Chapitre 1 — Introduction aux architectures de données

Introduction

Une organisation ne peut pas tirer parti de ses données si celles-ci sont dispersées, mal définies ou impossibles à retrouver. L'architecture de données fournit le cadre qui permet de faire circuler la donnée depuis sa source jusqu'à son exploitation métier, en garantissant autant que possible sa fiabilité, sa sécurité et sa disponibilité.

Ce chapitre présente :

- la définition et les composants d'une architecture de données ;
- son importance stratégique dans un contexte de volumes, de vitesse et d'usages croissants de l'intelligence artificielle ;
- six grandes familles d'architectures modernes ;
- les enjeux à arbitrer lors de la conception ;
- les tendances qui transforment actuellement les plateformes de données.

Objectifs pédagogiques

À l'issue du chapitre, il faut être capable de :

- comprendre ce qu'est une architecture de données et à quoi elle sert ;
- expliquer son importance face à l'augmentation des volumes, de la variété et de la vitesse des données ;
- distinguer les architectures centrées sur les données, événementielles, orientées services, Data Mesh, Lambda/Kappa et Lakehouse ;
- identifier les enjeux de scalabilité, de sécurité, de qualité, de coût, d'interopérabilité et de gouvernance ;
- repérer les tendances du cloud-native, du streaming, de l'IA, du DataOps et du self-service.

1.1 Concepts de base

Définition d'une architecture de données

Définition :

> une architecture de données est l'ensemble des règles, modèles et structures qui déterminent comment les données sont **collectées, stockées, organisées, transformées et consommées** au sein d'une organisation.

On peut la comparer au plan d'urbanisme d'une ville :

- les bâtiments représentent les applications ;
- les routes représentent les flux de données ;
- le plan évite que chaque application soit construite isolément et que les flux ne se rejoignent jamais.

L'architecture pose donc les fondations avant la construction des applications et des usages analytiques.

Les composants d’une architecture de données

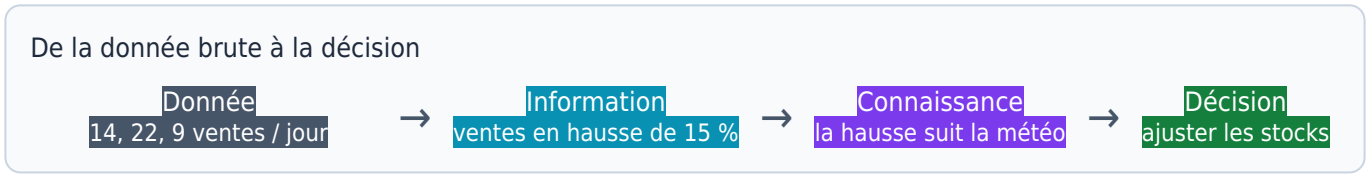
Une chaîne de données typique se lit de gauche à droite :



Chaque brique répond à une question essentielle : d’où vient la donnée ? Où est-elle rangée ? Comment est-elle transformée ? Qui l’utilise et pour quel objectif ?

De la donnée brute à la décision

Le parcours décisionnel peut être résumé ainsi :



Une architecture de données décisionnelle vise à **fiabiliser et accélérer le passage de la donnée brute à la décision métier**. Elle ne se limite donc pas à stocker des informations : elle organise leur transformation en éléments compréhensibles et actionnables.

Exemple : une enseigne de distribution

Une enseigne possédant 800 magasins, un site de commerce électronique et une application mobile produit des données de vente dans des formats différents. Sans architecture commune, le marketing, la logistique et la finance reconstruisent chacun leurs chiffres ; les indicateurs peuvent alors se contredire.

Une architecture bien conçue centralise les flux dans un entrepôt de données partagé et définit précisément ce qu’est « **une vente** ». Tous les services travaillent ainsi sur une même référence et sur des chiffres comparables.

