
**Proposition de projet**

# Mise en place d'un data werhouse pour l'entreprise Olist sur AWS

# Chapitre 1 : Le naratif du projet

Olist est une entreprise brésilienne spécialisée dans le e-commerce, dont l'ambition fondatrice était de devenir la plus grande marketplace du pays. C'est un facilitateur pour les petites et moyennes entreprises, qui permet aux commerçants de vendre sur plusieurs marketplaces et canaux de vente en ligne. Elle se positionne comme une plateforme intermédiaire qui facilite l'intégration des vendeurs indépendants dans un écosystème numérique vaste, techniquement exigeant, et souvent inaccessible pour des acteurs isolés. En pratique, Olist fournit au vendeur une interface unique pour gérer l'ensemble du cycle de vie de la vente : mise en ligne des produits, centralisation des commandes, suivi logistique, gestion du service client et analyse des performances commerciales.

Le dataset Olist, rendu public via la plateforme Kaggle contient environ 100 000 commandes enregistrées entre 2016 et 2018, provenant de différents marketplaces brésiliens (une plateforme en ligne qui met en relation des vendeurs tiers et des acheteurs, permettant la vente de produits ou services sans que la plateforme ne soit elle-même le fournisseur.)

Ce sont des données réelles issues de transactions commerciales, mais elles ont été anonymisées pour respecter la confidentialité : certaines identités ou références ont été remplacées, comme dans les avis clients où des noms ont été substitués par des maisons de Game of Thrones.

Le dataset est composé de 9 tables :

