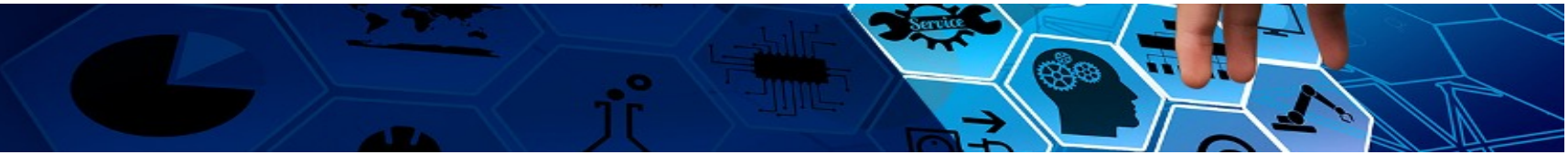
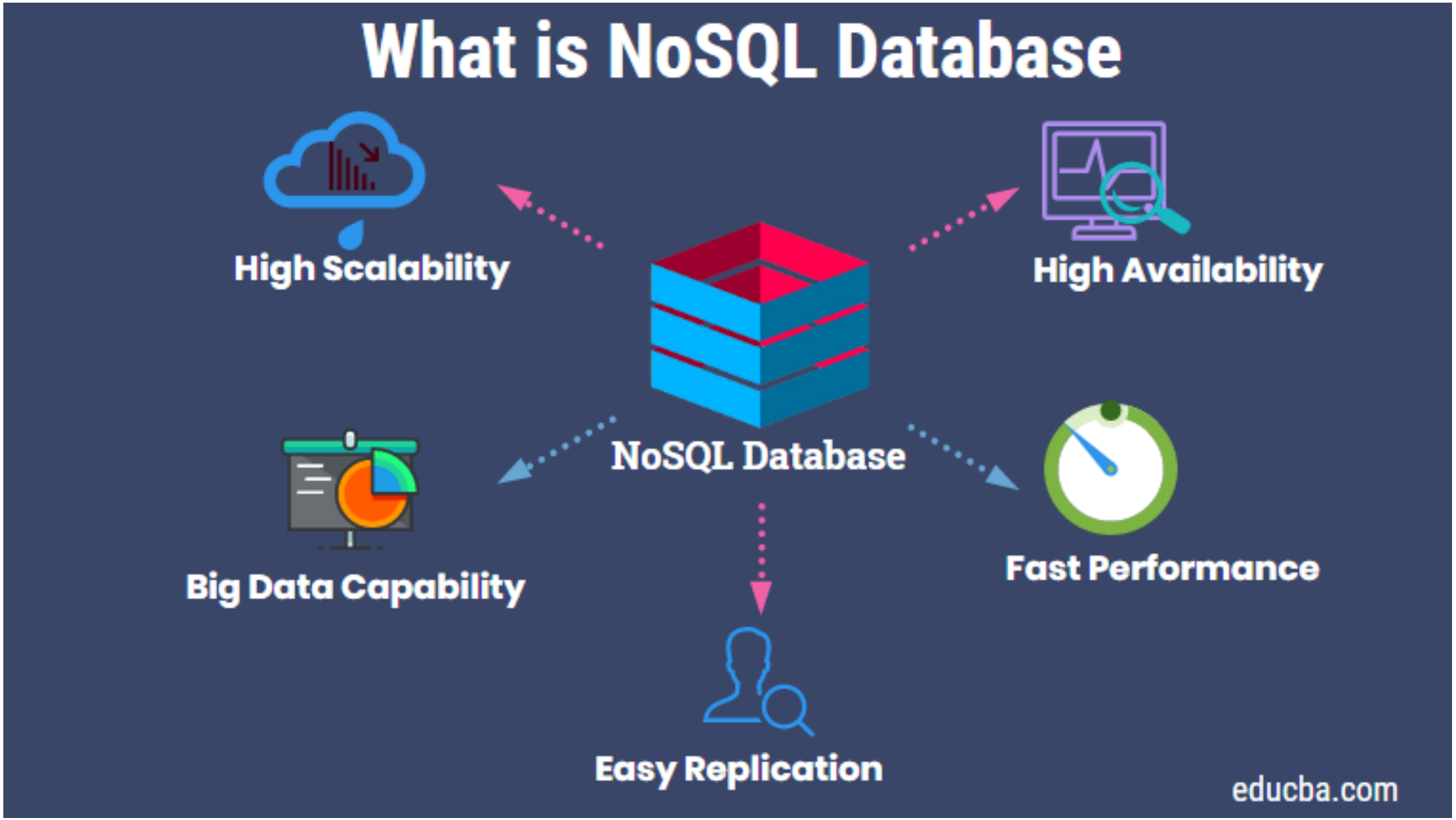

NO SQL
Know all about Non relational databases

Plan



- Les architectures distribuées
- Distribution des données
- Modèle de données
- Les familles NoSQL


What is NoSQL Database
High Scalability
High Availability
NoSQL Database
Big Data Capability
Fast Performance
Easy Replication
educba.com

Pourquoi le NoSQL ?

A partir de 2005, de nouvelles approches de traitement de données voient le jour sous le nom de base de données non relationnelles : ie Not Only SQL

Avantages :

L'évolutivité se fait de manière horizontale (pour augmenter les performances on ajoute des nouvelles machines).

Les données sont distribuées sur plusieurs machines (sharding).

La représentation des données est « souple » par l'absence de schéma (schemaless).

La majorité des solutions sont Open Source.

Inconvénients :

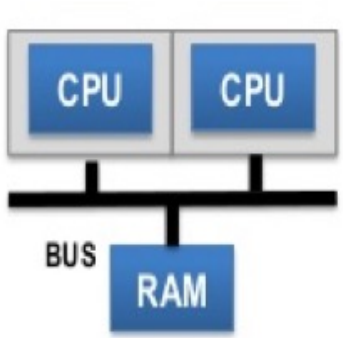
Il n'existe pas de langage d'interrogation standardisé (tentative avec le UnSQL).

Une cohérence éventuelle.

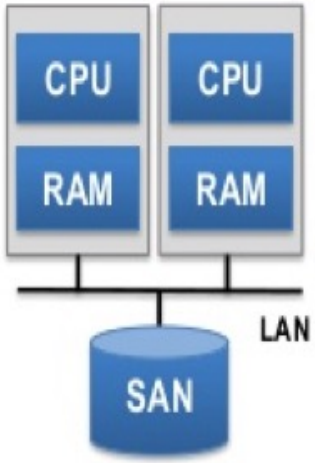
L'écriture de requêtes complexes est difficile à mettre en œuvre

L'offre NoSQL est segmentée en plusieurs familles où chacune répond à un besoin précis.

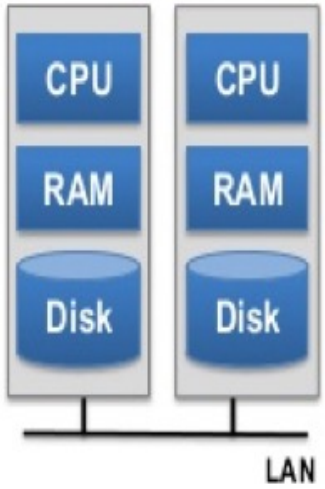
Les architectures



Shared Everything

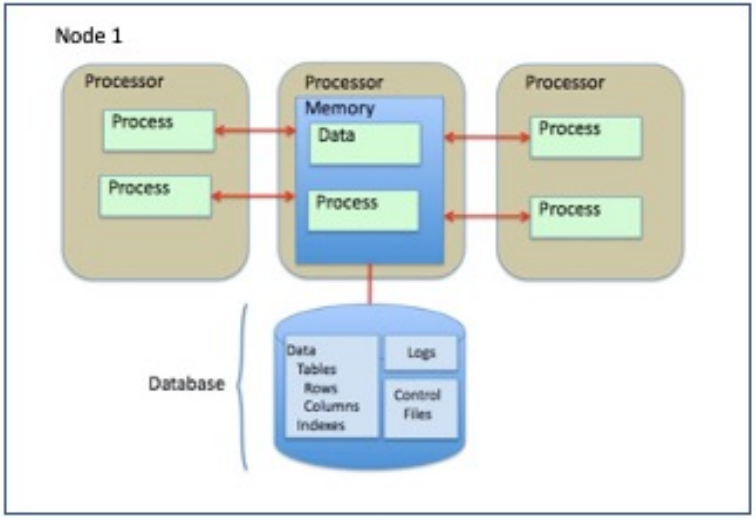


Shared Disk



Shared Nothing

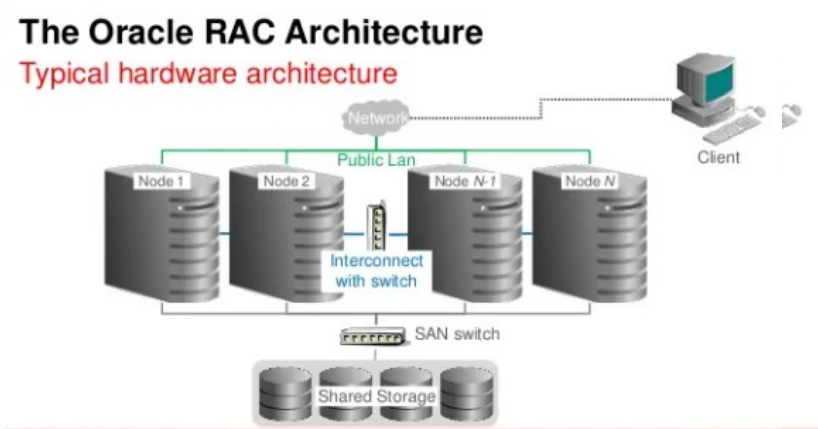
Les architectures



Shared-Everything / Partage tout

Une base de données mono-serveur où chaque processus partage la même mémoire, le même processeur et les mêmes ressources de disque. La capacité de calcul (CPU), la mémoire et les accès aux disques sont limités par la configuration matérielle. La gestion des verrous est intra-processus.

