

Séance 5

Modèle orienté-document CouchDB, MongoDB



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Rappels

- Modèle de base de données orienté **graphes**
 - Définition de graphe comme ensemble de nœuds et d'arêtes
 - Moteur Neo4j et son langage de requêtes CQL
 - Moteur OrientDB et son modèle de données orienté objets
- Représentation de graphes et **langages de requêtes**
 - Web sémantique avec format RDF et langage SPARQL
 - Description de traversées de graphes avec Gremlin
 - Requêtes sur des APIs avec GraphQL

Objectifs

- Le modèle **orienté-document**
 - Notion de document et ses caractéristiques
 - Design d'un document et cas d'utilisation
 - Documents intégrés ou références
- **Exemples** de bases de données
 - CouchDB
 - MongoDB

Modèle orienté-document (1)

- **Modèle orienté-document** cas particulier du modèle clé-valeur

Les valeurs sont des documents

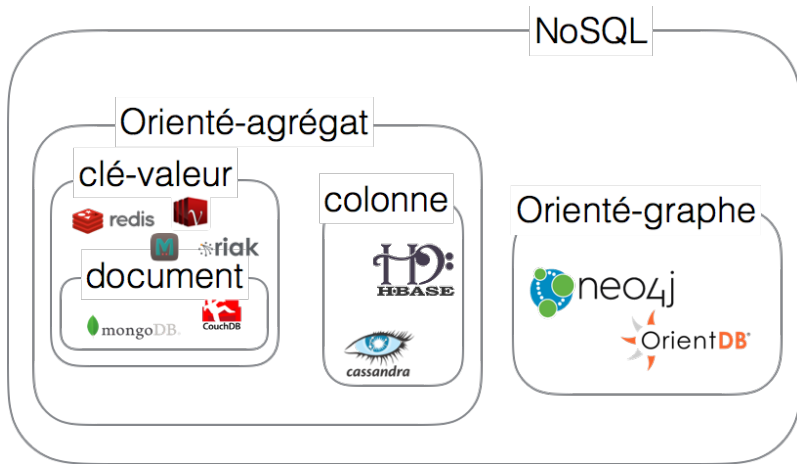
- Document est un **agrégat de données** complexe

Les données possèdent une structure interne

- Format des données proche des **langages orienté objets**

Résolution du problème d'impédance

Le modèle orienté-document (2)



Caractéristiques de l'orienté-document

- **Requêtes** sur tout ou sur une partie du document

Utilisation d'index et moteurs de recherche

- Généralement, absence de **schéma** prédéfini

Flexibilité vs gestion de la cohérence

- **Versioning** intégré dans le moteur

Historique des modifications et gestion des accès concurrents

Document (1)

- Un **document** est une entité structurée

Son contenu est complètement lisible par le moteur NoSQL

- Plusieurs **formats** de représentation possibles

JSON, XML, BSON, YAML...

- Aspect **auto-descriptif** d'un document

Format textuel lisible pour les humains

Document (2)

- Structure de données **hiérarchiques**

Maps, collections, scalaires

- Regroupement de documents dans des **collections**

Semblables aux tables dans les SGBDR

JavaScript Object Notation (JSON)

- **Format textuel** de données structurée

Dérivé de la notation des objets JavaScript

- Créé par Douglas Crockford, standardisé par le **RFC 7159**

- **Communication d'applications** dans environnement hétérogène

Ajax, sérialisation d'objets, fichiers de config...

- Lisible, **facile à parser**, facile à apprendre

Format JSON

- Valeurs **simples**

Chaines de caractères, nombres entiers, booléens, null

- Valeurs **composées**

- Tableaux

[elem1, elem2...]

- Objets

{ "clé1" : valeur1, "clé2" : valeur2 }

- Détails du **format JSON** sur <http://www.json.org/jsonfr.html>

Exemple de document JSON

```
{
  "class": "algebra",
  "students": [{
    "name": "Susan",
    "activities": [{
      "name": "soccer",
      "type": "sport"
    }, {
      "name": "band",
      "type": "music"
    }]
  }]
}
```

