

Création et intégration du Speed layer, sous-composant de l'architecture Lambda

Beebuzziness



Membres du Jury :
Gwenn Boussard
Michel Poitevin - Tuteur entreprise
Francis Brunet-Manquat - Tuteur universitaire

Auteur : Marie Kersalé
Licence professionnelle Métiers de l'informatique Application Web 2020



Déclaration de respect des droits d’auteurs

Par la présente, je déclare être le seul auteur de ce rapport et assure qu’aucune autre ressource que celles indiquées n’ont été utilisées pour la réalisation de ce travail. Tout emprunt (citation ou référence) littéral ou non à des documents publiés ou inédits est référencé comme tel.

Je suis informé(e) qu’en cas de flagrant délit de fraude, les sanctions prévues dans le règlement des études en cas de fraude aux examens par application du décret 92-657 du 13 juillet 1992 peuvent s’appliquer. Elles seront décidées par la commission disciplinaire de l’UGA.

A Grenoble,

Le 9 janvier 2020,

Signature

KERSALE MARIE

Remerciements

J'adresse tout d'abord mes remerciements à Pierre Nicodème-Taslé, directeur général et Bastien Guillard, directeur du développement, pour m'avoir accordée leur confiance durant cette année d'alternance et proposée un tuteur hors pair.

Je tiens donc à remercier considérablement mon tuteur, Michel Poitevin, développeur senior, pour avoir mis en place une mission sur mesure en s'adaptant au rythme de l'alternance une semaine de cours/une semaine d'entreprise, rythme qui n'était pas approprié au sprint de 3 semaines mises en place dans l'équipe. Il a su m'accompagner, m'aiguiller ainsi que valoriser mon travail lors de cette mission, un grand merci à lui.

Je remercie également tous les membres de mon équipe pour m'avoir soutenue et guidée tout au long de cette année scolaire grâce à leurs précieux conseils, leurs méthodologies et leur pédagogie: Antoine Begon, Julien Vienne, Clément Salin et Mokhtar Mial.

Table des matières

Introduction.....	5
I. Beebuzziness, entreprise d'accueil.....	6
I.1 Présentation.....	6
I.2 Activité.....	7
I.3 Organisation.....	9
II. Ma mission, Le Speed layer.....	12
II.1 L'architecture Lambda.....	12
II.2 Cadre technique.....	16
III. Réalisations.....	18
III.1 Les données.....	18
III.2 Implémentation.....	22
III.3 Interfaces.....	28
III.4 Test development driven.....	29
III.5 Intégration.....	32
III.6 Objectifs atteints, résultats obtenus.....	34
IV. Conclusion.....	35
V. Glossaire.....	36
VI. Sitographie.....	38
VII. Annexes.....	39

Table des figures

Figure 1 : Player [p7]
Figure 2 : Site officiel Ustream / Figure 3 : Hub officiel Ustream [p8]
Figure 4 : Hub description [p8]
Figure 5 : Panneau de statistiques [p9]
Figure 6 : Organigramme des services [p10]
Figure 7 : Schéma répartitions équipes de développement [p11]
Figure 8 : Schéma simplifié de l'architecture Lambda [p14]
Figure 9 : Schéma détaillé de l'architecture Lambda. [p15]
Figure 10 : Liste events / Figure 11 : Définition de la configuration de l'agrégation 'OpenMedia' [p19]
Figure 12 : Liste requêtes / Figure 13 : Définition de la configuration de la requête 'OpenMedia' [p20]
Figure 14 : Aggrations / Figure 15 : Factory [p21]
Figure 16 : Diagramme de classes du Speed layer [p22]
Figure 17 : Diagramme de séquence d'un évènement [p25]
Figure 18: Diagramme de séquence synchronisation du Batch layer et du Speed layer [p26]
Figure 19 : Diagramme de séquence d'une requête [p27]
Figure 20 : Interfaces du manager [p28]
Figure 21 : Exemple test unitaire mis en place [p30]
Figure 22 : Code implémenté avec le test unitaire [31]
Figure 23 : Schéma d'architecture du Speed layer service [p32]
Figure 24 : RabbitMQ [p33]

Introduction

Étudiante en Licence professionnelle Métiers de l'informatique Application web à l'IUT 2 à Grenoble, je suis en alternance au poste de développeuse web au sein de Beebuzziness, entreprise éditrice de logiciels, dans le service développement.

Cette entreprise propose sa propre solution, Ubstream qui est une plateforme permettant d'aider les marques et les propriétaires de contenus digitaux à optimiser leur organisation et leur diffusion sur le web. Le module principal est appelé hub, espace de stockage de documents digitaux, autour duquel s'articule différents modules de consultation, de partage et d'analyse de médias.

J'ai rejoint cette entreprise Grenobloise en tant qu'intégratrice il y a 4 ans. Depuis 2017 j'ai intégré une équipe de développeurs et commencé mes premiers pas dans le développement. J'ai pu ainsi participer à la conception de l'Interactive Smart Signage* (service de diffusion interactif de médias) et des statistiques.

Ma mission consiste à intervenir sur le système de statistiques disponibles dans les options d'un hub. L'architecture Lambda, système permettant de traiter des volumes importants de données, a été choisie et mise en production cette année. J'ai pour mission de réaliser et d'intégrer le Speed layer, pièce manquante aujourd'hui de ce système qui va permettre d'affiner les résultats de statistiques grâce au traitement de données temps réel.

Le déroulement de cette année d'alternance se découpe en trois étapes, la première consiste à analyser et comprendre le système Lambda dans son ensemble, la seconde concerne l'implémentation du code du Speed layer et cette dernière implique l'intégration de ce sous-composant dans le système Lambda.

Le choix d'effectuer cette alternance a été motivé par le fait de pouvoir à la fois valider mes acquis et enrichir mes compétences en développement pour une meilleure adaptation lors de nouveaux projets.

Je vais vous présenter dans un premier temps l'entreprise dans laquelle se déroule cette année d'alternance. J'évoquerais ensuite ma mission de manière globale puis ses spécificités techniques, ses objectifs, son découpage ainsi que sa réalisation.

I. Beebuzziness, entreprise d'accueil

I.1 Présentation

Beebuzziness est une entreprise éditrice de logiciels, fondée en 2001 par Pierre-Nicodème Taslé. Les équipes de Beebuzziness développent Ustream*, une plateforme de gestion de contenu en mode SaaS (Software as a Service) Logiciel en tant que Service en Français. C'est un modèle de distribution de logiciels au sein duquel les applications sont disponibles pour ses clients par l'intermédiaire d'internet. Le siège social et les activités commerciales sont implantés à Paris tandis que la production et R&D sont à Grenoble (45 employés).

La société Beebuzziness fût créée en 2001 dans le but de concevoir une technologie de dématérialisation et d'enrichissement de documents. La dématérialisation permet de convertir un document PDF en document virtuel, utilisée dans deux cas majeurs :

- Avec les grandes entreprises ou organisations pour la diffusion de leur communication corporate et aux actionnaires (rapport annuels, plaquettes d'information interne, lettre aux actionnaires...). Exemples de clients du dispositif : SPIE, BPI France, Fayat Énergie Services, CMA de l'Isère.
- Avec le secteur du retail dans lequel les consommateurs peuvent consulter les documents en ligne pour y trouver les promotions courantes sous forme de feuilletable avec des fiches produit et des informations détaillées sur les articles en vente dans les magasins.

La consultation de ces documents est facilitée par des enrichissement tels que des ZZO (zoom image), des liens internes, externes, QR code et animations et disponible dans un player pour une consultation fluide et interactive [Figure 1].



Figure 1: Player - consultation de documents en ligne

En 2016, Beebuzziness élargi ses services au-delà de la consultation et de l'enrichissement de documents digitaux et déploie la plateforme Ustream qui rassemble toutes les fonctions nécessaires à la gestion et à la diffusion de médias. Les médias représentent des documents, des photos, des vidéos et des liens. Je vous invite à aller sur le site officiel d'Ustream [Figure2].



Figure 2: Site officiel d'Ustream



Figure 3: Hub de Beebuzziness

I.2 Activité

Le hub

Le hub est un espace de stockage de documents virtuels et la pièce maîtresse d'Ustream autour duquel s'articulent différentes fonctionnalités de consultation, de diffusion et d'analyse d'activités autour des médias. Découvrez le hub de Beebuzziness via le Qrcode [Figure3 et 4]. On va s'intéresser aujourd'hui à la partie analyse avec des statistiques proposées aux clients retraçant le trafic de hubs et de médias.

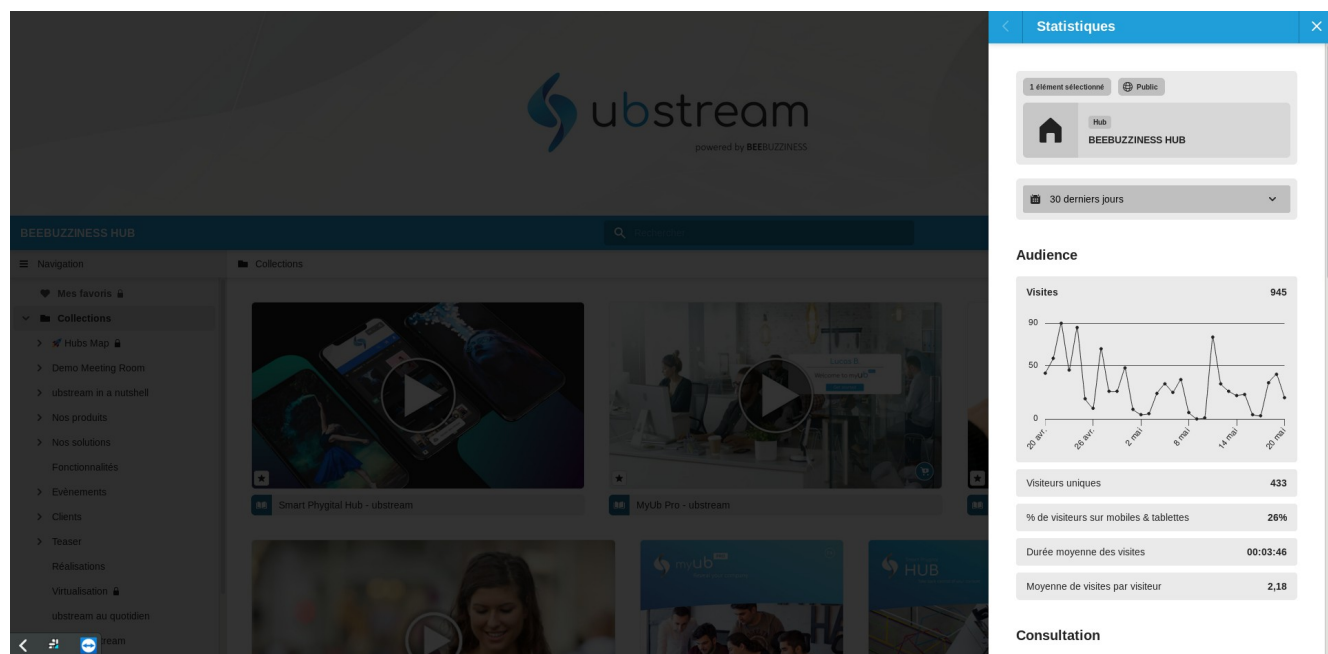


Figure 4: 1-Bannière / 2-Menu du hub / 3-Barre d'outils / 4- Arborescence / 5-Espace médias

Les statistiques

Les statistiques sont disponibles sur un panneau accessible via le menu d'un hub et permettent d'avoir une vue sur l'activité de vos médias tels que le nombre de visites sur un document, les pages les plus consultées, le temps moyen passé sur une page, le nombre de partage sur les réseaux [Figure 5]. Vous retrouverez le panneau détaillé en [annexe A](#) p38.

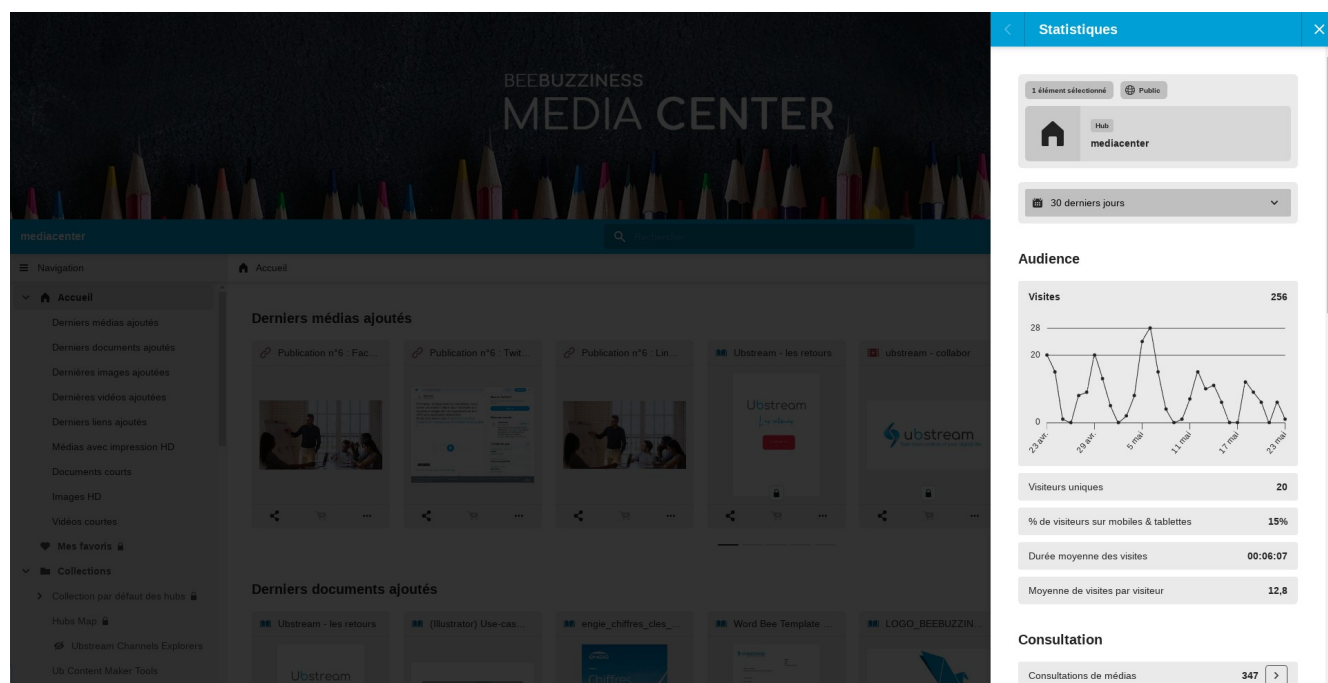


Figure 5: Panneau de statistiques d'un hub

I.3 Organisation

Services

Beebuzziness comprend 49 salariés repartis dans 5 services, la production, le marketing, la création, l'administration et le développement [Figure 6].

