



UNIVERSITÉ PARIS SACLAY

MASTER DATASCALE

Rapport de stage de fin de cycle

Maître de stage :
Mr. Mickael BENHASSEN

Superviseur :
Dr. Yehia TAHER

Dieu Merci KIMPOLO NKOKOLO
dieu-merci.kimpolo-nkokolo@ens.uvsq.fr

13 septembre 2021

Remerciements

Premièrement je remercie tout d'abord le Dieu tout Puissant pour le souffle de vie qu'il nous procure au quotidien et sa grâce .

Je remercie également toute l'équipe technique de Gigamesh, en particulier mon encadreur professionnel Mickaël BENHASSEN, pour sa disponibilité et son aide permanente.

Je tiens à remercier aussi, toute l'équipe pédagogique de l'Université Paris Saclay qui a assuré avec dévouement ma formation de Master DataScale et particulièrement Monsieur TAHER Yehia mon encadrant durant ce stage.

Enfin je voudrais aussi remercier très sincèrement ma famille pour avoir été toujours à mes côtés durant mes études.

Introduction

L'assurance de prêt généralement désignée par assurance emprunteur, est une garantie demandée par les prêteurs (les banques) lors d'une demande de prêt. Bien que ce ne soit pas une obligation légale en France, elle est exigée dans la quasi-totalité des cas. Cette assurance permet de couvrir les risques de défaut de paiement quelles que soient leurs causes, ce qui explique qu'elle soit ainsi exigée.

En France, en 2017, le chiffre d'affaire de l'assurance emprunteur (crédit à la consommation, prêt immobilier et prêt professionnel) était de 9 milliards d'euros dont 6 milliards d'euros provenant uniquement de l'assurance emprunteur des prêts immobiliers. Ces chiffres, sont toujours en augmentation. Et près de 50% du marché de l'assurance emprunteur est contrôlé par deux acteurs : CNP Assurances et Crédit Agricole.

Un petit nombre d'entreprises a le monopole du marché de l'assurance emprunteur en France, ce qui fait que le secteur manque d'innovation en terme d'amélioration des prix, de transparence et de simplicité des procédures de souscription. Aussi les assurances emprunteur traditionnelles ont construit leur analyse des risques sur une photographie figée de l'assuré, en décalage avec les nombreux changements que la vie réserve. Ils ont donc perdu de vue leur mission de départ : protéger en cas de problème. GigaMesh avec son produit d'assurance emprunteur Assurly compte révolutionner le domaine de l'assurance emprunteur en associant à son expertise assurantielle la transparence et les atouts technologiques de notre époque.

Pour GigaMesh l'assurance doit s'adapter aux moments de la vie des clients, tout au long de leur vie. En toute transparence, en cherchant à constamment simplifier la vie du client. Les technologies de pointe : Edge-computing, Machine learning, Cloud, Serverless, apis Data science doivent servir le client.

C'est pour participer à cette révolution et pour me permettre de compléter mon Master 2 Datascale qui exige un stage de 6 mois en entreprise que GigaMesh m'avait offert un stage de six mois dans son service IT comme développeur Backend.

L'essentiel de ce document sera constitué du développement dans l'ordre cité des points suivants : la présentation de l'entreprise d'accueil, un état de l'art par rapport aux missions qui m'avaient été confiées, la présentation des technologies utilisées, et la présentation de la méthodologie de travail et des solutions proposées par rapport à mes missions .

Résumé

GigaMesh est une entreprise oeuvrant dans le domaine de l'assurance emprunteur. GigMesh ambitionne de révolutionner le domaine de l'assurance en proposant un produit d'assurance simple accessible, à juste prix et aligné aux technologies de notre époque.

J'ai passé mon stage chez Gigamesh comme développeur Backend. Mon travail portait principalement sur des questions liées : au déploiement et l'intégration de microservices, à l'intégration des données, à la sécurisation de l'API et au déploiement des applications web dans l'environnement AWS.

J'avais ainsi amélioré et acquis les compétences techniques liées : au développement et à la sécurisation des API dans l'environnement AWS, à intégration de données dans Mysql avec Aurora, à la construction et au déploiement des conteneur Docker et aussi à l'utilisation des machines virtuelles dans le Cloud.

Mon stage chez Gigamesh m'avait également permis de découvrir le domaine d'assurance, d'acquérir des compétences en gestion de projet avec la méthodologie de travail Scrum et dans l'utilisation de certains outils tel que GitHub et Reflex.

Table des matières


1	Contexte du stage	1
1.1	Présentation de Gigamesh	1
1.2	L'organigramme de Gigamesh :	1
1.3	Localisation de Gigamesh :	3
1.4	Domaine d'activité de Gigamesh	3
1.4.1	Assurance emprunteur	3
1.4.2	Assurly	4
1.5	Objectif du stage	6
2	Etat de l'art	7
2.1	Le cloud computing	7
2.1.1	Les services du cloud computing	8
	IaaS (Infrastructure as a Service, en anglais)	8
	PaaS (Platform as a Service, en anglais)	8
	SaaS (Software as a Service, en anglais)	8
2.2	Les APIs(Application programming interface)	9
2.2.1	Objectif d'une API	9
2.2.2	Analogie d'une API avec la vie courante	9
2.3	Le modèle ou l'architecture Rest	9
2.3.1	Les principes de l'architecture REST	10
2.4	L'architecture serverless	11
2.5	Le protocole OAuth2 et la délégation d'autorisation	11
2.5.1	Exemple d'implémentation du protocole OAuth2	11
2.5.2	Le vocabulaire du protocole OAuth2	12
2.6	Intégration de données	15
2.6.1	Les avantages	15
2.6.2	Opérations ETL et intégration des données	15
2.7	Conteneurisation	16
2.7.1	Avantages	16
2.7.2	Exemple des systèmes de conteneurisation	16
3	Outils et technologies utilisés	17
3.1	Amazon Lambda	17
3.1.1	Fonctionnement	17
3.1.2	Avantages	17
3.2	Docker	18
3.2.1	Docker Hub	18
3.3	Amplify	19
3.4	Amazon Simple Storage Service (Amazon S3)	19
3.4.1	Avantages de l'utilisation d'Amazon S3	19
3.5	Aws Aurora	20
3.5.1	Que signifie « des performances trois fois supérieures à celles de PostgreSQL et MySQL »	20

3.6	Amazon Cognito	20
3.7	API Gateway	20
3.8	Amazon Dynamodb	21
3.9	CloudWatch	22
3.10	Reflex	23
3.10.1	Scénario d'intégration CEP	23
4	Approche suivie et solution proposée	25
4.1	Méthodologie de travail : Scrum avec Agile	25
4.1.1	Les principaux rôles de la méthodologie Scrum	26
4.2	Les différentes missions et solutions	27
4.2.1	Mission 1 :Etude de faisabilité sur le déploiement d'une application Front-End dans un environnement AWS	27
	Observations	27
	Solutions	27
4.2.2	Mission 2 :Le déploiement et l'intégration du service web (Reflex) dans le parcours de souscription	28
	Solutions	29
	Pipeline de déploiement	29
	Diagramme de séquence intégration	30
4.2.3	Mission 3 : Etude de la sécurité actuelle d'accès aux API	31
	Rappel des principes de sécurité	31
	La sécurité des API dans AWS	31
	Les contrôles d'accès	31
	Authentification et autorisations	32
	L'existent	32
	Observations	32
4.2.4	Mission 4(Mise en place d'un ETL d'intégration de données)	34
	Observations	34
	Solution	34

Table des figures

1.1	Direction	2
1.2	Organigramme	2
1.3	Illustration du système d'emprunt	3
1.4	L'architecture globale du SI	6
2.1	Le Cloud	7
2.2	API	9
2.3	Implicit grant	12
2.4	Code grant	13
2.5	Sever grant	13
2.6	Password grant	14
2.7	ETL	15
2.8	VM vs Conteneurisation	16
3.1	Docker	18
3.2	Exemple d'hébergement d'un site static	19
3.3	API Gateway	21
3.4	CloudWatch	22
3.5	Scénario d'intégration CEP	23
4.1	Solution à base de S3	27
4.2	Solution à base d'Amplify	28
4.3	Pipeline de déploiement de Reflex	29
4.4	Dockerfile de construction de l'image Reflex	29
4.5	Diagramme d'intégration de Reflex sur l'application web	30
4.6	Diagramme d'intégration de Reflex sur l'application web	30
4.7	Les contrôles d'accès	32
4.8	L'architecture du système de sécurité, Implicit grant	33
4.9	Etl d'intégration de données	34
4.10	La taille de la table où sont stockées nos données	34

Abbreviations

AWS	Amazon web service
CI/CD	Continuous integration/continuous delivery
CLI	Command line interface
MFA	Multiple factors authentication
B2B	Business to Business
B2C	Business to client
Rest	Representational state transfer
HTTP	Hypertext transfer protocol
CPU	Central processus unit 
SQL	Structured query language

Chapitre 1

Contexte du stage

Dans cette section je vais faire une présentation générale de Gigamesh, ses missions, et ses projets.

