est votre requête ?

Envoi

Besoins d'aide ?

```
{
  "results": [
    {
      "statement_id": 0,
      "series": [
        {
          "name": "avions",
          "columns": [
            "time", "alert", "altitude", "callsign", "callsign_1", "count", "emergency", "generated", "hexident", "hexident_1", "host", "latitude", "longitude", "onground", "speed", "spi", "squawk", "track", "verticalrate"
          ],
          "values": [
            [
              "2020-04-02T13:28:51Z", false, 39000, null, "AMC7101", 86, false, 1585841330, null, "4D21DB", "debian", 46.33566, 5.39777, false, 465, false, "2241", 134, 64
            ],
            [
              "2020-04-02T13:28:56Z", false, 38000, null, "AZA97U", 70, false, 1585841335, null, "4CA846", "debian", 46.19039, 5.24693, false, 427, false, "4021", 331, 64
            ]
          ]
        }
      ]
    }
  ]
}
```

Illustration 15: Test réussi de l'application

Lorsque l'on remplit les cases correctement et que la requête est correcte, on obtient le résultat sous la forme d'un affichage en .JSON.

Si on ne remplit pas correctement les cases, par exemple la requête est vide, on obtient le résultat :

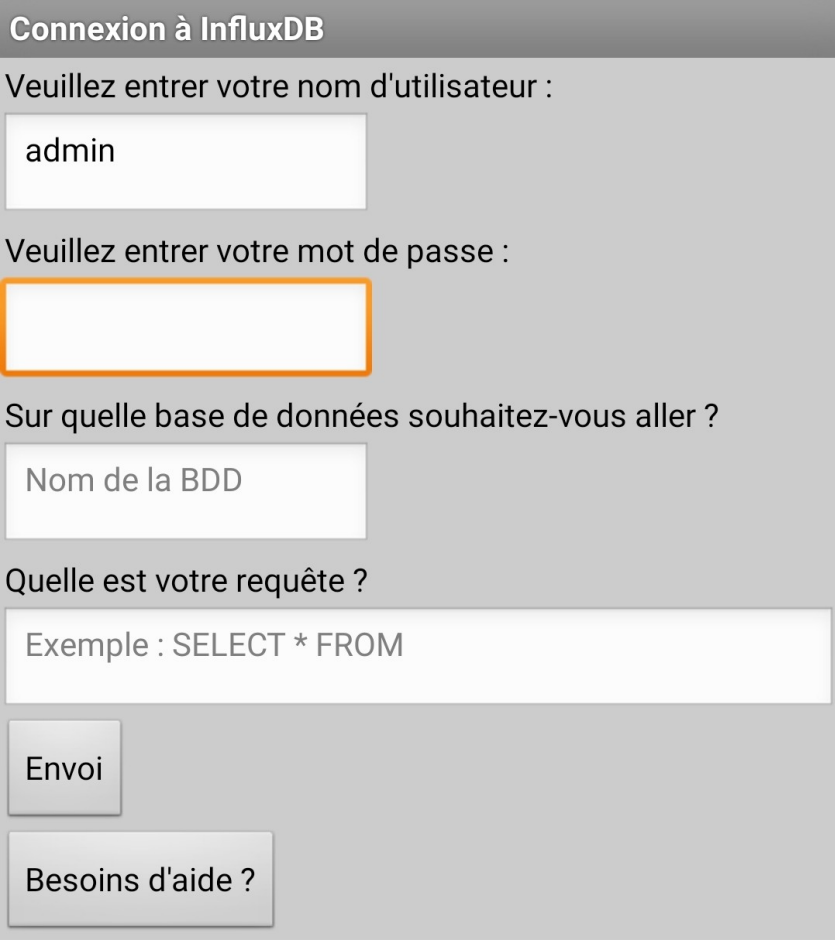
The screenshot shows a web interface for connecting to InfluxDB. It has a title bar 'Connexion à InfluxDB'. Below it are four input fields: 'Veuillez entrer votre nom d'utilisateur :' with 'admin', 'Veuillez entrer votre mot de passe :' with '.....', 'Sur quelle base de données souhaitez-vous aller ?' with 'dump1090', and 'Quelle est votre requête :'. The query field contains 'Exemple : SELECT * FROM' and is highlighted with an orange border. Below the fields are two buttons: 'Envoi' and 'Besoins d'aide ?'. At the bottom, a JSON error message is displayed:

```
{"error": "missing required parameter \"q\""} 
```

Illustration 16: Échec du à la requête

C'est la base de données qui nous répond, en spécifiant qu'il manque des paramètres.

