# Récupération de flux de données personnelles

Livrable unique – 08/05/2016

Polytech'Nantes – département Informatique

Binôme étudiant :
- Guobao LI

Tuteur enseignant :
- Benoît Parrein

Coordinateur :
- JPG

Organisme commanditaire : _____

Tuteur industriel :
- Eric Grall

# Catalogue

# Devoir de confidentialit é

*Le devoir de discrétion est une règle absolue. A remplir et signer dès le début du projet*

Les élèves-ingénieurs :
M. Guobao LI , né le 01/1993 à Canton

s'engagent à ne pas publier ni divulguer de quelque façon que ce soit les informations scientifiques, techniques ou commerciales recueillies ou obtenues par eux au cours de la réalisation du projet décrit dans ce présent rapport, sans l'accord écrit préalable de l'organisme commanditaire.

Cet engagement vaut pour la durée du projet et les 12 mois qui suivent son expiration.

Les élèves-ingénieurs s'engagent à ne conserver, emporter ou prendre copie d'aucun document ou logiciel, de quelque nature que ce soit, appartenant à l'organisme commanditaire, sauf accord de ce dernier.

Cette confidentialité peut s'appliquer aux soutenances de projet des phases 1 et 3 qui dans ce cas, et sur demande écrite de l'organisme commanditaire, se dérouleront à huis clos.

A Nantes, le 07/02/16

"Lu et approuvé"

Signature

Guobao LI

"Lu et approuvé"

Signature

# Charte contre la fraude et le plagiat

*Rappel de la charte signée lors de l'inscription,et que vous vous êtes engagé à appliquer :*

Définitions :
**La fraude** : moyen quelconque pour ne pas être honnête lors d'un devoir surveillé, d'un rendu de projet ou de TP, seul ou en groupe. Pour chaque évaluation réalisée des élèves ingénieurs, la note personnelle ou de groupe doit refléter au mieux l'état des connaissances ou compétences acquises.

**Le plagiat** : c'est l'utilisation non mentionnée de contenu intellectuel déjà réalisé par une tierce personne ou groupe de personnes en vue de réutilisabilité illicite pour ne pas avoir soi même à développer ce contenu. Le plagiat n'est pas plus tolérable ni acceptable que la fraude : en plus de faire croire que l'on est l'auteur de ce que l'on n'a pas fait, on dépouille le véritable auteur de ses droits intellectuels ce qui devient un délit dans la société du savoir. La bonne attitude consiste à beaucoup se documenter mais toujours citer ses sources (textes, code, rendu de tp, etc.).

La fraude et le plagiat sont passibles de sanctions qui peuvent aller jusqu'à l'expulsion de l'Université.

**La bonne attitude** consiste à beaucoup se documenter mais à toujours citer ses sources.
- Tout travail d'un(e) étudiant(e) doit être personnel.
- Lorsque l'on utilise un passage d'un livre, d'une revue ou d'une page Web (traduit ou non), il doit être mis entre guillemets avec mention de la source et de la date.
- Lorsque l'on utilise des images, des graphiques, des données, etc. provenant de sources externes, celles-ci doivent être mentionnées.
- Lorsqu'un travail produit pour un cours est réutilisé pour un autre cours, il convient d'en demander l'autorisation.

Premi ère partie

# Cahier des charges

# 1. Présentation de l'entreprise

Keeme est une startup basée sur un concept innovant, l'internet des objets, créée par Eric Grall. Elle est située 18 rue du calvaire 29000 Quimper, France.

Keeme concentre à fournir une suite de produits qui va collecter les données personnelles concernant la santé et l'activité physique, et ensuite les enrégistrer dans le cloud Keeme. Et puis Keeme pourrait vous proposer à vendre vos données avec d'autres participants afin de créer des packs à forte valeur ajoutée, vous rapportant de l'argent.

# 2. Contexte du projet

« Ces dernières années, le secteur des objets connectés a littéralement explosé. Ce sont notamment les bracelets fitness qui ont envahi le marché. Le succès est tel que de nombreux fabricants – le géant Apple… dernièrement – se sont lancés sur celui des montres connectées. »[1] Donc au fur et à mesure de cette tendance, la quantité de données générées est en plain essor. « Et une étude américaine réalisée en 2011 a estimé que la valeur totale des données personnelles des consommateurs européens valaient 315 milliards d'euros en 2011 et devrait atteindre 1 000 milliards en 2020. »[1] Dans ce cas-là, Keeme fournit une série de produit à collecter les données personnelles pour tous le monde.

« Keeme est une solution pour particulier de gestion de ses données personnelles à fin de stockage et de vente. Keeme s'appuie sur les objets du quotidien (pc, mac, smartphone,…), et sur les objets connectés (bracelet fitness, montre connecté,…). La solution récupère l'ensemble des données de chaque utilisateur afin de les centraliser dans un cloud. A partir de cette plateforme il peut gérer ses données à sa guise, et peut ainsi les revendre. »[1]

Dans ce cadre la start-up a besoin de développer des outils sur lesquels l'application s'appuiera. Il s'agit pour nous de traiter la récupération des données depuis les objets, et la mise en place d'éventuels traitements de ces données. Des grands axes ont déjà été tracés quant à l'architecture de cette partie et les technologies à employer.

Le système à mettre en place consiste en un broker sécurisé et scalable. Ce broker fera le lien entre les objets et le cloud tout en permettant d'implémenter des opérations de traitement sur les données. Le broker sera basé sur Apache Kafka : il fonctionnera donc sur le paradigme publisher-subscriber. Du côté objets, la communication passera par le protocole MQTT. Ainsi il faudra réaliser un connecteur MQTT pour Kafka, qui n'en possède pas pour le moment. Côté cloud la communication se fera via un module Spark Streaming. Pour ces technologies nous seront amenés à programmer en Scala.

De plus, le projet a pour objectif de traiter les données personnelles en appliquant des moyens de machine learning comme création d'un modèle de comportement. Pour cela, il faudra tout d'abord bien comprendre le but souhaité et puis trouver un chemin à le résoudre.

# 3. Modèle du domaine

À partir des différents manuels lus et la réunion avec le tuteur d'entreprise, je suis capable à donner un schéma du modèle du domaine.

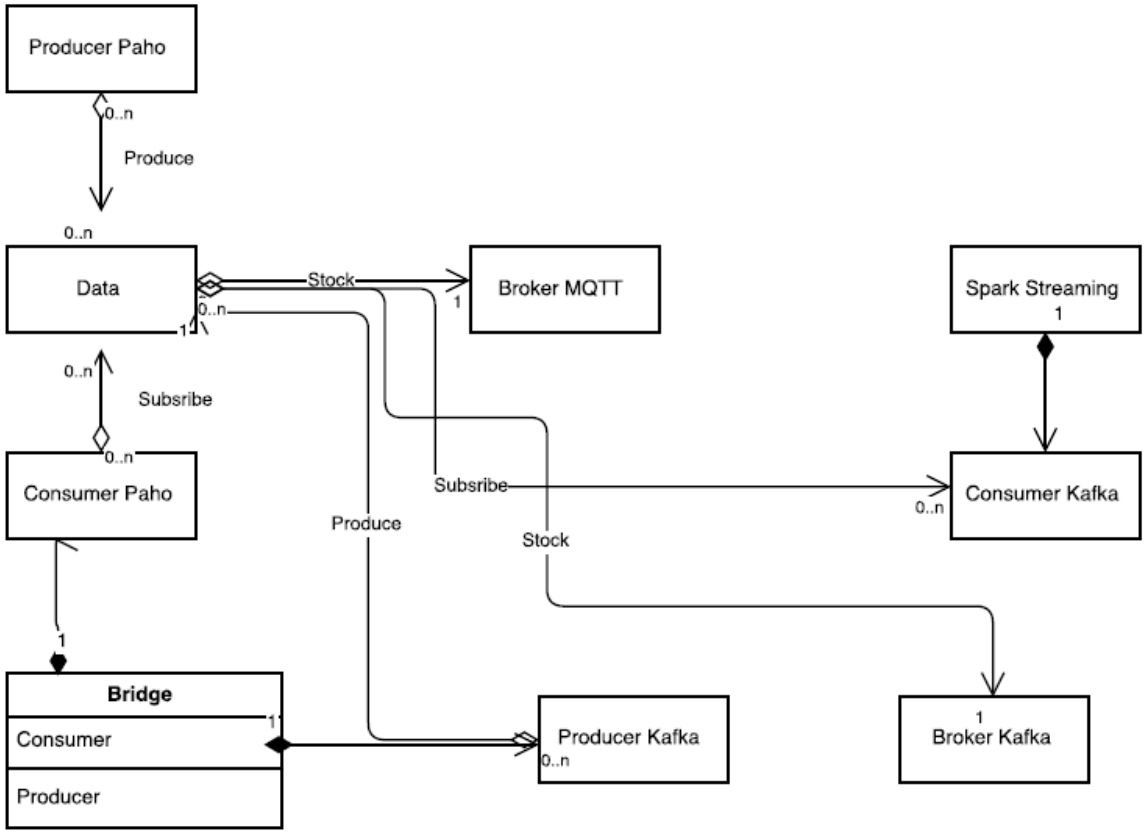



FIGURE 1.1 – Modèle du domaine

# 4. Analyse des exigences particulière par rapport à la qualité du logiciel

| | | | très Faible | faible | moyenne | important | très Important | commentaire |
|---|---|---|---|---|---|---|---|---|
| F | Functionality (fonctionnalité) | adéquation des Fonctions | | | X | | | Le but de projet concentre à l'architecture dans la première partie. |
| | | précision et fidélité des résultats | | | | | X | Pour la partie de traitement de données, les résultats seront importants sur la précision et la fidélité. |
| | | interopérabilité | | X | | | | |
| | | sécurité | | | | X | | Les données personnelles devront être en sécurité dans DB. |
| | | conformité aux exigences fonctionnelles | | | | X | | |
| U | usability (facilité D'emploi) | capacité et facilité de : | | | | | | |
| | | - compréhension | | X | | | | En implémentant le backend d'une service, cela sera une boîte noir pour les clients. |
| | | - apprentissage | | X | | | | |
| | | - exploitation | | X | | | | |
| | | - ergonomie IHM du point de vue métier | | X | | | | |
| R | reliability (diabilité - Sûreté) | maturité | | | X | | | La service devra être résistant à tomber en panne. |
| | | tolérance aux pannes | | | | | X | |
| | | remise en état de marche | | | | X | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| P | performance (efficiency) | temps de réponse t comportement dynamique | | X | | | | |
| | | utilisation des ressources (mémoire, débit en transactionnel, etc) | | X | | | | La service ne sera pas à répondre aux requêtes en temps réel. |
| S | serviceability, maintenability (garantie de service, MCO) | capacité et facilité de : | | | | | | |
| | | - analyse des défaillances | | | X | | | |
| | | - modification | | | X | | | |
| | | - stabilité (confinement des défaillances) | | | | | X | Le broker devra être scalable. |
| | | - test (automaticité, non régression etc) | | | X | | | |
| E | evolution, portability, adaptability (évolutivité) | capacité et facilité de : | | | | | | |
| | | - adaptation et évolution | | | X | | | |
| | | - installation et modifications | | | X | | | |
| | | - remplacement | | | X | | | |
| | | - cohabitation | | | X | | | |

FIGURE 1.2—FURPSE

# 5. Objectifs globaux du projet

Afin de donner les objectifs du notre projet, je dois tout d'abord établir une liste pour préciser les fonctionnalités. Et puis, pendant le processus du projet, en faisant le code je vais rédiger le cahier des charges et la bibliographie.

## 5.1 Liste des fonctionnalités

1. Implémentation d'un bridge faisant passer les données formalisées par le protocole MQTT à partir du broker MQTT au broker Kafka.
2. Implémentation d'un broker MQTT scalable qui permettra de faire passer les données éfficacement dans le cas où les données seront en grande quantité.
3. Implémentation des moyens à traiter et analyser les données en appliquant SparkML.
4. Implémentation du stockage des résultats obtenus dans HDFS et Cassandra, et de l'affichage en appliquant React.js.

# 6. Définition du premier sprint et avancement des premières semaines

| | | | | | |
|---|---|---|---|---|---|
| | | Estimation / affectation | | | |
| A Faire | MQTT et Kafka | 20h | Total plannifié (environ 28h) | 37 | |
| | | | Sous-Total Plannification | | 18 |
| | | | Lecture de la spécification du protocole MQTT | 3 | |
| | | | Lecture de la spécification du Kafka | 3 | |
| | | | L'installation de Kafka | 1 | |
| | | | Comparaison avec des autres connecteurs de MQTT existés | 10 | |
| | LU1 | 20h | Sous-Total Plannification | | 20 |
| | | | Dessin de la diagramme de classe en UML | 1 | |
| | | | Le plan global | 1 | |
| | | | Dessin de la diagramme de Gantt | 2 | |
| | | | Des autres chapitres | 16 | |
| En progès (Th / | Bibliographie Conception du connecteur MQTT de Apache Kafl | 20h | Sous-Total Plannification | | 7 |
| | | | Lecture de la spécification du protocole MQTT | 3 | 2 |
| | | | Lecture de la spécification du Kafka | 3 | 3 |
| | | | L'installation de Kafka | 1 | 1 |
| | | | Comparaison avec des autres connecteurs de MQTT existés | 10 | 1 |

FIGURE 1.3—Sprint1

# 7. Les rapports de stand-up meetings

Nous avons rédigé des fiches de suivi chaque semaine pour enregistrer le travail que nous avons fait. Et au-dessous c'est le rapport de stand-up réunion.

