PROJET FLY ME

-

Méthodologie d'évaluation de la performance hors ligne, de pilotage en production et de mise à jour en production du modèle du chatbot

Introduction

Fly Me est une agence qui propose des voyages clé en main pour les particuliers ou les professionnels.

Fly Me a lancé un projet ambitieux de développement d'un chatbot pour aider les utilisateurs à choisir une offre de voyage.

Pour cela, l'objectif du projet est de construire un MVP qui aidera les employés de Fly Me à réserver facilement un billet d'avion pour leurs vacances.

Ce document présente les éléments mis en place afin d'évaluer et suivre la performance du chatbot mis en place dans le cadre de ce projet.

Il est composé de 3 parties :

1 - Évaluation de la performance du modèle sous-jacent hors ligne

2 - Méthodologie de pilotage de la performance du modèle en production

3 - Méthodologie de mise à jour du modèle en production

1 - Évaluation de la performance du modèle sous-jacent hors ligne

Nous avons utilisé le service **Microsoft Azure LUIS** pour la modélisation. Le service LUIS permet d'une part de déterminer quelle est **l'intention de l'utilisateur** en fonction de sa demande et d'autre part d'identifier **les informations importantes** permettant de répondre à sa demande (*entities*).

Afin d'évaluer la performance du modèle, nous avons développé **des fonctions dédiées** dans le code Python permettant d'évaluer le modèle LUIS mis en place pour le chatbot.

Pour cela le jeu de données initial a été séparé en deux jeux de données :

- Un jeu de données pour **l'entrainement du modèle** (train)
- Un jeu de données pour **l'évaluation du modèle** (test)

Description des critères d'évaluation de la performance retenus :

Nous avons retenu les critères suivants pour l'évaluation de la performance du modèle :

- Pourcentage d'**intentions correctement détectées** par le modèle pour l'intention **OrderTravel** *(réservation de vol)*
- Pourcentage d'**entités correctement détectées** pour l'intention **OrderTravel**
- Pourcentage d'**intentions correctement détectées** pour l'intention **Greetings** *(salutations)*

Nous obtenons les résultats suivants :

- Pourcentage d'intentions correctement détectées pour l'intention OrderTravel = **99%**
- Pourcentage d'entités correctement détectées pour l'intention OrderTravel = **84%**
- Pourcentage d'intentions correctement détectées pour l'intention Greetings = **85%**

➔ **Ce sont de bons résultats qui montrent la qualité du modèle mis en place ainsi que sa bonne capacité de généralisation**

2 - Méthodologie de pilotage de la performance du modèle en production

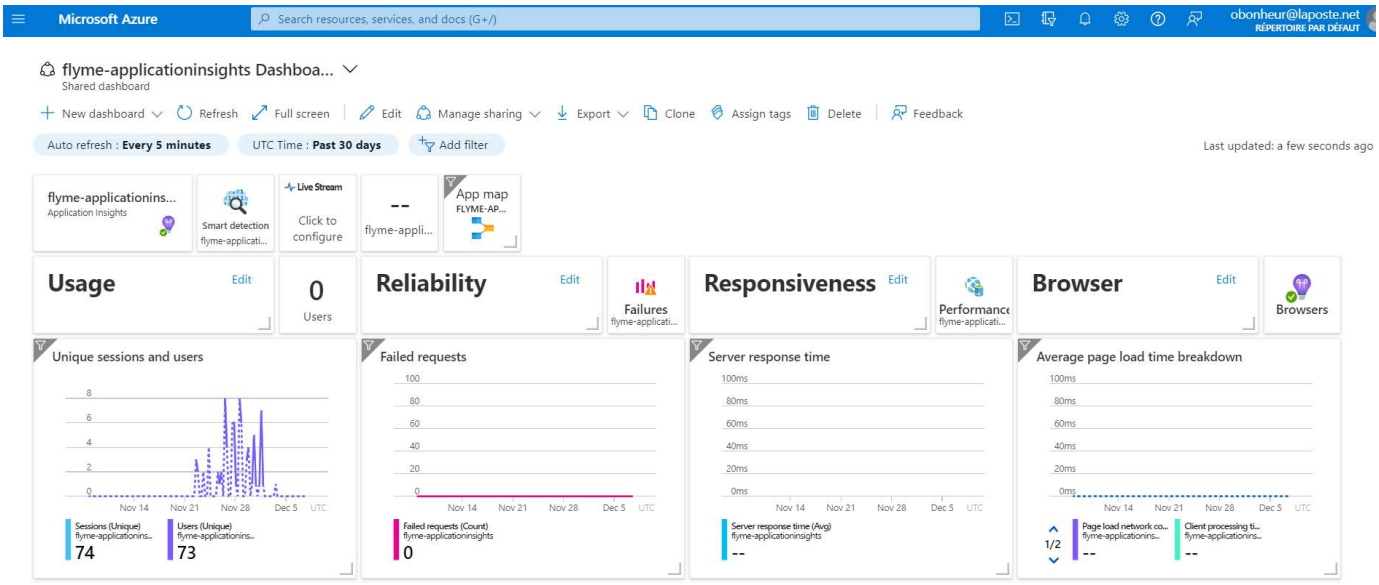
Pour le pilotage de la performance du modèle en production, nous avons utilisé le service dédié de Microsoft Azure : **Applications Insights.**

La mise en place se fait en 2 étapes :

