Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS
GTI795 / LOG795 – PROJET DE FIN D'ÉTUDES

TICKSMITH : INTERFACE CUSTOM ANALYTICS

ÉQUIPE 19
YVES MILLETTE – MILY01048700
RICHARD KANTCHIL – KAND18068306
MAXIME PAUL-DEGARIE – PAUM15118907
JEAN-PHILIPPE CHAN – CHAJ29068908

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

ALAIN APRIL

PATRICK CARDINAL

MONTRÉAL, 21 AOÛT 2017
ÉTÉ 2017

REMERCIEMENTS

Nous tenons à remercier les deux professeurs ayant offert temps et support dans la supervision de notre projet, soit les professeurs Alain April et Patrick Cardinal. Nous tenons également à remercier Tony Bussières et toute son équipe, chez Ticksmith, pour nous avoir permis de réaliser ce projet et nous avoir offert un support continu en plus d'avoir eu la patience de composer avec une équipe peu expérimentée avec les technologies Big Data.

TICKSMITH: INTERFACE CUSTOM ANALYTICS

YVES MILLETTE – MILY01048700
MAXIME PAUL-DEGARIE – PAUM15118907
RICHARD KANTCHIL – KAND18068306
JEAN-PHILIPPE CHAN – CHAJ29068908

RÉSUMÉ

Ce projet de fin d'études du baccalauréat en génie logiciel a été réalisé en partenariat avec l'École de Technologie Supérieure et l'entreprise TickSmith. Ce projet poursuit un projet de fin d'études réalisé à l'automne 2016. Le travail réalisé consiste en l'élaboration d'une interface web permettant à des experts financiers de saisir des formules mathématiques afin de les utiliser pour les exécuter sur des données massives (Big Data) provenant de la bourse. Diverses méthodes ont été explorées pour effectuer une preuve de concept et atteindre cet objectif. La difficulté principale était de trouver et de déployer un prototype logiciel permettant de gérer, en lots, l'exécution de ces formules impliquant des données massives sur une grappe de serveurs hébergés dans un système d'infonuagique d'Amazon, le AWS.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AJAX – Asynchronous JavaScript and XML
AWS – Amazon Web Services
Bootstrap – Cadriciel de développement web pour les interfaces
EC2 – Elastic Cloud Compute
EMR – Elastic MapReduce
HADOOP – Cadriciel pour le traitement de données massives distribuées
HTTP – HyperText Transfer Protocol
JDBC – Java Database Connectivity
jQuery – Cadriciel de développement web JavaScript
IMPALA – Moteur de base de données distribué
LIVY – Serveur HTTP pouvant démarrer des fil d'exécution Spark-Shell
S3 – Amazon Simple Storage Service
SCALA – Scalable Language, langage de programmation
SPARK - Moteur de traitement de données massives
SPRING – Cadriciel Java pour servir des applications Web.
RDS – Amazon Relational Database Service
REST – Representational state transfer

CHAPITRE 1 INTRODUCTION

1.1 Problématique et contexte

Ce projet appliqué a été réalisé avec la collaboration de l'entreprise TickSmith, dont les locaux sont situés au *3575 St Laurent Blvd Suite 512, Montréal, QC H2X 2T6*. Le projet a été encadré, pour les aspects clients, par Tony Bussières et Luiz Fernando Santos Pereira. La supervision académique du projet a été effectuée par le professeur Alain April et le professeur Patrick Cardinal. Ce projet s'inscrit dans un effort de collaboration continu entre TickSmith et l'École de Technologie Supérieure (ÉTS). Il s'agit du second projet de fin d'études à être réalisé collaborativement avec TickSmith et des étudiants de l'ÉTS. Ce projet s'inscrit dans un effort visant à continuer sur les bases déjà établies par le projet de la session d'automne 2016, réalisé par Philippe Grenier-Vallée et Luiz Fernando Santos Pereira, s'intitulant *Big Data, Small Numbers (*Grenier-Vallée, 2016).

Un prototype logiciel de génération de formules financière avait été élaboré dans le cadre du projet précédent. Ce prototype permet de convertir une formule mathématique, rédigée sous forme de texte, dans un format qui est exécutable par le langage Scala. Ce prototype permet également de créer une représentation visuelle sous forme de formule mathématique formatée grâce au logiciel LaTeX. Quelques formules financières ont également été implémentées en Scala dans le projet précédent pour démontrer la faisabilité de ce concept.

Ce projet vise à étendre les fonctionnalités du prototype existant en permettant l'exécution de ces formules sur une grappe AWS (Amazon web services) à l'aide de la technologie EMR (Elastic MapReduce, un service infonuagique, permettant l'usage sur demande de ressources de calcul en ligne.

Le prototype actuel a toutefois quelques lacunes. Il est présentement impossible de soumettre des formules pour analyse à partir de l'outil de génération de formules. L'usager ne peut donc rien accomplir présentement avec les formules générées. Ces mêmes formules ne sont d'ailleurs pas encore stockées et conséquemment sont perdues lorsque l'usager quitte le logiciel. Le nombre de formules financières qui peuvent être exécutées est aussi

restreint. La possibilité de soumettre des formules, suivre l'état de la tâche, et les stocker serait un avancement important pour prouver les concepts.

1.2 Objectifs du projet

Le projet proposé a donc pour objectif principal de continuer le développement de ce qui a été livré par les étudiants ayant participé au projet à l'automne 2016 afin de faire avancer l'état du prototype qui deviendra un nouveau service d'analyse de données financière pour leur produit logiciel TickVault. Nous avons listé les objectifs suivants afin de répondre à ces exigences.

- Concevoir et implémenter une interface graphique en HTML5 et JavaScript permettant la saisie, la consultation et l'enregistrement de formules financières ;
- Concevoir et implémenter une interface graphique en HTML5 et JavaScript permettant l'affichage et le filtrage personnalisable des données financières stockées en infonuagique sur Amazon S3 ;
- Concevoir et implémenter une interface graphique en HTML5 et JavaScript permettant l'affichage de graphiques personnalisés basés sur les données précédemment traitées ;
- Concevoir et implémenter un module Spark en Scala et Java pouvant être déployé sur une grappe de machines virtuelles EC2 d'Amazon et permettant la soumission de formules financières à distance ainsi que le lancement d'unités de traitements sur les données à partir de ces formules. Le module devra être en mesure de stocker les données traitées dans une base de données Cloudera Impala ;
- Concevoir une base de données MySQL permettant l'enregistrement des formules financières qui auront été saisies par les usagers du logiciel ;
- Concevoir des services Spring implémentant une liaison entre la base de données MySQL, le module Spark exécuté sur AWS EC3 ainsi qu'une base de données Cloudera Impala.

1.3 Méthodologie

Le travail pour la durée du projet a été effectué de façon itérative à l'aide d'une approche méthodologique agile. Pour ce faire, une rencontre SCRUM hebdomadaire a eu

lieu, aux bureaux de TickSmith, afin d'effectuer un suivi de l'avancement du projet. Un historique des tâches à effectuer pour une semaine donnée a été maintenu et les membres de l'équipe avaient la responsabilité de maintenir cet historique à jour en mentionnant ce qu'ils ont fait pour chacun des sprints. Ils devaient également relever les éléments les ayant bloqués dans l'avancement de leurs tâches.

Le code source du projet a été maintenu dans un dépôt GIT privé, hébergé par le service github.com. Ce service a permis de définir des jalons et des tâches devant être effectuées. Un jalon a été défini à chaque semaine de travail, ce qui représente la durée d'un « Sprint », et inclut un certain nombre de sous-tâches. Les membres de l'équipe peuvent s'assigner eux-mêmes à ces tâches au fur et à mesure qu'ils les complétaient.

La version initiale du prototype étant développée en Java et Scala, les membres de l'équipe devaient s'y initier et améliorer le prototype existant. Les membres de l'équipe étaient en mesure de tester leur code localement à l'aide d'échantillons réduits des données fournies par TickSmith. Pour effectuer des essais à l'échelle, ils pouvaient utiliser des grappes EMR d'AWS mise à leur disposition afin de tester l'implémentation de la solution avec un échantillon de données plus important.

Une partie considérable de la conception et de l'analyse ont été réalisées par TickSmith. L'équipe de projet devait s'assurer de respecter les exigences et la conception. Des prototypes d'interface ont notamment été réalisés par TickSmith.

1.4 Composition de l'équipe

Prénom	Responsabilités
1. Yves Millette	Gestion de l'interface entre le front-end et les job Spark sur AWS EMR. Regroupement du code source des deux projets de départ (interface de Luiz et jobs Spark de Philippe). Implémentation d'un système de déploiement pour les cluster AWS afin que l'on puisse lancer des jobs Spark à distance.
2. Jean-Philippe Chan	Design de l'interface graphique pour l'onglet de la base de données. Développer l'interface avec la base de données. Documenter l'exécution du projet sur une grappe AWS plutôt qu'en mode local.
3. Maxime Paul-Degarie	Implémentation de formules financières en Scala. Implémentation d'un script pour exécuter les formules en lot (en Scala) qui sont exécutées sur Spark (EMR).
4. Richard Kantchil	Design de l'interface usager web générale et design de l'onglet pour la base de données des formules mathématiques. Conception de l'architecture de la base de données pour les formules. Interface entre le front-end et la base de données pour la saisie et le chargement de formules financières.

Tableau 1.1 : Composition de l'équipe

1.5 Livrables

Nom de l'artefact	Description
Proposition de projet	Document contenant la proposition initiale du contenu du projet.
Front-end (layout)	Interface usager en HTML et JavaScript présentant les informations des analyses et permettant de générer et de visualiser des formules financières. Réalisé à l'aide de HTML5, Bootstrap, jQuery et AMCharts. Communique par requêtes HTTP (AJAX) avec les services de l'application Spring boot.
Application Spring Boot (back-end)	Application permettant au front-end de communiquer avec des services REST et de conséquemment communiquer avec le serveur Livy pour le lancement de tâches Spark.
Architecture de base de données	Script SQL permettant de réaliser la création des tables de la base de données du projet. Cette base de données servira principalement à effectuer le stockage des formules mathématiques saisies par les usagers.
Application Spark	L'application Spark d'analyse ayant été réalisée par Philippe Grenier-Vallée améliorée lors des itérations, tout au long du projet.
Rapport d'étape	Rapport de projet à rendre à la mi-session. Rapport détaillant l'avancement du projet et des difficultés rencontrées. Non-complété.
Rapport final	Ce document.
Présentation orale	Une présentation orale aura lieu entre les 3 et 8 août 2017 afin de présenter le projet à des collègues de classe et à des enseignants aux fins d'évaluation.

Tableau 1.2 : Livrables du projet

1.6 Risques

Risque	Impact	Probabilité	Mitigation / atténuation
Manque d'expérience (prog), Scala	Moyen	Élevé	Consultation de la documentation en ligne et poser des questions aux personnes ressources.
Manque de connaissances sur l'application qui a été réalisée en automne 2016	Faible	Élevé	Consulter les documents réalisés par Philippe l'an passé. Aussi, il est possible de poser des questions aux personnes ressources chez Ticksmith.
Manque de connaissance avec AWS	Faible	Élevé	Consultation de la documentation en ligne et poser des questions aux personnes ressources.
Manque de temps pour compléter le projet	Élevé	Moyen	Planification d'une réunion scrum hebdomadaire afin d'effectuer le suivi du projet. Planification rigoureuse des livrables du projet et de l'échéancier. Planification de périodes tampon pour les imprévus.
Manque d'expérience avec le framework Spring Boot	Faible	Élevé	Consultation du code du projet actuel et de la documentation disponible en ligne sur le framework Spring.
Manque de communication entre les membres de l'équipe	Élevé	Moyen	Planification d'une réunion scrum hebdomadaire afin d'effectuer le suivi du projet ainsi que des rencontres d'équipe. Utilisation du système de communication Slack.

Tableau 1.3 : Risques du projet

1.7 Techniques et outils

Nous listons ci-après, les outils dont nous avons fait usage dans le but de réaliser la conception attendue.

- *Scala :* Le langage Scala tire son nom du mot anglais « scalable ». En effet, ce langage basé sur Java a la faculté de pouvoir être étendu en fonction des besoins des programmeurs. Il possède en plus d'une syntaxe allégée puisque l'utilisation des fonctionnalités de Scala passe par l'appel aux fonctions des différentes bibliothèques. Supporté par Apache Spark, il sera utilisé en vue de la mise en œuvre des traitements de données.
- *Apache Spark :* Apache Spark est un framework de traitements de données massives (Big Data), open source construit pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation
- *JQuery :* Query est une bibliothèque JavaScript libre et multi-plateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web. Elle sera utilisée en vue de rendre dynamique les interfaces reliées à notre solution.
- *Bootstrap :* Bootstrap est un framework développé par l'équipe du réseau social Twitter. Il utilise les langages HTML, CSS et JavaScript dans le but de fournir aux développeurs des outils pour créer un site facilement. Cet outil sera utilisé en vue de concevoir le squelette des différentes interfaces.
- *Amcharts :* amCharts *est* une bibliothèque JavaScript graphique avancée qui nous permettra d'insérer des contenus graphiques et des analytiques au sein des interfaces qui seront implémentées dans le but d'offrir à l'utilisateur un contenu synthétique sur la métrique de son choix.
- *AWS EMR :* Elastic Mapreduce est un des services web (AWS) offerts par Amazon permettant de faire de faire rouler des scripts Mapreduce, Spark et autre sur une grappe aux dimensions sélectionnées.

- *S3 :* S3 est un des services web (AWS) offerts par Amazon permettant un entreposage de données distribué avec haute redondance et une grande tolérance à la faute.
- *Presto :* Presto est un moteur de requêtes SQL distribué open source, optimisé pour l'analyse ad hoc des données avec un faible temps de latence. Presto peut traiter des

données provenant de plusieurs sources, notamment le système de fichiers distribués Hadoop et Amazon S3. Nous utiliserons cet outil pour les requêtes sur S3.

- *Cloudera Impala :* Impala est un moteur de requêtes SQL propriétaire pour les données stockées dans des cluster d'ordinateurs exécutant Apache Hadoop. Impala est intégré avec Hadoop pour utiliser les mêmes fichier et formats de données, ainsi que les cadriciels de sécurité et management de ressource utilisés par MapReduce.
- *MySQL* : MySQL est un serveur de bases de données relationnelles développé dans un souci de performances élevées en lecture, ce qui signifie qu'il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes et fortement sécurisées. Il est multi-thread et multi-utilisateur. Ce type de base de données sera notamment utilisé dans le but de stocker les formules créées par les utilisateurs.
- *Spring Boot :* À la base, Spring est un framework libre pour construire et définir l'infrastructure d'une application web java, dont il facilite le développement et les tests. Spring Boot est conçu pour simplifier le démarrage et le développement de nouvelles applications Spring.
- CloudEra Livy (maintenant Apache Livy) : Serveur http offrant un API REST pour le lancement à distance sur une grappe de serveurs de jobs Spark. Il permet également le lancement de sessions interactive à laquelle il est possible de soumettre des lignes de code uniques de façon séquencée.

Les outils utilisés dans le cadre de la gestion du projet sont de deux ordres :

- *GitHub :* est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. L'équipe du présent projet utilise les fonctionnalités offertes par cet outil pour héberger et partager le code source de la solution conçue ainsi que pour la gestion du versionnage au cours du processus de développement.
- *Slack :* est un service en ligne de discussion instantanée orientée développement qui est utilisée par l'équipe dans le cadre de ses besoins de communication. En outre, nous avons effectué une connection au cadre de gestion du versionnage utilisé par le projet afin d'être informés de façon dynamique des modifications apportées au code source.

- Google Drive : Stockage des fichiers et documents liés à la gestion de projet. Tous les livrables pour l'ÉTS seront stockés dans Google Drive ainsi que les documents de spécification fournis par TickSmith.

CHAPITRE 2 CONTRIBUTION DE MAXIME PAUL-DÉGARIE

2.1 Infrastructure Amazon

Pendant la réalisation du projet, l'infrastructure Amazon Cloud a été utilisée. Plusieurs produits d'Amazon ont été utilisés, par exemple : AWS, EMR et S3.

Amazon Web Services (AWS) a été utilisé pour déployer des instances. Ubuntu est le système d'exploitation qui a principalement été utilisé lors du déploiement des instances. Une zone de sécurité avait déjà été définie, par Ticksmith, et c'est cette zone de sécurité qui a été utilisée. Il est important de noter qu'il est possible de créer des instances avec l'interface graphique, ou bien à l'aide de lignes de commandes.

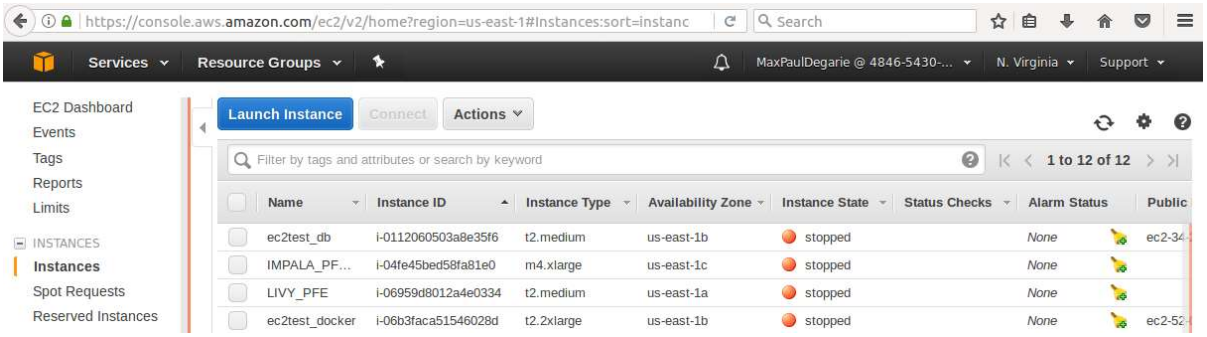


Figure 2.1 : Instances dans le tableau de bord AWS

Amazon Elastic MapReduce (EMR) a été utilisé pour héberger les grappes d'ordinateurs (c.-à-d. clusters) qui exécutent le code de Livy et d'Impala. Lors de la création d'un cluster, les paramètres sont relativement similaires à la création d'une instance. Par contre, il est possible de définir la quantité de noeuds dans un cluster. Dans le cas de ce projet, un noeud parent avec deux noeuds enfants a été utilisé. Voici une capture d'écran d'une commande effectuant la création d'un cluster Livy:

```
aws emr create-cluster --applications Name=Hive Name=Spark --tags 'Name=PFE_TEST_EMR' --ec2-attributes
'{"KeyName":"TickSmithPFE","InstanceProfile":"EMR_EC2_DefaultRole","SubnetId":"subnet-433c9d18","EmrManagedSlaveSecurityGroup":"sg-
eb9e2196","EmrManagedMasterSecurityGroup":"sg-fda01f80","AdditionalMasterSecurityGroups":["sg-d7003ba9"]}' --release-label emr-5.6.0
--log-uri 's3n://ticksmith-pfe/elasticmapreduce/' --instance-groups
'[{"InstanceCount":2,"InstanceGroupType":"CORE","InstanceType":"m3.xlarge","Name":"Core - 2"},
{"InstanceCount":1,"InstanceGroupType":"MASTER","InstanceType":"m3.xlarge","Name":"Master - 1"}]' --configurations
'[{"Classification":"spark-hive-site","Properties":
{"javax.jdo.option.ConnectionUserName":"hive","javax.jdo.option.ConnectionDriverName":"org.mariadb.jdbc.Driver","javax.jdo.option.Conne
hive?createDatabaseIfNotExist=true"},"Configurations":[]},{"Classification":"hive-site","Properties":
{"javax.jdo.option.ConnectionUserName":"hive","javax.jdo.option.ConnectionDriverName":"org.mariadb.jdbc.Driver","javax.jdo.option.Conne
hive?createDatabaseIfNotExist=true"},"Configurations":[]}]' --auto-scaling-role EMR_AutoScaling_DefaultRole --bootstrap-actions
'[{"Path":"s3://ticksmith-pfe/livy/livy","Name":"Custom action"}]' --ebs-root-volume-size 10 --service-role EMR_DefaultRole --enable-
debugging --name 'TestClusterPFE' --scale-down-behavior TERMINATE_AT_INSTANCE_HOUR --region us-east-1
```

Figure 2.2 : Ligne de commande pour le lancement du cluster EMR

Il est possible de voir, dans cette capture d'écran, qu'il y a beaucoup de paramètres qui y sont spécifiés. Dans ce cas-ci, plusieurs éléments tels que : le nom du cluster, la clé privée pour la connexion à distance, le réseau utilisé, la zone de sécurité, le type d'instance et bien d'autres y ont été spécifiés.

Amazon Simple Storage Service (S3) a été utilisé pour l'enregistrement de fichiers. C'est un espace qui est très souvent utilisé pour entreposer des fichiers de sauvegarde, des bases de données, et bien d'autres données. Ce qui est très pratique avec S3, c'est qu'il est accessible à partir de la zone de sécurité qui était utilisé pour le déploiement des clusters. Conséquemment, il était possible de spécifier des scripts, par exemple un script Bootstrap action, lors du déploiement d'un cluster et il allait directement le chercher dans S3. Dans la capture d'écran suivante, il est possible de voir l'environnement S3 que le script Bootstrap action utilise pour le déploiement du cluster Livy :

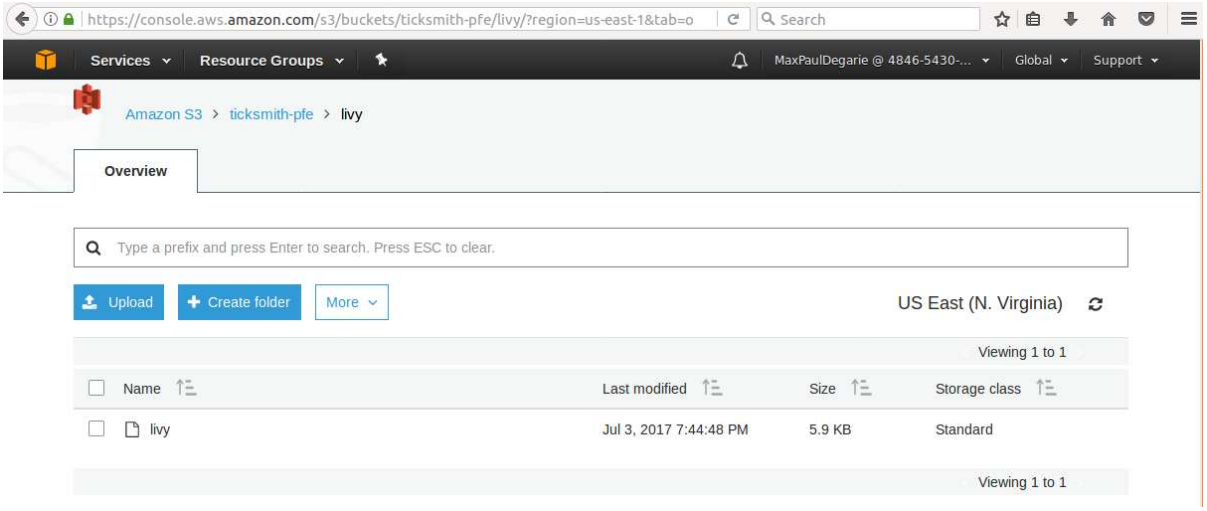


Figure 2.3 : Localisation du fichier de bootstrap pour Livy dans S3

2.2 Installation de déploiement d'applications

Plusieurs packages étaient nécessaires afin de développer et tester l'interface graphique du projet, exécuter des jobs Spark et pour bien d'autres tâches. Au niveau du traitement Big Data, c'est Spark et Livy qui étaient les plus centraux. Apache Spark est en cadriciel libre pour le Big Data qui est utilisé par des applications, telle que Livy, et qui permet d'effectuer des analyses sophistiquées sur de très grandes quantités de données. Livy possède une fonctionnalité d'interface REST qui interagit avec Spark.

Au début des essais, ces packages ont été installés localement sur une instance Linux Ubuntu. Plusieurs tutoriels ont été suivis en ligne afin de réaliser la première installation d'essai de Livy. La version 0.3.0 a été utilisée, car il y avait beaucoup plus de documentation en ligne pour son installation. Pour ce faire, il y a un fichier principal de configuration qui est important pour Livy. Ce fichier est « Livy.conf ». Il est aussi très important de définir deux variables d'environnement nécessaires pour Livy:

1) export SPARK_HOME=/usr/lib/spark
2) export HADOOP_CONF_DIR=/etc/hadoop/conf

Normalement, les dossiers contenant ces données seraient localisés à ces endroits, sinon il est possible de définir un autre chemin pour accéder à ces dossiers. Par la suite, il ne reste qu'à démarrer le serveur Livy avec la commande « *./bin/livy-server* ». Finalement, il sera possible d'accéder à l'interface de Livy avec un navigateur web, en utilisant la commande « *localhost:8998* ».

Des tests ont été réalisés pour permettre l'intégration de Apache Sentry à la base de données Impala/mysql. Sentry gère les accès à la base de données, puisqu'il est possible de définir des rôles qui seront ensuite utilisés par Sentry. Malheureusement, il possible de lire dans la documentation de Sentry que l'intégration avec la base de données MySQL n'est plus supportée depuis la dernière version de Sentry. En fait, depuis la version 8 de Sentry qui est sortie il y a quelques mois, il est préférable d'utiliser des bases de données tel que

PostGreSQL sinon des changements devront être faits au niveau de la construction de la base de données (avec MySQL).

2.3 Automatisation et scripting

Au niveau de l'automatisation, plusieurs scripts ont été utilisés pour réaliser des tâches. Premièrement, un Bootstrap action a été utilisé pour faire l'installation et la configuration de Livy sur un cluster Amazon EMR. Ce script était spécifié lors de la création du cluster et il était ainsi exécuté au démarrage. De plus, le script était conservé sur S3, alors il était accessible à partir de la zone de sécurité de Ticksmith. En bref, ce script télécharge les fichiers de Livy et il les copies aux endroits appropriés. De plus, il modifie les fichiers de configuration pour l'intégrer au cluster. Puis, il définit aussi certaines variables d'environnement pour le système d'exploitation, et bien d'autres tâches.

Deuxièmement, un script a été développé pour automatiser la création du cluster EMR avec Livy. Le but du script était de faire la création du cluster en ligne de commande, contrairement à l'interface graphique, et de retourner l'adresse IP ou le nom DNS qui avait été assigné à ce cluster. Ensuite, avec cette adresse, un tunnel SSH vers le serveur Livy était créé. Ce tunnel permettait d'accéder à l'interface de Livy sur le port 8998. Voici une capture d'écran qui démontre la création d'un cluster:

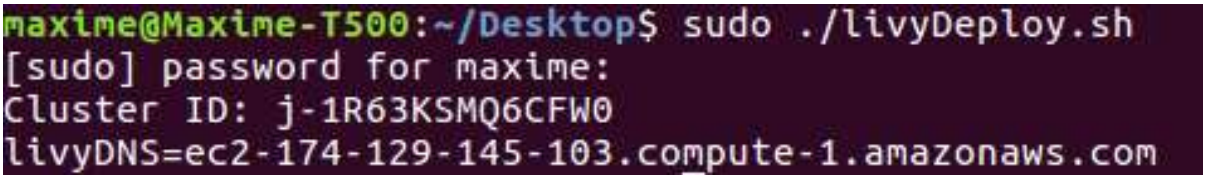


Figure 2.4 : Résultat du lancement du script de déploiement

Par la suite, un autre script a été développé afin de pouvoir éteindre toutes les instances EC2 en état *running* (en cours d'exécution). Ce script se connecte sur AWS pour lister toutes les instances qui sont en cours d'exécution et une à une, il exécute une requête pour les éteindre. C'est un script très pratique pour diminuer les frais d'utilisation lorsqu'on n'utilise pas les instances. Voici une capture d'écran qui présente l'exécution de ce script, dans ce cas-ci seulement deux instances étaient allumées :

```
maxime@Maxime-T500:~/Desktop$ ./shutdownInstances.sh
{
    "StoppingInstances": [
        {
            "InstanceId": "i-0214e4072afebcb9a",
            "PreviousState": {
                "Name": "running",
                "Code": 16
            },
            "CurrentState": {
                "Name": "stopping",
                "Code": 64
            }
        }
    ]
}
{
    "StoppingInstances": [
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-06b94b1c0deedbb28",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

Figure 2.5 : Ligne de commande pour le lancement du cluster EMR

CHAPITRE 3 CONTRIBUTION DE JEAN-PHILIPPE CHAN

3.1 Objectif du travail

Le but du travail effectué dans ce projet était de créer un fichier « .jar », qui est utilisé sur l'interface usager Analytics. Ce fichier « .jar » sera utilisé pour lire et sauvegarder un «Dataframe», dans Impala, en exécutant une requête SQL avec la commande «Spark-submit». Pour effectuer ce travail, il a été nécessaire de modifier deux fichiers Scala pour pouvoir exécuter correctement cette requête.

3.2 Analyzer.scala

Le fichier « **Analyzer.scala** » est l'élément clé de ce travail. Il effectue ce que le fichier « .jar » exécute, c'est-à-dire la lecture et la sauvegarde du « Dataframe » dans Impala. Il avait déjà été réalisé par les étudiants du PFE de l'automne 2016. Ma tâche était d'effectuer quelques modifications à ce fichier selon les nouveaux besoins du PFE.

3.3 La requête SQL

Avant cette modification, le fichier « **Analyzer.scala** » était utilisé seulement pour faire des tests et des « debugs ». Il faisait le travail de lecture et d'écriture du « Dataframe » localement. Ces « Dataframes » avaient leurs colonnes prédéfinies et étaient écrits en format « .csv » et en « .parquet ». Puisque ce fichier était fait pour exécuter localement, les données étaient lues et écrites sur Amazon S3 directement.

Suite à cette amélioration, le fichier « .scala » peut maintenant lire et sauvegarder le « Dataframe » à distance avec Impala en utilisant une requête SQL. Pour récupérer les données dans la base de données d'Impala, on doit créer des paramètres d'arguments. Tous ces paramètres sont insérés dans les instructions de manipulation, tel que présenté à la capture d'écran suivante :

```
//Select [cols, ...]
val p = new TickSmithFormulaParser
val formulaCols: String = "CONCAT('"+ params.formula+ "')"

var formuleSQL = p.toSql(p.expr, params.formula)
var selectCols = "SELECT ts, yyyymmdd, ticker, source, "+ formulaCols +" as formula, "+ formuleSQL +" as value"

//From [database.table]
var tableSelect = " FROM "+ params.databaseName + "." + params.tableInput

//Where Date between [dateBegin] and [dateEnd]
var dateSelect = " WHERE yyyymmdd between " + params.dateBegin + " and " + params.dateEnd

//Limit [nbRange] (Optional)
var limit = ""
if (params.limitTable != null){
  limit = " LIMIT "+params.limitTable
}

//println(selectCols + tableSelect + dateSelect + limit)
df = sqlContext.sql(selectCols + tableSelect + dateSelect + limit)


df.show()

df.write.mode(org.apache.spark.sql.SaveMode.Overwrite).saveAsTable(params.databaseName+"."+params.tableOutput)
val then = Calendar.getInstance.getTimeInMillis
println("This took : " + (then - now) + "ms")
```

Figure 3.1 : Code de traitement des Dataframes et enregistrement

3.4 Paramètres JCommander

Lors de l'exécution du fichier « .jar », dans la ligne de commande, on souhaite préciser ce que représente chaque argument. Pour se faire, on utilise un cadriciel nommé « JCommander » qui permet d'analyser les paramètres de ligne de commande. Ces paramètres, présentés dans la capture d'écran qui suit sont utilisés pour faire une requête SQL à Impala.

