



Le moteur de recherche SolR

david.thibau@gmail.com
2025

Agenda

- **Introduction / Installation**
- **Mise en place de coeur**
 - Création
 - Configuration
 - Schéma
- **L'indexation de données**
 - Analyseurs de texte
 - API d'indexation
 - *Update Processor Chain*
 - *SolrCell*
 - *Data Import Handler*
- **Recherche full-text**
 - Principes
 - Configuration des handlers de recherche
 - Calcul du score
 - Syntaxe des différents parseurs
- **Fonctionnalités de recherches**
 - Surbrillance
 - Recherche à facettes, groupes
 - Vérification orthographique
 - Auto-complétion
 - Recherches géographiques
 - Elévation
- **Production**
 - Recommandations générales
 - Distribution et réplication
 - *SolrCloud* : Concepts
 - *SolrCloud* : Mise en place

Problématique

« trouver de l'information
(*en général des documents*)
non-structurée
(*en général du texte*)
qui répond à une intention de
recherche, dans une large
collection
(*en général numérisée*) »







Implémentation SGBD ?

```
SELECT * FROM post
```

```
WHERE
```

```
topic LIKE
```

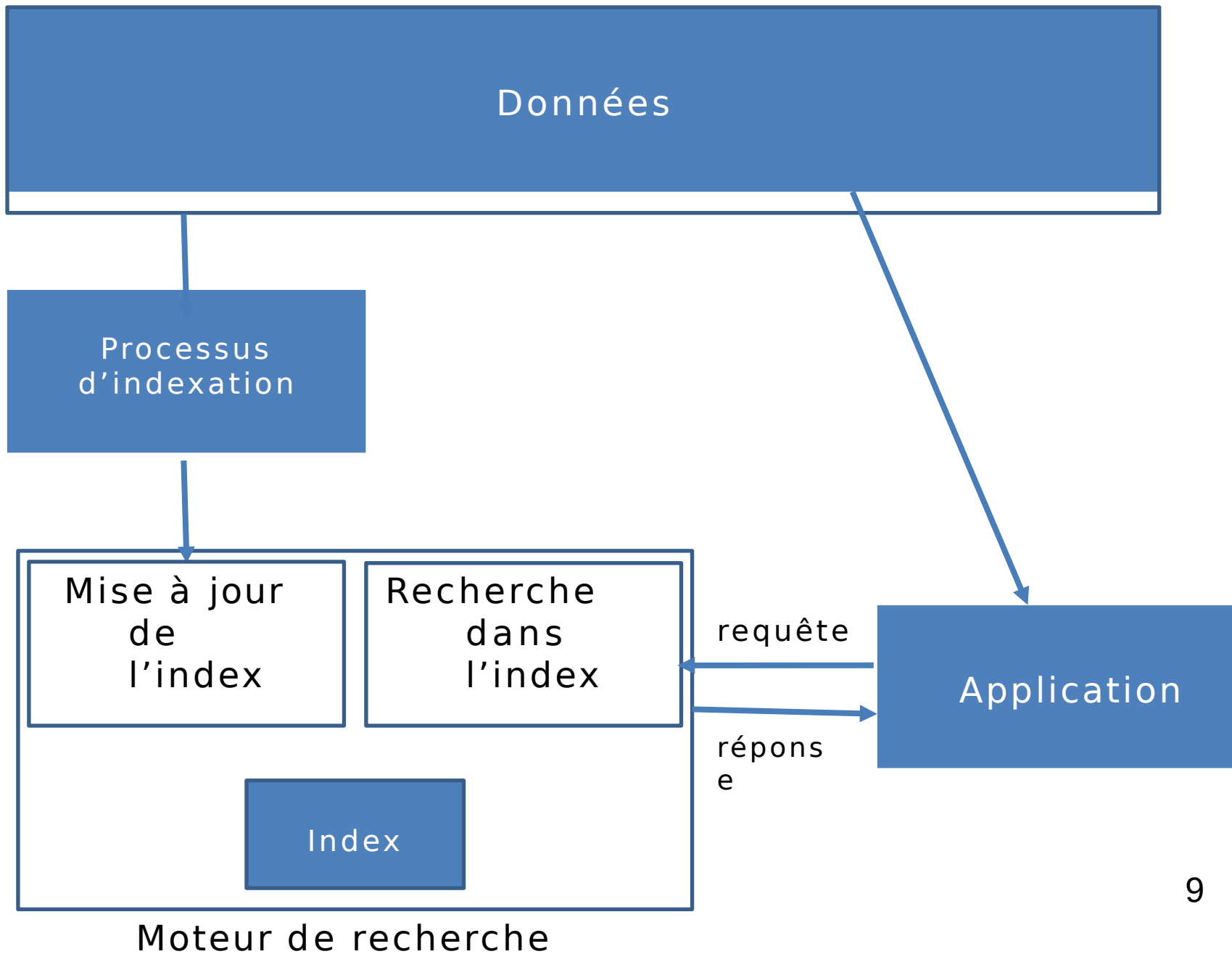
```
'%keyword%'
```

```
OR author LIKE '%keyword%'
```

```
ORDER BY id DESC
```

Les SGBD ne sont pas adaptés pour faire de la recherche plein-texte

- Comment fait-on pour les recherches avec plusieurs termes ? sur plusieurs tables ?
- Full table scan = mauvaise performances
- Pas de scoring/pertinence des résultats
- Gestion des langues, des recherches approchantes (pluriels, féminins, etc.)

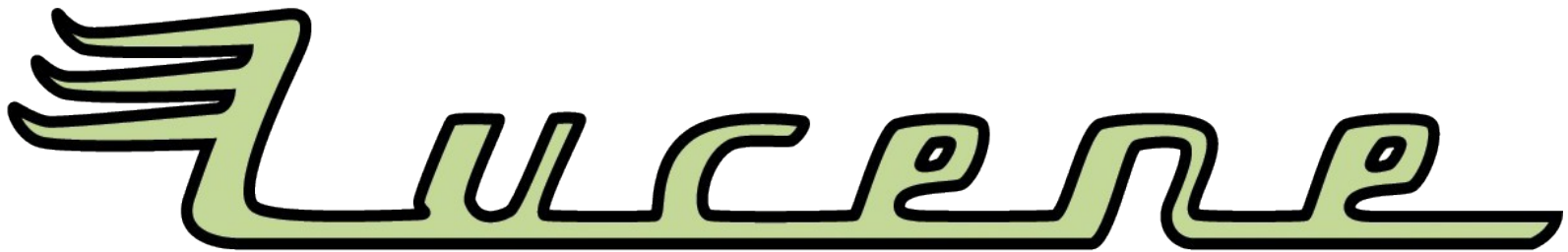


Critères pour une solution d'entreprise

- Rapide :
 - Temps d'attente
 - Suggestion au fur et à mesure de la saisie
- Pertinent
 - Les meilleurs documents présentés en premier
- Flexible
 - S'adapter à la langue, au métier, ...
- Scalable
 - Charge utilisateur
 - Volume de données



- Open-source, écrit en Java;
- Serveur de recherche autonome proposant une API
- 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;
- Un **index** contient des documents
 - Un **document** contient des champs (« fields »)
 - Un **champ** contient des termes (« terms »)
- SolR expose les fonctionnalités de Lucene dans un serveur, au travers de HTTP et des formats XML et JSON;

SolR : historique

- Développé au sein de CNET par Yonik Seeley
- Transféré à la fondation Apache en 2006;
- Incubé jusqu'en 2007 : v1.2;
- **2007 - 2021 (8.x)** Lucene et SolR sont dans le même projet Apache, et relasés ensemble¹;
 - Février 2015 : v5.x (mode standalone, SolrCloud)
- 9.x Lucene et SolR se sépare. (Committers Lucene sont principalement de la société Elastic). Les versions sont proches mais ne correspondent plus
- 9.2 Novembre 2022 : Intégration de knn, recherche vectorielle
- 9.7 : Intégration avec framework IA, RAG
- Version actuelle 9.9.0 utilise 9.9.2

ELS vs SolR

- SolR est souvent comparé à **Elastic Search** qui propose également une API REST pour Lucene

	SOLR	ELS
Installation & Configuration	Documentation très détaillée	Simple et intuitif
Indexation/ Recherche	Orienté Texte	Texte et autres types de données pour les agrégations
Scalability	Cluster via ZooKeeper et SolRCloud	Nativement en cluster
Communauté	Importante mais stagnante	A explosé ces dernières années
Documentation	Très complète et très technique	Très complète, facile d'accès, bcp de tutoriaux
Licence	Full Open-Source	Communautaire et Commerciale

API et Clients

- SolR s'arrête à fournir un flux de réponse en JSON, XML
- Il ne propose pas d'interface utilisateur de recherche mais une interface d'administration
- Pour mettre en place une application utilisant SolR ; on peut s'appuyer sur des librairies clientes

Voir

<https://wiki.apache.org/confluence/display/solr/IntegratingSolr>

- Java : SolRj
- Javascript : ajax-
- SolR Ruby :
- PHP, .Net, etc...

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 **anciennement** par SolR

Surbrillance des résultats / Recherche à facettes

HUMAN RIGHTS WATCH

العربية 簡中 繁体中文 English Français Deutsch 日本語 Português Русский Español More +

FAIRE UN DON

Pays ▾

Thèmes ▾

Rapports

Multimédia

Impact

Agir

À propos de HRW ▾

Résultats de la recherche

1165 résultats trouvés

Rechercher

Sort by ☒ Résultats les plus pertinents ☐ Résultats les plus récents



24 octobre 2025 | News

La France manque à ses obligations envers les enfants migrants non accompagnés

Un Comité de l'ONU exhorte la France à garantir les droits des enfants migrants

... violations graves et systématiques des droits des enfants **migrants** non accompagnés. Le comité a conclu qu'en raison de ... entre Menton et Vintimille, où des enfants **migrants** sont sommairement expulsés vers l'Italie, en ... l'ONU exhorte la France à garantir les droits des enfants **migrants** ...