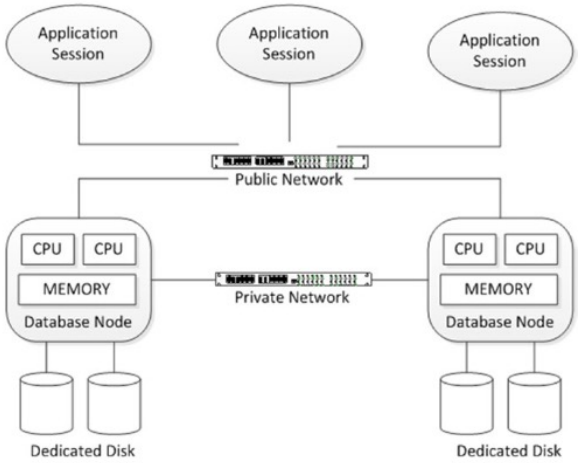
Les architectures



Shared-disk / disque partagé

Les processus de base de données peuvent exister sur des nœuds distincts du cluster et avoir accès aux cores CPU et à la mémoire du serveur sur lequel ils résident. Cependant, chaque processus a un accès égal aux périphériques de disque, qui sont partagés entre tous les nœuds du cluster . On augmente la capacité de calcul et de mémoire mais les accès au stockage restent limités. La gestion des verrous est intra-processus et inter-machines.

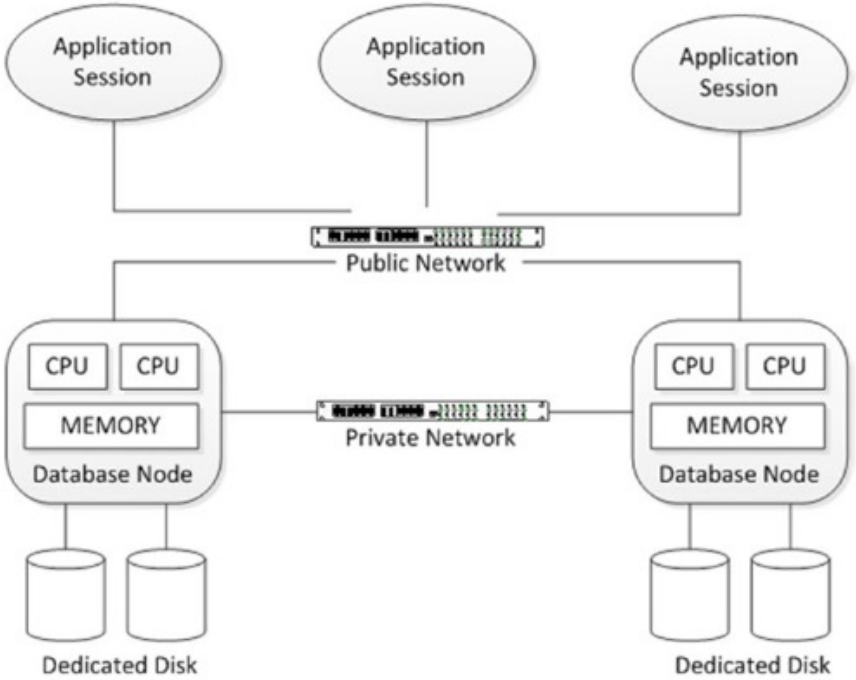
Les architectures

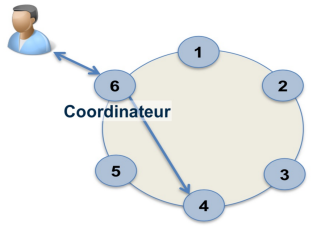


Shared-Nothing / Sans Partage

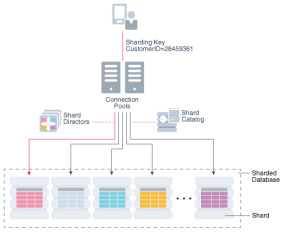
Chaque nœud du cluster a accès non seulement à sa propre mémoire et CPU, mais également à des périphériques de disque dédiés. Le cluster peut s'étendre à l'infinie sans aucune restriction de ressource. Dans la théorie, les performances dépendent linéairement du nombre de machines dans le cluster. Cependant, dans cette solution le réseau joue un rôle majeur pour la communication entre les machines.

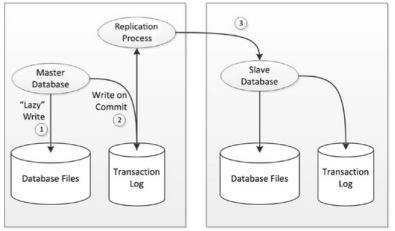
Les architectures MPP





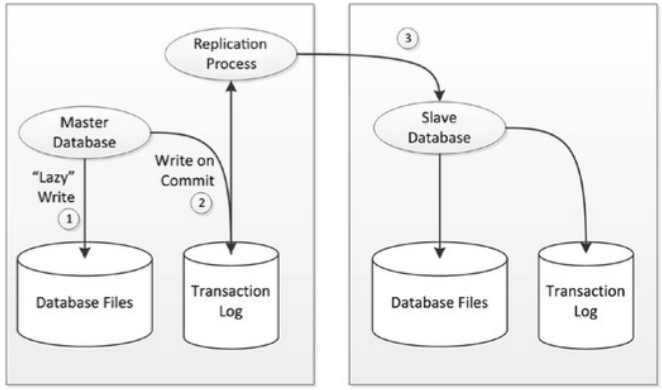
Architectures logicielles sans maître





Architectures logicielles avec maître

Architecture logicielle distribuée avec maître



La réplication est généralement une première étape vers la répartition de la charge en lecture sur plusieurs serveurs.

- Un modèle souvent utilisé au début du web 2.0.

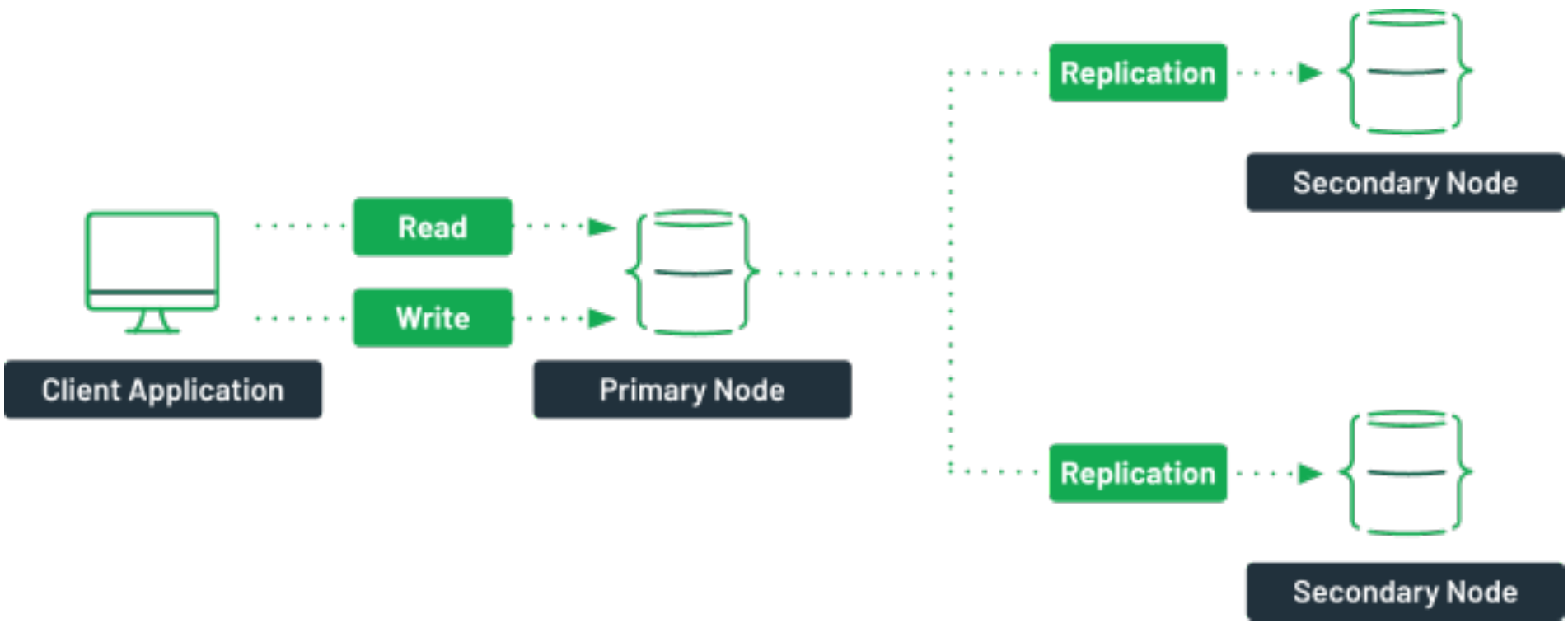
Une solution améliorant la disponibilité.

Les données sont répliquées via les journaux de transactions mais un compromis doit être fait entre performance et cohérence :

Si la réplication est synchrone, la transaction sera validée uniquement lorsque celle-ci sera validée sur le Master ET le Slave (Performances dégradées).