- **Valideur** en ligne <http://jsonlint.com/>

Permet de vérifier qu'un document JSON est bien formé

Opérations sur documents

- Document identifié par une **clé unique**

Création index sur l'ensemble des clés pour accélérer les requêtes

- Recherche **dans le contenu** sur base des champs

Utilisation de métadonnées rend recherches possibles et efficaces

- **Édition** d'un document sur base de ses champs

Ajout/suppression de champs, modification de valeurs

Design de document

- Deux manières de gérer les objets stockés
 - Objet directement **inclus** dans un document (embedded)
 - Objet stocké comme document séparé et ensuite **référéncé**
- Problématique de la **normalisation** et des **jointures**
- Exemple : *Adresse du client dans une facture*
 - **Séparée** dans un document spécifique ?
On ajoute la référence à l'objet adresse (id) dans l'objet facture
 - **Inclue** dans l'objet facture ? (embedded)
L'adresse est un champ dans l'objet JSON Facture

Embedding vs Reference

- Choix de l'**inclusion** ou de la **référence** selon type de relation

Relation entre l'objet contenant et l'objet interne

- Trois **types de relation** possibles

- **One-to-One** (embarqué)

L'adresse d'une personne...

- **One-to-Many** (embarqué si objet interne pas utilisé ailleurs)

Une personne possède plusieurs adresses...

- **One-to-Many** (références si répétitions d'un même objet)

On souhaite associer un livre à son éditeur...

Documents vs RDBMS (1)

- Présence vs absence de schéma
- Normalisation des données vs redondance de certaines données
Une adresse peut être stockée plusieurs fois (e.g. facturation)
- Optimisation vs difficulté pour les jointures
Requêtes à travers plusieurs tables/documents

Documents vs RDBMS (2)

- Problème de l'**impédance** objet/relation

Plus besoin d'ORMs! Récupération « directe » d'objets

- Facilité du **sharding** de données

Répartition des agrégats sur plusieurs clusters

Cas d'utilisation (1)

- Logs événementiels

*Permet de regrouper les **logs** de différentes applications, de formats souvent différents*

- CMS et blog

*Le format document convient souvent pour le **développement web** : sites, articles, commentaires, profils, ...*

Cas d'utilisation (2)

■ Statistiques

*Les documents peuvent servir de stockage pour des **données analytiques** en temps réel (ex : nombre de vues d'une page, visiteurs uniques...), avec possibilité d'ajout de nouvelles métriques*

■ Applications d'e-commerce

Permet d'avoir des formats flexibles pour les produits et les commandes (évolutions possible des schémas de données)

Cas de non utilisation

- Transactions complexes

Les DB orienté-document ne supportent pas les transactions atomiques à travers plusieurs documents

- Requêtes sur des agrégats variables

Si la structure des documents change trop et que des requêtes très spécifiques sur la structure des documents sont nécessaires, une DB orienté-document ne conviendra pas

CouchDB



CouchDB (1)

Couch : Cluster of Unreliable Commodity Hardware

- Créé par Damien Katz en 2005, publié sous Licence Apache en 2008.
- Objectif : un moteur de DB adapté à Internet.

Documents JSON manipulables en javascript à travers une interface API REST HTTP

- Grande **richesse fonctionnelle** : Langages multiples (souvent JS), support de MapReduce, génération d'index et de vues.

CouchDB (1)

Couch : Cluster of Unreliable Commodity Hardware

- ACID au niveau des documents : Cohérence des données assurée par une gestion des **versions**
- Développé en Erlang
- Version 2.0 : fonctionnement en mode single-node ou cluster

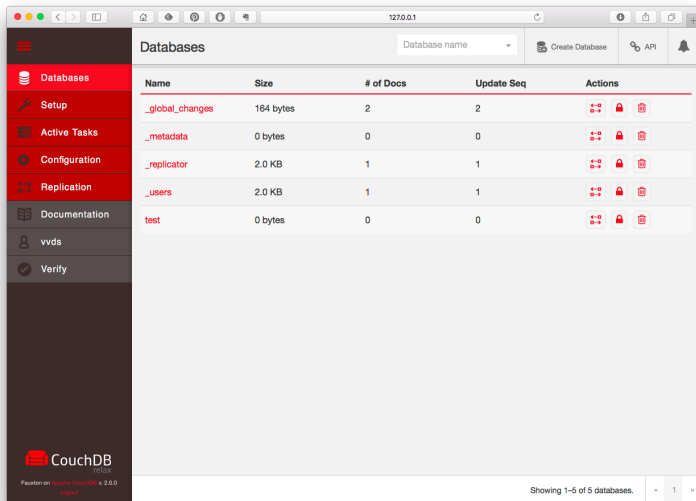
Utilisation

Interface Web, API REST ou Python

- Interface Web tourne sur `http://127.0.0.1:5984/_utils/` (installation single-node)
 - Configuration et mise en place de la réplication
 - Vérification du fonctionnement
 - Création de bases de données
- API REST
 - Utilisation de l'outil `curl` depuis la console
 - Envoi de requêtes HTTP sur base d'URI
- Python

```
pip install couchdb
```

Interface Web Fauxton



The screenshot displays the Fauxton web interface for CouchDB. The interface is divided into a sidebar on the left and a main content area. The sidebar contains navigation links: Databases, Setup, Active Tasks, Configuration, Replication, Documentation, vuds, and Verify. The main content area is titled "Databases" and features a search bar and a "Create Database" button. Below this is a table listing the databases in the instance.

