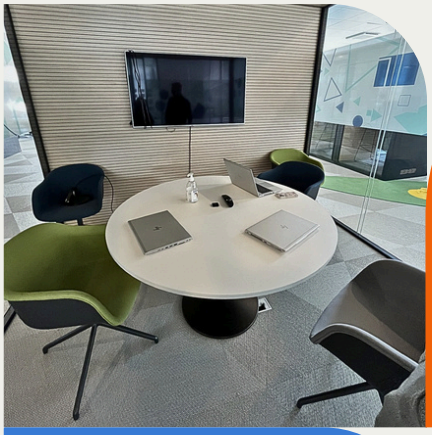
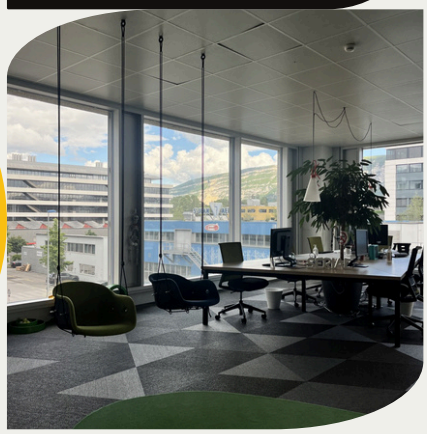
BARRACHIN Carlyne
IDU

2024
21 mai - 31 juillet

RAPPORT DE STAGE

Assistant Ingénieur - 4ème année





MAÎTRE DE STAGE ..... BRIFFOD Audrey

TUTEUR DE STAGE ..... VALET Lionel

Collaboration Betters The World

Plan-les-Ouates

Année scolaire 2023/2024

Remerciements

Je souhaite exprimer ma profonde gratitude envers l'entreprise CBTW pour m'avoir accueilli durant ces deux mois et demi de stage. Il n'est pas facile pour un étudiant en 4ème année d'école d'ingénieur de trouver un stage, et je suis particulièrement reconnaissant de cette opportunité.

Je tiens à remercier tout spécialement Mme **Audrey BRIFFOD**, qui a supervisé mon stage. À la suite de son intervention à Polytech, elle a pris le temps de me trouver un stage au sein de son entreprise, malgré l'absence initiale de sujet de stage. Elle m'a accompagnée tout au long du stage, en étant toujours présente, m'accordant sa confiance et m'assignant des missions valorisantes.

Je tiens également à exprimer ma gratitude envers toute l'équipe de l'entreprise pour leur accueil chaleureux. Je les remercie particulièrement pour avoir pris le temps de me montrer concrètement leur travail et les outils utilisés, que ce soit dans le domaine de la Data, du développement Full Stack ou de la RPA (Automatisation des Processus Robotisés).

Un merci spécial à l'équipe RPA pour m'avoir intégré et accueilli parmi eux.

Enfin, je souhaite remercier M. **Lionel VALET**, mon tuteur de stage, pour son accompagnement tout au long de cette période. Il a suivi attentivement le déroulement de mon stage et a toujours été disponible pour répondre à mes questions.

Table des matières

Table des illustrations

Introduction

Dans le cadre du cursus d'ingénieur **« Informatique, Données, Usages »** à Polytech Annecy, il est requis d'accomplir, à la fin de la 4ème année, un stage en tant qu'assistant ingénieur d'une durée minimale de 6 semaines. Ainsi, j'ai eu l'opportunité de rejoindre l'entreprise **CBTW** pendant 11 semaines, du 21 mai au 31 juillet 2024, pour mettre en pratique les compétences techniques acquises au cours de ces quatre dernières années.

Grâce à Polytech Annecy et au module du 8ème semestre « ISOC831 – Dimension métiers », j'ai développé un fort intérêt pour l'entreprise CBTW. En effet, ce module nous a permis de découvrir concrètement deux des nombreux métiers de CBTW. Le premier métier que nous avons exploré était le design, plus précisément sur le thème du **MobileFirst**, présenté par M. Thibault COUHIN le 4 avril 2024. Lors de cette présentation, il nous a expliqué de manière captivante en quoi consistait ce métier et l'importance de développer des applications compatibles avec les téléphones mobiles. Nous avons participé à un atelier pratique où nous devions, par groupe, réfléchir à une idée d'application mobile à inventer. Pour ce projet, il était crucial de prendre en compte de nombreux aspects, tels que les aspects techniques, l'expérience utilisateur, et bien d'autres éléments. Cette activité était à la fois très intéressante et concrète.

Dans la continuité de cette expérience, Mme Audrey BRIFFOD est intervenue le 8 avril pour nous présenter un autre métier dont je n'avais jamais entendu parler auparavant : l'**Hyperautomation**. Lors de cette présentation, nous avons compris concrètement en quoi consistait ce domaine et avons utilisé, au cours du cas pratique, l'outil UiPath pour créer une automatisation visant à récupérer des informations sur des ordinateurs sur le site de la Fnac, afin de les comparer par la suite. Cette expérience pratique nous a permis de saisir l'importance de ces outils pour éviter à de nombreuses personnes d'effectuer des tâches répétitives toute la journée. Impressionné par ce qu'il est possible d'automatiser, j'ai immédiatement demandé s'il était possible de réaliser un stage au sein de leur entreprise et dans ce secteur plus précisément.

Après de nombreux échanges par courriel, un entretien traditionnel et un entretien technique, j'ai eu la chance d'être acceptée pour un stage chez CBTW. Ce stage, que j'ai effectué en binôme avec Théo PLEBANI, également étudiant en IDU à Polytech Annecy, a été une opportunité inestimable pour approfondir mes compétences et découvrir le monde professionnel.

L'objectif de ce rapport est de retracer mon expérience lors de mon stage chez CBTW et d'analyser en profondeur les connaissances acquises et les compétences. Nous examinerons notamment le projet auquel j'ai contribué, les technologies utilisées et les enseignements tirés de cette expérience. En outre, nous évaluerons l'impact de ce stage sur mon parcours académique et professionnel, ainsi que les perspectives dans le domaine de l'Hyperautomation.

1 La structure d'accueil

Dans cette première partie, nous explorerons l'entreprise CBTW en nous penchant sur son fonctionnement global ainsi que sur ses diverses activités. Nous porterons une attention particulière au service dans laquelle j'ai effectué mon stage.

1.1 Présentation de CBTW

1.1.1 Son histoire

L'histoire de CBTW commence en **1986** avec la création de **SERIAL** par trois ingénieurs.

En **2020**, quatre ans après avoir rejoint le groupe **Positive Thinking Company**, **SERIAL** adopte ce même nom, mettant en avant ses six domaines d'expertise distincts : Data & Analytics, Digital Workplace, Hyperautomation, Software Product Engineering, Cloud et Security.

En **2022**, six partenaires fondateurs, à savoir Positive Thinking Company, Versett, AlisPharm, Otofacto, OneAston, et SteepConsult, cocréent **Collaboration Betters The World**.

Enfin, en **2024**, ces six fondateurs partenaires fusionnent sous une seule et même marque : **CBTW, Collaboration Betters The Word**.

Aujourd'hui, l'entreprise prônant des valeurs fondamentales telles que la **collaboration**, un **état d'esprit positif**, l'**évolution** et l'**engagement**, compte plus de 3000 collaborateurs répartis dans 21 pays (voir ***Figure 1***).

1.1.2 Ces services

Leur expertise couvre **trois principaux domaines** : la stratégie et les produits (Strategy & Product), la technologie (Tech) et les services sectoriels (Sector). En créant des liens (voir ***Figure 2***), c'est-à-dire en combinant ces différentes lignes de services appelées Services Lines, CBTW acquière une connaissance approfondie des secteurs qui les intéressent ainsi qu'un solide réseau de fournisseurs technologiques et de commerciaux. Plus précisément, ils sont spécialisés dans divers secteurs comprenant la Banque & la Finance, la Santé, la Production, la Vente au détail, les Télécommunications & les Médias, l'Assurance, le Transport… Ainsi pour répondre aux besoins spécifiques de ceux-ci, ils ont organisé leurs activités en 9 services Lines :

Domaine 'Strategy & Product': Strategy & Governance, Product Design & Growth
Domaine 'Tech': Software Engineering, Data Analytics & AI, Cyber Security, **Cloud & Enterprise Platforms**
Domaine 'Sector': Smart Industrial Solutions**,** Life Sciences Solutions**,** Banking Technology Solutions

1.2 Présentation du Service

1.2.1 Cloud & Enterprise Platforms

Mon stage s'est déroulé au sein du service **Cloud & Enterprise Platforms**, situé dans les locaux de Plan-les-Ouates en Suisse.

Ce service se divise en trois principaux domaines :

- Cloud & Infrastructure
- **Process Automation & Optimization**
- Low-Code & Enterprise Platforms

J'ai été affecté au domaine de **Process Automation & Optimization**, qui englobe des aspects essentiels de l'**Hyperautomation**, notamment la **Robotic Process Automation (RPA)**.

1.2.1.1 Hyperautomation

L'**Hyperautomation** représente une méthode avancée d'automatisation intégrant des technologies telles que **l'Intelligence Artificielle** (IA), **l'Apprentissage Automatique** (Machine Learning - ML) et la **Robotique** (RPA). Son application vise à automatiser et à optimiser une multitude de **processus métier**, allant des tâches simples aux plus complexes. L'objectif est d'améliorer l'efficacité opérationnelle et de faciliter la prise de décision au sein d'organisations. Cette approche va au-delà de l'automatisation de tâches isolées en s'attaquant à des processus interconnectés et complexes à travers toute l'entreprise, permettant ainsi une transformation profonde des opérations métier.

Par exemple, dans le département des Ressources Humaines, l'Hyperautomation pourrait permettre d'automatiser la génération des fiches de paie. En intégrant l'IA pour collecter les données, le ML pour les calculs d'erreurs, de coût et prévision, et la RPA pour la distribution, ce processus devient rapide et précis, réduisant la charge administrative.

1.2.1.2 RPA

La Robotic Process Automation (RPA) représente un élément essentiel de l'Hyperautomation, fournissant un ensemble d'outils technologiques dédiés à l'automatisation de processus. Son principal objectif est de cibler les tâches répétitives et manuelles, offrant ainsi une solution efficace pour se concentrer sur des activités plus stratégiques.

L'un des principaux attributs de la RPA est son imitation du comportement humain grâce à des robots logiciels, en interagissant de manière cohérente avec les applications du poste de travail. Cette capacité à travailler en parallèle avec les systèmes existants est illustrée dans la structure de la plateforme RPA, où l'on distingue deux types de robots : les robots "**Unattended**" et "**Attended**".

Les robots "**Unattended**" fonctionnent de manière autonome sur des postes de travail dédiés, exécutant des tâches planifiées ou déclenchées par des événements spécifiques. Ils déchargent entièrement les utilisateurs des tâches automatisées, libérant ainsi du temps et des ressources. En revanche, les robots "**Attended**" opèrent sur le poste de travail de l'utilisateur, travaillant en coopération avec ce dernier. Ils peuvent être déclenchés manuellement par l'utilisateur ou en réponse à des événements sur le poste de travail, permettant ainsi à l'utilisateur de maintenir le contrôle sur le processus.

Un exemple d'outil largement utilisé pour l'automatisation des processus est **UiPath**, qui offre une gamme de fonctionnalités et de solutions pour répondre aux besoins variés en matière d'automatisation des entreprises.

Ce logiciel RPA suit la structure présentée dans la ***Figure 3***, qui sera détaillée dans les sections suivantes de ce rapport.

1.2.2 L'équipe

Mme Audrey BRIFFOD, notre maître de stage, occupe le poste d'**ISM** (Innovation Solution Manager) au sein de **l'équipe RPA**, qu'elle supervise et gère. Ses missions tournent principalement autour de la **plateforme UiPath**, dont l'entreprise est partenaire depuis 2018. Elle accompagne les clients intéressés par la RPA à travers tout le processus d'intégration de ces outils d'automatisation. Cela inclut la vente de **licences**, la gestion des **parcs**, l'**implémentation** et l'**intégration** de solutions adaptées aux **besoins** des **clients**, ainsi que le **support** et la **maintenance**. Elle assure un **suivi continu** de ses clients, intervenant même sur site si nécessaire, toujours appuyée par son équipe.