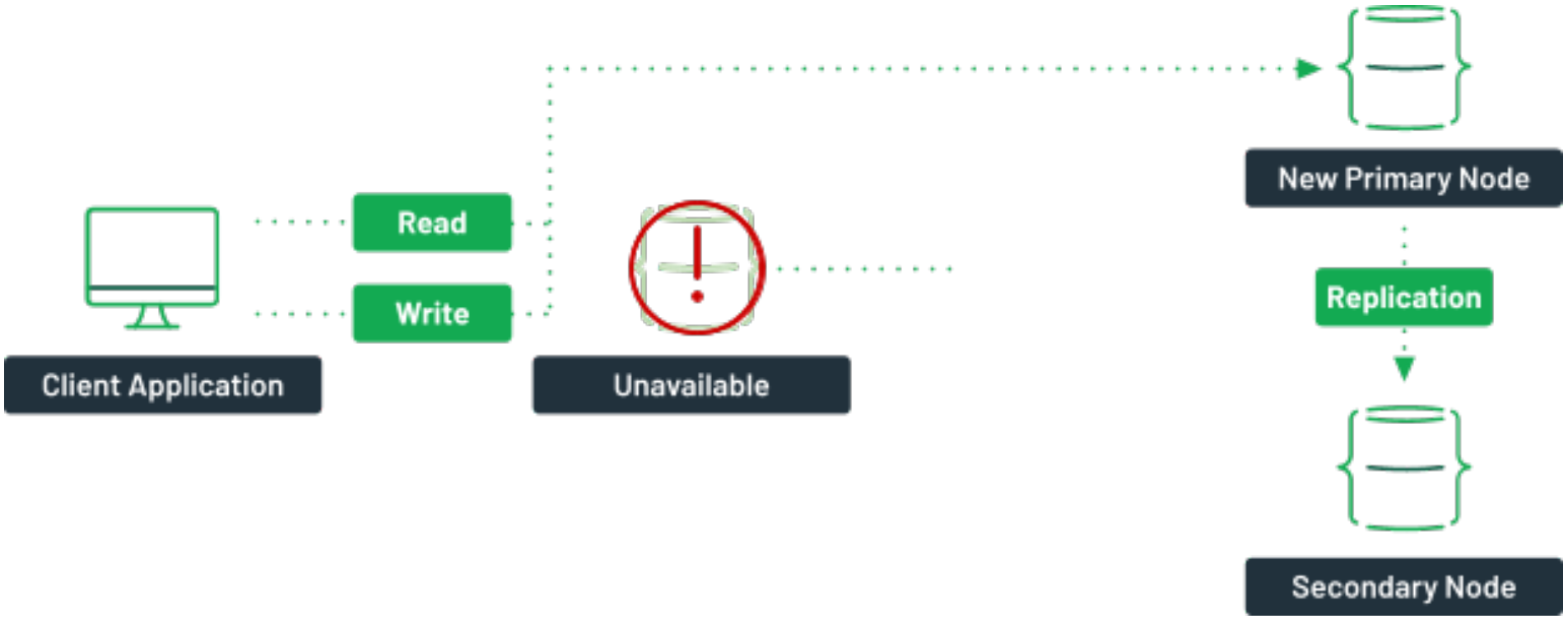
Si la réplication est asynchrone, il est possible de lire sur le Slave des données périmées.

Architecture logicielle distribuée avec maître



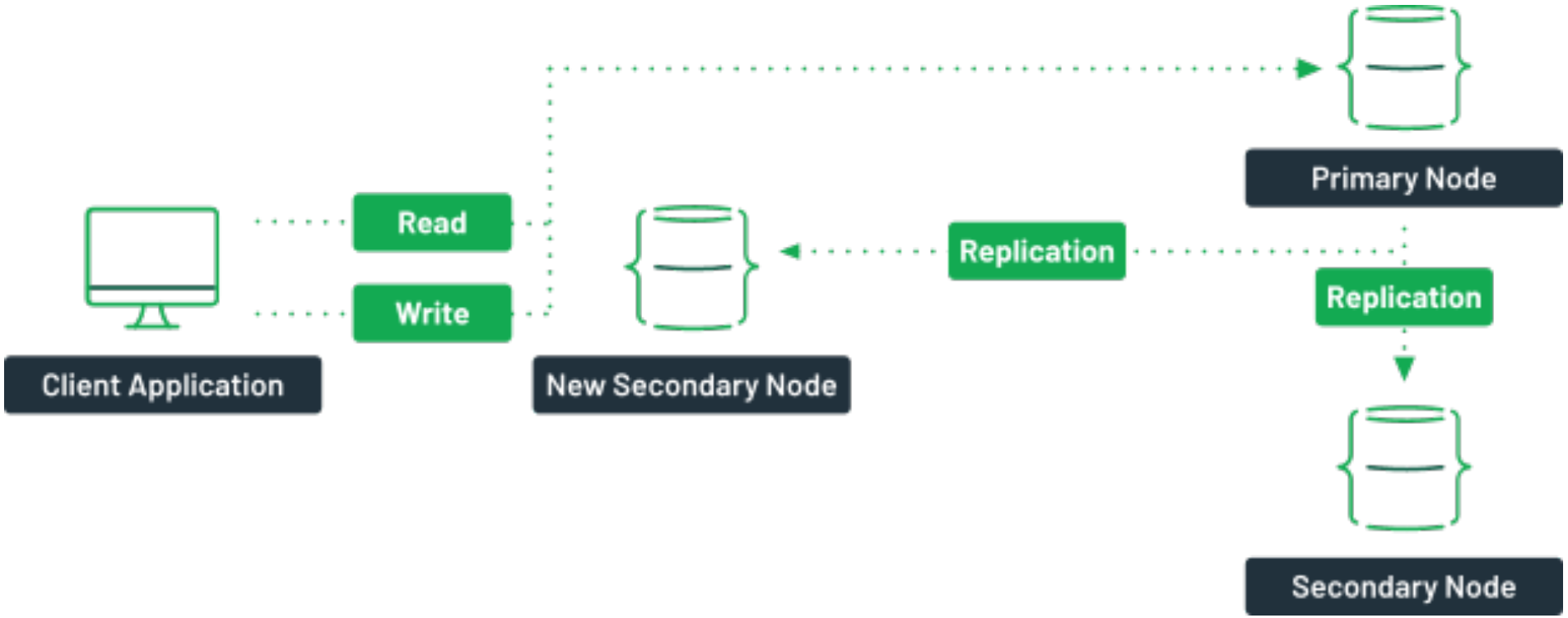
MongoDB.

Architecture logicielle distribuée avec maître



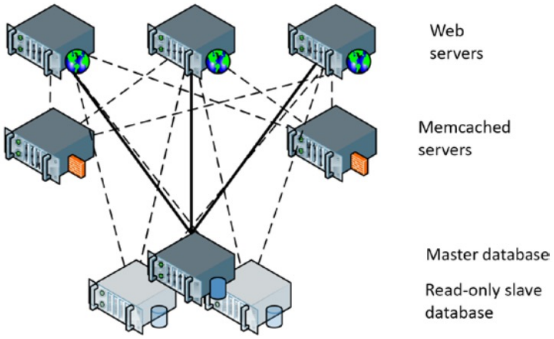
MongoDB.

Architecture logicielle distribuée avec maître



MongoDB.

Architecture logicielle distribuée avec maître: Cache de données



Scaling up by adding 2 web servers, 2 Memcached servers, and 2 read-only database slaves

Les données ne sont pas disponibles dans les serveurs Memcached alors l'application interroge les réplicas de bases de données pour ensuite les conserver dans un serveur Memcached

Les données sont disponibles dans un serveur Memcached, l'application obtient donc ses données en quelques nano secondes (base de données clé-valeur).