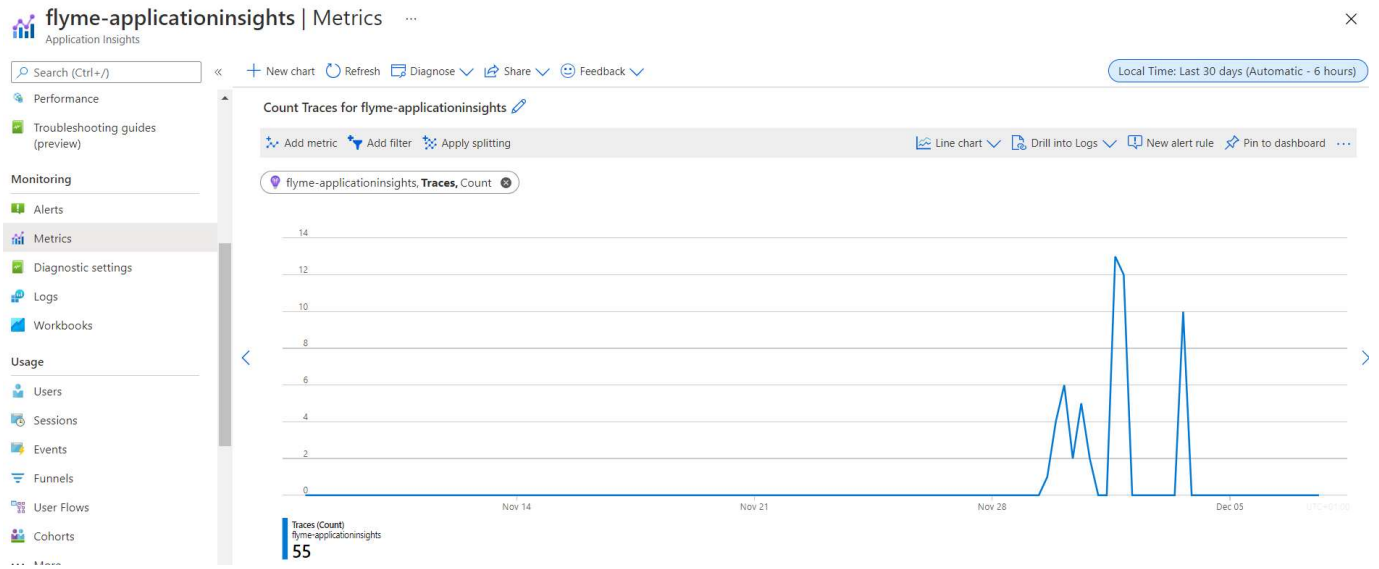
- On crée le service Applications Insights dans Azure
- On ajoute dans le code Python les éléments nécessaires permettant d'envoyer les informations au service Applications Insights

Applications Insights permet d'avoir accès aux services de pilotage de la performance suivants :

- **Dashboard** : tableau de suivi récapitulatif personnalisable



- **Metrics** : métriques de suivi de la performance



- **Usage** : métriques de suivi de l'utilisation du chatbot

- **Transaction Search** : permet de suivre les événements majeurs qui se sont produits (Exceptions, Traces et échanges entre l'utilisateur et le chatbot notamment les labellisations effectuées par LUIS)

- **Availability** : permet de suivre la disponibilité de l'application

- **Alertes** : permet la mise en place d'alertes en cas de dégradation de la performance

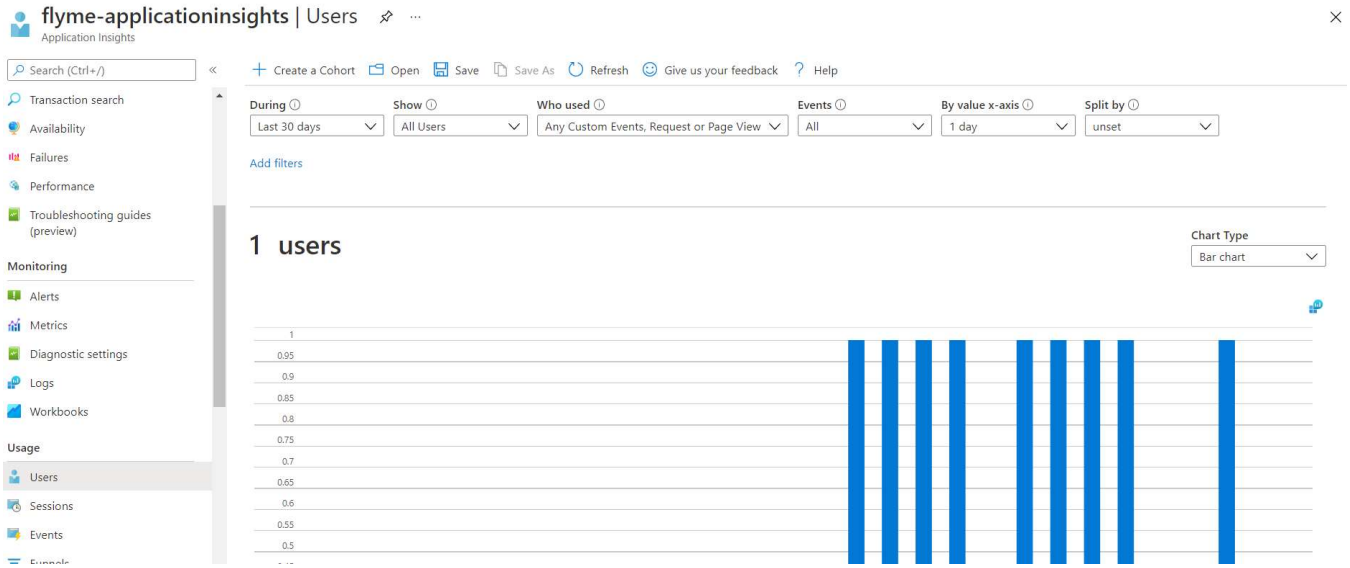
Description des critères de pilotage de la performance retenus :

Nous avons retenu les critères suivants pour le pilotage de la performance du modèle en production :

