

# Un système de collecte télémétrique pour NFV sur une plateforme Kubernetes au niveau d'un mini-centre de données

par

Assia TURKI

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE  
COMME EXIGENCE PARTIELLE À L'OBTENTION DE  
LA MAÎTRISE AVEC MÉMOIRE EN GENIE DES TECHNOLOGIES DE  
L'INFORMATION  
M. Sc. A.

MONTREAL, LE 28 AVRIL 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE  
UNIVERSITÉ DU QUÉBEC



Assia TURKI, 2021



Cette licence [Creative Commons](https://creativecommons.org/licenses/by-nc-nd/4.0/) signifie qu'il est permis de diffuser, d'imprimer ou de sauvegarder sur un autre support une partie ou la totalité de cette œuvre à condition de mentionner l'auteur, que ces utilisations soient faites à des fins non commerciales et que le contenu de l'œuvre n'ait pas été modifié.

**PRÉSENTATION DU JURY**

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Chamseddine TALHI, Directeur de Mémoire  
Département de génie logiciel et des TI à l'École de technologie supérieure

Mme Nadjia KARA, président du jury  
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Abdelouahed GHERBI, membre du jury  
Département de génie logiciel et des TI

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 08 AVRIL 2021

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE



## REMERCIEMENTS

Ce mémoire est l'aboutissement d'un dur labeur et de beaucoup de sacrifices. Sa réalisation a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Je tiens à exprimer toute ma reconnaissance à mon directeur de mémoire, Monsieur Chamseddine TALHI. Je le remercie de m'avoir encadré, orienté, aidé et conseillé.

Je souhaite également exprimer ma gratitude à Monsieur Rémi PAQUETTE de la compagnie KONTRON et son équipe de professionnels pour leur temps et leur collaboration qui a facilité la conduite de cette recherche.

Je remercie mes très chers parents, Mustapha et Hafida, qui ont toujours été là pour moi « Vous avez tout sacrifié pour vos enfants n'épargnant ni santé ni efforts. Je suis redevable d'une éducation dont je suis fier ». Je tiens à remercier mon oncle Aoued pour sa présence et ses conseils judicieux tout au long de mon parcours scolaire. Je remercie mes frères Zakaria et Aboubakr, mon neveu Yahia, et ma belle-sœur Karima pour leurs soutiens et encouragements.

Un grand merci à mon mari bien aimé, Nadjim, pour son support pendant mes moments difficiles, ses nombreux encouragements et sa contribution à l'accomplissement de ce mémoire.

Je remercie, particulièrement, ma très chère Jacynthe pour son amitié sincère, et à qui je dois ma reconnaissance et mon attachement.

J'adresse mes sincères remerciements à tous les professeurs, intervenants, et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions et ont accepté à me rencontrer et répondre à mes questions durant mes recherches.



# **Un système de collecte télémétrique pour NFV sur une plateforme Kubernetes au niveau d'un mini-centre de données**

Assia TURKI

## **RÉSUMÉ**

La technologie NFV est la solution idéale pour que les fournisseurs de services puissent offrir des réseaux fiables et efficaces et répondre aux besoins urgents. Les déploiements NFV font face à des défis importants comme l'observabilité qui couvre la collecte d'informations liées à la plateforme et aux applications (métriques). Cette opération (télémétrie) se faisait par des sondes physiques et des séparateurs dans le réseau. Cependant, NFV sur une plateforme K8s a fait apparaître de nouvelles métriques tel que la consommation de ressources des conteneurs. La télémétrie peut servir de source pour l'approvisionnement avancé et l'assurance de service, permettant non seulement des actions correctives, mais aussi préventives par rapport au fonctionnement des services réseaux. En plus, les centres de données exigent une détection d'anomalies et des réactions rapides afin de minimiser le risque de dysfonctionnement. Pour ce fait, il est important d'avoir de nouvelles approches de collecte. Cette recherche propose un système capable de collecter les nouvelles métriques qui adoptera le concept de la diffusion en temps réel et l'apprentissage automatique afin d'en déduire plus de données pour une meilleure visibilité sur les services réseaux. Notre système de collecte télémétrique s'appuie sur une comparaison entre les alternatives technologiques pour une collecte, une diffusion, un traitement et une visualisation de télémétrie bien réussis. Cette solution a pu être testée dans une entreprise où les résultats ont été prometteurs et positifs.

**Mots-clés :** NFV, Kubernetes, centre de données, télémétrie, métriques.



## **A telemetry collection system for NFV on a Kubernetes platform of a mini data center**

Assia TURKI

### **ABSTRACT**

NFV technology is the ideal solution for service providers to deliver reliable and efficient networks and meet urgent needs. NFV deployments face significant challenges such as observability which covers information's collection related to the platform and applications (metrics). This operation (telemetry) was done by physical probes and separators in the network. However, NFV on a K8s platform has revealed new metrics such as container resource consumption. Telemetry can serve as a source for advanced provisioning and service assurance, enabling not only corrective actions, but also preventive actions against the operation of network services. In addition, data centers require anomaly detection and rapid response to minimize the risk of malfunction. Therefore, it is important to have new collection approaches. This research provides a system capable of collecting new metrics that adopt the concept of real-time delivery and machine learning to derive more data for better visibility into network services. Our telemetry collection system is based on a comparison of technological alternatives for successful telemetry collection, delivery, processing, and visualization. This solution could be tested in a company where the results were promising and positive.

**Keywords:** NFV, Kubernetes, data center, telemetry, metrics.



## TABLE DES MATIÈRES

|                                                                                                         | Page       |
|---------------------------------------------------------------------------------------------------------|------------|
| INTRODUCTION .....                                                                                      | 1          |
| Problématique .....                                                                                     | 1          |
| Objectifs .....                                                                                         | 4          |
| Méthodologie .....                                                                                      | 6          |
| Organisation du rapport .....                                                                           | 9          |
| <br>CHAPITRE 1     REVUE DE LA LITTÉRATURE .....                                                        | <br>13     |
| 1.1    La virtualisation des fonctions réseau (NFV) .....                                               | 13         |
| 1.2    La surveillance (le monitoring) et l’observabilité .....                                         | 14         |
| 1.3    La télémétrie .....                                                                              | 15         |
| 1.4    L’approche DevOps et les pratiques continues .....                                               | 16         |
| 1.5    La Diffusion de données en temps réel .....                                                      | 17         |
| 1.6    Les microservices .....                                                                          | 18         |
| 1.7    Kubernetes .....                                                                                 | 18         |
| 1.7.1    Concepts de base .....                                                                         | 19         |
| 1.7.1.1    Un cluster .....                                                                             | 19         |
| 1.7.1.2    Un déploiement .....                                                                         | 19         |
| 1.7.1.3    Master Node .....                                                                            | 19         |
| 1.7.1.4    Worker Node .....                                                                            | 20         |
| 1.7.1.5    Pod .....                                                                                    | 21         |
| 1.7.1.6    Un service .....                                                                             | 21         |
| 1.7.1.7    Un espace de noms ou « namespace » .....                                                     | 21         |
| 1.7.1.8    Un réseau Kubernetes .....                                                                   | 21         |
| 1.7.2    Architecture Kubernetes .....                                                                  | 22         |
| 1.7.3    Sources de métriques dans Kubernetes .....                                                     | 22         |
| 1.8    Apprentissage automatique (Machine Learning) .....                                               | 24         |
| 1.9    La relation Kubernetes et NFV : télémétrie .....                                                 | 24         |
| 1.10   Approches de collecte de données .....                                                           | 27         |
| 1.11   Outils de collecte de télémétrie .....                                                           | 28         |
| 1.12   Kubernetes et centre de données NFV .....                                                        | 29         |
| <br>CHAPITRE 2     SOLUTION PROTOTYPE POUR LA COLLECTE DE DONNÉES<br>TÉLÉMÉTRIQUES SUR KUBERNETES ..... | <br><br>33 |
| 2.1    Introduction .....                                                                               | 33         |
| 2.2    Objectifs .....                                                                                  | 33         |
| 2.3    Processus suivi .....                                                                            | 34         |
| 2.3.1   Étude préalable : .....                                                                         | 34         |
| 2.3.2   Analyse approfondie : .....                                                                     | 35         |
| 2.3.3   Conception (Architecture de la solution) .....                                                  | 36         |
| 2.3.4   Implémentation .....                                                                            | 38         |
| 2.3.5   Test et évaluation .....                                                                        | 38         |

|         |                                                       |    |
|---------|-------------------------------------------------------|----|
| 2.3.6   | Maintenance.....                                      | 38 |
| 2.3.7   | Validation.....                                       | 39 |
| 2.4     | Choix des outils .....                                | 39 |
| 2.4.1   | Module de collecte.....                               | 39 |
| 2.4.2   | Module de diffusion et outils de traitement .....     | 40 |
| 2.4.3   | Module de visualisation .....                         | 42 |
| 2.4.4   | Module de stockage .....                              | 42 |
| 2.5     | Le bon outil pour la bonne information .....          | 44 |
| 2.5.1   | Expérimentation.....                                  | 44 |
| 2.5.1.1 | Environnement de déploiement .....                    | 44 |
| 2.5.1.2 | Les expérimentations et les outils de collectes ..... | 45 |
| 2.5.2   | Les cas d'études .....                                | 46 |
| 2.5.3   | Le choix des cas d'études : .....                     | 48 |
| 2.5.4   | Le choix des outils .....                             | 49 |
| 2.5.5   | Le choix des métriques de comparaison .....           | 51 |
| 2.5.6   | Résultats.....                                        | 52 |
| 2.5.7   | Discussion.....                                       | 55 |
| 2.5.8   | Conclusion .....                                      | 56 |
| 2.6     | Télémétrie et DevOps .....                            | 57 |
| 2.6.1   | Expérimentation.....                                  | 57 |
| 2.6.2   | Discussion .....                                      | 59 |
| 2.6.3   | Conclusion .....                                      | 60 |
| 2.7     | Discussion .....                                      | 60 |
| 2.8     | Conclusion .....                                      | 62 |

|            |                                                                                                               |    |
|------------|---------------------------------------------------------------------------------------------------------------|----|
| CHAPITRE 3 | UN SYSTÈME DE COLLECTE TÉLÉMÉTRIQUE SUR KUBERNETES AU NIVEAU D'UN MINI-CENTRE DE DONNEES NFV DE KONTRON ..... | 63 |
| 3.1        | Introduction.....                                                                                             | 63 |
| 3.2        | Contexte du projet.....                                                                                       | 63 |
| 3.2.1      | SYMKLOUD .....                                                                                                | 64 |
| 3.2.2      | Objectifs.....                                                                                                | 65 |
| 3.2.3      | Plan du travail .....                                                                                         | 65 |
| 3.3        | Le cas d'étude Kontron .....                                                                                  | 65 |
| 3.3.1      | Étude préalable.....                                                                                          | 66 |
| 3.3.2      | Analyse approfondie .....                                                                                     | 66 |
| 3.3.3      | Conception .....                                                                                              | 68 |
| 3.3.4      | Implémentation .....                                                                                          | 68 |
| 3.3.4.1    | Préparation de l'environnement : .....                                                                        | 69 |
| 3.3.4.2    | Déploiement du collecteur Prometheus : .....                                                                  | 70 |
| 3.3.4.3    | Déploiement du cluster Kafka : .....                                                                          | 73 |
| 3.3.4.4    | Déploiement du Cluster Apache Spark : .....                                                                   | 76 |
| 3.3.4.5    | Déploiement de Grafana : .....                                                                                | 80 |
| 3.3.4.6    | Intégration de Kafka et Prometheus : .....                                                                    | 81 |
| 3.3.4.7    | Intégration de Apache Spark et Kafka : .....                                                                  | 82 |





## LISTE DES TABLEAUX

|             | Page                                                                                           |
|-------------|------------------------------------------------------------------------------------------------|
| Tableau 1.1 | Comparaison entre des outils de collecte de métriques .....28                                  |
| Tableau 2.1 | Comparaison entre des outils de diffusion et messagerie (RabbitMQ, Kafka) .....40              |
| Tableau 2.2 | Comparaison entre deux outils de traitement de données (Spark et Hadoop).....41                |
| Tableau 2.3 | Comparaison entre deux outils de visualisation libres (Grafana, Kibana).....42                 |
| Tableau 2.4 | Comparaison entre deux bases de données de séries chronologiques .....43                       |
| Tableau 2.5 | Consommation CPU du conteneur et du Pod en core avec plusieurs .....52                         |
| Tableau 2.6 | Consommation CPU du conteneur et du Pod avec plusieurs groupes d'outils (Cas d'étude 2).....53 |
| Tableau 2.7 | Consommation CPU du conteneur et du Pod avec plusieurs groupes .....54                         |
| Tableau 2.8 | Résultats d'affichage d'un site web faisant partie d'un pipeline .....59                       |



## LISTE DES FIGURES

|             | Page                                                                                                                                 |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Figure 0.1  | Méthodologie de l'étude .....6                                                                                                       |
| Figure 0.2  | Processus de conception et mise en œuvre d'un système de collecte<br>télétriques global.....8                                        |
| Figure 1.1  | Architecture d'un cluster Kubernetes. ....22                                                                                         |
| Figure 1.2  | Les défis clés d'un déploiement NFV Bare Metal.....25                                                                                |
| Figure 1.3  | Deux méthodes de collecte de données .....27                                                                                         |
| Figure 1.4  | Un centre de données NFV .....30                                                                                                     |
| Figure 2.1  | Processus de création d'une solution de collecte de données<br>télétriques d'un déploiement NFV sur une plateforme Kubernetes.....34 |
| Figure 2.2  | Architecture globale du processus de la solution de collectes de<br>données télétriques d'un déploiement NFV sur Kubernetes.....36   |
| Figure 2.3  | Architecture prototype proposé comme solution de collecte<br>de données.....37                                                       |
| Figure 2.4  | Architecture du système de collecte de métriques .....46                                                                             |
| Figure 2.5  | Architecture de l'application BookInfo .....47                                                                                       |
| Figure 2.6  | Architecture de l'application BookInfo avec istio .....48                                                                            |
| Figure 2.7  | Les entités Pod et conteneur « details » ciblés .....53                                                                              |
| Figure 2.8  | Les entités Pod et conteneur « details » .....54                                                                                     |
| Figure 2.9  | Le processus du pipeline CI expérimenté avec.....58                                                                                  |
| Figure 2.10 | Le pipeline de métriques au sein d'un cluster Kubernetes .....61                                                                     |
| Figure 2.11 | Perspective Big Data "Big Data Landscap" .....61                                                                                     |
| Figure 3.1  | Cas d'utilisation simple d'un mini centre de données NFV .....64                                                                     |
| Figure 3.2  | La collecte de métriques de conteneurs par cAdvisor .....67                                                                          |
| Figure 3.3  | Architecture de la solution de collecte pour l'entreprise Kontron.....68                                                             |

|             |                                                                                            |    |
|-------------|--------------------------------------------------------------------------------------------|----|
| Figure 3.4  | Un fichier clusterRole.yml donnant.....                                                    | 70 |
| Figure 3.5  | Une copie du fichier du déploiement.....                                                   | 72 |
| Figure 3.6  | Une configuration Prometheus qui récupère.....                                             | 73 |
| Figure 3.7  | Principales terminologies.....                                                             | 74 |
| Figure 3.8  | Un cluster Kafka sur Kubernetes .....                                                      | 75 |
| Figure 3.9  | Architecture d'un cluster Spark .....                                                      | 76 |
| Figure 3.10 | Exemple d'un fichier yaml qui déploie un contrôleur .....                                  | 77 |
| Figure 3.11 | Exemple d'un fichier yaml qui déploie un contrôleur .....                                  | 78 |
| Figure 3.12 | Tableau de bord du cluster Spark déployer sur.....                                         | 79 |
| Figure 3.13 | Informations sur notre le cluster Spark déployé .....                                      | 79 |
| Figure 3.14 | L'outil Zeppeline déployé sur le cluster K8s de SymKloud.....                              | 80 |
| Figure 3.15 | intégration de Prometheus comme un consumer à Kafka.....                                   | 81 |
| Figure 3.16 | Intégration de Kafka et Spark .....                                                        | 82 |
| Figure 4.1  | Architecture d'un contrôleur de prestation de services (SDC) déployé<br>sur SymKloud ..... | 85 |
| Figure 4.2  | Architecture de l'intégration de Grafana .....                                             | 86 |
| Figure 4.3  | Le déploiement SDC sur Kubernetes Dashboard .....                                          | 88 |
| Figure 4.4  | Déploiements de Service Delivery Controller (SDC).....                                     | 88 |
| Figure 4.5  | Consommation mémoire du conteneur « Prometheus ».....                                      | 89 |
| Figure 4.6  | Consommation CPU du conteneur « redis » visualisée .....                                   | 89 |
| Figure 4.7  | Consommation mémoire des conteneurs au niveau.....                                         | 91 |
| Figure 4.8  | Consommation CPU des conteneurs au niveau .....                                            | 91 |
| Figure 4.9  | Trafic réseau entrant des conteneurs au niveau .....                                       | 92 |
| Figure 4.10 | Trafic réseau entrant des conteneurs au niveau .....                                       | 92 |
| Figure 4.11 | Consommation CPU et mémoire du cluster Kubernetes avant.....                               | 93 |

|             |                                                                      |     |
|-------------|----------------------------------------------------------------------|-----|
| Figure 4.12 | Consommation CPU et mémoire du cluster Kubernetes après .....        | 93  |
| Figure 4.13 | Le déploiement SDC sur Kubernetes Dashboard .....                    | 94  |
| Figure 4.14 | Le tableau de bord « K8s Container » après le déploiement.....       | 95  |
| Figure 5.1  | Architecture microservices proposée .....                            | 99  |
| Figure 5.2  | Exemple d'extraction de nouvelles fonctionnalités .....              | 100 |
| Figure 5.3  | Principe du fonctionnement de l'arbre.....                           | 102 |
| Figure 5.4  | Les parties fonctionnelles de l'architecture de la solution.....     | 104 |
| Figure 5.5  | Courbes des valeurs réelles et prédites de la consommation CPU ..... | 106 |



## **LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES**

|     |                                 |
|-----|---------------------------------|
| K8s | Kubernetes                      |
| NFV | Network function virtualization |
| VNF | Virtualized Network Function    |



## **INTRODUCTION**

La virtualisation des fonctions réseau (NFV) est une approche qui divise les fonctionnalités réseau et les virtualise dans le but d'augmenter la performance, la flexibilité et l'évolutivité du réseau. Et depuis son adoption par l'environnement industriel, diverses fonctionnalités quittent le matériel physique très coûteux et deviennent des instances virtualisées.

Cependant, la performance de ces services virtualisés dépend de certains facteurs et fait face à des défis divers. Ça peut être l'architecture dont ses blocs sont structurés, ou le principe adopté en allocation de ressources et bien d'autres obstacles. Il est aussi important de prendre en considération le fait que des nouvelles technologies émergent le marché actuel et que les compagnies les adoptent pour rester compétitifs et garder le pouvoir de gérer du trafic massif et évoluer rapidement selon leurs besoins d'affaires.

Dans ce contexte, de nouveaux paradigmes tel que la combinaison de microservices événementiels, de conteneurs, d'orchestrateurs, dont Kubernetes est le plus populaire, et de pipelines de données d'apprentissage automatique accélère le développement d'applications de nouvelle génération tel que NFV. Ces applications tirent parti de ces paradigmes informatiques nouveaux, et se reposent sur une infrastructure informatique moderne avec de nouveaux critères et des défis qui s'avèrent parfois assez complexes à relever comme l'incompatibilité à gérer, la gestion de ressources ou bien d'autres.

### **Problématique**

La technologie de virtualisation des fonctions réseau est largement utilisée par les entreprises qui fournissent les services réseaux. Un service réseau est une composition de fonctions de réseaux défini par sa spécification fonctionnelle et comportementale. La technologie NFV offre de nombreux avantages : évolutivité, flexibilité et coût relativement bas par rapport à la construction du réseau en réduisant le besoin des équipements réseaux et le cout des opérations. Elle est la clé et la solution idéale pour que les fournisseurs de services puissent offrir des

réseaux fiables et efficaces et répondre aux besoins urgents (ex : lancement de nouveaux services).

Le déploiement d'une fonction réseau virtuel (VNF) tel qu'un pare-feu ou un routeur est désormais sur des serveurs standards d'une haute performance. Ils existent plusieurs modèles à ce déploiement vu l'évolution de la technologie et l'apparition de nouveaux besoins en termes de services réseau. Ils existent des architectures NFV sur uniquement des machines virtuelles, ou des conteneurs, comme ça peut être un plan hybride regroupant les deux technologies avec un fonctionnement séparé ou unifié. Selon Ramakrishnan (2019) cette multitude de modèles offre plusieurs choix aux compagnies et souligne les avantages de ces déploiements NFV en combinaison avec ceux des technologies sous-jacentes.

Cependant, les déploiements NFV sont souvent très complexes et exigent beaucoup de temps pour leur mise en œuvre et font face à de nombreux défis. Ces derniers sont importants, ils peuvent être d'ordre opérationnel, technologique ou de gestion et d'orchestration (Gestion des services VNFs, surveillance, etc.).

La surveillance est un élément très nécessaire pour assurer la bonne fonctionnalité des services réseau virtualisés et la performance des déploiements NFV, dans le but de repérer tout dysfonctionnement et répondre rapidement aux différentes perturbations de fonctionnement de différentes causes.

« L'observabilité est un des piliers de la surveillance efficace et nouvellement perçue par la communauté des utilisateurs précoces de technologies comme la nouvelle surveillance » (Crosby, Hoogendoorn, Shnitger, & Tremel, 2018). Elle regroupe les concepts de la traçabilité, la journalisation, les alertes et les métriques (la télémétrie). Les données télémétriques sont des informations relatives au déploiement lui-même et à la plateforme, dans ce contexte on parle de déploiements NFV conteneurisés. La télémétrie peut servir de source pour l'approvisionnement avancé et l'assurance de service, permettant non seulement des actions correctives, mais aussi préventives comme résultat positif de la capacité de gestion.

On distingue trois types de métriques utiles à cette recherche : la bonne métrique, la nouvelle métrique et la métrique de performance. La bonne métrique est l'information que nous cherchons et qui répond à notre objectif de l'opération de collecte. La nouvelle métrique est l'information liée aux nouvelles unités de la plateforme moderne Kubernetes (c.à.d. les conteneurs, les pods, ...etc.) qui n'existaient pas dans l'architecture du réseaux traditionnel physique. En ce qui concerne la métrique de performance, c'est l'information qui permet de mesurer et d'évaluer la performance des services réseaux.

Pour des conteneurs orchestrés par un outil tel que Kubernetes, les informations télémétriques de la plateforme ne sont plus désormais les mêmes ou leur collecte se fait par des sondes physiques et des séparateurs dans le réseau, et ceci représente un vrai défi. La nouvelle approche de la surveillance doit prendre en considération la technologie des conteneurs et de l'orchestrateur Kubernetes qui se repose sur la répartition du trafic entre de nombreuses ressources virtualisées légères et transitoires qui ont exacerbé le problème de visibilité.

Ce problème de la télémétrie, confronté par les fournisseurs de services réseau, devrait être pris en considération et relevé pour une surveillance efficace et performante. C'est primordial pour un opérateur qui est amené à assurer aux fournisseurs de services la rapidité et l'agilité nécessaires pour déployer des fonctions réseau automatisées à un rythme compétitif sur le marché. Une seconde contrainte pour les entreprises est d'assurer la minimisation du risque d'échec de leurs propres centres donnés. Un manque d'informations télémétriques entraîne une mauvaise visibilité et influence gravement la disponibilité des fonctionnalités et des services TI d'une façon générale.

De ces faits, il est important d'améliorer la collecte de données télémétriques de la plateforme Kubernetes et du déploiement NFV sur les centres de données. Puisqu'il s'agit d'un ensemble de technologies combinées autour de ce défi (conteneurs, kubernetes, microservices, DevOps), la démarche commencera par comprendre la relation entre ces dernières avec un accent sur la télémétrie. Quelle influence ont ces technologies sur la bonne extraction de ces nouvelles mé-

triques ?! On cible la NFV, et l'orchestrateur Kubernetes qui est non seulement un orchestrateur de conteneurs, mais aussi un outil puissant d'automatisation faisant partie de la culture DevOps.

Cette recherche vise à fournir un système de collecte d'informations télémétriques qui s'appuie sur les besoins spécifiés de l'industrie, identifie les bons moyens de collecte et facilite l'adoption des technologies NFV sur une plateforme moderne comme Kubernetes qui peut être déployé sur un centre de données en prenant compte ses limitations. Des solutions antérieures existent comme le travail de Khelifi (2019) et celui de Ramakrishnan (2019), mais certaines ne sont pas flexibles pour répondre aux exigences de l'industrie, et d'autres n'ont pas été testées sur un environnement Kubernetes au niveau d'un centre de données dédié à la NFV. Elles ne prennent pas en considération qu'un déploiement NFV sur une plateforme Kubernetes engendre une masse de données très importantes potentiellement très coûteuse pour une entreprise qui a une limitation en ressources. Aussi, ces métriques ne sont pas toutes forcément nécessaires pour l'opérateur ou la compagnie et leur sauvegarde peut être inutile et un gaspillage.

## Objectifs

Après avoir souligné que les informations télémétriques des réseaux ne sont plus désormais les mêmes avec la modernisation de ces dernières et l'adoption de nouvelles technologies et approches (Conteneurs, Kubernetes, NFV, DevOps), le but est de faire progresser le processus de collecte de télémétrie dans ce milieu complexe. Ainsi, nous citons les objectifs de cette recherche comme suit :

1. La création d'une démarche étudiée et flexible qui s'appuie primordialement sur les besoins d'une entreprise en question. Ceci consiste à mettre en amont des questions très significatives et importantes pour s'assurer que la solution répond aux exigences et éviter les incompatibilités des composantes et outils du système de collecte télémétrique.

2. Une collecte télémétrique efficace et des métriques suffisantes pour qu'elles soient au cœur des préoccupations en offrant une visibilité complète selon l'objectif fixé.
3. Un système de collecte qui prend en considération les nouvelles métriques tel que la consommation CPU des conteneurs et surtout que la plateforme Kubernetes exacerbe le problème de la visibilité.
4. Une solution déployable sur Kubernetes capable à supporter la masse de données que cette plateforme génère en plus de ses applications et ses déploiements.
5. Une approche qui adhère à l'isolement des composants et des services pour faciliter la gestion et la localisation rapide des troubles et des pannes.
6. Des informations télémétriques fournies en temps réel. La diffusion à temps réel n'est pas toujours requise pour les environnements en TI comme pour un système de sauvegarde. Cependant, ceci est important pour les services, les déploiements et les centres de données NFV qui exigent une détection d'anomalies et des réactions rapides afin de minimiser le risque d'échec des fonctions réseaux.
7. Une solution avec un système de prédiction basé sur les informations collectées afin d'améliorer la visibilité sur les services réseaux, de prévoir les dysfonctionnements et d'éviter les échecs des centres de données NFV.

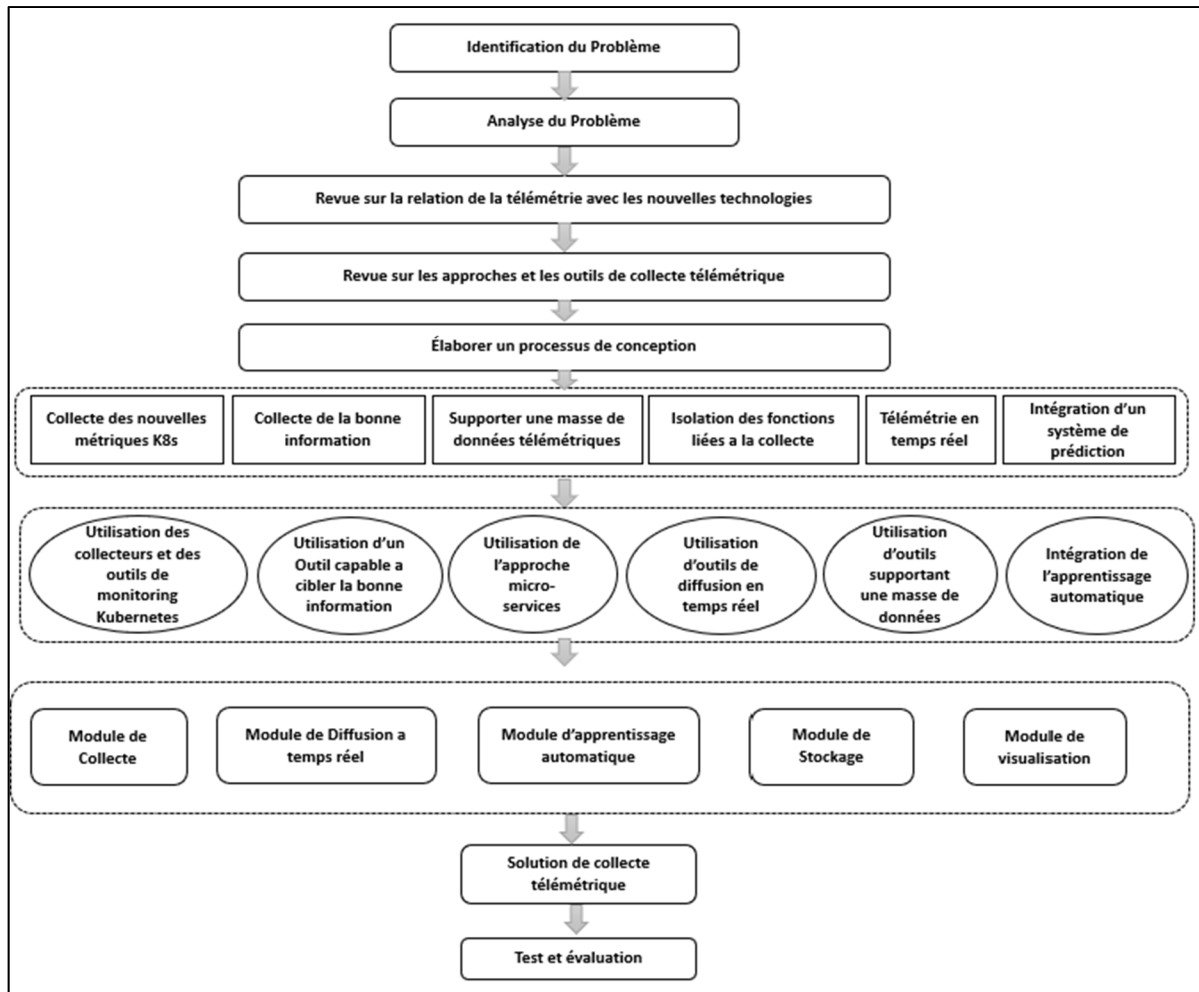


Figure 0.1 Méthodologie de l'étude

## Méthodologie

Une méthodologie précise est suivie dans cette étude et elle se compose de plusieurs étapes suivant les objectifs fixés (figure 0.1).

La première étape consiste à identifier et comprendre les défis devant une collecte télémétrique sur une plateforme de conteneurs orchestrés par Kubernetes dans un contexte NFV. À partir de ces données, une problématique a été formulée dans le but de fournir une solution efficace qui répond aux besoins.

Le traitement du problème débutera par une revue de littérature afin d’analyser et d’examiner les solutions déjà existantes et de clarifier certains concepts en relation avec notre question de recherche qui est la collecte télémétrique pour NFV sur une plateforme Kubernetes d’un mini-centre de données. Il s’agit de construire une bonne compréhension autour de la relation du concept télémétrie avec la technologie Kubernetes dans un cadre NFV, avec l’approche d’automatisation et DevOps et avec les centres de données (aspect matériel) qui produisent plus de limitations à une entreprise (limitations matérielles et en ressources). Il est important aussi d’étudier les approches et les outils de télémétrie pour pouvoir déterminer les avantages et les inconvénients qui nous permettront de fournir une solution cohérente dans un milieu technologique ou les alternatives causent confusion.