Architecture de données et architecture applicative

Ces deux notions sont complémentaires, mais ne répondent pas à la même question :

Architecture	Préoccupation principale	Exemples
Architecture applicative	Organisation des logiciels et services ; manière dont les systèmes communiquent	Microservices, API Gateway
Architecture de données	Localisation, forme, circulation et fiabilité des données produites et consommées	Data Warehouse, Data Lake, modèle en étoile

Une architecture applicative efficace ne suffit pas à garantir des données cohérentes. Des applications bien

organisées peuvent tout de même produire des définitions divergentes, des doublons ou des données impossibles à consolider.

1.2 Pourquoi l’architecture de données est-elle devenue critique ?

Les cinq facteurs de transformation

L’importance de l’architecture s’explique par cinq évolutions :

- **Volume :** une organisation peut produire en une journée davantage de données qu’elle n’en produisait en une année il y a dix ans ;
- **Variété :** textes, images, vidéos, mesures IoT et logs s’ajoutent aux données structurées classiques ;
- **Vitesse :** certaines décisions — par exemple la fraude ou la recommandation — doivent être prises en temps réel ;
- **Valeur :** la donnée est devenue un actif stratégique, parfois plus valorisé que certains actifs physiques ;
- **IA :** un modèle d’intelligence artificielle n’a de valeur que si les données qui l’alimentent sont fiables, accessibles et correctement gouvernées.

La donnée comme actif stratégique

Sans architecture solide, on observe généralement : données dispersées, doublons, définitions concurrentes d’un même indicateur, reporting lent et décisions guidées par l’intuition. Le « chiffre d’affaires » peut, par exemple, être calculé de trois manières différentes par trois équipes.

Avec une architecture maîtrisée, l’organisation obtient des données traçables, une vérité partagée, des tableaux de bord presque en temps réel et des décisions fondées sur les faits. Elle dispose aussi d’une base plus fiable pour entraîner des modèles d’IA.

Exemple : le moteur de recommandation de Netflix

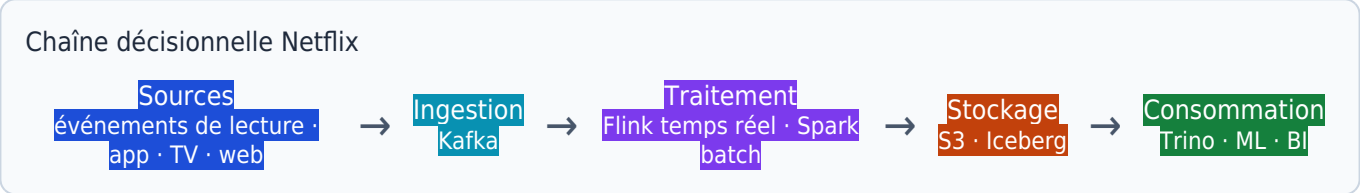
Netflix collecte en continu les événements de visionnage de plus de 260 millions d’abonnés : lecture, pause, arrêt et évaluation. Les événements sont ingérés en streaming, stockés dans un Data Lake, puis utilisés par des modèles de recommandation. Environ 80 % des contenus regardés proviennent des recommandations personnalisées.

Cet exemple montre qu’un service de recommandation dépend directement d’une architecture capable d’absorber un volume considérable et de traiter les événements rapidement.

Exemple d’une chaîne décisionnelle à grande échelle : Netflix

Outil	Type	Rôle
Apache Kafka	Bus d’événements	Ingestion d’environ 2 000 milliards de messages par jour : vues, clics, pauses
Apache Flink	Traitement temps réel	Calculs de personnalisation quasi instantanés ; plus de 20 000 jobs actifs
Apache Spark	Traitement batch	Entraînement des modèles et analyses de fond
Amazon S3 et Apache Iceberg	Stockage et format de tables	Data Lake analytique et historisation
Trino, anciennement Presto	Moteur de requêtage	Requêtes SQL sur le Data Lake à grande échelle

Le schéma fonctionnel est :



Cette chaîne fonctionne en continu, depuis l'événement de lecture jusqu'à la recommandation présentée à l'utilisateur.

