

# 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'entraînement 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

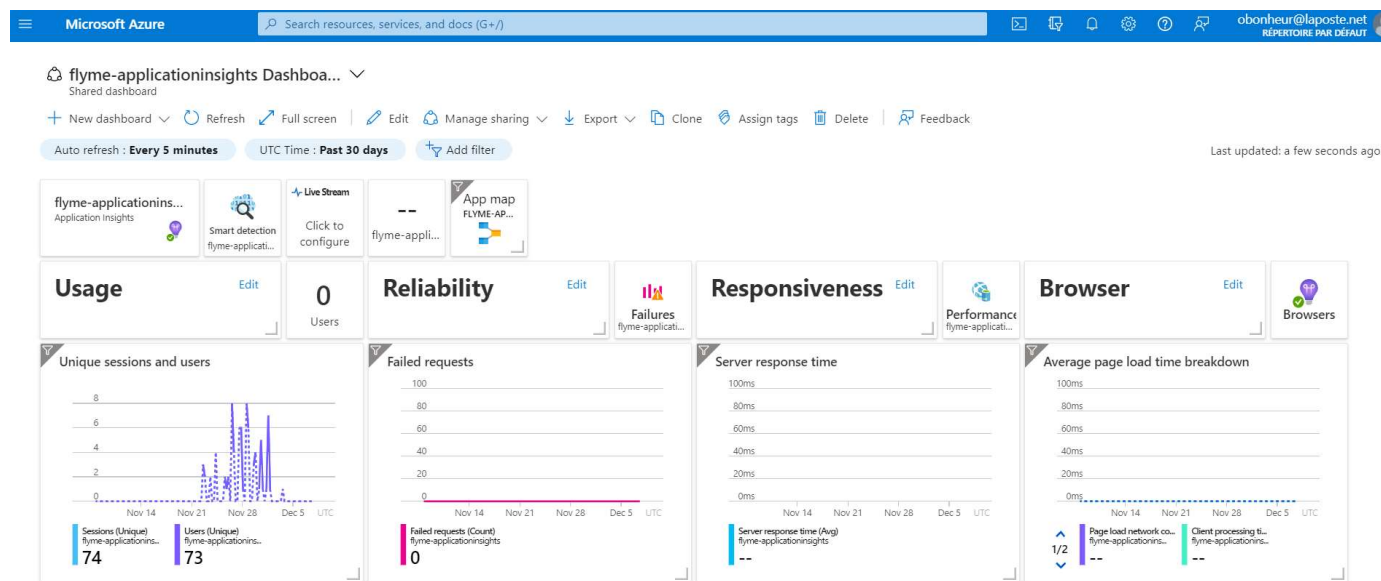
Pour le pilotage de la performance du modèle en production, nous avons utilisé le service dédié de Microsoft Azure : **Applications Insights**.

La mise en place se fait en 2 étapes :

