**BiG Data**

**TP2 - Traitement par Lot et Streaming avec Spark**

Annassiri Fatima Zahra \_ MBisd2

**Introduction : Traitement par Lot et Streaming**

D’après Wikipédia un traitement par lots est un enchaînement automatique d'une suite de commandes sur un ordinateur sans intervention d'un opérateur. Une fois que ce processus est terminé, l'ordinateur traite le lot suivant. Le traitement des lots se termine une fois que tous les lots de la pile ont été exécutés, or un flux (en anglais : Stream) est une suite infinie d'éléments gérés de façon temporel, et peuvent être comparées à des chaînes de traitement (ou pipelines), car elles agissent de manière analogue à la composition des fonctions.

Le tableau présente une Comparaison entre le traitement par lots et le traitement des flux fait par Amazon France :

|  |  |  |
| --- | --- | --- |
|  | Traitement par lots | Traitement des flux |
| Portée des données | Requêtes ou traitement portant sur la totalité ou la majorité des données de l'ensemble de données. | Requêtes ou traitement portant sur les données d'une fenêtre temporelle continue ou sur l'enregistrement de données le plus récent. |
| Taille des données | Lots de données volumineux. | Enregistrements individuels ou micro lots comprenant quelques enregistrements. |
| Performances | Latences de quelques minutes à plusieurs heures. | Nécessite une latence de l'ordre de quelques secondes ou millisecondes. |
| Analyses | Analyses complexes. | Fonctions de réponse simples, regroupements et métriques continues. |

**Comparatif des infrastructures big data temps réel**

Les Framework ci-dessus répondent à deux philosophies différentes**. Flink et Storm** font du traitement au fil de l'eau – soit du temps réel en natif - tandis que Spark procède par micro-batchs pour traiter et produire des résultats toutes les X millisecondes pour s'approcher du temps réel.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Apache Storm | Apache Spark | Apache Flink |
| Année de création | 2011 | 2009 | 2010 |
| Origine | Twitter | Université de Berkeley | Université de Berlin |
| Nature du traitement | Traitement d'événements | Micro-batchs | Traitement en temps réel |
| Langages supportés | Clojure, Java | Java, Python, R, Scala | Java, Python, Scala |
| Distributeurs Hadoop intégrant le Framework | - | Cloudera, Hortonworks et MapR | - |
| Partenariats éditeurs | - | Talend, Dataiku | Talend |
| Editeur commercial | - | Databricks | Data Artisans (racheté par Alibaba) |
| Service managé en mode cloud | Apache Storm sur Azure HDInsight | Apache Spark sur Amazon EMR, Spark sur Azure HDInsight, Cloud Dataproc de Google Cloud (Spark + Hadoop) | Apache Flink sur Amazon EMR |
| Références | Yahoo!, Twitter, Spotify, Groupon... | Uber, ING, Criteo, Zalando... | Alibaba, AWS, CapitalOne, OVH, Bouygues Telecom... |

En termes de popularité Spark est le plus populaire selon les recherches sur le moteur google. Spark a été créé en Scala mais supporte aussi Java, Python et R, ces deux derniers langages étant plébiscités par les statisticiens et data scientistes.

**Objectif du TP :**

Utilisation de Spark pour réaliser des traitements par lot et des traitements en streaming.

**Outils et version :**

On utilise pour ce TP :

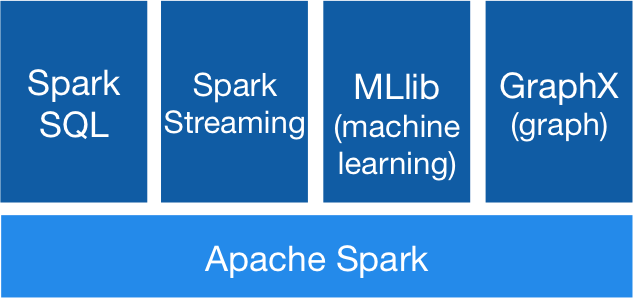
* [Apache Hadoop](http://hadoop.apache.org/) Version: 2.7.2
* [Apache Spark](https://spark.apache.org/) Version: 2.2.1
* [Docker](https://www.docker.com/) Version 17.09.1
* [IntelliJ IDEA](https://www.jetbrains.com/idea/download/) Version Ultimate 2016.1 (ou tout autre IDE de votre choix)
* [Java](http://www.oracle.com/technetwork/java/javase/downloads/index.html) Version 1.8
* Unix-like ou Unix-based Systems (Divers Linux et MacOS)

**Apach Spark :**

Spark est un Framework open source de calcul distribué. Il s'agit d'un ensemble d'outils et de composants logiciels structurés selon une architecture définie. Développé à l'université de Californie à Berkeley par AMPLab, Spark est aujourd'hui un projet de la fondation Apache.



