Université de Sophia
antipolis

SOA

Auteur :
Lefebvre Jeremy
Liechtensteger Michael
Génovèse Matthieu
Chennouf Mohamed

Responsable :
M. Sébastien Mosser

10 novembre 2017

Table des matières

1 Introduction **2**

1.1 Point de départ 2

1.2 Description du projet 2

1.3 Scénario 2

1.4 Choix technologique 2

2 Bus intégration **3**

2.1 Notre flot d'intégration 3

2.2 Justification du flot d'intégration 4

2.3 Cas d'erreur 4

3 Gestionnaire de preuves **5**

3.1 Paradigme Document 5

3.2 Détail du service 6

4 Les Tests **7**

5 Répartition du travail **7**

6 Conclusion **8**

1 Introduction

1.1 Point de départ

Dans un premier temps, nous devions mettre en place un système permettant à une entreprise de pouvoir proposer à ses employés des réservations d'hôtels, de voiture et de billets d'avions afin de faciliter les déplacements commerciaux de ces derniers. Nous devions aussi mettre en place un module permettant à un manager de valider ou non les réservations effectués par les employés. Ce système reposait sur trois services, le premier gérant exclusivement les réservations d'avions, le second gérant les réservations d'hôtels et de voitures et le dernier gérant l'approbation par le manager.

1.2 Description du projet

Dans un second temps, nous avons du étoffer notre premier système en incorporant des services extérieurs de réservations d'hôtels, avions et voiture afin de pouvoir effectuer un comparatif d'offres avec plusieurs sources différentes, l'entreprise préférant pouvoir payer moins cher pour le même service. De ce fait, il était important de mettre en place une structure permettant la communication entre nos services et les nouveaux. De plus, de nouvelles contraintes ont fait leur apparition, comme le fait de proposer un service permettant de récupérer des notes de frais lors du déplacement de l'employé, qui peuvent s'accompagner d'une description justifiant certains frais supplémentaires. La possibilité au manager de valider ou non le remboursement des frais supérieurs au seuil de remboursement en vigueur dans le pays de destination de l'employé et de garantir un archivage des rapports de mission qu'ils soient validés ou non.

1.3 Scénario

Lors de la planification d'un voyage d'affaires, un employé doit commencer par récupérer des informations sur les vols, l'hôtel et la location de voiture pour une certaine destination et demander l'accord de son supérieur pour ces dépenses. Lorsque plusieurs services sont disponibles pour ces recherches, l'entreprise préfère toujours l'offre la moins chère. Lorsque le voyage est approuvé, l'employé recueille des "preuves de dépenses" pendant le voyage (par exemple, reçu de taxi, facture de restaurant, facture d'hôtel...). Le but est de collecter ces pièces et de soutenir un rapport de dépenses de voyage. L'employé envoit ses pièces, qui sont analysées et utilisées pour pré-remplir son rapport de dépenses de voyage. Le manager et l'employé peuvent avoir accès aux différentes notes de frais. Si le montant total est inférieur à un seuil qui est différent selon chaque pays, le rapport est automatiquement accepté à la fin du voyage. Cependant, si le montant est supérieur, l'employé doit ajouter des explications sur la différence (Exemple : «Mon avion n'a pas décollé à cause d'une éruption volcanique en Islande, j'ai dû rester 5 nuits de plus»). C'est au manager d'accepter ou de rejeter le rapport. Lorsque le système déclenche un remboursement pour un employé, toutes les pièces et les justifications possibles sont archivées.

1.4 Choix technologique

Pour répondre au sujet nous avons utilisé les technologies suivantes :

- Docker permet d'exécuter nos services et des services extérieurs à l'intérieur de conteneurs. Sachant que chaque conteneur vit sa vie et gère son propre environnement d'exécution, cela nous permet d'avoir chaque service conteneurisé comme une boite noire.

- Le langage Camel pour l'intégration de nos services. Camel propose un large panel de fonctions pour réaliser nos flots d'intégration.

- ServiceMix est un ESB. On l'utilise car il permet la communication des services qui n'ont pas été conçues pour fonctionner. C'est à l'heure actuel, l'un des meilleurs ESB disponible.