```scala
package com.ticksmith.analytics

import com.beust.jcommander.Parameter


class Parameters {

      @Parameter(
        names = Array("-YorL", "--yarnOrLocal"),
        description = "Input File ",
        required = true)
      var yarnOrLocal: String = "local"

      @Parameter(
        names = Array("-l", "--length"),
        description = "Input File ",
        required = true)
      var sliceLength: String = "00300000000000000"

      @Parameter(
        names = Array("-dbn", "--databaseName"),
        description = "Input File ",
        required = true)
      var databaseName: String = null

      @Parameter(
        names = Array("-ti", "--tableInput"),
        description = "Output File",
        required = true)
      var tableInput: String = null

      @Parameter(
        names = Array("-to", "--tableOutput"),
        description = "Output File",
        required = true)
      var tableOutput: String = null

      @Parameter(
        names = Array("-f", "--formula"),
        description = "Output File",
        required = true)
      var formula: String = null

      @Parameter(
        names = Array("-dateB", "--dateBegin"),
        description = "Output File",
        required = true)
      var dateBegin: String = null

      @Parameter(
        names = Array("-dateE", "--dateEnd"),
        description = "Output File",
        required = true)
      var dateEnd: String = null

      @Parameter(
        names = Array("-limit", "--limitTable"),
        description = "Output File",
        required = true)
      var limitTable: String = null

}
```

Figure 3.2 : Définition des paramètres JCommander

3.5 TickSmithFormulaParser.scala

Le fichier « **TickSmithFormulaParser.scala »** est utilisé pour faire la formule de parseurs. La formule doit contenir la colonne de la table de la base de données. S'il y a lieu, il peut aussi contenir un parseur que la colonne est insérée. Cette dernière est préparée pour la demande de la requête SQL « SELECT » dans « **Analyzer.scala »**.

Ce fichier était déjà présent au moment du démarrage de notre PFE, à l'été 2017. TickSmith utilise maintenant de nouvelles colonnes dans leurs « Dataframes » pour Impala,

donc il a été nécessaire de modifier le code pour que ces nouvelles colonnes soient supportées.

3.6 Exécution avec terminal

Afin de pouvoir bien tester et exécuter le fichier « .jar », en ligne de commande, il faut créer un « hive thrift » qui permet de se connecter à distance sur Amazon EC2. Avant de pouvoir créer un « thrift », il faut créer deux fichiers de configuration en « .xml » dans le dossier Spark (voir la capture d'écran qui suit).

Edit hive-site.xml $SPARK_HOME/conf/hive-site.xml

```
<configuration>
 <property>
  <name>hive.metastore.uris</name>
  <!--Make sure that <value> points to the Hive Metastore URI in your cluster -->
  <value>thrift://localhost:9083</value>
  <description>URI for client to contact metastore server</description>
 </property>
</configuration>
```

Edit hdfs-site.xml $SPARK_HOME/conf/hdfs-site.xml

```
<configuration>
 <property>
  <name>fs.s3a.access.key</name>
  <value>my_access_key</value>
 </property>
 <property>
  <name>fs.s3a.secret.key</name>
  <value>my_secret_key</value>
 </property>
</configuration>
```

Figure 3.3 : Configuration HRDS et Hive

Une fois ces deux fichiers créés, il est possible de tout de suite démarrer, en ordre, les trois instances qui sont utiles pour ce test, c'est-à-dire : TickSmithPFEDB, IMPALA_PFE_STATESTORE et IMPALA_PFE_WORKER (voir la capture d'écran suivante).

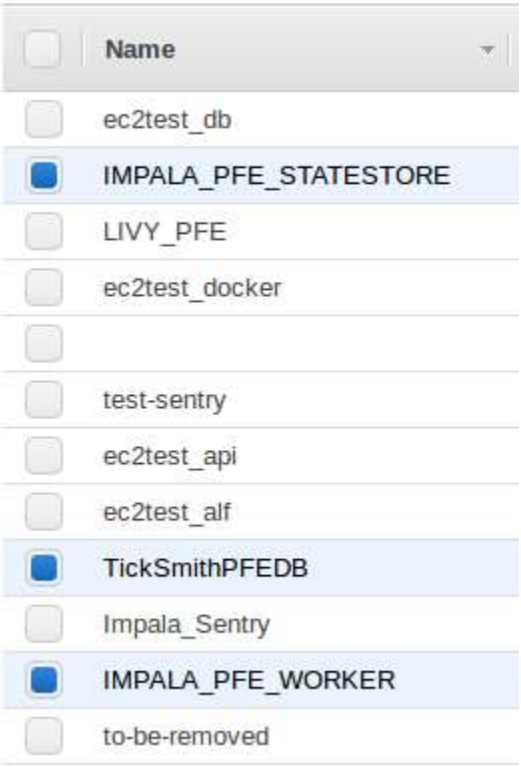


Figure 3.4 : Instances AWS nécessaires

Ensuite, il est possible de démarrer un tunnel pour le « thrift metastore» (voir la commande):

```
jeepchan@jeepchan-X556URK:~$ ssh -A -i ~/.ssh/TickSmithPFE.pem centos@ec2-34-229
-51-157.compute-1.amazonaws.com -N -f -L 127.0.0.1:9083:127.0.0.1:9083
```

Figure 3.5 : Création d'un tunnel SSH

Finalement, il est possible d'exécuter le fichier « .jar » à l'aide d'une commande « Spark-submit » (voir la commande) :

```
jeepchan@jeepchan-X556URK:~$ sudo ./spark-submit  --conf spark.sql.catalogImplementation=hive
 --conf spark.sql.parquet.compression.codec=snappy --packages org.apache.hadoop:hadoop-aws:2.
6.4 --class com.ticksmith.analytics.Analyzer /home/jeepchan/Desktop/PFE/Workspace/TickSmithPF
E/analytics/target/tickAnalytics-jar-with-dependencies.jar -YorL local -dbn pfe -ti hits_t -t
o hits_t_output -f "ask_price-bid_price" -dateB 20170605 -dateE 20170605 -limit 5
```

Figure 3.6 : Lancement avec spark-submit du fichier .jar

3.7 Résultat avec Spark-Shell

Afin de vérifier que le fichier « .jar » est bel et bien exécuté, il est possible de démarrer un « Spark-shell » à l'aide de la commande suivante (voir la commande):

Start spark shell

```
sudo ./spark-shell  --conf spark.sql.catalogImplementation=hive --conf
spark.sql.parquet.compression.codec=snappy --packages org.apache.hadoop:hadoop-aws:2.6.4
```

Figure 3.7 : Démarrage d'un spark-shell

Une fois le « Spark-shell » démarré, il est possible de vérifier si le nouveau « Dataframe » est bel et bien présent à l'aide de la commande « spark.sql("select * from [database].[table]") » (voir la capture d'écran qui suit :

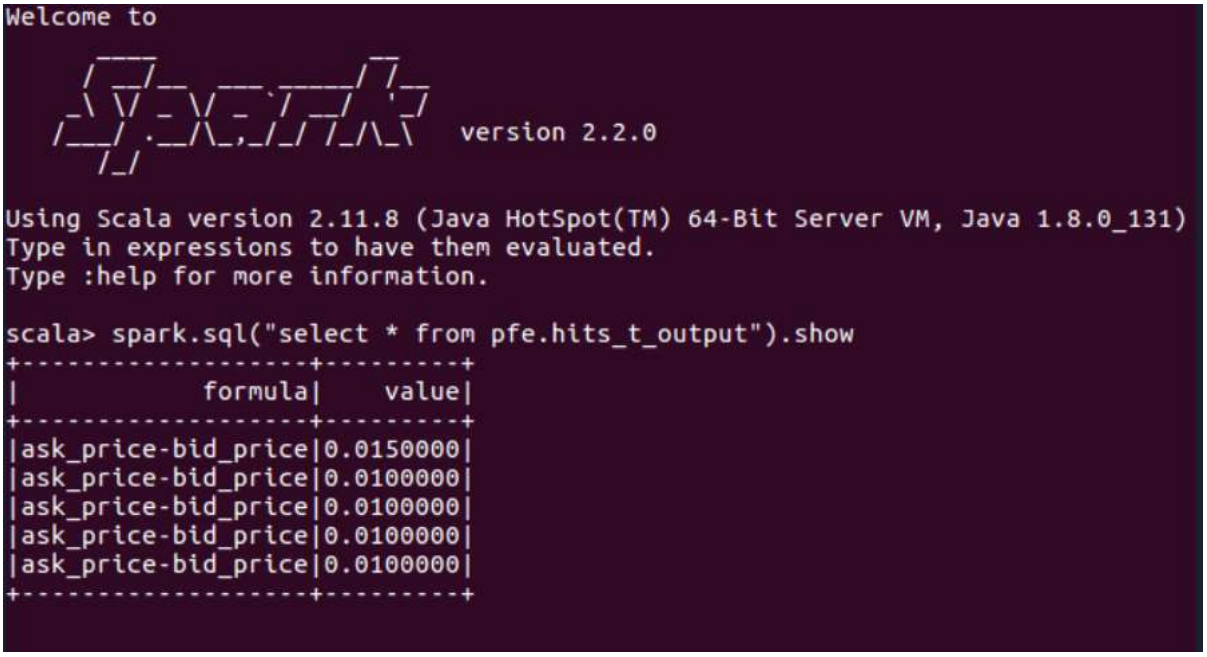


Figure 3.8 : Résultat dans la table destinataire de l'exécution

Cette capture d'écran présente la table de la base de données avec la valeur appropriée selon la formule du parseur.

CHAPITRE 4 CONTRIBUTION DE RICHARD KANTCHIL

4.1 Prototypage de l'interface initiale

Ma contribution première a été de produire la première version de l'interface. Ce prototype va ensuite évoluer vers une version définitive plus raffinée et ergonomique, intégrant la connexion pour exécuter les jobs Livy, le lien vers les tables Impala et la visualisation des Datasets. Les modules afférents à la connexion à la base de données MySQL seront toutefois maintenus ainsi que le mécanisme de gestion de la manipulation des données. Le prototype initial intègre l'outil de génération de formule financière antérieurement conçu dans le projet « TickSmith Analytics » réalisé à l'automne 2016 à partir duquel il était antérieurement impossible de soumettre des formules pour analyse. Le module de construction des formules financières étant indépendant, l'usager ne pouvait donc rien accomplir avec les formules générées. L'interface utilisateur du prototype initial, dans sa dernière version, avait l'apparence ci-après :

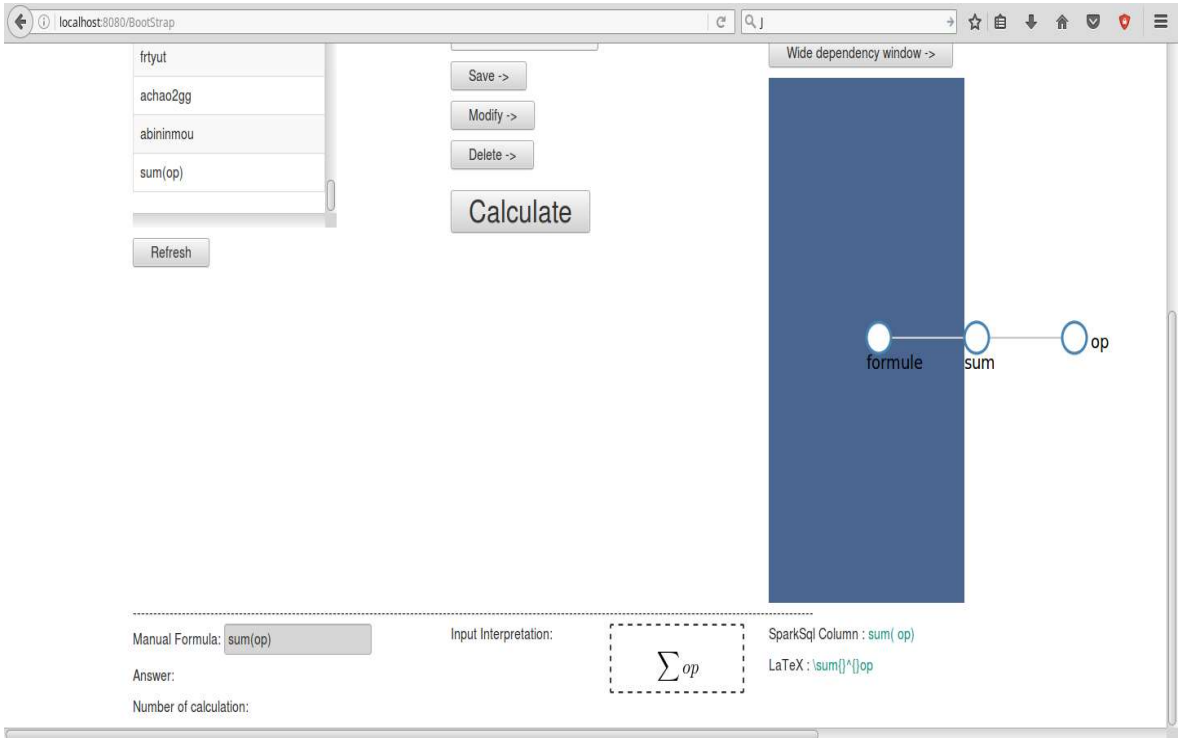


Figure 4.1 : Interface bootstrap

Dès l'ouverture de l'interface de l'utilisateur, l'application va chercher l'ensemble des formules financières disponibles et les affiche dans le tableau figurant dans le côté supérieur gauche de l'aperçu. Lorsqu'un utilisateur veut modifier la formule, il la sélectionne, et la formule sélectionnée apparaît dans la case « Manual Formula ». Il peut ainsi utiliser les boutons « Modify » et « Delete » pour supprimer la formule ou enregistrer une nouvelle version de celle-ci à partir du modèle sélectionné. De même, pour une formule sélectionnée, avec l'intégration du module de génération des formules financières, nous avons la possibilité de visualiser le schéma des dépendances, l'interprétation de la formule, localisé dans « Input Interpretation » ainsi que son équivalent représenté avec Latex. L'utilisateur peut enregistrer dans la base de données une nouvelle formule en l'entrant dans un « Input Manuel » situé au-dessus du bouton « Save » sur lequel il clique par la suite.

4.2 Création de la dynamique de connexion à MySQL sur AWS RDS

L'implémentation initiale offre le stockage des formules financières dans une base de données située sur une instance Amazon RDS (Rational Database Service) suivant un mécanisme dont nous effectuerons plus tard le descriptif.

4.2.1 Amazon Web Service RDS

AWS RDS est l'outil d'Amazon qui permet de configurer et gérer une base de données relationnelle dans le Cloud. Cette option a été retenue parce qu'elle offre une capacité économique et ajustable. Au-delà de l'aspect sécuritaire qu'offre cette option, les quatre avantages suivants ont influé son choix :

1. Facilité d'administration :

   Beaucoup de tâches liées à l'administration d'une base de données, en production, sont prises en charge par ce service. Un ensemble d'outils sont offerts en vue d'effectuer un « monitoring » efficient de la base de données MySQL ;

2. Évolutivité :

   Il est possible de dimensionner les ressources de calcul et de stockage en seulement quelques clics. On n'utilise que l'infrastructure nécessaire à nos besoins, ce qui influe sur le facteur coût ;

3. Coûts :

En plus de l'avantage de l'évolutivité, il est important de mentionner qu'aucune installation n'est nécessaire. De même aucun achat de matériel ou de logiciel n'est requis avec AWS RDS;

4. Disponibilité et durabilité:

   AWS RDS procède à une duplication synchrone des données sur une instance de secours située dans une zone de disponibilité quelque part dans le monde. Ceci offre la garantie de recouvrer les données en cas de catastrophe ou d'incident majeur.

4.2.2 Création de la base de données

L'instance Amazon RDS étant accessible via une console, dénommée « AWS Management Console ». Nous avons exécuté via cette console préalablement installée sur notre poste de travail. Le script de création de la base de données MySQL, appelé PFE, au sein duquel nous avons créé une table nommée « tick_formulas » qui sert à héberger les formules financières. Le script de création de la base de données est présenté dans les minutes du SCRUM, à l'annexe IV, de ce rapport.

4.2.3 Utilisation de JDBC Template

JDBC Template est une librairie « Spring » intégrée à notre projet via la dépendance appropriée intégrée au fichier POM. Le choix de ce « Template » vient du fait qu'il facilite grandement un certain nombre d'actions redondantes dévolues au développeur dans le cadre de la gestion de sa connexion à la base de données. Au nombre de celles-ci on peut mentionner l'ouverture de la connexion, la préparation et l'exécution de la requête (« Statement »), l'itération au travers des résultats (« resultset ») de la requête, la gestion des exceptions et la fermeture de la connexion.

Le processus de la connexion à la base de données a été mis en œuvre au sein de la classe « **DALFormulas.java ».** Après l'importation des classes nécessaires, nous avons créé un « dataSource » auquel nous avons assigné l'URL, la classe du connecteur ainsi que le « UserName » et le « Password » de la base de données MySQL.

À cette étape, une précision est de mise : quoique la base de données à interroger soit sur Amazon, l'URL indiquée est faite selon le procédé suivant :

`dataSource.setUrl("jdbc:mysql://localhost:3306/pfe?useSSL=true")`. Cette particularité vient du fait que la connexion SSH établie en vue de nous connecter à l'instance Amazon RDS génère pour effet que la BD distance est considérée comme hébergée par l'hôte local. Après la gestion de la connexion et la création de l'objet « Template », le reste de la classe contient des méthodes qui nous permettront de procéder à la manipulation des formules en lecture et en écriture. Par exemple, la méthode « `PushFormula` » va permettre de pousser la formule de l'utilisateur dans la base de données :

```
public void PushFormula(String formula) {
	jdbcTemplate.update("insert into tick_formulas (owner_id, name, definition,
	scala_definition,latex_definition)" + " values (1, \"\",?,\"\",\"\")", formula);
}
```

Il faut aussi mentionner que de manière prévisionnelle et dans la perspective de la différentiation des formules selon l'utilisateur qui se connecte au logiciel (notons que cette différentiation n'est pas encore effective à l'étape actuelle), nous avons prévu un ensemble de méthodes qui permettent des manipulations suivant les spécificités de l'utilisateur à l'instar du pendant de « `PushFormula` ». Ceci vise à jouer le même rôle, mais dans la perspective de chaque utilisateur et qui est « `PushFormulaId` » dont la structure est la suivante :

```
public void PushFormulaId(String formula, int id) {
	jdbcTemplate.update("insert into tick_formulas (owner_id, name, definition,
	scala_definition,latex_definition)" + " values (?, \"\",?,\"\",\"\")", id, formula);
}
```

Le reste de la classe pourra être consulté par inspection du code.

4.3 Création de services REST et endpoints pour usage des données MySQL

4.3.1 Principes des services REST

Les services REST constituent un style d'architecture reposant sur le protocole HTTP. Ils offrent l'accès à une ressource par un URI permettant de procéder à diverses opérations supportées nativement par HTTP. Il s'agit des opérations GET, POST, PUT et DELETE permettant respectivement des lectures, écritures, modifications ou suppressions au sein de la ressource.

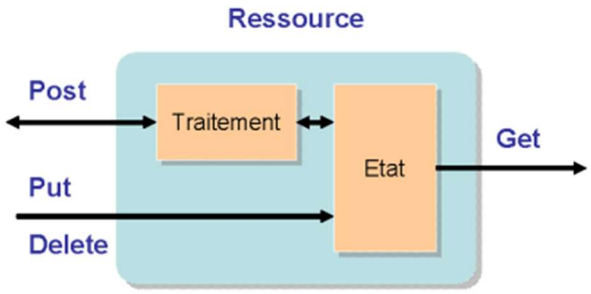


Figure 4.2 : REST

La ressource va être manipulée via l'offre de « end-point » qui constituent des URI établis suivant une hiérarchie convenue par le développeur. Les formats les plus souvent utilisés dans les services REST sont le XML et le JSON. Dans notre cas, nous avons opté pour le JSON du fait de sa convivialité. L'intérêt d'une architecture REST repose sur la simplicité, la lisibilité, et la facilité de test grâce à un simple navigateur. L'autre avantage vient de la démarcation qu'offre l'API entre la ressource et l'utilisateur, qui permet une sécurisation plus accrue de la première.

4.3.2 Implémentation des services d'accès a MySQL

Dans le cadre de la solution proposée, les services REST permettant d'accéder aux formules financières de notre base de données ont été bâtis dans la classe « **MySQLRestController.java** ». Une inspection de cette classe permettra d'obtenir tous les détails de l'implémentation des services. À titre illustratif, nous allons juste décrire le service permettant l'appel des formules de la base de données.

```
@ResponseBody
                @RequestMapping(value = "/formules", method = RequestMethod.GET, produces
                = MediaType.APPLICATION_JSON_VALUE)
                    public List<Formula> getFormulaList () {
                        List<Formula> list = new ArrayList<>()    ;
                        try {
                            DALFormulas getDat = new DALFormulas();
                            list = getDat.getFormulas();
                        } catch (DataAccessException e) {
                            Formula error = new Formula ();
                            error.setNom("Echec fatal");
                            list.add(error);
```

```
            }
            return list;
        }
```

L'extrait de code présenté ci-dessus décrit que la concaténation de l'adresse de l'hôte sur laquelle roule la solution, à l'aide de « /formules » qui retourne au format JSON la liste des formules contenues dans la base de données par l'appel des méthodes de la classe « DALFormulas » décrites plus haut.

4.4 Gestion de la dynamique d'affichage et de la modification des formules

Les interfaces ont été créés avec la technologie HTML5, CSS3 et BootStrap. Le rafraîchissement d'interfaces a été effectué à l'aide de JavaScript. J'ai proposé la bibliothèque « JQuery » afin de gérer l'appel aux services REST ainsi que l'affichage, la suppression et la mise à jour des formules.

4.5 Défis

La maîtrise des outils de développement a été ardue dans le délai imparti pour ce projet. Cela a entraîné de la lenteur lors de la réalisation et de l'implémentation.

4.6 Enseignements

Ce projet m'a permis de m'imprégner de la dynamique des projets d'entreprise, à échéance fixe, ainsi que de me familiariser avec un certain nombre de technologies qui m'étaient totalement inconnues, telles que : Spring, Maven, REST, AMAZON RDS, JQuery et JSON.

CHAPITRE 5 CONTRIBUTION DE YVES MILLETTE

5.1 Contexte de la contribution

Dans un contexte où TickSmith désirait être en mesure de permettre aux usagers de leur application de créer des fils d'exécution Spark personnalisés nous devions trouver une façon d'exécuter des traitements semblables mais avec des paramètres qui varient sur une grappe de serveurs EMR. Ces variations consistent en des changements de paramètres semblables à ceux qui ont été présentés au chapitre 3. Le but ultime était d'être en mesure de lancer à distance des fils d'exécution paramétrisés à partir d'un serveur distant qui héberge l'application client permettant à l'usager de lancer ces traitements sans avoir directement accès à la grappe de serveurs.

5.2 Choix de la solution à implémenter

Diverses solutions logicielles ont été envisagées afin de répondre aux exigences de TickSmith en matière de traitement Spark à distance.

5.2.1 Compilation et exécution d'un fichier « .jar » pour chaque traitement

Cette option fut considérée en premier lieu lors de nos discussions avec TickSmith et consiste en la compilation locale programmatique sur le serveur web d'un fichier binaire comprenant du code qui aurait été dynamiquement modifié pour paramétriser le traitement selon les entrées effectuées par l'usager de l'application web. Il aurait fallu implémenter à la fois la solution logicielle permettant d'effectuer la compilation du code en plus de gérer le téléversement du code compilé vers le serveur principal de la grappe de traitement EMR avec «Spark-submit». Cela se serait avéré une tâche ardue et il aurait été difficile de prendre en charge la gestion du statut de l'exécution une fois le traitement lancé. Pour ces raisons cette solution n'a pas été retenue.

5.2.2 Utilisation de Cloudera Livy

Cloudera Livy (maintenant Apache Livy) est un serveur http qui peut être installé sur le nœud principal d'une grappe de serveurs EMR. Ce logiciel expose un API http permettant de lancer à la fois des fils d'exécution Spark (de la même façon qu'un ««spark-submit) en plus de permettre le lancement de sessions interactives où il est possible de soumettre par appel http des lignes de code uniques à exécuter séquentiellement. La complétude de l'API fournit par le logiciel en plus du fait qu'il est possible d'appeler cet API pour obtenir le statut et les détails d'exécution des traitements lancés sur la grappe de serveurs le rendent très avantageux pour notre implémentation. Le fait que la solution ait été initiée par Cloudera et qu'il s'agit maintenant d'un projet incubé par la fondation Apache Software nous porte à croire que la solution sera maintenue à long terme et continuera d'évoluer. Cette solution est celle qui a été retenue pour la réalisation du projet de TickSmith.

5.2.3 Utilisation de Spark-Jobserver

Lors de nos recherches nous avons également considéré le projet « open-source » Spark-Jobserver qui offrait des fonctionnalités très semblables à Livy. Toutefois l'API était moins complet que celui qui est offert par Livy et nous avions un degré de confiance moindre dans la viabilité à long terme de ce projet et dans les chances qu'il soit maintenu à long terme.

5.3 Technologies adjacentes pour l'implémentation

Livy ayant été choisi comme pièce centrale pour la gestion de la soumission des traitements à effectuer sur les données nous avons dû déterminer quelles technologies mettre en place afin de supporter ce choix. Nous étions contraints à utiliser le cadriciel Spring-boot puisqu'il était déjà utilisé par l'interface nous ayant été fournie par TickSmith. Nous devions donc compléter l'interaction entre Spring et Livy en ajoutant un client http dans la solution de services Spring boot nous ayant été fournie. Pour ce faire, j'ai choisi d'utiliser Retrofit2 puisqu'il s'agît d'un logiciel dont je connaissais déjà les rouages. Il permet les appels synchrones et asynchrones mais de façon plus importante il permet la conversion de formats de données JSON en objets

Java par réflexion à l'aide du convertisseur Google Gson. Cela me permet de ne jamais gérer la création de chaînes de caractères JSON et de manipuler des objets uniquement. Les autres technologies employées en support à l'intégration de Livy étaient Bootstrap et jQuery qui géraient respectivement l'interface usager responsive et les appels AJAX ainsi que la modification du contenu de la page web dépendamment du résultat des appels.

5.4 Détails d'implémentation

5.4.1 Architecture globale

L'architecture globale de la solution mise en œuvre est composée de quatre nœuds principaux. Une interface web contient du code JavaScript qui permet lors de certains événements, tel un clic de bouton, de soumettre une requête http et obtenir une réponse qui modifie par la suite conditionnellement le contenu de la page. Ces appels sont envoyés à un serveur qui contient l'application de services Spring déployée. Ces services contiennent le client http basé sur Retrofit2 qui appelle les Livy. Puisque Livy se trouve sur le nœud racine de la grappe de serveur celui-ci est en mesure de démarrer le traitement. Ce traitement crée une table résultante dans la base de données distribuée Impala. Les services Spring peuvent à leur tour consulter cette base de données pour renvoyer de l'information à l'interface client. La structure de ces interactions peut être visualisée à la figure 5.1.

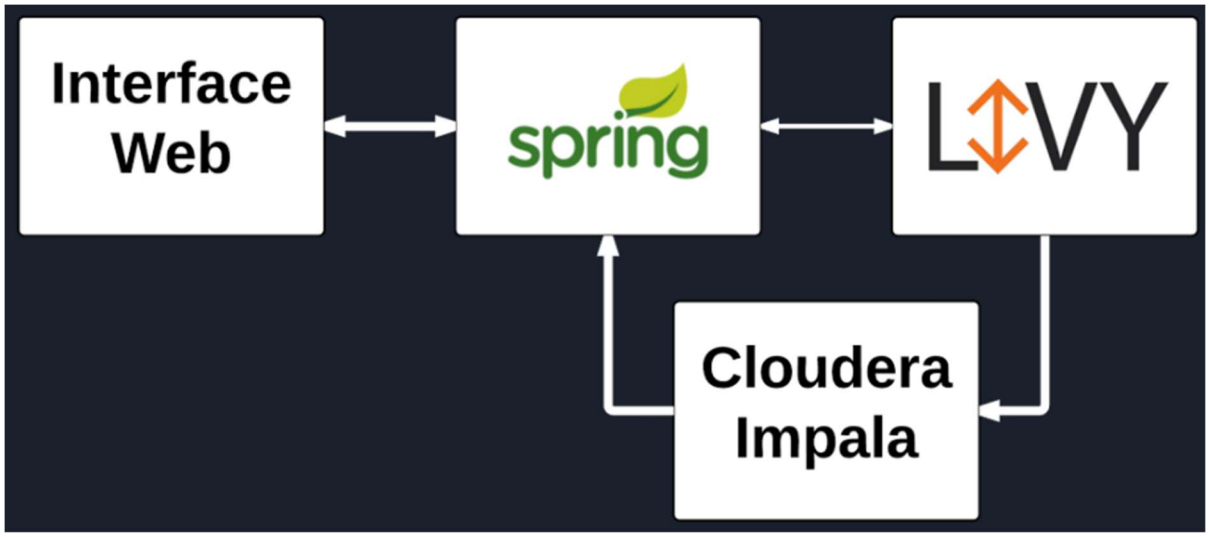


Figure 5.1 : Architecture de l'implémentation de Livy

5.4.2 Client http

Retrofit2 permet de définir les points d'accès d'un API que l'on désire consommer dans une seule classe interface. Des annotations permettent de spécifier le type de requête et de configurer le format de l'URL et tout paramètre qui s'y retrouve. On peut associer un nom de méthode interne à notre application pour accéder à la ressource spécifiée et définir le type d'objet que nous désirons créer par réflexion avec le résultat de la requête. La figure 5.2 présente quelques-unes des méthodes implémentées pour consommer les points d'accès fournis par Livy pour la gestion des fils d'exécution Spark.

