2024/2025

##### Rapport du Projet de NOSQL & BIG DATA

Amine Aichane Amira Belafkih Ikram Besbassi Hamza Chraim

Table des matières

[**I – INTRODUCTION** 2](#_Toc199170304)

[**II – Contexte Général** 4](#_Toc199170305)

[**III – Problématique** 7](#_Toc199170306)

[**IV – Description du projet** 10](#_Toc199170307)

[**V – Fonctionnement du projet** 13](#_Toc199170308)

[**VI – Réalisation** 16](#_Toc199170309)

[**VII – Conclusion** 20](#_Toc199170310)

# 

# **I – INTRODUCTION**

Le transport aérien, véritable pilier de la mondialisation, a connu une croissance exponentielle au cours des dernières décennies, au point de devenir un élément central des échanges économiques, touristiques et culturels à l’échelle planétaire. Cette expansion massive a engendré une complexification croissante de la gestion du trafic aérien, rendant indispensable l’utilisation de technologies avancées pour assurer la sécurité des vols, optimiser les itinéraires et fournir aux passagers une information fiable et en temps réel.

Dans ce contexte technologique en constante évolution, le traitement des données aériennes représente un enjeu stratégique majeur. Le suivi précis et en temps réel des avions permet non seulement d’améliorer la sécurité des vols, mais aussi d’anticiper les imprévus, de réduire les retards, d’optimiser les ressources aéroportuaires, et de renforcer l'expérience client. Il devient ainsi crucial pour les compagnies aériennes, les gestionnaires d’aéroports, les autorités de régulation et les fournisseurs de services numériques d’adopter des solutions innovantes capables de collecter, analyser, stocker et visualiser d’énormes volumes de données issues de multiples sources.

Le projet que nous présentons dans ce rapport s’inscrit pleinement dans cette dynamique. Il a pour objectif de concevoir une plateforme complète et modulaire, reposant sur des technologies Big Data et NoSQL, pour assurer le suivi en temps réel des vols aériens. Ce système s’appuie sur un ensemble d’outils modernes et performants (Kafka, MongoDB, Flask, Leaflet.js, Tailwind CSS) afin de traiter les données à haute fréquence, de manière fiable, rapide et évolutive. L’architecture choisie permet une forte résilience aux erreurs, une grande souplesse d’adaptation aux changements de charge, et une excellente expérience utilisateur grâce à une interface interactive et responsive.

À travers cette réalisation, nous mettons en œuvre les principes fondamentaux de l’ingénierie logicielle, de l’analyse de données, et de l’architecture distribuée. Il ne s’agit pas seulement d’un exercice académique, mais d’un véritable projet applicatif qui répond à des besoins concrets du secteur aéronautique. En offrant des outils d’analyse, de prédiction et de visualisation adaptés, ce projet illustre les apports des technologies Big Data dans la transformation numérique de l’industrie du transport aérien.

# **II – Contexte Général**

L’évolution fulgurante des technologies de l’information au cours des vingt dernières années a transformé notre manière de collecter, traiter et interpréter les données. Dans le secteur aérien, cette transformation est particulièrement visible avec l’essor des systèmes de suivi de vols en temps réel. Grâce à la démocratisation de l’accès à l’Internet, à la prolifération des capteurs embarqués (tels que les transpondeurs ADS-B) et à la mise à disposition de données ouvertes via des API publiques, il est aujourd’hui possible d’avoir une visibilité quasi instantanée sur les mouvements des aéronefs à travers le monde.

Les compagnies aériennes, les gestionnaires d’aéroports, les plateformes de réservation, les développeurs d’applications mobiles et même les particuliers peuvent désormais exploiter ces données pour divers usages : information des passagers, prévisions de trafic, gestion des ressources au sol, ou encore analyses prédictives. Cette richesse informationnelle ouvre la voie à de nouveaux services innovants, mais elle soulève également des défis techniques importants.