19 septembre 2025 | News

La détention de migrants dans les prisons provinciales du Canada prend fin

L'Ontario est la dernière province à interdire cette pratique ; Ottawa devrait mettre fin à l'expansion des prisons fédérales

... C'est une victoire majeure pour les droits des **migrants** et des

Affiner la recherche

Filtrer par année

- Any - ▾

Filtrer par pays

Choisir des options ▾

Filtrer par sujet

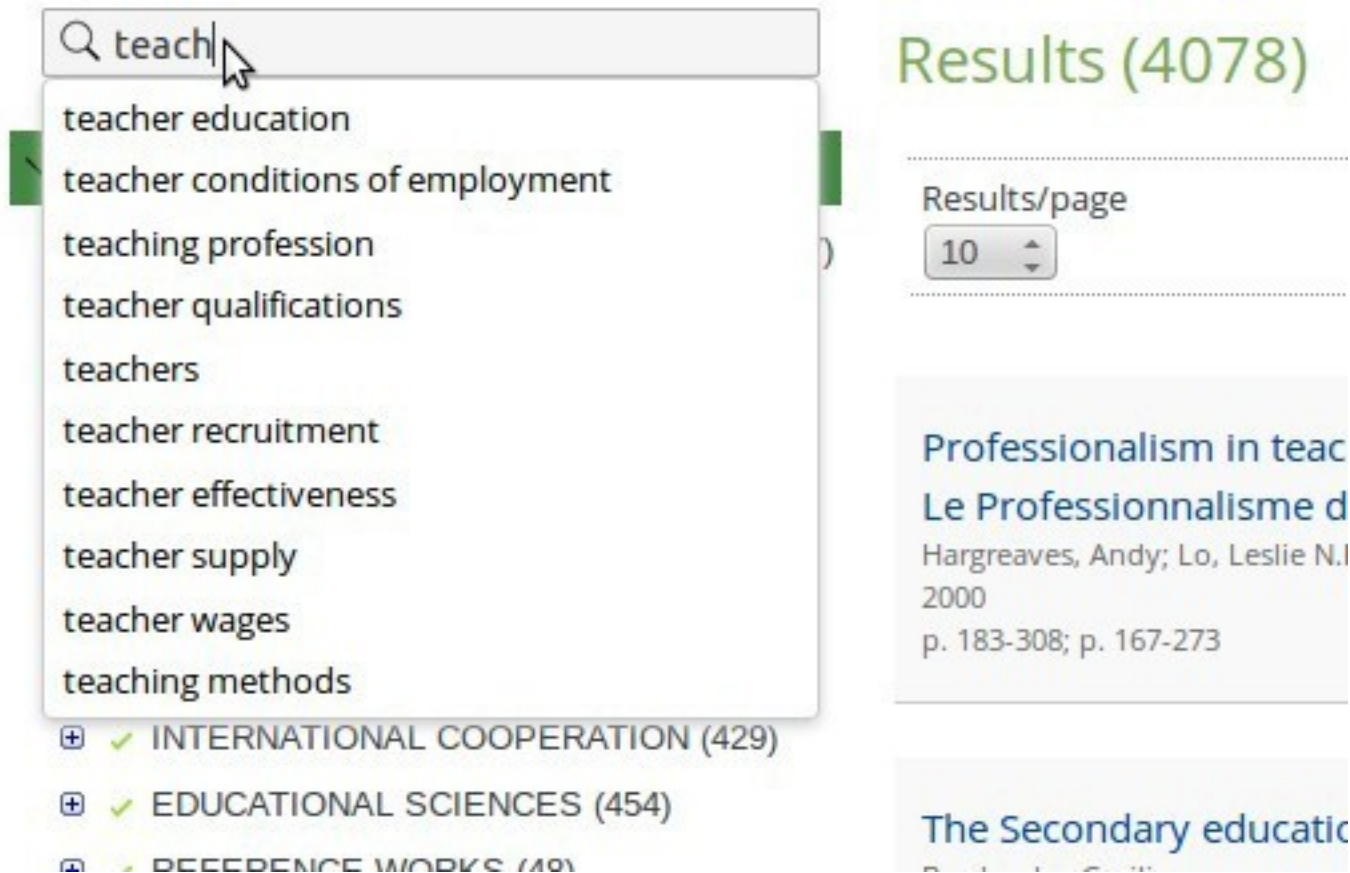
Choisir des options ▾

Filtrer par type

Choisir des options ▾

Filtrer par catégorie de «News»

Suggestions et corrections



The screenshot displays a search interface. On the left, a search bar contains the text 'teach' with a cursor. Below the search bar, a dropdown menu lists suggestions: 'teacher education', 'teacher conditions of employment', 'teaching profession', 'teacher qualifications', 'teachers', 'teacher recruitment', 'teacher effectiveness', 'teacher supply', 'teacher wages', and 'teaching methods'. Below the suggestions, there are three categories with checkmarks and counts: 'INTERNATIONAL COOPERATION (429)', 'EDUCATIONAL SCIENCES (454)', and 'REFERENCE WORKS (18)'. On the right, the search results are displayed. The title 'Results (4078)' is shown in green. Below it, a 'Results/page' dropdown menu is set to '10'. The first result is titled 'Professionalism in teaching' and 'Le Professionnalisme d'enseignement', by Hargreaves, Andy; Lo, Leslie N., published in 2000, with page ranges p. 183-308 and p. 167-273. The second result is titled 'The Secondary education'.

UNESCO IIEP (International Institute for Educational Planning) :

<http://plan4learning.iiep.unesco.org>

Recherche sur synonymes



Chauffeur poids lourd emploi chauffeur poids lourd

🔍 chauffeur poids lourd

📍 OÙ ?

Rechercher

Emploi Logistique > Chauffeur Poids Lourd

▼ Métier

- ☒ Chauffeur PL (148)
- ☐ Acheteur industriel (1)
- ☐ Acheteur (31)
- ☐ Adjoint de direction (1)
- ☐ Affréteur (30)
- ☐ Agent administratif (30)
- ☐ Agent de distribution (1)
- ☐ Agent de planning (8)
- ☐ Agent de quai (28)
- ☐ Agent de réception quai (4)

▼ Type de Contrat

- ☐ CDI (13)
- ☐ CDD (5)
- ☐ INTERIM (128)
- ☐ STAGE (2)

▼ Expérience

- ☐ Débutant (0 à 1 an) (11)
- ☐ Junior (2 à 4 ans) (20)
- ☐ Confirmé (5 à 9 ans) (6)

▼ Niveau d'étude

- ☐ Autodidacte (1)
- ☐ CAP (2)

► Cargaison

Votre recherche - Tout effacer

Métier

✖ Chauffeur PL

Localité

- ☐ Alsace (5)
- ☐ Aquitaine (8)
- ☐ Auvergne (4)
- ☐ Basse-Normandie (5)
- ☐ Bourgogne (7)

- ☐ Aisne - 02 - (1)
- ☐ Allier - 03 - (1)
- ☐ Alpes-Maritimes - 06 - (2)
- ☐ Aveyron - 12 - (1)
- ☐ Bas-Rhin - 67 - (2)

- ☐ Abbeville (1)
- ☐ Andrézieux-Bouthéon (1)
- ☐ Anse (1)
- ☐ Aulnay-sous-Bois (1)
- ☐ Avignon (1)

Trier par : ☐ Date ☒ Pertinence

148 annonces actives de - de 30 jours



Ne ratez plus d'opportunités !

Recevez les nouvelles annonces de cette recherche par e-mail



Conducteur PL/SPL grutier (H/F) Aulnay-sous-Bois

CDI

Île-de-France - Seine-Saint-Denis (93) - 93600 - Emploi GROUPE GT - 03/01/2014

: 182H/mois du lundi au samedi (prise de poste à 6h) GT « pour réussir ensemble » GT Ile de France Nord recrute pour soutenir son développement : **Conducteur...**



Conducteur PL grutier (H/F) Reims

CDI

Champagne-Ardenne - Reims (51) - 51100 - Emploi GROUPE GT - 03/01/2014

/ mois GT GT Nord recrute pour soutenir son développement : **Conducteur PL** grutier (H/F) Vous êtes titulaire du **Permis C**. Vous souhaitez travailler...



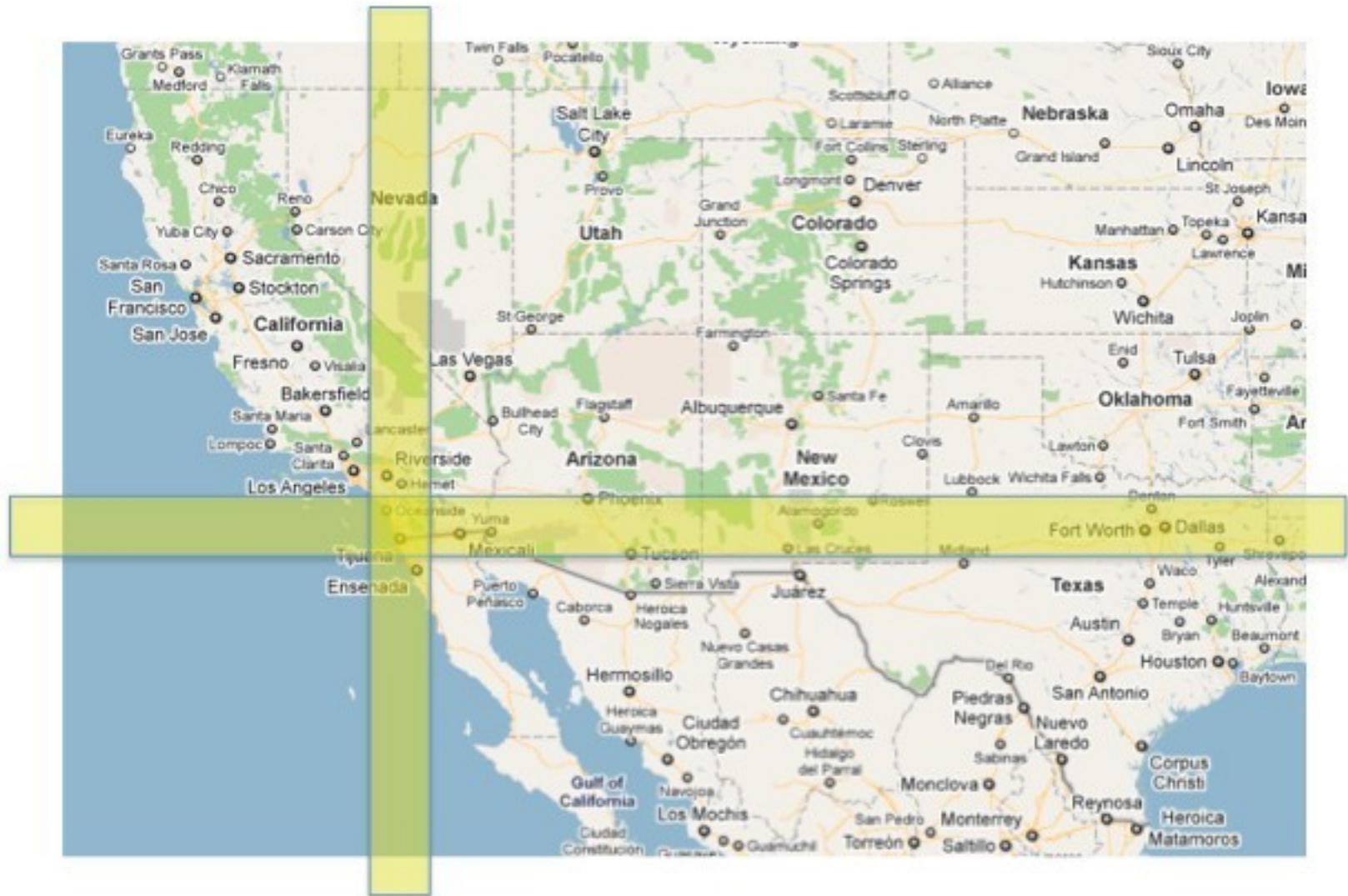
Conducteur PL (H/F) Distribution frigorifique Vire

CDD

Basse-Normandie - Calvados (14) - 14500 - Emploi GROUPE GT - 03/01/2014

Travail du lundi au vendredi prise de poste à 6h GT GT Nord recrute pour soutenir son développement : **Conducteur PL** (H/F) Distribution frigorifique Vous...

