Optimisation des performances des systèmes IdO basés sur les microservices grâce aux meilleures pratiques : revue et étude empirique

par

Yahia EL FELLAH

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE LOGICIEL
M. Sc. A.

MONTRÉAL, LE 23 AVRIL 2024

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

Mme. Naouel Moha, directrice de mémoire
Département de génie logiciel et des TI, École de Technologie Supérieure

M. Julien Gascon-Samson, codirecteur
Département de génie logiciel et des TI, École de Technologie Supérieure

M. Lokman Sboui, président du jury
Département de génie des systèmes , École de Technologie Supérieure

Mme. Ghizlane El Boussaidi, membre du jury
Département de génie logiciel et des TI, École de Technologie Supérieure

M. Yann-Gaël Guéhéneuc, examinateur externe
Département d'informatique et génie logiciel, Université Concordia

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

LE 12 AVRIL 2024

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je souhaite exprimer ma sincère gratitude envers toutes les personnes qui ont contribué, de près ou de loin, à l'élaboration de ce mémoire, que ce soit par leurs travaux, leurs idées, leurs présentations, leurs collaborations ou leurs relectures.

Tout d'abord, je tiens à exprimer ma reconnaissance envers les membres du jury qui ont accepté d'évaluer mon mémoire et qui ont consacré leur temps à une lecture attentive, fournissant ainsi des commentaires pertinents qui ont contribué à l'amélioration de mon travail.

Je tiens à exprimer ma plus profonde gratitude envers mes superviseurs pour leur disponibilité, leur encadrement et leurs conseils avisés qui ont joué un rôle déterminant dans la réussite de ce mémoire.

De plus, je souhaite exprimer ma sincère reconnaissance envers mes parents et l'ensemble de ma famille, à qui je suis profondément redevable pour leur confiance et les sacrifices consentis tout au long de ma formation académique.

Enfin, je voudrais adresser des remerciements chaleureux à tous mes amis et collègues du laboratoire de recherche pour leur précieux soutien et leur encouragement constants.

Optimisation des performances des systèmes IdO basés sur les microservices grâce aux meilleures pratiques : revue et étude empirique

Yahia EL FELLAH

RÉSUMÉ

Context : L'internet des objets (IoT) et l'informatique en périphérie (Edge) sont des concepts liés : alors que l'IdO fait référence aux dispositifs et systèmes interconnectés qui échangent des données sur Internet, l'informatique en périphérie fait référence au traitement de ces données plus près de la source, généralement sur les dispositifs eux-mêmes ou à proximité de ceux-ci. L'architecture en microservices est un style d'architecture logicielle qui structure une application comme une collection de petits services faiblement couplés. Ce style a été largement étudié et appliqué à la fois dans le monde universitaire et dans l'industrie pour le déploiement de systèmes IdO en périphérie. Toutefois, le déploiement de microservices sur la périphérie pour mettre en œuvre des systèmes IoT est notoirement difficile, et les développeurs IoT sont confrontés à de nombreux défis, liés par exemple à l'utilisation des ressources et à l'optimisation des performances.

Problème : Aucune étude exhaustive n'a identifié tous les défis liés au déploiement de systèmes IoT en périphérie. Certaines études ont abordé quelques-uns de ces défis, mais elles n'ont pas examiné en détail l'utilisation des pratiques de génie logicielle (GL) pour surmonter ces défis. En outre, nous n'avons pas identifié d'études empiriques menées pour évaluer l'impact des pratiques d'ingénierie logicielle sur les systèmes IoT déployés à la périphérie.

Objectif : L'objectif de cette étude est d'identifier les défis liés au déploiement de systèmes IoT basés sur des microservices à la périphérie. Elle vise également à identifier les meilleures pratiques de génie logicielle (GL) pour résoudre ces défis et, enfin, à évaluer l'impact de la mise en œuvre de certaines de ces pratiques dans les systèmes IoT basés sur les microservices déployés à la périphérie. Nous nous concentrons sur deux mesures de performance : le débit et la latence, tout en évaluant également les mesures d'utilisation du CPU et de la mémoire pour l'utilisation des ressources.

Méthode : Pour atteindre cet objectif, nous menons une étude à plusieurs facettes : (1) une revue systématique de la littérature (SLR) pour identifier les défis et les approches liés au déploiement de systèmes IoT basés sur les microservices ; (2) une revue de la littérature grise (GLR) pour identifier les pratiques d'ingénierie logicielle qui peuvent améliorer l'utilisation des ressources et la performance des systèmes IoT basés sur les microservices ; et (3) une étude empirique pour évaluer l'impact des pratiques sélectionnées dans (2) sur l'utilisation des ressources et la performance des systèmes IoT déployés à la périphérie (Edge) en utilisant deux versions d'une étude de cas, l'une avec et l'autre sans les pratiques sélectionnées.

Résultats : Nos résultats montrent que la plupart des défis signalés sont liés à l'utilisation des ressources et à l'optimisation des performances. Nous avons compilé un catalogue de 13 pratiques d'ingénierie logicielle, parmi lesquelles les microservices conteneurisés, l'utilisation

d'une passerelle API et l'utilisation d'une base de données par microservice sont prédominantes. L'étude empirique révèle que le système IoT basé sur les microservices présente une amélioration des performances (132 % de débit en plus et 49 % de réduction de la latence) et une économie moyenne de mémoire de 7 % et 13 % pour deux scénarios différents par rapport au système existant. Cependant, nous remarquons qu'une augmentation de la complexité du système peut conduire à une augmentation de l'utilisation totale du processeur (CPU).

Conclusion : Cette recherche fournit des indications en termes de pratiques et de mesures pour les praticiens afin d'améliorer l'utilisation des ressources et la performance des systèmes IoT basés sur des microservices dans des environnements à ressources limitées.

Mots-clés: Microservices, Conception de Systèmes IoT, Déploiement de Systèmes IoT, Bonnes Pratiques, Défis de Déploiement

Improving the Performance of Microservices-based IoT Systems through Best Practices : Review and Empirical Study

Yahia EL FELLAH

ABSTRACT

Context : The Internet of Things (IoT) and Edge Computing are related concepts : while the IoT refers to interconnected devices and systems exchanging data over the Internet, the Edge refers to the processing of this data closer to the source, typically on or near the devices themselves. Microservice is a software architectural style that structures an application as a collection of small, loosely coupled services. This style has been widely studied and applied in both academia and industry for the deployment of IoT systems on the Edge. However, deploying microservices on the Edge to implement IoT systems is notoriously difficult, and IoT developers face many challenges, for example, related to resource utilisation and performance optimisation.

Problem : No comprehensive study has identified all the challenges related to the deployment of IoT systems on the Edge. Some studies discussed some of the challenges, but they did not comprehensively discuss the use of software engineering (SE) practices to overcome these challenges. Furthermore, we did not identify empirical studies that have been conducted to assess the impact of SE practices on IoT systems deployed at the Edge.

Objective : The objective of this study is to identify the challenges related to the deployment of microservices-based IoT systems on the edge. It also aims to identify SE best practices to solve these challenges and, finally, assess the impact of implementing some of these practices in microservices-based IoT systems deployed on the Edge. We focus on two performance metrics : throughput and latency, while also assessing CPU and memory usage metrics for resource utilisation.

Method : To achieve this objective, we conduct a multi-faceted study : (1) a systematic literature review (SLR) to identify challenges and approaches related to the deployment of microservice-based IoT systems ; (2) a gray literature review (GLR) to identify SE practices that can improve the resource utilization and performance of microservices-based IoT systems ; and, (3) an empirical study to assess the impact of the selected practices in (2) on resource utilization and performance for IoT systems deployed at the Edge using two versions of a case study, one with/out the practices selected.

Results : Our results show that most of the reported challenges are related to resource utilization and performance optimization. We compiled a catalog of 13 SE practices, among which containerized microservices, the use of an API gateway, and the use of a database per microservice are prevalent. The empirical study reveals that the microservices-based IoT system shows performance improvement (132% throughput and 49% reduction in latency), and an average memory savings of 7% and 13% for two different scenarios compared to the legacy system. However, we notice that an increase in system complexity may lead to an increased total CPU usage.

Conclusion : This research provides insights in terms of practices and measures for practitioners to improve resource utilization and performance of microservice-based IoT systems in resource-constrained environments.

Keywords: Microservices, IoT System Design, IoT System Deployment, Best Practices, Deployment Challenges

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

IoT	Internet des Objets
GL	Génie Logiciel
CPU	Computing Processing Unit
KPI	Key Performance Indicators
UV	Utilisateur Virtuel
HTTP	Protocole de transfert hypertexte
RPC	Remote Procedure Calls
API	Interface de Programmation Applicative
REST	Representational State Transfer
QR	Question de Recherche

INTRODUCTION

0.1 Contexte de recherche

Chaque jour, les gadgets électroniques deviennent plus intelligents : nous portons des montres connectées qui non seulement nous donnent l'heure, mais s'assurent également que nous faisons suffisamment d'exercice tout en nous tenant informés des dernières actualités sur nos réseaux sociaux. Nous vivons dans des maisons intelligentes qui nous permettent de vérifier s'il reste du lait dans notre réfrigérateur pendant que nous faisons nos courses, ou de réguler le système de chauffage à distance afin que le salon soit chaleureux et accueillant lorsque nous rentrons du travail. Tous ces services ont un point commun : ils utilisent tous Internet pour relier leurs gadgets — le terme actuel est "Internet des objets" (IoT).

L'IoT est un concept dans lequel des dispositifs, des logiciels et des technologies de réseau sont interconnectés pour échanger des données avec divers appareils et systèmes via Internet Al-Masri (2018b). l'IoT interconnecte le monde physique avec les réseaux informatiques en utilisant des technologies telles que les capteurs, la RFID, le Bluetooth et les systèmes embarqués. Cela permet la détection, le contrôle, l'automatisation et l'analyse d'objets, ainsi que la création d'opportunités pour une efficacité et une intégration avancées dans divers secteurs et applications.

L'augmentation exponentielle du nombre d'appareils connectés rend la gestion et le traitement des flux de données IoT de plus en plus complexes (Minani, Sabir, Moha & Guéhéneuc, 2023). Cette situation met en évidence le besoin crucial d'un traitement et d'une analyse efficaces des données, compte tenu de leur volume, de leur vitesse et de leur variété. Les approches traditionnelles de l'informatique en nuage, qui consistent à envoyer toutes les données vers des centres de données centralisés, peuvent s'avérer inadéquates face à cette croissance exponentielle. En effet, ces approches peuvent entraîner des problèmes de latence, ce qui est inacceptable pour

les scénarios nécessitant une prise de décision en temps réel, comme les systèmes de surveillance de la santé, où les données des patients doivent être traitées sans délai.

L'edge computing vient compléter l'IoT en permettant un traitement et une analyse plus rapides des données à la périphérie du réseau. Le besoin de traitement en temps réel et efficace a conduit à l'adoption de l'edge computing pour les systèmes IoT, ce qui permet de distribuer et de traiter les données au plus près des appareils, favorisant ainsi une analyse en temps réel. Cependant, les ressources en périphérie peuvent être limitées, ce qui nécessite une optimisation minutieuse pour gérer le volume élevé de données, tout en répondant aux exigences de traitement en temps réel des applications critiques (Shi, Cao, Zhang, Li & Xu, 2016).

L'edge computing offre de nombreux avantages pour les systèmes IoT, notamment une latence réduite, une meilleure confidentialité des données et une diminution de la bande passante nécessaire. Toutefois, la mise en œuvre de l'edge computing présente également des défis, tels que la gestion de ressources limitées, la sécurité et la scalabilité. Pour relever ces défis, il est essentiel de développer des architectures efficaces et des algorithmes optimisés pour le traitement des données à la périphérie

0.2 Motivation

L'architecture de microservices a attiré l'attention pour sa capacité à gérer efficacement les complexités des systèmes IoT, tant dans le milieu académique que dans l'industrie (Ouyang *et al.*, 2023; Söylemez, Tekinerdogan & Kolukısa Tarhan, 2022). Par conséquent, les microservices sont devenus des *blocs de construction* fondamentaux dans de nombreuses applications IoT industrielles et constituent désormais une architecture prédominante dans la conception des systèmes IoT. Cette tendance a conduit à de nombreuses études explorant leur intégration dans les applications IoT et leurs effets (Siddiqui, Khendek & Toeroe, 2023; Rath, Mandal & Sarkar, 2022).

Bien qu'un style architectural basé sur les microservices offre des avantages dans un contexte IoT, il soulève plusieurs défis en raison des niveaux significatifs d'hétérogénéité des ressources dans les systèmes IoT (Lira, Batista, Delicato & Prazeres, 2023). En particulier, lors de la mise en œuvre de microservices dans des environnements de périphérie aux ressources limitées, plusieurs défis peuvent se poser, tels que les limitations de calcul et l'utilisation des ressources (Fowler & Lewis, 2019; Dragoni *et al.*, 2018). Par conséquent, plusieurs études (par exemple, Dragoni *et al.* (2017)) ont exploré le déploiement de microservices en périphérie et ont identifié des préoccupations et des défis clés. Surmonter ces défis et réaliser les bénéfices potentiels des microservices pour les systèmes IoT reste un problème ouvert. Cependant, il existe un manque de recherche abordant ces défis à l'aide des pratiques du génie logiciel.

Les pratiques de génie logiciel (GL) sont définies comme un ensemble de pratiques ou de techniques utilisées dans le GL pour une exécution réussie des projets logiciels et qui incluent, mais sans s'y limiter, la conception, le codage, la documentation, le déploiement, etc. (Heaton & Carver, 2015; IEEE Computer Society, 2014). Ces pratiques aident non seulement à l'exécution des projets logiciels, mais aussi à assurer leur qualité, leur maintenabilité et leur efficacité.

Parmi les pratiques de GL, la "containerisation des microservices" est devenue une pratique très prisée dans les systèmes de l'IoT. La conteneurisation implique l'encapsulation d'un microservice dans un conteneur avec son propre environnement d'exécution. Un certain nombre d'études (par exemple, Guidi, Lanese, Mazzara & Montesi (2017); Shadija, Rezai & Hill (2017); Tosatto, Ruiu & Attanasio (2015); Butzin, Golatowski & Timmermann (2016a)) ont étudié comment cette pratique est appliquée dans les contextes IoT. Ces études examinent spécifiquement comment la conteneurisation peut améliorer la gestion des ressources, en particulier sur la périphérie où les contraintes de ressources sont courantes.

Cependant, malgré le potentiel des pratiques de génie logiciel pour relever ces défis, la littérature existante présente certaines limitations. Une grande partie de la recherche actuelle reste conceptuelle, se concentrant sur les aspects théoriques de l'application des pratiques de génie logiciel aux systèmes IoT basés sur les microservices. De plus, les études empiriques disponibles ont tendance à valider des pratiques uniques à travers des études de cas limitées (Balalaie, Heydarnoori & Jamshidi, 2016), plutôt que d'examiner l'impact d'un ensemble intégré de pratiques. Par conséquent, il existe un manque de preuves empiriques complètes démontrant les avantages concrets, en termes de performance, de l'application d'un ensemble cohérent de pratiques de génie logiciel à ces systèmes. Cette lacune dans la recherche souligne la nécessité d'études plus approfondies et à plus grande échelle pour quantifier les bénéfices réels de l'adoption de ces pratiques dans les environnements edge.

0.3 Méthodologie et questions de recherche

L'objectif du mémoire est d'identifier les défis actuels liés au déploiement de systèmes IoT basés sur les microservices en périphérie, d'identifier les pratiques de génie logiciel pertinentes, et d'évaluer leur impact sur les performances des systèmes IoT basés sur les microservices. Pour atteindre cet objectif, nous menons une étude composée de trois étapes principales. Chaque étape vise à répondre à une question de recherche (QR), comme illustré dans la Figure 0.1, et décrite dans ce qui suit.

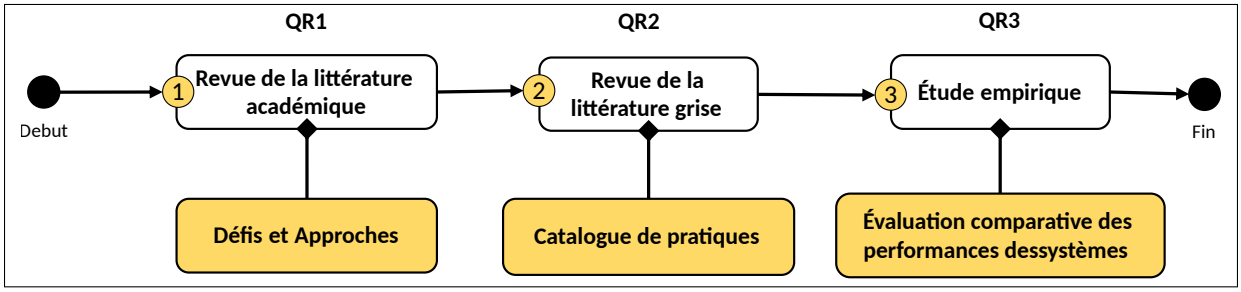


Figure 0.1 Approche d'étude à multiples facettes

QR1 : Quels sont les défis rencontrés par les développeurs de systèmes IoT lors du déploiement de systèmes IoT basés sur les microservices ?

Pour répondre à **QR1**, nous menons une revue systématique de la littérature (SLR) pour identifier les défis rencontrés par les développeurs de systèmes IoT lors du déploiement de systèmes IoT basés sur les microservices, ainsi que les approches pour atténuer ces défis. Nous avons identifié plusieurs défis liés au problème de placement (c'est-à-dire sur quel dispositif déployer chaque microservice), ainsi que des défis réseau (par exemple, optimisation de la latence et de la bande passante, gestion de différents types de connexions réseau). Cependant, peu de pratiques de génie logiciel ont été trouvées dans la SLR pour répondre aux défis de performance des systèmes IoT basés sur les microservices.

QR2 : Quelles sont les bonnes pratiques d'ingénierie logicielle qui pourraient optimiser les performances et l'utilisation des ressources des systèmes IoT basés sur les microservices en périphérie ?

Pour répondre à **QR2**, nous menons une revue de la littérature grise (GLR) pour identifier les bonnes pratiques de génie logiciel permettant d'améliorer les performances des systèmes IoT basés sur les microservices. Nous compilons un catalogue des bonnes pratiques de génie logiciel rapportées comme améliorant les performances des systèmes basés sur les microservices. Le catalogue contient des pratiques clés additionnelles comme la conteneurisation des microservices et l'utilisation du pattern de base de données par service. En identifiant et compilant un catalogue complet des bonnes pratiques pertinentes, nous établissons une base de connaissances pour guider notre évaluation empirique de pratiques sélectionnées issues du catalogue.

QR3 : Quel est l'impact sur les performances des bonnes pratiques d'ingénierie logicielle identifiées pour les systèmes IoT basés sur les microservices déployés en périphérie ?

Pour répondre à **QR3**, nous menons une étude empirique pour évaluer l'impact sur les performances de bonnes pratiques de génie logiciel sélectionnées sur un système IoT basé sur les microservices. Plus précisément, nous utilisons un système de preuve de concept appelé WIMP qui aide les étudiants à déterminer la disponibilité de leur professeur. Cette étude empirique implique la mise en œuvre de WIMP en utilisant une architecture basée sur les microservices et en adhérant aux bonnes pratiques de génie logiciel identifiées dans notre catalogue. Cette mise en œuvre sert de ressource précieuse aux chercheurs pour mener des expériences sur les systèmes IoT basés sur les microservices. Nous comparons ensuite empiriquement les performances de WIMP avec et sans les bonnes pratiques de génie logiciel sélectionnées. Les résultats démontrent les améliorations de performance réalisées en appliquant un sous-ensemble clé de bonnes pratiques, comme la réduction de la latence et de l'utilisation de la mémoire, dans un scénario concret de système IoT.

0.4 Plan du mémoire

La suite de ce mémoire est organisée comme suit : le chapitre 2 décrit le processus et les conclusions de la revue systématique de la littérature (SLR), qui vise à identifier les défis rencontrés lors du déploiement de systèmes IoT basés sur les microservices. Le chapitre 3 expose le processus et les résultats de la revue de la littérature grise (GLR), qui compile un catalogue de bonnes pratiques de génie logiciel pour améliorer les performances de ces systèmes. Les détails de la conception et des résultats de l'étude empirique, évaluant l'impact des pratiques sélectionnées sur les performances d'un système IoT basé sur les microservices, sont présentés dans le chapitre 4. Le chapitre 5 aborde les éventuelles menaces à la validité de notre étude et les mesures prises pour les atténuer.

Enfin, le dernier chapitre conclut en résumant les principales contributions de ce travail et en évoquant les perspectives de recherche future dans ce domaine.

CHAPITRE 1

NOTIONS GÉNÉRALES ET TRAVAUX CONNEXES

1.1 Notions generales

1.1.1 Internet des objets (IoT)

L'Internet des objets (IoT) fait référence à un réseau de dispositifs physiques interconnectés qui échangent des données via Internet (Leotta *et al.*, 2017). Cisco prévoit que ce réseau comptera 500 milliards d'appareils d'ici 2030 (Dias, Ferreira & Sousa, 2019), rendant ainsi la puissance informatique omniprésente au sein des systèmes IoT.

Assurer le bon fonctionnement des systèmes IoT est crucial en raison de leur impact direct sur la vie personnelle et la sécurité publique (Ahmed, Bures, Frajtak & Cerny, 2019). La croissance du nombre de systèmes IoT accroît le risque de problèmes de connectivité et d'évolutivité, soulignant ainsi la nécessité de la sécurité et de la fiabilité, notamment dans les applications critiques pour la sécurité.

Bien qu'un modèle de référence avec sept couches ait été proposé par Cisco, IBM et Intel (Cisco Systems, 2014), il n'existe pas d'architecture de système IoT universellement acceptée, car elle varie en fonction des besoins commerciaux (AltexSoft, 2020). Néanmoins, de nombreux systèmes IoT comportent quatre couches (Burhan, Muhammad and Rehman, Rana Asif and Khan, Bilal and Kim, Byung-Seo, 2018; Abdullah, Kaur & Biswas, 2020; Rao & Haq, 2018; Touqeer *et al.*, 2021) : dispositif, réseau, cloud et application.

1.1.2 Microservices

Selon Dragoni *et al.* (2017), l'émergence de l'architecture des microservices représente un récent paradigme dans la programmation des applications. Ce paradigme repose sur la création de petites unités de services, chacune opérant ses propres processus et communiquant via des mécanismes légers. Cette approche est solidement enracinée dans les fondements de l'architecture orientée

services (SOA), où l'évolution des flux de travail transfrontaliers a été intégrée au niveau des applications et de leurs architectures, illustrant ainsi une progression de la programmation orientée services de l'échelle globale à l'échelle locale (MacKenzie *et al.*, 2006).

Le terme "microservices" a été initialement présenté lors d'un atelier d'architecture en 2011 pour exprimer les idées partagées par les participants concernant les modèles d'architecture logicielle qui suivaient cette approche. Auparavant, cette approche était également identifiée sous diverses appellations.

Les microservices représentent maintenant une nouvelle tendance dans l'architecture logicielle, mettant l'accent sur la conception et le développement de logiciels hautement maintenables et évolutifs. Les microservices gèrent la complexité croissante en décomposant fonctionnellement de grands systèmes en un ensemble de services indépendants. En rendant les services totalement indépendants en développement et en déploiement, les microservices mettent l'accent sur un couplage lâche et une forte cohésion en poussant la modularité à un niveau supérieur. Cette approche offre toutes sortes d'avantages en termes de maintenabilité, d'évolutivité, et bien d'autres. Cependant, elle entraîne également un ensemble de problèmes hérités des systèmes distribués et de SOA, son prédécesseur. Ces problèmes doivent être pris en compte lors de la conception et du développement de systèmes basés sur les microservices.

1.1.3 Pratiques de génie logiciel

Les pratiques de génie logiciel (GL) désignent l'ensemble des méthodes et techniques utilisées dans le domaine du génie logiciel pour mener à bien les projets de développement logiciel. Elles englobent divers aspects tels que la conception, le codage, la documentation, le déploiement, et bien d'autres encore. L'objectif principal de ces pratiques est d'assurer le succès des projets logiciels en garantissant leur qualité, leur maintenabilité et leur efficacité.(Heaton & Carver, 2015; IEEE Computer Society, 2014)

Les pratiques de génie logiciel jouent un rôle essentiel dans la gestion et l'exécution efficaces des projets logiciels, en offrant des méthodologies structurées et des approches éprouvées pour

aborder les défis liés au développement de logiciels. Elles permettent de gérer les risques, d'optimiser les ressources et de respecter les délais et les budgets.

En adoptant les pratiques de génie logiciel, les équipes de développement peuvent bénéficier d'une meilleure collaboration, d'une meilleure communication et d'une plus grande cohérence dans leurs processus. Cela se traduit par des logiciels de meilleure qualité, plus fiables et plus faciles à maintenir à long terme.

