



INSTITUT
Mines-Télécom

Projet NoSQL *3 février 2016*

Wiki Trending & Rising Topics

AURELIEN

GUILLAUME

JEAN-BAPTISTE

JADE





Agenda

- Organisation du projet
- Architecture logicielle - flux de traitement
- Tables
- Infrastructure
- Demo data visualisation
- Lessons and Learns

Organisation du projet

■ Travail en équipe

- [Bitbucket](#): *partage des codes sources*
- [Google docs](#): *création d'une documentation (work-flow)*
- **Slack**: *Communication*

■ Infrastructure

- **Cassandra-pySpark** : 6 Noeuds EC2
- **MongoDB** : tests en local sur un noeud & client python


■ Datavisualisation

- **D3.js** : *Dashboard web dynamique*
- **iPython Notebook**: *Dashboard connecté à Cassandra*

Architecture logicielle - flux de traitement

Principes généraux:

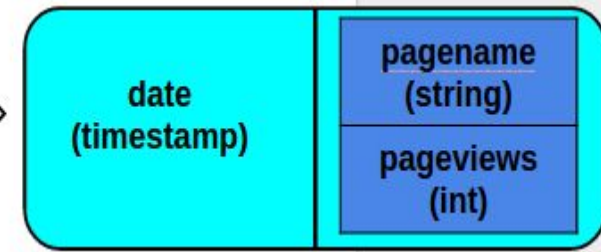
- Utilisation majoritaire de l'API PySpark DataFrame avec connecteur Cassandra
- Construire une table par type de requête nécessaire à la visualisation



| tâche(s) | API(s) | table cible |
|---|--|--------------------------------------|
| Ingestion des données depuis s3 et filtrage | PySpark DataFrame <code>sc.textFile(...).map(...).toDf().filter()...</code> | 150 Gb hrpageviews (1,5 Gb) |
| Aggrégation par jour | PySpark Dataframe <code>sqlContext.sql().groupBy(). ...</code> | dapageviews |
| Calcul des TOP100 | PySpark Dataframe <code>sqlContext.sql().groupBy(). ...</code> | top100rising, top100trending |
| Extraction des historiques (-24h, -30j) pour les TOP100 | PySpark Dataframe <code>sqlContext.sql().groupBy(). ...</code> | top100risinghist, top100trendinghist |
| Visualisation | cassandra-driver + matplotlib /D3js | |

Tables

| pagecounts-20090430-230000 | | | |
|----------------------------|----------|-----------|------|
| projectcode | pagename | pageviews | byte |



```
CREATE TABLE wikistats.hrpageviews ( event_time timestamp, title text, pageviews int, PRIMARY KEY ((event_time), title));
```

```
CREATE TABLE wikistats.dapageviews ( event_time timestamp, title text, pageviews int, PRIMARY KEY ((event_time), title));
```

