2018/2019

RETOUR SUR HIVE

RAPPELS

- Hive permet de requêter en SQL sur des données stockées dans HDFS.
- Hive génère des jobs d'un moteur de calcul compatible (Hadoop MapReduce, Tez, …)
- Il permet de lire et écrire des fichiers sur HDFS.
- Beaucoup de fonctions de transformation et d'agrégation de données sont déjà disponibles.

HIVE UDF

- User Defined Functions
- Permet d'ajouter des fonctions spécifiques à votre cas d'usage
- Les UDFs doivent être écrites en Java. Pour d'autres langage, il existe un mot clé TRANSFORM permettant d'appliquer des transformations depuis un script (ex: python)
- Une UDF opère sur une seule ligne, et produit une ligne en sortie. La plupart des fonctions sont de ce type.
- Une UDAF opère sur de multiples lignes et n'en produit qu'une. C'est le type des fonctions d'agrégation, comme COUNT, SUM, etc…
- Une UTDF opère sur une seule lignes et en produit plusieurs: explode()

- On la déclare avant notre requete:
  - CREATE FUNCTION strip AS 'com.hadoopbok.hive.Strip' USING JAR '/path/to/hive-examples.jar'
  - hive> SELECT strip(' bee ') FROM dummy;
  - bee

HIVE VIEWS

- Une vue est une sorte de « table virtuelle »
- Permet à une requête d'être sauvegardée et traitée comme une table
- Exemple:
  - CREATE VIEW shorter_joint AS SELECT * FROM people INNER JOIN cart ON (cart.people_id=people.id) WHERE firstname='john';
  - Requête: SELECT lastname FROM shorter_join WHERE id=3;

HIVE PARTITIONS

- Hive peut organiser les tables en partitions, une façon de diviser la table sur HDFS selon la valeur d'une colonne de partition
- Celle-ci peut être définie par l'utilisateur, ou selon la valeur d'une colonne existante.
- Permet d'optimiser les requêtes: un filtre sur une variable de partition permettra de limiter le nombre de fichiers que Hive doit parser.
- Les partitions peuvent être multiples

- Exemple:
  - CREATE TABLE tracking (…) PARTITIONED BY (jour STRING, site STRING)

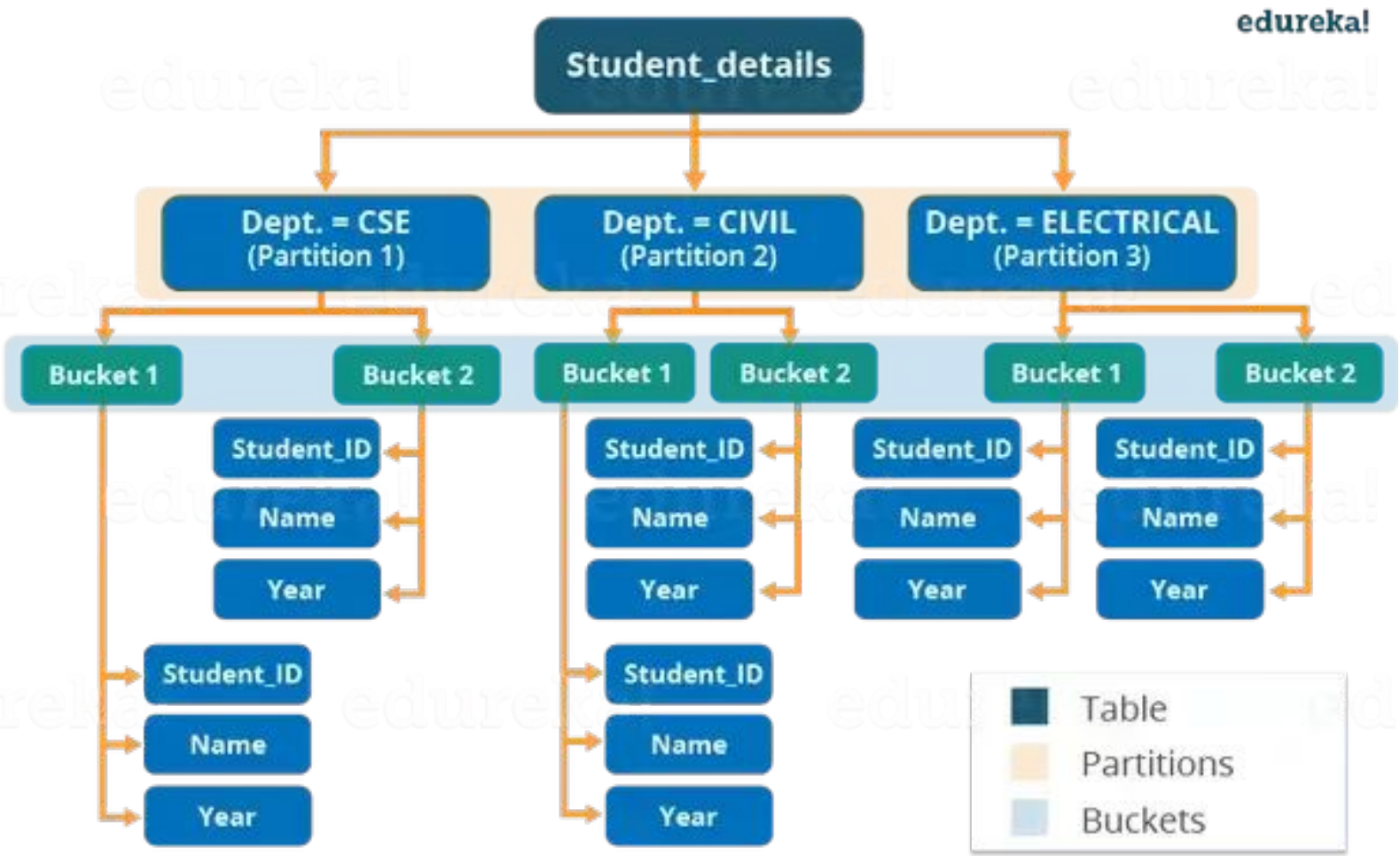
Sur HDFS:
/user/hive/warehouse/tracking/jour=20180930/site=android
/user/hive/warehouse/tracking/jour=20180930/site=ios
/user/hive/warehouse/tracking/jour=20180929/site=android
/user/hive/warehouse/tracking/jour=20180929/site=ios

HIVE BUCKETS

- Les partitions permettent de séparer les données sur HDFS et optimiser les requêtes.
- Comment « partitionner » sur des cardinalités plus grandes?
- Les BUCKETS permettent de résoudre ce problème. Les données sont réparties sur des fichiers différents HDFS via un hash.

