

ECOLE MAROCAINE DES SCIENCES DE L'INGÉNIEUR

5/IIR/2020

PROJET DE FIN D'ETUDES

THEME

**Etude et implémentation d'une solution
service mesh dans un contexte e-commerce**

FILIÈRE

Ingénierie Informatique et Réseaux (IIR)

OPTION

Méthodes Informatiques Appliquées à la Gestion des Entreprises (MIAGE)

RÉALISÉ PAR

Alalou Yassine

ENCADRÉ PAR

Mr Ali Id Mansour

Mr Nabil Moursli

Soutenu le : 08/09/2020

Année universitaire : 2019/2020

Dédicace

Je dédie ce travail, comme preuve de respect, de gratitude, et de reconnaissance à :

Ma famille avec tous les sentiments de respect d'amour pour tous les sacrifices déployés pour m'élever dignement et assurer mon éducation dans les meilleures conditions.

À mon frère et ma famille, je ne sais comment vous remercier pour tout ce que vous avez fait pour moi. Je vous souhaite un bon avenir et une vie pleine de bonheur.

Mes collègues et mes encadrants pour leur aide, leur temps, leur encouragement, leur assistance et soutien. A tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Nous espérons que ce rapport va satisfaire toute personne qui aura l'occasion de le lire.

Remerciement

Il m'est agréable de m'acquitter d'une dette de reconnaissance auprès de toutes les personnes, dont l'intervention au cours de ce projet, a favorisé son aboutissement.

*J'exprime ma profonde gratitude et mes sincères remerciements à mes encadrants au sein de SQLI monsieur **Zbair Lahcen**, monsieur **Moursli Nabil** et monsieur **Mrani Jaâfar**, ainsi que monsieur **Ali ID MANSOUR**, mon encadrant à l'EMSI Marrakech, pour leurs orientations et précieux conseils qui m'ont été d'un appui considérable dans ma démarche.*

Je tiens à remercier aussi tous les membres du jury qui m'ont fait l'honneur d'accepter de juger mon travail. Je tiens également à passer un salut spécial à l'ensemble du corps enseignant de l'EMSI Marrakech, pour avoir porté un vif intérêt à notre formation, et pour nous avoir accordé de leur temps, de leur attention et de leur énergie et ce dans un cadre agréable de complicité et de respect.

Enfin, que tous ceux et celles qui ont contribué de près ou de loin à l'accomplissement de ce travail trouvent l'expression de mes remerciements et mes considérations.

Abstract

As part of the end of study project to obtain my degree in computer Engineering and networks from the Moroccan School of Engineering Sciences.

I completed a six-month internship in Sqli Rabat Whose main objective is study and implementation of a service mesh in e-commerce context. The PFE project is divided into three phases:

- **Phase 1: Collect information and explore:** Documentation about the service mesh, explore different solutions, and implement demos to illustrate the advantages and disadvantages of each solution.
- **Phase 2: Comparative study:** Select the services mesh candidates. Define the criteria of comparison. The final choice of the technology.
- **Phase 3: Realization and installation:** Discovery and analyze the architecture and the e-commerce platform. Realization of implementations and required configuration. Deployment and test of the solution.

Through this document, I will describe in more detail each part of the realization of this phases.

Key Words: SQLI, Microservice, Service mesh, Kubernetes, Deployment, Benchmarking, E-commerce.

Résumé

Dans le cadre du projet de fin d'étude pour l'obtention de mon diplôme d'ingénieur en informatique et réseaux de L'Ecole Marocaine des Sciences de L'Ingénieur.

J'ai effectué un stage d'une durée de six mois au sein de SQLI Rabat dont l'objectif principal est étude et implémentation d'un service mesh (Maillage de services) dans un contexte e-commerce. Le projet PFE est scindé en trois phases:

- **Phase 1 : Collecte d'information et exploration :** Documentation guidée sur les services mesh. Exploration des différentes solutions, et implémenter des demos pour illustrer les avantages et inconvénients de chaque solution.
- **Phase 2 : Etude comparative :** Identification des service mesh candidates. Définition critères de comparaison et de choix. Le choix final de la technologie.
- **Phase 3 : Réalisation et mise en place :** Découverte et analyse de l'architecture de plateforme e-commerce. Réalisation des implémentations et configurations requises. Déploiement et tests de la solution.

A travers ce document, je vais décrire plus en détail chaque partie de la réalisation de ces phases.

Mots clés: SQLI, Microservice, Maillage de services, Kubernetes, Déploiement, Benckmarking, E-commerce.

Glossaire

AKS	Azure Kubernetes Service
API	Application Programming Interface
B2B	Business to Business
B2C	Business to Consumer
DDD	Domain-Driven Design
EDA	Event-Driven Architecture
gRPC	gRPC Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
ISC	Innovative Service Center
JPA	Java Persistence API
JSON	JavaScript Object Notation
JWKS	Json Web Key Set
JWT	Json Web Token
mTLS	Mutual Transport Layer Security
MVC	Model-View-Controller
REST	Representational State Transfer
SOA	Service Oriented Architecture
SQL	Structured Query Language
SSII	Société de Service d'Ingénierie Informatique
TCP	Transmission Control Protocol

Table des figures

Figure 1 Logo SQLI.....	4
Figure 2 Fiche signalétique	4
Figure 3 SQLI Organigramme	5
Figure 4 Métiers de SQLI	7
Figure 5 Chiffres clés.....	9
Figure 6 Client de SQLI.....	9
Figure 7 Partenaires de SQLI.....	10
Figure 8 Formation professionnelle.....	11
Figure 9 Le planning.....	14
Figure 10 Processus Scrum	15
Figure 11 Diagramme de Gantt	16
Figure 12 Timeline des modèles d'architecture logiciels	18
Figure 13 Architecture microservices.....	19
Figure 14 Les outils de Netflix.....	20
Figure 15 The Library architecture.....	24
Figure 16 Node Agent Architecture	24
Figure 17 L'architecture Sidecar.....	25
Figure 18 Istio logo.....	27
Figure 19 Linkerd logo	27
Figure 20 Consul logo.....	27
Figure 21 Répartition du trafics.....	31
Figure 22 retry et timeout dans Linkerd.....	32
Figure 23 retry et timeout dans Istio.....	32
Figure 24 Configuration d'un circuit breaker	33
Figure 25 Résultat de génération 400 connections pour l'application sans service Mesh.....	36
Figure 26 Résultat de génération 400 connections pour l'application injecté par Istio.....	37
Figure 27 Résultat de génération 400 connections pour l'application injecté par Consul.....	37
Figure 28 Résultat de génération 400 connections pour l'application injecté par Linkerd	37
Figure 29 Résultat de génération 1000 connections pour l'application sans service Mesh.....	38
Figure 30 Résultat de génération 1000 connections pour l'application injecté par Istio.....	38
Figure 31 Résultat de génération 1000 connections pour l'application injecté par Consul.....	39
Figure 32 Résultat de génération 1000 connections pour l'application injecté par Linkerd.....	39

Figure 33 Résultat des requêtes par seconds.....	40
Figure 34 Résultat de la latence et charge de l'application	40
Figure 35 L'architecture de l'application.....	48
Figure 36 UML Logo.....	51
Figure 37 diagramme de cas d'utilisation.....	52
Figure 38 Les deux types de diagramme de séquences	53
Figure 39 Diagramme de séquence du cas « Authentification »	54
Figure 40 Diagramme de séquence du cas « Ajouter un produit au panier »	55
Figure 41 Diagramme de séquence du cas « checkout »	56
Figure 42 Diagramme de package du microservice Users-service	57
Figure 43 Diagramme de packages du Microservice Checkout	58
Figure 44 Diagramme de class du Microservice ProductCatalog.....	59
Figure 45 Diagramme de class du Microservice Cart.....	59
Figure 46 Swagger logo	60
Figure 47 La documentation swagger du Microservice Users	60
Figure 48 La documentation Swagger du Microservice ProductsCatalogue	61
Figure 49 La documentation Swagger du Microservice Reviews	62
Figure 50 Dockerfile.....	66
Figure 51 Les espaces de nom du cluster Kubernetes	67
Figure 52 Déploiement de Microservice product-catalog	68
Figure 53 déploiement du productCatalog dans AKS	69
Figure 54 L'injection du service mesh Istio	70
Figure 55 API Gateway	71
Figure 56 Résultat de l'API Gateway et l'Host	71
Figure 57 Reviews version 1	72
Figure 58 Reviews version 2.....	72
Figure 59 Retry & Timeout configuration	73
Figure 60 La sécurité d'Istio	74
Figure 61 Kiali dashboard, mTLS est activé	75
Figure 62 envoi d'une requête sans token dans leur entête	76
Figure 63 L'envoi d'une requête avec un token valide	76
Figure 64 L'envoi d'une requête avec un token non valide	77
Figure 65 Kiali dashboard.....	78
Figure 66 Reviews visualisation de Kiali.....	78
Figure 67 Grafana : Graphe de nombre d'octet échangé via TCP connexion avec le service 'Review'	79

Figure 68 Grafana : graphe des ressources consommées par le total des proxy 80

Liste des tableaux

Tableau 1 Les solutions service mesh existantes.....	26
Tableau 3 Critères demandes par le client	41

Table des matières

Introduction générale	1
Chapitre 1 Contexte générale du projet.....	3
1.1 Introduction.....	4
1.2 Présentation de l'organisme d'accueil	4
1.2.1 Carte d'identité.....	4
1.2.2 L'organigramme.....	5
1.2.3 Métiers de SQLI.....	6
1.2.4 Structure du groupe.....	7
1.2.5 Les chiffres clés	8
1.2.6 Clients de SQLI.....	9
1.2.7 Partenaires de SQLI	10
1.3 Présentation du stage.....	10
1.3.1 Formation professionnelle	10
1.3.2 Plan d'intégration	11
1.3.3 Contexte générale du projet	12
1.3.4 Conduite du projet.....	14
1.4 Conclusion	16
Chapitre 2 Exploration et benchmark des services mesh.....	17
2.1 Introduction.....	18
2.2 Collecte d'information et d'exploration.....	18
2.2.1 Introduction au service mesh	18
2.2.2 Les solutions existantes et les solutions candidates	26
2.2.3 Présentation des solutions candidates	26
2.3 Etude comparative	28
2.3.1 Critères de comparaison.....	28
2.3.2 Comparaison des services mesh	30
2.3.3 Le choix de la solution	41
2.4 Conclusion	42

Chapitre 3	Spécification des besoins et l'architecture proposée	43
3.1	Introduction	44
3.2	Spécification des besoins globaux	44
3.2.1	Besoins fonctionnels	44
3.2.2	Besoins non fonctionnels	46
3.2.3	Division en Microservices	47
3.3	Conclusion	49
Chapitre 4	Conception et la documentation de l'API	50
4.1	Introduction	51
4.2	Conception	51
4.2.1	Le Langage de Modélisation UML	51
4.2.2	Diagramme de cas d'utilisation	51
4.2.3	Diagramme de séquence	52
4.2.4	Diagramme de package	56
4.2.5	Diagramme de classes	58
4.3	La documentation de l'API	59
4.3.1	Swagger	59
4.3.2	La documentation swagger	60
4.4	Conclusion	62
Chapitre 5	L'implémentation	63
5.1	Introduction	64
5.2	Technologies et outils utilisés	64
5.2.1	Outil de collaboration GitLab	64
5.2.2	Environnement de déploiement	64
5.2.3	Technologie de développement	65
5.3	Le déploiement des microservices	66
5.3.1	La containerisation	66
5.3.2	Structure et organisation	67
5.3.3	Le déploiement dans Kubernetes	68
5.3.4	Management du trafic	70
5.3.5	Sécurité	73
5.3.6	Observabilité	77

5.4 Conclusion	80
Conclusion générale	81
Références	83

Introduction générale

Afin de valider son parcours d'ingénierie au sein de l'EMSI, chaque étudiant doit effectuer un stage de fin d'étude. Ce stage est une étape importante pour un futur ingénieur, non seulement du point de vue de la scolarité, mais aussi d'un point de vue personnel. Ce stage est en effet nécessaire à la mise en pratique de l'enseignement et la formation reçu à l'EMSI Marrakech.

Les sites e-commerce prennent une part de plus en plus importante dans le chiffre d'affaires des sociétés bien positionnées sur internet. En effet, beaucoup de sociétés pensent que la vente sur internet se développe très rapidement dans le monde. Ce rythme soutenu pousse les entreprises à s'armer de toutes les nouveautés technologiques à forte valeur ajoutée et à être le plus concurrentiel possible sur ce marché.

En effet, plus de dix ans de développement continu des processus métier, le projet client fait face à certains problèmes tels que : la difficulté de maintenabilité, d'extensibilité et les retards de mise en production.

À cet égard, l'entreprise cliente a réfléchi de migrer l'architecture de l'application vers une autre plus moderne et qui va résoudre ces problèmes et d'utiliser un service Mesh pour sécuriser la communication et manager le trafic.

C'est dans ce cadre, que s'inscrit notre projet de fin d'études, notre tâche consiste de faire un Benchmarking des solutions service Mesh existantes, le choix d'une solution et l'implémenter dans une application e-commerce Microservice avant qu'il soit utilisé dans le projet client.

Le présent rapport trace le déroulement de ce travail, il est structuré en cinq chapitres, Le premier chapitre présente le contexte générale du projet : L'organisme accueillante ainsi que la formation offerte par l'entreprise et le contexte générale du projet. Le deuxième chapitre projette la phase de l'exploration et le benchmarking des différentes solutions des services mesh, les critères de comparaison adaptés aux besoins du client ainsi que le choix final de la solution. Le troisième chapitre décrit la spécification des besoins fonctionnels et non fonctionnels de notre projet, il présenté aussi l'architecture des microservices proposée. En suite le quatrième chapitre décrit illustre la conception et la documentation swagger de notre API. Et finalement le cinquième chapitre qui présente la phase

de l'implémentation qui décrit les configurations de déploiement de notre application et l'injection du service Mesh.

Chapitre 1

Contexte Générale du projet

1.1 Introduction

L'objectif de ce chapitre est de présenter le lieu du stage sa spécialité, ses services ainsi que son organisation. En suite la formation offerte par l'entreprise. Dans ce chapitre je présente aussi le contexte général du projet de mon stage, les objectifs à atteindre et aussi la définition du cahier des charges.

1.2 Présentation de l'organisme d'accueil

1.2.1 Carte d'identité



Figure 1 Logo SQLI

Créé en 1990, SQLI est un groupe européen de services dédié au monde du Digital, spécialisé dans la conception, la mise en œuvre, le déploiement mondial et l'exploitation de dispositifs omnicanal. Son positionnement de spécialiste du commerce et des technologies permet à ses équipes d'experts d'accompagner durablement les grandes entreprises et marques européennes dans le développement de leurs ventes et de leur notoriété ainsi que leur performance interne en réinventant l'expérience client, partenaire et collaborateur. [1]

Raison sociale	SQLI
Forme juridique	Société anonyme à directoire
Date de création	1990
Fondateurs	Alain Lefebvre
Siège social	166 rue Jules Guesde - 92300 Levallois Perret
Slogan	Digital Performance
Activité	Entreprise de services du numérique

Figure 2 Fiche signalétique

1.2.2 L'organigramme

Les différents centres de Rabat et Oujda étaient indépendants mais aujourd'hui avec la nouvelle organisation les deux sites ont mergé en donnant naissance à une seule entité « ISC Maroc ». La figure 2 montre l'organigramme de l'entreprise SQLI.

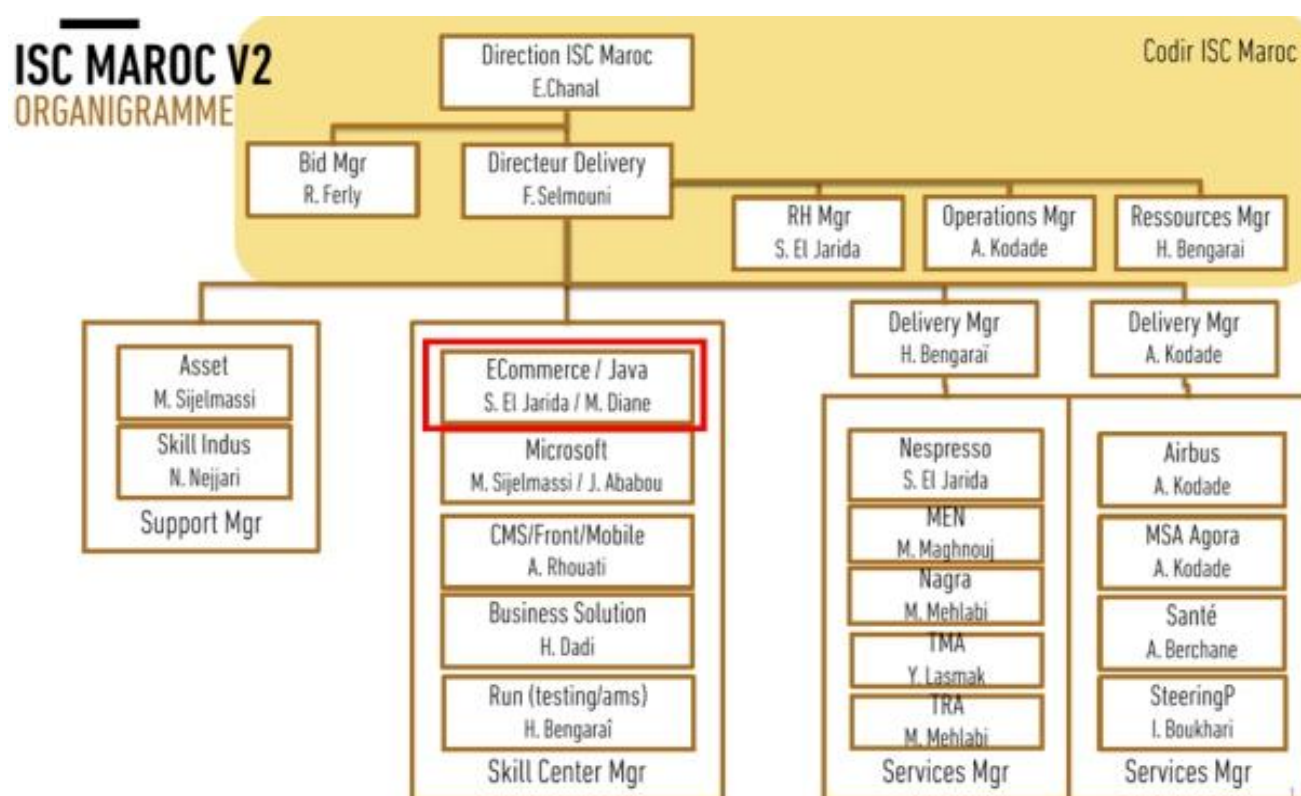


Figure 3 SQLI Organigramme

Éric CHANAL assure la Direction globale de l'ISC Maroc en se fondant sur Fouad SELMOUNI pour les livraisons et Renaud FERLY pour la détection des futurs projets et la bonne coordination des avant-ventes.

Les deux sites sont sous la responsabilité de Fouad SELMOUNI. Fouad se concentrera sur la livraison globale afin de garantir que les produits soient de haute qualité pour les clients disposant des moyens de production appropriés.

Pour cela, il aura sous sa responsabilité :

- Saïd EL JARIDA, Responsable RH sur les deux sites. Il assurera le bon fonctionnement des processus RH et proposera des actions visant à améliorer les conditions de travail et à développer l'expertise des collaborateurs.
- Abderrahmane KODALE, responsable des opérations sur les deux sites. Son rôle est d'anticiper les résultats des activités pour les mois à venir afin de prendre les meilleures décisions au plus vite et d'être prévisible dans les prévisions du groupe.
- Hicham BENGARAI devient Resource Manager sur les deux sites. Il sera informé de tous les besoins de recrutement et optimisera les affectations des projets afin d'utiliser au mieux les ressources des deux sites.

1.2.3 Métiers de SQLI

SQLI s'est spécialisée dans les projets e-commerce, liés aux systèmes d'informations intégrant l'utilisation des technologies internet. Afin que ses collaborateurs puissent bénéficier de ses services, SQLI déploie sur intranet plusieurs outils.

Pour aider les entreprises à tirer parti des technologies internet, SQLI propose un accompagnement global sur tout le cycle du projet :

- Des prestations de conseil pour aider les clients à faire les bons choix.
- La mise en œuvre concrète de ces choix par la réalisation et l'intégration.
- Un accompagnement dans le déploiement des projets et le transfert de compétences.

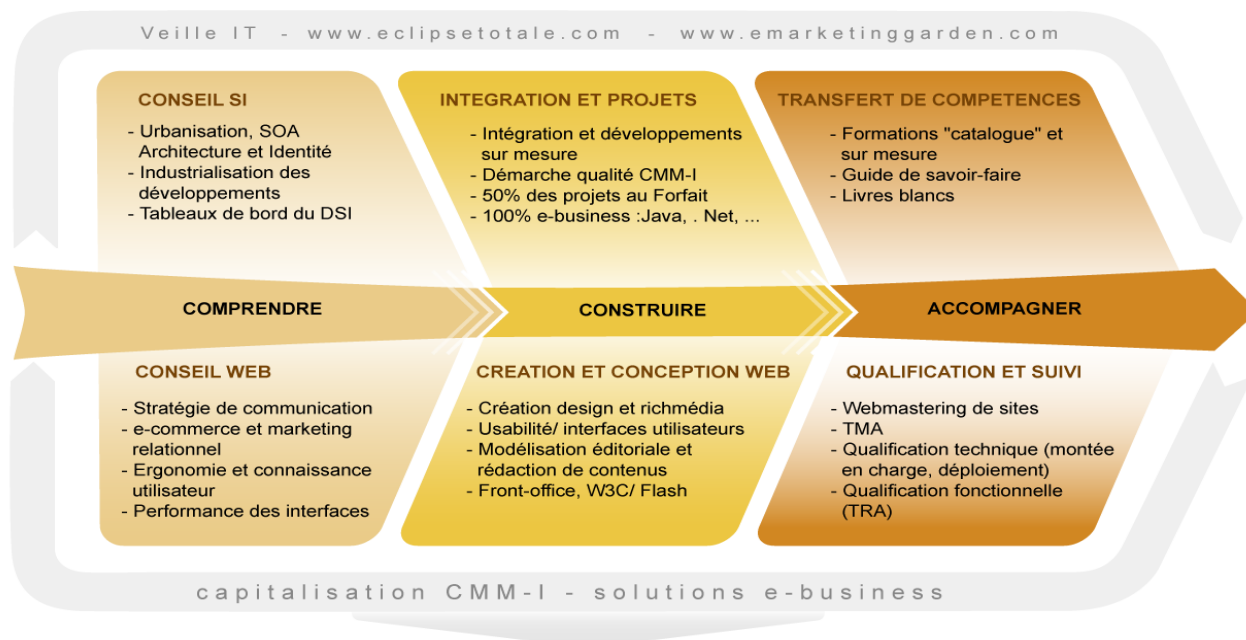


Figure 4 Métiers de SQLI

Le groupe SQLI fédère toutes les compétences indispensables au bon déroulement des projets de ses clients, du conseil à la réalisation en passant par l'ergonomie, le design, l'interface utilisateur et la formation.

Ainsi le métier du groupe SQLI est organisé en quatre pôles :

- Le pôle « Stratégie en systèmes d'informations ».
- Le pôle « Ingénierie et intégration ».
- Le pôle « Conception Web ».
- Le pôle « Formation et transfert de compétences ».

1.2.4 Structure du groupe

Le groupe fédère toutes les compétences indispensables au bon déroulement des projets de ses clients, du conseil à la réalisation en passant par l'ergonomie, le design, l'interface utilisateur et la formation.

L'agence SQLI Rabat, dans laquelle j'ai effectué mon stage de fin d'études, est composée de plusieurs équipes travaillantes sur des technologies différentes :

- L'équipe Microsoft : ASP.NET, Windows forms, Silverlight, SharePoint.
- L'équipe Open source.
- L'équipe FLEX
- L'équipe SAP
- L'équipe Agency: Contient les projets internes à l'entreprise.
- L'équipe E-Commerce (Java) : Front office, Back office.

Chaque équipe est composée de plusieurs groupes de travail chacun travaillant sur un projet. Un groupe de travail est composé d'habitude de :

- Directeur de projets : Gère tous les aspects du projet.
- Un chef de projet (Scrum Master) : Gère un des aspects du projet qui est le cycle de développement logiciel.
- Architectes
- Un expert technique
- Un Business analyste
- Des développeurs (Team)
- Testeurs

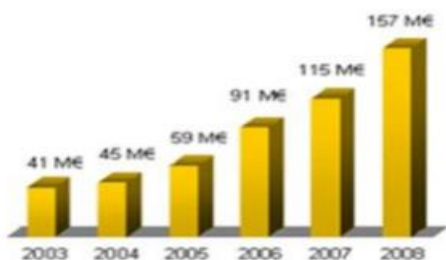
1.2.5 Les chiffres clés

Les chiffres clés suivants présentent la situation actuelle de SQLI :

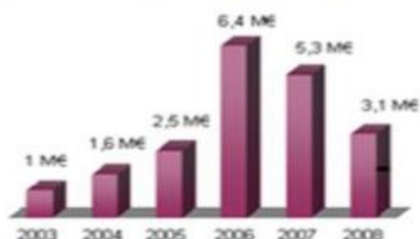
30 ans d'expérience et d'innovation : SQLI place son développement sur une expertise technologique de pointe et sur sa politique intense de veille. En 2005, SQLI devient la première SSII française à obtenir la certification CMMI niveau 3.