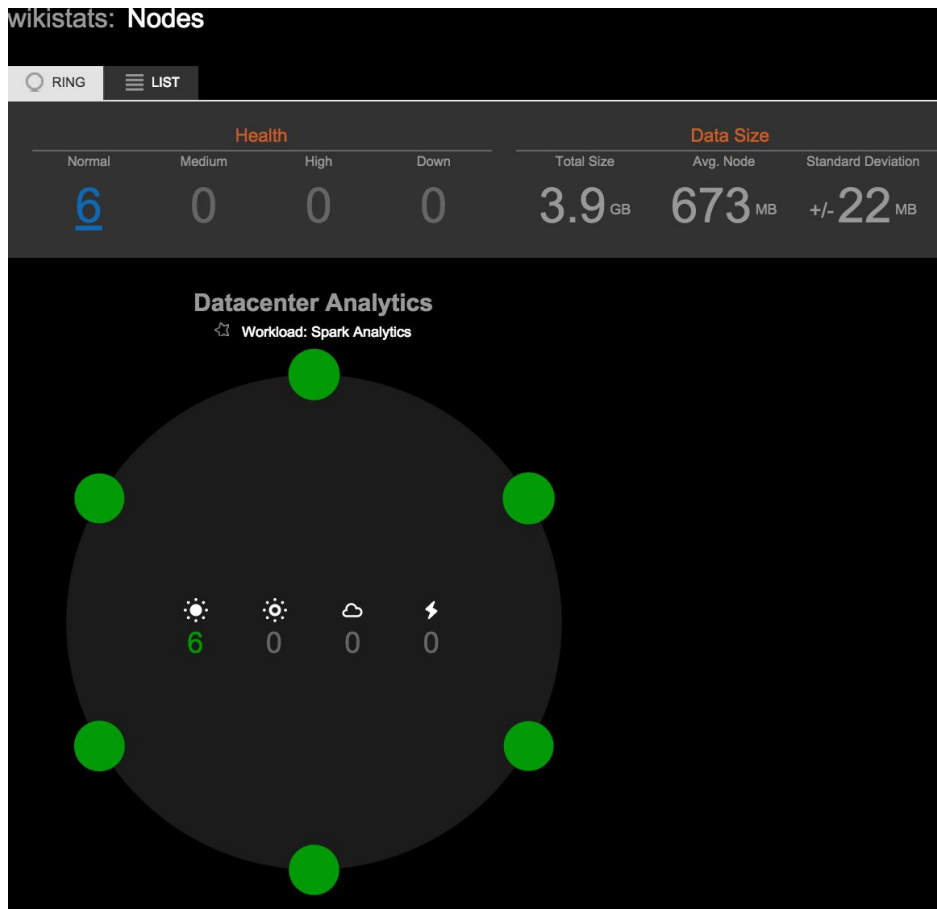
```
CREATE TABLE wikistats.top100trending ( event_time timestamp, pageviews int, title text, PRIMARY KEY ((event_time), title));
```

```
CREATE TABLE wikistats.top100rising ( event_time timestamp, pageviews int, title text, PRIMARY KEY ((event_time), title));
```

```
CREATE TABLE wikistats.top100trendinghist ( refhour timestamp, title text, event_time timestamp, pageviews int, PRIMARY KEY ((refhour, title), event_time)) WITH CLUSTERING ORDER BY (event_time ASC);
```

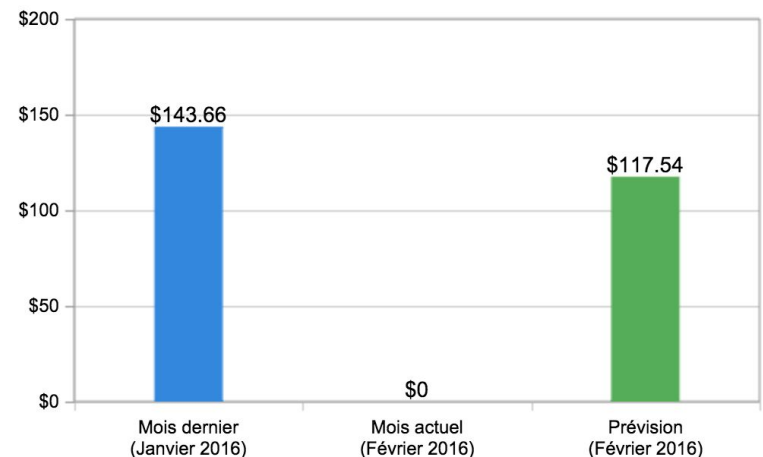
```
CREATE TABLE wikistats.top100risinghist ( refhour timestamp, title text, event_time timestamp, pageviews int, PRIMARY KEY ((refhour, title), event_time)) WITH CLUSTERING ORDER BY (event_time ASC);
```

Infrastructure



■ 4 installations sur AWS mises en place

- 6 nœuds m3.large (la plus récente, 38\$/jour)
- 6 nœuds m3.medium (19\$/jour)
- 2 nœuds m3.large (13\$/jour)
- 6 nœuds m3.xlarge (76\$/jour)