```
@POST("batches")
Call<Batch> runBatch(@Body CreateBatch batch);


@GET("batches/{batchId}/log")
Call<BatchLog> getBatchLog(@Path("batchId") int batchId);


@GET("batches/{batchId}/state")
Call<BatchState> getBatchState(@Path("batchId") int batchId);
```

Figure 5.2 : Définition de méthodes dans l'interface pour le client Livy

Une fois qu'une interface est définie pour tous les points d'accès du service à consommer il est possible d'initier un client à l'aide de cette interface en spécifiant que nous désirons convertir le résultat des requêtes à l'aide du convertisseur Google Gson. Le code pour ce faire peut être visualisé à la figure 5.3.

```
private void initRetrofitClient(){
   retrofit = new Retrofit.Builder()
           .baseUrl(livyUrl)
           .addConverterFactory(GsonConverterFactory.create())
           .build();
   service = retrofit.create(LivyHttpService.class);
}
```

Figure 5.3 : Création du client avec l'interface désirée et le convertisseur approprié

Le code qui est appelé lorsque l'interface de l'usager fait un appel AJAX vers le service Spring peut ensuite appeler le service à l'aide d'une méthode semblable à celle de la figure 5.4. On peut y noter que la gestion d'exception renvoie un objet « null » si la requête n'a pu être

effectuée. Cela pourrait être amélioré afin de minimalement transmettre le message d'erreur à la couche qui appelle cette méthode soit habituellement le point d'accès qui est exposé à l'interface client. Ceci est discutable puisqu'il pourrait être préférable d'obfusquer ce message et l'enregistrer à l'interne. Ce choix est donc laissé à la discrétion de TickSmith dans ses implémentations futures.

```
public BatchLog getBatchLog(int batchId){
    try{
        return service.getBatchLog(batchId).execute().body();
    }catch(IOException e){
        System.out.println(e.getMessage());
        return null;
    }
}
```

Figure 5.4 : Méthode appelée pour obtenir le registre d'un fil d'exécution sur Livy

Les objets qui sont créés afin de construire les requêtes et créer des objets sont conséquents de la documentation officielle de l'API de Livy. On peut voir cette relation en comparant les figures 5.5 et 5.6 qui représentent l'objet Java utilisé dans l'application Spring pour ce service et la documentation du service en soit.

```
package com.ticksmith.livy.LivyHttpTypes;

import java.util.ArrayList;
import java.util.Map;

public class CreateBatch {
    public String file;
    public String className;
    public ArrayList<String> args;
    public String driverMemory;
    public String name;
    public Map<String,String> conf;
}
```

Figure 5.5 : Objet représentatif du fil d'exécution à créer.

POST /batches

Request Body

name	description	type
file	File containing the application to execute	path (required)
proxyUser	User to impersonate when running the job	string
className	Application Java/Spark main class	string
args	Command line arguments for the application	list of strings
jars	jars to be used in this session	List of string
pyFiles	Python files to be used in this session	List of string
files	files to be used in this session	List of string
driverMemory	Amount of memory to use for the driver process	string
driverCores	Number of cores to use for the driver process	int
executorMemory	Amount of memory to use per executor process	string
executorCores	Number of cores to use for each executor	int
numExecutors	Number of executors to launch for this session	int
archives	Archives to be used in this session	List of string
queue	The name of the YARN queue to which submitted	string
name	The name of this session	string
conf	Spark configuration properties	Map of key=val

Figure 5.6 : Documentation de Livy pour le point d'accès de démarrage d'exécution

Lorsque l'interface client soumet une demande de démarrage d'exécution Spark au service Spring approprié celui-ci crée donc un objet de type « CreateBatch » avant de l'utiliser pour appeler la méthode « createBatch() » du client http qui transmet alors la requête vers Livy.

5.4.3 Interface client

Il avait d'abord été entendu que Richard devait réaliser l'interface client du projet avec l'aide de Jean-Philippe. Après plusieurs semaines qui ont résulté dans une interface peu satisfaisante j'ai pris en main cette section du travail afin d'intégrer dans une interface consolidée la soumissions de fil d'exécution à partir d'une formule donnée. L'outil de visualisation de

création de formules réalisé dans le projet précédent a pu être repris et intégré dans cette nouvelle interface. L'interface en soit est présentée à la figure 5.7.

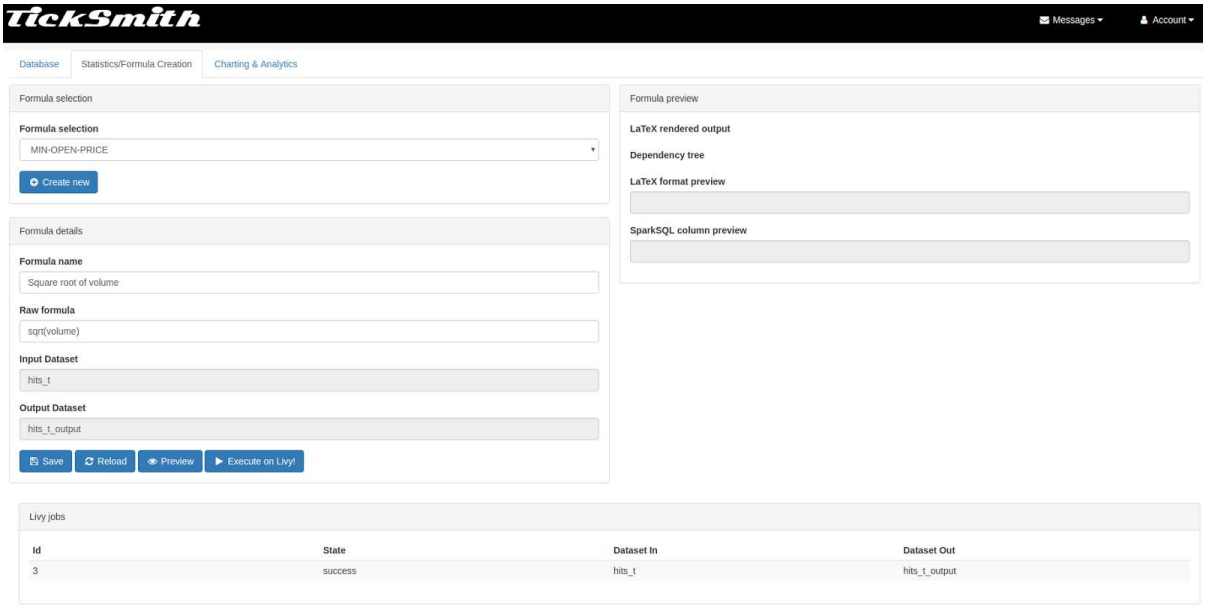


Figure 5.7 : Interface client améliorée

Cette interface comporte plusieurs appels AJAX notamment pour la création de formule et la soumission de formules à l'outil de prévisualisation mais de façon plus importante elle comporte un appel AJAX récursif permettant de suivre le statut d'exécution d'un fil d'exécution Spark ayant été démarré par Livy. Le code concerné est visible dans la figure 5.8.

```
function pollLivyState(id , dataSetIn, dataSetOut){

    fetchUrl = '/livy/getbatchstate/' + id + '/';
    $.ajax({headers: {'Accept': 'application/json',
            'Content-Type': 'application/json'
        },type: 'get',url: fetchUrl, dataType: 'json',
        success: function (data, textStatus, jqXHR) {
            if(data.success === "true"){
                ProcessJobsList(data);
                if(data.state !== "success" && data.state !== "dead"){
                    setTimeout(pollLivyState(id,dataSetIn,dataSetOut),3000);
                }
            }
        }
    });
}
```

Figure 5.8 : Fonction exécutant un appel AJAX récursif

5.5 Autres tâches effectuées

5.5.1 Gestion de projet

J'ai pris l'initiative en début de projet de mettre sur pied le dépôt de code sur GitHub et d'y intégrer le module Slack permettant de notifier l'équipe de toute contribution au code ou aux tâches à effectuer sur le projet. J'avais également pris le temps de créer des tâches et des jalons pour le travail à effectuer dans le ««kanban board de github. Cela s'est toutefois avéré infructueux puisque les autres membres de l'équipe ne consultaient pas les tâches ou n'en effectuaient pas le suivi. Vu le manque de participation cette initiative a été abandonnée avec l'accord de notre superviseur de projet Alain April. Certaines des tâches décrites peuvent toutefois être visualisées à la figure 5.9.

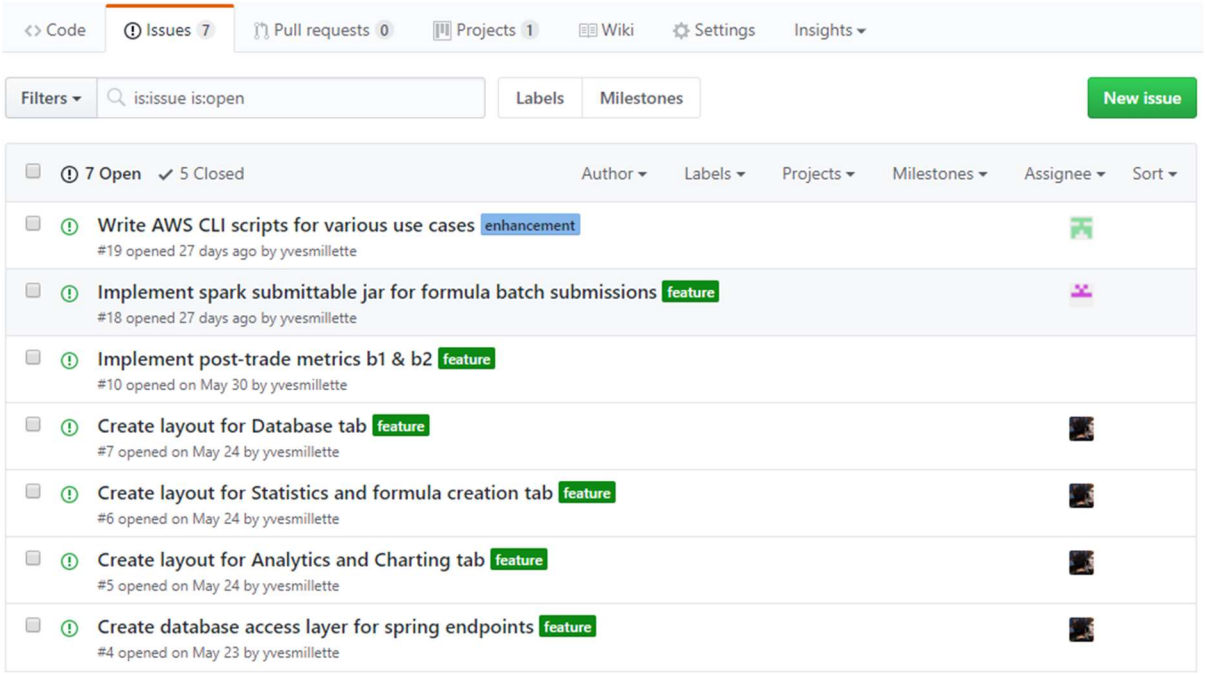


Figure 5.9 : Tâches pour le projet sur GitHub

5.5.2 Architecture globale du projet

J'ai pris le temps également lors d'une des premières semaines du projet d'élaborer un diagramme représentant toutes les composantes du projet. La figure 5.10 contient ce diagramme. Les composants en rouge sont les composants dont nous devions assurer le développement tandis que les verts étaient fournis. Les composants en jaune étaient partiellement fournis ou devaient être configurés.

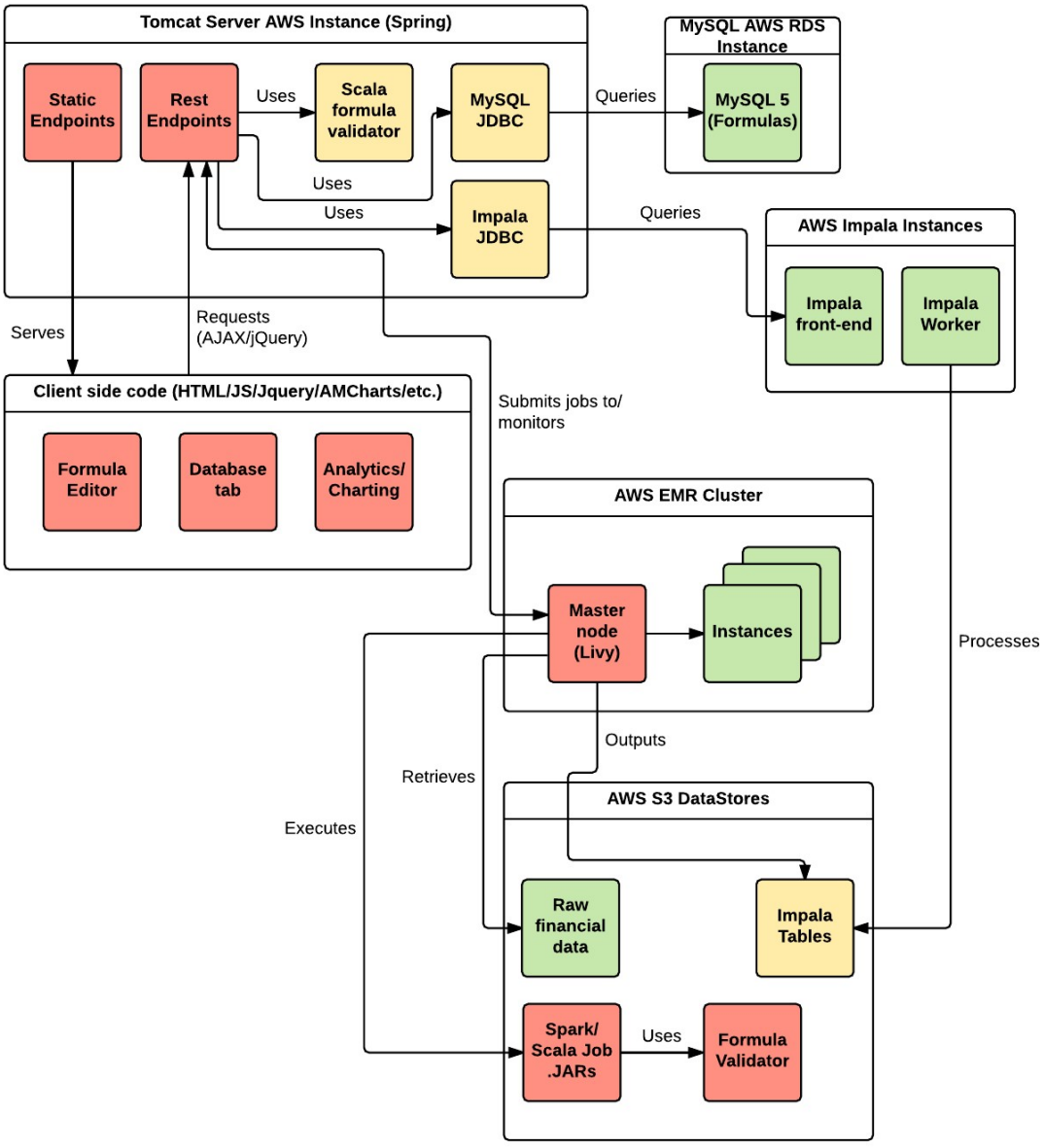


Figure 5.10 : Architecture globale du projet

5.6 Pistes d'amélioration

Le travail réalisé pour TickSmith demeure incomplet. Il serait possible d'améliorer la gestion des erreurs lors d'appels http vers Livy. Il faudrait déterminer quoi faire en cas d'erreur et savoir si on veut exposer à l'usager la cause ou simplement l'enregistrer et l'envoyer à un administrateur chez TickSmith. Il reste du travail considérable à faire au niveau de l'interface usager pour mieux supporter la soumission et le suivi de statut avec Livy. Pour le moment le statut change mais rien n'indique à l'utilisateur que la tâche est encore en progression. Il est également impossible pour le moment de consulter le registre ou log d'exécution de Livy à partir de l'interface. Un autre gros problème est le fait que les fils d'exécution lancés ne sont pas enregistrés dans une base de données. Cela fait en sorte que si on rafraîchit la page, toute l'information sur les traitements précédemment lancés est perdue. Il faudrait enregistrer cette information dans MySQL et maintenir un historique des traitements lancée ainsi que des tables dans lesquelles les données résultantes ont été enregistrées.

LISTE DE RÉFÉRENCES

(Grenier-Vallée, 2016) Rapport de projet de Philippe Grenier-Vallée

BIBLIOGRAPHIE

Grenier-Vallée, P. (2016). *Big data, Small Numbers*. (Rapport de projet de fin d'études, École de technologie supérieure, Montréal, Québec, Canada). Fourni par TickSmith.

Apache Foundation (2017) Apache Spark – Lightning-fast cluster computing, Reperé à https://spark.apache.org/

Apache Foundation (2017) Apache Hadoop, Reperé à http://hadoop.apache.org/

Apache Foundation (2017) Apache Hadoop, Reperé à http://avro.apache.org/

Apache Foundation (2017) Apache Impala, Reperé à https://impala.apache.org/

Apache Foundation (2017) Apache Livy, Reperé à https://livy.incubator.apache.org/

Amazon Inc. (2017) AWS Documentation, Reperé à https://aws.amazon.com/documentation/

Twitter (2017) Twitter Bootstrap, Reperé à http://getbootstrap.com/

TickSmith Corp (2017) Ticksmith, Reperé à www.ticksmith.com

ANNEXE I

CONCEPTION INITIALE DE TICKSMITH

Voir le document intitulé « Stats Engine Portal v 0.6 » inclut dans la remise.

ANNEXE II

AUTRES LIVRABLES

- Code source inclut dans la remise dans le fichier «TickSmithPFE-master.zip »
- Document de proposition de projet sous le nom de fichier «GTI_LOG_795_Proposition.pdf» dans la remise.
- Diapositives de la présentation orale finale dans la remise sous le nom de fichier «PresentationFinale.pdf»

ANNEXE III

SCRUMS POUR LE PROJET

SCRUM DU MARDI 8 AOÛT 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - Tony Bussières, Ticksmith
  - Patrick Cardinal, ÉTS
  - ~~Alain April, ÉTS~~
  - Luiz Fernando Dos Santos Pereira, Ticksmith
  - Yves Millette, ÉTS
  - ~~Maxime Paul Dégarie, ÉTS - Appel Conférence~~
  - Kantchil Dam-Hissey, ÉTS
  - ~~Jean-Philippe Chan, ÉTS~~

Effort estimé:

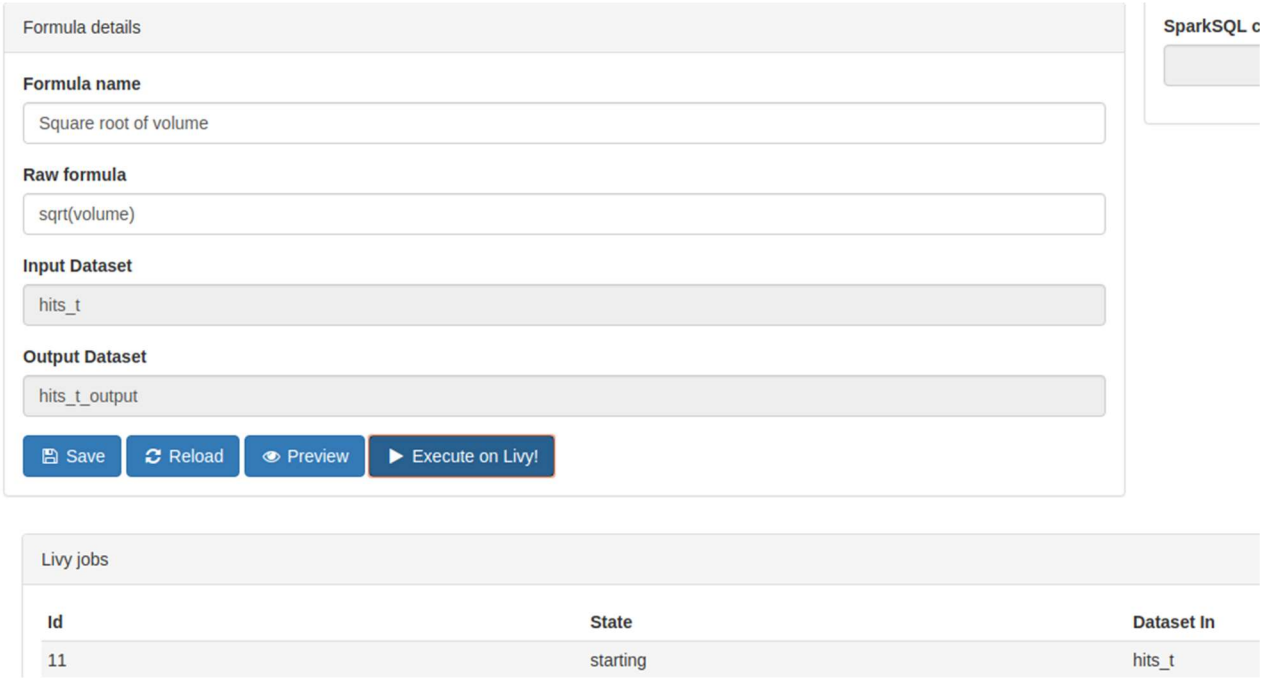
Ce qui a été fait:

<u>Yves</u>

- Implémentation de la soumissions de job Livy à partir de l'interface usager TickAnalytics.
- Débogage du .jar d'analytics de JP pour faire fonctionner les submit Livy
- Implémentation du polling sur les job Livy.
- Correction de certains bogues dans le client/api qui fait la liaison navigateur <-> spring <-> Livy qui avait été principalement élaboré dans les semaines du 27 juin au 18 juillet.

Statuts Livy dans le UI:

(au démarrage)



(en exécution)

(après la complétion de la batch)

Il faudrait penser à ajouter un indicateur de progrès (spin-wheel) pour que l'usager sache qu'il se passe toujours quelque chose. Il faudrait également stocker les batch dans MySQL afin de pouvoir les recharger à l'ouverture de la page ce qui n'est actuellement pas le cas.

Certains paramètres sont hardcodés tels que les dates de début et fin ainsi qu'une limite de 100 résultats. Cela est adéquat pour une démonstration mais nécessitera de l'optimisation de la part de TickSmith. Il serait également intéressant de rendre disponible les logs de l'application pour des cas de développement. De nombreux cas de validation doivent également être pris en compte pour ce qui est saisi par l'utilisateur afin d'éviter de se fier au cluster et l'application spark pour la détection d'erreurs.

La job est soumise via le javascript suivant:

```
$("#runLivyJobBtn").click(function(){

 formulaStr = $("#editFormulaRaw").val();
 dataInStr = $("#inputData").val();
   dataOutStr = $("#outputData").val();

 var json = {
      formula: formulaStr,
      datasetIn: dataInStr,
      datasetOut: dataOutStr
   }

jQuery.ajax({
      headers: {
         'Accept': 'application/json',
         'Content-Type': 'application/json'
      },
      type: 'post',
      url: '/livy/submitformula/',
      dataType: 'json',
      data: JSON.stringify(json),
      success: function (data, textStatus, jqXHR){
```

```
            if(data.success === "true"){
                pollLivyState(data.id, dataInStr, dataOutStr);
            }
        }
    });
});
```

Lorsque soumis avec succès le callback pollLivyState est appelé qui poll le statut de la job aux 3 secondes jusqu'à temps que la job soit un succès ou qu'elle ait échoué:

```
function pollLivyState(id , dataSetIn, dataSetOut){

 fetchUrl = '/livy/getbatchstate/' + id + '/';
 $.ajax({
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
    type: 'get',
    url: fetchUrl,
    dataType: 'json',
    success: function (data, textStatus, jqXHR) {
        if(data.success === "true"){
            $("#jobsList")
                .empty()
                .append("<td>"+id+"</td>")
                .append("<td>"+data.state+"</td>")
                .append("<td>"+dataSetIn+"</td>")
                .append("<td>"+dataSetOut+"</td>");
```

```
            if(data.state !== "success" && data.state !== "dead"){
                setTimeout(pollLivyState(id,dataSetIn,dataSetOut),3000);
            }
        }
    }
  });
}
```

Les données retournées par les endpoints de Livy sont les suivantes :

Soumission d'une batch:

Appel à /batches (POST)

```
{
 "file" : "s3://ticksmith-pfe/jarfile/ta.jar",
 "className" : "com.ticksmith.analytics.Analyzer",
 "args" : [  "--yarnOrLocal",
   "yarn",
   "--length",
   "00300000000000000",
   "--databaseName",
   "pfe",
   "--tableInput",
   "hits_t",
   "--tableOutput",
   "hits_t_output",
   "--formule",
   "sqrt(volume)",
   "--dateBegin",
   "20170605",
   "--dateEnd",
   "20170605",
```

```
  "--limitTable",
  "1000"
  ],
 "driverMemory" :"2g",
 "name" :"ticksmithtest",
 "conf" : {
  "spark.sql.catalogImplementation":"hive",
  "spark.sql.parquet.compression.codec":"snappy"
  }
 }
}
```

Data retourné:

```
{
  "id": 10,
  "state": "starting",
  "appId": null,
  "appInfo": {
     "driverLogUrl": null,
     "sparkUiUrl": null
  },
  "log": []
}
```

Appel à (batches/{id}) (GET)

Data retourné:

```
{
  "id": 10,
  "state": "success",
  "appId": "application_1502195124796_0010",
```

```
  "appInfo": {
     "driverLogUrl": null,
     "sparkUiUrl": "http://ip-172-31-16-20.ec2.internal:20888/proxy/application_1502195124796_0010/"
  },
  "log": [
     "\t diagnostics: N/A",
     "\t ApplicationMaster host: N/A",
     "\t ApplicationMaster RPC port: -1",
     "\t queue: default",
     "\t start time: 1502203882415",
     "\t final status: UNDEFINED",
     "\t tracking URL: http://ip-172-31-16-20.ec2.internal:20888/proxy/application_1502195124796_0010/",
     "\t user: hadoop",
     "17/08/08 14:51:22 INFO ShutdownHookManager: Shutdown hook called",
     "17/08/08 14:51:22 INFO ShutdownHookManager: Deleting directory /mnt/tmp/spark-68ff36e7-8323-4b1b-bde7-1b260d1022da"
  ]
}
```


Appel à (batches/{id}/log) (GET)
Data retourné:


```
{
  "id": 10,
  "from": 0,
  "total": 35,
  "log": [
     "Warning: Skip remote jar s3://ticksmith-pfe/jarfile/ta.jar.",
```

"17/08/08 14:56:59 WARN NativeCodeLoader: Unable to load native-hadoop library for your pla
classes where applicable",

"17/08/08 14:57:01 INFO RMProxy: Connecting to ResourceManager at ip-172-31-16-20.ec2.inter

"17/08/08 14:57:01 INFO Client: Requesting a new application from cluster with 2 NodeManagers"

"17/08/08 14:57:01 INFO Client: Verifying our application has not requested more than the maxim
the cluster (11520 MB per container)",

"17/08/08 14:57:01 INFO Client: Will allocate AM container, with 2432 MB memory including 384 M

"17/08/08 14:57:01 INFO Client: Setting up container launch context for our AM",

"17/08/08 14:57:01 INFO Client: Setting up the launch environment for our AM container",

"17/08/08 14:57:01 INFO Client: Preparing resources for our AM container",

"17/08/08 14:57:03 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling b
under SPARK_HOME.",

"17/08/08 14:57:06 INFO Client: Uploading resource file:/mnt/tmp/spark-a
c7f32d9e2781/__spark_libs__8174853157481569585.zip ->
20.ec2.internal:8020/user/hadoop/.sparkStaging/application_1502195124796_0011/__spark_libs__817

"17/08/08 14:57:09 INFO Client: Uploading resource s3://ticksmith-pfe/jarfile/ta.jar
20.ec2.internal:8020/user/hadoop/.sparkStaging/application_1502195124796_0011/ta.jar",

"17/08/08 14:57:09 INFO S3NativeFileSystem: Opening 's3://ticksmith-pfe/jarfile/ta.jar' for reading'

"17/08/08 14:57:13 INFO Client: Uploading resource file:/etc/spark/conf/hive-site.xml
20.ec2.internal:8020/user/hadoop/.sparkStaging/application_1502195124796_0011/hive-site.xml",

"17/08/08 14:57:13 INFO Client: Uploading resource file:/mnt/tmp/spark-a
c7f32d9e2781/__spark_conf__8329583807999223725.zip ->
20.ec2.internal:8020/user/hadoop/.sparkStaging/application_1502195124796_0011/__spark_conf__.zi

"17/08/08 14:57:13 INFO SecurityManager: Changing view acls to: hadoop",

"17/08/08 14:57:13 INFO SecurityManager: Changing modify acls to: hadoop",

"17/08/08 14:57:13 INFO SecurityManager: Changing view acls groups to: ",

"17/08/08 14:57:13 INFO SecurityManager: Changing modify acls groups to: ",

"17/08/08 14:57:13 INFO SecurityManager: SecurityManager: authentication disabled; ui acls d
permissions: Set(hadoop); groups with view permissions: Set(); users with modify permissions: Set(ha
permissions: Set()",

"17/08/08 14:57:13 INFO Client: Submitting application application_1502195124796_0011 to Reso

"17/08/08 14:57:13 INFO YarnClientImpl: Submitted application application_1502195124796_001