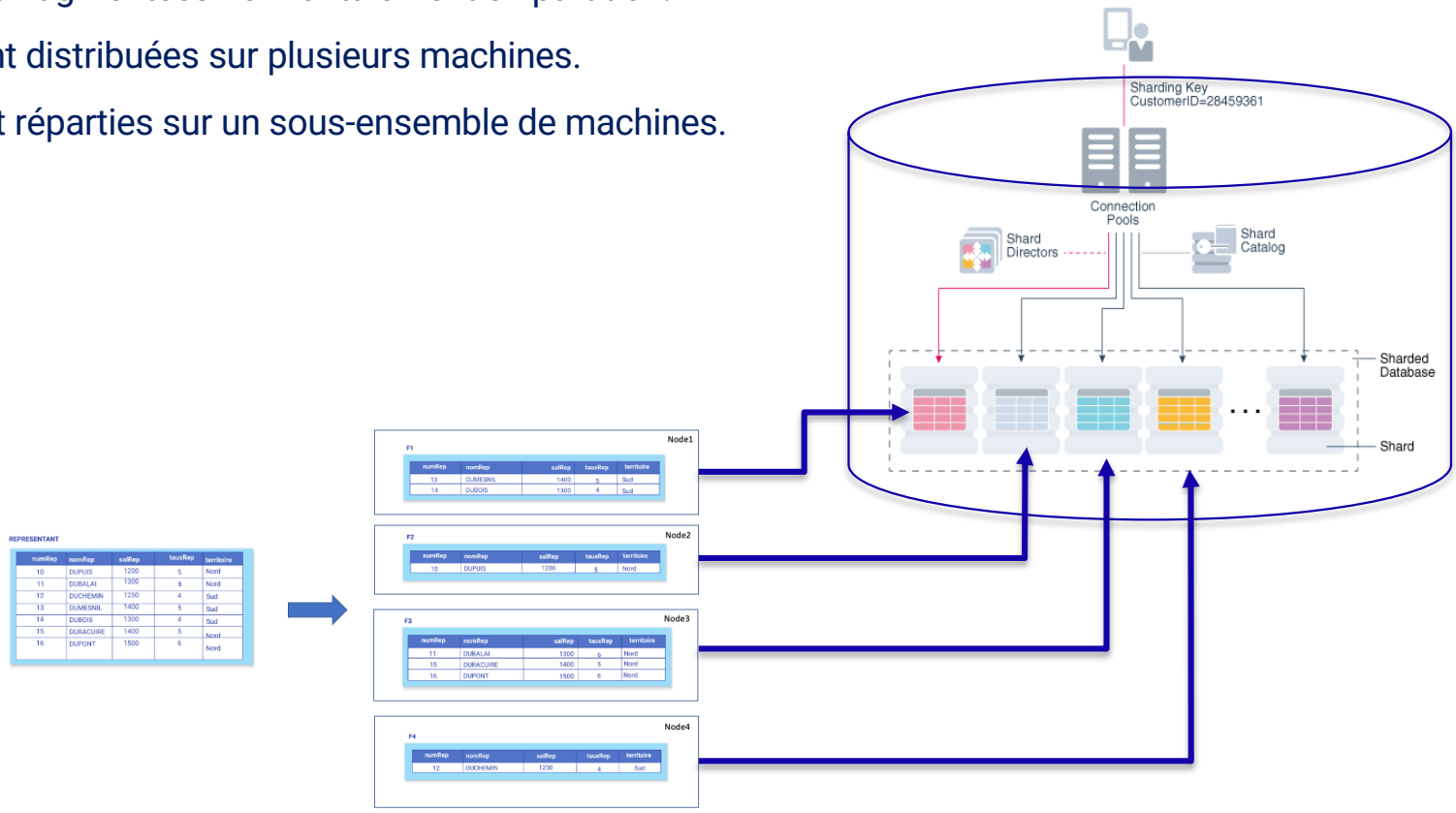
Memcached jumelé à la réplication permet de supporter une charge en lecture beaucoup plus importante. Malheureusement, l'activité en écriture, est quant à elle, limitée au master. Une contrainte qui amènera certains développeurs à explorer d'autres solutions.

Architecture logicielle distribuée avec maître : Sharding

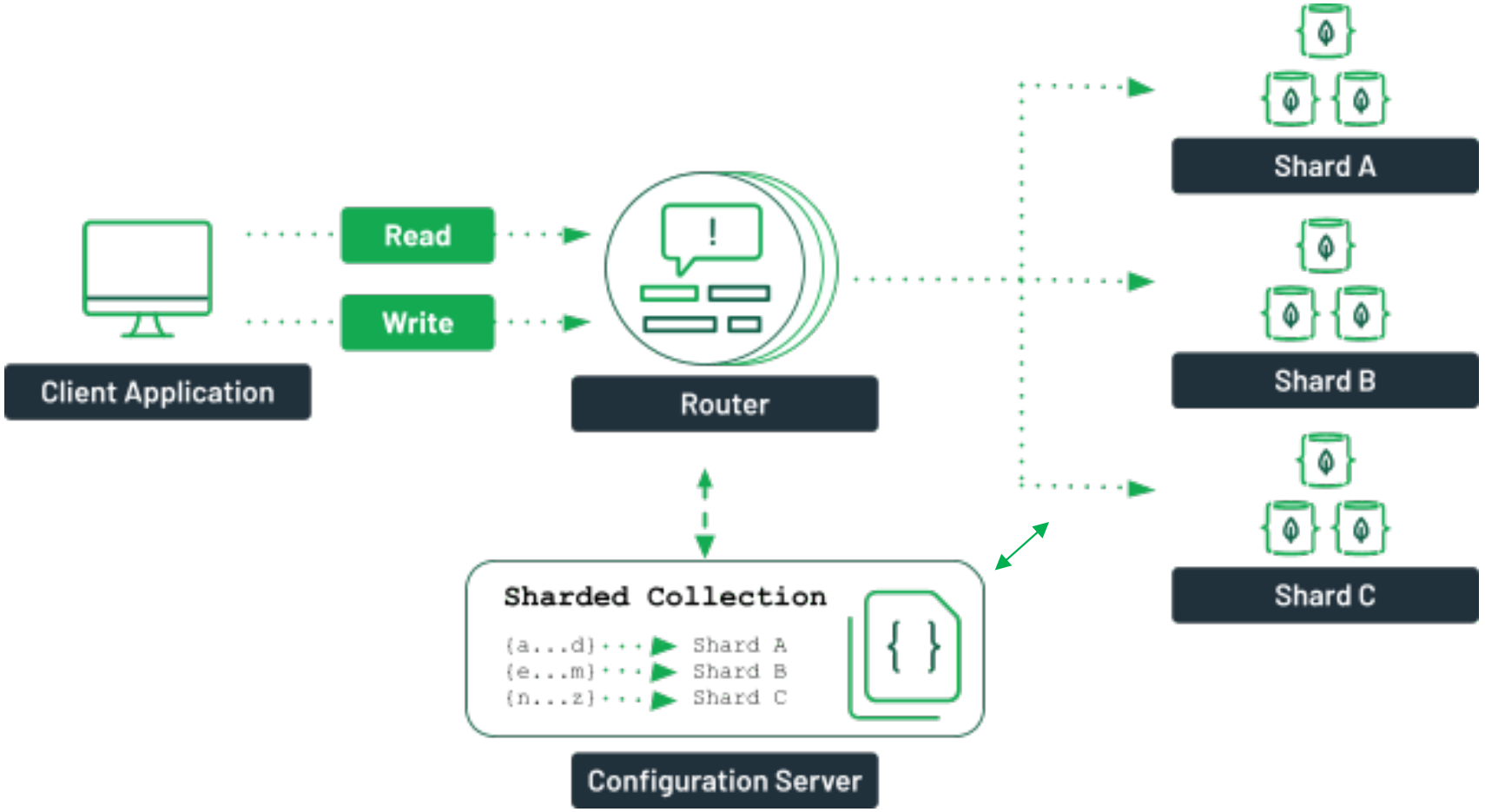
Les données sont fragmentées horizontalement en partition.

Les partitions sont distribuées sur plusieurs machines.

Les requêtes sont réparties sur un sous-ensemble de machines.

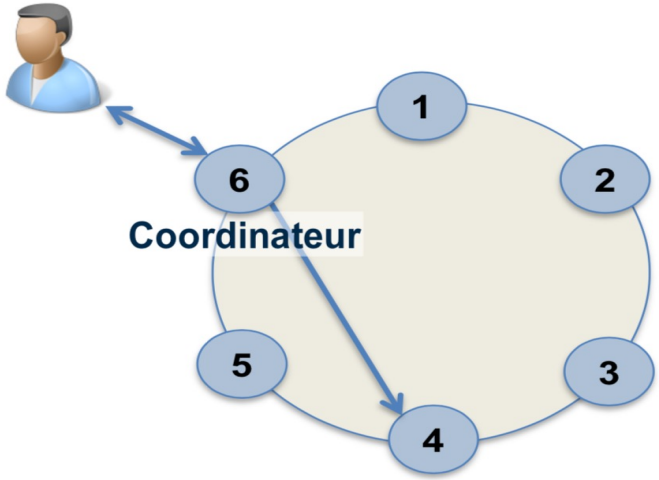


Architecture logicielle distribuée avec maître : Sharding



MongoDB.

Architecture logicielle distribuée sans maître



Tous les nœuds du cluster sont identiques, ils assurent tous les services de :

- cluster manager
- coordinator
- router (fonction de Hash)

Tous les nœuds jouent un rôle de coordinateur lequel redirige l'utilisateur vers le nœuds possédant les données.

L'éclatement des données est géré automatiquement par le coordinateur.

Théorème de CAP

- Illustration du théorème de CAP
- Prenons un jeux de données qu'on répartit sur plusieurs nœuds.
- En cas de panne d'un nœud, le jeux de données n'est plus complet et par conséquent :
  - Si le système répond alors la réponse ne sera pas cohérente, on parle de système **AP** **A**vailable - **P**artitioning-Fault.
  - Si le systéme attend que le résultat soit cohérent alors l'utilisateur peut attendre indéfiniment, le systéme privilégie la cohérence il est alors dit **CP** pour **C**onsistency - **P**artitioning-Fault.

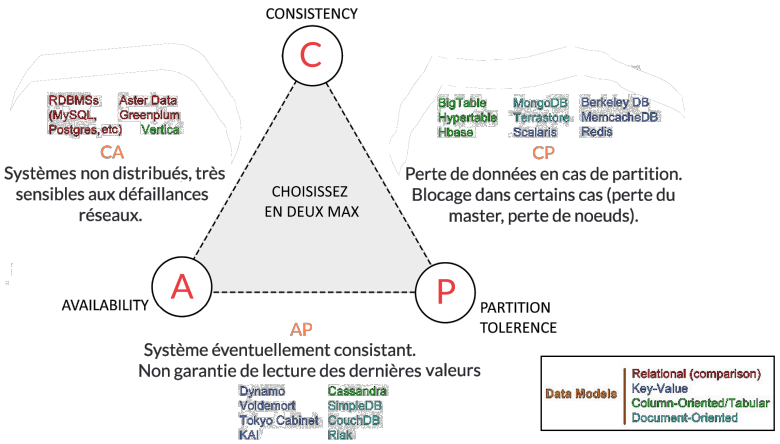
Théorème de CAP

Le scientifique Eric Brewer constate que tous les systèmes distribués de stockage de données ne peuvent garantir que 2 des 3 propriétés suivantes :

Consistency : la cohérence signifie que tous les nœuds du système voient les mêmes données au même moment.

