République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Blida 1

Faculté des Sciences
Département de l'informatique

Mémoire de fin d'études en vue de l'obtention du Diplôme

***MASTER* EN INFORMATIQUE**

Option : Système Informatique & Réseaux

THÈME :

Solution à base de microservices dans un environnement cloud computing

Réalisé par :

Menadi Zineeddine
Bendraouche Billel

Devant le jury :

Mme. AROUSSI Sana (Présidente)
Mme. Ghebghoub Yasmina (Examinatrice)
Mme. MANCER Yasmine (Promotrice)

Promotion : 2021/2022

REMERCIEMENTS

Nous tenons à remercier le bon **Dieu** de nous avoir donné la force et le courage pour accomplir ce modeste travail.

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui nous voulons témoigner toute notre reconnaissance.

Nous tenons à remercier tout particulièrement notre promotrice **Mme. MANCER** pour sa patience, et surtout pour sa confiance, ses remarques et ses conseils, sa disponibilité et sa bienveillance.

Nos remerciements vont également aux membres de jury pour avoir acceptés de participer à ce jury de mémoire comme examinateurs. Nous adressons nos sentiments les plus respectueux.

Nous remercions aussi tous les professeurs, intervenantes et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé nos réflexions et ont acceptéde répondre à nos questions durant notre recherche.

Finalement, nous remercions nos chers parents, nos familles et tous nos amis.

Merci à tous

Dédicaces

“

Je dédie ce modeste travail :

A mes chers parents pour tous leurs sacrifices, leur amour, leur tendresse, leursoutien et leurs prières tout au long de mes études.

A mes étoiles de ma vie, mes sources de patience, de soutien et d'amour, mes trois sœurs Noussaiba, Soumia à qui je souhaite beaucoup de succès et de réussite, sans oublier Roffaida Allah yarhamha .

A mon cousin Khierddine, qui est toujours présent à mes côtés. Merci pour ta compréhension, tes encouragements et ton amour.

A mon chère frère, ami et mon binôme Zineeddine qui a partagé avec moi ce modeste travail, merci pour ta disponibilité, ta patience, ton encouragement, je te remercie pour ton aide et surtout pour tous les moments inoubliables qu'on a pu vivre ensemble, Merci beaucoup Zineddine, je t'aime.

A tous mes cousins, mes cousines, mes oncles Mustapha, Djilali et Mokhtar et mes

tantes Hafidha, Djamila et Bahria, ainsi qu'à ma belle-famille.

A mes grandes mères qui n'ont pas cessé de prier pour moi, qu'Allah leur procure labonne santé dans leur vie et le Paradis, inchaAllah.

*Aux personnes qui m'ont toujours aidé et encouragé, qui étaient toujours à mes côtés, et qui m'ont accompagné durant mon chemin d'études, mes aimables amis, collègues d'étude Farah, Abdelmadjid, Rèdha , Riyadh, Ibrahim, Sohaibe , **Abdelfattah,** Omar et Aymen.*

Et à tous ceux qui ont contribué de près ou de loin pour que ce projet soit possible,je vous dis merci.

Billel”

“

Je dédie ce modeste travail

***A** mes chers parents qui m’ont soutenu et encouragé durant ces années d’études. Qu’ils trouvent ici le témoignage de mes profondes reconnaissances.*

***A** mes chers frères Amine, Leila qui ont partagé avec moi tous les moments d’émotion lors de la réalisation de ce travail. Ils m’ont chaleureusement supporté et encouragé tout au long de mon parcours.*

*A ma chère grande-mère**,** à qui je souhaite une bonne santé*

A mon cousin Oussama, qui est toujours présent à mes côtés. Merci pour ta compréhension, tes encouragements et ton amour.

A mon cher frère, ami et mon binôme Billel pour son soutien moral, sa patience etsa compréhension tout au long de ce projet.

*Sans oublier, tous mes amis **Abdelfattah**, Riyad, Omar , Nacer ,Yasser qui m’ont toujours encouragé, et à qui je souhaite plus de succès.*

***A** tous ceux que j’aime*

***M**erci !*

Zineeddine”

Abstract

Microservices or FaaS services have become very popular in the development field day by day. Many people consider microservices based architecture as the future of IT architecture. The use of context is helpful in finding the microservices that more precisely meet the needs of the users. Microservice response time is one of the most important quality metrics of a microservice. It is in this context that our dissertation focuses on the problem of selecting the microservice instance that has the smallest response time. Our work uses the context of the location of microservice instances as a selection criterion. We propose a middleware solution between clients and vendor microservices based on the microservices architecture. This solution allows customers to select the closest microservice instance that best meets the needs of these customers.

We have proven the effectiveness of our solution through a simulation of our application using a virtualization tool that allowed us to simulate the network between different microservices of the application.

Keywords: Cloud Computing, Microservice, Discovery, Selection, Function as a Service(FaaS), Context, Quality Of Service , Provider , Customer.

Résumé

Les microservices ou les services de type FaaS sont devenus de jour en jour très populaires dans le domaine du développement. Beaucoup de gens considèrent l'architecture basée sur les microservices comme l'avenir de l'architecture informatique. L'utilisation du contexte est utile pour trouver les microservices qui répondent plus précisément aux besoins des utilisateurs. Le temps de réponse de microservice est l'un des paramètres de qualité les plus importants d'un microservice. C'est dans ce cadre que notre mémoire se concentre sur le problème de sélection de l'instance de microservice qui a le plus petit temps de réponse. Notre travail utilise le contexte de la localisation des instances des microservices comme critère de sélection. Nous proposons une solution Middleware entre les clients et les microservices des fournisseurs basée sur l'architecture microservices. Cette solution permet aux clients de sélectionner la plus proche instance de microservice qui réponde au mieux aux besoins de ces clients.

Nous avons prouvé l'efficacité de notre solution à travers une simulation de notre application à l'aide d'un outil de virtualisation qui nous a permis de simuler le réseau entre différents microservices de l'application.

Mots-clés : Cloud Computing,Microservice, Découverte, Sélection, Fonction en tant que service (FaaS), Qualité de service, contexte, Fournisseur, Client.

ملخص

أصبحت الخدمات المصغرة أو الخدمات من نوع FaaS شائعة جدًا يومًا بعد يوم في مجال التطوير. يرى الكثير من الناس أن البنية القائمة على الخدمات المصغرة هي مستقبل هندسة تكنولوجيا المعلومات. يعد استخدام السياق مفيدًا في العثور على الخدمات المصغرة التي تلبي احتياجات المستخدم بدقة أكبر. يعد وقت استجابة الخدمة المصغرة أحد أهم معايير الجودة للخدمة المصغرة. في هذا السياق ، تركز أطروحتنا على مشكلة اختيار مثيل الخدمة المصغرة التي لديها أقل وقت استجابة. يستخدم عملنا سياق موقع مثيلات الخدمات المصغرة كمعيار اختيار. نحن نقدم حلاً للبرامج الوسيطة بين العملاء والخدمات الصغيرة للموردين استنادًا إلى بنية الخدمات المصغرة. يتيح هذا الحل للعملاء تحديد أقرب مثيل خدمة مصغرة يلبي احتياجات هؤلاء العملاء على أفضل وجه. لقد أثبتنا فعالية حلنا من خلال محاكاة تطبيقنا باستخدام أداة افتراضية سمحت لنا بمحاكاة الشبكة بين الخدمات الصغيرة المختلفة للتطبيق.

كلمات مفتاحية : حوسبة سحابية، الخدمات المصغرة ، الاكتشاف ، الاختيار ، الوظيفة كخدمة (FaaS) ، السياق ، جودة الخدمة ، المزود ، العميل

Table des matières

Remerciements . . . . . **I**

Dédicaces ..... **II**

Résumé ..... **VIII**

Abstract ..... **VI**

ملخص ..... **VII**

Introduction Générale ***1***

Chapitre I GÉNÉRALITÉS SUR LE CLOUD COMPUTING ***3***

I.1 Introduction **3**

I.2 Historique **3**

I.3 Définition du cloud computing **4**

I.4 Caractéristiques du Cloud Computing **5**

I.5 Types d'hébergement de Cloud Computing **7**

I.5.1 Cloud public 7

I.5.2 Cloud privé 8

I.5.3 Cloud hybride 9

I.5.4 Cloud Communautaire 11

I.6 Types des services de Cloud Computing **11**

I.6.1 Platform as a Service (PaaS) 11

I.6.2 Infrastructure as a Service (IaaS) 12

I.6.3 Software as a Service (SaaS) 12

I.6.4 Function as a Service (FaaS) 13

I.7 Principaux avantages du cloud computing **14**

I.7.1 Réduction des coûts 14

I.7.2 Vitesse et agilité 14

I.7.3 Mise à l'échelle mondiale 14

I.7.4 Flexibilité 14

I.7.5 Sécurité des données 15

I.7.6 Performances 15

I.8 Problèmes de cloud computing 15

I.9 Conclusion 16

Chapitre II LES ARCHITECTURES LOGICIELLES 17

II.1 Introduction 17

II.2 Architecture monolithique 17

II.2.1 Définition 17

II.2.2 Caractéristiques de l'architecture monolithique 18

II.2.3 Limites de l'architecture monolithique 19

II.3 Architecture orientée services SOA 19

II.3.1 Définition 19

II.3.2 Définition d'un service 20

II.3.3 Principes fondamentaux du SOA 20

II.3.4 Avantages de SOA par rapport à une approche monolithique: 21

II.3.5 Type de services SOA 21

II.3.6 Limitations du SOA 22

II.4 Architecture Microservices 22

Définition 23

II.4.1 Caractéristiques de l'architecture microservices 23

II.4.2 Défis de l'architecture microserivces 25

II.4.3 Migration de monolithe vers l'architecture microservices : 25

II.4.3.1 Comparaison entre les architectures monolithe et microservice 26

II.4.3.2 Transition des applications monolithes vers les application microservices basés sur le cloud 27

II.4.3.3 Les phases de migration 28

II.4.4 DevOps 29

II.4.4.1 Définition 29

II.4.4.2 Fonctionnement de DevOps 29

II.4.4.3 Relation des microservices avec DevOps 30

II.4.4.4 Méthode Agile 30

II.4.5 La virtualisation et la conteneurisation 31

II.4.5.1 La virtualisation 31

II.4.5.2 Conteneurisation 32

II.4.5.3 Différence entre la virtualisation et la conteneurisation [52] 33

II.4.5.4 Hyperviseur 34

II.4.6 Les microservices dans le cloud computing 35

II.4.7 Les microserivces et la conteneurisation 36

II.4.7.1 Microservice et docker 37

II.5 Conclusion 38

Chapitre III SOLUTIONS EXISTANTES POUR LA DÉCOUVERTE ET LA SÉLECTION DES MICROSERVICES 39

III.1 Introduction 39

III.2 La découvertes des microservices 39

III.2.1 Définition de la découvrabilité 39

III.2.2 Principaux modèles de la découverte des services 39

III.2.2.1 Coté client 39

III.2.2.2 Coté serveur 40

III.2.3 Les technologies existantes pour la découverte 41

III.2.3.1 Eureka 41

III.2.3.2 Zookeeper 41

III.2.3.3 Consul 41

III.2.3.4 Tableau comparatif entre différentes technologies de la découverte 42

III.3 La sélection des microservices 43

III.3.1 Solution de J.Shao et al. [61] 43

III.3.2 Solution de Zhiying Cao et al. [64] 44

III.3.3 Solution de Zeina Houmani el al. [66] 44

III.3.4 Étude comparative des solutions proposées pour la sélection 45

III.3.5 Discussion sur les solutions proposées par les chercheurs : 47

III.4 Les recherches liées aux api gateway existes pour la sélection des microservices 48

III.4.1 Définition d'API Gateway 48

III.4.2 API GATEWAY Zuul 49

III.4.3 Spring Cloud Gateway 50

III.4.3.1 Fonctionnement de spring cloud gateway 50

III.4.3.2 Traitement de la requête de client et les filtres appliquées 52

III.4.4 La différence principale Zull et Spring Cloud Gateway 52

III.5 Conclusion 53

Chapitre IV CONCEPTION DE LA SOLUTION PROPOSÉE 54

IV.1 Introduction 54

IV.2 Méthode de travail 54

IV.2.1 Les caractéristiques de la méthodes XP 55

IV.3 Analyse et expression des besoins 56

IV.3.1 Acteur de système 56

IV.3.2 Diagrammes de cas d'utilisation 56

IV.3.2.1 Côté client : 56

IV.3.2.2 Coté Fournisseur 57
IV.3.2.3 Côté administrateur 59
IV.4 Description de la Solution 59
IV.5 L'architecture générale de notre système 60
IV.5.1 Le fournisseur 61
IV.5.2 Le client 61
IV.5.3 Le Middleware 62
IV.5.3.1 Service configuration 62
IV.5.3.2 Le proxy « Gateway » 63
IV.5.3.3 Service registre 67
IV.5.3.4 Service Registration et Authentification Client/ Fournisseur 68
IV.6 Conclusion 69
Chapitre V IMPLÉMENTATION ET SIMULATION DE LA SOLUTION PROPOSÉE 70
V.1 Introduction 70
V.2 Ressources matérielles 70
V.3 Outils et environnement de développement 70
V.4 L'implémentation de l'application 76
V.4.1 Concepts de Base spring boot 76
V.4.1.1 Dépendances et annotations utilisés 76
V.4.1.2 Implémentation des entités 77
V.4.1.3 Implémentation des Contrôleurs 77
V.4.1.4 La couche d'accès aux données 79
V.5 Architecture de l'application 79
V.5.1 Implémentation du microservice config 80
V.5.2 Implémentation de microservice proxy 81
V.5.3 Implémentation du microservice Registration et Authentification 82
V.5.3.1 Partie Inscription : 83
V.5.3.2 Partie Authentification 85
V.5.4 Implémentation de microservice registre 86
V.6 Présentation des interfaces principales de l'application 89
V.6.1 Interface d'accueil 89
V.6.2 L'interface d'inscription 90
V.6.3 L'interface de connexion 91
V.7 Espace de fournisseur 91
V.7.1 Interface d'accueil de fournisseur 91

V.7.2 Interface d'ajout d'un microservice 92
V.7.3 Interface de liste des microservices.....93
V.8 Espace client.....93
V.9 Simulation de notre application : 96
V.9.1 Création de Jar pour chaque microservices :.....96
V.9.2 Creation des image docker des microservices : 98
V.9.3 Explication des instructions de docker file 99
V.9.4 Création de fichier yml de docker-compose 100
V.9.5 Lancement des conteneurs.....104
V.10 Évaluation de la solution par rapport à l'état del'art.....104
V.11 Conclusion.....106
CONCLUSION ET PERSPECTIVES*.....*107
Références bibliographique*.....*109

Liste des tableaux

Tableau 1 Caractéristiques du Cloud Computing 6

Tableau 2 Comparaison des fonctionnalités entre les monolithes et les microservices 27

Tableau 3 La différence entre la virtualisation la Conteneurisation 34

Tableau 4 Comparaison entre les différentes technologies proposées pour la découverte 42

Tableau 5 Solutions proposées pour la sélection des microservices. 47

Tableau 6 Le rôle de chaque acteur de système 56

Tableau 7 Description de cas d'utilisation 57

Tableau 8 Description de cas d'utilisation de fournisseur 58

Tableau 9 Description de cas d'utilisation de fournisseur 59

Tableau 10 Algorithme gateway_algo 64

Tableau 11 Explication des fonctions utilisées dans l'algorithme 66

Tableau 12 Fonctionnement des annotations de controlleur 78

Tableau 13 L'explication des instruction de docker file 99

Tableau 14 Explication des instruction de fichier yaml 103

Tableau 15 Évaluation de la solution par rapport à l'état de l'art 106

Liste des Figures

Figure 1 Cloud privé [10].....9

Figure 2 Cloud hybride [11].....10

Figure 3 Platfrom as a Service (PaaS) [11].....12

Figure 4 Architecture monolithique [26].....18

Figure 5 Principes fondamenteaux du SOA [33].....21

Figure 6 Architecture microservices [26].....23

Figure 7 Monolithiques vers microservices [24].....27

Figure 8 Les phases de migration [23].....28

Figure 9 Les phases DevOps[41].....29

Figure 10 Les phases de la méthode agile [48].....31

Figure 11 Comparaison entre virtualisation et conteneurisation [53].....34

Figure 12 Fonctionnement de docker [60].....37

Figure 13 Découverte coté client [70].....40

Figure 14 Découverte coté serveur [70].....40

Figure 15 Intégration de api Gateway [74].....49

Figure 16 Fonctionnement de proxy zull [76].....50

Figure 17 Fonctionnement de l'API spring cloud gateway [79].....51

Figure 18 Aperçu de haut niveau du fonctionnement de Spring Cloud Gateway[79].....52

Figure 19 Diagramme de cas d'utilisation de côté client.....57

Figure 20 Diagrame de cas d'utilisation de côté d'fournisseur.....58

Figure 21 Diagrame de cas d'utilisation de côté de middelware.....59

Figure 22 Architecture globale de la solution.....60

Figure 23 Service Configuration.....62

Figure 24 Microservice proxy (gateway).....66

Figure 25 Service Registre.....67

Figure 26 Service Registration et Autentification Client/ Fournisseur.....69

Figure 27 Controller de microservice registre 79
Figure 28 Class main de microservice config.....80
Figure 29 Fichier de configuration de service config 80
Figure 30 Configuration des microservices dans github.....81
Figure 31 Fonction qui reçoit et traite la requête d'inscription du client.....81
Figure 32 Fonction qui reçoit et traite la requête d'inscription de fournisseur.....82
Figure 33 Fonction qui traite la requête de demande de consommation d'un service.....82
Figure 34 Fonctions qui traitent les requêtes « upload » des document WSDL.....82
Figure 35 Application properties de microservice registration et authentification.....83
Figure 36 La class service de component registre 83
Figure 37 La class authController.....84
Figure 38 Base de donnée de fournisseur 84
Figure 39 La méthode login.....85
Figure 40 La méthode Sign in.....86
Figure 41 Serveur des fichiers WSDL 86
Figure 42 Table de base de données WSDL 87
Figure 43 Analyse de code XML (WSDL).....87
Figure 44 Méthodes fournies par DOM PARSER.....88
Figure 45 Liaison de microserivce registre avec la base de donnée db_wsdl.....88
Figure 46 Logo de CloubBroker Quick. 89
Figure 47 Page de présentation du site.....89
Figure 48 Page de présentation du site (suite) 90
Figure 49 Interface d'inscription de fournisseur.....90
Figure 50 Interface de connexion.....91
Figure 51 Page d'accueil fournisseur 92
Figure 52 Interface d'inscrire un microservice 92
Figure 53 Liste des microservices de fournisseur.....93

Figure 54 Interface de client pour choisi le service 93

Figure 55 Liste des instances fournis par les fournisseur 94

Figure 56 Interface graphique de service addition.....94

Figure 57 La classe service de component addition 95

Figure 58 ProxyControlleur de microservice proxy 95

Figure 59 Spring-boot-maven-plugin.....96

Figure 60 Cleaning a Maven project.....97

Figure 6 Installing a Maven project 97

Figure 62 Dockerfile de microservices config.....98

Figure 63 Dockerfile microservices config.....98

Figure 64 Construire une image à partir de Dockerfile 100

Figure 65 Fichier yaml de docker compose 103

Figure 66 Commande pour lancer tous les conteneurs 104

Figure 67 Liste des containers lancé dans Docker Desktop.....104

Liste des abréviations

IBM : International Business Machines Corporation.

AWS : Amazon Web Services

NIST : National Institute of Standards and Technology

VPN : Virtual Private Network

LAN : Local Area Network

SOA : Service Oriented Architecture

API : Application programming interface

XML : eXtended Markup Language

MSA : Microservices Architecture

VM : Virtual Machine

OS : Operating System

URL : Uniform Resource Locator

QoS : Quality of Service

http : Hypertext Transfer Protocol

Wsdl : Web Services Description Languages

XP : eXtreme Programming

JSON : JavaScript Object Notation

MVC : Model View Controller

DOM : Document Object Model

SQL : Structured Query Language

IT : Information Technologies

Introduction Générale

A l'époque, les entreprises étaient familiarisées à faire les traitements, les stockages de données et les applications informatiques localement sur leurs serveurs et disques durs. L'explosion numérique considérable rencontrée par les entreprises en ces dernières années a engendré des coûts énormes tels que l'entretien, l'achat et la gestion des équipements. Ces obstacles ont forcé les chercheurs à inventer l'idée du cloud computing .L'utilisation du cloud computing à permis de transformer les aspects de la vie quotidienne par l'omniprésence de logiciels fonctionnant sur des réseaux cloud. En tirant profit du cloud computing, les entreprises et les startups sont en mesure d'optimiser les coûts et d'augmenter leurs offres sans achats et gestion de matériels ou de logiciels. L'adoption d'une technique pour développer des applications cloud, où l'application est constituée d'un ensemble particulier de modules spécifiques dans lesquels l'application fonctionne de manière autonome et communique via des API, est appelée microservice [1].

À mesure que les organisations adoptent l'infrastructure cloud, les architectures d'application évoluent parallèlement vers les microservices. Ces deux tendances constituent toutes deux parties d'un objectif plus large : accroître l'efficacité des développeurs et permettre une livraison plus rapide de nouvelles fonctionnalités et capacités.

Les approches de microservices conviennent particulièrement bien au cloud computing, permettant aux avantages économiques des microservices de compléter les avantages économiques du cloud computing, tels que l'optimisation des coûts. Les microservices dans le cloud sont souvent déployés dans des conteneurs car c'est ainsi qu'on peut bénéficier au maximum de l'infrastructure et de ressources.

1-Problématique

Malgré le nombre considérable d'avantages et la réussite apportés par l'architecture microservices au développement d'applications dans les entreprises tels que l'évolutivité et la facilité des mise à jour, il existe beaucoup de problèmes non encore résolus et en attente des solutions par les chercheurs. Parmi ces problèmes on trouve ceux qui proviennent des services web classiques et autres provenant des microservices eux-mêmes. Notre problème concerne la sélection et la découverte des meilleures instances de microservices des fournisseurs selon le critère de minimisation de la latence.

2-Objectif

L'objectif principal de ce travail est de créer une solution Middleware entre les fournisseurs des services cloud de type FaaS (Function as a Service) et les clients, qui sont les consommateurs des services. Pour cela, nous avons réalisé une application qui permet au Middleware de sélectionner au client les meilleures instances des microservices fournis par les fournisseurs dans un environnement cloud selon le critère d'optimisation de temps de réponse (réduction de la latence).

3-Organisation du mémoire

Notre mémoire est composé de cinq chapitres. Dans le premier chapitre nous avons présenté des généralités sur le cloud computing. On a terminé ce chapitre par citer les avantages et les inconvénients du cloud computing. Le deuxième chapitre est une étude sur l'architecture microserivces. On a présenté quelques définitions liées à cette architecture puis on a comparé les architectures microservices, monolithique et SOA (Service Oriented Architecture). On a aussi parlé de la migration de l'approche monolithique vers les microservices. Le troisième chapitre est consacré à une revue de littérature sur quelques solutions proposées par les chercheurs basées sur le contexte. Ce chapitre est divisé en deux parties. Dans la première partie, la sélection des microservices est présentée et discutée, tandis que dans la deuxième partie, nous avons présentés les technologies existantes dans la découverte des microservices. Nous avons clôturé ce chapitre avec une comparaison entre les solutions proposées par les chercheurs, suivie par une discussion sur ces solutions. Le quatrième chapitre est consacré à la description de notre solution proposée où on a décrit la conception de l'architecture de notre solution. Le dernier chapitre est dédié aux tests et l'implémentation.

A la fin de ce mémoire une conclusion générale est présentée, et quelques perspectives pour le futur sont envisagées.

Chapitre I GÉNÉRALITÉS SUR LE CLOUD COMPUTING

I.1 Introduction

Au passé, les entreprises avaient l'habitude de traiter et de stocker leurs données informatiques et leurs applications, localement au niveau de leurs propres serveurs et disques durs. Avec l'accélération numérique considérable, la multiplication de leurs tâches, et la croissance démographique constatés ces dernières années, ceci avait obligé les entreprises à agrandir leurs infrastructures et espaces de travail. Malheureusement, cet agrandissement a causé des problèmes énormes à ces entreprises, à savoir l'élévation des coûts d'achats des matériels, problèmes des locaux, problèmes de maintenance et entretiens, gestion des pannes, gestion des servers par exemple les hostings, la sécurité, un coût supplémentaire induit par l'obligation de recrutement supplémentaire d'un personnel spécialisé ainsi que les risques de saturation en cas de piques …etc. Vu tous les problèmes cités, ceci a amené les chercheurs à faire des investigations et proposer des méthodes pour faire sortir les entreprises de cette impasse.

Dans ce chapitre, nous allons donner un aperçu global sur le Cloud Computing. Nous commençons à donner l'histoire du Cloud Computing, suivie de sa définition. On a présentera ensuite les types d'hébergement de Cloud Computing puis nous donnerons une description des différents types des services offerts par les fournisseurs de cloud. Après nous présentons certaines caractéristiques de Cloud Computing.

Nous terminons ce chapitre par donner quelques avantages et inconvénients du Cloud Computing.

I.2 Historique

L'histoire de Cloud Computing remonte aux technologies antérieures qui étaient utilisées en temps réel bien avant l'invention de Cloud Computing. Dans « Cloud Computing », le mot « Cloud » désigne un opérateur ou un fournisseur qui propose des services sur Internet. « Computing » est le traitement ou calcul de diverses ressources qui sont fournies par l'ordinateur. Le concept de cloud computing remonte à John McCarthy au MIT en 1961 « Si les ordinateurs du type que j'ai préconisé deviennent les ordinateurs du futur, alors l'informatique pourrait un jour être organisée comme un service public tout comme le système téléphonique est un service public. L'utilitaire informatique pourrait devenir la base

d'une nouvelle et importante industrie » [1]. De nombreux acteurs de l'industrie se sont lancés dans le cloud computing et l'ont mis en œuvre. Par exemple, Amazon a joué un rôle important et a lancé Amazon Web Service (AWS) en 2006. Parallèlement, Google et IBM ont également lancé des projets de recherche sur le Cloud Computing. Eucalyptus devient la première plateforme open source pour déployer les clouds privés [2]. En juillet 2010, la NASA et Rackspace ont lancé un projet conjoint appelé OpenStack avec plusieurs fournisseurs, dont AMD, Intel et Dell. Plus tard, de nombreuses autres organisations se sont jointes au projet. Aujourd'hui, plus de 500 entreprises soutiennent ce projet[3]. Actuellement, l'un des principaux utilisateurs des services cloud sont les développeurs d'applications. En 2016, le cloud a commencé à passer de convivial pour les développeurs à axé sur les développeurs. Bien que les conteneurs primitifs existent depuis 2004 (conteneurs Solaris), ces premiers conteneurs étaient très limités à certains systèmes informatiques. Ce n'est qu'en 2013, lorsque Docker a proposé un conteneur extrêmement fonctionnel, que ces outils ont fait leur chemin. Ce n'est pas un hasard si la croissance de Docker et de l'utilisation des conteneurs et de Docker s'est produite simultanément. En 2017, des centaines d'outils qui existaient depuis des années ont été modifiés et utilisés pour faciliter le travail avec les conteneurs.

I.3 Définition du cloud computing

Le cloud computing, littéralement "informatique dans le cloud" est un nouveau concept qui sert à déporter des services de stockages et de traitements des données informatiques traditionnellement situées sur des serveurs locaux au niveau de serveur de l'utilisateur vers des serveurs distants. Il consiste à proposer des services à la demande, disponibles et accessibles par n'importe qui, n'importe quand et n'importe où. Grâce à un système d'identification via un PC et une connexion internet. Cette définition est un peu difficile et complexe à comprendre, mais l'idée principale à retenir est que le cloud n'est pas un ensemble de technologies mais un modèle de fourniture et de gestion et de consommation de services et de ressources informatiques[4].

Il existe plusieurs autres définitions du Cloud Computing dont voici quelques-unes:

Selon IBM [5] , «Le cloud computing, parfois simplement appelé "cloud", est l'utilisation de ressources informatiques - serveurs, gestion de bases de données, stockage de données, mise en réseau, applications logicielles et capacités spéciales, telles que la blockchain et l'intelligence artificielle (IA) »

Selon Amazon Web Services[6] , «Le cloud computing est la mise à disposition de ressources informatiques à la demande via Internet, avec une tarification en fonction de l'utilisation. Au lieu d'acheter, de posséder et de gérer des serveurs et des centres de données physiques » .

Selon Google Cloud «Le cloud computing est la disponibilité à la demande des ressources informatiques en tant que services sur Internet. Cela évite aux entreprises d'avoir à effectuer des achats, ainsi qu'à configurer ou gérer les ressources elles-mêmes. De plus, elles ne paient que pour ce qu'elles utilisent». [1]

Selon Microsoft[7], «En termes simples, le cloud computing est la fourniture de services informatiques, y compris les serveurs, le stockage, les bases de données, la mise en réseau, les logiciels, l'analyse et l'intelligence, sur Internet ("le cloud") pour offrir une innovation plus rapide, des ressources flexibles et des économies d'échelle. Vous ne payez généralement que pour les services cloud que vous utilisez, ce qui aide à réduire les coûts d'exploitation, à gérer l'infrastructure plus efficacement et à évoluer en fonction de l'évolution des besoins de l'entreprise».

Pour Rajaraman et al. [8] «Le cloud computing est une méthode d'utilisation des ressources informatiques d'un fournisseur, à la demande, par un client utilisant un ordinateur connecté à Internet».

I.4 Caractéristiques du Cloud Computing

Le Cloud Computing a cinq caractéristiques essentielles définies par Bairagi et al. [2][21]:

Le tableau 1 suivant résume cinq caractéristiques essentielles définies par Bairagi :

Libre-service	Un client peut se prévaloir de n'importe quelle ressource informatique sous contrat telle que la puissance de traitement, l'espace de stockage ou les programmes d'application d'un fournisseur de services sans interaction humaine.
Large accès au	Les ressources informatiques sont accessibles partout et à tout

[1] Google Cloud, "Qu'est-ce que le Cloud Computing," *Journal of Industrial Engineering and Management*, 2013. https://www.numergy.com/centre-de-ressources/article/quest-c.e-que-le-cloud-computing (accessed Apr. 09, 2022).

réseau	moment avec n'importe quel appareil standard pouvant accéder au Web. En d'autres termes, les services cloud sont disponibles sur un réseau - idéalement une liaison de communication à haut débit - comme Internet, ou dans le cas d'un cloud privé, il peut s'agir d'un réseau local (LAN)
Mutualisation et des ressources	Les ressources informatiques d'un fournisseur sont mises en commun pour fournir le service contracté. Les ressources mises en commun peuvent être réparties géographiquement sur plusieurs datacenters. Les ressources informatiques d'un fournisseur sont partagées par plusieurs clients. Les ressources sont affectées dynamiquement aux clients en fonction de la demande. Habituellement, un client n'a aucune connaissance de l'emplacement des ressources qui peuvent se trouver n'importe où dans le monde
Élasticité et évolutivité rapides	Les ressources informatiques peuvent être utilisées de manière élastique par les clients. Un client peut demander plus de ressources lorsqu'il en a besoin et les libérer lorsqu'il n'en a pas besoin
Service mesuré	L'utilisation des ressources de cloud computing est mesurée et les entreprises de fabrication paient en conséquence pour ce qu'elles ont utilisé. L'utilisation des ressources peut être optimisée en tirant parti des capacités de facturation à l'utilisation. Cela signifie que l'utilisation des ressources cloud, qu'il s'agisse d'instances de serveur virtuel en cours d'exécution ou de stockage dans le cloud, est surveillée, mesurée et signalée par le fournisseur de services cloud. Le modèle de coût est basé sur le « payez pour ce que vous utilisez » - le paiement est variable en fonction de la consommation réelle par l'organisation de fabrication

Tableau 1: Caractéristiques du Cloud Computing

Les Clouds computing sont des systèmes adaptatifs. Ils équilibrent automatiquement les charges et optimisent l'utilisation des ressources. Un utilisateur est autorisé à surveiller et à contrôler l'utilisation des ressources, assurant ainsi la transparence des factures.

I.5 Types d'hébergement de Cloud Computing

L'hébergement dans le cloud offre un accès pour les utilisateurs aux applications et aux sites Web via des ressources de cloud.

Au passé les solutions étaient habituellement déployées sur un seul serveur (hébergement traditionnel). Par contre avec l'apparition de cloud, les applications et les sites Web deviennent hébergés sur un réseau de serveurs cloud virtuels et physiques connectés et donc plus d'évolutivité.

Il existe quatre modes d'hébergement de cloud dont chacun donne une solution avec des niveaux de contrôle, de flexibilité et de gestion différente des autres.

I.5.1 Cloud public

Selon RedHat,[2]« Les clouds publics sont généralement des environnements cloud créés à partir d'une infrastructure informatique qui n'appartient pas à l'utilisateur final. Les clouds publics étaient habituellement exécutés hors site, mais les fournisseurs de cloud public proposent désormais des services cloud dans les datacenters de leurs clients, ce qui rend les notions d'emplacement et de propriété obsolètes. Tous les clouds deviennent des clouds publics lorsque les environnements sont partitionnés et redistribués entre plusieurs clients. Les structures payantes ne sont plus nécessairement caractéristiques des clouds publics puisque certains fournisseurs de cloud (tels que le Massachusetts Open Cloud) autorisent les clients à utiliser leurs clouds gratuitement ».

Le fournisseur de cloud public possède et administre les datacenters où s'exécutent les traitements des clients. Les fournisseurs de services sont responsables de la maintenance de tout le matériel et de l'infrastructure, et fournissent des connexions réseau à haut débit pour

[2] RedHat, "Quels sont les différents types de cloud ?," 2018. https://www.redhat.com/fr/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud (accessed May 24, 2022)

garantir un accès rapide aux applications et aux données. Le fournisseur de cloud gère également le logiciel de virtualisation[3].

Alibaba Cloud, Microsoft Azure, Google Cloud, pCloud, VMware, RedHat, Amazon Web Services (AWS) et IBM Cloud sont les principaux fournisseurs de cloud public.