Et enfin si les identifiants sont faux, la base de données nous prévient avec le message suivant :



The screenshot shows a web form titled "Connexion à InfluxDB". It contains four input fields: "Veuillez entrer votre nom d'utilisateur :" with the value "admin", "Veuillez entrer votre mot de passe :" (highlighted with an orange border), "Sur quelle base de données souhaitez-vous aller ?" with the value "Nom de la BDD", and "Quelle est votre requête ?" with the value "Exemple : SELECT * FROM". Below the inputs are two buttons: "Envoi" and "Besoins d'aide ?".

```
{"error": "unable to parse authentication credentials"}
```

Illustration 17: Échec identifiants

Comme spécifié précédemment, nous avons créé une section Aide du nom de « Besoin d'aide ? » sur la première page, qui permet l'affichage de l'aide pour les requêtes.

Cette page sera présentée deux fois, une fois sur l'éditeur et une autre fois sur le téléphone à cause de l'affichage des deux terminaux :

Sur l'éditeur :

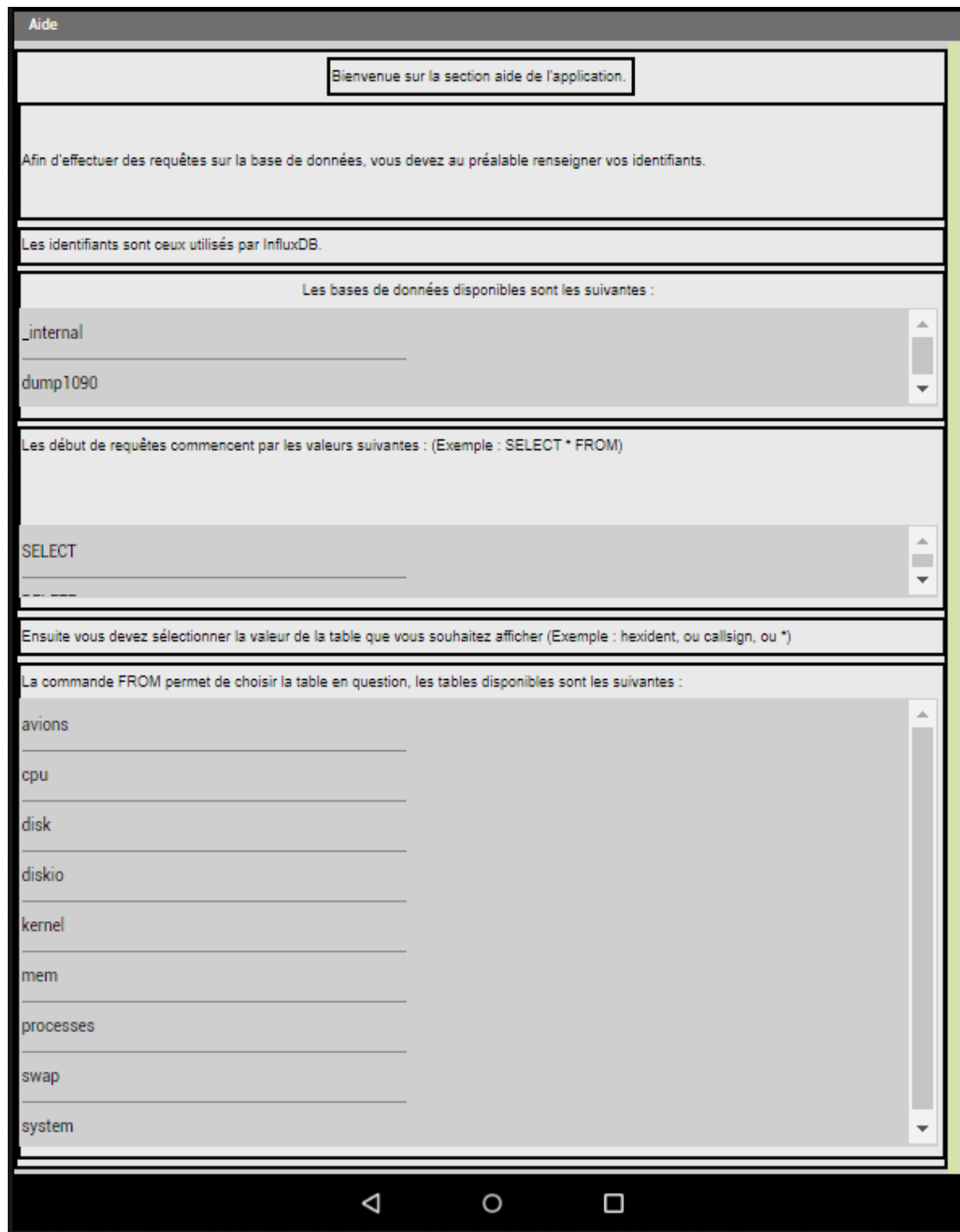


Illustration 18: Page aide de l'application sur l'éditeur

Nous avons différents textes qui apparaissent nous permettant d'écrire une requête de manière correcte. Nous avons la liste des différentes commandes présentes dans la requête. Ainsi qu'un exemple et l'explication pour le FROM et les tables.

Sur le téléphone cela apparaît de la manière suivante :

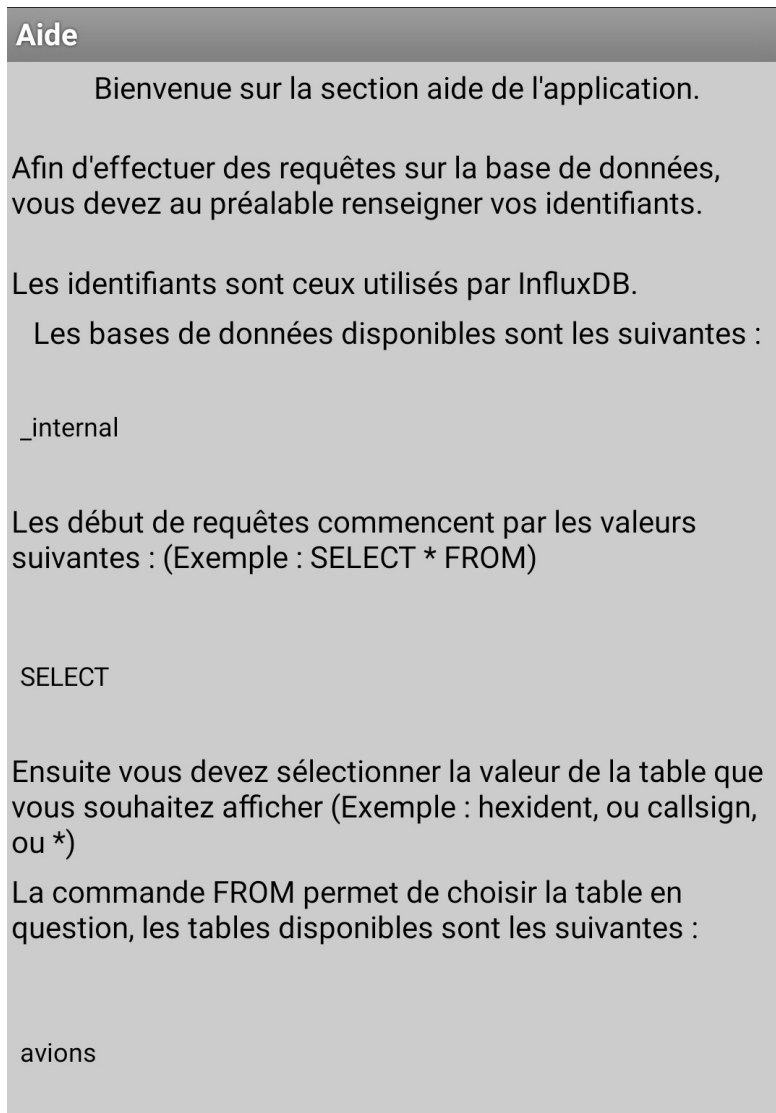


Illustration 19: Page aide sur téléphone

Les listes présentes sur l'éditeur sont confondues avec la couleur de la page.