Exemple : détection de fraude bancaire

Une banque doit analyser chaque transaction par carte en quelques millisecondes. Le calcul croise notamment l'historique du client, sa localisation et le montant de l'opération. L'architecture doit donc combiner :

- un traitement temps réel pour décider de bloquer ou non la transaction ;
- une analyse historique plus profonde pour entraîner et améliorer le modèle de détection.

Les catégories d'outils couramment rencontrées sont les suivantes — les stacks exactes des banques restant généralement confidentielles :

Outil ou catégorie	Type	Rôle
Kafka	Bus d'événements	Diffusion de chaque transaction en temps réel
Flink ou Spark Streaming	Traitement temps réel	Scoring de la transaction en quelques millisecondes
Feature store	Stockage de variables ML	Conservation de l'historique client précalculé pour le scoring
Moteur de règles	Traitement métier	Blocage automatique lorsqu'un seuil de risque est dépassé

Le coût d'une mauvaise architecture

Une architecture insuffisante entraîne trois conséquences majeures :

- **Décisions erronées :** un reporting fondé sur des données incohérentes provoque de mauvais choix stratégiques ;
- **Temps perdu :** les équipes peuvent consacrer 60 à 80 % de leur temps au nettoyage et à la réconciliation plutôt qu'à l'analyse ;
- **Risque réglementaire :** une mauvaise conception expose l'organisation sur les sujets de RGPD, de sécurité et d'auditabilité.



À retenir : l'architecture de données n'est plus un sujet réservé à l'IT. Elle constitue un levier de compétitivité qui conditionne la vitesse et la qualité des décisions et la capacité à exploiter l'IA.

1.3 Les grandes familles d'architectures

Panorama

Six familles sont particulièrement importantes :

- architecture centrée sur les données (*data-centric*) ;
- architecture événementielle (*event-driven*) ;
- architecture orientée services ou microservices ;
- Data Mesh ;
- architectures Lambda et Kappa ;
- Data Lakehouse.

Ces familles ne s'excluent pas. Une entreprise combine souvent plusieurs approches selon ses applications, ses volumes, ses exigences de temps réel et ses contraintes de gouvernance.

Architecture centrée sur les données

Dans une architecture **data-centric**, la donnée est placée au centre et reste indépendante des applications qui l'utilisent. Les applications lisent et écrivent dans un référentiel partagé au lieu de conserver chacune sa propre copie.

Chaîne type :

- sources applicatives : ERP, CRM, e-commerce ;
- intégration : ETL ou ELT ;
- référentiel central : Data Warehouse ;
- restitution : BI, rapports et API.

Exemple : groupe hôtelier multi-marques. Les applications de réservation, les hôtels et le programme de fidélité restent distincts, mais les données clients sont centralisées. Réservations, historique et préférences forment une vision unifiée ; le client est reconnu de façon cohérente quel que soit le canal ou la marque.

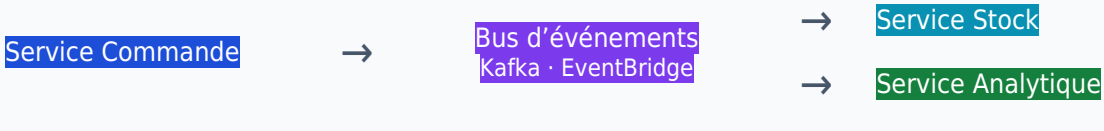
Outil ou catégorie	Type	Rôle
Fivetran ou Airbyte	Ingestion ELT	Connecteurs vers les applications sources
Snowflake	Entrepôt cloud	Référentiel partagé
dbt	Transformation	Construction des tables métiers à partir des données brutes
Tableau ou Power BI	BI	Tableaux de bord communs aux marques

Architecture événementielle

Dans une architecture **event-driven**, les composants communiquent par des événements publiés sur un bus de messages, plutôt que par des appels directs. Exemples d'événements : « commande créée » ou « paiement validé ». Chaque système réagit uniquement aux événements qui l'intéressent.

