

# SolR

## Cloud et Recherches avancées

*david.thibau@gmail.com*  
2018

# Agenda

- Rappels :
  - Historique projet
  - Configuration
  - Coeur et collections, Shémas
- L'indexation
  - Analyseur
  - REST Api
  - Nested, Optimisation
  - Update Request Processor
- Recherche Rappels
  - Query parser
  - Rappels sur la syntaxe Lucene
  - Recherche par facette
  - Recherche par groupe
  - Relation parent/child, Jointure
  - Recherche spatiale
- Distribution et réplication
  - Introduction
  - SolrCloud
  - Sécurité, HDFS, ...

# Rappels

Historique projet, versions

Création de coeurs

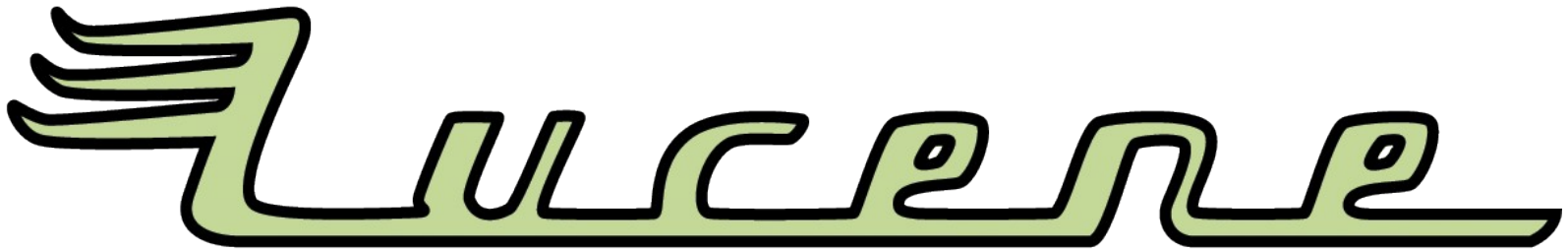
Configuration de schéma

Configuration API Rest

Interface d'administration



- Open-source, écrit en Java;
- 1ère release Septembre 2008
- Serveur de recherche autonome
- Basé sur Lucene, une librairie java de recherche plein-texte;



- La « couche basse » de SolR : une librairie Java pour écrire et rechercher dans les fichiers d'index;
- Utilisé dans de nombreuses solutions
- Projet démarré en 2000
- SolR expose les fonctionnalités de Lucene dans un serveur, au travers de HTTP;

# SolR : historique

- Développé au sein de CNET par Yonik Seeley
- Transféré à la fondation Apache en 2006;
- Incubé jusqu'en 2007 : v1.2;
- Aujourd'hui, Lucene et SolR sont dans le même projet Apache, et relasés ensemble;
  - On est donc passé directement de SolR 1.4 à 3.0 pour que les n° de version correspondent
- Février 2015 : v5.x (mode standalone)
- Actuellement, V7.5
- Grosse communauté d'utilisateurs
- LucidWorks : support commercial

# Données

- Lucene permet de stocker des **coeurs** ou **collections** (cloud)
- Un coeur contient des **documents**
- Un document contient des **champs** (« fields »)
- Un champ peut être décomposé en **termes** (« terms »)
- Les termes constituent **l'index** permettant d'accélérer la recherche d'un document à partir d'un de ses termes

# Qu'est-ce qu'un index ?

	A	B
1	term	docs
2	pizza	3, 5
3	solr	2
4	lucene	2, 3
5	sourcesense	2, 4
6	paris	1, 10
7	tomorrow	1, 2, 4, 10
8	caffè	3, 5
9	big	6
10	brown	6
11	fox	6
12	jump	6
13	the	1, 2, 4, 5, 6, 8, 9

- Chaque document a un *id* et est associé à une liste de termes
- Pour chaque terme, on garde la liste des *id* de documents qui contiennent ce terme



# Recherche plein-texte



6 results found in 8 ms Page 1 of 1 <<>>

<http://thetechietutorials.blogspot.com/2011/06/how-to-build-and-start-apache-solr.html> [More Like This](#)

[Techie Tutorials: How to build and start Apache Solr admin app from source with Maven](#)

<http://thetechietutorials.blogspot.com/2011/06/how-to-build-and-start-apache-solr.html>

<http://thetechietutorials.blogspot.com/2011/07/updated-pom-for-building-and-starting.html> [More Like This](#)

[Techie Tutorials: Updated POM for building and starting Solr Admin App from Solr 3.3 source](#)

<http://thetechietutorials.blogspot.com/2011/07/updated-pom-for-building-and-starting.html>

<http://thetechietutorials.blogspot.com/2011/06/solr-and-nutch-integration.html> [More Like This](#)

[Techie Tutorials: Solr and Nutch Integration](#)

<http://thetechietutorials.blogspot.com/2011/06/solr-and-nutch-integration.html>

« SolR browse », interface de recherche d'exemple fournie par SolR

# Types de recherche

- La mission de SolR est de fournir toutes les fonctionnalités demandées par un moteur de recherche :
  - Tri par pertinence ... ou autre
  - Surbrillance
  - Suggestion
  - Correction de typo
  - Gestion des synonymes
  - Phonétiques
  - Recherche géo-graphique
  - Aggrégation, facettes
  - QBE (Query By Example)
  - ...

# Interface d'admin

- SolR ne propose pas d'interface utilisateur mais offre différents moyen d'intégration
- Il offre une interface web d'administration qui permet :
  - Gérer et parcourir les index (simple noeud ou disribué) :
    - Tester les analyseurs
    - Effectuer des recherches
    - Indexer les données
    - Visualiser les fichiers de config
    - Modifier le Schema
  - Surveiller un serveur standalone
  - Surveiller un cloud et les noeuds qui le constitue

# Solr admin



- Dashboard
- Logging
- Cloud
- Collections
- Java Properties
- Thread Dump

gettingstarted

- Overview
- Analysis
- Dataimport
- Documents
- Files
- Query
- Stream
- Schema

Core Selector

Use [original UI](#)

## Collection: gettingstarted

Config name: gettingstarted  
Max shards per 2  
node:  
Replication 2  
factor:  
Auto-add   
replicas:  
Router name: compositeld

## Shards

### shard1

Range: 80000000-ffffff  
Active:   
Replicas: gettingstarted\_shard1\_replica2  
gettingstarted\_shard1\_replica1

### shard2

Range: 0-7ffffff  
Active:   
Replicas: gettingstarted\_shard2\_replica2  
gettingstarted\_shard2\_replica1

Documentation

Issue Tracker

IRC Channel

Community forum

Solr Query Syntax

Création de coeur

# Script de démarrage

- Pour créer un cœur/collection, SolR doit être démarré
- Le script solr et la commande **start**

```
bin/solr start [options]
```

```
bin/solr start -help
```

```
bin/solr restart [options]
```

```
bin/solr stop [options]
```

```
bin/solr status
```

# Options de démarrage

- **-c** : Mode cloud
- **-h** et **-p** : Host et port d'écoute
- **-d <dir>** : Répertoire du serveur par défaut : *server*
- **-z <zkHost>** : Mode cloud : Adresse de ZooKeeper
- **-m <memory>** : -Xms et -Xmx
- **-s <dir>** : *solr.solr.home* Répertoire home de la configuration
- **-t <dir>** : *solr.data.home*, Répertoire de dstockage des index
- **-e <example>** : cloud,techproducts,dih,schemaless
- **-a** : paramètres additionnels de la JVM
- **-v** et **-q** : Niveau de verbosité

# Création de coeur

- La création d'un cœur s'effectue alors avec le script ***solr*** et la commande ***create***

```
bin/solr create options
```

```
bin/solr create -help
```

La commande *create* détecte le mode d'exécution de SolrR et construit soit un cœur soit une collection (*SolrCloud*)



# Options de *create*

- **-c <name>** (requis) : Le nom du cœur ou de la collection à créer
- **-d <confdir>** : Le répertoire de configuration.
  - Soit une valeur prédéfinie :
    - **\_default**: Configuration minimale avec détection automatique de champs
    - **sample\_techproducts\_configs**: Configuration avec les fonctionnalités optionnelles de l'exemple *techproduct*
  - Soit un chemin vers une configuration spécifique contenant un schéma

# Autres commandes liées aux coeurs

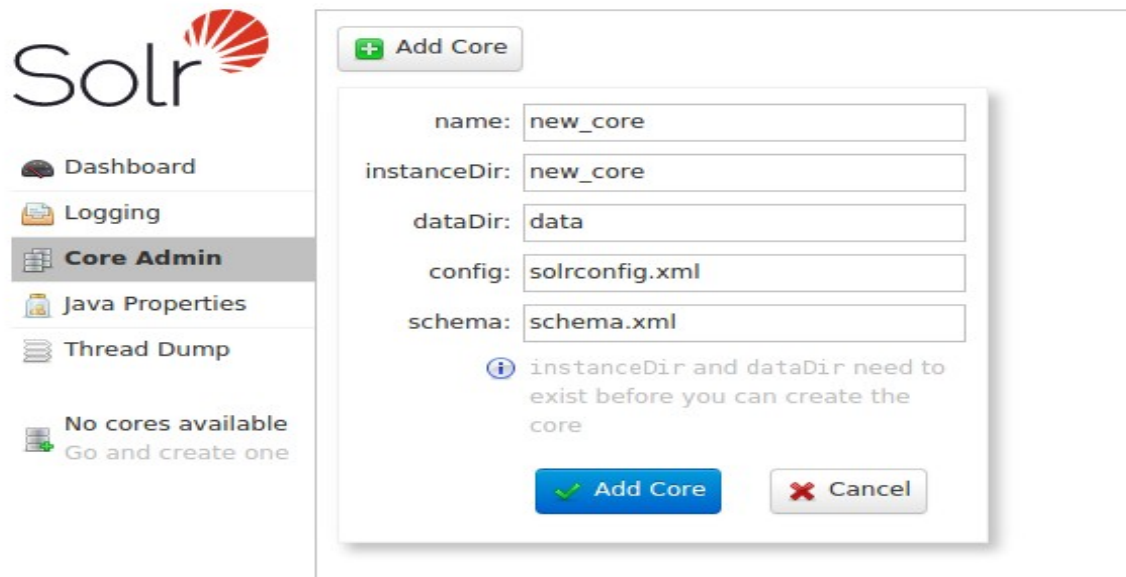
- *Solr* propose 2 autres commandes pour la gestion des coeurs

```
bin/solr healthcheck -c <name>
```

```
bin/solr delete -c <name>
```

# Interface d'administration

- L'interface d'administration permet également de créer un cœur
- Il faut cependant avoir préalablement créer les répertoires de configuration et de données avec les fichiers *solrconfig.xml* et *schema.xml*
- 



# API Rest

- Les 2 méthodes utilisent l'API Rest :
- `http://localhost:8983/solr/admin/cores?action=CREATE&name=formation&instanceDir=formation`

# ***Atelier***

## *Création de coeurs*

- Créer 1 cœur :
  - “*formation*” en utilisant la configuration prédéfinie ***\_default***
- Accéder à l'interface d'administration et vérifier le schema avec le schéma browser
- Regarder les fichiers de configuration créés

# Configuration d'un coeur

# Fichiers de configuration

Pour un *solr.home* donné (par défaut : *server*) :

Configuration des cœurs :

`<solr.home>/solr.xml`

Données d'un coeur :