Plus de 2400 collaborateurs au groupe SQLI : SQLI est déjà présente au Benelux (Belgique Nederland Luxembourg), en Espagne, au Maroc et à Singapour en plus de sa présence en France.

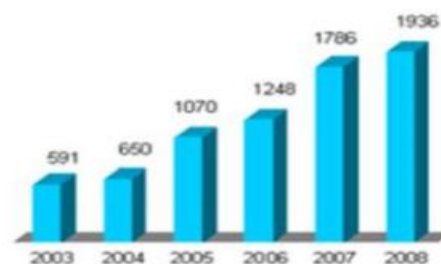
239 M€ de chiffre d'affaires en 2019.



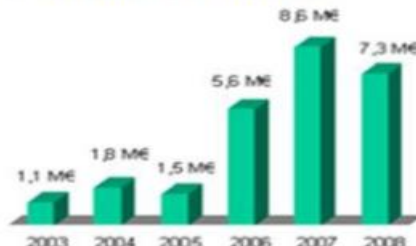
Evolution du Chiffre d'Affaires



Résultat net



Evolution des effectifs



Résultat opérationnel

Figure 5 Chiffres clés

1.2.6 Clients de SQLI

Dans une logique de développement de son activité, SQLI dispose d'un portefeuille riche composé d'une clientèle assez diversifiée, et opère dans différents secteurs d'activité, pour élargir le périmètre de son intervention et éviter le risque de concentration sur une catégorie limitée de clients. Plus de 1200 clients, opérant dans différents secteurs d'activités, dont des grands comptes et des PME, forment le portefeuille de SQLI.

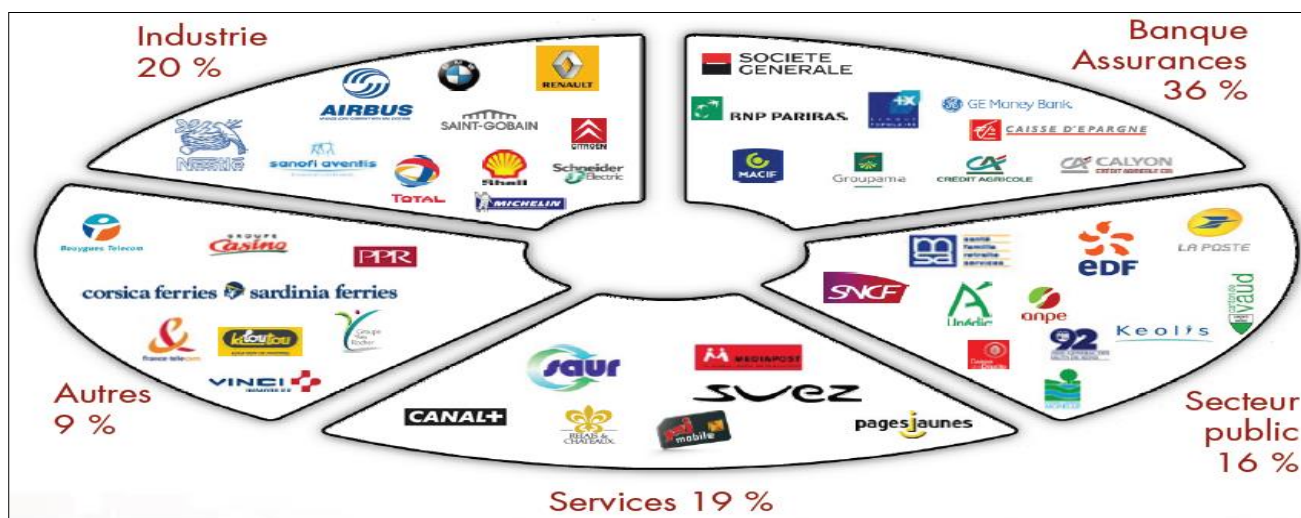


Figure 6 Client de SQLI [2]

1.2.7 Partenaires de SQLI

SQLI a réussi à se faire une place sur le marché grâce à ses partenariats efficaces avec de nombreux acteurs de référence en matière de technologie et du logiciel, mais aussi avec des acteurs débutants qui ont fait preuve de mérite, en présentant des produits et des solutions logiciel de grande valeur ajoutée pour les clients.



Figure 7 Partenaires de SQLI [3]

1.3 Présentation du stage

1.3.1 Formation professionnelle

SQLI insiste sur la qualité des livrables et les bonnes pratiques du codage, pour cela et avant l'intégration d'un projet, les stagiaires doivent passer une formation d'un mois. Durant cette formation, 10 jours était dédié au SAP Hybris, parce que la plupart des projets au sein d'SQLI sont réalisés par cette plateforme e-commerce. En plus pour l'installation d'un environnement de développement.

Le tableau ci-dessous résume les outils et technologies étudié, et le temps associer à chaque thème :

Thème	Description	Durée (en jours)
Séminaire d'intégration	Séminaire d'intégration	0.25
Environnement technique de Dév	IntelliJ (Checkstyle, Formattage, Racourcis...) Git Maven/Ans, Jenkins, Bitbucket...etc	1.75
Java / Clean Code /Design Pattern	JAVA (principe OO, convention nommage, ...) SOLID, SMELL, Immutability Design Pattern: Singleton, factory...etc TU, TI, Mock...etc	2
	Clean Code, Loi de Demeter Java efficace	2
Spring	IOC MVC ...	1
Web service, Microservice, et API	Web service, Microservice, et API: SOA, REST API, Swagger, Architecture Microservice, authentification OAuth 2.0 ...etc	1
Hybris Core Data model, extension, cronjob, impex, service layer, update sys , init sys, Product Modeling (Classification, Product Variants ...), Smart Edit, Interceptors, Validation, Restrictions	Hybris core + Hybris 123	5
Hybris commerce Solr, Order OMS, accelertor, Promotion Engine, CronJob Scripting, Cart & Checkout and Order, Management, Warehouse Integration, Promotions & Coupons, Intrduction à SAP Commerce Cloud, Spartacus	Pres + commerce trails	5
Devops	Introduction à l'univers du Devops: Bonnes pratiques CI/CD Outils (Docker, Puppet ...etc)	0.5
Skills	Formation officielle Skills: Orga Méthodologie ...	0.5
Agile	Jira Confluence Rituels scrum: SUM, spring planning...etc. Rigueur administrative, comment écrire un commentaire Jira constructif (observation, analyse, next step)	0.5

Figure 8 Formation professionnelle

1.3.2 Plan d'intégration

L'entreprise SQLI comme la plupart des SSII est divisée en plusieurs équipes projets, le chef du projet permet d'assurer les fonctions RH quotidiennes et le Scrum Master s'occupe du respect du processus.

Après avoir terminé la formation, j'ai été sélectionné pour intégrer le projet phare de la société, un projet d'un client suisse de production des machines de cafés. Le projet est l'un des projets clés de l'entreprise SQLI, il est certes son plus grand au Maroc, sa forte valeur ajoutée et sa grande rentabilité

le rend un projet critique et impose que tous les membres de l'équipes soient dans un bon niveau de connaissances et d'engagement.

Le projet est organisé en plusieurs équipe, chaque équipe a des objectifs prédéfinis et travaille de manière quasi-indépendante.

Un coordinateur de chaque équipe assure le bon déroulement des procédures, un responsable de l'environnement s'occupe de la maintenance des environnements dédiés à chaque équipe et enfin les développeurs et les testeurs travaillent conjointement pour réaliser les différentes tâches.

Les équipes du projet en question, situés à SQLI Rabat sont les suivantes :

- **Equipe Atlas** : Responsable des nouveaux développements.
- **Equipe Abtal** : Responsable des nouveaux développements.
- **Equipe Menara** : Responsable des nouveaux développements.
- **Equipe Release Team** : Responsable de la stabilisation des nouveaux développements avant du déploiement vers la production.
- **Equipe Emergency Room** : Responsable du suivie et de la correction des anomalies qui sont détectés en production.

1.3.3 Contexte générale du projet

1.3.3.1 *Le cadre du projet*

Notre projet intitulé « Étude et implémentation d'une solution Service Mesh, dans un contexte e-commerce » a été proposé dans le cadre de l'élaboration d'un stage de fin d'études au sien de la société SQLI pour répondre au besoin du client.

1.3.3.2 *La problématique*

Le projet client est âgé plus de dix ans, Il a commencé à investir dans le commerce électronique pour vendre ses produits.

Ainsi pour réduire le "Time To Market" et acquérir le plus possible des clients, notre client a adopté la stratégie e-commerce multicanale pour vendre ces produits dans différents canaux, à savoir des points de vente autour du monde, des sites web e-commerce, des applications mobiles ou bien des Boutiques Apps.

Ce projet a devenu un très grand projet. Et parmi les conséquences d'une application de grande taille est qu'une mise à jour peut prendre plusieurs mois pour le remettre en production.

L'adoption d'une architecture micro-service pour résoudre notre problème n'est pas facile. Dans ce cas, on remplace une problématique avec une autre. Dans une architecture micro-service, il peut y avoir une centaine de services qui se communiquent. Vous devez vous assurer qu'un service défaillant ne fait pas tomber l'ensemble de l'architecture, et la mise à jour et la maintenance d'un service ne mène pas vers la mise à jour d'un autre service ou bien de l'ensemble de l'application.

Alors la gestion de la communication de service à service dans une architecture de micro-service est un défi, car ils doivent communiquer entre eux sur un réseau bien sécurisé et facile à maintenir.

1.3.3.3 L'objectif du projet

Le but général de notre stage est de faire une étude globale des solutions service mesh proposées, cela permettra par la suite de bien choisir la meilleure solution; la plus adaptable et qui répond aux besoins du projet client, et enfin implémenter cette solution dans un contexte e-commerce pour bien maîtriser le domaine.

1.3.3.4 Description du sujet de stage

Titre : Etude et implémentation d'une solution Service mesh dans un contexte e-commerce

Description :

- Étude comparative des différentes solutions Service Mesh.
- Choisir, selon des critères bien défini, la solutions adaptées aux besoins.
- Intégration et déploiement de la solution choisie au sein d'une plateforme e-commerce.

Macro planning :

Phase	Taches	Livrables
Phase 1: Collecte d'information et exploration	Documentation guidée sur les services <u>Mesh</u> Exploration des différentes solutions dans l'industrie	Documents et implémentation pour illustrer les avantages / inconvénients / fonctionnalités / limitations de chaque Service <u>Mesh</u>
Raffinement du scope selon nos besoins projets		
Phase 2 : Etude Comparative	Identification des services <u>Mesh</u> candidates Définition des critères de l'évaluation et de choix. <u>Benchmark</u> des différentes technologies	Documents de l'étude comparative. <u>POCs</u> pour appuyer les résultats du <u>Benchmark</u> Décision finale pour la technologie à adopter
Raffinement du scope selon nos besoins projets		
Phase 3 : Réalisation et Mise en place	Découverte et Analyse de l'architecture de la plateforme e-commerce. Réalisation des implémentations et configurations requises Déploiement et tests de la solution	Document Techniques et d'implémentations Intégration de bout en bout de la solution finale Rapport global du stage

Figure 9 Le planning

1.3.4 Conduite du projet

1.3.4.1 La méthode scrum

Scrum est un cadre de travail permettant de répondre à des problèmes complexes et changeants, tout en livrant de manière productive et créative des produits de la plus grande valeur possible.

Scrum se base sur la théorie du contrôle empirique de processus, ou l'empirisme. L'empirisme soutient que les connaissances proviennent de l'expérience et d'une prise de décision basée sur des faits connus. Scrum utilise une approche itérative et incrémentale pour optimiser la prédictibilité et pour contrôler le risque. Trois piliers soutiennent l'implémentation d'un contrôle empirique de processus : la transparence, l'inspection et l'adaptation.

L'Équipe Scrum comprend un propriétaire de produit (Product Owner), une Équipe de développement (Development Team) et un Scrum Master. [4]

- **Le Product Owen :**

Le Product Owner est responsable de maximiser la valeur du produit et du travail de l'Équipe de Développement. La façon de jouer ce rôle peut varier grandement selon les entreprises, les Équipes Scrum et les individus.

- **Le Scrum Master :**

Le Scrum Master est responsable de s'assurer que Scrum est compris et mis en œuvre. Les Scrum Masters remplissent leur rôle en s'assurant que l'Équipe Scrum adhère à la théorie, aux pratiques et aux règles de Scrum.

- **L'Équipe de Développement :**

L'Équipe de Développement livrent à chaque Sprint un incrément « terminé » et potentiellement livrable du produit. Seuls les membres de l'Équipe de développement créent l'incrément. Les équipes de développement sont structurées et habilitées par l'entreprise à organiser et gérer leur propre travail.

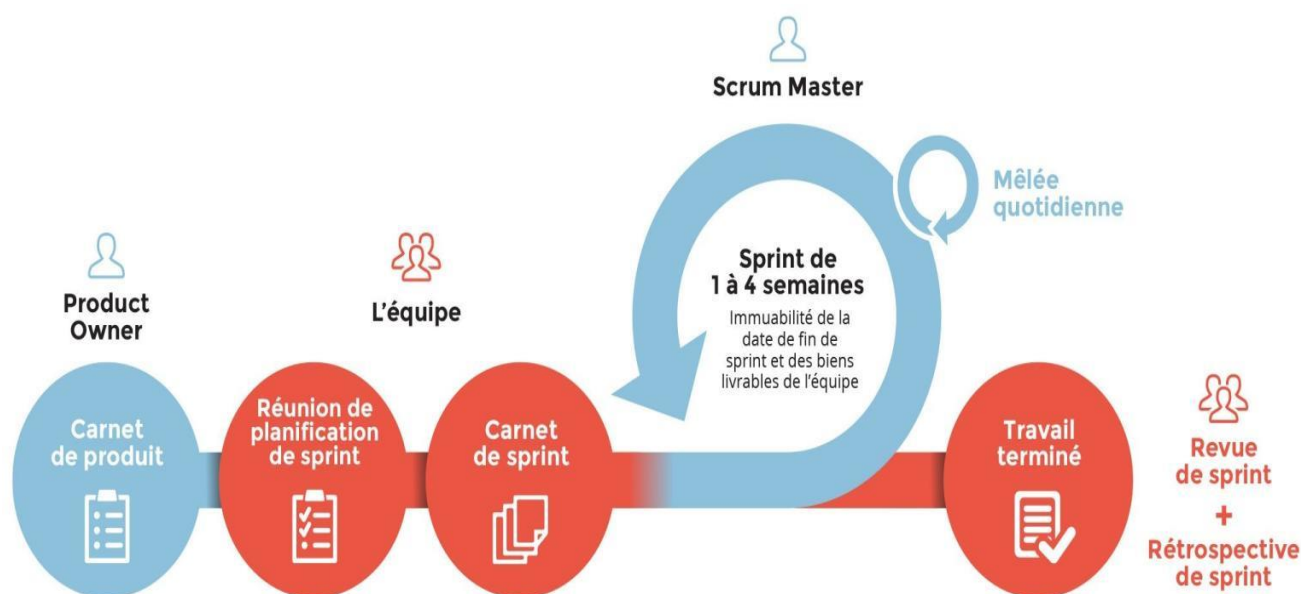


Figure 10 Processus Scrum

1.3.4.2 Planification du projet

Le projet n'a pas une planification exacte, on peut dire qu'il est agile.

Chaque semaine, on organise une réunion avec la présence des architectes SQLI et des membres du team de la part de client. Pour voir l'avancement des sprints, et de discuter les résultats achevés, pour définir les actions à suivre.

Ce diagramme de Gantt montre la planification de chaque partie de nos 3 phases a pris du temps :

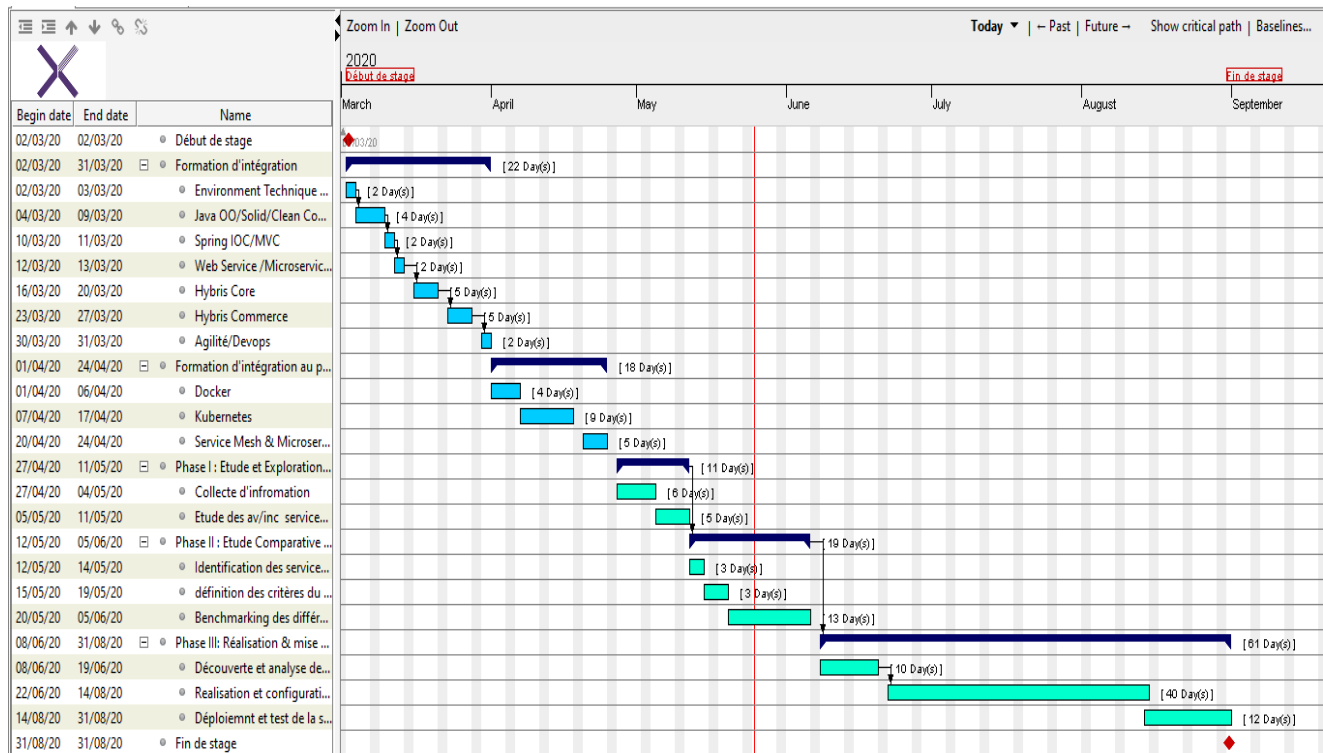


Figure 11 Diagramme de Gantt

1.4 Conclusion

Le chapitre a été dédié à présenter l'organisme d'accueil, le plan d'intégration ainsi que la formation que nous avons suivi durant le premier mois de la période du stage. Ce premier chapitre a entamé aussi une première description du projet, la problématique et la solution proposée afin d'introduire son contexte général et aussi la description du sujet de stage et finalement la planification du projet.

Chapitre 2

Exploration et benchmark des services mesh

2.1 Introduction

Durant ce chapitre je vais entamer la partie de l'exploration c'est-à-dire c'est quoi un service mesh et son rôle dans une architecture micro services. Après le chapitre représente le benchmarking ou une étude comparative des différents solutions service mesh. dans le but d'avoir une vision globale claire des fonctionnalités, limitations, avantages, inconvénients de chacun d'entre eux ; pour enfin choisir la solution la plus adéquate au besoin du projet client.

2.2 Collecte d'information et d'exploration

2.2.1 Introduction au service mesh

2.2.1.1 Les microservices

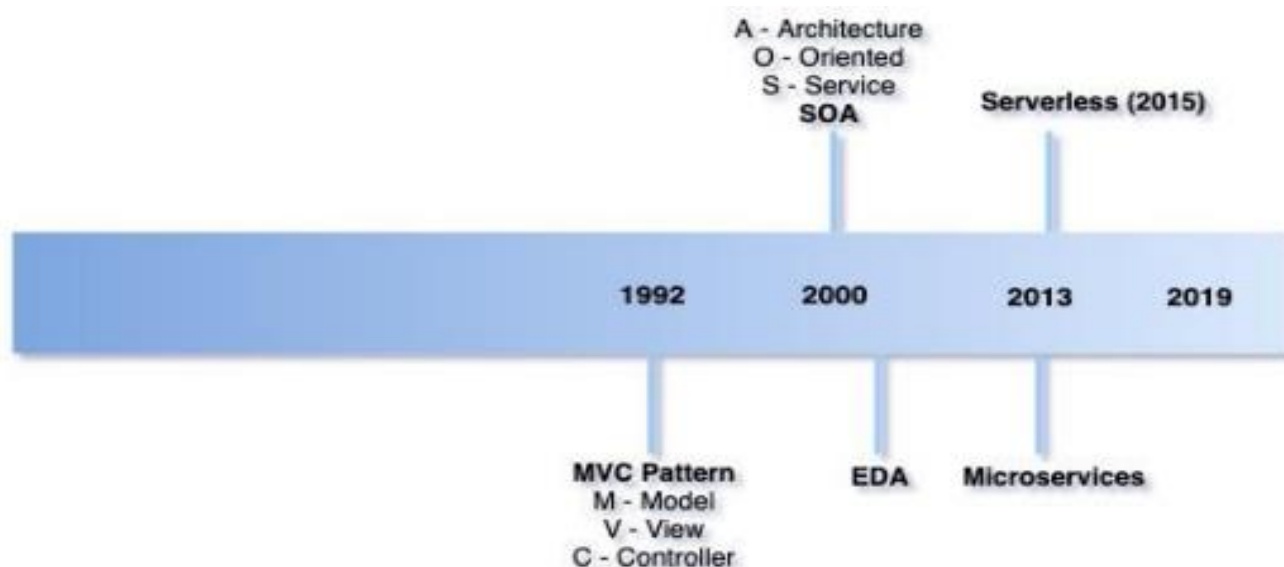


Figure 12 Timeline des modèles d'architecture logiciels

Il existe différents modèles d'architectures dans la timeline. Le modèle MVC, un des modèles les plus utilisés, SOA et EDA ont été popularisés respectivement en 2000 et 2003 ; les choses ont commencé à être distribuées. Nous avons commencé à ce moment-là à diviser les choses en petits logiciels, pour plusieurs raisons. Et enfin, Microservices et Serverless, en 2013 et 2015 respectivement. [5]

Cependant les microservices sont des services faiblement couplés avec des contextes limités qui vous permettent de développer, déployer et faire évoluer vos services indépendamment. Il peut

également être défini comme un modèle architectural pour construire des systèmes distribués qui sont développés et déployés indépendamment. La gestion de la communication de service à service dans une architecture de microservices est difficile, car ils doivent communiquer entre eux sur un réseau peu fiable.

- **Complexité des architectures microservices**

Il y a un certain nombre de composants mobiles dans une architecture de microservices, donc elle a plus de points de pannes. Les échecs peuvent être causés par diverses raisons : erreurs et exceptions dans le code, publication de nouveau code, mauvais déploiements, défaillances matérielles, défaillance du centre de données, mauvaise architecture, manque de tests unitaires, communication sur le réseau non fiable, services dépendants, etc.