1.1.4 Défis de déploiement dans l'IoT

L'architecture en microservices est devenue une approche populaire pour le développement et le déploiement de systèmes IoT (Dragoni *et al.*, 2018; Taibi, Lenarduzzi & Pahl, 2017), et des études antérieures ont exploré le déploiement de microservices dans des contextes IoT (par exemple, Butzin *et al.* (2016a)). Des recherches existantes (Razzaq, 2020; Campeanu, 2018) ont mené des examens approfondis des cadres architecturaux et des méthodologies pour la création et le déploiement de systèmes IoT, présentant ainsi des orientations stratégiques pour le déploiement de l'architecture en microservices dans les écosystèmes IoT. De plus, des études telles que Aksakalli,Karabey & Can,Ahmet Burak (2021) ont examiné et décrit les méthodologies de communication et de déploiement pour les microservices, et ont souligné les défis associés à travers des revues systématiques de la littérature (SLR).

Cependant, toutes les études mentionnées ci-dessus se sont concentrées sur la mise en évidence des défis liés au déploiement de microservices dans l'écosystème IoT dans un contexte général, sans aborder de manière approfondie les spécificités de certains domaines d'application.

1.1.5 Pratiques de GL dans l'IoT

Avec l'expansion de l'écosystème IoT, notamment au niveau de l'Edge, l'intérêt croît pour comprendre comment Les pratiques de génie logiciel (GL), telles que la conteneurisation, peuvent aider à relever les défis du déploiement. Des études antérieures ont exploré l'utilisation de la virtualisation basée sur des conteneurs dans les systèmes IoT Gaur, Budakoti & Lung (2018);

Alam *et al.* (2018); Guidi *et al.* (2017); Shadija *et al.* (2017); Tosatto *et al.* (2015); Butzin *et al.* (2016a). Ces études ont souligné l'importance d'adapter les implémentations de microservices et les stratégies d'orchestration de conteneurs aux exigences spécifiques du calcul en périphérie. De plus, une étude Butzin *et al.* (2016a) a exploré différents modèles et pratiques de GL spécifiques à l'approche microservice tels que "l'indépendance des services" et "l'orchestration" qui peuvent être adoptés par les systèmes IoT. De plus, le cas d'utilisation présenté par Berger, Nguyen & Benderius (2017) fournit des pratiques GL et des recommandations.

Cependant, aucune de ces études n'a fourni une liste exhaustive des pratiques de GL abordant les défis spécifiques du déploiement en périphérie. Toutes ces études ont offert des discussions et des recommandations théoriques.

1.1.6 L'évaluation des pratiques de GL dans l'IoT

Plusieurs études se sont concentrées sur l'évaluation des performances de l'architecture des microservices adoptée dans les systèmes IoT. Dans ce contexte, Lira *et al.* (2023) a examiné l'efficacité des systèmes IoT avec une architecture de microservices réactive dans divers contextes de déploiement au niveau de l'Edge. En parallèle, l'utilisation de la virtualisation basée sur les conteneurs a été explorée dans des études comme (Gaur *et al.*, 2018; Alam *et al.*, 2018), soulignant la nécessité d'adapter les microservices et les stratégies d'orchestration des conteneurs aux besoins spécifiques de l'Edge. D'autres études (Al-Masri, 2018a; Sun, Li & Memon, 2017; Macías, Navarro & González, 2019) ont proposé des conceptions architecturales et des frameworks pour répondre à des aspects de performance spécifiques. Ces études spécifiques soulignent le besoin d'explorer davantage de modèles de conception et de pratiques de GL pour établir leur efficacité dans un éventail plus large de scénarios IoT. De plus, Akbulut & Perros (2019) a analysé les performances des microservices dans divers scénarios de déploiement dans le cloud en utilisant des pratiques telles que la passerelle API, la chaîne de responsabilité et la messagerie asynchrone. Bien que cette étude soit informative, elle se limite à trois modèles de conception.

Bien que ces études aient abordé la manière dont les pratiques de GL, telles que les stratégies de déploiement et les décisions architecturales, influencent les systèmes IoT basés sur les microservices, elles offrent des perspectives limitées en se concentrant sur une pratique de GL unique ou en présentant des approches conceptuelles sans fournir de résultats empiriques de l'application de ces pratiques.

1.2 Conclusion

Les études existantes ont exploré de façon générale les défis liés au déploiement des architectures en microservices dans les systèmes IoT, ainsi que le potentiel des pratiques de génie logiciel pour relever ces défis. Cependant, ces travaux n'ont pas évalué de façon approfondie les compromis en termes de performance associés à la mise en œuvre de ces pratiques de GL pour les microservices IoT, en particulier dans un environnement Edge Computing soumis à diverses charges de travail.

CHAPITRE 2

REVUE SYSTÉMATIQUE DE LA LITTÉRATURE

Dans ce chapitre, nous effectuons une revue systématique de la littérature portant sur les défis liés au déploiement de systèmes basés sur les microservices, ainsi que les approches pour les aborder. Notre objectif est de rassembler, analyser et interpréter les différentes perspectives, recherches, et contributions académiques dans ce domaine. En examinant de manière critique les travaux antérieurs, nous identifierons les défis majeurs rencontrés lors du déploiement de systèmes basés sur les microservices, tout en explorant les approches et les solutions proposées pour les surmonter.

2.1 Stratégie de recherche

Notre étude s'est appuyée sur la méthodologie de la SLR telle que proposée par Kitchenham (2004). Nous avons choisi cette approche plutôt que d'interroger directement les praticiens pour plusieurs raisons :

1. La SLR permet une synthèse rigoureuse et exhaustive des connaissances existantes sur un sujet donné, en s'appuyant sur la littérature scientifique publiée et validée par des pairs. Cela assure une certaine objectivité et rigueur dans les résultats obtenus.
2. La SLR suit un protocole bien défini et reproductive, documentant chaque étape du processus. Cela renforce la transparence et la traçabilité de l'étude.
3. Sur des sujets techniques pointus comme les architectures en microservices pour l'IoT, la littérature scientifique est susceptible de capturer l'état de l'art des connaissances et les défis clés identifiés par la communauté de recherche, en lien étroit avec les problématiques industrielles.

Néanmoins, nous sommes conscients que combiner une SLR avec des entretiens de praticiens pourrait apporter une perspective complémentaire intéressante. Cela pourrait faire l'objet de travaux futurs pour confronter et enrichir les résultats de la présente SLR.

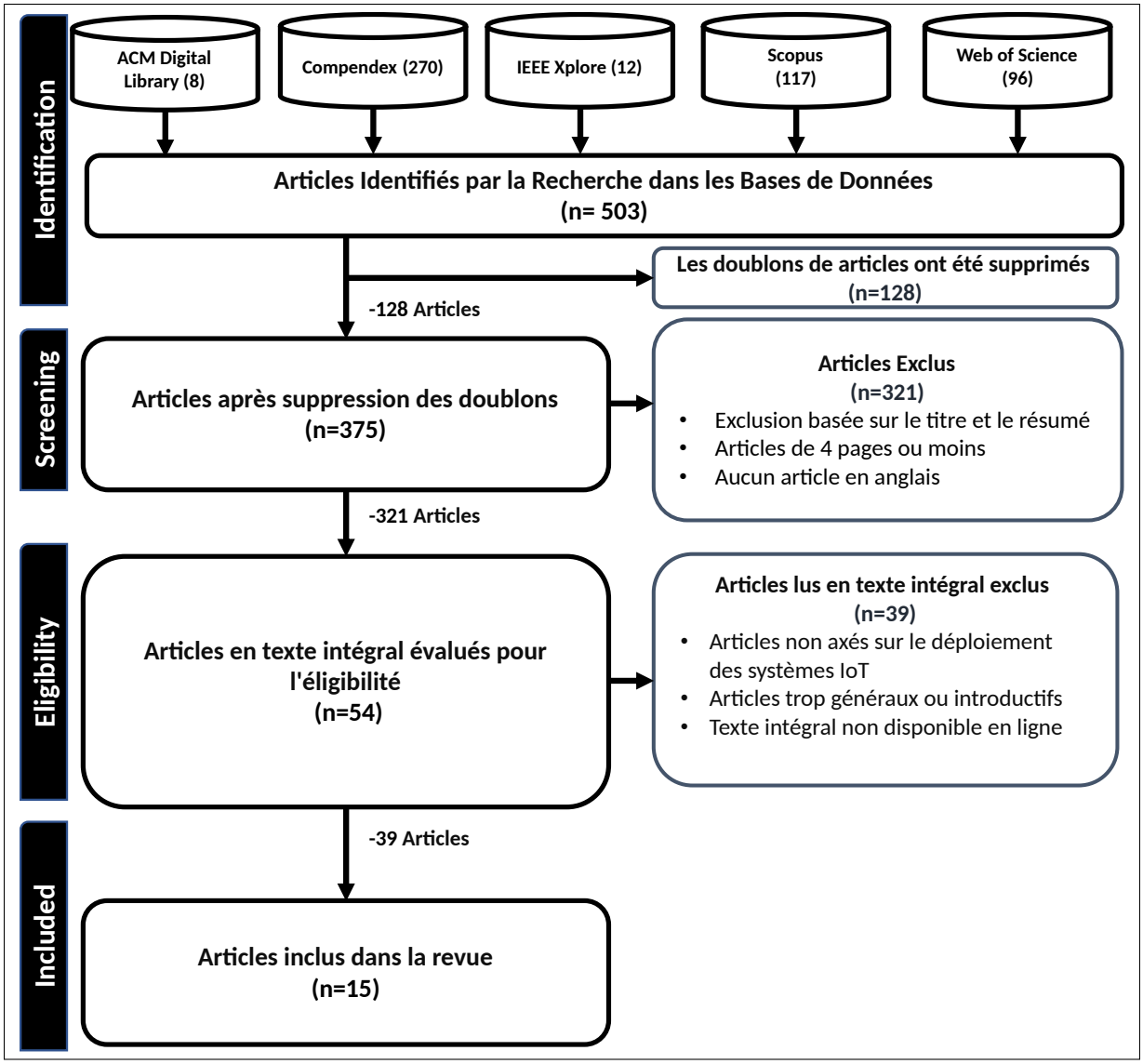


Figure 2.1 Étapes de la Revue de la Littérature

Notre objectif principal est d'identifier les défis rencontrés par les praticiens lors du déploiement de systèmes IoT basés sur des architectures en microservices. Pour atteindre cet objectif, nous posons la question de recherche suivante : ***QR1*** *: Quels sont les principaux défis auxquels sont confrontés les praticiens lors du déploiement de systèmes IoT adoptant une architecture en microservices ?*

2.2 Sources d'information et requêtes de recherche

Pour recueillir les articles, nous avons utilisé une chaînes de recherche spécifique et consulté cinq bibliothèques numériques :

1. **IEEE Xplore** [1] : une base de données de publications scientifiques et techniques de l'Institute of Electrical and Electronics Engineers (IEEE).
2. **Scopus** [2] : une base de données bibliographiques multidisciplinaire couvrant la littérature scientifique mondiale, avec des fonctionnalités d'analyse des citations.
3. **Compendex** [3] : une base de données bibliographiques spécialisée dans le domaine de l'ingénierie.
4. **Web of Science** [4] : une plateforme donnant accès à plusieurs bases de données permettant une exploration approfondie de sous-domaines spécialisés au sein de disciplines scientifiques.
5. **Bibliothèque numérique ACM** [5] : une collection complète de tous les articles publiés par l'Association for Computing Machinery (ACM).

Lors de la recherche dans ces bibliothèques numériques, nous avons optimisé nos chaînes de recherche en expérimentant différentes combinaisons et variations de mots-clés pertinents. Cette approche itérative visait à s'assurer de capturer un ensemble exhaustif d'études répondant à nos critères d'inclusion, sans omettre de publications clés sur le sujet. En peaufinant ainsi nos requêtes, l'objectif était de constituer un corpus d'articles représentatif de l'état de l'art et fournissant une base solide pour notre analyse de la littérature.

Pour mener notre SLR, nous avons formulé notre chaîne de recherche pour interroger des bibliothèques numériques en suivant le cadre PICO (Population, Intervention, Comparaison, Result) (Cooke, Smith & Booth, 2012). Les mots-clés sont identifiés en se concentrant sur des

[1] https://ieeexplore.ieee.org/

[2] https://www.scopus.com/

[3] https://www.engineeringvillage.com/

[4] https://www.webofscience.com/wos/woscc/basic-search

[5] https://dl.acm.org/

termes liés à l'essence de notre sujet de recherche, qui gravitent autour de "microservices", "deploy" et "IoT". Pour appliquer le PICO, nous avons effectué les étapes suivantes :

1. Identifier les termes clés à partir de la question de recherche.
2. Identifier des termes alternatifs pour les termes clés de l'étape précédente.
3. Appliquer l'opérateur booléen OR entre les termes clés et leurs alternatives.
4. Appliquer l'opérateur booléen AND pour combiner les expressions de l'étape précédente.

En appliquant soigneusement le cadre PICO dans notre processus de recherche, nous avons formulé la requête de recherche suivante :

(deploy OR install) AND (IoT OR Edge OR 'Internet of Things') AND (microservice)

Nous avons personnalisé la requête de recherche en fonction de la base de données cible comme indiqué dans la Table 2.1.

Tableau 2.1 Chaînes de recherche

Bibliothèque numérique	**Chaîne de recherche**	**Papers**
Compendex	(deploy OR install) AND (IoT OR Edge) AND (microservice)	270
Scopus	TITLE((deploy OR install) AND (IoT OR Edge) AND (microservice))	117
IEEE Xplore	("Document Title" :deploy OR install) AND ("Document Title" :IoT OR Edge) AND ("Document Title" :microservice)	12
WoS	TI=(deploy OR install) AND TI=(IoT OR edge) AND TI=(microservice)	96
ACM Digital Library	Title :(deploy OR install) AND"Title :(IoT OR Edge) AND Title :(microservice)"	8

➥ **WoS** :Web of Science

2.3 Critères d'inclusion et d'exclusion

Les résultats de la recherche ont initialement inclus un grand nombre d'articles. Par conséquent, nous avons défini des critères d'inclusion et d'exclusion pour restreindre la recherche aux études

pertinentes. Nous avons défini les critères d'inclusion et d'exclusion suivants pour affiner notre processus de sélection des études pour la SLR :

1. **Critères d'inclusion**
   (a) L'article est rédigé en anglais.
   (b) L'article contient 5 pages ou plus.
   (c) L'article se concentre sur le déploiement de systèmes IoT basés sur des microservices au niveau de l'Edge.
   (d) L'article fournit suffisamment de détails sur les défis du déploiement de systèmes IoT basés sur des microservices.
   (e) Le texte intégral est disponible en ligne.
2. **Critères d'exclusion**
   (a) L'article est un article qui n'a pas été évalué par des pairs.
   (b) L'article est un article qui est dupliqué.
   (c) L'article qui contient 4 pages ou moins.
   (d) L'article qui n'est pas rédigé en anglais.
   (e) L'article qui ne fournit pas suffisamment de détails sur les défis du déploiement de systèmes IoT basés sur des microservices.

Ces critères nous ont permis de supprimer les documents non pertinents et de concentrer notre analyse sur les études les plus pertinentes pour notre question de recherche.

2.4 Snowballing

Afin de compléter notre stratégie de recherche dans les bases de données, nous avons également appliqué la technique du "snowballing" (ou échantillonnage en boule de neige). Malgré une analyse minutieuse des références bibliographiques, ce processus de snowballing n'a pas permis d'identifier de nouvelles publications répondant à nos critères d'inclusion.

2.5 Extraction de données

Nous avons minutieusement examiné le texte intégral des études sélectionnées. Des informations clés ont été extraites et consignées selon un modèle de catégorisation développé au cours de la lecture des articles. Un tableau Excel, comme illustré dans le Tableau 2.2, a été utilisé pour collecter et organiser les informations de classification. Pour chaque publication analysée, ce tableau contenait des sections pour identifier l'article, son titre, l'année de publication, l'éditeur ainsi que le lieu de parution. Il récapitulait également l'objectif principal de la recherche, l'approche utilisée pour déployer des applications de microservices en lien avec l'IoT, les points clés de la méthode proposée et les contributions essentielles soulignées par l'étude. Le tableau incluait aussi les défis potentiels du déploiement évoqués dans l'article.

Tableau 2.2 Champs utilisés pour extraire les informations des articles

#	Élément de Données	Description
1	Code	Identifiant unique de la publication
2	Titre de l'Étude	Intitulé de la recherche
3	Année	Date de parution de la publication
4	Éditeur	Organisme responsable de la publication
5	Lieu	Origine de la publication (journal, conférence, etc.)
6	Référence (bibtex)	Informations bibliographiques pour citation
7	Objectif de l'Étude	Buts principaux de la recherche
8	Approche / Pratique	Méthodologie de déploiement pour les microservices IoT
9	Caractéristiques	Éléments majeurs de la méthode suggérée

Cette méthode rigoureuse assure une collecte et une analyse systématique des données importantes issues des documents que nous avons choisis pour notre étude.

2.6 Résultats

À travers cette recherche approfondie, nous avons obtenu un total de 503 articles, qui ont ensuite été soumis à un processus de sélection systématique. La sélection des articles a été effectuée en appliquant les critères d'inclusion/exclusion suivants. Suite à ce processus, nous avons réduit le nombre d'articles à 54.

Ensuite, nous avons procédé à une revue en texte intégral des 54 articles restants, ce qui nous a permis d'approfondir notre compréhension de la littérature et de nous assurer que les études sélectionnées étaient effectivement pertinentes pour notre question de recherche. Cette analyse approfondie a permis d'identifier 15 articles qui abordent spécifiquement les complexités des pratiques de déploiement et les défis rencontrés dans les systèmes IoT basés sur les microservices. Nous avons réalisé une analyse détaillée de ces articles pour répondre à **QR1**. Le Tableau I-1 présente les 15 articles inclus dans notre examen.

Ces articles sont accessibles au public dans le dossier de reproduction disponible sur le site Web de Zenodo.

2.6.1 Aperçu general

Les résultats de notre étude systématique de la littérature mettent en évidence la répartition temporelle et géographique des articles de recherche sur la gestion des microservices dans les environnements de calcul en périphérie. En ce qui concerne la répartition temporelle, nous observons une augmentation significative du volume de publications entre 2020 et 2022, ces trois années représentant à elles seules 73,33 % des articles analysés. Plus précisément, l'année 2022 se démarque avec 33,33 % des articles, ce qui souligne un pic d'activité dans la recherche. Ce schéma indique non seulement un intérêt croissant pour le domaine, mais également des avancées de recherche probables au cours de cette période.

2.6.2 Analyses

Nous avons analysé les données extraites, et nous avons décrit les approches et les défis liés au déploiement des systèmes IoT basés sur les microservices. Ces informations découlent soit des défis documentés par les auteurs, soit de notre propre interprétation du contexte des articles.

Dans notre analyse, nous avons catégorisé les résultats en fonction de l'objectif de l'étude et des défis que les auteurs ont tenté de résoudre dans leurs recherches, ce qui nous a permis de répondre à la question de recherche **QR1**. Les défis identifiés sont les suivants :

- **Défis liés au Problème de Placement**. Ce problème fait référence au défi d'optimiser le placement des microservices sur les serveurs cloud ou Edge les plus adaptés pour atteindre des objectifs d'optimisation tels que la réduction du temps de réponse moyen des demandes. Pour y remédier, diverses techniques ont été proposées, notamment des modèles mathématiques (Sami & Mourad, 2020; Tang, Guo, Cao, Tang & Li, 2022), des heuristiques (Zhao, Deng, Liu, Yin & Dustdar, 2022; Li, Tang, Xu, Guo & Zhang, 2022), des méta-heuristiques basées sur l'intelligence de groupe (Ma, Ni, Zhu & Zhao, 2019), des algorithmes évolutifs (Deng *et al.*, 2021), et des approches d'apprentissage automatique (Zhao *et al.*, 2022; Li *et al.*, 2022).
- **Défi d'Équilibrage de Charge Dynamique**. D'autres études ont introduit des cadres dynamiques pour relever ce défi. Par exemple, une approche, connue sous le nom de ProScale, fournit un composant prédictif qui utilise la méthode de la moyenne mobile simple (SMA) hautement précise et rapide pour prévoir régulièrement la charge de travail associée à chaque microservice (Cheng *et al.*, 2023). Une autre approche met en œuvre un service automatique qui tire parti d'un mécanisme de découverte de ressources pour permettre le déploiement efficace et dynamique de microservices légers (Islam, Kumar, Kovacevic & Harjula, 2021). En outre, une approche hybride de découverte de ressources a été introduite dans un cadre spécifique qui respecte les spécifications des répertoires de ressources (RD), visant à améliorer l'efficacité des dispositifs IoT à faible puissance (Sattari, Ehsani, Leppänen, Pirttikangas & Riekki, 2020).
- **Défis liés au Réseau**. Ils concernent la connectivité réseau, les problèmes de latence et de débit découlant de la nature distribuée des systèmes IoT basés sur les microservices. Des approches telles que le caching Edge, la compression de données, les protocoles de découverte de services efficaces et les stratégies de déploiement conscientes du réseau visent à atténuer ces défis. Par exemple, (Gholami *et al.*, 2022) propose une orchestration hiérarchique pour optimiser l'utilisation des ressources réseau à travers Edge, Fog et Cloud.
- **Défis liés à la Complexité de la Programmation**. Le développement de systèmes IoT basés sur des microservices décentralisés et distribués introduit une complexité par rapport aux approches monolithiques. Pour y remédier, Francisco & Donna (2018) propose un cadre

de microservices pour simplifier le développement d'applications IoT. D'autres solutions comprennent la conception axée sur le domaine, les bibliothèques réutilisables et les modèles.

- **Défi d'Orchestration et de Gestion des Ressources**. Il s'agit de la gestion efficace des ressources informatiques au niveau Edge, visant à optimiser les performances tout en minimisant la latence et les retards. Dans ce contexte, Wu, Peng, Xia, Jin & Hu (2023) ont introduit des modèles d'apprentissage automatique, y compris des réseaux neuronaux profonds, pour améliorer l'orchestration. De plus, un concept novateur appelé V-Edge (Virtual Edge) a été introduit pour relever le défi de la gestion efficace des ressources par Dressler *et al.* (2022). De plus, une approche appelée ROMA (Gholami *et al.*, 2022), a été proposée pour améliorer l'optimisation des ressources informatiques et réseau dans divers environnements informatiques Edge tout en maintenant des performances de haute qualité. De plus, Zhang *et al.* (2021) a proposé une solution basée sur l'architecture de microservices et une méthode de déploiement basée sur le modèle de mélange gaussien pour optimiser le nombre de dispositifs passerelle déployés tout en assurant l'équilibrage de charge.

2.6.3 À retenir

Les défis liés au déploiement des systèmes IoT basés sur les microservices incluent l'optimisation du placement, les problèmes réseau, la complexité de la programmation, l'équilibrage de charge dynamique, l'orchestration et la gestion des ressources. Pour y faire face, la littérature propose divers algorithmes, modèles de conception, stratégies de déploiement et approches basées sur l'apprentissage. La conception architecturale exploitant la conteneurisation, les API et les modèles pilotés par les événements ainsi que les algorithmes axés sur l'efficacité et les optimisations matérielles peuvent aider à surmonter les problèmes tels que la latence, la fiabilité et la complexité de la programmation dans les environnements Edge contraints en ressources.

2.7 Conclusion

La plupart des études se concentrent sur la proposition d'approches et de solutions particulières pour les défis rencontrés, comme des algorithmes optimisés ou des cadres ciblant un problème

spécifique tel que le placement ou la découverte. Cependant, aucun d'entre eux n'est entièrement résolu par les études existantes. Une pratique courante en génie logiciel observée dans de nombreuses solutions proposées est l'utilisation de la conteneurisation pour la mise en œuvre des microservices. Cependant, nous n'avons pas trouvé d'étude qui applique de manière exhaustive les meilleures pratiques en génie logiciel à travers l'architecture, la conception, l'implémentation, les tests, le déploiement et la surveillance pour optimiser les systèmes IoT basés sur les microservices dans leur ensemble. La littérature existante n'établit pas systématiquement de liens entre l'optimisation de l'utilisation des ressources, la latence, la fiabilité, la complexité de la programmation et d'autres mesures clés au sein des environnements Edge et les pratiques de génie logiciel. L'optimisation dans les environnements Edge reste une étude en cours.

CHAPITRE 3

REVUE DE LA LITTÉRATURE GRISE

Le but de ce chapitre est d'examiner et d'évaluer la littérature grise sur les meilleures pratiques des microservices. La littérature grise, qui inclut les documents de travail, les rapports, les livres blancs et d'autres publications non conventionnelles, offre fréquemment des études de cas, des informations spécifiques à l'industrie et des idées pratiques qui ne sont pas aisément accessibles dans les sources académiques classiques. Ce chapitre plonge dans cette base de connaissances pour trouver, rassembler et évaluer la vaste gamme de recommandations et solutions proposées dans la littérature grise afin de fournir une compréhension approfondie des meilleures pratiques pour créer et maintenir des architectures de microservices.