- Exemple: CREATE TABLE clients (id_client INT, …) CLUSTERED BY (id_client) INTO 10 BUCKETS;

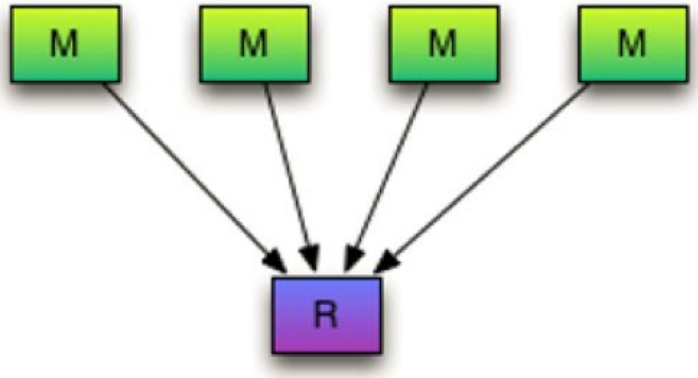
Schema Partitions et Buckets

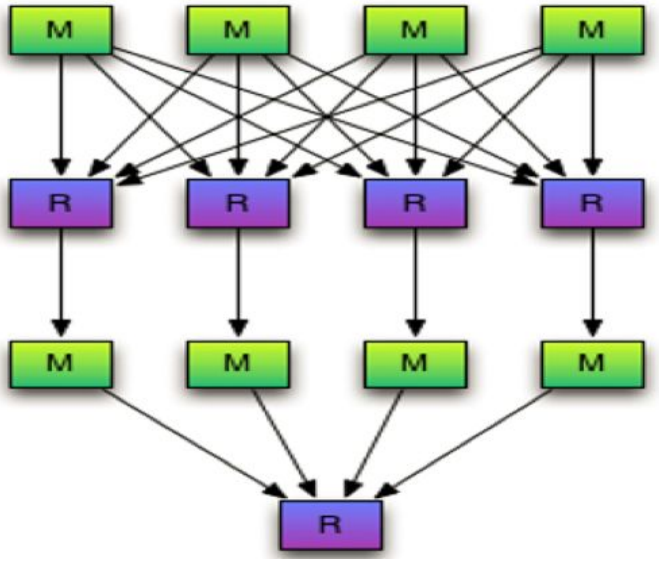


Optimisations

Laquelle de ces requêtes est la plus rapide ?

- SELECT COUNT(DISTINCT(word)) FROM table
- SELECT COUNT(1) FROM (SELECT DISTINCT(word) FROM table)





Optimisations

- SELECT COUNT(DISTINCT(word)) FROM table :
  - Les Mappers envoient leur résultats à un Reducer
  - Un seul Reducer compte tous les résultats

- SELECT COUNT(1) FROM (SELECT DISTINCT(word) FROM table) :
  - Les Mappers envoient leur résultats à plusieurs Reducer
  - Chaque Reducer génère une liste
  - Les Mappers suivants comptent la taille de leur liste
  - Le dernier Reducer additionne les tailles de chaque liste

Optimisations

Pushdown Predicate:

La condition est exécutée “au plus près de la donnée”.

Optimisation lors de la génération du logical plan:

```
SELECT SUM(s.unit_sales)
FROM product p
JOIN sales s
ON s.id = p.id
WHERE
p.brand_name = “Washington”
```

```
SELECT SUM(s.unit_sales)
FROM sales s
JOIN (
    SELECT id, brand_name
    FROM product
    WHERE brand_name = “Washington”
    ) p
ON s.id = p.id
```

Les lignes sont filtrées avant d’être envoyées au Reducer.

Optimisations

Pushdown Predicate Transitivity:

Propagation du prédicat dans le cas d'une jointure

```
CREATE TABLE invites (foo int, bar string) PARTITIONED BY (ds string);
CREATE TABLE invites2 (foo int, bar string) PARTITIONED BY (ds string);
SELECT COUNT(*)
FROM invites
JOIN invites2 ON invites.ds = invites2.ds
WHERE invites.ds = '2011-01-01';
```

Error in semantic analysis: No Partition Predicate Found for Alias "invites2" Table "invites2"

Des formats de fichiers comme Parquet ou ORC permettent de filtrer des blocs de données à lire, et permettent donc un gros gain de performance.