Spark alimente une pile de bibliothèques, notamment SQL et DataFrames, MLlib pour l'apprentissage automatique, GraphX et Spark Streaming.



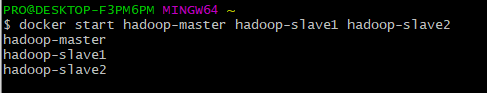
Spark peut s'exécuter sur plusieurs plateformes : Hadoop, Mesos, en standalone ou sur le cloud. Il peut également accéder diverses sources de données, comme HDFS, Cassandra, HBase et S3.



**Installation :**

Comme le téléchargement de l’image docker est faite dans le dernier tp, il suffit donc de lancer la machine via la commande :

docker start hadoop-master hadoop-slave1 hadoop-slave2

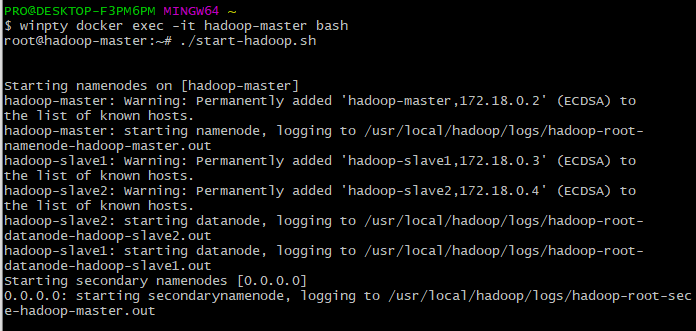


Puis d'entrer dans le centenaire master via la commande :

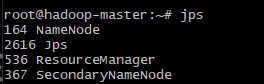
docker exec -it hadoop-master bash

et Lancer ensuite les démons yarn et hdfs via la commande :

./start-hadoop.sh

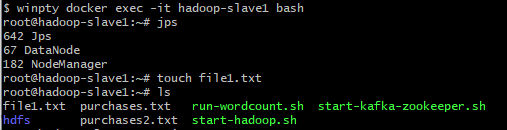


Pour vérifier que tous les démons sont lancer en tape : jsp



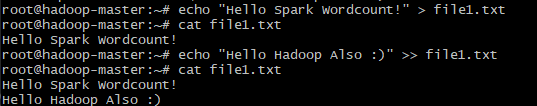
Pour vérifier que tous les nœuds esclaves sont lancer en tape :

docker exec -it hadoop-slave1 bash



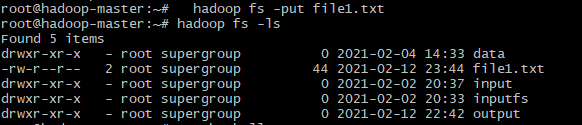
**Test de Spark avec Spark-Shell**

Création de fichier file1.txt dans le Node Master :



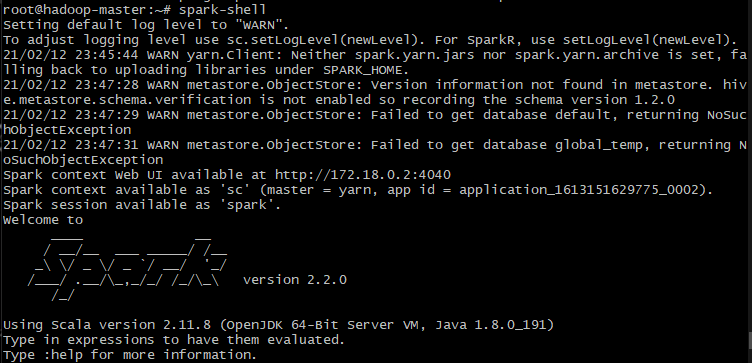
Charger ensuite ce fichier dans HDFS en utilisant la commande :

hadoop fs -put file1.txt



Vérification de l’installation du Spark, par la commande suivante :

spark-shell



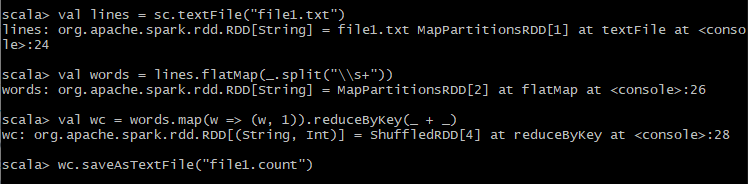
Tester Spark avec un code scala :

val lines = sc.textFile("file1.txt")

val words = lines.flatMap(\_.split("\\s+"))