Name	Size	# of Docs	Update Seq	Actions
_global_changes	164 bytes	2	2	
_metadata	0 bytes	0	0	
_replicator	2.0 KB	1	1	
_users	2.0 KB	1	1	
test	0 bytes	0	0	

The bottom of the interface shows the CouchDB logo and version information: "Fauxton on Apache CouchDB v 2.0.0". A status bar at the bottom right indicates "Showing 1-5 of 5 databases."

Création d'un document

- Avec Fauxton : Ajout de document en JSON
- API REST : Envoi de requetes HTTP sur base d'URI
 - Utilisation de l'outil curl depuis la console

```
1 $ curl -H 'Content-Type:application/json' -X POST http://localhost:5984/test/ -d '{ "nom":"Van den Schrieck", "prénom":"Virginie" }'
```

```
{"ok":true,"id":"29311ddf85e294038403fde75d00317d","rev":"1-573ee2fbf1d34ac351830b1216b4b07b"}
```

Numéro de révision : Permet de gérer les conflits de mise à jour

Création d'un document (2)

■ Python :

```
1 >>> import couchdb
2 >>> couch = couchdb.Server()
3 >>> db = couch['test']
4 >>> me = {'nom' : 'Van den Schrieck', 'prenom' : 'Virginie'}
5 >>> db.save(me)
6 (u'29311ddf85e294038403fde75d00191b', u'1-86510
d798bbc3f98c72d8dc1c934dae7')
7 >>> for item in db :
8 ...     print db[item]
9 ...
10 <Document u'1'@u'3-4f2c3cdb5580fe3683565ba2c454dc87' {u'pr\xe9nom':
u'S\xe9bastien', u'titre': u'Dr Ir.', u'nom': u'Comb\xe9fis'}>
11 <Document u'29311ddf85e294038403fde75d00191b'@u'1-86510
d798bbc3f98c72d8dc1c934dae7' {u'nom': u'Van den Schrieck', u'prenom
': u'Virginie'}>
12 >>> me
13 {'nom': 'Van den Schrieck', '_rev': u'1-86510
d798bbc3f98c72d8dc1c934dae7', '_id': u'29311
ddf85e294038403fde75d00191b', 'prenom': 'Virginie'}
```

Sélection de documents

■ API REST : Requêtes GET

- Utilisation de l'outil curl depuis la console
- URI/vue spéciale `_all_docs` pour retrouver tous les documents

```
1 $ curl -X GET http://localhost:5984/test/1
2 {"_id":"1","_rev":"2-78418b8fed3be090939beefdb8500b22","nom":"
3 Combefis","prenom":"Sebastien","specialites":["LaTeX","le gras"]}
4
5 $ curl -X GET http://localhost:5984/test/2
6 {"_id":"2","_rev":"2-3222ceef420d66b5d14a850a3534fb47","nom":"Van
7 den Schrieck","prenom":"Virginie","specialites":["Monocycle","BGP",
8 "Docker","Cookies"]}
9
10 $ curl -X GET http://localhost:5984/test/_all_docs
11 {"total_rows":2,"offset":0,"rows":[
12 {"id":"1","key":"1","value":{"rev":"2-78418
13 b8fed3be090939beefdb8500b22"}},
14 {"id":"2","key":"2","value":{"rev":"2-3222
15 ceef420d66b5d14a850a3534fb47"}}
16 ]}
```

Vues

Une vue représente le résultat d'une requête. Les vues sont calculées à l'aide de fonctions map/reduce écrites en Javascript. Elles sont regroupées dans des **design documents** :

```
1 {
2   "_id": "_design/test",
3   "_rev": "7-a7df36f73ea3acaab37135a1663f2a74",
4   "views": {
5     "all": {
6       "map": "function (doc) {\n  if(doc.nom)\n    emit(doc._id,\n      doc);\n}"
7     },
8     "num_spec": {
9       "reduce": "_sum",
10      "map": "function (doc) {\n  if(doc.specialites)\n    emit(doc\n      ._id, doc.specialites.length);\n}"
11    }
12  },
13  "language": "javascript"
14 }
```