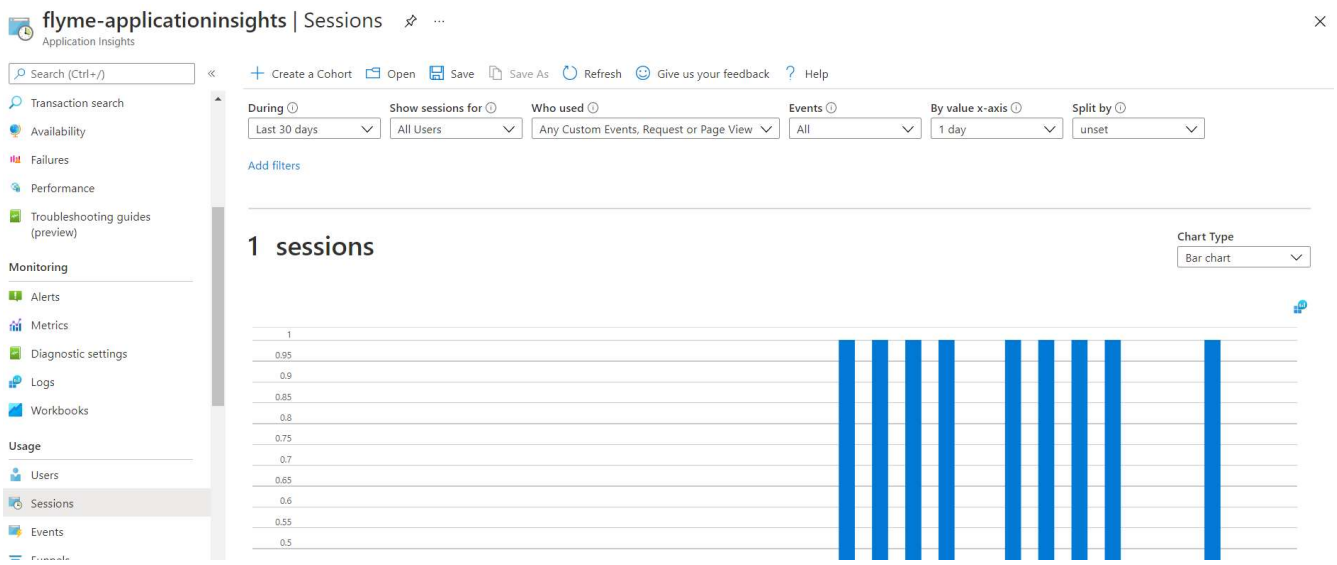
- Nombre d'utilisateurs sur l'application
- Nombre de sessions
- Disponibilité : mise en place d'un test de disponibilité
- Temps de réponse moyen aux requêtes
- Dégradation de la performance : alerte si le chatbot fait 3 erreurs ou plus sur une période de 1 heure

Le nombre d'utilisateurs et le nombre de sessions sont calculés en continu par Azure et visibles dans l'onglet **Usage.**

Suivi du nombre d'utilisateurs :

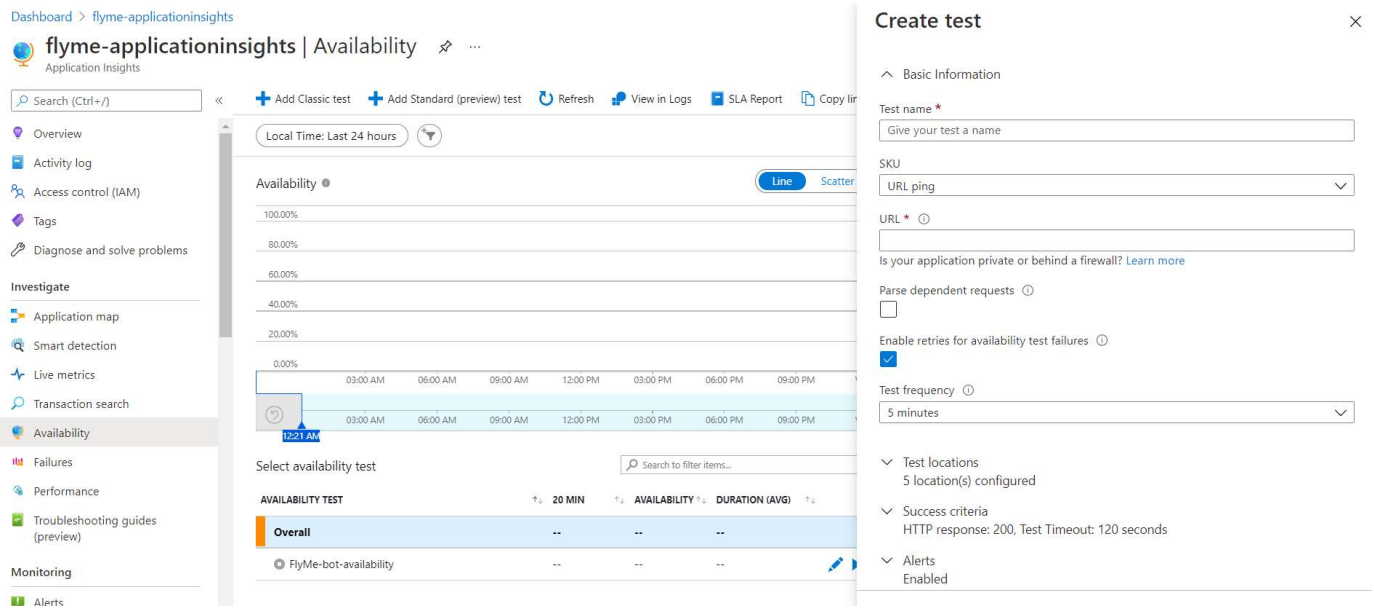


Suivi du nombre de sessions :