`<solr.home>/<core>/data`

Configuration d'un coeur:

`<solr.home>/<core>/conf`

# Configuration gestionnaires HTTP

- L'API REST de Solr est fournie par des « **gestionnaires de requêtes** » (**request Handler**) de différents types :
  - Parseur de requête et définition de la chaîne des composants de recherche
  - Indexeurs
- Ils sont configurés dans  
`<solr.home>/<core>/conf/solrconfig.xml`
- La configuration contient également :
  - Feuilles de style pour transformation de données XML (entrée/sortie)  
`<solr.home>/<core>/conf/xslt`
  - Des gabarits velocity permettant la génération d'une interface HTML (l'interface « browse ») :  
`<solr.home>/<core>/conf/velocity`



# *schema*

- « ***field*** », « ***dynamicField*** », « ***uniqueKey*** » définissent la structure d'un document dans l'index
- « ***copyField*** » duplique automatiquement les valeurs d'un champ dans un autre
- « ***fieldType*** » déclare un type de champs possible pour un « *field* »
  - Contient un ou parfois deux « analyzer »
- « ***analyzer*** » définit les traitements qui seront appliqués aux valeurs du *field*, à l'indexation **et** à la recherche

# *Attributs <field>*

***name*** : son nom;

***type*** : son type, une référence à *fieldType*;

***multivalued*** : si un document peut avoir plusieurs valeurs pour ce champ;

***required*** : si la valeur est obligatoire

(SolR renverra une erreur si on insère un document sans valeur pour ce field)

***indexed*** : si on veut pouvoir chercher ou trier sur les valeurs de ce champs;

***stored*** : si on veut ramener les valeurs de ce champs dans un résultat de recherche;

***docValues***="true" : Si on veut trier ou grouper sur un champ qui par défaut ne le supporte pas

# Schéma et types des champs

- Schéma ~ Définition des champs de l'index  
`<solr.home>/<core>/conf/schema`  
ou  
`<solr.home>/<core>/conf/managed-schema`
  - + tous les autres fichiers de ce répertoire, exemples fichiers de configuration par langue  
`<solr.home>/<core>/conf/lang`

# *fieldType*

***name*** : son nom (référéncé dans un field);

***class***: sa classe Java (1 classe = 1 type de données)

(dans ce fichier, « solr... » est un raccourci pour « *org.apache.solr.analysis* »)

- Quelques types de données :
  - ***solr.BoolField*** : booléen
  - ***solr.TrieIntField***/***solr.TrieLongField*** : entiers
  - ***solr.TrieFloatField*** : décimaux
  - ***solr.TrieDateField*** : date
  - ***solr.LatLonType*** : latitude/longitude
  - ***solr.CurrencyField*** : monnaie
  - ***solr.TextField*** : texte

# Attributs *required* et *multiValued*

- En dehors de son type, les 2 premiers attributs d'un champ sont :
  - ***required*** : Un document doit obligatoirement avoir ce champ renseigné lors de l'indexation
  - ***multiValued*** : Le champ peut contenir plusieurs valeurs (tableau)

# Attributs : *indexed* et *stored*

On peut avoir des champs qui ne servent qu'à rechercher sans jamais être ramenés dans un résultat de recherche : ***indexed***

Inversement, on peut avoir des champs qui ne servent qu'à être ramenés dans un résultat de recherche et sur lesquels on ne cherchera jamais : ***stored***

# Balise *<uniqueKey>*

- La balise *<uniqueKey>* permet de préciser le champ qui sert de clé pour ce schéma

`<uniqueKey>id</uniqueKey>`

- Cette balise n'est pas requise mais recommandée

# Balises *dynamicField*

- La balise ***dynamicField*** permet d'affecter un type à un champ en fonction de son nom

– Ex :

```
<dynamicField name="*_num" type="pdouble" indexed="true"
stored="true" multiValued="false" />
```

=> Tous les champs qui seront terminés par le suffixe `_nim` seront de type `pdouble` ....



# Balises *copyField*

- La balise ***copyField*** permet de dupliquer certaines valeurs dans certains champs afin de pouvoir appliquer différents analyseurs et donc proposer plusieurs types de recherche
  - Ex : copier une chaîne vers un *field*  
« phonétique » sur lequel portera la recherche phonétique

# Champs spéciaux

- Un index comporte généralement les champs suivants :
  - ***id*** : Identifiant du document
  - ***\_\_version*** : Identification de la version du document.  
(Un document est immuable ce champ est incrémenté à chaque mis à jour)
  - ***\_\_root*** : Nécessaire si l'on veut utiliser les nested documents
  - ***text*** : Réceptacle pour tous les champs “searchable “

# Indexation

Principe des analyseurs

API d'indexation

Recommandations pour la configuration

# Analyseurs


# Introduction

- SolR utilise 3 concepts pour traiter le texte des documents :
  - Les **analyseurs** de champs sont utilisés à l'indexation et lors de la recherche. Ils transforment un texte en un flux de “token”. Ils sont en général constitué de :
    - Les **filtres de caractères** effectuant du remplacement de caractères ( & devient et) ou en supprimant (suppression des balises HTML)
    - Un **tokenizer** : Responsable de splitter un texte en token ou terme
    - Les **filtres** prennent en entrée un flux de token et le transforme en un autre flux de token

# Analyseurs définis

- Lors d'une création de coeur, SolR créé par défaut de nombreux types de données associés à des analyseurs dédiés à un usage.
- Les plus utiles sont :
  - ***text\_general*** : Le meilleur choix lorsque le champ est dans des langues diverses. Il consiste à :
    - Séparer le texte en mots
    - Supprime la ponctuation
    - Supprimer certains mots (*stopword*)
    - Passe tous les mots en minuscule
  - ***text\_gen\_sort*** : Idema avec des capacités de tri
  - **Analyseurs de langues** : *text\_en*, *text\_fr*, .... Ce sont des analyseurs spécifiques à la langue. Ils incluent les « stop words » (enlève les mots les plus courant) et extrait la racine d'un mot. C'est le meilleur choix le champ est en une seule langue

# L'interface d'analyse



- Dashboard
- Logging
- Core Admin
- Java Properties
- Thread Dump
- collection1
- Overview
- Ping
- Query
- Schema
- Config
- Replication
- Analysis**
- Schema Browser
- Plugins / Stats
- Dataimport

Field Value (Index)  
Une formation débutant pour SolR.

Field Value (Query)

Analyse Fieldname / FieldType: text\_fr ? ☒ Verbose Output Analyse Values

ST	text	Une	formation	débutant	pour	SolR
	raw_bytes	[55 6e 65]	[66 6f 72 6d 61 74 69 6f 6e]	[64 c3 a9 62 75 74 61 6e 74]	[70 6f 75 72]	[53 6f 6c 52]
	start	0	4	14	23	28
	end	3	13	22	27	32
	type	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>
	position	1	2	3	4	5
EF	text	Une	formation	débutant	pour	SolR
	raw_bytes	[55 6e 65]	[66 6f 72 6d 61 74 69 6f 6e]	[64 c3 a9 62 75 74 61 6e 74]	[70 6f 75 72]	[53 6f 6c 52]
	position	1	2	3	4	5
	start	0	4	14	23	28
	end	3	13	22	27	32
	type	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>	<ALPHANUM>
LCF	text	une	formation	débutant	pour	solr
	raw_bytes	[75 6e 65]	[66 6f 72 6d 61 74 69 6f 6e]	[64 c3 a9 62 75 74 61 6e 74]	[70 6f 75 72]	[73 6f 6c 72]
	position	1	2	3	4	5
	start	0	4	14	23	28
	end	3	13	22	27	32

# **Atelier :**

## *Comprendre l'analyse*

- Testez l'analyse sur la chaîne « Une formation débutant sur SolR »
  - Avec le type de champ *text\_ws*
  - Avec le type de champ *text\_general*
  - Avec le type de champ *text\_fr*
  - Que constatez-vous ?
  - Lisez les commentaires associés à ces types de champs dans *schema.xml*
- Mettez une query dans le champ query, par exemple « se former sur SolR »
  - Que constatez-vous ?
- Avec le type de champ « string »
- Essayez avec :
  - « Administration Apache, l'essentiel »



# Un analyzer

Mission : analyser les valeurs texte, soit au moment de l'insertion d'une valeur, soit au moment de la recherche d'une valeur.

Un *fieldType* de classe *solr.Text* a :

- Un analyzer pour le texte indexé :

`type="index"`

- Un analyzer pour le texte de la query

`type="query"`

Si un seul analyzer est paramétré, il est utilisé pour l'indexation **et** la recherche

# Un tokenizer

Mission : découper la chaîne de caractères en tokens ou termes

Certains caractères d'entrée peuvent être supprimés (ex. Espace, tabulation), d'autres peuvent être remplacés ou ajoutés (ex. abréviation)

Des méta-données sont ajoutées à chaque token (ex. La position)

# Tokenizer : Quelques possibilités

- ***WhitespaceTokenizer***

- Découpe sur les espaces, tabulations, sauts de ligne

```
<tokenizer class="solr.WhitespaceTokenizerFactory"/>
```

- ***StandardTokenizer***

- Espaces et ponctuation. Marche pour toutes langues européennes. A utiliser par défaut.

```
<tokenizer class="solr.StandardTokenizerFactory"/>
```

- ***KeywordTokenizer***

- Aucune tokenization ! Utile pour les valeurs à stocker telles quelles

```
<tokenizer class="solr.KeywordTokenizerFactory"/>
```

- ***PatternTokenizerFactory***

- Découpe en fonction d'une expression régulière

# Un filter

- Mission : prendre un flux de *token* en entrée, retourner un autre flux de token
- Les filtres sont en général chaînés et l'ordre a une importance

# Filtres communs

- ***LowerCaseFilterFactory***
  - Met tout en minuscule. A utiliser quasi-systématiquement, à l'index ET à la query
- ***LengthFilterFactory***
  - Pour ne garder que les tokens d'une certaine taille
- ***PatternReplaceFilterFactory***
  - Pour faire du rechercher-remplacer dans les tokens

# Filter : Elision

- Elision : effacement d'une voyelle.  
Le filtre supprime l'article et l'apostrophe
- Utile pour le français, le catalan, l'italien et l'irlandais

```
<filter class="solr.ElisionFilterFactory"  
ignoreCase="true" articles="lang/contractions_fr.txt"/>
```

– Exemple :

L'histoire de l'art => histoire de art

# Filter : Stopwords

```
<filter class="solr.StopFilterFactory"
ignoreCase="true" words="stopwords.txt"
enablePositionIncrements="true" />
```

- Format du fichier : un terme par ligne

a  
à  
et  
un

- ***solr.KeepWordFilterFactory*** : inverse de *stopWords* (ne garde que les termes spécifiés)

# Filter : Stemming

- Stemming : ramener les formes fléchies à un radical
  - Pluriels, féminins, conjugaisons
  - « cheval », « chevaux » => « cheval »
  - « portera », « porterait » => « porte »
- Plusieurs algorithmes possibles :
  - ***FrenchLightStemFilterFactory*** : Défaut
  - ***FrenchMinimalStemFilterFactory*** : Moins de contraction
  - ***SnowballPorterFilterFactory*** : Plus de contraction



# Filter : Synonyms

```
<filter class="solr.SynonymFilterFactory"  
synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
```