Schéma conceptuel :

Architecture événementielle : diffusion vers plusieurs consommateurs



Exemple : Uber. Chaque déplacement génère un flux de géolocalisation. Plusieurs consommateurs indépendants calculent le prix, estiment l'heure d'arrivée, détectent les anomalies et alimentent les tableaux de

bord. Un nouveau consommateur — par exemple un modèle de tarification — peut être ajouté sans modifier le reste du système.

Outil	Type	Rôle
Apache Kafka	Bus d'événements	Diffusion continue des positions GPS et événements de course
Apache Flink ou Spark	Traitement temps réel et batch	Prix dynamique et agrégations
Apache Hudi	Format transactionnel	Upserts incrémentaux du Data Lake
HDFS	Stockage distribué	Data Lake de plus de 100 pétaoctets
Presto/Trino et Apache Pinot	Requêtage	Requêtes interactives et analytique temps réel



Architecture orientée services et microservices

Chaque service métier — paiement, catalogue, client, etc. — possède sa propre base et expose ses fonctions par API. Les données sont distribuées par domaine plutôt que centralisées.

L'avantage principal est l'autonomie et la possibilité de faire évoluer la capacité de chaque équipe indépendamment. La difficulté est de reconstituer une vue globale : une couche d'agrégation, souvent un entrepôt de données, est alors nécessaire.

Exemple : Spotify. Des équipes autonomes, organisées en « squads », gèrent la lecture, les playlists, les recommandations ou la facturation avec leurs propres données. Le déploiement peut être réalisé des centaines de fois par jour sans bloquer les autres équipes. En contrepartie, une plateforme centrale doit recomposer une vision du comportement d'écoute.

Outil	Type	Rôle
Google Cloud Pub/Sub	Ingestion d'événements	Collecte des lectures, pauses et passages
Scio et Dataflow	Batch et streaming	Pipelines Scala basés sur Apache Beam
BigQuery	Entrepôt cloud	Stockage et requêtes à grande échelle
dbt	Transformation	Tables analytiques
Flyte	Orchestration	Pilotage de plus de 38 000 jobs
Looker	BI	Tableaux de bord produit et métier



Data Mesh

Le **Data Mesh** fait de chaque domaine métier le propriétaire de ses données. Le domaine publie ses données

comme un **produit de données** : le produit doit être documenté, fiable, gouverné et assorti d'engagements de service (SLA). Les autres équipes peuvent le découvrir et le consommer en self-service.

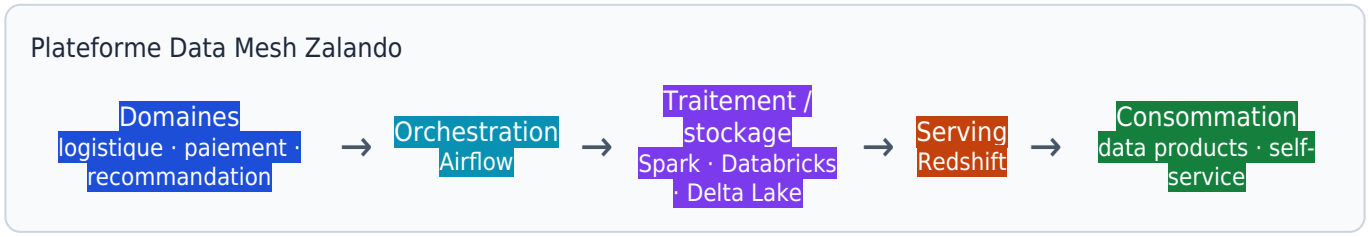
Exemples de produits :

- Ventes : « Ventes journalières » ;
- RH : « Effectifs » ;
- Marketing : « Campagnes ».

Cette approche s'oppose à une équipe data centrale qui serait l'unique passage obligé et deviendrait un goulot d'étranglement.

Exemple : Zalando. Face à des centaines d'équipes produit, les domaines logistique, paiement et recommandation publient leurs propres data products auto-documentés. Les équipes consomment les données sans attendre un ticket auprès d'une équipe centrale surchargée.

