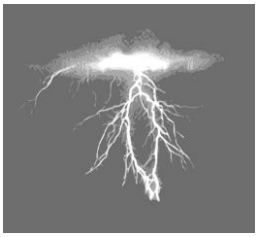

Apache STORM

avec des fichiers

Version 1.1

Rédacteurs V1 : Philippe Lacomme (placomme@isima.fr), Raksmey Phan (phan@isima.fr)

Date : 3 septembre 2015

Rédacteurs V2 : Philippe Lacomme (placomme@isima.fr), Raksmey Phan (phan@isima.fr), étudiants en projets ISIMA (Soriano Baptiste et Zouggari Yannis)

Date : 14 juin 2016

Installation réalisée sur : Ubuntu 15.04

Environnement : Vmware

Documents à consulter :

https://storm.apache.org/releases/0.10.0/index.html

Licence :

Objectifs :

1 objectif dans ce tutoriel :

- apprendre à manipuler des fichiers en entrée

1. Modifier le Spout : Open()

Dans le fichier SentenceSpout.java, dans la classe, il faut ajouter deux variables l'une pour le contexte et l'autre pour lire le contenu du fichier :

```
// les outils de lecture du fichier
private TopologyContext context;
private FileReader fileReader;
```

Il faut ensuite modifier le fichier open pour :

1) récupérer le nom du fichier dans la Map en cherchant inputFile ;

2) ouvrir le fichier et stocker le tout dans la variable fileReader.

```
@Override
public void open(Map map, TopologyContext tc, SpoutOutputCollector soc) {
  try {
        this.context = tc;
        String nom = map.get("inputFile").toString();
        this.fileReader = new FileReader(nom);
        } catch (FileNotFoundException e) {
             throw new RuntimeException("Error reading file "
                                        + map.get("inputFile"));
      }
        this.flux_sortie = soc;
    }
```

La classe ressemble alors à celle de la figure 1.

```
    // les outils de lecture du fichier
    private TopologyContext context;
    private FileReader fileReader;


    @Override
    public void declareOutputFields(OutputFieldsDeclarer ofd) {

        Fields champ = new Fields("sentence");
        ofd.declare(champ);

    }

    @Override
    public void open(Map map, TopologyContext tc, SpoutOutputCollector soc) {

        try {
            this.context = tc;
            String nom = map.get("inputFile").toString();
            this.fileReader = new FileReader(nom);
            } catch (FileNotFoundException e) {
                throw new RuntimeException("Error reading file "
                                           + map.get("inputFile"));
                }


        this.flux_sortie = soc;
    }
```

Figure 1. Classe SentenceSpout après modification de la méthode Open()

2. Modifier le Spout : nextTuple()

Les modifications sont simples et consistent à remplacer la lecture des phrases du tableau par un parcours du fichier. Notons toutefois qu'il faut utiliser un **try…catch()** ce qui alourdit un peu la procédure.

```
@Override
public void nextTuple() {

if (termine==0)
{
 String str;
 BufferedReader reader = new BufferedReader(fileReader);
 try {
     while ((str = reader.readLine()) != null) {
```

```
        Values valeur = new Values(str);

        Calendar cal = Calendar.getInstance();
        int hour = cal.get(Calendar.HOUR_OF_DAY);
        int minute = cal.get(Calendar.MINUTE);
        int secondes= cal.get(Calendar.SECOND);
        int msecondes= cal.get(Calendar.MILLISECOND);
        String heure = hour+":"+minute+":"+secondes+":"+msecondes;

        System.out.println(""+heure+" : Spout : emission de "+str);

        this.flux_sortie.emit(valeur);

      }
     } catch (Exception e) {
              throw new RuntimeException("Error reading typle", e);
      } finally {
         termine = 1;
        }
 }
Utils.waitForMillis(1);
}
```

3. Modifier le Spout : Créer la procedure close()

La procédure soit simplement fermer le fichier.

```
@Override
public void close() {
  try {
        fileReader.close();
      } catch (Exception e) {
          e.printStackTrace();
        }
}
```