- Le remplacement de synonyme peut se faire de 3 façons :
  - Expansion simple : Si un des termes est rencontré, il est remplacé par tous les synonymes listés  
*"jump,leap,hop"*
  - Contraction simple : un des termes rencontré est remplacé par un synonyme  
*"leap,hop => jump"*
  - Expansion générique : un terme est remplacé par plusieurs synonymes  
*"puppy => puppy,dog,pet"*

# Synonymes :

## index-time ou query-time ?

- Les synonymes peuvent être utilisés au moment de l'indexation ou au moment de la query
- Il est conseillé de les utiliser au moment de l'indexation
  - question de performance
  - problèmes liés aux synonymes comportant plusieurs mots à la query

# Recherche phonétique

```
<filter class="solr.DoubleMetaphoneFilterFactory"  
inject="false"/>
```

- A mettre à l'index ET à la query
- Plusieurs algorithmes possibles
  - *Caverphone*, *Metaphone*, *DoubleMetaphone*, etc.
  - *DoubleMetaphone* donnerait de meilleurs résultats même en dehors de l'anglais
- Attention, peut donner des résultats hasardeux
- A utiliser dans un champ dédié

# Filtres de caractères

- Les filtres de caractères traitent des caractères en entrée, ils peuvent être chaînés comme les filtres de token
- Ils peuvent ajouter, supprimer ou changer des caractères tout en préservant l'offset original des caractères pour supporter la surbrillance
  - ***solr.MappingCharFilterFactory*** : Basé sur un fichier de correspondance
  - ***solr.HTMLStripCharFilterFactory*** : Supprime les balises HTML
  - ***solr.ICUNormalizer2CharFilterFactory*** : Normalisation Unicode avec icu4j
  - ***solr.PatternReplaceCharFilterFactory*** : Utilisation d'expression régulières

# Indexation

# Introduction

- L'indexation consiste à ajouter du contenu à l'index SolR et éventuellement en modifier ou en supprimer
- L'indexation s'effectue via **l'API REST**, cela consiste à poster des données via des requêtes HTTP utilisant les formats XML, JSON ou CSV
- Pour indexer, Apache SolR fournit également
  - l'outil en ligne de commande *post*
  - L'interface d'administration
  - L'API Java *Solrj* et sa classe *SolrClient*

# Plugins

- SolR propose 2 plugins :
  - ***Solr Cell*** basé sur Apache Tika pour ingérer des fichiers binaires ou structurés comme des documents Offices, des PDF ou autres
  - ***DataImportHandler*** pour aspirer du contenu à partir d'un support persistant (BD ou autre)

# Index Handler

- Par défaut, SolR configure un `updateRequestHandler` capable de supporter les formats XML, CSV et JSON

```
<requestHandler name="/update"  
class="solr.UpdateRequestHandler" />
```



# Le format XML : add

```
<add overwrite="true">
  <doc>
    <field name="id">5432a</field>
    <field name="type">Album</field>
    <field name="name">Murder Ballads</field>
    <field name="artist">Nick Cave</field>
    <field name="release_date">2012-07-31T09:40:00Z</field>
  </doc>
  <doc boost="2.0">
    <field name="id">myid</field>
    <field name="type">Album</field>
    <field name="name">Ilo veyou</field>
    <!-- etc. -->
  </doc>
</add>
```

# Le format XML

- ***overwrite***

- Basé sur le champ *uniqueKey*
- Mettre à *false* si on est sûr de n'envoyer que des nouveaux record

- ***boost***

- Le document sortira plus haut dans les résultats que les autres

# Le format XML : delete

```
<delete>  
  <id>5432a</id>  
  <id>monId</id>  
</delete>
```

```
<delete>  
  <query>Artist:"Nick Cave"</query>  
</delete>
```

```
<delete>  
  <query>*:*</query>  
</delete>
```

- Pour supprimer tout l'index, il faut mieux supprimer le répertoire *data* et relancer *So/R*

# Le format XML : commit

```
<commit />
```

```
<rollback />
```

```
<optimize />
```

- Commits :
  - Lents : donc faire un seul gros commit à la fin
  - Pas de transactions par clients (commit global)
- Optimize : committe et optimise l'index
- Aucune de ces opérations ne bloque la recherche
- Un *commit* ou un *optimize* peuvent aussi via la query string : ?commit=true

# Transformation XSL

- En utilisant le paramètre **tr**, il est possible d'appliquer une transformation XSL au document d'origine

Le feuille de style XSLT doit être dans le dossier *conf/xslt*

Exemple :

```
curl
"http://localhost:8983/solr/my_collection/update
?commit=true&tr=updateXml.xsl"
-H "Content-Type: text/xml" --data-binary
@myexporteddata.xml
```

# ***Atelier***

## ***Indexation***

- Utiliser les coeurs précédemment créés :
- Modifier *schema.xml* pour pouvoir indexer les données de test
- Indexer le XML, tester une recherche

# Le format JSON

- Des requêtes au format JSON peuvent également être envoyées au gestionnaire de requête */update*
- Il faut alors préciser le mime-type  
*Content-Type:application/json* ou *Content-Type:text/json*
- Les mises à jour via JSON peuvent prendre 3 formes de base :
  - Un unique document à ajouter  
le paramètre *json.command=false* est alors nécessaire.
  - Une liste de documents à ajouter (JSON Array)
  - Une séquence de commandes de mises à jour

# URLs JSON

- En plus du gestionnaire de requête */update*, il existe des gestionnaires spécifiques JSON qui surcharge le comportement des paramètres de requêtes

`/update/json => stream.contentType=application/json`

`/update/json/docs => stream.contentType=application/json  
json.command=false`

- Il est également possible d'appliquer des transformations sur les documents JSON d'origine. Cela s'effectue alors avec des paramètres spécifiques.



# Curl JSON : Exemples

## # Indexation listes de document

```
curl -X POST -H 'Content-Type: application/json'
'http://localhost:8983/solr/my_collection/update' --data-binary '
[
{
  "id": "1",
  "title": "Doc 1"
},
{
  "id": "2",
  "title": "Doc 2"
}
]'
```

## # Indexation commande

```
curl -X POST -H 'Content-Type: application/json'
'http://localhost:8983/solr/my_collection/update' --data-binary '
{
  "delete": { "id":"ID" },
  "delete": { "query":"QUERY" }
/* delete by ID */
/* delete by query */
}'
```

# Curl JSON : Transformation

```
curl 'http://localhost:8983/solr/my_collection/update/json/docs'\
'?split=/exams'\
'&f=first:/first'\
'&f=last:/last'\
'&f=grade:/grade'\
'&f=subject:/exams/subject'\
'&f=test:/exams/test'\
'&f=marks:/exams/marks'\
-H 'Content-type:application/json' -d '
{
  "first": "John",
  "last": "Doe",
  "grade": 8,
  "exams": [
    {
      "subject": "Maths",
      "test"
      : "term1",
      "marks" : 90},
    {
      "subject": "Biology",
      "test"
      : "term1",
      "marks" : 86}
  ]
}'
```

# ***Atelier :***

## Indexation JSON

- Utiliser les cœurs précédemment créés pour ajouter un document au format JSON
- Tester la recherche

# Le format CSV

- Des requêtes au format CSV peuvent également être envoyées au gestionnaire de requête */update*
- Il faut alors préciser le mime-type :  
*Content-Type: application/csv* ou  
*Content-Type: text/csv*
- On peut également utiliser l'URL :  
*/update/csv => stream.contentType=application/csv*
- Les noms des colonnes du CSV doivent correspondre au noms des *fields*
- Ce format est le seul qui soit le même en input et en output de query
- On pourrait donc faire une query dans un *SoIR* en demandant un résultat CSV et réinjecter ce résultat dans un autre *SoIR*

# Paramètres d'une requête CSV

Paramètres possibles de l'URL :

*separator* : séparateur à utiliser (',' par défaut)

*header=true* : indique que la première ligne est une ligne d'entête

*fieldnames=field1,field2* : indique le nom des fields à utiliser si le CSV ne contient pas d'entête

*overwrite=false* : si on n'est sûr que l'on n'overwrite rien

*skipLines=1000* : si on veut sauter des lignes

*skip=field1,field2* : si certaines colonnes ne doivent pas être importées

*escape=\\* caractère d'échappement pour échapper le séparateur dans les valeurs

*encapsulator="* : indique le caractère qui entoure les valeurs de champs qui contiennent le séparateur

# Near Real Time

- Lorsque la cadence d'indexation est élevée et que l'on veut que les documents soient rapidement disponibles à la recherche, on configure un soft commit
- Le recherchabilité d'un document est contrôlée par les commits
  - **soft** : Le document est visible mais pas stocké sur disque. Si le serveur crash, il faudra rejouer la transaction (*tlog*)
  - **hard** : Le document est stocké sur le disque. La transaction correspondante ne fait plus partie du journal des transactions
- La cadence des commits soft et hard est configurable dans *solrconfig.xml*

# Exemple de configuration

```
<!-- Configuration des hardCommit tous les 60 secondes -->
```

```
<autoCommit>
```

```
<maxTime>${solr.autoCommit.maxTime:60000}</maxTime>
```

```
<openSearcher>>false</openSearcher>
```

```
</autoCommit>
```

```
<!-- Configuration des softCommit tous les 30 secondes -->
```

```
<autoSoftCommit>
```

```
<maxTime>${solr.autoSoftCommit.maxTime:30000}</maxTime>
```

```
</autoSoftCommit>
```

# Nested documents

- *SolR* supporte les “nested documents” permettant de modéliser des relations 1-N entre documents
- L’indexation se fait par bloc : il faut fournir le document parent et ses documents enfants en même temps (même si un seul enfant a été modifié !)



# Règles sur les nested

- Le schema doit inclure le champ *\_root\_* ('indexé et non stocké). Le valeur du champ est identique pour tous les documents du bloc (indépendamment de sa profondeur dans le graphe)
- Certaines contraintes sur les documents imbriqués :
  - Le schéma doit spécifier leurs champs
  - Impossible d'utiliser *required* sur le champ des enfants
  - Nécessite un *id* unique
- Un champ doit permettre d'identifier un document parent.
- Si le document enfant est associé à un champ, celui ne doit pas être défini dans le schéma. Il n'y a pas de type "child"

# Exemple XML

```
<add>
  <doc> <!-- Les documents d'id 1 et 2 ont la même valeur dans le champ _root_ -->
    <field name="id">1</field>
    <field name="title">Solr adds block join support</field>
    <field name="content_type">parentDocument</field> <!-- Champ identifiant un document parent -->
    <field name="content"> <!-- Le champ n'est pas défini dans le schéma -->
      <doc>
        <field name="id">2</field>
        <field name="comments_txt_en">SolrCloud supports it too!</field>
      </doc>
    </field>
  </doc>
  <doc> <!-- Les documents d'id 3 et 4 ont la même valeur dans le champ _root_ -->
    <field name="id">3</field>
    <field name="title">New Lucene and Solr release is out</field>
    <field name="content_type">parentDocument</field>
    <doc>
      <field name="id">4</field>
      <field name="comments">Lots of new features</field>
    </doc>
  </doc>
</add>
```

# Update Request Processor

# Introduction

- Toutes les requêtes de mise à jour sont traitées par une chaîne de *plugins* : les ***Update Request Processor***
- Ils peuvent être utilisés pour ajouter un champ au document, changer sa valeur ou pour éviter une mise à jour si certaines conditions ne sont pas respectées