Availability : Toutes les requêtes obtiendront un résultat cohérent ou non.

Partition tolerance : un système partitionné est un système dont les données sont réparties sur plusieurs nœuds/machines, aucune panne moins importante qu'une coupure totale du réseau ne doit l'empêcher de répondre.



Les systèmes boursiers ? CA

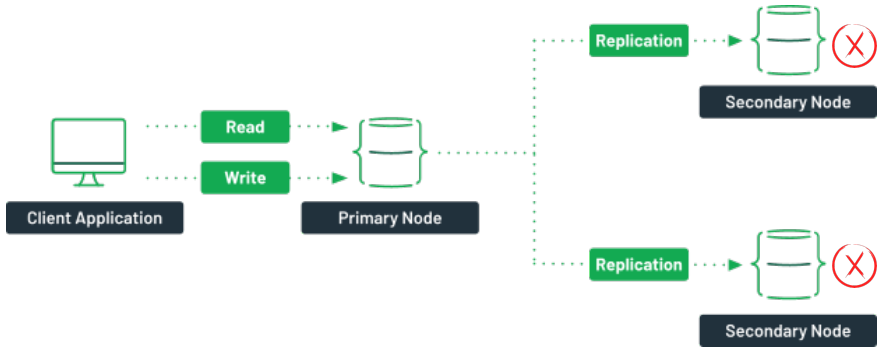
Des réseaux de guichets automatiques ? CP

La gestion de traces applicatives ? PA

La gestion de la cohérence

- Différentes formes de cohérences :
  - Strong/strict consistency : tous les serveurs sont cohérents après une mise à jour (e.g., transactions distribuées)
  - Weak consistency : aucune garantie de cohérence à terme (e.g., perte d'une mise à jour suite à la défaillance d'une serveur)
  - Eventual consistency (cohérence à long terme) : si aucune nouvelle mise à jour n'est effectuée, alors toutes les prochaines lectures finiront à lire plus ou moins à long terme les mêmes données.

La gestion de la cohérence : Eventual consistency



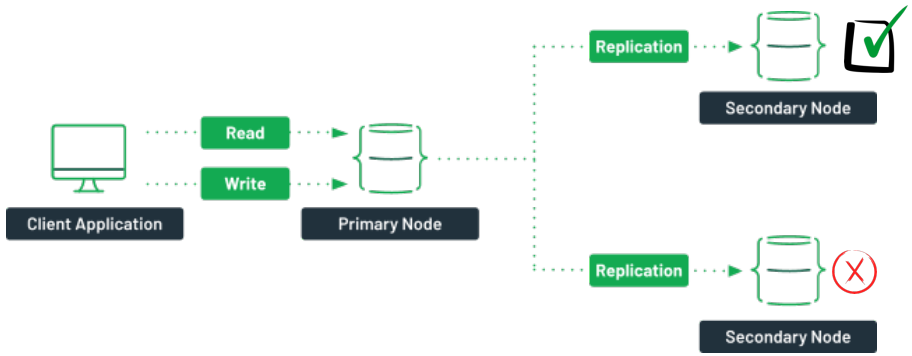
Facteur de réplication FR : 3
Validation en écriture W : 1
Le système confirme l'écriture dés que l'écriture sur le primaire est validée et la réplication sur les 2 nœuds restants se fait de manière asynchrone.

Facteur de réplication FR : 3
Validation en lecture R : 1
Le système renvoie le résultat à partir de n'importe quel nœud.
Résultat éventuellement incohérent.

Facteur de réplication FR : 3
Validation en lecture R : 2
Le système renvoie le résultat en se mettant d'accord sur le résultat avec un second nœud.
Résultat éventuellement incohérent.

Facteur de réplication FR : 3
Validation en lecture R : 3
Le système renvoie le résultat en se mettant d'accord entre les 3 nœuds.
Résultat cohérent

La gestion de la cohérence : Eventual consistency



Facteur de réplication FR : 3
Validation en écriture W : 2
Le système confirme l'écriture dés que l'écriture sur le primaire est validée et que la réplication est validée sur au moins 1 réplica.

La cohérence des données lues est établie selon le niveau de cohérence demandé en écriture :

W + R > FR

Facteur de réplication FR : 3
Validation en lecture R : 1
Le système renvoie le résultat à partir de n'importe quel nœud.
Résultat éventuellement incohérent.

Facteur de réplication FR : 3
Validation en lecture R : 2
Le système renvoie le résultat en se mettant d'accord sur le résultat avec un second nœud.
Résultat cohérent.

Facteur de réplication FR : 3
Validation en lecture R : 3
Le système renvoie le résultat en se mettant d'accord entre les 3 nœuds.
Résultat cohérent

SGBD Relationnel où NoSQL ?

OU

Relationnel

Cohérence

NOSQL

Extensibilité

Le NoSQL

Les règles ACID sur une base de données distribuées sont difficiles à respecter mais sont elles indispensables ?

Les bases NoSQL vont alors être décrites sur le modèle BASE :

Basicly **A**vailable, les SGBD conforment aux régles BASE intégrerons les techniques de réplication et de partitionnement (sharding) pour répondre, quelque soit la panne rencontrée sur le cluster.

Soft state, l'état du système change en permanence par conséquent la cohérence des données n'est pas garantie. Toutefois, les SGBD fournissent des options pour définir un niveau de cohérence ajustable.

Eventually consistent, le système avec le passage à l'échelle va passer par des états de synchronisation qui ne garantiront pas la cohérence des données suivant le nœud et le moment où vous les consulter. Les données seront donc éventuellement consistantes. Le modèle BASE n'impose pas une cohérence immédiate ce qui ne signifie pas qu'il ne l'atteint jamais. Cependant, jusqu'à ce qu'il le fasse, les lectures de données peuvent être incohérentes.

Le NoSQL : Not only SQL

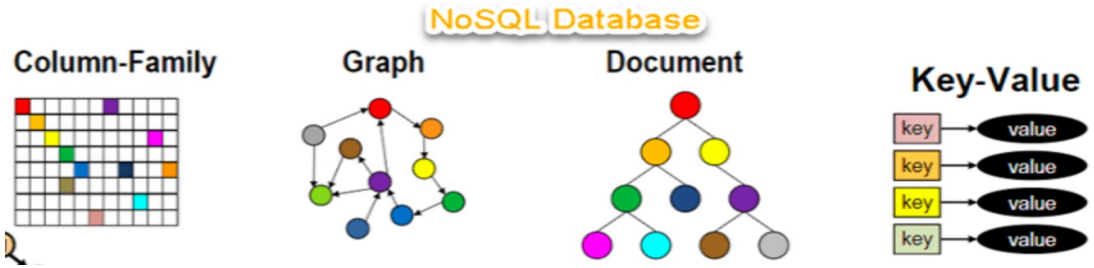
Des SGBD capable de proposer :

Un modèle de données sans schéma/structure imposé.

Des temps de réponse très faibles quelque soit la volumétrie.

Une évolutivité horizontale (PA ou PC).

Différentes solutions sont alors implémentées :



