

# Méthodologie sur le pilotage de la performance du modèle en production

---

## Sommaire

I.	Contexte .....	2
II.	Zoom sur Luis .....	2
III.	Notre Bot.....	4
a)	Son fonctionnement.....	4
b)	Communiquer avec le bot .....	5
IV.	Comment définir des critères d'évaluation du modèle ?.....	6
V.	Le schéma du mécanisme d'évaluation du modèle.....	7
VI.	Points critiques.....	13
VII.	Conclusion : Pistes d'améliorations .....	13

## I. Contexte

Notre problématique était de développer un Bot pour l'entreprise "Fly Me". L'intérêt est d'aider les clients à choisir une offre de voyage.

Pour cela, nous utilisons deux modules:

- Bot Framework SDK : il permet aux développeurs de modéliser des conversations et de réaliser des bots facilement.
- LUIS (Language Understanding Service) : Il fait partie du module Cognitives Services d'Azure utilisé pour la compréhension naturelle du langage, il permet aux utilisateurs d'interagir avec les bots.

## II. Zoom sur Luis

Création de deux applications dans Azure :

- Une application « autorité » pour entraîner le modèle,
- Une application « prédition » pour mettre en production le modèle et réaliser des prédictions.

Pour entraîner le modèle, il nous faut réaliser plusieurs étapes :

- 1) Création des modules Luis : « autorité » et « prédition »
- 2) Création d'une application conversationnelle APP-PROJECT10CHATBOTFLY-WEU-001
- 3) Création d'un schéma conversationnel
  - a) Intentions : capturer les actions : bookFlight
  - b) Entités : Extraire les informations pertinentes du texte :
    - Ajout d'entités prédéfinies : Datetime\_V2, Geography\_V2, number
    - Ajout d'entités personnalisées : Book Flight (qui contient des sous-entités voir figure 1)

The screenshot shows the Luis builder interface with the 'Book Flight' entity selected. The entity structure is as follows:

- dst\_city**: Sub-entity of Book Flight. Contains a 'city' entity with a 'geographyV2' feature.
- or\_city**: Sub-entity of Book Flight. Contains a 'city' entity with a 'geographyV2' feature.
- str\_date**: Sub-entity of Book Flight. Contains a 'date' entity with a 'datetimeV2' feature.
- end\_date**: Sub-entity of Book Flight. Contains a 'date' entity with a 'datetimeV2' feature.
- budget**: Sub-entity of Book Flight. Contains a 'value' entity with a 'number' feature.

Figure 1 : les sous-entités de « Book Flight »

- 4) Etiquetage de nos données : nous ajoutons des exemples de phrases qui seront étiquetées avec les entités et les sous-entités (voir un exemple figure 2)

The screenshot shows the Luis builder interface with a sentence being annotated:

he'llllo newlywed young lover here looking to skeedaddle off to **sao paulo** for 9 days or so after our reception on **august 27** in **athens** , two of us have about **2700** dollars left after the wedding so keep that in mind

Annotations:

- 'sao paulo' is labeled as 'dst city'
- 'august 27' is labeled as 'str date'
- 'athens' is labeled as 'city'
- '2700' is labeled as 'value'

Figure 2 : Etiquetage d'une phrase avec les entités et les sous-entités

Ensuite, il suffit d'entrainer le modèle Luis comme vu dans le notebook (joint avec le projet) et de publier le modèle dans l'application de production de Luis.

Concernant le temps d'entraînement de Luis pour 1 000 données environ, nous avons un délai d'entraînement de 30 secondes environ. Ainsi si nous devons entraîner :

- 10 000 données : nous aurons un délai de 300 secondes
- 1 000 000 données : nous aurons un délai de 30 000 secondes soit 8h environ

