

République Algérienne Démocratique et Populaire  
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Djilali BOUNAAMA de Khemis Miliana

Faculté des Sciences et de la Technologie

Département de Mathématiques et d'Informatique



# Mémoire de fin d'études

En vue de l'obtention du diplôme **Master** en Informatique

**Spécialité :**

Génie Logiciel et Systèmes Distribués

## Thème

---

Mise en place d'une Architecture Orientée Services basée sur le  
Cloud Computing

---

**Présenté par :**

*M. KABORE* Abdul Razzaq

*M. SAVADOGO* Mahama

**Devant le jury composé de :**

Examineur 1 : M.Azzouza Nouredine

Examineur 2 : M.Khalfi Ali

Encadreur : M. Haniche Fayçal

**Année universitaire 2019/2020**

# Remerciements

Nous tenons à remercier d'abord notre encadreur de mémoire Monsieur Haniche Fayçal qui nous a encadrer durant toute la réalisation de ce projet. Malgré la crise de Covid-19 et ses risques, il ne manquait pas de passer à la résidence universitaire pour nous encadrer. Nous remercions ensuite l'ensemble des professeurs du département des Mathématiques et Informatique pour leur enseignement durant le cursus universitaire, ainsi que toute l'administration de l'Université Djilali Bounaama de Khemis Miliana. Enfin nous remercions nos collègues Algériens et internationaux avec qui nous avons passé les cinq ans de cursus universitaire.

# Dédicaces

Je dédie ce mémoire à :

Mes chers parents mon père et ma mère qui ont toujours été mon premier soutien dans mes études et mon modèle à suivre. Je ne saurais exprimer par des mots l'affection, la patience et l'attention qu'ils ont à mon égard. Que cette dédicace leur témoigne ma profonde gratitude Mes oncles Hamidou et Assami qui ont toujours été là pour moi depuis mon admission au cycle secondaire. Je leur témoigne ma gratitude. Mes frère Abdoulaye et Oumarou, pour leur soutien et leur encouragement Mes enseignants et personnel administratif du primaire Ecole de Tougué-Mossi, du secondaire et du lycée Complexe Scolaire Horizon International qui m'ont donné les outils nécessaires pour la suite de mes études.

Mahama SAVADOGO

Je voudrais tout d'abord dédié ce mémoire à ma mère, à mon père à toute ma famille ainsi qu'à mes proches pour le soutien permanent qu'ils m'ont apporté. Et aussi une pensée particulière à mes enseignants, camarades et tout le personnel du Lycée Municipal de Sig-Noghin et de l'école la Verdure qui ont grandement participer à mon épanouissement scolaire et aussi social.

Abdul Razzaq KABORE

# Résumé

Les entreprises cherchent inlassablement à être compétitives sur le marché à travers l'amélioration de la qualité du service. Au centre de toutes les stratégies qui mènent à l'atteinte de cet objectif se trouve le système d'information SI qui supporte le métier. Ce SI se compose de plusieurs applications hétérogènes qui doivent être intégrées. Ainsi le système doit disposer d'une certaine agilité pour supporter les changements futurs à un coût minimal. Pour atteindre cet objectif, le concept de l'architecture orientée services (SOA) qui consiste à décomposer le SI en un ensemble de fonctionnalités appelées services, alignées avec les fonctions du métier de l'entreprise semble prometteur. La SOA accorde plus d'importance à la réutilisation des fonctionnalités du SI de l'entreprise et facilite l'adaptation au changement. Dans ce projet, nous avons effectué une étude de l'architecture orientée service SOA et des concepts liés au Cloud Computing. L'objectif était de tenter l'exploitation de la technologie du cloud computing pour la mise en place d'un système à base d'SOA. Nous avons d'abord installé et configuré OpenNebula ( un système de gestion de cloud Computing) qui a permis la mise en place d'un cloud privé minimisé. Ensuite nous avons fait la conception et la réalisation d'un système à base d'SOA, de gestion de bibliothèque universitaire . Les services ont été mis en place en tant que microservices. Ces derniers ont été implémentés grâce au Framework Spring Boot et la plateforme Spring Cloud qui offre les outils nécessaires au déploiement et à l'exécution.

Mots clés : **Architecture orientée services, Architecture microservice, Cloud Computing**

# Abstract

Companies tirelessly seek to be competitive in the market by improving the quality of production. At the center of all the strategies that lead to the achievement of this objective is the IS information system that supports the business. This IS is made up of several heterogeneous applications that must be integrated. So the system must have some agility to support future changes at minimal cost. To achieve this goal, the concept of service-oriented architecture (SOA), which consists of breaking down the IS into a set of functionalities called services, aligned with the functions of the business of the company seems promising. SOA places more emphasis on reusing enterprise IS functionality and makes it easier to adapt to change. In this project, we carried out a study of SOA service-oriented architecture and concepts related to Cloud Computing. The objective was to attempt to exploit cloud computing technology for the implementation of an SOA-based system. We first installed and configured OpenNebula (a cloud computing management system) which allowed the implementation of a minimized private cloud. Then we designed and built a system based on SOA, university library management. Services were set up as microservices. These were implemented using the Spring Boot Framework and the Spring Cloud platform which provides the tools necessary for deployment and execution. Keyword : Service Oriented Architecture, Microservice Architecture, Cloud Computing

**Keywords** : Service Oriented Architecture, Microservice Architecture, Cloud Computing.

## المُلخَص

تسعى الشركات بلا كلل إلى أن تكون قادرة على المنافسة في السوق من خلال تحسين جودة الخدمة. في قلب جميع الاستراتيجيات التي تؤدي إلى تحقيق هذا الهدف يوجد نظام المعلومات الذي يدعم و يرافق النشاطات القاعدية للمؤسسة. يتكون نظام المعلومات عادةً من عدة تطبيقات, قد تكون غير متجانسة, يتوجب دمجها. من جانب آخر, يجب أن يتمتع النظام ببعض المرونة لدعم التغييرات المستقبلية بأقل تكلفة. لتحقيق هذا الهدف, يبدو نموذج الهندسة الموجهة نحو الخدمات (SOA) واعدًا. هذا النموذج يعتمد أساسا على تقسيم نظام المعلومات إلى مجموعة من الوظائف تسمى خدمات, بما يتماشى مع نشاطات المؤسسة, يرتكز نموذج SOA بشكل أكبر على ميزة إعادة استخدام الخدمات و تركيبها في مسارات جديدة, مما يجعل من السهل التكيف مع التغيير.

في هذا المشروع ، أجرينا دراسة لنماذج هندسة النظم الموجهة للخدمات والمفاهيم المتعلقة بالحوسبة السحابية. كان الهدف هو محاولة استغلال تقنية الحوسبة السحابية لتضييف وتنفيذ نظام قائم على نموذج SOA. قمنا أولاً بتهيئة و تخصيص OpenNebula (وهو نظام لإدارة الحوسبة السحابية) والذي سمح بإنشاء سحابة خاصة مصغرة. من جانب آخر, قمنا بتصميم وبناء نظام قائم على نموذج SOA متعلق بإدارة أعمال المكتبة الجامعية (اكتفينا فقط ببعض الوظائف من أجل التجريب). تم إقامة الخدمات على شكل خدمات مصغرة. و التي قمنا ببرمجتها و إنجازها باستخدام Spring Boot Framework وتضييفها و تنفيذها باستخدام منصة Spring Cloud التي توفر الأدوات اللازمة للنشر والتنفيذ.

**الكلمات المفتاحية:** نموذج الهندسة الموجهة للخدمات ، هندسة الخدمات المصغرة ، الحوسبة السحابية.

# Table des matières

Liste des tableaux	8
Liste des figures	10
<b>1 L'Architecture Orientée Service</b>	<b>15</b>
1.1 Introduction	15
1.2 Comprendre l'architecture SOA	15
1.2.1 Aperçu sur l'avant SOA : Les applications monolithiques	15
1.2.2 Définition de la SOA	17
1.2.3 Notion de Service	17
1.2.4 Les principes de l'Architecture Orientée Service	19
1.3 Les technologies utilisés pour la mise en oeuvre de la SOA	20
1.3.1 Les technologies à base d'objets distants	20
1.3.2 Les technologies à base de composants logiciels	22
1.3.3 Les middlewares	23
1.3.4 Les technologies à base de web services	26
1.3.5 ESB	26
1.4 L'Architecture Microservices : la SOA évoluée ?	27
1.4.1 SOA vs Microservices	28
1.4.2 Les différences clés entre microservices et SOA	29
1.5 Conclusion	30
<b>2 Les Services Web</b>	<b>31</b>
2.1 Introduction	31
2.2 Définition des Services Web	31
2.3 Architecture à base de web services	32
2.3.1 WSDL	33
2.3.2 UDDI	35
2.3.3 Protocoles de Communication	36
2.4 Pile des Services Web (Architecture étendue)	37
2.4.1 Gestion des services web dans l'entreprise	37
2.5 Conclusion	39
<b>3 Le Cloud Computing</b>	<b>40</b>
3.1 Introduction	40
3.2 Définition du cloud computing	40
3.3 L'évolution du Cloud Computing	40
3.4 Les caractéristiques du Cloud	42

3.5	Les services du Cloud Computing ou modèles de livraison . . . . .	42
3.5.1	IaaS (Infrastructure as a Service) . . . . .	42
3.5.2	Pass(Plateforme as a Service) . . . . .	44
3.5.3	SaaS (Software as a Service) . . . . .	45
3.6	Les modèles de déploiement du Cloud Computing . . . . .	46
3.6.1	Les Clouds publics . . . . .	46
3.6.2	Les clouds privés . . . . .	46
3.6.3	Les clouds hybrides . . . . .	46
3.6.4	Les clouds communautaires . . . . .	46
3.7	Les principaux fournisseurs de services cloud . . . . .	46
3.7.1	Les fournisseurs du SaaS . . . . .	46
3.7.2	Fournisseurs IaaS . . . . .	47
3.7.3	Fournisseurs PaaS . . . . .	47
3.8	Les avantages du Cloud Computing . . . . .	47
3.9	Inconvénients du Cloud Computing . . . . .	47
3.10	Technologies et Composants du cloud Computing . . . . .	48
3.10.1	Les serveurs . . . . .	48
3.10.2	La virtualisation . . . . .	49
3.10.3	Les solutions de type IaaS . . . . .	49
3.10.4	Les solutions de type PaaS . . . . .	53
3.10.5	Les solutions de type SaaS . . . . .	53
3.11	Conclusion . . . . .	53
<b>4</b>	<b>Exploitation du Cloud et des services web pour la mise en place d'une architecture orientée services</b>	<b>54</b>
4.1	Introduction . . . . .	54
4.2	Présentation des outils de mise en place d'un système à base de SOA . . . . .	54
4.2.1	Mise en place d'une Plateforme de gestion du cloud . . . . .	54
4.3	Mise en place des microservices . . . . .	64
4.3.1	Description des microservices à mettre en œuvre . . . . .	64
4.3.2	Présentation des différents diagrammes de l'application . . . . .	65
4.4	Présentation des outils de mise en œuvre . . . . .	69
4.4.1	Le Framework Spring boot . . . . .	69
4.4.2	Spring Cloud . . . . .	69
4.4.3	Présentation de l'architecture microservice . . . . .	72
4.4.4	Tests des services développés . . . . .	73
4.4.5	Test des services Composés . . . . .	81
4.4.6	Validation des résultats . . . . .	83
4.5	Comparaison entre SOA et applications monolithiques . . . . .	84
	<b>Bibliographie</b>	<b>85</b>



# Liste des tableaux

3.1	Différence entre serveur dédié physique et serveur cloud . . . . .	48
3.2	Comparatif entre les solutions libres de types IaaS . . . . .	52
4.1	Opérations et attributs . . . . .	66
4.2	Diagramme de cas d'utilisation . . . . .	67
4.3	Comparaison entre SOA et applications monolithiques . . . . .	84

# Table des figures

1.1	Passage du monolithique au SOA . . . . .	16
1.2	Structure de la SOA . . . . .	17
1.3	Les caractéristiques d'un service . . . . .	18
1.4	Fonctionnement de RPC . . . . .	21
1.5	Fonctionnement de RMI . . . . .	22
1.6	L'architecture EJB . . . . .	22
1.7	Architecture CORBA . . . . .	24
1.8	Architecture MOM . . . . .	25
1.9	Du monolithique aux microservices . . . . .	27
1.10	Architecture SOA . . . . .	28
1.11	Monolithe/SOA/Microservices . . . . .	29
1.12	La portée des microservices par rapport à la SOA . . . . .	29
2.1	Architecture à base des services web . . . . .	32
2.2	Structure d'un fichier WSDL . . . . .	34
2.3	Exemple de document WSDL . . . . .	35
2.4	Fichier SOAP . . . . .	36
2.5	Orchestration avec BPEL . . . . .	38
3.1	Evolution du cloud computing . . . . .	41
3.2	Infrastructure en tant que Service . . . . .	43
3.3	Plateforme en tant que service . . . . .	44
3.4	Logiciel en tant que service . . . . .	45
4.1	Page d'accueil Sunstone . . . . .	56
4.2	Interface de gestion de OpenNebula . . . . .	57
4.3	Création d'un host étape 1 . . . . .	58
4.4	Création d'un host étape 2 . . . . .	58
4.5	Création d'image étape 1 . . . . .	59
4.6	Création d'image étape 2 . . . . .	60
4.7	Création VM étape 1 . . . . .	61
4.8	Création VM étape 2 . . . . .	62
4.9	Bureau de la machine virtuelle . . . . .	63
4.10	Machine virtuelle Linux Mint . . . . .	64
4.11	Diagramme de classe de l'application . . . . .	65
4.12	Processus d'inscription . . . . .	68
4.13	Processus d'emprunt . . . . .	68
4.14	Processus de remise . . . . .	68

4.15	Fonctionnement de la passerelle Spring Cloud (gateway)	71
4.16	Architecture microservice de l'application	72
4.17	Interface de Postman	73
4.18	Controller du microservice student-service	74
4.19	Controller du microservice book-service	75
4.20	Liste des étudiants	75
4.21	Liste des étudiants	76
4.22	Insertion d'un nouvel étudiant dans la liste	76
4.23	Liste des étudiants après insertion d'un élément	77
4.24	Insertion d'un nouveau livre	77
4.25	Liste des livres après l'insertion	78
4.26	Suppression d'un étudiant	78
4.27	Liste des étudiants après la suppression	79
4.28	Suppression d'un livre	79
4.29	Liste des livres après la suppression	80
4.30	Service d'enregistrement des services	81
4.31	Résultat d'un emprunt	82
4.32	Suppression d'un emprunt	82
4.33	liste des requêtes semi-dynamique du service Gateway	83
4.34	Exemple de requête à partir du gateway	83
4.35	Les packets opennebula sur UBUNTU	88

# Liste Des Abréviations

**SOA** : Service Oriented Architecture

**SI** : Système d'Information

**RMI** : Remote Method Invocation

**RPC** : Remote Protocol Call

**WSDL** : Web Service Description Language

**W3C** : World Wide Web Consortium

**UDDI** : Universal Description Discovery Integration

**SOAP** : Simple Object Access Protocol

**BPEL** : Business Process Execution Language

**WS-BPEL** : Web Services Business Process Execution Language

**XML** : Extensible Markup Language

**HTTP** : Hypertext Transport Protocol

**TCP/IP** : Transmission Control Protocol/Internet Protocol

**EJB** : Enterprise Java Beans

**Java EE** : Java Enterprise Edition

**API** : Application Programming Interface

**UDP/IP** : User Datagram Protocol

**DCOM** : Distributed Component Object Model

**COM** :Component Object Model

**CORBA** :Common Object Request Broker Architecture

**IDL** :Interface Definition Language

**ORB** : Object Request Brocker

**MOM** : Middleware Oriented Message

**OMG** : Object Management Group

**NASSL** : Network-Accessible Service Specification Language

**ESB** : Entreprise Service Bus

**IBM** :International Business Machines

**JSON** :JavaScript Object Notation

# Introduction Générale

Pour des raisons de performance et de concurrence, les entreprises doivent se doter de système d'information de plus en plus efficace et flexible pour de meilleures relations avec leur environnement. Le système d'information était traditionnellement composé d'applications monolithiques ; ce sont des applications dont l'ensemble des fonctionnalités est inclut dans un seul bloc applicatif et développé avec la même technologie. Ces types d'architectures sont difficiles à faire évoluer à cause de leur manque d'agilité, et une possibilité d'intégration très limitée et coûteuse. Ainsi, la maintenance sur une partie de l'application impose un redéploiement complet engendrant une forte consommation en temps et en ressources ; ce qui impacte l'évolution de l'entreprise. Pour résoudre ces problèmes, le groupe GARTNER propose en 1996 l'Architecture Orientée Service (SOA) qui consiste globalement à convertir les applications en un ensemble de services autonomes plus simples, utilisés pour construire les processus métier de l'entreprise. Cette architecture s'est avérée efficace du fait de sa maniabilité. En effet elle accorde plus d'importance à la réutilisation des fonctionnalités du système d'information (SI) de l'entreprise et facilite son adaptation aux changements imprévus. La décomposition du système en services facilite sa maintenance et sa résilience. La matérialisation de ce concept se fait à travers des technologies standardisées résolvant les problèmes d'intégration et d'agilité. L'apparition des services web a permis véritablement de mettre en oeuvre tous les principes de l'Architecture Orienté Services (SOA).

Ainsi, la SOA permet aux entreprises d'avoir un SI beaucoup plus souple et adaptatif. Cependant ce système devient plus exigeant en disponibilité de services et de stratégie d'accès.

L'essor technologique a permis l'apparition des centres de données avec des capacités de stockage et de traitement énormes. Dans un autre volet nous avons eu l'apparition du concept de virtualisation et de ses outils. Cette avancée technologique a permis de concrétiser les idées du cloud computing qui avait été énoncé au début des années 60. Muni de serveurs virtualisés, le cloud offre des services scalables, et souples. Le cloud computing avec son service à la demande a révolutionné l'offre technologique au sein des entreprises et des institutions.

Le cloud computing est une nouvelle orientation pour les entreprises qui veulent externaliser l'ensemble ou une partie de leur système d'information dans un contexte d'urbanisation du SI. Dans ce projet, nous avons effectué une étude de l'architecture orientée service SOA et des concepts liés au Cloud Computing. L'objectif est de tenter l'exploitation de la technologie du cloud computing pour la mise en place d'un système à base d'SOA. Nous allons d'abord installé et configuré OpenNebula ( un système de gestion de cloud Computing) qui va permettre la mise en place d'un cloud privé minimisé. Ensuite nous ferons la conception et la réalisation d'un système à base d'SOA, de gestion de bibliothèque universitaire . Les services seront mis en place en tant que microservices. Ces derniers seront implémentés grâce au Framework Spring Boot et la plateforme Spring Cloud de Netflix qui offre les outils nécessaires au déploiement et à l'exécution.

C'est sur ce principe que se base ce mémoire que nous avons structuré comme suit :

- **Chapitre 1 : L'architecture orientée services.** Ce qui sera évoqué dans ce chapitre est la

notion réelle de SOA, son fonctionnement. Ensuite les technologies utilisées pour sa mise en place, avant de terminer avec les microservices qui sont une évolution de la SOA.

- **Chapitre 2** : intitulé « **Les Services Web** » sera composé d'une partie explicative sur le fonctionnement de son architecture. Une seconde partie sur la structure interne d'un service web. Nous expliquerons les Service Web SOAP et les Services Web REST. Et finir par l'orchestration de BPEL des services web dans l'entreprise.

- **Chapitre 3** : il sera basé entièrement sur le **Cloud Computing**. Nous y reviendrons sur les différents types de clouds, leurs fournisseurs et leur « plus-value » pour les utilisateurs. Nous détaillerons ainsi le fonctionnement du cloud et ses modes de déploiement. Nous étudierons les solutions libres (open-source) de cloud infrastructurel que sont Eucalyptus, OpenStack et Opennebula.

- **Chapitre 4** : Nous consacrerons ce dernier à la pratique. C'est-à-dire **la mise en place de notre architecture basée sur le cloud**. Nous exposerons d'abord les outils de travail, ensuite la conception architecturale et finir avec la l'application de gestion d'une bibliothèque qui respecte les normes des microservices basée sur le cloud.

Nous terminerons ce mémoire par une conclusion générale qui va faire le bilan de notre recherche faite en amont. Et les différentes perspectives dans l'éventualité de la continuation de ce travail.

# Chapitre 1

## L'Architecture Orientée Service

### 1.1 Introduction

Avec les perpétuels et rapides changements de besoins technologiques de la société, les entreprises sont confrontées à des défis d'adaptabilité, d'agilité et surtout d'efficacité. Le terme SOA proposé par le Gartner Group en 1996, est apparue pour résoudre des problématiques d'interopérabilité dans les technologies en entreprise. Son objectif est de faciliter l'intégration de nouvelles applications au sein de la structure IT (Information Technology) de l'entreprise en optimisant ses échanges et son fonctionnement. L'architecture orientée services est donc née pour donner à l'entreprise l'agilité nécessaire pour évoluer dans un environnement économique et concurrentiel en perpétuel changement. Basée sur des principes, des approches technologiques et des outils de mise en œuvre, l'architecture orientée services est aujourd'hui rendu possible grâce aux services web. Ainsi ce chapitre sera consacré à l'étude des fondements de l'architecture orientée services, les approches technologiques de sa mise en œuvre, et étudier les éventuelles améliorations en son sein.

### 1.2 Comprendre l'architecture SOA

#### 1.2.1 Aperçu sur l'avant SOA : Les applications monolithiques

Une application monolithique est une application dont la totalité ou la plupart des fonctionnalités se trouve dans un seul processus ou conteneur et est intégrée dans des couches internes ou des bibliothèques[1]. Une application monolithique est autonome et indépendante des autres applications informatiques. Elle se compose de quatre éléments principaux : une interface utilisateur, des logiques métiers, une interface de données et une base de données.

Cette architecture présente des avantages grâce à :

- sa simplicité au niveau de sa construction, son test et son déploiement.
- Problèmes transversaux : Avec une base de code unique, les applications monolithiques peuvent facilement gérer les problèmes transversaux, tels que l'enregistrement des logs, la configuration et le contrôle des performances.
- Performances : Les composants d'une architecture unifiée partagent la mémoire vive, ce qui est plus rapide que la communication de service à service, dépendante de mécanismes comme l'IPC (Communications inter-processus) ou autres. Cependant, les applications unifiées ne répondent pas aux besoins des entreprises en matière d'évolutivité. Notamment son couplage étroit ( les composants deviennent étroitement liés et enchevêtrés). Les monolithes connaissent d'autres désavantages tels :
  - Fiabilité : un problème dans un module peut affecter l'application toute entière.



-Problèmes de maintenance : à chaque mise à jour l'application doit être entièrement redéployée ce qui engendre des coûts supplémentaires.

-Pile technologique : une application monolithique est développée avec une seule technologie ce qui limite les besoins d'évolutivité et d'intégration dans le temps.[2]

La SOA est venue pour résoudre les problèmes liés aux applications monolithiques en facilitant l'intégration de nouvelles technologies et d'applications dans le système d'information. La figure ci-dessous illustre le passage du monolithique au SOA.

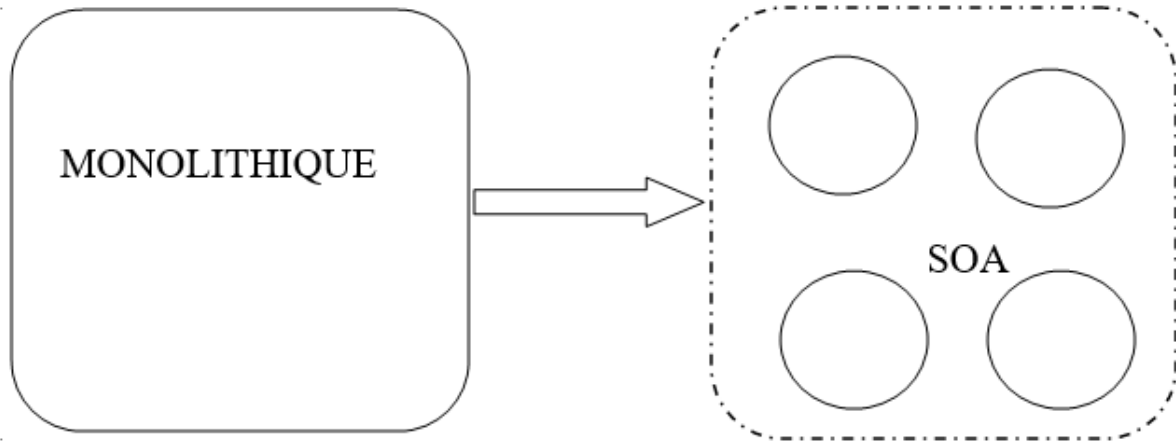


FIGURE 1.1 – Passage du monolithique au SOA

La figure 2 présente un aperçu global de la structure d'une SOA[3]

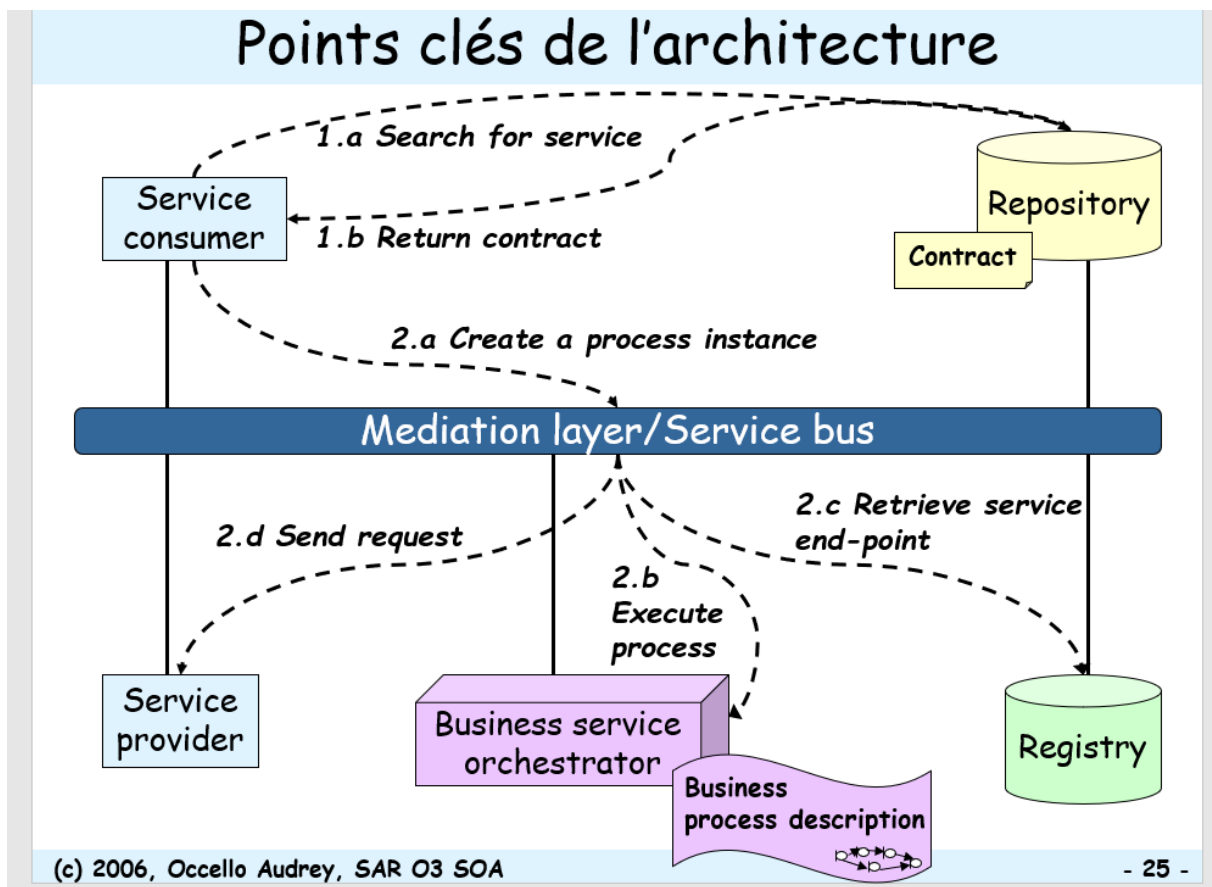


FIGURE 1.2 – Structure de la SOA

### 1.2.2 Définition de la SOA

L'architecture orientée services est une architecture logicielle s'appuyant sur un ensemble de services simples. L'objectif est de décomposer une fonctionnalité en un ensemble de fonctions basiques, appelées services; fournis par des composants logiciels et de décrire finement le schéma d'interaction entre ces services. On cesse de construire la vie de l'entreprise autour d'applications, pour faire en sorte de construire une architecture logicielle globale décomposée en services correspondant aux processus métiers de l'entreprise[4]. La construction d'applications dans une SOA se base sur la chorégraphie et l'orchestration des services dans des applications composites et de les invoquer par des protocoles standards. L'architecture orientée services ajoute l'aspect d'agilité à l'architecture, ce qui permet de faire face aux changements dans les systèmes en utilisant une couche de configuration plutôt que d'avoir constamment à redévelopper ces systèmes.[5]

### 1.2.3 Notion de Service

La notion de service dans une architecture orientée services fait référence à une fonction encapsulée dans un composant interrogeable via une requête et fournissant une ou plusieurs réponses. La requête est le plus souvent paramétrée. Un service au sens SOA (Architecture Orientée Service) a des propriétés qui lui sont propres qui sont d'ailleurs référencées dans la figure 2 :

Un service doit être indépendant de tout protocole de transport. Ainsi, l'invocation d'un service doit pouvoir se faire en utilisant http, TCP/IP, RMI, ... Le principal protocole de transport des services aujourd'hui est le http.

Un service doit être réutilisable. La réutilisabilité d'un service se mesure à son indépendance de la

logique de son métier c'est-à-dire le service doit retourner des réponses les plus neutres possibles

Le service doit également être interopérable et utilisant des standards car faisant partie d'un système qui de plus est généralement hétérogène. Le service est conçu pour être joignable par d'autres composants ou applications. Ainsi, suivant l'architecture orientée services, un service doit avoir une description, un annuaire de publication où il est publié et un moyen approprié pour son invocation.

Un service doit être autonome : avoir ses propres ressources, bibliothèques, conteneurs

Un service doit être abstrait : publier seulement le nécessaire à l'invocation des services. Le comportement interne du service ne doit pas être connu du demandeur de service

Couplage faible : des services indépendants et faiblement couplés aux autres composants du système d'information

Sans état : aucune requête ne doit dépendre de la réponse d'une autre requête

La composabilité : on doit être capable de regrouper plusieurs services dans un processus métier.

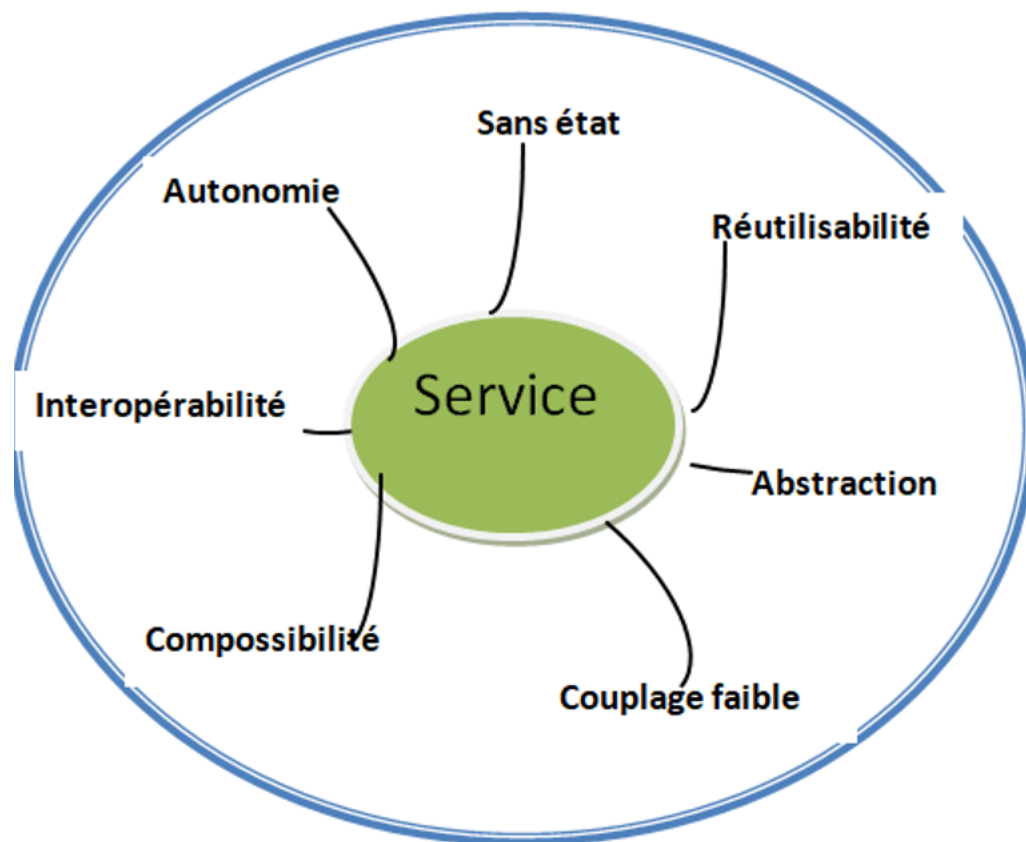


FIGURE 1.3 – Les caractéristiques d'un service

### 1.2.3.1 La description de service

La description du service consiste à décrire les paramètres d'entrée du service et le format et le type des données retournées. Le principal format de description de services est WSDL (Web Services Description Language), normalisé par le W3C.

