Projet Cloud et BigData: Le smart City au secours du délestage

Alain Tchana, ENSEEIHT, IRIT/Equipe SEPIA
email: alain.tchana@enseeiht.fr
http://tchana.perso.enseeiht.fr

1 Présentation du problème

Dans plusieurs pays du monde, il arrive très souvent que la demande en électricité soit supérieure à la production. En conséquence, le producteur et fournisseur d'électricité (ENEO) pratique régulièrement des actes dits de "délestage". Il s'agit de la privation totale d'électricité à des quartiers ou villes pendant des heures, voire des jours. Cette situation impacte fortement l'économie du pays (une entreprise privée d'électricité est dans l'incapacité de produire), source d'échec scolaire pour les uns, de mal des yeux pour ceux qui utilisent les moyens d'éclairage rudimentaires (lampe à pétrole par exemple), et au final aboutie à l'exacerbation des tensions dans la population (le ras-le-bol se fait de plus en plus ressentir). A production équivalente, ce document décrit une solution qui permet de limiter (voire annuler) les délestages.

Dans le but de faire passer l'idée générale de la solution, nous allons utiliser des hypothèses simples tout en restant proche de la réalité. Supposons qu'un quartier n'est composé que de maisons. Soit une maison avec les sources de consommation d'électricité suivantes : TV, frigo, congélateur, des ampoules. Soit C la consommation électrique de l'ensemble de ces sources dans la maison. Pour simplifier, nous supposons que toutes les maisons ont la même consommation C. Soit N le nombre de maisons dans un quartier. Soit Q le nombre de quartiers dans le pays. Soit P la capacité de production électrique du fournisseur. Le problème actuel peut être formalisé de la façon suivante :

$$P < C \times N \times Q \tag{1}$$

L'approche actuelle des fournisseurs est la suivante. Choisir Q' quartiers de telle sorte que

$$P \geq C \times N \times Q' \tag{2}$$

$Q - Q'$ quartiers seront totalement privés d'électricité, voir la figure 1.

2 Présentation de la solution

2.1 Idée de base

L'idée de base derrière notre solution vient du constat suivant : l'électricité est consommée de façon non "intelligente" par la population. En effet, à chaque fois qu'un quartier a de l'élec-

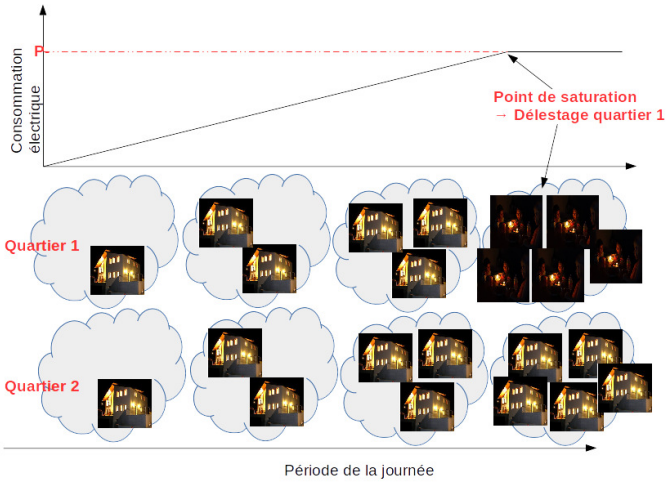

FIG. 1 – *La situation actuelle : délestage total.*

tricité, chaque maison consomme le plus possible le quartier consomme $N \times C$. Notre solution consiste à inciter la population à consommer peu d'électricité lorsque le fournisseur le souhaite de telle sorte que chaque maison consomme C' avec $C' < C$. Ceci signifit par exemple n'alimenter dans la maison que des ampoules lorsque la situation est critique : il s'agit d'une alimentation "dégradée" de la maison, voir la figure 2. Nous expliquons plus bas comment mettre en place la solution. Ainsi, au lieu qu'un quartier consomme $N \times C$, il consommera $N \times C'$. De cette façon, le fournisseur pourra alimenter presque tous les quartiers. C' sera calculer de telle sorte que

$$P \geq C' \times N \times Q \tag{3}$$

L'on peut observer que contrairement à l'inéquation 2 où Q' est utilisé (certains quartiers sont privés d'électricité), l'inéquation 3 utilise Q (tous les quartiers ont de l'électricité en mode dégradé). La question que l'on se pose est celle de savoir comment inciter la population à fonctionner en mode dégradé. La section suivante répond à cette question.