Avant de passer à l’identification des caractéristiques que notre solution doit avoir, nous avons élaboré un processus de conception. C’est une suite d’étapes organisées qui englobe et structure toute notre démarche, nos réflexions et nos expérimentations pour aboutir à un système de télémétrie qui répond aux objectifs de notre recherche et à ceux d’une entreprise. Comme le montre la figure 0.2, c’est une partie conception et le résultat d’une étude préalable pour une identification des exigences, et d’une analyse approfondie des besoins. Cette dernière a plusieurs tâches comme identifier les métriques exigées, la bonne méthode et les outils efficaces à la collecte. Ensuite, l’architecture de la solution résultante est implémentée au niveau l’environnement de test ou de la préproduction afin d’être testée et évaluée. Si les résultats sont positifs, la solution est validée pour être mise en œuvre en production. Dans le cas contraire, nous nous retrouvons dans une phase de maintenance qui peut demander un retour à une des phases précédentes selon le niveau et le type du problème détecté. À titre d’exemple, l’évaluation montre que le système ne collecte pas une certaine métrique essentielle ou une simple mise à jour de la configuration d’un outil de collecte pour résoudre le problème, et cela ne nécessite pas un retour à des phases antérieures. Ce processus de travail est abordé avec plus de détails dans la section 3.3 (chapitre 3, section 3).

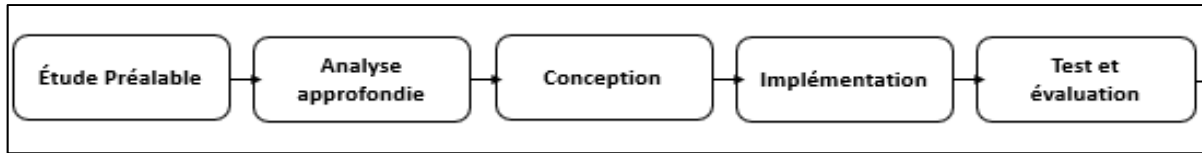


Figure 0.2      Processus de conception et mise en œuvre d'un système  
de collecte télémetrique global

Les caractéristiques de notre système de collecte établies selon les objectifs sont comme suit :

1. Utiliser des outils de monitoring Kubernetes; les outils de monitoring munis des composants dont la fonction est la collecte de métriques. Il est très utile de les utiliser pour assurer la collecte des nouvelles métriques liées à la surveillance des déploiements NFV conteneurisés en global.
2. Utiliser des méthodes efficaces et des outils de collecte configurables pour atteindre plusieurs sources de données et trouver la bonne information.
3. Utiliser des outils capables à gérer une bonne masse de données et des espaces de stockage de données libres compatibles à nos outils de collectes.
4. Utiliser l'approche microservices qui est légère, facile à enrichir de fonctions et traite rapidement les problèmes qui surviennent grâce à l'isolation de ses éléments.
5. Utiliser des outils de diffusion de données en temps réel.
6. Séparer des fonctions du système de collecte; créer des modules de collecte de métriques, de diffusion, de stockage, de visualisation et de prédiction (apprentissage automatique).
7. Intégrer l'approche de l'apprentissage automatique pour avoir un module de prédiction au niveau du système de collecte.

Les fonctions des composants du système de collecte à fournir sont les suivantes :

1. Module de collecte : ce module consiste à cibler la source des données télémetriques de la plateforme NFV et de récupérer seulement celles qui correspondent aux besoins définis de l'entreprise.

2. Module de diffusion : les informations télémétriques ont besoin d'être diffusées vers d'autres étapes de la surveillance; ce module est chargé de la diffusion en temps réel et du stockage des données.
3. Module de visualisation : le but est de fournir un module qui permet de voir et faciliter la lecture de ces données télémétriques sur un tableau de bord flexible avec des représentations graphiques et des taux différents selon les objectifs voulus.
4. Module de prédiction ou d'apprentissage automatique : ce composant doit être capable aussi d'assurer l'intégration de l'apprentissage machine automatisée pour de futurs objectifs liés à la surveillance d'un déploiement NFV sur une plateforme Kubernetes.

La dernière étape de cette méthodologie est une suite de tests afin de pouvoir évaluer l'efficacité et la performance de notre solution. Une grande partie de ses tests sont déjà inclus dans notre processus proposé comme le montre la figure 0.2 qui assure que le système proposé répond aux objectifs fixés et qu'il est adaptable aux spécifications d'une entreprise.

Une compagnie leader mondiale de la technologie informatique embarquée, sous le nom de « Kontron », nous a servis comme un cas d'étude pour réaliser cette évaluation du système de collecte télémétrique. Le travail a été mené à partir du laboratoire Multimédia de l'ÉTS et sur la plateforme SYMKLOUD qui se trouve au niveau de Kontron.

## **Organisation du rapport**

La partie introduction est une entrée à notre étude qui souligne la question de recherche qui a nécessité ce travail. Elle contient aussi les objectifs fixés avec un aperçu sur notre méthodologie de recherche suivie afin d'aboutir à notre solution.

Dans le premier chapitre, une revue de littérature présente plusieurs concepts liés à la collecte de métriques d'un déploiement NFV sur un environnement de conteneurs orchestrés par l'outil

Kubernetes sur un centre de données. Le chapitre inclut aussi des solutions existantes et des travaux antérieurs en citant leurs limitations et en effectuant des comparaisons. Jusqu'ici, nous avons réalisé une bonne analyse du problème et des défis devant une bonne collecte télémétrique dans un environnement technologiquement moderne. Ceci nous a permis de fixer les objectifs et d'identifier les caractéristiques du système que nous souhaitons concevoir et qui sont illustrés dans la figure 0.1 par des formes ovales.

Le deuxième chapitre décrit la solution proposée par ce travail et la démarche appliquée qui a mené à un système qui détient les caractéristiques déterminées pendant l'étude préalable et l'analyse du problème de recherche. Ce processus de conception est montré par la figure 0.2.

Bien que notre processus de conception, présenté dans la figure 0.2, inclut la phase de test et d'évaluation, j'ai choisi de la réaliser sur un environnement industriel qui est la compagnie Kontron. Elle a servi comme un cas d'étude avec des contraintes et des besoins réels tel que la limitation du matériel (mini-centre de données) en ressource et le besoin de mesurer la consommation d'un déploiement NFV en ressource sur une plateforme Kubernetes. Il a été très intéressant d'étudier la faisabilité et l'efficacité d'un tel système sur un vrai mini-centre de données NFV et fournir des résultats plus concrets. Ce travail est le contenu du chapitre trois.

Le chapitre quatre présente les résultats de notre implémentation et de l'évaluation de la solution télémétrique en termes de fiabilité et de performance. Une autre étape d'évaluation est celle du chapitre cinq. Elle présente un travail de recherche de Khelifi (2019) avec un système très similaire qui a été inspirant et influençant dans la conception de notre système de collecte. Il a été réalisé sur des conteneurs Docker non orchestrés, avec un système de traitement de données Apache Spark qui est muni du module «MLib» chargé de la prédiction basée sur l'apprentissage automatique. L'idée est d'injecter nos informations télémétriques prises de notre système sur Kubernetes à ce système sur Docker et étudier cette faisabilité à prédire d'autres informations avec efficacité, ce qui améliora la visibilité et l'observabilité de la collecte télémétrique.

À la fin, une conclusion est rédigée en se basant sur le déroulement de notre recherche et les résultats obtenus de l'évaluation de notre système conçu pour la collecte télémétrique sur une plateforme Kubernetes hébergée sur un mini-centre de données avec des déploiements NFV.



## **CHAPITRE 1**

### **REVUE DE LA LITTÉRATURE**

La communauté de recherche reconnaît qu’une nouvelle combinaison de technologies (conteneurs, orchestrateurs de conteneurs, microservices, apprentissage automatique, DevOps) puissante est en train de surgir dans l’industrie, et qu’en parallèle, elle est venue avec de nouveaux défis à soulever.

Dans ce chapitre, nous allons présenter plusieurs concepts nécessaires à ce travail. En premier, les notions de la surveillance et de l’observabilité dans le domaine de la virtualisation des fonctions réseau. Ceci va expliquer l’importance de la télémétrie et la nécessité d’approfondir la compréhension de la relation de NFV avec de nouvelles technologies émergentes telles que DevOps et Kubernetes. Ce chapitre est important vu qu’il fixe notre compréhension aux définitions de certains concepts qui, souvent, sont ambiguës et comprises différemment d’un chercheur ou d’un ingénieur à un autre.

#### **1.1 La virtualisation des fonctions réseau (NFV)**

Par la recherche (Gil Herrera & Botero, 2016), la virtualisation des fonctions réseau (NFV) est une nouvelle technologie et un nouveau cadre d’architecture réseau dans lequel les fonctions réseau qui utilisaient traditionnellement du matériel dédié (boîtiers intermédiaires ou équipement réseau) sont désormais implémentées dans des logiciels qui s’exécutent sur du matériel à usage général tel qu’un serveur à volume élevé. NFV émerge comme une initiative de l’industrie (opérateurs de réseaux, opérateurs et fabricants) afin d’augmenter la flexibilité de déploiement et l’intégration de nouveaux services réseau avec une agilité accrue au sein des réseaux d’opérateurs et d’obtenir des réductions significatives des dépenses d’exploitation et des dépenses d’investissement.

## 1.2 La surveillance (le monitoring) et l'observabilité

Dans Wikipédia, le monitoring ou le monitoring sont l'anglicisme du terme surveillance et définissent la mesure d'une activité. Il est aussi possible d'utiliser le mot supervision dans un contexte technique ou informatique. C'est une acquisition de données (mesures, alarmes, retour d'état de fonctionnement) et des paramètres de commande des processus généralement confiés à des automates programmables. Dans le cadre de ce travail, les quatre termes ont le même sens abordé.

La surveillance réseau est un processus continu dans le temps qui récupère toutes les informations possibles des composants réseau (routeur, pare-feu, serveur...) pour repérer tout dysfonctionnement et évaluer leurs santé, performances et disponibilités. Dans un processus de surveillance, il est nécessaire d'identifier les entités qui doivent être observées et suivies, et surtout ceux qui ont un impact sur la performance du réseau. Ceci permet une détection précoce des problèmes et empêche plusieurs pannes et dégâts.

L'observabilité est un terme qui est beaucoup plus utilisé avec l'apparition des nouvelles technologies où les outils de surveillance ont beaucoup évolué, et depuis que les développeurs sont aussi concernés par l'environnement de production et ses retours d'informations. Cette collaboration issue de la culture DevOps a bien éliminé la tradition qui dit que les opérateurs sont les meneurs de la surveillance. Désormais, les ingénieurs DevOps peuvent observer des applications évolutives s'exécutant sur des conteneurs dans un environnement orchestré tel que Kubernetes. Cette notion d'observabilité repose sur quatre piliers :

1. **Métriques** : les métriques sont un moyen quantitatif et qualitatif de vérifier un comportement souhaité, c'est une agrégation d'événements similaires pour obtenir un niveau de compréhension plus élevé.
2. **Alerte** : ce sont des notifications au moment où le comportement de l'entité observée n'est pas acceptable et pourrait devenir problématique.
3. **Traçage** : c'est fournir et enregistrer un contexte basé sur la liaison des événements et des comportements.
4. **Journalisation (Logging)** : l'enregistrement de tout événement d'une façon centralisée pour qu'ils puissent être recherchés et consultés. (Crosby et al., 2018)

En se référant au papier scientifique de Crosby et al., (2018), la surveillance a une signification différente aujourd'hui. Dans le passé, les développeurs construisaient les applications. Les déploiements ont été effectués par des équipes opérationnelles qui ont géré les applications en production. L'intégrité des services a été principalement déterminée en fonction des commentaires des clients et des mesures matérielles telles que l'utilisation du disque, de la mémoire ou de l'unité centrale de traitement (CPU). Ceci sous-entend que l'observabilité est la nouvelle surveillance.

Cependant, un document google à une autre perspective :

Les objectifs de la surveillance et de l'observabilité sont différents. L'observabilité ne remplace pas la surveillance et n'élimine pas la nécessité de la surveillance ; ils sont complémentaires. "L'observabilité" pourrait être un nouveau terme fantaisiste à l'horizon, mais ce n'est vraiment pas une idée nouvelle. Les événements, le traçage, le suivi des exceptions sont tous des dérivés des journaux (Logs), et si l'on a utilisé l'un de ces outils, on a déjà une certaine observabilité. Certes, les nouveaux outils et les nouveaux fournisseurs auront leur propre définition et compréhension du terme, mais en substance, l'observabilité capture ce que la surveillance ne fait pas (et idéalement, ne devrait pas). (Google, 2017)

Dans le cadre de ce travail, la perspective est qu'observer est autre que surveiller. Un système d'observabilité est un système qui permet, seulement, la collecte de l'information et sa visualisation. Il est considéré comme une première étape, ou un module, d'un processus de surveillance complet. Et pour cela, dire que la télémétrie est importante pour implémenter un système d'observabilité assez complet et efficace.

### **1.3 La télémétrie**

D'un point de vue holistique, la télémétrie est la fonction de collecte et l'agrégation d'un ensemble d'informations à partir de variables (métriques) ciblées à partir d'un système via des requêtes bien pertinentes et réactives. C'est un pilier nécessaire pour l'observabilité et le monitoring de l'entité à surveiller.

En réseau, elle représente à la fois les anciennes et les nouvelles formes de statistiques de réseau, ainsi que de nouvelles méthodes de collecte et d'exposition des données. Cependant, l'industrie a tendance à utiliser ce terme « Télémétrie » pour souligner cette évolution technologique qui a fait que les données et les informations collectées aujourd'hui n'ont pas le même format, et que les méthodes de collecte sont de plus en plus sophistiquées.

La télémétrie des applications est la collecte des mesures et des métriques spécifiques à l'application. Mais elles ne sont en aucun cas complètes pour assurer la fonction de surveillance. Il existe de nombreuses autres mesures à prendre en compte, notamment la surveillance des mesures spécifiques à l'application, les problèmes liés à l'environnement (Infonuagique, conteneurs, machines virtuelles...), les failles de sécurité potentielles et, surtout, la rétroaction d'utilisateur. Ce fait affirme qu'il existe une relation entre les différents types de métriques.

#### **1.4 L'approche DevOps et les pratiques continues**

DevOps est de plus en plus adopté par les industries du domaine du génie logiciel. Cette production du logiciel a connu beaucoup d'évolution et plusieurs grands changements. Et DevOps a bien changé les perceptions, les méthodes et les outils dans ce domaine.

« DevOps est perçue comme une culture organisationnelle qui met l'accent sur le travail collaboratif entre développeurs et opérateurs de logiciels. Mais le point de vue le plus global est de dire que DevOps est une collaboration des personnes pour fournir des logiciels, mais en quatre façons : culture, automatisation, mesure et partage » (Humble & Molesky, 2011, p.7), et sur la même ligne Lwakatare, Kuvaja, & Oivo (2015, p.212) identifient quatre éléments de DevOps : la collaboration, l'automatisation, la mesure et la surveillance. Ces éléments sont capables d'éclaircir la nature de l'ambiguïté entre DevOps et les pratiques continues. L'intégration continue (CI), la livraison continue (CD) et le déploiement continu, appelés pratiques continues pour cette étude, sont quelques-unes des pratiques visant à aider les organisations à accélérer leur développement et la livraison de fonctionnalités logicielles sans compromettre la qualité.

« L'intégration continue (CI) est une pratique de développement largement établie dans l'industrie du développement logiciel » (Fitzgerald, & Stol, 2017, p.176), dans laquelle les membres d'une équipe intègrent et fusionnent le travail de développement (par exemple, le code) fréquemment, par exemple plusieurs fois par jour. CI permet aux éditeurs de logiciels d'avoir un cycle de publication plus court et plus fréquent, d'améliorer la qualité des logiciels et d'accroître la productivité de leurs équipes selon (Fitzgerald, & Stol, 2017, p.176). Cette pratique comprend la construction et les tests de logiciels automatisés.

La livraison continue (CD) vise à garantir qu'une application est toujours prête à la production après avoir réussi les tests automatisés et les contrôles de qualité comme identifiés par Weber, Nepal, & Zhu (2016, p.74) et Humble (2016). « La livraison continue utilise un ensemble de pratiques, par exemple, CI et l'automatisation du déploiement pour fournir automatiquement des logiciels à un environnement de production » (Puppet, 2015). Selon Chen (2015, p.52) et Humble (2016), cette pratique offre plusieurs avantages tels que des risques de déploiement réduits, des coûts réduits et une rétroaction des utilisateurs plus rapide. La figure 1.1 indique qu'avoir une pratique de livraison continue nécessite une pratique d'intégration continue.

Un pipeline se compose d'étapes qui sont exécutées en fonction d'un déclencheur (par exemple, pousser vers le référentiel de code source). Habituellement, les étapes ont des responsabilités spécifiques (par exemple, compiler le code source, exécuter des tests unitaires). Chaque étape peut contenir plusieurs tâches qui sont exécutées étape par étape. En outre, ils permettent d'atteindre une fréquence élevée d'incrémentations de fonctionnalités dans des cycles d'incrémentations rapides avec un retour rapide aux développeurs. (Nygard, 2007)

## **1.5 La Diffusion de données en temps réel**

La diffusion de données en temps réel est une collecte de données et d'informations en continu et en temps réel en les poussant vers un emplacement centralisé. Le terme « Streaming de données » est son anglicisme, et il est le plus utilisé par les ingénieurs et les chercheurs de nos jours.

Ces données doivent être traitées de manière séquentielle et incrémentielle au cas par cas ou sur des fenêtres temporelles flexibles, et utilisées pour différentes analyses, notamment les corrélations, les agrégations, le filtrage et l'échantillonnage. Les informations tirées de cette analyse donnent aux entreprises une bonne visibilité sur de nombreux aspects de leurs affaires et des activités des clients, tels que l'utilisation des services (à des fins de mesure/facturation), l'activité des serveurs, les clics effectués sur les sites Web et la géolocalisation d'appareils, de personnes et de biens physiques, et leur permettent de répondre rapidement aux nouvelles situations. (AWS, s.d).

## **1.6 Les microservices**

La définition la plus claire donnée est :

Les microservices désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Contrairement à une approche monolithique classique, selon laquelle tous les composants forment une entité indissociable, les microservices fonctionnent en synergie pour accomplir les mêmes tâches, tout en étant séparés. Chacun de ces composants ou processus est un microservice. Granulaire et léger, ce type de développement logiciel permet d'utiliser un processus similaire dans plusieurs applications. Il s'agit d'un élément essentiel pour optimiser le développement des applications en vue de l'adoption d'un modèle cloud-native. (Redhat, 2020).

## **1.7 Kubernetes**

Kubernetes est un outil open source (libre) de gestion de clusters et de conteneurs. Il a été créé par Google en 2014. « Il fournit une plateforme qui permet d'automatiser le déploiement, la mise à l'échelle et les opérations des conteneurs d'applications sur les clusters d'hôtes. Surtout, cela réduit le coût des dépenses de l'infonuagique (cloud-computing) et simplifie les opérations et l'architecture » (Developintelligence, 2017).

### **1.7.1 Concepts de base**

L'idée de base de Kubernetes est d'éloigner davantage les machines, le stockage et les réseaux de leur implémentation physique. Il s'agit donc d'une interface unique pour déployer des conteneurs sur toutes sortes de Cloud, de machines virtuelles et de machines physiques. Dans ce qui suit, nous résumons quelques concepts de bases de Kubernetes :

#### **1.7.1.1 Un cluster**

Il s'agit d'une collection de Nœuds qui partagent leurs ressources disponibles et forment une machine plus puissante. En déployant des programmes sur le cluster, ce dernier gère intelligemment la distribution du travail vers les nœuds individuels. Au minimum, un cluster contient un plan de contrôle et une ou plusieurs machines de calcul, ou nœuds. Les nœuds exécutent en fait les applications et les charges de travail.

#### **1.7.1.2 Un déploiement**

Un déploiement est une couche d'abstraction supplémentaire à Kubernetes pour gérer les pods et les lancer dans un cluster. Son objectif principal est de déclarer le nombre de répliques d'un Pod à exécuter à la fois. Lorsqu'un déploiement est ajouté au cluster, il entraîne automatiquement le nombre de modules requis, puis les surveille. Si un Pod meurt, le déploiement le recrée automatiquement.

Avec un déploiement, on ne traite pas les pods manuellement. Ils seront gérés automatiquement.

#### **1.7.1.3 Master Node**

Le master Node (le nœud maître) est une collection de composants du panneau de contrôle de Kubernetes. Il reçoit des entrées d'une CLI (interface de ligne de commande) ou d'une interface utilisateur (interface utilisateur) via une API. Ce sont les commandes que vous fournissez à Kubernetes. Ils sont utilisés pour la prise de toutes les décisions du cluster. Ce master Node

comprend à la fois la planification et la réponse aux événements de cluster. Il a divers composants :

1. **Serveur API (API Server)** : le serveur API est le frontal du cluster Kubernetes. Toutes les opérations sur les pods, les services, etc., sont exécutées par programme en communiquant avec les points de terminaison fournis par celui-ci.
2. **Contrôleur (Controller)** : le rôle du contrôleur est d'obtenir l'état souhaité du serveur API. Il vérifie l'état actuel des nœuds qu'il est chargé de contrôler, détermine s'il existe des différences et les résout, le cas échéant.
3. **Planificateur ou ordonnancer (Scheduler)** : l'ordonnancer est responsable de l'attribution du travail aux différents nœuds. Il surveille la capacité des ressources et garantit que les performances d'un nœud de travail se situent dans un seuil approprié.
4. **Etc** : c'est une base de données utilisée par Kubernetes pour sauvegarder toutes les données du cluster. Elle stocke l'intégralité de la configuration et de l'état du cluster. Le master Node interroge etcd pour récupérer les paramètres de l'état des nœuds, des pods et des conteneurs.

#### 1.7.1.4 Worker Node

Le worker node (nœud de travail) est le groupe de composants qui contient tous les services requis pour gérer la mise en réseau entre les conteneurs, et pour communiquer avec le master node, ce qui nous permet d'affecter des ressources aux conteneurs planifiés. Il a d'autres composants que les Pod et les conteneurs :

1. **Kubelet** : le principal agent de Kubernetes qui s'exécute sur chaque nœud du cluster ; il obtient la configuration d'un pod à partir du serveur API et s'assure que les conteneurs décrits sont opérationnels.
2. **Le Kube-proxy** : il agit comme un équilibreur de charge et un proxy réseau pour effectuer le service sur un seul nœud de travail.

#### **1.7.1.5 Pod**

Le Pod est l'unité d'organisation principale de Kubernetes. C'est un ensemble de conteneurs administrés en tant que groupe sur un nœud, la même machine physique, ou virtuelle et conçue pour pouvoir communiquer facilement entre eux.

#### **1.7.1.6 Un service**

Défini par Hale (2019) : « Un service K8s crée un point d'accès unique pour un groupe de pods. Un service fournit une adresse IP et un port cohérent pour accéder aux pods sous-jacents. Les utilisateurs externes et les pods internes utilisent les services pour communiquer avec d'autres pods. ».

#### **1.7.1.7 Un espace de noms ou « namespace »**

La définition donnée par Google Cloud Est : « un espace de noms peut être considéré comme un cluster virtuel à l'intérieur de notre cluster Kubernetes. Il est possible d'avoir plusieurs espaces de noms dans un même cluster Kubernetes, et ils sont tous logiquement isolés les uns des autres. Ils peuvent nous aider, nous et nos équipes, dans l'organisation, la sécurité et même les performances ! » (Dinesh, 2018). Cette notion permet la gestion, la division et la limitation des ressources du cluster entre plusieurs utilisateurs et déploiement.

#### **1.7.1.8 Un réseau Kubernetes**

Configurer le réseau Kubernetes consiste à utiliser l'interface réseau de conteneur (CNI) et un plug-in de réseau. L'artefact (Love, 2017) a défini le CNI comme suit : « CNI est une définition de bibliothèque et un ensemble d'outils regroupés dans le projet Cloud Native Computing Foundation ». Après sa mise en place, les configurations de base sont générées automatiquement ; ainsi la création et l'administration de réseaux deviennent plus faciles. Un CNI dispose d'un large éventail de supports dont la spécification est simple à mettre en œuvre. Il assure la

connectivité réseau des conteneurs et la suppression des ressources allouées lorsqu'un conteneur est supprimé ; ceci comprend l'attribution d'adresse IP et l'installation de toutes les routes requises.

### 1.7.2 Architecture Kubernetes

La figure 1.1 ci-dessous rassemble l'ensemble des concepts de base de Kubernetes qui compose un cluster.

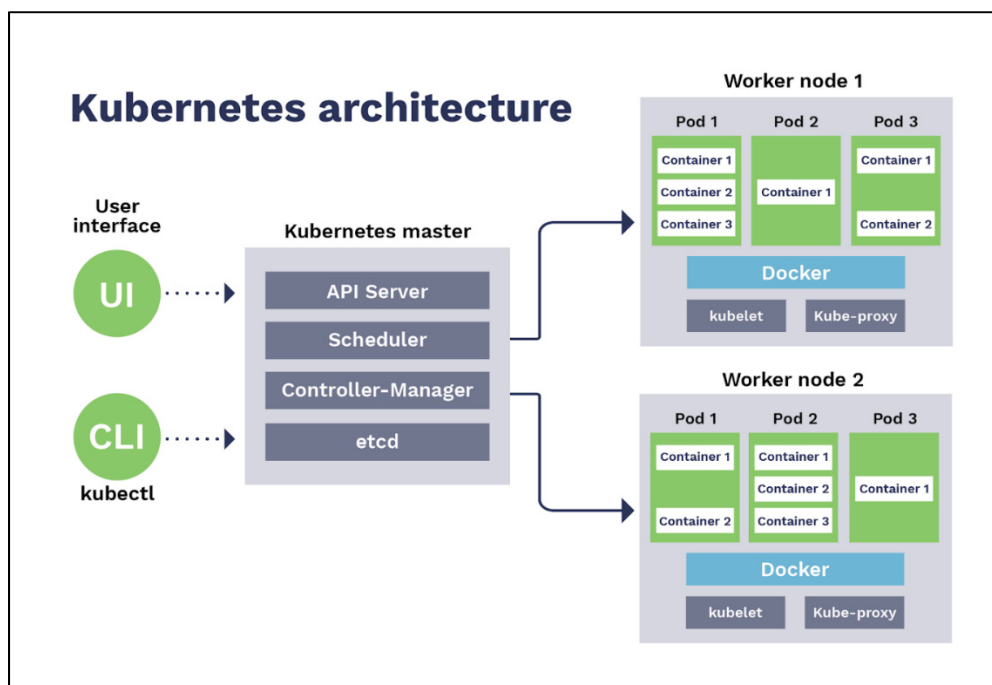


Figure 1.1 Architecture d'un cluster Kubernetes.

Tiré de Spaleta (2019)

### 1.7.3 Sources de métriques dans Kubernetes

L'une des meilleures énumérations des sources de données d'un cluster Kubernetes rédigées par la plateforme epsagnon est comme suit :

Dans Kubernetes, vous pouvez récupérer des métriques au niveau du système à partir de diverses sources prêtes à l'emploi tel que cAdvisor, Metrics Server et Kubernetes API Server. Vous pouvez également récupérer des métriques au niveau de l'application à partir d'intégrations telles que kube-state-metrics et Prometheus Node Exporter.

### **cAdvisor**

Container Advisor est un excellent outil de surveillance qui fournit des métriques au niveau du conteneur et expose l'utilisation des ressources et les données de performances des conteneurs en cours d'exécution. Il fournit un aperçu de l'utilisation du processeur, de l'utilisation de la mémoire et de la réception/transmission réseau des conteneurs en cours d'exécution. cAdvisor est intégré dans le kubelet, vous pouvez donc gratter le kubelet pour obtenir des métriques de conteneur, stocker les données dans un magasin de séries chronologiques persistant comme Prometheus/InfluxDB, puis le visualiser via Grafana.

### **Metrics Server**

Metrics Server est un agrégateur de données d'utilisation des ressources à l'échelle du cluster et collecte des métriques de base telles que l'utilisation du processeur et de la mémoire pour les nœuds, les pods et les conteneurs Kubernetes. Il est utilisé par l'autoscaler de pod horizontal et le tableau de bord Kubernetes lui-même, et les utilisateurs peuvent accéder à ces métriques directement à l'aide de la commande `kubectl top`. Metrics Server remplace Heapster en tant qu'agrégateur de métriques principal dans le cluster, qui a été marqué comme obsolète dans la nouvelle version de Kubernetes.

### **Kubernetes API Server**

L'interface du plan de contrôle Kubernetes est le serveur d'API exposant toutes les fonctionnalités fournies par Kubernetes. C'est comme une passerelle et un hub de communication pour l'ensemble du cluster Kubernetes. Toutes les demandes des utilisateurs passent par le serveur API, où il effectue la validation de la demande du client et interagit avec etcd pour la persistance de l'état du cluster. Tous les composants du cluster communiquent entre eux via le serveur API. Le serveur API est responsable du traitement des opérations API et du stockage des objets API dans un backend de stockage persistant. Il fournit également un certain nombre de métriques critiques du point de vue des opérations de cluster - principalement les métriques de taux de demande, de taux d'erreur et de durée (RED) - pour les ressources Kubernetes.

### **Node Exporter**

Node Exporter est l'exportateur Prometheus pour les métriques du matériel et du système d'exploitation. Il vous permet de surveiller les métriques au niveau du nœud tel que le processeur, la mémoire, l'espace du système de fichiers, le trafic réseau et d'autres métriques de surveillance, que Prometheus supprime d'une instance d'exportateur de nœuds en cours d'exécution. Vous pouvez ensuite visualiser ces métriques dans Grafana.

### **Kube-State-Metrics**

Kube-state-metrics est un agent complémentaire qui écoute le serveur d'API Kubernetes. Il génère des métriques sur l'état des objets Kubernetes à l'intérieur du cluster, tels que les déploiements, les jeux de réplicas, les nœuds et les pods.

Les métriques générées par kube-state-metrics sont différentes des métriques d'utilisation des ressources, qui sont principalement axées davantage sur l'utilisation du processeur, de la mémoire et du réseau. Kube-state-metrics expose des métriques critiques sur l'état de votre cluster Kubernetes :

- Demandes de ressources et limites
  - Nombre d'objets : nœuds, pods, espaces de noms, services, déploiements
  - Nombre de pods dans un état en cours d'exécution/terminé/en échec.
- (Ribenzaft, 2020)

## **1.8 Apprentissage automatique (Machine Learning)**

L'apprentissage automatique, aussi appelé apprentissage machine ou apprentissage artificiels et machine Learning en anglais.

L'apprentissage automatique est un sous-ensemble de l'IA qui se concentre sur les capacités d'apprentissage. Il comprend tous les algorithmes qui utilisent des données pour apprendre à se comporter à l'avenir. La principale différence entre l'IA et l'apprentissage automatique réside dans le fait que l'IA consiste à donner des solutions statiques programmées pour exécuter une tâche qui est utilisée par l'homme, comme résoudre des problèmes de mathématiques, des calculs, jouer aux échecs... Mais ML consiste à faire l'ordinateur capable d'acquérir ses connaissances à partir de l'expérience donnée par certaines données et de fonctionner automatiquement tout en traitant des données invisibles. Les données massives collectées sont l'instrument qui permet à ce modèle d'apprendre pour prédire l'utilisation des ressources. (Ng, 2017)

## **1.9 La relation Kubernetes et NFV : télémétrie**

L'approche de la virtualisation des fonctions réseau (NFV) et l'orchestrateur Kubernetes des conteneurs Docker sont deux technologies très puissantes qui ont apporté beaucoup d'avantages en termes de performance de services réseaux offerts aux utilisateurs. Elles s'appuient toutes les deux sur plusieurs nouveaux concepts et abstractions combinés, ce qui augmente la complexité de leur combinaison en conséquence.