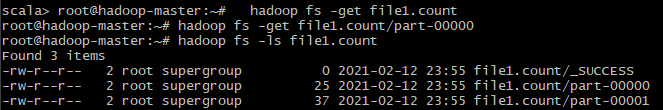
val wc = words.map(w => (w, 1)).reduceByKey(\_ + \_)

wc.saveAsTextFile("file1.count")

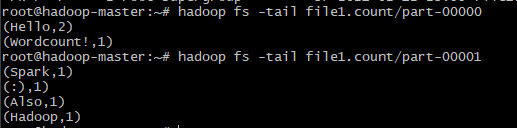


Pour afficher le résultat, en charge le répertoire *file1.count* créé dans HDFS avec la commande :

hadoop fs -get file1.count

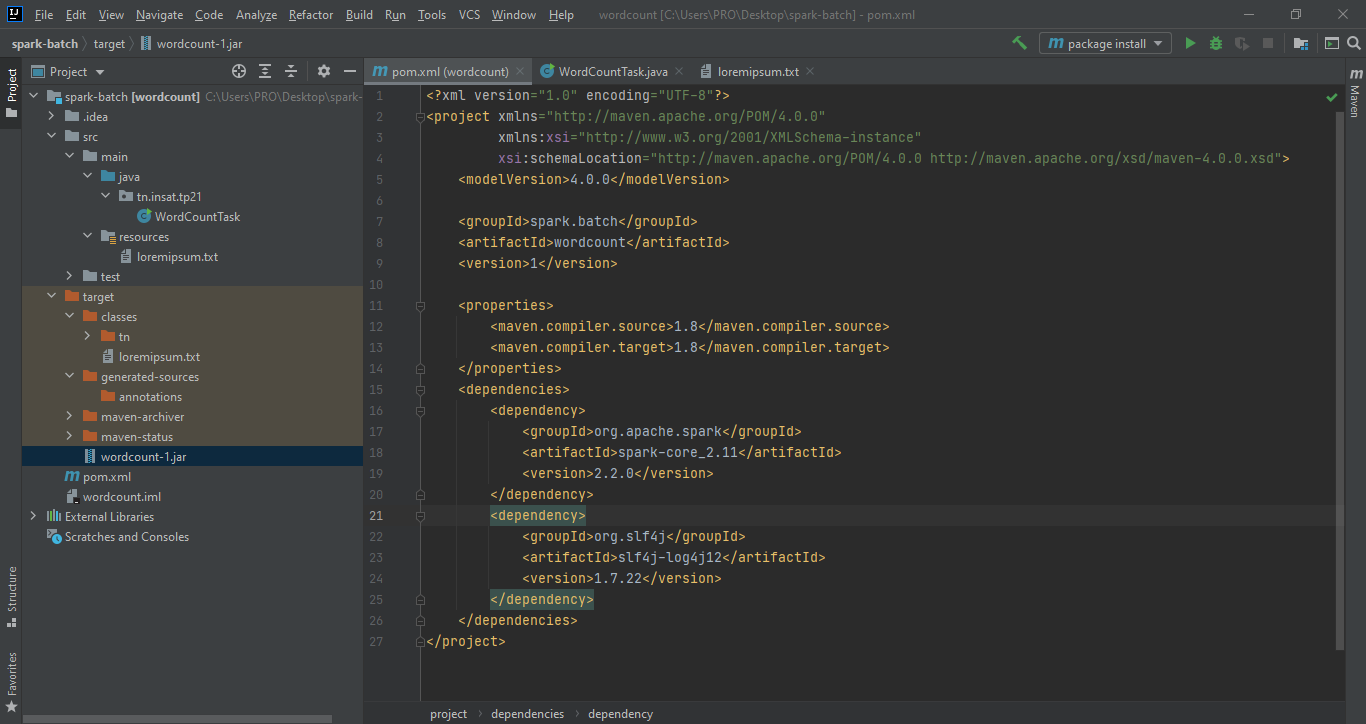


Le contenu donner par :