En effet, le volume et la vélocité des données générées exigent des systèmes capables de gérer des flux continus (data streams) tout en maintenant une faible latence. Il ne s’agit plus uniquement de stocker des données de manière statique, mais bien de les traiter en temps réel pour fournir des visualisations actualisées et des alertes pertinentes. Les technologies traditionnelles de base de données relationnelle atteignent ici leurs limites, ce qui justifie le recours à des solutions Big Data comme Apache Kafka pour la gestion des flux, et MongoDB pour le stockage flexible et rapide des documents structurés.

Les enjeux sont nombreux : il faut garantir la fraîcheur des données (c’est-à-dire leur actualisation fréquente), leur fiabilité (en filtrant les erreurs ou doublons), leur accessibilité (via des API performantes), et leur visualisation (à travers des interfaces intuitives, interactives et adaptatives). Par ailleurs, la montée en charge progressive du système – avec l’augmentation constante du trafic aérien et du nombre d’utilisateurs – impose une architecture hautement scalable et tolérante aux pannes.

Notre projet répond précisément à ces besoins. Il vise à concevoir un système capable d’agréger les données de vol en temps réel, de les traiter automatiquement, de les stocker de manière optimisée, et de les afficher sur une interface utilisateur ergonomique. Cette approche nous permet de mettre en pratique les concepts clés étudiés dans le cadre du module NoSQL & Big Data, tout en proposant une solution réaliste et pertinente pour les professionnels du secteur aérien et les utilisateurs finaux.

# **III – Problématique**

À l’ère du numérique, où l’instantanéité de l’information est devenue une exigence, la gestion des données aériennes en temps réel représente un défi technologique de premier plan. Les avions génèrent en continu une grande quantité de données liées à leur position, altitude, vitesse, statut de vol, trajectoire, etc. Or, ces données sont volatiles, souvent hétérogènes, et parfois incomplètes. Pour répondre efficacement aux besoins des différents acteurs du secteur aérien, il est indispensable de mettre en place un système capable d’ingérer, de traiter, de stocker et de visualiser ces données avec un haut niveau de fiabilité et de performance.

La problématique centrale de notre projet peut ainsi être formulée comme suit :  
**Comment concevoir un système robuste, évolutif et performant, capable de collecter, traiter, stocker et afficher en temps réel les positions des avions, tout en garantissant la fiabilité, la scalabilité, la sécurité des échanges, et une expérience utilisateur optimale ?**

Cette question générale se décline en plusieurs sous-problématiques techniques et fonctionnelles, que nous avons dû anticiper et résoudre tout au long de la conception du projet :

* **Gestion du flux massif de données en temps réel** : les positions des avions changent toutes les secondes, impliquant une collecte fréquente de données depuis l’API AviationStack. Cela crée une charge importante sur le système, qui doit pouvoir gérer ces volumes tout en maintenant des performances constantes.
* **Robustesse face aux erreurs** : les interruptions de service sont inévitables dans un système distribué (API injoignable, réseau instable, panne de la base de données ou du broker Kafka). Le système doit donc inclure des mécanismes de reprise automatique, de journalisation précise et d’alerte.
* **Performance et latence** : la visualisation des avions sur la carte doit être fluide et instantanée, même en cas de forte affluence (ex. : heures de pointe dans le trafic aérien). Il faut donc optimiser le traitement et la restitution des données via des index bien conçus, des requêtes ciblées, et des mises à jour fréquentes.
* **Fiabilité et qualité des données** : il est essentiel de filtrer les incohérences (doublons, données corrompues), de normaliser les champs (unités, format de date/heure), et de valider chaque donnée avant son stockage. La cohérence des données entre le producteur et le consommateur doit être assurée.
* **Anticipation de la position entre les mises à jour** : dans certains cas, l’API peut ne pas envoyer les données toutes les secondes. Pour maintenir une visualisation fluide, le système doit être capable d’estimer temporairement la position probable de l’avion sur la base de sa trajectoire et de sa vitesse.
* **Sécurité et confidentialité** : bien que les données aériennes ne soient pas sensibles au sens strict, la sécurité des échanges (notamment API et communication interne) doit être assurée pour prévenir les risques de détournement ou de surconsommation de ressources.
* **Expérience utilisateur** : enfin, l’interface de visualisation doit être intuitive, responsive, et agréable à utiliser sur tous les types d’écrans. Elle doit proposer des fonctionnalités de filtre, de zoom, d’affichage détaillé, et permettre une exploration dynamique et fluide.