29/09/2015 Au 06/10/2015
TRAVAIL EFFECTUE

Nous avons travaillé en binôme sur le sujet afin de préparer la première réunion. Nous avons tenté de retracer le travail demandé dans ce projet, et de découvrir les technologies impliquées. Nous avons ce mardi rencontré Monsieur Parrein. Monsieur Grall nous a joint par téléphone. Nous avons abordé de nombreux points lors de cet échange : généralités du projet, contexte, objectifs, et notamment les technologies(le broker MQTT, Apache Kafka, Spark Streaming, HDFS, Cassandra, React.js). Nous comprenons que notre travail consistera à mettre en place un broker. Celui-ci fonctionnera avec le protocole MQTT. Il permettra de faire le lien entre des flux de données « publish » et « suscribe », entre des objets connectés et le cloud Keeme. Les données seront stockées par le broker. Nous nous appuierons sur Apache Kafka (ou éventuellement RabbitMQ) et Spark Streaming pour réaliser ce broker. Dans un deuxième temps nous pourrons développer des modules de machine learning associés à un outil de visualisation. Dans la semaine à venir nous allons nous document sur ces technologies. Edgar se chargera de Kafka tandis que Kévin se chargera de Spark. Nous travaillerons ensemble sur MQTT. Pour communiquer avec la start-up nous envisageons d'utiliser Slack.

# 8. Planning du projet global contenant

Nous allons diviser notre projet en quartre phases selon les fonctionnalités implémentées. Dont la première c'est la partie d'implémentation d'un bridge, et la deuxième c'est la partie d'implémentation d'un broker MQTT scalable, la troisième c'est l'implémentation des moyens à traiter et analyser les données en appliquant SparkML, la dernière c'est l'implémentation du stockage et de l'affichage des résultats.

## 8.1 Définition des sprints

Dans la première phase, nous allons le diviser en deux sprints qui est montré à la suite.

| | | Estimation / affectation | | | |
|---|---|---|---|---|---|
| A Faire | | | Total plannifié (environ 28h) | 37 | |
| | MQTT et Kafka | 20h | Sous-Total Plannification | | 18 |
| | | | Lecture de la spécification du protocole MQTT | 3 | |
| | | | Lecture de la spécification du Kafka | 3 | |
| | | | L'installation de Kafka | 1 | |
| | | | Comparaison avec des autres connecteurs de MQTT existés | 10 | |
| | LU1 | 20h | Sous-Total Plannification | | 20 |
| | | | Dessin de la diagramme de classe en UML | 1 | |
| | | | Le plan global | 1 | |
| | | | Dessin de la diagramme de Gantt | 2 | |
| | | | Des autres chapitres | 16 | |

FIGURE 1.4—Sprint 1

| | | | | | |
|---|---|---|---|---|---|
| A Faire | Bridge | 10h | Sous-Total Plannification | | 11 |
| | | | Discussion avec l'entreprise par email | 1 | |
| | | | Comparaison avec un connecteur existant | 2 | |
| | | | Conception général | 2 | |
| | | | Diagramme de classe | 1 | |
| | | | Apprentisage de Scala | 2 | |
| | | | Codage de prototype | 3 | |
| | | | | | |
| | Connecteur de Kafka pour Spark Streaming | 10h | Sous-Total Plannification | | 9.5 |
| | | | Apprentisage de SparkStreaming | 3 | |
| | | | Conception général | 1 | |
| | | | Diagramme de classe | 0.5 | |
| | | | Codage de prototype | 3 | |
| | | | Test de prototype | 2 | |
| | | | | | |
| | LU2 | 10h | Sous-Total Plannification | | 9 |
| | | | Planification de sprints | 2 | |
| | | | La diagramme d'architecture de projet | 0.5 | |
| | | | La bibliographie pour la technologie de Kafka | 1 | |
| | | | La bibliographie pour la technologie de MQTT | 0.5 | |
| | | | La bibliographie pour la technologie de Spark | 1 | |
| | | | La bibliographie pour la technologie de HDFS | 0.5 | |
| | | | La bibliographie pour la technologie de d3.js | 0.5 | |
| | | | La bibliographie pour le microservice | 1 | |
| | | | La bibliographie pour le modèle comportement | 1 | |
| | | | La bibliographie pour la technologie de SparkML | 1 | |
| | | | | | |
| | Préparation de la soutenance | 4h | Sous-Total Plannification | | 4 |
| | | | Préparation des slides | 2 | |
| | | | Préparation du discours pour la soutenance | 2 | |

FIGURE 1.5—Sprint 2

Dans la deuxième phase, nous allons le diviser en deux sprints. Pourtant, les sprints des phases suivantes ne sont pas définies encore, qui seront résoulu à la suite.

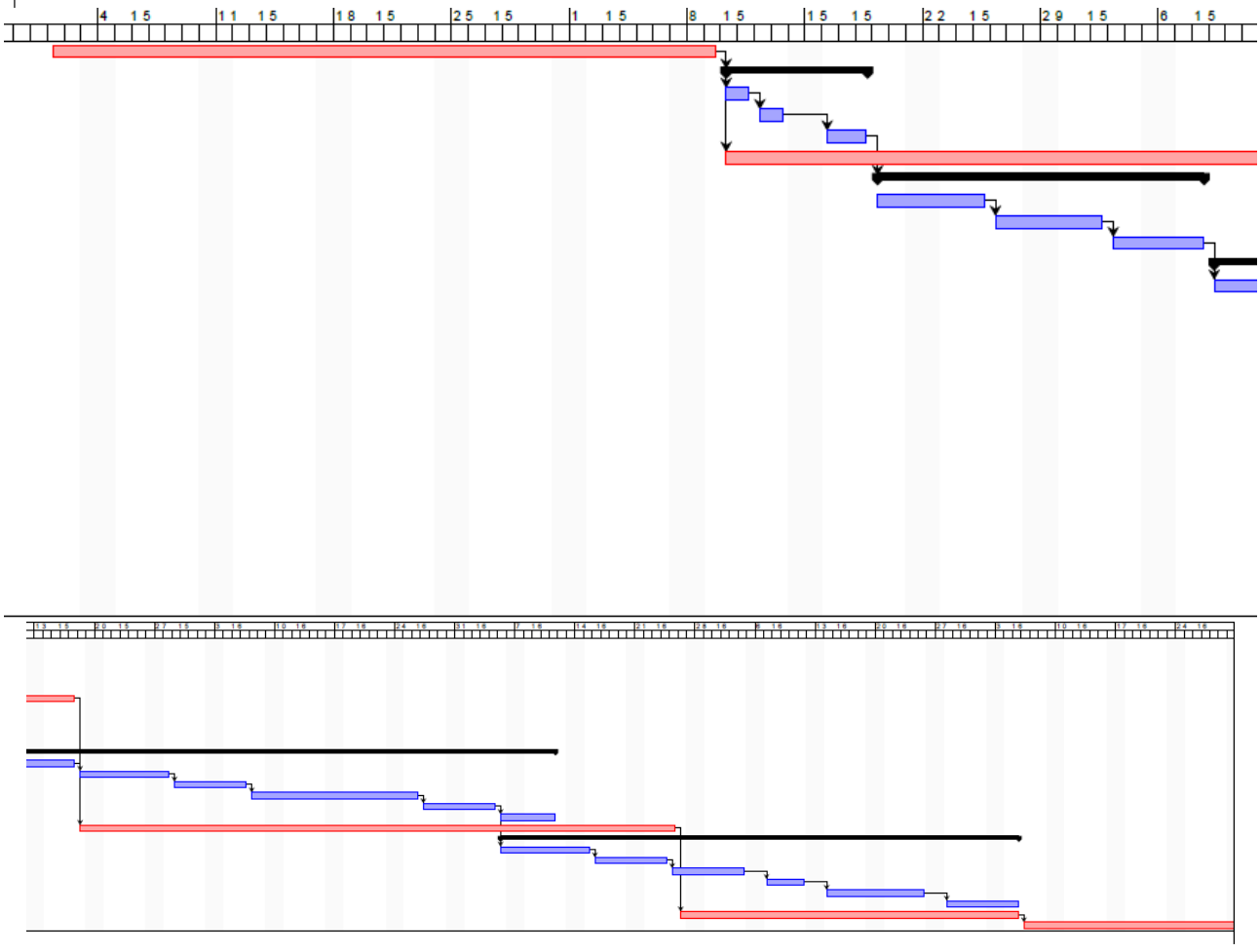
| | | | | | |
|---|---|---|---|---|---|
| | | Estimation / affectation | Total plannifié (environ 28h) | 28 | |
| | Broker MQTT scalable | 10h | Sous-Total Plannification | | 12 |
| | | | Lecture de la référence | 1 | |
| | | | Bibliographie de cette partie | 3 | |
| | | | Conception général | 2 | |
| | | | Codage de prototype | 3 | |
| | | | Test de prototype | 3 | |
| | | | | | |
| | | | | | |
| | | | | | |
| | La partie de Spark Streaming | 20h | Sous-Total Plannification | | 16 |
| | | | Apprentisage de SparkStreaming | 3 | |
| | | | Apprentisage de SparkML | 3 | |
| | | | Modélisation du problème | 3 | |
| ire | | | Codage de prototype | 5 | |
| | | | Test de prototype | 2 | |

FIGURE 1.6—Sprint3

## 8.2 Schéma de Gantt

Nous allons donner le schéma de Gantt à la suite.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | Cahier de charge | 28 | 15-10-1 8:00 | 15-11-9 5:00 | | |
| 2 | | Bridge | 7 | 15-11-10 8:00 | 15-11-18 5:00 | 1 | |
| 3 | | Spécification | 2 | 15-11-10 8:00 | 15-11-11 5:00 | 1 | |
| 4 | | Implémentation | 2 | 15-11-12 8:00 | 15-11-13 5:00 | 3 | |
| 5 | | Test Unitaire | 3 | 15-11-16 8:00 | 15-11-18 5:00 | 4 | |
| 6 | | Bibliographie | 28 | 15-11-10 8:00 | 15-12-17 5:00 | 1 | |
| 7 | | Broker scalable | 14 | 15-11-19 8:00 | 15-12-8 5:00 | 5 | |
| 8 | | Spécification | 5 | 15-11-19 8:00 | 15-11-25 5:00 | | |
| 9 | | Implémentation | 5 | 15-11-26 8:00 | 15-12-2 5:00 | 8 | |
| 10 | | Test Unitaire | 4 | 15-12-3 8:00 | 15-12-8 5:00 | 9 | |
| 11 | | Spark Streaming | 47 | 15-12-9 8:00 | 16-2-11 5:00 | | |
| 12 | | Apprentisage de Spark | 7 | 15-12-9 8:00 | 15-12-17 5:00 | 10 | |
| 13 | | Apprentissage de SparkML | 7 | 15-12-18 8:00 | 15-12-28 5:00 | 12 | |
| 14 | | Analyse de problème | 7 | 15-12-29 8:00 | 16-1-6 5:00 | 13 | |
| 15 | | Modélisation de problème | 14 | 16-1-7 8:00 | 16-1-26 5:00 | 14 | |
| 16 | | Implémentation | 7 | 16-1-27 8:00 | 16-2-4 5:00 | 15 | |
| 17 | | Test Unitaire | 5 | 16-2-5 8:00 | 16-2-11 5:00 | 16 | |
| 18 | | LU3 | 50 | 15-12-18 8:00 | 16-2-25 5:00 | 6 | |
| 19 | | Le stockage et l'affichage | 43 | 16-2-5 8:00 | 16-4-5 5:00 | | |
| 20 | | Apprentisage de HDFS | 7 | 16-2-5 8:00 | 16-2-15 5:00 | 16 | |
| 21 | | Apprentisage de Cassandra | 7 | 16-2-16 8:00 | 16-2-24 5:00 | 20 | |
| 22 | | Apprentisage de React.js | 7 | 16-2-25 8:00 | 16-3-4 5:00 | 21 | |
| 23 | | Spécification | 5 | 16-3-7 8:00 | 16-3-11 5:00 | 22 | |
| 24 | | Implémentation | 10 | 16-3-14 8:00 | 16-3-25 5:00 | 23 | |
| 25 | | Test Unitaire | 7 | 16-3-28 8:00 | 16-4-5 5:00 | 24 | |
| 26 | | LU4 | 28 ? | 16-2-26 8:00 | 16-4-5 5:00 | 18 | |
| 27 | | LU5 | 28 | 16-4-6 8:00 | 16-5-13 5:00 | 26 | |



