LOG 8371

Ingénierie de la qualité en logiciel

TP 1

Ahmed Hammami
1796523

Anas Bouzinane
1508962

Andrei Catana
1792552

Wassim Maatous
1971584

17-02-2020

Table des matières

1. Introduction

Ce plan d'assurance qualité met au point les méthodes, standards et procédures qui seront utilisés pour assurer le plan de qualité de l'application Elasticsearch selon trois critères : Aptitude fonctionnelle, Fiabilité et Maintenabilité [0].

C'est un document décrivant les dispositions spécifiques prises par une entreprise pour évaluer la qualité du produit ou du service considéré. Il comprend les techniques et les activités à caractère opérationnel qui ont pour but à la fois de piloter un processus et d'éliminer les causes de fonctionnement non satisfaisante à toutes les phases de la boucle de la qualité en vue d'atteindre la meilleure efficacité économique. Il contient : le produit auquel il est prévu d'être appliqué, les objectifs relatifs à la qualité du produit (exprimer ces objectifs en termes mesurables chaque fois que cela est possible), les exclusions spécifiques ainsi que la période de validité.

1.1 Description du système logiciel

Elasticsearch est un moteur de recherche open source (sous la licence Apache) et est écrit en langage Java utilisant la librairie Apache Lucene. Sa fonction principale serait l'indexation et la recherche optimale des mégas donnés. Il peut se comporter comme une base de données NoSQL qui aurait la particularité d'indexer des documents orientés textes. En d'autres termes, c'est un moteur de recherche dont on peut personnaliser des besoins spécifiques de recherche. Il est très apprécié pour sa facilité d'utilisation et sa vitesse d'exécution ce qui fait de lui le moteur de recherche le plus utilisé.

Elasticsearch peut être implémenté comme un moteur de recherche de site web. Il peut également être utilisé pour l'analyse des fichiers de journaux ou dans le cas d'un moteur d'analyse pour la sécurité.

De par sa caractéristique RESTful, Elasticsearch apporte une utilisation plus facile sur le cloud. Parmi autres caractéristiques :

- Multi-locataire :
    - Prise en charge de plusieurs index.
    - Configuration au niveau de l'index (nombre de fragments, stockage d'index, etc.).

- Recherche (presque) en temps réel.
- Cohérence par opération :
    - Les opérations au niveau d'un seul document sont atomiques, cohérentes, isolées et durables.
- Écriture asynchrone fiable pour une persistance à long terme.
- Orienté document :
    - Pas besoin de définition de schéma initial.
    - Le schéma peut être défini pour la personnalisation du processus d'indexation. [1]

1.2 Description des modules évalués

1.2.1 Ingest-user-agent

Ce module sert à extraire les détails à partir de nom d'utilisateur envoyé par le navigateur web avec ses requêtes. Ce module ajoute ces informations par défaut sous le champ ingest-user-agent.[2]

1.2.2 Transport-netty4

Ce module sert à transporter la communication interne entre les nœuds du cluster. En effet chaque appel qui se produit entre un nœud A vers un nœud B utilise le module transport :par exemple lorsqu'une demande HTTP GET est traitée par une premièr nœud puis elle doit être traitée par un deuxième nœud contient les données.

Le mécanisme de transport est de nature complètement asynchrone, ce qui signifie qu'il n'y a pas de thread de blocage en attente d'une réponse.[3]

1.2.3 Ingest-geoip

Le rôle de module ingest-geoip consiste à ajouter des informations sur l'emplacement géographique des adresses IP utilisées. Ce module ajoute ces informations par défaut sous le champs « geoip » .Il est aussi capable de résoudre des adresses à la fois ipv6 et ipv6.[5]

1.2.4 Systemd

Les packages DEB et RPM incluent une unité de service pour les systèmes Linux avec systemd. Sur ces systèmes on peut gérer Metricbeat à l'aide des commandes systemd habituelles.[4]

1.3 Importance de la qualité pour le logiciel dans Digging Corp.

Élue parmi les 3 plus grandes firmes de traitement de données, Digging Corp., certifiée CMMI niveau 3 et spécialisée dans l'apport des solutions utilisant des techniques dans le domaine de la fouille de données, se fixe l'objectif d'améliorer davantage sa position pour le premier trimestre de l'année 2020.

Les 3 produits phares qui font de Digging Corp. un leader dans son domaine sont :

- B&M-Enhancer (ver. 4.8.0) : solution qui permet d'extraire de l'information spécifique dans le domaine des ventes d'une entreprise commerciale. Génère 7.8M$ de revenu brut par année.
- Analyzer (ver. 2.1.8) : solution utilisée par le gouvernement qui permet de miner et traiter des infos pour le facteur de bonheur (*Happiness Factor*). Génère 874.38k$ brute par année.
- MR2I (ver. 10.2.7) : solution qui se base aussi du mining pour faire de la détection des symptômes cérébrale en se basant solennellement sur l'imagerie médicale sans intervention humaine. Génère 12.3M$ brute par année.

Ces trois produits sont intégrés avec Kibana afin d'assurer une certaine visualisation des données.

La solution actuelle commence à démontrer des limitations concernant l'indexation et la recherche rapide des grandes quantités de données passé un certain seuil de l'ordre de 12 TB. Cela est davantage très limitant pour les 3 produits énoncés précédemment de par leurs énormes quantités de données. En effet, les 3 solutions sont critiques en plusieurs aspects et de ce fait nécessitent une solution d'indexation plus performante.

Le rapport financier de la période 2019-2020 fait par le département de finance en étroite collaboration avec le département de recherche et développement a prévu une baisse significative de revenus de 10% par année pour B&M-Enhancer et Analyzer et de 12.3% pour MR2I si la solution actuelle reste en place. L'étude a aussi prévu une hausse de 20% de revenus en moyenne pour les 3 produits si une solution aussi performante que Elasticsearch est appropriée à la place de la solution actuelle.