- Gatling est un outil de test de montée en charge, nous l'avons utilisé pour notre nouveau service de gestionnaire de preuve

- Cucumber est un outil permettant de réaliser des tests automatisés. Ces tests sont écrits dans le style compréhensible par une personne lambda. On y retrouve la story, les scénarios ainsi que les steps (given, when, then). Nous avons utilisé Cucumber pour faire des tests sur notre nouveau service de gestionnaire de preuve.

- Junit qui est un framework de tests unitaires reposant sur des assertions qui testent les résultats attendus. On l'utilise pour réaliser nos tests intégration afin de vérifier nos résultats.

2 Bus intégration

2.1 Notre flot d'intégration

Voici le schéma correspondant à notre flot d'intégration que nous allons décrire juste après :

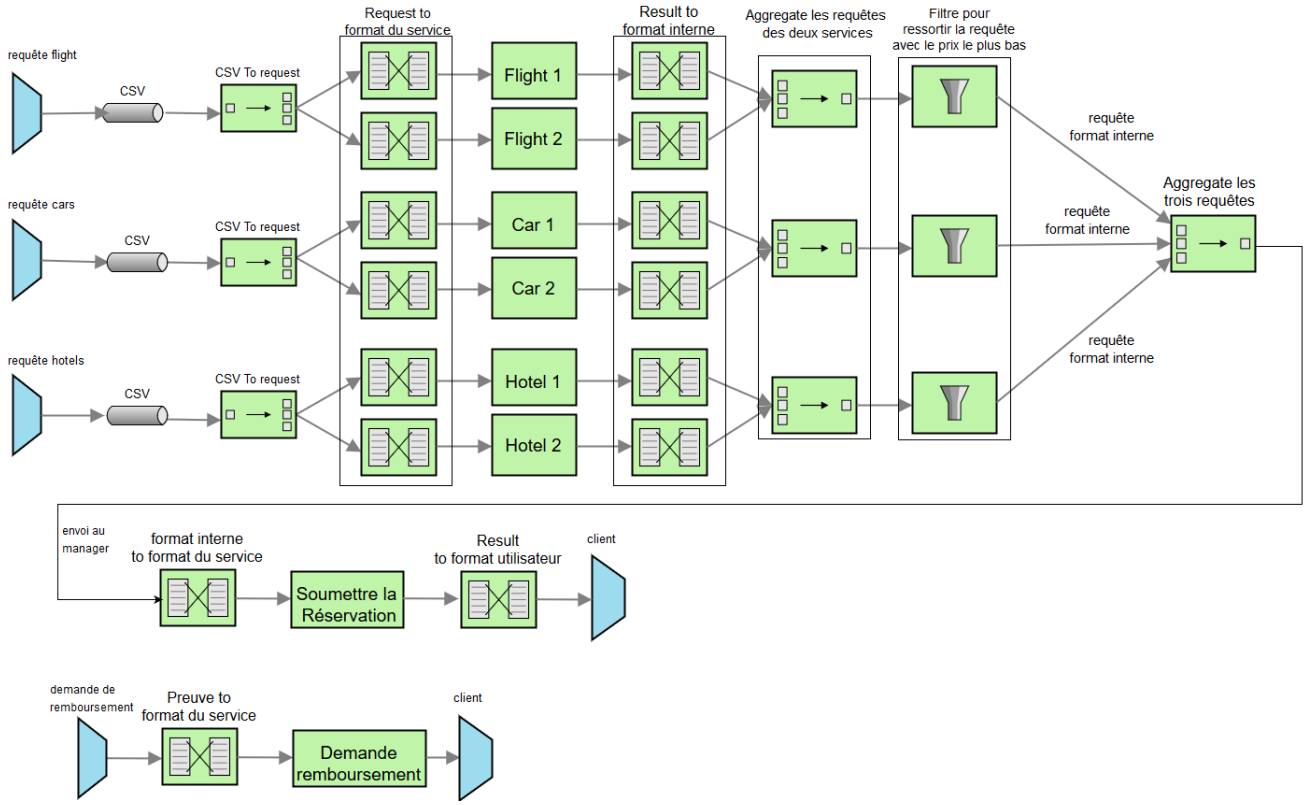

Figure 1 : Diagramme de flow d'intégration

On commence par envoyer nos trois requêtes "flight", "hôtel" et "car". Chaque requête va être stockée dans une queue, cela nous permet en outre d'éviter les erreurs lorsque le nombre de requêtes est élevé. Ensuite on transforme la demande utilisateur en un format interne de requête puis on multicast chacune des 3 queues qui va envoyer les requêtes vers un transformateur. Une fois ces requêtes reçues, on les transforme au format interne des services. Lorsque les services nous répondent, nous transformons leurs requêtes au format interne que l'on va agréger puis filtrer en fonction du prix. Pour finir, le résultat de chacune des routes passe par 3 queues pour récupérer les réponses aux requêtes de façon

asynchrone, dans le cas éventuel d'un grand nombre de réponse aux requêtes. On agrège la réponse de la requête de chacune des routes, puis on transforme la réponse du format interne au format du service "Approbation Manager". Le service d'approbation va retourner un résultat que l'on soumet au client.

Concernant l'intégration du service de Demande de remboursement, le service gère plusieurs entrées. La première, concerne l'employé et lui permet d'envoyer ses pièces justificatives de dépenses au fur et à mesure de son déplacement. La seconde concerne le manager qui peut valider ou non les dépenses de ses employés, qu'elles dépassent ou non le seuil de remboursement autorisé.

2.2 Justification du flot d'intégration

