

## **TODOs: Deadline le 11 Novembre**

Regarder les différentes routes de [opensky](#) pour voir si vous avez d'autres informations qui peuvent vous permettre de relier un avion un aéroport une compagnie etc.

⇒ *Chaque mois, un Dump sous format csv est mis à disposition (<https://opensky-network.org/datasets/metadata/>). Ce dump ayant pour nom de fichier aircraft-Database-complete-\$year-\$month.csv contient des informations sur les avions, leurs caractéristiques ainsi que la compagnie aérienne.*

Explorer la composition du code

Définir la stratégie de stockage (choisir et justifier le choix des technologies)

Créer les fichiers de création de base, de chargement de base et de requêtes  
compléter le Readme de votre repo Git

## **Prochain objectif:**

### **Étape 2/ Organisation des données :**

Il s'agira de la partie la plus importante de votre projet où vous ferez le cœur du métier de Data Engineer.

On vous demande d'organiser les données via différentes bases de données :

Relationnelle

NoSQL

Il faudra penser à l'architecture des données, notamment comment relier les différentes données entre elles.

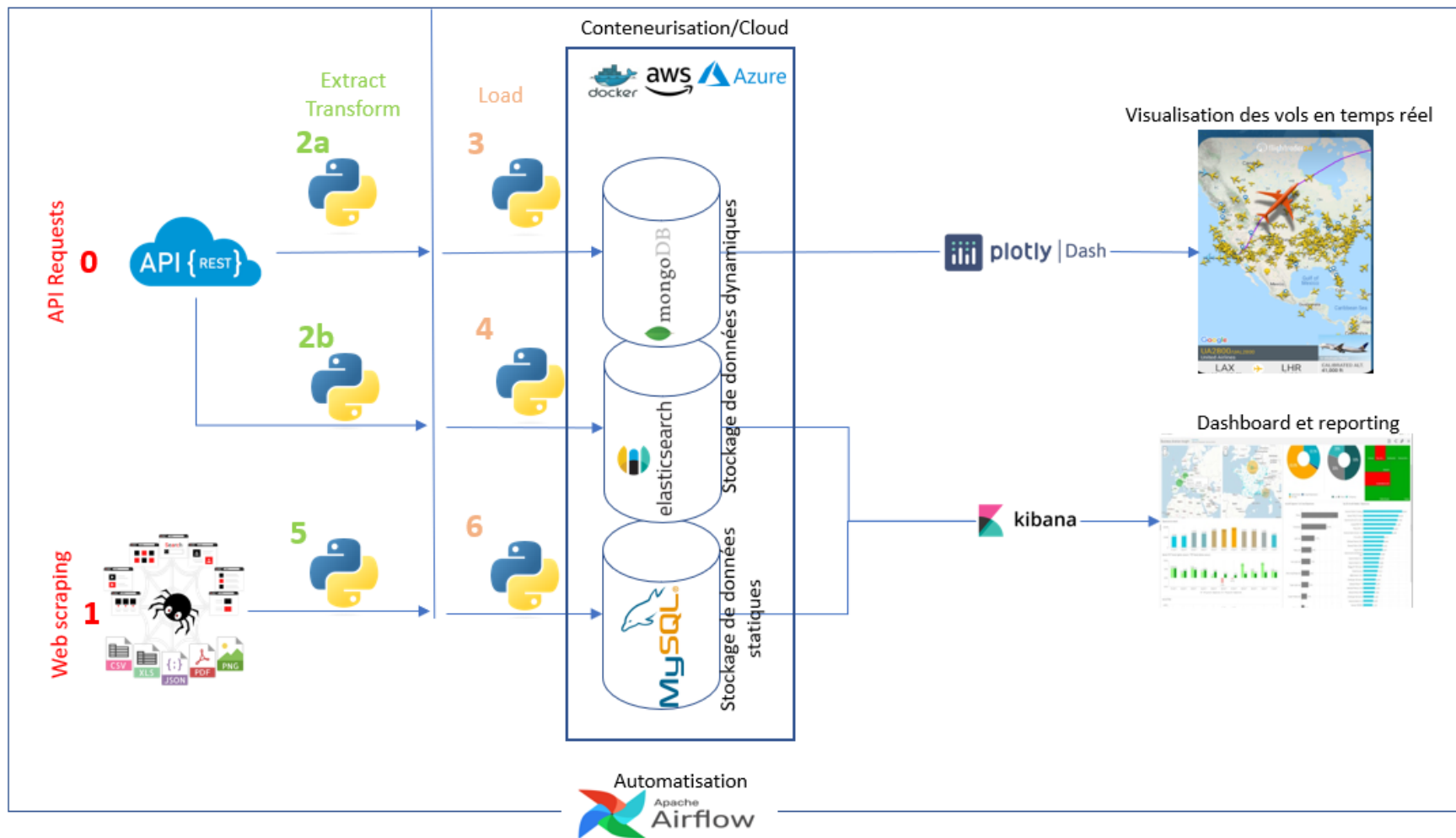
Livrable :