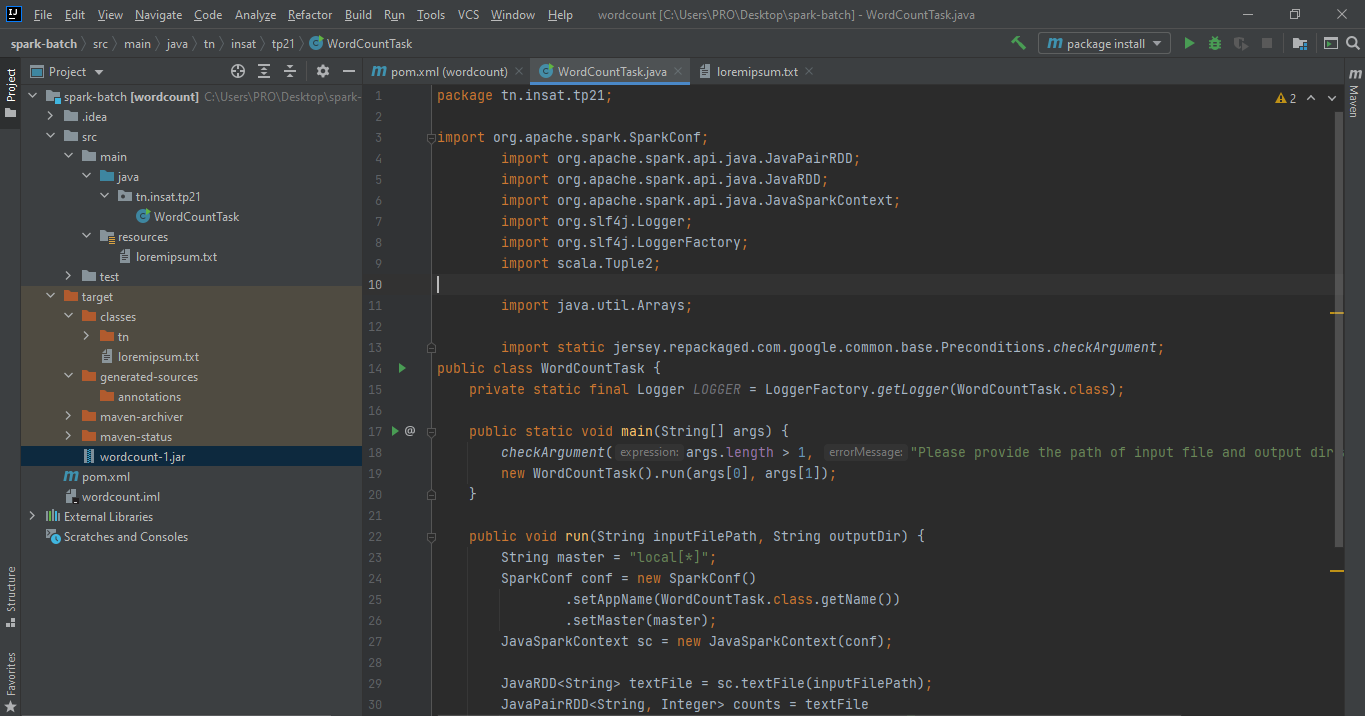


**Spark Batch en Java**

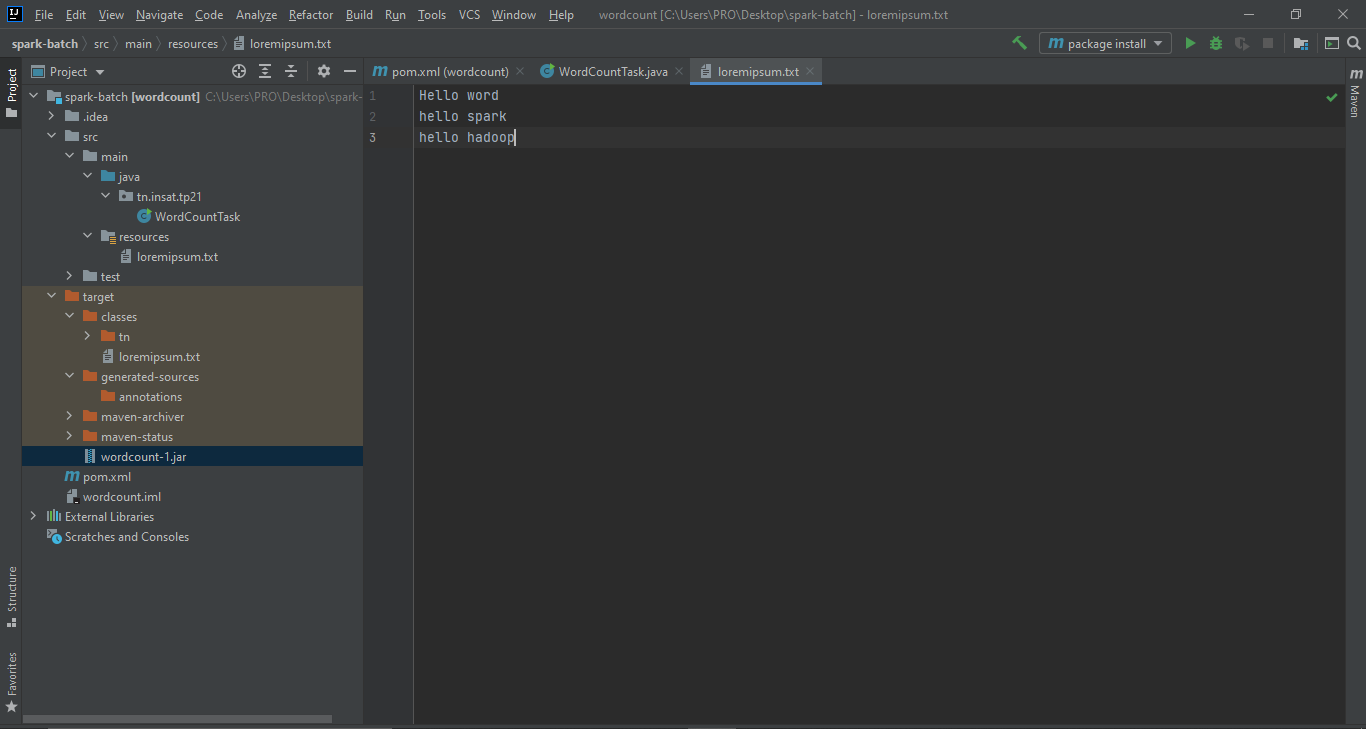
Dans cette étape on ‘a préparer l’environnement et le code ; commençons par la création du  projet Maven avec IntelliJ IDEA et le configurer ,