Ces différents axes de réflexion ont guidé notre travail technique et notre choix d’outils. Ils constituent également un cadre d’analyse pertinent pour toute solution Big Data orientée temps réel. C’est à travers ces défis concrets que nous avons pu expérimenter la puissance des architectures distribuées modernes, et tester les limites des technologies NoSQL dans des cas d’usage réels et exigeants.

# **IV – Description du projet**

Le projet que nous avons réalisé vise à développer une solution complète et modulaire de suivi en temps réel des vols aériens. Il repose sur une architecture distribuée fondée sur les principes du traitement de flux (stream processing), de la persistance orientée documents, et de la visualisation interactive de données géospatiales. Cette solution s’articule autour de trois modules principaux, chacun remplissant une fonction clé dans le cycle de vie de la donnée : la collecte, le traitement/stockage, et la visualisation.

#### **1. Producteur de données de vol (flight\_producer.py)**

Le premier module agit comme une passerelle entre le monde extérieur (l’API AviationStack) et notre infrastructure. À intervalles réguliers, il interroge l’API pour obtenir des informations sur les vols actifs : numéro de vol, position géographique, altitude, vitesse, horaire de départ, destination, etc. Les données brutes obtenues sont ensuite transformées via plusieurs opérations :

* **Conversion d’unités** (mètres vers pieds, nœuds vers km/h) pour assurer la cohérence d’affichage.
* **Normalisation des champs** (standardisation des noms de colonnes, uniformisation des formats de date et heure).
* **Enrichissement contextuel** : ajout d’informations utiles telles que le nom des aéroports, la distance restante, le temps estimé d’arrivée, etc.

Après traitement, les données sont publiées sur un **topic Kafka** nommé flight\_data. Ce mécanisme asynchrone permet de découpler la collecte des données de leur traitement, assurant ainsi robustesse et scalabilité. Le producteur intègre également une gestion fine des erreurs (reconnexion, dépassement de quota API, clés expirées) ainsi qu’une configuration adaptable via des variables d’environnement.

#### **2. Serveur Flask (app.py)**

Le second module constitue le cœur du traitement applicatif. Il s’agit d’un serveur Python basé sur le micro-framework **Flask**, qui joue le rôle de **consommateur Kafka**. À l’aide d’un thread séparé, il écoute en continu le topic flight\_data et, pour chaque message reçu, effectue les opérations suivantes :

* Mise à jour de la base de données MongoDB, en appliquant un **upsert** (mise à jour ou insertion) basé sur un identifiant unique (flight\_id), garantissant que seule la position la plus récente de chaque avion est conservée.
* Indexation intelligente sur les champs flight\_id et processed\_at pour accélérer les requêtes ultérieures.
* Exposition d’une **API REST** (notamment à travers le point d’accès /api/flights) permettant à l’interface web d’interroger les données de manière performante.
* Gestion des erreurs, reconnexions automatiques et journalisation avancée pour le monitoring et la maintenance.

Ce module assure la cohérence des données dans la base, optimise le stockage et facilite la consommation par l’interface cliente.

#### **3. Interface utilisateur (index.html)**

Enfin, la dernière brique du système est l’interface graphique, accessible via un navigateur web. Développée avec **Leaflet.js** pour l’affichage de la carte interactive, et stylisée avec **Tailwind CSS**, elle offre une visualisation fluide et attrayante des avions en temps réel. Parmi ses fonctionnalités avancées, on trouve :

* Le **filtrage dynamique** des vols par identifiant ou compagnie.
* L’affichage contextuel des trajectoires, incluant le point de départ, la position actuelle, la destination.
* Un panneau d’informations enrichi présentant des détails comme le modèle d’avion, la vitesse, l’altitude, l’heure estimée d’arrivée.
* Une **prédiction de position** entre deux mises à jour, assurant une continuité visuelle même lorsque les données sont momentanément absentes.
* Une interface **responsive**, optimisée pour desktop, tablette et mobile, avec prise en charge des thèmes clair/sombre et animations douces.

