HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département : Technologies de l'information et de la communication
Filière : Informatique et systèmes de communication
Orientation : Informatique logicielle

Travail de Bachelor

Analyse et implémentation d'un système de logging multi-niveau pour une plateforme Smart Grid

Étudiant : Jael Dubey

Travail proposé par : Jonathan Bischof
DEPsys SA
Route du Verney 20B
1070 Puidoux

Enseignant responsable : Dr Nastaran Fatemi

Année académique : 2019-2020

Yverdon-les-Bains, le 31 juillet 2020

Département : TIC
Filière : Informatique et systèmes de communication
Orientation : Informatique logicielle
Étudiant : Jael Dubey
Enseignant responsable : Nastaran Fatemi

Travail de Bachelor 2019-2020

Analyse et implémentation d'un système de logging multi-niveau pour une plateforme Smart Grid

Nom de l'entreprise/institution

DEPsys SA

Résumé publiable

Dans le paysage informatique actuel, passé et très probablement futur, quasiment tous les programmes génèrent des logs, ou journaux en français. Ces logs sont bien souvent oubliés, ou simplement stockés automatiquement dans un fichier qui tombera lui aussi dans l'oubli. Les logs sont en effet très pénibles à lire pour un humain, ils se ressemblent et parfois contiennent une partie incompréhensible, comme des codes numériques.
Par contre, un travail de lecture d'informations qui se ressemblent est totalement adapté à un ordinateur. C'est pourquoi il existe des logiciels appelés « système de gestion de logs », dont le but est de lire les informations contenues dans les logs et d'en faire des visualisations agréables à utiliser. L'objectif ce Travail de Bachelor était d'étudier ces différents systèmes afin d'en choisir un, puis de le déployer dans une plateforme Smart Grid.
Pour ce faire, une évaluation des différents systèmes actuellement utilisés dans le monde a d'abord permis de choisir un outil adapté à notre projet. Ensuite, un déploiement a été fait avec les outils Docker et Docker-Compose. Enfin, une recherche a été effectuée sur les différents algorithmes et techniques utilisés afin de soutirer encore plus d'information à ces logs. En effet, un journal en soi véhicule déjà une information, mais en mettant en relation plusieurs logs, il est possible d'anticiper des problèmes, ou encore de détecter des problèmes complexes de manière automatique.
Dans ce Travail de Bachelor, le programme qui a été choisi lors de l'évaluation est la Suite Elastic. Une architecture fonctionnelle et facilement déployable a ensuite été créé. Pour finir, des recherches sur les algorithmes actuellement utilisés dans ce domaine ont expliqué les fonctionnalités de chacun, puis une implémentation a montré les capacités de ces algorithmes.

Étudiant :	Date et lieu :	Signature :
Jael Dubey	Yverdon-les-Bains, le 30.07.2020	

Enseignant responsable :	Date et lieu :	Signature :
Nastaran Fatemi	Yverdon-les-Bains, le 30.07.2020	

Nom de l'entreprise/institution :	Date et lieu :	Signature :
Jonathan Bischof DEPsys SA	Puidoux, le 30.07.2020	

Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'Ecole.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Vincent Peiris
Chef de département TIC

Yverdon-les-Bains, le 31 juillet 2020

Authentification

Le soussigné, Jael Dubey, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Yverdon-les-bains, le 31 juillet 2020

Jael Dubey

Cahier des charges

Résumé du problème

DEPsys est une entreprise Suisse leader technologique du marché énergétique. Fondée en 2012 et basée à Puidoux, elle fournit des solutions évolutives basées sur sa plateforme GridEye permettant aux réseaux de distribution d'énergie traditionnels de faire face aux nouvelles contraintes de la production décentralisée des sources d'énergie renouvelable, tels que les systèmes photovoltaïques et les technologies de mobilité électrique.

La plateforme GridEye offre une solution technologique innovatrice pour les gestionnaires de réseau de distribution (GRD, p. ex. Romande Énergie). Positionné de manière unique avec sa simplicité de déploiement, c'est la seule solution réellement Plug & Play qui évite tous les problèmes d'installation. La surveillance du réseau électrique en temps réel et les statistiques fournissent des informations détaillées sur les conditions du réseau. Les algorithmes de contrôle et de gestion garantissent la qualité et la stabilité du réseau.

Ce Travail de Bachelor consiste en une évaluation de différentes technologies de logging et de l'analyse des performances, pour ensuite mettre en place la meilleure solution pour l'infrastructure GridEye.

Étapes de réalisation du projet

- Étude des systèmes de gestion de logs actuels.

  Choix des systèmes à évaluer.
- Évaluation théorique des systèmes.
- Prise en main et configuration des outils pour une comparaison théorique & pratique.

  Choix d'un système.
- Implémentation d'un cas d'utilisation concret fourni par DEPsys.

  Avec un programme de simulation minimaliste de la plateforme GridEye.
- Implémentation d'une démonstration de faisabilité (Proof of Concept, PoC).
- Implémentation de librairies (SDK) pour l'interfaçage avec le Back-End de la solution GridEye (puis Front-End, puis outils externes).

Informations diverses et technologies

Les logs sont constitués de plusieurs types (System, User Action, Notification, ...) avec différents niveaux de priorités (debug, info, ...). Les logs utilisateurs doivent pouvoir être consultés depuis le Front-End, alors que les logs systèmes peuvent être accessible depuis une dashboard interne. Les notifications, dépendant du niveau de priorité, doivent pouvoir être transmises en temps réel aux gestionnaire de réseau de distribution électrique par e-mail, notification push, etc. ou sous forme de rapport journalier/hebdomadaire. La mise en place de librairies (SDK) est nécessaire pour communiquer avec les différents composants de l'infrastructure. Le but étant de pouvoir s'interfacer avec l'infrastructure actuelle afin de pouvoir remplacer la solution de logging actuelle.

Technologies :

- Base de données : SQL, JSON, ... (dépendant de la solution choisie).
- Back-End : Java
- Front-End : Javascript
- Outils externes : Python 3

Table des matières

1. Introduction

La plateforme GridEye de DEPsys [7] est une solution permettant d'optimiser les réseaux électriques. Elle s'adresse aux gestionnaires de réseau de distribution, et permet de faire face aux contraintes de la production décentralisée, qui correspondent par exemple aux panneaux photovoltaïques, ou encore aux batteries de stockage. Autant d'équipements que de plus en plus de particulier se procurent, tout autant que les entreprises. Cette transition énergétique est accélérée en Suisse avec la « Stratégie Energétique 2050 »[8], un texte de loi accepté par le peuple en 2017, qui promeut les énergies renouvelables.

Concrètement, GridEye est une solution qui peut se greffer à tout système déjà existant, ce qui évite tous les problèmes liés à l'installation. GridEye permet une optimisation efficace du réseau grâce à plusieurs applications, ayant principalement pour but d'assurer une grande qualité et une haute fiabilité. Ces qualités sont atteintes avec des outils de surveillance du réseau, d'analyses des événements via des algorithmes sophistiqués, et d'autres fonctionnalités diverses. Le service de GridEye fourni également l'accès à tout un support hautement qualifié. La figure 1 montre les différentes applications de GridEye, ainsi que ses composants et services.

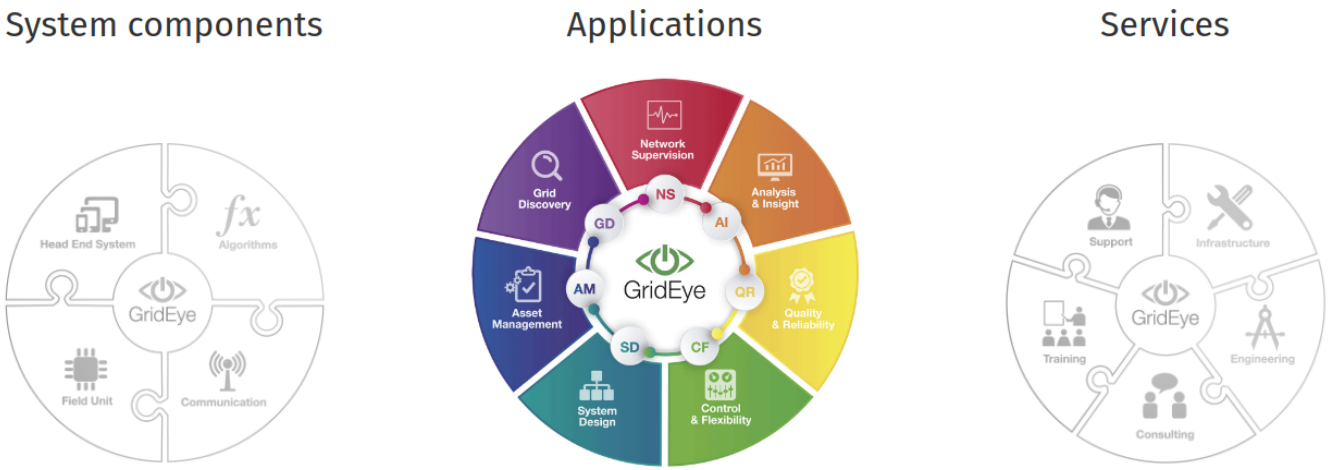


FIGURE 1 – Description de la plateforme GridEye [7]

Ce Travail de Bachelor se positionne dans les applications d'analyse et de surveillance du réseau électrique. Le but est, à terme, de remplacer la solution actuelle par un système de gestion de logs. Ce but sera atteint à travers plusieurs étapes. Tout d'abord, il s'agira de réaliser une étude de l'état de l'art en matière de système de gestion de logs. Il faudra voir quels sont les logiciels les plus utilisés actuellement et juger leurs qualités ou inconvénients respectifs. Cette étude permettra de choisir quelques systèmes à évaluer. L'évaluation de ces systèmes consistera en une recherche approfondie de leurs capacités et fonctionnalités théoriques, puis d'une prise en main et d'un essai concret de chacun. Après toutes ces investigations et considérations, il sera possible de prendre une décision de manière convaincue quant au logiciel à utiliser pour le futur de la plateforme GridEye. Après avoir fait le choix de l'outil, il faudra s'assurer de sa pertinence dans notre domaine de travail au travers de l'implémentation d'un cas d'utilisation se rapprochant de la réalité. Cette implémentation pourra être vue comme une démonstration de faisabilité. Cette étape supplémentaire pourra, dans le meilleur des cas, nous conforter dans notre choix basé sur les prises en main et fonctionnalités théoriques, ou alors, dans le cas contraire, nous faire changer d'avis et nous raviser sur un autre choix.

Dans tous les cas, le choix sera ici clairement défini et nous pourrons nous lancer dans un développement plus effectif, qui sera utilisé dans l'infrastructure GridEye. Il sera possible de développer des algorithmes permettant de détecter efficacement des anomalies dans le réseau, et si possible de les prévoir, afin de toujours assurer une fiabilité aussi souveraine que possible. Il faudra finalement interfacer la nouvelle solution avec la solution GridEye actuelle.

2. Comparaison des outils

2.1. Critères de comparaison

Pour réaliser une évaluation de différents système de gestion de log, il faut obligatoirement choisir des critères de comparaison. Ces critères se basent sur deux sources. La première étant les demandes formulées par DEPsys, et la deuxième provient des différentes fonctionnalités nécessaires à un système de gestion de logs [53] [12]. Voici les critères retenus :

- La collecte des logs
  Approche minimaliste ou maximaliste, est-ce que le système installe un agent sur le dispositif émetteur de log qui lui envoie uniquement les informations les plus importantes (approche minimaliste, méthode PUSH), ou est-ce que le système reçoit tous les logs et les enregistre tous (approche maximaliste, méthode PULL) ?

- L'agrégation centralisée des logs
  L'agrégation des logs est un défi, car après avoir collecté les logs, il faut tous les regrouper dans un même endroit, alors qu'ils peuvent avoir des formats différents. De plus, ils peuvent être généré très rapidement (s'exprime en EPS, Event Per Second), il faut donc être capable de traiter et regrouper ces logs de manière efficace.

- Le stockage à long terme et la durée de rétention des logs
  Après avoir agrégé ces informations, il faut maintenant faire des choix quant à leur stockage. L'idéal serait de garder tous les logs indéfiniment, mais chaque information stockée a un coût. Il faut donc avoir une stratégie de rétention qui permette de supprimer ou garder tel type de log.

- La rotation des fichiers de logs
  La rotation consiste à rendre automatique la stratégie de rétention et/ou de stockage des logs.

- L'analyse des logs (en temps réel et en vrac après une période de stockage)
  L'analyse des logs est, en quelque sorte, le but de tout le système de gestion de logs. En effet, il ne sert à rien de stocker de l'information sur un système si l'on en fait rien. L'analyse est donc la pour synthétiser les informations contenues dans les logs.

- Les rapports
  Il doit être possible pour un système de gestion des logs d'effectuer des recherches sur les informations stockées et de rédiger des rapports.

- Visionnage et gestion des alertes
  Un des buts d'un système de gestion de logs est de pouvoir réagir très vite à un problème, voire même de l'anticiper. Ceci passe par une émission d'alerte et d'une possibilité de visionnage des données, si possible en temps réel.

- Popularité
  Voici un critère qui change en permanence, mais qui a son importance lors d'un choix d'outil informatique. En effet, un logiciel sur le déclin sera de plus en plus dur à supporter, alors qu'un outil trop jeune n'a souvent que trop peu d'utilisateurs qui pourraient partager leurs connaissances sur les forums. L'idéal étant donc un logiciel populaire et qui est en pleine croissance.

En plus de ces critères, les coûts d'utilisation seront, si possible, évalués.

2.2. Choix des différents systèmes à évaluer

Comme pour le choix des critères de comparaison, les différents systèmes qui vont être analysés ont été définis soit par DEPsys, soit par des recherches dans les nombreux classement de « Log Management Tools »disponibles sur internet.
Quatre classements différents ont été sélectionnés afin d'avoir plusieurs avis différents, tout en restant dans une quantité raisonnable d'analyses. Les classements qui allaient être utilisés devaient être neutres. On entend par là que le site réalisant le classement ne doit pas proposer, par exemple, une solution cloud utilisant un certain outil de gestion de log, auquel cas son classement serait forcément biaisé. Il fallait également que le classement soit récent, étant donné la vitesse d'évolution générale des outils informatiques. Une période d'un an maximum a été définie.

Les classements suivants ont été choisis :

- Top 8 BEST Log Management Software [51]

  Classement datant de mars 2020, et publié par *SoftwareTestingHelp*.

- Best Log Manager & Monitoring Software & Tools [6]

  Classement datant de janvier 2020, et publié par *iTT Systems*.

- 6 Best Log Management Tools [2]

  Classement datant de août 2019, et publié par *AddictiveTips*.

- 13 Best Log Management & Analysis Tools [45]

  Classement datant de août 2019, et publié par *Comparitech*.

Les quatre sites internet retenus sont de simples portails informatiques, publiant des articles sur divers sujets informatiques. Lorsque l'on effectue une recherche Google [13] afin de trouver des classements de système de gestion de logs, on tombe régulièrement sur des articles de *DNSStuff* [9] et *Sematext* [47]. Ceux-ci n'ont pas été utilisé car le premier appartient à SolarWinds [28], une entreprise proposant plusieurs logiciels, dont des systèmes de gestion de logs, et le second propose des solutions cloud utilisant des outils de gestion de logs. Ils ont donc été jugés non-neutre.

Afin de réaliser un classement regroupant le contenu de ces 4 tops, une note a été attribuée à chaque position de chaque classement. Deux critères ont été intégré dans le calcul de la note : la position (mieux on est positionné dans son propre classement, plus on aura de points), ainsi que le nombre total d'outils cités. Il est en effet plus facile d'être bien classé dans un classement contenant 5 outils que dans un classement en contenant 10. Pour finir, afin de faciliter la lecture, la note attribuée se situe entre 0 (le plus mauvais), et 100 (la meilleure note possible). La formule suivante a donc été appliquée :

$$\frac{i * 100}{n}$$

i étant la position dans le classement, et n le nombre total d'outils dans le classement.

Les tableaux 1 et 2 montrent les points obtenus par chaque système.

SoftwareTestingHelp	
Système	Points
SolarWinds Log Analyzer	100
Sematext Logs	89
Splunk	78
ManageEngine EventLog Analyzer	67
LogDNA	56
Fluentd	44
Logalyze	33
Graylog	22
Netwrix Auditor	11

(a) Classement de SoftwareTestingHelp

Comparitech	
Système	Points
ManageEngine EventLog Analyzer	100
SolarWinds Papertrail	92
Loggly	83
PRTG Network Monitor	75
Splunk	67
Fluentd	58
Logstash	50
Kibana	43
Graylog	33
XpoLog	25
ManageEngine SyslogForwarder	17
TekWire Managelogs	8

(b) Classement de Comparitech

TABLE 1 – Classements SoftwareTestingHelp et Comparitech

AddictiveTips	
Système	Points
SolarWinds Papertrail	100
SolarWinds Loggly	83
Splunk	67
Nagios Log Server	50
Graylog	33
ManageEngine EventLog Analyzer	17

(a) Classement de AddictiveTips

iTT Systems	
Système	Points
SolarWinds Log & Event Manager	100
PRTG Network Monitor	83
Lepide	67
McAfee Enterprise Log Manager	50
Veriato	33
Splunk	17

(b) Classement de iTT Systems

TABLE 2 – Classements AddictiveTips et iTT Systems

Et le tableau 3 montre le classement global, calculé d'après l'addition des points obtenus dans les différents classements.

	Système	Points
1	Splunk	229
2	SolarWinds Papertrail	192
3	ManageEngine EventLog Analyzer	184
4	SolarWinds Loggly	166
5	PRTG Network Monitor	158
6	Fluentd	102
7	SolarWinds Log Analyzer	100
8	SolarWinds Log & Event Manager	100
9	ELK Stack (Kibana + Logstash)	93
10	Sematext Logs	89
11	Graylog	88
12	Lepide	67
13	LogDNA	56
14	Nagios Log Server	50
15	McAfee Enterprise Log Manager	50
16	Logalyze	33
17	Veriato	33
18	XpoLog	25
19	ManageEngine Syslog Forwarder	17
20	Netwrix Auditor	11
21	TekWire Managelogs	8

TABLE 3 – Classement global des systèmes de gestion de log

Après une étude légèrement plus approfondie sur ces différents systèmes, il a été décidé de garder les suivants pour la comparaison :

1. Elastic Stack [42]

   Choisi par DEPsys, probablement la plus populaire.

2. Splunk [48]

   1er du classement.

3. SolarWinds Loggly [27]

   De tous les outils appartenant à SolarWinds, je voulais n'en choisir qu'un. Loggly me paraissaît le plus approprié au cas d'utilisation de ce travail de Bachelor.

4. Graylog [15]

   Suggéré par DEPsys, et semble avoir une bonne documentation.

2.3. Comparaison théorique

2.3.1. Elastic Stack

La « Elastic Stack », ou « Suite Elastic »en français, anciennement appelée « ELK Stack »est composée de plusieurs outils :

- Elasticsearch [10]

  Un moteur de recherche RESTful.

- Kibana [25]

  Un outil de visualisation.

- Logstash [29]

  Un pipeline d'ingestion de log.

- Beats [3]

  Une famille d'agent dédié au transfert de données.

La table 4 montre un résumé des fonctionnalités disponibles ainsi que des coûts lié à la Suite Elastic.

Elastic Stack	
Collecte	La collecte des logs se fait en approche minimaliste. La suite Elastic contient l'outil Beat, qui est donc une famille d'agent léger. On installe un agent beat (p. ex. Filebeat, Metricbeat, etc.) sur le système générant les logs, et cet agent envoie les données vers le serveur.
Agrégation centralisée	Se fait via Logstash. Peux supporter beaucoup d'événements par seconde (> 10'000 EPS). Compatible avec énormément de type de logs. Permet d'analyser et transformer les logs en temps réel. Logstash dispose d'une API permettant de créer nos propres plug-in, si les sources de données ne sont pas compatible nativement.
Stockage et rétention	Le stockage se fait avec Elasticsearch. Il n'y a pas de rétention des données de bases avec Elasticsearch. Il est cependant possible de le faire avec Elastic-Curator, qui est un outil permettant de gérer un cluster Elasticsearch.
Rotation	La rotation se fait avec Elastic-Curator
Analyse	Elasticsearch et ses requêtes poussées permettent de faire des recherches avancées.
Rapport	Les rapports peuvent être générés depuis Kibana.
Visionnage et alertes	La visualisation des données en temps réels peut se faire avec Kibana. La gestion des alertes se fait également via Kibana. Il est possible de paramétrer des alertes classiques, qui se déclenchent suivant des règles précises. Et il est également possible de paramétrer des alertes suivant un algorithme d'apprentissage automatique, qui détectera des événements inhabituels.
Popularité	La suite Elastic est très certainement la plus populaire actuellement. Elle bénéficie d'une grande communauté active. Au niveau de la tendance, on peut voir une grande croissance entre les années 2016 et 2019. Ces derniers mois, cela semble se stabiliser.
Coûts	La suite Elastic propose différents abonnements. Il y a une offre gratuite, mais celle-ci ne contient pas de gestion d'alerte et de création de rapport. Les prix pour les offres payantes ne sont pas publics. Il faut contacter Elastic et les prix varient en fonction de la taille du système à implémenter.

TABLE 4 – Résumé théorique de Elastic Stack

2.3.2. Graylog

Graylog un outil de gestion de logs. Il dispose de deux versions : Open Source et Enterprise. La table 5 montre les fonctionnalités du systèmes Graylog.

Graylog	
Collecte	La collecte des logs se fait en approche minimaliste. Graylog possède un outil appelé « Sidecar »qui permet de gérer plusieurs type d'agent, y compris l'outil Beat de la suite Elastic.
Agrégation centralisée	Graylog permet de gérer « d'énormes »jeux de donnée et de les traiter selon des règles définies par l'utilisateur. En plus des règles d'agrégation classiques, comme l'origine géographique, le niveau d'alerte, etc., Graylog permet de faire des listes noires de logs.
Stockage et rétention	Le stockage se fait avec MongoDB et Elasticsearch. Graylog offre une solution (Graylog Archive) de rétention des données, disponible avec la version Enterprise, qui peut être paramétrée.
Rotation	La rotation se fait avec Graylog Archive.
Analyse	Graylog utilisant Elasticsearch, il dispose de ses requêtes poussées permettant de faire des recherches avancées. Graylog possède également son langage de requête, basé sur Apache Lucene.
Rapport	La création de rapport est une fonctionnalité de Graylog Enterprise. Il est possible de les configurer depuis l'interface de Graylog.
Visionnage et alertes	La visualisation des données en temps réels se fait avec l'interface de Graylog. La gestion des alertes est incluse dans le Graylog de base. Elle permet de définir des alertes selon des règles. Graylog possède également un « Store »proposant, entre autres, des fonctionnalités liées aux alertes, développées par la communauté.
Popularité	Graylog n'est pas arrivé très haut de manière générale dans les tops, mais il est en revanche souvent cité. La courbe de tendance de Graylog est en croissance régulière depuis 2008.
Coûts	Graylog propose deux versions : « Open Source »et « Enterprise ». La première est gratuite mais propose quelques fonctionnalités en moins, comme les rapports programmés ou le support technique. Les coûts de la version « Enterprise »ne sont pas disponibles, il faut contacter Graylog. À noter que la version « Enterprise »est gratuite jusqu'à une utilisation de 5 GB par jour.

TABLE 5 – Résumé théorique de Graylog

2.3.3. SolarWinds Loggly

SolarWinds Loggly un outil de gestion de logs SaaS (Software-as-a-Service, dans le cloud). Il dispose de plusieurs versions, dont une gratuite. La table 6 montre les fonctionnalités et des informations sur les coûts de SolarWinds Loggly.

SolarWinds Loggly	
Collecte	L'envoi de données à Loggly est relativement simple. La seule contrainte est qu'il s'agisse de texte. Il n'y a pas besoin d'avoir d'agent (envoi de log via un endpoint), mais il est également possible d'en utiliser.
Agrégation centralisée	Loggly permet de parser les logs des formats les plus connus, comme les logs MySQL ou Java, mais ne possède pas d'outil permettant de réaliser un filtrage et une agrégation de logs personnalisés. Pour ce faire, il préconise d'utiliser un logiciel comme Logstash.
Stockage et rétention	La rétention des données est plutôt courte (7 jours dans la version gratuite), et ensuite, les logs peuvent être sauvegardés dans une instance S3 de AWS.
Rotation	La rétention se fait automatiquement après un certain nombre de jour.
Analyse	Loggly possède son propre langage de requête, qui est basé sur Apache Lucene. Il est également possible d'analyser les logs en temps réel via l'interface de Loggly.
Rapport	Il est possible de réaliser des rapport dans plusieurs format depuis l'interface graphique.
Visionnage et alertes	Il est possible de configurer des alertes selon des règles classiques, et également sur des événements inhabituels. Le visionnage des données en direct se fait via l'interface graphique de Loggly.
Popularité	Loggly était en croissance entre 2008 et 2016, mais depuis cette année-ci, sa courbe de tendance est en décroissance.
Coûts	Loggly propose une version gratuite et trois versions payantes. La version gratuite est limitée à un volume de 200 MB par jour, un seul utilisateur pouvant se connecter sur une instance, et ne contient pas certaines fonctionnalités comme la gestion des alertes. Les versions payantes varient entre 79 et 279 USD par mois, et proposent chacune quelques fonctionnalités en plus, et un volume de moins en moins limité.

TABLE 6 – Résumé théorique de SolarWinds Loggly

2.3.4. Splunk

Splunk est un outil de gestion de logs. Il est séparé en trois « parties », chacune étant responsable de plusieurs choses :

- Universal Forwarder

  Effectue la collecte et envoie les données à l'indexer.

- Indexer

  Effectue le stockage des logs.

- Search Head

  Permet de lire dans l'index.

Chacune de ces fonctions peut être transformé en cluster si de grand volume de données sont à traiter. Splunk peut être disponible en version « sur site »ou « cloud ». La table 7 montre les fonctionnalités et les coûts relatifs à Splunk.