3.1 Stratégie de la recherche

Le principal défi inhérent à notre étude découle de notre dépendance fondamentale à la "littérature grise" (Garousi, Felderer & Mäntylä, 2019), qui comprend des recherches et des documents produits par des institutions en dehors des circuits classiques de publication académique et commerciale. Parmi les exemples courants de cette littérature, on retrouve des rapports (annuels, de recherche, techniques, de projet, etc.), des documents de travail, des publications gouvernementales, des livres blancs technique (en anglais "White paper"), des vidéos et des évaluations. Il existe un intérêt croissant pour exploiter cette littérature grise dans le domaine du génie logiciel et l'utiliser pour évaluer les pratiques actuelles. Cependant, l'utilisation de cette littérature grise comporte des risques, car ces sources sont souvent dépourvues de données scientifiques approfondies ou d'analyses approfondies (Soldani, Tamburri & Van Den Heuvel, 2018).

Pour garantir la rigueur et la reproductibilité de notre revue de littérature, nous avons suivi les bonnes pratiques définies dans les recommandations PRISMA (Page *et al.*, 2021). Ces lignes directrices décrivent en détail les étapes à suivre pour mener une revue systématique de la littérature. Plus précisément, notre stratégie de recherche a consisté à :

1. Utiliser des moteurs de recherche web généralistes pour explorer la littérature grise de manière extensive ;
2. Restreindre les types de littérature grise examinée à des publications telles que les articles de blogue, les livres blancs, les revues industrielles et les vidéos ;
3. Combiner des critères d'inclusion et d'exclusion prédéfinis avec des indicateurs de contrôle qualité pour évaluer de manière approfondie chaque source identifiée ;

Ce processus, illustré dans la Figure 3.1, nous assure d'avoir exploré de manière exhaustive la littérature disponible sur notre sujet. Dans la suite, nous utiliserons le terme "documents" pour désigner l'ensemble des sources d'information identifiées lors de cette revue systématique. Cela inclut les livres blancs, articles de blog, rapports techniques et autres documents issus de la littérature grise, en plus des articles scientifiques publiés dans des revues académiques.

Nous détaillons ici les étapes parcourues, débutant par la définition du problème, puis la collecte des pratiques pertinentes, suivi du processus de catégorisation, et enfin la sélection des pratiques et modèles privilégiés. Cette section expose la méthodologie adoptée pour élaborer une compilation exhaustive.

3.2 La définition du problème de recherche et la question de recherche

Avec la nécessité d'optimiser les performances dans des environnements de périphérie contraints en ressources, les pratiques d'ingénierie logicielle telles que les modèles indépendants de la plate-forme favorisent le développement de microservices réutilisables et simplifiés en abstrayant la logique métier des plates-formes sous-jacentes (Newman, 2021). Cela permet la portabilité sur divers matériels, facilitant ainsi l'optimisation au fur et à mesure des besoins.

Notre objectif est de se concentrer sur les pratiques et les approches d'ingénieure logicielle qui peuvent répondre à l'optimisation des performances et des ressources pour les systèmes IoT basés sur les microservices en Edge.

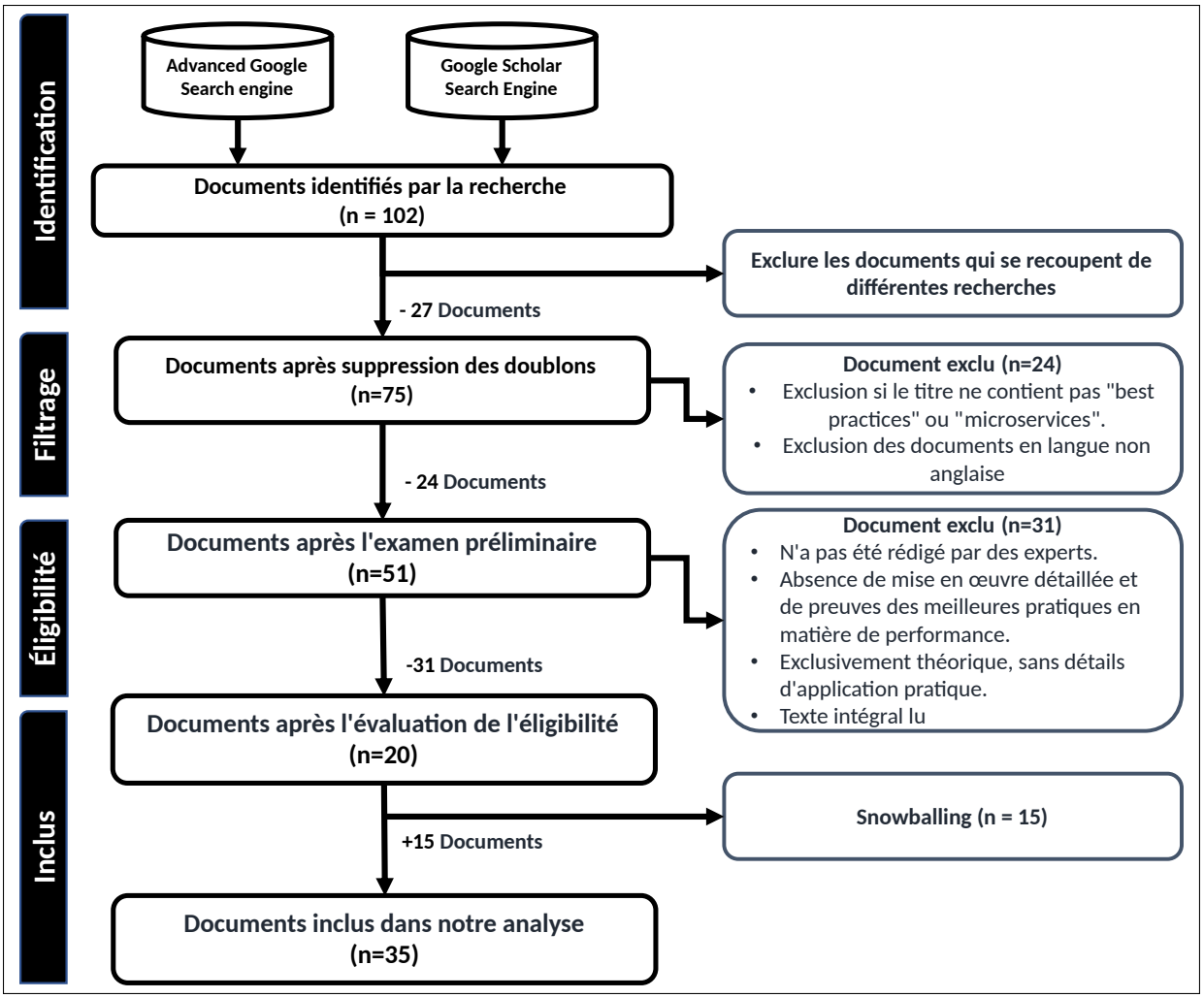


Figure 3.1 Étapes dans le processus de GLR

Notre objectif est de déduire et d'extraire les meilleures pratiques qui peuvent répondre à l'optimisation des performances et des ressources pour les systèmes IoT/basés sur les microservices en Edge. Pour faire ceci, on se concentre sur des situations pratiques et des cas documentés de réussites et d'échecs trouvés dans la littérature grise. Pendant cette étape, on vise à répondre à la question de recherche : ***QR2*** - *Quelles sont les bonnes pratiques d'ingénierie logicielle qui pourraient optimiser les performances et l'utilisation des ressources des systèmes IoT basés sur les microservices en périphérie ?*

Cette question vise à identifier de déterminer les pratiques spécifiques ont été mises en œuvre avec succès pour surmonter les obstacles courants rencontrés dans le processus de développement logiciel IoT.

3.3 Sources d'information et requêtes de recherche

Nous avons effectué une recherche exhaustive de sources non académiques pour recueillir des informations sur les meilleures pratiques en ingénierie logicielle dans le contexte des microservices en adoptant le modèle de gestion (Adams, Smart & Huff, 2017) pour classer différents types de sources et en suivant le critère "expertise de l'auteur" décrit dans les directives décrites par Garousi *et al.* (2019).

Notre stratégie de recherche a impliqué l'utilisation deux principales sources de documents : le moteur de recherche *Google* et *Google Scholar*. Nous avons choisi *Google Scholar*, car il génère beaucoup plus de résultats que les bibliothèques numériques. De même, nous avons également utilisé le moteur de recherche *Google* pour trouver des contenus non académiques. Ensuite, nous avons formulé de manière collaborative la requête de recherche suivante :

("Best practices for microservices" **AND** "Performance") **OR** ("Microservices deployment best practices" **AND** "Performance")

Nous avons utilisé des techniques de recherche avancées sur *Google*, y compris les opérateurs booléens, la recherche par phrase et les filtres pour restreindre les résultats de recherche. Lorsque plusieurs requêtes produisaient des résultats chevauchants, nous ne prenions en compte que ceux de la requête initiale. De plus, nous avons examiné attentivement le contenu apparaissant sur les cinq premières pages des résultats de recherche sur *Google* et *Google Scholar*.

3.4 Critères d'inclusion et d'exclusion

Nous avons exécuté ces requêtes séparément dans le *moteur de recherche Google* et dans le *moteur de recherche Google Scholar*, et nous avons récupéré respectivement 88 et 14 documents.

Nous avons identifié un total de 102 documents potentiels. Nous avons manuellement supprimé les doublons et avons conservé 75 documents. Nous avons appliqué les critères d'inclusion et d'exclusion suivants pour sélectionner les documents à analyser dans le cadre de cette thèse.

1. **Critères d'inclusion**
    (a) Le document doit être un article de revue, une publication de conférence ou un chapitre de livre.
    (b) Le document aborde des techniques, des principes ou des architectures lié à l'optimisation des performances dans les microservices.
    (c) Le document fournit des preuves empiriques d'améliorations de performances grâce à l'application de pratiques d'ingénierie logicielle.
    (d) Le document présente une mise en œuvre pratique et réelle des meilleures pratiques.
2. **Critères d'exclusion**
    (a) Le document n'est pas rédigé en anglais.
    (b) Le titre ne contient pas les termes "meilleures pratiques" ou "microservices".
    (c) Le document se concentre entièrement sur des concepts théoriques et manque d'exemples ou de détails.
    (d) Le document n'est pas publié dans une revue académique ou par des experts reconnus dans le domaine.

Nous avons appliqué les deux premiers critères d'exclusion et a exclu 24 documents. Après ce premier filtrage, nous avons conservé 51 documents. Nous avons ensuite appliqué les trois derniers critères d'exclusion et anovs retiré 30 documents, résolvant les désaccords par des discussions impliquant tous les auteurs de cette thèse. Nous avons finalement retenu 20 documents à cette étape.

Toutes les recherches ont été effectuées en utilisant un réseau public et une extension de navigateur qui supprime les cookies pour éviter la personnalisation du moteur de recherche.

3.5 Snowballing

Pour élargir la portée de notre revue de littérature, nous avons utilisé une technique de boule de neige où nous avons examiné les références et les citations à l'intérieur de notre ensemble initial de documents. En conséquence, nous avons inclus 15 documents supplémentaires pertinents, après avoir appliqué les mêmes critères d'inclusion/exclusion, pour examen.

3.6 Compilation d'un catalogue de pratiques et de motifs

Nous avons rassemblé toutes les pratiques ou recommandations issues des différents documents trouvés en une liste unique. Nous avons sélectionné les pratiques d'ingénierie logicielle les plus recommandées pour formuler notre catalogue de pratiques en réponse à la **QR2**.

Nous nous sommes concentrés sur les défis d'optimisation des performances et d'utilisation des ressources des systèmes IoT basés sur des microservices au niveau de l'Edge. Nous avons évalué si le document rapportait comment les pratiques sélectionnées peuvent aider à relever ces défis. Nous avons identifié et sélectionné les pratiques les plus adaptées qui s'alignent sur l'objectif principal de notre étude. Par exemple, des pratiques telles que la granularité des services, la communication asynchrone et l'équilibrage de charge sont directement liées à l'optimisation des performances dans des environnements Edge à ressources limitées. En liant les pratiques identifiées avec les défis des systèmes IoT basés sur des microservices au niveau de l'Edge.

3.7 Résultats

Dans cette section, nous procédons à l'analyse de la littérature grise sélectionnée. Dans un premier temps, nous présentons un aperçu général des sources sélectionnées (Section 3.7.1).
Par la suite, nous examinons en détail les meilleures pratiques et recommandations des praticiens dans le domaine des applications basées sur les microservices, issues de la littérature grise sélectionnée. Notre méthode pour compiler cette liste de pratiques a impliqué un examen minutieux des sources utilisées. Nous avons commencé par rassembler toutes les pratiques, analyses ou recommandations partageant des objectifs similaires. il faut noter que certains

documents mentionnent simplement leurs objectifs, tandis que d'autres proposaient des directives détaillées pour les atteindre. Cette analyse a été effectuée dans le but d'évaluer la question de recherche ***QR2*** et de résumer les conclusions tirées de notre analyse.

3.7.1 Aperçu général des documents

Dans l'ensemble, nous avons inclus 35 documents dans notre revue. Ces documents sont accessibles au public dans le package de réplication disponible sur le site Web de Zenodo. De plus, les documents que nous avons référencés dans cette étude peuvent être consultées en ligne en consultant les liens fournis dans le package de réplication. On illustre dans le Tableau 3.1 un aperçu des documents trouvés dans notre recherche.

Le Tableau 3.1 illustre notre processus de classification pour les documents trouvés en se basant sur les critères suivants : source (par exemple, site web de l'entreprise, magazine axé sur l'industrie, communauté de praticiens, blog ponctuel géré par des experts ou sommet industriel) ; type de contribution (par exemple, article de blog, livre blanc technique ou livre) ; et l'auteur dont nous vérifions l'expertise.

3.7.2 Analyse

Le Tableau 3.2 résume les pratiques que nous avons identifiées. Après notre processus de révision, nous avons réduit la liste à 13 pratiques pertinentes. Nous avons identifié les pratiques GL pour chacun des défis identifiés dans le SLR :

1. **Défis liés à la problématique de placement**. Les microservices conteneurisés sont un sujet fréquemment abordé dans la littérature scientifique en raison de leur capacité à isoler efficacement les systèmes. Les conteneurs offrent des avantages clés pour optimiser les performances et l'utilisation des ressources, comme le soulignent Turnbull (2014); Sayfan (2019). Ils permettent une utilisation minimale des ressources en partageant le système d'exploitation hôte, tout en fournissant à chaque conteneur son propre système de fichiers et

Tableau 3.1 Aperçu des documents trouvés dans notre recherche

Title	**Auteur(s)**	**Année**	**Source**	**Type de contribution**
Performance issue considerations for Microservices APIs	J. Mueller	2015	Smartbear	Blog entreprise
Performance patterns in microservices-based integrations	R. Dhall	2016	Dzone	Blog communautaire
15 Best Practices for Building a Microservices Architecture – BMC . . .	Sudip Sengupta	2023	Bmc	Blog entreprise
Microservices Best Practices for Enhanced App Performance	Irina Linnik	2024	Softteco	Blog entreprise
10 Best Practices for Microservices Deployment and Management	Mario Schaefer	2023	Xalt	Blog entreprise
Best Practices for Configuring Microservices Apps - NGINX	Javier Evans	2023	Nginx	Blog entreprise
How to Improve Performance of Microservices : Best Practices	Ranju R	2023	Sayonetech	Blog entreprise
Best Practices for Microservices : Building Scalable and Efficient . . .	Lav Kumar	2023	Dzone	Blog communautaire
Developing High-Performance Applications & Microservices	Saurabh Gupta	2022	Medium	Blog communautaire
Microservices at Netflix : Architectural Best Practices	Tony Mauro	2015	Nginx	Blog entreprise
How do you optimize microservice performance ?	Communauté LinkedIn	2023	LinkedIn	Blog communautaire
Monitoring Performance in Microservices Architecture	Joydip Kanjilal ; et al.	2023	Semaphore	Blog entreprise
Microservices Guide	Martin Fowler	n.d	Martinfowler	Blog personnel
Microservices architecture style	Microsoft Docs	n.d	Microsoft	Documentation
Designing Microservices	Sam Newman	2015	Oreilly	Livre
Microservices best practices for Java	Hofmann et al.	2016	IBM Redbooks	Livre
Microservices best practices | MuleSoft	MuleSoft Devs	2024	MuleSoft	Livre blanc

ses propres ressources isolées. Cette isolation permet d'éviter les conflits de ressources et garantit des performances prévisibles pour chaque microservice conteneurisé.

Les plateformes d'orchestration comme Kubernetes jouent un rôle crucial dans l'optimisation des performances et de l'utilisation des ressources des microservices conteneurisés. Elles facilitent la portabilité des conteneurs et leur gestion automatisée grâce à des fonctionnalités

telles que la surveillance de l'état de santé, l'allocation des ressources et le contrôle de la configuration (Anon, n.d.b). En automatisant la mise à l'échelle et l'administration des conteneurs, ces plateformes permettent une utilisation efficace des ressources, en allouant dynamiquement les ressources en fonction des besoins réels et en évitant le gaspillage. De plus, la standardisation des environnements par les conteneurs et les plateformes d'orchestration (Schaefer, 2023; Hofmann, Schnabel, Stanley & Organization, 2016a; Ranju, 2023; Bonagiri, 2023) contribue à améliorer les performances et la fiabilité des déploiements de microservices en réduisant les incohérences et les problèmes de compatibilité.

Le déploiement continu, une autre pratique pertinente en génie logiciel, vise à automatiser et à rationaliser le processus de déploiement des modifications apportées au code du système dans les environnements de production. Cette pratique intègre des pipelines prêts à être déployés et des outils tels que "Infrastructure as Code" (IaC). L'IaC aide à configurer automatiquement les environnements et à construire l'infrastructure, permettant un approvisionnement réutilisable et axé sur le code. En automatisant le processus de déploiement, cette pratique réduit les erreurs manuelles et les temps d'arrêt, améliorant ainsi les performances et la disponibilité des systèmes. De plus, l'IaC permet une allocation efficace des ressources en définissant de manière déclarative les exigences en matière d'infrastructure, ce qui garantit que les microservices disposent des ressources nécessaires pour fonctionner de manière optimale (Butzin *et al.*, 2016a; Berger *et al.*, 2017; Anon, n.d.b).

Enfin, les techniques de minification et de regroupement (Schulz, 2015; Anderson, 2022) sont conçues pour améliorer les performances des microservices en production. La minification réduit la taille des fichiers de code en supprimant les caractères inutiles, ce qui entraîne des temps de chargement plus rapides et une utilisation réduite de la bande passante. Le regroupement combine plusieurs fichiers en un seul, réduisant ainsi le nombre de requêtes HTTP nécessaires et améliorant les temps de réponse. Ces pratiques optimisent l'empaquetage et le déploiement des microservices, garantissant ainsi une haute qualité de service dans des scénarios réels. Dans le contexte des systèmes IoT basés sur des microservices en périphérie, où les ressources peuvent être limitées, ces techniques peuvent

considérablement améliorer les performances en réduisant la charge sur le réseau et en accélérant le chargement des microservices.

En résumé, les microservices conteneurisés, associés à des plateformes d'orchestration, au déploiement continu et à des techniques d'optimisation telles que la minification et le regroupement, offrent de nombreux avantages en termes de performances et d'utilisation des ressources. Ces pratiques en génie logiciel permettent d'isoler efficacement les systèmes, d'automatiser la gestion des ressources, de rationaliser les déploiements et d'optimiser l'empaquetage des microservices. En adoptant ces pratiques, les architectures de microservices peuvent atteindre des niveaux élevés d'efficacité opérationnelle, de fiabilité et de qualité de service, tout en utilisant de manière optimale les ressources informatiques disponibles.

2. **Défis liés au réseau**. Les architectures de microservices apportent de nombreux avantages, mais posent également des défis en termes de performance et de fiabilité. Il s'agit notamment des problèmes de connexion et de trafic entre les microservices. Parmi les différentes pratiques de conception, la pratique "Passerelle API" se distingue comme un choix architectural discuté dans la littérature pour relever ce défi. Elle sert de point d'entrée centralisé pour les demandes et aide à gérer le flux de trafic entre les microservices. La "Passerelle API" agit comme une façade unifiant l'accès aux différents services, permettant d'appliquer des politiques de sécurité, de surveiller le trafic et de router intelligemment les requêtes. L'absence de "Passerelle API" peut être considérée comme une mauvaise pratique (Tighilt *et al.*, 2020; Akbulut & Perros, 2019; Newman, 2021; Paul, n.d.; Scott, 2018; Hofmann, Schnabel, Stanley & Organization, 2016b). Notamment, des entreprises comme Netflix ont adopté cette pratique avec succès, ce qui souligne son importance (Blog, 2020). Netflix utilise sa "Passerelle API" pour gérer efficacement des milliards de requêtes par jour provenant de diverses plateformes clientes.

   D'autres pratiques de génie logiciel telles que "Maillage de Services (Service Mesh)", le patron "Disjoncteur (Circuit Breaker)" (Paul, n.d.), et la distribution géographique des services peuvent contribuer à réduire la latence et à améliorer la résilience, la fiabilité et la réactivité du système basé sur les microservices. Le "Maillage de Services (Service Mesh)"

fournit une couche d'infrastructure dédiée pour contrôler et observer les communications de service à service. Le patron "Disjoncteur (Circuit Breaker)" aide à prévenir les défaillances en cascade en détectant les erreurs et en empêchant l'appel de services défectueux. La distribution géographique des services sur plusieurs régions permet de rapprocher les services des utilisateurs, minimisant ainsi la latence et atténuant l'impact des défaillances localisées. En outre, un bon acheminement des demandes, un équilibrage de charge efficace (Raili & Savi, 2021) et un couplage lâche sont d'autres considérations liées aux pratiques de génie logiciel pour atténuer ces défis. Un routage intelligent peut diriger les requêtes vers les instances de service les plus performantes. L'équilibrage de charge répartit le trafic uniformément. Enfin, un couplage lâche réduit les dépendances entre services, améliorant la résilience.

Certaines études (par exemple, Lira *et al.* (2023); Gupta (2022); Sengupta (2023)) recommandent l'utilisation de la communication asynchrone entre les microservices et de la mise à l'échelle automatique pour améliorer les performances. La communication asynchrone, basée sur des événements et des files d'attente de messages, découple les services, permettant une exécution indépendante et une meilleure résilience aux pannes. La mise à l'échelle automatique ajuste dynamiquement le nombre d'instances de service en fonction de la charge, optimisant l'utilisation des ressources. Ces recommandations peuvent aider à résoudre les problèmes courants des réseaux de microservices tels que la complexité, la congestion, les défaillances en cascade et l'efficacité. De plus, des protocoles tels que "RPC (Remote Procedure Call)" offrent des avantages en termes de performances grâce à leur utilisation de HTTP/2 pour des flux efficaces et multiplexés et une latence réduite. Les tampons du protocole RPC permettent d'obtenir des messages de taille compacte, ce qui accélère la transmission et économise la bande passante. Grâce à la prise en charge du flux de données et à des contrats de service clairement définis, le protocole RPC optimise la communication interservices, ce qui le rend très efficace pour les microservices et permet d'améliorer les performances (LinkedIn community, 2023; Bolanowski *et al.*, 2022; Gancarz, 2023). Enfin, d'autres techniques, telles que l'optimisation de la taille de la charge utile, la mise en commun des connexions, la limitation du taux (Mueller, 2015), et l'activation de la

tolérance aux pannes (Dhall, 2016; Schaefer, 2023) peuvent contribuer à des systèmes plus efficaces et plus fiables. L'optimisation de la taille de la charge utile réduit le trafic réseau. La mise en commun des connexions réutilise les connexions pour économiser les ressources. La limitation du taux protège les services de la surcharge. La tolérance aux pannes, via des mécanismes tels que les réessais et les disjoncteurs, améliore la résilience.

3. **Défis liés à la complexité de la programmation**. Nous avons identifié des pratiques en génie logiciel dans certains documents (Anon, n.d.b; Humble & Farley, 2010; Martin, 2014; Kim, Humble, Debois, Willis & Forsgren, 2021) qui ont un impact sur la performance et l'utilisation des ressources dans les architectures de microservices. L'adoption de méthodologies DevOps, avec une emphase sur la gestion de la configuration et le contrôle des versions, permet d'optimiser le déploiement et la maintenance des microservices, réduisant ainsi les temps d'arrêt et améliorant l'utilisation des ressources (Humble & Farley, 2010; Kim *et al.*, 2021). Le suivi des modifications du code via le contrôle de version (Martin, 2014) facilite la résolution des problèmes de performance en permettant de revenir à des versions antérieures plus stables et efficaces.

   De plus, l'adoption d'une stratégie de gestion de la configuration comme le versionnement mono-repo, illustrée par Google (Potvin & Levenberg, 2016), peut avoir un impact positif sur les performances. En consolidant les référentiels de code en un seul, on réduit la complexité et on accélère les temps de compilation et de déploiement, ce qui se traduit par une utilisation plus efficace des ressources de calcul.

   Les leaders de l'industrie comme Netflix (Anon, n.d.b) soulignent également l'importance de maintenir des microservices cohérents et performants. Lorsqu'il devient nécessaire d'introduire de nouvelles fonctionnalités, la création de nouveaux microservices dédiés est préconisée. Cette pratique évite de surcharger les microservices existants, préservant ainsi leur stabilité et leurs performances. En répartissant la charge de travail sur de nouveaux services, on optimise l'utilisation des ressources et on maintient des temps de réponse rapides.