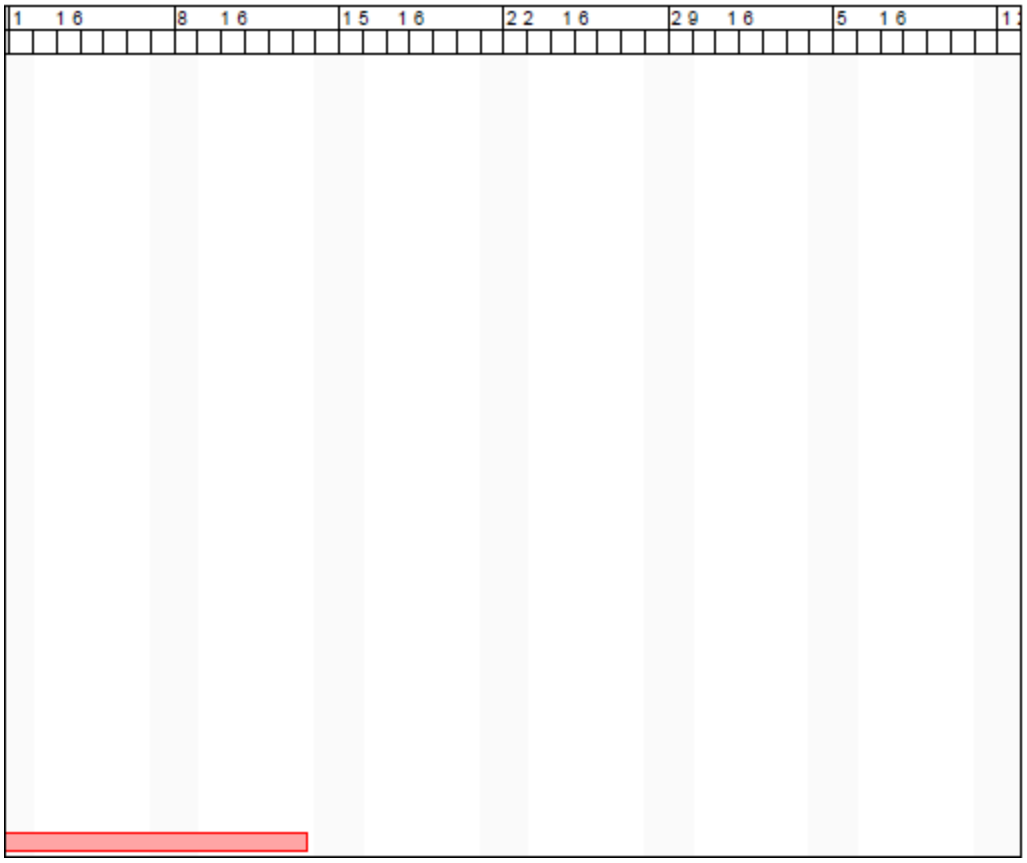


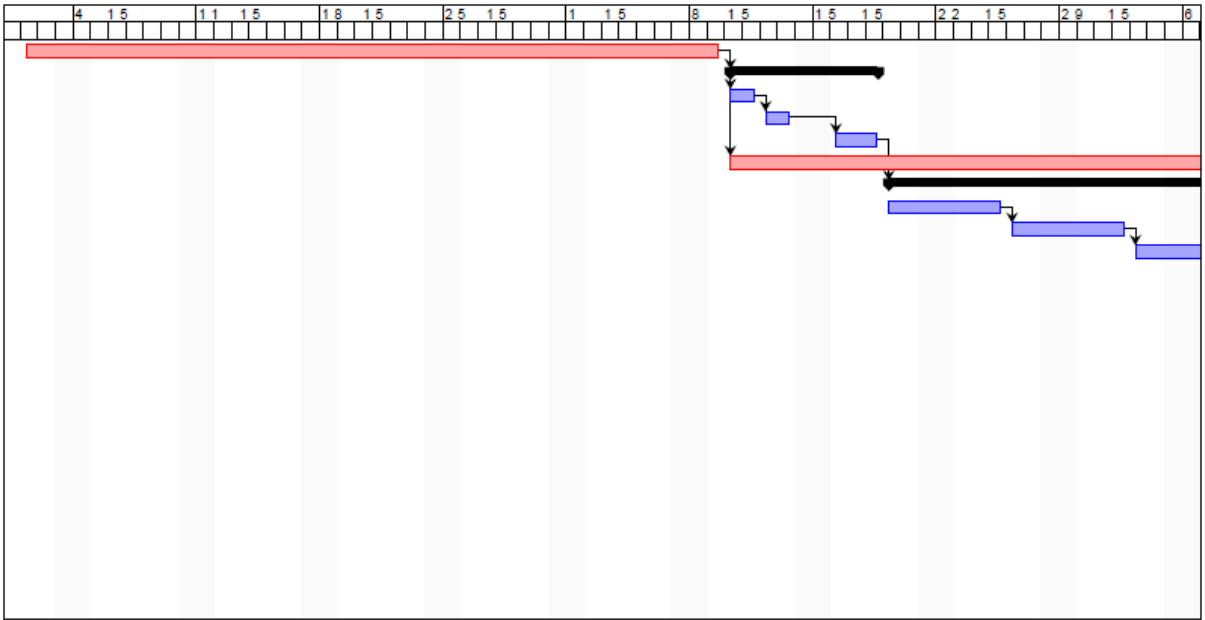
FIGURE 1.7—Schéma Gantt

## 8.3 La modification du schéma de Gantt

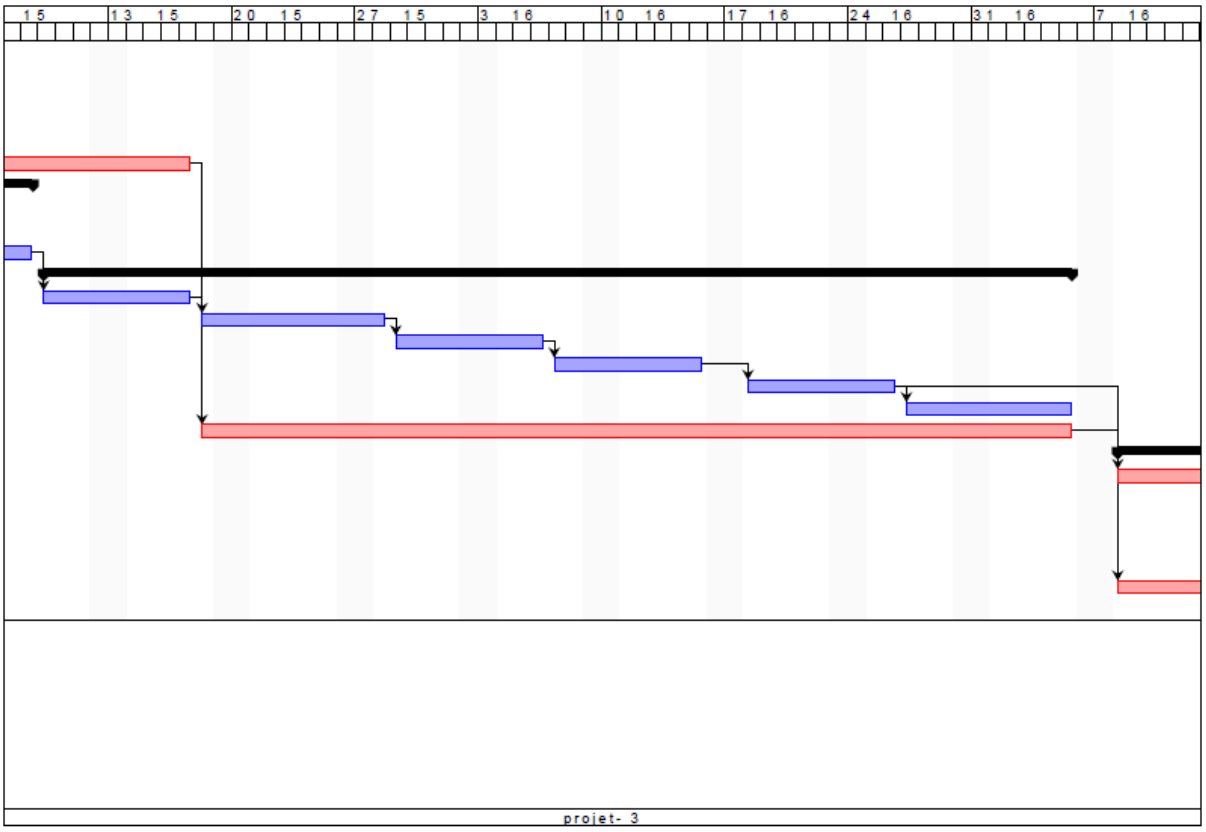
Au fil du temps, on peut se rendre compte que les différents choix sur la création de notre serveur. A la fin, l'idée est qu'on applique l'architecture de microservice à la place de notre proposition. Cela peut apporter le changement de notre plan.

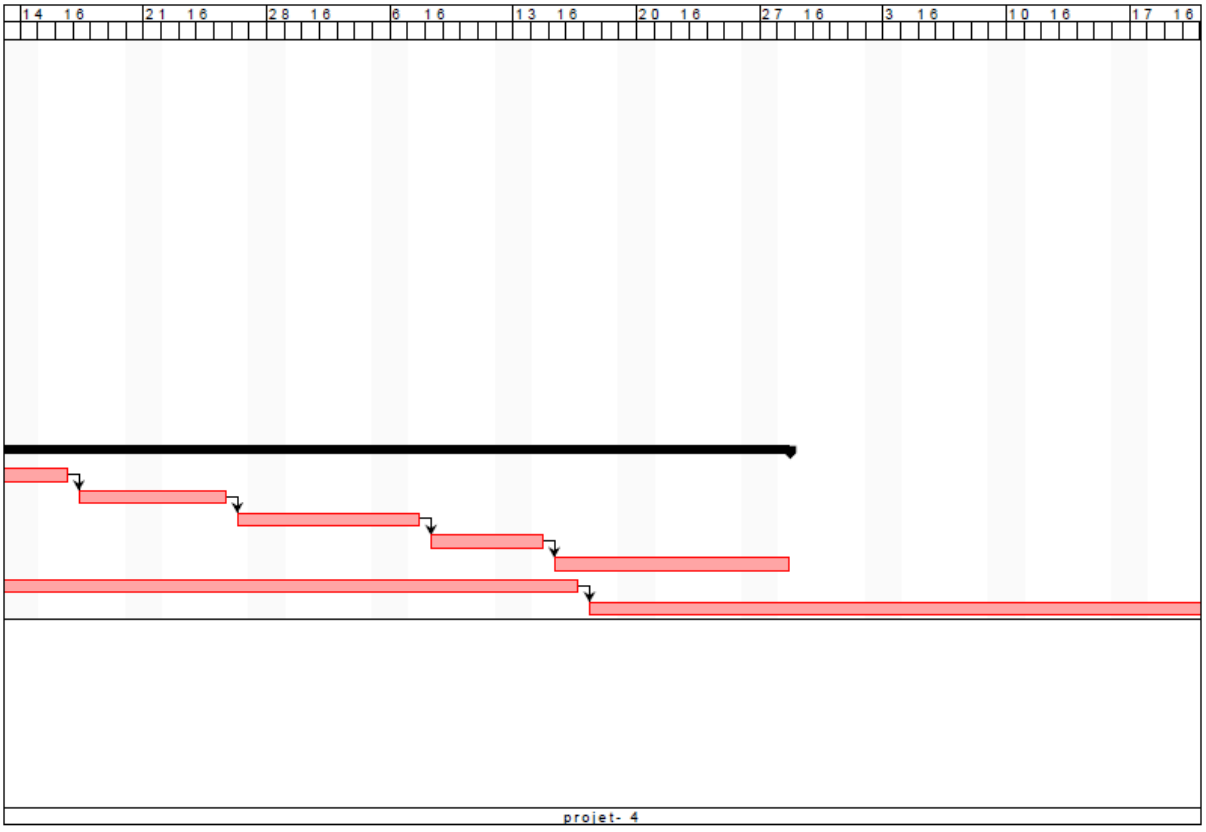
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | Cahier de charge | 2 8 | 15-10-1 8:00 | 15-11-9 5:00 | | | |
| 2 | | Bridge | 7 | 15-11-10 8:00 | 15-11-18 5:00 | 1 | | |
| 3 | | Spécification | 2 | 15-11-10 8:00 | 15-11-11 5:00 | 1 | | |
| 4 | | Implémentation | 2 | 15-11-12 8:00 | 15-11-13 5:00 | 3 | | |
| 5 | | Test Unitaire | 3 | 15-11-16 8:00 | 15-11-18 5:00 | 4 | | |
| 6 | | Bibliographie | 2 8 | 15-11-10 8:00 | 15-12-17 5:00 | 1 | | |
| 7 | | Broker scalable | 1 4 | 15-11-19 8:00 | 15-12-8 5:00 | 5 | | |
| 8 | | Spécification | 5 | 15-11-19 8:00 | 15-11-25 5:00 | | | |
| 9 | | Implémentation | 5 | 15-11-26 8:00 | 15-12-2 5:00 | 8 | | |
| 10 | | Test Unitaire | 4 | 15-12-3 8:00 | 15-12-8 5:00 | 9 | | |
| 11 | | Conception | 4 3 | 15-12-9 8:00 | 16-2-5 5:00 | | | |
| 12 | | Apprentisage de Docker | 7 | 15-12-9 8:00 | 15-12-17 5:00 | 10 | | |
| 13 | | Apprentisage de Consul | 7 | 15-12-18 8:00 | 15-12-28 5:00 | 12 | | |
| 14 | | Apprentisage de Registrator | 7 | 15-12-29 8:00 | 16-1-6 5:00 | 13 | | |
| 15 | | Apprentisage de Microservice | 7 | 16-1-7 8:00 | 16-1-15 5:00 | 14 | | |
| 16 | | Conception Généralle | 7 | 16-1-18 8:00 | 16-1-26 5:00 | 15 | | |
| 17 | | Conception Détaillée | 8 | 16-1-27 8:00 | 16-2-5 5:00 | 16 | | |
| 18 | | LL3 | 3 6 | 15-12-18 8:00 | 16-2-5 5:00 | 6 | | |
| 19 | | Implémentation | 3 6 | 16-2-8 8:00 | 16-3-28 5:00 | | | |
| 20 | | Implémentation de Serveur | 7 | 16-2-8 8:00 | 16-2-16 5:00 | 16 | | |
| 21 | | Implémentation de Loadbal... | 7 | 16-2-17 8:00 | 16-2-25 5:00 | 20 | | |
| 22 | | Test de Serveur | 7 | 16-2-26 8:00 | 16-3-7 5:00 | 21 | | |
| 23 | | Test de Loadbalancer | 5 | 16-3-8 8:00 | 16-3-14 5:00 | 22 | | |
| 24 | | Test de recette | 1 0 | 16-3-15 8:00 | 16-3-28 5:00 | 23 | | |
| 25 | | LL4 | 2 8 ? | 16-2-8 8:00 | 16-3-16 5:00 | 18 | | |
| 26 | | LL5 | 2 8 | 16-3-17 8:00 | 16-4-25 5:00 | 25 | | |

