ESD

Bases de données massives

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Présentation

- Erwan KOFFI
- Data engineer @ Saagie
- Formateur Spark et Hadoop

Programme

- Présentation du concept de NoSQL
- Théorème CAP
- MongoDB
  - Architecture
  - Modèles de données (document, BSON, partitionnement)
  - CRUD
  - Agrégations
  - Performances
  - Haute disponibilité
- Cassandra
  - Architecture
  - Modèle de données (modélisation, partitionnement, consistance)
  - CQL
  - Driver Spark
- Apache Kafka
  - Kafka Streams

Présentation du concept de NoSQL

Présentation du concept de NoSQL

Rappels: ACID et bases relationnelles

- **Atomicité**
  Une transaction est effectuée dans son intégralité. Si ce n'est pas possible toute action de la transaction est supprimée.

- **Cohérence**
  Chaque transaction à partir d'un système valide produira un autre état valide de ce dernier.

- **Isolation**
  L'exécution des transactions est indépendante, le résultat doit être le même qu'elle soient exécutées en parallèle ou en série, ceci est assuré via verrous et points de synchronisation.

- **Durabilité**
  Une transaction confirmée est enregistrée et ce quelque soit les problèmes pouvant survenir sur le système.

Présentation du concept de NoSQL

Naissance du mouvement NoSQL

- **Originaire du Web**
  Les grandes entreprises du web traitant de gros volumes de données sont à l'origine de sa démocratisation

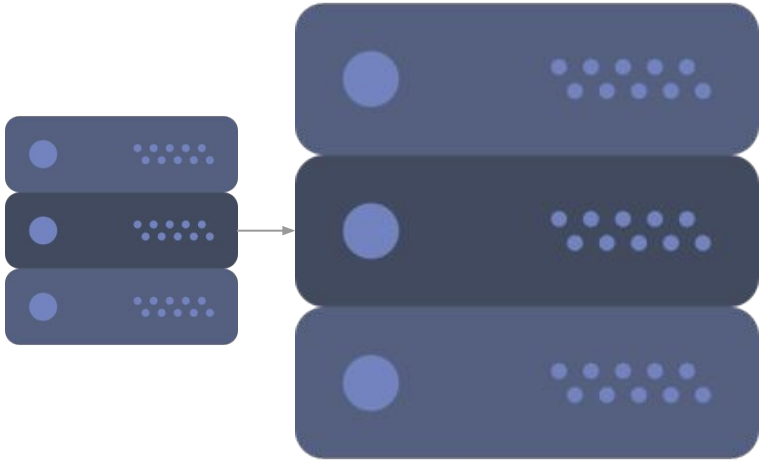
- **Manque de scalabilité des bases relationnelles**
  - Les bases relationnelles sont prévues pour ne fonctionner que sur un seul serveur (verticale).
  - Ceci est dû à leur propriété ACID.

- **Architectures distribuées**
  Ces bases de données reposent sur des architectures matérielles distribuées pour permettre la scalabilité (horizontale).

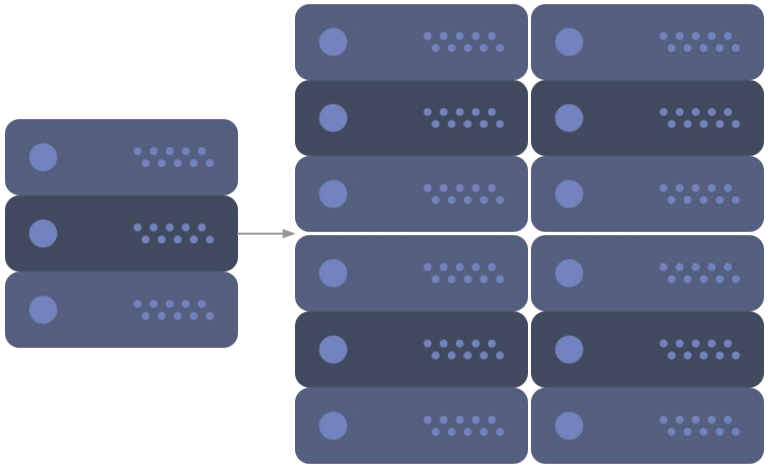
- **NoSQL = Not Only SQL ≠ No more SQL**

Présentation du concept de NoSQL

Scalabilité



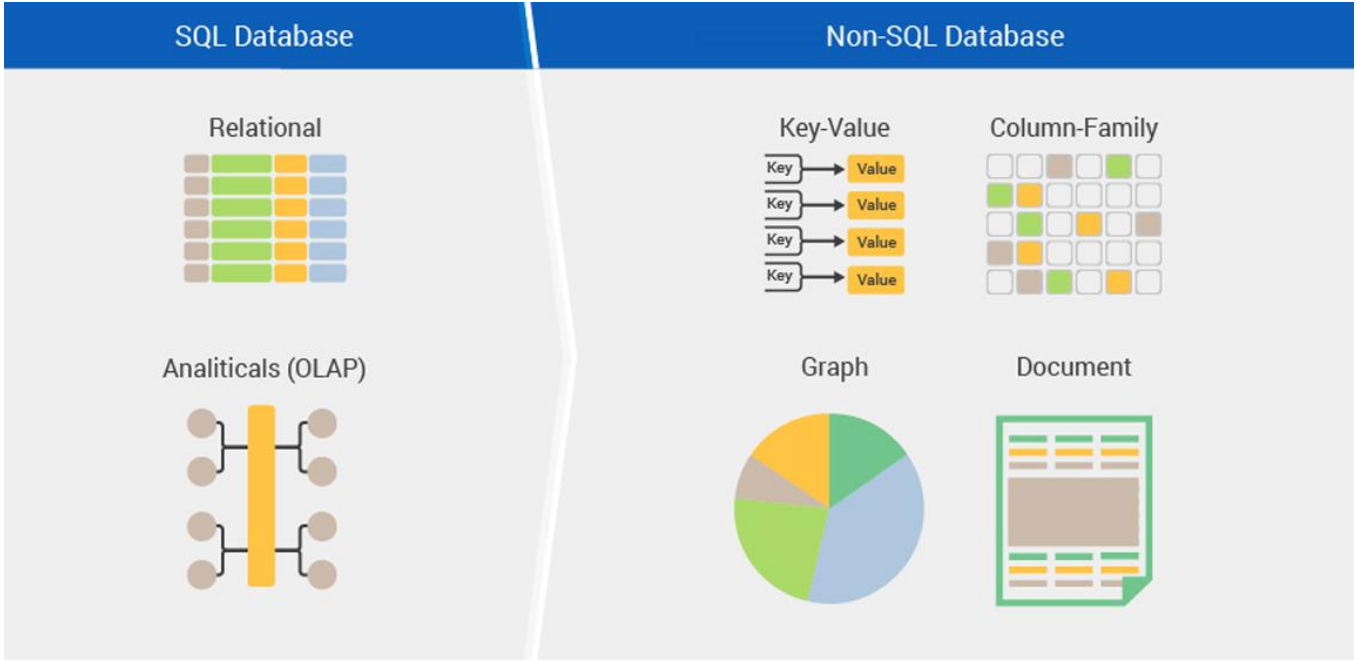
Scalabilité Verticale



Scalabilité Horizontale

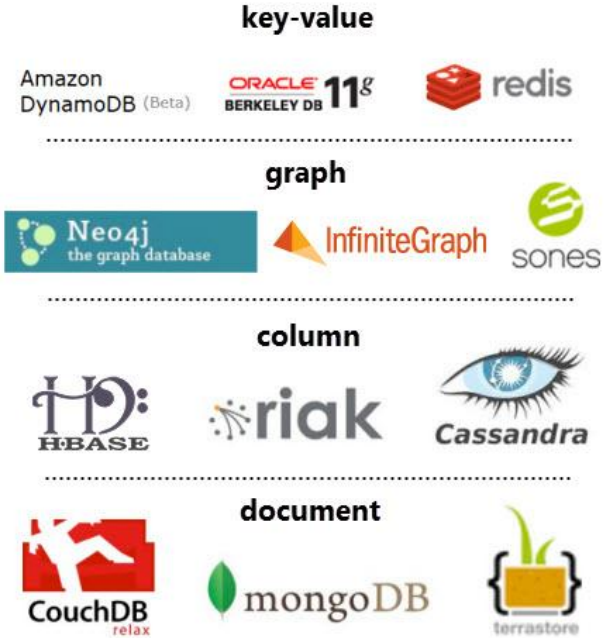
Présentation du concept de NoSQL

Types de bases de données NoSQL



Présentation du concept de NoSQL

Quelques exemples



Présentation du concept de NoSQL

Bases clé-valeur

- **Table de hachage**
  Associe une clé unique à un ensemble de valeurs. Orienté performances sur des requêtes simples.
- **Hash de la clé**
- **Plusieurs partitions par machine**
- **Cas d'usages**
  - Collecte d'évènement
  - Gestion de profils utilisateurs
  - Mise en cache de données

Présentation du concept de NoSQL

Bases orientées colonnes

- **Stocker les informations par colonnes et non plus par lignes.**
- **Les colonnes sont dynamiques et triées sur le disque.**
- **Conçues pour stocker des millions de colonnes (one-to-many).**
- **Familles de colonnes (conteneurs de colonnes).**
- **Implique la mise à jour de toutes les valeurs d'une colonne**

Présentation du concept de NoSQL

Bases orientées colonnes

Row-oriented (1)

name	age	sex	zipcode
thomas	18	male	1416
martin	33	male	1645
bob	25	male	1613

Column-oriented (2)

name	age	sex	zipcode
thomas	18	male	1416
martin	33	male	1645
bob	25	male	1613

Column Family: User

rowid	Col_name	ts	Col_value
u1	name	v1	Ricky
u1	email	v1	ricky@gmail.com
u1	email	v2	ricky@ya
u2	name	v1	Sam
u2	phone	v1	650-3456

Column Family: Social

rowid	Col_name	ts	Col_value
u1	friend	v1	u10
u1	friend	v1	u13
u2	friend	v1	u10
u2	classmate	v1	u15

- One File per Column Family
- Data inside file is physically sorted
- Sparse: NULL cell does not materialize

Présentation du concept de NoSQL

Bases orientées documents

- **Base de données clé valeur**
  Chaque clé n’est plus associée à une valeur mais à un document structuré.
- **Permet de stocker toute structure de données (one-to-one ou one-to-many).**
- **Moins de flexibilité**
  Le requêtage ou le chargement des données est plus complexe que lors de la modélisation relationnelle.

Présentation du concept de NoSQL

Bases orientées graphes

- **Stockage des données sous forme de graphe.**
- **La lecture des données s'effectue comme un parcours de graphe.**
- **Implémentation des algorithmes de parcours de graphes natifs.**
- **Cas d'usages**
  - Réseau social
  - Routage réseau

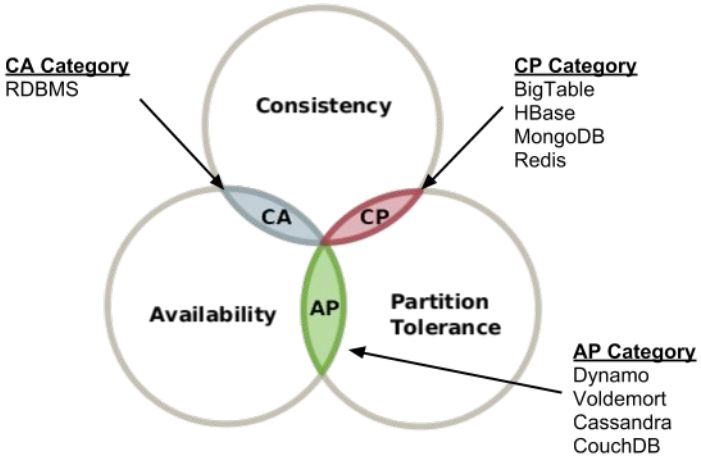
Présentation du concept de NoSQL

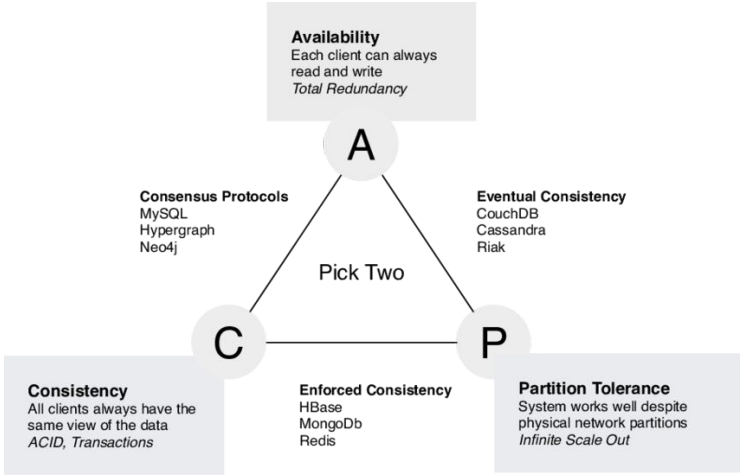
Théorème de Brewer

- Autrement connu sous le nom de théorème CAP.
- *Il est impossible sur un système informatique distribué de garantir les trois contraintes suivantes:*
  - Cohérence (**C**onsistency)
    Tous les noeuds du système ont la même donnée au même moment.
  - Disponibilité (**A**vailability)
    Toutes les requêtes reçoivent une réponse.
  - Tolérance au partitionnement (**P**artition tolerance)
    Aucune panne, hormis une coupure totale du réseau ne doit empêcher le système de répondre correctement.