Recherche spatiale



Recherche vectorielle

The diagram illustrates a vector search process. On the left, a yellow hexagon labeled "internet" is connected by a lightning bolt to a search bar. The search bar contains the query "How fast should my internet be?". To the right, a document snippet is shown with the text: "In order to stream from our service you will need a high quality connection. The required connection speed for using the service will vary depending on the quality of your connection. For most customers we recommend at least...". The words "required", "connection", and "speed" in the document snippet are highlighted in pink, yellow, and green respectively, matching the highlights in the query. A button labeled "Sélectionner le texte" (Select the text) is positioned above the document snippet. In the bottom right corner, there is a button with a flag icon and the text "Traduire" (Translate).

Sélectionner le texte

internet

How fast should my internet be?

In order to stream from our service you will need a high quality connection. The required connection speed for using the service will vary depending on the quality of your connection. For most customers we recommend at least...

Traduire

Interface d'administration

SolR propose une interface web d'administration :

- Gérer et parcourir les index (simple nœud ou distribué) :
 - Tester les analyseurs
 - Effectuer des recherches
 - Indexer les données
 - Visualiser les fichiers de config,
 - Modifier le Schema (définition des champs d'un document)
- Surveiller un serveur standalone ou un cluster

Solr admin



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

gettingstarted

Overview

Analysis

Dataimport

Documents

Files


Query

Stream

Schema


Core Selector

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

Use [original UI](#)

Application exemple

- SolR inclut une application exemple servant de démonstration des fonctionnalités.

Find:

☐ Boost by Price

Field Facets

cat
[electronics](#) (2)
[hard drive](#) (2)
manu_exact
[Maxtor Corp.](#) (1)
[Samsung Electronics Co. Ltd.](#) (1)

Query Facets

[GB](#) (2)

Range Facets

price
[50.0 - 100.0](#) (1)
[350.0 - 400.0](#) (1)

2 results found in 28 ms Page 1 of 1


Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133 [More Like This](#)

Id: SP2514N

Price: 92,USD

Features: 7200RPM, 8MB cache, IDE Ultra ATA-133 NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor

In Stock: true


[Larger Map](#)


Maxtor DiamondMax 11 - hard drive - 500 GB - SATA-300 [More Like This](#)

Id: 6H500F0

Price: 350,USD

Features: SATA 3.0Gb/s, NCQ 8.5ms seek 16MB cache

In Stock: true


[Larger Map](#)

2 results found. Page 1 of 1

Options: [enable debug](#) [enable annotation](#) [XML](#)

Exemples

Option de démarrage -e

- La distribution propose 4 exemples :
 - **cloud** : Permet de démarrer 1 à 4 nœuds en utilisant des *réplica* et des *shards*
 - **techproducts** : Mode standalone avec un schéma correspondant aux documents du répertoire *exampledocs*
 - **dih** : Mode standalone avec activation du *DataImportHandler* (contenu stocké en RDBMS ou autre)
 - **schemaless** : Mode standalone avec possibilité de créer des champs à la volée

Mise en place de cœur

Création de cœur

Configuration

Schéma

Concepts

- Un **coeur/core** contient :
 - Un **index** Lucene (les données indexées),
 - Un fichier de configuration principale (***solrconfig.xml***),
 - Un **schéma** (managed-schema ou schema.xml) décrivant les champs et types,
 - Éventuellement des **fichiers spécifiques** (stopwords, synonymes...).
- Chaque cœur est indépendant, mais plusieurs cores peuvent tourner dans une même instance Solr (installation multicore).

Cycle de vie

- 1) SolR doit être démarré
- 2) Créer le coeur :
 - Via la ligne de commande
 - Manuellement via la création de dossier
- 3) Personnaliser le schéma en définissant les types et les analyseurs
- 4) Indexer les données
- 5) Interroger
- 6) Exploiter

Script de démarrage

- Le script solr et la commande ***start***

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

- Les autres commandes de contrôle :

```
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
- **--server-dir <dir>** : Répertoire du serveur. défaut : *server*
- **-z <zkHost>** : Mode cloud : Adresse de ZooKeeper. Défaut embarqué
- **-m <memory>** : -Xms ET -Xmx. Défaut 510Mo
- **--server-dir <dir>** : *solr.solr.home* Répertoire home de la configuration
- **--data-home <dir>** : *solr.data.home*, Répertoire de stockage des index
- **-e <example>** : *cloud, techproducts, dih, schemaless*
- **-a** : paramètres additionnels de la JVM
- **-v** et **-q** : Niveau de verbosité

Création de cœur

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

```
bin/solr create -c mycœur
```

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

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

Options de *create*

- **-c <name>** (requis) : Le nom du cœur ou de la collection à créer
- **-d <confdir>** : Le répertoire de configuration :
 - **_default**: Configuration minimale avec détection automatique de champs
 - **sample_techproducts_configs**: Configuration avec les fonctionnalités optionnelles de l'exemple *techproduct*
 - ou un chemin personnalisé vers son propre répertoire de configuration
- **-n <confname>** (optionnel) : Nom du répertoire ou la configuration est copiée

Autres commandes liées aux cœurs

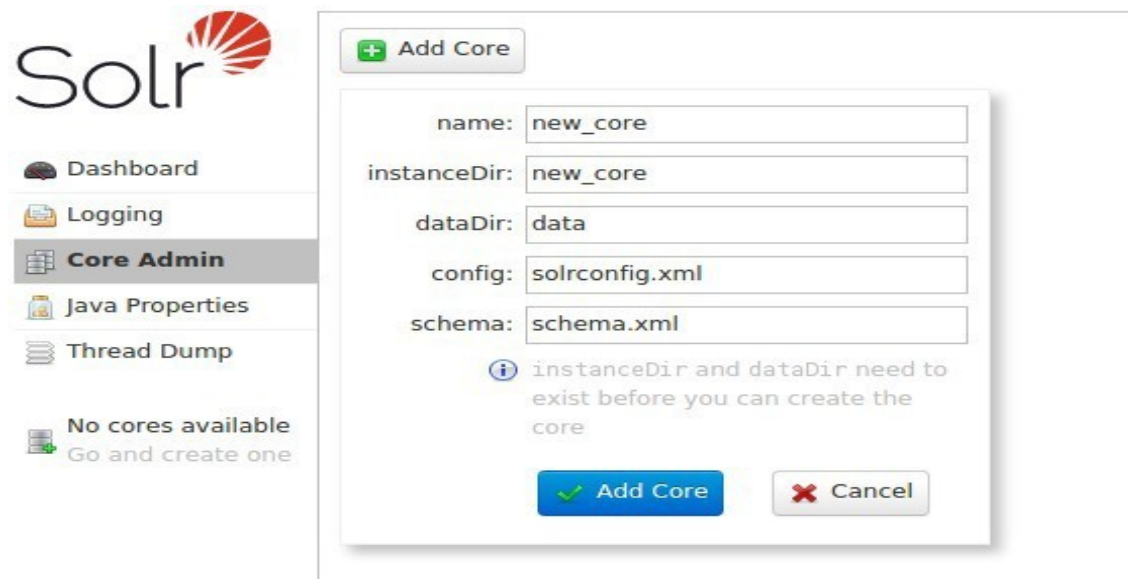
- *Solr* propose 2 autres commandes pour la gestion des cœurs

```
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



The screenshot displays the Solr Admin interface. On the left is a sidebar with the Solr logo and navigation links: Dashboard, Logging, Core Admin (highlighted), Java Properties, Thread Dump, and a message 'No cores available' with a link 'Go and create one'. The main content area shows a modal dialog titled '+ Add Core'. The dialog contains five input fields: 'name' (new_core), 'instanceDir' (new_core), 'dataDir' (data), 'config' (solrconfig.xml), and 'schema' (schema.xml). Below these fields is an information icon and a message: 'instanceDir and dataDir need to exist before you can create the core'. At the bottom of the dialog are two buttons: a green 'Add Core' button and a grey 'Cancel' button.

API Rest

- Les 2 méthodes précédentes utilisent l'API HTTP « Rest »

```
curl http://localhost:8983/solr/admin/cores?  
action=CREATE&name=formation&instanceDir=formation
```

Le répertoire spécifié doit contenir :
conf/

```
|— solrconfig.xml  
|— managed-schema  
|— ... (autres fichiers éventuels)
```

Autres Actions avec la même API

- **RELOAD** : Recharge la configuration d'un cœur sans le redémarrer

/solr/admin/cores?action=RELOAD&core=formation

- **UNLOAD** : Décharge un cœur (il n'est plus accessible mais les fichiers restent)

/solr/admin/cores?action=UNLOAD&core=formation

- **STATUS** : Vérifie l'état d'un cœur

/solr/admin/cores?action=STATUS&core=formation

- **SWAP** : Échange deux cœurs (utile pour les mises à jour sans interruption)

/solr/admin/cores?action=SWAP&core=old&other=new

Mise en place de coeur

Création de coeur
Configuration
Schéma

Répertoires importants

Pour un *solr.home* donné :

- Configuration commune aux cœurs :
`server/solr/solr.xml`
- Configuration d'un cœur:
`server/solr/<core>/conf`
- Données d'un cœur :
`server/solr/<core>/data`

Axes de configuration

2 axes de configuration

- Les **gestionnaires de requêtes** définissent les points d'accès HTTP disponibles pour l'indexation et la recherche et leur configuration des traitements correspondants :
<solr.home>/<core>/conf/solrconfig.xml
- La **structure de l'index** : les différents champs composant les documents.
2 modes
 - **Dynamique** : Le schéma est mis à jour via Solr, via son API ou une configuration « découverte »
Utile plutôt en développement
Fichier : **<solr.home>/<core>/conf/managed-schema**
 - **Verrouillé**. Le schéma n'évolue ou seulement via des éditions manuelles demandant un rechargement.
Fichier : **<solr.home>/<core>/conf/schema.xml**

Configuration schéma contrôlé par l'appli

La balise **<schemaFactory>** dans *solrConfig.xml* permet de configurer comment est contrôlé le schéma

- Par défaut, Solr utilise *ManagedIndexSchemaFactory* qui s'appuie sur le fichier *managed-schema*
- Pour contrôler complètement le schéma, éditer *solrconfig.xml* :

```
<!-- ClassicIndexSchemaFactory nécessite l'utilisation d'un fichier de configuration schema.xml et interdit toute modification programmatique du schéma au moment de l'exécution. Le fichier schema.xml doit être modifié manuellement et n'est chargé que lorsque la collection est chargée. -->
```

```
<schemaFactory class="ClassicIndexSchemaFactory" />
```

```
<!-- Interdiction d'ajout automatique de champ -->
```

```
<updateRequestProcessorChain name="add-unknown-fields-to-the-schema" default="${update.autoCreateFields:false}"
```


API config

- Il n'est pas toujours aisé de voir la configuration réelle d'un cœur seulement à partir de *solrconfig.xml*
Les configurations par défaut n'y sont pas toujours visibles
- Pour voir la configuration effective (, il faut utiliser l'API Rest.