Outil ou catégorie	Type	Rôle
Apache Airflow	Orchestration	Planification des pipelines comme service central
Databricks et Delta Lake	Traitement et stockage	Cœur de la plateforme et socle du Data Mesh
Apache Spark	Traitement	Analytique et machine learning à grande échelle
Amazon Redshift	Serving layer	Requêtage rapide des data products
Catalogue de données self-service	Gouvernance	Découverte par environ 350 équipes décentralisées



Le support mentionne plus de 20 pétaoctets et plus de 5 000 data products pour cet exemple.

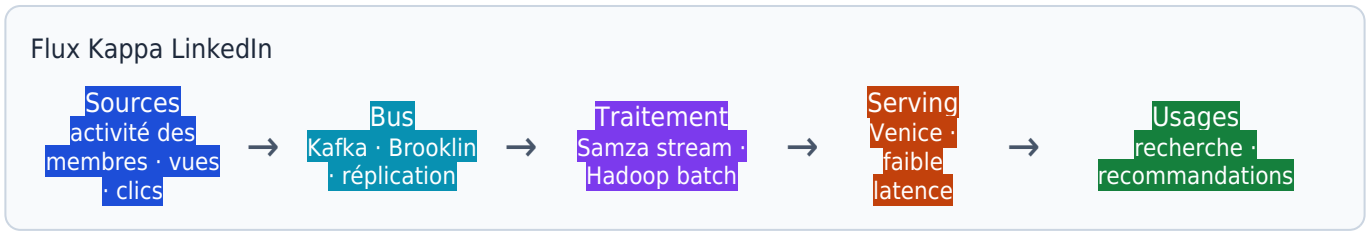
Architectures Lambda et Kappa

Ces architectures répondent au même besoin : combiner un traitement temps réel et un traitement historique fiable.

Architecture Lambda	Architecture Kappa
Deux circuits : un flux batch précis mais lent, et un flux temps réel rapide mais potentiellement approximatif	Un seul circuit de streaming, rejouable pour recalculer l'historique
Deux codes à maintenir ; solution plus complexe	Solution plus simple, mais dépendante d'une plateforme de streaming robuste comme Kafka

Exemple : LinkedIn. L'entreprise a créé Kafka pour standardiser ses événements — vues de profils, connexions et clics — et a adopté une logique Kappa. Un flux unique et rejouable alimente à la fois les tableaux de bord temps réel et les calculs historiques.

Outil	Type	Rôle
Apache Kafka	Bus central	Plus de 7 000 milliards de messages par jour
Apache Samza	Traitement de flux	Traitement temps réel sur Kafka, YARN et Hadoop
Brooklin	Réplication	Synchronisation de topics entre centres de données
Venice	Serving faible latence	Mise à disposition de données dérivées pour recherche et recommandation
Hadoop	Stockage batch	Historisation et calculs de fond

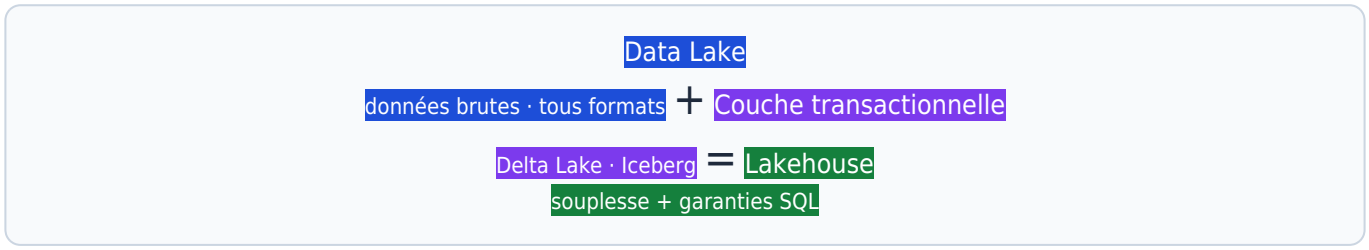


Data Lakehouse

Le **Data Lakehouse** est une architecture hybride. Il associe :