### 1.2.3.2 La publication de service

Après la description du service, le client ne peut pas utiliser le service car il ignore son existence. C'est pour cela qu'on a besoin d'un annuaire de service où publier les informations relatives à ce service.

L'annuaire stocke à la fois des informations techniques et formelle relatives aux services tels que l'adresse d'accès, la description détaillée de leur fonction, ainsi que les métadonnées. Ces informations doivent couvrir l'essentiel de l'ensemble des fonctions nécessaires à la bonne gestion du cycle de vie des services et à la mise en œuvre d'une véritable gouvernance dans une architecture orientée services.[6]

### 1.2.3.3 L'invocation du service

L'invocation concerne la connexion et l'interaction du client avec le service. Cette interaction se fait avec des techniques et des protocoles basés sur des standards afin de faciliter l'accès et l'utilisation des services concernés. Les technologies les plus utilisés sont SOAP et REST(Services Web); mais les entreprises peuvent décider d'utiliser le protocole qui leur convient le mieux. Nous détaillerons plus amplement la notion d'invocation de services dans le prochain chapitre.

## 1.2.4 Les principes de l'Architecture Orientée Service

### 1.2.4.1 Utilisation de standards

L'accès à un service dans une architecture orientée service doit être indépendant de tout langage de programmation et de toute plateforme de mise en œuvre. Un processus développé dans un langage quelconque et voulant utiliser un service développé dans un autre langage doit être rendu possible. Ainsi, il faut trouver un mécanisme de communication client fournisseur, compréhensible par toute technologie de mise en œuvre. Le langage de balisage XML est l'actuel standard de base utilisé. Selon la stratégie de mise en œuvre, on trouve d'autre standard dérivée du XML comme :

- Le WSDL (Web Services Description Language) : est un Langage XML de description de services web
- L'UDDI (Universal Description Discovery and Integration) : est un annuaire de services fondé sur XML particulièrement destiné aux services Web.
- SOAP (Simple Object Access Protocol) : est un protocole de communication basé sur XML permettant aux services de s'échanger des informations via http ou smtp.
- BPEL ou WS-BPEL (Web Services Business Process Execution Language) : est un langage à base de XML qui définit les règles de dialogue entre les services. Il permet l'orchestration des services web.

### 1.2.4.2 Amélioration technologique de l'entreprise

L'architecture orientée service est venu répondre aux besoins d'évolution et de changement technologique de l'entreprise. Elle vise aussi à réduire les coûts et délai de mise en œuvre des systèmes d'information et de leur maintenance. Le système existant est généralement composé d'applications hétérogènes. Certaines de ces applications sont propriétaires donc le code non accessible à l'entreprise. C'est pour toutes ces raisons que l'architecture orientée services opte pour une extension de l'existant et évite de toucher au code de l'existant. Mais il peut arriver qu'il y ait des modifications structurelles du code existant en utilisant des techniques de re-ingéniering ; par exemple transformer un code procédural en un code orienté objet afin de l'inclure dans une nouvelle technologie.

### 1.2.4.3 Découplage entre fournisseur et consommateur de Services

Un service est indépendant des protocoles de transmission de requêtes ou de résultats. Ceci signifie que le service peut être appelé en utilisant n'importe quel protocole de transmission (HTTP, TCP/IP, RMI, et ainsi de suite). Le point d'interaction entre un consommateur de service et le service lui-même est les message SOAP. Aujourd'hui, le protocole primaire de transport des données entre les services est le http.

## 1.3 Les technologies utilisés pour la mise en oeuvre de la SOA

### 1.3.1 Les technologies à base d'objets distants

#### 1.3.1.1 RPC

Le Remote Procedure Call (en français « appel de procédure à distance ») est un outil central permettant de réaliser des structures opérationnelles et basées sur la division du travail dans des réseaux et des architectures client-serveur. Selon les informaticiens Andrew Birell et Bruce Nelson, RPC est un mécanisme synchrone « transférant le flux de contrôle et les données sous la forme d'un appel de procédure entre deux espaces d'adressage via un réseau à bande étroite ». RPC est donc un protocole réseau permettant de faire des appels de procédure sur un ordinateur distant à l'aide d'un serveur d'applications. Ainsi, les appels RPC se déroulent toujours selon un modèle défini : un client contacte par exemple un serveur de base de données central lors de la recherche d'une pièce de rechange. Le serveur situé à distance vérifie ensuite les données disponibles et envoie le résultat au client. Ce dernier traite les données obtenues et affiche par exemple une liste des données disponibles dans le logiciel de gestion. Des instances spécifiques appelées « stubs » participent à l'implémentation d'un Remote Procedure Call côté expéditeur et destinataire. Côté client, le stub client vient remplacer la procédure du serveur à distance. Côté serveur, le stub serveur remplace le code client appelant. Les stubs simulent une unité locale fonctionnelle d'un point de vue opérationnel en dissimulant l'« éloignement » du code à l'autre côté. Par ailleurs, ils fonctionnent comme des interfaces pour la procédure. Le déroulement habituel d'un appel RPC est caractérisé par les étapes suivantes :

- le code client appelle une procédure stub (stub client local).
- À partir des paramètres transmis par l'appel de procédure, le stub client génère un message prêt à l'envoi suivant le protocole RPC. Une sérialisation a lieu lors du transfert pendant laquelle les données structurées sont transférées sous une forme séquentielle. Ce processus de traduction est également appelé marshalling (de l'anglais « to marshal », ce qui signifie en français « lister », « trier »).
- Le stub client contacte ensuite le système de communication de l'ordinateur local qui utilise TCP/IP ou UDP/IP pour l'échange de messages qui se déroule ensuite entre le client et le serveur.
- Une fois le message envoyé parvenu au destinataire, le stub serveur exécute un demarshalling ou unmarshalling, en décompressant les paramètres contenus dans le message (sur la base du protocole RPC).
- Le stub serveur transmet les paramètres décodés et assure ainsi l'appel local d'une procédure de serveur.
- La valeur de fonction qui en résulte est communiquée au stub serveur.
- Le processus est alors exécuté dans le sens inverse : génération d'un message prêt à l'envoi conformément au protocole RPC, échange de messages entre le serveur et le client, puis transport de la valeur de retour au code client en attente. L'application est poursuivie sur l'ordinateur d'origine.[7]

La figure ci-dessous résume le fonctionnement global du protocole RPC.[7]

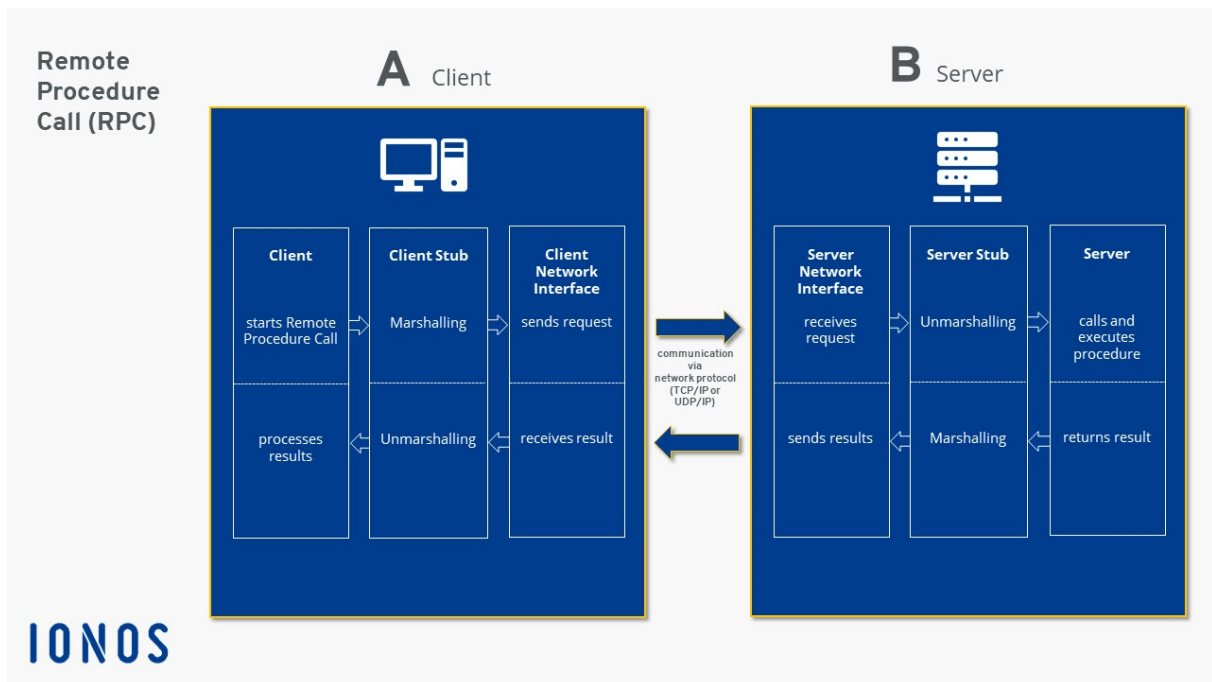


FIGURE 1.4 – Fonctionnement de RPC

Bien que traitant la communication interprocessus de façon fiable et permettant une modularisation considérable, RPC présente néanmoins des inconvénients notables. Notamment son manque d'utilisation de standards. D'autre part, les niveaux de transfert et de transmission des systèmes basés sur le RPC induisent des pertes de vitesse, ce qui n'est pas le cas avec les appels de procédure purement locaux. Le client et le serveur utilisant des environnements d'exécution différents pour leurs routines respectives, l'utilisation des ressources (par ex. des fichiers) est plus complexe.

### 1.3.1.2 RMI

RMI est une API java qui permet à un objet d'appeler un objet sur un autre objet qui existe dans un espace d'adressage différent qui peut se trouver sur la même machine ou sur une autre. RMI permet à un objet s'exécutant dans une JVM (Java Virtual Machine présente sur un ordinateur (côté client) peut invoquer des méthodes sur un objet présent dans une autre JVM (côté serveur). RMI crée un objet serveur distant public qui permet les communications côté client et côté serveur via des appels de méthode simples sur l'objet serveur. A la différence de RPC qui lui est procédurale, elle se base sur l'orienté objet. La communication entre les deux machines est gérée à l'aide de deux objets intermédiaires. L'objet stub et l'objet skeleton (serveur). L'objet stub sur l'ordinateur client crée un bloc d'informations et envoie ces informations au serveur. Le bloc se compose de :

- un identifiant de l'objet distant à utiliser
- un nom de la méthode à invoquer
- les paramètres de la JVM distante

L'objet skeleton transmet la demande de l'objet stub à l'objet distant. Il effectue les tâches suivantes : Il appelle la méthode souhaitée sur l'objet réel présent sur le serveur. Il transmet les paramètres reçus de l'objet stub à la méthode. [8] [9]

La figure ci-dessous illustre un peu le fonctionnement de RMI.

Les inconvénients de java RMI sont pratiquement les mêmes que ceux du RPC notamment la latence des réponses, les problèmes de sécurité, etc. . .

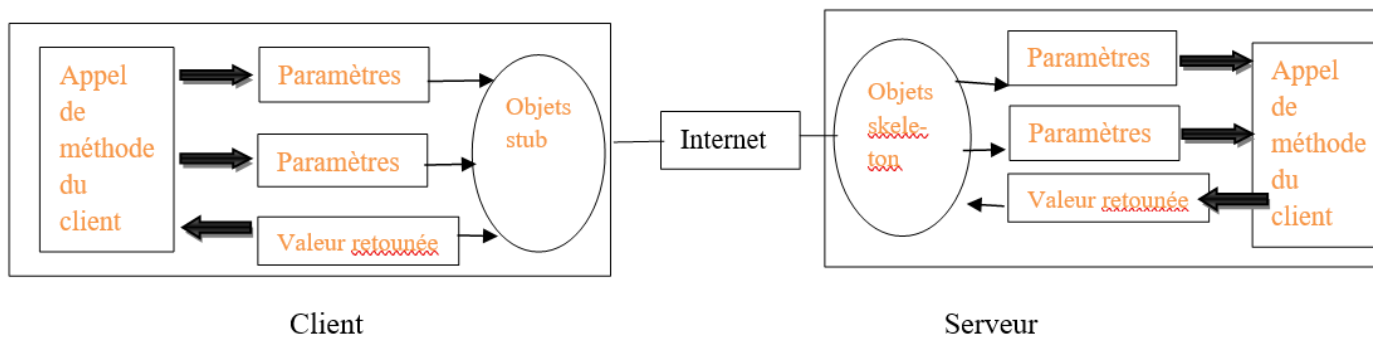


FIGURE 1.5 – Fonctionnement de RMI

### 1.3.2 Les technologies à base de composants logiciels

#### 1.3.2.1 EJB

Enterprise JavaBeans (EJB) est une architecture de composants logiciels côté serveur pour la plateforme de développement Java EE.[10] Cette architecture propose un cadre pour créer des composants distribués comme le présente la figure 6 (c'est-à-dire déployés sur des serveurs distants) écrit en langage de programmation Java hébergés au sein d'un serveur applicatif permettant de représenter des données (EJB dit entité), de proposer des services avec ou sans conservation d'état entre les appels (EJB dit session), ou encore d'accomplir des tâches de manière asynchrone (EJB dit message). Tous les EJB peuvent évoluer dans un contexte transactionnel comme l'illustre la figure ci-dessous.[11] [12]

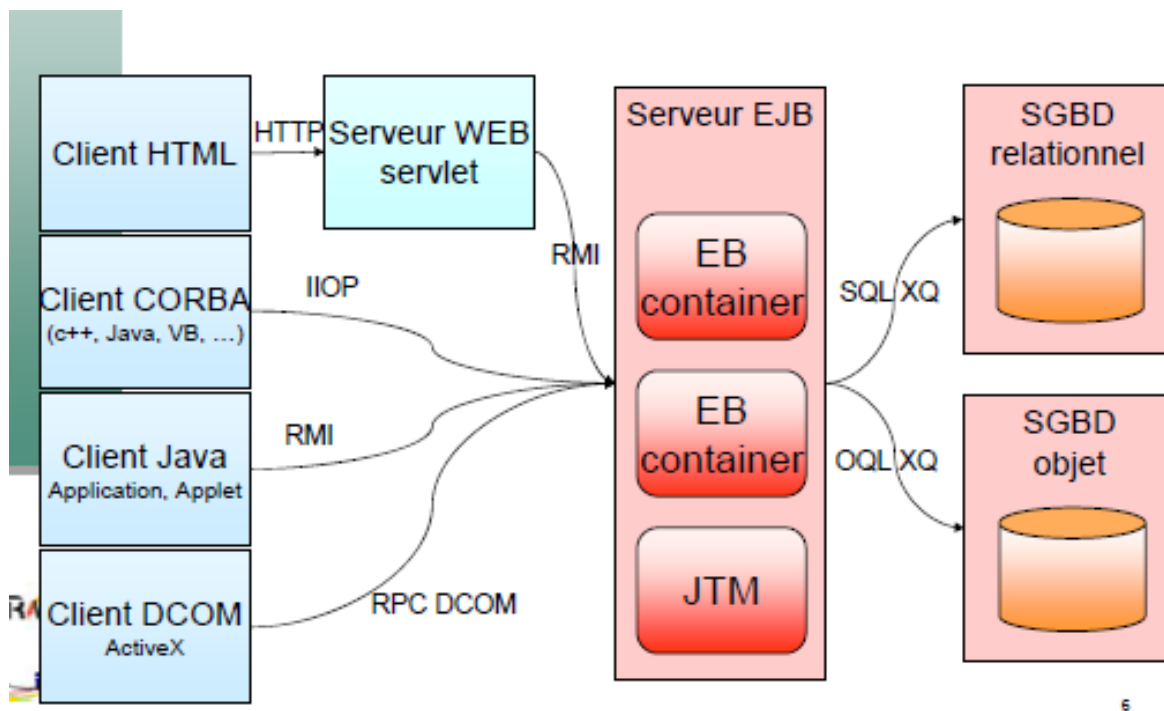


FIGURE 1.6 – L'architecture EJB

### 1.3.2.2 DCOM

Distributed Component Object Model (abr. DCOM) est une technique propriétaire de Microsoft qui permet la communication entre des composants logiciels distribués au sein d'un réseau informatique. DCOM, appelé à l'origine « Network OLE », étend COM et fournit le substrat sous l'infrastructure du serveur d'application COM+. Il a été rendu obsolète par Microsoft .NET. L'addition du « D » à COM est due à l'utilisation intensive de DCE/RPC, plus précisément sa version Microsoft, connue sous le nom de MSRPC. En termes d'extensions ajoutées à COM, DCOM devait résoudre les problèmes de : Marshalling :sérialisation et désérialisation des paramètres et des valeurs de retour des appels de méthode à travers le réseau. Ramasse-miettes distribué s'assurant que les références des clients des interfaces soient libérées quand, par exemple, le processus client plantait, ou la connexion réseau était perdue. Un des facteurs clés de la résolution de ces problèmes est l'utilisation de DCE/RPC comme le mécanisme RPC derrière DCOM. DCE/RPC possède des règles strictement définies à propos de la sérialisation et de la responsabilité de la libération de la mémoire.[1]

### 1.3.3 Les middlewares

En architecture informatique, un middleware est un logiciel tiers qui crée un réseau d'échange d'information entre différentes applications informatiques. Son rôle est de supporter un modèle de développement de haut niveau focalisé sur le métier en repartit. Il assure la portabilité et l'interopérabilité entre les applications. Le paradigme orienté objet est centre du fonctionnement des middlewares[13]

- L'orienté objet : les technologies intergicielles proposées dans l'industrie proposent la programmation orientée objet en environnement repartit. Celui -ci est axé sur l'encapsulation du code et des données, la modularité par la notion de classe, la spécialisation par l'héritage et le polymorphisme.
- L'interopérabilité : elle est assurée par l'utilisation de protocole standard. C'est l'exemple d'IIOIP (Internet Interoperable Object Protocol)
- La portabilité du code : Avoir un code portable permet de s'affranchir de l'hétérogénéité des infrastructures matérielles et logicielles sur lesquelles il doit s'exécuter Java utilise la JVM (Java Virtual Machine) pour générer le code machine compréhensible par toutes les plateformes d'exécution.

#### 1.3.3.1 Etude de quelques middlewares

#### 1.3.3.2 CORBA

CORBA a été mis en place par OMG (Object Management Group). OMG prône l'utilisation d'une approche basée sur les objets pour offrir une vue unique d'un système distribué et hétérogène. CORBA est de type ORB (Object Request Broker ou bus d'objets distribués). Un ORB assure l'interaction entre les objets distants (déploiement d'objets sur des sites distants, la localisation de leurs instances, l'invocation distante de méthodes) de manière transparente[14]. CORBA fait une séparation entre interface et implémentation d'objet répartis. Le langage IDL (Interface Definition Language) sert à exprimer l'interface fonctionnelle des objets indépendamment de toute considération liée à leur implémentation, offrant ainsi une indépendance vis-à-vis de l'hétérogénéité des réseaux, des machines, des systèmes d'exploitation et des langages de programmations. L'interface d'un objet liste l'ensemble des opérations (les méthodes) fournies, ainsi que leur signature fortement typée (le résultat, les paramètres, exceptions). Il permet d'exprimer sous forme d'un contrat la coopération entre le fournisseur et les utilisateurs de services.[13] La compilation d'un contrat IDL crée une souche (talon) IDL (ou SII, interface d'invocation statique) dans l'environnement de programmation du client et une souche (squelette) IDL (ou SSI, interface de squelette statique) dans l'environnement de programmation du fournisseur. Le client invoque localement la souche



pour accéder à l'objet. La souche construit alors la requête qui est ensuite transportée par le bus logiciel pour être délivrée au squelette IDL (coté serveur donc) qui la délègue à l'objet.

Ainsi, une requête client se déroule comme suit :

1. Le client détient une référence sur un objet CORBA qui permet de le localiser sur le bus.
2. Le client dispose de l'interface de l'objet CORBA (type abstrait de l'objet CORBA) qui définit ses opérations et ses attributs (exprimés dans le langage IDL : le talon sur le schéma).
3. Le client réalise une requête (invoque) une opération sur l'objet CORBA.
4. Le bus CORBA achemine cette requête vers l'objet CORBA tout en masquant les problèmes d'hétérogénéité liés aux langages, systèmes d'exploitation, machines, réseaux.
5. L'objet CORBA est associé à un objet d'implantation
6. Le serveur détient l'objet d'implantation qui code l'objet CORBA (cette implantation pouvant évoluer au cours du temps) et gère son état temporaire. La figure 1.7 montre un aperçu de l'architecture CORBA[9]

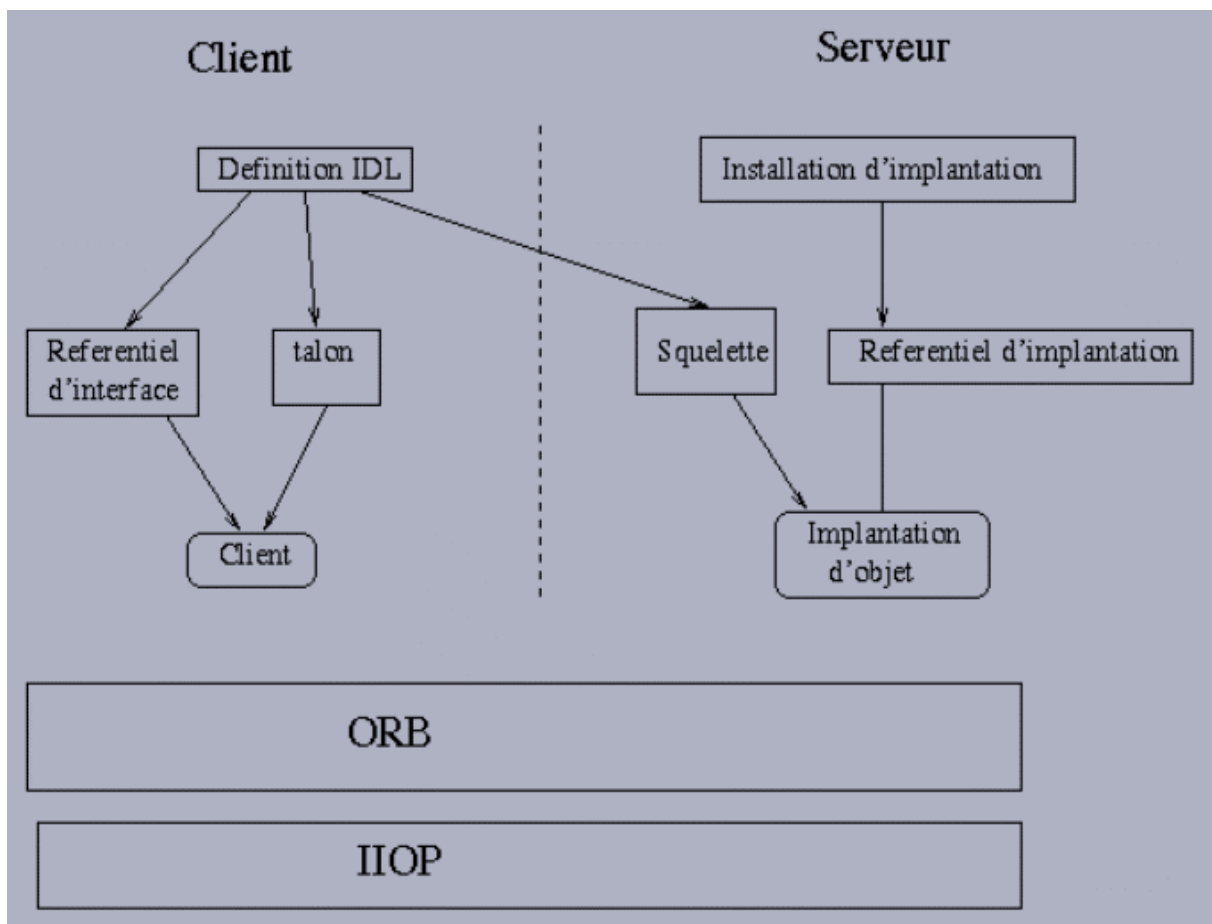


FIGURE 1.7 – Architecture CORBA

### 1.3.3.3 MOM

Un MOM est un élément clé de l'architecture ESB, car il fournit les fondements du réseau de canaux virtuels qu'un ESB utilise pour acheminer les messages dans une entreprise étendue et au-delà.[15] L'intergiciel orienté message est un concept qui implique le passage de données entre des applications à l'aide d'un canal de communication qui transporte des unités d'information autonomes (messages). Dans un environnement de communication basé sur MOM, les messages sont généralement envoyés et reçus

de manière asynchrone. En utilisant des communications basées sur des messages, les applications sont découplées de manière abstraite ; les expéditeurs et les destinataires ne se connaissent jamais. Au lieu, ils envoient et reçoivent des messages vers et depuis le système de messagerie. Il est de la responsabilité du système de messagerie (MOM) d'acheminer les messages vers leurs destinations prévues. Dans un système de messagerie, une application utilise une API pour communiquer via un client de messagerie fourni par le fournisseur MOM[16]. Le client de messagerie envoie et reçoit des messages via un système de messagerie, comme illustré à la figure 1.8.

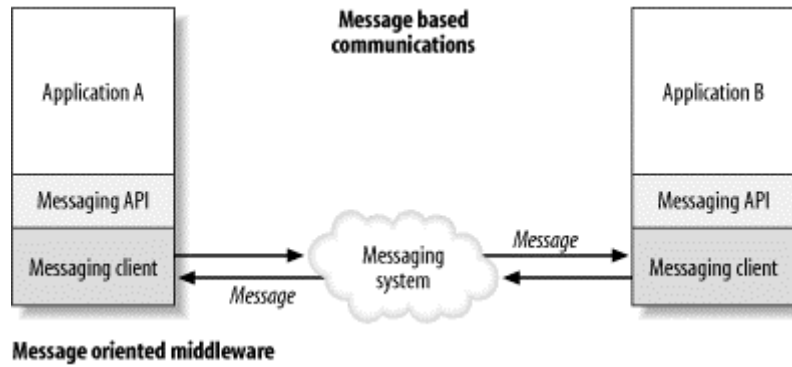


FIGURE 1.8 – Architecture MOM

### 1.3.4 Les technologies à base de web services

Les services Web sont des applications modulaires autonomes qui peuvent être décrites, publiées, localisées et appelées sur un réseau, généralement le World Wide Web. L'architecture des services Web décrit trois rôles : fournisseur de services, demandeur de services et courtier de services ; et trois opérations de base : publier, rechercher et lier. Un composant réseau peut jouer tout ou une partie de ces rôles. Deux documents distincts décrivent les services Web : un document WDS (Well-Defined Service) décrit des informations de service non opérationnelles, telles que la catégorie de service, la description du service et la date d'expiration, ainsi que des informations commerciales sur le fournisseur de services, telles que le nom de l'entreprise, l'adresse, et coordonnées. Un document NASSL (Network-Accessible Service Specification Language) décrit des informations opérationnelles sur le service, telles que l'interface de service, les détails d'implémentation, le protocole d'accès et les points de contact. Une implémentation d'architecture de services Web doit permettre une sécurité incrémentielle et des modèles de qualité de service facilités par la configuration d'un ensemble de prérequis environnementaux (par exemple, mécanisme d'authentification, facturation, etc.) pour contrôler et gérer les interactions.

Nous détaillerons le volet services web dans le prochain chapitre.

### 1.3.5 ESB

#### 1.3.5.1 Définition d'un ESB

Un ESB, ou bus de service d'entreprise, est un modèle dans lequel un composant logiciel centralisé effectue des intégrations aux systèmes backend (et des traductions de modèles de données, une connectivité approfondie, un routage et des demandes) et rend ces intégrations et traductions disponibles en tant qu'interfaces de service pour une réutilisation par de nouvelles applications [17]. Le modèle ESB est généralement implémenté à l'aide d'un runtime d'intégration et d'un ensemble d'outils spécialement conçus qui garantissent la meilleure productivité possible.

#### 1.3.5.2 ESB et SOA

Un ESB est un composant essentiel de l'architecture SOA, ou architecture orientée services, une architecture qui a émergé à la fin des années 1990. SOA définit un moyen de rendre les composants logiciels réutilisables via des interfaces de service. Ces interfaces utilisent des normes de communication communes de telle sorte qu'elles peuvent être rapidement intégrées dans de nouvelles applications sans avoir à effectuer une intégration profonde à chaque fois.[18] Chaque service d'une SOA incarne les intégrations de code et de données nécessaires pour exécuter une fonction commerciale complète et discrète (par exemple, vérification du crédit d'un client, calcul d'un paiement mensuel de prêt ou traitement d'une demande de prêt hypothécaire). Les interfaces de services fournissent un couplage lâche, ce qui signifie qu'elles peuvent être appelées avec peu ou pas de connaissances sur la façon dont l'intégration est mise en œuvre en dessous. Les services sont exposés à l'aide de protocoles réseau standard - tels que SOAP (protocole d'accès aux objets simples) / HTTP ou JSON / HTTP - pour envoyer des demandes de lecture ou de modification de données. Les services sont publiés de manière à permettre aux développeurs de les trouver rapidement et de les réutiliser pour assembler de nouvelles applications. Ces services peuvent être créés à partir de zéro, mais sont souvent créés en exposant les fonctions des anciens systèmes d'enregistrement en tant qu'interfaces de service. C'est là que le besoin d'un ESB se fait sentir. Les anciens systèmes et systèmes d'enregistrement utilisent généralement d'anciens protocoles et des formats de données propriétaires qui doivent être traduits et intégrés pour fonctionner avec les protocoles de réseau SOA. Un ESB effectue ces traductions et intégrations à la volée. Vous pouvez implémenter une SOA sans ESB, mais les propriétaires d'applications devraient chacun trouver leur propre moyen d'exposer les interfaces de service, ce qui

représente beaucoup de travail (même si les interfaces sont éventuellement réutilisables) et crée un défi de maintenance important à l'avenir.

En théorie, un ESB centralisé offre la possibilité de standardiser - et de simplifier considérablement la communication et l'intégration des services dans toute l'entreprise. Les coûts matériels et logiciels peuvent être partagés, l'approvisionnement des serveurs ne doit être effectué qu'une seule fois et une seule équipe de spécialistes peut être chargée (et, si nécessaire, formée) de développer et de maintenir les intégrations. Les développeurs peuvent utiliser un protocole unique pour " parler " à l'ESB et émettre des commandes qui dirigent les interactions entre les services et laissent à l'ESB le soin de traduire les commandes, d'acheminer les messages et de transformer les données selon les besoins pour exécuter les commandes. Cela permettrait aux développeurs de passer considérablement moins de temps à intégrer et beaucoup plus de temps à configurer et à améliorer leurs applications. Et la possibilité de réutiliser ces intégrations d'un projet à l'autre offrait le potentiel de gains de productivité et d'économies encore plus importants en aval.

## 1.4 L'Architecture Microservices : la SOA évoluée ?

Les microservices désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Contrairement à une approche monolithique classique, selon laquelle tous les composants forment une entité indissociable comme le montre la figure suivante[19], les microservices fonctionnent en synergie pour accomplir les mêmes tâches telles visibles sur la même figure, tout en étant séparés. Chacun de ces composants ou processus est un microservice. Granulaire et léger, ce type de développement logiciel permet d'utiliser un processus similaire dans plusieurs applications. Il s'agit d'un élément essentiel pour optimiser le développement des applications en vue de l'adoption d'un modèle Cloud-native. Les microservices permettent aussi de restructurer les équipes de développement et la communication entre les services pour mieux se préparer aux inévitables pannes, aux évolutions futures et à l'intégration de nouvelles fonctions.[19]

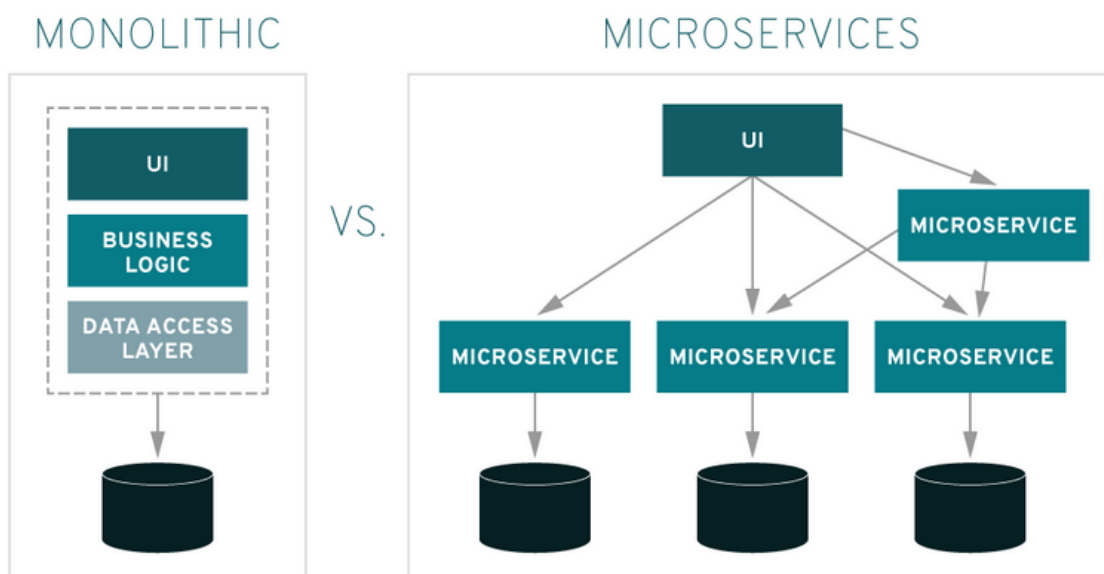


FIGURE 1.9 – Du monolithique aux microservices

Un microservice est :

**élastique** : scalable indépendamment des autres microservices.

**résilient** : si ce service crash, il ne doit pas impacter les autres microservices.

**composable** : il doit offrir une interface favorisant son utilisation

**minimal** : le service ne doit faire qu'une seule chose (on parle de forte cohésion)

**complet** : le service doit fonctionner seul.

On dit souvent qu'un microservice doit être minimal mais complet.