Les 9 membres de l'équipe RPA sont dispersés dans différentes localisations telles que Lausanne, Genève, Toronto et Paris, avec des contrats variés, certains étant entièrement dédiés à la clientèle avec un contrat "Forfait" à 100%. La **communication** au sein de l'équipe se fait principalement via des plateformes telles que **Teams** et par **courriel**. Même lorsque nous sommes physiquement présents au même endroit, nos deux jours de télétravail par semaine peuvent différer. Il est donc essentiel de savoir **s'adapter** et de bien **communiquer** avec **l'équipe**.

L'équipe compte maintenant deux stagiaires : Théo PLEBANI et moi-même. Travaillant sur les mêmes projets, nous mettons l'accent sur **le travail d'équipe**, la **collaboration** et la **communication** pour atteindre nos objectifs communs, toujours avec le soutien des équipes si nous avons la moindre question.

Voyons maintenant pourquoi nous avons intégré ce groupe.

2 Problématique et objectifs

Il est ensuite essentiel de comprendre l'**origine** de notre stage, les **attentes** du groupe en termes de **résultats**, ainsi que l'utilité et les **bénéficiaires** de notre travail.

2.1 Contexte

Le groupe RPA gère **plusieurs clients**, chacun disposant d'une licence **UiPath**. Le groupe a la possibilité d'accéder aux comptes clients pour vérifier le bon fonctionnement de leurs processus sur la plateforme. Cependant, avec une dizaine de clients, il est difficile de se connecter quotidiennement à chaque compte pour **surveiller les performances et anticiper les problèmes éventuels**. Actuellement, la seule façon pour le groupe de détecter les erreurs de certains processus est d'attendre que les clients signalent les problèmes, les informant ainsi avant même que le groupe RPA ne soit au courant.

2.2 Résultat final

L'objectif principal est d'avoir accès à une **plateforme** où tout serait directement **centralisé** permettant de **détecter** et **analyser** les **problèmes** potentiels avant même que les clients ne les remarquent, facilitant ainsi une **résolution rapide**. De plus, il s'agissait d'établir un **système de notification automatique** pour alerter le groupe RPA dès qu'un problème survient, éliminant ainsi le besoin de vérifications manuelles quotidiennes.

Sur cette plateforme, il est également intéressant d'intégrer des **statistiques** globales visibles instantanément sur **un tableau de bord**. Cela inclut un aperçu direct des problèmes pour chaque client, ainsi que des **détails** spécifiques sur les erreurs rencontrées.

2.3 Utilisation

Cette plateforme sera utilisée par l'ensemble du **groupe RPA**. Notre maître de stage et certains autres membres l'utiliseront principalement pour **planifier** rapidement des **réunions** avec les clients en cas de détection de problèmes. Elle sera également employée par d'autres membres de l'équipe chargés **d'effectuer des processus**. Ils pourront ainsi identifier rapidement toutes les erreurs, comprendre leur origine et les corriger efficacement. Cela permettra de gagner du temps en évitant de devoir naviguer et chercher sur la plateforme UiPath et dans les processus du client les informations nécessaires à la résolution des problèmes.

3 Technologies utilisées

Cette troisième partie est essentielle pour comprendre les **missions** effectuées. Il est important d'appréhender les **outils** utilisés afin de comprendre leur utilité et la raison de leur emploi. De nombreuses **notions** associées à ces outils doivent être assimilées pour bien saisir nos méthodes et objectifs.

3.1 UiPath

Les outils présentés dans les parties suivantes, à savoir UiPath Studio, UiPath Orchestrator et les Robots, sont illustrés dans la ***Figure 3***. Ensemble, ils forment une suite complète et intégrée, essentielle pour mettre en place et maintenir des solutions RPA robustes et performantes.

3.1.1 Notions Importantes

UiPath est l'outil central de notre stage, il est donc important de définir certains termes clés associés à cet outil.

Processus

Un processus, dans le contexte de l'automatisation robotique (RPA), est une **séquence d'étapes** automatisées conçue pour accomplir une fonction spécifique. Ces tâches peuvent inclure des actions telles que la lecture de données, l'interaction avec des applications, l'envoi de courriels, ou la manipulation de fichiers. Les processus sont créés dans des environnements de développement comme **UiPath Studio** et peuvent être **déployés** et **exécutés** par des robots RPA.

Job

Un Job est une **instance d'exécution d'un processus automatisé**. Lorsqu'un processus est déclenché pour s'exécuter, cela lance un Job. Les Jobs peuvent être initiés **manuellement** par un utilisateur ou **automatiquement** selon une planification définie ou des conditions spécifiques. Chaque Job a son propre **état** et **historique d'exécution**, permettant de suivre et de gérer l'exécution des processus. Les états d'un Job peuvent inclure "**Successful**" (réussi), "**Failed**" (échec), "**In Progress**" (en cours), "**Pending**" (en attente) et d'autres encore. Chaque état communique des informations précieuses sur la progression et l'issue du Job, facilitant ainsi la supervision et la résolution des problèmes éventuels.

Robot

En RPA un robot est un **logiciel** qui **imite les actions humaines** pour effectuer des tâches spécifiques de manière **automatisée**. Les robots peuvent être **physiques** (sur des postes de travail) ou **virtuels** (sur des serveurs ou dans le cloud). Ils **exécutent** les **processus** définis dans des outils comme **UiPath Studio** et sont gérés et orchestrés par des plateformes comme **UiPath Orchestrator**. Les robots peuvent être classés en **deux catégories** principales comme vu précédemment : **Attended** et **Unattended**.

Lorsqu'un processus est lancé par un robot, celui-ci renvoie des **logs détaillés** ou journaux qui permettent de **suivre** et d'**analyser** l'exécution du processus. Les logs affichent des informations sur chaque étape, y compris les messages d'information (**Info**), les avertissements (**Warning**) et les erreurs (**Error**). Ces logs sont essentiels pour comprendre précisément ce qui s'est passé lors de l'exécution d'un Job, identifier les points de défaillance et prendre des mesures correctives si nécessaire.

Transaction

Une transaction est une **tâche spécifique** au sein d'un **processus** automatisé. Dans les systèmes RPA, une transaction représente généralement une **opération** ou une **interaction unique** avec un système ou une base de données. Par exemple, une transaction pourrait être la lecture d'une ligne de données dans un fichier Excel. Grâce aux logs des robots, il est possible de vérifier si **chaque transaction a été effectuée correctement**. Les logs permettent de voir les détails de chaque transaction, indiquant si elle a réussi ou échoué (Info, Warning, Error), et en fournissant des informations précises sur les éventuelles erreurs rencontrées.

WebHook

Un WebHook est une **méthode de communication** en **temps réel** entre applications. Contrairement à une API traditionnelle où une application doit interroger régulièrement pour obtenir des données (requête/réponse), un WebHook **envoie automatiquement des données** à une URL prédéfinie dès qu'un **événement** spécifique se produit (voir ***Figure 4***). Cela permet une surveillance **proactive** et **réactive**, ainsi que **l'envoi de notifications en temps réel**.

3.1.2 UiPath Studio

UiPath Studio est un **environnement de développement intégré** (IDE) utilisé pour **concevoir** et **créer** des **processus** d'automatisation robotique (**RPA**). Il permet de créer des workflows automatisés en utilisant une **interface visuelle**. Cette application permet d'automatiser des tâches répétitives et chronophages, améliorant ainsi l'efficacité opérationnelle et réduisant les erreurs humaines.

Ce logiciel offre une vaste gamme de fonctionnalités pour la création et la gestion des workflows automatisés, telles que :

- **Conception visuelle** : Utilisation d'une interface de type "glisser-déposer" pour créer des workflows automatisés sans nécessiter de compétences avancées en programmation. Il est possible de réaliser des actions comme dans le développement traditionnel, telles que la création de boucles, la gestion des conditions et l'utilisation de variables.
- **Bibliothèque d'activités** : Large collection d'activités préconstruites pour diverses actions, comme la manipulation des fichiers, la gestion des courriels, l'interaction avec les bases de données, etc.

- **Enregistrement des actions** : Enregistreurs qui capturent les actions de l'utilisateur tels que des clics, saisies de texte, navigations et génèrent automatiquement les activités correspondantes sans aucune interactions humaines.
- **Publication** : Une fois le workflow testé et validé, il peut être publié directement depuis UiPath Studio vers UiPath Orchestrator ou être exporté pour une utilisation indépendante.

3.1.3 UiPath Orchestrator

UiPath Orchestrator est une **plateforme web** qui permet de **déployer**, **gérer** et **surveiller** les **processus** automatisés créés avec UiPath Studio. Elle sert de point central pour l'**administration** et l'**orchestration** des **robots RPA** dans une organisation ou une entreprise.

Dans un Orchestrator, il peut y avoir plusieurs **tenants**. Chacun représente un sous-groupe ou une entité organisationnelle distincte avec ses propres utilisateurs, robots, processus et configurations. Cela permet une gestion séparée et sécurisée des ressources et des données pour différentes équipes ou départements au sein de la même organisation.

Nous pouvons retrouver de nombreuses fonctionnalités, notamment :

- **Déploiement** : Les processus automatisés publiés depuis UiPath Studio vers l'Orchestrator sont ensuite disponibles pour être déployés sur les robots connectés.
- **Lancement manuel des processus** : Il est possible de lancer manuellement un processus depuis cette plateforme pour des interventions ponctuelles ou urgentes.
- **Planification des tâches** : Programmation d'exécutions de processus.
- **Gestion des robots** : Attribution des robots à différents processus et contrôle de leurs performances et de leur utilisation en fonction de leur état et de leurs capacités.
- **Surveillance en temps réel** : Suivre l'état et les performances des robots et des processus en temps réel.
- **Journalisation et rapports** : Récupérer des logs détaillés des activités des robots et générer des rapports pour l'analyse des performances et le suivi des incidents. Ces rapports permettent une analyse détaillée des performances et aident à identifier les goulots d'étranglement ou les opportunités d'optimisation des processus.

3.2 API Orchestrator

C'est à partir de l'outil Orchestrator de UiPath que nous pouvons récupérer toutes les informations nécessaires, grâce à son API associée.

3.2.1 SWAGGER

Swagger est un langage de **description d'interface** utilisé pour définir des APIs à l'aide de **JSON**. Cet outil offre une **plateforme centralisée** qui facilite la **conception**, le **test** et la **documentation** des APIs, tout en fournissant une **vue claire et structurée** des fonctionnalités disponibles.

Dans mon cas, les fonctionnalités principales que j'ai exploitées incluent :

- **Accès aux Endpoints** : Utilisation de l'interface Swagger pour accéder à l'ensemble des endpoints de l'API Orchestrator, permettant ainsi d'explorer et de tester chaque fonctionnalité disponible.
- **Exécution de Requêtes** : Utilisation des requêtes HTTP pour tester diverses opérations de l'API, telles que la récupération de données comme les logs envoyés par les robots, les Jobs, et toutes les informations connexes. Cela permet d'explorer les capacités de l'API et de vérifier la disponibilité des données à récupérer.
- **Visualisation des Résultats** : Possibilité de visualiser les résultats des requêtes effectuées, évaluant ainsi la qualité et la conformité des réponses de l'API.

3.2.2 Postman

Postman est un **outil de développement API** utilisé pour **tester**, **déboguer** et **collaborer** sur des APIs. Il offre une interface permettant aux utilisateurs de **créer**, d'**envoyer** et de **recevoir** des requêtes HTTP et des réponses API. L'outil peut être utilisé pour diverses tâches liées au développement API, telles que le test de l'API, la création de collections de requêtes, la documentation API, la gestion des environnements de test et l'automatisation des tests.

Dans mon utilisation de Postman, j'ai bénéficié des fonctionnalités clés suivantes :

- **Enregistrement des Requêtes** : Postman permet d'enregistrer toutes les requêtes effectuées, y compris les paramètres et les corps de requête, facilitant ainsi leur réutilisation ultérieure. J'ai pu enregistrer les requêtes testées au préalable avec Swagger, ce qui m'a permis d'y accéder plus rapidement par la suite.
- **Organisation des Requêtes** : Postman offre des fonctionnalités d'organisation avancées, telles que la création de collections et de dossiers, permettant de structurer et de catégoriser les requêtes en fonction de leur objectif ou de leur type.