Vues (2)

Pour afficher une vue avec l'API REST, on peut utiliser son URI :

```
1 $ curl -X GET http://localhost:5984/test/_design/test/_view/all
2 {"total_rows":2,"offset":0,"rows":[
3 {"id":"1","key":"1","value":{"_id":"1","_rev":"2-78418
4 b8fed3be090939beefdb8500b22","nom":"Combefis","prenom":"Sebastien",
5 "specialites":["LaTeX","le gras"]}},
6 {"id":"2","key":"2","value":{"_id":"2","_rev":"2-3222
7 ceef420d66b5d14a850a3534fb47","nom":"Van den Schrieck","prenom":"
8 Virginie","specialites":["Monocycle","BGP","Docke","Cookies"]}}
9 ]}
10
11 $ curl -X GET http://localhost:5984/test/_design/test/_view/
12 num_spec
13 {"rows":[
14 {"key":null,"value":6}
15 ]}
16
17 $ curl -X GET http://localhost:5984/test/_design/test/_view/
18 num_spec?group_level=1
19 {"rows":[
20 {"key":"1","value":2},
21 {"key":"2","value":4}
22 ]}
```

Vues (3)

Même chose en Python :

```
1 >>> for i in db.interview("test/all",10) :
2 ...     print i
3 ...
4 <Row id=u'1', key=u'1', value={u'nom': u'Combefis', u'_rev': u'
5 2-78418b8fed3be090939beefdb8500b22', u'_id': u'1', u'prenom': u'
6 Sebastien', u'specialites': [u'LaTeX', u'le gras']}>
7
8 <Row id=u'2', key=u'2', value={u'nom': u'Van den Schrieck', u'_rev'
9 : u'2-3222ceef420d66b5d14a850a3534fb47', u'_id': u'2', u'prenom': u
10 'Virginie', u'specialites': [u'Monocycle', u'BGP', u'Docker', u'
11 Cookies']}>
12
13
14 >>> for i in db.interview("test/num_spec",10) :
15 ...     print i
16 ...
17 <Row key=None, value=6>
18
19
20 >>> for i in db.interview("test/num_spec",10, group_level=1) :
21 ...     print i
22 ...
23 <Row key=u'1', value=2>
24 <Row key=u'2', value=4>
```


CouchBase Server

- CouchDB riche au niveau fonctionnel, mais non distribué

Plus adapté à petits développements que applications critiques

- Membase : moteur NoSQL orienté clé-valeur distribué et en mémoire
- Couchbase : cluster élastique et décentralisé, deux types de **buckets**
 - memcached : stockage en RAM, système de cache distribué
 - couchbase : persistance sur le disque
- Couchbase intègre deux moteurs différents : Attention aux performances selon les cas.

MongoDB



MongoDB (1)

- Créé en 2009, développé par la société MongoDB Inc. (10gen à l'origine).
- A l'origine, partie d'un projet de plateforme applicative distribuée conçue pour le cloud. La DB ayant attiré l'attention, elle a été publiée en **open-source**
- Rare SGBD NoSQL développé en **C++** (performances !)

MongoDB (2)

- Format interne de stockage des documents : **BSON** (plus compact que JSON). Taille max d'un document : 16Mo

Mais JSON comme format de travail

- Notion de **collections** : Similaire aux tables en relationnel

Pas de contrainte de structure mais clé unique (`_id`)

- Possibilité de stocker des **objets larges**

- Soit BSON, soit à l'aide de GridFS
- Morceaux de fichiers dans la collection `chunks`, métadonnées dans `files`
- Outil en ligne de commande pour importer ou extraire des documents : `mongofiles`

MongoDB (3)

- Transactions

- Atomicité uniquement lors de la mise à jour d'un document
- Durabilité assurée par journalisation avant écriture

- Consistence et réplication

Possibilité de définir des replica sets et d'indiquer le nombre de serveurs sur lequel une écriture doit être absolument répliquée

- Sharding

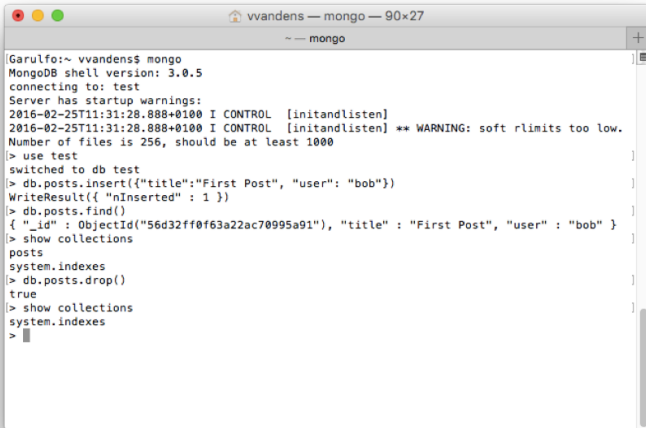
Ajout dynamique de nouveaux noeuds, répartition sur base de clés