```
    "17/08/08 14:57:13 INFO Client: Application report for application_1502195124796_0011 (state: A
    "17/08/08 14:57:13 INFO Client: ",
    "\t client token: N/A",
    "\t diagnostics: N/A",
    "\t ApplicationMaster host: N/A",
    "\t ApplicationMaster RPC port: -1",
    "\t queue: default",
    "\t start time: 1502204233757",
    "\t final status: UNDEFINED",
    "\t tracking URL: http://ip-172-31-16-20.ec2.internal:20888/proxy/application_1502195124796_001
    "\t user: hadoop",
    "17/08/08 14:57:13 INFO ShutdownHookManager: Shutdown hook called",
    "17/08/08 14:57:13 INFO ShutdownHookManager: Deleting directory /mnt/tmp/spark-a
c7f32d9e2781"
  ]
}
```


Une collection postman a été créé pour permettre à TickSmith de tester les endpoints à l'avenir elle a été postée dans slack et sera disponible dans le dossier google drive du projet.

Éléments bloquants :

- Beaucoup de temps perdu à faire fonctionner le .jar d'analytics qui était non-fonctionnel. Le fait que JP n'avait pas poussé son code sur lequel il avait travaillé avec Luiz jeudi a été problématique pour l'intégration.
- Cela fait en sorte que du retard a été accumulé et que le temps est limité pour travailler sur les sections Database et Charting de l'application. Je vais me concentrer sur la section Database afin qu'on puisse minimalement voir le data résultant pour une démo. Si cela s'avère impossible une démo avec impala-shell sera suffisante.

Kantchil

https://docs.google.com/presentation/d/10sinAQS5Ylf4qwbW5Hws6YDBkOUbpvryrjGAYl6dHl4/edit#slide=id.g242a7af832_1_10

Jean-Philippe

Maxime

- Version de Livy utilisé est 0.3.0, Yves a réussi a l'utiliser sans problèmes.
- Pour Sentry:
  - Utiliser ce tutoriel: https://docs.sentry.io/server/installation/python/
  - Dans mon cas, j'ai utilisé la méthode avec Python.
  - Il faut s'assurer d'installer pip et redis-server, mais c'était avec PostGreSQL.
  - mysql ne serait plus supporté avec Sentry "...but of critical note, we are no longer supporting MySQL for installations." (https://blog.sentry.io/2016/01/08/sentry-8-on-premise.html)

SCRUM DU MARDI 1 AOÛT 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - Tony Bussières, Ticksmith
  - Patrick Cardinal, ÉTS
  - ~~Alain April, ÉTS~~
  - Luiz Fernando Dos Santos Pereira, Ticksmith
  - Yves Millette, ÉTS
  - ~~Maxime Paul-Dégarie, ÉTS - Appel Conférence~~
  - Kantchil Dam-Hissey, ÉTS
  - Jean-Philippe Chan, ÉTS

Effort estimé:

Ce qui a été fait:

Yves

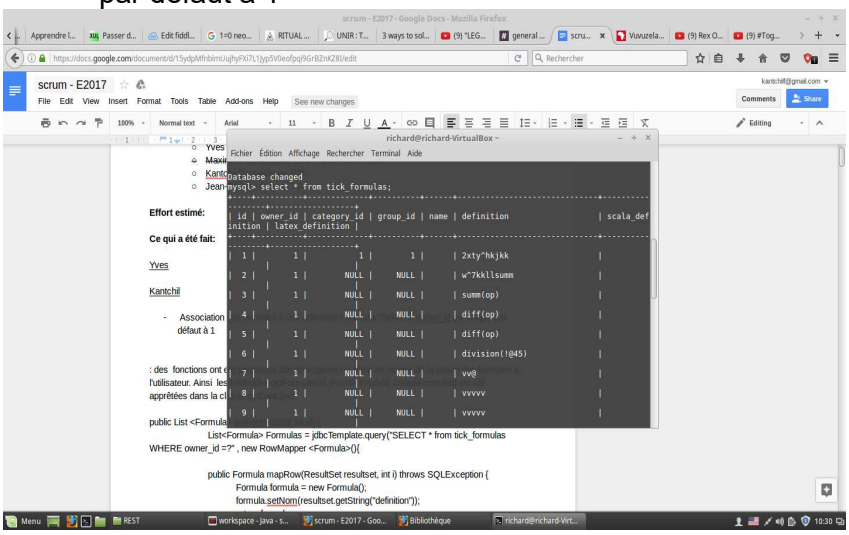
- Tests avec le UI et AMCharts
- Recherche sur le plugin https://datatables.net/ pour jQuery pour voir si je peux faire une implémentation server-side pour charger du data à partir d'Impala. Curieux de

savoir ce que TickSmith préfère/utilise déjà pour charger le data sous forme de tableaux dans leurs pages web.

- Recherche et documentation sur le workflow d'exécution d'un spark-submit, enregistrement de dataframes dans des database, etc. Un petit walkthrough pour me sauver du temps pourrait être bénéfique.
- Avancement limité cette semaine dû à un horaire très chargé par d'autres travaux (fin de session) Rattrapage en vue durant la semaine à venir.

Kantchil

- Association des formules à des utilisateurs car pour l'heure le owner_id est unique par défaut à 1



: des fonctions ont ete preparees afin de supporter la mise en oeuvre de la liaison des formules à l'utilisateur. Ainsi les méthodes getFormulasId, PushFormulaId, DeleteFormulaId ont été apprêtées dans la classe getData.java

```
public List <Formula> getFormulasId( int id) {
                List<Formula> Formulas = jdbcTemplate.query("SELECT * from tick_formulas
WHERE owner_id =?" , new RowMapper <Formula>(){

                public Formula mapRow(ResultSet resultset, int i) throws SQLException {
                        Formula formula = new Formula();
                        formula.setNom(resultset.getString("definition"));
                        return formula;
```

```
                }
                }, id);
        return Formulas;
       }

public void PushFormulaId(String formula, int id) {
        jdbcTemplate.update("insert into tick_formulas (owner_id, name, definition, scala_definition,latex_definition)"
                        + " values (?,  \"\",?,\"\",\"\")", id, formula);
  }

public void DeleteFormulaId (String formula, int id) {
        jdbcTemplate.update("DELETE FROM tick_formulas WHERE definition=? and owner_id =? ", formula, id);
  }
```

Lors de l'évolution de l'interface vers une version incluant l'identification de l'utilisateur lors de sa connexion, les différentes fonctions ci-dessus réalisées vont être mises en oeuvre dans la classe StaticController.java. Ainsi donc suivant le cas, les formule seront affichées, ajoutées et supprimées respectivement en fonction de l'id du user connecté qui sera recueilli par javascript a partir de l'interface.

```
@ResponseBody
  @RequestMapping(value = "/formules", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
  public List <Formula> getFormulaList () {
        ----------
list = getDat.getFormulasId();
au lieu de  list = getDat.getFormulas();
----------

          }
  @ResponseBody
```

```
  @RequestMapping(value = "/insertion/formules/{formula}", method = RequestMethod.GET,
produces = MediaType.APPLICATION_JSON_VALUE)
  public List <Formula> insertFormula (@PathVariable("formula") String formula) {
        List<Formula> list =  new ArrayList<Formula>()    ;
        ----------
list = getDat.PushFormulaId();
au lieu de  list = getDat.PushFormula();
        ----------
  }

  @ResponseBody
  @RequestMapping(value = "/suppression/formules/{formula}", method =
RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
  public List <Formula> deleteFormula (@PathVariable("formula") String formula) {
        List<Formula> list =  new ArrayList<Formula>()    ;
        ----------
list = getDat.DeleteFormulaId();
au lieu de  list = getDat.DeleteFormula();
        ----------
  }
```

- Les modifications ont été poussées sur la branche interface 1.3.
- Une segmentation des tâches de la présentation orale puis éventuellement de la rédaction du rapport doit être envisagée dans un cours terme

Elements bloquants :

- Pas pu finaliser la liaison des formules avec les datasets

Jean-Philippe Chan

- Analyser.scala lit maintenant les formules que l'on va mettre, au lieu de faire "*".

```
val p = new TickSmithFormulaParser
var formuleSQL = p.toSql(params.formule)
println(formuleSQL);


//sqlContext.sql("select count(*) from pfe.hits_t").show()
var tableSelect = params.databaseName + "." + params.tableInput
var condition = " WHERE yyyymmdd between " + params.dateBegin + " and " + params.dateEnd
println("select "+ formuleSQL +" FROM "+tableSelect+condition)
//println("select count(*) from "+tableSelect+condition)

//Testing
//val dfEntry = sqlContext.sql("select * FROM "+tableSelect+condition +" limit "+params.limitTable)
//var dataframe: DataFrame = sqlContext.sql("select "+ formuleSQL +" FROM "+tableSelect+condition +" limit "+params.limitTab
val dfEntry = sqlContext.sql("select "+ formuleSQL +" FROM "+tableSelect+condition +" limit "+params.limitTable)
dfEntry.show()
```

- Les colonnes du hits_t sont tous mis sur le fichier TickSmithFormulaParser. Donc, le fichier jar (sur S3) est capable de lire les colonnes du fichier comme formule.

```
[localhost:21000] > describe hits_t;
Query: describe hits_t
+-------------------+-----------------+-------------------------+
| name              | type            | comment                 |
+-------------------+-----------------+-------------------------+
| ts                | timestamp       | Inferred from Parquet file. |
| line_number       | bigint          | Inferred from Parquet file. |
| line_type         | string          | Inferred from Parquet file. |
| ticker            | string          | Inferred from Parquet file. |
| last_price        | decimal(20,7)   | Inferred from Parquet file. |

| volume            | bigint          | Inferred from Parquet file. |
| trade_ref         | string          | Inferred from Parquet file. |
| buyer_id          | string          | Inferred from Parquet file. |
| seller_id         | string          | Inferred from Parquet file. |
| bid_price         | decimal(20,7)   | Inferred from Parquet file. |

| bid_size          | int             | Inferred from Parquet file. |
| ask_price         | decimal(20,7)   | Inferred from Parquet file. |
| ask_size          | int             | Inferred from Parquet file. |
| trade_attribute   | string          | Inferred from Parquet file. |
| cross_type        | string          | Inferred from Parquet file. |

| short_exempt      | string          | Inferred from Parquet file. |
| halted            | string          | Inferred from Parquet file. |
| listing_market    | string          | Inferred from Parquet file. |
| source            | string          | Inferred from Parquet file. |
| trade_initiator_side | string       | Inferred from Parquet file. |
```

```
def _cols: Parser[String] =
    "ts" ^^ {
      case "ts" => {
        val node: NodeElement = new NodeElement("ts", "Colonne de la table xx
        _tree.subtree += node
        _tree = node
        val parsed = "ts"
        _tree = _tree.father
        parsed
      }
    }|
    "line_number" ^^ {
      case "line_number" => {
        val node: NodeElement = new NodeElement("line_number", "Colonne de la
        _tree.subtree += node
        _tree = node
        val parsed = "line_number"
        _tree = _tree.father
        parsed
      }
    }|
    "line_type" ^^ {
      case "line_type" => {
        val node: NodeElement = new NodeElement("line_type", "Colonne de la ta
        _tree.subtree += node
        _tree = node
        val parsed = "line_type"
        _tree = _tree.father
        parsed
      }
    }|
    "ticker" ^^ {
      case "ticker" => {
        val node: NodeElement = new NodeElement("ticker", "Colonne de la table
        _tree.subtree += node
        _tree = node
        val parsed = "ticker"
        _tree = _tree.father
        parsed
      }
    }|
    "last_price" ^^ {
```

Elements bloquants :

- Il faut que j'écris le code pour éliminer les valeurs "null" ou vide.

Maxime

- Développement du script pour démarrer un cluster Livy sur AWS EMR (il est sur Google Drive pour le moment sous EMR/scripts):

```
maxime@Maxime-T500:~/Desktop$ sudo ./livyDeploy.sh
[sudo] password for maxime:
Cluster ID: j-1R63KSMQ6CFW0
livyDNS=ec2-174-129-145-103.compute-1.amazonaws.com
```

- Développement du script pour éteindre toutes les instances qui sont allumées (aussi sur Google Drive pour le moment):

```
maxime@Maxime-T500:~/Desktop$ ./shutdownInstances.sh
{
    "StoppingInstances": [
        {
            "InstanceId": "i-0214e4072afebcb9a",
            "PreviousState": {
                "Name": "running",
                "Code": 16
            },
            "CurrentState": {
                "Name": "stopping",
                "Code": 64
            }
        }
    ]
}
{
    "StoppingInstances": [
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-06b94b1c0deedbb28",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```

- Pas beaucoup d'avancement au niveau de Sentry, plus de recherche qu'autre chose.
- Bloquant: Au niveau des scripts, pour faire un tunnel SSH pour accéder au port de Livy a partir de l'externe, il faudrait avoir un autre port que 22 d'ouvert. Je ne sais pas si c'est vraiment nécessaire qu'il soit disponible de l'externe.

Plan d'actions:

- Richard / Yves (integrer REST services).
- Richard : commencer rapport
- Yves :
    - 1 - Integrer Livy submit
    - 2 - Job status
    - 3 - Amchart (simple). Assumons que la table de destination est toujours la meme.

- JP : Finaliser Jar (fixer lorsque null)
- Maxime : Mise a jour Livy. Continuer recherche sur Sentry (documenter ce qui a ete fait)

SCRUM DU MARDI 25 JUILLET 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
    - ~~Tony Bussières, Ticksmith~~
    - Patrick Cardinal, ÉTS
    - ~~A~~lain April, ÉTS
    - ~~Marc-André Hétu, Ticksmith~~
    - Luiz Fernando Dos Santos Pereira, Ticksmith
    - Yves Millette, ÉTS
    - ~~Maxime Paul-Dégarie, ÉTS - Appel Conférence~~
    - Kantchil Dam-Hissey, ÉTS
    - Jean-Philippe Chan, ÉTS

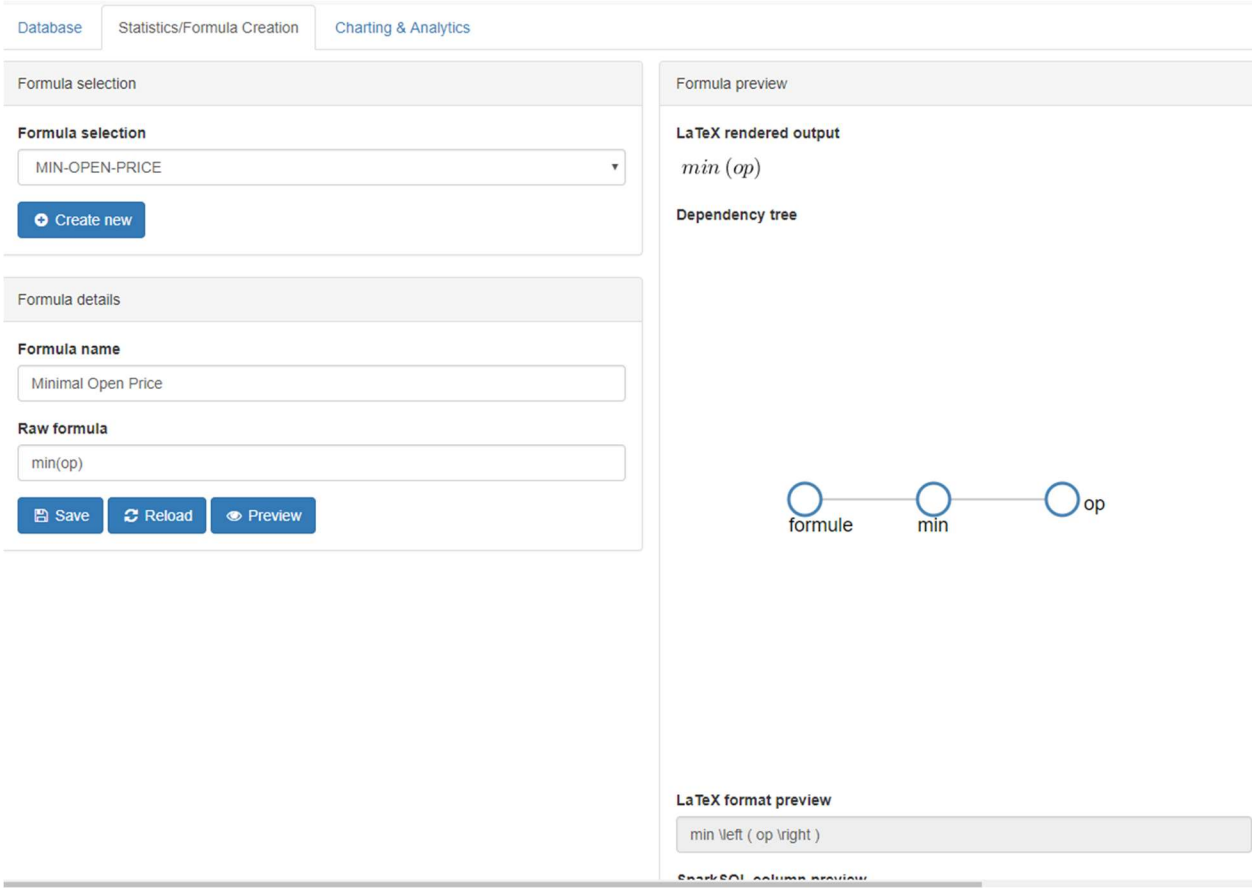
Effort estimé:

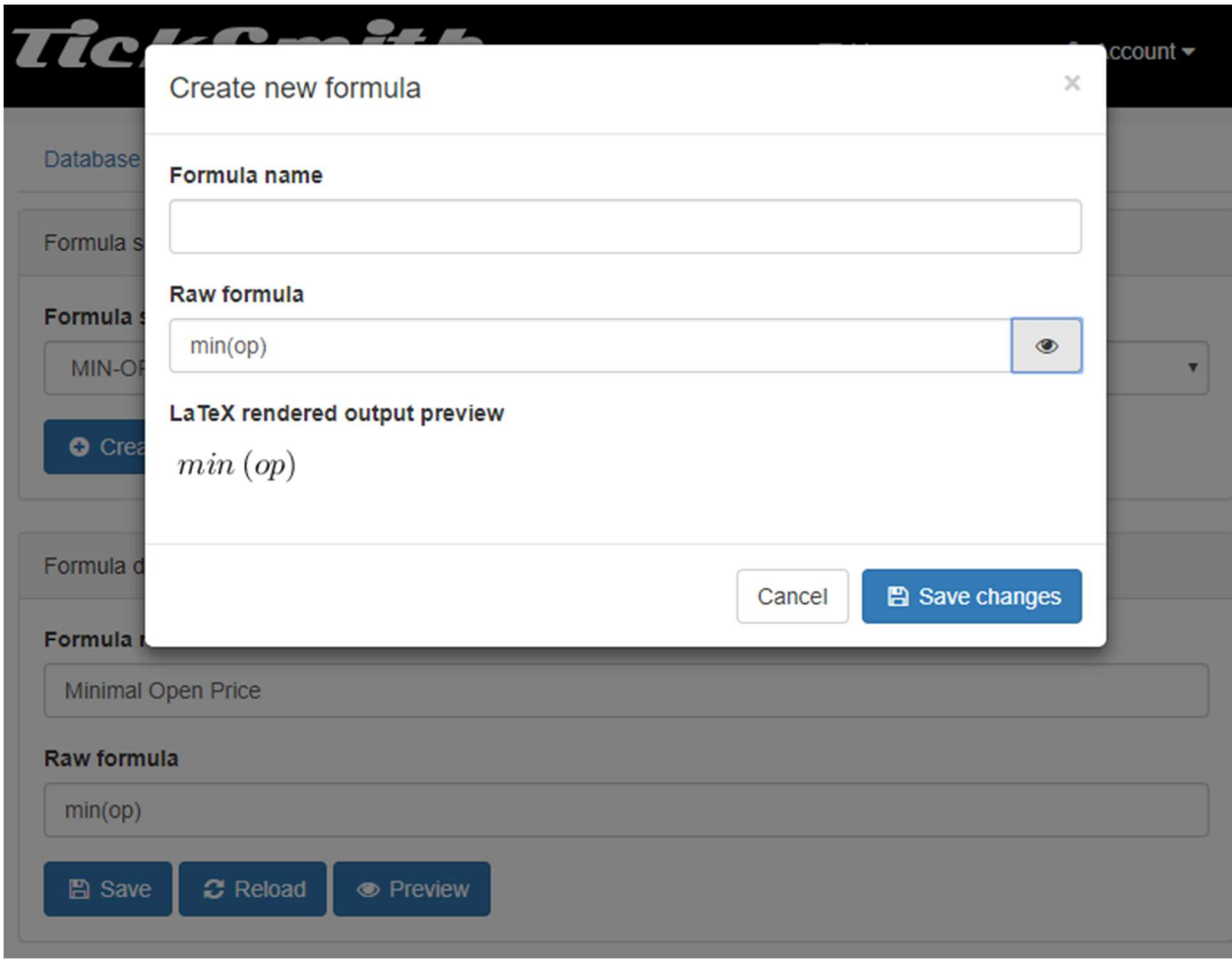
Ce qui a été fait:

<u>Yves</u>

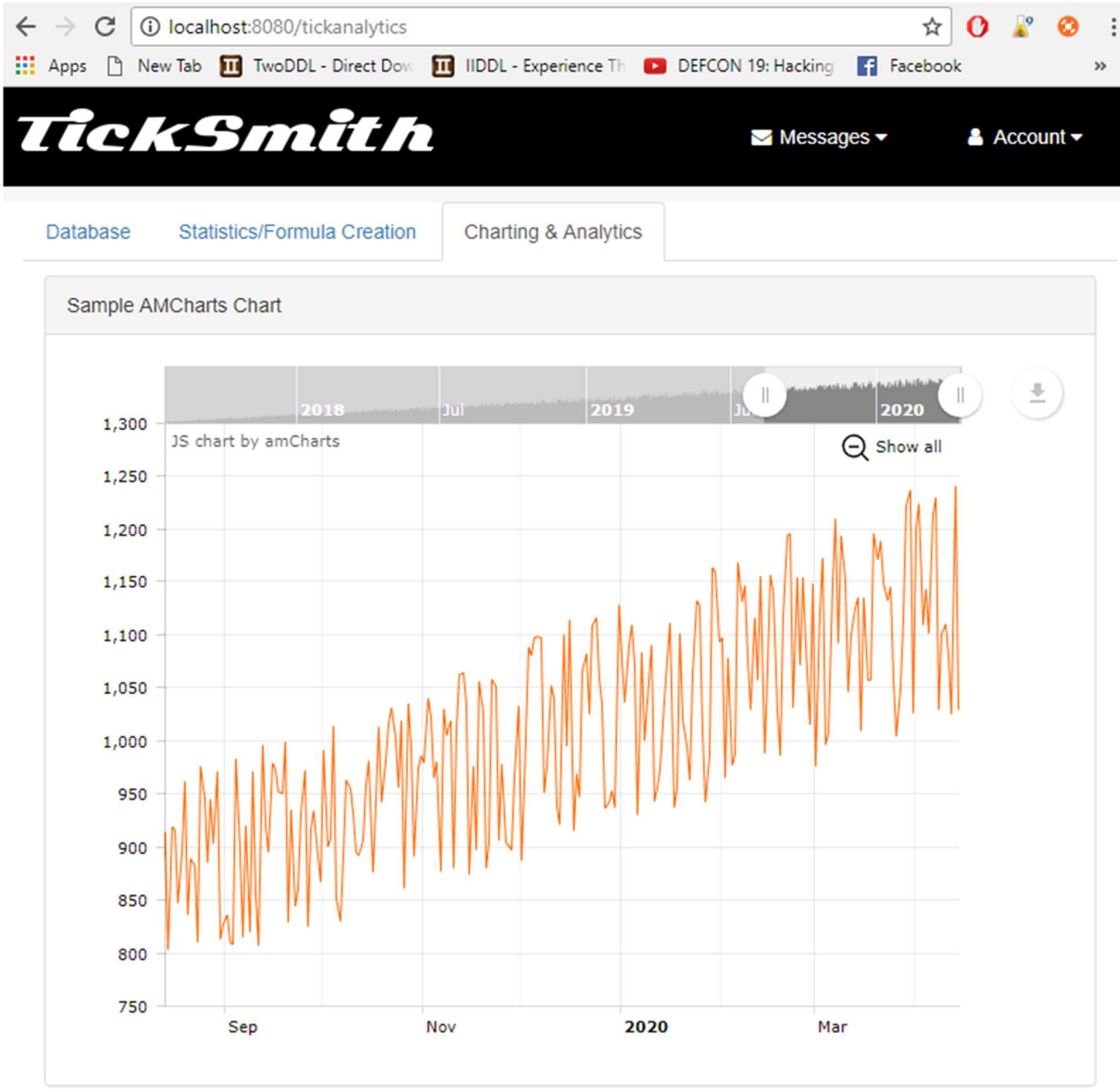
- Déploiement d'un repo maven 2 pour rendre accessible par maven au projet les dépendances propriétaires requises pour le driver JDBC d'impala à partir de la dernière version des packages sur cloudera.

- http://13.82.217.171:8081/#browse/browse/assets:ticksmithpfe
- Intégration de ces dépendances au projet de services:
  - ```
    <!-- custom repo cloudera impala dependencies -->
    <dependency>
        <groupId>com.cloudera.impala.jdbc</groupId>
        <artifactId>hive_metastore</artifactId>
        <version>2.5.38</version>
    </dependency>
    <dependency>
        <groupId>com.cloudera.impala.jdbc</groupId>
        <artifactId>hive_service</artifactId>
        <version>2.5.38</version>
    </dependency>
    <dependency>
        <groupId>com.cloudera.impala.jdbc</groupId>
        <artifactId>ImpalaJDBC41</artifactId>
        <version>2.5.38</version>
    </dependency>
    <dependency>
        <groupId>com.cloudera.impala.jdbc</groupId>
        <artifactId>ql</artifactId>
        <version>2.5.38</version>
    </dependency>
    <dependency>
        <groupId>com.cloudera.impala.jdbc</groupId>
        <artifactId>TCLIServiceClient</artifactId>
        <version>2.5.38</version>
    </dependency>
    ```
  - ```
    <repositories>
        <repository>
            <id>ticksmithpfe</id>
            <url>http://13.82.217.171:8081/repository/ticksmithpfe/</url>
            <name>ImpalaJDBC custom repo</name>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
    ```
- Création du tabbed-based layout pour le UI du projet

- 
Database
Statistics/Formula Creation
Charting & Analytics
Formula selection
Formula selection
MIN-OPEN-PRICE
Create new
Formula details
Formula name
Minimal Open Price
Raw formula
min(op)
Save
Reload
Preview
Formula preview
LaTeX rendered output
min (op)
Dependency tree
formule
min
op
LaTeX format preview
min \left (op \right)

- 

- https://github.com/yvesmillette/TickSmithPFE/issues/19
- Disponible dans master sur localhost:8080/tickanalytics
- Intégration de AMCharts3 au projet



-
- Mise à jour des issues sur github.

Richard

-suppression de la possibilité d'ajouter des formules déjà existantes

- apurement des anciens chemins absolus dans l'appel des services REST a partir de l'ensemble des scripts js.

Exemple : `var tableau = document.getElementById("tableau");`

```
ajaxGet(" /formules", function (reponse) {
-----------------
}); au lieu de
var tableau = document.getElementById("tableau");
ajaxGet(" localhost:8080/formules", function (reponse) {
-----------------
});
```

- Changement des noms de classes hors du contexte de l'application. Suppression de la classe Student remplacee par la classe Formula

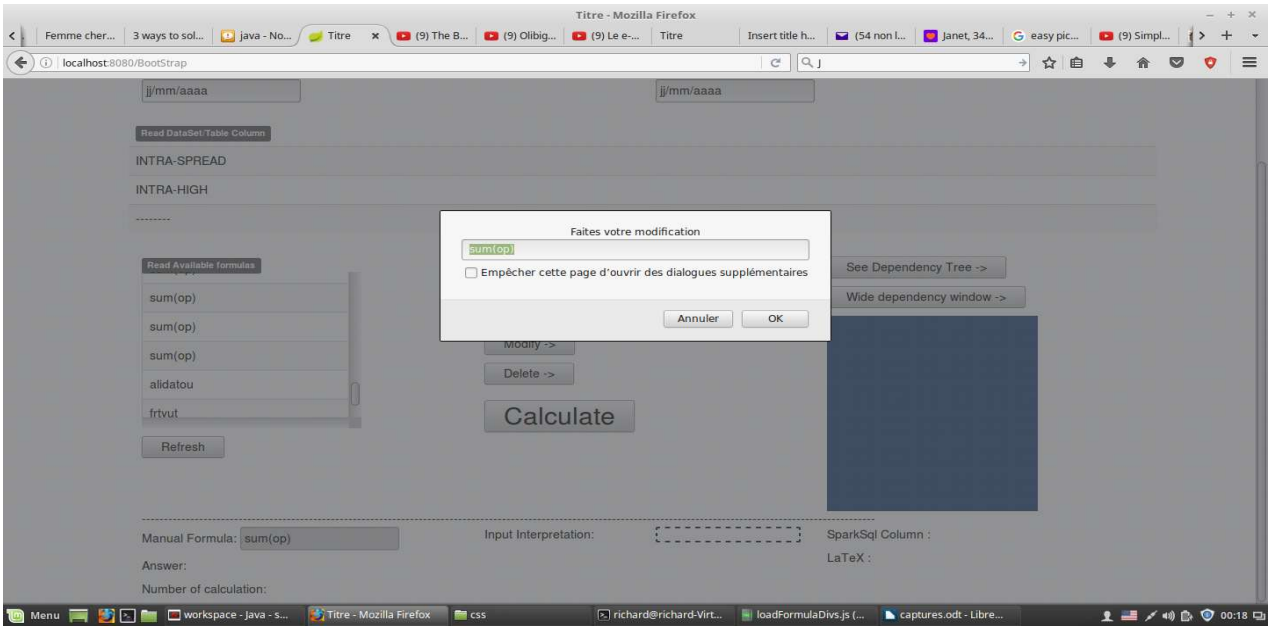
- Ajout de la fonctionnalite de suppression d'une formule et creation d'un service REST dédié a cet effet et fonctionnant a partir des classes GetData et StaticController. Nous avons la methode

public void DeleteFormula(String formula) de la classe GetData

puis le service REST suivant dans la classe StaticController:

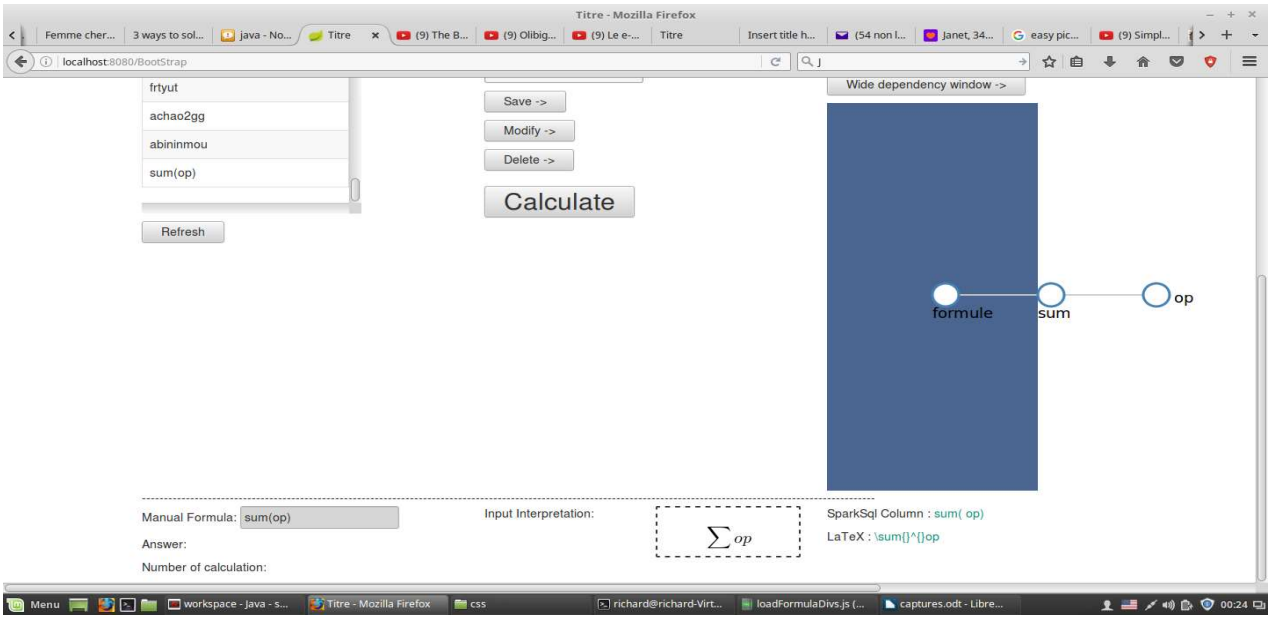
```
@ResponseBody
	@RequestMapping(value = "/suppression/formules/{formula}", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
	public List <Formula> deleteFormula (@PathVariable("formula") String formula) {
		List<Formula> list = new ArrayList<Formula>();
		try {
			getData getDat = new getData ();
			getDat.DeleteFormula(formula);
		}
	catch (DataAccessException e) {
		Formula error = new Formula ();
		error.setNom("Echec fatal");
		list.add(error);
		System.out.print( e.getMessage());
	}

		return list;
```

- ajout de la fonctionnalité de modification d'une formule existante :



modification du script insert js en conséquence.