# Chaîne

- Un *Update Request Processor* fait partie d'une **chaîne**
- Une chaîne par défaut est fournie par SolR et il est possible de configurer sa propre chaîne
- Un *UpdateRequestProcessor*
  - dans ses conditions normales, appelle l'*UpdateRequestProcessor* suivant après son traitement
  - Dans des conditions d'exception, il peut interrompre le traitement et éviter la mise à jour de l'index

# Chaîne par défaut

- La chaîne par défaut comprend :
  - ***LogUpdateProcessorFactory*** : Trace les commandes d'update lors de la requête
  - ***DistributedUpdateProcessorFactory*** : Responsable de distribuer les requêtes de mise à jour au bon noeud dans le cas d'un cloud
  - ***RunUpdateProcessorFactory*** : Exécute les mises à jour en utilisant l'API interne de SolR

# Configuration d'une chaîne spécifique

```
<!-- Une chaîne incluant un filtre évitant les duplications -->
<updateRequestProcessorChain name="dedupe">
  <processor class="solr.processor.SignatureUpdateProcessorFactory">
    <bool name="enabled">true</bool>
    <str name="signatureField">id</str>
    <bool name="overwriteDups">false</bool>
    <str name="fields">name,features,cat</str>
    <str name="signatureClass">solr.processor.Lookup3Signature
  </str>
</processor>
<processor class="solr.LogUpdateProcessorFactory" />
<processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

# Configuration d'une chaîne spécifique (2)

```
<updateProcessor
class="solr.processor.SignatureUpdateProcessorFactory"
name="signature">
  <bool name="enabled">true</bool>
  <str name="signatureField">id</str>
  <bool name="overwriteDups">false</bool>
  <str name="fields">name,features,cat</str>
  <str name="signatureClass">solr.processor.Lookup3Signature</str>
</updateProcessor>
<updateProcessor
class="solr.RemoveBlankFieldUpdateProcessorFactory"
name="remove_blanks"/>
...
<updateProcessorChain name="custom"
processor="remove_blanks,signature">
  <processor class="solr.RunUpdateProcessorFactory" />
</updateProcessorChain>
```



# Utilisation des chaînes

- Le paramètre **update.chain** permet d'indiquer la chaîne que l'on veut utiliser lors d'une requête
- On peut également créer une chaîne à la volée en indiquant le paramètre **processor** est la liste des processeurs à utiliser

`?processor=remove_blanks,signature&commit=true`

# Quelques Processeurs disponibles

- Ajout automatique de champ
- Valeur par défaut pour un champ
- Ajout automatique d'un champ *timestamp*
- Traitement de date d'expiration
- Attribution d'une valeur de *boost* en fonction de la valeur d'un champ ... et d'une regexp
- Calcul d'une signature pour éviter la duplication
- Concaténation automatique de champs
- Suppression de balises HTML
- Détection de langues
- ....

# Exemple détection de langue

- SolR peut tenter de détecter la langue d'un champ et de l'associer ensuite à un champ dédié à une langue. Le processeur s'appelle *langid* et SolR fournit 2 implémentations :
  - **Tika**
  - **LangDetect** (qui apparemment donne de meilleurs résultats actuellement)
  -

# Configuration

- La configuration minimale consiste à spécifier les champs pour l'identification et un champ pour stocker le résultat (un code langue)

```
<processor
class="org.apache.solr.update.processor.LangDetectLanguage
IdentifierUpdateProcessorFactory">
<lst name="defaults">
<str name="langid.fl">title,subject,text,keywords</str>
<str name="langid.langField">language_s</str>
</lst>
</processor>
.
```

# Exemple JSON

```
[
  {
    "id": "1",
    "title": "Solr adds block join support",
    "content_type": "parentDocument",
    "comment": {
      "id": "2",
      "comments": "SolrCloud supports it too!"
    }
  },
  {
    "id": "3",
    "title": "New Lucene and Solr release is out",
    "content_type": "parentDocument",
    "_childDocuments_": [
      {
        "id": "4",
        "comments": "Lots of new features"
      }
    ]
  }
]
```

# *Atelier*

## *Nested Documents*

# Optimisation index

- Minimiser l'index en fonction des cas d'utilisation de recherche
  - index=false, pour les champs non utilisés pour la recherche
  - copyField pour rassembler dans un seul champ les champs texte utilisé pour la recherche

# Optimisation Indexation

- L'optimisation du temps d'indexation se fait généralement
  - En augmentant la taille du batch (nombre de documents en 1 requête d'update)
  - En multipliant le nombre de threads effectuant le travail d'indexation



# Recherches

Query parser

Rappels sur la syntaxe Lucene

Recherche par facette

Recherche par groupe

Relation parent/child, Jointure

Recherche spatiale

# Introduction

- Solr propose un mécanisme de recherche très flexible
- Une requête de recherche est traitée par un ***RequestHandler*** (défini dans la configuration SolR)
- Le *RequestHandler* fait appel à un ***Query parser*** qui prend en général des paramètres d'entrée :
  - La chaîne à chercher
  - Des paramètres de tuning de la requête
  - Des paramètres contrôlant la présentation de la réponse
- Les parseurs ont en commun certains paramètres d'entrée, d'autres leur sont spécifiques

# Query Parsers

- Les parseurs les plus courants sont :
  - ***StandardQueryParser*** : Extension du parseur Lucene. Très puissant mais syntaxe compliquée et peu tolérante
  - ***DisMaxParser*** : Fait pour traiter de simples phrases directement saisies par l'utilisateur. Effectue la recherche sur différents champs qui ont différents poids (boosts)
  - ***ExtendedDisMax*** : Ajoute des fonctionnalités avancées à *DisMax*

# Cache, Réponse

- Les paramètres de recherche peuvent également spécifier un **query filter** qui met en cache les résultats et permet d'améliorer les performances
- Une requête de recherche peut demander la **surbrillance** de certains termes
- Les réponses peuvent également inclure des document **snippets** (extrait du document)

# Regroupement des résultats

- SolR permet également de regrouper les résultats de recherche de 2 façons, facilitant l'exploration de données :
  - Les **facettes** groupent les résultats de la recherche dans des catégories (qui sont basées sur les termes indexés).
  - Le **clustering** groupent les résultats selon des similarités découvertes algorithmiquement lors de la recherche
- SolR supporte des requêtes de type **MoreLikeThis** qui se basent sur des termes retournés par une requête précédente

# Présentation des résultats

- Les composants Solr de type ***Response Writer*** gèrent la présentation finale de la réponse.
- Solr fournit différents *Response Writers* comme :
  - *XML Response Writer*
  - *JSON Response Writer*
  - *Velocity Response Writer*

# Pertinence

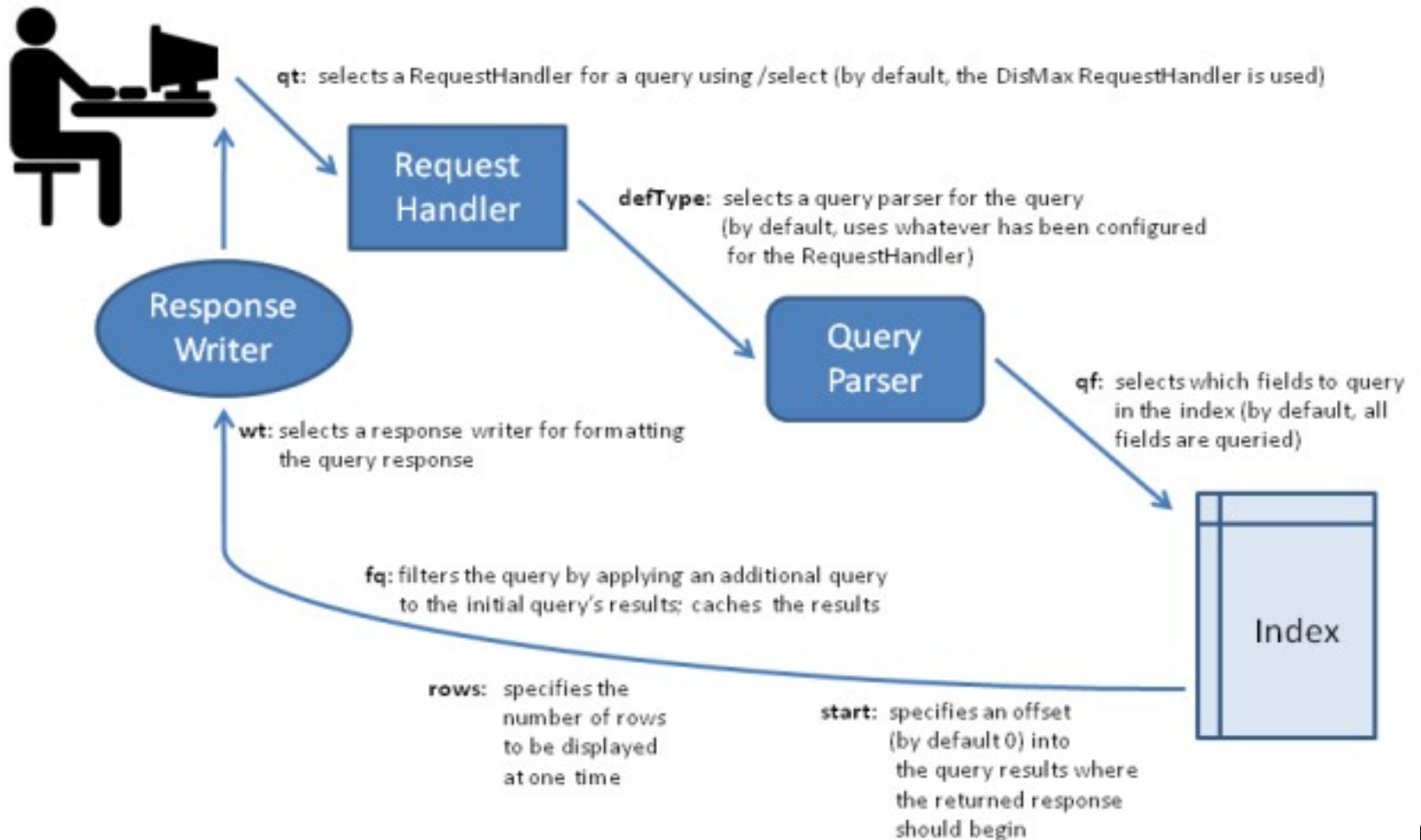
- La pertinence mesure l'adéquation de la réponse à la requête
- Elle est fortement dépendante du contexte de la requête
- Il est souvent utile dans les étapes de préparation au déploiement de spécifier les réponses que l'application doit retourner pour des exemples de requêtes.
- Il est alors possible de jouer sur la configuration pour obtenir les résultats voulus

# Réponses