I.5.2 Cloud privé

Selon Amazon Web services «Le déploiement de ressources sur site à l'aide de la virtualisation et d'outils de gestion des applications est parfois appelé « cloud privé ». Le déploiement sur site ne présente pas les avantages qu'offre le cloud computing, mais il est parfois souhaité pour sa capacité à fournir des ressources dédiées. Dans la plupart des cas, ce modèle de déploiement est identique à l'infrastructure informatique héritée et fait appel à la gestion des applications et aux technologies de virtualisation pour tester et accroître l'utilisation des ressources».[4]

Une architecture de cloud privé regroupe les ressources des data centers dans un pool de ressources unique. En virtualisant les composants matériels, les entreprises peuvent accroître l'efficacité et l'utilisation de leur infrastructure de cloud privé. Les solutions de cloud privé proviennent de grands éditeurs de logiciels, tels que VMware, Microsoft, etc., tandis que les solutions open source de niveau entreprise proviennent de RedHat, OpenStack, etc . Un cloud privé peut s'étendre sur le réseau mondial pour inclure plusieurs emplacements de serveurs ou de l'espace loué dans des installations de colocation internationales. Les solutions de cloud privé fournissent des outils logiciels pour l'orchestration de réseaux complexes sur des serveurs baremetal afin de gérer la sécurité des données directement au sein de l'entreprise[9].

Un cloud privé est un type de cloud computing qui offre des avantages similaires aux clouds publics, notamment l'évolutivité et le libre-service, via une architecture propriétaire. Contrairement au cloud public, qui dessert plusieurs organisations, le cloud privé est spécialisé à une seule organisation ou une seule entreprise. Par conséquent, il s'agit d'une solution utile et préférable pour les entreprises qui satisfont leurs besoins.

---

[3] IBM, "What is public cloud," *SearchCloudComputing.com*, 2010. https://www.ibm.com/cloud/learn/public-cloud (accessed Jun. 21, 2022).

[4] Amazon Web Services, "Modèles et types de cloud computing | AWS," 2020. https://aws.amazon.com/fr/types-of-cloud-computing/ (accessed Jun. 03, 2022).

Les clouds privés peuvent être plus sécurisés que les clouds publics. Il peut offrir de nombreux avantages en matière de sécurité. Étant donné que les clouds privés sont limités à des machines physiques spécifiques, la sécurité physique peut être assurée plus facilement.

Voici l'illustration la figure 1 qui représente le changement de vision lors de la mise en place d'un cloud privé :

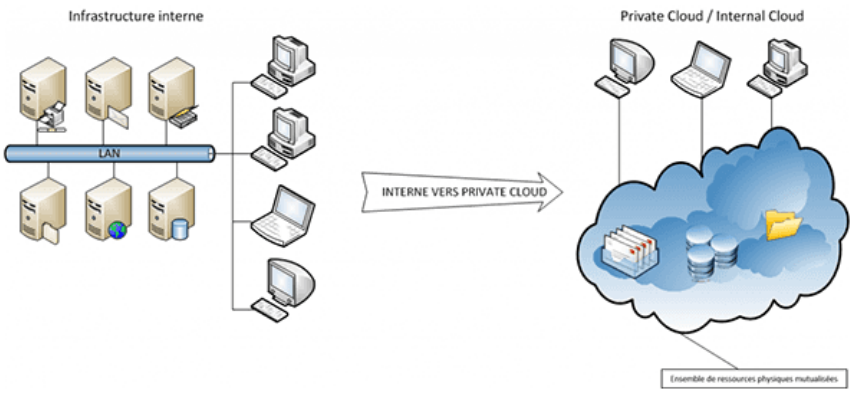

Figure 1 Cloud privé [10]

I.5.3 Cloud hybride

Selon VMware, « Un Cloud hybride est un modèle de Cloud Computing qui réunit au moins un Cloud privé et au moins un Cloud public, qui fonctionnent ensemble pour fournir une combinaison flexible de services de Cloud Computing. Le Cloud Computing hybride étend l'infrastructure et les opérations de manière cohérente pour fournir un modèle d'exploitation unique qui gère les charges de travail applicatives sur les deux environnements, ce qui permet une migration transparente des charges de travail du Cloud privé vers ou depuis le Cloud public en fonction des besoins de l'entreprise. Les solutions de Cloud hybride offrent un pool de ressources unique et transparent qui prend en charge les stratégies applicatives modernes et les efforts de transformation digitale d'une entreprise ».[5]

[5] VMware, "Qu'est-ce qu'un Cloud hybride _ _ Glossaire VMware _ FR," 2020. https://www.vmware.com/fr/topics/glossary/content/hybrid-cloud.html (accessed Apr. 21, 2022)

Selon Alibaba cloud [6]«Le cloud hybride est un modèle de Cloud Computing qui combine un ou plusieurs environnements de cloud public et privé via une connexion réseau, ce qui permet de partager des données et des applications entre les différents environnements de cloud. Un environnement de cloud hybride permet aux entreprises de répartir les charges de travail sur différents environnements de cloud en fonction de leurs besoins. Par exemple, une entreprise peut exécuter des services de base dans un environnement de cloud privé pour mieux contrôler et personnaliser l'environnement en fonction de ses besoins. Lorsque la charge de travail dépasse les limites des ressources disponibles, le travail supplémentaire peut être transféré automatiquement vers l'environnement de cloud public. Cette approche offre une capacité supplémentaire aux services à la demande, méthode connue sous le nom de "cloud bursting". Étant donné que les exigences supplémentaires seront gérées sur le cloud public, les capacités de stockage, de calcul et autres ne sont pratiquement pas limitées »

Le cloud hybride nécessite une connexion entre au moins un cloud privé et un cloud public. Pour cela il y a deux méthodes pour établir cette connexion : VPN (Virtual private network) pour l'échange des données sécurisées et express connect (connexion dédiée de point à point). L'architecture cloud hybride est la réponse pour les organisations à la recherche de sécurité, d'évolutivité et d'une approche économique du cloud computing. La clé pour y parvenir est d'assurer une intégration parfaite entre les différents composants sur le cloud privé et public. Voici une illustration (Fig.2) qui représente une connexion entre un cloud public et un cloud privé pour former un cloud hybride

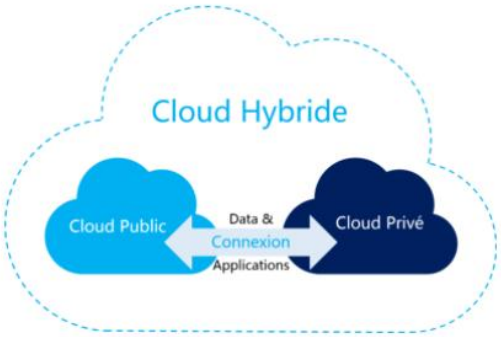

Figure 2: Cloud hybride [11]

[6] Alibabacloud, "Différents Types de Cloud_ Cloud Public, Cloud Privé et Cloud Hybride - Alibaba Cloud," 2020. https://www.alibabacloud.com/fr/knowledge/public-cloud-private-cloud-hybrid-cloud (accessed May 11, 2022).

I.5.4 Cloud Communautaire

Un cloud communautaire en cloud computing est un effort collaboratif dans lequel l'infrastructure est partagée entre plusieurs organisations d'une communauté spécifique ayant des préoccupations communes [12]. Quelques organisations construisent et proposent ensemble une infrastructure cloud similaire, ainsi que des politiques, des exigences, des valeurs et des préoccupations. Le cloud communautaire forme un niveau d'évolutivité économique et d'équilibre démocratique. L'infrastructure cloud peut être facilitée par un fournisseur tiers ou à l'intérieur de l'une des organisations de la communauté. Le cloud est supervisé par quelques organisations et soutient une communauté particulière qui a une intrigue similaire. le cloud communautaire est plus sécurisé que le cloud public [13]. Ce modèle de déploiement prend en charge plusieurs organisations partageant des ressources informatiques faisant partie d'une communauté. Cette situation peut être trouver dans des institutions académiques qui travaillent dans certains domaines de recherche communs, ou des services de police au sein d'un secteur ou d'un État partageant des ressources informatiques [14].

I.6 Types des services de Cloud Computing

Les types de cloud computing sont des modèles de déploiement de services qui permet de choisir le degré de contrôle souhaité sur les données et les types de services offrir.

I.6.1 Platform as a Service (PaaS)

Une PaaS (Platform as a Service) permet d'écrire des applications qui s'exécutent sur le cloud. Il s'agit d'un développement d'applications basées sur le cloud et utilisées par les déployeurs et les développeurs et les DevOps en général. Il possède une architecture multiniveau hautement évolutive, par ex. Azure et salesforces.com. La différence entre PaaS et SaaS est que le SaaS n'héberge que l'application cloud terminée où PaaS offre une plate-forme de développement pour les applications cloud terminées et en cours. PaaS offre un environnement dans lequel les développeurs peuvent créer et réussir des applications et n'ont pas nécessairement besoin de connaître la quantité de mémoire et le nombre de processeurs que leur application utilisera. Le modèle PaaS offre des avantages au développeur en termes de développement du cycle de vie du logiciel complémentaire, de la planification à la conception en passant par la création d'applications, le déploiement et la maintenance [16].

L'architecture PaaS, ou Platform-as-a-Service, s'éloigne de la gestion de l'infrastructure sur site. Le fournisseur héberge le matériel et les logiciels sur sa propre infrastructure et met une

plate-forme, sous la forme d'une solution intégrée, d'une pile de solutions ou d'un service, à la disposition de l'utilisateur via Internet. Le PaaS s'adresse principalement aux développeurs et aux programmeurs, permet aux utilisateurs de créer, d'exécuter et de gérer leurs propres applications sans avoir à créer ou à maintenir l'infrastructure ou la plate-forme généralement associée au processus. Voici une illustration montre un PaaS (Figure 3).

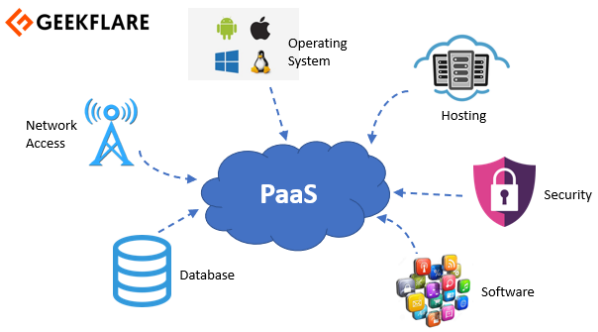

Figure 3 Platform as a Service (PaaS) [11]

I.6.2 Infrastructure as a Service (IaaS)

Selon Microsoft Azure « Infrastructure as a service (IaaS) est un type de service de cloud computing qui offre des ressources de calcul, de stockage et de mise en réseau essentielles à la demande, et sur une base de paiement à l'utilisation. IaaS est l'un des quatre types de services cloud, parallèlement à Software as a Service (SaaS), platform as a service (PaaS) et server less. La migration de l'infrastructure d'une organisation vers une solution IaaS permet de réduire la maintenance des centres de données locaux. Les solutions IaaS offrent la flexibilité nécessaire à la mise à l'échelle (vers le haut ou le bas) des ressources informatiques par rapport à la demande. Elles permettent également d'approvisionner rapidement de nouvelles applications et d'augmenter la fiabilité de l'infrastructure sous-jacente.»[7]

I.6.3 Software as a Service (SaaS)

SaaS ou Software as a Service est un modèle de distribution de logiciels dans lequel les applications sont hébergées par un fournisseur de services et mises à la disposition des clients sur un réseau, généralement Internet. Le SaaS devient un modèle de livraison de plus en plus répandu à mesure que les technologies sous-jacentes qui prennent en charge les services Web

[7] MicrosoftAzure, "Infrastructure as a Service | Microsoft Azure," 2022. https://azure.microsoft.com/en-us/overview/what-is-iaas/ (accessed Jun. 22, 2022).

et l'architecture orientée services (SOA) arrivent à maturité et que de nouvelles approches de développement deviennent populaires. Le SaaS est également souvent associé à un modèle de licence d'abonnement à la carte. Les applications SaaS doivent également pouvoir interagir avec d'autres données et d'autres applications dans une variété tout aussi large d'environnements et de plates-formes[17]. Cependant, malgré les avantages en termes de coûts qui sont attractifs pour les consommateurs SaaS, le déploiement de SaaS suscite toujours des inquiétudes. Ces préoccupations sont le respect des exigences techniques, la sécurité, la facilité d'intégration et la fonctionnalité [18].

Le coût d'utilisation de la solution est généralement calculé comme une redevance mensuelle basée sur le nombre d'utilisateurs. Ce dernier est forfaitaire et comprend l'ensemble des services souscrits par le client.

I.6.4 Function as a Service (FaaS)

FaaS ou (Function as a Service) sous-type de serverless fournit les moyens de s'éloigner des serveurs sur lesquels les logiciels développés sont censés être exécutés. Il offre essentiellement un environnement événementiel et évolutif dans lequel la facturation est basée sur l'invocation de fonctions et non sur le provisionnement de ressources [19]. La FaaS et le server less sont souvent utilisées conjointement, mais il y a une différence. Server less en général est une expression plus abstraite, alors que FaaS se concentre sur les fonctions cloud pilotées par les événements [20].

FaaS (Function-as-a-Service) est un service de cloud computing qui permet aux développeurs de créer, calculer, exécuter et gérer des packages d'applications en tant que fonctions sans maintenir leur propre infrastructure. FaaS est un modèle d'exécution piloté par les événements qui s'exécute dans des conteneurs sans état. Les fonctions en tant que service gèrent l'état et la logique côté serveur via un service fourni par le fournisseur. Les solutions FaaS sont disponibles sur les principaux clouds publics et peuvent être configurées sur site. En tant que tels, ils apportent de nouvelles fonctionnalités intéressantes au développement d'applications d'entreprise[8]. Voici quelques exemples de FaaS fréquemment utilisés: IBM Cloud Functions, AWS Lambda d'Amazon, Google Cloud Functions, Microsoft Azure Functions (Open Source) .

[8] RedHat, “Le FaaS, qu’est-ce que c’est,” 2019. https://www.redhat.com/fr/topics/cloud-native-apps/what-is-faas (accessed Jun. 10, 2022).

I.7 Principaux avantages du cloud computing

Le cloud computing est radicalement différent de l'approche traditionnelle que les entreprises adoptent en matière de ressources informatiques[9]. Voici sept raisons courantes pour lesquelles les entreprises optent pour les services de cloud computing [10]:

I.7.1 Réduction des coûts

L'utilisation du cloud computing contribue souvent à réduire les coûts. Le cloud computing élimine la nécessité d'investir dans du matériel et des logiciels, et de configurer et de gérer des datacenters sur site : racks de serveurs, alimentation électrique permanente pour l'alimentation et le refroidissement, experts informatiques pour la gestion de l'infrastructure. Grâce au cloud, les entreprises paient par abonnement uniquement pour les ressources, infrastructures et services nécessaires. Cela veut dire (payer que ce qu'on utilise).

I.7.2 Vitesse et agilité

La majorité des services de cloud computing sont en libre-service et à la demande. D'énormes ressources de calcul peuvent ainsi être déployées en quelques minutes et en quelques clics, offrant aux entreprises une grande flexibilité et allégeant les contraintes de planification des capacités.

I.7.3 Mise à l'échelle mondiale

L'un des avantages des services de cloud computing est la possibilité d'évoluer selon les besoins. Dans la terminologie du cloud, cela veut dire qu'il est possible de mettre en œuvre la quantité nécessaire de ressources informatiques, par exemple plus ou moins de puissance de calcul, de stockage ou de band-width, peut être mise en œuvre quand et où elles sont nécessaires.

I.7.4 Flexibilité

Le cloud est très flexible car le payement est selon l'utilisation. Il permet aussi de minimiser le délai d'obtention de nouvelles ressources, déployer rapidement de nouvelles applications et

[9] IBM, "Avantages du cloud computing - France _ IBM." https://www.ibm.com/fr-fr/cloud/learn/benefits-of-cloud-computing (accessed Jun. 21, 2022).

[10] Microsoft Azure, "Qu'est-ce que le cloud computing ?Microsoft Azure," 2021. https://azure.microsoft.com/fr-fr/overview/what-is-cloud-computing/ (accessed May 08, 2022).

ajouter de nouveaux utilisateurs en un seul clic. Cela signifie également une flexibilité financière, car le bon fournisseur peut adapter le plan en fonction des besoins.

I.7.5 Sécurité des données

Les pannes matérielles n'entraînent pas de perte de données grâce aux sauvegardes réseau. De nombreux fournisseurs de cloud proposent une variété de politiques, de technologies et de contrôles pour aider les développeurs à améliorer l'ensemble de leurs posture de sécurité et à protéger leurs données, leurs applications et leur infrastructure contre les menaces.

I.7.6 Performances

Les services de cloud computing les plus populaires sont hébergés sur un réseau sécurisé de Datacenter dont le matériel est régulièrement mis à niveau pour assurer des performances rapides et efficaces. Cela présente un certain nombre d'avantages par rapport à un datacenter traditionnel, notamment une latence du réseau d'application réduit et de plus grandes économies d'échelle.

I.8 Problèmes de cloud computing

En plus des avantages, il existe également un certain nombre d'inconvénients et de risques associés à l'utilisation du stockage en clouds. Certains de ces inconvénients sont présentés ci-dessous [22] :

• Les fuites et l'accès aux données sans autorisation entre les appareils virtuels fonctionnant sur le même serveur.

• Erreurs de la part d'un fournisseur de cloud dans la gestion et la sauvegarde correctes des données sensibles.

• La nécessité d’une connexion Internet constante

• Parfois, le service cloud peut être indisponible pendant de longues périodes en raison d'erreurs et de pannes du système.

• Les pirates peuvent s'introduire par effraction dans les applications d'un client hébergées sur le cloud, et ainsi accéder et distribuer des données sensibles.

• Problèmes de fiabilité.

• Incapacité possible à maintenir l'intégrité des données.

• Les données stockées peuvent ne pas être sécurisées.

• Configurations limitées : les fournisseurs de solutions de cloud public disposent d'un ensemble standard de configurations de base qui répondent aux besoins de l'ensemble de la population. Dans certains cas, un matériel exceptionnellement spécialisé est nécessaire pour faire face à des problèmes de calcul intensifiés. Dans de tels cas, l'utilisation des solutions de cloud public est souvent impossible, au motif que la fonctionnalité requise est simple et non proposée par le fournisseur.

I.9 Conclusion

Dans ce chapitre, nous avons donné un aperçu sur le cloud computing commençant de l'historique, ensuite on a présenté plusieurs définitions de cloud computing prises des sites officiels des fournisseurs de cloud les plus réputés dans le monde.

Au cœur de ce chapitre nous avons fourni quelques détails concernant les types d'hébergement appelés aussi les modèles de déploiement de cloud. Ensuite on a expliqué les différents types de service du cloud. A la fin de ce chapitre nous avons cité quelques avantages et quelques problèmes liés à l'utilisation de Cloud Computing.

Dans le prochain chapitre, nous allons parler en détails sur quelques architectures logicielles, Monolithe, SOA et l'architecture microservice.

Chapitre II LES ARCHITECTURES LOGICIELLES

II.1 Introduction

La transformation numérique exige que les organisations soient agiles et adoptent des méthodes rapides et innovantes, permettant de fournir de nouveaux services numériques aux clients [23]. A cet effet, les organisations cherchent à développer des applications Cloud flexibles, qui facilitent la mise en œuvre des services numériques. Afin d'assurer ceci, les organisations ont besoin d'implémenter un certain type d'architecture logicielle. En effet, il existe trois types d'architecture à savoir :

L'architecture monolithique est une approche qui est apparue vers les années 1990, basée sur la construction d'une application en un seul bloque. Avec le temps les organisations qui optent pour ce type d'architecture traditionnelle ont conclu qu'elle ne répond plus aux besoins d'évolutivité et de rapidité du cycle de développement et reste un obstacle à la transformation numérique. Cela a conduit à l'émergence d'un autre type d'architecture vers les années 2005, appelé l'architecture orienté service (SOA) qui est basée sur la décomposition de l'application en quelque services communs réutilisables. Ce type d'architecture SOA n'a pas apporté le succès escompté et son utilisation est souvent limitée aux grands projets. Actuellement, dans les grandes organisations le recours à d'autres concepts de services s'avère très nécessaire pour remédier aux limitations des deux approches logicielles citées précédemment. A cet effet, l'architecture microservices s'est apparue, qui est une amélioration du SOA et qui est plus adaptée à la création de solutions Cloud agiles.

Dans le présent chapitre, nous nous intéressons aux concepts architecturaux de base utilisés dans le domaine informatique. Les différentes approches (monolithe, SOA, micro services) sont présentées et leurs avantages et limitations ainsi qu'une étude détaillée sur les concepts des microservices et leurs relations avec cloud computing.

II.2 Architecture monolithique

II.2.1 Définition

L'architecture monolithique représente un modèle traditionnel unifié pour la conception de programmes informatiques. Il s'agit d'une application développée en un seul bloc avec la même technologie, avec toutes les fonctionnalités encapsulées dans une seule application et déployées sur un serveur d'application[24]. Une application monolithique est souvent

associée à une base de données et à une interface utilisateur côté client comme l'illustre la figure 4. Ses composants sont interconnectés et interdépendants plutôt qu'associés de manière flexible comme dans le cas des programmes modulaires. Ce type d'architecture est étroitement couplé, et toute la logique de traitement d'une demande s'exécute dans un seul processus [25].

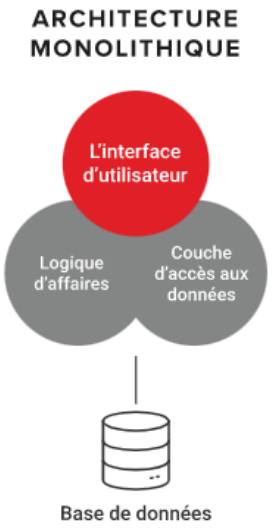

Figure 4: Architecture monolithique [26]

II.2.2 Caractéristiques de l'architecture monolithique

L'architecture monolithique comporte en effet plusieurs caractéristiques :

- ✓ Simplicité : les applications monolithiques sont faciles à créer, à tester et à déployer. Lorsqu'une applications monolithiques est conçue avec une seule base de code, elle devient plus facile à développer et gérer les problèmes de code et la configuration ainsi que , surveillance des performances.

- ✓ Test : Il est beaucoup plus facile de tester une structure qui est un monolithe. Des composants tels que des Frameworks, des modèles ou des scripts peuvent être facilement appliqués.

- ✓ Pile technologique: Une application monolithique développer avec une seule technologie [27].

II.2.3 Limites de l'architecture monolithique

L'architecture monolithique permet de développer des applications entières en un seul bloc. Malgré qu'elle existe depuis plusieurs décennies, un nombre important d'entreprises reste à l'utiliser avec succès à nos jours, vu sa fonctionnalité. Cependant, avec la rapidité de l'évolution du marché et de l'apparition des nouvelles technologies l'application monolithique présente certaines limitations qui peuvent être énumérés comme suit :

- ✓ Dès que le nombre de fonctions augmente, un monolithe devient plus complexe a gérer et nécessite une équipe de développeurs plus importante pour prendre en charge l'application [28],
- ✓ Difficile de travailler en équipe dans la même base de code [29],
- ✓ Les fichiers de déploiements sont volumineux et lents au démarrage d'application,
- ✓ L'évolutivité horizontal ajoute un cout et complexité a un monolithe, ce qui le rend peu pratique pour le déploiement de Cloud,
- ✓ Les développeurs sont verrouillés dans la pile de technologie, c'est-à-dire le choix des technologies est fixé avant que le développement de l'application commence[11],
- ✓ Une modification dans une petite partie de l'application nécessite un retest et redéploiement entier de tout l'application [23],
- ✓ La tolérance aux pannes est limitée, une panne sur une partie de l'application a un impact sur toute l'application,

II.3 Architecture orientée services SOA

II.3.1 Définition

De nombreux chercheurs ont défini la SOA en fonction de différentes perspectives (telles que la technologie, l'entreprise, et l'architecture). Il n'y a pas de définition exacte de ce terme. Selon [30] « L'architecture orientée services (SOA) est un style de développement et d'intégration de logiciels. Elle implique de décomposer une application en services communs et reproductibles pouvant être utilisés par d'autres applications, tant internes qu'externes,

[11] IBM Cloud Education, "What are Microservices?," 2021. https://www.ibm.com/cloud/learn/education (accessed Jun. 01, 2022).

d'une organisation, indépendamment des applications et des plateformes informatiques sur lesquelles l'entreprise et ses partenaires s'appuient ».

Selon [31] OasisGroup's définies SOA comme « L'architecture orientée services (SOA) est un style architectural qui prend en charge l'orientation services. L'orientation vers les services est une façon de penser en termes de services et de développement basé sur les services et les résultats des services ».

Selon Redhat[12] « L'architecture orientée services (ou SOA, Service-Oriented Architecture) est un modèle de conception qui rend des composants logiciels réutilisables, grâce à des interfaces de services qui utilisent un langage commun pour communiquer via un réseau. »

II.3.2 Définition d'un service

Un service est une unité de programme qui peut être appelée à l'aide de procédures normalisées et peut exécuter les fonctions qui lui sont assignées de manière indépendante. Chaque service s'exécute dans un environnement hétérogène avec différents matériels, systèmes d'exploitation, langages de programmation, etc. Par conséquent, les services peuvent être facilement ajoutés, échangés ou réutilisés [32].

II.3.3 Principes fondamentaux du SOA

L'Architecture SOA repose sur trois acteurs principaux (Fig. 5) qui peuvent être résumés comme suit :

a) Le fournisseur de service (service provider) :

- ✓ Celui qui définit le service : création, implémentation, description, contrat d'utilisation,
- ✓ Publier sa description dans annuaire UDDI.

b) Registre de services (Service registry)

- ✓ Représente où sont enregistrer les descriptions des services publiés par les fournisseur,
- ✓ Recevoir et répondre aux recherches de services lances par les clients.

c) Consommateur de service :

- ✓ Le consommateur de service est l'acteur qui utilise les services en fonction de ses besoins,
- ✓ Permet de lancer des recherches sur les services appropriés,

[12] RedHat, "L'architecture orientée services (SOA), qu'est-ce que c'est ?," 2020 https://www.redhat.com/fr/topics/cloud-native-apps/what-is-service-oriented-architecture

✓ Obtenir la description du service grâce à l'annuaire.

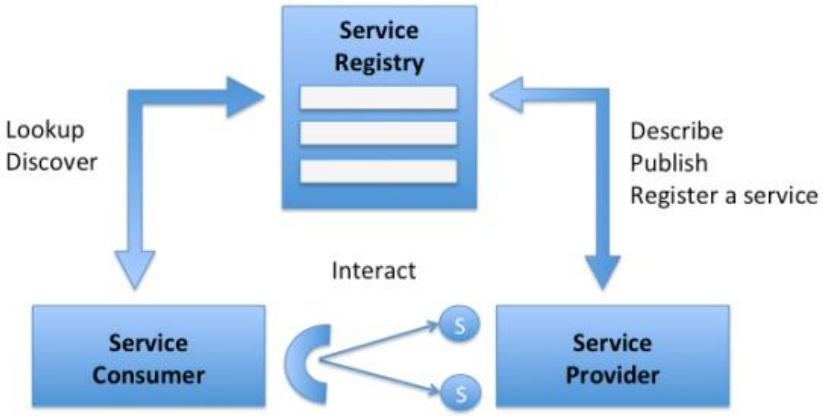

Figure 5 :Principes fondamentaux du SOA [33]

II.3.4 Avantages de SOA par rapport à une approche monolithique:

SOA comporte en effet plusieurs avantages dont :

- ✓ Amélioration de la collaboration entre l'entreprise et les services informatiques [13],
- ✓ Offre une maintenance simple.
- ✓ Pas de logiciel sur 1e poste client (client léger).
- ✓ La réutilisation de code.
- ✓ Offre une plus grande tolérance aux pannes[14].
- ✓ Coûts réduits grâce à une meilleure agilité et à un développement plus efficace.

II.3.5 Type de services SOA

La SOA fournit quatre types de services différents :

- ✓ Les services fonctionnels (c'est-à-dire les services d'entreprise), qui sont essentiels pour les applications d'entreprise.
- ✓ Les services d'entreprise, qui servent à mettre en œuvre les fonctionnalités.

[13] IBM Cloud Education, "Qu'est -ce que l'architecture orientée services (SOA, Service-Oriented Architecture) ?," 2019. (accessed Jun. 21,2022)

[14] RedHat, "L'architecture orientée services (SOA), qu'est-ce que c'est ?," 2020 https://www.redhat.com/fr/topics/cloud-native-apps/what-is-service-oriented-architecture (accessed Jun. 21,2022)

- ✓ Les services d'application, qui sont utilisés pour développer et déployer des applications.
- ✓ Les services d'infrastructure, qui jouent un rôle essentiel dans les processus de backend tels que la sécurité et l'authentification.

II.3.6 Limitations du SOA

SOA comporte plusieurs inconvénients qui peuvent être présenter comme suit[15]:

- ✓ Coûts élevés en termes de ressources humaines, de développement et de technologie.
- ✓ Recentralisation des systèmes (charge très importante sur le serveur).
- ✓ L'architecture SOA ne bénéfice pas clairement des services Cloud.
- ✓ N'adapte pas à la technologie de conteneurisation.
- ✓ L'architecture orientée services ne convient pas aux applications qui sont basées sur un l'échange de données[16].

II.4 Architecture Microservices

Le terme « microservice » a été inventé pour la première fois en 2005 lors d'une conférence sur le cloud computing par le Dr. Peter Rodgers [34], et au printemps 2011, le terme « microservices » est apparu lors d'une conférence d'architectes logiciels. Ils deviennent de plus en plus populaires car ils nous aident à faire face à de nombreux changements dans le monde informatique d'aujourd'hui, comme pas nouveau. Google, Facebook et Amazon ont utilisé cette approche à des degrés divers pendant plus d'une décennie. Par exemple, une simple recherche sur Google utilise plus de 70 microservices avant l'affichage de la page de résultats. De plus, d'autres architectures ont été développées pour résoudre certains des problèmes que les microservices peuvent résoudre.

[15] IBM, "SOA vs. Microservices: What's the Difference?," https://www.ibm.com/cloud/blog/soa-vs-microservices 2021. (accessed Jun. 21,2022)

[16] RedHat, "L'architecture orientée services (SOA), qu'est-ce que c'est ?," 2020 https://www.redhat.com/fr/topics/cloud-native-apps/what-is-service-oriented-architecture (accessed Jun. 01, 2022)

Définition

L'architecture de microservices (MSA) est un style architectural natif de Cloud "Cloud native" , inspiré de l'architecture orientée services (SOA) [30], Basé sur découpage d'un grand problème en petites unités implémentées sous forme des microservices (Figure 6), dans laquelle une application unique est composée de nombreux petits services faiblement couplés[17], ou chaque microservice est responsable d'une fonctionnalité et il est développé, testé et déployé indépendamment des autres. Ainsi, qu'ils pouvant être développé par le biais de multiples piles technologiques, chaque service MSA s'exécute sur son propre processus et communique entre eux sur le réseau via les API qu'ils exposent.

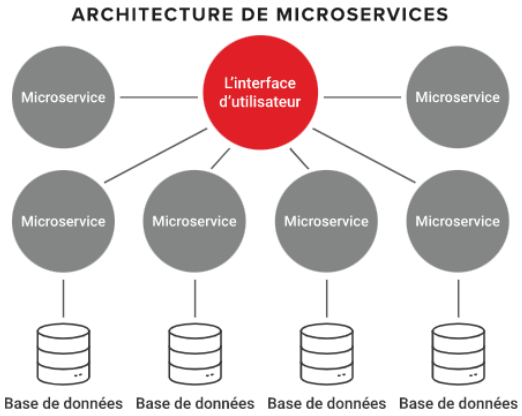

Figure 6 : Architecture microservices [26]

II.4.1 Caractéristiques de l'architecture microservices

L'architecture microservice a plusieurs caractéristiques parmi eux[18] :

Autonomie :

- ✓ Tous les services composants peuvent être développés, déployés, gérés et mis à l'échelle indépendamment.
- ✓ Les microservices n'ont pas besoin de partager leur code ou leur implémentation avec d'autres services.

[17] IBM Cloud Education, "What are Microservices?," 2021. https://www.ibm.com/cloud/learn/education (accessed Jun. 01, 2022).

[18] A. AMAZON, "Que sont les microservices ? https://aws.amazon.com/fr/microservices/," 2020. (Accéder 2022-06-19)

Agilité :

- ✓ Les microservices facilitent l'organisation de petites équipes indépendantes responsables des services.
- ✓ Les équipes peuvent travailler dans un petit contexte facile à comprendre d'une manière plus indépendante et plus rapide. Cela réduit la durée du cycle de développement [37].

Pile Technologique :

- ✓ Dans une architecture de microservices, chaque service peut utiliser la technologie, le langage et la plate-forme les plus appropriés à ses besoins.

L'évolution :

- ✓ L'évolution de l'application consiste à ajouter de nouvelles fonctionnalités qui se traduit par création de nouveaux microservices et/ou mise à jour de services existants qui implique seulement la mise à jour et le redéploiement des microservices concernés.

Déploiement :

- ✓ Les microservices fournissent une intégration et une livraison continues ,ce qui facilite la mise en œuvre de nouvelles idées et rafraichir des modifications en cas de problème. La réduction des coûts d'échec accélère les tests, facilite les mises à jour du code et accélère la mise sur le marché des nouvelles fonctionnalités.

Test :

- ✓ Les microservices sont petits et peuvent être testé rapidement.

Tolérance aux pannes :

- ✓ Les applications en microservices, gèrent complètement les pannes de service en dégradant les fonctionnalités, mais sans interrompre l'ensemble de l'application.
- ✓ Le couplage faible des microservices permet également d'isoler les défauts et d'améliorer la résilience des applications [37] .