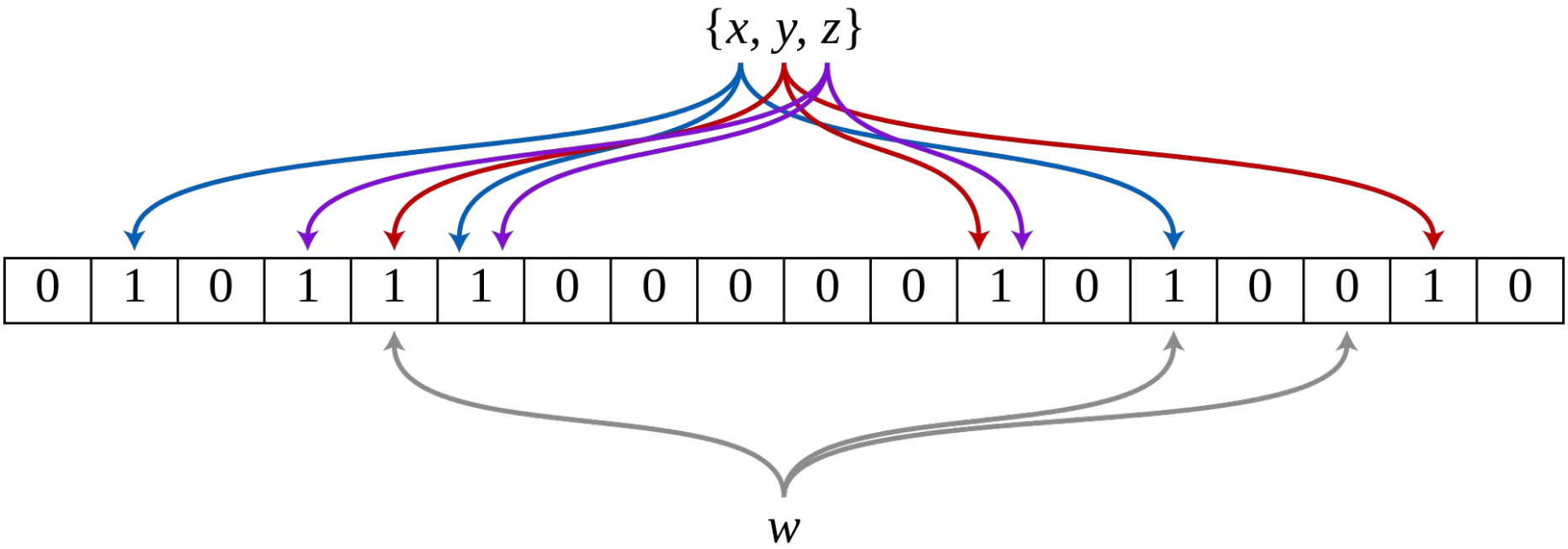
Optimisations

Bloom Filters:

Structure de données qui permet de savoir:

- avec certitude que l'élément est absent de l'ensemble (il ne peut pas y avoir de faux négatif)
- avec une certaine probabilité que l'élément peut être présent dans l'ensemble (il peut y avoir des faux positifs)

Très utile pour éviter de lire des blocs de données => ORC ou Parquet



Sort By, Order By, Distribute By, Cluster By

- **Order by:**
  Un seul reducer car l'ensemble des outputs des Mappers doivent être ordonnés.
  En "strict mode", le LIMIT est imposé. => utilisé pour restreindre les requêtes trop coûteuses.

- **Sort By:**
  Les outputs des Mappers sont triés avant d'être traités par les reducers.
  Plusieurs outputs sont possibles car les résultats sont triés et ordonnés par Reducer (par fichier de sortie)

- **Distribute By:**
  Toutes les lignes de la même colonne spécifiées par le Distribute By vont vers le même Reducer.

- **Cluster By:**
  Distribute By + Sort By sur la même colonne = Cluster By

HIVE FILE FORMATS

- Exemples de formats de sérialisation des données:
  - AVRO File
  - ORC File
  - Parquet File
  - Text File

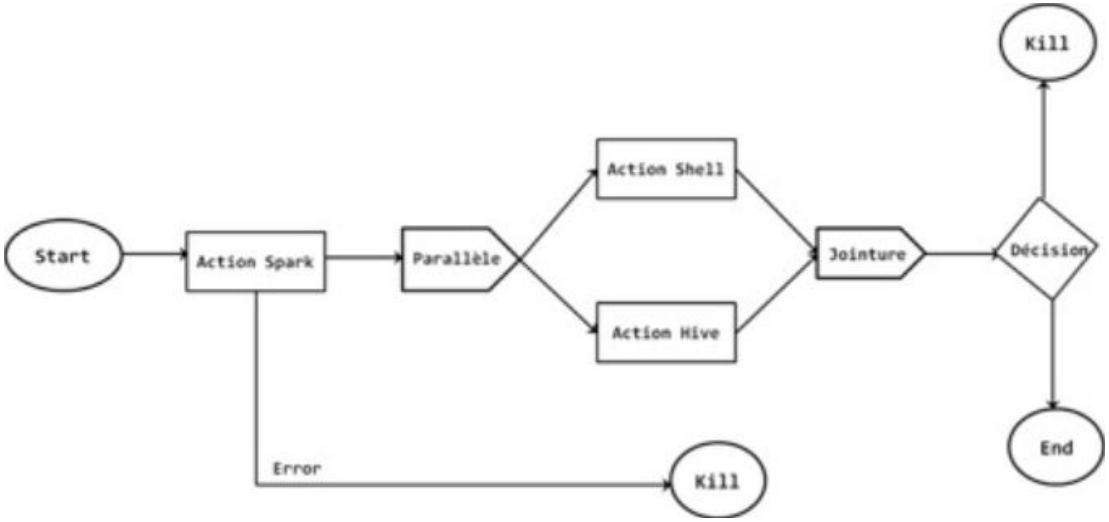
- Permet de gérer des jobs Hadoop récurrents ou ponctuels
- Intègre de nombreuses applications de l'écosystème Hadoop
  - Ex: Spark, Hive, Pig, …
- Permet de coordonner des jobs. (Lancements réguliers, selon une condition)
- Un workflow est un ensemble d'actions et de conditions organisées sous la forme d'un graphe orienté acyclique (DAG)
- Les actions (action nodes) peuvent être des jobs MapReduce ou Spark, des requêtes Hive, des scripts Pig, Java ou Shell, etc…)
- Les conditions (decision nodes) peuvent porter sur le bon déroulement des actions précédents, ou sur des métriques exportées.
- Les Workflow sont définis en XML.