Pour le code, ici tout se fait sur l'éditeur graphique, le seul code est celui du retour à la page 1.

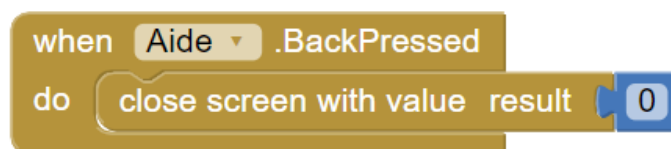


Illustration 20: Code page Aide

Conclusion de la partie « Développement de scripts et d'applications »

Cette partie a permis la connexion entre les dispositifs permettant la réception des messages des avions et les dispositifs permettant la lecture de la base de données en plus des programmes permettant la visualisation graphique des données.

Les dispositifs qui réceptionnent les messages des avions, ont la particularité de ne pas être forcément fixe, c'est-à-dire que nous pouvons en faire des stations mobiles. Afin d'avoir un suivi détaillé des avions sur tout le territoire nous pouvons fournir ces dispositifs aux routiers qui ont des itinéraires précis et aux chemineaux dont les lignes de trains sont aussi précises et régulières. Nous pouvons donc avoir des stations fixes et mobiles afin d'avoir un suivi le plus précis possible sur le territoire.

En ce qui concerne les dispositifs permettant la lecture des données, ils pourront être utilisés afin d'avoir un historique des avions (nom, vitesse, position, altitude, ...). Il permet un accès à la base de données sans avoir besoin d'un ordinateur à proximité.

Nouvelle version

En parlant de la version en développement, cette version, va effectuer une requête permettant d'avoir les derniers avions reçu (sur les deux derniers jours), la requête va s'afficher sur le navigateur de l'application.

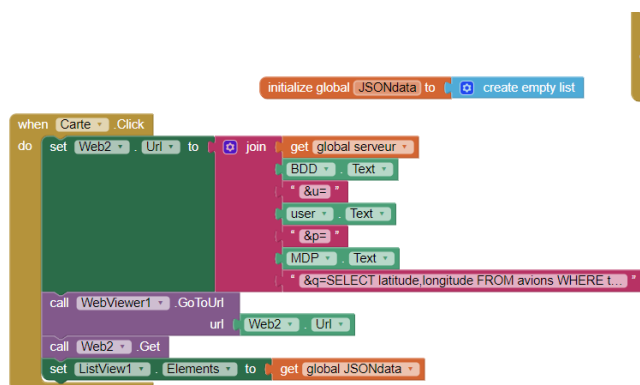


Illustration 21: Bouton Avions

Une fois reçu le texte en json de la page va être décodé avec la variable « jsondecodetext », les valeurs décodées vont être mises dans une liste.