- intégration des modules préexistant de conversion en Latex des formules à l'interface en creation. Pour le rendu suivant laissant en outre la possibilité à l'utilisateur d'afficher la formule dans un onglet dédié



- mise a jour sur gitHub de la branche testInterface1.3 sur laquelle le code est disponible.

Jean-Philippe Chan

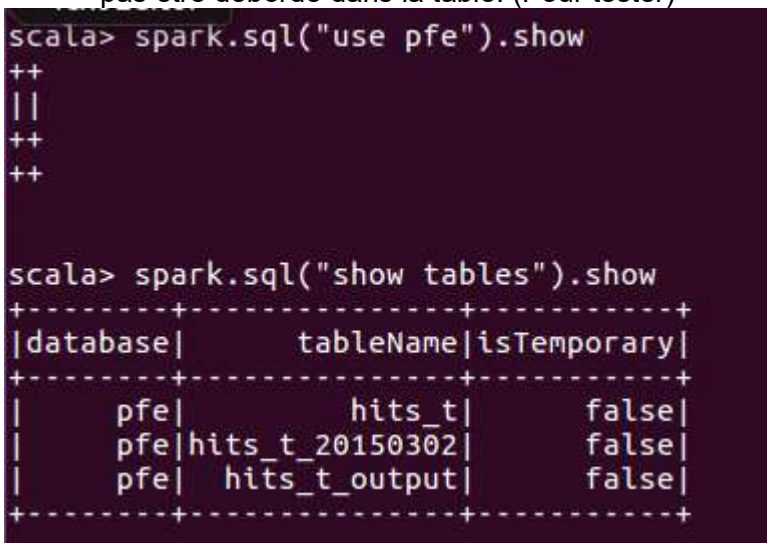
- Le jar est complété.
    - Tous les paramètres se sont appelées. (avec exception→voir éléments bloquants)

```
def main(args: Array[String]): Unit = {
  val params = new Params
    if (!args.isEmpty) {
        new JCommander(params, args.toArray: _*)
        println("Results: ")
        println(params.yarnOrLocal)
        println(params.sliceLength)
        println(params.databaseName)
        println(params.tableInput)
        println(params.tableOutput)
        println(params.formule)
        println(params.dateBegin)
        println(params.dateEnd)
    }
    else{
      println("Missing arguments");
      System.exit(1);
    }

  sliceLength = params.sliceLength

  var conf = new SparkConf()
        .setAppName("avro_loader")
        //.setMaster("local[*]") //if condition: Local ==> Remove setMaster for non-local
        .set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

    if(params.yarnOrLocal == "local"){
      println("Local mode");
      conf.setMaster("local[*]")
    }
```

```
val sc = new SparkContext(conf);
sc.setLogLevel("WARN")
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val now = Calendar.getInstance.getTimeInMillis;

//Already loading the dataframe
sqlContext.sql("show databases").show()

val p = new TickSmithFormulaParser
var formuleSQL = p.toSql(params.formule)
println(formuleSQL);


//sqlContext.sql("select count(*) from pfe.hits_t").show()
var tableSelect = params.databaseName + "." + params.tableInput
var condition = " WHERE yyyymmdd between " + params.dateBegin + " and " + params.dateEnd
println("select "+ formuleSQL +" FROM "+tableSelect+condition)
//println("select count(*) from "+tableSelect+condition)

//val dfEntry = sqlContext.sql("select "+ formuleSQL +" FROM "+tableSelect+condition)

//Testing
val dfEntry = sqlContext.sql("select * FROM "+tableSelect+condition +" limit 20")
dfEntry.show()
```

```
//TODO write the df in a impala table (that is given in parameter)
//Optionally write to csv file / parquet for debugging and local work

        //finalDF.sort(stockCol, sliceCol, analyticsCol).coalesce(1).write.option("header", "true").option("codec", "bzip2").csv(returnPath)
        //finalDF.coalesce(1).write.parquet(returnPath)
        //finalDF.write.parquet(returnPath)

        //Exemple de sauvegarde d'un table en spark: datafame.write.mode(org.apache.spark.sql.SaveMode.Append).saveAsTable("bd.nom_de_la_tab
        //finalDF.write.mode(org.apache.spark.sql.SaveMode.Append).saveAsTable(params.databaseName+"."+params.tableOutput)
        dfEntry.write.mode(org.apache.spark.sql.SaveMode.Append).saveAsTable(params.databaseName+"."+params.tableOutput)
        val then = Calendar.getInstance.getTimeInMillis
        println("This took : " + (then - now) + "ms")
    }
```

- L'exécution spark.sql est maintenant capable de lire la table du database selon les paramètres et de le sauvegarder.
    - La table sauvegardée est "pfe.hits_t_output". J'ai limité à 20 pour ne pas être débordé dans la table. (Pour tester)



```
scala> spark.sql("use pfe").show
++
||
++
++


scala> spark.sql("show tables").show
+--------+---------------+-----------+
|database|      tableName|isTemporary|
+--------+---------------+-----------+
|     pfe|         hits_t|      false|
|     pfe|hits_t_20150302|      false|
|     pfe|  hits_t_output|      false|
+--------+---------------+-----------+
```

- Le jar a été mis sur **S3**. On est capable de l'exécuter avec spark-submit.

```
bash: syntax error near unexpected token `('
jeepchan@jeepchan-X556URK:~/Software/spark/bin$ sudo ./spark-submit  --conf spark.sql.catalogImplementation=hive --co
spark.sql.parquet.compression.codec=snappy --packages org.apache.hadoop:hadoop-aws:2.6.4 --class com.ticksmith.analyt
.Analyzer /home/jeepchan/Desktop/PFE/Workspace/TickSmithPFE/analytics/target/tickAnalytics-jar-with-dependencies.jar
rL local -l "00300000000000000" -dbn pfe -ti hits_t -to hits_t_output -f "sqrt(op)" -dateB 20170605 -dateE 20170605
```

<u>Éléments bloquants:</u>

- Le paramètre "formule" m'a posé un problème à mon exécution, disant que c'est un problème de Regex.

```
sqrt(op)
     ^

select  string matching regex `-?(\d+(\.\d*)?|\d*\.\d+)([eE][+-]?\d+)?[fFdD]?' expected but `o' found

sqrt(op)
     ^ FROM pfe.hits_t WHERE yyyymmdd between 20170605 and 20170605
```

  - En ce moment, j'ai mis l'appel sur SQL à "*" au lieu de la formule.
- Tentative de rouler sur le cluster via "setupscript.sh" amène à un échec. Relative avec le paramètre "formule".
- import com.ticksmith.TickSmithFormulaParser → Problème avec "maven install"
  - Temporairement fait un copier-coller du fichier "TickSmithFormulaParser.scala" vers le package "com.ticksmith.analytics".

```
[ERROR] /home/jeepchan/Desktop/PFE/Workspace/TickSmithPFE/analytics/src/com/ticksmith/analytics/analyzer.scala:35: error: object TickSmithFormulaParser is not a member of package c
[ERROR] import com.ticksmith.TickSmithFormulaParser
[ERROR]            ^
[ERROR] /home/jeepchan/Desktop/PFE/Workspace/TickSmithPFE/analytics/src/com/ticksmith/analytics/analyzer.scala:169: error: not found: type TickSmithFormulaParser
[ERROR]         val p = new TickSmithFormulaParser
[ERROR]                     ^
[ERROR] two errors found
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] services ........................................... SUCCESS [ 20.219 s]
[INFO] analytics .......................................... FAILURE [  8.220 s]
[INFO] TickSmithPFE ....................................... SKIPPED
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 28.557 s
[INFO] Finished at: 2017-07-25T12:42:32-04:00
[INFO] Final Memory: 68M/999M
[INFO] ------------------------------------------------------------------------
```

<u>Maxime</u>

- Essaie de ce tutoriel, sur Ubuntu par contre (@Tony: C'est mieux que le postgre?): https://supportex.net/blog/2014/10/installing-sentry-centos-6-5/
- J'aimerais avoir les informations de connexion pour la db Impala/mysql pour tester.
- Rempli le document d'évaluation des pairs

Plan d'actions:

- Yves:
  - Voir avec Richard les procédures pour sauvegarder les formules dans la base de données MySQL.
  - associer les formules avec les datasets
- Richard:
  - Associer les formules à des utilisateurs / dataset

- Maxime:
  - Voir https://github.com/yvesmillette/TickSmithPFE/issues/19
  - Avancement avec Sentry/Impala/MySQL
- JP:
  - (URGENT) Rendre disponible sur git le code du jar ayant été réalisé pour la job spark-submit.
  - Regarder / Modifier le code du Scala parser ajoutant des fonctionnalités.

SCRUM DU MARDI 18 JUILLET 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - Tony Bussières, Ticksmith
  - Patrick Cardinal, ÉTS
  - ~~Alain April, ÉTS~~
  - ~~Marc-André Hétu, Ticksmith~~
  - Luiz Fernando Dos Santos Pereira, Ticksmith
  - Yves Millette, ÉTS
  - ~~Maxime Paul-Dégarie, ÉTS - Appel Conférence~~
  - Kantchil Dam-Hissey, ÉTS
  - Jean-Philippe Chan, ÉTS

Effort estimé:

Ce qui a été fait:

<u>Yves</u>

- Rencontre avec Jean-Philippe ce matin à 10h avant le scrum afin de plancher sur la configuration des paramètres du .jar de la job Spark et la soumission de celle-ci sur Livy.
- Écriture de la base du code pour exécuter des Batch sur Livy (L'équivalent de Livy d'un spark submit.
  - Requiert qu'un .jar soit déjà uploadé sur HDFS(hadoop filesystem)/S3
  - Exemple de spark-submit:

```
./bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master yarn \
--deploy-mode cluster \
 --executor-memory 20G \
/path/to/examples.jar 1000
```

- Exemple de structure json pour Batch Livy:

```
{
        "className":  "org.apache.spark.examples.SparkPi",
      "executorMemory":                              "20g",
      "args":                                        [2000],
      "file":                          "/path/to/examples.jar"
}
```

- D'autres arguments existent.
- Le paramètre "args" prend une liste de string donc on peut en passer plusieurs.
- On envoie la requête à LIVY_URL/batches
- Tous les paramètres existants:

POST /batches

Request Body

name	description	type
file	File containing the application to execute	path (required)
proxyUser	User to impersonate when running the job	string
className	Application Java/Spark main class	string
args	Command line arguments for the application	list of strings
jars	jars to be used in this session	List of string
pyFiles	Python files to be used in this session	List of string
files	files to be used in this session	List of string
driverMemory	Amount of memory to use for the driver process	string
driverCores	Number of cores to use for the driver process	int
executorMemory	Amount of memory to use per executor process	string
executorCores	Number of cores to use for each executor	int
numExecutors	Number of executors to launch for this session	int
archives	Archives to be used in this session	List of string
queue	The name of the YARN queue to which submitted	string
name	The name of this session	string
conf	Spark configuration properties	Map of key=val

- Méthode offerte aux services Spring qui appelle l'API Livy:

```
public Batch submitBatch(String inDataset, String outDataset, St
    CreateBatch cBatch = new CreateBatch();
    cBatch.args = new ArrayList<>();
    //specify JCommander args and values here
    cBatch.args.add("--formula");
    cBatch.args.add(formula);
    cBatch.args.add("--dataIn");
    cBatch.args.add(inDataset);
    cBatch.args.add("--dataOut");
    cBatch.args.add(outDataset);

    try{
        return service.runBatch(cBatch).execute().body();
    }catch(IOException e){
        System.out.println(e.getMessage());
        return null;
    }
}
```

Collapse ↑

Définition du service spring:

```
@RequestMapping(path = "/com/ticksmith/livy/submitformula/", method = RequestMethod.POST, produces = MediaType.
    public String submitFormula(@RequestBody Document formulaJson){
        String formula = formulaJson.get("formula").toString();
        String datasetIn = formulaJson.get("datasetIn").toString();
        String datasetOut = formulaJson.get("datasetOut").toString();

        Batch batch = new LivyRestClient().submitBatch(datasetIn,datasetOut,formula);

        //return batch ID
        Document responseDoc = new Document();
        responseDoc.append("success", batch != null ? "true" : "false");
        if(batch != null){
            responseDoc.append("batchId", batch.id);
        }
        return responseDoc.toJson();
    }
```

Collapse ↑

Implémentation JavaScript côté client (Web Browser)

```
var json = {
    formula: "sum(op)",
    datasetIn: "s3a://ticksmith-pfe/datain",
    datasetOut: "s3a://ticksmith-pfe/dataout"
};

jQuery.ajax({
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
    type: 'post',
    url: '/livy/submitformula/',
    dataType: 'json',
    data: JSON.stringify(json),
    success: function (data, textStatus, jqXHR) {
        $("#requestData").empty().append(data.success);
        if(data.success == true){
            $("#requestData").append("<br>");
            $("#requestData").append(data.batchId);
        }
    }
});
```

Le tout est à adapter en fonction des paramètres du .jar sur lequel travaille Jean-Philippe.

Éléments bloquants:

- Besoin d'écrire un bash script AWS CLI pour lancer le cluster avec le bootstrap action et récupérer l'IP du cluster + le mettre dans une variable d'environnement pour sauver du temps.
- Besoin que Jean-Philippe commit son code pour adapter les paramètres en ligne de commande à ce qui a été fait de son côté.
- Travail à réaliser au niveau du front-end afin d'avoir un workflow plus complet avec Livy (suivi de la progression d'une batch, output des logs de la batch dans le browser pour permettre un meilleur débogage, etc.). Donc pas mal de javascript avec des Callback à écrire pour appeler les services Spring connectés à Livy et "poller" à un intervalle de temps le statut des Batch qui sont en exécution.

Richard

- Traitement des exceptions dans le code, l'utilisateur est averti lorsqu'il se log sans connexion à la BD ou lorsque lors de sa session la connexion se perd pendant une tentative d'insertion de formule :

. la classe getdata lance une exception lors de la tentative de connexion avec la méthode ci-après qui en a la charge auprès du constructeur

```
public Student mapRow(ResultSet resultset, int i) throws SQLException {
                                .....
                        }
```

. l'exception est gérée par les services REST qui contrôlent la page. Dans le cas par exemple d'une insertion de formule on a :

```
@ResponseBody
        @RequestMapping(value = "/insertion/formules/{formula}", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
        public List <Student> insertFormula (@PathVariable("formula") String formula) {

                List<Student> list = new ArrayList<Student>() ;
                try {
                        getData getDat = new getData ();
                        getDat.PushStudents(formula);
                }
        catch (DataAccessException e) {
                Student error = new Student ();
                error.setNom("Echec fatal");
                list.add(error);
        }

                return list;
        }
```

ce qui est géré par Javascript dans le fichier insert.js pour le rendu final

```
$("#Save").click(function() {
—----------
if (article.nom == "Échec fatal"){
```

```
        alert("Vous venez de perdre votre connection à la base de données ! Vérifiez SVP
!!!"            );      }
--------------------
});
```

- normalisation de la nomenclature des chemins d'appels vers les services REST suivant les instructions du précédent SCRUM

-activation de la fonctionnalité de visualisation de l'arbre de dépendance de la formule accessible par le bouton «See dependance Tree «

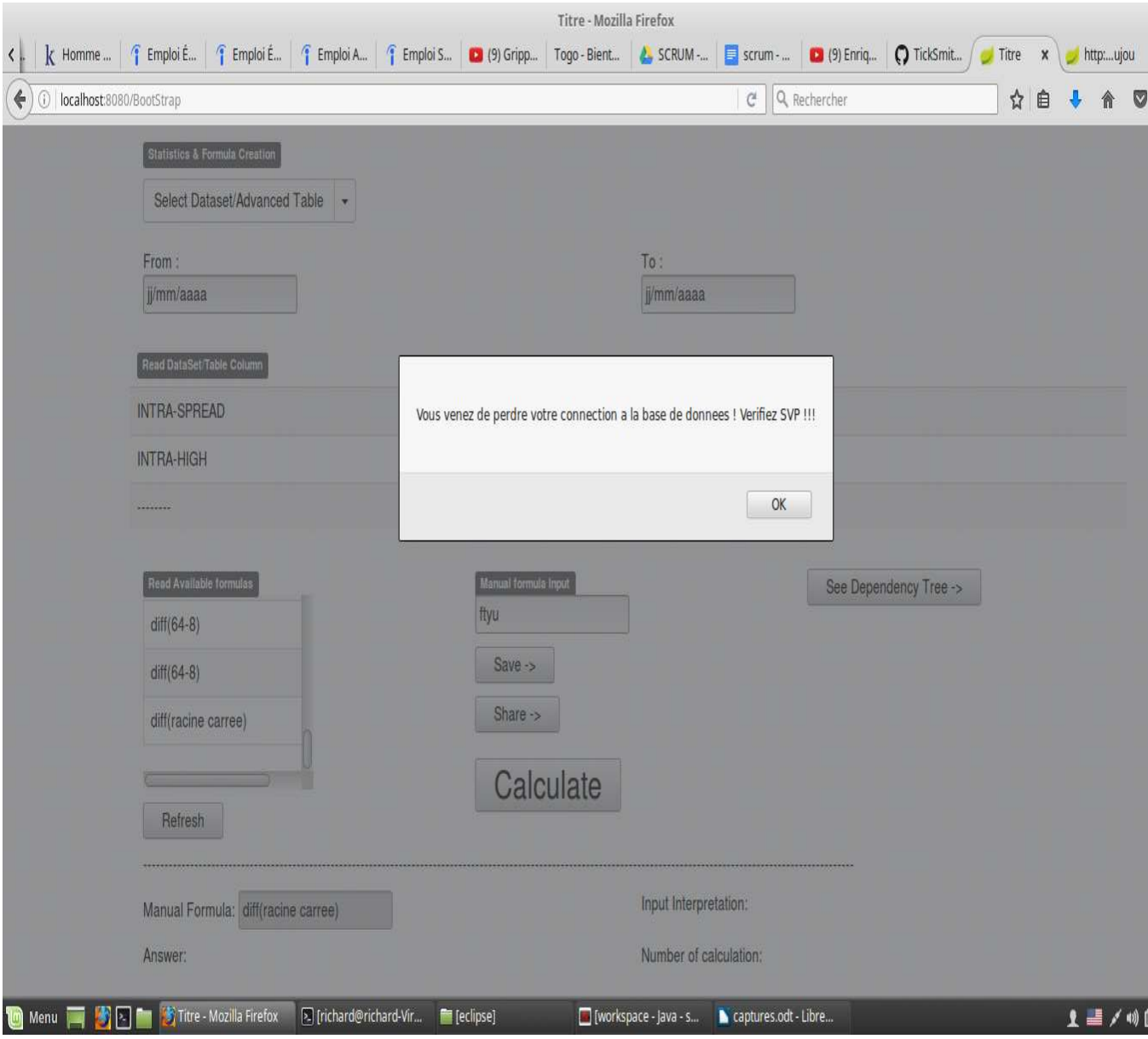
-utilisation du code de Luis pour gérer la présentation de l'image de la formula entrée ainsi que le code latex. L'interface a été accommodée en vue de cette prise en charge.
- dynamisation du chargement visuel des formules en temps réel après leur ajout. Possibilité de sélectionner la formule voulue en vue de visualiser ses dépendances.La formule selectionnee est envoyée par la méthode POST à partir de l'URL

```
$("#dependency").click(function() {
var url = "
http://localhost:8080/TickSmith?formule="+document.getElementById("selection").valu
e
window.document.location.href=url;
});
```

l'information est recueillie et traitee par une procédure insérée au sein du script loadformula.js

```
var parameters = location.search.substring(1).split("&");
var temp = parameters[0].split("=");
l = unescape(temp[1]);
document.getElementById("formuleTextInput").value = l;
```

- mise à jour sur gitHub de la branche testInterface1.3 sur laquelle le code est disponible.



Jean-Philippe Chan

- Capable de rouler spark-submit du fichier analyzer en jar avec les paramètres.

Éléments bloquants:

- J'avais eu un gros problème par rapport avec Ubuntu que je dois réinstaller complètement.

Maxime

- Suivi un tutoriel pour installer Sentry avec PostGreSQL (semblait pas en avoir pour impala)
- Test sur une instance
- Je vais devoir adapter le tutoriel pour remplacer PostGreSQL par Impala.
- Si je comprends bien, le but de Sentry avec Impala, c'est de définir les droits aux différents utilisateurs.
- Il faudrait penser comment on divise au niveau de la présentation: je pourrais parler au niveau de l'infrastructure, etc.

Plan d'actions:

- Richard: Intégrer le code de Luiz pour l'affichage des formules dans une interface consolidée. Permettre l'édition de formules existantes, permettre la suppression de formules existantes.
- JP:
    - Finaliser le .jar utilisé pour le spark-submit
    - supporter la sauvegarde dans une table
    - ajout paramètre pour permettre setMaster(local) e.g --local si --local n'est pas passé, alors ne pas faire setMaster (ca sera le cluster automatiquement)
- Yves: Intégrer le .jar de JP dans le workflow avec Livy, Implémenter un GUI avec un workflow permettant de soumettre une job spark à partir du navigateur web, Implémenter le driver JDBC Impala pour permettre la visualisation des datasets résultants.
- Maxime: Voir commentaire

SCRUM DU MARDI 11 JUILLET 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
    - Tony Bussières, Ticksmith
    - Patrick Cardinal, ÉTS
    - ~~Alain April, ÉTS~~
    - ~~Marc-André Hétu, Ticksmith~~
    - Luiz Fernando Dos Santos Pereira, Ticksmith
    - ~~Yves Millette, ÉTS~~
    - ~~Maxime Paul-Dégarie, ÉTS - Appel Conférence~~

- Kantchil Dam-Hissey, ÉTS
- Jean-Philippe Chan, ÉTS

Effort estimé:

Ce qui a été fait:

Yves

- Rencontre avec Maxime à l'ÉTS le 5 juillet pour déboguer le bootstrap action pour le déploiement d'un cluster avec Livy.
- Semaine difficile (malade + fièvre + antibiotiques). Désolé des inconvénients.

Richard

- Finition de l'API en ecriture dans la Bd MysQL

. Ajout de la methode PushStudents dans le controleur getDtaa

```
public void PushStudents(String student) {
        jdbcTemplate.update("insert into tick_formulas (owner_id, name, definition, scala_definition,latex_definition)"
                + " values (1, \"\",?,\"\",\"\")", student);
    }
```

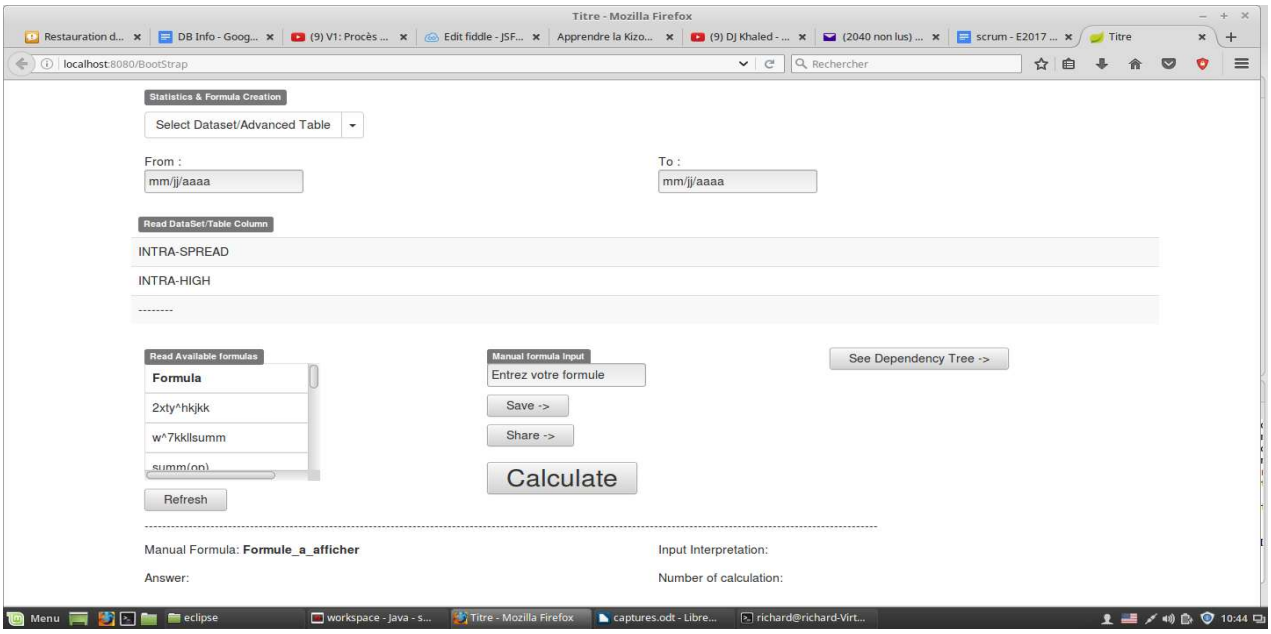
.Appel depuis la classe Static Controler

```
@ResponseBody
    @RequestMapping(value = "/{formula}", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
    public void insertFormula (@PathVariable("formula") String formula) {

        getData getDat = new getData ();
        getDat.PushStudents(formula);
    }
```

-Appel de l'API en insertion a travers le script insert.js

```
$("#Save").click(function() {
var input = document.getElementById("formula");
var url = " http://localhost:8080/"+input.value;
// alert(input.value);
// alert(url);
ajaxGet(url, function () {
// alert(input.value || input.placeholder);
// alert(input.value || input.getAttribute("placeholder"));
input.value = "";
});
});


$("#refresh").click(function() {
location.reload();
});
```

- correction de l'interface pour prise en compte des modifications ci-dessus. Nous avons a présent fonctionnelles l'ajout et la lecture de formules dans la BD comme ci-apres :



- Création de la branche testinterface1.3 dans le repo distant pour tester le résultat a ce niveau implémentation.

Elements bloquants : jai pas capture les erreurs insertion dans le cas ou le tunnel ssh serait off. Je me demande si c'est vraiment utile vu que la version finale de la bd sera locale.

Jean-Philippe Chan

- Continuité sur la modification du code Analyzer.scala afin de prendre les paramètres pour faire un appel avec spark-submit.

Maxime

- Le script bootstrap action est maintenant sur S3, TicksmithPFE (sous le dossier Livy).
- Il est possible de déployer un nouveau Cluster avec le bootstrap action en le spécifiant (en mode advanced).
- Pour le test, c'est des instances M3.medium qui avaient été utilisées dans la zone TicksmithOffice

- J'ai testé le déploiement avec Yves et ça fonctionne bien. Par contre, il ne semble pas avoir une procédure pour fermer un cluster quand il n'est pas utilisé. Alors, on doit le redéployer à chaque fois.
- Rencontre avec Yves la semaine passée, nous avons testé le déploiement et il sait aussi comment redéployer un cluster avec le bootstrap action Livy.

Plan d'actions:

- Richard:
  - Traiter les exceptions dans le code
  - Lire les documentation REST pour mieux nomer les méthodes dans l`API
  - Présenter l'image de la formula entrée ainsi que du code latex
  - Voire l'arbre de dépendance de la formule.
- JP
  - Utiliser le parseur pour générer le SQL basee sur la formule entree. Ensuite soumettre ce SQL dans spark (dataframe).
  - Adapter le parseur pour répondre aux besoins des formules. (commencer avec des formules simples)
- Yves
  - Aider Richard sur la soumission de la formule sur le cluster.