1.1 Présentation de Gigamesh

En octobre 2017, la société GigaMesh SAS est lancée. Créée par Toufik GOZIM et Mickaël BENHASSEN, elle est prévue pour être une InsurTech qui va se lancer dans le domaine de l'assurance emprunteur. Afin d'y parvenir, l'entreprise a travaillé depuis sa création à l'obtention de l'agrément de l'ACPR (Autorité de Contrôle Prudentiel et de Résolution). L'ACPR est une institution dont le rôle est de surveiller les banques et assurances en France. Cet agrément qu'elle doit fournir est indispensable afin de pouvoir fournir une offre d'assurance.

Le positionnement d'Assurly dans le domaine de l'assurance emprunteur est judicieux. Il s'agit d'un secteur détenu majoritairement depuis des dizaines d'années par des banques.

Seulement, en février 2017, le loi BOURQUIN a ouvert ce marché à d'autres acteurs en permettant aux assurés de pouvoir changer leur contrat d'assurance chaque année à la date anniversaire. De plus, le préavis pour effectuer ce changement est passé de 15 jours (loi HAMON) à 2 mois. Cette loi a permis de faire jouer la concurrence pour faire gagner du pouvoir d'achat aux assurés.

L'idée a ainsi pris forme petit à petit avant que le projet ne devienne concret en octobre 2017 avec la création de la société GigaMesh SAS. La société a depuis lors changé de statut, le 30 mars 2020, de SAS à SA, afin de pouvoir devenir une société d'assurance quand l'agrément serait obtenu.

Assurly est en réalité le nom donné au produit de l'assurance emprunteur de la société Gigamesh. Le souhait avec Assurly est de révolutionner le secteur de l'assurance emprunteur sur tous les fronts. Redonner du pouvoir d'achat, avoir une approche technologique différente et développer de nouvelles relations avec les clients sont au cœur de leur vision.

1.2 L'organigramme de Gigamesh :

La direction est composée de trois membres : le CEO Toufik GOZIM, le CTO Mickaël BENHASSEN et le Directeur Général Délégué Alexandre RISPAL. Elle dirige une équipe d'une vingtaine de personnes.



FIGURE 1.1 – Direction

M. Mickaël BENHASSEN est à la tête du département Tech dans lequel j’ai passé mon stage. Ce département est celui qui dispose du plus grand nombre de recrues. J’ai pu travailler avec deux développeurs Pierre GANCEL, développeur front-end et Anderson LOYEM, développeur back-end, ainsi que quatre autres stagiaires comme moi : Nikita KUMARI, Meriem BANOUNE, Josias SEMANOU et Ghaith MAGROUNE. Ensemble, nous avons réalisé les missions confiées en travaillant parfois à deux sur un sujet et j’ai pu bénéficier de l’aide de chacun d’eux durant mon expérience.

Les autres départements sont celui des Opérations, chargé de la stratégie opérationnelle de l’entreprise et du développement de son activité, le Juridique et Finances qui veille au suivi des procédures juridiques et établit la stratégie financière, le Marketing et Communication qui élabore les plans marketing et s’occupe de la promotion de l’entreprise et enfin le département Commercial qui gère les ventes. Depuis peu, l’entreprise a embauché un responsable des Ressources Humaines, dont la mission sera de veiller au recrutement et à permettre la bonne santé et un environnement de travail sans stress à tous ses salariés.

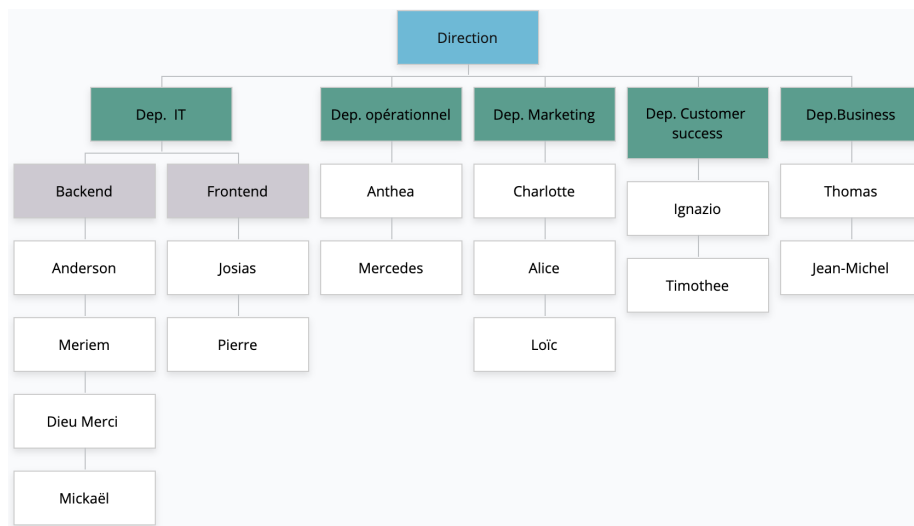


FIGURE 1.2 – Organigramme

1.3 Localisation de Gigamesh :

Localisé à Paris dans un espace de travail : **platform58, Gigamesh** profite d'un environnement de coworking et de bureaux privés. Un espace de coworking est un espace de travail partagé mettant en avant l'échange et l'ouverture aux autres. Située au **58, rue de la Victoire 75009 Paris**, la **platform58** est un lieu d'innovation de la Banque Postale qui repose sur deux briques :

- Un programme d'accompagnement de startups
- Un lieu physique proposant la location d'espaces de travail

1.4 Domaine d'activité de Gigamesh

1.4.1 Assurance emprunteur

L'assurance de prêt généralement désignée par assurance emprunteur (article L. 313-29 du code de la consommation) est une garantie demandée par les prêteurs (les banques) lors d'une demande de prêt. Bien que ce ne soit pas une obligation légale, elle est exigée dans la quasi-totalité des cas. Cette assurance permet de couvrir les risques de défaut de paiement quelles que soient leurs causes, ce qui explique qu'elle soit ainsi exigée. Elle comporte des garanties couvrant les risques :

- D'incapacité
- D'invalidité
- Voire de perte d'emploi

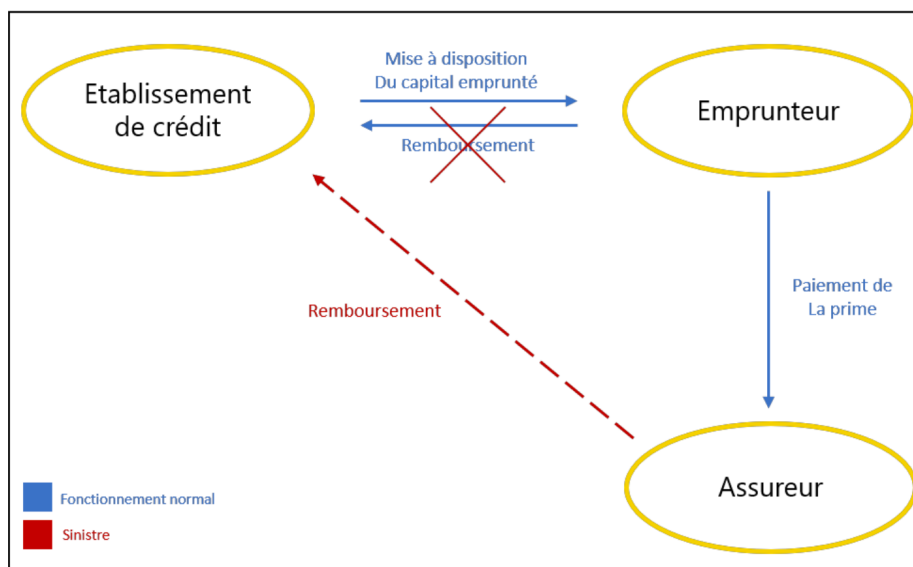


FIGURE 1.3 – Illustration du système d'emprunt.

1.4.2 Assurly

Assurly est le nom donné au produit d'assurance de Gigamesh et représente tout le travail effectué par l'entreprise depuis sa création. Assurly est donc le résultat du projet de Gigamesh. Assurly est une assurance emprunteur qui bouleverse le secteur de l'assurance en créant un produit clair, simple et au juste prix.

Tout est parti du constat que les assurances emprunteur traditionnelles ont construit leur analyse des risques sur une photographie figée de l'assuré, en décalage avec les nombreux changements que la vie réserve. Ils ont donc perdu de vue leur mission de départ : protéger en cas de problème.

Suite à 3 ans de recherche et développement, Gigamesh a réussi à associer les apports de l'innovation technologique et leur expertise assurantielle pour créer Assurly.