Création de package *tn.insat.tp21*, et dedans, la classe *WordCountTask*.



Le fameux Lorem ipsum



ensuite en lance l’installation des package pour générer le fichier  *worcount-1.jar* , qui sera créé sous le répertoire Target comme illustré dans la figure précédente .

pour copier le fichier jar dans docker en lance la commande sur terminal :

docker cp target/wordcount-1.jar hadoop-master:/root/wordcount-1.jar

Revenir à votre contenaire master, et lancer un job Spark en utilisant ce fichier jar généré, avec la commande spark-submit, un script utilisé pour lancer des applications spark sur un cluster.

spark-submit --class tn.insat.tp21.WordCountTask

--master local

--driver-memory 4g --executor-memory 2g --executor-cores 1

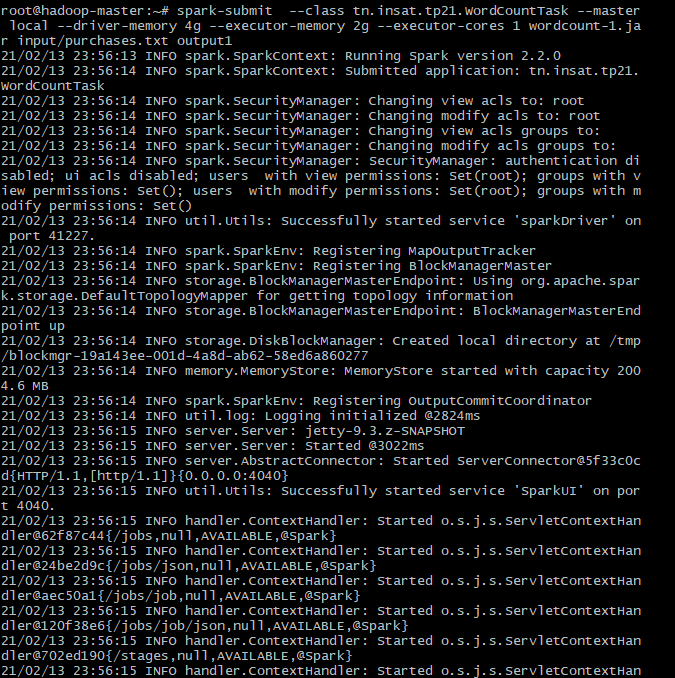
wordcount-1.jar

input/purchases.txt

output

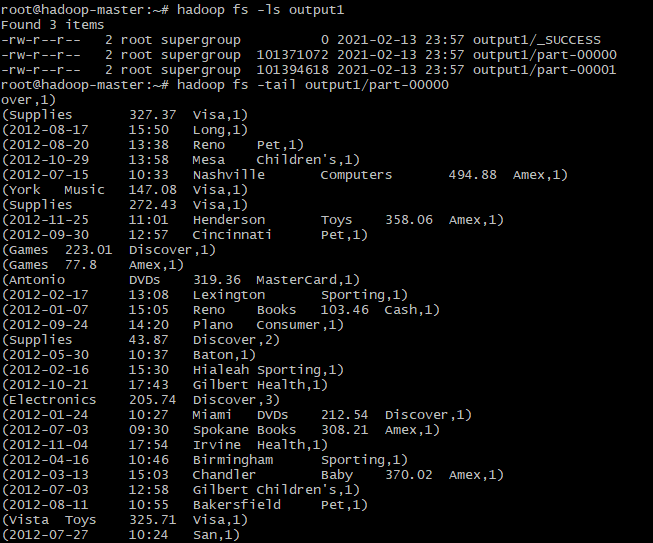
Nous allons lancer le job en mode local, pour commencer.