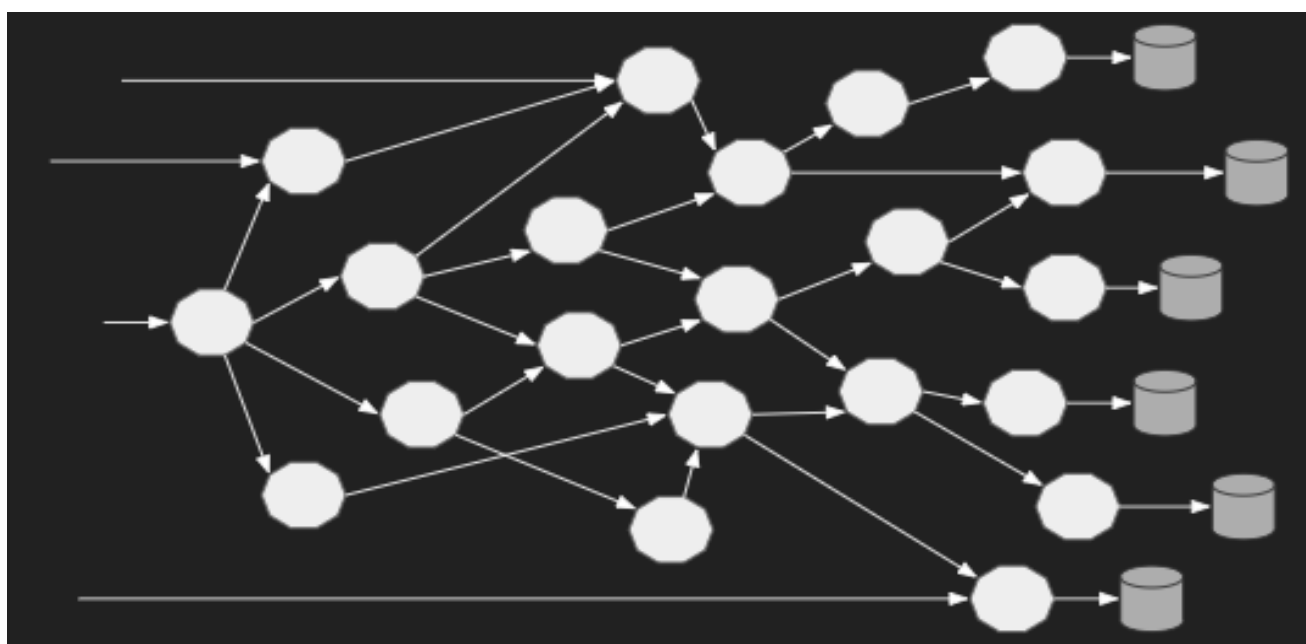


Figure 13 Architecture microservices [6]

Un des problèmes avec les applications distribuées est qu'elles communiquent sur un réseau ce qui n'est pas fiable. Par conséquent, vous devez concevoir vos microservices afin qu'ils soient tolérants aux pannes et gèrent les pannes avec élégance. Dans votre architecture de microservices, il peut y avoir une douzaine de services qui se parlent. Vous devez vous assurer qu'un service défaillant ne fait pas tomber l'ensemble de l'architecture.

2.2.1.2 La solution de Netflix

Netflix est l'un des premiers à adopter les microservices. Pour suivre son taux de croissance, Netflix a pris la décision de passer des centres de données monolithiques à une architecture de microservices basée sur le cloud pour atteindre une disponibilité, une évolutivité et une vitesse élevées. Sur la base de son histoire de réussite, Netflix a un certain nombre d'outils et technologies Open source qu'il sont utilisé dans son architecture de microservices.

Après le succès de Netflix, plusieurs entreprises ont adapté ces outils et composants dans leur transition du monolithe aux microservices.

Netflix OSS est un ensemble de bibliothèques et de frameworks ouverts par Netflix pour résoudre les problèmes de conception de systèmes distribués à grande échelle.

Avec quelques annotations simples utilisant le framework Spring boot, vous pouvez rapidement activer et configurer les modèles communs dans votre application et créer de grands systèmes distribués avec des composants Netflix. Les modèles fournis incluent la découverte de services « Service Discovery » (Eureka), le Circuit Breaker (Hystrix), le routage intelligent « Intelligent Routing » (Zuul) et l'équilibrage de charge côté client (Ribbon). [7]



Figure 14 Les outils de Netflix

- **Les limitations de Netflix OSS**

Les bibliothèques de Netflix sont couplées à la plate-forme Java et conviennent parfaitement si vous développez des services dans la plate-forme Java. Cependant, dans une architecture multi langages de programmation, vous devez rechercher d'autres bibliothèques pour gérer la communication entre les services.

Les bibliothèques de Netflix doivent être intégrées à l'intérieur de chaque microservice ainsi que le code fonctionnel de notre application. Cette configuration doit être dupliquer dans tous les services.

Le fait d'avoir à la fois la logique métier et la logique d'infrastructure dans le service, augmente la complexité globale de l'application.

La complexité opérationnelle est également augmentée, car les mises à niveau des composants Netflix implique la mise à niveau de tout le service.

Des outils supplémentaires sont nécessaires pour améliorer l'observabilité de votre architecture de microservices.

2.2.1.3 Les service mesh

2.2.1.3.1 Définition

Un Service Mesh est un moyen de contrôler la façon dont différents éléments d'une application partagent des données les uns avec les autres. Contrairement à d'autres systèmes de gestion des communications, un Service Mesh est une couche d'infrastructure dédiée, créée directement dans l'application. Cette couche d'infrastructure visible peut indiquer la manière dont les différents éléments d'une application interagissent entre eux. Il devient dès lors plus facile d'optimiser les communications et d'éviter les temps d'arrêt lorsque l'application évolue. [8]

2.2.1.3.2 Fonctionnalités et cas d'utilisation

En se basant sur la documentation [8], le Service mesh offre des capacités essentielles, notamment :

- **La découverte de services « Service Discovery » :**

Lorsqu'une instance doit interagir avec un autre service, elle doit découvrir une instance disponible de l'autre service, utilisant une recherche DNS. L'infrastructure d'orchestration de conteneurs conserve une liste d'instances prêtes à recevoir des demandes et fournit l'interface pour les requêtes DNS.

- **L'équilibrage de charge « Load balancing » :**

La plupart des frameworks d'orchestration fournissent un équilibrage de charge de couche transport du modèle TCP/IP. Le service mesh implémente un équilibrage de charge de la couche application du modèle TCP/IP, avec des algorithmes pour l'équilibrage et une gestion du trafic plus puissante. Les paramètres d'équilibrage de charge peuvent être modifiés, ce qui permet d'orchestrer Blue/Green ou Canary déploiements.

- **Le chiffrement « Encryption » :**

Le Service mesh peut crypter et décrypter les demandes et les réponses, et il peut également améliorer les performances en priorisant la réutilisation des connexions existantes, ce qui réduit le besoin de créer de nouvelles connexions.

Le service mesh utilise TLS mutuel (mTLS) pour le chiffrement du trafic, où une infrastructure à clé publique génère et distribue des certificats et des clés à utiliser par les mandataires du side-car.

- **Surveillance et Observabilité, « Monitoring & Observability » :**

Le service mesh offre une surveillance et une observabilité.

La surveillance rend compte de l'intégrité globale du système.

L'observabilité se concentre sur des informations très granulaires sur le comportement des systèmes.

- **La traçabilité « Traceability » :**

Service mesh fournit une traçabilité distribuée, est un processus qui permet aux développeurs de profiler, surveiller, déboguer et mieux comprendre les applications de microservices. Il est particulièrement utile pour identifier les problèmes racine et identifier les performances médiocres et les emplacements de défaillance dans les systèmes distribués.

Le suivi distribué vous permet de suivre les activités résultant des demandes vers une application, afin que vous puissiez suivre les chemins des demandes, trouver la latence dans ces chemins et identifier les composants créant des problèmes.

- **L'authentification et l'autorisation :**

Le service mesh peut autoriser et authentifier les demandes effectuées à l'extérieur et à l'intérieur de l'application, en envoyant uniquement des demandes validées aux instances.

- **Le disjoncteur « Circuit Breaker » :**

Le circuit breaker permet de contrôler la collaboration entre différents services afin d'offrir une grande tolérance à la latence et à l'échec.

Pour cela, en fonction d'un certain nombre de critères d'erreur (timeout, nombre d'erreurs, élément dans la réponse), ce patron de conception permet de désactiver l'envoi de requêtes au service appelé et de renvoyer plus rapidement une réponse alternative de repli (fallback), aussi appelée graceful degradation.

2.2.1.3.3 Les architectures des services mesh

- **L'architecture de la bibliothèque « The Library architecture »**

Chacun des microservices comporte une copie de la bibliothèque qui contient toutes les fonctions de service mesh (circuit breaker, traçabilité, ...) souhaitées.

L'approche architecturale de la bibliothèque était :

- La première architecture à adopter.
- La méthode la plus simple à mettre en œuvre.
- Présente quelques inconvénients en termes de performances.
- Plus difficile à maintenir.
- La demande et les performances conflictuelles sont plus difficiles à déterminer et à résoudre rapidement.

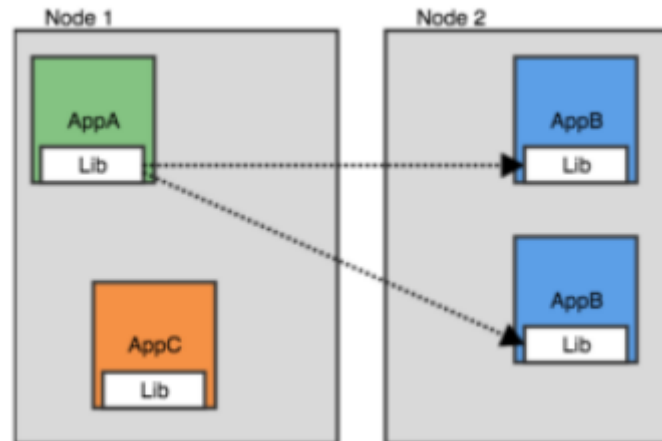


Figure 15 The Library architecture [9]

- **L'architecture d'agent de nœud « Node Agent Architecture »**

L'architecture d'agent de nœud est plus facile à gérer et à maintenir que l'architecture de bibliothèque :

- C'est l'opposé du modèle de bibliothèque : peut être utilisé avec n'importe quel langage de programmation.
- Il distribue une copie de la configuration à chaque nœud.
- Dans l'architecture Node Agent, un agent distinct s'exécute sur chaque nœud de travail d'un cluster.

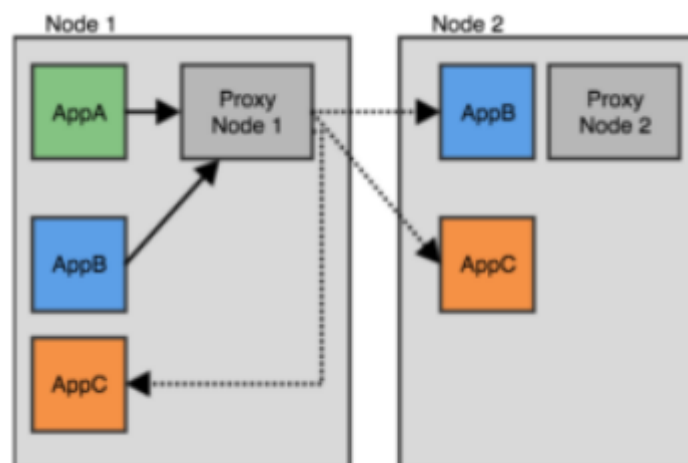


Figure 16 Node Agent Architecture [9]

L'architecture d'agent de nœud est un point de défaillance unique « Single point of failure ». Si le nœud proxy injecter par le service mesh échoue, nous perdons tout le trafic vers ce nœud.

- **L'architecture Sidecar**

Sidecar est la dernière méthode développée pour le service mesh :

- Le service mesh sidecar déploie un proxy dans un conteneur adjacent pour chaque conteneur d'application.
- Le conteneur sidecar gère tout le trafic réseau entrant et sortant du conteneur d'application.
- Pour éliminer le potentiel d'une attaque réseau, le sidecar a les mêmes privilèges que l'application à laquelle il est attaché.
- La communication entre le Sidecar et l'application est plus sécurisée que de l'architecture d'agent de nœud, dans ce dernier il faut traverser le réseau vers un Node Agent externe pour chaque intercommunication.

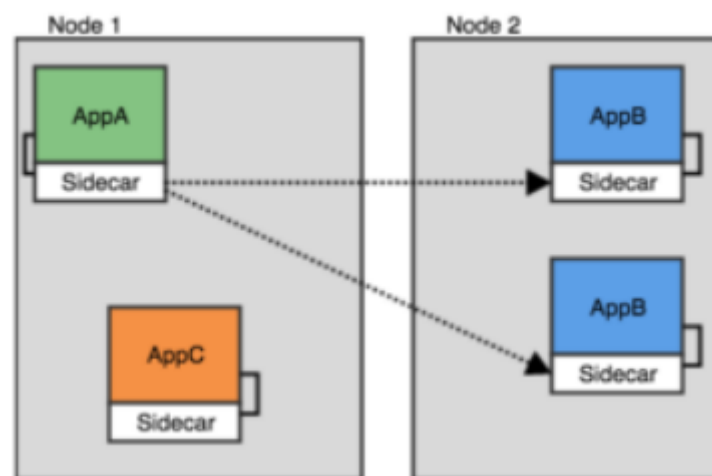


Figure 17 L'architecture Sidecar [9]

Envoy en tant que sidecar : Envoy est bien adapté au déploiement en tant que sidecar, ce qui signifie qu'il est déployé aux côtés de l'application et l'application interagit avec le monde extérieur via Envoy Proxy. Utilisant Envoy en tant que sidecar signifie que les microservices ne doivent pas inclure de nombreuses bibliothèques Netflix OSS.

2.2.2 Les solutions existantes et les solutions candidates

Après la compréhension de notre domaine d'étude, nous avons collecter les informations qui concernent les différents services mesh existants. , parmi les solutions que nous avons trouvées :

Solutions existantes :			
Istio	Aws App Mesh	Consul	Maesh
Mesher	Kuma	Aspen Mesh	Network Service Mesh
Linkerd 1.x	Linkerd 2.x	Grey Matter	Maistra
A10 Secure Service Mesh	Yggdrasil	Rotor	SOFAMesh

Tableau 1 Les solutions service mesh existantes [10]

- **Le choix initial**

Le but de cette étape est d'identifier les services mesh candidates, la sélection a été faite en se basant sur les réponses des questions suivantes :

- Est-ce que ce service mesh utilise l'architecture Sidecar ou bien l'architecture Agent des nœuds ?
- Est-ce que ce service mesh utilise le proxy Envoy ?
- Est-ce que ce service mesh peut être utilisé avec une plateforme e-commerce ?
- Est-ce que ce service mesh est encore supporté par sa communauté ?

Après avoir fait une réunions avec les architectes, nous avons sélectionné trois services mesh comme solution candidates : **Istio, Linkerd, Consul.**

2.2.3 Présentation des solutions candidates

- **Istio**

La sortie d'Istio est le résultat d'une collaboration entre IBM, Google, Lyft et Redhat pour fournir la gestion des flux de trafic, l'application des règles d'accès et l'agrégation des données de télémétrie entre les microservices. Tout cela est réalisé sans nécessiter de modification du code d'application. Ainsi, les développeurs peuvent se concentrer sur la logique métier et intégrer rapidement de nouvelles fonctionnalités. [11]



Figure 18 Istio logo

- **Linkerd**

Linkerd est un service mesh pour les Kubernetes. Il rend l'exécution des services plus facile et plus sûre en vous offrant une observabilité, une fiabilité et une sécurité d'exécution, le tout sans nécessiter de modifications de votre code. [12]



Figure 19 Linkerd logo

- **Consul**

Consul « connect », est le service mesh de HashiCorp, fournit un réseau de service à service et une sécurité par l'autorisation de connexion et le cryptage en utilisant la sécurité mutuelle de la couche transport (mTLS). Les applications déployées avec la fonction « connect » ; peuvent utiliser des proxies sidecar dans une configuration service mesh pour établir des connexions TLS pour les connexions entrantes et sortantes, sans avoir connaissance du Consul. [13]

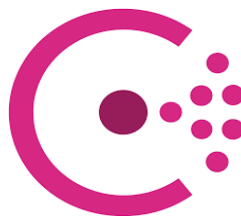


Figure 20 Consul logo

2.3 Etude comparative

La comparaison fonctionnelle sera basée sur les critères qui sont au-dessous, le rôle du client est de choisir parmi ces critères celles qui répond bien à son besoin, et notre rôle est de livrer un tableau contient les fonctionnalités qui sont fourni par le service mesh.

2.3.1 Critères de comparaison

- **Plateforme** : Est-ce le client besoin d'un mesh mono- ou multi-plateforme (compatible seulement avec kubernetes ou bien aussi avec Nomad, virtuelle machine et Mesos).
- **Mesh Expansion** : Est- ce le client besoin d'un mesh qui supporte l'option « Mesh Expansion », c'est une option pour connecter les ressources externes (VMs, ..) avec un service mesh déployé en kubernetes.
- **Multi-cluster Mesh** : Est le client besoin d'un mesh qui peut contrôler et observer une application déployée dans multi-cluster.
- **Supported Protocols** : Quels sont les protocoles utilisés par le client, Savoir si le service mesh supporte ces protocoles ou non. Parmi ces protocoles, TCP, http/1.1, http/2, gRPC.
- **Fonctionnalités de routage** : Quelle sont les fonctionnalités de routage présente le besoin de client :
 - **Traffic Access Control** : Permet de définir une stratégie de contrôle d'accès pour les applications.
 - **Load Balancing** : Le service mesh est ce qu'il supporte l'équilibrage des charges.
 - **Traffic specs** : Permet de spécifier à quoi ressemble leur trafic.
 - **Traffic Split** : Diriger le trafic entre divers services utilisant un pourcentage pour chaque service ou bien de diviser le trafic en fonction de l'en-tête « Header » et le chemin « Path ».
 - **Traffic Metrics** : Bénéficier de la consommation de métriques liées au trafic http.
- **Fonctionnalités de surveillance** : Quelle sont les fonctionnalités de surveillance présente le besoin de client :
 - **Tableau de bord** : Offre une vue d'ensemble de ce qui se passe avec les services en temps réel.
 - **Prometheus** : Une boîte à outils de surveillance et d'alerte.

- **Grafana** : Un logiciel de visualisation et d'analyse. Il nous permet de visualiser, d'alerter et d'explorer les mesures.
- **Access Log « Journaux d'accès »** : Option pour stocker les journaux pour un système distribué.
- **Distributed Tracing « Traçage distribué »** : permet de localiser les défaillances.
- **Gestion du trafic « Traffic Management »** : Quelle sont les fonctionnalités de gestion du trafic présente le besoin de client :
 - **Circuit breaking** : Un modèle permettant de surveiller les services d'application.
 - **Green/Blue deployments** : Technique pour exécuter deux environnements de production identiques appelés Vert et Bleu.
 - **L'injection de défauts « Fault injection »** : Une technique de test, qui aide à comprendre comment un système se comporte lorsqu'il est soumis à des contraintes inhabituelles.
 - **Retry & Timeout** : Limiter le délai d'attente qu'un utilisateur client attend avant d'abandonner complètement.
 - **L'injection des retards « Delay injection »** : Injection des retards pour tester la disponibilité de l'application en cas de retard d'un service.
 - **La limitation des tarifs « Rate limiting »** : Est une technique défensive, protège les services d'une utilisation excessives.
- **Sécurité** : Quelle sont les fonctionnalités de sécurité présente le besoin de client :
 - **Authentication mTLS** : Offre une sécurité côté client et serveur pour les communications de service à service.
 - **La sécurité des communication multi-cluster** : Définissez comment le service mesh peut gérer la sécurité des communications entre un service du cluster et un autre service d'un autre cluster.
 - **Les règles d'autorisation « Authorization Rules »** : Spécifie la stratégie de sécurité qui s'applique à un objet en fonction de diverses conditions, telles que le contexte et l'environnement

- **Les certificats d'autorisation externe (ECA) :** Est une entité tierce de confiance qui émet des certificats numériques et gère les clés publiques et les informations d'identification pour le chiffrement des données.

2.3.2 Comparaison des services mesh

2.3.2.1 Comparaison des services Mesh basée sur ces critères

Cette partie résume une comparaison des services Mesh en se basant sur les fonctionnalités et les limitations de chaque Mesh, le travail sera découpé en 3 sections :

- Benchmarking de l'observabilité et surveillance
- Benchmarking du management de trafic
- Benchmarking de la sécurité

Benchmarking de l'observabilité et la surveillance :

- Pour le traçage distribué, dans Istio nous pouvons utiliser un traçage distribué automatiquement utilisant les outils Jaeger ou bien Zipkin, tant qu'ils sont déjà installés avec Istio, juste il nous faut d'envoyer les « endpoints » de ces outils vers l'application et de les configurer dans notre application pour que l'application puisse envoyer les métriques vers les outils. Par contre dans Linkerd, les outils de traçage ne sont pas installés, nous avons besoin d'utiliser un proxy comme OpenCensus qui va être un intermédiaire entre la communication de l'application avec les outils. Dans Consul, les outils de traçage distribué ne sont pas installés mais dans la documentation de Consul, il est possible de configurer les outils Jaeger et Datadog, donc lorsque nous avons essayé de les installer nous avons trouvé que la configuration dans Consul de ces outils est vraiment très difficile et il y a un manque de documentation surtout dans Kubernetes.
- À propos des journaux d'accès « Access Logs » les 3 services Meshs le fournissent avec les mêmes résultats.

Benchmarking du management de trafic :

- Pour la répartition du trafic « Traffic Split », dans ce cas nous parlons de « Canary Release », « Blue/Green deployment » et « A/B testing », la complexité de les configurer dans Istio et Linkerd est presque la même complexité, dans Istio nous avons besoin de configurer un «

VirtualService » et pour Linkerd nous utilisons un « TrafficSplit workload », dans Istio nous avons besoin d'une configuration « DestinationRule » pour définir les services destinataires, par contre dans Linkerd nous avons besoin de créer une configuration « Service » pour chaque hôte destinataire, qui va jouer le rôle de « DestinationRule » dans Istio. Pour la répartition du trafic qui arrive de l'extérieur du « Cluster », Istio contient son propre config « Ingress Gateway » qui va jouer le rôle d'un « API Gateway », par contre Linkerd ne fournit pas son propre config, dans ce cas, nous avons besoin d'utiliser un « Ingress Controller » externe comme Nginx.

- Pour les résultats, Les deux services Meshs donnent les mêmes résultats, par contre Istio a la possibilité de répartir le trafic basé sur les entêtes et les chemins « Paths », est c'est un avantage pour Istio.
- Voici un exemple de répartition du trafic utilisant 90% de trafic vers la version 1 de notre application et 10% du trafic vers la version 2 :

```
root@debug-689dd7d7f-lrgvh:/# for i in {1..10}; do curl hello-sqli.bench-linkerd.svc.cluster.local.; echo; done;
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 1
Hello Sqli from version 2
Hello Sqli from version 1
Hello Sqli from version 1
```

Figure 21 Répartition du trafics

- À propos les réessays et les délais d'attentes « Timeout and Retry », Linkerd est la puissance dans cette fonctionnalité, parce qu'il donne la possibilité d'être basé sur les chemins d'http, par contre dans Istio est globale pour toute l'application.

```
spec:
  routes:
  - condition:
      method: GET
      pathRegex: /
      name: GET /
      timeout: 0.1s      #timeout
      isRetryable: true

  retryBudget:
    retryRatio: 0.2
    minRetriesPerSecond: 10
    ttl: 2s
```

Figure 22 retry et timeout dans Linkerd

```
http:
- route:
  - destination:
      host: hello-sqli-v2
      subset: v2
    timeout: 10s
  retries:
    attempts: 3
    perTryTimeout: 2s
```

Figure 23 retry et timeout dans Istio

- Il reste dans le management du trafic le « Circuit breaker », cette fonctionnalité qui est très importante dans les architectures Microservices, Linkerd ne fournit pas cette fonctionnalité.
- A propos Consul, nous n'avons pas trouvé une documentation qui peut nous aider de configurer cette fonctionnalité dans Kubernetes.

```
trafficPolicy:
  connectionPool:
    http:
      http1MaxPendingRequests: 1
      maxRequestsPerConnection: 1
    tcp:
      maxConnections: 1
  outlierDetection:
    baseEjectionTime: 20s
    consecutiveErrors: 3
    interval: 10s
    maxEjectionPercent: 100
```

```
fortio load -c 100 -H "Host:hello-sqli.com" http://52.224.16.59
```

```
Code 200 : 6 (15.0 %)
Code 503 : 34 (85.0 %)
```

Figure 24 Configuration d'un circuit breaker

Après l'envoi de 100 requêtes de connexions, le test montre que 85.0 % d'accès est limité

Benchmarking de la sécurité :

- Ces trois services Meshs fournissent une configuration mTLS automatique pour la communication interne entre les services, ils fournissent aussi la possibilité d'utiliser une solution externe pour la gestion de certificats. [14] [15] [16]

Synthèse :

- D'après les résultats de Benchmarking l'observabilité et la surveillance, nous avons remarqué que Istio est le meilleur en terme l'observabilité.
- Les résultats de Benchmarking de management du trafic, montrent que les « Timeouts » dans Linkerd peuvent être basés sur le chemin http, qui n'est pas fourni par les autres services Meshs et c'est un avantage pour Linkerd, mais il n'offre pas la configuration des « Circuits Breaker » donc un point faible aussi pour cette solution.
- Dans le cas de Consul, nous avons réalisé une documentation et une étude spécifique pour le service Mesh, parce qu'on n'a pas trouvé de documentations qui vont nous aider à implémenter ses fonctionnalités dans Kubernetes surtout celles de management du trafic.

D'après cette étude, nous avons synthétisé que Consul a été annoncé en 2014, et c'était avant la naissance de Kubernetes.

2.3.2.2 Comparaison de performance

Après la phase I de la collecte des informations sur les différents Meshs, et la première étape de cette phase qui a été basée sur une comparaison fonctionnelle.

L'étape 2 résume le résultat d'une étude de la performance des services Meshs au niveau de leur latence.