OOZIE

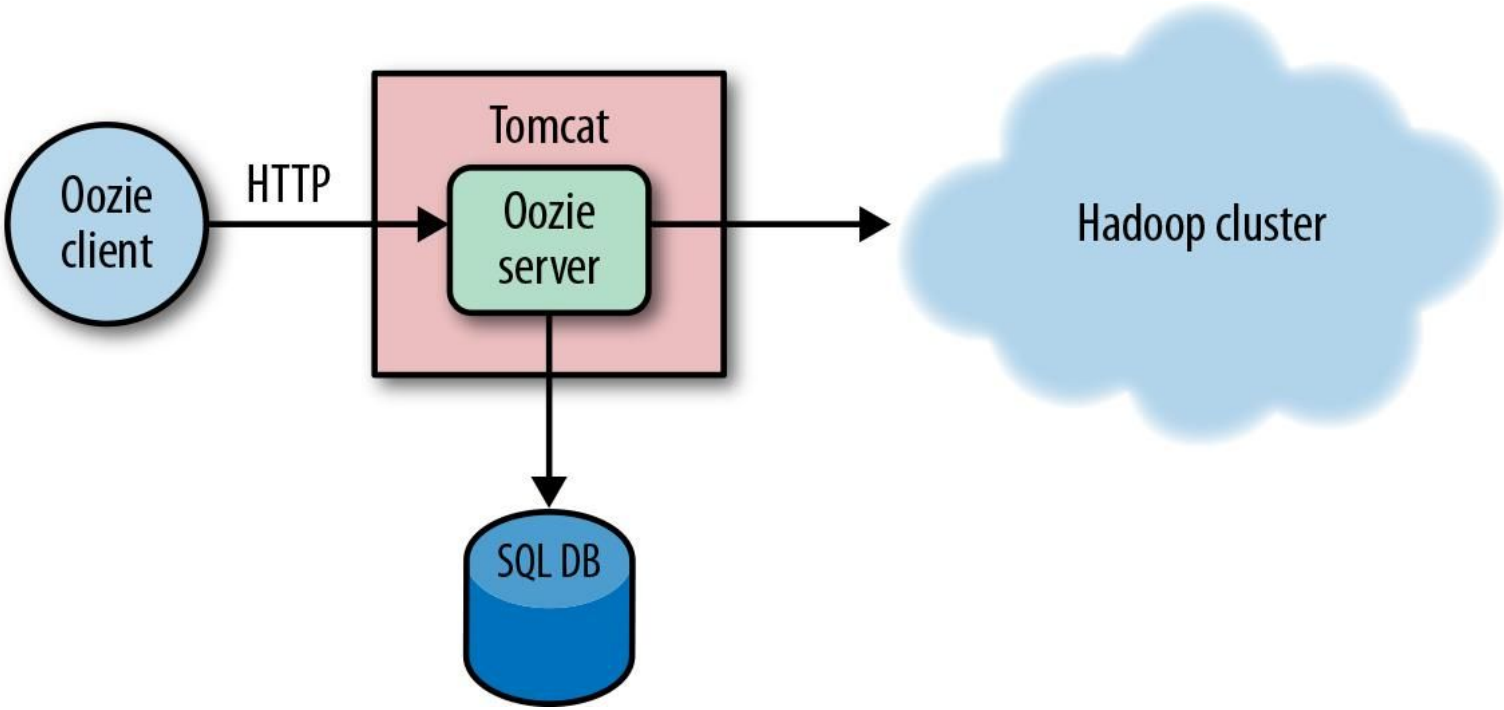
- Open Source - java web app
- Workflow scheduler dans Hadoop
- Un workflow est un ensemble d'actions (MR, interactions HDFS, Pig, script Java ou shell, etc.)
- Oozie permet de créer les jobs, de les piloter et les automatiser
- 2 types de jobs dans Oozie :
  - Oozie workflow : les jobs sont des Directed Acyclical Graphs (DAGs), une séquence d'actions à exécuter.
  - Oozie coordinator : Oozie workflow de façon récurrente (exemple : lancés tous les matins au moment de l'arrivée de nouvelle donnée)

OOZIE: TYPES DES NOEUDS

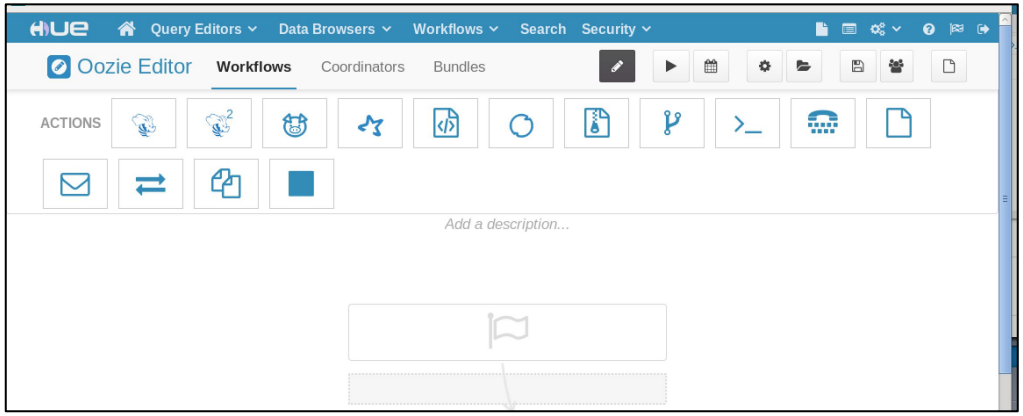
- START, END, KILL: Nœuds obligatoires
- Action Nodes: Jobs lancés sur le cluster
  - 2 possibilités en sortie: ok ou error.
  - Possibilité de faire un « capture-output » pour par exemple donner des informations aux decision nodes.
- Decision Nodes: Permettent de diriger la suite du WF selon une condition.
- Fork, Join: Sépare et joint le workflow pour exécuter des tâches non dépendantes en parallèle.



OOZIE - ARCHITECTURE



OOZIE - USAGE



Depuis Hue :

- Drag&Drop pour créer son job
- De grandes limitations, il est préférable de travailler directement en XML ou avec des outils customs.