Dans le but de comprendre la relation entre NFV et Kubernetes, Intel a lancé la recherche (Ramakrishnan, 2019) qui a été résumée dans une conférence au titre : ‘ Kubernetes et la virtualisation des fonctions réseau (NFV) : Amis ou Ennemies ? ‘

Les modèles de déploiements NFV sont multiples. Ça peut être des machines virtuelles, des conteneurs, ou un système hybride des deux technologies. L’étude Intel a ciblé le modèle sur conteneurs de type déploiement « Bare Metal ». Le « Bare Metal » signifie une installation d’un système d’exploitation sur aucune couche de virtualisation. Quand il s’agit de déploiement, les défis rencontrés sont souvent technologiques, opérationnels ou organisationnels. Un nombre de ces derniers a été énuméré ensuite regroupé par catégorie comme l’illustre la figure 1.2 ci-dessous.

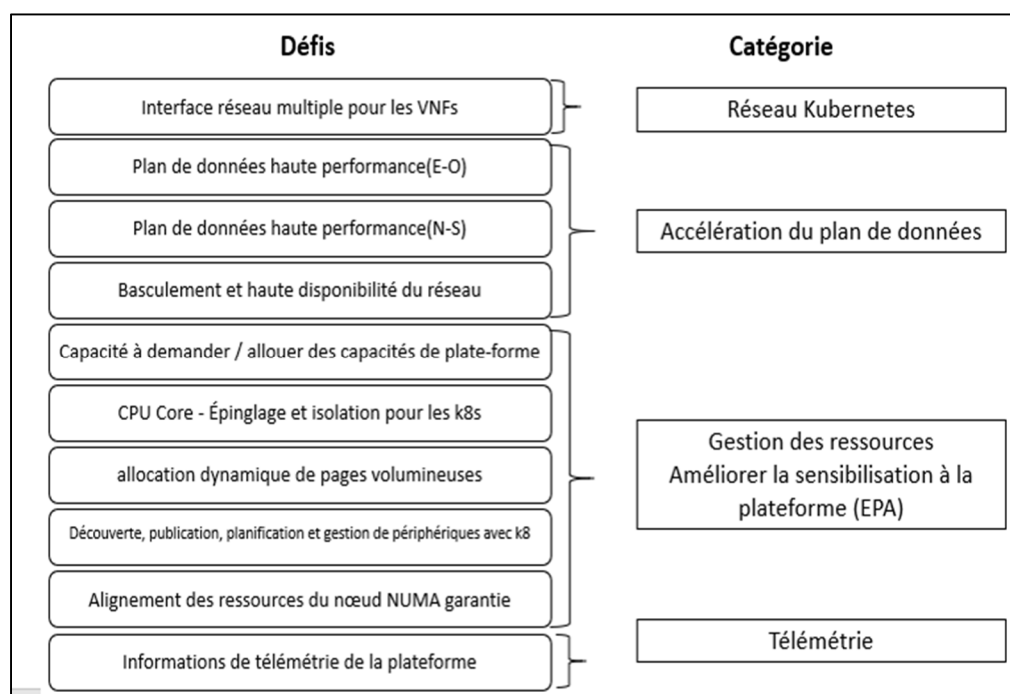


Figure 1.2 Les défis clés d’un déploiement NFV Bare Metal  
Tiré de Ramakrishnan (2019)

À titre d’exemple, le premier problème à relever, de la figure 1.2, vient du fait qu’à la base tout service réseau virtuel a besoin de deux interfaces réseau pour être lié avec le plan de contrôle NFV et le plan de données NFV. Cependant, sur une plateforme Kubernetes, l’unité logique

appelée Pod supporte une seule interface réseau. Pour remédier à cela, un projet Multus CNI du projet (Intel Corporation, 2016) a été créé. C'est un plug-in qui va permettre de connecter plusieurs interfaces réseau sur un Pod et de cette façon le problème multiple interface réseaux est résolu.

Cette même étude de Ramakrishnan (2019) a confirmé l'existence d'un problème de la télémétrie au déploiement NFV sur un environnement Kubernetes. Il s'agit, souvent, d'un manque d'informations liées à la plateforme et aux composantes NFV. Cette recherche a proposé l'intégration du démon collectd (Collectd, 2016) de surveillance qui regroupe des mesures et fournit des mécanismes pour stocker et distribuer les valeurs collectées de différentes manières. Ce sont des statistiques sur le système de la plateforme qu'il sauvegarde pour être ensuite utilisé dans des processus d'analyse de performance et la prédiction des charges futures du système ou ce qu'on appelle la planification de la capacité.

Selon Sridhar, Ranganath, Ilangoan, & Lin (2018, p.2), l'analyse des informations obtenues via collectd est très spécifique à l'application et aux cas d'utilisation. L'analyse peut inclure la création de profils de consommation de cache des charges de travail intéressées, l'utilisation des informations pour définir la stratégie d'allocation de cache, la détection des anomalies, etc.

On conclut que la télémétrie est un défi clé devant un déploiement NFV sur une plateforme Kubernetes. On constate aussi, plusieurs limitations à la proposition d'Intel de (Ramakrishnan, 2019). Premièrement, les tests et les configurations proposés et expérimentés sont tous basés sur la technologie Intel. Ceci implique que les configurations et les solutions proposées peuvent ne pas être valides pour d'autres. Ensuite, Intel a relevé l'ennui de la télémétrie par l'outil collectd qui est limité devant l'analyse des informations liée à la plateforme.

### 1.10 Approches de collecte de données

Lorsqu'il s'agit de collecter des métriques à partir d'une application, il existe deux approches (modèles). La première est la méthode Push ou l'outil de collecte s'attend à recevoir les données. Elle est préférée pour les ensembles de données de séries chronologiques basés sur des événements pour sa précision, car chaque événement est envoyé lorsqu'il est déclenché à la source. Cependant, elle prend plus de temps pour savoir si un service est défectueux, du fait que l'intégrité de l'instance est basée sur l'événement qu'il reçoit.

En revanche, quand un outil de collecte utilise un service pour récupérer les données, il s'agit de la méthode pull. Elle est bonne pour une grande partie de cas d'utilisation, car elle applique la convention en utilisant un langage standard. Les appareils Internet des objets (IoT) et les événements de navigateur sont un défi à ce modèle et extraire les données demande plus d'effort.

Faire un choix entre les deux modèles dépend du cas d'utilisation, mais : « Dans un environnement cloud natif, les entreprises ont tendance à privilégier le modèle pull par rapport au modèle push pour sa simplicité et son évolutivité » (Crosby et al., 2018).

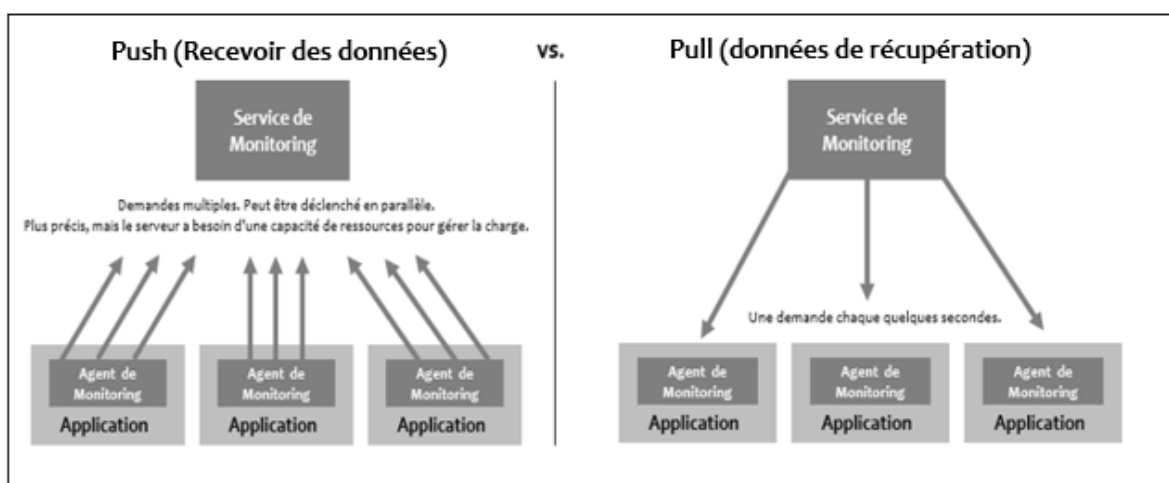


Figure 1.3 Deux méthodes de collecte de données

Tiré de Crosby et al., (2018, p.14)

### 1.11 Outils de collecte de télémétrie

Nous avons sélectionné un groupe d'outils de collecte de données télémétriques les populaires et modernes afin de les comparer selon des critères établis pour servir les objectifs de notre étude comme suit (tableau 1.1) :

Tableau 1.1 Comparaison entre des outils de collecte de métriques

| Critères /Outils                            | ElasticSearch                                                                                                                                                                                                                              | Prometheus                                                                                                                                                                                                                                                        | Télégraf                                                                                                                                                                                                                                                                                                 |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------|-------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----------|-------|------|------|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----------|-------|-----|-----|---|
| Description                                 | Un moteur de recherche et d'analyse distribuée et en open source pour tout type de données structurées et non structurées.                                                                                                                 | Un système de surveillance des systèmes et des services. Il collecte des métriques à partir de cibles configurées à des intervalles donnés, évalue les expressions de règle, affiche les résultats et peut déclencher des alertes si une condition s'avère vraie. | C'est un agent de collecte, de traitement, d'agrégation et d'écriture de métriques. Les objectifs de conception sont d'avoir une empreinte mémoire minimale avec un système de plug-ins afin que les développeurs de la communauté puissent facilement ajouter un support pour la collecte de métriques. |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| Popularité<br>Tiré de Stackshare.<br>(2019) | <div><br/>Elasticsearch</div> <table><tr><td>Stacks</td><td>Followers</td><td>Votes</td></tr><tr><td>17.6K</td><td>12.6K</td><td>1.6K</td></tr></table> | Stacks                                                                                                                                                                                                                                                            | Followers                                                                                                                                                                                                                                                                                                | Votes | 17.6K | 12.6K | 1.6K | <div><br/>Prometheus</div> <table><tr><td>Stacks</td><td>Followers</td><td>Votes</td></tr><tr><td>1.7K</td><td>1.9K</td><td>209</td></tr></table> | Stacks | Followers | Votes | 1.7K | 1.9K | 209 | <div><br/>Telegraf</div> <table><tr><td>Stacks</td><td>Followers</td><td>Votes</td></tr><tr><td>128</td><td>120</td><td>5</td></tr></table> | Stacks | Followers | Votes | 128 | 120 | 5 |
| Stacks                                      | Followers                                                                                                                                                                                                                                  | Votes                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| 17.6K                                       | 12.6K                                                                                                                                                                                                                                      | 1.6K                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| Stacks                                      | Followers                                                                                                                                                                                                                                  | Votes                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| 1.7K                                        | 1.9K                                                                                                                                                                                                                                       | 209                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| Stacks                                      | Followers                                                                                                                                                                                                                                  | Votes                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| 128                                         | 120                                                                                                                                                                                                                                        | 5                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| Configuration                               | Moins facile                                                                                                                                                                                                                               | Moins facile                                                                                                                                                                                                                                                      | Facile                                                                                                                                                                                                                                                                                                   |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
| Coût                                        | Open source/gratuit                                                                                                                                                                                                                        | Open source/gratuit                                                                                                                                                                                                                                               | Open source/gratuit                                                                                                                                                                                                                                                                                      |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |
|                                             |                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                          |       |       |       |      |                                                                                                                                                                                                                                      |        |           |       |      |      |     |                                                                                                                                                                                                                                  |        |           |       |     |     |   |

| Critères /Outils                             | ElasticSearch                                                                                                                                                                               | Prometheus                                                                                       | Télégraf                                                                   |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| Aspect temps réel                            | Oui                                                                                                                                                                                         | Oui                                                                                              | Oui                                                                        |
| Métriques en pourcentage                     | Oui                                                                                                                                                                                         | Oui                                                                                              | Oui                                                                        |
| Approche distribuée                          | Oui                                                                                                                                                                                         | Oui                                                                                              | Oui                                                                        |
| Points forts.<br>Tiré de<br>Biseul, X (2019) | Vitesse, API RESTful, évolutif, indexation<br>Stockage et traitement distribués                                                                                                             | Couverture fonctionnelle complète, basé sur l'extraction (Le Pull) popularité, communauté active | Écosystème riche en plug-ins, profondeur de l'historique                   |
| Limites.<br>Tiré de Biseul,X<br>(2019)       | Peut être difficile avec l'infrastructure sous-jacente, demande une expertise pour faire évoluer avec succès un cluster ElasticSearch et s'assurer qu'il fonctionne de manière transparente | Durée de rétention réduite                                                                       | Passage à la version payante pour répondre à la croissance des traitements |

Choisir un outil parmi ceux en tableau 1.1 dépend du contexte et des exigences de l'entreprise qui souhaite concevoir un système de collecte de télémétrie, et ce tableau aide à la prise de cette décision.

### 1.12 Kubernetes et centre de données NFV

À ce propos, l'article scientifique de Rafael, S et al., (2020, p. 51713) souligne que NFV augmente la fonction, l'allocation et la flexibilité du réseau pour la mise en œuvre de nouveaux services réseau. D'un autre côté, les avantages de l'optimisation de l'allocation des ressources dans le centre de données NFV comprennent la réduction des efforts opérationnels, la réduction des coûts d'investissement et l'automatisation des processus opérationnels. En ce sens, certains

aspects impliqués dans les implémentations actuelles de NFV nécessitent une enquête plus approfondie, tels que l'évaluation des performances et la surveillance NFV.

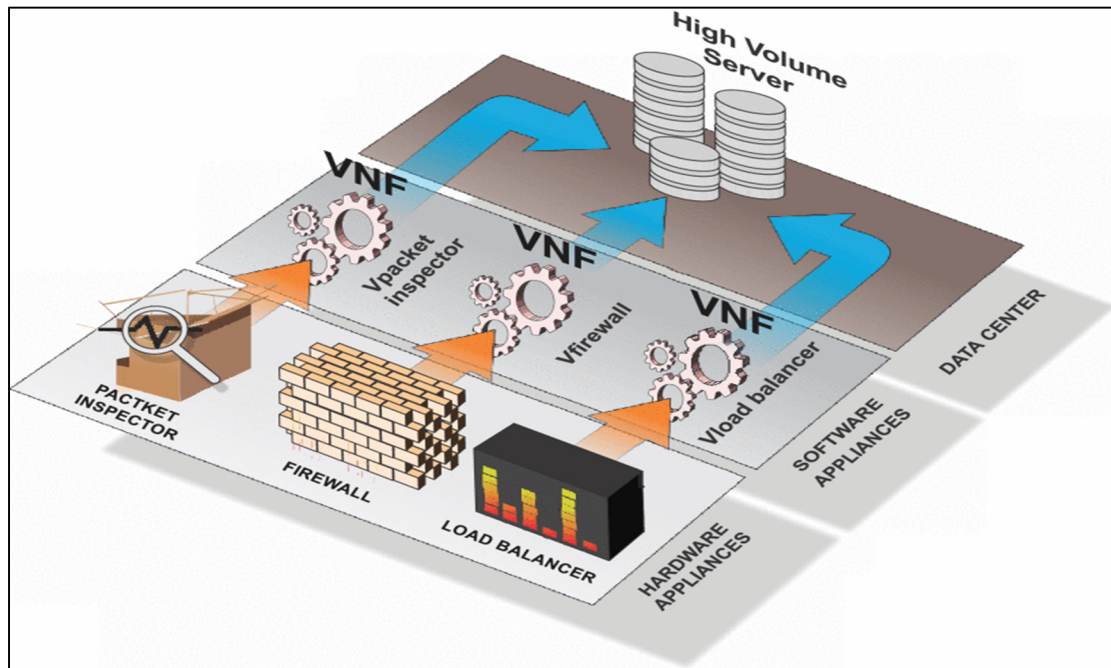


Figure 1.4 Un centre de données NFV

Tiré de Rafael, S et al., (2020, p. 51716)

Les centres de données NFV souffrent de plusieurs problèmes liés aux performances, à la fiabilité, à l'allocation des ressources, au coût, à la gestion et à l'interopérabilité des ressources. Une revue systématique de la littérature (SLR) a été réalisée pour faire une analyse approfondie des applications NFV dans les centres de données et consolider leur intégration. Cette étude (Rafael, S et al., 2020) a analysé et filtré environ 1408 articles, en mettant en évidence sur un total de 65 études pertinentes qui traitent les principales questions de recherche concernant l'architecture NFV. Les sources de ces articles sont les bases de données suivantes : IEEEExplore, ACM Digital Library, SpringerLink, ScienceDirect et Scopus.

Cette SLR a identifié plusieurs problèmes après analyse. L'un des principaux défis pour le déploiement de NFV dans les centres de données est de parvenir à une allocation efficace des ressources des services requis dans les infrastructures réseau. L'allocation des ressources vise

à optimiser le NFV dans le centre de données, en utilisant des algorithmes pour déterminer le meilleur positionnement du VNF ainsi que la planification VNF, entre autres exigences. Sur cette base, l'étude (Rafael, S et al., 2020) a cité une grande partie des aspects suivants des centres de données NFV qui doivent être améliorés : la disponibilité, la fiabilité, la maintenabilité, le temps de réponse, l'économie d'énergie, la tolérance aux pannes, le cout, le délai (de bout en bout), le débit et la performabilité. La performabilité est un terme qui parle de la notion qui permet de combiner performance et disponibilité (une compréhension et une analyse des scénarios de dégradation des performances).

Les résultats de la recherche du SLR menée sont des solutions proposées après études et analyse des défis relevés. Elles sont énumérées de 1 à 9 ci-dessous comme mentionner :

1. L'amélioration du rapport coût / coût de VNF-SFC (SFC : La chaîne des fonctions de service) dans le centre de données augmente la disponibilité du réseau et réduit la complexité du problème de placement de VNF.
2. La réduction du trafic réseau et du gaspillage des ressources informatiques augmente les performances du réseau et réduit la complexité du problème de placement VNF.
3. L'utilisation de VIM dans une architecture multiservice améliore les performances du réseau et du stockage. L'utilisation de VIM réduit également le problème de conception croisée dans l'architecture NFVI.
4. L'amélioration de l'utilisation des ressources réseau et la réduction des économies de consommation réduisent la complexité du problème de placement dans l'architecture NFV-MANO.
5. L'amélioration du contrôle de flux réseau, grâce à des politiques d'attribution de rôle pour les nouveaux scénarios de communication, augmente les performances du réseau et réduit le problème d'allocation de ressources dans l'architecture NFV-MANO
6. L'automatisation de la planification VNF augmente la disponibilité et réduit la dégradation des performances.
7. L'amélioration de l'allocation des ressources en utilisant des mesures préventives de sécurité comme modèles d'affinité minimise l'impact sur la durée de vie du matériel physique.
8. L'amélioration de l'évaluation dans les centres de données NFV avec des mesures de performabilité est essentielle pour obtenir des paramètres de conception optimaux. En fonction des performances de distribution NFV requises, les paramètres incluent, entre autres, la taille du système, la taille du sous-système et les taux de performances de l'état des composants.
9. Enfin, des composants spécifiques, tels que : les processeurs, les modules de mémoire, les disques et les cartes d'interface réseau, peuvent limiter

les performances d'une implémentation VNF. Ainsi, une possible solution de recherche automatique avec un modèle d'apprentissage par renforcement, y compris les composants qui peuvent déterminer la taille de déploiement optimale pour une charge de travail donnée en temps réel. (Rafael, S et al., 2020, p. 51726)

D'un autre côté, il a été souligné que ces centres de données NFV ont bien acquis une grande popularité parmi la communauté universitaire, mais le domaine de recherche en est encore à ses débuts. Il existe plusieurs lacunes en matière de recherche et par la suite des défis et de nouvelles directions de recherches se manifestent pour l'industrie comme pour les universités. Sur cette base, une collaboration de recherche entre les universités et l'industrie est recommandée pour identifier les définitions conceptuelles et opérationnelles précises requises pour intégrer les résultats et générer des connaissances cohérentes sur le centre de données NFV.

## **CHAPITRE 2**

### **SOLUTION PROTOTYPE POUR LA COLLECTE DE DONNÉES TÉLÉMÉTRIQUES SUR KUBERNETES**

#### **2.1 Introduction**

Un prototype d'un système télémétrie Kubernetes est proposé et présenté dans ce chapitre.

L'objectif est de fournir un système de collecte de métriques adaptable aux différents besoins des entreprises selon une démarche théorique proposée pour améliorer la visibilité sur un système NFV. L'idée est de suivre une chaine d'étapes et un processus qui conduit à une architecture prometteuse et pas seulement à la solution elle-même pour avoir cet esprit de conception avec un accent sur l'objectif et une flexibilité au changement des exigences de l'entreprise.

#### **2.2 Objectifs**

Le premier objectif est de fournir une coquille d'une solution d'un système de collecte de données télémétriques avec les caractéristiques suivantes :

1. Utiliser des outils de monitoring Kubernetes pour la collecte de métriques liées aux déploiements NFV ou à la plateforme Kubernetes.
2. Utiliser des outils de collectes capables à cibler la bonne information. Ils existent des diverses sources de données à prendre en considération, mais pas forcément toutes en même temps. Pour cela, les collecteurs devraient être configurables pour déterminer les métriques à chercher et celles à négliger.
3. Adopter l'approche microservices.
4. Assurer une diffusion des données télémétriques en temps réel.
5. Assurer une capacité à prendre en charge une bonne masse de données télémétriques.
6. Intégrer d'un système de prédiction basé sur l'apprentissage automatique

## 7. Visualiser des données collectées.

Ces caractéristiques sont le fruit de l'analyse faite dans notre revue de littérature et surtout en prenant en considération les solutions proposées dans la section 3.6 afin de pallier les principaux défis d'un déploiement NFV dans les centres de données.

### 2.3 Processus suivi

Le processus présenté a comme but d'aider une compagnie à bien identifier et clarifier son objectif d'un système d'observabilité avant de procéder à sa conception et son implémentation. Une collecte de données ne doit pas être aléatoire et le système chargé à accomplir cette fonction est un module exposé à des changements dus à l'évolution des besoins de l'industrie. Par conséquent, il est possible qu'un système de collecte ne soit plus efficace s'il ne fournit pas les informations essentielles à une nouvelle fonctionnalité demandée. Ce processus de la figure 2.1 est aperçu comme une démarche structurée qui donne naissance à un système télémétrique adéquat aux exigences spécifiques d'une compagnie donnée.

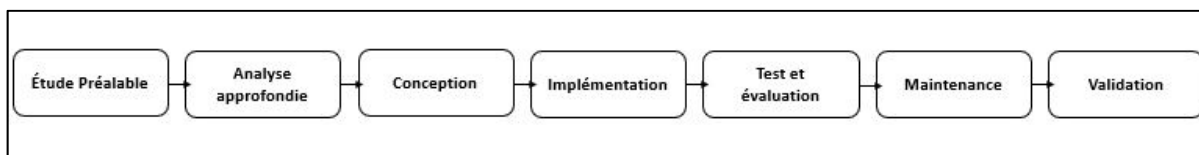


Figure 2.1 Processus de création d'une solution de collecte de données télémétriques d'un déploiement NFV sur une plateforme Kubernetes

#### 2.3.1 Étude préalable :

C'est une phase exploratrice pour éclaircir les conditions générales et l'orientation du travail. Il est important de comprendre le besoin de la compagnie qui demande un système de surveillance et d'observabilité, pour concevoir efficacement dans un temps optimal avec des coûts réduits. Cette phase est importante, vu que les spécifications ne sont pas les mêmes pour toutes les entreprises ni pour les différents types de déploiements NFV. L'étude préalable répond aux

questions : qu'elle est le besoin de l'entreprise ? Que doit-elle observer et surveiller et pourquoi ?

### **2.3.2 Analyse approfondie :**

Il s'agit d'un raffinement des éléments et des besoins de l'entreprise discuté au cours de la phase précédente pour voir de plus près les spécifications du cas d'étude en question et savoir concevoir une solution adéquate et efficace. Ceci veut dire que pour tout système de surveillance avec un module d'observabilité complet nécessite la traduction des besoins soulignés au cours de l'étude préalable en besoins techniques précis. Dans le cas de ce travail concentré sur le module télémétrie, cette analyse approfondie consiste à identifier les trois points suivants : les Métriques ciblées, la bonne méthode de collecte, et les outils efficaces de collecte.

L'identification de ces trois derniers à besoin de prendre en considération de la différence entre théories et technique. Par exemple, pour identifier le bon outil de collecte il faut avoir une perspective d'un ingénieur technique pour éviter beaucoup de défis d'incompatibilités avec d'autres technologies surtout celle de la plateforme Kubernetes. Le travail inclut une étude comparative afin de faire un choix entre une variété d'alternatives technologiques. Ce choix détermine la qualité et l'efficacité du système télémétrique conçu.

Il est aussi intéressant, à cette phase, d'évaluer l'apport et la relation de certaines technologies à la conception de la solution et de décider si leur intégration a de la valeur à ajouter. On parle de la diffusion en temps réel ou du stockage à long terme des données télémétriques à titre d'exemple. Cette démarche doit se faire non seulement en gardant l'accent sur l'objectif, mais aussi avec une vision projeter sur le futur, et comment peut-elle servir l'étape suivante dans un système de surveillance. On peut citer la prédiction de la consommation de ressources comme un exemple concret.

### 2.3.3 Conception (Architecture de la solution)

Après avoir identifié les métriques à surveiller, la méthode de collecte et les outils nécessaires pour la compagnie qui souhaite un système de surveillance efficace a ses déploiements NFV, il est plus facile à concevoir l'architecture du produit et de définir ses modules principaux qui satisfait les objectifs établis par l'entreprise. Cette étape fournit un design à notre solution qui peut être globale ou détaillée selon la situation, et pour le fait que l'étape de son implémentation peut faire face à des défis techniques d'où il faut revoir la composition.

D'une manière générale, l'architecture de notre solution de collecte des données télémétriques est représentée dans la figure 2.2 et comprend, principalement, un module de collecte, un module de diffusion et un module de visualisation. Un module est composé d'un ou de plusieurs outils selon les besoins identifiés dans la phase d'analyse. Ils existent des outils capables d'assurer plusieurs fonctions tels que Prometheus qui est collecteur et diffuseur de données en même temps.

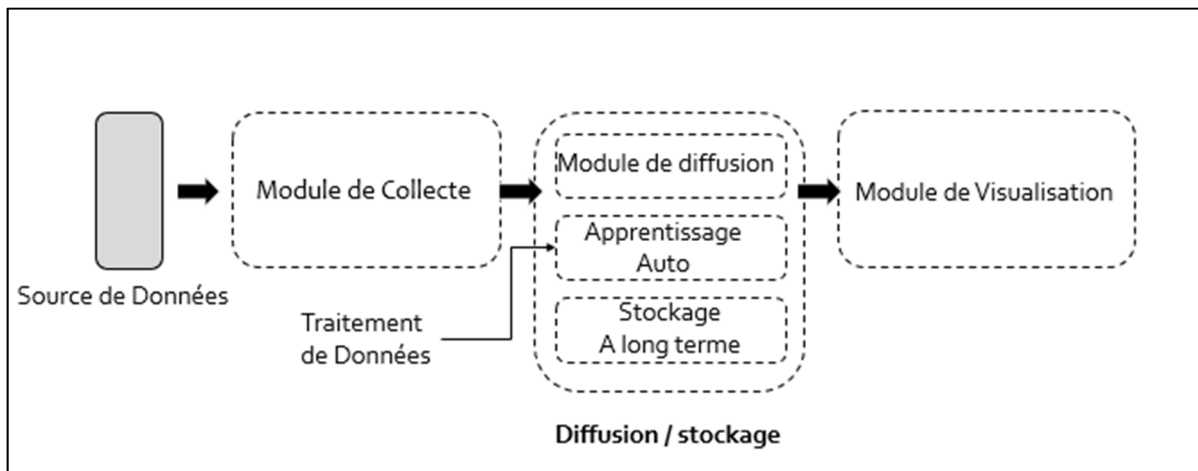


Figure 2.2 Architecture globale du processus de la solution de collectes de données télémétriques d'un déploiement NFV sur Kubernetes

Tel qu'il est présenté dans la figure 2.2, le module diffusion est élargi avec un composant de stockage à long terme des données et un module de traitement et d'analyse des métriques collectées. Ces deux derniers font partie du processus de la collecte et de l'observabilité et elles sont nécessaires pour avoir une visibilité finie.

Les données tirées avec le module de collecte sont diffusées en temps réel via le module de diffusion qui les envoie vers le dernier composant (Module de Visualisation) pour être affiché sur des tableaux de bord. Il est aussi important de visualiser les données collectées pour s'assurer que cette collecte est bien fonctionnelle avant de les envoyer à une autre étape dans le système de surveillance.

Le module de stockage à long terme concerne le stockage des données télémétriques collectées au niveau du module de collecte sur des bases de données qui peuvent être internes ou externes par rapport au cluster Kubernetes.

En nous appuyant sur le travail des chapitres précédents et les travaux antérieurs abordés au cours de notre revue de littérature, nous avons créé la composition d'outils illustrée dans la figure 2.4. Ceci est un développement de l'architecture globale de la Figure 2.2.

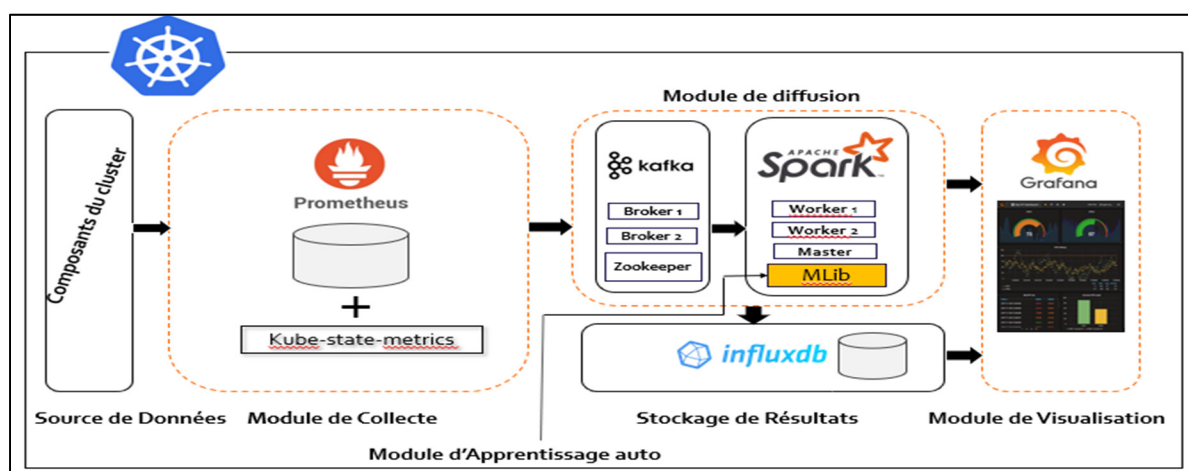


Figure 2.3 Architecture prototype proposé comme solution de collecte de données télémétriques d'un déploiement NFV sur Kubernetes

### **2.3.4 Implémentation**

C'est une étape de transition complète de la théorie vers le réel. Déployer une composition d'outils différents sur une plateforme complexe comme Kubernetes est un chemin plein de défis, et souvent demande des compétences variées pour le déploiement de la solution. Il faut bien trouver une démarche d'implémentation efficace et soulever les erreurs signalées.