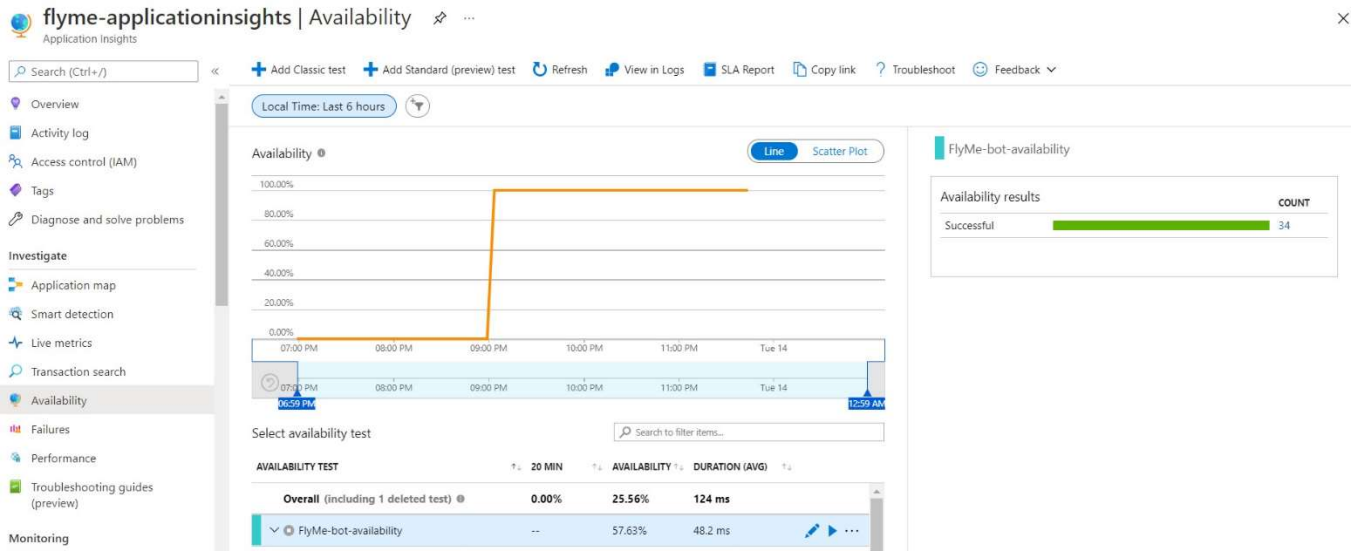


Pour le suivi de la **disponibilité**, nous avons mis en place d'un test de disponibilité qui basé sur la réponse de l'application à un ping.

<u>Mise en place du test de disponibilité :</u>

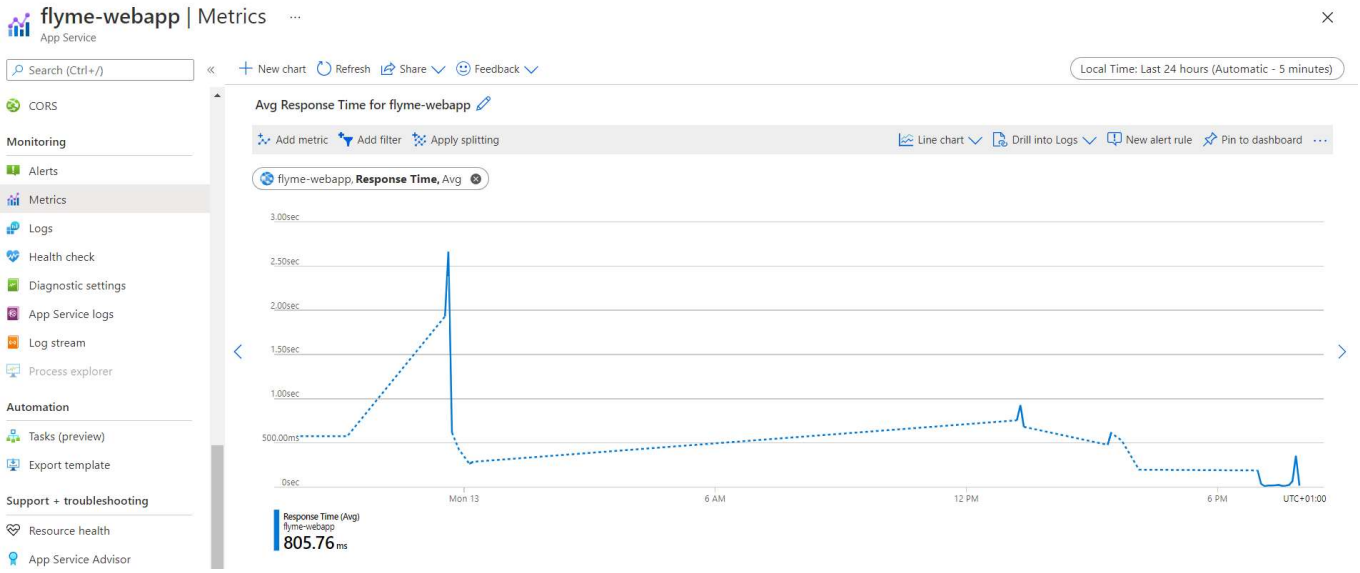


<u>Résultat du test de disponibilité :</u>



Le **temps de réponse** moyen aux requêtes est calculé par Azure et visible dans l'onglet **Metrics.**

Temps de réponse moyen aux requêtes :



Pour le critère de **dégradation de la performance** nous avons mis en place une alerte envoyant automatiquement un mail d'alerte si le chatbot fait **trois erreurs ou plus sur une période d'une heure**.