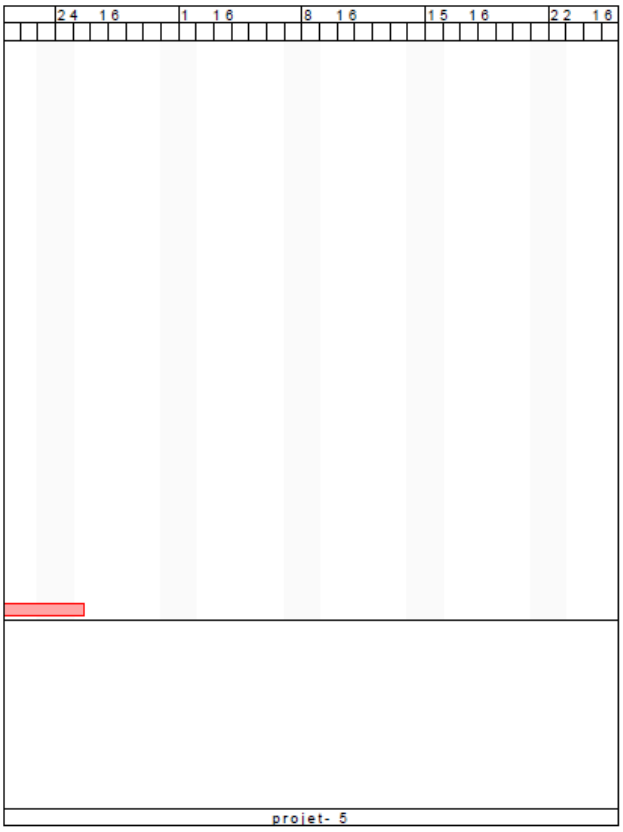
projet- 1




projet- 2


1 5
1 3 1 5
2 0 1 5
2 7 1 5
3 1 6
1 0 1 6
1 7 1 6
2 4 1 6
3 1 1 6
7 1 6
projet- 3


1 4 1 6
2 1 1 6
2 8 1 6
6 1 6
1 3 1 6
2 0 1 6
2 7 1 6
3 1 6
1 0 1 6
1 7 1 6
projet- 4


24 16
1 16
8 16
15 16
22 16
projet- 5

## 8.4 Estimation de l'effort

Ensuite, nous avons estimé l'effort de notre projet, au-dessous c'est le résultat :

PREC : 2.48 (notion : High, raison : Nous avons bien compris le but de ce produit mais manque des experience par rapport à ce technologie.)
FLEX : 5.07 (notion : Very Low, raison : Notre projet est nécessairement adapté à le protocole MQTT, donc nous avons des contraintes.)
RESL : 4.24 (notion : Nominal, raison : Nous ferons attention à la gestion de risqué)
TEAM : 1.10 (notion : Very High, raison : Nous avons une très bonne équipe.)
PMAT : 4.68 (notion : Nominal, raison : Le niveau de maturité de notre produit est moyen.)
**Somme : 17.57**
*Estimez les facteurs linéaires d'un projet PTRANS :*
RELY : 1.10 (notion : High, raison : C'est un projet par rapport à les données personnelles du client.)
*DATA :* 1.00 (notion : Nominal)
*RUSE :* 0.95 (notion : Low)
*DOCU :* 1.00 (notion : Nominal, raison : Le documentation est démandé à rédiger.)
*CPLX :* 1.17 (notion : High, raison :C'est un projet par rapport à le modification de Kafka.)
*TIME :* 1.11 (notion : High)
*STOR :* 1.05 (notion : High, raison :Le Kafka enrégistre les données de façon du fichier.)
*PVOL :* n/a (notion : Very Low, raison : Nous prévoyons aucune changement de l'environement d'exécution du programme.)
*ACAP :* 1.00 (notion : Nominal)
*PCAP :* 0.88 (notion : High, raison : Nous avons des très bons programmateurs.)
*PCON :* 0.90(notion : High)
APEX : 1.00 (notion : Nominal, raison : Nous avons assez des expériences pour faire un tel produit.)
*PLEX :* 1.00 (notion : Nominal)
*LTEX :* 1.00 (notion : Nominal)
*TOOL :* 1.00 (notion : Nominal)
*SITE :* 0.93 (notion : High)
*SCED :* 1.00 (notion : Nominal, raison : Nous avons assez du temps.)
**Produit : 1.05**

Nous allons travailler 2 mois à une personne dans le coeur du projet.
*A=2.9, B=0.91*
*Donc*
Taille en KSLOC : 0.679

# 9. Analyse des risques(Mis à jour)

**Risques sur les hommes et les compétences**
L'expression d'oral sera un défi pour moi. Pourtant, je fais mes effort à surmonter cette difficulté.

**Risques sur le planning**
Jusqu'a maintenant, je peux prévoir que notre projet sur la partie de serveur et loadbalancer ne sera pas un problème pour moi. Cela peut être implémenté avant la dernière date.

**Risques sur les technologies**
Nous avons consacré beaucoup de temps à choisir une proposition entre plusieurs. Ensuite, je suis capable à réaliser cette conception. Et puis, je vais utiliser un outil de test à mieux tester notre serveur pour qu'il puisse marcher sans arrêt et sans erreur.

# 10. Test de recette

| | |
|---|---|
| Outil : | Android ou mqtt-malaria |
| Processus : | Depuis android, il va envoyer les métadonnées à notre serveur. |
| Variable : | On va envoyer les métadonnées avec différents dimension pour tester la capacité de notre système. |
| Résultat : | Les métadonnées seront stockées dans notre base de données. |

# 11. Le sprint passé(LU3)

# INFO4 – ANNEE 2015/2016

## Fiche de suivi de projet

Titre du projet : Récupération de flux de données

Nom étudiant : LI GUOBAO

Nom étudiant : ……

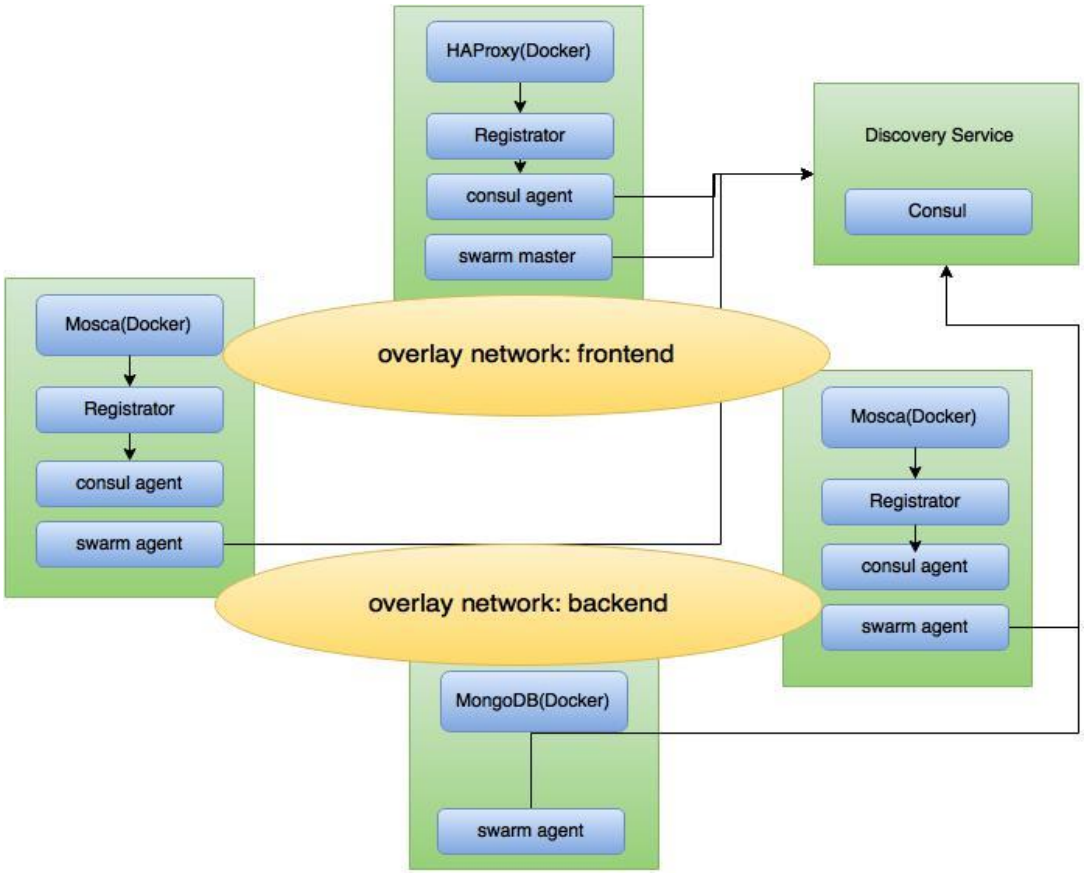
Nom tuteur enseignant : Benoît Parrein

Signature tuteur : ……..

Semaine du 25th Janvier Au 31th Janvier

**TRAVAIL EFFECTUE**

Pendant cette semaine, j'ai travaillé sur la conception de notre projet. Et après avoir comparé les deux possibilité de mettre en place notre serveur, je préfère la proposition qui nous permet d'installer le serveur avec docker swarm, consul(service de découvert), consul agent, registrator et le réseaux overlay. Car en appliquant registrator et consul agent installé sur chaque hôte cela permet àloadbalancer de trouver l'addresse ip et le port de chaque mosca. Je vais montrer l'architecture de cette proposition et aussi la fin de ce sprint.




| | | | | Plan | Réal |
|---|---|---|---|---|---|
| Fini | Le serveur MQTT | 10h | Sous-Total Plannification | 10 | 10 |
| | | | Discussion avec l'entreprise par email | 1 | 0.5 |
| | | | Apprentisage de Mosca | 2 | 2 |
| | | | Codage de Mosca | 2 | 2 |
| | | | Apprentisage du cluster docker swarm | 1 | 0.5 |
| | | | Mis en place du Mosca et Mongodb | 4 | 5 |
| | | | | | |
| | Le loadbalancer | 10h | Sous-Total Plannification | 10 | 3 |
| | | | Apprentisage de HAProxy | 1 | |
| | | | Apprentisage de confd | 1 | |
| | | | Apprentisage de service découvert | 0.5 | |
| | | | Apprentisage de réseaux overlay | 0.5 | 3 |
| | | | Installation de loadbalancer | 5 | |
| | | | | | |
| | LU3 | 10h | Sous-Total Plannification | | 6 |
| | | | Conception de serveur et loadbalancer | 4 | |
| | | | Rédaction de test | 2 | |
| | | | | | |
| | | | | | |
| | Préparation de la soutenance | 4h | Sous-Total Plannification | | 4 |
| | | | Préparation des slides | 2 | |
| | | | Préparation du discours pour la soutenance | 2 | |
| | | | | | |
| | | | | | |
| | Réunion avec le tuteur d'entreprise par téléphone | 0.5h | Sous-Total Plannification | | 0.5 |
| | | | Réunion avec le tuteur d'entreprise par téléphone | 0.5 | |

Deuxième partie

# Bibliographie

# Chapitre 1 L'architecture du projet

## 1 Schéma

Pour l'instant, j'ai conçu ce schéma pour nous faire bien comprendre l'architecture de notre projet, et il est tiré la figure 2.1

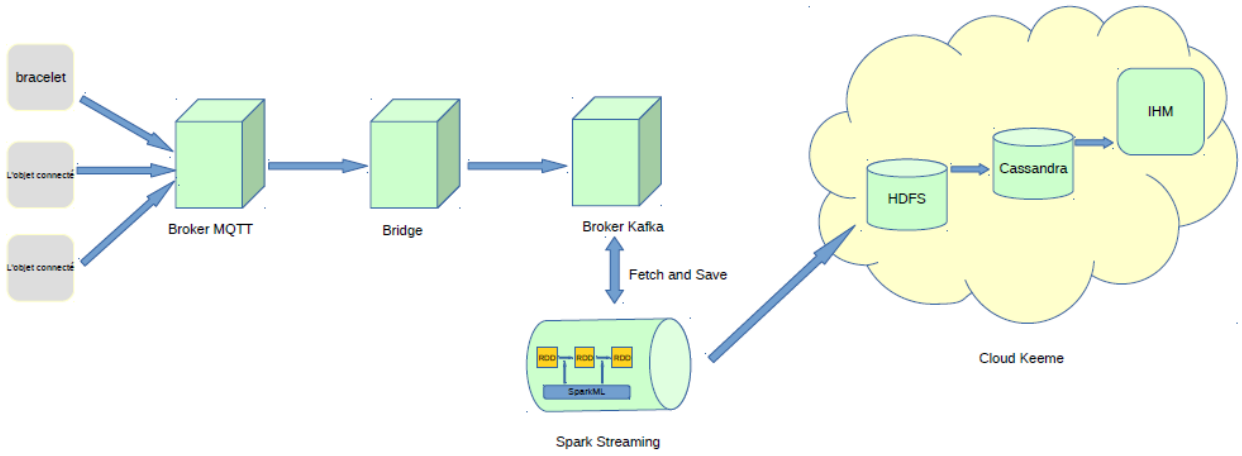



FIGURE 2.1—Architecture

Notre projet consiste quelques composantes :

- Du côté objets, la communication passera par le protocole MQTT en appliquant le Paho. Ainsi il faudra réaliser un connecteur MQTT pour Kafka, qui récupera les messages depuis le broker MQTT et les renvoyera à Kafka.
- Le système à mettre en place consiste en un broker sécurisé et scalable. Ce broker fera le lien entre les objets et le cloud tout en permettant de transmettre et guider les données vers le cloud Keeme. Le broker sera basé sur Apache Kafka: il fonctionnera donc sur le paradigme publisher-subscriber.
- Du côté cloud, la communication se fera via un module Spark Streaming, qui analysera et traitera les données en appliquant SparkML.
- Dans le cloud Keeme, les données seront sauvegardées en permanance dans le système de fichier HDFS.
- Pour la facilité et l'efficace à requêter les données en grande quantité, une base de données Cassandra sera mis en place.
- La partie de IHM, React.js sera appliqué pour l'affichage du résultat.