4. Modifier le Spout : modification de la classe DemoStorm

Il faut ajouter une variable globale dans la classe :

```
private static final String nom_fichier = "poeme.txt";
```

Il faut modifier le contenu de la variable configuration à laquelle on ajouter le couple **("inputFile", "poeme.txt")**.

```
Config configuration = new Config();
configuration.put("inputFile", nom_fichier);
configuration.setDebug(true);
```

La classe ressemble à celle de la figure 2.

```
public class DemoStorm {

    private static final String SPOUT_ID = "Spout-generateur-de-phrases";
    private static final String BOLT_DECOUPE_ID = "bolt-decoupage-en-mots";
    private static final String BOLT_COMPTEUR_ID = "bolt-compteur-de-mots";
    private static final String BOLT_AFFICHAGE_ID = "bolt-affichage";
    private static final String TOPOLOGY_NOM = "Topology-compter-les-mots";

    private static final String nom_fichier = "poeme.txt";

    public static void main(String[] args) throws Exception {

        SentenceSpout spout_genere_phrases = new SentenceSpout();
        SplitSentenceBolt bolt_decoupe = new SplitSentenceBolt();
        WordCountBolt bolt_compte = new WordCountBolt();
        ReportBolt bolt_affiche = new ReportBolt();
        TopologyBuilder topologie = new TopologyBuilder();

        topologie.setSpout(SPOUT_ID, spout_genere_phrases);
        topologie.setBolt(BOLT_DECOUPE_ID, bolt_decoupe,2).shuffleGrouping(SPOUT_ID);
        topologie.setBolt(BOLT_COMPTEUR_ID, bolt_compte).fieldsGrouping(BOLT_DECOUPE_I
        topologie.setBolt(BOLT_AFFICHAGE_ID, bolt_affiche).globalGrouping(BOLT_COMPTEU

        Config configuration = new Config();
        configuration.put("inputFile", nom_fichier);
        configuration.setDebug(true);

        if (args != null && args.length > 0) {
```

Figure 2. La fonction main() après modification

5. Tester le programme

Il faut copier le poème dans un fichier nommé par exemple poeme.txt (figure 3).

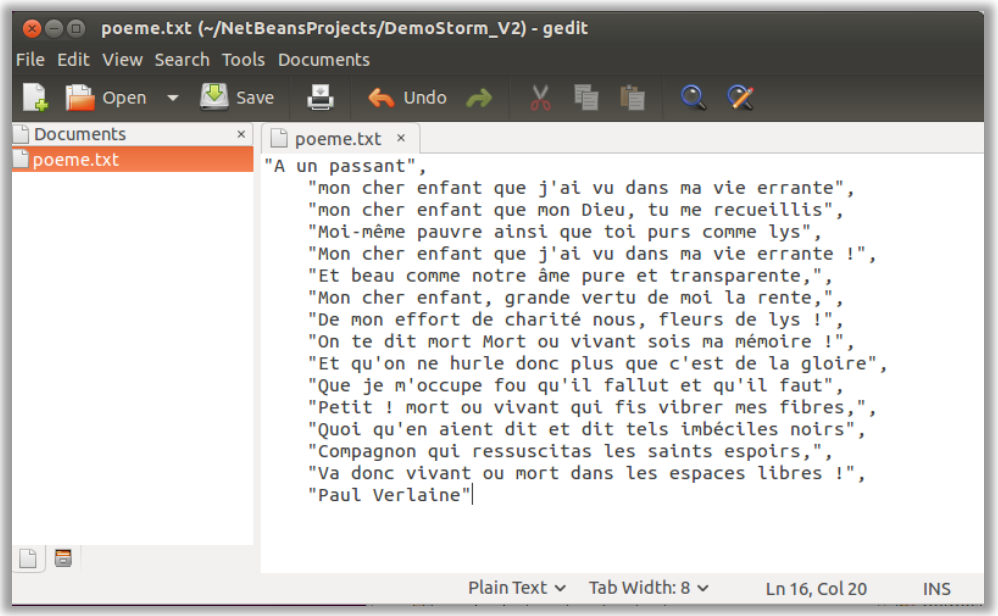

Figure 3. Création d'un fichier texte contenant le poème

On peut ensuite lancer le programme Java, ce qui donne le résultat de la figure 4.