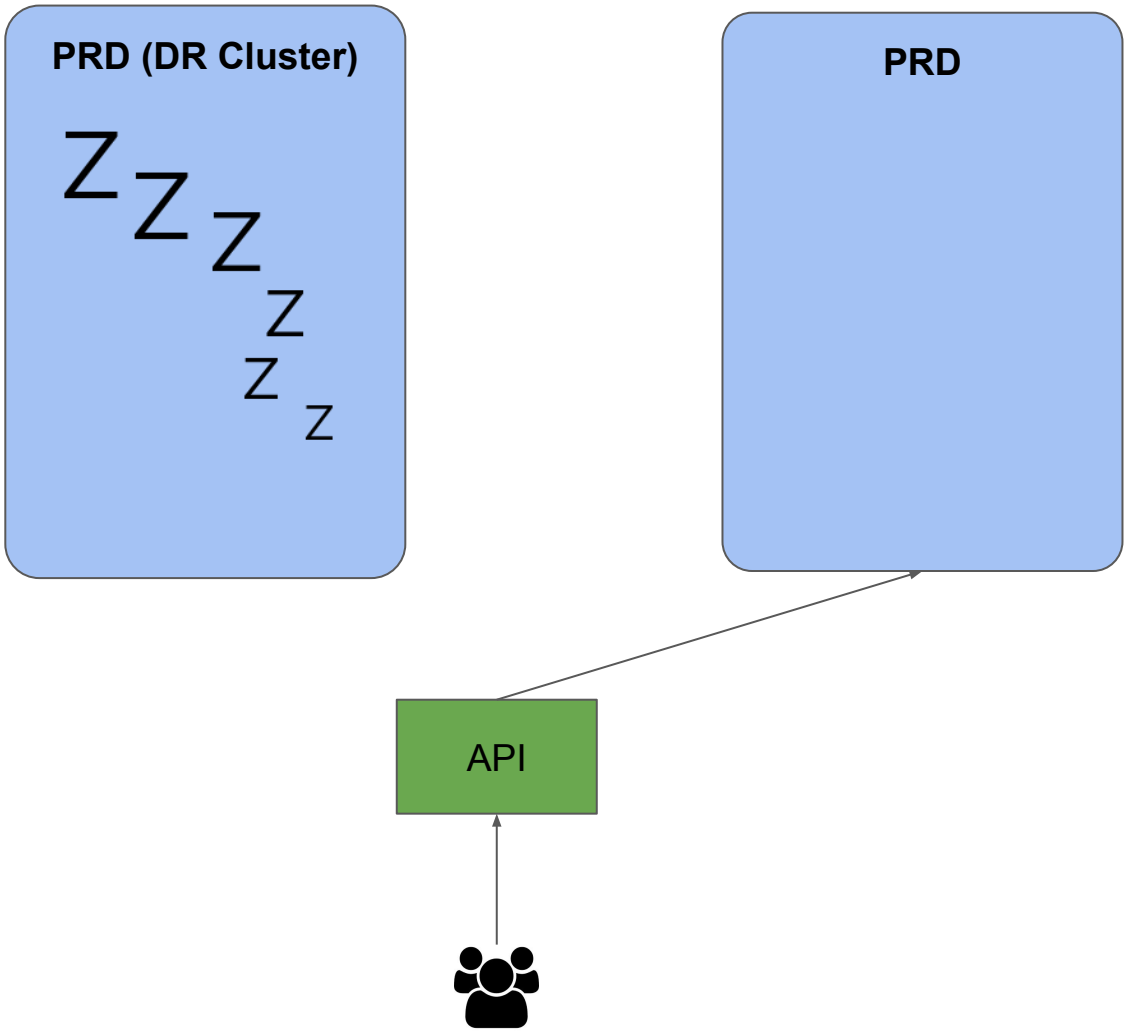
```
<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "simple-Workflow">
  <start to = "Create_External_Table" />

  <!—Step 1 -->

  <action name = "Create_External_Table">
    <hive xmlns = "uri:oozie:hive-action:0.4">
      <job-tracker>${JobTracker}</job-tracker>
      <name-node>hdfs://clustername:8020</name-node>
      <script>hdfs_path_of_script/external.hive</script>
    </hive>

    <ok to = "Create_orc_Table" />
    <error to = "kill_job" />
  </action>
```

……

HADOOP: DR et Architectures

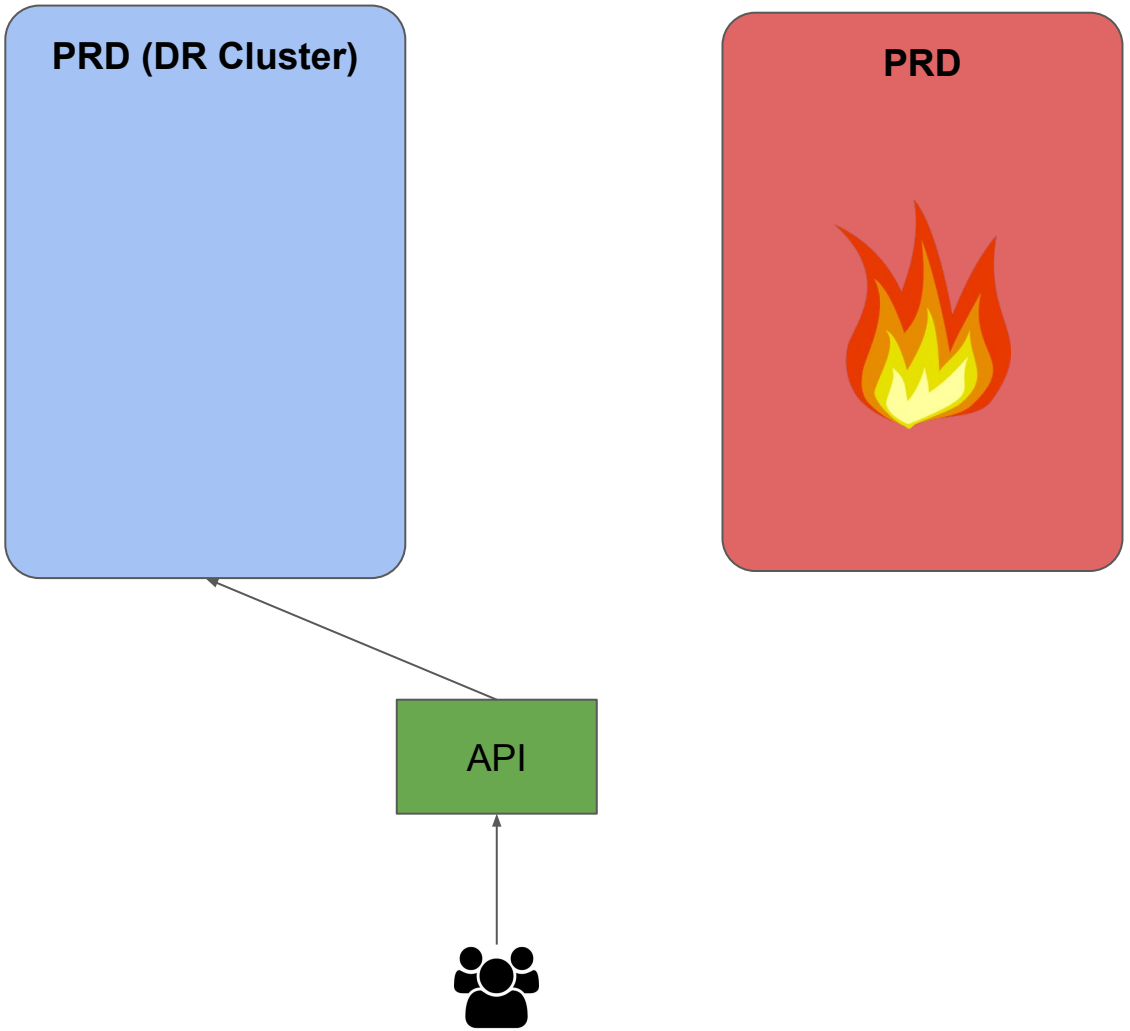
Qu’est-ce que le concept de DR (Disaster Recovery) ?