```
{
  "responseHeader":{
    "status":0, // Code retour
    "Qtime":0,  // Temps d'exécution
    "params":{
      "q":"_root_:1",
      "_":"1542533170700"}},
  "response":{"numFound":2,"start":0,"docs":[ // Total et position de départ
    {
      "id":"2",
      "comments_txt":["SolrCloud supports it too!"],
      "_version_":1617463248575004672},
    {
      "id":"1",
      "title":["Solr adds block join support"],
      "content_type":["parentDocument"],
      "_version_":1617463248575004672}]
  }}
```



# Big Picture



# Interface utilisateur exemple

- *SolR* fournit une interface exemple basée sur *VelocityResponseWriter* qui démontre certaines fonctionnalités de SolR (recherche, facette, surbrillance, autocomplete et recherche spatiale)

*<http://localhost:8983/solr/techproducts/browse>*

# *RequestHandler*

- Configure un type de recherche (ou d'update)
  - Indique des paramètres par défaut
  - Indique quels composants de recherche utiliser (*<searchComponent/>*)

*=> Recommandation : Configurer un RequestHandler par type de recherche dans votre application*

- Chercher le *RequestHandler* « */browse* » de l'exemple *techproducts* pour voir sa longue liste de paramètres

# Appeler un *RequestHandler*

- Il y a 2 façons d'appeler un *RequestHandler* spécifique :
  1. Soit à l'URL */select* avec le paramètre « **qt** » (pour « *queryType* ») contenant le nom du requestHandler

<http://localhost:8389/core01/select/?q=primaire&qt=myRequestHandler>

2. Soit en appelant l'URL correspondant au nom du *RequestHandler*

<http://localhost:8389/core01/myRequestHandler?q=primaire>

# Handler par défaut

- */select* : Handler générique permettant de dispatcher via le paramètre *qType*
- */query* : Format JSON identé
- */browse* : Interface par défaut pour parcourir la collection exemple techproducts. (Velocity)

# Paramètres de recherche communs

- La query :

q (query)	query de recherche
fq (filterQuery)	queries de filtrage (~ WHERE SQL)
defType	le parser de query (lucene ou edismax)

- Pagination

rows	Nombre de résultats
start	Offset de départ de la liste

- Output :

fl (fieldList)	Champs à remonter
sort	Critère de tri (la pertinence en général)
wt (writer type)	Format de la réponse

- Diagnostic :

indent	Indentation du résultat
explainOther	ce que SolR a compris de la recherche
debug	votre ami pour le tuning

# *RequestHandler* : liste « defaults »

- La liste « **defaults** » donne les valeurs des paramètres qui seront utilisés si aucune valeur n'est précisée explicitement dans la query

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <lst name="defaults">  
    <str name="echoParams">explicit</str>  
    <!-- etc... -->  
  </lst>  
  <!-- ... -->  
</requestHandler>
```

# RequestHandler : liste « *appends* »

- La liste « ***appends*** » donne les valeurs des paramètres qui seront ajoutées aux paramètres multivalués de la query (comme *fq*)

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <lst name="appends">  
    <str name="fq">a_type:group</str>  
    <!-- etc... -->  
  </lst>  
  <!-- ... -->  
</requestHandler>
```



# RequestHandler : liste « *invariants* »

- La liste « ***invariants*** » donne les valeurs des paramètres qui seront toujours utilisés, quelque soit les valeurs indiquées dans la requête (utile pour sécuriser les *requestHandler*)

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <lst name="invariants">  
    <str name="facets">false</str>  
    <!-- etc... -->  
  </lst>  
  <!-- ... -->  
</requestHandler>
```

# Les « composants » de recherche

- Un « ***search component*** » exécute une fonctionnalité de recherche : *highlight*, *facettes*, *MLT*, etc.
- Chaque « *component* » est déclaré dans une section de *solrconfig.xml*

```
<searchComponent class="solr.HighlightComponent" name="highlight">  
  <highlighting>  
    <!-- etc... -->  
  </highlighting>  
</searchComponent>
```

# *SearchComponents* par défaut

- Les composants de recherche s'appliquant par défaut sont dans l'ordre :
- ***query*** : Parser.
- ***facet*** : Facette
- ***Mlt*** : MoreLikeThis.
- ***highlight*** : Surbrillance
- ***stats*** : Génère des statistiques
- ***debug*** : Debug
- ***expand*** : Gestion des groupes

# RequestHandler : array

## « components »

- Un *RequestHandler* peut redéfinir la liste des composants de recherche par défaut via une balise **array nommée « components »**

```
<requestHandler name="/myHandler" class="solr.SearchHandler">
  <arr name="components">
    <!-- This is default components ! -->
    <str>query</str>
    <str>facet</str>
    <str>mlt</str>
    <str>highlight</str>
    <str>stats</str>
    <str>debug</str>
  </arr>
</requestHandler>
```

# « first-components » et « last-components »

- On peut ajouter un composant de recherche au début ou à la fin de la liste par défaut avec « **first-components** » et « **last-components** »

```
<requestHandler name="/myHandler" class="solr.SearchHandler">  
  <arr name="last-components">  
    <str>elevator</str>  
  </arr>  
</requestHandler>
```

# Atelier

## Configuration request handler

- Reparamétrer le handler */browse*
- Commentez toutes les parties non nécessaires :
  - Mlt
  - Facets
  - highlight
  - Spellcheck
  - Conserver seulement les parties « *VelocityResponseWriter* » et « *Query settings* »
  - Ajuster le paramètre *qf*
- Copier les nouveaux templates Velocity dans le sous-répertoire « *velocity* » pour correspondre à notre schéma

# Le score et le tri

- Champ spécial « *score* »
  - on peut le ramener avec le paramètre « ***fl=score*** »
- « *sort* » se fait sur ce pseudo-champ « *score* » par défaut
- Tester avec *sort=titre\_en* pour trier par ordre alphabétique des titres

# Calcul du score

- Les facteurs influant le score sont :
  - **tf** (*term frequency*): La fréquence du terme dans le document
  - **idf** (*inverse document frequency*): La fréquence du terme dans l'index
  - **coord** : Le nombre de termes de la requête trouvés dans le document
  - **lengthNorm** : L'importance du terme par rapport au nombre total de terme pour ce champ
  - **queryNorm** : Facteur de normalisation permettant de comparer les requêtes
  - **boost** (index) : Le boost du champ au moment de l'indexation
  - **boost** (query) : Le boost du champ au moment de la requête



# Implémentation et implication

- *tf* :  $\text{sqrt}(\text{freq}) \Rightarrow$  Plus le terme apparaît dans le document plus le score est élevé
- *idf* :  $\log(\text{numDocs} / (\text{docFreq} + 1)) + 1 \Rightarrow$  Plus le terme est présent dans l'index moins le score est élevé
- *coord* :  $\text{overlap} / \text{maxOverlap} \Rightarrow$  Plus il y a de termes plus le score est élevé
- *lengthNorm* :  $1 / \text{sqrt}(\text{numTerms}) \Rightarrow$  Moins il y a de termes pour ce champ plus le score est élevé

# Fonctions

- Les fonctions permettent de générer un score de pertinence à partir de la valeur de champs numériques :
- Les fonctions peuvent être :
  - Une constante (string ou nombre)
  - Un champ
  - Une autre fonction
- Elles permettent de changer le rang d'un résultat à partir d'un calcul et influe donc l'ordre de présentation des résultats

# Usage

- Il y a plusieurs façons d'utiliser les fonctions dans une requête SolR:

- En utilisant un parseur spécifique

```
q={!func}div(popularity,price)&fq={!frange  
l=1000}customer_ratings
```

- Dans une expression de tri

```
sort=div(popularity,price) desc, score desc
```

- Dans une expression d'un pseudo champs

```
&fl=sum(x, y),id,a,b,c,score
```

- Dans un paramètre qui le supporte (ex:  
boost de EDisMax ou bf DisMax )

```
q=dismax&bf="ord(popularity)^0.5  
recip(rord(price),1,1000,1000)^0.3"
```

# Syntaxe de recherche

- La syntaxe dépend du parser de query (paramètre *defType*)
  - **Standard** (Lucene)
  - Ou **dismax** ou **edismax**
- Le parser « lucene » est le plus précis et permet de combiner des critères sur tous les champs
  - mais une « search box » dans une interface est généralement branchée sur un *defType=edismax* qui est plus simple pour du plein-texte

# Syntaxe Lucene : indication des champs

- **\*:\*** matche tous les documents
- Indiquer un champ spécifique :  
***pays:France***

# Syntaxe Lucene : clauses

- Une recherche est un ensemble de clauses :
  - education*** : clause optionnelle
  - +education*** : clause obligatoire
  - education*** : clause interdite
- Combiner les clauses :
  - AND / && : ***+education AND pays:france***
  - OR / || : ***+education OR pays:france***
- Parenthèses
  - (education AND teacher) OR (quality AND plan)***
  - (+education +teacher) (+quality +plan)***
  - différent de* ***+(education teacher) +(quality plan)***

# Syntaxe Lucene : phrase et wildcard

- « Phrase query » :  
***quality education*** vs. ***“quality education”***
- Proximité des mots pour une phrase  
***“ quality education ”~3***
- Wildcard queries  
***educat\****  
***In\*tion -innovation***  
***Innova????***  
***\*tion***

# Syntaxe Lucene : fuzzy, range, boost

- Recherches floues (fuzzy, correction de typo)

***auteur:ewing*** vs. ***auteur:ewing~*** vs. ***auteur:ewing~0.8***

- Range queries

***education AND annee:[1999 TO 2001]***

***education AND annee:[2001 TO \*]***

- Score boosting (joue sur la pertinence)

***child primary^10***



# Mr « Eddy Smax » (edismax)

- edismax vs. lucene :
  - Fait pour traiter des phrases simples
  - Supporte un sous-ensemble simplifié de la syntaxe Lucene
  - Recherche dans plusieurs champs en même temps avec différents poids
  - Query par défaut
  - + Nombre minimum de mots à matcher
  - + Smart boosting (proximity)
- Activer edismax avec le paramètre
  - *defType*=« *edismax* »

# Edismax : paramètres

- **qf** : query fields

*children education*

*titre\_en^10 resume^2 vs. titre\_en^2  
resume^10*

- **q.alt** : alternative query si *q* n'est pas précisé

– Souvent *\*:\**

- **mm** : minimum de clauses devant matchée

- Voir

[http://wiki.apache.org/solr/DisMaxQParserPlugin#mm\\_.28Minimum\\_.27Should.27\\_Match.29](http://wiki.apache.org/solr/DisMaxQParserPlugin#mm_.28Minimum_.27Should.27_Match.29)

# Bloc Join Query Parsers

- Il y a 2 parseurs qui supportent les jointures sur les blocs de documents (i.e. nested documents, parent/children)
  - ***Block Join Children Query Parser*** : Critère sur le parent et retourne les documents enfants
  - ***Block Join Parent Query Parser*** : Critère sur les enfants et retourne des documents parent

# Children

- La syntaxe est :  
*q={!child of=<allParents>}<someParents>*
  - ***allParents*** est un critère qui retourne tous les parents. (On utilise en général le champ qui identifie un document parent)
  - ***someParents*** : critère sur les parents
- Exemple :  
`q={!child of="content_type:parentDocument"}title:lucene&wt=xml`

# Parent

- La syntaxe est :

`q={!parent which=<allParents>}<someChildren>`

- ***allParents*** est un critère qui retourne tous les parents. (On utilise en général le champ qui identifie un document parent)
- ***someChildren*** : critères sur les enfants

- Exemple :

`q={!parent  
which="content_type:parentDocument"}comments:SolrCloud&wt=xml`

# Boolean Query

- **BqueryParser** permet de combiner plusieurs requêtes via des opérateurs qui influe sur le type de requête effectué et comment le score de pertinence est calculé :
  - **must** : Une liste de requêtes que les documents doivent matcher, les requêtes contribuent au score.
  - **must\_not** : Une liste de requêtes que les documents ne doivent pas matcher, ne contribuent pas au score
  - **should** :
    - Si pas de requêtes must : Les documents retournés doivent matcher au moins une requête should
    - Sinon, ne fait qu'augmenter le score.
  - **filter** : Les documents doivent matcher mais les requêtes ne contribuent pas au score
- Exemples :

```
{!bool must=foo must=bar}  
{!bool filter=foo should=bar}
```

# Join Query

- Le **JoinQParser** permet d'exprimer des jointures entre documents.
- Il prend les paramètres :
  - **from** : La “clé étrangère”
  - **to** : La “clé primaire”
- Exemple :

```
q={!join from=manu_id_s to=id}ipod
```
- La jointure peut également s'effectuer entre deux collections, si on a fait attention aux shards:

```
fq={!join from=region_s to=region_s  
fromIndex=people}mgr_s:yes
```

# Recherches par facettes



# Les facettes

- Les facettes enrichissent un résultat de recherche avec des informations **agrégées** sur la liste de résultats :  
*Similaire à un « GROUP BY » en SQL, fait automatiquement lors d'une recherche;*

# Atelier

## Les facettes

- Dans l'interface d'admin, cocher « *facet* »  
Spécifier « année » dans « *facet.field* » et lancer la recherche
  - Regarder la partie « *facet\_counts* » à la fin du résultat de recherche;
- Combiner avec une recherche texte;
- spécifier « motcle » à la place et relancer la recherche
  - Regarder la partie « *facet\_counts* » à la fin du résultat de recherche;
  - Que constatez-vous ?