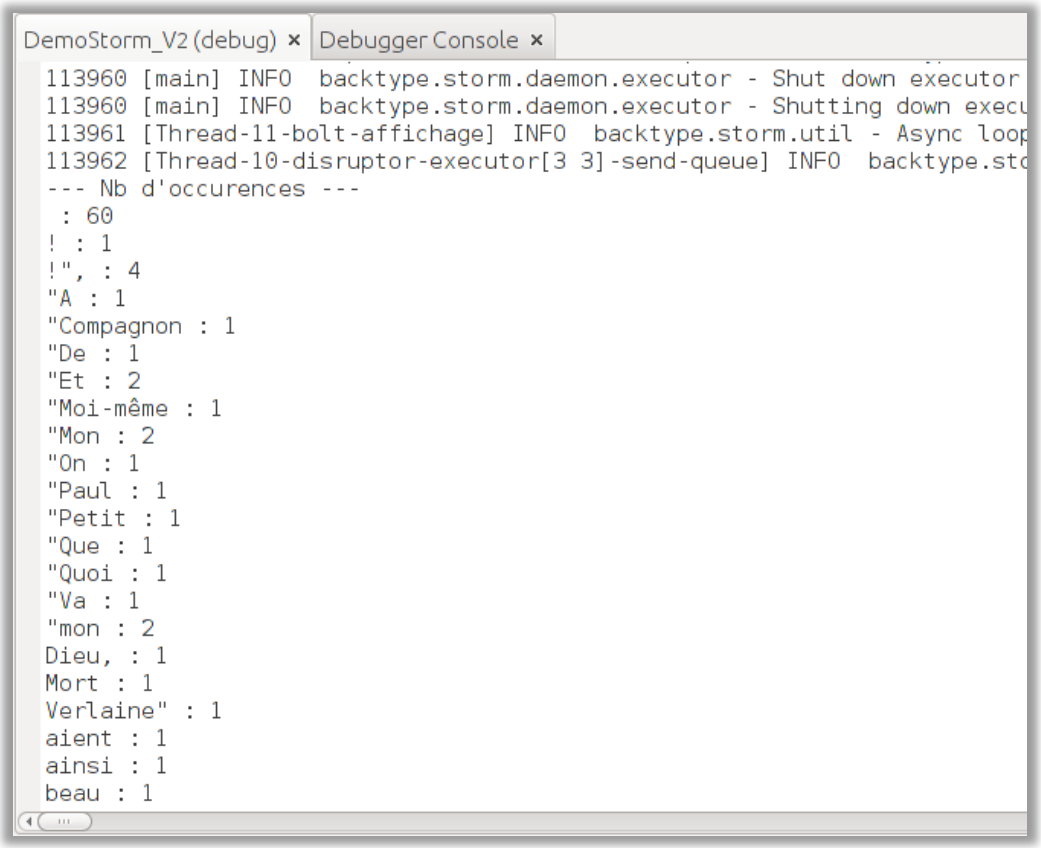

Figure 4. Résultat de l'exécution

6. Cas particulier des fichiers pour une exécution dans une cluster Storm

Il **n'est pas possible** d'exécuter le code précédent sur un cluster Storm en dehors de l'environnement NetBeans. En effet, le fichier poeme.txt est présent sur le disque dur de la machine de développement mais n'existera pas sur les nœuds du cluster après un démarrage en ligne de commande.

Deux règles sont à respecter :

1) les fichiers doivent être inclus dans le fichier Jar 2) il faut accéder aux fichiers avec des flux.

Il est possible de créer un répertoire et de regrouper l'ensemble des fichiers de données dans ce répertoire. On peut par exemple créer un répertoire **data** comme sur la figure 5 ou bien directement dans le répertoire contenant les classes java. Inutile de préciser que cette deuxième solution est à éviter… !