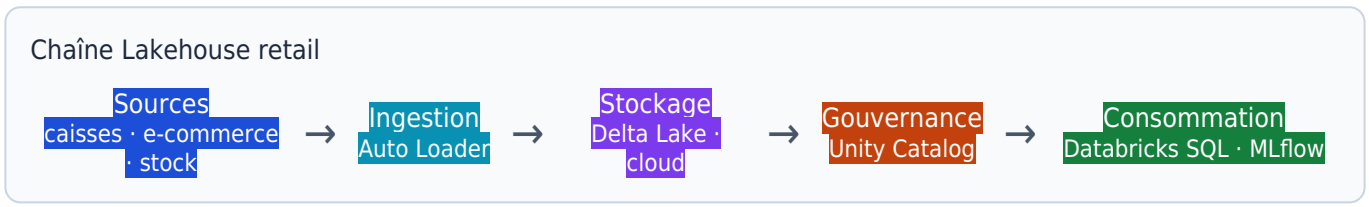
- la souplesse et le faible coût du Data Lake, qui accueille des données brutes de tous formats ;
- les garanties du Data Warehouse, notamment les transactions, la structure et les requêtes SQL rapides.

Une couche transactionnelle, telle que Delta Lake ou Iceberg, est ajoutée au stockage brut :



Exemple : H&M Group et Walgreens. Les tickets de caisse semi-structurés et les tables agrégées de reporting financier sont stockés sur une même plateforme. Il n’est pas nécessaire de dupliquer l’infrastructure entre un Lake et un Warehouse séparés : une seule copie peut servir plusieurs niveaux d’usage, ce qui réduit les coûts et simplifie la gouvernance.

Outil	Type	Rôle
Auto Loader	Ingestion	Chargement incrémental des flux de caisse, e-commerce et stock
Delta Lake	Format de tables	Fiabilité transactionnelle ACID sur le stockage cloud brut
Unity Catalog	Gouvernance	Catalogue et gestion des droits sur toutes les données
Databricks SQL	Requêtage et BI	Reporting financier et opérationnel
MLflow	Machine learning	Prévision de la demande et personnalisation



Comparaison des architectures

Architecture	Idée clé	Exemple
Centrée sur les données	Référentiel central partagé	Entrepôt de données de groupe
Événementielle	Communication par événements	Uber et le suivi des courses
Orientée services	Une base par service métier	Spotify et ses microservices
Data Mesh	Données publiées comme produits par domaine	Zalando
Lambda/Kappa	Combinaison du temps réel et de l’historique	LinkedIn et Kafka

Architecture	Idée clé	Exemple
Lakehouse	Réunion du Lake et du Warehouse	Databricks dans le retail

1.4 Les enjeux de conception

Les six enjeux majeurs

Toute architecture doit traiter simultanément :

- la **scalabilité** ;
- la **sécurité et la conformité**, notamment le RGPD ;
- la **qualité des données** ;
- l'**interopérabilité** ;
- la **maîtrise des coûts** ;
- la **gouvernance**.

Scalabilité

Définition : la scalabilité est la capacité à absorber une croissance des volumes et de la charge sans dégrader les performances ni faire exploser les coûts.

Lors du Black Friday, le trafic et les commandes d'un commerçant en ligne peuvent être multipliés par dix en quelques heures. Une architecture cloud élastique peut absorber ce pic ; une infrastructure rigide risque de ralentir fortement ou de tomber.

Sécurité et conformité réglementaire

Il faut protéger les données personnelles, financières et de santé contre les accès non autorisés tout en respectant les cadres légaux tels que le RGPD.

Exemple : Doctolib. Les données de santé sont particulièrement sensibles. L'architecture doit intégrer dès la conception le chiffrement, un hébergement certifié HDS et la traçabilité des accès.

Outil ou catégorie	Type	Rôle
Hébergeur certifié HDS	Infrastructure	Stockage conforme aux exigences relatives aux données de santé
Chiffrement au repos et en transit	Sécurité	Protection à chaque étape
IAM, gestion des identités et accès	Gouvernance	Contrôle de qui peut consulter quelle donnée
Plateforme data gouvernée	Stockage et analytique	Mise à disposition encadrée pour l'analytique et l'IA

La stack détaillée de Doctolib n'étant pas publique, ces catégories sont représentatives des architectures de santé sécurisées.