Splunk	
Collecte	La collecte des logs se fait en approche minimaliste. Ceci via les « Universal Forwarder »qui sont des agents à installer sur le système générant les logs. Il envoie ensuite les données vers l'indexer.
Agrégation centralisée	Pour Splunk, l'agrégation des logs ne se fait pas dans le pipeline lors de l'ingestion des logs, mais après coup. Cela se fait donc une fois que les logs ont été indexés. Il est possible de faire des filtres personnalisés ainsi que d'utiliser des filtres pour les formats courants, et ceci avec une fonctionnalité s'appelant « Field Extraction ».
Stockage et rétention	Le stockage des logs se fait avec l'indexer. Concernant la rétention des données, Splunk la gère avec des « buckets ». Concrètement, un bucket possède une durée qui détermine le temps qu'une donnée va passer dedans. P. ex., on peut avoir un bucket de 30 jours où les logs seront accessible et analysable librement. Puis, passé ces 30 jours, ils iront dans un autre bucket où ils seront compressé pour le stockage à long terme.
Rotation	Se fait via les buckets.
Analyse	L'analyse est possible via l'interface de Splunk. Celle-ci permet de trier et filtrer les logs selon de nombreux critères.
Rapport	La génération de rapport est possible depuis l'interface de Splunk.
Visionnage et alertes	Le visionnage et la gestion des alertes est également possible depuis l'interface de Splunk. Pour les alertes, elles sont définissables selon des règles classiques (p. ex. logs provenant d'une adresse IP particulière, etc.).
Popularité	D'après les courbes de Google Trends, le système Splunk est en constante croissance depuis 2010. Dans le classement des différents tops consultés, Splunk arrive en bonne position. Splunk bénéficie d'une documentation relativement grande et d'un forum.
Coûts	Les coûts sont en « infrastructure-based pricing »et ne sont donc pas fixe. Mais Splunk Enterprise commence à 150$ par mois. Splunk ne propose pas de version gratuite (mise à part l'essai de 60 jours).

TABLE 7 – Résumé théorique de Splunk

2.3.5. Popularité des systèmes

La comparaison de popularité des différents systèmes permet de se faire un idée sur les tendances actuelles et passées d'utilisation de ces systèmes. Elle a également pour but de pouvoir éventuellement anticiper les futures tendances. Ces données sont importantes lors du choix d'un logiciel, afin de pouvoir compter sur un support de la communauté lors des problèmes qui arriveront pendant le développement.

Conditions de tests

- Les courbes de tendances seront tirées de Google Trends [14].
- Les courbes porteront sur les 10 dernières années.

  Du 23 avril 2010 au 23 avril 2020
- Les courbes porteront sur les recherches dans tous les pays du monde.
- Il n'y aura pas d'autre restriction de recherche (catégorie, outil Google)

Résultats

La figure 2 montre les courbes de tendances des quatre systèmes. Les courbes de Loggly et Graylog étant particulièrement basses et donc difficilement comparable, la figure 3 permet de les comparer entre elles.

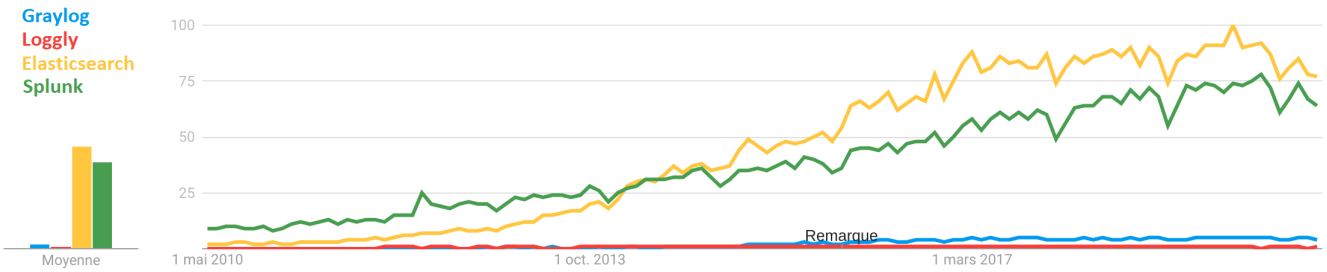


FIGURE 2 – Courbes Google Trends des quatre systèmes

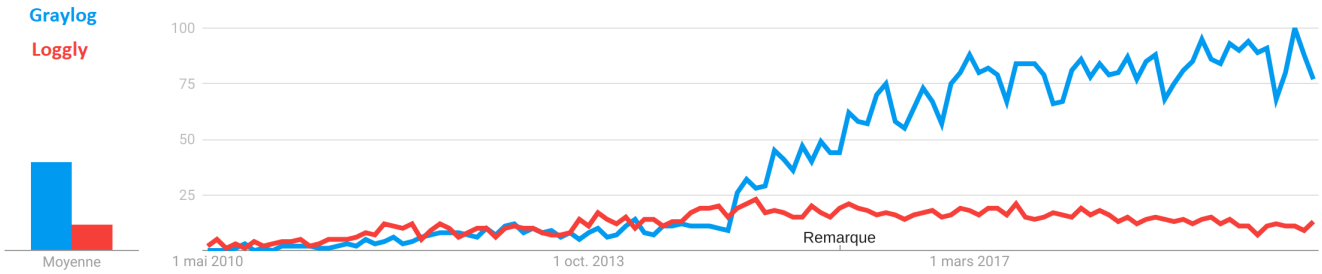


FIGURE 3 – Courbes Google Trends de Graylog et Loggly

La Suite Elastic contenant 4 logiciels séparés, le choix ici a été d'inclure uniquement « Elasticsearch », car c'est la plus connue des quatre applications, et que les résultats sont meilleurs qu'avec le terme « Elastic Stack »(il faut également savoir que la Suite à changé de nom en 2016, passant de « ELK Stack »à « Elastic Stack », ce qui ne favorise pas la recherche dans Google Trends).

Conclusion
On distingue deux groupes dans cette analyse : Elasticsearch et Splunk d'un côté, Graylog et Loggly de l'autre. Le premier est très populaire et en croissance, alors que le deuxième n'est que peu tendance, et Loggly n'est pas en croissance. Au sein même du groupe Graylog-Loggly, on peut encore constater une grande différence, Graylog étant beaucoup plus recherché sur Google que Loggly. On peut donc affirmer que la Suite Elastic est le système le plus populaire, devant Splunk. Loin derrière vient Graylog et encore plus loin, on trouve Loggly.

2.4. Tests réels et prise en main des logiciels

Plusieurs tests ont été effectués afin de pouvoir effectuer une comparaison de ces systèmes. Comme la plupart de ces tests obligent une prise en main concrète des logiciels, ceux-ci ont également permis d'avoir une idée de la facilité d'utilisation, de la flexibilité ainsi que de l'utilisabilité (« user-friendliness ») des systèmes.

Voici les versions des logiciels utilisés pour effectuer ces tests :

- Elastic Stack

  Elasticsearch 7.6.2

  Filebeat 7.6.2

  Logstash 7.6.2

  Kibana 7.6.2

- Graylog

  Graylog 3.2.4-1 (Virtual Appliance)

  Filebeat 7.6.2

- Splunk

  Splunk 8.0.2.1

  Splunk Universal Forwarder 8.0.2.1

- Loggly

  Loggly Lite (Cloud)

2.4.1. Test de débit d'ingestion de logs

Le premier test de cette comparaison consiste en un test de débit. Cela permet d'avoir une première idée de la performance des applications.

Conditions de tests

- Chargement d'un fichier de logs.

  Le fichier contient 98'305 logs identiques.

  Log : 2020-02-27 09 :06 :24.596 INFO o.s.s.c.ThreadPoolTaskScheduler -> Shutting down ExecutorService 'taskScheduler'

- On effectue un test d'ingestion simple (on stocke le log brut).
- On effectue un test d'ingestion avec filtre (on parse les différentes parties du log).

  Date et heure : 2020-02-27 09 :06 :24.596

  Niveau du log : INFO

  Classe Java : o.s.s.c.ThreadPoolTaskScheduler

  Message : Shutting down ExecutorService 'taskScheduler'

- Le résultat est un débit exprimé en EPS (Event Per Second), qui correspond au nombre de logs que le système aura pu ingéré en une seconde.

La table 8 montre **les résultats** des tests d'ingestion.

	Elastic Stack	Graylog	Splunk	Loggly
Sans filtrage	4'898	14'044	-	46'295 (*)
Avec filtrage	9'181	7'868	2'628	-

TABLE 8 – Résultats des tests de débit d'ingestion

(*) Loggly n'acceptant que des fichiers de taille inférieure à 5 MB, le fichier à été réduit à 3'999 logs.

Splunk favorise l'extraction d'informations (parsing) après avoir stocké les logs dans le système. Il est alors compliqué d'ajouter un filtrage avant le stockage et ça n'a donc pas été réalisé. Par contre, Splunk effectue automatiquement une extraction de la date des logs. Son débit à donc été classé dans la catégorie « avec filtrage ». Loggly ayant été écarté assez tôt de l'évaluation, la partie « avec filtrage »n'a pas été testée.

Conclusion
Sans filtrage, on constate que Graylog possède un débit supérieur à Elastic Stack. Loggly n'acceptant pas des fichiers de grande taille, il est difficilement comparable. Avec filtrage, étonnemment, la Suite Elastic gagne en débit. Elle possède donc un meilleur débit que Graylog et Splunk. Après ce test, outre les résultats comptables, il se dessine surtout une tendance vers Graylog et Elastic Stack, car ces deux systèmes sont plus « ouvert »que les autres. Il n'y a pas eu de difficultés particulières lors de l'utilisation, pas de contraintes comme la taille du fichier ou encore le moment du filtrage au sein du pipeline.

2.4.2. Test d'ingestion d'une grande quantité de logs

Ce test a pour but de mettre en évidence les limites de certains systèmes lors de l'ingestion de gros fichier de logs. Il permet également d'avoir une brève vue sur la constance du débit mesuré lors du test précédent.

Conditions de tests

- Chargement d'un fichier de logs.

  Le fichier contient 999'999 logs identiques.

  Log : 2020-02-27 09 :06 :24.596 INFO o.s.s.c.ThreadPoolTaskScheduler -> Shutting down ExecutorService 'taskScheduler'
- On n'effectue aucun traitement sur le log.
- Le résultat est soit une limite supérieure, soit « > 1'000'000 »

La table 9 montre **les résultats** du test d'ingestion d'une grande quantité de logs.

Elastic Stack	Graylog	Splunk	Loggly
> 1'000'000	> 1'000'000	> 1'000'000	~4'000 (5 MB)

TABLE 9 – Résultats des tests d'ingestion de grande quantité de logs

Conclusion
Ce test permet simplement de montrer que les systèmes travaillant avec un agent, soit selon la méthode PUSH, n'ont pas de problème de taille de fichier, car l'agent lit et envoie les logs au fur et à mesure. Loggly est le seul ici utilisant la méthode PULL. Au niveau des débits, par rapport au test d'ingestion du paragraphe 2.4.1, ceux d'Elastic Stack et de Graylog ont été divisés par 2, alors que celui de Splunk est resté stable. Pour Loggly, il n'y a pas eu de différence, étant donné qu'il est limité en taille de fichier.

2.4.3. Test de consommation CPU

Ce test-ci a pour but de pouvoir comparer les systèmes de gestion de logs Elastic Stack et Graylog en terme de performance.

Conditions de test

- Chargement du même fichier de log que pour le test de débit du paragraphe 2.4.1.
- L'utilisation du CPU sera réduite au minimum hors système à tester.
- Ordinateur de test : ASUS UX360UAK
- Processeur : Intel Core i7-7500U CPU @ 2.70GHz x 4
- Lors de la réception, l'agent Filebeat se situera sur une autre machine.

Résultats

La figure 4 montre la consommation CPU et l'historique du trafic réseau lors de la réception des logs pour le système Elastic Stack. La figure 5 montre les mêmes informations, mais pour le système Graylog. Enfin, la figure 6 montre ces informations-ci avec l'agent Filebeat envoyant des logs.

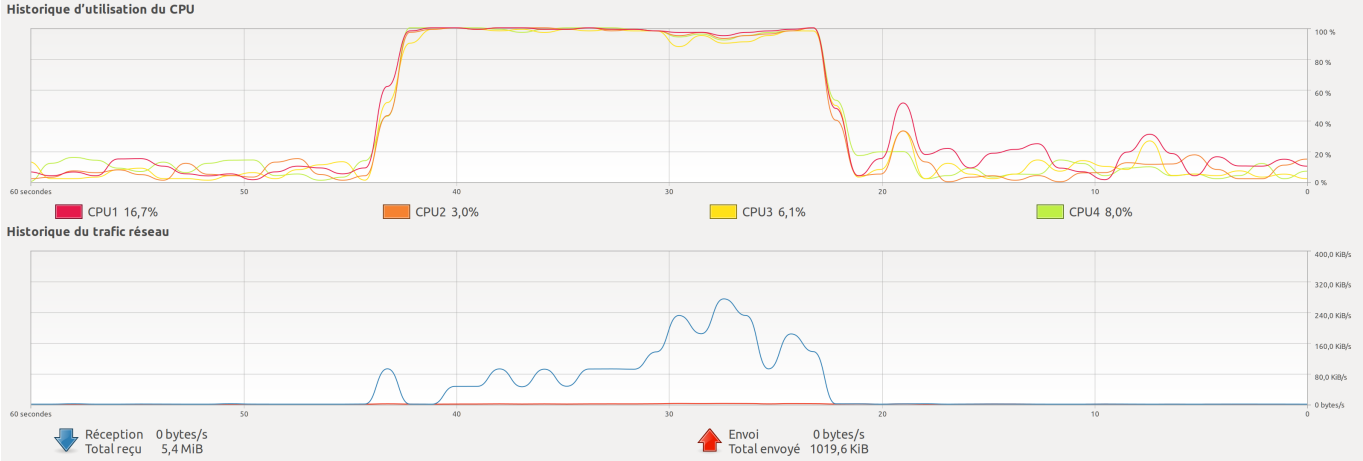


FIGURE 4 – Consommation CPU Elastic Stack

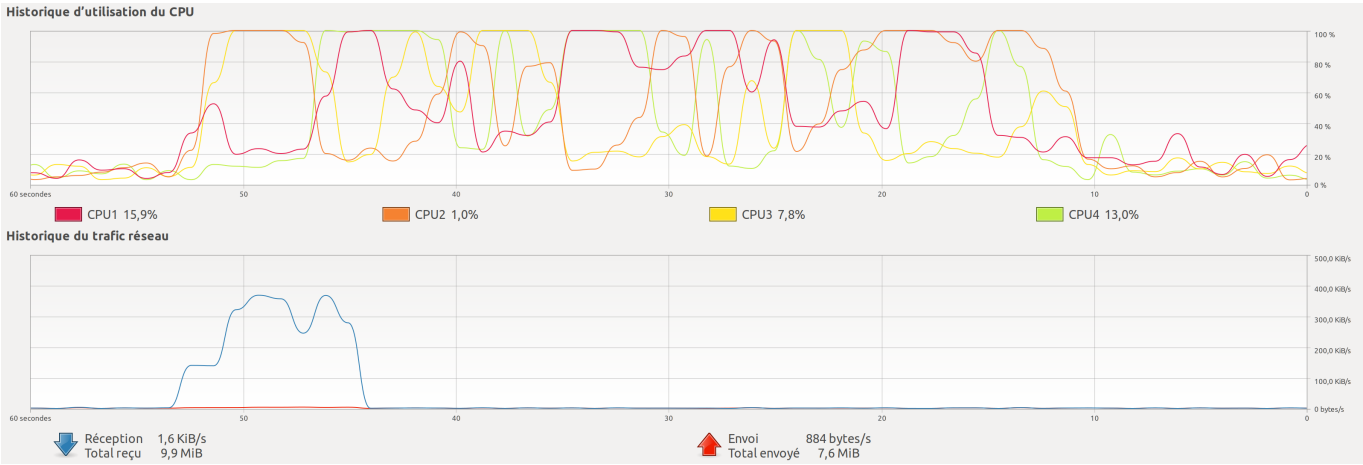


FIGURE 5 – Consommation CPU Graylog

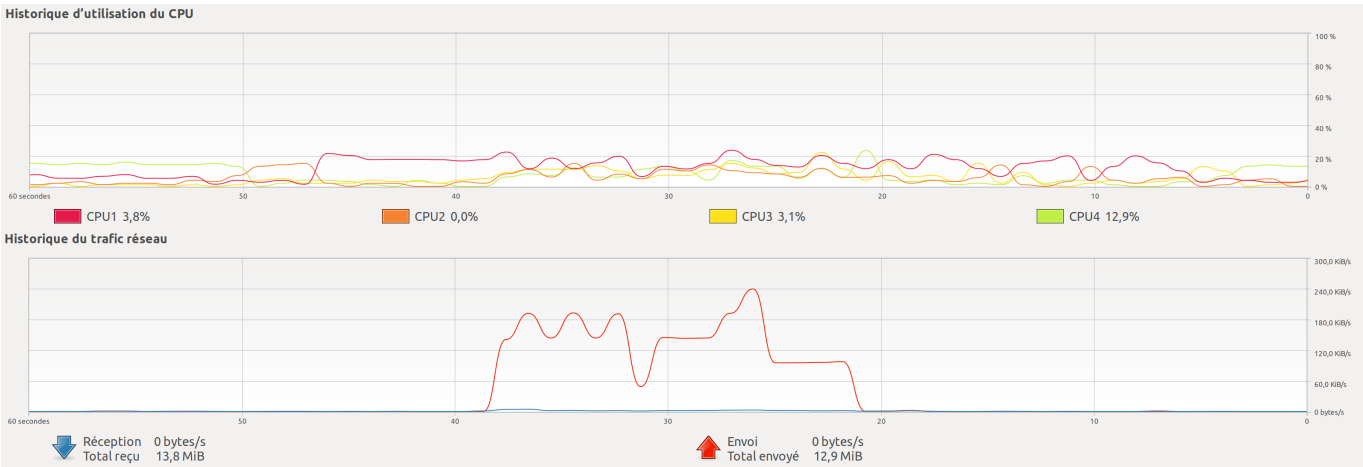


FIGURE 6 – Consommation CPU Filebeat

Conclusion
Premièrement, on constate que l'agent Filebeat est un agent très léger puisque son action ne se fait quasiment pas ressentir sur le taux d'utilisation du CPU. Pour ce qui est du taux d'utilisation CPU par la Suite Elastic, on remarque que lors de la réception et le stockage des logs, tous les CPUs tournaient à 100% de leur capacité, et la charge dure environ 25 secondes. Pour Graylog, les CPUs ne sont pas autant sollicité, ou du moins pas tous en même temps. Ils varient entre 20% et 100% et la distribution est plutôt égale entre les CPUs. On peut donc estimer un taux d'utilisation moyen à 60-70%. La charge dure cette fois-ci environ 45 sec. À noter que Graylog ingère les logs en env. 7 secondes, puis travaille encore pendant env. 35 secondes, alors que la Suite Elastic fait tout en même temps. On peut constater ceci grâce à la courbe de l'historique du trafic réseau.

2.5. Conclusion de la comparaison des outils

2.5.1. Analyse théorique des systèmes

Au niveau fonctionnel, les quatre systèmes de gestion de logs ne se différencient pas énormément. Ils suivent à peu près tous le même schéma et permettent tous de faire les mêmes actions (reporting, alerting, analyse, etc.). Loggly est légèrement différent dans le sens où son approche de collecte de log par défaut est maximaliste (approche PULL). Il met en effet un endpoint à disposition, sur lequel on peut envoyer des données. Il s'agit la d'une différence mineure car bien que les autres systèmes privilégient et facilitent l'approche minimaliste, il est également possible de configurer un endpoint semblable pour les systèmes Elastic Stack, Graylog et Splunk.
Pour ce qui est du non-fonctionnel, il y a quelques points où les systèmes se démarquent. Premièrement, Loggly de SolarWinds est uniquement disponible dans une version Cloud. La Suite Elastic et Splunk sont eux disponibles tant dans une version Cloud que dans une version sur site. Graylog lui est uniquement disponible dans une version locale. Il y a également l'architecture des systèmes. La Suite Elastic est formée de 3 applications indépendantes, et en utilise une $4^{\text{ème}}$ pour l'agent Beat. Graylog est composé d'un système, et utilise des agents ne lui appartenant pas, comme Beat. Splunk suit la même architecture que Graylog, mais possède son propre agent, le Splunk Forwarder. Et Loggly est une infrastructure cloud sans agent.
Reste la comparaison de la popularité des applications. Celle-ci a montré une nette avance pour Splunk et Elastic Stack, devant Graylog, puis Loggly qui est vraiment peu populaire.
La table 10 résume ces différentes propriétés.

	Elastic Stack	**Splunk**	**Graylog**	**Loggly**
Fonctionnalités	Reporting, alerting, analyse, etc.			
Cloud / On-Prem	Cloud ou Sur site		Sur site	Cloud
Architecture	Suite (3 outils + agent)	Un système + agent		Cloud (sans agent)
Popularité	Très populaire		Peu populaire	Très peu populaire

TABLE 10 – Résumé de la conclusion de l'analyse théorique

Globalement, l'analyse théorique des systèmes de gestion de logs ne permet pas de faire un choix concret sur un système ou un autre pour la réalisation d'une solution destinée au système GridEye. Elle permettrait tout au plus de se détacher de Loggly, qui n'est vraiment pas recommandable du point de vue de sa courbe de tendance.

2.5.2. Analyse pratique des systèmes

Le but des tests réels était de pouvoir confronter les systèmes de gestion de log au niveau de leur performance, et également de réaliser une prise en main des différents logiciels, avec tout ce que cela implique (installation, configuration, etc.). Cette deuxième partie avait également pour but d'évaluer l'utilisabilité des divers systèmes. Au niveau des performances, on a pu voir que les systèmes Graylog et Elastic Stack se distinguaient

légèrement avec un meilleur débit d'ingestion. Que tous les systèmes, excepté Loggly, pouvaient, grâce à leur approche minimaliste, recevoir des grandes quantités de logs.
À ce point-ci de la comparaison, un premier bilan de l'« user-friendliness »des systèmes à également pu être tiré :

- SolarWinds Loggly est relativement simple d'utilisation. Sa caractéristique Cloud facilite également l'installation (il n'y en a pas, il suffit de se créer un compte et nous recevons un lien vers notre version d'essai).
- Graylog est un système ayant une interface plutôt complète et efficace. L'interface graphique permet directement de configurer le système de manière simple (elle permet d'ajouter des règles d'extraction d'informations dans les logs, de configurer les entrées ouvertes aux agents, etc.).
- Elastic Stack est une « suite »de logiciels, ce qui la rend très ouverte. On peut en effet choisir d'utiliser ses softwares de base, mais également utiliser d'autres applications. Par exemple, on peu prendre un agent de la famille Beat, mais on pourrait aussi choisir un agent Prometheus. Chaque logiciel est configurable via des fichiers de configuration. On peut également choisir d'utiliser les 4 logiciels, ou pas. Par exemple enlever Logstash si l'on a pas besoin de ces filtres. Cette ouverture le rend agréable à utiliser.
- Splunk est comparable à Graylog et Elastic Stack dans tous les tests, mais l'expérience utilisateur est de son côté moins bonne. Le système est plus une sorte de « boîte noire »dans laquelle il n'est pas possible de tout configurer. L'extraction des informations des logs n'a par exemple pas pu avoir lieu avant l'indexage, car le système est prévu pour le faire après.

Avec ces premiers tests, nous avons déjà pu écarter la solution SolarWinds Loggly, qui ne correspondait pas aux attentes et dont les contraintes liées au Cloud ne sont pas souhaitées. Splunk n'est également pas recommandé à cause de son manque d'ouverture. Ensuite, il y a encore eu le test d'utilisation du CPU entre la Suite Elastic et Graylog. Avec celui-ci, on pourrait donner un très léger avantage à Graylog.

2.5.3. Recommandation

En conclusion, il est difficile de faire un choix entre les deux systèmes Graylog et la Suite Elastic. Mais de part sa très grande popularité, amenant son lot de réponse dans les forums et autres articles permettant d'accélérer le développement d'un logiciel, ainsi que de son caractère très ouvert lié à son « architecture suite », la **Suite Elastic** est le meilleur choix dans l'optique de mise en place d'une solution de gestion de logs pour l'infrastructure GridEye.

3. Implémentation du cas d'utilisation

Un cas d'utilisation à implémenter à été fourni par DEPsys et demande les éléments suivants :

Le système générant les logs sera composé d'une base de donnée PostgreSQL [37] et d'une application Java [24] simulant différents comportements, comme une surcharge du CPU, de la RAM ou autres problèmes.

Le système Elastic Stack devra mesurer plusieurs métriques :

- Redémarrage de conteneur.
- Taux d'utilisation de la mémoire.
- Occupation de la base de données (PostgreSQL).
- Taux d'utilisation du CPU.
- Les crashs Java
- Et éventuellement d'autres métriques comme les logs de l'application Java.

Plusieurs instances peuvent être lancées en même temps, ce qui permet de simuler au mieux la réalité (il y a plusieurs « field device »).

Il doit être possible de savoir si le système est fonctionnel, et à quel point.

Pour répondre à toutes ces demandes, on va commencer par installer et configurer tous les outils de la Suite Elastic qui seront utilisés, et ce avec Docker [11]. Ensuite, nous allons créer des visualisations avec Kibana. Elles permettront de connaître l'état du système en temps réel, et de retrouver la cause des problèmes, s'il y en a.

3.1. La Suite Elastic avec Docker

Afin de faciliter les étapes d'installation, de configuration et de lancement de la Suite Elastic, une version « dockerisée »des différents logiciels est une solution idéale. Elle permet également d'uniformiser ces étapes-ci. En effet, l'installation des applications de la Suite Elastic sans conteneur fonctionnera de la même manière sur des systèmes Ubuntu, voire Linux, mais sera différente sur un système d'exploitation Microsoft Windows par exemple. L'utilisation de conteneurs comble ces problèmes.

L'installation à été séparée en deux, car deux machines sont nécessaires pour simuler l'interface GridEye : une machine serveur, recevant les logs, et une machine client, envoyant les logs.

3.1.1. Machine hôte de « ELK »

Par « ELK », qui est l'ancien nom de la Suite Elastic, on parle des trois outils Logstash, Elasticsearch et Kibana. C'est la machine qui ne sera pas sur le terrain et qui permettra l'agrégation, le stockage et l'analyse des logs, entre autres. La figure 7 montre un schéma de la topologie Docker prévue permettant de faire fonctionner ensemble les différents outils de la Suite Elastic.