Les entrées dans le bus sont des fichiers csv, afin de mocker des requêtes utilisateurs. Nous avons choisi le format csv pour représenter les requêtes des utilisateurs car il nous semblait facile avec ce format de récupérer une information voulue dans un certain champ, pour les envoyer aux différents services. Différentes queues sont utilisées dans notre bus d'intégration, notamment pour stocker les requêtes effectuées aux différents services de vols, hôtels et voitures. Cela permet de ne pas surcharger le bus en cas de forte charge. Nous disposons des traducteurs de part et d'autres des services pour pouvoir effectuer les traductions adéquates pour que les services fonctionnent et que l'on puisse manipuler nos propres jeu de données qu'importe le format de réponse des services.

On utilise un pattern Split/Aggregator afin de pouvoir consulter plusieurs services différents avec la même requête utilisateur. De plus, l'utilisateur doit obligatoirement effectuer une requête pour chaque service sinon le flot ne se poursuit pas. C'est assez contraignant, dans nos choix de départ du premier projet nous voulions permettre à l'employé d'effectuer de 1 à 3 réservations selon ses besoins sur place, cela n'a pas été implémenté car nous n'avons pas réussi à spécifier à l'aggregator situé juste après les filtres de poursuivre son chemin au bout d'un certain temps sans les requêtes non voulues par l'utilisateur et sans prendre d'autre requêtes pour compenser. Aussi, ce dernier aggregator n'est relié à aucun split, nous nous sommes rendu compte qu'une possibilité qui n'a pas été explorée était de demander à l'utilisateur une seule entrée qu'on aurait pu split et envoyer dans les queues de début de notre flot d'intégration. Cette possibilité permet à l'utilisateur de ne pas saisir plusieurs fois son entrée tout en effectuant obligatoirement 3 demandes de réservations aux autres services. Finalement cette idée n'avait pas été explorée puisque notre idée de design ne s'y prêtait pas.

Concernant la récupération des requêtes des différents services, nous avons fait le choix d'utiliser comme objet interne un format ayant pour attributs, les champs en commun entre le service développé par notre équipe et le service récupéré. Cela permet à notre format interne d'être identique en fonction du service appelé.

2.3 Cas d'erreur

En cas de panne de l'un des services de vols, d'hôtels ou voitures, la requête va seulement être effectuée sur le service encore actif.

Si un service de réservation n'a pas de réponses à proposer à l'utilisateur, on regarde l'autre service du même type. S'il a des réponses à proposer, alors aucun problème, sinon la requête n'aboutira pas. L'utilisateur devra alors changer des paramètres dans sa requête pour pouvoir finalisé sa demande de réservations.