2.2 Description détaillée

La figure 3 illustre le fonctionnement de notre solution. Chaque maison disposera d'un compteur électrique "intelligent" capable de transmettre périodiquement au fournisseur sa consommation électrique. Ainsi, le fournisseur dispose en temps réel de la consommation électrique de chaque quartier. Lorsque le point de saturation P approche, le fournisseur calcul la quantité d'électricité maximale que chaque quartier devra atteindre. Ensuite, un SMS est envoyé à tous les foyers leur demandant de passer en mode dégradé (extinction de certains appareils) dans les 30 minutes par exemple. Si un quartier joue le jeu, alors le fournisseur continue de l'alimenter. Sinon, ce quartier est privé d'électricité. Au bout de plusieurs privations, les foyers récalcitrants joueront le jeu. **En effet, il est préférable d'avoir de l'électricité en mode dégradé (une ou deux ampoules) que de ne pas en avoir du tout**. Le niveau d'effort

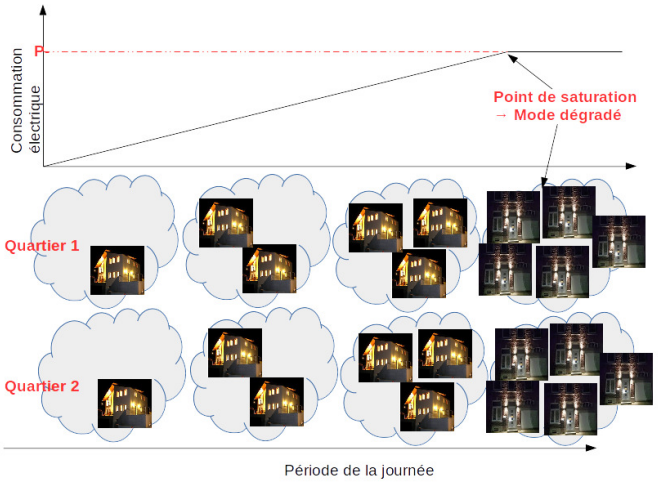

FIG. 2 – *Conséquence positive de notre solution : certains quartiers fonctionnent en mode dégradé.*

de fonctionnement en mode dégradé ne sera pas toujours le même pour un quartier tout le temps. Nous pouvons imaginer une solution de tourniquet.

Cette solution se situe dans le contexte du "smart city" (ville intelligente). Il s'agit d'une pratique/discipline récente qui favorise l'utilisation judicieuse des ressources. L'application du smart city ici prend tout son sens. Elle fait intervenir les technologies telles que :

- Les objets connectés (compteurs, transformateurs) ;
- Le bigdata : pour l'analyse en temps réel des données collectées ;
- Le cloud : pour la puissance de calcul nécessaire à l'analyse ;
- Et évidemment l'électricité

3 Travail à faire

Votre travail est de mettre en place l'infrastructure cloud et bigdata nécessaire pour la mise en oeuvre de cette solution. Pour cela, nous vous fournissons un simulateur de consommation de maisons. La section suivante présente ce simulateur.

3.1 Le simulateur

3.1.1 Description

Les maisons sont classées par types (T1, T2, etc.) et organisées par quartiers. Un quartier peut être obéissant ou récalcitrant (ne baisse pas sa consommation lorsque le fournisseur le demande). Il s'agit d'une application web, donc constituée d'une vue et d'un backend. La vue (voir la figure 4) permet de réaliser les actions suivantes :