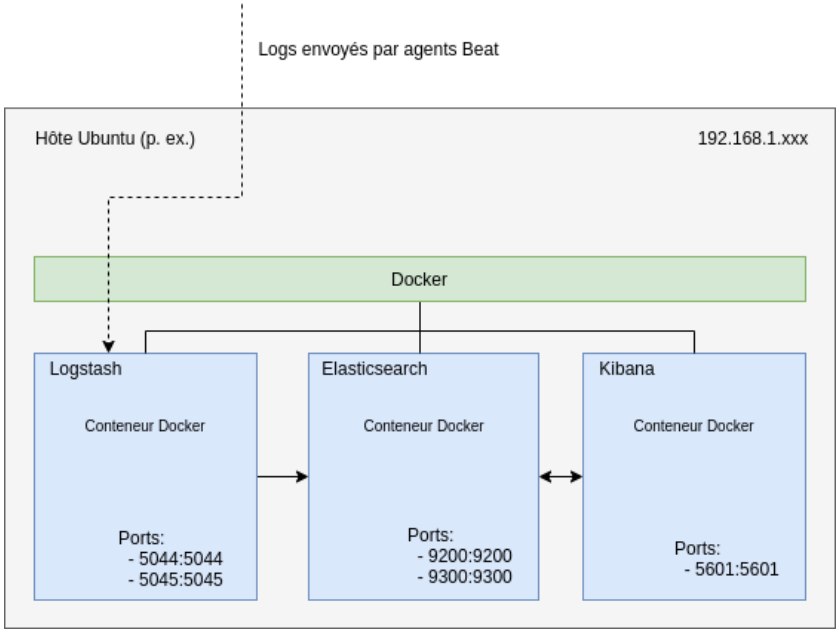


FIGURE 7 – Architecture de l'implémentation du use-case - côté « ELK »

Comme il y a plusieurs conteneurs Docker devant être créés, nous allons utiliser l'outil `docker-compose` [35] qui permet de gérer et d'orchestrer de multiples conteneurs. Nous aurons trois images à aller télécharger : Logstash, Elasticsearch et Kibana. L'agent Beat se situant sur un ordinateur distant, il n'est pas inclus dans cette topologie. La figure 8 montre l'arborescence des dossiers et fichiers nécessaires.

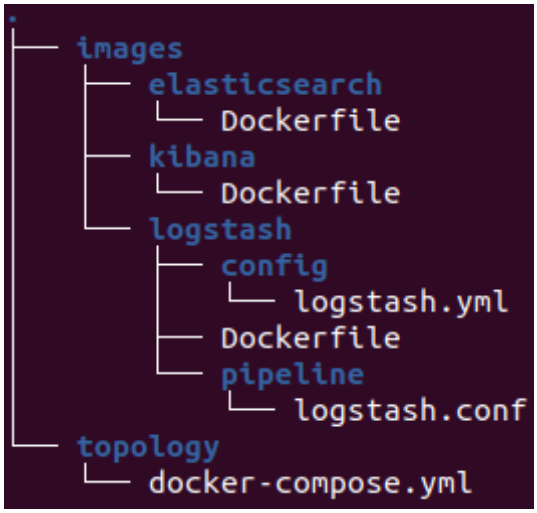


FIGURE 8 – Arborescence de la topologie Docker - côté ELK

Les `Dockerfile` des images Elasticsearch et Kibana sont très simples, ils vont simplement télécharger les images correspondantes depuis le `Elastic Docker registry`.
`Dockerfile` d'Elasticsearch :

```
1 FROM docker.elastic.co/elasticsearch/elasticsearch:7.6.2
```

`Dockerfile` de Kibana :

```
1 FROM docker.elastic.co/kibana/kibana:7.6.2
```

Pour l'image de Logstash, il y aura deux fichiers de configurations. Un pour le « pipeline », qui contiendra les configurations d'entrées, filtres et sorties, et un pour les paramètres de l'outil.
Fichier de configuration des paramètres (`logstash.yml`) :

```
http.host: "0.0.0.0"
xpack.monitoring.elasticsearch.hosts: [ "http://elasticsearch:9200" ]
```

La configuration du pipeline se fait dans un fichier `.conf` nommé `logstash.conf` et qui ira se copier à l'adresse `/usr/share/logstash/pipeline/` du système de fichier du conteneur. Le fichier peut avoir un nom différent, tant qu'il porte l'extension `.conf`.

Le fichier de configuration de Logstash est divisé en trois parties :

1. input

   Cette première partie permet de configurer l'arrivée des logs. Il est possible d'y configurer un agent Beat, ce qui nous concernera ici, mais également la lecture directe d'un fichier, ou de beaucoup d'autres sources, comme Twitter, JDBC, etc.

2. filter

   La partie filtre permet quant à elle de parser les logs. Il existe là aussi beaucoup de filtres différents. Nous utiliserons une structure `grok` qui permet de parser des événements non structurés.

3. output

   La sortie sert à choisir où Logstash enverra ses logs traités. En général, il s'agit d'Elasticsearch.

Pour la configuration de l'entrée, nous allons simplement écouter deux agents Beat, chacun sur un port donné. Nous allons également ajouté un tag `filebeat` qui sera utilisé pour la partie « filtre ».

```
input {
    beats {
        port => 5044
        tags => ["filebeat"]
    }

    beats {
        port => 5045
    }
}
```

En ce qui concerne le filtre, nous allons parser les logs envoyés par le logiciel de simulation de GridEye. Le format des logs est défini par la ligne suivante :

```
%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %logger{100} -> %msg%n
```

Le site `grokconstructor` [50] permet de créer des `grok pattern` de façon incrémentale. Il suffit de lui donner quelques lignes de logs générés par notre application et de le configurer. Il va ensuite nous proposer des motifs qu'il détecte dans les lignes de logs données. Voici le résultats après lui avoir donné quelques lignes générés par le simulateur :

```
\A%{TIMESTAMP_ISO8601}%{SPACE}%{LOGLEVEL}%{SPACE}%{JAVACLASS} -> %{JAVALOGMESSAGE}
```

À savoir que le tout premier champ, qui a été défini comme un `TIMESTAMP_ISO8601`, n'en est pas vraiment un [20]. Mais c'est le motif qui correspond le mieux à notre format de date-heure. Il reste à définir des labels décrivant les différents motifs. Ces labels seront utilisés par Kibana.

```
\A%{TIMESTAMP_ISO8601:date_hour}%{SPACE}%{LOGLEVEL:log_level}%{SPACE}%{JAVACLASS:class} ->
    %{JAVALOGMESSAGE:message}
```

Comme ce `grok pattern` permet de parser uniquement nos logs applicatifs, il faut ajouter une condition, qui indique que si les logs viennent d'ailleurs (p. ex. de la base de données PostgreSQL), ils ne doivent pas être parsés par ce filtre. Nous allons utiliser le tag `filebeat`.

```
filter {
    if ("filebeat" in [tags]) {
  grok {
      match => {"message" => "%{TIMESTAMP_ISO8601:date_hour}%{SPACE}%{LOGLEVEL:log_level
    }%{SPACE}%{JAVACLASS:class} -> %{JAVALOGMESSAGE:log_message}"}
  }
}
```

Pour finir, la partie sortie défini uniquement l'adresse et le port de notre instance Elasticsearch.

```
output {
    elasticsearch {
        hosts => ["elasticsearch:9200"]
    }
}
```

Le fichier complet `logstash.conf` est disponible dans l'annexe A.1.

Le `Dockerfile` de Logstash télécharge l'image, puis ajoute les fichiers de configuration au système de fichier du conteneur.

```
FROM docker.elastic.co/logstash/logstash:7.6.2
RUN rm -f /usr/share/logstash/pipeline/logstash.conf
ADD pipeline/ /usr/share/logstash/pipeline/
ADD config/ /usr/share/logstash/config/
```

Pour finir, le fichier `docker-compose.yml` organise toute la topologie. Il lancera d'abord Elasticsearch, puis Kibana, et enfin Logstash. Les ports par défaut des différents logiciels seront exposés, afin d'être accessible depuis l'extérieur. Ceci est nécessaire pour permettre la communication entre les différents conteneurs.

```
version: '3'

services:

  elasticsearch:
    container_name: elsearch
    build: ../images/elasticsearch
    environment:
            discovery.type: "single-node"
    ports:
      - 9200:9200
      - 9300:9300

  kibana:
    container_name: kib
    build: ../images/kibana
    links:
      - "elasticsearch"
    ports:
     - 5601:5601
    depends_on:
      - elasticsearch

  logstash:
    container_name: logst
    build: ../images/logstash
    ports:
      - 5044:5044
    depends_on:
      - elasticsearch
      - kibana
```

Un répertoire GitHub [21] contient toute l'architecture décrite.

Pour lancer le déploiement de la Suite Elastic, il suffit de lancer la commande `docker-compose up --build -d` à l'endroit où se situe le fichier `docker-compose.yml`. Dans ce cas-ci, le dossier `topology`. Pour tester si tout le déploiement s'est bien déroulé, une vérification d'Elasticsearch et de Kibana s'impose.

Premièrement, avec un navigateur, la visite de l'adresse `http://localhost:9200/_cat/indices` permet de constater le fonctionnement d'Elasticsearch et de Logstash. En effet, si tout se passe bien, on devrait voir les différents indices contenu dans la base de données, dont un s'appelant `logstash...`. La figure 9 montre cette configuration réussie.

```
green  open .kibana_task_manager_1     jxlhPGoPQCmLbgD3fOHxvA 1 0 0 0  230b  230b
green  open .apm-agent-configuration   koTkBCISQMmF3dqPn1iMew 1 0 0 0  230b  230b
green  open ilm-history-1-000001       RCUFq0GrTW2QdtbQE5k8Rw 1 0 3 0 7.9kb 7.9kb
yellow open logstash-2020.06.17-000001 jNW-irlITVyl9-jIt4z__g 1 1 0 0  230b  230b
green  open .kibana_1                  J-NO7wASRV-JStVtpsvWvw 1 0 1 0  230b  230b
```

FIGURE 9 – Indices contenus dans la base de données Elasticsearch, avec Logstash

Deuxièmement, toujours dans un navigateur, à l'adresse `localhost:5601`, il devrait y avoir l'application Kibana qui est lancée. La figure 10 montre ceci.

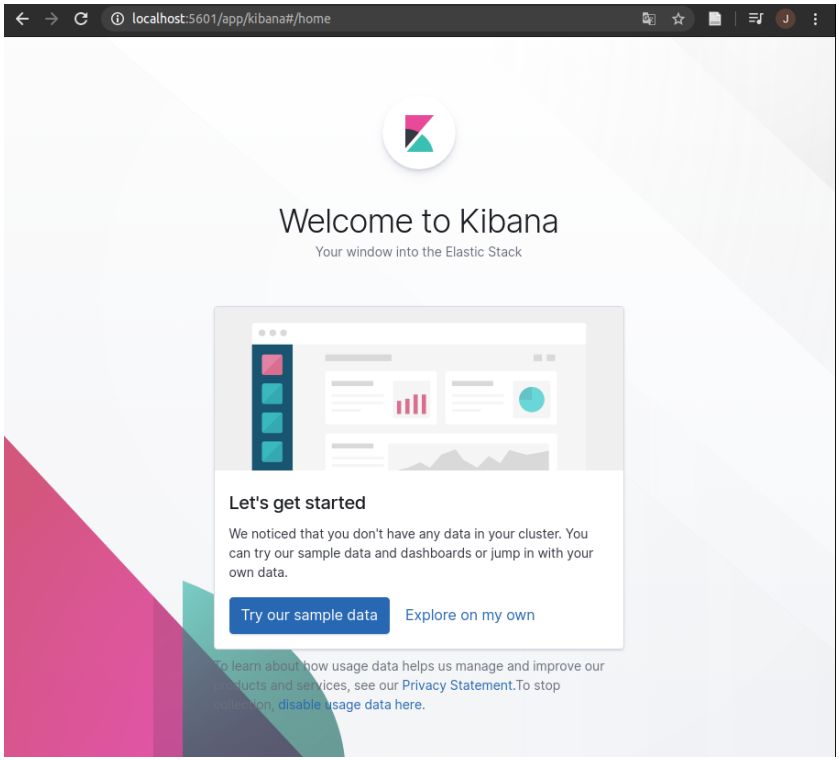


FIGURE 10 – Application Kibana lancée

3.1.2. Machine « terrain »

La machine sur le terrain contiendra les agents Beat nécessaires, ainsi que les programmes et applications générant les logs. Il y aura également une base de données contenant des données arbitraires. La figure 11 montre l'architecture définie pour l'implémentation du use-case sur la machine « terrain ».

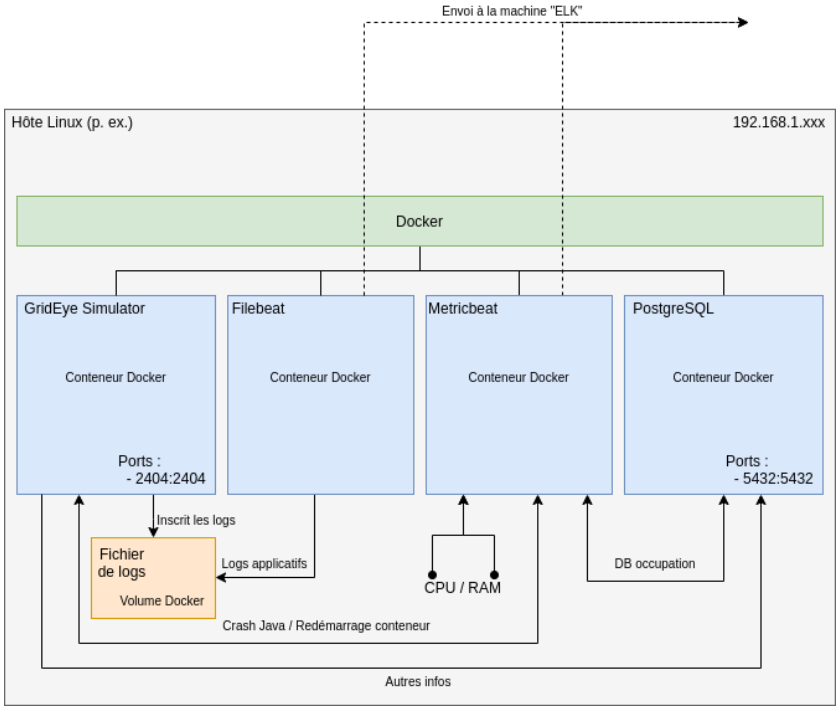


FIGURE 11 – Architecture de l'implémentation du use-case - côté terrain

Il y aura donc le simulateur GridEye générant les logs, un agent Filebeat qui va lire les logs applicatifs et les envoyer à la Suite Elastic (les logs seront partagés entre les deux conteneurs via un Volume Docker, qui sera lié à un chemin sur la machine hôte), un agent Metricbeat qui va surveiller le système hôte ainsi que la base de données. L'agent Metricbeat contiendra également le module de monitoring de Docker, qui nécessite un Volume Docker permettant l'accès au « Daemon Docker ». Il enverra finalement les logs vers la machine hôte ELK. Pour finir, il y aura la base de donnée, qui sera un système de gestion de bases de données relationnelle (SGBDR) PostgreSQL. Comme pour la machine contenant les outils « ELK », nous allons travailler avec un `docker-compose`, qui permettra d'orchestrer tous les conteneurs. La figure 12 montre l'arborescence des dossiers et fichiers contenant les images Docker ainsi que le fichier `docker-compose.yml`. Cette arborescence reprend les mêmes bases que celle du côté ELK.

FIGURE 12 – Arborescence de la topologie Docker - côté terrain

Les images `filebeat` et `metricbeat` ont la même structure. Un `Dockerfile` qui télécharge l'image correspondante depuis le `Elastic Docker registry`, puis qui copie le fichier de configuration de l'agent, respectivement `filebeat.yml`, et `metricbeat.yml`.

`Dockerfile` de filebeat

```
FROM docker.elastic.co/beats/filebeat:7.6.2
COPY config/filebeat.yml /usr/share/filebeat/filebeat.yml
USER root
RUN chown root:filebeat /usr/share/filebeat/filebeat.yml
USER filebeat
```

`Dockerfile` de metricbeat

```
FROM docker.elastic.co/beats/metricbeat:7.6.2
COPY config/metricbeat.yml /usr/share/metricbeat/metricbeat.yml
USER root
RUN chown root:metricbeat /usr/share/metricbeat/metricbeat.yml
```

La configuration de Filebeat se fait dans un fichier `YAML`, qui se nomme `filebeat.yml`, et qui sera copié dans le répertoire `/usr/share/filebeat/` du conteneur. Il y a un grand nombre de paramètres de configuration disponibles. Dans le cas de notre utilisation, il n'y a que deux paramètres à configurer : l'entrée à lire ainsi que la sortie. Pour l'entrée, il suffit de faire pointer le paramètre vers le fichier dans lequel le simulateur GridEye écrit ses logs.

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
            - /usr/share/filebeat/logs/GridEye_sim.log
```

Pour la sortie, on va informer l'adresse de l'instance de Logstash configuré plus tôt.

```
output.logstash:
  hosts: ["<IP>:5044"]
```

Il faut remplacer `<IP>` par l'adresse IP de la machine hébergeant Logstash.

Pour Metricbeat, la configuration est similaire. Elle se fait également dans un fichier `YAML`, qui se nomme `metricbeat.yml`, et qui sera copié dans le répertoire `/usr/share/metricbeat/` du conteneur.

Il y a deux paramètres à configurer : les modules de métriques à surveiller, ainsi que la sortie.

Pour les modules de métriques, nous allons utiliser `system`, qui permettra d'obtenir les informations sur la mémoire et le CPU. Le module `postgresql` nous permettra lui d'obtenir les métriques de la base de données. Et pour finir, le module `docker` enverra lui les métriques concernant les redémarrages de conteneur, et les crashs Java.

La configuration de sortie est la même que pour Filebeat, mais avec le port 5045.

Les fichiers de configuration des deux agents sont disponibles aux annexes A.3 (filebeat) et A.4 (metricbeat).

L'image du simulateur de la plateforme GridEye contient elle un `Dockerfile` qui va lancer une application Java depuis un fichier `jar`. Cette manière de lancer une application Java dockerisée est la plus simple pour ce cas présent (le fichier compilé ne change jamais car il n'y a pas de mise à jour du programme). Il y a également un dossiers `logs` qui contiendra le fichier de log et sera donc la base du Volume Docker partagé avec l'agent filebeat. Le code, ainsi qu'une description de l'application sont disponible dans un répertoire GitHub [23].

`Dockerfile` du simulateur GridEye

```
FROM openjdk:8-jdk-alpine

COPY app/grideye-service-log-generator-1.0.0-SNAPSHOT.jar grideye_sim.jar
CMD java -jar grideye_sim.jar
```

Finalement, le dossier de l'image de la base de données PostgreSQL est également simple. On va simplement télécharger l'image, puis copier le fichier `sql` qui permettra de peupler la base de données avec des informations arbitraires (en l'occurrence, des informations sur des pays et des villes).

`Dockerfile` de la base de données