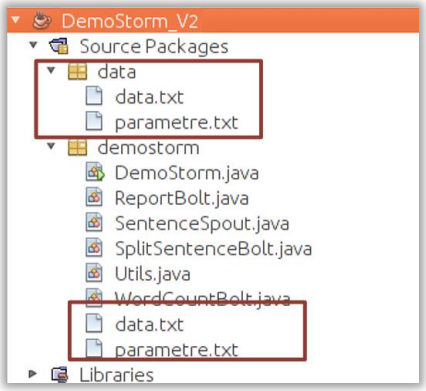

Figure 5. Inclusion de fichiers (ressources) dans le projet

Modification de la classe SentenceSpout.java

Il faut définir une variable de type InputStream nommée par exemple MonInputStream et accompagner cette définition de deux autres variables, l'une qui contiendra le nombre total de ligne à lire et la deuxième le nombre de ligne déjà lues.

```
    // -- variables pour la manip du flux --
    // -- ------------------------------ --
    InputStream MonInputStream;
    int nb_ligne_totale=0;
    int nb_ligne_lue = 0;
```

Il faut procéder comme pour un fichier classique et stocker dans la variable MonInputStream l'accès à la ressource (évitons le mot fichier!). Ceci passe par la manipulation d'une URL et ensuite une utilisation de openStream() qui donnera finalement le stream (enfin peut-on dire).

```
// identification des datas
URL urlClass=DemoStorm.class.getResource("data.txt");
System.out.println("url = "+urlClass.toString());
InputStream ss = urlClass.openStream();
MonInputStream = ss;
```

Le caractère peut agréable des manipulations arrive maintenant hélas… car le stream permet une récupération caractères par caractères des données…..
De ce point de vue, il faut mieux formater les fichiers avec des informations sur le nombre de ligne à lire et un séparateur spécifique pour les fins de ligne. La figure 6 donne un exemple d'un tel fichier, qui peut être comparé à celui de la figure 7.