Cette phase consiste aussi à réaliser beaucoup de tests de fonctionnalité, des améliorations et des optimisations sur les plans possibles. Par exemple, réduire le temps nécessaire pour implémenter un système d'observabilité ou le temps de collecte des données télémétriques de la plateforme.

### **2.3.5 Test et évaluation**

Avant de valider le système pour son utilisation, plusieurs tests sont indispensables comme suit:

1. Tests de fiabilité : il est important après l'implémentation du système de collecte à vérifier son fonctionnement en déroulant des scénarios représentant des cas d'utilisation; ces tests par scénarios doivent être rejoués après chaque évolution du système.
2. Tests de performance : ça concerne la consommation CPU, l'évolution de l'utilisation mémoire, le nombre de requêtes par seconde, le flux d'entrée/sortie... Le test de performance détermine la bonne exécution en mesurant les temps de réponse sans contexte particulier.

### **2.3.6 Maintenance**

Il peut s'agir d'une maintenance corrective qui consiste à remettre en état le fonctionnement d'un des composants du système. Ainsi, ce type de correction peut être nécessaire si après un

audit, il est noté que la solution ne répond pas à un des objectifs fixés au début de l'étude ou à la suite d'un nouveau besoin de l'entreprise (un besoin à de nouvelles métriques).

### **2.3.7 Validation**

La validation est l'approbation à déployer et faire fonctionner la solution dans l'environnement de production, en s'appuyant sur le résultat des tests de l'étape précédente. Le taux de réussite des tests acceptés pour prendre cette décision de validation revient à chaque compagnie.

## **2.4 Choix des outils**

Notre conception de solution est théorique et globale vu qu'elle ne concerne pas encore un cas précis d'étude ou une entreprise donnée. Cependant, il est possible de créer un prototype basé sur la comparaison d'outils et les alternatives technologiques ainsi que les objectifs fixés dans ce chapitre. Ensuite, nous transférant cette théorie vers le réel en l'étudiant sur une entreprise concrète et ceci fera l'objet du chapitre suivant. Nos choix comme le montre la figure 2.3 sont comme suit :

### **2.4.1 Module de collecte**

Le module de collecte comportera l'outil Prometheus et le composant kube-state-metrics. Notre choix est basé sur notre revue de littérature et la comparaison entre outils et approches de collecte dans la section 3.3 et la section 3.4 (comparaison de Prometheus avec d'autres outils collecteurs et de kube-state-metrics avec Metrics-server).

Notre collecteur Prometheus est un outil de surveillance libre avec un modèle de données dimensionnelles, un langage de requête flexible, une base de données de séries chronologiques efficace. Nous avons choisi de le configurer pour gratter les métriques du Kube-state-metrics aussi. Ce dernier se comporte comme un service qui parle au serveur d'API Kubernetes pour obtenir toutes les informations sur la santé des composants individuels et sur la santé des divers objets à l'intérieur du cluster, tels que les déploiements, les nœuds et les pods.

Nous n'avons pas choisi Elasticsearch malgré ces grands avantages (tableau 1.1). Cela revient à la difficulté à l'adapter à des architectures sous-jacentes et l'expertise qui demande pour le faire évoluer avec succès et s'assurer qu'il fonctionne de manière transparente. Et bien évidemment nous souhaitons un outil configurable et intégrable avec d'autres modules et architectures plus complexes pour concevoir un système de collecte avec le minimum de problèmes d'incompatibilité et de fonctionnalité.

#### 2.4.2 Module de diffusion et outils de traitement

Nous avons choisi Kafka parmi deux outils de diffusion populaires et modernes. Notre choix adhère à la comparaison prise de l'étude (Khelifi, 2019).

Tableau 2.1 Comparaison entre des outils de diffusion et messagerie (RabbitMQ, Kafka)  
Tiré de Khelifi (2019, p.37)

| Outil                    | Description                                                                                                                                                                                                                                                                                                                                                 | Critères                                                                                                            |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Kafka                    | Kafka est une plateforme de streaming (diffusion) distribuée. Il permet aux systèmes qui génèrent des données (appelés producteurs) de conserver leurs données en temps réel dans une rubrique Apache Kafka. Tout sujet peut alors être lu par n'importe quel nombre des systèmes qui ont besoin de ces données en temps réel (appelés consommateurs).      | Diffusion massive de messages  |
|                          |                                                                                                                                                                                                                                                                                                                                                             | Évolutivité                    |
|                          |                                                                                                                                                                                                                                                                                                                                                             | Mode distribué                 |
|                          |                                                                                                                                                                                                                                                                                                                                                             | Plusieurs consommateurs        |
| RabbitMQ (Pote, S. 2016) | RabbitMQ est un outil de courtage de messages traditionnel. Il peut être déployé dans des configurations distribuées et fédérées pour répondre aux exigences de haute disponibilité et à grande échelle. Mais RabbitMQ ne s'est pas adapté à la création d'une application de microservice. Le problème rencontré lors de la création d'un tel système avec | Diffusion massive de messages  |
|                          |                                                                                                                                                                                                                                                                                                                                                             | Évolutivité                    |
|                          |                                                                                                                                                                                                                                                                                                                                                             | Mode distribué                 |

| Outil | Description                                                                 | Critères                                                   |
|-------|-----------------------------------------------------------------------------|------------------------------------------------------------|
|       | RabbitMQ est la suppression des données après consommation dans ce système. | Plusieurs consommateurs <span style="color: red;">✗</span> |

Pour le composant du traitement faisant partie de notre module de diffusion, nous choisissons Apache Spark en nous basant sur des critères énumérés dans le tableau 2.2, appuyé par notre propre expérience d'utilisation. La capacité à diffuser, à traiter et à modifier les données en temps ainsi que la présence des bibliothèques d'apprentissage automatique est une raison principale à notre choix.

Tableau 2.2 Comparaison entre deux outils de traitement de données (Spark et Hadoop)  
Tiré de Pote, S (2016)

| Critère                   | Spark                                                                                                                     | Hadoop                                                                          |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| Stockage de données       | En mémoire                                                                                                                | Sur le disque                                                                   |
| Tolérance aux pannes      | Modèle de stockage de données Sparks, ensembles de données distribuées résilientes (RDD) garantit la tolérance aux pannes | Utilise la réplication pour atteindre la tolérance aux pannes                   |
| Vitesse                   | Plus rapide grâce au calcul de la mémoire                                                                                 | Relativement plus lent que Spark                                                |
| Diffusion de données      | Peut être utilisé pour traiter et modifier des données en temps réel avec Spark Streaming                                 | Peut traiter un lot de données stockées                                         |
| Apprentissage automatique | Possède son propre ensemble de bibliothèques d'apprentissage automatique 'MLib'                                           | Nécessite une interface avec une autre bibliothèque d'apprentissage automatique |

### 2.4.3 Module de visualisation

Le tableau 2.3 suivant est une comparaison entre deux outils de visualisation moderne et populaire. Grafana est notre choix vu sa facilité d'utilisation avec notre collecteur Prometheus. Le choix s'est aussi appuyé sur des critères comme suit :

Tableau 2.3 Comparaison entre deux outils de visualisation libres (Grafana, Kibana)  
Tiré de Biseul, X (2019)

|              | Grafana                                                                                             | Kibana                                                                                        |
|--------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Description  | Outil de visualisation                                                                              | Outil de visualisation                                                                        |
| Écosystème   | Intégration à Prometheus, InfluxDB et aux services de monitoring des principaux cloud providers     | Stack réunissant Elasticsearch, Logstash, Kibana et Beats                                     |
| Points forts | Facilité de prise en main, nombreux plug-ins, Tableaux de bord prêts à l'emploi, version entreprise | L'association avec Elasticsearch, permet de traiter une grosse volumétrie de fichiers de logs |
| Limites      | Difficulté dans les premiers temps à configurer les sources de données                              | Prise en main peu intuitive                                                                   |

### 2.4.4 Module de stockage

InfluxDB et TimescaleDB sont deux bases de données très populaires et utilisées. Nous avons fait une comparaison et nous avons choisi InfluxDB pour sa simplicité à être intégré avec Prometheus précédemment choisi comme collecteur de métriques au niveau de la solution à concevoir. Si nous avions choisi TimescaleDB, ça aurait été un très bon outil aussi avec un peu plus d'effort en configuration et d'expertise avec la base de données PostgreSQL et l'adaptateur qui le connecte à Prometheus.

Tableau 2.4 Comparaison entre deux bases de données de séries chronologiques  
(InfluxDB et TimescaleDB)

|                                  | InfluxDB                                                                                                                                         | TimescaleDB                                                                                                                                                                                           |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description                      | Une base de données de séries chronologiques libre conçue pour gérer des charges d'écriture et de requêtes élevées. Prends en charge "SQL-like". | Une base de données de séries chronologiques libre optimisée pour l'ingestion rapide et les requêtes complexes prenant en charge SQL complet.                                                         |
| Qualités                         | Stockage de données évolutif, durable et à long terme                                                                                            | Stockage de données évolutif, durable et à long terme. SQL est son point fort vu la tendance actuelle à ce langage, ainsi une requête SQL complexe et beaucoup plus optimale qu'une requête SQL-like. |
| Temps réel                       | Oui                                                                                                                                              | Oui                                                                                                                                                                                                   |
| Intégration avec Prometheus      | Facile avec une simple configuration Prometheus proposée par InfluxDB (Influxdata. s.d)                                                          | Moins facile que InfluxDB et demande deux composants : Prometheus PostgreSQL Adapter et une base de données PostgreSQL avec deux extensions (pg_prometheus, timescaledb)                              |
| Support Grafana et visualisation | Possible avec une Source de données Prometheus                                                                                                   | Possible avec une source de données Prometheus (qui se connecte ensuite à PostgreSQL)                                                                                                                 |

## **2.5 Le bon outil pour la bonne information**

Dans ce contexte, la télémétrie représente ces données et ces informations collectées de la plateforme conteneurisée d'un déploiement NFV donnée. Cette notion qui ne se repose plus sur l'approche de collecte traditionnelle à partir de sondes physiques a fait référence à de nouvelles métriques. Ces dernières doivent être prises en considération pour une meilleure compréhension de la relation NFV et Kubernetes, pour une meilleure visibilité, et pour une efficacité de l'observation et de la surveillance.

De nouvelles métriques impliquent de nouvelles approches de collecte. La question qui se pose dans cette section est : l'approche ou la manière à collecter, l'outil de collecte et la source de collecte ont-ils un impact sur le système afin d'aboutir à la bonne information ?!

Ce qui suit est une exploitation d'autres outils collecteurs de données télémétriques innovants sur trois cas d'études divers.

### **2.5.1 Expérimentation**

Nous avons sélectionné quatre outils avec la capacité à collecter des données télémétriques de la plateforme Kubernetes.

#### **2.5.1.1 Environnement de déploiement**

Nous avons deux environnements à notre disposition : un cluster Kubernetes via Minikube, où nous avons un nœud master et un nœud worker, et un cluster Kubernetes d'un nœud master et huit nœuds worker déployés sur un mini-centre de données du matériel Kontron (section 5.2.1). Le choix de Minikube est éliminé pour la limitation en ressources qui peut être un obstacle devant l'expérimentation qui doit cibler le point source de la donnée télémétrique et l'outil de collecte.

### 2.5.1.2 Les expérimentations et les outils de collectes

Quatre expérimentations ont été réalisées où chacune utilise un outil ou un groupe d'outils pour la récupération des informations télémétriques. Grafana est l'outil de visualisation pour toutes nos épreuves. Il suffit de l'implémenter et d'ajouter une source de données au niveau de ses paramètres de configuration.

1. **Expérimentation 1** (Prometheus) : Il s'agit d'un déploiement du collecteur Prometheus sur le cluster Kubernetes avec une configuration basique. Cette dernière consiste à ne pas configurer d'autres points de collecte pour Prometheus autre que ceux en défaut globalement : kubelet, CAdvisor, API-server. Pour la visualisation, la source de données configurée pour l'outil Grafana est Prometheus.
2. **Expérimentation 2** (Prometheus, kube-state-metrics) : Dans ce cas, nous allons configurer Prometheus de telle façon qu'il récupère d'autres métriques à partir de Kube-state-metrics. Nous nous sommes inspirés et appuyés sur le projet libre (Thakur, 2019) pour déployer kube-state-metrics et ceci inclut les points suivants :
  - A. Création d'un compte de service, d'un rôle du cluster « cluster-role » et de la liaison de rôle de cluster « cluster-role-binding » nécessaire pour Kube-state-metrics (les autorisations d'accès).
  - B. Déploiement de Kube-state-metrics avec une réplique en cours d'exécution.
  - C. Création d'un service Kube-state-metrics où les métriques seront exportées sur le point de terminaison HTTP (les métriques sur le port d'écoute 80 par défaut).

Prometheus est aussi la source d'informations télémétriques pour Grafana dans cette épreuve.

3. **Expérimentation 3** (Prometheus, kube-state-metrics, Apache Spark) : L'architecture illustrée dans la figure 2.4 a été déployée et utilisée comme un système de récupération de

données télémétriques en temps réel sans utiliser le module d'apprentissage automatique MLlib d'Apache Spark. Les composants sont le collecteur Prometheus configuré avec Kube-state-metrics pour la collecte, Apache Spark pour la diffusion en temps réel, la base de données libre InfluxDB pour stocker les résultats de diffusion, et Grafana pour la visualisation. Pour ce cas, nous avons configuré deux points sources de données télémétriques sur Grafana, Prometheus et InfluxDB.

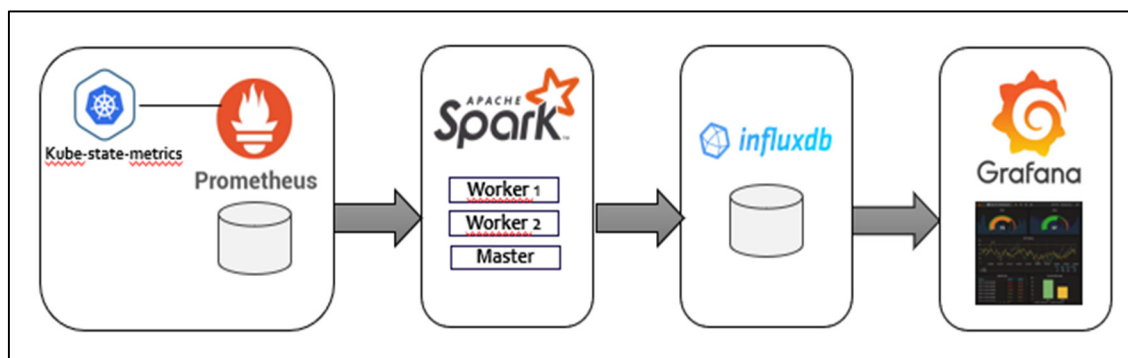


Figure 2.4 Architecture du système de collecte de métriques implémenté sur Kubernetes

4. **Expérimentation 4** (Prometheus, kube-state-metrics, istio) : Cette expérimentation consiste à déployer Prometheus et Kube-state-metrics, ainsi que Istio sur notre cluster Kubernetes. À ce niveau, le package istio déployé inclut Prometheus. Et pour cela, nous allons avoir deux points de collecte : Prometheus-1 qu'on a déployé nous-mêmes indépendamment et avant l'installation de Istio, et Prometheus-2 qui vient avec ce dernier. Il est facile de les distinguer vu que istio crée son propre espace de noms « Namespaces » qui comporte tous les composants de son package y compris Prometheus. Il est aussi possible de les distinguer par les noms des Pods qui les englobent.

## 2.5.2 Les cas d'études

Plusieurs types de déploiement ont été choisis pour exploiter les quatre groupes d'outils de collecte télémétrique.

1. **Cas d'étude 1** : Déployer une image d'un serveur web libre Nginx.
2. **Cas d'étude 2** : Déployer une application typique microservice appelé BookInfo tiré de (Istio, s.d) sur notre cluster Kubernetes. L'application est un catalogue d'une librairie en ligne qui fournit des informations sur ses livres. Elle est divisée en quatre microservices distincts :
  - A. « Produce Page » : appelle les détails et examine les microservices pour remplir la page.
  - B. « Details » : contiens des informations sur le livre.
  - C. « Reviews » : contiens des critiques de livres. Il appelle également le microservice « ratings ».
  - D. « ratings » : contiens des informations sur le classement des livres qui accompagnent une critique de livre.

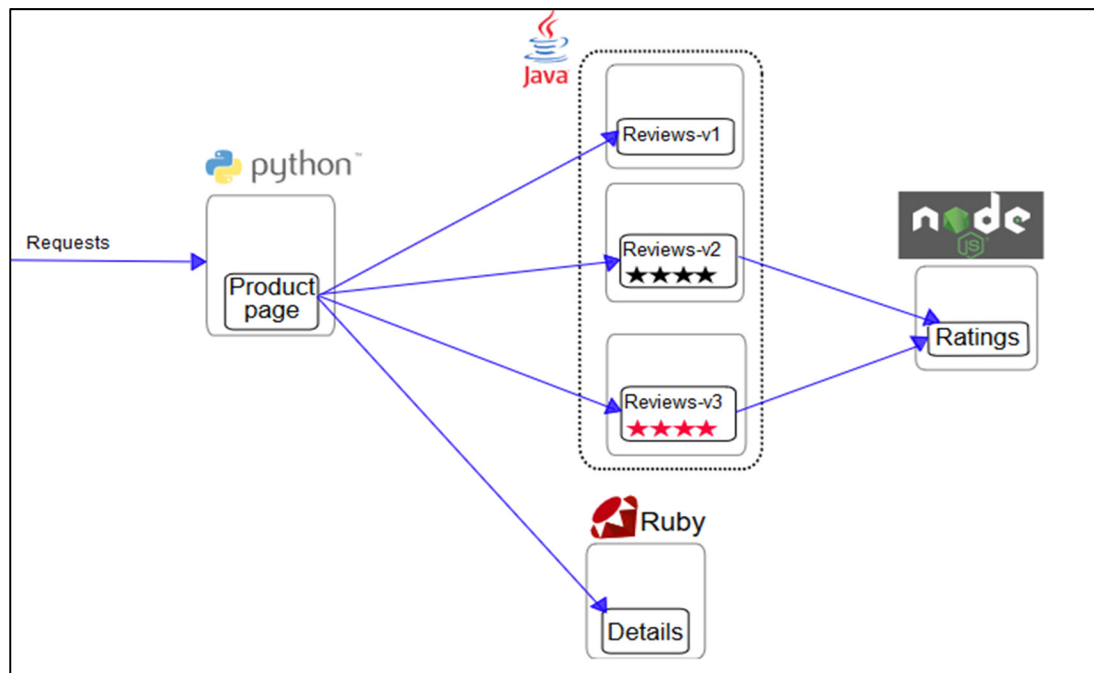


Figure 2.5 Architecture de l'application BookInfo  
Tiré de Istio (s.d)

- 3. Cas d'étude 3 :** Il s'agit de déployer la même application BookInfo, mais une version développée avec istio prise de (Istio, s.d). Ceci veut dire que istio va injecter des points de communication Envoy près de chaque microservice comme illustré dans la figure 2.5.

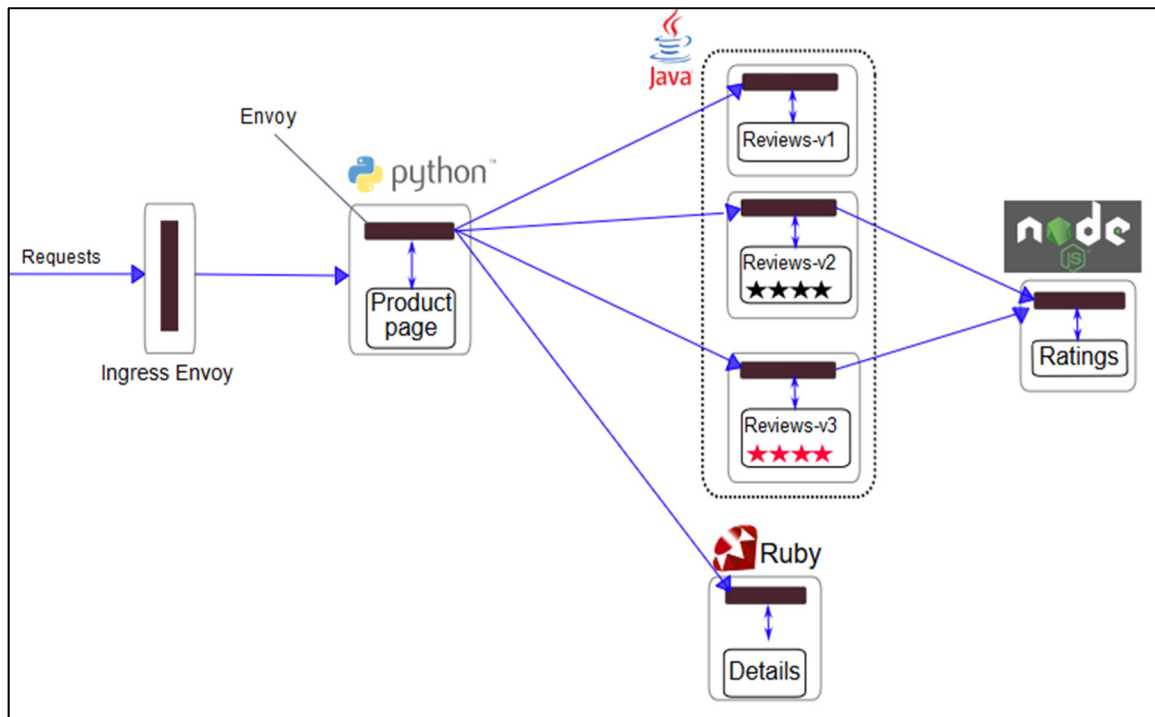


Figure 2.6 Architecture de l'application BookInfo avec istio  
Tiré de Istio (s.d)

### 2.5.3 Le choix des cas d'études :

Certes que NFV est une notion qui apporte beaucoup d'avantages, néanmoins elle se caractérise par une complexité supplémentaire. Ce n'est pas tout, le fait que NFV est, de plus en plus, dans une combinaison avec d'autres technologies tel que les microservices, le maillage des microservices, l'orchestrateur Kubernetes et bien d'autres, les défis s'empilent.

En nous basant sur cette réalité, nous avons choisi l'étude de cas n° 2 qui présente une architecture microservices que nous pouvons considérer comme un service VNF. Les microservices sont très utilisés pour créer des services NFV et de simplifier ses grands systèmes logiciels complexes en les décomposant en sous-composants et en les répartissant sur de nombreux serveurs informatiques ou dans le cloud.

Le troisième cas d'étude est ce même service du cas d'étude antérieure avec l'intégration de Istio. Istio est une plateforme ouverte qui fournit un moyen uniforme de connecter, gérer et sécuriser les microservices. Il est capable de prendre en charge la gestion des flux de trafic entre les microservices, l'application des politiques d'accès et l'agrégation des données de télémétrie, sans modifications du code du microservice en question.

Le premier cas d'étude est un exemple basique, qui est un déploiement d'une image d'un serveur web libre Nginx, servant comme repère ou une étape évidente pour tester nos outils de collectes télémétriques.

#### **2.5.4 Le choix des outils**

Prometheus est un outil de surveillance libre avec un modèle de données dimensionnelles, un langage de requête flexible, une base de données de séries chronologiques efficace et une approche d'alerte moderne comme définit par Prometheus (2014). Ce collecteur de métriques est très populaire pour les avantages et la flexibilité qu'il offre, et le choisir pour expérimenter semblait très évident. 'Robust Perception' est une compagnie spécialisée qui fournit conseil, assistance et services autour du système de surveillance Prometheus, témoigne que cet outil est très efficace.

La tâche de tout instrumenter signifie beaucoup de données et Prometheus est la meilleure classe pour une efficacité de stockage sans perte (3,5 octets par point de données), sachant qu'un seul serveur peut générer des millions de métriques et des centaines de milliers de points de données par seconde selon Brazil (2015). Prometheus présente d'autres principales caractéristiques comme la simplicité et son langage de requête puissant appelé PromQL.

Metrics-server est un composant qui peut être déployé au niveau d'un cluster kubernetes avec quelques prérequis. Il nous permet d'interroger les métriques de ressources en utilisant kubectl directement à partir de la ligne de commande. Cependant, le composant kube-state-metrics fournit des informations du cluster supplémentaires que Metrics Server ne fournit pas. L'industrie DataDog appuie ce fait :

Metrics Server expose des statistiques sur l'utilisation des ressources des objets Kubernetes, tandis que kube-state-metrics écoute l'API Kubernetes et génère des métriques sur l'état des objets Kubernetes : état du nœud, capacité du nœud (CPU et mémoire), nombre souhaité / disponible / réplicas non disponibles / mis à jour par déploiement, état du pod (par exemple, en attente, en cours d'exécution, prêt), etc. Les documents kube-state-metrics détaillent toutes les métriques disponibles une fois kube-state-metrics déployé. (Sapnaro & Mathson, 2016)

Et pour notre expérimentation, nous avons choisi kube-state-metrics pour son avantage en télémétrie.

Apache Spark est un moteur d'analyse unifié pour le traitement de données à grande échelle.

Une conférence scientifique résume :

Spark cache les détails du traitement distribué derrière une API pratique. Avec Spark, les chercheurs peuvent cartographier leurs sources de données dans des trames de données de liste immuable et les transformer à l'aide d'une API déclarative basée sur des primitives de programmation fonctionnelles (fonctions d'ordre supérieur telles que 'map'). En arrière-plan, Spark partitionne les données d'origine, distribue les partitions à un cluster de machines, optimise les calculs fournis par l'utilisateur de manière à minimiser le mouvement des données et les applique en parallèle. (Gousios, 2018, p.542)

En raison de son API pratique et de sa vitesse d'exécution brute, Spark a pris d'assaut le monde des mégadonnées depuis son introduction en 2010 selon Matei, Z. (2010). Il est rapide, flexible et convivial pour les développeurs. Cette tendance de l'industrie envers son utilisation fait de lui un bon choix pour notre expérimentation.

Istio est une plateforme libre de maillage de service « Service Mesh » qui peut connecter, surveiller et sécuriser des microservices. Ces microservices peuvent être déployés sur site ou dans

le cloud, au sein de plates-formes d'orchestration, tels que Kubernetes, dans des services exécutés sur des machines virtuelles, etc. Istio a des fonctions très importantes qui permettent et facilite l'exécution des architectures de microservices distribuées telles que les architectures NFV. Ces fonctions sont la gestion du trafic (routage, contrôle de flux...), la sécurité (permet de gérer l'authentification, l'autorisation et le chiffrement des communications entre les services à grande échelle.), et l'observabilité (fonctions de suivi, de surveillance et de journalisation). La compagnie Twistlock publie en 2018 que Istio offre une visibilité sur la communication réseau, mais la façon dont cela se fait est ce qui est unique et différent des outils de mise en réseau ou de surveillance de réseau traditionnels. Istio propose des modules complémentaires prêts à l'emploi pour les outils de surveillance tels que Prometheus, Grafana et Zipkin. Une fois configuré, il permet l'observabilité dans les microservices, et la meilleure partie est qu'il le fait sans avoir à apporter de modifications aux microservices eux-mêmes. Et pour cette capacité à fournir une visibilité sur les performances de tous les services, nous sélectionnant Istio pour cette expérimentation.

### 2.5.5 Le choix des métriques de comparaison

En global, les deux métriques sélectionnées pour réaliser une comparaison sont la consommation CPU d'un Pod et la consommation CPU d'un conteneur. Cependant, le choix du conteneur ou du Pod à cibler va dépendre de chaque cas d'étude vu que nous avons des déploiements différents comme ce qui suit :

1. **Cas d'étude 1** : nous avons un seul conteneur nginx dans un seul Pod et leur consommation CPU est notre cible.
2. **Cas d'étude 2** : nous avons choisi de calculer la consommation CPU en « core » du Pod « details-pod » et la consommation CPU en « core » du conteneur « details ». Ce déploiement est d'une architecture microservices. Après déploiement, notre cluster Kubernetes a automatiquement déployé le conteneur « details » avec un réplica seuls et sur le même Pod (il n'existe pas d'autres conteneurs autres que « details » et son réplica).

3. **Cas d'étude 3** : le déploiement est un exemple microservices exécuté avec istio. Par conséquent, des side-cars Envoy sont injectés le long de chaque service. Nous avons sélectionné le Pod « details-pod » et le conteneur « details ». Le Pod « details-pod » englobe deux conteneurs (le conteneur Envoy et le conteneur « details »)

Il est important de savoir qu'il faut avoir une certaine stabilité du cluster et du déploiement avant de noter la valeur de notre métrique à l'instant « j ». Ce « j » exprime le moment où le déploiement a complété l'étape de création et n'a subi aucune requête ou action pour assurer que nos valeurs ont les mêmes circonstances. Ceci est très nécessaire pour une comparaison fiable.

## 2.5.6 Résultats

### 1. Cas d'étude1 (Nginx) :

Tableau 2.5 Consommation CPU du conteneur et du Pod en core avec plusieurs groupes d'outils (Cas d'étude 1)

| Outil(s)                                     | CPU du Pod | CPU du Conteneur |
|----------------------------------------------|------------|------------------|
| Prometheus                                   | X          | 2 m              |
| Prometheus, Kube-state-metrics               | 2 m        | 2 m              |
| Prometheus, kube-state-metrics, Apache Spark | 2 m        | 2m               |
| Prometheus-1, kube-state-metrics, Istio      | 2m         | 2 m              |
| Prometheus-2, kube-state-metrics, Istio      | 2 m        | 2 m              |

### 2. Cas d'étude 2 (BookInfo) :

Les métriques choisies sont :

- A. La consommation CPU en « core » du Pod « details-pod ».
- B. La consommation CPU en « core » du conteneur « details ».



Figure 2.7 Les entités Pod et conteneur « details » ciblés dans le cas d'étude 1

Tableau 2.6 Consommation CPU du conteneur et du Pod avec plusieurs groupes d'outils (Cas d'étude 2)

| Outil(s)                                        | CPU du Pod | CPU du Conteneur |
|-------------------------------------------------|------------|------------------|
| Prometheus                                      | X          | 10 m             |
| Prometheus,<br>Kube-state-metrics               | 14 m       | 10 m             |
| Prometheus, kube-state-metrics,<br>Apache Spark | 14 m       | 12m              |
| Prometheus-1, kube-state-metrics,<br>Istio      | 14m        | 10 m             |
| Prometheus-2, kube-state-metrics,<br>Istio      | 14 m       | 10 m             |

### 3. Cas d'étude 3 (BookInfo avec Istio) :

Les métriques choisies sont :

- A. La consommation CPU en « core » du Pod « details-v1 ».
- B. La consommation CPU en « core » du conteneur « details ».



Figure 2.8 Les entités Pod et conteneur « details » ciblés dans le cas d’étude 2

Tableau 2.7 Consommation CPU du conteneur et du Pod avec plusieurs groupes d’outils (Cas d’étude 3)

| Outil(s)                                       | CPU du Pod | CPU du Conteneur |
|------------------------------------------------|------------|------------------|
| Prometheus                                     | X          | X                |
| Prometheus, Kube-state-metrics                 | 25 m       | X                |
| Prometheus-1, kube-state-metrics, Apache Spark | 25m        | X                |
| Prometheus-1, kube-state-metrics, Istio        | 25m        | X                |
| Prometheus-2, kube-state-metrics, Istio        | 25 m       | 10 m             |

Pour les trois cas d’études, le symbole X veut dire que la donnée est absente et que nous n’arrivons pas à la voir sur Dashboard Grafana. Dans ce cas, et en ce qui concerne le CPU Pod, nous vérifions la bonne configuration Grafana et Prometheus et si elle est correctement établie, nous interrogeons Kubectl par la commande « kubectl top pod ». Chaque X des tableaux précédents a eu le résultat suivant : Affichage d’un message d’erreur « pas de Métriques pour le Pod... » et il affiche le Pod en question.

