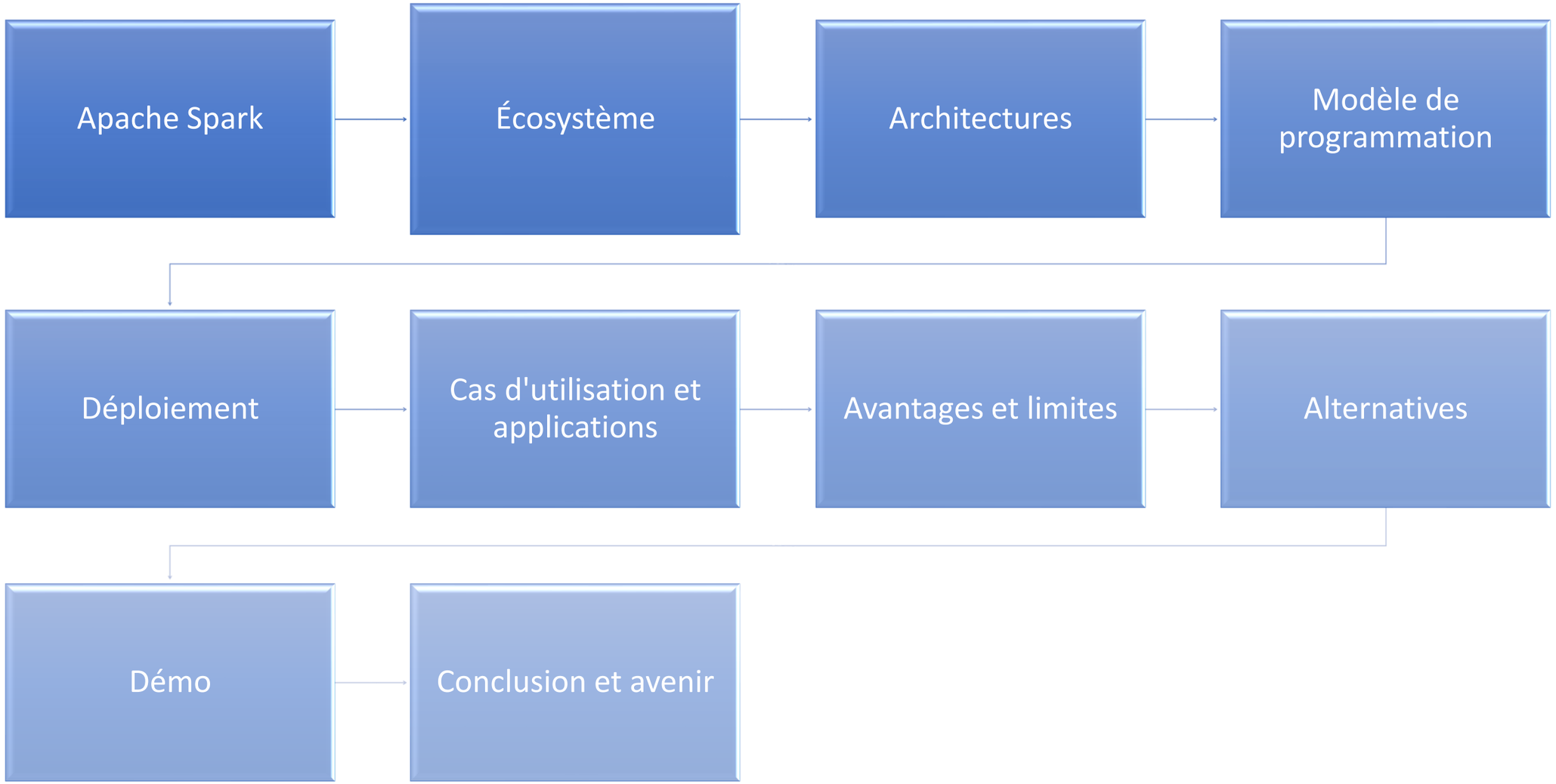
Par : Mouhamadou Adji BARRY, DIT/Atos

Encadreur : Ibrahima Oumar LY, Data Engineer

Plan


Apache Spark
Écosystème
Architectures
Modèle de programmation
Déploiement
Cas d'utilisation et applications
Avantages et limites
Alternatives
Démo
Conclusion et avenir

Histoire et Évolution d'Apache Spark

2009

1

Développement initial par AMPLab à l'Université de Californie, Berkeley.

2010

2

Open-source sous licence BSD.

2013

3

Apache Spark devient un projet Apache Top-Level.

2014

4

Spark 1.0 est lancé.

Aujourd'hui

5

Largement utilisé dans l'industrie pour le traitement de données.

Caractéristiques Clés d'Apache Spark

1. **Rapidité**

   Traitement en mémoire très rapide.

2. **Polyvalence**

   APIs simples en Java, Scala, Python, et R.

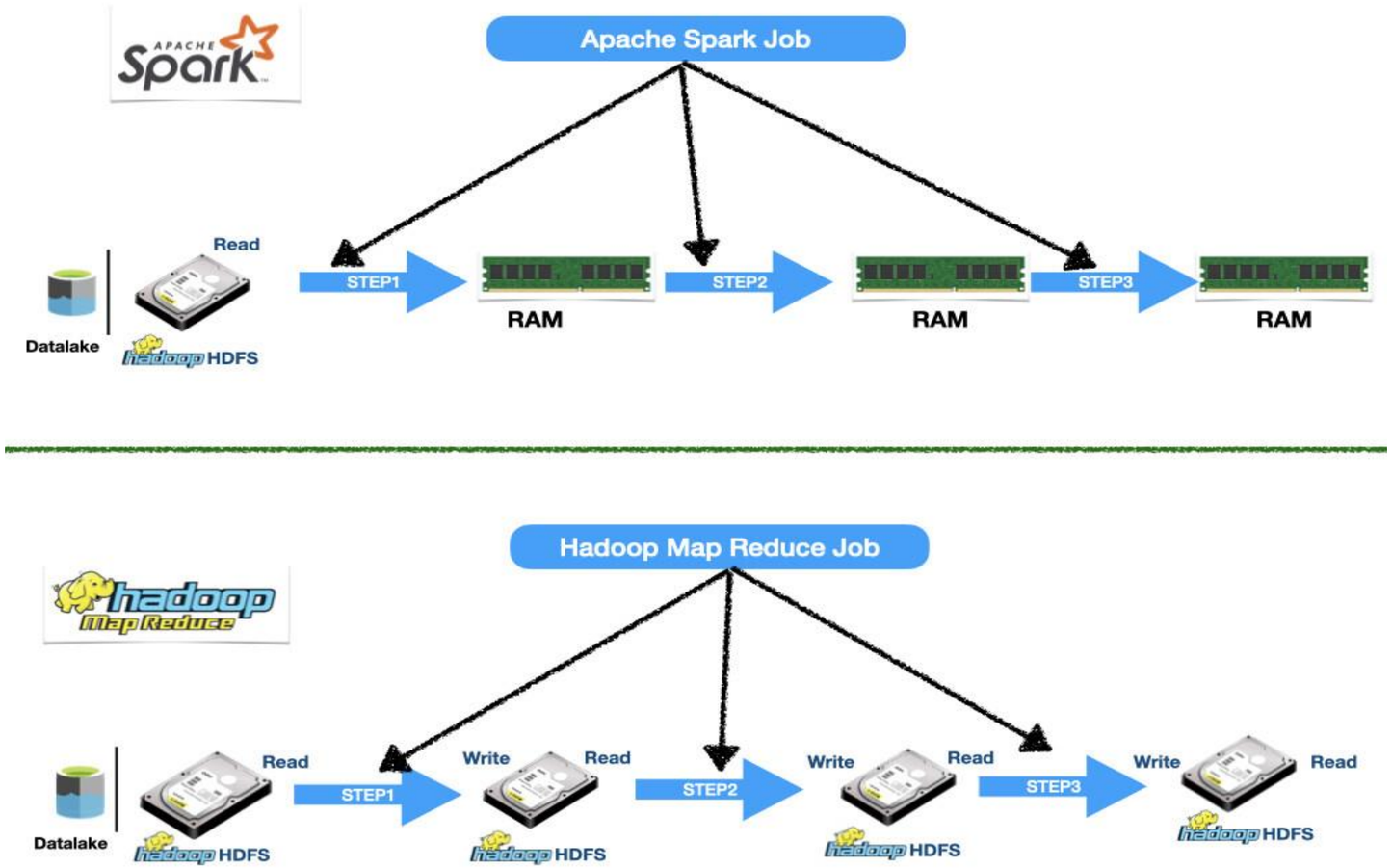
3. **Facilité d'Utilisation**

   Large gamme d'applications (batch, streaming, ML, graphes).

4. **Écosystème Complet**

   Intégration facile avec Hadoop, Cassandra, HBase, etc.

Rapidité de Spark par rapport à Hadoop MapReduce



L'Écosystème de Spark

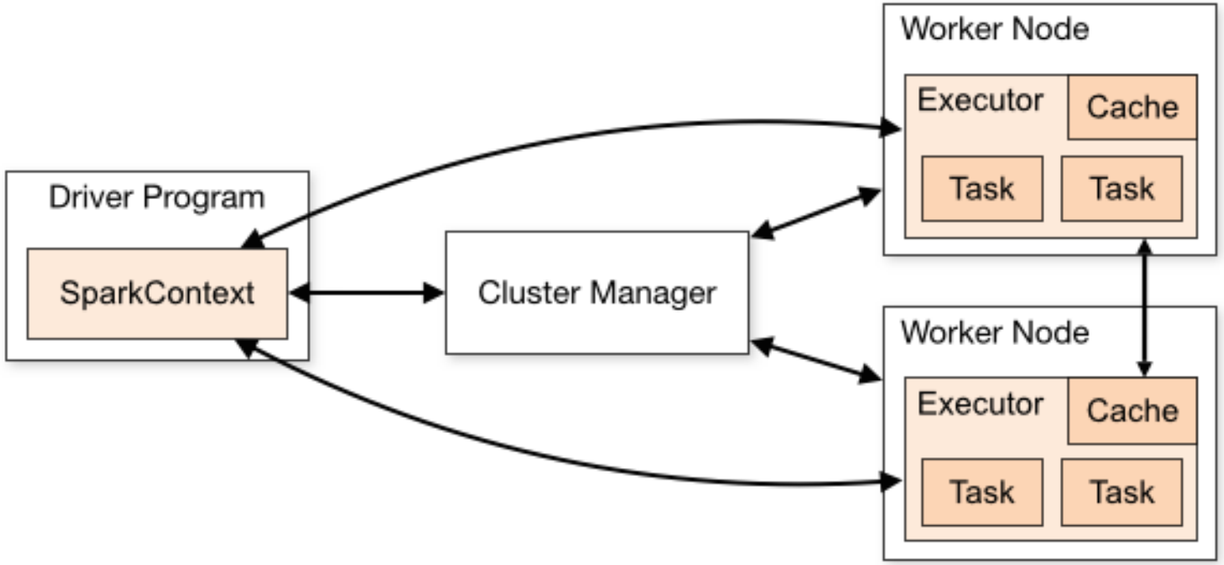
Spark SQL	Spark Streaming	GraphX	Mllib
Traitement de données structurées	Traitement de flux de données	Traitement de graphes	Bibliothèque de machine learning