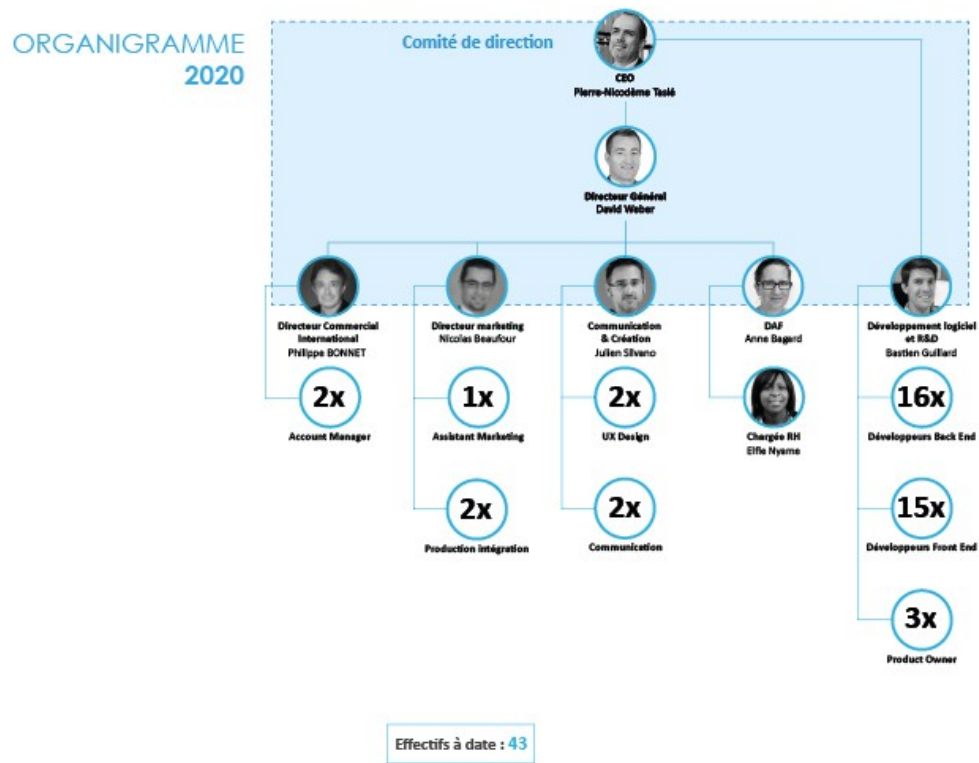


Figure 6: Figure 2: Organigramme des services de Beebuzziness avec la mise en valeur du comité directionnel.

Equipes de développement

Le pôle de développement axé sur la R&D (Recherche et Développement), regroupe 30 employés, 26 CDI dont 4 externes et 4 alternants. Voici la structure type de ces équipes dirigées par Bastien Guillard notre directeur du développement [Figure 7]. Sachant que trois Product Owner se partagent les 5 équipes de développement.

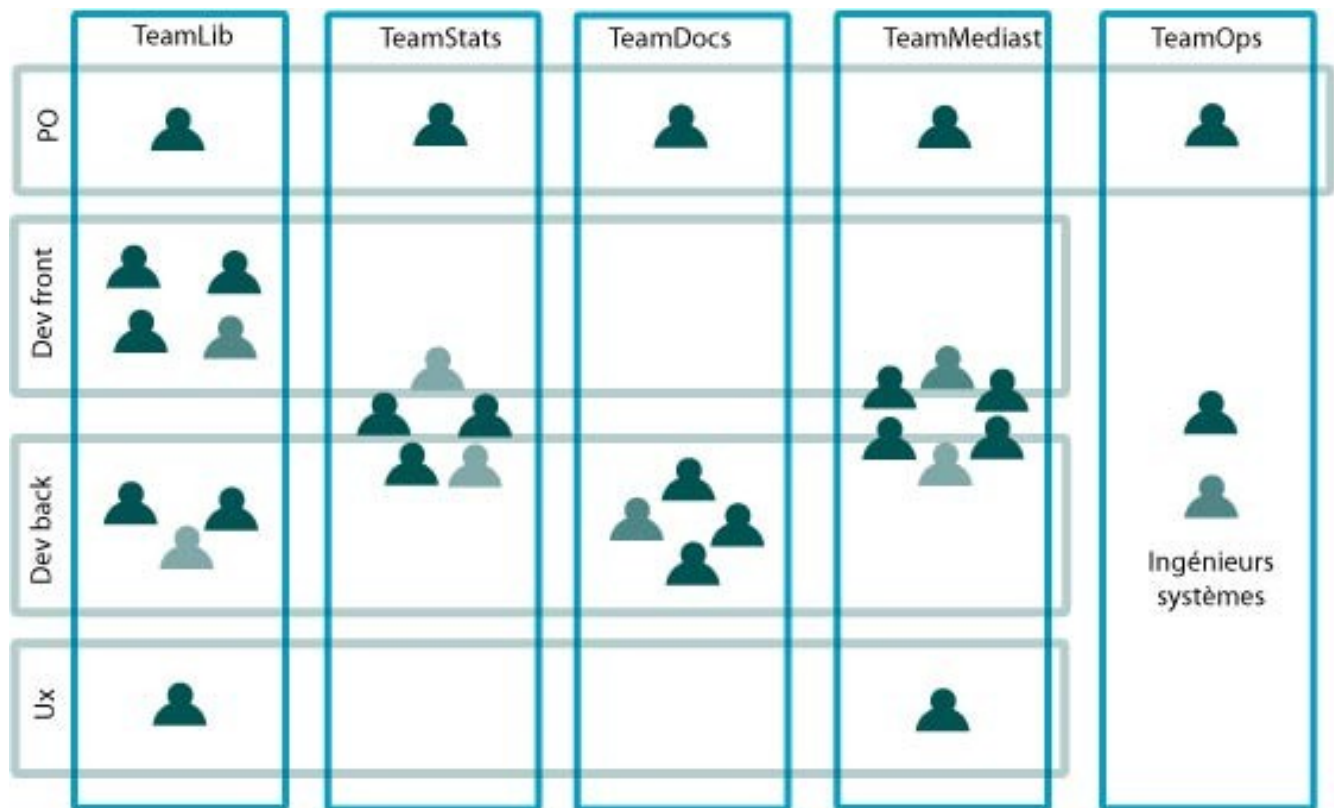


Figure 7: Composition des équipes de développement

La «TeamLib» s'occupe de tout ce qui attrait au hub ainsi qu'au player de documents virtualisés.

La «TeamStats» se charge des modules s'articulant autour d'un hub c'est à dire les statistiques, l'impression à la demande des documents d'un hub et l'Interactive Smart Signage* permettant la diffusion de contenu interactifs à partir des médias stockés dans un hub.

La «TeamDocs» se charge de l'upload, la virtualisation et le stockage de tous les types de documents gérés par le hub (PDF, Microsoft Office, Open Office) et de manière plus globale de la scalabilité et de la robustesse de la plateforme Ustream.

La «TeamMedias» traite les sujets d'upload, de virtualisation, de stockage et de download de tous les types de médias hors documents gérés par le hub (images, vidéos, sons), ainsi que les sujets de permissions (accès, modification, suppression,...) associées aux médias et aux hubs.

La «TeamOps» se charge d'automatiser et de programmer l'infrastructure.

Méthodes agiles, Scrum

L'organisation de ce service de développement s'appuie sur les méthodes agiles depuis 2015.

Une **méthode Agile** est une approche itérative et collaborative, capable de prendre en compte les besoins initiaux du client et ceux liés aux évolutions. La méthode Agile se base sur un cycle de développement qui porte le client au centre. Le client est impliqué dans la réalisation du début à la fin du projet. Grâce à la méthode agile le demandeur obtient une meilleure visibilité de la gestion des travaux qu'avec une méthode classique. L'implication du client dans le processus permet à l'équipe d'obtenir un feedback régulier afin d'appliquer directement les changements nécessaires. Cette méthode vise à accélérer le développement d'un logiciel. De plus, elle assure la réalisation d'un logiciel fonctionnel tout au long de la durée de sa création.

Scrum est l'une des méthodes agiles. Ce terme signifie « mêlée » au rugby. La méthode scrum s'appuie sur des « sprints » qui sont des espaces temps assez courts, pouvant aller de quelques heures jusqu'à un mois. Généralement à Beebuzziness un sprint s'étend sur trois semaines. À la fin de chaque sprint, l'équipe présente ce qu'elle a ajouté au module dont elle est en charge devant l'ensemble de l'entreprise. L'équipe clôture le sprint par une rétrospective, bilan du déroulement de cette période qui permet de souligner les points positifs et définir de nouveaux axes d'amélioration.

II. Ma mission, Le Speed layer

Contexte

Ma mission consiste à implémenter le Speed layer, sous-composant de l'architecture Lambda. Je réalise cela au sein de la TeamStats en charge aujourd'hui de l'outil d'analyse. Depuis début 2019, cette équipe est en charge du développement de la nouvelle version de l'outil de statistiques délivrée avec un hub.

Cette équipe comprend un product owner, Clément Salin et 5 développeurs fullstack*, Michel Poitevin, Julien Vienne, Antoine Begon, Mokthar Mial et Rony Dormevil alternant. Un UI Designer* intervient ponctuellement afin de nous délivrer des specs*.

Je travaille avec mon tuteur entreprise sur cette mission et la durée prévisionnelle est de 9 mois soit le temps de l'alternance. J'interviens directement sur le panneau de statistiques dont l'accès se fait via un hub.

L'objectif principal est le déploiement du Speed layer sur nos dockers locaux* et sur mon hub sur l'infrastructure de production afin de montrer les différences entre les statistiques avec et sans le Speed layer. Son intégration en production sur tous les hubs se fera ultérieurement après une batterie de tests applicatifs et l'accord de notre directeur du développement.

Un docker est une plateforme logicielle open source permettant de créer, de déployer et de gérer des containers d'applications virtualisées sur un système d'exploitation. Les services ou fonctions de l'application et ses différentes bibliothèques, fichiers de configuration, dépendances et autres composants sont regroupés au sein du container. Chaque container exécuté partage les services du système d'exploitation.

II.1 L'architecture Lambda

Description générale

Le but de l'architecture Lambda est de fournir un modèle de traitement temps réel sur des volumes importants de données (Big Data), en proposant un nouveau modèle de calcul.

La conception d'une architecture lambda est guidée par les contraintes suivantes :

- la robustesse avec une faible tolérance aux erreurs évitée et la traçabilité de la correction des erreurs.
- la scalabilité en gérant de manière indépendante chaque niveau.
- réduire latence avec l'écriture des événements et l'exécution des requêtes dans les tableaux de bord statistiques.
- la normalisation des événements pour pouvoir répondre à de futurs besoins métiers avec les données accumulées.
- la facilité de maintenance puisque les événements sont simples et les agrégations répondent à un besoin précis.

- les requêtes à la demande sont guidées par le besoin utilisateur et construites sur les agrégations précalculées dans des vues.