Tout document expliquant l'architecture choisie (Diagramme UML)

Fichier implémentant les bases de données

Fichier de requête

Nous proposons de développer une application permettant d'afficher le trafic aérien en temps réel et faire des analyses et des requêtes sur les informations collectées et stockées dans des bases de données. Pour expliquer notre démarche, nous proposons le workflow ci-dessous.



Tous les scripts se trouvent dans le lien github:

**[https://github.com/rim-DE/DST-Airlines/tree/feature/collect\\_save\\_flight\\_data](https://github.com/rim-DE/DST-Airlines/tree/feature/collect_save_flight_data)**

Les différentes étapes du framework (numérotés dans la figure ci dessus) ainsi que les choix proposés sont expliqués ci-dessous:

## 0. API Requests

Les données utilisées pour ce projet sont collectées en utilisant l'API OpenSky (<https://openskynetwork.github.io/opensky-api/>). Nous avons choisi cette API, car elle est gratuite et offre un nombre très important d'appels à l'API (2000 appels par jour).

Deux types d'informations sont collectées:

- Les informations des positions d'avions: ces informations doivent être récupérées en continu et seront utilisées pour afficher les positions des avions dans le monde entier en temps réel.

⇒ Affichage sur une map en utilisant Dash.

PS: avec 2000 appels autorisés par jour, nous pouvons récupérer les données toutes les 45 secondes.

- Les informations sur les vols (eg. aéroport de départ, aéroport d'arrivée, date de départ,, etc.). Ces informations sont mises à jour par un traitement batch la nuit, c'est-à-dire que seuls les vols de la veille ou d'avant sont disponibles. Pour cela, nous avons choisi de récupérer ces informations une fois par jour.

⇒ Dashbord et reporting

## 1. Web Scraping

Nous avons utilisé les données de cette page web: <https://www.world-airport-codes.com/alphabetical/airport-code/a.html>

Les codes ICAO sont classifiés par leurs initiales alphabétiques. Pour chaque alphabet, les codes sont répartis sur plusieurs pages. Nous avons donc parcouru toutes les pages de chaque lettre alphabétique. Pour chaque page on scrappe les données suivantes :

- ICAO: est un code de classement géographique à quatre lettres attribué à chaque aéroport à travers le monde par l'Organisation de l'aviation civile

internationale. Ils sont utilisés lors du contrôle de la circulation aérienne et dans les opérations telles que le plan de vol. Ce code correspond au “*estDepartureAirport*” et “*estArrivalAirport*” qui sont deux variables extraites par l’API Opensky.

- IATA: Ces codes ne sont pas les mêmes que les codes ICAO. Les codes IATA sont plus généralement visibles du grand public et utilisés pour les horaires des lignes aériennes, les réservations, et le marquage des bagages.
- Taille aéroport: il prendra une valeur parmi celles-ci: {“Small airport”, “Medium airport”, “Large airport ”, “closed”}
- Pays : c’est le pays dans lequel se trouve l’aéroport.
- Ville : la ville exacte où se situe l’aéroport
- Nom de l’aéroport

## **2.a Extraction des données de l’API (les positions)**

Il s’agit d’extraire les données des positions des avions en temps réel. Pour ce faire, nous avons créé la classe *extract\_aircraft\_position\_data.py* permettant de sauvegarder les données d’un avion avec son path (liste de points enregistrés pour cet avion depuis son départ jusqu’à l’instant actuel, latitude longitude...). A partir de ces points on peut tracer le chemin d’un vol, ou on peut récupérer sa dernière position. On peut enrichir / modifier cette classe selon notre besoin.