1.4 Parties prenantes

Les parties prenantes impliqués dans le projet de migration vers le moteur d'indexation Elasticsearch sont présentés ci-dessous :

- Directeur des opérations : Son rôle sera de reporter directement au CEO concernant la décision finale d'adoption de la solution Elasticsearch.
- Chef du produit technique (*Technical product manager*): Sera impliqué tout au long du processus d'adoption. Les tâches de vérifications et de respects d'assurance qualité lui seront délégués.
- Équipe de marketing : L'équipe de marketing est chargée de promouvoir l'amélioration qui résultera de l'adoption du système *Elasticsearch* aux futurs clients potentiels.
- Clients : Les clients principaux des trois produits de Digging Corp. sont les plus concernés dans le projet. Mais tout autre client potentiel (Hôpital, Firme de marketing, etc.) est aussi impliqué.
- Ingénieurs consultants logiciel : Les ingénieurs qui se chargeront de l'adoption du système Elasticsearch.
- Ingénieurs de maintenance : Travaillerons en étroite collaboration avec les ingénieurs consultants logiciel. Ils sont chargés d'assurer la bonne adoption du système *Elasticsearch* comme moteur d'indexation dans les trois produits visés.
- Utilisateur final : Dans ce cas-ci, l'utilisateur final représente le client utilisant les produits de Digging Corp.

1.5 Rôles et responsabilités

Tableau 1: Rôles et responsabilités

Rôle	Nom	Responsabilité
QA Manager	Natan Drake	Supervise l'assurance qualité.
Business Analyst	Stéphanie Barette	Analyser les exigences d'affaires.
Équipe de test	• Catherine Lafleur • Rajul Sunil	L'équipe chargé des tests boites blanche (tests unitaires).
Product Manager	Harold Tran	Se charge de guider le projet et de gérer les équipes impliquées dans le projet.
System Owner	Rami Shawab	Représente l'utilisateur du produit. Donne l'approbation finale du projet Elasticsearch
Analyste Fonctionnel	Martin Bertinghwijg	Étudier les besoins du client. Rédiger/ modifier l'analyse fonctionnelles. S'assurer de la clarté de l'information recueilli.
Ingénieur QA	Ndubueze Fabian Mmagu	Se charge d'assurer la qualité du projet.

2. Éléments de qualité logiciel

2.1 Qualité du logiciel

2.1.1 Définition ISO

Ensemble des traits et des caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés ou implicites.[6]

2.1.2 Définition IEEE

La qualité du logiciel correspond au degré selon lequel un logiciel possède une combinaison d'attributs désirés.[6]

2.2 Plan de qualité logiciel

Le plan de qualité logiciel est un document décrivant les dispositions spécifiques prises par une entreprise pour obtenir la qualité du produit ou du service considéré. Il comprend les

techniques et les activités à caractère opérationnel qui ont pour but à la fois de piloter un processus et d'éliminer les causes de fonctionnement non satisfaisant à toutes les phases de la boucle de la qualité en vue d'atteindre la meilleure efficacité économique. Il contient : le produit auquel il est prévu d'être appliqué, les objectifs relatifs à la qualité du produit (exprimer ces objectifs en termes mesurables chaque fois que cela est possible), les exclusions spécifiques ainsi que la période de validité.[8]

2.3 Critères et sous-critères de l'évaluation

ISO/IEC 25010 repose sur 8 critères de qualité du produit logiciel dont chacun contient plusieurs sous-critères. L'étude faite pour le projet d'intégration est faite selon les 3 critères de fonctionnalité, de fiabilité, et de maintenance. On présentera dans le tableau suivant les sous-critères de chaque critère ainsi qu'une courte définition, de l'objectif et de la mesure de validation prise pour l'évaluer.[7]

2.3.1 Maintenabilité

Le tableau ci-dessous présente les sous critères de la maintenabilité tout en décrivant leurs objectifs et mesures de validation associés.

Tableau 2: Mesures de validation de maintenabilité

Sous-critère	Définition	Objectifs	Mesure de validation
Testabilité – couverture des tests	« La facilité avec laquelle les critères d'essai peuvent être établis pour un système ou une composante et les essais peuvent être réalisés afin de déterminer si ces critères ont été satisfaits ».[7]	Lorsque la suite de test est lancée les lignes de code couvertes sont surlignées avec la couleur verte. $Couverture = \frac{Lignes\ vertes}{Nr.total\ lignes}$	Avoir une couverture de lignes de code supérieure à 65 %.

Modularité	« Mesure dans laquelle un composant du logiciel peut être séparé de manière à minimiser l'impact sur les autres composants lors d'un changement ».[7]	Le nombre de modules couplé entre eux doit rester relativement faible.	La métrique de couplage CBO (*Coupling Between Objects*) entre les modules ne doit pas dépasser 100 dépendences. La métrique de couplage entre les modules de doit pas dépasser 8 liasons.

2.3.2 Fiabilité

Le tableau ci-dessous présente les sous critères de la fiabilité tout en décrivant leurs objectifs et mesures de validation associés.

Tableau 3: Mesures de validation de fiabilité

Sous-critère	Définition	Objectifs	Mesure de validation
Récupérabilité (*Recoverabilty*)	« La mesure dans laquelle le produit peut récupérer les données affectées et ré-établir l'état désiré si une interruption ou une panne se produit ».[7]	Le taux de récupération doit être haut. **Le taux de récupération des données suite à une défaillance est calculé en divisant la quantité de données récupérées par la quantité de données totale.*	Avoir un taux de récupération de 92% en cas de panne et avoir un temps de rétablissement du serveur inférieur à 5 minutes.

Tolérance aux pannes – *(Fault tolerance)*	« La mesure dans laquelle un système ou composant fonctionne comme prévu, malgré la présence de défauts matériels ou de défauts logiciels ».[7]	Assurer l'intégrité des données malgré les défaillances.	Avoir un taux d'intégrité de 99% en tout temps.

2.3.3 Fonctionnalité

Le tableau ci-dessous présente les sous critères de la fonctionnalité tout en décrivant leurs objectifs et mesures de validation associés.