II.4.2 Défis de l'architecture microserivces

Les avantages liés à l'utilisation de l'architecture micro-services sont très nombreux. Cependant la nature distribuée du MSA est l'un des principaux défis qui produit une nouvelle forme de complexité dans le développement et le débogage des systèmes basés sur ce style architectural ,En effet [35] [36] :

- ✓ Temps de réponse élevé généré par la communication entre les différents microservices qui sont situés dans des zones différentes de monde, et ici notre problème est concentré.
- ✓ Le besoin des efforts de conception et d'analyse fonctionnelle pour créer des unités fonctionnelles autonomes et hautement cohérentes
- ✓ Difficulté à localiser le problème dans un enivrement distribué
- ✓ La mise en réseau entre les dockers conteneurs .
- ✓ La gestion opérationnelle des microservices est plus complexe, en particulier du point de vue du déploiement, des tests et de la surveillance.
- ✓ Le partage des données de base entre différents services compte tenu des limitations de performances
- ✓ le débogage d'un microservice qui s'appuie sur d'autres services peut être délicat
- ✓ Le besoin de nouvelles compétences pour les équipes de développement et les équipes opérationnelles
- ✓ Il faut que l'infrastructure soit compatible avec les microservices pour prendre en charge les nouveaux workflows de livraison continue.
- ✓ La latence sera nécessairement introduite par la nécessité de traverser le réseau afin d'exécuter un workflow complet.

II.4.3 Migration de monolithe vers l'architecture microservices :

L'architecture des microservices est devenue extrêmement populaire car les architectures monolithiques traditionnelles ne répondent plus aux besoins d'évolutivité et de cycle de développement rapide, et le succès de certaines grandes entreprises dans la création et le déploiement de services est une forte motivation pour que d'autres envisagent de faire le changement. [38]

Dans cette section en parlant sur les techniques de migration d'une application monolithe vers une application basée sur l'architecture microservices.

On commence par rappelons rapidement les définitions de monolithe et microservices puis on rentre dans les détails de migration.

II.4.3.1 Comparaison entre les architectures monolithe et microservice

Le tableau suivant représente une comparaison entre l'architecture monolithe et microservice :

Option	**Monolithe**	**Microservice**
Technologie	-Verrouillé à la pile et au cadre de technologie d'origine. -Toutes les fonctions développées avec un seul langage de programmation.	-Chaque microservice peut être développé à l'aide d'une technologie différente, adaptée à l'objectif ou basée sur les préférences du développeur.
Evolutivité	-Les fonctions d'un monolithe ne peuvent pas être mises à l'échelle indépendamment. -Une mise à l'échelle horizontale de l'ensemble du monolithe est nécessaire.	-Chaque microservice peut être mis à l'échelle indépendamment via des conteneurs. - Des outils sont nécessaires pour le suivi et la gestion des conteneurs
Gestion du changement	- Pour tout petit changement, la totalité -le monolithe doit être retesté. - Les processus de changement/test sont complexe et chronophage.	-Les microservices sont petits et peuvent être testé rapidement. -Les microservices ont des cadences de relâchement.

Déploiement	- Le fichier de déploiement est volumineux, lent à démarrage, peut entraîner des temps d'arrêt. - Utilisation de méthodes DevOps agiles et les outils n'est pas pratique.	- Les microservices peuvent être déployés indépendamment. - Utilisation de méthodes DevOps agiles et les outils sont appropriés

Tableau 2 : Comparaison des fonctionnalités entre les monolithes et les microservices [23]

II.4.3.2 Transition des applications monolithes vers les application microservices basés sur le cloud

Au cours des dernières années, de nombreux grands systèmes sont passés d'applications monolithiques autonomes construites à partir de composants interconnectés et interdépendants à des ensembles de grands services (c'est-à-dire des SOA traditionnels), puis à des ensembles de petits services autonomes et légers connectés. (voir Fig. 7). Le rythme élevé de la demande du marché pour de nouvelles fonctionnalités d'application nécessite des changements à la fois dans les applications elles-mêmes (couplage lâche et grande évolutivité) et dans la façon dont elles sont construites (dépendances d'équipe lâches et déploiement rapide) [24]. Les microservices répondent à ces deux préoccupations puisque de petits services peuvent être créés et déployés par des équipes de développement indépendantes ; la liberté concomitante permet aux équipes de se concentrer sur l'amélioration de chaque service et d'augmenter la valeur commerciale. Par conséquent, dans la pratique, DevOps (développement et opérations) et la livraison continue conviennent parfaitement aux architectures de microservices [39].

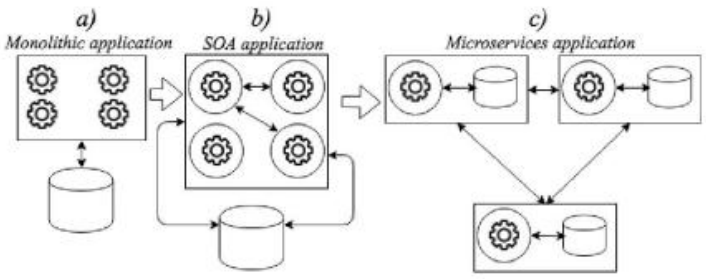

Figure 7: Monolithiques vers microservices [24]

II.4.3.3 Les phases de migration

Dans cette partie, nous présentons une approche proposée par des chercheurs pour migrer d'un monolithe vers une architecture de microservices basée sur le Cloud [23] , comme illustré à la figure 8 ci-dessous et détaillé dans la section qui suit. Les phases de migration présentées ici sont basées sur une migration réelle du système bancaire central menée dans un cadre universitaire dans le cadre d'un projet appelé SMU tBank [40], dans lequel un système bancaire de détail Oracle Flexcube a été directement remplacé par plus de 200 microservices.

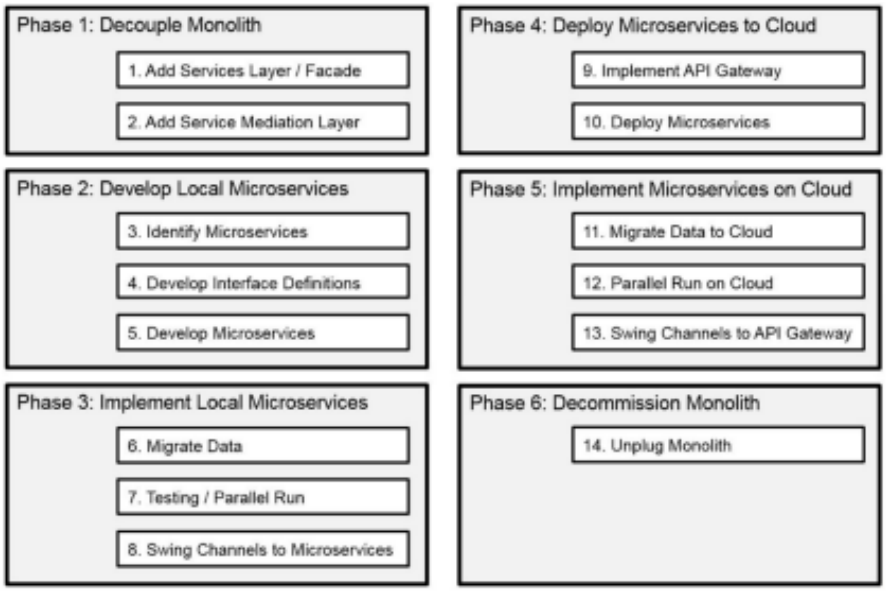

Figure 8: Les phases de migration [23]

Les phases de migration sont les suivants :

- ✓ Phase 1 : découpler le monolithe.
- ✓ Phase 2 : Développer des microservices locaux.
- ✓ Phase 3 : implémenter des microservices locaux.
- ✓ Phase 4 : Déployer des microservices dans le cloud.
- ✓ Phase 5 : implémenter des microservices sur le cloud.
- ✓ Phase 6 : déclasser le monolithe .

Les détails de chaque phase est bien expliqué par [23]

II.4.4 DevOps

II.4.4.1 Définition [19]

DevOps est une combinaison de philosophies, de pratiques et d'outils culturels qui améliorent la capacité d'une organisation à fournir des applications et des services à grande vitesse et une meilleure communication et collaboration entre les équipes, Cela permet de d'évoluer et d'optimiser les produits plus rapidement que les organisation qui utilisent les processus traditionnels de développement de logiciels et de gestion des infrastructures[20]. Cette rapidité permet aux organisations de mieux servir leurs clients et de devenir plus compétitives et d'atteindre plus rapidement les objectifs commerciaux (Microsoft).

Le modèle DevOps fait le lien entre les applications existantes et les nouvelles applications et infrastructures cloud-native . La figure 9 exprime les phases de DevOps :

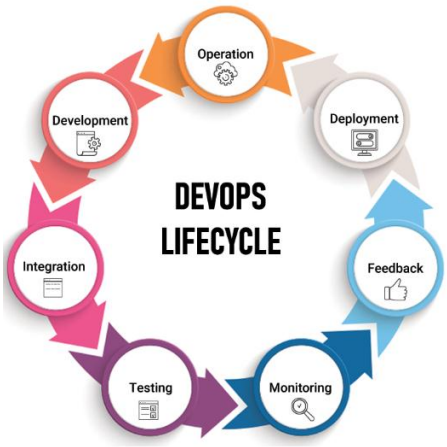

Figure 9: les phases DevOps [41]

II.4.4.2 Fonctionnement de DevOps

✓ Dans le modèle DevOps, les équipes de développement et d'exploitation ne sont plus isolées. Ils peuvent être fusionnés en une seule équipe. Les ingénieurs de l'équipe travaillent tout au long du cycle de vie de l'application [42], de la conception à la mise en service, en passant par les tests et le déploiement, et développent une gamme de compétences dans différentes fonctions.

---

[19] Azure microsoft, "What is DevOps?," 2020. https://azure.microsoft.com/en-us/products/devops/ . (accessed Jun. 21, 2022).

[20] RedHat, "L'automatisation du DevOps, qu'est-ce que c'est ?," 2020. https://www.redhat.com/fr/topics/automation/what-is-devops-automation (accessed Apr. 19, 2022)

- ✓ Ces équipes utilisent des méthodes pratiques pour automatiser des processus autrefois manuels et lents. Ils tirent parti des systèmes et des outils technologiques qui les aident à exécuter et à faire évoluer les applications rapidement et de manière fiable [43].
- ✓ DevOps est l'extension de la méthode agile de développement logiciel [44]. Il se concentre sur la livraison continue du logiciel et l'intégration continue. Ainsi que Automatisation qui joue également un rôle essentiel dans la réduction du temps de latence des produits [42].
- ✓ DevOps améliore non seulement la collaboration et la communication mais aussi la livraison rapide et continue, les mises à jour régulières, l'augmentation de la fiabilité, etc [45].

II.4.4.3 Relation des microservices avec DevOps

Les microservices et l'approche DevOps sont deux des technologies les plus couramment adoptées qu'il offrent une grande agilité et une efficacité opérationnelle pour l'entreprise [21] .Architecture de microservices conçue pour DevOps avec une approche basée sur les services qui permet aux organisations de décomposer les applications en services plus petits. Cela permet aux équipes distribuées de traiter les services individuels comme des entités distinctes, ce qui simplifie le développement, les tests et le déploiement. La combinaison des microservices et du DevOps a d'abord été utilisée par de grandes entreprises comme Amazon, Netflix, Google...etc. Le rôle des microservices dans DevOps comprend la simplification du processus DevOps et l'augmentation de la productivité et de la qualité des applications tout en déplaçant le développement vers une architecture agile [46]. Cela conduit au développement d'applications cloud natives capables de répondre à tous les besoins des utilisateurs.

II.4.4.4 Méthode Agile

Agile est une approche itérative de la gestion de projet et du développement logiciel qui encourage la collaboration, l'organisation et la rétroaction entre les clients et les équipes de développements et de créer de la valeur plus rapidement et plus facilement [47].Une équipe Agile livre son travail par petits incréments exploitables. Les exigences, les plans et les résultats sont évalués en permanence. Par conséquent, les équipes disposent d'un mécanisme naturel pour réagir rapidement au changement.

[21] S. Sharma and B. Coyne, *DevOps For Dummies®, 3rd IBM Limited Edition Published*. 2017.

La figure 10 exprime les phases de la méthode agile :

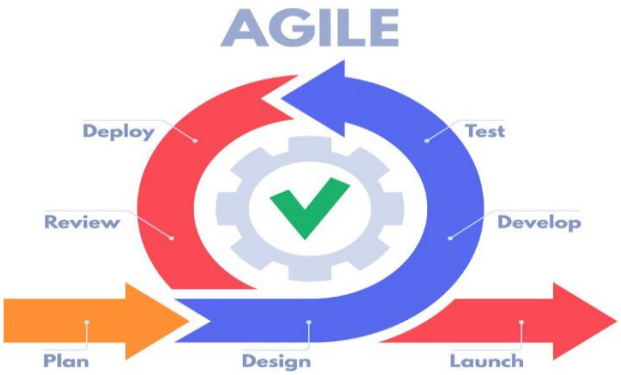

Figure 10: les phases de la méthode agile [48]

II.4.5 La virtualisation et la conteneurisation

Le cloud computing est un concept dynamique qui applique la virtualisation et des techniques connexes pour faciliter la fourniture de services aux utilisateurs. Pour prendre en charge la fourniture de ressources aux utilisateurs par les modèles de service et de déploiement, des technologies de base telles que la virtualisation, la conteneurisation et l'orchestration sont utilisées sur le cloud.

II.4.5.1 La virtualisation

Selon Redhat [22]« la virtualisation est une technologie qui permet de créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique. Son logiciel, appelé hyperviseur qui a une intéraction directe avec le matériel. L'hyperviseur permet de fragmenter le système unique en plusieurs environnements sécurisés distincts. C'est ce que l'on appelle les machines virtuelles. Ces dernières exploitent la capacité de l'hyperviseur à séparer les ressources du matériel et à les distribuer convenablement» .

La virtualisation repose sur des logiciels pour simuler les fonctionnalités matérielles et créer des systèmes informatiques virtuels. [23]Ce modèle permet aux services informatiques

[22] Kilien Sando, "Comprendre la virtualisation," Mar. 18, 2018. https://www.redhat.com/fr/topics/virtualization (accessed Jun. 23, 2022).

[23] VMware, "Que sont la technologie de virtualisation et les machines virtuelles ? | VMware | FR," 2020. https://www.vmware.com/fr/solutions/virtualization.html (accessed Jun. 23, 2022)

d'exécuter plusieurs systèmes virtuels (et plusieurs systèmes d'exploitation et applications) sur un seul serveur.

Les techniques de virtualisation sont largement utilisées dans de nombreux domaines informatiques importants, notamment le cloud computing, l'Internet des objets, la virtualisation des réseaux et le déploiement d'applications logicielles. Cette révolution s'appuie sur les avantages de l'isolation, de l'indépendance matérielle, de la sécurisation des environnements utilisateurs des systèmes d'exploitation, de l'évolutivité et surtout de l'optimisation massive des ressources physiques [51].

II.4.5.2 Conteneurisation

Selon Redhat [24] « la conteneurisation consiste à rassembler le code du logiciel et tous ses composants (bibliothèques, frameworks et autres dépendances) de manière à les isoler dans leur propre « conteneur ». Le logiciel ou l'application dans le conteneur peut ainsi être déplacé et exécuté de façon cohérente dans tous les environnements et sur toutes les infrastructures, indépendamment de leur système d'exploitation. Le conteneur fonctionne comme une sorte de bulle, ou comme un environnement de calcul qui enveloppe l'application et l'isole de son entourage. C'est en fait un environnement de calcul portable complet. Avec les conteneurs, les développeurs n'ont plus besoin de coder pour une plateforme ou un système d'exploitation en particulier, une méthode qui complique le déplacement des applications étant donné que le code n'est pas toujours compatible avec le nouvel environnement. De plus, ces transferts génèrent souvent des bogues, des erreurs et des problèmes qui font perdre du temps, diminuent la productivité et engendrent une grande frustration. En plaçant une application dans un conteneur facile à déplacer entre les plateformes et infrastructures,».

Selon IBM [25] « La conteneurisation est l'emballage du code logiciel avec uniquement les bibliothèques du système d'exploitation (OS) et les dépendances nécessaires pour exécuter le code afin de créer un seul exécutable léger, appelé conteneur, qui s'exécute de manière cohérente sur n'importe quelle infrastructure. Plus portables et économes en ressources que

[24] Redhat, "La conteneurisation, qu'est-ce que c'est ?," Apr. 08, 2021. https://www.redhat.com/fr/topics/cloud-native-apps/what-is-containerization (accessed Jun. 23, 2022).

[25] IBM cloud Education, "Containerization Explained | IBM," Jun. 23, 2021. https://www.ibm.com/cloud/learn/containerization (accessed Jun. 23, 2022).

les machines virtuelles (VM), les conteneurs sont devenus de facto les unités de calcul des applications cloud natives modernes ».

Le conteneur est une partie de software qui comprend un ensemble de packages, des dépendances et des outils nécessaires pour l'exécution d'application. Le conteneur est très léger car il comprend que ce que l'application a besoin pour s'exécute sans aucune erreur.

La Conteneurisation permettre de couvrir l'application avec une image, cette dernière contient tous les choses nécessaires pour exécuter l'application.

II.4.5.3 Différence entre la virtualisation traditionnelle et la conteneurisation [52]

Le tableau et la figure 11 ci-dessous montre la différence entre la virtualisation et la conteneurisation :

	Virtualisation	**Conteneurisation**
Isolation	Fournit une isolation complète du système d'exploitation hôte et des autres machines virtuelles	Fournit généralement une isolation légère de l'hôte et des autres conteneurs, mais ne fournit pas une limite de sécurité aussi forte qu'une machine virtuelle
Système D'exploitation (OS)	Exécute un système d'exploitation complet, y compris le noyau, nécessitant ainsi plus de ressources système telles que le processeur, la mémoire et le stockage	Exécute la partie en mode utilisateur d'un système d'exploitation et peut être personnalisé pour contenir uniquement les services nécessaires à votre application en utilisant moins Ressources
Compatibilité Guest	Exécute à peu près n'importe quel système d'exploitation à l'intérieur de la machine	S'exécute sur la même version du système d'exploitation que l'hôte
Déploiement	Déployez des machines virtuelles individuelles à l'aide du logiciel Hypervisor	Déployez des conteneurs individuels à l'aide de Docker ou déployez plusieurs conteneurs à l'aide d'un orchestrateur tel que Kubernetes
Stockage	Utilisez un disque dur virtuel (VHD) pour le stockage local d'une	Utilisez des disques locaux pour le stockage local d'un seul nœud ou

	seule machine virtuelle ou un partage de fichiers SMB (Server Message Block) pour le stockage partagé par plusieurs serveurs	SMB pour le stockage partagé par plusieurs nœuds ou serveurs
Équilibrage de charge	L'équilibrage de charge des machines virtuelles est effectué en exécutant des machines virtuelles sur d'autres serveurs dans un cluster de basculement	Un orchestrateur peut automatiquement démarrer ou arrêter des conteneurs sur des nœuds de cluster pour gérer les changements de charge et de disponibilité.

Tableau 3 : La différence entre la virtualisation la Conteneurisation

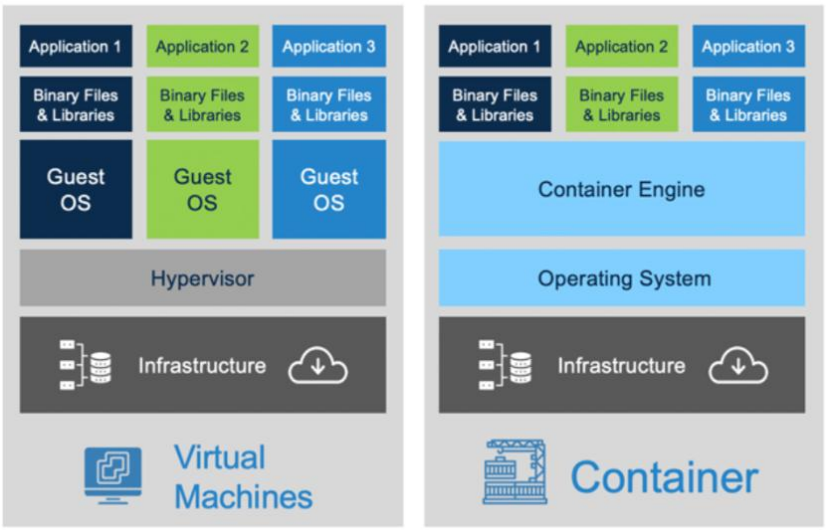

Figure 11: Comparaison entre virtualisation et conteneurisation[53]

II.4.5.4 Hyperviseur

Selon VMware [26]« Un hyperviseur, également appelé moniteur de machine virtuelle, est un processus qui crée et exécute des machines virtuelles (VM). Il permet à un ordinateur hôte de prendre en charge plusieurs VM clientes en partageant virtuellement ses ressources, telles que la mémoire et la capacité de traitement. En général, il existe deux types d'hyperviseurs : L'hyperviseur de type 1, nommé « bare metal » s'exécute directement sur le matériel de l'hôte. L'hyperviseur de type 2, nommé « hébergé » s'exécute sous forme d'une couche

[26] VMware, "Qu'est-ce qu'un hyperviseur ? | Glossaire VMware | FR | FR," Jan. 09, 2022. https://www.vmware.com/fr/topics/glossary/content/hypervisor.html (accessed Jun. 23, 2022).

logicielle sur un système d'exploitation, comme n'importe quel autre programme informatique ». Les hyperviseurs sont souvent déployés pour permettre à une seule machine d'héberger plusieurs applications non liées, qui peuvent s'exécuter pour le compte d'organisations indépendantes, comme c'est souvent le cas lorsqu'un datacenter consolide plusieurs serveurs physiques. Les applications dans un tels scénario n'ont pas besoin de partager des informations. En effet, il est important qu'ils n'aient pas d'impact l'un sur l'autre. Pour cette raison, les hyperviseurs privilégient fortement l'isolement complet au partage. Cependant, lorsque chaque machine virtuelle exécute le même noyau et des distributions de système d'exploitation similaires, le degré d'isolement offert par les hyperviseurs se fait au détriment de l'efficacité par rapport à l'exécution de toutes les applications sur un seul noyau [54]. Le système de virtualisation basé sur un hyperviseur existant déploie des applications cloud à l'aide d'une architecture de service monolithique. Par conséquent, les surcharges de performances créées par le système hyperviseur entravent son applicabilité dans les environnements HPC. Cela a conduit à l'émergence d'une nouvelle technologie de virtualisation appelée conteneurisation [55].

II.4.6 Les microservices dans le cloud computing

Les approches de microservices conviennent particulièrement bien au cloud computing, permettant aux avantages économiques des microservices de compléter les avantages économiques du cloud computing, tels que l'optimisation des coûts et de l'expérience utilisateur. De plus, la publication rapide de microservices fonctionne bien avec les systèmes en ligne basés sur le cloud qui ne nécessitent pas de distribution supplémentaire de mises à jour logicielles. Ce que nous appelons le cloud est en fait une masse de microservices, se parlant sur Internet avec le langage universel des API. Une compréhension des microservices ouvre la porte à la participation à cet énorme nouveau marché et/ou à l'exploitation de leurs avantages pour obtenir un avantage concurrentiel [54].

Les microservices sont très utiles pour les applications en cloud computing. L'utilisation de microservices dans le cloud computing permet d'augmenter la popularité du cloud. L'utilisation de microservices offre plus de choix et d'options pour faire évoluer le service de manière indépendante [56].

Les microservices dans le cloud sont souvent déployés dans des conteneurs car c'est ainsi qu'on peut bénéficier au maximum de l'infrastructure et de ressources. Et comme les conteneurs sont isolées, cela permet s'exécutent dans n'importe où d'une façon très facile et

rapide, sans oublier que il est aussi possible de déployer des microservices dans le cloud sans utiliser de conteneurs mais cela on perdre un peu l'utilité d'utiliser les micro services.

Les plates-formes cloud populaires disposent de plusieurs fonctionnalités adaptées aux microservices, telles que :

- Ressources disponibles selon la demande,
- Tarification réduite et selon la consommation réelle,
- Déploiement et livraison continus,
- Plusieurs Services gérés par les fournisseurs des cloud (par exemple, la mise à l'échelle, la configuration, les mise à jour des logiciels),
- L'Ouverture et le confort dans le choix des technologies notamment les langages de programmations et les data bases, et aussi le choix des système d'exploitation,
- Outils de DevOps intégrés tels que Docker et Kubernetes et d'autres outils d'automatisation.

II.4.7 Les microserivces et la conteneurisation

Le modèle monolithique traditionnel de déploiement d'applications sur site est rapidement remplacé par un paradigme de microservices basé sur le cloud, en partie grâce à la montée en puissance de nombreux fournisseurs d'infrastructure cloud offrant un accès transparent à une variété de matériel informatique, et au besoin d'applications pour servir un public toujours plus important nécessitant une évolutivité. Bien que la virtualisation basée sur des conteneurs ait été la méthode préférée de déploiement de microservices, les consommateurs de cloud n'ont jusqu'à présent pas eu beaucoup d'opportunités d'optimiser les coûts et les ressources[57]. Les conteneurs sont généralement utilisés pour séparer des fonctions individuelles (microservices) qui exécutent une tâche particulière. Les microservices sont le produit de la décomposition des applications en services plus petits et plus spécialisés. Ils permettent aux développeurs de travailler sur une partie spécifique d'une application sans affecter ses performances globales[27]. Les conteneurs combinent les applications et leurs configurations et leurs dépendances en une seule entité déployable autonome. Les conteneurs sont idéaux pour regrouper et déployer des microservices indépendants [76].

[27] Redhat, "La conteneurisation, qu'est-ce que c'est ?," Apr. 08, 2021. https://www.redhat.com/fr/topics/cloud-native-apps/what-is-containerization (accessed Jun. 23, 2022).

II.4.7.1 Microservice et docker

Les conteneurs docker et les microservices sont devenus plus ou moins analogues à l'alimentation du cloud. Aucune conversation sur le cloud n'est sans ces termes et les défenseurs des microservices ont tendance à le présenter comme un remède aux grands maux monolithiques. C'est aussi une solution pour s'adapter au reste des nouveaux logiciels en cours de développement. Il y a aussi beaucoup de confusion et d'hésitation, en particulier chez les entreprises, quant à la migration complète vers le cloud avec ces technologies. L'approche de Docker en matière de conteneurisation repose sur la décomposition des applications. En d'autres termes, il s'agit de la capacité de réparer ou de mettre à jour des parties d'une application sans mettre hors service l'intégralité de l'application. En plus de cette approche basée sur les microservices[28], Docker permet de partager des processus entre différentes applications[59].

Dans cette relation entre les microservices et les conteneurs, une bonne règle empirique est un microservice par conteneur. Le concept de mise à l'échelle dynamique est mieux traité si microservice est hébergé par conteneur. Voici l'illustration (Fig.12) de docker sur la façon d'utiliser docker swarm et compose ensemble pour gérer les clusters de conteneurs

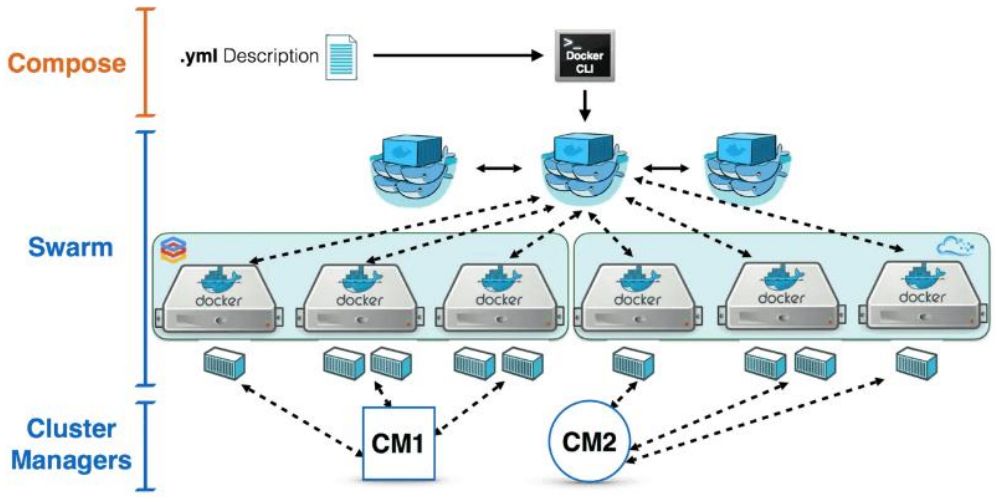

Figure 12: Fonctionnement de docker [60]

[28] RedHat, "Docker : définition, fonctionnement et avantages," Jan. 09, 2018. https://www.redhat.com/fr/topics/containers/what-is-docker (accessed Jun. 23, 2022).

Par définition, un microservice doit avoir trois composants [29]:

- Interface utilisateur
- Logique métier
- Connexion de données

Les microservices doivent également être autonomes avec toutes les bibliothèques disponibles dans le conteneur. En bref, la technologie Docker offre une approche granulaire, contrôlable, rapide et basée sur des microservices qui place l'efficacité au cœur de ses objectifs.

II.5 Conclusion

Dans le présent chapitre, on a présenté le développement des différentes approches logicielles, notamment l'architecture monolithique, SOA et microservices. Une attention particulière a été donnée à l'architecture microservice. En comparaison avec d'autres architectures, les microservices sont plus efficaces dans le cas où l'application doit gérer un plus grand nombre de requêtes. Ils permettent de construire un logiciel de haute qualité, facile à mettre à l'échelle, plus fiable et à long terme plus pratique à entretenir. D'une manière générale, les microservices offrent plus de choix et d'options pour faire évoluer le service de manière indépendante. En plus, ils sont très avantageux et conviennent mieux pour les applications cloud computing. Leurs utilisations permettent d'augmenter la popularité du cloud computing et sont souvent déployés dans des conteneurs afin de bénéficier au maximum de l'infrastructure et de ressources.

Dans le prochain chapitre, nous allons survoler quelques travaux connexes dans sélection et la découverte des microservices.

[29] Amzone Web Service, "Que sont les microservices ? | AWS," 2020. https://aws.amazon.com/fr/microservices/ (accessed Jun. 24, 2022)

Chapitre III SOLUTIONS EXISTANTES POUR LA DÉCOUVERTE ET LA SÉLECTION DES MICROSERVICES

III.1 Introduction

Les défis des microservices mentionnés dans le chapitre 2 ont fait l'objet de plusieurs efforts ayant visé la résolution de la problématique de la sélection et la découverte. Dans ce chapitre nous présenterons ces efforts. Nous présentons et nous discuterons les solutions proposées pour réaliser la sélection et la découverte des microservices dans un environnement Cloud.

III.2 La découvertes des microservices

III.2.1 Définition de la découvrabilité

Les applications basées sur des microservices s'exécutent généralement dans des environnements virtualités ou conteneurisés. Le nombre et l'emplacement des instances de service changent de manière dynamique [68]. Pour qu'une requête atteigne le microservice cible, nous devons savoir où se trouvent ces instances et comment elles s'appellent. C'est là que des tactiques telles que la détection de service sont utiles. Le mécanisme de découverte de service aide à savoir où se trouve chaque instance, c'est-à-dire un processus consiste à prendre le nom d'un service en entrée et à obtenir l'emplacement réseau d'une instance de ce service en sortie. De cette manière, le composant de découverte de service agit comme un registre qui conserve une trace des adresses de toutes les instances. L'instance a un chemin réseau attribué dynamiquement. Par conséquent, si le client demande un service, sa demande est traité avec la de découverte des services.

III.2.2 Principaux modèles de la découverte des services

Il existe deux principaux modèles de découverte de service : la découverte côté client et la découverte côté serveur. Examinons d'abord la découverte côté clients.

III.2.2.1 Coté client

La découverte de services côté client permet aux applications clientes de trouver emplacement réseau des instances de services disponibles en parcourant ou en interrogeant un registre de services [68], qui est une base de données des instances de service disponibles (fig13). Le client utilise ensuite un algorithme d'équilibrage de charge pour sélectionner l'une des instances de service disponibles et effectue une requête [69]. La figure 13 exprime la découverte coté client :

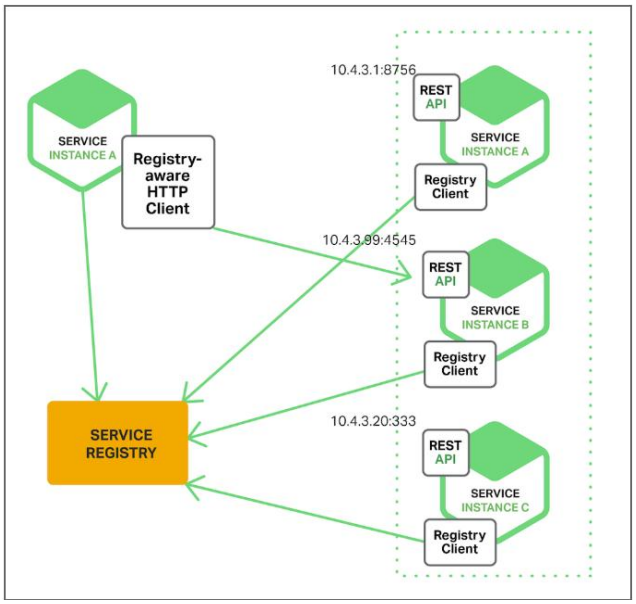

Figure 13 Découverte coté client [69]

III.2.2.2 Coté serveur

La découverte de services côté serveur permet aux applications clientes de trouver des services via un intermédiaire qui agit comme un équilibreur de charge [70]. Le client fait une demande à un service via un équilibreur de charge qui agit comme un orchestrateur. L'équilibreur de charge interroge le registre des services et achemine chaque demande vers une instance de service disponible. La figure 14 représente la découverte cotée serveur :

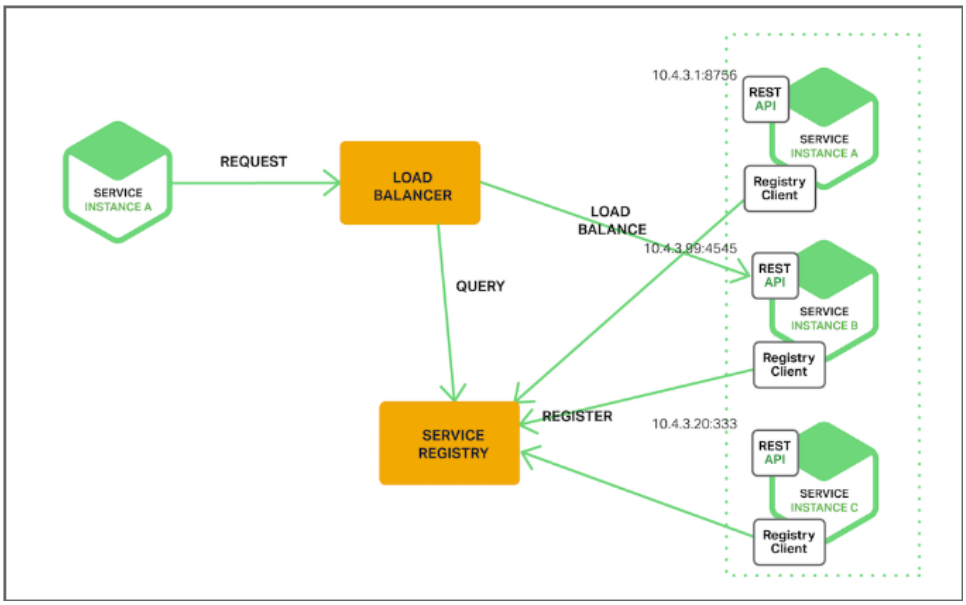

Figure 14: Découverte coté serveur [70]

III.2.3 Les technologies existantes pour la découverte

III.2.3.1 Eureka

Le serveur Eureka est un modèle de découverte des microservices fournit par Netflix, où les microservices peuvent s'enregistrer pour que d'autres puissent les découvrir. C'est une application qui contient les informations de tous les services clients. Chaque micro service s'enregistrera sur le serveur Eureka et le serveur Eureka connaît toutes les applications client exécutées sur chaque port et adresse IP [71]. Ce serveur Fournit une API REST pour enregistrer et interroger les instances de service. L'instance de service utilise une requête POST pour enregistrer l'emplacement réseau. Vous devez mettre à jour l'enregistrement avec une requête PUT toutes les 30 secondes. Les enregistrements sont supprimés soit à l'aide d'une requête HTTP DELETE, soit à la date d'expiration de l'enregistrement d'instance[69]. Comme vous pouvez l'imaginer, les clients peuvent utiliser une requête HTTP GET pour obtenir une instance de service enregistrée. Eureka Server est également connu sous le nom de Discovery Server.