Présentation du concept de NoSQL

Théorème CAP





MongoDB

MongoDB

Introduction

```
{
  "nom":"MongoDB",
  "version":"4.0.2",
  "description":"base de données NoSQL open-source orientée documents",
  "caractéristiques":[
    "haute performance",
    "haute disponibilité",
    "mise à l'échelle"
  ],
  "enregistrements":{
    "description":"un enregistrement est un document qui est composé de paires de champ/valeur",
    "valeurs":[
      "types primitifs",
      "autres documents",
      "tableaux",
      "tableaux de document"
    ]
  }
}
```

MongoDB

Moteurs

- **WiredTiger**
  - Moteur par défaut.
  - Concurrence au niveau document.
  - Checkpointing, Journaling.
  - Compression.
  - Meilleure utilisation du hardware.
- **NMAPv1**
  - Ancien moteur.
  - Concurrence au niveau de la collection.
  - Journaling.
- **In-Memory Storage Engine**
  - Seulement en version entreprise.

MongoDB

Documents

```
{
  "titre":"Pourquoi des documents?",
  "raisons":[
    "correspondent aux types natifs de nombreux langages de programmation",
    "réduction des coûts de jointure grâce aux documents imbriqués et aux tableaux",
    "supporte le polymorphisme facilement"
  ],
  "stockage":{
    "format":"BSON documents",
    "description":"Le BSON est une représentation binaire du JSON avec des types en plus (date et données binaires)",
    "types":[
      "string", "integer(32-or-64-bit)", "double", "date",
      "byte array(binary data)","boolean", "null", "BSON object", "BSON array"
    ],
    "caractéristiques":["léger", "traversable facilement", "efficace"]
  }
}
```

MongoDB

Caractéristique d'un document

- **Taille maximale de 16 MB**
  - GridFS stocke les documents en plusieurs parties et permet de dépasser cette limite.
- **Index unique sur le champ "_id"**
  - Par défaut pour BSON:
    - 12 octets
    - 4 octets ➜ timestamp unix
    - 3 octets ➜ identifiant machine
    - 2 octets ➜ id de processus
    - 3 octets ➜ compteur
  - Premier champ d'un document.
  - Peut être tout type de données à l'exception d'un tableau.
- **"." permet d'accéder à un élément d'un tableau ou d'un document.**
- **Ref et DBRefs.**
  - Ref: référence à un document par son **_id.**
  - DBREFS: référence à un document par **_id**, **collection** et éventuellement sa **base de données.**

MongoDB

Modélisation orientée document

- **Schéma flexible**
  - Pas besoin de définir un schéma pour insérer des données.
- **Facilite le mapping document ↔ entité**
- **Chaque document peut avoir sa version d'un entité**
- **Denormalized documents**
  - Aussi appelé embedded documents
  - Permet de manipuler les données en relation lors de la même opération sur une base de données
- **Normalized documents**
  - Les relations sont stockées au sein des documents

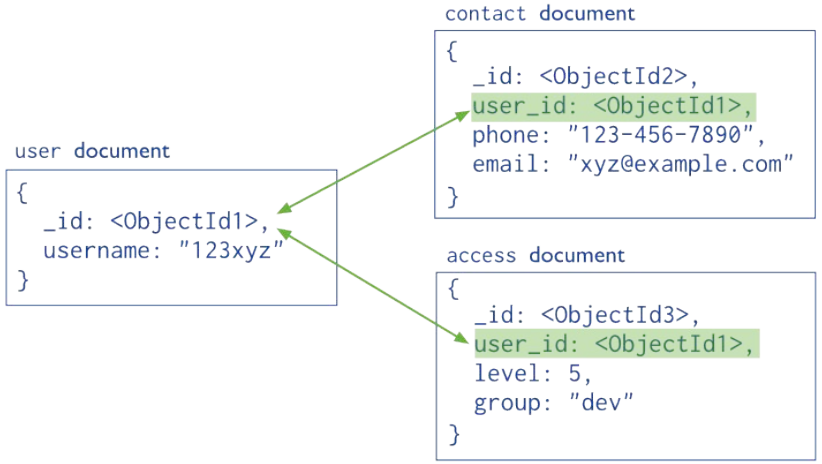
MongoDB

Modélisation: différences

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document



MongoDB

Modélisation: Document dénormalisé

- **Atomicité des écritures au niveau du document.**
- **Permet d’éviter plusieurs opérations d’écritures qui ne sont pas collectivement atomique.**
- **Meilleures performances en lecture pour les relations one-to-one et one-to-many.**
- **MMAPv1: Déplacement du document en cas du dépassement de la taille qui lui est allouée.**

MongoDB

Modélisation: Document normalisé

- **Coût de duplication des données.**
- **Représentation des relations many-to-many.**
- **Pour modéliser des ensembles de données hiérarchiques.**
- **Plus de souplesse de manipulation.**

MongoDB

Modélisation: Facteurs opérationnels

- **Croissances du document.**
  - MongoDB utilise une stratégie d'allocation en puissance de 2 pour limiter le nombre de déplacements de fichiers sur le disque.
- **Atomicité des opérations d'écriture.**
  - Une opération d'écriture unique ne peut modifier plus d'un document.
  - Si votre application peut tolérer les mises à jour non atomiques pour deux entités, elles peuvent être stockées dans des documents distincts.
- **Sharding**
  - Permet de distribuer les documents de manière horizontale.
  - Utilise une clé de sharding pour la distribution des documents.
- **Index**
  - Améliorer les performances des opérations de lectures
  - A un impact négatif sur les opérations d'écriture

MongoDB

Modélisation: Structure d’arbre

- **Référence sur le parent**
  - Solution simple mais nécessite plusieurs requêtes pour parcourir les sous arbres.
- **Référence sur les enfants**
  - Valable si aucune opération sur les sous arbres n’est nécessaire.
  - Utile lorsqu’un noeud peut avoir plusieurs parents.
- **Tableau des ancêtres et référence sur le parent**
  - Rapide pour trouver les ancêtres et les enfants d’un noeud en créant un index sur les éléments du champ des ancêtres.
- **Arbre binaire**
  - Rapide pour trouver tous les sous arbres mais inefficace pour modifier sa structure.

MongoDB

Validation des documents

- **validationLevel**
  - "strict" Appliqué pour tous les insert/update.
  - "moderate" Appliqué pour les inserts et update de documents existants valide, ignoré pour les autres.
- **validationAction**
  - "error" Rejette toute insertion ou mise à jour qui viole les règles de validation.
  - "warn" Les violations des règles sont sauvegardés mais les insert et update sont quand même effectués.

MongoDB

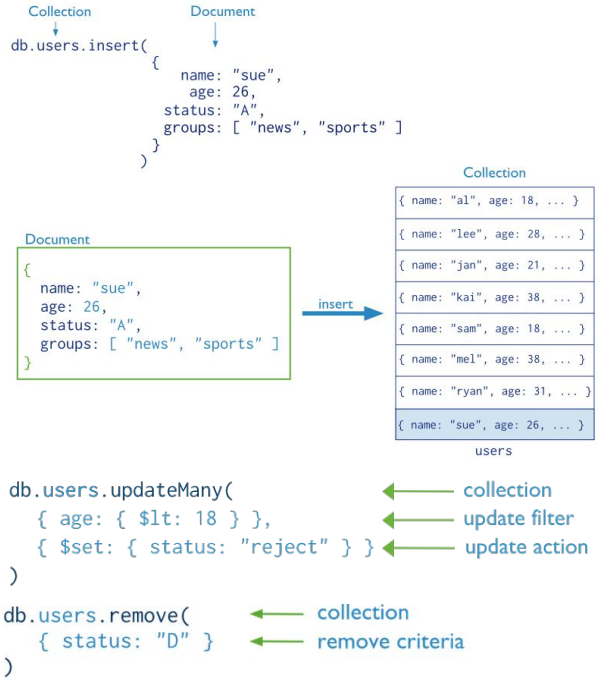
Validation des documents: Exemple

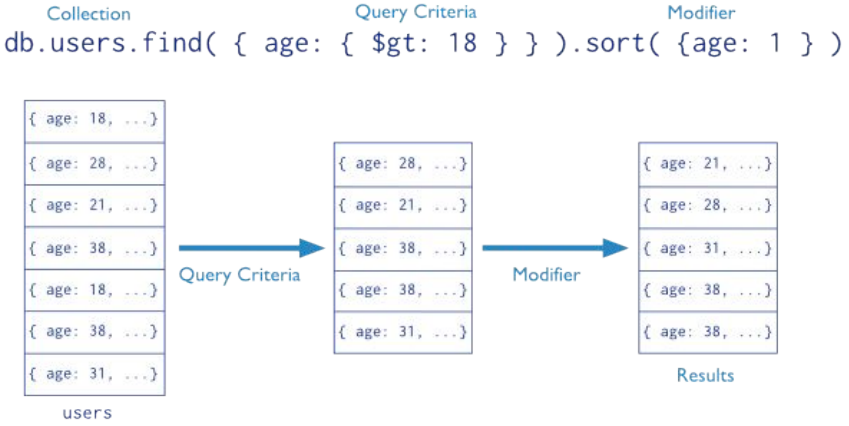
```
db.createCollection("contacts",
  { validator: { $or:
    [
      { phone: { $type: "string" } },
      { email: { $regex: /@mongodb\.com$/ } },
      { status: { $in: [ "Unknown", "Incomplete" ] }
}
    ]
  }
} )
```

```
db.runCommand( {
  collMod: "contacts",
  validator: {
    $or: [
      { phone: { $exists: true } },
      { email: { $exists: true } }
    ]
   },
  validationLevel: "moderate"
} )
```

MongoDB

Opérations de CRUD





MongoDB

Opérations: Collections

- **Les collections regroupent un ensemble de documents**
- **Elles sont créées implicitement lorsqu'elle sont référencées pour la première fois.**
- **db.createCollection() permet de spécifier des options particulières.**

MongoDB

Opérations: Écriture

- **Le champ "_id" est obligatoire et unique dans tous les documents.**
- **L’insertion peut se faire via plusieurs méthodes.**
  - db.collection.insertOne()
  - db.collection.insertMany()
  - db.collection.insert()

MongoDB

Opérations: Update

- **La mise à jour permet l’upsert.**
  - db.collection.updateOne()
  - db.collection.replaceOne()
  - db.collection.updateMany()
  - db.collection.update()
  - db.collection.findOneAndUpdate()

MongoDB

Opérations: Delete

- **Exemple de fonction de suppression.**
  - db.collection.deleteOne()
  - db.collection.deleteMany()
  - db.collection.remove()
  - db.collection.findOneAndDelete()

MongoDB

Opérations: Bulk write

- **Si elle est ordonnée s'arrête en cas d'erreur sur une opération.**
- **Les exécutions non ordonnées s'exécutent en parallèle.**
- **Utile pour les écritures de masse sur les collections shardées.**

```
db.collection.bulkWrite(
  [
    { insertOne : { "document" : { name : "sue", age : 26 } } },
    { insertOne : { "document" : { name : "joe", age : 24 } } },
    { insertOne : { "document" : { name : "ann", age : 25 } } },
    { insertOne : { "document" : { name : "bob", age : 27 } } },
    { updateMany: {
      "filter" : { age : { $gt : 25} },
      "update" : { $set : { "status" : "enrolled" } }
      }
    },
    { deleteMany : { "filter" : { "status" : { $exists : true } } } }
  ]
)
```

MongoDB

Atomicité et transactions

- **Les opérations d'écriture sont atomique au niveau d'un document unique. Même si l'opération modifie plusieurs documents imbriqués.**
- **La modification de plusieurs documents est non atomique et d'autres opérations peuvent s'entrelacer.**
- **$isolated**
  - Lock exclusif de la collection durant toute l'opération.
  - Ne fonctionne pas sur les clusters shardés.
- **Le Two-Phases commit ne permet de garantir que la consistance des données mais peut renvoyer des données intermédiaires.**

MongoDB

CRUD: Manipulation **robo3t** **asi-tdm.insa-rouen.fr**

- **Créer un document contenant un champ "**nom**" libre et un champ "**age**" valant 25.**
  - db.<collection>.insertOne({"nom": "Jean", "age": 25 })
- **Afficher la liste de tous les documents dans la collection.**
  - db.<collection>.find({})
- **Mettre à jour le champ "**age**" en lui ajoutant une année.**
  - db.<collection>.updateOne({"nom": "Jean"}, {$inc: {"age", 1}})
- **Mettre à jour le champ "**age**" pour que sa valeur soit 666.**
  - db.<collection>.updateOne({"nom": "Jean"}, {$set: {"age": 666}})
- **Créer un document contenant un champ "**nom**" libre et un champ "**age**" valant 42.**
- **Afficher uniquement le document ayant un "**age**" de 42.**
  - db.<collection>.find({ age: {$eq: 42}})

MongoDB

Opérateurs de mise à jour des champs

$inc	Incrémente un champ de la valeur spécifiée.
$mul	Multiplie la valeur du champ par la valeur spécifiée.
$rename	Renomme un champ.
$set	Défini la valeur d'un champ dans le document.
$unset	Supprime le champ.
$min	Met à jour le champ si la valeur spécifiée est inférieure à celle du champ.
$max	Met à jour le champ si la valeur spécifiée est supérieure à celle du champ.
$currentDate	Permet d'assigner la date actuelle au champ de type date ou timestamp.

MongoDB

Opérateurs de mise à jour des tableaux

$	Permet de mettre à jour le premier élément du tableau qui correspond à la condition.
$addToSet	Ajoute des élément dans un tableau s'ils n'existent pas.
$pop	Supprime le premier ou dernier élément du tableau.
$pullAll	Supprime tous les élément d'un tableau à partir d'une liste de valeur.
$pull	Supprime tous les éléments d'un tableau qui correspondent à une requête donnée.
$pushAll	Ajoute plusieurs documents dans un tableau (dépréciée).
$push	Ajoute un élément dans un tableau.