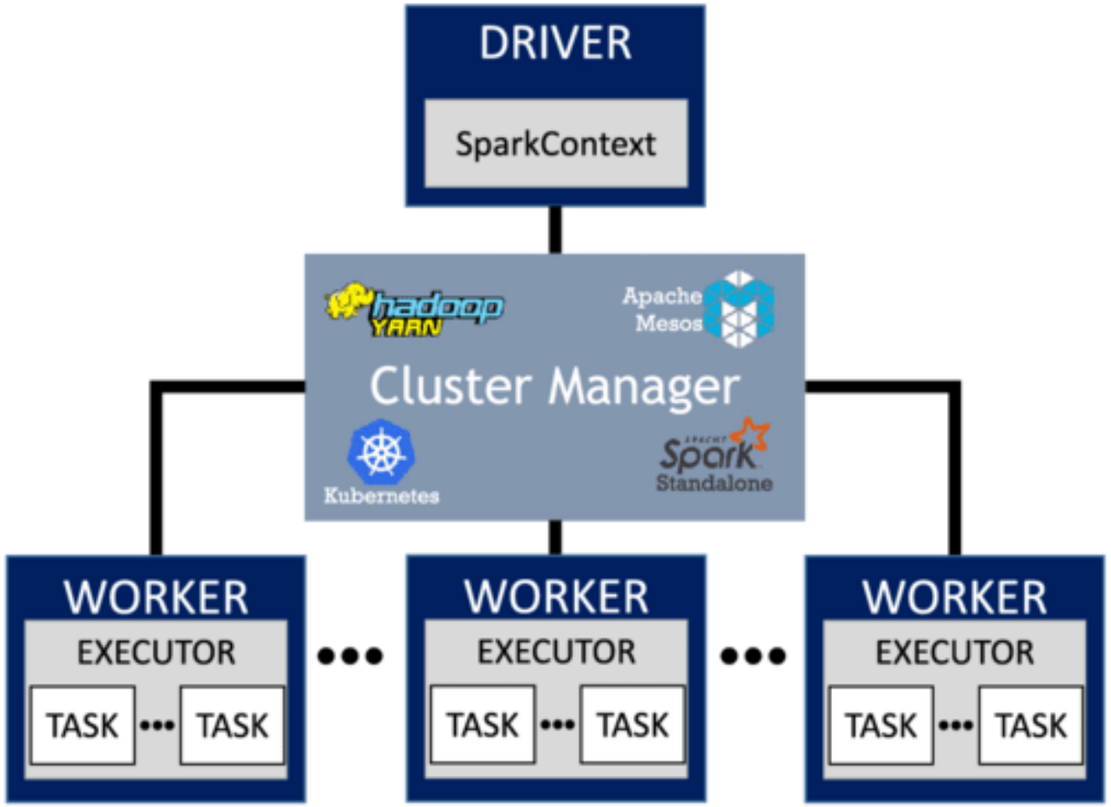
Spark Core

Moteur d'exécution

Architectures de Spark

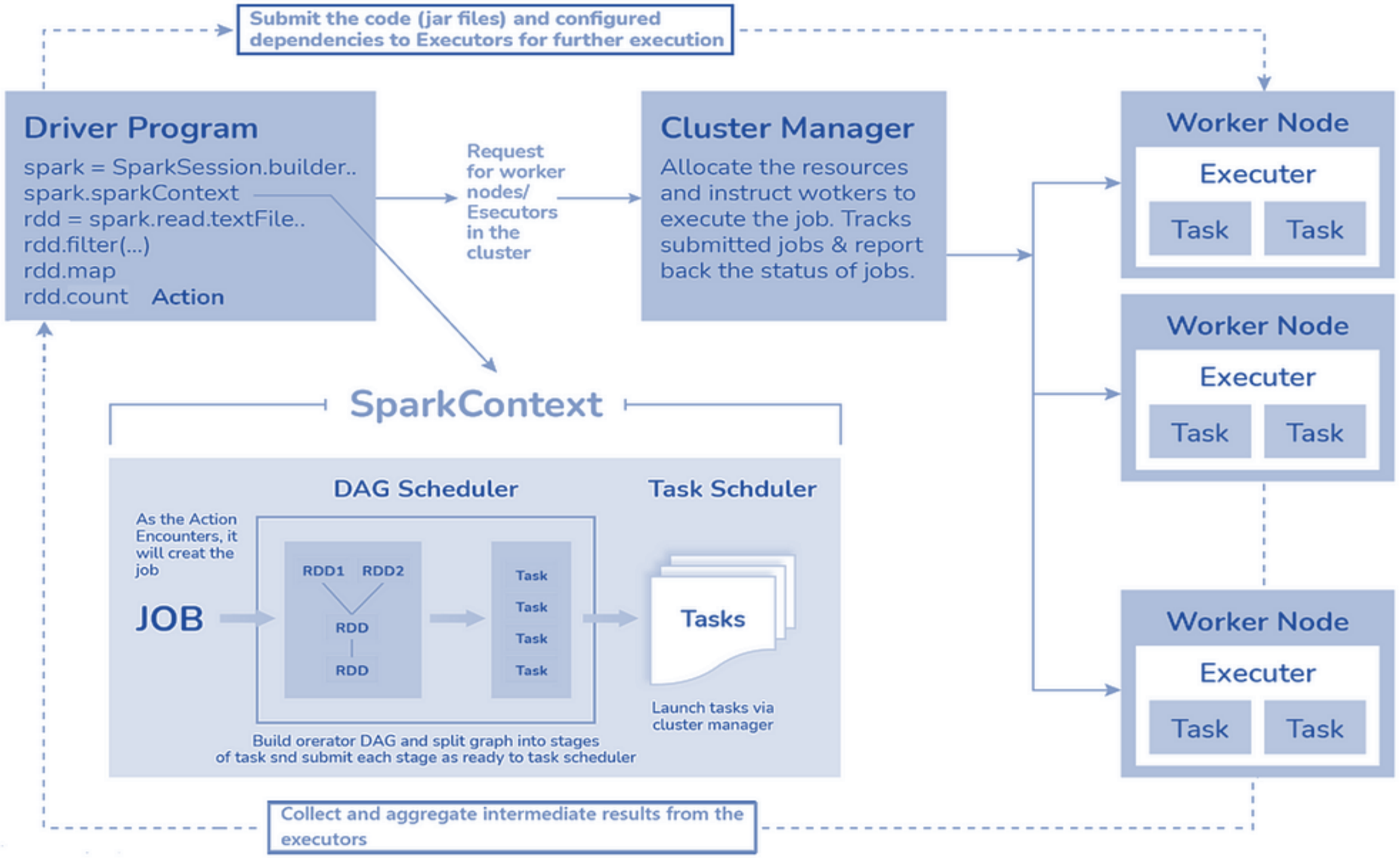
Composants





Architectures de Spark

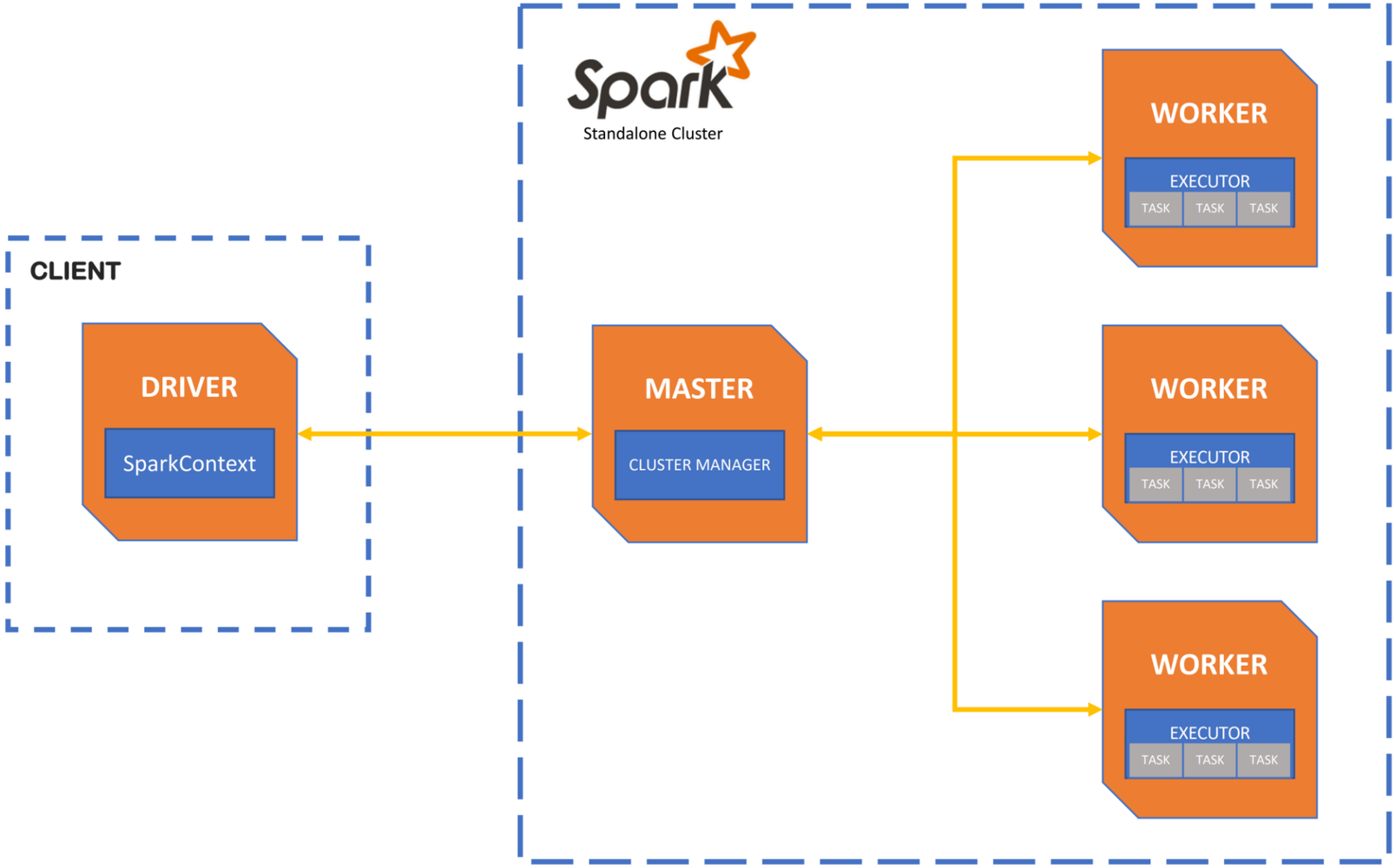
Flux d'Exécution



Architectures de Spark

Modes de Cluster

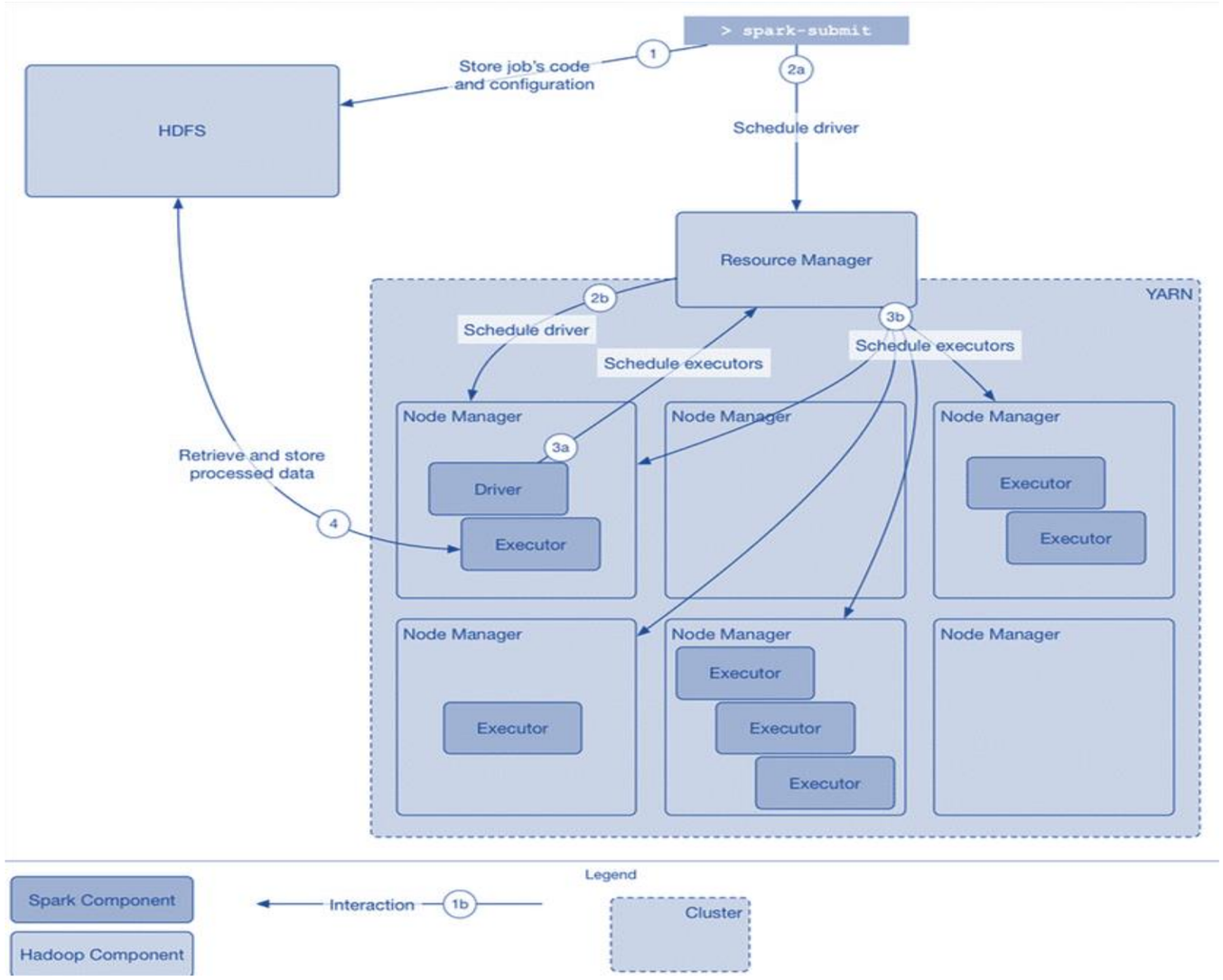
Local



Architectures de Spark

Modes de Cluster

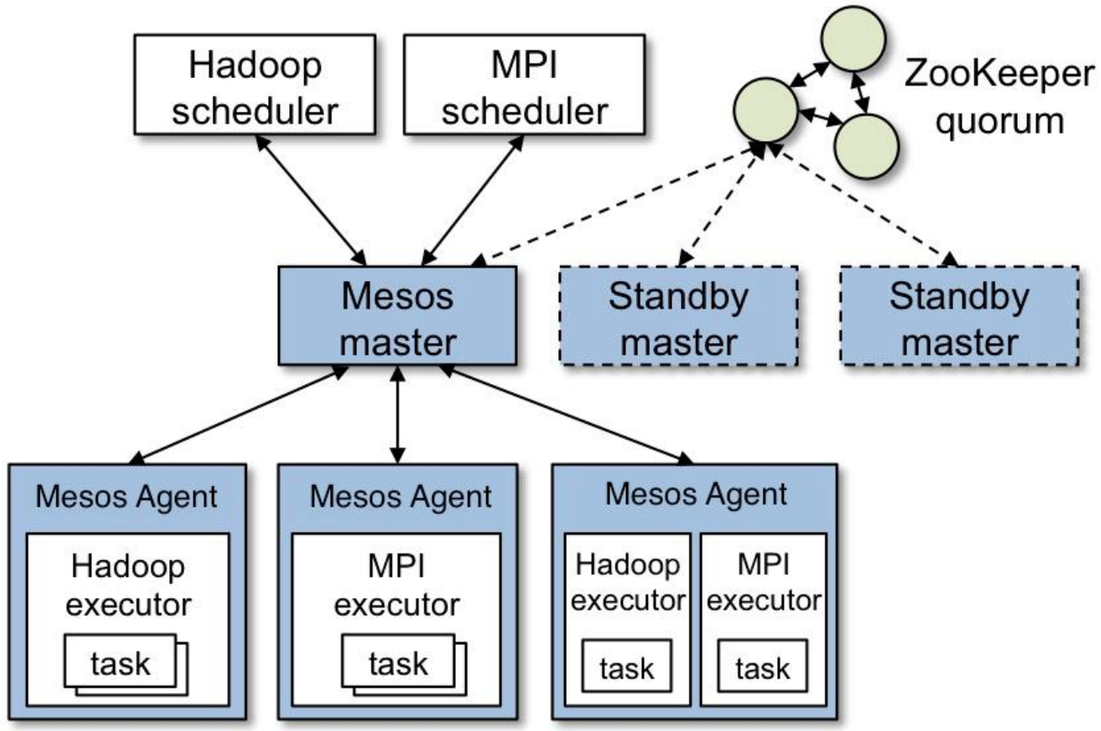
YARN

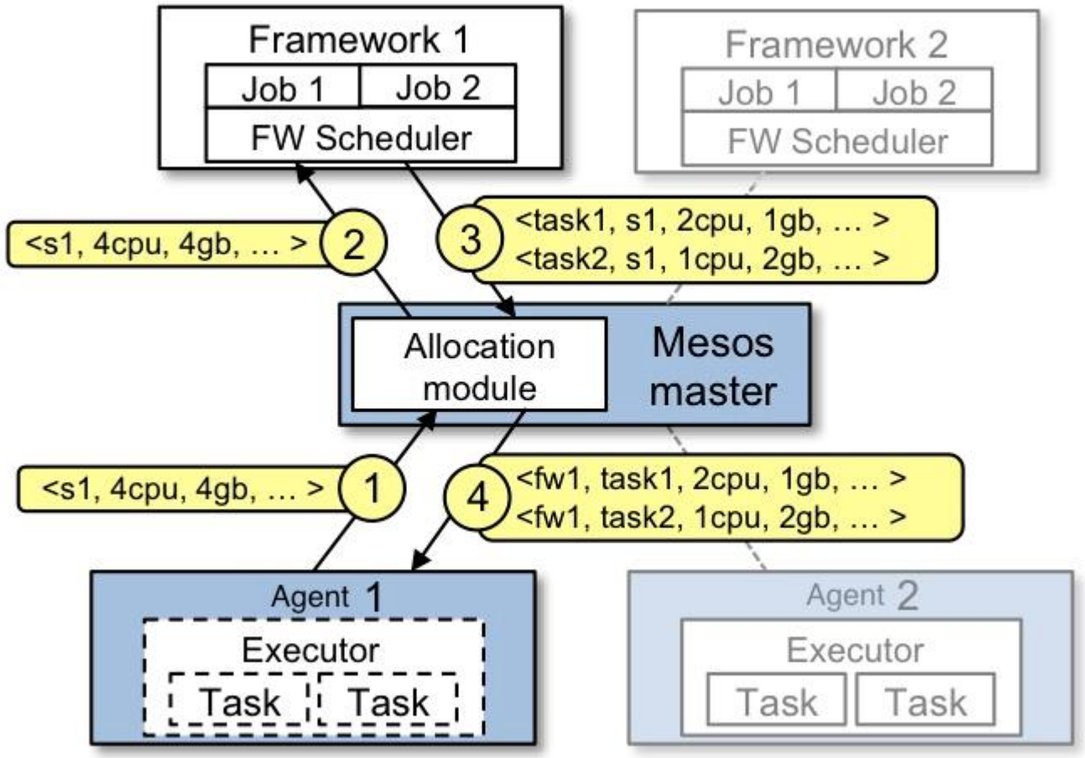


Architectures de Spark

Modes de Cluster

Mesos

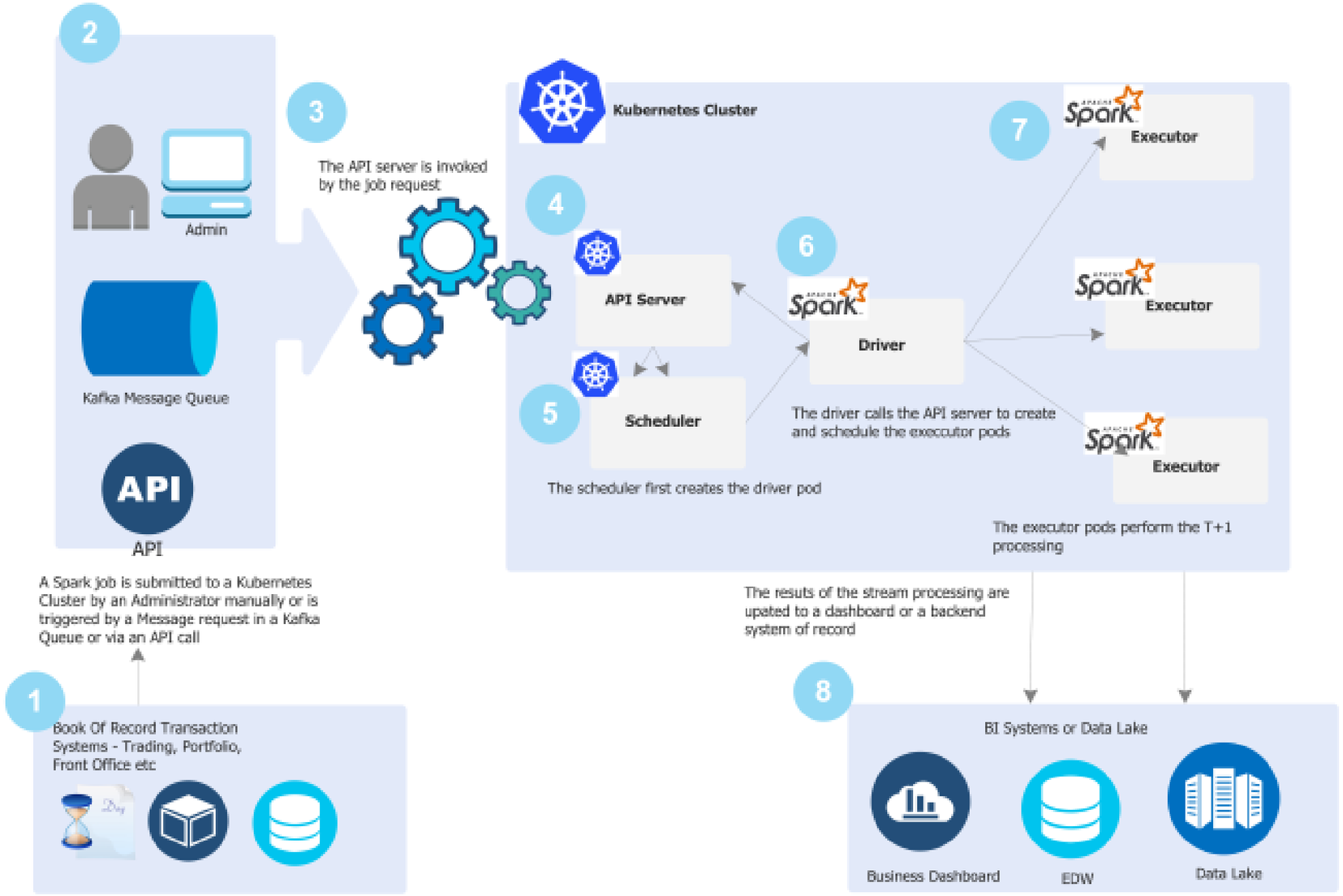




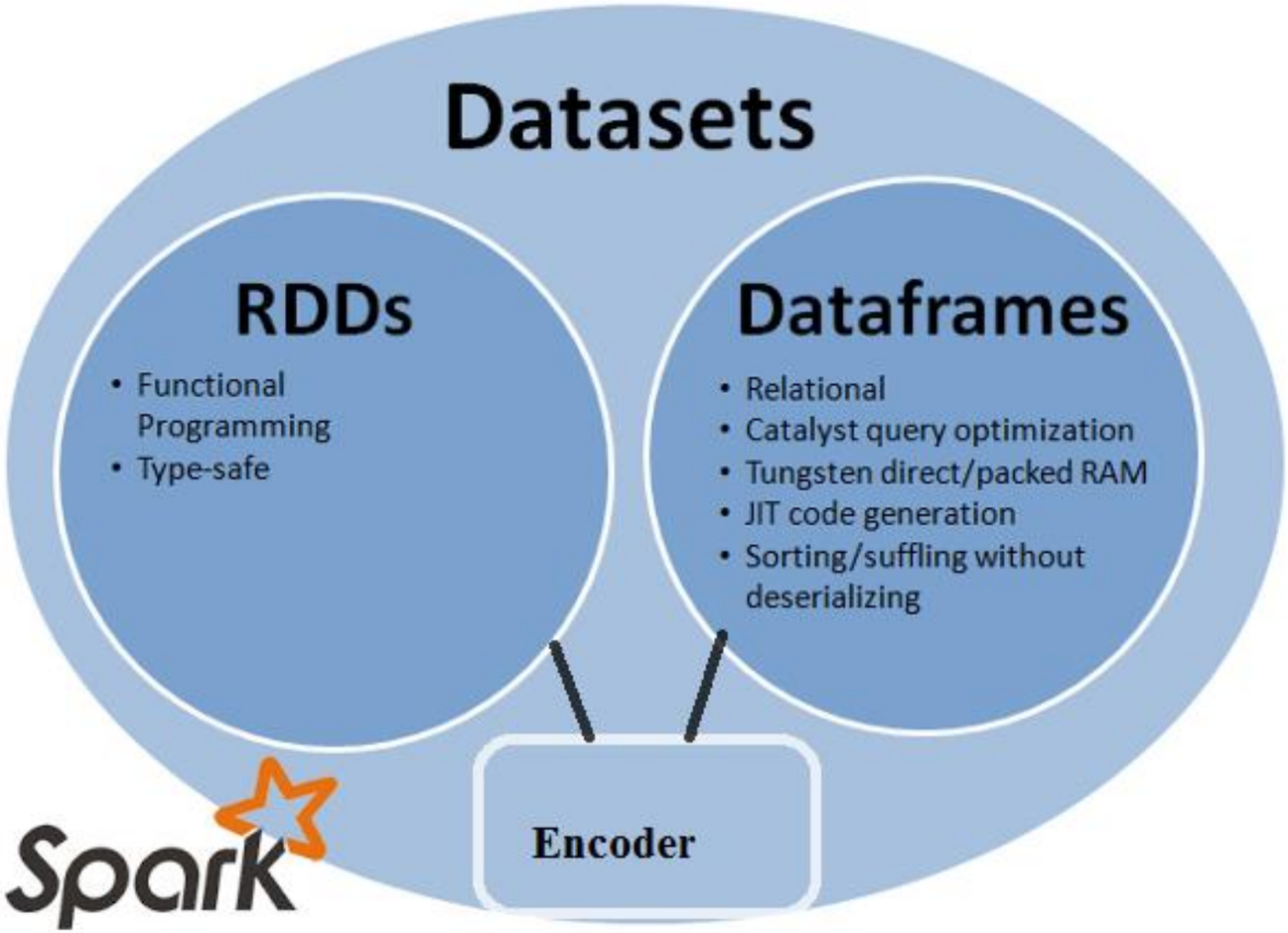
Architectures de Spark

Modes de Cluster

Kubernetes



Modèle de Programmation de Spark



Transformations
map (func)
flatMap(func)
filter(func)
groupByKey()
reduceByKey(func)
mapValues(func)
sample(...)
union(other)
distinct()
sortByKey()
...

Actions
reduce(func)
collect()
count()
first()
take(n)
saveAsTextFile(path)
countByKey()
foreach(func)
...

Modèle de Programmation de Spark

RDDs (Resilient Distributed Datasets)

Spark 1.0

Méthodes de transformation :
map(), filter() et reduce()

Les méthodes d'action :
collect() et saveAsObjectFile().

Exemple RDD

```
rdd.filter(_.age > 21) // transformation
   .map(_.last)// transformation
   .saveAsObjectFile("under21.bin") // action
```

Modèle de Programmation de Spark

DataFrames

Spark 1.3.

Concept de schéma pour décrire les données

Plan de requête relationnel

Exemple en style SQL

```
df.filter("age > 21");
```

Limitations :

Comme le code fait référence aux attributs de données par leur nom, le compilateur ne peut pas détecter d'erreurs. Si les noms des attributs sont incorrects, l'erreur ne sera détectée qu'au moment de l'exécution, lors de la création du plan de requête.

Modèle de Programmation de Spark

Dataset

Exemple en style SQL

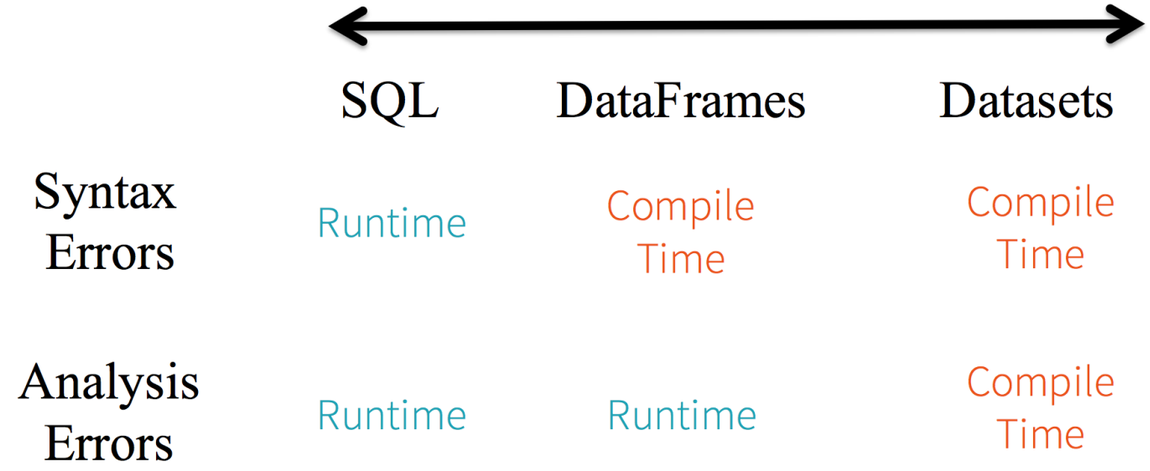
```
dataset.filter(_.age < 21);
```

Spark 1.6

Encodeurs

Beans

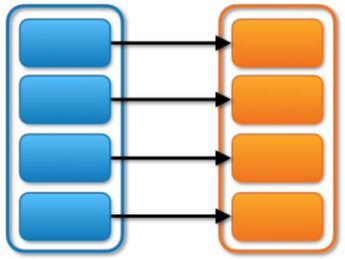
Evaluations diff. entre DataFrame & DataSet :