La mise en place de cette alerte a été réalisée en deux étapes:

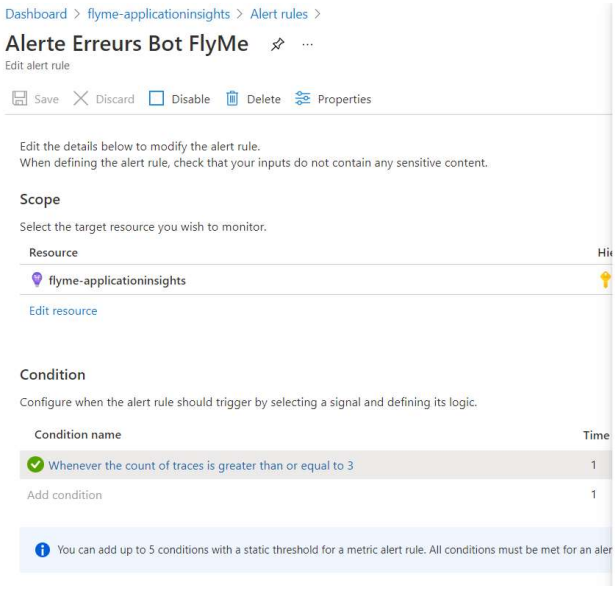
- Dans **le code du chatbot** : utilisation de la méthode ***track_trace()*** permettant d'envoyer à Applications Insights le résultat des échanges avec l'utilisateur :
    - Si l'utilisateur a validé la bonne compréhension de sa demande par le chatbot : envoie d'une information de type **INFO** indiquant que **l'utilisateur a validé la transaction**
    - Si l'utilisateur n'a pas validé la bonne compréhension de sa demande par le chatbot : envoie d'une information de type **ERROR** indiquant que **l'utilisateur n'a pas validé la transaction**
    - Les échanges entre l'utilisateur et le chatbot ainsi que la labellisation des messages par LUIS sont également enregistrés

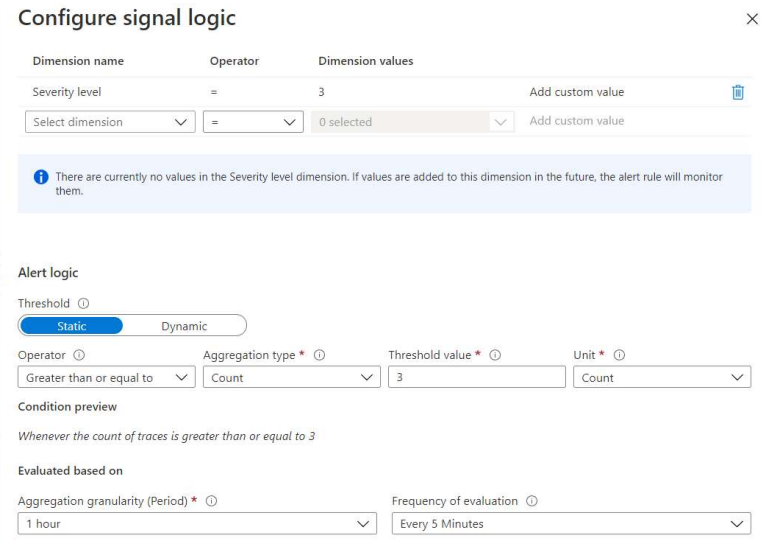
```
if step_context.result:
    self.telemetry_client.track_trace("Transaction confirmed by the user : YES", booking_info, "INFO")
    return await step_context.end_dialog(booking_details)

else:
    self.telemetry_client.track_trace("Transaction confirmed by the user : NO", booking_info, "ERROR")
    return await step_context.end_dialog()
```

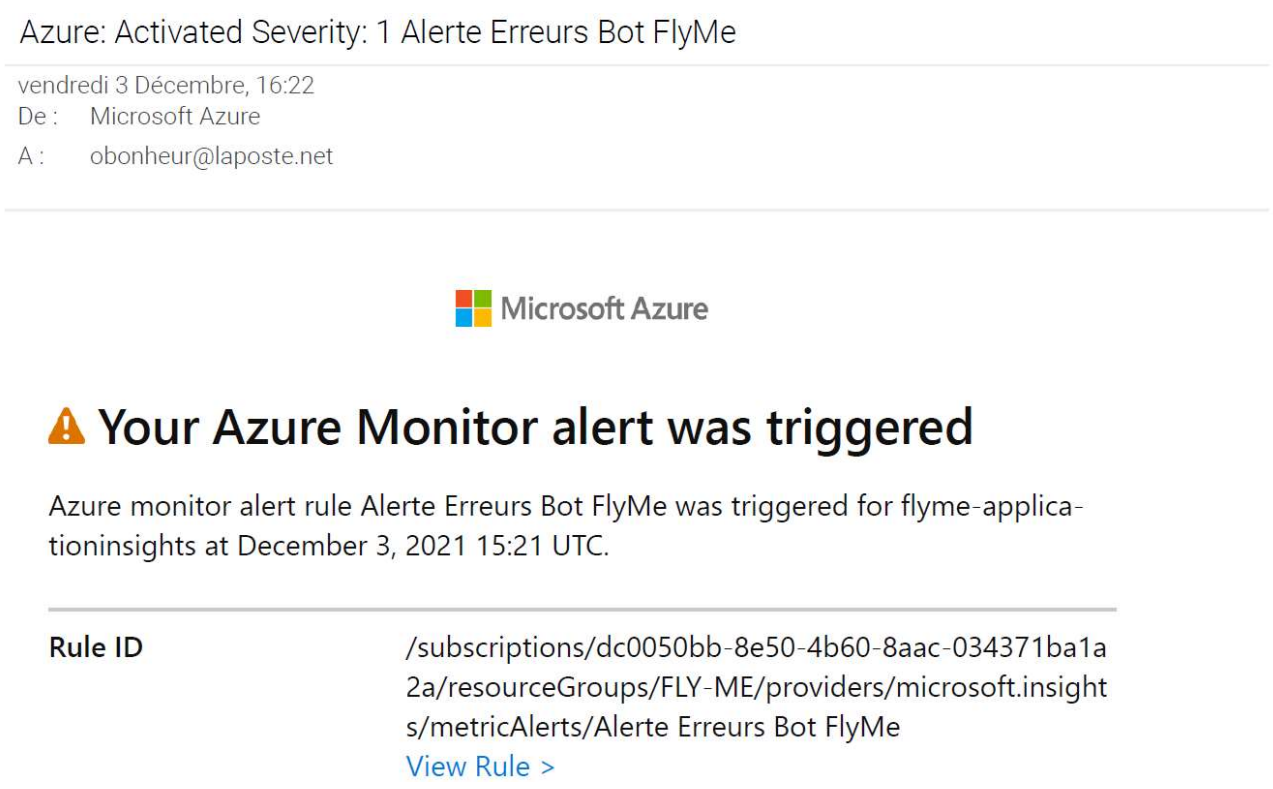
- Dans **Applications Insights** : mise en place d'une règle avec les caractéristiques suivantes :
    - Calcul du nombre de fois où le chatbot a fait une erreur (c'est-à-dire que l'utilisateur n'a pas validé la transaction)
    - Si le nombre d'erreurs est supérieur ou égal à 3 sur une période de 1 heure, alors il y a un envoi automatique d'un message par email décrivant l'erreur