Si le cluster de production est Down, les applications doivent malgré tout continuer de fonctionner.
Il y a donc nécessairement un autre cluster disponible afin que les activités puissent continuer.

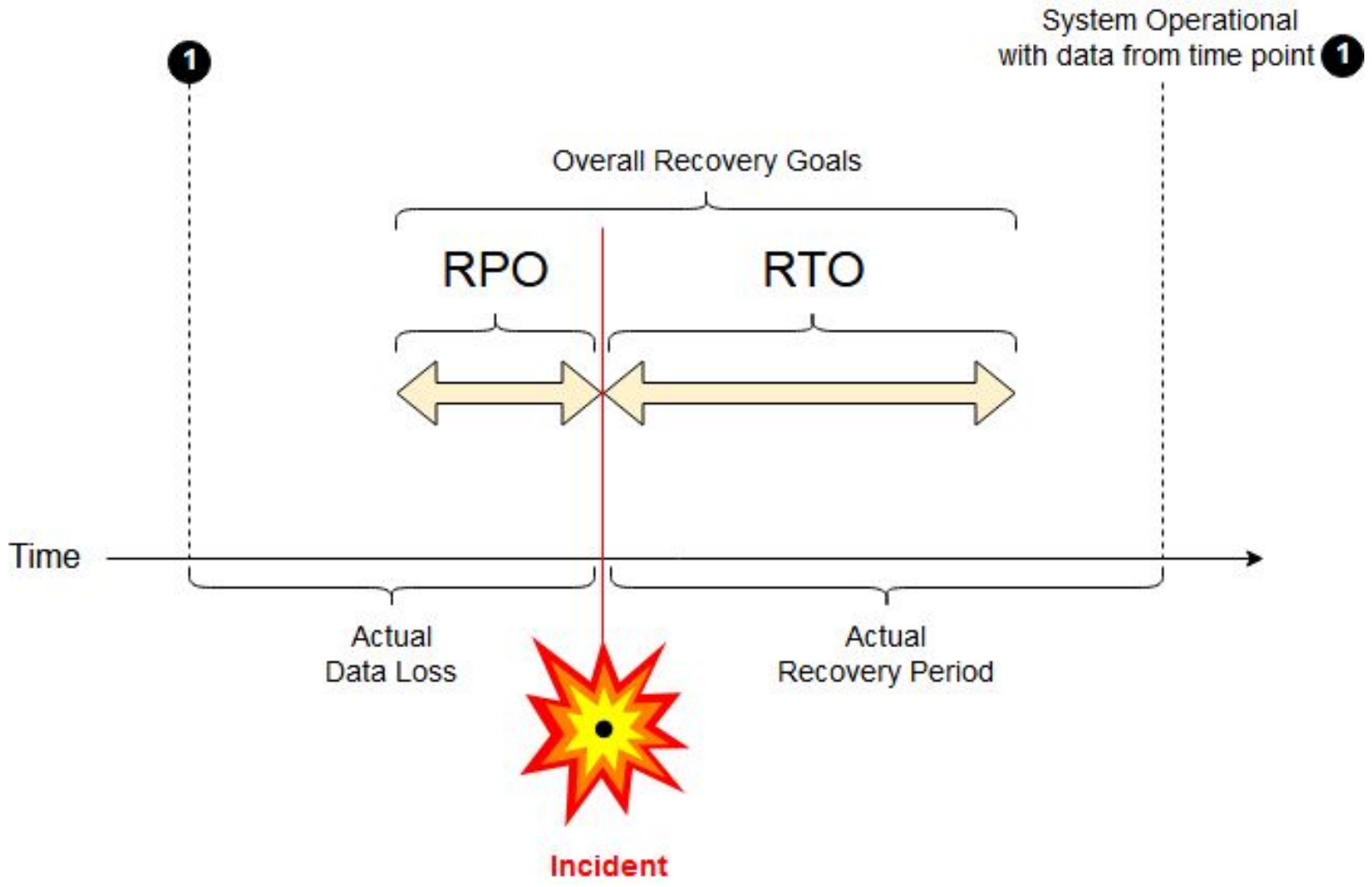


DR - Scenario

Tous les clients doivent switcher sur le second cluster. Cela doit être transparent pour les ends users (Switch automatique des APIs)