Si la décomposition d'une application en fonctions pour éviter les pièges des architectures monolithiques nous semble être un concept familier, c'est parce que l'architecture de microservices ressemble à l'architecture orientée services (SOA), une conception logicielle déjà bien établie.

### 1.4.1 SOA vs Microservices

Si vous travaillez dans l'informatique ou dans le domaine du Cloud Computing, vous êtes probablement bien conscient du débat entre l'architecture orientée services (SOA) et les microservices. La SOA est apparue pour régler des problèmes d'interopérabilités ; lorsqu'il s'agit par exemple de faire communiquer deux applications fonctionnant sur des technologies différentes (java et .Net, Windows et Linux, etc...). Dans une SOA, toutes les applications doivent se connecter à un ESB (Enterprise Service Bus) afin de communiquer avec les autres applications. L'ESB agit comme la colonne vertébrale de l'intégration, à travers laquelle les services logiciels et les composants applicatifs circulent. L'ESB permet de transformer les données, de les compléter et de les enrichir.[20] La figure suivante illustre cette architecture :

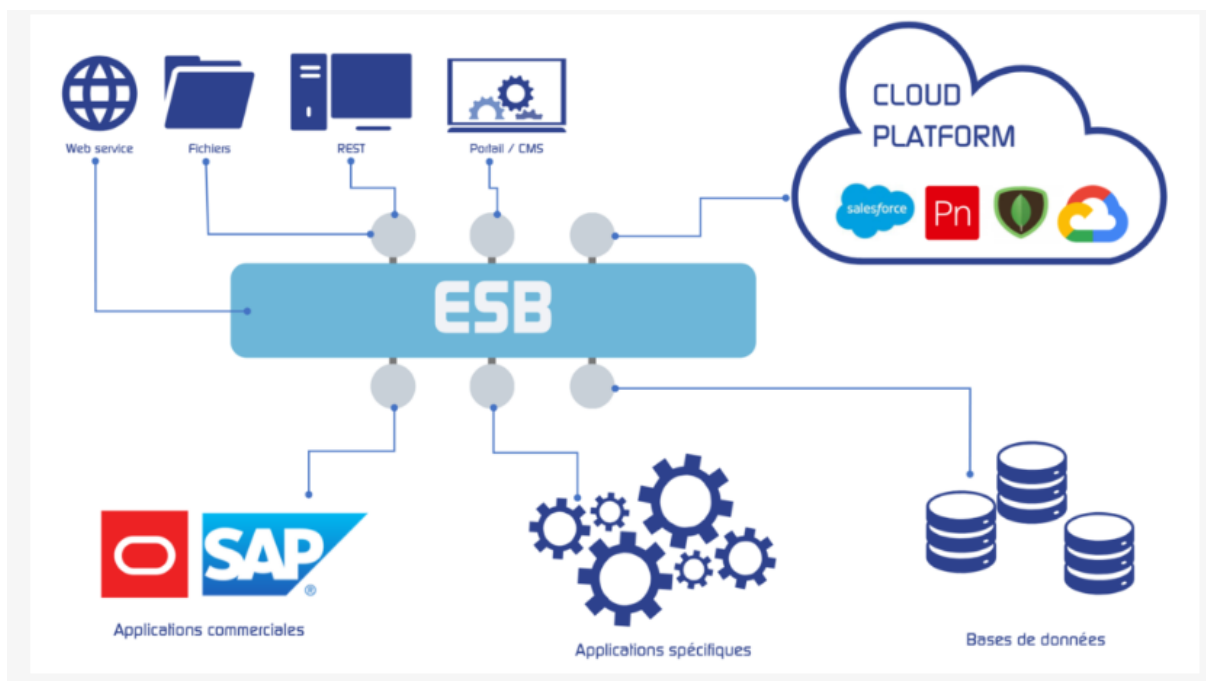


FIGURE 1.10 – Architecture SOA

Une application microservice est un ensemble de microservices faiblement couplés. Chaque microservice peut être modifié et livré de façon indépendante, sans impacter les autres services. On peut dire en cela que l'architecture microservice est une évolution du SOA. Elle propose une granularité plus fine pour répondre à de nouveaux besoins, comme baisser le « time to market » en ne livrant et ne testant que le strict nécessaire. La figure suivante montre la différence entre les applications monolithiques, SOA et microservices.

## Architectures monolithique / SOA / Microservices

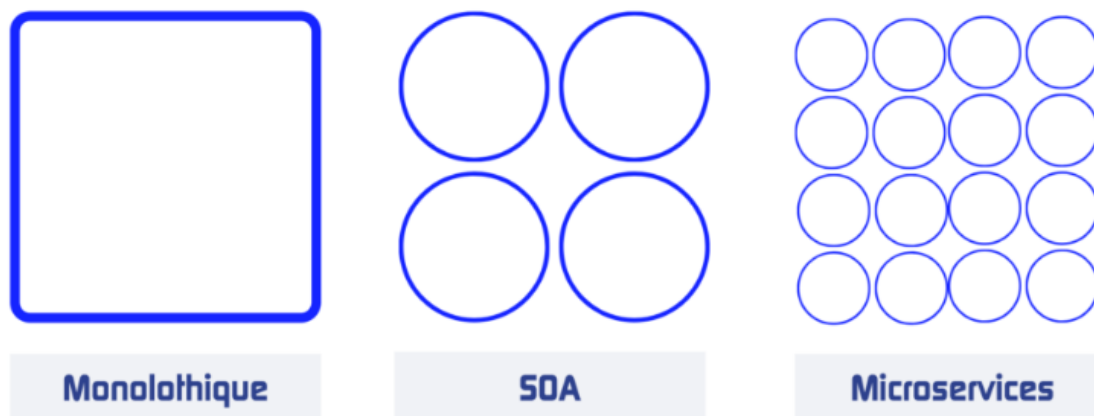


FIGURE 1.11 – Monolithe/SOA/Microservices

### 1.4.2 Les différences clés entre microservices et SOA

La principale distinction entre les deux approches se résume à la portée. Pour faire simple, l'architecture orientée services (SOA) a une portée d'entreprise, tandis que l'architecture des microservices a une portée d'application. Bon nombre des principes fondamentaux de chaque approche deviennent incompatibles lorsque vous négligez cette différence. Si vous acceptez la différence de portée, vous vous rendez peut-être vite compte que les deux peuvent potentiellement se compléter, plutôt que se concurrencer. La figure suivante montre la portée des microservices par rapport à la SOA.

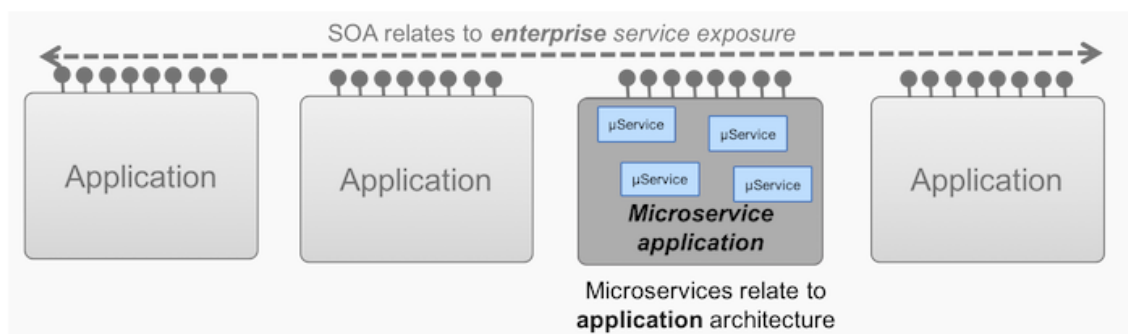


FIGURE 1.12 – La portée des microservices par rapport à la SOA

Les microservices viennent résoudre un bon nombre de problèmes liés à l'architecture SOA :

- **Communication** : Dans une architecture de microservices, chaque service est développé indépendamment, avec son propre protocole de communication. Avec SOA, chaque service doit partager un mécanisme de communication commun appelé bus de service d'entreprise (ESB). L'ESB peut devenir un point de défaillance unique pour toute l'entreprise, et si un seul service ralentit, l'ensemble du système peut être affecté.

- **Interopérabilité** : dans l'intérêt de la simplicité, les microservices utilisent des protocoles de messagerie légers tels que HTTP / REST. Les SOA sont plus ouverts aux protocoles de messagerie hétérogènes.

- **Granularité des services** : les architectures de microservices sont constituées de services hautement spécialisés, chacun étant conçu pour faire très bien une chose. Les services qui composent les SOA, en revanche, peuvent aller des petits services spécialisés aux services à l'échelle de l'entreprise

- **Rapidité** : en tirant parti des avantages du partage d'une architecture commune, les SOA simplifient le développement et le dépannage. Cependant, cela tend également à faire fonctionner les SOA plus lentement que les architectures de microservices, ce qui minimise le partage au profit de la duplication.

## 1.5 Conclusion

L'Architecture orientée services SOA consiste en la décomposition des fonctionnalités du système d'information en de petites applications (services) communiquant à l'aide d'un bus de communication appelé esb (Enterprise Service Bus). Le bus est l'intermédiaire entre tous les services du système d'information. Chaque service doit respecter les principes de la SOA et utiliser des standards. Au fil du temps, les chercheurs ont proposé plusieurs outils de mise en œuvre de la SOA. Le plus connu et le plus complet est « les services web » que nous allons étudier dans le prochain chapitre.

# Chapitre 2

## Les Services Web

### 2.1 Introduction

Les Web services sont nés de l'effort de plusieurs organisations qui ont partagé un intérêt commun en développant et en maintenant "un marché électronique". Celles-ci souhaitent pouvoir communiquer plus simplement et sans avoir à se concerter sur chacune de leur transaction pour pouvoir interpréter leurs différentes données. Elles souhaitent supprimer l'isolement de leur système informatique avec les autres. En effet, les Services web sont apparues avec l'avènement du défi de communication entre des applications s'exécutant sur différentes plateformes, différents réseaux, et différents systèmes d'exploitation. Nous voyons à travers cette petite historique que les services web sont venus concrétiser les concepts théoriques et les objectifs de la SOA qui est d'offrir à l'entreprise un environnement d'intégration facile pour ses applications et les nouvelles technologies. C'est à cet effet que nous étudierons dans ce chapitre les Service Web en passant de sa compréhension globale à ses composants et structuration.

### 2.2 Définition des Services Web

Plusieurs définitions ont été proposées pour les Services Web, parmi lesquelles nous retenons les définitions suivantes :

- Les services Web sont des applications autonomes, modulaires, distribuées et dynamiques qui peuvent être décrites, publiées, localisées ou invoquées sur le réseau pour créer des produits, des processus et des chaînes d'approvisionnement. Ces applications peuvent être locales, distribuées ou basées sur le Web. Les services Web sont construits selon des standards ouverts tels que TCP / IP, HTTP, Java, HTML et XML.
- Les services Web sont des systèmes d'échange d'informations basés sur XML qui utilisent Internet pour une interaction directe application-application. Ces systèmes peuvent inclure des programmes, des objets, des messages ou des documents.
- Un service Web est un logiciel qui se rend disponible sur Internet et utilise un système de messagerie XML normalisé. XML est utilisé pour coder toutes les communications vers un service Web. Par exemple, un client invoque un service Web en envoyant un message XML, puis il attend pour une réponse XML correspondante. Comme toutes les communications sont en XML, les services Web ne sont liés à aucun système d'exploitation ou langage de programmation (Les applications Windows peuvent dialoguer avec des applications Unix.)
- Un service Web est une collection de protocoles ouverts et de normes utilisées pour échanger des données entre des applications ou des systèmes. Les applications logicielles écrites dans différentes langues de programmation et fonctionnant sur diverses plateformes peuvent utiliser des services Web pour échan-



ger des données sur des réseaux informatiques comme Internet de manière similaire à la communication interprocessus sur un seul ordinateur.

## 2.3 Architecture à base de web services

L'architecture de base des services web tourne autour des éléments suivants :

- Un fournisseur de services qui correspond au propriétaire du service. Il est constitué d'une plateforme d'hébergement de services.
- Le client qui correspond au demandeur de services. Il est constitué d'une application qui recherche et invoque un service.
- L'annuaire de services qui correspond à un registre de description offrant des facilités de publications de services à l'intention des fournisseurs et des facilités de recherches de services à l'intention des clients.

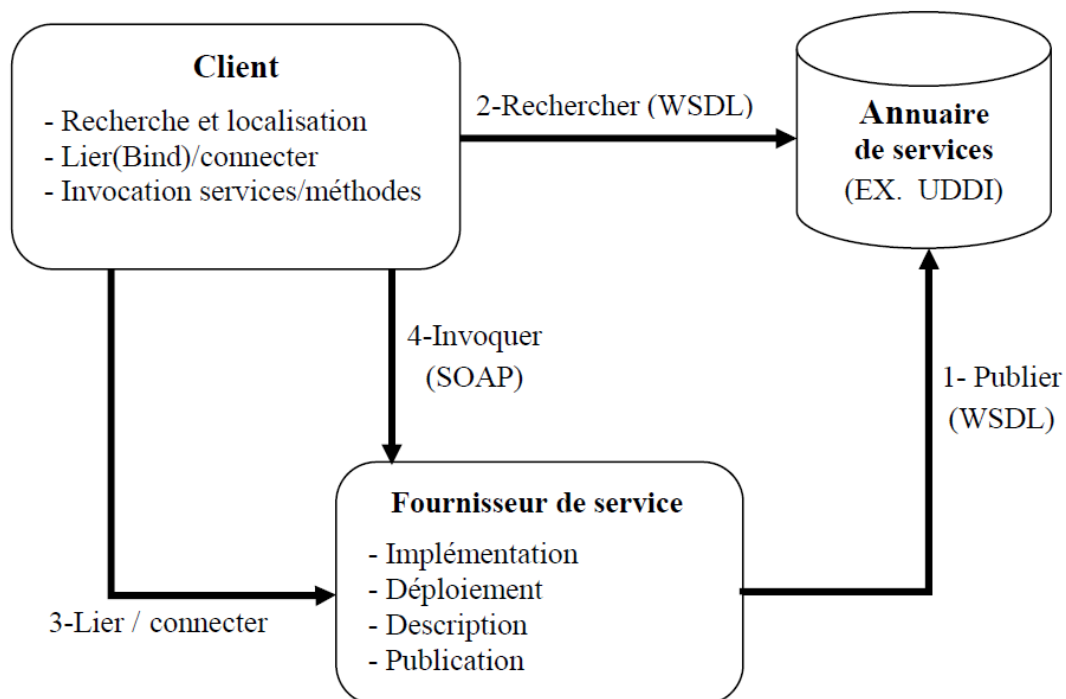


FIGURE 2.1 – Architecture à base des services web

### 2.3.1 WSDL

WSDL est un langage qui a pour objectif la description d'un service web quel que soit le langage d'implémentation et la plateforme de déblayement de ce service. Pour que le WSDL soit compréhensible par les différents langages d'implémentation et soit compatible avec les différentes plateformes utilisées, il est totalement écrit en XML. WSDL décrit un service web en deux étapes fondamentales : une abstraite et une concrète. Et chaque étape est construite pour favoriser la réutilisation et la séparation des préoccupations de conception indépendantes.

Au niveau abstrait, WSDL décrit un service Web en termes des messages qu'il envoie et reçoit ; les messages sont décrits de façon indépendante d'un format spécifique en utilisant un système de types, typiquement un schéma XML. La partie abstraite est composée de définitions de "port type". Chaque "port type" est une collection logique d'opérations. Une "operation" associe un modèle d'échange de message à un ou plusieurs messages. Un message est une unité de communication avec un service web. Il représente les données échangées dans une unique transmission logique. Un modèle d'échange de messages identifie l'ordre et la cardinalité des messages envoyés et/ou reçus. Une interface groupe un ensemble d'opérations. Au niveau concret, un "binding" indique des détails de format de transport pour une ou plusieurs interfaces. Un "endpoint" (i.e. point final) associe une adresse de réseau à un "binding". Finalement, un service groupe un ensemble d'endpoints qui implémente une interface commune.[21] Les figures 2.2 présente la structure d'un fichier WSDL.

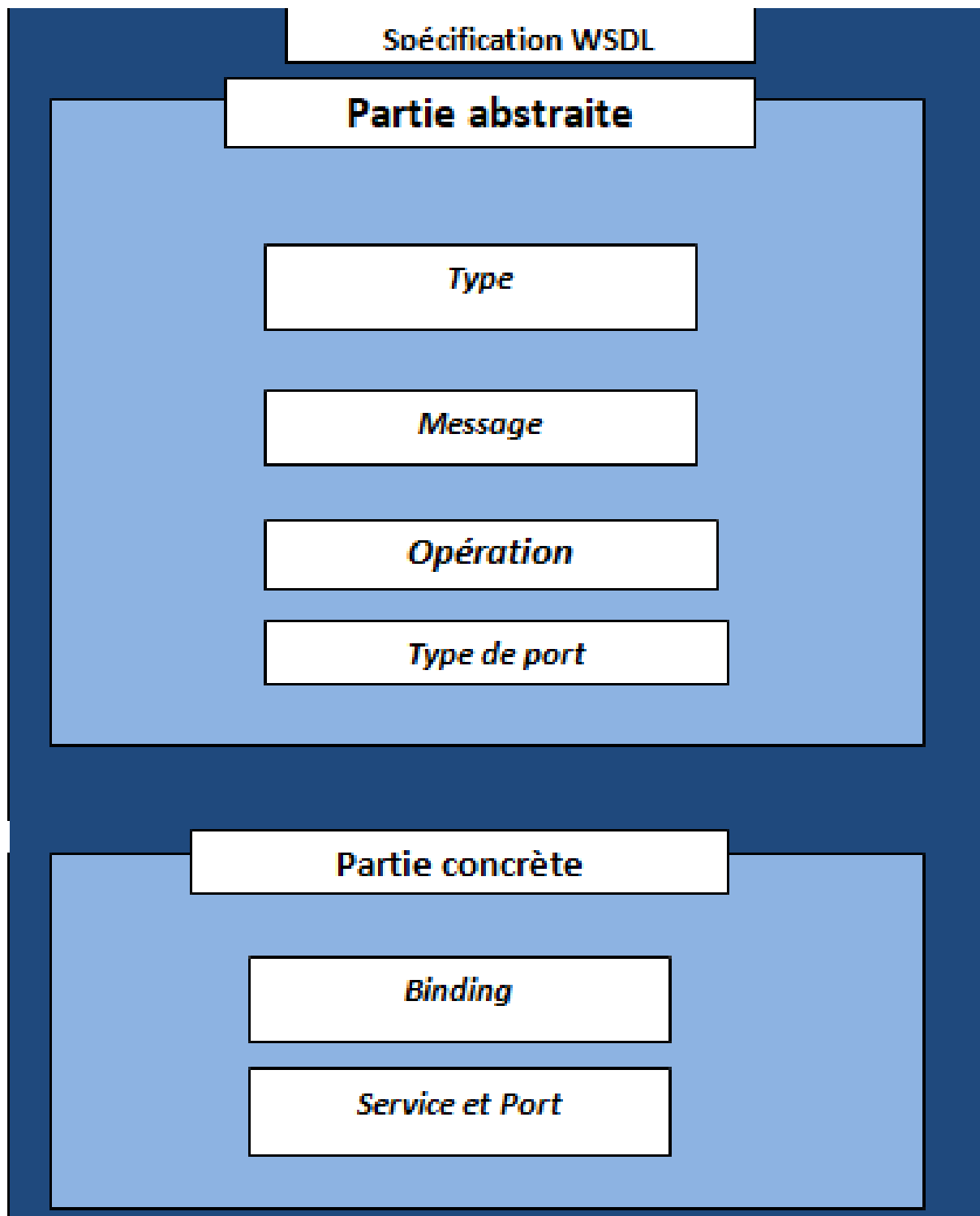


FIGURE 2.2 – Structure d'un fichier WSDL

La figure suivante montre un exemple de fichier WSDL.

```

<definitions Déclaration de « namespace »...>
  <types>
    Définitions des types...
  </types>
  <message>
    Définitions d'un message...
  </message>
  <portType>
    <operation>
      Définitions d'une opération...
    </operation>
  </portType>
  <binding>
    Définitions d'un binding...
  </binding>
  <service>
    Définitions d'un service...
  </service>
</definitions>

```

FIGURE 2.3 – Exemple de document WSDL

La structure d'un document WSDL se compose de plusieurs balises XML que sont :

- L'élément `<definition>` : c'est l'élément racine d'un fichier WSDL, il contient essentiellement le nom de service web et la déclaration du «namespace» utilisés dans le fichier entier. L'élément `<type>` : pour définir tous les types de données utilisées dans le fichier WSDL, le langage WSDL ne spécifie aucun type à utiliser malgré que le schéma XML soit le type de données par default.

- L'élément `<portType>` : cet élément regroupe la définition abstraite de l'ensemble des opérations offertes par ce service. Chaque opération est la définition abstraite d'une méthode dans l'implémentation de service, pour laquelle élément `<operation>` spécifie le nom de l'opération ainsi qu'un message d'entrée et un message de sortie pour l'exécution de l'opération Universal Description Discovery and Integration(UDDI).

- L'élément `<message>` : Chaque opération désigne deux messages, un pour les entrées et l'autre pour les sorties. Dans l'élément `<opération>`, WSDL exige de donner juste les noms de deux messages, le reste des informations comme le types, les noms de paramètres doivent être décrits dans l'élément `<message>`.

- L'élément `<binding>` : contient une définition concrète de types de données et les protocoles pour un ensemble d'opérations et de messages définis dans un porttype.

### 2.3.2 UDDI

Après la description du service web, les clients ne peuvent pas l'utiliser parce qu'ils ignorent l'existence de ce service. C'est pour cela on a besoin d'un moyen qui sert comme une publicité qui annonce la création d'un nouveau produit. Les chercheurs ont défini un annuaire appelé UDDI (Universal Description, Discovery, and Integration) qui sert comme un ensemble de normes pour la publication et la localisation

de services web. Les spécifications du « registre » UDDI ont deux buts principaux en ce qui concerne la découverte d'un service : Le premier est de soutenir les développeurs dans la découverte d'informations sur les services et le deuxième est de permettre la liaison dynamique et en conséquence habilite les clients pour interroger le registre et obtenir des références aux services d'intérêt.

Les informations d'enregistrement UDDI comprennent les cinq types de structure de données suivants : businessEntity, la structure de niveau supérieur, décrivant l'entreprise ou autre entité pour laquelle les informations sont enregistrées. Les autres structures sont reliées par des références de cette structure.

- businessService, le nom et la description du service en cours de publication.
- bindingTemplate, informations sur le service, y compris une adresse du point d'entrée pour accéder au service.
- tModel, une empreinte digitale ou une collection d'informations identifiant de manière unique le service. Cette structure de données prend également en charge les recherches de niveau supérieur.
- publisherAssertion, une structure relationnelle mettant en association deux ou plus de structures businessEntity selon un type spécifique de relation, tel en tant que filiale ou département. [21]

### 2.3.3 Protocoles de Communication

#### 2.3.3.1 SOAP

SOAP est un protocole de transmission de messages basé sur XML. Il définit un ensemble de règles pour structurer des messages principalement pour exécuter des dialogues requête-réponse de type RPC (Remote Protocol Call). SOAP se compose de trois éléments essentiels : l'Enveloppe, le Header, et le Body du message. Supposons un service basé sur le protocole SOAP prenant en paramètres deux entiers et retourne leur somme. La structure du fichier SOAP la présente comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:Add xmlns:ns2="http://ServicesPackage/">
      <entier1>1</entier1>
      <entier2>2</entier2>
    </ns2:Add>
  </S:Body>
</S:Envelope>
```

---

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:AddResponse xmlns:ns2="http://ServicesPackage/">
      <return>3</return>
    </ns2:AddResponse>
  </S:Body>
</S:Envelope>
```

FIGURE 2.4 – Fichier SOAP

Son objectif est de permettre aux applications clientes de se connecter facilement aux services distants et invoquer des méthodes distantes.

Le protocole SOAP présente les avantages suivant par rapport à son homologue REST que nous décrirons dans la section suivante.

- Non dépendance par rapport à la langue, la plate-forme et le transfert (REST nécessite l'utilisation de HTTP)
- Fonctionne bien dans des environnements distribués (REST nécessite une communication directe point à point)
- Standardisé
- Fournit une extensibilité de pré-compilation significative sous la forme de normes WS standards
- Intègre la gestion des erreurs
- Automatisé lorsqu'il est utilisé avec des produits utilisant certains langages de programmation

### 2.3.3.2 L'Architecture REST(Representational State Transfer)

REST résout la complexité du protocole SOAP en se basant sur une simple URL. Dans certaines situations, vous devez fournir des informations supplémentaires de façon particulière, mais la plupart des services Web qui utilisent REST compte exclusivement sur l'obtention de l'information nécessaire en utilisant l'approche de l'URL. REST peut utiliser quatre différents méthodes HTTP (GET, POST, PUT et DELETE) pour effectuer des tâches. Contrairement à SOAP, REST peut utiliser en plus de XML d'autres formats pour fournir la réponse. Vous pouvez trouver des services Web en REST qui ont comme sortie des données au format CSV (Command Separated Value), JavaScript Object Notation (JSON) ou encore Really Simple Syndication (RSS). Vous pouvez donc obtenir la sortie dont vous avez besoin sous une forme qui est facile à analyser avec le langage dont vous avez besoin pour votre application. En majorité, REST est plus facile à utiliser et est plus souple. Il a les avantages suivants par rapport à SOAP :

- Aucun besoin d'outils coûteux pour interagir avec le service Web
- Une courbe d'apprentissage plus petite pour les développeurs
- Efficace (SOAP utilise XML pour tous les messages, REST peut utiliser des formats de message plus petits)
- Rapide (pas de traitement étendu requis)
- Proche d'autres technologies Web dans sa philosophie de conception et est également plus léger

## 2.4 Pile des Services Web (Architecture étendue)

L'architecture étendue est une extension de l'architecture de référence (architecture de base). L'architecture étendue ou Pile de services web est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. On retrouve, au-dessus de la couche de transport, les trois couches formant l'infrastructure de base décrite précédemment. Ces couches s'appuient sur les standards émergents SOAP, WSDL et UDDI.

### 2.4.1 Gestion des services web dans l'entreprise

#### 2.4.1.1 BPEL

Le langage BPEL est un langage d'orchestration des processus métier, écrit en XML et géré par OASIS. Il a été conçu avec l'idée d'orchestrer des intégrations système à système avec des services Web. Chaque entreprise ayant sa manière de définir son flux de processus, l'objectif principal de BPEL est de

normaliser le format de la définition du flux de processus métier afin que les entreprises puissent travailler ensemble de manière transparente à l'aide de services Web. BPEL est basé sur les services Web, au sens où chaque processus métier impliqué est supposé être implémenté en tant que service Web. Les processus écrits dans BPEL peuvent orchestrer des interactions entre des services Web à l'aide de documents XML de manière normalisée. Ces processus peuvent être exécutés sur toute plate-forme ou produit conforme à la spécification BPEL. BPEL prend en charge deux types de processus métier différents : Processus exécutables : Modélise le comportement réel d'un participant dans une interaction commerciale. Ils suivent le paradigme de l'orchestration et peuvent être exécutés par un moteur d'orchestration. Processus abstraits : Utilise des descriptions de processus qui spécifient le comportement d'échange de messages mutuellement visible de chacune des parties impliquées dans le protocole, sans révéler leur comportement interne. BPEL est utilisé pour modéliser le comportement des processus exécutables et abstraits.[22]

La figure 2.5 illustre l'orchestration faite par BPEL :

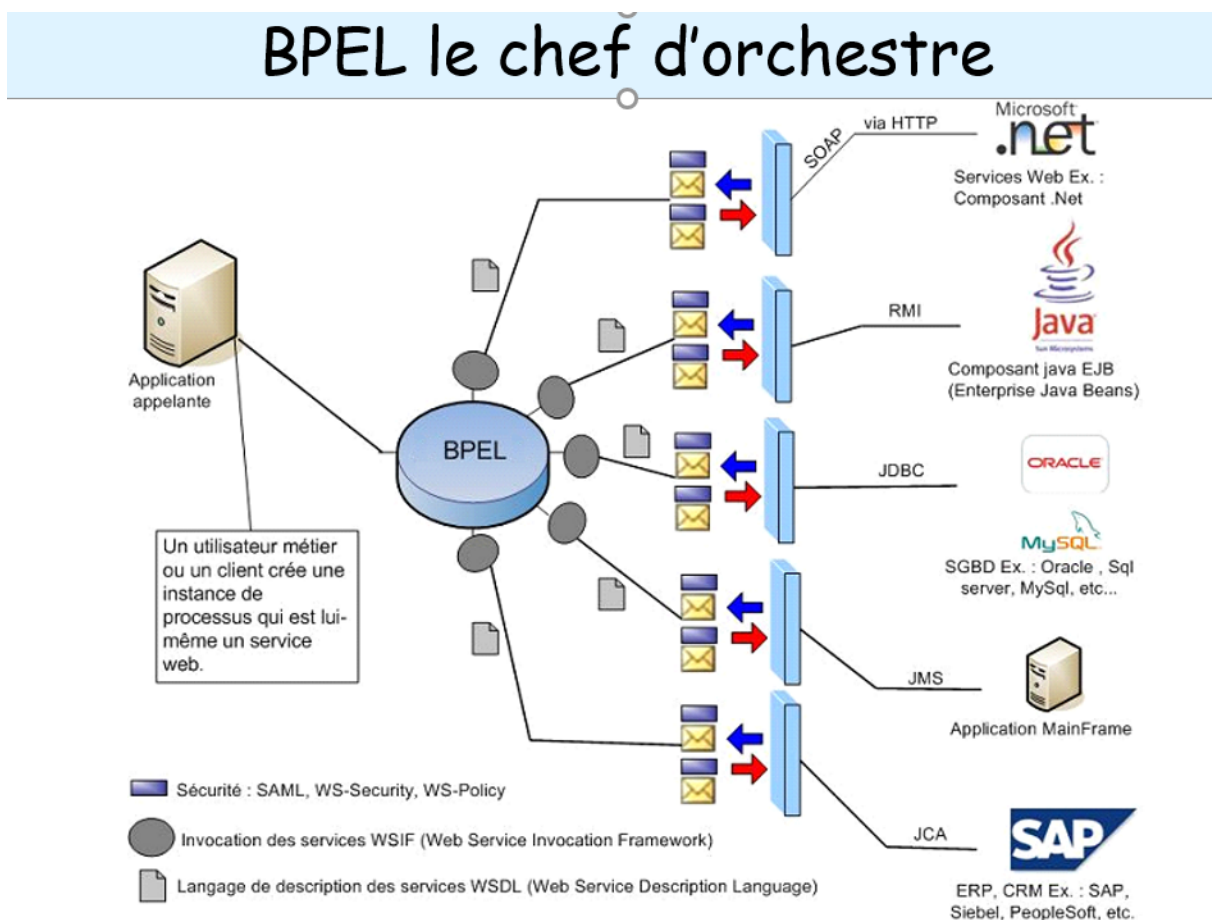


FIGURE 2.5 – Orchestration avec BPEL

## 2.5 Conclusion

Les services web sont utilisés dans plusieurs domaines informatiques, pour permettre à des applications de « dialoguer » à distance via le web. Cette technologie est utilisée comme outil de standardisation de l'accès au service cloud computing. Dans le prochain chapitre, nous évoquerons le Cloud Computing et ses différents composants ainsi que son utilisation dans les systèmes informatiques.



# Chapitre 3

## Le Cloud Computing

### 3.1 Introduction

Avec l'avènement du « big data » (flux massifs de données), les entreprises et les grandes firmes doivent fournir aux clients un accès sécurisé et plus rapide aux données. Le Cloud Computing est venu en tant que parrain de la nouvelle génération d'entrepôts de données et d'accès aux données via le réseau. Cette nouvelle façon de déployer et d'utiliser les données a résolu de nombreux problèmes. Cependant, comme toute nouvelle technologie, elle a ses propres problèmes et besoins. Dans ce chapitre nous irons à la découverte du Cloud Computing, ses différentes formes disponibles, et expliquerons son fonctionnement.