GET http://<server>/solr/<cœur>/config

- La réponse contient plusieurs blocs, Citons
 - ***RequestHandler*** : Endpoints
 - ***Query*** : Configuration générale de la recherche
 - ***SearchComponents*** : Composants de recherche réutilisable
 - ***ResponseWriter*** : Format des réponses
 - ***UpdateHandlers*** : Paramètres d'indexation

Surcharge de la config et API config

- Le fichier *solrconfig.xml* contient des notations de type
 - `${update.autoCreateFields: false}`
la valeur de la propriété utilisateur `update.autoCreateFields` si elle est définie, sinon `false`
- Ces propriétés peuvent être indiquées dans le fichier *core.properties* ou modifiées via l'API **solr/config**

```
bin/solr config -c formation -p 8983 -action set-user-property \  
-property update.autoCreateFields -value false  
POST solr/config -d {"set-property":{"<property>": "<value>"}}
```
- Dans ce cas, *solrconfig.xml* n'est pas modifié. Les modifications sont stockées dans le fichier **configoverlay.json** qui surcharge *solrconfig.xml*.

Mise en place de cœur

Création de cœur
Configuration
Schéma

Balises du schema

- Balises
 - ***fieldType*** : déclare un type possible pour les champs. Des types sont pré-définis, si type full-text, associé à un ou plusieurs « analyzer »
 - ***analyzer*** : définit les traitements qui seront appliqués aux valeurs du *field*, à l'indexation **et** à la recherche
 - ***field*, *dynamicField*, *uniqueKey*** : définissent un champ d'un document
 - ***copyField*** : duplique automatiquement les valeurs d'un champ dans un autre pour pouvoir l'indexer différemment

Balise *<fieldType>*

name : son nom (référéncé dans un field);

class: sa classe Java¹

- ***solr.BoolField*** : booléen
 - ***solr.IntPointField/solr.LongPointField*** : entiers
 - ***solr.FloatPointField*** : décimaux
 - ***solr.DatePointField, solr.DateRangeField***: date
 - ***solr.LatLonPointSpatialField*** : latitude/longitude
 - ***solr.CurrencyFieldType*** : Monnaie
 - ***solr.TextField*** : texte
- Des attributs communs à *<field>*
(*multivalued, docValues, ...*)
 - Des attributs dépendants de la classe
 - Pour les champs textes, généralement une balise *<analyzer>* imbriquée

1. Dans ce fichier, « *solr...* » est un raccourci pour « *org.apache.solr.analysis* »

Examples

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100"
multiValued="true">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt" />
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt" />
    <filter class="solr.SynonymGraphFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>

<fieldType name="location_rpt" class="solr.SpatialRecursivePrefixTreeFieldType"
geo="true" distErrPct="0.025" maxDistErr="0.001"
distanceUnits="kilometers" />
```

Balise *<uniqueKey>*

- La balise *<uniqueKey>* permet de préciser le champ qui sert de clé pour les documents `<uniqueKey>id</uniqueKey>`
- Cette balise n'est pas requise mais recommandée
- L'*updateProcessor uuid*, permet de générer une valeur unique lors de l'indexation.

Configuration *solrconfig.xml* :

```
<updateProcessor class="solr.UUIDUpdateProcessorFactory"
name="uuid">
  <str name="fieldName">id</str>
</updateProcessor>
```

Attributs de *<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

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. (Compatible avec certains types de champs)

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***

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 *_num* 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

Analyseurs de texte

API d'indexation

Compléments

Update Processor Chain

Documents bureautique

Base de données

Introduction

- Pour l'indexation et la recherche SolR utilise des **analyseurs** qui transforment un texte en un flux de "termes".

Ils sont en général constitués de :

- Les **filtres de caractères** effectuent du remplacement/suppression de caractères
Ex : « & devient et » ou « suppression des balises HTML »
- Un **tokenizer** : Responsable de splitter un texte en token ou termes
- 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 crée 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 si 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

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 l'indexation : `type="index"`
- Un analyzer pour la recherche :
`type="query"`

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

*En général, on utilise le **même** analyseur pour l'indexation et la recherche pour un champ donné*

Exemples

Un seul analyseur pour l'indexation ET la recherche

```
<fieldType name="nametext" class="solr.TextField">  
  <analyzer class="org.apache.lucene.analysis.WhitespaceAnalyzer"/>  
</fieldType>
```

2 analyseurs pour l'indexation ET la recherche

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
```

```
  <analyzer type="index">  
    <tokenizer class="solr.StandardTokenizerFactory"/>  
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"  
    />  
    <filter class="solr.LowerCaseFilterFactory"/>  
  </analyzer>
```

```
  <analyzer type="query">  
    <tokenizer class="solr.StandardTokenizerFactory"/>  
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />  
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"  
    ignoreCase="true" />  
    <filter class="solr.LowerCaseFilterFactory"/>  
  </analyzer>
```

```
</fieldType>
```

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 : 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

Analyseurs de texte

API d'indexation

Compléments

Update Processor Chain

Documents bureautique

Base de données

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, en postant des données aux formats XML, JSON ou CSV.
 - 3 facilités sont fournies par SolR :
 - l'outil en ligne de commande **post**
 - L'interface d'administration
 - Les librairies clientes propres à chaque langage
- Pour Java : *Solrj* et sa classe *SolrClient*

Utilitaire post

- Dans un environnement Linux, Apache SolR propose le shell **post** pour poster différents types de contenus vers un serveur *ApacheSolR*¹
- `post -c <collection>[OPTIONS]
<files| directories|urls|-d
["...",...]>`
- **Exemple :**
 - `bin/post -c gettingstarted *.xml`
 - `bin/post -c signals -params "separator=%09" -type
text/csv data.tsv`
 - `bin/post -c gettingstarted afolder/`
 - `bin/post -c gettingstarted -filetypes ppt,html
afolder/`

Environnement Windows

- En environnement Windows, il est possible d'utiliser le programme Java sous-jacent :

SimplePostTool

```
java -jar example/exampledocs/post.jar -h
SimplePostTool version 5.0.0
Usage: java [SystemProperties] -jar
post.jar [-
h|-] [<file|folder|url|arg>
[<file|folder|url|arg>...]]
```

Update Handler

- Par défaut, SolR configure un ***updateHandler*** capable de supporter les formats XML, CSV et JSON
Accessible à l'URL */update*

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

Indexation et Commit

Pour qu'un document indexé soit visible dans les recherches, Solr doit effectuer un **commit**.

- Cette opération écrit en dur sur le disque (segments Lucene) les documents encore en mémoire.

Le commit est une opération coûteuse → il ne faut pas la déclencher après chaque document.

Il peut être effectué :

- manuellement (via l'API ou une commande explicite),
- ou automatiquement selon une politique configurée dans solrconfig.xml.

Solr propose aussi également un **soft commit**, plus rapide car il ne réécrit pas les données sur disque.

⚠ Cependant, en cas de crash, les modifications non "hard commitées" peuvent être perdues.

Options de configuration du commit

```
<updateHandler class="solr.DirectUpdateHandler2">
```

```
  <autoCommit>
```

```
    <maxTime>15000</maxTime>  <!-- commit toutes les 15 secondes -->
```

```
    <maxDocs>1000</maxDocs> <!-- ou tous les 1000 documents -->
```

```
    <openSearcher>true</openSearcher> <!-- nouveaux documents sont visibles de suite -->
```

```
  </autoCommit>
```

```
  <autoSoftCommit>
```

```
    <maxTime>3000</maxTime>
```

```
  </autoSoftCommit>
```

```
</updateHandler>
```

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 documents

- ***commitWithin***

- Committé au bout d'un certain nombre de ms.

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 *SolR*

Le format XML : commit

```
# Ecriture des nouveaux documents (en mémoire) sur le disque
<commit />
# Annulation des derniers ordres d'indexation
<rollback />
# Commit + fusion segments Lucene
<optimize />
```

- Commits :
 - Lents : donc faire un seul gros commit à la fin
 - Pas de transactions par clients (commit global)
 - Mode auto-commit dans *solrconfig.xml*
 - Tant que les documents ne sont pas commités, ils ne sont pas recherchable
- Optimize : committe + optimisation de l'index (merge de segments)
- Aucune de ces opérations ne bloque la recherche
- Un *commit* ou un *optimize* peuvent se faire via une recherche et 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.xml"
```

```
-H "Content-Type: text/xml" --data-binary
```

```
@myexporteddata.xml
```

Opérations bulk

Il est possible de faire plusieurs opérations en une seule requête:

```
<update>
  <add>
    <doc><!-- doc 1 content --></doc>
  </add>
  <add>
    <doc><!-- doc 2 content --></doc>
  </add>
  <delete>
    <id>0002166313</id>
  </delete>
</update>
```

Format de réponse

La réponse fournit 2 informations :

- Status : == 0 OK, != 0 NOK
- Le temps de traitement en ms

```
<response>  
<lst name="responseHeader">  
<int name="status">0</int>  
<int name="QTime">127</int>  
</lst>  
</response>
```

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. Tableau JSON ou chaque élément représente un document
 - 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

Ajout d'un seul document

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

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"
}]'
```

Curl JSON : Examples

Indexation commande

```
curl -X POST -H 'Content-Type: application/json'
'http://localhost:8983/solr/my_collection/update' --data-binary '
{
  "add": {
    "doc": {
      "id": "DOC1",
      "my_field": 2.3,
      "my_multivalued_field": [ "aaa", "bbb" ]
    }
  },
  "add": {
    "commitWithin": 5000,
    "overwrite": false,
    "doc": {
      "f1": "v1",
      "f1": "v2"
    }
  },
  "commit": {},
  "optimize": { "waitSearcher":false },
  "delete": { "id":"ID" },
  "delete": { "query":"QUERY" }
}'
```


Transformation JSON

- Plusieurs opérations peuvent être effectuées sur le json d'entrée :
 - Le paramètre **split** permet de découper un JSON d'entrée en plusieurs documents SolR
 - **f** : permet de faire du mapping entre les attributs json et le nom du champ SolR
 - **mapUniqueKey**, **df** ne mappe que la clé et tout le contenu json est dans le champ df
 - **srcField** : Permet de stocker le json complet dans un champ du document

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}
  ]
}'
```

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

Exemple curl

- curl

```
'http://localhost:8983/solr/techproducts/  
update?commit=true' --data-binary  
@example/exampledocs/books.csv -H  
'Content-type:application/csv'
```

Example SolrJ

```
final String solrUrl = "http://localhost:8983/solr";  
final SolrClient client = new HttpSolrClient.Builder(solrUrl)  
    .withConnectionTimeout(10000)  
    .withSocketTimeout(60000)  
    .build();
```

```
final SolrInputDocument doc = new SolrInputDocument();  
doc.addField("id", UUID.randomUUID().toString());  
doc.addField("name", "Amazon Kindle Paperwhite");
```