   En adoptant ces pratiques de génie logiciel axées sur la performance, les organisations peuvent créer des architectures de microservices efficaces et évolutives. L'alignement sur les

principes DevOps permet de rationaliser les processus de déploiement et de maintenance, réduisant ainsi les gaspillages de ressources.

4. **Défis liés à l'optimisation des ressources**. Nous avons identifié diverses pratiques en génie logiciel pour le stockage, la récupération et la synchronisation des données, ainsi que pour assurer la cohérence et la résilience des données dans les systèmes distribués, tout en optimisant les performances et l'utilisation des ressources. L'ajout de couches de mise en cache peut contribuer à réduire les demandes redondantes adressées aux bases de données ou aux services externes, afin d'éviter la répétition d'opérations gourmandes en ressources, améliorant ainsi les performances et l'utilisation des ressources. Cette pratique permet de servir rapidement les données fréquemment demandées à partir de la mémoire cache, réduisant ainsi la charge sur les composants backend et améliorant les temps de réponse. L'une des pratiques en génie logiciel, comme le soulignent Richardson (2019) et Anon (n.d.b), est la "Pratique architecturale de la base de données par service", qui permet à chaque service de fonctionner, de s'étendre et de se déployer indépendamment, évitant ainsi un couplage étroit avec d'autres services et optimisant les performances et l'utilisation des ressources. Cette approche permet une mise à l'échelle granulaire de chaque service en fonction de ses besoins spécifiques en ressources, évitant ainsi le sur-provisionnement ou le sous-provisionnement. En outre, Tighilt *et al.* (2020) considèrent cette pratique comme une solution pour traiter l'anti-modèle connu sous le nom de "Shared Persistence", qui maintient la séparation des préoccupations et optimise la modularité, réduisant ainsi les conflits de ressources et améliorant les performances globales du système.

   Une autre pratique pertinente est le "Command Query Responsibility Segregation (CQRS)". Cette pratique est conseillée pour les systèmes connaissant de gros volumes de trafic (Kumar, 2023; MuleSoft Devs, 2024), et elle permet de séparer les opérations de lecture et d'écriture, ce qui peut conduire à un traitement plus optimisé et plus efficace des opérations sur les données, améliorant ainsi les performances et l'utilisation des ressources. En séparant les responsabilités, CQRS permet d'optimiser indépendamment les performances de lecture et d'écriture, en allouant les ressources de manière appropriée à chaque type d'opération. Les performances de la base de données bénéficient également d'un réglage et d'une indexation

méticuleux (Gupta, 2022), garantissant un traitement efficace des requêtes et une utilisation optimale des ressources. Un réglage adéquat des paramètres de la base de données et la création d'index appropriés peuvent considérablement réduire les temps de réponse des requêtes et minimiser l'utilisation des ressources, en évitant les balayages complets de table et en ciblant efficacement les données pertinentes.

Ensemble, ces pratiques visent à éliminer les tâches redondantes ou inefficaces, à réduire les conflits, à permettre une mise à l'échelle indépendante et à améliorer les performances, en se concentrant sur l'utilisation optimale des ressources informatiques en structurant l'architecture et le système pour éviter les gaspillages et les goulets d'étranglement. L'objectif est de garantir une utilisation efficace des ressources de calcul, de stockage et de réseau, en répartissant intelligemment la charge de travail et en minimisant les surcharges inutiles.

Une autre pratique clé pour l'optimisation des ressources est l'utilisation des "Techniques de Minification et de Bundling" (Schulz, 2015; Anderson, 2022), qui permet d'optimiser le processus de construction en minifiant le code et en regroupant les ressources, ce qui permet de réduire la taille des artefacts. Cela réduit la quantité de données qui doivent être transférées et chargées par le système, optimisant ainsi l'utilisation de la bande passante, l'utilisation de la mémoire et les temps de chargement. La minification élimine les caractères inutiles du code source, tels que les espaces, les commentaires et les sauts de ligne, tandis que le regroupement combine plusieurs fichiers en un seul, réduisant ainsi le nombre de requêtes HTTP nécessaires. L'amélioration de la livraison, de la minification et du regroupement des ressources conduit à une utilisation plus efficace des ressources, en minimisant les transferts de données et en accélérant le chargement des applications.

En outre, le maintien d'un conteneur unique pour exécuter tous les composants du système est une autre pratique en génie logiciel liée aux politiques de déploiement, qui garantit un équilibre optimal entre les performances et la consommation d'énergie (Lira *et al.*, 2023). En consolidant tous les composants dans un seul conteneur, on réduit la surcharge liée à la gestion de plusieurs conteneurs et on optimise l'utilisation des ressources sous-jacentes. Cette approche permet également de mieux contrôler l'allocation des ressources et de

s'assurer que les composants fonctionnent de manière efficace et coordonnée, tout en minimisant la consommation d'énergie globale du système.

En adoptant ces pratiques en génie logiciel axées sur les performances et l'utilisation des ressources, les architectures de microservices peuvent atteindre des niveaux élevés d'efficacité opérationnelle et de rentabilité. En optimisant le stockage, la récupération et la synchronisation des données, en réduisant les tâches redondantes, en permettant une mise à l'échelle granulaire et en minimisant les surcharges, ces pratiques contribuent à la création de systèmes hautement performants et économes en ressources.

Tableau 3.2 Liste exhaustive des pratiques en génie logiciel

#	Titre des pratiques	N.de.refs
1	Utiliser des pipelines de déploiement continu (CD) et des pipelines d'automatisation	9
2	Utiliser une passerelle API	7
3	Maintenir un code clair et un versionnement	6
4	Utiliser une base de données par microservice pour maintenir l'isolation des données	6
5	Utiliser la conteneurisation pour les microservices pour un déploiement léger, cohérent et portable	6
6	Utiliser l'intégration continue (CI)	4
7	Utiliser des protocoles de communication efficaces et des motifs de communication	3
8	Utiliser la gestion des requêtes (par exemple, optimiser la charge utile, équilibrage de charge)	3
9	Utiliser des bases de données spécialisées comme NoSQL ou des bases de données en mémoire	3
10	Mettre en œuvre des vérifications de l'état de santé et utiliser des outils de surveillance	3
11	Surveiller les performances des microservices en temps réel pour identifier et résoudre les goulets d'étranglement potentiels	3
12	Utiliser le regroupement et la minification	2

N.de.refs : Nombre de Références

3.7.3 À retenir

Les pratiques de génie logiciel susceptibles d'améliorer les performances et l'utilisation des ressources des systèmes IoT basés sur des microservices en périphérie comprennent :

1. **Une passerelle API pour l'exposition à l'API REST** : Cette pratique permet de gérer efficacement les requêtes entrantes, de réduire la charge sur les microservices individuels et

de mettre en œuvre des fonctionnalités transversales, améliorant ainsi les performances et la sécurité globales du système.

2. **RPC pour une communication efficace des microservices** : Le protocole RPC permet une communication légère et rapide entre les microservices, réduisant la surcharge du réseau et améliorant les temps de réponse, ce qui est particulièrement bénéfique dans les environnements de périphérie où la bande passante et les ressources de calcul peuvent être limitées.
3. **Une approche de base de données par service avec des bases de données NoSQL ou en mémoire spécialisées** : Cette pratique permet à chaque microservice de gérer ses propres données de manière autonome, évitant les goulots d'étranglement centralisés et permettant une mise à l'échelle indépendante. L'utilisation de bases de données optimisées pour des cas d'utilisation spécifiques peut améliorer considérablement les performances en termes de latence et de débit.
4. **Une stratégie de version mono-référentiel pour la maintenabilité** : L'adoption d'une stratégie de version mono-référentiel favorise la maintenabilité et la collaboration en simplifiant la gestion des dépendances, les déploiements et la coordination entre les équipes de développement, ce qui se traduit par des cycles de développement plus rapides, moins de bogues et une allocation plus efficace des ressources.
5. **Des techniques de conteneurisation, de minification et de regroupement pour créer des composants système légers et portables** : La conteneurisation facilite le déploiement, la mise à l'échelle et la gestion des microservices dans des environnements hétérogènes. Les techniques de minification et de regroupement réduisent la taille des artefacts de microservices, entraînant des temps de chargement plus rapides et une consommation de bande passante réduite.
6. **la minification et le bundling** : l'utilisation de ce pratique permet d'optimiser l'utilisation de la bande passante, de la mémoire et des temps de chargement, ce qui est particulièrement important dans les environnements Edge où les ressources sont limitées.

Bien que l'intégration continue et le déploiement continu (CI/CD) soient des pratiques largement recommandées, leur application dans les environnements Edge présente des défis uniques, contrairement au cloud. En effet, les ressources limitées des devices Edge (CPU, mémoire, stockage) rendent plus difficile l'exécution des tâches lourdes des pipelines CI/CD. De plus, la grande diversité des devices et des configurations à la périphérie (OS, hardware, etc.) est un frein à l'automatisation, là où le cloud est plus standardisé. Cependant, on considère la pratique de la "minification et du bundling". La minification consiste à réduire la taille des fichiers de code en supprimant les caractères inutiles, tandis que le bundling regroupe plusieurs fichiers en un seul. Ces techniques permettent d'optimiser l'utilisation des ressources dans les environnements Edge.

3.8 Conclusion

Nous avons identifié six pratiques de génie logiciel qui peuvent améliorer les performances et l'utilisation des ressources des systèmes IoT basés sur des microservices en périphérie. Ces pratiques de génie logiciel ont été recommandées, mais n'ont pas été testées empiriquement. La prochaine phase de notre étude consiste à mener une étude empirique pour évaluer l'impact des performances et de l'utilisation des ressources de ces pratiques.

CHAPITRE 4

ETUDE EMPIRIQUE

Ce chapitre se concentre sur la description des éléments critiques et des étapes suivies lors de l'étude empirique. Il offre une explication approfondie de l'approche systématique soigneusement élaborée pour collecter des données, le processus de développement et de déploiement, l'examen des résultats et la formulation de conclusions finales dans le cadre de cette recherche. Il présente un cadre complet incluant les outils essentiels, les méthodologies et les métriques de performance utilisés de manière stratégique pour évaluer les différents aspects alignés sur les objectifs de l'étude.

Commençant par une exposition détaillée du choix des pratiques et de la méthodologie de recherche choisie, cette section aborde directement la ***QR3*** - *Quel est l'impact sur les performances des bonnes pratiques d'ingénierie logicielle identifiées pour les systèmes IoT basés sur les microservices déployés en périphérie ?*. À travers une étude empirique, l'approche adoptée implique la mise en œuvre de pratiques sélectionnées en génie logiciel identifiées dans le Chapitre 3, spécifiquement adaptées à notre étude de cas spécifique.

L'étude empirique comprend une expérience comparative mettant en contraste deux versions de notre étude de cas : l'une intégrant les pratiques de génie logiciel identifiées et l'autre dépourvue de ces pratiques. Les sections suivantes de ce chapitre offrent un aperçu complet de notre conception expérimentale, détaillent l'étude de cas, expliquent les métriques utilisées pour l'analyse comparative, décrivent les différents scénarios de test exécutés, et présentent une analyse approfondie des résultats obtenus.

Pour le reste de ce chapitre, nous utiliserons "système IoT basé sur les microservices" pour faire référence à la version WIMP basée sur les pratiques de génie logiciel sélectionnées, tandis que "système hérité" fait référence à la version WIMP sans les pratiques de génie logiciel sélectionnées.

4.1 Choix des pratiques logicielles

Le processus de sélection des pratiques logicielles est intimement lié à l'analyse approfondie de la GLR, conduite et détaillée dans le Chapitre 3. Cette étude exhaustive de la GLR a servi de fondement crucial pour identifier et sélectionner les pratiques logicielles les plus pertinentes et efficaces. Elle a constitué une base solide permettant de comprendre les tendances, les défis et les solutions actuelles présentes dans le domaine, facilitant ainsi une sélection éclairée des pratiques à adopter. En tenant compte des connaissances et des informations extraites de cette analyse approfondie, les pratiques logicielles choisies ont été spécifiquement élaborées pour répondre aux besoins et défis identifiés dans la SLR, afin d'optimiser les performances du système étudié.

4.1.1 Passerelle API

La passerelle API constitue un élément central de l'architecture des microservices, agissant comme le principal point d'accès au système. Sa principale fonction est de diriger les requêtes entrantes vers les microservices appropriés, permettant ainsi l'invocation de multiples services et l'agrégation de leurs résultats, comme l'illustre la Figure 4.1.

En essence, la passerelle API agit comme un proxy inversé vers les microservices et représente le point d'entrée unique dans le système. Elle s'apparente au motif de conception *Façade* en programmation orientée objet et rappelle la notion de "Couche Anti-Corruption" dans le Domain Driven Design. Son rôle consiste à simplifier considérablement les processus de conception, d'implémentation et de gestion des API, rendant leur coordination plus cohérente.

La passerelle API agit comme un élément central pour résoudre diverses problématiques majeures dans les systèmes, regroupant la gestion unifiée de fonctions telles que la sécurité, le throttling, le caching et la surveillance. Son rôle inclut la limitation des échanges excessifs entre les clients et les microservices, l'adaptation aux besoins variés des utilisateurs, le cheminement des requêtes vers les microservices en arrière-plan, la localisation et la gestion des instances fonctionnelles des microservices, ainsi que la détection des pannes dans ces instances.

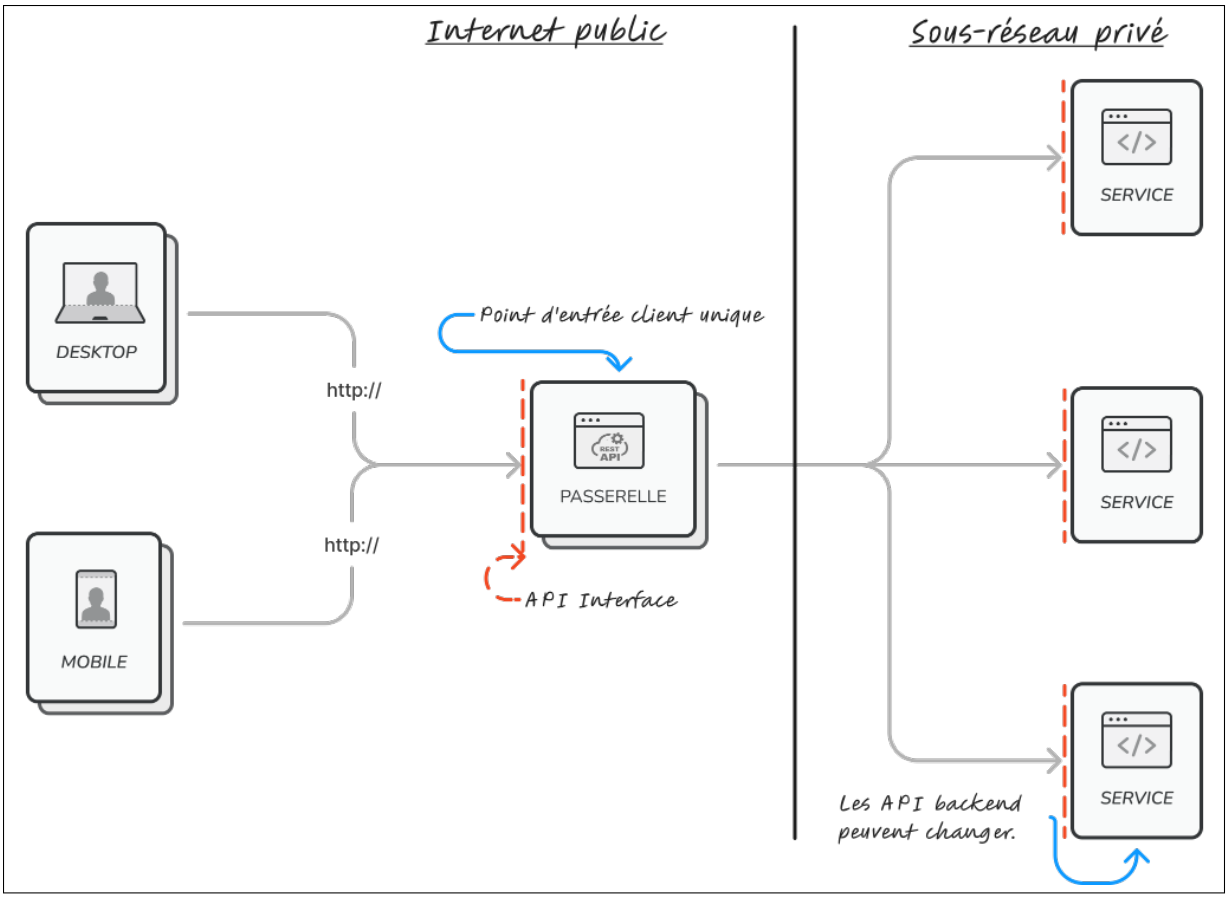


Figure 4.1 Architecture de passerelle API

En agissant comme un pont entre les microservices et leurs consommateurs, cette passerelle API orchestre le flux de données en fournissant une API adaptée aux clients. Elle facilite le routage des requêtes, la transformation des protocoles, et met en œuvre des fonctionnalités partagées comme l'authentification et la limitation du débit. Cette passerelle assume également diverses responsabilités telles que l'authentification, la surveillance et le traitement des réponses statiques. Dans certains scénarios, elle peut même agir comme un équilibreur de charge en distribuant le trafic entrant parmi les services disponibles, utilisant sa connaissance des emplacements et des adresses des services.

4.1.2 Base de données par service

La conception des bases de données évolue rapidement, ce qui pose des défis pour le développement des solutions basées sur les microservices. Dans cette architecture, chaque service est conçu pour être petit, spécialisé et autonome. Il s'occupe d'une fonctionnalité précise.

Chaque microservice doit gérer ses propres données de façon indépendante. Le modèle "Base de données par service" permet cela en donnant à chaque microservice sa propre base de données séparée.

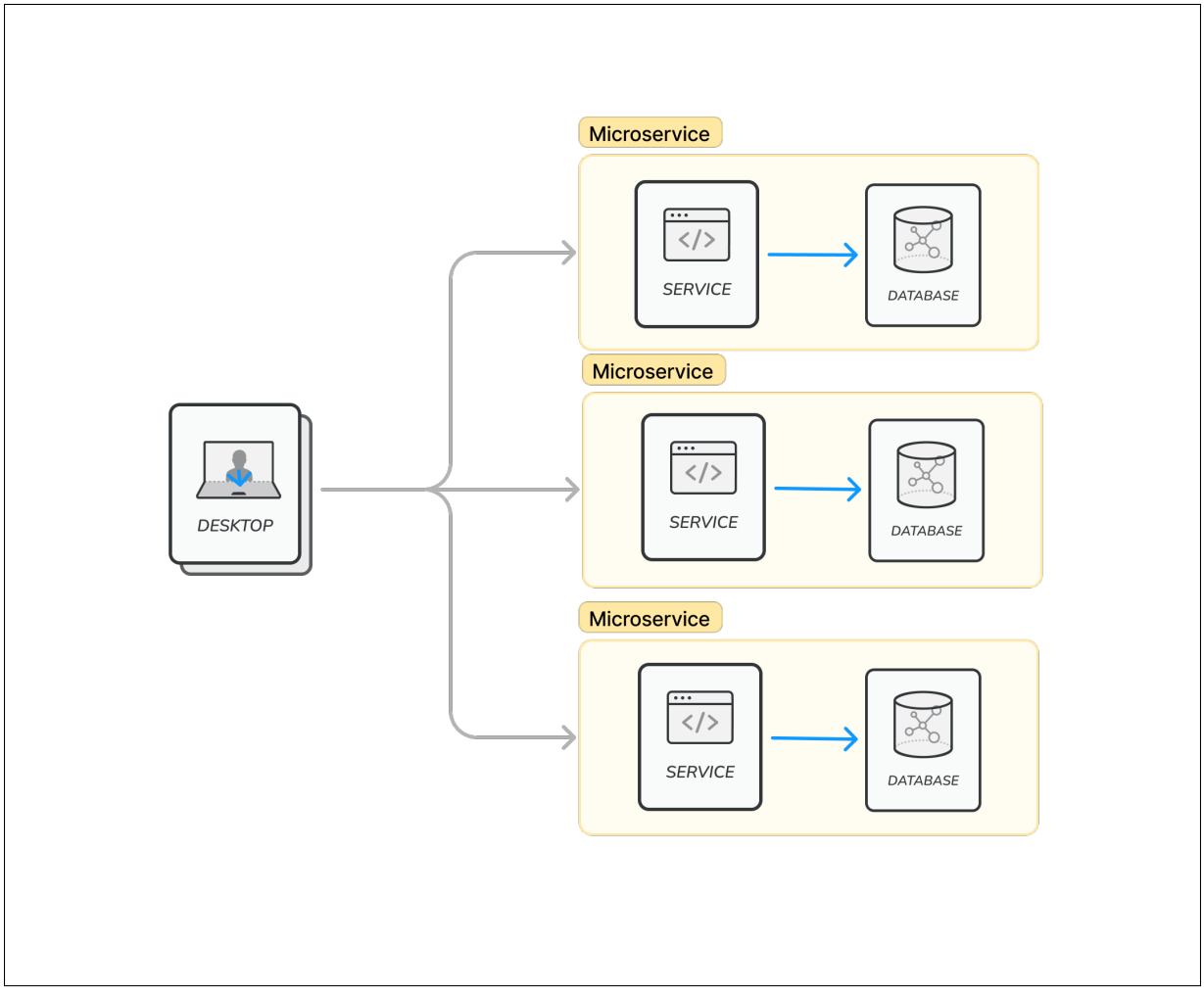


Figure 4.2 Architecture base de données par service

L'approche "base de données par service" dans l'architecture des microservices est définie et implique la dédication d'un stockage de données spécifique pour chaque microservice, que ce soit par un schéma complet ou une table. Cette configuration restreint l'accès des autres services aux bases de données qu'ils ne supervisent pas, apportant divers avantages. Cette individualisation du stockage de données facilite l'évolutivité et encapsule les données dans le domaine spécifique d'un microservice, simplifiant ainsi la compréhension globale du service et de ses données associées. Cette clarté s'avère particulièrement précieuse pour les nouveaux membres d'équipe, réduisant ainsi le temps et les efforts nécessaires pour assimiler leurs domaines de responsabilité. Cependant, elle nécessite un mécanisme de secours pour gérer les défaillances de communication au sein de cette configuration de service de base de données.

4.1.3 Conteneurisation

La conteneurisation est une technique informatique qui consiste à encapsuler une application et ses dépendances dans un conteneur logiciel autonome et isolé (Morabito, Kjällman & Komu, 2015; Pahl, 2015). Ces conteneurs, popularisés par des technologies comme Docker, permettent de déployer et d'exécuter des applications de manière cohérente et fiable, indépendamment de l'environnement de déploiement (Felter, Ferreira, Rajamony & Rubio, 2015b; Merkel, 2014b). Ils incluent tout ce dont une application a besoin pour s'exécuter, tels que le code, les bibliothèques, les outils système et les paramètres d'environnement (Merkel, 2014b; Pahl, Brogi, Soldani & Jamshidi, 2017). Cette approche offre une portabilité accrue, une efficacité opérationnelle et une facilité de gestion, car elle simplifie le déploiement, la mise à l'échelle et la gestion des applications (Mouat, 2016b; Burns, Grant, Oppenheimer, Brewer & Wilkes, 2016b). Les conteneurs isolent les applications les unes des autres, assurant une sécurité renforcée et une plus grande efficacité en termes d'utilisation des ressources informatiques (Burns *et al.*, 2016b; Pahl *et al.*, 2017). De plus, la conteneurisation facilite l'adoption de pratiques de développement modernes telles que l'intégration continue et le déploiement continu (CI/CD), ce qui accélère le cycle de développement et de déploiement des applications (Cito *et al.*, 2017; Lwakatare *et al.*, 2019).

4.1.4 Protocoles de communication efficaces

Dans une architecture de microservices, optimiser les modèles de communication est crucial, notamment en ce qui concerne l'impact du volume des requêtes sur les performances. En tant que domaine de recherche, l'amélioration de la communication entre microservices pour renforcer les performances globales du système est un axe majeur. Une approche notable dans l'évolution de ces modèles de communication est l'adoption du protocole RPC (Remote Procedure Call), un mécanisme permettant à un programme d'exécuter une procédure (un sous-programme) sur un autre ordinateur sans avoir à se préoccuper des communications réseau entre ces deux. Dans ce contexte, gRPC se présente comme un RPC framework particulièrement efficace.