## **2.b Extraction des données de l’API (les vols)**

Le script utilisé pour extraire les données des vols s’appelle *extract\_flight\_data.py*. Les données récupérées par ce script concernent les vols qui ont eu lieu la veille avant les informations concernant l’aéroport de départ, d’arrivée, la date de départ, etc.

Étant donné que seuls les vols de la veille ou d’avant sont disponibles, nous avons choisi de lancer ce script tous les jours à 8h du matin. Les données sont récupérées par plage de deux heures et sont sauvegardées dans un fichier json.

Nous proposons de programmer un crontab afin d’automatiser cette tâche et d’exécuter ce script tous les jours à 8h du matin.

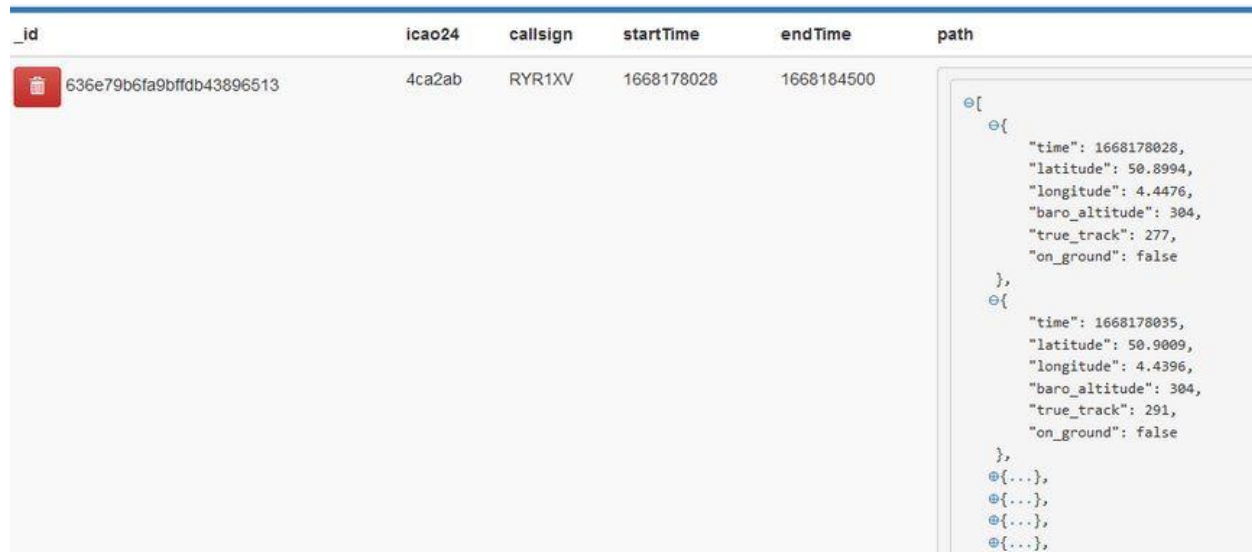
## **3. Importation des positions des avions dans une base MongoDB:**

MongoDB est une base de données orientée document avec d’excellentes performances, de très bonne scalabilité et qui est compatible avec les technologies cloud comme AWS, Google Cloud et Azure. Pour toutes ces raisons, nous avons choisi

de stocker les données récupérées en temps réel dans une base MongoDB. Les données stockées peuvent être analysées plus tard.

Le script utilisé s'appelle *load\_aircraft\_position\_data\_in\_mongodb*

Ci-dessous une capture d'écran montrant les données importées dans mongoDB.



_id	icao24	callsign	startTime	endTime	path
636e79b6fa9bffd43896513	4ca2ab	RYR1XV	1668178028	1668184500	[{"time": 1668178028, "latitude": 50.8994, "longitude": 4.4476, "baro_altitude": 304, "true_track": 277, "on_ground": false}, {"time": 1668178035, "latitude": 50.9009, "longitude": 4.4396, "baro_altitude": 304, "true_track": 291, "on_ground": false}, ...]