3.3 Microsoft Azure

La personne responsable de Managed Services Platform nous a demandé d'utiliser Azure pour ce projet étant donné que leur plateforme l'utilise déjà en grande partie. Nous avions déjà eu l'occasion de l'utiliser à Polytech lors d'un TP, ce qui a facilité sa prise en main.

Azure est une plateforme de **cloud computing** développée par Microsoft. Le cloud computing consiste à utiliser des serveurs informatiques à distance, hébergés dans le monde entier et connectés via un réseau, pour le **stockage**, la **gestion** et le **traitement** des **données**. Azure propose donc de nombreux services cloud tel que des services de calcul, de stockage, de bases de données, de machine learning, d'intelligence artificielle et bien plus encore.

Pour mieux comprendre comment cette plateforme peut être utilisée pour développer des applications flexibles et évolutives, examinons de plus près certains de ses services clés, notamment **Azure Functions**, les **Functions App** et les **Chaining Functions**.

3.3.1 Azure Functions

Grâce à ce service cloud, il suffit de se concentrer uniquement sur **l'écriture du code** dans le langage souhaité. Ce service **gère** tous les **aspects techniques** tels que la gestion des infrastructures sous-jacentes, le déploiement et la maintenance des serveurs. L'infrastructure cloud mise à disposition assure la disponibilité de tous les serveurs à jour nécessaires.

Il est possible de développer dans l'IDE souhaité et déployer le code directement dans le cloud Azure. Cela est facilité en utilisant par exemple Visual Studio Code avec l'extension Azure disponible. Pour l'**hébergement**, plusieurs options sont disponibles : le serverless complet où l'on paie seulement le temps d'exécution, ou des instances prêtes en permanence.

L'objectif principale de ce service est de **réagir aux événements** en implémentant des fonctions Azure individuelles déclenchées par exemple par une requête HTTP ou une modification dans une base de données.

3.3.2 Function App

Une Function App, ou application de fonctions, regroupe **une ou plusieurs Azure Functions** dans un même environnement. Elle permet d'héberger l'exécution de ces fonctions et de les regrouper en une unité logique pour faciliter la gestion, le déploiement et le partage des ressources. En effet, la configuration, la connexion et l'autorisation au niveau de cette Function App peuvent être appliquées uniformément à l'ensemble des fonctions qui composent cette application.

De plus, depuis cette Function App, il est possible de suivre en détail chaque fonction grâce aux outils intégrés pour gérer et surveiller leur fonctionnement au sein de l'application.

3.3.3 Function Chaining

Ce modèle de conception est particulièrement utile pour les tâches complexes, car il permet de **répartir les différentes étapes d'un processus entre plusieurs fonctions**, chacune jouant un rôle précis. Chaque fonction est modulaire et interchangeable, ce qui facilite l'évolution du système. Dans un modèle de fonctions chaînées classique, chaque fonction est déclenchée **successivement**, souvent avec la **sortie de l'une devenant l'entrée de l'autre**. Cependant, la gestion de l'état entre les fonctions et la coordination des workflows doivent être implémentées manuellement, ce qui peut devenir compliqué pour des processus nécessitant des étapes multiples.

Pour résoudre ces problèmes, Azure propose les **Durable Functions**, des fonctions avec état ce qui permet à l'orchestrateur de reprendre exactement là où il s'est arrêté en cas d'interruption. Une utilisation basique des Durable Functions comprend trois types de fonctions :

- **Fonction orchestrateur** : décrit un workflow qui orchestre d'autres fonctions. Elle gère l'ordre d'exécution des fonctions d'activité et maintient l'état tout au long du processus.
- **Fonction d'activité** : appelée par la fonction orchestrateur, elle exécute une tâche spécifique. Chaque fonction d'activité est indépendante et peut être réutilisée dans différents workflows.

- **Fonction client** : une Azure Function régulière qui démarre la fonction orchestrateur. Elle peut être déclenchée par divers événements, par exemple une requête HTTP, pour initier le workflow.

3.4 Notion

Ce **logiciel de productivité**, combinant des fonctionnalités de **planification**, de **gestion de projets**, de **prise de notes** et de **collaboration**, m'a été très utile tout au long de mon stage. Mon binôme de stage, Théo PLEBANI, et moi-même l'avons utilisé de notre propre initiative pour assurer une collaboration efficace et optimiser notre gestion de projets.

3.4.1 Collaboration

Nous avons utilisé **un dossier partagé** où étaient centralisés tous les **documents** relatifs à notre projet. Ce dossier contenait par exemple les **tâches** à effectuer, les **comptes rendus** de chaque réunion, les **documentations** du travail réalisé et la partie **POC** (preuve de concept) avec toutes nos **recherches**. Cela a grandement facilité notre collaboration, en nous assurant d'avoir un **accès commun** et **instantané** à toutes les informations importantes, permettant ainsi une gestion fluide et organisée de notre travail.

3.4.2 Planification et répartition des tâches

La répartition des tâches dans Notion est simplifiée grâce à ses fonctionnalités de **gestion collaborative**. Il est possible d'assigner des tâches aux membres de l'équipe, définir des priorités, et suivre l'avancement de chaque tâche en temps réel.

Lors de ce stage, nous avons utilisé les fonctionnalités de répartition des tâches suivantes :

- **Assignation de Tâches** : Assignation des tâches spécifiques aux membres de l'équipe avec des dates et heures d'échéance, le projet associé et le type de la tâche.
- **Tableaux Kanban** : Visualisation du flux de travail et du statut des tâches, de "À faire" à "Terminé".
- **Chronologie** : Visualisation de toutes les tâches sous forme de ligne du temps avec les dates d'échéance, similaire à un diagramme de Gantt.
- **Commentaires et Collaboration** : Ajout de commentaires et descriptions sur les tâches pour la communication en temps réel.

3.5 OpenSearch

OpenSearch, un outil Open Source développé par Amazon Web Services (AWS), est une famille de logiciels composé d'un **moteur de recherche** et de **tableaux de bord**. OpenSearch permet l'**analyse** et la **recherche distribuée** de données, ce qui signifie que l'on peut exécuter des analyses de données et des recherches sur plusieurs ordinateurs/serveurs (appelés nœuds) de manière **coordonnée**. Cet outil permet donc aux utilisateurs de **stocker**, **indexer**, **rechercher** et **analyser** efficacement des données à grande échelle, tout en offrant des fonctionnalités avancées

de **sécurité** et de gestion des **performances**. Sa capacité à gérer l'indexation en temps réel et à exécuter des requêtes complexes le rend idéal pour :

- L'analyse de logs
- Le monitoring des infrastructures
- La visualisation des données avec des outils comme Kibana

3.5.1 Architecture et Composants

3.5.1.1 Clusters et Nœuds

OpenSearch fonctionne **en cluster**, ce qui permet de distribuer la charge de travail sur **plusieurs nœuds pour gérer de grandes quantités de données et de requêtes simultanées**. Un cluster peut comprendre un ou plusieurs nœuds, configurés en tant que nœuds maîtres, nœuds de données ou nœuds de coordination :

- **Nœud** : Une instance du logiciel OpenSearch fonctionnant sur une machine ou un serveur.
- **Cluster** : Un ensemble de nœuds.
- **Nœuds maîtres** : Coordonnent les opérations du cluster.
- **Nœuds de données** : Stockent les données.
- **Nœuds de coordination** : Gèrent le routage des requêtes.

Dans le cas d'un cluster à nœud unique tout doit être géré par celui-ci. Mais il existe de nombreux avantages à avoir un **Cluster à Plusieurs Nœuds** :

- **Capacité de stockage augmentée** : Chaque nœud supplémentaire augmente la capacité totale de stockage, permettant de gérer des ensembles de données plus volumineux.
- **Tolérance aux pannes** : Si un nœud tombe en panne, les autres nœuds peuvent prendre le relais, assurant ainsi la continuité des opérations et la disponibilité des données.
- **Répartition de la charge de travail** : La distribution des données et des requêtes sur plusieurs nœuds permet de réduire la charge sur chaque nœud individuel, entraînant des temps de réponse plus rapides et une meilleure utilisation des ressources.
- **Scalabilité** : Il est possible de gérer des volumes de données plus importants et traiter un plus grand nombre de requêtes simultanées, améliorant ainsi la performance globale.
- **Optimisation des ressources** : Différents nœuds peuvent être optimisés pour différentes tâches, par exemple comme énoncé précédemment, certains pour le stockage de données et d'autres pour le traitement des requêtes, améliorant ainsi l'efficacité du cluster.

3.5.1.2 Indexation et Recherche

Les données sont stockées dans des **index**, qui sont des **collections de documents**. Un **document** est une unité de données au format **JSON** stockant des informations (texte, données structurées). Un **index** est similaire à une **table** dans une base de données traditionnelle et un **document** correspond à une **ligne de cette table**.

Par exemple, dans une base de données d'étudiants, un **document** peut être un **étudiant** individuel incluant ses détails tel que son nom, sa moyenne, son adresse électronique… Un **index** représente, dans ce cas, tous **les étudiants** de la base de données. Si on définit l'index "etudiants" et que l'on cherche des informations sur les étudiants nous devons renseigner cet index. En effet lorsque l'on recherche des informations, on interroge les données contenues dans un index.

De plus, les index sont divisés en **shards**, ou fragments, pour permettre la **distribution** et la **parallélisation** des opérations, comme illustré dans la ***Figure 5.*** Ces **shards** sont distribués uniformément sur les nœuds d'un cluster. Par exemple, pour un index de 200 Go sur un cluster comportant 4 nœuds, l'index peut être divisé en 4 partitions (shards), chacune de 50 Go.

La ***Figure 6*** présente un exemple avec deux index : Index 1 et Index 2. Les shards de ces deux index sont répartis sur deux nœuds, Node 1 et Node 2, du cluster OpenSearch existant. L'Index 1, représenté dans la ***Figure 5***, est divisé en 2 shards, tandis que l'Index 2 est divisé en 4 shards.

Il faut également savoir que ces shards peuvent être des shards **primaires** (originaux) ou des shards **répliques** (copies).

- **Shards primaires** : Stockent les données originales.
- **Shards répliques** : Fournissent une redondance pour la tolérance aux pannes et améliorent les performances des requêtes de recherche.

Ces réplicas servent de **sauvegarde** en cas de défaillance d'un nœud. Ils sont distribués sur des nœuds différents des nœuds primaires correspondants, ceci est illustré dans la ***Figure 7***. L'avantage de cette distribution est qu'elle améliore la vitesse à laquelle le cluster traite les différentes requêtes. Il est également possible d'avoir plus d'un réplica par index pour mieux gérer la charge de travail, augmentant ainsi la disponibilité et la résilience du système.

3.5.1.3 API RESTful

OpenSearch expose une **API RESTful** pour interagir avec les clusters via des méthodes **HTTP** (GET, POST, PUT, DELETE). Cette API prend en charge des opérations telles que :

- L'**indexation** : Ajouter de nouveaux documents aux index
- La **recherche** : Effectuer des requêtes pour retrouver des documents
- La **suppression** : Supprimer des documents ou des index entiers
- La **gestion des index** : Créer, modifier et supprimer des index

Ces méthodes sont envoyées avec des **corps JSON** définissant les opérations et les données à manipuler. Un corps JSON peut être :

```
{
   "name": "Toto",
   "email": "toto@example.com",
   "age": 29
}
```

Puis, la **recherche** de données utilise une **syntaxe JSON** pour formuler des requêtes, filtrer les résultats et appliquer des agrégations pour l'analyse de données. Les données à insérer doivent donc être au format JSON :

```
{
    "query": {
        "match": {
            "name": "Toto"
        }
    }
}
```

Dans le cadre des outils DevTools, la **Console OpenSearch** permet d'exécuter des requêtes RESTful directement sur le cluster OpenSearch. Cela facilite l'**Exploration des données**, la **Gestion des index** et le **Débogage et analyse**.

3.6 Grafana

Cet outil open-source permet la **visualisation** de données et la **surveillance** via des tableaux de bord. Ces tableaux de bord sont **interactifs**, composés de différents **panels dynamiques**, et peuvent être créés à partir de **diverses sources de données**. En effet, il est possible de se connecter à différentes bases de données telles que des bases de données SQL comme MySQL, des systèmes de séries temporelles comme InfluxDB, des systèmes de logs comme Elasticsearch, et bien d'autres.

