http://webdam.inria.fr/

# Web Data Management

## Mise en pratique : HADOOP

Serge Abiteboul
INRIA Saclay & ENS Cachan

Ioana Manolescu
INRIA Saclay & Paris-Sud University

Philippe Rigaux
CNAM Paris & INRIA Saclay

Marie-Christine Rousset
Grenoble University

Pierre Senellart
Télécom ParisTech



http://webdam.inria.fr/Jorge/

# Contents



Ce chapitre est une introduction à HADOOP et suggère plusieurs exercices pour expérimenter ce système en pratique. Il suppose que vous disposez d'un sytème de type Unix (Mac OS X également ; Windows nécessite l'utilisation de Cygwin). HADOOP peut fonctionner dans un mode pseudo-distribué qui ne nécessite pas d'infrastructure de type *cluster* pour tester le logiciel, et la plupart des instructions qui vont suivre utilisent ce mode. Passer à un véritable cluster nécessite d'autres réglages qui sont présentés à la fin du chapitre. Comme HADOOP est un système relativement réçent et en constante évolution, consulter la documentation en ligne à jour est bien sûr recommandé pour un usage réel. Nous illustrons les usages d'HADOOP, MAPREDUCE et PIG sur des extraits du jeu de données DBLP (DBLP est une vaste base de données de reférences bibliographiques, que vous pouvez fouiller en ligne sur `http://dblp.uni-trier.de` (cherchez-y les publications de vos enseignants !).

## 1 Installation et démarrage d'HADOOP

1. Récupérer l'archive HADOOP sur le *share* et décompressez-là dans votre dossier que nous appelerons dans la suite «dossier d'installation »(pour d'autres version, le site officiel est `http://hadoop.apache.org/`).

2. Afin de faire une fois pour toutes les configurations, modifiez votre fichier d'initialisation du shell (`~/.bashrc`) avec les commandes figurant dans le fichier *configuration*.

3. Remplacez le fichier `etc/hadoop/hadoop-env.sh` par celui disponible sur le *share*.

4. Votre `hadoop` devrait maintenant fonctionner :

   ```
   hadoop version
   ```

5. Nous allons commencer avec des manipulations simples du système de fichiers distribué géré par HADOOP, dans un mode dit «pseudo-distribué », qui lance les serveurs HADOOP sur votre machine locale (donc, sans distribution ! Mais cela facilite les tests). Pour utiliser ce mode, vous devez éditer le fichier de configuration *etc/hadoop/core-site.xml* et ajouter les paramètres suivants :

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Ceci indique à HADOOP que son environnement est constitué d'un noeud maître local HDFS (appelé le «NameNode »dans la terminologie HADOOP) sur le port 9000. HDFS signifie HADOOP *distributed file system*.

6. Initialisez HDFS avec la commande suivante :

```
$ hadoop namenode -format
```

Ceci réserve un dossier dans */tmp/hadoop-<username>/dfs/name*.

L'usage général de la commande *hadoop* est le suivant :

```
$ hadoop <command> [paramètres]
```

où `command` est soit `namenode` (commande envoyée au noeud maître), `fs` (commande au système de fichiers), `job` (commande aux jobs MAPREDUCE), etc.

Une fois le système de fichier formaté, vous devez lancer le noeud maître HDFS (*namenode*). Le noeud maître est responsable de la gestion des noeuds serveurs de fichiers du cluster (appelés *datanodes* dans HADOOP), et cela via la commande *ssh*. Vous devez vérifier que SSH est bien configuré et, en particulier, que vous pouvez vous connecter à la machine locale avec SSH *sans* avoir besoin de votre mot de passe. Pour cela, essayez la commande suivante :

```
$ ssh localhost
```

Si cela ne marche pas (si un mot de passe est demandé), vous devez exécuter ceci :

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

Si le problème persiste, consultez la FAQ à la fin de ce document.

Ces commandes engendrent une nouvelle clé SSH sans mot de passe de contrôle, et l'ajoute aux connexions auxquelles vous faites confiance.