DR - Scenario

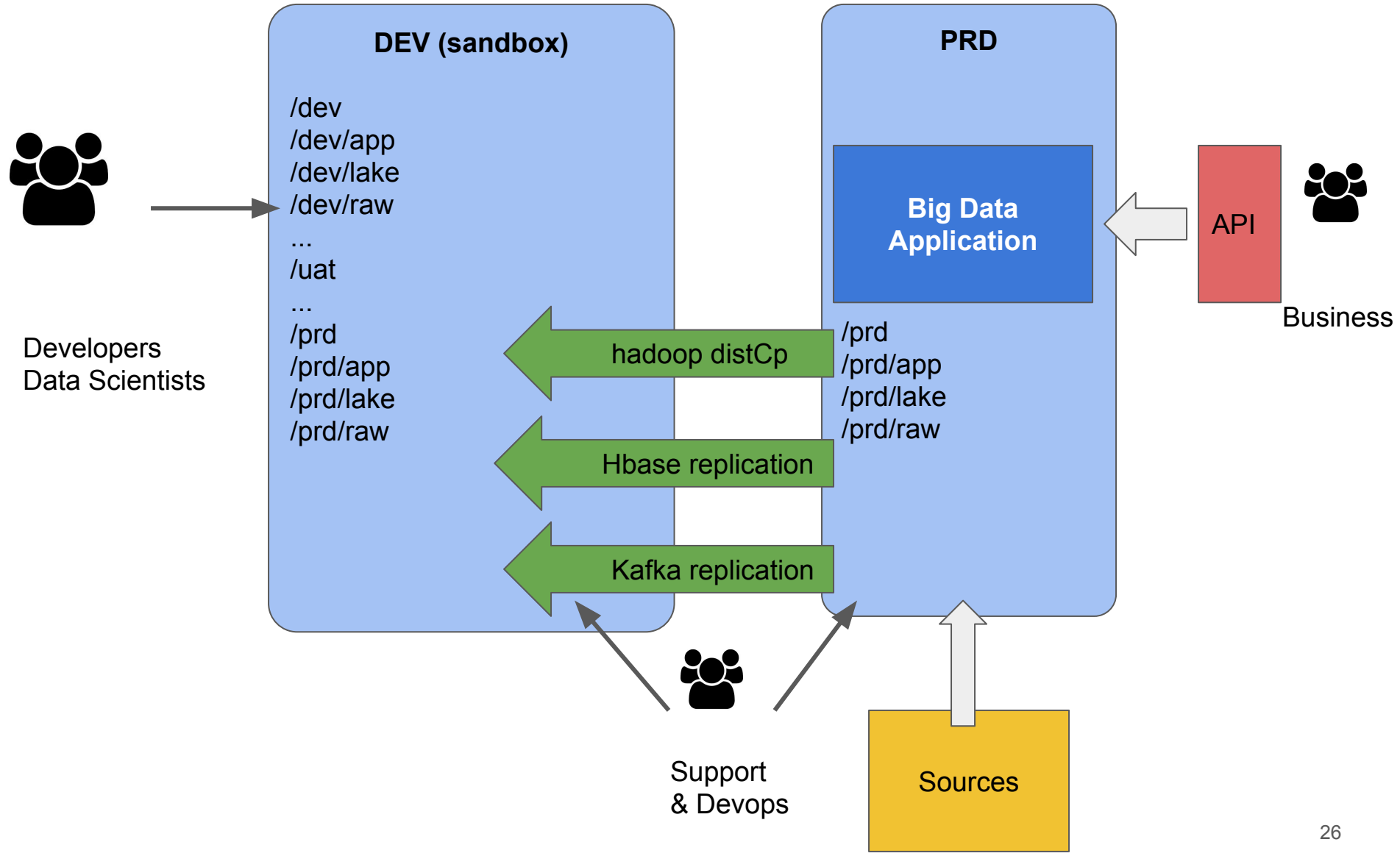


RPO: Recovery Point Objective
RTO: Recovery Time Objective

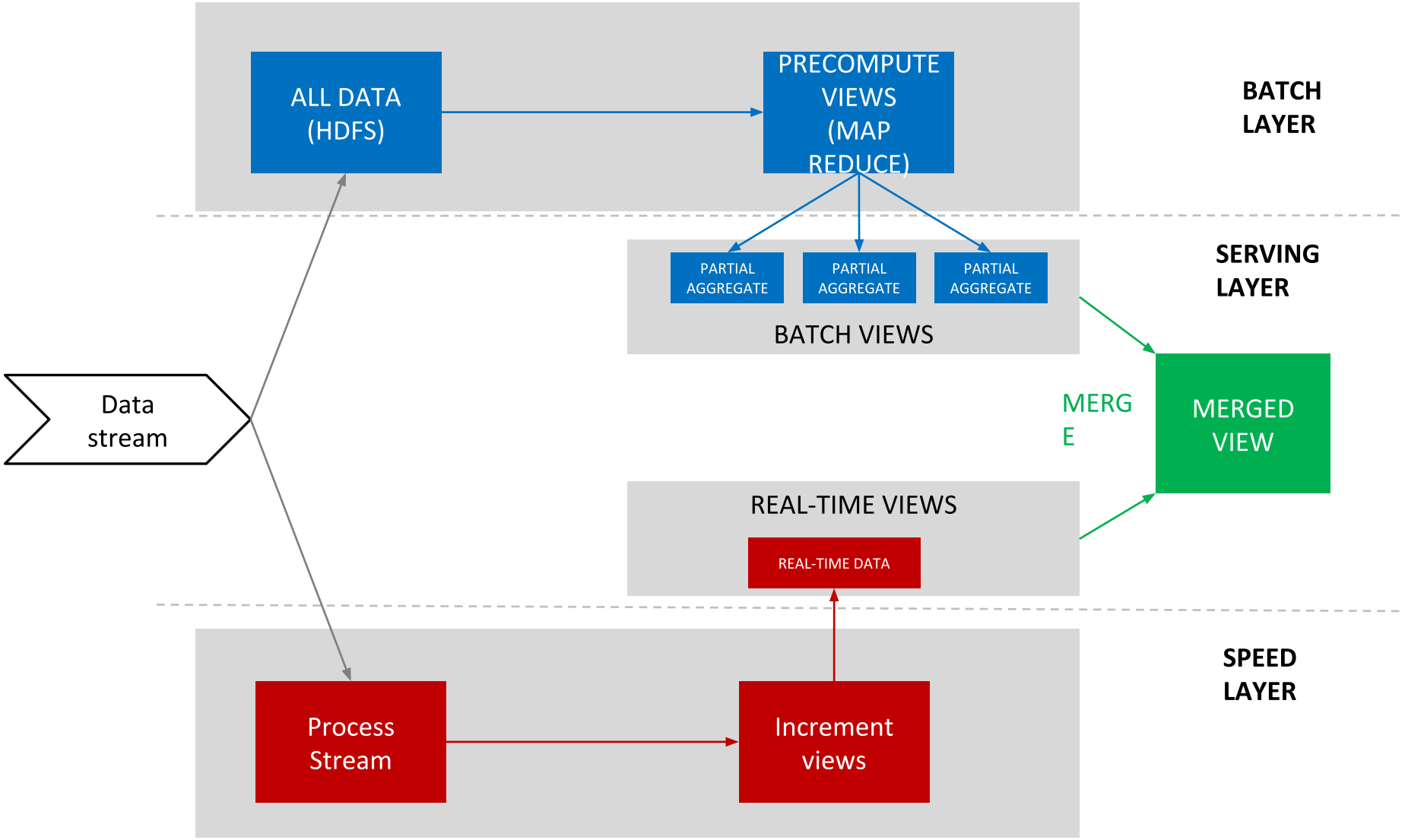
A whole cluster sleeping ?

Clusters Setup

On peut par exemple utiliser le cluster de DEV comme cluster de DR. Il doit donc être plus gros que le cluster de PRD.