Les composants de l'Architecture lambda

Le système lambda est composé de 3 couches [Figure 8] :

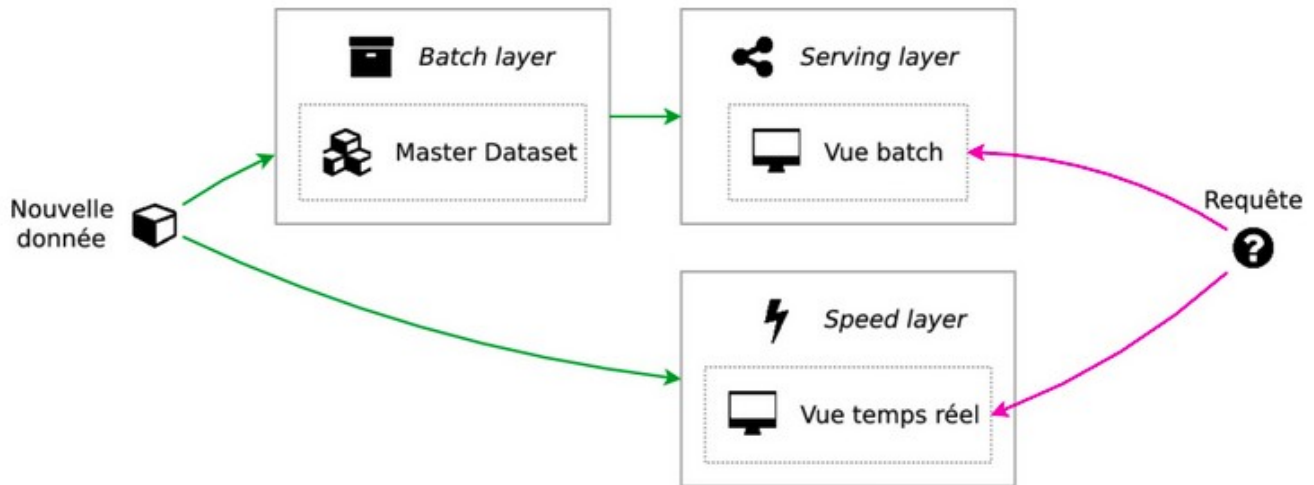


Figure 8: Schéma simplifié de l'architecture Lambda

Le **Batch layer** vise une précision parfaite en permettant de traiter toutes les données disponibles lors de la génération de vues, toutes les heures en se basant sur les heures pleines ou tous les jours en se basant sur l'heure GMT*. Ce composant peut corriger les erreurs en recalculant l'ensemble des données, puis en mettant à jour les vues existantes. La sortie des couches Batch layer répond aux requêtes ad hoc en renvoyant des vues précalculées ou en construisant des vues à partir des données traitées. Le fait de garder les données dans un format brut permet aucune perte de données et cela assure une plus grande flexibilité. Des calculs peuvent être refaits sans toucher aux données brutes.

Le **Speed layer** ne traite que les données récentes et vient compléter, réguler, les chiffres du Batch layer. Les données du Speed layer garantissent le temps réel du système, elles sont par définition éphémères et seront oubliées, lorsque l'agrégation du Batch layer sera à nouveau calculée. Ainsi, on peut se permettre des approximations ou des erreurs, qui seront temporaires et minimiser le temps de latence de mise à disposition de l'utilisateur final le résultat du calcul des agrégations car il est effectué de manière incrémentale à la réception de chaque évènements.

Le **View layer** permet de stocker et d'exposer aux clients/utilisateurs les vues créées par les couches Batch layer dès qu'elles sont disponibles, c'est-à-dire dès que le calcul par la couche Batch layer est complété. Cette couche renvoie des vues pré-calculées ou construit des vues à partir des données traitées. C'est de la donnée grise, donnée construite qui peut être reconstruite au besoin.

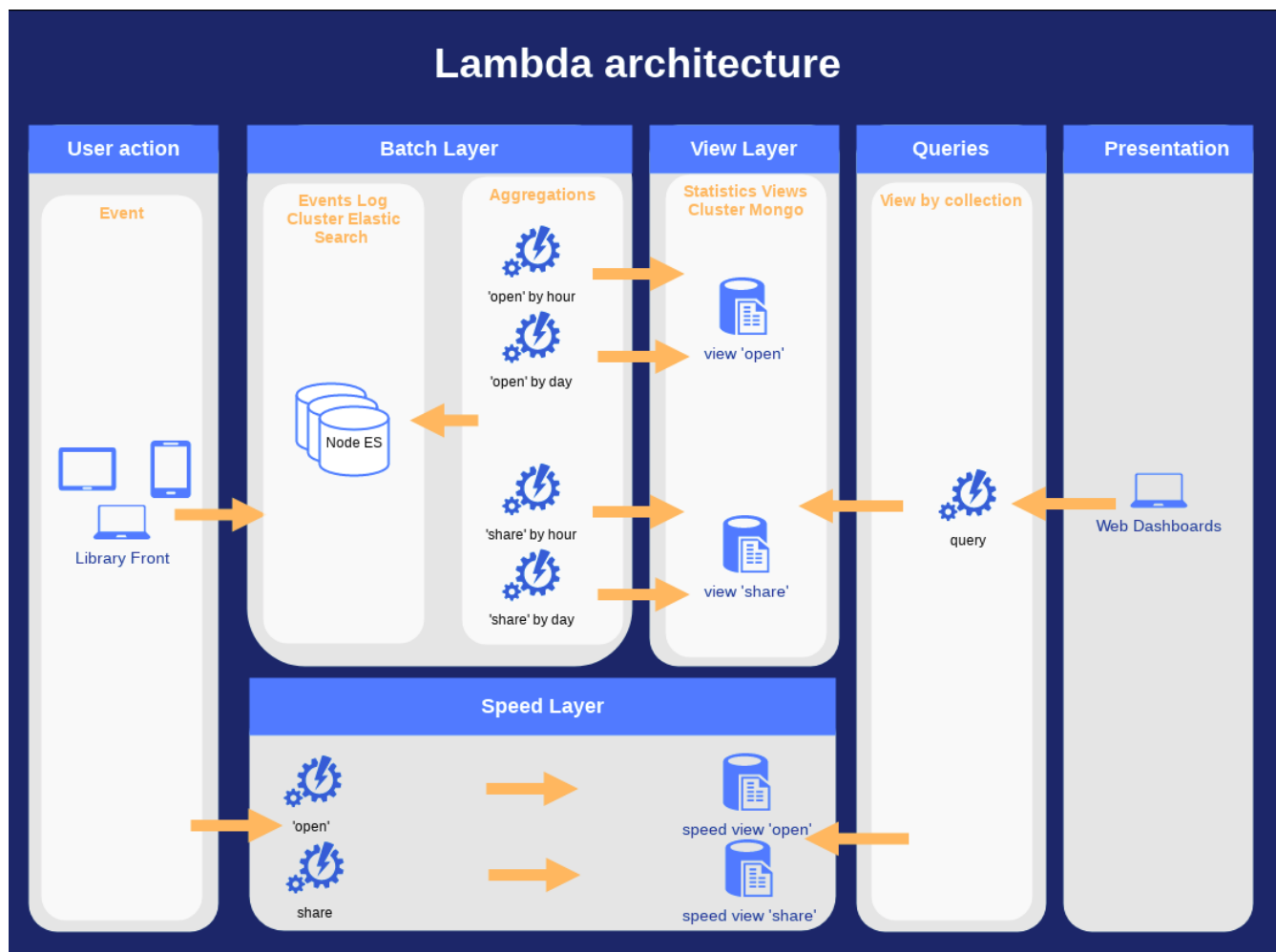


Figure 9: Schéma détaillé de l'architecture Lambda mis en place à terme en production et aujourd'hui disponible sur des dockers locaux

Vous trouverez en **annexe C** p41 le schéma de l'ancienne organisation des anciennes statistiques.

Avantages

Ce système global va apporter précision et richesse à la donnée collectée et permettre d'améliorer les performances de l'outil de statistiques en séparant le stockage, la consommation et la complexité. Il permet de dépasser les limitations d'une base de données classique : compromis entre la disponibilité et la latence, la robustesse et le partitionnement des données.

Ce système est évolutif puisque chaque niveau est géré de manière indépendante et conçu pour gérer de grands volumes de données.

Il est robuste car il permet l'utilisation de cluster sur Mongo et Elastic search qui améliorent notre tolérance aux pannes. En cas de panne d'un des serveurs Mongo ou Elastic search, le cluster gère la

redondance. En cas de surcharge sur les évènements, c'est le système de messagerie RabbitMq qui nous permet d'absorber les pics de charge.

Les tolérances aux erreurs sont évitées puisque l'insertion des évènements est idempotente ce qui signifie qu'une opération a le même effet qu'on l'applique une ou plusieurs fois, ou encore qu'en la réappliquant on ne modifiera pas le résultat, dans notre cas cela signifie qu'un évènement ne sera inséré qu'une seule fois dans le puits de données. C'est la donnée brute qui est insérée. Dans le batch layer, chaque agrégation est idempotente pour un bucket (plage horaire) donné.

Ce système est rapide, avec une latence très faible pour l'écriture des évènements, l'exécution des requêtes et l'affichage de données même complexes en temps réel grâce à ses différentes couches. Enfin la normalisation des évènements permettant de répondre à des besoins futurs avec les données accumulées permet une grande évolutivité.

L'architecture Lambda est indépendante de la technologie car elle précalcule des résultats, les stocke en base, puis interroge cette base pour répondre aux requêtes du demandeur.

II.2 Cadre technique

Langages

Nodes.js est une plateforme de développement Javascript. C'est le langage Javascript avec des bibliothèques. Node propose une solution rapide pour développer des applications back end et coder les deux parties d'une application web dans le même langage. C'est un gain de temps pour le développeur, une flexibilité pour les membres d'une équipe qui peuvent travailler sur le front end et le back end et donc une économie d'argent pour l'entreprise.

TypeScript

Langage de programmation libre et open source développé par Microsoft, il améliore et sécurise la production de code Javascript en typant les objets manipulés. Nous utilisons ce langage avec node.js.

Lodash est une bibliothèque JavaScript réunissant des outils bien pratiques pour manipuler des données, là où peuvent manquer des instructions natives :

- pour des tableaux, objets, chaînes de texte (notamment des itérations, du clonage)
- pour tester et manipuler des valeurs
- pour créer des fonctions composites

Outils

RabbitMQ est un système de file d'attente dans lequel les événements sont stockés temporairement et permet de transporter et router les messages depuis les producteurs vers les consommateurs.

Elasticsearch est une base de données scalable horizontalement, un outil de recherche distribué en temps réel et un outil d'analyse. Une base de données scalable horizontalement permet de rajouter des espaces de stockage sans ré-indexer la donnée existante.

MongoDB est une base de données NoSQL orientée documents ce qui signifie qu'elle stocke les données au format de documents JSON.

Chai est une bibliothèque d'assertions BDD / TDD qui nous permet de vérifier et comparer la valeur de variables ou de propriétés avec la valeur attendue. Différentes possibilités d'écriture, le style assert ou le style BDD décliné en deux colorations: expect et should.

Sinon est la bibliothèque la plus utilisée dans le monde de JavaScript pour générer des espions*, des bouchons et autres mocks (objets simulés). Sinon.js embarque même un ensemble d'assertions rendant son utilisation encore plus simple.

Mocha est un framework de test JavaScript riche en fonctionnalités exécuté sur node.js et le navigateur, ce qui rend les tests asynchrones simples. Ces tests s'exécutent en série, permettant des rapports flexibles et précis, tout en mappant les exceptions non capturées aux cas de test corrects.

Objectifs, Découpage prévisionnel

Octobre	Découverte et compréhension de l'architecture Lambda.
Nov-Déc	Écriture des composants de base de ce sous système : aggregation, aggregationManager.
Janvier-Fev	Gestion d'un double buffering sur le Speed layer.
Mars	Création des composants de communication entre le Batch layer et le Speed layer.
Avril	Intégration du Speed layer avec le service de consommation des évènements.
Mai	Le Speed layer est synchronisé avec le Batch layer pour effectuer des calculs sur un double buffer. Les calculs ne sont pas exploités par l'IHM.
Fin Mai	Validation et déploiement en production.
Bonus	Création d'un service externe pour gérer le Speed layer.
Fin mission	L'IHM bénéficie des données calculées par le Speed layer.

III. Réalisations

III.1 Les données

On a besoin de différents types de données afin de gérer les agrégations en temps réel, comme les données externes de configuration des événements et requêtes puis des données internes pour les agrégations et factory afin d'obtenir le format escompté. Dans un premier temps on va formater la donnée d'un événement puis celle d'une requête.

Liste de tous les événements disponible

```
2  export enum LambdaEventType {
3      openMedia = 'openMedia',
4      openLib = 'openLib',
5      openMediaReferrer = 'openMediaReferrer',
6      session = 'session',
7      libraryDuration = 'libraryDuration',
8      mediaDuration = 'mediaDuration',
9      shareMedia = 'shareMedia',
10     shareLib = 'shareLib',
11     pageDuration = 'pageDuration',
12     openPage = 'openPage',
13     buyOnline = 'buyOnline',
14     objectStorageSize = 'objectStorageSize',
15 }
16
```

Figure 10: Liste des événements disponibles de l'architecture lambda

```
107
108 export const openMediaCount: IAggregationOnEventSettings = {
109     buckets: [BucketGranularity.day, BucketGranularity.hour],
110     eventType: EventType.openMedia,
111     outputViewName: 'openMedia',
112     eventKey: ['libId', 'mediaId'],
113     output: {
114         count: {
115             aggregator: Aggregator.count,
116         },
117     },
118     enableSpeedLayer: true,
119 };

```

Figure 11: Définition de la configuration de l'agrégation 'OpenMedia'

Requêtes

```
184 export const viewQueriesSettings: IViewQueriesSettings = {
185   [QueryType[QueryType.openMedia]]: openMediaQuery,
186   [QueryType[QueryType.libraryDuration]]: libraryDurationQuery,
187   [QueryType[QueryType.mediaDuration]]: mediaDurationQuery,
188   [QueryType[QueryType.openPage]]: openPageQuery,
189   [QueryType[QueryType.shareMedia]]: shareMediaQuery,
190   [QueryType[QueryType.session]]: sessionQuery,
191   [QueryType[QueryType.visitor]]: visitorQuery,
192   [QueryType[QueryType.mediaSizeInByte]]: mediaSizeInByteQuery,
193   [QueryType[QueryType.mediaDurationByOpenMediaId]]: mediaDurationByOpenMediaIdQuery,
194   [QueryType[QueryType.pagesDurationByOpenMediaId]]: pagesDurationByOpenMediaIdQuery,
195   [QueryType[QueryType.openMediaByMediaId]]: openMediaByMediaIdQuery,
196 };
197
```

Figure 12: Liste des requêtes disponibles

```
54
55 const openMediaQuery: IViewQuerySettings = {
56   inputViewName: 'openMedia', // nom collection dans mongo
57   output: {
58     openMediaTotal: {
59       aggregator: QueryAggregator.add,
60       inputPropertyName: 'count',
61     },
62     openMediaHistogram: {
63       aggregator: QueryAggregator.buildHistogram,
64       inputPropertyName: 'count',
65     },
66     mediaIds: {
67       aggregator: QueryAggregator.buildProjectionAdd,
68       groupBy: 'mediaId',
69       inputPropertyName: 'count',
70     },
71   },
72 };
73
```

Figure 13: Définition de la configuration de la requête 'OpenMedia'

III.2 Implémentation

Les classes

Les responsabilités ont été séparées en différentes classes pour traiter les données du Speed layer.