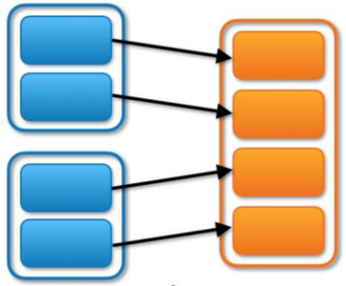
Modèle de Programmation de Spark

Transformations

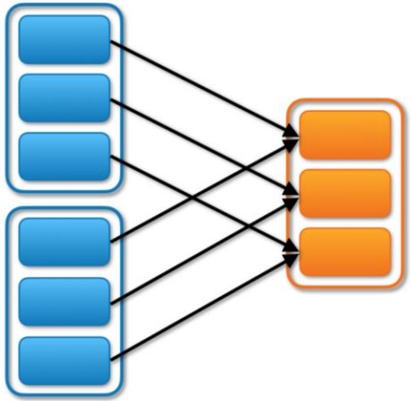
Narrow transformations



map, filter

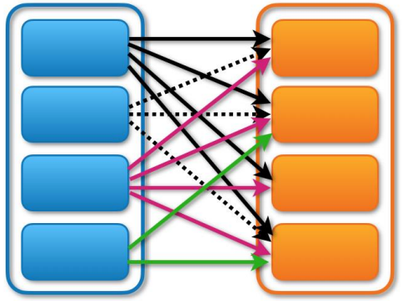


union

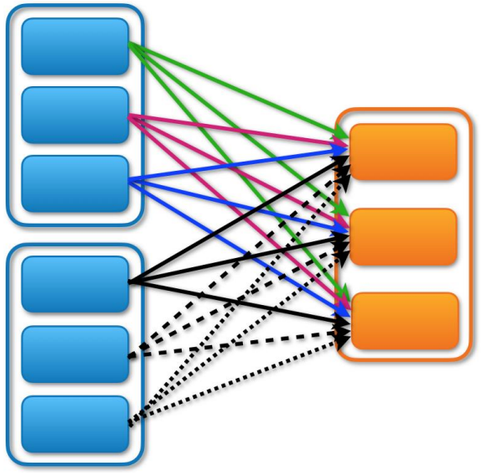


join w / inputs
co-partitioned

Wide transformations



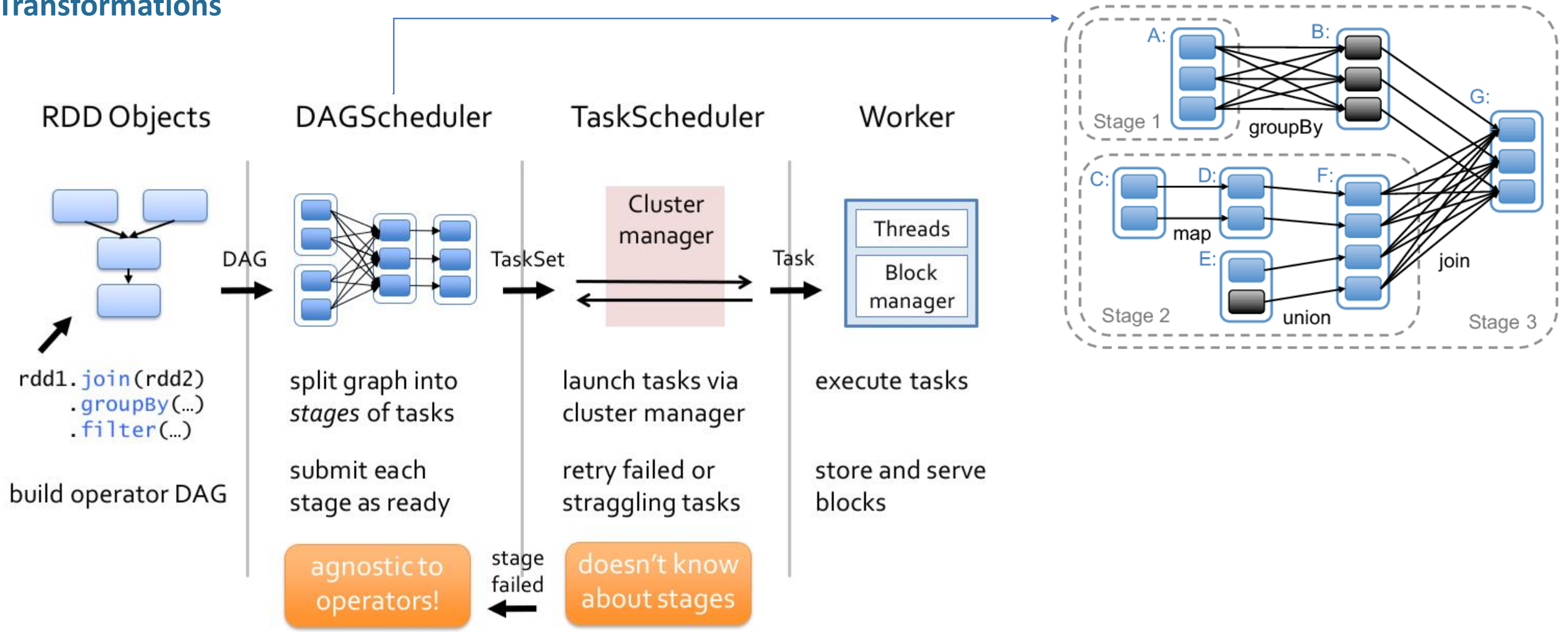
groupByKey



join w / inputs not
co-partitioned

Modèle de Programmation de Spark

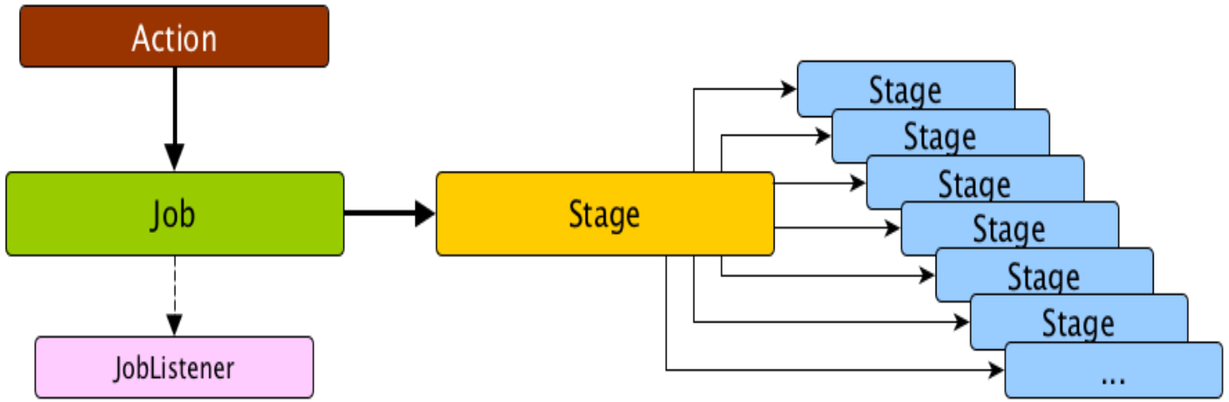
Transformations



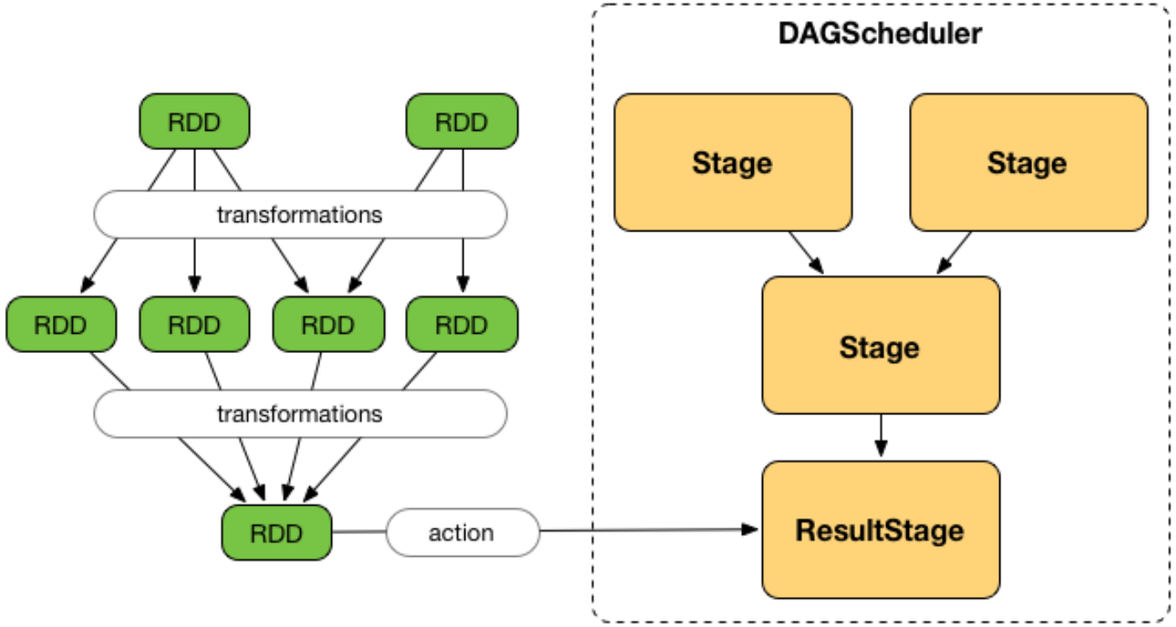
Modèle de Programmation de Spark

Actions

Stage Execution



DAGScheduler



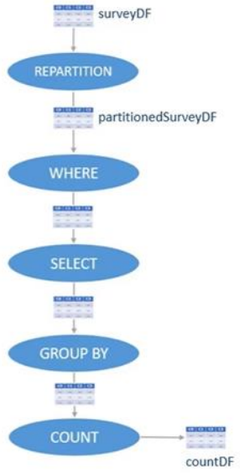
Modèle de Programmation de Spark

Actions

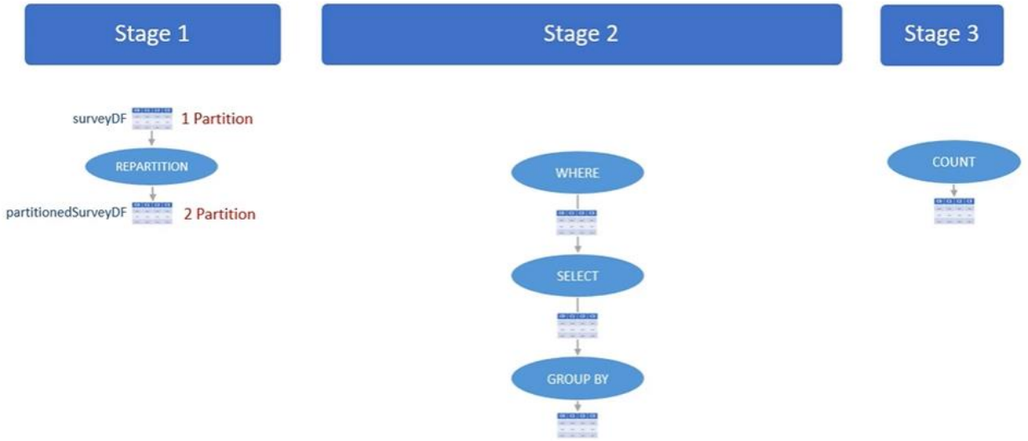
Plan logique 1 & 2

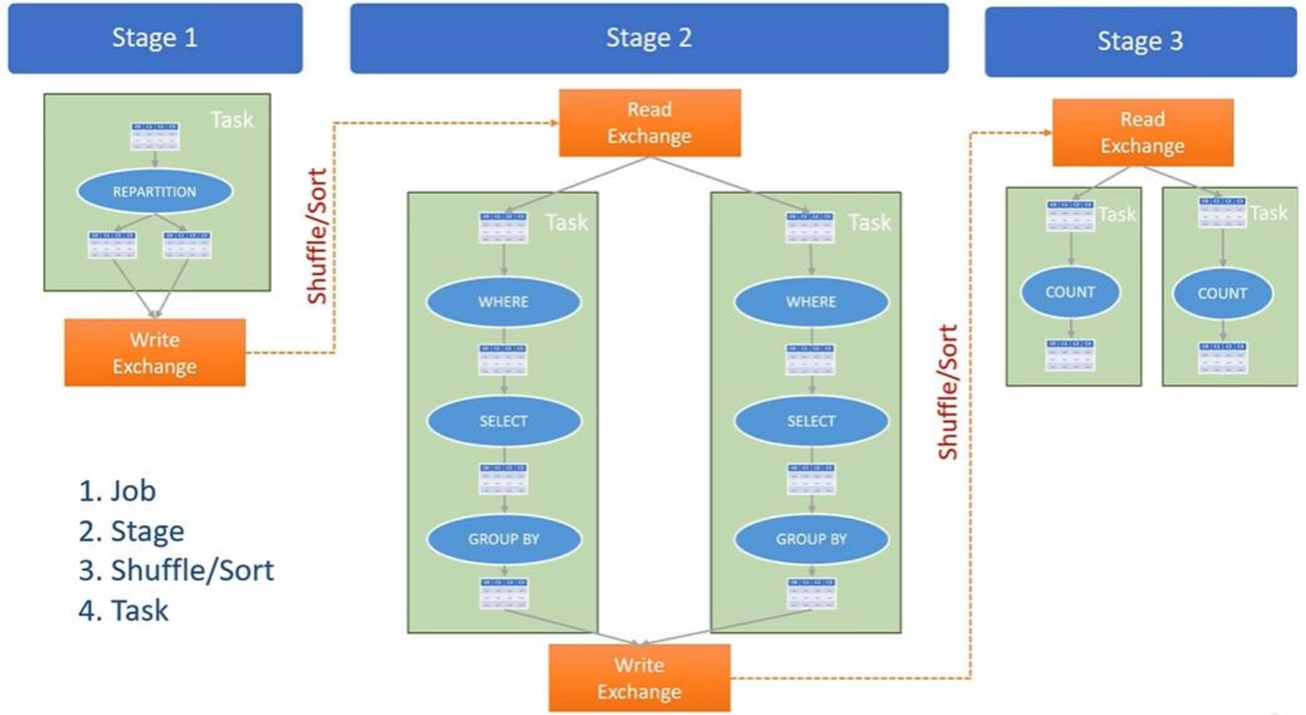
```
val partitionedSurveyDF = surveyRawDF.repartition( numPartitions = 2)
val countDF = partitionedSurveyDF.where( conditionExpr = "Age < 40")
  .select( col = "Age", cols = "Gender", "Country", "state")
  .groupBy( col1 = "Country")
  .count()
logger.info(countDF.collect().mkString("->"))
```

Job 1



Logical Plan

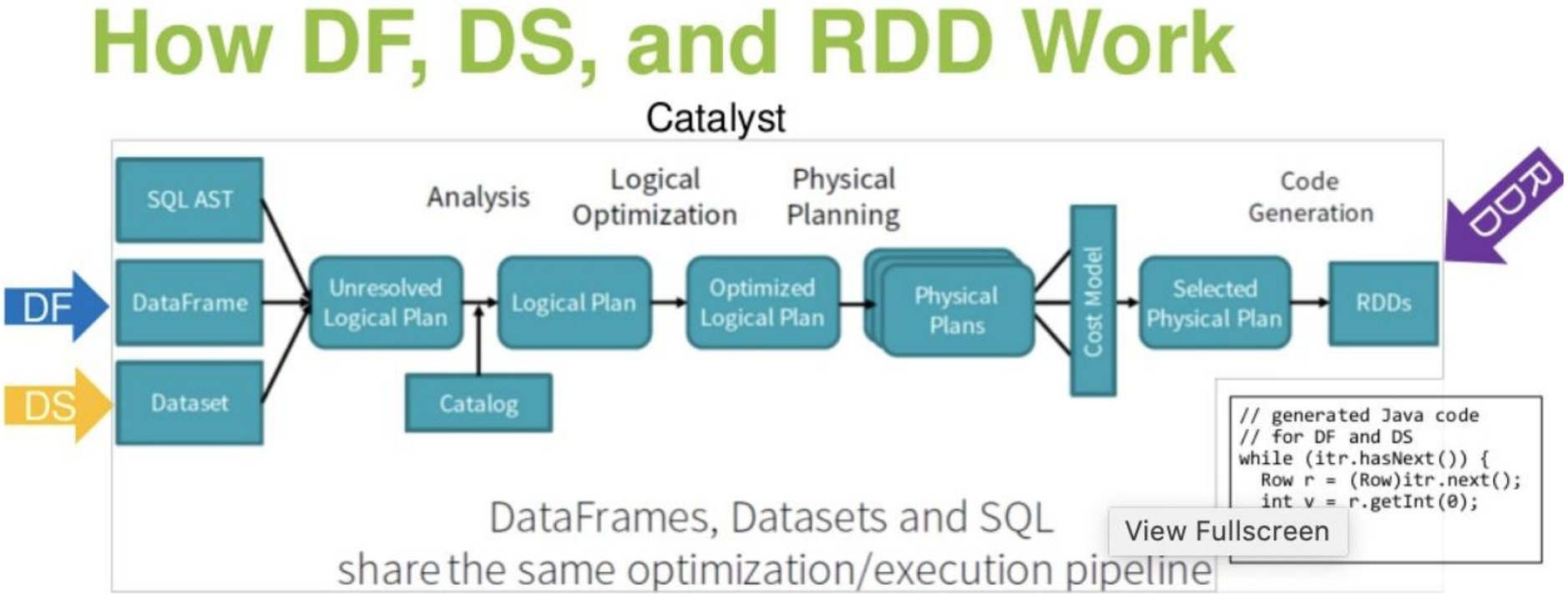




1. Job
2. Stage
3. Shuffle/Sort
4. Task

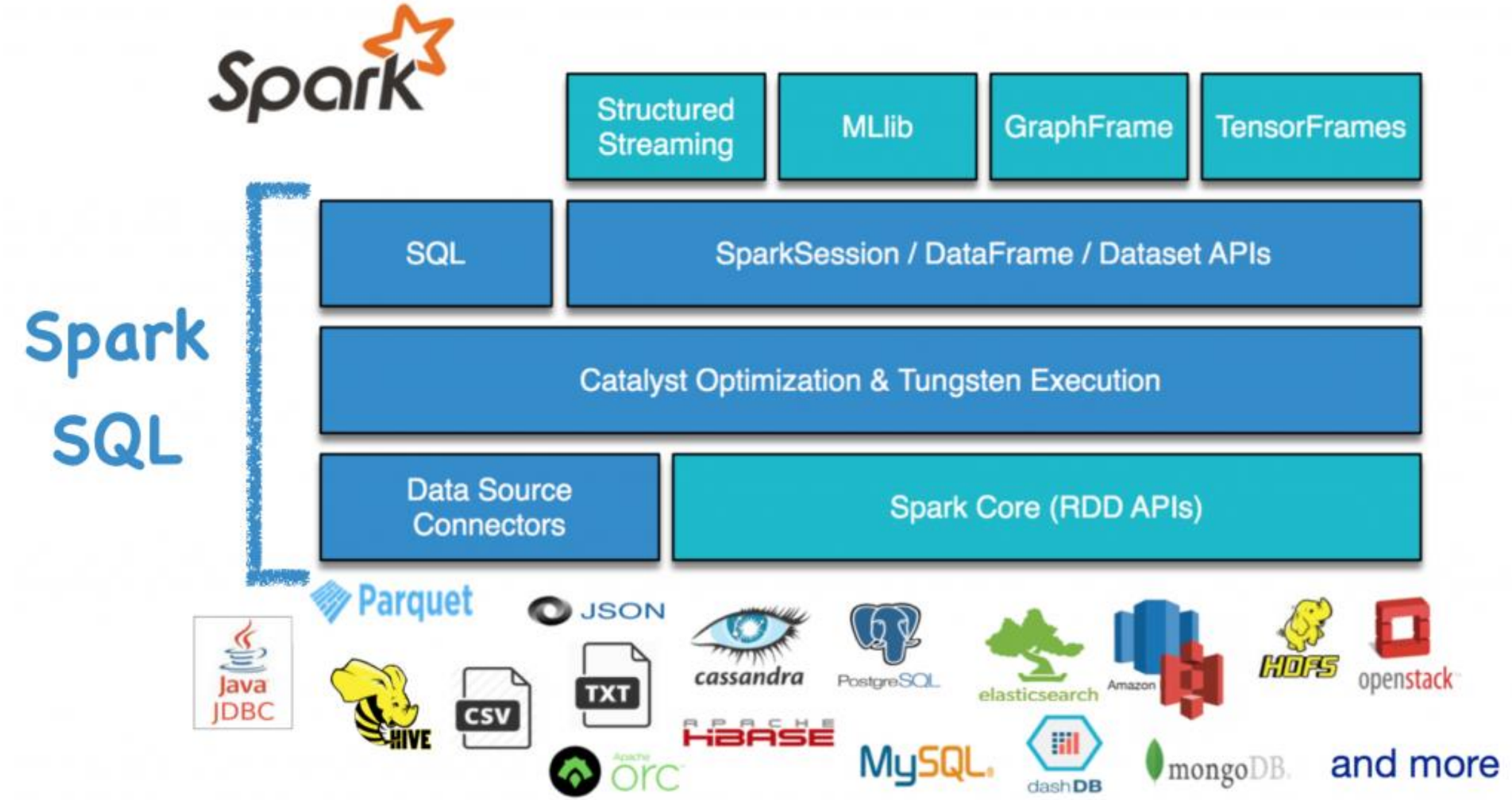
Modèle de Programmation de Spark

Synthèse

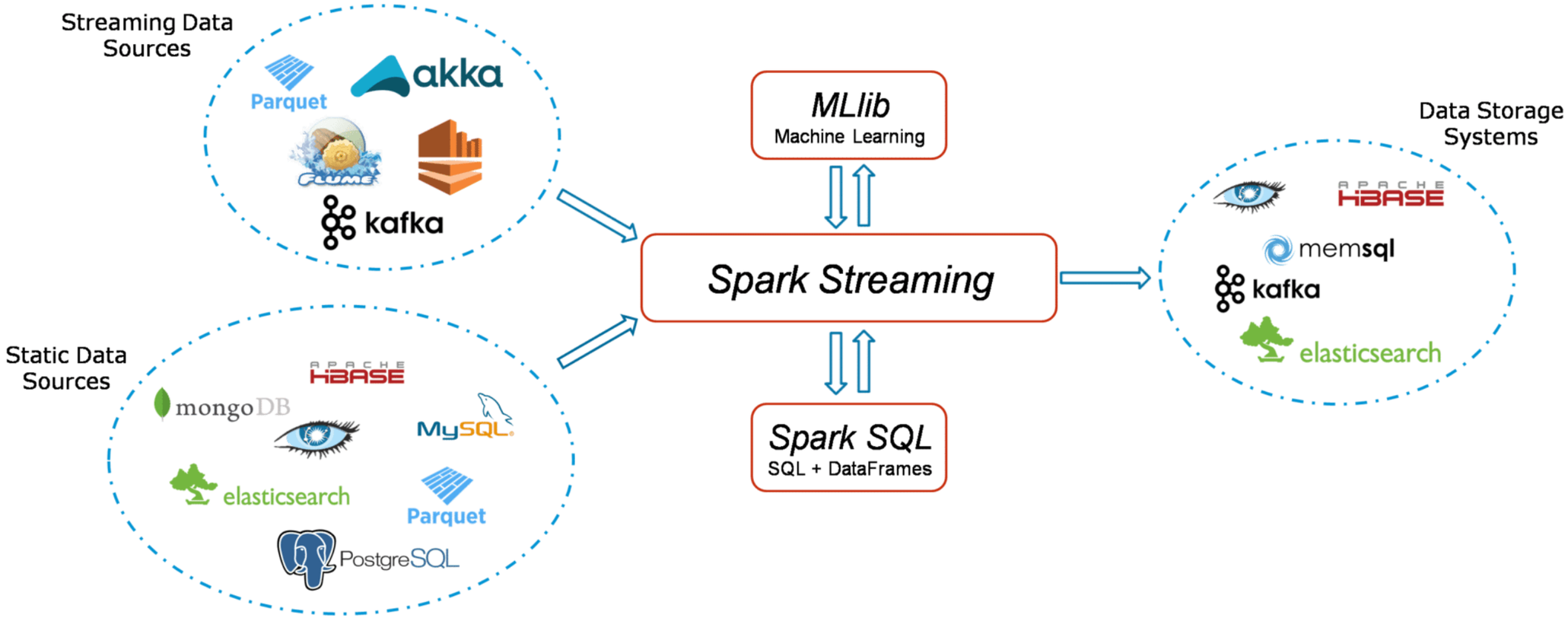



```
// select specific fields from the Dataset, apply a predicate using the where() method, convert to an RDD, and show first 10 RDD rows
val deviceEventsDS = ds.select($"device_name", $"cca3", $"c02_level").where($"c02_level" > 1300)

// convert to RDDs and take the first 10 rows
val eventsRDD = deviceEventsDS.rdd.take(10)
```

Spark SQL: Traitement de Données Structurées



Streaming avec Apache Spark



Traitement en Temps Réel

Micro-Batchs

DStreams (Discretized Stream)