MongoDB

Modificateur

$each	Modifie le $push et le $addToSet pour ajouter plusieurs éléments dans un tableau.
$slice	Limite le nombre d'éléments dans un tableau. S'utilise avec un $each.
$sort	Ordonne les élément d'un tableau lors d'un push.
$position	Permet d'indiquer à quel emplacement du tableau la valeur doit être insérée.

```
db.students.update( { _id: 5 },
  { $push: {
     quizzes: {
       $each: [ { wk: 5, score: 8 }, { wk: 6, score: 7 }, { wk: 7, score: 6 } ],
       $sort: { score: -1 },
       $slice: 3
     }
   }
  })
```

MongoDB

Opérations: Lecture

- **Sur une collection unique.**
- **Application d'un modifier pour imposer des limit, skip, sort...**
- **L'ordre des documents n'est pas défini.**
- db.collections.find() **Méthode qui accepte des critères et des projections.**
- db.collections.findOne() **Retourne un seul document.**

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

MongoDB

Opérations: Lecture et projection

- **Une projection permet de définir les champs à retourner ou à exclure.**
- **Il est possible en utilisant les deux types de projections d'exclure le champ "_id".**

```
db.records.find( {"user_id": { $lt: 42 }} )

db.records.find( {"user_id": { $lt: 42 }}, { "age": 0 } )

db.records.find( {"user_id": { $lt: 42 }}, { "name": 1 }, { "age": 1 } )

db.records.find( {"user_id": { $lt: 42 }}, { "_id": 0 }, { "name": 1 }, { "age": 1 } )
```

MongoDB

Opérateurs de comparaison

$eq	Les documents dont le champ est égal à la valeur.
$ne	Les documents dont le champ est différent de la valeur.
$gt	Les documents dont un champ est supérieur à la valeur.
$gte	Les documents dont un champ est supérieur ou égal à la valeur.
$lt	Les documents dont un champ est inférieur à la valeur.
$lte	Les documents dont un champ est inférieur ou égal à la valeur.
$in	Les documents dont un champ est contenu dans un tableau de valeurs.
$nin	Les documents dont un champ n’est pas contenu dans un tableau de valeurs ou qu’il est inexistant.

MongoDB

Opérateurs logiques

$or	Permet de joindre deux clauses afin de récupérer les documents correspondant à l'une d'entre elles.
$and	Permet de joindre deux clauses afin de récupérer les documents correspondant au deux.
$not	Retourne les documents ne correspondant pas à la clause.
$nor	Retourne tous les documents dont les clauses ne correspondent pas.

```
db.inventory.find( { $nor: [
  { price: 1.99 }, { price: { $exists: false } },
  { sale: true }, { sale: { $exists: false } }
] } )
```

MongoDB

Opérateurs sur les éléments

$exists	Retourne les documents contenant le champ.
$type	Retourne les documents dont le champ correspond au type spécifié.

```
db.inventory.find( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
db.records.find( { b: { $exists: false } } )


db.addressBook.find( { "zipCode" : { $type : 2 } } );
db.addressBook.find( { "zipCode" : { $type : "string" } } );
```

MongoDB

Opérateurs sur les éléments

$exists	Retourne les documents contenant le champ.
$type	Retourne les documents dont le champ correspond au type spécifié.

```
db.inventory.find( { qty: { $exists: true, $nin: [ 5, 15 ] } } )
db.records.find( { b: { $exists: false } } )


db.addressBook.find( { "zipCode" : { $type : 2 } } );
db.addressBook.find( { "zipCode" : { $type : "string" } } );
```

MongoDB

Opérateurs d'évaluation

$mod	Effectue une opération de modulo sur un champ et retourne les documents vérifiant cette condition.
$regex	Retourne les documents dont les champs correspondent à l'expression rationnelle spécifiée.
$text	Effectue une recherche "full text".
$where	Permet de spécifier une condition en JavaScript.

```
db.inventory.find( { qty: { $mod: [ 4, 0 ] } } )
db.inventory.find( { qty: { $mod: [ 4 ] } } )

{ name: { $regex: /acme.*corp/i, $nin: [ 'acmeblahcorp' ] } }
{ name: { $regex: /acme.*corp/, $options: 'i', $nin: [ 'acmeblahcorp' ] } }
{ name: { $regex: 'acme.*corp', $options: 'i', $nin: [ 'acmeblahcorp' ] } }

db.myCollection.find( { $where: "this.credits == this.debits" } );
db.myCollection.find( { $where: "obj.credits == obj.debits" } );
```

MongoDB

Opérateurs sur les tableaux

$all	Retourne les documents dont les tableaux contiennent tous les éléments spécifiés.
$elemMatch	Retourne les documents dont les éléments correspondent à toutes les conditions spécifiées.
$size	Retourne les documents dont le tableau est de la taille spécifiée.

```
db.inventory.find( { tags: { $all: [ "appliance", "school", "book" ] } } )

db.inventory.find( {
        qty: { $all: [
                { "$elemMatch" : { size: "M", num: { $gt: 50} } },
                { "$elemMatch" : { num : 100, color: "green" } }
               ] }
    } )

db.collection.find( { field: { $size: 2 } } )
```

MongoDB

Opérateurs de projections

$	Retourne le premier élément du tableau. La condition ne doit contenir qu'un tableau.
$elemMatch	Projette le premier élément du tableau qui correspond à la condition spécifiée.
$meta	Projette le score du document attribué lors d'une recherche texte.
$slice	Limite le nombre d'éléments projetés d'un tableau.

```
db.students.find( { semester: 1, grades: { $gte: 85 } }, { "grades.$": 1 } )

db.students.find( { "grades.mean": { $gt: 70 } },  { "grades.$": 1 })

db.schools.find( { zipcode: "63109" },  { students: { $elemMatch: { school: 102 } } } );

db.collection.find( { field: value }, { array: {$slice: count } } );

db.posts.find( {}, { comments: { $slice: [ -20, 10 ] } } )
```

MongoDB

Mongo Shell: Curseur

- db.collections.find **retourne un curseur et affiche les 20 premiers documents.**
- toArray() **charge tout le curseur en RAM.**
- **MongoDB retourne les documents par batchs.**
  - 101 documents
  - 1 MB

```
var myCursor = db.inventory.find( { type: 'food' } );

while (myCursor.hasNext()) {
  printjson(myCursor.next());
}

var myCursor =  db.inventory.find( {type: 'food' } );
myCursor.forEach(printjson);
```

```
var myCursor = db.inventory.find( { type: 'food' } );
var documentArray = myCursor.toArray();
var myDocument = documentArray[3];

var myCursor = db.inventory.find();
var myFirstDocument = myCursor.hasNext() ?
myCursor.next() : null;
myCursor.objsLeftInBatch();
```

MongoDB

Optimisation des requêtes

- **Via la création d'index.**
- **Utiliser des critères d'inclusion plutôt que d'exclusion ($nin, $ne...).**
- **Les "Covered query".**
  - Tous les champs de la requête font partie d'un index.
  - Tous les champs retournés dans les résultats sont dans le même index.
- **Les "No covered query".**
  - Un des champs indexé ou retourné est un document imbriqué.
  - Un des champs indexé dans l'un des documents de la collection est un tableau.
  - Les collections shardées sauf si l'index comprend la clé de sharding.

MongoDB

Les types d’index

Single Field Index	Index sur un champ du document ou un champ d’un sous document.
Compound Index	Index sur plusieurs champs d’une collection.
Multikey Index	Index sur des champs de type Array.
Geospatial Index	Index sur des paires de coordonnées ou des objets GeoJSON.
Text Index	Index sur des champs de type String.
Hashed Index	Index sur le hash de valeurs d’un champ, principalement pour le sharding.

MongoDB

Types d'index: Single Field Index

- **Le champ "_id" est indexé.**
- **Possible sur un champ imbriqué ou un sous document.**
- **L'ordre des champs est à respecter dans la requête.**

```
db.friends.createIndex( { "name" : 1 } )
db.people.createIndex({ "address.zipcode": 1})
```

```
{"_id": ObjectId(...),
 "name": "John Doe",
 "address": {
      "street": "Main",
      "zipcode": "53511",
      "state": "WI"
      }
}
```

```
db.factories.createIndex( { metro: 1 } )
db.factories.find( { metro: { state: "NY", city: "New York" } } )
```

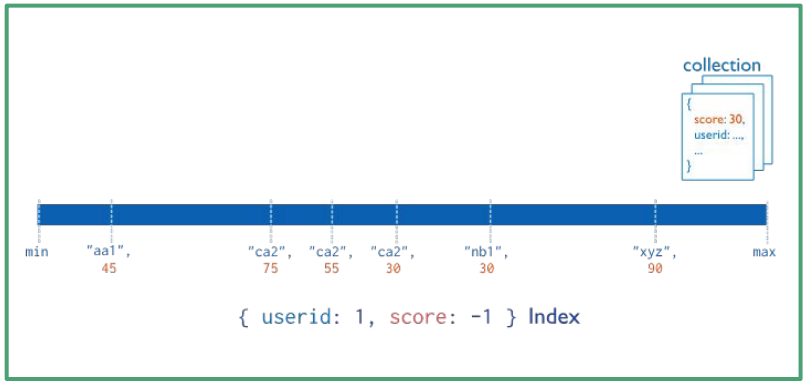
```
{
  _id: ObjectId(...),
  metro: {
        city: "New York",
        state: "NY"
       },
  name: "Giant Factory"
}
```

MongoDB

Types d'index: Compound Index

- **Limite de 31 champs par index.**
- **L'ordre des champs est important lors de la création de l'index.**
  - Lorsque l'on effectue une requête triée.
  - Pour l'utilisation de l'index lors des requêtes sur les préfixes.

```
db.products.createIndex( { "item": 1, "stock": 1 } )
db.events.createIndex( { "username" : 1, "date" : -1 } )

db.events.find().sort( { username: 1, date: -1 } )
db.events.find().sort( { username: -1, date: 1 } )
db.events.find().sort( { username: 1, date: 1 } )

{ "item": 1, "location": 1, "stock": 1 }
```



MongoDB

Types d’index: Multikey Index

- **Index sur les champs tableau.**
- **Crée une clé d’index pour chaque élément.**
- **Peut être utilisé sur des valeurs scalaires et des documents imbriqués.**
- **Un tableau au maximum si utilisé dans un "compound index".**
- **Ne supporte pas les "covered query".**
- **Clé de sharding**
  - Seulement si la clé est un préfixe d’un "compound multikey index"
  - Le préfixe ne contient pas le champ tableau.

MongoDB

Types d’index: Text Index

- **Maximum un par collection sur des types string ou tableau de string.**
- **Le score de recherche est la somme pour tous les champs indexé du nombre d'occurrences du terme recherché multiplié par son poids.**
- **Wildcard pour tout indexer.**
- **L’index est "sparse" (seulement les documents ayant le champ sont indexés).**
- **Ignore la casse et les signes diacritiques.**
- **N’indexe pas les "stop word" par défaut.**
- **Les termes des champs indexé sont tokenized et stemmed.**
- **Peut être inclu dans un compound index s’il n’y a pas d’autres clés de type multi-key ou geospatial.**

MongoDB

Types d’index: Text Index

```
db.reviews.createIndex( { comments: "text" } )

db.reviews.createIndex(
  {
    subject: "text",
    comments: "text"
  }
)
```

MongoDB

Types d'index: Hashed Index

- **Les valeurs des champs indexés sont hashées.**
- **Sur les documents imbriqué le hash de l'ensemble des champs est effectué.**
- **Pas de support pour les "Multikey Index".**
- **Supporte le sharding d'une collection.**
- **Ne supporte pas les requêtes de type interval.**

```
db.active.createIndex( { a: "hashed" } )
```

MongoDB

Types d'index: TTL Index

- **Index spécial pour indiquer un "Time To Live" sur les documents.**
- **Si le champ n'est pas une date ou un tableau de dates, le document n'expire pas.**
- **Si le champ est un tableau, la date utilisée sera la première insérée.**

```
db.eventlog.createIndex( { "lastModifiedDate": 1 }, { expireAfterSeconds: 3600 } )
```

MongoDB

Types d'index: Unique Index

- **Permet d'assurer l'unicité des valeurs d'un champ pour tous les documents de la collection.**
- **N'empêche pas un champ tableau d'avoir la plusieurs fois la même valeur.**
- **Permet d'avoir qu'un seul document dans la collection dont le champ indexé n'existe pas "null".**

```
db.collection.createIndex( { "x": 1 }, { unique: true } )
```

MongoDB

Types d'index: Partial Index

- **Index les documents par rapport à un filtre.**
  - Moins de stockage.
  - Coûts réduits en maintenance et insertion.
- **Ne sera pas utilisé sur les opérations retournant un résultat incomplet.**
- **Si il est indiqué comme unique, l'unicité ne concerne que les documents concernés par l'index.**
- **Permet de créer des index "sparse"**

```
db.restaurants.createIndex( { cuisine: 1 },
  { partialFilterExpression: { rating: { $gt: 5 } } })
```

```
db.restaurants.find({ cuisine: "Italian", rating: { $gte: 8 } })

db.restaurants.find({ cuisine: "Italian", rating: { $lt: 8 } } )

db.restaurants.find({ cuisine: "Italian" } )
```

```
db.contacts.createIndex(
  { name: 1 },
  { partialFilterExpression: { name: { $exists: true } } }
)

db.users.createIndex(
  { username: 1 },
  { unique: true, partialFilterExpression: { age: { $gte: 21 }}}
)
```

MongoDB

Types d’index: Sparse Index

- **Seuls les documents dont le champ indexé existe le sont**
- **N’est pas utilisé lors des requêtes ou des tris si le résultat est incomplet.**
- **Si il s’agit d’un "compound index" un document sera indexé si il contient l’une des clés.**
- **"Sparse" et "Unique" n’empêche pas plusieurs documents omettant la clé.**

```
db.collection.createIndex( { "field": 1 }, { sparse: true } )
```

MongoDB

Création d’index

- **En avant-plan.**
  - Bloque toutes les opérations sur la base de données et la collection est indisponible tant que l’index n’est pas totalement construit.
- **En arrière-plan.**
  - La base de données reste disponible.
  - L’index n’est utilisé que lorsque sa construction est finie.
  - Utilise une approche incrémentale, si l’index ne tient pas dans la RAM a un impact sur les performances.
  - Sera aussi lancé en background sur les noeuds secondaires.