On voit les données suivantes :

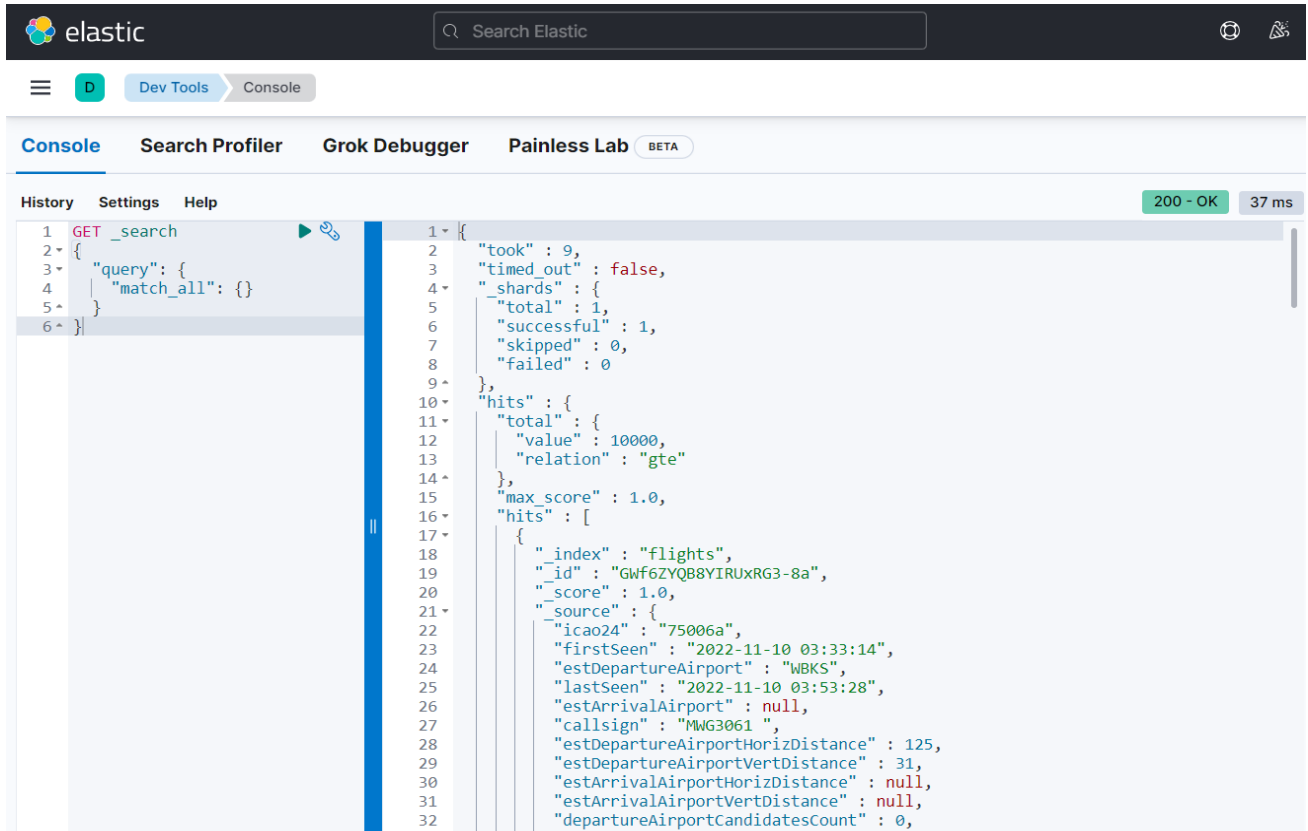
- Le code icao24 pour identifier l'avion
- Le path: une liste de toutes les positions enregistrées pour ce vol, la dernière étant sa position en cours.
  - Dans chaque élément de cette liste:
  - Time: le timestamp de la position
  - Latitude & longitude : la latitude et longitude du vol
  - On\_ground: un boolean qui précise si l'avion a atterri ou non.

#### 4. Importation des données des vols dans une base Elasticsearch

ElasticSearch est une base de données orientée document qui est optimisée pour faire de la recherche. De plus, il embarque une composante Kibana, permettant de faire de la visualisation de données. En connectant notre base de données MySQL à ElasticSearch et en stockant les données collectées en temps réel comme dans une architecture Lambda, nous pouvons visualiser ces données dans un tableau de bord conçu sous Kibana et effectuer des requêtes sur celles-ci.

Le script permettant de faire l'importation de nos données dans Elasticsearch s'appelle *load\_flight\_data\_in\_elasticsearch.py* et il se trouve dans notre repo Github. Ce script permet de stocker les données des vols qui on

Ci-dessous une capture d'écran montrant les données importées dans elasticsearch.