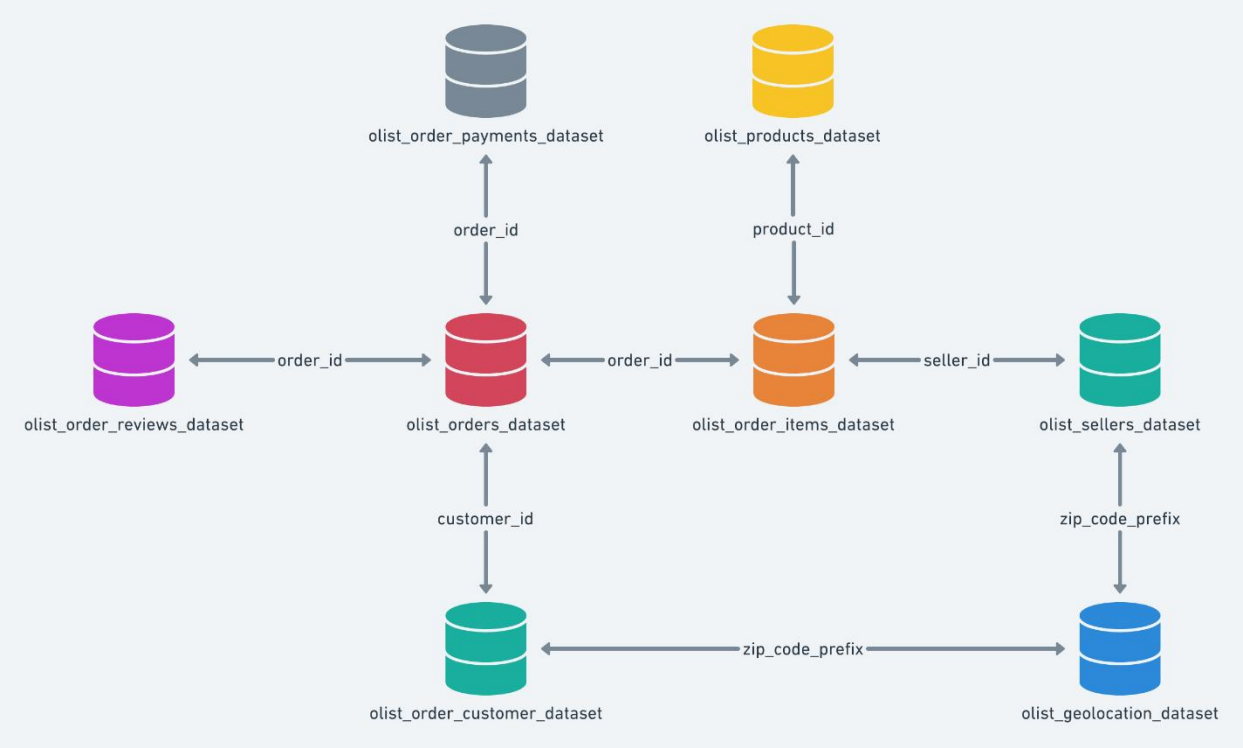


*Tableau 1:Description des tables du dataset Olist*

| Nom de la table | Déscription |
|---|---|
| **Orders** | Contient les informations de commande (status, timestamps comme la date d'achat, d'approbation, de livraison estimée ou effective, etc.) |
| **Order_items** | Pour chaque commande, cette table détaille les articles (produits) associés, leur prix, le fret, le poids, les dimensions, etc |
| **Order_payments** | Décrit les méthodes de paiement, le nombre de versements le montant payé, etc. |

| | |
|---|---|
| **Order_reviews** | Regroupe les retours clients — une fois la commande livrée, les clients peuvent noter leur expérience et laisser un commentaire. |
| **Products** | Décrit les produits : catégories, poids, dimensions, etc. |
| **Product_category_*name_translation*** | Un fichier de traduction des noms de catégories, permettant de passer des catégories en portugais à des intitulés en anglais, utile pour l'analyse. |
| **Sellers** | Concerne les vendeurs (sellers) qui ont expédié les articles : localisations, identifiants, etc. |
| **Customers** | |
| **Geo_locations** | Un fichier de géolocalisation qui associe des codes postaux brésiliens à des coordonnées latitude / longitude, ce qui permet d'analyser la dimension spatiale (géographique) des commandes ou des clients. |