Comme mentionné sur le tableau 2.7, nous avons noté le troisième groupe d'outils « Prometheus-1, kube-state-metrics, Apache Spark » au lieu de « Prometheus, kube-state-metrics, Apache Spark » pour montrer que le collecteur de donnée Prometheus intégrer avec Apache Spark n'est pas celui déployé avec Istio.

### 2.5.7 Discussion

Les résultats obtenus dans les trois cas d'études montrent qu'avec une configuration basique de Prometheus, il n'est pas possible de récupérer les informations sur la consommation de CPU d'un Pod.

Kube-state-metrics est définis comme un service qui parle au serveur d'API Kubernetes pour obtenir tous les détails sur tous les objets API. Il ne se concentre pas sur la santé des composants individuels, mais plutôt sur la santé des divers objets à l'intérieur du cluster, tels que les déploiements, les nœuds et les pods. Le résultat des trois cas d'études affirme qu'ajouter ce collecteur a Prometheus a permis d'avoir la métrique consommation CPU du pod.

Le groupe de « Prometheus-1, kube-state-metrics, Istio » a été capable de récupérer les données souhaitées sauf dans le cas du déploiement de l'application BookInfo avec Istio (troisième case d'étude) ou tous les micro-services sont fournis avec un sidecar envoy. En revanche, le groupe « Prometheus-2, kube-state-metrics, Istio » en a réussi.

En revenant à la documentation officielle de Istio (2018), nous comprenons que l'outil Prometheus fourni et déployé automatiquement en déployant Istio à une configuration autre que le basique que nous avons déployé nous-mêmes. Sa préconfiguration fournit un mécanisme de stockage persistant et d'interrogation des métriques Istio. Une application déployée avec Istio a d'autres composants comme les sidecars envoy et ceci implique l'existence de nouvelles métriques appelées métriques Istio. Ces dernières n'ont pas été récupérables par d'autres outils, et ceci est clairement du à l'architecture du Pod qui contient le conteneur du microservice et le

sidecar qui se charge de certaines tâches dont la télémétrie. La télémétrie est primordiale pour que le plan de contrôle Istio puisse fonctionner.

Le troisième groupe d'outils « Prometheus, Kube-state-metrics et Apache Spark » nous a donné approximativement les mêmes valeurs des deux métriques en question en comparaison avec les autres groupes d'outils, mis à part Prometheus à la configuration basique. Le résultat du tableau 2.7, montre qu'il n'est pas possible de récupérer le CPU du conteneur « details » dans le cas d'une application déployée avec Istio et avec Prometheus sans préconfiguration Istio. En se basant encore sur la documentation (Istio. 2018), pour comprendre la spécificité de ses applications, nous déduisant qu'Apache Spark est incapable de récupérer les métriques Istio seul avec un collecteur de données non lié à ses sidecars envoy responsable de cette télémétrie.

### 2.5.8 Conclusion

Ils existent plusieurs technologies sur le marché pour relever le défi de la télémétrie. Cette étude nous a montré l'importance de bien choisir notre outil de collecte de métriques qui dépend de l'information ciblée et du type du déploiement et on ne pas dire qu'il existe un outil plus fiable que l'autre sans le mettre dans un cadre de spécifications précises.

Nous concluons que diversifier les points de collecte de métriques influence clairement la visibilité sur le système. Et surtout, quand il s'agit des déploiements ou des contextes complexes tels qu'un environnement NFV qu'on peut décrire comme systèmes récursifs, constitués de services complexes en plus des services élémentaires.

Un cluster Kubernetes dispose d'un grand nombre de ressources fréquemment mises à jour et ceci inclut un grand nombre de données qui sont ingérées dans ces métriques et ses mesures. En ajoutant la quantité de données qu'on peut avoir d'un système NFV complexe déployée sur un cluster, la pile de métriques est énorme, et ça peut entraîner des coûts élevés pour certains fournisseurs de cloud ou en espace mémoire, s'il s'agit d'un centre de données. La solution est

de configurer nos outils pour exposer seulement les métriques souhaitées. Ainsi, il est aussi important d'envisager des outils capables à gérer l'information à grande échelle comme Spark, et compatibles avec les plateformes modernes tels que Kubernetes.

## **2.6 Télémétrie et DevOps**

DevOps est une nouvelle culture en génie logiciel qui a appuyé la collaboration et la communication entre les développeurs et les opérateurs. De plus, il a automatisé la livraison des logiciels tout en réduisant le temps et l'effort humain.

Pour une haute performance, les industries étudient comment garantir un retour rapide et continu d'informations des opérations au développement et en temps réel. De ce fait, il est important de mettre la lumière sur les facteurs clés qui favorisent une telle rétroaction. Dans ce chapitre, nous allons voir la relation entre une information retournée (une rétroaction) par un opérateur et un pipeline DevOps.

### **2.6.1 Expérimentation**

On a un site web statique très basique avec une seule page `index.html` sur un cluster Kubernetes et assurer sa mise à jour par un déploiement automatisé déclenché après toute modification au niveau du code source. En d'autres termes, un pipeline d'intégration continue a été réalisé pour automatiser la mise jour.

Ils existent plusieurs façons de réaliser ce pipeline et vu que notre objectif est de comprendre la relation entre l'automatisation et la télémétrie, nous choisissons une des méthodes simples à implémenter.

En simplifiant, les étapes de l'épreuve sont comme suit :

1. Créer un emplacement sur GitHub : GitHub est un gestionnaire de versions qui va nous permettre de déposer notre code source de notre site web.

2. Préparer ou créer la plateforme : Installer un cluster Kubernetes et pour simplifier la tâche, nous avons installé Docker, Minikube, et Kubectl sur un système Linux (Ubuntu 16.04).
3. Créer un registre d'images local (les images dockers de la page web).
4. Créer et construire un pipeline dans Jenkins.
5. Transférer les modifications de code dans le pipeline.
6. Choisir pour la surveillance et le monitoring kubernetes Dashborad.

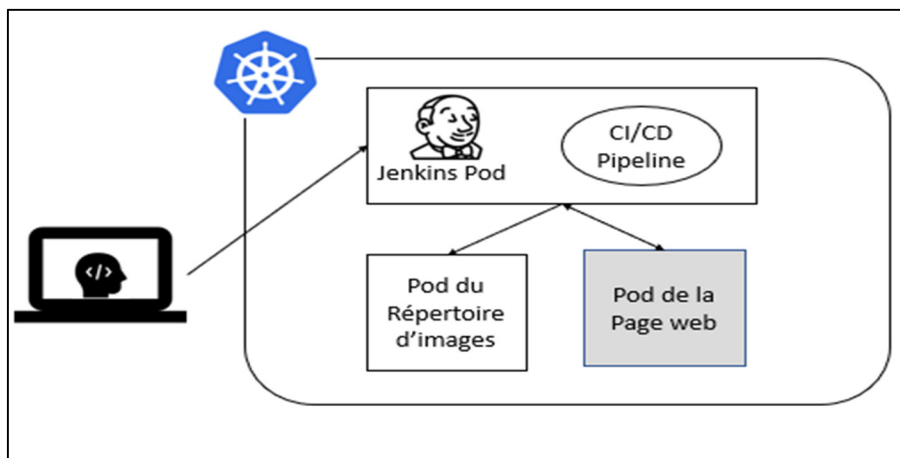


Figure 2.9 Le processus du pipeline CI expérimenté avec Jenkins et Kubernetes

Le code source de la page web a été déposé sur le compte GitHub lié avec le serveur d'intégration Jenkins via un moyen sécurisé d'authentification avec jeton (token). Au niveau de l'interface graphique offerte par Jenkins, un pipeline est créé et un déclencheur (nommé trigger sur Jenkins) est activé.

À ce niveau, nous procédons à l'exécution d'une suite d'actions :

1. Le développeur ajoute des lignes de codes pour un affichage de la phrase « une phrase test ».

2. Le développeur supprime les lignes de codes de l’Action 1 et inclut une phase de test qui cherche l’acceptation de l’opérateur. Dans notre cas on joue les rôles et on valide le test, ensuite on refait l’action et cette fois on ne l’approuve pas.

L’observation est sur le lien externe de la page index.html et sur le tableau de bord Kubernetes « Kubernetes Dashboard ». Ce dernier est capable de collecter et de retourner d’une façon automatisée des informations du cluster tel que la santé et la fonctionnalité des pods et des conteneurs.

### Résultats :

Tableau 2.8 Résultats d’affichage d’un site web faisant partie d’un pipeline CI avec Jenkins après modifications

| Actions                                       | Résultats                                                         |
|-----------------------------------------------|-------------------------------------------------------------------|
| Action 1                                      | Ligne affichée                                                    |
| Action 2 (acte 1) : L’opérateur valide        | La page index.html est visible, « une phrase test » non affichée. |
| Action 2 (acte 2) : L’opérateur ne valide pas | La page index.html n’est pas visible                              |

### 2.6.2 Discussion

Le résultat de l’Action 1 montre que le pipeline de l’intégration continue est fonctionnel. Le code source modifié de la page index.html a été mis à jour avec succès et le pod correspondant est bien affiché sur le tableau de bord.

Le résultat de l’Action 2 souligne que la métrique acceptation de l’opérateur qui est une information issue de ce pipeline, retourne au développeur (à la première phase du pipeline) et influence clairement le passage du test au déploiement. En d’autres termes, l’information retournée est nécessaire pour déployer la mise à jour et d’une manière automatisée.

### **2.6.3 Conclusion**

Les données télémétriques au niveau applicatif ou d’environnement de production permettent aux ingénieurs DevOps d’automatiser les mécanismes de retours d’informations. Elles leur permettent aussi une surveillance à temps réel et une réaction rapide aux dysfonctionnements et aux problèmes qui surgissent. Par conclusion, la télémétrie est un facteur clé pour automatiser les retours d’informations.

Lorsqu’elle est correctement instrumentée, la télémétrie peut également être utilisée pour voir et comprendre en temps réel comment les clients interagissent avec l’application. Il peut s’agir d’informations cruciales pour les chefs de produit, les équipes marketing et le support client. Il est donc important que les mécanismes de rétroaction partagent des informations continues avec toutes les parties prenantes.

## **2.7 Discussion**

Nous avons créer une architecture (figure 2.2) qui est composée globalement de trois blocs : collecte, diffusion et visualisation, d’où deux concepts sont à l’issue de cette composition. Le premier est la façon de faire de Kubernetes pour collecter et visualiser les métriques qui présente le pipeline de données tel illustré dans la figure 2.10.

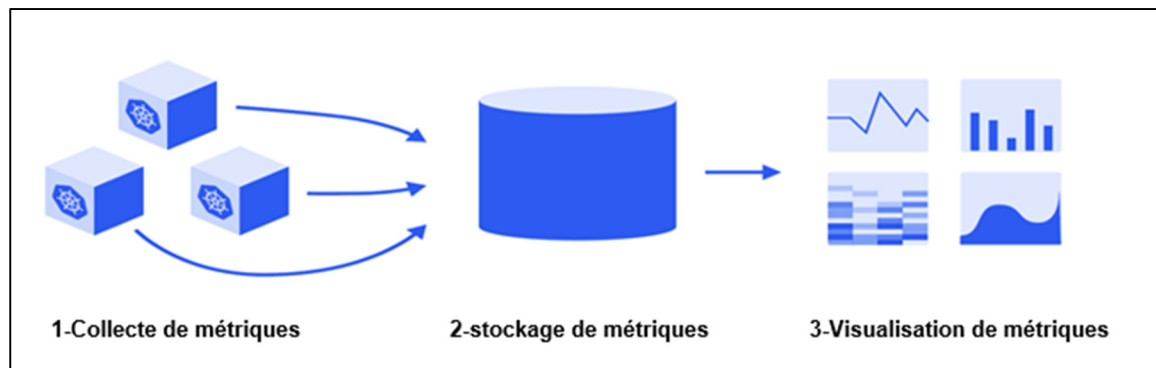


Figure 2.10 Le pipeline de métriques au sein d'un cluster Kubernetes

Tirée de Dtdog.Inc (2016)

Le deuxième concept est 'Big Data Landscap' ou la perspective Big Data qui se définit dans quatre étapes (figure 2.11).

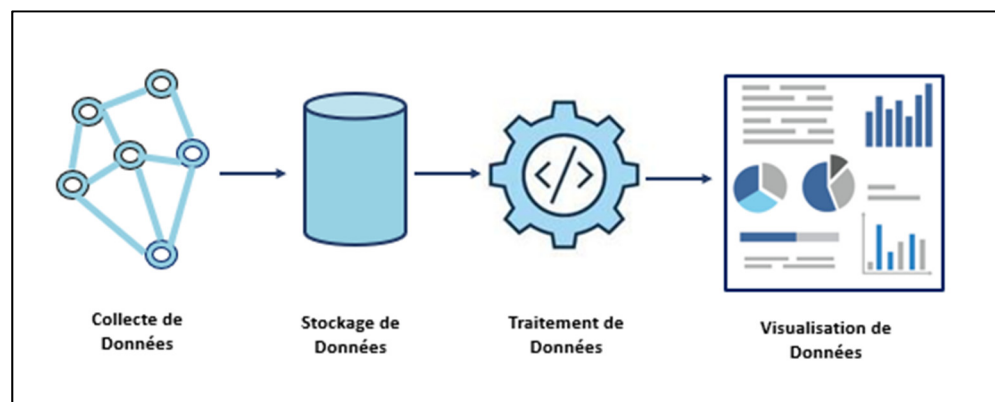


Figure 2.11 Perspective Big Data "Big Data Landscap"

Le traitement des informations télémétriques et leurs stockages ont été inclus comme deux sous-composantes séparées au niveau de la diffusion pour la simple raison que l'objectif est la collecte de télémétrie et non pas l'observabilité complète ou le traitement pour un premier objectif. L'observabilité est un concept plus large qui inclut l'analyse des données collectées, le traitement et les résultats déduites pour aider le système de surveillance et de monitoring à

prendre des décisions. Cependant, toutes ces parties sont complémentaires et il est plus efficace de construire l'étape et projeter la vision sur celle qui va suivre.

L'outil Prometheus avec une configuration Kube-state-metrics comme un module complet de collecte capable a tiré un grand nombre d'informations des composantes du cluster Kubernetes. Pour cibler un groupe de métriques ou supprimer un autre, il suffit d'adapter le fichier de configuration Prometheus ce qui fait que cette solution a la capacité a collecter l'information jugée la bonne selon les spécifications de l'analyse approfondie des besoins de l'entreprise. Dans ce contexte, il est important de fournir un module de collecte basé sur le fait d'accorder une grande importance à cibler la bonne information télémétrique de la plateforme adéquate au type de déploiement NFV et aux exigences de l'entreprise (une collecte sélective et intelligente). La masse d'informations tirées d'un déploiement NFV ou d'un environnement complexe tel qu'un centre de données NFV avec une plateforme Kubernetes est énorme. Les entreprises n'ont pas forcément besoin de toutes ses informations en même temps pour réaliser un objectif. Bien au contraire, prendre toute cette masse va alourdir le processus dans son ensemble, et il va lui faire perdre son temps, ses ressources et de l'argent bien évidemment.

## **2.8 Conclusion**

Le présent chapitre est une présentation de notre méthode à concevoir un système de télémétrie pour une meilleure visibilité sur un déploiement NFV sur une plateforme Kubernetes. Il s'agit d'un processus d'une étude qui a mené à une solution personnalisée d'une architecture micro-services. Au niveau de l'industrie Kontron, la solution a pu être déployée et tester et ceci est l'objet du chapitre 3

## **CHAPITRE 3**

### **UN SYSTÈME DE COLLECTE TÉLÉMÉTRIQUE SUR KUBERNETES AU NIVEAU D'UN MINI-CENTRE DE DONNEES NFV DE KONTRON**

#### **3.1 Introduction**

Toute entreprise avec une vision projetée sur le futur et un bon esprit de compétition est dans l'obligation de suivre l'évolution rapide des nouvelles technologies. Kubernetes est une contribution de la communauté open source qui aide à la minimisation du risque d'échec des centres de données des entreprises qui influence gravement la disponibilité des fonctionnalités et des services TI. Il a aussi facilité l'adaptation de l'architecture émergente des microservices, le maillage de service et a ouvert la voie à l'ensemble de l'écosystème natif du cloud.

Pour toutes ces raisons, les entreprises reconnaissent la valeur de ce standard qui a remplacé les technologies physiques. Cependant, les centres de données NFV augmentent la barre des défis. Une bonne gestion ou une évaluation de performance d'un déploiement NFV sur un environnement conteneurisé au niveau d'un centre de données s'avère encore plus complexe.

Ce chapitre est une application du processus de conception d'un système télémétrique présenté précédemment dans le chapitre 2. Nous allons pouvoir tester et évaluer notre travail dans le milieu SYMKLOUD, au niveau d'un mini-centre de données NFV de l'entreprise Kontron. Il s'agit d'une étude de faisabilité d'efficacité de notre démarche théorique ainsi qu'une évaluation de notre système de collecte télémétrique dans plusieurs aspects.

#### **3.2 Contexte du projet**

Ce projet est une proposition de recherche dans le cadre d'un contrat de partenariat entre Kontron et l'ÉTS afin de combler un besoin industriel de la compagnie. Le travail a été mené à

partir du laboratoire Multimédia de l'ÉTS et sur la plateforme SYMKLOUD qui se trouve au niveau de Kontron.

### 3.2.1 SYMKLOUD

SYMKLOUD est un produit de la compagnie Kontron. Cette dernière est un leader mondial de la technologie informatique embarquée. « Elle propose un portefeuille combiné de matériel, de middleware et de services sécurisés pour les applications Internet des objets (IoT) et Industrie 4.0. Avec ses produits standard et ses solutions sur mesure basées sur des technologies embarquées de pointe hautement fiables, Kontron fournit des applications sécurisées et innovantes pour une variété d'industries » (Kontron, 2018).

Ainsi, « SYMKLOUD est un concept de plate-forme convergente unique, conçu dès le départ pour les déploiements d'infrastructure cloud / cloud et NFV / SDN de câbles et d'opérateurs cloud. Réalisez des charges de travail d'applications virtualisées très efficaces pour la livraison de vidéo / contenu, Big Data / IoT, Mobile / Telco et Cloud » (Kontron, 2019).

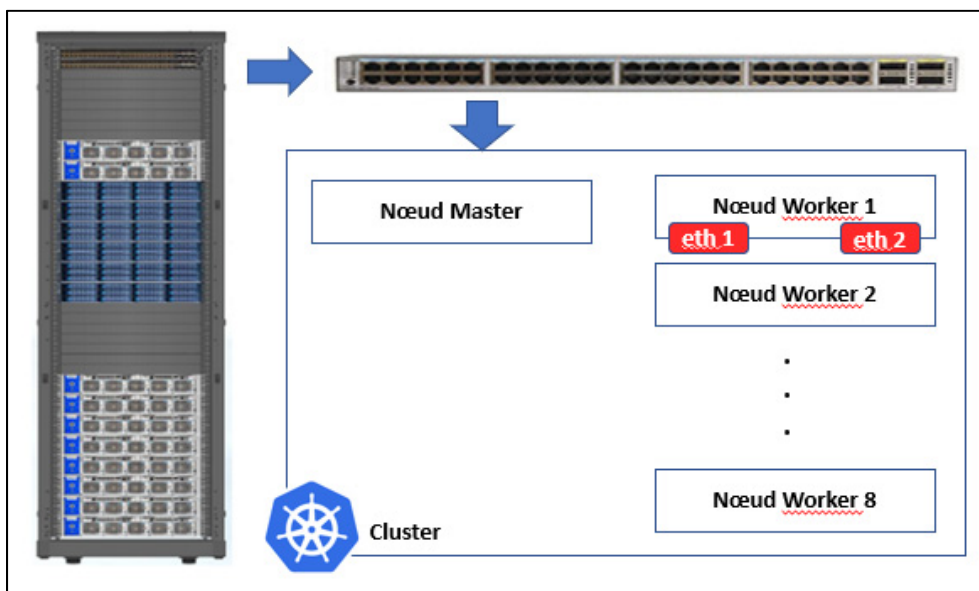


Figure 3.1 Cas d'utilisation simple d'un mini centre de données NFV

La figure 3.1 est un cas d'utilisation d'un centre de données NFV sur lequel s'est déroulée notre expérimentation, il est composé d'un seul nœud Master et huit nœuds Worker.

### **3.2.2 Objectifs**

1. Appliquer notre processus de conception (figure 2.1) présenté dans le chapitre antérieur pour concevoir un système amélioré de collecte télémétrique.
2. Implémenter notre solution de collecte conçue sur l'environnement SYMKLOUD.
3. Évaluer la solution.

### **3.2.3 Plan du travail**

Le plan est de suivre les étapes de notre processus proposées afin de concevoir un système de collecte d'informations télémétriques essentielles pour une meilleure visibilité sur les déploiements NFV au niveau des mini-centres de données de la compagnie. Il est indispensable de comprendre le besoin avant d'entamer une conception ou d'adopter une solution, et ceci est déjà inclus et pris en considération dans notre processus d'une façon détaillée. Cette manière a faire commence par une étude préalable jusqu'à l'implémentation de l'architecture établie.

## **3.3 Le cas d'étude Kontron**

Au niveau de la compagnie Kontron, ce processus a eu l'occasion d'être appliqué sous des conditions bien particulières à l'entreprise. La compagnie a adhéré à notre recherche dans le but de tester quelques déploiements NFV et d'évaluer quelques équipements en matière de consommation de ressources principalement. Un cluster Kubernetes (un environnement de test sur SYMKLOUD), au niveau d'un mini centre de données NFV pour tester la démarche comme suit :

### 3.3.1 Étude préalable

Dans ce cas d'étude, la compagnie avait souligné vouloir évaluer des déploiements NFV sur des équipements conçus à leur niveau en matière de consommation de ressources pour étudier la faisabilité d'élargir leur exploitation. Le groupe des ingénieurs Kontron a exprimé un besoin à collecter des métriques au niveau basique (conteneur) pour un début, et ce qui leur permettra si le projet est autant prometteur pour les objectifs d'affaires de la compagnie.

Après avoir clarifié l'objectif de l'entreprise attendu d'un déploiement NFV, il a été nécessaire d'identifier l'environnement conteneurisé qui représente la plateforme des expérimentations. Cet effort à définir et comprendre ce qui est SYMKLOUD fait partie de l'étude préalable dans la démarche de conception de notre solution, mais elle a déjà été abordée au début de ce chapitre.

### 3.3.2 Analyse approfondie

Cette phase est très importante, car elle aborde les besoins identifiés précédemment avec plus de précision et avec une perspective technique d'un ingénieur et d'un chercheur en même temps.

Afin d'évaluer un déploiement NFV en matière de consommation en ressources, la démarche de notre méthodologie a été comme suit :

1. **Identification des Métriques** : Pour observer la consommation de ressources des conteneurs d'un déploiement précis, nous identifions les métriques les plus communes comme suit :
  - A. Consommation CPU** : l'utilisation moyenne, en cœurs de processeur, sur une période donnée. Cette valeur est dérivée en prenant un taux sur un compteur CPU cumulatif fourni par le noyau.
  - B. Consommation mémoire** : la quantité de mémoire en cours d'utilisation mesurée en octets. Elle est notée au moment même de sa collecte.

- 2. Identification de la méthode de collecte :** Vu que le besoin, dans ce cas d'étude, se limite aux métriques des conteneurs, au niveau d'un cluster Kubernetes, il est suffisant de les collecter à partir du cAdvisor qui est intégré dans le kubelet. C'est le composant qui contrôle tout le conteneur sur chaque nœud du cluster et il est chargé de collecter toutes ces données (du conteneur). Kubelet expose toutes ces métriques d'exécution, ainsi que toutes les métriques cAdvisor, sur un nœud final `/metrics` au format d'exposition Prometheus qui va les récupérer à son tour.

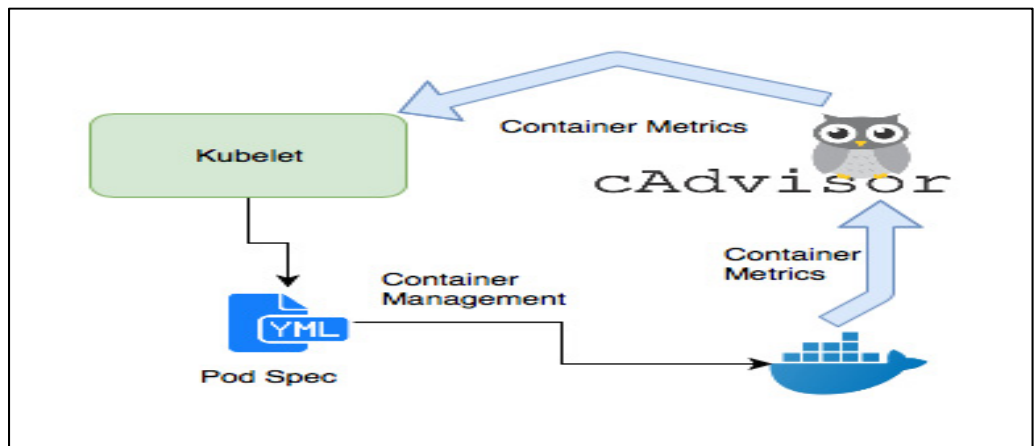


Figure 3.2 La collecte de métriques de conteneurs par cAdvisor  
Tiré de Goltsman, K (2019)

- 3. Identification des outils :** En s'appuyant sur notre étude et comparaison d'approches et d'outils de collecte (section 3.2 et section 3.3) et l'analyse comparative (section 4.4) qui fait partie de la présentation de notre approche, nos choix sont comme suit :
- A. **Module de collecte :** Prometheus at Kube-state-metrics.
  - B. **Module de diffusion (à temps réel) :** Apache Kafka et Apache Spark.
  - C. **Module de visualisation :** Grafana.
  - D. **Module de Stockage :** InfluxDB.
  - E. **Module d'apprentissage automatique :** MLib d'Apache Spark.

Une autre chose à prendre en considération est que nous devons avoir Kafka version 0.10.0 ou supérieure pour qu'il soit intégrable avec Spark.

### 3.3.3 Conception

La solution potentielle conçue se base sur le prototype global de la figure 2.2 et l'architecture de la figure 2.3.

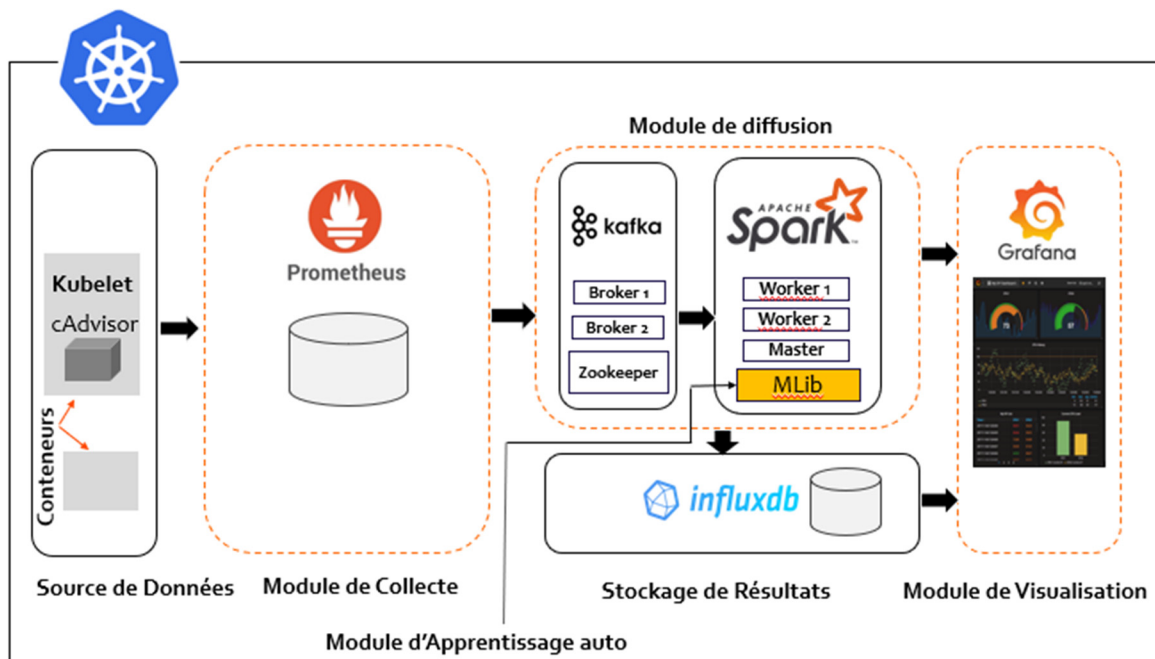


Figure 3.3 Architecture de la solution de collecte pour l'entreprise Kontron

### 3.3.4 Implémentation

Déployer la solution (figure 3.3) à demander plusieurs expérimentations vu le nombre de composants intégrés et la complexité de l'environnement Kubernetes. La méthode d'implémentation suivie ici consiste à déployer chaque outil séparément et assurer son fonctionnement avant de l'intégrer avec son sous-jacent dans l'architecture de la figure 3.3.

### 3.3.4.1 Préparation de l'environnement :

Avant d'installer nos outils (les composants de la solution), nous précisons que le tout sera déployé au niveau d'un seul namespace que nous avons choisi de nommer « monitoring ». Cette action peut être exécutée avec Kubectl selon (Kubernetes, 2020), ou « Kubernetes Dashboard » selon (Cynixit, 2020).

Kubectl est un outil en ligne de commandes de Kubernetes qui permet d'exécuter des commandes dans les clusters. Il sert à déployer des applications, inspecter et gérer les ressources du cluster et consulter les logs.

L'outil « Kubernetes Dashboard » est l'interface utilisateur Web officielle de Kubernetes et le tableau de bord qui fournit une vue d'ensemble du cluster ainsi que les ressources individuelles qui y sont exécutées. « Kubernetes Dashboard » est une alternative de Kubectl.

Ensuite, nous avons créé le fichier « clusterRole.yml » qui est utilisé pour configurer les rôles du cluster. Il est important pour que nos outils et surtout notre collecteur Prometheus disposent des autorisations appropriées sur l'API Kubernetes.

```

apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: default
  namespace: monitoring

```

Figure 3.4 Un fichier clusterRole.yml donnant des privilèges d'accès à Prometheus

### 3.3.4.2 Déploiement du collecteur Prometheus :

Le déploiement de Prometheus sur Kubernetes est pratiquement très simple. Il suffit d'exécuter quelques fichiers YAML pour le déployer. Plusieurs projets libres et tutoriels sont disponibles pour ce fait, et nous nous sommes inspirés de celui de Wilson (2019) pour notre cas.

Prometheus obtiendra sa configuration à partir d'un ConfigMap Kubernetes. C'est un fichier qui va nous permettre la mise à jour de la configuration séparément de l'image.

Pour le déploiement sur le cluster, nous exécutons la requête « `kubectl create -f prometheus-config.yaml` ». Cette configuration contient l'image officielle de Prometheus depuis le docker hub (le répertoire d'images Docker).

Le fichier `prometheus.yaml` contient toute la configuration pour découvrir dynamiquement les pods et les services exécutés dans le cluster Kubernetes. Tel mentionner dans notre référence (Wilson, 2019) nous avons les tâches de scrape (ramener des métriques) suivantes dans notre configuration de scrape Prometheus :

1. **kubernetes-apiservers** : il obtient toutes les métriques des serveurs d'API.
2. **kubernetes-nodes**: toutes les métriques de nœuds Kubernetes seront collectées avec cette tâche.
3. **kubernetes-pods**: toutes les métriques du pod seront découvertes si les métadonnées du pod sont annotées avec les annotations `prometheus.io/scrape` et `prometheus.io/port`.
4. **kubernetes-cadvisor** : il collecte toutes les métriques cAdvisor.
5. **kubernetes-service-endpoints**: tous les points de terminaison du service seront supprimés si les métadonnées du service sont annotées avec les annotations `prometheus.io/scrape` et `prometheus.io/port`. Ce sera une surveillance de la boîte noire.