Streaming Structuré Structuré

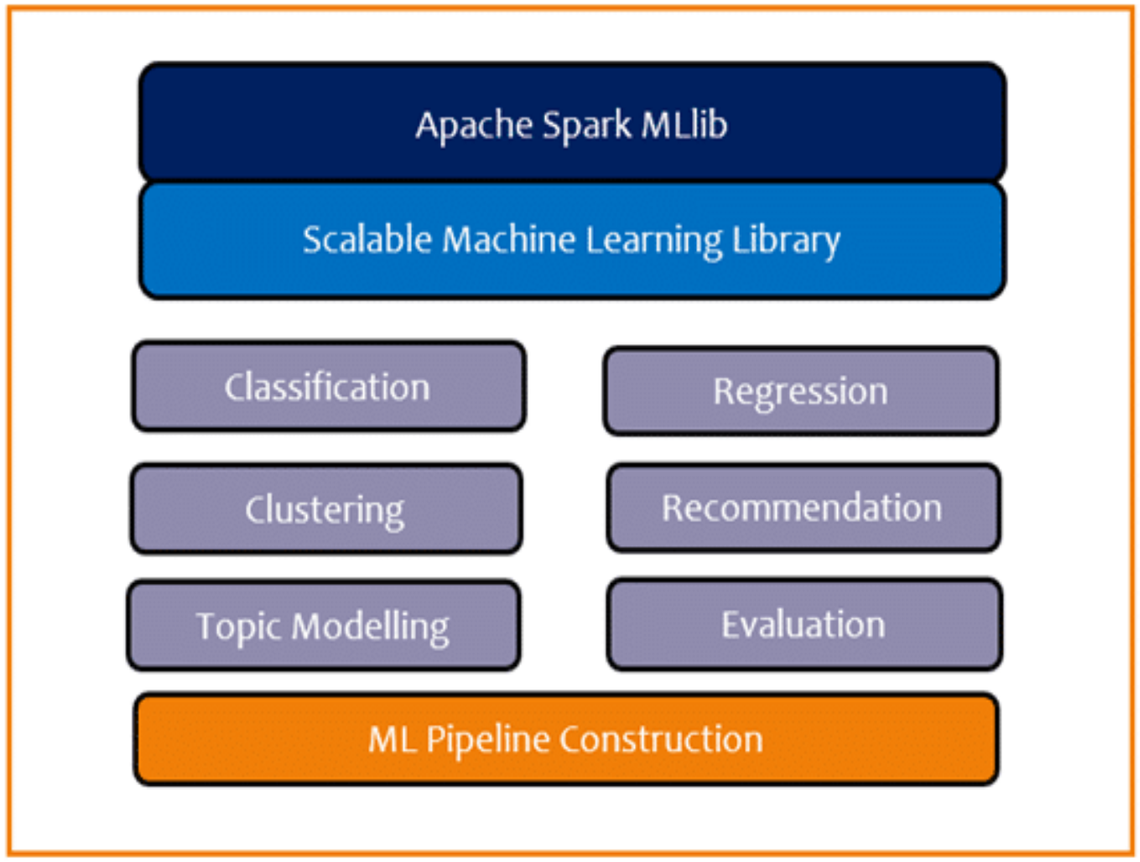
Exemple Spark SQL – Spark Streaming

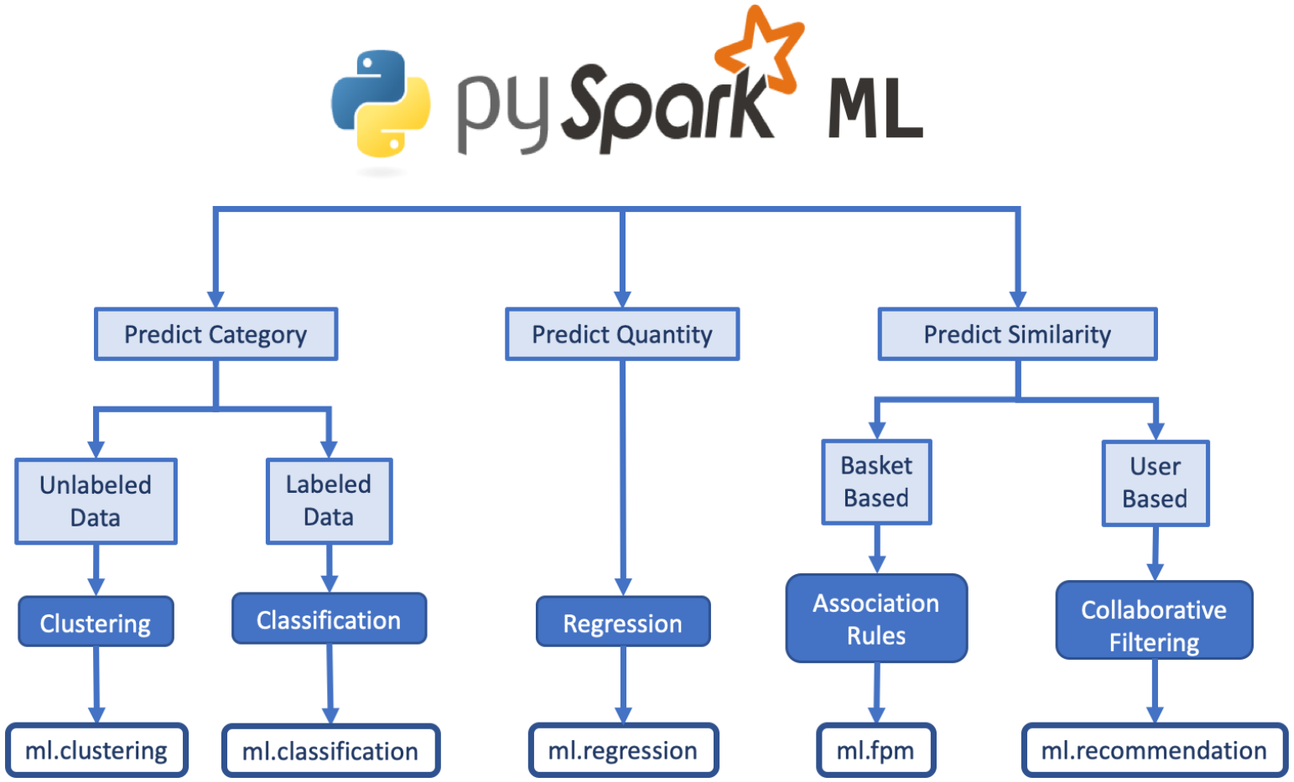
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Initialiser une SparkSession
spark = SparkSession.builder \
    .appName('Spark SQL and Streaming Example') \
    .enableHiveSupport() \
    .getOrCreate()

# Lire les données de la base de données PostgreSQL
jdbcDF = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:postgresql://localhost:5432/database") \
    .option("dbtable", "table") \
    .option("user", "username") \
    .option("password", "password") \
    .load()

# Enregistrer le DataFrame en tant que vue SQL temporaire
jdbcDF.createOrReplaceTempView("table")
```

```python
Effectuer une requête SQL
sqlDF = spark.sql("SELECT * FROM table WHERE some_condition")

# Créer un DataFrame à partir du flux de données
streamingDF = spark.readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Appliquer une transformation sur le flux de données
transformedDF = streamingDF.selectExpr("CAST(value AS STRING)")

# Écrire le DataFrame transformé dans la base de données Hive
transformedDF.write \
    .format("hive") \
    .saveAsTable("hive_table")
```

Machine Learning avec Spark (MLlib)

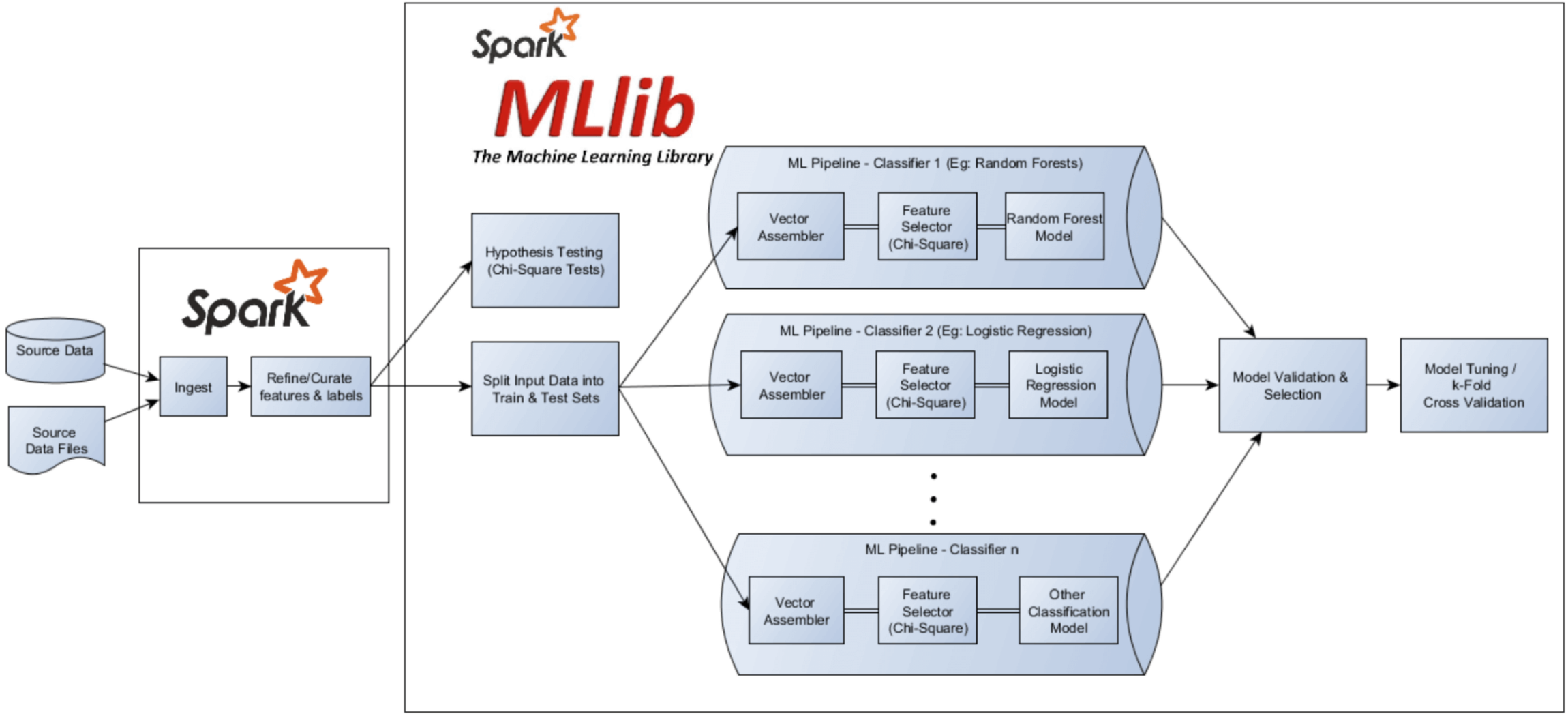
Algorithmes de Machine Learning





Machine Learning avec Spark (MLlib)

API de Pipeline



Machine Learning avec Spark (MLlib)

Algorithmes de Machine Learning

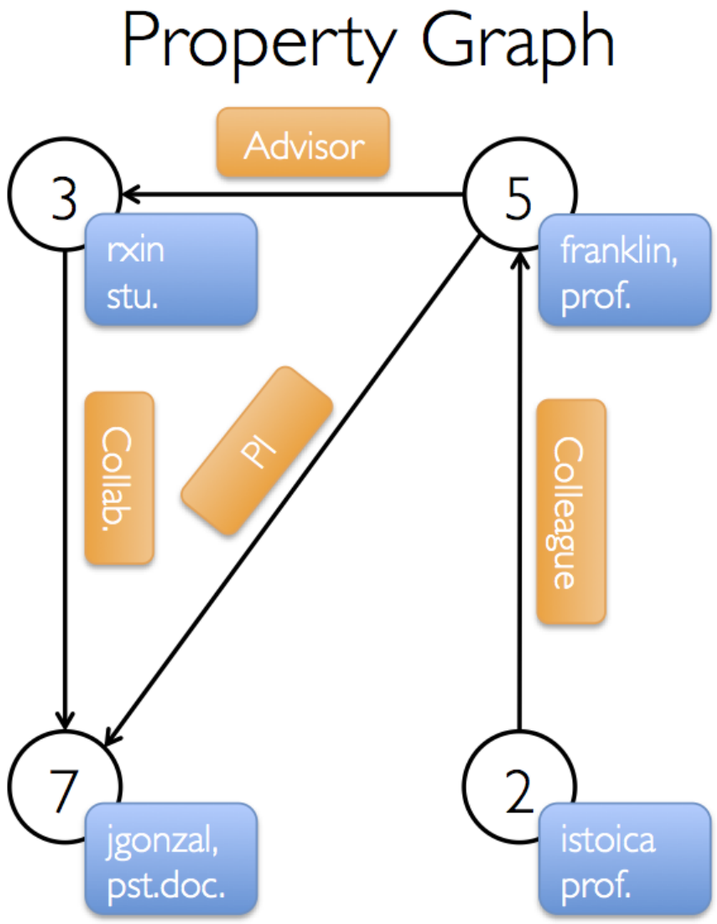
Régression Linéaire

```python
from pyspark.sql import SparkSession
from pyspark.ml.regression import LinearRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

spark = SparkSession.builder.appName('linear_regression').getOrCreate()

# Charger les données
data = spark.read.csv("votre_fichier.csv", inferSchema=True, header=True)

# Créer un VectorAssembler qui convertit toutes les caractéristiques en un seul vecteur
assembler = VectorAssembler(
  inputCols=["feature1", "feature2"],
  outputCol="features")
```

```python
# Transformer les données
output = assembler.transform(data)
# Sélectionner les colonnes "features" et "target"
final_data = output.select("features", "target")

# Diviser les données en un ensemble d'entraînement et un ensemble de test
train_data, test_data = final_data.randomSplit([0.7, 0.3])