III.2.3.2 Zookeeper

Zookeeper fournit une solution open source aux divers problèmes de coordination dans les grands systèmes distribués. ZooKeeper, en tant que service de coordination centralisé, est distribué et hautement fiable, fonctionnant sur un cluster de serveurs appelé ensemble ZooKeeper [72]. Le consensus distribué, la gestion de groupe, les protocoles de présence et l'élection du leader sont implémentés par le service afin que les applications n'aient pas besoin de réinventer la roue en les implémentant par elles-mêmes. En plus de cela, les primitives exposées par ZooKeeper peuvent être utilisées par des applications pour créer des abstractions beaucoup plus puissantes pour résoudre une grande variété de problèmes.

III.2.3.3 Consul

Consul est un outil multi-réseaux qui offre une solution de maillage de services complète qui résout les problèmes de mise en réseau et de sécurité liés à l'exploitation de microservices et d'une infrastructure cloud [73]. Il propose un Framework de découverte de services avec une interface et certaines fonctionnalités telles que la vérification de l'état, la segmentation des services avec son propre magasin de clé-valeur distribué interne. Consul inclut la découverte de services, mais aussi la vérification de l'état de santé, le verrouillage, la clé/valeur, la fédération multi-centres de données, un système d'événements, la gestion des pannes, les tentatives et l'observabilité du réseau. Chacune de ces fonctionnalités peut être utilisée

individuellement selon les besoins ou elles peuvent être utilisées ensemble pour créer un maillage de services complet et atteindre une sécurité zéro confiance.

III.2.3.4 Tableau comparatif entre les différentes technologies de la découverte

Le tableau ci-dessous représente la comparaison entre les différentes technologies proposées dans la découverte des microservices :

Caractéristique	**Eureka**	**Consul**	**Zookeeper**
Intégration de spring cloud	Supporté (version Angel SR3)	Supporté (version Brixton)	Supporté (version 3.4.12)
Gestion des conteneurs	Supporte docker	Supporte docker	Ne supporte pas docker
Plusieurs centres de données	Supporté	Supporté	Ne supporte pas plusieurs data centers
HTTP	Utilise des requêtes http	Utilise des requêtes http	Utilise des requêtes http
Nombre des instances recommandé par centre de données	2 ou plusieurs instances de serveur connectées en tant que pairs	3 à 5 instances par Datacenter avec la possibilité de connecter des centres de données.	Ne supporte pas
Collecte des données du bilan de santé	-Hystrix avec (Hystrix Dashboard) -Agrégation via Turbine	Intégré à la page d'accès Web Consul.io.	Données de santé non collecté

Tableau 4: comparaison entre les différentes technologies proposées pour la découverte des microservices

III.3 La sélection des microservices

Dans cette partie, nous mentionnerons certains travaux des chercheurs pour la sélection des microservices. L'environnement de déploiement et les exigences d'exécution pour les microservices et les services Web sont différents. Lors du choix d'un service web, il faut tenir compte des aspects de qualité de service tels que la fiabilité, la disponibilité et le temps de réponse. Contrairement aux choix des microservices, il faut pas seulement prendre en compte les informations d'attribut QoS, mais il faut prendre aussi autres critères, tels que la charge du serveur et d'autres aspects [61].

III.3.1 Solution de J.Shao et al. [61]

J.Shao et al. [61] ont proposé un algorithme de sélection d'instances contextuelle de microservices nommée ACISM. Cet algorithme utilise des informations de la base de données de contexte historique pour déterminer dynamiquement le poids de chaque information contextuelle. L'algorithme ACISM utilise cinq paramètres (Utilisation de CPU, nombre de connexions de requête sur le microservice, temps de réponse attendu des instances de microservice, disponibilité prévue des instances de microservice, fiabilité attendue des instances de microservice) dans son contenu, puis il a travaillé avec les valeurs normalisées de ces cinq paramètres, ensuite il a calculé les ecart-types des paramètres normalisées des instances des microservices. L'écart type est calculé sur la base des informations contextuelles historiques acquises. Ces informations sont utiles pour définir et calculer les pondérations des informations sur les instances de microservice. ACISM est utilisé dans Spring Cloud et ne considère qu'un seul microservice, mais un microservice comprend de nombreuses instances. Dans la démarche expérimentale, les auteurs de [61] ont principalement analysé deux situations. L'une est qu'un grand nombre d'utilisateurs accèdent au même microservice dans le système en même temps, et l'autre est que le même utilisateur accède plusieurs fois au même microservice dans le système. Grâce à la comparaison et à l'analyse des résultats expérimentaux, ces auteurs ont conclu que l'ACISM garantit le temps de réponse le plus court lorsqu'un grand nombre d'utilisateurs accèdent au même microservice, et que la précision de la sélection des instances est bonne lorsque le même utilisateur accède plusieurs fois au même microservice. Les résultats obtenus par l'algorithme permettent de sélectionner la meilleure instance de microservice. Toutes ces informations ont aidé à affaiblir le caractère décisif de la sélection subjective de l'utilisateur et donc l'algorithme a conduit à un équilibrage entre l'influence des facteurs subjectifs et les facteurs objectifs, et a aussi amélioré la précision de la sélection des instances de microservice et a

réduit le temps de réponse. Ces auteurs ont aussi comparé leurs algorithmes avec deux autres algorithmes. Le premier est un algorithme de sélection de services basé sur la QoS dynamique pour la fluctuation de la QoS des services Web dans des environnements dynamiques de sélection des services, c'est-à-dire, il est basé sur le vecteur de préférence subjective de l'utilisateur [62]. Le deuxième est un algorithme de sélection de service basé sur les caractéristiques de service [63]. Les résultats obtenu ont montré que l'algorithme de [61] a une meilleure précision dans la sélection des services et un temps de réponse plus court.

III.3.2 Solution de Zhiying Cao et al. [64]

Zhiying Cao et al. [64] ont proposé un algorithme de la sélection et la découverte de microservices en cluster de contexte nommé ACCMD où ils ont classé le contexte en deux parties (Contexte Service, Contexte utilisateur). Selon ces auteurs, l'utilisation de contexte est utile pour trouver des services qui répondent mieux aux besoins des utilisateurs. Le contexte de microservices est défini par deux paramètres (SInfo, QoS), tels que SInfo représente les informations de base de microservice et QoS représente les informations sur la qualité de service. Le contexte utilisateur est divisé en deux parties : les caractéristiques de utilisateurs et la préférence de l'utilisateur. Les microservices regroupés en fonction de la similarité du contexte de service et un ensemble de services candidat est choisi par la correspondance entre la requête et les clusters de services. Pour chaque cluster, un service est choisi qui est nommé le service central. Celui-ci est un représentant des services de ce cluster. Lorsqu'une demande est reçue, la similarité de contexte est calculée entre la requête et les services centraux des clusters. Selon la similarité du contexte, les utilisateurs qui ont utilisé les services candidats sont regroupés, et l'ensemble de services candidats est affiné en faisant correspondre les informations du demandeur avec le contexte des utilisateurs dans ces clusters. Enfin, la qualité de service (QoS) et la préférence du demandeur sont utilisées pour filtrer les services candidats et les résultats sont renvoyés au client. Les auteurs de [64] ont fait des études et des tests expérimentaux comparant leur algorithme avec d'autres algorithmes de découverte Eureka et ZooKeeper et[65]. Les résultats ont montré qu'en raison du regroupement des services et des utilisateurs par contexte de service et contexte d'utilisateur, l'algorithme ACCMD est meilleur que les autres algorithmes en termes d'efficacité de découverte sur les services, et que les taux de rappel et la précision sont également plus élevés.

III.3.3 Solution de Zeina Houmani el al. [66]

Zeina Houmani el al. [66] ont proposé un Framework de découverte de services piloté par les données basé sur la mise en correspondance de profils à l'aide de descriptions de services

centrées sur les données. Le profil est spécifié par le client afin de satisfaire ses besoins. Donc on peut définir le profil comme le besoin de service au client. La plateforme élaborée par les auteurs de [66] permet aux programmes clients de découvrir les fonctionnalités et les microservices disponibles en fonction de leurs objets de données, ce qui veut dire que les données d'objets des clients sont examinées au niveau de la plateforme afin de bien déterminer le microservice correspondant. Les anciennes découvertes des services sont basées généralement sur des informations simples notamment le nom de service et des informations sur l'emplacement réseau du fournisseur de services. Le projet Consul [67], par exemple, utilise un modèle de description de service qui contient des configurations de réseau ainsi que des données d'identification de service telles que le nom, l'ID et l'étiquette du service. Cependant dans la découverte des services pilotés par les données, des informations supplémentaires liées à la performance du service et à la fonctionnalité fournie doivent être spécifiées. La solution proposée repose principalement sur le calcul d'adéquation entre les exigences des produits de données et les services fournis par les fournisseurs. Cette solution utilise une architecture de microservices pilotée par les données avec un réseau peer to peer qui permet une découverte de service évolutive avec la possibilité d'intégrer des caractéristiques géographiques. Les auteurs de [66] ont déployés cette architecture sur un banc d'essai réel. Les résultats montrent que la plateforme est capable d'adapter et de maintenir la qualité de service en termes de temps de réponse et de pourcentage de demandes acceptées lors de la réception de débits entrant qui dépassent la capacité du système.

III.3.4 Étude comparative entre les solutions proposées pour la sélection des microservices

Le tableau ci-dessous présente un résumé des différentes solutions que nous avons examinées et que nous avons présentées dans les sections précédentes :

Auteur	[61]	[64]	[66]
Nombre de microservices	Un seul microservice avec plusieurs instances	Plusieurs microservices qui sont différents	Plusieurs microservices qui sont différents
Paramètres pris en considération	- Utilisation de CPU - nombre de connexions de requête sur le microservice - temps de réponse attendu des instances de	-les informations de base d'un microservice tel que : le nom, input/output, condition, résultats, loadbalnceType. - Critère de QoS : prix,	-les caractéristiques d'objet - QoS (des informations sur le réseau et sur la

	microservice - Disponibilité prévue des instances de microservice - Fiabilité attendue des instances de microservice	temps de réponse, disponibilité, SuccessRate (taux de réussite de l'appel de MS), Feedback	performance de microservice) -la fonctionnalité de microservice
Objectifs	-équilibrage entre l'influence des facteurs subjectifs et les facteurs objectifs - amélioré la précision de la sélection des instances de microservice -réduire le temps de réponse	-améliorer le taux de rappel et le taux de la précision de sélection des microservices. -augmenter la vitesse de recherche sur les microservices	- maintenir la qualité de service en termes de temps de réponse -augmenter le pourcentage de demandes acceptées lors de la réception de débits entrant qui dépassent la capacité du système
Environnement	-cloud	- cloud	- cloud
Requête utilisateur	-pas traité	- pas traité	- pas traité
Composition des microservices	-Pas traité	- pas traité	- traité
Principes	- Calculé l'écart types des valeurs normalisés des critères de sélection pour définir et calculer les pondérations des informations sur les instances de microservice	- Les microservices regroupés en fonction de la similarité du contexte en clusters, Les utilisateurs regroupés en fonction de la similarité du contexte. Puis à travers la correspondance puis un ensemble de services candidat est choisi par la correspondance entre la	-calcule de l'appariement entre les produits des données et les services, les données d'objets des clients sont examinées au niveau de la plateforme afin de bien déterminer le microservice

		requête et les clusters de services enfin, la qualité de service (QoS) et la préférence du demandeur sont utilisées pour filtrer les services candidats et les résultats sont renvoyés au client	correspondant.

Tableau 5: Solutions proposées pour la sélection des microservices.

III.3.5 Discussion sur les solutions proposées par les chercheurs :

Les solutions représentées dans le tableau 5 résolvent des problèmes liés à la découverte et à la sélection des microservices. Chaque solution est basée sur son propre contexte. Le contexte est l'ensemble des informations qui peuvent être utilisées pour décrire une entité. Une entité peut être l'utilisateur ou bien le système lui-même, ou peut aussi être l'objet pertinent dans l'interaction entre le système et l'utilisateur.

La solution de [64] classe le contexte en deux parties: contexte service, contexte utilisateur. Ces deux contextes ont été décrits à l'aide d'un ensemble de critères de QoS et d'autres informations de base de microservices.

La Solution de [61] utilise des critères de QoS, tels que le temps de réponse, la disponibilité et d'autres critères liés à la charge des serveurs. Par contre la solution de [64] utilise plus de critères de QoS en comparaison avec celle de [61]. Plus précisément la solution de [64] utilise les critères prix, SuccessRate et le FeedBack, que la solution de [61] n'a pas considéré. La solution de n'a pas pris en considération les critères des solutions de [61] et de [64]. Elle s'est basée sur des informations sur les réseaux et les performances des microservices, et elle a tenu compte des caractéristiques d'objet et la fonctionnalité de microservices dans la sélection. Les solutions des [64] et [66] travaillent sur des microservices différents. Par contre la solution de [61] travaille sur plusieurs instances pour le même microservice. Toutes les solutions travaillent dans en environnement cloud. La composition des microservices est traitée par la solution [66], par contre les solutions de [61] et [64] n'ont pas tenu compte de la composition. L'aspect sémantique n'a pas été pris en charge par toutes les solutions du tableau 5.

En ce qui concerne les objectifs, Les solutions [61] et [64] partagent un objectif en commun qui est l'amélioration de taux de rappel. La solution [61] s'intéresse aussi à l'équilibrage entre l'influence des facteurs subjectifs et les facteurs objectifs tandis que la solution de [64] possède autres objectifs qui sont l'augmentation de la vitesse de recherche des microservices et l'amélioration du taux de précision. De même les solutions [61] et [66] possèdent un objectif en commun qui est la minimisation du temps de réponse. En dehors de cet objectif en commun, la solution [66] admet un autre objectif qui est l'augmentation du pourcentage de demandes acceptées lors de la réception de débits entrant qui dépassent la capacité du système.

III.4 Les recherches liées aux api gateway existes pour la sélection des microservices

III.4.1 Définition d'API Gateway

Une API Gateway est un serveur qui est le point d'entrée unique dans le système. Elle permet d'encapsuler l'architecture système interne et fournit une API adaptée à chaque client. API Gateway est responsable du routage des demandes, de la composition et de la traduction du protocole. Toutes les demandes des clients passent d'abord par la passerelle API, cette dernière achemine ensuite les demandes vers le microservice approprié. La passerelle API traite souvent une demande en appelant plusieurs microservices et en agrégeant les résultats. L'utilisation de la gateway implique la traduction entre les protocoles Web tels que HTTP et WebSocket et les protocoles Web non conviviaux qui sont utilisés en interne. Elle peut avoir d'autres responsabilités telles que l'authentification, la surveillance, l'équilibrage de charge, la mise en cache, la mise en forme et la gestion des demandes ainsi que la gestion des réponses statiques [74].

La figure suivante montre comment une passerelle API s'intègre généralement dans l'architecture :

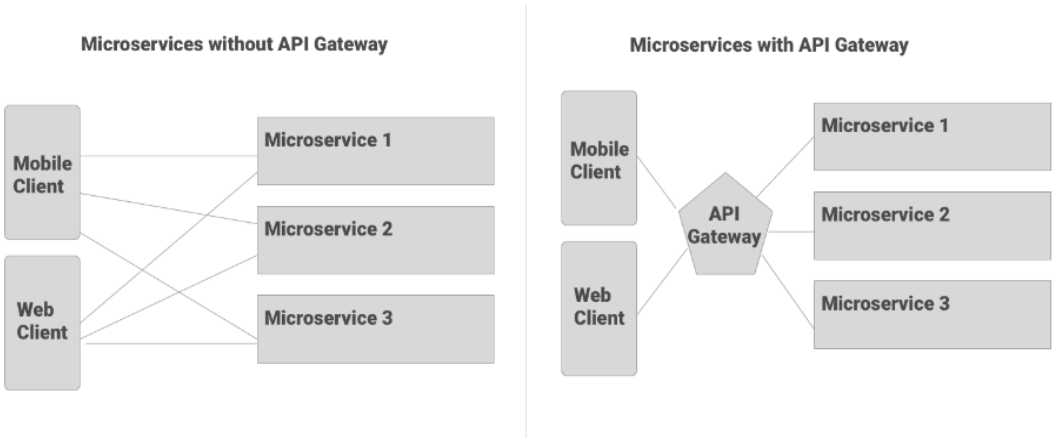

Figure 15: Intégration de api Gateway [74]

III.4.2 API GATEWAY Zuul

Zuul Server est un routeur basé sur JVM et un équilibreur de charge côté serveur de Netflix. Il fournit une entrée unique à notre système, ce qui permet à un navigateur, une application mobile ou une autre interface utilisateur de consommer les services de plusieurs hôtes sans gérer le partage des ressources cross-origin (CORS) et l'authentification pour chacun, c'est-à-dire il gère toutes les requêtes et effectue le routage dynamique des applications de microservices [75]. Zull fonctionne comme une porte d'entrée pour toutes les demandes. Netflix utilise Zuul pour les éléments suivants[76]:

Routage dynamique: il achemine dynamiquement les demandes vers différents clusters sauvegardés selon les besoins.

Authentification et sécurité : il fournit les exigences d'authentification pour chaque ressource.

Insights et surveillance: il suit des données et des statistiques significatives qui nous donnent une vue précise de la production.

Test: Il augmente le trafic vers un cluster afin de tester les performances.

Délestage (load shedding): il alloue la capacité pour chaque type de demande et abandonne une demande qui dépasse la limite.

Gestion des réponses statiques: il génère des réponses directement à la périphérie au lieu de les transmettre à un cluster interne.

Résilience : elle achemine les demandes entre les régions AWS afin de diversifier notre utilisation ELB (Elastic Load Balancing). La figure 16 représente le fonctionnement de proxy Zull

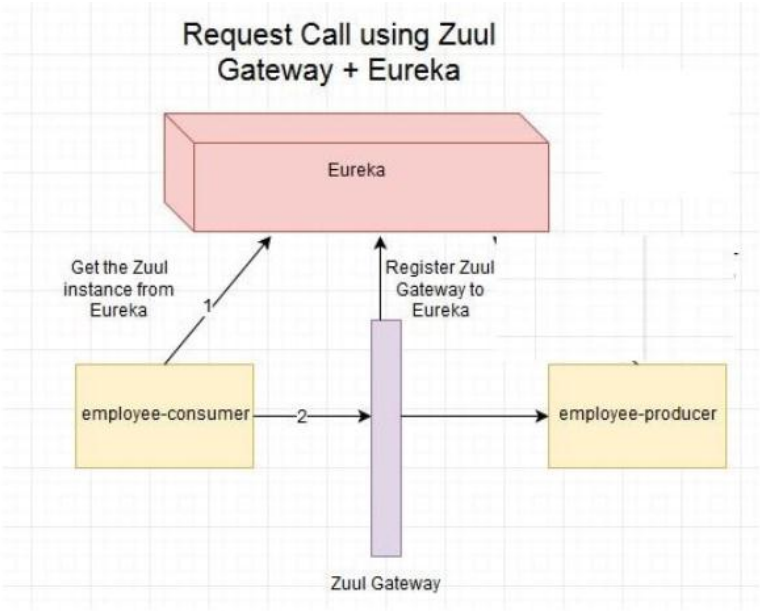

Figure 16 : Fonctionnement de proxy Zull [76]

III.4.3 Spring Cloud Gateway

Spring Cloud Gateway fournit une bibliothèque pour créer des passerelles API sur Spring et Java. Il fournit un moyen flexible d'acheminer les demandes en fonction d'un certain nombre de critères, ainsi que de se concentrer sur des préoccupations transversales telles que la sécurité, la résilience et la surveillance[30].

III.4.3.1 Fonctionnement de spring cloud gateway

La bibliothèque Spring Cloud Load Balancer nous permet de créer des applications qui communiquent avec d'autres applications de manière équilibrée. En utilisant n'importe quel algorithme que nous voulons. Nous pouvons facilement mettre en œuvre l'équilibrage de charge lors des appels de service à distance [77].

La passerelle Spring Cloud fonctionne très bien avec l'écosystème Spring et est un choix évident pour les microservices développés à l'aide du Framework Spring. De plus, il agit comme une couche de façade unique pour tout type de client comme le mobile, le web, etc. L'illustration 17 montre le fonctionnement de spring cloud Gateway [78] :

[30] VMware, "What is Spring Cloud Gateway? | VMware Tanzu Developer Center," 2019. https://tanzu.vmware.com/developer/guides/scg-what-is/ (accessed Jul. 21, 2022)

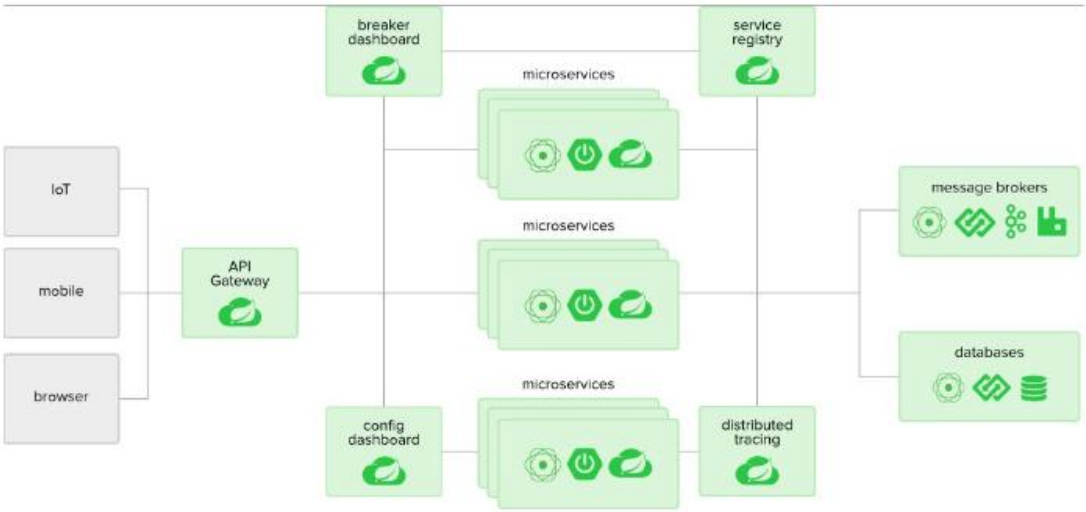

Figure 17 : Fonctionnement de l'API spring cloud gateway[79]

Comme le montre le diagramme ci-dessus, tous les clients utilisent un seul composant et Spring Cloud Gateway s'occupe d'acheminer la demande vers le microservice respectif et obtient la réponse au client. Dans un environnement cloud, Spring Cloud Gateway s'exécute dans le sous-réseau public (adresse IP publique) et tous les autres microservices s'exécutent dans le sous-réseau privé (adresse IP privée). La passerelle Spring Cloud est également un client de découverte avec d'autres microservices et, par conséquent, elle découvre le microservice à partir du registre de découverte de services et achemine vers ces services à l'aide de son adresse IP privée. Ce faisant, nous réduisons le risque d'exposer l'ensemble de l'architecture des microservices au public. Spring Cloud Gateway peut également gérer l'authentification et l'autorisation. L'authentification se fait en intégrant Spring Security. L'autorisation des jetons JWT peut être effectuée ici à un endroit centralisé, réduisant ainsi les frais généraux pour tous les autres microservices [99].

Voici quelques fonctionnalité fournis par l'API spring cloud gateway:

-Capable de faire correspondre les itinéraires sur n'importe quel attribut demandé,

-Les prédicats et les filtres sont spécifiques aux routes,

-Intégration du disjoncteur,

-Intégration du client Spring Cloud Discovery,

-Prédicats et filtres faciles à écrire,

-Intégration de la sécurité Spring.

III.4.3.2 Traitement de la requête de client et les filtres appliquées

La figure suivante fournit un aperçu de haut niveau du fonctionnement de Spring Cloud Gateway :

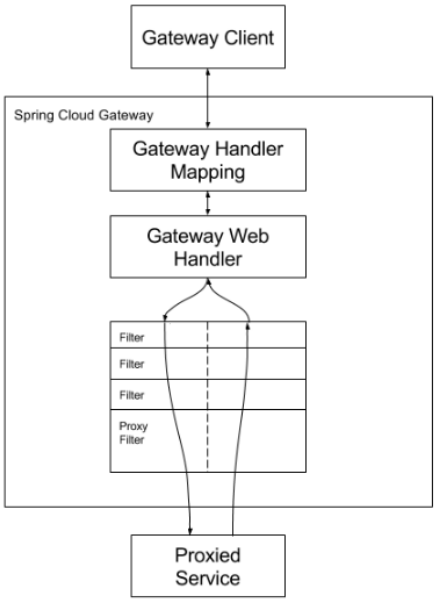

Figure 18: Aperçu de haut niveau du fonctionnement de Spring Cloud Gateway [79]

Les clients font des requêtes à Spring Cloud Gateway. Si le mappage du gestionnaire de passerelle détermine qu'une requête correspond à une route, elle est envoyée au gestionnaire Web de passerelle. Ce gestionnaire exécute la requête via une chaîne de filtres spécifique à la demande. La raison pour laquelle les filtres sont divisés par la ligne pointillée est que les filtres peuvent exécuter une logique à la fois avant et après l'envoi de la demande de proxy. Toute la logique de « pré » filtrage est exécutée. Ensuite, la demande de procuration est faite. Une fois la demande de proxy effectuée, la logique de filtre "post" est exécutée [79].

III.4.4 La différence principale Zull et Spring Cloud Gateway

Zuul développé par Netflix utilise des API bloquantes , c'est-à-dire que pour chaque requête**,** un thread est assigné et ce thread sera utilisé jusqu'à ce que la réponse soit générée. Ainsi, le thread sera bloqué par la demande du cycle de vie complet. La mise à l'échelle dans ce cas serait obtenue en augmentant le nombre de threads qui auront une limite supérieure à un moment donné [80].

Venant à Spring Cloud Gateway qui est développé par spring cloud**,** il prend en charge les API non bloquantes et les connexions de longue durée comme les sockets Web. Il sera plus

facile de faire évoluer l'application puisqu'un thread peut accepter la requête et la laisser pour le traitement et accepter de nouvelles requêtes, et renvoyer les réponses au fur et à mesure qu'elles sont prêtes, afin de ne pas perdre de temps à être bloqué [80].

III.5 Conclusion

Dans ce chapitre nous avons présenté des travaux proposés dans le contexte de la sélection et la découverte des microservices dans un environnement cloud pour avoir une idée sur les différentes solutions proposées par les chercheurs. Nous avons terminé le chapitre par une étude comparative des différentes approches proposées, tenant compte de plusieurs critères, pour voir la différence entre ces approches et pour définir les objectifs de recherche pour chaque approche. Dans le prochain chapitre, nous allons présenter la conception de notre solution proposée.

Chapitre IV CONCEPTION DE LA SOLUTION PROPOSÉE

IV.1 Introduction

Dans le chapitre précédent, nous avons présenté une revue de littérature où on a cité et comparer des solutions existantes pour des problèmes liés à la sélection des microservices basé sur le contexte dans le Cloud Computing. Dans ce chapitre nous présenterons notre solution, qui est un Middleware entre les fournisseurs des services cloud de type FaaS (Function as a Service) et les clients, qui sont les consommateurs des services. Notre Middleware permet aux clients de choisir les bonnes instances des microservices dans un environnement Cloud, selon le critère d'optimisation de temps de réponse (réduction de la latence).Nous commençons par une description de notre solution, puis nous décrivons la méthode agile de travail utilisée pour le développement de notre solution. Ensuite nous présentons une partie comportant l'analyse des besoins de notre système. Enfin, nous présentons notre solution proposée en montrant l'architecture de notre système et en détaillant chaque fonctionnalité essentielle pour chaque acteur de système.

IV.2 Méthode de travail

Pour réaliser notre projet nous avons utilisé la méthode XP (Extreme Programming (XP)) L'Extreme Programming (XP) est une discipline de développement logiciel basée sur des valeurs de simplicité, de communication et de feedback. Cela fonctionne en réunissant toute l'équipe en présence de pratiques simples, avec suffisamment de commentaires pour permettre à l'équipe de vérifier où elle en est et d'adapter les pratiques à sa situation unique [81].

Ce qui distingue XP des autres méthodologies agiles, c'est qu'il met l'accent sur les aspects techniques du développement logiciel. La programmation extrême est précise sur la façon dont les ingénieurs travaillent, car le respect des pratiques d'ingénierie permet aux équipes de fournir un code de haute qualité à un rythme durable.

Extreme Programming (méthode XP) améliore un projet logiciel avec cinq valeurs essentielles ; communication, simplicité, rétroaction, respect et courage. Les programmeurs

extrêmes communiquent constamment avec leurs clients et leurs collègues programmeurs [82].

IV.2.1 Les caractéristiques de la méthodes XP

L'extrême programming base sur cinq valeurs principales: [81][83]

- ✓ **Communication**: XP emploie des pratiques qui nécessitent des communications, telles que :
  - tests unitaires (programmeur-programmeur, client-programmeur)
  - programmation en binôme (programmeur-programmeur)
  - -estimation de tâche (programmeur-gestionnaire, programmeur-client)

- ✓ **Simplicité**: XP nécessite de choisir la tâche la plus simple sur laquelle vous pourriez éventuellement travailler.

- ✓ **feedback**: Les feedback sont fournis par :
  - Versions itératives et incrémentales agressives
  - Versions fréquentes pour les utilisateurs finaux
  - Colocation avec les utilisateurs finaux
  - Tests unitaires automatisés
  - Tests fonctionnels automatisés

- ✓ **Le courage**: XP encourage la prise de mesures/actions drastiques et inattendues telles que jeter du code ou freiner les tests qui ont déjà été exécutés et corriger la faille.
- ✓ **le respect**: Une prémisse fondamentale de XP est que tout le monde se soucie de son travail. Aucune excellence technique ne peut sauver un projet s'il n'y a pas d'attention et de respect

L'extrême Programming est notamment basé sur les concepts suivants[84]:

- Les équipes de développement travaille directement avec le client sur des cycles très courts d'une à deux semaines maximums.
- Les livraisons de versions du logiciel interviennent très tôt et à une fréquence élevée pour maximiser l'impact des retours utilisateurs.
- L'équipe de développement travaille en collaboration totale sur la base de binômes.

- Le code est testé et nettoyé tout au long du processus de développement.
- Des indicateurs permettent de mesure l'avancement du projet afind e permettre de mettre à jour le plan de développement.

IV.3 Analyse et expression des besoins

Dans cette partie, nous présentons les besoins de notre système en décrivant les acteurs qui interagissent avec le système en spécifiant le rôle de chaque acteur à travers un diagramme de cas d'utilisation :

IV.3.1 Acteur de système

Un acteur est une entité qui définit le rôle joué par un utilisateur ou par un système qui interagit avec le système modélisé.

Les acteurs de notre Système sont : Client, Administrateur, Fournisseur.

Le tableau ci-dessous montre le rôle de chaque acteur :

Acteurs	**Rôle**
Client	Consommation des services
Administrateur	Intermédiaire entre les clients et les fournisseurs
Fournisseur	Fournir différents services

Tableau 6: Le rôle de chaque acteur de système

Nous détaillons ce tableau avec un diagramme de cas d'utilisation qui décrivent les fonctions de chaque acteur, comme suit :

IV.3.2 Diagrammes de cas d'utilisation

Le diagramme de cas d'utilisation décrive les fonctions générales et la portée d'un système. Il permet d'identifient également les interactions entre le système et ses acteurs.