# Plusieurs types de facettes

- `facet_fields` : Facette sur valeur de champs (les plus communes)
  - Pour chaque valeur d'un champ, compte le nombre de résultats qui ont cette valeur;
  - « combien ai-je d'items par pays ? Par mot-clé ? »
- `facet_ranges` : Facette sur plage de valeur numérique ou date
  - Pour une liste de plages, compte le nombre de résultats dans chaque plage;
  - « combien d'items font moins de 100 euros, entre 100 et 200 euros, 200 et 300 euros, plus de 300 euros ? »
- `facet_queries` : Facette sur des query dynamiques
  - Pour une liste de queries, compte le nombre de résultats pour chaque requêtes;

# Les facettes : contraintes

- Un champ facetté doit être indexé  
« `indexed=true` »
- Un champ facetté ne doit pas être tokenisé (en général) => Type string sans analyse, entier, date, booléen

# Paramètres de la requête

- `facet=true`
  - Active les facettes
- `facet.field=<nom_du_champ>`
  - Donne le nom d'un champ sur lequel on veut facetter
  - On peut **répéter** ce paramètre plusieurs fois pour facetter sur plusieurs champs (ce que ne permet pas l'interface d'admin)

# Les facettes : paramètres

- Tous les paramètres suivants peuvent être mis :
  - Soit tels quels pour s'appliquer à toutes les facettes;
  - Soit préfixés par « `f.<nom_du_champ>.<parametre>` » pour s'appliquer à une seule facette; c'est préférable.
- `facet.sort=count` ou `facet.sort=index`
  - Tri la liste de valeurs par nombre de documents associés (défaut) ou par ordre alphabétique
  - Exemple : `f.motcle.facet.sort=index`
- `facet.limit=100`
  - Nombre maxi de valeurs pour une facette (défaut : 100)
  - Exemple : `f.motcle.facet.limit=10`
- `facet.mincount=1`
  - Nombre mini de résultats associés à une valeur pour que celle-ci s'affiche
  - Par défaut, `mincount=0`, ce qui veut dire que même les valeurs sans résultat s'affichent
  - Exemple : `f.annee.facet.mincount=1`

# Les facettes : paramètres

- `facet.missing=on`
  - Inclut une valeur vide supplémentaire pour compter les résultats qui n'ont pas de valeur pour ce champ
  - Exemple : `f.motcle.facet.missing=on`
- `facet.offset=10`
  - Offset de départ dans la liste des valeurs (à combiner avec `facet.limit` pour paginer dans les valeurs de facettes)
  - Exemple : `f.motcle.facet.offset=10`
- `facet.prefix=<chaine>` (advanced)
  - Ne garde que les valeurs de facettes qui commencent par la chaine indiquée
  - Utilisée pour des facettes hiérarchiques ou de l'autocomplétion

# Atelier

## Les facettes

- Modifier *schema.xml* pour permettre une recherche à facette sur les pays.
  - Que faut-il ajouter ?
  - Réindexer les données après modification
  - Tester une query avec facette sur les pays pour valider
- Modifier *solrconfig.xml* pour ajouter les facettes « annee » et « motcle » au handler « /browse »
  - Tester en naviguant à <http://localhost:8983/solr/formation/browse>
  - Ajuster les paramètres de chaque facette : limiter le nombre de mot-clés, ne pas afficher les facettes avec 0 résultats, trier la facette année par ordre alphabétique;



# Les facettes ranges : paramètres

- `facet.range=<nom_du_champ>`
  - Fait une facette range sur un champ
- Tous les paramètres suivants peuvent être mis :
  - Soit tels quels pour s'appliquer à toutes les facettes;
  - Soit préfixés par  
« `f.<nom_du_champ>.<parametre>` » pour s'appliquer à une seule facette; c'est préférable.

# Les facettes ranges : paramètres

- `facet.range.start=<valeur>`
  - Valeur de début de la facette
- `facet.range.end=<valeur>`
  - Valeur de fin de la facette
- `facet.range.gap=<valeur>`
  - Le pas de l'itération pour chaque range (10 en 10, 5 en 5, etc.)
- `facet.range.hardend=true`
  - Tronque le dernier range si `facet.range.gap` va au-delà de `facet.range.end` (défaut : `false`)
- `facet.range.other=all|none|before|after|between`
  - Inclut des valeurs supplémentaires pour compte le nombre de résultats avant/après/dans la place spécifiée; « none » (par défaut) ne compte rien de plus, « all » compte avant/après/dans la place en même temps.

# Les facettes : pratique

- Modifier solrconfig.xml pour remplacer la facette « annee » par une facette range de 5 ans en 5 ans
  - Tester en naviguant à <http://localhost:8983/solr/formation/browse>

```
<str name="facet.range">annee</str>
<int name="f.annee.facet.range.start">2000</int>
<int name="f.annee.facet.range.end">2014</int>
<int name="f.annee.facet.range.gap">5</int>
<str name="f.annee.facet.range.hardend">true</str>
<str name="f.annee.facet.range.other">all</str>
```

# Regroupement

# Introduction

- Le **groupement de résultat** regroupe les documents ayant en commun la valeur d'un champ et retourne les meilleurs documents pour chaque groupe
- La fonctionnalité ***Collapse et Expand*** de SolR est plus récente que le grouping et offre de meilleure performance. Elle est donc préférée au regroupement.

# Composants

- La fonctionnalité de regroupement est basée sur 2 composants :
  - Le parseur de requête ***Collapsing*** qui groupe le document en fonction des paramètres fournis
  - Le composant ***Expand*** qui fournit un accès aux documents d'un groupe
- Attention : Avec SolrCloud, les documents doivent être sur le même shard

# Paramètres

- **field** : Le champ de regroupement. Type String, Int ou Float.
- **min** ou **max** ou **sort** : Permet de sélectionner les documents d'entête via la valeur *min* ou *max* du champ, d'une fonction ou d'un critère de tri. Si aucun de ces paramètres n'est spécifié, les documents d'entêtes sont sélectionnés en fonction du score
- **nullPolicy** :
  - **ignore** (défaut): ignore les documents ayant le champ à null
  - **expand** : crée un groupe pour chaque document ayant le champ à null,
  - **collapse** : crée un seul groupe pour tous les documents ayant le champ à null .

# Exemples

- Sélection du document avec le meilleur score dans chaque groupe

```
fq={!collapse field=group_field}
```

- Sélection du document ayant la valeur minimale dans chaque groupe :

```
fq={!collapse field=group_field  
min=numeric_field}
```



# Expand

- Le paramètre ***expand*** utilisé avec une query collapse permet de visualiser les documents des groupes
- Des paramètres additionnels peuvent être précisés :
  - ***expand.sort*** : Le tri utilisé à l'intérieur d'un groupe (par défaut le score).
  - ***expand.rows*** : Le nombre de documents à l'intérieur d'un groupe à retourner (Par défaut 5)

# Recherche géographique

# Introduction

- Solr a du support pour des recherches géographiques.
- Il permet de :
  - Indexer des points ou des formes
  - Filtrer des résultats via des distance ou des formes
  - Trier ou influencer le score via la distance ou la surface d'intersection entre 2 formes
  - Générer des données pour le tracing de points ou des agrégations thermiques sur une carte.

# Types de données

- Il y a 3 principaux types de données liés à la géoloc. :
  - ***LatLonPointSpatialField*** : Le plus commun, représente un couple latitude, longitude
  - ***SpatialRecursivePrefixTreeFieldType*** (RPT), incluant *RptWithGeometrySpatialField* . Formation GeoJSON et autres
  - ***BBoxField*** : Permettant de stocker des formes

# Indexation

- Pour indexer des points géographiques

- Schéma :

- ```
<fieldType name="location"
class="solr.LatLonPointSpatialField" docValues="true"/>
```

- Indexation :

- fournir la latitude, longitude séparé des virgules
    - Utiliser le paramètre *format* et fournir les coordonnées en :
      - WKT
      - GeoJSON

# Filtres

- Lucene SolR propose 2 filtres spatiaux permettant de filtrer des documents en fonction de leurs coordonnées géographiques :
  - **geofilt** retourne les documents à partir de leur distance à un point d'origine  
I.e les documents dont les coordonnées géographiques sont inclus dans un cercle autour d'un point d'origine
  - **bbox** retourne les documents dont les coordonnées géographiques sont inclus dans un carré autour d'un point d'origine

# Recherche spatiale

- Les paramètres pour ces 2 filtres sont :
  - **d** : la distance (unité par défaut : km, ou précisée par la configuration *distanceUnits*).
  - **pt** : Le point central avec le format "lat,lon"
  - **sfield** : Le champ spatial.
  - **score** : Indique le mode de calcul du score. Par exemple : *kilometers* ou *overlapRatio*
  - **filter** : Si false, la requête ne filtre pas mais est utilisée pour le calcul du score

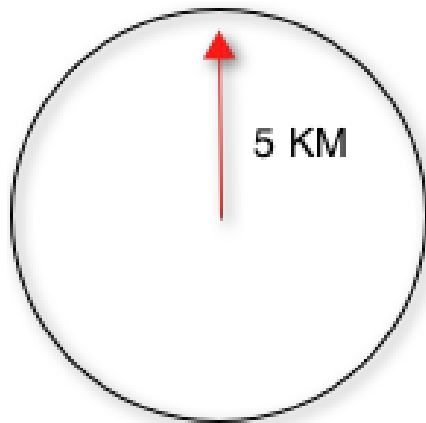
# Exemples

**# Distance de 5km par rapport à un point central**

```
&q=*&fq={!geofilt sfield=store}&pt=45.15,-93.85&d=5
```

**# Carré autour d'un point d'origine (+rapide)**

```
&q=*&fq={!geofilt sfield=store}&pt=45.15,-93.85&d=5
```





# Rectangle arbitraire

- Il est également possible de filtrer les documents par un rectangle arbitraire en utilisant une *range query*.
  - La première valeur correspond aux coordonnées du coin haut-gauche
  - La seconde au coin bas-droit

```
&q=*:*&fq=store:[45,-94 TO 46,-93]
```