# Initialiser le modèle de régression linéaire
lr = LinearRegression(labelCol='target')

# Entraîner le modèle
lrModel = lr.fit(train_data)

# Prédire sur l'ensemble de test
predictions = lrModel.transform(test_data)

# Afficher les prédictions
predictions.show()
```

Traitement de Graphes avec Spark (GraphX)

Représentation des Graphes



Vertex Table

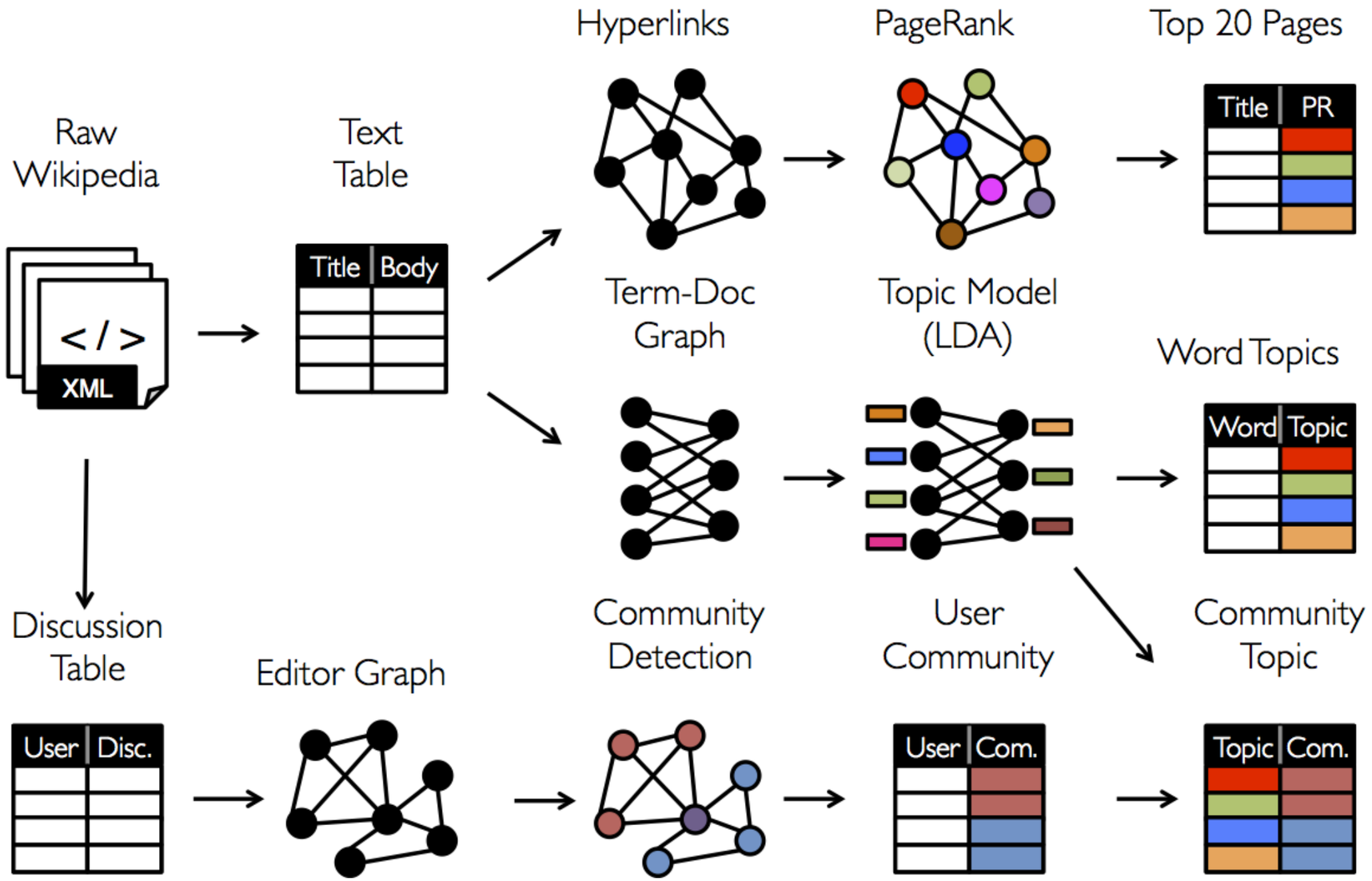
Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Traitement de Graphes avec Spark (GraphX)

Algorithmes de Graphes



Traitement de Graphes avec Spark (GraphX)

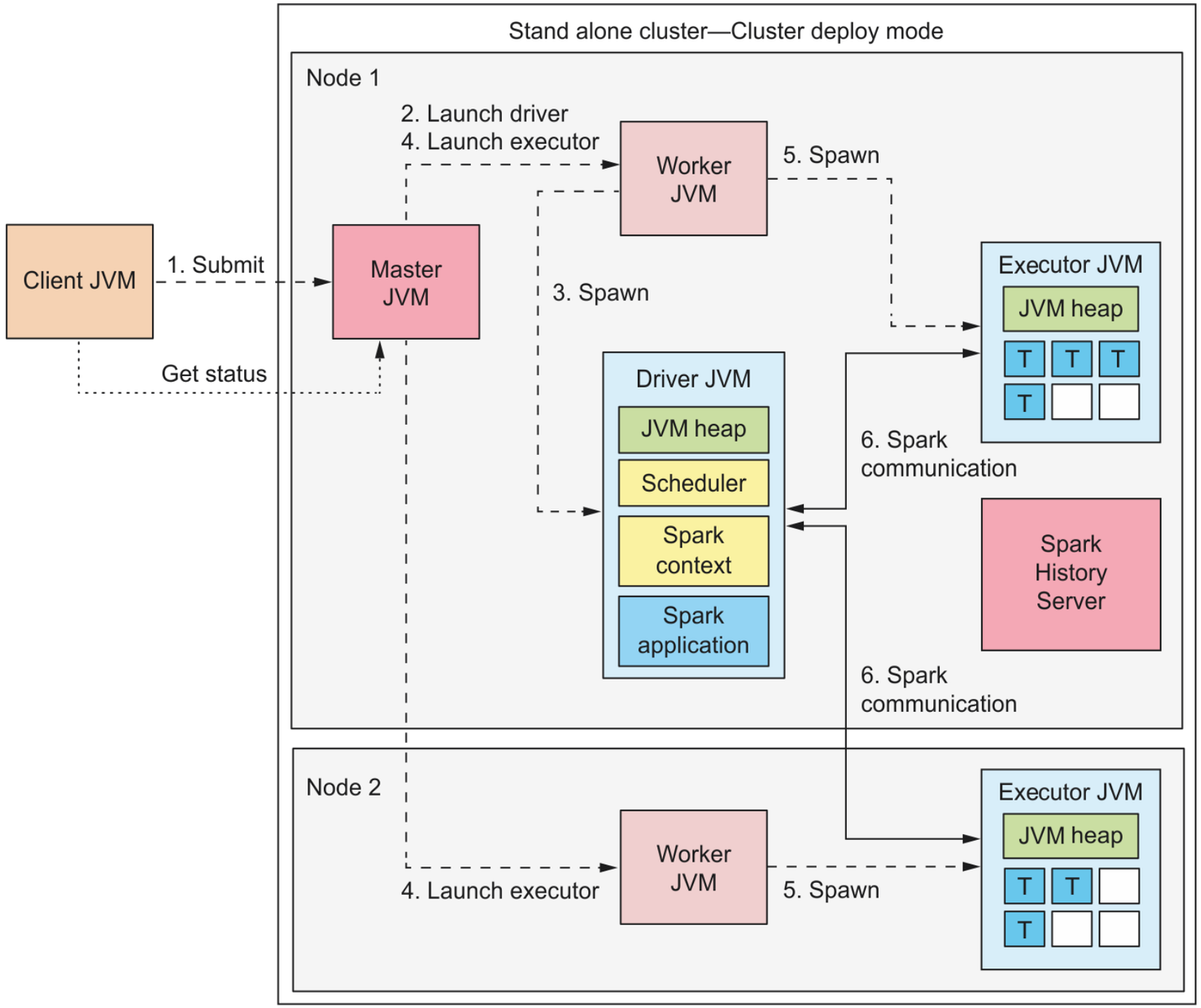
Algorithmes de Graphes

PageRank