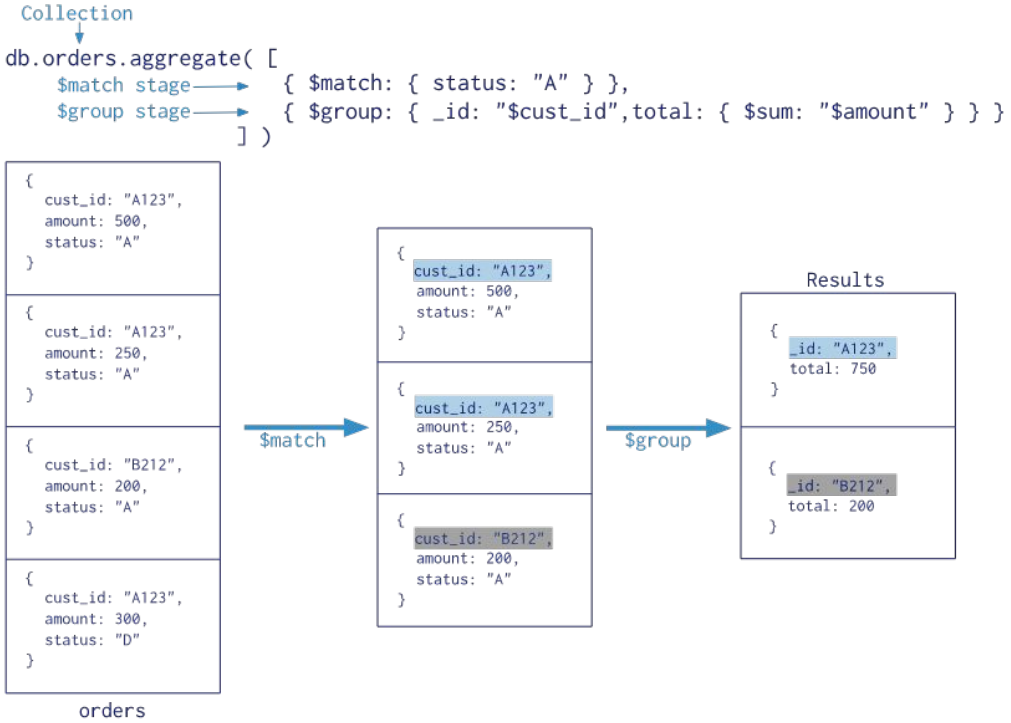
MongoDB

Impact sur les performance des opérations d’écriture

- **Les index ont un impact sur les performances d’écriture.**
- **MMAPv1.**
  - L’augmentation de la taille d’un document peut entraîner un déplacement du document sur le disque.
  - Depuis la version 3.0.0 utilise une allocation sur la base de puissance de 2.
- **Le hardware et plus particulièrement le type de disques.**

MongoDB

Agrégations



MongoDB

Agrégations: Pipeline

- **aggregate s’applique à toute une collection.**
- **Un pipeline est une suite d’étape.**
- **Chaque étape transforme les documents.**
- **Une étape peut changer la cardinalité des documents.**
- **Les expressions le peuvent pas faire références à d’autres documents.**
- **Les expressions sont stateless hormis les accumulateurs (sum, min, max...).**

MongoDB

Agrégations: Optimisation du pipeline

- **$mach et $sort peuvent utiliser les index.**
- **$match, $sort et $limit doivent être utilisés au plus tôt pour réduire la cardinalité.**
- **Des optimisations automatiques ton effectives**
  - $sort $match -> $match $sort
  - $skip(10) $limit(5) -> $limit(15) $skip(10)
  - $project $skip $limit -> $skip $limit $project
  - coalescence $sort $limit quand $sort avant le $limit
  - coalescence $lookup $unwind
  - coalescence $limit $limit, $skip $skip, $match $match

{ $sort: { age : -1 } }, { $skip: 10 }, { $limit: 5 } -> { $sort: { age : -1 } }, { $limit: 15 } { $skip: 10 }

{ $limit: 100 }, { $skip: 5 }, { $limit: 10 }, { $skip: 2 } -> { $limit: 15 }, { $skip: 7 }

MongoDB

Agrégations: Limites

- **db.collection.aggregate() retourne un curseur mais peut être stocké dans une collection.**
- **Il y a une restriction sur la taille des résultats.**
  - Pour les curseurs la limite est la même que pour les documents BSON.
  - Si le résultat est stocké dans un document, la limite est la même que pour celle des documents BSON.
- **Un pipeline ne peut excéder 100 MB en RAM.**
- **Il est possible de contourner la limite de taille des pipeline en leur faisant utiliser le disque.**

MongoDB

Agrégations: Collections shardées

- **Si un $match se fait sur une clé de sharding, le pipeline s'exécutera sur le shard correspondant.**
- **Pour les agrégations ne pouvant se lancer sur plusieurs shards, si les opérations ne nécessitent pas l'utilisation d'un shard principal elles seront lancées sur sur un shard aléatoire.**
- **Lorsqu'un pipeline est divisé en deux partie, cela est fait de manière à ce que chaque shard effectue les étapes de manière optimisée.**

MongoDB

Agrégations: opérateurs

$project	Permet de modifier le document en entrée (ajouter/supprimer des champs...).
$match	Filtre le flux de document sans le modifier.
$redact	Cumule les opérations de $match et $project.
$limit	Retourne les n premiers documents.
$skip	Ignore les n premiers documents.

MongoDB

Agrégations: opérateurs

$unwind	Éclate un champ de type tableau.
$group	Groupe les documents suivant l'expression définie et applique un accumulateur.
$sample	Sélectionne au hasard n documents.
$sort	Trie les documents suivant la clé spécifiée.
$geonear	Retourne un flux trié de documents basé sur la distance géospatiale.
$lookup	Effectue une jointure sur une autre collection pour filtrer les documents de la première collection.

MongoDB

Agrégations: opérateurs

$out	Écrit les documents issu de l'agrégation dans une collection. Dernière étape du pipeline.
$indexStats	Retourne les statistique sur l'utilisation de chaque index pour la collection.

MongoDB

Agrégations: Ensembles

$setEquals	Retourne true si les ensembles ont les mêmes éléments.
$setIntersection	Retourne les éléments contenus dans les deux ensembles en entrée.
$setUnion	Retourne les éléments des deux ensembles en entrée.
$setDifference	Retourne les éléments présents dans le premier ensemble mais pas dans le second.
$setIsSubset	Retourne true si tous les éléments du sous ensemble sont présent dans un second.
$AnyElementTrue	Retourne true si des éléments de l'ensemble valent true.
$AllElementsTrue	Retourne true si tous les éléments de l'ensemble ne valent pas false.

MongoDB

Agrégations: Commandes

aggregate	Effectue les tâches d'agrégation.
count	Compte le nombre de documents dans une collection.
distinct	Affiche les éléments distincts d’une collection.
group	Groupe les documents dans une collection par clé spécifiée et effectue une agrégations simple.
mapReduce	Applique une agrégation map reduce pour les gros jeux de données.

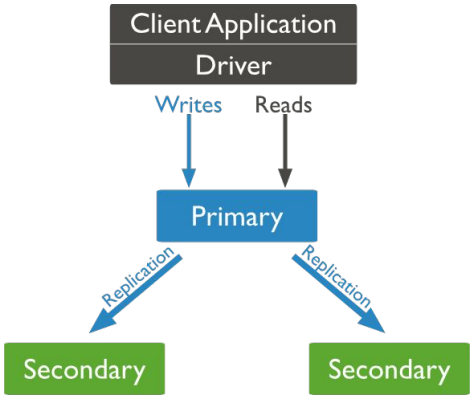
MongoDB

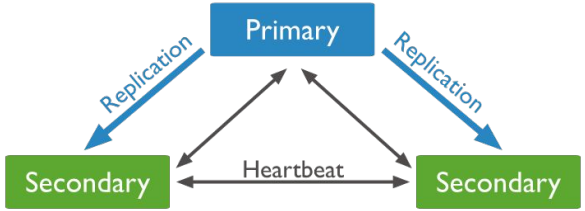
Agrégations: Variables système

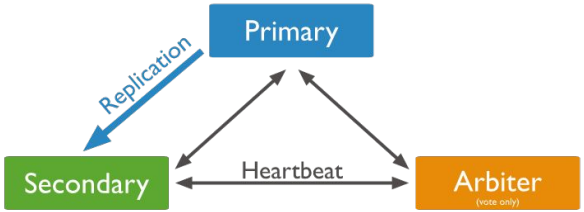
ROOT	Représente le document racine, soit le document de plus haut niveau.
CURRENT	Référence le chemin d'un champ. $champ est équivalent à $$CURRENT.champ.
DESCEND	$redact ignore les documents imbriqués. Cette variable permet de les inclure.
PRUNE	$redact ignore tous les champs du niveau courant.
KEEP	$redact garde tous les champs au niveau courant du document.

MongoDB

Réplication







MongoDB

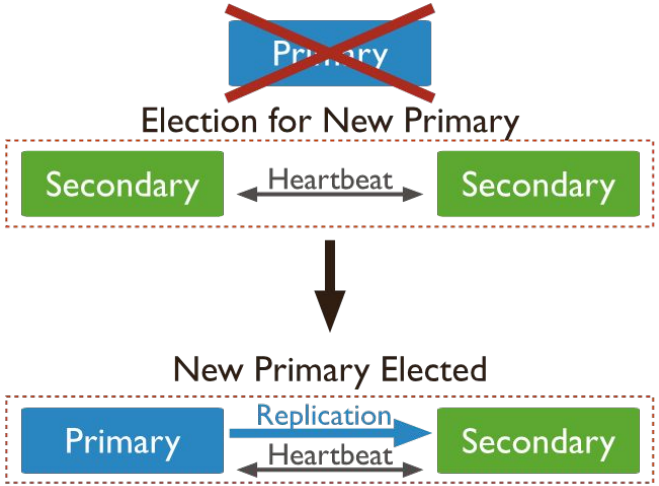
Réplication: Noeuds Secondaires

- **Conserve une copie de l'ensemble de données du noeud primaire.**
- **Réplication provenant de l'oplog du primaire pour définir ses données dans un processus asynchrone.**
- **Un client peut voir les résultats d'écriture avant qu'elles soient durables.**
- **Plusieurs configurations sont possibles**
  - Empêcher la possibilité de devenir un noeud primaire.
  - Empêcher la lecture depuis certaine applications.
  - Conserver un snapshot pour la récupération d'erreur.

MongoDB

Réplication: Reprise

Nombre de noeuds	Majorité nécessaire pour l'élection d'un nouveau noeud primaire	Tolérance à la panne
3	2	1
4	3	1
5	3	2
6	4	2

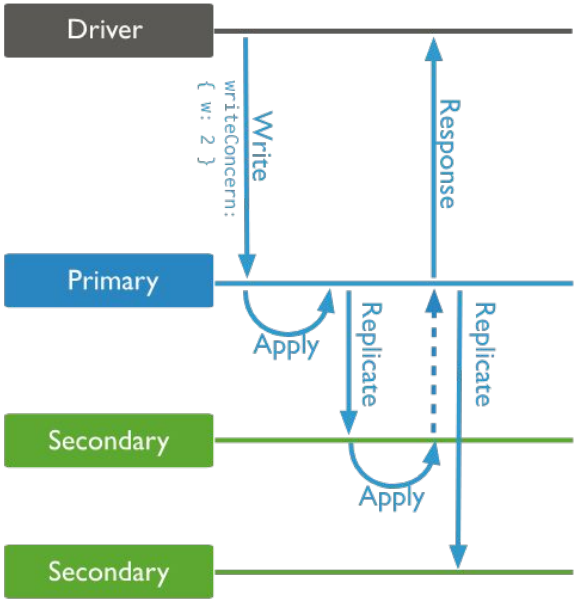


MongoDB

Replication: Write concern

- Permet de configurer la gestion des acknowledgement

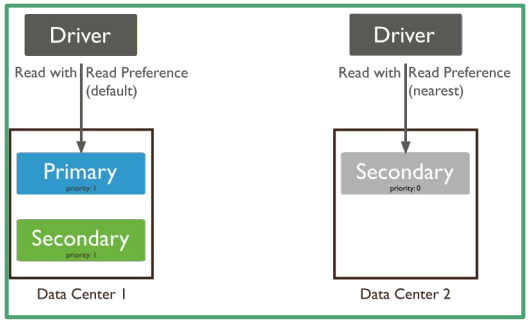
```
db.products.insert(
  { item: "envelopes", qty : 100, type: "Clasp" },
  { writeConcern: { w: 2, wtimeout: 5000 } }
)

cfg = rs.conf()
cfg.settings = {}
cfg.settings.getLastErrorDefaults = { w:"majority",
wtimeout: 5000 }
rs.reconfig(cfg)
```



MongoDB

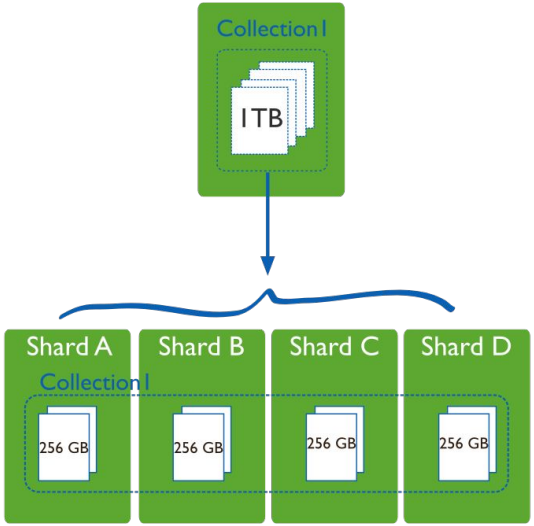
Replication: Read Concern

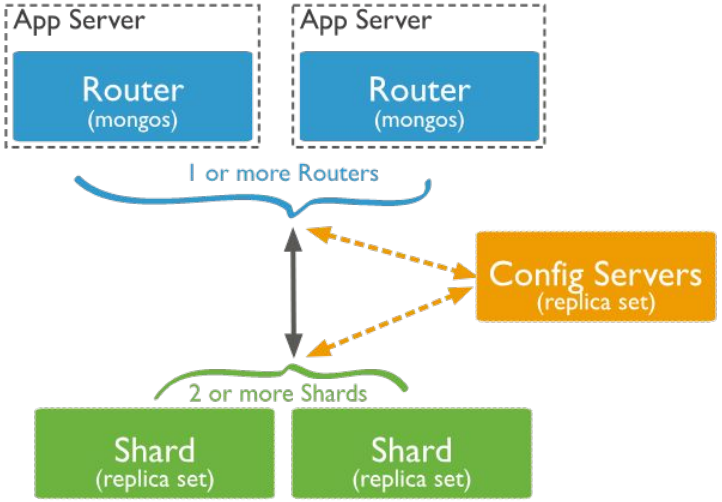
primary	Mode par défaut, les lectures sont effectuées depuis le noeud primaire.
primaryPreferred	Les opérations sont faites sur le noeud primaire s'il est disponible, sur les secondaires sinon.
secondary	Toutes les lectures sont faites sur les noeuds secondaires.
secondaryPreferred	Les opérations sont faites sur les noeuds secondaires s'il sont disponible, sur le primaire sinon.
nearest	Utilise le noeud avec le moins de latence réseau.