Après avoir dresser notre configuration, nous passons à la création du déploiement Prometheus par l'exécution du fichier « `prometheus-deployment.yaml` » via la requête « `kubectl create -f prometheus-deployment.yaml` »

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-deployment
  namespace: monitoring
  labels:
    app: prometheus-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server
  template:
    metadata:
      labels:
        app: prometheus-server
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--storage.tsdb.path=/prometheus/"
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: prometheus-config-volume
              mountPath: /etc/prometheus/
            - name: prometheus-storage-volume
              mountPath: /prometheus/
      volumes:
        - name: prometheus-config-volume
          configMap:
            defaultMode: 420
            name: prometheus-server-conf
        - name: prometheus-storage-volume
          emptyDir: {}

```

Figure 3.5 Une copie du fichier du déploiement  
prometheus-deployment.yaml

Nous avons procédé à l'installation de l'exportateur de métriques Kube-state-metrics via le package libre de fichiers yaml du projet de Kamol Hasan (2019). Les données télémétriques sont ramenées dans un format d'exposition des métriques Prometheus, ce qui facilitera leur lecture par ce dernier.

Le projet (Kamol Hasan, 2019) prépare et installe ces quatre éléments :

1. **Déploiement** : où le conteneur récupère l'image : quay.io/coreos/kube-state-metrics:v1.6.0.
2. **Compte de service** : compte de service de l'objet de déploiement.
3. **Role and RoleBinding**: rôle de cluster RBAC et liaison de rôle pour le compte .de service
4. **Service** : service k8s qui écoute les pods en cours de déploiement.

Ensuite, nous revenons à la configuration de Prometheus et nous ajoutant la partie qui configure kube-state-metrics (figure 3.6) pour extraire les métriques du service k8s que nous venons de créer via le projet (Kamol Hasan, 2019).

```
global:
  scrape_interval:    15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'kube-state-metrics'
    static_configs:
      - targets: ['address'] //address of the k8s service
```

Figure 3.6 Une configuration Prometheus qui récupère les métriques kube-state-metrics

Tiré de Kamol (2019).

Ainsi, notre module de collecte est bien implémenté, configuré et prêt pour une prochaine étape d'intégration avec un autre outil afin d'envoyer les métriques vers InfluxDB et Kafka.

### 3.3.4.3 Déploiement du cluster Kafka :

Kafka est un système de messagerie et d'intégration potentielles qui est chargé de transférer les données d'une application à une autre. Les applications peuvent ainsi se concentrer sur les données.

Nous nous appuyant sur la documentation officielle de Kafka pour éclaircir les notions de base et la terminologie liées à cet outil :

1. **Topic** : il conserve les données qu'il reçoit dans des rubriques (topics), correspondant à des catégories de données. Les deux systèmes échangent des données en s'inscrivant sur le même sujet.
2. **Producer** : il publie des données dans les topics Kafka. En effet, Kafka ne supprime pas ces messages, mais les conserve. Ainsi, tout nouveau producteur peut récupérer soit tous les messages du début, soit uniquement les messages futurs à venir.
3. **Consumer** : c'est le système qui lira les données des rubriques.
4. **Broker** : il existe une forme de middleware qui traduit les messages dans une langue généralement plus courante.
5. **Partition** : il est un moyen de distribuer des données à partir du même sujet. Chaque partition contient une séquence de messages, avec un identifiant séquentiel, ordonné et immuable commençant de l'ancien au plus récent.

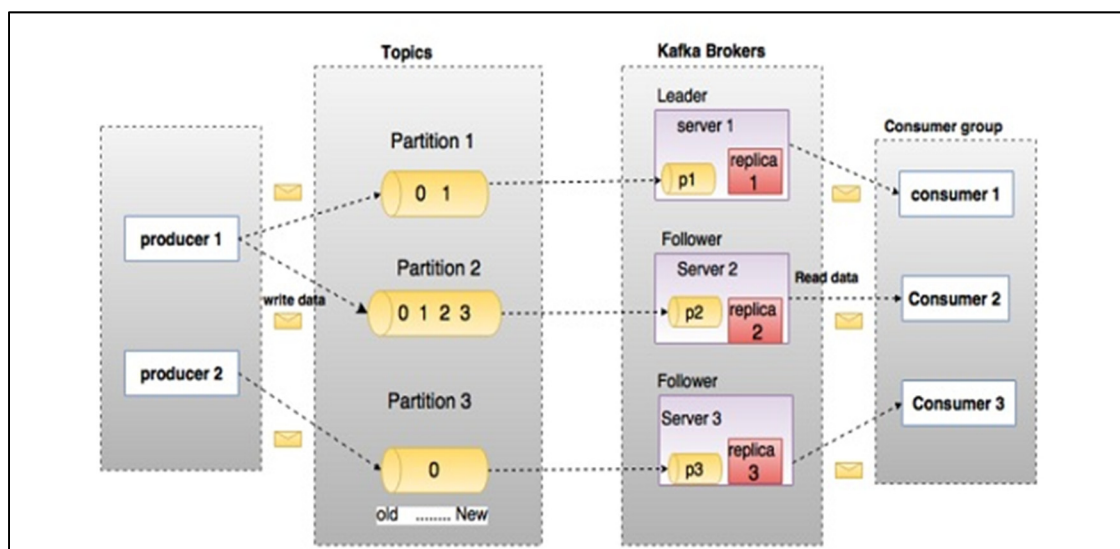


Figure 3.7 Principales terminologies

Tirée de Tutorialspoint (s.d)

Comme illustré dans la figure 3.8, il est composé de producteurs. Un producteur est un point de récupération des données et une source de données comme un outil de collecte de métriques. En parallèle, il existe un ou plusieurs consommateurs qui vont prendre ces données pour les utiliser.

Le cluster Kafka 1.0.2 (version compatible avec Apache Spark) déployé au sein de notre environnement Kubernetes est composé de deux « broker » et un ZooKeeper et nous nous sommes appuyé sur le projet open source de Yolean (2016).

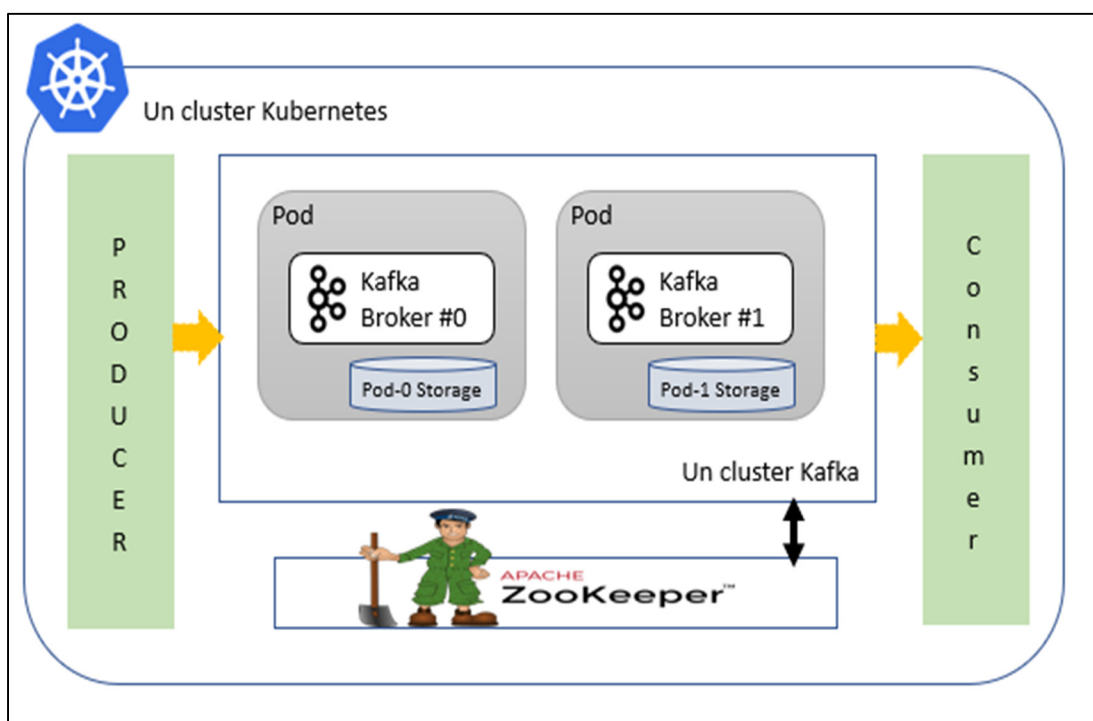


Figure 3.8 Un cluster Kafka sur Kubernetes

Les étapes d'implémentation sont les suivants :

1. **Déployer ZooKeeper** : sans un déploiement opérationnel de Zookeeper, Kafka ne pourra pas fonctionner. Pour cela il faut un fichier `zookeeper.yml` qui démarre un service et un déploiement qui planifie des pods Zookeeper sur un cluster Kubernetes.

2. **Créer un service Kafka :** il s'agit de la création d'un fichier `kafka-service.yml` qui définira le service kafka. Ce service gère les déploiements du Broker Kafka en équilibrant la charge des nouveaux pods Kafka.
3. **Définir le contrôleur de réplication Kafka :** en déployant un fichier `kafka-repcon.yml` contrôleur de réplication kafka.
4. **Démarrez le serveur Kafka :** les propriétés de configuration d'un serveur Kafka sont définies dans le fichier `config/server.properties`, et pour démarrer le serveur kafka il suffit d'exécuter la requête « `kafka-server-start.sh config/server.properties` »

#### 3.3.4.4 Déploiement du Cluster Apache Spark :

Le cluster Spark a été créé sur Kubernetes en utilisant le projet open source de Kubernetes(2017) et la documentation officielle d'Apache Spark (2019).

La première étape est la création du Master Node de Spark qui contient le « Driver Program » comme montrée dans la figure 3.9. Le « Driver Program » est le processus exécutant la fonction `main ()` de l'application et créant le « SparkContext ». Ce dernier est un service externe pour l'acquisition de ressources sur le cluster.

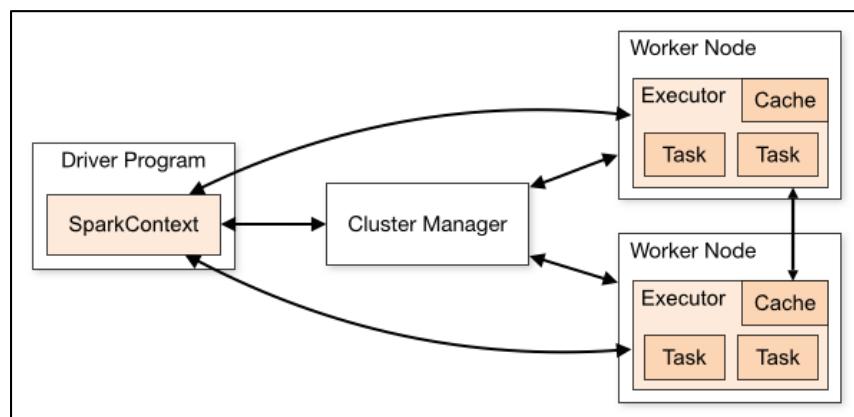


Figure 3.9 Architecture d'un cluster Spark

Tiré de Apache Spark (2019)

La création du Master Spark et le démarrage de son service consiste à déployer un fichier yaml (figure 3.10) pour créer un contrôleur de réplication exécutant le service Spark Master via la requête suivante : « `kubectl create -f examples/staging/spark/spark-master-controller.yaml` » selon la hiérarchisation du projet open source utilisé.

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: spark-master-controller
spec:
  replicas: 1
  selector:
    component: spark-master
  template:
    metadata:
      labels:
        component: spark-master
    spec:
      containers:
        - name: spark-master
          image: k8s.gcr.io/spark:1.5.2_v1
          command: ["/start-master"]
          ports:
            - containerPort: 7077
            - containerPort: 8080
          resources:
            requests:
              cpu: 100m
```

Figure 3.10 Exemple d'un fichier yaml qui déploie un contrôleur de réplication exécutant le service Spark Master.

Tiré de Kubernetes (2017)

Ensuite, créer un point de terminaison de service logique que les Workers Spark peuvent utiliser pour accéder au Pod Master via la requête : « `kubectl create -f examples/staging/spark/spark-master-service.yaml` »

Afin d'accéder à Spark WebUI, nous avons besoin d'un proxy spécialisé. Ce proxy est nécessaire pour accéder aux journaux de travail à partir de l'interface utilisateur Spark. La requête exécutée est « `kubectl create -f examples/staging/spark/spark-ui-proxy-controller.yaml` ».

En plus du proxy, nous avons besoin d'un service d'équilibrage « Loadbalanced Servicecorrespondant pour notre proxy Spark. Pour cela on exécute « `kubectl create -f examples/staging/spark/spark-ui-proxy-service.yaml` ».

La seconde étape est le démarrage des nœuds Worker Spark en créant un contrôleur de réplication spécial aux Workers de Spark. Ces derniers fournissent des ressources d'exécution et des capacités de cache de données pour notre programme. La requête utilisée est « `kubectl create -f examples/staging/spark/spark-worker-controller.yaml` ». Elle déploie un fichier yaml comme celui présenté dans la figure 3.11.

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: spark-worker-controller
spec:
  replicas: 2
  selector:
    component: spark-worker
  template:
    metadata:
      labels:
        component: spark-worker
    spec:
      containers:
        - name: spark-worker
          image: k8s.gcr.io/spark:1.5.2_v1
          command: ["/start-worker"]
          ports:
            - containerPort: 8081
          resources:
            requests:
              cpu: 100m
```

Figure 3.11 Exemple d'un fichier yaml qui déploie un contrôleur de réplication qui gère les modules Workers de Spark.

La figure 3.12 est le Dashboard en résultat qui montre la structure de notre cluster Spark déployé. Et comme illustré dans la figure 3.13 on remarque que l'outil nous fournit des informations de notre déploiement Spark comme le nombre de nœuds de type « worker », la consommation de ressources et le statut actuel du cluster.

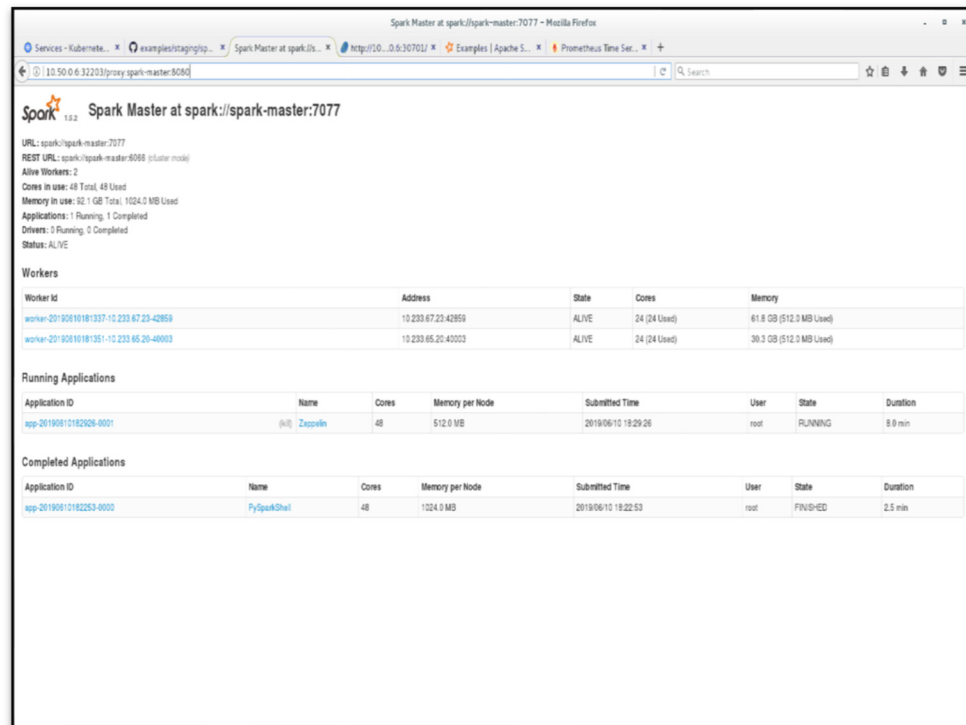


Figure 3.12 Tableau de bord du cluster Spark déployer sur le cluster Kubernetes de SymKloud

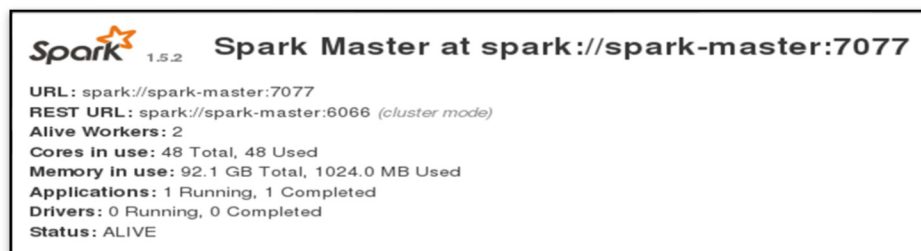


Figure 3.13 Informations sur notre le cluster Spark déployé sur SymKloud prise du Dashboard de la figure 3.12

Après avoir déployé notre cluster Spark avec deux Workers, nous déployons un outil Zeppelin qui sert à lancer des tâches dans le cluster Spark via une interface Web ou une ligne de commande Spark traditionnelle. Le résultat de notre déploiement est aperçu dans la figure 3.14. Enfin, notre outil Spark est prêt à être utilisé.

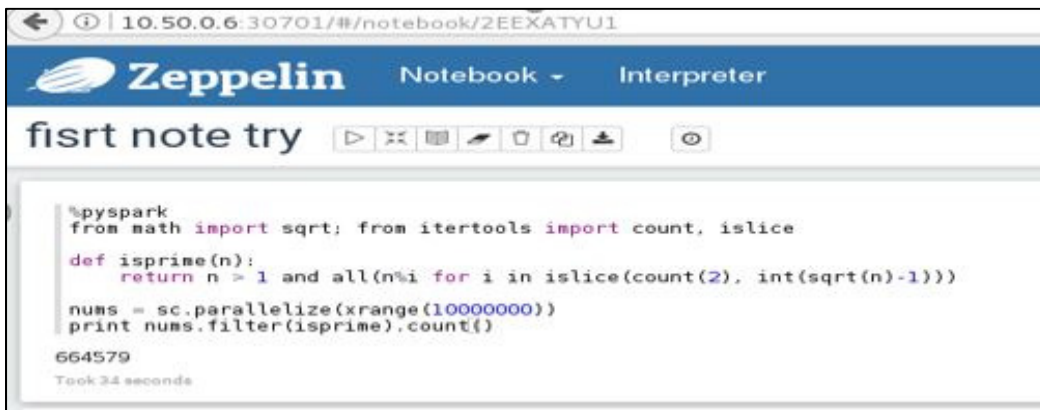


Figure 3.14 L’outil Zeppeline déployé sur le cluster K8s de SymKloud

Nous allons utiliser Spark Streaming pour faciliter la diffusion de métriques vers d’autres bases de données et des outils de visualisation. C’est une extension de Spark Core qui fournit des capacités de traitement tolérant aux pannes des données de flux en direct. Il divise le flux entrant en microlots d’intervalles spécifiés et renvoie Dstream. Dstream représente un flux continu de données ingérées à partir de sources telle que Kafka. Les flux Dstream sont traités et transférés vers les systèmes de fichiers, les bases de données et les tableaux de bord en direct.

### 3.3.4.5 Déploiement de Grafana :

Nous utilisons l’image officielle de Grafana sur Docker Hub et nous la déployons avec la requête « `kubectrl create deployment grafana --image=docker.io/grafana/grafana:5.4.3` » selon la version choisie.

Le service Grafana est censé être accessible depuis l’extérieur du cluster afin d’accéder au tableau de bord de notre outil de visualisation. Pour cela, nous exposons le service en utilisant

le type de service LoadBalancer au lieu de ClusterIP, avec la requête : « `kubectl expose deployment grafana --type=LoadBalancer --port=80 --target-port=3000 --protocol=TCP` ».

Enfin, notre outil Grafana est prêt à être exploité sur le port 80 par défaut. Pour visualiser une source de données comme Prometheus ou InfluxDB, il suffit d'ajouter le chemin dans la configuration du tableau de bord voulu via l'interface Grafana facile à manipuler.

### 3.3.4.6 Intégration de Kafka et Prometheus :

Cette étape revient au simple fait que Prometheus est notre collecteur de métriques choisi et ces informations, auxquelles nous nous intéressons, doivent être envoyées à Kafka. Ainsi, il faut définir Prometheus comme un producteur pour le composant Kafka. Cependant, cette intégration des deux outils ne se fait pas directement, elle nécessite un adaptateur comme illustré dans la figure 3.15. Prometheus-kafka-adapter est une des solutions open source qui existent. C'est un service qui reçoit les métriques Prometheus via « `remote_write` », les marshalise dans JSON et les envoie vers un topic Kafka.

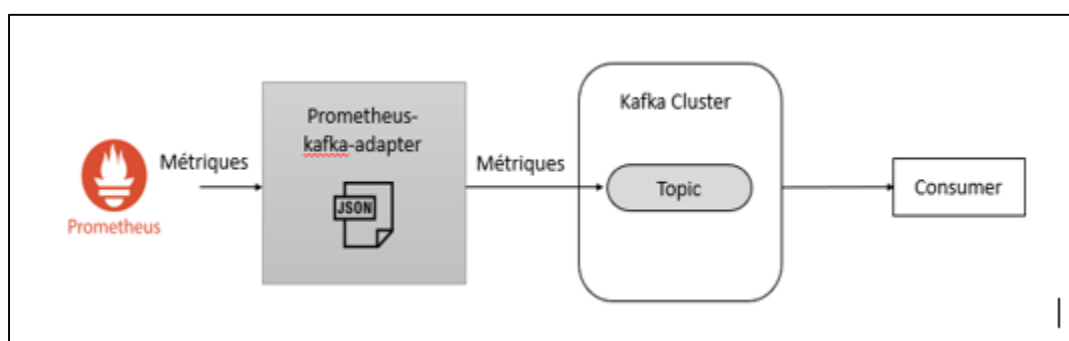


Figure 3.15 intégration de Prometheus comme un consommateur à Kafka  
via Prometheus-kafka-adapter

Nous utilisons l'image Docker « `telefonica/prometheus-kafka-adapter:1.7.0` » et le projet open source (Telefonica. 2019) pour déployer Prometheus-kafka-adapter. Ensuite, Prometheus doit avoir une URL `remote_write` configurée, pointant vers le point de terminaison `'/' receive'` de

l'hôte et du port sur lesquels le service prometheus-kafka-adapter est en cours d'exécution. À titre d'exemple :