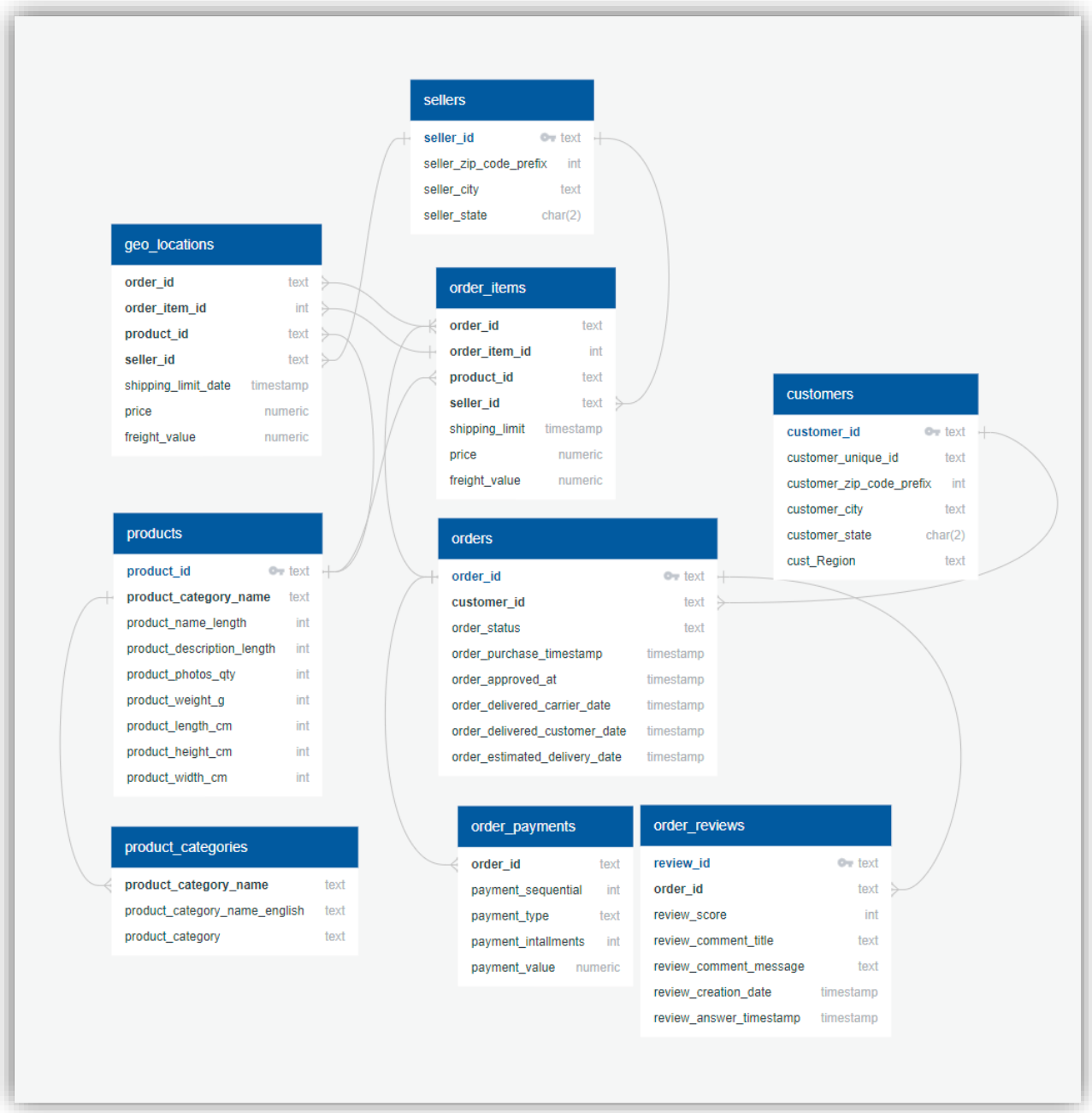


*Figure 1: schema du dataset Olist*

## 1. L' ère du monolithe : le succès et ses sombres coûts

Dans ses premières années, Olist a connu une croissance fulgurante, portée par une architecture informatique simple et efficace pour l'époque.

Le cœur battant de l'entreprise était un unique et gigantesque serveur de base de données PostgreSQL, surnommé en interne " Turing-DB-V5". Ce monolithe, tournant sur une infrastructure "on-premise[1]" dans un data center de São Paulo, était à la fois le cœur et le cerveau de l'entreprise.

Ce système était une pure architecture OLTP (Online Transaction Processing). Il était conçu pour l'efficacité transactionnelle : enregistrer une nouvelle commande, mettre à jour un stock, traiter un paiement. Chaque clic sur le site olist.com, chaque ajout au panier, chaque mise à jour de stock par un vendeur provoquait une lecture ou une écriture sur cette base de données centrale. Pour la croissance initiale, ce fut une bénédiction. Mais à mesure que les volumes explosaient, cette bénédiction s'est transformée en goulot d'étranglement, puis en véritable boulet stratégique.

## 2. Les Limites Stratégiques

Chez Olist, il n'y avait pas de "Data Team" ou de "Pôle BI". Il y avait Marcélo, un analyste, brillant et surchargé, qui portait sur ses épaules l'entièreté du reporting de l'entreprise. Sa journée type illustre parfaitement les faiblesses critiques qui rongeaient Olist de l'intérieur.

### 2.1 : Scénario 1 : La simple demande du PDG :

À 9h00, un email du PDG tombe : Marcélo, j'ai besoin des chiffres de ventes d'hier, ventilés par catégorie de produit et par ville, pour la réunion de 10h00."

L'estomac de Marcelo se noue. Il sait que la demande, en apparence simple, est impossible à satisfaire. Il ne peut absolument pas lancer sa requête sur la base de production Turing-DB-V5. Une requête d'analyse complexe (avec des SUM(), GROUP BY, et de multiples JOIN sur des tables de millions de lignes) pourrait verrouiller les tables. Un tel verrouillage arrêterait net le site web, empêchant les clients de finaliser leurs paiements, ce qui risque de couter beaucoup de bénéfice à l’entreprise.

[1] les serveurs, le matériel, les logiciels et les ressources nécessaires pour stocker et gérer les données sont installés et maintenus physiquement dans les locaux de l’entreprise, plutôt que dans le cloud.

Marcelo doit donc se connecter à " Turing-DB-V5-REPORT-01", une réplique de la base de données. Cette réplique n'est synchronisée que la nuit, lors d'un batch[2] à 3h00 du matin, pour ne pas surcharger le serveur principal. Au mieux, les données qu'il analyse ont déjà 12 heures de retard. Le concept de "temps réel" est de la science-fiction.