# 2. Conclusion

Nous allons diviser le projet en quartre partie, et les chaptires suivantes se servent à donner les détailles :

a) le bridge
b) le broker scalable
c) le traitement du flux de données
d) l'enrégistrement et l'affichage du résultat obtenu

# Chapitre 2 Le bridge

## 1 Définition

Pour la première partie, nous allons implémenter un bridge pour faire passer le message en format de MQTT entre le broker MQTT et le broker Kafka, car il n'existe pas un MQTT connecteur pour Kafka, c'est-à-dire que le protocole MQTT n'adapte pas à Kafka. Donc nous allons créer un connecteur MQTT pour Kafka. Pourtant, avant l'implémentation, il nous faut d'apprendre le protocole MQTT, le modèle publisher-subscriber de messagerie appliqué dans le broker MQTT et le broker Kafka. Ensuite, nous allons présenter les technologies necessaires pour cette partie.

### 1.1 MQTT

« MQTT est un protocole simple et extrêmement léger, qui est dédié à les objets limits et les réseaux faibles bande ou incertains. Les principles sont pour minimiser la bande de réseau et la demande de ressource, alors que chercher à assurer la fiabilité et dans le mesure de l'assurance d'envoi. Cette principles font le protocole idéal pour servir à les objets connectés où la bande et la batterie sont considérés en prime. » [2]

Ensuite nous allons présenter les termes techniques dans le protocole MQTT :

**Message** : « Les données portées par le protocole MQTT à travers du réseau. Lorsque une message est transmise par MQTT, elle possede la Qualité de Service et le nom de topic. » [2]

**Client** : « Un programme ou appareil qui utilise MQTT. Un client crée souvent une connection de réseau avec le serveur. Il peut :

- Publier les messages à quelles les autres clients pourraient s'intéresser.
- S'abonner à le topic au quel il s'intéresse.
- Désabonnement à enlever un requêst pour un message.
  Déconnecter à le serveur » [2]

**Serveur** : « Un programme ou appareil qui est en tant que un intermédiaire entre le client qui publie les messages et le client qui a faire des abonnements. Un serveur :

- Accepte la connection de réseau à partir des clients.
- Accepte les messages publiées par des clients.
- Traite les requêsts d'abonnement et de désabonnement à partir des clients.
- Passe les messages qui correspondent à les abonnements de client. » [2]

**Abonnement** : « Un abonnement consititue un filtrage de topic et un Qos. Il est associé à une session. Une session peut contenir plus d'un abonnement. Chaque abonnement dans une session possède un filtrage de topic différent. » [2]

**Le nom de topic** : « Une étiquette attachée à une message. Le serveur envoie une copie de message dont l'étiquette à laquelle le client abonne. » [2]

**Filtrage de topic** : «Une expression contenue dans un abonnement, à indiquer une préférence à un ou plus topic.» [2]

**Session** : «Une interaction avec l'état entre un client et un serveur. Certaines sessions durent seulement aussi longuement que la connection de réseau, l'autres peuvent traverse plusieurs connections de réseau consécutives entre un client et un serveur.» [2]

**Le contrôle paquet** : «Un paquet d'information qui est envoyé à travers de la connection de réseau. La structure est suivante :

- La tête fixe indique le type de ce paquet et dont la taille.
- La tête variée apparait dans certaines situations indiquant l'identifiant de paquet.
- La charge apparait dans certaines situations indiquant le message porté» [2]

**Qualité de service** : «Elle définit combine de l'efforts le serveur ou le client essaie à assurer que un message est reçu.

- Qos0 : Le serveur ou le client délivrera le message une fois, sans acquittement.
- Qos1 : Le serveur ou le client délivrera le message au moins une fois, avec acquittement.
- Qos2 : Le serveur ou le client délivrera le message exactement une fois en appliquant une poignée de main en quartre fois.» [2]

## 1.2 Paho

«Le projet Paho fournit un client implementé par le protocole MQTT pour le Iot.» [3] Le client est implémenté sur plusieurs plate-formes, par exemple, le Java et Android etc.

En appliquant Paho, ça nous permet d'envoyer un message comme un produteur ou de recevoir un message comme un consommateur.

## 1.3 Broker MQTT

Ensuite, à propos du choix d'un broker MQTT, nous avons plusieurs propositions: ActiveMQ, Apollo, ZeroMQ, Mosquitto, RabbitMQ. Donc nous avons besoin de chercher les références à comparer les avantages et les inconvénients entre ces brokers. La description à venir est dédiée à le test et le résultat.

- «RabbitMQ est un des l'implémentation de AMQP protocole plus utilisé. Donc, il implémente une architecture de broker, c'est-à-dire que les messages sont mis dans un noeud central avant avoir été envoyé à les clients. Il permet d'être appliqué et mis en place facilement, grâce à le routeur, l'équilibrage de charge ou le message queuing, NACK sont soutenus dans quelques lignes de code. Pourtant, il le rend moin scalable et lent, car le noeud central ajoute la latence.» [4]

- «ZeroMQ est un système messagerie assez légère dédié à les scénarios haut débit/faible latence. Il soutient plusieurs scénario messagerie avancé mais par rapport à ActiveMQ et RabbitMQ, nous devons implémenter la plupart nous-même par la combinaison des pièces de cadre.» [4]

- «ActiveMQ est entre RabbitMQ et ZeroMQ. Comme ZeroMQ, il peut être mis en place avec le broker et P2P topologies. Comme RabbitMQ, il est plus facile à implémenter les scénarios avancées mais souvent au prix de la performance brute. »[4]

- «ActiveMQ Apollo est un broker messagerie plus rapid, plus fiable, plus facile à maintenir le broker messagerie, qui est créée par la foundation de ActiveMQ. Il l'accomplit en appliquant un threading différent et une architecture de l'envoie messagerie. »[4]

| | ActiveMQ / Apollo | RabbitMQ | ZeroMQ |
|---|---|---|---|
| **Brokerless/ Decentralized** | No | No | Yes |
| **Clients** | C,C++, Java, Others | C,C++, Java, Others | C,C++, Jav Others |
| **Transaction** | Yes | Yes | No |
| **Persistence/ Reliability** | Yes (configurable) | Yes (built-in) | No persistenc requiring higher laye to manage persistenc |
| **Routing** | Yes (easier to implment) | Yes (easier to implment) | Yes (compl to implmen |
| **Failover/ HA** | Yes | Yes | No |
| **Unlimited Queue** | Yes | Yes | Yes |
| **Scalabilty** | Yes | Yes | Yes |
| **Users** | FuseSource, CSC, GatherPlace, UW Tech, Enterprise Carshare | Mozilla , AT&T, UIDAI | |
| **Licence/ Community** | Apache (openSource) | Spring Source.Licen ced under Mozilla Public License | IMatix . Licensed General Public License |

FIGURE 2.2—Comparaison

Selon la figure du résumé, nous pourrons tirer quelques conclustions:

- «ActiveMQ ou Apollo est un choix pertinent quand cela vient à l'aisance de configuration au prix de la performance dans le mode de persistance.
- RabbitMQ est plus pertinent pour le messagerie avancé avec le routage et l'équilibrage de charge.
- ZeroMQ est plus pertinent quand cela vient à un besoin du broker compliqué »[4]

## 1.4 Kafka

Kafka est un log service distribué, partitionné, scalable et fiable.[5] Il fournit la fonctionalité d'un système de messagerie.

Dans notre projet, en appliquant Kafka comme un messagerie, on pourra fournir un stockage temporel scalable et fiable pour l'analyse du flux de données par SparkStreaming.

Grosso modo, le producer publiera les messages étiquetés par le topic vers le broker Kafka, ensuite le broker acceptera les abonnements à certains topics à partir des consommateurs qui recevoiront les messages intéressés.

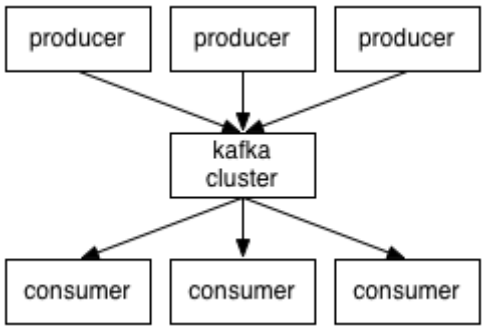



FIGURE 2.3—Mode de messagerie

### 1.4.1 Topic

Nous avançons vers le topic en détail. Pour chaque topic, le cluster Kafka maintenira un log partitionné comme suivante :

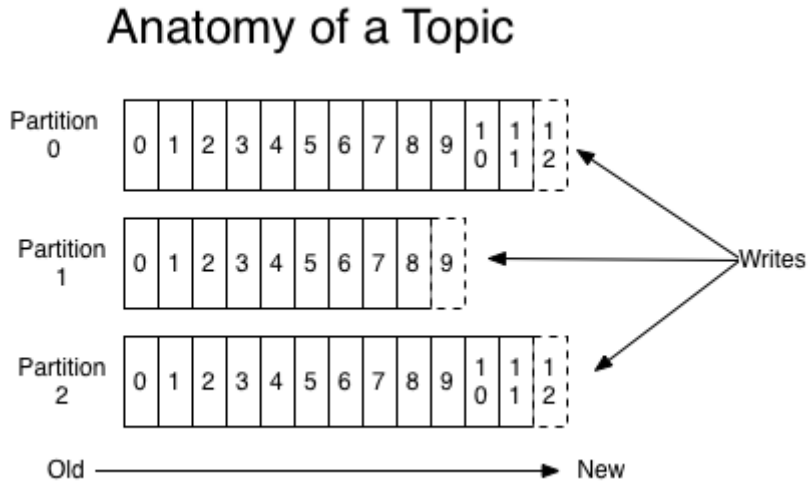



FIGURE 2.4—Topic

« Chaque partition est une séquence ordonnée et immuable des messages qui est ajoutée sans cesse à un log commit. Les messages dans les partitions sont chaque donnés un id numéro en séquence qui s'appelle l'offset qui identifie uniquement chaque message dans la partition. »[5]

« Le cluster Kafka retient tous les messages publiés, qu'elle que soit ils ont été consumés, pour une période configurable. Par exemple, si la rétention de log est fixé à deux jours, alors depuis deux jours après la publication d'un message, il est disponible pour la consommation, et après il va être jeté à libérer l'espace. La performance de Kafka est stable concernant la taille de données, donc la rétention de beaucoup de données ne pose pas de problème. »[5]

«En fait, la seule métadonnée retenue dans chaque consommateur est la position de consommateur dans le log, qui s'appelle 'offset'. Cet offset est contrôlé par le consommateur : en général, un consommateur fera avancer son offset en linéaire lorsqu'il lit les messages, mais en fait, la position est contrôlé par le consommateur et il peut consommer les messages par l'ordre qu'il veut. Par exemple, un consommateur peut remettre le offset en avant pour le retraiter. »[5]

«Les partitions dans le log ont quelques buts. D'abord, ça permet le log à agrandir au-del à une taille qui adaptera à un serveur seul. Chaque partition doit s'adapter à les serveur, mais un topic peut avoir plusieurs partitions, donc ça permet à traiter les données en grande taille. Ensuite, elles servent en tant que la unité de parallélisme. »[5]

### 1.4.2 Distribution

«Les partitions de log sont distribuées sur les serveurs dans le cluster Kafka avec chaque serveur traitant les données et les requêsts pour une partage de partitions. Chaque partition est duplicate à travers des serveurs pour la tolérance d'erreur. »[5]

«Chaque partition possède un serveur qui sert du leader et rien ou plus serveurs qui sert des suiveurs. Le leader traite tous les requêsts de la lecture et l'écriture pour la partition alors que les suiveurs donnent les duplicates de données. Si un échec arrive à le leader, un des suiveurs devenira automatiquement le nouvel leader. Chaque serveur sert en tant qu'un leader pour certaines partitions et ainsi qu'un suiveur pour l'autres, donc le charge est bien équilibré dans le cluster. »[5]

### 1.4.3 Producteurs

«Les producteurs publient les données aux topics de leur choix. Le produteur est chargé de choisir quel message est réparti à quelle partition dans le topic. Ça peut être fait dans un round-robin simplement à équilibrer le charge ou ça peut être fait selon quelques fonctions sémantique de partition. »[5]

### 1.4.4 Consommateurs

«En appliquant la modèle de publisher-subscriber, Kafka fournit une abtraction d'un seul consommateur qui généralise les deux, et c'est la groupe de consommateur. »[5]

«Les consommateurs étiquetés par un nom de groupe consommateur, et chaque message publié à un topic sera délivré à une instance de consommateur dans chaque groupe de consommateur. »[5]

«Kafka est capable à assurer l'ordre dans une partition de topic et à fournir une équilibrage de charge. D'une côté, une partition se limitera à être accès par une instance de consommateur dans chaque groupe de consommateur, cela permet d'assurer l'ordre des messages reçus. D'une autre côté, en même temps, les partitions pourront équilibrer le charge sur les consommateurs connectés. »[5]