Assurly est née avec l'ambition de redonner le pouvoir aux assurés en proposant un produit clair avec des garanties all-inclusive au meilleur tarif. Assurly transforme l'assurance emprunteur grâce à une expérience client simplifiée et 100% digitale : une souscription en moins de 10 minutes depuis son portable avec zéro paperasse, zéro rendez-vous, zéro stress!

Assurly propose de couvrir ces risques avec les garanties suivantes :

1. La garantie perte totale et irréversible d'autonomie (PTIA).

Elle intervient lorsque l'assuré se trouve dans un état particulièrement grave, nécessitant le recours permanent à une tierce personne pour exercer les actes ordinaires de la vie.

La couverture Assurly : Gigamesh sera là pour vous épauler et rembourser 100% des mensualités du reste de votre prêt. La garantie PTIA cesse au 71ème anniversaire de l'assuré.

2. La garantie incapacité temporaire totale (ITT) dénommée incapacité temporaire totale dans le contrat.

Elle intervient lorsque la personne assurée est temporairement inapte à exercer son activité professionnelle ou, si il n'en a pas, d'observer un repos complet l'obligeant à interrompre toutes ses occupations de la vie quotidienne

La couverture Assurly : Gigamesh sera là pour vous épauler et payer 100% vos mensualités de remboursement de prêt durant votre temps d'impossibilité de travailler avec un plafond de 7500 euros par mois, quelle que soit votre perte de revenu. La garantie ITT cesse au plus tard au jour du 65ème anniversaire de l'assuré. Les affections dorsales, psychiques et psychiatriques causant l'ITT sont couvertes sans condition d'hospitalisation ou d'intervention chirurgicale.

La garantie Invalidité prend deux formes :

- (a) La garantie invalidité permanente partielle (IPP) :

Elle intervient lorsque la personne assurée est, de façon définitive, incapable d'exercer strictement son activité après la reconnaissance de l'état d'invalidité estimé avec un taux entre 33% et 66%.

La couverture Assurly : Gigamesh sera là pour vous épauler et payer 50% de l'indemnité garantie en cas d'ITT, quelle que soit votre perte de revenu, avec un plafond de 7500€ par mois. Le délai de franchise maximale est de 90 jours après l'interruption de l'activité. Les affections dorsales, psychiques et psychiatriques causant l'IPP sont couvertes sans condition d'hospitalisation ou d'intervention chirurgicale

(b) La garantie Invalidité Permanente Totale (I.P.T.)

Elle intervient lorsque la personne assurée est, de façon définitive, incapable d'exercer strictement son activité professionnelle après la reconnaissance de l'état d'invalidité estimé avec un taux supérieur à 65%

La couverture Assurly : Gigamesh sera là pour vous épauler et rembourser 100% des mensualités du reste de votre prêt avec un plafond de 3 000 000 euros, quelle que soit votre perte de revenu. La garantie invalidité cesse au jour du 65ème anniversaire de l'assuré.

3. La garantie décès

Elle intervient en cas de décès de la personne assurée. Nous serons là pour épauler votre famille et payer 100% de vos mensualités de remboursement de votre prêt.

1.5 Objectif du stage

Dans le cadre de son activité, le système d'Information de Gigamesh est constitué d'un frontend, d'un backend et d'applications tiers :

- Une application mobile B2C, pour la simulation de l'assurance et l'espace client,
- Une application B2B, pour la simulation de l'assurance et de la création d'un assuré par des partenaires,
- Un ensemble d'application de gestion, paiement en ligne et service client,
- Une application backend pour la gestion et données et la fourniture des APIs.

Les applications frontend communiquent avec le backend via une API de type Rest. Dans le cadre de mon stage j'intervenais plus souvent dans le backend et sur les missions suivantes :

- Le déploiement et l'intégration du service web (Reflex) dans un environnement AWS
- Intégration des données en format txt dans une base de données sql dans l'environnement AWS
- Etude de faisabilité sur le déploiement d'une application Front-End dans un environnement AWS
- Etude de la sécurité actuelle d'accès aux API.

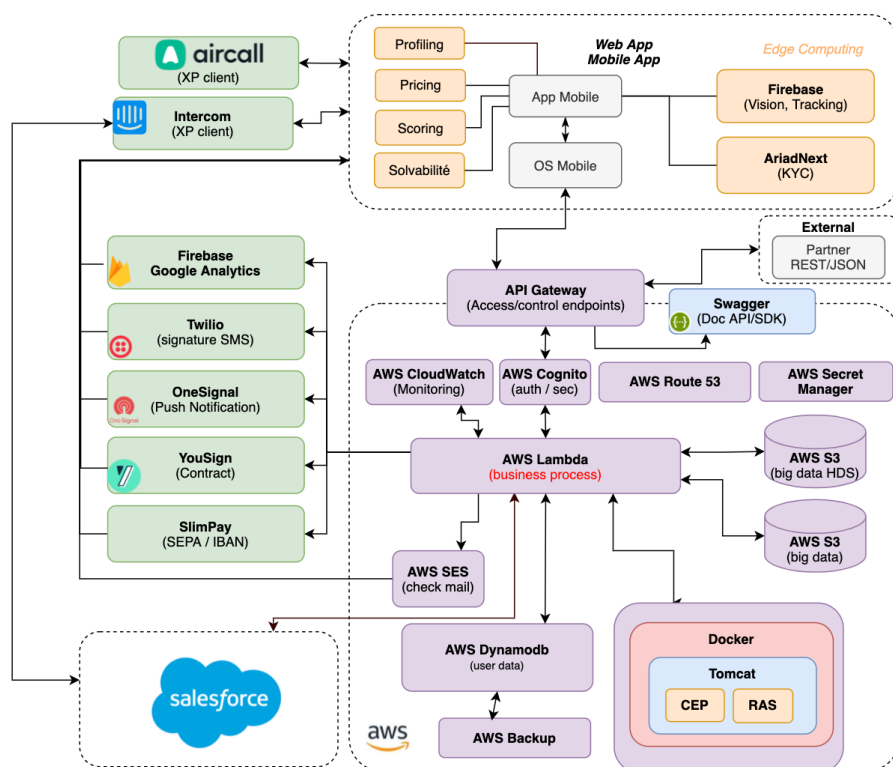


FIGURE 1.4 – L'architecture globale du SI

Chapitre 2

Etat de l'art

Dans cette partie je vais présenter le positionnement des mes missions par rapport à l'état de l'art scientifique ou technique.

2.1 Le cloud computing

Le cloud computing ou informatique en nuage est une infrastructure dans laquelle la puissance de calcul et le stockage sont gérés par des serveurs distants auxquels les usagers se connectent via une liaison Internet sécurisée.

L'ordinateur de bureau ou portable, le téléphone mobile, la tablette tactile et autres objets connectés deviennent des points d'accès pour exécuter des applications ou consulter des données qui sont hébergées sur les serveurs.

Le cloud se caractérise également par sa souplesse qui permet aux fournisseurs d'adapter automatiquement la capacité de stockage et la puissance de calcul aux besoins des utilisateurs.

Comme illustration, le cloud computing se matérialise notamment par les services de stockage et de partage de données numériques type Box, Dropbox, Microsoft OneDrive ou Apple iCloud sur lesquels les utilisateurs peuvent stocker des contenus personnels (photos, vidéos, musique, documents...) et y accéder n'importe où dans le monde depuis n'importe quel terminal connecté.

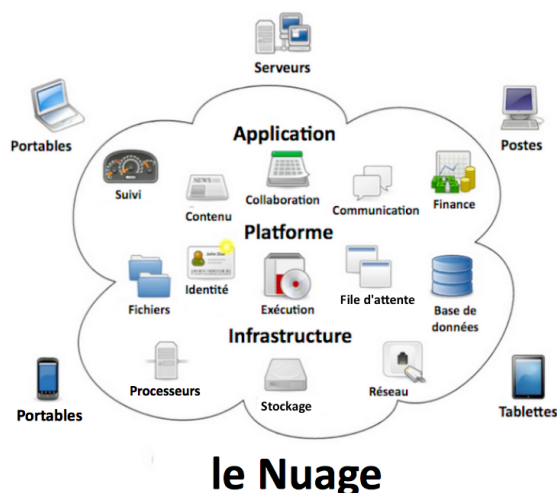


FIGURE 2.1 – Le Cloud

2.1.1 Les services du cloud computing

On distingue plusieurs types de services cloud

IaaS (Infrastructure as a Service, en anglais)

C'est un modèle du cloud computing où l'entreprise dispose sur abonnement payant d'une infrastructure informatique (serveurs, stockage, sauvegarde, réseau) qui se trouve physiquement chez le fournisseur qui est aussi responsable pour la sécurité de l'infrastructure.

Cela peut représenter pour certaines directions des systèmes d'information (DSI) un moyen de réaliser des économies, principalement en transformant des investissements en contrats de location.

PaaS (Platform as a Service, en anglais)

C'est un modèle du cloud computing qui permet de mettre à disposition des entités clientes un environnement d'exécution rapidement disponible, en leur laissant la maîtrise des applications qu'elles peuvent installer, configurer et utiliser elles-mêmes.