Cette interface permet aux utilisateurs de visualiser le trafic aérien en temps réel de manière intuitive, tout en leur donnant un accès rapide à des informations techniques et contextuelles précises.

# **V – Fonctionnement du projet**

Le bon fonctionnement de notre solution repose sur une coordination fluide et efficace entre les différents modules du système. Chacun d’eux intervient dans une étape précise de la chaîne de traitement des données, depuis leur collecte auprès d’une source externe jusqu’à leur affichage final sur une interface web interactive. Ce fonctionnement repose sur une architecture modulaire, pensée pour garantir la résilience, l’extensibilité et les performances à chaque niveau.

#### **1. Collecte des données**

Tout commence par le **producteur de données de vol**, un script Python qui interroge régulièrement l’API AviationStack pour obtenir les dernières informations sur les vols actifs. Cette opération se fait selon une fréquence configurable (par exemple toutes les 10 secondes), afin de s’adapter à la charge et à la précision souhaitée. Le producteur récupère des données brutes qu’il transforme en appliquant des opérations d’enrichissement (ajout de distance restante, estimation de l’heure d’arrivée), de filtrage (élimination des données incomplètes), de normalisation (standardisation des unités et formats), et de gestion des erreurs (interruption de l’API, dépassement de quota, invalidation de clé). Une fois prêtes, les données sont publiées sur un **topic Kafka**, qui joue le rôle de file d’attente asynchrone, assurant la séparation entre la collecte et le traitement.

#### **2. Traitement et stockage**

Une fois les messages publiés sur Kafka, ils sont consommés en temps réel par le **serveur Flask**. Celui-ci lance un thread indépendant qui se connecte au broker Kafka et écoute continuellement les messages du topic flight\_data. À chaque réception, le serveur traite le message et met à jour la base de données **MongoDB**. Grâce à l’usage d’un mécanisme **upsert**, chaque enregistrement est soit mis à jour, soit inséré s’il n’existe pas encore. Cela permet d’économiser de l’espace disque et d’éviter les doublons. La base MongoDB est structurée pour répondre efficacement aux requêtes de lecture rapide, notamment grâce à des **index** définis sur les champs critiques (flight\_id, processed\_at). Le serveur assure également la gestion des exceptions, la reconnexion automatique aux services en cas d’interruption, et l’enregistrement des logs pour le débogage.

#### **3. Exposition des données**

Afin de permettre à l’interface web de récupérer les données, le serveur Flask expose une **API REST**. Cette API permet notamment d’accéder aux dernières positions des avions grâce à la route /api/flights. Les réponses sont au format JSON, optimisées en taille pour éviter les temps de chargement excessifs. Grâce à l’indexation de MongoDB, les temps de réponse sont très courts, même avec une base contenant des milliers de documents. L’API est conçue pour être facilement extensible, et pourrait à l’avenir inclure des endpoints supplémentaires (statistiques globales, historiques de trajectoires, alertes, etc.).

#### **4. Visualisation et interaction**

Enfin, les données sont exploitées dans l’**interface utilisateur**, qui présente une carte interactive construite à l’aide de **Leaflet.js**. Chaque avion est représenté par un marqueur dont la position est actualisée automatiquement à chaque rafraîchissement. L’utilisateur peut cliquer sur un marqueur pour ouvrir un **panneau d’informations** contenant des détails précis : compagnie, indicatif, vitesse, altitude, heure estimée d’arrivée, origine et destination. Un **algorithme de prédiction** basé sur la dernière vitesse connue permet d’interpoler la position de l’avion lorsque les données réelles ne sont pas encore disponibles. L’interface propose aussi des **filtres de recherche** dynamiques, un historique de vols récents, et une interface adaptative (responsive) qui s’adapte aux tailles d’écrans variées.

Grâce à l’interopérabilité entre ses composants, le projet garantit un **flux de données fluide**, une visualisation en temps réel et une capacité d’extension rapide. Il est ainsi capable de répondre à une montée en charge progressive sans altérer l’expérience utilisateur ni compromettre la stabilité du système.