Il ouvre son script SQL, "rapport_ventes_v12_final_FINAL.sql". C'est un monstre de 800 lignes qu'il a mis des mois à perfectionner, un enchevêtrement de 15 jointures pour lier les commandes, les clients, les produits, les adresses, les paiements et les catégories. Il lance la requête. Et il attend.

Le script tourne... pendant deux heures[3]. L'architecture OLTP n'est pas faite pour ce type de lecture. Pendant ce temps, Marcelo ne peut rien faire d'autre.

À 11h15, la requête termine enfin. Elle exporte un fichier ventes_hier.csv de 700 Mo. Marcelo essaie de l'ouvrir dans Excel. Excel plante. Il réessaye, en important les données. Il parvient enfin à créer un tableau croisé dynamique. Il fait une capture d'écran et la colle dans son email de réponse au PDG, bien après la fin de la réunion. La décision stratégique a été prise "à l'instinct", sans données pour la soutenir.

## 2.2 : Scénario 2 : La réunion stratégique mensuelle :

La salle est pleine. Le PDG lance la réunion mensuelle en projetant la capture d'écran fournie par Marcélo. À l'écran, le tableau croisé montre un chiffre : 30,2 millions. Le Directeur Administratif et Financier lève la main, visiblement irrité, et explique que ses rapprochements bancaires donnent 29,8 millions. La directrice Marketing, qui suit ses propres rapports d'attribution, clame 31,1 millions. La discussion tourne immédiatement au procès d'intention : qui a tort, qui a triché, qui a

[2] Un "batch" (ou traitement par lots) désigne un mode d'exécution automatique de tâches informatiques regroupées, généralement planifiées à des moments précis (souvent la nuit), afin de traiter de grandes quantités de données sans intervention humaine et sans perturber les opérations en temps réel.

[3] Une requête analytique de grande ampleur peut s'étendre sur plusieurs heures lorsque l'architecture sous-jacente n'est pas conçue pour ce type de charge. Dans le cas d'une base OLTP, les tables sont fortement normalisées, les index sont optimisés pour des lectures ponctuelles plutôt que pour des agrégations massives, et les jointures profondes sur des millions de lignes forcent souvent PostgreSQL à effectuer de longues opérations de tri, de hachage ou de fusion. Dès que ces traitements dépassent la mémoire de travail allouée, ils basculent dans des fichiers temporaires sur disque, ce qui multiplie les accès I/O et fait exploser les temps de calcul. Le problème se renforce si les statistiques internes sont obsolètes, car le planificateur choisit alors un plan inefficace fondé sur de mauvaises estimations de cardinalité. Les verrous, les transactions longues, la contention sur la réplique de reporting, la saturation des canaux d'I/O et la faiblesse du parallélisme dans certaines opérations s'additionnent et allongent l'exécution. Dans ces conditions, un script complexe peut aisément dépasser une ou deux heures de traitement, même sans erreur apparente, simplement parce que l'environnement n'est pas dimensionné pour de l'analyse intensive.

mal exporté ses CSV. En réalité, personne n'a « tort » au sens strict. Chacun parle de choses différentes à partir de données différentes, extraites à des moments différents et traitées selon des règles différentes.

Pourquoi ces chiffres ne coïncident pas ?

D'abord, la définition de la « vente » n'est pas uniforme. Pour la finance, une vente signifie une commande payée et expédiée, nette des retours et des remboursements. Pour le marketing, la « vente » c'est la valeur brute d'une commande au moment où l'utilisateur clique et valide son panier ,sans que l'on sache si le paiement aboutira ou si la commande sera retournée. Pour Marcélo, son script prend les données de la réplique de reporting qui a été synchronisée à 03:00 du matin, et il compile toutes les commandes créées la veille, indépendamment du statut final. Ces écarts de définition produisent automatiquement des totaux différents.