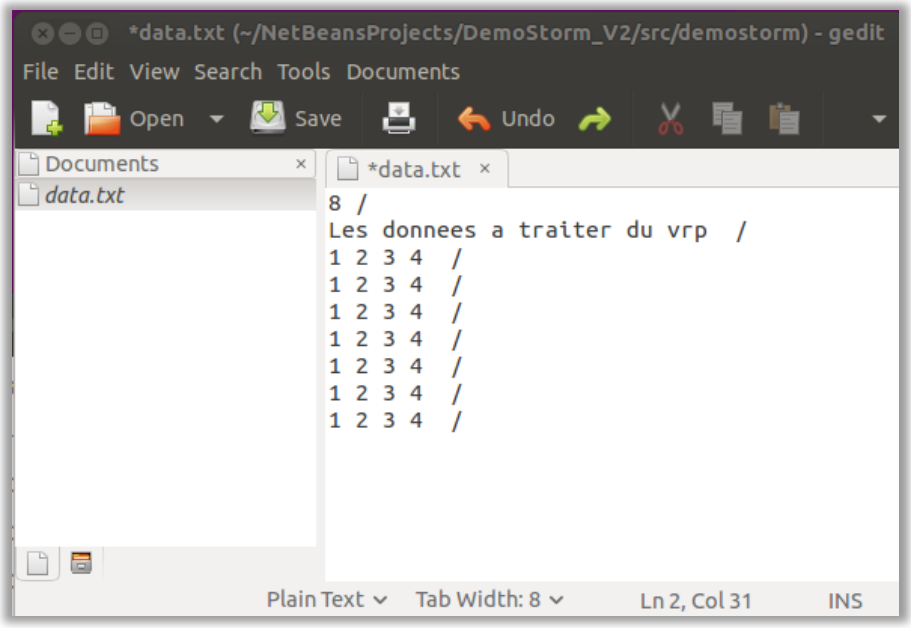

Figure 6. Un fichier formaté… facilitant la relecture

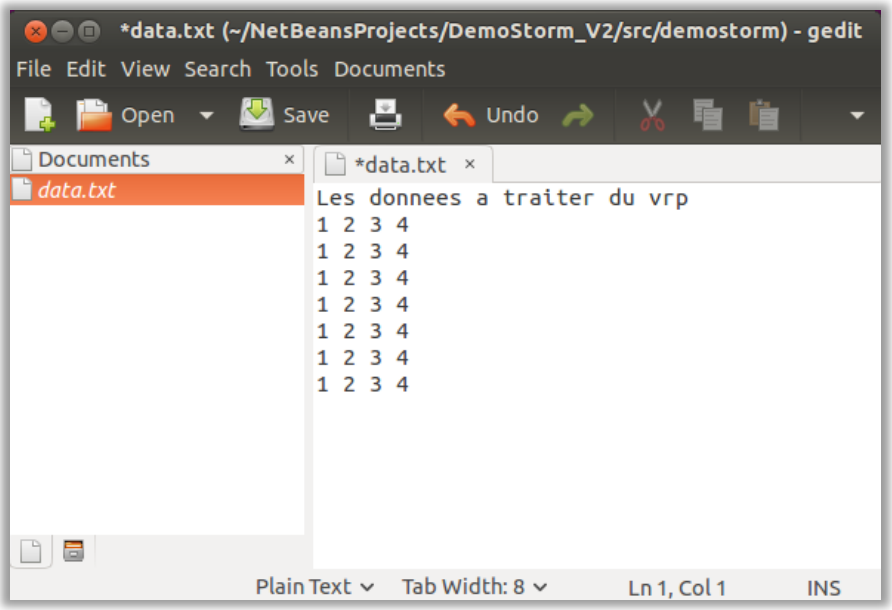

Figure 7. Un fichier formaté… ne facilitant pas la relecture

Deux autres à considérer peut être celui des figures 8 et 9.

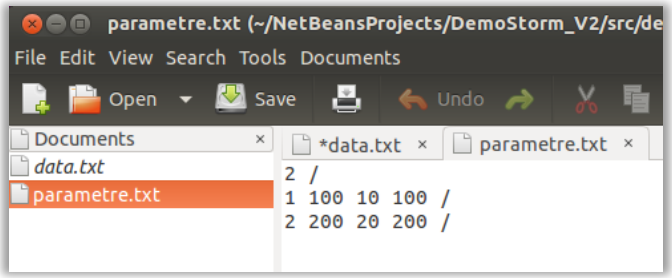

Figure 8. Un fichier formaté… ne facilitant pas la relecture

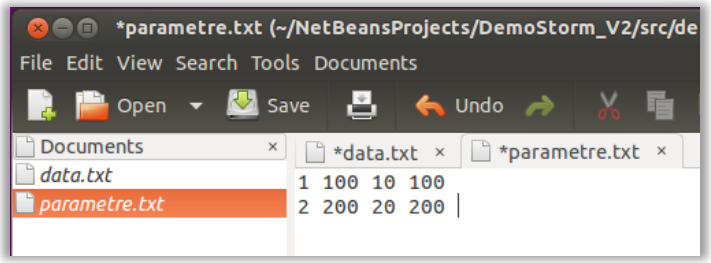

Figure 9. Un fichier formaté… ne facilitant pas la relecture

Le fichier du poème peut ainsi être reformaté comme montrer à la figure 10.