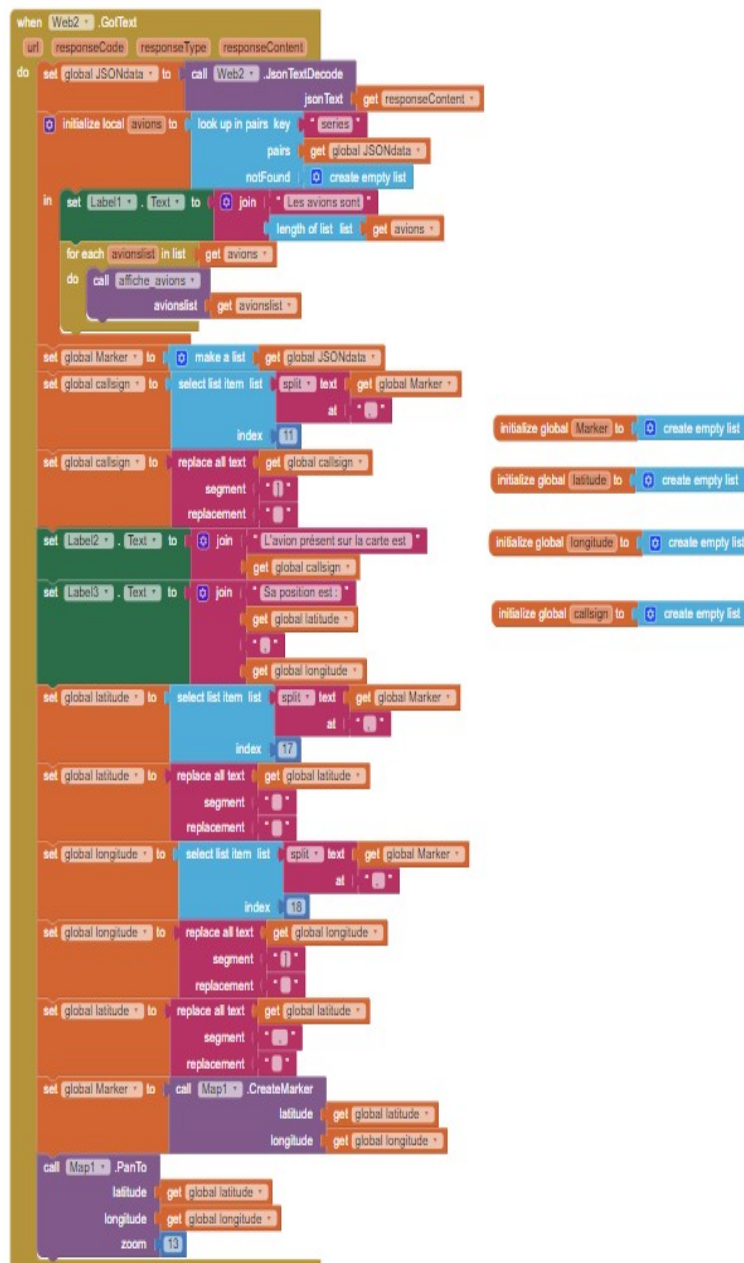


Illustration 22: Partie 2 Code

Ici, on définit une première liste nommé JSONdata qui va récupérer le contenu de la page de la requête. Ensuite, on va initialiser une variable locale qui va chercher la valeur « series » et ajouter cela à la liste JSONdata et afficher le contenu de la liste. La partie « for each avionlist » est la partie qui va essayer de chercher des valeurs précises et les afficher avec la fonction affiche_avions.