```
FROM postgres:12.1
COPY data/world.sql /docker-entrypoint-initdb.d/10-init.sql
```

Pour ce qui est du docker-compose, disponible à l'annexe A.5, il contient les quatre services correspondant aux quatre conteneurs Docker. Les services `filebeat` et `grideye_simulator` partagent un Volume Docker, alors que `metricbeat` possède également un Volume Docker qui pointe sur le fichier `docker.sock`, permettant de gérer Docker depuis un conteneur.

Afin de tester l'installation de la partie terrain, il suffit de lancer la commande `docker-compose up --build -d` à l'endroit où se situe le fichier `docker-compose.yml`. Dans ce cas-ci, le dossier `topology`. Si tout s'est bien installé, des logs devraient arriver rapidement à l'adresse indiqué dans les fichiers de configurations, si la Suite Elastic y est bien lancée. On y retrouvera des logs applicatifs via filebeat, ainsi que des logs contenant les métriques via metricbeat.

Pour voir ces logs, on va utiliser Kibana. Il faut tout d'abord créer un `Index Pattern`. Cet outil est disponible dans `Management` → `Kibana` → `Index Patterns` → `Create index pattern`. Le nom du motif doit commencer par les mêmes lettres que l'index stocké dans Elasticsearch. Dans notre cas, on peut donc l'appeler simplement `logstash`. À l'étape 2, il est utile de choisir un `timefilter`, comme `@timestamp`, pour pouvoir effectuer des recherches et des graphes avec la notion de temps.

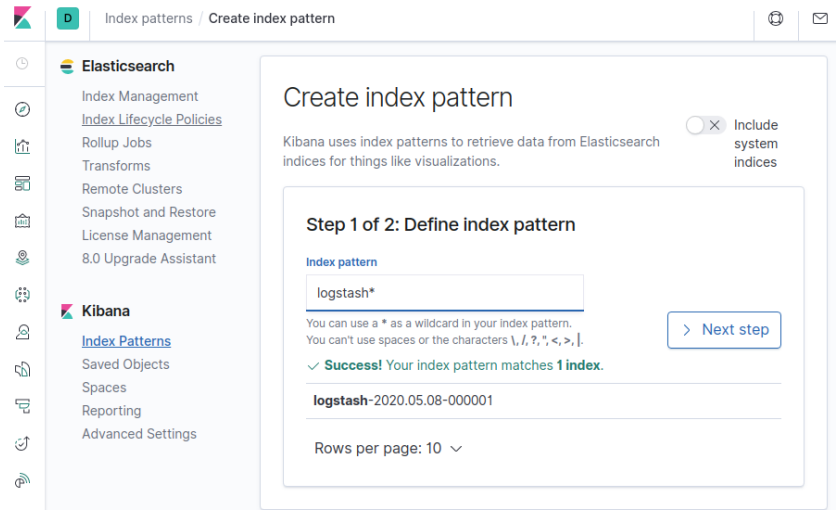


FIGURE 13 – Création de l'index pattern dans Kibana

Une fois créé, on peut voir les logs dans l'onglet `Discover` de Kibana. Il est également possible d'afficher uniquement les champs qui nous intéressent, comme ceux créés dans le filtre.

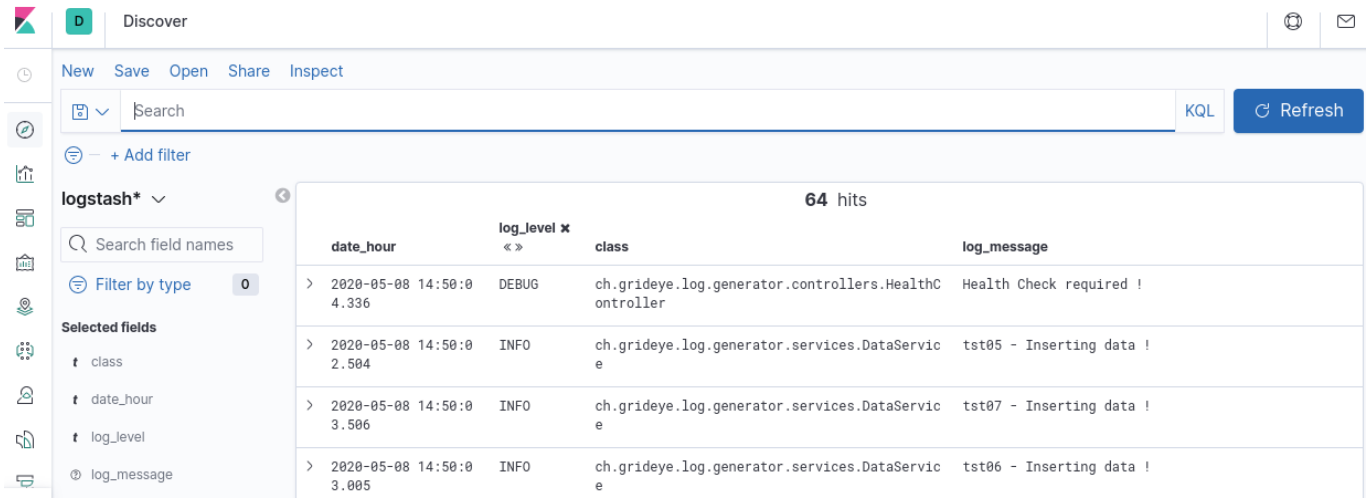


FIGURE 14 – Affichage des logs dans Kibana

3.2. Visualisation avec Kibana

Une fois tous les outils installés, il ne reste plus qu'à créer des visualisations permettant de suivre l'état du système en temps réel, ou encore de retrouver l'origine d'un problème sur le système.

Pour ce faire, nous allons créer plusieurs visualisations, elles-même contenues dans des tableaux de bord.

Un premier tableau de bord montrera tous les problèmes de l'infrastructure. Ainsi, un opérateur pourra voir immédiatement s'il y a eu un problème dernièrement, ou au contraire si le tableau est vide.

Pour ce cas d'utilisation, il y aura ensuite un autre tableau de bord mettant en relation les taux d'utilisations CPU de chaque conteneur Docker. Il permettra, au cas où l'opérateur remarque une surcharge CPU sur le tableau de bord des problèmes, de déterminer quelle application pose problème.

3.2.1. Tableau de bord des problèmes

Pour ce tableau de bord, nous avons inséré quatre informations susceptibles d'être considérées comme des « problèmes » :

- Les logs de crash du simulateur GridEye
- Les redémarrages de conteneur Docker
- Les sur-exploitations du processeur
- Les sur-exploitations de la mémoire

La figure 15 montre ce tableau de bord. Les trois premières visualisations, concernant les crashs du simulateur ainsi que les sur-exploitations sont de simples recherches dans les logs. Pour les crashs, on affiche tous les logs ayant le champ `class` indiquant le service de crash. Pour le CPU, tous les logs de pourcentage d'utilisation CPU avec une valeur plus grande que 3 sont affichés (les pourcentages des 4 coeurs sont additionnés). Et pour l'utilisation de la mémoire, une valeur plus grande que 0.8 a été choisie. Pour la dernière visualisation, on a un tableau affichant le nombre de logs contenant l'information de Docker `container status` à `Up 2 minutes`.

Sur la capture d'écran de la figure 15, on peut voir un crash du simulateur, plusieurs logs de CPU sur-exploité, aucun pour la mémoire, et quelques redémarrages de conteneur. Pour les redémarrage conteneur, les premiers histogrammes correspondent au premier lancement de l'application, donc les redémarrages qui doivent être interprétés comme des erreurs sont seulement les suivants.

À la vue de ce tableau de bord, on peut donc rapidement conclure que dans l'heure qui précède, on a eu un crash, des problèmes de CPU et un redémarrage de conteneur.

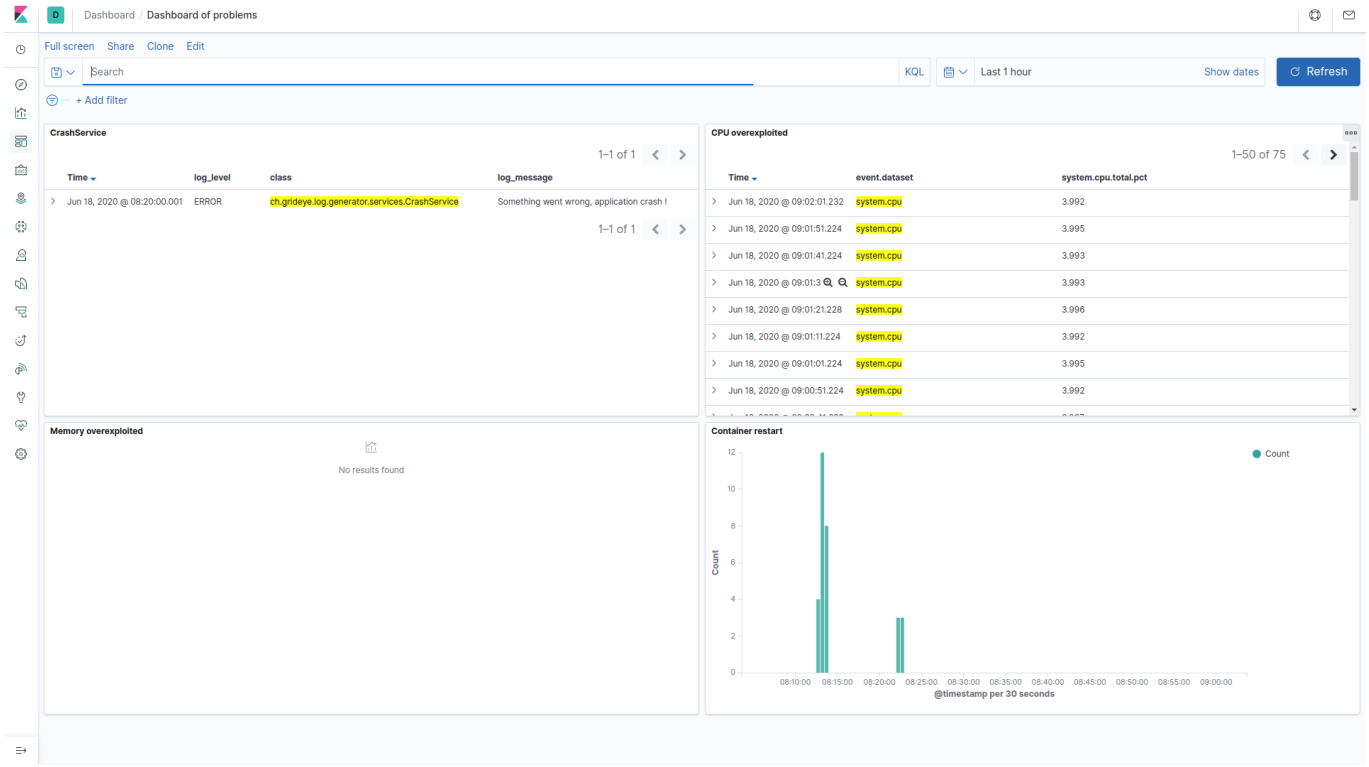


FIGURE 15 – Tableau de bord des problèmes - Kibana

3.2.2. Tableau de bord du CPU

Après avoir constaté ces différents problèmes, on pourrait imaginer que l'opérateur veuille s'attaquer à la surcharge du processeur, et voudrait en savoir un peu plus sur l'origine de cette anomalie. Un tableau de bord contenant les informations détaillées des consommations de CPU a donc été créé.

La figure 16 montre ce tableau de bord.

Les valeurs d'utilisation du CPU sont les mêmes que dans le tableau des problèmes : les pourcentages des 4 coeurs sont additionnés. Elles varient donc de 0 à 4. Dans cette capture d'écran, on remarque que le problème vient du simulateur GridEye, car il utilise périodiquement les pleines capacités du CPU, alors que les trois autres conteneurs ne dépassent pas les 3% sur la globalité des 4 coeurs. L'opérateur pourra encore pousser ces recherches en regardant les logs de GridEye aux heures où il consomme tout le processeur.

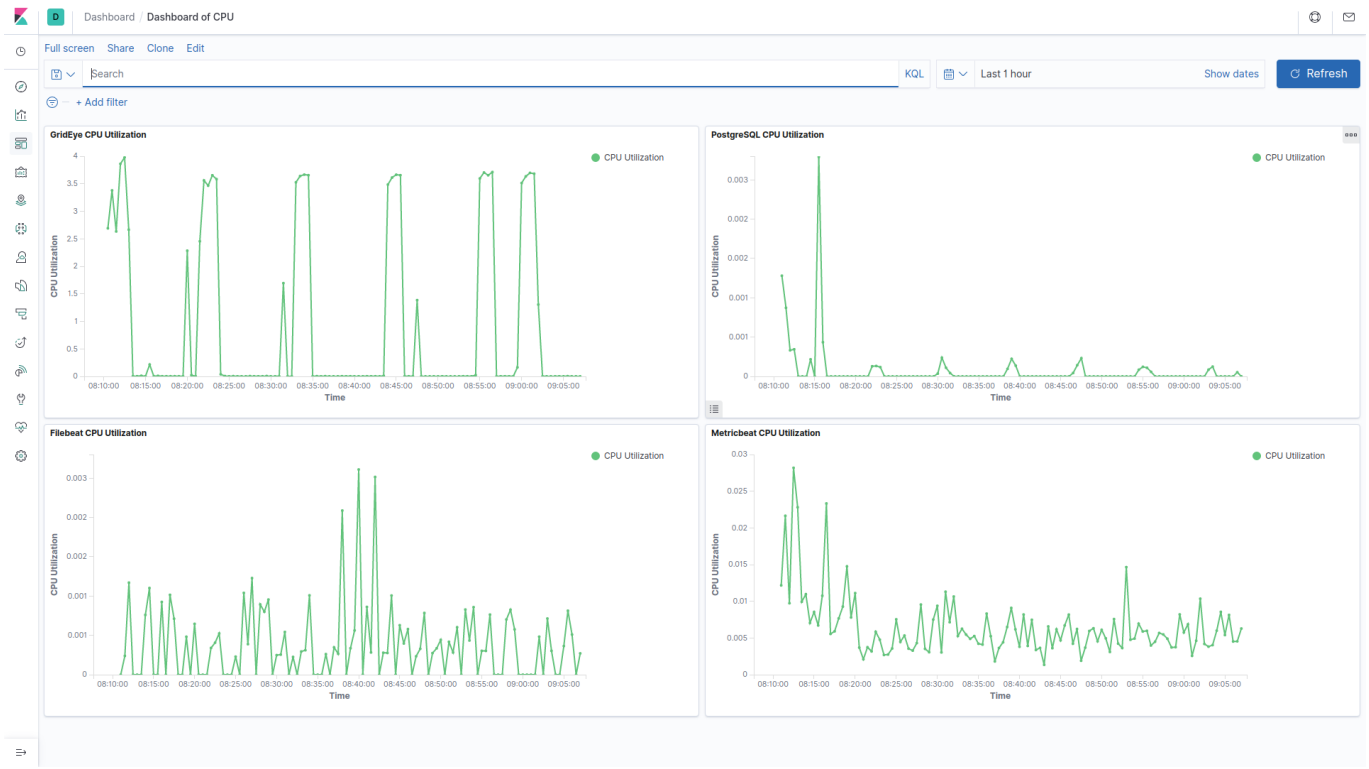


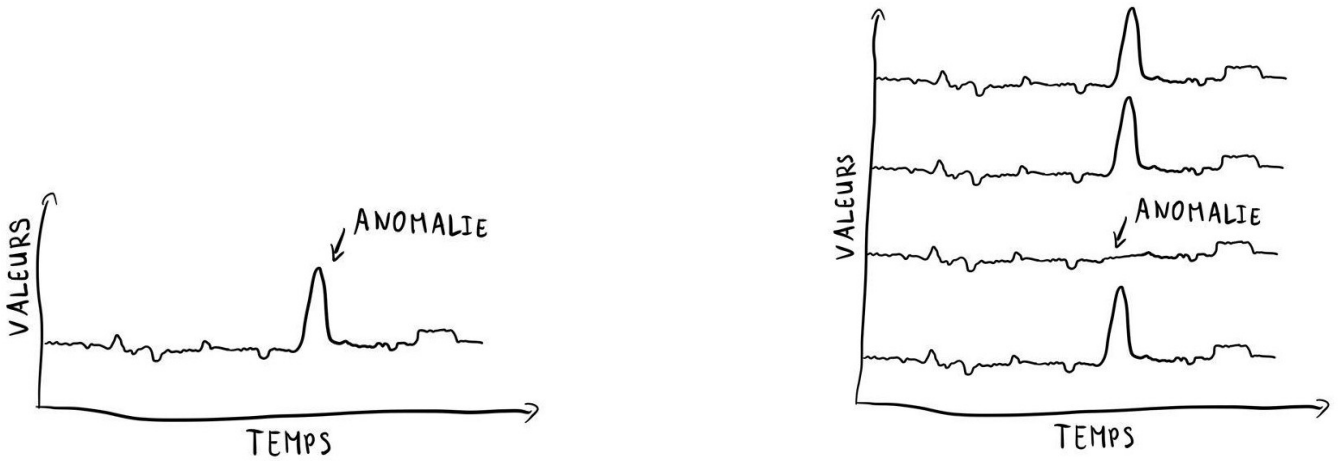
FIGURE 16 – Tableau de bord du CPU - Kibana

3.3. Amélioration de l'architecture Docker

Des améliorations mineures ont été réalisées sur les fichiers `Dockerfile` et `docker-compose.yml`, notamment sur la manière de gérer les versions de la Suite Elastic, qui est passé de 7.6.2 au début du Travail de Bachelor à 7.8.0 à l'heure où ces lignes sont écrites. Ces fichiers se trouvent à l'annexe A.8. Les différents répertoire GitHub mentionnés dans ce rapport sont également à jour [21, 22, 23].

4. Détection d'anomalies et corrélation d'événements

Maintenant que la Suite Elastic est fonctionnelle, nous avons des données de logs et de métriques à disposition. L'étape suivante consiste à rendre ces données parlantes, à en tirer des informations utiles et éloquentes. Pour ce faire, plutôt que de simplement créer des visualisations et des tableaux de bord, nous allons utiliser des algorithmes évolués.
Ceux-ci permettront de faire de l'analyse de métriques, ce qui permettra d'agir sur des problèmes de manière anticipée. Cette analyse de métrique s'appelle la **détection d'anomalies** (*anomaly detection* ou *outlier detection* en anglais). Formellement, la détection d'anomalies dans les séries temporelles a pour but de différencier les moments où une valeur est dite normale, ou anormale, puis de le signaler. Il y a plusieurs manières de définir la normalité. On peut, par exemple, être normal par rapport à soi-même, en prenant compte de ses états historiques. On pourra donc se dire anormal si notre valeur est trop différente par rapport aux valeurs habituelles. On peut également être normal par rapport aux autres. Si beaucoup de même composants ont un comportement similaire, et que notre comportement est bien différent, alors on sera anormal. La figure 17 illustre cette notion.
On voudra également faire de la prédiction de série temporelle, à savoir estimer la valeur qu'aura notre série dans X temps, en fonction de la tendance actuelle et passée. Concrètement, dans ce projet, un exemple basique consiste en un disque mémoire qui se rempli petit à petit, nous aimerions être informé de son plein remplissage avant qu'il n'arrive, sachant qu'il y a un certain délai de remplacement. Nous aimerions également avoir des seuils d'alertes « actifs », qui s'adaptent à une courbe de série temporelle, plutôt que de devoir définir des seuils d'alerte constants, qui demandent une réflexion pour la valeur à fixer.
Nous allons également étudier des algorithmes de corrélation entre différents événements, dans l'optique de découvrir les « symptômes »des pannes les plus fréquentes. L'exemple ici serait de définir une surcharge du CPU comme étant une panne. Le but de l'algorithme serait alors de découvrir quels sont les événements qui arrivent les plus souvent avant cette panne. Il serait donc possible de donner une alerte avant la panne, qui donnerait un message de la forme « symptôme X détecté, n% de probabilité qu'une surcharge CPU suive ».
Pour ce faire, une étude des différents algorithmes actuellement utilisés dans des cas similaires sera faite, afin de choisir les algorithmes les plus adaptés, puis une implémentation de ces-dits algorithmes sera réalisée.



(a) Anomalie par rapport à soi-même

(b) Anomalie par rapport aux autres

FIGURE 17 – Illustration de la détection d'anomalies - inspiré par [19]

4.1. État de l'art

Il existe de nombreux algorithmes dans le domaine de la détection d'anomalies, qui ont chacun leurs fonctionnalités, et leurs avantages et inconvénients. Chacun de ces algorithmes possède son propre but, et il n'y a pas de méthode permettant de faire de la prédiction, de l'adaptation de seuil actif et d'autres choses en même temps. Pour avoir un système complet, il faut donc utiliser et implémenter plusieurs algorithmes. Glo-

balement, la détection d'anomalie n'est pas quelque chose de simple, et sa résolution est donc par définition complexe également.
Pour commencer, il existe de nombreuse manière de détecter des anomalies dans une série temporelle. Il y aura des algorithmes basiques, ou des algorithmes d'apprentissage automatique. Dans cette dernière catégorie, il y aura des algorithmes à apprentissage supervisé ou non-supervisé, etc. Il faudra également choisir entre des algorithmes permettant d'analyser des séries temporelles historiques, ou d'autres qui permettent d'analyser des données en temps réel. Globalement, pour déterminer l'algorithme à utiliser, il faut connaître sa série temporelle et savoir ce que l'on veut obtenir, afin de choisir le bon algorithme.

4.1.1. Le Machine Learning de la Suite Elastic

Dans le cadre de ce projet, comme la Suite Elastic est utilisée, et qu'elle propose elle-même des fonctionnalités de détection d'anomalies dans sa version Premium, le modèle à suivre sera tiré de cette suite. Toutes les fonctionnalités de détection d'anomalies se regroupent, chez Elastic, sous l'appellation « Machine Learning »[30]. Dans la version actuelle (7.8.0), la Suite Elastic propose de l'apprentissage automatique supervisé, et non-supervisé. La différence entre ces deux modes de fonctionnement est que pour utiliser les algorithmes supervisés, il faut fournir au système des données « normales », sans anomalies.
Le Machine Learning d'Elastic est séparé en deux parties : *Anomaly Detection*, qui est l'apprentissage automatique non-supervisé, et *Data Frame Analytics*, qui lui est l'apprentissage automatique supervisé. La partie supervisée est pour l'instant expérimentale et Elastic n'exclut pas de la retirer totalement du produit dans le futur, ou de grandement la modifier, c'est pourquoi elle ne sera pas détaillée ici.

4.1.1.1. Anomaly Detection

Dans le domaine du machine learning non-supervisé d'Elastic, on trouve également plusieurs fonctionnalités. Tout d'abord, la détection de normalité par rapport à soi-même et ses états passés. Cette fonctionnalité permet d'avoir des seuils dynamiques, et de définir quelles anomalies on veut détecter. On entend par là que la Suite Elastic propose des fonctions de détection. Par exemple, pour la surcharge CPU, seules les détections positives nous intéressent. En effet, si l'utilisation du processeur est trop basse (proche de 0%), ce n'est pas un problème. La figure 18 montre une utilisation de cette fonctionnalité, avec une métrique d'utilisation CPU. On peut voir qu'après un certain temps, le modèle est calculé et adapte ses seuils dynamiquement. Il détecte alors des anomalies sur la deuxième surcharge CPU.

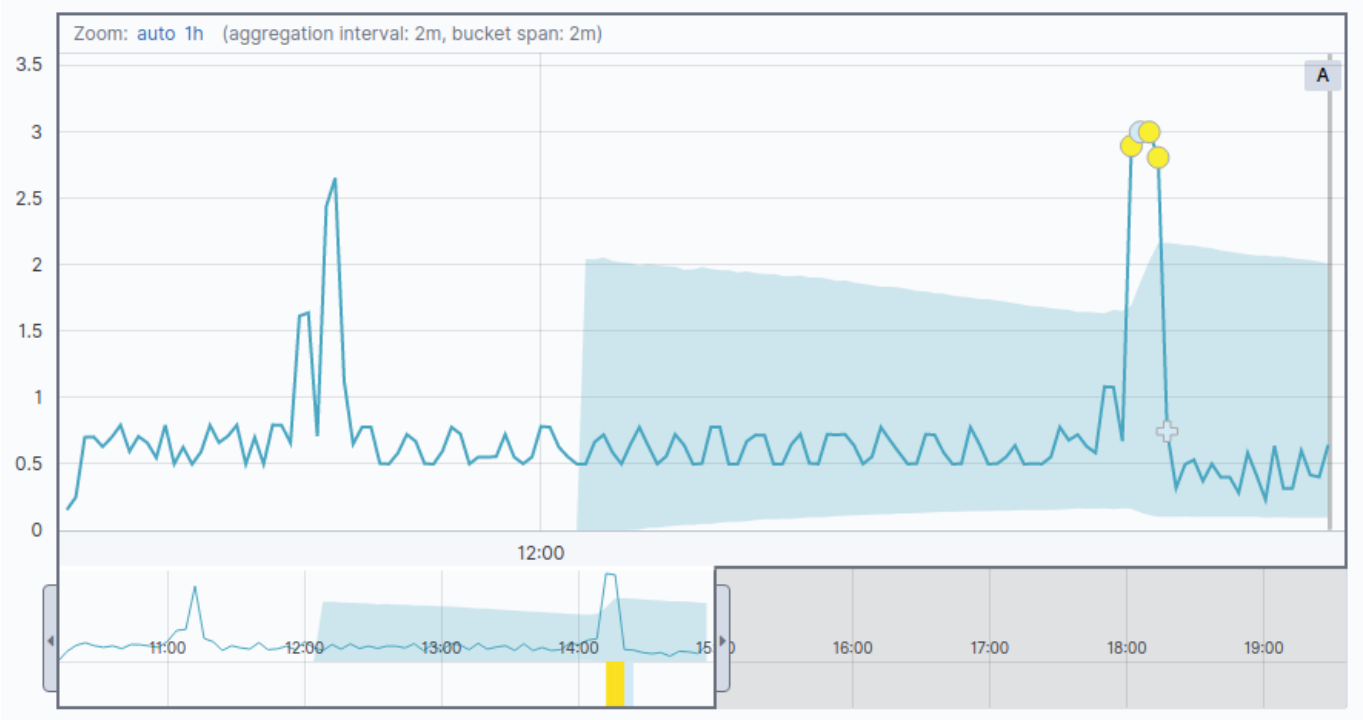


FIGURE 18 – Kibana - Détection d'anomalies d'une métrique CPU

Avec cette même fonctionnalité, il est également possible d'effectuer une prédiction de valeurs, comme le montre la figure 19 (la prévision est la partie en jaune de la série temporelle). Avec cette image, on peut également constater qu'il faut que le modèle ingère quelques valeurs avant qu'il soit réellement opérationnel. En effet, au début de la série, les bornes sont très larges et ne suivent pas réellement la tendance. Ceci est dû, mise à part les propriétés de l'apprentissage automatique, au fait que le modèle est calculé au moment de l'ingestion. Il ne va en effet pas se calculer sur les données historiques, mais plutôt s'améliorer à chaque nouvelle donnée ingérée.

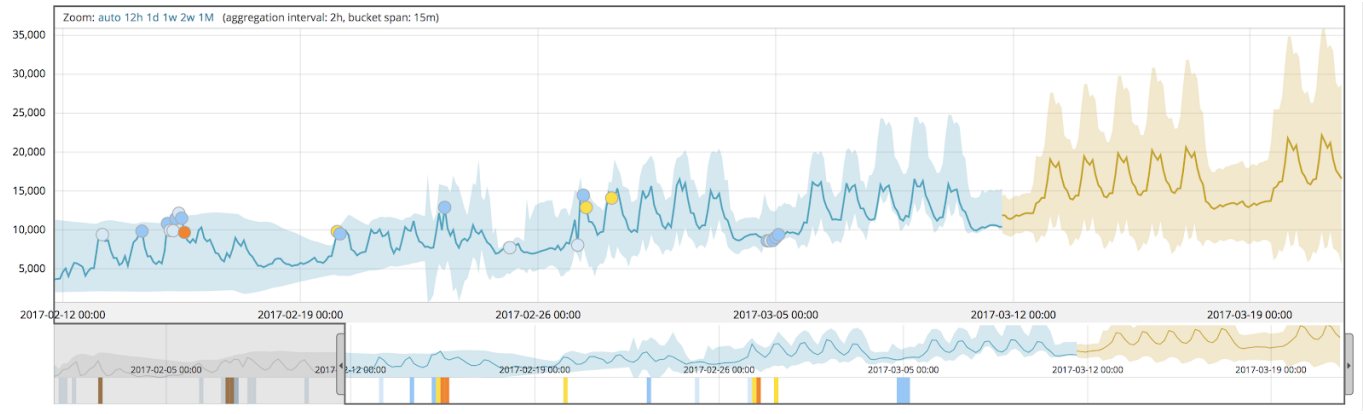


FIGURE 19 – Kibana - Prévision de valeurs [34]

Une autre grande fonctionnalité de la détection d'anomalies est l'analyse de population, autrement dit, l'analyse de normalité par rapport aux autres, comme expliqué à la section 4. Le but est donc d'avoir un grand nombre de machines, ou de composants de même type, afin de les analyser et de voir si un d'eux sort de la normalité. La figure 20 montre cette détection. On a plusieurs serveurs et l'information contenue dans le graphe est le nombre de requêtes reçues par l'adresse IP `29.64.62.83`. On constate alors qu'un des serveurs en reçoit plus que les autres. Une anomalie est alors signalée.

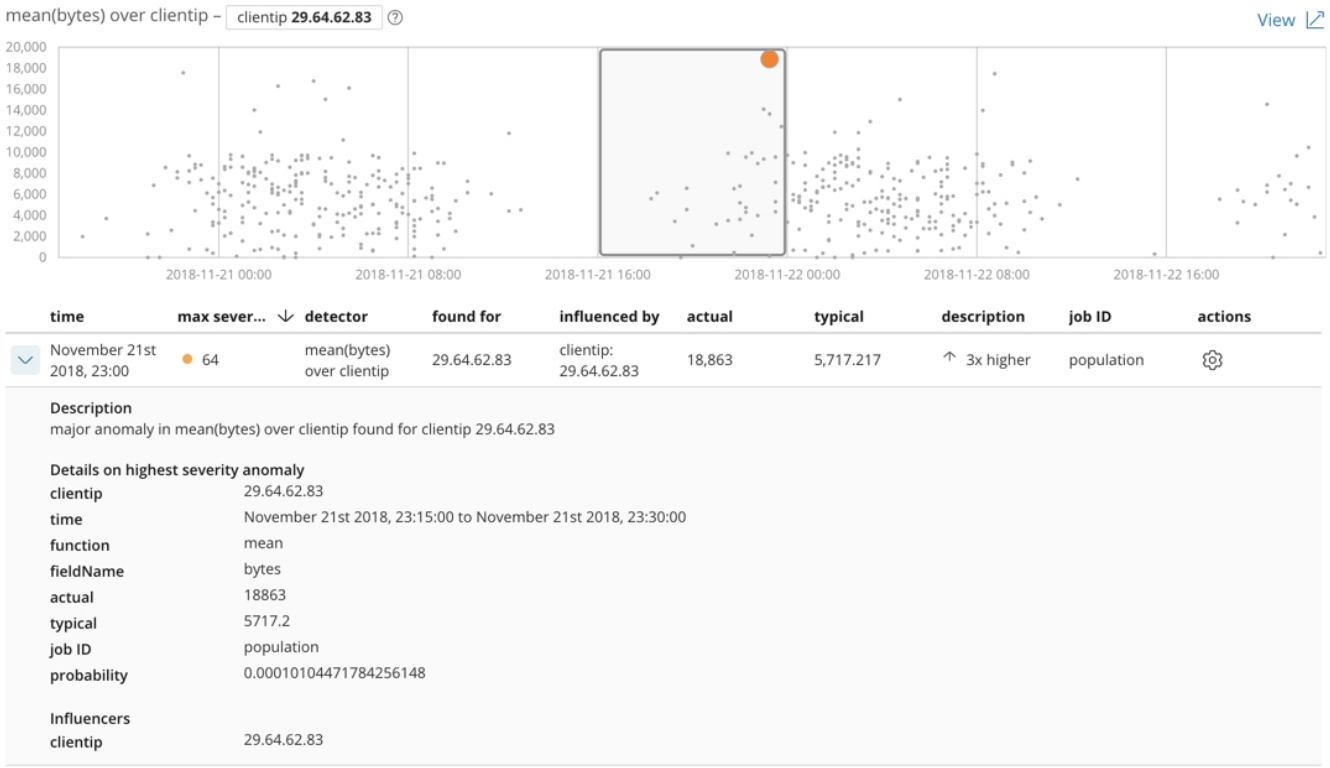
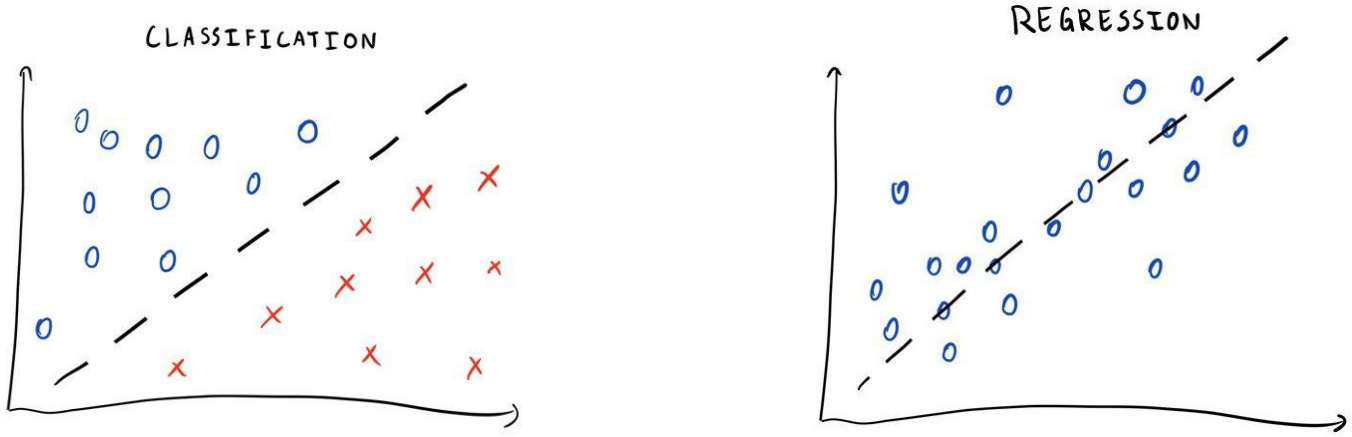


FIGURE 20 – Kibana - Détection d'anomalie dans une population [36]

4.1.1.2. Data Frame Analytics

La partie supervisée du Machine Learning d'Elastic contient, pour le moment, deux fonctionnalités. La première est une classification, et la deuxième est une régression. En théorie, la classification permet de faire de la prédiction de valeur discrète, en se basant sur d'autres données. Un exemple informatique est la classification des e-mails, entre *spam* et *ham*. En se basant sur plusieurs données dans le mail (heure d'envoi, texte, etc.), et connaissant un grand nombre d'e-mail étant des spam ou des ham, l'algorithme arrive à classer de nouveau mail dans une des deux catégories.
Pour la régression, le principe est le même, à savoir s'entraîner sur des données pour prédire des nouvelles, mais cette fois sur des valeurs continues. L'exemple donné par Elastic est le prix d'un appartement. On connaît plusieurs données comme la surface habitable, la distance du centre-ville, etc. d'un grand nombre d'appartement, ainsi que leur prix. Et le programme arrive à déterminer le prix d'un autre appartement, en se basant sur ces mêmes critères (surface, distance, etc.). La figure 21 illustre ces deux fonctionnalités.



(a) Machine Learning - Classification

(b) Machine Learning - Régression

FIGURE 21 – Illustration des techniques d'apprentissage automatique

Le Machine Learning d'Elastic Stack est un de leur principaux axes de travail actuel. Il est amélioré et a été garni de nouvelles fonctionnalités avec beaucoup des derniers passages à de nouvelles versions. Il y aura donc probablement encore d'autres fonctionnalités qui vont venir s'y greffer dans les prochaines versions, ou des modifications aux possibilités actuelles.
Comme il n'est pas possible dans les conditions d'un Travail de Bachelor d'implémenter autant de fonctionnalités que propose Elastic, il a été choisi de se focaliser sur la partie non-supervisée, qui serait la plus adaptée à notre projet. En effet, les seuils dynamiques sont une fonctionnalité qui apporte réellement un plus. Il y a plusieurs métriques à analyser sur la plateforme Smart Grid de DEPsys et il est compliqué de fixer un seuil supérieur et inférieur pour chacune de ces métriques. Cela demande une réflexion, puis un suivi non négligeable. Un algorithme non-supervisé permet alors d'annuler cette réflexion. De plus, pour utiliser des algorithmes supervisés, il est nécessaire de fournir au modèle une base assez conséquente de séries temporelle sans anomalie. Dans notre cas, nous n'avons pas ces données à disposition, et les générer demanderait un travail supplémentaire.
La partie prédiction de l'apprentissage automatique non-supervisé est également intéressant. En étant combiné à un mécanisme d'alerte, elle s'inscrit dans la logique pro-active du système de gestion de logs, en anticipant les problèmes.
Comme la Suite Elastic n'informe pas sur les algorithmes se cachant derrière les décors de leur Machine Learning non-supervisé, le vendant plus comme une fonctionnalité utilisable en quelques clics, une recherche sur les algorithmes actuellement les plus utilisés s'impose.

Voici quelques algorithmes, ou bibliothèques d'algorithmes, n'utilisant pas forcément l'apprentissage automatique, proposant une fonctionnalité de détection d'anomalies.

4.1.2. Décomposition STL

La décomposition STL basé sur Loess [43] n'est pas une méthode de détection d'anomalies à première vue, mais beaucoup d'algorithmes sont basés sur cette décomposition de signal. Concrètement, la plupart des séries temporelles sont saisonnières [46], ce qui signifie qu'elles ont certaines variations qui se répètent tous les x temps, p. ex. chaque semaine. Un exemple typique est la baisse du nombre de requête sur un serveur en pleine nuit, alors qu'en journée le taux augmente. Cette décomposition consiste à séparer la série temporelle en 3 :

- Saisonalité
- Tendance
- Bruit (reste du signal)

Après avoir décomposé une série, on peut détecter plus facilement les anomalies, ceci sur la courbe du bruit. En effet, l'algorithme ne sera plus perturbé par les variations de saisonalité et de tendance, qui ne sont pas des anomalies. Comme le montre la figure 22, sur le signal non décomposé, on ne voit pas forcément d'anomalie. Par contre, après décomposition, sur la série du bruit (*remainder*) on constate clairement une baisse anormale en début de l'année 1952.

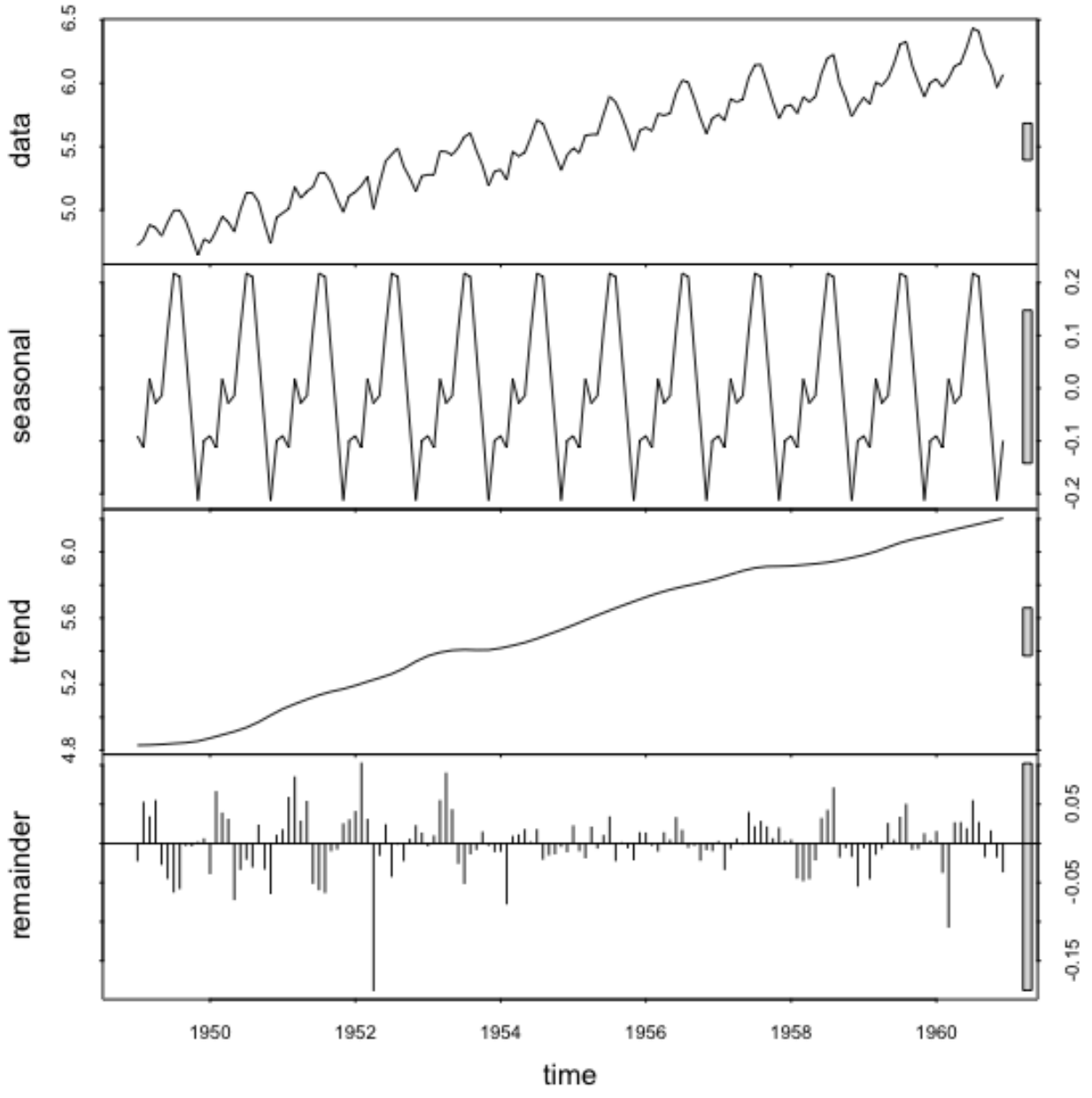


FIGURE 22 – Décomposition STL d'une série temporelle [54]

4.1.3. ARIMA

La méthode ARIMA [5] permet de faire de la prédiction de valeurs de série temporelle. Ce modèle est un algorithme d'apprentissage automatique qui est une contraction de plusieurs « sous-modèles ». On compte d'abord le modèle AR, qui est un modèle auto-régressif. Ce type de modèle fonctionne sur le principe que les valeurs passées ont un impact sur les valeurs actuelles, ce qui permet de prédire une valeur selon les

précédentes.
Ensuite, il y a le modèle MA, qui est un modèle de moyenne mobile [33]. La moyenne mobile consiste à calculer la moyenne de n termes d'une série (qui se nomme régulièrement une *fenêtre*), puis de passer à la valeur suivante. On fait ainsi glisser la fenêtre de moyenne sur les valeurs de la série. La table 11 montre un exemple avec une fenêtre de taille n = 2.

Valeurs	3	6	12	0	4
Moyenne mobile	-	$\frac{3+6}{2} = 4.5$	$\frac{6+12}{2} = 9$	$\frac{12+0}{2} = 6$	$\frac{0+4}{2} = 2$

TABLE 11 – Exemple de moyenne mobile

En combinant ces deux approches, on obtient alors un modèle ARMA. Et ARIMA ajoute un élément I, pour intégration, qui signifie que les valeurs de données ont été remplacées par la différence entre leurs valeurs et les valeurs précédentes. ARIMA est une généralisation du modèle ARMA.
Globalement, la méthode est basée sur une approche selon laquelle plusieurs points du passé génèrent une prévision du point suivant, et avec l'ajout d'une variable aléatoire.

Chaque parties du modèle demande un paramètre, et donc la méthode ARIMA en demande trois, qui ne sont pas forcément facile à déterminer :

- Le nombre de termes autorégressif (AR)

  Correspond à l'ordre du modèle auto-regréssif [38]

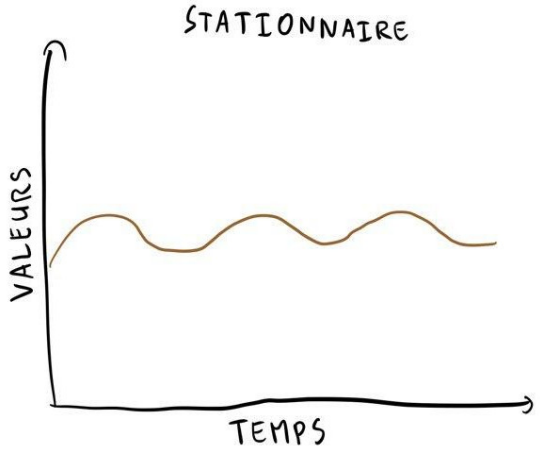
- Le nombre de différence non-saisonnière (I)

  Correspond aux nombres de fois que la différence a été soustraite à la valeur de donnée.

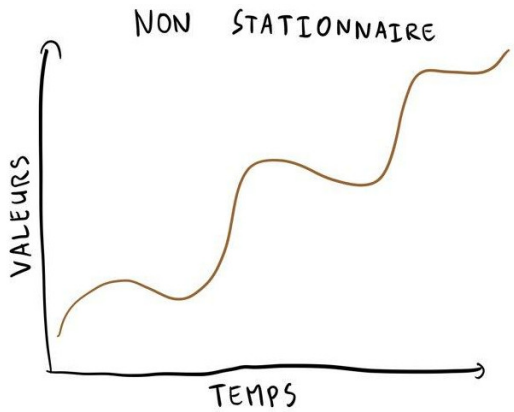
- Le nombre de termes moyens mobiles (MA)

  Correspond à la taille de la fenêtre de la moyenne mobile.

Une autre difficulté est que la série temporelle à analyser doit être stationnaire [49]. Une série stationnaire est une série qui ne dépend pas du temps, la figure 23 illustre ce concept de stationnarité. Il est cependant possible de rendre une série stationnaire en appliquant une opération mathématique sur les valeurs, selon la non-stationnarité. Par exemple, pour une série qui croît proportionnellement au temps, on peut appliquer ARIMA sur le logs des valeurs.



(a) Série temporelle stationnaire



(b) Série temporelle non stationnaire

FIGURE 23 – Illustration de la stationnarité des séries temporelles

4.1.4. Twitter AnomalyDetection

Twitter a développé une bibliothèque de détection d'anomalies [52, 16], utilisant la décomposition STL, ainsi qu'un algorithme Seasonal Hybrid ESD (S-H-ESD), basé sur le test ESD généralisé [1], qui est utilisé pour détecter des valeurs aberrantes dans une série temporelle.
Leur système permet de détecter aussi bien des anomalies globales que locales, tout comme des anomalies positives et négatives. Ces aspects sont montrés avec la figure 24.

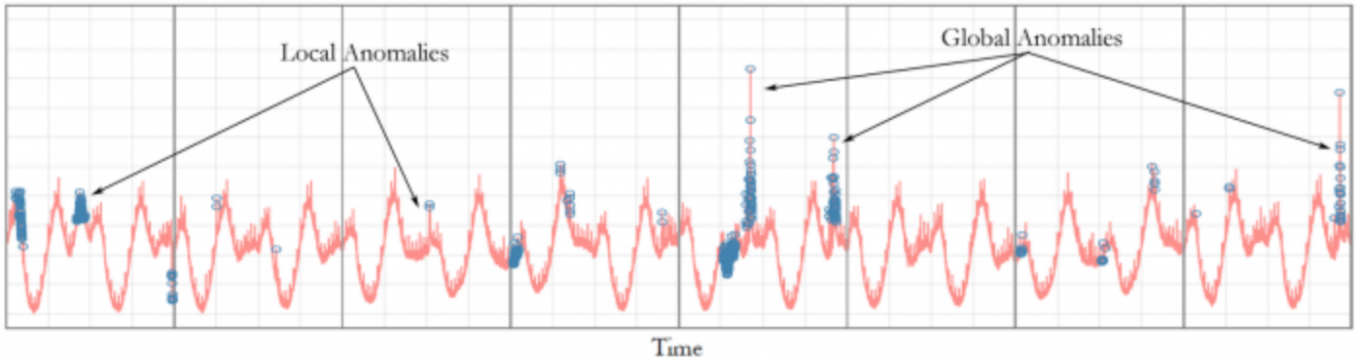


FIGURE 24 – Résultat de l'algorithme de Twitter pour la détection d'anomalie [16]

4.1.5. Facebook Prophet

Comme Twitter, Facebook a également développé une bibliothèque de détection d'anomalies, appelée Prophet [39]. Il est spécialisé dans la prédiction de valeurs dans des séries temporelle et est basé sur un modèle additif. Il est particulièrement adapté à des séries ayant une saisonalité marquée, et gère bien les données manquantes. La figure 25 montre la bibliothèque Prophet en action sur une série temporelle de 8 ans, avec une prédiction de valeurs sur une année.

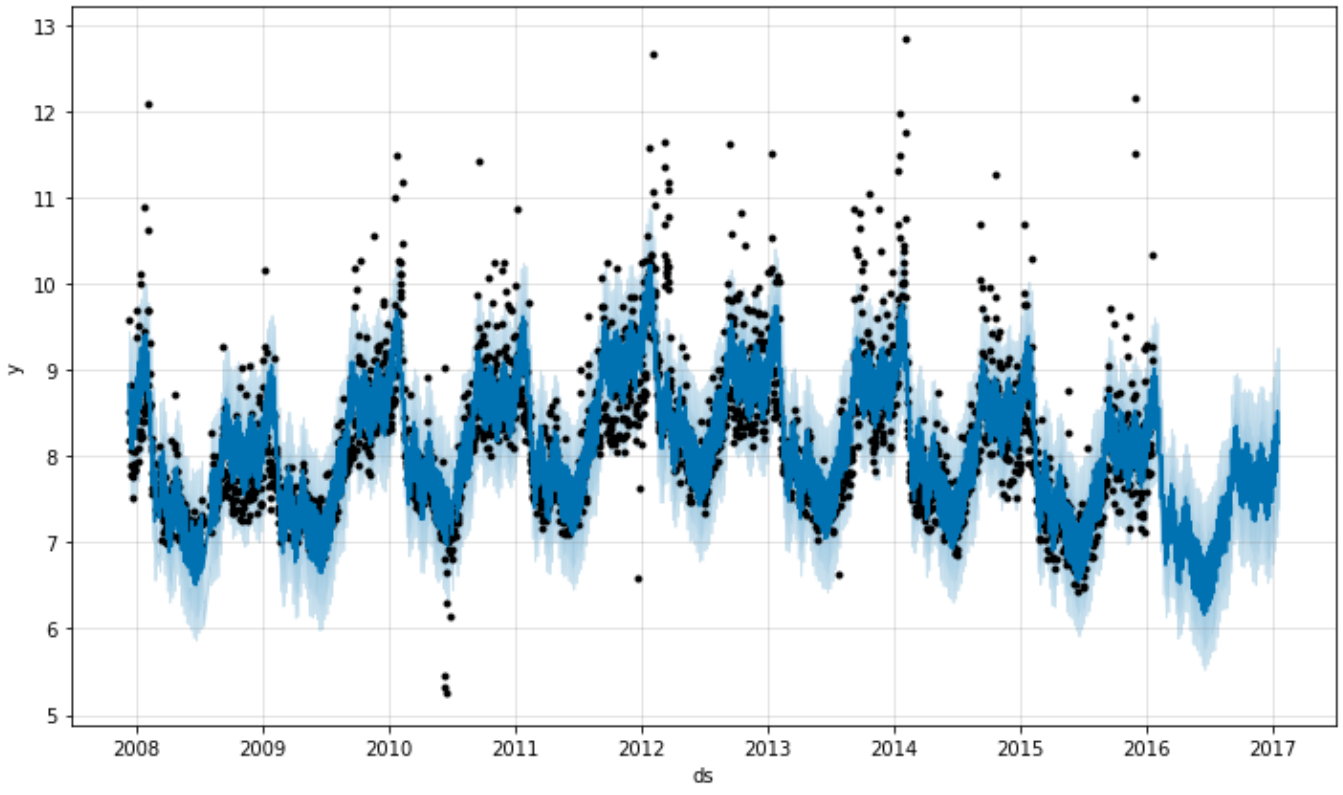


FIGURE 25 – Résultat de l'algorithme de Prophet pour la détection d'anomalie [40]

4.1.6. Autres algorithmes de détection d'anomalies

Le domaine de la détection d'anomalies est réellement vaste et beaucoup d'article ainsi que de papier universitaire parlent de ce sujet. Il existe un répertoire GitHub [55] listant des ressources concernant cette thématique et qui est régulièrement mise à jour. Il existe également beaucoup d'autres d'algorithmes/bibliothèques/méthodes traitant de ce sujet que ceux qui ont été présentés dans ce document. En voici une liste non exhaustive :

- Arbres de classification et de régression

  Apprentissage automatique, peux être utilisé en supervisé ou en non-supervisé. Permet de faire de la prédiction de séries temporelles.

- Algorithme de Welford

  Algorithme statistique qui peut être utilisé pour faire de la détection d'anomalie sur des flux de données (en temps réel) [26].

- Boîte à moustaches

  Les boîtes à moustaches sont des outils statistiques très simples qui indiquent des valeurs médianes ainsi que des valeurs aberrantes. Ils peuvent être utilisés comme outil de détection d'anomalies [26].

- Half-Space Tree (HST)

  Une méthode d'apprentissage automatique supervisé qui peut être utilisé sur des flux de données [26].

- Long Short-Term Memory (LSTM)

  Consiste en un réseau de neurone récurrent [44].

4.1.7. Corrélation d'événements

Le domaine de la recherche de corrélation d'événements dans les outils de gestion de logs est moins développé que celui de la détection d'anomalies. Il n'existe pas, par exemple, de fonctionnalité dans la Suite Elastic permettant de trouver un « symptôme »de panne. Par symptôme de panne, on entend un événement qui permet de deviner une panne à venir. Il y a également peu ou pas de papier universitaire parlant de cette thématique. Nous avons donc tenté d'utiliser un algorithme très utilisé dans le monde du commerce : l'analyse du ticket de caisse (plus connu sous le nom anglais de *Market Basket Analysis*).

4.1.7.1. Market Basket Analysis

Le but de cet algorithme est de lier des événements à des autres afin de déterminer des liens de type $EvenementA \rightarrow EvenementB$, avec la flèche voulant dire « a pour conséquence ». Le côté gauche de la règle est « l'antécédent », alors que le côté droit est la « conséquence ». Il est beaucoup utilisé dans le commerce afin d'organiser les rayons des magasins. Il nécessite d'enregistrer tous les tickets de caisse des clients, que l'on nommera « transactions », puis d'appliquer l'algorithme de Market Basket Analysis qui sortira les événements ayant le plus souvent lieu ensemble. Le but, dans un commerce, est de trouver l'organisation des rayons permettant de vendre le plus de produit possible. Par exemple, si on remarque que la plupart des clients qui achètent des couches pour bébé achètent également des bières, alors on disposera le rayon « bières »à côté du rayon « couches pour bébé ».
Pour utiliser cet algorithme, on a besoin d'un set de donnée contenant des transactions. Une transaction est une liste d'événements ayant eu lieu ensemble. Dans le cas d'un commerce, une transaction est un ticket de caisse, car il renseigne une liste d'article ayant été acheté ensemble. Dans le cas de notre projet, la transaction sera des événements (logs) ayant eu lieu dans une certaine fenêtre de temps. Le résultat de l'algorithme sera des règles d'associations. Par exemple, $\{Bieres, Chips\} \rightarrow \{Couches\}$ signifie qu'un client qui achète des bières et des chips achète souvent des couches. Dans la même idée, la règle $\{AnalyseDisque\} \rightarrow$

$\{SurchargeCPU\}$ indiquera qu'après une analyse de disque, une surcharge CPU à des grandes chances d'arriver.
Les règles d'associations produite par l'algorithme sont accompagnées de trois métriques permettant d'évaluer la force de la règle : [31]

- Support

  Le pourcentage de transaction qui contient tous les éléments de la règle. Par exemple, pour la règle $\{Bieres, Chips\} \rightarrow \{Couches\}$, ce sera le pourcentage des tickets contenant les 3 articles bières, chips et couches. Il est donc préférable de se fier à des règles avec un haut niveau de support.

- Confiance

  La probabilité qu'une règle qui contient les mêmes antécédents (et peut-être d'autres) contient l'article conséquent dans sa partie de droite. Par exemple, dans notre cas, la probabilité qu'une règle qui contient, dans sa partie antécédent, entre autres, les articles bières et chips, a dans sa partie « conséquence »l'article couches. Une grande confiance permet d'assurer la véracité de la règle.

- Lift

  On prend le support, et on le divise par les probabilités des éléments antécédents et conséquents, comme s'ils n'étaient pas liés par la règle. Cela permet de montrer la force de l'association entre les produits de gauche et de droite. Plus le lift est grand, plus le lien entre les deux parties est important.

Voici un exemple d'application simpliste de l'algorithme, adaptée à ce projet :

- Soit une base de 1000 transactions contenant des événements normaux comme « Analyse de disque », « transmission de logs », ou encore « insertions de données dans la BDD », et des événements problématiques comme « surcharge CPU », « surcharge RAM », ou encore « crash Java ».
- Dans ces transactions, 127 d'entre elles contiennent une analyse de disque, 43 une surcharge CPU et 37 les deux.
- On analyse la règle $\{AnalyseDisque\} \rightarrow \{SurchargeCPU\}$.
- Le support sera $P(analyse \cap surcharge) = \frac{37}{1000} = 0.037$
- La confiance sera $\frac{support}{P(analyse)} = \frac{0.037}{0.127} = 0.29$
- Le lift sera $\frac{support}{P(analyse) \cdot P(surcharge)} = \frac{0.037}{0.127 \cdot 0.043} = 6.78$

L'algorithme peut donc être utilisé dans d'autres cas que celui du commerce. Cependant, la notion de transaction est moins évidente qu'un ticket de caisse lorsqu'il s'agit de logs et de problèmes comme des pannes. Il faut tout d'abord définir les problèmes à analyser, puis créer les transactions avec des fenêtres temporelles, en utilisant le même principe que pour la moyenne mobile, tel que montré avec la table 11. Il faut donc déterminer la taille de la fenêtre permettant d'obtenir un problème et sa cause. Ce n'est pas quelque chose de trivial et cela demande une certaine analyse manuelle.
Une autre contrainte de cette méthode est le nombre de donnée à mettre à disposition de l'algorithme. En effet, plus il y aura de donnée, plus il sera performant. En l'occurrence, comme notre cas d'utilisation est sensé découvrir la causalité de problèmes, et que les problèmes ne sont pas sensés arriver de manière nombreuse, il sera forcément plus difficile d'entraîner l'algorithme que dans le cas d'un ticket de caisse, qui est une transaction banale.

4.2. Implémentation

Après avoir analysé les différentes possibilités théoriques permettant d'avoir une analyse semblable à celle de la Suite Elastic, vient l'étape d'implémentation. Trois fonctions présentant donc trois algorithmes ont été réalisées. Elles ne sont pas directement liées avec un mécanisme d'analyse en temps réel sur les données du

simulateur GridEye. Ces fonctions permettent de créer un résultat d'analyse statique sur des données, soit directement lues dans Elasticsearch, soit dans un fichier `CSV` annexe.

4.2.1. ARIMA

L'algorithme ARIMA ici utilisé permet de faire de la prédiction de valeur de série temporelle. Il lit en entrée un fichier `CSV` contenant les données à analyser. Elle applique ensuite l'algorithme et affiche sur un graphe le résultat.

Le fichier `CSV` est de la forme montrée dans la table 12.

Date	Valeurs
"2020-07-01 08 :13 :57"	3.763
"2020-07-01 08 :14 :07"	1.033
"2020-07-01 08 :14 :17"	0.308
"2020-07-01 08 :14 :27"	0.556
"2020-07-01 08 :14 :37"	0.535
...	...

TABLE 12 – Exemple fichier CSV de données pour l'algorithme ARIMA

Voici le code de la fonction : [32]