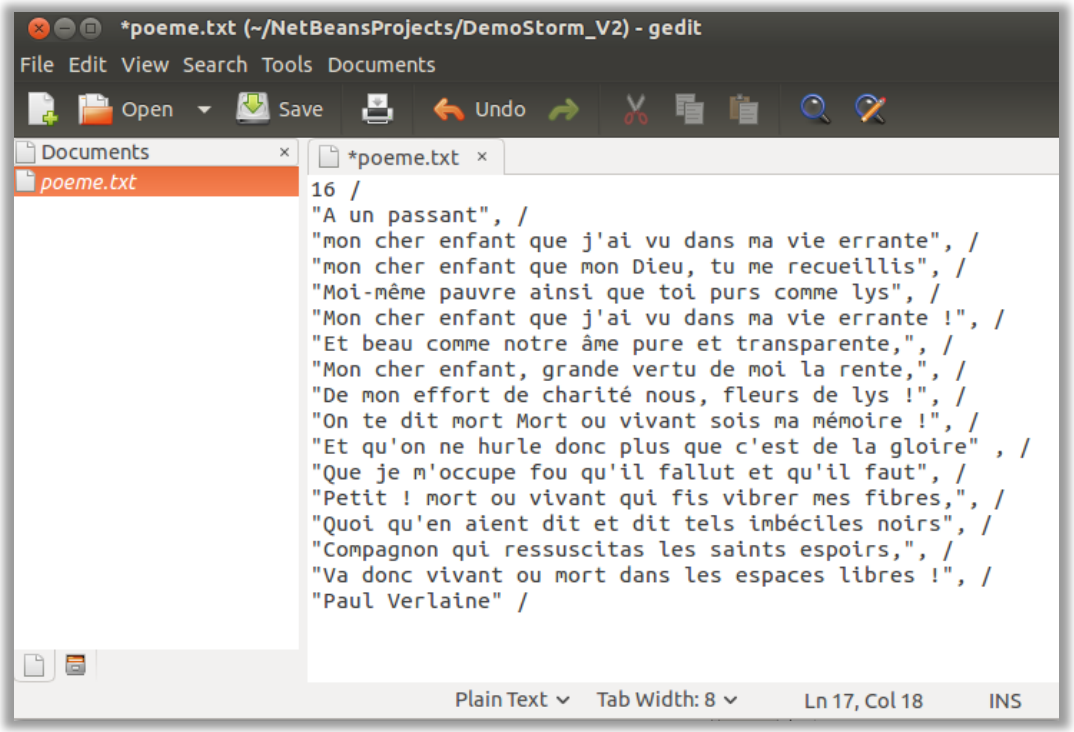

Figure 10. Le poème reformaté….

Compte tenu des remarques qui précèdent, on peut détailler le début de solution pour la classe SentenceSpout.java.

On peut munir la classe d'une procédure permettant de lire le contenu de la première ligne du fichier et dont le code serait :

```
   void consulter_nb_ligne_dans_fichier()
    {
       try{
        char c;
        String ligne="";
        do
        {
          int nb = MonInputStream.read();
          c = (char) nb;
          ligne=ligne+c;
        }while (c!='/');
        String[] parts = ligne.split("/");
        String chaine1=parts[0].trim();
        nb_ligne_totale = Integer.parseInt(chaine1);
        System.out.print(ligne);
       }catch(Exception E)
          {}
    }
```

Toujours selon le même principe, on peut créer une procédure lire_ligne() dont le code serait :

```
 String lire_ligne()
    {
      if (nb_ligne_lue<nb_ligne_totale)
      {
       try{
        char c;
```

```
        String ligne="";
        do
        {
          int nb = MonInputStream.read();
          c = (char) nb;
          ligne=ligne+c;
        }while (c!='/');
        String[] parts = ligne.split("/");
        String chaine1=parts[0].trim();
        System.out.print(chaine1);
        nb_ligne_lue++;
        return chaine1;
       }catch(Exception E)
          {}
      }
      return "";
    }
```

La procédure nextTuple() peut alors se réécrire facilement comme suit :

```
@Override
  public void nextTuple() {
    if (termine==0)
    {
     consulter_nb_ligne_dans_fichier();
     String str;
     try {
         while (nb_ligne_lue<nb_ligne_totale)
           {
           str =  lire_ligne();
          Values valeur = new Values(str);

            Calendar cal = Calendar.getInstance();
            int hour = cal.get(Calendar.HOUR_OF_DAY);
            int minute = cal.get(Calendar.MINUTE);
            int secondes= cal.get(Calendar.SECOND);
            int msecondes= cal.get(Calendar.MILLISECOND);
            String heure = hour+":"+minute+":"+secondes+":"+msecondes;

            System.out.println(""+heure+" : Spout : emission de "+str);

            numero_courant++;
            this.flux_sortie.emit(valeur, numero_courant);
            Utils.waitForMillis(3);
           }
          } catch (Exception e) {
             throw new RuntimeException("Error reading typle", e);
          } finally {
             termine = 1;
              }
        }
      Utils.waitForMillis(1);
    }
```

Le lecteur pourra facilement vérifier qu'en déployant le .jar sur le cluster, l'ensemble du code fonctionne alors normalement….