Tableau 4: Mesures de validation de fonctionnalité

Sous-critère	Définition	Objectifs	Mesure de validation
Exactitude fonctionnelle – *(Functional correctness)*	« La mesure dans laquelle le produit donne des résultats corrects avec le degré de précision nécessaire ».[7]	L'indice d'exactitude est calculé en divisant le nombre de réponses correctes par le nombre de réponses total.	L'indice d'exactitude doit être à l'ordre de 99%.
Complétude fonctionnelle *(Functional completness)*	« La mesure dans laquelle le produit couvre L'ensemble des fonctions couvrent toutes les tâches spécifiées et les objectifs de l'utilisateur ».[7]	L'indice de complétude est calculé en divisant le nombre des requêtes récupérées par le nombre de requêtes insérées.	L'indice de complétude doit être à l'ordre de 97%.

2.3.4 Efficacité et performance

Tableau 5: Mesures de validation de l'efficacité et de la performance

Sous-critère	Définition	Objectifs	Mesure de validation
Capacité	« Degré auquel les limites maximales d'un produit ou les paramètres du système répondent aux exigences ».[7]	Trouver le nombre de requêtes maximales qui saturent le système en une seconde.	Taux d'erreur au maximum 0.01% pour les requêtes http envoyées en une seconde.
Utilisation des ressources	« Les quantités et les types de ressources utilisés lorsque le produit exerce sa fonction sous les conditions établies par rapport à une référence établie (benchmark) » [7]	Être capable de contrôler l'utilisation des ressources (CPU et mémoire).	Le taux d'occupation de CPU et de la mémoire.
Efficacité du temps	« Le temps de réponse et les délais de traitement et les taux de débit d'un système lors de l'exercice de ses fonctions dans des conditions déterminées par rapport à une référence établie ».[7]	Le temps de réponse moyen doit être au plus 15 ms.	Le temps de réponse moyen pour une requête HTTP. (La somme du temps de réponse de chaque requête divisé par le nombre de requêtes).

2.4 Assurance qualité logicielle

2.4.1 Définition

L'assurance qualité logicielle est un modèle planifié et systématique des actions nécessaires à fournir pour garantir la conformité des exigences techniques d'un article ou d'un produit. Elle comprend un ensemble d'activités conçu pour évaluer le processus par lequel les produits sont développés ou fabriqués.

C’est aussi un ensemble d’actions de prévention des défauts qui accompagnent le processus de développement des artefacts logiciels. Ce modèle passe par l’élaboration d’un manuel de qualité. [8]

2.4.2 Objectifs de l’assurance qualité logicielle

Les objectifs principaux de l’assurance qualité logicielle sont :

- Assurer un niveau de confiance acceptable pour que le logiciel soit conforme aux exigences fonctionnelles techniques et aux exigences de gestion concernant l'échéancier et le budget.
- Instaurer et gérer des activités visant l'amélioration et garantir une plus grande efficience des activités de développement de maintenance et d'assurance de qualité logicielle. [8]

2.4.3 Contrôle de la qualité

Le contrôle de la qualité correspond à :

- La validation effectuée à la fin du processus de développement pour assurer la conformité aux exigences du produit.
- La vérification effectuée à la fin d'une phase doit s'assurer que les besoins établis pendant la phase précédente ont été satisfaits. [8]

2.5 Stratégie de validation

Digging Corp. repose sur 3 des stratégies les plus répondues pour valider le processus d’intégration du système Elasticsearch. Ces techniques sont présentées dans les sous-sections suivantes :

2.5.1 Revues personnelles

Les développeurs directement impliqués dans le plan d’intégration du système Elasticsearch sont amenés à effectuer une revue personnelle sur le code qu’ils fournissent en guise de mécanisme d’autocorrection. Ils suivront un processus

de revue structuré et auront la responsabilité de détecter la majorité des défauts du système avec une attention particulière aux bogues critiques. Afin de contrôler les revues fournies, chaque développeur devra remplir une liste de tâches définissant les tâches fonctionnelles et les tâches incomplètes et/ou manquantes. [8]

2.5.2 Revues par les paires

Une vérification reposant sur des revues par les paires sera aussi incluse. Cela permet aux développeurs logiciels d'échanger leurs critiques afin d'assurer une meilleure qualité du travail fourni. Chaque personne ayant fait la revue doit remplir une liste dont le rôle sera de déterminer le niveau d'acceptabilité du travail. Pour chaque fonctionnalité, la personne devra fournir une note entre 0 et 2 selon la classification suivante :

- 0 : Il faut refaire la partie évaluée.
- 1 : acceptable, mais peut être optimisé.
- 2 : acceptable et optimisé.

2.5.3 Inspection et Walkthrough

Afin de détecter davantage de problèmes dans le plan d'intégration du système Elasticsearch, il faut organiser des inspections régulières pour chaque code source modifié. L'équipe de gestion de projet Elasticsearch s'assurera de la vérifiabilité des processus, tandis que l'équipe d'assurance qualité s'assurera de l'établissement et du bon traitement des éléments. Le système des *pull requests* et la validation du code sur GitHub sont utilisés dans le processus d'inspection.

3. Plan de tests

Pour assurer la qualité des modules choisis, nous avons évalué le logiciel Elasticsearch selon les critères établis dans la section 1.5. Par ailleurs, nous avons élaboré un plan de tests conforme aux exigences définies avec l'équipe de qualité présentée dans le tableau ci-dessous.

Tableau 6: Objectifs et stratégie te tests

	Sous-critère	Test ou diagnostic	Outil utilisé	Objectif à atteindre
Maintenabilité	Testabilité	Tests de couverture	- Tests unitaires JUnit - Couverture avec Intelij	>= 65%
	Modularité	Diagnostic de couplage	- *Scitool Understand*	Entre les modules <= 8 Entre les classes <= 100
Fiabilité	Récupérabilité	Test de stress	- Script Python de surcharge du serveur	99%
	Tolérance aux pannes	Test de chaos	- Script Python - Coupure d'électricité - Vérification des données	99%
Fonctionnalité	Exactitude fonctionnelle	Test d'exactitude personnalisé	- Envoie de 10000 requêtes - Tests en boite noire	99.99%
	Complétude fonctionnelle	Test de complétude personnalisé	- Script personnalisé - Validation du contenu des 10000 requêtes	99.99%
Efficacité et performance	Capacité	Test de capacité Test de stress	- JMeter	Supporter 500 requêtes simples simultanées d'une taille moyenne de 500 octets par requête.
	Utilisation des ressources	Test de charge	- JProfiler - JMeter	Avoir le taux d'utilisation de CPU au plus à 80% et au plus 70% pour la mémoire.
	Efficacité de temps	Test de charge	- JMeter	Temps de réponse maximum 15ms en 99% des requêtes.