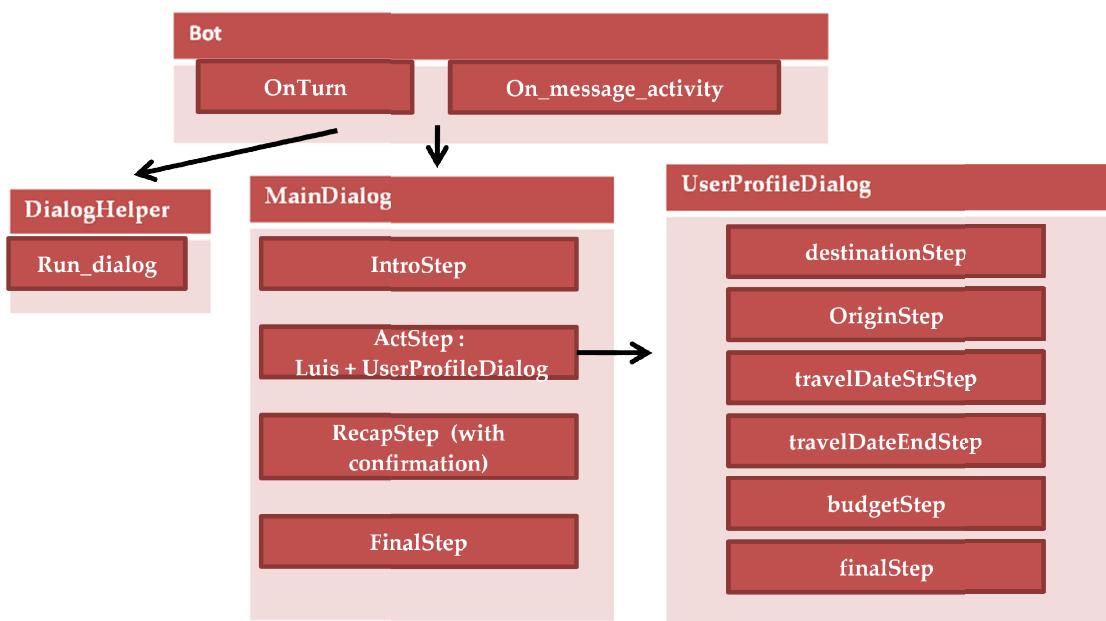
A noter que la mise en production de notre modèle a pris 201 secondes (soit 3min21).

### III. Notre Bot

#### a) Son fonctionnement

Grâce aux deux modules précédant, nous avons réalisé notre bot.

Voici un schéma décrivant le fonctionnement de notre Bot :



Lors de l'envoi d'un premier message, le bot commence un tour avec « on\_turn » et commence le dialogue principal qui suit les étapes suivantes :

- IntroStep qui correspond à la vérification de la configuration de Luis, s'il est configuré ou non. Si Luis est configuré, nous demandons à l'utilisateur « What can I help you today ? », sinon nous passons à l'étape suivante.
- ActStep : si Luis n'était pas configuré, nous passons directement au dialogue UserProfileDialog qui consiste à poser une série de questions (description plus bas). Si Luis était configuré, nous envoyons la réponse obtenue à la question précédente au modèle Luis pour réaliser une prédiction. Il nous renvoie la réponse avec le descriptif des entités et des sous-entités. Nous reformatons ceci pour extraire les informations et remplir la classe nommée UserProfil. Si toutes les informations de la classe sont

- remplies, nous passons à l'étape suivante, si non, nous lançons le dialogue UserProfileDialog.
- UserProfileDialog consiste à poser une série de question pour obtenir les informations suivantes :
    - o La ville de destination
    - o La ville de départ
    - o La date de début souhaitée : avec une vérification quelle soit supérieure à aujourd'hui
    - o La date de fin souhaitée: avec une vérification quelle soit supérieure à aujourd'hui
    - o Le budget : avec une vérification que la valeur entrée est un nombre et supérieure à 0

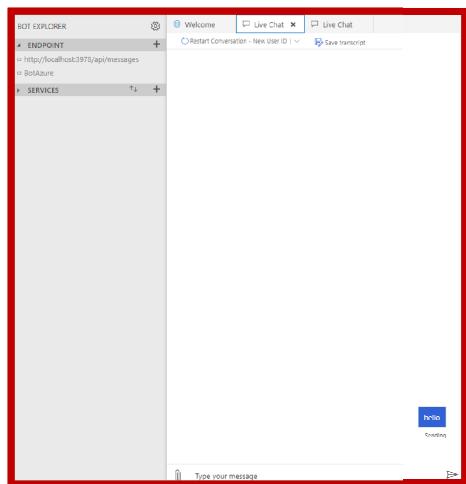
Si toutes les questions ont été correctement remplies, nous repassons à l'étape suivante de MainDialog.