Le NoSQL

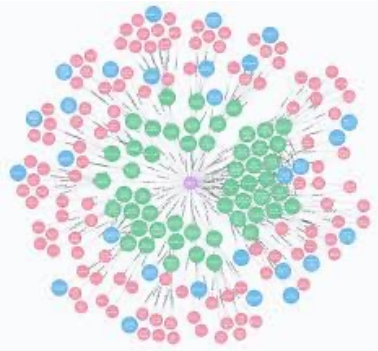
Orienté Clé-Valeur

Key		Value
City	→	Denver
State	→	Colorado
Country	→	USA

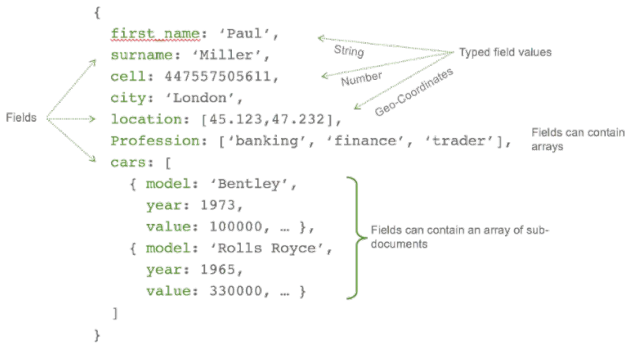
Orienté Colonne

train6101	PARIS 06:07	VALENCE 08:19	AVIGNON 09:07	AIX 09:30	MARSEILLE 09:41
train2917			AVIGNON 11:30		11:59 MARSEILLE

Orienté Graphe

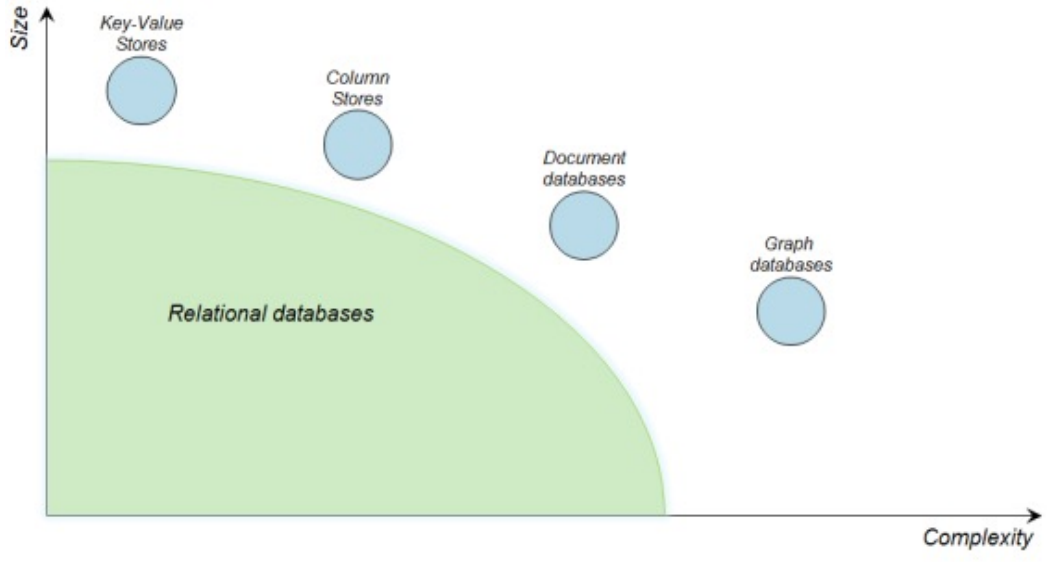


Orienté Document



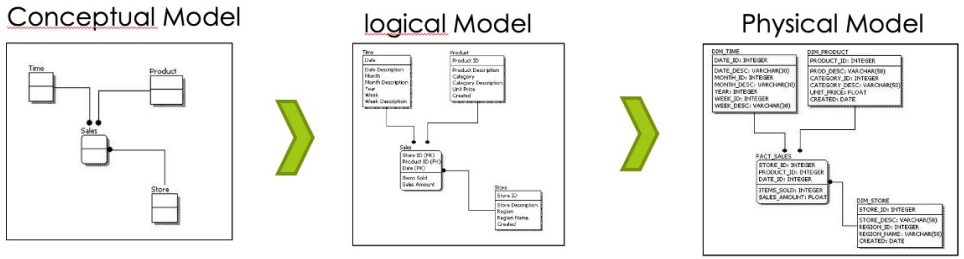
SGBD Relationnel où NoSQL ?

Positionnement des bases de données NoSQL et relationnelles



= connectivity between data

Le NoSQL et la modélisation



Modèle de données conceptuel : il explique ce que le système doit contenir en ce qui concerne les données et comment elles sont liées. Ce modèle est généralement construit avec l'aide des experts métier. Il représente la logique métier de l'application et est souvent utilisé comme base pour le modèle logique.

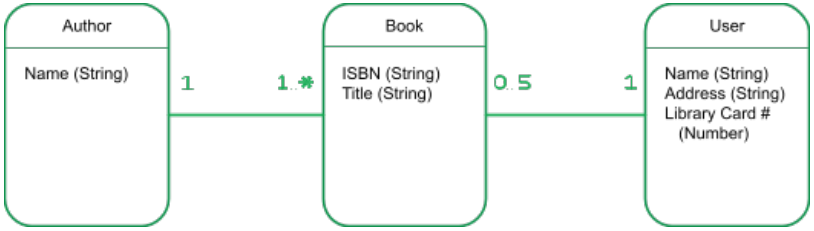
Modèle de données logique : Le modèle de données logique décrit comment les données seront structurées. Dans ce modèle, la relation entre les entités est établie à un niveau élevé. Vous listerez également les attributs des entités représentées dans le modèle.

Modèle de données physique : le modèle de données physique représente la manière dont les données seront stockées dans le SGBD choisi. Avec ce modèle, vous établiriez vos contraintes de domaine (clés primaire et étrangère) dans une base de données relationnelle ou décideriez d'intégrer ou de lier vos données dans une base de données orienté document, graphe, colonne ou clé-valeur. Vous établirez également les types de données pour chacun de vos champs pour construire votre schéma de base de données.

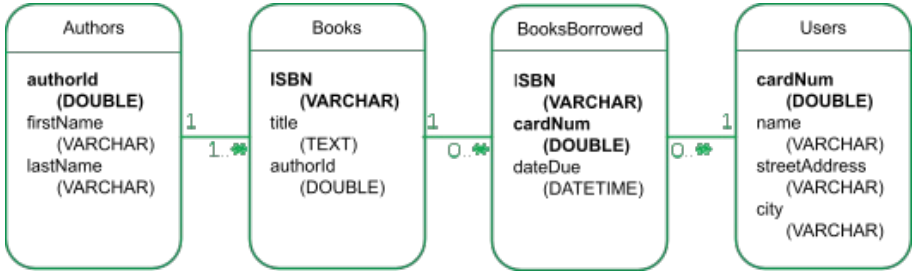
Le NoSQL et la modélisation

Modèle conceptuel

Modèle logique

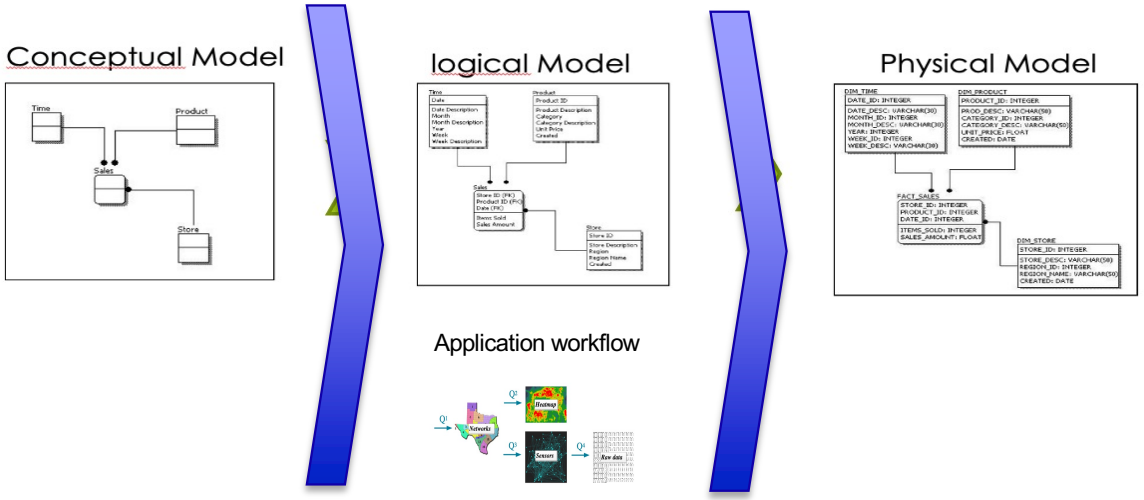


Modèle Physique



@mongoDB

Le NoSQL et la modélisation



Dans le monde NoSQL, la performance est un ingrédient décisif et c'est pour cette raison que le modèle logique est souvent marqué par la nature des accès aux données : on parle alors d'une **approche Query-Driven.**

Avec le NoSQL, il n'est pas rare de dénormaliser les données afin d'améliorer les performances.

La flexibilité qu'offre le NoSQL sur les structures de données facilite les évolutions du modèle de données.

On ne cherche pas un modèle parfait mais un modèle qui satisfait les contraintes métiers en terme de performance et de disponibilité.

Le NoSQL : la famille orientée clé/valeur

Les données sont simplement représentés par un couple clé/valeur
Interrogation et manipulation seulement à partir d'une clé (SET – GET ...)
L'intégrité des données doit être gérée par l'application.

Avantage

Recherche très performante.
Peut répondre à tous les besoins au détriment de la simplicité.

Inconvénient

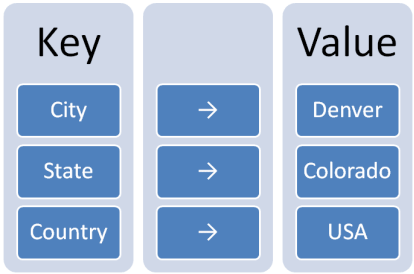
Aucun schéma
Pas de garantie d'intégrité

Exemple d'application:

Gestion d'un panier web, des cookies

Bases de données NoSQL :

Redis, Dynamo, Riak



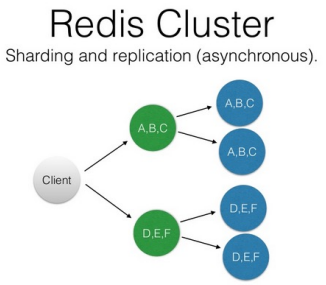
Le NoSQL : la famille orientée clé/valeur

Le nom des clés peut contenir des données

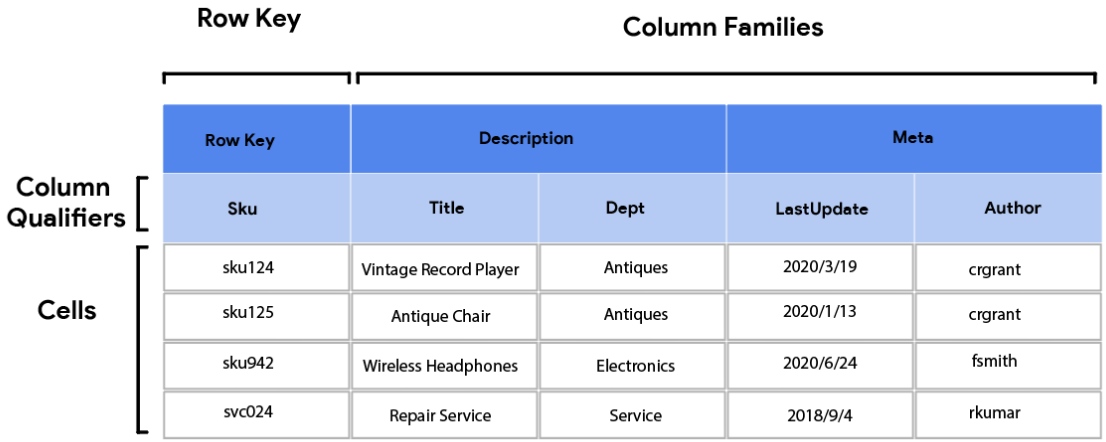
Key	Value
User:1:name	Bob
url:index	https://www.univ-lyon1.fr

Possibilité d'étendre la base de données à plusieurs machines :

- Sharding
- Réplication



Le NoSQL : la famille orientée colonne



Row Key	Description		Meta	
Sku	Title	Dept	LastUpdate	Author
sku124	Vintage Record Player	Antiques	2020/3/19	crgrant
sku125	Antique Chair	Antiques	2020/1/13	crgrant
sku942	Wireless Headphones	Electronics	2020/6/24	fsmith
svc024	Repair Service	Service	2018/9/4	rkumar

Les bases de données orientées colonne s'appuient sur le principe de fragmentation verticale.