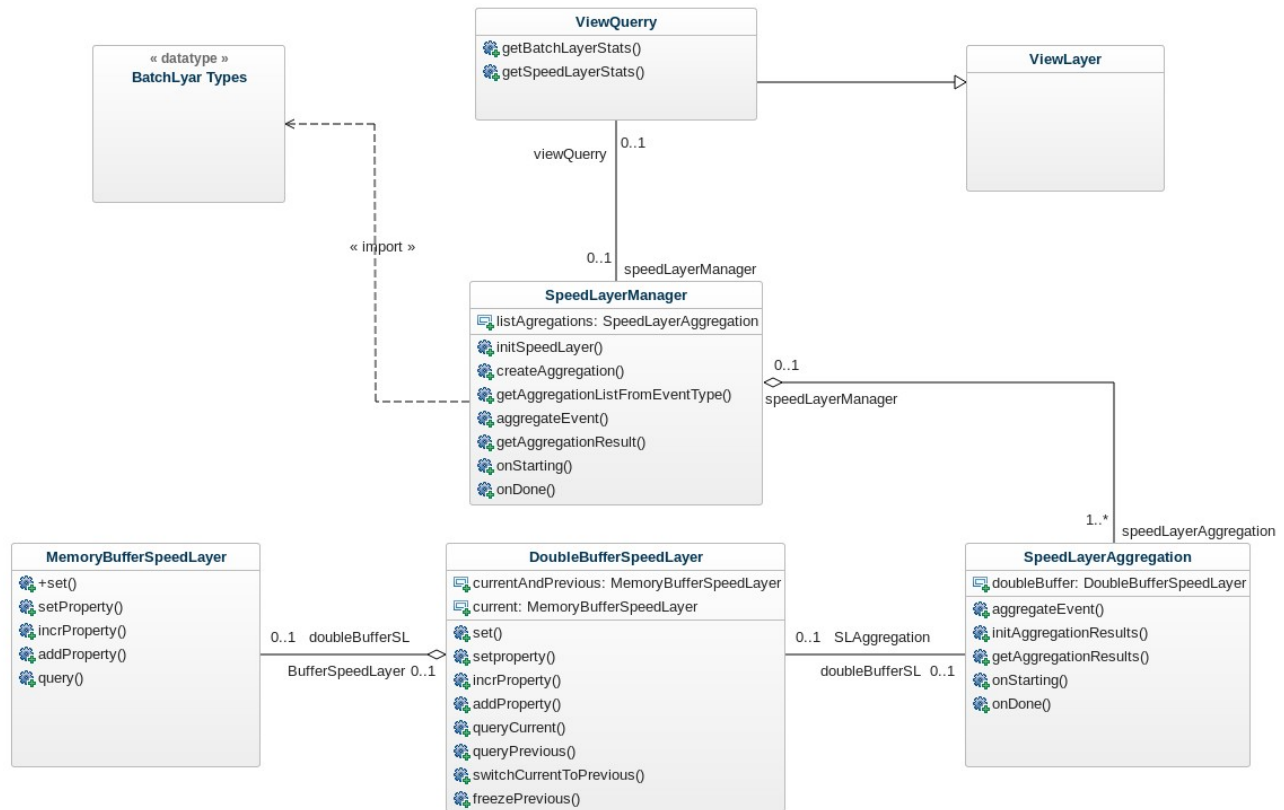


Figure 14: Diagramme de classes du Speed layer

Le point d'entrée du front se fait au niveau du service Library avec le fichier ViewQuery qui va appeler soit le View layer pour les données du Batch layer soit le Speed layer ou les deux.

On part du Speed layer puis à partir des données recueillies, le Batch layer ajoute ces données disponibles.

SpeedLayerManager

A l'arrivée d'un évènement en temps réel, l'objet de type SpeedLayerManager sélectionne les instances de SpeedLayerAggregation concernées par ce type d'évènement et transmet à chaque instance de la classe SpeedLayerAggregation l'évènement correspondant pour qu'il soit agrégé.

Afin de trouver les instances de SpeedLayerAggregation adéquates on consulte la liste des configurations d'agréations disponibles du Batch layer type, il faut aussi que l'option Speed layer soit activée.

Les configurations du Batch layer type non activées tels que *visitorAggregated* ou encore *objectStorageSize* nous permettent de garder dans un même fichier l'ensemble des configurations de l'architecture Lambda. Elles ne sont pas utilisées dans le cadre du Speed layer car nous n'avons pas encore eu la réflexion pour calculer les extrapolations de ces valeurs dans le Speed layer. Un calcul incrémental sur l'intervalle de temps connu du Speed layer n'est pas suffisant pour construire un résultat. Une autre raison de ne pas activer le Speed layer est liée au ratio coût/valeur ajoutée. Si le calcul temps réel coûte très cher, il faut évaluer la valeur ajoutée de ce calcul dans l'application. De même, pour certaines valeurs, nous n'avons pas la nécessité de calculer des valeurs en temps réel, par exemple pour le calcul de la performance des influenceurs...

Le manager est également en charge du routage des événements de synchronisation des calculs de chaque agrégation du Batch Layer vers chaque agrégation du Speed Layer.

Enfin, pour les requêtes le manager récupère le résultat de l'instance agrégée et la transmet au view Query qui renvoie le résultat vers le panneau de suivi des statistiques.

SpeedLayerAggregation

Dans cette classe s'opère la partie agrégation.

Les évènements sont récupérés avec l'instance créée par le manager afin de les agréger. On agrège libraryDuration en appelant la méthode AggregateEvent() sur l'évènement concerné.

Cette classe gère la synchronisation avec le Batch layer afin de savoir où il en est grâce aux fonctions onStart et onDone et pouvoir ainsi dispatcher la donnée agrégée dans le previous ou current buffer. Ces méthodes nous sont très utiles lorsque l'on arrive à la frontière entre la donnée agrégée et la donnée temps réel.

La synchronisation entre le calcul des agrégations dans le Speed layer et les instances du Batch associées s'opère à ce niveau-là.

Pour les requêtes, on récupère les informations stockées et c'est ici que l'on va agréger la donnée en fonction des opérateurs d'agrégation. On va donc construire une liste de résultat de valeurs agrégées pour chaque propriété.

Double buffer

Cette classe gère deux buffers de données, l'un représente l'heure courante et le second l'heure précédente. Elle ne connaît pas le code métier et les concepts d'évènements et d'agrégations. C'est une zone d'aiguillage pour le stockage des données vers l'un ou l'autre des buffers. Elle sait router les ajouts de données vers l'un des deux buffers et interroger chaque buffer pour extraire de la donnée.

Pendant toute la durée d'une agrégation la donnée est stockée dans les deux buffers, les opérations se font en double et cela permet de ne pas avoir à merger ultérieurement et d'éviter des bogues lors de la fusion d'une propriété plus complexe.

Buffer

Cette classe gère le stockage de la donnée en temps réel par type d'évènement lors de l'arrivée d'un évènement et l'extraction de cette donnée brute pour effectuer le calcul incrémentiel des agrégations impliquées dans la réponse à une requête. Elle s'occupe donc de compacter de la donnée en réalisant des calculs simples afin d'optimiser l'espace mémoire.

La notion de Buffer correspond à la mémoire tampon, zone située dans un disque dur ou dans la mémoire vive. Pour ce projet, le travail s'effectue en mémoire vive. Si le trafic est amené à augmenter fortement, nous envisagerons d'utiliser Redis* comme stockage afin de partager le buffer entre différentes instances du Speed layer.

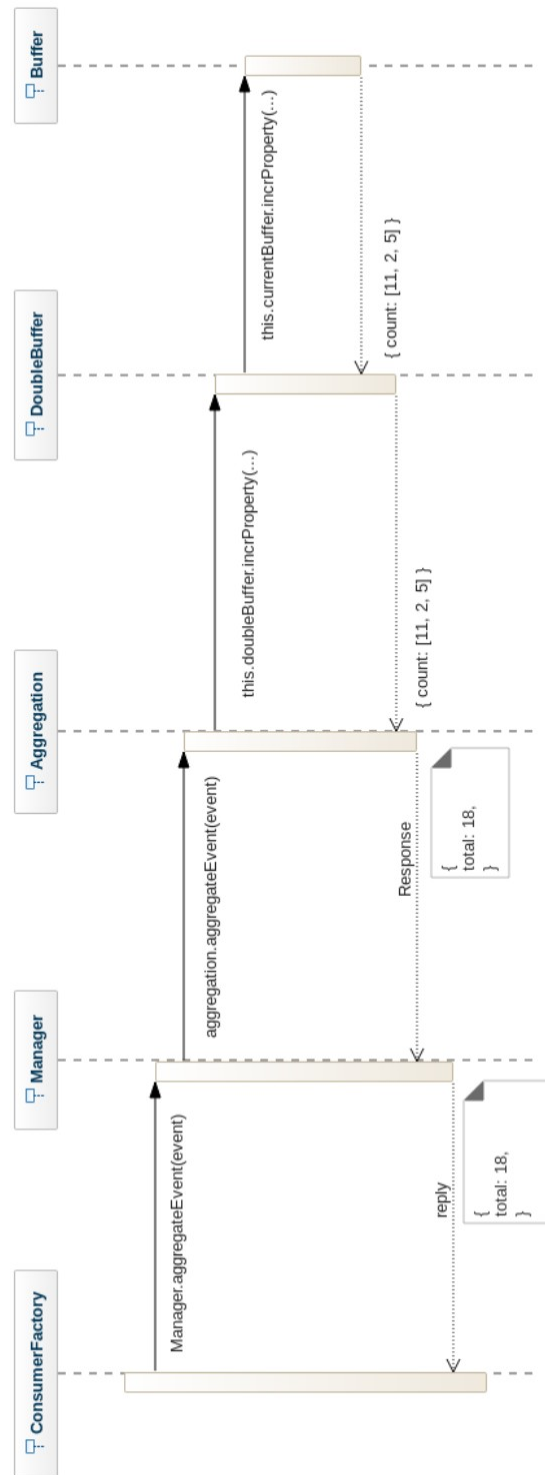
Dans notre cas les buffers nous permettent de stocker nos résultats du Speed layer car on ne veut pas les conserver dans le temps, le travail se fait au travers de données éphémères. On travaille au maximum sur deux heures et plus précisément, sur les deux dernières heures, l'heure précédente et l'heure courante.

Lorsque le calcul commence sur la troisième heure, on supprime l'heure précédente, l'heure en cours devient l'heure précédente et l'heure en cours se réinitialise. Ce concept évite l'encombrement des données. Le fonctionnement du buffer permet de stocker des données sur une courte période et conserver les données avant leur utilisation.

Les entrées du speed layer

Découverte du fonctionnement du speed layer à travers les 3 points d'entrées des flots d'exécutions du speed layer