SCRUM DU MARDI 04 JUILLET 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - Tony Bussières, Ticksmith
  - Patrick Cardinal, ÉTS
  - ~~Alain April, ÉTS~~
  - ~~Marc-André Hétu, Ticksmith~~
  - Luiz Fernando Dos Santos Pereira, Ticksmith
  - Yves Millette, ÉTS
  - ~~Maxime Paul-Dégarie, ÉTS - Appel Conférence~~
  - Kantchil Dam-Hissey, ÉTS
  - Jean-Philippe Chan, ÉTS

Effort estimé:

Ce qui a été fait:

<u>Yves</u>

- Service client definition pour Livy:

```java
/**
 * Obtain all active Livy sessions
 * @param from The start index to fetch sessions
 * @param size Number of sessions to fetch
 * @return SessionsResponse containing list of sessions.
 */
@GET("sessions")
Call<SessionsResponse> getSessions(@Query("from") int from, @Query("size") int size);

/**
 * Get a specific Session given it's ID
 * @param sessionId Id of the session
 * @return Session containing session information.
 */
@GET("sessions/{sessionId}")
Call<Session> getSession(@Path("sessionId") int sessionId);

/**
 * Get a specific Session's state given it's ID
 * @param sessionId Id of the session
 * @return SessionState containing the session's state
 */
@GET("sessions/{sessionId}/state")
Call<SessionState> getSessionState(@Path("sessionId") int sessionId);

/**
 *
 * @param sessionId Id of session to fetch log for.
 * @param from Offset of log lines to fetch
 * @param size Number of log lines to fetch
 * @return SessionLog containing the lines for the session's log.
 */
@GET("sessions/{sessionId}/log")
Call<SessionLog> getSessionLog(@Path("sessionId") int sessionId, @Query("from") int from, @Query("size") int size);

/**
 * Get a list of all statements in a session.
 * @param sessionId Id of the session to fetch
 * @return List of Statements for a session
 */
@GET("sessions/{sessionId}/statements")
Call<List<Statement>> getStatements(@Path("sessionId") int sessionId);
```

```java
/**
 * Get a specific statement in a session.
 * @param sessionId Id of the session.
 * @param statementId Id of the
 * @return Statement fetched.
 */
@GET("sessions/{sessionId}/statements/{statementId}")
Call<Statement> getStatement(@Path("sessionId") int sessionId, @Path("statementId") int statementId);


/**
 * Cancel the execution of a specific statement.
 * @param sessionId Id of the session.
 * @param statementId Id of the statement.
 * @return Message (Will always be "cancelled")
 */
@POST("sessions/{sessionId}/statements/{statementId}/cancel")
Call<String> cancelStatement(@Path("sessionId") int sessionId, @Path("statementId") int statementId);

/**
 * Run a code statement in a session.
 * @param sessionId Id of the session.
 * @param code Code to be executed.
 * @return Statement object of the launched statement.
 */
@POST("sessions/{sessionId}/statements")
Call<Statement> postStatement(@Path("sessionId") int sessionId, @Body String code);

/**
 * Create a new ineractive Scala/Python/R shell in the cluster.
 * @param session Session parameters for session to be created
 * @return Object describing the session (contains it's ID)
 */
@POST("sessions")
Call<Session> createSession(@Body CreateSession session);

/**
 * Delete a specific session upon execution completion (We will want
 * to track session IDs and purge inactive sessions from time to time)
 * @param sessionId Id of the session
 * @return OkHttp response body indicating success or failure of the call.
 */
@DELETE("sessions/{sessionId}")
Call<ResponseBody> deleteSession(@Path("sessionId") int sessionId);
```

Éléments bloquants:

- Je suis rendu à exécuter la job sur EMR et je dois modifier le .jar que l'on veut déployer sur AWS afin de pouvoir l'appeler dans Spark-shell en lui passant les paramètres de *Input dataset*, date range, *output dataset* et la *formule à exécuter*.

Richard

- Ecriture d'un API en SpringBoot en substitution de l'abandon de l'utilisation des JSP
-Mise jour de l'interface en vue de la prise en charge dynamique de l'affichage des données à l'utilisateur
-Ajout de deux scripts js en vue du chargement visuel des données :
. ajax js :

```
// Exécute un appel AJAX GET

function ajaxGet(url, callback) {
var req = new XMLHttpRequest();
req.open("GET", url, true);
req.addEventListener("load", function () {
if (req.status >= 200 && req.status < 400) {
// Appelle la fonction callback en lui passant la réponse de la requête
callback(req.responseText);
} else {
console.error(req.status + " " + req.statusText + " " + url);
}
});
req.addEventListener("error", function () {
console.error("Erreur réseau avec l'URL " + url);
});
req.send(null);
}
```

. cours js :

```
var tableau = document.getElementById("tableau");
ajaxGet(" http://localhost:8080/Student", function (reponse) {
// Transforme la réponse en un tableau d'articles
```

```
var articles = JSON.parse(reponse);
articles.forEach(function (article) {
// Ajout du titre et du contenu de chaque article
var ligne = tableau.insertRow(-1);
var colonne1 = ligne.insertCell(0);//on a une ajouté une cellule
        colonne1.innerHTML = article.nom;//on y met le contenu de titre

});
});
```

- Résolution d'une erreur de cross domain ajax lors de l'appel de l'API sur le localhost
- Ajout de trois dépendances dans le projet :

1. Pour résoudre le problème ci-haut : Un CORS Filter (mécanisme permettant de rendre possibles aux application web java les requêtes cross domain au sein des navigateur et serveurs web)

```
<!-- https://mvnrepository.com/artifact/com.thetransactioncompany/cors-filter -->
<dependency>
<groupId>com.thetransactioncompany</groupId>
<artifactId>cors-filter</artifactId>
<version>2.5</version>
</dependency>
```

2. Une bibliothèque de lancement de pool de connection JDBC de haute performance appelée hikaricp

```
https://mvnrepository.com/artifact/com.zaxxer/HikariCP -->
        <dependency>
            <groupId>com.zaxxer</groupId>
            <artifactId>HikariCP</artifactId>
            <version>2.5.1</version>
        </dependency>
        <dependency>
```

3. JDBC template qui est une bibliothèque facilitant l'accès a la BD mySQL

```
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -
->
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>3.2.0.RELEASE</version>
        </dependency>
```

- Insertion de 2API de test pour simuler l'affichage afin de ne pas garder en permanence l'innstance ouverte. Dans la classe Static Controler

API 1 : lance le nouveau formulaire

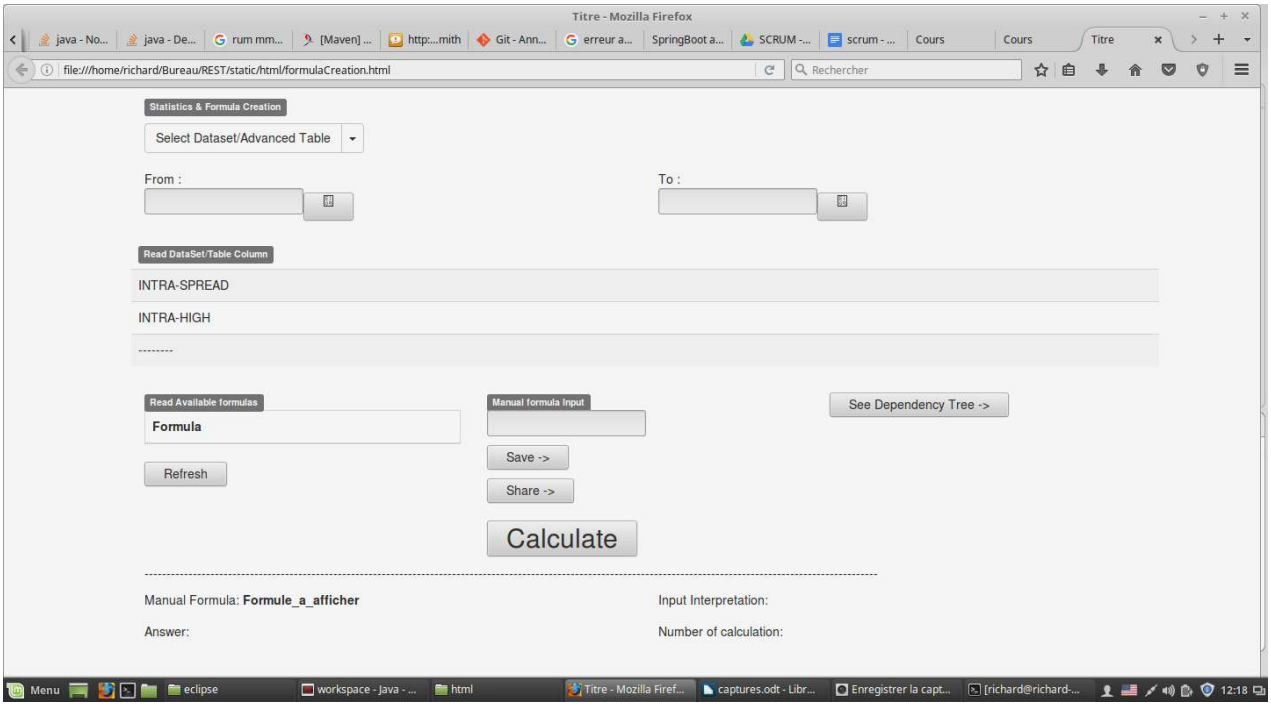
```
@RequestMapping("/BootStrap")
    public String bootstrap(){
        return "html/formulaCreation.html";
    }
```

API 2 : Elle est appelee par le call AJAX dans le script cours.js

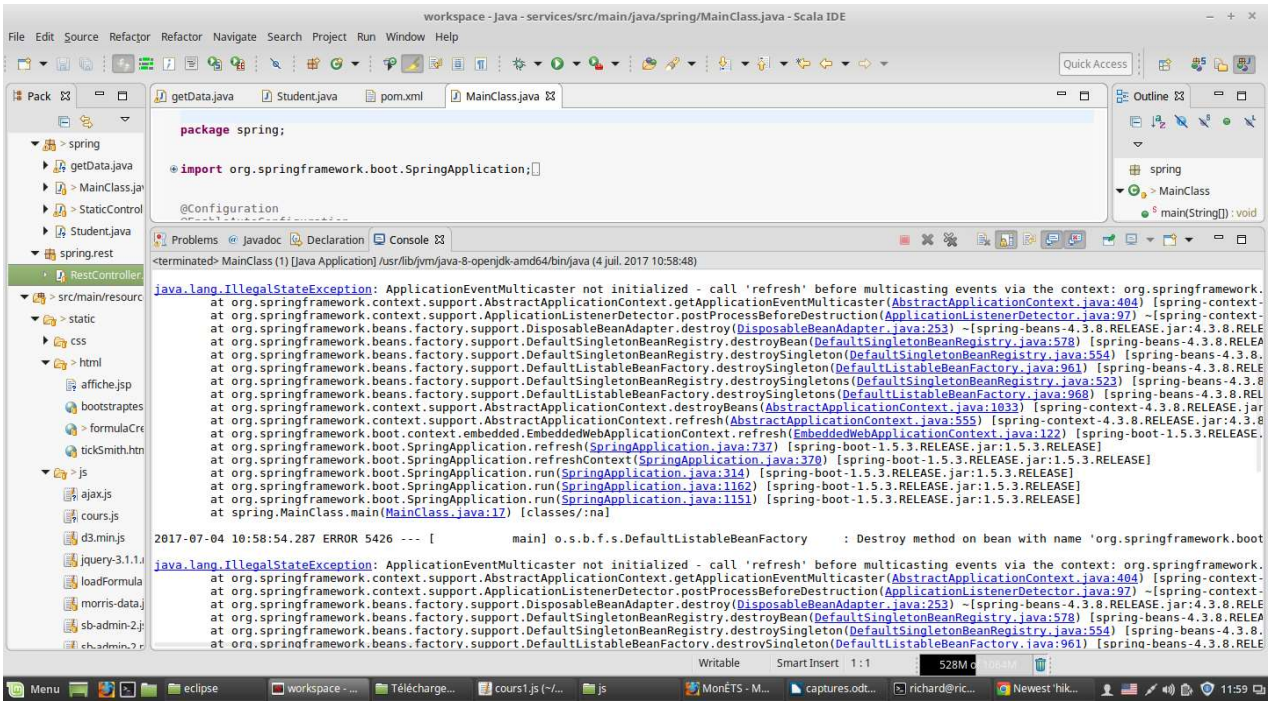
```
    @ResponseBody
    @RequestMapping(value = "/Student", method = RequestMethod.GET, produces
= MediaType.APPLICATION_JSON_VALUE)
    public ArrayList <Student> getStudentsList () {

        Student student1 = new Student();
        student1.setNom("Formule 1");

        Student student2 = new Student();
        student2.setNom("Formule 2");

        ArrayList <Student> studentsList = new ArrayList <Student> ();
        studentsList.add(student1);
        studentsList.add(student2);
```

```
            return studentsList;
        }
```

- Nouvelle interface :

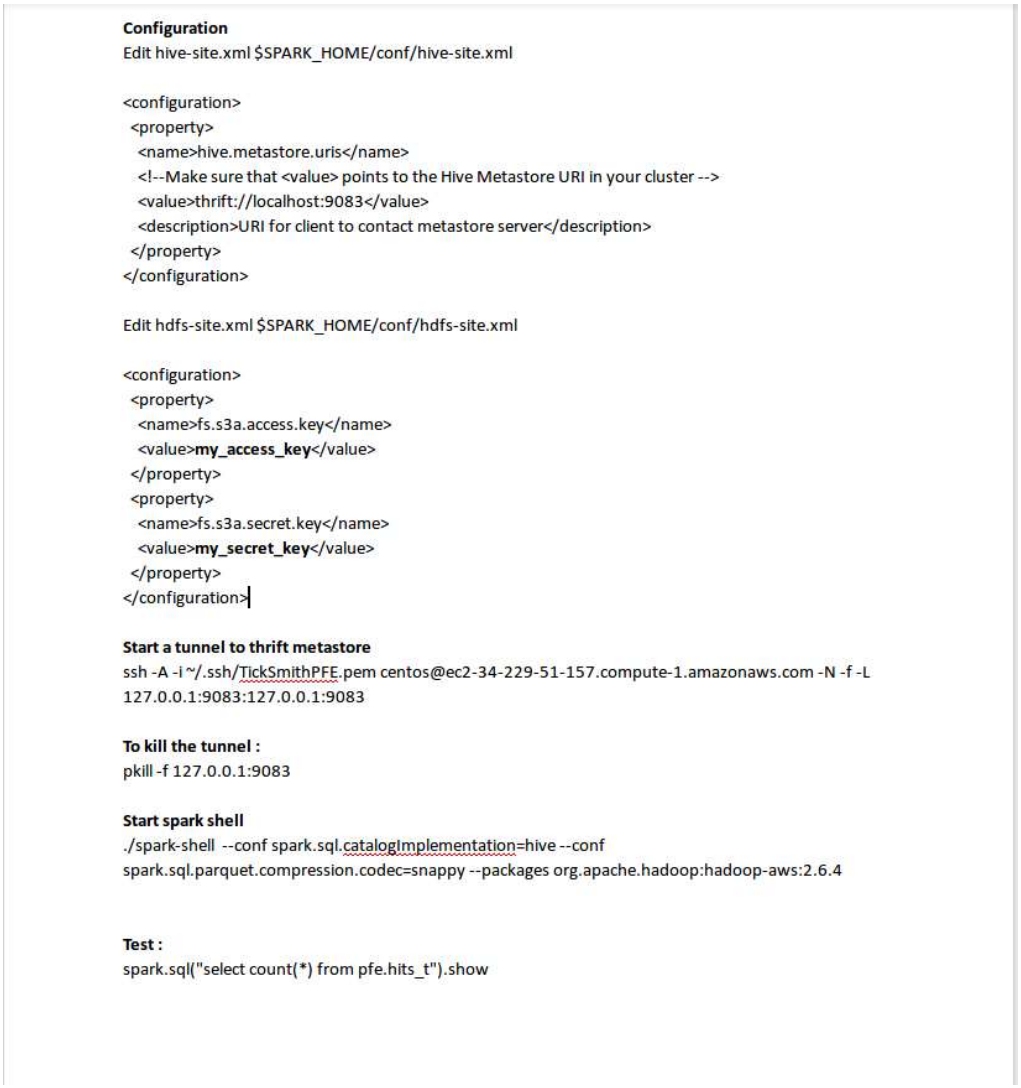


<u>Elements bloquants :</u> J'avais prévue une demo pour aujourd'hui, cependant depuis hier a 23h, j'ai une erreur rédhibitoire au lancement de Spring dont je n'ai absolument aucune idée de la source. Je soupçonne toutefois le dysfonctionnement d'un jar.
Je m'apprête à ramener le projet en l'état initial sans l'ensemble de mes modif afin de visualiser pas a pas quand apparaît le problème.
Vous pourrez visualiser le projet avec cette erreur sur la branche ErreurRichard que je viens de pousser.



<u>Jean-Philippe Chan</u>

- Connection avec Spark sur hive metastore, capable d'accéder au Impala, donc on est capable de faire afficher une requête avec impala.
    - Tony a créé un instruction pour avoir accès à Impala via Hive.
- Creation du branche de ma partie.
- Ajouter les paramètres JCommander pour la recherche de database
    - Presentement non utiliser:



Configuration
Edit hive-site.xml $SPARK_HOME/conf/hive-site.xml

```
<configuration>
 <property>
  <name>hive.metastore.uris</name>
  <!--Make sure that <value> points to the Hive Metastore URI in your cluster -->
  <value>thrift://localhost:9083</value>
  <description>URI for client to contact metastore server</description>
 </property>
</configuration>
```

Edit hdfs-site.xml $SPARK_HOME/conf/hdfs-site.xml

```
<configuration>
 <property>
  <name>fs.s3a.access.key</name>
  <value>my_access_key</value>
 </property>
 <property>
  <name>fs.s3a.secret.key</name>
  <value>my_secret_key</value>
 </property>
</configuration>
```

Start a tunnel to thrift metastore
ssh -A -i ~/.ssh/TickSmithPFE.pem centos@ec2-34-229-51-157.compute-1.amazonaws.com -N -f -L 127.0.0.1:9083:127.0.0.1:9083

To kill the tunnel :
pkill -f 127.0.0.1:9083

Start spark shell
./spark-shell --conf spark.sql.catalogImplementation=hive --conf spark.sql.parquet.compression.codec=snappy --packages org.apache.hadoop:hadoop-aws:2.6.4

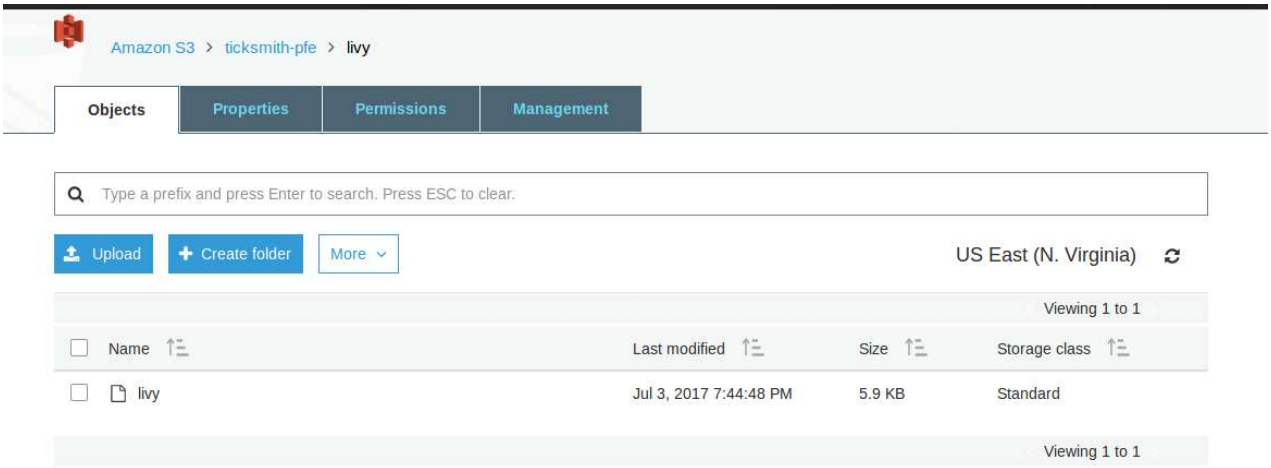
Test :
spark.sql("select count(*) from pfe.hits_t").show

Éléments bloquants:

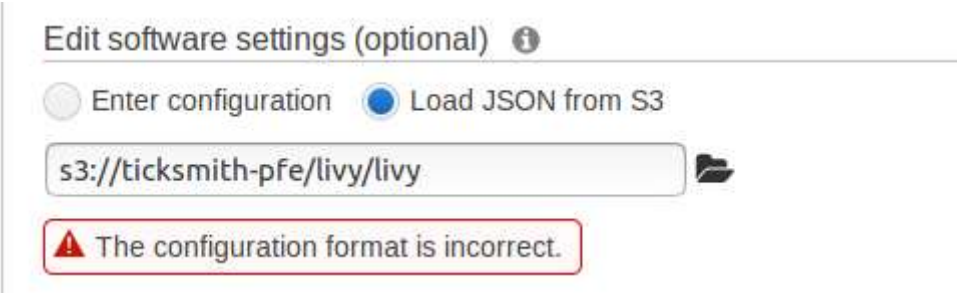
- Est-ce qu'il y a un problème avec le tunnel récemment? Je n'ai pas capable de me connecter sur "centos@ec2-34-229-51-157.compute-1.amazonaws.com", donc je n'ai pas capable de me faire une image pour faire l'exemple.
- Présentement en train de débloquer un problème de connection via analyzer.scala avec Hive.

Maxime

- Upload du json script pour installer Livy sur un cluster EMR sur S3, en re-utilisant le script sur un git



- Tentative de création d'un cluster EMR avec le script sur S3, je dois l'adapter pour notre environnement.



Plan d'actions:

- Yves : regarder spark-submit avec livy
- Richard : faire fonctionner REST + MySQL (testable a partir de docker compose)
-
- JP : connecter dataframe avec impala dans spark. Idealement faire un test sur EMR
- Maxime : faire fonctionner bootstrap action livy.

SCRUM DU MARDI 27 JUIN 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - ~~Tony Bussières~~, Ticksmith
  - Patrick Cardinal, ÉTS
  - Alain April, ÉTS
  - Marc-André Hétu, Ticksmith
  - Luiz Fernando Dos Santos Pereira, Ticksmith
  - ~~Yves Millette~~, ÉTS - Absent à cause du travail.
  - Maxime Paul-Dégarie, ÉTS - Appel Conférence
  - Kantchil Dam-Hissey, ÉTS
  - Jean-Philippe Chan, ÉTS

Effort estimé:

Ce qui a été fait:

Yves

- Écriture d'un client HTTP pour communiquer avec l'API REST de LIVY (en cours).
- Recherches sur le fonctionnement des spark-sessions de Livy.
  - L'api programmatique fourni par com.cloudera.livy et le package https://mvnrepository.com/artifact/com.cloudera.livy/livy-api semble insuffisant pour nos besoins.
  - L'API rest permet toutefois de gérer de multiples sessions similaires à des Spark-Shell: https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.0/bk_spark-component-guide/content/livy-interactive-session.html
  - On pourra donc gérer les sessions à partir de notre api et soumettre nos formules par l'API REST en générant dans nos services spring la formule à exécuter.
- Quelques endpoints utiles:
  - POST /sessions
    - Démarre une session spark-shell en R, Scale ou Python
    - Possible de définir les .jar que l'on veut utiliser dans la session (équivalent d'utiliser `./spark-shell --jars pathOfjarsWithCommaSeprated`) ainsi que d'autres paramètres notamment la plupart des paramètres de Spark.
    - On peut donc "packager" le validateur et le parser de formules dans un .jar qu'on place dans S3 et composer notre statement en utilisant une classe dans le jar. Par exemple, notre .jar pourrait alors contenir tout le code pour exécuter la requête en passant la formule en paramètre:
      `scala> new FormulaRunner("sqrt(sum(col1))").execute();` La requête HTTP au endpoint REST de livy contiendrait donc ceci: new FormulaRunner("sqrt(sum(col1))").execute();
    - GET /sessions/{sessionId}/state

        - Obtient l'état d'une session
    - GET /sessions/{sessionId}
        - Obtient l'information d'une session
    - DELETE /sessions/{sessionId}
        - Kill (termine une session)
    - GET /sessions/{sessionId}/statements
        - Obtient les commandes ayant été exécutées dans une session
    - POST /sessions/{sessionId}/statements
        - Exécute une commande dans une session
    - GET /sessions/{sessionId}/statements/{statementId}
        - Obtient une commande spécifique ayant été éxécutée dans une session
    - POST /sessions/{sessionId}/statements/{statementId}/cancel
        - Annule une commande ayant été exécutée dans une session
- L'API est détaillé ici:
  https://github.com/cloudera/livy#rest-api
- Diagramme de séquence à venir pour détailler le flux d'exécution d'une requête avec une formule (job).
- Utilisation de la librairie retrofit2 (http://square.github.io/retrofit/) basée sur OkHttp (http://square.github.io/okhttp/) pour la communication par HTTP avec Livy à partir des services Spring. Je suis toutefois ouvert à l'usage de d'autres librairies j'ai simplement choisi des librairies avec lesquelles j'avais déjà travaillé.

Éléments bloquants:

- Ce n'est pas nécessairement un bloquant mais il faudra déterminer de quelle façon on veut maintenir les sessions Livy du côté des services Spring. Peut-être qu'il serait pertinent de maintenir les id des sessions dans une table MySQL pour chaque utilisateur et rouler un TimerTask (http://docs.oracle.com/javase/8/docs/api/java/util/TimerTask.html) ou dequoi de semblable pour purger les sessions inactives à chaque 20 minutes par exemple.

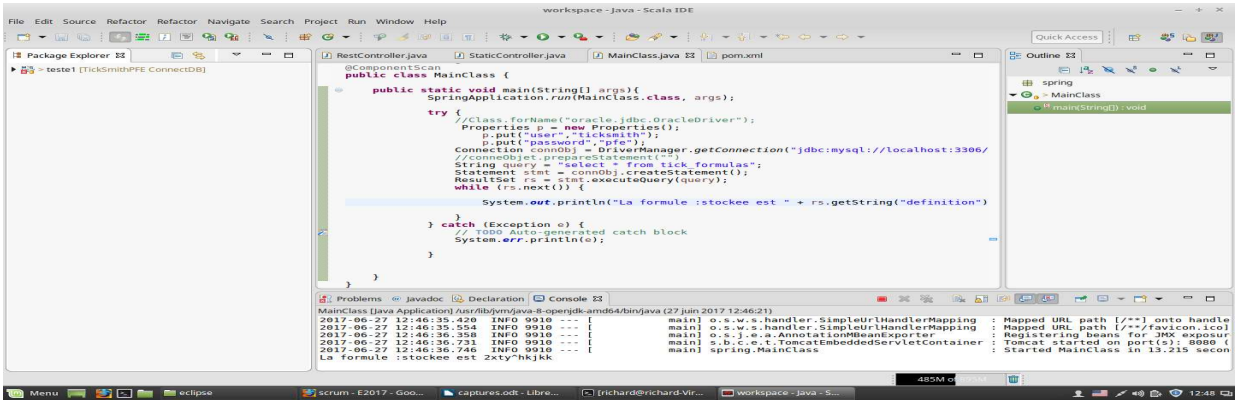
Richard

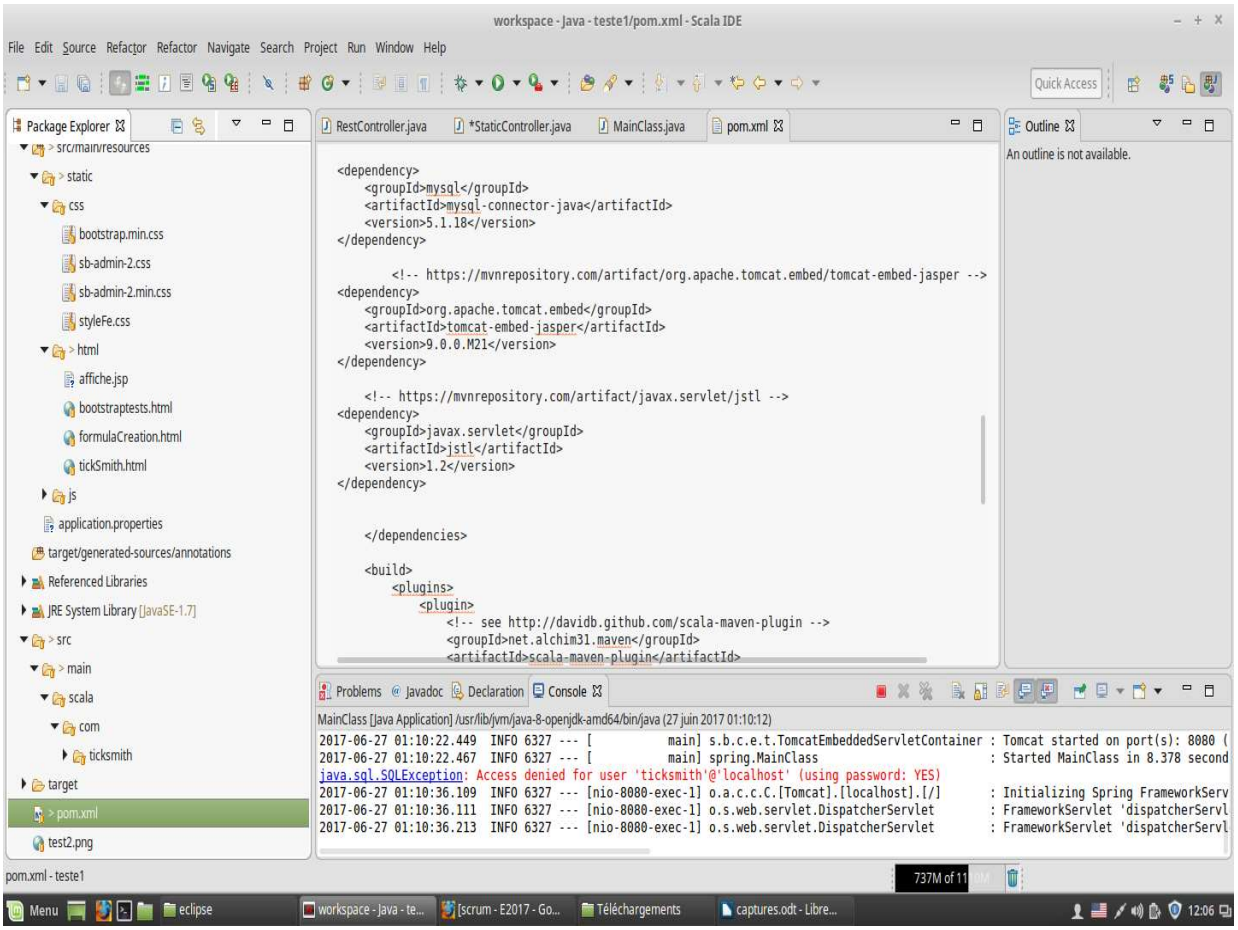
- Ajout dans le fichier pom.xml des dépendances nécessaires a la connexion a la BD MySQL située sur l'instance Amazon

-Ajout de dépendances nécessaires à l'utilisation d'un serveur Tomcat embedded et de javax.servlet. Le but de ceci est d'obtenir une page jsp questionnable par javascript en vue de renvoyer les données reçues de la BD

- Ajout d'un enregistrement de test dans la table tick_formulas

- Création d'une instance de connexion a la BD et affichage des formules recueillies dans la console a l'exécution
- Ajout d'un fichier application.properties dans l'intention de contenir les chemins vers les jsp
- Création d'une branche de l'etat actuel (ConnectDB). NOTEZ QUE : La formule reçue s'affiche uniquement dans la console.