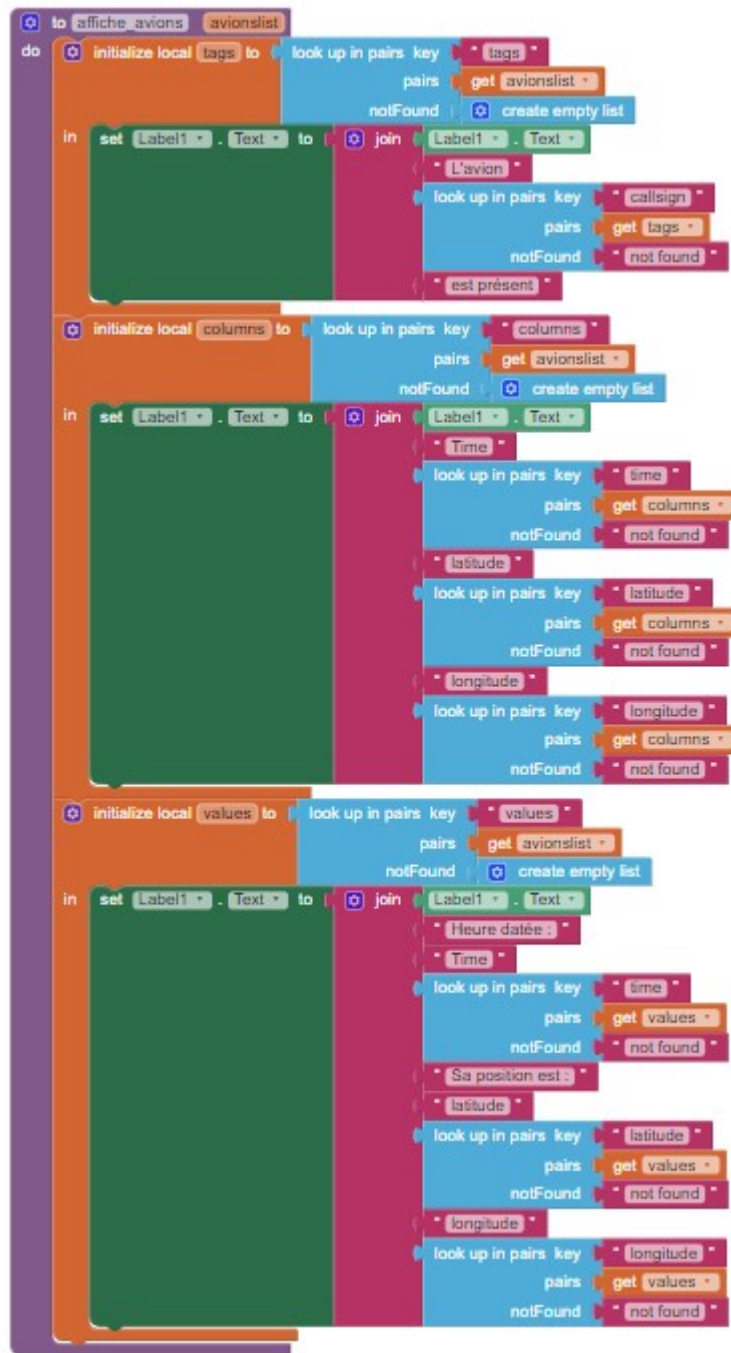


Illustration 23: Fonction *affiche_avions*

C'est cette fonction qui permet la recherche des valeurs dans la liste et de les afficher dans une « list view » sur la page principale. À chaque variable locale, on cherche les valeurs de la liste avec la fonction « look up in pairs key ». On lui donne la valeur à chercher, là où il doit chercher.

À partir de global Marker, il s'agit d'afficher un marqueur avec les coordonnées des avions. On arrive à récupérer les premières coordonnées de la liste. On les sélectionnent dans la liste et on les repère avec comme critère « , » :

```
{
  "results": [
    {
      "statement_id": 0,
      "series": [
        {
          "name": "avions",
          "tags": {
            "callsign": "CND511",
            "columns": [
              "time",
              "latitude",
              "longitude"
            ],
            "values": [
              [
                "2020-07-31T15:37:59Z",
                43.36806,
                3.1551
              ]
            ]
          }
        }
      ]
    }
  ]
}
```

Ainsi que son numéro par rapport à la virgule ici l'index 17 et 18. Pour le moment on se focalise sur un seul point afin de voir si le point s'affichera sur la carte. De plus, on prend la première valeur car on est sûr que l'index ne changera pas, car on n'a pas le même nombre d'informations par avions ni la limite des envois de messages.

Il faudra aussi mettre dans le bon format les valeurs pour les coordonnées, nous avons eu des soucis par rapport à cela, on avait comme valeurs pour la latitude des espaces en trop et une virgule qui empêchait la lecture, pour la longitude il s'agissait d'un «] ».