3.1 Rapports des tests et contre-mesures

3.1.1 Mesure de maintenabilité

Pour évaluer le critère de maintenabilité, nous avons lancé les tests unitaires pour chaque module et nous avons effectué une analyse de couverture permettant de valider la testabilité. Les tests unitaires passent à 100% et leurs résultats sont présentés dans l'Annexe II joint avec ce rapport.

3.1.1.1 Testabilité

Ingest.geoip

Le Tableau 6 présente la couverture du module ingest.geoip. Le niveau de couverture du module est de 68% qui est un taux acceptable. En même temps nous avons effectué l'analyse du code source pour identifier les parties non couvertes.

Tableau 7: Couverture des tests du module ingest-geoip

Package	Class, %	Method, %	Line, %
org.elasticsearch.ingest.geoip	88.9% (8/ 9)	50% (26/ 52)	61.4% (224/ 365)

À la suite de l'analyse, nous avons constaté que la quasi-moitié des cas non couverts sont dus à la gestion des cas d'exception. Une des solutions d'optimisation serait de tester encore plus ces cas limites pour assurer le bon fonctionnement de l'application et d'éviter toute défaillance possible due à cette vulnérabilité. Aussi, nous avons identifié deux méthodes non couvertes :

- GeoIpProcessor create( )
- Property parseProperty( )

Netty4

Pour le module Netty4, celui-ci a une bonne couverture de tests avec un taux de 67% comme le montre le tableau suivant :

Tableau 8: Couverture des tests du module netty4

Package	Class, %	Method, %	Line, %
org.elasticsearch.http.netty4	100% (10/ 10)	65.8% (50/ 76)	67.1% (212/ 316)

En analysant le code source, nous avons remarqué que les classes sont très bien testées. Comme dans le cas du module précédent, la plupart des lignes non couvertes sont dues à l'absence de la couverture des cas d'exceptions (ex. : throw, catch etc.).

En ce qui concerne les cas plus spéciaux, la classe Netty4HttpRequest présente un taux faible de couverture de 38%. Parmi les méthodes non couvertes, nous pouvons énumérer :

- List<string> strictCookies()
- HttpRequest removeHeader()

Systemd

Le module systemd quant à lui a une couverture de 72% comme le montre le tableau suivant :

Tableau 9: Couverture des tests du module systemd

Package	Class, %	Method, %	Line, %
org.elasticsearch.systemd	50% (1/ 2)	50% (6/ 12)	72.5% (37/ 51)

C'est le meilleur taux parmi tous les modules analysés. En effet, ce module n'est pas très complexe. Il est composé de deux classes Libsystemd et SystemdPlugin qui comptent au total 47 lignes de code. C'est ce qui explique le taux aussi élevé.

Ingest-useragent

Le module Ingest-useragent est parmi les modules le moins couvertes avec un taux de 45% comme montré par le tableau suivant.

Tableau 10: Couverture des tests du module ingest-useragent

Package	Class, %	Method, %	Line, %
org.elasticsearch.ingest.useragent	63.6% (7/ 11)	55.8% (24/ 43)	45.4% (153/ 337)

Conclusion

Même si les résultats produits par les tests unitaires nous révèlent certain manque de couverture, les tests d'intégration peuvent palier à certaines parties de ce problème. Malheureusement, l'outil d'IntelliJ ne permet pas présentement de tracer les lignes de code couvertes par les tests d'intégration.

3.1.1.2 Modularité

Pour mesurer le couplage des modules, nous avons utilisé le logiciel « ***Understand*** » de Scitools qui nous a permis de visualiser mieux le couplage.[9] Les images de couplage générées indiquent la dépendance entre les modules et les classes en même temps.

En analysant le couplage d'***ingest-geoip,*** nous observons que les liens sont unidirectionnels ce qui indique un bon niveau de modularité.

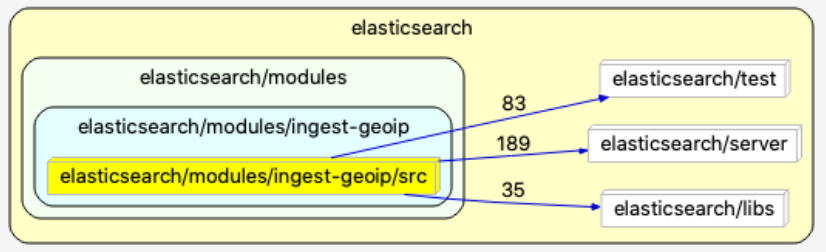


Figure 1: Couplage du module ingest.geoip

En effet, le module a un fort couplage avec deux autres : « server » et « libs », ce qui est normal. « Libs » est la partie du code commune et « server » est la composante principale d'Elasticsearch.

Le module ***transport-netty4*** présente un fort couplage selon la Figure 2. En effet, il est responsable de la gestion des requêtes http : c'est une couche de liaison entre le client et le serveur. Ainsi, le couplage est justifié.

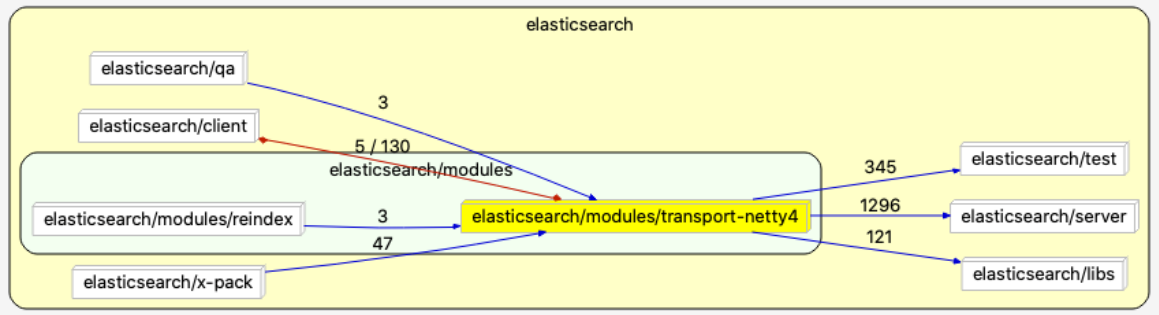


Figure 2: Couplage du module netty4

Le module ***systemd*** et **ingest-user-agent** ont un faible niveau de dépendance comme le montre le deux figures suivantes 3 et 4.