MongoDB

Sharding





MongoDB

Sharding

- **1 shard principal.**
- **Les config servers.**
  - Ils peuvent être déployés en replicaset (pas d'arbitres, delayed member build index).
  - Stockent les métadonnées du cluster dans la base de donnée config. Mongo met ces donnés en cache et les utilise pour router les lectures et écritures sur les shards.
- **Les données sont écrites sur les config servers lors de changements sur les métadonnées (balancing/split).**

MongoDB

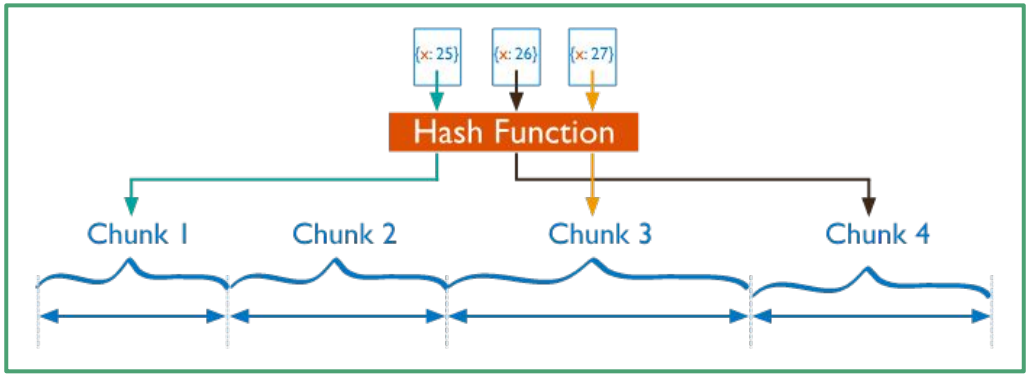
Sharding: clé de sharding

- **Soit un champ indexé ou un champ composé indexé qui existe dans chaque document de la collection.**
- **Les valeurs de clés de sharding sont divisées en chunks et distribuées de manière uniforme sur les shards.**
- **La clé de sharding**
  - Déterminer les champs les plus fréquemment inclus dans les requêtes et le plus dépendant des performances.
  - La clé doit avoir une haute cardinalité.
  - La croissance doit être non monotone (éviter le auto-increment).

MongoDB

Sharding: Hash based partitioning

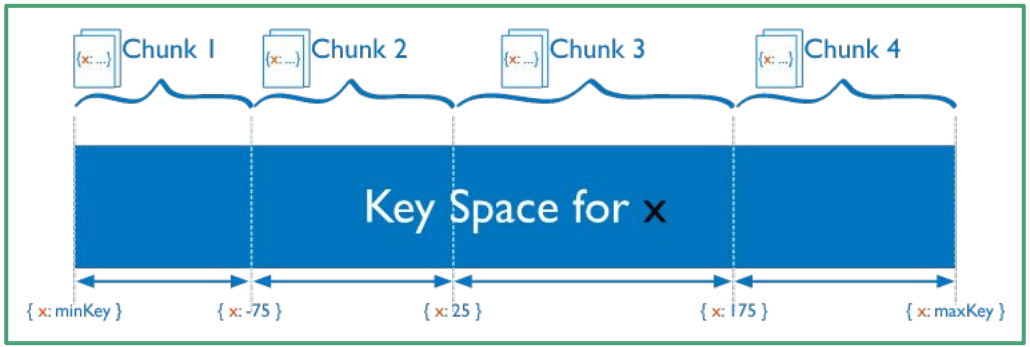
- **Calcul du hash d'une valeur d'un champ pour créer des chunks.**
- **La distribution aléatoire rend plus probable qu'une requête intervalle sur la clé de sharding ne sera pas en mesure de cibler quelques shards.**



MongoDB

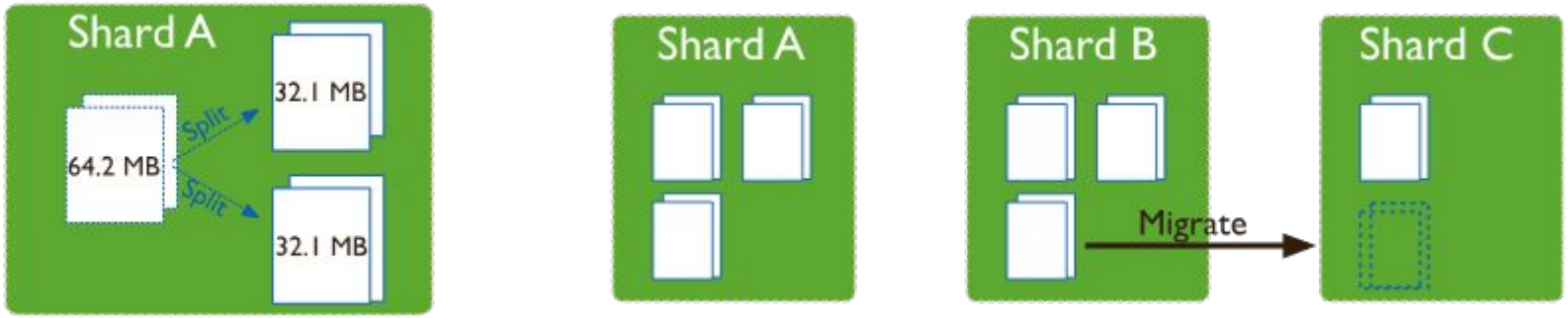
Sharding: Range based partitioning

- **Divise l'ensemble des valeurs en plages appelées "chunk" déterminé par les valeurs des clés de sharding.**
- **Chaque valeur de la clé de sharding est dans une plage.**
- **Permet de router seulement vers les shards contenant ces chunks.**
- **Peut entraîner une répartition inégale des données ce qui annule certains avantages du sharding.**



MongoDB

Split / Balancing



Cassandra

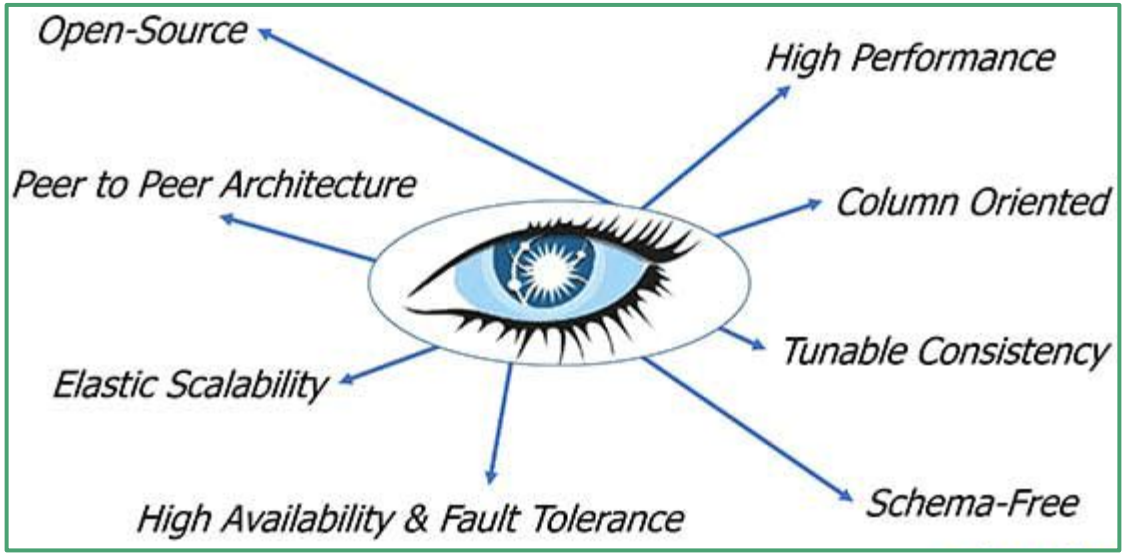
Cassandra

Histoire

- **Initialement développé par Facebook pour la gestion de son service de messagerie.**
- **Publié comme projet open-source en Juillet 2008.**
- **Devient un projet d'Apache Incubator en Mars 2009.**
- **Le 17 Février 2010 devient un projet Apache top-level.**

Cassandra

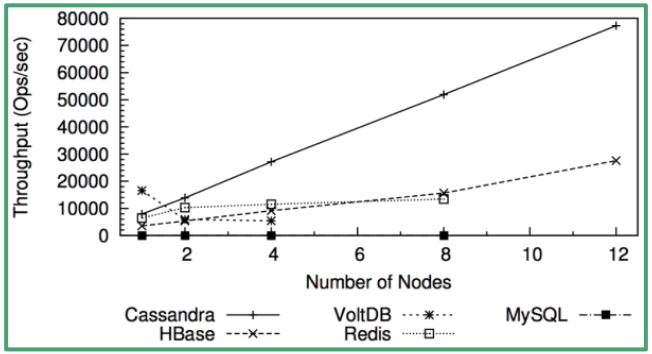
Caractéristiques



Cassandra

Introduction

- **Tous les noeuds ont le même rôle.**
- **Distribution automatique des données au travers d'un "ring".**
- **La réplication est possible sur un ou plusieurs datacenters.**
- **Fournit une scalabilité linéaire.**



Cassandra

Modèle de données

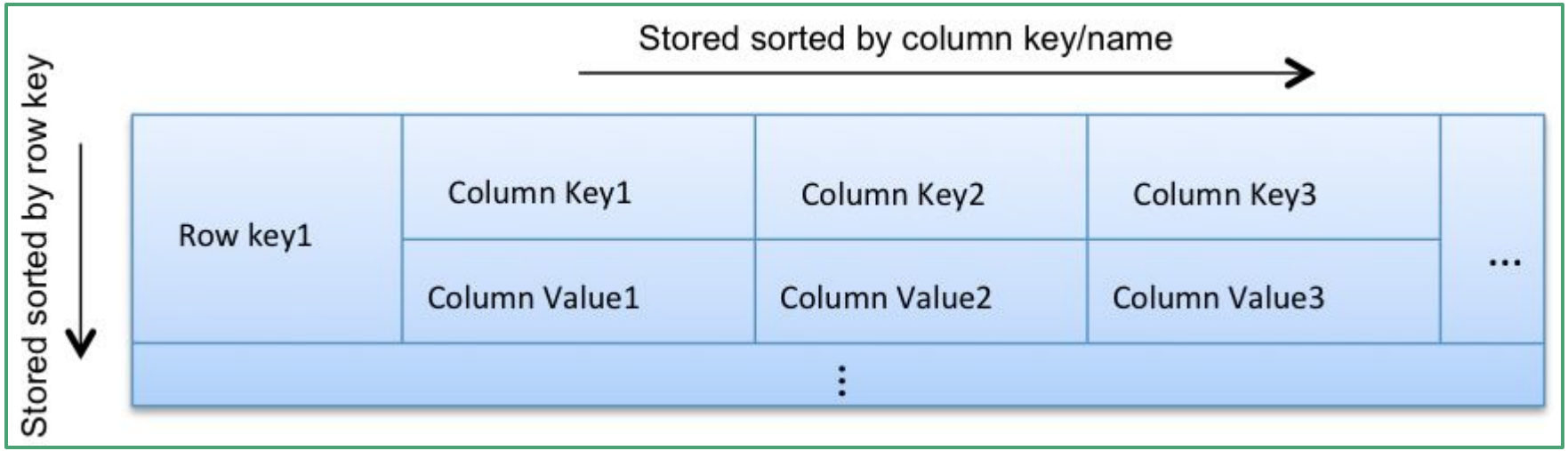
- **Schema optionnel et modèle de données orienté colonnes.**
- **Chaque enregistrement peut avoir des colonnes différentes.**
- **Un modèle de données est composé de keyspaces, famille de colonnes, de clés et de colonnes.**

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Cassandra

Modèle de données

- **Une famille de colonne équivaut à un tableau associatif trié (Map).**
- Map<RowKey, SortedMap<ColumnKey, ColumnValue>>



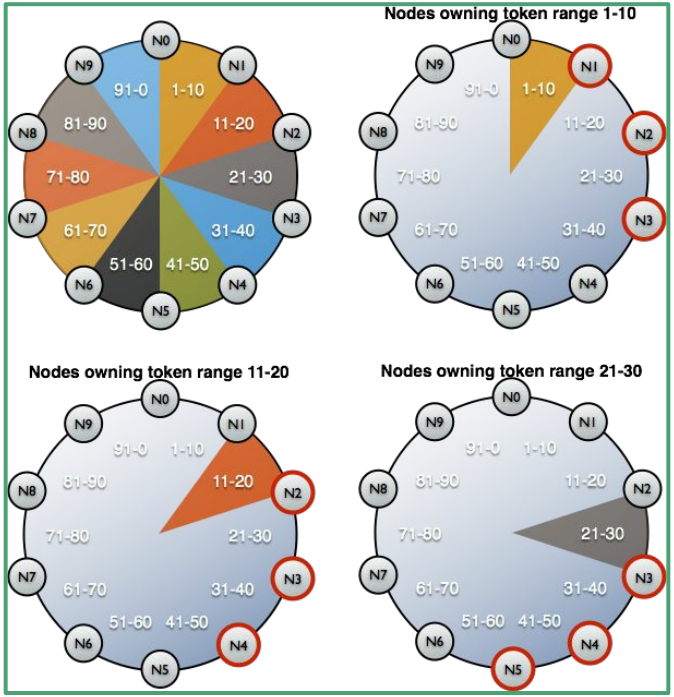
Cassandra

Clustering

- **Cluster**
  - Regroupement logique de noeuds.
  - Fait référence à tous les noeuds de tous les datacenters qui sont des pairs (communiquant entre eux).
- **Ring**
  - Est un synonyme de cluster.
- **Token rings**
  - Distribution des données dans Cassandra via le hachage de la clé de partition appelé "token".