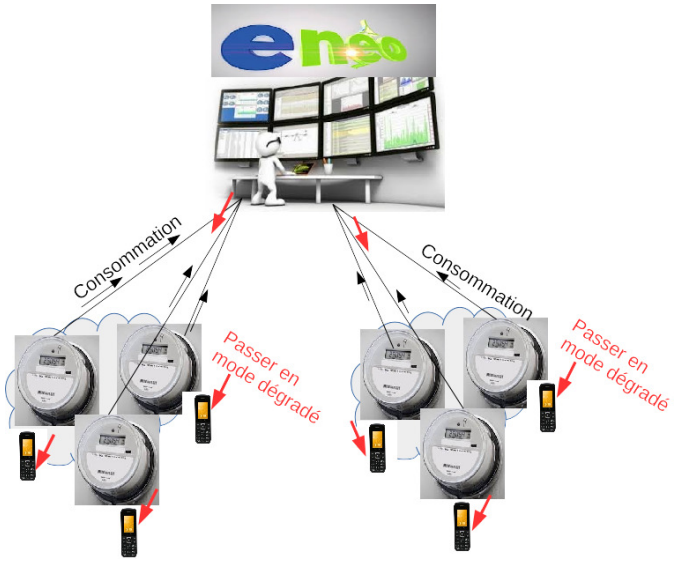

FIG. 3 – *Illustration du fonctionnement général de notre solution.*

FIG. 4 – *Dashboard de simulation.*

- Gérer les maisons, voir Figure 5
- Gérer les quartiers, voir Figure 6
- Gérer la simulation, voir Figure 7

Le backend implante toutes les actions initiées depuis la vue. Les données de configuration des maisons et quartiers sont stockées dans un fichier json. La configuration par défaut de l'application contient 3 quartiers avec au total plus de 500 maisons. La génération de la consommation se fait toutes les secondes. Les technologies utilisées pour l'implantation du backend sont : java, web-socket, springboot, et kafka[1]. Cette dernière est utilisée pour communiquer les données de consommation à l'extérieur, afin que les applications du fournisseur

[1] https ://kafka.apache.org/

FIG. 5 – *Ecran de gestion des maisons.*

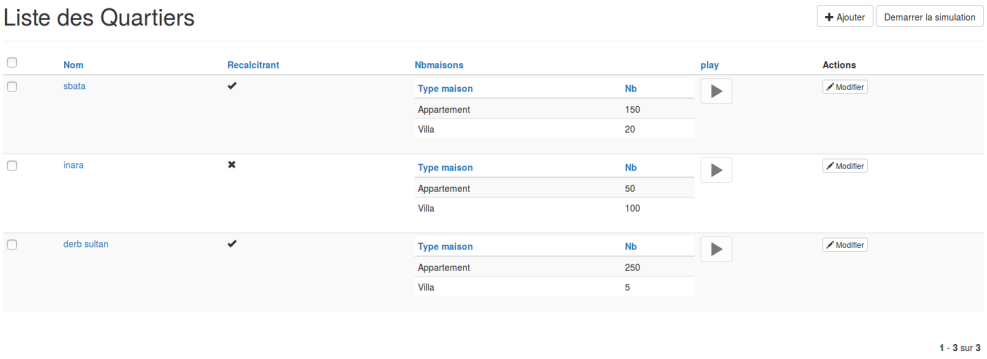

FIG. 6 – *Ecran de gestion des quartiers.*

FIG. 7 – *Ecran de gestion de la simulation.*

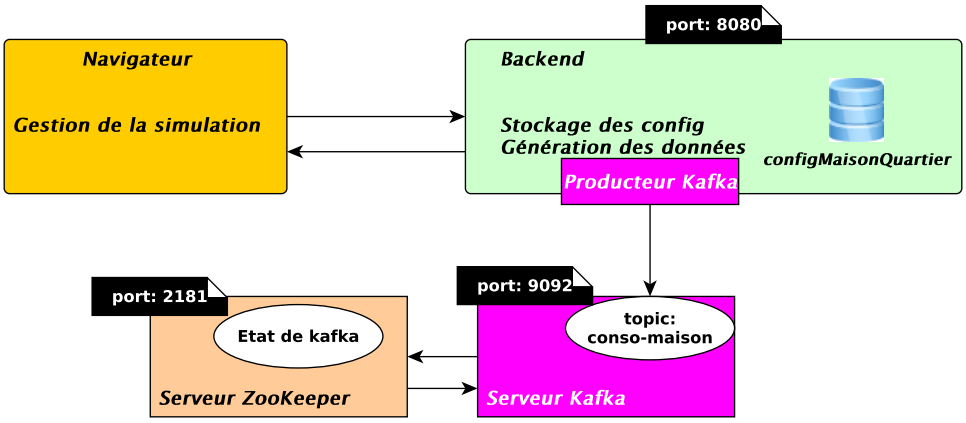

FIG. 8 – *Architecture générale contenant le simulateur.*

puissent les recevoir à la centrale. Pour cela, le backend embarque un "producteur"[2] kafka, qui publie ses données sur le topic appelé "conso-maison". Toute application qui souhaite recevoir ces données, devra s'abonner à ce topic via un consommateur kafka. Pour assurer la fiabilité de l'envoie des messages, Kafka utilise ZooKeeper[3]. La figure 8 présente l'architecture générale de cette partie simulation.