```
final UpdateResponse updateResponse =  
client.add("techproducts", doc);  
// Indexed documents must be committed  
client.commit("techproducts");
```

Indexation

Analyseurs de texte

API d'indexation

Compléments

Update Processor Chain

Documents bureautique

Base de données

Near Real Time

- Le « recherchabilité » d'un document est contrôlée par les commits. 2 commits peuvent être effectués par Solr
 - **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*
- 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

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 schéma 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>
```

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"
      }
    ]
  }
]
```

Indexation

Analyseurs de texte

API d'indexation

Compléments

Update Processor Chain

Documents bureautique

Base de données

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 une valeur, éviter une mise à jour si certaines conditions ne sont pas respectées, détecter une langue, ...

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
- Dans les conditions normales, un *UpdateRequestProcessor* appelle le prochain élément de la chaîne 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 nœud dans le cas d'un cloud
 - ***RunUpdateProcessorFactory*** : Exécute les mises à jour en utilisant l'API interne de Solr

```
<updateRequestProcessorChain
  name="add-unknown-fields-to-the-schema"
  default="{update.autoCreateFields:false}"
  processor="uuid,remove-blank,field-name-mutating,parse-boolean,parse-
long,parse-double,parse-date,add-schema-fields">
  <processor class="solr.LogUpdateProcessorFactory"/>
  <processor class="solr.DistributedUpdateProcessorFactory"/>
  <processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
```

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="overwriteDupes">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 3 implémentations :
 - ***Tika***
 - ***LangDetect***
 - ***OpenNLP***

configuration minimale

```
<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>
```

Indexation

Analyseurs de texte API
d'indexation
Update Processor Chain
Documents bureautique
Base de données

Introduction extensions

- SolR propose 2 extensions :
 - ***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)
- Pour les utiliser, les librairies doivent être chargées dans *solrconfig.xml* via une balise ***<lib>***

Solr Cell

Solr utilise Apache Tika pour intégrer différents formats de fichiers.

Le handler ***ExtractingRequestHandler*** utilise Tika pour extraire le texte des documents bureautiques avant indexation.

Pour le mettre en place, il faut placer des librairies additionnelles dans le classpath

```
<lib dir="${solr.install.dir:../../../../..}/contrib/extraction/lib"
regex=".*\.jar" />
<lib dir="${solr.install.dir:../../../../..}/dist/"
regex="solr-cell-\d.*\.jar" />
```

Concepts clés

- Tika essaie de déterminer **automatiquement le type** du document. Il est possible de positionner le mime-type explicitement avec le paramètre *stream.type*
- Tika produit un **flux XHTML** qui alimente un parseur SAX : *ContentHandler*. SolR répond aux événements SAX et crée les champs à indexer. Il est possible d'implémenter son propre *ContentHandler*
- Il est possible de fournir une expression **XPath** afin de restreindre le contenu

Concepts clés (2)

- Tika produit des méta-données comme le **Titre**, le **sujet** et **l'auteur** selon les spécifications des formats. Il est possible de les associer aux champs SolR et de leur affecter un facteur de boost
- Tika ajoute tout le texte du fichier au champ **content**
- Il est possible de surcharger les valeurs parsées par Tika via ses **propres valeurs**

Quelques paramètres disponibles

- ***capture*** : Captures des éléments XHTML afin de les ajouter à des champs séparés du document
- ***captureAttr*** : Indexation des attributs
- ***extractFormat*** : Par défaut XML mais peut également être text
- ***fmap.source_field*** : Associe un champ du document à un champ Solr
fmap.content=text
- ***literal.fieldname*** : Fixe la valeur d'un champ
- ***literalsOverride*** : Si true (défaut), *literal.fieldName* écrase la valeur éventuelle
- fournie par le document. ***uprefix*** : préfixe tous les champs qui ne sont pas connus du schéma

Examples

Indexation

```
curl 'http://localhost:8983/solr/techproducts/  
update/extract?literal.id=doc1&commit=true'  
-F "myfile=@example/exampledocs/solr-  
word.pdf"
```

```
bin/post -c techproducts  
example/exampledocs/solr-word.pdf -params  
"literal.id=a"
```

Recherche

```
http://localhost:8983/solr/techproducts/select?q=  
pdf
```

Exemples avancés

- Capture les tags <div>, les fait correspondre au champ foo_t et le booste par 3

```
bin/post -c techproducts
example/exampledocs/sample.html -params
"literal.id=doc3&captureAttr=true&defaultField=_text
_&capture=div&fmap.div=foo_t&boost.foo_t=3"
```

- Utilisation de littéral

```
bin/post -c techproducts -params
"literal.id=doc4&captureAttr=true&defaultField=text&captur
e=div&fmap.div=foo_t&boost.foo_t=3&literal.blah_s=Bah"
example/exampledocs/sample.html
```

- XPath

```
bin/post -c techproducts -params
"literal.id=doc5&captureAttr=true&defaultField=text&captur
e=div&fmap.div=foo_t&boost.foo_t=3&xpath=/xhtml:html/
xhtml:body/xhtml:div//node()"
example/exampledocs/sample.html
```

Extraire sans indexer

Il est possible d'extraire les données du document sans les indexer. Cela permet de tester l'extraction et d'adapter le mapping

```
curl "http://localhost:8983/solr/techproducts/update/extract?  
&extractOnly=true"  
--data-binary @example/exampledocs/sample.html -H  
'Content-type:text/html'
```

```
bin/post -c techproducts -params  
"extractOnly=true&wt=ruby&indent=true" -out yes  
example/exampledocs/sample.html
```

Mise en place

- La mise en place de Tika consiste à :
 - Ajouter les librairies dans le classpath :

```
<lib dir="${solr.install.dir:../../..}/contrib/extraction/lib" regex=".*\.jar" />  
<lib dir="${solr.install.dir:../../..}/dist/" regex="solr-cell-.*\.jar" />
```

- Configurer le request handler pointant vers
[solr.extraction.ExtractingRequestHandler](#)

```
<requestHandler name="/update/extract" startup="lazy"  
  class="solr.extraction.ExtractingRequestHandler" >  
  <lst name="defaults">  
    <str name="lowernames">true</str>  
    <str name="fmap.content">_text_</str>  
  </lst>  
</requestHandler>
```


Indexation

Analyseurs de texte
API d'indexation
Update Processor
Chain
Documents
bureautique
Base de données

Data Import Handler

Le **Data Import Handler (DIH)** fournit un mécanisme pour importer du contenu à partir d'un support persistant

Il est capable d'indexer du contenu structuré à partir d'une base de données relationnelles, des flux RSS ou ATOM, des e-mails ou du contenu XML extrait via *XPath*

Un exemple est disponible :

```
bin/solr -e dih
```

Pour l'utiliser, il est nécessaire de charger le plugin dans *solrconfig.xml*:

```
<lib dir="${solr.install.dir:../../../../..}/dist/" regex="solr-dataimporthandler-.*\.jar" />
```

Concepts

- *DIH* utilise plusieurs concepts :
 - ***Datasource*** : Emplacement du data store
 - ***Entity*** : Une entité génère un ensemble de documents avec plusieurs champs. Pour
 - une base de données : une vue ou une table
 - ***Processor*** : Responsable d'extraire le contenu et de l'indexer. Il est possible de fournir ses propres *processors* qui surchargent ceux fournis par défaut
 - ***Transformer*** : Chaque champs de l'entité peuvent être transformés. *ApacheSolR* fournit des *transformers* et il est possible d'écrire ses propres transformers

Configuration

Le request handler *DIH* doit être configuré dans *solrconfig.xml*

Le seul paramètre nécessaire est ***config*** qui donne le chemin vers le fichier de configuration spécifique du DIH

```
<requestHandler name="/dataimport"  
  class="org.apache.solr.handler.dataimport.DataImportHandler">  
  <lst name="defaults">  
    <str name="config">/path/to/my/DIHconfigfile.xml</str>  
  </lst>  
</requestHandler>
```

Configuration RDMS

- La configuration de la datasource consiste à :
 - Ajouter les **drivers** jdbc dans le classpath
 - Fournir **l'URL jdbc**
 - Les attributs **autocommit** et **batch-size**
 - Un élément **document** contenant plusieurs balises **entity** potentiellement imbriquées permettant de suivre les relations de la base
 - Les éléments **entity** contiennent des éléments **field** sur lesquels peuvent être appliqués des transformers
 - Ils contiennent des attributs de requête :
 - **query** : Requête permettant de récupérer **toutes** les entités et leurs champ associés
 - **delta-query** : Requête permettant de récupérer les **nouvelles** entités et leurs champ associés
 - **parentDelta-query** : Requête de jointure entre 2 entités liées

Example