Qualité des données

Des données incomplètes, dupliquées ou erronées rendent l'analyse peu fiable. La qualité se pilote avec des règles de validation, des contrôles automatisés et des indicateurs de suivi.

Exemple : un même client enregistré sous « Jean Dupont », « J. Dupont » et « Dupont Jean » fausse le nombre réel de clients et le ciblage marketing. Le dédoublonnage doit donc être prévu dès l'ingestion.

Interopérabilité

Les données doivent circuler entre des systèmes hétérogènes — anciens ou récents, internes ou externes — sans perte d'information. Cette circulation repose sur des formats et des API standardisés.

Dans le secteur bancaire, la directive DSP2 impose l'exposition d'API standardisées afin que des applications tierces, comme les agrégateurs de comptes, puissent lire les données de manière interopérable et sécurisée.

Maîtrise des coûts

Le stockage et le traitement à grande échelle sont coûteux, particulièrement dans le cloud où la facturation dépend de l'usage. La conception doit arbitrer la performance et le budget.

Un exemple de dérive consiste à conserver pendant cinq ans tous les logs bruts dans un stockage haute performance alors qu'un stockage archivé, beaucoup moins cher, suffirait pour des données rarement consultées.

Gouvernance des données

La gouvernance précise :

- qui est responsable de chaque donnée ;
- qui peut y accéder ;
- comment elle est documentée ;
- quel est son niveau de confidentialité.

Un **data catalog** recense les tables, leurs définitions, leurs propriétaires et leur confidentialité. Il joue un rôle comparable à celui d'une bibliothèque qui attribue à chaque livre une fiche et un emplacement.

Arbitrer entre les contraintes

Les enjeux sont souvent en tension : renforcer la sécurité peut ralentir l'accès ; augmenter la scalabilité peut accroître les coûts. Concevoir une architecture consiste donc à arbitrer consciemment entre ces contraintes, en fonction des priorités métier et du niveau de risque acceptable.

1.5 Tendances actuelles

Cinq tendances structurantes

Les architectures modernes évoluent sous l'effet de cinq tendances :

- le cloud-native et l'élasticité par défaut ;
- le temps réel et le streaming généralisés ;
- l'intégration de l'IA générative et prédictive ;
- le DataOps et l'observabilité ;
- la démocratisation du self-service.

Cloud-native et élasticité

Les plateformes cloud séparent de plus en plus le stockage et le calcul. AWS, Azure, GCP, Snowflake et Databricks peuvent ajuster les ressources à la charge.

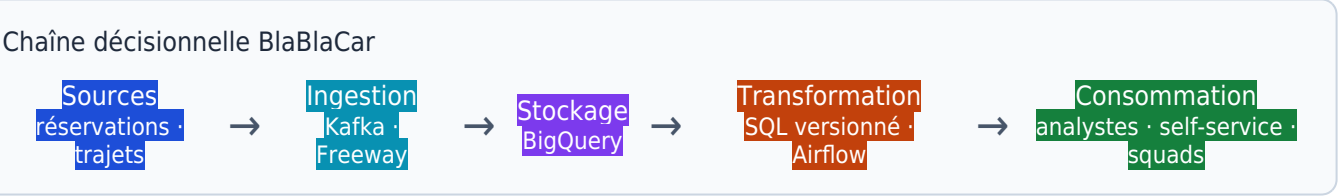
Par exemple, une entreprise lance en quelques minutes un calcul massif pour une clôture mensuelle le dimanche soir, puis éteint la puissance de calcul. Elle paie l'usage réel, contrairement à une infrastructure physique fixe.

Temps réel et streaming

Le traitement batch, effectué par exemple une fois par nuit, cède du terrain au traitement continu en flux. Les décisions peuvent ainsi être prises à la seconde.

Exemple : BlaBlaCar. La disponibilité des trajets affichés est ajustée en continu selon les réservations en cours, plutôt que par une mise à jour nocturne.