# Tri et fonctions

- Il y a 4 fonctions se basant sur les distances, la plus appropriée est ***geodist***

## # Utilisation comme clé de tri

```
&q=*&fq={!geofilt}&sfield=store&pt=45.15,-93.85&d=50&sort=geodist() asc
```

## # Utilisation dans le score

```
&q={!func}geodist()&sfield=store&pt=45.15,-93.85&sort=score+asc&fl=*,score
```

## # Facettes par distance

```
&q=*&sfield=store&pt=45.15,-93.85&  
facet.query={!frange l=0 u=5}geodist()&  
facet.query={!frange l=5.001 u=3000}geodist()
```

# *RPT vs LatLonPoint*

- RPT apporte des améliorations fonctionelles vis à vis du type *LatLonPointSpatialField* :
  - Prise en charge de coordonnées non-géographiques (*geo=false*). Juste des x & y
  - Requêtes via des polygones et autres formes, en plus des cercles et des rectangles
  - Possibilité d'indexer des formes (polygones)
  - Agrégation de type Heatmap

# Options de Configuration

- **geo** : *true* ou *false*. Les calculs de distance sont alors différentes.
- **format** : Définit le format de description des formes. Par défaut WKT, mais possible de configurer pour GeoJSON
- **distanceUnits** : Unité de distance. Par défaut *kilometers* si *geo=true*, *degrees* sinon
- **distErrPct** : Définit la précision (influe sur la taille de l'index et la performance de la recherche), une fraction 0.0 (complètement précis) jusqu'à 0.5 .
- **maxDistErr** : Le plus haut niveau de détail pour les données indexées.  
Par défaut : 1m
- **distCalculator** : Algorithme de calcul de distance . Par défaut : si *geo=true* , *haversine* sinon *cartesian*
- **prefixTree** : Définit l'implémentation de grille. Par défaut : si *geo=true* *geohash* , sinon *quad*.
- **maxLevels** : Profondeur maximale de grille pour les données indexés.

# Formes prédéfinies

- Le champ RPT supporte des formes standard : points, cercles, rectangles, lignes, polygones, ...
- En sous-main, la librairie Spatial4j est utilisée
- Exemples d'indexation :

```
<field name="rptField">CIRCLE(28.57,77.32 d=0.051)</field>
```

```
<field name="rptField">POLYGON(20 50, 18 60, 24 68, 26 22, 30  
55, 20 50)</field>
```

# Agrégations de type *heatmap*

- Un champ RPT permet d'agréger des documents en utilisant une grille associée à la carte ou l'espace. Tous les documents de la même cellule sont agrégés.
  - Les cellules de grilles sont déterminées à l'indexation.
  - La précision d'agrégation est fournie à la requête
- *Solr* retourne les données sous forme de tableau d'entiers à 2 dimensions ou au format PNG
- Cette fonctionnalité étend la fonctionnalité de facettes

# Paramètres

- ***facet*** : *true* pour activer l'agrégation.
- ***facet.heatmap*** : le nom du champ RPT.
- ***facet.heatmap.geom*** : La région à prendre en compte (top-left, bottom-right). Exemple : ["-180 -90" TO "180 90"] .
- ***facet.heatmap.gridLevel*** : Le niveau d'agrégation. Une valeur par défaut est calculée
- ***facet.heatmap.format*** : le format .

# Réponse

- La réponse rappelle la dimension de la grille et fournit le décompte pour chaque cellule

```
{gridLevel=6,columns=64,rows=64,minX=-180.0,  
maxX=180.0,minY=-90.0,maxY=90.0,counts_ints2D=[[0, 0, 2,  
1, ....],[1, 1, 3, 2, ...],...]}
```



# BBox

- Le type Bbox permet d'associer un rectangle à un document
- Il supporte les recherches spatiales et permet d'affiner la pertinence par la zone d'intersection entre 2 formes

```
<fieldType name="bbox" class="solr.BBoxField"
geo="true" distanceUnits="kilometers"
numberType="pdouble" />
```

```
<fieldType name="pdouble" class="solr.DoublePointField"
docValues="true"/>
```

# Indexation et recherche

- Pour indexer un document, la syntaxe WKT est nécessaire :

```
ENVELOPE (-10, 20, 15, 10)
```

- Pour rechercher :

```
&q={!field f=bbox  
score=overlapRatio}Intersects (ENVELOPE (-10, 20, 15,  
10))
```

# Distribution et réplication

Distribution et réplication  
SolrCloud : Concepts  
Configuration pour la production

# Introduction à la distribution

- Si les recherche commencent à être longues ou si la taille de l'index approche les limites de l'infrastructure, SolR propose de distribuer un index sur plusieurs serveurs
- L'index est divisé en plusieurs parties : les shards.
- La recherche s'effectue en parallèle sur chaque shard et les résultats sont ensuite agrégés

# Introduction à la réplication

- Un index ou shard peut être répliqué. La réplication d'un index est intéressante :
  - La charge d'un serveur est trop importante pour une seule machine. La charge peut alors être équilibrée sur plusieurs répliques "read-only".
  - Le débit des requêtes d'indexation consomme trop de ressources et pénalise les recherches. On sépare alors l'indexation de la recherche.
  - Pour tout simplement faire un backup

# Legacy vs SolrCloud

- SolR seul permet la distribution et la réplication
- Cependant, pour avoir un réel cluster de noeuds Solr avec équilibrage de charge et tolérance aux pannes, il vaut mieux se tourner vers SolrCloud

# SolR Cloud

# Introduction

- ***SolrCloud*** permet de mettre en place un cluster de serveurs Solr permettant la tolérance aux pannes et la scalabilité.
- Les caractéristiques de la solution :
  - Configuration centralisée de l'intégralité du cluster
  - Equilibrage de charge et fail-over pour les requêtes
  - Intégration de *ZooKeeper* pour la coordination et la configuration des noeuds.



# ZooKeeper

- Solr utilise *ZooKeeper* pour coordonner les nœuds du cluster
- Les recherches et les demandes d'indexations peuvent être effectuées sur n'importe quel nœud du cluster
  - Solr utilise les informations de la BD ZooKeeper pour déterminer quel serveur doit traiter la requête.

# Collections et Shards

- Un cluster peut héberger plusieurs **Collections** de Documents.
- Une collection peut être partitionnée en plusieurs **shards**, contenant un sous-ensemble des Documents de la Collection.
- Le nombre de Shards d'une Collection détermine :
  - La limite théorique du nombre de documents qu'une Collection peut contenir.
  - Le degré de parallélisation possible pour chaque requête.

# Noeuds et répliques

- Un Cluster est constitué de un ou plusieurs noeuds exécutant le serveur Solr.
  - Chaque noeud peut héberger une réplique physique d'un shard
  - Chaque réplique utilise la même configuration spécifiée pour sa Collection
- Le nombre de répliques d'un shard détermine :
  - Le niveau de redondance et la tolérance aux pannes d'un noeud
  - La limite théorique du nombre de requête concurrente

# Réplique leader et indexation

- Chaque shard a au moins une réplique : la réplique leader automatiquement élue.
- Lors de l'indexation,
  - La requête parvient à un des noeuds du cluster qui détermine le Shard auquel appartient le document.
  - Le document est ensuite transmis au noeud hébergeant la réplique leader
  - Le leader fait ensuite suivre la mise à jour aux autres répliques

# Routage de documents

- La stratégie de routing des documents est indiquée par le paramètre ***router.name*** lors de la création de la collection. Il peut prendre les valeurs :
  - ***compositeld*** ( défaut) : calcule une clé de hash à partir de l'ID du document. Il est possible de fournir un préfixe à l'ID pour contrôler le routage.  
Ex : IBM!12345
  - ***implicit*** : Permet d'indiquer le champ (*router.field*) qui servira à router. Si le champ n'est pas présent, le document est rejeté

# Recherche distribuée

- Lorsqu'un noeud Solr reçoit une requête de recherche, elle est routée vers une réplique appartenant à la collection.
- Le noeud agit alors comme un aggregateur :
  - Il crée des requêtes internes qu'il route vers un réplique de chaque shard de la collection
  - Il coordonne les réponses et peut effectuer d'autres requêtes pour compléter la réponse
  - Finalement, il construit la réponse finale pour le client
- Si le paramètre **debug=track**, la requête est tracée et des informations de temps sont disponibles pour chaque phase de la requête distribuée.

# Limitation à des shards

- Dans certains cas, la recherche peut être limitée à certains shards. Les performances sont alors accrues

```
http://localhost:8983/solr/gettingstarted/select?  
q=*:*&shards=shard1,shard2
```

```
http://localhost:8983/solr/gettingstarted/select?  
q=*:*&shards=localhost:7574/solr/  
gettingstarted,localhost:8983/solr/gettingstarted
```

- Ou utiliser le paramètre *\_route\_*

# Répartition de charge

- Pour répartir la charge sur les noeuds du cluster, il faut un répartiteur externe sachant interroger et lire les méta-données stockées dans *ZooKeeper*
- Solr fournit un client Java : ***CloudSolrClient*** qui utilise les données de ZooKeeper pour répartir la charge sur les noeuds up du cluster



# Tolérance aux pannes en écriture

- Un journal de transaction est créé pour chaque noeud permettant de rejouer les mises à jour en cas de crash. Le journal est remis à zéro lors d'un hard commit
- Lors de la création d'une réplique, le journal du Leader est utilisé pour synchroniser la réplique.
- Lors du crash d'un leader, il se peut que certaines répliques n'est pas les dernières mises à jour, après la réélection du leader un processus de synchronisation s'assure que toutes les répliques sont cohérentes

# Tolérance aux pannes en lecture

- Tant qu'au moins une réplique de chaque shard est accessible
- Possibilité d'accepter des résultats incomplets ou incertain. Paramètre ***shards.tolerant*** :
  - ***zkConnected*** : Tous les shards sont accessibles et le noeud servant la requête doit pouvoir obtenir des informations correctes auprès de ZooKeeper
  - ***false*** : Erreur si un des shards n'est pas disponible, même si il est impossible de connecter zooKeeper
  - ***true*** : Réponse même si un shard n'est pas accessible

# Changer le nombre de shards

- Le nombre de shards ne peut être changé sans de lourdes conséquences :  
Création de nouveaux coeurs et réindexation
- Solr propose quand même de diviser un shard existant en 2. (API Collections)
  - Dans ce cas, 2 nouveaux coeurs sont créés et le shard de départ n'est pas touché.
  - Une fois l'opération terminée le shard original peut être supprimé

Mise en place

# Etapes

- Installation ZooKeeper
  - Décompresser
  - Créer un répertoire de stockage de données (*/var/lib/zookeeper*)
  - Créer un fichier de configuration *zoo.cfg*
- Démarrer ZooKeeper
- Lancer les nœuds Solr avec l'option *-k*

# Cluster de *ZooKeeper*

- SolR propose un un serveur ZooKeeper embarqué. Par défaut, le premier noeud du cluster le démarre. Problème : Si le noeud crash il n'y a plus de serveur *ZooKeeper*
- En production, il est nécessaire d'installer un cluster de ZooKeeper externe qui apporte le fail-over.
- Pour que le cluster fonctionne, il lui faut un quorum de serveurs up  
=> En général 3 noeuds *ZooKeeper* est un bon chiffre, cela permet la tolérance d'une panne d'un des noeuds