Un des avantages majeurs de cet outil est la **mise à jour en temps réel des tableaux de bord**, qui se **synchronisent** directement avec la base de données. Pour afficher les données souhaitées sous forme de graphiques, il est nécessaire de les interroger en écrivant des requêtes spécifiques au langage de la source de données. Par exemple, on utilisera du SQL si la source est une base de données SQL.

Un autre élément utile est le **système d'alertes et de notifications**. Il est possible de configurer des alertes basées sur des seuils prédéfinis par exemple. Ainsi, lorsqu'un événement défini dans une alerte se produit, une notification peut être envoyée directement par courriel ou via d'autres outils.

4 Études et Recherches Préliminaires

Cette section vise à décrire les méthodes utilisées pour recueillir toutes les informations nécessaires et exploitables avant d'accéder à tous les outils requis pour la phase de développement.

4.1 Recueil des besoins

La première étape, essentielle, consistait à bien comprendre notre **rôle** au sein de l'entreprise et plus précisément les **besoins** spécifiques du **groupe** dans lequel nous avons été accueillis. Avant notre premier jour, nous n'avions pas une vision claire des enjeux de notre stage. Ce n'est que lorsque notre maître de stage, Mme Audrey BRIFFOD, a commencé à nous expliquer notre mission et leurs attentes au sein du groupe RPA que nous avons réellement saisi notre rôle.

Nous avons immédiatement pris contact avec plusieurs membres du groupe recommandés par Mme BRIFFOD, et dès le lendemain, nous avons organisé une réunion. L'objectif était d'**analyser** et d'**identifier** les besoins spécifiques du projet, notamment ce qu'ils souhaitaient voir sur l'application en termes d'informations affichées sur un **tableau de bord**. Cela inclut les données qu'ils utilisent régulièrement et qui leur seraient utiles pour améliorer leur **efficacité** au travail.

4.2 Identification des attentes

Grâce aux échanges avec le groupe RPA nous avons pu concrètement identifier l'objectif du projet et leurs besoins. Ainsi l'**objectif principal** est de leur fournir une plateforme permettant la surveillance centralisée des environnements UiPath de leurs clients.

4.2.1 Besoins primaires

Nous avons tout d'abord relevé les besoins les plus importants, à entreprendre en premier :

- Visualiser les Jobs en erreur avec une description rapide de la cause de l'échec.
- Être informé lorsqu'un Job est en erreur en envoyant une notification par courriel à l'aide d'un WebHook.
- Afficher sur un tableau de bord les statistiques les plus importantes, telles que les pourcentages de réussite/échec.

4.2.2 Besoins secondaires

Ensuite d'autres besoins, bien que moins urgents, ont également été mentionnés :

- Visualiser les détails des journaux d'erreurs (logs) pour un Job avec le pourcentage de transactions échouées/réussies.
- Sélectionner une organisation correspondant à un client et voir les détails des processus:
    - Filtrer par années/mois pour voir les succès/échecs pour les périodes souhaitées.
    - Filtrer par état : échecs ou réussites.

4.3 Proof of concepts (POC)

Une fois les besoins clairement définis, les recherches ont débuté par l'accès à l'API de l'Orchestrator UiPath et l'identification des données à récupérer. Puis, il a été constaté que l'utilisation des WebHooks constituait une solution efficace pour automatiser la récupération des informations nécessaires et pour envoyer des notifications aux parties concernées. Ces deux aspects, que nous allons explorer maintenant, étaient donc essentiels pour la réussite du projet.

4.3.1 Requêtes API

Lors de mes premières recherches sur l'API Orchestrator, j'ai découvert qu'il était possible d'utiliser l'outil **SWAGGER** pour tester l'API. Pour ce faire, il suffit de se connecter à l'Orchestrator du client sur internet et d'ajouter "SWAGGER/index.html" à l'URL. Par exemple : https://cloud.uipath.com/{orchestratorName}/{tenantName}/orchestrator_/SWAGGER/index.html.

4.3.1.1 Obtention du Token d'Authentification

La première étape consistait à **récupérer le token** nécessaire pour effectuer n'importe quelle requête par la suite. Swagger a la particularité d'en générer un directement, ce qui facilite grandement l'authentification et l'accès aux différentes fonctionnalités de l'API. Toutefois, nous souhaitions découvrir comment le récupérer pour pouvoir réaliser les requêtes ultérieurement en ligne de code. Pour cela, nous avons consulté la documentation de l'API en ligne pour trouver celle appropriée. Celle-ci nécessite des données spécifiques relatives aux clients : **Client ID et User Key**. Malheureusement, ces données doivent être récupérées **manuellement** pour chaque client directement dans l'**application UiPath**, car il n'existe aucun moyen d'y avoir accès via une requête API. Face à cette difficulté, nous avons envisagé, en accord avec notre maître de stage, une solution temporaire : **stocker ces informations dans une base de données** qui sera alimentée à chaque nouvel ajout de client. Étant donné qu'il y a environ une dizaine de clients, cette tâche ne prendra finalement pas beaucoup de temps.

4.3.1.2 Utilisation des Endpoints pour la surveillance des Jobs

Après une exploration approfondie des Endpoint existants, nous en avons retenu deux :

***Requête 1** : États des Jobs*

La première requête permet de connaître **l'état d'un Job**, qu'il soit réussi, en pause, en erreur ou dans un autre état, ainsi que le **nom du processus** et des **informations succinctes** expliquant son état actuel. Cette requête utilise l'URL suivante et nécessite d'avoir le nom de l'orchestrateur du client, le nom du tenant et l'id du Job en question :
https://cloud.uipath.com/{orchestratorName}/{tenantName}/orchestrator_/odata/Jobs({JobId})

***Requête 2 :** Gestion des Logs des Jobs*

Avant d'expliquer la deuxième requête utilisée, il est important de noter que les clients peuvent organiser leurs processus depuis l'Orchestrator, les plaçant dans des **dossiers** distincts. Chaque dossier contient également les Jobs associés aux processus. Ainsi, pour obtenir des informations,

comme les logs d'un Job, il est nécessaire de spécifier le dossier dans lequel il est situé. La **clé** identifiant ce dossier est donc nécessaire afin d'effectuer une requête visant à récupérer les logs. Cette clé doit être placée dans les headers de la requête au niveau du champ «X-UIPATH-OrganizationUnitId ».

Pour effectuer cette deuxième requête il est nécessaire d'avoir le nom de l'orchestrateur du client, le nom du tenant, le token, la clé associée au Job et la clé associée au dossier.
Voici l'URL de la requête :
https://cloud.uipath.com/{orchestratorName}/{tenantName}/orchestrator_/odata/RobotLogs?$filter=JobKey eq {JobKey}&$orderby=TimeStamp asc

4.3.1.3 Script Python pour Automatiser les Requêtes et Calculs Statistiques

Nous avons ensuite développé un **script Python** qui utilise ces Endpoints, depuis la récupération du token jusqu'au calcul de statistiques. Ce script effectue des requêtes API pour récupérer les transactions d'un Job avec leurs différents niveaux : Info et Error avec le type d'exception.

Implémentation du Script

Nous avons testé ce script avec un Job spécifique qui avait le statut « Successful ». Ce choix était stratégique car ce Job comprenait des transactions réussies ainsi que des exceptions telles que Business Rule Exception et System Exception, offrant ainsi un ensemble représentatif des différents scénarios d'erreurs possibles.

Analyse Statistique

Théo s'est chargé de la mise en place des requêtes précédentes tandis que je me suis concentrée sur la partie statistique. Cette dernière a pour objectif de calculer les pourcentages de transactions réussies et échouées, ainsi que les détails des pourcentages d'exceptions rencontrées.
Ces informations sont essentielles pour vérifier rapidement si tout s'est bien passé ou non lors de l'exécution du processus grâce à une vue d'ensemble plus précise et complète.
De plus, ces statistiques mettent en avant le fait qu'un Job peut réussir tout en rencontrant certaines erreurs au cours de certaines transactions. Bien que ces erreurs ne nécessitent pas une résolution immédiate, elles doivent tout de même être traitées.

4.3.2 WebHooks

L'envoi de notifications relatives aux erreurs d'exécution des processus est indispensable. Pour rappel, le groupe n'est pas informé directement en cas d'échec d'un Job ; les clients doivent signaler les problèmes eux-mêmes. Pour remédier à cela, lors de notre première réunion avec le groupe RPA, un membre nous a parlé des WebHooks, intégrés directement à l'Orchestrator.

Configuration des WebHooks dans l'Orchestrator UiPath

Lorsque nous accédons à l'interface web de l'Orchestrator, il suffit de se rendre dans l'onglet approprié et de renseigner les informations suivantes :

- **Nom** : Utile pour différencier les WebHooks dans l'interface web de l'Orchestrateur si nous souhaitons en créer plusieurs.
- **Description** : Pour indiquer l'objectif du WebHook.
- **URL** : L'URL du serveur sur laquelle nous voulons que les données soient envoyées. Dans notre cas, ce WebHook pointe vers l'URL de la fonction trigger HTTP de Microsoft Azure. Chaque WebHook client est associé à cette même URL pour la réception des événements.
- **Secret** : Utilisé, dans notre cas, pour identifier l'origine du WebHook envoyé. Chaque tenant dispose d'un WebHook avec un Secret unique, permettant de différencier les sources des événements reçus après déchiffrement du champ "Secret", où le nom du client et du tenant est enregistré.
- **Event Type** : Permet de choisir les événements pour lesquels nous voulons être informés. Par exemple, cela inclut les actions liées aux robots ainsi que les événements concernant les processus (création, modification, suppression…).

Utilisation des WebHooks pour les Notifications de Jobs

Notre objectif ici était d'établir un WebHook spécifiquement dédié aux événements liés aux Jobs. Cela implique que chaque fois qu'un Job échoue, réussit ou est arrêté, le WebHook initie le processus d'envoi automatique d'alertes par courriel aux personnes concernées, avec les détails affichés directement sur un Dashboard de la plateforme. Cette configuration permettra au groupe d'être informé en temps réel des incidents, facilitant ainsi une intervention rapide et efficace.

4.4 Managed Services Platform

En parallèle du recueil des besoins et de la récupération des informations nécessaires, nous avons collaboré avec une autre équipe. Dès le début du stage, nous avons appris qu'il existait déjà une plateforme appelée **Managed Services Platform** et que notre objectif était de la compléter en y intégrant les fonctionnalités nécessaires pour le groupe RPA, associées à UiPath.

Une réunion a été rapidement planifiée avec M. Koen WARSON, l'un des responsables de l'outil Managed Services Platform, et M. Frank VANDEBERGH, chargé de la partie technique. Celle-ci avait pour objectif d'exposer les besoins du groupe RPA et de découvrir leur plateforme.

A l'issu de cette présentation, nous leur avons fournis une documentation regroupant :

- L'**objectif** précis de cet ajout dans la plateforme existante.
- Nos **besoins en termes d'API**, incluant nos recherches concernant la récupération du token, les endpoints nécessaires et les WebHooks.
- Les **besoins de la plateforme**, spécifiant ce que nous voulons afficher sur le tableau de bord final ainsi que les besoins en termes de notifications, d'alertes.

Ce document a été relu par Mme Audrey BRIFFOD et un autre membre du groupe RPA. Nous avons pu faire un point pour vérifier la **clarté** et la **compréhension** des attentes, bénéficiant ainsi à toute l'équipe : les responsables de la plateforme et nous-même.

Par la suite, une deuxième réunion avec M. Frank VANDEBERGH a confirmé que nos recherches liées à l'API fonctionnaient de leur côté. Mais il ne voyait pas encore comment réaliser la partie sur la récupération des logs. Comme nous avions déjà effectué des recherches sur ce point, nous avons pu lui partager notre méthode sous forme d'un document détaillant le processus en question et d'un script Python l'exécutant. Ce script est mentionné dans la partie **Requêtes API**.

Ce processus, crucial pour comprendre et corriger les erreurs, comprend les étapes suivantes :

1. Récupérer les données nécessaires envoyées par **WebHook** lorsqu'un Job réussit ou échoue, telles que : Job Id, Job Key et Folder Id.
2. Effectuer une requête pour obtenir les **détails du Job** : Status et Info.
3. Effectuer une requête pour récupérer les **logs du Job**, incluant le niveau de chaque transaction du log (Info, Warning, Error) ainsi qu'un message détaillé associé.