3.1.2 Déploiement

La procédure de déploiement du générateur est la suivante. Utiliser une machine virtuelle Linux (j'ai réalisé les tests sur un Ubuntu 16.04.2 TLS). Vous devez être *root* sur cette machine. Cette machine virtuelle doit avoir accès à internet (voir les TPs cloud). Dans ce déploiement, nous installerons à la fois le backend, le server kafka et ZooKeeper sur la même machine (pour la version finale du projet, les mettre dans des VMs différentes). Suivre les étapes suivantes (problème d'accent dans le listing ci-dessous).

```
#Installation de openJDK 8
apt-get update
add-apt-repository ppa :openjdk-r/ppa
apt-get update
apt-get install openjdk-8-jdk

#Installation de scala 2.10.6
wget http://www.scala-lang.org/files/archive/scala-2.10.6.tgz
tar xvfz scala-2.10.6.tgz
cp -r scala-2.10.6 /usr/local/src

#Installation de kafka 2.10
wget http ://mirror.ibcp.fr/pub/apache/kafka/0.10.0.0/kafka_2.10-0.10.0.0.tgz
tar xvfz kafka_2.10-0.10.0.0.tgz
cp -r kafka_2.10-0.10.0.0 /usr/local/src
```

[2]https ://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example
[3]https ://zookeeper.apache.org/

```
echo "#Kafka" >> ~/.bashrc
echo "export KAFKA_HOME=/usr/local/src/kafka_2.10-0.10.0.0" >> ~/.bashrc
echo "export PATH=$PATH:$KAFKA_HOME/bin" >> ~/.bashrc

echo "#Scala" >> ~/.bashrc
echo "export SCALA_HOME=/usr/local/src/scala-2.11.11" >> ~/.bashrc
echo "export PATH=$PATH:$SCALA_HOME/bin" >> ~/.bashrc

echo "#JAVA_HOME" >> ~/.bashrc
echo "export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64" >> ~/.bashrc
echo "export PATH=$PATH:$JAVA_HOME/bin" >> ~/.bashrc

#Diviser le terminal en plusieurs parties afin de controller tous les logs en parallele↩
    !

#Demarrer ZooKeeper (l'ordre est tres important)
cd /usr/local/src/kafka_2.10-0.10.0.0
zookeeper-server-start.sh config/zookeeper.properties
#Attendre le demarrage complet. Vous aurez les logs suivants:
[2017-09-30 08:31:14,674] INFO tickTime set to 3000 (org.apache.zookeeper.server.↩
    ZooKeeperServer)
[2017-09-30 08:31:14,674] INFO minSessionTimeout set to -1 (org.apache.zookeeper.server↩
    .ZooKeeperServer)
[2017-09-30 08:31:14,674] INFO maxSessionTimeout set to -1 (org.apache.zookeeper.server↩
    .ZooKeeperServer)
[2017-09-30 08:31:14,683] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.↩
    zookeeper.server.NIOServerCnxnFactory)

#Changer de terminal
#Vider ZooKeeper de l'etat precedent de Kafka (si c'etait le cas)
cd /usr/local/src/kafka_2.10-0.10.0.0
zookeeper-shell.sh localhost:2181
#saisir
rmr /brokers
#taper entree
#Ctr-C dans ce terminal

#Demarrage de Kafka
cd /usr/local/src/kafka_2.10-0.10.0.0
kafka-server-start.sh config/server.properties
#Attendre le demarrage complet. Vous aurez les logs suivants:
[2017-09-30 08:43:09,769] INFO Registered broker 0 at path /brokers/ids/0 with ↩
    addresses: PLAINTEXT -> EndPoint(Sepia,9092,PLAINTEXT) (kafka.utils.ZkUtils)
[2017-09-30 08:43:09,780] INFO New leader is 0 (kafka.server.↩
    ZookeeperLeaderElector$LeaderChangeListener)
[2017-09-30 08:43:09,788] INFO Kafka version : 0.10.0.0 (org.apache.kafka.common.utils.↩
    AppInfoParser)
[2017-09-30 08:43:09,788] INFO Kafka commitId : b8642491e78c5a13 (org.apache.kafka.↩
    common.utils.AppInfoParser)
[2017-09-30 08:43:09,789] INFO [Kafka Server 0], started (kafka.server.KafkaServer)

#Vous aurez dans ZooKeeper les logs qui ressemblent a ceci (a cause du fait que nous ↩
    avons vide les brokers, ici kafka, precedemment):