Le démarrage du noeud maître devrait être maintenant possible :

```
$ start-dfs.sh
```

Ceci exécute un processus namenode et un processus datanode sur la machine locale. Vous devriez voir apparaître :

```
Starting namenodes on ...
localhost: starting datanode, logging to (...)
localhost: starting secondary namenode, logging to (...)
Starting secondary namenodes on (...)
(...)
```

Le namenode secondaire est un miroir du principal, utilisé en cas de panne de ce dernier. Vous pouvez voir les processus Java correspondants en tapant

```
$ jps
```

7. Maintenant vérifier que HDFS est fonctionnel avec la commande

```
$ hadoop fs -ls /
```

Rien ne s'affiche, c'est normal, votre HDFS est vide. Nous allons pouvoir charger les données utiles pour le TP dans le système de fichiers.

8. Copiez depuis le share les fichiers *author-small.txt* et *author-large.txt* dans votre compte.
9. Chargez-les dans HDFS avec les commandes suivantes :

```
$ hadoop fs -put /author-small.txt hdfs:///author-small.txt
$ hadoop fs -put /author-large.txt hdfs:///author-large.txt
```

Ceci créé un dossier *dblp* à la racine de la hiérarchie du système de fichiers HDFS. Toutes les commandes de base sur le système de fichier peuvent être réalisée à travers l'interface *hadoop*. Voici un exemple d'utilisation classique à la Unix :

```
$ hadoop fs -ls /
Found 2 items
-rw-r--r--   1 wdmd supergroup       7878 2010-03-29 10:40 /author-small.txt
-rw-r--r--   1 wdmd supergroup  719304448 2010-03-29 10:40 /author-large.txt

$ hadoop fs -mkdir /dossier
$ hadoop fs -cp /author-small.txt /dossier
$ hadoop fs -ls /dossier
Found 1 items
-rw-r--r--   1 wdmd supergroup 7878 2010-03-29 10:55 /dossier/author-small.txt

$ hadoop fs -get /dossier/author-small.txt  fichier.txt

$ ls -l fichier.txt
-rw-r--r--  1 wdmd  staff  7878 29 mar 10:56 fichier.txt
```

La sortie de la commande `ls` ressemble fortement à la commande standard d'Unix, à l'exception de la seconde valeur de chaque ligne qui indique le facteur de réplication dans le système de fichiers distribué. Cette valeur est ici de 1 (pas de réplication), c'est-à-dire la valeur par défaut qui peut être modifiée dans le fichier de configuration *etc/hadoop/hdfs-site.xml*.

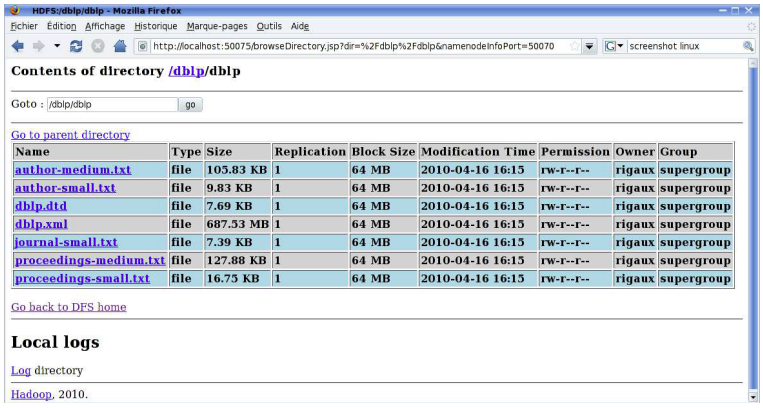


Figure 1: Naviguer dans le système de fichier HDFS

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

Le namenode instancie également un serveur web rudimentaire à l'URL *http://localhost:50070/*. Il propose des informations sur l'état courant du système de fichiers et offre une interface web pour naviguer dans la hiérarchie de fichiers. La figure 1 montre une copie d'écran de cette interface, avec la liste des fichiers d'exemple chargés dans le serveur HDFS. Observez le facteur de réplication (ici, 1), et la grande taille de blocs de 64 Mo.