En effet, nous cherchons à comprendre les performances du Mesh dans des conditions normales de fonctionnement d'un cluster sous charge. Cela signifie que les applications de clusters, bien qu'elles soient sollicitées, sont toujours capables de répondre dans un délai raisonnable. Pendant que le système est en charge (test), l'expérience de l'utilisateur lors de l'accès aux pages web desservies par le cluster ne devrait pas souffrir au-delà d'un degré acceptable. Si, en revanche, les latences étaient régulièrement repoussées dans la plage des secondes (ou des minutes), une application de cluster réelle offrirait une mauvaise expérience à l'utilisateur, et les opérateurs (ou les auto-scalers) réduiraient l'échelle.

Le trafic HTTP à un taux constant de requêtes par seconde (rps) est le stimulus de test, et nous mesurons la latence de réponse pour déterminer la performance globale d'un Mesh. Le même point de repère des rqs est également effectué pour la même application sans l'injection d'un service Mesh afin de comprendre le niveau de rendement du cluster et les applications.

- **Stratégie**

Au début, nous avons essayé de chercher des Benchmarking de performance qui existent dans le web, et dans la réalisation de notre étude, nous avons inspiré par l'étude de Michael Kipper dans [17] et l'étude de Thilo Fromm dans [18].

Pour tester la performance de chaque service Mesh, nous avons besoin de déployer une application 4 fois : un déploiement pour chaque service Mesh et un autre pour le cas « sans utilisation » d'un Mesh.

Utilisation d'un outil de Benchmarking pour tester la performance de chaque application.

- **Outils utilisés**

Fortio : est une bibliothèque de test de charge de Microservices (http, grpc), un outil en ligne de commande, un serveur d'écho. Fortio permet de spécifier un chargement de requête par seconde et d'enregistrer des histogrammes de latence et d'autres statistiques utiles.

Azure Cluster Service : Pour le déploiement de mes applications.

Azure Virtual Machine : Pour l'installation de Fortio

Istio

Linkerd

Consul

Kubernetes : Comme outils d'orchestration de mes conteneurs.

- **Environnement**

Tant que notre client utilise le Cloud Azure, nous avons choisis Azure Cluster AKS comme un environnement de déploiement, avec les informations suivantes :

Le cluster était déployé sur la région de centre de données « Datacenter » : East US

Le cluster contient 2 nœuds

Chaque nœud avec 2 CPU, 7 en mémoire.

Total : 4 Core Processeurs virtuels (vCPUs) et 14 Gio de mémoire. (RAM)

- **Démarche**

Avant de commencer l'analyse comparative, nous devons bien organiser les choses, pour cela, nous avons créé pour chaque service Mesh un espace de noms « Namespace » au but d'isoler les applications.

Pour l'application qui sera déployé sans service Mesh, nous avons utilisé l'espace de noms 'default'.

Nous avons installé Istio, Linkerd et consul dans notre cluster et injecté chacun sur un espace de donnée différent. Après l'injection des services Mesh, on a déployé l'application du test dans chaque « Namespace ».

Après que le déploiement des applications, On a installé Fortio dans une machine virtuelle dans Azure pour avoir une latence minimale lors de la génération des requêtes.

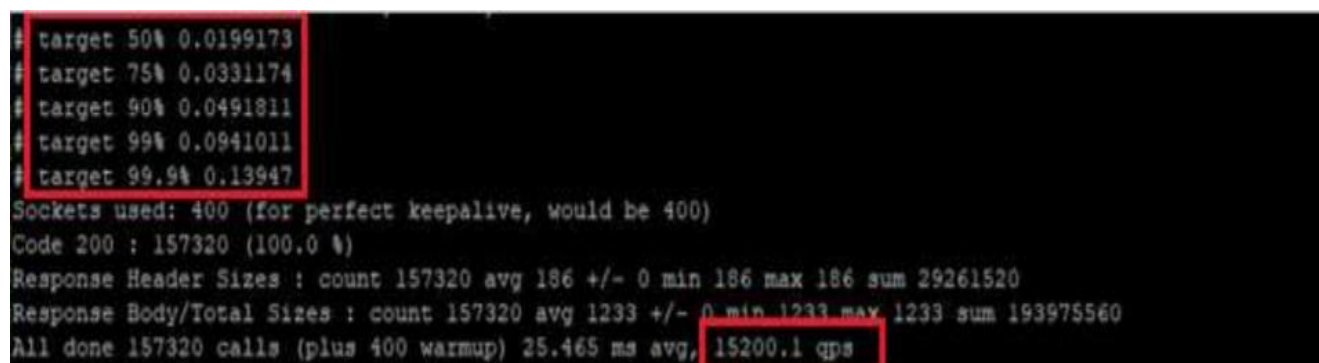
On a commencé à générer des mêmes nombres de requêtes pour les quatre versions de l'application. Cela est fait en utilisant la commande : « *fortio load -c <nombre de cnx> -qps <nombre de requêtes> -t <temps de test par seconds> http://URL* ».

Notre stratégie est de faire plusieurs tests. Dans chacun, on va connecter un nombre N de connexions vers l'application, avec le maximum possible des requêtes par seconds pour chaque connexion et de tester si l'application résiste à ce nombre. Si la latence est parfaite, on augmente le nombre de connexion dans le deuxième test.

Test 1 : 400 connexions pour 10 seconds :

Dans notre premier test, nous avons utilisé 400 connexions, et nous observons quelle service Mesh résistera.

App sans Service Mesh :



```
# target 50% 0.0199173
# target 75% 0.0331174
# target 90% 0.0491811
# target 99% 0.0941011
# target 99.9% 0.13947
Sockets used: 400 (for perfect keepalive, would be 400)
Code 200 : 157320 (100.0 %)
Response Header Sizes : count 157320 avg 186 +/- 0 min 186 max 186 sum 29261520
Response Body/Total Sizes : count 157320 avg 1233 +/- 0 min 1233 max 1233 sum 193975560
All done 157320 calls (plus 400 warmup) 25.465 ms avg, 15200.1 qps
```

Figure 25 Résultat de génération 400 connexions pour l'application sans service Mesh

Le résultat montre que le serveur qui répond à 15 200 requêtes par seconds, et 99.9% de la page sera chargée dans 0.13 seconds.

App avec Istio :

```
# target 50% 0.0244343
# target 75% 0.0397202
# target 90% 0.0585879
# target 99% 0.0987306
# target 99.9% 0.15578
Sockets used: 400 (for perfect keepalive, would be 400)
Code 200 : 130896 (100.0 %)
Response Header Sizes : count 130896 avg 186 +/- 0 min 186 max 186 sum 24346656
Response Body/Total Sizes : count 130896 avg 1233 +/- 0 min 1233 max 1233 sum 161394768
All done 130896 calls (plus 400 warmup) 30.583 ms avg, 13056.4 qps
```

Figure 26 Résultat de génération 400 connections pour l'application injecté par Istio

Le résultat montre que le serveur qui répond à 13 056 requêtes par seconde, et 99.9% de la page sera chargée dans 0.15 secondes.

App avec Consul :

```
# target 50% 0.0214718
# target 75% 0.0332332
# target 90% 0.0501006
# target 99% 0.107036
# target 99.9% 0.186603
Sockets used: 400 (for perfect keepalive, would be 400)
Code 200 : 147928 (100.0 %)
Response Header Sizes : count 147928 avg 186 +/- 0 min 186 max 186 sum 27514608
Response Body/Total Sizes : count 147928 avg 1233 +/- 0 min 1233 max 1233 sum 182395224
All done 147928 calls (plus 400 warmup) 27.081 ms avg, 14747.2 qps
```

Figure 27 Résultat de génération 400 connections pour l'application injecté par Consul

Le résultat montre que le serveur qui répond à 14 747 requêtes par seconde, et 99.9% de la page sera chargée dans 0.18 secondes.

App avec Linkerd :

```
# target 50% 0.0597509
# target 75% 0.0833586
# target 90% 0.113841
# target 99% 0.24788
# target 99.9% 0.489285
Sockets used: 400 (for perfect keepalive, would be 400)
Code 200 : 57646 (100.0 %)
Response Header Sizes : count 57646 avg 186 +/- 0 min 186 max 186 sum 10722156
Response Body/Total Sizes : count 57646 avg 1233 +/- 0 min 1233 max 1233 sum 71077518
All done 57646 calls (plus 400 warmup) 69.550 ms avg, 5734.3 qps
```

Figure 28 Résultat de génération 400 connections pour l'application injecté par Linkerd

Le résultat montre que le serveur que répondre à 5734 requêtes par seconds, et 99.9% de la page sera chargé dans 0.48 seconds.

En guise de conclusion dans le premier test, nous voyons que Linkerd n'a pas résister au 400 connections, notre page web se charge dans 0.5 second et il répond juste aux 5700 requêtes par contre consul répond au 14 700 ou bien Istio répond au 13 000 requête.

Test 2 : 1000 connexions pour 180 seconds (3 minutes) :

Dans notre deuxième test, nous avons envoyé 1000 connexions, et nous observons quelle service Mesh résistera.

App sans Service Mesh :

```
# target 50% 0.0533129
# target 75% 0.0870085
# target 90% 0.129673
# target 99% 0.228324
# target 99.9% 0.324421
Sockets used: 1000 (for perfect keepalive, would be 1000)
Code 200 : 2714071 (100.0 %)
Response Header Sizes : count 2714071 avg 186 +/- 0 min 186 max 186 sum 504817206
Response Body/Total Sizes : count 2714071 avg 1233 +/- 0 min 1233 max 1233 sum 3.34644954e+09
All done 2714071 calls (plus 1000 warmup) 66.332 ms avg, 15071.9 qps
```

Figure 29 Résultat de génération 1000 connexions pour l'application sans service Mesh

Le test a montré que le serveur peut répondre à 15 071 requêtes par seconds, et 99.9% de la page sera charger dans 32.4 ms.

Par rapport au premier test, nous observons que 15 000 est le nombre maximal de requêtes que le serveur peut répondre par second, et la latence ont diminué de 0.13 second vers 0.32 second.

Avec Istio :

```
# target 50% 0.0671039
# target 75% 0.101003
# target 90% 0.146331
# target 99% 0.243202
# target 99.9% 0.327598
Sockets used: 1000 (for perfect keepalive, would be 1000)
Code 200 : 2268956 (100.0 %)
Response Header Sizes : count 2268956 avg 186 +/- 0 min 186 max 186 sum 422025816
Response Body/Total Sizes : count 2268956 avg 1233 +/- 0 min 1233 max 1233 sum 2.79762275e+09
All done 2268956 calls (plus 1000 warmup) 79.340 ms avg, 12601.5 qps
```

Figure 30 Résultat de génération 1000 connexions pour l'application injecté par Istio

Le test a montré que le serveur peut répondre à 12 601 requêtes par seconds, et 99.9% de la page sera charger dans 0.32 second.

Par rapport au premier test, nous observons que les requêtes que le serveur peut répondre par second ont diminué par 400 r/s de 13 000 vers 12 600 r/s, et la latence a diminué de 0.15 second vers 0.32 second.

Avec Consul :

```
# target 50% 0.0554625
# target 75% 0.0832136
# target 90% 0.121971
# target 99% 0.237467
# target 99.9% 0.355172
Sockets used: 1000 (for perfect keepalive, would be 1000)
Code 200 : 2694813 (100.0 %)
Response Header Sizes : count 2694813 avg 186 +/- 0 min 186 max 186 sum 501235218
Response Body/Total Sizes : count 2694813 avg 1233 +/- 1.346e-05 min 1233 max 1233 sum 3.32270443e+09
All done 2694813 calls (plus 1000 warmup) 66.802 ms avg, 14965.9 qps
```

Figure 31 Résultat de génération 1000 connections pour l'application injecté par Consul

Le test a montré que le serveur peut répondre à 14 965 requêtes par seconds, et 99.9% de la page sera charger dans 0.35 second.

Par rapport au premier test, nous observons que 15 000 est le nombre maximal de requêtes que le serveur peut répondre par second, et la latence ont diminué de 0.13 second vers 0.35 second.

Avec Linkerd :

```
# target 50% 0.21468
# target 75% 0.283506
# target 90% 0.349268
# target 99% 0.550219
# target 99.9% 0.716796
Sockets used: 1000 (for perfect keepalive, would be 1000)
Code 200 : 771448 (100.0 %)
Response Header Sizes : count 771448 avg 186 +/- 0 min 186 max 186 sum 143489328
Response Body/Total Sizes : count 771448 avg 1233 +/- 0 min 1233 max 1233 sum 951195384
All done 771448 calls (plus 1000 warmup) 233.417 ms avg, 4282.0 qps
```

Figure 32 Résultat de génération 1000 connections pour l'application injecté par Linkerd

Le test a montré que le serveur peut répondre aux 4282 requêtes par seconds, et 99.9% de la page sera charger dans 0.71 second.

Par rapport au premier test, nous observons que les requêtes que le serveur peut répondre par second ont diminué de 5 700 vers 4 200 r/s, et la latence a diminué de 0.48 second vers 0.71 second.

- **Synthèse**

Le diagramme ci-dessous résume les résultats de la performance des différentes Mesh :

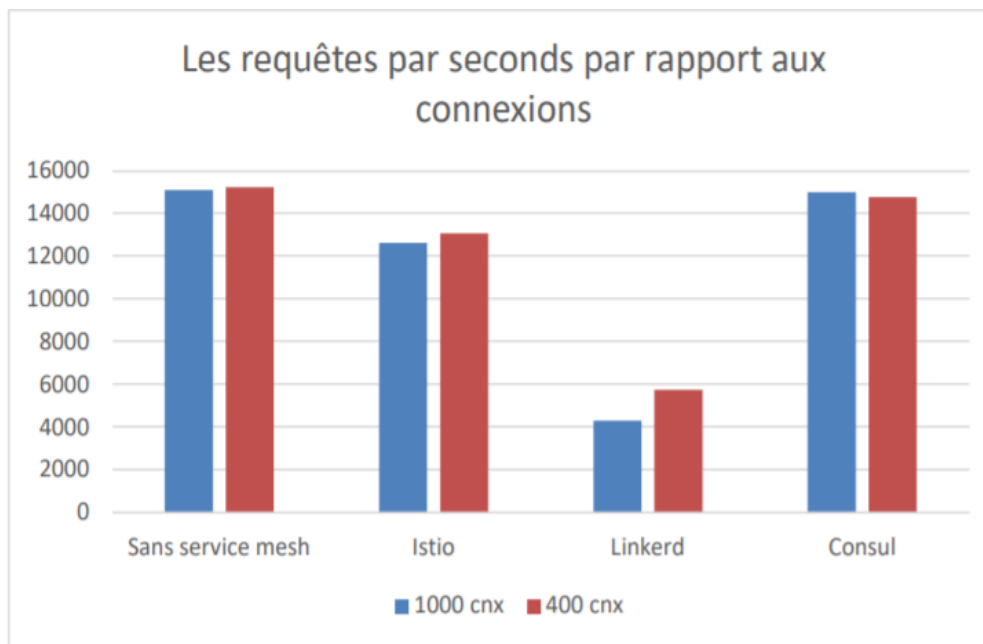


Figure 33 Résultat des requêtes par seconds

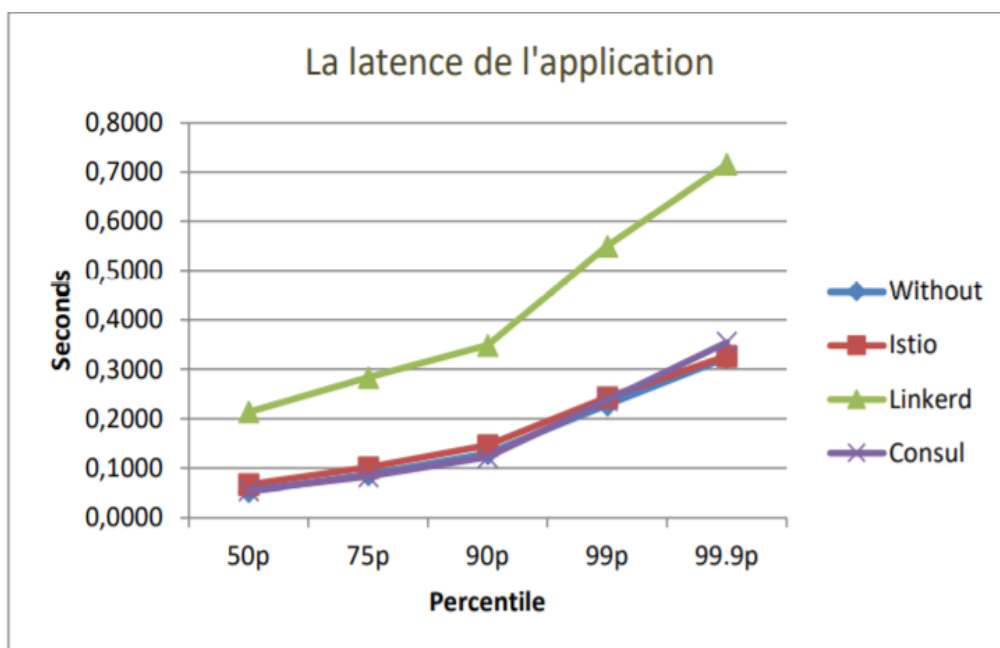


Figure 34 Résultat de la latence et charge de l'application

D'après les tests, nous concluons que Linkerd prend plus de temps pour charger l'application +100%. Sans utiliser un maillage de service ou utiliser Consul, le serveur peut répondre à 25% des requêtes par second plus qu'Istio, et 350% plus que Linkerd.

2.3.3 Le choix de la solution

Avant la prise de décision, nous avons réalisé des Demos pour appuyer les résultats de l'étape précédente c'est-à-dire illustrer les avantages et les inconvénients et les limitations de chaque service mesh, en plus pour donner preuve de notre choix et convaincre le client. Le rôle du client est de préciser les critères exigés qui répondent à ses besoins.

Le tableau suivant présente les besoins et les exigences du projet client :

Architecture	Sidecar
Platform	Kubernetes
Protocoles supportés	TCP
	HTTP
Routage	Load balancing
Observabilité	Dashboard
	Distributed tracing
Gestion du trafic	Circuit breaking
	Rate limiting
Sécurité	mTLS authentication
	External Certification Authorities (ECA)

Tableau 2 Critères demandes par le client

D'après la comparaison des résultats de benchmarking, l'étude comparative, et les besoins du projet client, on a choisi **Istio** comme la solution service Mesh la plus adapté aux besoins du projet client.

2.4 Conclusion

Le chapitre à présenter la phase de l'exploration et la collecte des informations puisque le service mesh a répondu à des problématiques qu'on peut rencontrer en implémentant l'architecture microservices. En suite le chapitre a entamé la partie du benchmarking et l'étude comparative, ainsi que les résultats des tests effectués des solutions candidates . Et finalement la décision du choix final de la solution.

Chapitre 3

Spécification des besoins et l'architecture proposée

3.1 Introduction

Ce chapitre présente les besoins fonctionnels, les besoins non fonctionnels, ainsi que l'identification de nos domaines, et finalement l'architecture globale de notre Micro-service.

3.2 Spécification des besoins globaux

La spécification des besoins représente la première phase du cycle de développement d'un logiciel. Elle sert à identifier les acteurs réactifs du système et leur associer chacun l'ensemble d'actions avec lesquelles il intervient dans l'objectif de donner un résultat optimal.

3.2.1 Besoins fonctionnels

- **Fonctionnalités assurées par le système :**

Il s'agit des fonctionnalités à assurer par l'application, l'application doit permettre de :

Gérer les comptes utilisateur :

Créer des nouveaux utilisateurs.

Spécifier un rôle à un utilisateur.

Changer le rôle d'un utilisateur.

Lister les utilisateurs.

Supprimer un Utilisateur.

Modifier un utilisateur.

Gérer les produits.

Ajouter un produit.

Consulter un produit.

Lister les produits.

Modifier un produit.

Gérer les avis des clients sur les produits.

Ajouter un avis.

Ajouter un commentaire.

Consulter les avis.

Gérer le panier.

Ajouter un ordre temporairement avec une quantité dans le panier.

Modifier la quantité d'un ordre.

Consulter le panier.

Supprimer un panier.

Supprimer un ordre du panier.

Gérer les ordres.

Ajouter un ordre.

Consulter un ordre.

Consulter tous les ordres d'un client.

Supprimer un ordre.

Gérer le check-out.

Payer l'ordre.

Notifier le client lorsque le paiement est accepté.

Livrer les ordres.

Notifier le client lorsque la livraison est envoyée.

Gérer les paiements.

Consulter les revenus

Gérer la livraison.

Envoie des livraisons.

Changer le statut d'une livraison.

- **Identification des acteurs :**

Notre application e-commerce dispose 4 types d'acteurs :

L'administrateur: L'utilisateur qui a le contrôle de toute l'application, il peut gérer les comptes utilisateur, de gérer les produits, les ordres les avis et les livraisons.

L'agent: est un utilisateur qui a un rôle spécifique dans l'application selon le besoin.

Le livreur: c'est lui qui suit la commande et de notifier le client lors du changement de statut d'une livraison.

Le client : est un utilisateur qui a le droit de payer un ordre, de consulter les produits et d'ajouter des avis.

3.2.2 Besoins non fonctionnels

Les besoins non fonctionnels de notre application sont :

- **Maintenabilité**

L'architecture doit permettre la facilité de corriger un bug trouvé dans l'application.

- **Extensibilité**

L'architecture doit permettre l'intégration de nouvelles fonctionnalités.

- **Sécurité**

L'application doit être sécurisée du côté de la communication interne des services, le trafic entrant et sortant et aussi du côté des autorisations et authentification.

- **Scalabilité**

L'application peut être mise à l'échelle verticalement et horizontalement.

- **Tolérance aux pannes**

L'application doit être conçue pour l'échec.

- **Portabilité**

L'architecture peut être utilisable dans différents environnements.

- **La simplicité**

Le système fonctionne mieux s'il reste simple plutôt que compliqués.

3.2.3 Division en Microservices

3.2.3.1 Spécification des domaines

Le besoin fonctionnel nous a conduits aux principaux contextes de notre application, qui sont :

- Contexte Utilisateur
- Contexte Ordre
- Contexte Panier
- Contexte Livraison
- Contexte notification
- Contexte La caisse
- Contexte Paiement
- Contexte Revue
- Contexte Catalogue des Produits

3.2.3.2 l'architecture de l'application

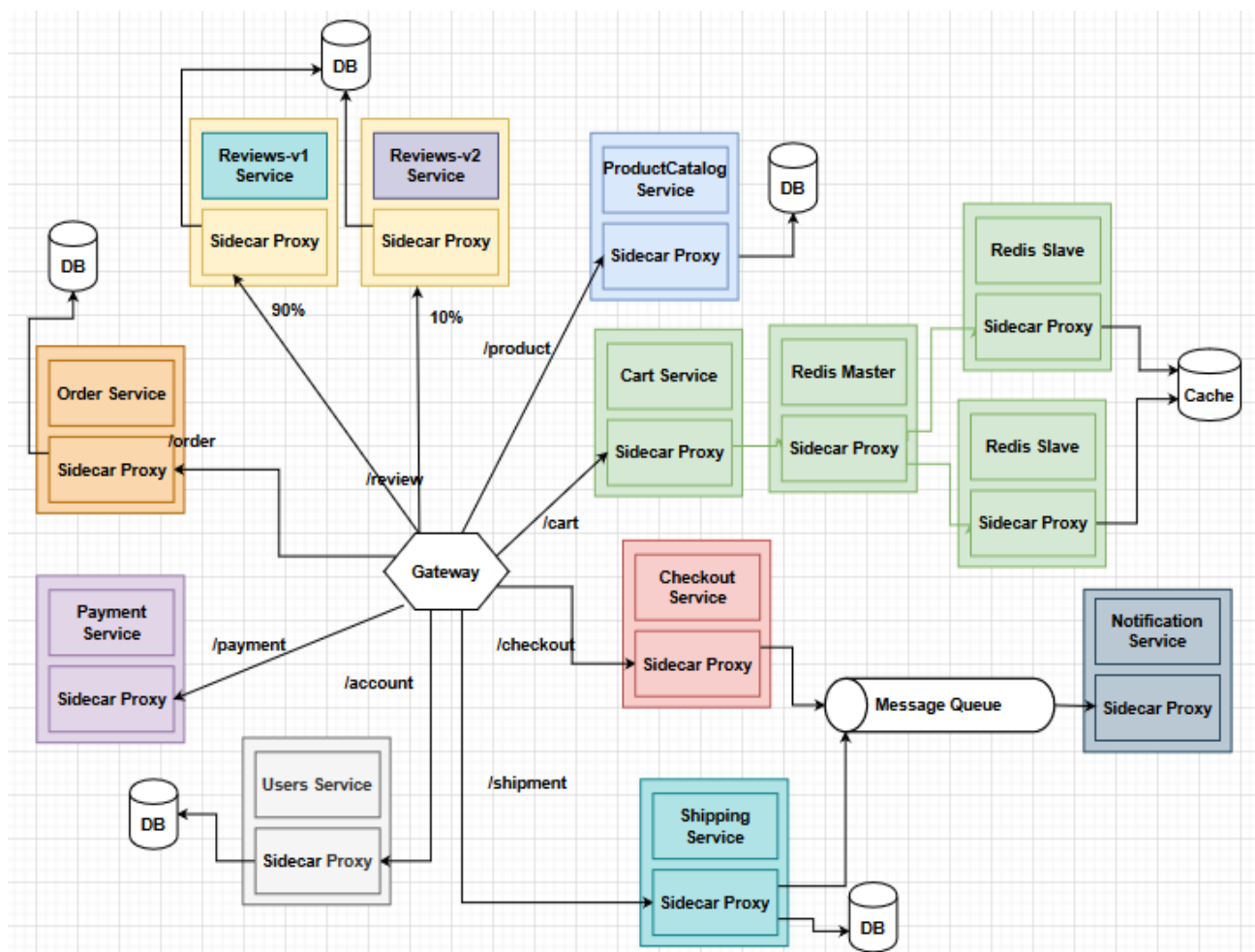


Figure 35 L'architecture de l'application

Nous présentons dans La figure ci-dessus chaque contexte avec une couleur différente. La couleur jaune présente le contexte des revues, il était demandé de réaliser deux versions pour appliquer plusieurs règles de management du et ils partagent la même base de données. La couleur orange présente le contexte des ordres, la couleur verte présente le contexte du panier qui persiste les ordres temporairement dans le cache de Redis, la couleur bleue présente le contexte des livraisons, la couleur blanche casser présente le contexte des utilisateurs et d'authentification, le couleur violet présente le contexte de paiement, le couleur rouge présente le contexte de la caisse, et on a aussi le contexte de catalogue des produits, et le contexte de notification qui attende un message arrive utilisant le broker.