```
1 def arima_forecast(csv_file, start_plot, end_plot, arima_parameters):
2     """Compute a forecasting operation with an ARIMA algorithm on a timeseries read in a
      CSV file"""
3     df = pd.read_csv(csv_file, parse_dates=['Date'], index_col=['Date'])
4 
5     model = ARIMA(df, order=arima_parameters)
6     results = model.fit(disp=-1)
7     fig = results.plot_predict(start_plot, end_plot)
8     fig.show()
```

Cette fonction prend en paramètre le nom du fichier `CSV` les bornes dans lesquelles les données doivent être analysée, ainsi que les paramètres de la fonction ARIMA. La fonction ARIMA utilisée est celle de la bibliothèque `statsmodels` [17]. Les résultats sont relativement bons sur des données qui vont bien, par exemple l'historique des passagers en avion dans les années 50-60, car la série est croissante et le programme arrive à prédire les valeurs de manière intéressante, comme le montre la figure 26. Comme nous ne disposons pas de données comparables relative au système GridEye, il n'est malheureusement pas possible de le tester. Sur des données plus « plates », comme l'utilisation CPU, le résultat ne donne rien d'utilisable, comme le montre la figure 27.

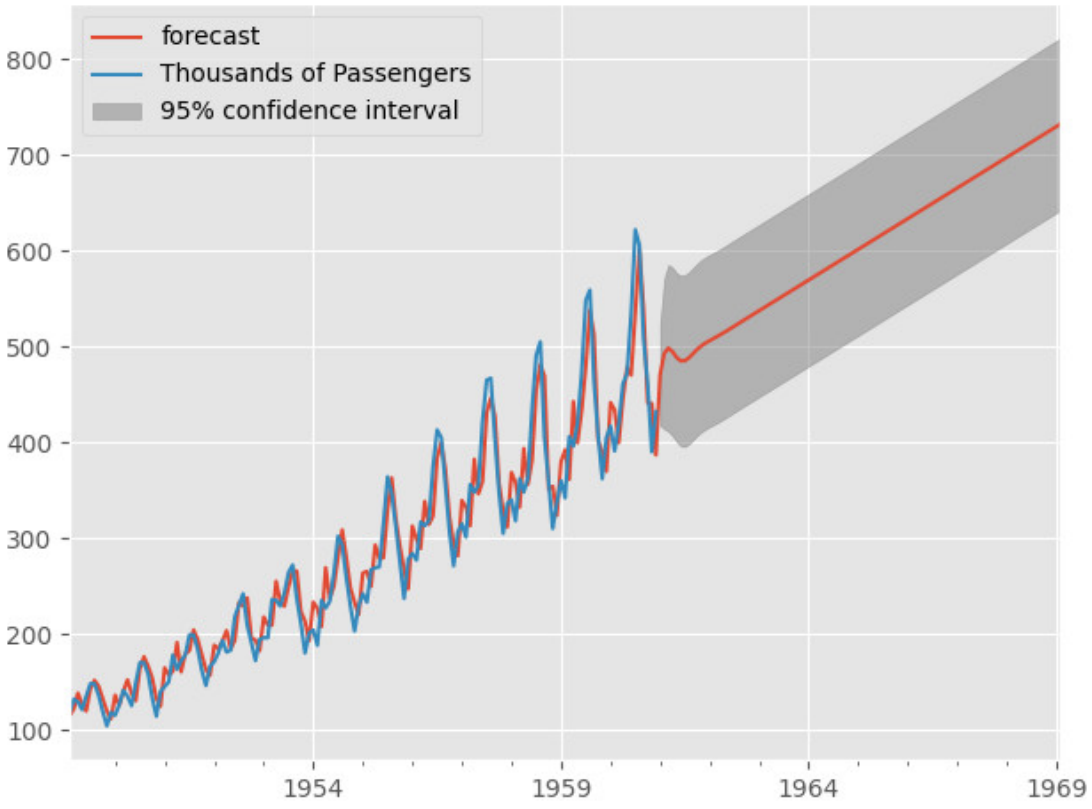


FIGURE 26 – Résultat de l'algorithme ARIMA sur des données de passagers de vols

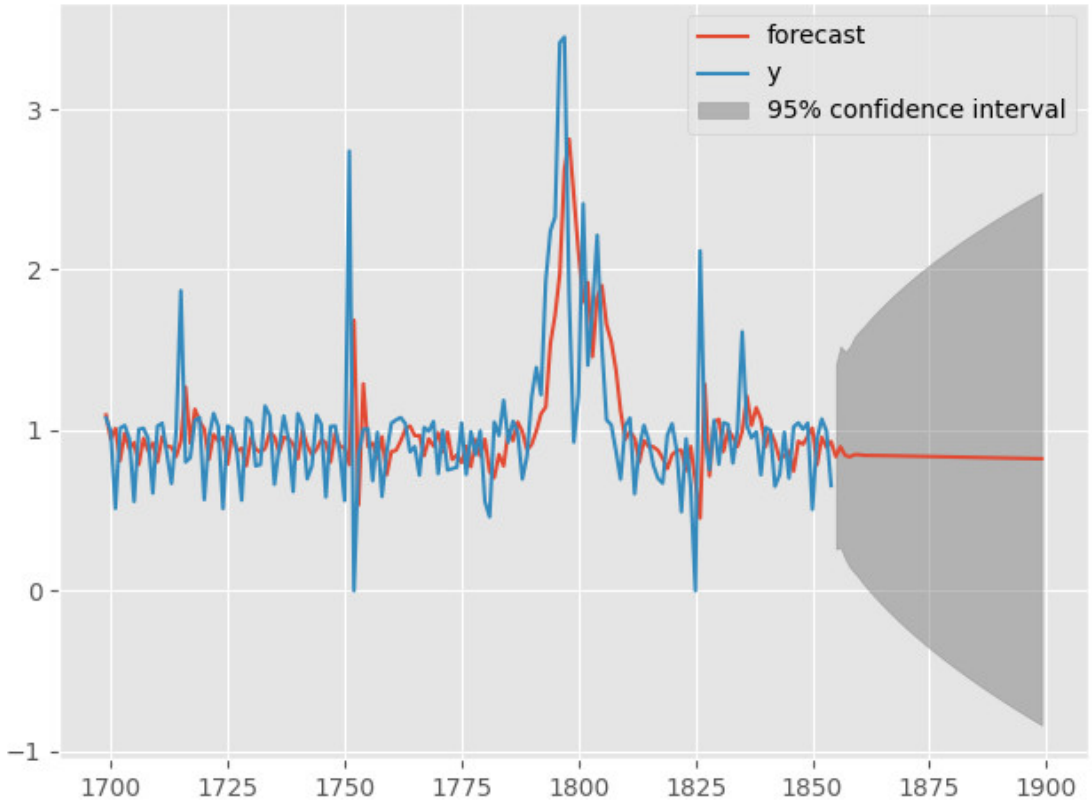


FIGURE 27 – Résultat de l'algorithme ARIMA sur des données de CPU

4.2.2. Facebook Prophet

Dans notre cas, l'utilisation de la librairie Prophet de Facebook permet de faire de la détection d'anomalie grâce à des seuils actifs. Les fonctions implémentées prennent un fichier `CSV` en entrée, puis montrent le résultat dans un graphe. Le fichier `CSV` en entrée doit obligatoirement contenir une colonne `ds` contenant les dates, et une autre colonne `y` contenant les données, comme le montre la table 13.

"ds"	"y"
"2020-07-01 08 :13 :57"	3.763
"2020-07-01 08 :14 :07"	1.033
"2020-07-01 08 :14 :17"	0.308
"2020-07-01 08 :14 :27"	0.556
"2020-07-01 08 :14 :37"	0.535
...	...

TABLE 13 – Exemple fichier CSV de données pour l'algorithme de Prophet

L'annexe A.6 montre le code du programme utilisant Facebook Prophet [4].

Sur un set de donnée CPU généré sur un machine faisant tourner le simulateur GridEye pendant plusieurs heures, Prophet arrive à obtenir de bons résultats, que l'on peut voir à la figure 28. On peut également voir que Prophet gère bien les trous de données. En effet, entre 13h05 et 13h50, il n'y a aucune donnée présente, et tout reprend normalement par la suite.

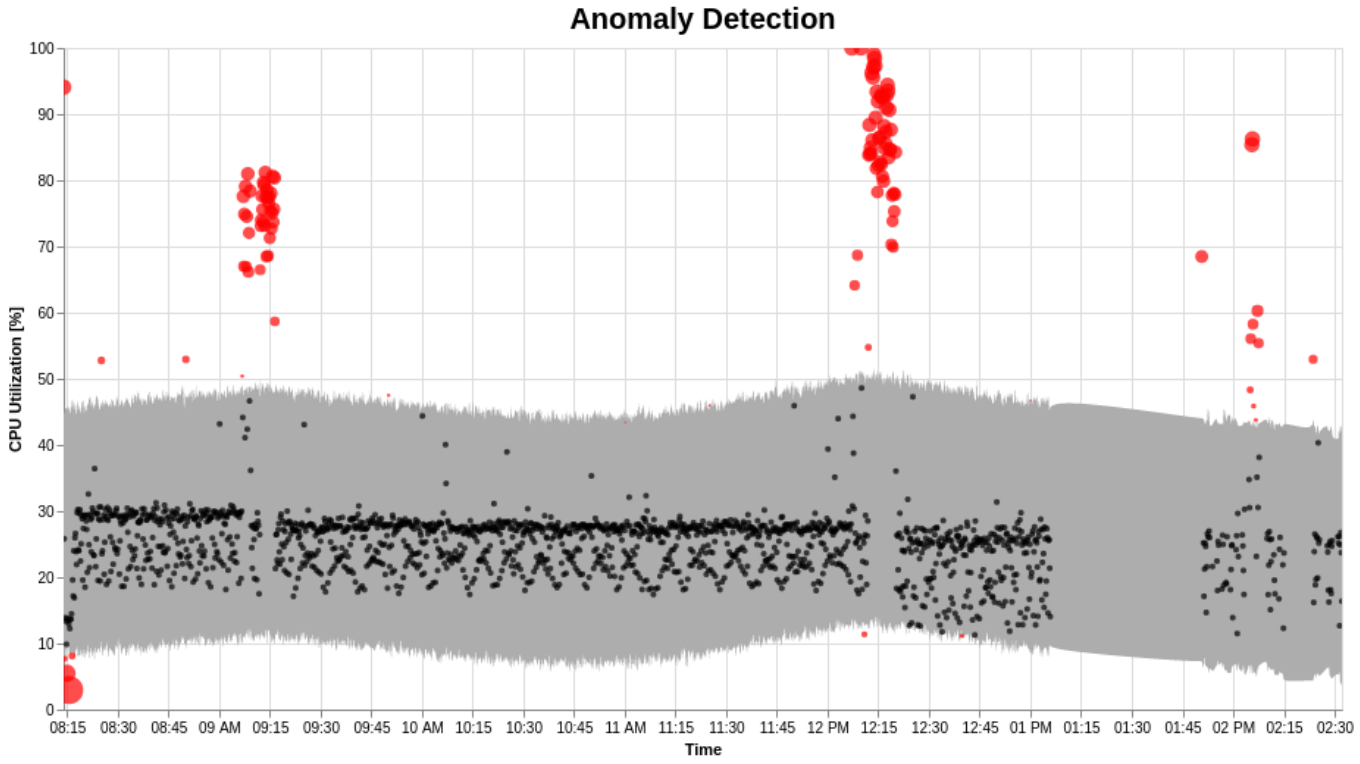


FIGURE 28 – Résultat de l'algorithme de Prophet sur des données de CPU

4.2.3. Market Basket Analysis

L'implémentation du Market Basket Analysis n'a pas donné de résultats probants. En effet, comme décrit à la section 4.1.7.1, il est très difficile d'utiliser un algorithme de Market Basket Analysis sans un grand set de donnée. Ce code fonctionne donc en théorie, mais n'a pas été testé avec des données liées au projet.

Voici le code python de la fonction : [18]