Alerte mise en place dans Azure :

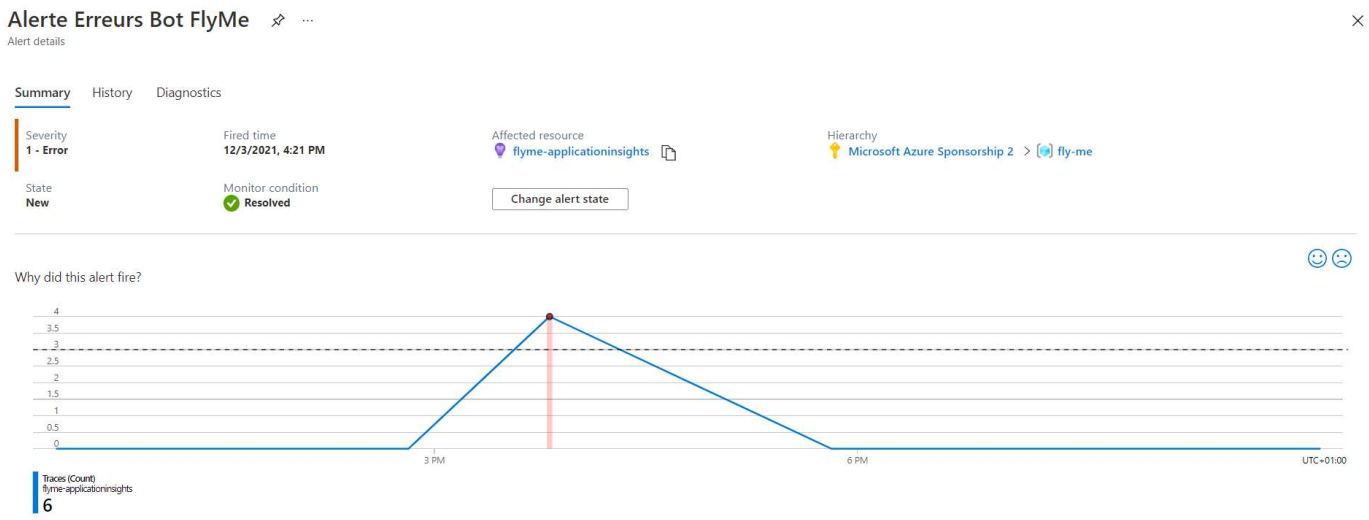




Mail reçu en cas d'alerte :



Alerte visible dans Azure :



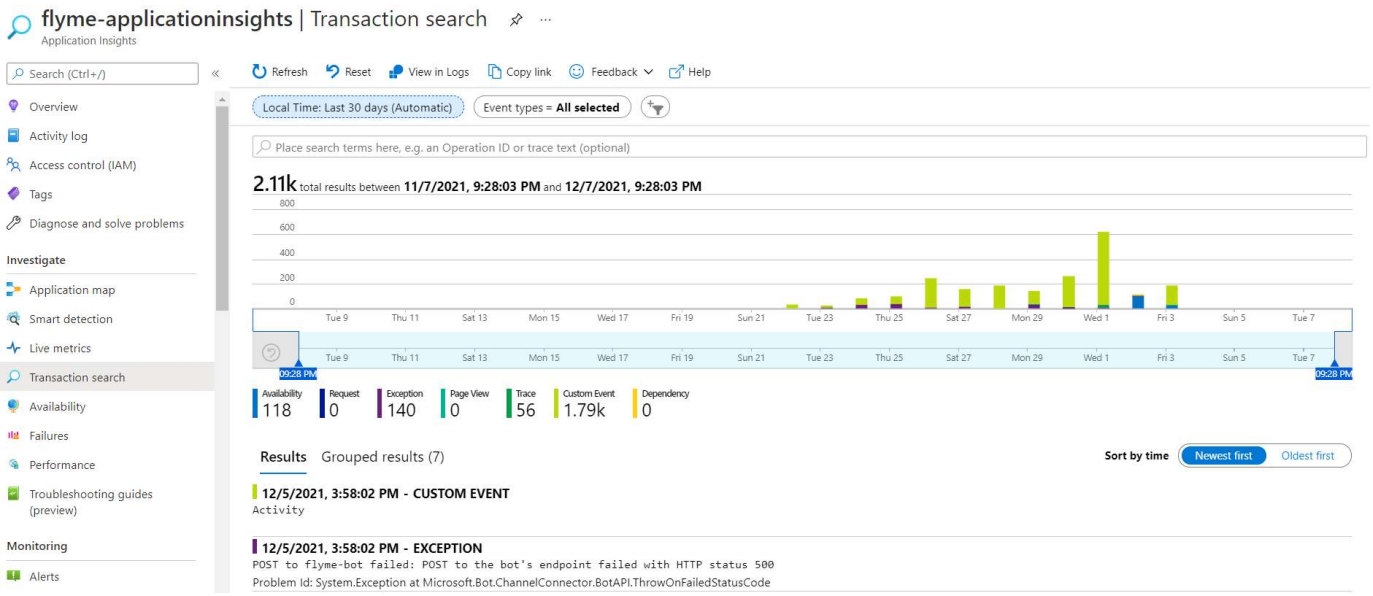
En plus de ces critères, le service **Transaction Search** permet de suivre les principaux événements liés à la vie du chatbot :