La mise en cache, la messagerie et le « Gateway » seront expliqué plus tard. Le proxy sidecar était déjà expliquer dans le chapitre Etude et Benchmarking des services Mesh.

Chaque service a sa propre base de données.

3.3 Conclusion

Tout au long de ce chapitre, nous avons spécifié, tout d'abord, nos besoins fonctionnels et non-fonctionnels ce qui nous a aidés à avoir une vision plus claire et plus profonde sur notre projet et nous a aidé de conclure les contextes de nos applications. Ensuite, nous avons présenté l'architecture globale de notre application Microservices.

Chapitre 4

Conception et la documentation de l'API

4.1 Introduction

Après avoir spécifié et analysé les besoins globaux des utilisateurs, dans ce chapitre nous entamons la conception et documentation Swagger de notre l'API.

4.2 Conception

4.2.1 Le Langage de Modélisation UML

Le Langage de Modélisation Unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet. [19]



Figure 36 UML Logo

4.2.2 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est un diagramme UML utilisé pour donner une vision globale du comportement fonctionnel d'un système logiciel.

Un cas d'utilisation représente une unité discrète d'interaction entre un utilisateur (humain ou machine) et un système. Il est une unité significative de travail. Dans un diagramme de cas d'utilisation, les utilisateurs sont appelés acteurs, ils interagissent avec les cas d'utilisation. [20]

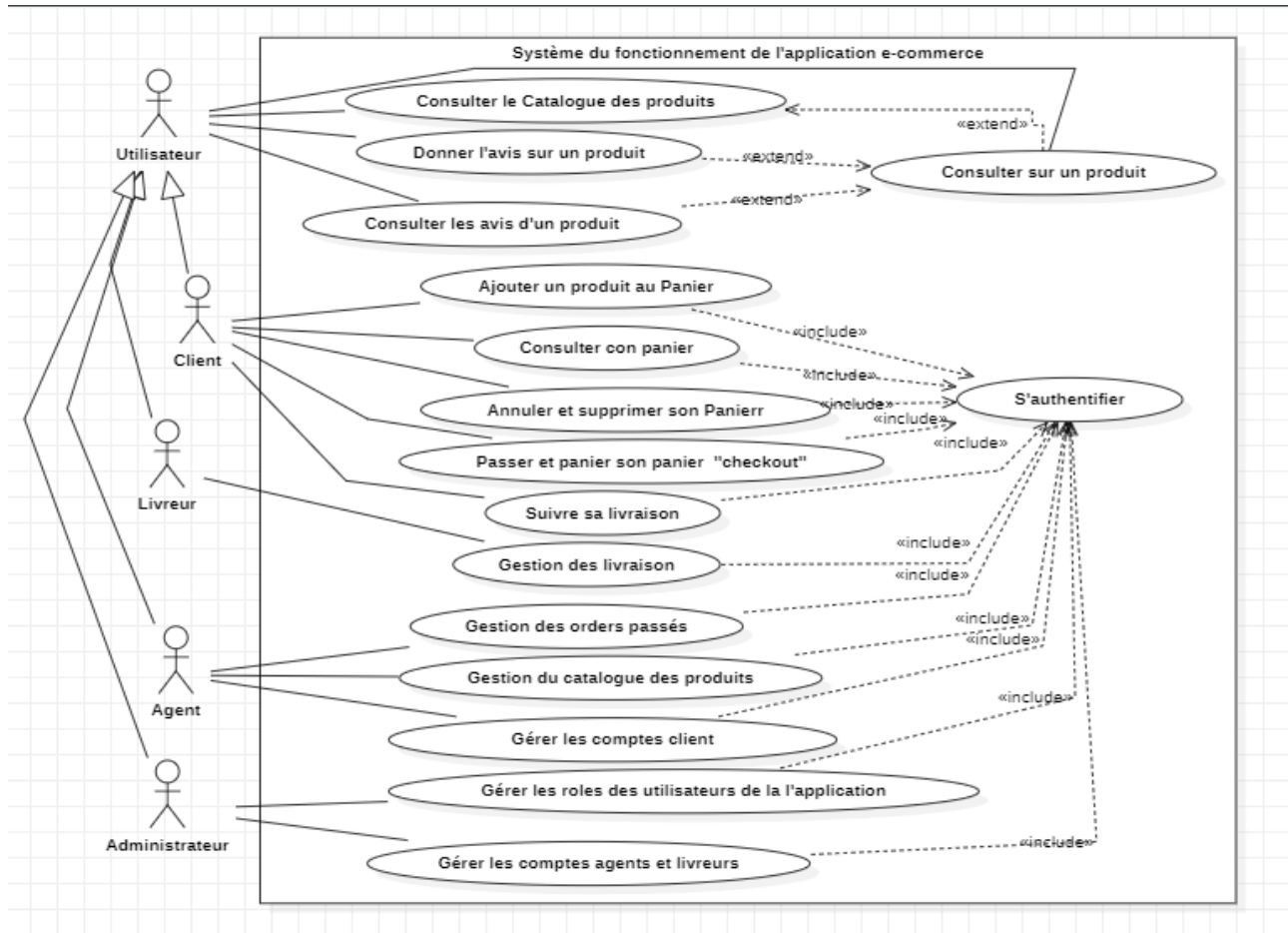


Figure 37 diagramme de cas d'utilisation

4.2.3 Diagramme de séquence

Les diagrammes de séquences modélisent l'aspect dynamique du système. Ils permettent de décrire comment les éléments du système interagissent entre eux et avec les acteurs.

Les objets au cœur d'un système interagissent en s'échangeant des messages.

Les acteurs interagissent avec le système au moyen d'IHM (Interfaces Homme-Machine). [21]

Il existe deux types de diagrammes de séquence : diagramme de séquence analyse et diagramme de séquence conception

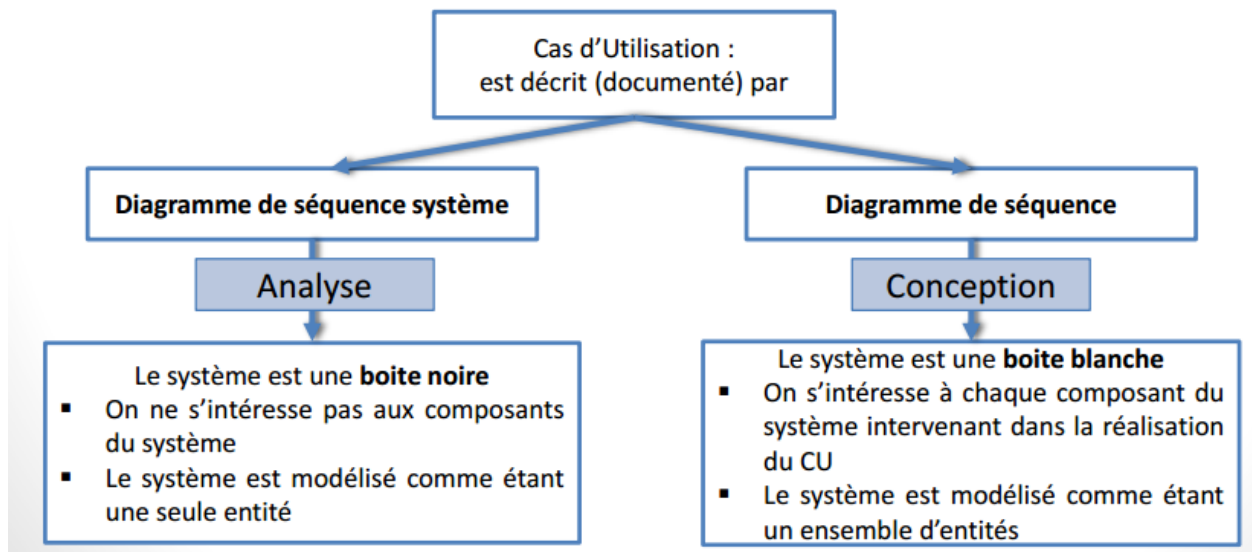


Figure 38 Les deux types de diagramme de séquences

Nous modéliserons dans ce projet des diagrammes de séquence analyse (nous avons pris comme exemple les cas d'utilisation « Authentification » et «Ajouter un produit au panier»).

Diagramme de séquence du cas « Authentification » :

Quand l'utilisateur souhaite se connecter, Il doit fournir son nom d'utilisateur et le mot de passe.

L'utilisateur saisi donc ses informations d'identification, qui seront vérifiées juste après par le système.

Tant que le nom d'utilisateur ou mot de passe saisi est erroné, le système ne donne pas le Jwt Token.

Une fois les bonnes informations sont saisies, l'utilisateur aura le Token et les autorisations selon son Rôle.

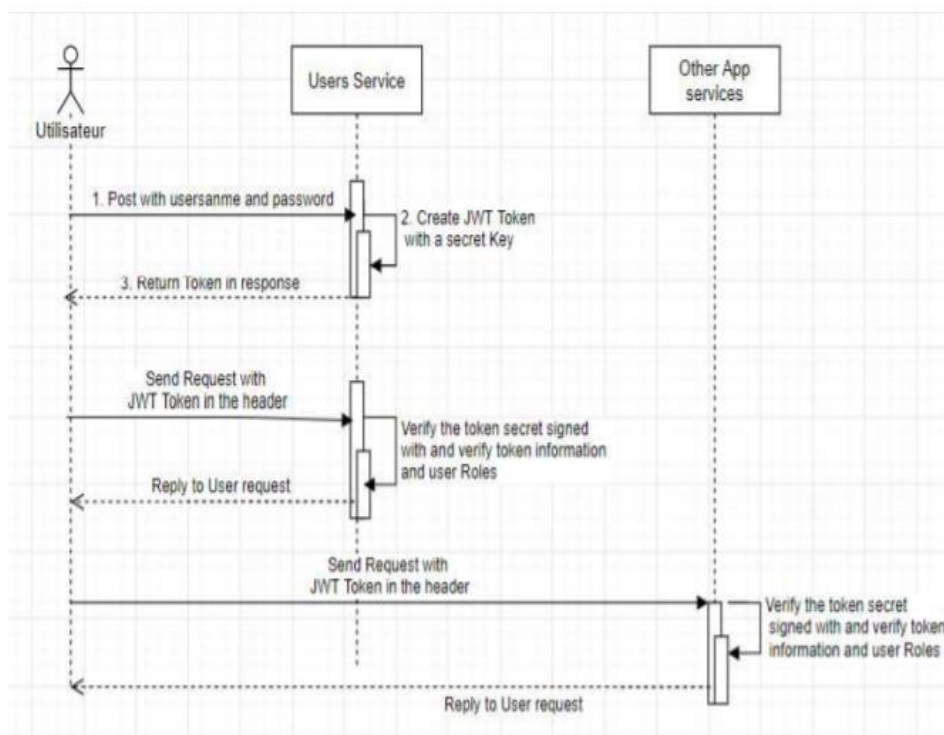


Figure 39 Diagramme de séquence du cas « Authentification »

Diagramme de séquence du cas « Ajouter un produit au panier » :

Ce diagramme de séquence décrit l'interaction du client avec le catalogue des produit. Le client peut aussi ajouter un avis, et par la suite ajouter le produit au panier, mais cette dernière opération nécessite l'authentification.

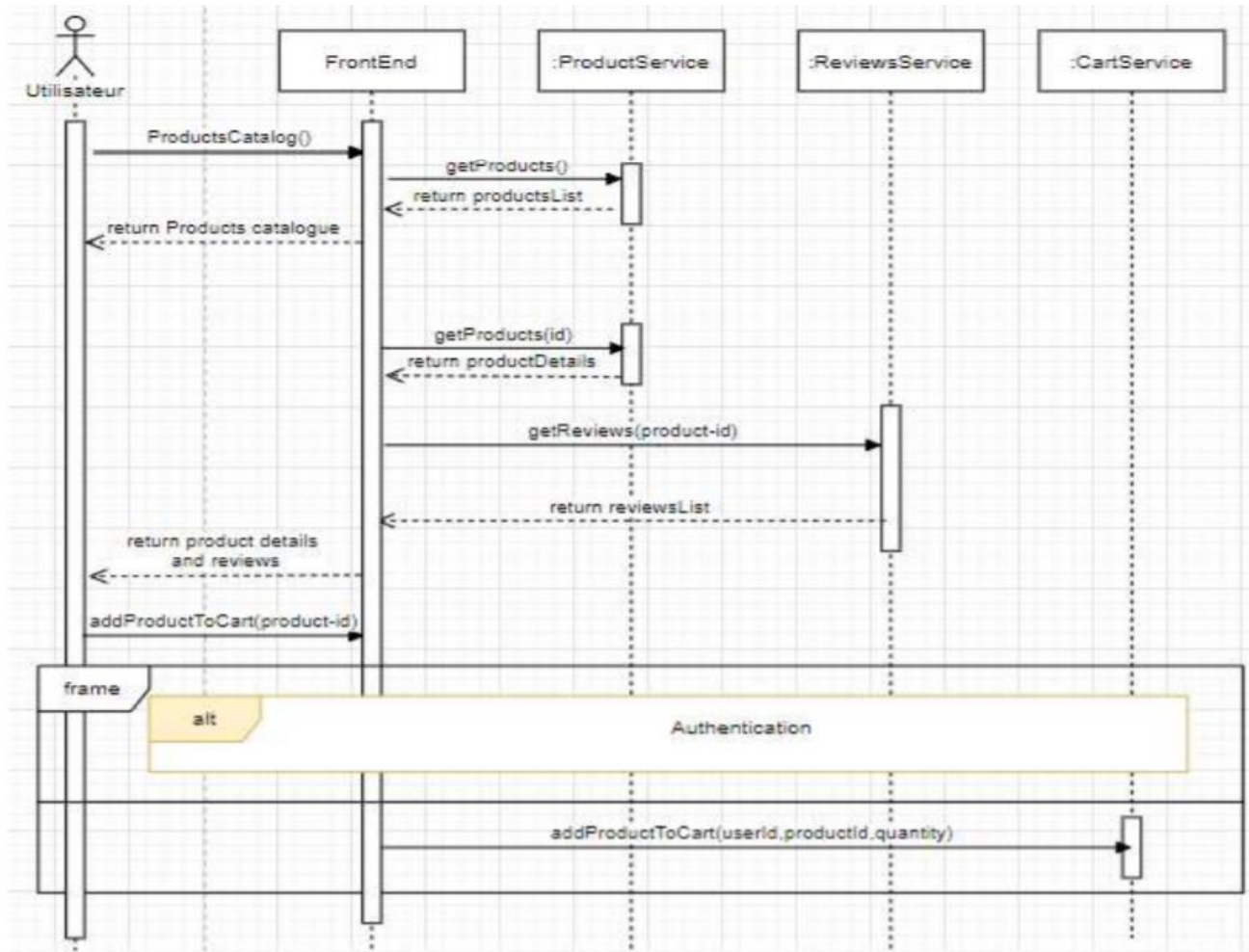


Figure 40 Diagramme de séquence du cas « Ajouter un produit au panier »

Diagramme de séquence du cas « checkout » :

Ce diagramme explique les étapes de la caisse, après l'authentification, le client entre les données de la carte bancaire, l'adresse de livraison, le Frontend envoie ces données et les données d'utilisateur comme email, et son identificateur au service « Checkout », ce service récupère les ordres temporaires en communiquant avec le service « Cart », le service « Payment » essaye de payer les ordres, si le paiement est accepté, le service « Checkout » va notifier le client par email de son paiement, il va persister les ordres du service « Cart » au service « Ordre » et il supprime les données du panier « Cart ». Il envoie l'ordre au service « Shipment » pour livrer l'ordre, le service « Shipping » crée une nouvelle livraison et il va notifier le client par email de l'Id de sa livraison.

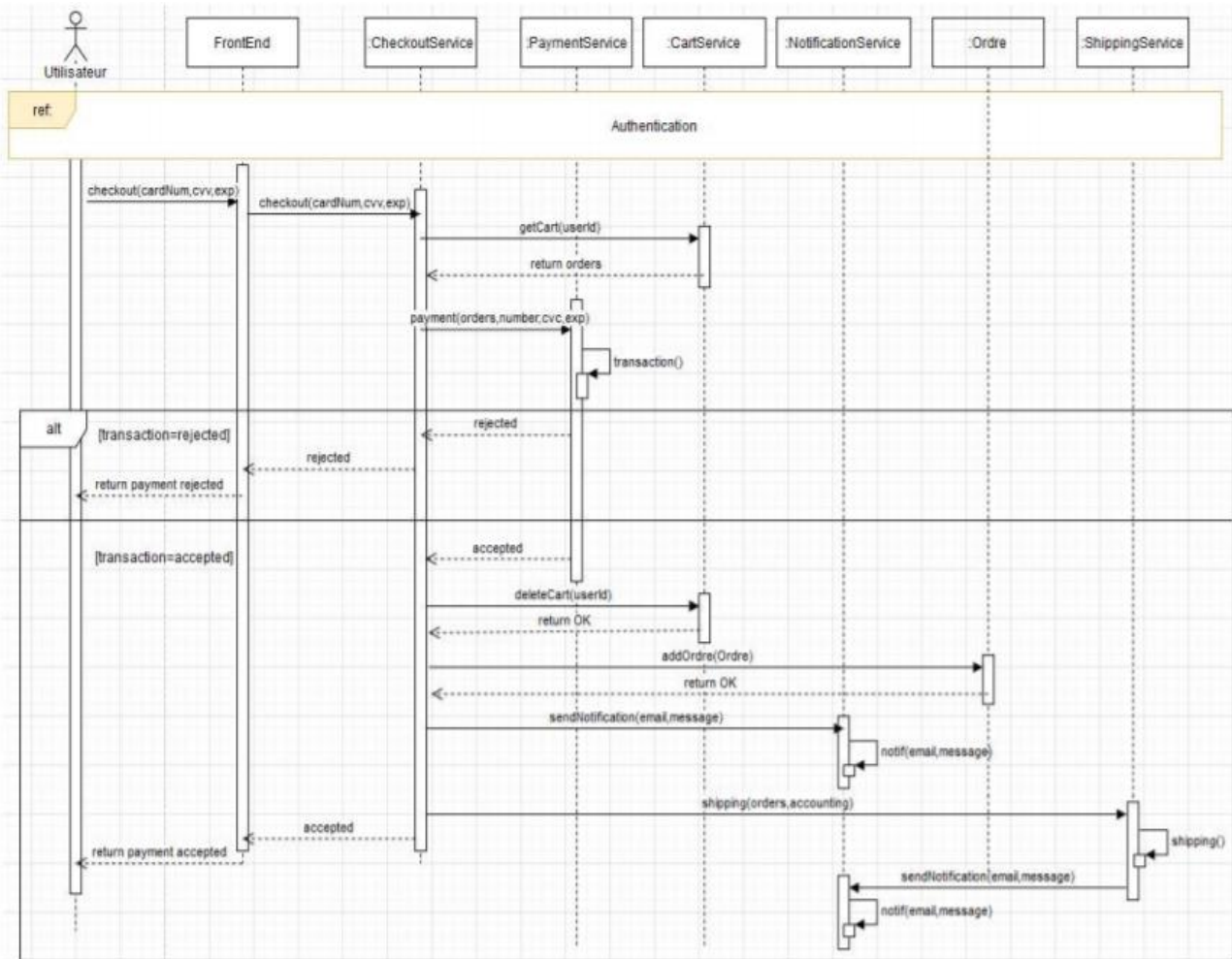


Figure 41 Diagramme de séquence du cas « checkout »

4.2.4 Diagramme de package

Les Diagrammes de Paquetages sont utilisés pour refléter l'organisation de paquets et de leurs éléments. Lorsqu'il est utilisé pour représenter des éléments de classe, diagrammes de paquets permettent de visualiser les espaces de noms. L'utilisation la plus courante pour diagrammes de paquets est d'organiser des Diagrammes de Cas d'Utilisation et des Diagrammes de Classes. Bien que l'Utilisation des Diagrammes de Paquetages ne se limite pas à ces éléments UML. [22]

Diagramme de package du micro-service « Users-service » :

Le Microservice « Users » se compose de quatre paquets principaux, et chacun d'entre eux se compose soit des class métiers seuls, ou bien elle s'organise sous forme des sous packages, ce diagramme est présenté par la figure ci-dessous.

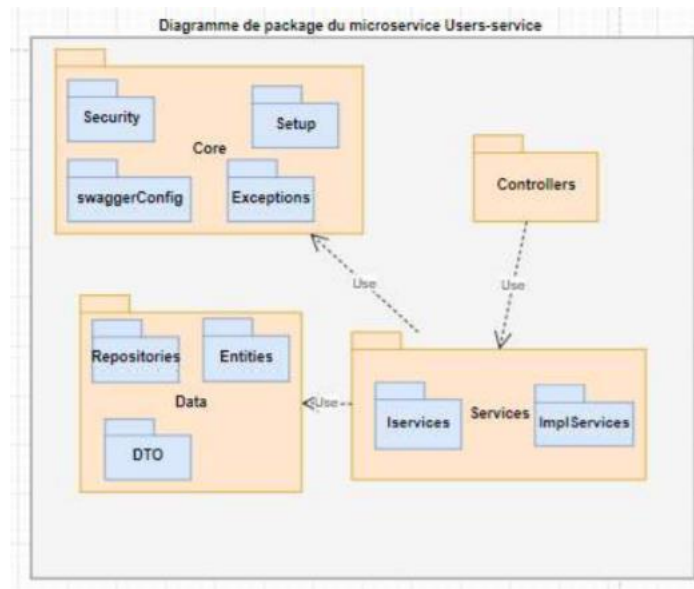


Figure 42 Diagramme de package du microservice Users-service

Ce diagramme décrit les package du Microservice « Users », il est constitué de quatre paquetages principaux :

- **Core** : Ce paquetage se compose de tous ce qui est partagé entre tous les autres packages : configuration de Swagger, les exceptions, la sécurité, initialisation des données.
- **Data** : Ce package contient :
Entities : les entités du Microservice.
Repositories : consistent à rendre la création de la couche d'accès aux données plus rapides.
DTO : définit les objets qui définissent la structure des informations à échanger, généralement, Réalisable.
- **Services** : Il contient les classes services. Ces classes constituent le traitement métier du Microservice.
- **Controllers** : Il s'agit d'un Controller qui réalise les Endpoints REST API du Microservice.

Diagramme de package du micro-service « checkout » :

Le diagramme de packages du Microservice « Checkout » est présenté par la figure ci-dessous.

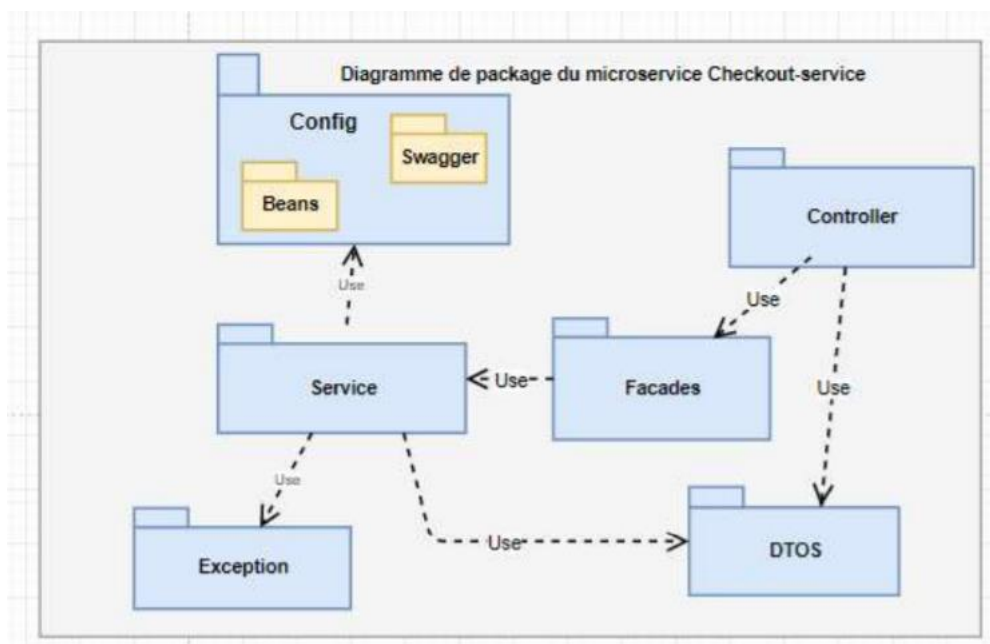


Figure 43 Diagramme de packages du Microservice Checkout

4.2.5 Diagramme de classes

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe. [23]

La figure ci-dessous présente la structure en classe du Microservice ProductCatalog.