Il se distingue ainsi du modèle SaaS où la même application est mise à disposition des nombreux utilisateurs finaux

SaaS (Software as a Service, en anglais)

C'est un modèle du cloud computing qui fournit des applications sous forme de services clés en mains auxquels les utilisateurs se connectent via des logiciels dédiés ou un navigateur Internet.

Exemple Les service de messageries électroniques type Gmail, Yahoo, Outlook.com ou de suites bureautiques type Office 365 ou Google Apps.

2.2 Les APIs(Application programming interface)

une API est une façade clairement délimitée par laquelle une application offre des services à d'autres applications. Cette façade peut comporter des classes, des méthodes ou des fonctions, des types de données et des constantes. Les API peuvent être publiques, partenaires ou privées

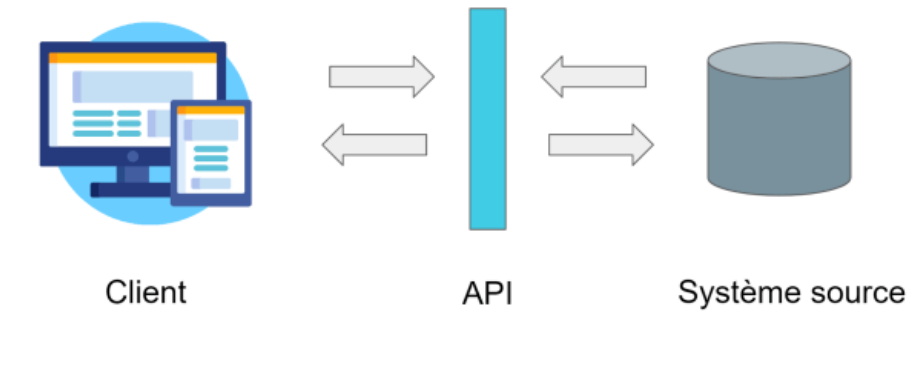


FIGURE 2.2 – API

2.2.1 Objectif d'une API

Tout comme une interface utilisateur graphique facilite l'utilisation des programmes pour les profanes, une API a pour objectif de fournir une porte d'accès à une ou plusieurs fonctionnalités d'un système en exposant uniquement les objets ou les actions dont le développeur a besoin et en cachant les détails de la mise en œuvre. Elles permettent ainsi aux développeurs d'utiliser plus facilement certaines technologies pour créer des applications. Une API simplifie la programmation.

2.2.2 Analogie d'une API avec la vie courante

Prenons l'exemple d'un appel téléphonique, la plupart des gens ne savent pas comment se déroule un appel téléphonique : le mécanisme électronique qui régit un appel téléphonique. Pourtant tout le monde peut effectuer un appel juste en composant le numéro de son correspondant et en appuyant simplement sur un bouton (envoyer ou ok) du clavier. Le clavier et l'écran du téléphone sont l'équivalent de l'API ; ils représentent l'interface du système électronique qui régit un appel téléphonique

2.3 Le modèle ou l'architecture Rest

Les APIs peuvent être intégrées suivant plusieurs modèles ou styles d'architecture : le modèle à base des simples fonctions, le modèle RPC, le modèle SOAP, le modèle Rest puis le modèle GraphQL. Le modèle Rest, est un style orienté ressources et utilisant principalement le protocole HTTP pour la communication. Avec le modèle Rest les données entre le client et le fournisseur sont transmises en format JSON ou XML. Ainsi, Rest utilise des notions qu'il faut obligatoirement comprendre : les ressources, les identifiants, les méthodes HTTP, et les formats de données.

2.3.1 Les principes de l'architecture REST

L'architecture Rest est régit par six principes suivants :

1. La séparation entre client et serveur
les responsabilités du côté serveur et du côté client sont séparées, si bien que chaque côté peut être implémenté indépendamment de l'autre. Le code côté serveur (l'API) et celui côté client peuvent chacun être modifiés sans affecter l'autre, tant que tous deux continuent de communiquer dans le même format. Dans une architecture REST, différents clients envoient des requêtes sur les mêmes endpoints, effectuent les mêmes actions et obtiennent les mêmes réponses.
2. . L'absence d'état de sessions (stateless)
la communication entre client et serveur ne conserve pas l'état des sessions d'une requête à l'autre. Autrement dit, l'état d'une session est inclus dans chaque requête, ce qui signifie que ni le client ni le serveur n'a besoin de connaître l'état de l'autre pour communiquer. Chaque requête est complète et se suffit à elle-même : pas besoin de maintenir une connexion continue entre client et serveur, ce qui implique une plus grande tolérance à l'échec. De plus, cela permet aux APIs REST de répondre aux requêtes de plusieurs clients différents sans saturer les ports du serveur. L'exception à cette règle est l'authentification, pour que le client n'ait pas à préciser ses informations d'authentification à chaque requête.
3. L'uniformité de l'interface
les différentes actions et/ou ressources disponibles avec leurs endpoints et leurs paramètres spécifiques doivent être décidés et respectés religieusement, de façon uniforme par le client et le serveur. Chaque réponse doit contenir suffisamment d'informations pour être interprétée sans que le client n'ait besoin d'autres informations au préalable. Les réponses ne doivent pas être trop longues et doivent contenir, si nécessaire, des liens vers d'autres endpoints.
4. La mise en cache
les réponses peuvent être mises en cache pour éviter de surcharger inutilement le serveur. La mise en cache doit être bien gérée : l'API REST doit préciser si telle ou telle réponse peut être mise en cache et pour combien de temps pour éviter que le client ne reçoive des informations obsolètes.
5. L'architecture en couches
un client connecté à une API REST ne peut en général pas distinguer s'il est en communication avec le serveur final ou un serveur intermédiaire. Une architecture REST permet par exemple de recevoir les requêtes sur un serveur A, de stocker ses données sur un serveur B et de gérer les authentifications sur un serveur C.
6. Le code à la demande Cette contrainte est optionnelle. Elle signifie qu'une API peut retourner du code exécutable au lieu d'une réponse en **JSON** ou en **XML** par exemple. Cela signifie qu'une API **RESTful** peut étendre le code du client tout en lui simplifiant la vie en lui fournissant du code exécutable tel qu'un script **JavaScript**.

Une API REST ne peut être qualifiée de **RESTful** si elle ne respecte pas les six contraintes, mais on peut tout de même la qualifier d'API REST si elle n'enfreint que deux ou trois principes. REST est sans doute le standard le plus utilisé pour concevoir des architectures d'API, mais il en existe bien d'autres qui pourraient le compléter, voire un jour le détrôner

2.4 L'architecture serverless

C'est une architecture où l'utilisateur n'a pas à gérer la moindre infrastructure. L'architecture serverless ne signifie pas pour autant qu'il n'y a pas de serveurs : cela veut dire qu'ils sont invisibles pour l'utilisateur et sont gérés par les fournisseurs et non par les consommateurs. Sans trop penser à leur maintenance, les ressources informatiques sont utilisées comme des services.

L'architecture serverless fait changer la façon dont on conçoit et maintient les applications. Elle permet aux consommateurs de bâtir des plateformes en utilisant exclusivement des services managés, de se concentrer sur les aspects liés à la logique business, et non sur les contraintes de déploiement ou de scalabilité.

- Exemples fournisseur des architectures Serverless

- AWS

- * Amazon S3

- Amazon Dynamodb

- * Amazon Lambda

- Google :

- * Google Cloud Functions

- Microsoft

- * Azure Functions

2.5 Le protocole OAuth2 et la délégation d'autorisation

Le OAuth2 est la seconde version du protocole OAuth(Open Authorization). Connu comme protocole de délégation d'autorisations, le OAuth2 est un protocole qui permet à une application d'obtenir une autorisation d'accès limitée aux ressources disponibles sur un serveur accessible via HTTP. Si ces ressources n'appartiennent pas à l'application, l'autorisation d'accès lui est déléguée par le détenteur des ressources ; au cas contraire l'application obtient l'autorisation en s'authentifiant avec ses identifiants. Avec le protocole OAuth2 le propriétaire de ressources ne partage pas ses identifiants avec l'application qui sollicite ses ressources. Le protocole OAuth2 fournit un modèle dans lequel le détenteur de ressource peut accorder un accès limité à ses données en émettant simplement un jeton temporaire, qui sera utilisé par cette application sollicitant ses données pour s'identifier auprès du serveur de ressources. Le protocole OAuth2 met le propriétaire de ressources au cœur du système d'octroi d'autorisation. C'est le propriétaire de ressources qui fait le lien entre ses comptes sur différentes applications sans que des administrateurs de la sécurité aient besoin d'intervenir directement sur chaque application.