```
<dataConfig>
  <dataSource driver="org.hsqldb.jdbcDriver"
url="jdbc:hsqldb:./example-DIH/hsqldb/ex" user="sa" />
  <document>
    <entity name="item" query="select * from item"
      deltaQuery="select id from item where last_modified >
        '${dataimporter.last_index_time}'">
      <field column="NAME" name="name" />
      <!-- One to Many -->
      <entity name="feature"
        query="select DESCRIPTION from FEATURE where
          ITEM_ID='{item.ID}'"
        deltaQuery="select ITEM_ID from FEATURE
          where last_modified > '$
            {dataimporter.last_index_time}'"
        parentDeltaQuery="select ID from item where ID=$
          {feature.ITEM_ID}">
        <field name="features" column="DESCRIPTION" />
      </entity>
    </entity>
  </document>
</dataConfig>
```

API Rest : Commandes DIH

Les commandes *DIH* sont lancées via des requêtes HTTP :

- ***full-import*** : Importe toutes les entités
- ***delta-import*** : Importation incrémentale et détection de changement
- ***reload-config*** : Rechargement de la configuration
- ***status*** : Retourne des statistiques (documents créés, ...)

Exemples

Import global

curl

'[http://localhost:8983/solr/db/dataimport?](http://localhost:8983/solr/db/dataimport?command=full-import&entity=item)
command=full-import&entity=item'

Statistiques

curl

'[http://localhost:8983/solr/db/dataimport?](http://localhost:8983/solr/db/dataimport?command=status)
command=status'

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 utilisés 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
 - En utilisant *SolrCloud*

Recherche full-text

Principes

Configuration des handlers

Syntaxes des différents
parseurs

Calcul du score

Introduction

Solr propose un mécanisme de recherche très flexible.

Une requête de recherche est traitée par un ***RequestHandler***

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 d'organiser les résultats de recherche pour faciliter l'exploration et la compréhension des données.
- 2 approches principales existent :
 - Les **facettes** regroupent les résultats selon des champs existants du schéma (catégorie, auteur, date...).
 - S'appuient sur les valeurs indexées → regroupement structuré et prévisible.
 - Très utilisé pour la navigation à facettes (filtres latéraux, statistiques, etc.).
- Le **clustering** regroupe les résultats selon leurs similarités de contenu.
 - Analyse les textes des documents retournés pour créer des groupes thématiques.
 - Les regroupements sont découverts automatiquement à partir du contenu, pas des champs.

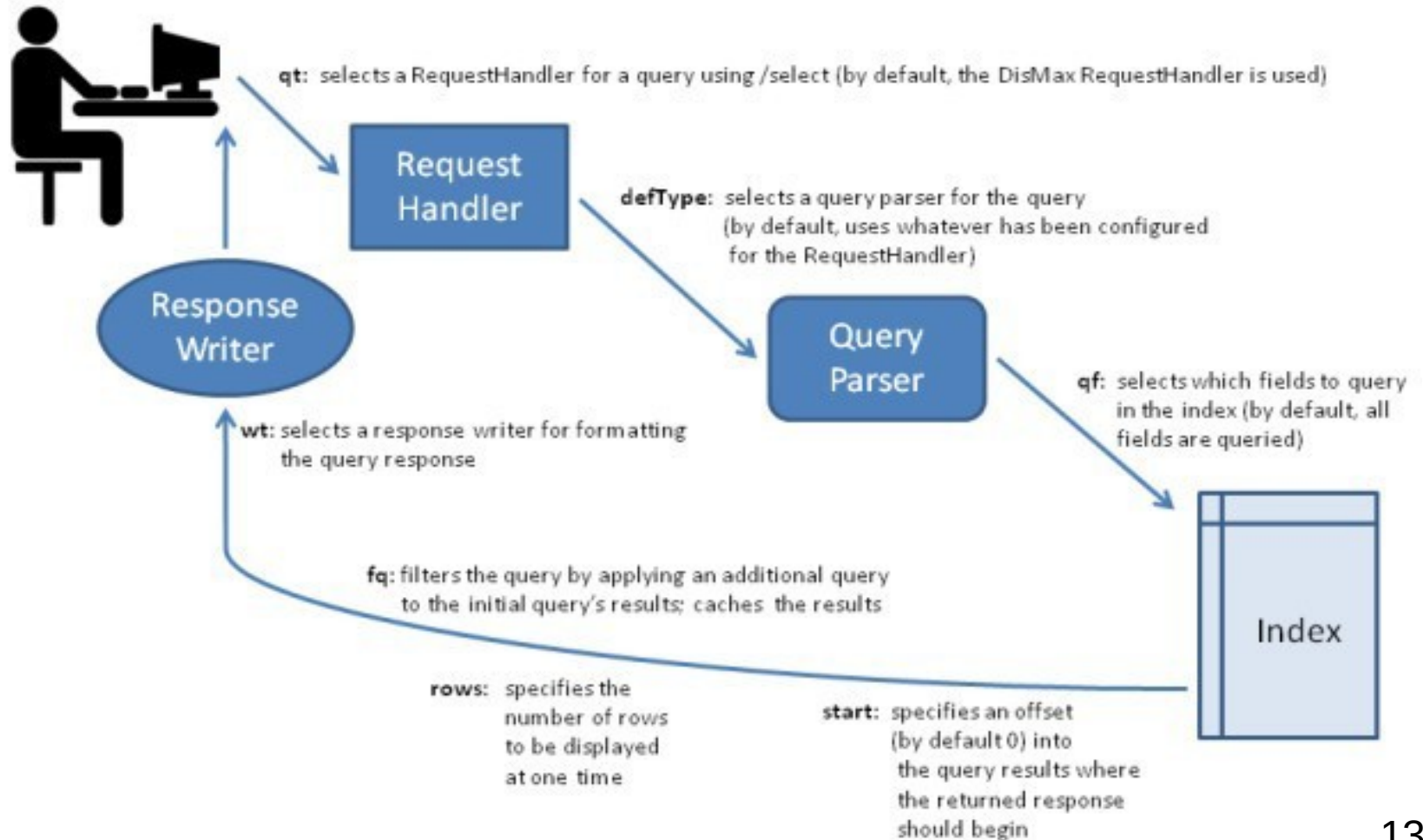
Documents similaires

- Solr propose aussi un mode de recherche par similarité grâce au composant **MoreLikeThis**.
- À partir d'un document donné, Solr cherche d'autres documents similaires.
- Il s'appuie sur les termes les plus représentatifs du document (analyse TF-IDF).
- Utile pour les recommandations ou la suggestion de contenu lié.

Présentation des résultats

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

Big Picture



Réponses

```
{
  "responseHeader":
    { "status":0, // Code
      retour
      "Qtime":0, // Temps
      d'exécution
      "q": "_root_:1",
      "params":{"_":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}]
  }}
```

Recherche full-text

Principes

Configuration des handlers

Syntaxes des différents parseurs

Calcul du score

Introduction

- La configuration consiste en :
 - Un endpoint de recherche :
requestHandler name="/search" class="solr.SearchHandler"
 - Des paramètres par défaut / invariants : defType, qf, pf, mm, rows, fl, sort, etc.
 - Les composants de recherche utilisés : <searchComponent/> et leur ordre via <arr name="components">.

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

Handler par défaut

- */select* : Handler générique permettant de dispatcher vers un autre requestHandler via le paramètre *qType*
- */query* : Format JSON indenté, champ par défaut *text*

Paramètres de recherche communs

- La requête

q (query) : la requête full-text parsé différemment en fonction des parser

fq (filterQuery) : Le filtre éventuel (~ WHERE en SQL)

defType : Le type de parseur à utiliser

- Pagination

rows : Nombre de résultats

start : Offset de départ de la liste

- Sortie

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

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
 - **facet_module** : JSON facetting
 - **mlt** : MoreLikeThis.
 - **highlight** : Surbrillance
 - **stats** : Génère des statistiques
 - **expand** : Collapse et expand
 - **terms** : Termes
 - **debug** : Debug

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>
```

Recherche full-text

Principes

Configuration des handlers

Syntaxes des différents parseurs

Calcul du score

Syntaxes 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, corrige les typos)

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

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`

Recherche full-text

Principes
Configuration des handlers
Syntaxes des différents
parseurs
Calcul du score

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 du déploiement SolR de spécifier les types de réponses que l'application doit retourner pour des exemples de requêtes.
 - On affine alors la configuration pour obtenir les résultats voulus

Le score et le tri

- Champ spécial « *score* »
on peut le ramener avec le
paramètre
« ***fl = score*** »
- « *sort* » se fait sur ce pseudo-
champ 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* : **$\text{sqrt}(\text{freq})$** $= >$ Plus le terme apparaît dans le document plus le score est élevé
- *idf* : **$\log(\text{numDocs}/(\text{docFreq}+1)) + 1$** $= >$ Plus le terme est présent dans l'index moins le score est élevé
- *coord* : **$\text{overlap} / \text{maxOverlap}$** $= >$ Plus il y a de termes plus le score est élevé
- *lengthNorm* : **$1/\text{sqrt}(\text{numTerms})$** $= >$ 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={!frangel=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"`

Recherche full-text

Principes

Configuration des handlers

Calcul du score

Syntaxes des différents parseurs

API Json

Introduction

- Solr supporte (depuis ses versions récentes) une alternative pour la recherche basée sur JSON
- Les paramètres de la requête peuvent alors être fournis dans un corps JSON :

```
curl http://localhost:8983/solr/techproducts/query -d '{
  "query" : "memory",
  "filter" : "inStock:true"
}'
```

Correspondance

- Seulement certains paramètres de requêtes sont actuellement supportés :

Paramètre	Champ JSON équivalent
q	query
fq	filter
start	offset
rows	limit
fl	fields
sort	sort
json.facet	facet
json.<param>	param
json.queries.<query>	queries

Query DSL

- L'API Json accepte des valeurs pour l'élément *query* sous 3 formats différents :
 - Une chaîne de caractères utilisant le parseur par défaut (defType)
`title:solr`
 - Une chaîne de caractère spécifiant le parseur :
`{!dismax qf=title}`
 - Un objet JSON spécifiant le parseur et les paramètres additionnels

Examples

- BoolQParser

```
{
  "query": {
    "bool": {
      "must": [
        {"lucene": {"df": "name", query: "iPod"}}
      ],
      "must_not": [
        {"frange": {"l": "0", "u": "5", "query": "popularity"}}
      ]
    }
  }
}
```

- Avec filtre :

```
{
  "query": {
    "bool": {
      "must_not": "{!frange l=0 u=5}popularity"
    }
  },
  "filter": [
    "name:iPod"
  ]
}
```

Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

Auto-complétion

Recherche géographique

Élévation

Highlight : paramètres

- **hl=true**
 - Active le highlight
- **hl.fl=<field1>,<field2>,<fieldN>**
 - Liste des champs à highlighter
 - Par défaut, liste des champs dans le paramètre « qf » de edismax
- **hl.usePhraseHighlighter=true**
 - Fait en sorte qu'une « phrase query » comme « a b c » highlighte « b » seulement dans un morceau de texte « a b c »
- Beaucoup d'autres options avancées existent, voir

<http://wiki.apache.org/solr/HighlightingParameters>

Highlight : résultats

- Le highlight est un composant à part entière, et ses résultats sont donc séparés de la liste de résultat des documents en tant que telle !

Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

Auto-complétion

Recherche géographique

Élévation

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;

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=<chaîne>** (advanced)
 - Ne garde que les valeurs de facettes qui commencent par la chaîne indiquée
 - Utilisée pour des facettes hiérarchiques ou de l'autocomplétion

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.

Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

Auto-complétion

Recherche géographique

Élévation

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)

Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

Auto-complétion

Recherche géographique

Élévation

Spellcheck : configuration du schema

- « Voulez-vous dire... ? »
- Les suggestions orthographiques doivent se baser sur un dictionnaire :
 - Un fichier texte
 - Ou bien les valeurs d'un champ de l'index (recommandé)
 - En général, un champ spécifiquement dédié à cela et rempli via des *copyField*
 - stored=false
 - Pas de stemming
 - lowercasing

Spellcheck : configuration du component

- Le component « *spellcheck* » existe déjà dans *solrconfig.xml*. Ses options :
 - *name* : un nom de configuration
 - *classname* :
 - ***solr.DirectSolrSpellchecker*** : Basé directement sur un index Solr
 - ***solr.IndexBasedSpellChecker*** : Basé sur un index *Solr* et crée un autre index dédié pour le spellchecking
 - ***solr.FileBasedSpellChecker*** : Basé sur un fichier externe

Configuration DirectSolr

- ***field*** : nom du champ dans lequel lire les valeurs
- ***distanceMeasure*** : Le mode de calcul de la distance entre 2 mots (internal = Levenshtein)
- ***accuracy*** : Le seuil de distance pour la suggestion
- ***thresholdTokenFrequency*** : entre 0 et 1; pour exclure les termes qui n'apparaissent pas souvent (défaut : 0, tous les termes. A priori 0.01 élimine les mots bizarres)
- ***maxEdits*** : nombre de lettres de différence autorisé (1 ou 2) pour effectuer la requête
- ***minPrefix*** : nombre de lettres communes au début du terme
- ***minQueryLength*** : nombre mini de lettres dans la recherche pour déclencher un suggest

Example :DirectSolr

```
<searchComponent name="spellcheck"
class="solr.SpellCheckComponent">
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="field">name</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="distanceMeasure">internal</str>
    <float name="accuracy">0.5</float>
    <int name="maxEdits">2</int>
    <int name="minPrefix">1</int>
    <int name="maxInspections">5</int>
    <int name="minQueryLength">4</int>
    <float name="maxQueryFrequency">0.01</float>
    <float name="thresholdTokenFrequency">.01</float>
  </lst>
</searchComponent>
```