Figure 17 - Agréger un évènement



2. Réponse à une requête avec en entrée queryId, filtre, date début, date de fin [Figure 18].

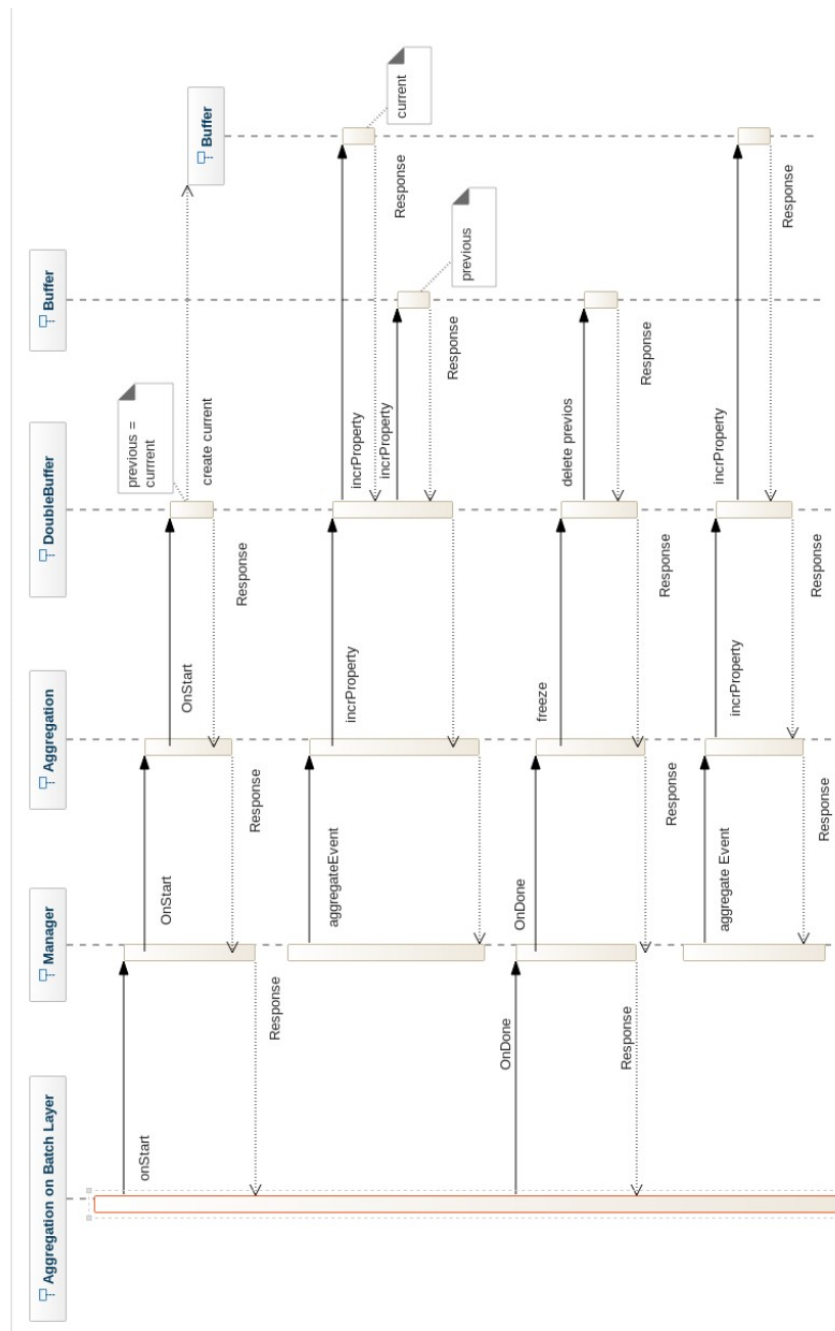


Figure 18 - Parcours d'une requête

3. Synchronisation du Batch layer vers Speed layer pour informer ce dernier qu'une agrégation est en cours ou se termine [Figure 19]

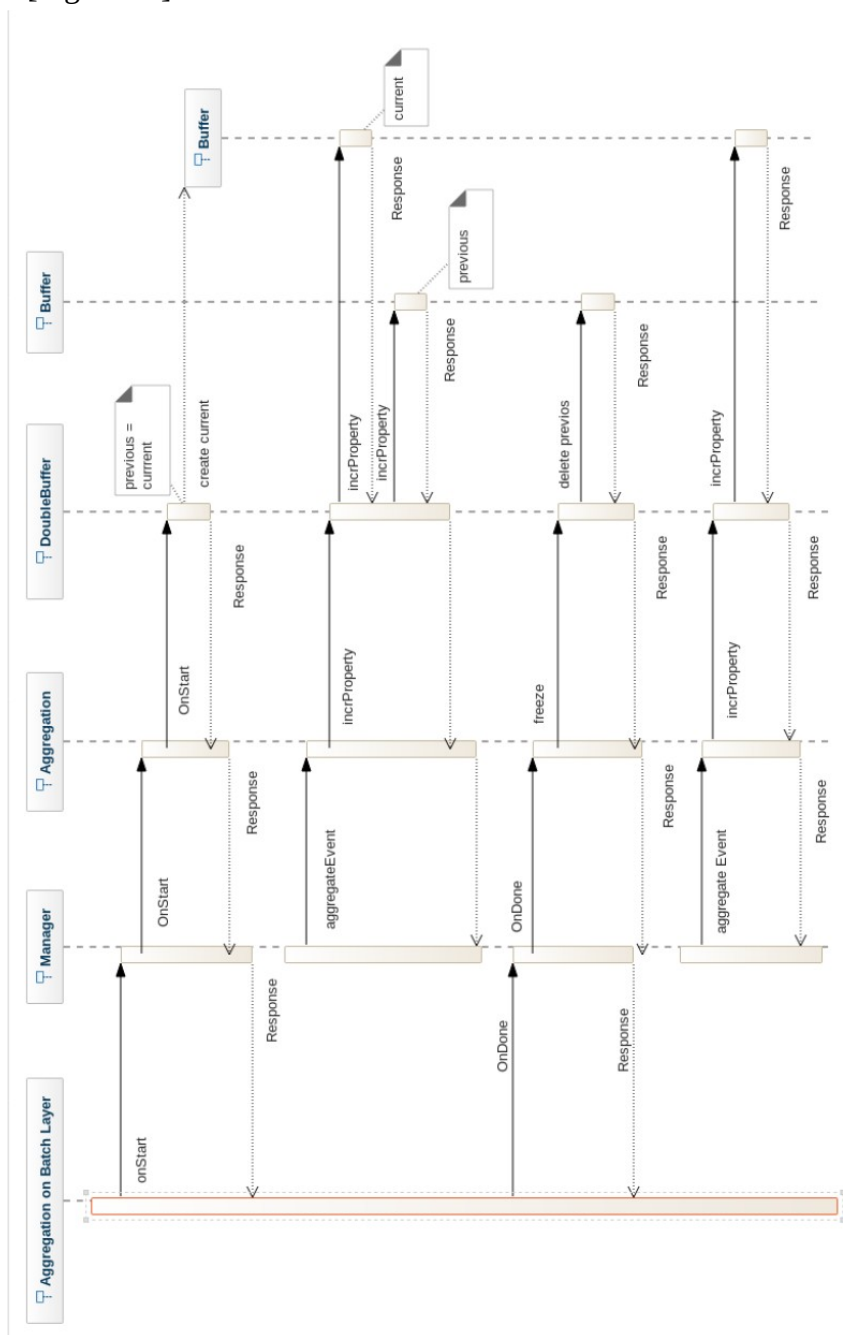


Figure 19 – Synchronisation entre le Batch et le Speed layer.

Retrouvez en [annexe 4](#) p43 le cheminement d'un évènement et d'une requête, vue complémentaire de ces trois diagrammes de séquences.

III.3 Interfaces

Une interface définit un ensemble de méthodes sans leurs implémentations et les classes associées seront tenues d'utiliser les méthodes de l'interface mise en place. Cela permet de définir des contrats.

On distingue quatres interfaces dans le Manager, l'interface d'initialisation, des évènements, du Batch layer et des requêtes.

```
25 export interface IInitSpeedLayer {
26   initSpeedLayer(aggregationsSettings?: ISpeedLayerAggregationSettings): void;
27
28   createOneAggregation(aggregationSettings: IAggregationSettings): SpeedLayerAggregation;
29 }
30
31 export interface ISpeedLayerAggregation {
32   aggregateEvent(event: IEvent): IMapAggregation;
33 }
34
35 export interface IQuerySpeedLayer {
36
37   getAggregationResult(query: IViewQuerySettings, params: IFilterKeys, startTimestamp: number, endTimestamp: number): IViewSummary;
38 }
39
40 export interface ISyncSpeedLayerFromBatchLayer {
41
42   onStartCurrentBucketAggregationOnBatchLayer(aggregationType: AggregationType, timestampBeginningBatchLayerAggregation: number): void;
43
44   onCurrentBucketAggregationDoneOnBatchLayer(aggregationType: AggregationType, timestampEndingBatchLayerAggregation: number): void;
45 }
46
47 export interface ISpeedLayerAggregationManager extends IInitSpeedLayer, ISpeedLayerAggregation, IQuerySpeedLayer, ISyncSpeedLayerFromBatchLayer {}
```

Figure 20: Interfaces externes du manager

III.4 Test development driven

Définition

Le TDD (Test Driven Development) est une technique de développement mêlant l'écriture des tests unitaires, la programmation et l'amélioration continue du code (encore appelée refactorisation*). La méthode traditionnelle de la rédaction des tests unitaires consiste à rédiger les tests d'une portion d'un programme (appelé unité) afin de vérifier la validité de l'unité implémentée. On rédige donc les tests unitaires avant de procéder à la phase de codage.

Le cycle de développement préconisé par TDD comporte cinq étapes comprenant l'écriture d'un test, exécuter ce test et vérifier qu'il échoue (car le code qu'il teste n'a pas encore été implémenté), puis écrire l'implémentation pour faire passer le test, pour ensuite exécuter les tests afin de contrôler que les tests passent et enfin remanier (Refactorer) du code afin d'en améliorer la qualité tout en conservant les mêmes fonctionnalités.

Intérêts

Commencer par les tests permet de s'assurer que leur écriture ne sera remise à plus tard voir pas du tout. Cela nous pousse à penser aux détails de la méthode dont on a besoin pour écrire la spécification. On va également s'interroger sur le nom de la méthode, sa valeur de retour, ses paramètres, son comportement. Cela permet de clarifier la conception et d'écrire seulement du code utile. Le fait de disposer d'un grand nombre de test permet de s'assurer de la solidité et la garantie du code. Lorsqu'on modifie une méthode existante, on peut relancer les tests unitaires afin de s'assurer que sa modification n'a pas impacté l'existant, cela offre donc un feedback immédiat.

Mise en situation dans le Speed layer

```
902
903 it( expectation: 'Should manage several values when aggregation has several output', callback: () => {
904     const speedLayer = aggregationFactory(aggregationSettingsWithTwoResults);
905     const { doubleBufferForTest } = createStubDoubleBufferFactory(speedLayer);
906     const spyDoubleBufferIncrProperty = sinon.spy(doubleBufferForTest, 'incrProperty');
907     const spyDoubleBufferAddProperty = sinon.spy(doubleBufferForTest, 'addProperty');
908     const eventMediaDuration: IEvent = {
909         timeStamp: 1,
910         eventType: EventType.mediaDuration,
911         libId: 'libId1',
912         mediaId: 'mediaId1',
913         duration: 123,
914     };
915
916     const aggregationsStep1: IEventAggregation = speedLayer.aggregateEvent(eventMediaDuration);
917     expect(aggregationsStep1).toEqual( { value: {
918         count: 1,
919         durationSum: 123,
920     } });
921     const aggregationsStep3: IEventAggregation = speedLayer.aggregateEvent(eventMediaDuration);
922     expect(aggregationsStep3).toEqual( { value: {
923         count: 2,
924         durationSum: 123 * 2,
925     } });
926     sinon.assert.calledTwice(spyDoubleBufferIncrProperty.withArgs({
927         libId: 'libId1',
928         mediaId: 'mediaId1',
929     }, 'count'));
930     sinon.assert.calledTwice(spyDoubleBufferAddProperty.withArgs({
931         libId: 'libId1',
932         mediaId: 'mediaId1',
933     }, 'durationSum', 123));
934     const filter: IFilterKeys = { libId: [], mediaId: [] };
935     const queryOutput = {
936         count: createOutputDescription(QueryAggregator.add, { inputPropertyName: 'count' },
937         totalDuration: createOutputDescription(QueryAggregator.add, { inputPropertyName: 'durationSum' },
938     });
939     expect(_.invoke(speedLayer, { path: 'getAggregationResultForCurrentBucket', queryOutput, filter })).toEqual( { value: {
940         count: 2,
941         totalDuration: 123 * 2,
942     } });
943 }
```

Figure 21: Test unitaire de la méthode `aggregateEvent()`

Je devais implémenter les nouvelles méthodes à partir des tests unitaires créés par mon tuteur.

Explication du test unitaire ci-dessus :

La fonction `aggregateEvent()` doit gérer plusieurs valeurs lorsque l'agrégation comporte plusieurs sorties. On crée une instance de Speed layer avec sa factory. On appelle `incrProperty` et `addProperty` sur le double buffer.

On définit l'événement `media duration`. On appelle deux fois la fonction du test sur l'événement `media duration`. On vérifie avec `"assert.calledTwice"` les appels des 2 méthodes `incrProperty` et `addProperty`. Les filtres ici sont vides donc on prend toutes les agrégations. On attend en retour un objet contenant deux propriétés la première `count` et la seconde `totalDuration`.

Code issu du test unitaire:

```
80
81 public aggregateEvent(event: IEvent): IEventAggregation {
82
83     let result = {};
84
85     const resultOutput = this.settings.output;
86
87     const keyIncr = _.pick(event, this.settings.eventKey);
88
89     _.mapValues(resultOutput, (value, key, object) => {
90
91         switch (value.aggregator) {
92             case Aggregator.count:
93                 return result = this.doubleBuffer.incrProperty(keyIncr, key);
94
95             case Aggregator.add:
96                 result = this.doubleBuffer.addProperty(keyIncr, key, event[value.inputPropertyName]);
97
98                 return result;
99             case Aggregator.exists:
100                 return result = this.doubleBuffer.setProperty(keyIncr, key, {value: 1});
101
102             default:
103                 return result;
104         }
105     });
106
107     return result;
108 }
```

Figure 22: Code de la méthode aggregateEvent()

Dans cette fonction on crée une variable de résultat à laquelle on affecte un objet vide. On parcourt les sorties des settings avec la méthode lodash « mapValues » qui va regrouper le résultat des clés identiques pour ensuite comparer la clé de l'évènement avec l'opérateur d'agréations et appliquer la méthode adéquate construite dans le double buffer.