```python
def market_basket_analysis(logs, transaction):
    """Compute a market basket analysis on the logs stored in the class."""
    list_articles = list(logs)

    list_occ = []

    for k, v in transactions.items():
        list_occ.append([])

        for article in list_articles:
            list_occ[k].append(v[article] if (article in v) else 0)

    df = pd.DataFrame(list_occ, columns=list_articles)

    frequent_itemsets = apriori(df, min_support=0.05, use_colnames=True)
```

Cette fonction prend des logs, qui correspondent aux différents « articles », en reprenant l'analogie avec le monde du commerce. C'est donc un `set` qui contient tous les logs existants. La fonction prend également une liste de transactions, qui est une `map` ayant pour clé un id de transaction, et pour valeur une nouvelle `map`, qui elle a pour clé un log, puis pour valeur un 0 si le log est absent dans la transaction, et un 1 s'il est présent.
L'algorithme `apriori` utilisé, qui est la base du Market Basket Analysis, est celui de la librairie mlxtend [41]. Il est ensuite possible de créer des règles pour trouver des informations de la manière suivante :

```python
# Create the rule
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

Le code permettant d'aller chercher ces informations de logs et de transactions est disponible à l'annexe A.7.

5. Conclusion

Après cette étape d'implémentation vient la conclusion et donc la fin de ce Travail de Bachelor. Cette conclusion sera en trois parties. Tout d'abord, une conclusion purement technique sur les choses réalisées par rapport aux demandes du cahier des charges. Suivra une conclusion personnelle et mon avis sur le travail réalisé. Puis, pour finir, une liste des améliorations possibles.

5.1. Conclusion technique

Dans les grandes lignes, le cahier des charges demandait une évaluation des différents système de gestion de logs, afin de mettre en place une solution pour l'infrastructure GridEye de DEPsys. Si on s'en tient aux demandes du cahier des charges, on peut conclure que pour la première partie, à savoir la comparaison, les différentes étapes réalisées ont été pertinentes. Elles nous ont permis de choisir un système, la Suite Elastic, qui offre toutes les fonctionnalités dont nous sommes en mesure d'attendre d'un système de gestion de logs, tout en proposant un environnement flexible et agréable à utiliser. Sa documentation, que ce soit dans les instructions d'installations, ou dans les notices d'utilisations de fonctionnalités comme le Machine Learning, est riche et à jour. Ils proposent, de plus, un bon nombre de vidéos « webinars », en anglais et en français, expliquant par l'exemple l'utilisation de leurs services.
Toujours en suivant le cahier des charges, la suite mentionnée est la mise en place de la solution pour l'infrastructure GridEye. À nouveau, on peut conclure que les différentes installations réalisées sont intéressantes, de part leur nature « dockerisée », elles sont flexibles est faciles de mise en fonction.
La dernière étape du cahier des charges mentionne l'implémentation de librairies pour l'interfaçage avec le Back-End de la solution GridEye. À ce niveau-ci, le travail a plutôt été réalisé avec un simulateur de la plateforme GridEye, étant donné la difficulté technique pour interfacer un PoC avec la solution GridEye. En effet, GridEye n'est pas encore prête à générer des logs et d'autres événements pertinents à un niveau permettant d'être utilisé et testé avec une démonstration de faisabilité. De plus, la situation sanitaire liée au coronavirus n'aide pas à une telle manoeuvre, étant donné que les différentes parties ne se sont pas rencontrées chez DEPsys.
Au delà des spécifications du cahier des charges, si on s'en tient aux différentes demandes évoquées lors du Travail de Bachelor, on peut conclure que la première partie a été remplie (installation avec Docker, respect du cas d'utilisation, etc.). Pour la partie implémentation des algorithmes avancés, sensés permettre de détecter des anomalies ou de trouver des corrélations entre les événements, une disparité entre les attentes et les résultats peut se faire ressentir. En effet, nous sommes loin des fonctionnalités de la partie Machine Learning de la Suite Elastic.

5.2. Conclusion personnelle

Pour ma part, dans la globalité du Travail de Bachelor, je suis plutôt content du résultat. Premièrement, dans la grande majorité des tâches effectuées, j'ai pris du plaisir à découvrir de nouvelles technologies que je ne connaissais pas. Dans la première partie, lors de l'évaluation des différents systèmes de gestion de log, j'ai découvert un nouveau monde, le monde du log, qui à première vue n'est pas le plus passionnant des domaines de l'informatique, mais qui cache une grande complexité et un potentiel pas assez exploité. Ensuite, lors de la réalisation de la démonstration de faisabilité, j'ai pu créer deux architectures mêlant Docker et Docker-compose. Ce sont des technologies que je connaissais mais que je n'avais jamais utilisée « de zéro », en créant son propre `Dockerfile` et `docker-compose.yml` partant d'un fichier vide. Ces expériences m'ont servi et m'ont plu. La partie suivante, qui consistait en des recherches et implémentations d'algorithmes de détection d'anomalies, m'a ouvert quelque peu le monde de la « Data Science », dans lequel j'ai pu remarquer que mes connaissances mathématiques étaient enfouies très loin dans ma mémoire.
Si j'ai grandement apprécié découvrir ces différents domaines, le trop plein de nouvelles connaissances s'est probablement emparé de moi dans les dernières semaines, et il devenait difficile d'avancer sereinement de plusieurs pas, sans devoir refaire des pas en arrière suite à de mauvais choix effectué par manque de connaissance, ou peut-être d'un manque de soif d'apprendre encore de nouvelle chose.

Je pense que j'aurais été plus à l'aise avec un Travail de Bachelor portant sur un projet de réalisation de Back-end, ou la réalisation d'une application Java, mais je n'aurai pas ouvert mon horizon comme j'ai pu le faire avec ce projet, et je pense qu'il est important de le faire. C'est également pourquoi j'ai d'autant plus apprécié le suivi constant de Nastaran, ainsi que de Jonathan et Pascal, que je remercie chaleureusement pour leurs nombreux conseils.
Au niveau technique, je pense que toutes les parties, hors implémentation des algorithmes de détection d'anomalies, sont utilisables. J'en suis donc heureux, même si je regrette de n'avoir pas pu aller plus loin dans ces algorithmes. Je pense toutefois que le défi était trop grand pour une personne seule dans le temps imparti. En effet, la grandeur du domaine de la détection d'anomalies, et la lecture d'article implique un temps énorme pour comprendre ces algorithmes, pouvoir choisir lequel utiliser et en faire un programme fonctionnel avec la solution GridEye.

5.3. Améliorations possibles

Je pense que la plupart des améliorations peuvent se faire dans l'analyse des logs et la mise en relation des informations obtenues dans ces logs. On pourrait donc imaginer réaliser des implémentations fonctionnelles des différents algorithmes présentés à la section 4, qu'ils fonctionnent en temps réel, et que les modèles s'améliorent constamment.
Pour l'architecture actuelle de la Suite Elastic, il faudrait rendre persistant la base de données Elasticsearch. Il faudrait éventuellement gérer la rétention des logs. Il faudrait également rendre persistant les visualisations et les tableaux de bord générés.

ANNEXES

Liste des tableaux

Table des figures

Références

[1] *1.3.5.17.3. Generalized Extreme Studentized Deviate Test for Outliers*. URL : https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h3.htm (visité le 29/07/2020).

[2] *6 Best Log Management Tools For Linux in 2020*. AddictiveTips. Library Catalog : www.addictivetips.com Section : Network Admin. 23 août 2019. URL : https://www.addictivetips.com/net-admin/linux-log-management-tools/ (visité le 08/05/2020).

[3] *Agents Beats : Agents de transfert de données pour Elasticsearch*. Elastic. URL : https://www.elastic.co/fr/beats (visité le 14/07/2020).

[4] Insaf ASHRAPOV. *Anomaly detection in time series with Prophet library*. Medium. Library Catalog : towardsdatascience.com. 4 juin 2020. URL : https://towardsdatascience.com/anomaly-detection-time-series-4c661f6f165f (visité le 30/07/2020).

[5] *Autoregressive integrated moving average*. In : *Wikipedia*. Page Version ID : 966190870. 5 juil. 2020. URL : https://en.wikipedia.org/w/index.php?title=Autoregressive_integrated_moving_average&oldid=966190870 (visité le 29/07/2020).

[6] *Best Log Manager Software & Tools for Log Monitoring & Events for 2020*. ITT Systems. URL : https://www.ittsystems.com/log-manager-software-and-tools/ (visité le 21/03/2020).

[7] *DEPsys solution : GridEye*. DEPsys. URL : https://www.depsys.ch/solutions/ (visité le 16/06/2020).

[8] *DETEC - Stratégie énergétique 2050*. URL : https://www.uvek.admin.ch/uvek/fr/home/energie/strategie-energetique-2050.html (visité le 14/07/2020).

[9] *DNSstuff - Reviews | Opinions | Tools*. DNSstuff. URL : https://www.dnsstuff.com/ (visité le 14/07/2020).

[10] *Elasticsearch : Le moteur de recherche et d'analyse distribué officiel*. Elastic. URL : https://www.elastic.co/fr/elasticsearch (visité le 14/07/2020).

[11] *Empowering App Development for Developers | Docker*. URL : https://www.docker.com/ (visité le 14/07/2020).

[12] *Gestion des logs*. In : *Wikipédia*. Page Version ID : 160258700. 19 juin 2019. URL : https://fr.wikipedia.org/w/index.php?title=Gestion_des_logs&oldid=160258700 (visité le 08/05/2020).

[13] *Google*. URL : https://www.google.com/ (visité le 14/07/2020).

[14] *Google Trends*. Google Trends. Library Catalog : trends.google.fr. URL : https://trends.google.fr/trends/explore?date=2010-04-23%202020-04-23&q=graylog,Loggly,Elasticsearch,Splunk (visité le 08/05/2020).

[15] *Industry Leading Log Management | Graylog*. URL : https://www.graylog.org/ (visité le 14/07/2020).

[16] *Introducing practical and robust anomaly detection in a time series*. Library Catalog : blog.twitter.com. URL : https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html (visité le 29/07/2020).

[17] *Introduction — statsmodels*. URL : https://www.statsmodels.org/stable/index.html (visité le 30/07/2020).

[18] *Introduction to Market Basket Analysis in Python - Practical Business Python*. URL : https://pbpython.com/market-basket-analysis.html (visité le 30/07/2020).

[19] *Introduction to Predictive Analytics Anomaly Detection*. Stratus Innovations Group. 25 avr. 2019. URL : https://stratusinnovations.com/blog/introduction-to-predictive-analytics-anomaly-detection/ (visité le 27/07/2020).

[20] *ISO 8601*. In : *Wikipédia*. Page Version ID : 169232031. 6 avr. 2020. URL : https://fr.wikipedia.org/w/index.php?title=ISO_8601&oldid=169232031 (visité le 08/05/2020).

[21] JAEL24. *Jael24/TB_ElasticStack*. original-date : 2020-04-15T08 :28 :17Z. 13 mai 2020. URL : https://github.com/Jael24/TB_ElasticStack (visité le 14/05/2020).

[22] JAEL24. *Jael24/TB_fieldDevices*. original-date : 2020-05-20T12 :44 :21Z. 29 mai 2020. URL : https://github.com/Jael24/TB_fieldDevices (visité le 05/06/2020).

[23] JAEL24. *Jael24/TB_GridEye_Simulator*. original-date : 2020-05-08T10 :12 :58Z. 13 mai 2020. URL : https://github.com/Jael24/TB_GridEye_Simulator (visité le 18/06/2020).

[24] *Java | Oracle*. URL : https://www.java.com/fr/ (visité le 14/07/2020).

[25] *Kibana : Exploration, visualisation et découverte des données | Elastic*. URL : https://www.elastic.co/fr/kibana (visité le 14/07/2020).

[26] Jesus L. LOBO. *Detecting real-time and unsupervised anomalies in streaming data : a starting point*. Medium. Library Catalog : towardsdatascience.com. 12 fév. 2020. URL : https://towardsdatascience.com/detecting-real-time-and-unsupervised-anomalies-in-streaming-data-a-starting-point-760a4bacbdf8 (visité le 29/07/2020).

[27] *Log Analysis | Log Management by Loggly*. URL : https://www.loggly.com/ (visité le 14/07/2020).

[28] *Logiciels de gestion informatique et outils de surveillance à distance | SolarWinds*. URL : https://www.solarwinds.com/fr (visité le 14/07/2020).

[29] *Logstash : collecte, transformation et analyse de logs*. Elastic. URL : https://www.elastic.co/fr/logstash (visité le 14/07/2020).

[30] *Machine Learning pour Elasticsearch*. Elastic. Library Catalog : www.elastic.co. URL : https://www.elastic.co/fr/what-is/elasticsearch-machine-learning (visité le 28/07/2020).

[31] *Market Basket Analysis : Understanding Customer Behaviour*. Select Statistical Consultants. 24 jan. 2017. URL : https://select-statistics.co.uk/blog/market-basket-analysis-understanding-customer-behaviour/ (visité le 30/07/2020).

[32] *Modèle ARIMA avec Python – Prévisions de séries temporelles*. MonCoachData. Library Catalog : moncoachdata.com Section : Analyse de données. 23 jan. 2020. URL : https://moncoachdata.com/blog/modele-arima-avec-python/ (visité le 29/07/2020).

[33] *Moyenne mobile*. In : *Wikipédia*. Page Version ID : 171277243. 25 mai 2020. URL : https://fr.wikipedia.org/w/index.php?title=Moyenne_mobile&oldid=171277243 (visité le 29/07/2020).

[34] *On-demand forecasting with machine learning in Elasticsearch*. Elastic Blog. 10 jan. 2018. URL : https://www.elastic.co/fr/blog/elasticsearch-machine-learning-on-demand-forecasting (visité le 28/07/2020).