Interactions avec MongoDB

Comme les autres solutions NoSQL, MongoDB offre plusieurs moyens d'**interaction** :

- Utilisation
 - Depuis la console, avec un langage de requête basé sur JSON
 - Depuis des programmes écrits dans différents langages, à l'aide de drivers : Python, NodeJS, Java, C, ...
- Configuration
 - En ligne de commande
 - A travers le fichier de configuration `mongod.conf` en YAML
 - Possibilité d'API REST (dépend des versions - frameworks externes)

Console MongoDB



```
Garulfo:~ vvandens$ mongo
MongoDB shell version: 3.0.5
connecting to: test
Server has startup warnings:
2016-02-25T11:31:28.888+0100 I CONTROL [initandlisten]
2016-02-25T11:31:28.888+0100 I CONTROL [initandlisten] ** WARNING: soft rlimits too low.
Number of files is 256, should be at least 1000
[> use test
switched to db test
[> db.posts.insert({"title":"First Post", "user": "bob"})
WriteResult({ "nInserted" : 1 })
[> db.posts.find()
{ "_id" : ObjectId("56d32ff0f63a22ac70995a91"), "title" : "First Post", "user" : "bob" }
[> show collections
posts
system.indexes
[> db.posts.drop()
true
[> show collections
system.indexes
> █
```

Requêtes

- MongoDB possède un langage de requêtes exprimées via JSON avec possibilité de filtrage et de tri
- Exemple : Requête SQL

```
SELECT * FROM Posts WHERE Title LIKE '%mongo%';
```

- En MongoDB :

```
db.posts.find({ title:/mongo/ });
```


Opérations MongoDB

<u>CRUD</u>	MongoDB	HTTP	SQL
C reate	Insert	POST	INSERT
R ead	Find	GET	SELECT
U ppdate	Update	PUT	UPDATE
D eleate	Remove	DELETE	DELETE

Ajout de documents

- Par insertion

```
> db.posts.insert({"title": "Second Post", "user": "alice"})
```

- Par mise à jour d'un document n'existant pas encore (Alice pas encore créée)

```
> db.posts.update(  
  {"user": "alice" },  
  {"title": "Second Post", "user": "alice"}, {upsert: true})
```

- avec `save()` sur un document json sans champ `_id`, ou avec un `_id` non existant

```
> db.posts.save({"title": "Second Post", "user": "alice"})
```

Recherche de documents

Utilisation d'objets JSON pour spécifier les champs de la recherche

```
> db.posts.find()

> db.posts.find({ "user": "alice" })

> db.posts.find({ "user": { $in: ["alice", "bob"] } })

> db.posts.find(
{ "user": "alice", "commentsCount": { $gt: 10 } })

> db.posts.find(
{ $or: [{ "user": "alice" }, { "user": "bob" }] })
```

Mise à jour de documents

■ Avec update()

- premier paramètre permet de trouver le document à changer,
- deuxième précise la modification,
- troisième indique s'il faut mettre à jour un seul ou tous les documents répondant au critère.

```
> db.posts.update(  
  {"user": "alice"},  
  {$set: {"title": "Second Post"}}, {multi: true})
```

■ avec save() (document créé si id non existant)

```
> db.posts.save({  
  "_id": ObjectId("50691737d386d8fadbd6b01d"),  
  "title": "Second Post",  
  "user": "alice"  
});
```

Suppression de documents

- Suppression d'une collection

```
> db.posts.remove()
```

- Suppression de tous les posts répondant à un critère

```
> db.posts.remove({ "user": "alice" })
```

Alternative à partir de la version 3.2 : `deleteMany()`

- Suppression du premier post répondant à un critère (paramètre `justOne`)

```
> db.posts.remove({ "user": "alice" }, true)
```

Alternative à partir de la version 3.2 : `deleteOne()`

Crédits

- Photos des logos depuis Wikipédia
- <https://www.flickr.com/photos/nerdcoregirl/4335907588/>
- <https://www.flickr.com/photos/stan7826/262616133>
- <https://www.flickr.com/photos/crodas/4941192194>