Elements bloquants :

- La formule recueillie de la BD distance doit être maintenant populee sur une page html d'où mon intention d'utiliser des jsp. Suis ouvert a toute suggestion de procédé plus simple.

Jean-Philippe Chan

- Capable de rouler le fichier analyzer.scala avec spark-submit avec paramètre (pratique) pour “df table input”. (Rouler en fichier .Jar)
- Petit travail avec l’interface de database.

Maxime

- Re-installation de Livy dans la zone US East avec la clef TicksmithPFE.pem.
- Consultation et recherche sur l’installation Livy avec Bootstrap. J’ai mis un script sur Google Drive qu’Yves m’avait parlé. Il me semble correct sauf quelques détails que je vais devoir changer!
- Est-ce que je peux déployer un cluster sur AWS pour tester, ou on doit en utiliser un existant?
- Comment on met un fichier script sur notre S3?
- Je vais regarder cette semaine pour nos SCRUM si c’est possible pour moi le mardi vers 16h.

Plan d’actions:

- Jean-Philippe:
  - Essayer d’utiliser le table d`Impala
  - Preparer a presenter le code lors d`scrum

- Cree une nouvelle branche par etudiant
- Richard
  - Faire un service REST pour afficher la formule (et la connection avec la bd)
- Ticksmith: Montrer comment connecter spark / Impala dans un nouveau cluster EMR.

SCRUM DU MARDI 20 JUIN 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - Tony Bussières, Ticksmith
  - Jean-Philippe Chan, ÉTS
  - Alain April, ÉTS
  - Kantchil Dam-Hissey, ÉTS
  - Marc-André Hétu, Ticksmith
  - Luiz Fernando Dos Santos Pereira, Ticksmith
  - Maxime Paul-Dégarie, ÉTS - Appel Conférence
  - Patrick Cardinal, ÉTS

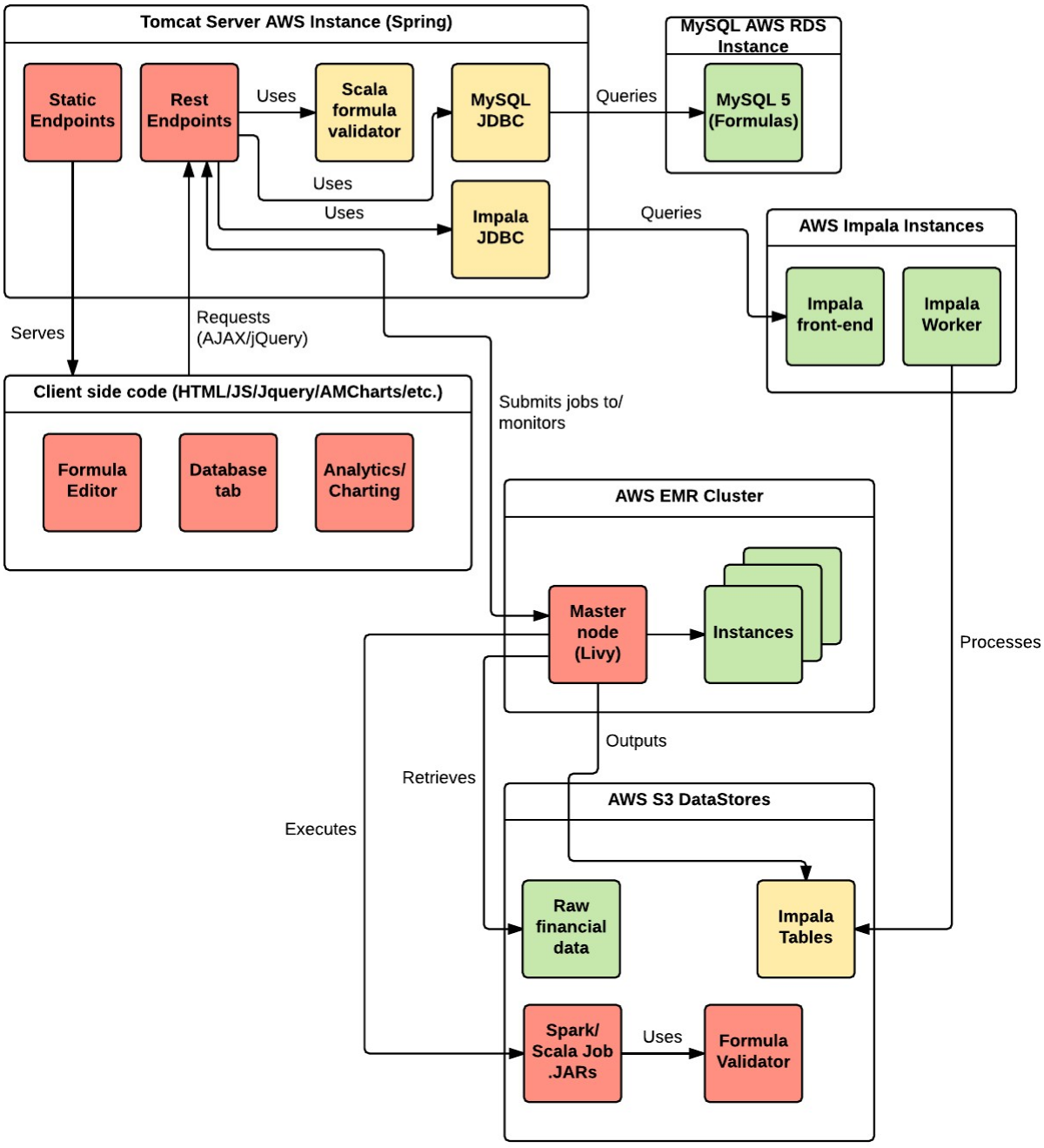
Effort estimé:

Ce qui a été fait:

Yves

- Diagramme d'architecture des composants du projet pour discussion/clarification de ce qui est à faire.
    - Rouge: Ce qui est à faire
    - Jaune: Ce qui est partiellement réalisé/fourni
    - Vert: Ce qui nous est fourni et que l'on doit simplement utiliser.
- Tests en local avec Livy (toujours la grande question de voir si possible de récupérer le JAR dans s3 ou si on doit absolument faire un spark-submit et le transférer à partir des services spring (cas dans lequel le .JAR pourrait résider dans S3 et être soumis par REST à Livy).

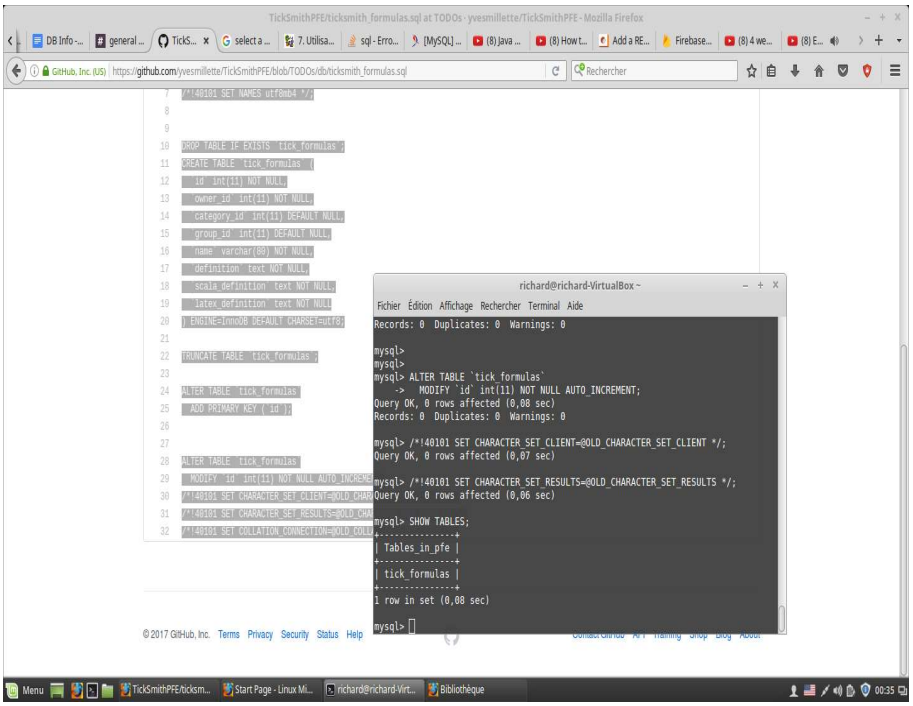
- Éléments bloquants:
    - Installation sur us-east de livy avec les security group adéquats réalisée par Maxime.

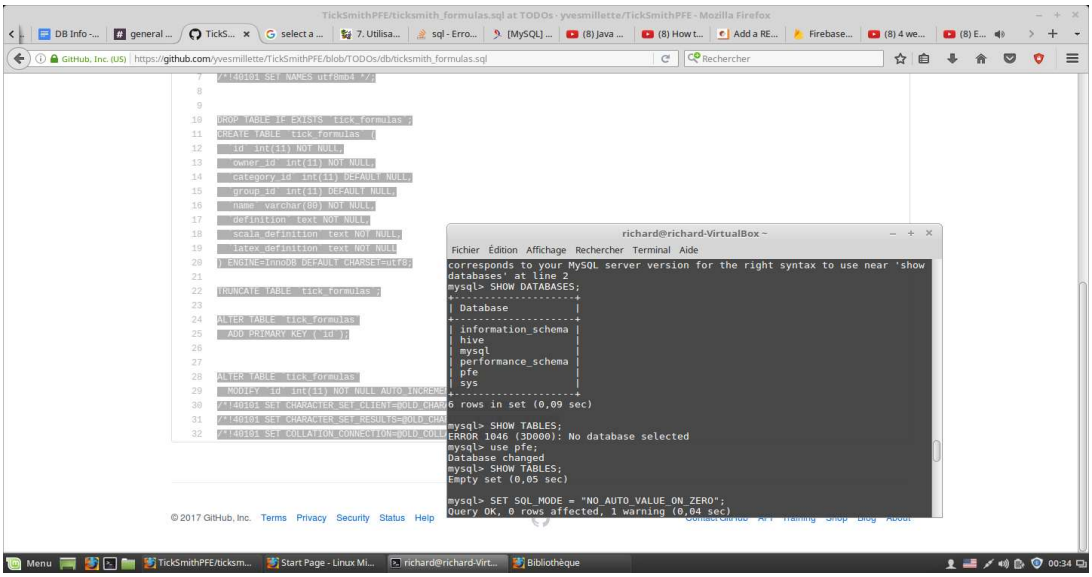


Richard

- Création de la BD pfe sur le serveur MySQL de l'instance dédiée;
- Utilisation du Script de Yves pour créer la table tick_formulas avec les bonnes colonnes

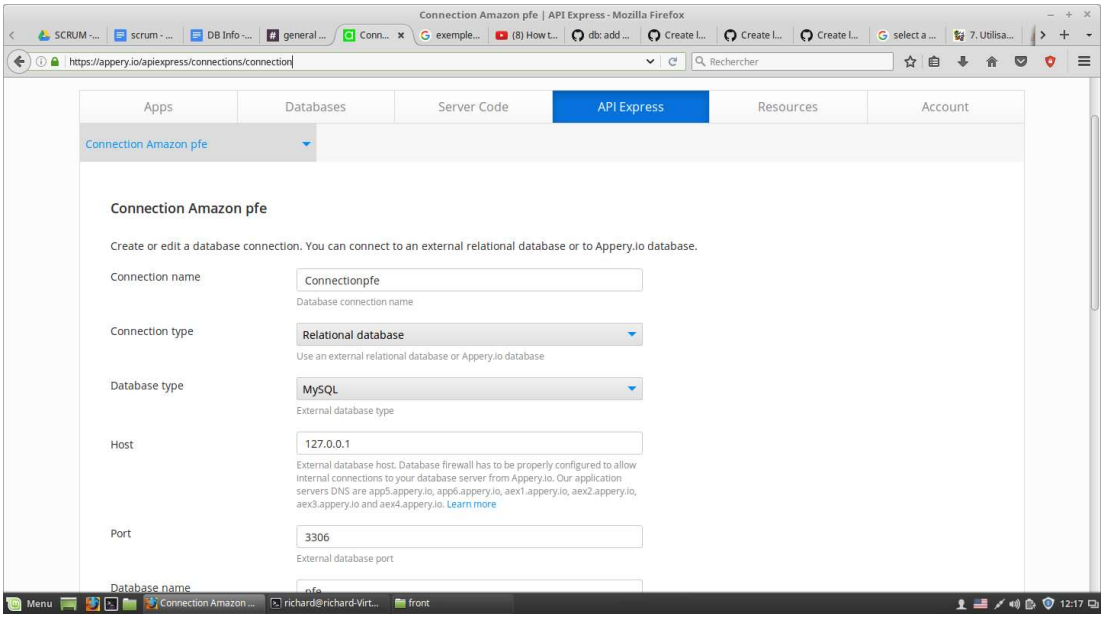
pour le stockage des formules





- Recherche sur l'utilisation d'un API REST pour interroger une BD MySQL et recueillir les données au format JSON en vue de leur affichage par une fonction javascript.

Dans le processus j'ai essayé d'utiliser la solution https://appery.io/ qui livre l'api REST. Cependant l'idée a été abandonnée vu qu'à terme la solution sera payante. Je vais donc créer l'api.



Éléments bloquants : C'est ma première utilisation des services REST

Jean-Philippe Chan

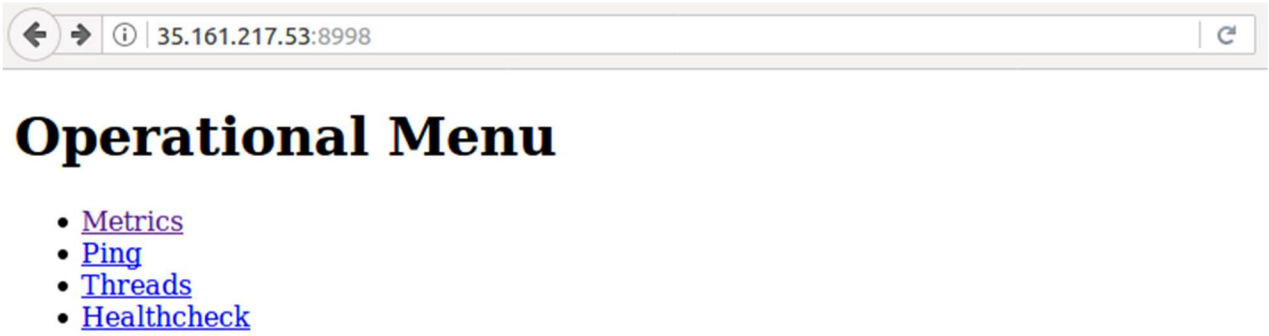
- Comprendre comment fonctionne impala-shell et de le documenter en doc.
  - Tenter de faire un exemple avec un des parquets que j'ai mis sur s3.
- Tenter de créer une vue pour afficher une table de base de données. (spark?)

Maxime

- Livy installé manuellement sur une machine EC2, avec Spark!

```
ubuntu@ip-172-31-10-203:~/livy-server-0.3.0$ ./bin/livy-server
17/06/19 22:55:39 INFO StateStore$: Using BlackholeStateStore for recovery.
17/06/19 22:55:39 INFO BatchSessionManager: Recovered 0 batch sessions. Next session id: 0
17/06/19 22:55:39 INFO InteractiveSessionManager: Recovered 0 interactive sessions. Next session id: 0
17/06/19 22:55:39 INFO InteractiveSessionManager: Heartbeat watchdog thread started.
17/06/19 22:55:40 INFO WebServer: Starting server on http://ip-172-31-10-203.ec2.internal:8998
```

- Par défaut, j'étais dans la zone de l'ouest (Oregon), j'avais fait une instance, mais je ne trouvais pas la zone "Ticksmith Office". Donc, j'ai refait une instance dans la zone appropriée.



- Je devais ouvrir le port TCP 8998 pour pouvoir accéder l'interface web et tester.
- Questions:
    - Quel(s) OS vous utilisez? J'ai mis Ubuntu 14.04 LTS.
    - Security Zone: Ticksmith Office? C'était à revérifier il me semble?
    - Ressources matérielles de la VM? CPU, storage, etc.?
    - Meetings SCRUM possibles vers 4pm, à reconfirmer de mon côté avec ma nouvelle job. J'aimerais savoir si ce serait une possibilité pour tout le monde?

Plan d'actions:

- Déterminer horaire prochains scrum
- Mardi 27, Luiz va prendre le lead coté ticksmith (Tony en congé)
- Max : Faire une job bootstrap pour installer livy sur le master EMR
- Yves : Soumettre une job spark via le serveur Livy
- Donner des indications sur les paramètres de la job spark
- Richard : Connecter JDBC avec MySQL
- JP : Connecter Spark avec Hive metastore sur EMR

SCRUM DU MARDI 13 JUIN 2017

Rencontre:

- Lieu : Ticksmith, 15h30
- Personnes présentes:
    - Tony Bussières, Ticksmith
    - ~~Jean-Philippe Chan, ÉTS~~

- ~~Alain April, ÉTS~~
- Kantchil Dam-Hissey, ÉTS
- ~~Marc-André Hétu, Ticksmith~~
- Luiz Fernando Dos Santos Pereira, Ticksmith
- Maxime Paul-Dégarie, ÉTS - Appel Conférence
- ~~Patrick Cardinal, ÉTS~~

Effort estimé:

Ce qui a été fait:

Yves

- Cleanup du code du projet (nom des packages, code inutilisé).
- Rencontre de 45 mins avec Richard Kantchil pour lui expliquer comment exécuter le projet de services.
- Recherches sur la soumission et l'exécution de jobs Spark par http avec Livy.

Éléments bloquants:

- Besoin de l'installation de spark sur un cluster pour tester le workflow avec Livy.
- Incertain si le paramètre jars est capable de prendre des URL tel que s3://path-to-jar/somejar.jar.
- Examens intra le samedi 10 juin et mardi 13 juin.

POST /sessions

Creates a new interactive Scala, Python, or R shell in the cluster.

Request Body

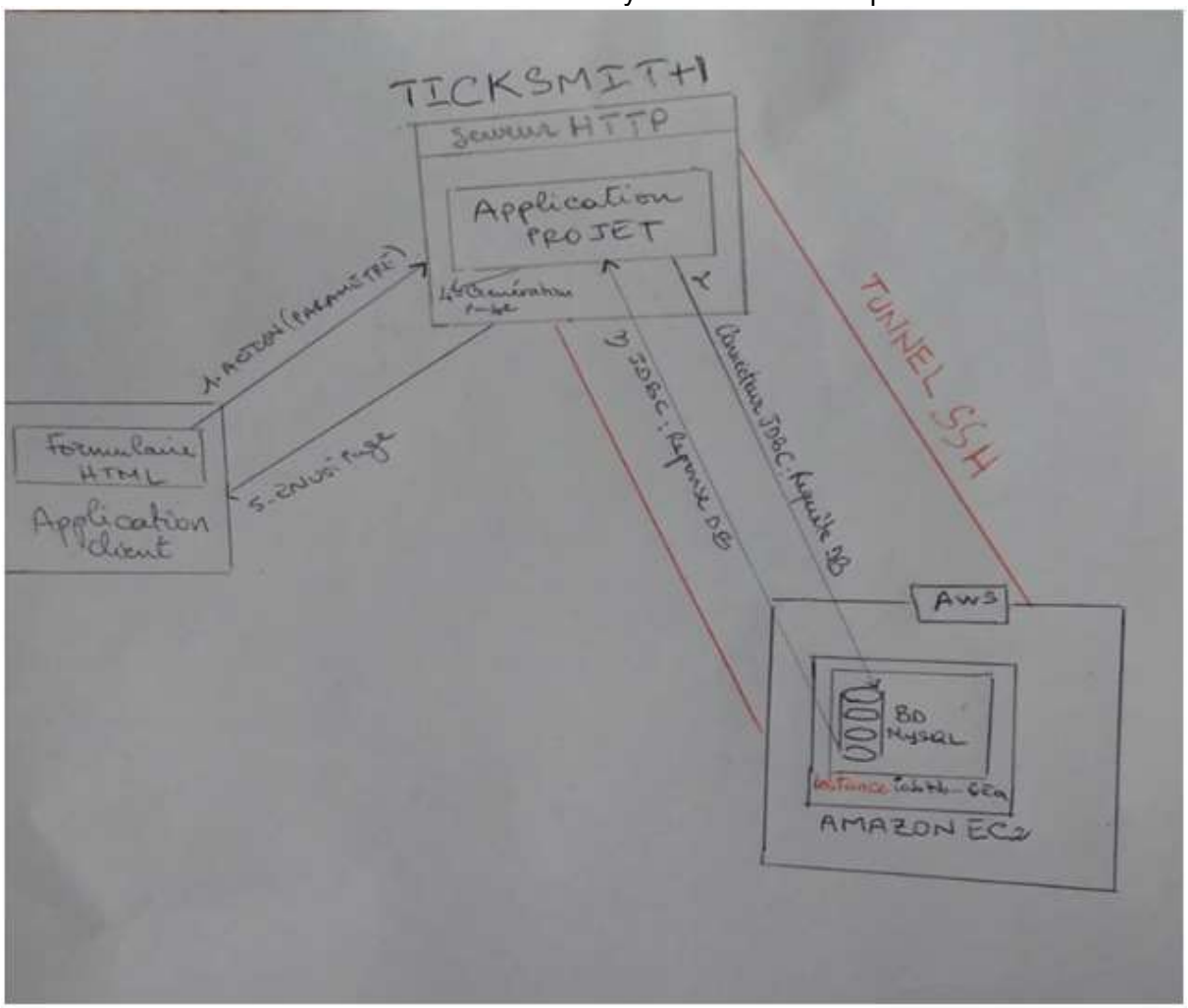
name	description	type
kind	The session kind (required)	session kind
proxyUser	User to impersonate when starting the session	string
jars	jars to be used in this session	List of string
pyFiles	Python files to be used in this session	List of string
files	files to be used in this session	List of string
driverMemory	Amount of memory to use for the driver process	string
driverCores	Number of cores to use for the driver process	int
executorMemory	Amount of memory to use per executor process	string
executorCores	Number of cores to use for each executor	int
numExecutors	Number of executors to launch for this session	int
archives	Archives to be used in this session	List of string
queue	The name of the YARN queue to which submitted	string
name	The name of this session	string
conf	Spark configuration properties	Map of key=val
heartbeatTimeoutInSecond	Timeout in second to which session be orphaned	int

Richard

- Recherche sur les conditions d'intégration d'un connecteur JDBC dans un projet Maven,

j'ai pris connaissance des moyens d'utilisation du connecteur dans une classe java, manipulation simple a effectuer

- installation de python et de la console aws sur ma machine linux, installation de PHPMyadmin et de la suite *LAMP* (Linux Apache, MySQL, PHP)
- tests de connection sur l'instance Ec2 i-0b7b92a595d3be6ea
- création réussie d'un tunnel pour accéder à la BDD MySQL située sur l'instance : succès sauf que hier lundi en matinée, l'accès à l'instance était impossible.
- parcours des nouveaux éléments bootstrap fournis par Yves. Choix des éléments à intégrer pour la prochaine itération
- Réunion de 45 mn avec Yves pour enfin rouler avec succès sur ma VM linux l'application de Luiz : succès
- Dernière correction du document de proposition de projet. Je sais pas si il est finalement transmis
- le mécanisme de connection à la BD se synthétise schématiquement comme suit :



Éléments bloquants

- Une fois le tunnel créé, je n'ai pas pu accéder à la BD MySQL qui y est hébergée en utilisant PHPMyAdmin alors que selon mon entendement une fois le tunnel créé, PHPMyAdmin devrait regarder la BD distante comme étant locale : certainement qu'il va falloir regarder au niveau des fichiers de configuration de l'éditeur de BD ou plutôt changer pour MySQL Workbench. Il va falloir que je puisse réussir rapidement ce test pour vérifier la bonne insertion dans la BD les formules saisies dans l'interface.
- Une fois la connection à la BD insérée dans l'application interface, je me suis buté à la problématique de l'affichage des données réponses sur une interface HTML. La seule technique que je connais relevant des JSP qui ne semblent pas utilisée dans le présent projet. Il faudra que j'inspecte mieux le code de Luiz, qui contient peut être une façon d'y procéder.

Jean-Philippe Chan

Maxime

- Installation de Spark et Livy en local, semble avoir un problème à détecter Spark en démarrant Livy.
- Ajout d'une instance AWS sur Ubuntu pour essayer d'installer Spark et Livy, au lieu d'en local
- Ajout de ma tâche dans le document du PFE: *Implémentation et déploiement* d'un script (Bootstrap action) pour EMR afin d'installer Livy sur le master node AWS.
- Ajout/modification des risques dans le document du PFE

Plan d'actions:

- Maxime : Livy installé manuellement sur une machine EC2 (idéalement sur la machine master EMR)
- Richard : Retourner le résultat d'un query JDBC dans l'écran
- Yves : Spark submit
- JP : JDBC connection vers impala
- À tous : Diagramme d'infrastructure et flux de connection (Impala, EMR, MySQL, Livy, etc)

SCRUM DU MARDI 6 JUIN 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
    - Tony Bussières, Ticksmith
    - Jean-Philippe Chan, ÉTS
    - Alain April, ÉTS
    - Kantchil Dam-Hissey, ÉTS
    - Marc-André Hétu, Ticksmith
    - Luiz Fernando Dos Santos Pereira, Ticksmith
    - Yves Millette, ÉTS
    - Patrick Cardinal, ÉTS
    - Stéphane Gazaille, ÉTS

Effort estimé: 80h (20h par personne)

Ce qui a été fait:

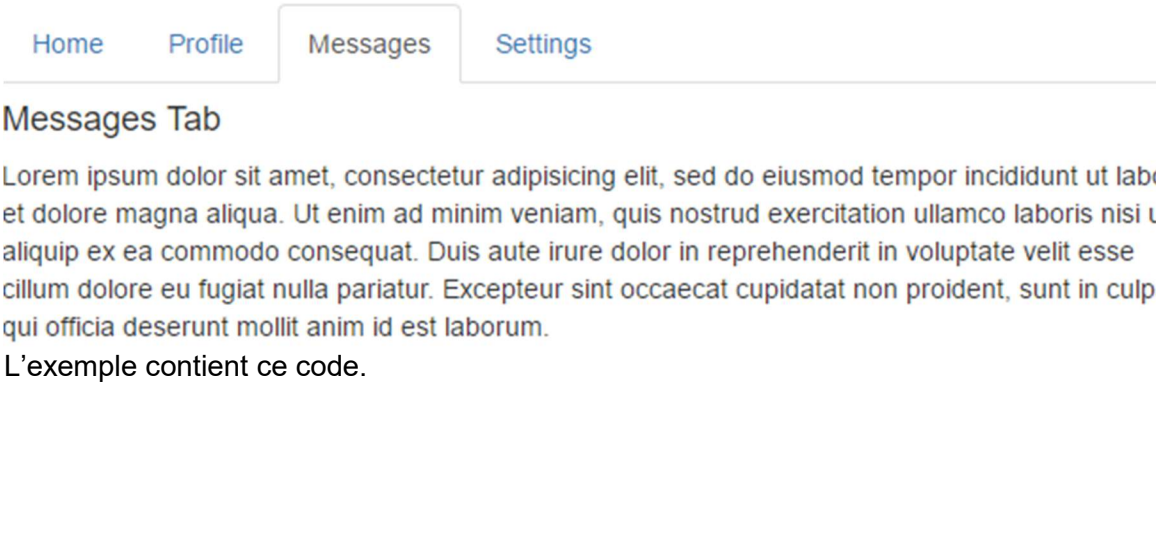
Yves

- Création d'une interface graphique de test avec presque tous les composants bootstrap. Elle est accessible par le projet services sous l'url http://localhost:8080/BootStrap lorsque démarré pour visualiser et tester. Cela devrait permettre à l'équipe d'itérer sur l'interface usager et mieux comprendre le fonctionnement de bootstrap et ses composants. L'interface est basée sur https://startbootstrap.com/template-overviews/sb-admin-2/ Mais a été largement simplifiée (suppression de Flot charts, il faudrait remplacer morris.js par Amcharts quoiqu'on peut maintenir d'autres libs). Liens vers les librairies faits avec l'usage de CDN pour éviter de garder les libs bootstrap et autre sour gestion de code source.
    - On peut prendre l'interface réalisée à date et créer un nouveau endpoint pour partir ce celle-ci pour compléter les interfaces à partir des prototypes.

- Tabs Bootstrap (Important pour le layout de TickSmith)

```
<!-- Nav tabs -->
▼<ul class="nav nav-tabs">
  ::before
  ▼<li class="active">
    <a href="#home" data-toggle="tab" aria-expanded="true">Home</a>
  </li>
  ▼<li class>
    <a href="#profile" data-toggle="tab" aria-expanded="false">Profile</a>
  </li>
  ▼<li>
    <a href="#messages" data-toggle="tab">Messages</a>
  </li>
  ▼<li>
    <a href="#settings" data-toggle="tab">Settings</a>
  </li>
  ::after
</ul>
<!-- Tab panes -->
▼<div class="tab-content">
  ▼<div class="tab-pane fade active in" id="home">
    <h4>Home Tab</h4>
    ▶<p>…</p>
  </div>
  ▼<div class="tab-pane fade" id="profile">
    <h4>Profile Tab</h4>
    ▶<p>…</p>
  </div>
  ▼<div class="tab-pane fade" id="messages">
    <h4>Messages Tab</h4>
    ▶<p>…</p>
  </div>
  ▼<div class="tab-pane fade" id="settings">
    <h4>Settings Tab</h4>
    ▶<p>…</p>
  </div>
</div>
::after
</div> == $0
```

Donne un résultat similaire à ceci:



L'exemple contient ce code.