### 3.2 Définition du cloud computing

Le Cloud Computing est un modèle de livraison de services aux clients, en se basant sur des infrastructures virtuelles.[21] Les utilisateurs font appel aux services web pour exécuter des fonctionnalités sur le Cloud. Dans la plupart des cas, les requêtes des utilisateurs ne sont pas réalisables avec un seul service, mais ils sont le résultat de l'exécution de plusieurs services. Pour obtenir le résultat, les services web doivent être composés en un seul service global appelé le service composé. Tout cela est rendu possible grâce à la coopération de différents composants indépendants à savoir :

- La partie virtualisation ou compute responsable de la virtualisation des ressources de mémoire vive et de calculs de l'infrastructure.
- Réseau ou Networking qui va gérer la connectivité entre les machines virtuelles
- Le stockage ou Storage : cette composante gère la persistance et le partage des données du disque.[23]

En d'autres termes, le Cloud Computing, le Cloud, le nuage symbolise la multitude d'unités informatiques de traitement et de stockage disponibles et accessibles depuis l'internet. L'utilisateur n'a pas à savoir précisément où le traitement demandé sera effectué. Il n'a pas non plus à se préoccuper des capacités de traitement si ce n'est pour gérer ses factures. Comme lorsque l'on branche un appareil électrique, on ne se préoccupe pas de savoir de quelle centrale électrique provient l'énergie consommée. Cette quête de souplesse et de disponibilité façonne l'outil informatique et aussi la conception des systèmes d'information.

### 3.3 L'évolution du Cloud Computing

L'idée du Cloud a bien été exposée depuis avant le 21e siècle contrairement à ce que l'on a tendance à penser. Après tout, ce n'est que depuis une dizaine d'années que le Cloud Computing a vraiment

commencé à devenir le géant, omniprésent et tout-puissant que nous connaissons aujourd'hui.

Mais la vérité est que les concepts du Cloud existent depuis de très nombreuses années et peuvent en fait remonter aux années 1950 avec le calcul mainframe. À cette époque, les ordinateurs centraux étaient de grosses machines et très, et même trop chers à acheter et à entretenir pour chaque employé. Et bien sûr, tous les employés n'avaient pas toujours besoin d'en avoir un comme ils le font aujourd'hui. À ce titre, la plupart des organisations n'achèteraient qu'une ou deux machines, puis mettraient en œuvre des horaires de « partage de temps » qui permettraient à plusieurs utilisateurs d'accéder à l'ordinateur central à partir des stations connectées. Ces stations étaient appelées « dumbterminals » (« terminaux stupides » en français) et ne fournissaient aucune puissance de traitement propre. Néanmoins, ce type de puissance de calcul partagée est la prémisse fondamentale et sous-jacente du Cloud Computing, et où tout a commencé.

Au milieu des années 1960, une avancée majeure dans le cloud computing est survenue lorsque l'informaticien américain J.C.R. Licklider a conceptualisé un système interconnecté d'ordinateurs. En 1969, "Lick", comme il est souvent connu, a aidé à développer une version très primitive d'Internet, connue sous le nom de Réseau de l'Agence des Projets de Recherche Avancée (ARPANET :Advanced Research Projects Agency Network ). ARPANET a été le premier réseau à permettre le partage de sources numériques entre des ordinateurs qui ne se trouvaient pas au même emplacement physique. La vision de Lick était également celle d'un monde où tout le monde serait interconnecté par le biais d'ordinateurs et capable d'accéder aux informations de n'importe où. C'est le cas d'Internet tel que nous le connaissons, et une nécessité pour accéder à tous les avantages que le cloud réalise. Au cours des dix dernières années environ, le cloud computing est passée de simple recommandation faite par les fournisseurs de services aux entreprises, à une nécessité pour ces mêmes entreprises. En effet ces organisations ont décidé d'opter pour soit l'utilisation de Cloud par la location chez des tiers, ou de mettre en place leur propre Cloud afin de permettre à des entreprises plus petites et aux utilisateurs de bénéficier des services proposés par le Cloud Computing. La figure ci-dessous résume cette évolution du cloud jusqu'en 2014.[24]

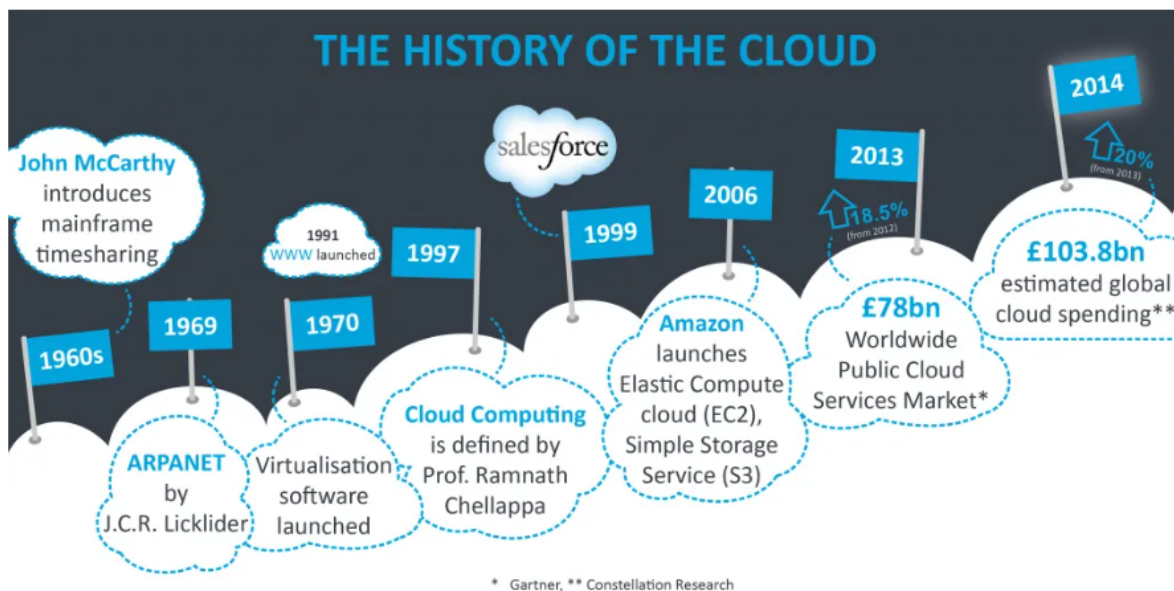


FIGURE 3.1 – Evolution du cloud computing

## 3.4 Les caractéristiques du Cloud

Le Cloud Computing définit essentiellement cinq caractéristiques que sont :

- Accès à la demande par le consommateur : Un consommateur peut utiliser des ressources informatiques, telles que le temps de calcul et la capacité de stockage, automatiquement au besoin sans nécessiter d'interaction humaine avec chaque fournisseur de services.
- Large accès au réseau : Les ressources de Cloud Computing sont accessibles via l'Internet et en utilisant des techniques standardisées. Ce qui donne la possibilité de s'en servir en utilisant un téléphone portable, une tablette ou bien un PC.
- Réservoir de ressources (ressource pooling) : Les ressources informatiques sont partagées pour servir de multiples consommateurs. Elles peuvent être physiques ou virtuelles, et elles sont dynamiquement allouées et libérées selon les demandes des consommateurs.
- Redimensionnement rapide (élasticité) : En fonction de la demande, les ressources et les capacités peuvent être vendues rapidement et même dans certains cas automatiquement, provisionnées et libérées élastiquement. Pour le consommateur, les capacités disponibles pour l'approvisionnement semblent souvent illimitées et peuvent être appropriées à n'importe quelle quantité à tout moment.
- Paiement à l'usage : La facturation est calculée en fonction de la durée et de la quantité de ressources utilisées. Cette caractéristique permet au consommateur et même au fournisseur de contrôler l'utilisation d'un service de cloud (il n'y a pas généralement un coût de mise en service c'est l'utilisateur qui réalise les opérations).

## 3.5 Les services du Cloud Computing ou modèles de livraison

### 3.5.1 IaaS (Infrastructure as a Service)

Les IaaS se présente comme le montre le schéma ci-dessous :

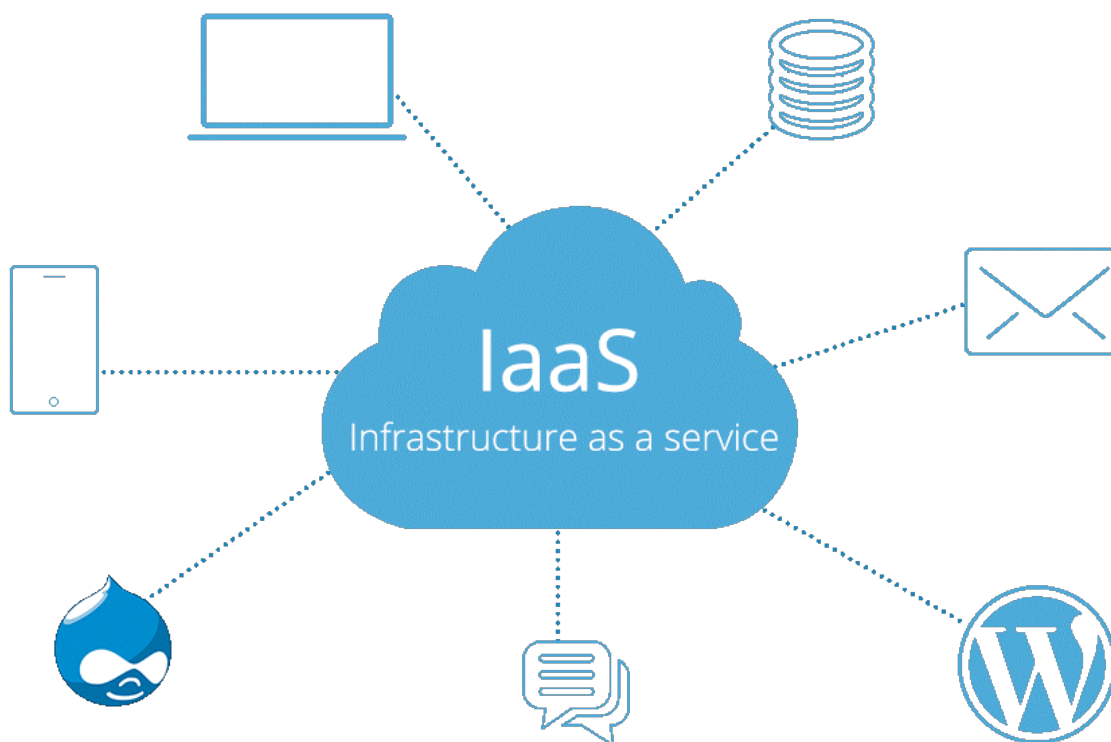


FIGURE 3.2 – Infrastructure en tant que Service

L'infrastructure en tant que Service (IaaS) est l'une des composantes principales du Cloud Computing. Ce service (IaaS) est une forme de Cloud Computing offrant des ressources informatiques au sein d'un environnement virtualisé (le Cloud) par le biais d'internet ou d'une autre connexion comme illustré dans la figure ci-dessus. Au sein d'un modèle IaaS, un fournisseur tiers héberge le hardware, le software, les serveurs, les connexions réseau, la bande passante, l'adresse IP, le stockage et les autres composants de l'infrastructure à la place des utilisateurs. Les fournisseurs d'IaaS se chargent également d'héberger les applications et de gérer les tâches telles que la maintenance de système, le backup (sauvegardes), et la planification de la résilience. Les plateformes IaaS offrent des ressources hautement scalables (ajustables), pouvant être ajustées sur demande en fonction des besoins de l'utilisateur. Les IaaS sont donc optimales pour les charges de travail temporaires, expérimentales ou soumises à des changements inattendus. L'infrastructure permet également à l'utilisateur d'accéder au service depuis n'importe quel endroit du moment qu'il bénéficie d'une connexion internet et que le protocole de sécurité du Cloud le lui permet. Par ailleurs, la sécurité physique des Data Centers hébergeant les IaaS garantit la sécurité des données. De plus, si un serveur subit une erreur, le service dans son ensemble n'est pas affecté, puisqu'il repose sur une multitude de ressources matérielles. Les IaaS permettent de nombreuses utilisations de la part des utilisateurs. Notamment aux entreprises de mettre en place un cloud privé qui leur servira de gestion des espaces de stockage et le lancement des applications en adaptant leur infrastructure en fonction de leurs besoins afin de garantir la sécurité du stockage et le transfert des données sensibles. Elles permettent également d'être utilisée pour de l'Hébergement cloud et pour les Virtuals Data Centers. Pour le premier, le service hébergera des sites sur des serveurs virtuels reposant sur des bassins de ressources en provenance de serveurs physiques. Un site internet hébergé sur le Cloud pourra profiter d'un vaste réseau de serveurs physiques et d'une scalabilité à la demande pour gérer une demande plus élevée qu'escompté. Concernant les VDC, qui sont des réseaux virtualisés de serveurs interconnectés.[25]

### 3.5.2 Pass(Plateforme as a Service)

Une Plateforme en tant que Service (PaaS) est un service Cloud Computing permettant aux entreprises d'externaliser l'hébergement des outils logiciels et matériels de développement d'applications. Le fournisseur Cloud gère l'accès aux outils et services de développement utilisés pour fournir les applications aux utilisateurs en proposant des outils hardware et logiciels en tant que service via internet, permettant à l'utilisateur de développer des applications. Le hardware et le software sont hébergés sur l'infrastructure du fournisseur. Ainsi, les utilisateurs n'ont pas besoin d'installer leur propre hardware et leurs logiciels en interne pour développer ou lancer de nouvelles applications. Parmi les principales fonctionnalités proposées par les fournisseurs de PaaS, on dénombre le système d'exploitation, l'environnement de programmation, le système de gestion de base de données, le logiciel de serveur, le support, le stockage, l'accès réseau, les outils de design et de développement, et l'hébergement. Les fournisseurs se distinguent donc en proposant des fonctionnalités supplémentaires plus spécifiques pour des raisons de concurrence. Les utilisateurs se connectent généralement à la plateforme par le biais d'une interface web. La plupart des plateformes en tant que service sont orientées vers le développement de logiciel. Dans ce contexte, elles offrent plusieurs avantages. Par exemple, une Plateforme en tant que Service permet aux développeurs de changer ou de mettre à jour le système d'exploitation fréquemment. Les PaaS sont également utiles pour permettre aux équipes de développement de collaborer sur divers projets. C'est ainsi que le schéma de la figure suivante présente le processus d'exécution des applications sur un PaaS.

## PaaS = Platform as a Service

### Application créée, déployée et exécutée sur le cloud



FIGURE 3.3 – Plateforme en tant que service

### 3.5.3 SaaS (Software as a Service)

Le Software as a Service (SaaS), ou Logiciel en tant que Service en Français, est le modèle de distribution de logiciel au sein duquel un fournisseur tiers héberge les applications et les rend disponibles pour ses clients par l'intermédiaire d'internet. Modèle de livraison le plus utilisé, les services peuvent être des applications traditionnelles et simples comme le traitement de texte ou d'autres comme la messagerie, les outils de communication ; et pour les entreprises : la gestion de ventes, la gestion de relations client, la gestion de finances, la gestion des ressources humaines, la facturation et la collaboration. On retrouve également le SaaS dans l'Internet des Objets (IOT) et dans l'industrie des jeux vidéo. Le fournisseur Cloud gère l'accès aux services qui sont fournis via internet. Avec le modèle de Logiciel à la Demande, le fournisseur offre aux clients un accès basé sur le réseau à une simple copie d'une application spécifiquement créée par le fournisseur pour la distribution Software as a Service comme l'illustre la figure 19. Le code source de l'application est le même pour tous les clients. Quand de nouvelles fonctionnalités sont déployées, tous les clients peuvent en profiter. En fonction du niveau de service, les données du client peuvent être stockées localement, sur le Cloud, ou les deux à la fois.[26]

La figure 3.4 montre les interactions permises par les SaaS :



FIGURE 3.4 – Logiciel en tant que service

## 3.6 Les modèles de déploiement du Cloud Computing

### 3.6.1 Les Clouds publics

Les applications et les services proposés par un Cloud public sont accessibles par tous. Les services sont gratuits ou payants selon une formule pay-Per-Use (facturation à l'utilisation). Le Cloud public fournit les services à travers l'internet.

### 3.6.2 Les clouds privés

Les applications et les services proposés par un Cloud privé sont destinés à une entreprise ou à une entité spécifique, par exemple une entreprise, le gouvernement. Le Cloud privé peut être configuré via le réseau privé de l'entreprise, plus coûteux ou par un réseau externe.

### 3.6.3 Les clouds hybrides

Un Cloud hybride comprend au moins « deux Clouds ». Ces Clouds restent indépendants mais sont connectés au sein d'une architecture unique. Ses utilisateurs disposent de différents niveaux d'accès.

### 3.6.4 Les clouds communautaires

Un Cloud communautaire est créé exclusivement pour une communauté spécifique. Le Cloud communautaire diffère du Cloud public par ses besoins personnalisés pour répondre aux exigences de la communauté.[23]

## 3.7 Les principaux fournisseurs de services cloud

Les fournisseurs de cloud sont des entreprises qui, comme leur nom l'indique, fournissent des environnements informatiques tels que les clouds publics ou les clouds privés gérés qui extraient, rassemblent et partagent des ressources évolutives sur un réseau. Les « GAFAM » (Google Amazon Facebook Apple Microsoft) et Alibaba occupe une large partie du marché du cloud computing.

### 3.7.1 Les fournisseurs du SaaS

Les premiers fournisseurs de solutions SaaS ont été Yahoo, Google, et Amazon. Des entreprises comme Salesforce (CRM à la demande) et Google (Google Apps) ont adapté ce type d'offre au monde professionnel, pour un certain nombre d'entre elles en s'appuyant sur l'offre Cloud Computing de Amazon EC2. Depuis 2009, les éditeurs « historiques » comme IBM, Microsoft, HP, Sage, SAP, Oracle, Sidetrade, Cegid, Anaplan... , proposent des offres en mode SaaS. À titre d'exemple, Microsoft propose Microsoft Office 365 qui intègre des produits de communication et de collaboration (Exchange, SharePoint, Live Meeting, Communications Server) hébergés dans des datacenters gérés par l'éditeur, et Oracle propose NetSuite qui, depuis 1996, comprend une solution complète ERP, CRM et E-commerce. De nombreuses entreprises développent aujourd'hui des logiciels en mode SaaS, telles Sage (CRM), Oodrive (Sauvegarde de données), Oxatis (création de site e-commerce), Boomerang Web (Gestion des achats), ENFOS, Equity, Netside Planning, Atemis, Caps, Compta... certaines en s'appuyant sur des acteurs du marché tel que Exoscale ou Digital Ocean.

### 3.7.2 Fournisseurs IaaS

De nombreux fournisseurs de services proposent différents types d'Infrastructure as a service, que l'infrastructure virtuelle soit développée à partir de solutions VMware ou Citrix comme Amazon EC2, Microsoft Azure, Scaleway, Oracle4, Outscale et OVH.

### 3.7.3 Fournisseurs PaaS

Parmi les fournisseurs PaaS, on trouve les principaux acteurs de la high-tech :

- Microsoft avec Microsoft Azure ;
- Google avec Google App Engine ;
- Amazon avec Amazon Web Services ;
- Oracle ;
- IBM avec BlueMix ;
- SAP.

## 3.8 Les avantages du Cloud Computing

les avantages du cloud computing peuvent être résumés dans les points suivants :

- **L'adaptabilité des ressources**, au niveau de la capacité de stockage et de la puissance de calcul, selon le besoin de l'utilisateur. Le coût dépend ainsi de l'usage du client ;
- **L'accessibilité, via n'importe quel matériel (téléphone, tablette, PC) et de n'importe quel endroit (connexion internet) ;**
- **Le partage d'informations, avec la mutualisation des données.**
- **La réduction de coût** : c'est un avantage majeur du cloud computing car il permet notamment aux entreprises d'éviter la levée de capital pour le matériel (serveurs, data centers...). Egalement un gain conséquent au niveau de la maintenance puisque la maintenance est gérée par le fournisseur.
- **Haute vitesse** : Le cloud computing vous permet de déployer votre service rapidement en moins de clics. Ce déploiement plus rapide vous permet d'obtenir les ressources requises pour votre système en moins de minutes.
- **Sauvegarder et restaurer les données** : Une fois que les données sont stockées dans un cloud, il est plus facile d'obtenir la sauvegarde et la récupération de celles-ci, ce qui autrement prend beaucoup de temps sur site.
- **Intégration automatique des logiciels** : Dans le cloud, l'intégration logicielle est quelque chose qui se produit automatiquement. Par conséquent, vous n'avez pas besoin de faire des efforts supplémentaires pour personnaliser et intégrer vos applications selon vos préférences.
- **Fiabilité** : La fiabilité est l'un des plus grands avantages du cloud computing. Vous pouvez toujours être instantanément mis à jour sur les changements. [27]

## 3.9 Inconvénients du Cloud Computing

Bien que possédant de multiples avantages, on peut néanmoins dénombrer quelques désavantages du cloud. Nous pouvons citer entre autres :

**Problèmes techniques** : La technologie cloud est toujours sujette à une panne et à d'autres problèmes techniques. Même les meilleures sociétés de fournisseurs de services cloud peuvent faire face à ce type de problème malgré le maintien de normes de maintenance élevées.

**Menace de sécurité dans le cloud** : Un autre inconvénient lors de l'utilisation des services de cloud computing est le risque de sécurité. Avant d'adopter la technologie cloud, vous devez être bien conscient du



fait que vous partagerez toutes les informations sensibles de votre entreprise à un fournisseur de services de cloud computing tiers. Les pirates pourraient accéder à ces informations.

**Temps d'arrêt** : Les temps d'arrêt doivent également être pris en compte lors de l'utilisation du cloud computing. En effet, votre fournisseur de cloud peut faire face à une perte d'alimentation, à une faible connectivité Internet, à la maintenance des services, etc.

**Connectivité Internet** : Une bonne connectivité Internet est indispensable dans le cloud computing. Vous ne pouvez pas accéder au cloud sans connexion Internet. De plus, vous n'avez aucun autre moyen de collecter des données à partir du cloud.

Malgré tous les avantages et les inconvénients, nous ne pouvons pas nier le fait que le cloud computing est la partie la plus rapide de l'informatique en réseau. Il offre un grand avantage aux clients de toutes tailles : utilisateurs simples, développeurs, entreprises et tous types d'organisations. [28]

## 3.10 Technologies et Composants du cloud Computing

### 3.10.1 Les serveurs

Un serveur cloud est une infrastructure physique ou virtuelle puissante qui exécute des applications et stocke des informations. Les serveurs cloud sont créés en divisant un serveur physique (baremetal) en plusieurs serveurs virtuels à l'aide d'un logiciel de virtualisation. Les entreprises utilisent un modèle IaaS (infrastructure sous forme de service) pour traiter les charges de travail et stocker des informations. Elles peuvent accéder aux fonctions du serveur virtuel à distance via une interface en ligne. Le tableau suivant fait la différence entre un serveur cloud et un serveur dédié.

Caractéristiques	Server Cloud	Serveur Dédié
Type d'infrastructure	Serveur virtuel avec ressources dédiées	Serveur physique avec ressources dédiées
Accès root	Oui	Oui
Configuration	Evolutif à tout moment	Non modifiable étant donné qu'un matériel y est dédié
Stockage partagé	Oui	Oui
Réseaux privés	Oui	Oui
Réseaux privés virtuels (VPN)	Oui	Oui
Gestion des utilisateurs	Oui	Oui
Gestion des rôles	Oui	Oui

TABLE 3.1 – Différence entre serveur dédié physique et serveur cloud

### 3.10.2 La virtualisation

La virtualisation est l'ensemble des techniques matérielles ou logicielles employées pour faire tourner plusieurs systèmes d'exploitation sur une même machine physique. Le système d'exploitation ainsi virtualisé est appelé système invité (guest system) et le système d'exploitation servant d'accueil s'appelle le système hôte (host system). La machine physique utilisée est appelée machine hôte tandis que celle émulée au système invité est la machine virtuelle. Une machine virtuelle peut donc être vue comme un logiciel au même titre qu'un navigateur Web qui fournit au système d'exploitation invité un environnement similaire à celui d'une machine physique. Le cloud computing se base sur la technologie de virtualisation de serveurs qui consiste à créer plusieurs serveurs virtuels totalement indépendants à partir d'un seul serveur physique. Cette technologie permet d'optimiser les ressources de chaque serveur en créant des environnements distincts ou ressources dédiées. On utilise pour cela une solution logicielle de virtualisation appelée Hyperviseur.[21]

### 3.10.3 Les solutions de type IaaS

#### 3.10.3.1 Les Solutions propriétaires

Il existe une liste pléthorique d'entreprises fournissant du Service IaaS. Nous choisirons d'étudier quelques-unes occupant la grande partie du marché du Cloud qui est notamment dominé par les grandes firmes tel que Amazon, Google et Microsoft

**Microsoft Azure** : a été créé par Microsoft pour la création, le test, déploiement et la gestion des applications et des services via leurs centres de données. Il est sorti en 2010. Microsoft a créé son Cloud public Azure par-dessus Windows Server et Hyper-V. Cette proximité logicielle facilite la migration des VM entre les datacenters locaux et Azure. Il est possible également de connecter ce dernier à votre réseau d'entreprise via un VPN point à point. Sur le marché de l'IaaS, l'approche de Microsoft est complète, surtout après le lancement d'Azure Stack, sa plateforme de déploiement de cloud hybride. Microsoft a défini 17 régions pour Azure, situées un peu partout aux États-Unis, en Europe, en Asie, en Amérique du Sud et en Australie.

**Amazon Web Services (AWS)** : AWS est sans conteste la référence dès qu'on évoque le cloud et en particulier l'IaaS (Infrastructure as a service). Sur ce segment qui a connu la plus forte croissance en 2016, Amazon détient près de 45 Premièrement, son interface graphique : AWS Management Portal for vCenter permet de bénéficier d'un chemin de migration des VM (et notamment les VM VMware) très clair. Les instances des VM peuvent être hébergées dans des régions spécifiques. En Europe, AWS dispose de 13 datacenters. Deuxième atout : sa fonction Enhanced Networking assure une connectivité réseau qui est plus performante (faible latence notamment) que celle proposée en standard.

**Google Compute Engine** offre toutes les fonctionnalités de base de la connectivité réseau d'un Cloud, directement ou via un VPN. Mais il ne prend pas en charge les migrations des VM dans le Cloud Google Compute Engine. Il est nécessaire de passer par des fournisseurs tiers. Google permet de choisir les régions d'hébergement des VM. Le géant a des datacenters en Europe (Belgique, Royaume-Uni, Allemagne, Pays-Bas et Finlande).

#### 3.10.3.2 3.10.3.2 Les Solutions Open-Source

##### 3.10.3.3 3.10.3.2.1 Openstack

Openstack est une plateforme qui permet de créer et de gérer des Cloud privés ou publics à partir de pools de ressources virtuelles. Les outils composants de la plateforme assurent les différentes fonctionnalités telles que les calculs, la mise en réseau, le stockage, la gestion des identités et la gestion des images d'un Cloud. Openstack utilise des API pour booster les limites de l'abstraction des ressources

virtuelles en les répartissant dans des pools individuels, qui pilotent des outils de Cloud Computing avec lesquels les administrateurs et les utilisateurs interagissent. Openstack s'appuie sur deux logiciels pour créer l'environnement Cloud :

- Un logiciel de virtualisation qui crée une couche de ressources virtuelles à partir du matériel
- Un système d'exploitation de base qui exécute les commandes transmises par les scripts Openstack

Openstack ne virtualise pas les ressources mais utilise ces dernières pour construire des Clouds. Il n'exécute pas les requêtes mais les transmet au système d'exploitation de base. Plusieurs composants contribuent à la mise en place de la solution Openstack :

- **NOVA** : La partie compute d'OpenStack, Nova, permet de communiquer avec l'hyperviseur pour lancer et gérer les machines virtuelles. Plusieurs types d'hyperviseurs sont actuellement disponibles tels que Xen, VMWare (ESX) et Hyper-V. Il est divisé en plusieurs composants : Un composant de traitement des requêtes des clients : nova-api Le composant de gestion des machines virtuelles ou daemon : nova-compute Un composant d'ordonnancement du déploiement des machines virtuelles : nova-scheduler Un intermédiaire entre nova-compute et la base de données : nova-conductor.

- **NEUTRON** : il s'occupe de la gestion du réseau du Cloud Openstack. Neutron possède également une interface d'API. Très utile pour les administrateurs, elle permet d'avoir une vue abstraite des ressources utilisées, telles que le segment réseau où la machine virtuelle se trouve, configuration de l'adressage réseau, des informations sur les ports de connexion etc.

- **Swift** : On peut l'utiliser indépendamment des autres modules d'Openstack afin de créer un cluster de stockage objet. Swift fournit aux entreprises qui cherchent à stocker des données efficacement une solution hautement disponible à moindre coût. Il est divisé en deux parties principales :

- **Storage nodes** : Les requêtes venant des clients REST passent par le proxy qui se coordonne ensuite avec les nœuds de stockage appropriés pour traiter la demande. Lors de la sauvegarde d'un objet, Swift en fait trois copies et tente de les stocker sur trois serveurs distincts. Swift retourne ensuite un code réponse au client qui est alors assuré que ses données sont sauvegardées et répliquées

- **CINDER-Block Storage** : Le stockage par bloc est à la base de la sauvegarde des machines virtuelles et de leurs données. Par défaut, les VM utilisent un stockage éphémère (les données stockées sont détruites en même temps que la VM). Cinder, de son côté, permet de créer des volumes persistants de stockage par bloc (block Storage).

- **Glance** : s'occupe de gérer les images utilisées par les machines virtuelles pour démarrer. Nova communique avec Glance en utilisant son API lorsqu'une instance est déployée

En plus de ces composants de bases, Openstack dispose d'autres composants tels que Keystone qui prend en charge l'authentification, Heat qui orchestre les instances de machines virtuelles, et Horizon qui fournit le tableau de bord.

### 3.10.3.4 Opennebula

Opennebula est une plateforme open-source de gestion de Cloud. Il se caractérise par sa simplicité, la richesse de ses fonctionnalités et sa flexibilité. La première version publique d'Opennebula est sortie en 2008. Il est construit autour de trois briques :

- La brique **SDC** (Software DefinedCompute) : Cette brique gère la virtualisation des serveurs et permet ainsi d'installer plusieurs services différents sur un seul serveur.

- La brique **SDS** (Software Defined Storage) pour la virtualisation du stockage

- La brique **SDN** (Software Defined Network) : gère la virtualisation du réseau. Ainsi, on peut regrouper les équipements réseaux virtuels tels que les commutateurs, les routeurs, les pare-feu dans une console commune. Ces concepts permettent l'automatisation de la gestion liée à la virtualisation, le réseau et le stockage. Opennebula dispose de plusieurs composants qui lui permet offrir toutes les fonctionnalités

recherchées :

- **Opennebula**, le cœur de la solution Opennebula et ainsi le daemon qui dans le cas d'Openstack.
- **Opennebula-sunstone** : offre une interface graphique pour l'administration et un portail self-service pour les clients finaux.
- **Opennebula-gate** : il permet la collection des paramètres et les problèmes sur les machines virtuelles.
- **Opennebula-flow** : chargé de la gestion de services installés sur différentes machines virtuelles et la mise en place de l'auto-scaling.

### 3.10.3.5 Eucalyptus

Eucalyptus est un outil open source issue d'un projet de recherche de l'université de Californie. Cette solution est la plus connue, car elle est intégrée dans les distributions Ubuntu Server et Debian. Eucalyptus est écrit en C, Java et Python ; et permet de créer des Clouds IaaS de type privé ou hybride. Il supporte les machines virtuelles Linux ainsi que les hyperviseurs Xen et KVM. Son avantage majeur est le fait qu'il est compatible avec Amazon EC2. Il possède également une version entreprise (payante) de la société Eucalyptus Systems qui apporte des fonctionnalités supplémentaires comme le support de VMware.

Le tableau suivante présente la comparaison entre les solutions open-source (OpenNebula, OpenStack ; Eucalyptus) :

Caractéristiques	<b>Openstack</b>	<b>Opennebula</b>	<b>Eucalyptus</b>
Source code	Open Source	Open Source	Open Source
Produit par	Rackspace, NASA, Dell, Citrix, Cisco, Canonical et plus que 50 autres organisations	Communauté Opennebula sous licence Apache	Apparu au début dans l'université Santa Barbara de l'université de Californie Eucalyptus System company
But	Créer et ouvrir des fonctionnalités de Cloud Computing en utilisant un logiciel open-source fonctionnant sur du matériel standard	Un Cloud privé pur	Une réponse open source pour le Cloud commerciale
Domaine d'utilisation	Les sociétés, les fournisseurs de services, les chercheurs et les centres de données qui cherchent à déployer à grande échelle Cloud privés ou publiques	Les chercheurs dans le domaine du Cloud Computing et de la virtualisation	Les entreprises
Systèmes d'exploitation supportés	-Linux -Windows -Exige x86 processeur	Linux (Ubuntu, RedHat Enterprise Linux, Fedora et SUSE Linux Enterprise Server)	Linux (Ubuntu, Fedora, CentOS, OpenSUSE et Debian)
Langage de Programmation	Python	Java, C++, Ruby	Java, C, Python
Hyperviseur	Xen, KVM	Xen, KVM, VMWare	XEN, KVM
Installation	Facile, installation automatisée et documentée.	Manuelle, installation facile sur les distributions supportées (dont Debian et Ubuntu).	Problématique, dépend de l'environnement réseau et matériel, difficulté en environnement hétérogène
Orientation	Cloud public et privé	Cloud privé	Cloud public et privé

TABLE 3.2 – Comparatif entre les solutions libres de types IaaS

[28]

### 3.10.4 Les solutions de type PaaS

Une solution PaaS fournit les services de l'IaaS (dématérialisation du matériel) ainsi que les applications middlewares : système d'exploitation, le serveur web, la base de données, etc. L'avantage de cette solution est son côté évolutif, ainsi qu'une pré-configuration de l'environnement technique. On y retrouve les mêmes grandes firmes qu'au niveau de l'IaaS. Nous avons :

- **Google AppEngine** : Avec un SDK conçu par Google, l'entreprise offre à ses utilisateurs un développement en local pour ensuite déployer l'application sur internet. L'idée est de permettre aux utilisateurs d'employer l'infrastructure de Google pour héberger leurs applications avec la possibilité de définir le groupe d'utilisateurs de cette dernière. Ces applications bénéficient de la haute disponibilité des infrastructures de Google. Google a aussi mis à disposition Google Colab qui facilite aux développeurs de Machine et Deep Learning l'apprentissage et l'exécution de leur code avec de la puissance de calcul.