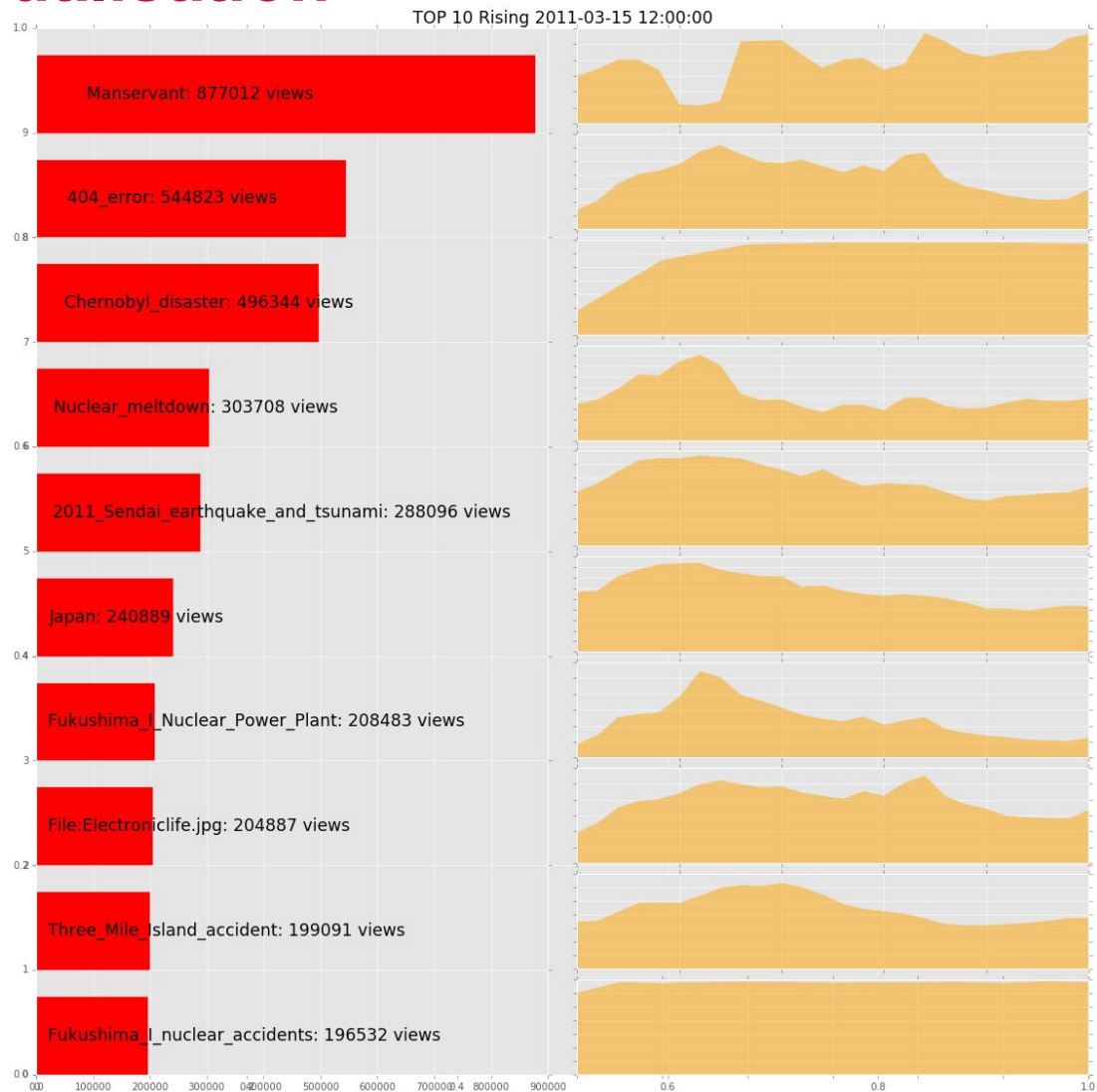
Demo data visualisation



D3.js



iPython Notebook



Lessons and Learns



Beaucoup de difficultés et de pertes de temps

- **no @ in datastax entreprise pwd**
- **gestion des dépendances du Cassandra Connector entre Spark, Scala, Cassandra**
- **peu de documentation PySpark SQL DataFrame / peu de retours d'expérience utilisateur**
- **peu de tâches parallélisables au sein de l'équipe**
- **difficultés diverses d'installation (zeppelin, etc ...)**

Apprentissage

- **une bonne structure de table accélère l'ingestion de donnée**
- **le temps d'initialisation d'une requête entre Spark et Cassandra est long - privilégier la lecture / écriture en block**
- **df.cache() - un job passe de 60s à 2s (x 2160 ...)**