Outil ou organisation	Type	Rôle
Apache Kafka	Bus d'événements	Diffusion continue des événements applicatifs
Freeway, outil interne	Ingestion	Application Java qui lit Kafka, valide le schéma et écrit dans BigQuery
BigQuery	Entrepôt cloud	Une table par type et version d'événement
Airflow et SQL versionné	Orchestration et transformation	Pipelines testés comme du code applicatif
Organisation Data Mesh	Gouvernance	Six squads autonomes alignées sur les équipes produit



Intégration de l'IA

L'IA générative et le machine learning ajoutent de nouveaux besoins d'architecture :

- bases vectorielles pour la recherche sémantique ;
- pipelines d'entraînement ;
- gestion de variables réutilisables dans des feature stores.

Un assistant d'entreprise fondé sur le RAG (*retrieval augmented generation*) doit indexer les documents dans un espace vectoriel afin de retrouver les passages pertinents. L'entrepôt de données classique ne suffit donc pas toujours à lui seul.

DataOps et observabilité

Inspiré du DevOps, le **DataOps** automatise les tests, le déploiement et la surveillance des pipelines. L'objectif est de détecter les anomalies avant qu'elles n'atteignent les utilisateurs métier.

Des outils comme Great Expectations ou Monte Carlo peuvent alerter automatiquement si le nombre de lignes

chargées chute anormalement. Il s'agit d'appliquer aux données une logique comparable aux tests automatisés du logiciel.

Self-service et démocratisation

Le self-service permet aux utilisateurs métier d'accéder directement à des données certifiées par l'équipe data, grâce à des outils BI no-code. Ils n'ont plus besoin de solliciter l'IT pour chaque requête.

Exemple : un responsable marketing construit lui-même un tableau de bord de campagnes dans Power BI ou Looker, connecté à des données gouvernées, sans ticket ni développement spécifique.

1.6 Synthèse du chapitre

Les idées essentielles

- Une architecture de données organise le parcours de la donnée, de sa source jusqu'à la décision.
- Elle est devenue un enjeu stratégique et pas seulement technique, notamment avec l'essor de l'IA.
- Les familles centrée données, événementielle, orientée services, Data Mesh, Lambda/Kappa et Lakehouse répondent à des besoins différents et peuvent être combinées.
- Toute conception doit prendre en compte la scalabilité, la sécurité, la qualité, l'interopérabilité, le coût et la gouvernance.
- Le cloud, le temps réel, l'IA, le DataOps et le self-service redessinent les architectures actuelles.

Résumé pédagogique

Une bonne architecture met en cohérence les sources, l'ingestion, le stockage, le traitement et la consommation. Elle transforme des données hétérogènes en informations fiables, puis en décisions. Le choix d'une architecture n'est jamais purement technologique : il dépend du métier, du niveau de temps réel attendu, des volumes, des risques, des compétences et du budget.

L'architecture centrée sur les données privilégie un référentiel partagé ; l'architecture événementielle privilégie la réaction en temps réel ; les microservices distribuent la responsabilité par domaine ; le Data Mesh traite les données comme des produits ; Lambda et Kappa organisent la coexistence du temps réel et de l'historique ; le Lakehouse rapproche la souplesse du Data Lake des garanties du Data Warehouse.

Question de vérification

Situation : une entreprise doit absorber un pic de trafic multiplié par dix pendant les soldes sans tomber.

Enjeu directement concerné : la **scalabilité**. Une architecture cloud-native élastique peut ajuster automatiquement les ressources, à la différence d'une infrastructure fixe.

Mise en pratique

Identifier trois architectures de données réelles dans des entreprises différentes — par exemple un e-commerçant, une banque et un service de streaming. Pour chaque cas :

- préciser la famille d'architecture dominante ;
- décrire les principaux flux et composants ;

- expliquer pourquoi cette architecture est importante pour l’entreprise ;
- relier le choix aux enjeux de volume, temps réel, sécurité, qualité, coût ou gouvernance.

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/eadl/bloc5/fm1/chap1?rev=1788102330

Last update: **2026/08/30 17:05**