- **Exceptions :**
  - Erreurs au niveau du service (exemple : erreur HTTP 500)

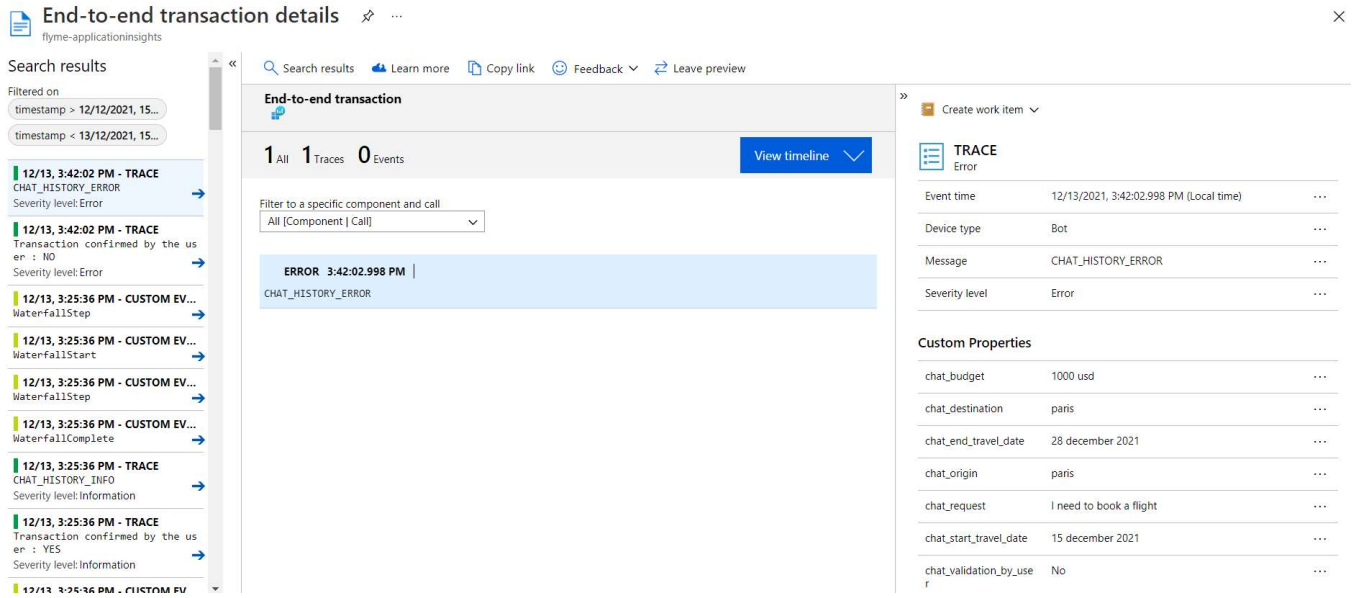
- **Traces :**
  - Evénements envoyés par le programme Python grâce à l'utilisation de la fonction ***track_trace()***
  - On trouve notamment les **échanges** entre l'utilisateur et le chatbot, ainsi que les **résultats labellisés** par LUIS

- **Custom Events :**
  - Evénements envoyés automatiquement par le programme Python grâce à la classe ***BotTelemetryClient***, faisant partie du Framework SDK Bot de Microsoft

Tableau de suivi Transaction Search :



Exemple de suivi d'un échange en erreur (transaction non validée par l'utilisateur) :



Exemple de suivi des dialogues labellisés par LUIS :