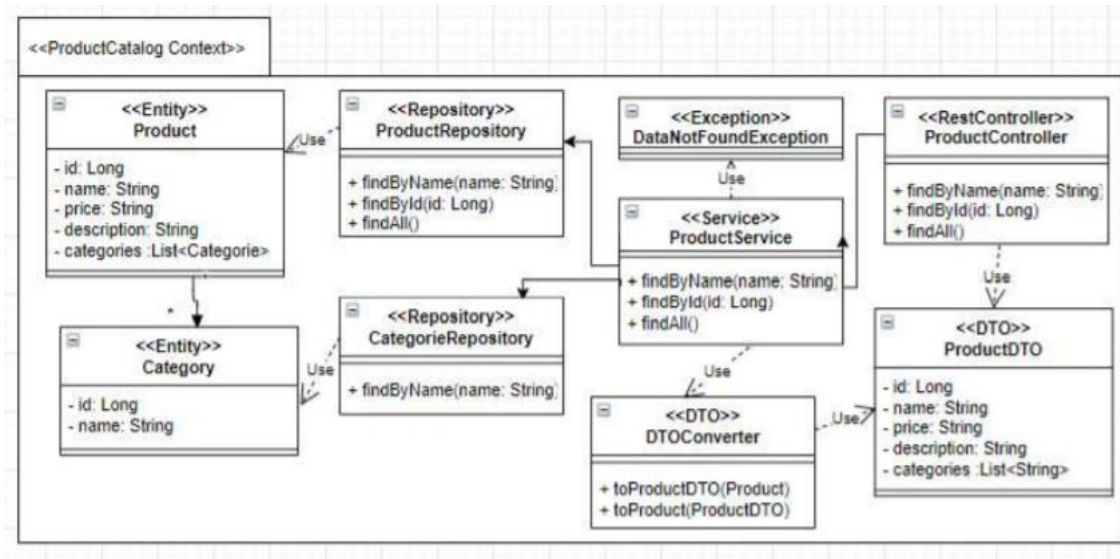


Figure 44 Diagramme de class du Microservice ProductCatalog

La figure ci-dessous présente la structure en classe du Microservice « Cart ».

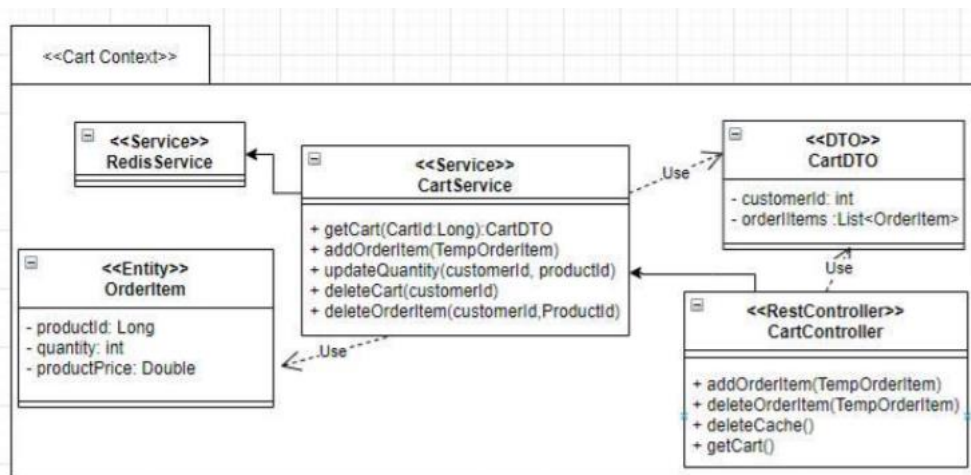


Figure 45 Diagramme de class du Microservice Cart

4.3 La documentation de l'API

4.3.1 Swagger

Swagger est essentiellement un langage de description d'interface pour décrire les API RESTful exprimées à l'aide de JSON. Swagger est utilisé avec un ensemble d'outils logiciels open source pour concevoir, créer, documenter et utiliser les services Web RESTful. [24]



Figure 46 Swagger logo

4.3.2 La documentation swagger

- La documentation swagger « Users-service »

Nous exposons l'interface Swagger liée au Microservice Users. La figure ci-dessous présente l'interface de la documentation Swagger :

Authentication Management REST API

Created by Lhoussaine ouarhou
[Contact the developer](#)
 Apache 2.0

account-role-controller : Account Role Controller		Show/Hide List Operations Expand Operations
GET	/roles	Get All AccountRoles
POST	/roles	Add new AccountRole
GET	/roles/search	Get AccountRole using the RoleName
DELETE	/roles/{id}	Delete AccountRole using the id.
GET	/roles/{id}	Get AccountRole using the id.
PATCH	/roles/{id}	Update AccountRole using the id.

account-user-controller : Account User Controller		Show/Hide List Operations Expand Operations
GET	/users	Get All AccountUsers
POST	/users	add new AccountUser
GET	/users/search/	get AccountUser by username
DELETE	/users/{userId}	Delete AccountUser using the id.
GET	/users/{userId}	get AccountUser by userId
PATCH	/users/{userId}	Update AccountUser using the id.
GET	/users/{userId}/roles	Add new AccountRole to AccountUser using the userId.

auth-controller : Auth Controller		Show/Hide List Operations Expand Operations
POST	/authenticate	createAuthenticationToken

basic-error-controller : Basic Error Controller		Show/Hide List Operations Expand Operations

jwk-set-endpoint : Jwk Set Endpoint		Show/Hide List Operations Expand Operations
GET	/keyInfos/.well-known/jwks.json	get jwks information
GET	/keyInfos/public/key	get public key information

Figure 47 La documentation swagger du Microservice Users

Ce Microservice est composé de trois ressources principales :

Auth-controller : présente une seule méthode qui nous permet de s'authentifier avec un nom d'utilisateur et mot de passe, et nous créons un jeton après la vérification de la validité des données.

Account-user-controller : présente les méthodes permettant à un administrateur de gérer les comptes utilisateur et un client de créer un compte.

Account-role-controller : présente les méthodes permettant à un administrateur de gérer les rôles associés à chaque utilisateur.

- **La documentation swagger « ProductsCatalog-service »**

Nous exposons l'interface Swagger liée au Microservice ProductCatalog. La figure ci-dessous présente l'interface de la documentation swagger :

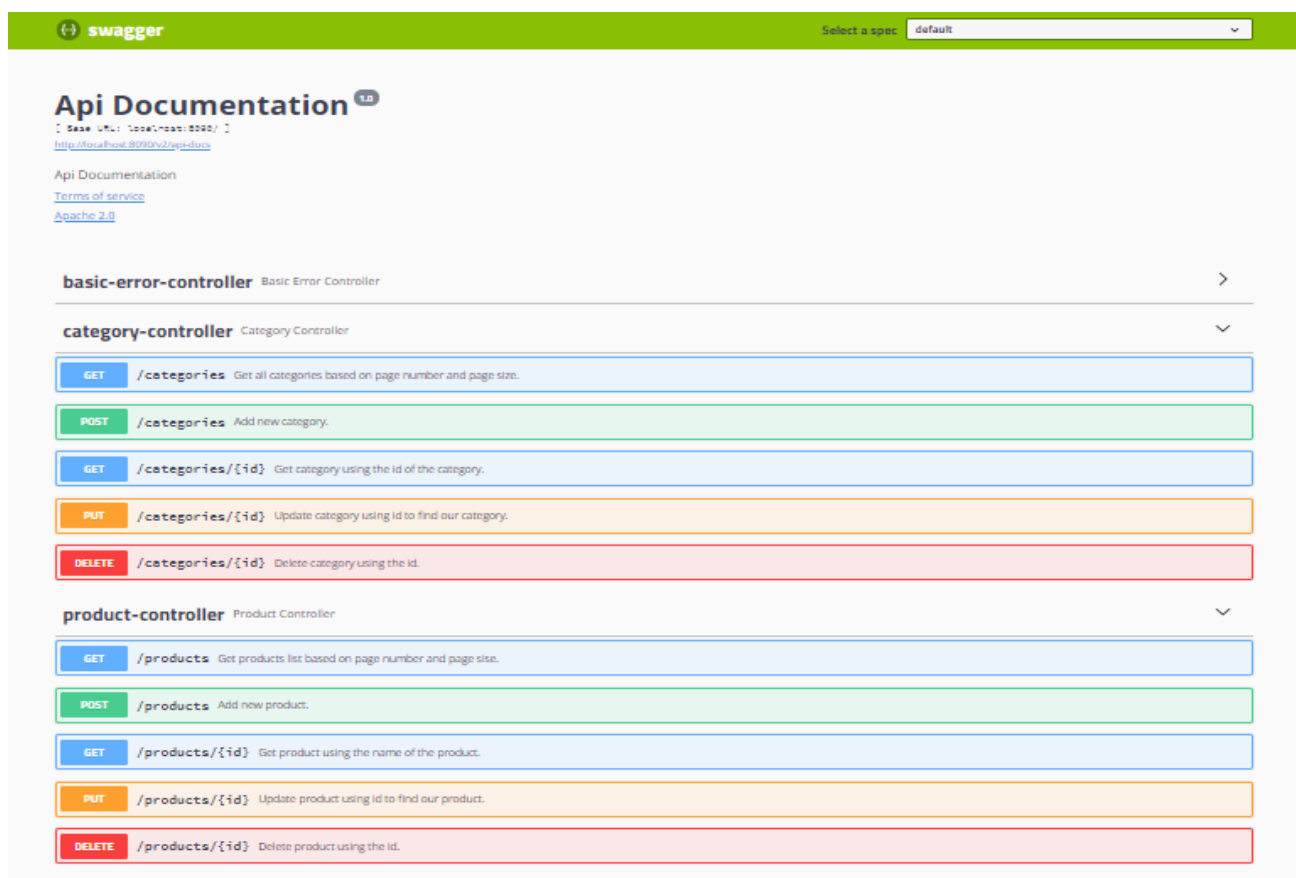


Figure 48 La documentation Swagger du Microservice ProductsCatalogue

Category-controller : présente les méthodes permettant à un agent de gérer les catégories des produits.

Product-controller : présente les méthodes permettant à un agent de gérer les produits.

- **La documentation swagger « Reviews-service »**

Nous exposons l'interface Swagger liée au Microservice Reviews-service. La figure ci-dessous présente l'interface de la documentation swagger :

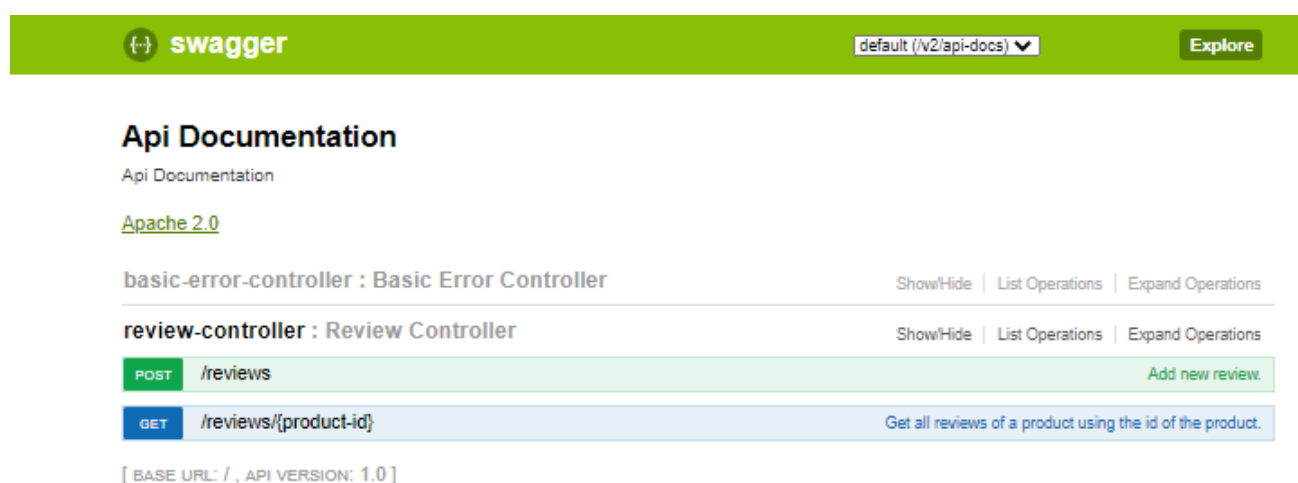


Figure 49 La documentation Swagger du Microservice Reviews

Review-controller : présente les méthodes permettant à un utilisateur de donner un avis et consulter les avis concernant un produit.

4.4 Conclusion

Dans ce chapitre nous avons essayé de donner vue globale des Microservices utilitaires principaux ; pour cela nous avons détaillé la conception, en suite nous avons présenté la documentation swagger de notre API.

Chapitre 5

L'implémentation

5.1 Introduction

Après la spécification des besoins et la conception de notre application, durant ce chapitre nous allons présenter la phase de l'implémentation. Premièrement nous allons situer le projet dans son cadre technique ; à savoir les différents technologies et outils utilisés durant toutes les phases du projet, en suite la réalisation et le déploiement dans un cluster Kubernetes, et par la suite injecter le service mesh Istio.

5.2 Technologies et outils utilisés

5.2.1 Outil de collaboration GitLab



C'est une plateforme de développement collaborative qui couvre l'ensemble des étapes du DevOps. Se basant sur les fonctionnalités du logiciel Git, elle permet de réaliser des dépôts et de gérer les versions de vos codes sources. Son usage est particulièrement indiqué pour les développeurs qui souhaitent disposer d'un outil réactif et accessible. [25]

5.2.2 Environnement de déploiement

- **Docker**



Docker est un logiciel libre Open Source qui permet d'automatiser le déploiement d'applications. Il a été développé par Solomon Hykes de la société dotCloud et a été distribué à partir de mars 2013. C'est une plateforme de virtualisation par conteneur qui va permettre de concevoir, tester et déployer des applications rapidement. Grâce à Docker, il est facile de déployer et dimensionner les applications dans n'importe quel environnement en s'assurant que le code s'exécutera automatiquement. [26]

- **Kubernetes**

**kubernetes**

Kubernetes (communément appelé « K8s ») est un système open source qui vise à fournir une « plate-forme permettant d'automatiser le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs ». Il fonctionne avec toute une série de technologies de conteneurisation, et est souvent utilisé avec Docker. Il a été conçu à l'origine par Google, puis offert à la Cloud Native Computing Foundation. [27]

- **Azure Kubernetes Service (AKS)**



Azure Kubernetes Service

Azure Kubernetes Service (AKS) est un service géré d'orchestration de conteneurs, basé sur le système open source Kubernetes, qui est disponible sur le nuage public Azure de Microsoft. Une organisation peut utiliser AKS pour déployer, mettre à l'échelle et gérer des conteneurs Docker et des applications basées sur des conteneurs dans un groupe d'hôtes de conteneurs. [28]

5.2.3 Technologie de développement

- **Spring boot**



Publiée en 2012, Spring Boot est une solution de « convention plutôt que configuration » destinée à l'infrastructure logicielle Java Spring qui réduit la complexité de la configuration de nouveaux projets Spring. À cette fin, Spring Boot définit une configuration de base incluant des directives pour l'utilisation de l'infrastructure logicielle ainsi que toutes les bibliothèques de prestataires tiers pertinentes, ce qui permet de faciliter autant que possible la création de nouveaux projets. Cette méthode simplifie considérablement la création d'applications indépendantes prêtes pour la production, ce qui explique pourquoi la majorité des nouvelles applications Spring reposent en grande partie sur Spring Boot.

Les caractéristiques de Spring Boot peuvent être résumées comme suit :

- l'intégration directe d'applications de serveur Web/de conteneur comme Apache Tomcat ou Jetty sans utiliser de fichiers WAR (Web Application Archive)
- la configuration simplifiée de Maven grâce à des POM (Project Object Models) « Starter »
- lorsque c'est possible, la configuration automatique de Spring
- la mise à disposition de capacités non fonctionnelles telles que des outils de mesure ou des configurations délocalisées []

5.3 Le déploiement des microservices

5.3.1 La containerisation

Tout d'abord, pour mettre nos Microservices dans des conteneurs nous avons besoin évidemment d'avoir docker installée sur la machine ; par la suite nous devrions créer un fichier appelé Dockerfile, ce fichier contient les consignes et les commandes à exécuter afin de monter notre image docker et lancer le Microservice.

Ci-dessous l'exemple de Dockerfile pour le Microservice « Users » :

```
FROM openjdk:8-jdk-alpine
VOLUME /users
ADD /target/users-service-0.0.1-SNAPSHOT.jar /users/
EXPOSE 8091
CMD java -jar /users/users-service-0.0.1-SNAPSHOT.jar --spring.application.name=${SPRING_APPLICATION_NAME}
```

Figure 50 Dockerfile

- Chaque Docker file commence par l'instruction "FROM". Cette instruction spécifie l'image de base à partir de laquelle nous construisons notre image.
- Dans notre exemple nous choisissons une distribution alpine qui contient Java JDK pour notre Microservice. - Nous créons le dossier racine de notre 'Microservice'.
- Nous copions tous les fichiers nécessaires à la construction du projet vers ce dossier.
- On expose les ports afin de libérer l'accès à l'hôte.
- Et finalement on fournit à docker la commande clé qui lui permettra de lancer notre projet. En Utilisant cette commande « **docker build -t username/imageName:tagversion app_root_path** » dans la racine du projet on crée notre image de notre Microservice.

5.3.2 Structure et organisation

Pour bien organiser le déploiement de l'application et isoler les configurations d'un Microservice à celles des autres, nous avons créé pour chaque Microservice un espace de nom « namespace » dédié à lui qui doit contenir toutes les configurations de ce service.

Un autre espace de nom qui va contenir les configurations dédiées à toute l'application comme le Gateway.

Tant que nous sommes dans l'environnement de développement, les espaces des noms seront préfixés par « dev ».

En utilisant la commande « `kubectl get namespaces` », on peut consulter les espaces de noms « namespaces » dans notre cluster.

NAME	STATUS	AGE
default	Active	4d17h
dev-cart	Active	3d11h
dev-checkout	Active	14s
dev-configs	Active	21m
dev-notification	Active	3d17h
dev-ordres	Active	15s
dev-payment	Active	41h
dev-productcatalog	Active	4d1h
dev-rabbitmq-broker	Active	3d21h
dev-review	Active	4d1h
dev-shipping	Active	3d14h
dev-users	Active	9s
istio-system	Active	3d22h
kube-node-lease	Active	4d17h
kube-public	Active	4d17h
kube-system	Active	4d17h
kubernetes-dashboard	Active	3d21h

Figure 51 Les espaces de nom du cluster Kubernetes

La figure ci-dessus montre que pour chaque service on a créé une espace de nom. Pour l'espace de nom « istio-system », contient le gateway ingress qui va être utilisé avec notre API gateway, et aussi « Grafana » et « Kiali » qui sont préconfiguré pour visualiser et surveiller notre application.

5.3.3 Le déploiement dans Kubernetes

Le déploiement de nos Microservices consiste à déployer les besoins de chaque service. Après la ‘dockerisation’ de nos Microservices en utilisant Docker, nous avons besoin de les déployer dans kubernetes, chaque application sera encapsulée dans un pod de kubernetes pour qu’il soit une unité d’exécution d’un Microservice.

- Exemple : Service ProductCatalog

Ce service utilise une base de données MySQL, donc nous avons besoin d’installer le pod pour le Microservice, et le pod de la base de données MySQL pour avoir de l’indépendance entre les Microservices.

NAME	READY	STATUS	RESTARTS
mysql-767856cf45-rxtgv	2/2	Running	0
productcatalog-67d4d9fff8-hvhn9	2/2	Running	0
productcatalog-67d4d9fff8-kfx9k	2/2	Running	0
productcatalog-67d4d9fff8-tfscw	2/2	Running	0

Figure 52 Déploiement de Microservice product-catalog

Notre application contient trois répliques à l'objectif si une réplique est tombée en panne, l'application continue de fonctionner.

Chaque « pod » contient deux conteneurs, le premier est le conteneur docker de notre application, la deuxième est le proxy « Sidecar » injecté par Istio.