```scala
import org.apache.spark.graphx.GraphLoader

// Load the edges as a graph
val graph = GraphLoader.edgeListFile(sc, "data/graphx/followers.txt")
// Run PageRank
val ranks = graph.pageRank(0.0001).vertices
// Join the ranks with the usernames
val users = sc.textFile("data/graphx/users.txt").map { line =>
  val fields = line.split(",")
  (fields(0).toLong, fields(1))
}
val ranksByUsername = users.join(ranks).map {
  case (id, (username, rank)) => (username, rank)
}
// Print the result
println(ranksByUsername.collect().mkString("\n"))
```

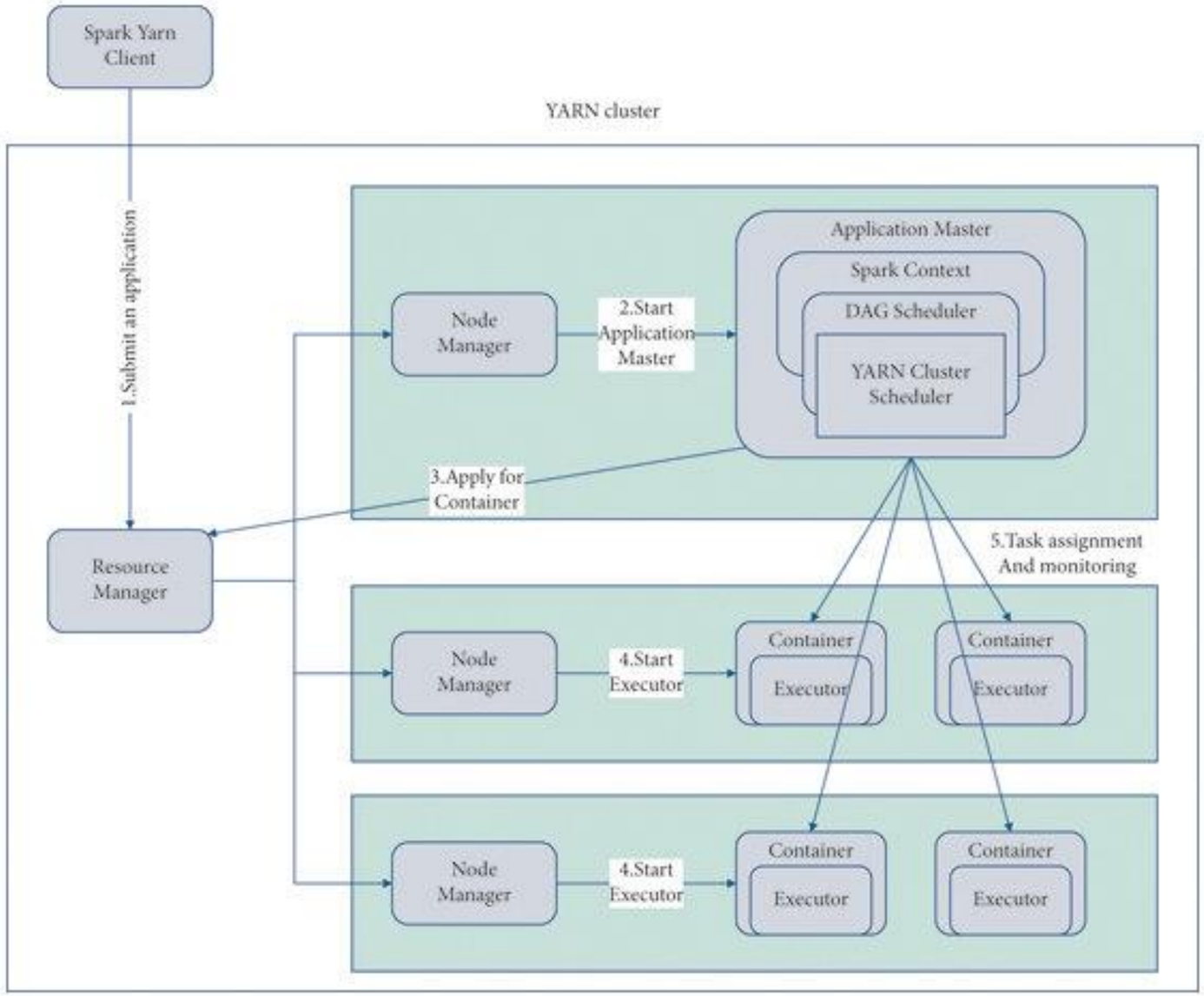
Options de Déploiement de Spark

Mode Local



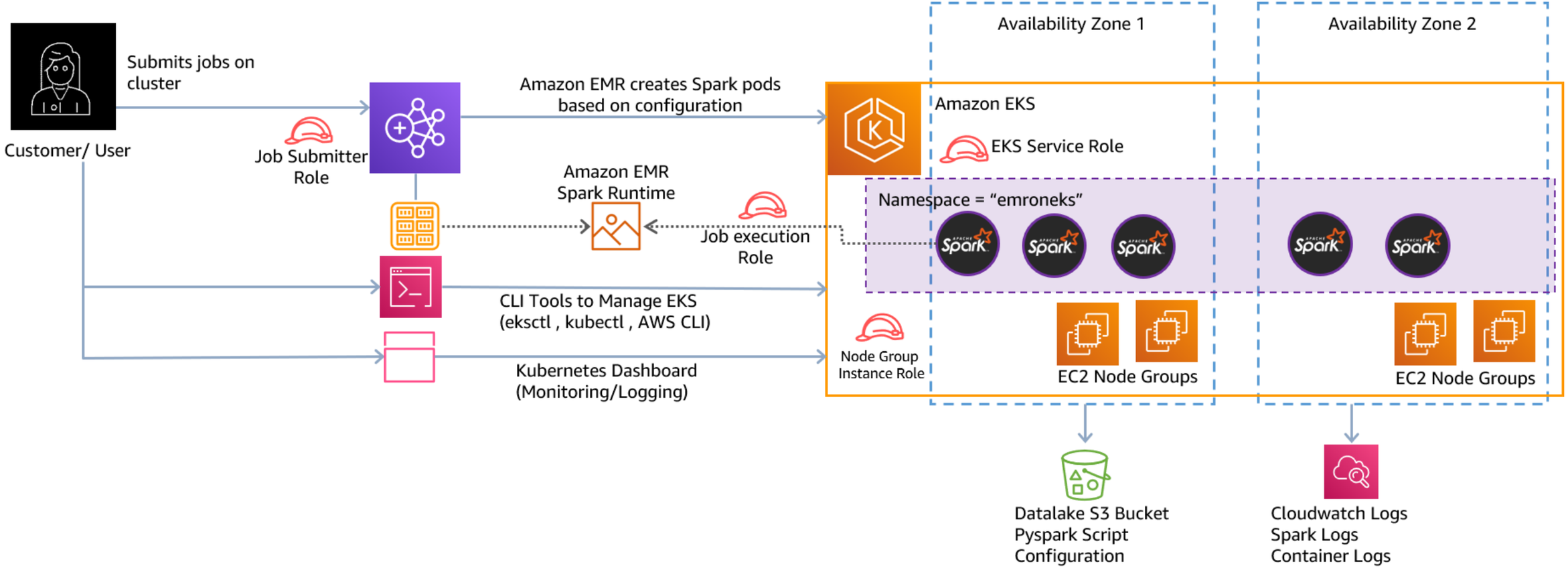
Options de Déploiement de Spark

Mode YARN Cluster



Options de Déploiement de Spark

Déploiement sur le Cloud



Cas d'Utilisation et Applications

Cas d’utilisations réels

Recommandation de contenu en temps réel.

Analyse des données de trajets pour l'optimisation de l'algorithme de tarification.

Analyse des données des utilisateurs pour améliorer l'expérience client.

Applications industrielles

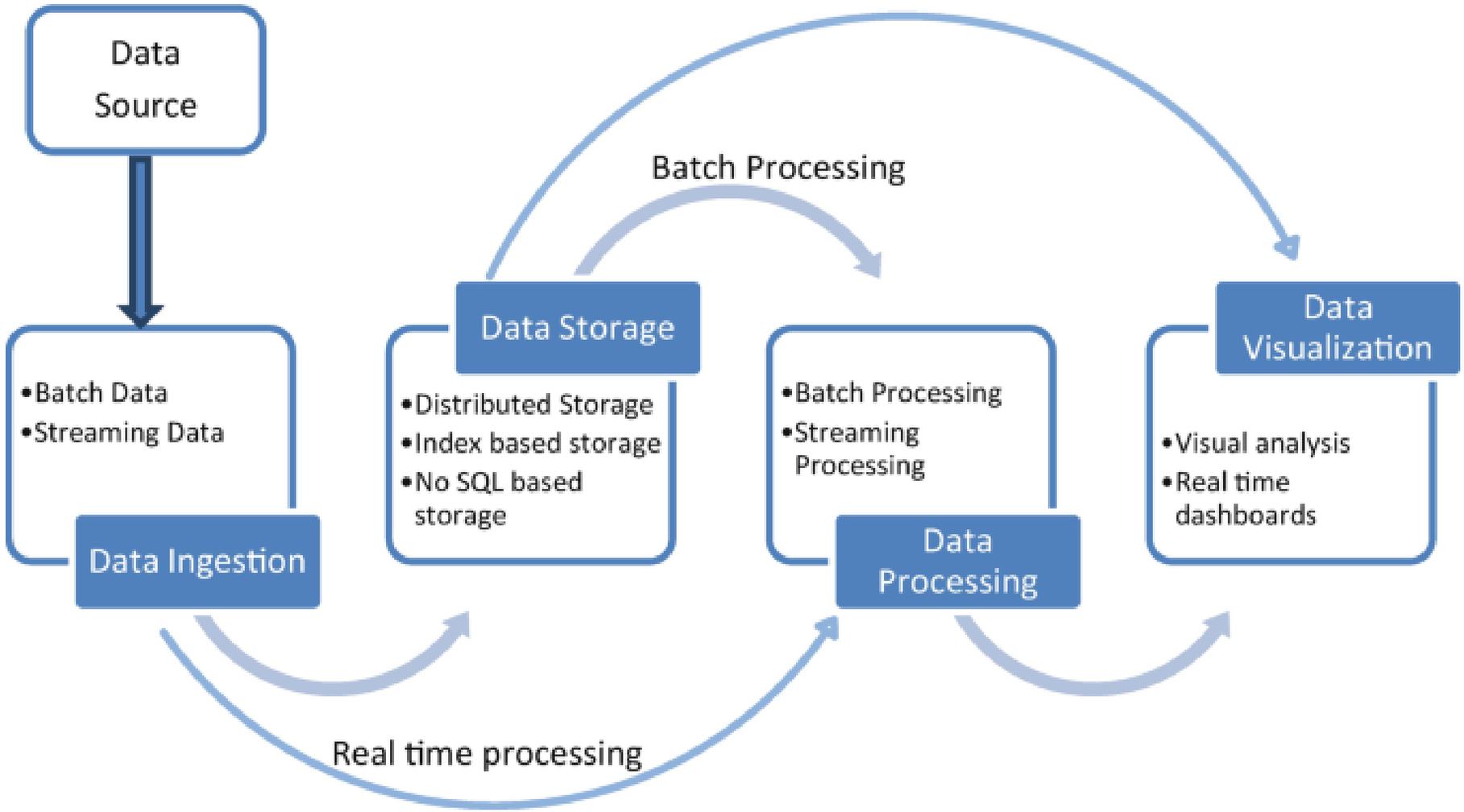
Finance : Détection de fraudes en temps réel.

Santé : Analyse de données génomiques.

Télécommunications : Analyse de données de réseau pour la détection de pannes et l'optimisation de la performance.

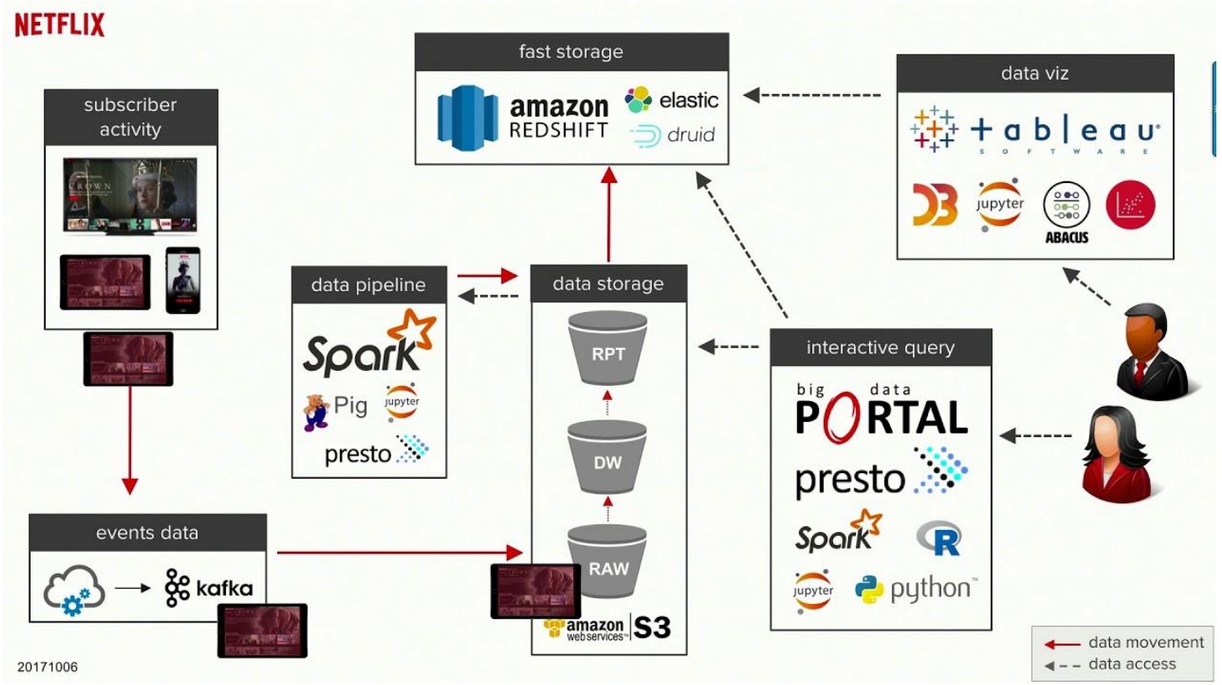
Cas d'Utilisation et Applications

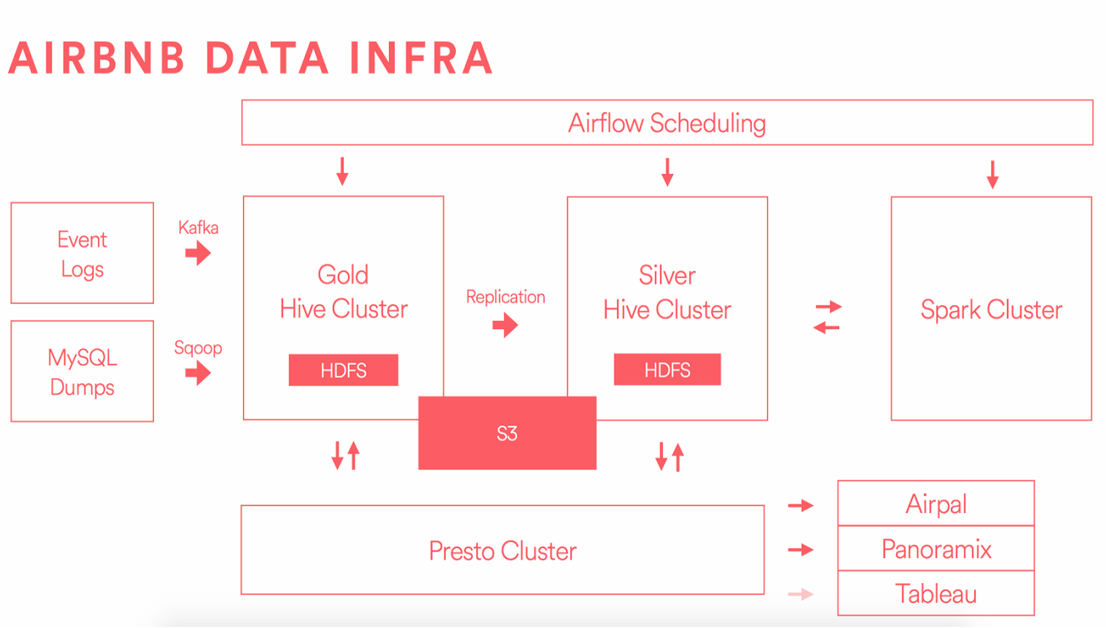
Cas d’utilisations reels en temps réel



Cas d'Utilisation et Applications

Exemples de Cas d’utilisations réels





Avantages et Limites de Spark

Avantages

Performances élevées

Flexibilité et extensibilité

Support de langages multiples

Large communauté et écosystème

Inconvénients

Complexité de la configuration

Ressources requises

Courbe d'apprentissage

Dépendance à Java ou Scala

Alternatives à Spark

Hadoop MapReduce

Hadoop MapReduce est un framework de traitement de données en lots qui a été largement utilisé avant l'arrivée de Spark. Il est fiable mais peut être lent pour certains types de tâches de traitement.

Flink

Apache Flink est une plateforme de calcul distribué open source conçue pour le traitement de flux de données en temps réel. Il est connu pour ses performances élevées et son support pour la programmation de fenêtres de temps.

Storm

Apache Storm est un framework de traitement de flux de données en temps réel distribué, conçu pour le traitement de flux de données à haut débit. Il est souvent utilisé pour des applications de streaming telles que le suivi des événements, l'analyse en temps réel et les systèmes de recommandation.

Démo

Gestion de Données avec Spark et Hive : De la Connexion au Cluster à l'Exploration des Données

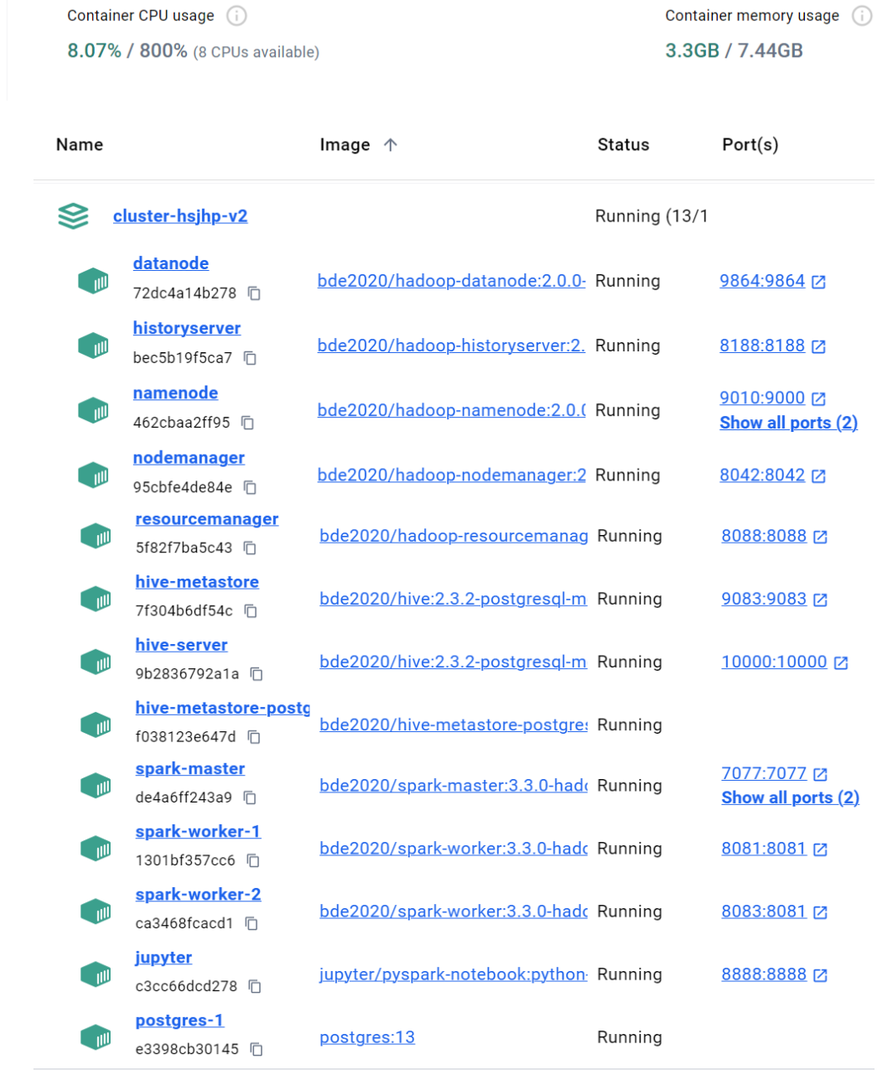
1. **Configuration de l'environnement** : Cluster Big Data Hadoop, Spark, Hive (Postgres) et Jupyter
2. **Chargement des données dans Hadoop** : Charger le fichier 'demo.csv' dans Hadoop en utilisant la commande **hadoop fs -put**
3. **Transformation des données avec Spark** : Transformer les données en les nettoyant et en convertissant la colonne **age** en **int**
4. **Stockage des données transformées dans Hive** : Stocker les données dans une table '**adulte**' en utilisant la methode **write.mode()** et l'option **saveAsTable()**
5. **Interrogation des données avec SQL** : Utiliser la methode **spark.sql()** pour afficher les adultes