## 2 Exécuter des jobs MAPREDUCE

Nous pouvons maintenant exécuter des jobs MAPREDUCE pour traiter des fichiers stockés dans HDFS. Le job suivant est un premier exemple parcourant des fichiers texte extraits du jeu de données DBLP. Nous suggérons ensuite certaines améliorations et expérimentations.

1. Vous devez d'abord lancer le serveur MAPREDUCE :

   ```
   start-yarn.sh
   ```

   Notre exemple traite les fichiers de données extraits de DBLP et transformés en fichiers plats pour plus de simplicité. Vous pouvez trouver ces fichiers de tailles variées, nommés *authors-xxx.txt* sur le *share* avec le code Java associé. La plus petite taille de fichier est de quelques Ko, la plus grande 300 Mo. Il s'agit cependant de petits jeux de données pour HADOOP, mais cela est suffisant pour nos exercices préliminaires.

   Le format de fichier est simple. Il consiste en une ligne par paire (author,title), avec des champs séparés par des caractères de tabulation, de la façon suivante :

```
<author name> <conference name> <title> <year>
```

Notre job MAPREDUCE compte le nombre de publications trouvées pour chaque auteur.Nous avons décomposé le code en deux fichiers Java, disponibles sur le site. Le premier, ci-dessous, fournit une implantation des opérateurs MAP et REDUCE simultanément.

```
/**
 * Import the necessary Java packages
 */

import java.io.IOException;
import java.util.Scanner;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

/**
 * A Mapreduce example for Hadoop. It extracts some basic
 * information from a text file derived from the DBLP dataset.
 */
public class Authors {

  /**
   * The Mapper class -- it takes a line from the input file and
   * extracts the string before the first tab (= the author name)
   */
  public static class AuthorsMapper extends
          Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text author = new Text();

        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {

          /* Open a Java scanner object to parse the line */
          Scanner line = new Scanner(value.toString());
          line.useDelimiter("\t");
          author.set(line.next());
          context.write(author, one);
        }
  }

  /**
   * The Reducer class -- receives pairs (author, <list of counts>)
   * and sums up the counts of publications per author
   */
  public static class CountReducer extends
          Reducer<Text, IntWritable, Text, IntWritable> {
```

```
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context)
                throws IOException, InterruptedException {

          /* Iterate on the list to compute the count */
          int count = 0;
          for (IntWritable val : values) {
                count += val.get();
          }

          result.set(count);
          context.write(key, result);
        }
  }
}
```

HADOOP fournit deux classes abstraites, `Mapper` et `Reducer`, qui doivent être dérivées et spécialisées par l'implantation des méthodes *map()* et *reduce()* respectivement. Les paramètres formels de chaque classe abstraite décrivent respectivement les types des clés d'entrée, des valeurs d'entrée, et des valeurs de sortie. L'environnement fournit également une liste de type de données sérialisables qui doivent être utilisés pour représenter les valeurs échangées dans un enchaînement de tâches MAPREDUCE. Notre exemple repose sur trois types : `LongWritable` (utilisé pour la clé d'entrée, c'est-à-dire le numéro de ligne), `IntWritable` (utilisé pour compter les occurrences) et `Text` (un type générique pour les chaînes de caratères). Enfin, la classe `Context` permet au code utilisateur d'interagir avec le système MAPREDUCE.

Considérons d'abord la méthode *map()* de notre classe `Mapper` (dérivée). Elle prend en entrée des paires *(key, value)*, *key* étant un numéro de ligne du fichier d'entrée (automatiquement engendré par le système, et non utilisé par notre fonction), et *value* le contenu de la ligne elle-même. Notre code prend simplement la partie de la ligne qui précède la première tabulation, interprétée comme le nom d'auteur, et produit une paire *(author, 1)*.