# **VI – Réalisation**

La réalisation de ce projet a nécessité une planification rigoureuse, une répartition claire des tâches et une mise en œuvre progressive, fondée sur une approche incrémentale. Chaque étape de développement a été documentée, testée et validée afin de garantir la cohérence de l’ensemble du système. La démarche adoptée a suivi les grands principes de l’intégration continue, de la modularité, et de l’automatisation du déploiement.

#### **a) Mise en place de l’environnement**

Avant de débuter le développement fonctionnel, nous avons procédé à l’installation et à la configuration de tous les outils nécessaires. Cela comprenait :

* L’installation de **Kafka** en environnement local avec Zookeeper pour gérer les topics et assurer la persistance des messages.
* Le déploiement de **MongoDB** avec un schéma de base adapté à l’insertion fréquente de documents JSON.
* La configuration de **Flask**, le micro-framework Python choisi pour gérer les routes de l’API REST et l’intégration du consommateur Kafka.
* L’intégration de **Leaflet.js** pour la carte interactive et de **Tailwind CSS** pour le design, garantissant une interface à la fois esthétique et responsive.
* La création de **fichiers .env** pour stocker les variables sensibles (clés API, URL de MongoDB, ports de Kafka, etc.) et faciliter la configuration multi-environnement.
* L’écriture de **scripts de déploiement** Docker et de batch shell pour automatiser le lancement des services et assurer une reproductibilité parfaite.

#### **b) Développement du producteur (flight\_producer.py)**

Le développement du producteur a constitué une étape cruciale. Il s’agissait de mettre en place une connexion robuste avec l’API AviationStack, en prenant en compte :

* La **gestion des quotas** (limitation du nombre de requêtes par minute) grâce à un système de temporisation dynamique.
* La récupération de données précises pour chaque vol actif, avec des champs parfois manquants ou incohérents, nécessitant une phase de **nettoyage**.
* La **normalisation** des formats de données (unités, chaînes de caractères, dates) et l’ajout de **métadonnées** enrichies comme la distance restante ou l’estimation de l’heure d’arrivée.
* La publication sur le topic Kafka avec un système de **gestion des accusés de réception**, de tentatives de reconnexion et de journalisation des erreurs.
* Des **tests de charge** ont été réalisés pour simuler plusieurs dizaines de publications par minute et valider la tenue du système.

#### **c) Développement du serveur Flask (app.py)**

Le module Flask a été conçu pour fonctionner de manière autonome. Il assure les fonctions suivantes :

* Connexion à Kafka via un thread indépendant, évitant ainsi le blocage du serveur web.
* **Traitement parallèle** des messages Kafka avec insertion dans MongoDB via des opérations upsert, assurant la non-duplication et l’actualisation des données.
* Mise en place d’un **système d’indexation** dans MongoDB pour les champs critiques, améliorant la performance de l’API REST.
* Création de routes sécurisées et documentées, avec gestion des CORS, vérification de la validité des requêtes et mécanismes de cache pour améliorer la latence.
* Écriture de **tests unitaires et d’intégration** pour valider le bon fonctionnement du serveur dans différents scénarios (erreurs API, coupure de Kafka, surcharge).

#### **d) Développement de l’interface utilisateur**

Cette phase s’est concentrée sur l’ergonomie et la richesse de l’expérience utilisateur. Nous avons intégré :

* Une **carte interactive** avec Leaflet.js, permettant de visualiser les avions en temps réel avec des marqueurs dynamiques.
* Des **filtres** permettant à l’utilisateur de rechercher un vol par identifiant, compagnie ou destination.
* Un **panneau d’informations** contextuel pour chaque vol, affichant les données essentielles : altitude, vitesse, modèle d’avion, aéroport d’origine et d’arrivée, etc.
* Un **système de prédiction** de la position entre deux mises à jour, calculée à partir de la vitesse et de la direction du vol.
* Une personnalisation visuelle avec Tailwind CSS, incluant les thèmes clair/sombre, l’adaptation mobile, la prise en charge de différents navigateurs, et une gestion fluide des animations.