- **Microsoft Azure** a également une plateforme cloud pour faciliter le développement et le stockage des applications sur internet.

Il existe également des solutions PaaS de type open-source. On peut citer Openshift lancé en 2011 par RedHat ; Dokku lancé en 2013 par Docker ; Cloud foundry, Flynn...

### 3.10.5 Les solutions de type SaaS

La solution de type SaaS étant la plus répandue sur le marché offre de large gamme de possibilités car son marché prolifère rapidement. On retrouve encore une fois les GAFAM (Google Amazon Facebook Apple Microsoft) qui rafle la grande part du marché du SaaS ; même si Facebook est un peu en retrait dans le domaine du cloud. Des entreprises comme Salesforce (CRM à la demande) et Google (Google Apps) ont adapté ce type d'offre au monde professionnel en B2B, pour un certain nombre d'entre elles en s'appuyant sur l'offre Cloud Computing de Amazon EC2. Depuis 2009, les éditeurs « historiques » comme IBM, Microsoft, HP, Sage, SAP, Oracle, Sidetrade, Cegid, Anaplan... proposent des offres en mode SaaS. À titre d'exemple, Microsoft propose Microsoft Office 365 qui intègre des produits de communication et de collaboration (Exchange, SharePoint, Live Meeting, Communications Server) hébergés dans des datacenters gérés par l'éditeur, et Oracle propose NetSuite qui, depuis 1996, comprend une solution complète ERP, CRM et E-commerce. De nombreuses entreprises développent aujourd'hui des logiciels en mode SaaS, telles Sage (CRM), Oodrive (Sauvegarde de données), Oxatis (création de site e-commerce), Boomerang Web (Gestion des achats), ENFOS, Equity, Netside Planning, Atemis, Caps, Compta... certaines en s'appuyant sur des acteurs du marché tel que Exoscale ou Digital Ocean.

## 3.11 Conclusion

Elon Musk, fondateur de Paypal et de Tesla affirmait dans une interview réalisée en mai 2020 que : « ...dans 10 ans, le cloud détiendrait plus d'informations sur notre vie que nous sur la nôtre ». Cela promet un avenir fleurissant pour le cloud Computing. Et étant flexible, adaptable et surtout moins coûteux pour les entreprises, ces dernières migrent de plus en plus vers son utilisation. Evidemment cela soulève des problèmes de sécurité et des questions essentielles sur la vie privée en générale. Mais le cloud se présente comme une solution plus que jamais incontournable pour les particuliers et les entreprises pour une gestion des données et des infrastructures.

## Chapitre 4

# Exploitation du Cloud et des services web pour la mise en place d'une architecture orientée services

### 4.1 Introduction

Le cloud computing s'est imposé au fil des années comme une norme pour les entreprises et propose des services performants et scalables qui permettent aux applications de fonctionner adéquatement. L'utilisation du cloud computing pour la mise en place d'une SOA permettra au système d'information de répondre plus efficacement aux demandes. Nous présenterons donc opennebula et ses composants qui une solution IaaS de cloud privé ; ensuite la technologie de développement des microservices et terminer avec les différents tests.

### 4.2 Présentation des outils de mise en place d'un système à base de SOA

#### 4.2.1 Mise en place d'une Plateforme de gestion du cloud

##### 4.2.1.1 Opennebula

OpenNebula à la différence des solutions de Cloud Computing classiques, fournit une boîte à outils complète permettant de gérer de façon centralisée une infrastructure virtuelle hétérogène. L'outil est compatible avec les hyperviseurs classiques : Vmware, Xen, KVM. OpenNebula opère comme un ordonnanceur des couches de stockage, réseau, supervision et de sécurité. C'est une solution adaptée à la conversion d'une infrastructure virtuelle en Plateforme IaaS. Cette fonction d'orchestration centralisée, d'environnements hybrides est le cœur de l'outil. Ce projet initié en 2005 a livré sa première version en 2008 et reste depuis actif. De nombreux releases ont permis d'obtenir aujourd'hui des évolutions fonctionnelles importantes sur le support des nœuds de stockage, la haute disponibilité des environnements et l'ergonomie des interfaces d'administration. Opennebula est distribuée sous licence Apache 2.0.

#### 4.2.1.2 Description de l'architecture Opennebula

#### 4.2.1.3 Les composants Opennebula

#### 4.2.1.4 Le front-end

Le Front-end est la partie centrale d'une installation OpenNebula. Il s'agit de la machine sur laquelle le logiciel serveur est installé et sur laquelle vous vous connectez pour gérer votre cloud. Il peut s'agir d'un nœud physique ou d'une instance virtuelle. La recommandation minimale des ressources pour le front-end est la suivante :

Ressources	Configuration minimale recommandée
mémoire	8 GB
CPU	1(4 cores)
Espace disque	100 GB
réseau	2 NICS

#### 4.2.1.5 L'hôte (Node)

Un hôte est un serveur qui a la capacité d'exécuter des machines virtuelles et qui est connecté au serveur frontal d'OpenNebula. OpenNebula peut travailler avec des hôtes avec une configuration hétérogène, c'est-à-dire que vous pouvez connecter des hôtes au même OpenNebula avec différents hyperviseurs ou distributions Linux.

#### 4.2.1.6 Présentation des étapes nécessaire pour créer une machine virtuelle

Dans la réalisation de notre Cloud, nous avons installé le front-end et un seul nœud sur la même machine physique. Si l'installation du front-end se passe bien, nous pouvons nous connecter en tant que oneadmin à travers l'interface sunstone ou la ligne de commande CLI. L'interface de connexion à sunstone est représentée dans la figure suivante :



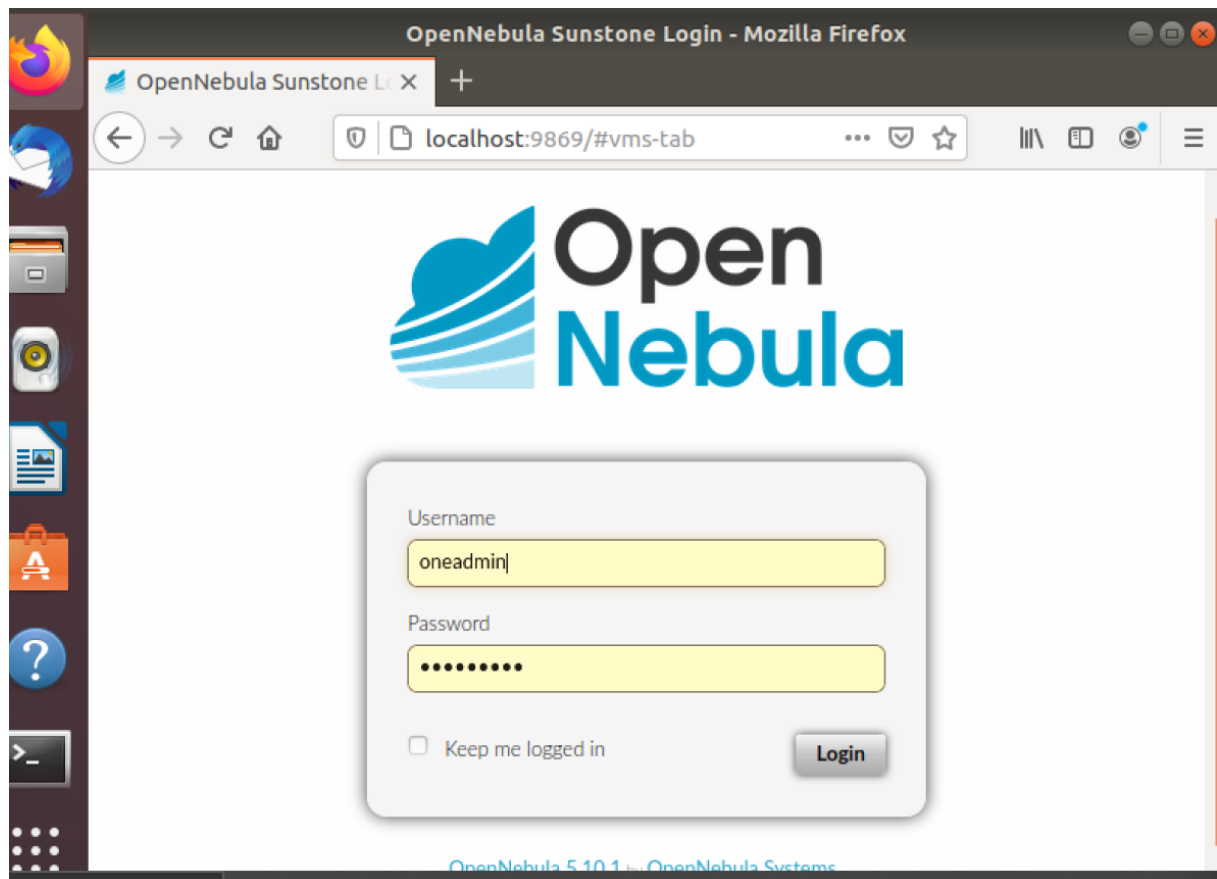


FIGURE 4.1 – Page d'accueil Sunstone

Après la saisie de mot de passe valide nous devons voir l'interface ci-dessous :

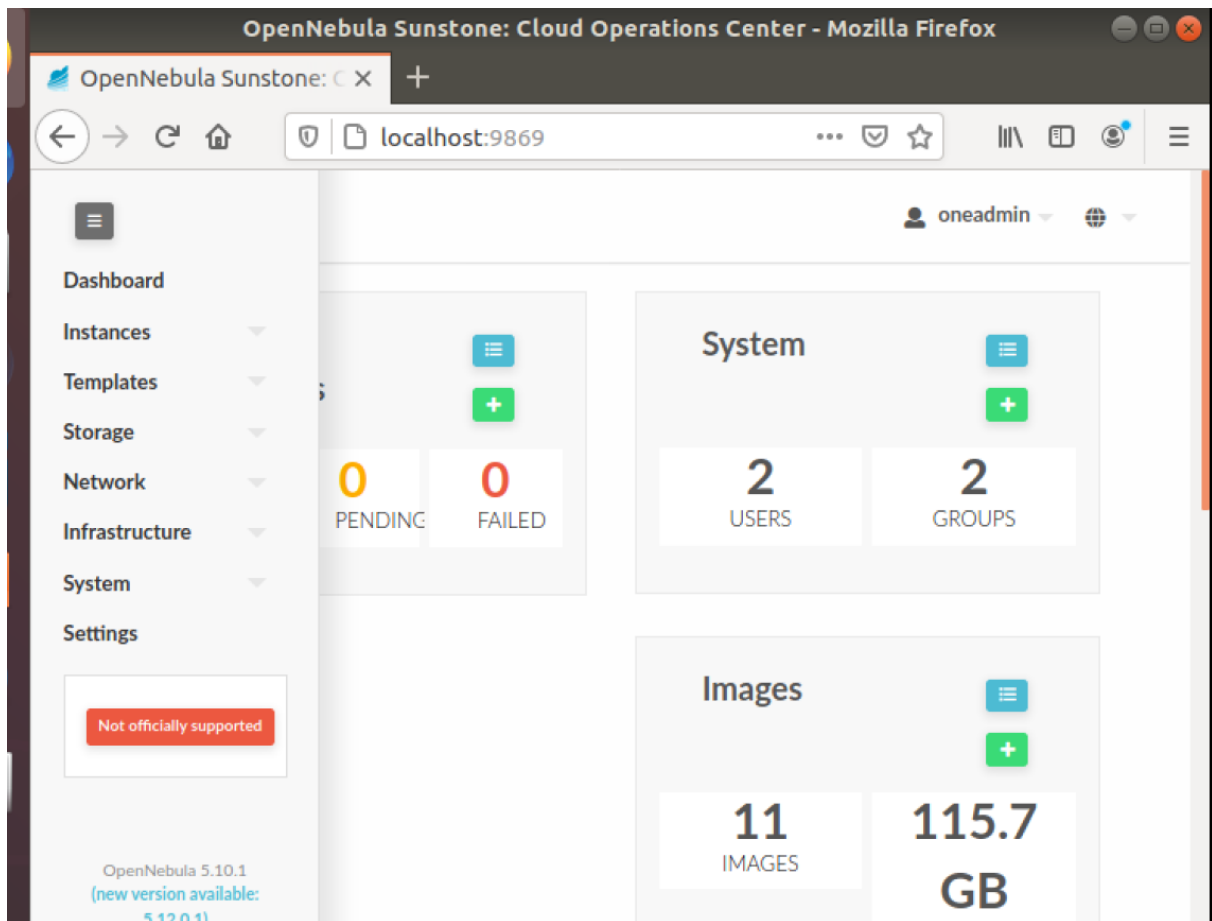


FIGURE 4.2 – Interface de gestion de OpenNebula

Nous allons maintenant décrire les différentes ressources essentielles à la création de machine virtuelle :

- **Storage** : il contient les sous ressources comme :

- **Datastore** qui permet de créer des espaces de stockages pour les machines virtuelles et les autres fichiers utiles.

- **Images** : Cette ressource permet la création des images qui seront utilisées pour installer le système d'exploitation de la machine virtuelle.

- **MarketPlaces** : On y trouve des systèmes d'exploitation téléchargeables prêts à l'emploi.

- **Infrastructure** : Le plus important pour nous ici est hosts qui permet d'ajouter des hôtes (nœud openNebula) au Cloud. Clusters est un ensemble constitué d'hôtes et une zone est un ensemble de clusters.

- **Networks** : permet de créer les réseaux dans lesquels les machines virtuelles seront exécutées.

- **Templates et instances** : Le template permet de créer un modèle de machine virtuelle qui sera instancié au besoin à l'aide de la ressource instance.

Dans la suite nous allons d'abord montrer comment ajouter un hôte, ensuite comment créer une image et en fin l'instanciation de la machine virtuelle elle-même.

#### 4.2.1.7 Ajout d'un nœud OpenNebula

Dans cette partie nous allons enregistrer le nœud que nous avons installé dans openNebula front-end pour permettre à openNebula d'y créer des machines virtuelles. Ici nous allons le faire à travers l'interface Sunstone . Ouvrir Sunstone comme décrit précédemment. Dans le menu à gauche, aller sur Infrastructure -> Hosts puis cliquer sur le bouton + comme illustré sur la figure suivante :

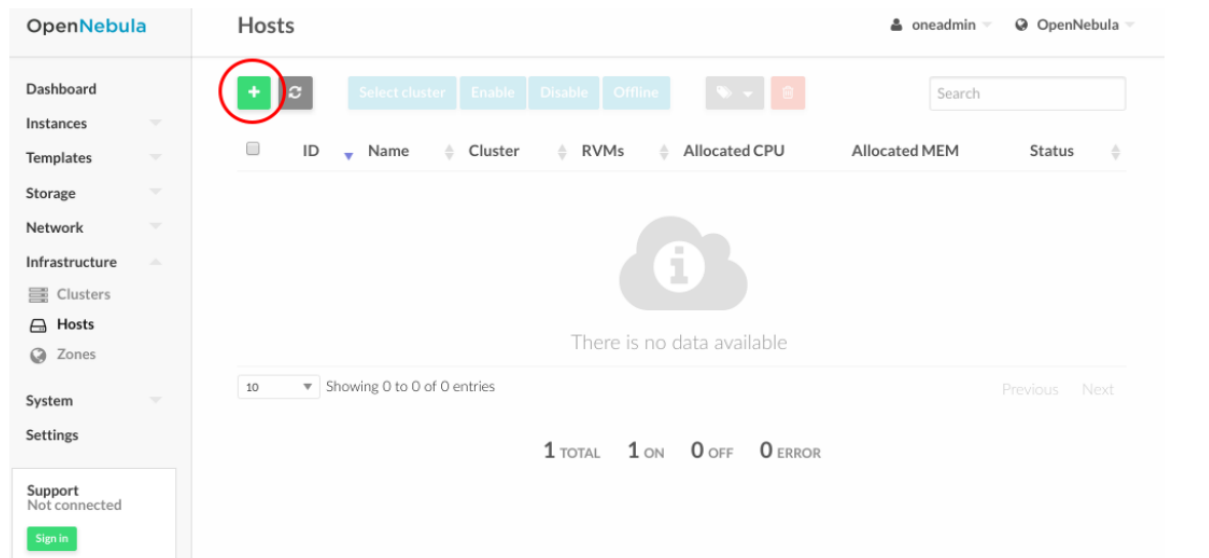


FIGURE 4.3 – Création d'un host étape 1

Ensuite Remplir le nom ou l'adresse Ip de la machine (hôte) et cliquer sur le bouton create. En fin

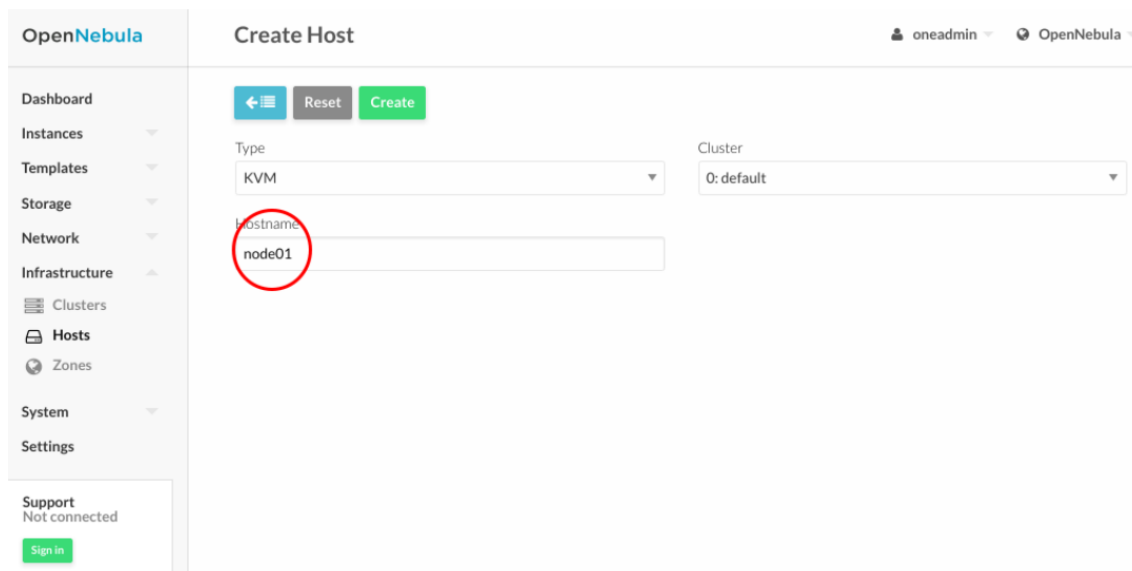


FIGURE 4.4 – Création d'un host étape 2

retourner à la liste des hosts pour vérifier que l'hôte est dans l'état ON.

#### 4.2.1.8 Création d'une image

La création de l'image se fait comme suit : Pour la création d'image, nous devons d'abord penser au système d'exploitation que nous allons utiliser. Nous pouvons soit préparer notre système d'exploitation et l'importer lors de la création de l'image, soit l'importer directement du MarketPlaces. La figure suivante montre le début de la création de l'image par Storage -> Images puis un clic sur le boutons +. La liste que nous voyons est une liste d'images que nous avons déjà créées.

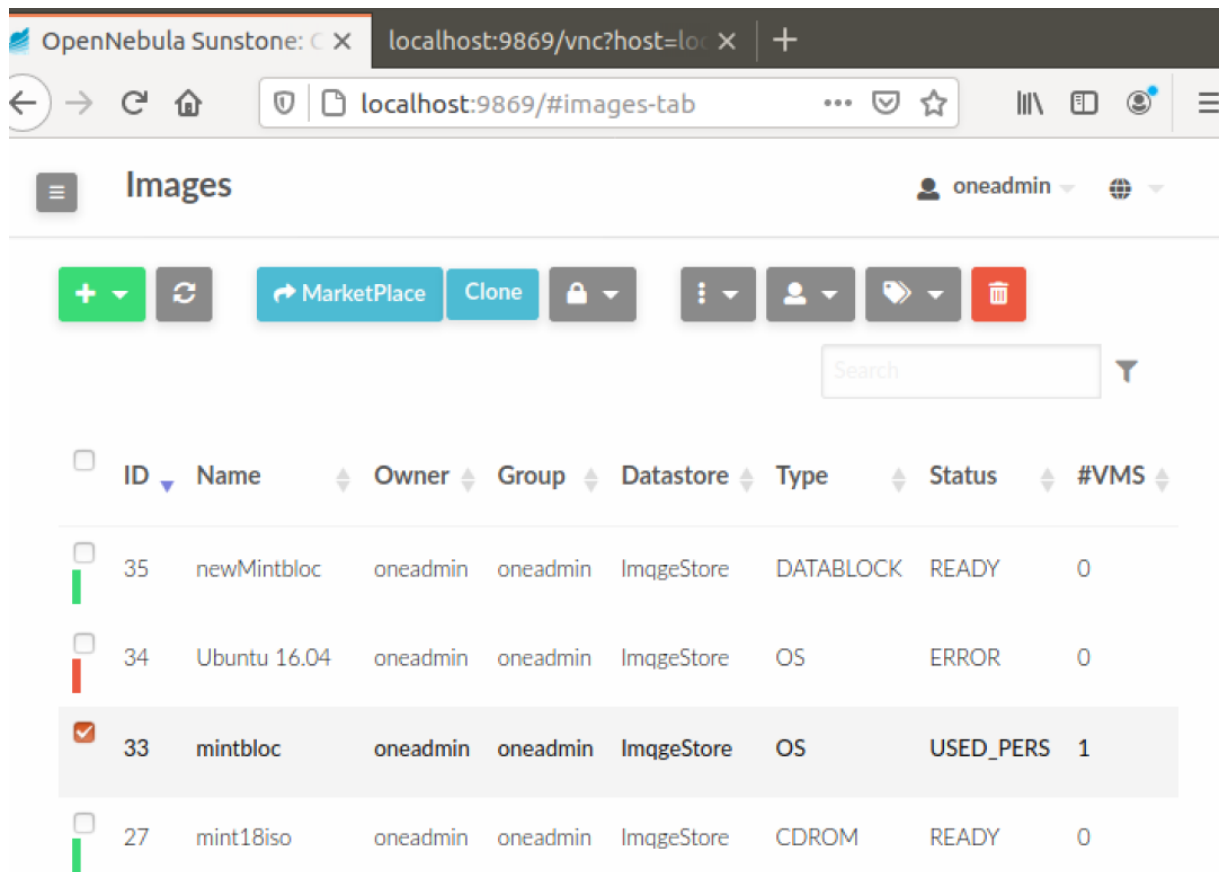
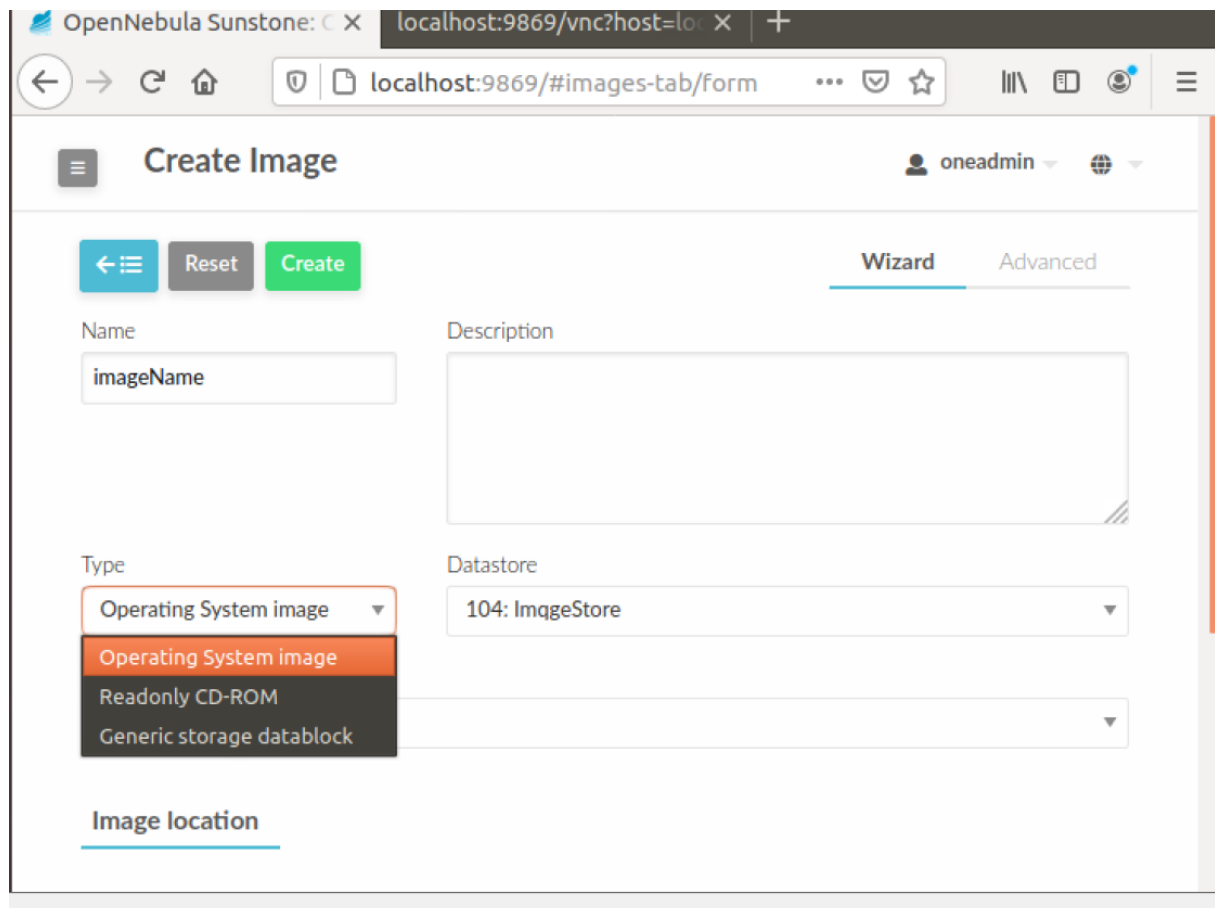


FIGURE 4.5 – Création d'image étape 1

Remplir les différents champs et sélectionner l'image système à installer si elle est sur notre ordinateur localement ou télécharger la à partir du MarketPlaces.



The screenshot shows a web browser window with the URL `localhost:9869/vnc?host=loc` and a sub-page `localhost:9869/#images-tab/form`. The page title is "Create Image" and the user is logged in as "oneadmin". The interface is in "Wizard" mode, with "Advanced" also visible. At the top left, there are navigation buttons: a back arrow, a "Reset" button, and a "Create" button. The form contains the following fields:

- Name:** A text input field containing "imageName".
- Description:** A large text area for entering a description.
- Type:** A dropdown menu with the following options: "Operating System image" (selected), "Operating System image", "Readonly CD-ROM", and "Generic storage datablock".
- Datastore:** A dropdown menu with the selected option "104: ImageStore".

At the bottom of the form, there is a section titled "Image location" which is currently empty.

FIGURE 4.6 – Création d'image étape 2

#### 4.2.1.9 Présentation des résultats de création de la machine virtuelle

Nous n'allons pas montrer ici la création du Template et du réseau pour la création de la machine virtuelle. Dès que le Template est créé avec succès nous pouvons maintenant créer des instances. La figure suivante montre le début de la création de la machine virtuelle en passant par instance -> VMs puis un clic sur le bouton +.

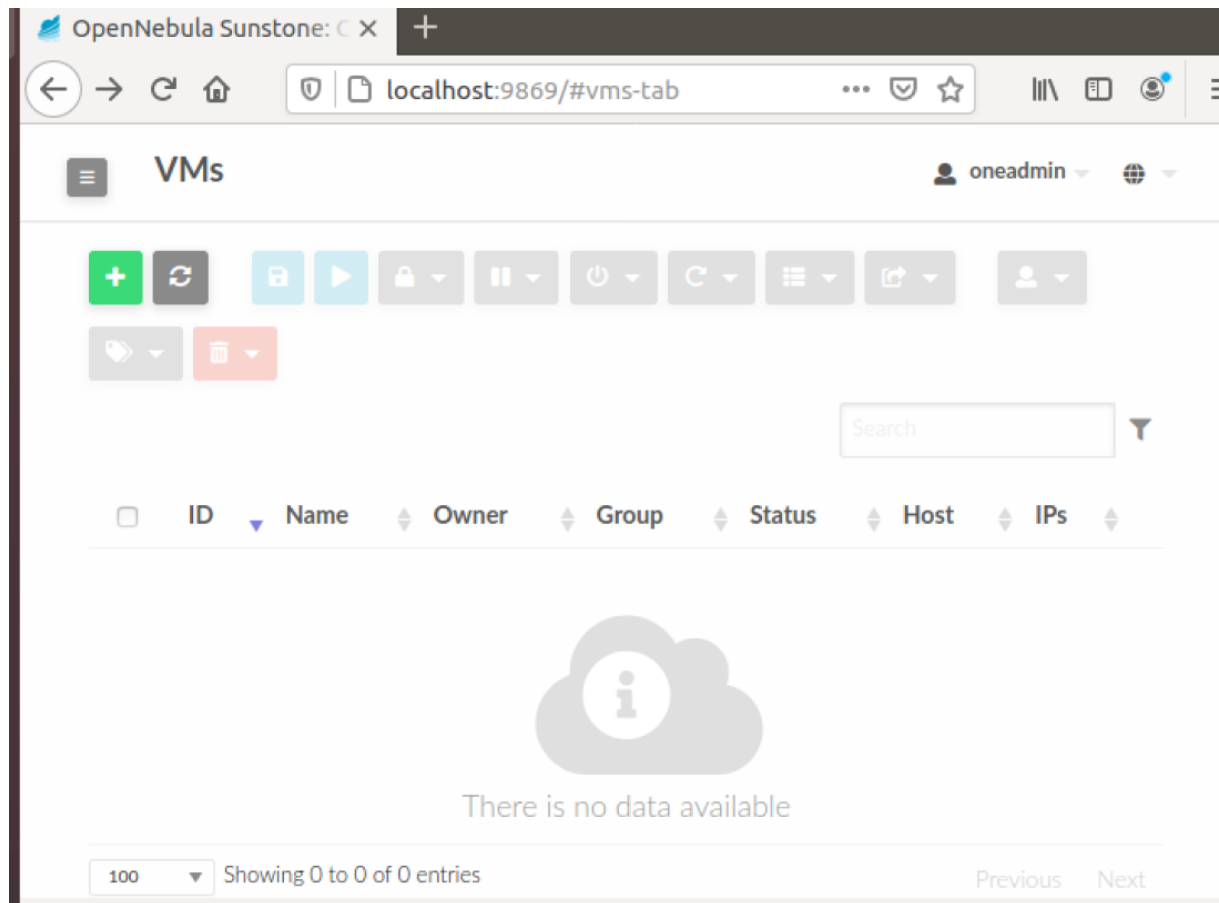


FIGURE 4.7 – Création VM étape 1

Ensuite nous verrons le Template que nous avons préparé dans la liste. Juste un clic sur ce Template permet de créer notre machine virtuelle dont l'installation de son système suit la procédure habituelle d'une installation de système d'exploitation sur une machine physique. La figure ci-dessous montre les Templates que nous avons utilisés pour les tests de notre projet.