La méthode *reduce()* est presque aussi simple. L'entrée consiste en une clé (le nom d'un auteur) et une liste du nombre de ses publications trouvé par tous les mappers de cet auteur. Nous parcourons simplement cette liste pour additionner ces nombres.

Le second programme Java montre comment un job est soumis à l'environnement MAPREDUCE. Les commentaires du code expliquent ce code. Un aspect important est l'objet `Configuration` qui charge les fichiers de configuration qui décrivent les paramètres de l'environnement MAPREDUCE. Le même job peut être exécuté indifféremment en mode local ou distribué, selon la configuration choisie à l'exécution. Ceci permet de tester une paire de fonctions (map,reduce) sur des jeux de données à la fois petits et locaux, avant de soumettre au système de plus longues tâches.

```
/**
```

```
 * Example of a simple MapReduce job: it reads
 * file containing authors and publications, and
 * produce each author with her publication count.
 */

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/**
 * The follozing class implements the Job submission, based on
 * the Mapper (AuthorsMapper) and the Reducer (CountReducer)
 */
public class AuthorsJob {

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

        /*
         * Load the Haddop configuration. IMPORTANT: the
         * $HADOOP_HOME/conf directory must be in the CLASSPATH
         */
        Configuration conf = new Configuration();

        /* We expect two arguments */

        if (args.length != 2) {
          System.err.println("Usage: AuthorsJob <in> <out>");
          System.exit(2);
        }

        /* Allright, define and submit the job */
        Job job=Job.getInstance(conf);

        /* Define the Mapper and the Reducer */
        job.setMapperClass(Authors.AuthorsMapper.class);
        job.setReducerClass(Authors.CountReducer.class);

        /* Define the output type */
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        /* Set the input and the output */
        FileInputFormat.setInputPaths(job,new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        job.setJarByClass(AuthorsJob.class);
        job.submit();
  }
}
```

Le second objet, important également, est l'instance de la classe `Job` qui agit comme une interface avec l'environnement MAPREDUCE pour spécifier l'entrée, la sortie, et les fonctions *map()* et *reduce()*. Notre exemple propose une spécification minimale.

Le job peut être directement exécuté par `java AuthorsJob <inputfile> <outputdir>`. Cela produit un dossier de sortie `outputdir` (qui ne doit pas exister avant l'exécution du job) avec un ensemble de fichier, un pour chaque reducer, contenant le résultat.

2. Dans votre fichier de configuration du shell (`~\.bashrc`), modifiez votre variable CLASSPATH :

```
export CLASSPATH=$CLASSPATH:.: <mettre ici le résultat de la
commande 'hadoop classpath'>
```

3. Compilez votre job MAPREDUCE :

```
javac Authors.java AuthorsJob.java
```

4. Lancez le traitement du fichier *author-small.txt* :

```
java AuthorsJob hdfs:///author-small.txt hdfs:///resultat
```

Voici par exemple un extrait des résultats obtenus dans le dossier `hdfs:///resultat`:

```
Dominique Decouchant    1
E. C. Chow      1
E. Harold Williams      1
Edward Omiecinski       1
Eric N. Hanson  1
Eugene J. Shekita       1
Gail E. Kaiser  1
Guido Moerkotte 1
Hanan Samet     2
Hector Garcia-Molina    2
Injun Choi      1
(...)
```

Observez que les auteurs sont classés en ordre alphabétique, ce qui est un effet de bord souhaitable de l'environnement map reduce. Le tri est effectué lors de la phase de shuffle qui rassemble les paires intermédiaires qui partagent la même valeur de clé.

# 3 Feuille de route des 3 prochaines séances

1. Vous avez obtenu le résultat de la dernière question ? Bravo ! Vous avez déjà 11 à votre TP ! (montrez cela à votre chargé de TP).
2. Traitez le fichier *author-large* pour trouver le nombre de publications de votre chercheur favori (choisissez bien, si si...). Le bon résultat ? 12 points.
3. Comptez le nombre d'occurrence de chaque mot dans les titres d'article, et donnez le mot significatif le plus populaire (utilisez la commande Unix `sort -n -k` pour trier). Cette mission rapporte un 14.