- Rédaction du document de proposition de projet
  - Corrections dans la section 7 (technique et outils)
  - Réfection de la section des objectifs du projet
  - Ajout du suivi des changements du document basé sur les révisions de google drive.
  - Ajout de risques au projet (section 6)
  - Commenter et modifier la planification (section 5.2)
  - Suppression des instructions de rédaction dans presque toutes les sections
  - Ajout de références pertinentes au projet
  - Il reste la planification du projet et les risques à réviser. Il faudrait envoyer le plus rapidement possible le document à Alain April (s'informer sur la méthode de remise).
- Comparaison de livy, spark-jobserver et l'API d'AWS pour la soumission de jobs Spark.
  - Penchant pour l'utilisation de livy puisqu'il permet l'usage d'un URL que l'on peut "poll" de façon constante pour obtenir le statut d'une job si elle prend plus de quelques millisecondes. On pourrait s'en servir pour afficher des barres de progression dans le UI.
  - On doit tout de même soumettre un .jar avec le code que l'on veut fournir pour la job (pas très clair si le jar peut déja être déployé et si on peut simplement l'appeler avec notre formule en argument. On pourrait toujours soumettre une spark job avec le même jar à chaque fois et donner un argument différent.
  - L'autre méthode serait d'utiliser http://docs.oracle.com/javase/8/docs/api/javax/tools/JavaCompiler.html pour compiler le code localement en .jar et soumettre le .jar avec la formule du client sur le cluster Apache Hadoop YARN avec Livy.
  - Spark-jobserver semble avoir les même fonctionnalités que Livy mais est pas officiellement supporté et maintenu par Cloudera donc ça le rend moins intéressant pour un usage à long terme.
  - L'api AWS permet lui aussi la soumission de .jar et l'exécution à distance. Semblable aux deux autres mais à moins que je ne me trompes offre moins de retour sur la progression de la job. Un api REST comme Livy nous permettrait également d'utiliser javascript pour executer des jobs dans certains cas s'il y a un avantage à ne pas passer par Spring (peu probable puisqu'à long terme notre spring devrait être authentifié).
- Mise à jour des jalons sur github.

Éléments bloquants

- Choix à faire pour la soumission de jobs spark (livy, aws api, spark-jobserver).
- Temps passé sur la documentation.

Richard

- Rédaction du document de proposition de projets de la section 7
  - Additifs de liens dans la section 8 (références)

        - Élaboration détaillée des techniques et outils
    - Rédaction de l'annexe A (plan du travail) et proposition d'un échéancier avec attribution indicative de l'effort pour chaque membre
    - Ajout du suivi des changements du document basé sur les révisions de google drive

- Rencontre hebdomadaire d'équipe avec JP Chan
- Création d'une branche afin de pousser l'interface créée et de permettre de nouvelle iterations
- Recherches en vue de pouvoir accéder à la BD de stockage de formule a partir de la première interface créée

Éléments bloquants

- Je ne suis pas certain de comprendre l'utilisation du serveur impala et son usage dans le contexte de la connexion à une BD

Jean-Philippe Chan

Continuité de la création layout HTML du Database

Rencontre avec Kantchill (Mercredi 31 mai)

Maxime

- Continuer les recherches sur Scala et tutoriels
- Manque de temps à cause de mon déménagement de la semaine/fin de semaine passée, rattrapage en vue :)

Plan d'actions:

- Yves :Installation et tests avec Livy
- JP : Première job sur EMR.
- JP : Ajouter des paramètres à la job spark pour prendre une table comme entrée.
- Richard : Diagramme de connection
- Richard : Test de connection JDBC avec Impala et MySQL
- Alain : Parler avec Maxime

SCRUM DU MARDI 30 MAI 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
    - Tony Bussières, Ticksmith

- Jean-Philippe Chan, ÉTS
- Kantchil Dam-Hissey, ÉTS
- Maxime Paul-Dégarie, ÉTS - Appel Conférence
- Marc-André Hétu, Ticksmith
- Luiz Fernando Dos Santos Pereira, Ticksmith
- Yves Millette, ÉTS
- Patrick Cardinal, ÉTS

Effort estimé: 40h (10h par personne)

Ce qui a été fait:

Kantchil

- Séance de travail le mercredi 24 mai 2017 avec l'équipe. (revue des outils a utiliser en vue de faire rouler les branches du projet)
- Tutoriel sur le fonctionnement du projet ParserApproach (Vendredi, 25 mai)
- Création d'un premier layout statique en HTML et BootStrap en guise de demo pour les statistiques et la création de formule

Maxime

- Poursuite configuration de mon poste Linux (Spark, Scala, Git, Maven)
- Recherches sur Scala et tutoriels
- Tenter de faire fonctionner le tutoriel "Initiation" fait par Jean-Philippe
- Continuer à remplir le rapport de proposition
- Attendre la confirmation du document à signer pour Alain, pour officialiser le projet.

Jean-Philippe Chan

- Commencement de la création layout HTML pour le database.
  - Utilisation: Bootstrap/DataTables
- Comprendre l'utilisation Spark-Shell avec les parquets.
- Rencontre avec l'équipe (Mercredi, 24 mai 2017)

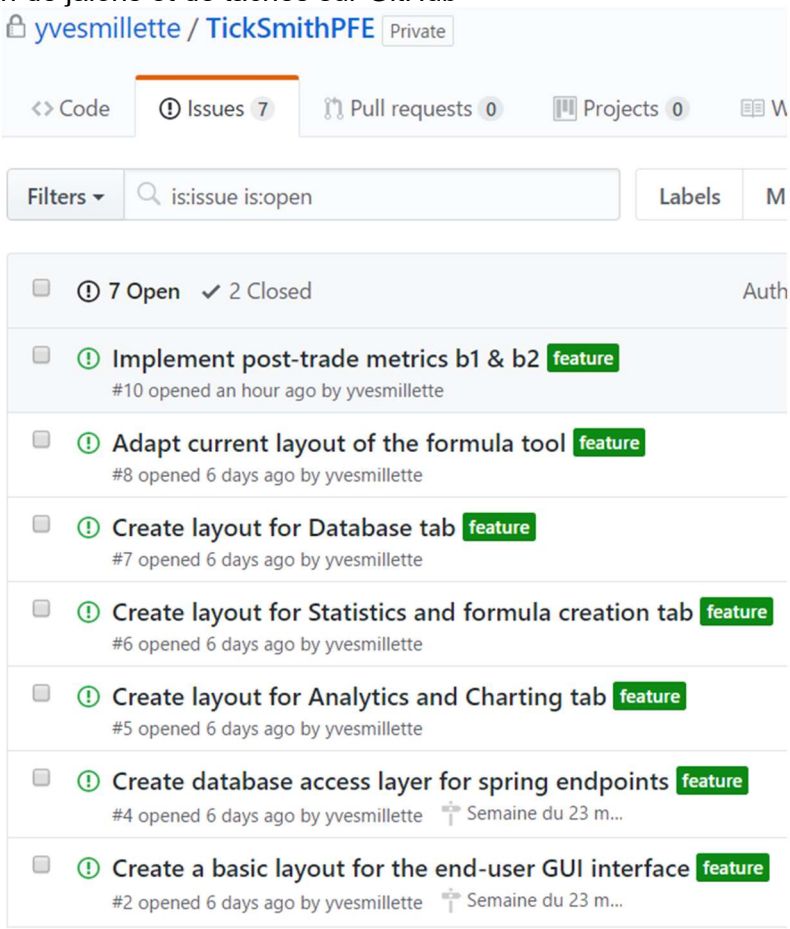
Yves

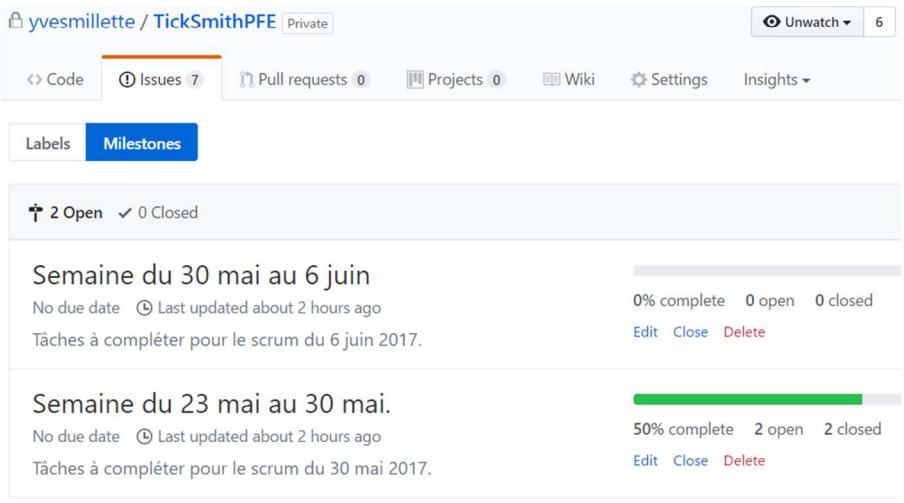
- Rencontre d'équipe le mercredi 24 mai à 18h.
  - Explication à Richard et Jean-Philippe comment exécuter le projet spring boot.
  - Explication à Richard et Jean-Philippe le fonctionnement des issues sur GitHub.
  - Discussion sur les livrables de la semaine dont les prototypes d'interface HTML. Explication à Richard et Jean-Philippe l'usage de base de bootstrap et définition du résultat attendu pour les interfaces à réaliser.
  - Introduction de J-P et Richard à l'utilisation de Postman et Postman Interceptor pour l'analyse de requêtes http envoyées par jQuery vers les services spring.

Nous avons analysé ensemble les requêtes pour les formules et les réponses reçues.

- Division des tâches selon le document de proposition de projet.

- Création de jalons et de tâches sur GitHub





- Déploiement d'une base de données MySQL sur une VM EC2 (i-068a4acb741e795fa)

- Pour que l'on puisse activer/désactiver l'instance lorsqu'elle est utilisée ou pas
- Clés d'accès SSH (RSA) disponible dans le dossier DB du google drive et du repo git.
- Guide d'utilisation rédigé également disponible dans le dossier DB du google drive et du repo git.
- Usager de la BD: ticksmith
- Mot de passe de la BD: pfe

- Merge des projets d'analyse de formules et Spring boot
  - Maintien des commits de Philippe et Luiz après le merge.
  - Création de deux modules maven pour le build.
  - Déplacement des projets dans des dossiers *services* et *analytics* selon leurs objectifs respectifs.
  - Le fichier docker-compose.yml bouge dans le dossier services. Peut-être que ce serait pertinent de créer un script shell à la racine pour le lancer.
  - Pom.xml du nouveau projet principal:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.ticksmith</groupId>
        <artifactId>multi</artifactId>
        <version>1.0-SNAPSHOT</version>
        <name>multi</name>
        <packaging>pom</packaging>
        <modules>
                <module>analytics</module>
                <module>services</module>
        </modules>
</project>
```

  - Changement du projet parent du pom.xml des services. Ceci est pour éviter des warnings concernant le parent du module et le non-respect de la hiérarchie des projets maven.
    La définition de parent suivante causait des warnings puisque le parent réel est le nouveau projet qui définit les deux sous-modules et non le projet spring boot:

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.2.RELEASE</version>
</parent>
```

    Remplacement de cette approche par le chargement de la dépendance vers spring boot et spring boot starter web ainsi que le plugin spring boot. Mise à jour de la version de spring framework en même temps (passé de 1.3.2 à 1.5.3):

```
<plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>1.5.3.RELEASE</version>
        <executions>
            <execution>
                <goals>
                    <goal>repackage</goal>
                </goals>
            </execution>
        </executions>
        <configuration>
                <mainClass>spring.MainClass</mainClass>
        </configuration>
</plugin>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-dependencies</artifactId>
    <type>pom</type>
    <version>1.5.3.RELEASE</version>
</dependency>
```

Le tout est visible dans le changeset 5df52e4

- Compilation et packaging fonctionnels à l'aide de mvn compile & mvn package à la racine. Il est également toujours possible de faire ce traitement sur un des deux projets individuellement.

- Création de la structure de la table pour le stockage de formules mathématiques dans MySQL

- Script de création de table

```
DROP TABLE IF EXISTS `tick_formulas`;
CREATE TABLE `tick_formulas` (
  `id` int(11) NOT NULL,
  `owner_id` int(11) NOT NULL,
  `category_id` int(11) DEFAULT NULL,
  `group_id` int(11) DEFAULT NULL,
  `name` varchar(80) NOT NULL,
  `definition` text NOT NULL,
  `scala_definition` text NOT NULL,
  `latex_definition` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

TRUNCATE TABLE `tick_formulas`;

ALTER TABLE `tick_formulas`
  ADD PRIMARY KEY (`id`);


ALTER TABLE `tick_formulas`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

- Rédaction du document de proposition de projet
  - Section 1. Problématique et contexte
  - Section 3. Méthodologie
  - Section 4. Composition de l'équipe et rôles
  - Section 5.1. Artéfacts
- Recherche sur les options pour la soumission de jobs spark sur AWS
  - https://github.com/spark-jobserver/spark-jobserver (serveur pour soumission HTTP devant être déployé sur l'instance du cluster)
  - http://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-submit-step.html (soumission programmatique avec l'API AWS)

Éléments bloquants:

- Interrogations quant à la façon de compiler un .jar programmatiquement afin d'y intégrer les formules générées par les usagers dans l'outil de formule avant de le soumettre à un cluster sur AWS (spark-submit ou requête à spark-jobserver). Ceci n'est pas un bloquant immédiat puisque nous ne sommes pas rendus à cette étape mais ce serait important à démystifier.

Plan d'actions:

- regarder livy , spark-jobserver et api AWS (spark-submit-step) pour soumettre jar et spark-submit.
- Document proposition
- Tony : donner acces aux images impala + documentation comment utiliser

SCRUM DU MARDI 23 MAI 2017

Rencontre:

- Lieu : Ticksmith, 13h30

Personnes présentes:

- Tony Bussières, Ticksmith
- Alain April, ÉTS
- Jean-Philippe Chan, ÉTS
- Kantchil Dam-Hissey, ÉTS
- Marc-André Hétu, Ticksmith
- Patrick Cardinal, ÉTS
- Fernando Santos Pereira, Ticksmith

Effort estimé: 48h (12h par personne)

Ce qui a été fait:

Maxime

- Rencontre mercredi passé avec les membres de l'équipe

- Réinstallation/configuration de mon poste en Linux (Ubuntu 16.04 LTS)

- Consultation et commencer à remplir le document GTI_LOG_792_Proposition.doc

- Analyse du code sur git

Kantchil

- Séance de travail le mercredi 17 mai 2017 avec l'équipe au complet. (organisation des taches, point des applications nécessaires, accès au bucket S3 avec nos identifiants respectifs)
- Configuration d'un partition linux sur mon poste de travail (Linux Mint) et Installation d'Apache Spark sur la partition
- Installation de scala 2.22.6 et de Hadoop 2.6.2
- Installation de Scala IDE sur Eclipse neon3
- Mise à jour des points 7 et 8 du document de proposition du projet de fin d'études.

Éléments bloquants:

- Le plugin JT weaving nécessaire pour scala dans Eclipse ne s'active jamais en dépit des messages incessants d'activation a valider au lancement de l'IDE. Je poursuis les tentatives d'activation manuelle. Essaie avec Scala-IDE -Yves
- Le manque de dextérité avec l'environnement Linux n'a pas facilité une installation plus rapide des outils du projet du fait de recours systématique aux tutos à chacune des étapes , toutefois une prise en main progressive se met en place

Jean-Philippe

- Tutoriel sur le fonctionnement du projet analytics avec Tony. (Vendredi, 19 mai)
- Installation Linux et les autres logiciels importants pour le projets
- Rencontre d'équipe (17 mai 2017)
- Analyse du projet (Git: TicksmithPFE)

Éléments bloquants:

- Non seulement le tutoriel en Windows nous a pris longtemps pour fonctionner localement, mais il peut avoir un problème quand on doit le faire avec s3, puisque la commande pour . Donc, il est recommandé d'installer sur Linux.

Yves

- Installation des outils de développement nécessaires au projet sous linux sur mon laptop.
- Configuration et exécution des projets.
- Tests avec l'outil de génération de formules.
- Correction de la prise en charge des sommes dans les formules sous format LaTeX.
- Rencontre d'équipe le mercredi 17 mai.
- Recherches sur la façon de subdiviser les projets maven afin de combiner les deux projets dans la même branche. (Je suis plus habitué avec gradle donc c'est une petite adaptation.)

Éléments bloquants:

- Incertitude quant à la façon désirée de joindre les projets dans la même branche.
- Incertitude quant à savoir si le document pour officialiser au projet correspond à la fiche de renseignements sur moodle.

Plan d'actions:

- Faire rouler le code réalisé par Mr Grenier en utilisant les ID AWS et a partir de l'interface de M. Louis avant de proposer des améliorations et veiller à ce que toute l'équipe soit au même niveau pour la prochaine rencontre entre étudiants à l'ÉTS.
- Se familiariser avec Spark Shell.
- Merger les deux branches du repo en une seule
- Effectuer le stockage des formules de l'utilisateur dans un modèle de données à proposer et voir la possibilité de faire rouler des jobs Spark a partir de l'interface
- Finaliser le document de proposition de projet dans lequel les rôles vont etre distribues (Surtout la section 5.2)
- Débuter un prototype d'interface utilisateur avec Bootstrap/jQuery/AMCharts ou avec un outil de prototypage adéquat.
- Prendre connaissance du document CSA Data Fees Methodology (drive)
- Concentrer sur le post-trade metrics (b1 & b2)

SCRUM DU MARDI 16 MAI 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
  - Yves Millette, ÉTS
  - Tony Bussières, Ticksmith
  - Alain April, ÉTS
  - Jean-Philippe Chan, ÉTS
  - Kantchil Dam-Hissey, ÉTS
  - Maxime Paul-Dégarie, ÉTS - Appel Conférence
  - Marc-André Hétu, Ticksmith
  - Fernando Santos Pereira, Ticksmith

Effort estimé: 8h par personne (32 heures)

Ce qui a été fait:

<u>Maxime</u>

- Consultation du "drive"
- Installation des outils demandés dans le fichier "Cadre techno".

- Lecture de la présentation et consultation du rapport PFE précédent
- Remplir le document NDA nécessaire et le signer.
- Envoie de mon compte Github sur slack
- Demande pour fixer une rencontre entre étudiants

Kantchil

- Lecture du rapport de Philippe Grenier-Vallée
- Familiarisation avec jQuery & Bootstrap
- Envoi du compte Github à Yves sur Slack
- Signature et lecture du document NDA

Jean-Philippe

- Installation des outils demandés dans le fichier “Cadre techno”.
- Lecture de la présentation “Big Data, Small Numbers” et “Stats Engine Portal v. 0.6” pour recherche de proposition.
- Remplir le document NDA nécessaire et le signer.

Éléments bloquants:

- Mal de communication entre l’équipe:
    - Oubli de remettre mon compte Github pour accès aux code et aux documentations du projet PFE.

Yves

- Création du repo GitHub pour le projet.
- Création d’un channel Slack pour le projet et invitation des membres
- Création d’une liste de diffusion Google Groups pour l’envoi de courriels à tous les intervenants. (ticksmithpfe@googlegroups.com)
- Contacter les membres de l’équipe pour remplir les informations sur le formulaire de NDA.
- Faire circuler le NDA rempli pour obtenir les signatures des intervenants.
- Consultation du code source des projets pour comprendre le travail ayant été fait lors du projet de l’automne 2016.
- Entamer la rédaction du document de proposition de PFE.
- Consultation de documentation variée portant sur les façon de créer une application Spark soit avec Scala, Java ou Python.
- Lecture de la documentation disponible dans le google drive. (Stats engine + présentation de Philippe de l’automne 2016)

Scala

Copy

```
package org.apache.spark.examples
import scala.math.random
import org.apache.spark._

/** Computes an approximation to pi */
object SparkPi {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("Spark Pi")
    val spark = new SparkContext(conf)
    val slices = if (args.length > 0) args(0).toInt else 2
    val n = math.min(100000L * slices, Int.MaxValue).toInt // avoid overflow
    val count = spark.parallelize(1 until n, slices).map { i =>
      val x = random * 2 - 1
      val y = random * 2 - 1
      if (x*x + y*y < 1) 1 else 0
    }.reduce(_ + _)
    println("Pi is roughly " + 4.0 * count / n)
    spark.stop()
  }
}
```

Java

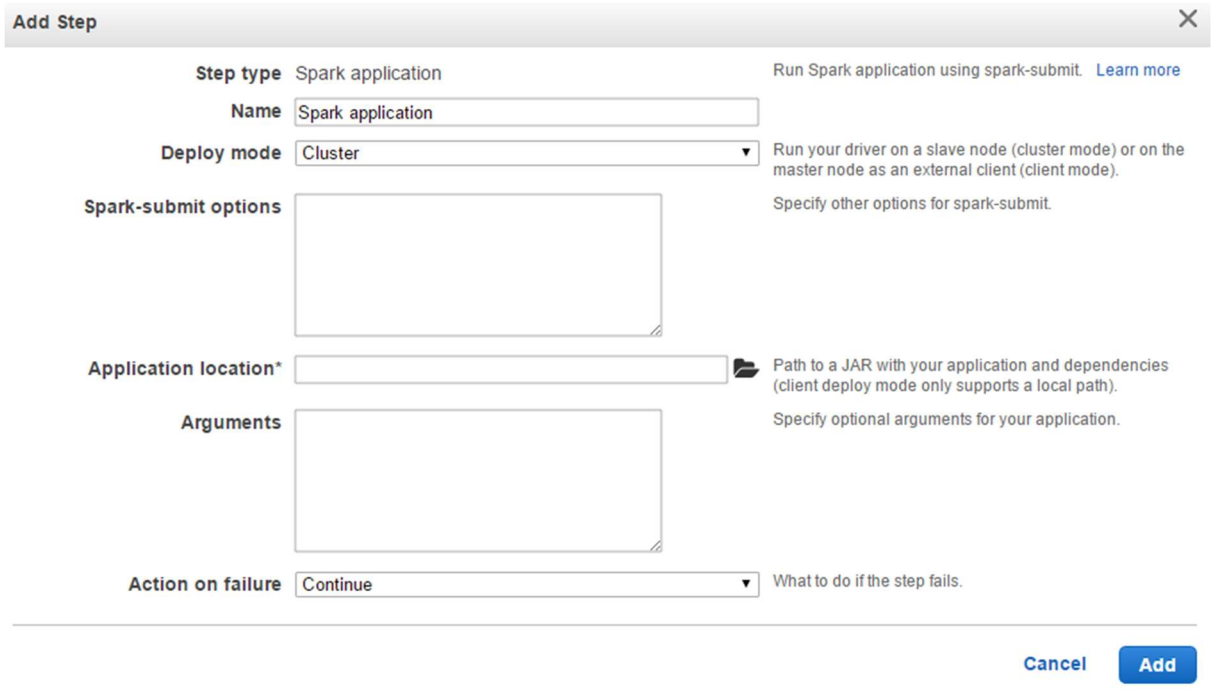
Copy

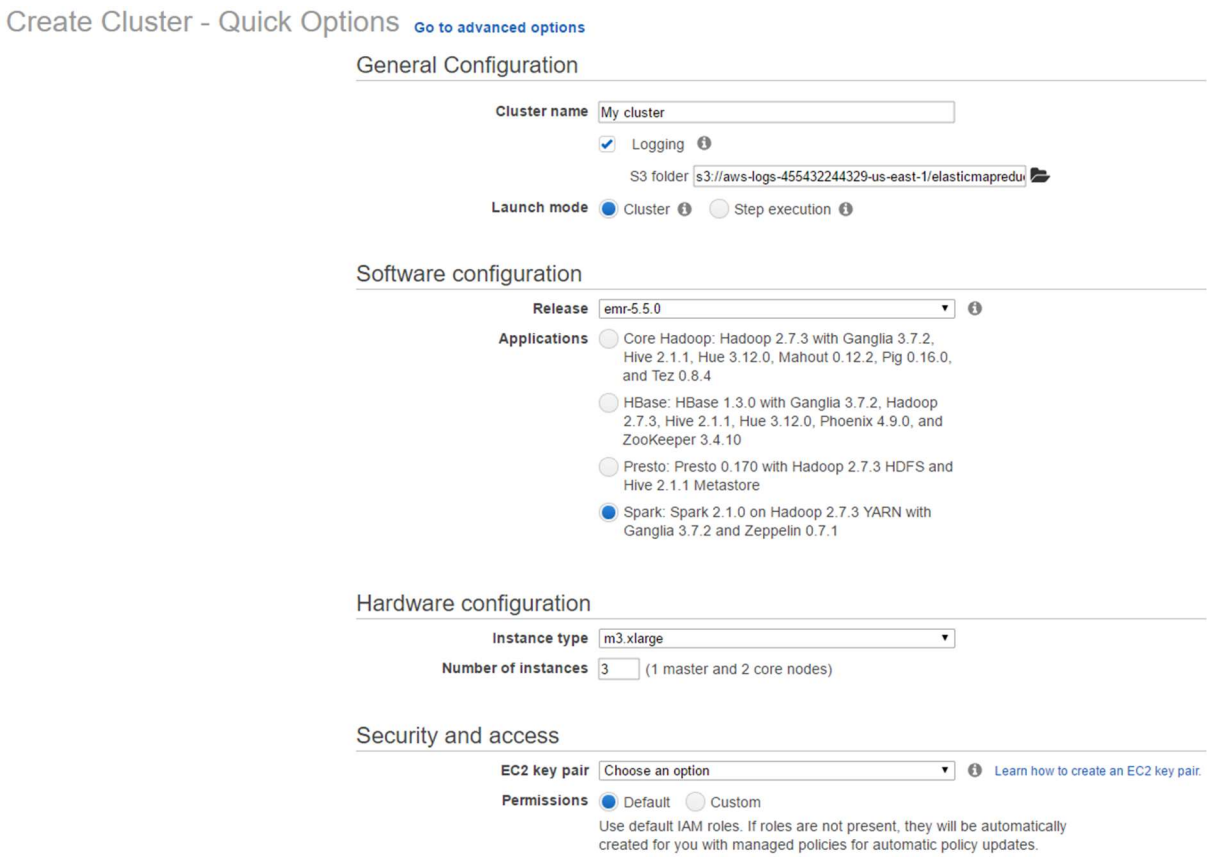
```
package org.apache.spark.examples;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.Function2;

import java.util.ArrayList;
import java.util.List;

/**
 * Computes an approximation to pi
 * Usage: JavaSparkPi [slices]
 */
public final class JavaSparkPi {

  public static void main(String[] args) throws Exception {
    SparkConf sparkConf = new SparkConf().setAppName("JavaSparkPi");
    JavaSparkContext jsc = new JavaSparkContext(sparkConf);

    int slices = (args.length == 1) ? Integer.parseInt(args[0]) : 2;
    int n = 100000 * slices;
    List<Integer> l = new ArrayList<Integer>(n);
    for (int i = 0; i < n; i++) {
      l.add(i);
    }

    JavaRDD<Integer> dataSet = jsc.parallelize(l, slices);

    int count = dataSet.map(new Function<Integer, Integer>() {
      @Override
      public Integer call(Integer integer) {
        double x = Math.random() * 2 - 1;
        double y = Math.random() * 2 - 1;
        return (x * x + y * y < 1) ? 1 : 0;
      }
    }).reduce(new Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer integer, Integer integer2) {
        return integer + integer2;
      }
    });

    System.out.println("Pi is roughly " + 4.0 * count / n);

    jsc.stop();
  }
}
```

Python

```
import sys
from random import random
from operator import add

from pyspark import SparkContext


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    sc = SparkContext(appName="PythonPi")
    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    count = sc.parallelize(xrange(1, n + 1), partitions).map(f).reduce(add)
    print "Pi is roughly %f" % (4.0 * count / n)

    sc.stop()
```

- Consultation des méthodes de déploiement d'applications Spark sur AWS soit par Cluster ou Spark step





http://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-submit-step.html

https://aws.amazon.com/blogs/big-data/submitting-user-applications-with-spark-submit/

Éléments bloquants:

- Accès aux données sur S3 pour exécuter le code fourni.
- Disponibilité des ID aws pour l'accès aux données et l'exécution des applications
- Signature du NDA par les membres de l'équipe pour avoir accès aux données

Plan d'actions:

- Finaliser le document de proposition de PFE
- Utiliser les ID AWS pour déployer l'application spark de Philippe disponible sur github et pouvoir exécuter le conteneur docker de l'application de Luiz (UI avec les formules, etc)
- Définir les objectifs à atteindre afin d'améliorer la solution (quelles formules implémenter par exemple).
- Débuter un prototype d'interface utilisateur avec Bootstrap/jQuery/AMCharts ou avec un outil de prototypage adéquat.
- Rencontre les mercredis à 18h pour les étudiants à l'ÉTS.
- Merger les deux branches du repo en une seule.

- Prendre connaissance du document CSA Data Fees Methodology (drive)
- Concentrer sur le post-trade metrics (b1 & b2)
- Se familiariser avec Spark Shell.

SCRUM DU MARDI 9 MAI 2017

Rencontre:

- Lieu : Ticksmith, 13h30
- Personnes présentes:
    - Yves Millette, ÉTS
    - Kantchil Dam-Hissey, ÉTS
    - Jean-Philippe Chan, ÉTS
    - Maxime Paul-Dégarie, ÉTS - Appel conférence
    - Alain April, ÉTS
    - Patrick Cardinal, ÉTS
    - Tony Bussières, Ticksmith
    - Marc-André Hétu, TickSmith
    - Luiz Fernando Santos Pereira, TickSmith

Effort estimé: 16 heures (4h par personne)

Plan d'actions: Recherche sur les technos utilisées

- Recherche sur le domaine d'affaires
- Consulter Drive (specs, cadre techno)
- Yves: Créer repo (privé) dans github (et accès push à @codingtony, @lufisp, @aapril)
- Tony: Pousser code sur repo
- Tony: Donner NDA à l'équipe pour le data
- Tony : Document Data Fee Modeling
- Alain : ID AWS
- Planifier rencontres en personne
- Regarder rapport de proposition de PFE
- Consultation du code source fourni par ticksmith
- Familiarisation avec AWS (surtout EMR)

Ce qui a été fait:

- Présentation des membres de l'équipe à tour de rôle
- Présentation de Ticksmith par Tony
- Présentation de l'interface TickVault par Luiz
- Présentation de l'outil de formule par Tony

    - Rendering de la formule par Scala Parser Combinator
    - Module thirdparty pour la génération de l'image de la formule (généré à partir d'une formule LaTex)
- Présentation des prototypes d'interface du *Stats engine portal* par Tony
- Voir le fichier Cadre Technologique pour connaître les outils à utiliser. Fichier présenté par Tony.