Ensuite, la fraîcheur des données compte. Quand la réplique de reporting est actualisée une fois par nuit, les chiffres analysés ont un décalage temporel. Des paiements validés après la synchronisation, des annulations tardives ou des retours traités le matin même n'apparaîtront pas dans le rapport de Marcélo mais seront visibles dans les rapprochements bancaires du Directeur Administratif et Financier. Le problème est aggravé si des ETL (procédures d'extraction/transformation/chargement) ne sont pas atomiques ou si des jobs[4] échouent ponctuellement : un lot manquant suffit à faire varier un million.

La méthode de calcul et le périmètre technique sont tout aussi critiques. Les jointures multiples, les doublons non filtrés, la conversion de devises, et la prise en compte ou non des frais de plateforme modifient le montant final. Un export CSV peut contenir des lignes intermédiaires (par exemple une ligne par article) alors que la comptabilité veut des lignes par commande agrégée ; sans règles claires de transformation, les sommes divergent.

Enfin, il y a des contraintes d'outils et d'infrastructure qui biaisent la circulation de l'information. Un export massif (700 Mo) n'est pas exploitable directement dans des outils bureautiques comme

[4] Un job désigne une tâche automatisée programmée pour exécuter une séquence d'opérations (extraction, transformation, chargement, calculs ou agrégations). Chaque job est généralement planifié à intervalles réguliers (par exemple, la nuit) et son exécution doit être complète et atomique pour garantir la cohérence des données. L'échec ou l'absence d'un seul job peut entraîner des écarts significatifs dans les chiffres présentés, car certaines transactions ou mises à jour ne sont alors pas intégrées dans le lot de données attendu.

Excel ou Google Sheets : Excel plafonne à un peu plus d'un million de lignes et ralenti beaucoup avant d'atteindre cette limite, et l'ouverture d'un fichier si volumineux provoque des crashs ou des pertes de temps. Pour obtenir un chiffre utilisable rapidement, les équipes bricolent des extractions, tronquent des colonnes, ou s'appuient sur des échantillons — autant d'opérations qui introduisent des divergences.

Résultat : la réunion dégénère en querelle d'interprétation au lieu de débattre de la stratégie. Le PDG manque d'un indicateur consensuel, la finance continue son travail de réconciliation manuel, le marketing mesure l'efficacité de ses campagnes avec une métrique utile pour lui mais impropre à la comptabilité. Cette fragmentation crée un coût caché important : temps perdu, décisions retardées ou prises sans base fiable, et risque d'erreurs financières.

## 2.3 : Scénario 3 : Le point de rupture

Tout a basculé il y a trois mois, lorsque la direction a dévoilé sa nouvelle stratégie de croissance : un service baptisé « Olist Pro ». L'idée paraissait simple sur le papier, presque évidente pour une marketplace cherchant à fidéliser ses meilleurs clients : une livraison garantie en vingt-quatre heures, un support prioritaire, des avantages exclusifs, et une promesse claire de performance. Pour la direction, ce service représentait la prochaine étape naturelle de l'expansion d'Olist. Pour l'équipe technique, ce fut un cataclysme annoncé.

La réunion de lancement, tenue dans la grande salle vitrée du siège, a rapidement viré à l'interrogatoire général. Le PDG, enthousiaste mais pragmatique, voulait comprendre comment mesurer en temps réel la rentabilité du nouveau service, alors que chaque minute de retard logistique allait coûter cher. Le responsable des opérations, visiblement inquiet, rappelait que garantir une livraison en vingt-quatre heures exigeait une vision précise et immédiate de l'état de chaque commande. Il ne savait même pas, à l'instant présent, identifier lesquelles n'avaient pas encore quitté les entrepôts des vendeurs. Quant au manager e-commerce, il expliquait que ses campagnes de promotion « flash » perdaient leur efficacité si les données de vente mettaient douze heures à émerger : il lui fallait suivre, minute par minute, les comportements d'achat des clients « Pro » pour ajuster les prix, les stocks et les incitations.

Silencieux au fond de la salle, Marcélo observait la scène. Depuis des années, il fabriquait chaque nuit les rapports que tout Olist attendait au matin. Son univers était celui du « lendemain », pas

celui de « l'instant ». Mais face aux ambitions de « Olist Pro », son métier était devenu anachronique. Il ne pouvait rien produire à la minute, ni même à l'heure. L'écosystème technique d'Olist reposait encore sur Turing-DB-V5 et sur ses répliques différées, dont les extractions massives mettaient déjà des heures à produire un simple fichier CSV.