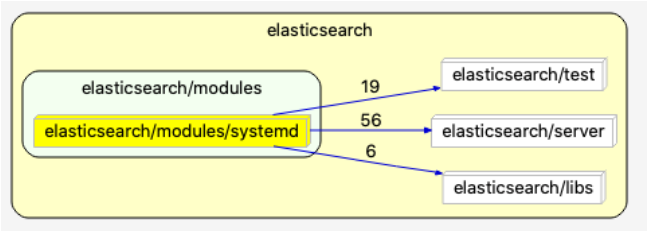


Figure 3: Couplage du module systemd

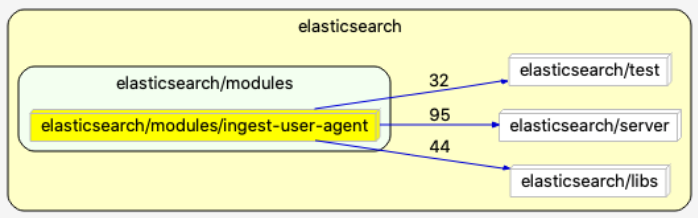


Figure 4: Couplage du module ingest-user-agent

En analysant globalement les 4 module, nous remarquons que le niveau de dépendance entre les modules est faible et ne dépasse pas le seuil établi (8 dépendances entre les modules). Les liaisons sont unidirectionnelles ce qui facilite l'intégration des modules. Cependant, lorsque nous regardons les dépendances entre les classes, le taux varie considérablement. Les modules ingest-user-agent et systemd ne dépassent pas le seuil de 100 dépendances. Le taux CBO maximale du module ingest-geoip est de 189 dépassants le seuil. Finalement, le module transport-netty4 dépasse le seuil CBO avec un taux de 1296. Par contre, ce module est responsable de la communication du protocole http, ce qui explique cette dépendance.

3.1.2 Mesure de fiabilités

3.1.2.1 Récupérabilité

Pour valider les métriques de ce sous-critère, on a simulé une surcharge de requêtes envoyées à l'instance qui roule dans Elasticsearch. En effet, 8 clients roulant des boucles infinies de requêtes simultanées ont essayé de faire tomber le serveur tout en calculant le nombre de requêtes traités. Après avoir répondu à une moyenne de 288 requêtes par secondes pour répondre aux clients $\left(\frac{\frac{(6779+5049+4244+3466+2858+2341+1889+1605)}{8}}{\frac{18+15+14+13+11+10+9+8}{8}} = 288.07\ requêtes\right)$, on remarque que le serveur n'a pas planté. Au contraire, il a coupé toutes les communications courantes et a mis un temps de recouvrement de 43 secondes pour répondre de nouveau aux requêtes des clients. Le code de ce test ainsi que ses résultats se trouvent dans les figures 5 et 6 de l'annexe I joint avec ce rapport.

3.1.2.2 Tolérance aux pannes

Pour valider ce sous-critère, on a inséré des données dans la base de données du système Elasticsearch. Afin de mesurer la capacité du système Elasticsearch à la tolérance aux fautes, on a simulé une coupure d'électricité : on a roulé une instance du serveur du système Elasticsearch puis on a performé la commande « Ctrl + C » pour faire crasher le serveur. Une fois le serveur stoppé, on a relancé Elasticsearch pour vérifier l'intégrité des données insérées. Elasticsearch a donné un taux de 100% de récupération à la suite de ce test et n'a pas perdu son système d'indexation. Vous trouvez les résultats dans la figure 8 de l'annexe I.

3.1.3 Mesure de fonctionnalité

3.1.3.1 Exactitude fonctionnelle

Le test de validation de cette métrique s'étale comme suit :

Tout d'abord, on a lancé 10000 requêtes d'insertion étant toutes différentes dans la base de données du système Elasticsearch. Les 10000 requêtes ont été toutes passées avec succès. Pour mesurer l'exactitude fonctionnelle, on a performé 10000 autres requêtes pour récupérer les données précédemment insérées. Une fois toutes les réponses http reçues, on a comparé les réponses envoyées par le serveur et les réponses attendues et on avait eu un taux d'exactitude

de 99.99%. Vous trouvez le code performant ce test ainsi que ses résultats dans les figures 8 et 9 de l'annexe I.

3.1.3.2 Complétude fonctionnelle

Le même test de l'exactitude fonctionnelle teste aussi la complétude fonctionnelle du Système d'Élasticsearch. Effectivement, le fait d'envoyer 10000 requêtes différentes et récupérer les mêmes 10000 requêtes prouvent que le système d'Elasticsearch à un taux d'exactitude de 99.99% non seulement sur les requêtes stockées, mais aussi sur leurs vrais contenus et leur indexation dans son système de base de données.

3.1.4 Mesure de l'efficacité et de la performance

En utilisant les outils de monitorage nous pouvons tester l'efficacité et la performance du système. Plus précisément, les tests de charge nous permettent de bombarder le système avec des requêtes http afin d'inciter d'avantage l'utilisation des ressources. Les outils comme JProfiler et JMeter nous permettent d'effectuer ces tests.

Pour les effectuer nous avons établi quatre scénarios de charge: réduite, moyenne, élevée et exceptionnelle.

Les résultats de nos tests sont présentés dans la section « Résultats des tests de l'efficacité et de la performance ».

4. Intégration continue

L'intégration continue est une méthode de développement de logiciels permettant d'automatiser plusieurs processus avant le déploiement ou la mise sur le marché de ce logiciel. Cette méthode intervient dans le but de gagner du temps par l'automatisation de tâches récurrentes. Il est préférable d'automatiser l'ensemble de ces tâches dans un script, et de laisser un outil les exécuter afin de ne pas créer d'erreur lors du déploiement et des tests du projet informatique. On a ainsi choisi ***Jenkins*** comme notre outil d'intégration continue. Il intègre non seulement des fonctionnalités intuitives et simples d'utilisation, mais fournit également une visualisation globale et détaillée du processus de déploiement de l'application de bout en bout.