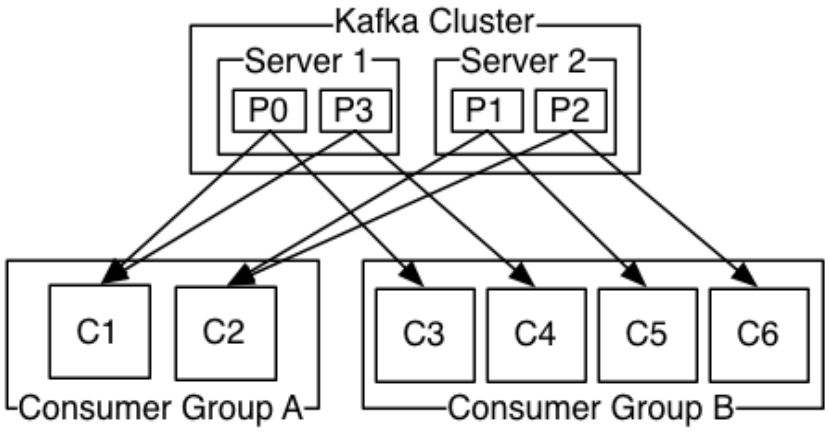



FIGURE 2.5—Groupe de consommateurs

## 2. Architecture

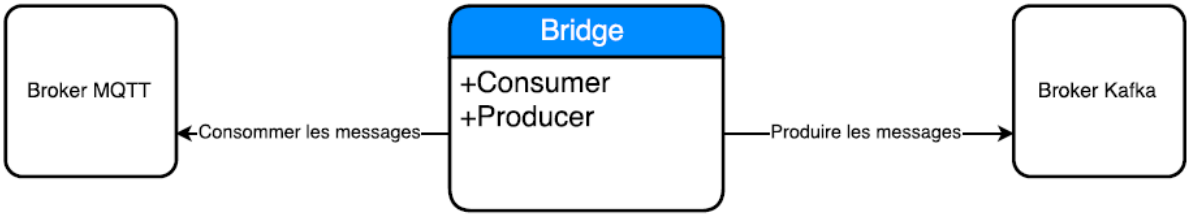



FIGURE 2.6—Architecture

# Chapitre 3 Le broker scalable

## 1 Définition

Dans un environement réel de IOT(Internet Of Things), la disponibilité et la scalabilité sont des enjeus auquels on doit prendre beaucoup attention. Car pour l'objet connecté, la connectivité avec le serveur devra être stable pour que ils puissent envoyer ou recevoir les messages à partir du serveur. Donc notre but sera installer un broker MQTT scalable dans quelque mesure. Nous installerons deux brokers MQTT pour la service, cela permet de équilibrer la charge des requêtes et continuer à fournir la service dans le cas où un des brokers tombera en panne.[6]

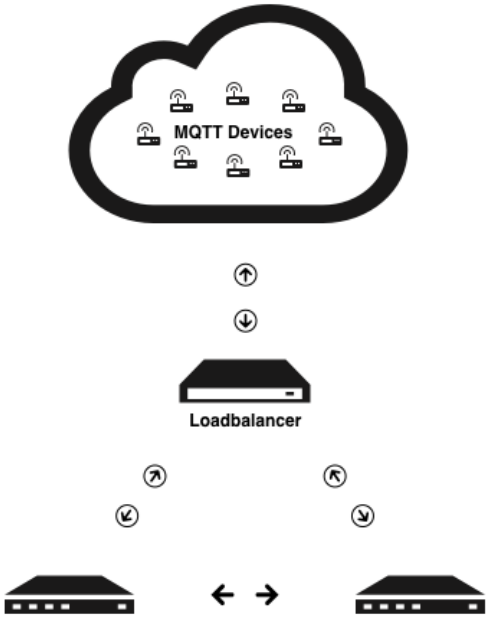



FIGURE 2.7—Architecture

## 2 Plan

Ce plan nous fournira une possibilité à déployer un broker MQTT scalable. Ensuite, nous allons diviser l'installation à cinq étapes.[6]

1. Installer un serveur MQTT
2. Dockerize le serveur MQTT
3. Ajouter HAProxy en tant qu'un équilibreur de charge
4. Faire MQTT en sécurité avec SSL
5. Configurer nscale à automatiser le déploiement de flux de travail

### 2.1 Installer un serveur MQTT

Selon la comparaison des brokers MQTT dans le chapitre 2, il y a quelques possibilités tel que RabbitMQ, ZeroMQ, ActivMQ et Apollo. A mon sens, nous pourrons choisir un broker scalable mais pas compliqué à le configurer, de façon à fournir une service plus fiable et plus efficace. Donc

nous allons choisir Apollo comme le broker MQTT, car il est scalable, rapid et fiable utilisant une architecture de message dispatching.

Au départ, nous configurerons le réglage de broker en appliquant Redis, cela permet de créer une communication entre le broker et les autres microservices. Pourtant, le microservice n'est pas encore mis en place dans notre projet. Et puis, dans le but de la sécurité au broker, nous appliquerons le mécanisme fournit par Apollo à authentifier les utilisateurs losqu'ils envoyeront une requête à demander l'accès au broker.[6]

## 2.2 Dockerize le serveur MQTT

Pour l'instant, la technologie Docker est en plein essor, cela permet de déployer les systèmes de production et d'exécuter le code dans tous les machines sans savoir l'environement de la machine.[6]

Au début, nous créerons un fichier qui s'appellera Dockerfile, cela permet d'initializer l'environment désiré.

```
#obtenir une image existant qui constitue Ubuntu, Java à partir de Docker Hub
FROM ....

#installer Apollo dans cette image obtenu
RUN ...

#exposer le port de Apollo
EXPOSE ...

#exécuter le script .sh
ENTRYPOINT [...]
```

FIGURE 2.8—Dockerfile

A la fin, ce que nous aurons besoin de faire, c'est à exécuter cette image, et notre broker sera mis en place.

## 2.3 Ajouter HAProxy en tant qu'un équilibreur de charge

Cela vient de HAProxy, il est un équilibreur de charge à la base de TCP/HTTP ainsi qu'une solution de proxy destiné à améliorer la performance et la fiabilité de l'environment de serveur, repartant le charge de travail au plusieurs serveurs. En plus, il est implémenté en langage C et caractérisé par être rapid et efficace. [6]

Au début, nous téléchargerons le conteneur HAProxy à partir de Docker Hub, et cela permettra de déployer automatiquement HAProxy. Et puis, nous configurerons HAProxy, que HAProxy entendra à tous les requêtes vers le port 1883, les avançant vers deux ou plus serveurs MQTT en appliquant leastconn(choisir le serveur qui aura le moin requête), une façon équilibre. [6]

```
# Listen to all MQTT requests (port 1883)
listen mqtt
  # MQTT binding to port 1883
  bind *:1883
  # communication mode (MQTT works on top of TCP)
  mode tcp
  option tcplog
  # balance mode (to choose which MQTT server to use)
  balance leastconn
  # MQTT server 1
  server apollo_1 check
  # MQTT server 2
  server apollo_2 check
```

FIGURE 2.9—Configuration

## 2.4 Faire MQTT en sécurité avec SSL

SSL est un standard accepté pour la communication en sécurité entre un serveur et un client assurant que tous les données transférées entre eux maintiennent en secret et en intégrité. Plus en détail, dans le but de l'implémentation de SSL avec HAProxy, le certificat et la paire de clé devront sous la forme de PEM. Donc, au début, nous combinerons simplement notre certificat SSL(fournit par les autorités de certificat) avec notre clé privée(générée par nous). [6]

```
cat edgar.crt edgar.key > edgar.pem
```

FIGURE 2.10—Génération de .pem

Et ensuite, nous chargerons le fichier .pem en appliquant Docker volumes, cela permettra de partager le certificat uniquement dans le serveur HAProxy mais pas en public. Car une fois que le certificat SSL sera généré, il devra être en secret. A la fin, une fois que le certificat SSL sera mis en disponible dans le Docker volume, ce que nous devrons faire, c'est que nous ferons passer les requêtes arrivé au port 8883(un port acceptant le requête du type SSL) au certificat SSL.[6]

```
# Listen to all MQTT requests
listen mqtt
  # MQTT binding to port 1883
  bind *:1883
  # MQTT binding to port 8883
  bind *:8883 ssl crt /certs/edgar.pem
  ...
```

FIGURE 2.11—SSL requête

## 2.5 Configurer nscale à automatiser le déployement de flux de travail

Nous allons appliquer nscale à configurer, créer et déployer une suite de conteneurs connectés. [6]

# Références

[1] Keeme. Keeme Vos Données – Vos Règles. Disponible sur : http://www.keeme.io/?lang=fr

[2] MQTT Version 3.1.1. Rédigé par Andrew Banks et Rahul Gupta. 29 Octobre 2014. Disponible sur: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[3] Apache Paho. Disponible sur : http://www.eclipse.org/paho/

[4] Kuntal Ganguly. ActiveMQ vs RabbitMQ vs ZeroMQ vs Apache Qpid vs Kafka vs IronMQ - Message Queue Comparision. 03/08/14. Disponible sur : http://www.kuntalganguly.com/2014/08/message-queue-comparision.html

[5] Kafka 0.8.2 Documentation. Disponible sur : http://kafka.apache.org/documentation.html#quickstart

[6] Lelylan Blog. How to build an High Availability MQTT Cluster for the Internet of Things. Disponible sur : https://medium.com/@lelylan/how-to-build-an-high-availability-mqtt-cluster-for-the-internet-of-things-8011a06bd000#.7uc1b57x1

Troisième partie

# Conception Générale

# 1 Introduction

Le système consiste à fournir une service stable et efficace dans certain niveau. Il est constitué par deux sous-systèmes qui sont le serveur et le loadbalancer.

# 2 Présentation du matériel et du logiciel de base

- MongoDB : une base de données de clé valeur fournissant la service de « publish subscribe ».
- Mosca : un serveur basé sur nodejs et dédié à recevoir et envoyer les messages de MQTT vers la base de données.
- HAProxy : un serveur dédié à repartir le charge de requête.
- Consul :une base de données fournissant la service de découvert.
- Docker :une service fournissant la technologie de conteneurisation.
- Swarm :un outil dédié à la gestion et surveillance de conteneurs.
- Registrator :un outil à faire enresgistre les informations de conteneur.

# 3 Architecture générale du système

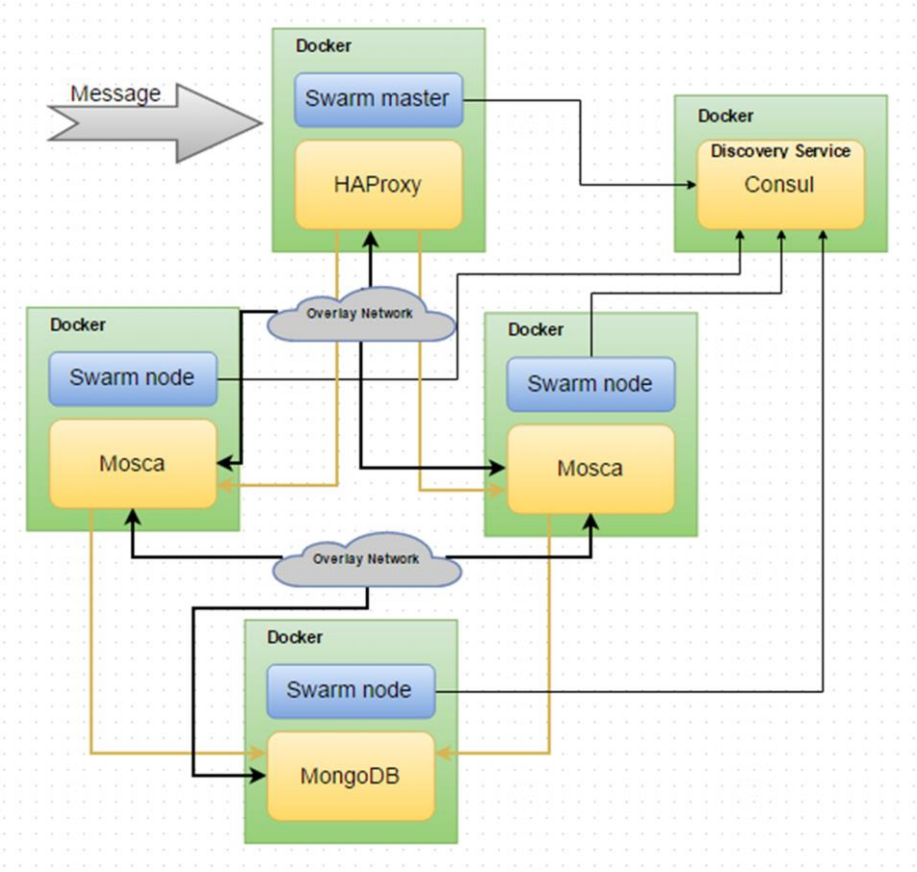



*Figure 3.1 – l'architecture*

Selon la fonctionalité, le système peut être divisé à deux sous-systèmes :

- La partie du serveur MQTT constitué par Mosca, MongoDB, docker, swarm, consul et registrator.
- La partie du loadbalancer constitué par HAProxy, confd , docker, swarm, consul et registrator.

# 4 Organisation de sous-système : le serveur

## 4.1 Présentation de la struture du sous-système

Le serveur Mosca est dédié à fournir le service de faire passer les données sous la forme de MQTT à partir des objets connectés. Le protocole MQTT est un protocole caractérisé par sa légérité. Ensuite, Mosca sert comme un connecteur qui accepte les messages sous la forme de MQTT, alors que MongoDB fournit le service du modèle « publish subsribe ». Et puis, en appliquant l'architecture de microservice, Mosca et MongoDB sont mis en place dans les conteneurs différents, c'est-à-dire multi-host. Dans ce cas là, un cluster des conteneurs est mis en place en utilisant Docker Swarm. Et Consul fournit la service de découvert pour le cluster. Dans le but de l'isolation entre les conteneurs, un réseaux overlay est installé entre Mosca et MongoDB. A la fin, le registrator est en charge de surveiller les événements des conteneurs (c'est-à-dire la création et l'arrêt des conteneurs) et de les enrégistrer sur Consul pour que les autres conteneurs peuvent communiquer avec eux. Et l'agent de consul est en charge d'envoyer l'information d'enrégistrement au serveur de Consul.