.



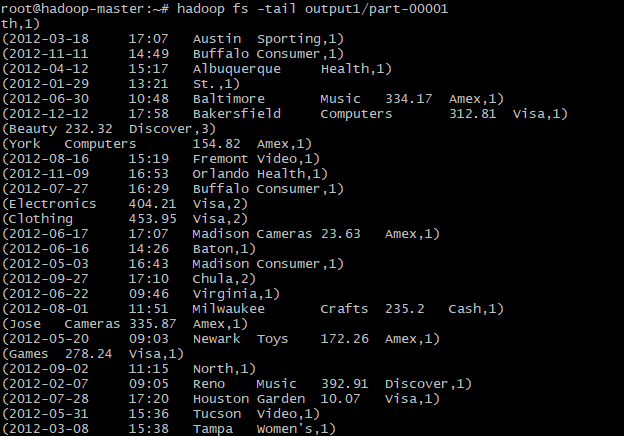
La commande :

Hadoop fs – ls output1



La commande :

Hadoop fs -tail output1/part-00001



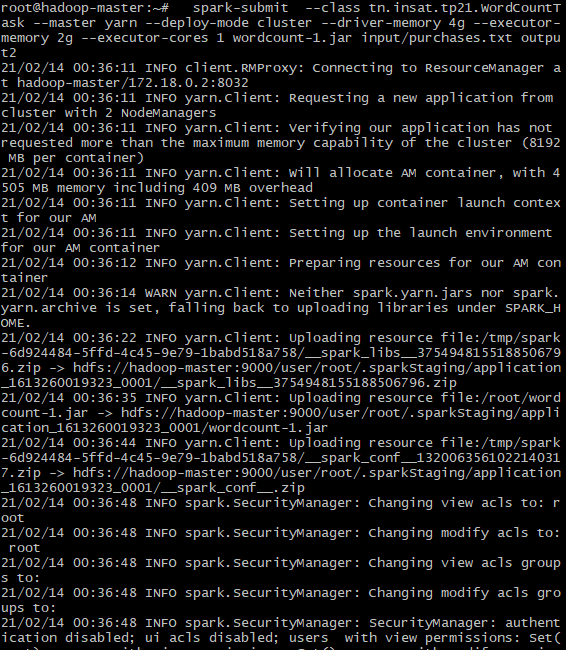