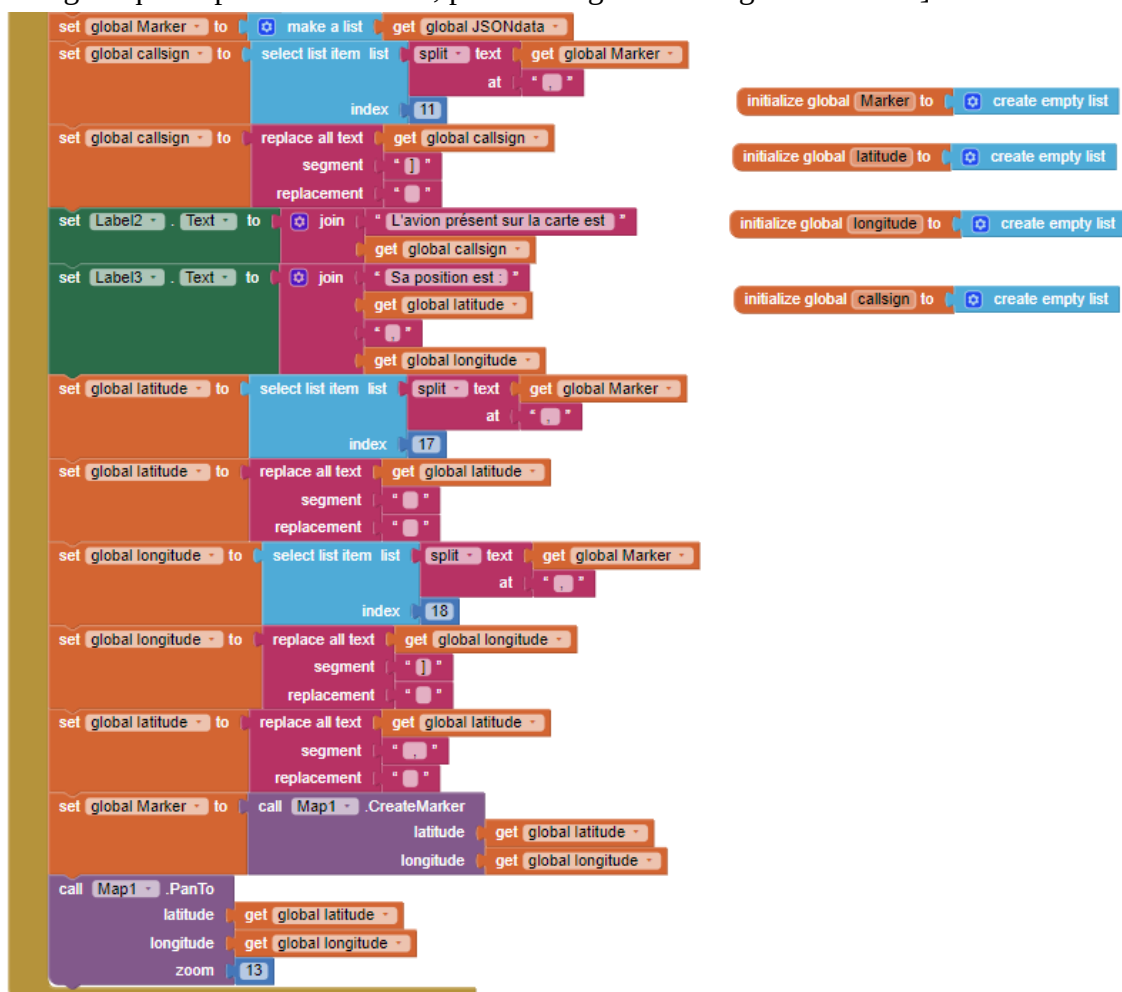


Illustration 24: Partie carte

Le schéma ci-dessus est la partie correspondant à la récupération des points, on crée des variables/listes pour le « marker », la latitude et la longitude. Comme indiqué plus haut, on récupère les index 17 et 18, que l'on remet dans le bon format, afin d'avoir « latitude, longitude ».

On lance ensuite la carte avec les variables latitudes et longitude, et on lui indique de créer un « marker » avec ces points-là. On affiche aussi l'avion qui apparaît sur le point (Label2), ainsi que sa position (Label3).