Cassandra

Token



Cassandra

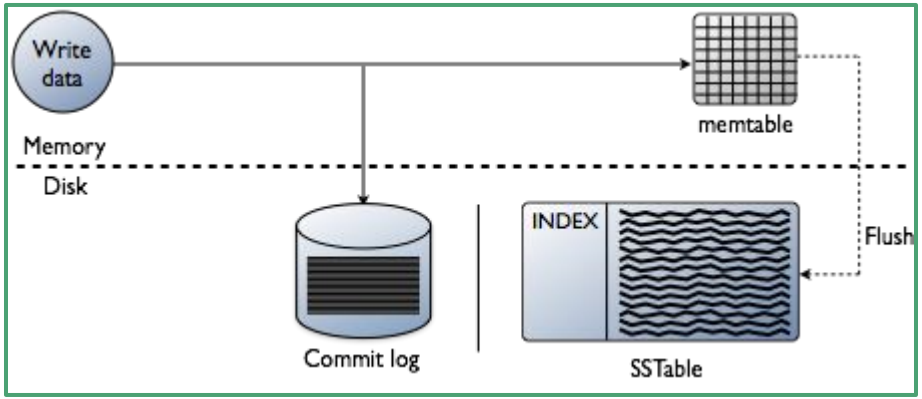
Moteur

- **Commit log**
  - Fichier où les mises à jour sont enregistrées.
  - Assure la durabilité des transactions.
- **Memtable**
  - Cache en mémoire qui contient les écritures qui n’ont pas encore été persistés dans un SSTable.
  - En général un memtable par table.
  - S’il dépasse sa taille maximale, il est flushé dans un SSTable.
- **SSTables**
  - Fichier de données immuables que Cassandra utilise pour la persistance des données sur le disque.
- **En cas de défaillance d’un noeud, les évènements sont rejoués à partir du commit log.**

Cassandra

Moteur: Écriture de la donnée

- **Cassandra trie les memtables par token puis écrit les données sur le disque séquentiellement.**
- **Un index de partition est également créé sur le disque qui associe les token à un emplacement sur le disque.**
- **Les données contenues dans le commit log sont purgée après que les données correspondantes dans le memtable ont été vidées dans un SSTable.**



Cassandra

Moteur: SSTables

- **Index de partition**
  - Liste des clés de partition et la position de départ des lignes dans le SSTable.
- **Résumé de la partition**
  - Échantillon de l’index de partition stocké dans la mémoire.
- **Bloom Filter**
  - Structure stockée dans la mémoire qui vérifie si les données d’une ligne existe dans le memtable avant d’accéder aux SSTables sur le disque.

Cassandra

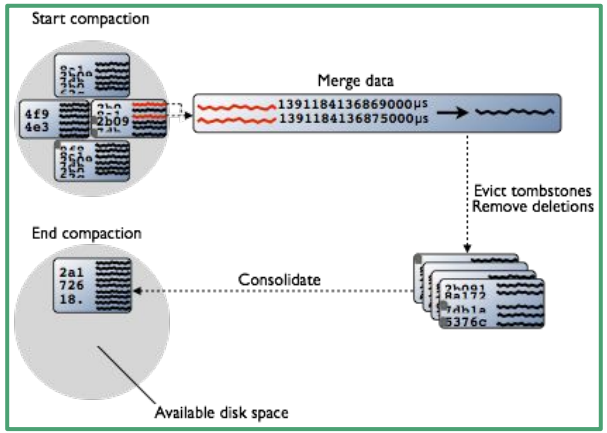
Moteur: Conservation de la donnée.

- **Cassandra conserve les données sur le disque en consolidant les SSTables.**
- **Les SSTables sont immuable et s'accumulent sur le disque.**
- **Les SSTables doivent être périodiquement fusionnées et compactées.**
- **A chaque insertion ou mise à jour, au lieu d'écraser les lignes, Cassandra écrit une nouvelle version horodatée des données dans un autre SSTable.**
- **Cassandra ne supprime pas les données (Imuabilité des SSTables) mais les marque à l'aide d'un** "tombstone"**.**
- **Au fil du temps, de nombreuses versions d'une ligne peuvent exister dans plusieurs SSTables. Chaque version a un ensemble de colonnes différent, il est donc nécessaire de lire plusieurs SSTables pour récupérer un enregistrement.**

Cassandra

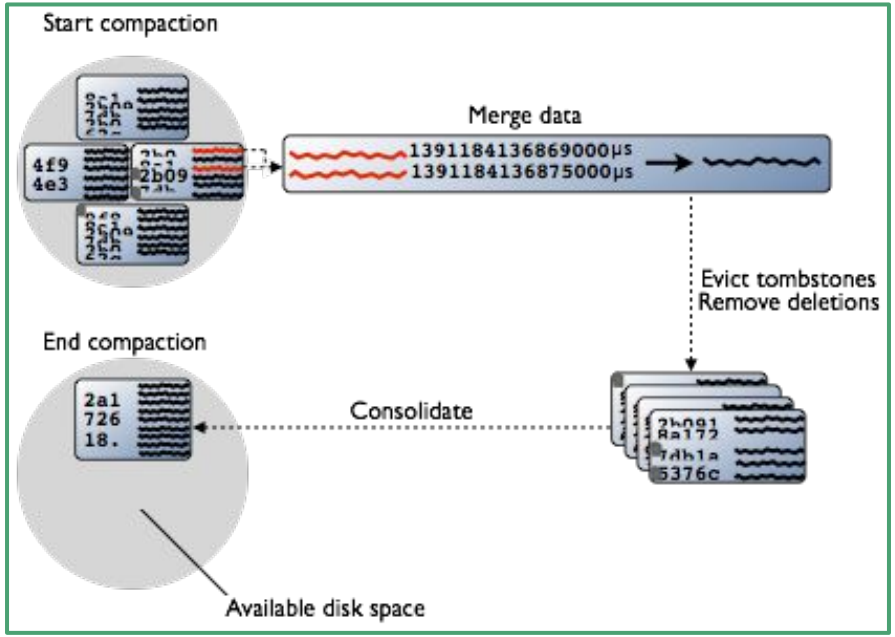
Moteur: Compaction

- **La compaction fusionne les données de chaque SSTable par clé de partition dans un nouveau fichier SSTable unique en ne sélectionnant que les données les plus récentes.**
- **Les anciens fichiers SSTables sont supprimés dès que toute lecture en attente les utilisant sont terminées.**



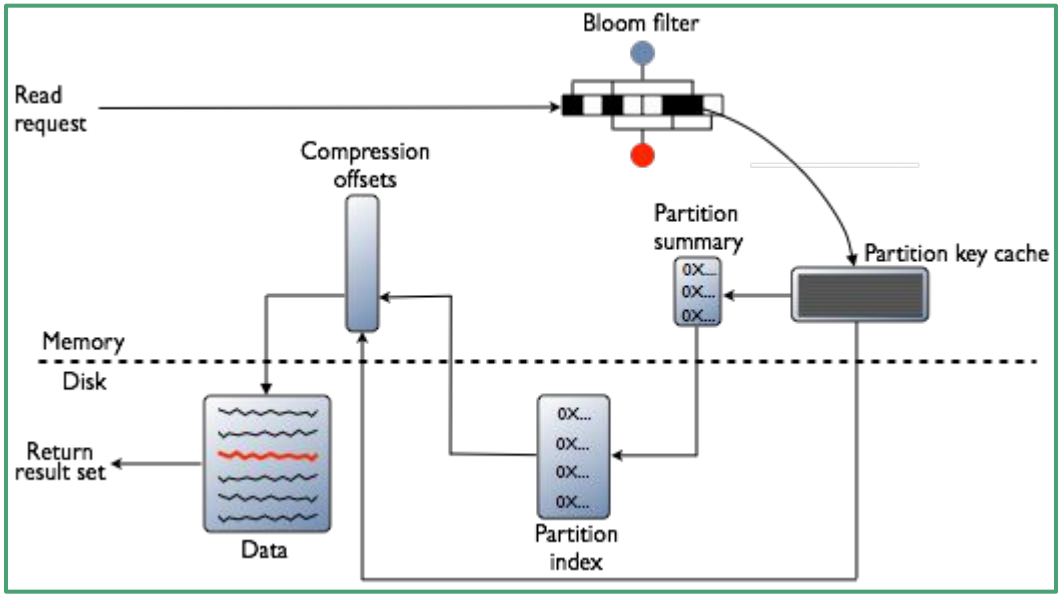
Cassandra

Compaction



Cassandra

Moteur: Lecture



Cassandra

Moteur: Lecture

- **Cassandra doit combiner les résultats de plusieurs SSTables et du memtable actif.**
- **Cassandra traite les données à plusieurs étapes pour découvrir où les données sont stockées.**
  - Memtable
  - Row cache si activé
  - Bloom filter
  - Partition key cache si activé
    - Si la clé est dans le cache
      - Accède à l'index de partition
    - Sinon
      - Vérification du résumé de partition
      - Accède à l'index de partition
  - Récupère les données sur le disque à l'aide de l'index de partition
  - Récupère les données à partir des SSTables sur le disque.

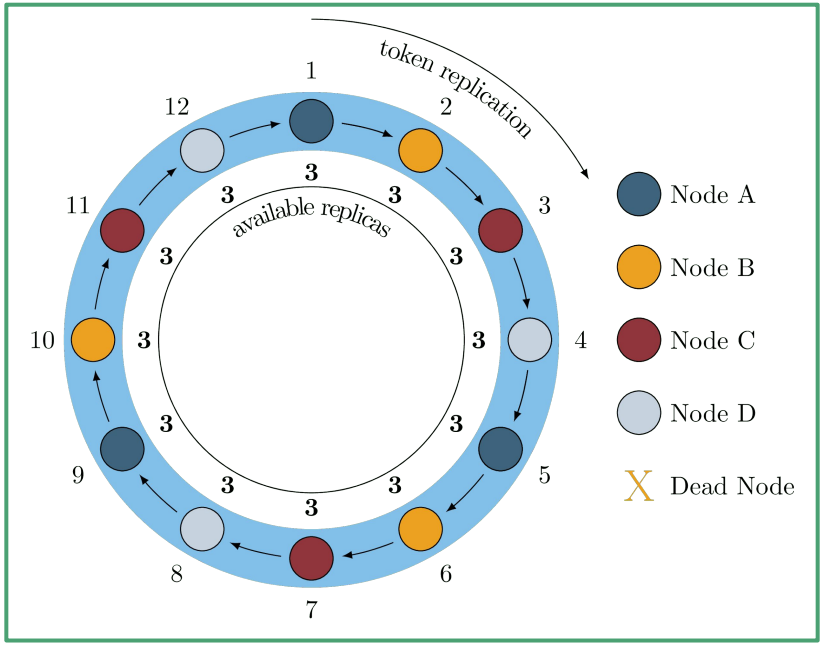
Cassandra

Réplication: SimpleStrategy

- **Permet de définir un facteur de réplication qui détermine le nombre de noeuds qui doivent contenir une copie de chaque ligne.**
- **Traite tous les noeuds de la même manière quel que soit leur datacenter ou leur rack.**
- **Pour déterminer les réplicas pour une plage de token Cassandra itère dans le token ring.**
- **Pour chaque token vérifie si le noeud propriétaire a été ajouté à l'ensemble des réplicas.**

Cassandra

Token



Cassandra

Réplication: NetworkTopologyStrategy

- **Permet de définir un facteur de réplication pour chaque datacenter dans le cluster.**
- **Tente de choisir les réplicas d'un même datacenter sur plusieurs racks.**
  - Si le nombre de racks est supérieur ou égal au facteur de réplication, chaque réplica sera sur un rack différent.
  - Sinon chaque rack contiendra un ou plusieurs réplicas.
- **Attention à la topologie du cluster.**
  - S'il n'y a pas un nombre pair de noeuds dans chaque rack, la charge de données sur le plus petit peut être beaucoup plus élevée.
  - Si un seul noeud est démarré dans un nouveau rack, il sera considéré comme le réplica de l'ensemble des données.