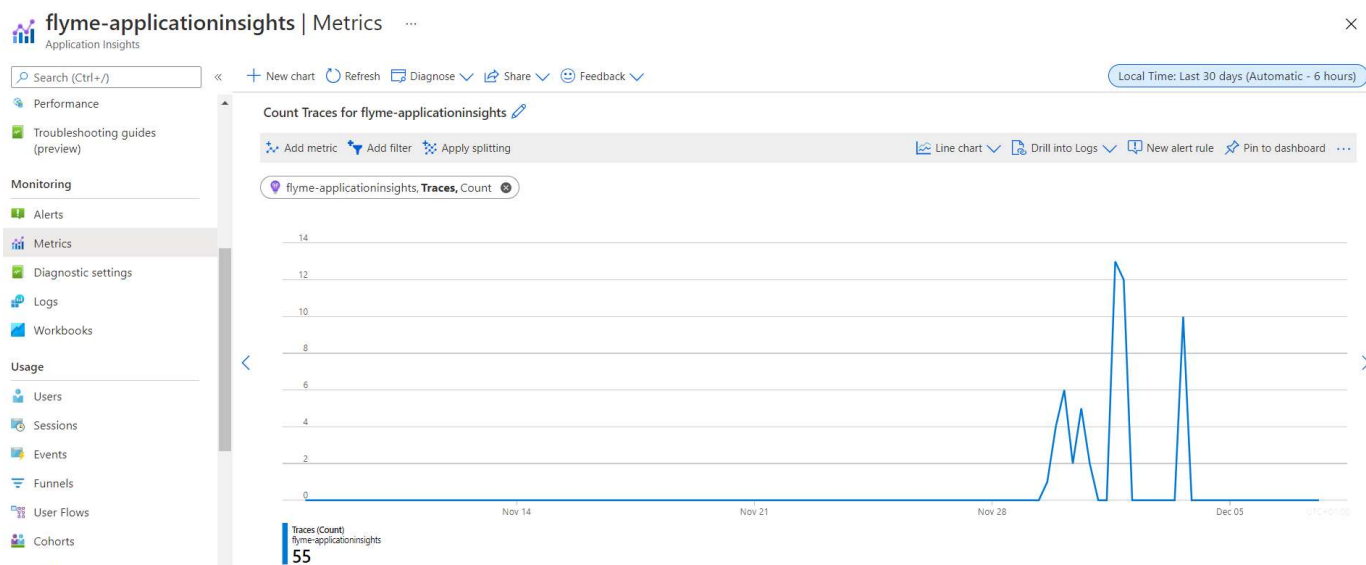
- On crée le service Applications Insights dans Azure
- On ajoute dans le code Python les éléments nécessaires permettant d'envoyer les informations au service Applications Insights

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

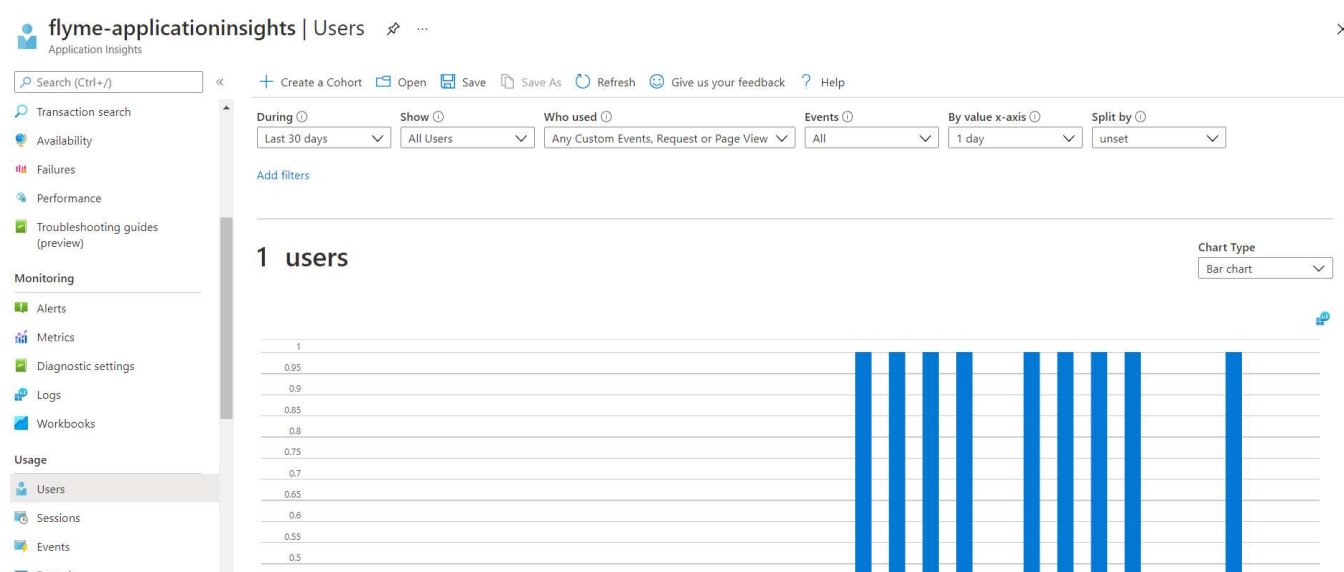
## Description des critères de pilotage de la performance retenus :