Toutefois, notre plan d'intégration et déploiement continu ne se limite pas juste pour Jenkins, mais pourrait être utilisé avec n'importe quel autre outil de CICD. En effet, nous avons 3 étapes majeures dans notre pipeline :

- Étape 1 : Prendre une version spécifique de l'entrepôt git du projet d'Elasticsearch : on spécifierait par exemple le tag de la version 7.6 lors du clone de l'entrepôt git.
- Étape 2 : Construire et lancer les tests unitaires pour chaque module choisi :
  - Construire et tester le module ***ingest-user-agent***
  - Construire et tester le module ***ingest-geoip***
  - Construire et tester le module ***systemd***
  - Construire et tester le module ***transport-netty4***
- Étape 3 : Construire et lancer tout Elasticsearch. Notez que cette étape a été conçue pour s'assurer du bon fonctionnement des 4 modules qu'on avait choisis afin d'exécuter des tests utilisateur, des tests de performance et d'autres types de tests qu'on pourrait lancer. Elle pourrait être remplacée par le vrai déploiement de notre application qui intégrerait les 4 modules d'Elasticsearch.

La figure suivante montre l'exemple de notre pipeline traduit dans le langage de Jenkins (l'outils choisi pour notre plan d'intégration) :

```
pipeline {
    agent any
    stages("elastic search pipline") {
        stage('Checkout the project to version $VERSION') {
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/$VERSION']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[url: 'https://github.com/elastic/elasti
            }
        }
        stage ('Build and test elastic search chosen modules') {
            steps {
                sh './gradlew :modules:ingest-user-agent:cleanTest :modules:ingest-user-agent:test'
                sh './gradlew :modules:ingest-geoip:cleanTest :modules:ingest-geoip:test'
                sh './gradlew :modules:systemd:cleanTest :modules:systemd:test'
                sh './gradlew :modules:transport-netty4:cleanTest :modules:transport-netty4:test'
            }
        }
        stage ('Deploy elastic search') {
            steps {
                sh 'cd build/elasticsearch-distros/extracted_elasticsearch_$VERSION.0-SNAPSHOT_archive_linux_default/elasticsearch-$VERSION.0-SNAPSHOT/bin && ./elasticsearch &'
            }
        }
    }
}
```

Figure 5: Pipeline Jenkins

5. Nouveau module pour Elasticsearch

5.1 Intégration continue

Un cinquième module pourrait s'avérer utile à être évalué dans la mesure où l'équipe de développement nécessiterait de gérer les sauvegardes (*backups*) des 3 produits B&M-Enhancer, Analyzer et MR2I.

Dans l'affirmative une seule étape de modification dans le pipeline est nécessaire pour intégrer le nouveau module *repository-url.* À l'étape 2 du pipeline de Jenkins nous devons insérer une nouvelle commande pour tester et construire le nouveau module. Il s'agit de la commande :

```
sh './gradlew :modules:repository-url :cleanTest :modules:repository-url:test'
```

Avec cette intervention nous assurons la vérification des critères de qualité du nouveau module. Tout le processus reste inchangé et l'ajout de ce module reste indépendant de l'intégration du projet Elasticsearch dans les 3 produits Digging Corp. C'est une optimisation certes, mais elle n'affecte pas directement les modules critiques du système Elasticsearch.

5.2 Mise à jour du plan de qualité

Après avoir fait les premiers tests selon le plan d'assurance qualité, on assure qu'il est conforme aux exigences définit par l'équipe.

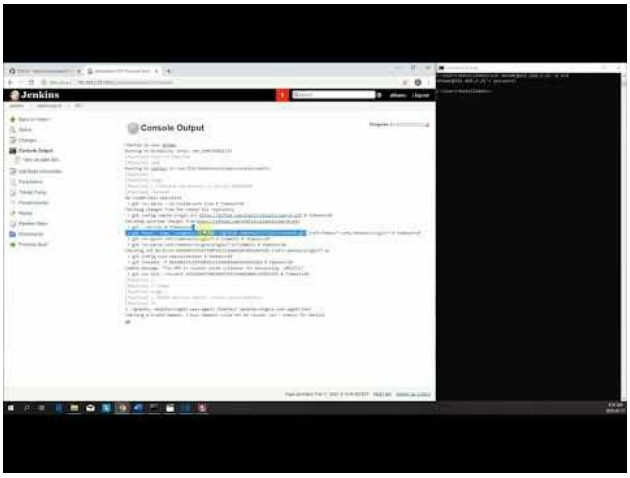
En effet, on a testé l'intégration des 4 modules et d'un cinquième par la suite, et on a trouvé que nos suppositions étaient réalistes. Le 5e module n'affecte pas la qualité des 4 autres modules. Plus encore, assurer la qualité sur quelque chose qui ne le nécessite pas relèvera à un coût supplémentaire de $137.31k pour chacun des 3 produits Digging Corp.

Aussi, la mise à jour du plan d'assurance qualité n'est pas requise, car les sous-critères de performances ne sont pas directement liés aux anciens sous-critères de qualité. Le but étant d'avoir une idée sur la performance d'Elasticsearch. Les 3 produits sont stables et bien définis. Ils suivent la procédure d'intégration continue. Nous avons seulement besoin d'ajouter des tests supplémentaires aux tests déjà existant.

Révision	Date	Auteur	Commentaires
	23/01/2020	Équipe Victor	Première ébauche
1.0	27/01/2020	Équipe Victor	Première version du plan de qualité
1.1	04/02/2020	Équipe Victor	Rectification du plan de qualité
2.0	17/02/2020	Équipe Victor	Version finale du plan de qualité avec l'ajout du nouveau module
3.0	15/03/2020	Équipe Victor	Ajout du critère d'efficacité et performance.
3.1	30/03/2020	Équipe Victor	Mise à jour du plan de qualité.