Nous avons initialement pensé que M. VANDEBERGH voulait simplement vérifier la faisabilité de nos projets futurs en réalisant des **tests** de son côté. Cependant, le 5 juin, il nous a présenté un document détaillant son intention de réaliser entièrement le projet sans notre intervention. Après avoir clarifié la situation avec notre maître de stage, nous n'avons pu reprendre contact avec lui le 18 juin, en raison du temps nécessaire à la **mise en place de son côté et de ses obligations**.

La mise en place d'un environnement accessible pour nous s'est révélés complexe car la plateforme ne peut pas être exécutée localement. Elle nécessite un déploiement partiel sur Amazon Web Services et Azure. La plateforme comprend une multitude de composants, dont la plupart ne sont pas accessibles au public pour des raisons de sécurité. Ils ont dû **configurer minutieusement l'ensemble** pour éviter toute interruption du fonctionnement existant.

M. VANDEBERGH a donc configuré l'infrastructure nécessaire pour intégrer notre script Python, que j'avais traduit en Node.js pour assurer sa compatibilité avec leur environnement. Cette configuration était destinée à nous permettre de nous concentrer sur le traitement des événements provenant de UiPath et à tester nos solutions.

À partir du 18 juin, une fois leur infrastructure opérationnelle, nous avons organisé des réunions quotidiennes en anglais pour présenter notre progression, poser des questions et résoudre nos problèmes. Progressivement, nous avons obtenu l'accès aux outils nécessaires, commençant par Azure, puis par la base de données OpenSearch et Grafana pour la création de tableaux de bord.

5 Mise en œuvre

Après avoir recueilli toutes les informations nécessaires et obtenu l'accès aux outils requis, nous entamons la mise en œuvre concrète du projet.

5.1 Étape 1 : Microsoft azure

Avant d'avoir accès à la plateforme Managed Services Platform, nous avons été informés que nous devions utiliser Microsoft Azure pour déployer notre code afin qu'il fonctionne avec leur plateforme. Nous avons donc commencé à travailler de notre côté en utilisant notre portail étudiant Microsoft Azure pour faire des tests et mieux appréhender l'environnement. Nous avons eu l'occasion de créer des fonctions chaînées, une notion vue en TP à Polytech, afin de reprendre toutes les étapes du script en Node.js et de comprendre comment mettre en place l'architecture.

Une fois les accès fournis, nous avions accès à la Function App où nous devions introduire le processus lié à la réception d'un WebHook. Initialement, la structure utilisée était une simple fonction Azure HTTP Trigger où nous devions ajouter tout le processus. Cependant, ayant étudié les fonctions chaînées, nous lui avons demandé si nous pouvions les utiliser à la place. Ne connaissant pas bien cette approche, il nous a demandé de lui fournir un document expliquant leur utilisation, mais a finalement accepté ce choix qui l'intéressait car il pourrait potentiellement utiliser ce pattern pour de futurs projets.

Théo a donc pu mettre en place l'architecture souhaitée que nous avions déjà testée lors de nos premiers essais.

5.1.1 Structure du Workflow

La vue d'ensemble, illustrée dans la ***Figure 8***, détaille les différentes étapes de notre processus visant à récupérer des informations sur un Job. Cette structure correspond à celle de base d'une Durable Function.

Tout commence avec le **WebHook** configuré sur UiPath Orchestrator, qui nous alerte principalement lorsqu'une exécution de processus échoue. L'objectif de ce WebHook est de transmettre les données du Job, dataWebHook, à la fonction **trigger HTTP DurableFunctionsHttpStart**. Concrètement, une requête HTTP initie le processus. Lorsque cette requête HTTP POST est déclenchée, via l'URL de la fonction, par un évènement associé au WebHook et que cet évènement est reçu par la fonction, le client Durable Function est instancié, **lançant ainsi une nouvelle orchestration** avec les données reçues. C'est à ce stade qu'un lien est généré, permettant au client de suivre l'état de l'orchestration en temps réel.

Ensuite, **l'Orchestrator Function,** déclenchée par la HTTP Trigger Function **DurableFunctionsHttpStart**, orchestre la séquence des tâches à accomplir. Elle appelle

plusieurs **Activity Functions** dans un ordre précis et **attend les résultats** de celles-ci avant de passer à l'étape suivante. Notre workflow est composé de quatre **Activity Functions** :

1. FindOrganisation : Cette fonction d'activité est appelée en premier afin d'identifier le client (Customer) et le tenant (Instance) à l'origine du WebHook. Les données relatives à celui-ci permettent d'identifier l'émetteur à partir du déchiffrage de la clé secrète. Une fois les informations sur l'organisation récupérées, elles sont transmises à la fonction suivante : dataOrganization.

2. GenerateTokenAPI : L'objectif est de récupérer le token d'authentification en effectuant une requête sur l'API Orchestrator à partir des informations récupérées depuis la fonction FindOrganisation. Le token, utile pour toutes les autres requêtes, est ensuite retourné par cette fonction : accessToken.

3. GetJobDetails : Cette troisième étape correspond à la récupération des détails du job. Cela inclut son état (Successful, Faulted…), de brèves précisions sur la raison de l'état et le nom du processus via à l'API Orchestrator. Cette fonction reprend le même fonctionnement que le script de la section 'Script Python pour Automatiser les Requêtes et Calculs Statistiques' concernant la récupération des détails.

4. GetJobLogs : Cette fonction récupère les logs des robots correspondant aux transactions du Job, fournissant un historique détaillé de toutes les actions et les éventuelles erreurs. Le déroulement suit également le même principe que celui détaillé dans le script Python pour la récupération des logs.

Théo s'est concentré sur le déploiement des fonctions **FindOrganisation** et **GenerateTokenAPI**, tandis que j'ai réalisé les fonctions **GetJobDetails** et **GetJobLogs**. Ces dernières sont lancées en parallèle car elles nécessitent seulement les données du token et du WebHook.

5.1.2 Interface Azure

Du côté de la plateforme Azure, notre **Function App** appelée **ptcadc-tst-uipath-webhook,** détaillée ci-dessus, est composée de nombreuses Azure Functions visibles dans l'interface illustrée dans la ***Figure 9***. Nous pouvons y voir le type de chaque fonction : HTTP, Orchestrator et Activity. Il est également possible de consulter le code de celles-ci. L'utilité principale de cette interface est de vérifier que les fonctions ont bien été déployées et de suivre l'affichage des logs (partie Log Stream) de tout le processus d'enchaînement afin de s'assurer que tout se passe bien.

La prochaine étape consiste à stocker les données des Jobs pour les visualiser par la suite.

5.2 Étape 2 : OpenSearch

L'outil utilisé pour le stockage de données est OpenSearch, présenté dans une des parties précédentes : OpenSearch.

5.2.1 Script intégration des données

Nous utilisons un script Node.js, non conçu comme une Azure Function, mais intégré et utilisé dans la fonction d'orchestration Azure. Ce script est spécifiquement conçu pour insérer des

données dans OpenSearch en utilisant le package **@opensearch-project/opensearch**, qui permet d'interagir avec cet outil. Plutôt que d'utiliser un navigateur et risquer d'exposer les données au public, ce package offre un moyen plus sûr d'envoyer des requêtes aux clusters.

Voici les différentes étapes du script :

1. **Initialisation du client OpenSearch :** Il est nécessaire de configurer un client pour se connecter au cluster OpenSearch. Ce cluster est associé à une URL, celle renseignée ici est ELASTIC_URL, une variable d'environnement configurée que nous ne connaissons pas pour des raisons de sécurité. Ce client sera utilisé pour toutes les interactions avec le cluster.

```
const elasticClient = new Client({ node: process.env.ELASTIC_URL });
```

2. **Création d'une fonction asynchrone ElasticIngest** à appeler lorsque l'on souhaite insérer des données. Elle prend en paramètre **data** et **index: ElasticIngest(data,index)**, c'est à dire les **données à ingérer** et le **nom de l'index** OpenSearch où les données seront insérées. Dans cette fonction nous retrouvons :

    1. Définition d'une fonction **onDocument** qui retourne un objet au format JSON indiquant l'index, celui indiqué en paramètre, pour chaque document à ingérer. Cet objet respecte alors le format requis par la base de données OpenSearch.

```
const onDocument = () => ({ index: { _index: index } });
```

    2. Ingestion des données en utilisant la méthode **helpers.bulk** du client OpenSearch pour ingérer les données **data** en utilisant la fonction onDocument afin de déterminer où et de quelle manière le stocker.

```
const result = await elasticClient.helpers.bulk({
      datasource: data,
      onDocument,
    });
```

5.2.2 Insertion de données

Nous avons décidé d'effectuer un seul appel à la base de données. À la suite de la séquence d'exécution des différentes fonctions dans la fonction d'orchestration nous insérons directement toutes les données nécessaires à partir des informations récupérées par ces fonctions.

```
data = [
        {
           Timestamp: data_tokenAPI.timestamp,
           Job_Id: data_tokenAPI.jobId,
           Client: data_tokenAPI.organization["customerName"],
           Tenant: data_tokenAPI.instance["instanceName"],
           Status: data_jobDetails["state"],
           Process_Name: data_jobDetails["process_name"],
           Message: data_jobDetails["info"],
           Transactions: data_jobLogs
        }
]

ElasticIngest(data,'uipath-jobs-logs');
```

5.2.3 Vérification insertion des données

Pour vérifier le bon fonctionnement, il suffit d'accéder à l'interface web d'OpenSearch, notamment à l'outil DevTool mentionnée dans la partie API RESTful.

Par exemple, pour vérifier l'insertion correcte de nos données, nous pouvons exécuter la requête suivante dans la console :

```
GET uipath-jobs-logs/_search
{
  "query": {
    "match_all": {}
  }
}
```

Cette requête nous permet de récupérer l'ensemble des données associées à l'index `uipath-jobs-logs`.

Il est également possible de filtrer les résultats. Par exemple, pour récupérer seulement les jobs qui sont arrêtés (stopped), on peut utiliser la requête suivante :

```
GET uipath-jobs-logs/_search
{
  "query": {
    "match": {
      "status": "stopped"
    }
  }
}
```

Le type de requête GET nous intéresse particulièrement, car notre objectif est de vérifier que les données ont été correctement insérées. Cependant, nous pouvons également effectuer des tests en utilisant la requête POST pour ajouter des simulations de Job et les afficher ensuite sur le tableau de bord Grafana.

5.3 Étape 3 : Grafana

5.3.1 Dashboard

Une partie du projet, encore en développement, consiste à créer un tableau de bord. En tant que novice avec cet outil, j'ai d'abord dû en apprendre le fonctionnement. Bien que je ne le maîtrise pas complètement, j'ai pu accomplir plusieurs tâches.

J'ai réussi à créer des **filtres** associés à des variables pour sélectionner un client, un tenant, une transaction et l'état du Job d'intérêt. Pour créer une variable, il faut sélectionner la source de données et effectuer une requête dans le langage correspondant. Par exemple, nous utilisons ElasticSearch, et la requête pour récupérer tous les clients est : **{"find": "terms", "field": "Client.keyword"}**.

J'ai également découvert qu'il est possible de lier deux variables. Plusieurs tenants sont associés à l'orchestrateur d'un client. Lors de la création du filtre pour les tenants, j'ai lié cette variable à celle du client pour n'afficher que les tenants liés au client sélectionné, avec la requête : **{"find": "terms", "field": "Tenant.keyword", "query": "Client: '$Client'"}.**

Un autre filtre, appelé Transactions, permet de récupérer tous les types de transactions possibles grâce à la requête : **{"find": "terms", "field": "Transactions.transaction_status.keyword"}.** Cela nous permet de sélectionner les transactions à observer, telles que celles du type Successful, Business Exception, et System Exception.

Une seconde tâche consistait à **configurer différents types de panels**. Les panels utilisés jusqu'à présent incluent : Table et Donut Chart. Pour récupérer les données nécessaires à chaque panel, il faut sélectionner une source de données et spécifier une requête dans le langage adapté. Si aucune requête n'est spécifiée, toutes les données restent statiques. Pour rendre le graphique interactif, des filtres doivent être inclus dans la requête. Par exemple, pour filtrer une table en fonction du client sélectionné, il faut préciser dans le champ 'Query' du graphique : Client:$Client_filter. Ici, "Client" correspond au nom du champ de la table suivi du nom du filtre Client_filter.