#### **e) Tests et validation**

Enfin, une batterie de tests a été menée pour valider la robustesse globale du projet :

* **Tests unitaires** pour chaque module (producteur, serveur Flask, interface).
* **Tests d’intégration** pour vérifier la communication entre Kafka, MongoDB et Flask.
* **Simulations de pannes** (déconnexion API, saturation Kafka, crash de la base) pour observer les comportements du système et ajuster les mécanismes de reprise.
* **Mesures de performance** pour évaluer la latence, la consommation mémoire, le débit maximal supporté.
* **Recueil de retours utilisateurs** via un questionnaire test et ajustements de l’interface pour améliorer la lisibilité, la réactivité et l’intuitivité du site.

# **VII – Conclusion**

La réalisation de ce projet a été l’occasion d’explorer concrètement les défis techniques, architecturaux et fonctionnels que posent les systèmes Big Data temps réel dans un contexte distribué. En nous appuyant sur des technologies modernes et puissantes – telles que Kafka pour la gestion des flux de données, MongoDB pour le stockage NoSQL, Flask pour l’interfaçage applicatif, Leaflet.js pour la visualisation cartographique et Tailwind CSS pour la conception d’une interface élégante – nous avons conçu une solution cohérente, modulaire et scalable, capable de répondre à des besoins réels dans le domaine du suivi aérien.

Ce projet ne s’est pas limité à une simple application technique ; il a mis en œuvre une véritable **démarche d’ingénierie logicielle**, de la définition de la problématique à la mise en production d’une solution fonctionnelle. Nous avons dû faire preuve de rigueur, de planification et de capacité d’adaptation face aux imprévus (problèmes d’API, erreurs de compatibilité, saturation réseau, etc.). Il a également été nécessaire de travailler avec des outils en ligne de commande, de gérer les dépendances interservices, de structurer une base de données efficace et de garantir des performances constantes, même sous charge.

Ce travail nous a permis de maîtriser les **principes fondamentaux des architectures orientées événements**, de mieux comprendre le paradigme du traitement en flux (stream processing), et de mettre en pratique des modèles de stockage flexibles adaptés aux environnements Big Data. L’approche distribuée adoptée (producteur Kafka – consommateur Flask – base MongoDB – interface web) nous a offert une grande flexibilité dans la gestion du cycle de vie des données, tout en assurant une **découplage fort entre les composants**, ce qui facilite la maintenance et les futures évolutions.

Du point de vue de l’utilisateur final, le système offre une expérience riche et interactive, avec une visualisation en temps réel du trafic aérien, des filtres dynamiques, un affichage précis des trajectoires, et une personnalisation avancée de l’interface. Cette dimension front-end, souvent sous-estimée, a joué un rôle fondamental dans la réussite du projet, car elle permet de **valoriser visuellement les données collectées**, de manière pédagogique et engageante.

Les perspectives d’amélioration de ce projet sont nombreuses. Parmi les axes envisagés, nous pourrions citer :

* **L’intégration de sources de données supplémentaires**, comme les données radar ou satellite, pour améliorer la précision et la couverture géographique.
* **Le développement d’outils analytiques avancés**, permettant par exemple de prédire les retards, d’analyser le trafic global par région, ou de détecter des comportements anormaux (détours, accélérations soudaines).
* **La mise à disposition d’une API publique** destinée à d’autres développeurs ou applications, dans une logique d’ouverture et d’écosystème numérique.
* **La déclinaison mobile native** de l’interface web, via une application Android ou iOS, offrant une expérience utilisateur encore plus fluide.
* **La création de tableaux de bord décisionnels** (dashboards) pour les professionnels du secteur aérien : gestionnaires de trafic, compagnies aériennes, analystes, etc.

En conclusion, ce projet représente bien plus qu’un simple exercice académique. Il s’inscrit dans une logique de veille technologique et d’innovation, en apportant des solutions concrètes à des problématiques actuelles de gestion de données en temps réel. Il constitue une base solide et évolutive pour toute application nécessitant une collecte, un traitement et une restitution rapide de données géolocalisées, et peut facilement être adapté à d’autres domaines comme la logistique, le transport maritime, la gestion de flottes, ou encore la surveillance environnementale.