Dans cette salle de crise, chacun cherchait des réponses immédiates, des tableaux mis à jour automatiquement, des alertes déclenchées dès qu'un vendeur tardait à expédier une commande « Pro ». La direction réclamait un pilotage continu, une lecture en streaming de l'activité, une capacité à anticiper avant même que les problèmes n'apparaissent. Et pourtant, derrière cette façade d'innovation, la réalité technique était brutale : l'architecture héritée ne supportait même pas les requêtes analytiques nocturnes sans vaciller. Tenter de faire du temps réel sur ce système revenait à pousser un camion lancé à pleine vitesse dans un virage conçu pour une bicyclette.

Le constat s'imposait, implacable et douloureux : « Olist Pro » était incompatible avec la manière dont Olist fonctionnait depuis dix ans. Le cœur technologique, figé dans un monde de transactions synchrones et de reporting du lendemain, se retrouvait soudain face à un univers d'alertes instantanées, et de décisions prises seconde par seconde. L'ambition commerciale était entrée en collision directe avec les limites physiques et logicielles .

Marcelo le comprit avant tout le monde : le lancement d'« Olist Pro » n'était pas une simple évolution. C'était une rupture. Une frontière nette entre « avant » et « après ». Un moment où une organisation découvre, brutalement, que son propre système d'information ne peut plus soutenir ses ambitions.

Et dans cette salle silencieuse, après les questions, les inquiétudes et les promesses, une vérité flottait déjà dans l'air : si rien ne changeait, Olist ne serait pas capable de tenir sa nouvelle promesse. Et l'entreprise venait, sans le savoir, de franchir son point de rupture.

## 2. Les Avengers à la rescousse : Notre Mission

La salle est lourde d’attentes. Le PDG fixe l’écran où s’affichent les chiffres contradictoires ;. DataSmart, groupe d’anciens étudiants de l’ENSAE reconvertis en consultants, a été invité pour proposer une issue praticable. Ben, leur porte-parole, ouvre la discussion en posant le diagnostic de façon simple et clinique :

Le problème est que votre architecture actuelle vous force à choisir entre vendre et analyser. Chaque requête de Marcelo risque de bloquer un paiement. Et l'ambition "Olist Pro" est tout simplement incompatible avec un rapport de la veille.

« Vous avez besoin de voir ce qui se passe maintenant, pas ce matin à 3h00»

Le Directeur réagit aussitôt : « Mais vous n'aurez pas accès à notre base de production. C'est hors de question.

« Nous ne la demandons pas, » répond calmement Bén. Donnez-nous la réplique `Turing-DB-V5-REPORT-2016-2017, cela nous donnera l'historique. »

« Mais alors, comment gérer le temps réel ? », demande le responsable des opérations.

« Nous allons le simuler, » explique Bén. « Pour prouver que l'architecture fonctionne, nous n'avons pas besoin de vos données en prod »

« Nous allons construire un Jumeau Numérique. C'est une API amimenté par un modèle de data augmentation qui va générer des événements (de nouvelles commandes, des avis, des paiements,…) pour simuler l'activité de votre site, et en particulier les flux de "Olist Pro". Nous allons augmenter les données de votre réplique avec ce flux contrôlé. »

Bén projette un schéma simple.

Voici la stratégie :

1. Le Fleuve de données[5] :

« Chaque nouvel événement de notre simulateur  n'est pas écrit dans une base de données qui verrouille. Il est publié dans un fleuve de données en temps réel. L'information circule instantanément, et plusieurs services peuvent la lire en même temps, sans se gêner. »

2. La Double Destination

Ce fleuve alimente deux destinations en parallèle :

- Pour le besoin immédiat : « Nous branchons des tableaux de bord Power BI directement sur le fleuve. Le responsable des opérations verra les commandes "Pro" s'afficher à la seconde près. C'est ce dont vous avez besoin pour piloter votre promesse des 24h.
- En même temps, un autre service prend ce même flux, le convertit, le compresse et l'archive automatiquement dans un Data Lake[6]. C'est votre nouveau coffre-fort de données brutes, fiable et peu coûteux. »

3. La Source unique de vérité

Marcelo lève la main : « Et mes rapports ? Je dois toujours faire mes jointures de 800 lignes ?»