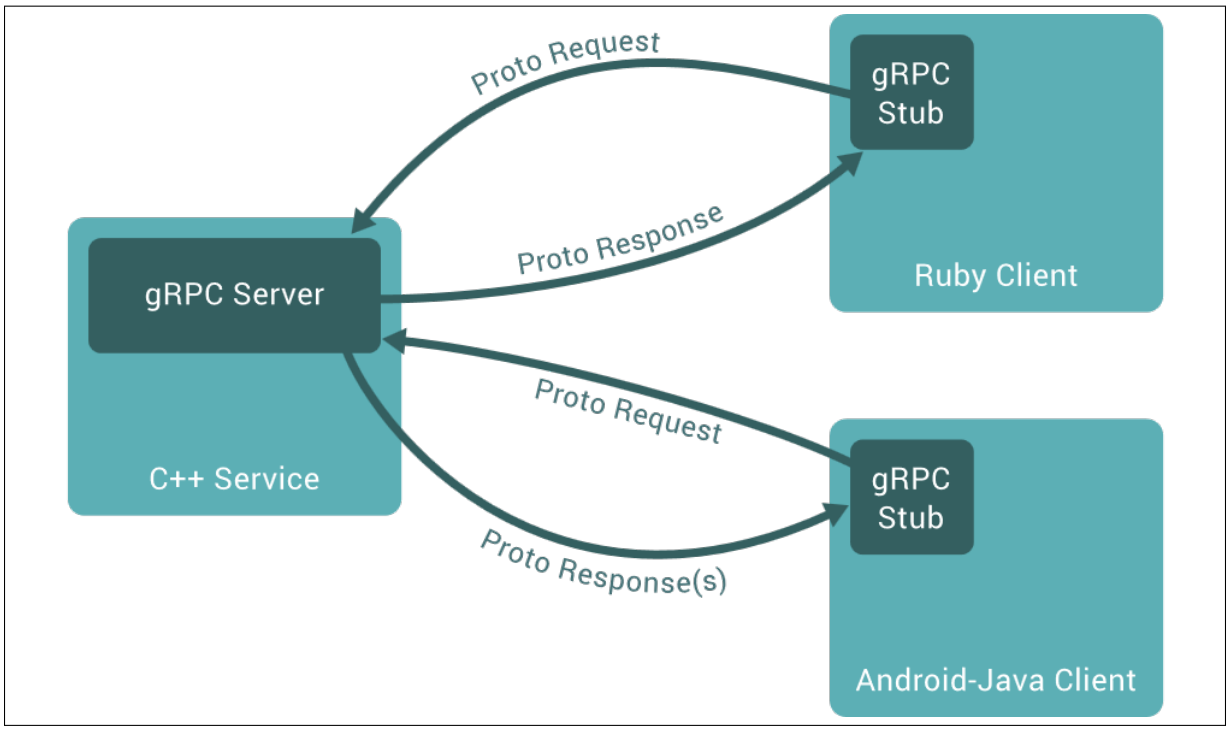


Figure 4.3 Architecture gRPC tirée de grpc.io (gRPC Authors, 2023)

gRPC est un cadre d'appel de procédure à distance (RPC) open source développé par Google. Google a ajouté la prise en charge d'un mécanisme RPC complet via les protocoles buffers (protobuf). gRPC peut être exécuté n'importe où et permet aux applications client et serveur de communiquer de manière transparente, facilitant ainsi la création de systèmes connectés.

En bref, avec gRPC, une application cliente peut appeler directement une méthode sur une application serveur située sur une autre machine, comme s'il s'agissait d'un objet local, ce qui facilite la création d'applications et de services distribués.

gRPC utilise les protocoles buffers comme langage de définition d'interface (IDL) et format d'échange de messages. Il a été introduit comme une alternative aux styles de construction d'API traditionnels tels que SOAP et REST. gRPC est open source, multiplateforme et construit sur HTTP/2. Il prend en charge les appels en continu et fournit des cadres de génération de code. gRPC offre une sérialisation efficace, une vérification des types et des validations, ce qui en fait un choix moderne et performant pour le développement de systèmes distribués et de microservices.

4.1.5 Les techniques de minification et bundling

Le bundling et la minification sont deux techniques couramment utilisées dans le développement web pour optimiser les performances des applications et réduire le temps de chargement des pages.

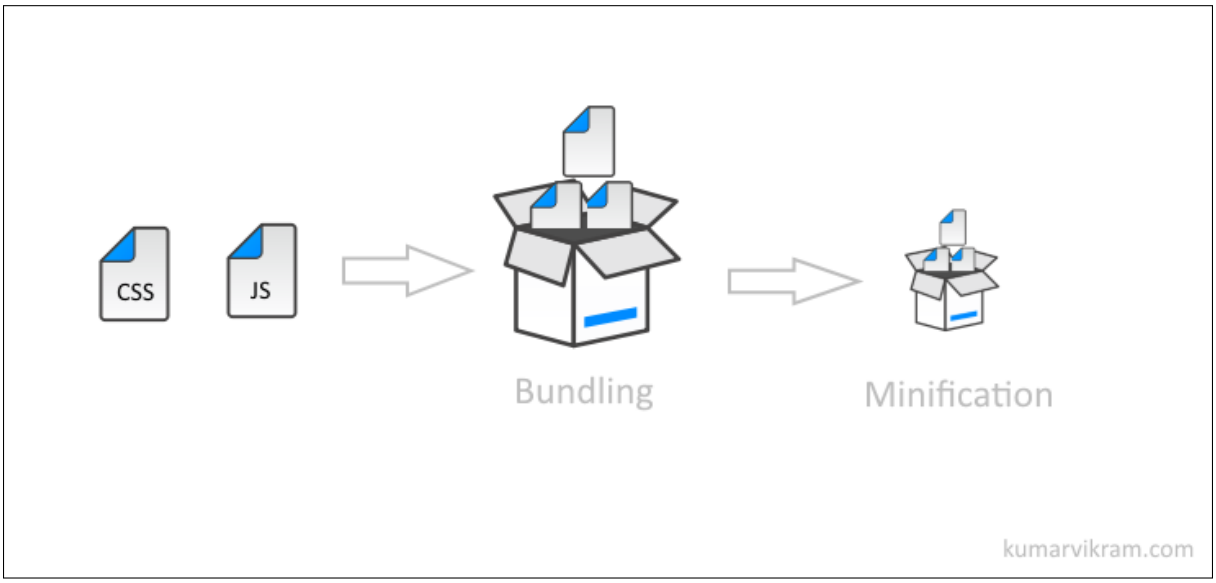


Figure 4.4 Les techniques de minification et regroupement tirée de Vikram Kumar blog (Kumar, 2024)

4.1.5.1 Le regroupement (bundling)

Dans le domaine du génie logiciel, un *regroupement ou bundle* désigne un ensemble de codes, de fichiers ou de ressources regroupés en vue d'une distribution ou d'un déploiement efficace. Les *regroupements* sont utilisés pour organiser et distribuer des composants logiciels, des bibliothèques ou des ressources de manière cohérente. La nature spécifique d'une offre groupée peut varier en fonction de la technologie ou du cadre utilisé. Voici quelques types d'offres groupées courantes :

1. **Ensembles JavaScript** : Dans le développement web, les fichiers JavaScript sont souvent regroupés en un seul fichier. Ce fichier regroupé est minifié et parfois transpilé afin de réduire sa taille et d'améliorer les performances. Le regroupement minimise le nombre de requêtes HTTP nécessaires au chargement d'une page web, ce qui se traduit par des temps de chargement plus courts ;
2. **Ensembles de paquets** : Les langages de programmation disposent souvent de gestionnaires de paquets (par exemple, pip pour Python) qui permettent aux développeurs de regrouper des modules ou des bibliothèques. Ces paquets peuvent être facilement installés et utilisés dans des projets. Ils comprennent généralement du code, des ressources et des métadonnées relatives au paquet ;
3. **Ensembles d'actifs** : Dans le développement de jeux et de sites web, les ensembles de ressources sont utilisés pour regrouper des ressources multimédias telles que des images, des sons et des textures. En regroupant ces ressources, elles peuvent être chargées efficacement dans le jeu ou l'application web, ce qui réduit les temps de chargement et optimise la gestion des ressources ;
4. **Ensembles de dépendances** : Les *bundles* peuvent également faire référence à des collections de dépendances ou de bibliothèques tierces qui sont regroupées pour un projet ou une application spécifique. Le regroupement des dépendances simplifie leur gestion et leur version au sein du projet.

Le regroupement est effectué pour rationaliser la distribution, améliorer les performances et gérer efficacement les dépendances. Les regroupements sont généralement créés au cours du processus de construction d'un projet logiciel et peuvent être optimisés pour une utilisation en production. Le code ou les actifs regroupés sont ensuite déployés vers les utilisateurs finaux ou les serveurs pour être consommés.

4.1.5.2 La minification

Les techniques de minification sont des méthodes de compression utilisées pour réduire la taille des fichiers sources en éliminant les espaces, les commentaires, les sauts de ligne et en raccourcissant les noms de variables :

1. **Suppression des espaces blancs et des sauts de ligne :** élimination des espaces inutiles et des caractères de retour à la ligne pour réduire l'encombrement des fichiers ;
2. **Suppression des commentaires :** élimination des commentaires de code qui ne sont pas nécessaires pour l'exécution du programme final ;
3. **Optimisation des noms de variables :** réduction de la longueur des noms de variables tout en préservant leur fonctionnalité, souvent par le biais de techniques de renommage ou d'abréviation ;
4. **Regroupement de fichiers et de ressources :** fusion de plusieurs fichiers en un seul ou regroupement de ressources similaires pour réduire les appels réseau et améliorer les performances de chargement ;
5. **Réduction de la taille des données :** utilisation de méthodes de compression pour réduire la taille des données sans compromettre leur intégrité ;

4.2 Conception de l'expérience

Pour fournir une explication plus détaillée de notre expérience de performance examinant l'impact des pratiques de génie logicielles choisit, nous avons adopté la conception d'expérience "Un Facteur, Deux Niveaux" (Montgomery, 2009). Nous avons mené cette expérience pour étudier l'effet d'une seule variable ("l'utilisation des pratiques") sur deux niveaux ou paramètres

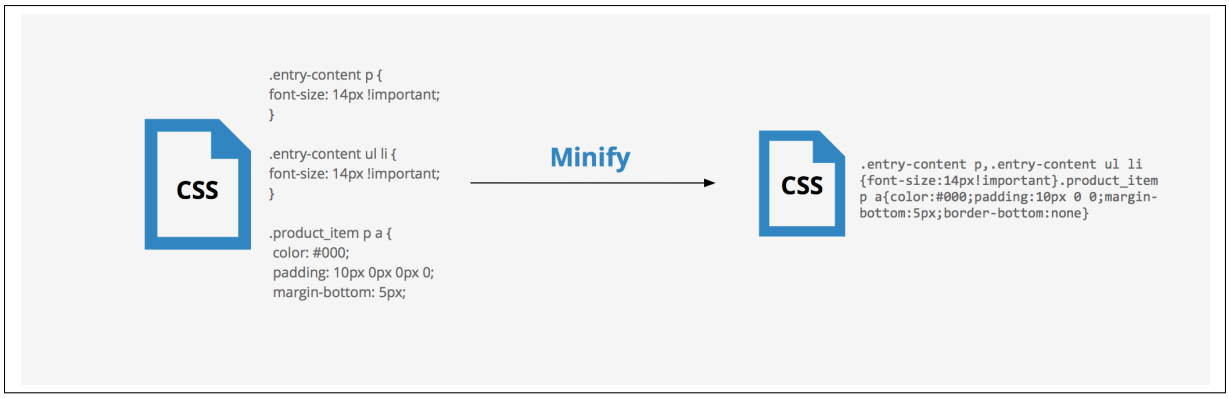


Figure 4.5 Technique de minification

distincts (“les deux systèmes IoT”). La Figure 4.6 illustre le processus d’évaluation et de comparaison des performances et de l’utilisation des ressources de deux versions de WIMP : l’une sans ces pratiques et l’autre avec ces pratiques.

Pour mener à bien notre expérience, nous avons décidé de faire évoluer l’architecture du système WIMP d’une architecture monolithique vers une architecture basée sur des microservices. Cette décision a été motivée par notre volonté d’appliquer des pratiques de génie logiciel sélectionnées.

Dans un premier temps, nous avons identifié les principales fonctionnalités du système WIMP existant et les avons isolées en différents microservices autonomes. Ensuite, nous avons appliqué les pratiques de génie logiciel sélectionnées à l’architecture basée sur les microservices. L’une de ces pratiques est la mise en place d’une passerelle API, qui sert de point d’entrée unique pour les requêtes des clients et effectue le routage vers les microservices appropriés. La passerelle API offre également des fonctionnalités supplémentaires telles que l’authentification, la limitation du débit et la gestion des erreurs.

À l’issue de ce processus, nous disposons de deux versions du système WIMP :

1. La version existante, basée sur une architecture monolithique.
2. La nouvelle version, basée sur une architecture de microservices avec les pratiques de génie logiciel appliquées.

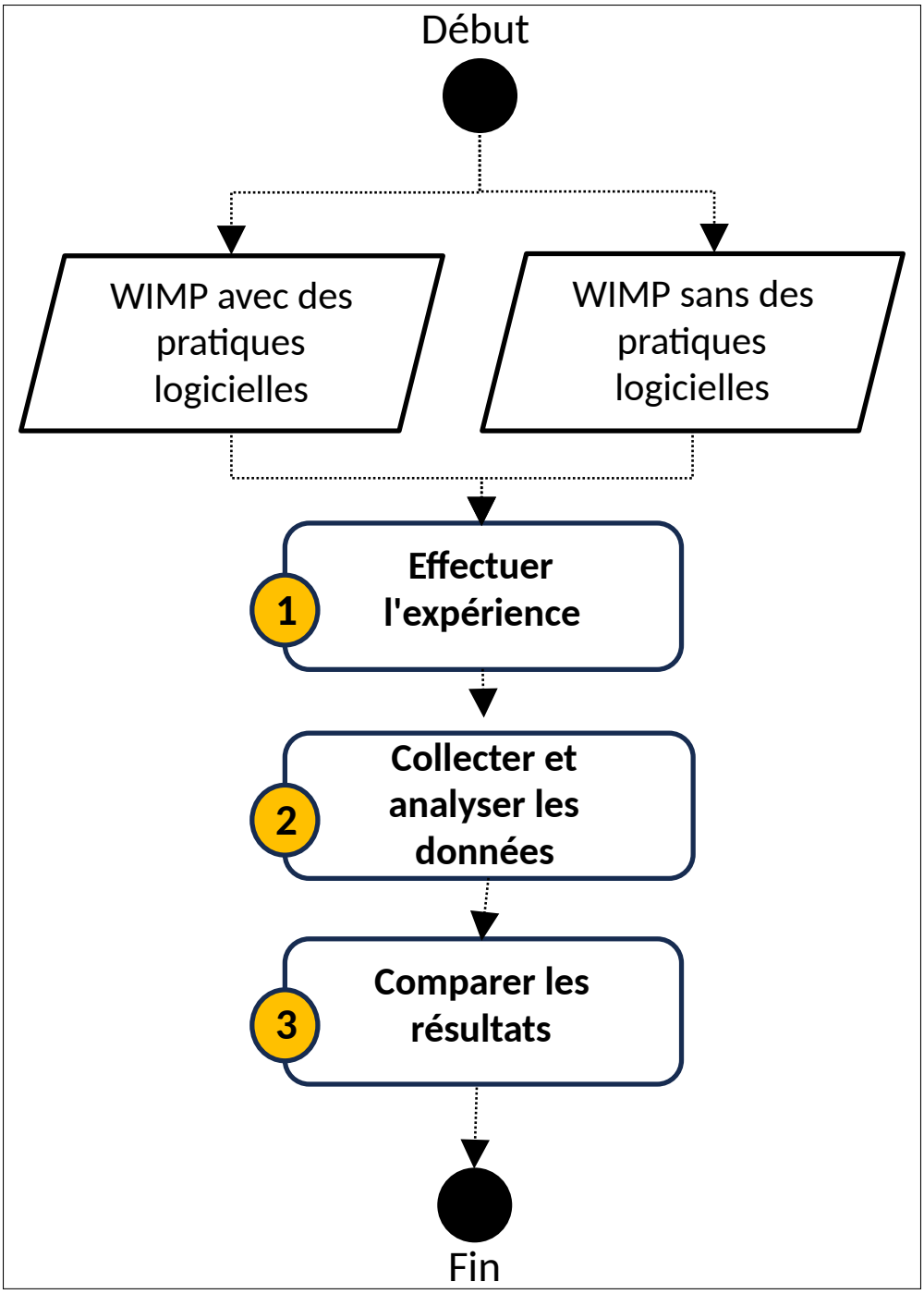


Figure 4.6 Conception de l'expérience

Cette approche nous permet de comparer les performances des deux versions du système dans le cadre de notre expérience. Nous pourrons ainsi évaluer les avantages et les défis liés à l'adoption

d'une architecture basée sur les microservices et à l'application de pratiques de génie logiciel sélectionnées.

Les résultats de cette expérience contribueront à enrichir notre compréhension des architectures logicielles modernes et des bonnes pratiques de développement, tout en offrant des perspectives d'amélioration pour le système WIMP. Nous avons modifié le code source des deux versions pour intégrer le client *Prometheus* afin d'exposer et de collecter diverses métriques. L'expérience se compose des 3 étapes suivantes :

❶ **Effectuer l'expérience**. Nous avons utilisé un outil de test de charge pour évaluer le comportement des deux versions de WIMP dans différentes conditions de charge et observer les performances et l'utilisation des ressources de chacune. Nous avons utilisé un outil de test de charge open-source appelé *Artillery* [1]. Cet outil est utilisé pour envoyer des requêtes concurrentes et évaluer les performances du système.

❷ **Collecte et analyse des données**. Nous avons collecté les données pour différentes métriques des deux versions de WIMP de deux manières. Nous avons collecté les métriques générées par *Artillery*. Cet outil génère des rapports détaillés qui incluent la latence, le débit et le nombre de requêtes.Nous avons utilisé *Prometheus* [2] et *Grafana* [3] pour collecter et visualiser l'utilisation du CPU et de la mémoire.

❸ **Comparer les résultats**. Nous avons réalisé une analyse comparative des données provenant de ces deux systèmes. Cette analyse nous a permis d'évaluer les performances de chaque système.

Dans le cadre de notre expérience avec le système WIMP , nous avons obtenu l'autorisation des professeurs participants pour collecter et utiliser leurs données de localisation et de disponibilité. Ces données seront strictement limitées à l'usage prévu dans le contexte de notre expérience et ne seront en aucun cas utilisées à d'autres fins sans le consentement explicite des professeurs concernés.

[1] https://www.artillery.io/docs

[2] https://prometheus.io/

[3] https://grafana.com/

4.3 Description de cas d'étude

WIMP [4] ou "Where Is My Professor" est un système basé sur l'IoT qui permet aux étudiants de suivre la disponibilité de leurs professeurs en temps réel. L'objectif principal de ce système est de collecter des données à partir de divers appareils IoT, tels que des capteurs, des caméras et des balises, pour savoir où se trouvent les professeurs à l'intérieur du bâtiment universitaire et d'autres informations pertinentes. Le système peut offrir une réponse automatisée sur la disponibilité du professeur en analysant les données collectées.

Le scénario envisagé vise à améliorer la planification et la communication entre les étudiants et les professeurs. Avec ce système, les étudiants peuvent utiliser des capteurs pour déterminer si le professeur est dans son bureau, engagé dans des conversations avec d'autres personnes, et réceptif à l'interaction avec les étudiants. Ce système dispose de capteurs intelligents, tels que la prise intelligente WEMO [5], ainsi que des appareils comme la montre Fitbit [6], qui servent à suivre la localisation et jouent un rôle essentiel dans le processus décisionnel. Les principales fonctionnalités seront résumées comme suit :

1. Gestion des utilisateurs internes : cela inclut les comptes utilisateur, les profils, les autorisations, etc.
2. Réponses automatisées sur la disponibilité du professeur générées grâce à l'analyse des données collectées à partir de divers appareils IoT.
3. Suivi en temps réel de la disponibilité et de la localisation du professeur en utilisant les données collectées à partir d'appareils IoT.

Le but ultime du développement de WIMP est de fournir un système IoT fonctionnel aux chercheurs pour mener des expériences de test.

---

[4] https://ptidejteam.github.io/wimp-wiki

[5] https://www.belkin.com/products/wemo-smart-home/

[6] https://www.fitbit.com/global/en-ca/home

4.3.1 Système IoT hérité

Au niveau architectural, le projet WIMP comprend deux parties principales : la partie Étudiant et la partie Enseignant/Administration.

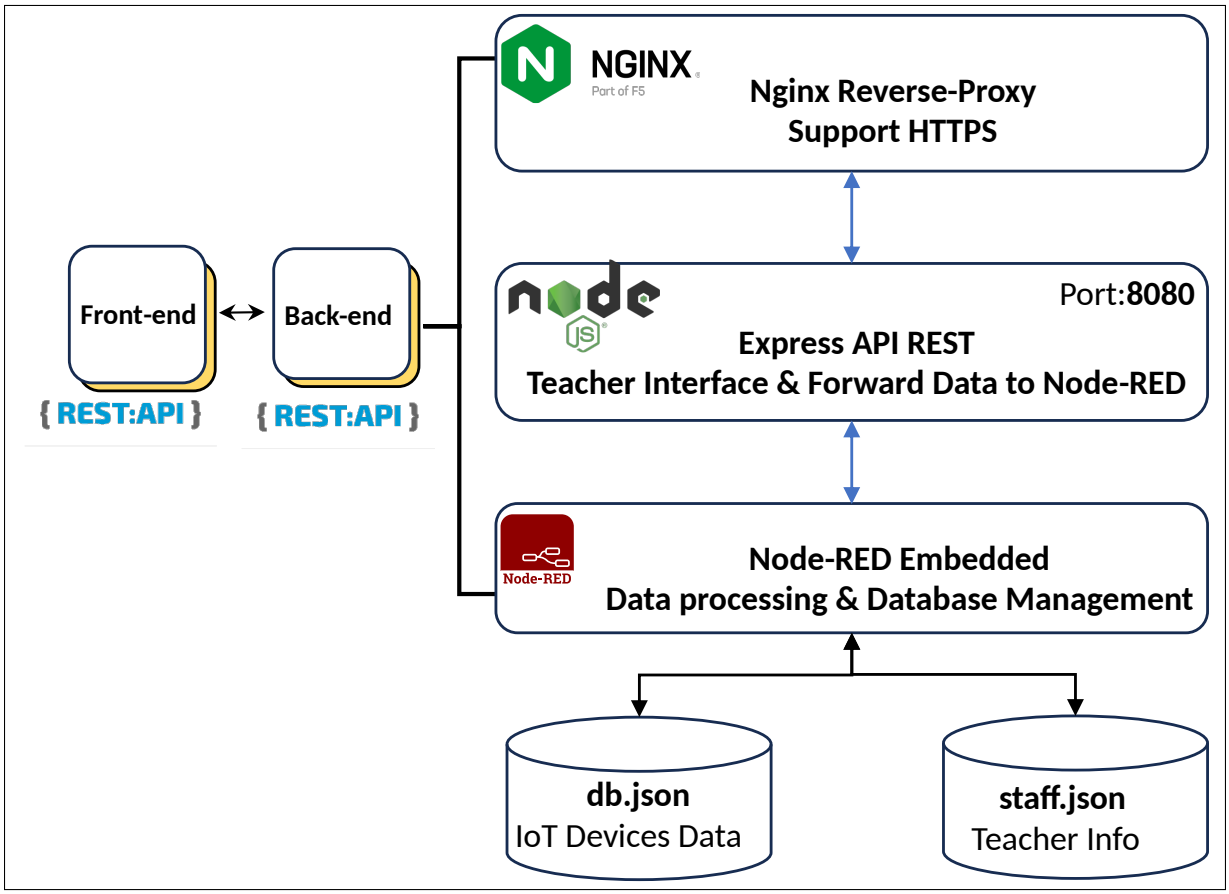


Figure 4.7 Architecture du système IoT hérité

La partie “Étudiant” fournit un serveur web accessible aux étudiants, offrant une interface graphique résumant l’état de disponibilité de leur enseignant. Dans ce wiki, cette partie est appelée le Frontend du système WIMP, permettant aux étudiants et aux chercheurs de visualiser l’état de leurs enseignants via une interface web. Techniquement, le frontend du système WIMP est un serveur web Express.js avec une gestion de base de données simple-json-db et une authentification via passport.js. Ce serveur communique avec le backend du système pour obtenir des informations sur l’état des enseignants. Les étudiants doivent se connecter pour

accéder au site depuis le réseau global, leur niveau d'études déterminant les informations qu'ils voient. Par exemple, un chercheur pourrait avoir accès à des informations plus spécifiques sur un professeur qu'un étudiant en licence.

La partie "Enseignant/Administration" représente le cœur du projet WIMP, appelé Backend du système WIMP dans ce wiki. Il permet aux enseignants de définir et de gérer leur état et de construire leur propre logique dans un flux Node-RED pour calculer leur état actuel. Le backend du système fonctionne sur un serveur Node-RED, qui traite les données provenant de différents appareils. Ce serveur est intégré dans un serveur Express.js et remplit trois fonctions principales. Premièrement, il agit comme pare-feu pour l'accès à Node-RED. Deuxièmement, il expose une API REST accessible via le frontend du système. Enfin, il héberge les pages web disponibles pour les enseignants. Le Backend peut être divisé en deux sous-parties : le Front-backend, qui gère l'interface Enseignant à l'aide de pages web, et le Back-backend, qui contient la partie Node-Red et l'API accessible via le Backend ou le Frontend du système. Le serveur Express.js connecte ces deux sous-parties.