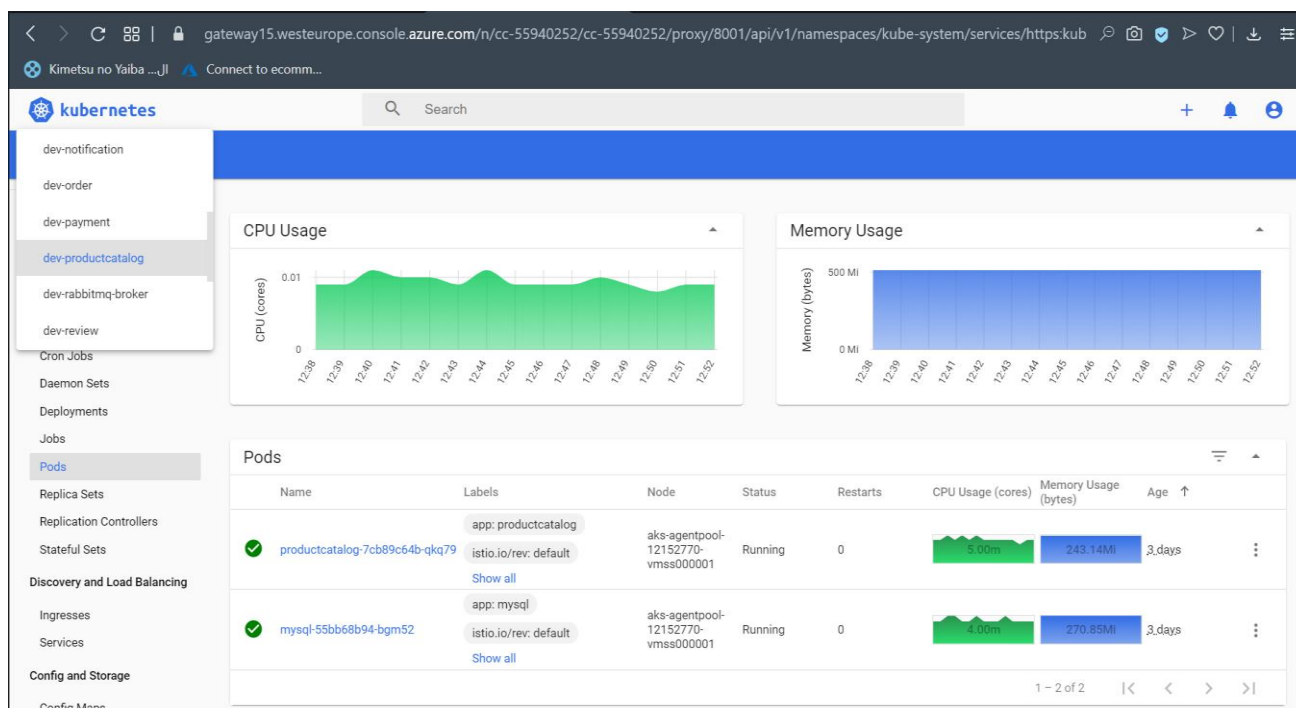
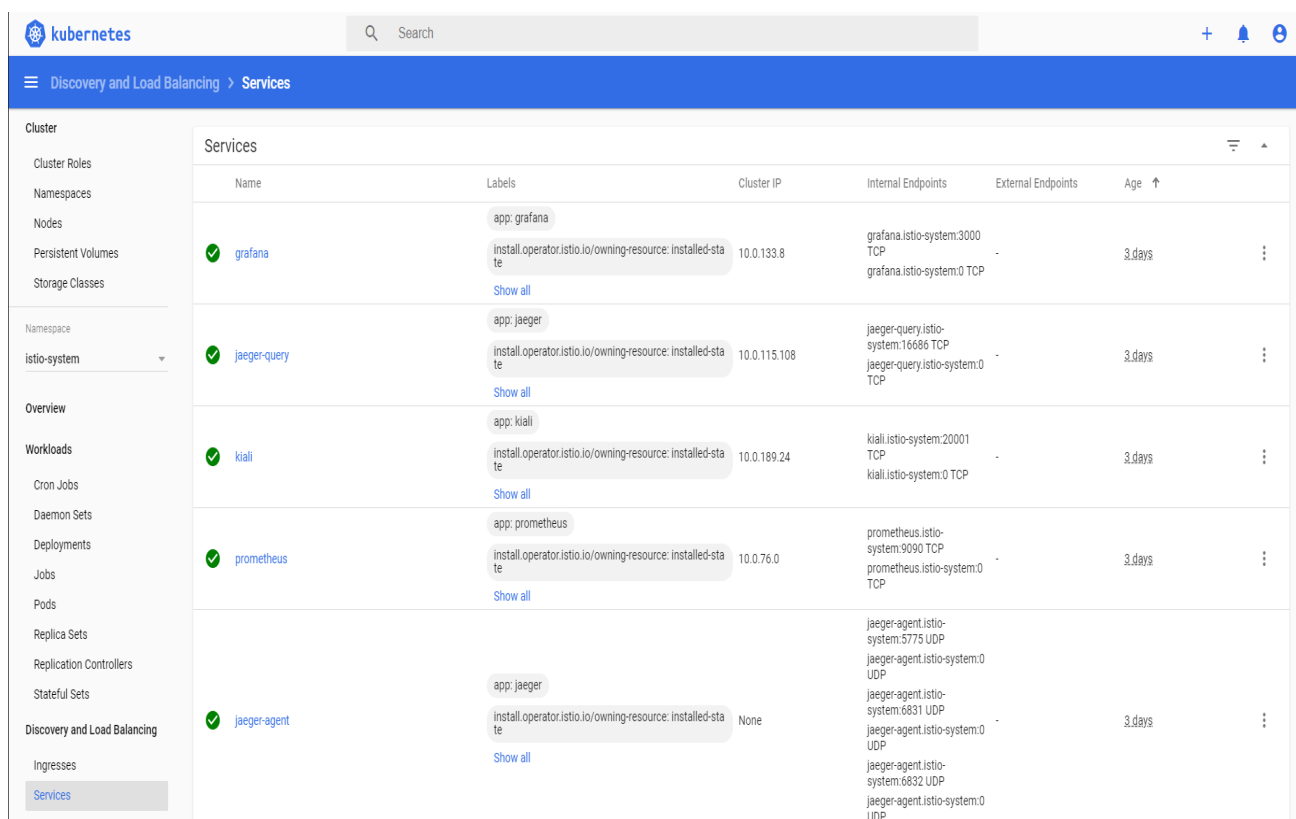
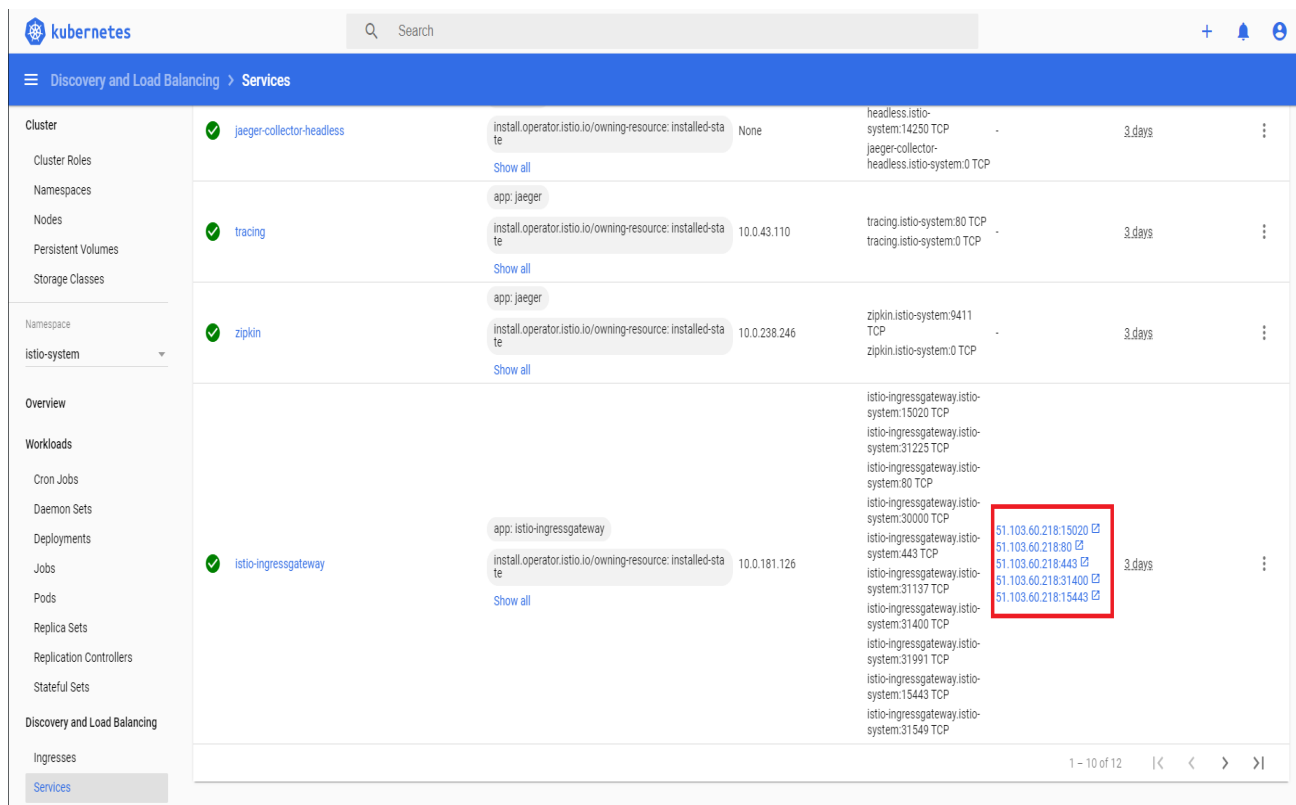


Figure 53 deployment du productCatalog dans AKS

• L'injection d'Istio





Cluster	Service	Install Operator	IP Address	Ports	Duration
jaeger-collector-headless	jaeger-collector-headless	install.operator.istio.io/owning-resource: installed-state	None	headless.istio-system:14250 TCP jaeger-collector-headless.istio-system:0 TCP	3.days
tracing	tracing	install.operator.istio.io/owning-resource: installed-state	10.0.43.110	tracing.istio-system:80 TCP tracing.istio-system:0 TCP	3.days
zipkin	zipkin	install.operator.istio.io/owning-resource: installed-state	10.0.238.246	zipkin.istio-system:9411 TCP zipkin.istio-system:0 TCP	3.days
istio-ingressgateway	istio-ingressgateway	install.operator.istio.io/owning-resource: installed-state	10.0.181.126	istio-ingressgateway.istio-system:15020 TCP istio-ingressgateway.istio-system:31225 TCP istio-ingressgateway.istio-system:80 TCP istio-ingressgateway.istio-system:30000 TCP istio-ingressgateway.istio-system:443 TCP istio-ingressgateway.istio-system:31137 TCP istio-ingressgateway.istio-system:1400 TCP istio-ingressgateway.istio-system:31991 TCP istio-ingressgateway.istio-system:15443 TCP istio-ingressgateway.istio-system:31549 TCP	3.days

Figure 54 L'injection du service mesh Istio

La figure ci-dessus montre l'injection du service mesh istio au niveau de AKS, et les services préconfigurés qu'offre Istio.

On remarque aussi l'adresse IP donnée par le contrôleur d'istio.

5.3.4 Management du trafic

5.3.4.1 Introduction

Les règles de routage du trafic d'Istio nous permettent de contrôler facilement le flux de trafic et les appels d'API entre les services. Istio simplifie la configuration des propriétés de niveau de service comme le « Circuit breaker », les délais d'expiration et les tentatives « Retry & Timeout », et facilite la configuration de tâches importantes comme les tests A / B, les déploiements canaries et les déploiements par étapes avec des répartitions de trafic basées sur un pourcentage. Il fournit également des fonctionnalités de récupération après panne qui aident à rendre votre application plus robuste contre les défaillances des services dépendants ou du réseau. [30]

5.3.4.2 Passerelle API « Gateway »

La passerelle consiste que l'application fonctionne et accessible à l'intérieur de notre cluster, dans ce cas, la passerelle va nous donner la possibilité d'être accessible via l'extérieur.

Pour le faire, nous avons utilisé le contrôleur d'Istio au but de nous donner une adresse IP, et on a donné un hôte pour chaque service. C'est ce que montre la figure suivante :

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
istio-ingressgateway	LoadBalancer	10.0.181.126	51.103.60.218	15020:31225/TCP,80:30000/TCP,443:31137/TCP,31400:31991/TCP,15443:31549/TCP

NAME	GATEWAYS	HOSTS	AGE
virtual-service-cart	[store-gateway]	[cart.sqli.com]	15h
virtual-service-checkout	[store-gateway]	[checkout.sqli.com]	15h
virtual-service-order	[store-gateway]	[order.sqli.com]	15h
virtual-service-payment	[store-gateway]	[payment.sqli.com]	15h
virtual-service-product	[store-gateway]	[products.sqli.com]	15h
virtual-service-reviews	[store-gateway]	[reviews.sqli.com]	15h
virtual-service-shipping	[store-gateway]	[shipping.sqli.com]	15h
virtual-service-user	[store-gateway]	[user.sqli.com]	15h

Figure 55 API Gateway

Maintenance notre application est accessible à l'extérieur du cluster, utilisant Postman, nous pouvons tester l'accessibilité :

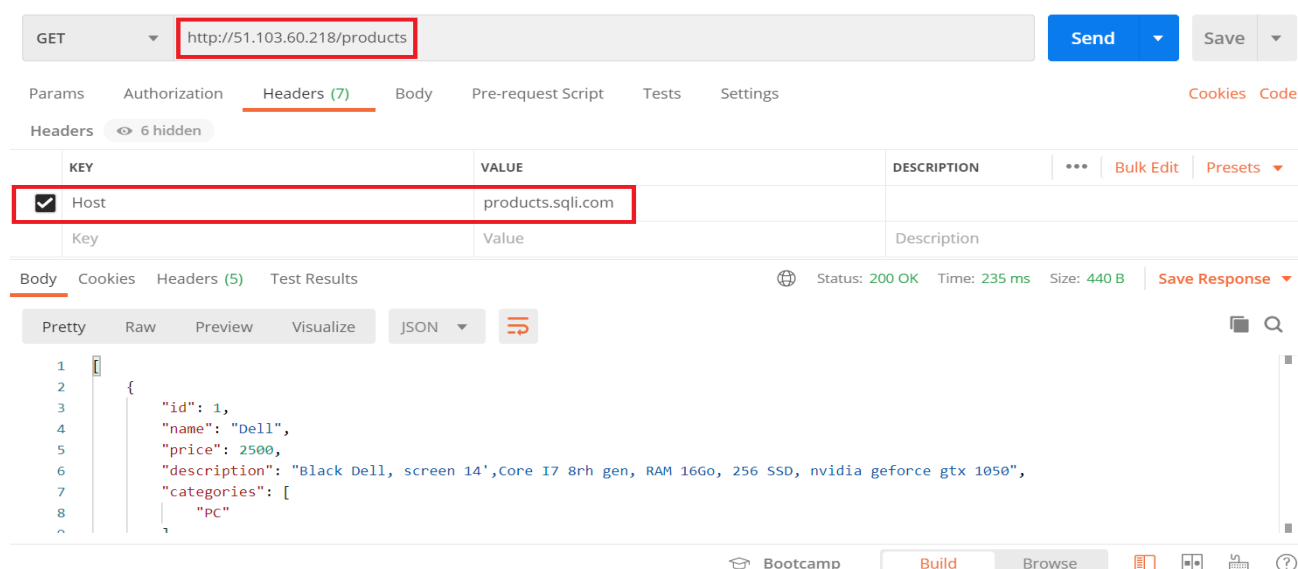


Figure 56 Résultat de l'API Gateway et l'Host

5.3.4.4 nouvelles tentatives et de délai d'expiration « Retry & Timeout »

« Retry » et le « Timeout » sont déjà définis dans le chapitre 3 d'Etude et Benchmarking des services Mesh, dans cette partie nous les configurons.

Pour chaque service nous avons configuré qu'après 5 secondes, si le serveur n'a pas répondu, nous renvoyons l'envoi de la requête (3 tentatives possibles) et le délai d'attente « Timeout » sera 20 secondes.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-service-order
  namespace: dev-configs
spec:
  hosts:
    - 'order.sqli.com'
  gateways:
    - store-gateway
  http:
    - route:
        - destination:
            port:
              number: 8095
            host: order.dev-order.svc.cluster.local
          timeout: 20s
          retries:
            attempts: 3
            perTryTimeout: 5s
```

Figure 59 Retry & Timeout configuration

5.3.5 Sécurité

5.3.5.1 Introduction

La décomposition d'une application monolithique en services atomiques offre divers avantages, notamment une meilleure agilité, une meilleure évolutivité et une meilleure capacité de réutilisation des services. Cependant, les Microservices ont également des besoins de sécurités particulières :

Pour se défendre contre les attaques « Man in the middle », ils ont besoin d'un chiffrement du trafic.

Pour fournir un contrôle d'accès aux services flexibles, ils ont besoin d'un TLS mutuel et de politiques d'accès.

Pour déterminer qui a fait quoi, à quel moment, ils ont besoin d'outils d'audit.

La sécurité d'Istio fournit une solution de sécurité complète pour résoudre ces problèmes. En particulier, la sécurité Istio atténue les menaces internes et externes contre nos données, nos « endpoints », nos communications. [31]

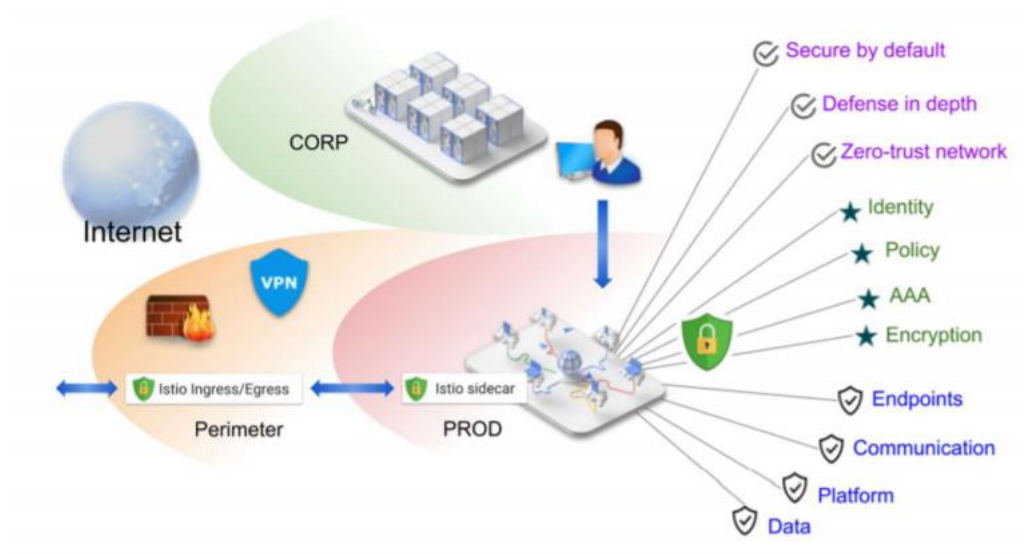


Figure 60 La sécurité d'Istio [31]

5.3.5.2 TLS mutuelle « mTLS »

Istio configure automatiquement les sidecars pour utiliser le TLS mutuel lors de l'appel d'autres charges de travail. Par défaut, Istio configure les charges de travail de destination en utilisant le mode PERMISSIVE. Lorsque le mode PERMISSIVE est activé, un service peut accepter à la fois du « plain text » et du trafic TLS mutuel. Afin de n'autoriser que le trafic TLS mutuel, la configuration doit être changée en mode STRICT.

Istio ne peut pas agréger ses stratégies pour le trafic TLS mutuel sortant vers un service. Pour le faire nous avons besoin d'ajouter une règle de destination pour gérer ce comportement. [32]

La figure ci-dessous montre que la communication entre les services est sécurisée entre le service « user » et la base de données MySQL :

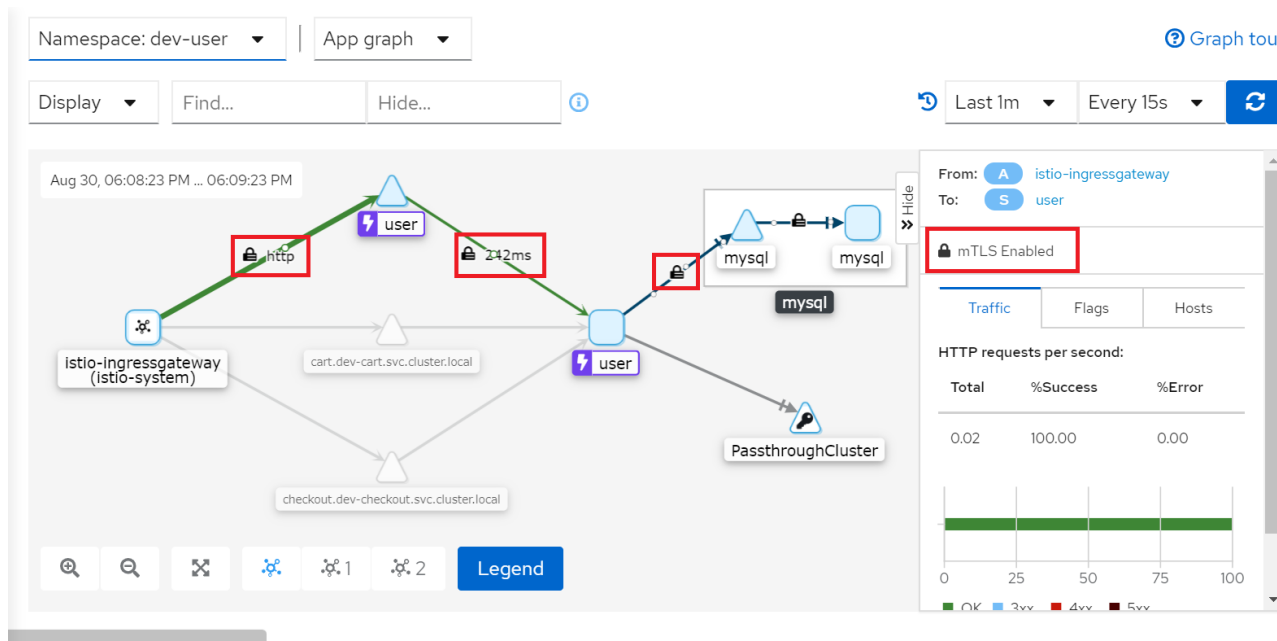


Figure 61 Kiali dashboard, mTLS est activé

5.3.5.3 Authentification des requêtes « request authentication »

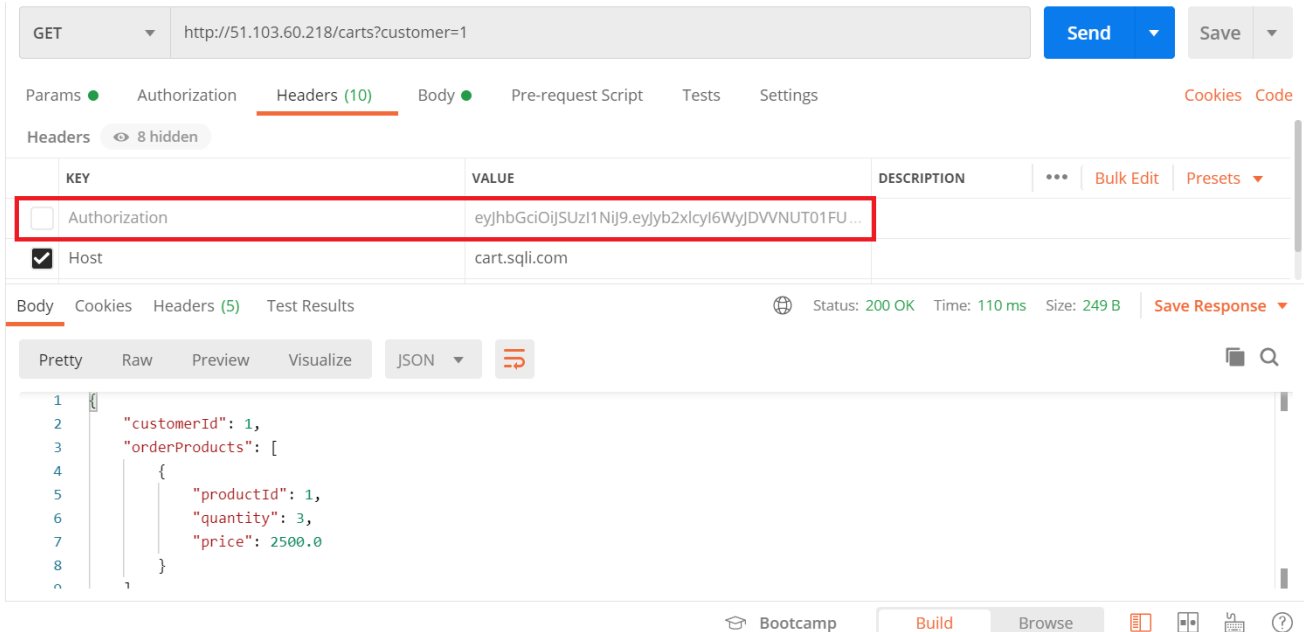
La configuration de l'authentification des requêtes permet de préciser les valeurs des paramètres nécessaires à la validation des jetons « tokens » reçu dans l'entête des requêtes envoyé à notre application. Ces valeurs-là inclus :

- La location du « token » dans une requête.
- L'émetteur 'issuer' du token.
- Le lien de la clé public présenté par le jwks.

Istio vérifie le token présenté dans une requête, s'il est validé en fonction des règles de la politique d'authentification des requêtes, et rejette les requêtes avec des tokens non valides. Lorsque les requêtes n'ont pas de token, elles sont acceptées par défaut. Pour les rejeter, ajouter des règles d'autorisation qui spécifient les restrictions pour des opérations spécifiques, par exemple des chemins ou des actions. Comme nous allons voir dans la partie qui suit.

On va tester les trois cas, pour bien montrer comment cette fonctionnalité se traite :

- **Cas 1** : l'envoi d'une requête sans spécifier la valeur du token; dans cas-là la requête est bien validé comme il est prévu.



GET ▼ http://51.103.60.218/carts?customer=1 Send Save ▼

Params ● Authorization Headers (10) Body ● Pre-request Script Tests Settings Cookies Code

Headers 8 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input type="checkbox"/>	Authorization	eyJhbGciOiJSUzI1NiJ9.eyJyb2xlcyI6WyJDNVNTUT01FU...				
<input checked="" type="checkbox"/>	Host	cart.sqli.com				

Body Cookies Headers (5) Test Results ⌐ Status: 200 OK Time: 110 ms Size: 249 B Save Response ▼

Pretty Raw Preview Visualize JSON ≡

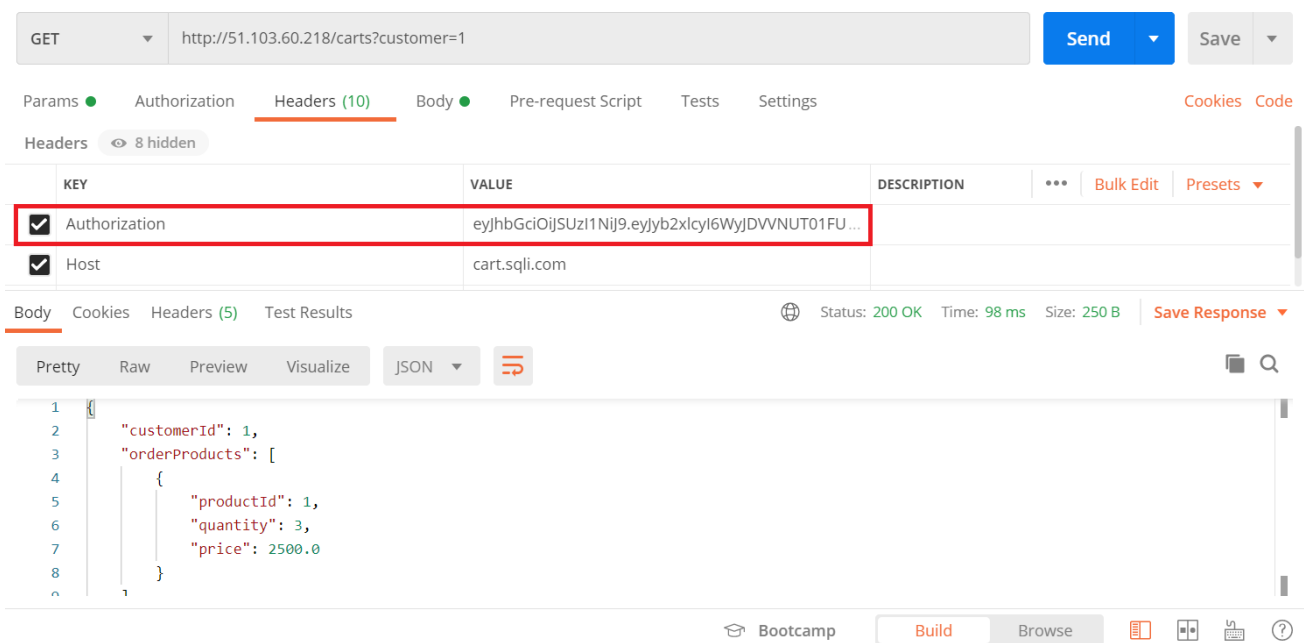
```