Vous pouvez ensuite, par ordre de difficulté :

- +1 : Faire un produit scalaire de 2 vecteurs de bonne taille (10 000 nombres). Les 2 vecteurs constitueront les 2 colonnes d'un fichier.
- +3 : Réalisez les questions précédentes (comptage des publications, comptage des mots, produit scalaire) avec PigLatin (voir plus bas).
- +2 : Faire une multiplication de matrices de bonne taille (en Java ou Pig).
- +1 : Déployez le calcul sur 2 machines.
- +2 :Déployez le calcul sur plusieurs machines en démontrant une accélération (prévenir avant de lancer l'exécution).

# 4 Scripts PIGLATIN

Une version de PIGLATIN est diponible sur le *share* (adresse officielle : `http://hadoop.apache.org/pig/`). Décompressez-là (par exemple dans *$home/pig*). Ajoutez le dossier *pig/bin* à votre PATH (modifiez les chemins pour vous adaptez à votre configuration) :

```
$ export PATH=$PATH:$HOME/pig/bin
```

Vérifiez que l'interface en ligne de commandes fonctionne. La commande `pig -x local` devrait produire le résultat suivant :

```
$ pig -x local
[main] INFO  org.apache.pig.Main - Logging error messages to: xxx.log
grunt>
```

PIG accepte des commandes d'un interprèteur appelé `grunt` qui fonctionne en mode "local" (les fichiers sont lus depuis le système de fichiers local) ou en mode "MAPREDUCE". Le premier mode est sufisant pour avoir un aperçu des caractéristiques principales de PIGLATIN. Il est également utile en phase de test de nouveaux scripts pouvant nécessiter plusieurs heures de calcul sur de grands jeux de données.

Nous nous référons à la présentation de PIGLATIN qui peut être trouvée dans le chapitre dédié au calcul distribué. Exécuter PIGLATIN avec l'interprêteur de commandes est un jeu d'enfant. Comme objectif initial, nous suggérons au lecteur de fabriquer un script équivalent au job MAPREDUCE décrit à la section précédente.

# 5 Exécution en mode cluster (optionnel)

Nous donnons maintenant quelques pistes pour exécuter HADOOP sur un vrai cluster. Tant qu'il s'agit de données de test, ceci n'est pas réellement nécessaire, car ni les principes ni le code ne change par rapport au mode pseudo-distribué. Si vous avez besoin de traiter des jeux de données véritablement de grande taille, et / ou d'utiliser HADOOP dans un environnement réel, un vrai cluster de machine est évidemment nécessaire. Comprendre l'architecture d'un vrai cluster HADOOP peut également être intéressant en soi. Vous avez bien sûr besoin d'au moins machines interconnectées, qui idéalement partagent un système de fichiers classique comme NFS, avec les droits d'administration système. Avant d'aller plus loin, remarquez que si votre objectif est de traiter de très grands jeux de données, vous n'avez pas besoin d'installer votre propre cluster, mais vous pouvez un environnement de calcul «dans le nuage »(Cloud) utilisant HADOOP, comme par exemple Amazon Web Services (*http://aws.amazon.com* ) ou Cloudera (*http://www.cloudera.com*).

## 5.1 Configuration d'HADOOP en mode cluster

La plupart des paramètres qui influencent le mode d'exécution d'HADOOP sont réglés dans les fichiers de configuration situés dans le dossier *conf*. Afin de passer facilement d'un mode à l'autre, vous pouvez simplement copier *conf* en par exemple *conf-cluster*. Le choix parmi les deux configurations est réalisé par la variable d'environnement `HADOOP_CONF_DIR`. Modifiez cette variables pour la valeur choisie :

```
export HADOOP_CONF_DIR=$HADOOP_HOME/conf-cluster
```

Pour simplifier, nous supposons que tous les noeuds du cluster partagent le même fichier de configuration (accessible via un système de fichier à la NFS). Si vos machines sont hétérogènes, vous pouvez adapter la configuration pour chaque machine.