6. **Visualisation des données** : Utiliser la bibliothèque **matplotlib** pour créer un graphique à partir de vos données

Démo

Configuration de l'environnement

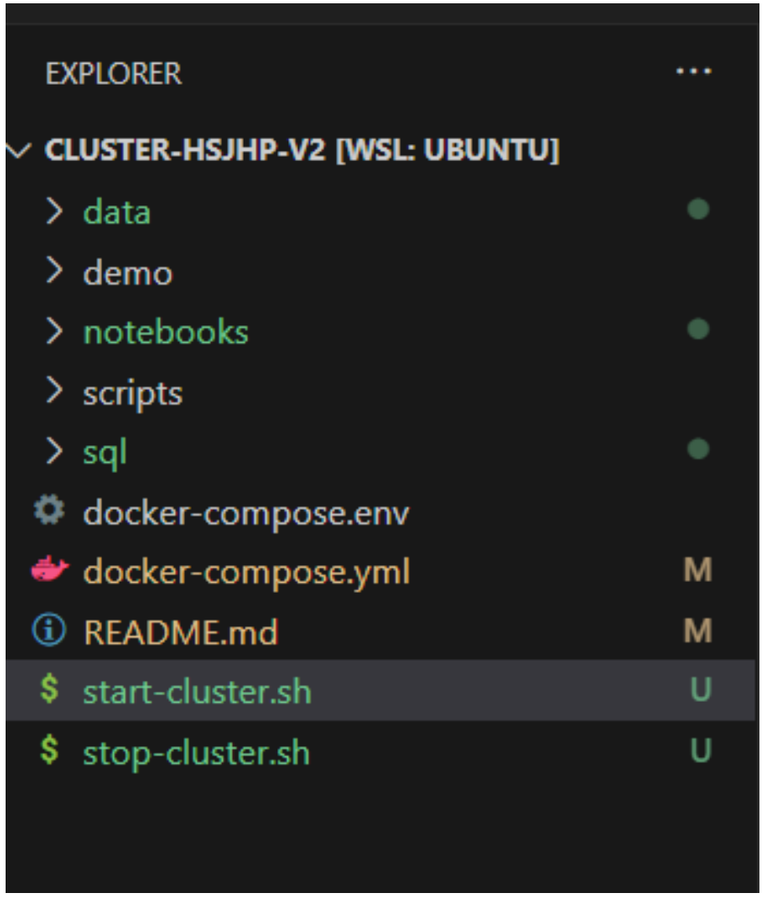
```
barryma@DESKTOP-LBFN3RA
(base) barryma@DESKTOP-LBFN3RA:~$ conda deactivate
barryma@DESKTOP-LBFN3RA:~$ docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"
NAMES                       STATUS                    PORTS
hive-server                 Up 13 minutes             0.0.0.0:10000->10000/tcp, 10002/tcp
spark-worker-2              Up 13 minutes             0.0.0.0:8083->8081/tcp
spark-worker-1              Up 13 minutes             0.0.0.0:8081->8081/tcp
cluster-hsjhp-v2-postgres-1 Up 13 minutes             5432/tcp
namenode                    Up 13 minutes (healthy)   0.0.0.0:9870->9870/tcp, 0.0.0.0:9010->9000/tcp
resourcemanager             Up 12 minutes (healthy)   0.0.0.0:8088->8088/tcp
spark-master                Up 13 minutes             0.0.0.0:7077->7077/tcp, 6066/tcp, 0.0.0.0:8080->8080/tcp
hive-metastore-postgresql   Up 13 minutes             5432/tcp
nodemanager                 Up 13 minutes (healthy)   0.0.0.0:8042->8042/tcp
jupyter                     Up 13 minutes (healthy)   4040/tcp, 0.0.0.0:8888->8888/tcp
hive-metastore              Up 13 minutes             10000/tcp, 0.0.0.0:9083->9083/tcp, 10002/tcp
historyserver               Up 13 minutes (healthy)   0.0.0.0:8188->8188/tcp
datanode                    Up 13 minutes (healthy)   0.0.0.0:9864->9864/tcp
```

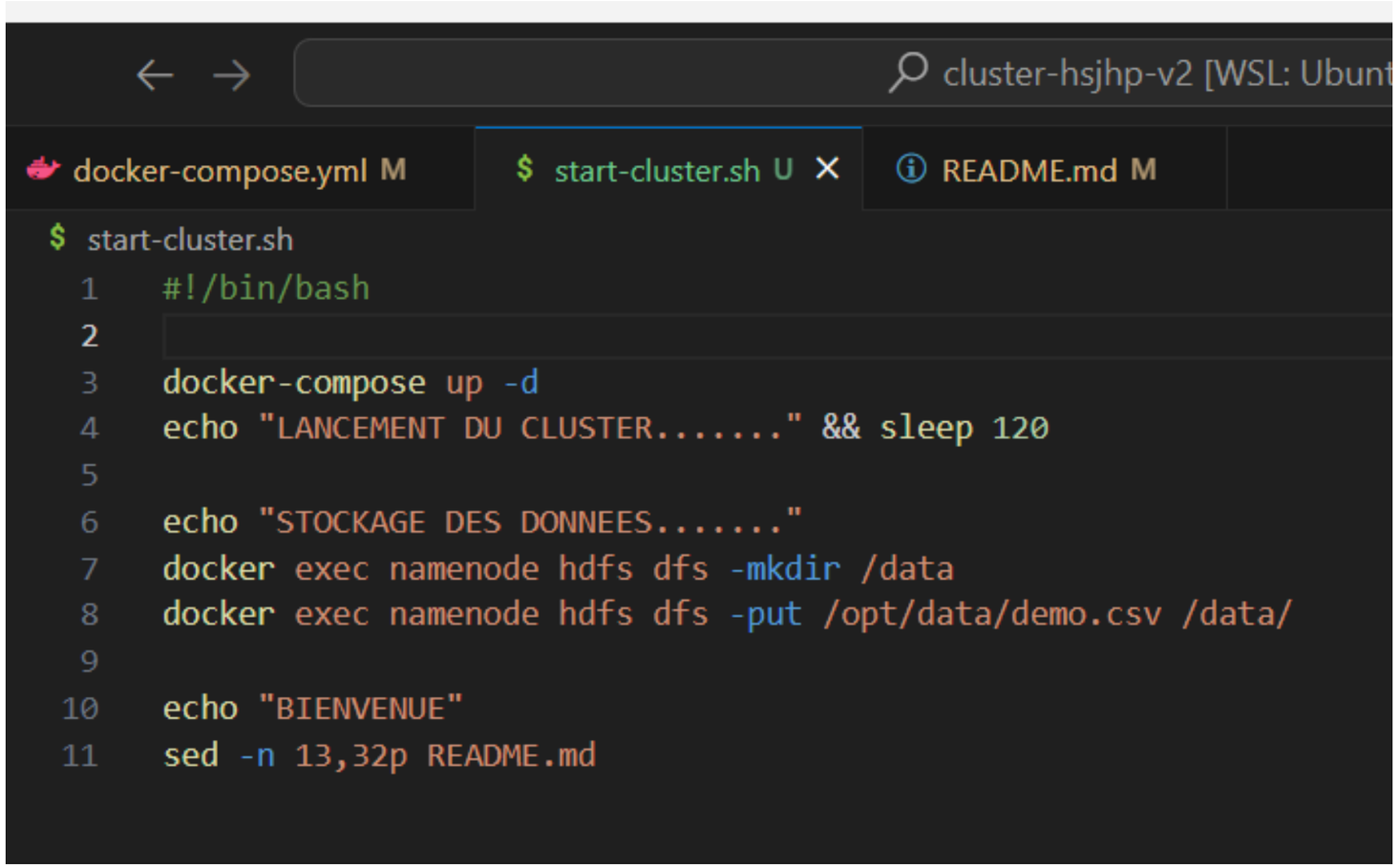
```
barryma@DESKTOP-LBFN3RA
barryma@DESKTOP-LBFN3RA:~$ docker stats --no-stream
CONTAINER ID   NAME                        CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I
/O    PIDS
9b2836792a1a   hive-server                 0.87%     328.8MiB / 7.623GiB   4.21%     7.04kB / 7.34kB   0B / 0B
      37
ca3468fcacd1   spark-worker-2              0.13%     275.3MiB / 7.623GiB   3.53%     20.5kB / 149kB    0B / 0B
      41
1301bf357cc6   spark-worker-1              0.15%     270.6MiB / 7.623GiB   3.47%     20.5kB / 149kB    0B / 0B
      41
e3398cb30145   cluster-hsjhp-v2-postgres-1 0.00%     21.41MiB / 7.623GiB   0.27%     2.21kB / 0B       0B / 0B
      7
462cbaa2ff95   namenode                    0.23%     297.9MiB / 7.623GiB   3.82%     338kB / 301kB     0B / 0B
      64
5f82f7ba5c43   resourcemanager             0.78%     475.1MiB / 7.623GiB   6.09%     588kB / 188kB     0B / 0B
      237
de4a6ff243a9   spark-master                0.14%     283.3MiB / 7.623GiB   3.63%     300kB / 36.8kB    0B / 0B
      42
f038123e647d   hive-metastore-postgresql   0.00%     45.49MiB / 7.623GiB   0.58%     183kB / 153kB     0B / 0B
      10
95cbfe4de84e   nodemanager                 0.71%     303.4MiB / 7.623GiB   3.89%     172kB / 565kB     0B / 0B
      99
c3cc66dcd278   jupyter                     0.00%     148.1MiB / 7.623GiB   1.90%     226kB / 458kB     0B / 0B
      15
7f304b6df54c   hive-metastore              0.89%     382MiB / 7.623GiB     4.89%     159kB / 183kB     0B / 0B
      32
bec5b19f5ca7   historyserver               0.24%     266MiB / 7.623GiB     3.41%     4.26kB / 1.95kB   0B / 0B
      54
72dc4a14b278   datanode                    0.95%     284.2MiB / 7.623GiB   3.64%     73.2kB / 304kB    0B / 0B
```



Démo

Chargement des données dans Hadoop

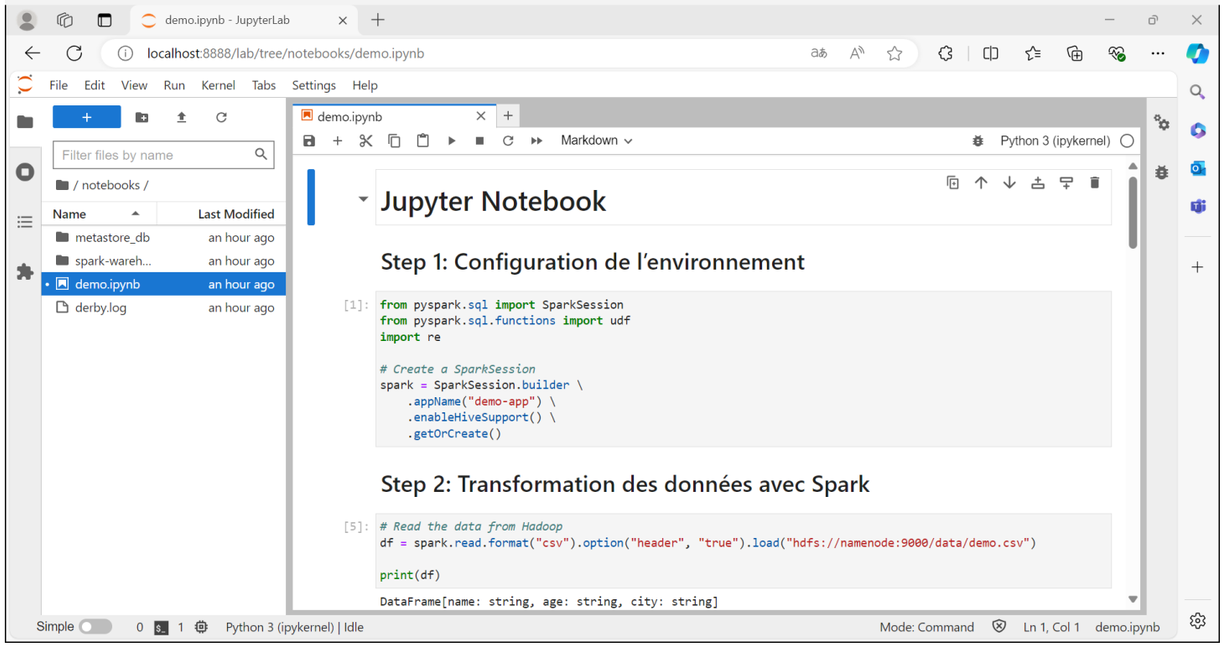


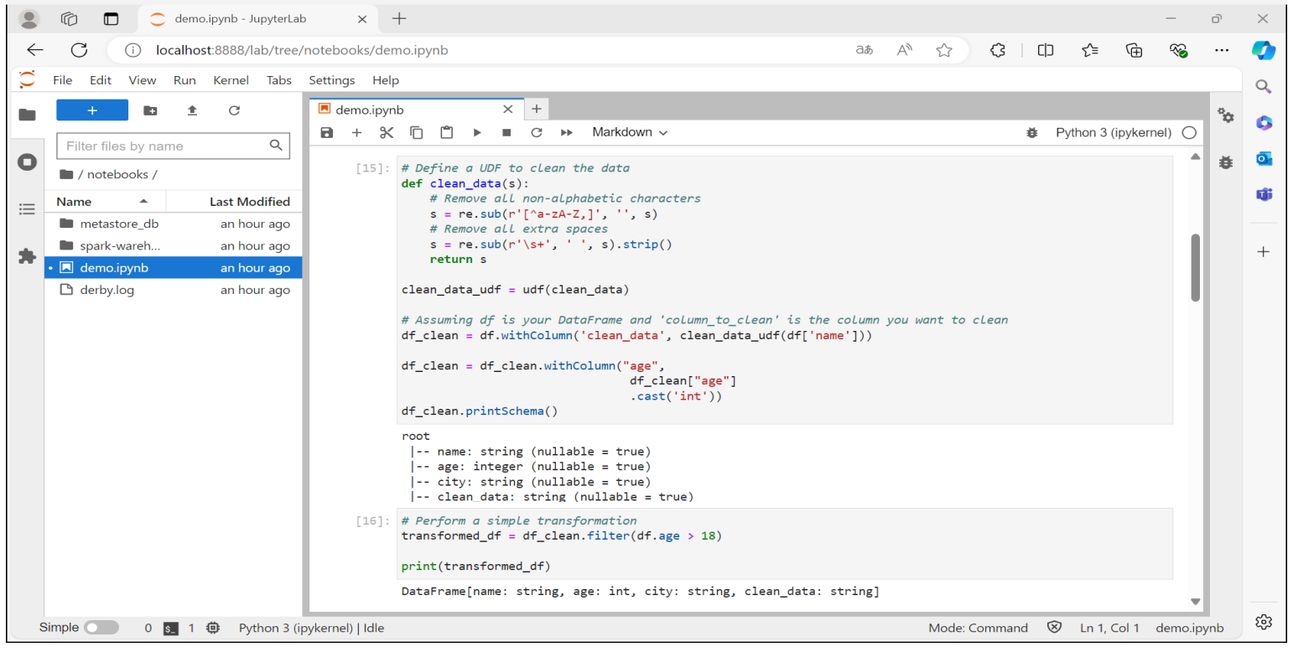