Cassandra

Cohérence

- **Compromis par opération**
  - ONE
  - TWO
  - THREE
  - QUORUM
  - ALL
  - LOCAL_QUORUM
  - EACH_QUORUM
  - LOCAL_ONE
  - ANY
- **Les opérations d’écriture sont envoyées à tous les réplicas, la cohérence ne défini que le nombre de réponses de validité d’écriture avant de confirmer l’écriture au client.**
- **Lors de la lecture, les opérations sont distribuées pour atteindre le niveau de cohérence.**

Cassandra

Requêtage: Types de données

Internal Type	CQL Name	Description
BytesType	blob	Arbitrary hexadecimal bytes (no validation)
AsciiType	ascii	US-ASCII character string
UTF8Type	text, varchar	UTF-8 encoded string
IntegerType	varint	Arbitrary-precision integer
Int32Type	int	4-byte integer
InetAddressType	inet	IP address string in xxx.xxx.xxx.xxx form
LongType	bigint	8-byte long
UUIDType	uuid	Type 1 or type 4 UUID
TimeUUIDType	timeuuid	Type 1 UUID only (CQL3)
DateType	timestamp	Date plus time, encoded as 8 bytes since epoch
BooleanType	boolean	true or false
FloatType	float	4-byte floating point
DoubleType	double	8-byte floating point
DecimalType	decimal	Variable-precision decimal
CounterColumnType	counter	Distributed counter value (8-byte long)

Cassandra

CQL: Type de données Map

```
CREATE TABLE users (
  id text PRIMARY KEY,
  name text,
  favs map<text, text> // A map of text keys, and text values
);

INSERT INTO users (id, name, favs)
      VALUES ('jsmith', 'John Smith', { 'fruit' : 'Apple', 'band' : 'Beatles' });

// Replace the existing map entirely.
UPDATE users SET favs = { 'fruit' : 'Banana' } WHERE id = 'jsmith';

UPDATE users SET favs['author'] = 'Ed Poe' WHERE id = 'jsmith';
UPDATE users SET favs = favs + { 'movie' : 'Cassablanca', 'band' : 'ZZ Top' } WHERE id = 'jsmith';

DELETE favs['author'] FROM users WHERE id = 'jsmith';
UPDATE users SET favs = favs - { 'movie', 'band'} WHERE id = 'jsmith';
```

Cassandra

CQL: Type de données Set

```
CREATE TABLE images (
   name text PRIMARY KEY,
   owner text,
   tags set<text> // A set of text values
);

INSERT INTO images (name, owner, tags)
        VALUES ('cat.jpg', 'jsmith', { 'pet', 'cute' });

// Replace the existing set entirely
UPDATE images SET tags = { 'kitten', 'cat', 'lol' } WHERE id = 'jsmith';

UPDATE images SET tags = tags + { 'gray', 'cuddly' } WHERE name = 'cat.jpg';

UPDATE images SET tags = tags - { 'cat' } WHERE name = 'cat.jpg';
```

Cassandra

CQL: Type de données Set

```
CREATE TABLE plays ( id text PRIMARY KEY,
  game text,
  players int,
  scores list<int>)

INSERT INTO plays (id, game, players, scores) VALUES ('123-afde', 'quake', 3, [17, 4, 2]);
// Replace the existing list entirely
UPDATE plays SET scores = [ 3, 9, 4] WHERE id = '123-afde';

UPDATE plays SET players = 5, scores = scores + [ 14, 21 ] WHERE id = '123-afde';
UPDATE plays SET players = 6, scores = [ 3 ] + scores WHERE id = '123-afde';

UPDATE plays SET scores[1] = 7 WHERE id = '123-afde';

DELETE scores[1] FROM plays WHERE id = '123-afde';
UPDATE plays SET scores = scores - [ 12, 21 ] WHERE id = '123-afde';
```

Cassandra

CQL: Keyspace

```
CREATE KEYSPACE Excelsior
    WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};

CREATE KEYSPACE Excalibur
    WITH replication = {'class': 'NetworkTopologyStrategy', 'DC1' : 1, 'DC2' : 3}
     AND durable_writes = false;
```

Cassandra

CQL: Table

```
CREATE TABLE monkeySpecies (
   species text PRIMARY KEY,
   common_name text,
   population varint,
   average_size int
) WITH comment='Important biological records' AND read_repair_chance = 1.0;

CREATE TABLE timeline (
   userid uuid,
   posted_month int,
   posted_time uuid,
   body text,
   posted_by text,
   PRIMARY KEY (userid, posted_month, posted_time)
) WITH compaction = { 'class' : 'LeveledCompactionStrategy' };
```

```
CREATE TABLE loads (
   machine inet,
   cpu int,
   mtime timeuuid,
   load float,
   PRIMARY KEY ((machine, cpu), mtime)
) WITH CLUSTERING ORDER BY (mtime DESC);
```

Cassandra

CQL: Table Clustering Order

- **Prend la liste de la colonne de regroupement séparé par des virgules et un sens de tri.**
- **Par défaut toutes les colonnes sont triées de manière ascendante.**
- **A plusieurs impacts.**
  - Limite les possibilités de tri lors des requêtes.
  - Change l’ordre des résultats par défaut.

Cassandra

CQL: Table options

- **Les stratégies de compaction définissent comment se passent les merge des SSTables.**
- **Les options de compression permettent de définir comment les tables sont compressées.**
- **Les options de mise en cache permettent de configurer le cache de la clé ainsi que des enregistrements.**

Cassandra

CQL: Définition des colonnes

- **Une table est définie par un ensemble de colonnes au moment de la création de la table.**
- **Une colonne est définie par son nom et son type.**
- **Deux modificateurs sont disponibles:**
  - **STATIC**: La colonne sera partagée par toutes les lignes qui appartiennent à la même partition.
  - **PRIMARY KEY**: Déclare la colonne comme était la seule composante de la clé de la table.

Cassandra

CQL: Primary Key

- **Une clé primaire est composée de 2 parties**
  - Partition key: Une ou plusieurs colonnes. Obligatoire pour toutes les tables. Elle est partagée pour un ensemble d'enregistrements.
  - Clustering column: Colonnes servant à définir l'ordre de clustering pour la partition de la table.

PRIMARY KEY (a): a est une clé de partitionnement sans colonne de clustering.

PRIMARY KEY (a, b ,c): a est la clé de partition, b et c sont des colonnes de clustering.

PRIMARY KEY ((a, b), c): a et b sont la clé de partition et c est la colonne de clustering.

Cassandra

CQL: SELECT

- **Lit une ou plusieurs colonnes pour plusieurs enregistrements d'une table une table.**
- **Renvoie l'ensemble des colonnes pour les enregistrements correspondant à la requête.**
- **Contient au moin une clause de sélection et le nom de la table sur laquelle la sélection est effectuée.**
- **Peut contenir une clause WHERE ainsi que des clauses de tri et de limite de résultats.**
- **Il n'est pas possible d'effectuer des jointures.**
- **Le sélecteur**
  - Un nom de colonne.
  - Un terme ou une valeur.
  - Un cast qui permet de changer le type de notre colonne.
  - Un appel de fonction

Cassandra

CQL: WHERE

- **Permet d'appliquer un filtre à notre requête. Ne peut concerner que les colonnes de la clé primaire ou des colonnes indexées.**
- **Pour filtrer sur une colonne de clustering, toutes les colonnes précédentes doivent avoir été également filtrées.**

```
CREATE TABLE posts (
   userid text,
   blog_title text,
   posted_at timestamp,
   entry_title text,
   content text,
   category int,
   PRIMARY KEY (userid, blog_title, posted_at)
)
```

```
SELECT entry_title, content FROM posts
 WHERE userid = 'john doe'
  AND blog_title='John''s Blog'
  AND posted_at >= '2012-01-01' AND posted_at < '2012-01-31'

SELECT * FROM posts WHERE token(userid) > token('tom') AND token(userid) < token('bob')

SELECT * FROM posts WHERE userid = 'john doe'  AND (blog_title, posted_at) > ('John''s Blog', '2012-01-01')

SELECT * FROM posts  WHERE userid = 'john doe' AND blog_title > 'John''s Blog'AND posted_at > '2012-01-01'
```

Cassandra

CQL: GROUP BY

- **Permet d’effectuer une agrégation de tous les enregistrement qui partagent les mêmes valeurs pour un ensemble de colonnes.**
- **Il est possible de grouper les enregistrements au niveau de la clé de partition.**
- **Si une colonne de clé primaire est filtrée, elle ne peut pas être présente dans le GROUP BY.**
- **Si une colonne est sélectionnée sans fonction d’agrégation, la première valeur du groupe est retournée.**

Cassandra

CQL: ORDER BY

- **Permet de sélectionner l’ordre des résultats renvoyés.**
- **Les tris sont limités par le clustering order.**
  - S’il n’y a aucun clustering order spécifique, alors seuls les ordres permis par les colonnes sélectionnées et son inverse est possible.
  - Il n’est possible de trier que par une seule colonne avec les clés primaires composites.

Cassandra

CQL: ALLOW FILTERING

- **CQL permet uniquement d'effectuer des filtres que lorsque tous les enregistrements lus seront retournés.**
- **L'option ALLOW FILTERING permet d'autoriser de filtrer sur des colonnes non indexées.**
- **Peut avoir des performances imprévisibles.**

Cassandra

CQL: Update

- **Update permet de mettre à jour une ou plusieurs colonnes pour un enregistrement.**
- **SET permet de mettre à jour une colonne n'appartenant pas à la clé primaire.**
- **Ne vérifie pas l'existence de la ligne.**
  - Agit comme un upsert.
  - Impossibilité de déterminer s'il s'agit d'une création ou d'une mise à jour.
  - Il est possible d'appliquer des conditions à l'application de la mise à jour.
- **Toutes les mises à jour au sein de la même clé de partition sont appliquées de manière atomique.**
- **Les opérations de CUD peuvent avoir des options supplémentaires**
  - TIMESTAMP: Définit le timestamp de la transaction.
  - TTL: Défini un Time To Live pour les valeurs insérées.

Cassandra

CQL: DELETE

- **Supprime des colonnes ou des enregistrements.**
  - Si les noms des colonnes sont fournies, seules celles-ci sont supprimées de l'enregistrement.
  - Sinon l'engistrement complet est supprimé.
- **La clause WHERE indique quel enregistrement doit être supprimé.**
  - Plusieurs lignes peuvent être supprimées grâce à l'opérateur IN.
  - Il est possible de supprimer plusieurs lignes grâce à un opérateur d'inégalité.
- **Toutes les suppression sur la même clé de partition sont appliquées atomiquement.**
- **Peut être conditionnelle à l'aide d'un IF comme pour l'UPDATE ou l'INSERT.**

Cassandra

CQL: BATCH

- **Permet de faire des modifications multiples au sein de la même instruction.**
- **Permet de limiter le trafic réseau.**
- **Toutes les mises à jour appartenant à une partition donnée sont réalisées de manière isolée.**
- **Si aucun TIMESTAMP n'est pas spécifié toutes les opérations auront le même TIMESTAMP.**

```
BEGIN BATCH
  INSERT INTO users (userid, password, name) VALUES ('user2', 'ch@ngem3b', 'second user');
  UPDATE users SET password = 'ps22dhds' WHERE userid = 'user3';
  INSERT INTO users (userid, password) VALUES ('user4', 'ch@ngem3c');
  DELETE name FROM users WHERE userid = 'user1';
APPLY BATCH;
```

Cassandra

CQL: Index secondaires

- **Permet de créer un nouvel index secondaire pour une colonne existante.**
- **Un nom peut être donné à l'index.**
- **Si des données sont déjà présentes dans la colonne, elle sera indexée de manière asynchrone.**
- **Les nouvelles données seront indexées au moment de l'insertion.**
- **Sur un MAP il est possible d'indexer les clés et les valeurs.**
  - Si la fonction KEYS() est utilisée, l'index sera sur les clés de MAP (permet d'utiliser CONTAINS KEY dans le WHERE).
  - Sinon l'index est sur les valeurs.

Cassandra

CQL: MATERIALIZED VIEW

- **CREATE MATERIALIZED VIEW permet de créer une nouvelle vue matérialisée.**
- **Une vue est un sous ensemble des enregistrements d'une table.**
- **Une mise à jour sur la table est visible sur la vue.**
- **La création d'une vue matérialisée se découpe en trois parties.**
  - Instruction SELECT pour limiter les données incluses dans la vue.
  - Définition de la clé primaire pour la vue.
  - Options pour la vue.
- **Doit avoir une clé primaire.**
  - Doit contenir toutes les clés primaire de la table de base.
  - Ne peut contenir qu'une seule colonne qui ne soit pas une colonne de clé primaire de la table de base.