III.5 Intégration

Schéma architecture du Speed layer service

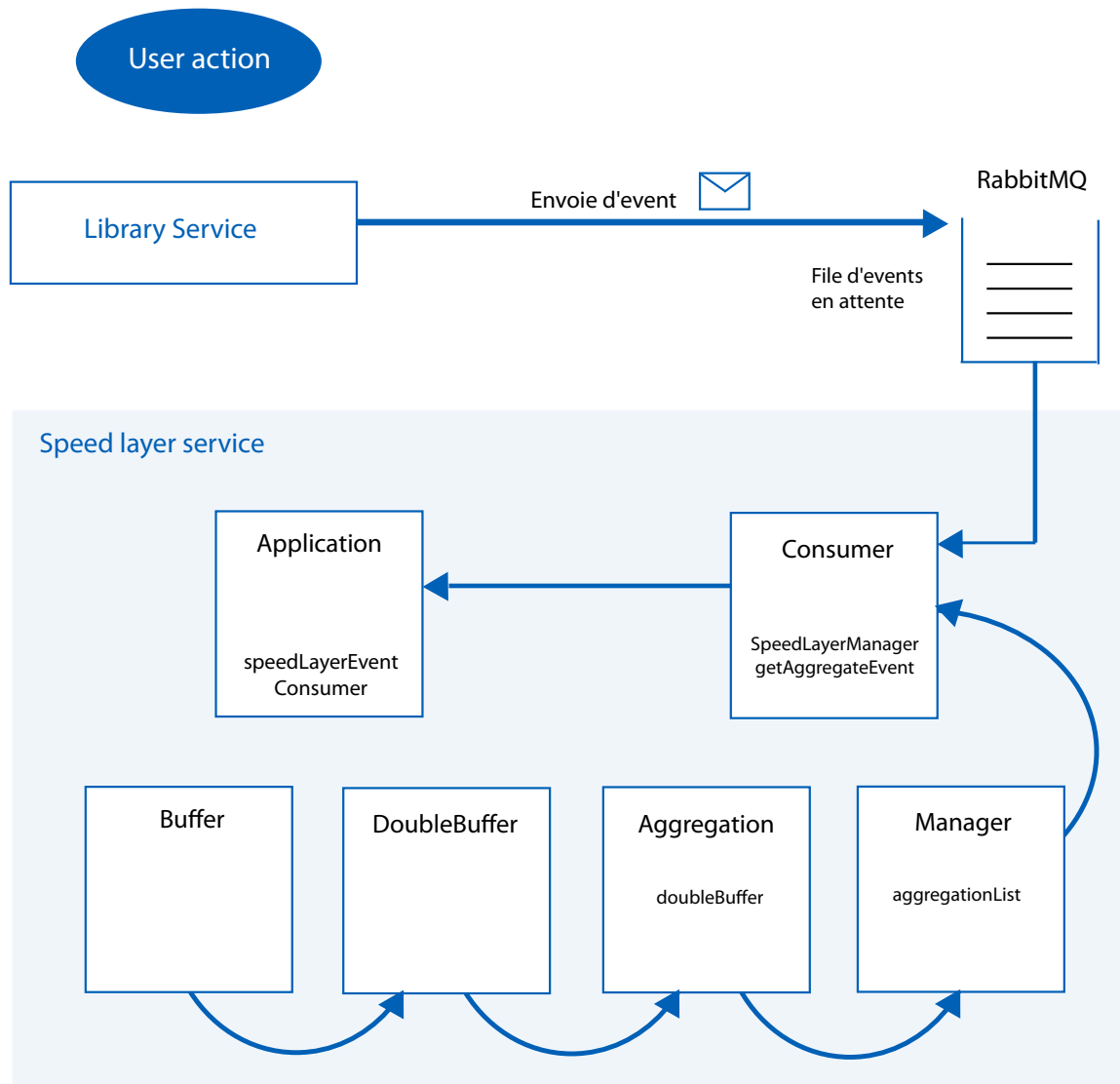


Figure 15: Schéma d'architecture du service Speed layer

La librairie envoie une requête sur le système de statistiques via un appel HTTP et des appels RPC. Le gestionnaire des appels RPC pour l'exécution des requêtes se nomme ViewQuery. Les paramètres de la requêtes sont l'identifiant de la requête, les filtres à appliquer sur la donnée et la période concernée par cette requête. Ainsi, le ViewQuery fera appel au Batch Layer uniquement ou au Batch et au Speed

Layer. Dans ce dernier cas, il devra fusionner les résultats obtenus par le Batch Layer avec les données temps réel du Speed Layer.

Etapes 1

On a créé le service Speed layer qui contient l'application, le consommateur d'événements et ajouté les classes du Speed layer puis on a relié ce service aux statistiques globales de l'application et aux dockers.

Etapes 2

Dans cette étape on vient créer les nouvelles routes et câbler le Speed layer à RabbitMQ.

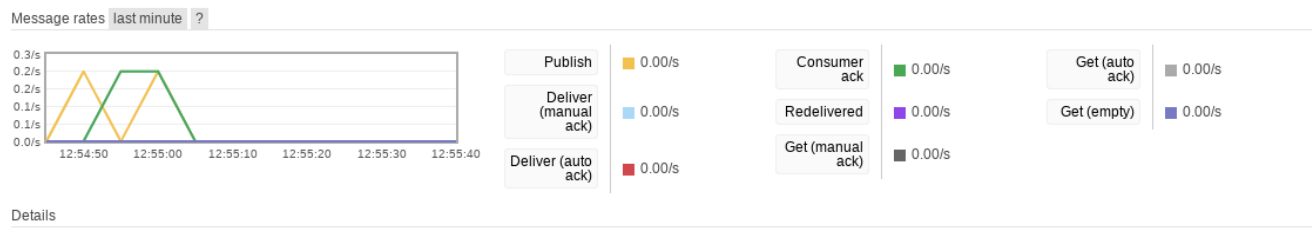


Figure 24: Extrait de RabbitMQ lors de l'ouverture d'un document sur un hub

La partie ascendante de la courbe nous montre la prise en charge de l'événement par RabbitMQ qui le transmet au Consommateur et la partie descendant représente la prise en charge de l'événement par le Speed layer.

Etape 3

On a intégré les 3 interfaces, l'agrégation des événements, la gestion des requêtes et la synchronisation des calculs Batch layer / Speed layer dans le service Speed layer et on a ajouté environnement client/serveur.

Fonctionnalités gérées par mon tuteur

- Ajout d'un feature flag afin de pouvoir activer/désactiver le Speed layer sur un hub. Ceci permet d'isoler nos développements en encapsulant justement tous ces développements dans ce qu'on appelle des fonctionnalités et ainsi de pouvoir les activer ou non à la demande de l'utilisateur. Très utile pour tester l'application en interne.
- Ajout d'un autre feature flag pour activer / désactiver le Speed layer dans bee-cluster
- Désactivation la persistance des événements de statistiques reçus par le Speed layer dans rabbitMQ :

La donnée temps réel ne doit pas être gardée sur du long terme, contrairement aux données du Batch layer, et sera effacée une fois la donnée accessible dans le Batch layer afin d'éviter toutes confusions avec une donnée ancienne qui serait gardée par défaut. Pour se faire on a besoin de spécifier à RabbitMQ qu'une persistance des données n'est pas utile puisque cela se fait par défaut.

- Création d'un `helmChart` Speed layer pour le déploiement.

III.6 Objectifs atteints, résultats obtenus

Objectifs réalisés

Le Speed layer est désormais disponible sur nos docker locaux et en production sur mon hub. Cela va nous permettre de réaliser une batterie de tests applicatifs et d'adapter le code suivant les résultats obtenus.

Les objectifs ont donc été atteints puisqu'il est désormais possible de voir la différence entre des statistiques avec et sans Speed layer.

Changements en cours de processus :

Des changements ont eu lieu au cours de ce projet puisque nous avons travaillé de manière itérative.

Le premier changement est la mise en place du buffer et du Double buffer avec l'arrivée du previous. Au début il était prévu de travailler seulement avec un current bucket géré dans le SpeedLayerAggregation et on s'est rendu compte qu'avec le temps d'une agrégation on avait un recouvrement à prendre en compte et que nous ne pouvions supprimer la donnée du Speed layer sans avoir la certitude de perdre des données. Le temps de recouvrement correspond au temps d'une agrégation. Ces deux buffers nous permettent dans un premier temps de supprimer le previous puis celle du current après s'être assuré que le Batch layer ait fini de les calculer. Ce qui nous a amené à créer deux nouvelles classes, le buffer et Double buffer afin d'alléger le code du SpeedLayer pour plus de clarté dans le code et également dans les tests.

L'utilisation de Redis* se fera plus tard sachant que la donnée est conditionnée au format clé valeur cela nous permettra une mise en place rapide. Pour l'instant la donnée du Speed layer est stockée en mémoire.

La première implémentation du code dans le ViewQuery consistait à récupérer en parallèle les données du Speed layer et du Batch layer puis de les merger. Maintenant on part des données du Speed layer et le Batch layer vient compléter les données déjà emmagasinées par le Speed layer. Ceci nous évite un merge qui pourrait mal se passer et aussi de réduire fortement le code.

IV. Conclusion

En conclusion cette immersion dans l'architecture Lambda fût très enrichissante puisqu'elle reposait sur la création d'un nouveau service avec ses classes, son environnement client/serveur et ses interactions avec de nombreux services.

La réalisation et l'intégration du Speed layer, à la fois complexe et très intéressant a été mené à bien, les objectifs d'intégration de ce sous composant, déployé sur les dockers locaux et en production sur mon hub étant respectés.

Lors de cette mission j'ai bénéficié d'un encadrement technique et humain poussé, cela m'a permis de prendre confiance et d'appréhender cette mission au mieux.

Suite au confinement, la charge de travail est restée la même et le télétravail a ajouté des difficultés avec la non possibilité de poser mes questions en direct aux membres de l'équipe. Mon tuteur entreprise a été très présent et beaucoup d'échanges avec lui ont palliés le manque d'échanges avec le reste de l'équipe.

Le rythme soutenu des cours ainsi que les flots d'informations à emmagasiner rapidement m'ont demandé de la rigueur et beaucoup d'investissement avec lesquels j'ai pu acquérir des automatismes et une plus grande rapidité d'exécution.

Les notions de développement apprises en cours et en entreprise ont été complémentaires puisque les langages utilisés étaient différents. Réaliser des projets de zéro en cours a été bénéfique afin de mieux appréhender l'orienté objet car le code en entreprise est dispatché dans de nombreux services.

Je suis satisfaite de cette mission qui m'a permis de voir processus de réalisation et d'intégration d'un composant dans un système donné.

V. Glossaire

Interactive Smart Signage : Application de diffusion de contenus interactifs, accessible via un hub* et faisant partie de l'écosystème Ustream*. Elle est utilisée sur des écrans mis à disposition dans des points de vente et permet de créer du contenu personnalisé en fonction des stocks à écouler. Ces programmes/playlists publicitaires personnalisés sont diffusables à la demande sur tous types d'écrans, y compris sur les terminaux mobiles.

Espions : Les espions sont des fonctions qui enregistrent un certain nombre de paramètres, comme le nombre de fois qu'elles ont été appelées, avec quels paramètres...
Ils sont utiles pour tester le comportement de composants qui manipulent des fonctions, comme un émetteur d'événements.

Fullstack : Un développeur intervenant sur le back-end et le front-end d'un site Web ou d'une application.

GMT : signifie "Greenwich Mean Time" (temps moyen de Greenwich). Il s'agit de l'heure locale calculée à l'observatoire astronomique de Greenwich, situé près de Londres en Angleterre. Cette heure sert de référence dans le monde entier (on dit par exemple "GMT+5h").

Redis est une base clé-valeur en mémoire, qui peut éventuellement persister sur disque les données. Redis est classé dans la catégorie des bases NoSQL, une solution open-source codée intégralement en C. Outil très rapide. Il est possible d'attribuer une durée de vie aux données insérées, ce qui permet de mettre en place un système de cache. Il est possible d'utiliser des streams pour travailler sur l'ensemble des données qui sont stockées. On peut le partitionner avec des préfixes pour regrouper/isoler de la donnée.

Refactorisation : Permet d'assurer un suivi de l'existant, de faire un ménage qui en facilitera la maintenance. La refactorisation se traduit par éliminer du code mort, documenter, renommer et optimiser du code.

Spec : Les spécifications décrivent le fonctionnement du dispositif digital. L'UX Designer* et l'UI Designer* spécifie le comportement de chaque écran de l'interface utilisateur. On clarifie la réponse que l'interface doit apporter lorsque l'utilisateur réalise une action ou active une fonction. Il faut détailler tous les interactions à l'écran, c'est à dire tout ce qui peut se passer sur le plan interactionnel dans la manipulation de l'interface utilisateur. Elles peuvent aussi bien s'appuyer sur des maquettes fonctionnelles (wireframe*) que sur des maquettes graphiques.

Le périmètre couvert par la spécification fonctionnelle se rapporte au fonctionnement de l'interface côté utilisateur.

Le périmètre couvert par la spécification technique concerne le fonctionnement technique de l'interface côté administrateur, *en back-office*

UI Designer : S'occupe du traitement graphique du wireframe* et applique une charte. Il intervient quand la maquette est réalisée avec les recommandations de l'Expérience Utilisateur placées par l'UX designer dans le maquetage. Il se charge ainsi de réaliser une interface agréable et utile pour les utilisateurs. Plus axé sur le graphisme et la création, il conçoit et positionne les éléments graphiques et contenu texte d'une interface web par exemple.

Ux Designer : Améliore et optimiser l'Expérience Utilisateur quelque soit leur support (smartphone, tablette, grands écrans...). Il peut être amené à faire des interviews pour comprendre les besoins et les habitudes des utilisateur par des questionnaires.

Wireframe : Maquette fonctionnelle est un schéma utilisé lors de la conception d'une interface utilisateur pour définir les zones et composants qu'elle doit contenir. À partir d'un wireframe peut être réalisée l'interface proprement dite par un graphiste. La démarche de recourir à des wireframes s'inscrit dans une recherche d'ergonomie. Elle est surtout utilisée dans le cadre du développement et des sites et applications Web. Le wireframe consiste concrètement en un croquis, un collage papier ou un schéma numérique.