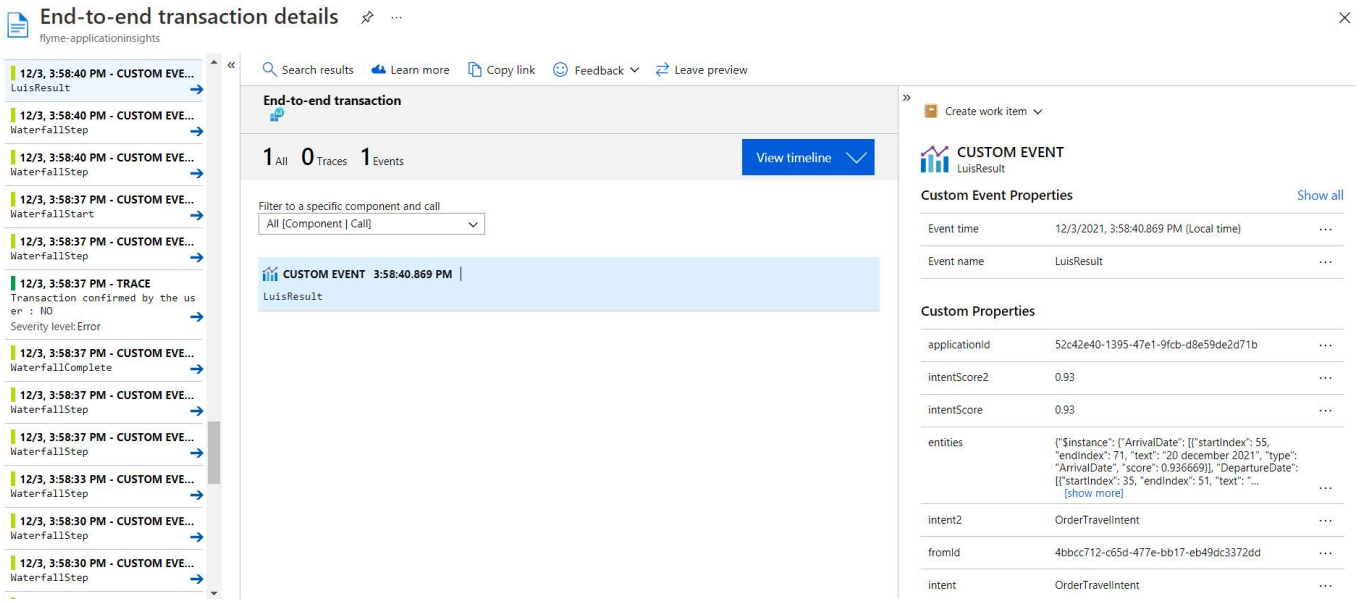
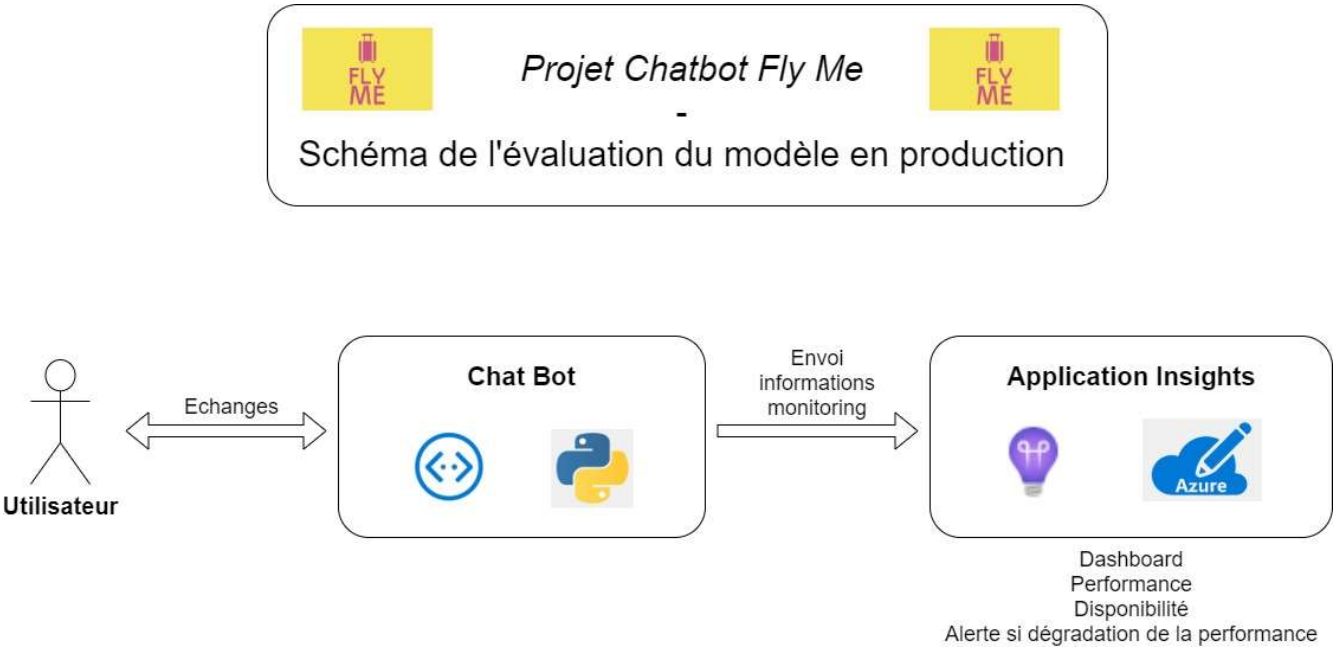


Schéma récapitulatif de l'évaluation du modèle en production :



3 - Méthodologie de mise à jour du modèle en production

La méthodologie proposée pour la mise à jour du modèle en production est la suivante :

1. Les données échangées entre l'utilisateur et le chatbot en production sont enregistrées, dans une base de données par exemple

2. En cas d'un nombre d'erreurs trop importantes par le chatbot, le modèle est mis à jour :
    a. Pour cela, on récupère tout d'abord les échanges préalablement sauvegardés
    b. On procède à la labellisation de ces échanges (intention et entities)
    c. Puis on ré-entraine le modèle LUIS avec ces échanges en lançant le script dédié à cette tache (*FLYME_SCRIPT-LUIS*)

3. On déploie le nouveau modèle

4. Et enfin, on observe le nouveau modèle en production :
    a. S'il fait moins d'erreurs que le précédent on le garde
    b. Sinon, s'il fait plus d'erreurs que le précédent, on revient au précédent modèle

Au niveau de la fréquence de mise à jour, nous préconisons de mettre en place cette méthodologie de façon continue et de la faire se déclencher automatiquement lorsque l'alerte mise en place précédemment s'active, c'est-à-dire quand le chatbot fait trois erreurs ou plus sur une période d'une heure.

➔ **Cette méthodologie permet d'essayer d'améliorer le modèle de façon continue avec de nouveaux échanges (c'est-à-dire de nouvelles données) et de changer le modèle uniquement si le nouveau est meilleur que le précédent.**