Le fichier *slaves* contient la liste des noeuds dans le cluster, référencés par leur nom ou leur IP. Voici par exemple le contenu de *slaves* pour un simple cluster de 10 noeuds situé à l'INRIA :

```
node1.gemo.saclay.inria.fr
node2.gemo.saclay.inria.fr
node3.gemo.saclay.inria.fr
...
node10.gemo.saclay.inria.fr
```

Il existe un fichier *masters* qui contient le nom du Nameserver (serveur de noms) secondaire. Vous pouvez le laisser inchangé.

Avant d'essayer de démarrer votre cluster, vous devriez consulter les fichiers de configuration XML *core-site.xml*, *hdfs-site.xml* et *mapred-site.xml*. Ils contiennent les paramètres (ou *properties*) concernant respectivement HADOOP proprement-dit, HDFS et MAPREDUCE. Voici par exemple un fichier *hdfs-site.xml* commenté avec quelques paramètres importants.

```
<?xml version="1.0"?>

<configuration>
```

```
<! Amount of replication of
                each data chunk -->
<property>
    <name>dfs.replication</name>
     <value>3</value>
  </property>

  <!-- Disk(s) and directory/ies
            for filesystem info -->
<property>
    <name>dfs.name.dir</name>
    <value>/disk1/hdfs/name</value>
  </property>

  <!-- Disk(s) and directory/ies
            for data chunks -->
<property>
    <name>dfs.data.dir</name>
    <value>/disk1/hdfs/data</value>
  </property>
</configuration>
```

### 5.2 Démarrer, stopper et administrer HADOOP

Les serveurs /hadoop/ sont lancés avec la commande *start-dfs.sh* (située dans *bin*). Elle lance le Namenode (serveur de noms) sur la machine locale (c'est-à-dire la machine exécutant la commande), un datanode (noeud de données) sur chacune de machines listées dans le fichier *slaves*, et un Namenode secondaire sur la machine listée dans le fichier *masters*. Ces scripts rapportent leurs messages d'erreurs dans les fichiers de logs dans le dossier *HADOOP_HOME/logs*. Avant que votre cluster ne soit opérationnel, vous aurez probablement à parcourir ces fichiers plus d'une fois pour trouver et corriger les problèmes (que nous espérons peu nombreux) relatifs à votre environnement. Le système HDFS est bien sûr interrompu avec *stop-dfs.sh*.

Un environnement MAPREDUCE est lancé avec la commande *start-mapred.sh* qui exécute un JobTracker (un noeud maître MAPREDUCE) sur la machine locale, et un tasktracker (les Workers dans la terminologie MAPREDUCE de Google) sur les machines listées dans *slaves*.

Une autre commande utile est *hadoop-env.sh* qui permet de régler de nombreux paramètres modifiant le comportement d'HADOOP. Le tampon mémoire utilisé par chaque noeud est par exemple déterminé par `HADOOP_HEAPSIZE`. La liste de ces paramètres dépasse le cadre de cette introduction. Nous renvoyons le lecteur à la documentation en ligne.

## 6 FAQ

1. Erreur «could only be replicated to 0 nodes, instead of 1 »: si la perte des données n'est pas un problème, arrêtez le dfs, effacez les repertoires hadoop dans /tmp, reformatez le namenode, relancez le dfs.

2. Erreur de ssh (message d'erreur ou demande d'ajout au known_hosts systématique) : faire `export SSH_AUTH_SOCK=0`.

3. Erreur de ssh (demande de mot de passe ssh systématique) : attention au guillemets si vous faites un copier-coller de la commande `ssh-keygen`. Egalement, faites en sorte que votre répertoire `~/.ssh` soit positionné avec un `chmod 0700`, de plus le fichier `~/.ssh/authorized_keys` doit être en `chmod 0600`. Appliquez également la commande `ssh-add`.

4. De (nombreux) fichiers de log sont disponibles dans */logs* pour exploration en cas d'erreur.