1 {
2   "customerId": 1,
3   "orderProducts": [
4     {
5       "productId": 1,
6       "quantity": 3,
7       "price": 2500.0
8     }
9   ]
10 }
  
```

Bootcamp Build Browse 📄 🔍 ?

Figure 62 envoi d'une requête sans token dans leur entête

- **Cas 2 :** L'envoi de la requête avec un token valide. La requête va être vérifié et valider. Comme il est montré dans la figure ci-dessous :



GET ▼ http://51.103.60.218/carts?customer=1 Send Save ▼

Params ● Authorization Headers (10) Body ● Pre-request Script Tests Settings Cookies Code

Headers 8 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	eyJhbGciOiJSUzI1NiJ9.eyJyb2xlcyI6WyJDNVNTUT01FU...				
<input checked="" type="checkbox"/>	Host	cart.sqli.com				

Body Cookies Headers (5) Test Results ⌐ Status: 200 OK Time: 98 ms Size: 250 B Save Response ▼

Pretty Raw Preview Visualize JSON ≡

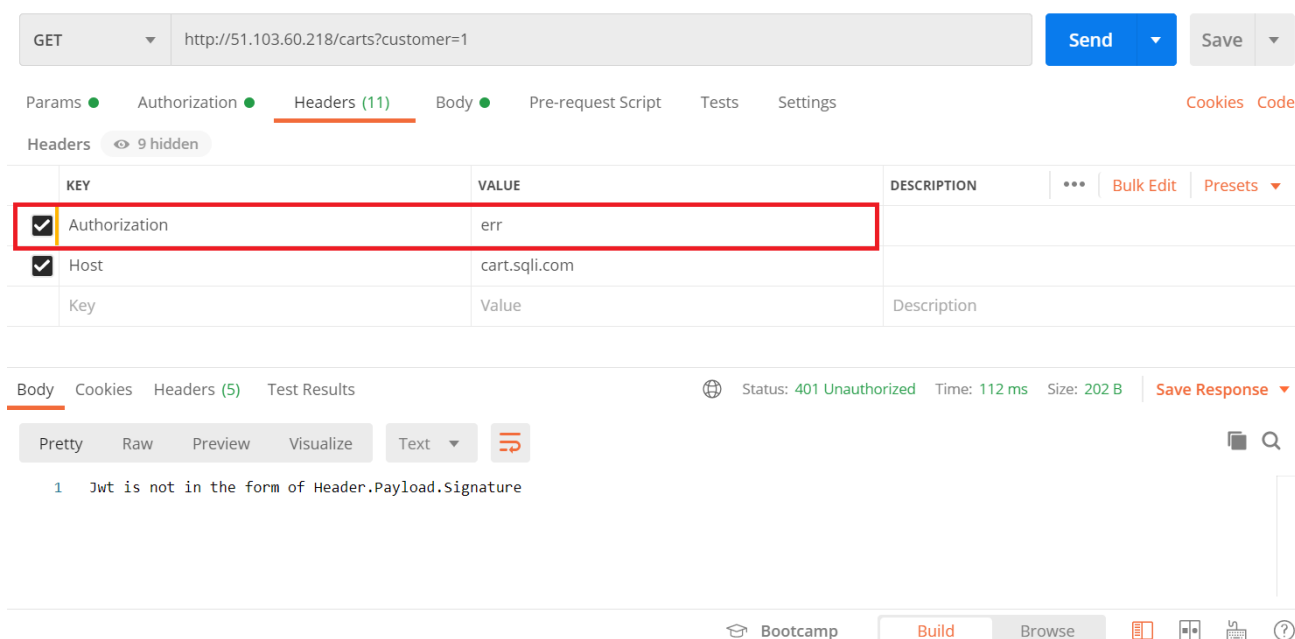
```

1 {
2   "customerId": 1,
3   "orderProducts": [
4     {
5       "productId": 1,
6       "quantity": 3,
7       "price": 2500.0
8     }
9   ]
10 }
  
```

Bootcamp Build Browse 📄 🔍 ?

Figure 63 L'envoi d'une requête avec un token valide

- **Cas 3 :** on va tester le dernier scenario, dans lequel on va envoyer la requête avec une valeur du token non valide.



GET http://51.103.60.218/carts?customer=1 Send Save

Params Authorization Headers (11) Body Pre-request Script Tests Settings Cookies Code

Headers 9 hidden

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	err				
<input checked="" type="checkbox"/> Host	cart.sqli.com				
Key	Value	Description			

Body Cookies Headers (5) Test Results Status: 401 Unauthorized Time: 112 ms Size: 202 B Save Response

Pretty Raw Preview Visualize Text

1 Jwt is not in the form of Header.Payload.Signature

Bootcamp Build Browse

Figure 64 L'envoi d'une requête avec un token non valide

5.3.6 Observabilité

5.3.6.1 Kiali dashboard

Kiali est un outil d'observation et de configuration de maillage de service, il répond aux questions : Quels Microservices font partie de mon maillage de service Istio ? Comment sont-ils connectés ? Comment fonctionnent-ils ?

Istio utilise Kiali pour la visualisation des services.

La figure ci-dessous montre que Kiali peut nous donner des informations sur la santé de nos services, comme dans cette figure, tous les services fonctionnent :

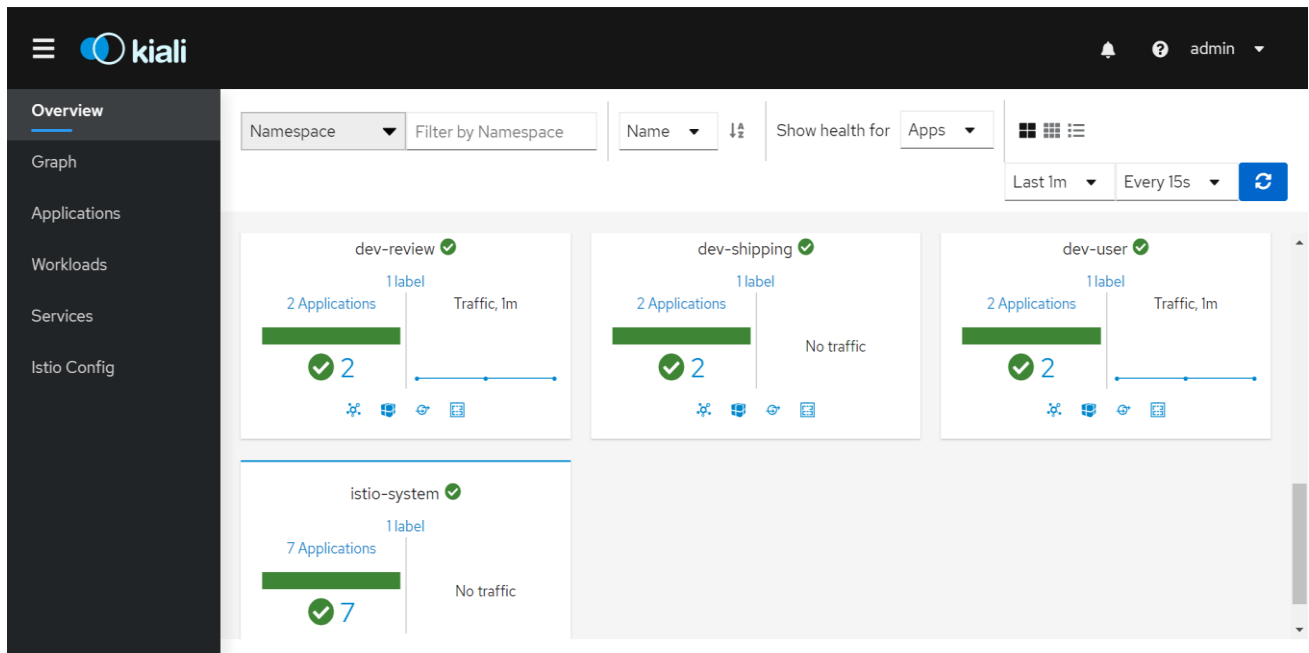


Figure 65 Kiali dashboard

Kiali peut aussi répondre aux questions suivantes :

- À quelle version du service, le trafic accède ?
- Est-ce que mTLS est activé ? comme nous avons vu dans la partie de mTLS.

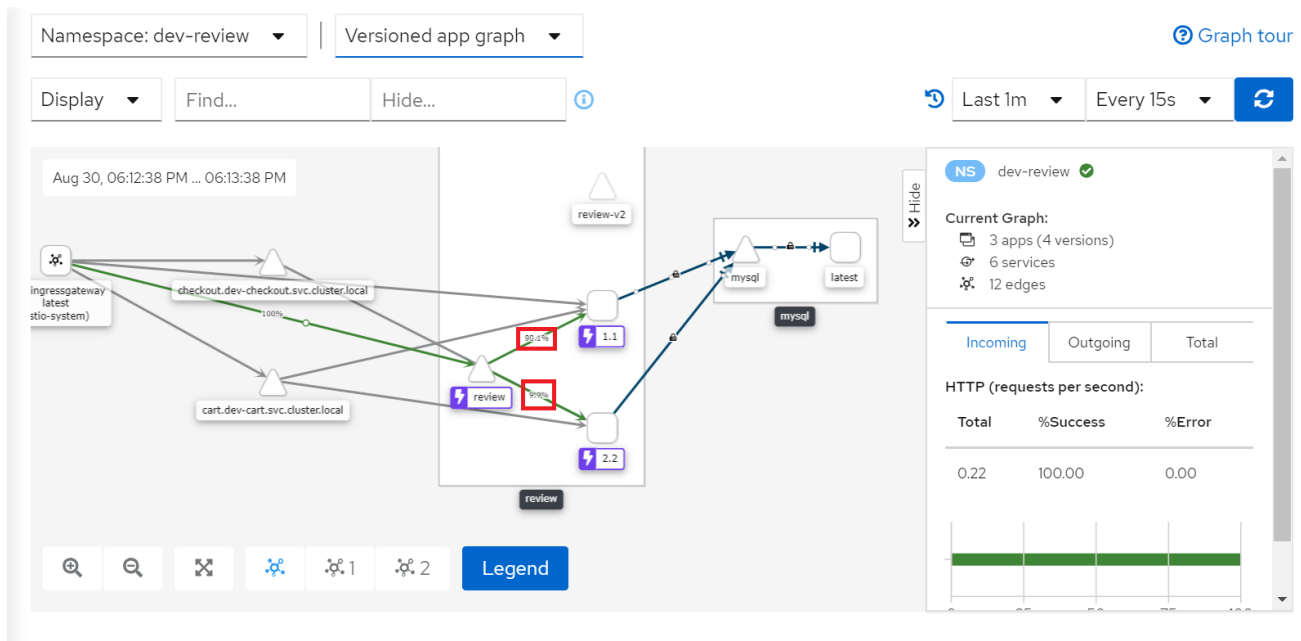


Figure 66 Reviews visualisation de Kiali

La figure ci-dessus nous informe que le trafic arrive de l'API Gateway, vers les deux versions de notre service Review, en plus de ça la communication est sécurisée avec la base de donnée MySQL.

5.3.6.2 Grafana dashboard

Grafana est une plateforme open source taillée pour la surveillance, l'analyse et la visualisation de métriques, livrée avec un serveur web permettant d'y accéder. C'est un logiciel libre qui génère des graphiques et tableaux de bord. Cette plateforme indépendante est aussi un outil indispensable pour créer des alertes.

Elle nous permet de visualiser :

Le trafic dans chaque service, et vérifier le pourcentage de succès des requêtes reçu pour un service.

- Les ressources mémoires utilisées par chaque déploiement, et aussi par l'ensemble des services de la couche du contrôle d'Istio.
- La performance des services injectés par Istio, en montrant des graphes pour : l'utilisation des ressources mémoire, et le nombre de requêtes exécutées à chaque instant pour un service donné.
- Le mode d'authentification utilisé. [33]

Par exemple la figure ci-dessous représente le tableau de bord qui montre le nombre des octets échangés via TCP connexion au niveau du namespace **dev-review** :

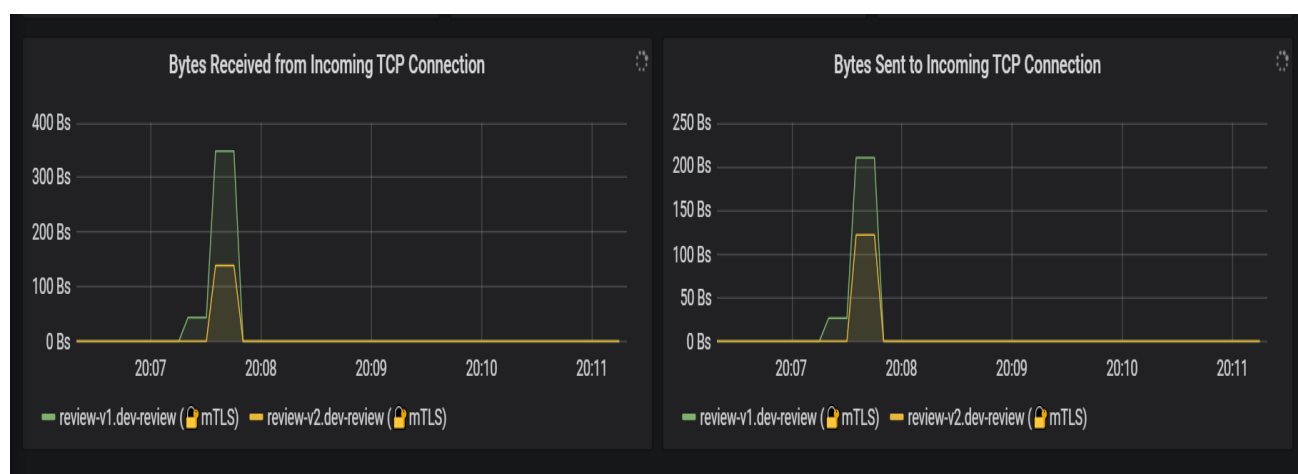


Figure 67 Grafana : Graphe de nombre d'octet échangé via TCP connexion avec le service 'Review'

La figure ci-dessous illustre le graphe des ressources consommées par le total des Proxy :

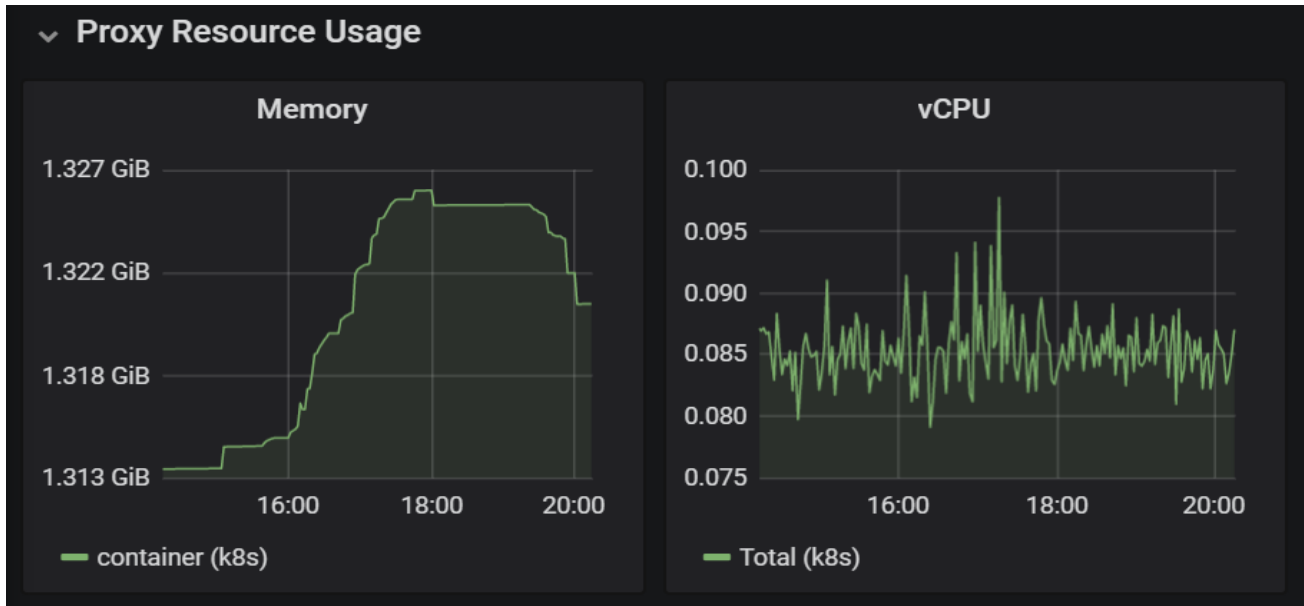


Figure 68 Grafana : graphe des ressources consommées par le total des proxy

5.4 Conclusion

Ce chapitre a été dédié pour décrire le déploiement de notre application e-commerce. Nous avons commencé par la description des outils et technologies utilisés, en suite une description du processus de conteneurisation de nos Microservices, puis on a passé à la description des configurations de déploiement dans le cluster du Cloud Azure, et enfin on a présenté quelques fonctionnalités du service mesh injecté.

Conclusion générale

Le but de ce travail est l'étude et le benchmarking des solutions du service Mesh, et l'intégration de la meilleure solution dans un contexte e-commerce. Ce projet était le sujet de mon stage de PFE qui a été effectué dans l'entreprise SQLI Rabat. Grâce à ce dernier j'ai pu apprendre et découvrir toute une nouvelle palette de technologies tendance et mettre en pratique mes connaissances dans le monde de l'informatique, ainsi que les différentes disciplines apprises au cours de la formation à l'EMSI Marrakech.

La réalisation de ce projet m'a permis de mettre en application l'esprit d'étude et d'analyse. Également, de mettre en pratique le savoir acquis au cours des années d'études lors de la période de ma formation à l'EMSI Marrakech, et de découvrir la différence entre les projets professionnels et ceux à caractère pédagogique.

La réalisation du présent travail fut une expérience très bénéfique dans la mesure où elle m'a permis d'apprendre les méthodes de travail, ainsi d'aborder le projet dans toutes les phases de son cycle de vie, de manière à avoir une vue globale qui facilitera la compréhension de toutes les phases de la réalisation d'un projet clientèle, dès la phase de la conception et d'analyse jusqu'au déploiement et livraison du projet au client.

Dans ce rapport, nous avons exposé les étapes de Benchmarking des services Meshs, de conception, de développement et de déploiement de notre application e-commerce, qui a été basée sur une architecture Microservice. Ensuite, l'injection d'une solution de service Mesh pour sécuriser la communication interne entre les services et de bénéficier d'autres fonctionnalités pour résoudre les problèmes d'une architecture Microservices.

Notre travail s'est déroulé sur trois phases. Nous avons commencé par la compréhension de notre domaine, la collecte des différentes solutions de service Mesh existant avec leurs avantages, inconvénients, fonctionnalités et limitations. Puis un choix initial des services Meshs candidats qui seront comparés dans la deuxième phase.

La deuxième phase, est une étude entre les Meshs choisis initialement comparative en se basant sur des exemples et des demos que nous avons réalisé ; une comparaison de performance et une autre

des fonctionnalités, puis à l'aide du client nous avons pu choisir la solution la plus adaptée à ces besoins.

La troisième phase consiste à implémenter notre solution dans une application e-commerce. Nous avons commencé par la spécification des besoins fonctionnels et non fonctionnels, que doit respecter l'application, suivis par des diagrammes des cas d'utilisation mettant en jeu les acteurs qui interagissent avec le système, puis nous sommes allés au fond dans chaque service pour spécifier ces besoins et les technologies que nous avons utilisé à la réalisation de ce service. Après que nous avons fini le développement, nous avons déployé l'application dans un cluster dans le Cloud Azure en injectant notre solution service Mesh.

Comme perspectives, Notre étape suivante est de finaliser tous les services d'une application de e-commerce, puis configurer le reste des fonctionnalités d'un service Mesh tel que le traçage distribué, et la gestion des logs, et finalement la réalisation de la partie frontend.

Références

- [1] SQLI - Page d'accueil <https://www.sqli-digital-experience.com/> (Consulté le 25/07/2020)
- [2] SQLI - Clients d'SQLI <https://www.sqli.com/Accueil/References> (Consulté le 25/07/2020)
- [3] SQLI - Partenaires d'SQLI <https://www.sqli.com/Accueil/Groupe/Partenaires> (Consulté le 26/07/2020)
- [4] SQLI - <https://www.thierry-pigot.fr/scrum-en-moins-de-10-minutes/> (Consulté le 26/07/2020)
- [5] Claudio Eduardo de Oliveira, "What is Service Mesh and Why you should consider it" <https://sensedia.com/en/api/what-is-service-Mesh-and-why-you-should-consider-it/>, 4 septembre 2019 (Consulté le 15/08/2020)
- [6] Burr Sutter - Red Hat, Présentation de conférence "Java Microservices : Cloud Ultra Native", https://docs.google.com/presentation/d/1FA9lN2DGoNJsO9LZKkUrCqd3UaR3-4BEnpkqbfVOi_M (Consulté le 15/08/2020)
- [7] Samir Behara, "Microservices Journey from Netflix OSS to Istio Service Mesh" <https://dzone.com/articles/Microservices-journey-from-netflix-oss-to-istio-se>, 11 Juin 2019 (Consulté le 16/08/2020)
- [8] Hashicorp, "Consul announcement" <https://www.hashicorp.com/blog/consulannouncement/>, 17-Avril-2014 (Consulté le 16/08/2020)
- [9] Adrew Jenkins, "Service Mesh Architectures" <https://aspenMesh.io/service-Mesharchitectures/> 23 Mars 2018 (consulté le 17/08/2020)
- [10] INNOQ, <https://servicemesh.es/> 22 Aout 2020 (consulté le 17/08/2020)
- [11] Istio Authors, <https://istio.io/latest/docs/concepts/what-is-istio/> 15 juillet 2020 (consulté le 17/08/2020)
- [12] Karthikeyan Shanmugam, "What is Service Mesh and Why Do We Need It?" <https://containerjournal.com/topics/container-ecosystems/what-is-service-Mesh-and-whydo-we-need-it/> 12 Décembre 2018 (consulté le 17/08/2020)
- [13] Consul, "Connect" <https://www.consul.io/docs/connect> (consulté le 17/08/2020)
- [14] Istio, "Security" <https://istio.io/latest/docs/concepts/security/> (Consulté le 18/08/2020)
- [15] Linkerd, "Automatic mTLS" <https://linkerd.io/2/features/automatic-mtls/> (Consulté le 18/08/2020)
- [16] Hashicorp, "Secure and Route Service Mesh Communication Across Kubernetes Clusters" <https://learn.hashicorp.com/consul/kubernetes/Mesh-gateways> (Consulté le 18/08/2020)
- [17] Michael Kipper, "Benchmarking Istio & Linkerd CPU" https://medium.com/@michael_87395/Benchmarking-istio-linkerd-cpu-c36287e32781 22 Avril 2019 (consulté le 19/08/2020)

- [18] Thillo Fromm, “Performance Benchmark Analysis of Istio and Linkerd” <https://kinvolk.io/blog/2019/05/performance-benchmark-analysis-of-istio-and-linkerd/> 18 Mai 2019 (consulté le 19/08/2020)
- [19] [https://fr.wikipedia.org/wiki/UML_\(informatique\)](https://fr.wikipedia.org/wiki/UML_(informatique)) décembre 2017 (consulté le 20/08/2020)
- [20] <https://www.lucidchart.com/pages/fr/diagramme-de-cas-dutilisation-uml> (consulté le 20/08/2020)
- [21] https://fr.wikipedia.org/wiki/Diagramme_de_s%C3%A9quence (consulté le 20/08/2020)
- [22] <https://www.lucidchart.com/pages/fr/diagramme-package-uml> (consulté le 20/08/2020)
- [23] https://fr.wikipedia.org/wiki/Diagramme_de_classes (consulté le 20/08/2020)
- [24] [https://en.wikipedia.org/wiki/Swagger_\(software\)](https://en.wikipedia.org/wiki/Swagger_(software)) (consulté le 21/08/2020)
- [25] <https://fr.wikipedia.org/wiki/GitLab> (consulté le 21/08/2020)
- [26] <https://www.redhat.com/fr/topics/containers/what-is-docker> (consulté le 21/08/2020)
- [27] <https://kubernetes.io/fr/docs/concepts/overview/what-is-kubernetes/> (consulté le 21/08/2020)
- [28] Chronique de Christophe Pichaud Et Les Experts Infeeny, le 03/04/20, <https://www.journaldunet.com/web-tech/developpeur/1490227-introduction-a-azure-kubernetes-services-aks/#:~:text=Qu'est%2Dce%20qu',l'analyse%20et%20la%20maintenance.> (consulté le 21/08/2020)
- [29] https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm (consulté le 21/08/2020)
- [30] <https://istio.io/latest/docs/concepts/traffic-management/> (consulté le 30/08/2020)
- [31] <https://istio.io/latest/docs/concepts/security/> (consulté le 30/08/2020)
- [32] <https://istio.io/latest/docs/concepts/security/> (consulté le 30/08/2020)
- [33] <https://www.syloe.com/glossaire/grafana/> (consulté le 30/08/2020)