The screenshot shows the Elastic Dev Tools interface. The 'Console' tab is active, displaying a GET \_search query and its response. The query is a match\_all query. The response is a JSON object indicating a successful search with 10,000 hits. The first hit is expanded, showing flight data for index 'flights' and id 'GWf6ZYQB8YIRUXRG3-8a'.

```
1 GET _search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }

1 {
2   "took": 9,
3   "timed_out": false,
4   "_shards": {
5     "total": 1,
6     "successful": 1,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 10000,
13      "relation": "gte"
14    },
15    "max_score": 1.0,
16    "hits": [
17      {
18        "_index": "flights",
19        "_id": "GWf6ZYQB8YIRUXRG3-8a",
20        "_score": 1.0,
21        "_source": {
22          "icao24": "75006a",
23          "firstSeen": "2022-11-10 03:33:14",
24          "estDepartureAirport": "WBKS",
25          "lastSeen": "2022-11-10 03:53:28",
26          "estArrivalAirport": null,
27          "callsign": "MWG3061",
28          "estDepartureAirportHorizDistance": 125,
29          "estDepartureAirportVertDistance": 31,
30          "estArrivalAirportHorizDistance": null,
31          "estArrivalAirportVertDistance": null,
32          "departureAirportCandidatesCount": 0,
```

On voit les données suivantes:

- Icao24 (identifiant de l'avion)
- firstSeen (date de départ)
- estDepartureAirport (aéroport de départ)
- etc

## 5. Extraction des données scrappée

Le script Python aillant permis la collecte est disponible sur le lien suivant:  
[https://github.com/rim-DE/DST-Airlines/blob/main/airport\\_info\\_scrapping\\_modifi%C3%A9.py](https://github.com/rim-DE/DST-Airlines/blob/main/airport_info_scrapping_modifi%C3%A9.py)

Ci-dessous un aperçu des données collectées:

	Aéroport	Ville	Pays	Taille	IATA	ICAO
2460	Manas International	Bishkek	Kyrgyzstan	Largeairport	FRU	UCFM
2461	Francistown	Francistown	Botswana	Mediumairport	FRW	FBFT
2462	Eastern Slopes Regional	Fryeburg	United States	Smallairport	FRY	KIZG
2463	Fritzlar	Fritzlar	Germany	Mediumairport	FRZ	ETHF
2464	Figari Sud-Corse	Figari Sud-Corse	France	Mediumairport	FSC	LFKF
2465	Joe Foss Field	Sioux Falls	United States	Mediumairport	FSD	KFSD
2466	Henry Post Army Air Field (Fort Sill)	Fort Sill	United States	Mediumairport	FSI	KFSI
2467	Fossil Downs	Fossil Downs Station	Australia	Closed	FSL	
2468	Fort Smith Regional	Fort Smith	United States	Largeairport	FSM	KFSM
2469	Haley Army Airfield	Fort Sheridan	United States	Closed	FSN	
2470	St Pierre	Saint-Pierre	Saint Pierre and Miquelon	Mediumairport	FSP	LFVP
2471	RAF Kinloss	Kinloss	United Kingdom	Mediumairport	FSS	EGQK
2472	Fort Stockton Pecos County	Fort Stockton	United States	Mediumairport	FST	KFST
2473	Fort Sumner Municipal	Fort Sumner	United States	Smallairport	FSU	KFSU
2474	Mt. Fuji Shizuoka	Makinohara / Shimada	Japan	Largeairport	FSZ	RJNS

## 6. Importation des données scrappés dans une base MySQL:

### *Etude du choix de la Base de données relationnelle*

Afin qu'on puisse faire le choix de la base des données relationnelle la plus adaptée à nos données, nous allons comparer les plus populaires: SQLServer, Oracle, MySQL et Postgree. C'est vrai que notre base contient une seule table de moins de 1000 lignes, donc n'importe quelle base peut être utilisée. Mais dans une vision de long terme, nous allons essayer de mettre des choix techniques avec des données évolutives au cours du temps.

#### ORACLE/SQLServer

- Très performant côté sécurité et récupération des données et capacité de stockage .
- Documentation très complète et support technique solide
- Coût très élevé
- Consomme énormément de ressources lors de l'installation et pour la mise à jour.

#### MySQL

- Syntaxe simple et faible complexité
- Installation gratuite

- Pris en charge avec la plupart des fournisseurs de cloud (microsoft azure, amazon, etc)
- Pas adaptée pour des données à évolution infini

### PostgreSQL

- SGBD objet relationnel (orienté objet)
- entièrement compatible avec SQL
- prend en charge le stockage d'objets volumineux
- Flexible

### PostgreSQL vs MySQL

#### ❖ Vitesse:

**MySQL** : plus rapide pour des commandes de lecture seule

**PostgreSQL**: fonctionne mieux avec des opérations en lecture-écriture, des ensembles de données massifs et des requêtes compliquées.