VI. Sitographie

Site de Beebuzziness :

<https://ubstream.com/>

Lexique Agile Scrum

<https://agiliste.fr/lexique-agile-scrum/>

Méthode agile

<https://toucantoco.com/blog/methodes-agiles-kanban-revolutionner-management/>

<https://www.ideematic.com/actualites/2015/01/methodes-agiles-definition/>

BDM :

<https://www.blogdumoderateur.com/definition-developpement-web/ss>

rabbitMQ

<https://www.rabbitmq.com/>

TDD :

<https://putaindecode.io/articles/se-lancer-dans-le-tdd/>

UX Designer / UI Designer :

<https://www.journalducmm.com/metier-ux-designer/>

Sinon.js

<https://www.editions-eni.fr/open/mediabook.aspx?idR=39ae4beb4e4edad9da04dfda4034eb36>

VII. Annexes

A. Panneau de statistiques détaillé:

Les informations sont classées en trois catégories : Audience, consultation, engagement.

Audience

- * Visites
- * % de visiteurs sur mobiles et tablettes
- * Durée moyenne des visites
- * Moyenne de visites par visiteur

Consultation



Figure 17: Statistiques disponibles dans la catégorie Consultation

Audience



Figure 16: Statistiques disponibles dans la catégorie Audience

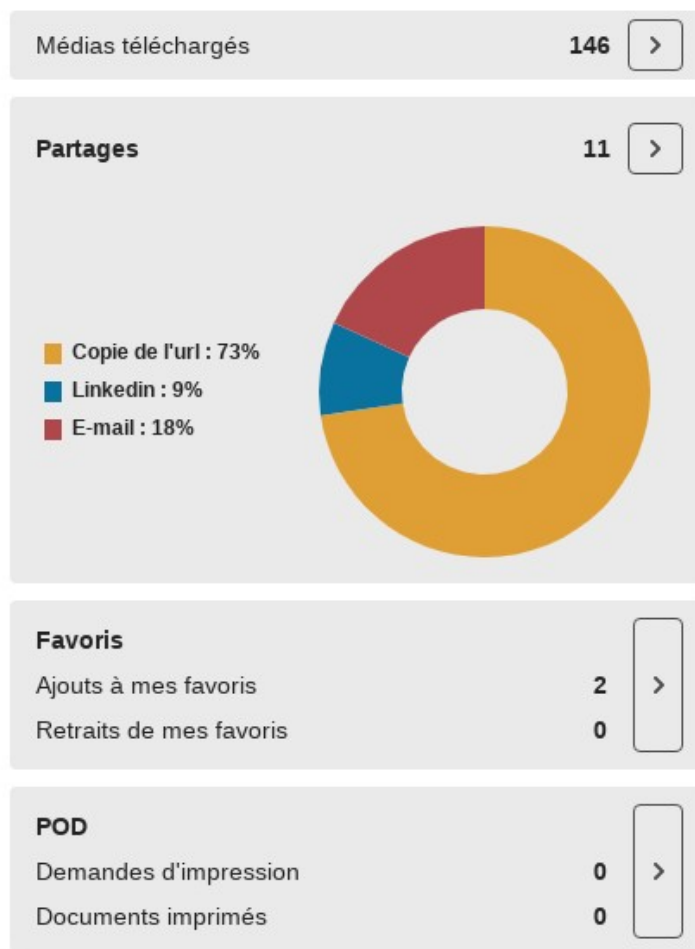
Consultations

- * Consultations des médias
- * Médias consultés au moins une fois
- * Médias les plus consultés / pages les plus consultées (stat d'un média de type document)
- * Confirmation de lecture
- * Médias déposés

Engagement

- * Médias téléchargés
- * Partages
- * POD
- * Prise de contact
 - Demande de rappel
 - Appel audio
 - Appel vidéo

Engagement



Abonnements

Numérique	
Abonnements	28
Désabonnements	1

Figure 18: Statistiques disponibles dans la catégorie Engagement et Abonnements

[Retour partie statistiques](#)

B. Voici la liste des statistiques futures rendu possible avec la mise en place de l'architecture Lambda:

- Trier les médias d'un hub par nombre de consultations ou de partage
- Consulter les statistiques de médias détaillées par variante (langue) d'un même média
- Consulter les statistiques d'une sélection manuelle de médias
- Consulter les statistiques globales de Ustream
- Consulter le parcours d'un utilisateur et son historique
 - o Nombre de sessions
 - o Géolocalisation
 - o Devices utilisés
 - o Durée des sessions
 - o Contenu des sessions
 - Consultations de médias
 - Durée de consultation
 - Partages / Ajout aux favoris
 - Changement de hub
 - Changement de collection
- Suivre le parcours de dissémination (Lib-it/Ajout aux favoris) d'un média entre différents hubs et pouvoir calculer des éléments de rétribution (Rétribution d'un influenceur générant beaucoup de consultations/partages/... de médias appartenant à des marques depuis son hub.

C. Statistiques

Anciennes organisations

L'ancien système de statistiques datant de 2014 arrivait à ses limites car l'écriture des données et le calcul à la demande des agrégations sur une même base était très coûteuse en puissance machine et le format de données qui avait été créé était peu évolutif.

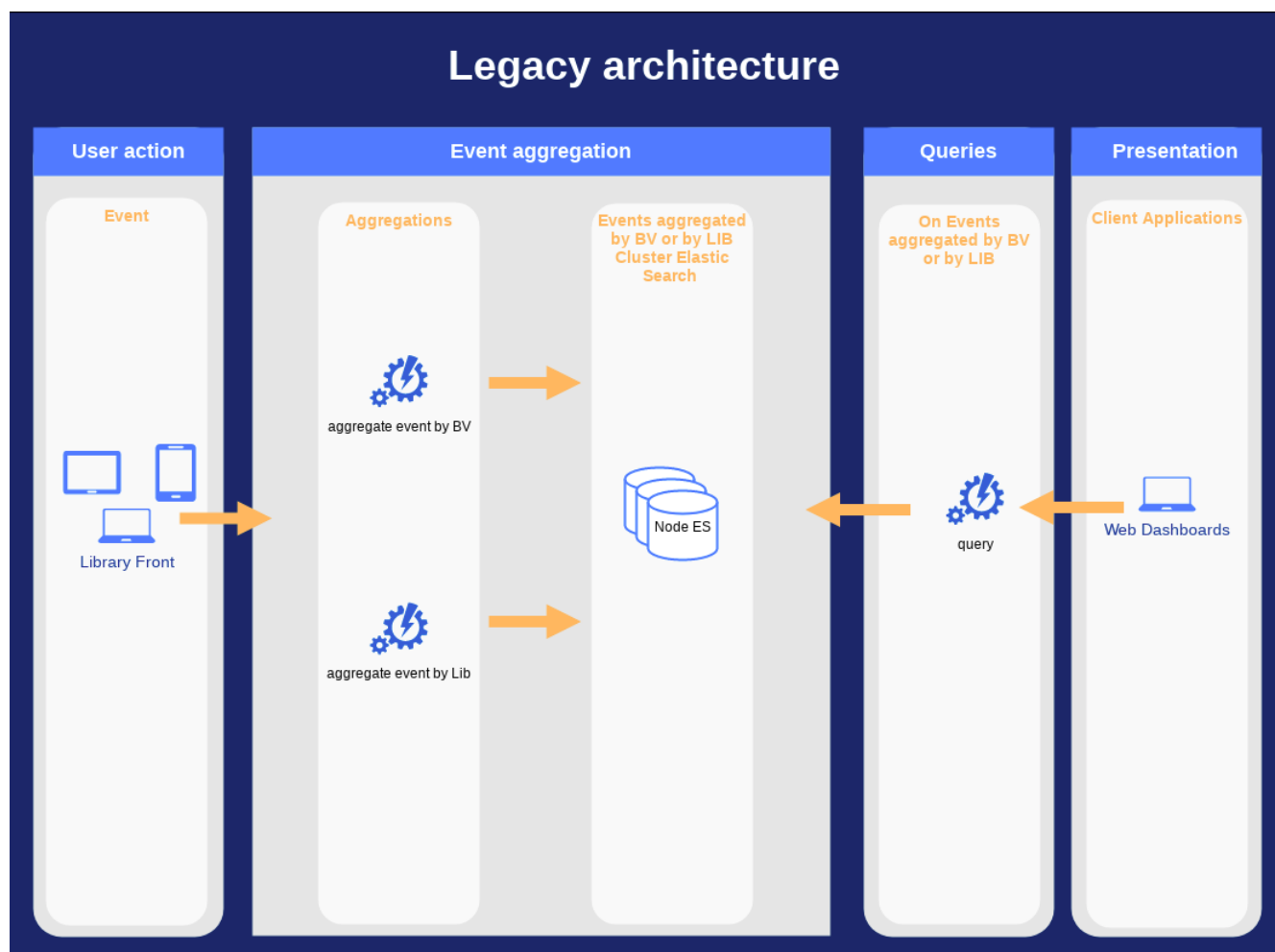


Figure 19: Modèle utilisé avant la mise en place de l'architecture Lambda

Ce schéma représente la nouvelle organisation du système de statistiques sans le sous-composant Speed layer. Les événements côté Aggregation layer et les requêtes côté View layer sont traités séparément avec l'intégration de RabbitMQ pour réguler les événements entrants avec un mode de file d'attente.

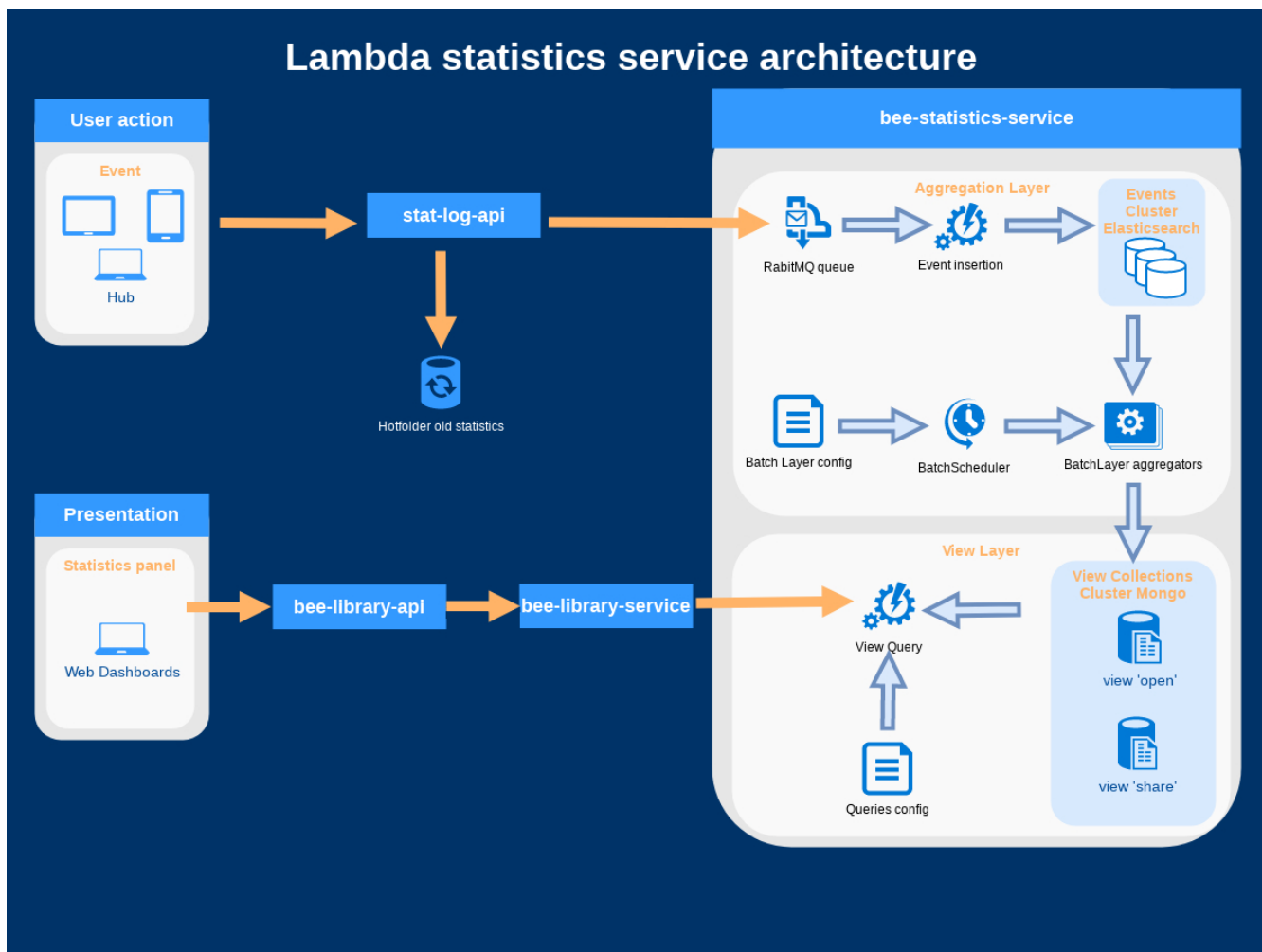


Figure 20: Architecture Lambda sans le traitement de données en temps réel

Le Batch layer est sollicité plus régulièrement sans le Speed layer, environ toutes les minutes alors qu'avec l'intégration du Speed layer il sera sollicité toutes les heures.