```
remote_write : - url : "http://prometheus-kafka-adapter : 8080/receive"
```

### 3.3.4.7 Intégration de Apache Spark et Kafka :

La plateforme de messagerie Kafka se charge de la diffusion en continu pour Spark. En d'autres termes, Kafka centralise les flux de données en temps réel et les diffuse vers Spark Streaming pour subir un traitement à l'aide d'algorithmes complexes (figure 3.16).

Nous nous sommes assuré que les deux outils sont compatibles vu que Spark requiert une version Kafka 0.10.0 ou supérieure et nous avons choisi Kafka 1.0.2.

Ils existent deux façons pour une intégration Apache Kafka Spark streaming de telle manière que Spark reçoit les données de Kafka. La première consiste à utiliser des récepteurs et l'API de haut niveau de Kafka, et une seconde, ainsi qu'une nouvelle approche, est sans utiliser de récepteurs. Il existe différents modèles de programmation pour les deux approches, tels que les caractéristiques de performance et les garanties sémantiques. Plusieurs travaux existent dans cette voie et nous avons sélectionné le projet open source de Gour, R (2018) qui a donné un résultat positif.

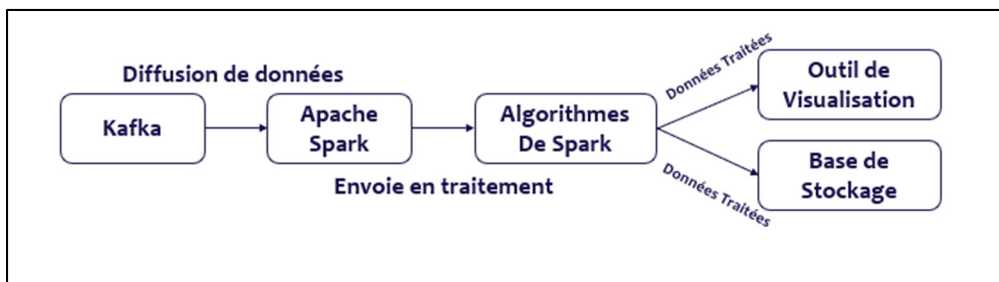


Figure 3.16 Intégration de Kafka et Spark

Pour les deux approches, cette intégration demande une expertise pour comprendre la terminologie et la variété de concepts liés à ces outils. L'approche basée sur les récepteurs consiste

à créer un récepteur comme consommateur ‘consumer’ au niveau de kafka et les données reçues sont stockées dans des exécuteurs Spark. Puis les jobs lancés par « Kafka - Spark Streaming » traite les données. Cette approche peut perdre des données en cas d’échec dans la configuration par défaut. Par conséquent, nous devons également activer les journaux à écriture anticipée dans Kafka Spark Streaming, pour éviter la perte de données.

En revanche, l’approche directe garantit des transferts de bout en bout plus solides. Elle interroge périodiquement Kafka pour les derniers décalages dans chaque rubrique et partition, plutôt que d’utiliser des récepteurs pour recevoir des données. Sur cette base, nous avons bien choisi l’approche directe pour son avantage.

### **3.4 Discussion**

Après avoir réussi l’implémentation du prototype, on remarque que le système conçu pour surveiller des déploiements NFV sur un environnement Kubernetes peut être perçu comme un service (concept du monitoring-as-a-service) et ce dernier a besoin d’être surveillé à son tour et continuellement. Le système lui-même est capable de détecter un dysfonctionnement d’une partie de ses composants, mais ceci n’est pas toujours vrai.

### **3.5 Conclusion**

Au cours de ce chapitre, nous avons implémenté notre solution proposée dans ce chapitre (Figure 3.3) de la collecte télémétrique destinée aux déploiements NFV sur une plateforme Kubernetes (chapitre 2) sur SymKloud, comme un cas d’étude spécifique à la compagnie Kontron. Le chapitre suivant illustrera les tests d’évaluations de cette solution mis en œuvre.



## CHAPITRE 4

### RÉSULTATS ET ÉVALUATION

#### 4.1 Introduction

Des tests ont été réalisés dans le but d'évaluer notre système de collecte télémétrique dédié au déploiement NFV sur une plateforme Kubernetes au niveau d'un centre de données.

#### 4.2 Outils et environnement de test

Notre déploiement NFV pour cette expérimentation de test est un contrôleur de prestation de services (SDC). C'est un ensemble de services réseaux orientés Web combinés dans le but de contrôler et d'inspecter le flux de trafic HTTP entrant dans un maillage de service Istio dans Kubernetes. Tel qu'il est décrit par (OpenNFV.s.d) : « il permet de démontrer le bac à sable Istio, y compris les concepts de maillage de service et les outils environnants, notamment le traçage, la surveillance et la journalisation ».

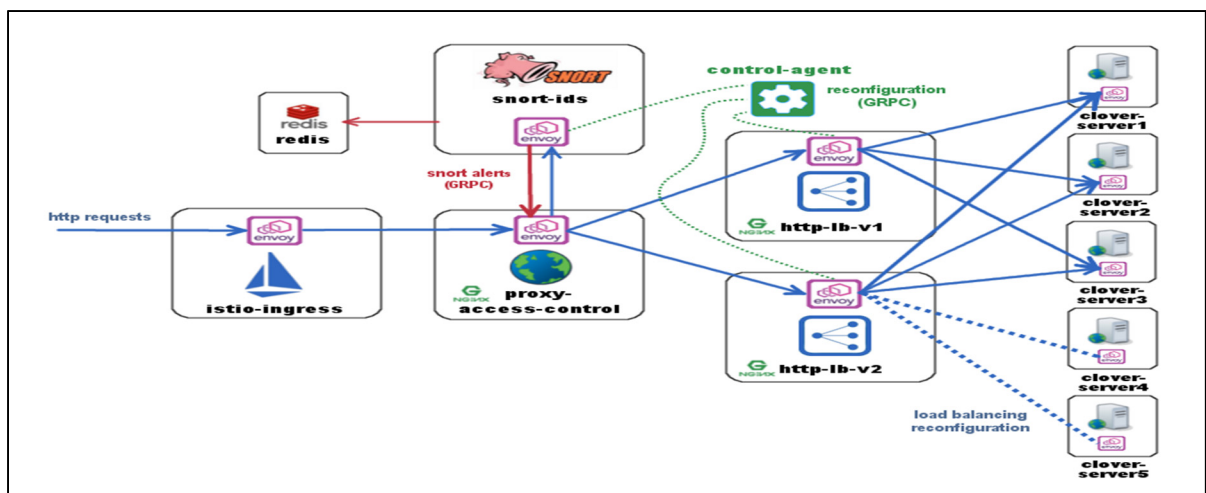


Figure 4.1 Architecture d'un contrôleur de prestation de services (SDC) déployé sur SymKloud

Tiré de OpenNFV (s.d)

Cet exemple est déployé sur notre cluster Kubernetes au niveau de l'environnement de test SymKloud (section 5.2.1) via le projet open source (OpenNFV.s.d). Ce dernier inclut l'outil de traçage « Jaeger » pour surveiller cet exemple qui est un VNF en microservices avec un maillage Istio.

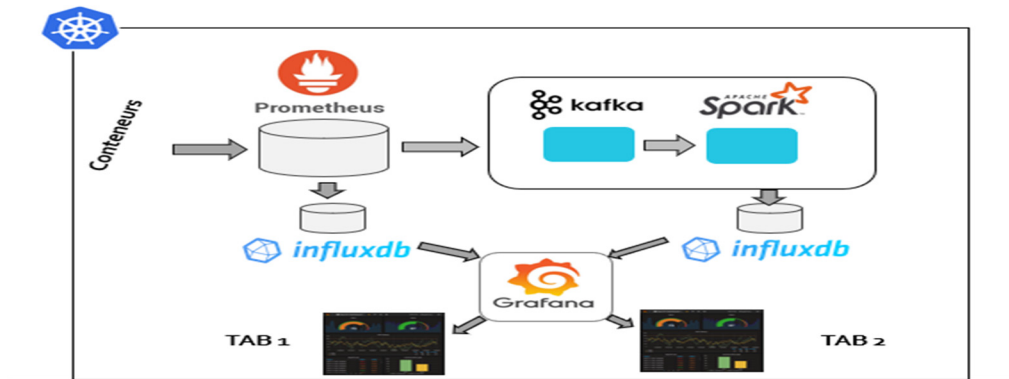


Figure 4.2 Architecture de l'intégration de Grafana avec deux points finaux de métriques

Ensuite, nous avons besoin d'assurer que notre outil de visualisation Grafana nous donne une visibilité sur deux points finaux de métriques et les visualiser sur deux tableaux de bord comme suit :

1. TAB 1 : la source de données et InfluxDB qui provient de Prometheus.
2. TAB 2 : la source de données et InfluxDB qui provient de « Kafka-Spark Streaming ».

En d'autres termes, nous voulons une visibilité au début du processus de notre collecte et à la fin du processus de la diffusion en temps réel pour une meilleure observation sur le flux des données collectées.

Il est possible de connecter Grafana directement à la source de données de Prometheus, mais nous souhaitons avoir deux sources avec le même type et format de données pour nous assurer que la comparaison entre les nos résultats est fiable.

Les tableaux de bord configurés pour les deux sources sont « K8s Container » qui nous aident à visualiser les métriques liées aux conteneurs dans notre cluster.

Ainsi nous utiliserons Kubernetes Dashboard qui fournit une vue d'ensemble du cluster ainsi que des ressources individuelles qui y sont exécutées.

### **4.3 Test de fiabilité et d'efficacité**

Dans le but de vérifier si le système de collecte est en fonctionnement nous avons les tâches suivantes :

1. Vérifier que les métriques aboutissent à notre tableau de bord Grafana. Dans ce but, nous nous appuyons sur Kubernetes Dashboard pour nous assurer que notre déploiement est réussi, sur Jaeger pour s'assurer que les microservices du SDC sont correctement liés, et sur le tableau de bord Grafana TAB 2 pour vérifier que la collecte des métriques est faite.
2. Vérifier que les données collectées au niveau de l'outil Prometheus sont celles visualisées sur notre tableau de bord Grafana avec notre système de diffusion en temps réel. Nous prenons la métrique consommation mémoire d'un conteneur « Prometheus » des deux tableaux de bord TAB 1 et TAB 2 sur un petit intervalle de temps  $[t, t+15]$  (en secondes) et nous comparons. Nous faisons la même chose avec la métrique consommation CPU du conteneur « redis ». Ce sont deux conteneurs pris du déploiement NFV (SDC) et d'un élément de notre solution de collecte qui est Prometheus lui-même. Cette procédure de comparaison est réalisée avec un échantillon de 10 Conteneurs vis-à-vis les deux métriques consommation CPU et mémoire.

#### **4.3.1 Résultats**

1. Les résultats du déploiement SDC sur notre Kubernetes Dashboard et Jaeger sont illustrés dans les figures 4.3 et 4.4.

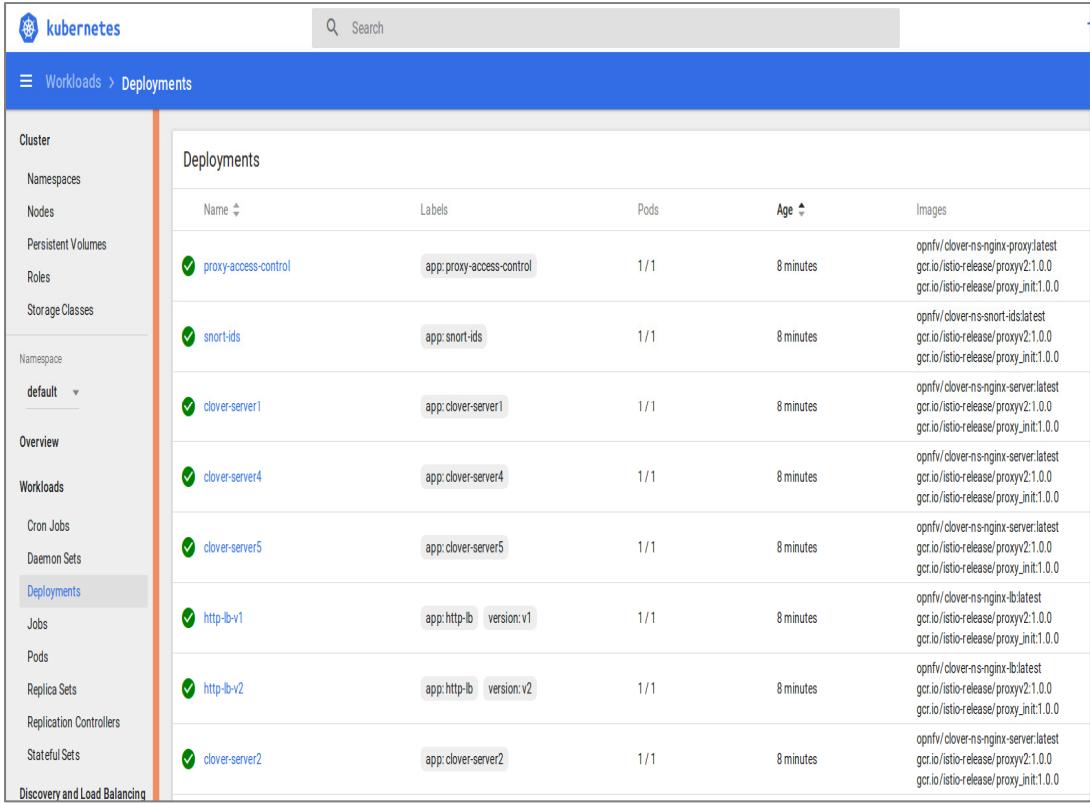


Figure 4.3 Le déploiement SDC sur Kubernetes Dashboard

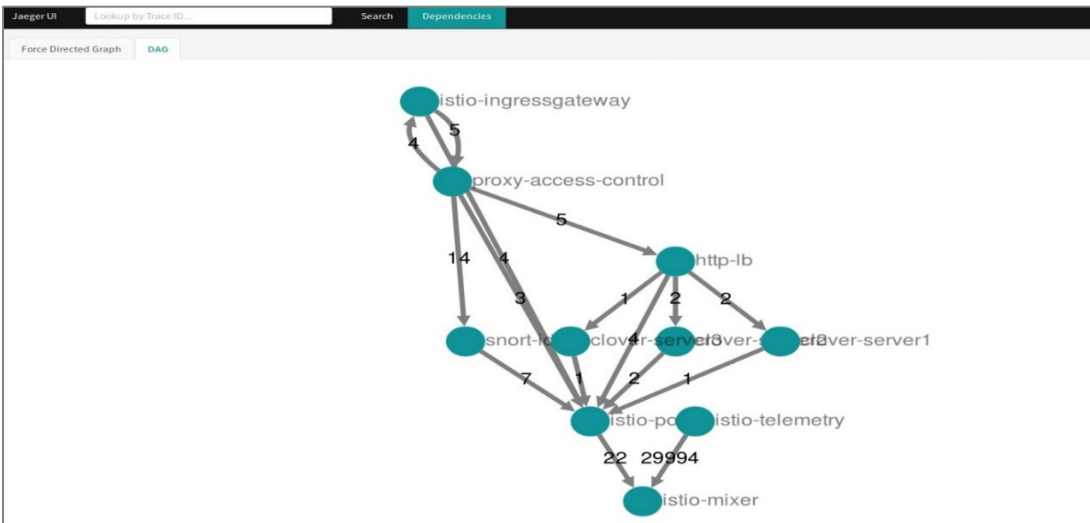


Figure 4.4 Déploiements de Service Delivery Controller (SDC)  
sur Jaeger - Graphique des dépendances

2. Les Résultats obtenus à partir de TAB 1 et TAB 2 présentent la consommation mémoire du conteneur « Prometheus » collecté respectivement avec notre collecteur Prometheus et notre système de collecte télémétrique. Les résultats pour les huit autres conteneurs sont très similaires et ces deux cas dont nous nous sommes limités.

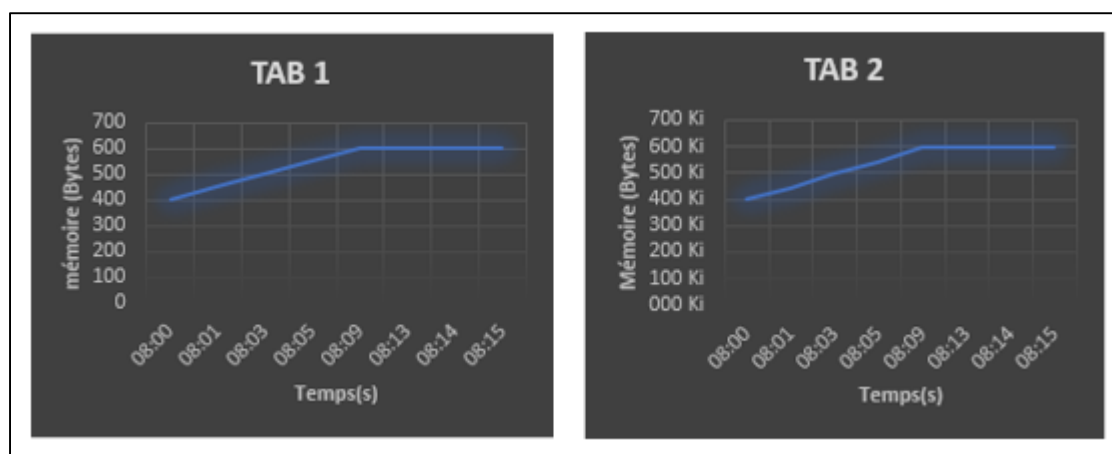


Figure 4.5 Consommation mémoire du conteneur « Prometheus » visualisée sur deux tableaux de bord TAB1 et TAB2.

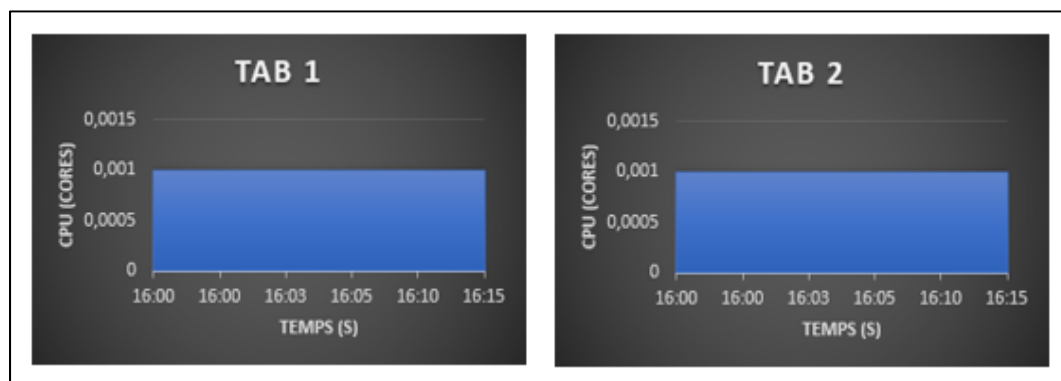


Figure 4.6 Consommation CPU du conteneur « redis » visualisée sur deux tableaux de bord TAB1 et TAB2.

### 4.3.2 Discussion

Les figures 4.3 et 4.4 nous montrent que notre déploiement SDC est réussi et avec une image sur les dépendances entre les services. Quant aux figures 4.5 et 4.6, nous concluons que les métriques sont bien collectées et aboutissent à notre outil de visualisation Grafana (TAB 2) via notre système implémenté. La comparaison entre les deux points de collecte à montrer que notre solution est fiable et rapporte les bonnes informations malgré la légère différence (0.02ki en suivant l'échelle de l'outil Grafana dans les valeurs de la consommation CPU du conteneur « Prometheus » remarqué dans les courbes de la figure 4.5.

En choisissant un conteneur du déploiement NFV « redis » et celui du système « Prometheus », nous soulignons que notre solution est capable de faire de la collecte télémétrique au sein d'un cluster en couvrant tous les conteneurs. En surveillance, ceci est appelé « monitoring-as-a-service » ou la surveillance comme un service.

## 4.4 Test de performance

Nous allons vérifier la capacité de notre solution à collecter les métriques conteneurs de nos déploiements. Le tableau de bord utilisé sur Grafana est « K8s container ». Ensuite, nous évaluons les ressources mémoire et CPU consommés par ce système de collecte en s'appuyant sur les informations du tableau de bord « Cluster health ». Ce dernier est implémenté sur Grafana par une simple importation d'un fichier Jason via l'interface Grafana. Ils existent plusieurs tableaux de bord capables d'importer ces mêmes métriques liées à un cluster.

Le dernier test est d'augmenter le volume de notre déploiement NFV en augmentant le nombre du serveur web « clover-server » de cinq à dix, et vérifier si notre système est toujours en fonctionnement.

#### 4.4.1 Résultats

1. **Métriques de conteneurs** : l'accent n'est pas sur la fonctionnalité de ce service VNF, mais sur la performance de notre solution de collecte télémétrique. Le tableau de bord « K8s Container » nous a permis de visualiser les métriques rassemblées par notre système de collecte comme illustré dans les figures 4.7, 4.8, 4.9, et 4.10. Parmi les métriques collectées, ces figures présenteront comme échantillon : consommation mémoire, consommation CPU, trafic réseau entrant, et trafic réseau sortant. Ceci concerne tous les conteneurs au niveau de notre cluster, c.-à-d. les conteneurs du déploiement SDC et ceux du système de collecte lui-même.

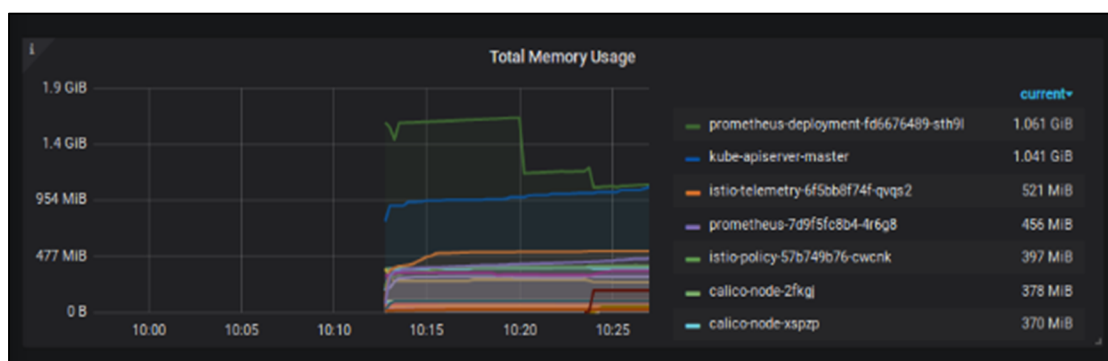


Figure 4.7 Consommation mémoire des conteneurs au niveau du cluster Kubernetes sur « K8s Container »

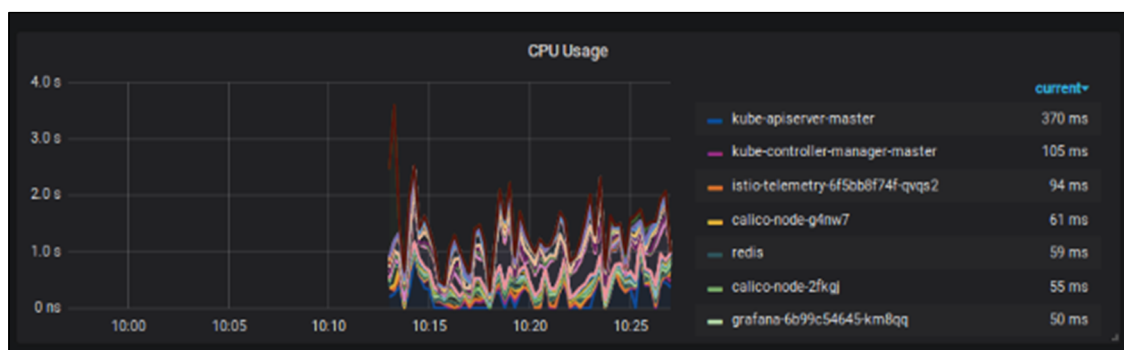


Figure 4.8 Consommation CPU des conteneurs au niveau du cluster Kubernetes sur « K8s Container »

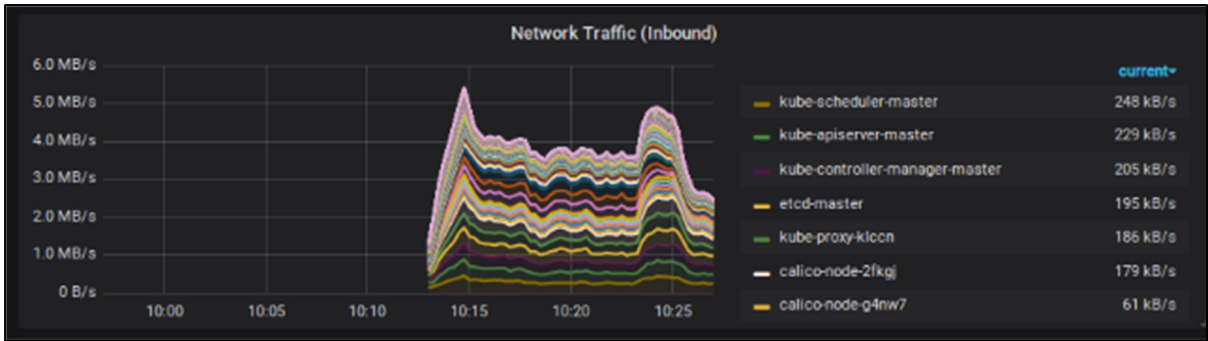


Figure 4.9      Trafic réseau entrant des conteneurs au niveau du cluster Kubernetes sur « K8s Container »

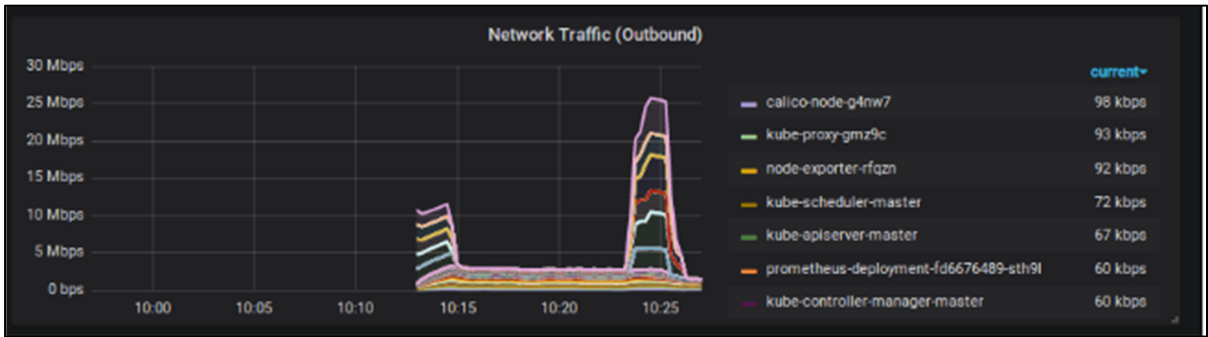


Figure 4.10      Trafic réseau entrant des conteneurs au niveau du cluster Kubernetes sur « K8s Container »

2. **Consommation du système en ressources** : la consommation des ressources du système de collecte en mémoire et CPU est récupérée via notre tableau de bord Grafana « Cluster health » (figure 4.11 et figure 4.12).



Figure 4.11 Consommation CPU et mémoire du cluster Kubernetes avant le déploiement du système de collecte télémétrique

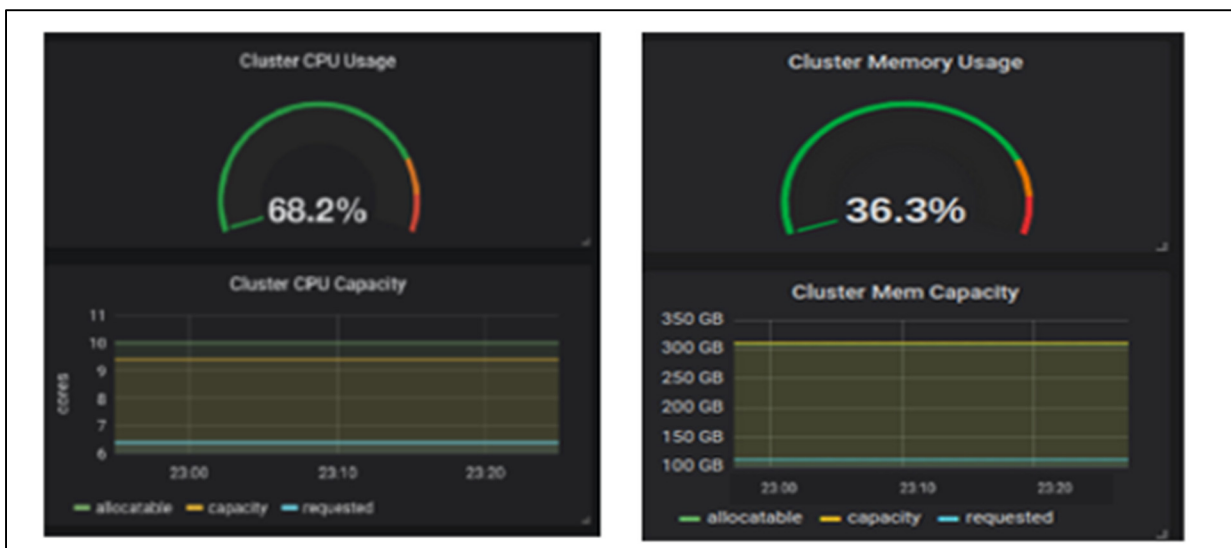


Figure 4.12 Consommation CPU et mémoire du cluster Kubernetes après le déploiement du système de collecte télémétrique

3. **Évoluer le déploiement NFV** : la figure 4.13 récupérée de « Kubernetes Dashboard » afin de vérifier notre nouveau déploiement SDC augmenté à dix server web énuméré de « clover-server 1 » à « clover-server 10 ».

| Name                   | Labels                                       | Cluster IP     | Internal endpoints                                                                   |
|------------------------|----------------------------------------------|----------------|--------------------------------------------------------------------------------------|
| ✓ clover-server9       | app: clover-server9                          | 10.96.215.62   | clover-server9:50054 TCP<br>clover-server9:9180 TCP                                  |
| ✓ clover-server10      | app: clover-server10                         | 10.106.157.132 | clover-server10:50054 TCP<br>clover-server10:9180 TCP                                |
| ✓ clover-server8       | app: clover-server8                          | 10.105.63.187  | clover-server8:50054 TCP<br>clover-server8:9180 TCP                                  |
| ✓ clover-server6       | app: clover-server6                          | 10.104.161.139 | clover-server6:50054 TCP<br>clover-server6:9180 TCP                                  |
| ✓ clover-server7       | app: clover-server7                          | 10.108.1.84    | clover-server7:50054 TCP<br>clover-server7:9180 TCP                                  |
| ✓ http-lb-v2           | -                                            | 10.101.41.128  | http-lb-v2:50054 TCP                                                                 |
| ✓ http-lb-v1           | -                                            | 10.110.237.104 | http-lb-v1:50054 TCP                                                                 |
| ✓ http-lb              | app: http-lb                                 | 10.111.105.190 | http-lb:9180 TCP                                                                     |
| ✓ proxy-access-control | app: proxy-access-control                    | 10.111.97.227  | proxy-access-control:50054 TCP<br>proxy-access-control:9180 TCP                      |
| ✓ snort-ids            | app: snort-ids                               | 10.106.195.124 | snort-ids:50052 TCP<br>snort-ids:50054 TCP<br>snort-ids:6379 TCP<br>snort-ids:80 TCP |
| ✓ clover-server5       | app: clover-server5                          | 10.110.128.90  | clover-server5:50054 TCP<br>clover-server5:9180 TCP                                  |
| ✓ clover-server4       | app: clover-server4                          | 10.107.161.253 | clover-server4:50054 TCP<br>clover-server4:9180 TCP                                  |
| ✓ clover-server3       | app: clover-server3                          | 10.103.122.132 | clover-server3:50054 TCP<br>clover-server3:9180 TCP                                  |
| ✓ clover-server2       | app: clover-server2                          | 10.106.89.7    | clover-server2:50054 TCP<br>clover-server2:9180 TCP                                  |
| ✓ redis                | -                                            | 10.107.130.162 | redis:6379 TCP                                                                       |
| ✓ clover-server1       | app: clover-server1                          | 10.105.46.159  | clover-server1:50054 TCP<br>clover-server1:9180 TCP                                  |
| ✓ kubernetes           | component: apiserver<br>provider: kubernetes | 10.96.0.1      | kubernetes:443 TCP                                                                   |

Figure 4.13 Le déploiement SDC sur Kubernetes Dashboard  
avec dix serveurs web

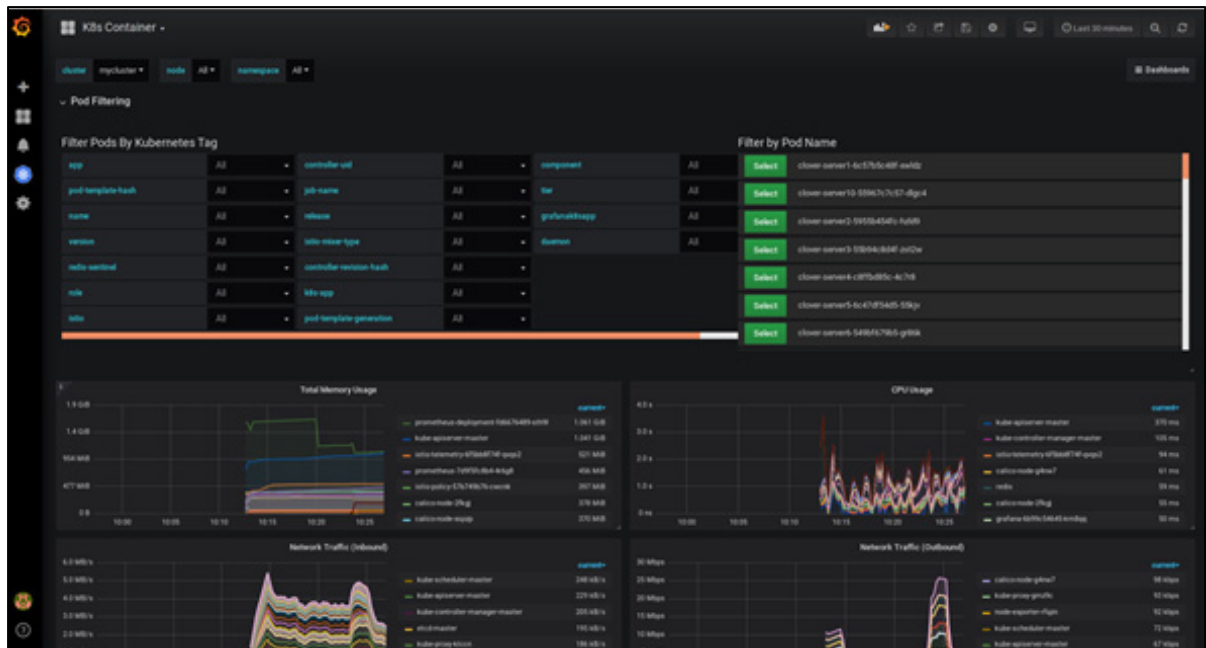


Figure 4.14 Le tableau de bord « K8s Container » après le déploiement du SDC a dix serveurs web.

#### 4.4.2 Discussion

Les résultats des figures 4.7, 4.8, 4.9, et 4.10 montrent que notre solution de collecte est fonctionnelle et les métriques de conteneurs récupérées sont visualisées sur le tableau de bord « k8s container » sur l'outil Grafana. Ces métriques sont celles ciblées au début de notre analyse : consommation CPU, consommation mémoire et bien d'autres informations concernant tous les conteneurs du cluster de déploiement (Trafic réseau entrant figures 4.9 et 4.10).

La comparaison entre la consommation CPU et mémoire du cluster Kubernetes avant et après le déploiement du système de collecte télémetrique a montré que notre solution a utilisé 8,3 % du CPU du cluster et 18,62 % de mémoire.

Notre solution a été capable de récupérer les informations en redéploiement SDC évolué en doublon ces serveurs web (figures 6.13 et 6.14).

## **4.5 Conclusion**

Dans ce chapitre, nous avons pu tester et étudier la capacité de notre solution de collecte télé-métrique à répondre aux objectifs attendus comme la récupération des métriques conteneurs en temps réel avec fiabilité et performance. Les résultats ont été prometteurs, mais il reste à étudier la partie où ces métriques collectées sont utilisées dans un processus d'apprentissage automatique pour faire de la prédiction. Ce dernier point est le sujet de notre chapitre suivant.

## CHAPITRE 5

### UN SYSTÈME DE PRÉDICTION POUR KUBERNETES BASE SUR L'APPRENTISSAGE AUTOMATIQUE

#### 5.1 Introduction

Afin d'améliorer la collecte télémétrique et la visibilité d'un déploiement sur une plateforme Kubernetes au niveau des mini-centres de données NFV, nous avons proposé une architecture microservices (figure 2.3) capable à prendre en charge une masse d'informations essentielles pour une observabilité complète et efficace ainsi que pour la surveillance des systèmes NFV et la plateforme hébergent (les clusters Kubernetes). Cette solution inclut un module de diffusion et de traitement utilisant l'outil Apache Spark qui est muni d'un composant MLib assurant la fonction de l'apprentissage automatique. L'idée est d'utiliser les données collectées, diffusées et visualisées pour prédire d'autres informations.

En termes de gestion NFV : « la demande de ressources des VNF peut être prédite à l'avenir pour gérer efficacement la qualité de service (QoS) et les problèmes d'allocation des ressources. Par conséquent, les chercheurs étudient et construisent un système de gestion où les prédictions basées sur l'apprentissage automatique des informations VNF sont utilisées pour gérer la mise à l'échelle automatique, le déploiement et la migration des VNF » (Kim et al., 2019).

Pour évaluer et juger que notre télémétrie collectée a été efficace et utile pour prédire la consommation de ressource, à titre d'exemple, nous avons utilisé une solution d'un projet antérieur de Khelifi (2019) fonctionnant sur des conteneurs Docker qui utilise aussi Spark et son module d'apprentissage automatique (Machine Learning).

Le chapitre commencera par une description du système de prédiction pour Docker du projet (Khelifi, 2019). La seconde partie est une expérimentation dont l'idée est d'exporter les informations collectées au niveau de notre cluster Kubernetes sur SymKloud comme un dataset à ce système dédié et déployé sur Docker afin d'essayer de prédire d'autres informations.

## **5.2 Objectif**

L'objectif de cette expérimentation est d'évaluer la qualité de notre télémétrie d'un cluster Kubernetes via notre solution (figure 3.3). Ainsi, il serait intéressant que ces données collectées soient utiles pour une prédiction efficace de certaines données. Dans le cas de cette étude, nous nous intéressons à la prédiction de la consommation de ressources des conteneurs (CPU et Mémoire).

## **5.3 Un Système de prédiction sur Docker basé sur l'apprentissage automatique**

Pour garantir l'évolutivité et la disponibilité, Kubernetes est déjà muni d'un mécanisme qui prend en charge les algorithmes de mise à l'échelle automatique de base pour la gestion de ressources des conteneurs de docker. Cependant, il ne fournit pas cette cohérence forte et peut corrompre l'état de l'application. Il s'agit d'un mécanisme de provisionnement dynamique de ressources naïf qui ne prend en compte que l'utilisation du processeur et n'est donc pas efficace, car de nombreux facteurs affectent les performances de l'application.

Afin de remédier à ce problème, le projet (Khelifi, 2019) a fourni une solution avec une architecture microservices pour la collecte de big data, transfert, traitement et prévision de la demande future de ressources (CPU et Mémoire) en utilisant un système de prédiction basé sur l'apprentissage automatique. Le fonctionnement est automatisé et permet d'optimiser le temps de stockage et de transfert des données afin de pouvoir faire circuler l'information en temps réel.

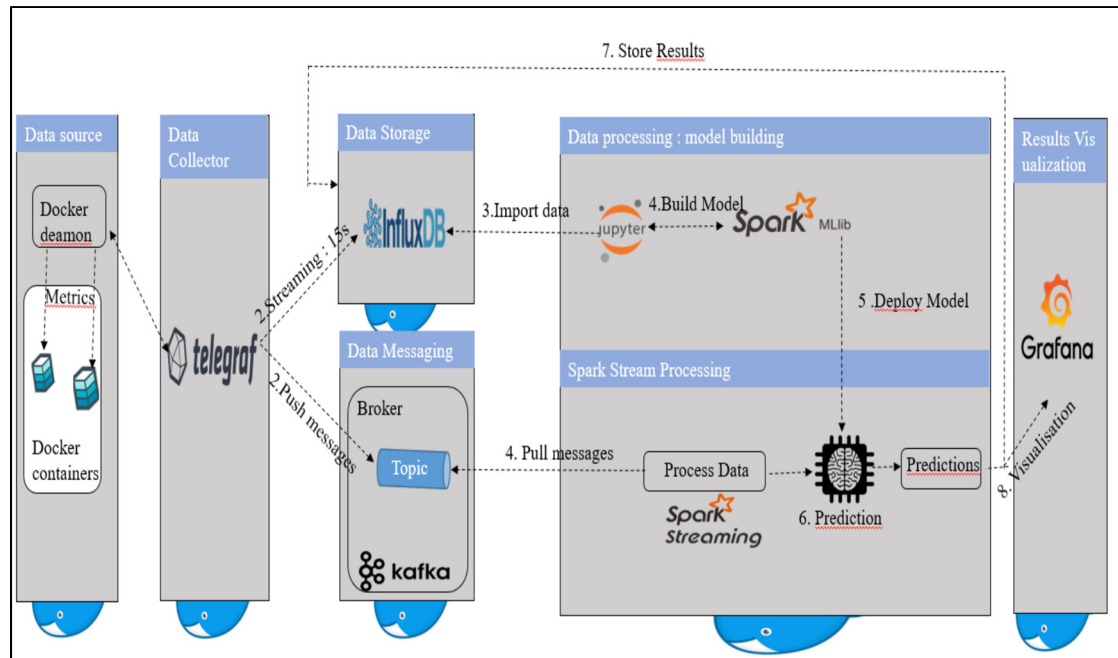


Figure 5.1 Architecture microservices proposée

Tiré de Khelifi (2019, p.41)

L'architecture telle présentée dans la figure 5.1 a été conteneurisée. L'auteur du projet (Khelifi, 2019) a utilisé Docker-compose pour faire fonctionner les conteneurs docker et assurer la connexion entre eux.

### 5.3.1 Construction du modèle machine Learning

Arrivant à la phase de construction du modèle machine Learning, nous mentionnons que deux conteneurs de stress "cpu100" et "progrum / stress" ont été créés. Ces deux derniers causent un stress CPU et mémoire et leur utilisation des ressources est variable afin que des données importantes soient obtenues. La construction du modèle se fait en plusieurs étapes :

1. Préparation de données prédictives.
2. Construction, formation et validation du modèle de prédiction.
3. Déploiement du modèle de prédiction.

### 5.3.1.1 Préparation de données :

La préparation de données se réalise en plusieurs étapes :

1. **Ingénierie des fonctionnalités** : préparer les données est très important afin d'éviter un mauvais impact sur la performance du modèle d'apprentissage. Les données collectées sont des séries chronologiques et pour les simplifier et les rendre plus efficaces, des fonctionnalités supplémentaires ont été engendrées.



| Time                | container_name | CPU_usage            | Memory_usage        |
|---------------------|----------------|----------------------|---------------------|
| 2019-05-10 12:58:01 | telegraf       | 2.7908964303178485   | 0.0828499192876676  |
| 2019-05-10 12:58:01 | jupyter        | 0.005513193588162762 | 0.6887452364981904  |
| 2019-05-10 12:58:01 | grafana        | 0.029132163164400492 | 0.09931179530271424 |
| 2019-05-10 12:58:01 | kafka          | 7.665678522167488    | 8.147645828880028   |
| 2019-05-10 12:58:01 | influxdb       | 1.0495354264524104   | 2.4416893324586426  |
| 2019-05-10 12:58:01 | zookeeper      | 0.17841698765432099  | 0.7143471391962182  |