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

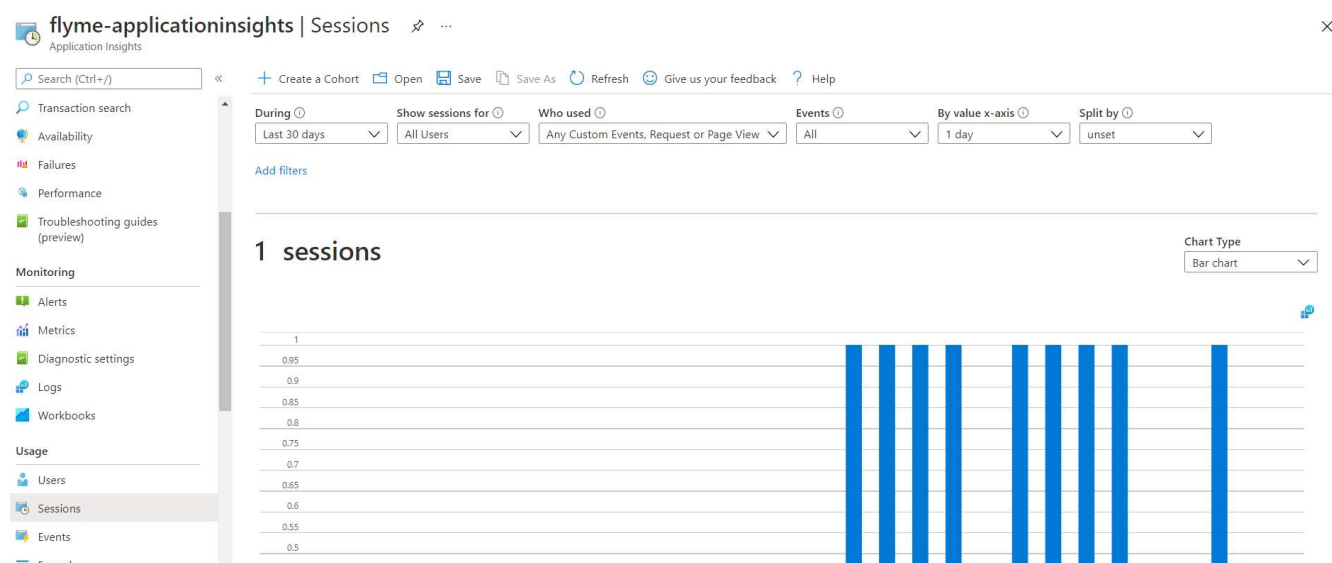
- Nombre d'utilisateurs sur l'application
- Nombre de sessions
- Disponibilité : mise en place d'un test de disponibilité
- Temps de réponse moyen aux requêtes
- Dégradation de la performance : alerte si le chatbot fait 3 erreurs ou plus sur une période de 1 heure

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.

### Mise en place du test de disponibilité :

The screenshot shows the 'Create test' dialog in the flyme-applicationinsights interface. The dialog is titled 'Create test' and has a close button (X) in the top right corner. It contains a 'Basic Information' section with the following fields:

- Test name \***: A text input field with the placeholder 'Give your test a name'.
- SKU**: A dropdown menu with 'URL ping' selected.
- URL \***: A text input field with a help icon (i).
- Is your application private or behind a firewall?**: A checkbox with the text 'Learn more' next to it.
- Parse dependent requests**: A checkbox.
- Enable retries for availability test failures**: A checked checkbox.
- Test frequency**: A dropdown menu with '5 minutes' selected.

Below the 'Basic Information' section, there are three sections with checkmarks:

- Test locations**: 5 location(s) configured
- Success criteria**: HTTP response: 200, Test Timeout: 120 seconds
- Alerts**: Enabled

The background shows the 'Availability' dashboard with a line chart showing availability over time. The chart has a y-axis from 0.00% to 100.00% and an x-axis with time slots from 03:00 AM to 09:00 PM. A blue line shows the availability, which is at 0.00% until 03:00 AM and then jumps to 100.00%.

### Résultat du test de disponibilité :

The screenshot shows the 'Availability' dashboard in the flyme-applicationinsights interface. The dashboard has a search bar (Ctrl+/) and a 'Local Time: Last 6 hours' filter. The main chart shows availability over time, with a y-axis from 0.00% to 100.00% and an x-axis with time slots from 07:00 PM to 11:00 PM. A blue line shows the availability, which is at 0.00% until 09:00 PM and then jumps to 100.00%.