Exemple : *IndexBase*

- ```
<searchComponent name="spellcheck"
class="solr.SpellCheckComponent">
 <lst name="spellchecker">
 <str name="classname">solr.IndexBasedSpellChecker</str>
 <!-- Emplacement de l'index dédié spellecheck -->
 <str name="spellcheckIndexDir">./spellchecker</str>
 <!-- Champ utilisé pour créer l'index
 (Ne pas prendre un champ avec trop de traitement de termes /
 synonym, stem -->
 <str name="field">content</str>
 <!-- Reconstruction de l'index lors de l'ajout de doc. -->
 <str name="buildOnCommit">true</str>
 </lst>
</searchComponent>
```

# Exemple : *FileBase*

```
<searchComponent name="spellcheck"
class="solr.SpellCheckComponent">
 <lst name="spellchecker">
 <str name="classname">solr.FileBasedSpellChecker</str>
 <str name="name">file</str>
 <!-- Source du fichier -->
 <str name="sourceLocation">spellings.txt</str>
 <str name="characterEncoding">UTF-8</str>
 <str name="spellcheckIndexDir">./spellcheckerFile</str>
 </lst>
</searchComponent>
```



# Correcteur d'espaces

- Un autre correcteur orthographique nommé ***WordBreakSolrSpellChecker*** permet de corriger les mauvais espaces (supprimer ou ajouter)

```
<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
<lst name="spellchecker">
<str name="name">wordbreak</str>
<str name="classname">solr.WordBreakSolrSpellChecker</str>
<str name="field">lowerfilt</str>
<str name="combineWords">true</str>
<str name="breakWords">true</str>
<int name="maxChanges">10</int>
</lst>
</searchComponent>
```

# Spellcheck : configuration du handler

- Le spellcheck component est normalement toujours ajouté à un *requestHandler* existant (jamais appelé séparément)

```
<requestHandler name="/myHandler" class="solr.SearchHandler">
 <lst name="defaults">
 <!-- Active le spellchecking -->
 <str name="spellcheck">true</str>
 <!-- Nom du spellchecker (attribut "name") -->
 <str name="spellcheck.dictionary">default</str>
 <!-- Nombre de propositions par mot -->
 <str name="spellcheck.count">1</str>
 </lst>
 <arr name="last-components">
 <str>spellcheck</str>
 </arr>
</requestHandler>
```

# Spellcheck : paramètres de query

- **spellcheck=true** : active le spellcheck
- **spellcheck.dictionary** : doit correspondre à « name » du component
- **spellcheck.count** : nombre de suggestion à retourner
- **spellcheck.onlyMorePopular** : Retourne seulement des termes de l'index + fréquents que le terme d'origine
- **spellcheck.extendedResults** : Retourne les informations de fréquence des termes (pour du debug)
- **spellcheck.collate=true** : dans le cas de query multi-termes, retourne en plus une suggestion de query complète agrégeant les suggestions pour chaque terme
  - **spellcheck.maxCollations=1** nombre max de suggestions de collations
  - **spellcheck.maxCollationsTries=0** si >0, nombre d'essais pour voir si la suggestion de collation donne des résultats (mettre à 5)
  - **spellcheck.collateExtendedResults** : ajoute des détails, pour du debug

# Spellcheck : résultats

```
<response>...
<lst name="spellcheck">
 <lst name="suggestions">
 <lst name="dell">
 <int name="numFound">1</int>
 <int name="startOffset">18</int><int name="endOffset">23</int>
 <int name="origFreq">0</int>
 <arr name="suggestion"><lst>
 <str name="word">dell</str>
 <int name="freq">2</int>
 </lst></arr>
 </lst>
 <lst name="ultrashar">
 <int name="numFound">1</int>
 <int name="startOffset">24</int><int name="endOffset">33</int>
 <int name="origFreq">0</int>
 <arr name="suggestion"><lst>
 <str name="word">ultrasharp</str>
 <int name="freq">2</int>
 </lst></arr>
 </lst>
 <bool name="correctlySpelled">false</bool>
 <str name="collation">price:[80 TO 100] dell ultrasharp</str>
 </lst>
</lst>
</response>
```

# Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

**Auto-complétion**

Recherche géographique

Élévation

# Plusieurs types d'auto-complétion

## 1. Recherche instantanée

Chaque proposition correspond à un résultat de recherche. Utilisation de l'analyseur N-Gram

## 2. Suggestion de recherche basée sur les logs

Chaque proposition correspond à une recherche fréquemment effectuée (à la Google)

## 3. Suggestion de recherche basée sur l'index

Chaque proposition correspond à un terme présent dans l'index

## 4. Suggestion de valeur de facette

Chaque proposition correspond à une valeur de facette possible (avec le nom de la facette indiquée)

# Suggestions basées sur l'index :

## solution 1 : *facet.prefix*

- Les facettes peuvent être utilisées pour implémenter l'autocomplete
- Attention, prends beaucoup de mémoire !
- Recherche « michael ja »
  - Les premiers mots de la requête sont pris comme une query normale
  - q=michael
  - On facette sur le champ texte :
  - facet=on
  - facet.field=text
  - facet.limit=5 (on veut 5 propositions)
  - facet.mincount=1 (on ne veut que des termes pour lesquels il y a effectivement un résultat)
  - facet.sort=count (le défaut) pour avoir les termes les plus utilisés au début de la liste
  - Le dernier mot incomplet est pris comme préfixe de facette, on ne veut que les valeurs de facette qui commence par ces lettres
  - facet.prefix=ja
  - On ne veut retourner que les résultats des facettes, pas les documents résultats !
- rows=0

# Suggestions basées sur l'index :

## solution 1 : facet.prefix

[../solr/core01/select/?](#)

[q=michael&facet=on&rows=0&facet.limit=5&facet.mincount=1&facet.field=text  
&facet.sort=count&facet.prefix=ja&qt=myHandler&wt=json&indent=on&](#)

```
{ "responseHeader" : {
 "status":0,
 "QTime":5},
 "response":{"numFound":2498,"start":0,"docs":[]},
 "facet_counts":{"
 "facet_queries":{},
 "facet_fields": {
 "text":[
 "jackson",18,
 "james",16,
 "jason",4,
 "jay",4,
 "jane",3]
 }
 }
 }
}
```



# Suggestions basées sur l'index :

## solution 2 : Suggester

- Suggester : basé sur le component « Spellcheck »
  - Désavantage par rapport à *facet.prefix* : propose tous les termes, pas uniquement ceux valides par rapport au début de la recherche
- Nécessite de définir un nouveau « component », qu'il faut ajouter dans un handler
- Peut lire ses valeurs depuis un fichier (mais attention aux problèmes de caractères accentués)
- Voir <http://wiki.apache.org/solr/Suggester>

# Suggestions basées sur l'index : solution 2 : Suggester

```
<searchComponent name="suggest" class="solr.SpellCheckComponent">
 <lst name="spellchecker">
 <str name="name">suggest</str>
 <str name="classname">org.apache.solr.spelling.suggest.Suggester</str>
 <str name="lookupImpl">org.apache.solr.spelling.suggest.tst.FSTLookupFactory</str>
 <str name="field">name</str> <!-- nom du champ d'index -->
 <float name="threshold">0.005</float>
 <str name="buildOnCommit">true</str>
 <!-- <str name="sourceLocation">chemin-vers-un-fichier.txt</str> -->
 </lst>
</searchComponent>

<requestHandler name="/suggest" class="org.apache.solr.handler.component.SearchHandler">
 <lst name="defaults">
 <str name="spellcheck">true</str>
 <str name="spellcheck.dictionary">suggest</str>
 <str name="spellcheck.onlyMorePopular">true</str>
 <str name="spellcheck.count">5</str>
 <str name="spellcheck.collate">true</str>
 </lst>
 <arr name="components">
 <str>suggest</str>
 </arr>
</requestHandler>
```

# Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

Auto-complétion

**Recherche  
géographique**

Élévation

# 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* .  
Format 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 séparées 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 incluses dans un cercle autour d'un point d'origine
 - **bbox** retourne les documents dont les coordonnées géographiques sont incluses 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

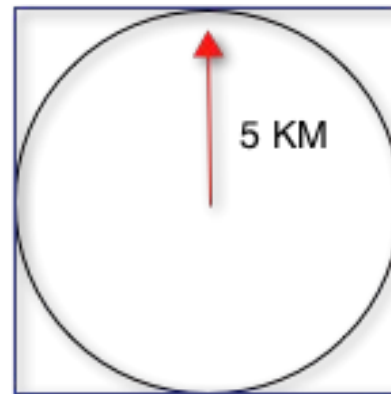
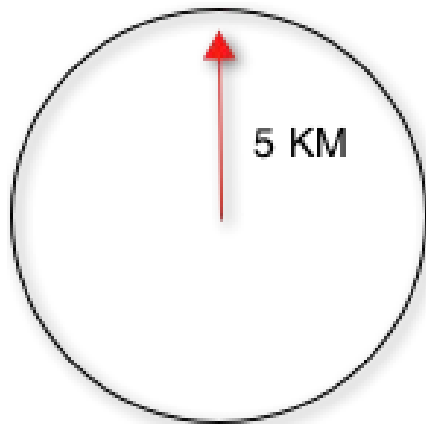
Exemples

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))
```

Fonctionnalités de recherche

Surbrillance

Recherche par facette

Recherche par groupe

Spell check

Auto-complétion

Recherche géographique

Élévation

Component

« QueryElevation »

- Consiste à tuner query par query les résultats en remontant certains documents au début de la liste
- Ne remplace pas une bonne configuration d'index ! Scénarios :
 - Corrections d'erreurs évidentes sur des recherches populaires
 - Mots-clés payants pour un site de pages jaunes
 - Choix éditoriaux pour un site de news

QueryElevation : component

```
<searchComponent name="elevator"
class="org.apache.solr.handler.component.QueryElevationComponent">
  <!-- fieldType pour comparer 'q' au fichier de config -->
  <str name="queryFieldType">text</str>
  <!-- Référence au fichier de config dans conf -->
  <str name="config-file">elevate.xml</str>
  <!-- Force l'élévation indépendamment du param 'sort' -->
  <str name="forceElevation">true</str>
</searchComponent>
```

```
<requestHandler name="/elevate" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="echoParams">explicit</str>
  </lst>
  <arr name="last-components">
    <str>elevator</str>
  </arr>
```

```
</requestHandler>
```

QueryElevation : fichier de config

```
<elevate>
  <query text="education">
    <doc id="A" />
    <doc id="B" />
    <doc id="C" exclude="true" />
  </query>