- RecapStep : Ici, nous envoyons un récapitulatif des réponses entrées par l'utilisateur et nous lui demandons si cela correspond à sa demande. Nous traçons sa réponse dans une App insight APS-P10CHATBOT-WEU-001 (un service de gestion des performances des applications et dont les bots). Nous passons ensuite à l'étape suivante.
- FinalStep : Nous demandons à nouveau à l'utilisateur « What else can I do for you? » et nous récréons un nouveau dialogue au début.

## b) Communiquer avec le bot

Pour communiquer avec le bot, nous pouvons utiliser Bot Framework Emulator que ce soit pour tester notre bot en local ou communiquer avec le bot en production.

Figure 3 : Bot Framework Emulator



Pour communiquer avec le Bot avec ce « logiciel » :

- En local, il suffit de configurer l'URL local : <http://localhost:port/api/messages> pour réaliser les tests en local,
- en production, il faut :
  - 1) Créer un tunnel via ngrok (plus d'information ici :  
[https://github.com/Microsoft/BotFramework-Emulator/wiki/Tunneling-\(ngrok\)](https://github.com/Microsoft/BotFramework-Emulator/wiki/Tunneling-(ngrok)))
  - 2) Entrer les informations sur URL de la webApp : <https://lienwebapp/api/messages>
  - 3) Entrer les informations concernant l'application Azure Bot : BOT-PROJECT10FLY-WEU-001, son ID et son mot de passe.

#### IV. Comment définir des critères d'évaluation du modèle ?

Nous avons réalisé le calcul d'une accuracy d'évaluation en prenant en compte les confirmations des utilisateurs via une requête dans l'app insight :

Accuracy d'évaluation = nombre de confirmation utilisateurs à oui / nombre de confirmation utilisateurs oui + non.

L'accuracy calculée sur le jeu de test lors de l'entraînement du modèle vaut 0.77 (cf. notebook).

Nous pourrons définir les alertes suivantes quand :

- Accuracy d'évaluation est inférieur à 0.77 par jour,
- Taux de bugs du bot est supérieur à 0.50 par jour (nombre de bugs / nombre de confirmations d'utilisateurs oui/non)

Si une de ses alertes est reçue, il faudra :

- pour l'alerte concernant le modèle Luis : programme une mise à jour du modèle en y intégrant les nouvelles données. Les durées d'entraînement dépendent du nombre de données à entraîner (pour 1 000 données environ, l'entraînement dure 30 secondes). De plus, pour réaliser cette mise à jour, il sera nécessaire de suivre les performances du Bot et de voir à quel moment, cette nouvelle version pourra être publiée.
- Concernant l'alerte bugs, il faudra analyser les logs de l'app service et étudier les conversations avec les utilisateurs pour comprendre les erreurs.

La création d'une alerte dans Azure est résumée dans le schéma ci-dessous.