Les illustrations suivantes montrent le diagramme de cas d'utilisation de notre système :

IV.3.2.1 Côté client :

L'illustration suivante montre le diagramme de cas d'utilisation de côté client

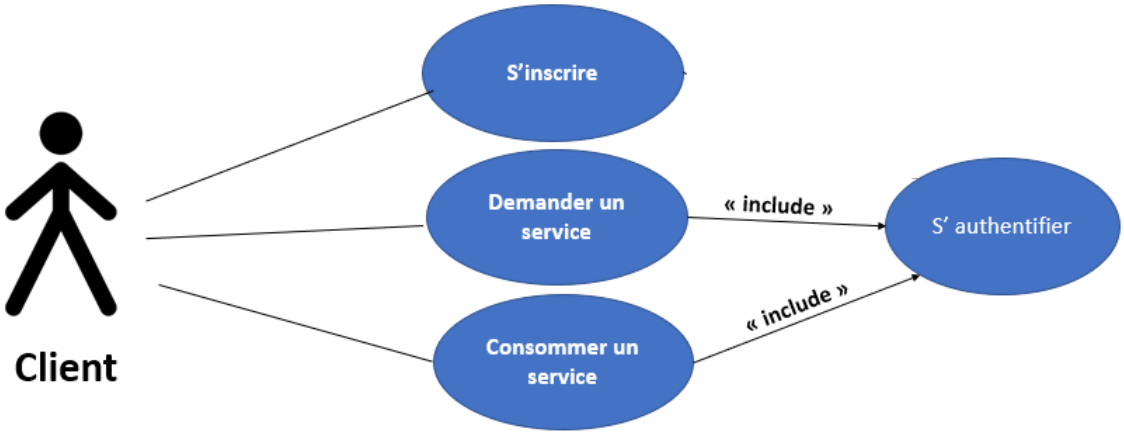

Figure 19: Diagramme de cas d'utilisation de côté client

Le tableau ci-dessous montre la description de cas d'utilisation :

Cas d'utilisation	**Description de cas d'utilisation**
S'inscrire	Les clients s'inscrivent dans le middleware pour être abonnés, pour ceci ils remplissent un formulaire d'inscription
Demande un service	Les clients peuvent demander des services dans n'importe quel moment selon leurs besoins
Consommer un service	Les clients peuvent bénéficier des services des fournisseurs
S'authentifier	Les clients ne peuvent pas accéder au système que s'ils ont le droit, ils doivent donc entrer un nom d'utilisateur et un mot de passe.

Tableau 7: Description de cas d'utilisation

IV.3.2.2 Coté Fournisseur

L'illustration suivante montre le diagramme de cas d'utilisation de côté fournisseur

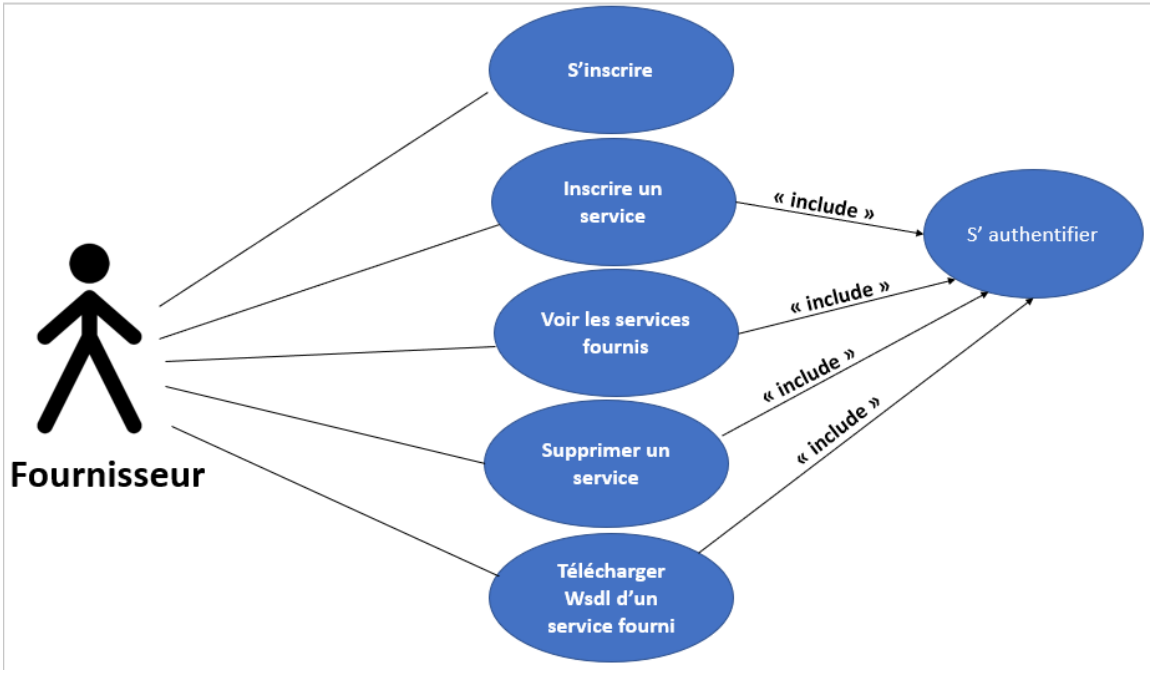

Figure 20 : Diagrame de cas d'utilisation de côté fournisseur

Le tableau ci-dessous montre description de cas d'utilisation de fournisseur :

Cas d'utilisation	Description de cas d'utilisation
S'inscrire	Les fournisseurs s'inscrivent dans le middleware pour être abonnés, pour ceci ils remplissent un formulaire d'inscription
Inscrire un service	Les fournisseurs peuvent fournir différents services, pour ceci ils doivent entrer les contrats WSDL des services
Voir les services fournis	Les fournisseurs peuvent voir leurs services fournis à n'importe quel moment.
Supprimer un service	Un fournisseur peut supprimer un service parmi les services fournis à n'importe quel moment
Télécharger WSDL d'un service fourni	Un fournisseur peut télécharger le fichier WSDL d'un service fourni
S'authentifier	Les fournisseurs ne peuvent pas accéder au système que s'ils ont le droit, ils doivent donc entrer un nom d'utilisateur et un mot de passe.

Tableau 8 : Description de cas d'utilisation de fournisseur

IV.3.2.3 Côté administrateur

L'illustration 21 montre le diagramme de cas d'utilisation de côté Middleware

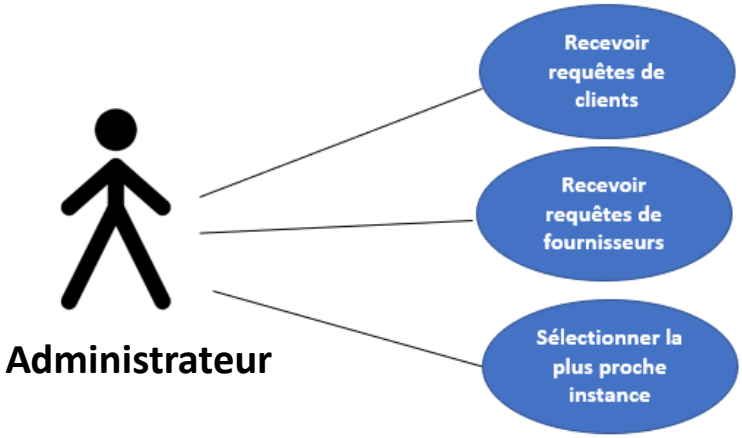

Figure 21: diagrame de cas d'utilisation de côté administrateur

Le tableau ci-dessous montre la description de cas d'utilisation d'administrateur:

Cas d'utilisation	Description de cas d'utilisation
Recevoir requêtes de clients	Le Middleware recevoir les requêtes de clients et acheminés ces derniers vers les microserivces corresponds
Recevoir requêtes des fournisseurs	Le Middleware recevoir les requêtes de fournisseurs et acheminés ces derniers vers les microserivces corresponds
Sélectionner la plus proche instance	Lorsque le client envoie au Middleware une demande de microserivce de fournisseur, le Middleware sélectionne la plus proche instance de microserivce par rapport au client

Tableau 9: Description de cas d'utilisation de fournisseur

IV.4 Description de la Solution

Comme la localisation du serveur par rapport aux clients où l'instance de microservice à été déployé affecte le temps de réponse de microservice, cela nous a conduits à réfléchir de choisir le serveur le plus proche au client. D'une manière équivalente notre problème revient à minimiser la distance entre le client et le serveur exécutant de microservice. Pour être bien précis notre solution proposée est basé sur le choix de l'itinéraire optimal de la requête du client pour consommer un service. Et pour que nous puissions atteindre notre objectif on a réalisé une solution Middleware entre les

fournisseurs des services cloud de type FaaS (Function as a Service) et les clients, qui sont les consommateurs des services. À cette fin, nous avons réalisés une application qui permet de sélectionner aux clients les meilleures instances des microservices fournis par les fournisseurs dans un environnement cloud selon le critère d'optimisation de temps de réponse (réduction de la latence). Nous allons prouvé l'efficacité de notre solution à travers une simulation de notre application à l'aide de l'outil docker compose. Ceci nous a permis de simuler le réseau entre différents microservices de l'application afin de bien communiquer facilement entre différents parties de l'application.

IV.5 L'architecture générale de notre système

La figure ci-dessous représente le schéma général de l'architecture de notre solution qui est basée sur l'architecture microservice :

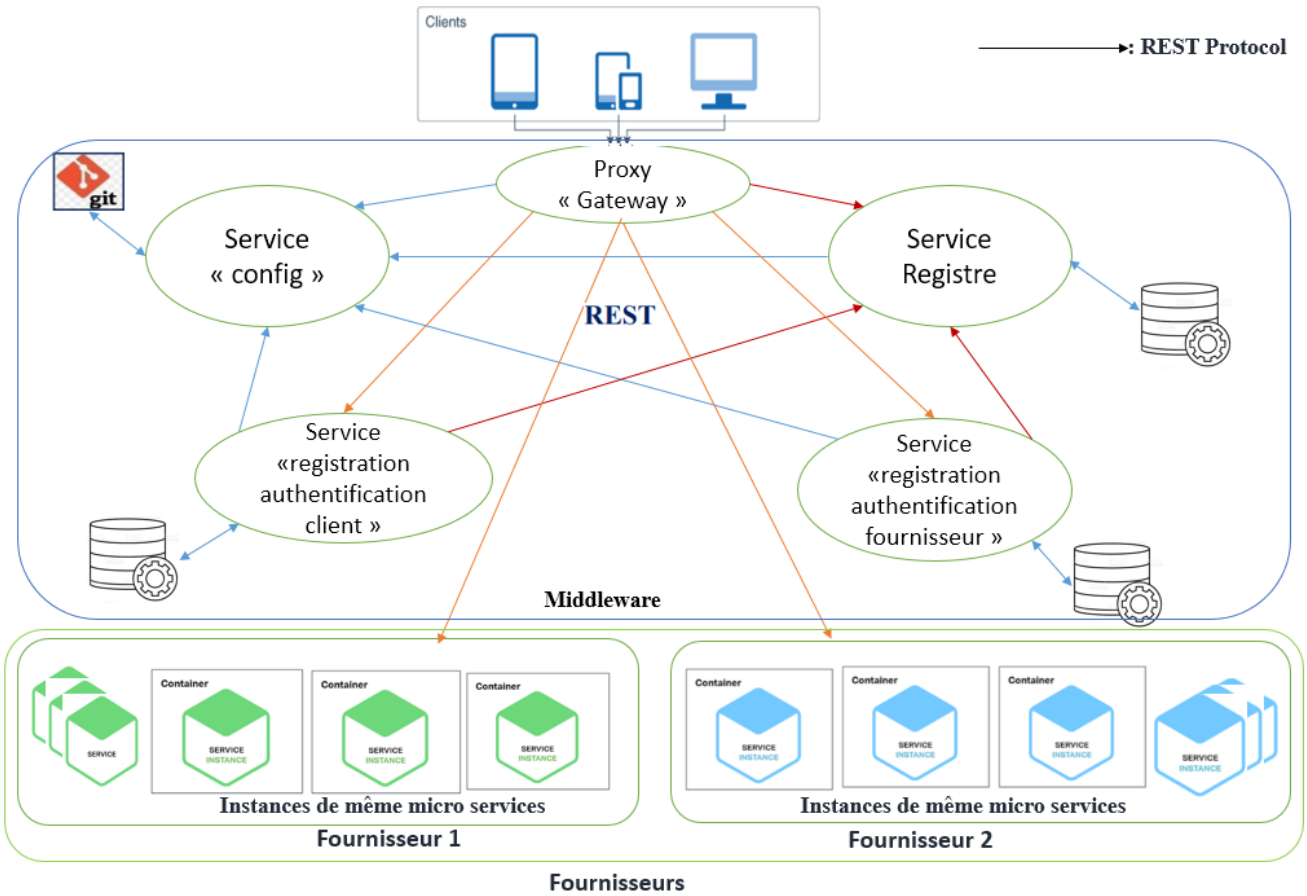

Figure 22: Architecture globale de la solution

Notre architecture composée de trois unités principales : Client, Middleware et le Fournisseur, qui sont détaillés comme suit :

IV.5.1 Le fournisseur

Un fournisseur de cloud est une entreprise qui fournit des services et des solutions basés sur le cloud computing aux entreprises et/ou aux particuliers. Cette organisation de services peut fournir du matériel virtuel, des logiciels, une infrastructure et d'autres services connexes loués et gérés par le fournisseur.

Le fournisseur peut fournir plusieurs microservices, qui peuvent être déployé sur différents Serveurs possédant des adresse IP différentes dispatchée dans le plusieurs zones dans le monde. Le fournisseur peut dupliquer le même microservice plusieurs fois.

Pour que le fournisseur puisse travailler avec le Middleware, il faut s'inscrire et s'abonner dans le Middleware. Pour cela le fournisseur envoie une requête d'inscription au microservice proxy de Middleware. Le proxy achemine cette requête au microservice d'inscription fournisseur. Après l'abonnement du ce dernier, il serait capable d'inscrire ces services. Et pour le faire, Le Middleware donne un accès au fournisseur pour faire entrer les contrats WSDL de consommation des services. Le WSDL contient toutes les informations nécessaires utilisées pour la consommation des services par les clients notamment l'adresse IP, le numéro de port, le nom d'opération, le nom de service et l'URI d'accès au service. Les WSDL sont enregistrées dans un serveur des fichiers.

IV.5.2 Le client

Le client dans notre système est une manière simplifiée de fournir des postes de travail aux utilisateurs pour un accès à tout moment et en tout lieu au microservice proposés par les fournisseurs cloud qui sont inscrits dans le Middleware.

Pour que le client puisse consommer les microservices des fournisseurs qui sont inscrits dans le Middleware, il faut qu'il s'inscrit et s'abonne dans le Middleware. Pour cela le client envoie une requête d'inscription au microservice proxy de Middleware. Le proxy achemine cette requête au microservice d'inscription du client. Ce dernier traite la requête et accepte l'inscription. Après cet évènement, le client serait capable de consommer les services qu'il veut et exploite les bénéfices de Middleware dans la sélection de la meilleure instance de services.

IV.5.3 Le Middleware

IV.5.3.1 Service configuration

Il s'agit d'un service de configuration qui vise à centraliser les fichiers de configuration de divers microservices en un seul endroit. Chaque microservice qu'on a créé possède un ensemble de configurations. Tous les fichiers de configuration de cette architecture sont centralisés sur GitHub. Après le déploiement des microservice, parfois on a besoin de modifier les paramètres de configuration pour notre microservice qui a plusieurs instances. Dans ce cas on est obligé d'arrêter toutes les instances, mais ce n'est pas une bonne pratique. Notre solution pour ce problème consiste à externaliser tous ces fichiers de configuration sur le serveur central comme l'illustre la figure 23.

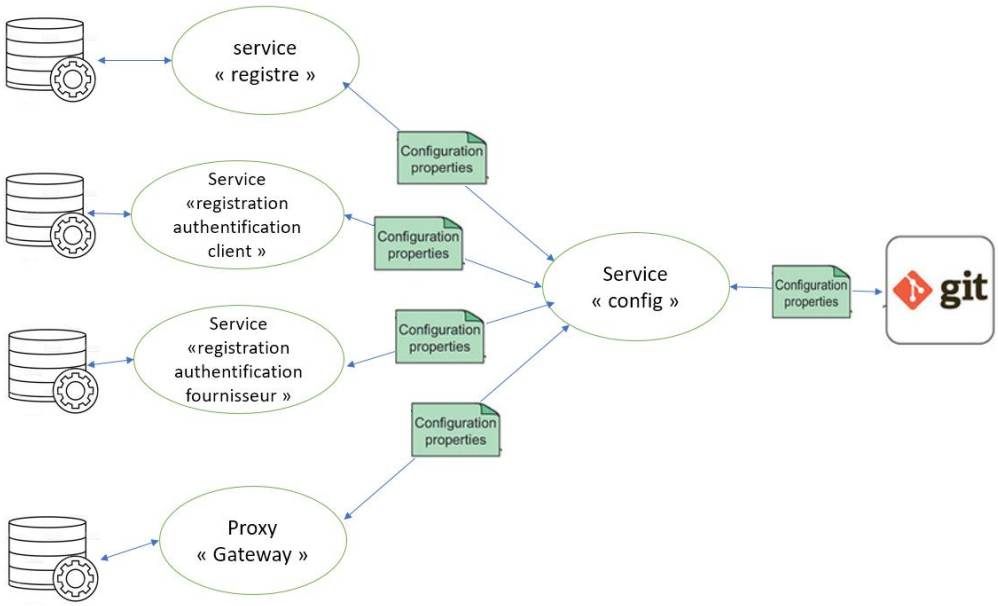

Figure 23 : Service Configuration

Nous avons utilisé cette approche de configuration centralisée dont la démarche peut être résumée comme suit :

- ✓ Dès qu'on démarre les microservices (registre, proxy, registration et authentification client, registration et authentification fournisseur), chaque microservice peut extraire ou récupérer ses données de configuration du service config. Ceci, permet de

simplifier énormément la gestion de nombreux microservices en centralisant leur configuration en un seul endroit.

- ✓ Tous les microservices de notre architecture ont un accès au microservice de configuration
- ✓ Ce service de configuration contient toute la configuration des microservices (microservices registre, microservices proxy, microservices registration et authentification client / fournisseur qu'on a implémenté dans notre architecture.

IV.5.3.2 Le proxy « Gateway »

Le proxy ou la passerelle API représente un point d'entrée unique dans le système, fournissant le routage des demandes coté client et coté fournisseur. API Gateway est un proxy inverse qui accepte tous les appels d'API, applique divers services pour répondre aux appels et renvoie la sortie appropriée. Toutes les requêtes venant du client ou du fournisseur doivent être passer par le proxy. Ce dernier acheminera les requêtes vers les microservices appropriées Cet acheminement est déterminer par un pseudo-algorithme « gateway_algo » que nous avons élaboré et qui est décrit dans le tableau 11 comme suit :

```
Begin
R = requête en cours ;
 If (R == requête client){
                        If (R==demande d'inscription client) {
                              Execute(R,MS_inscription_client) ; }
                        If (R==demande service de fournisseur) {
                              Opération=operation_name(R) ;
                              Liste_address_ip=find_ip_by_operation_name(operation) ;
                              ipMin=min_distance(Liste_address_ip);
                              Execute(R, MS_fournissuer,ipMin) ;
                        } }
If (R == requête de fournisseur ){
                        If (R==demande d'inscription de fournisseur) {
                        Execute(MS_inscription_fournisseur) ; }
                        If (R==demande d'inscrire d'un service ){
                        wsdl=extract_file_wsdl(R);
                        Upload(wsdl);
                        Execute(MS_register) ;      }}
End ;
```

Tableau 10: Algorithme gateway_algo

Cet algorithme traite les requêtes qui proviennent du client et du fournisseur. Au début on affecte la requête venue dans la variable R. Si la requête R vienne du client, le proxy testera si R est une demande d'inscription du client, dans ce cas il envoie directement une requête d'inscription au microservice d'inscription client pour s'inscrire. Sinon, si la requête R est une demande de service de fournisseur, le proxy extrait de R le nom de l'opération souhaitée

par le client, puis il sélectionne la plus proche instance de microserivce qui contient cette opération. Et par suite le proxy Gateway envoie la requête à l'instance sélectionnée.

Dans le cas où la requête provient du fournisseur, le proxy testera si R est une demande d'inscription du fournisseur. Si c'est le cas, il envoie directement la requête d'inscription au microservice inscription fournisseur pour s'inscrire, et si R est une requête dont le fournisseur souhaite inscrire un microservice, le proxy extrait le fichier wsdl de microservice et envoie une requête vers le microservice registre pour l'enregistrement.

Explication des fonctions utilisées dans l'algorithme

Annotation	**Objectif**
Operation_name(R)	permet d'extraire le nom d'opération à partir de la requête R du client.
Find_ip_by_operation_name (operation)	retourner une liste des adresse IP qui correspond au nom d'opération « operation ».
Min_distance (Liste_address_ip)	retourner la plus proche adresse IP par rapport au proxy de Middleware à partir de la liste des adresse IP « Liste_address_ip».
Execute (MS_inscription_client)	le proxy achemine la requête du client directement au microservice inscription client « MS_inscription_client »
Execute (R, MS_fournissuer,ipMin)	le proxy achemine la requête du client vers l'instance de microservice « MS_fournisseur » déployé sur l'adresse IP « ipMin » .
Execute (MS_inscription_fournisseur)	le proxy achemine la requête du fournisseur directement au microservice de l'inscription de fournisseur « MS_inscription_fournisseur ».
Upload (wsdl)	Télécharger contrat wsdl de service par le fournisseur.

Execute(MS_register)	le proxy achemine la requête de fournisseur directement au microservice register (MS_register).

Tableau 11 Explication des fonctions utilisées dans l'algorithme

On représente le cas où le client veut un micoservice de fournisseur donné par la figure 24 :

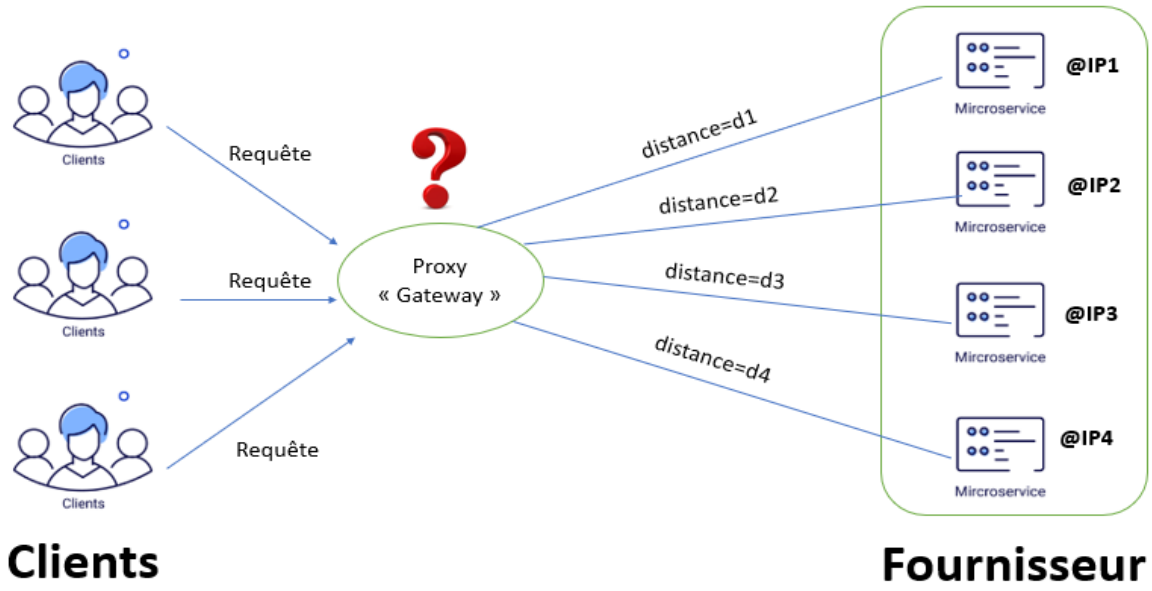

Figure 24: Microservice proxy (gateway)

Le proxy est en face des décisions de choix de la meilleure instance de microservice parmi les quatre instances de même microservice d'un seul fournisseur, sachant qu'elles sont déployées dans des zones différentes.

Solution = Min(di) tels que i ∈ {1,2,3,4}

IV.5.3.3 Service registre

Il s'agit d'un service qui enregistre des instances de microservice afin qu'elle puisse être découverte par d'autres microservices. Si on a besoin d'une forte demande de microservices, on doit lancer plusieurs instances de microservices pour répondre à la forte demande. Le microservice registre permet d'enregistrer les instances de microservice afin qu'elle puisse être découverte par d'autres microservices. Pour éviter un couplage fort entre les microservices, il est fortement recommandé d'utiliser un service de découverte capable de stocker les propriétés des différents services et d'éviter d'appeler directement le service. Au lieu de cela, le service de découverte fournit dynamiquement les informations requises. Cela garantit l'élasticité et le dynamisme inhérents aux architectures de microservices.

Dans la présente étude de notre application basée sur les microservices, les fournisseurs ont des nombreuses instances de plusieurs microservices déployés sur différents serveurs Cloud. En raison des changements dynamiques de l'emplacement de ces services, il est beaucoup plus difficile de gérer la découverte des services dans l'application client. A cet effet, nous avons réalisé un microservice de registration comme l'indique la figure 25, permettant de faciliter aux fournisseurs la registration de leurs microservices en suivant les étapes suivantes:

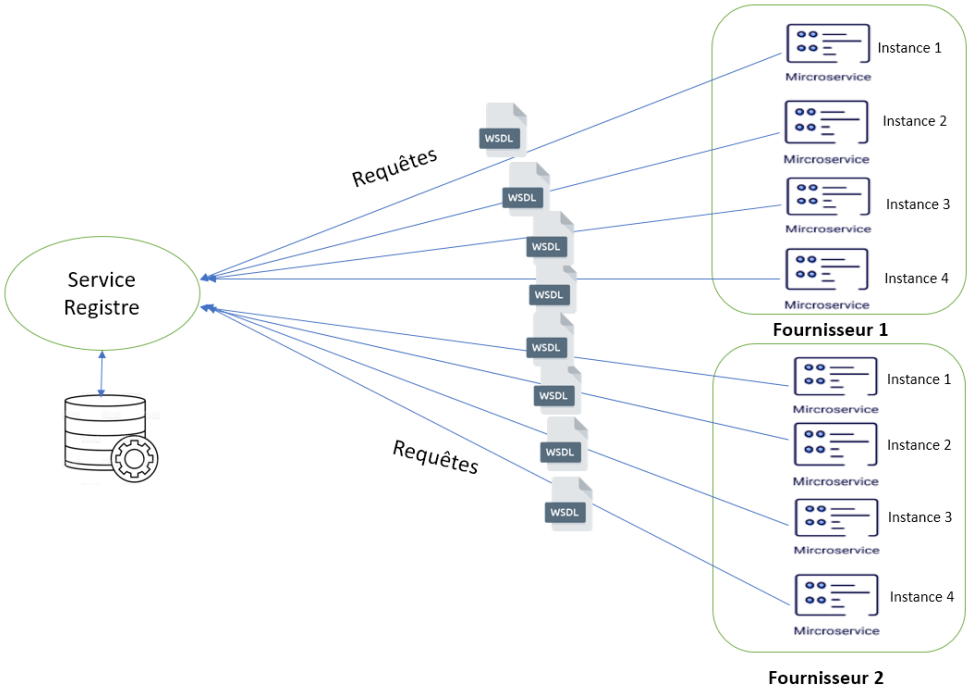

Figure 25 : Service Registre

En premier lieu, les fournisseurs registrent et s'authentifient dans le service d'authentification afin d'être abonnés dans le middleware. Comme nous avons précisé précédemment, les fournisseurs utilisent le standard WSDL (web service description langage). Ce Standard décrit les services de fournisseurs et offre un schéma formel de la description des instances, de tels façon que chaque instance à son propre fichier WSDL avec une description générale détaillée, c'est-à-dire un ensemble d'informations importantes afin que le service registre peut découvrir ces instances à partir de ces informations du WDSL.

Secondo, les fournisseurs déposent leurs fichiers du WDSL des instances pour que le service de registre découvre ces services. Ce service registre est relié avec une base de données alimentée par des informations sur la manière de distribuer les demandes des fournisseurs aux instances des microservices. A ce stade notre service vas parser dans le fichier WSDL de chaque instance et extraire les informations nécessaires (nom du service, @ip, numéro de port, localisation, nom d'opération) et le mettre dans une base de donnée pour que le service proxy l'utilise dans la sélection des meilleurs instances selon critère de distance.

IV.5.3.4 Service Registration et Authentification Client/ Fournisseur

Il s'agit d'un microservice qui fait la registration et Authentification du client et du fournisseur, qui offre la capacité au client de puisse consommer les microservices fournis par les fournisseurs. Ce microservice, nous permet de connecter des clients pour accéder aux middleware et aux fournisseurs de déposé ces instances. En effet, pour que les clients et les fournisseurs s'inscrire et s'abonnent dans le middleware, ils envoient des requêtes qui passe toujours en premier lieu par le proxy afin d'acheminer ces requêtes vers le microservice registration et authentification client / fournisseur comme l' illustre la figure 26, tels que chaque microservice registration et authentification client/fournisseur a sa propre base de donnée qui contient des informations de la registration de client /fournisseur (nom d'utilisateur (username)/nom de fournisseur, email ,password). Pour vérifier l'authentification des clients/fournisseurs, ces deux services comparent les informations saisies sur le formulaire avec les informations stocker dans la base de données et autorise l'accès à ses comptes.

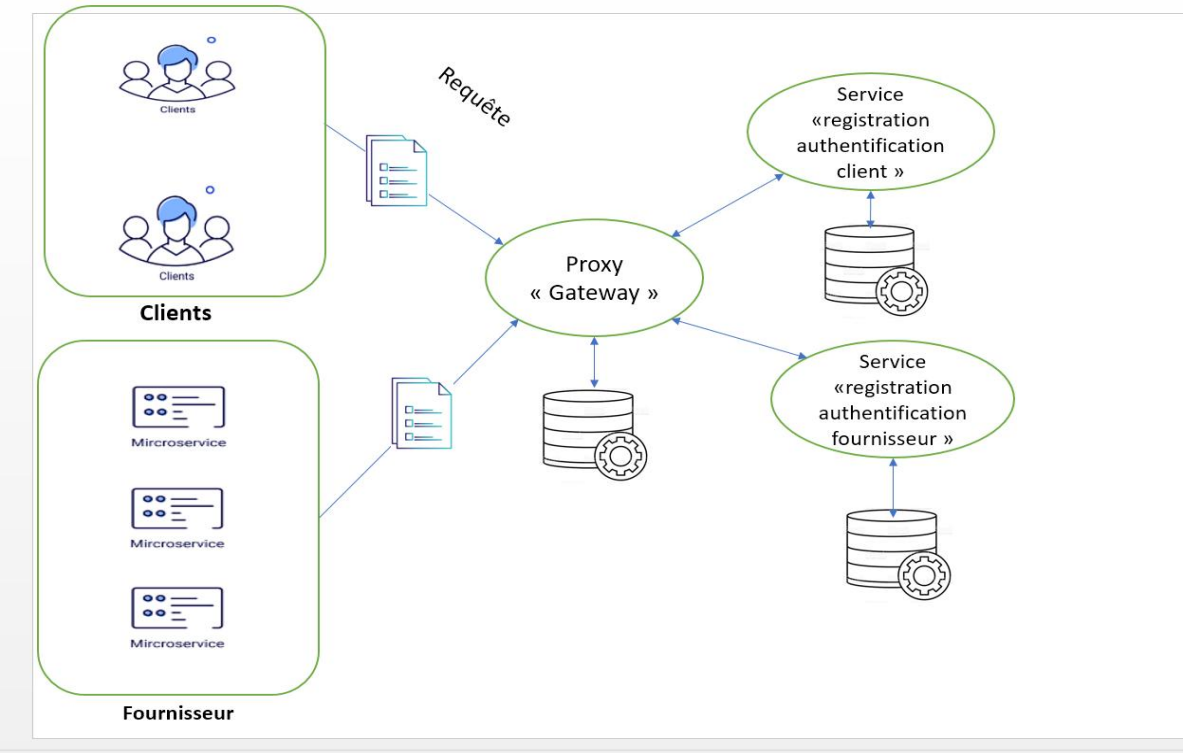

Figure 26 : Service Registration et Authentification Client/ Fournisseur

IV.6 Conclusion

Dans ce chapitre, nous avons présenté la conception de notre solution Middleware qui est basée sur l'architecture microservice. Nous avons commencé par introduire la démarche de travail suivie, puis nous avons fait une analyse des besoins pour identifier les acteurs du système et leurs actions respectives. Ensuite nous nous sommes lancées dans la présentation de notre solution en commençant par son architecture suivie de ses modules. Nous avons expliqué en détails chacun des modules de découverte et de la sélection à l'aide des exemples illustratifs. L'implémentation de la solution proposée sera décrite dans le chapitre suivant.

Chapitre V IMPLÉMENTATION ET SIMULATION DE LA SOLUTION PROPOSÉE

V.1 Introduction

L'implémentation consiste à implémenter la solution retenue au niveau de la phase de conception, c'est la phase au cours de laquelle les structures et les algorithmes définis pendant la conception sont traduits dans des langages de programmation et des bases de données.

Ce chapitre va comporter une partie pour les ressources matérielles et nous allons lister ensuite l'ensemble des outils de développement utilisés ainsi que des captures écrans qui montrent/présentent les interfaces de notre application et leurs codes sources.

V.2 Ressources matérielles

Afin de réaliser notre application, nous avons utilisés les ressources matérielles suivantes:

- Ordinateur portable MSI GL63 8RC
- Processeur Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30GHz
- Mémoire RAM 8.00 Go (7.89Go utilisable)
- Système d'exploitation 64 bits, processeur x64 - Windows 10 Professionnel

V.3 Outils et environnement de développement

Nous avons utilisés plusieurs outils logiciels et environnements de développement pour mieux réaliser notre projet. Ces différents outils nous permettent de créer tous les diagrammes présentés dans la partie conception et de mettre en œuvre divers programmes informatiques.