2.5.1 Exemple d'implémentation du protocole OAuth2

Le protocole OAuth2 est utilisé par plusieurs entreprises comme Facebook, Google et Twitter. L'exemple que nous proposons ici est celui qui permet d'afficher instantanément les tweets sur Facebook sans avoir besoin de votre mot de passe Facebook

2.5.2 Le vocabulaire du protocole OAuth2

Les rôles : Le protocole OAuth2 identifie quatre rôles : Client, Propriétaire de ressources, Serveur d'autorisation et Serveur de ressources.

- **Le détenteur des ressources(Resource Owner) :**
Le détenteur ou le propriétaire de ressources est une entité capable d'accorder l'accès à une ressource protégée. Lorsque le propriétaire de la ressource est une personne, on parle d'utilisateur final.
Le serveur de ressources (Resource Server) :
Le serveur de ressources est le serveur qui héberge les ressources protégées ou à accès limité. Un exemple du serveur des ressources peut être Facebook et Google qui hébergent les informations des profils des utilisateurs.
- **Le client (Client Application) :**
C'est une application qui sollicite les ressources du serveur de ressources, celle-ci peut être une application PHP, une application mobile, une application Javascript.
- **Le serveur d'autorisation (Authorization Server) :**
Le serveur d'autorisation représente le serveur qui délivre des jetons au client. Ces jetons seront utilisés lors des requêtes du client vers le serveur de ressources. Ce serveur peut être le même que le serveur de ressources (physiquement et applicativement), et c'est souvent le cas.

Les jetons : Un jeton est une chaîne des caractères, générée par le serveur d'autorisation au client une fois qu'une autorisation lui est offerte.

- **Le jeton d'accès(Access token) :**
est un jeton nécessaire pour accéder aux ressources partagées par OAuth2. Il a une durée de vie qui est généralement assez courte.
- **Le jeton de renouvellement (Refresh Token) :**
est un jeton que le serveur d'autorisation peut émettre aux clients et il peut être échangé contre un nouveau jeton d'accès, sans répéter le processus d'autorisation. Il n'a pas de temps d'expiration.

Les types d'autorisations(Grant types)

- **L'autorisation implicite (Implicit Grant) :**
Elle doit être utilisée quand l'application se trouve côté client(typiquement une application Javascript). Il ne permet pas d'obtenir de token de renouvellement.

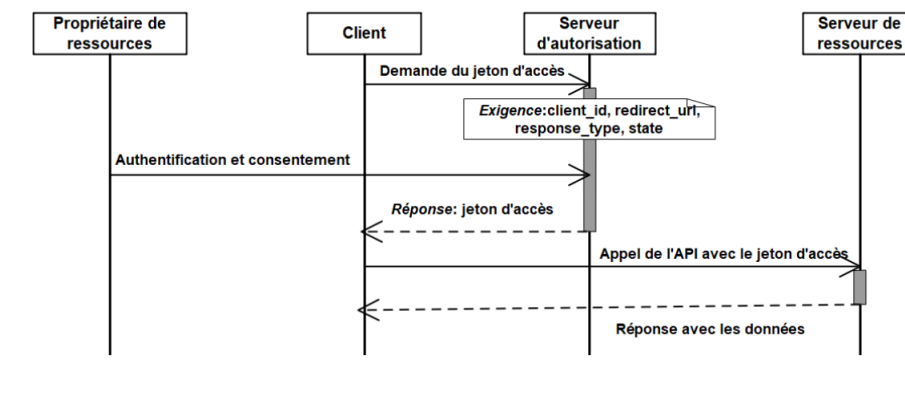


FIGURE 2.3 – Implicit grant

L'autorisation via un code (Authorization Code Grant) :

Ce type d'autorisation permet d'obtenir deux jetons : le jeton d'accès et le jeton de renouvellement. Le client dans ce cas de figure interagit avec le propriétaire des ressources via un client web généralement un navigateur

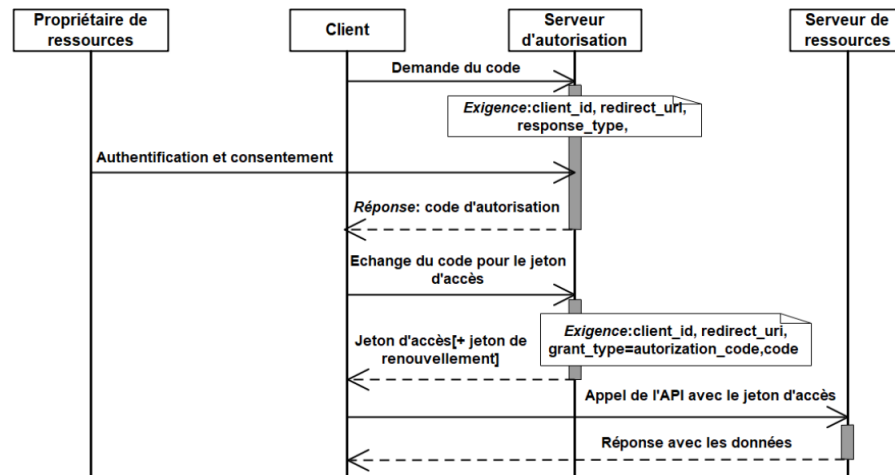


FIGURE 2.4 – Code grant

- L'autorisation serveur à serveur (Client Credentials Grant) :
Elle doit être utilisée lorsque le client est lui-même le détenteur des données. Il n'y a pas d'autorisation à obtenir de la part de l'utilisateur

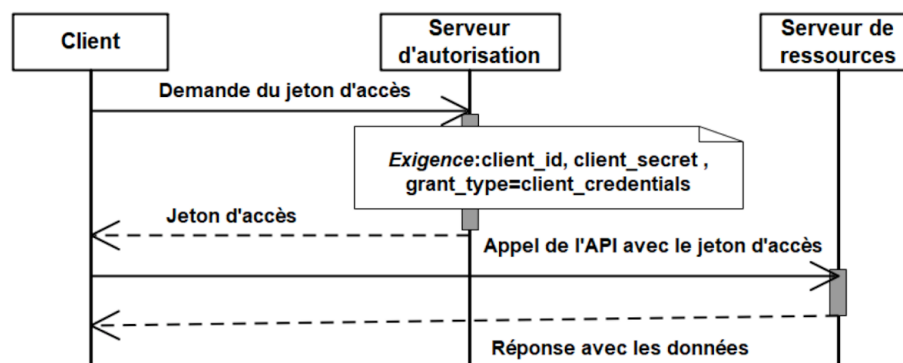


FIGURE 2.5 – Server grant

- L'autorisation via mot de passe (Resource Owner Password Credentials Grant) :
Avec ce type d'autorisation, les identifiants sont envoyés au client et ensuite au serveur d'autorisation. Il est donc impératif qu'il y ait une confiance absolue entre ces 2 entités. Ce type d'autorisation est principalement utilisé lorsque le client a été développé par la même autorité que celle fournissant le serveur d'autorisation final.

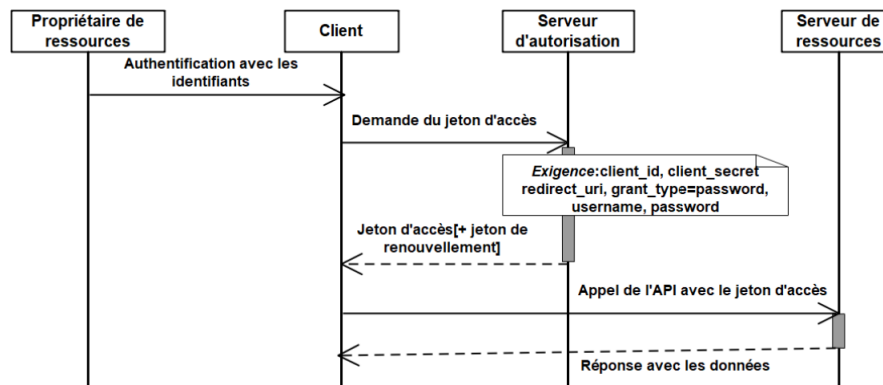


FIGURE 2.6 – Password grant

2.6 Intégration de données

L'intégration des données est le processus qui consiste à combiner des données provenant de différentes sources dans une vue unifiée : de l'importation au nettoyage en passant par le mapping et la transformation dans un gisement cible, pour finalement rendre les données plus exploitables et plus utiles pour les utilisateurs qui les consultent.

2.6.1 Les avantages

- L'intégration des données améliore l'unification des systèmes et la collaboration globale

L'intégration des données fait gagner du temps

- L'intégration des données réduit les erreurs (et les besoins de modifications)

2.6.2 Opérations ETL et intégration des données

Les opérations d'extraction, de transformation et de chargement (ETL) forment un processus d'intégration à part entière dans lequel les données sont extraites du système source et livrées au data warehouse. Il s'agit d'un processus continu que le data warehousing exécute pour transformer plusieurs sources de données en informations cohérentes et utiles destinées à la Business Intelligence et aux analyses.