Cassandra

CQL: Fonctions

- **CQL permet d'utiliser 2 catégories de fonctions.**
  - Les fonctions scalaires (n paramètre -> une sortie).
  - Les fonctions d'agrégations.
- **Plusieurs fonctions existent nativement.**
  - CAST, TOKEN, WRITETIME...
  - MIN, MAX, COUNT
- **Il est possible de définir de nouvelles fonctions en JAVA ou JavaScript.**

Kafka

Kafka

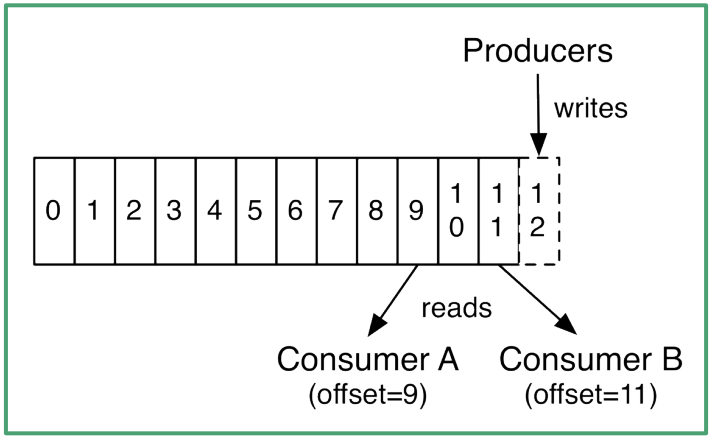
Introduction

- **Système de messagerie distribué.**
- **Créé par LinkedIn.**
- **Tolérant aux pannes.**

Kafka

Les logs

- **Un log est le niveau d'abstraction de stockage le plus simple possible, il s'agit d'une séquence d'enregistrements.**
- **Les enregistrements sont ajoutés à la fin du log et les lectures s'effectuent de manière séquentielles.**



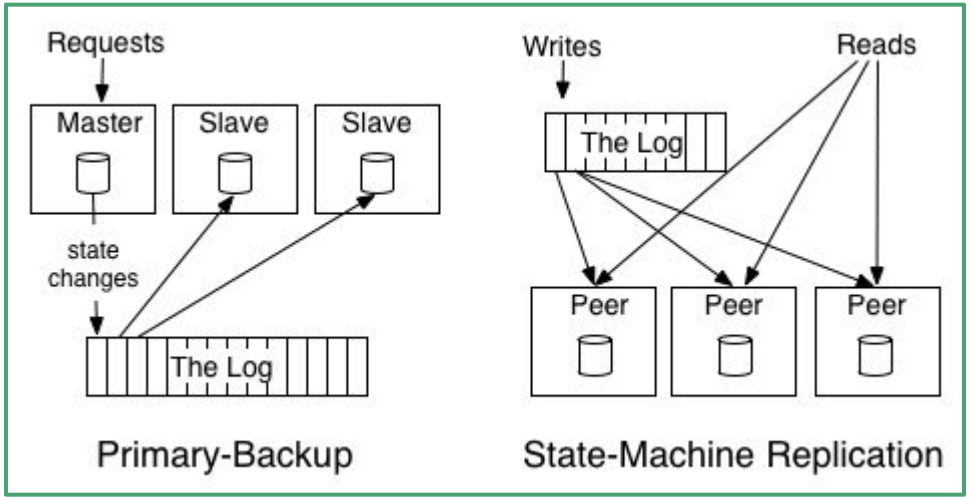
Kafka

Commit log

- **Un commit log dans les bases de données est l’application d’une transaction à la base de données.**
- **Il est utilisé pour garder une trace de ce qui se passe et pour permettre de récupérer un état stable après une panne.**

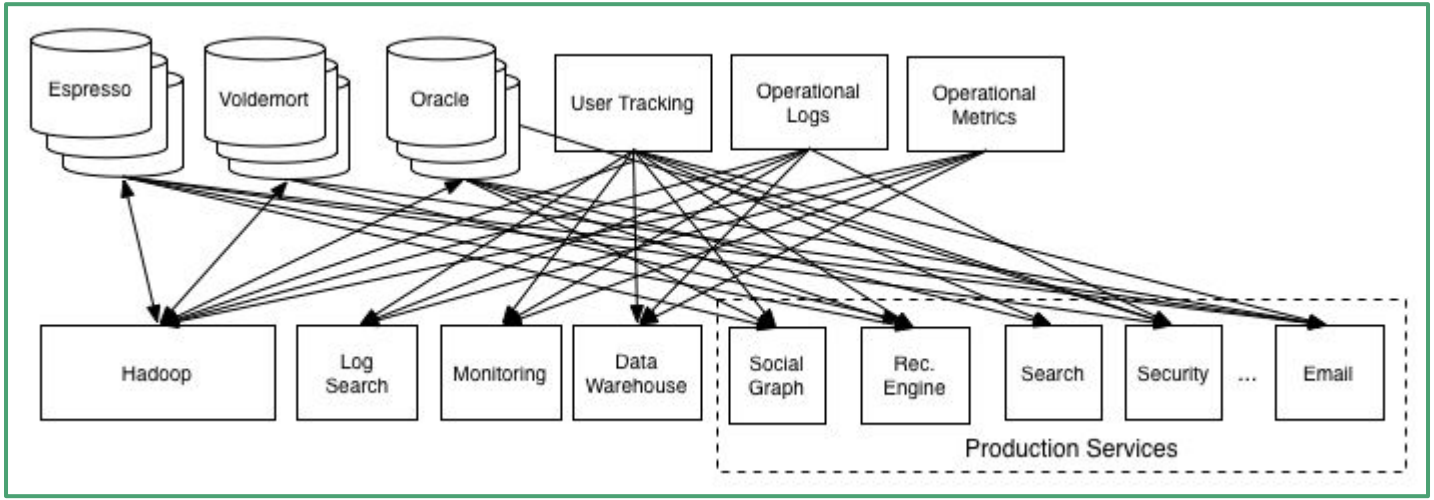
Kafka

Log dans un système distribué



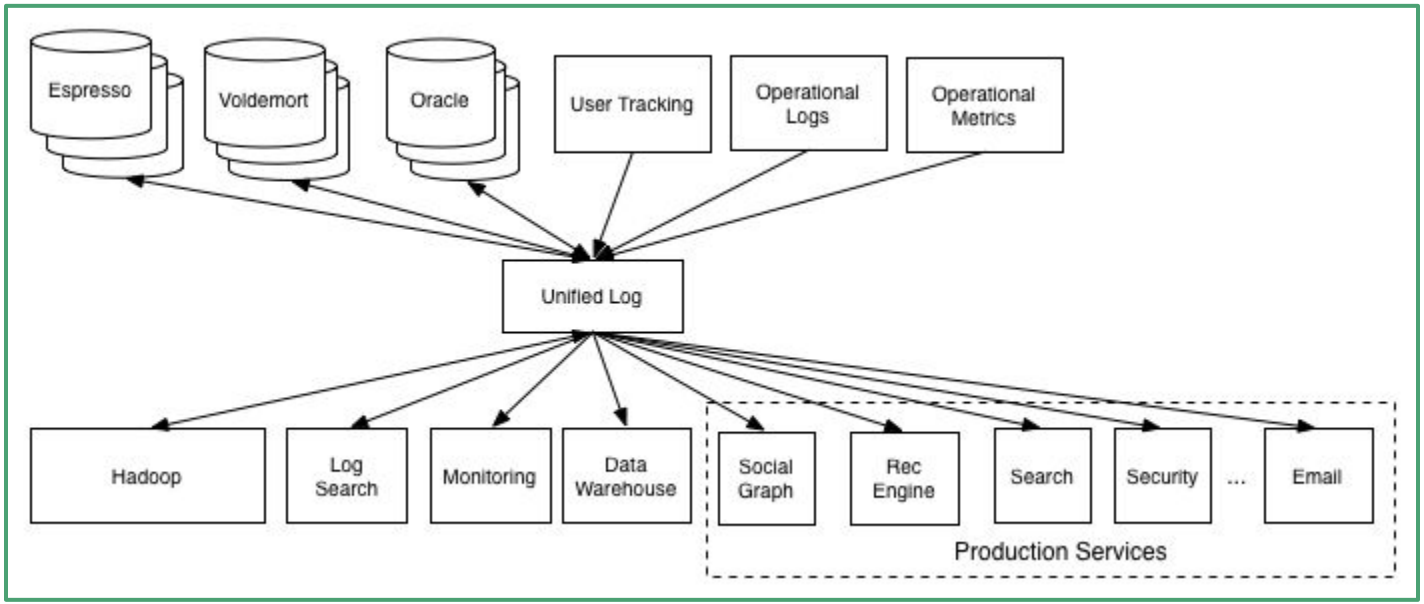
Kafka

Linkedin: Avant Kafka



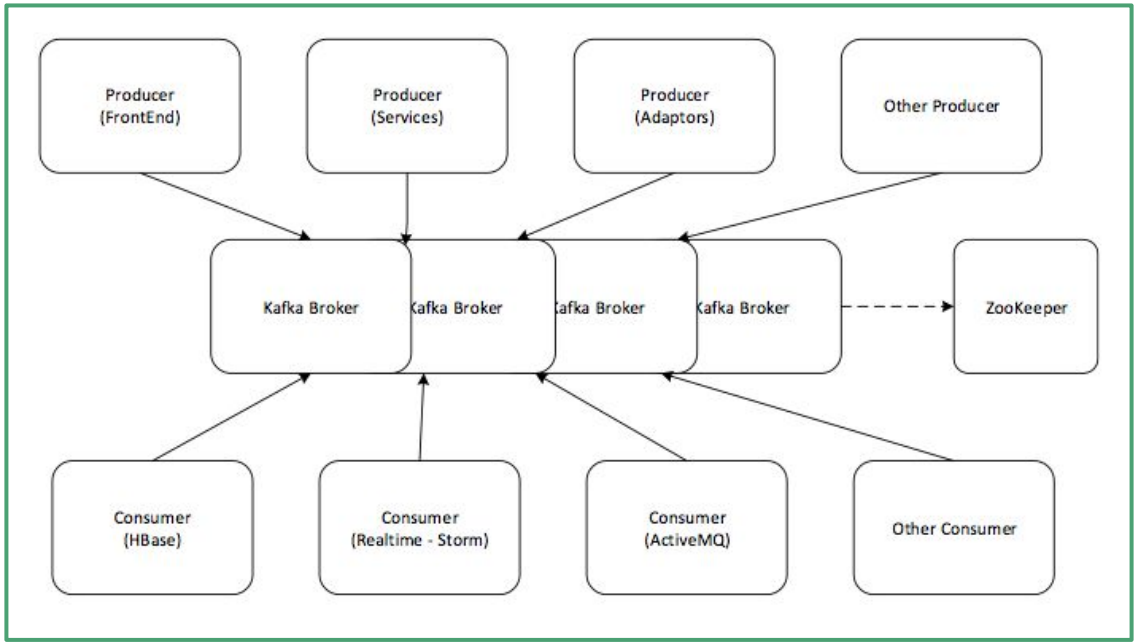
Kafka

Linkedin: Logs unifiés



Kafka

Producer et consumers



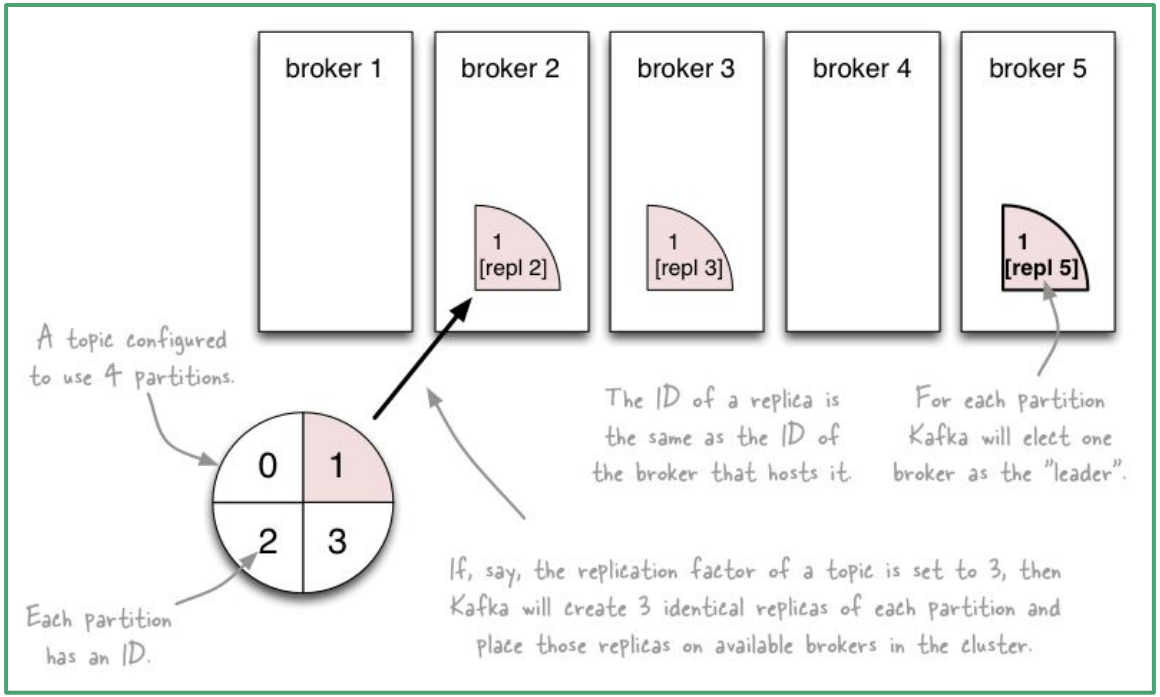
Kafka

Concepts

- **Se lance sur des serveurs appelés brokers.**
- **Les producers sont les processus qui publient des messages dans les brokers.**
- **Les messages sont sauvegardés dans des topics.**
- **Les topics sont divisés en partitions.**
- **Les consumers souscrivent à des topics et traitent les messages des partitions.**

Kafka

Partition



Kafka

Offsets

- **L’offset est la position où un consumer donné consomme une partition.**
- **C’est un identifiant unique permettant de suivre l’évolution du log.**
- **Un consumer utilise le tuple (offset, partition, topic) pour suivre leur consommation dans le log.**

Kafka

Consumer group

- **Les consumer sont responsables du suivi de leurs offsets.**
- **Les consumers sont regroupés par consumer group.**
- **Si un message doit être lu par plusieurs consumer alors ils doivent être dans des groupes différents.**
- **Les brokers ne savent pas quel consumer a consommé le topic, cependant, il garde l'avancement du consumer group.**

Kafka

Exemple

```
val props = new Properties()
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, broker)
props.put(ProducerConfig.CLIENT_ID_CONFIG, appName)
// …
val producer = new KafkaProducer[Any, String](props)
val rnd = new Random()
sys.addShutdownHook(producer.close())

while (!Thread.currentThread.isInterrupted)
 producer.send(new ProducerRecord[Any, String](
  topic,
  "Hi i'm a key",
  rnd.nextString(4)
 ))
```

```
val props = new Properties()
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, broker)
props.put(ConsumerConfig.GROUP_ID_CONFIG, group)
// …
val consumer = new KafkaConsumer[String, String](props)
consumer.subscribe(Collections.singletonList(topic))
sys.addShutdownHook(consumer.close())

while (!Thread.currentThread.isInterrupted)
 consumer.poll(1000).forEach(r =>
  println(r.key + ", " + r.value)
 )
```