[2017-09-30 08:43:09,684] INFO Got user-level KeeperException when processing sessionid↩
    :0x15ed18627d10000 type:delete cxid:0x26 zxid:0x4ca txntype:-1 reqpath:n/a Error ↩
    Path:/admin/preferred_replica_election Error:KeeperErrorCode = NoNode for /admin/↩
    preferred_replica_election (org.apache.zookeeper.server.PrepRequestProcessor)
[2017-09-30 08:43:09,763] INFO Got user-level KeeperException when processing sessionid↩
    :0x15ed18627d10000 type:create cxid:0x2d zxid:0x4cb txntype:-1 reqpath:n/a Error ↩
    Path:/brokers Error:KeeperErrorCode = NodeExists for /brokers (org.apache.↩
    zookeeper.server.PrepRequestProcessor)
[2017-09-30 08:43:09,764] INFO Got user-level KeeperException when processing sessionid↩
    :0x15ed18627d10000 type:create cxid:0x2e zxid:0x4cc txntype:-1 reqpath:n/a Error ↩
    Path:/brokers/ids Error:KeeperErrorCode = NodeExists for /brokers/ids (org.apache.↩
    zookeeper.server.PrepRequestProcessor)
[2017-09-30 08:43:36,000] INFO Expiring session 0x15ed17ea96b0000, timeout of 30000ms ↩
    exceeded (org.apache.zookeeper.server.ZooKeeperServer)
```

```
[2017-09-30 08:43:36,000] INFO Processed session termination for sessionid: 0↩
    x15ed17ea96b0000 (org.apache.zookeeper.server.PrepRequestProcessor)

#Demarrer le simulateur, dans une autre partie du terminal
wget http://tchana.perso.enseeiht.fr/courses/practicalClasses/projetCloudBigData.tgz
tar xvfz projetCloudBigData.tgz
cd projetCloudBigData/simulateur
java -jar GenerateurDonnees-0.0.1-SNAPSHOT.jar
#Vous aurez des logs qui ressemble a ceci (le fichier exempleSortieLogsDemarrage ↩
    contient le log entier de demarrage). Si vous observez des "error" dans vos logs, ↩
    alors reprendre le demarrage des differents logiciel (ZooKeeper, Kafka et le ↩
    simulateur):
2017-09-30 08:46:56.783  INFO 26074 --- [           main] ↩
    oConfiguration$WelcomePageHandlerMapping : Adding welcome page: ServletContext ↩
    resource [/index.html]
2017-09-30 08:46:56.929  INFO 26074 --- [           main] o.s.j.e.a.↩
    AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2017-09-30 08:46:56.935  INFO 26074 --- [           main] o.s.c.support.↩
    DefaultLifecycleProcessor  : Starting beans in phase 2147483647
2017-09-30 08:46:56.936  INFO 26074 --- [           main] o.s.m.s.b.↩
    SimpleBrokerMessageHandler     : Starting...
2017-09-30 08:46:56.936  INFO 26074 --- [           main] o.s.m.s.b.↩
    SimpleBrokerMessageHandler     : BrokerAvailabilityEvent[available=true, ↩
    SimpleBrokerMessageHandler [DefaultSubscriptionRegistry[cache[0 destination(s)], ↩
    registry[0 sessions]]]]
2017-09-30 08:46:56.936  INFO 26074 --- [           main] o.s.m.s.b.↩
    SimpleBrokerMessageHandler     : Started.
2017-09-30 08:46:56.984  INFO 26074 --- [           main] s.b.c.e.t.↩
    TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-09-30 08:46:56.991  INFO 26074 --- [           main] o.p.g.g.w.↩
    GenerateurDonneesApplication   : Started GenerateurDonneesApplication in 4.518 ↩
    seconds (JVM running for 5.09)

#Ouvrir un navigateur et se connecter a "addresseIPVM:8080/"
#Aller dans le menu "Quartier" et cliquer sur "Demarrer la simulation"
#Vous aurez les logs suivants dans le simulateur:
2017-09-30T11:46:55.766,14,inara,243,118.03607773971528
2017-09-30T11:46:55.766,14,inara,244,125.80502861521198
2017-09-30T11:46:55.766,14,inara,245,68.25639548434893
2017-09-30T11:46:55.766,14,inara,246,94.67136842107581
```