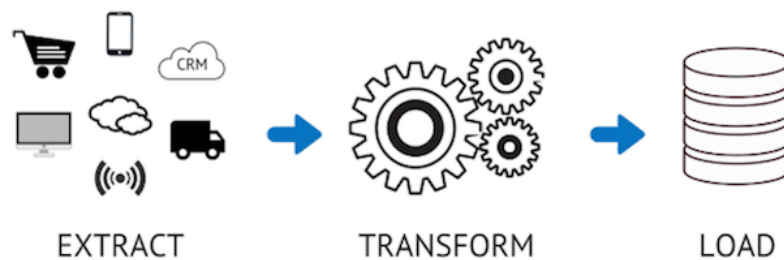


FIGURE 2.7 – ETL

2.7 Conteneurisation

Un conteneur est un système qui permet de stocker et d'isoler des objets devant être transportés ou déployés dans un environnement d'exploitation étendu (applications, logiciels, librairie, etc.). Il permet également au code applicatif d'être transporté de l'environnement de développement vers celui de production de manière aisée et sûre. La conteneurisation permet :

- De virtualiser, à l'intérieur d'un conteneur les ressources matérielles dont une application a besoin pour être exécutée (mémoire, réseau, processeur, etc.)
- D'embarquer les composants logiciels nécessaires à l'application (données, fichiers, etc.)

La conteneurisation est parfois confondue avec la virtualisation. Quand la première nécessite que chaque machine virtuelle possède son système d'exploitation, les conteneurs eux se connectent au noyau des machines, kernel, pour en exploiter les ressources. Moins lourds, ils sont plus faciles à déplacer et à stocker.

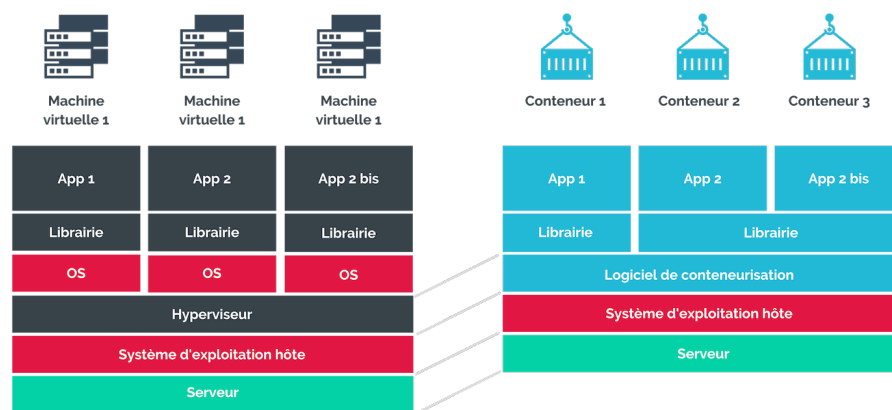


FIGURE 2.8 – VM vs Conteneurisation

2.7.1 Avantages

- Elle évite de se soucier d'interactions ou d'incompatibilités avec les conteneurs déjà présents ou à venir sur cette machine.
- Elle permet de ne pas occuper autant de ressources que réclamerait une machine virtuelle (ou virtual machine, VM), qui emporte son propre système d'exploitation et bloque des ressources à son lancement.

2.7.2 Exemple des systèmes de conteneurisation

- Docker
- Kubernetes
- CoreOs rkt
- OpenV

Chapitre 3

Outils et technologies utilisés

Dans cette section je vais présenter les outils et les technologies que j'avais utilisé durant mon stage.

3.1 Amazon Lambda

AWS Lambda est un service de calcul Serverless qui vous permet d'exécuter du code sans :

- provisionner ou gérer des serveurs, créer une logique de dimensionnement de cluster prenant en charge la charge de travail,
- maintenir les intégrations d'événements ou gérer les environnements d'exécution.

Avec Lambda, vous pouvez exécuter du code pour pratiquement n'importe quel type d'application ou service backend , sans aucune tâche administrative. Il suffit de télécharger votre code sous forme de fichier ZIP ou d'image de conteneur, et Lambda alloue automatiquement et précisément la puissance d'exécution de calcul et exécute votre code en fonction de la demande ou de l'événement entrant, pour n'importe quelle échelle de trafic. Vous pouvez configurer votre code de sorte qu'il se déclenche automatiquement depuis plus de 200 applications SaaS et services AWS, ou l'appeler directement à partir de n'importe quelle application web ou mobile. Vous pouvez écrire des fonctions Lambda dans votre langage préféré (Node.js, Python, Go, Java, etc.)

3.1.1 Fonctionnement

Chaque fonction Lambda s'exécute dans son propre conteneur. Lorsqu'une fonction est créée, Lambda l'empaquette dans un nouveau conteneur, puis exécute ce conteneur sur un cluster de machines mutualisées géré par AWS. Avant que les fonctions ne commencent à s'exécuter, le conteneur de chaque fonction se voit allouer la mémoire RAM et la capacité CPU nécessaires. Une fois que les fonctions ont fini de s'exécuter, la RAM allouée au début est multipliée par le temps que la fonction a passé à s'exécuter. Les clients sont ensuite facturés en fonction de la mémoire allouée et de la durée d'exécution de la fonction.

3.1.2 Avantages

- Aucun serveur à gérer : AWS Lambda exécute automatiquement votre code, sans que vous ayez à mettre en service ou à gérer des serveurs

Dimensionnement continu : AWS Lambda dimensionne automatiquement votre application en exécutant le code en réponse à chaque déclencheur. Votre code s'exécute en parallèle et traite chaque déclencheur indépendamment. La charge de travail est ainsi mise à l'échelle de façon précis

- Optimisation des coûts grâce au comptage en millisecondes : Avec AWS Lambda, vous ne payez que pour le temps de calcul que vous consommez
- Performances constantes à n'importe quelle échelle : Avec AWS Lambda, vous pouvez optimiser le temps d'exécution de votre code en choisissant la bonne taille de mémoire pour votre fonction

3.2 Docker

Docker est un système de containerisation le plus utilisé ; qui vous permet de créer, déployer et lancer vos applications en utilisant des conteneurs. Pour mettre en place ces conteneurs, on crée des images Docker. L'image Docker permet de configurer tout l'environnement dans lequel le conteneur va s'exécuter. Pour créer ces images, Docker utilise un fichier spécial appelé Dockerfile, qui grâce à une syntaxe simple et élégante va nous permettre de préparer nos images. L'image est ensuite construite par le démon Docker via l'utilisation de commandes dans le terminal qui sont regroupées dans ce qu'on appelle un CLI. Pour gérer l'ensemble des conteneurs d'une application, on utilise Docker Compose.

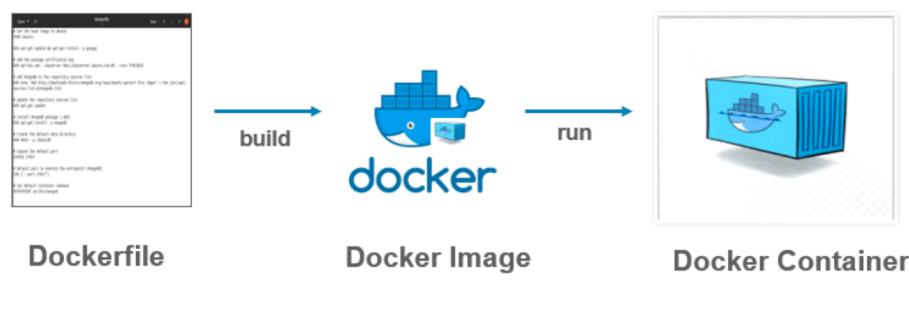


FIGURE 3.1 – Docker

Un conteneur est une instance d'une image et une image est obtenue en compilant le fichier Dockerfile.

3.2.1 Docker Hub

Docker Hub est un service fourni par Docker pour rechercher et partager des images de conteneurs avec votre équipe. Il s'agit du plus grand référentiel au monde d'images de conteneurs. Docker Hub fournit les fonctionnalités principales suivantes :

- Repositories : permet le push et le pull des images des conteneurs
Teams and Organizations : permet de gérer l'accès aux référentiels privés d'images de conteneurs
- Docker Official Images : permet de récupérer et d'utiliser des images de conteneurs de haute qualité fournies par Docker
- Builds : permet de créer automatiquement des images de conteneur à partir de GitHub et Bitbucket et de les transférer vers Docker Hub.

Docker fournit un outil Docker Hub CLI et une API qui vous permet d'interagir avec Docker Hub.

3.3 Amplify

AWS Amplify est un ensemble d'outils et de services qui peuvent être utilisés ensemble ou un par un, pour :

- **Authentification** : accéder à des workflows prêts à l'emploi pour MFA, authentification unique, mot de passe oublié, etc.
Hébergement : déployer des applications Web statiques en quelques clics et facilement gérer le contenu (JavaScript, React, Angular, Flutter,...)
- **Notifications push** : gérer facilement les campagnes et envoyer des messages aux utilisateurs via plusieurs canaux, notamment par SMS, e-mail et push.
- **Analytique** : suivre les sessions des utilisateurs et créer des rapports sur leur comportement. Configurer des attributs personnalisés et analyser les entonnoirs de conversion.