6. Vidéo démontrant l'intégration et le déploiement en continu

Afin de visualiser le processus d'intégration continue mis en place pour notre projet, on a enregistré une vidéo expliquant les étapes du pipeline Jenkins. La vidéo suivante illustre l'utilisation de Jenkins pour réaliser les étapes décrites dans la section 4 de ce rapport :



Vidéo : Intégration continue et déploiement du projet [10]

Annexe I

test_disponibility.py

C: > Users > RetailAdmin > Desktop > test_disponibility.py

```python
import requests
import time

now = time.time()
i = 0
try:
    while True:
        response = requests.get('http://localhost:9200/twitter/_doc/{}'.format(1))
        if response.status_code not in [200, 201]:
            print("call to the server failed")
        end = time.time()

        i += 1

except Exception as e:
    end = time.time()
    print("Number of calls sent to the server : {}".format(i))
    print("Time for all calls to the server : {} s".format(int(end-now)))

    print("{} at time {}".format(str(e), time.time()))

    now = time.time()

    while True:
        try:
            response = requests.get('http://localhost:9200/twitter/_doc/{}'.format(1))
            if response.status_code in [200, 201]:
                break
        except:
            pass


    end = time.time()
    print("Time to get a response from the server again : {} s".format(int(end-now)))

```

Figure 6: Code Python pour mesurer les sous-critères de la fiabilité du serveur

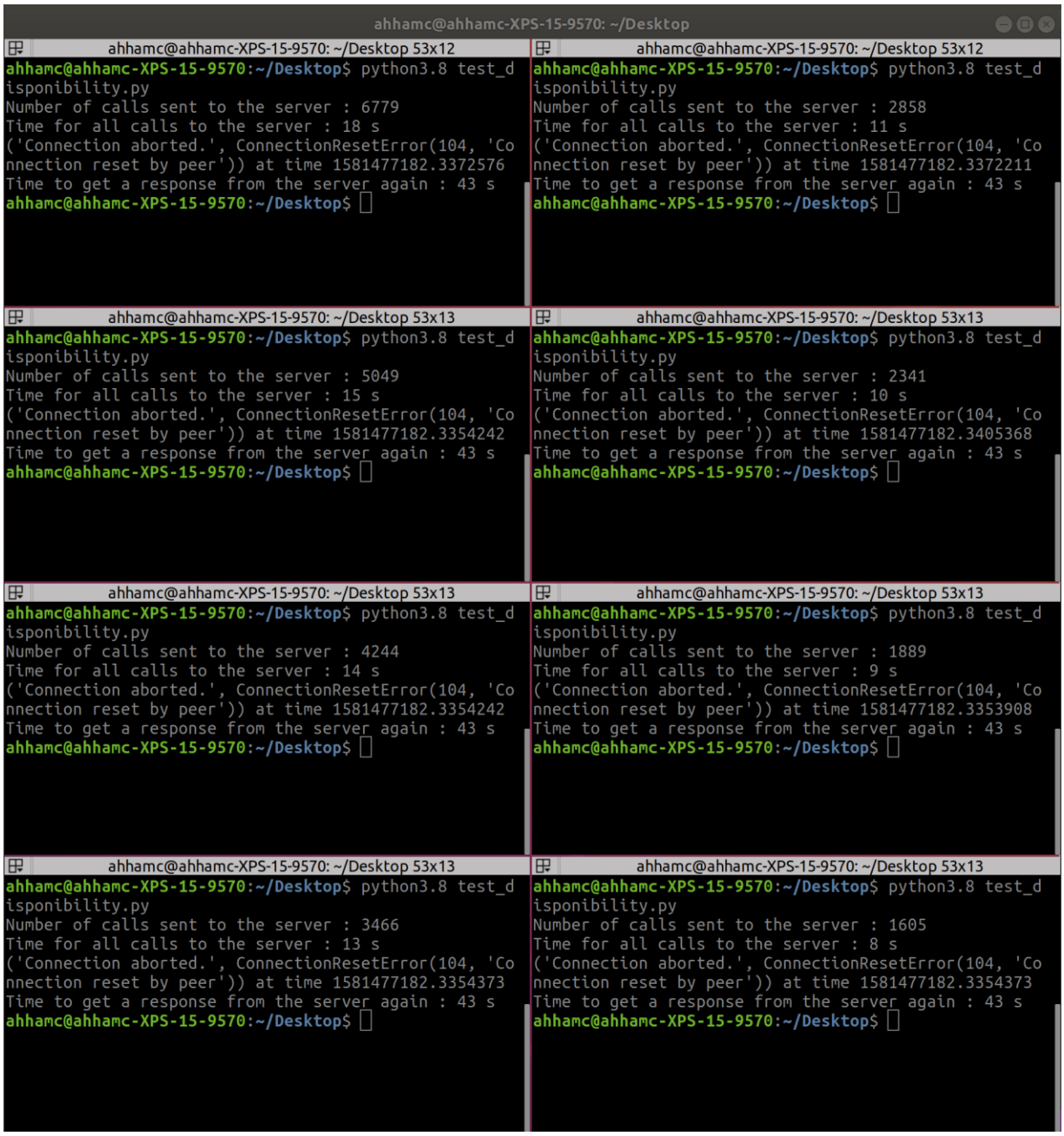


Figure 7: sorties des 8 clients effectuant le code de la figure 5

test_disponibility.py | test_fault_recuperability.py

C: > Users > RetailAdmin > Desktop > test_fault_recuperability.py

```python
import requests

headers = {
    'Content-Type': 'application/json',
}

params = (
    ('pretty', ''),
)

data = '\n{\n    "user": "kimchy",\n    "post_date": "2009-11-15T13:12:00",\n    "message": "Trying out Elasticsearch, so far so good?"\n}'

put_calls_number = 10000
failing_put_calls = 0
failing_get_calls = 0


# for i in range(3, put_calls_number):
#     response = requests.put('http://localhost:9200/twitter/_doc/{}'.format(i), headers=headers, params=params, data=data)
#     if response.status_code not in [200, 201] :
#         failing_put_calls += 1

for i in range(3, put_calls_number):
    response = requests.get('http://localhost:9200/twitter/_doc/{}'.format(i))
    if not response.json().get('found'):
        failing_get_calls += 1

# print("Number of put calls launched : {}\n".format(put_calls_number))
# print("Number of put calls succeeded : {}\n".format(put_calls_number - failing_put_calls))
print("Number of get calls succeeded : {}\n".format(put_calls_number - failing_get_calls))