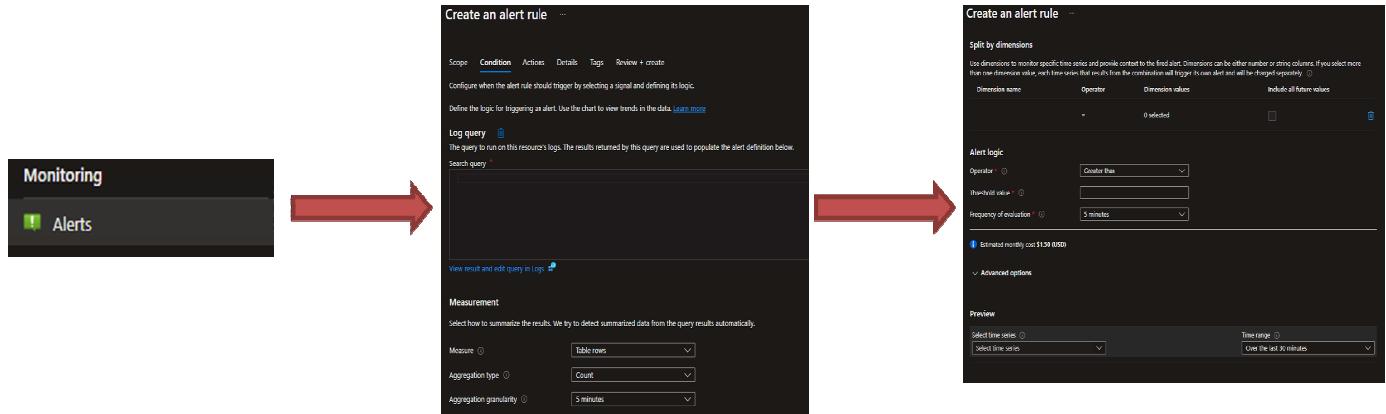
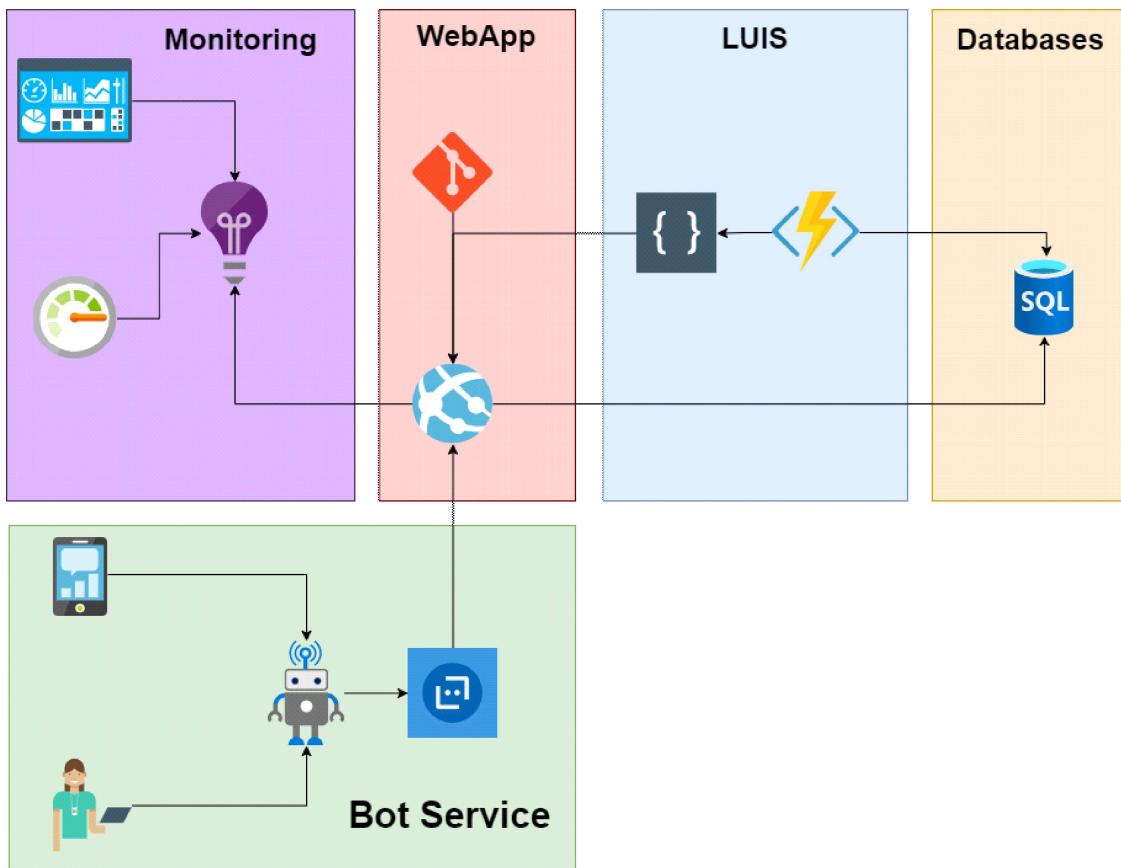


Figure 4 : Définir une alerte dans App insight (Azure)

## V. Le schéma du mécanisme d'évaluation du modèle

Nous avons définis l'architecture cible ci-dessous.

Figure 5 : Architecture Cible



## Description du scénario de l'architecture (figure 5)

- 1) Les données circulent dans le scénario comme suit :
- 2) Le client accède au chatbot avec une application mobile ou web.
- 3) L'utilisateur va intéragir avec le Bot et va entrer des informations sur son vol, les données seront enregistrées dans la base de données SQL et des logs seront envoyées dans App insight
- 4) Luis traite la demande en langage naturel pour comprendre la communication client.
- 5) Une fois que le bot a demandé les éléments manquants si nécessaire, le bot demande une confirmation du client et ajoute ou met à jour les données dans une base de données Azure SQL.
- 6) L'application insight collecte la télémétrie d'exécution tout au long du processus pour détecter les bugs, améliorer les performances et l'utilisation du bot.
- 7) Pour un suivi des performances en temps réel et quotidiennement, un dashboard est mis à disposition, il interroge l'app insight. Des alertes sont aussi présentes pour suivre l'expérience client (voir la partie 4 pour le descriptif des alertes).