1-Docker

Docker est une plate-forme ouverte pour le développement, la livraison et l'exécution d'applications. Docker vous permet de séparer vos applications de votre infrastructure afin

que vous puissiez livrer rapidement des logiciels. Avec Docker, vous pouvez gérer votre infrastructure de la même manière que vous gérez vos applications[31].

2-Docker Compose

Docker Compose est un outil qui a été développé pour aider à définir et partager des applications multi-conteneurs. Avec Compose, nous pouvons créer un fichier YAML pour définir les services et avec une seule commande, nous pouvons tout faire tourner ou tout détruire. Le grand avantage de l'utilisation de Compose est que vous pouvez définir votre pile d'applications dans un fichier, la conserver à la racine de votre référentiel de projet (il est désormais contrôlé par version) et permettre facilement à quelqu'un d'autre de contribuer à votre projet [2] .

3-Docker Hub

Docker Hub est un service de référentiel hébergé fourni par Docker pour rechercher et partager des images de conteneurs avec votre équipe. Les fonctionnalités clés incluent: Référentiels privés: images de conteneur push et pull. Builds automatisés: créez automatiquement des images de conteneur à partir de GitHub et Bitbucket et transférez-les vers Docker Hub[32].

4-Java spring

Java Spring Framework (Spring Framework) est un cadre d'entreprise open source populaire pour la création d'applications autonomes de niveau production qui s'exécutent sur la machine virtuelle Java (JVM)[33].

[31] Docker, "Docker overview | Docker Documentation," 2020. https://docs.docker.com/get-started/overview/ (accessed Jun. 24, 2022).

[32] Docker, "Docker Hub - Docker," *What is docker Hub ?*, 2021. https://www.docker.com/products/docker-hub/ (accessed Jun. 24, 2022).

[33] IBM Cloud Education, "Spring boot," *What is spring boot ?*, 2020. https://www.docker.com/products/docker-hub/ (accessed Jun. 24, 2022).

5-Spring boot

Java Spring Boot (Spring Boot) est un outil qui accélère et facilite le développement d'applications Web et de microservices avec Spring Framework grâce à trois fonctionnalités principales : 1-Autoconfiguration, 2-Une approche avisée de la configuration, 3-La possibilité de créer des applications autonomes. Ces fonctionnalités fonctionnent ensemble pour vous fournir un outil qui vous permet de configurer une application basée sur Spring avec une configuration et une configuration minimale[4].

6-Spring Cloud

Spring Cloud fournit des outils permettant aux développeurs de créer rapidement certains des modèles courants dans les systèmes distribués (par exemple, la gestion de la configuration, la découverte de services, les disjoncteurs, le routage intelligent, le micro-proxy, le bus de contrôle, les jetons à usage unique, les verrous globaux, l'élection des dirigeants, les systèmes distribués). Sessions, état du cluster). La coordination des systèmes distribués conduit à des modèles de plaque de chaudière, et en utilisant Spring Cloud, les développeurs peuvent rapidement mettre en place des services et des applications qui implémentent ces modèles[34] .

7-Spring Data JPA

Spring Data JPA, qui fait partie de la grande famille Spring Data, facilite la mise en œuvre de référentiels basés sur JPA. Ce module traite de la prise en charge améliorée des couches d'accès aux données basées sur JPA. Spring Data JPA vise à améliorer considérablement la mise en œuvre des couches d'accès aux données en réduisant l'effort au montant réellement nécessaire. En tant que développeur, vous écrivez vos interfaces de référentiel, y compris les méthodes de recherche personnalisées, et Spring fournira automatiquement l'implémentation[35].

---

[34] Spring.io, "Spring Cloud," Jan. 03, 2021. https://spring.io/projects/spring-cloud (accessed Jun. 24, 2022).

[35] Spring.io, "Spring Data JPA ," 2020. https://spring.io/projects/spring-data-jpa (accessed Jun. 24, 2022).

8-Database H2 spring

H2 est une base de données intégrée, open source et en mémoire. C'est un système de gestion de base de données relationnelle écrit en Java. C'est une application client/serveur. Il est généralement utilisé dans les tests unitaires[36].

9-Postman

Postman est une plate-forme API pour la création et l'utilisation d'API. Postman simplifie chaque étape du cycle de vie des API et rationalise la collaboration afin que vous puissiez créer de meilleures API plus rapidement. Postman peut stocker et gérer les spécifications des API, la documentation, les recettes de flux de travail, les cas de test et les résultats, les métriques et tout ce qui concerne les API[37].

10-GitHub

GitHub est une plate-forme d'hébergement de code pour le contrôle de version et la collaboration. Il vous permet, à vous et à d'autres, de travailler ensemble sur des projets où que vous soyez[38].

11-IntelliJ IDEA

IntelliJ IDEA est un environnement de développement intégré (IDE) pour les langages JVM conçu pour maximiser la productivité des développeurs. Il effectue les tâches routinières et répétitives pour vous en fournissant une complétion de code intelligente, une analyse de code statique et des refactorisations, et vous permet de vous concentrer sur le bon côté du développement logiciel, ce qui le rend non seulement productif mais aussi agréable[39].

12-NodeJs

---

[36] Spring.io, "Spring Data JPA ," 2020. https://spring.io/projects/spring-data-jpa (accessed Jun. 24, 2022).

[37] Postman, "Postman API Platform," *2020*. https://www.postman.com/product/what-is-postman/ (accessed Jun. 24, 2022)

[38] Github, "GitHub Introduction and definition," 2021. https://docs.github.com/en/get-started/quickstart/hello-world (accessed Jun. 24, 2022).

[39] M.Jacks, "IntelliJ IDEA overview | IntelliJ IDEA," May 10, 2022. https://www.jetbrains.com/help/idea/discover-intellij-idea.html (accessed Jun. 24, 2022).

Selon NodeJs « Node.js est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles hautement concurrentes qui doivent pouvoir monter en charge. Elle utilise la machine virtuelle V8, la librairie libuv pour sa boucle d'évènements, et implémente sous licence MIT les spécifications CommonJS »[40].

13-Angular

Angular est une plateforme de développement, basée sur TypeScript. En tant que plate-forme, Angular comprend [41]:

-Un cadre basé sur des composants pour créer des applications Web évolutives.

-Une collection de bibliothèques bien intégrées qui couvrent une grande variété de fonctionnalités, y compris le routage, la gestion des formulaires, la communication client-serveur, etc. Une suite d'outils de développement pour vous aider à développer, construire, tester et mettre à jour votre code

14-Bootstrap

Bootstrap est le framework CSS le plus populaire pour développer des sites Web réactifs et mobiles. Bootstrap utilise SAAS pour une architecture modulaire et personnalisable. Importez uniquement les composants dont vous avez besoin, activez les options globales telles que les dégradés et les ombres, et écrivez votre propre CSS avec nos variables, cartes, fonctions et mixins [42]

15-MySQL

MySQL Database Service est un service de base de données entièrement géré pour déployer des applications natives du cloud en utilisant la base de données open source la plus populaire au monde. Ce service est développé, géré et supporté à 100% par l'équipe de MySQL[43].

[40] Node.org, "Node.js definition," Jun. 21, 2022. https://nodejs.org/fr/ (accessed Jun. 24, 2022).

[41] Angular.io, "What is angular front end ," 2022. https://angular.io/guide/what-is-angular (accessed Jun. 24, 2022)

[42] Bootstrap, "Bootstrap · The most popular HTML, CSS, and JS library in the world.," 2021. https://getbootstrap.com/ (accessed Jun. 24, 2022).

[43] K.Hellal, "MySQL," 2020. https://www.mysql.com/fr/(accessed Jun. 11, 2022)

16-JSON

JSON (JavaScript Object Notation) est un format d'échange de données léger. Il est facile pour les humains de lire et d'écrire. JSON est un format de texte complètement indépendant du langage mais utilise des conventions familières aux programmeurs de la famille C de

langages, y compris C, C++, C#, Java, JavaScript, Perl, Python et bien d'autres. Ces propriétés font de JSON un langage d'échange de données idéal[44].

17-API Rest

Une API REST (également appelée API RESTful) est une interface de programmation d'application (API ou API web) qui respecte les contraintes du style d'architecture REST et permet d'interagir avec les services web RESTful. L'architecture REST (Representational State Transfer) a été créée par l'informaticien Roy Fielding.[45]

18-WebStorm

WebStorm est un environnement de développement intégré pour le codage en JavaScript et ses technologies associées, notamment TypeScript, React, Vue, Angular, Node.js, HTML et les feuilles de style. Tout comme IntelliJ IDEA et les autres IDE JetBrains, WebStorm rend votre expérience de développement plus agréable, automatisant le travail de routine et vous aidant à gérer facilement les tâches complexes.[46]

19-phpMyAdmin

phpMyAdmin est un outil logiciel gratuit écrit en PHP, destiné à gérer l'administration de MySQL sur le Web. phpMyAdmin prend en charge un large éventail d'opérations sur MySQL et MariaDB. Les opérations fréquemment utilisées (gestion des bases de données, des tables, des colonnes, des relations, des index, des utilisateurs, des autorisations, etc.)

[44] json.org, "JSON," 2018. https://www.json.org/json-en.html (accessed Jun. 24, 2022)

[45] Redhat, "API rest," 2018. https://www.redhat.com/fr/topics/api (accessed Jun. 24, 2022)

[46] S.cxiona, "Web storm jetBrains." https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html (accessed Jun. 24, 2022)

peuvent être effectuées via l'interface utilisateur, tandis que vous avez toujours la possibilité d'exécuter directement n'importe quelle instruction SQL[47].

20-XAMPP

Selon Apache[85]« XAMPP est l'environnement de développement PHP le plus populaire .XAMPP est une distribution Apache entièrement gratuite et facile à installer contenant MySQL, PHP et Perl. Le paquetage open source XAMPP a été mis au point pour être incroyablement facile à installer et à utiliser. »[48]

V.4 L'implémentation de l'application

Pour implémenter notre application web, nous avons utilisé IntelliJ IDEA (backend) et WebStorm (frontend) comme des environnements de développement.

Notre application est réalisée en JavaEE qui utilise le framwork Springboot pour le coté backend tels que nos fonctionnalités sont codées en Java . Pour les interfaces graphiques (partie UI) c'est-à-dire coté Frontend, nous avons utilisé Angular qui est un Framework JavaScript basé sur système des components et qui est l'un des Framework web les plus utilisés par les développeurs,

V.4.1 Concepts de Base spring boot

V.4.1.1 Dépendances et annotations utilisés

D'abord pour créer un projet spring, Il faut aller à spring initializr [https ://start.spring.io/] pour télécharger le framework du projet spring boot. Ensuite nous ajoutons les dépendances nécessaires

- **Les dépendances**

Spring Web : Permet de créer des applications Web, y compris RESTful, à l'aide de Spring MVC. Il utilise Apache Tomcat comme conteneur intégré par défaut

[47] phpmyadmin.net, "phpMyAdmin," 2020. https://www.phpmyadmin.net/ (accessed Jun. 24, 2022).

[48] apachefriends.org, "XAMPP ," 2021. https://www.apachefriends.org/fr/index.html (accessed Jun. 24, 2022).

Spring Data JPA : Ce module traite de la prise en charge améliorée des couches d'accès aux données basées sur JPA. Il facilite la création d'applications alimentées par Spring qui utilisent des technologies d'accès aux données.

REST : Repositories Exposition des référentiels Spring Data sur REST via Spring Data REST.

H2 : Base de donnée mémoire pour les tests préliminaire .

MySQL Driver : Dépendance vers le pilote de base de données JDBC

Lombok : Bibliothèque d'annotations Java qui permet de réduire le code passe-partout comme les constructeurs, les getters et setters.

Par exemple:

-**@AllArgsConstructor** generates a constructorrequiring argument for everyfield in the annotated class

-**@NoArgsConstructor** generates a constructorwith no parameter

-**@Data** est une annotation de raccourci pratique qui regroupe les fonctionnalités de @ToString, @EqualsAndHashCode, @Getter / @Setter et @RequiredArgsConstructor

-**@Entity** nous indique que cette classe est une classe persistante. Elle peut prendre un attribut name , qui fixe le nom de cette entité.

V.4.1.2 Implémentation des entités

Dans notre implémentation, nous avons utilisé les principes du mapping objet/relationnel (O/R) à travers JPA (Java Persistence API). JPA Permet de proposer un niveau d'abstraction supérieur à l'utilisation de JDBC : Le mapping peut assurer la transformation des objets vers la base de données. Nous avons utilisé l'annotation Lombok dans la lecture et les mises à jour des données qui permettre de simplifier notre code.

V.4.1.3 Implémentation des Contrôleurs

Dans cette partie, nous discuterons un peu sur l'implémentation des contrôleurs. Il y a principalement deux contrôleurs utilisés dans spring boot, « contrôleur » avec l'annotation @controller et le « RestController» avec l'annotation @restcontroller. La principale différence entre le @restcontroller et le @controller est que @restcontroller est la combinaison entre les deux annotations @controller et @ResponseBody tel que @ReponseBody est une annotation Spring qui lie une valeur de retour de méthode au corps de la réponse Web [86].

Spring MVC Spring MVC est considéré comme le framework Web basé sur le contrôleur , la vue et le modèle sous le framework Spring [87]. Il facilite grandement l'écriture des classes de contrôleur et de méthodes pour les gestionnaires de requêtes. On Ajoute simplement des annotations comme @ GetMapping, @ PostMapping, @ PutMapping, @ DeleteMapping, @ PatchMapping, @RequestMapping

Le tableau ci-dessous représente fonctionnement de ces annotations.

Annotation	Type	Objectif
@GetMapping	**GET**	Permet de récupérer des données
@PostMapping	**POST**	Pour l'envoi de données. Cela sera utilisé par exemple pour créer un nouvel élément.
@PatchMapping	**PATCH**	pour effectuer une mise à jour l'élément envoyé.
@PutMapping	**PUT**	Pour le remplacement complet de l'élément envoyé.
@DeleteMapping	**DELETE**	Pour la suppression de l'élément envoyé.
@RequestMapping	**/**	mapper toutes les URL de requête http entrantes aux méthodes de contrôleur correspondantes.

Tableau 12: fonctionnement des annotations de controlleur

Exemple d'implémentation de contrôleur de microservice Registre

Le figure ci-dessous représente Le contrôleur de microservice registre dont on a utilisé annotation @PostMapping pour télécharger le fichier WSDL.

Figure 27: Controller de microservice registre

V.4.1.4 La couche d'accès aux données

Les implémentations de persistance doivent être séparées au niveau de la couche d'accès aux données. DAO signifie Data Access Object. Les classes DAO sont généralement responsables de deux concepts. Encapsule les détails de la couche persistante et expose une seule interface CRUD d'entité [88], ainsi qu'il garde la couche métier indépendante des sources de données .

V.5 Architecture de l'application

Dans cette partie nous présentons le code métier de notre application, tel que nous expliquons en détail le fonctionnement de chaque Microservices d'application.

Les microservices implémentés pour notre projet sont : Microservices config , Microservices Proxy , Microservices Registre, Microservices Registration et authentification fournisseur client .

V.5.1 Implémentation du microservice config

Pour ce microservice, Spring Boot fournit une dépendance Spring Cloud Config. Il gère la gestion centralisée de la configuration via le Git.La partie principale de l'application est une classe de configuration, qui récupère toute la configuration requise via l'annotation de configuration automatique @EnableConfigServer comme illustrer dans la figure 28

```
package sid.org.serviceconfig;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@EnableConfigServer
@SpringBootApplication
public class ServiceConfigApplication {

    public static void main(String[] args) { SpringApplication.run(ServiceConfigApplication.class, args); }

}
```

Figure 28: La Class main de microservice config

les paramètres de configuration qui sont centralisés dans le microservice config est illustré dans la Figure suivant

```
spring.application.name=configService
server.port=8085
#spring.cloud.config.server.git.uri=file://${user.home}/cloud-conf
spring.cloud.config.server.git.uri=https://github.com/Zinomenadi/cloud-conf
```

Figure 29 Fichier de configuration de service config

Notez également que le fichier de configuration du microservice registre et microservice proxy sont externalisés respectivement dans registre.properties et proxy.properties. Ces deux derniers sont dans le repository git distant, comme indiqué dans la Figure 30

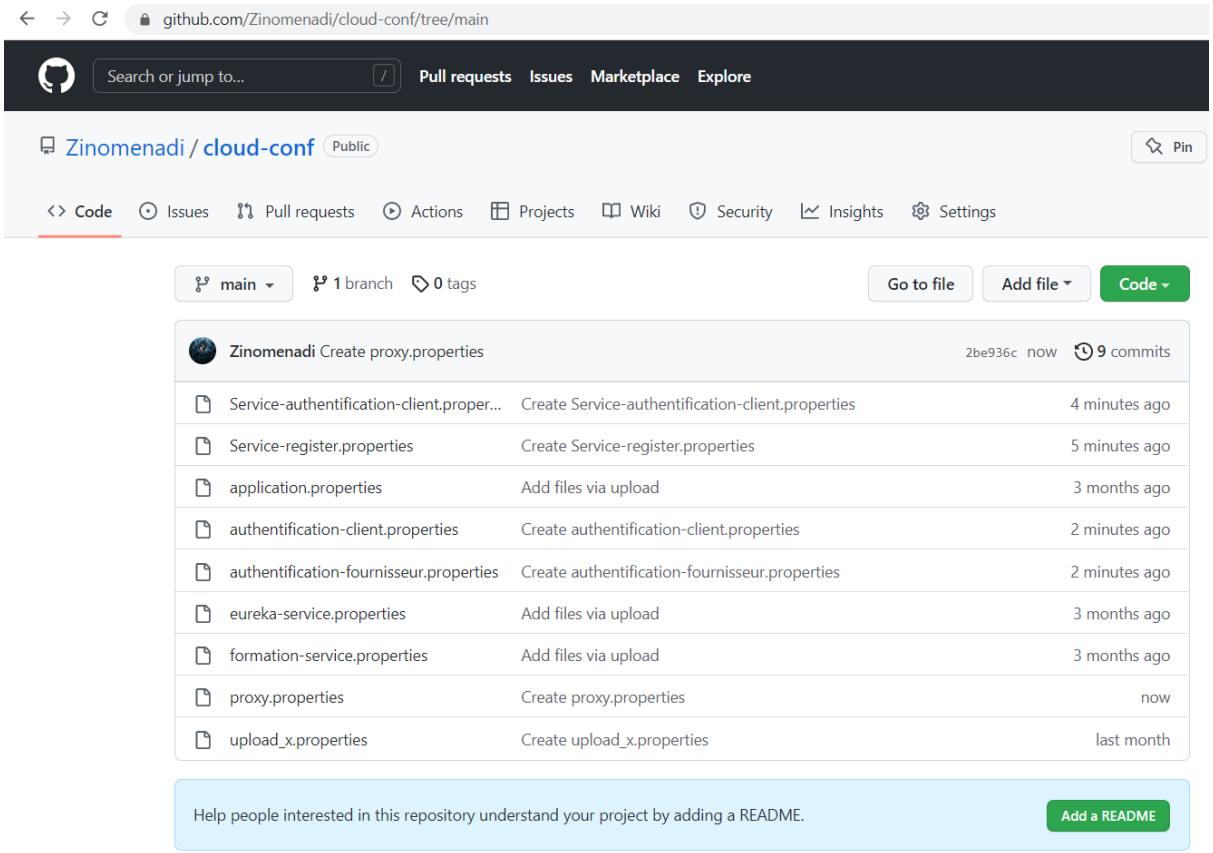

Figure 30: Configuration des microservices dans github

V.5.2 Implémentation de microservice proxy

Le proxy ou la Gateway représente le point d'entrée unique de notre Middleware. Cela veut dire que toutes les requêtes, que ce soit du fournisseur ou du client passent premièrement par le proxy, puis ce dernier exécute l'algorithme « gateway_algo » expliqué dans la partie conception, afin d'acheminer la requête vers la bonne destination. Le proxy dispose d'une classe nommée proxyController. Cette classe joue le rôle du api REST à travers l'annotation @RestController qui permet de créer des contrôleurs Restful. Cette api REST contient un ensemble de fonctions qui permettent de structurer la communication entre le microservice proxy et les autres microservices .les figures 31, 32 , 33 , 34 montrent toutes les fonctions définies dans le microservice proxy.

Figure 31 Fonction qui reçoit et traite la requête d'inscription du client

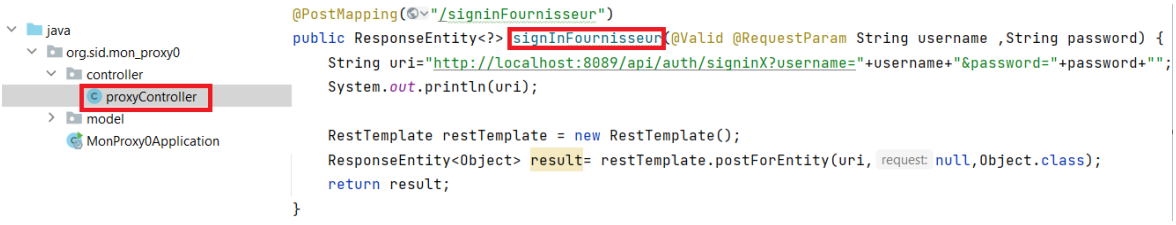

Figure 32 Fonction qui reçoit et traite la requête d'inscription de fournisseur

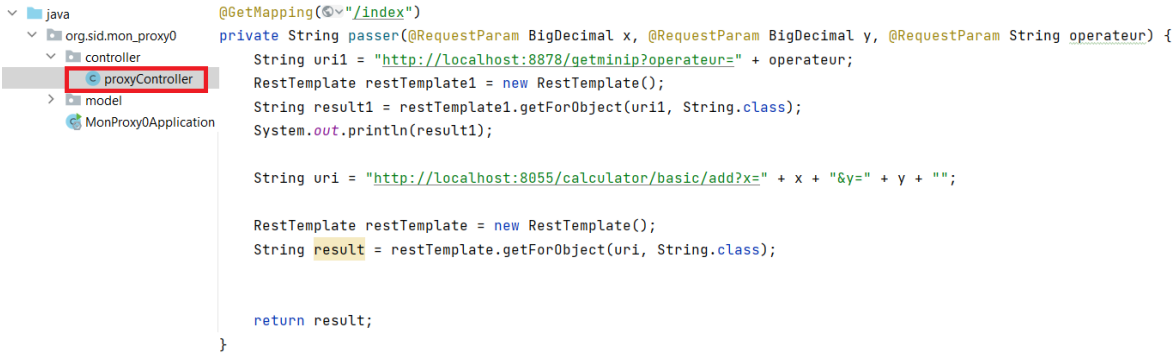

Figure 33 Fonction qui traite la requête de demande de consommation d'un service du fournisseur

Figure 34 Fonctions qui traitent les requêtes « upload » , « download » des document WSDL

V.5.3 Implémentation du microservice Registration et Authentification

Nous avons réalisé deux microservices, dédiés pour les fournisseurs et les clients, qui permettent de faire l'inscription et l'authentification sur le Middleware. Avec ces microservices, nous pouvons accéder aux comptes des clients et fournisseurs pour bénéficier de toutes les fonctionnalités développées. Dans cette partie, on expliquera uniquement le fonctionnement du microservice du fournisseur, qui a la même fonction que le microservice

du client. Le microservice Registration et Authentification est relié avec une base des données dédiées pour stocker les informations nécessaires des fournisseurs (username, email, password). La configuration de microservice Registration et Authentification est présente dans le server de fichier relier avec microservice config .la figure 35 montre la configuration de microservice registration et authentification de fournisseur.

```
server.port=8089

spring.datasource.url=jdbc:mysql://localhost:3308/ms-fournisseur?JDBCCompliant=false&serverTimezone=UTC
spring.datasource.username= root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto= create
```

Figure 35: Application properties de microservice registration et authentification

V.5.3.1 Partie Inscription :

- Lorsque le fournisseur rempli le formulaire d'inscription présenté dans la partie des interface,il doit appuyer sur le button Sign Up pour s'inscrire, le coté frontend va réagi en exécutant la fonction onSubmit(). Ceci permet de passer vers la classe services qui contient la fonction registrer. Cette fonction envoie les informations saisies par le fournisseur sous forme des paramètres avec la méthode post dans une requête qui contient le lien vers le coté backend comme l'illustre dans la figure 36

```
const AUTH_API = 'http://localhost:8089/api/auth/';

const httpOptions = {
  headers: new HttpHeaders( headers: { 'Content-Type': 'application/json' })
};

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  constructor(private http: HttpClient) { }

  login(username: string, password: string): Observable<any> {
    return this.http.post( url: AUTH_API + 'signin', body: {
      username,
      password
    }, httpOptions);
  }

  register(username: string, email: string, password: string): Observable<any> {
    return this.http.post( url: AUTH_API + 'signup', body: {
      username,
      email,
      password
    }, httpOptions);
  }
}
```

Le lien vers l'API

La fonction registre

Figure 36: La class service de component registre

Lorsque la requête est reçu par API AuthControlleur, elle va automatiquement être exécuté par la méthode RegisterUser() pour traiter la requête envoyée par le frontend comme illustré dans la figure 37.

Figure 37: La class authController

Ensuite, la méthode citée ci-dessus enregistre toutes les informations des fournisseurs (username, email, password) dans leurs base de données. Nous avons utilisés la méthode JWT pour sécuriser les mots de passes des comptes des fournisseurs comme l'illustre la figure 38.

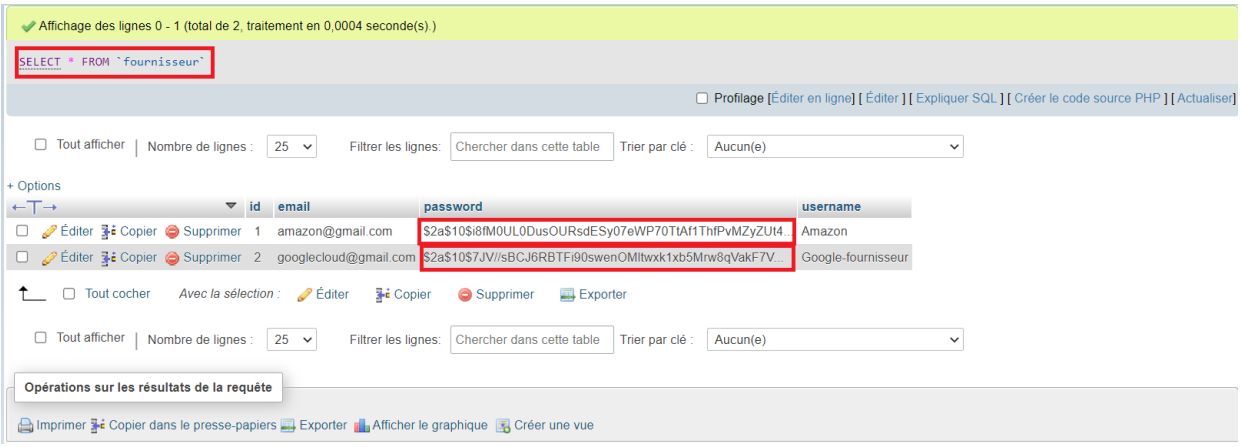

Figure 38: Base de donnée de fournisseur

V.5.3.2 Partie Authentification

Lorsque le fournisseur click sur le bouton Login, le coté frontend vas réagi en exécutant la fonction onSubmit () qui permet de passer à la classe service. Cette classe contient la fonction login () qui va envoyer les informations saisies par le fournisseur sous forme des paramètres en utilisant la méthode post avec une requête contenant le lien vers le coté backend comme l'indique la figure 39

Figure 39: La méthode login

-Ensuite, lorsque la requête reçue par le backend, il exécute la méthode Sign In qui permettre de vérifier si les informations de fournisseur (username, password) est compatible avec les informations de la base de donnée (Fig.40).

```java
@PostMapping("/signin")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {

    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));

    SecurityContextHolder.getContext().setAuthentication(authentication);
    String jwt = jwtUtils.generateJwtToken(authentication);

    UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();

    return ResponseEntity.ok(new JwtResponse(jwt,
            userDetails.getId(),
            userDetails.getUsername(),
            userDetails.getEmail()
    ));
}
```

Figure 40: La méthode Sign in

V.5.4 Implémentation de microservice registre

Ce microservice est spécialisé dans la découverte des services de fournisseurs, il permet au fournisseur l'enregistrement des instances de ses microservices en vue d'être découverts par les clients . La découvrabilité a été traitée avec les contrats WSDL de chaque microservice, de sorte que le microserivce « register » donne la possibilité au fournisseur de faire entrer le WSDL.

-Le fournisseur, choisit ensuite le fichier WSDL correspondant à son microservice, il click après sur le button upload pour télécharger le fichier wsdl dans un serveur de fichier situé dans le même serveur avec le microserivce « register ». le dossier nommé uploads repésente le serveur de fichiers des wsdl comme illustré dans figure 41.

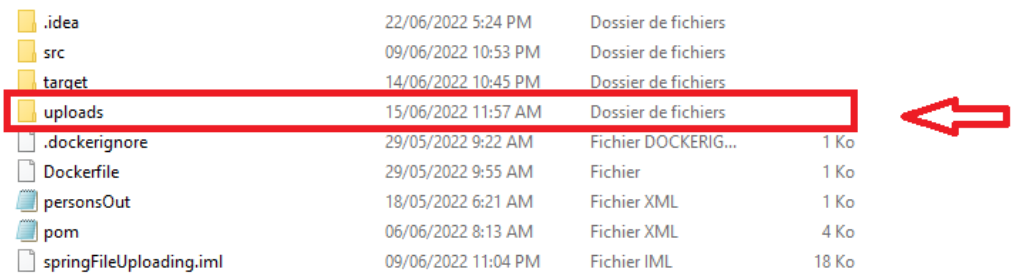

Figure 41: serveur des fichiers WSDL

-le microserivce « registre » est lié à une base de données dédié de stocker les informations nécessaires extraites des WSDL des microserivces du fournisseur comme illustré dans figure 42

	id	address_ip	location	nom_ms	operation	port
Éditer Copier Supprimer	2	41.120.144.25	http://41.120.144.25:7000/	add_ms	add	7000
Éditer Copier Supprimer	3	50.50.144.25	http://50.50.144.25:6200/	add_ms	add	6200
Éditer Copier Supprimer	4	130.10.10.10	http://130.10.10.10:6000/	add_ms	add	6000

Figure 42: Table de base de données WSDL

-La récupération des champs pour le remplissage de cette table de la base de données, a été réalisée à travers deux bibliothèques de java « DOMparser » et « SAXparser ». Ces deux bibliothèques sont utiles pour l'analyse du code XML (code de WSDL) et extraient les informations qu'on a besoin.

Nous avons utilisé plusieurs méthodes fournies par ces deux bibliothèques pour extraire des informations inclues dans les balises de fichier WSDL . Les méthodes sont exprimées dans la figure 43 .

```
File xmlFile = new File(filePath);
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder;
try {
  dBuilder = dbFactory.newDocumentBuilder();
  Document doc = dBuilder.parse(xmlFile);
  doc.getDocumentElement().normalize();
  System.out.println("Root element :" + doc.getDocumentElement().getNodeName());
  NodeList nodeList = doc.getElementsByTagName( s: "wsdl:service"); //

  NodeList nodeList2=doc.getElementsByTagName( s: "wsdl:portType");


NodeList children = doc.getElementsByTagName( s: "wsdl:port").item( i: 0).getChildNodes();
  System.out.println(children);
  for(int i=0;i<children.getLength();i++) {
    if(children.item(i).getNodeType()== Node.ELEMENT_NODE) {
      Element child = (Element)children.item(i);
      System.out.println(child.getAttribute( s: "location")); //
    }
  }
```

Figure 43 Analyse de code XML (WSDL)

En premier lieu, on a créé une instance de type DocumentBuilderFactory puis on a créé notre document xml nommé doc, qui est de type Document. Doc est le XmlFile qui représente le

fichier WSDL de microservice . Après une normalisation de doc, on a créé un objet nommé nodeList de type NodeList qui contient la liste des balises inclus dans la balise wsdl:service .

Ensuite, on a créé un autre objet de type NodeList contenant la liste des balises inclus dans la balise wsdl :portType. Et puis, c'est grâce à l'utilisation de quelques méthodes fournies par la bibliothèque parser par ex (getAttribute , getChildNodes ..etc), on a pu accéder aux sous balises internes des deux balises (wsdl :service et wsdl :portType) .

Voila des illustrations qui montrent toutes les méthodes utilisées avec le rôle de chaque méthode (Fig 44)

Méthode	Rôle
String getAttribute(String)	Renvoyer la valeur de l'attribut dont le nom est fourni en paramètre
removeAttribut(String)	Supprimer l'attribut dont le nom est fourni en paramètre
setAttribut(String, String)	Modifier ou créer un attribut dont le nom est fourni en premier paramètre et la valeur en second
String getTagName()	Renvoyer le nom du tag
Attr getAttributeNode(String)	Renvoyer un objet de type Attr qui encapsule l'attribut dont le nom est fourni en paramètre
Attr removeAttributeNode(Attr)	Supprimer l'attribut fourni en paramètre
Attr setAttributNode(Attr)	Modifier ou créer un attribut
NodeList getElementsByTagName(String)	Renvoyer une liste des noeuds enfants dont le nom correspond au paramètre fourni

Figure 44 : Méthodes fournies par DOM PARSER

Nous avons lié ce microservice avec la base de données à travers une configuration spécifiée au niveau de fichier bootstrap.properties de microservice comme illustré dans la Figure 45.