| Year | Month | Day | Second | dayofweek | container_name | CPU_usage_min/hour   | CPU_usage_max/hour   | CPU_usage            | Memory_usage        |
|------|-------|-----|--------|-----------|----------------|----------------------|----------------------|----------------------|---------------------|
| 2019 | 5     | 10  | 1      | 6         | telegraf       | 2.7908964303178485   | 2.9470149019607845   | 2.7908964303178485   | 0.0828499192876676  |
| 2019 | 5     | 10  | 1      | 6         | jupyter        | 0.005513193588162762 | 0.005683125763125764 | 0.005513193588162762 | 0.6887452364981904  |
| 2019 | 5     | 10  | 1      | 6         | grafana        | 0.020999207920792077 | 0.029132163164400492 | 0.029132163164400492 | 0.09931179530271424 |
| 2019 | 5     | 10  | 1      | 6         | kafka          | 7.665678522167488    | 8.98305950980392     | 7.665678522167488    | 8.147645828880028   |
| 2019 | 5     | 10  | 1      | 6         | influxdb       | 1.0495354264524104   | 1.3320418811881187   | 1.0495354264524104   | 2.4416893324586426  |
| 2019 | 5     | 10  | 1      | 6         | zookeeper      | 0.1651596049382716   | 0.17841698765432099  | 0.17841698765432099  | 0.7143471391962182  |

Figure 5.2 Exemple d'extraction de nouvelles fonctionnalités

Tiré de Khelifi (2019, p.54)

La figure 5.2 montre un exemple d'obtention de nouvelles fonctionnalités à partir de la fonction "Temps". Il a été également ajouté : processeur et mémoire utilisés en terme de maximum et minimum pour chaque conteneur dans chaque heure.

2. **Indexation des variables :** « L'indexation est utilisée pour présenter les valeurs de chaîne de chaque colonne d'un DataFrame par un index. Ce travail est effectué à l'aide de la fonction "StringIndexer ()" de la bibliothèque Spark MLlib » (Khelifi, 2019, p.54).
3. **Assemblage des variables :** prenons l'exemple des quatre variables prédictives : l'indice année, mois, jour et conteneur. Ces valeurs sont rassemblées dans la colonne « caractéristiques » sous le format d'un vecteur et l'aide de la fonction "VectorAssembler ()" de la bibliothèque Spark MLlib. Elle implique les variables dans un seul vecteur afin de l'utiliser dans l'apprentissage du modèle.
4. **Division des données :** les données sont divisées en deux parties : 70 % des observations seront consacrées à la phase de formation (apprentissage du modèle) et les 30 % seront utilisés dans la partie test.

#### 5.3.1.2 Choix de l'algorithme de régression

Le modèle de régression utilisé est SparkMLlib avec l'algorithme le plus utilisé en temps réel qui est « Arbre de décision » qui garantit un temps de formation réduit, un temps d'exécution réduit et une faible complexité.

« L'algorithme d'arbre de décision connu sous son nom plus moderne "CART" est un outil d'aide à la décision représentant un ensemble de choix sous forme de graphique d'un arbre. Les différentes décisions possibles se situent aux extrémités des branches (les feuilles de l'arbre), et sont affectées en fonction des décisions prises à chaque étape » (Brownlee, 2016).

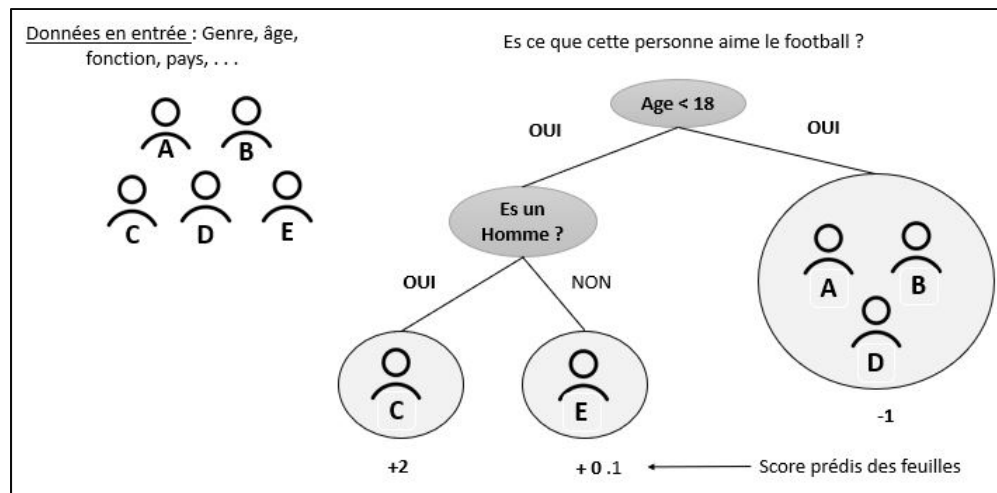


Figure 5.3 Principe du fonctionnement de l'arbre de décision « CART »

On se référant à la figure 5.3, notre « dataset » est les données de certaines personnes (Age, Gender, Occupation).

### 5.3.2 Formation et validation du modèle

En premier lieu, l'étude de Khelifi (2019) utilise la fonction de régression "DecisionTreeRegressor ()" en indiquant la colonne à prédire et la colonne contenant le vecteur de prédicteurs. Ensuite, elle indique les étapes qui concernent la préparation de données dans une chaîne de traitement (pipeline). Ce pipeline est instancié à l'aide de la fonction "Pipeline ()" qui prend en entrée les indexeurs, l'assembleur des variables prédictives et le régresseur du modèle, et se charge de l'application de ces étapes.

Cette série d'étapes est appliquée une fois aux données d'entraînement, puis aux données de test lorsque nous appelons la méthode "fit ()" effectuée sur un Pipeline. Nous pouvons ensuite appliquer un évaluateur (RegressorEvaluator) au modèle pour comparer les étiquettes prédites avec les étiquettes réelles.

Le calcul d'erreur donne une précision de 0,949 64, ce qui signifie que 95 % des données ont été correctement prédites.

Une dernière étape précisée : « Une fois que le modèle est piloté sur des données, il renvoie un objet Model qui aura généralement une méthode "predict ()". À ce niveau, nous avons sauvegardé notre modèle dans un emplacement accessible et sécurisé pour son application à un nouveau flux de données. » (Khelifi, 2019, p.57)

### **5.3.3 Déploiement du modèle de prédiction**

Enfin, le flux de données émis en temps réel par Kafka via Spark Streaming est chargé sous la forme de DataFrames. Le modèle entraîné sera téléchargé pour faire des prédictions sur les données reçues par lots en appliquant la fonction "prédire ()". L'import du modèle se fait via la fonction "load ()" en insérant et précisant le chemin de son emplacement.

## **5.4 Des données télémétriques Kubernetes**

Nous allons utiliser la solution de prédiction déployer sur Docker et former le modèle d'apprentissage automatique avec des données collecte d'un cluster Kubernetes via notre solution proposée dans le chapitre 3.

### **5.4.1 Environnement**

L'expérimentation est réalisée au laboratoire multimédia sur un système Ubuntu avec Docker installé.

### **5.4.2 Expérimentation**

Il s'agit de plusieurs étapes à réaliser comme suit :

1. Exporter les données télémétriques collectées via notre solution avec une extraction de la base de données InfluxDB (celle connectée à Apache Spark). Nous nous intéressons aux métriques conteneurs seulement.
2. Injecter la base de données InfluxDB extraite du cluster Kubernetes dans le modèle de prédiction sur Docker (figure 5.4). Ceci consiste à changer le chemin vers la base dans la configuration de Spark (sur Docker) et le fichier Docker compose qui implémente la relation entre les éléments de la solution. Nous avons aussi suspendu les liens avec Télecraf et l'ancienne base InfluxDB pour ne pas perturber notre expérimentation et s'assurer qu'aucune métrique de l'environnement Docker est ajoutée à notre étude. La partie fonctionnelle est celle en pointillé dans la figure 5.4.

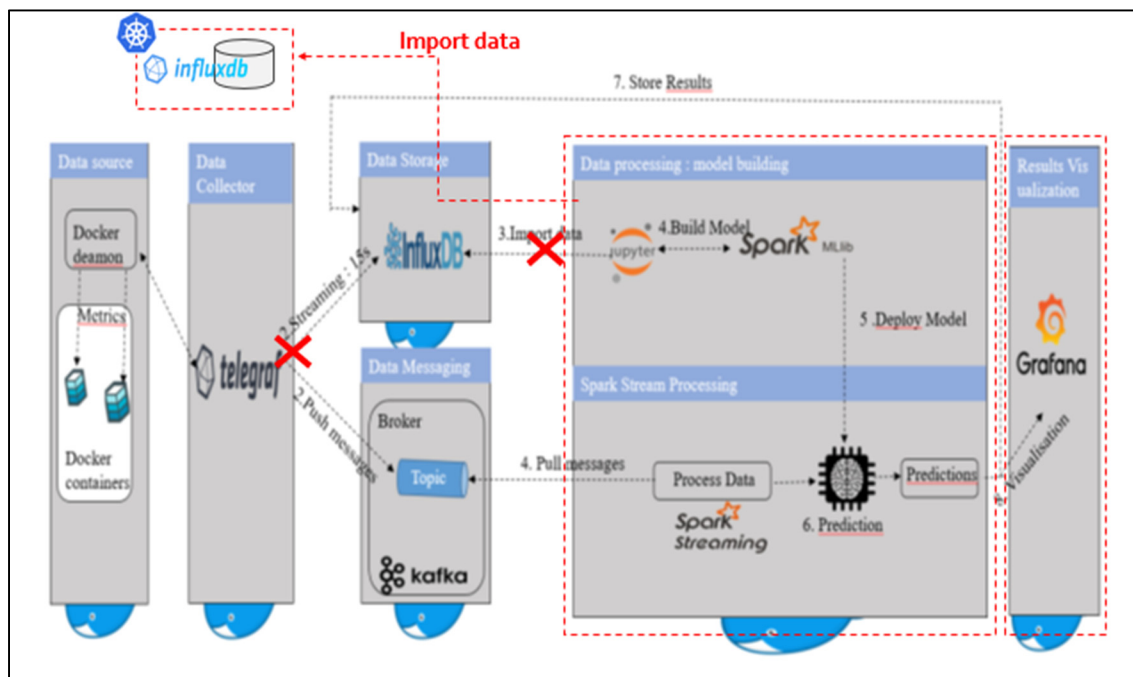


Figure 5.4 Les parties fonctionnelles de l'architecture de la solution de prédiction sur Docker de Khelifi (2019, p.41) afin d'utiliser les données des conteneurs d'un cluster Kubernetes

3. Construire le modèle d'apprentissage automatique en suivant les mêmes étapes de l'étude (Khelifi, 2019) :

A. **Préparation de données prédictives** : Nous avons préservé les mêmes étapes et principes (fonctionnalités, indexation, assemblage, et division de données) vu qu'on parle toujours des métriques conteneurs (Consommation Mémoire et CPU). La différence est que notre dataset est plus large et contient plus d'informations Kubernetes liées à l'entité conteneur (ex. : Nom du Pod où se trouve le conteneur). Il n'est pas probable que nous ayons une confusion entre les conteneurs parce que chacun a un nom unique, et par conséquent, nous garderons dans notre dataset que les métriques les plus nécessaires comme : nom du conteneur, consommation CPU, consommation mémoire, et temps. On maintient aussi le modèle de régression utilisé SparkMLlib avec l'algorithme le plus utilisé en temps réel qui est « Arbre de décision » pour son temps de formation et d'exécution réduit et sa faible complexité.

B. **Formation et validation du modèle** : La figure 5.4 présente deux courbes à la base d'un échantillon de valeurs sur un intervalle de temps pour comparer entre les valeurs réelles de la consommation CPU et la consommation mémoire, et les valeurs prédites via l'apprentissage automatique d'Apache Spark.

Le calcul d'erreur donne une précision de 0,938 97, ce qui signifie que 94 % de nos données ont été correctement prédites.

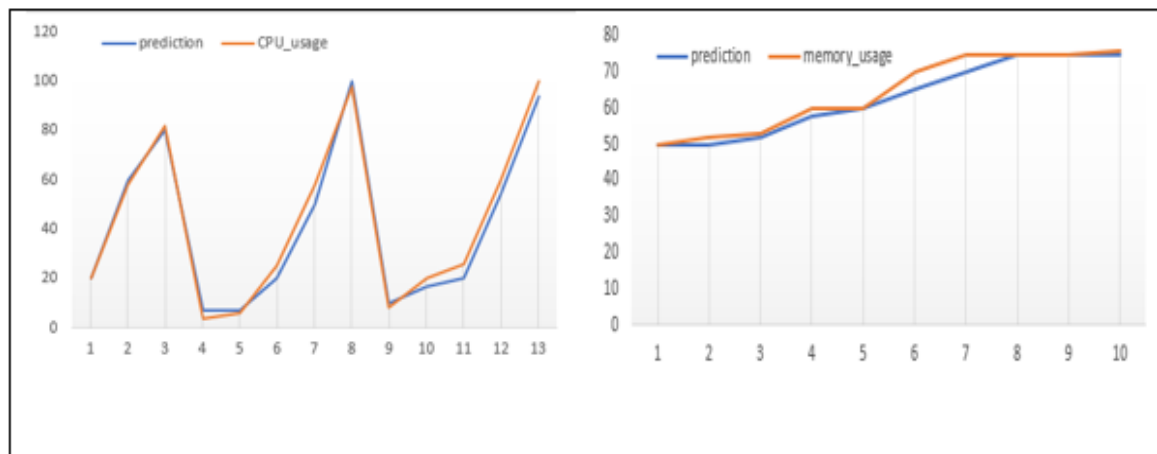


Figure 5.5 Courbes des valeurs réelles et prédites de la consommation CPU et la consommation mémoire

C. **Déploiement du modèle de prédiction** : À ce niveau, nous chargeons plus de DataFrames issues des données collectées au niveau du cluster Kubernetes, ainsi que le modèle entraîné pour faire plus de prédiction.

## 5.5 Discussion

Notre expérimentation a montré que les données télémétriques collectées via notre système de collecte NFV sur Kubernetes sont utiles et fiables pour une prédiction basée sur l'apprentissage automatique.

Il vrai que nous nous sommes limités aux métriques conteneurs et cela devrait être élargi en expérimentant la partie machine Learning de notre solution elle-même déployée sur Kubernetes et en prenons en considération un autre niveau de métriques comme la consommation des ressources pour un Pod, un Node ou un cluster au complet.

## **5.6 Conclusion**

Dans ce chapitre, nous avons montré qu'il est possible d'intégrer un module de prédiction basé sur l'apprentissage automatique en utilisant les informations télémétriques de notre solution de collecte proposée pour les déploiements NFV sur une plateforme Kubernetes



## CONCLUSION ET TRAVAUX FUTURS

La collecte télémétrique est un pilier d'une bonne observabilité. Avec la technologie de conteneurisation et de Kubernetes, il est apparu de nouvelles métriques (métriques de conteneurs, de pods, ...etc.) qui doivent être prises en considération pour une meilleure visibilité sur les déploiements NFV et les centres de données de fournisseurs de services. Afin de minimiser les risques d'échec des centres de données et des services réseaux offerts, notre recherche a abordé le défi de la collecte des nouvelles métriques dans un contexte NFV avec plusieurs contraintes (Gestion des services VNFs, surveillance, etc.).

Pour améliorer la collecte télémétrique, des solutions antérieures existent comme le travail de Khelifi (2019) et celui de Ramakrishnan (2019). Certaines ne sont pas flexibles pour répondre aux exigences de l'industrie, et d'autres n'ont pas été testées sur un environnement Kubernetes au niveau d'un centre de données dédié à la NFV. Elles ne prennent pas en considération qu'un déploiement NFV sur une plateforme Kubernetes engendre une masse de données très importantes potentiellement très coûteuse pour une entreprise qui a une limitation en ressources.

Nous avons proposé un système de collecte compatible avec la plateforme Kubernetes d'une architecture microservices. Ce système a été capable de collecter les données en temps réel avec une méthode qui cible les sources de données pour ne pas alourdir le fonctionnement. La solution est dotée d'un système de prédiction pour plus d'informations aidant à améliorer la visibilité et minimiser les risques de dysfonctionnement.

L'implémentation de notre solution de collecte télémétrique consistait à déployer chaque composante et chaque outil individuellement. C'était une phase assez complexe avec des efforts considérables afin de gérer beaucoup d'erreurs d'incompatibilités. Ceci ouvre une piste pour un futur travail d'automatisation de la conception et du déploiement du prototype de la solution.

Il serait très intéressant de Re-expérimenter la conception de la solution de collecte en mettant l'accent sur les métriques de performances et ne pas se limiter aux nouvelles ou aux bonnes métriques.

## BLIOGRAPHIE

- AWS. (s. d). Qu'est-ce que les données diffusées en continu ? Repéré à [https : //aws.amazon.com/fr/streaming-data/](https://aws.amazon.com/fr/streaming-data/)
- Biseul, X. (2019). Monitoring open source: le couple Prometheus / Grafana impose sa marque. Repéré à <https://www.journaldunet.com/solutions/dsi/1459455-quatre-solutions-open-source-de-monitoring-d-infrastructures/>
- Brazil, B. (2015). Prometheus Overview. [Présentation PowerPoint]. Repéré à [https : //www.slideshare.net/brianbrazil/prometheus-overview](https://www.slideshare.net/brianbrazil/prometheus-overview)
- Brownlee, J. (2016). Classification And Regression Trees for Machine Learning. Repéré à [https : //machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/](https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/)
- Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. IEEE Softw. vol. 32, no. 2, pp. 50–54.
- Collected. (2016). Collectd - System information collection daemon. Repéré à <https://github.com/collectd/collectd>
- Crosby, I., Hoogendoorn, M., Shnitger, T., & Tremel, E. (2018,11 decembre). MONITORING IN THE CLOUD-NATIVE ERA. The New Stack, PP 89-114.
- Cynixit, S. (2020). Kubernetes Dashboard. Repéré à [https : //medium.com/faun/kubernetes-dashboard-b7b9ecff5f15](https://medium.com/faun/kubernetes-dashboard-b7b9ecff5f15)
- Developintelligence. (2017). What does Kubernetes actually do and why use it? Repéré à [https : //www.developintelligence.com/blog/2017/02/kubernetes-actually-use/](https://www.developintelligence.com/blog/2017/02/kubernetes-actually-use/)
- Dinesh, A. (2018). Kubernetes best practices: Organizing with Namespaces [Blog]. Repéré à [https : //cloud.google.com/blog/products/gcp/kubernetes-best-practices-organizing-with-namespaces](https://cloud.google.com/blog/products/gcp/kubernetes-best-practices-organizing-with-namespaces)
- Dtadog, Inc. (2016). How to collect and graph Kubernetes metrics. Repéré à [https : //github.com/DataDog/the-monitor/blob/master/kubernetes/how-to-collect-and-graph-kubernetes-metrics.md](https://github.com/DataDog/the-monitor/blob/master/kubernetes/how-to-collect-and-graph-kubernetes-metrics.md)
- Ericsson. (2018). Ericsson NFV – Here & now, but how? [Vidéo en ligne]. Repéré à [https : //www.youtube.com/watch?v=QysjL5EqMjk](https://www.youtube.com/watch?v=QysjL5EqMjk)

- Fitzgerald, B. & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *J. Syst. Softw.*, vol. 123, pp. 176–189.
- Gil Herrera, J. & Botero, J. F. (2016). Resource Allocation in NFV: A Comprehensive Survey. 2016 in *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532.
- Goltsman, K. (2019). Introduction to Kubernetes Monitoring Architecture. Repéré à <https://medium.com/kubernetes-tutorials/introduction-to-kubernetes-monitoring-architecture-98a265e0917d>
- Google, Inc. (2017). Google SREs (O'Reilly Media). Repéré à <https://landing.google.com/sre/sre-book/toc/>
- Gour, R. (2018). Apache Kafka + Spark Streaming Integration. Repéré à <https://medium.com/@rinu.gour123/apache-kafka-spark-streaming-integration-af7bd87887fb>
- Gousios, G. (2018). Big Data Software Analytics with Apache Spark. *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pp. 542–543.
- Hale, J. (2019). Key Kubernetes Concepts. Repéré à <https://towardsdatascience.com/key-kubernetes-concepts-62939f4bc08e>
- Humble, J. (2016). Continuous Delivery vs Continuous Deployment. accessed on Mar. [blog]. Repéré à <https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>
- Humble, J. (2016). What is Continuous Delivery? Repéré à <https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>
- Humble, J. & Molesky, J. (2011). Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, vol. 24, no. 8, p. 6, 2011.
- Influxdata. (s.d). Prometheus endpoints support in InfluxDB. Repéré à [https://docs.influxdata.com/influxdb/v1.8/supported\\_protocols/prometheus](https://docs.influxdata.com/influxdb/v1.8/supported_protocols/prometheus)
- Intel Corporation. (2016). Multus CNI. Repéré à <https://github.com/intel/multus-cni>
- Istio, (s.d). Bookinfo Application. Repéré à <https://istio.io/docs/examples/bookinfo/>
- Istio. (2018). Querying Metrics from Prometheus. Repéré à <https://istio.io/latest/docs/tasks/observability/metrics/querying-metrics/>

- Kamol Hasan, M. (2019). kamolhasan /K8s-Monitoring-Pipeline. Repéré à [https : //github.com/kamolhasan/K8s-Monitoring-Pipeline/tree/master/examples/kube-state-metrics](https://github.com/kamolhasan/K8s-Monitoring-Pipeline/tree/master/examples/kube-state-metrics)
- Kamol, H. (2019). kube-state-metrics. Repéré à [https : //stackoverflow.com/questions/57574364/how-to-install-and-configure-kube-state-metrics-for-external-prometheus-server](https://stackoverflow.com/questions/57574364/how-to-install-and-configure-kube-state-metrics-for-external-prometheus-server)
- Kaplarevic, V. (2019). Understanding Kubernetes Architecture with Diagrams. Repéré à [https : //phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams](https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams)
- Khelifi, H. (2019). Management de l'innovation technologique : MIT (Rapport de recherche). Tunis : École supérieure des communications de Tunis en collaboration avec École supérieure des technologies (ETS-Montréal).
- Kim, H., Jeong, S., Lee, D., Choi, H., Yoo, J., & Hong, J.W. (2019). A Deep Learning Approach to VNF Resource Prediction using Correlation between VNFs. 2019 IEEE Conference on Network Softwarization (NetSoft), pp. 444-449
- Kontron. (2018). About Kontron. Repéré à [https : //www.kontron.com/about-kontron/company-overview/about-kontron](https://www.kontron.com/about-kontron/company-overview/about-kontron)
- Kontron. (2019). SYMKLOUD MS2910. Repéré à [https : //www.kontron.com/products/systems/cloud-systems/symkloud-ms2910.html](https://www.kontron.com/products/systems/cloud-systems/symkloud-ms2910.html)
- Kubernetes. (2016). Documentation. Repéré à [https : //github.com/kubernetes/kube-state-metrics/tree/master/docs](https://github.com/kubernetes/kube-state-metrics/tree/master/docs)
- Kubernetes. (2016). Kube-state-metrics. Repéré à [https : //github.com/kubernetes/kube-state-metrics](https://github.com/kubernetes/kube-state-metrics)
- Kubernetes. (2017). Spark example. Repéré à [https : //github.com/kubernetes/examples/tree/master/staging/spark](https://github.com/kubernetes/examples/tree/master/staging/spark)
- Kubernetes. (2020). Installer et configurer Kubectl. Repéré à [https : //kubernetes.io/fr/docs/tasks/tools/install-kubectl/](https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/)
- Kubernetes. (2020). Solution professionnelle d'orchestration de conteneurs. Repéré à <https://kubernetes.io/fr/>
- Love, C. (2017). Choosing a CNI Network Provider for Kubernetes. Repéré à [https : //chrishlovecnm.com/kubernetes/cni/choosing-a-cni-provider/](https://chrishlovecnm.com/kubernetes/cni/choosing-a-cni-provider/)
- Lwakatare, L. E., Kuvaja, p. & Oivo, M (2015). Dimensions of devops. International Conference on Agile Software Development. Springer, 2015, pp. 212–217

- Matei, Z. Chowdhury, M. Franklin, M.J. Shenker, S & Stoica, I. (2010). Spark: Cluster computing with working sets. HotCloud10, 10-10(2010), 95.
- Ng, A. (2017). What is Machine Learning? Cours Apprentissage automatique. Université de Stanford. Repéré à <https://www.coursera.org/lecture/machine-learning/what-is-machine-learning-Ujm7v>
- Nygard, M. (2007). Release It!: Design and Deploy Production-Ready Software (1re Édition). USA: Pragmatic Bookshelf.
- OpenNFV, s.d. Clover SDC Sample Configuration Guide. Repéré à [https://opnfv-clover.readthedocs.io/en/latest/release/configguide/sdc\\_config\\_guide.html#advanced-usage](https://opnfv-clover.readthedocs.io/en/latest/release/configguide/sdc_config_guide.html#advanced-usage)
- Pote, S. (2016). Spark Vs Hadoop. Repéré à <https://datasciencegyan.com/spark-vs-hadoop/>
- Prometheus. (2014). From metrics to insight. Repéré à <https://prometheus.io/>
- Puppet. (2015). State of DevOps Report. Repéré à <https://puppetlabs.com/2015-devops-report>
- Rafael, S., Kelvin, D., & Frenandes, S. (2020). NFV Data Centers: A Systematic Review. Repéré à <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8998242&tag=1>
- Ramakrishnan, K. (2019). K8s & NFV: Friends or foes? Séminaire en ligne. Intel Corporation. Repéré à <https://www.brighttalk.com/webcast/12229/347830/kubernetes-and-network-function-virtualization-nfv-friends-or-foes>
- Redhat. (2020). Que sont les microservices ? Repéré à <https://www.redhat.com/fr/topics/microservices>
- Ribenzaft, R. (2020). The Top 5 Kubernetes Metrics You Need to Monitor. Repéré à <https://epsagon.com/observability/the-top-5-kubernetes-metrics-you-need-to-monitor/>
- Saponaro, J. & Mathson, J. (2016). Collecting metrics with built-in Kubernetes monitoring tools. Blog.DataDog. Repéré à <https://www.datadoghq.com/blog/how-to-collect-and-graph-kubernetes-metrics/>
- Spaleta, J. (2019). How Kubernetes works. Repéré à <https://blog.sensu.io/how-kubernetes-works>
- Spark, A. (2019). Running Spark on Kubernetes. Repéré à <https://spark.apache.org/docs/latest/running-on-kubernetes.html>

- Sridhar, K. N., Ranganath, S., Ilangovan, D & Lin, Y. (2018). Dataplane Performance Optimization with Last-Level Cache Management. *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Verona, Italy, 2018, pp. 1-2.
- Stackshare. (2019). Elasticsearch vs Prometheus vs Zabbix. Repéré à <https://stackshare.io/stackups/elasticsearch-vs-prometheus-vs-zabbix>
- Stackshare. (2019). Prometheus vs Telegraf. Repéré à <https://stackshare.io/stackups/prometheus-vs-telegraf>
- Telefonica. (2019). prometheus-kafka-adapter . Repéré à <https://github.com/Telefonica/prometheus-kafka-adapter>
- Thakur, V. (2019). Kubernetes monitoring and alerting in less than 5 minutes. Repéré à <https://github.com/Thakurvaibhav/k8s/tree/master/monitoring>
- Tutorialspoint. (s. d). Apache Kafka - Fundamentals. Repéré à [https://www.tutorialspoint.com/apache\\_kafka/apache\\_kafka\\_fundamentals.htm](https://www.tutorialspoint.com/apache_kafka/apache_kafka_fundamentals.htm)
- Weber, W. Nepal, S., & Zhu, L (2016). Developing dependable and secure cloud applications. *IEEE Internet Comput.*, vol. 20, no. 3, pp. 74–79.
- Wilson, B. (2019). How to Setup Prometheus Monitoring On Kubernetes Cluster. Repéré à <https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/>
- Yolean. (2016). Kafka for Kubernetes. Repéré à <https://github.com/Yolean/kubernetes-kafka>