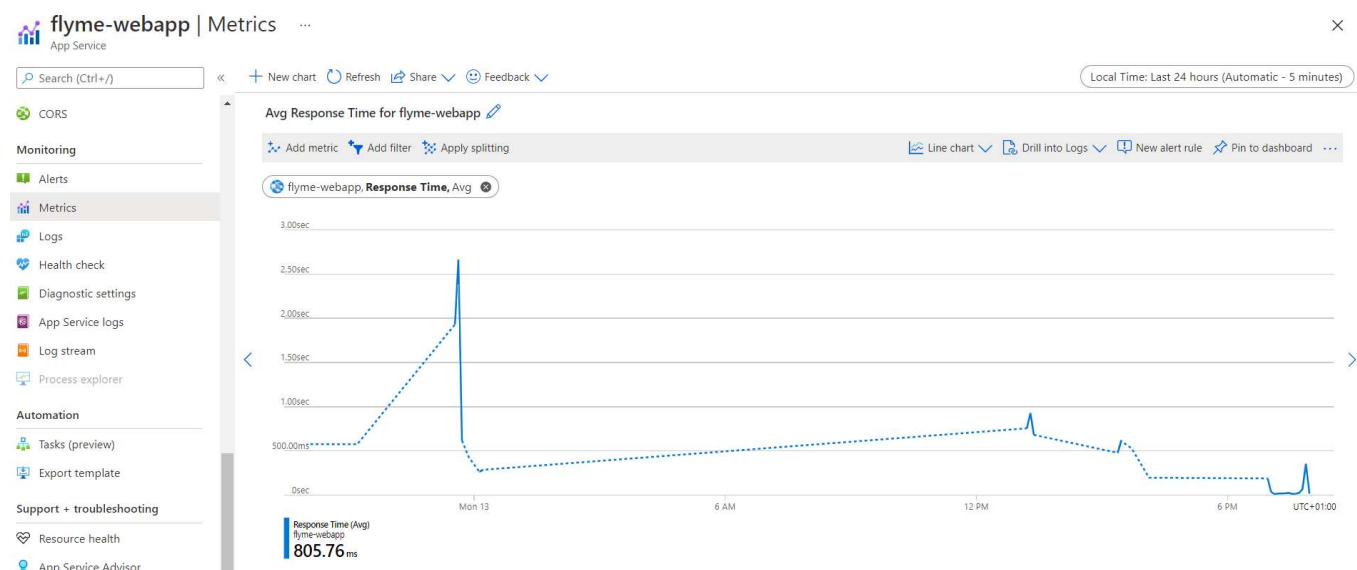
Below the chart, there is a 'Select availability test' section with a search bar. Below that, there is a table with the following data:

AVAILABILITY TEST	20 MIN	AVAILABILITY	DURATION (AVG)
Overall (including 1 deleted test)	0.00%	25.56%	124 ms
FlyMe-bot-availability	--	57.63%	48.2 ms

On the right side of the dashboard, there is a 'FlyMe-bot-availability' section with a bar chart showing 'Availability results' and a 'COUNT' of 34.

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 : envoi 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 : envoi 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 :

Dashboard > flyme-applicationinsights > Alert rules >

### Alerte Erreurs Bot FlyMe

Edit alert rule

Save Discard Disable Delete Properties

Edit the details below to modify the alert rule.  
When defining the alert rule, check that your inputs do not contain any sensitive content.

#### Scope

Select the target resource you wish to monitor.

Resource

flyme-applicationinsights

Edit resource

#### Condition

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Condition name	Time
✓ Whenever the count of traces is greater than or equal to 3	1
Add condition	1

You can add up to 5 conditions with a static threshold for a metric alert rule. All conditions must be met for an alert.

### Configure signal logic

Dimension name	Operator	Dimension values
Severity level	=	3 Add custom value
Select dimension	=	0 selected Add custom value

There are currently no values in the Severity level dimension. If values are added to this dimension in the future, the alert rule will monitor them.