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_wsdl
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=create
```

Figure 45: Liaison de microserivce registre avec la base de donnée db_wsdl

V.6 Présentation des interfaces principales de l'application

Dans cette partie, nous faisons une présentation des fonctionnalités de notre Platform dont en montrant toutes les interfaces graphiques de l'application. Les interfaces graphiques sont développées avec Angular qui est un Framework JavaScript. C'est un framework côté client, open source, basé sur TypeScript, et co-dirigé par l'équipe du projet « Angular » à Google et par une communauté de particuliers et de sociétés. Notre application est responsive donc s'adapte aux différents types d'écran.

La réalisation de notre solution a pris la forme d'une application Web pour laquelle nous avons choisi le nom "QUICK". Nous avons aussi conçu un logo simple est significatif pour représenter notre application. Notre logo apparaît dans la figure suivante :

Figure 46 : Logo de CloubBroker Quick.

V.6.1 Interface d'accueil

Au début, lorsqu'un utilisateur ou un fournisseur tente d'accéder à notre application, via son navigateur Web, il atterrit dans la page de présentation du site. Comme représenté ci-dessous :

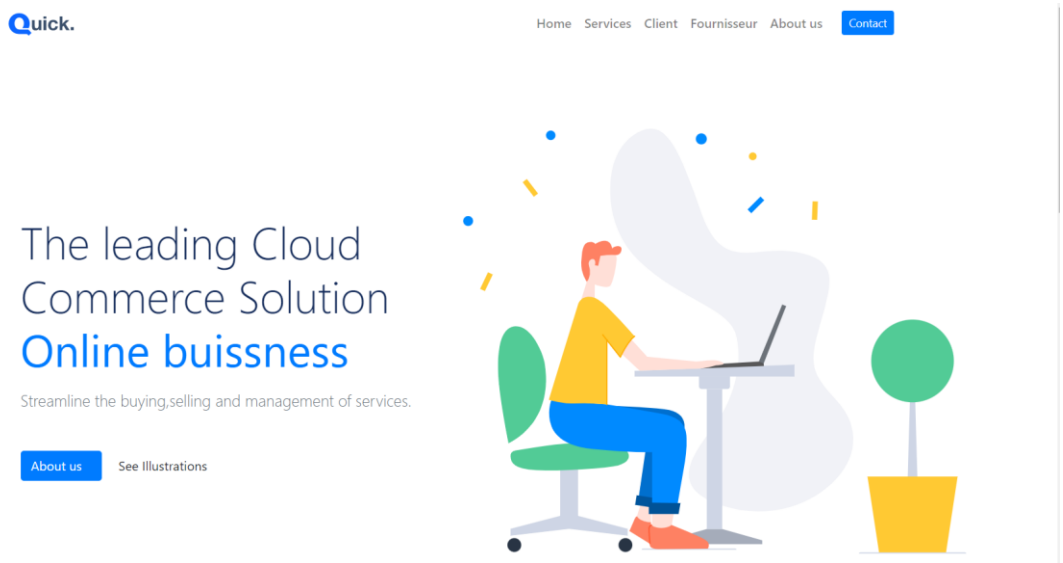

Figure 47 : Page de présentation du site

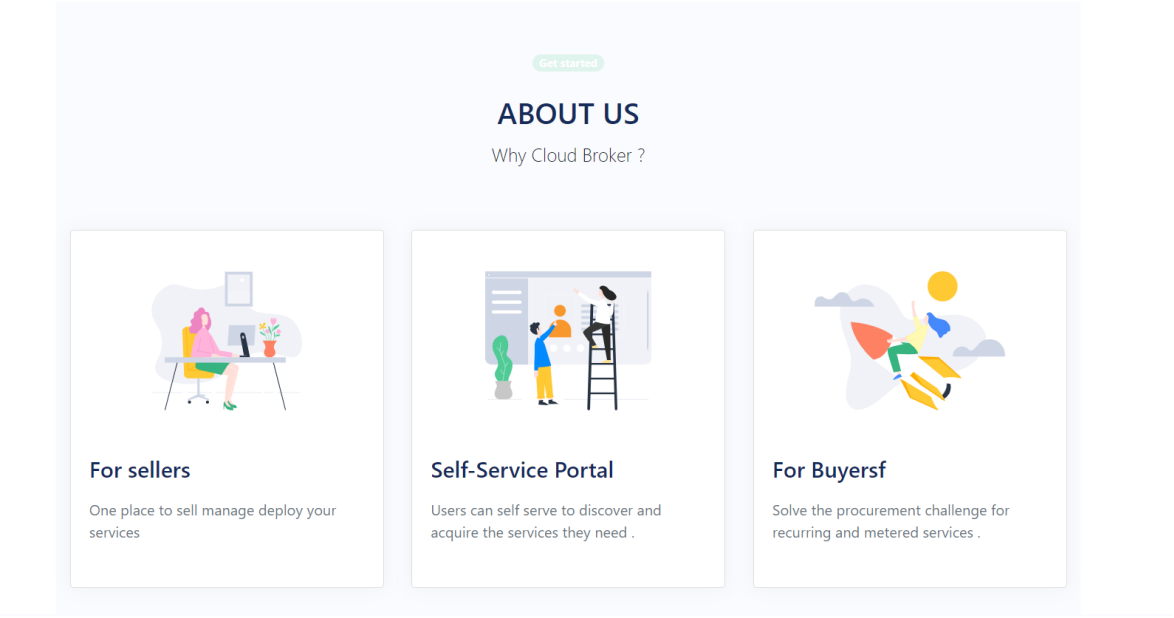

Figure 48: Page de présentation du site (suite)

V.6.2 L'interface d'inscription

L'utilisateur peut se connecter s'il a déjà un compte. Sinon, il peut créer un compte en cliquant sur le bouton Login . Ce bouton va le redirigé vers l'interface d'inscription où il doit introduire ses informations comme le montre la figure 49. Comme représenté ci-dessous :

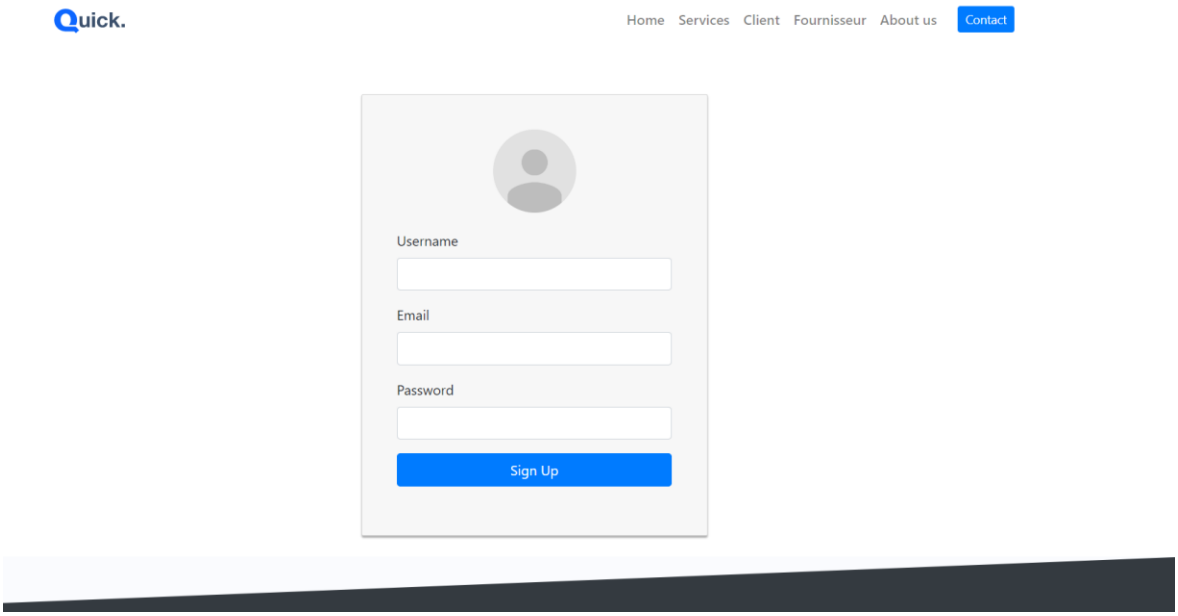

Figure 49 Interface d'inscription de fournisseur

V.6.3 L'interface de connexion

Cette interface permet aux différents utilisateurs de la Platform de se connecter. Si les informations (username et password) saisies dans l'interface sont compatibles avec les informations de base de données, il accède vers l'interface de fournisseur et affiche la page qui comporte un message de succès, sinon un message d'erreur lui est envoyé. La figure 50 illustre la page de connexion de notre application.

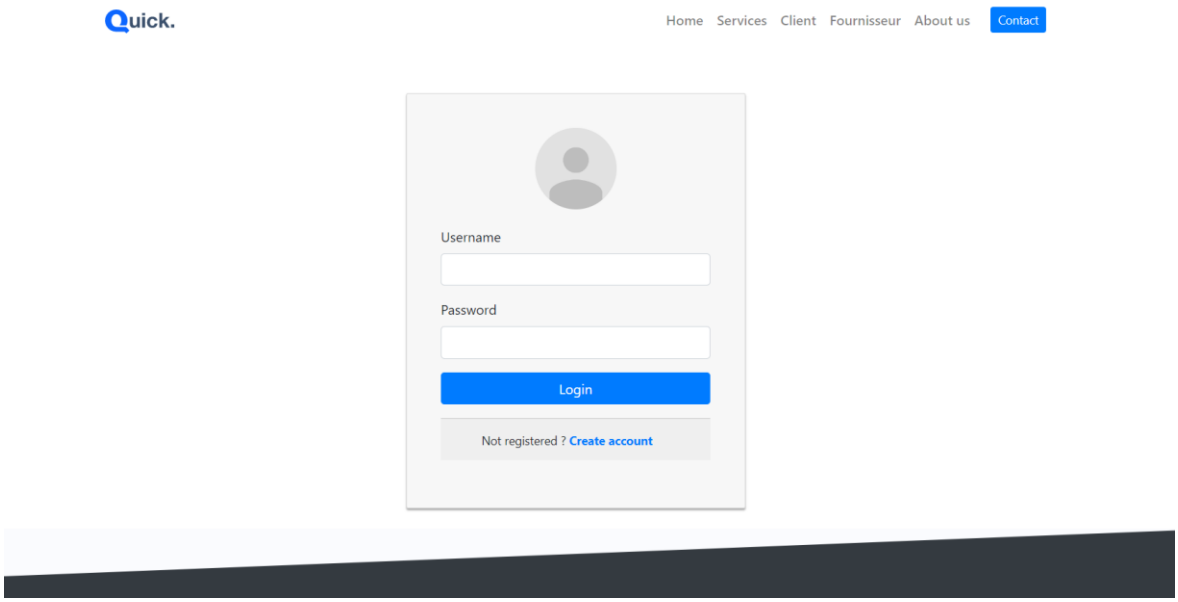

Figure 50 : Interface de connexion

V.7 Espace de fournisseur

V.7.1 Interface d'accueil de fournisseur

Une fois que le fournisseur se connecte, il est dirigé directement vers sa page d'accueil où il peut choisir une tâche à effectuer, il peut consulter ses services, ajouter un service par son WSDL , supprimer un service ,télécharger le fichier WSDL des services fournis comme il peut même voir les statistiques des services. Comme représenté ci-dessous :

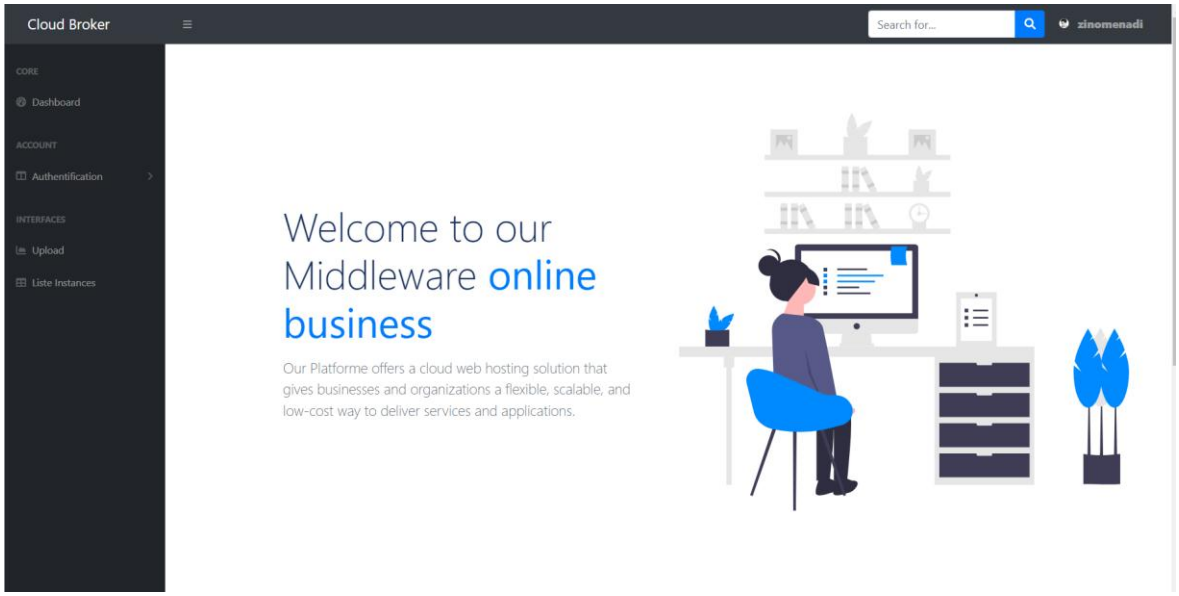

Figure 51 : Page d'accueil fournisseur

V.7.2 Interface d'ajout d'un microservice

Cette interface permet au fournisseur l'enregistrement des instances de ses microservices en vue d'être découverts par les clients. Il clique sur « Choisir un fichier » puis il sélectionne le fichier WSDL qui correspond à le microservice qu'il veut fournir comme illustrer dans la figure 52

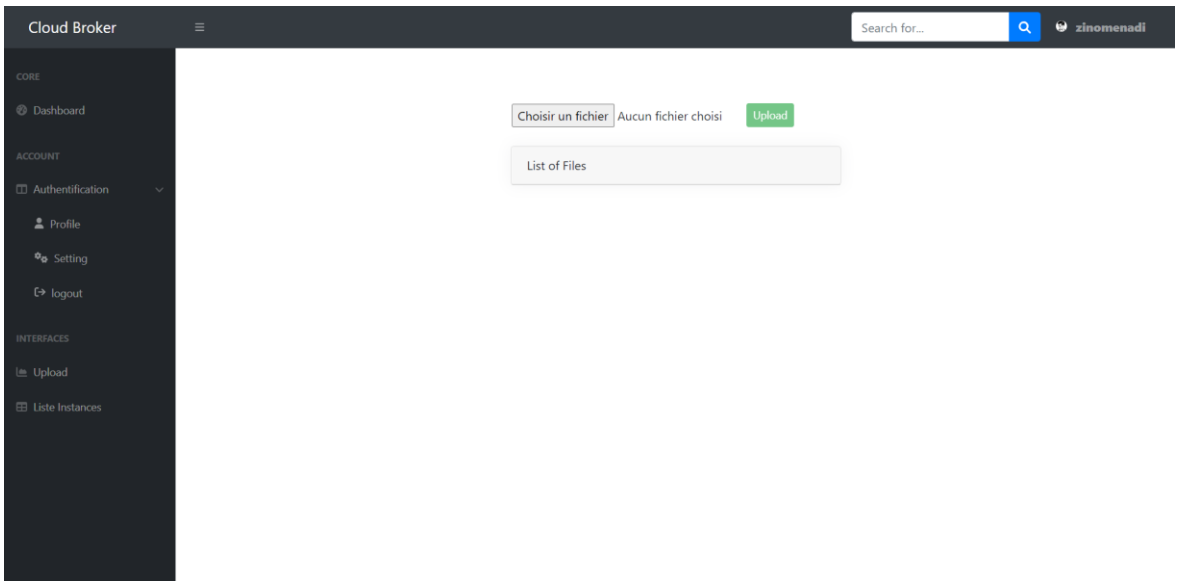

Figure 52 interface d'inscrire un microservice

V.7.3 Interface de liste des microservices

Cette interface permet aux fournisseurs de consulter toutes les informations des instances de ses microservices. Le fournisseur a la possibilité de téléchargé le contrat wsdl et voir le statut de son microservice, il peut aussi supprimer un microservice et voir les statistiques comme illustré dans l'interface de Figure 53.

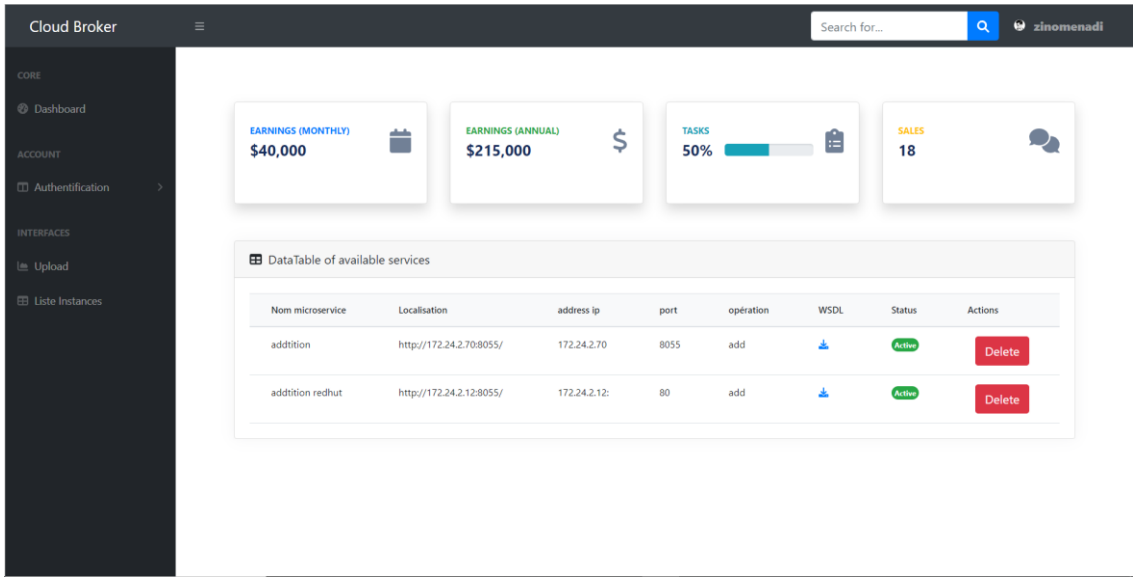

Figure 53: Liste des microservices de fournisseur

V.8 Espace client

Une fois que le client se connecte dans middleware , il est dirigé directement vers la page qui contient la liste des services fournis par les fournisseurs, donc il choisira le service dont il a besoin comme l'illustre dans la figure 54 ci-dessous :

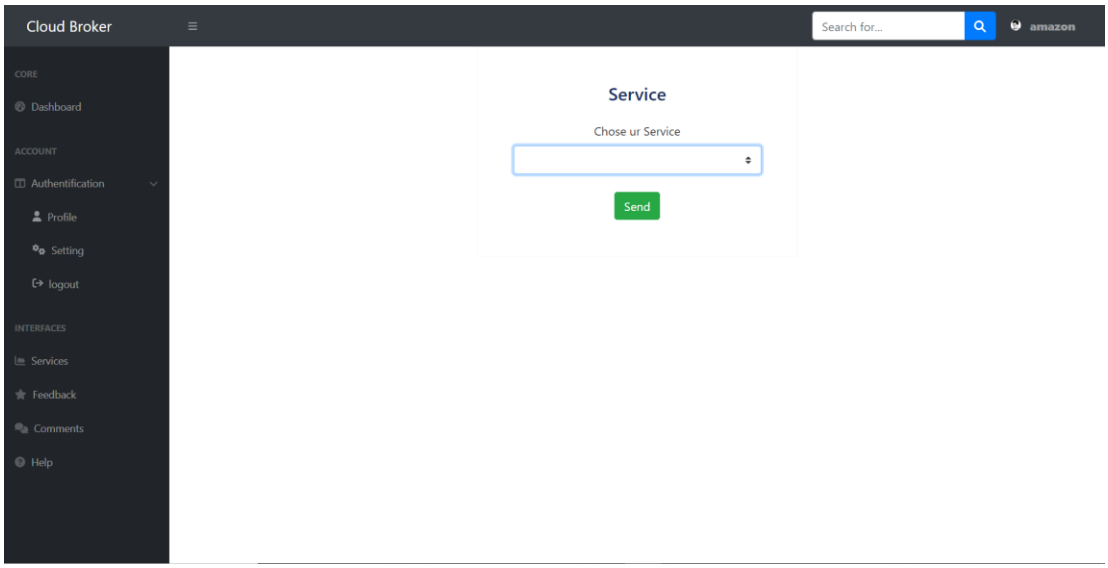

Figure 54: Interface de client pour choisi le service

Cette interface permet aux clients de consulter les instances de microservices dont il a besoin qui ont fournis par différents fournisseurs comme l'illustre dans la figure 55 ci-dessous

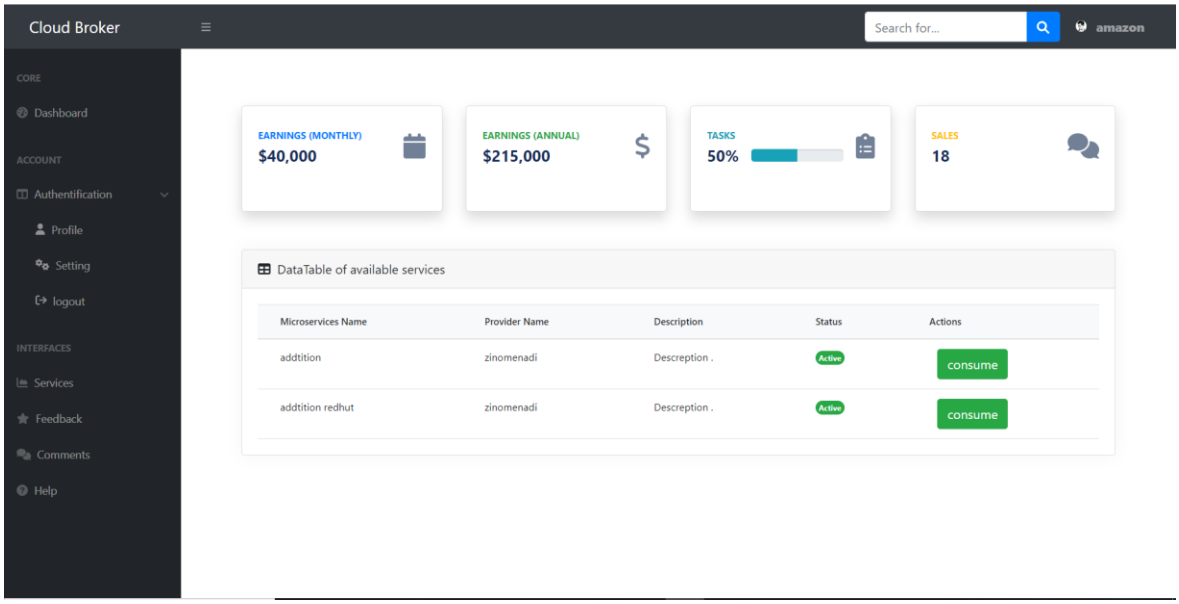

Figure 55 : Liste des instances fournis par les fournisseur

Lorsque le client choisi le service et appuyé sur le bouton consume, le coté front end vas réagi en exécutant la fonction onSubmit () qui permet de passer vers le service choisi (addition), qui lui offre le fournisseur comme représenté la figure 56.

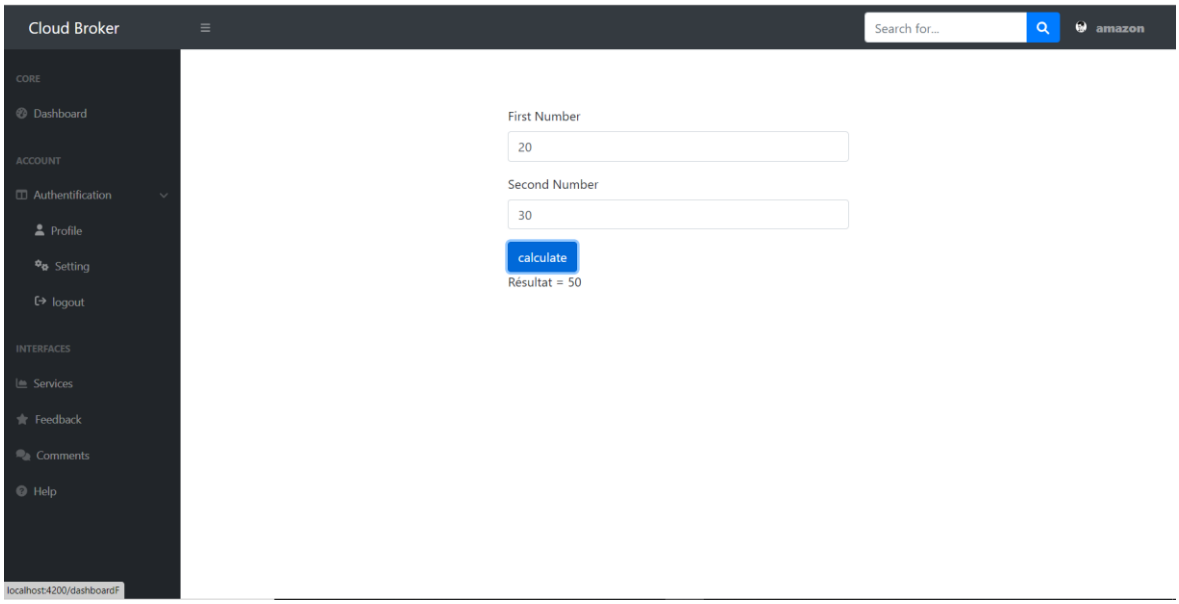

Figure 56: Interface graphique de service addition

Ensuite, Le client utilise le service addition qu'il a demandée pour répondre à ses besoins. Afin d'utiliser ce service le client remplit le formulaire avec des nombres pour faire l'opération addition. Après il appuyé sur submit pour voir le résultat. La fonction onSubmit

() sera exécuter automatiquement pour passer vers la classe de service addition, Cette classe contient la fonction findresult() qui va envoyer les informations saisies par le client sous forme des paramètres en utilisant la méthode GET avec une requête contenant l'URL vers le coté backend comme l'indique la figure suivant :

```
import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class AdditionService {

  url="http://localhost:8500/index";


  constructor(private http:HttpClient) { }

  findresult(x:Number,y:Number){
    return this.http.get<Number>( url: this.url+'?x='+x+'&y='+y+'&operateur="add"');
  }
}
```

Figure 57: la classe service de component addition

Lorsque la requête reçue est par le coté backend, elle passera directement par le proxy et sera exécutée par la méthode de proxyControlleur passer (). Cette méthode permettra de sélectionner la plus proche instance au client comme illustré dans la figure 58

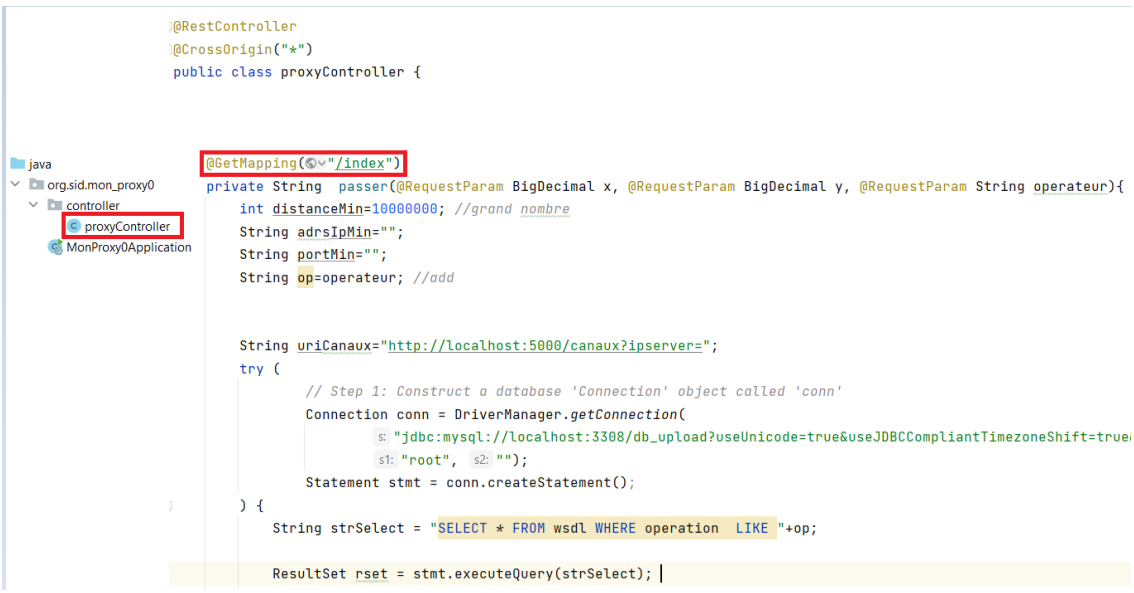

Figure 58: ProxyControlleur de microservice proxy

V.9 Simulation de notre application :

Dans cette partie, nous avons fait une simulation de notre application à l'aide de l'outil docker compose qui permettant d'orchestrer et exécuter plusieurs application (ou plusieurs microservices) au même temps qui sont déployés sur plusieurs conteneurs tel que chaque conteneur est déployé sur une machine indépendante aux autres. Toutes ces machines ayant des adresse IP différentes qui sont inclus dans le même réseau (même Bridge). Docker compose utilise un fichier YAML pour configurer les microservices d'application. Notre objectif principal d'utiliser le docker compose est la simulation de réseau entre nos microservices afin de puisse simuler la distance entre le Client et les microservices des fournisseurs.

V.9.1 Création de Jar pour chaque microservices :

-En premier lieu, il faut créer un fichier JAR pour chaque microservice spring boot. JAR est un fichier d'archive Java utilisé pour stocker tous les programmes et les dépendances nécessaires, pour que le microservice s'exécute correctement.

-On a utilisé Maven pour gérer le cycle de production d'une application Java. Avec Maven Il est possible de créer des Jar pour votre application et celles-ci pourront être exécutés. On a utilisé le plugin Maven dans pom.xml pour spécifier que la Jar de notre application doit inclure toutes les dépendances nécessaires. La création du fichier JAR est utile pour pouvoir exécuter le microservice dans le conteneur. Ci-dessous des illustrations qui montrent les étapes à suivre pour la génération de l'exécutable JAR pour un microservice.

Etape 1 : **Configuration de Maven plugin dans le fichier pom.xml (Figure 59)**

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <version>${project.parent.version}</version>
        </plugin>
    </plugins>
    <finalName>service-config</finalName>
</build>
```

Figure 59: spring-boot-maven-plugin

Etape 2 : Création d'un fichier jar exécutable à l'aide de maven

Cette étape compose de deux phases principales :

La phase Clean : cette phase est destinée à faire un build reproductible, c'est à dire qu'il nettoie tout ce qui a été créé par les versions précédentes. Dans la plupart des cas, il le fait en appelant nettoyage:nettoyer, qui supprime le répertoire lié à ${project.build.directory} (généralement appelé "Target") comme illustre la figure 60

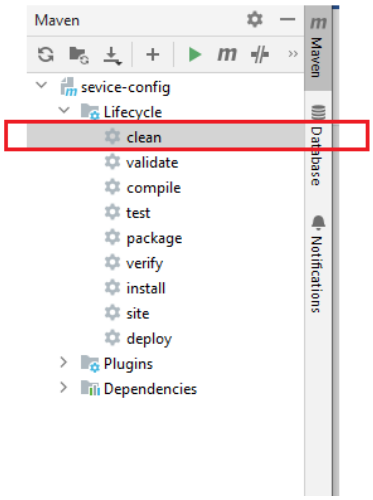

Figure 60: Cleaning a Maven project

La phase Install : pour créer le fichier jar sous le dossier **Target.** La figure 61 représente installation d'un projet maven

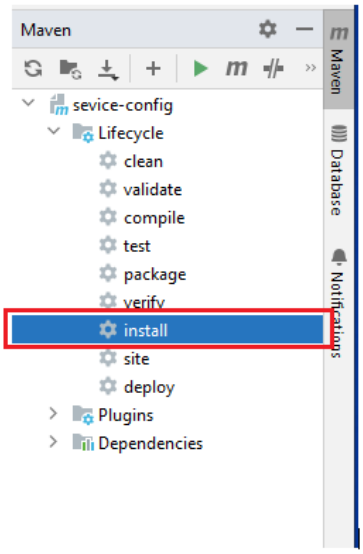

Figure 61: Installing a Maven project

V.9.2 Creation des image docker des microservices :

Dans cette partie , nous expliquons comment créer des images de microservices à l'aide d'un fichier docker (dockerfile) . La première chose **à** faire est de créer un fichier **appelé "Dockerfile** qui est un document texte contenant toutes les étapes nécessaires à la création d'une image Docker pour nos microservices .

Voici un exemple sur docker file de microservice config :

Figure 62 : Dockerfile de microservices config

La figure63 ci-dessus représente un exemple sur docker file de la partie frontend angular ui :

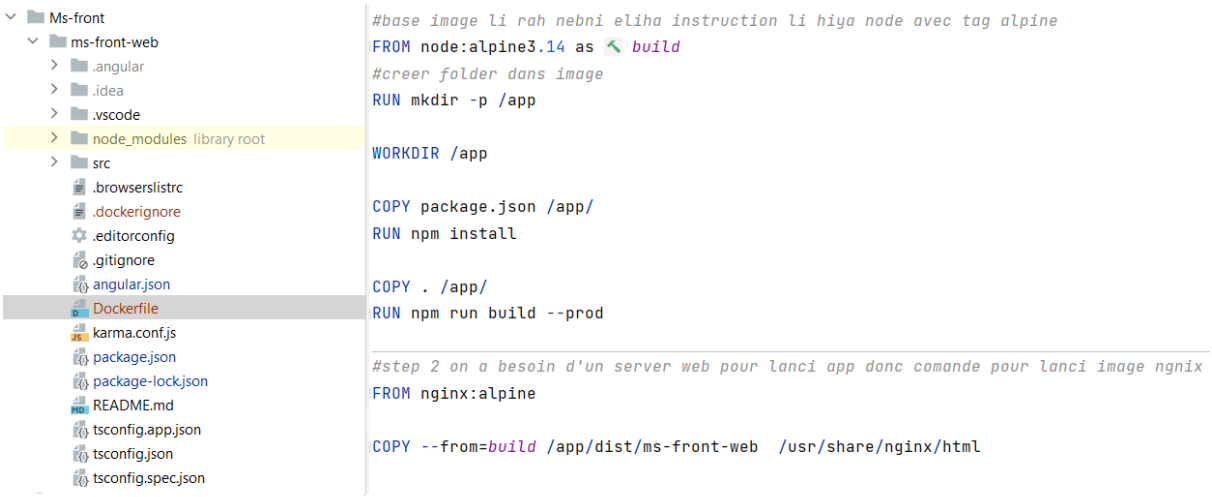

Figure 63 : Dockerfile microservices config

V.9.3 Explication des instructions de docker file

Le tableau ci-dessous représente l'explication des instructions de docker file

Instruction	Objectif
FROM	Définit l'image de base qui sera utilisée par les instructions suivantes.
COPY	Permet de copier des fichiers depuis notre machine locale vers le conteneur Docker.
EXPOSE	Expose un port.
ENTRYPOINT	Comme son nom l'indique, c'est le point d'entrée de votre conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur. Il prend la forme de tableau JSON (ex : CMD ["cmd1","cmd1"]) ou de texte
RUN	Exécute des commandes Linux ou Windows lors de la création de l'image. Chaque instruction RUN va créer une couche en cache qui sera réutilisée dans le cas de modification ultérieure du Dockerfile.
WORKDIR	Définit le répertoire de travail qui sera utilisé pour le lancement des commandes CMD et/ou ENTRYPOINT et ça sera aussi le dossier courant lors du démarrage du conteneur.