[35] *Overview of Docker Compose*. Docker Documentation. 13 juil. 2020. URL : https://docs.docker.com/compose/ (visité le 14/07/2020).

[36] *Performing population analysis | Machine Learning in the Elastic Stack [7.8] | Elastic*. Library Catalog : www.elastic.co. URL : https://www.elastic.co/guide/en/machine-learning/current/ml-configuring-populations.html (visité le 28/07/2020).

[37] *PostgreSQL : The world's most advanced open source database*. URL : https://www.postgresql.org/ (visité le 14/07/2020).

[38] *Processus autorégressif*. In : *Wikipédia*. Page Version ID : 171811973. 8 juin 2020. URL : https://fr.wikipedia.org/w/index.php?title=Processus_autor%C3%A9gressif&oldid=171811973 (visité le 29/07/2020).

[39] *Prophet*. Prophet. Library Catalog : facebook.github.io. URL : http://facebook.github.io/prophet/ (visité le 29/07/2020).

[40] *Quick Start*. Prophet. Library Catalog : facebook.github.io. URL : http://facebook.github.io/prophet/docs/quick_start.html (visité le 29/07/2020).

[41] Sebastian RASCHKA. « MLxtend : Providing machine learning and data science utilities and extensions to Python's scientific computing stack ». In : *Journal of Open Source Software* 3.24 (22 avr. 2018), p. 638. ISSN : 2475-9066. DOI : 10.21105/joss.00638. URL : http://joss.theoj.org/papers/10.21105/joss.00638 (visité le 30/07/2020).

[42] *Recherche & Analyses en Open Source • Elasticsearch | Elastic*. URL : https://www.elastic.co/fr/ (visité le 14/07/2020).

[43] *Régression locale*. In : *Wikipédia*. Page Version ID : 147969596. 29 avr. 2018. URL : https://fr.wikipedia.org/w/index.php?title=R%C3%A9gression_locale&oldid=147969596 (visité le 29/07/2020).

[44] *Réseau de neurones récurrents*. In : *Wikipédia*. Page Version ID : 168319282. 11 mar. 2020. URL : https://fr.wikipedia.org/w/index.php?title=R%C3%A9seau_de_neurones_r%C3%A9currents&oldid=168319282 (visité le 29/07/2020).

[45] Ayushi Sharma SAYS. *Ten alternatives to Cronolog*. Comparitech. Library Catalog : www.comparitech.com Section : Net Admin. 1er mar. 2019. URL : https://www.comparitech.com/net-admin/log-management-tools/ (visité le 08/05/2020).

[46] *Seasonality*. In : *Wikipedia*. Page Version ID : 966842186. 9 juil. 2020. URL : https://en.wikipedia.org/w/index.php?title=Seasonality&oldid=966842186 (visité le 29/07/2020).

[47] *Sematext : Cloud Monitoring & Management Tools*. Sematext. URL : https://sematext.com/ (visité le 14/07/2020).

[48] *SIEM, AIOps, Gestion des applications, Gestion des logs, Machine learning et Conformité*. Splunk. URL : https://www.splunk.com/fr_fr (visité le 14/07/2020).

[49] *Stationnarité d'une série temporelle*. In : *Wikipédia*. Page Version ID : 171114537. 21 mai 2020. URL : https://fr.wikipedia.org/w/index.php?title=Stationnarit%C3%A9_d%27une_s%C3%A9rie_temporelle&oldid=171114537 (visité le 29/07/2020).

[50] *Test grok patterns*. URL : http://grokconstructor.appspot.com/do/match#result (visité le 09/04/2020).

[51] *Top 8 BEST Log Management Software | Log Analysis Tool Review 2020*. URL : https://www.softwaretestinghelp.com/log-management-software/ (visité le 08/05/2020).

[52] *twitter/AnomalyDetection*. original-date : 2014-12-09T17 :46 :24Z. 28 juil. 2020. URL : https://github.com/twitter/AnomalyDetection (visité le 29/07/2020).

[53] *What Is Log Management ? A Complete Logging Guide | The Graylog Blog*. URL : https://www.graylog.org/post/what-is-log-management-a-complete-logging-guide (visité le 08/05/2020).

[54] Willie WHEELER. *Anomaly Detection Using STL*. Medium. Library Catalog : medium.com. 14 juil. 2018. URL : https://medium.com/wwblog/anomaly-detection-using-stl-76099c9fd5a7 (visité le 29/07/2020).

[55] Yue ZHAO. *yzhao062/anomaly-detection-resources*. original-date : 2018-05-16T20 :02 :54Z. 20 juil. 2020. URL : https://github.com/yzhao062/anomaly-detection-resources (visité le 20/07/2020).

A. Fichiers

A.1. logstash.conf

```
input {
    beats {
        port => 5044
        tags => ["filebeat"]
    }

    beats {
        port => 5045
    }
}

filter {
    if ("filebeat" in [tags]) {
  grok {
      match => {"message" => "%{TIMESTAMP_ISO8601:date_hour}%{SPACE}%{LOGLEVEL:log_level
    }%{SPACE}%{JAVACLASS:class} -> %{JAVALOGMESSAGE:log_message}"}
  }
    }
}

output {
    elasticsearch {
        hosts => ["elasticsearch:9200"]
    }
}
```

A.2. filebeat.yml

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - <REPOSITORY>/GridEye_Simulator.log


output.logstash:
  hosts: ["<IP>:5044"]
```

Remplacer `<REPOSITORY>` par la localisation du répertoire contenant le fichier de logs.
Remplacer `<IP>` par l'adresse IP de la machine sur laquelle Logstash tourne.

A.3. filebeat.yml de l'image Docker pour l'implémentation du cas d'utilisation

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
            - /usr/share/filebeat/logs/GridEye_sim.log


output.logstash:
  hosts: ["192.168.1.119:5044"]
```

A.4. metricbeat.yml de l'image Docker pour l'implémentation du cas d'utilisation

```
metricbeat.modules:
- module: system
  metricsets: ["cpu","memory"]
  enabled: true
  period: 10s
  processes: ['.*']
```

```
7
8     cpu.metrics:  ["percentages"]
9     core.metrics: ["percentages"]
10
11  - module: postgresql
12    enabled: true
13    period: 10s
14    metricsets: ["database"]
15    hosts: ["postgres://postgres:5432?sslmode=disable"]
16    username: postgres
17    password: postgres
18
19  - module: docker
20    metricsets:
21      - "container"
22      - "cpu"
23      #- "diskio"
24      - "event"
25      #- "healthcheck"
26      - "info"
27      #- "image"
28      - "memory"
29      - "network"
30    hosts: ["unix:///var/run/docker.sock"]
31    period: 10s
32    enabled: true
33
34  output.logstash:
35    hosts: ["192.168.1.119:5045"]
```

A.5. docker-compose.yml du côté terrain pour l'implémentation du cas d'utilisation

```
1   version: '3'
2
3   services:
4
5     grideye_simulator:
6       container_name: grideye_sim
7       build: ../images/GridEye_Simulator
8       restart: on-failure
9       ports:
10        - 2404:2404
11      volumes:
12        - applicativeLogs:/logs
13
14    filebeat:
15      container_name: fbeat
16      build: ../images/filebeat
17      volumes:
18        - applicativeLogs:/usr/share/filebeat/logs
19
20    metricbeat:
21      container_name: mbeat
22      build: ../images/metricbeat
23      depends_on:
24        - db
25      volumes:
26        - /var/run/docker.sock:/var/run/docker.sock
27
28    # Non-persistant DB
29    db:
30      container_name: postgres
31      build: ../images/postgresql
32      ports:
33        - 5432:5432
```

```
34
35
36 volumes:
37   applicativeLogs:
38     driver: local
39     driver_opts:
40       type: 'none'
41       o: 'bind'
42       device: '<DOCKER_FIELD_PATH>/images/GridEye_Simulator/logs'
```

Remplacer `<DOCKER_FIELD_PATH>` par la localisation du répertoire contenant l'arborescence de la machine « terrain ».

A.6. prophet.py

```
1  import altair as alt
2  import matplotlib
3  import pandas as pd
4  from fbprophet import Prophet
5  from pandas.plotting import register_matplotlib_converters
6
7  register_matplotlib_converters()
8
9  matplotlib.style.use('ggplot')
10
11 def fit_predict_model(dataframe, interval_width=0.85, changepoint_range=0.95):
12     """Create the model used by prophet to detect anomalies
13        Inspired by https://towardsdatascience.com/anomaly-detection-time-series-4
       c661f6f165f"""
14     m = Prophet(daily_seasonality=False, yearly_seasonality=False, weekly_seasonality=
       False,
15                 interval_width=interval_width,
16                 changepoint_range=changepoint_range)
17     m = m.fit(dataframe)
18     forecast = m.predict(dataframe)
19     forecast['fact'] = dataframe['y'].reset_index(drop=True)
20     return forecast
21
22 def detect_anomalies(forecast):
23     """Detect anomalies in the forecast realized by Prophet
24        Copied from https://towardsdatascience.com/anomaly-detection-time-series-4
       c661f6f165f"""
25     forecasted = forecast[['ds', 'trend', 'yhat', 'yhat_lower', 'yhat_upper', 'fact']].
       copy()
26
27     forecasted['anomaly'] = 0
28     forecasted.loc[forecasted['fact'] > forecasted['yhat_upper'], 'anomaly'] = 1
29     forecasted.loc[forecasted['fact'] < forecasted['yhat_lower'], 'anomaly'] = -1
30
31     # anomaly importances
32     forecasted['importance'] = 0
33     forecasted.loc[forecasted['anomaly'] == 1, 'importance'] = \
34         (forecasted['fact'] - forecasted['yhat_upper']) / forecast['fact']
35     forecasted.loc[forecasted['anomaly'] == -1, 'importance'] = \
36         (forecasted['yhat_lower'] - forecasted['fact']) / forecast['fact']
37
38     return forecasted
39
40 def plot_anomalies(forecasted):
41     """Show plot with anomalies
42        Inspired from https://towardsdatascience.com/anomaly-detection-time-series-4
       c661f6f165f"""
43     interval = alt.Chart(forecasted).mark_area(interpolate="basis", color='#adadad').
```

```
    encode(
        x=alt.X('ds:T', title='Time'),
        y='yhat_upper',
        y2='yhat_lower',
        tooltip=['ds', 'fact', 'yhat_lower', 'yhat_upper']
    ).interactive().properties(
        title='Anomaly Detection'
    )

    fact = alt.Chart(forecasted[forecasted.anomaly == 0]).mark_circle(size=15, opacity
    =0.7, color='Black').encode(
        x='ds:T',
        y=alt.Y('fact', title='CPU Utilization [%]'),
        tooltip=['ds', 'fact', 'yhat_lower', 'yhat_upper']
    ).interactive()

    anomalies = alt.Chart(forecasted[forecasted.anomaly != 0]).mark_circle(size=30, color=
    'Red').encode(
        x='ds:T',
        y=alt.Y('fact', title='CPU Utilization [%]'),
        tooltip=['ds', 'fact', 'yhat_lower', 'yhat_upper'],
        size=alt.Size('importance', legend=None)
    ).interactive()

    return alt.layer(interval, fact, anomalies) \
        .properties(width=870, height=450) \
        .configure_title(fontSize=20)


alt.renderers.enable('altair_viewer')

if __name__ == '__main__':
    # a = AnalyseLogs()

    # Dataframe used by the Prophet library
    df = pd.read_csv('test_data_cpu.csv', parse_dates=['ds'])
    df_sorted = df.sort_values(by=['ds'])
    df_sorted['y'] = df['y'].apply(lambda x: x * 100 / 4)

    # Analyze the time series with Prophet model, and show the results
    forecast = fit_predict_model(df_sorted)
    forecasted = detect_anomalies(forecast)
    fig1 = plot_anomalies(forecasted)
    fig1.show()
```

A.7. retrieve_logs.py

```
    def get_all_events(self, index_name):
        """Retrieve all the events, and filter them for detecting CPU overload"""
        es = Elasticsearch()
        all_cpu_utilization = es.search(index=index_name, body={"query": {
            "bool": {
                "must": [
                    {
                        "match": {
                            "agent.type": "metricbeat"
                        }
                    },
                    {
                        "match": {
                            "service.type": "system"
                        }
                    },
                    {
                        "match": {
```

```
                            "event.dataset": "system.cpu"
                        }
                    }
                ]
            }
        }
        }, size=10000)

        all_applicative_logs = es.search(index=index_name, body={"query": {
            "bool": {
                "must": [
                    {
                        "match": {
                            "agent.type": "filebeat"
                        }
                    }
                ]
            }
        }
        }, size=10000)

        # Read all log_message
        for i in range(all_applicative_logs['hits']['total']['value']):
            if "_grokparsefailure" not in all_applicative_logs['hits']['hits'][i]['_source
    ']['tags']:
                self.logs.add(all_applicative_logs['hits']['hits'][i]['_source']['
    log_message'])

        # For each CPU Utilization data, store the applicative logs who arrived in the
    last 20 minutes
        for i in range(all_cpu_utilization['hits']['total']['value']):
            last_applicative_logs = es.search(index=index_name, body={"query": {
                "bool": {
                    "must": [
                        {
                            "match": {
                                "agent.type": "filebeat"
                            }
                        },
                        {
                            "range": {
                                "@timestamp": {
                                    "gte": all_cpu_utilization['hits']['hits'][i]['_source
    ']['@timestamp'][
                                           :-1] + "||-20m",
                                    "lt": all_cpu_utilization['hits']['hits'][i]['_source'
    ]['@timestamp'][:-1]
                                }
                            }
                        }
                    ]
                }
            }
            })

            self.transactions[i] = dict.fromkeys(self.logs, 0)

            if all_cpu_utilization['hits']['hits'][i]['_source']['system']['cpu']['total'
    ]['pct'] > 2.0:
                self.transactions[i]["CPU Overload"] = 1
            else:
                self.transactions[i]["Nominal CPU utilization"] = 1

            for j in self.logs:
```

```
77                  log_is_present = False
78                  for k in last_applicative_logs['hits']['hits']:
79                      if "_grokparsefailure" not in k['_source']['tags'] and k['_source']['
    log_message'] == j:
80                          log_is_present = True
81                          break
82
83                  self.transactions[i][j] = 1 if log_is_present else 0
```

A.8. Version améliorées des topologies Docker

A.8.1. Côté ELK

A.8.1.1. docker-compose.yml

```
1  version: '3'
2
3  services:
4
5    elasticsearch:
6      container_name: elsearch
7      build:
8              context: ../images/elasticsearch
9              args:
10                     elastic_version: 7.8.0
11     environment:
12             discovery.type: "single-node"
13     ports:
14       - 9200:9200
15       - 9300:9300
16
17   kibana:
18     container_name: kib
19     build:
20             context: ../images/kibana
21             args:
22                     elastic_version: 7.8.0
23     links:
24       - "elasticsearch"
25     ports:
26      - 5601:5601
27     depends_on:
28       - elasticsearch
29
30   logstash:
31     container_name: logst
32     build:
33             context: ../images/logstash
34             args:
35                     elastic_version: 7.8.0
36     ports:
37       - 5044:5044
38       - 5045:5045
39     depends_on:
40       - elasticsearch
41       - kibana
```

A.8.1.2. Dockerfile elasticsearch

```
1  ARG elastic_version
2  FROM docker.elastic.co/elasticsearch/elasticsearch:${elastic_version}
```

A.8.1.3. Dockerfile kibana

```
1  ARG elastic_version
2  FROM docker.elastic.co/kibana/kibana:${elastic_version}
```

A.8.1.4. Dockerfile logstash

```
ARG elastic_version
FROM docker.elastic.co/logstash/logstash:${elastic_version}
RUN rm -f /usr/share/logstash/pipeline/logstash.conf
ADD pipeline/ /usr/share/logstash/pipeline/
ADD config/ /usr/share/logstash/config/
```

A.8.2. Côté terrain

A.8.2.1. docker-compose.yml

```
version: '3'

services:

  grideye_simulator:
    container_name: grideye_sim
    build: ../images/GridEye_Simulator
    restart: on-failure
    ports:
      - 2404:2404
    volumes:
      - applicativeLogs:/logs

  filebeat:
    container_name: fbeat
    build:
            context: ../images/filebeat
            args:
                    elastic_version: 7.8.0
    volumes:
      - applicativeLogs:/usr/share/filebeat/logs

  metricbeat:
    container_name: mbeat
    build:
            context: ../images/metricbeat
            args:
                    elastic_version: 7.8.0
    depends_on:
      - db
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock

  # Non-persistant DB
  db:
    container_name: postgres
    build: ../images/postgresql
    ports:
      - 5432:5432


volumes:
  applicativeLogs:
    driver: local
    driver_opts:
      type: 'none'
      o: 'bind'
      device: '<PATH>/images/GridEye_Simulator/logs'
```

Remplacer `<PATH>` par la localisation du répertoire contenant l'arborescence de la machine « terrain ».

A.8.2.2. Dockerfile filebeat

```
ARG elastic_version
FROM docker.elastic.co/beats/filebeat:${elastic_version}
COPY config/filebeat.yml /usr/share/filebeat/filebeat.yml
USER root
RUN chown root:filebeat /usr/share/filebeat/filebeat.yml
USER filebeat
```

A.8.2.3. Dockerfile metricbeat

```
ARG elastic_stack
FROM docker.elastic.co/beats/metricbeat:${elastic_stack}
COPY config/metricbeat.yml /usr/share/metricbeat/metricbeat.yml
USER root
RUN chown root:metricbeat /usr/share/metricbeat/metricbeat.yml
```

B. Détails des tests

B.1. Tests de débit d'ingestion

Graylog (3.2.4 Virtual appliance) avec Filebeat :

Sans extractor (module de Graylog permettant le parsing des logs) :

Début du chargement : 10 :09 :17
Fin du chargement : 10 :09 :24
Temps de chargement : 7 secondes
Débit moyen : ~14'043,47 EPS

Avec extractor :

Début du chargement : 07 :46 :47.153
Fin du chargement : 07 :46 :59.648
Temps de chargement : 12.495 secondes
Débit moyen : ~7'867,55 EPS

Elastic Stack sans Logstash :

Début du chargement : 10 :34 :34.200
Fin du chargement : 10 :35 :03.622
Temps de chargement : 29.422 secondes
Débit moyen : ~3'341,21 EPS

Elastic Stack complet (sans filtrage du log) :

Début du chargement : 10 :44 :11.851
Fin du chargement : 10 :44 :31.920
Temps de chargement : 20.069 secondes
Débit moyen : ~4'898,35 EPS

Elastic Stack complet (avec filtrage du log) :

Début du chargement : 10 :26 :37.308
Fin du chargement : 10 :26 :48.015
Temps de chargement : 10.707 secondes
Débit moyen : ~9'181,38 EPS

Splunk :

Avec filtrage :
Donné (metrics.log de Splunk) : 2'628 EPS

Solarwinds Loggly avec endpoint bulk : Avec Loggly, il est impossible de charger un fichier de plus de 5 MB. Il a donc été réduit à 3'999 lignes.

Début du chargement : 08 :42 :11.360
Fin du chargement : 18 :42 :12.224
Temps de chargement : 0.864 seconde
Débit moyen : ~46'295,14 EPS

C. Journal de travail

C.1. Jeudi 13 février

Première réunion avec Nastaran, Jonathan et Pascal. Jonathan et Pascal ont expliqué leur vision du TB à travers une présentation, puis nous avons planifié le travail de Bachelor. Notamment les dates de fin d'évaluation (avec la présentation à DEPsys), et de fin de développement du use-case.

C.2. Mercredi 19 février

Début du Travail de Bachelor. J'ai commencé par suivre un tuto afin de maîtriser les bases du langage LaTeX, ce qui me sera utile pour tout ce qui est rédactionnel. Ensuite, j'ai commencé à revoir la présentation de Pascal afin de bien comprendre (notamment les technologies que je ne connais pas).

C.3. Jeudi 20 février

J'ai regardé plusieurs vidéos qui présentent les différentes technologies que je dois évaluer.

C.4. Mercredi 26 février

J'ai décidé de commencer à évaluer plus en profondeur Elasticsearch en premier, car Prometheus à comme contrainte de ne pas gérer les logs textuels, mais uniquement des métriques numériques. Cependant, d'après plusieurs lectures, je pense qu'il pourrait être intéressant de mixer les deux solutions. J'ai donc installé les outils de la suite ELK, et suivi des tutos plus concret en ce qui concerne Elasticsearch (insertion de donnée, recherches, etc.).

C.5. Jeudi 27 février

Deuxième réunion avec Nastaran. Elle me propose de recentrer mes recherches sur la partie « Log Analysis », donc rechercher directement l'intégration de l'analyse de logs avec ELK par exemple. Après la réunion, j'ai donc continué mes recherches dans ce sens et ai suivi la vidéo d'Elastic qui concerne l'analyse de logs.

C.6. Mercredi 04 mars

J'ai commencé à écrire ce journal afin de mieux me rappeler de ce que j'ai fais, ainsi que d'être plus structuré. Je commence également à utiliser Zotero, qui permet d'enregistrer tous les liens que je trouve intéressants, ainsi que de créer une bibliographie. J'ai également décidé de m'intéresser à Graylog en plus de ELK.

C.7. Jeudi 05 mars

J'ai exploré plus en profondeur les articles de type « Elastic Stack versus Graylog », et je vais donc inclure la stack « Graylog server, MongoDB et Elasticsearch »dans le comparatif. Cette suite-là me semble très appropriée au traitement et à l'analyse de logs. J'ai été à la réunion avec Nastaran à 11h30. Suite à cette réunion, nous avons décidé qu'il fallait que je fasse une synthèse des mes recherches et que je la présente en quelques slides le jeudi 12 mars.

C.8. Lundi 09 mars

J'ai commencé à faire la synthèse de mes recherches. Je vais donc la faire en 3 étapes :

1. Choix des critères d'évaluations
    a) Selon des recherches au sujet des caractéristiques d'un « Log Management Tool »

2. Choix des outils à évaluer
    a) Pour cette étape, je vais consulter plusieurs classements de système de gestion de logs et choisir ceux qui sont le plus souvent cités. Je vais probablement en prendre 5 ou 6.
3. Synthèse et rédaction des slides

Ce lundi, j'ai défini les critères d'évaluation, selon les demandes de DEPsys ainsi que les critères lu lors de mes recherches.

C.9. Mercredi 11 mars

J'ai fais un tableau pour le choix des outils à tester. J'ai donc effectué un classement selon 4 tops de système de gestion de logs. J'ai également commencé l'évaluation à proprement parler, en particulier sur Elastic Stack et Graylog. J'ai également eu un problème de stockage de la base de donnée Zotero et j'ai perdu toute ma bibliographie.

C.10. Jeudi 12 mars

J'ai continué l'évaluation avec Loggly, j'ai créé la présentation de synthèse pour la réunion avec Nastaran, puis je l'ai présentée.

C.11. Mercredi 18 mars

J'ai remis en place Zotero, cette fois avec une synchronisation en ligne de ma bibliographie. J'ai analysé les différents systèmes de gestion de logs que j'hésitais à inclure dans l'évaluation. J'ai donc écarté ManageEngine EventLog Analyzer pour sa popularité vraiment faible et son manque de documentation, et PRTG Network Monitor, qui est très axé sur l'analyse d'un réseau, comme son nom l'indique. Je vais donc évaluer Splunk.

C.12. Jeudi 19 mars

J'ai fais l'évaluation de Splunk. J'ai également eu la réunion hebdomadaire avec Nastaran.

C.13. Samedi 21 mars

J'ai reformaté mon rapport avec le template LaTeXécrit par Mateo Tutic. J'ai également développé la partie Choix des différents systèmes à évaluer.

C.14. Dimanche 22 mars

J'ai continué la partie Choix des différents systèmes à évaluer. J'ai également téléchargé la Suite Elastic et testé avec les logs systèmes de Ubuntu. Cela fonctionne normalement.

C.15. Lundi 23 mars

J'ai mis en place les systèmes de gestion de logs Elastic Stack, Graylog, Splunk et Loggly. J'ai testé (en insérant des logs et regardant le débit) les 3 premiers. Encore quelques problèmes pour Loggly (pour l'instant, il sauvegarde 1 log avec n lignes dans le message plutôt que n logs avec 1 ligne). J'ai également terminé les tableaux récapitulatifs de l'évaluation de chaque système.

C.16. Mardi 24 mars

J'ai terminé les tests d'ingestions de logs pour les 4 systèmes. J'ai commencé à faire mes slides pour la présentation du 25 mars.

C.17. Mercredi 25 mars

J'ai terminé les slides de la présentation. J'ai fais la présentation du travail de Bachelor à l'entreprise DEPsys. S'en est suivi une discussion avec Pascal, Jonathan, Nastaran et moi au sujet de la suite de l'évaluation de mon TB, puis un ajustement du cas d'utilisation à implémenter fourni par Pascal.

C.18. Jeudi 26 mars

J'ai commencé à refaire des tests d'ingestion de log. Cette fois-ci avec tout le pipeline. Je commence avec Elastic Suite, en y intégrant Logstash afin qu'il filtre les données.

C.19. Mercredi 01 avril

J'ai continué les tests d'ingestion avec Elastic Stack et rencontré beaucoup de problème. J'arrive à faire fonctionner un pipeline Logstash-Elasticsearch-Kibana, et un pipeline Filebeat-Elasticsearch-Kibana, mais pas un contenant les 4 logiciels de la suite.

C.20. Jeudi 02 avril

J'ai continué les tests en tentant plusieurs tutoriaux trouvé sur internet. Mais je rencontre toujours des problèmes. Ils sont probablement liés à la communication entre Filebeat et Logstash. J'ai également suivi les tuto officiels de la Suite Elastic, mais ça n'a pas fonctionné non plus. Ceci est peut-être dû à mes fichiers de configurations des logiciels Filebeat et Logstash. J'ai ensuite eu une réunion avec Nastaran.

C.21. Mercredi 08 avril

J'ai continué les tests d'ingestion. En suivant les guides du site d'Elastic, j'ai remarqué qu'il y avait toute une section expliquant l'utilisation de la Suite avec Docker. Je me suis dit qu'il y avait plus de chance que cela fonctionne étant donné l'uniformité que propose Docker. Malheureusement, j'ai toujours les mêmes problèmes. Même en prenant un git public sensé fonctionner. Je me dit alors que le problème vient peut-être de mes fichiers de logs de tests (ils contiennent le même log multiplié n fois).