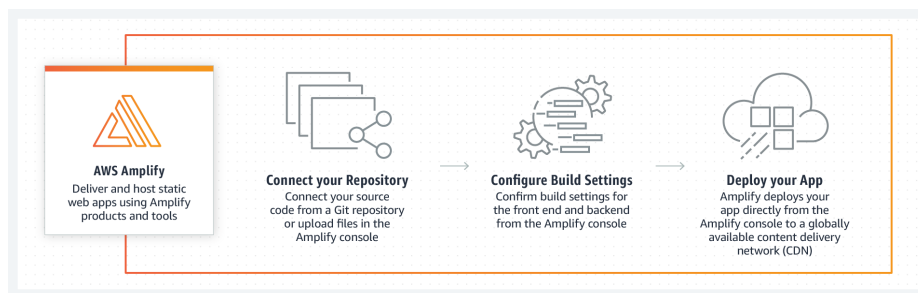


FIGURE 3.2 – Exemple d'hébergement d'un static

3.4 Amazon Simple Storage Service (Amazon S3)

Amazon S3 est un stockage d'objets conçu pour stocker et récupérer n'importe quelle quantité de données, n'importe où. Il s'agit d'un service de stockage simple qui offre une durabilité, une disponibilité, des performances, une sécurité, et une scalabilité de pointe pratiquement illimitée à un tarif très bas. S3 supporte tout type de fichier et peut être utilisé comme repos d'hébergement des sites web statics :

3.4.1 Avantages de l'utilisation d'Amazon S3

Amazon S3 est intentionnellement conçu avec un ensemble de fonctionnalités minimal qui met l'accent sur la simplicité et la robustesse. Voici quelques-uns des avantages de l'utilisation d'Amazon S3 :

- **Création de compartiments** : Créez et nommez un compartiment qui stocke les données. Les compartiments sont les conteneurs fondamentaux d'Amazon S3 pour le stockage de données.
Stockage de données : Stockez une quantité infinie de données dans un bucket. Chargez autant d'objets que vous le souhaitez dans un compartiment Amazon S3. Chaque objet peut contenir jusqu'à 5 To de données. Chaque objet est stocké et récupéré à l'aide d'une clé unique attribuée par le développeur.
- **Téléchargement de données** : Téléchargez vos données ou permettez à d'autres de le faire. Téléchargez vos données à tout moment ou permettez à d'autres de faire de même.

- **Autorisations** : Accordez ou refusez l'accès à d'autres personnes qui souhaitent charger ou télécharger des données dans votre compartiment Amazon S3. Accordez des autorisations de chargement et de téléchargement à trois types d'utilisateurs. Les mécanismes d'authentification peuvent aider à protéger les données contre les accès non autorisés.
- **Interfaces standard** : Utilisez des interfaces REST et SOAP basées sur des normes conçues pour fonctionner avec n'importe quelle boîte à outils de développement Internet.

3.5 Aws Aurora

Amazon Aurora est un moteur de base de données relationnelle qui associe la vitesse et la fiabilité des bases de données commerciales haut de gamme à la simplicité et la rentabilité des bases de données open source. Amazon Aurora MySQL offre des performances jusqu'à cinq fois supérieures à celles de MySQL sans nécessiter de modifications de la plupart des applications MySQL. De la même manière, Amazon Aurora PostgreSQL offre des performances jusqu'à trois fois supérieures à celles de PostgreSQL. Amazon RDS gère vos bases de données Amazon Aurora en prenant en charge les tâches chronophages telles que la mise en service, l'application des correctifs, la sauvegarde, la récupération, la détection des pannes, ainsi que les réparations. Vous payez un forfait mensuel pour chaque instance de base de données Amazon Aurora utilisée. Aucun coût initial ou engagement à long terme n'est requis.

3.5.1 Que signifie « des performances trois fois supérieures à celles de PostgreSQL et MySQL »

Amazon Aurora fournit des augmentations significatives des performances de PostgreSQL et MySQL en intégrant étroitement au moteur de base de données une couche de stockage virtualisée basée sur SSD, conçue principalement pour les charges de travail des bases de données, ce qui permet de réduire les opérations d'écritures dans le système de stockage, minimiser la contention de verrouillage et éliminer les retards créés par les threads de processus de la base de données

3.6 Amazon Cognito

Amazon Cognito permet d'ajouter facilement et rapidement l'inscription et la connexion des utilisateurs ainsi que le contrôle d'accès aux applications Web et mobiles. Amazon Cognito s'adapte à des millions d'utilisateurs et prend en charge la connexion avec les fournisseurs d'identité sociale tels qu'Apple, Facebook, Google et Amazon. Amazon Cognito prend en charge les normes de gestion des identités et des accès, telle que OAuth 2.0.

3.7 API Gateway

Amazon API Gateway est un service entièrement opéré, qui permet aux développeurs de créer, publier, gérer, surveiller et sécuriser facilement des API à n'importe quelle échelle. Les API servent de « porte d'entrée » pour que les applications puissent accéder aux données, à la logique métier ou aux fonctionnalités de vos services backend. À l'aide d'API Gateway, vous pouvez créer des API RESTful et des API WebSocket qui permettent de

concevoir des applications de communication bidirectionnelle en temps réel. API Gateway prend en charge les charges de travail conteneurisées et sans serveur, ainsi que les applications web.

API Gateway gère toutes les tâches liées à l'acceptation et au traitement de plusieurs centaines de milliers d'appels d'API simultanés, notamment la gestion du trafic, la prise en charge de CORS, le contrôle des autorisations et des accès, la limitation, la surveillance et la gestion de la version de l'API. Aucuns frais minimum ou coûts initiaux ne s'appliquent à API Gateway. Vous payez pour les appels d'API que vous recevez et la quantité de données transférées et, avec le modèle de tarification par paliers de l'API Gateway, vous pouvez réduire vos coûts en fonction de l'utilisation de votre API

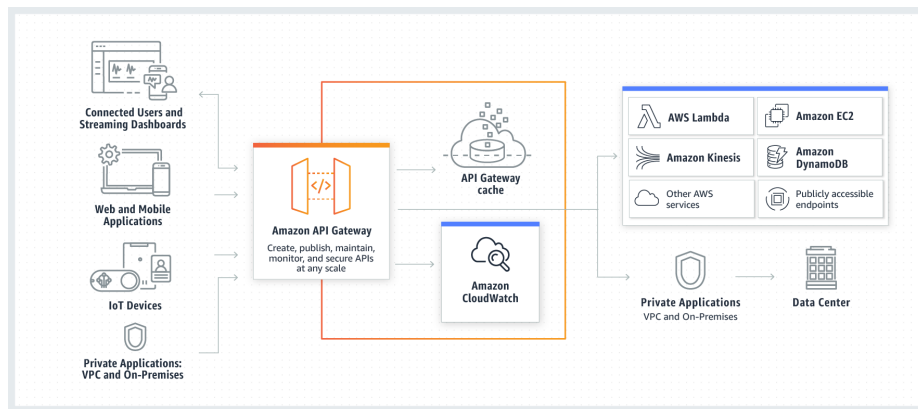


FIGURE 3.3 – API Gateway

3.8 Amazon DynamoDB

Amazon DynamoDB est une base de données NoSql de type clé-valeur et de documents, offrant des performances de latence de l'ordre de quelques millisecondes, quelle que soit l'échelle. Il s'agit d'une base de données multi-région, multi-active et durable entièrement gérée, avec des systèmes intégrés de sécurité, de sauvegarde, de restauration et de mise en cache en mémoire pour les applications à l'échelle d'Internet. DynamoDB peut traiter plus de 10 mille milliards de demandes par jour et supporte des pics de 20 millions de demandes par seconde.

La plupart des entreprises du monde qui connaissent la croissance la plus rapide, comme Lyft, Airbnb et Redfin, ainsi que Samsung, Toyota et Capital One s'appuient sur la mise à l'échelle et les performances de DynamoDB pour prendre en charge leurs charges de travail stratégiques. Des centaines de milliers de clients AWS ont choisi DynamoDB comme base de données de clés-valeurs et de documents pour leurs applications mobiles, Web, de jeux, de technologie publicitaire, IoT, etc. nécessitant un accès à faible latence aux données, quelle que soit l'échelle

3.9 CloudWatch

Amazon CloudWatch est un service de surveillance et d'observabilité conçu pour les ingénieurs DevOps, les développeurs, les ingénieurs en fiabilité de sites (SRE) et les responsables informatiques.

CloudWatch collecte des données de surveillance et opérationnelles sous forme de journaux, de métriques et d'événements. Ensuite, il les visualise à l'aide de tableaux de bord automatisés pour vous permettre d'avoir une appréciation unifiée de vos ressources, applications et services AWS opérationnels sur AWS et sur site.