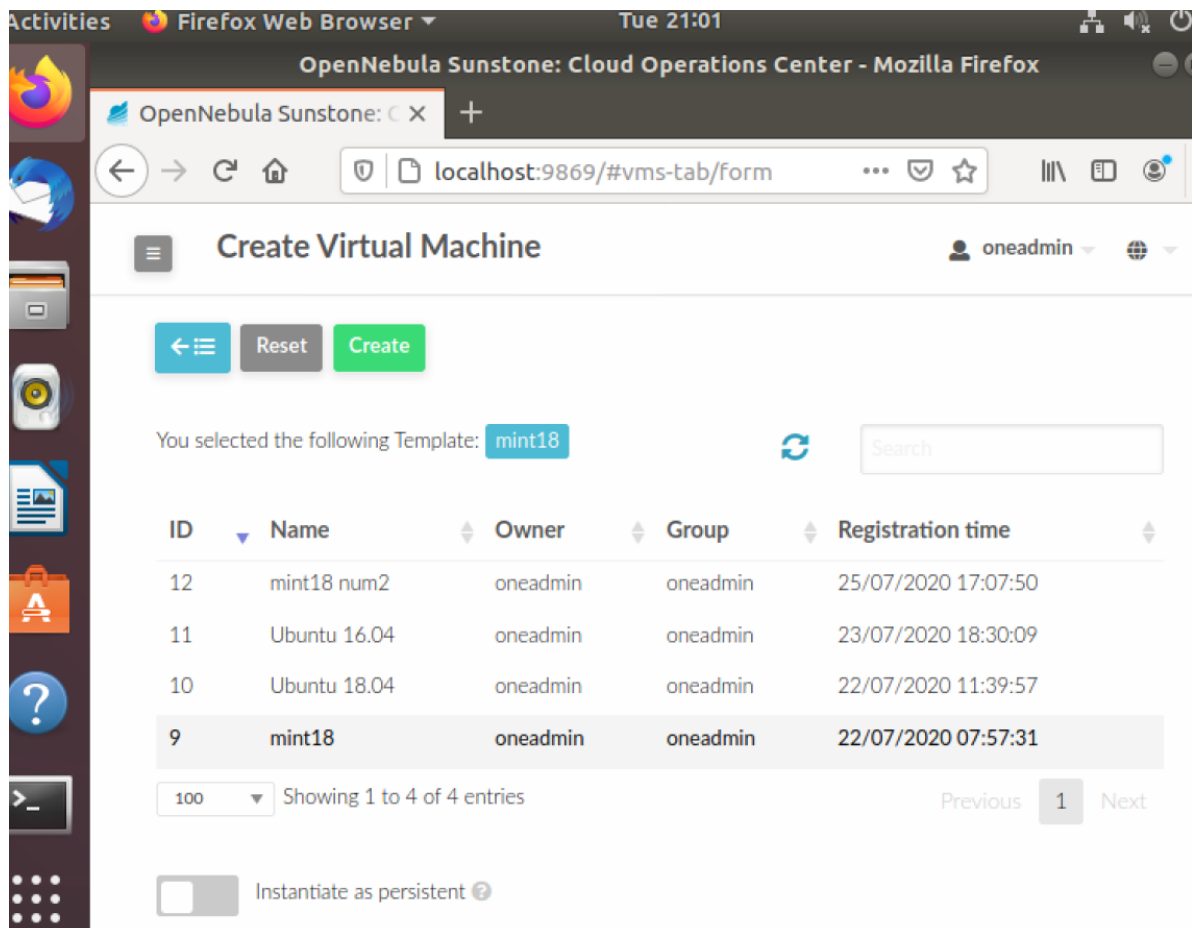


FIGURE 4.8 – Création VM étape 2

Une fois l'installation terminée nous pouvons démarrer notre machine au besoin. Nous avons installé une machine virtuelle avec le système d'exploitation Linux Mint. Les deux figures suivantes montrent la machine virtuelle démarrée avec le système d'exploitation Linux Mint.

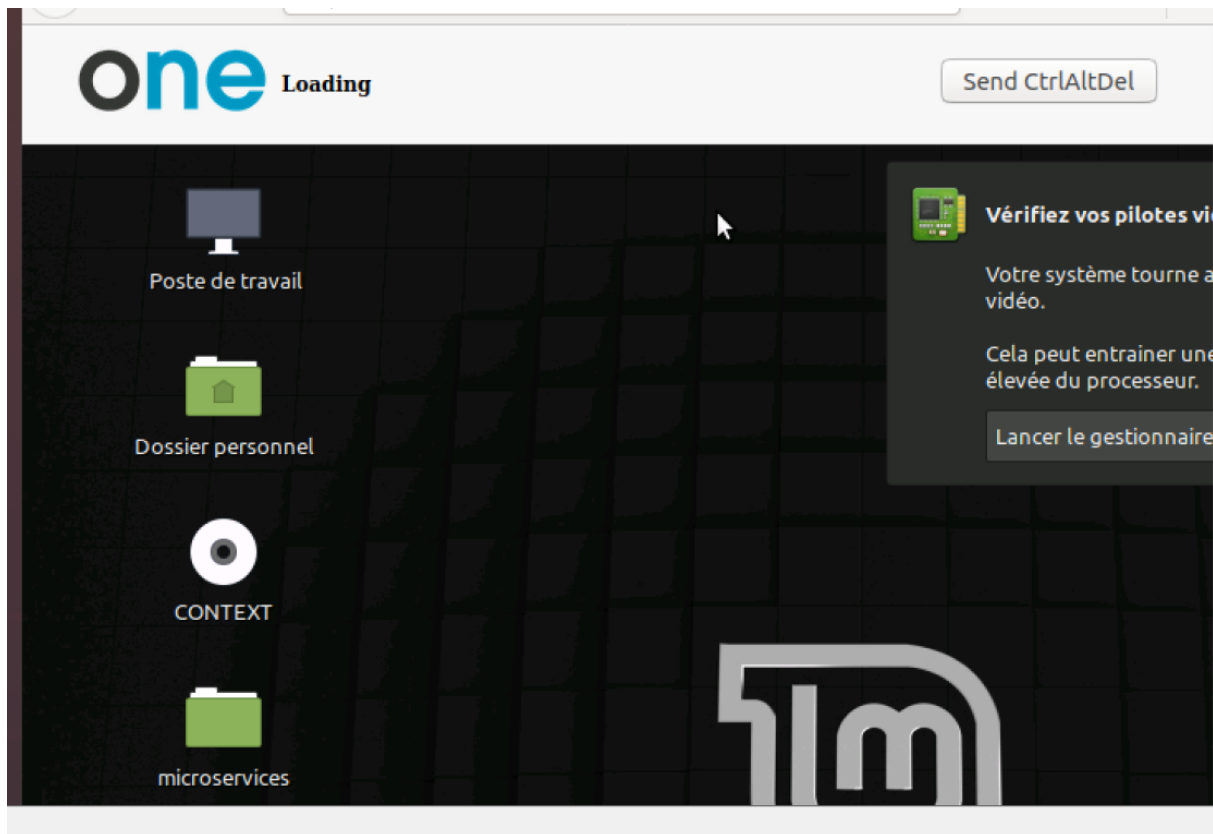


FIGURE 4.9 – Bureau de la machine virtuelle

La seconde image de la machine virtuelle.



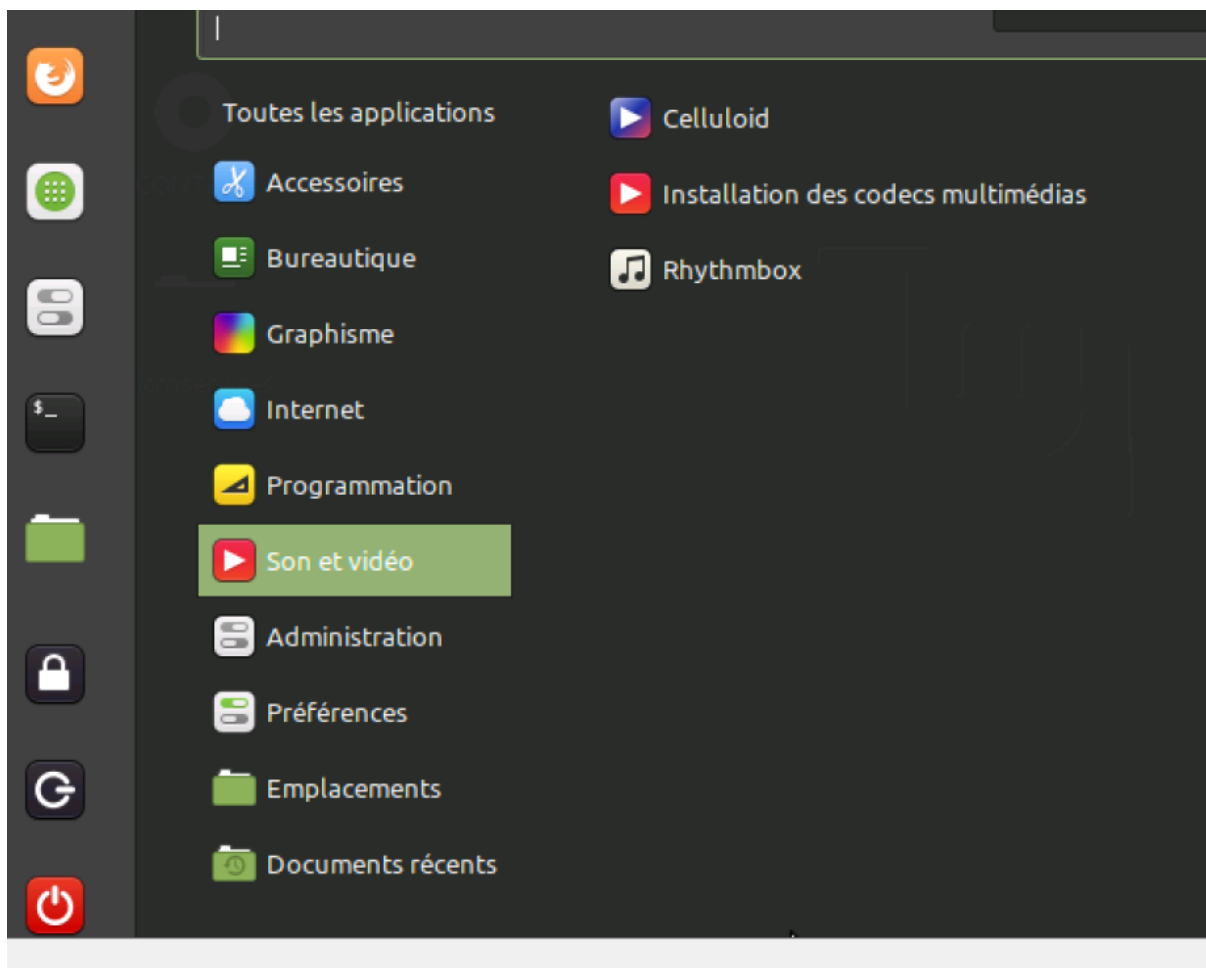


FIGURE 4.10 – Machine virtuelle Linux Mint

L'objectif de la mise en place de ce Cloud est d'y déployer des services à base de la SOA. Ainsi, après la mise en place des services, nous présenterons les résultats de déploiement de ces services dans la Cloud.

## 4.3 Mise en place des microservices

### 4.3.1 Description des microservices à mettre en œuvre

Nous allons mettre en place une application de gestion d'une bibliothèque universitaire à l'aide des microservices. Les différents microservices à développer sont les suivants :

- Un microservice de gestion des étudiants. Ce microservice gère les tâches suivantes :
  - Insertion d'un étudiant dans la base de données
  - Suppression d'un étudiant de la base de données
  - Mise à jour des informations d'un étudiant dans la base de données - Récupérer la liste de tous les étudiants de la base de données
  - Vérifier l'existence d'un étudiant dans la base de données
- Un microservice de gestion des livres. Ce microservice gère les tâches suivantes :
  - Insertion d'un livre dans la base de données
  - Suppression d'un livre de la base de données
  - Mise à jour des informations d'un livre dans la base de données
  - Récupérer la liste de tous les livres de la base de données

- Vérifier l'existence d'un livre dans la base de données
- Un service d'envoi d'email aux étudiants pour confirmation des inscriptions ou toutes autres informations.
- Service de génération des identifiants. Ce service gère les tâches suivantes :
  - Génération des identifiants des étudiants
  - Génération des identifiants des livres
- des services démarrés. Ainsi tout service qui démarre publie sa référence dans le service d'enregistrement.
- Un service de configuration : Ce service gère les fichiers de configuration des services.
- Un service Gateway/proxy : Ce service est le proxy, l'intermédiaire entre le client des services et les services.

### 4.3.2 Présentation des différents diagrammes de l'application

Dans cette partie nous allons présenter les différents diagrammes de modélisation de l'application. La base de données doit contenir les entités « student » pour étudiant, « book » pour livre et « borrow » pour les emprunts comme le montre la figure suivante :

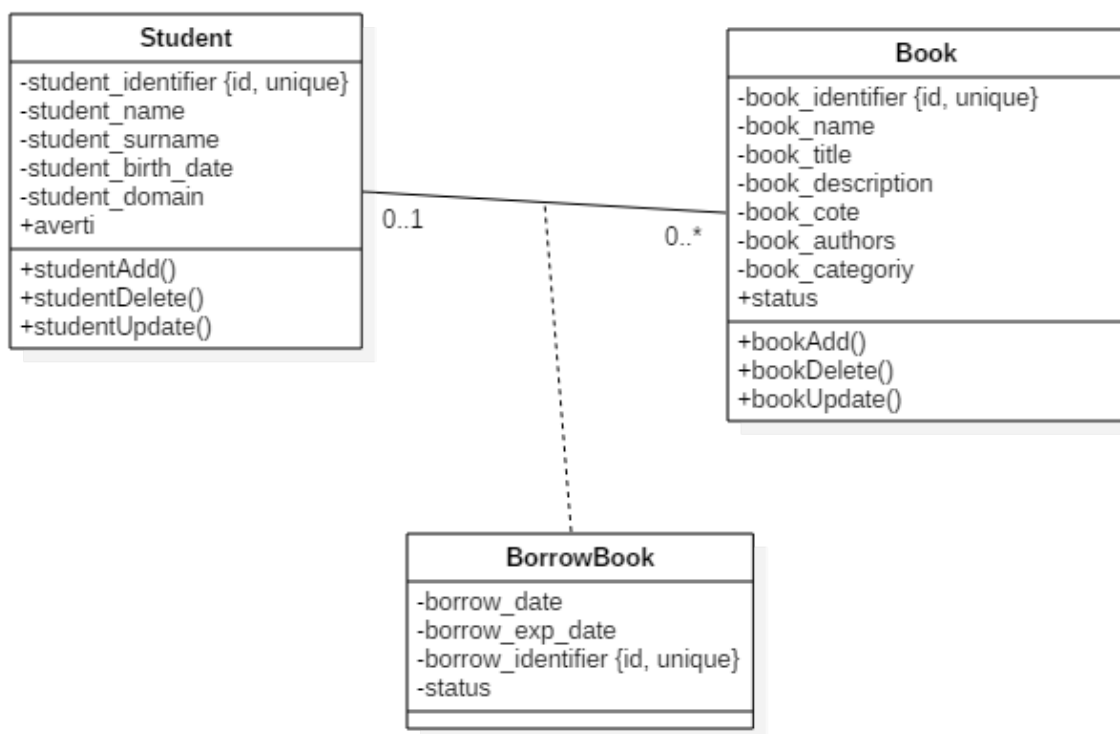


FIGURE 4.11 – Diagramme de classe de l'application

Noms des attributs	Type attributs	Description des attributs
StudentIdentifier	Integer	Identifiant unique de l'étudiant
StudentName	String	Nom étudiant
StudentSurname	String	Prénom étudiant
StudentBirthDate	Date	Date naissance étudiant
StudentDomain	String	Filière d'étude étudiant
Averti	Boolean	Vrai si l'étudiant a reçu un avertissement
BookIdentifier	Integer	Identifiant unique du livre
BookTitle	String	Title du livre
BookDescription	String	Description du livre
BookCote	String	Emplacement du livre
BookAuthor	String	Auteur du livre
BookCategory	string	Domaine d'étude du livre
Status	Boolean	Vrai si le livre est en emprunt
BorrowDate	Date	Date d'emprunt
BorrowExpdate	Date	Date d'expiration de l'emprunt
Status	Boolean	Vrai si le livre a été retourné
studentAdd	Student	Insérer un étudiant dans la base de données
studentDelete	Void	Supprime un étudiant
studentUpdate	Student	Met à jour les information d'un étudiant
bookAdd	Book	Inserer un livre dans la base de données
bookDelete	Void	Supprime un livre
bookUpdate	Book	Met à jour les informations d'un livre

TABLE 4.1 – Opérations et attributs

Les fonctions des attributs et des opérations sont décrites sur le tableau suivant :

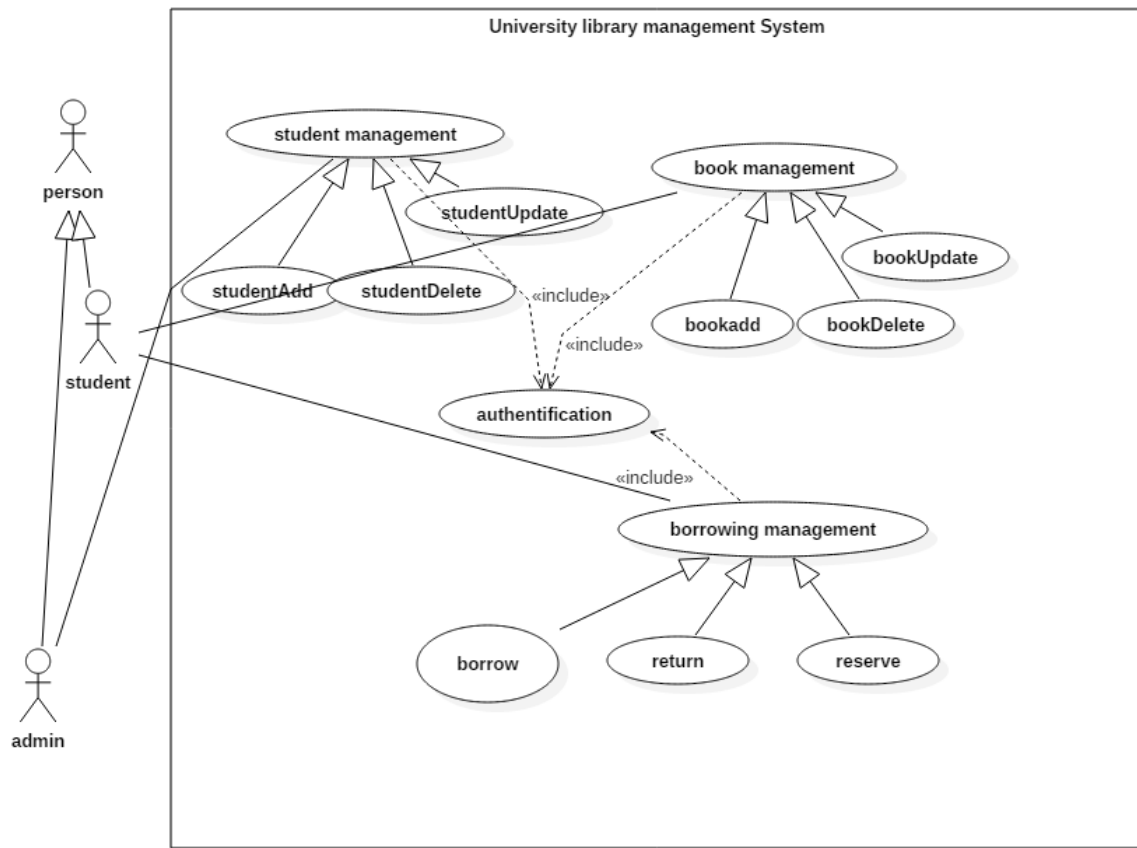


TABLE 4.2 – Diagramme de cas d'utilisation

Les fonctionnalités sont décrites comme suit :

- Student Management : c'est la fonctionnalité gestion des étudiants composée des sous fonctionnalités suivantes :

- studentAdd pour l'insertion d'une entité étudiant
- studentDelete pour la suppression d'une entité étudiant
- studentUpdate pour la mise à jour des informations d'une entité étudiant

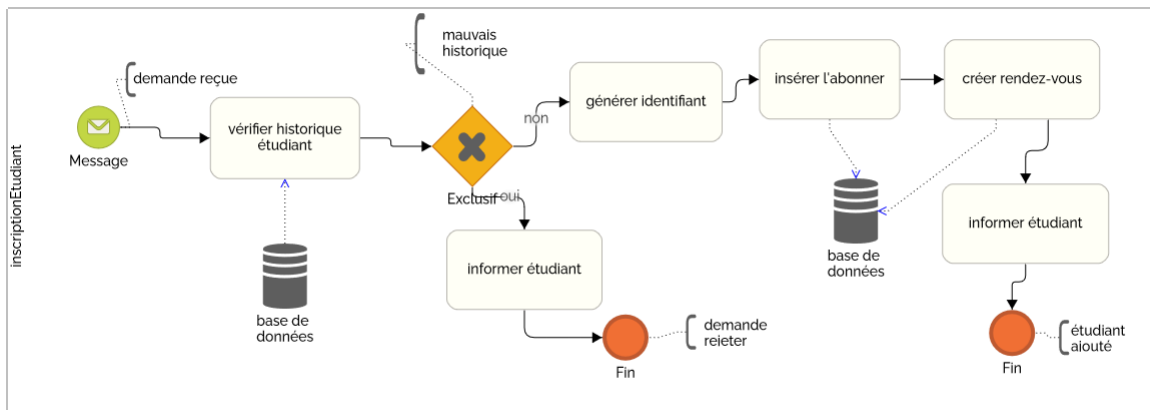
- Book Management : c'est la fonctionnalité gestion des livres composée des sous fonctionnalités suivantes :

- BookAdd pour l'insertion d'une entité livre
- BookDelete pour la suppression d'une entité livre
- BookUpdate pour la mise à jour des informations d'une entité livre

- Borrowing Management : C'est la fonctionnalité de gestion des emprunts. Elle se compose des sous fonctionnalités suivantes :

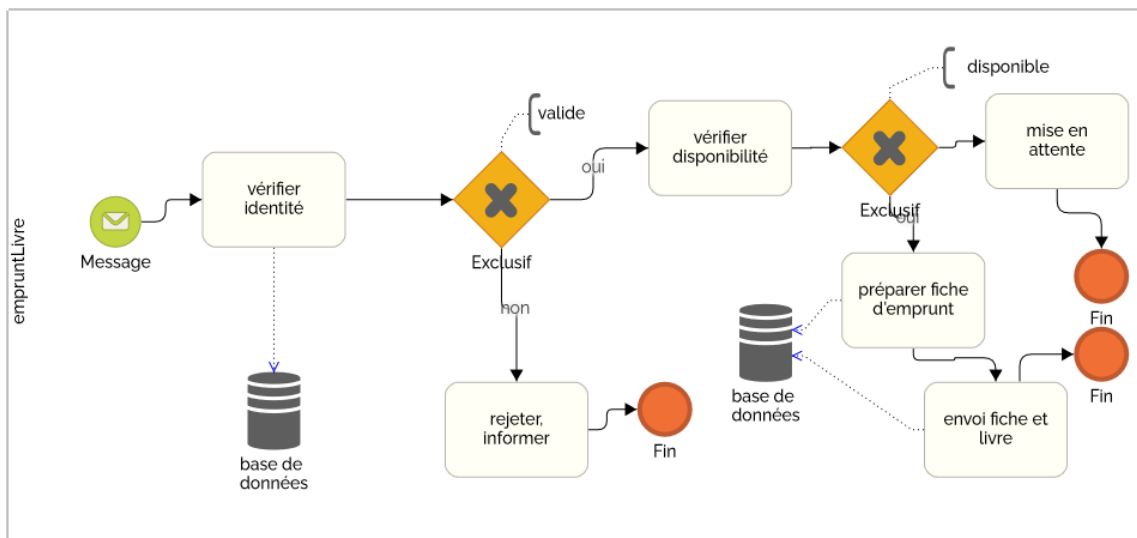
- Borrow : permet à un étudiant d'emprunter un livre
- Return : permet à un étudiant de rendre un livre
- Reserve : permet à un étudiant de réserver un livre

L'exécution de certaines fonctionnalités nécessite l'appel d'autres fonctionnalités. Le service appelant est appelé processus. Nous présentons ci-dessous les diagrammes BPMN des différents processus. La figure 4.3 présente le processus d'inscription d'un étudiant à la bibliothèque, la figure 4.4 le processus d'emprunt et la figure 4.5 le processus de remise d'un livre.



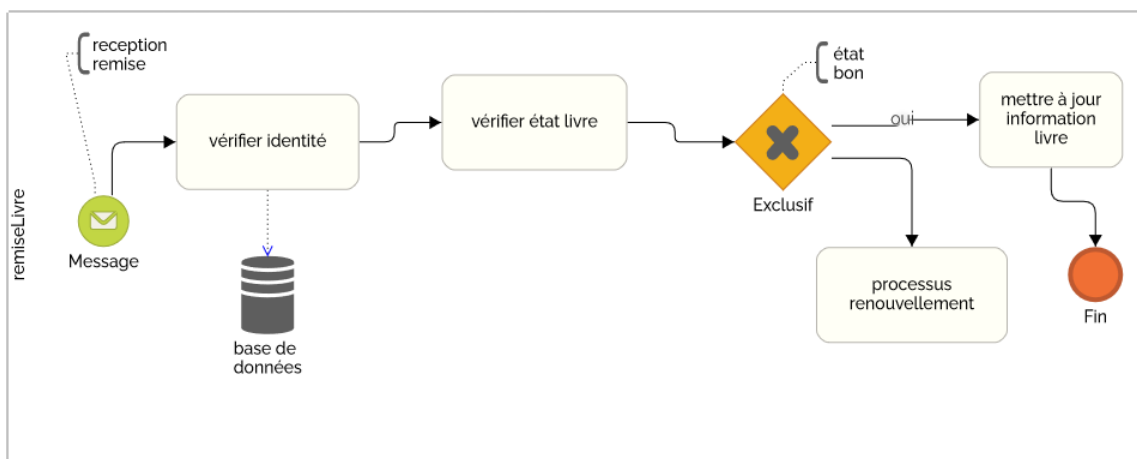
HEFLO

FIGURE 4.12 – Processus d’inscription



HEFLO

FIGURE 4.13 – Processus d’emprunt



HEFLO

FIGURE 4.14 – Processus de remise

## 4.4 Présentation des outils de mise en œuvre

### 4.4.1 Le Framework Spring boot

Spring Boot est un framework open source basé sur Java utilisé pour créer des microservices. Il est développé par Pivotal Team et est utilisé pour créer des applications de ressort autonomes et prêtes pour la production. Spring Boot fournit une bonne plate-forme aux développeurs Java pour développer une application Spring autonome. Vous pouvez commencer avec des configurations minimales sans avoir besoin d'une configuration complète de Spring. Spring Boot est conçu avec les objectifs suivants :

- Pour éviter une configuration XML complexe
- Pour développer plus facilement des applications Spring prêtes pour la production
- Pour réduire le temps de développement et exécuter l'application indépendamment
- Offrir un moyen plus simple pour démarrer avec l'application

Spring boot a les fonctionnalités et les avantages suivants :

- Il offre un moyen flexible de configurer des Java Beans, des configurations XML et des transactions de base de données.

- Il fournit un traitement par lots puissant et gère les points de terminaison REST.

- Dans Spring Boot, tout est configuré automatiquement ; aucune configuration manuelle n'est nécessaire.

- Il offre une application de ressort basée sur l'annotation

- Facilite la gestion des dépendances

- Il comprend un conteneur de servlet intégré Spring Boot configure automatiquement votre application en fonction des dépendances que vous avez ajoutées au projet à l'aide de l'annotation `@EnableAutoConfiguration` . Par exemple, si la base de données MySQL se trouve sur votre chemin de classe, mais que vous n'avez configuré aucune connexion à la base de données, Spring Boot configure automatiquement une base de données en mémoire. Le point d'entrée de l'application Spring Boot est la classe qui contient `@SpringBootApplication` annotation et la méthode principale. Spring Boot analyse automatiquement tous les composants inclus dans le projet à l'aide de l'annotation `@ComponentScan` . La gestion des dépendances est une tâche difficile pour les grands projets. Spring Boot résout ce problème en fournissant un ensemble de dépendances pour la commodité des développeurs. Par exemple, si vous souhaitez utiliser Spring et JPA pour accéder à la base de données, il suffit que vous incluez la dépendance `spring-boot-starter-data-jpa` dans votre projet

### 4.4.2 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 du leadership, la distribution sessions, état du cluster). La coordination des systèmes distribués, et à l'aide de Spring Cloud, les développeurs peuvent rapidement mettre en place des services et des applications qui implémentent ces modèles. Ils fonctionneront bien dans n'importe quel environnement distribué, y compris le propre ordinateur portable du développeur, les centres de données bare métal et les plates-formes gérées telles que Cloud Foundry. C'est par exemple avec le service de découverte que nous allons développer le service Gateway/proxy, la fonctionnalité de gestion de la configuration pour créer le service de configuration.

#### 4.4.2.1 Découverte (automatique) de services : clients Eureka

La découverte de services est l'un des principes clés d'une architecture basée sur les microservices. Essayer de configurer manuellement chaque client ou une forme de convention peut être difficile à faire et peut être fragile. Eureka est le serveur et client Netflix Service Discovery. Le serveur peut être configuré et déployé pour être hautement disponible, chaque serveur répliquant l'état des services enregistrés vers les autres. Lorsqu'un client s'enregistre auprès d'Eureka, il fournit des métadonnées sur lui-même, telles que l'hôte, le port, l'URL de l'indicateur de santé, la page d'accueil et d'autres détails. Eureka reçoit des messages de pulsation de chaque instance appartenant à un service. Si le battement de cœur échoue sur un calendrier configurable, l'instance est normalement supprimée du registre. Pour déclarer un service comme client Eureka, il suffit d'ajouter la dépendance Eureka Client à ce service et l'activer.

#### 4.4.2.2 Passerelle Spring Cloud

La plateforme Spring Cloud de Netflix fournit une passerelle API construite au-dessus de l'écosystème Spring, comprenant : Spring 5, Spring Boot 2 et Project Reactor. Spring Cloud Gateway vise à fournir un moyen simple mais efficace d'acheminer vers les API et de leur fournir des préoccupations transversales telles que : la sécurité, la surveillance / métriques et la résilience. Le diagramme suivant fournit un aperçu de haut niveau du fonctionnement de Spring Cloud Gateway :

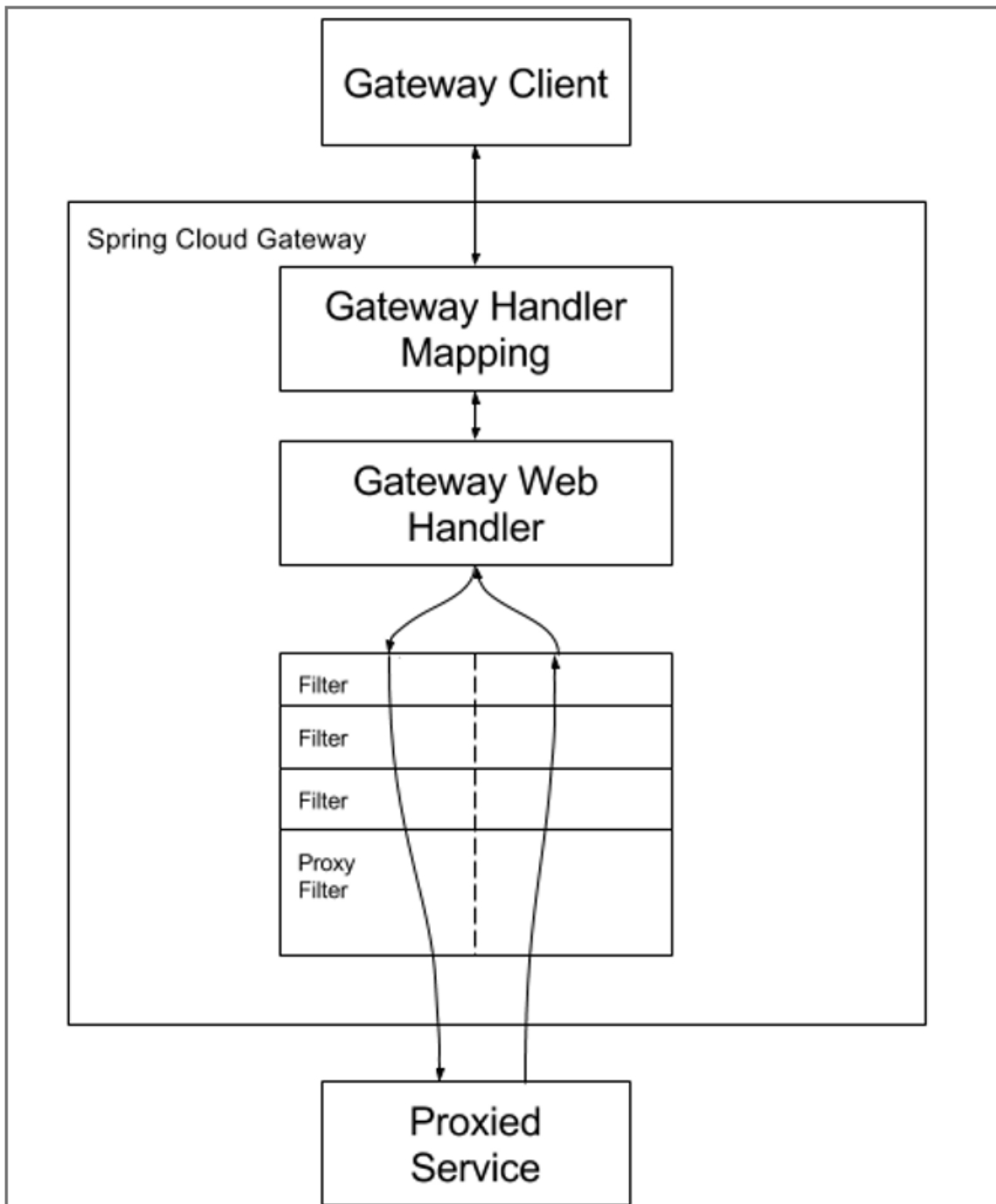


FIGURE 4.15 – Fonctionnement de la passerelle Spring Cloud (gateway)

Les clients adressent des demandes à Spring Cloud Gateway. Si le mappage du gestionnaire de passerelle détermine qu'une demande correspond à un itinéraire, elle est envoyée au gestionnaire Web de passerelle. Ce gestionnaire exécute la demande via une chaîne de filtres spécifique à la demande. La raison pour laquelle les filtres sont divisés par la ligne en pointillés est que les filtres peuvent exécuter une logique avant et après l'envoi de la demande de proxy. Toute la logique de «pré» filtre est exécutée. Ensuite, la demande de proxy est effectuée. Une fois la demande de proxy effectuée, la logique de filtrage «post» est exécutée.