#### ❖ Architecture:

**MySQL**: base de données purement relationnelle,

**PostgreSQL**: base de données objet-relationnelle --> offre des types de données plus sophistiqués et permet aux objets d'hériter de propriétés → le travail plus complexe

**MySQL**: utilise un seul processus et maintient un seul thread pour chaque connexion.

**PostgreSQL**: génère un nouveau processus système via son allocation de mémoire pour chaque connexion client établie. Cela nécessite beaucoup de mémoire sur les systèmes comportant un grand nombre de connexions client.

#### Choix effectué est MySQL, parce que:

→ Est plus approprié pour les applications d'envergure inférieure à celle d'une entreprise (Telle notre projet)

→ S'adapte à nos données parce que nous avons just deux tables dont le nombre de lignes (aéroports/ compagnies aérienne) qui n'évolue que rarement (lors du construction d'un nouvel aéroport dans le monde/ naissance d'une nouvelle compagnie aérienne)

→ Dans notre projet on va juste lire de la base, on ne va pas la modifier. Comme déjà expliqué, MySQL est plus rapide en mode lecture.

Explication des scripts dans le répertoire MySQL :[lien-git](#)



1. Création d'une image Docker MySQL initialisée avec un script de création de la base des données MySQL (Dstairlines) et les tables (schema.sql [schema.sql](#)).
2. Nettoyage et enrichissement du script qui scrappe les données HTML: encapsuler le code dans une méthode, ajout d'une méthode qui génère une liste des tuples à partir des listes des données scrappées. Cette méthode facilitera l'insertion des données dans la table de mysql dans la clause VALUES. [html\\_scrapping.html](#)
3. Création d'un script qui établit la connexion avec la base, remplit les tables et ferme la connexion. Le traitement des exceptions est aussi abordé (try, except, finally) [remplissage\\_dbMySQL.py](#)
4. Mise à jour de Docker-compose en ajoutant l'image MySQL: [docker-compose](#)

Imprime-écran d'un échantillon de la base remplie:

The screenshot shows the DBeaver 22.2.4 interface with the 'aeroports' table selected. The table contains 600 rows of airport data. The columns are: ICAO, IATA, nom, taille, pays, and ville. The data is displayed in a grid view.

	ICAO	IATA	nom	taille	pays	ville
437	AUB	AUB	Itauba	Closed	Brazil	Itauba
438	SKUC	AUC	Santiago Perez	Mediumairport	Colombia	Arauca
439	YAGD	AUD	Augustus Downs	Smallairport	Australia	
440	AUE	AUE	Abu Rudeis	Smallairport	Egypt	Abu Rudeis
441	LFLA	AUF	Auxerre-Branches	Mediumairport	France	Auxerre/Branc
442	KAUG	AUG	Augusta State	Mediumairport	United States	Augusta
443	OMAA	AUH	Abu Dhabi International	Largeairport	United Arab Emirates	Abu Dhabi
444	AYND	AUI	Aua Island	Smallairport	Papua New Guinea	Aua Island
445	AYAT	AUJ	Ambunti	Smallairport	Papua New Guinea	Ambunti
446	PAUK	AUK	Alakanuk	Smallairport	United States	Alakanuk
447	AUL	AUL	Aur Island	Smallairport	Marshall Islands	Aur Atoll
448	KAUM	AUM	Austin Municipal	Smallairport	United States	Austin
449	KAUN	AUN	Auburn Municipal	Smallairport	United States	Auburn
450	KAUO	AUO	Auburn Opelika Robert	Smallairport	United States	Auburn
451	AYAG	AUP	Agaun	Smallairport	Papua New Guinea	
452	NTMN	AUQ	Hiva Oa-Atuona	Mediumairport	French Polynesia	Hiva Oa Island
453	LFLW	AUR	Aurillac	Mediumairport	France	Aurillac
454	KAUS	AUS	Austin Bergstrom Inter	Largeairport	United States	Austin
455	WPAT	AUT	Atauro	Smallairport	Timor-Leste	Atauro
456	YAUR	AUU	Aurukun	Smallairport	Australia	Aurukun