On stocke les données en segmentant les données par ensemble de colonnes qu'on appelle des familles.

Les familles de colonnes sont définis de manière à ce que les colonnes accédées ensembles soit stockées ensembles.

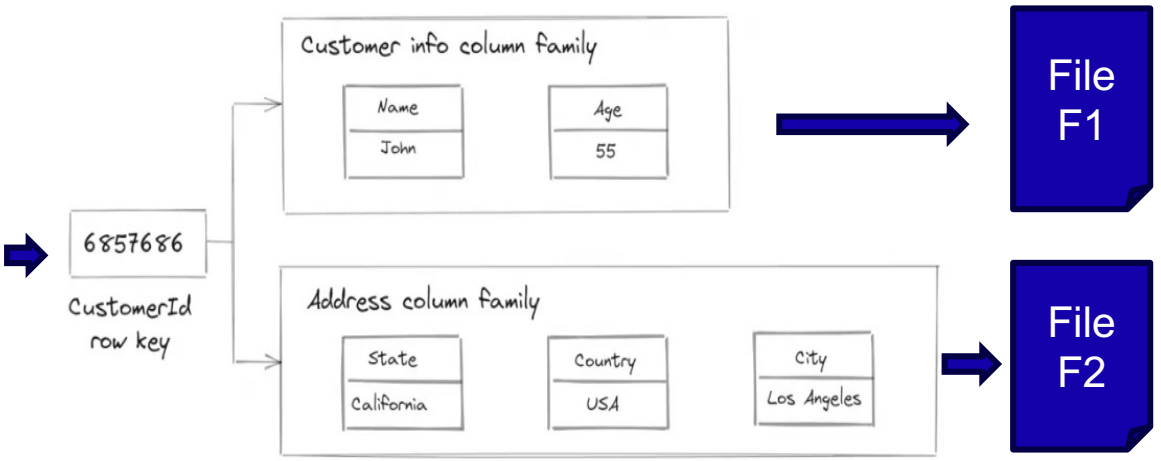
Toutes les colonnes appartenant à une même famille de colonne sont stockées dans des fichiers physiquement indépendants pour améliorer les performances.

Le NoSQL : la famille orientée colonne

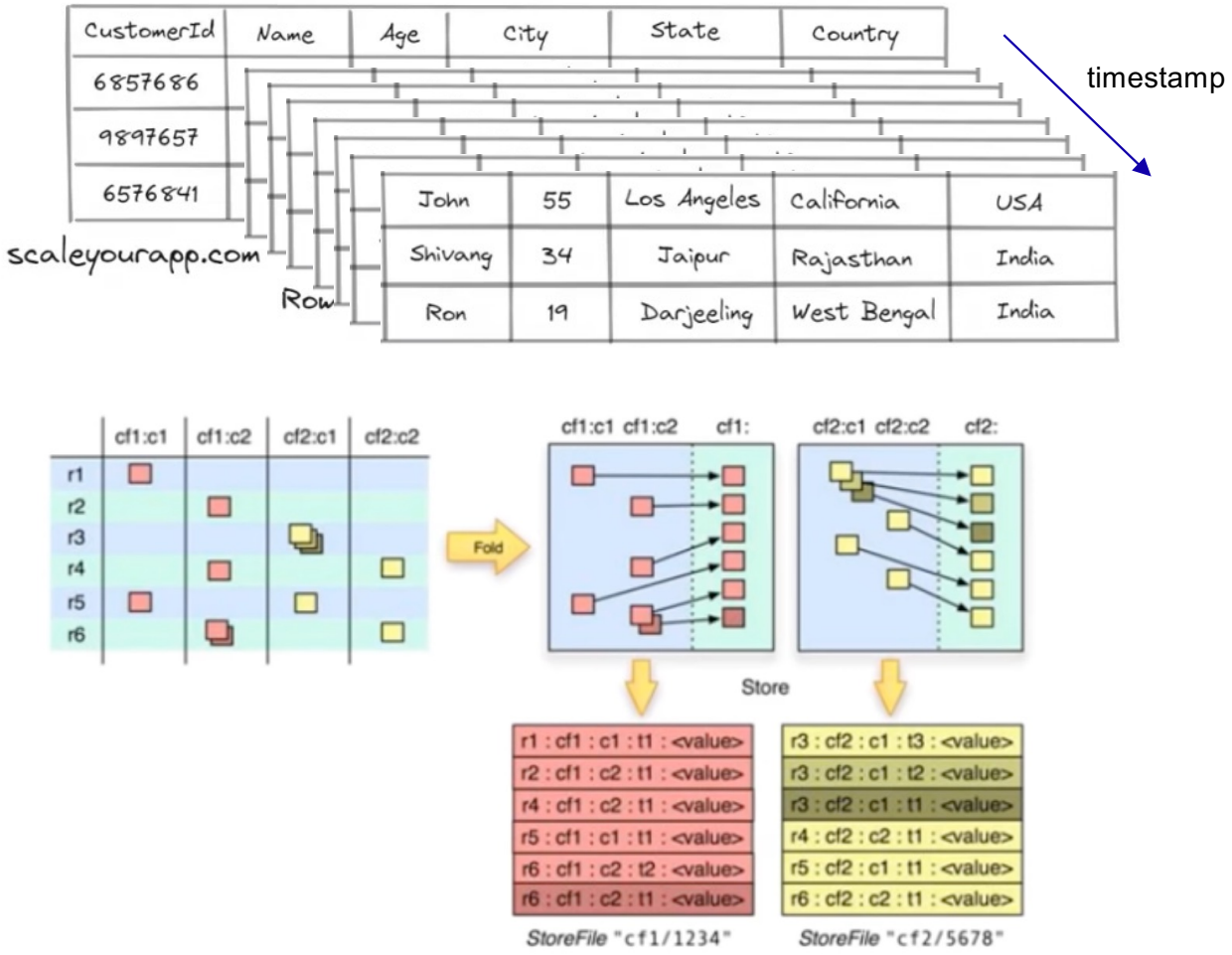
CustomerId	Name	Age	City	State	Country
6857686	John	55	Los Angeles	California	USA
9897657	Shivang	34	Jaipur	Rajasthan	India
6576841	Ron	19	Darjeeling	West Bengal	India

scaleyourapp.com

Row-oriented data storage



Le NoSQL : la famille orientée colonne

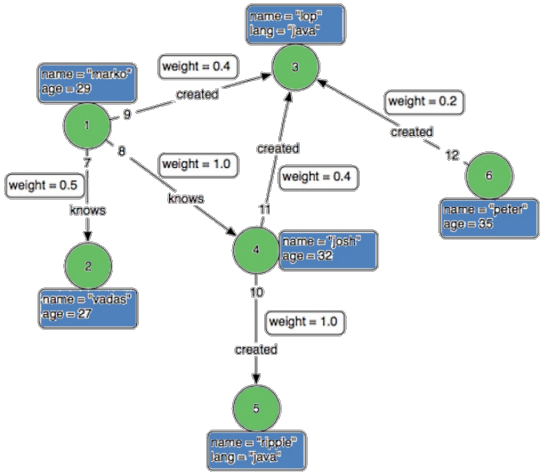


Le NoSQL : la famille orientée colonne

- BigTable(by Google) est la première base de données orientée colonne.
- Très performant pour les opérations de lecture et notamment sur les opération d'agrégation (SUM, AVG …).
- Le découpage des lignes par famille de colonne pénalise les écritures.
- Les jointures ne sont pas supportées.
- Les données sont souvent dupliquées pour assurer la performance au détriment de la cohérence.
- Exemple d'utilisation
  - Les entrepôts de données de nouvelle génération.
  - Ebay pour la gestion des commandes.
- Bases de données NoSQL :
  - HyperBase, HBASE …
  - Attention, CASSANDRA est une base de données orientée clé valeur.