  <query text="africa">
    <doc id="D" />
  </query>

  <!-- etc... -->
</elevate>
```

Solr Cloud

Concepts

Installation Zookeeper

Installation nœud

Upgrade

Sécurisation

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 possible de l'intégralité du cluster
 - Équilibrage de charge et fail-over pour les requêtes
 - Intégration de *ZooKeeper* pour la coordination et la configuration des nœuds

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 des données *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 nœud 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 nœud
 - 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 nœud Solr reçoit une requête de recherche, elle est routée vers une réplique appartenant à la collection.
- Le nœud agit alors comme un agrégateur :
 - Il crée des requêtes internes qu'il route vers une 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 nœuds du cluster, il faut un répartiteur externe sachant interroger et lire les méta-données stockées dans *ZooKeeper*
- *SolrJ* fournit un client Java : ***CloudSolrClient*** qui utilise les données de ZooKeeper pour répartir la charge sur les nœuds 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 nœud 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 changer sans de lourdes conséquences :
Création de nouveau cœurs 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.
 - Un fois l'opération terminée le shard original peut être supprimé

Création de collections

- A la création d'une collection, on indique le nombre de shards et le facteur de réplication :

```
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`

Production

Concepts

Installation Zookeeper

Installation nœud

Upgrade

Sécurisation

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*

Ensemble *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 ensemble de *ZooKeeper* externes qui apporte le fail-over.
- Pour que l'ensemble fonctionne, il lui faut un quorum de serveurs up
=> En général 3 nœuds *ZooKeeper* est un bon chiffre, cela permet la tolérance d'une panne d'un des nœuds

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 synchros

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

Contexte *ZooKeeper*

- 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 connexion à *ZooKeeper*

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

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

Production

Concepts

Installation Zookeeper

Mise en service nœud

Upgrade

Sécurisation

Mise en service

2 packages binaires sont disponibles au téléchargement :

- solr-<version>.tgz : Distribution complète avec tous les modules (comme l'exporteur prometheus par exemple)
- solr-slim-<version>.tgz juste le minimum

2 images dockers associées sont également gérées par SolR

Script d'installation de service

SolR fournit pour CentOS, Debian, Red Hat, SUSE and Ubuntu Linux : **`bin/install_solr_service.sh`**

Avant de l'exécuter choisir :

- Un répertoire pour la distribution.
Par défaut **`/opt/solr`**
- Un répertoire pour l'écriture des données (index, traces, ...).
Par défaut **`/var/solr`**
- Un user pour démarrer le process. Par défaut **`solr`**

Typiquement :

```
sudo bash install_solr_service.sh solr-9.9.0.tgz -i /opt -d /var/solr -u solr -s solr -p 8983
```

```
sudo service solr status
```

Fichiers et répertoires

Répertoires

- /opt/solr*** → installation (binaire)
- /var/solr/data*** → Solr home (cores, indexes)
- /var/solr/logs*** → logs Solr
- /etc/default/solr.in.sh*** → overrides (PID, heap, ZK, ports...)
- /etc/init.d/solr*** → script de démarrage (start/stop/status)

Fichier d'environnement ***/etc/default/solr.in.sh***

```
SOLR_PID_DIR=/var/solr  
SOLR_HOME=/var/solr/data  
SOLR_HEAP="8g"  
SOLR_LOGS_DIR=/var/solr/logs  
LOG4J_PROPS=/var/solr/log4j2.xml
```

Trace si log4j

```
/var/solr/log4j2.xml
```

Optimisation et production

Paramètres système :

Processus : ulimit -u, $\geq 65\ 000$

Descripteurs de fichiers : ulimit -n $\geq 65\ 000$

Mémoire virtuelle / taille : ulimit -v / -m illimité

Swappiness /proc/sys/vm/swappiness 1 (ou désactiver le swap)

Mémoire & GC

SOLR_HEAP="8g"

GC_TUNE, éventuellement le Garbage Collector

Merge scheduler (solrconfig.xml)

```
<mergeScheduler class="ConcurrentMergeScheduler">  
  <int name="maxThreadCount">4</int>  
  <int name="maxMergeCount">9</int>  
</mergeScheduler>
```

SolrCloud

- Pas de Zookeeper embarqué
- La variable ZK_HOST doit pointer vers l'ensemble ZooKeeper
ZK_HOST=zk1,zk2,zk3
OU
ZK_HOST=zk1,zk2,zk3/solr (si zookeeper partagé)

Production

Concepts

Installation Zookeeper

Mise en service nœud

Upgrade

Sécurisation

Plan de migration

- 1) Consultez les notes de mise à niveau de Solr
- 2) Si pas de réplication, sauvegarde de chaque collection.
- 3) Via l'API, déterminez quel nœud Solr héberge le processus Overseer leader, il doit être mis à niveau en dernier.
- 4) Prévoir d'effectuer la mise à niveau pendant une période de maintenance du système si possible même si l'on effectue un redémarrage progressif (chaque nœud, un par un)
- 5) Vérifiez que le cluster est sain et que toutes les répliques sont actives.
- 6) Recompilez et testez tous les composants serveur personnalisés avec les nouveaux fichiers JAR de Solr.
- 7) Déterminez les valeurs des variables utilisées par les scripts de contrôle Solr : ZK_HOST, SOLR_HOST, SOLR_PORT, SOLR_HOME

Processus de migration

L'approche consiste à upgrader nœud par nœud. Pendant une période transitoire il y a des nœuds avec des versions différentes. Les répertoires d'index ne bougent pas.

- 1) Arrêter le nœud, si réplication attendre que les répliques hébergées par ce nœud aient migré
- 2) Installer Solr comme service
- 3) Mettre à jour les variables de `/etc/default/solr.in.sh`
- 4) Démarrer Solr
- 5) Vérifier avec le healthcheck API

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>

Production

Concepts

Installation Zookeeper

Mise en service nœud

Upgrade

Sécurisation

Sécurisation

- Pas mal de support :
 - L'authentification et l'autorisation peuvent être configurés avec différents plugins
 - On doit activer SSL/TLS,
 - Avec SolrCloud, les données de ZooKeeper peuvent être protégées par des ACLs
- En plus, firewall et écoute sur une interface (par défaut seulement localhost)
 - SOLR_JETTY_HOST=

SolrCloud

- Pas de Zookeeper embarqué
- La variable ZK_HOST doit pointer vers l'ensemble ZooKeeper
ZK_HOST=zk1,zk2,zk3
OU
ZK_HOST=zk1,zk2,zk3/solr (si zookeeper partagé)

Approches liées à la sécurité

- Protection via contrôle IP :
 - Propriétés : *SOLR_IP_WHITELIST/SOLR_IP_BLACKLIST*
- Activer TLS (SSL) pour les communications avec le client
 - *SOLR_SSL_ENABLED*
- Authentifier les utilisateurs. Plugins d'authentification :
 - Kerberos, Basic, Hadopp
 - JWT authentication plugin (OpenID Connect)
- Mettre des ACLs sur les ressources (Utilisateurs/Rôles/Permissions)
 - Rule-based authorization:
 - External Role (compatible JWT et oAuth2)
- Traces
 - Des requêtes : Propriété *SOLR_REQUESTLOG_ENABLED=true*
 - Des événements sécurité : Plugin Audit Login

Paramètres SSL

```
# Enables HTTPS. It is implicitly true if you set SOLR_SSL_KEY_STORE. Use this config  
# to enable https module with custom jetty configuration.
```

```
SOLR_SSL_ENABLED=true
```

```
# Uncomment to set SSL-related system properties
```

```
# Be sure to update the paths to the correct keystore for your environment
```

```
SOLR_SSL_KEY_STORE=etc/solr-ssl.keystore.p12
```

```
SOLR_SSL_KEY_STORE_PASSWORD=secret
```

```
SOLR_SSL_TRUST_STORE=etc/solr-ssl.keystore.p12
```

```
SOLR_SSL_TRUST_STORE_PASSWORD=secret
```

```
# Require clients to authenticate
```

```
SOLR_SSL_NEED_CLIENT_AUTH=false
```

```
# Enable clients to authenticate  
(but not require)
```

```
SOLR_SSL_WANT_CLIENT_AUTH=false
```

```
# SSL Certificates contain  
host/ip "peer name" information  
that is validated by default.  
Setting
```

```
# this to false can be useful to disable these checks when re-using a certificate on many  
hosts
```

```
SOLR_SSL_CHECK_PEER_NAME=true
```

security.json

- Les plugins se configurent via un fichier ***security.json***
 - placé dans *SOLR_HOME* en mode standalone
 - Uploadé dans Zookeeper pour SolrCloud
- Ex : *BasicAuth* et *RuleBased*

```
{
  "authentication":
    { "class":"solr.BasicAuthPlugin
      ",
      "credentials":{"solr":"IV0EHq10nNrj6gvRCwvFwTrZ1+z1oBbnQdiVC3otuq0=
Ndd7LKvVBAAzIF0QAVi1ekCfAJXr1GGfLtRUXhgrF8c="}
    },
  "authorization":{
    "class":"solr.RuleBasedAuthorizationPlugin",
    "permissions":[{"name":"security-edit",
      "role":"admin"}],
    "user-role":{"solr":"admin"}
  }
}
```

Production

Concepts

Installation Zookeeper

Mise en service nœud

Upgrade

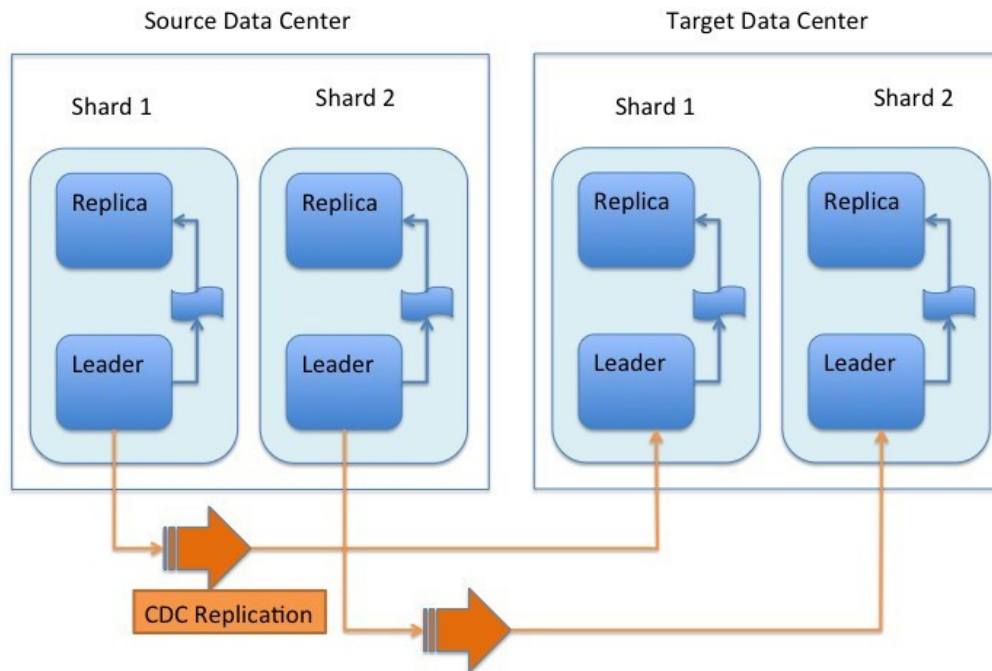
Sécurisation

Cross Data Center

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>
  </updateLog>
  <!--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*)