### 4.4.3 Présentation de l'architecture microservice

La figure ci-dessous représente l'architecture microservice que nous avons construit :

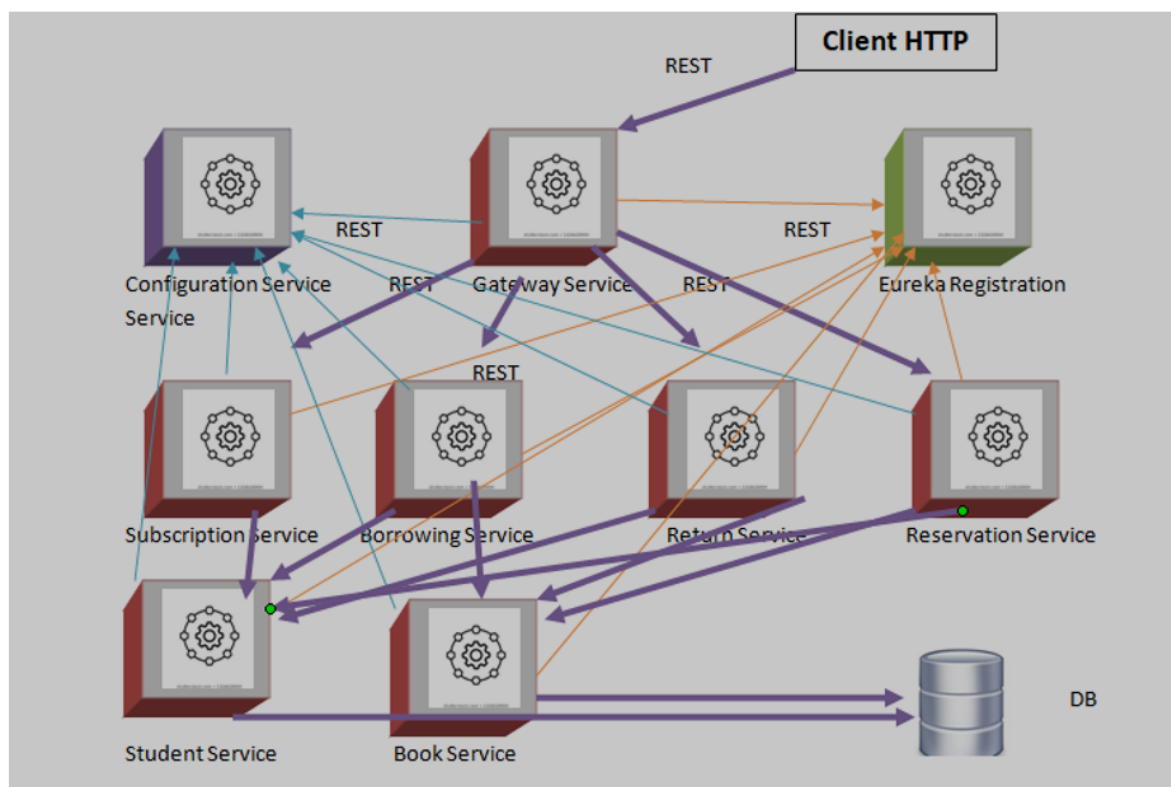


FIGURE 4.16 – Architecture microservice de l'application

-Le client http : C'est l'utilisateur qui cherche à consommer le service grâce par exemple à un navigateur web.

-Le microservice « Gateway Service » : C'est le service proxy que le client contacte pour demander l'exécution d'un processus (ensemble de microservices). Ce microservice dès qu'il reçoit une requête contacte le microservice « Eureka registration » pour localiser une instance du processus demandé. Il contacte ensuite les microservices concernés par la requête, qui lui retournent le résultat destiné au client.

-Le microservice « configuration service » : C'est le microservice qui contient les configurations initiales communes à tous les microservices. Chaque microservice qui démarre contacte ce microservice pour compléter sa configuration. Par exemple les microservices qui se connectent à une base de données vont contacter le microservice « configuration service » pour récupérer les identifiants de connexion à la base de données. -Le microservice « student service » : Il contient les fonctionnalités d'ajout, de mise à jour et de suppression des informations d'un étudiant. Ce microservice est utile pour l'inscription d'un étudiant et l'emprunt des livres. -Le microservice « book service » : Il contient les fonctionnalités d'ajout, de mise à jour et de suppression des informations d'un livre. Ce microservice est utile pour l'ajout et l'emprunt des livres.

-DB est la base de données de la bibliothèque. Elle contient les informations sur les livres, les étudiants et les emprunts.

-« Subscription Service » : C'est le processus d'inscription d'un étudiant à la bibliothèque. Ce processus contacte le microservice « Student Service » pour s'exécuter. -« Borrowing Service » : C'est le processus d'emprunt des livres. Il contacte les microservices « Student Service » et « book service » pour s'exécuter. -« Return Service » : C'est le processus de remise des livres. Il contacte les

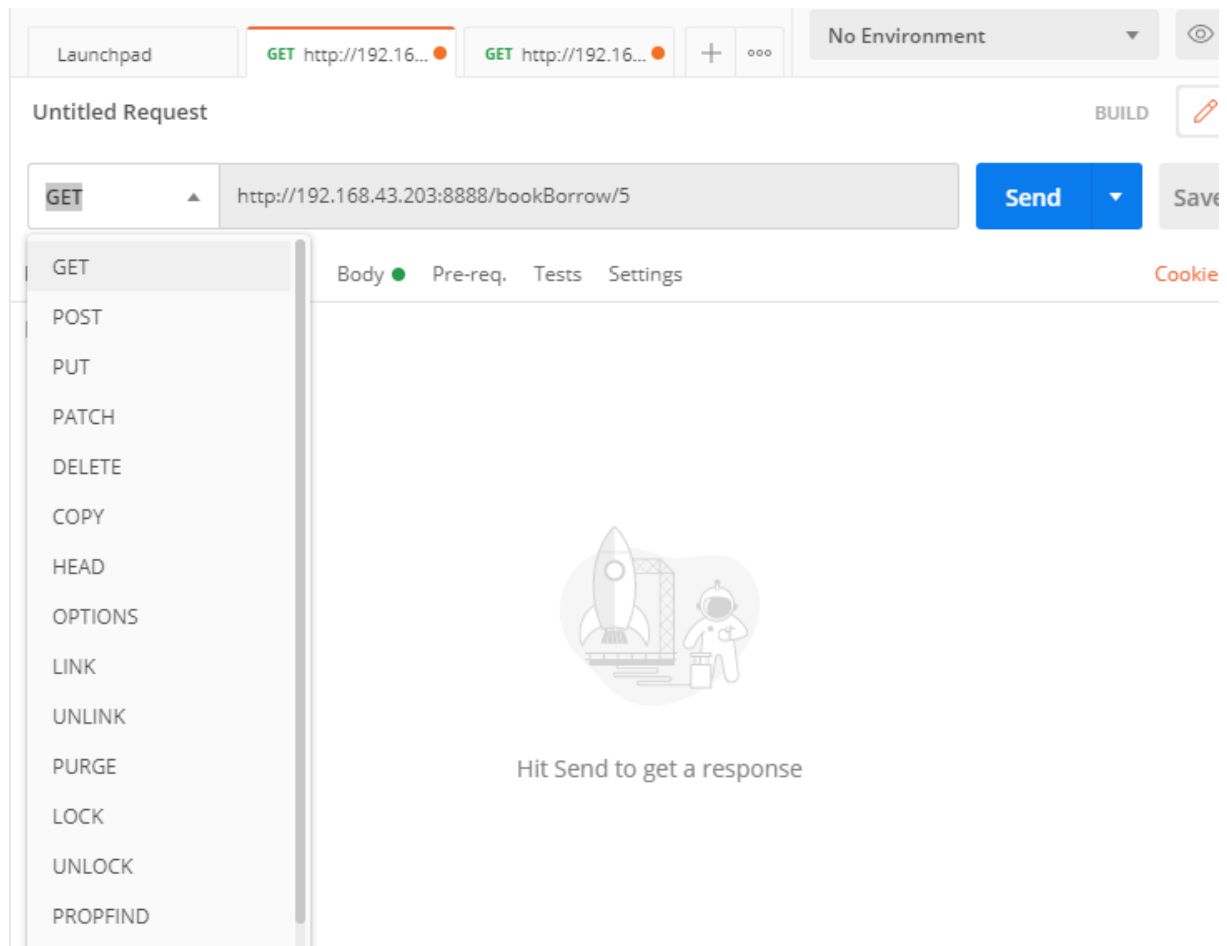


FIGURE 4.17 – Interface de Postman

microservices « **Student Service** », « **book service** » et « **Borrowing Service** » pour s'exécuter. - « **Reservation Service** » C'est le processus réservation des livres pour un emprunt ultérieur. Il contacte les microservices « **tudent Service** » et « **Book Service** » pour s'exécuter.

#### 4.4.4 Tests des services développés

Les tests ont été réalisés avec l'application postman qui est une plateforme dédiée au test des services. La figure ci-dessous montre l'interface graphique de postman :

#### 4.4.4.1 Les tests unitaires

Les tests unitaires consistent à vérifier que les microservices (services non composés) et les services de gestion comme le service d'enregistrement et de configuration fonctionnent comme prévu. • Test du student-service (service de gestion des étudiants) et des book-service (service de gestion des livres) Dans ces services sont développées les fonctionnalités essentielles comme l'insertion, la suppression, trouver les informations d'un élément (étudiant ou livre par son identifiant) et la récupération de la liste des étudiants ou de livres. Les tests ont été réalisés avec l'application postman qui est une plateforme dédiée au test des services. Les deux figures ci-dessous montrent ces fonctions dans la classe Controller (classe java dans laquelle sont implémentées les fonctions du service).

```
@RestController
@CrossOrigin(origins = "*")
public class StudentController {
    @Autowired
    StudentRepository studentRepository;

    @GetMapping(value="/studentList")
    List<Student> getStudentList() { return studentRepository.findAll(); }
    @GetMapping(value="/studentAdd")
    Student studentAdd(@RequestBody Student student) { return studentRepository.save(student); }
    @GetMapping(value="/studentDelete/{id}")
    void studentDelete(@PathVariable int id){
        studentRepository.deleteById(id);
    }
    @GetMapping(value="/studentFind/{id}")
    Optional<Student> studentFind(@PathVariable int id) { return studentRepository.findById(id); }
    @GetMapping(value="/studentExist/{id}")
    boolean StudentExist(@PathVariable int id) { return studentRepository.existsById(id); }
}
```

FIGURE 4.18 – Controller du microservice student-service

```

@RestController
@CrossOrigin(origins = "*")
public class StudentController {
    @Autowired
    StudentRepository studentRepository;

    @GetMapping(value="/studentList")
    List<Student> getStudentList() { return studentRepository.findAll(); }
    @GetMapping(value="/studentAdd")
    Student studentAdd(@RequestBody Student student) { return studentRepository.save(student); }
    @GetMapping(value="/studentDelete/{id}")
    void studentDelete(@PathVariable int id){

        studentRepository.deleteById(id);
    }
    @GetMapping(value="/studentFind/{id}")
    Optional<Student> studentFind(@PathVariable int id) { return studentRepository.findById(id); }
    @GetMapping(value="/studentExist/{id}")
    boolean StudentExist(@PathVariable int id) { return studentRepository.existsById(id); }
}

```

FIGURE 4.19 – Controller du microservice book-service

Les deux figures ci-dessous montre le résultat de la récupération de la liste des étudiants et des livres à l'aide de l'application Postman sous un format JSON.

The screenshot shows a Postman interface for a GET request to `http://192.168.43.203:8081/studentList`. The response status is 200 OK, with a time of 23.39 s and a size of 1.88 KB. The response body is displayed in JSON format, showing a list of three students:

```

1 [
2   {
3     "studentIdentifier": 44231501,
4     "studentName": "Rahim",
5     "studentSurname": "Savadogo",
6     "studentBirthDate": "2020-09-26T12:58:59.000+00:00",
7     "studentDomain": "informatique",
8     "email": "savhama@gmail.com"
9   },
10  {
11   "studentIdentifier": 44231502,
12   "studentName": "Mohammed",
13   "studentSurname": "Kabore",
14   "studentBirthDate": "2020-09-26T12:59:06.000+00:00",
15   "studentDomain": "informatique",
16   "email": "soacloud2020@gmail.com"
17  },
18  {
19   "studentIdentifier": 44231503,
20   "studentName": "Malik",
21   ...

```

FIGURE 4.20 – Liste des étudiants

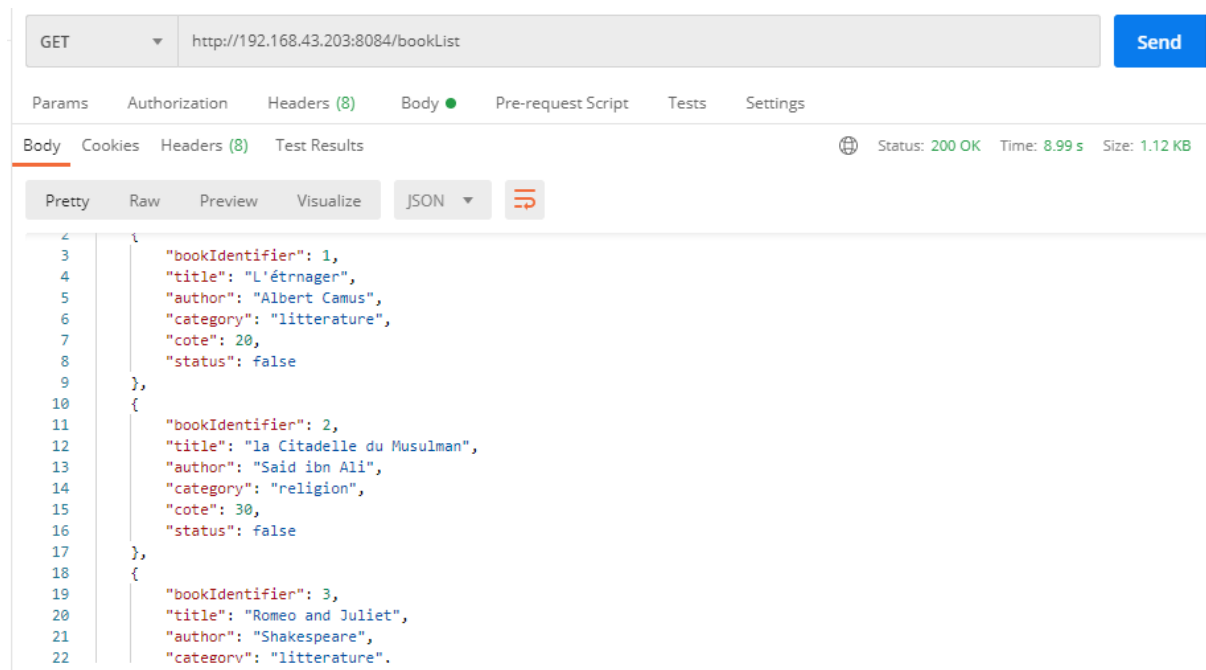


FIGURE 4.21 – Liste des étudiants

Les quatre figures ci-dessous montrent le résultat de la l'insertion d'un étudiant et d'un livre à l'aide de l'application Postman sous un format JSON La première montre le résultat d'une nouvelle insertion :

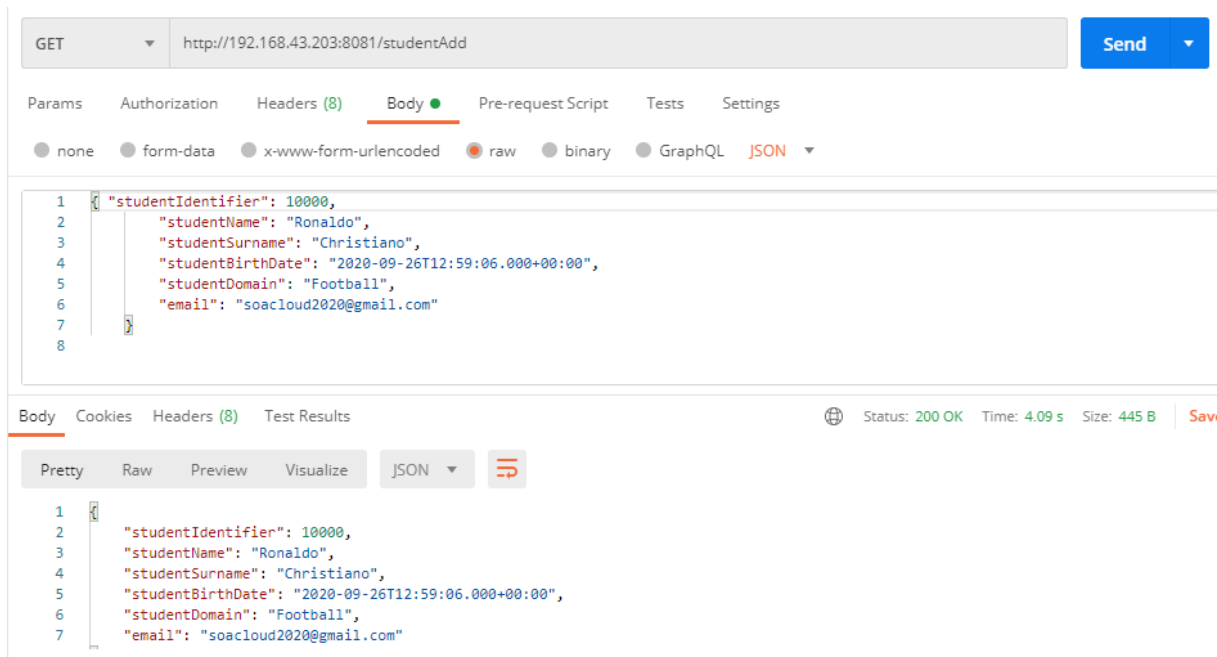


FIGURE 4.22 – Insertion d'un nouvel étudiant dans la liste

Et une nouvelle récupération de la liste des étudiant montre bien ce nouvel élément dans la figure suivante :

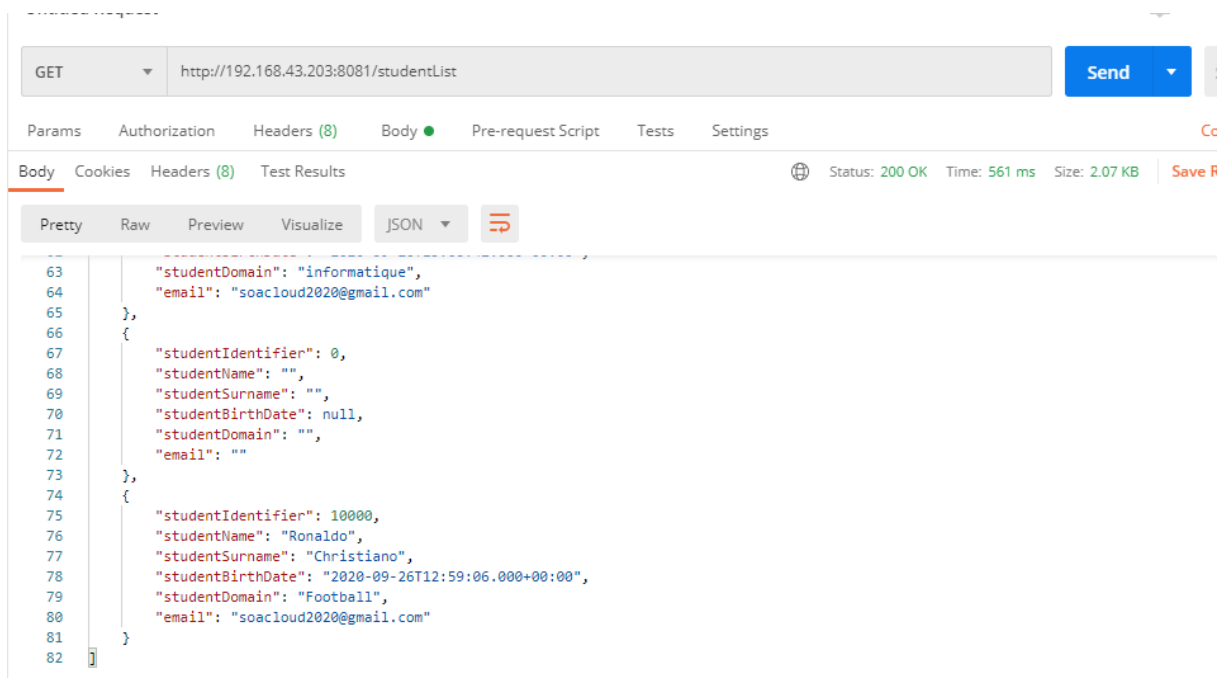


FIGURE 4.23 – Liste des étudiants après insertion d’un élément

Comme vu avec le microservice student-service, les deux figures qui suivent montrent le résultat d’insertion du microservice book-service :

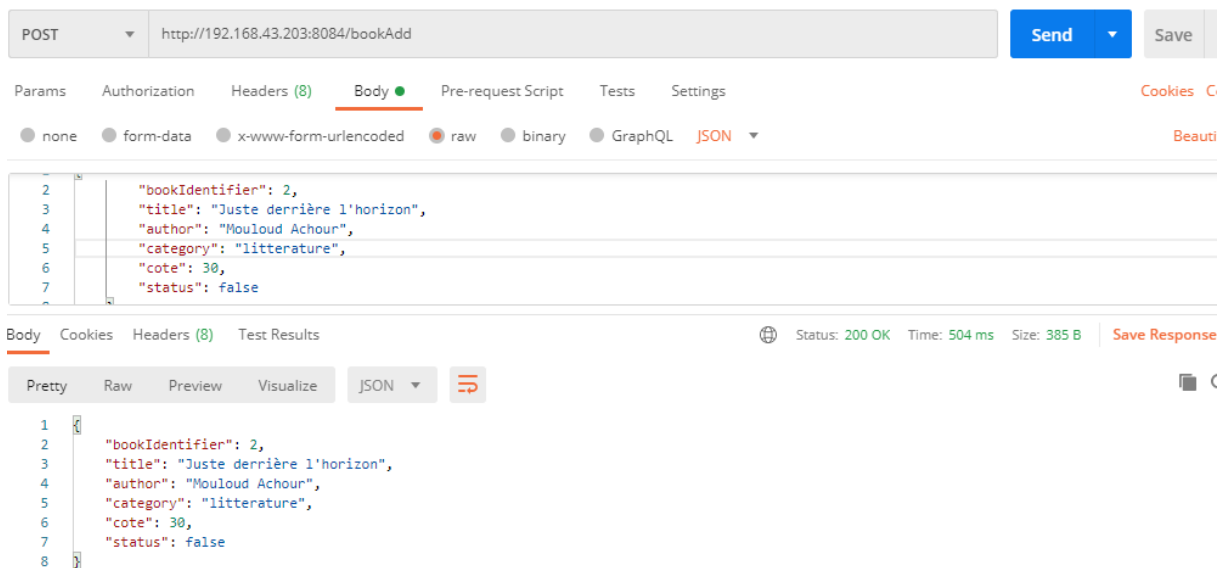


FIGURE 4.24 – Insertion d’un nouveau livre

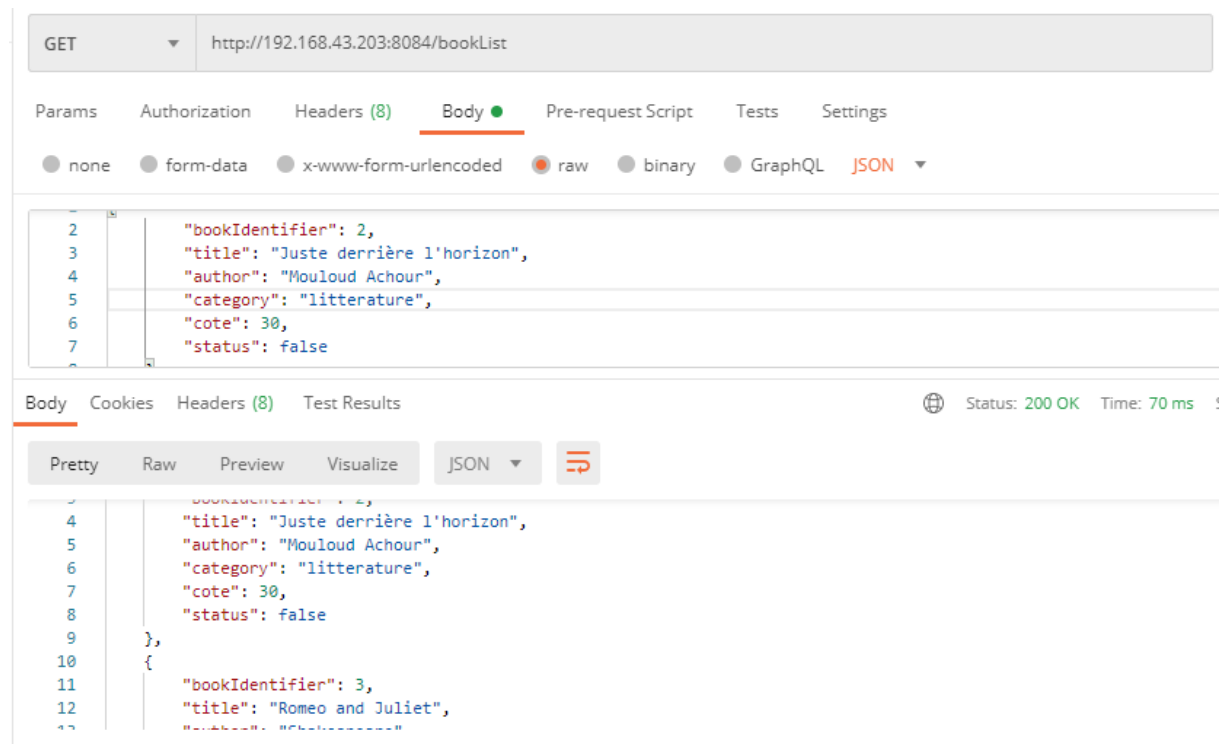


FIGURE 4.25 – Liste des livres après l’insertion

Nous allons également présenter le cas de suppression dans les figures qui suivent :

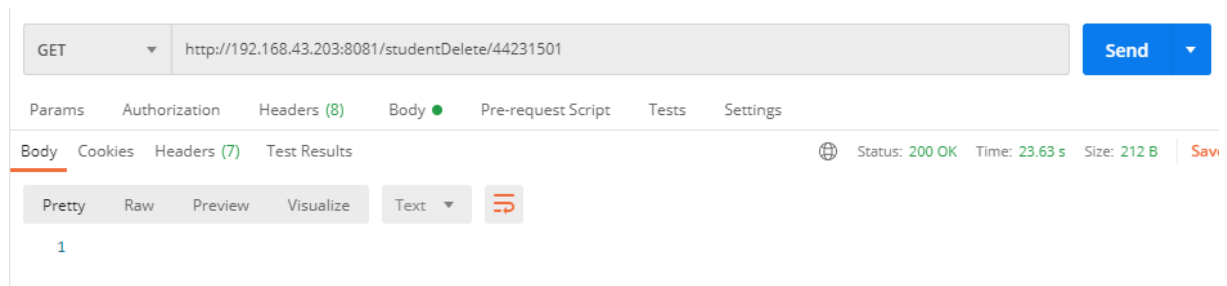
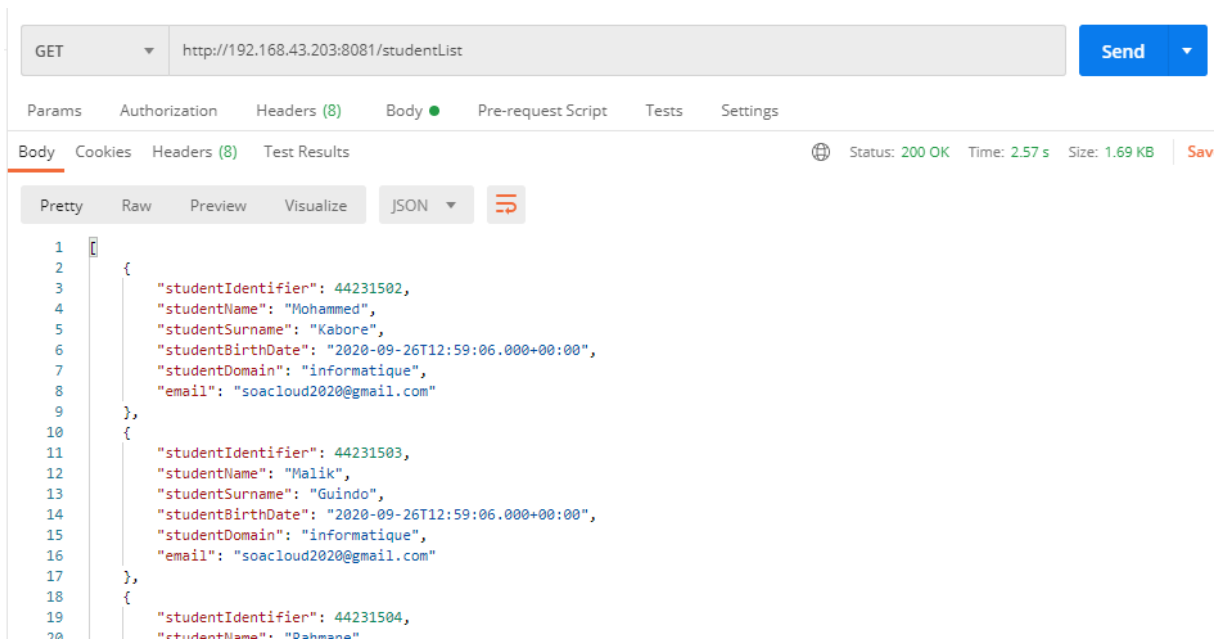


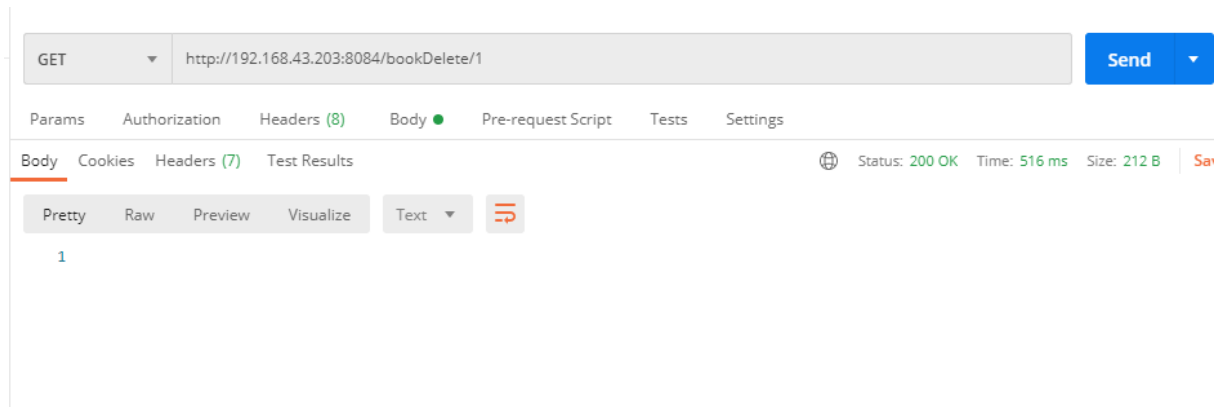
FIGURE 4.26 – Suppression d’un étudiant



The screenshot shows a REST client interface with a GET request to `http://192.168.43.203:8081/studentList`. The response is a JSON array of three student objects. The status is 200 OK, the time is 2.57 s, and the size is 1.69 KB.

```
1 [
2   {
3     "studentIdentifier": 44231502,
4     "studentName": "Mohammed",
5     "studentSurname": "Kabore",
6     "studentBirthDate": "2020-09-26T12:59:06.000+00:00",
7     "studentDomain": "informatique",
8     "email": "soacloud2020@gmail.com"
9   },
10  {
11     "studentIdentifier": 44231503,
12     "studentName": "Malik",
13     "studentSurname": "Guindo",
14     "studentBirthDate": "2020-09-26T12:59:06.000+00:00",
15     "studentDomain": "informatique",
16     "email": "soacloud2020@gmail.com"
17   },
18  {
19     "studentIdentifier": 44231504,
20     "studentName": "Dahmane"
```

FIGURE 4.27 – Liste des étudiants après la suppression



The screenshot shows a REST client interface with a GET request to `http://192.168.43.203:8084/bookDelete/1`. The response is empty. The status is 200 OK, the time is 516 ms, and the size is 212 B.