L'application a été développée en utilisant un processus traditionnel appelé modèle "cascade", qui est une approche séquentielle du développement logiciel. Dans ce modèle, chaque phase de développement doit être terminée avant que la phase suivante puisse commencer. L'équipe de développement a d'abord créé le frontend de l'application, suivi du backend. Après avoir terminé ces phases, ils ont intégré les deux parties, effectué les tests nécessaires et configuré les appareils avant de déployer l'application sur eux. Les références des dépôts du système se trouvent respectivement dans le dépôt backend [7] et le dépôt frontend [8].

4.3.2 Système IoT à base de microservices

Nous avons effectué une restructuration architecturale du système existant et introduit des composants de microservices. Cette transformation a abouti à un système IoT à base de

[7] https://github.com/ptidejteam/wimp-backend

[8] https://github.com/ptidejteam/wimp-frontend

microservices exploitant la pratique architecturale de la passerelle API. Le système est décomposé en composants de logique métier utilisant gRPC et HTTP/2 comme principaux protocoles de communication interne, et nous rendons le système accessible à l'extérieur en utilisant HTTP/1.x. Nous présentons une illustration de cette architecture dans la Figure 4.7.

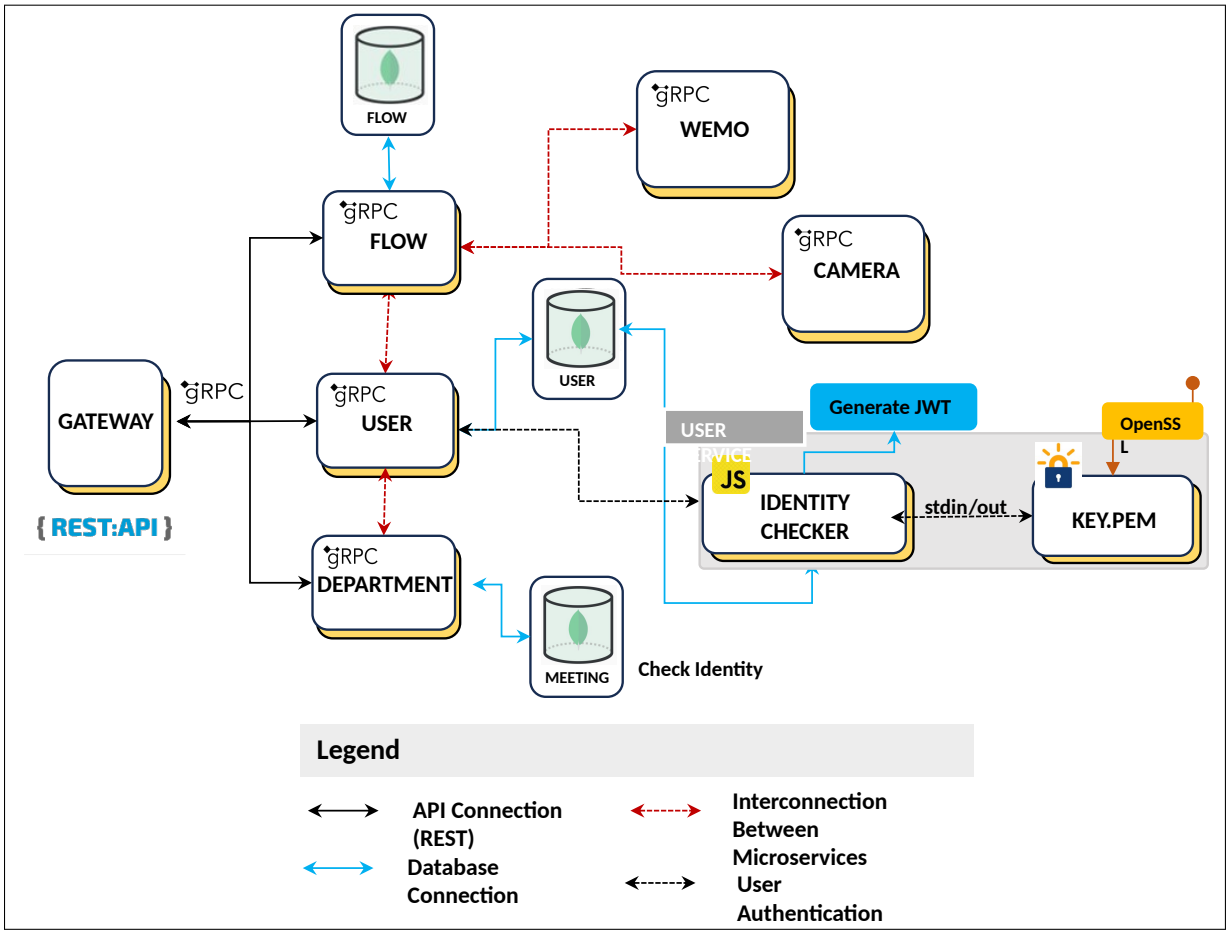


Figure 4.8 Architecture de la nouvelle version du système IoT

Le développement de cette version du système IoT a suivi les recommandations de codage émises par Google [9], tout en adoptant une structure de monoréférentiel gérée par Lerna [10]. Cela a été réalisé en mettant l'accent sur l'application des meilleures pratiques et des normes de codage efficaces afin d'améliorer la qualité et la maintenabilité du code. La base de code du

[9] https://google.github.io/styleguide/jsguide.html

[10] https://lerna.js.org

projet a été organisée selon une structure de monoréférentiel, administrée par Lerna, simplifiant ainsi la gestion du code et garantissant un processus de développement harmonieux. Dans notre architecture, un microservice, appelé la "passerelle", sert de point d'entrée unique dans notre système, gérant le routage des requêtes d'Interface de Programmation Applicative (API) vers tous les services et fonctionnant comme un serveur proxy pour les requêtes HTTP/1.x. De plus, il gère des fonctions telles que la vérification de l'autorisation. Ce microservice a été implémenté en utilisant *GoLang*. Go, plus communément appelé Golang, représente un langage de programmation open source développé par Google. Sa conception vise à fournir une efficacité élevée, une simplicité et une concision lors de la rédaction du code, en particulier pour la création de logiciels système, de programmes réseau et de vastes systèmes distribués.

Ce langage intègre le plugiciel *gRPC-Gateway* [11] qui interprète les définitions de service *protobuf* pour produire un serveur proxy inversé. Ce serveur traduit une *API HTTP RESTful* en *gRPC*. Il est généré en fonction des annotations *google.api.http* présentes dans les définitions de service. Cette fonctionnalité permet de proposer vos API dans les styles gRPC et REST simultanément.

Nous aurions pu implémenter la passerelle en JavaScript / Node.js, qui est un choix populaire pour ce type de composant. Cependant, bien que Node.js excelle pour les entrées/sorties non bloquantes, Go offre de meilleures performances brutes et une empreinte mémoire plus faible. Son modèle de concurrence natif avec les goroutines et les channels est également avantageux dans un contexte de fort trafic.

C'est donc la combinaison de bonnes performances, d'une excellente intégration avec gRPC via *gRPC-Gateway*, et d'un outillage robuste qui nous a fait choisir Go plutôt que JavaScript pour notre passerelle. Cela nous permet d'avoir un point d'entrée efficace et maintenable dans notre architecture de microservices.

Les autres composants importants du système IoT à base de microservices reposent sur *Node.js* comme environnement d'exécution côté serveur pour *JavaScript*, fondé sur le moteur V8 de Google. Il permet le développement d'applications web et temps réel côté serveur grâce à

[11] https://github.com/grpc-ecosystem/grpc-gateway

son architecture asynchrone et modulaire. Ce choix a été fait en raison des caractéristiques de JavaScript en tant que langage de programmation à haut niveau, léger, dynamique et non typé, le rendant compatible avec les paradigmes de programmation fonctionnelle et orienté objet (Jaimini & Dhaniwala, 2016; Oliveira & Mattos, 2019). Cela était également dû à sa large communauté de support. Ces composants suivent une pratique architecturale de base de données par service, où nous utilisons des bases de données spécialisées telles que *NoSQL* ou des bases de données en mémoire avec *MongoDB* comme plateforme de données.

En plus de Node.js, nous avons opté pour *Python* pour concevoir les composants spécifiques aux microservices. Ces composants fonctionnent comme une couche intermédiaire interagissant avec les dispositifs et capteurs IoT.

Le premier microservice, “User”, sert de système de gestion d’utilisateurs dédié, gérant efficacement les opérations de création, lecture, mise à jour et suppression (CRUD) tout en gérant l’authentification des utilisateurs et la vérification de l’identité. Le système classe les utilisateurs en différents niveaux de permission, et un logiciel médiateur personnalisé impose des permissions d’accès en appliquant la pratique de l’autorisation par “jeton”.

Le microservice “Flow” se concentre sur les évaluations de disponibilité en utilisant des données provenant de dispositifs et se connecte à des nanomicroservices spécifiques aux dispositifs via des flux Node-RED. Ce microservice s’appuie sur des nanomicroservices spécialisés plus petits, tels que “Camera”, qui utilisent YOLOv3 pour la détection de présence individuelle dans le bureau d’un professeur, et “Smart Plug” pour la détection des dispositifs *WEMO*.

Le microservice “Departement” se spécialise dans la fourniture d’informations complètes sur les utilisateurs, en tenant compte des affiliations des utilisateurs à des départements spécifiques, et prend en charge les opérations CRUD pour une gestion efficace des données utilisateur.

Enfin, dans le cadre de notre déploiement système dans un environnement “Docker”, nous utilisons des techniques de minification et de regroupement pour générer les microservices construits.

4.4 Scénarios de test

Nous définissons deux scénarios de test pour faciliter l'évaluation. Ces scénarios sont basés sur les principales fonctionnalités décrites dans la Section 4.3. Bien que ces scénarios fournissent un aperçu de la performance du système, il est difficile de déterminer à quel point ils sont vraiment représentatifs sans plus de contexte sur les nombres et les modèles d'utilisation réels attendus.

Dans nos tests des deux scénarios, nous avons envoyé des connexions concurrentes en faisant varier le nombre d'utilisateurs virtuels générés par l'outil de charge de test. Grâce à un processus itératif d'essais et d'erreurs avec l'outil, nous sommes arrivés à une plage spécifique de 1, 10, 50, 100 à 400 utilisateurs virtuels pour nos tests afin d'observer la réponse du système à des charges utilisateur différentes. Sur la base de rounds de pré-tests, nous avons déterminé que l'envoi de 10 000 requêtes depuis l'outil fournirait un point de comparaison approprié, comme rapporté par l'outil :

1. **Scénario 1** : Envoi de 10 000 requêtes pour l'authentification dans le système IoT ;
2. **Scénario 2** : Envoi de 10 000 requêtes pour obtenir la disponibilité du professeur.

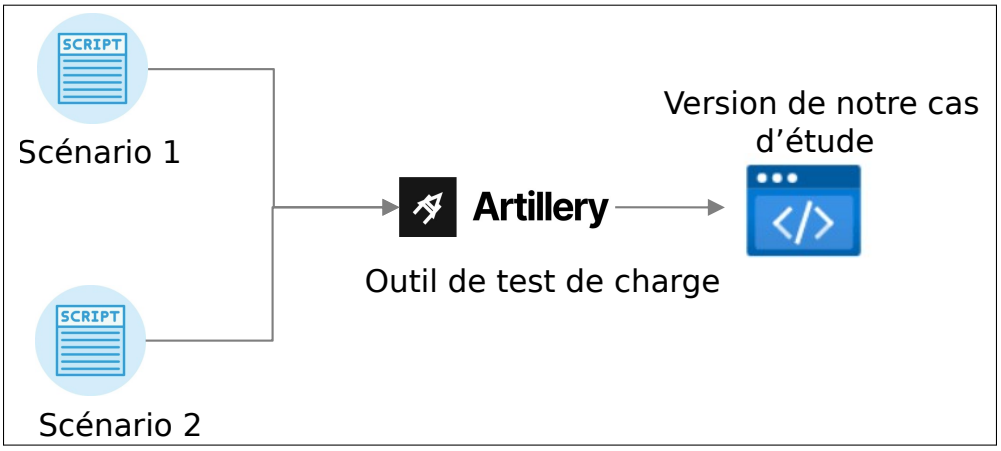


Figure 4.9 Scénarios de test

Le premier scénario simule le processus d'authentification de l'utilisateur dans le système. Ce scénario couvre la plupart des métriques définies, ce qui en fait un test crucial.

Le deuxième scénario simule la demande de l'utilisateur pour connaître le statut de disponibilité d'un professeur. Ce scénario aide non seulement à évaluer la performance du système, mais aussi à explorer les dynamiques complexes des appareils IoT et l'intercommunication entre les microservices, en particulier dans le cas du système IoT basé sur les microservices.

4.5 Métriques

Pour évaluer la performance de chaque système, nous identifions diverses métriques qui s'inspirent de Lira *et al.* (2023) et qui sont alignées sur les indicateurs clés de performance (ICP) :

- **Performance du processeur.** Pour évaluer les performances du processeur, nous analysons la métrique *Utilisation du CPU*, qui indique le pourcentage de capacité du CPU. L'utilisation du CPU est généralement exprimée en pourcentage et est calculée en utilisant la formule suivante :

$$\text{Utilisation du CPU (\%)} = \frac{\text{Temps CPU utilisé}}{\text{Temps total}} \times 100\% \tag{4.1}$$

  Le pourcentage d'utilisation du CPU est obtenu en divisant le temps pendant lequel le processeur est effectivement utilisé par le temps total, puis en multipliant le résultat par 100 pour obtenir le pourcentage. Cela permet d'évaluer la charge de travail que le CPU gère par rapport à sa capacité totale sur une période donnée. Par exemple, lors de l'examen du scénario d'authentification référencé comme *Scénario 1* dans la Section 4.4, l'utilisation du CPU combine les mesures à la fois de la passerelle et des microservices utilisateurs dans le contexte du système basé sur les microservices. Quant au scénario de disponibilité, la métrique inclut la combinaison du processus de la passerelle, du processus de flux et du processus *WEMO*. En revanche, dans le système hérité, les mesures proviennent d'un seul processus pour les deux scénarios ;
- **Performance de la mémoire.** Pour évaluer les performances de la mémoire, nous examinons l'utilisation de la mémoire des processus, qui prend en compte la quantité totale de mémoire physique. Le calcul de l'utilisation de la mémoire d'un processus peut varier en fonction de la mesure spécifique souhaitée. Pour une métrique courante telle que le pourcentage

d'utilisation de la mémoire par un processus, nous pouvons utiliser la formule suivante :

$$\text{Mémoire utilisée par processus (\%)} = \frac{\text{Mémoire utilisée par le processus}}{\text{Mémoire totale du système}} \times 100\% \quad (4.2)$$

Cette formule évalue le rapport entre la mémoire utilisée par un processus spécifique et la mémoire totale du système, le tout multiplié par 100 pour obtenir le pourcentage d'utilisation de la mémoire par ce processus ;

- **Performance du réseau.** Nous effectuons une évaluation du réseau en prenant en compte des métriques distinctes :
  - **Le débit**, qui mesure le nombre de requêtes ou de transactions traitées par seconde. En termes mathématiques, il peut être exprimé comme suit :

$$\text{Débit} = \frac{\text{Nombre de requêtes}}{\text{Temps écoulé}} \quad (4.3)$$

    Cette formule évalue le nombre de requêtes ou de transactions traitées par unité de temps, généralement par seconde (mais cela peut être par minute, par heure, etc., selon le contexte), représentant ainsi la vitesse ou le taux de traitement du système.
  - **Le temps de latence** représente le temps nécessaire pour que la requête de l'utilisateur atteigne le système et que la réponse du système revienne à l'utilisateur. Cela inclut le temps de transmission sur le réseau. Mathématiquement, la formule de la latence peut être représentée comme suit :

$$\text{Latence} = \text{Heure de réception} - \text{Heure d'envoi} \quad (4.4)$$

Cette évaluation est générée avec l'outil de test de charge "*Artillery*" pour les deux scénarios.

Lors de la comparaison des performances des différents systèmes sur la base de ces métriques, nous avons choisi d'utiliser la moyenne comme mesure agrégée. Bien que d'autres aspects puissent être pris en compte pour affiner l'analyse, tels que l'écart-type ou la médiane, nous nous sommes concentrés sur la moyenne afin d'obtenir une vue d'ensemble de la performance de chaque système de manière simple et directe. Cette approche permet une comparaison initiale

des systèmes, tout en reconnaissant que des analyses plus approfondies pourraient apporter des nuances supplémentaires.

Le tableau 4.2 offre une synthèse détaillée des mesures, des outils employés, ainsi que des requêtes ou métriques spécifiques associées à chaque mesure. Cette approche méthodique permet d'évaluer et de comparer de manière exhaustive les performances de chaque système. Dans notre banc d'essai, nous avons utilisé le *compilateur GoLang, Node.js et Python* pour le développement de nos microservices. *MongoDB* a servi de plateforme de source de données choisie, et nous avons encapsulé l'ensemble de l'environnement à travers *Docker Compose*.

4.6 Environnement de test

Nous avons déployé les deux versions de notre cas d'utilisation sur le même *Raspberry Pi 4 Model B* pour permettre une comparaison équitable. Le Raspberry Pi 4 Model B est un nano-ordinateur abordable et polyvalent, largement utilisé pour le développement et le prototypage de diverses applications. Ses caractéristiques techniques en font un choix intéressant pour le déploiement de notre cas d'utilisation.

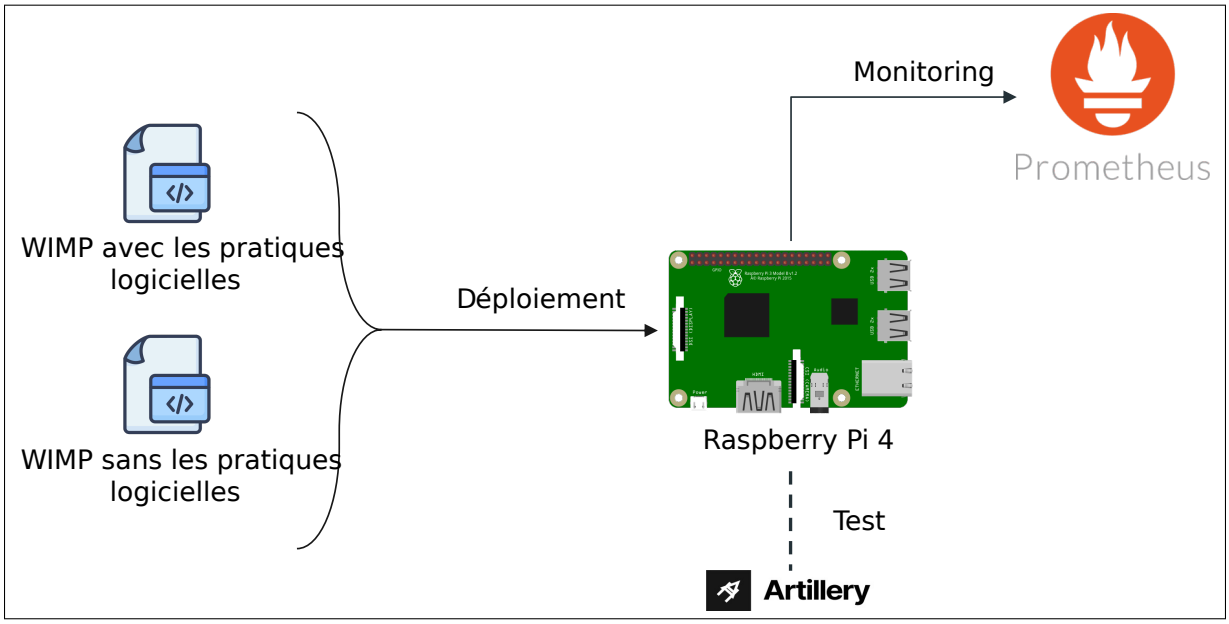


Figure 4.10 Architecture du système IoT à base de microservices

Tableau 4.1 Caractéristiques techniques du Raspberry Pi 4 Model B

Caractéristique	**Spécification**
Processeur	Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Mémoire vive	4GB LPDDR4-3200 SDRAM
Connectivité	2.4 GHz et 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet 2 ports USB 3.0 ; 2 ports USB 2.0
GPIO	GPIO standard 40 broches
Vidéo	2 × micro-HDMI (jusqu'à 4Kp60 pris en charge) Port DisplayPort 2 voies dans le connecteur USB-C
Alimentation	5 V DC via le connecteur USB-C (minimum 3 A) 5 V DC via les broches GPIO (minimum 3 A) Power over Ethernet (PoE) activé (nécessite un HAT PoE séparé)

Tableau 4.2 Tableau des métriques et des outils utilisés

Catégorie	**Outil**	**Métriques / Requêtes**
Performance du CPU	Prometheus	cpu_user_seconds_total
Performance de la mémoire	Prometheus	process_resident_memory_bytes
Performance du réseau	Artillery	latence, débit

En utilisant le même matériel pour les deux versions de notre cas d'utilisation, nous nous assurons que les différences de performance observées sont principalement dues aux choix d'implémentation et d'architecture, plutôt qu'à des variations de matériel. Le Raspberry Pi 4 Model B offre des performances suffisantes pour exécuter notre cas d'utilisation, tout en restant une plateforme abordable et facilement accessible pour la reproduction des résultats.

4.7 Résultat et Analyse

Le but de cette expérience est d'évaluer les deux versions de notre cas d'utilisation en mesurant le temps de réponse par requête tel que fourni par les outils de test de charge. Cette expérience nous a fourni une quantité importante de données, nous permettant d'obtenir des informations précieuses sur les performances des systèmes IoT considérés. Nous avons collecté et illustré les

métriques moyennes recueillies à travers quatre scénarios distincts d'utilisateurs virtuels (VU) (1, 10, 50, 100, 400) en tenant compte de l'intervalle de confiance statistique.

Pour évaluer la fiabilité de nos résultats dans cette analyse, nous utilisons l'intervalle de confiance statistique, représenté par le pourcentage d'erreur. Ce paramètre joue un rôle crucial dans la détermination de la précision de nos conclusions. Son calcul est basé sur la formule suivante :

$$\text{Pourcentage d'erreur} = \frac{\text{Intervalle de confiance} \times 100}{\text{Moyenne des mesures}} \tag{4.5}$$

Dans notre étude, nous avons défini un pourcentage d'erreur de 5%, ce qui signifie que l'intervalle de confiance autour de nos mesures moyennes est de ±5% de la moyenne elle-même. Ce niveau de précision est crucial pour évaluer avec précision la variation possible des performances et garantir que nos conclusions sont robustes et fiables.

Ainsi, en fixant un seuil d'erreur à 5 %, nous nous assurons que les mesures présentées dans les illustrations de ce chapitre demeurent dans une marge d'incertitude restreinte, renforçant ainsi notre confiance dans les conclusions tirées sur les performances des systèmes IoT examinés. Ce paramètre nous permet de mieux appréhender la variation des données recueillies lors des tests et d'évaluer de manière critique la précision de nos résultats.

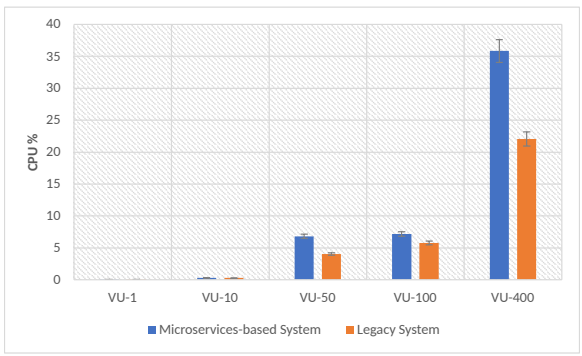
Pour garantir des données valides et fiables, nous avons utilisé des outils éprouvés pour les tests de charge, et nous avons répété nos tests pour obtenir des résultats plus précis. Le processus de collecte de données a également été conçu pour minimiser les biais et les erreurs en générant le rapport final directement à partir des outils de test de charge ou de surveillance. Cependant, l'étude a utilisé un environnement de test simulé, qui peut ne pas reproduire tous les comportements du système dans un environnement de production réel et pourrait avoir un impact sur les résultats de performance.

Dans les figures à venir, nous définissons les étiquettes de l'axe des abscisses comme suit : VU-1 représente 1 utilisateur virtuel, VU-10 correspond à 10 utilisateurs virtuels, VU-50 désigne 50

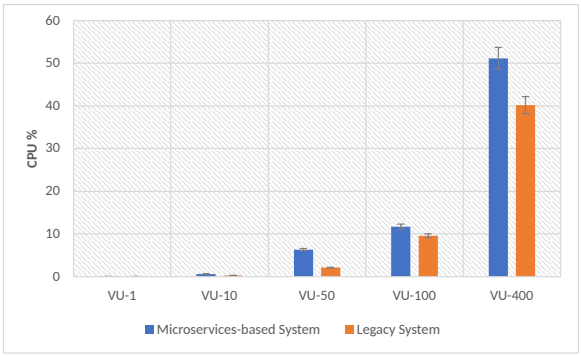
utilisateurs virtuels, VU-100 indique 100 utilisateurs virtuels et VU-400 indique 400 utilisateurs virtuels.