Pour finir, j'ai commencé à utiliser diverses **transformations**, configurable lors de l'édition du panel. Elles incluent la suppression de certains champs que nous ne souhaitons pas afficher ainsi que des calculs statistiques pour les Donut Chart.

5.3.2 Alertes

Il est possible de configurer des alertes dans Grafana pour informer l'équipe RPA des problèmes liés aux Jobs. Cette fonctionnalité est essentielle pour l'application mais pour l'instant, nous n'avons pas encore testé cette fonctionnalité. J'ai simplement commencé à examiner où les configurer et quel type d'informations il faudrait spécifier.

Voici ce que j'ai retenu des étapes à suivre lors de leurs configurations dans la section "Alerting" de l'outil Grafana :

1. Donner un nom à l'alerte.

2. Définir une **requête** et une **condition** d'alerte. C'est dans cette étape qu'il faut indiquer que l'on souhaite recevoir une alerte lorsque qu'un nouveau Job échoué apparaît sur le tableau de bord.

3. Configurer le **comportement d'évaluation** : Déterminer l'intervalle et la fréquence à laquelle Grafana évalue l'état des Jobs. Par exemple, il est possible de configurer Grafana pour évaluer l'état des Jobs toutes les 5 minutes et de spécifier combien de fois un Job peut être en état 'Échec' avant qu'une notification ne soit envoyée. Ce point nécessitera une attention particulière car, dans notre cas, nous souhaitons être alertés dès qu'un Job présente une erreur.

4. Configurer les notifications : Sélectionner les canaux de notification appropriés, tels que les courriels ou d'autres outils de communication.

5. Valider et **activer** l'alerte.

Comparons maintenant les résultats attendus à la fin du projet avec ceux affichés sur le tableau de bord actuel.

6 Résultats

6.1 Dashboard prévisionnel

Afin d'expliquer clairement nos besoins en termes de Dashboard à M. VANDEBERGH, j'ai créé un **schéma** représentant notre **vision globale de ce que nous désirions obtenir** (voir ***Figure 10***). Ceci afin de vérifier si tout ce que nous envisagions était réalisable avec l'outil.

Notre objectif est de créer **trois principaux panels** pour le moment :

1. Le premier, en haut, regrouperait toutes les **informations concernant les Jobs**. Il permettrait de visualiser directement les derniers processus exécutés.
2. Le panel "Transaction Statistics" fournira des détails sur les différents types de transactions, ce qui est particulièrement utile lorsque vous sélectionnez une ligne dans le premier panneau pour examiner les détails d'un job spécifique. Les pourcentages associés à chaque type permettront de voir rapidement si le processus s'est déroulé correctement ou s'il a rencontré des erreurs lors des transactions.
3. Le dernier panel donnera des **détails supplémentaires sur ces transactions**, précisant les **lignes** où les erreurs se sont produites. Cela facilitera l'accès direct au processus concerné par la transaction échouée, simplifiant ainsi la résolution des problèmes.

De plus, nous retrouvons les filtres en haut de la page comme mentionnés dans la section **Dashboard**. Ainsi qu'un filtre permettant de sélectionner une période.

Il est important de souligner que nos prévisions actuelles pourront être ajustées en fonction des demandes du groupe ou des limitations techniques de l'outil.

6.2 Dashboard actuel

Actuellement, j'ai créé le tableau de bord illustré à la ***Figure 11***. Cependant, mon stage n'étant pas terminé, il manque encore quelques éléments par rapport à notre plan initial, notamment concernant les transactions.

Nous devons maintenant calculer les pourcentages des différents types de transactions et trouver une méthode pour structurer les données en deux colonnes distinctes : une pour le numéro de ligne et l'autre pour le type de transaction. Avec des jobs comportant jusqu'à 50 transactions, il devient difficile d'examiner chaque détail individuellement.

De plus, nous envisageons d'afficher uniquement la dernière exécution de chaque processus dans le troisième graphique. Si nous suivons le schéma prévu, plusieurs lignes peuvent correspondre à une même transaction numéro X sous un même nom de processus, ce qui pourrait être résolu en triant les données par date et heure ou en ajoutant la colonne Timestamp.

7 Retour d’expérience

Le stage que je réalise actuellement chez CBTW m'a offert l’opportunité de développer mes compétences dans les domaines de l’Hyperautomation, du développement, des bases de données de type "moteurs de recherche", notamment avec OpenSearch et de visualisation. Cette expérience m'a permis de plonger dans des défis techniques, tels que l'intégration d'un script Python via Node.js dans Microsoft Azure et l'adoption rapide d'outils comme Grafana et OpenSearch.

Un des défis techniques auxquels nous avons été confrontés était la nécessité de déployer nos fonctions directement sur Azure pour effectuer des tests, car nous ne pouvions pas exécuter le projet en local. Ce processus de déploiement prenait généralement au moins 5/10 minutes, ajoutant une contrainte temporelle notable à nos activités de développement.

Ensuite, la collaboration étroite avec l'équipe internationale de CBTW a été un pilier essentiel de ce stage, favorisant une communication régulière et des réunions quotidiennes en anglais pour suivre notre progression et résoudre les problèmes rencontrés.

J'ai été chaleureusement accueilli au sein de l'entreprise où règne une ambiance conviviale et une collaboration exemplaire. Notre maître de stage nous a offert l'opportunité enrichissante et novatrice de l'accompagner lors de rendez-vous clients, ce qui a été très instructif. Chacun s'est investi pour nous intégrer pleinement dans l'équipe, prenant le temps de nous présenter leurs projets en cours. Mon expérience dans cette entreprise est pour l’instant très positive, renforçant ainsi mon désir de continuer à évoluer professionnellement dans un tel environnement.

Enfin le déploiement sur Microsoft Azure a constitué une étape cruciale du projet, où j'ai pu approfondir mes connaissances en utilisant des fonctions chaînées, des Function Apps et des Durable Functions pour gérer la réception de WebHooks. Ces compétences, ainsi que ma maîtrise croissante de JavaScript, de la gestion des API et de la visualisation de données avec Grafana, représentent des réalisations significatives au cours de ce stage.

En conclusion, ce stage est une expérience enrichissante qui me permet de relever des défis techniques tout en renforçant mes compétences en développement logiciel, en gestion de projet et en collaboration internationale. Je suis enthousiaste à l'idée de continuer à contribuer au succès de ce projet dans les semaines à venir et de continuer à progresser techniquement.

8 Conclusion et Perspectives

Bilan du Projet

Le projet progresse vers l'atteinte de ses objectifs principaux : mettre en place une plateforme pour surveiller de manière centralisée les environnements UiPath des clients. Nous avons réussi à récupérer efficacement les données nécessaires depuis l'API Orchestrator grâce à l'intégration du WebHook sur Microsoft Azure. Actuellement, nous nous concentrons sur la création du tableau de bord avec Grafana, que j'ai commencé à maîtriser. Bien que la visualisation des jobs et des détails des transactions ait débuté, le système n'est pas encore complètement opérationnel. Nous prévoyons une présentation avec l'ensemble du groupe pour vérifier si cela correspond à leurs attentes et pour identifier les éventuels éléments manquants. De plus, la mise en place du système d'alerte est en cours de réflexion.

Perspectives d'Amélioration

Malgré nos avancées, des améliorations sont envisagées pour l'affichage des données sur Grafana. Nous envisageons de revoir l'insertion des données des transactions dans la base de données, étant donné les difficultés d'accès actuelles aux informations stockées dans un champ au format JSON ou tableau. Il pourrait être nécessaire de créer un second index ou d'ajuster le format d'insertion existant si aucune solution n'est trouvée. Je vais d'abord contacter M. VANDEBERGH pour voir s'il a des suggestions à proposer.

Perspectives Professionnelles

Le sujet de mon stage est très enrichissant et complet, m'offrant une immersion approfondie dans le développement web et la manipulation de données pour extraire des informations pertinentes. J'apprécie tous les aspects de ce projet, notamment la conception et la visualisation des tableaux de bord.

Après avoir échangé avec des personnes du secteur Product Design & Growth, je suis particulièrement attiré par ce domaine qui correspond parfaitement à mes intérêts pour la visualisation, le design et l'amélioration continue des produits. Je serais intéressée de réaliser un stage dans ce secteur, pourquoi pas au sein de cette même entreprise.