FIGURE 4.28 – Suppression d'un livre



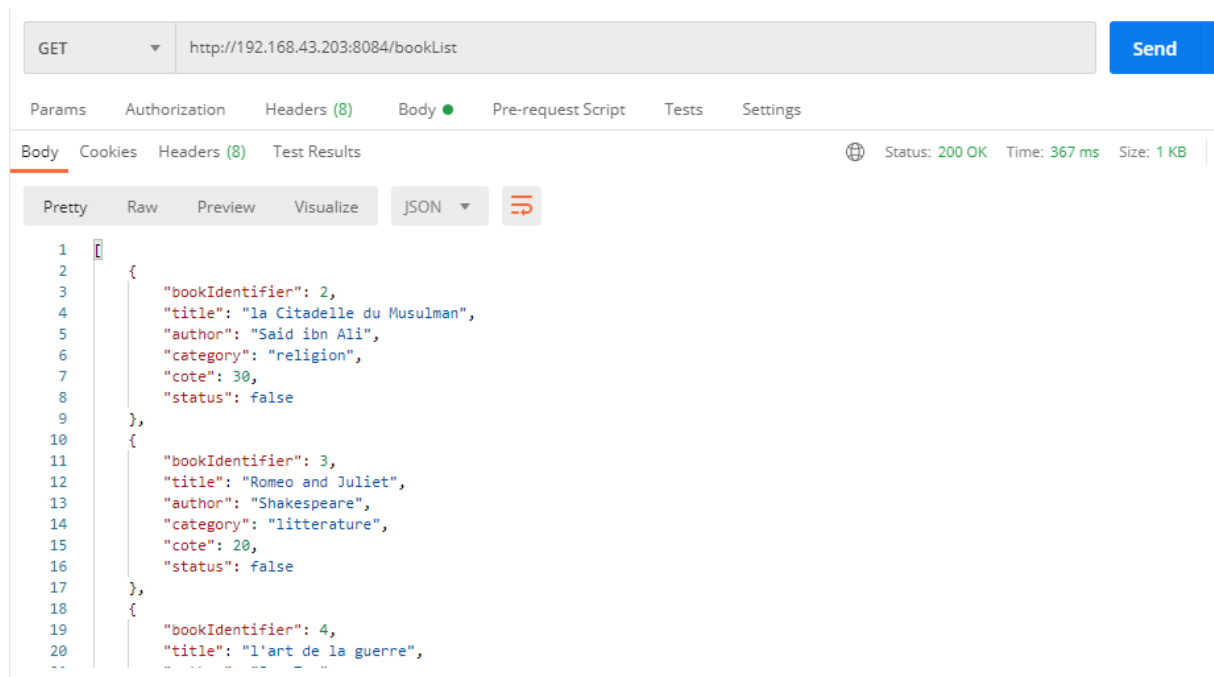


FIGURE 4.29 – Liste des livres après la suppression

La figure suivante montre le résultat d'exécution du service registry-service dans lequel chaque service qui démarre s'enregistre.

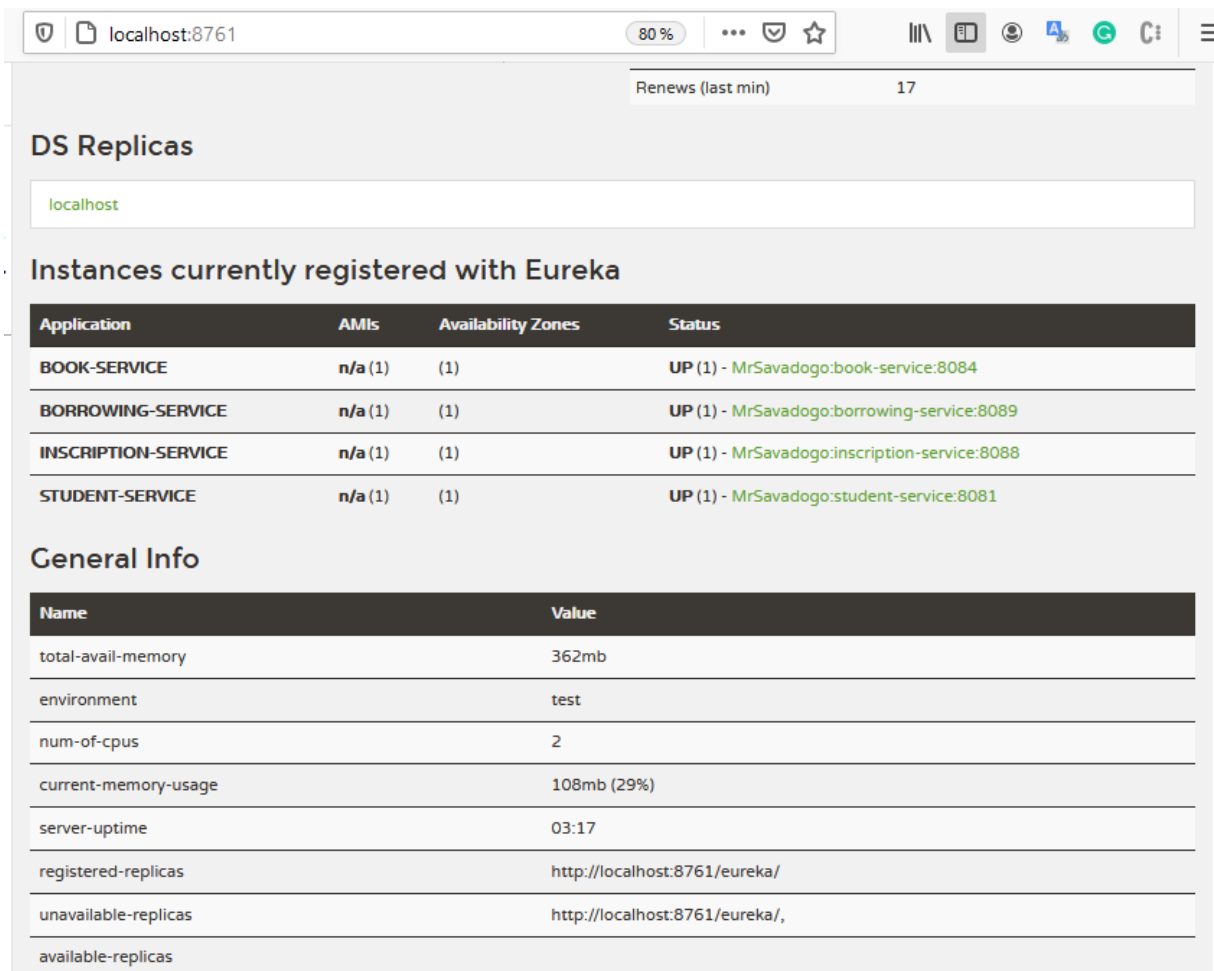


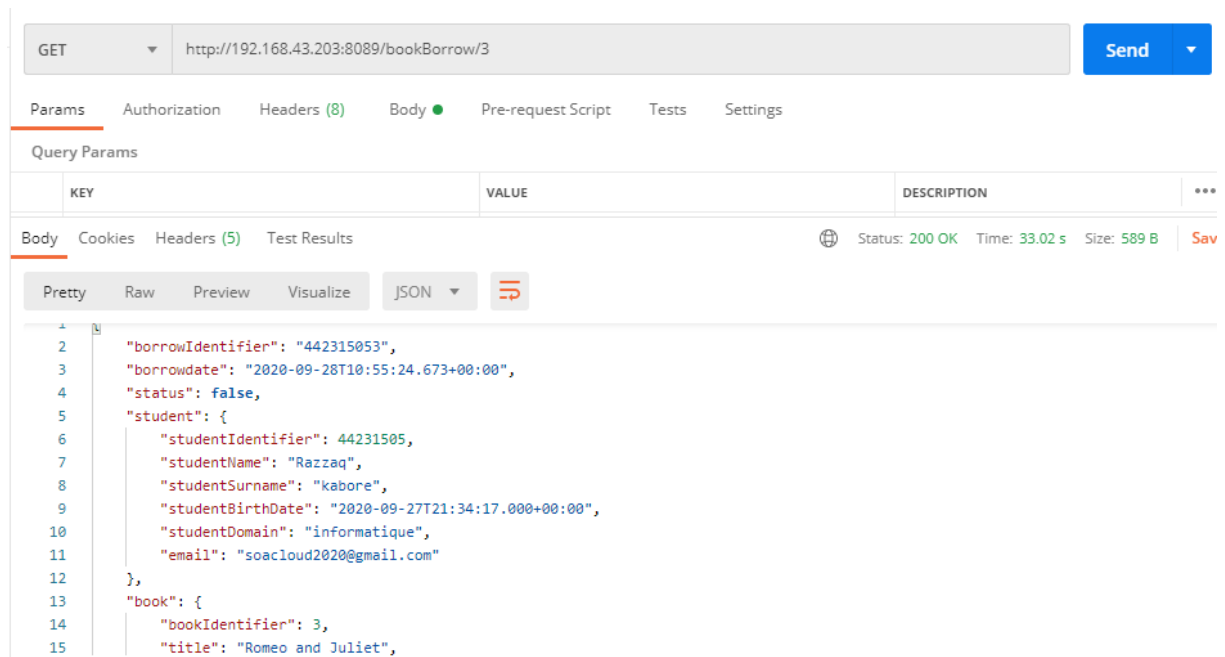
FIGURE 4.30 – Service d’enregistrement des services

Nous voyons que les services démarrés tels book-service, borrowing-service, inscription-service et student-service ont tous publié leur référence dans ce service d’enregistrement. Ces références seront utilisées par le service proxy pour les différentes requêtes.

#### 4.4.5 Test des services Composés

Les services composés sont les services d’inscription des étudiants, d’emprunt et de remise des livres. Nous profiterons faire les requêtes depuis le proxy/gateway pour certains services.

- Le service d’emprunt (borrowing-service). Il permet d’emprunter un livre connaissant son identifiant. Il démarre sur le port 8089. La figure ci-dessous montre le résultat d’une requête d’emprunt de livre identifié par 3. Celui qui fait l’emprunt est défini par défaut pour les tests d’identifiant 44231505. L’identifiant de l’emprunt est la concaténation des deux identifiant, ce qui donne 442315053.



GET http://192.168.43.203:8089/bookBorrow/3

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION	...

Body Cookies Headers (5) Test Results Status: 200 OK Time: 33.02 s Size: 589 B Sav

Pretty Raw Preview Visualize JSON

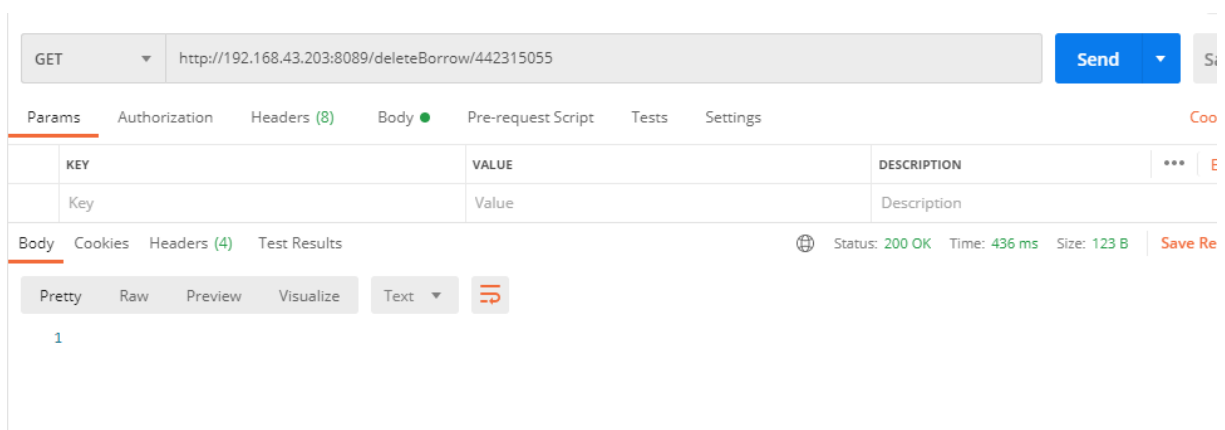
```

1
2  "borrowIdentifier": "442315053",
3  "borrowdate": "2020-09-28T10:55:24.673+00:00",
4  "status": false,
5  "student": {
6    "studentIdentifier": 44231505,
7    "studentName": "Razzaq",
8    "studentSurname": "kabore",
9    "studentBirthDate": "2020-09-27T21:34:17.000+00:00",
10   "studentDomain": "informatique",
11   "email": "soacloud2020@gmail.com"
12  },
13  "book": {
14    "bookIdentifier": 3,
15    "title": "Romeo and Juliet",

```

FIGURE 4.31 – Résultat d'un emprunt

La figure ci-dessous montre le résultat de la suppression d'un emprunt. C'est une fonction qui va servir au service remise pour supprimer l'instance d'emprunt dans la base de données. La requête retourne 1 ce qui signifie que l'opération a été effectuée.



GET http://192.168.43.203:8089/deleteBorrow/442315055

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION	...
Key	Value	Description	

Body Cookies Headers (4) Test Results Status: 200 OK Time: 436 ms Size: 123 B Save Re

Pretty Raw Preview Visualize Text

```

1

```

FIGURE 4.32 – Suppression d'un emprunt

- Quelques requêtes à partir du service gateway. Ce service récupère une requête d'un client et la redirige vers le service approprié. Avant la redirection, il doit contacter le service d'enregistrement pour récupérer l'adresse d'une instance du service. La figure suivante montre les différentes URL acceptés par ce service proxy.

```

public RouteLocator routes(RouteLocatorBuilder builder) {
    return builder.routes()
        .route(r -> r.path(...patterns: "/studentList/**").uri("lb://student-service").id("studentList"))
        .route(r -> r.path(...patterns: "/studentFind/**").uri("lb://student-service").id("studentFind"))
        .route(r -> r.path(...patterns: "/studentDelete/**").uri("lb://student-service").id("studentDelete"))
        .route(r -> r.path(...patterns: "/studentAdd/**").uri("lb://student-service").id("studentAdd"))
        .route(r -> r.path(...patterns: "/studentExist/**").uri("lb://student-service").id("studentExist"))

        .route(r -> r.path(...patterns: "/bookList/**").uri("lb://book-service").id("bookList"))
        .route(r -> r.path(...patterns: "/bookAdd/**").uri("lb://book-service").id("bookAdd"))
        .route(r -> r.path(...patterns: "/bookDelete/**").uri("lb://book-service").id("bookDelete"))
        .route(r -> r.path(...patterns: "/bookFind/**").uri("lb://book-service").id("bookFind"))
        .route(r -> r.path(...patterns: "/bookExist/**").uri("lb://book-service").id("bookExist"))

        .route(r -> r.path(...patterns: "/bookId/**").uri("lb://id-service").id("bookId"))
        .route(r -> r.path(...patterns: "/studentId/**").uri("lb://id-service").id("studentId"))
        .route(r -> r.path(...patterns: "/sendEmail/**").uri("lb://email-service").id("sendEmail"))
        .route(r -> r.path(...patterns: "/bookBorrow/**").uri("lb://bookBorrow-service").id("bookBorrow"))
        .route(r -> r.path(...patterns: "/bookReturn/**").uri("lb://bookReturn-service").id("bookReturn"))
        .route(r -> r.path(...patterns: "/inscription/**").uri("lb://inscription-service").id("inscription"))
}

```

FIGURE 4.33 – liste des requêtes semi-dynamique du service Gateway

Lorsque le client envoie la requête `/studentList` comme on peut le voir sur la figure par le paramètre `path`, le service Gateway consulte le service d'enregistrement pour récupérer l'instance du service correspondant qui est indiqué par le paramètre « `lb://student-service` ». Dès qu'il localise l'instance du service demandé, il lui transfère la requête et attend le résultat de l'exécution. On entend par attendre le fait que la réponse à la requête sera retransmise au service Gateway. Les requêtes sont asynchrones. La figure suivante montre le résultat de l'exécution de la requête `/studentList` à partir du service Gateway.

```

GET http://192.168.43.203:8888/studentList
Status: 200 OK Time: 1 m 48.90 s Size: 2.02 KB
Save Response

Body
Pretty Raw Preview Visualize JSON
1 {
2   {
3     "studentIdentifier": 44231501,
4     "studentName": "Rahim",
5     "studentSurname": "Savadogo",
6     "studentBirthDate": "2020-09-29T10:19:41.000+00:00",
7     "studentDomain": "informatique",
8     "email": "savhama@gmail.com"
9   },
10  {
11   "studentIdentifier": 44231502,
12   "studentName": "Mohammed",
13   "studentSurname": "Kabore",
14   "studentBirthDate": "2020-09-29T10:19:44.000+00:00",
15   "studentDomain": "informatique",
16   "email": "soaccloud2020@gmail.com"
17  },
18  {

```

FIGURE 4.34 – Exemple de requête à partir du gateway

#### 4.4.6 Validation des résultats

De façon générale les services développés fonctionnent et les requêtes sont exécutables depuis le service proxy/Gateway à partir duquel on peut sécuriser l'accès aux services fonctionnels. Mais la question de

sécurité n'a pas été réellement traitées et pourrait devenir l'objet de nos futures recherches.

## 4.5 Comparaison entre SOA et applications monolithiques

Architecture orientée service	Application monolithique
Assure une interopérabilité intrinsèque : l'adoption du concept de service permet aux différentes solutions du système TI d'échanger des données et des fonctionnalités entre elles, même s'ils sont développés en différents langages.	Ne supporte pas l'interopérabilité : Les applications monolithiques sont développées avec une seule technologie et intègre toutes leurs fonctionnalités en leur sein. Cette structure ne leur permet pas de communiquer avec d'autres applications
Minimise l'investissement initial et permet la réutilisation : dans une SOA on peut utiliser des composants déjà existants, quel que soit le langage de développement utilisé et quel que soit la plateforme sur laquelle ils tournent. Ceci permet un développement rapide de nouveaux services métier et un gain en termes de temps et d'argent.	Les applications monolithiques manquent de flexibilité. Les composants forment un bloc et l'exécution du code est possible seulement si tous les composants de l'application sont ensemble. Elles sont également dépendantes de leur plateforme de développement
Accroître l'agilité de l'organisation : cette agilité est mesurée par l'efficacité par laquelle une entreprise peut répondre au changement (réactivité).	Les applications monolithiques sont sensibles aux changements. La modification d'une partie du code peut entraîner un dysfonctionnement de l'application entière et un redéploiement complet
La liberté technologique : la SOA ne suit pas une approche universelle. Les équipes peuvent choisir le meilleur outil pour résoudre des problèmes spécifiques.	Les équipes sont obligées d'utiliser la même technologie durant tout le processus de développement.

TABLE 4.3 – Comparaison entre SOA et applications monolithiques

Les données du tableau montrent que l'adoption de la SOA améliore les rendements de l'entreprise en termes d'agilité, de temps de production et la durée de production. Elle minimise les risques et augmente l'évolutivité. En plus de ces avantages de la SOA par rapport aux applications monolithiques, l'avènement du Cloud Computing a permis de mettre à disposition des départements des systèmes d'informations des SI performants et innovants. L'intégration de l'architecture SOA dans les SI existants permet de tirer le meilleur de la flexibilité et de la performance apportée par le Cloud Computing tout en rapprochant le SI des directions métiers. La synthèse de ces outils permet l'éclosion d'un SI réellement flexible, capable rester au plus proche des besoins métiers et en apportant une performance de long terme à l'entreprise.

# Conclusion Générale

L'architecture orientée service est un sujet de recherche au cœur de la stratégie du SI de l'entreprise. Elle a permis au système d'information d'avoir une agilité nécessaire pour supporter les processus métier. Plusieurs technologies ont été proposées pour sa mise en œuvre dont CORBA, RMI, RPC, les technologies basées sur les composants logiciels, et les services web. Cette dernière avec l'utilisation des standards comme XML, demeure la solution la plus utilisée et la plus appropriée car respectant tous les principes de la SOA.

Le cloud computing est un modèle de livraison de services aux clients basé sur des infrastructures virtuelles. Les services peuvent être sous formes d'infrastructures(IaaS), de plateformes(PaaS), ou de logiciels (SaaS);le client ne s'occupe pas de l'entretien des services offerts. Les services sont gratuits ou payant selon une formule Pay-Per-Use. L'accès aux services (services cloud) se fait via le web, ce qui fait de lui une technologie puissante et durable pour les SI des entreprises. Ce projet a traité de la mise en place d'une architecture orientée services basée sur le Cloud Computing. Le travail a consisté en clair à l'étude de mise en œuvre de services, liés aux fonctions métier de l'entreprise, à l'aide du cloud computing.Pour enrichir et donner un aspect concret à notre étude, Nous avons d'abord créé un Cloud privé à l'aide de Opennebula qui est un système de gestion de cloud simple et flexible. Ensuite nous avons réalisé une application à base d' SOA de gestion de bibliothèque. Les services ont été implémentés à l'aide du Framework Spring boot en tant que microservices qui respectent les principes de la SOA avec une granulométrie plus fine et des processus conçus par orchestration automatique des microservice ont été créés en tant que services.Enfin l'ensemble des services a été déployé sous une machine virtuelle du cloud utilisant Linux Mint comme système d'exploitation. Le travail réalisé donne des résultats concluants. Cependant, nos recherches sont basées essentiellement sur l'aspect fonctionnel des services. L'aspect technique comme la sécurité n'a pas fait l'objet d'une recherche poussée et sera pris en considération dans nos futures recherches.

# Bibliographie

- [1] Soa. <https://www.microsoft.com>.
- [2] <https://www.lemagit.fr/conseil/Architecture-monolithique-vs-microservices-avantages-et-inconvenients>.
- [3] Occello Audrey. *Introduction à l'architecture orientée service*, 2006.
- [4] Haniche Fayçal. *Une contribution à l'agilité du système d'information de l'entreprise par intégration des legacy systems dans une architecture SOA*. PhD thesis, Université des Sciences et Technologie Faculté Electronique et Informatique, 2011.
- [5] Syntec Informatique. *Les architectures orientées services*. Collection Thematic, 2007.
- [6] <https://www.commentcamarche.net/contents/1241-soa-architecture-orientee-service>.
- [7] Rpc. <https://www.ionos.fr/digitalguide/serveur/know-how/quest-ce-que-le-remote-procedure-call/>.
- [8] Sacha Krakowiak. *Introduction aux objets répartis Java RMI*, 2004.
- [9] Serge Midonnet. Middlewares java rmi corba.
- [10] Chouki TIBERMACHINE. *Introduction au développement par composants (distribués) Java EE*.
- [11] Michel RIVEILL. Les entreprises java beans.
- [12] Jean-Charles Tournier. *Une Architecture à Base de Composants pour la Gestion de la Qualité de Service dans les Systèmes Embarqués Mobiles*. PhD thesis, CITI/INSA Lyon, 2005.
- [13] Eric Cariou. Middlewares java rmi corba.
- [14] Virginie Amar. *Le standard CORBA*, 1998.
- [15] Didier DONSEZ. *Message Oriented Middleware (MOM), Java Message Service (JMS)*, 2012.
- [16] jean-michel Douin. Jms,mom, mdb java message service, message-oriented middleware message-driven bean. 2013.
- [17] Martin Keen Amit Acharya Susan Bishop Alan Hopkins Sven Milinski Chris Nott Rick Robinson Jonathan Adams Paul Verschueren. *Patterns : Implementing an SOA Using an Enterprise Service Bus*. IBM, 2004.
- [18] Esb. <https://www.ibm.com/cloud/learn/esb>.
- [19] <https://redhat.com/fr/topics/microservices/what-are-microservices>.
- [20] <https://bluesoft-group.com/soa-microservices-deux-architectures-si/>.
- [21] BARKAT Abdelbasset. *Composition de service web dans le cloud computing*. PhD thesis, Université Mohamed KHIDER - BISKRA Faculté des Sciences exactes et de Sciences de la nature et de la vie Département Informatique, 2018.
- [22] Jan Mendling. Business process execution language for web service (bpel).
- [23] ADNANI Yahia BELHABIB Amine. *Architecture pour le Cloud Computing Mobile en mode P2P centralisé*. PhD thesis, Université Abderrahmane Mira de Béjaia, 2013.

- 
- [24] Evolution du cloud. <https://the-report.cloud/the-evolution-of-cloud-computing-wheres-it-going-next>.
- [25] Infrastructure as a service.
- [26] Weijie Ou Cheng Zeng, Xiao Guo and Dong Han. Cloud computing service composition and search based on semantic. 2009.
- [27] <https://www.guru99.com/advantages-disadvantages-cloud-computing.html>.
- [28] Carlos A. F. Maron† Claudio Schepke Adriano Vogel, Dalvan Griebler and Luiz Gustavo Fernandes. Private iaas clouds : A comparative analysis of opennebula, cloudstack and openstack. 2016.
- [29] <https://docs.opennebula.io/5.10>.



# Annexe :Installation et configuration de Opennebula

Pour ajouter le référentiel OpenNebula sur Debian / Ubuntu, exécutez en tant que root :

```
$wget -q -O- https://downloads.opennebula.org/repo/repo.key | apt-key add  
_   
$echo "deb https://downloads.opennebula.org/repo/5.8/Ubuntu/18.04 stable  
opennebula" > /etc/apt/sources
```

Pour installer le front-end, Il faut exécuter la commande suivante en temps que root :

```
$apt-get update  
$apt-get install opennebula opennebula-sunstone opennebula-gate  
opennebula-flow
```

La figure suivante montre les differents packets disponible pour une distribution UBUNTU :

## Debian/Ubuntu Package Description

These are the packages available for these distributions:

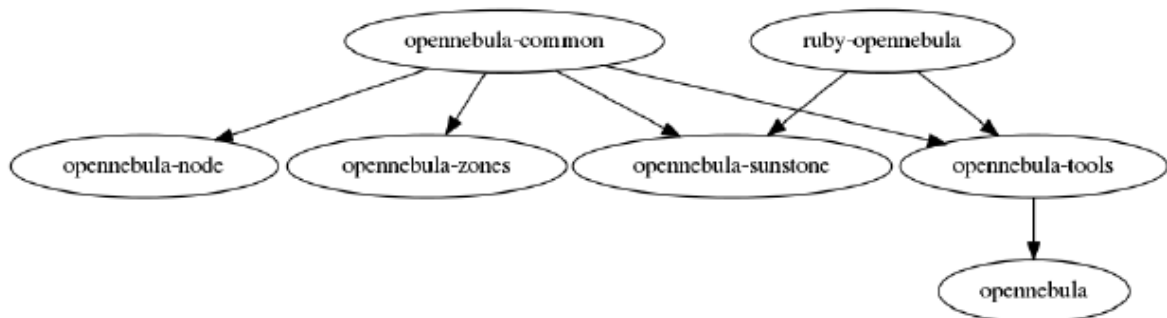


FIGURE 4.35 – Les packets opennebula sur UBUNTU

**opennebula-common** : Fournit l'utilisateur et les fichiers communs.

**ruby-opennebula** : API Ruby.

**libopennebula-java** : API Java.

**python-pyone** : API Python.

**libopennebula-java-doc** : documentation de l'API Java.

**opennebula-node** : prépare un nœud en tant que nœud KVM opennebula.

**opennebula-node-lxd** : prépare un nœud en tant que nœud opennebula LXD.

**opennebula-lxd-snap** : installe LXD snap (uniquement sur Ubuntu 16.04 et 18.04).

**opennebula-sunstone** : Sunstone (l'interface graphique).

**opennebula-tools** : interface de ligne de commande.

**opennebula-gate** : serveur OneGate qui permet la communication entre les VM et OpenNebula.

**opennebula-flow** : OneFlow gère les services et l'élasticité.

**opennebula** : Démon OpenNebula.

Le fichier `/var/lib/one/.one/one_auth` aura été créé avec un mot de passe généré aléatoirement. Cela devrait contenir les éléments suivants : `oneadmin : <password>`. N'hésitez pas à changer le mot de passe avant de démarrer OpenNebula comme ceci par exemple : `$ echo "oneadmin:mypassword"`

```
> /.one/one_auth
```

Pour démarrer les services opennebula, il faut exécuter la commande suivante en tant que root :

```
$systemctl start opennebula
$systemctl start opennebula-sunstone
```

Après le premier démarrage d'OpenNebula, vous devez vérifier que les commandes peuvent se connecter à OpenNebula démon. Vous pouvez le faire dans la CLI Linux ou dans l'interface utilisateur graphique : Sunstone En CLI Linux il faut exécuter la commande « `oneuser show` » comme montré sur la figure suivante :

```

$ oneuser show
USER 0 INFORMATION
ID           : 0
NAME        : oneadmin
GROUP       : oneadmin
PASSWORD    : 3bc15c8aae3e4124dd409035f32ea2fd6835efc9
AUTH_DRIVER : core
ENABLED     : Yes

USER TEMPLATE
TOKEN_PASSWORD="ec21d27e2fe4f9ed08a396cbd47b08b8e0a4ca3c"

RESOURCE USAGE & QUOTAS

```

Vous pouvez maintenant essayer de vous connecter à l'interface Web de Sunstone. Pour ce faire, pointez votre navigateur sur **http :// "frontend\_address" :9869**. Si tout va bien, vous serez accueilli avec une page de connexion. L'utilisateur est `oneadmin` et le mot de passe est celui du fichier `/var/lib/one/.one/one_auth` dans votre Front-end. Si la page ne se charge pas, assurez-vous de cocher `/var/log/one/sunstone.log` et `/var/log/one/sunstone.error`. Assurez-vous également que le port TCP 9869 est autorisé à travers le pare-feu.

**Installation et configuration MySQL** Le back-end MySQL / MariaDB est une alternative au back-end SQLite par défaut. Dans ce guide et dans le reste de la documentation et des fichiers de configuration d'OpenNebula, nous appellerons cette base de données MySQL, cependant OpenNebula vous pouvez utiliser MySQL ou MariaDB. Les deux back-ends ne peuvent pas coexister (SQLite et MySQL), et vous devrez décider lequel sera utilisé lors de la planification de votre installation OpenNebula.

Tout d'abord il faut installer un serveur MySQL fonctionnel. Après l'installation, la configuration se présente comme suit : Vous devez ajouter un nouvel utilisateur et lui accorder des privilèges sur la `opennebulabase` de données. Cette nouvelle base de données n'a pas besoin d'exister, OpenNebula la créera la première fois que vous l'exécuterez. En supposant que vous utilisiez les valeurs par défaut, connectez-vous à votre serveur MySQL et exécutez les commandes suivantes :

```
$ mysql -u racine -p
```

Entrer le mot de passe : Bienvenue sur le moniteur MySQL. [...]

```
mysql> GRANT TOUS LES PRIVILÈGES SUR opennebulabase.* À 'oneadmin' IDENTIFIÉ PAR '<le mot de passe>';
```

Requête OK, 0 ligne affectée (0,00 s) Avant d'exécuter OpenNebula pour la première fois, vous devez définir dans `oned.conf` les détails de connexion et la base de données sur laquelle vous avez accordé des privilèges.

Exemple de configuration pour MySQL :

```

DB = [ backend = "mysql" ,
server = "localhost" ,
port = 0 ,
user = "oneadmin" ,
passwd = "<thepassword>" ,
db_name = "opennebulabase" ]

```

Des champs :

- `server` : URL de la machine exécutant le serveur MySQL.
- `port` : port pour la connexion au serveur. S'il est défini sur 0, le port par défaut est utilisé.
- `user` : nom d'utilisateur MySQL. • `passwd` : mot de passe MySQL.
- `db_name` : nom de la base de données MySQL qu'OpenNebula utilisera. OpenNebula peut maintenant être utilisé avec MySQL server.

#### 4.5.0.1 Installation du Nœud KVM

Pour ajouter le référentiel OpenNebula sur Debian / Ubuntu, exécutez en tant que root :

```
$ wget -q -O- https://downloads.opennebula.org/repo/repo.key | ajout de
clé apt -
$ echo "deb https://downloads.opennebula.org/repo/5.8/Ubuntu/18.04 stable
opennebula" > /etc/apt/sources.list.d/opennebula.list
```

Exécutez les commandes suivantes pour installer le package de nœuds et redémarrez libvirt pour utiliser le fichier de configuration fourni par OpenNebula :

```
$ sudo apt-get update
$ sudo apt-get install opennebula-node
$ sudo service libvirtd restart / debian
$ sudo service libvirt-bin restart / ubuntu
```

Configurer SSH sans mot de passe OpenNebula Front-end se connecte aux hôtes de l'hyperviseur à l'aide de SSH. Vous devez distribuer la clé publique de l'oneadmin utilisateur de toutes les machines au fichier `/var/lib/one/.ssh/authorized_keys` de toutes les machines. Il existe de nombreuses méthodes pour réaliser la distribution des clés SSH, l'administrateur doit finalement choisir une méthode (la recommandation est d'utiliser un système de gestion de configuration). Dans ce guide, nous allons scp manuellement les clés SSH. Lorsque le package a été installé dans le Front-end, une clé SSH a été générée et le `authorized_keys` fichier. Nous synchroniserons le `id_rsa`, `id_rsa.pub` et `authorized_keys` du Front-end vers les nœuds. De plus, nous devons créer un `known_hosts` fichier et le synchroniser également avec les nœuds. Pour créer le `known_hosts` fichier, nous devons exécuter cette commande en tant qu'utilisateur oneadmin dans le Front-end avec tous les noms de nœuds et le nom du Front-end comme paramètres :

```
$ ssh-keyscan <frontend> <node1> <node2> <node3> ... »
/var/lib/one/.ssh/known_hosts
```

Nous devons maintenant copier le répertoire `/var/lib/one/.ssh` sur tous les nœuds. Le moyen le plus simple consiste à définir un mot de passe temporaire oneadmin dans tous les hôtes et à copier le répertoire à partir du front-end :

```
$ scp -rp /var/lib/one/.ssh <node1>: / var / lib / one /
$ scp -rp /var/lib/one/.ssh <node2>: / var / lib / one /
$ scp -rp / var / lib / one / .ssh <noeud3>: / var / lib / one /
```

[29]

Vous devez vérifier que la connexion à partir du Front-end, en tant qu'utilisateur oneadmin, aux nœuds et au Front-end lui-même, et des nœuds au Front-end, ne demande pas de mot de passe :