```
(base) barryma@DESKTOP-LBFN3RA:~/Workspace/cluster/cluster-hsjhp-v2$ docker exec namenode hdfs dfs -mkdir /data
(base) barryma@DESKTOP-LBFN3RA:~/Workspace/cluster/cluster-hsjhp-v2$ docker exec namenode hdfs dfs -put /opt/data/demo.csv /data/
2024-07-19 12:31:12,502 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

Démo

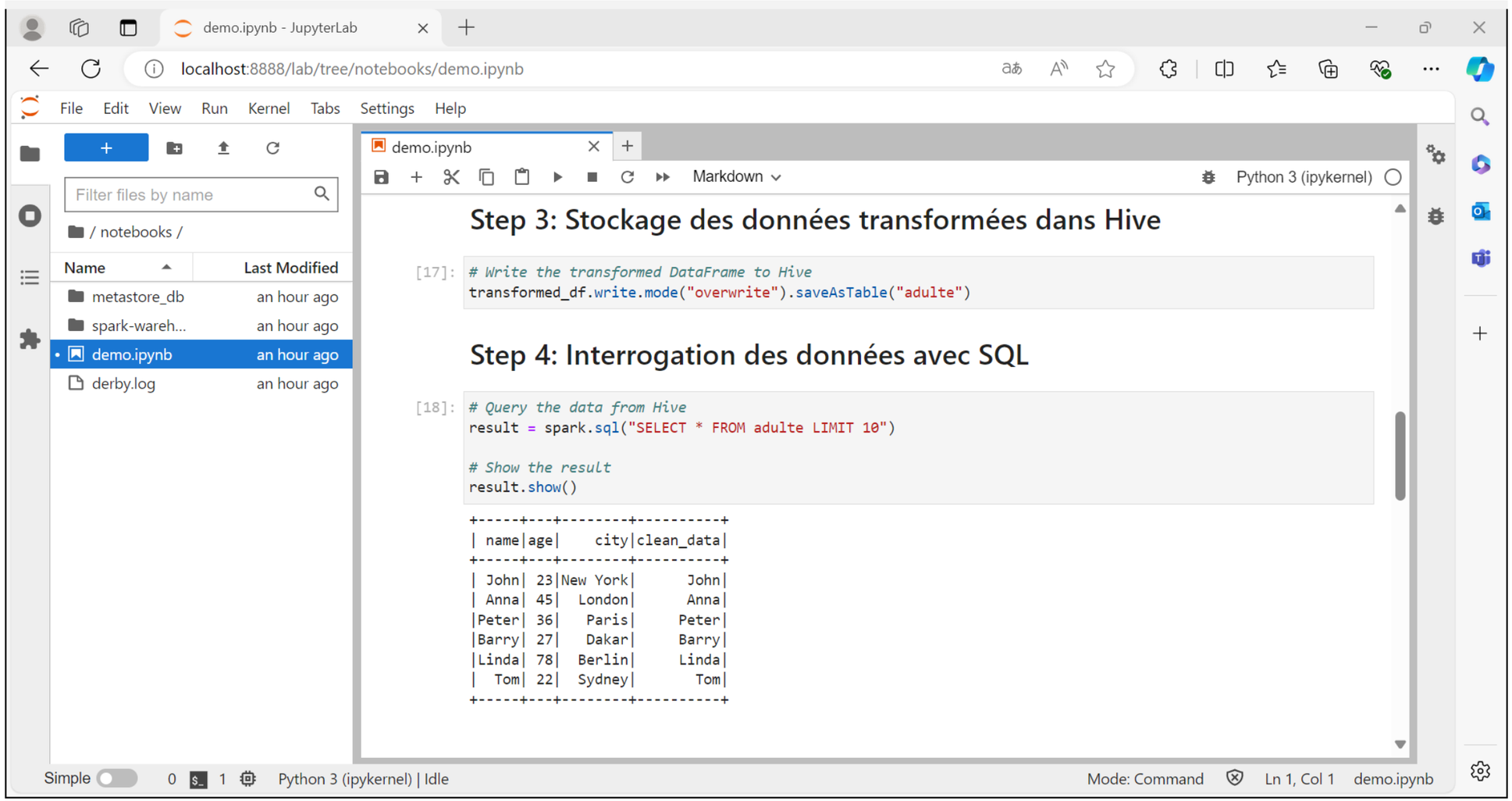
Transformation des données avec Spark





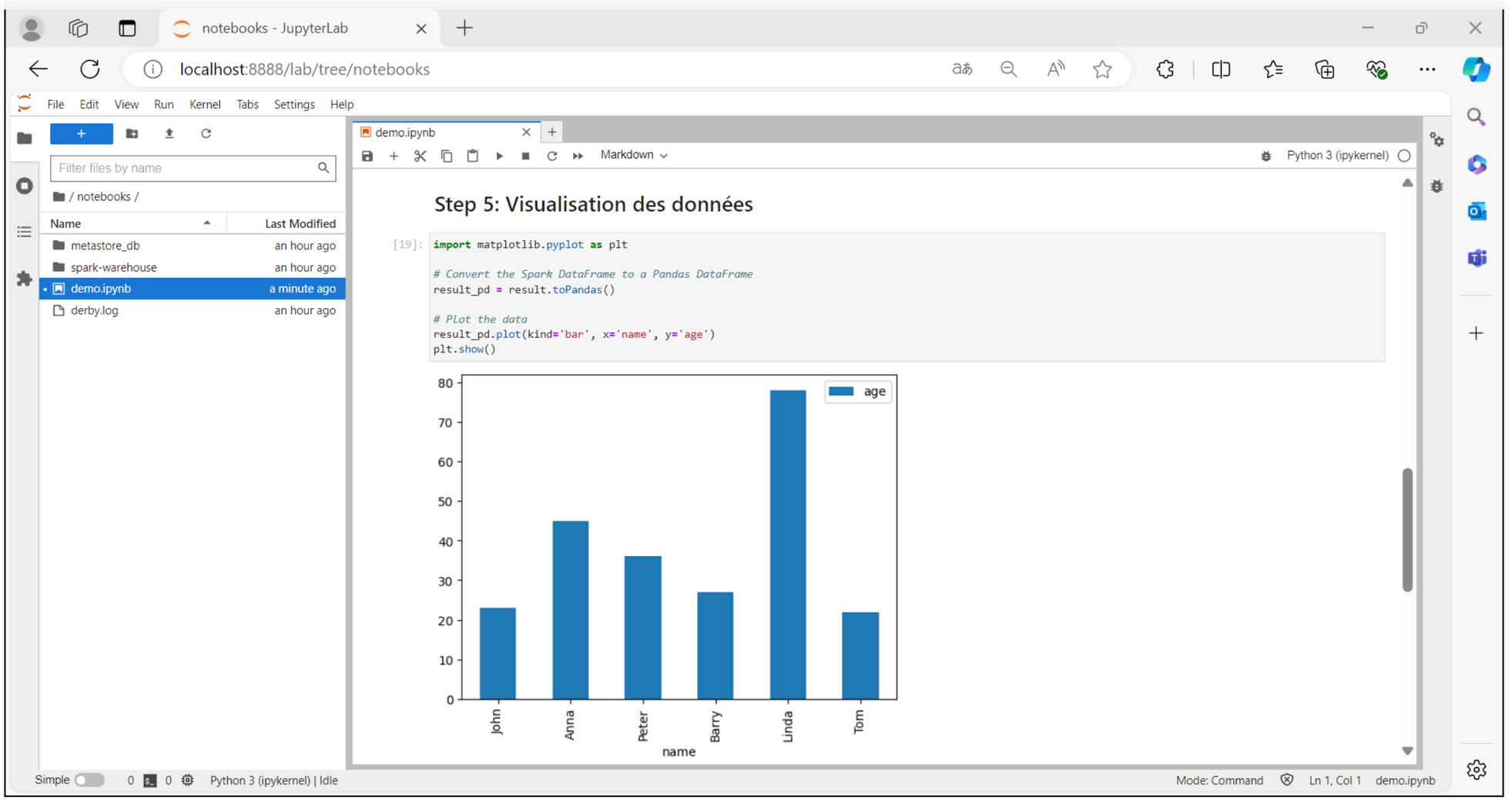
Démo

Stockage des données transformées dans Hive & Interrogation des données avec SQL



Démo

Visualisation des données



Merci de votre attention !

1 Résumé

Apache Spark est une plateforme puissante et flexible pour le traitement de données à grande échelle. Il offre des performances élevées, une large gamme de cas d'utilisation et une communauté active. Sa capacité à gérer des volumes importants de données et à exécuter des calculs de manière efficace en fait un choix populaire pour les entreprises et les organisations qui s'occupent de big data.

2 Tendances Futures

Le domaine du big data continue d'évoluer, avec des tendances émergentes telles que l'intelligence l'intelligence artificielle, l'apprentissage automatique, l'Internet des objets (IoT) et les données en données en temps réel. Spark est bien placé pour s'adapter à ces tendances et continuer à jouer un à jouer un rôle majeur dans l'analyse et le traitement des données à grande échelle.

3 Sources

https://www.edureka.co/blog/spark-sql-tutorial/
https://spark.apache.org/
https://stackoverflow.com/