# Configuration : zoo.cfg

```
# Vérification si les serveurs sont up toutes les 2s
tickTime=2000

# Répertoire de stockage
dataDir=/var/lib/zookeeper

# Port d'écoute
ClientPort=2181

##### Ensemble ZooKeeper #####

# En nombre de ticks, le temps autorisé pour le démarrage et pour les synchronisations
initLimit=5
syncLimit=2

# Adresses de tous les serveurs
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888

#
autopurge.snapRetainCount=3
autopurge.purgeInterval=1
```

# *chroot*

- L'ensemble ZooKeeper peut être utilisé par d'autres applications que SolR
- Dans ce cas, il faut créer un ***znode***  
`bin/solr zk mkroot /solr -z zk1:2181,zk2:2181,zk3:2181`
- Ensuite le *znode* sera ajouté à la string de connection à ZooKeeper



# Mise en service, serveurs SolR

- SolR fournit pour CentOS, Debian, Red Hat, SUSE and Ubuntu Linux :  
*bin/install\_solr\_service.sh*
- Choix : SolR HOME et le user associé
- Recommandation : Séparer les logs et les index de la distribution

# Dimensionnement mémoire

- Maximum Heap size à 10 ou 20 Go est courant sur les environnements de production

Le script *bin/oom\_solr.sh* est exécuté lors d'un *OutOfMemory*  
Il prévient ZooKeeper dans une configuration Cloud

- Dans le fichier `include` :  
`SOLR_JAVA_MEM="-Xms10g -Xmx10g"`

# Configuration ZooKeeper pour SolR

- Il faut indiquer à SolR où se trouvent les serveurs ZooKeeper
- Soit en ligne de commande :

```
bin/solr start -e cloud -z zk1:2181,zk2:2181,zk3:2181/solr
```

- Soit dans le fichier de configuration *`solr.in.sh`* (*`.cmd`*) de SolR

```
ZK_HOST="zk1:2181,zk2:2181,zk3:2181/  
solr"
```

# Rôle de ZooKeeper et fichiers de configuration

- Les fichiers de configuration SolrCloud sont conserés par ZooKeeper. Ils sont chargés lorsque :
  - l'on démarre SolrCloud via le script bin/solr
  - l'on crée une collection via le script bin/solr.
  - Lors d'un chargement explicite

# Création de collections

- Usage:  
`solr create_collection [-c collection]  
[-d confdir] [-n configName] [-shards  
#] [-replicationFactor #] [-p port]`

# Collections API

- Création

`/admin/collections?action=CREATE&name=name`

- Principaux paramètres :

- *router.name*
- *numShards*
- *replicationFactor*
- *maxShardsPerNode*
- ...

# Collections API (2)

- Mise à jour

`/admin/collections?`

`action=MODIFYCOLLECTION&collection=name`

- Attributs pouvant être modifiés :

- *replicationFactor*

- *maxShardsPerNode*

- ...

- Rechargement

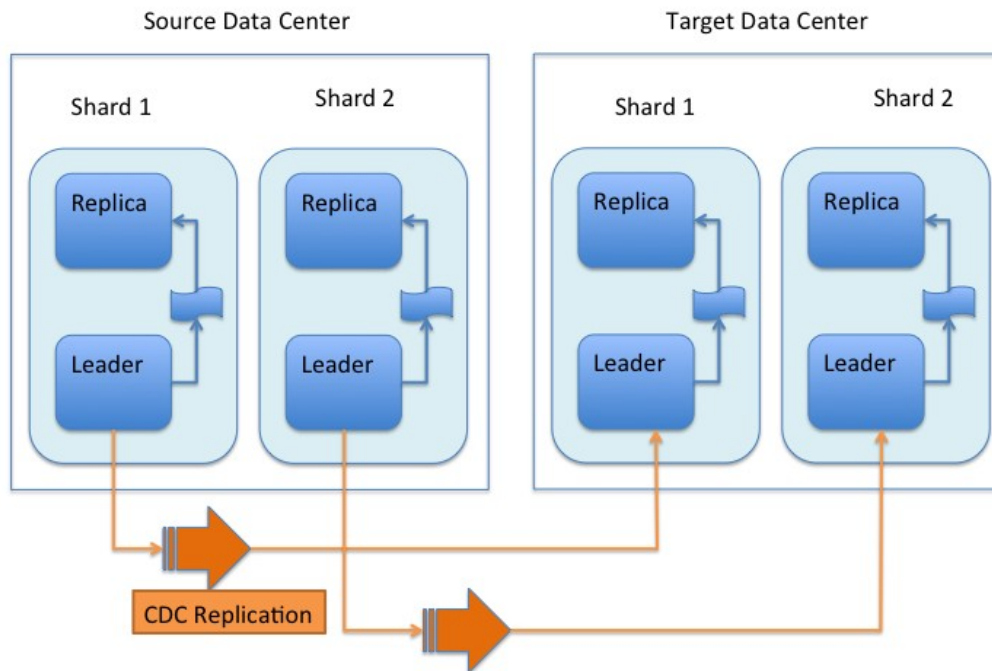
`/admin/collections?action=RELOAD&name=name`

# Cross Data Center Replication

- CDCR permet plusieurs scénarios :
  - Répliquer une collection vers une autre collection
    - À l'intérieur d'un même cluster
    - Entre 2 clusters distinct
  - Synchroniser 2 cluster distincts. (Les écritures peuvent se faire indifféremment sur les clusters)



# Réplication



# Configuration

- L'adresse de ZooKeeper est la seule information requise pour la communication avec le cluster cible
- La configuration peut être affinée par :
  - Le dimensionnement du nombre de threads de réplication
  - La configuration du batch
  - ...

# Configuration source

```
<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
<lst name="replica">
<str name="zkHost">10.240.18.211:2181,10.240.18.212:2181/solr</str>
<str name="source">collection1</str>
<str name="target">collection1</str>
</lst>
<lst name="replicator">
<str name="threadPoolSize">8</str>
<str name="schedule">1000</str>
<str name="batchSize">128</str>
</lst>
<lst name="updateLogSynchronizer">
<str name="schedule">1000</str>
</lst>
</requestHandler>
<!-- Modify the <updateLog> section of your existing <updateHandler> in your config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
<updateLog class="solr.CdcrUpdateLog">
<str name="dir">${solr.ulog.dir}</str>
<!--Any parameters from the original <updateLog> section -->
</updateLog>
<!-- Other configuration options such as autoCommit should still be present -->
</updateHandler>
```

# Configuration cible

```
<requestHandler name="/cdcr" class="solr.CdcrRequestHandler">
  <!-- recommended for Target clusters -->
  <lst name="buffer">
    <str name="defaultState">disabled</str>
  </lst>
</requestHandler>
<requestHandler name="/update" class="solr.UpdateRequestHandler">
  <lst name="defaults">
    <str name="update.chain">cdcr-processor-chain</str>
  </lst>
</requestHandler>
<updateRequestProcessorChain name="cdcr-processor-chain">
  <processor class="solr.CdcrUpdateProcessorFactory"/>
  <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
<!-- Modify the <updateLog> section of your existing <updateHandler> in your
config as below -->
<updateHandler class="solr.DirectUpdateHandler2">
  <updateLog class="solr.CdcrUpdateLog">
    <str name="dir">${solr.ulog.dir}</str>
  <!--Any parameters from the original <updateLog> section -->
</updateLog>
<!-- Other configuration options such as autoCommit should still be present -->
```

# Colocation

- Un collection peut être liée à une autre via la propriété `withCollection`
- Cela garantit que la collection sera allouée sur le même noeud que l'autre, permettant des jointures
- Attention, la collection n'a alors qu'un shard (nommé *shard1*)

# Sécurité, HDFS, Backup, Réplication

# Approches liés à la sécurité

- Différentes approches sont possibles pour sécuriser Solr
  - Utiliser un plugin pour l'authentification et l'autorisation
    - Basic authentication: SolrCloud seulement.
    - Kerberos authentication: SolrCloud ou mode standalone
    - Rule-based authorization: SolrCloud seulement.
    - PKI authentication: SolrCloud seulement – pour sécuriser le trafic inter-noeud
  - Utiliser SSL
  - Contrôler les accès via ZooKeeper (Solr Cloud)

# Mise en place des plugins

- La mise en place de plugins pour l'authentification et les autorisations est différente selon le mode de SolR :
  - SolRCloud : un fichier *security.json* doit être créé et chargé dans ZooKeeper avant utilisation
  - Mode standalone : La propriété système `-DauthenticationPlugin=<pluginClassName`



# Démarrer Solr sur HDFS

- Il est possible que Solr stocke ses index et ses logs de transactions sur un système de fichier HDFS (Apache Hadoop)
- Exemple standalone :  
`bin/solr start -`  
`Dsolr.directoryFactory=HdfsDirectoryFactor`  
`y`  
`-Dsolr.lock.type=hdfs`  
`-Dsolr.data.dir=hdfs://host:port/path`  
`-Dsolr.updatelog=hdfs://host:port/path`

# Backup / Restore

- 2 approches pour la sauvegarde et la restauration d'index !

- Collections API pour SolrCloud

/admin/collections?

**action=BACKUP**&name=myBackupName&collection=myCollectionName&location=/path/to/my/shared/drive

/admin/collections?**action=RESTORE**&name=myBackupName&location=/path/to/my/shared/drive&collection=myRestoredCollectionName

- Gestionnaire de réplication pour le mode standalone

# Gestionnaire de réplication

```
<requestHandler name="/replication" class="solr.ReplicationHandler">
  <lst name="master">
    <str name="replicateAfter">optimize</str>
    <str name="backupAfter">optimize</str>
    <str name="confFiles">schema.xml,stopwords.txt,elevate.xml</str>
    <str name="commitReserveDuration">00:00:10</str>
  </lst>
  <int name="maxNumberOfBackups">2</int>
  <lst name="invariants">
    <str name="maxWriteMBPerSec">16</str>
  </lst>
</requestHandler>
```

<http://localhost:8983/solr/gettingstarted/replication?command=backup>

# Configuration des traces

- Interface d'administration
- Logging API :  
# Set the root logger to level WARN  
`curl -s http://localhost:8983/solr/admin/info/logging --data-binary "set=root:WARN&wt=json"`
- Au démarrage du serveur :
  - Variable d'environnement : *SOLR\_LOG\_LEVEL*
  - Options *-v* ou *-q*
- Changement total de la configuration :  
`server/resources/log4j.properties`
- Logger sous forme de WARN les requêtes lentes :  
`<slowQueryThresholdMillis>1000</slowQueryThresholdMillis>`

# **Annexes**

# Quelques sources de doc

1. Wiki SolR : <http://wiki.apache.org/solr>
2. SolR Reference guide de LucidWorks :  
<https://cwiki.apache.org/confluence/display/solr>  
(parfois plus à jour que le wiki)
3. Livre Apache SolR Enterprise Search Server :  
<http://www.solrenterprisesearchserver.com>  
(voir le PDF « search quick reference »)

# Optimisation recherche

- Caches, intéressant si peu d'indexation, possibilité d'autowarm:
  - ***filterCache*** : résultats de chaque paramètre fq. Configuration par nombre d'entrées
  - ***queryResultCache*** : Cache de requêtes full-text avec le tri des documents. Configuration par nombre d'entrées + RAM
  - ***documentCache*** : Cache les documents (champs stockés)