#### Alert logic

Threshold

Static Dynamic

Operator: Greater than or equal to Aggregation type: Count Threshold value: 3 Unit: Count

Condition preview: Whenever the count of traces is greater than or equal to 3

Evaluated based on

Aggregation granularity (Period): 1 hour Frequency of evaluation: Every 5 Minutes

## Mail reçu en cas d'alerte :

Azure: Activated Severity: 1 Alerte Erreurs Bot FlyMe

vendredi 3 Décembre, 16:22

De : Microsoft Azure

A : obonheur@laposte.net



## ⚠ Your Azure Monitor alert was triggered

Azure monitor alert rule Alerte Erreurs Bot FlyMe was triggered for flyme-applicationinsights at December 3, 2021 15:21 UTC.

Rule ID /subscriptions/dc0050bb-8e50-4b60-8aac-034371ba1a2a/resourceGroups/FLY-ME/providers/microsoft.insights/metricAlerts/Alerte Erreurs Bot FlyMe  
[View Rule >](#)

## Alerte visible dans Azure :

### Alerte Erreurs Bot FlyMe ✨ ...

Alert details

Summary History Diagnostics

Severity  
1 - Error

Fired time  
12/3/2021, 4:21 PM

Affected resource  
flyme-applicationinsights

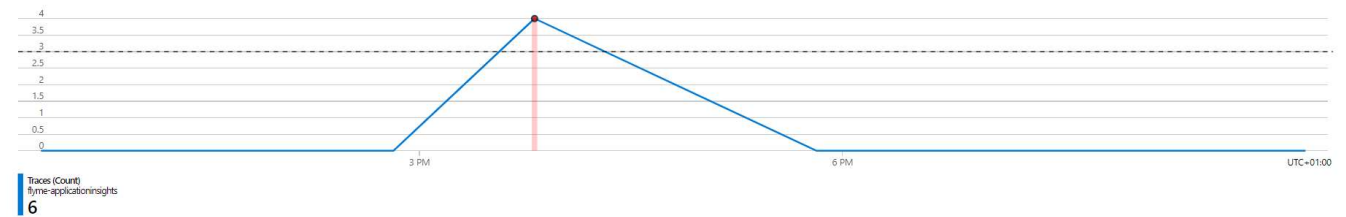
Hierarchy  
Microsoft Azure Sponsorship 2 > fly-me

State  
New

Monitor condition  
Resolved

Change alert state

Why did this alert fire?