Tableau 13: l'explication des instruction de docker file

Ensuite, Une fois le fichier docker créé, vous pouvez créer l'image Docker. Pour ce faire, exécutez simplement la commande build comme illustré dans la figure 64.

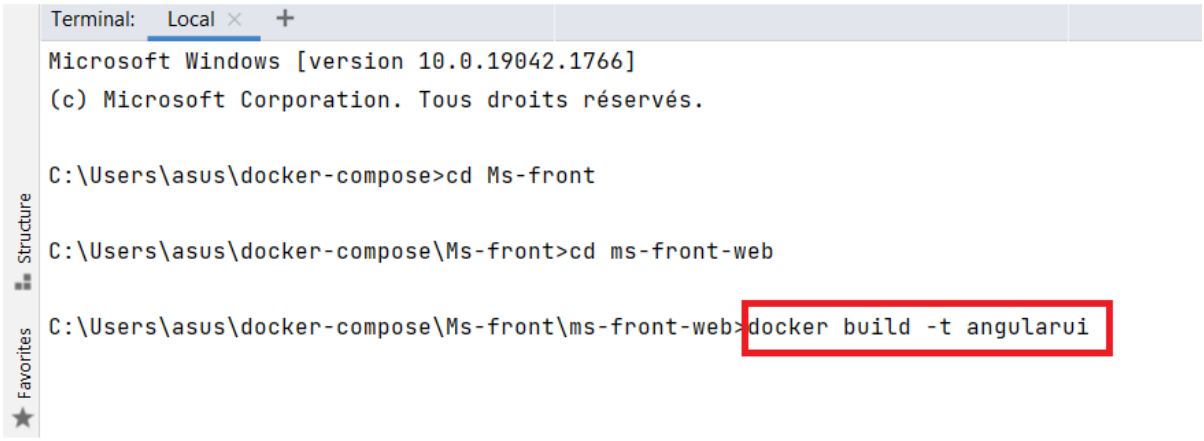

Figure 64: Construire une image à partir de Dockerfile

V.9.4 Création de fichier yml de docker-compose

Le fichier docker-compose.yml au format YAML "décrit" tous les docker run possibles avec images, noms, volumes, liens, ports, variables d'environnements, adresses réseaux.

Docker compose permet de déployer, gérer, combiner et configurer plusieurs conteneurs Docker en même temps.

```yaml
version: '3.9'
services:
  sqldb:
    image: mysql
    restart: always
    environment:
      MYSQL_DATABASE: db_wsdl
      MYSQL_ROOT_PASSWORD: root
      MYSQL_PASSWORD: root
    volumes:
      - db_data:/var/lib/mysql:rw
    ports:
      - 3306:3306
    container_name: sqldb
    networks:
      appnet:
        ipv4_address: 172.24.2.1

  service-register:
    #nom de image formation-service jar
    image: ${DOCKER_REGISTRY-}service-register
    build:
      context: .
      dockerfile: ../IdeaProjects/PFE_project/service-register/Dockerfile
    container_name: service-register
    ports:
      - 8081:8081
    links:
      - sqldb
    volumes:
    - db_data:/var/lib/mysql:rw
    environment:
```

```
      SERVICE_DB_HOST: sqldb
      SPRING_DATASOURCE_URL:
jdbc:mysql://sqldb:3306/db_wsdl?autoReconnect=true&useSSL=false
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "root"
    networks:
      appnet:
        ipv4_address: 172.24.2.2

  ms-fournisseur-authentification:
    image: ${DOCKER_REGISTRY-}ms-fournisseur-authentification
    build:
      context: .
      dockerfile: ../IdeaProjects/PFE_project/ms-fournisseur-
authentification/Dockerfile
    container_name: ms-fournisseur-authentification
    ports:
      - 8089:8089
    links:
      - sqldb
    environment:
      SERVICE_DB_HOST: sqldb
      SPRING_DATASOURCE_URL:
jdbc:mysql://sqldb:3306/dbfournisseur?autoReconnect=true&useSSL=false
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "root"
    networks:
      appnet:
        ipv4_address: 172.24.2.3

  service-config:
    image: ${DOCKER_REGISTRY-}service-config
    build:
      context: .
      dockerfile: ../IdeaProjects/PFE_project/service-config/Dockerfile
    container_name: service-config
    ports:
      - 8085:8085
    networks:
      appnet:
        ipv4_address: 172.24.2.4

  service-gateway:
    image: ${DOCKER_REGISTRY-}service-gateway
    build:
      context: .
      dockerfile: ../IdeaProjects/PFE_project/service-gateway/Dockerfile
    container_name: Service-gateway
    ports:
      - 8500:8500
    networks:
      appnet:
        ipv4_address: 172.24.2.5

  ms-client-authentification:
    image: ${DOCKER_REGISTRY-}ms-client-authentification
    build:
      context: .
      dockerfile: ../IdeaProjects/PFE_project/ms-client-
authentification/Dockerfile
    container_name: ms-client-authentification
```

```
    ports:
      - 8088:8088
    networks:
      appnet:
        ipv4_address: 172.24.2.6
    links:
      - sqldb
    environment:
      SERVICE_DB_HOST: sqldb
      SPRING_DATASOURCE_URL:
jdbc:mysql://sqldb:3306/dbclient?autoReconnect=true&useSSL=false
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "root"

  rest-calculator-master:
    image: ${DOCKER_REGISTRY-}rest-calculator-master
    build:
      context: .
      dockerfile: ../IdeaProjects/PFE_project/rest-calculator-
master/Dockerfile
    container_name: addition-service
    ports:
      - 8055:8055
    networks:
      appnet:
        ipv4_address: 172.24.2.70

  phpmyadmin:
    depends_on:
      - sqldb
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - "8800:80"
    container_name: phpmyadmin
    environment:
      PMA_HOST: sqldb
      MYSQL_ROOT_PASSWORD: root
    networks:
      appnet:
        ipv4_address: 172.24.2.8

  angular-frontend:
    build: ./angular-11-client
    container_name: Frontend-angular-PFE
    ports:
      - 4200:80
    networks:
      appnet:
        ipv4_address: 172.24.2.9

  json-server:
    image: vimagick/json-server
    command: -H 0.0.0.0 -p 5000 -w db.json
    container_name: json-server
    ports:
      - "5000:5000"
    networks:
      appnet:
        ipv4_address: 172.24.2.10
    volumes:
```

```
      - ./data:/data
    restart: always

networks:
  appnet:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: "172.24.2.0/16"

volumes:
  db_data: {}
```

Figure 65: Fichier yaml de docker compose

Le tableau14 ci-dessous représente le rôle de chaque instruction écrite en fichier yml :

Instruction	Rôle
Image	Image Docker à utiliser
Container_name	Nom du conteneur qui va apparaître dans la liste (plutôt que de générer un nom au hasard)
Environment	Variables d'environnement à passer au conteneur
Ports	Correspondance des ports ouverts ; il est recommandé de les entourer de quotes " sinon leur valeur pourrait être mal interprétée
Volumes	Volumes à créer entre la machine hôte et le conteneur
Build	Si le conteneur dépend d'un autre pour son exécution (ex : une base de données)
Links	Pour spécifier les liens entre les services
Networks	Permet de spécifiez des réseaux personnalisés
Ipv4_address:	Permet de spécifier l'adresse IP de la machine où le container sera déployé

Tableau 14: explication des instruction de fichier yaml

V.9.5 Lancement des conteneurs

Une fois le fichier yml de docker compose est prêt il suffira d'une seule commande pour lancer tous les conteneurs au même temps comme illustré dans la figure suivante :

```
PS C:\Users\bbend\IdeaProjects\Docker_compose> docker compose up
```

Figure 66: commande pour lancer tous les conteneurs

Lors de l'exécution de cette commande, Docker Compose commence par vérifier si nous disposons bien en local des images nécessaires au lancement des stacks. Dans le cas contraire, il les télécharge depuis une registre ou les construire via un docker build . la résultat dans la liste des containers de Docker Desktop . la figure 67 représente la liste des containers lancé dans Docker Desktop

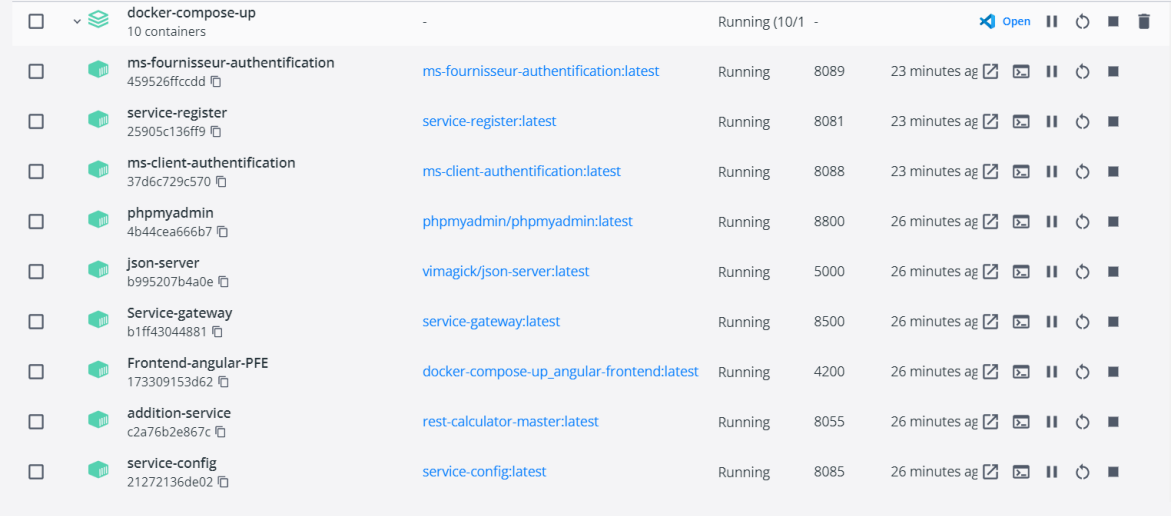

Figure 67 Liste des containers lancé dans Docker Desktop

V.10 Évaluation de la solution par rapport à l'état del'art

Pour évaluer notre solution, nous la comparons avec les solutions que nous avons étudiées dans le chapitre III selon les critères de comparaison utilisés dans l'étude comparative. Cette évaluation est présentée dans les deux tableaux ci-dessous :

Le tableau 16 représente une comparaison de notre solution avec les solutions proposées dans la découverte des microservices mentionnées dans le Chapitre 3 (tableau 4)

Critères	Notre solution
Spring Cloud intégration	Supporté
Gestion des conteneurs	Supporté docker (12 conteneurs lancés au moment de test)
Plusieurs data - centers	Supporté
HTTP	Utilise les requêtes HTTP REST
Nombre des instances recommandé par centre de données	2 ou plusieurs instances de serveur connectées en tant que pairs
Collecte des données du bilan de santé	Données de santé pour chaque microservice sont collectées

Le tableau 15 représente une comparaison de notre solution avec les solutions proposées dans la sélection des microservices mentionnées dans le Chapitre 3 (tableau 5)

Critères	Notre solution
Nombre de microservices	Un seul microservice avec plusieurs instances
Paramètres pris en considération	-La distance entre le client et les différents instances de microservice demandé - Critère de QoS : temps de réponse
Objectifs	-réduire temps de réponse
Environnement	-Cloud
Requête utilisateur	-Pas traité

Composition des microservices	-Pas traité
Principes	- Calculé toutes les distances entre le client et les serveurs où les instances des microservices sont situés puis prendre le serveur qui la distance minimale

Tableau 15 Évaluation de la solution par rapport à l'état de l'art

V.11 Conclusion

Ce chapitre est dédié à la mise en œuvre de notre approche proposée qui est une solution Middleware entre les clients qui sont les consommateurs des services de types FaaS et les fournisseurs des services. Au début, nous avons exposé les ressources matérielles sur lesquelles notre solution a été réalisée et les outils utilisés ainsi que l'environnement de développement. Ensuite, nous avons présenté quelques concepts de bases sur springboot. Des explications détaillées sur le code source des services implémentés sont données. La présentation des interfaces graphiques de notre application est aussi discutée. Enfin, nous avons fait une simulation de notre application en utilisant l'outil docker compose et une comparaison de l'approche proposée avec les approches décrites dans l'état de l'art est présentée

CONCLUSION ET PERSPECTIVES

L'objectif principal de ce mémoire est de présenter une solution Middleware entre les fournisseurs des services cloud de type FaaS (Function as a Service) et les clients, qui sont les consommateurs des services. Pour cela, nous avons réalisés une application qui permet au Middleware de sélectionner au client les meilleures instances des microservices fournis par les fournisseurs dans un environnement cloud selon le critère d'optimisation de temps de réponse (réduction de la latence).

Afin d'aboutir à cet objectif, nous avons entamé notre travail par un aperçu général sur les différents concepts du cloud computing. Une revue sur les architectures logicielles et leurs utilisations est présentée, ainsi qu'une étude détaillée sur les concepts des microservices et leurs relations avec cloud est aussi discutée. Ensuite, nous avons étudié et comparé les travaux antérieurs réalisés ayant traité la sélection et la découvrabilité des instances de microservices dans le cloud basé sur le contexte. D'après cette revue de littérature, nous avons constaté que la localisation du serveur par rapport aux clients où l'instance de microservice a été déployée affecte significativement le temps de réponse des microservices, ce qui nous a conduit à réfléchir pour trouver une solution afin de choisir le serveur le plus proche au client. D'une manière équivalente notre problème revient à minimiser la distance entre le client et le serveur exécutant des microservices, c'est à dire le choix de l'itinéraire optimal de la requête du client pour consommer un service. A cet effet, nous avons proposé une solution middleware basée sur la sélection et la découverte des microservices dans un environnement cloud en utilisant un algorithme (gateway_algo) pour la sélection développé dans le cadre de notre travail. La découverte des microservices est réalisée à travers le document WSDL. L'efficacité de notre solution a été prouvée à travers une simulation de notre application à l'aide de l'outil docker compose. Ceci nous a permis de simuler le réseau entres différents microservices de l'application.

Pour l'implémentation de la solution proposée, nous avons réalisé une application JAVA EE basée sur le Framework SPRINGBOOT pour le coté back end et ANGULAR pour le coté front end.

Perspectives

Cette étude nous a permis d'acquérir des nouvelles connaissances dans le domaine de l'informatique, notamment les microservices dans un environnement Cloud. Ce travail reste limité et peut être compléter par d'autres travaux futurs qui peuvent se résumer comme suit :

- l'utilisation des algorithmes intelligents dans la découverte et la sélection des microservices.

- Adopter cette solution à des environnements multi Cloud.

- Implémenter la notion de composition des services.

Références bibliographique

[1] J. Surbiryala and C. Rong, "Cloud computing: History and overview," *Proc. - 2019 3rd IEEE Int. Conf. Cloud Fog Comput. Technol. Appl. Cloud Summit 2019*, vol. 23, no. 6, pp. 44–61, 2019, doi: 10.1109/CloudSummit47114.2019.00007.

[2] S. I. Bairagi and A. O. Bang, "Cloud Computing : History , Architecture , Security Issues," vol. 31, no. 6, pp. 132–139, 2018.

[3] I. M. A.Kalvin, "Companies supporting the openstack foundation," 2018. https://www.openstack.org/community/supporting-organizations/ (accessed Jun. 25, 2022).

[4] P. Kapoun, P. Faculty, F. Šrámka, and O.- Mariánské, "Global Journal of Information Technology," *Glob. J. Inf. Technol.*, vol. 4, no. 2, pp. 114–119, 2014.

[5] IBM, "Cloud computing | IBM," *Ibm*, 2020. https://www.ibm.com/cloud/learn/cloud-computing (accessed Jun. 23, 2022).

[6] A. W. Services, "AWS | Qu'est-ce que le cloud computing – Les avantages du cloud." https://aws.amazon.com/fr/what-is-cloud-computing/ (accessed Jun. 03, 2022).

[7] Microsoft Corporate, "What is cloud computing? A beginner's guide| Microsoft Azure," *Cloud computing*, 2016. https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/#uses (accessed Apr. 27, 2022).

[8] V. Rajaraman, "Cloud computing," *Resonance*, vol. 19, no. 3, pp. 242–258, 2014, doi: 10.1007/s12045-014-0030-1.

[9] VMware, "What is a Private Cloud? | VMware Glossary," 2021. https://www.vmware.com/topics/glossary/content/private-cloud (accessed Apr. 08, 2022).

[10] A.kein, "Private cloud, public cloud et hybrid cloud - Global SP - Hébergement Cloud France." https://www.globalsp.com/private-cloud-public-cloud-et-hybrid-cloud/ (accessed Apr. 13, 2022).

[11] Geekflare, "Modèles de services cloud - PaaS, SaaS, IaaS, FaaS et plus," 2019. https://geekflare.com/fr/cloud-service-models/ (accessed Apr. 25, 2022).

[12] F. Hao *et al.*, "An Optimized Computational Model for Multi-Community-Cloud Social Collaboration," *IEEE Trans. Serv. Comput.*, vol. 7, no. 3, pp. 346–358, 2014, doi: 10.1109/TSC.2014.2304728.

[13] M. I. Malik, "Cloud Computing-Technologies," *Int. J. Adv. Res. Comput. Sci.*, vol. 9, no. 2, pp. 379–384, 2018, doi: 10.26483/ijarcs.v9i2.5760.

[14] Vic (J.R.) Winkler, "Cloud Deployment Model - an overview | ScienceDirect Topics," *Science Direct*. 2011. [Online]. (accessed Apr. 25, 2022) https://www.sciencedirect.com/topics/computer-science/cloud-deployment-model

[15] Redhat, "Qu'est-ce que le multiCloud," *RedHat*, 2020. https://www.redhat.com/fr/topics/cloud-computing/what-is-multicloud (accessed Jul. 01, 2022).

[16] D. Rani and R. K. Ranjan, “A Comparative Study of SaaS , PaaS and IaaS in Cloud Computing,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 4, no. 6, pp. 458–461, 2014.

[17] S.Stan, “CLOUD COMPUTING : SAAS,” *Comput. Sci. Telecommun.*, vol. 36, no. 4, pp. 1512–1232, 2012.

[18] C. Tan, K. Liu, and L. Fernando, “A design of evaluation method for SaaS in cloud computing,” *J. Ind. Eng. Manag.*, vol. 6, no. 5, pp. 50–72, 2013, doi: 10.3926/jiem.661.

[19] T. Backir and V. A. B, *Ceiling fans cool down a very hot plant*, vol. 56, no. 12. Springer International Publishing, 2001. doi: 10.1007/978-3-319-99819-0.

[20] J. Manner, M. Endreb, T. Heckel, and G. Wirtz, “Cold start influencing factors in function as a service,” *Proc. - 11th IEEE/ACM Int. Conf. Util. Cloud Comput. Companion, UCC Companion 2018*, vol. 21, no. 7, pp. 181–188, 2018, doi: 10.1109/UCC-Companion.2018.00054.

[21] V. V. Arutyunov, “Cloud computing: Its history of development, modern state, and future considerations,” *Sci. Tech. Inf. Process.*, vol. 39, no. 3, pp. 173–178, 2012, doi: 10.3103/S0147688212030082.

[22] P. A. Abdalla, “Small-Sized Business,” *2019 7th Int. Symp. Digit. Forensics Secur.*, vol. 11, no. 9, pp. 136–147, 2019.

[23] A. Ali, M. Megargel, A. Megargel, V. Shankararaman, and D. K. Walker, “Institutional Knowledge at Singapore Management University Migrating from monoliths to cloud-based microservices : A banking industry example Chapter 1 : Migrating from Monoliths to Cloud-Based Microservices : A Banking Industry Example,” *IOSR J. Comput. Eng. Ver. IV*, vol. 33, no. 6, pp. 110–114, 2020.

[24] F. Ponce, G. Marquez, and H. Astudillo, “Migrating from monolithic architecture to microservices: A Rapid Review,” *Proc. - Int. Conf. Chil. Comput. Sci. Soc. SCCC*, vol. 19, pp. 10–17, 2019, doi: 10.1109/SCCC49216.2019.8966423.

[25] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, “From monolith to microservices: A classification of refactoring approaches,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11350 LNCS, no. 4, pp. 128–141, 2019, doi: 10.1007/978-3-030-06019-0_10.

[26] Amazon, “Microservices,” 2021. https://aws.amazon.com/fr/microservices/ (accessed Jul. 13, 2022).

[27] M. D. Cookson and P. M. R. Stirk, “architecture monolithique vs microservices,” *L’architecture monolithique*, vol. 11, pp. 203–241, 2019.

[28] C. Richardson and F. Smith, “Microservices - From Design to Deployment,” *Nginx*, vol.13 pp. 90–115,2016.

[29] JeremyLikness, Erjain, Alexbuckgit, DCtheGeek, Youssef1313, and Mairaw, “architecture-approaches. vol.09 pp. 24-39,2017”

[30] K. B. Laskey and K. Laskey, “Service oriented architecture,” *Wiley Interdiscip. Rev.*

Comput. Stat., vol. 34, no. 6, pp. 101–120, 2009, doi: 10.1002/wics.8.

[31] G. OasisGroup, "Service-Oriented Architecture," 2017vol. 16, no. 19, pp. 88-111, 2009, doi: 12.3211/AEA35899.8966111.

[32] N. Komoda, "Service Oriented Architecture (SOA) in industrial systems," *2006 IEEE Int. Conf. Ind. Informatics, INDIN'06*, pp. 1–5, 2006, doi: 10.1109/INDIN.2006.275708.

[33] F. Houacine, S. Architecture, and C. Computing, "Service-Oriented Architecture for the Mobile Cloud Fatiha Houacine To cite this version : HAL Id : tel-01829893 Service-Oriented Architecture for the Mobile Cloud Computing," 2018.

[34] R. DOUMA, "Comment construire (et faire évoluer) les systèmes avec les microservices," *IEEE Cloud Comput.*, 2019.

[35] C. Hochreiner, O. Kopp, and O. Kopp, "ZEUS 2018 10 th ZEUS Workshop , ZEUS 2018 ," no. 10, pp. 8–9, 2018.

[36] G. Y. Eric Manuguerra, "Architectures micro-services : objectifs, bénéfices et défis - Partie 2 | SQLI," 2019. https://www.sqli.com/fr-fr/insights-news/blog/architectures-micro-services-objectifs-benefices-et-defis-partie-2 (accessed Jun. 23, 2022).

[37] IBM Cloud Education, "What are Microservices?," 2021. https://www.ibm.com/cloud/learn/education (accessed Jun. 01, 2022).

[38] Microsoft, "Monoliths to microservices using domain-driven design - Azure Architecture Center | Microsoft Docs," 2021. https://docs.microsoft.com/en-us/azure/architecture/microservices/migrate-monolith (accessed Jun. 24, 2022).

[39] S. J. Fowler, "Production-ready Microservices: Building Standardized Systems Across an Engineering Organization," *IEEE Cloud Comput.*, 2016 no.4, pp. 21-27, 2018.

[40] A. Megargel, "Digital banking: Overcoming barriers to entry (Doctoral Dissertation)," Retrieved from Singapore Management University," 2018. [Online]. (accessed Jun. 10, 2022) https://ink.library.smu.edu.sg

[41] Remya Mohanan, "What Is DevOps Lifecycle? Definition, Key Components, and Management Best Practices," https://www.spiceworks.com/tech/devops/articles/what-is-devops-lifecycle (accessed Jun. 04, 2022)

[42] D. Taibi, V. Lenarduzzi, and C. Pahl, "Continous Architecting with Microservices and DevOps " no. 10, pp. 10–12, 2019.

[43] A. AMAZON, "En quoi consiste le DevOps ?," https://aws.amazon.com/fr/devops/what-is-devops (accessed Jun. 04, 2020).

[44] R. Jabbari, N. Bin Ali, K. Petersen, and B. Tanveer, "What is DevOps? A systematic mapping study on definitions and practices," *ACM Int. Conf. Proceeding Ser.*, vol. 24-May-201, 2016, doi: 10.1145/2962695.2962707.

[45] S. I. Mohamed, "DevOps Shifting Software Engineering Strategy Value Based Perspective," *IOSR J. Comput. Eng. Ver. IV*, vol. 17, no. 2, pp. 2278–661, 2015, doi: 10.9790/0661-17245157.

[46] T.Jack, “The Role of Microservices in DevOps,” https://www.journaldunet.com/solutions/cloud-computing/1166432-microservices-est-ce-realiste/1167036-microservices-et-devops. (accessed Jun. 09, 2022).

[47] Henry, “Agile vs DevOps: What’s the Difference?,” https://phoenixnap.com/blog/devops-vs-agile (accessed Jun. 03, 2021.

[48] B.Owen, “Qu’est-ce que la méthodologie Agile ?,” 2020. https://etcdigital.fr/blog/quest-ce-que-la-methodologie-agile/ (accessed Jun. 05, 2022).

[49] N.Kalem, “Différence entre DevOps et Agile,” 2019. https://waytolearnx.com/2019/04/difference-entre-devops-et-agile.html (accessed Jun. 21, 2022).

[50] Kilien Sando, “Comprendre la virtualisation,” Mar. 18, 2018. https://www.redhat.com/fr/topics/virtualization (accessed Jun. 24, 2022).

[51] A. Abuabdo and Z. A. Al-sharif, “Virtualization vs. Containerization: Towards A Multithreaded Performance Evaluation Approach,” *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA*, vol. 6, pp. 12–19, 2021.

[52] AlibabaCloud, “Différences entre les Conteneurs et les Machines Virtuelles - Alibaba Cloud,” Sep. 03, 2021. https://www.alibabacloud.com/fr/knowledge/difference-between-container-and-virtual-machine (accessed Jun. 23, 2022).

[53] Akafine Dayde, “What’s the difference between VMs & Containers? | AKF Partners,” 2019. https://akfpartners.com/growth-blog/vms-vs-containers (accessed Jun. 24, 2022).

[54] A. Singleton, “The Economics of Microservices,” *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 16–20, 2016, doi: 10.1109/MCC.2016.109.

[55] A. Bhardwaj and C. R. Krishna, “Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey,” *Arab. J. Sci. Eng.*, vol. 46, no. 9, pp. 8585–8601, 2021, doi: 10.1007/s13369-021-05553-3.

[56] V. K. Pachghare, “Microservices Architecture for Cloud Computing,” *J. Inf. Technol. Sci.*, vol. 2, no. 1, pp. 1–13, 2016.

[57] N. D. Keni and A. Kak, “Adaptive Containerization for Microservices in Distributed Cloud Systems,” *2020 IEEE 17th Annu. Consum. Commun. Netw. Conf. CCNC 2020*, vol. 11, pp. 14–23, 2020, doi: 10.1109/CCNC46108.2020.9045634.

[58] Microsoft, “Microservices with .NET and Docker containers,” 2021. https://dotnet.microsoft.com/en-us/apps/aspnet/microservices (accessed Jun. 23, 2022).

[59] Clarence, “Take the confusion out of Docker, VMs, and microservices - IBM Developer,” Jan. 02, 2019. https://developer.ibm.com/articles/breaking-down-docker-and-microservices/ (accessed Jun. 23, 2022).

[60] Sancixo Baniya, “Docker builds mongodb cluster | Develop Paper,” 2022. https://developpaper.com/docker-builds-mongodb-cluster/ (accessed Jun. 24, 2022).

[61] J. Shao, X. Zhang, and Z. Cao, “Research on context-based instances selection of microservice,” *ACM Int. Conf. Proceeding Ser.*, vol. 6, pp. 22–24, 2018, doi:

10.1145/3207677.3277954.

[62] Swan and Z. Y. Chen Fang, Jindong Wang, “Web Service Selection Method Based on Dynamic QoS,” *Comput. Sci.*, vol. 44, no. 5, pp. 245–250, 2017.

[63] T. H. Lin Bai, Dan Ye, Jun Wei, “An efficient service selection method based on service function specification,” *J. Softw.*, vol. 26, no. 8, pp. 36–45, 2015.

[64] H. Liu, Z. Cao, and X. Zhang, “An efficient algorithm of context-clustered microservice discovery,” *ACM Int. Conf. Proceeding Ser.*, vol. 10, pp. 22–24, 2018, doi: 10.1145/3207677.3277949.

[65] N. G. Sonia Ben Mokhtar, Davy Preuveneers, “Efficient semantic service discovery in pervasive computing environments with QoS and context support,” *J. Syst. Softw.*, vol. 81, no. 5, pp. 785–808, 2008.

[66] Z. Houmani, D. Balouek-Thomert, E. Caron, and M. Parashar, “Enhancing microservices architectures using data-driven service discovery and QoS guarantees,” *Proc. - 20th IEEE/ACM Int. Symp. Clust. Cloud Internet Comput. CCGRID 2020*, vol. 7, pp. 290–299, 2020, doi: 10.1109/CCGrid49817.2020.00-64.

[67] F.nuxus, “Consul data model,” 2020. https://www.consul.io/docs/agent/services.html (accessed Jun. 10, 2022).

[68] T. Hunter II, *Advanced Microservices*. 2017. vol. 05, pp. 18–20, 2018 doi: 10.1007/978-1-4842-2887-6.

[69] S. Cusimano, “Service Discovery in Microservices,” 2022. https://www.baeldung.com/cs/service-discovery-microservices (accessed Jun. 10, 2022).

[70] C. Richardson, “Service Discovery in a Microservices Architecture,” 2015. https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/ (accessed Jun. 14, 2022).

[71] Netflix, “Discovery service Eureka,” 2014. https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance (accessed Jun. 10, 2022).

[72] G. McDonald, “What is ZooKeeper?,” 2021. https://zookeeper.apache.org/ (accessed Jun. 03, 2022).

[73] K. Cardenas, “Introduction to Consul,” 2021. https://www.consul.io/ (accessed Jun. 07, 2022).

[74] M.Cris, “Building Microservices Using an API Gateway ,” 2021. https://www.nginx.com/blog/building-microservices-using-an-api-gateway/ (accessed Jul. 21, 2022).

[75] I.Albert, “Announcing Zuul: Edge Service in the Cloud | by Netflix Technology Blog | Netflix TechBlog,” 2013. https://netflixtechblog.com/announcing-zuul-edge-service-in-the-cloud-ab3af5be08ee (accessed Jul. 21, 2022).

[76] M.Nilza, “Spring Cloud Tutorial- Netflix Zuul + Eureka ,” 2017. https://www.javainuse.com/spring/spring-cloud-netflix-zuul-tutorial (accessed Jul. 21,

2022).

[77] A.baldwin, “Spring Cloud Load Balancer | Baeldung,” 2021. https://www.baeldung.com/spring-cloud-load-balancer (accessed Jul. 21, 2022).

[78] F.Dineshchandgr, “Spring Cloud Gateway,” 2020. https://blog.devgenius.io/what-is-spring-cloud-gateway-34dfdc9548bc (accessed Jul. 21, 2022).

[79] spring.io, “Spring Cloud Gateway API spring.io ,” 2019. https://spring.io/projects/spring-cloud-gateway (accessed Jul. 21, 2022).

[80] Abdi Khan, “Differences between Netflix zuul and Spring cloud gateway - Knoldus Blogs,” 2020. https://blog.knoldus.com/differences-netflix-zuul-and-spring-cloud-gateway/ (accessed Jul. 21, 2022).

[81] J. Blankenship, M. Bussa, and S. Millett, “eXtreme Programming,” *Pro Agil. .NET Dev. with Scrum*, vol. 31, no. 6, pp. 29–51, 2011, doi: 10.1007/978-1-4302-3534-7_3.

[82] ExtremeProgramming.org, “Extreme Programming: A Gentle Introduction.,” 2019. http://www.extremeprogramming.org/ (accessed Jun. 24, 2022).

[83] R. Juric, “WestminsterResearch,” *IEEE*, vol. 41, no. 3, pp. 32–40, 2014, [Online](accessed Jun. 17, 2019) http://www.westminster.ac.uk/westminsterresearch%0AExtreme

[84] Alex Cia, “extreme-programming-values-principles-and-practices,” 2017. https://www.altexsoft.com/blog/engineering/striking-a-balance-between-manual-and-automated-testing-when-two-is-better-than-one/ (accessed Jun. 24, 2022).

[85] apachefriends.org, “XAMPP ,” 2021. https://www.apachefriends.org/fr/index.html (accessed Jun. 24, 2022).

[86] Baeldung, “The Spring @Controller and @RestController Annotations,” 2022. https://www.baeldung.com/spring-controller-vs-restcontroller (accessed Jun. 09, 2022).

[87] P.Nazin, “Key Differences Between Spring vs Spring Boot vs Spring MVC,” 2022. https://www.simplilearn.com/tutorials/spring-boot-tutorial/spring-vs-spring-boot (accessed Jun. 15, 2022).

[88] Baeldung, “A Controller, Service and DAO Example with Spring Boot and JSF,” 2022. https://www.baeldung.com/jsf-spring-boot-controller-service-dao (accessed Apr. 19, 2022).