9 Annexes

Point de lecture associé à *Figure 1* : **1.1.1 Son histoire**

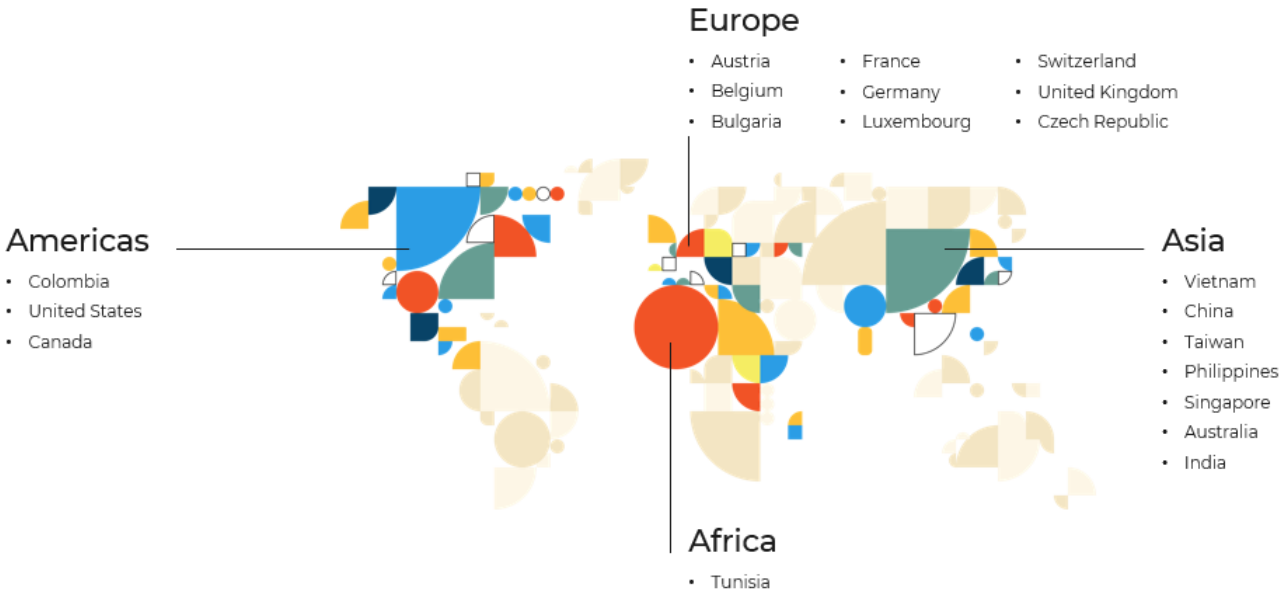


***Figure 1**: Collaborateurs à travers le monde.*

Point de lecture associé à *Figure 2* : **1.1.2 Ces services**

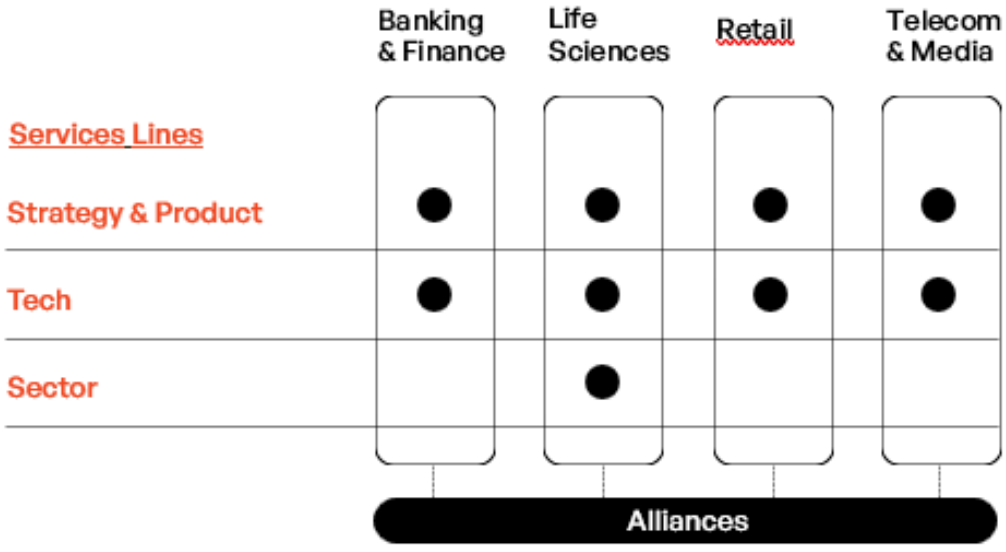


***Figure 2:** Les Services Lines et leurs alliances dans différents secteurs.*

Points de lecture associé à *Figure 3* : 1.2.1.2 RPA et **3.1** UiPath

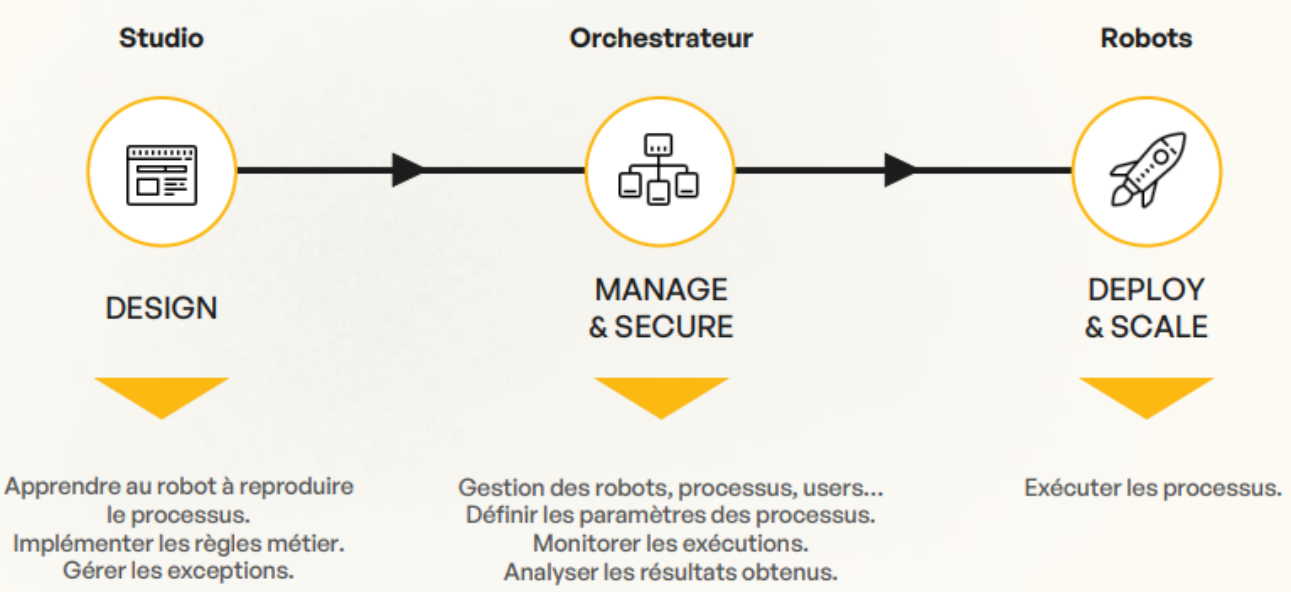


Figure 3: *Structure d'une plateforme RPA*

Point de lecture associé à *Figure 4* : **3.1.1 Notions Importantes**

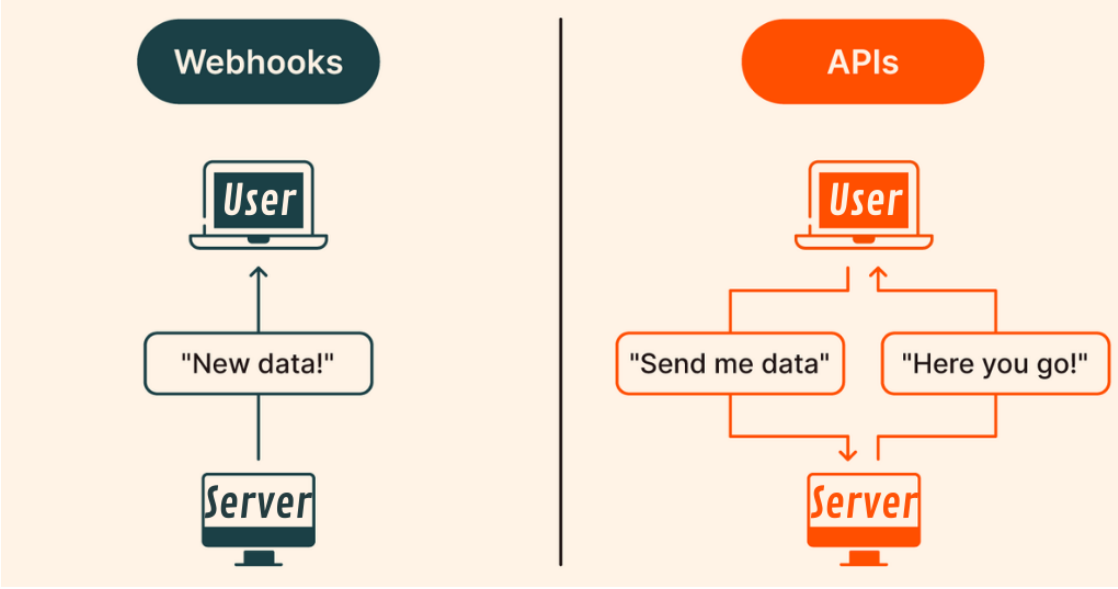


Figure 4: *Différence APIs & WebHooks*

Point de lecture associé à *Figure 5, Figure 6 et Figure 7* : 3.5.1.2 Indexation et Recherche

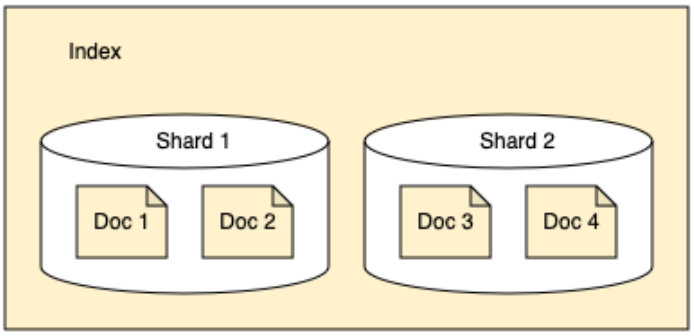


Figure 5: *Division d'un index en deux Shards comprenant des Documents différents*