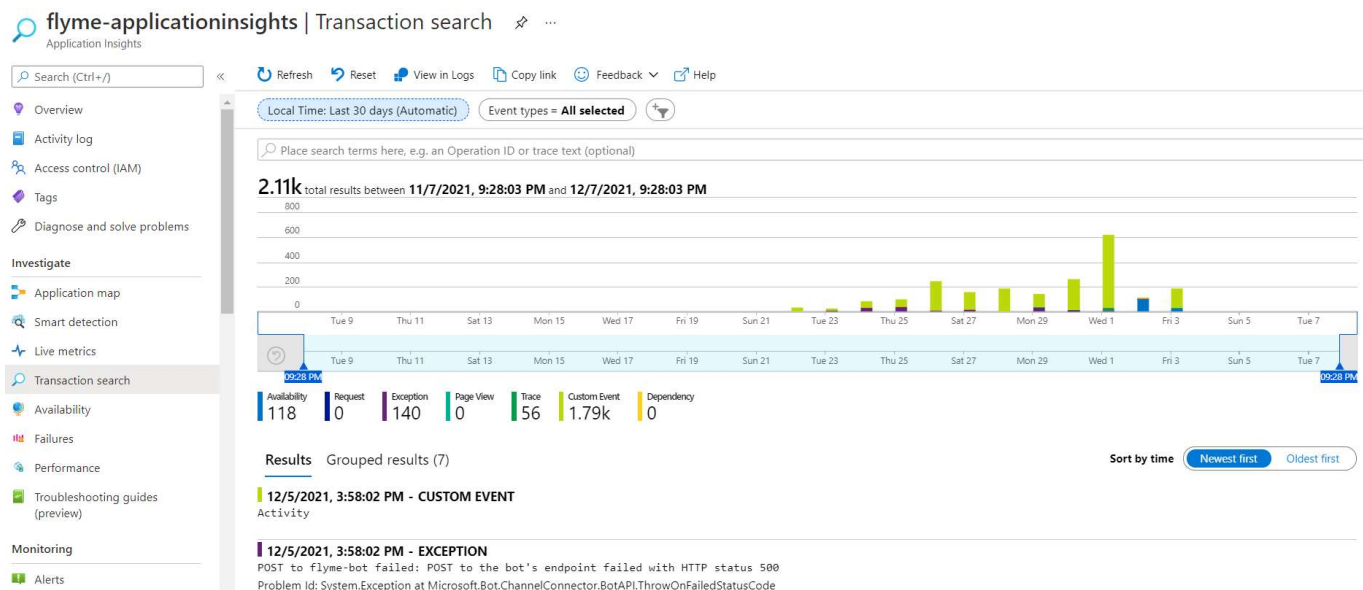


En plus de ces critères, le service **Transaction Search** permet de suivre les principaux événements liés à la vie du chatbot :

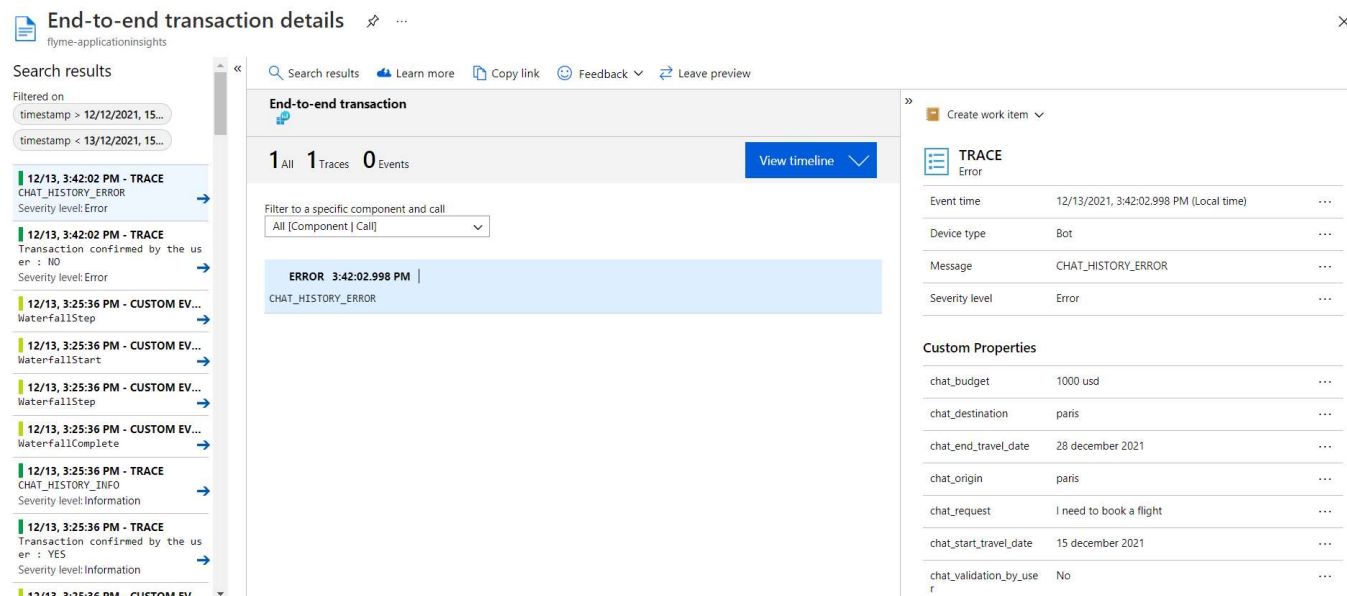
- **Exceptions :**
  - Erreurs au niveau du service (exemple : erreur HTTP 500)
- **Traces :**
  - Événements envoyés par le programme Python grâce à l'utilisation de la fonction ***track\_trace()***
  - On trouve notamment les **échanges** entre l'utilisateur et le chatbot, ainsi que les **résultats** labellisés par LUIS
- **Custom Events :**
  - Événements 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 :

flyme-applicationinsights

### End-to-end transaction details

Search results Learn more Copy link Feedback Leave preview

**End-to-end transaction**

1 All 0 Traces 1 Events [View timeline](#)

Filter to a specific component and call  
All [Component | Call]