« Non, » dit Bén. « Car ce lac de données alimente votre nouvel entrepôt de données (Redshift Serverless). C'est un moteur d'analyse conçu pour des requêtes complexes. Il est bâti pour la vitesse. »

Le résultat :

[5] Le terme fleuve de données désigne une architecture de diffusion continue d'événements ou de messages, souvent implémentée via des systèmes de type event streaming comme Apache Kafka, Amazon Kinesis ou Azure Event Hubs. Contrairement aux bases de données transactionnelles, ces flux permettent une lecture simultanée, non bloquante, et scalable par plusieurs consommateurs, favorisant la découplage des services et la réactivité en temps réel dans les systèmes distribués.

[6] Le Data Lake désigne ici un espace de stockage cloud (Amazon S3) où les données brutes sont archivées sous forme de fichiers Parquet — un format colonne optimisé pour la compression et la performance analytique. Ces fichiers sont partitionnés (par date, type, etc.) pour accélérer les requêtes, et catalogués automatiquement via AWS Glue, ce qui permet aux outils comme Athena ou SageMaker de les interroger facilement. Ce mécanisme assure une conservation fiable, scalable et peu coûteuse des flux de données, tout en préparant le terrain pour des analyses stratégiques à long terme.

La production est intacte : « Marcelo, vous pouvez lancer les requêtes les plus lourdes sur Redshift. La base de production `Turing-DB-V5` ne le sentira même pas. Le site ne ralentira jamais. »

Les Chiffres sont Réconciliés : « Fini la querelle des chiffres en réunion. Le Marketing et la Finance n'auront plus des totaux différents. Les règles de calcul (ce qu'est une "vente", nette ou brute sont définies une seule fois avant d'arriver dans l'entrepôt. Ce sera votre source unique de vérité. »

L'analyse est rapide : « Le rapport qui prenait deux heures prendra quelques secondes. »

« Et le coût ? »

La directrice financière prend la parole : « Tout cela semble très cher. Des serveurs qui tournent en permanence... »

« C'est le contraire, » la rassure Bén.

« L'architecture que nous proposons est Serverless[7]. Pensez-y comme à l'électricité : vous ne payez pas pour la centrale, vous ne payez que les millisecondes où vous allumez la lumière. Notre simulateur (Lambda) ne coûte rien s'il n'est pas appelé. L'entrepôt (Redshift Serverless[8]) s'éteint automatiquement s'il est inactif. »

« De plus, » ajoute-t-il, « la gestion des coûts  n'est pas une option, c'est au cœur du projet. Dès le premier jour, nous mettons en place des alertes budgétaires. Si nous approchons 80% du budget alloué , vous êtes la première informée. Vous gardez un contrôle absolu et en temps réel sur les dépenses. »

Bén conclut : « Notre mission est de vous prouver que cette solution fonctionne. Donnez-nous la réplique `Turing-DB-V5-REPORT-2016-2017.

[7] Serveless :Modèle d'architecture cloud dans lequel les ressources de calcul sont automatiquement allouées et facturées à l'usage, sans gestion explicite de serveurs par les développeurs. Les services comme AWS Lambda ou Redshift Serverless s'activent uniquement en réponse à des événements ou requêtes, puis s'arrêtent automatiquement, permettant une scalabilité élastique et une optimisation fine des coûts. Ce paradigme est particulièrement adapté aux charges intermittentes, aux PoC, et aux pipelines événementiels.

[8] Redshift Serverless :Variante serverless du data warehouse Amazon Redshift, permettant d'exécuter des requêtes SQL analytiques sans provisionner ni gérer de clusters. Le moteur s'active à la demande, s'adapte automatiquement à la charge, et s'éteint en cas d'inactivité. Ce modèle offre une facturation à la seconde, une scalabilité élastique, et une intégration native avec les services AWS tels que S3, Lambda, QuickSight ou SageMaker. Idéal pour les PoC, les workloads intermittents, ou les pipelines analytiques modulaires.

# Chapitre 2 : Présentation technique du projet