C.22. Jeudi 09 avril

Je suis repassé sur une version non dockerisée de la Suite Elastic. J'ai téléchargé un fichier de log d'un serveur Apache afin de tester la Suite avec un fichier de log réel, et fait d'autres modifications, notamment sur les fichiers de configuration (j'ai créé la partie « filtrage »de Logstash avec un site internet permettant de créer ces filtres de manière incrémentale). Et ça à fonctionné. J'ai ensuite eu la réunion avec Nastaran.

C.23. Lundi 13 avril

J'ai effectué les tests de performances avec filtrage de la Suite Elastic. Après ceci, je me suis lancé dans les tests avec filtrage de Graylog. Cette fois-ci, ce n'est pas avec un logiciel intermédiaire comme Logstash, mais avec une fonctionnalité intégrée à Graylog : les Graylog Extractors.

C.24. Mercredi 15 avril

J'ai commencé à cherché une façon d'effectuer le filtrage avec le système de gestion de logs Splunk. Malheureusement, j'ai l'impression que cela va être plus compliqué car Splunk favorise l'extraction des informations après l'indexage. Je vais encore chercher une journée, et si ce n'est pas concluant, je passerai outre.

C.25. Jeudi 16 avril

J'ai tenté d'effectuer l'extraction d'informations dans les logs durant la phase de "parsing" du pipeline de Splunk. J'ai vu qu'il devait être possible de le faire en modifiant des fichiers de configuration dans le répertoire de Splunk, mais cela n'a pas fonctionné.

C.26. Lundi 20 avril

J'ai rédigé les résultats des tests d'ingestion. J'ai ensuite commencé à étudié les manières de faire des tests de consommation CPU. Sachant qu'avec Amazon Web Service (AWS), comme je l'avais vu quelques semaines plus tôt dans un cours de Cloud Computing, il est possible de monitorer différentes métriques d'une instance, entre autres l'utilisation du CPU, je me suis lancé dans une installation de la Suite Elastic sur une instance t2.micro d'AWS. Malheureusement, ces instances sont trop petites et ne supportent pas simplement Elasticsearch. Ne voulant pas payer pour des instances plus grosses, je me suis rabattu sur la solution locale. Je vais donc simplement stopper le maximum de processus et monitorer l'utilisation de mon CPU avec l'outil natif d'Ubuntu. Je dois aussi installer Filebeat et Graylog sur un autre ordinateur afin de pouvoir faire ces tests (dans la réalité, le serveur et le client ne seront pas sur la même machine).

C.27. Mercredi 22 avril

J'ai effectué les tests de consommation CPU de la Suite Elastic, de Graylog, ainsi que de Filebeat. J'ai ensuite commencé à rédiger la synthèse de ces tests.

C.28. Jeudi 23 avril

J'ai continué la rédaction, j'ai ajouté le test d'ingestion d'un grand nombre de logs, la comparaison de popularité.

C.29. Vendredi 24 avril

J'ai rédigé le cahier des charges, puis ai eu une réunion avec Nastaran. Nous avons discuté du cahier des charges, des améliorations à y apporter. Ensuite, nous avons un peu regardé l'état du rapport actuel. Il faut entre autres pense a y ajouter les références (tableaux, image, bibliographie). Nous avons ensuite convenu de fixer une réunion avec Jonathan et Pascal afin de leur présenter les résultats de l'évaluation. J'ai donc fixé cette réunion au jeudi 30 avril.

C.30. Mercredi 29 avril

J'ai modifié le rapport pour y ajouter les références sur les tableaux et sur les figures. J'ai également amélioré les annexes avec la table des tableaux et la table des figures. J'ai également commencé la conclusion finale de l'évaluation, en rédigeant le paragraphe concernant l'analyse théorique.

C.31. Jeudi 30 avril

J'ai continué le rapport. J'y ai ajouté une conclusion finale à l'évaluation. J'ai ensuite eu une réunion avec Pascal, Jonathan et Nastaran. Je leur ai présenté les résultats de mon évaluation et nous avons convenu ensemble d'effectuer l'implémentation du cas d'utilisation avec le système Elastic Stack.

C.32. Mercredi 6 mai

J'ai pris en main le programme de Jonathan qui simule l'infrastructure GridEye de façon minimale. Je l'ai modifié légérement pour qu'il stocke les logs dans un fichier, plutôt que de les afficher dans la console. J'ai ensuite commencé à rédiger la partie implémentation du cas d'utilisation dans le rapport, tout en testant les commandes que je met dans le rapport sur une installation neuve d'Ubuntu.

C.33. Jeudi 7 mai

J'ai continué la rédaction et la réalisation de l'implémentation du use case. J'ai rédigé les parties installations de la Suite Elastic, Configuration de la Suite Elastic.

C.34. Vendredi 8 mai

J'ai continué la rédaction et la réalisation de l'implémentation du use case. J'ai maintenant un pipeline entier qui fonctionne avec le simulateur GridEye. J'ai également décrit tout cela dans le rapport. J'ai complété la bibliographie.

C.35. Mercredi 13 mai

Je me suis lancé dans la dockerisation de la Suite Elastic. J'ai commencé par le faire avec 3 images Docker séparée, que je lançais avec un Docker run, comme indiqué dans la documentation. Cela fonctionnait bien avec Elasticsearch et Kibana, mais Logstash n'arrivait pas à communiquer avec Elasticsearch. J'ai remarqué que ceci était dû à la séparation des conteneurs effectuée par Docker. En effet, il n'était pas possible pour Logstash d'atteindre Elasticsearch avec l'adresse localhost, car pour le conteneur Logstash, il n'y a que lui sur « son »localhost. J'ai donc décidé de passer à une topologie avec un docker-compose. Ayant plus d'expérience avec les configurations docker-compose que les simples commandes Docker, je pouvais plus facilement créer les liens entre les différentes applications. J'ai donc créé une topologie faisant communiquer toutes les applications de la Suite Elastic. J'ai ensuite rédigé mon rapport et j'y ai inclus un schéma montrant cette topologie.

C.36. Jeudi 14 mai

J'ai continué le rapport avec la partie ELK et l'installation avec Docker et docker-compose. J'ai ensuite commencé à faire un schéma similaire pour la partie terrain, qui est un peu plus complexe. J'ai envoyé une première idée du schéma à DEPsys et Nastaran ainsi que mes questions, comme le rôle de la base de donnée PostgreSQL dans cette architecture. J'ai également eu une réunion avec Nastaran.

C.37. Mercredi 20 mai

Après avoir reçu des réponses sur mes questions ainsi que l'approbation de Pascal quant à mon schéma. Je me suis lancé dans la réalisation de la topologie du côté terrain. J'ai décidé de construire mon conteneur Docker contenant l'application Java du simulateur GridEye à partir d'un fichier `.jar`. J'ai eu quelques problèmes avec le fichier de logs dans lequel le simulateur doit écrire ses logs, et l'agent filebeat lire. J'ai rapidement compris qu'il me fallait là un Volume Docker partagé entre les deux conteneurs.

C.38. Jeudi 21 mai

Je me suis renseigné sur les Volume Docker et leur intégration avec docker-compose. J'en ai donc intégré dans mon architecture pour partager le fichier de logs entre le simulateur GridEye et Filebeat. J'ai continué la réalisation de la topologie côté terrain avec l'intégration de Metricbeat et PostgreSQL.

C.39. Mercredi 27 mai

En vue de la réunion du 28 mai, qui a pour but de présenter mes avancées dans l'implémentation du cas d'utilisation, j'ai terminé la topologie Docker du côté terrain, et ait tout fait fonctionné ensemble (terrain et ELK). J'ai aussi commencer à créer mes premières visualisations et tableau de bord (dashboard) dans Kibana. J'ai également avancé le rapport.

C.40. Jeudi 28 mai

J'ai tenté de lancer un deuxième « device terrain »à l'aide d'une machine virtuelle. Le test s'est bien passé. J'ai ensuite préparé ma présentation, et fait ma présentation à Jonathan, Pascal et Nastaran. Lors de cette réunion, DEPsys était globalement content de l'avancée et de l'architecture mise en place, qui correspondait à leurs attentes. Les prochaines étapes pour le cas d'utilisation étaient principalement situées dans le dashboard de Kibana, afin de créer, par exemple, un dashboard des problèmes, permettant à un opérateur de voir rapidement si une anomalie est présente, ou a été présente, dans le système.

C.41. Jeudi 4 juin

J'ai avancé le rapport, dans lequel j'avais pris un peu de retard suite aux nombreuses implémentations pour le cas d'utilisation.

C.42. Mercredi 10 juin

J'ai commencé à faire des corrections sur le pipeline de Logstash, par exemple faire en sorte que le tag `_grokparsefailure` ne se mette plus sur les logs de metricbeat, ou encore utiliser la date des logs applicatifs comme `time filter`.

C.43. Jeudi 11 juin

J'ai continué les corrections du 10 juin. J'ai également eu une réunion avec Nastaran qui m'a parlé d'algorithme permettant de trouver des corrélations entre les événements, comme le « market basket analysis ».

C.44. Mercredi 17 juin

En vue du rendu du rapport intermédiaire (19 juin), j'ai relu tout le rapport, et modifié certaines parties. J'ai principalement supprimé la partie d'installation de la Suite Elastic sans Docker, qui me semblait peu pertinente. Et j'ai corrigé les fautes d'orthographe/grammaire/etc.

C.45. Jeudi 18 juin

J'ai eu une réunion avec Nastaran, Pascal et Jonathan, dont le but était de parcourir le rapport pour voir s'il leur convenait pour le rapport intermédiaire. Il en est ressorti qu'il était ok, je dois juste modifier l'introduction, et faire une conclusion. Après cette réunion, j'ai donc fait ces deux parties.

C.46. Vendredi 19 juin

J'ai relu et corrigé une dernière fois le rapport intermédiaire avant de le rendre.

C.47. Mardi 23 juin

J'ai commencé aujourd'hui le TB à plein temps (nous avons encore des choses à rendre pour les autres cours d'ici ce soir, c'est pourquoi je n'ai pas avancé plus ces deux premiers jours). J'ai pris en compte les remarques que Pascal m'avait envoyées par mail concernant mon rapport intermédiaire. J'ai donc modifié le rapport par rapport à ses suggestions.

C.48. Mercredi 24 juin

J'ai continué avec ce que j'avais décrit dans la section "suite" de mon rapport intermédiaire. Il y a d'abord des petites améliorations à faire. Je me suis lancé dans l'amélioration des fichiers de configurations, notamment de logstash, et j'ai directement remarqué que la Suite Elastic était passé de la version 7.6.2 utilisé lors de la première partie à la version 7.8.0. J'ai donc fait en sorte de passer à la dernière version en date.

C.49. Jeudi 25 juin

J'ai décidé de rendre plus simple le passage de version, comme la Suite Elastic va encore changer. J'ai donc dans un premier lieu tenté de le faire avec un fichier « .env ». et la fonctionnalité `ENV` de docker/docker-compose, mais j'ai par la suite remarqué que pour ce cas d'utilisation, avec cette architecture, il fallait plutôt utiliser la fonctionnalité `ARG`, qui n'est pas compatible avec les fichiers « .env ». Cela n'est pas aussi bien que ce je voulais au début, mais je préfère ça que de devoir changer l'architecture. J'ai eu un premier rendez-vous avec Nastaran depuis le début du travail à plein temps. Elle m'a donné ses conseils/remarques concernant le rapport intermédiaire. Nous avons ensuite fixé un nouveau rendez-vous pour mardi prochain (30 juin).

C.50. Vendredi 26 juin

J'ai continué avec la modification des problèmes « mineurs », comme le parsing des logs applicatifs, notamment au niveau de la date. Je me suis lancé la-dessus, avec le module `date` de Logstash, mais j'ai rencontré des problèmes. Comme, à priori, je m'attendais à ce que ce genre de correction soit rapide et ne me cause pas de perte de temps, ce qui n'a pas été le cas, je vais, dès la semaine prochaine, m'attaquer aux algorithmes dont nous avons parlés (Market Basket Analysis p. ex.). Ce sont des choses plus centrales du TB, et si le temps me le permet, je reviendrai sur les petites choses comme la date des logs à la fin.

C.51. Lundi 29 juin

J'ai fais des recherches sur les deux algorithmes dont Nastaran m'avait parlé : le Market Basket Analysis, qui devrait permettre de trouver une causalité entre différents événements. Et ARIMA, qui lui devrait permettre de prédire des dépassements de seuils selon les valeurs actuelles d'une *time series*. Je garde aussi en tête ce qu'avait dit Pascal : le but est de transformer les métriques que l'on a actuellement en des visualisations montrant des incidents. Après mes recherches du jour, je conclue qu'il y a deux possibilités principales : Soit je développe moi-même des algorithmes comme MBA ou ARIMA, soit on utilise la version Premium de la Suite Elastic. J'ai regardé quelques vidéos d'Elastic présentant son Machine Learning, qui a l'air très performant, mais qui a un coût. Un développement perso serait gratuit, mais forcément moins bon.

C.52. Mardi 30 juin

J'ai eu un rendez-vous avec Nastaran, dans lequel nous avons conclu qu'il fallait continuer les recherches, commencer à implémenter les algorithmes. J'ai donc continué mes recherches en essayant de trouver des

articles incluant des implémentations d'algorithmes. J'ai également choisi de développer en Python, qui est un langage adapté à ce type d'algorithme (machine learning).

C.53. Mercredi 1 juillet

J'ai commencé l'implémentation de l'algorithme de Market Basket Analysis. J'ai décidé de le tester avec le problème de la surcharge CPU. Pour l'instant j'ai des problèmes avec la manière de l'exécuter, notamment pour créer des « transactions », qui correspondent aux tickets de caisse dans la version pour les magasins. J'essaie de le faire en direct, en lisant les nouveaux logs toutes les 10 secondes. Mais il faut donc récupérer les logs arrivés plus tôt pour créer une transactions avec les causes d'un problème, et le problème en lui-même.

C.54. Jeudi 2 juillet

J'ai continué l'implémentation du Market Basket Analysis, et j'ai décidé de faire des « fenêtres glissantes »qui permettent de créer les transactions. J'ai également décidé de travailler sur l'ensemble des logs et pas uniquement sur les logs en direct, ce qui me permet d'avoir plus de données. J'ai ensuite eu un rendez-vous avec Nastaran. Je lui ai expliqué mon avancée tant dans mon implémentation que dans mes recherches. Nous avons conclu qu'il fallait dans un premier temps finir l'implémentation avec les fenêtres glissantes, malgré les résultats qui ne s'annoncent pas excellent, ceci étant dû principalement à la faible quantité de données. Le MBA est un algorithme qui demande un grand nombre de donnée pour être efficace. Dans un deuxième temps, il faudra que je me renseigne plus en profondeur sur les algorithmes utilisés par la version Premium de la Suite Elastic, et éventuellement l'essayer. Puis, dans un troisième temps, de me renseigner sur l'algorithme ARIMA, qui permet de prédire des valeurs dans le futur. Nous avons fixé le prochain rendez-vous au lundi 6 juillet.

C.55. Vendredi 3 juillet

J'ai continué et terminé l'implémentation du Market Basket Analysis, mais les résultats ne sont pas concluant, ceci étant dû au manque de donnée. En effet, il faudrait en théorie beaucoup de transactions incluant des problèmes de surcharge CPU pour que cela fonctionne, et la, en ayant laissé tourné le simulateur GridEye une bonne partie de la journée, nous avons que 3 surcharges CPU.

C.56. Lundi 6 juillet

Je me suis renseigné sur la version Premium d'Elastic Stack, qui propose un essai gratuit de 30 jours. Je l'ai donc commencé, et il me servira jusqu'à la fin du Travail de Bachelor, qui se termine le 31 juillet. Cette version contient beaucoup de fonctionnalités, regroupée sous le nom de "Machine Learning". Globalement, pour ces fonctionnalités d'apprentissage automatique, il faut toujours un grand nombre de donnée pour que les modèles puisse s'entraîner et fournir des résultats de qualités. Si je veux pouvoir les utiliser convenablement, il faudrait donc que je modifie le simulateur GridEye. Ensuite, j'ai rapidement pu me renseigner sur l'algorithme ARIMA. Il semble plus complexe que le Market Basket Analysis (mathématiquement parlant), et je n'ai pas encore tout saisi. Je dois encore me renseigné, principalement sur comment l'appliquer concrètement à notre utilisation. J'ai ensuite eu un rendez-vous avec Nastaran. Je lui ai expliqué mes recherches et mes avancées sur les 3 axes que l'ont avait définis. Elle m'a dit d'encore m'informer et essayer de voir quels algorithmes sont utilisés par la Suite Elastic, afin de pouvoir les implémenter par moi-même ensuite.

C.57. Mardi 7 juillet

Je suis aujourd'hui allé travaillé à Fribourg avec Mateo Tutic dans un espace de co-working où il a l'habitude de travailler. Ce qui m'a, en soit, fait perdre un peu de temps. J'ai du transférer mes implémentations sur mon ordinateur portable avant de partir, puis de re-lancer le tout avec une machine virtuelle, étant donné

que pour transmettre les logs, il me faut 2 machines. J'ai donc pris le matin pour installer tout ça. Cela m'a permis de voir que mon architecture avec Docker et Docker-compose avait quelques problèmes auxquels je n'avais plus pensé, comme des chemins en dur qui correspondent à mon disque sur mon ordinateur fixe. Il faudra que je change ces choses avant la fin du TB, si j'ai le temps. J'ai ensuite continué à me renseigner sur ARIMA et j'ai commencé une implémentation.

C.58. Mercredi 8 juillet

J'ai rencontré quelques problèmes avec l'implémentation d'ARIMA en python, notamment sur l'utilisation de certaines librairies, probablement dû à la version de python. J'ai également ensuite eu quelques problèmes avec l'utilisation de la librairie Pandas, qui permet de créer et d'utiliser des structures Dataframe. J'ai ensuite, comme discuté avec Nastaran, recherché des informations sur les algorithmes utilisés par la Suite Elastic dans leur outil « Machine Learning ». Malheureusement, pour la partie la plus intéressante, qui est l'apprentissage automatique non-supervisé, Elastic essaie plutôt de présenter ceci comme une boîte noire utilisable en quelques clics. Ils ne donnent donc pas les algorithmes utilisés derrière. J'ai ensuite regardé des webinars d'Elastic qui expliquent l'utilisation de leurs algorithmes, ce qui m'a permis de mieux les comprendre. Il me faudrait maintenant l'avis de DEPsys afin de savoir quels algorithmes seraient les plus utiles à ce projet.

C.59. Jeudi 9 juillet

J'ai eu un rendez-vous avec Nastaran, au cours duquel je lui ai présenté l'avancée de mon travail et de mes recherches. Nous avons conclu que nous avions maintenant besoin de l'avis de Pascal et Jonathan pour choisir où s'orienter. Il faut en effet leur connaissance du terrain pour choisir les algorithmes.

C.60. Lundi 13 juillet

Après cette première partie de travail de Bachelor à plein temps, je n'avais plus beaucoup écrit dans le rapport. J'ai donc utilisé cette journée pour appliquer les conseils que Nastaran m'avait donnés après le rendu du rapport intermédiaire, et j'ai également commencé à écrire la synthèse de mes recherches. Jusque là, j'ai fais beaucoup de recherche pour peu de résultats concrets.

C.61. Mardi 14 juillet

J'ai lu des articles, en prévision de la réunion de demain avec DEPsys. J'ai également encore testé l'algorithme ARIMA sur des données autres que celles du simulateurs, afin de m'assurer de son bon fonctionnement.

C.62. Mercredi 15 juillet

J'ai eu un premier rendez-vous avec DEPsys (Jonathan et Pascal), et Nastaran depuis le début du travail à plein temps. Nous avons parlé de l'avancée du projet et des recherches. Je leur ai expliqué les différentes propositions d'algorithmes que j'ai rencontré, que ce soit dans mes recherches hors Elastic, ou dans la Suite Elastic et son outil Machine Learning. Je leur ai également expliqué qu'un des problèmes était le nombre de données à disposition, qui est faible, et qui réduise ainsi la possibilité de test des algorithmes. En conclusion de cette réunion, DEPsys a indiqué que les algorithmes les plus intéressants étaient l'analyse de métriques qui permet d'avoir des seuils « actifs »ainsi que de faire de la prédiction de valeurs. Et un algorithme permettant de trouver des symptômes de panne. Pascal et Jonathan trouvent aussi qu'il sera difficile de fournir un résultat comparable à ce que propose Elastic, et donc propose d'effectuer encore des recherches dans le domaine de la détection d'anomalies, qui est très vaste. Le planning pour la fin du TB est donc de faire des recherche durant cette fin de semaine, puis d'utiliser la semaine prochaine pour implémenter ce qui aura été trouvé, et la dernière semaine sera utilisée pour le rapport.

C.63. Jeudi 16 juillet

J'ai re-commencé à faire de la recherche en utilisant cette fois des mots-clés comme *anomaly detection* ou *outlier detection*. Le nombre de documents universitaires parlant de ce sujet est grand et les documents sont eux-mêmes relativement long à lire. Je remarque qu'il existe énormément d'algorithmes autres que ceux utilisés jusque là et qu'ils sont, en général, assez compliqué à comprendre au niveau des mathématiques utilisés.

C.64. Vendredi 17 juillet

J'ai continué mes recherches, et je prend note des différents algorithmes que je rencontre. J'essaie d'en apprendre plus à chaque fois mais cela est rapidement chronophage, car chaque article cite de nouveau article, et ainsi de suite.

C.65. Lundi 20 juillet

J'ai décidé de m'accorder une journée de plus de recherche, car j'ai remarqué que j'avais une grande quantité d'information sur différents algorithmes, permettant de faire différentes opérations, mais que je ne les connaissais pas encore suffisamment et qu'il m'était impossible de choisir avec lequel tenter une implémentation. J'ai ensuite repris l'implémentation d'ARIMA permettant de prédire une valeur de série temporelle dans le temps, en suivant des tutos. J'ai ensuite remarqué qu'il me faudrait des données afin de tester la prédiction. J'ai donc commencé à modifié le simulateur GridEye pour y ajouter un endpoint qui lance une fonction avec une fuite mémoire. Je n'ai pas pu finir cette fonctionnalité aujourd'hui.

C.66. Mardi 21 juillet

J'ai eu un rendez-vous avec Nastaran et Jonathan. Je leur ai expliqué que le domaine est plus grand que ce que j'avais vu dans un premier lieu, et que je n'arriverais probablement pas à développer quelque chose d'une performance comparable à ce qu'Elastic propose. Ils m'ont répondu que ce n'était pas un problème, qu'il fallait que je développe ce que je pouvais et que je n'ésite pas à écrire dans le rapport si je jugeais impossible d'arriver à quelque chose de concluant seul, et dans le temps d'un TB. Il a aussi été soulevé que si les algorithmes ne pouvaient pas être testés avec des données se rapprochant de l'infrastructure réelle, ce n'était pas un problème. Les programmes fourniraient un début si par la suite DEPsys voulait continuer le développement, dans le cadre d'un autre TB ou en interne. J'ai ensuite continué l'implémentation d'ARIMA, mais avec une plus grande connaissance de la base de l'algorithme.

C.67. Mercredi 22 juillet

J'ai un algorithme d'ARIMA fonctionnel, en utilisant une bibliothèque python nommée statsmodels. Elle fonctionne très bien sur des données fourni ayant une croissance dans le temps, et détermine une valeur future selon la tendance globale. Ensuite, j'avais beaucoup vu le nom de la détection d'anomalie de Twitter dans mes recherches, j'ai donc voulu l'essayer. Malheureusement, cette bibliothèque est disponible en langage R. J'ai cependant trouvé plusieurs implémentations en Python. J'en ai essayé deux (tad de Marcnuth, et pyculiarity), mais malheureusement, j'ai rencontré des problèmes à chaque fois, probablement à cause des versions de python.

C.68. Jeudi 22 juillet

J'ai encore essayé de suivre un tutoriel utilisant pyculiarity, une bibliothèque Python de l'implémentation du Twitter's AnomalyDetection. J'ai essayé avec un python normal (fichier `.py` et virtualenv), puis avec `Jupyter notebook` et `conda` pour l'environnement virtuel, mais je rencontre à chaque fois des problèmes. Étant donné le peu de temps qu'il me reste, j'ai décidé de me lancer dans la bibliothèque de Facebook,

Prophet, que j'ai découvert plus tard, et qui est elle disponible en R et en Python. J'ai réussi à obtenir quelque chose de convaincant.

C.69. Vendredi 23 juillet

J'ai commencé à écrire un peu le rapport, et j'ai amélioré le code, qui n'est pas très élégant.

C.70. Lundi 27 juillet

J'ai essayé d'utiliser le filtre « date »de Logstash afin d'utiliser la date-heure des logs applicatifs, sans succès. Je ne vois malheureusement pas ce qui pourrait poser problème. Je suis ensuite passé sur le rapport. J'ai commencé par y ajouter les pages "formelles" comme l'authentification. Je me suis fortement inspiré du template du Prof. Sylvain Pasini. J'ai ensuite écrit la suite du rapport, et fait un schéma de l'illustration des détections d'anomalies.

C.71. Mardi 28 juillet

J'ai écrit le rapport, en particulier le résultat de mes recherches concernant les différents algorithmes et méthodes existantes.

C.72. Mercredi 29 juillet

J'ai continué le rapport. J'ai ajouté des illustrations aux textes afin de rendre la lecture plus agréables. Après avoir vu une illustrations faite à la main en cherchant une image pour la détection d'anomalies, j'ai aimé et décidé de faire mes illustrations moi-même.

C.73. Jeudi 30 juillet

J'ai continué le rapport. J'ai notamment écrit les parties officielles comme le résumé publiable, l'affiche, etc. J'ai également fait signer les documents à faire signer (clause de confidentialité, résumé). J'ai aussi écrit le rapport concernant la partie de mes implémentations. Puis la conclusion. Puis l'affiche.

C.74. Vendredi 31 juillet

J'ai tout relu et corrigé les fautes de d'orthographe/grammaire. J'ai ajouté les signatures qui m'ont été envoyée. Et j'ai pour finir rendu le rapport en PDF ainsi que les codes en annexes, et l'affiche.