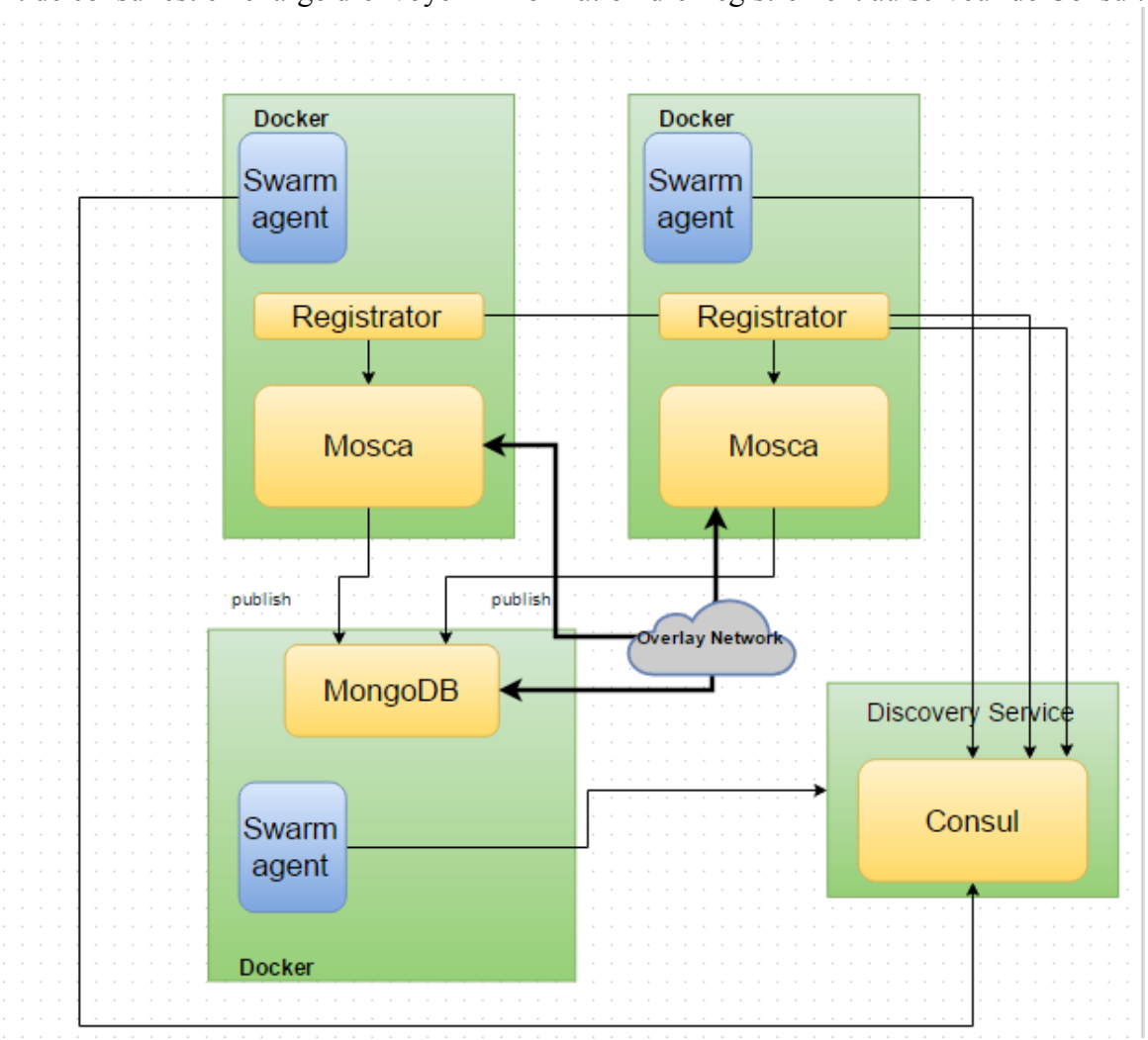



*Figure 3.2—l'architecture de serveur*

### 4.2 Tâches du sous-système

- Le cluster multi-host de docker est installé en appliquant Docker Machine, Docker Swarm et Consul.
- MongoDB est lancé sous la forme de conteneur et cela peut marcher sans tombe en panne.
- Mosca est installé sous la forme de conteneur et cela doit se connecter avec MongoDB.
- Le réseaux overlay est mis en place entre Mosca et MongoDB.
- Le Registrator et Consul agent sont mis en place dans le même hôte que Mosca.

# 5 Organisation de sous-système : le loadbalancer

### 5.1 Présentation de la struture du sous-système

Dans le cadre de notre projet, nous avons plusieurs serveurs Mosca mis en place pour le but de fournir une service plus efficace et stable. Donc nous devons appliquer un loadbalancer à répartir les requêtes à certain serveur. De plus le loadbalancer HAProxy utilisera confd à générer la ficher de la configuration automatiquement. Tous les composants sont mis en place sous la forme de conteneur dans les hôtes différents. En appliquant Registrator et Consul agent, l'information des conteneurs sont capable à être stockée dans Consul de manière à ce que les autres conteneurs puissent communiquer avec eux.

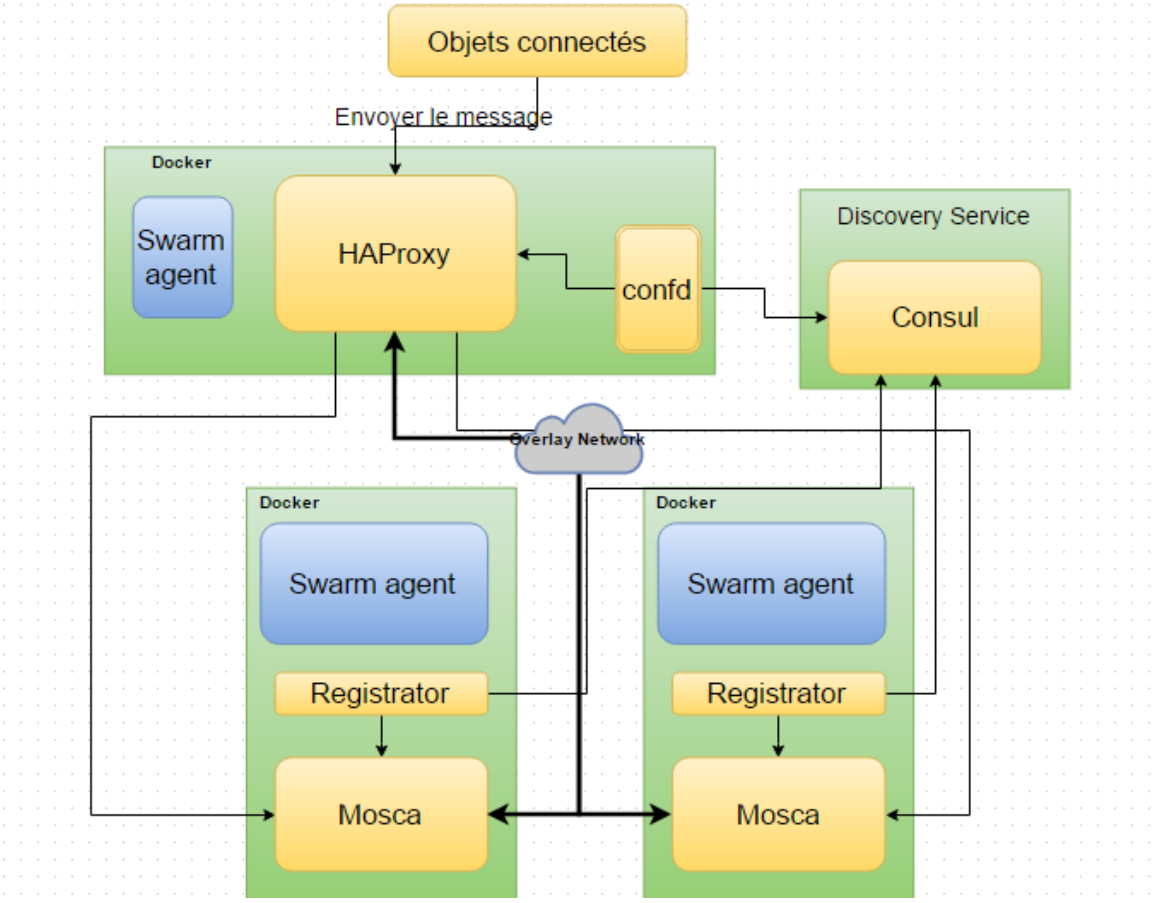



*Figure 3.3—l'architecture de loadbalancer*

### 5.2 Tâches du sous-système

- Tous les components sont lancés dans les conteneurs Docker sur les hôtes différents.

- La technologie confd est mis en place dans le même conteneur que HAProxy pour qu'elle puisse consulter les addresses ip et les ports de tous les serveurs Mosca et qu'elle puisse générer dynamiquement la configuration pour HAProxy.
- Le réseaux overlay est mis en place entre Mosca et HAProxy.
- Le Registrator et Consul agent sont mis en place dans chaque hôte.

# 6 Test d'intégration

En appliquant un outil de test qui peut envoyer les messages sous la forme de MQTT au serveur, le test sera capable d'être mis en place. Je vais concevoir quelques situations à démarrer le test.

a) Envoyer les messages à HAProxy quelques messages(le chiffre est variée) chaque secondes, et puis on verifiera les données dans la base de données mongodb.
b) Envoyer les messages à HAProxy mosca cinq messages dont la taille est variée chaque secondes, et puis on verifiera les données dans la base de données mongodb.

Quatrième partie

# Conception Détaillée

# 1 Introduction

Dans cette partie, nous allons préciser chaque sous-système le processus de traitement et la façon à réaliser ce sous-système.

# 2 Organisation de sous-système : le serveur

## 2.1 Processus de la création

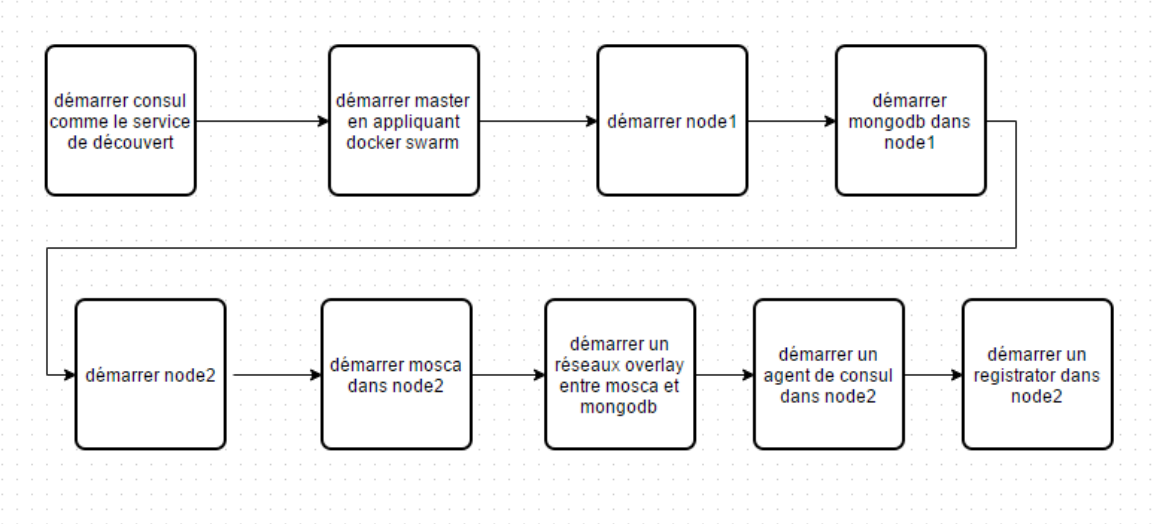



*Figure 4.1—Processus de la création*

## 2.2 Conception de mosca

Sur le module mosca, je vais utiliser une stratégie à répartir le charge dans un hôte. Je vais appliquer mongodb comme la base de données. En gros, je vais montrer le code au-dessous :

```
var mosca = require('mosca');
var cluster = require('cluster');
var numCPUs = require('os').cpus().length; //obtenir le nombre de CPU

var ascoltatore = {
  //using ascoltatore
  type: 'mongo',
  url: 'mongodb://' + process.env.MONGODB_PORT_27017_TCP_ADDR + ':27017/mqtt',
  pubsubCollection: 'ascoltatori',
  mongo: {}
};

var moscaSettings = {
  port: 1883,
  backend: ascoltatore,
  persistence: {
```

```
    factory: mosca.persistence.Mongo,
//    url: 'mongodb://localhost:27017/mqtt'
  url: 'mongodb://' + process.env.MONGODB_PORT_27017_TCP_ADDR + ':27017/mqtt',
  }
};

if (cluster.isMaster) {
   // Fork workers.
   for (var i = 0; i < numCPUs; i++) {
      cluster.fork();
   }


   cluster.on('death', function(work) {
     console.log('worker ' + worker.pid + ' died');
   });
} else {
   // Worker processes have a mosca server.

   var server = new mosca.Server(moscaSettings);
   server.on('ready', setup);

   server.on('clientConnected', function(client) {
       console.log('client connected', client.id);
   });

   // fired when a message is received
   server.on('published', function(packet, client) {
     console.log('process ' + process.pid + ' received message');
     console.log('Published', packet.payload);
   });
}

// fired when the mqtt server is ready
function setup() {
  console.log('Mosca server is up and running')
}
```

### 2.3 Test de mosca

En appliquant un outil de test qui peut envoyer les messages sous la forme de MQTT au serveur, le test sera capable d'être mis en place. Je vais concevoir quelques situations à démarrer le test.

c) Envoyer les messages à un serveur mosca quelques messages(le chiffre est varié) chaque secondes, et puis on verifiera les données dans la base de données mongodb.

d) Envoyer les messages à un serveur mosca cinq messages dont la taille est varié chaque secondes, et puis on verifiera les données dans la base de données mongodb.