**Lancement du code sur le cluster**

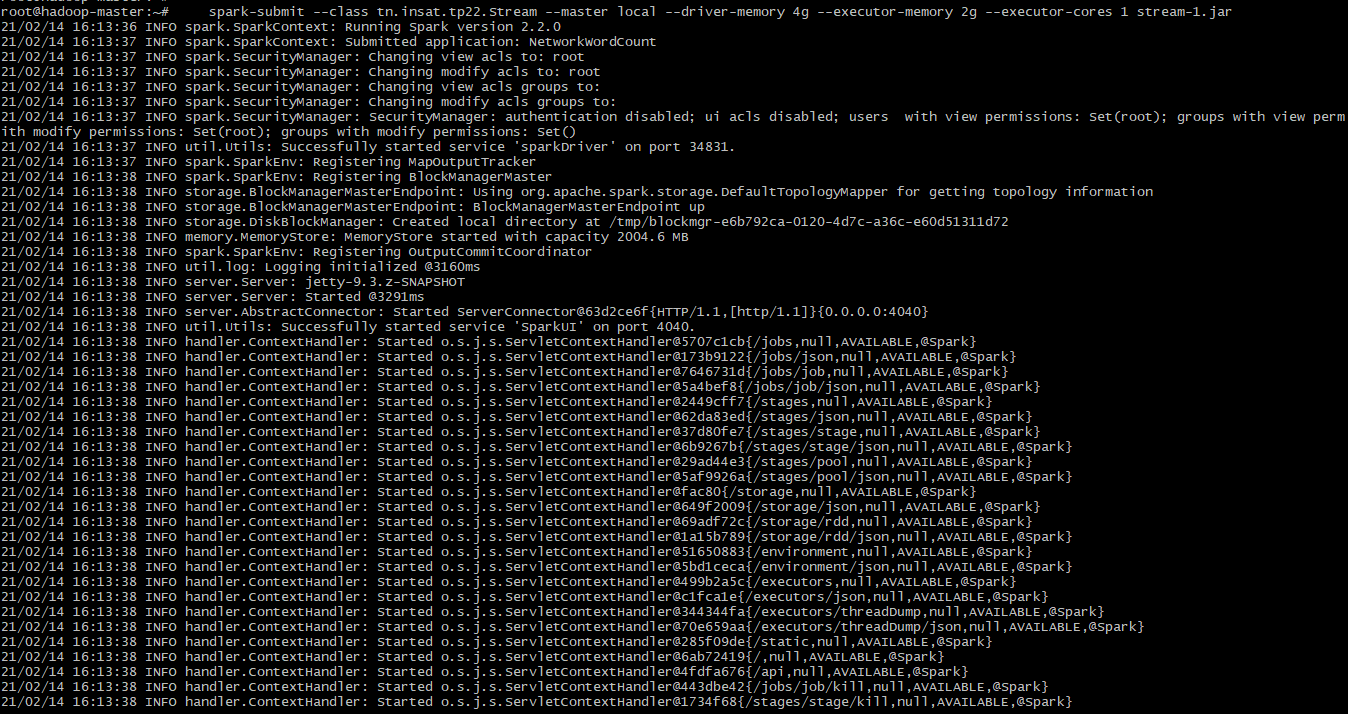
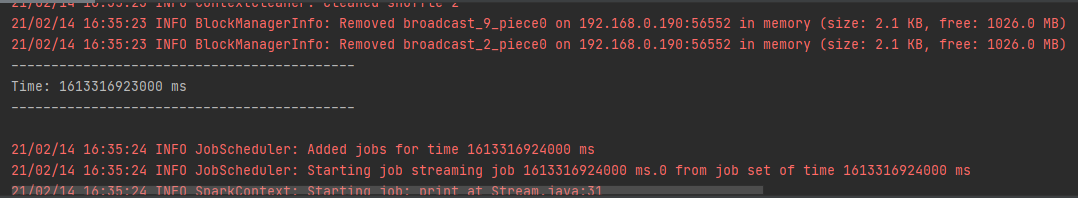
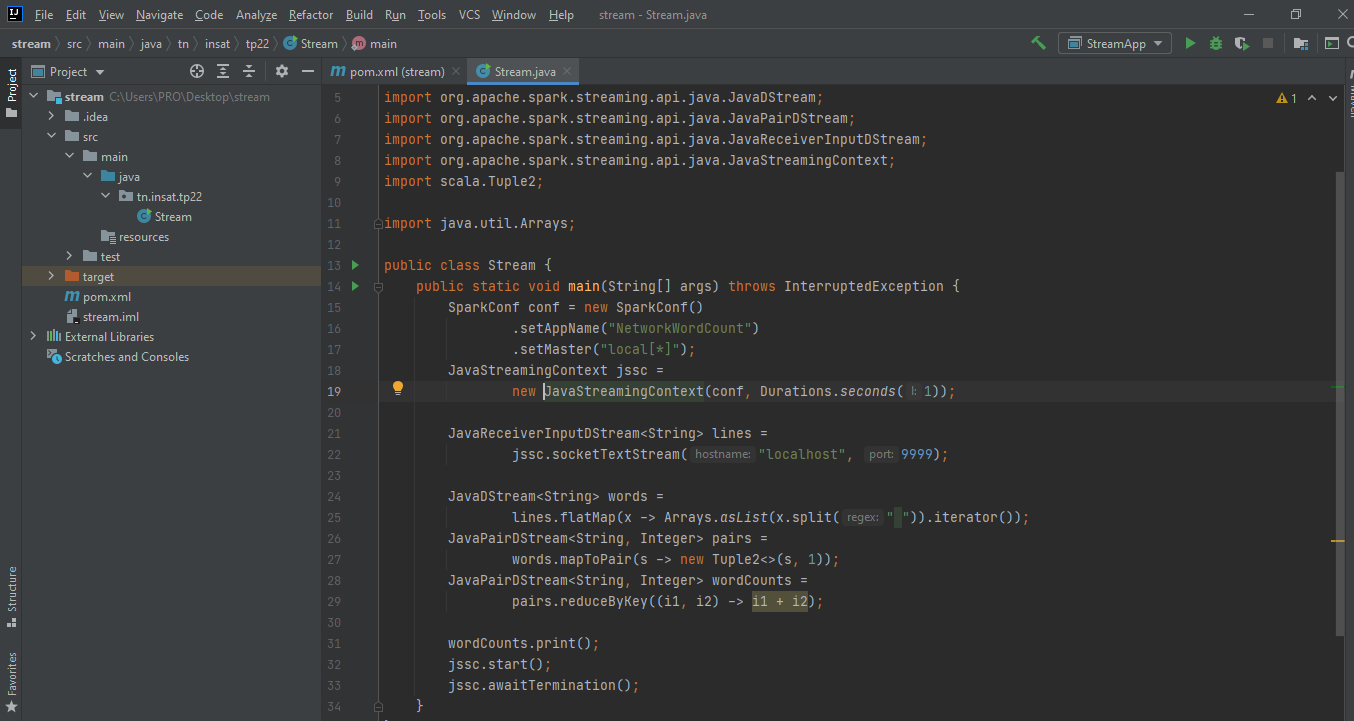
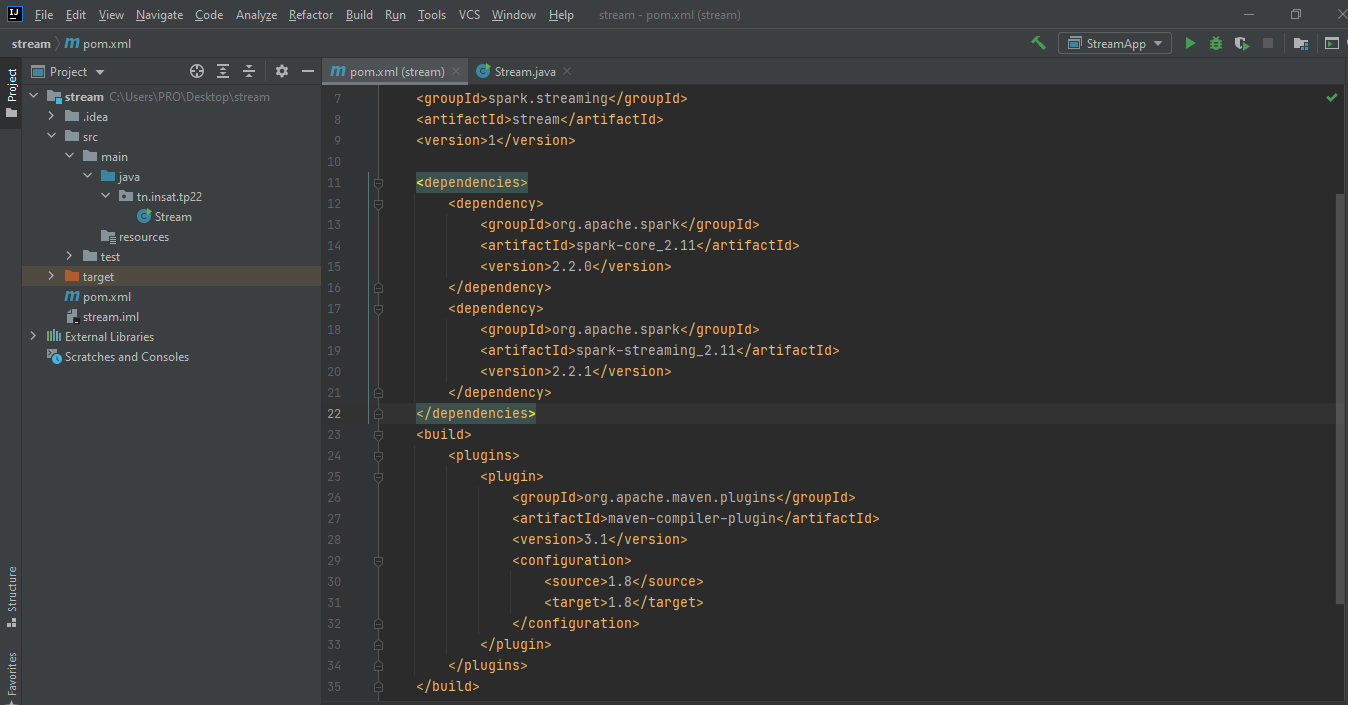
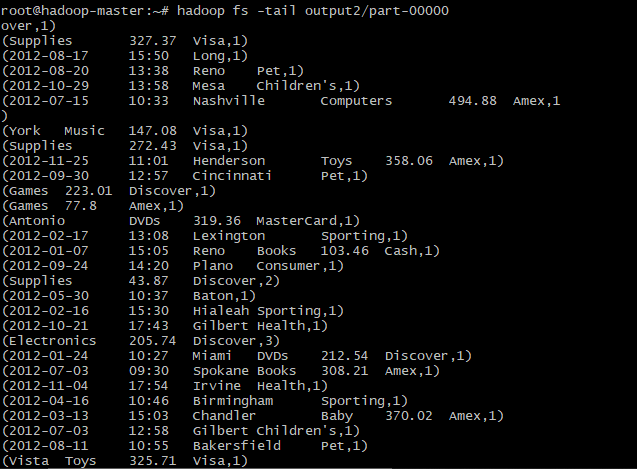
spark-submit --class tn.insat.tp22.Stream

--master local

--driver-memory 4g --executor-memory 2g --executor-cores 1

stream-1.jar





* The end