A partir du moment où une entrée est invalide, l'utilisateur est prévenu que sa requête ne peut aboutir, cela ne plante pas le bus et l'utilisateur peut fournir des requêtes valides par la suite et compléter sa demande de réservation sans problème.

Si aucun des deux services n'est actif, la requête est annulée. Et d'après notre implémentation où notre aggregator attend spécifiquement 3 réponses, la requête utilisateur ne parviendra jamais à la

fin de notre flot. Dans ce cas précisément, le fait que deux services du même type soient déconnectés implique que tous le bus est bloqué. En poursuite du projet, il serait possible dans ce cas de créer un type de réservations factice crée spécialement lors de la détection de ce genre de problème et qui permettrait à l'aggregator suivant de ne pas être bloquant. Une autre piste de résolution de ce problème serait de notifier l'aggregator qu'il faut qu'il attende autant de réponse que de demandes de réservations de l'utilisateur (ex : Si l'utilisateur ne veut réserver qu'un hôtel et un avion, alors il n'a qu'a requêter ces deux services pour que l'aggregator accepte cette demande et permette à la demande de réservations de l'utilisateur de se finalisé sans soucis).

3 Gestionnaire de preuves

Pour la gestion de preuves nous avons considéré que nos anciens services étaient des "boites noires" et qu'il ne fallait pas les modifier. De ce fait, au lieu d'enrichir le service "ApprobationTravelManagement" pour qu'il puisse aussi valider ou refuser les preuves du voyage, nous avons choisi de faire un nouveau service "BillManagement". Un des intérêts de créer un nouveau service est de permettre à nos services d'être plus spécifiques. De plus, les preuves envoyées pouvant être de forme différentes, nous n'avons pas voulu réutiliser l'ancien service qui utilisait le paradigme RPC, car nous avons estimé que le paradigme document était plus adapté à ce service.

3.1 Paradigme Document

Nous avons donc choisi le paradigme "document" pour ce service car le nombre de dépense varie et ne peut être défini. Ce cas ressemble au service de réservation d'avions dont le nombre de paramètres était inconnu à l'avance, le paradigme Document nous offrant une plus grande liberté concernant les entrées utilisateurs et ce sera au service d'utiliser les bonnes méthodes en fonction des entrées.

Requête Post :

L'employé peut soumettre ces dépenses :


```
{
        "type":"submit",
        "bills":{
                "id":99,
                "identity":{
                            "firstName":"toto",
                            "lastName":"pouloulou",
                            "email":"toto@etu.unice"},
                "spends":
                [{
                        "id":"0",
                        "reason":"Restaurant",
                        "date":"05/02/2018",
                        "country": "AM" ,
                        "prix" :
                            {"price":120,
                            "currency":"EUR"}
                        },
                        {
                        "id":"1",
                        "reason":"Restaurant",
                        "date":"01/02/2005",
                        "country": "AM" ,
                        "prix" :
                            {"price":90,
```

```
                                    "currency":"EUR"}
                        }]
    }
}
```

L'employé peut ajouter une dépense à ces dépenses déjà existants :

```
{ "type":"addSpend",
  "id":99,
  "spends": {
    "id":"3",
    "reason":"Taxi",
    "date":"05/02/2018",
    "country": "AM" ,
    "prix" : {
        "price":44,
        "currency":"EUR"
        }
    }
}
```

L'employeur, à la fin du mois, peut accepter les dépenses si celle si ne dépasse le seuil, elles seront archivé dans un fichier :

```
{     "type":"validate",
      "id":99}
```

L'employeur peut refuser les dépenses si celles ci dépasse le seuil , elles seront archivées dans un fichier :

```
 {    "type":"reject",
      "id":99 }
```

L'employé peut avoir les détails des dépenses qu'il a soumit :

```
{    "type":"retrieve",
     "id":99}
```

L'employé peut ajouter une justification des dépenses :

```
{    "type":"addJustification",
     "id":22,
     "justification":"Mon avion n'a pas decolle"}
```

Pour faciliter nos tests on peut vider la base de donnée avec :

```
{"type":"purge"}
```

3.2 Détail du service

Pour notre service, le calcul du seuil est calculé de la façon suivante : - On fait une requête Post sur l'URL : `https://www.economie.gouv.fr/dgfip/fichiers_taux_chancellerie/missiontxt`, afin de récupérer les informations sur les seuils que l'on stock dans un tableau.