4.7.1 Analyse des performances du processeur

L'analyse des performances du processeur (CPU), comme illustrée dans les figures 4.11a et 4.11b, révèle des perspectives intrigantes sur le comportement des différents scénarios d'UV lors de l'exécution des deux systèmes IoT.



a) Performances du processeur - Scénario 1



b) Performances du processeur - Scénario 1

Figure 4.11 les performances du processeur dans différents scénarios de UV

En examinant ces graphiques, une tendance notable se dégage : le système IoT basé sur les microservices impose notablement une demande plus élevée sur le CPU par rapport à son homologue hérité. Cette demande accrue découle principalement de la décomposition de la structure monolithique du système hérité en microservices distincts, formant collectivement l'architecture du système IoT basé sur les microservices.

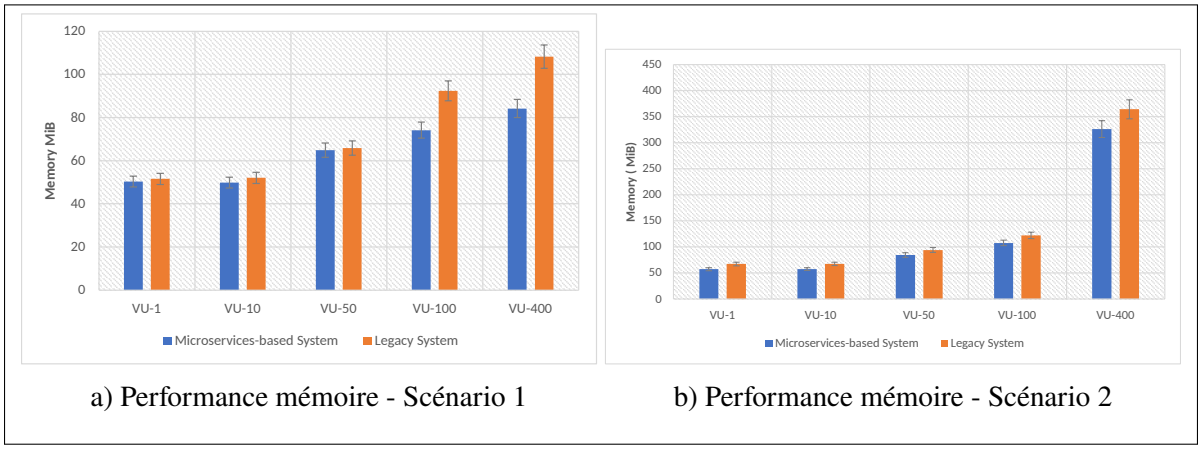
Dans le scénario initial, le système IoT basé sur les microservices a montré une utilisation du CPU comprise entre 0,04 % et 35,85 % lorsque nous avons modifié le nombre d'UV de 1 à 10, 50, 100 et 400. Tandis que l'utilisation du CPU pour le système hérité varie de 0,07 % à 22,07 %.

Dans le second cas, lorsque le nombre d'UV variait de 1 à 10, 50, 100 et 400, le système IoT basé sur les microservices a montré une utilisation du CPU allant de 0,10 % à 51,15 %, tandis que pour le système hérité, elle variait de 0,04 % à 40,20 %.

Ces résultats indiquent une tendance constante : le système IoT basé sur les microservices démontre systématiquement une utilisation plus élevée du CPU à travers différents scénarios d'UV par rapport au système hérité. Cette disparité met en évidence l'impact des changements architecturaux dans la conception du système, soulignant les compromis potentiels entre une modularité accrue du système et une demande de ressources plus élevée, notamment en ce qui concerne l'utilisation du CPU.

4.7.2 Analyse des performances mémoire

Après avoir analysé les données présentées dans la Figure 4.12a et la Figure 4.12b, une observation distincte émerge : le système basé sur les microservices présente systématiquement une utilisation de mémoire plus faible par rapport au système hérité.



a) Performance mémoire - Scénario 1 b) Performance mémoire - Scénario 2

Figure 4.12 Les performances mémoire dans différents scénarios de UV

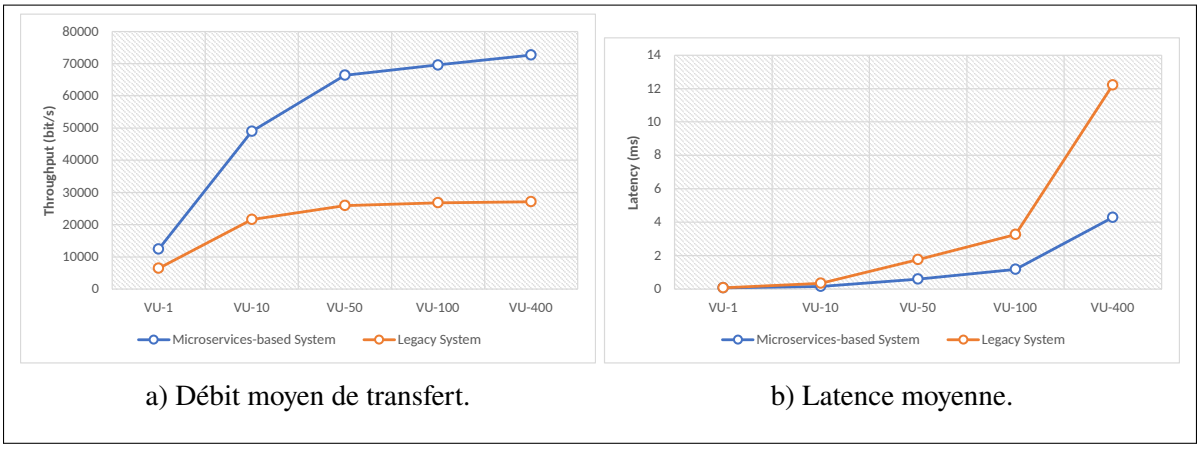
Pour approfondir cette observation, dans le scénario initial, nous avons constaté des économies de mémoire de 2,36%, 4,29%, 1,5%, 19,76% et 22,24% lorsque le nombre d'UV variait de 1 à 10, 50, 100 et 400 par rapport à l'utilisation de mémoire du système hérité. De plus, dans le

deuxième scénario, nous avons observé des économies de mémoire de 14,85%, 14,78%, 10,25%, 11,94% et 10,49% au sein du système basé sur les microservices par rapport au système hérité.

Il est important de noter certaines fluctuations dans le pourcentage d'économies de mémoire dans le premier scénario. Ces variations peuvent être attribuées à l'allocation dynamique de mémoire et aux cycles périodiques de collecte des déchets. Ces processus jouent un rôle significatif dans la gestion de l'utilisation de la mémoire, entraînant des fluctuations occasionnelles dans les économies de mémoire observées.

4.7.3 Analyse des performances réseau

Les figures 4.13a et 4.13b présentent des données sur la latence moyenne et le débit observés par les deux systèmes IoT pour différentes configurations d'utilisateurs virtuels lors des tests. Une tendance évidente se dégage, démontrant de manière cohérente la supériorité des performances du système IoT basé sur les microservices, utilisant le protocole gRPC HTTP/2.0.



a) Débit moyen de transfert. b) Latence moyenne.

Figure 4.13 les performances réseau dans différents scénarios de UV

Cet avantage se manifeste à la fois dans le débit et le temps de réponse, indépendamment du nombre d'utilisateurs virtuels impliqués. Notamment, nos observations révèlent une différence de performance substantielle, le système IoT basé sur les microservices affichant un débit nettement supérieur à celui du système hérité. Plus précisément, alors que le nombre d'utilisateurs virtuels

variait de 1 à 10, 50, 100 et 400, le système basé sur les microservices affichait des débits supérieurs de 93,07%, 126,50%, 156,25%, 160,28% et 167,81% par rapport au système hérité. Cette différence notable implique que le système basé sur les microservices possède une capacité supérieure à traiter un plus grand volume de requêtes dans le même laps de temps.

De plus, notre analyse de la latence a révélé des conclusions intrigantes. La latence du système IoT basé sur les microservices a régulièrement démontré une réduction par rapport au système hérité à mesure que le nombre d'utilisateurs virtuels variait. Plus précisément, la latence observée était respectivement inférieure de 12,5%, 54,28%, 66,47%, 63,91% et 64,94% dans le système basé sur les microservices par rapport au système hérité pour différentes configurations d'utilisateurs virtuels. Cela suggère que le système hérité subit davantage de retards soit dans la transmission, soit dans le traitement des données, entraînant des périodes de latence accrues.

Ces résultats soulignent les gains de performance substantiels réalisés par le système IoT basé sur les microservices, mettant en évidence son efficacité dans le traitement des requêtes réseau et la fourniture de temps de réponse plus rapides par rapport au système hérité, témoignant de l'efficacité de l'utilisation du protocole gRPC HTTP/2.0.

Cependant, il est important de noter que le goulot d'étranglement dans le système basé sur les microservices ne se situe pas nécessairement au niveau du réseau, mais pourrait plutôt se trouver au niveau d'un des microservices spécifiques. Cette observation soulève une question intéressante : bien que le débit soit plus élevé dans l'architecture basée sur les microservices, cela pourrait également impliquer une utilisation accrue de la bande passante. Il convient donc d'examiner si cet aspect est positif ou négatif dans le contexte global du système et des ressources disponibles.

Pour approfondir cette analyse, il serait judicieux d'évaluer la performance individuelle de chaque microservice et d'identifier les éventuels goulots d'étranglement. Cela permettrait d'optimiser davantage le système en allouant les ressources de manière appropriée et en résolvant les problèmes de performance spécifiques à chaque composant.

En conclusion, bien que le système IoT basé sur les microservices démontre des avantages significatifs en termes de débit et de latence, il est crucial de considérer l'impact sur l'utilisation des ressources, notamment la bande passante, et d'identifier les goulots d'étranglement potentiels au niveau des microservices individuels pour garantir une performance optimale et une scalabilité efficace du système dans son ensemble.

4.7.4 À retenir

Les résultats révèlent que le système IoT basé sur les microservices consomme une quantité plus importante de CPU par rapport au système hérité. En moyenne, la charge CPU a augmenté de 3,57% à 2,53% dans le Scénario 1 et de 3,52% à 4,66% dans le Scénario 2. Cependant, malgré cette utilisation accrue du CPU, on a constaté une réduction notable de l'utilisation de la mémoire, entraînant des économies moyennes de 6,98% dans le Scénario 1 et de 12,95% dans le Scénario 2 par rapport au système hérité.

De plus, la mise en œuvre du protocole gRPC dans le système IoT basé sur les microservices a présenté une amélioration impressionnante de 134,02% du débit par rapport au système hérité. En termes de métriques de latence, notre analyse a révélé que le système basé sur les microservices excellait, démontrant une efficacité 49,29% supérieure dans la livraison des réponses ou des données par rapport au système hérité.

En résumé, tout en reconnaissant l'augmentation de l'utilisation du CPU dans le système IoT basé sur les microservices, il est crucial de souligner les avantages globaux observés dans les économies de mémoire, la latence et l'amélioration notable du débit. Ces avantages sont complétés par l'intégration des meilleures pratiques en génie logiciel, telles que l'utilisation d'une architecture plus modulaire à travers des microservices conteneurisés, entre autres stratégies. Cela suggère que l'utilisation accrue du CPU dans le système IoT basé sur les microservices représente un compromis acceptable compte tenu des avantages plus larges obtenus en termes de performances système et d'améliorations architecturales.

4.8 Conclusion

Notre évaluation empirique fournit des informations précieuses sur les compromis de performance liés à l'utilisation des pratiques de génie logiciel dans un système IoT basé sur les microservices par rapport à un système IoT hérité. Bien que nous ayons observé une augmentation de l'utilisation du CPU avec l'architecture des microservices, la mise en œuvre des pratiques de génie logiciel a conduit à des gains notables en termes d'efficacité de la mémoire, de débit et de latence acceptable par rapport au système hérité.

CHAPITRE 5

MENACES À LA VALIDITÉ

Les préoccupations concernant la validité dans cette étude englobent la validité de construction, la validité interne et la validité externe.

5.1 Validité de construction

La validité de construction concerne la sélection des articles analysés et la méthodologie d'extraction des données de ces articles. Dans notre étude, nous avons relevé plusieurs préoccupations potentielles :

- **Omission d'articles pertinents** : La sélection des articles est cruciale pour assurer une couverture complète des connaissances existantes. Il est possible que des études pertinentes soient omises malgré nos efforts. Pour atténuer ce problème, nous avons effectué une recherche exhaustive dans les cinq bibliothèques numériques les plus utilisées en génie logiciel. Cette méthode est conforme aux standards établis et reconnus pour les revues de littérature systématiques dans le domaine, tel que recommandé par Dyba, Dingsoyr & Hanssen (2007).
- **Biais dans le processus de sélection des articles** : Le biais de sélection est une préoccupation majeure, car il peut influencer les résultats de la revue. Afin de réduire ce biais, nous avons procédé indépendamment à la sélection d'articles en appliquant des critères d'inclusion et d'exclusion définis au préalable. Nous avons ensuite confronté et discuté nos sélections pour résoudre toutes les divergences et parvenir à un consensus, garantissant ainsi une plus grande objectivité dans le processus de sélection.
- **Biais dans l'extraction des données** : L'extraction de données est susceptible de subir l'influence de préjugés personnels, entraînant potentiellement des erreurs d'interprétation. Pour minimiser ces biais, un processus d'extraction des données a été mis en place, suivi d'un cycle de révisions croisées entre les deux auteurs principaux. Toute incohérence ou

divergence identifiée a été discutée et ajustée de manière approfondie pour assurer la fiabilité des données extraites.

- **Dépendance à la littérature grise** : Notre recherche inclut une considérable quantité de littérature grise, qui, bien que de plus en plus acceptée dans les études de génie logiciel, présente des risques de validité en raison de la possible absence de rigueur scientifique et d'examen par les pairs. Nous avons inclus des références à des blogues, des rapports techniques et des livres blancs, qui, bien que précieux pour leur actualité et leur perspective pratique, peuvent varier considérablement en ce qui concerne la qualité et la profondeur d'analyse. La qualité inégale de ces documents peut donc menacer la validité des résultats que nous présentons. Pour atténuer ce risque, chaque source de la littérature grise a été soigneusement évaluée en fonction de sa crédibilité et de sa pertinence par rapport au sujet étudié (Garousi *et al.*, 2019; Soldani *et al.*, 2018).

5.2 Validité interne

La validité interne concerne la robustesse de la conception et des méthodes de l'étude et si les résultats peuvent réellement être attribués aux variables étudiées. Dans notre étude, il y a deux menaces principales :

- **Exhaustivité de la revue** : Notre étude cible spécifiquement les défis et les pratiques de déploiement des systèmes IoT basés sur des microservices. Il existe un risque non négligeable que des problématiques pertinentes aient été omises si elles n'ont pas été décrites dans des sources facilement accessibles ou reconnues. Pour contrer cette menace, notre revue de littérature a été étendue pour englober une variété de sources documentaires, y compris des livres blancs, des rapports techniques, et des contributions sur des blogues, en plus des articles de recherche traditionnels évalués par des pairs. Cette approche multidimensionnelle augmente la probabilité de capturer une vision complète des défis existants dans le domaine.
- **Dépendance à un seul système** : S'appuyer sur un unique système pour valider notre méthode de déploiement peut limiter la portée des conclusions tirées et leur applicabilité à d'autres contextes. Pour aborder cette limitation, il est prévu d'inclure une gamme plus

large de systèmes IoT dans les futures études. Cela permettrait de tester la robustesse et la généralisabilité de nos résultats. Par ailleurs, nous envisageons de comparer les performances opérationnelles en utilisant deux modes d'évaluation distincts : un *mode de rafale*, pour tester la réactivité du système face à une charge élevée et soudaine, et un *mode normal*, pour observer les performances du système sous une charge représentative de l'utilisation typique quotidienne. Cette analyse comparative est essentielle pour évaluer de manière exhaustive la fiabilité et l'efficacité des pratiques de déploiement proposées.

- **Comparaison entre architectures différentes** : Il est important de noter que la comparaison entre les deux versions du système WIMP présente certaines limites, car il ne s'agit pas d'une comparaison directe entre deux architectures équivalentes, mais plutôt d'une comparaison entre une architecture client-serveur traditionnelle et une architecture moderne de microservices. Bien que la comparaison ne soit pas parfaitement équitable, notre expérience vise à évaluer les avantages et les défis de l'adoption d'une architecture basée sur les microservices dans le contexte du système WIMP, en appliquant des pratiques de génie logiciel spécifiques.

5.3 Validité externe

La validité externe se rapporte à la capacité de généraliser les résultats de l'étude à des contextes différents de ceux spécifiquement étudiés. Elle est essentielle pour comprendre si les conclusions de la recherche peuvent être appliquées à une gamme plus large de situations ou restent limitées à des cas particuliers.

Notre étude présente certaines limites en matière de validité externe, notamment :

- **Limitation due à l'étude de cas spécifique** : L'étude se base sur un cas spécifique d'application IoT utilisant des microservices, ce qui pourrait ne pas refléter la diversité complète des systèmes IoT en exploitation réelle. La rareté des systèmes pleinement opérationnels qui utilisent de manière efficace les architectures à la fois héritées et basées sur des microservices restreint la généralisabilité des conclusions de l'étude.

- **Plan d'extension de la recherche** : Nous reconnaissons que les résultats actuels sont limités par la nature de l'étude de cas unique. Ainsi, nous proposons d'élargir la recherche pour inclure une variété de cas d'utilisation et examiner de manière plus approfondie les pratiques d'ingénierie individuelles. De telles études futures pourraient augmenter la validité externe en démontrant comment les résultats peuvent s'appliquer à un éventail plus large de scénarios et de contextes différents au sein de l'ingénierie des systèmes IoT.
- **Contributions futures au domaine du génie logiciel** : En tenant compte des limitations actuelles et en planifiant des recherches futures qui adressent ces préoccupations, nous visons à fournir un corpus de connaissances plus riche et généralisable. Cela peut apporter une contribution significative au domaine du génie logiciel pour les systèmes IoT qui adoptent des architectures basées sur des microservices.

CONCLUSION ET RECOMMANDATIONS

Cette étude a mis en lumière l'orientation des recherches existantes sur l'adoption des microservices dans les systèmes IoT. Bien que les microservices offrent l'avantage d'être autonomes et de pouvoir être orchestrés pour fournir des fonctionnalités spécifiques dans des environnements aux ressources limitées, les praticiens de l'IoT sont fréquemment confrontés à des problèmes d'optimisation des performances et d'utilisation des ressources.

Pour répondre à ces préoccupations, Nous avons identifié les défis actuels liés au déploiement de systèmes IoT basés sur les microservices en périphérie, l'utilisation des ressources et l'optimisation des performances étant les défis les plus fréquemment rapportés. Ces défis sont particulièrement prégnants dans le contexte de l'IoT, où les dispositifs en périphérie disposent souvent de ressources limitées en termes de puissance de calcul, de mémoire et de stockage. L'optimisation de l'utilisation de ces ressources tout en maintenant des performances élevées est donc cruciale pour garantir le bon fonctionnement et le passage à l'échelle des systèmes IoT.

Nous avons compilé un catalogue de pratiques d'ingénierie logicielle qui peuvent répondre à ces défis et avons constaté que les microservices conteneurisés, l'utilisation d'une passerelle API et l'emploi d'une base de données par microservice sont les pratiques d'ingénierie logicielle les plus répandues. Ces pratiques visent à découpler les différents composants du système, à faciliter leur déploiement et leur gestion, et à optimiser l'utilisation des ressources. Les microservices conteneurisés permettent un packaging et un déploiement aisés, tandis que la passerelle API simplifie les communications entre les microservices. L'utilisation d'une base de données par microservice évite les goulots d'étranglement et améliore les performances.

Nous avons mené une étude empirique pour mettre en œuvre ces pratiques et comparer leur impact. Nous avons utilisé des métriques de débit et de latence pour l'évaluation des performances, ainsi que des métriques d'utilisation du CPU et de la mémoire pour évaluer l'utilisation des ressources. Les résultats ont démontré des améliorations significatives du débit, de la latence

et des économies de mémoire lors de l'application de ces pratiques. Ces améliorations sont particulièrement importantes dans le contexte de l'IoT, où les ressources sont souvent limitées et où les performances et la réactivité sont essentielles. Cependant, nous avons observé qu'une augmentation de la complexité du système peut entraîner une utilisation totale du CPU plus élevée, ce qui souligne l'importance d'une conception et d'une implémentation minutieuses pour éviter les surcoûts liés à la complexité.

Nos travaux futurs étudieront les impacts individuels de ces pratiques et évalueront leurs effets sur des systèmes IoT plus grands et plus complexes. Il sera important de valider la scalabilité et la robustesse de ces pratiques dans des scénarios réels à grande échelle. Il y a également une opportunité d'étendre l'évaluation pour inclure des pratiques et des scénarios supplémentaires, afin de couvrir un éventail plus large de cas d'utilisation et de contextes IoT. Nous explorerons en outre les combinaisons potentielles de ces pratiques afin de fournir des directives d'implémentation concrètes aux praticiens. L'objectif est de proposer un ensemble de bonnes pratiques éprouvées et validées empiriquement pour la conception, le développement et le déploiement de systèmes IoT efficaces, performants et passant à l'échelle basés sur une architecture de microservices en périphérie.

ANNEXE I

ARTICLES INCLUS DANS SLR

Tableau-A I-1 Articles inclus dans notre Revue de la littérature

Titre	Année	Éditeur	Publication
A comprehensive improved salp swarm algorithm on redundant container deployment problem	2019	IEEE Access	Journal
AI-Enabled Secure Microservices in Edge Computing : Opportunities and Challenges	2023	IEEE Transactions On Services Computing	Journal
Application Deployment in Mobile Edge Computing Environment Based on Microservice Chain	2022	2022 IEEE 25th International Conference on CSCWD	Conférence
Cost-aware Deployment of Microservices for IoT Applications in Mobile Edge Computing Environment	2022	IEEE Transactions on Network and Service Management	Journal
Distributed Redundant Placement for Microservice-based Applications at the Edge	2022	IEEE Transactions on Services Computing	Journal
Dynamic On-Demand Fog Formation Offering On-the-Fly IoT Service Deployment	2020	IEEE Transactions on Network and Service Management	Journal
Edge-supported Microservice-based Resource Discovery for Mist Computing	2020	2020 IEEE 11th International Conference on DESSERT	Conférence
Optimal Application Deployment in Resource-Constrained Distributed Edges	2021	IEEE Transactions on Mobile Computing	Journal
ProScale : Proactive Autoscaling for Microservice With Time-Varying Workload at the Edge	2023	IEEE Transactions on Parallel and Distributed Systems	Journal
Research on deployment method of edge computing gateway based on microservice architecture	2021	IOP Conference Series : Earth and Environmental Science	Journal
Resource-Aware Dynamic Service Deployment for Local IoT Edge Computing : Healthcare Use Case	2021	IEEE Access	Journal
ROMA : Resource Orchestration for Microservices-based 5G Applications	2020	IEEE/IFIP Network Operations and Management Symposium	Chapitre de livre
Towards an Easily Programmable IoT Framework Based on Microservices	2018	Journal of Software	Journal
Towards cost-effective and robust AI microservice deployment in edge computing environments,	2022	Future Generation Computer Systems	Journal
V-Edge : Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing	2022	IEEE Network	Journal

➥ **DESSERT** :Dependable Systems, Services, and Technologies
➥ **CSCWD** : Computer Supported Cooperative Work in Design

BIBLIOGRAPHIE

Abdullah, A., Kaur, H. & Biswas, R. (2020). Layers of IoT architecture and its security analysis. *New Paradigm in Decision Science and Management : Proceedings of ICDSM 2018*, pp. 293–302.

Adams, R. J., Smart, P. & Huff, A. S. (2017). Shades of grey : guidelines for working with the grey literature in systematic reviews for management and organizational studies. *International Journal of Management Reviews*, 19(4), 432–454.

Ahmed, B. S., Bures, M., Frajtak, K. & Cerny, T. (2019). Aspects of quality in Internet of Things (IoT) solutions : A systematic mapping study. *IEEE Access*, 7, 13758–13780.

Akbulut, A. & Perros, H. G. (2019). Performance Analysis of Microservice Design Patterns. *IEEE Internet Computing*, 23(6), 19-27. doi : 10.1109/MIC.2019.2951094.

Aksakalli,Karabey, T. & Can,Ahmet Burak, B. (2021). Deployment and communication patterns in microservice architectures : A systematic literature review. *Journal of Systems and Software*, 180, 111014.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M. (2015). Internet of things : A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4), 2347–2376.

Al-Masri, E. (2018a). Enhancing the Microservices Architecture for the Internet of Things. *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5119-5125. doi : 10.1109/BigData.2018.8622557.