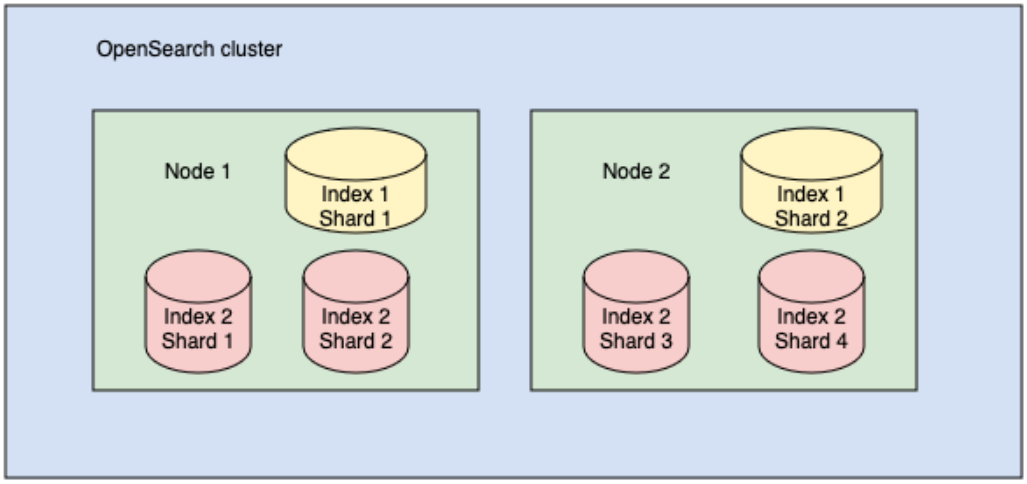


Figure 6*: Répartition des Shards dans les nœuds du cluster*

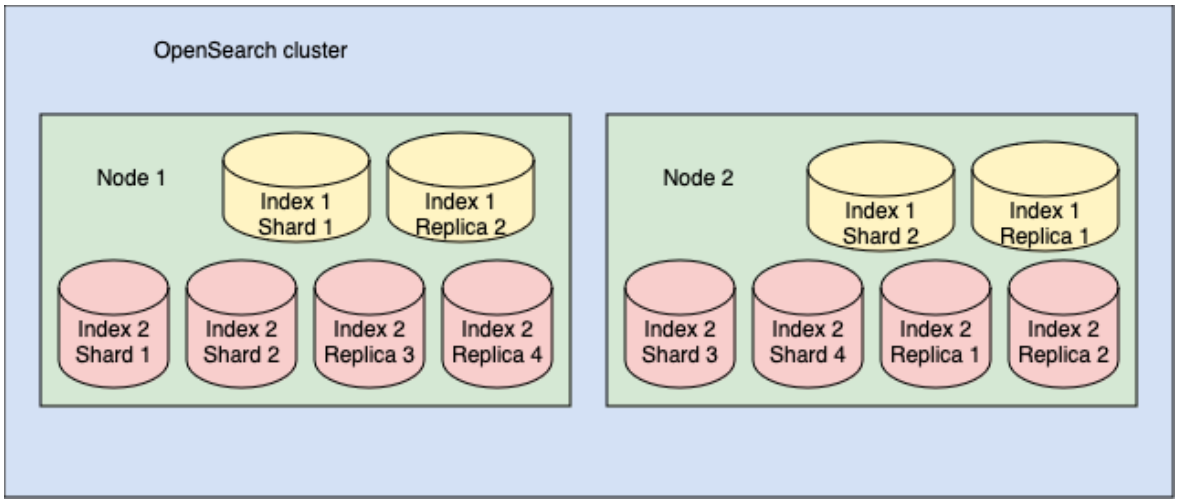


Figure 7*: Shards primaires et Réplicas*

Point de lecture associé à *Figure 8* : **5.1.1 Structure du Workflow**

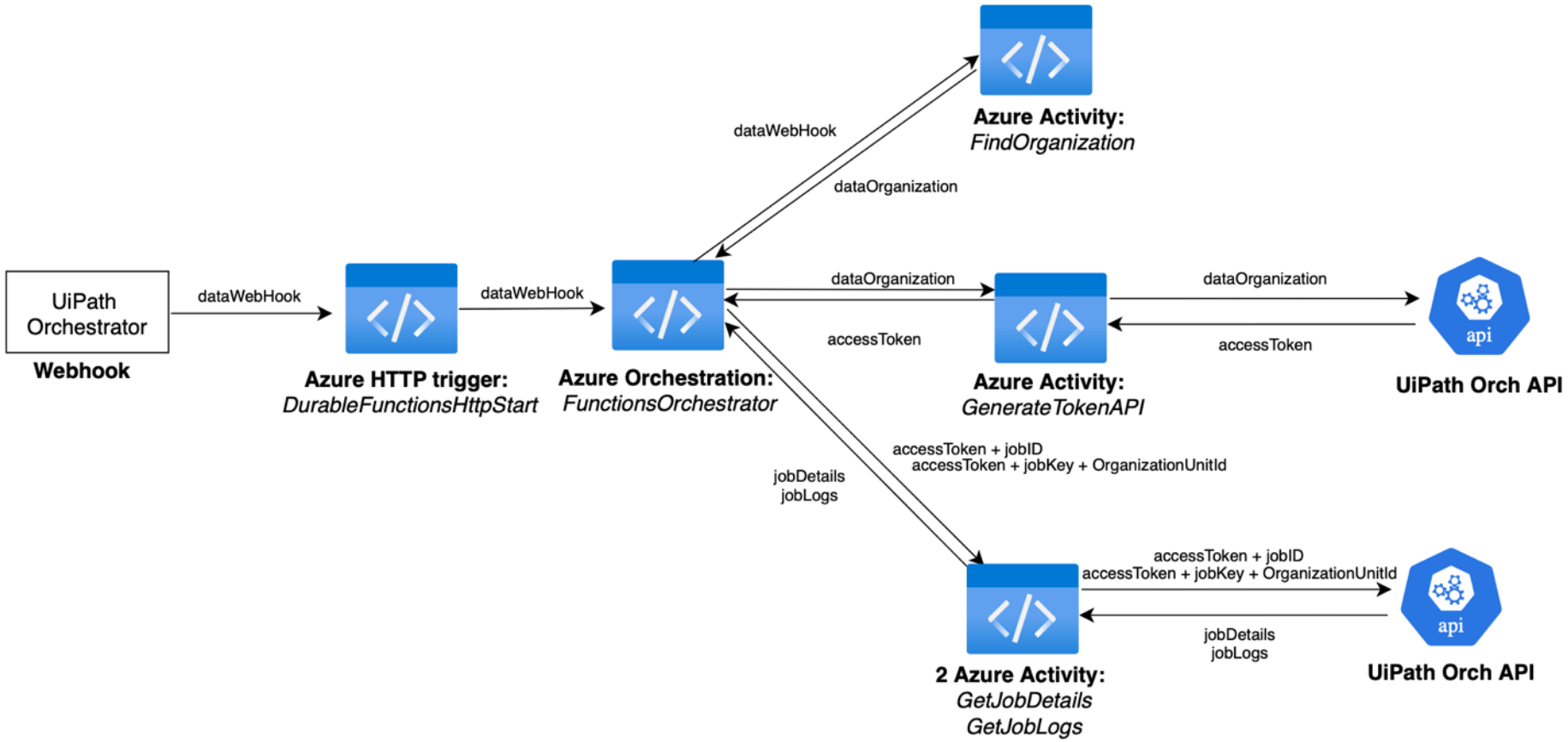


***Figure 8**: Vue d'ensemble de l'architecture mise en place*

Point de lecture associé à *Figure 9* : **5.1.2 Interface Azure**

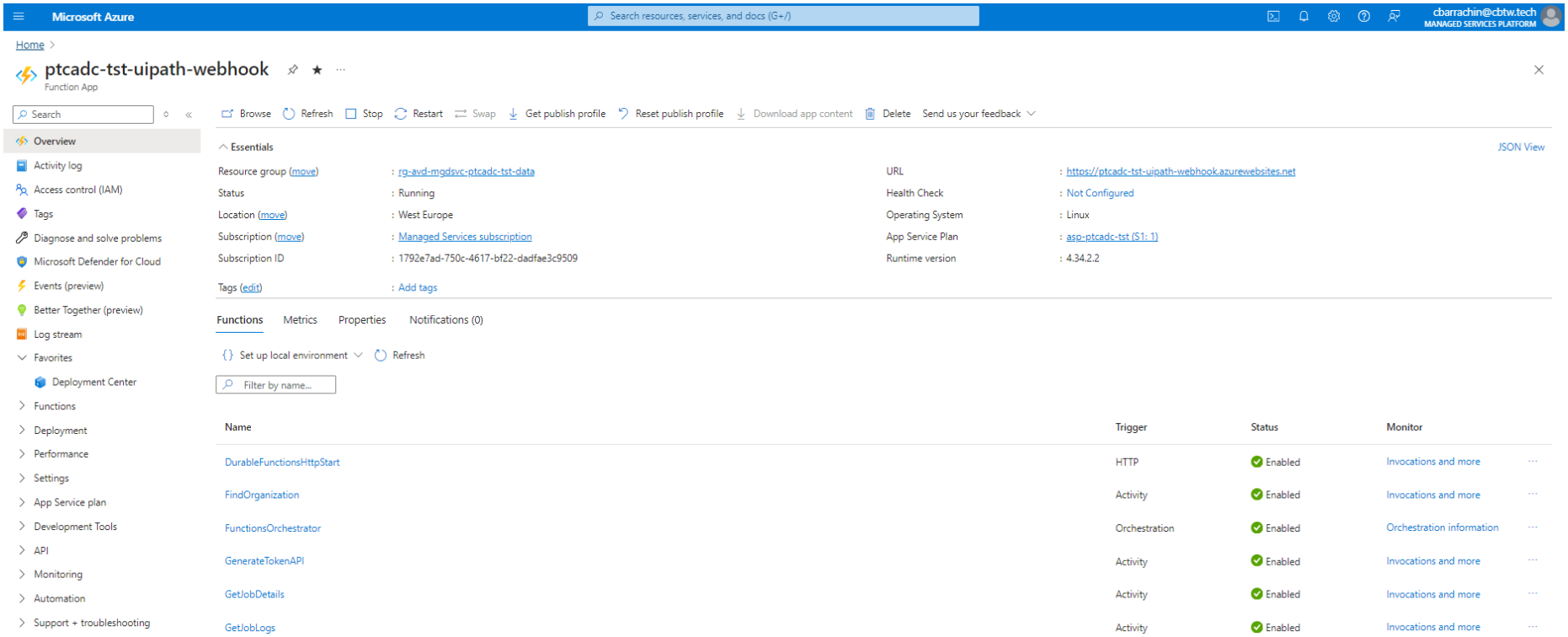


Figure 9: *Interface Function App de Microsoft Azure*

Point de lecture associé à *Figure 10* : **6.1 Dashboard**

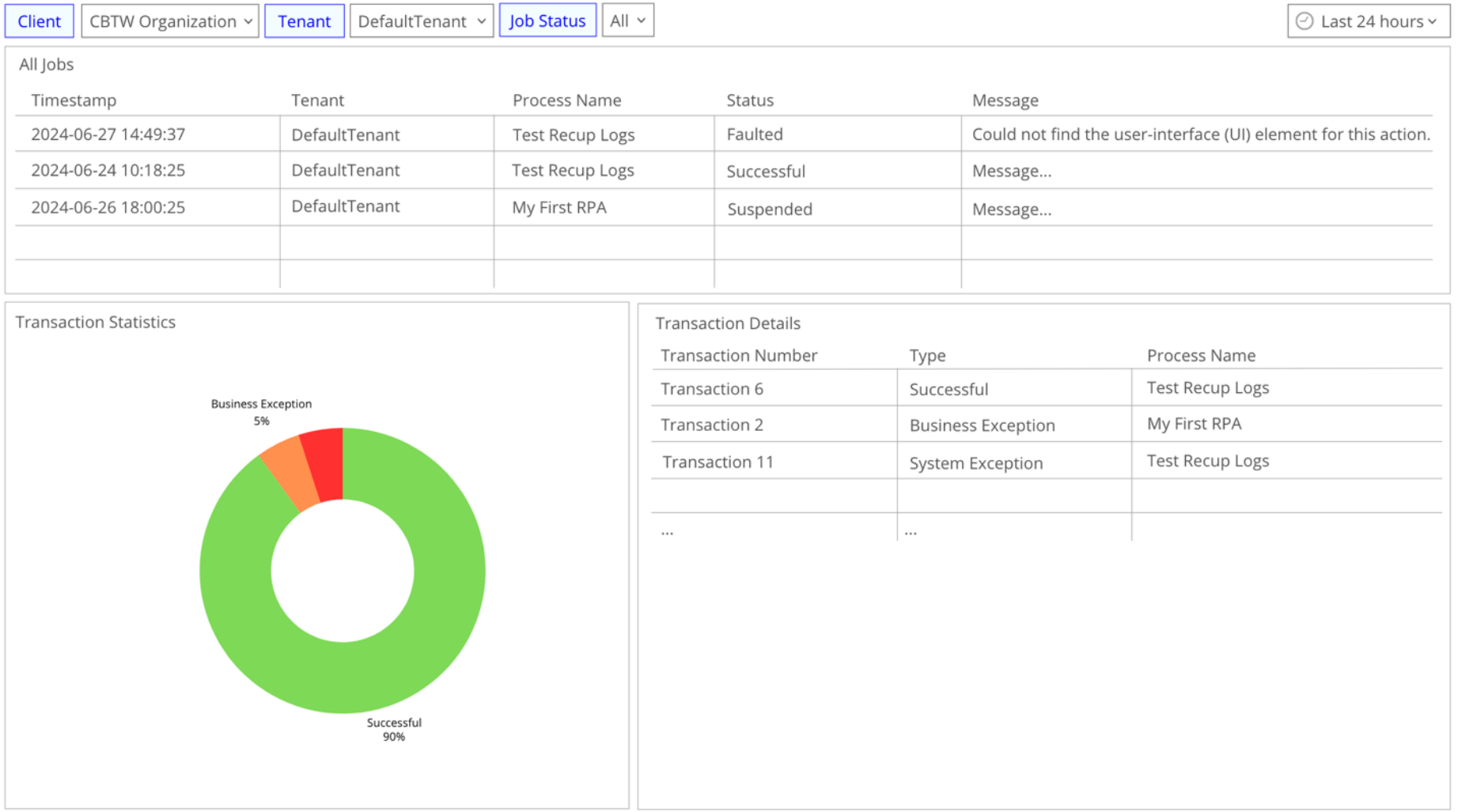


Figure 10: *Dashboard Prévisionnel*

Point de lecture associé à *Figure 11* : **6.2 Dashboard actuel**

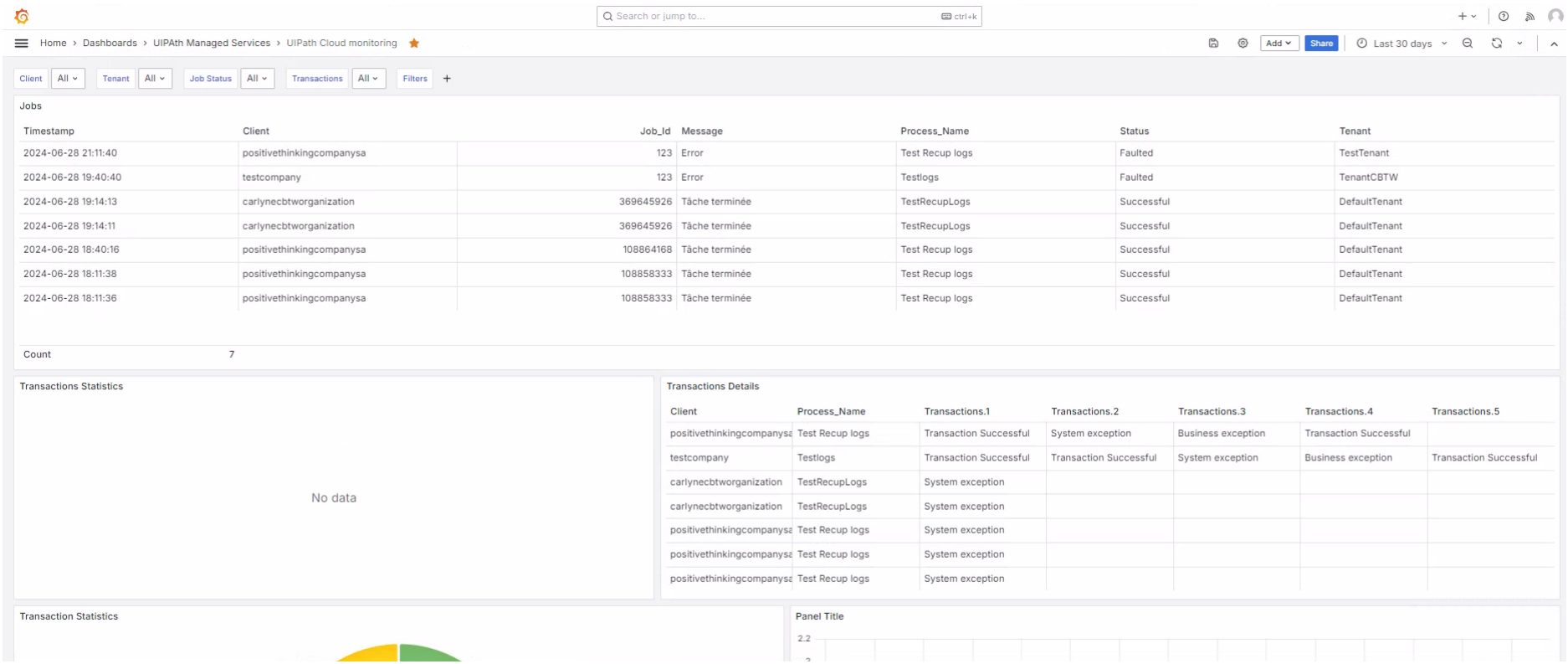


***Figure 11**: Dashboard Actuel*