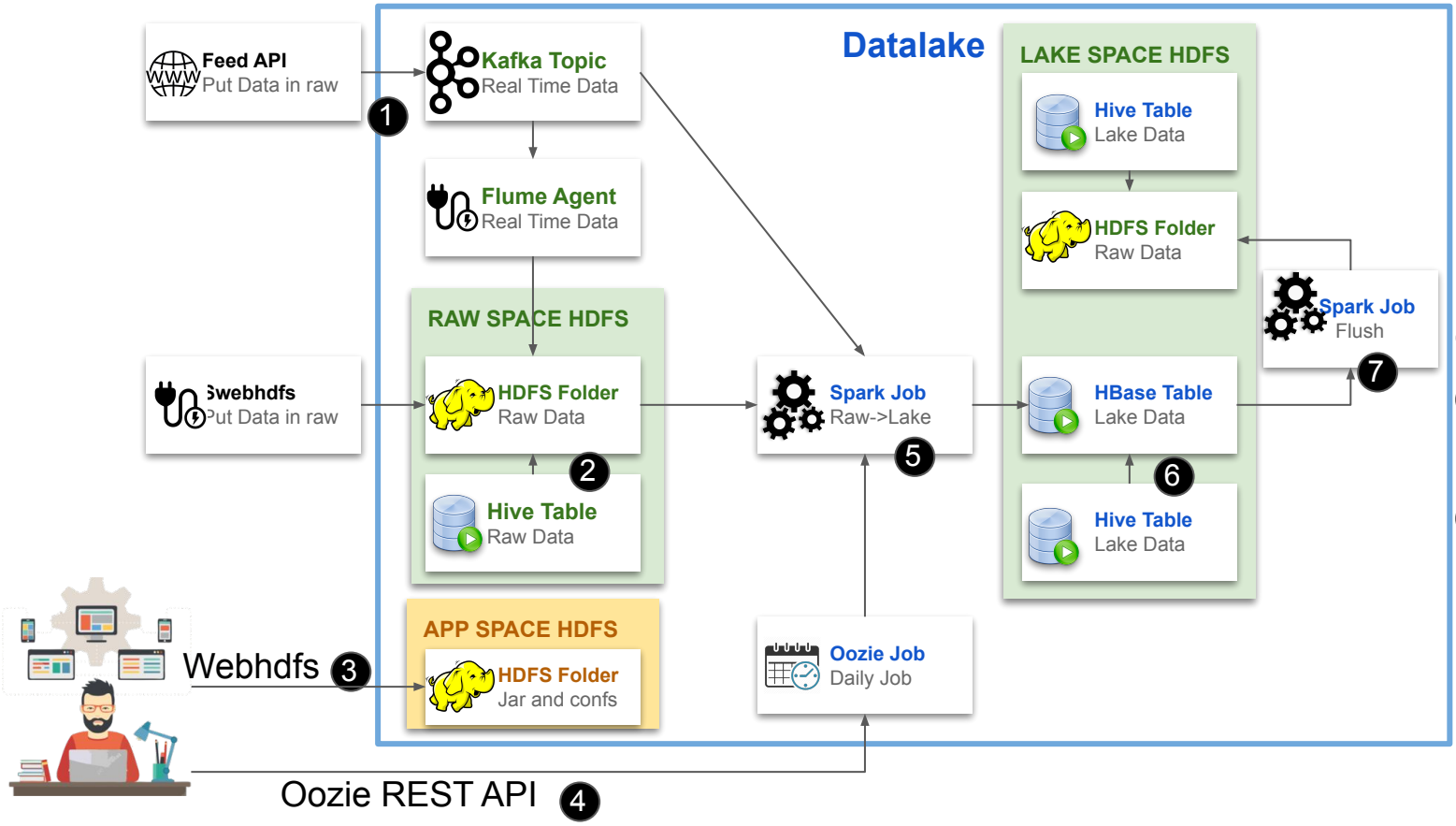


ARCHITECTURE LAMBDA



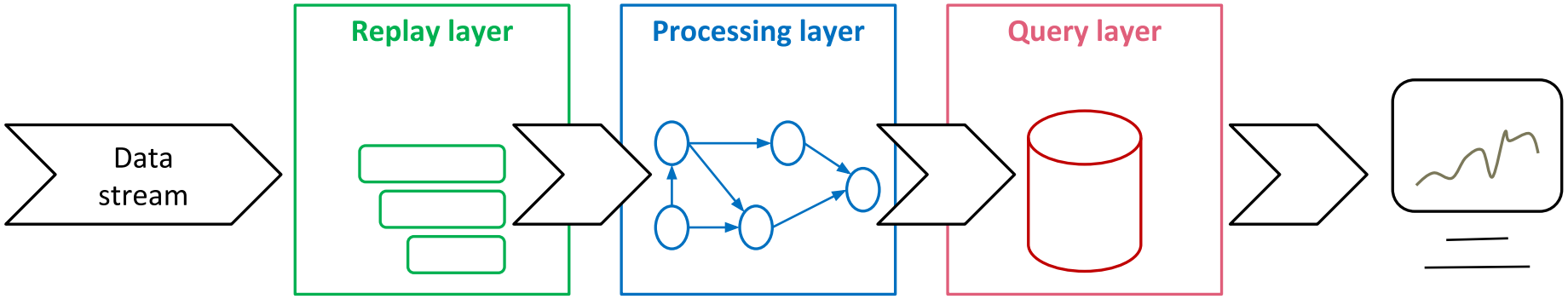
A Raw To Lake Architecture

Tous les projets qui envoient des données sur le datalake doivent créer un job Raw -> Lake. Comparables à un ELT, pour normaliser les données. Pour construire un datalake propre que tout le monde peut requêter.



1. Source pushes data to Feed API and arrives in a Kafka Topic
2. Data are stored in HDFS and an external Hive Table is on created on it.
3. Job code is uploaded in private space.
4. The user launches the job via Oozie API
5. The job converts raw to Lake Format either in Batch or real time (same code)
6. An Hive Table is mapped on the HBase Data to be able to query it easily.
7. Each day we flush data from HBase to Parquet to another Hive table with historical data.

ARCHITECTURE KAPPA



kafka

RabbitMQ

ØMQ

Spark Streaming

STORM

samza

Flink

redis

cassandra

APACHE HBASE