```

Figure 8: Code Python pour mesurer les sous-critères de la fonctionnalité du serveur

```
ahhamc@ahhamc-XPS-15-9570: ~/Desktop
ahhamc@ahhamc-XPS-15-9570: ~/Desktop 108x55
ahhamc@ahhamc-XPS-15-9570:~/Desktop$ python3.8 test_fault_recuperability.py
Number of put calls launched : 10000

Number of put calls succeeded : 10000

ahhamc@ahhamc-XPS-15-9570:~/Desktop$ python3.8 test_fault_recuperability.py
,Number of get calls succeeded : 10000

ahhamc@ahhamc-XPS-15-9570:~/Desktop$ ,
```

Figure 9: Sorties d'exécution du code de la figure 7

Annexe II

105 test executions in 1 project taking 24.003s

Project	Tests	Outcome	Duration
:modules:transport-netty4	105 passed	PASSED	24.003s
test	105 passed	PASSED	24.003s
org.elasticsearch.http.netty4	18 passed	PASSED	20.144s
Netty4HttpServerTransportTests	7 passed	PASSED	15.443s
Netty4HttpServerPipeliningTests	1 passed	PASSED	2.305s
Netty4BadRequestTests	1 passed	PASSED	2.245s
Netty4CorsTests	5 passed	PASSED	0.090s
Netty4HttpPipeliningHandlerTests	4 passed	PASSED	0.061s
org.elasticsearch.transport.netty4	86 passed	PASSED	9.982s
org.elasticsearch.transport	1 passed	PASSED	3.878s
CopyBytesSocketChannelTests	1 passed	PASSED	3.878s
testSendAndReceive		PASSED	3.878s

11 test executions in 1 project taking 2.711s

Project	Tests	Outcome	Duration
:modules:systemd	11 passed	PASSED	2.711s
test	11 passed	PASSED	2.711s
org.elasticsearch.systemd	11 passed	PASSED	0.435s
SystemdPluginTests	11 passed	PASSED	0.435s
testOnNodeStartedSuccess		PASSED	0.378s
testInvalid		PASSED	0.012s
testCloseFailure		PASSED	0.006s
testCloseNotEnabled		PASSED	0.006s
testCloseSuccess		PASSED	0.006s
testOnNodeStartedFailure		PASSED	0.006s
testIsEnabled		PASSED	0.005s
testOnNodeStartedNotEnabled		PASSED	0.005s
testIsExplicitlyNotEnabled		PASSED	0.004s
testIsNotPackageDistribution		PASSED	0.004s
testIsImplicitlyNotEnabled		PASSED	0.003s

36 test executions in 1 project taking 18.978s

Project	Tests	Outcome	Duration
:modules:ingest-geoip	36 passed	PASSED	18.978s
test	36 passed	PASSED	18.978s
org.elasticsearch.ingest.geoip	36 passed	PASSED	27.281s
GeoIpProcessorTests	17 passed	PASSED	14.425s
GeoIpProcessorNonIngestNodeTests	1 passed	PASSED	7.992s
testLazyLoading		PASSED	7.992s
GeoIpProcessorFactoryTests	15 passed	PASSED	3.858s
IngestGeoIpPluginTests	3 passed	PASSED	1.006s
testThrowsFunctionsException		PASSED	0.642s
testCachesAndEvictsResults		PASSED	0.337s
testInvalidInit		PASSED	0.027s

16 test executions in 1 project taking 3.497s

Project	Tests	Outcome	Duration
:modules:ingest-user-agent	16 passed	PASSED	3.497s
test	16 passed	PASSED	3.497s
org.elasticsearch.ingest.useragent	16 passed	PASSED	0.775s
UserAgentProcessorFactoryTests	8 passed	PASSED	0.433s
UserAgentProcessorTests	8 passed	PASSED	0.342s

Figure 10: Les résultats des test unitaires des 4 modules

Bibliographie

[0]. Smith Jones Rapid Transit, « *Software Quality Assurance Plan Draft* ». U.S. Department of Energy. [En ligne]. Disponible:

- https://www.energy.gov/sites/prod/files/cioprod/documents/qa_plan1.pdf

[1]. GitHub, «*A Distributed RESTful Engine*». [En ligne]. Disponible :

- https://github.com/elastic/elasticsearch/blob/master/README.asciidoc

[2]. Elasticsearch B.V., « *User Agent Processor* ». [En ligne]. Disponible :

- https://www.elastic.co/guide/en/elasticsearch/reference/master/user-agent-processor.html

[3]. Elasticsearch B.V., « *Transport* ». [En ligne]. Disponible :

- https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-transport.html

[4]. Elasticsearch B.V., « *Metricbeat and systemd* ». [En ligne]. Disponible :

- https://www.elastic.co/guide/en/beats/metricbeat/master/running-with-systemd.html

[5]. Elasticsearch B.V., « *GeoIP Processor* ». [En ligne]. Disponible :

- https://www.elastic.co/guide/en/elasticsearch/reference/current/geoip-processor.html

[6]. Université de Montréal, « *Qualité du logiciel et métriques* ». Cours IFT3913. [En ligne]. Disponible :

- http://www.iro.umontreal.ca/~dift3913/cours/A16/Chap1_Introduction.pdf

[7]. Alain April, Claude Y. Laporte, « *Assurance qualité logicielle 1 : Concepts de base* ». Hermes (2011) Cours IFT3913.

[8]. Soumia Ziti, « *Qualité des logiciels* ». Université Mohammed-V de Rabat. [En ligne]. Disponible :

- ftp://ftp.fsr.ac.ma/cours/informatique/ziti/QualiteLogicielle/PolyQL.pdf

[9]. Scientific Toolworks Inc., « *Understand* » software. [En ligne]. Disponible :

- https://www.scitools.com

[10]. Ahmed Hammami, vidéo « *Intégration continue pour Elasticsearch : LOG8371* ». [En ligne]. Disponible :

- https://youtu.be/cJkob2p6Ugo