## Description des composants

- **Webapp :**

Une App Service dit « webapp » permet de créer et d'héberger des applications Web dans le langage de programmation que l'on souhaite. L'intérêt est qu'Azure gère toute la partie infrastructure (mise à jour etc..), cela permet donc de réduire les coûts.

Nous pouvons aussi réaliser un déploiement automatique sur Azure lors du push sur la branche Main de Github après lancement des tests unitaires.

- **Luis**

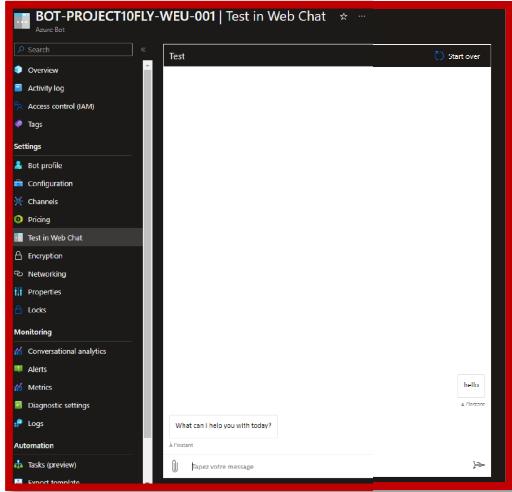
La WebApp fait appel un modèle Luis (Language Understanding) préalablement entraîné. C'est un service d'IA conversationnelle qui applique une intelligence de machine learning personnalisée au texte en langage naturel des conversations d'un utilisateur afin de prédire le sens et d'extraire les informations pertinentes. LUIS fournit un accès via une API.

Nous pouvons entraîner le modèle Luis avec nos données et le mettre à jour quand nous avons besoin via une Azure Function. Elle permettra de créer une nouvelle version du modèle Luis à partir des données SQL (historiques des conversations).

- **Bot Service**

Le module Bot service permet d'avoir des outils pour créer, tester, déployer et gérer les bots intelligents.

Le Bot service fait appel à l'app service sur le web lors de l'envoi d'un message par un client au Bot. Elle permet aussi de tester le bon fonctionnement du Bot.



- **Database SQL**

Nous pouvons y stocker les conversations entre les clients et le bot afin de pouvoir mettre à jour notre modèle. Le client pourra ainsi retrouver une conversation, s'il possède un compte client (nous pouvons utiliser Azure Ad pour cette partie en complément si besoin qui permet de gérer des comptes utilisateurs).

- **Monitoring en temps réel :**

Plusieurs éléments permettent de réaliser ce monitoring :

- Intégration des logs dans App insight qui est un service de gestion des performances des applications et dont les bots.  
Nous pouvons ainsi intégrer toutes les logs nécessaires au suivi de notre bot comme les réponses des utilisateurs suite à leur conversation avec les bots, mais aussi les bugs détectés.
- Monitoring avec envoi d'alerte par Mail ou une application de type Slack. Ceci permet d'être alerté par une channel souhaité comme les e-mails ou slack et d'être immédiatement informé lors du dépassement des seuils définis (voir IV pour le descriptif des alertes).
- Dashboard pour suivre les performances quotidienement du Bot : ci-dessous un dashboard que l'on a paramétré quotidiennement (cf. figure 6):