Le NoSQL : la famille orientée Graphe

- Basé sur la théorie des graphes
- Les données sont connectées entre elles sous forme de graphe.
- un langage de requête graphe déclaratif, simple et efficace.
- Dans un graphe, une entité (ex Personne) est stockée comme un nœud et la relation qui lie une entité avec une autre entité est représentée par un arc dirigé.
- Avantages
  - Adapté pour les réseaux sociaux
  - Langage standardisé
  - Le moteur peut être transactionnel (ACID)
- Inconvénient
  - L'extensibilité horizontale est très limitée.
- Bases de données NoSQL :
  - Neo4j
  - InfiniteGraph
  - OrientDB



Le NoSQL : la famille orientée Document

- Un document est une collection de clé/valeur ou la valeur peut être elle-même un document.
- Un document est basé sur une structure en arbre.
- Une collection est un ensemble de documents qui peuvent être hétérogènes
  - Le contenu des documents est interrogeable
  - Avantage
    - Technologie connue surtout dans le web.
  - Inconvénient
    - Ne supporte pas les jointures.
  - Bases de données NoSQL :
    - MongoDB
    - CouchDB
    - Riak

JSON

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

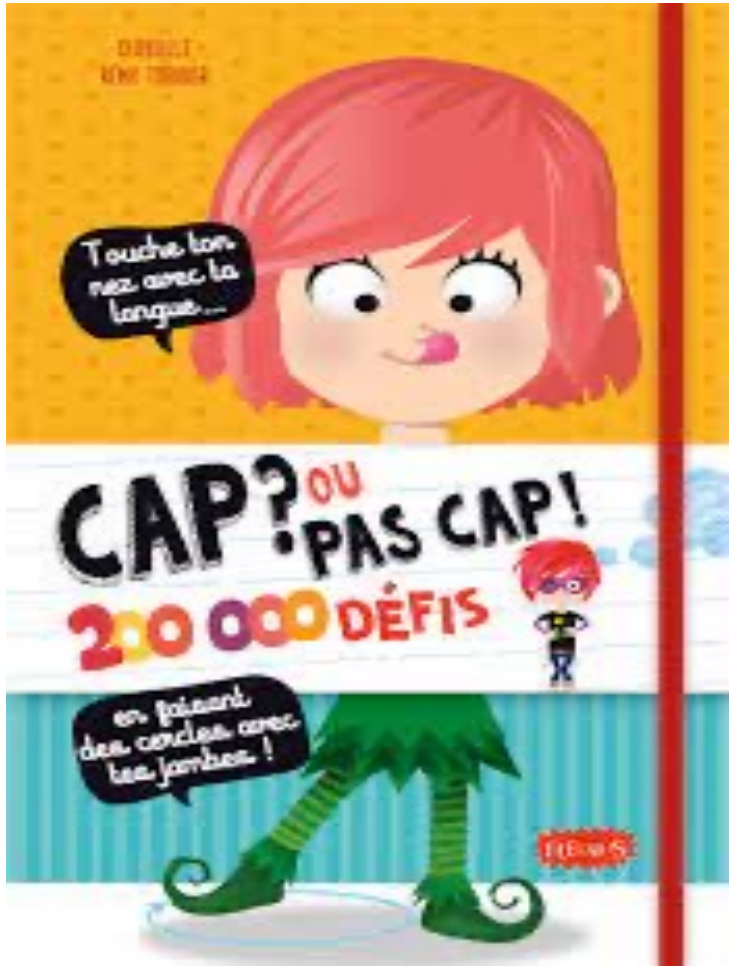
Document XML

```
<menu id="file" value="File">


  <popup>
    <menuitem value="New"
onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

TP : ReplicaSet avec MongoDB

- Démarrer le csontainer jupyter



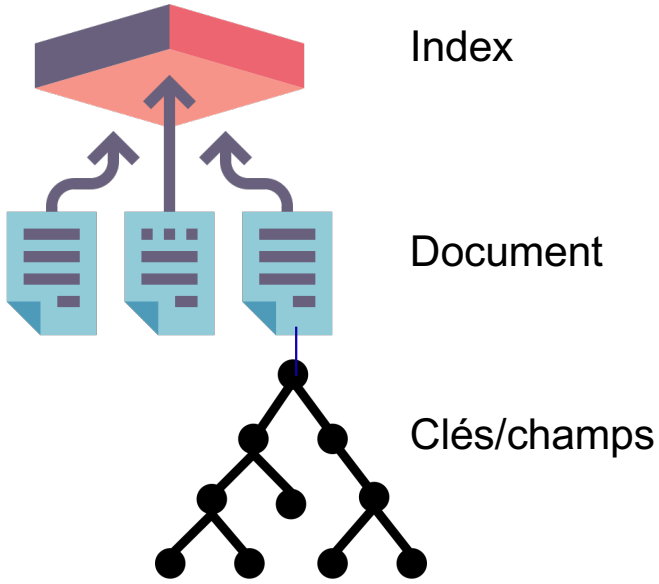
elasticsearch

OpenSearch

Qu’est ce qu’ElasticSearch ?

- Une base de données orientée Document.
- Construit sur la base du moteur de recherche Lucene : recherche textuelle avancée (multi-langue, complétion, géo-localisation)
- Une architecture distribuée
- Mécanisme de tolérance aux pannes
- Outil à l’origine OpenSource mais une tendance forte vers le privé.
- OpenSearch un fork OpenSource d’ElasticSearch
- Les utilisateuts connus :
  - CNRS
  - CERN
  - Github
  - LinkedIn

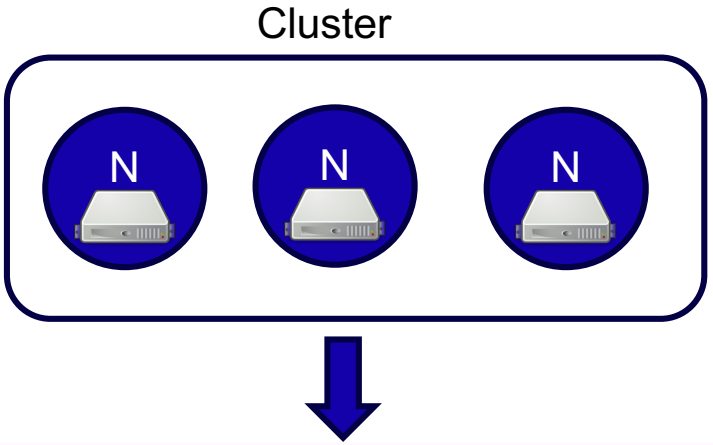
Le concept

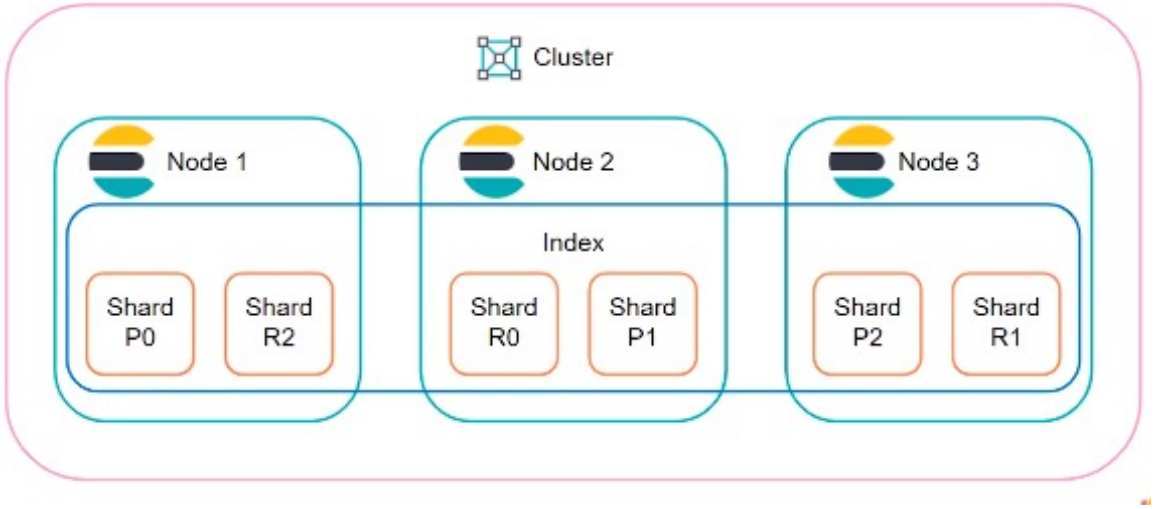


Index

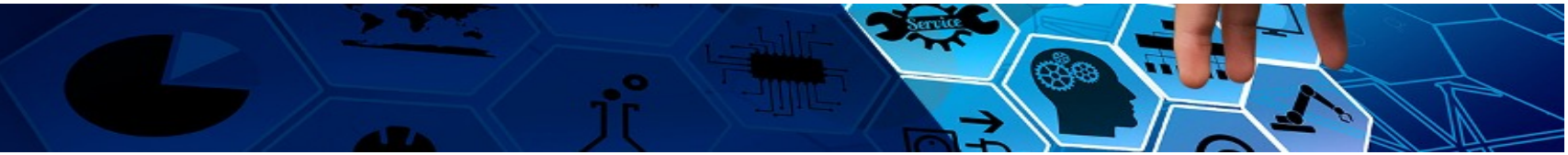
Document

Clés/champs





Le concept



- Un index peut être associé à une base de données et contient une collection de documents.
- Un document est un ensemble de clé/valeur.
- Une clé peut être de type scalaire ou complexe.
- Le mapping peut être assimilé à la définition d'un schema. Chaque index a un mapping lequel définit le type des clés. Le mapping peut être défini manuellement ou dynamiquement lorsqu'un document est indexé.
- Un shard est une partition de notre index. Un index se compose d'au moins un shard. Il existe 2 sortes de shard :
  - Les shards primaires, les écritures sont réalisées sur les shards primaires.
  - Les shards secondaires/ réplicas, sont des copies des shards primaires.


C'EST QUI
LE PATRON ?
DreamWorks
BABY
BOSS