Le résultat ressemble à cela :

```

(results (((series (((columns
(time latitude longitude))
(name avions) (tags
((callsign CND511))) (values
((2020-07-31T15:37:59Z
43.36806 3.1551)
(2020-07-31T15:38:04Z 43.35726
{"results":[{"statement_id":0,"series":
[{"name":"avions","tags":
{"callsign":"CND511"},"columns":
[["time","latitude","longitude"],"values":
[["2020-07-31T15:37:59Z",43.36806,3.1551],
[["2020-07-31T15:38:04Z",43.35726,3.15301],
[["2020-07-31T15:38:09Z",43.34861,3.15138],
[["2020-07-31T15:38:15Z",43.33498,3.1488],
[["2020-07-31T15:38:20Z",43.32678,3.1472],
[["2020-07-31T15:38:26Z",43.3163,3.14522],
[["2020-07-31T15:38:31Z",43.30215,3.14248],
[["2020-07-31T15:38:36Z",43.28811,3.13971]

```

*Illustration 25: Vision
détaillée de l'application en
fonctionnement*

Si besoin, le bouton « Besoins d'aide » vous sera utile.

N.B : L'application et son projet seront disponible avec l'archive (les deux versions).

Conclusion du projet

Avant de démarrer ce projet, je ne pensais pas qu'il était possible de récupérer les informations des avions. Je pensais que seules les compagnies qui s'occupaient de ce domaine et les aéroports avaient ce genre d'informations. Mais en fait, le suivi des appareils est communautaire, tout le monde avec un peu de matériel peut réceptionner et envoyer les informations. Ce genre d'initiative est pratique en cas de perte d'un appareil par les tours de contrôle ou les autres avions.

Tenter de développer une solution de surveillance et de stockage avec la possibilité de d'avoir accès à l'intégralité des données depuis un terminal Android afin d'avoir un suivi et permettre d'afficher une carte avec la position d'au moins un avion a été très instructif. J'ai pu utiliser les API d'InfluxDB, afin d'effectuer des requêtes pour sauvegarder et lire les données. Mais j'ai aussi pu développer ma propre application Android permettant d'avoir accès à cette base. Il n'y a pas de documentation spécifique pour InfluxDB avec Mit App Inventor. Ce qui fait de cette application une base pour d'autres projets tournant autour d'InfluxDB.

Bien que je dois rendre mon rapport, nous allons continuer d'explorer les différentes solutions possibles.

Sources :

- https://fablab.coagul.org/SDR_%C3%A0_l%27aide_d%27un_r_%C3%A9cepteur_DVB-T_avec_une_puce_RTL2832U#R.C3.A9ception_ADS-B.2C_transpondeur_d.27avion
- <https://www.dataero.fr/comment-ecouter-les-avions-a-la-radio/>
- <https://cincinnatiavionics.com/ads-b-out-difference-1090-978/>
- https://www.icao.int/APAC/Documents/edocs/cns/ADSB_AIGD7.pdf
- [https://www.icao.int/SAM/Documents/2017-ADSB/08%20FAA%20Briefing%20ADS_B%20Rules%20and%20Airspace%20\(2\).pdf](https://www.icao.int/SAM/Documents/2017-ADSB/08%20FAA%20Briefing%20ADS_B%20Rules%20and%20Airspace%20(2).pdf)
- <https://github.com/antirez/dump1090>
- <https://github.com/flightaware/dump1090>
- <https://github.com/kanflo/ADS-B-funhouse>
- <https://medium.com/@arunvenkats/automating-the-capture-of-airplane-pictures-with-raspberry-pis-ads-b-and-iot-software-39e25ddcf3ea> (Curiosité)
- <https://www.raspberrypi.org/forums/viewtopic.php?t=182268>
- https://docs.w3cub.com/influxdata/telegraf/v1.3/concepts/data_formats_input/#json
- https://docs.influxdata.com/telegraf/v1.12/data_formats/output/json/
- <https://github.com/claws/dump1090-exporter>
- <https://grafana.com/grafana/dashboards/768>
- <https://community.grafana.com/t/using-latitude-and-longitude-as-influx-tags-in-world-map-panel/10996>
- <https://grafana.com/grafana/plugins/pr0ps-trackmap-panel>
- <https://docs.python.org/fr/3.5/library/stdtypes.html#typesnumeric>
- <https://docs.python.org/3/howto/regex.html>
- <https://docs.python.org/3/library/re.html>
- https://docs.influxdata.com/influxdb/v1.8/guides/query_data/
- <https://docs.influxdata.com/influxdb/v1.8/tools/api/#influxdb-1-x-http-endpoints>
- <https://appinventor.mit.edu/>