3.1.3 Arrêt

Arrêt de la simulation.

```
#Se connecter sur la page d'administration ("Quartier") de la simulation et cliquer sur↩
     "Arreter la simulation"

#Ensuite faire un Ctr-C dans le terminal ou vous aviez demarrer le generateur

#Arreter d'abord Kafka
#Ctr-C dans le terminal ou il avait ete demarre
#Attendre l'arret de Kafka

#Arreter ZooKeeper
#Ctr-C dans le terminal ou il avait ete demarre
```

3.2 Proposition d'une piste de travail

Sachant qu'il existe plusieurs déploiements possibles, compte tenu du plétore de frameworks existants, je vous propose ici une architecture (voir la figure 9). Il s'agit d'une architec-

ture basée sur Kafka-SparkStreaming-ElasticSearch-Kibana. Vous vous servirez d'un consommateur kafka pour la récupération en temps réel des données générées (voir le simulateur). Ce consommateur Kafka doit être intégré à Spark-Streaming afin que ce dernier applique un premier traitement sur les données. En effet, nous souhaitons que chaque ligne représentant la consommation d'une maison contienne un champ qui stipule que la maison appartient ou pas à un quartier VIP (les quartiers VIP ne seront jamais mis en mode dégradé). Comme les données viennent en flux, Spark-Streaming est un bon candidat. Une fois la donnée modifiée, elle est stockée dans le cluster HDFS. La suite de la pile logiciele permet le traitement des données par un administrateur, ainsi que la visualisation des traitements. Pour cela, vous allez brancher ElasticSearch au cluster HDFS et Kibana à ElasticSearch.

Une fois l'environnement logiciel mis sur pied, votre travail est d'utiliser Kibana pour réaliser les dashboards suivants :

- avoir la consommation par quartier
- classification des maisons d'un quartier en fonction de leur consommation (utilisation du machine learning, k-means dans Kibana ?)
- voir les quartiers dont la consommation dépasse un seuil sur une période donnée.

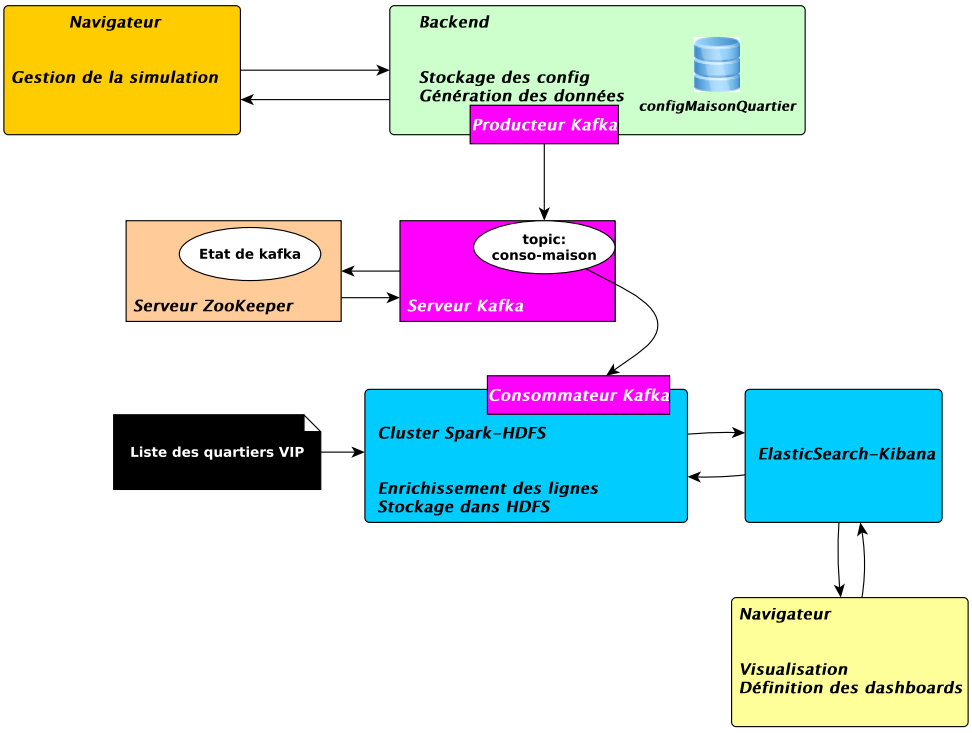

FIG. 9 – *Architecture du projet final.*