**CUSTOM EVENT 3:58:40.869 PM**  
LuisResult

**CUSTOM EVENT**  
LuisResult

Create work item

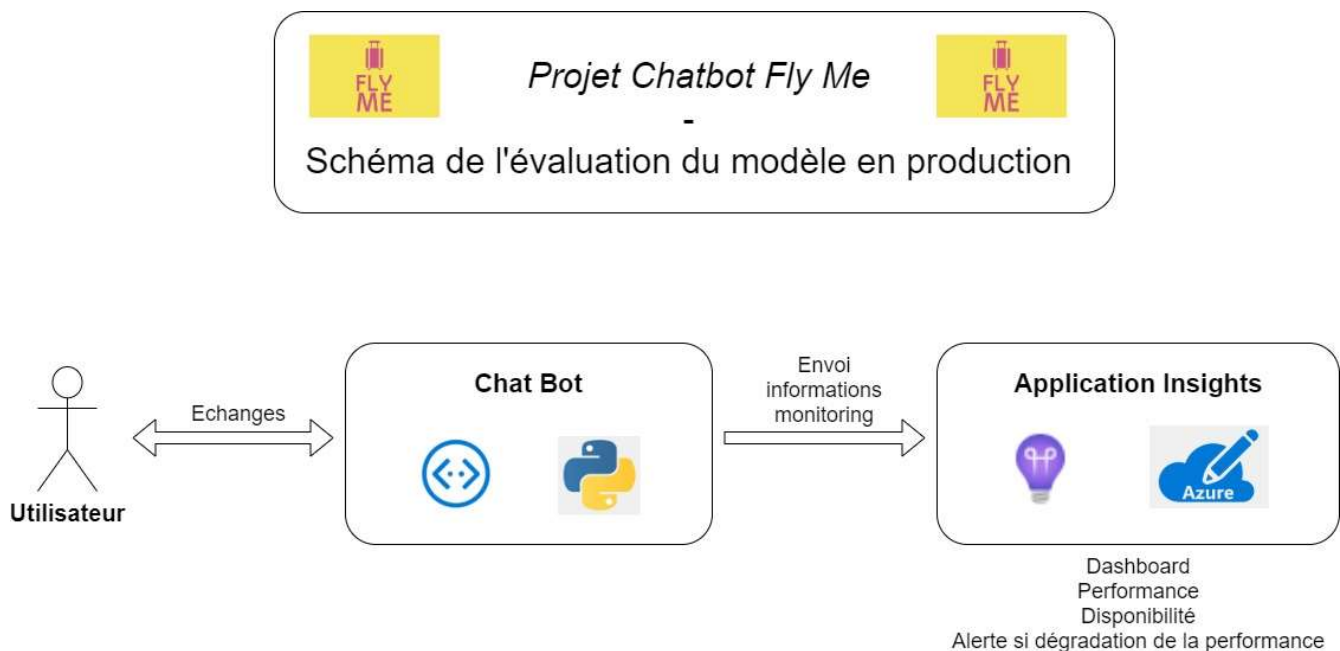
**Custom Event Properties** [Show all](#)

Event time	12/3/2021, 3:58:40.869 PM (Local time)	...
Event name	LuisResult	...

**Custom Properties**

applicationId	52c42e40-1395-47e1-9fcb-d8e59de2d71b	...
intentScore2	0.93	...
intentScore	0.93	...
entities	[{"instance": [{"ArrivalDate": [{"startIndex": 55, "endIndex": 71, "text": "20 december 2021", "type": "ArrivalDate", "score": 0.936669}], "DepartureDate": [{"startIndex": 35, "endIndex": 51, "text": "..."}]}], [{"show more"}]	...
intent2	OrderTravellIntent	...
fromId	4bbcc712-c65d-477e-bb17-eb49dc3372dd	...
intent	OrderTravellIntent	...

## 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 entités)
  - c. Puis on ré-entraîne le modèle LUIS avec ces échanges en lançant le script dédié à cette tâche (*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.**