Al-Masri, E. (2018b). Enhancing the microservices architecture for the internet of things. *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5119–5125.

Alam, M., Rufino, J., Ferreira, J., Ahmed, S. H., Shah, N. & Chen, Y. (2018). Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine*, 56(9), 118–123. doi : 10.1109/MCOM.2018.1701233.

AltexSoft. [[Accessed 01-Mar-2023]]. (2020). IoT Architecture : Key Layers and Components. Repéré à https://www.altexsoft.com/blog/iot-architecture-layers-components/.

Anderson, R. [Consulté le 5 octobre 2023]. (2022). Bundle and Minify Static Assets in ASP.NET Core. Repéré à https://learn.microsoft.com/en-us/aspnet/core/client-side/bundling-and-minification?view=aspnetcore-7.0.

Anon. (2021). Shared Signature Key for JWT in Various Microservices. Repéré à https://stackoverflow.com/questions/43559197.

Anon. (n.d.a). Microservices at Netflix : Architectural Best Practices. Repéré à https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/.

Anon. (n.d.b). Microservices at Netflix : Architectural Best Practices. Repéré à https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/.

Atzori, L., Iera, A. & Morabito, G. (2010). The internet of things : A survey. *Computer networks*, 54(15), 2787–2805.

Balalaie, A., Heydarnoori, A. & Jamshidi, P. (2016). Microservices architecture enables devops : Migration to a cloud-native architecture. *Ieee Software*, 33(3), 42–52.

Berger, C., Nguyen, B. & Benderius, O. (2017). Containerized Development and Microservices for Self-Driving Vehicles : Experiences 'I&' Best Practices. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 7-12. doi : 10.1109/ICSAW.2017.56.

Blog, N. T. (2020). How Netflix Scales its API with GraphQL Federation (Part 1). *Netflix TechBlog*. Repéré à https://netflixtechblog.com/how-netflix-scales-its-api-with-graphql-federation-part-1-ae3557c187e2.

Bolanowski, M., Żak, K., Paszkiewicz, A., Ganzha, M., Paprzycki, M., Sowiński, P., Lacalle, I. & Palau, C. E. (2022). Efficiency of REST and gRPC in realizing communication tasks in microservice-based ecosystems. *arXiv preprint arXiv :2208.00682*.

Bonagiri, S. [Available online]. (2023, 10, 15). 14 Best Practices for Microservices : Boosting Efficiency with DevOps. Passionate Programmers. Repéré le 2024-01-22 à https://passionateprogrammers.com.

Burhan, Muhammad and Rehman, Rana Asif and Khan, Bilal and Kim, Byung-Seo. (2018). IoT Elements, Layered Architectures and Security Issues : A Comprehensive Survey. *Sensors*, 18(9). doi : 10.3390/s18092796.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. (2016a). Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5), 50–57.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. (2016b). Borg, omega, and kubernetes. *Queue*, 14(1), 70–93.

Butzin, B., Golatowski, F. & Timmermann, D. (2016a). Microservices approach for the internet of things. *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1-6. doi : 10.1109/ETFA.2016.7733707.

Butzin, B., Golatowski, F. & Timmermann, D. (2016b). Microservices approach for the internet of things. *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1-6. doi : 10.1109/ETFA.2016.7733707.

Campeanu, G. (2018, June). A Mapping Study on Microservice Architectures of Internet of Things and Cloud Computing Solutions. *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1-4. doi : 10.1109/MECO.2018.8406008.

Castillo Rivas, D. A. & Guamán, D. (2021). Performance and security evaluation in microservices architecture using open source containers. *Applied Technologies : Second International Conference, ICAT 2020, Quito, Ecuador, December 2–4, 2020, Proceedings 2*, pp. 484–498.

Cheng, K., Zhang, S., Tu, C., Shi, X., Yin, Z., Lu, S., Liang, Y. & Gu, Q. (2023). ProScale : Proactive Autoscaling for Microservice With Time-Varying Workload at the Edge. *IEEE Transactions on Parallel and Distributed Systems*, 34(4), 1294–1312. doi : 10.1109/TPDS.2023.3238429.

Cisco Systems, I. [[consulte 01-Mar-2023]]. (2014). IoT World Forum 2014. Repéré à https://www.globenewswire.com/news-release/2014/10/14/1271271/0/en/The-Internet-of-Things-World-Forum-Unites-Industry-Leaders-in-Chicago-to-Accelerate-the-Adoption-of-IoT-Business-Models.html.

Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S. & Gall, H. C. (2017). An empirical analysis of the Docker container ecosystem on GitHub. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 323–333.

Cooke, A., Smith, D. & Booth, A. (2012). Beyond PICO : the SPIDER Tool For Qualitative Evidence Synthesis. *Qualitative health research*, 22(10), 1435–1443.

Deng, S., Xiang, Z., Taheri, J., Khoshkholghi, M. A., Yin, J., Zomaya, A. Y. & Dustdar, S. (2021). Optimal Application Deployment in Resource Constrained Distributed Edges. *IEEE Transactions on Mobile Computing*, 20(5), 1907–1923. doi : 10.1109/TMC.2020.2970698.

Dhall, R. [Available online]. (2016). Performance patterns in microservices-based integrations. Dzone. Repéré le 2024-01-22 à https://dzone.com.

Dias, J. P., Ferreira, H. S. & Sousa, T. B. (2019). Testing and deployment patterns for the internet-of-things. *Proceedings of the 24th European Conference on Pattern Languages of Programs*, pp. 1–8.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. & Safina, L. (2017). Microservices : yesterday, today, and tomorrow. *Present and ulterior software engineering*, 195–216.

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R. & Safina, L. (2018). Microservices : How to make your application scale. *Perspectives of System Informatics : 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11*, pp. 95–104.

Dressler, F., Chiasserini, C. F., Fitzek, F. H., Karl, H., Cigno, R. L., Capone, A., Casetti, C., Malandrino, F., Mancuso, V., Klingler, F. & Rizzo, G. (2022). V-Edge : Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing. *IEEE Network*, 36(3), 24-31. doi : 10.1109/MNET.001.2100491.

Dyba, T., Dingsoyr, T. & Hanssen, G. K. (2007). Applying systematic reviews to diverse study types : An experience report. *First international symposium on empirical software engineering and measurement (ESEM 2007)*, pp. 225–234.

Felter, W., Ferreira, A., Rajamony, R. & Rubio, J. (2015a, May). An Updated Performance Comparison of Virtual Machines and Linux Containers. *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172.

Felter, W., Ferreira, A., Rajamony, R. & Rubio, J. (2015b). An updated performance comparison of virtual machines and linux containers. *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172.

Fowler, M. & Lewis, J. (2019). Microservices guide. *Retrieved May*, 15, 2020.

Francisco, O. & Donna, O. (2018). Towards an Easily Programmable IoT Framework Based on Microservices. *Journal of Software*, 13(1), 90–102. doi : 10.17706/jsw.13.2.90-102.

Gancarz, R. (2023). LinkedIn Adopts Protocol Buffers for Microservices Integration and Reduces Latency by up to 60*InfoQ*, 2023. Repéré à https://www.infoq.com/news/2023/07/linkedin-protocol-buffers-restli/.

Garousi, V., Felderer, M. & Mäntylä, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 106, 101-121. doi : https://doi.org/10.1016/j.infsof.2018.09.006.

Gaur, A. S., Budakoti, J. & Lung, C.-H. (2018). Design and Performance Evaluation of Containerized Microservices on Edge Gateway in Mobile IoT. *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 138-145. doi : 10.1109/Cybermatics_2018.2018.00055.

Gerla, M., Lee, E.-K., Pau, G. & Lee, U. (2014). Internet of vehicles : From intelligent grid to autonomous cars and vehicular clouds. *2014 IEEE world forum on internet of things (WF-IoT)*, pp. 241–246.

Gholami, A., Rao, K., Hsiung, W.-P., Po, O., Sankaradas, M. & Chakradhar, S. (2022). ROMA : Resource Orchestration for Microservices-Based 5G Applications. *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9. doi : 10.1109/NOMS54207.2022.9789821.

gRPC Authors. [Consulté le 26 mars 2024]. (2023). Introduction to gRPC. Repéré le 2024-03-26 à https://grpc.io/docs/what-is-grpc/introduction/.

Gubbi, J., Buyya, R., Marusic, S. & Palaniswami, M. (2013). Internet of Things (IoT) : A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645–1660. doi : 10.1016/j.future.2013.01.010.

Guidi, C., Lanese, I., Mazzara, M. & Montesi, F. (2017). Microservices : a language-based approach. *Present and Ulterior Software Engineering*, 217–225.

Gupta, S. [Available online]. (2022, 7, 15). Developing High-Performance Applications Microservices. Medium. Repéré le 2024-01-22 à https://medium.com.

Heaton, D. & Carver, J. C. (2015). Claims about the use of software engineering practices in science : A systematic literature review. *Information and Software Technology*, 67, 207–219.

Hofmann, M., Schnabel, E., Stanley, K. & Organization, I. B. M. C. I. T. S. (2016a). *Microservices best practices for Java*. IBM Redbooks.

Hofmann, M., Schnabel, E., Stanley, K. & Organization, I. B. M. C. I. T. S. [1 online resource (1 volume) : illustrations]. (2016b). Microservices Best Practices for Java. Poughkeepsie, NY : IBM Corporation, International Technical Support Organization. Repéré à https://etsmtl.on.worldcat.org/oclc/968749377.

Humble, J. & Farley, D. (2010). *Continuous Delivery : Reliable Software Releases through Build, Test, and Deployment Automation* (éd. xxxiii, 463 pages ; 24 cm). Upper Saddle River, NJ : Addison-Wesley.

IEEE Computer Society. [Accessed : 2024-03-10]. (2014). Guide to the Software Engineering Body of Knowledge (SWEBOK V3). Repéré à https://www.computer.org/education/bodies-of-knowledge/software-engineering.

Islam, J., Kumar, T., Kovacevic, I. & Harjula, E. (2021). Resource-Aware Dynamic Service Deployment for Local IoT Edge Computing : Healthcare Use Case. *IEEE Access*, 9, 115868–115884. doi : 10.1109/ACCESS.2021.3102867.

Islam, S. R., Kwak, D., Kabir, M. H., Hossain, M. & Kwak, K. S. (2015). The internet of things for health care : a comprehensive survey. *IEEE access*, 3, 678–708.

Jaimini, U. & Dhaniwala, M. (2016). Javascript empowered internet of things. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 2373–2377.

Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I. & Ahmed, A. (2018). Scaling IoT applications into cloud through edge computing : taxonomy and open challenges. *Internet of Things*, 97–133.

Kim, G., Humble, J., Debois, P., Willis, J. & Forsgren, N. (2021). *The DevOps handbook : How to create world-class agility, reliability, & security in technology organizations*. IT Revolution.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004), 1–26.

Krylovskiy, A., Jahn, M. & Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. *2015 3rd international conference on future internet of things and cloud*, pp. 25–30.

Kumar, L. [Available online]. (2023, 10, 10). Best Practices for Microservices : Building Scalable and Efficient ... Dzone. Repéré le 2024-01-22 à https://dzone.com.

Kumar, V. [Consulté le 26 mars 2024]. (2024). Vikram Kumar - Programmer, Blogger, Traveller. Repéré le 2024-03-26 à https://kumarvikram.com/.

Lee, J., Bagheri, B. & Kao, H.-A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, 3, 18–23.

Leotta, M., Ricca, F., Clerissi, D., Ancona, D., Delzanno, G., Ribaudo, M. & Franceschini, L. (2017). Towards an Acceptance Testing Approach for Internet of Things Systems. *ICWE Workshops*, pp. 125–138.

Li, H., Tang, B., Xu, W., Guo, F. & Zhang, X. (2022). Application Deployment in Mobile Edge Computing Environment Based on Microservice Chain. *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 531–536. doi : 10.1109/CSCWD54268.2022.9776307.

Li, S., Da Xu, L. & Zhao, S. (2015). The internet of things : a survey. *Information Systems Frontiers*, 17(2), 243–259.

Lin, J.-C., Mauro, J., Røst, T. B. & Yu, I. C. (2017). A model-based scalability optimization methodology for cloud applications. *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, pp. 163–170.

LinkedIn community. [Available online]. (2023, 10, 7). How do you optimize microservice performance ? LinkedIn. Repéré le 2024-01-22 à https://linkedin.com.

Lira, C., Batista, E., Delicato, F. C. & Prazeres, C. (2023). Architecture for IoT Applications Based on Reactive Microservices : A Performance Evaluation. *Future Generation Computer Systems*, 145, 223–238. doi : 10.1016/j.future.2023.03.026.

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J. et al. (2019). DevOps in practice : A multiple case study of five companies. *Information and Software Technology*, 114, 217–230.

Ma, B., Ni, H., Zhu, X. & Zhao, R. (2019). A Comprehensive Improved Salp Swarm Algorithm on Redundant Container Deployment Problem. *IEEE Access*, 7, 136452-136470. doi : 10.1109/ACCESS.2019.2933265.

Macías, A., Navarro, E. & González, P. (2019). A microservice-based framework for developing internet of things and people applications. *Multidisciplinary Digital Publishing Institute Proceedings*, 31(1), 85.

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R. & Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. *OASIS standard*, 12(S18), 1–31.

Martin, R. [Accessed Jan. 26, 2022]. (2014, May 8). Clean Coder Blog. Repéré à https://blog.cleancoder.com/unclebob/2014/05/08/SingleReponsibilityPrinciple.html.

martinekuan. [Consulté le 13 décembre 2023.]. (s d). Microservice architecture style - Azure Architecture Center. Repéré à https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.

Merkel, D. (2014a). Docker : Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*, (239), 2.

Merkel, D. (2014b). Docker : lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239), 2.

Minani, J. B., Sabir, F., Moha, N. & Guéhéneuc, Y.-G. (2023). A Multi-Method Study of Internet of Things Systems Testing in Industry. *IEEE Internet of Things Journal*, 1-1. doi : 10.1109/JIOT.2023.3291233.

Montgomery, D. C. (2009). *Design and analysis of experiments* (éd. 7th). Hoboken, N.J. : Wiley.

Morabito, R. (2015). Virtualization on Internet of Things Edge Devices with Container Technologies : A Performance Evaluation. *IEEE Access*, 3, 8835–8850.

Morabito, R., Kjällman, J. & Komu, M. (2015). Hypervisors vs. lightweight virtualization : a performance comparison. *2015 IEEE International Conference on Cloud Engineering*, pp. 386–393.

Morabito, R. (2017). *Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge*. New York : IEEE.

Mouat, A. (2016a). *Using Docker : Developing and Deploying Software with Containers*. O'Reilly Media, Inc.

Mouat, A. (2016b). *Using Docker : Developing and Deploying Software with Containers*. O'Reilly Media, Inc.

Mueller, J. [Available online]. (2015). Performance issue considerations for Microservices APIs. Smartbear. Repéré le 2024-01-22 à https://smartbear.com.

MuleSoft Devs. [Available online]. (2024). Microservices best practices. MuleSoft. Repéré le 2024-01-22 à https://mulesoft.com.

Newman, S. (2021). *Building Microservices*. Sebastopol, UNITED STATES : O'Reilly Media, Incorporated. Repéré à http://ebookcentral.proquest.com/lib/etsmtl-ebooks/detail.action?docID=6683096.

Nguyen, T. et al. (2019). Asynchronous messaging patterns in microservices communication. *ACM Transactions on Software Engineering and Methodology*, 28(4), 1-28.

Oliveira, F. & Mattos, J. (2019, 11). State-of-the-Art Javascript Language for Internet of Things. pp. 149-154. doi : 10.5753/sbesc_estendido.2019.8651.

Ouyang, R., Wang, J., Xu, H., Chen, S., Xiong, X., Tolba, A. & Zhang, X. (2023). A Microservice and Serverless Architecture for Secure IoT System. *Sensors*, 23(10). doi : 10.3390/s23104868.

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E. et al. (2021). The PRISMA 2020 Statement : An Updated Guideline For Reporting Systematic Reviews. *International journal of surgery*, 88, 105906.

Pahl, C. (2015). Containerization and the PaaS cloud. *IEEE Cloud Computing*, 2(3), 24–31.

Pahl, C., Brogi, A., Soldani, J. & Jamshidi, P. (2017). Cloud container technologies : a state-of-the-art review. *IEEE Transactions on Cloud Computing*, 7(3), 677–692.

Pahl, C. & Jamshidi, P. (2016). Microservices : A Systematic Mapping Study. *6th International Conference on Cloud Computing and Services Science*, pp. 137–146. doi : 10.5220/0005785501370146.

Parks, Y. [Available online]. (2023, 3, 10). Best practices for maintaining consistent API performance over time. TOROCLOUD. Repéré le 2024-01-22 à https://torocloud.com.

Paul, J. [Available online - 6 october 2023]. (n.d.). Top 10 Microservices Design Patterns and Principles - Examples. Repéré à https://javarevisited.blogspot.com/2021/09/microservices-design-patterns-principles.html.

Petersen, K., Feldt, R., Mujtaba, S. & Mattsson, M. (2008). Systematic mapping studies in software engineering. *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pp. 1–10.

Potvin, R. & Levenberg, J. (2016). Why Google stores billions of lines of code in a single repository. *Communications of the ACM*, 59(7), 78–87.

Qu, Q., Xu, R., Nikouei, S. Y. & Chen, Y. (2020). An Experimental Study on Microservices based Edge Computing Platforms. *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 836-841. doi : 10.1109/INFOCOMWKSHPS50562.2020.9163068.

Raili, N. & Savi, M. (2021). *Architecting Continuous Integration and Continuous Deployment for Microservice Architecture*. INFOTEH Symposium.

Ramu, V. B. (2023). Performance Impact of Microservices Architecture. *The Review of Contemporary Scientific and Academic Studies*, 3(6).

Ranju, R. [Available online]. (2023, 9, 11). How to Improve Performance of Microservices : Best Practices. sayonetech. Repéré le 2024-01-22 à https://sayonetech.com.

Rao, T. A. & Haq, E. (2018). Security challenges facing IoT layers and its protective measures. *International Journal of Computer Applications*, 179(27), 31–35.

Rath, C., Mandal, A. & Sarkar, A. (2022). Microservice based scalable IoT architecture for device interoperability. *Computer Standards 'I&' Interfaces*, 84, 103697. doi : 10.1016/j.csi.2022.103697.

Razzaq, A. (2020). A Systematic Review on Software Architectures for IoT Systems and Future Direction to the Adoption of Microservices Architecture. *SN Computer Science*, 1(6), 350. doi : 10.1007/s42979-020-00359-w.

Rezaei Nasab, A., Shahin, M., Hoseyni Raviz, S. A., Liang, P., Mashmool, A. & Lenarduzzi, V. (2023). An empirical study of security practices for microservices systems. *Journal of Systems and Software*, 198, 111563. doi : https://doi.org/10.1016/j.jss.2022.111563.

Richardson, C. [1 online resource (xxvii, 490 pages) : illustrations]. (2019). Microservices Patterns : With Examples in Java. Manning Publications. Repéré à https://etsmtl.on.worldcat.org/oclc/1103265366.

Samardžić, M., Šajina, R., Tanković, N. & Grbac, T. G. (2020). Microservice Performance Degradation Correlation. *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pp. 1623-1626. doi : 10.23919/MIPRO48935.2020.9245234.

Sami, H. & Mourad, A. (2020). Dynamic On-Demand Fog Formation Offering On-the-Fly IoT Service Deployment. *IEEE Transactions on Network and Service Management*, 17(2), 1026–1039. doi : 10.1109/TNSM.2019.2963643.

Sattari, A., Ehsani, R., Leppänen, T., Pirttikangas, S. & Riekki, J. (2020). Edge-Supported Microservice-Based Resource Discovery for Mist Computing. *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 462–468. doi : 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00087.

Sayfan, G. [1 online resource (494 pages)]. (2019). Hands-On Microservices with Kubernetes : Build, Deploy, and Manage Scalable Microservices on Kubernetes. Packt Publishing, Limited. Repéré à https://etsmtl.on.worldcat.org/oclc/1108562746.

Schaefer, M. [Available online]. (2023, 2, 21). 10 Best Practices for Microservices Deployment and Management. Xalt. Repéré le 2024-01-22 à https://xalt.com.

Schulz, M. (2015). Bundling and minification : an introduction. January.

Scott, J. (2018). A Practical Guide to Microservices and Containers. *MapR Data Technologies*.

Sengupta, S. [Available online]. (2023, 12, 23). 15 Best Practices for Building a Microservices Architecture – BMC ... Bmc. Repéré le 2024-01-22 à https://bmc.com.

Shadija, D., Rezai, M. & Hill, R. (2017). Towards an understanding of microservices. *25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*.

Shi, W., Cao, J., Zhang, Q., Li, Y. & Xu, L. (2016). Edge computing : Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646.

Siddiqui, H., Khendek, F. & Toeroe, M. (2023). Microservices based architectures for IoT systems - State-of-the-art review. *Internet of Things*, 23, 100854. doi : https://doi.org/10.1016/j.iot.2023.100854.

Soldani, J., Tamburri, D. A. & Van Den Heuvel, W.-J. (2018). The pains and gains of microservices : A Systematic grey literature review. *Journal of Systems and Software*, 146, 215-232. doi : https://doi.org/10.1016/j.jss.2018.09.082.

Sun, L., Li, Y. & Memon, R. A. (2017). An open IoT framework based on microservices architecture. *China Communications*, 14(2), 154–162.

Söylemez, M., Tekinerdogan, B. & Kolukısa Tarhan, A. (2022). Challenges and Solution Directions of Microservice Architectures : A Systematic Literature Review. *Applied Sciences*, 12(11). doi : 10.3390/app12115507.

Taibi, D., Lenarduzzi, V. & Pahl, C. (2017). Processes, Motivations, and Issues for Migrating to Microservices Architectures : An Empirical Investigation. *IEEE Cloud Computing*, 4(5), 22-32. doi : 10.1109/MCC.2017.4250931.

Tang, B., Guo, F., Cao, B., Tang, M. & Li, K. (2022). Cost-Aware Deployment of Microservices for IoT Applications in Mobile Edge Computing Environment. *IEEE Transactions on Network and Service Management*, 1–1. doi : 10.1109/TNSM.2022.3232503.

Team, W. [Consulté le 13 décembre 2023.]. (2016). Microservices Design & Development : Free Ebook from NGINX. Repéré à https://www.nginx.com/blog/microservices-from-design-to-deployment-ebook-nginx/.

Tighilt, R., Abdellatif, M., Moha, N., Mili, H., Boussaidi, G. E., Privat, J. & Guéhéneuc, Y.-G. (2020). On the study of microservices antipatterns : A catalog proposal. *Proceedings of the European Conference on Pattern Languages of Programs 2020*, pp. 1–13.

Tosatto, A., Ruiu, P. & Attanasio, A. (2015). Container-based orchestration in cloud : state of the art and challenges. *2015 Ninth international conference on complex, intelligent, and software intensive systems*, pp. 70–75.

Touqeer, H., Zaman, S., Amin, R., Hussain, M., Al-Turjman, F. & Bilal, M. (2021). Smart home security : challenges, issues and solutions at different IoT layers. *The Journal of Supercomputing*, 77(12), 14053–14089.

Turnbull, J. (2014). *The Docker Book : Containerization is the New Virtualization*. James Turnbull.

Wan, J., Chen, B., Imran, M., Tao, F., Li, D., Liu, C. & Ahmad, S. (2018). Toward dynamic resources management for IoT-based manufacturing. *IEEE Communications Magazine*, 56(2), 52–59.

Wu, C., Peng, Q., Xia, Y., Jin, Y. & Hu, Z. (2023). Towards cost-effective and robust AI microservice deployment in edge computing environments. *Future Generation Computer Systems*, 141, 129-142. doi : https://doi.org/10.1016/j.future.2022.10.015.

Yarygina, T. & Bagge, A. H. (2018). Overcoming security challenges in microservice architectures. *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pp. 11–20.

Yi, S., Li, C. & Li, Q. (2015). A survey of fog computing : concepts, applications and issues. *Proceedings of the 2015 workshop on mobile big data*, pp. 37–42.

Zhang, H., Liu, Z., Zhang, Y., Zhan, S., Yang, J., Wang, G. & Sun, Y. (2021). Research on Deployment Method of Edge Computing Gateway Based on Microservice Architecture. *IOP Conference Series : Earth and Environmental Science*, 675(1), 012164. doi : 10.1088/1755-1315/675/1/012164.

Zhang, Y., Qiao, D., Liu, X. & Wang, J. (2017). A performance model for IoT-fog computing systems. *IEEE Access*, 6, 1271–1278.

Zhao, H., Deng, S., Liu, Z., Yin, J. & Dustdar, S. (2022). Distributed Redundant Placement for Microservice-Based Applications at the Edge. *IEEE Transactions on Services Computing*, 15(3), 1732–1745. doi : 10.1109/TSC.2020.3013600.

Zimmermann, O. (2017). Microservices tenets. *Computer Science - Research and Development*, 32(3), 301–310. doi : 10.1007/s00450-016-0337-0.