Points positifs :

- si les données de l'URL sont mise à jour, nous n'avons rien à changer.

- On ne s'encombre pas à avoir un gros fichier qui contient les seuils et qui potentiellement prend beaucoup de place dans le projet.

Points négatifs :

- La requête passe par le réseau et peut prendre un temps élevé mais étant donnée que c'est une requête qui est appelé seulement lorsque l'on soumet des dépenses, cela n'est pas problématique

On a des algorithmes qui nous permettent avoir un seuil filtré en fonction du pays , de la date mais aussi de l'argent utilisé :

- Les pays sont de la forme AD, AE, AG... Dans le cas où l'utilisateur met un pays qui n'existe pas, on a décidé de mettre une valeur par défaut pour le seuil qui est de 150 EUR/USD...
- Pour la date, on choisi le seuil qui a la date la plus proche de la date de dépense.On prend en compte que la date du seuil est forcément antérieur à la date de la dépense.Dans le cas où il n'y a aucune date antérieur ou que le format de la date : "16/06/1999" n'est pas respecté, le seuil de remboursement est de 150 EUR/USD...

- On récupère le seuil aussi en fonction de la monnaie , si le format de la monnaie n'est pas respecté : "EUR", "AED", "USD"... alors le seuil de remboursement est de 150 EUR/USD...

  Lorsque le manager valide ou refuse les dépenses de l'employé, les informations sont archivées dans un fichier "dépenses_"+id.

4 Les Tests

Notre bus étant l'élément central de notre intégration, nous avons réalisés un grand nombre de tests afin de s'assurer de son bon fonctionnement dans la majorité des cas. Nous avons donc effectuer des tests sur :

- Les routes du flot d'intégration. Les communications entre composant qui mène du point de départ au point d'arrivée
- La pertinence des résultats. On s'est assuré qu'on avait en sorti l'offre de réservation la moins cher pour chaque service de réservations.
- Les cas de plantages et d'erreurs évoqués plus haut.

Aussi, nous avons effectuer des tests de charge et d'acceptation du nouveau service de gestion des remboursements.

5 Répartition du travail

Matthieu :
Création du service hotels et cars en REST
Gestion maven / docker
Bus d'intégration
Quelques tests

Michael :
Création du service flights en document + tests acceptation et stress de ce service
Route pour les service hotels
Participation aux tests de la deathpool
tests de la route des hotels
Tests d'acceptation pour le service BillManagement

Mohamed :
service d'approbation de voyage en RPC,
service de dépenses en document,
test de stress sur le service de dépenses,
test d'intégration pour les gestions erreurs

Jeremy :
Principalement tests d'acceptation de plusieurs services et des routes du Bus
Fix de bugs
Intégration des services voitures au bus

Mohamed	Michael	Jeremy	Mattieu
22%	22%	19%	37%

6 Conclusion

Ce projet nous a permis d'utiliser et de mieux comprendre les fonctionnements de différents paradigmes de requêtes utilisés par les web services, que sont REST, DOCUMENT et RPC, et de mieux entrevoir leurs caractéristiques spécifiques.
Nous avons pu aussi nous familiariser avec des concepts tel que les architectures orientée services ainsi que la mise en place d'un bus de message afin de les faire communiquer et les intégrer ensemble. Ce procédé est nécessaire pour faire évoluer notre système et y ajouter de la valeur, mais cette intégration fastidieuse à un prix non négligeable de temps et de compétences. Il serait intéressant de continuer à faire évoluer notre architecture afin de proposer de plus en plus de services comme un système de notation par exemple, ou la possibilité d'effectuer des réservations de restaurants afin d'améliorer notre maîtrise de ce paradigme et d'aboutir à une architecture orientée micro-services.