Figure 6 : Dashboard Suivre les performances du Bot en temps réel

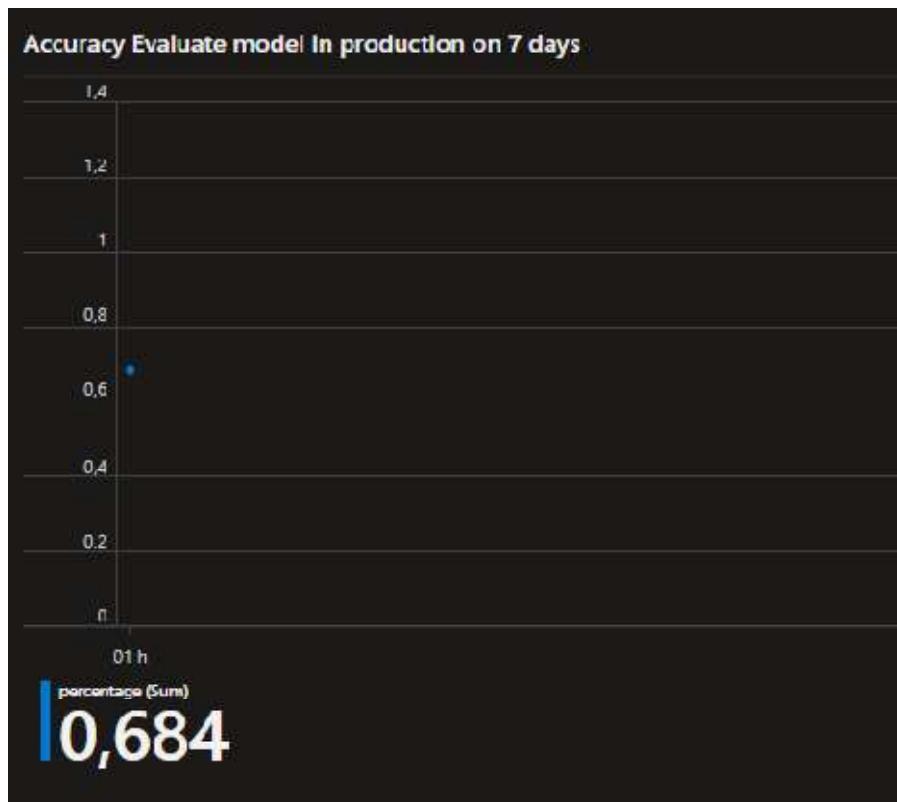
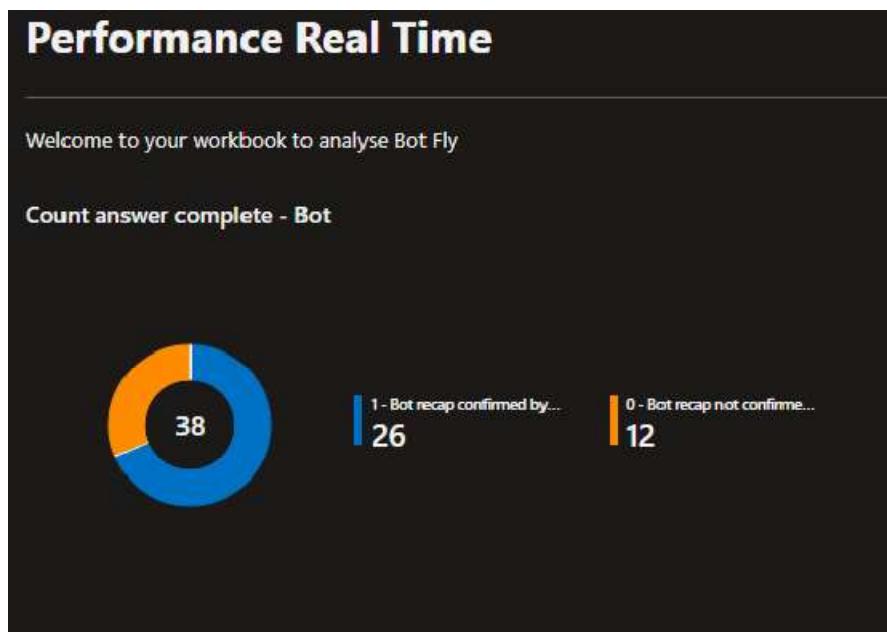


Figure 6 : Dashboard Suivre les performances du Bot en temps réel (suite)