# 3 Organisation de sous-système : le loadbalancer

## 3.1 Processus de la création

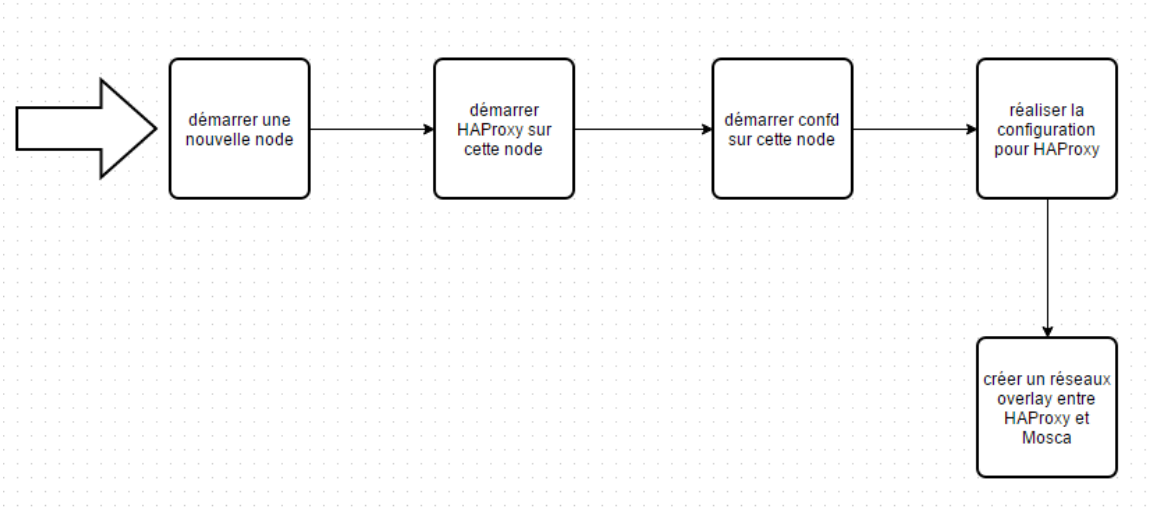



*Figure 4.2—Processus de la création*

## 3.2 Conception de HAProxy et confd

La configuration de HAProxy sera fait en appliquant un outil confd. Cet outil nous permet de générer la configuration selon une fichier de template. Je vais montrer cette fichier au-dessous :

a) Haproxy.toml

```
[template]
src = "haproxy.cfg.tmpl"
dest = "/etc/haproxy/haproxy.cfg"
keys = [
  "mosca",
]
reload_cmd = "/etc/init.d/haproxy reload"
```

b) Haproxy.toml

```
global
#        log 127.0.0.1 local3
        maxconn 5000
#        uid 99
#        gid 99

defaults
#        log 127.0.0.1 local3
        mode tcp
#        option dontlognull
        retries 3
#        option redispatch
        maxconn 2000
```

```
        timeout connect 5000ms
        timeout client 50000ms
        timeout server 50000ms

listen frontend 0.0.0.0:18888
         mode tcp
         balance roundrobin
         maxconn 2000
#         option forwardfor
         {{range gets "/mosca/*"}}
         server {{base .Key}} {{.Value}}
         {{end}}

backend stats_auth
         stats enable
#         stats uri /admin-status
         stats auth admin:123456
         stats admin if TRUE
```

### 3.3 Test de HAProxy

En appliquant un outil de test qui peut envoyer les messages sous la forme de MQTT au serveur, le test sera capable d'être mis en place. Je vais concevoir quelques situations à démarrer le test.

a) Envoyer les messages à HAProxy quelques messages(le chiffre est varié) chaque secondes, et puis on verifiera les données dans la base de données mongodb. En plus, on surveillera le charge de CPU de chaque conteneur mosca.

b) Envoyer les messages à HAProxy cinq messages dont la taille est variée chaque secondes, et puis on verifiera les données dans la base de données mongodb. En plus, on surveillera le charge de CPU de chaque conteneur mosca.

Cinquième partie

# Test

# 1 Introduction

Dans cette partie, on va introduire la partie de test sur notre projet.

# 2 Test unitaire

Dans notre système, on peut surveiller l'état de chaque conteneur et chaque noeur en appliquant Docker Swarm. Dans ce cas là, on peut être au courant si les nœuds et les conteneurs sont bien lancés ou tombés en panne.

# 3 Test de recette

## 3.1 Les outils

Dans notre système, on va lancer le test en appliquant les outils mqtt-malaria, mqtt-spy et sematext. Mqtt-spy est un outil qui nous permet d'envoyer les messages vers notre système en configurant les paramètres comme le nombre des messages, la taille du message, le nombre des processus et etc. Sematext est un outil qui permet de surveiller les conteneurs en montrant l'état réel des conteneurs.

## 3.2 Résultat

- Cas de test :

On va envoyer 10000 messages dont la taille est 100bytes vers notre système en utilisant 10 processus. Et on va lancer 2 serveurs de mosca.

- Résultat :

Dans ce cas là, les deux serveurs de mosca ont bien réparti le charge de requête depuis malaria grâce à Haproxy. Enfin, les messages sont bien stockés dans mongodb.

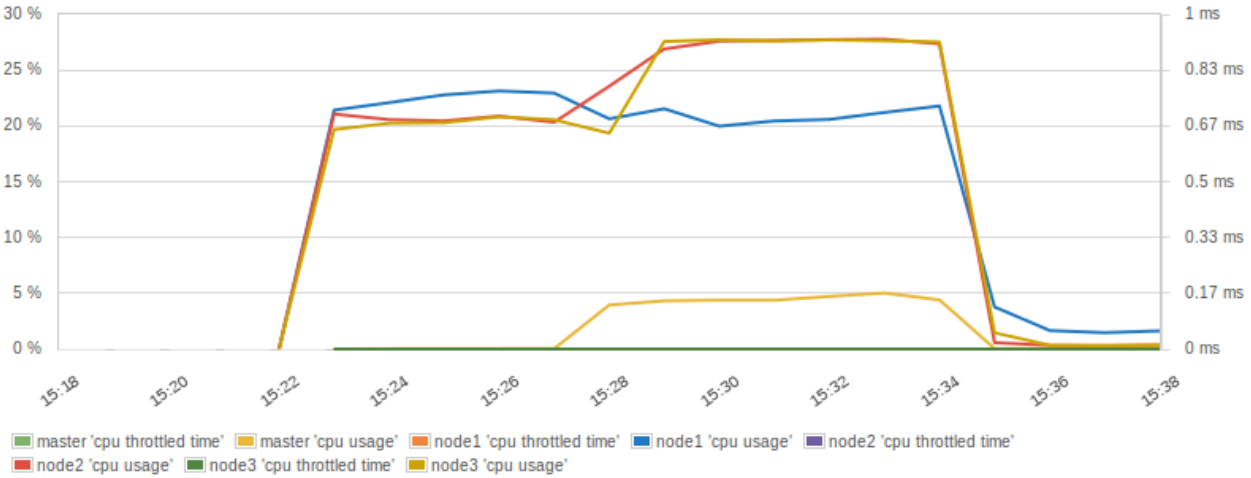



*Figure 5.1 CPU*

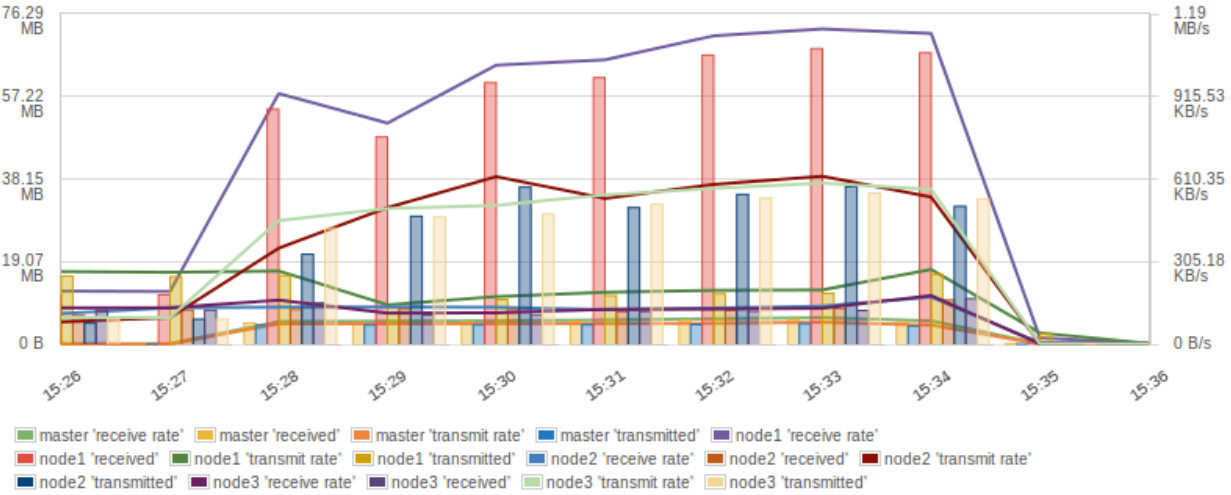



*Figure 5.2 Mémoire*

- Cas de test :

On va envoyer 10000 messages dont la taille est 100 bytes vers notre système en utilisant 10 processus. Et on va lancer 2 serveurs de mosca au début. Et après, on va ajouter un serveur de mosca et puis l'enlever. Le but est de contrôler si le système est toujours en service.

- Résultat :

Quand les deux serveurs de mosca sont en service, les messages sont bien reçus et puis bien stockés dans la base de données. Et après, une fois qu'un serveur est arrêté, le système est encore en service car il y a encore un serveur restant. Mais une fois qu'un serveur est rajouté, Haproxy va être au courant et va le rajouter dans la liste de serveurs de backend. Donc il s'agit que le serveur mosca peut être scalable dans notre système.

- Cas de test :

On va envoyer 10 messages en appliquant mqtt-spy. Et puis on va vérifier si les messages sont bien stockés dans mongodb.

```
function publish()
{
    var Thread = Java.type("java.lang.Thread");

    for (i = 0; i < 10; i++)
    {
        mqttspy.publish("gaoying", "hello" + i);

        // Make sure you wrap the Thread.sleep in try/catch, and
do return on exception
        try
        {
            Thread.sleep(1000);
        }
        catch(err)
        {
            return false;
        }
    }

    // This means all OK, script has completed without any issues
and as expected
```

```
    return true;
}

publish();
```

- Résultat :

On va montrer les messages stockés dans mongodb.

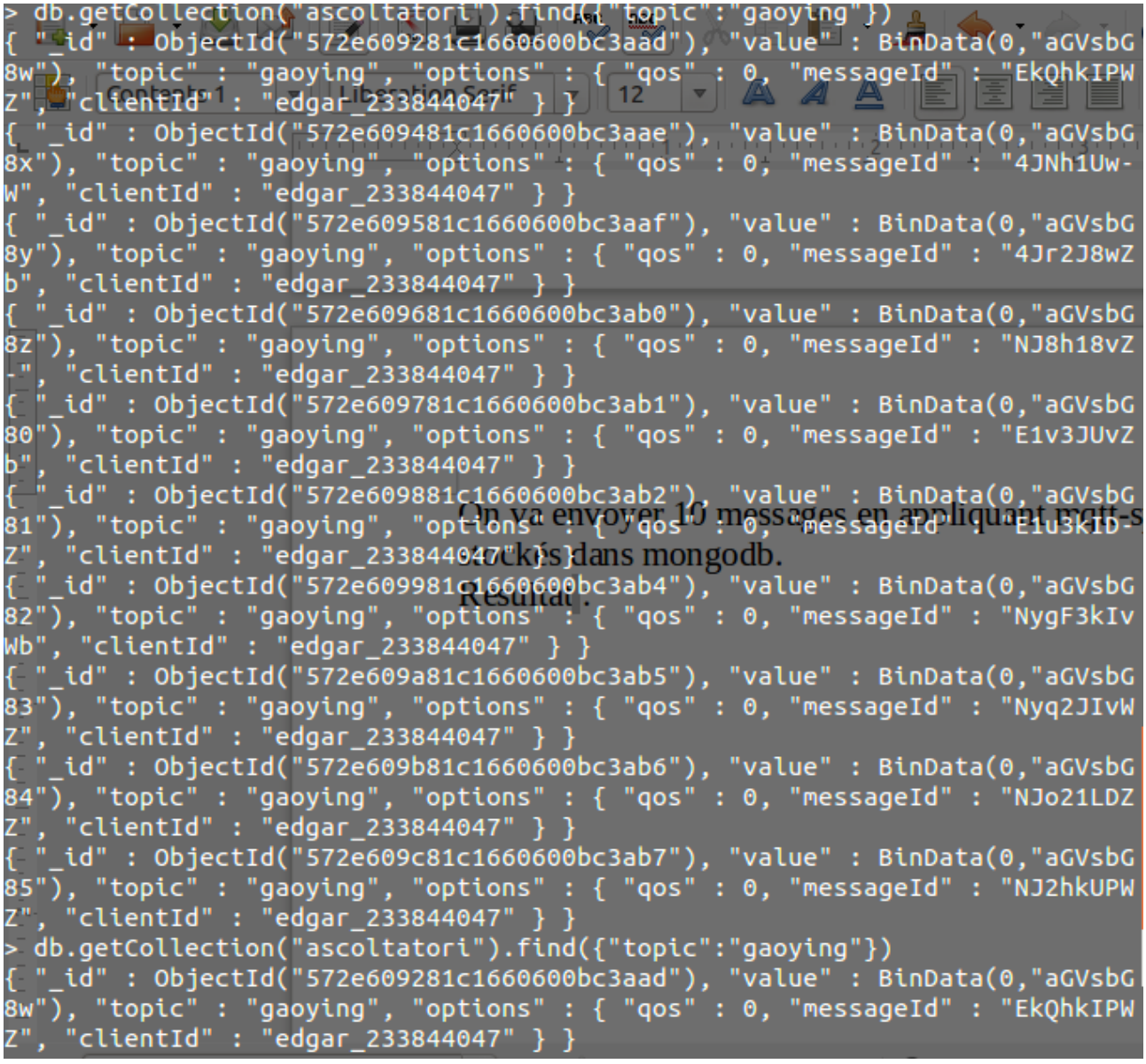



*Figure 5.3 Le résultat de mongodb*