[Retour chapitre architecture Lambda](#)

Agrégations

```
5 export enum AggregationType {
6     shareMedia = 'shareMedia',
7     openMediaRefererCount = 'openMediaRefererCount',
8     openMediaCount = 'openMediaCount',
9     libraryDuration = 'libraryDuration',
10    mediaDuration = 'mediaDuration',
11    openPageCount = 'openPageCount',
12    session = 'session',
13    visitor = 'visitor',
14    visitorAggregated = 'visitorAggregated',
15    objectStorageSize = 'objectStorageSize',
16    logicalDatastoreSize = 'logicalDatastoreSize',
17    mediaDurationByOpenMediaId = 'mediaDurationByOpenMediaId',
18    pageDurationByOpenMediaId = 'pageDurationByOpenMediaId',
19    openMediaCountByMediaId = 'openMediaCountByMediaId',
20
21 }
```

Figure 21: Liste des types d'agrégations disponibles

Factory

```
21 const baseEventFactory = (item, now, eventType: EventType): ILambdaBaseEvent => {
22     const { sessionId, tracking } = item;
23     const baseEvent: ILambdaBaseEvent = {
24         sessionId,
25         eventType,
26         timeStamp: getTimeStamp(item, now),
27     };
28     if (tracking) {
29         const { id, isAnonymous } = tracking;
30         baseEvent.tracking = { id, isAnonymous };
31     }
32     return baseEvent;
33 };
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58 const openMediaFactory: EventFactory = (item, now = Date.now): IOpenMediaEvent => {
59     const { libId, mediaId, documentId, context, mediaType } = item;
60     return {
61         ...baseEventFactory(item, now, EventType.openMedia),
62         libId,
63         mediaId,
64         documentId,
65         context,
66         mediaType,
67     };
68 };
```




D Cheminement des évènements à un instant T

Ce fichier excel permet de visualiser les différents cas possible lorsqu'un évènement est déclenché en détaillant les différents statuts du Batch layer et du Speed layer à un moment donné.

Cheminement de l'évènement « OpenMedia » :

Spécification cas pratiques liées à un évènement : OpenMedia					
Jour J	Time	Batch Layer = BL	Event from BL	Speed Layer Current = SPC	Speed Layer Previous = SPP
10-01-2020	9h59	[origine BL .. 9h [[9h .. Time [[8h .. 9h [
10-01-2020	10h00	[origine BL .. 9h [Starting Computing BL	[10h .. Time [[9h .. 10h [
10-01-2020	10h01	[origine BL .. 9h [[10h .. Time [[9h .. 10h [
10-01-2020	10h02	[origine BL .. 10h [BL Compured	[10h .. Time [[9h .. 10h [
10-01-2020	10h15	[origine BL .. 10h [[10h .. Time [[9h .. 10h [
10-01-2020	10h59	[origine BL .. 10h [[10h .. Time [[9h .. 10h [

On peut regrouper les colonnes en différentes catégories :

-  Temps qui passe
-  État du Batch layer et du Speed layer à l'instant T
-  Évènement du Batch layer envoyé au Speed layer

Cas 1 => Evènement pris en charge par le current Speed layer

Colonne 1-2 : Lorsque l'évènement est déclenché le 10 janv à 9h59

Colonne 3 : Le Batch layer travaille sur la plage horaire de l'origine jusqu'à 9h

Colonne 4 : Le Batch layer est en cours de traitement sur cette heure (9h à 10h02) donc n'envoie pas l'évènement au Speed layer.

Colonne 5 : C'est le current buffer qui va traiter cet évènement car il travaille de 9h à 10h02

Colonne 6 : Le previous buffer a travaillé de 8h à 9h

Cas 2 => Evènement pris en charge par le current Speed layer

Colonne 1-2 : Lorsque l'évènement est déclenché le 10 janv à 10h00

Colonne 3 : Le Batch layer travaille sur la plage horaire de l'origine jusqu'à 9h

Colonne 4 : Le Batch layer démarre une nouvelle plage horaire et envoie l'évènement onStarting au Batch layer au Speed layer ce qui va autoriser le switch des données du current au previous et vider le current – On est autorisé à changer de buffer.

Colonne 5 : Le current travaille désormais de 10h à maintenant.

Colonne 6 : Le previous a travaillé de 9h à 10h.

Cas 3 => Event pris en charge

Colonne 1-2 : Lorsque l'évènement est déclenché le 10 janv à 10h01.

Colonne 3 : Le B. travaille sur la plage horaire de l'origine jusqu'à 9h.

Colonne 4 : Le B. est en cours de traitement sur cette heure (9h à 10h02) donc n'envoie pas d'event au Speed layer.

Colonne 5 : C'est le Current qui va traiter cet event car il travaille de 10h à maintenant.

Colonne 6 : Le previous a travaillé de 9h à 10h.

Cas 4 => Event pris en charge

Colonne 1-2 : Evènement est déclenché le 10 janv à 10h02.

Colonne 3 : le Batch layer travaille sur la plage horaire de l'origine jusqu'à 10h

Colonne 4 : Le Batch layer a terminé le traitement jusqu'à 10h incluse et envoie l'évènement onDone au Speed layer . A ce moment Le contenu du current est copié dans le previous.

Colonne 5 : C'est le Current qui va traiter cet évènement car il travaille de 10h à maintenant.

Colonne 6 : Le previous a travaillé de 9h à 10h et garde son contenu jusqu'au prochain évènement onStarting.

CAS 5 => Evènement pris en charge

Colonne 1-2 : L'évènement est déclenché le 10 janv à 10h15.

Colonne 3 : Le Batch layer travaille sur la plage horaire de l'origine jusqu'à 10h.

Colonne 4 : Le Batch layer est en cours de traitement sur cette heure (9h à 10h02) donc n'envoie pas d'évènement au Speed layer.

Colonne 5 : C'est le Current qui va traiter cet évènement car il travaille de 10h à maintenant.

Colonne 6 : Le previous a travaillé de 9h à 10h.

Cas 6 => Evènement pris en charge

Colonne 1-2 : L'évènement est déclenché le 10 janv à 10h59.

Colonne 3 : Le Batch layer travaille sur la plage horaire de l'origine jusqu'à 10h.

Colonne 4 : Le Batch layer est en cours de traitement sur cette heure (9h à 10h02) donc n'envoie pas d'évènement au Speed layer.

Colonne 5 : C'est le current qui va traiter cet évènement car il travaille de 10h à maintenant.

Colonne 6 : Le previous a travaillé de 9h à 10h.

Cheminement de la requête «Combien de médias ont été consultés sur les 30 derniers jours» :

Spécification cas pratiques liées à une requête utilisateur : Combien de médias ont été consultés les 30 derniers jours									
Jour J	Time	Batch Layer = BL	Event from BL	Speed Layer Current = SPC	Speed Layer Previous = SPP	Query period	query period	query result	
10-01-2020	9h59	[origine BL .. 9h [[9h .. Time [[9h .. 9h [les 30 dernier jours	[J-29 0h00 ..aujourd'hui 9h59 [BL[J-29 0h00..J 9h00[+ SPC[9h..Time[
10-01-2020	10h00	[origine BL .. 9h [Starting Computing BL	[10h .. Time [[9h .. 10h [les 30 dernier jours	[J-29 0h00 ..aujourd'hui 10h00 [BL[J-29 0h00..J 9h00[+ SPP[9h..10h[+ SPC[10h..Time[
10-01-2020	10h01	[origine BL .. 9h [[10h .. Time [[9h .. 10h [les 30 dernier jours	[J-29 0h00 ..aujourd'hui 10h01 [BL[J-29 0h00..J 9h00[+ SPP[9h..10h[+ SPC[10h..Time[
10-01-2020	10h02	[origine BL .. 10h [BL Computed	[10h .. Time [[9h .. 10h [les 30 dernier jours	[J-29 0h00 ..aujourd'hui 10h02 [BL[J-29 0h00..J 10h00[+ SPC[10h..Time[
10-01-2020	10h15	[origine BL .. 10h [[10h .. Time [[9h .. 10h [les 30 dernier jours	[J-29 0h00 ..aujourd'hui 10h02 [BL[J-29 0h00..J 10h00[+ SPC[10h..Time[
10-01-2020	10h59	[origine BL .. 10h [[10h .. Time [[9h .. 10h [les 30 dernier jours	[J-29 0h00 ..aujourd'hui 10h02 [BL[J-29 0h00..J 10h00[+ SPC[10h..Time[

On peut catégoriser les colonnes :

- Temps qui passe
- État du Batch layer et du Speed layer à l'instant T
- Évènement du Batch layer envoyé au Speed layer
- Requête actions

CAS 1 => Cheminement requête à 9h59

Colonne 1 : Une requête est lancée sur les 30 derniers jours (l'utilisateur d'un hub ouvre le panneau de statistiques).

Colonne 2 : La période concernée est de J-29 0h00 à aujourd'hui 9h59.

Colonne 3 : On récupère les infos du Batch layer de j-29 0h00 à 9h00 et les informations récoltées du current de 9h à maintenant.

CAS 2 => Cheminement requête à 10h00

Colonne 1 : Une requête est lancée sur les 30 derniers jours.

Colonne 2 : La période concernée est de J-29 0h00 à aujourd'hui 10h00.

Colonne 3 : On récupère les infos du B. de j-29 0h00 à 9h00, les infos récoltées du previous de 9h à 10h et les infos du current de 10h à maintenant.

sachant qu'à ce moment là le B. envoie l'évènement onStarting au Speed layer ce qui permet de spécifier à celui-ci que la période de 9h à 10h n'est pas prise en charge par le Batch layer.

CAS 3 => Cheminement requête à 10h01

Colonne 1 : Une requête est lancée sur les 30 derniers jours.

Colonne 2 : La période concernée est de J-29 0h00 à aujourd'hui 10h01.

Colonne 3 : On récupère les informations du Batch layer de j-29 0h00 à 9h00, les informations récoltées du previous de 9h à 10h et les informations du current de 10h à maintenant.

CAS 4 => Cheminement requête à 10h02

Colonne 1 : Une requête est lancée sur les 30 derniers jours.

Colonne 2 : La période concernée est de J-29 0h00 à aujourd'hui 10h02.

Colonne 3 : On récupère les infos du Batch layer de j-29 0h00 à 10h00 et les informations du current de 10h à maintenant.

A ce moment là le Batch layer envoie l'aujourd'hui Batch layer onDone au Speed layer ce qui permet de ne plus se baser sur le previous puisque le Batch layer a pris le relais.

CAS 5 => Cheminement requête à 10h15

Colonne 1 : Une requête est lancée sur les 30 derniers jours.

Colonne 2 : La période concernée est de J-29 0h00 à aujourd'hui 10h15.

Colonne 3 : On récupère les informations du Batch layer de j-29 0h00 à 10h00 et les informations du current de 10h à maintenant.

CAS 6 => Cheminement requête à 10h59

Colonne 1 : Une requête est lancée sur les 30 derniers jours.

Colonne 2 : La période concernée est de J-29 0h00 à aujourd'hui 10h59.

Colonne 3 : On récupère les informations du Batch layer de j-29 0h00 à 10h00 et les informations du current de 10h à maintenant.

[Retour aux diagrammes de séquences](#)

Résumé : création et intégration du Speed layer, sous-composant de l'architecture Lambda.

Marie kersalé

Beebuzziness, entreprise éditrice de logiciels, situé à Grenoble, propose sa propre solution, Ustream, une plateforme en ligne composée de différents modules permettant d'aider les marques et les propriétaires de contenus digitaux à optimiser leur organisation et la diffusion sur le web.

Un hub est un module central d'Ustream, espace de stockage de documents virtuels autour duquel s'articule des options de consultation, de partage et d'analyse.

Ma mission était basée sur la fonctionnalité d'analyse statistique de l'audience proposée aux clients via un hub. Ces statistiques permettent aux clients d'avoir une vue globale sur le trafic de leur médias au sein d'un hub tel que les médias les plus consultés, comment sont-ils partagés sur les réseaux sociaux.

Au cours de cette mission j'ai intégré la pièce manquante de cet outil reposant sur l'architecture Lambda, le Speed layer amène plus de précisions sur les statistiques en prenant en compte les événements en temps réel. La mission consistait donc à concevoir l'écriture du code du Speed layer et intégrer ce sous-composant dans l'architecture Lambda.

Les objectifs de la mission ont été atteints, le Speed layer est maintenant disponible sur nos dockers locaux et sur mon hub en production et prochainement sur tous les hubs après une série de tests applicatifs et l'accord du directeur du développement.

Mots clés : Ustream, hub, architecture Lambda, Speed layer

Abstract : Creation and integration of the Speed layer, subcomponent of the Lambda architecture.

Marie kersalé

Beebuzziness, a software publisher located in Grenoble, offers its own solution, Ustream, an online platform made up of different modules that help brands and owners of digital content to optimize their organization and distribution on the web.

A hub is a central module of abstream, a space for storing virtual documents around options for consultation, sharing and analysis.

My mission was based on the analysis functionality offered to customers via a hub. These statistics allow customers to have a global view on the traffic of their media within a hub such as the most consulted media, how are they shared on social networks.

During this mission I integrated the missing part of this tool based on the Lambda, the Speed layer brings more details on the statistics by taking into account the events in real time. The mission therefore consisted of designing the writing of the Speed layer code and integrating this subcomponent into the Lambda architecture.

The objectives of the mission have been reached, the Speed layer is now available on our local dockers and on my hub in production and soon on all the hubs after a series of application tests and the agreement of the development director.

Key words : Ustream, hub, Lambda architecture, Speed layer