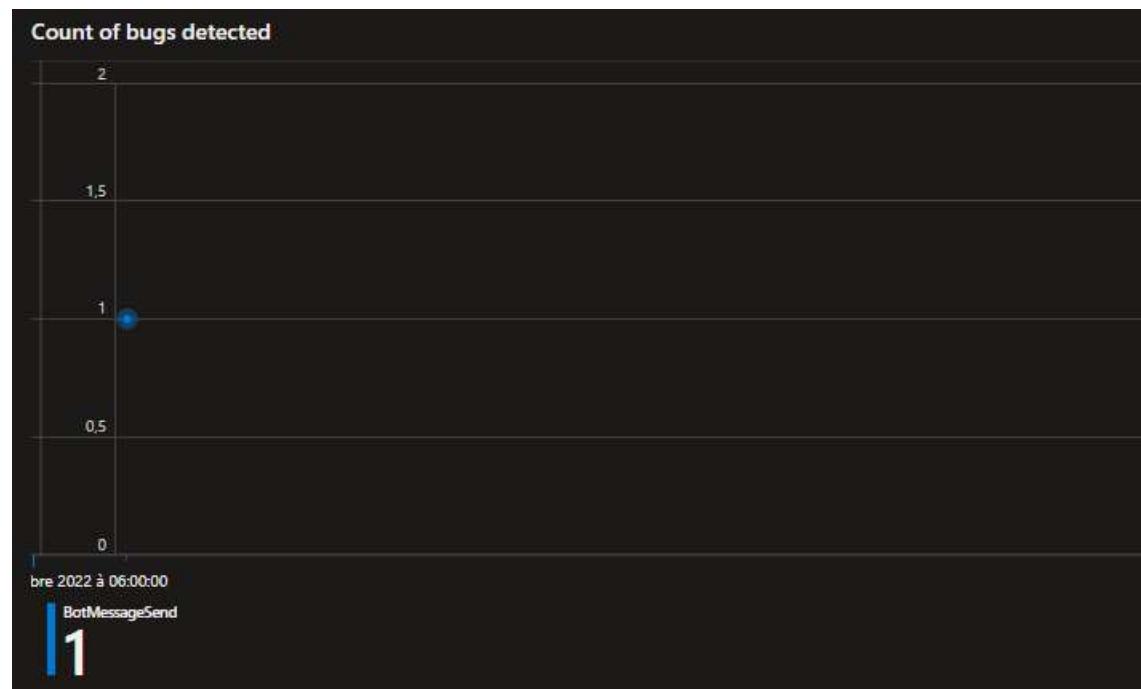
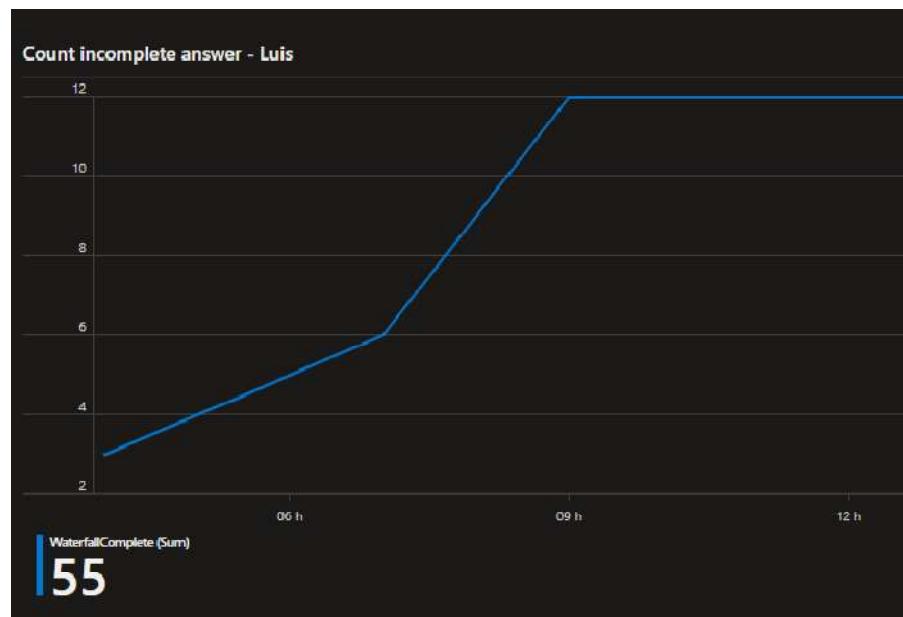
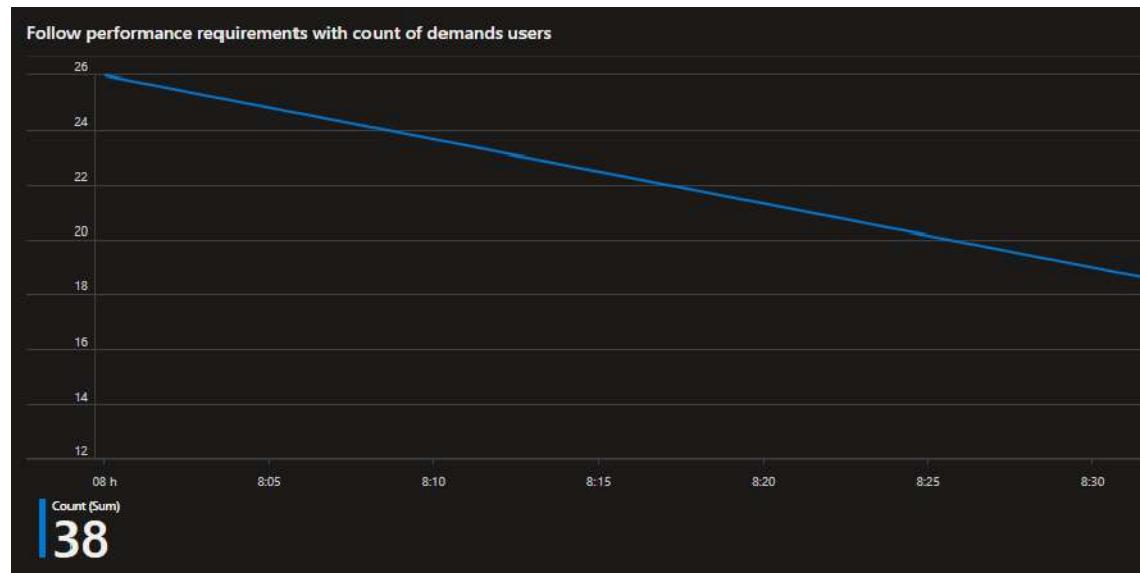


Figure 6 : Dashboard Suivre les performances du Bot en temps réel (suite)



Grâce à ce Dashboard, nous pouvons suivre :

- les réponses des utilisateurs suite à leur conversation avec le bot,
  - l'accuracy d'évaluation du modèle en production sur 7 jours,
  - le nombre de réponses incomplètes de Luis,
  - le nombre de bugs rencontrés par les clients par jour
  - la charge du Bot par heure (nombre de demandes utilisateurs) afin de prévoir quand la charge est la plus présente et nous pourrons ainsi définir une mise à jour du modèle lorsque la demande est faible. De plus, même si un utilisateur réalise une demande le bot prévoit une série de question pour connaître la réservation du vol du client lorsque le modèle Luis est indisponible.
- **Avantages des composants utilisés dans ce scénario**

Nous pouvons souligner que ce scénario permet d'obtenir tous les avantages du cloud à savoir notamment :

- Évolutivité : Ce scénario utilise Azure App Service. Avec App Service, nous pouvons dimensionner automatiquement le nombre d'instances qui exécutent votre bot. Cette fonctionnalité permet de répondre à un pique de demande des clients pour notre chatbot.
- Elasticité : Ce scénario utilise Azure SQL Database pour stocker les réservations des clients. SQL Database inclut des bases de données redondantes de zone, des groupes de basculement, la géo-réplication et des sauvegardes automatiques. Ces fonctionnalités permettent au chatbot de continuer à fonctionner en cas d'événement

de maintenance ou de panne. De plus nous pouvons ajouter une alerte afin de surveiller l'intégrité du chatbot.

Le scénario utilise aussi l'app insight qui permet de suivre les performances, de répondre aux problèmes qui affecteraient l'expérience client et la disponibilité du chatbot.

## VI. Points critiques

Pour soulever les points critiques de l'application, nous avons mis en place des tests unitaires pour vérifier :

- le bon déroulement du dialogue principal (MainDialog)
- le bon déroulement du dialogue UserProfileDialog qui concerne les séries de questions
- Luis répond lors de l'envoi d'une demande complète et renvoie tous les attributs de la classe remplies

Quelques points critiques à noter :

- La date de fin du voyage peut être inférieure à la date de début du voyage,
- Si Luis ne répond pas, nous posons automatiquement toutes les questions pour connaître la demande utilisateur.

## VII. Conclusion : Pistes d'améliorations

Nous pouvons définir plusieurs pistes d'amélioration :

- Réaliser une intégration continue avec un module de type Jenkins (<https://www.jenkins.io/>)
- Monitorer les temps de réponses du Bot pour améliorer l'expérience client