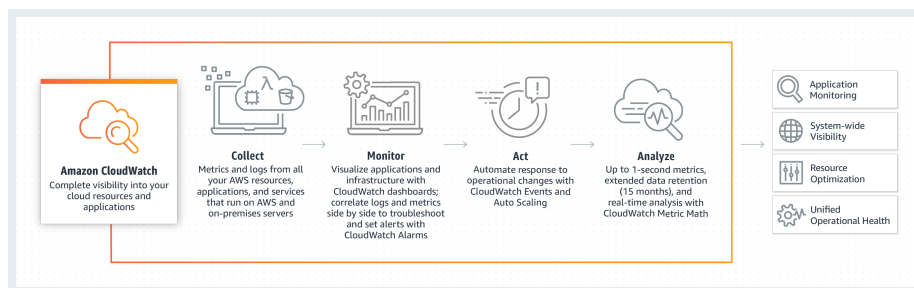


FIGURE 3.4 – CloudWatch

3.10 Reflex

ReFlex est un système de souscription automatisé modulaire. Il ne s'agit pas d'une application(service web) autonome, elle doit être intégrée au paysage applicatif du client. Les composants ReFlex requis sont hébergés dans l'environnement du client. Chaque instance de Reflex est fournie avec une base de connaissance paramétrée en fonction des produits d'assurance du client(entreprise d'assurance). Les principaux modules de Reflex sont :

- **CEP** : Customer Experience Platform
- **RAS** : Risk Assessment Service
- **DCS** : Document Creation Service

3.10.1 Scénario d'intégration CEP

En incluant CEP dans le système, toutes les demandes adressées au composant RAS sont filtrées par le backend CEP. Et avant que l'évaluation des risques puisse être lancée, il doit y avoir une toute première étape d'initialisation pour initialiser l'utilisation du CEP. Cette étape comprend l'appel au service d'intégration, qui fait partie du backend CEP, pour créer un jeton Web JSON (JWT) qui sera transmis entre l'interface utilisateur et le backend pour identifier et autoriser l'utilisateur actuel.

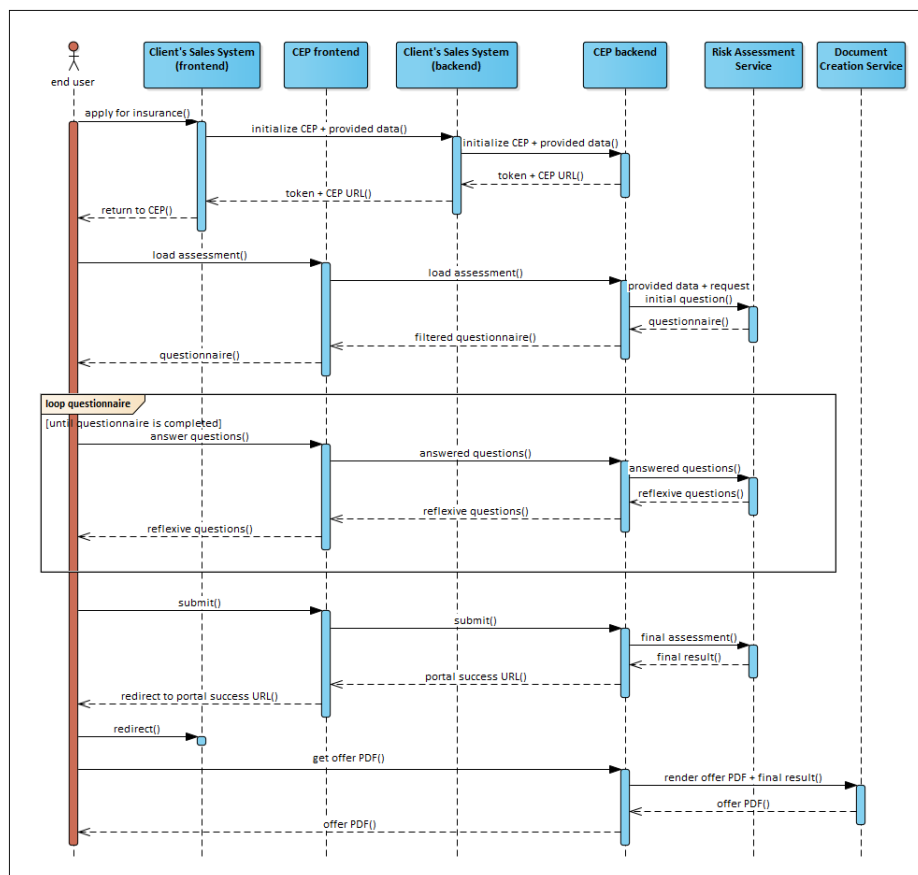


FIGURE 3.5 – Scénario d'intégration CEP

Chapitre 4

Approche suivie et solution proposée

Dans cette section nous ferons une présentation détaillée des solutions que nous avons proposées par rapport à nos missions.

4.1 Méthodologie de travail : Scrum avec Agile

L'agrandissement de l'équipe tech et le besoin de fournir des livrables réguliers pour montrer l'évolution aux fonds d'investissement a nécessité la mise en place d'une nouvelle méthodologie de travail avec la méthode Agile Scrum.

La méthode Agile est organisée en cycles de développements itératifs qui place le produit avant le projet. L'objectif n'est pas de terminer le projet mais de mettre en place un produit fini qui répond à toutes les exigences du client. Cette méthode part du principe que les besoins ne sont pas figés et peuvent évoluer avec le projet.

L'objectif final n'est pas défini au départ du projet mais des objectifs sont définis à court terme et fonctionne par étape. Une fois atteinte, un point est fait, les améliorations possibles sont mises en avant et l'étape suivante est définie en conséquence. Ce processus est ensuite répété jusqu'à l'obtention du produit final répondant à toutes les attentes. Avec ce fonctionnement, le client devient acteur du projet et peut être sollicité à chaque étape. Chaque étape doit durer 2-3 semaines maximum, comprenant des phases de test et le produit en sortie est incomplet mais doit être fonctionnel.

La méthodologie Scrum reprend le cadre de travail Agile pour des projets plus complexes. Le projet est décomposé en «sprints» qui sont les étapes de développement. Afin de produire des livrables régulièrement et rapidement, nos sprints durent une semaine. Une réunion est mise en place à chaque début de sprint afin de définir les objectifs de la semaine et leurs priorités. Durant toute la durée du sprint, une réunion quotidienne d'une durée de 30 minutes, « standup », a lieu, dans l'idée les participants restent debout pour éviter de s'éterniser. Chaque participant prend la parole afin de présenter son travail de la veille, valider les objectifs terminés et définir ceux du jour. Cette courte réunion doit servir à mettre en avant les points de blocage et synchroniser tous les membres de l'équipe. Le Scrum Master, chargé de vérifier le bon déroulement du sprint, doit valider avec l'équipe si les délais peuvent toujours être tenus et si le sprint est toujours réalisable dans son intégralité.

Chaque fin de sprint comprend un point sur tous les objectifs validés ainsi qu'une démonstration du travail réalisé durant la semaine. Dans notre cas, un test de l'application est effectué afin d'obtenir des retours de tous les membres de l'équipe tech. Ces remarques seront ensuite prises en compte ou non dans le prochain sprint.

4.1.1 Les principaux rôles de la méthodologie Scrum

- **Product Owner** : il est chargé de la vision du produit à réaliser et de définir les objectifs prioritaires.
- **Scrum Master** : membre de l'équipe chargé de vérifier la bonne application de la méthodologie de travail, il anime généralement les réunions quotidiennes. Ce rôle n'est pas figé et peut passer d'un membre à l'autre au sein de l'équipe.
- **L'équipe de développement** : équipe chargée de réaliser les objectifs définis dans le sprint et se composant généralement de profil différent

4.2 Les différentes missions et solutions

4.2.1 Mission 1 : Etude de faisabilité sur le déploiement d'une application Front-End dans un environnement AWS

L'application frontend (web app flutter) de Gighamesh est hébergée sur Firebase, tandis que l'essentiel de ses ressources se trouve dans le cloud d'Amazon. C'est dans le but d'unifier nos environnements cloud que cette mission m'avait été confiée.

Observations

Les applications web Flutter sont développées avec le langage de programmation Dart. Avant le déploiement d'une application Flutter un build est nécessaire, et le résultat de celui-ci est un ensemble de fichiers statiques (html, css, js, ...). Ce sont ces fichiers statiques qui sont déployés. Pour résoudre j'avais exploré deux possibilités :

Solutions

Solution1 : L'utilisation de S3 comme repos de d'hébergement La solution ici consiste à créer un repos S3, de le rendre public, afin de pouvoir accepter tous les trafics. De configurer Code Pipeline pour les besoins de CI/CD avec Github, De configurer Cloudfront pour la gestion du trafic et Route 54 pour le routage.

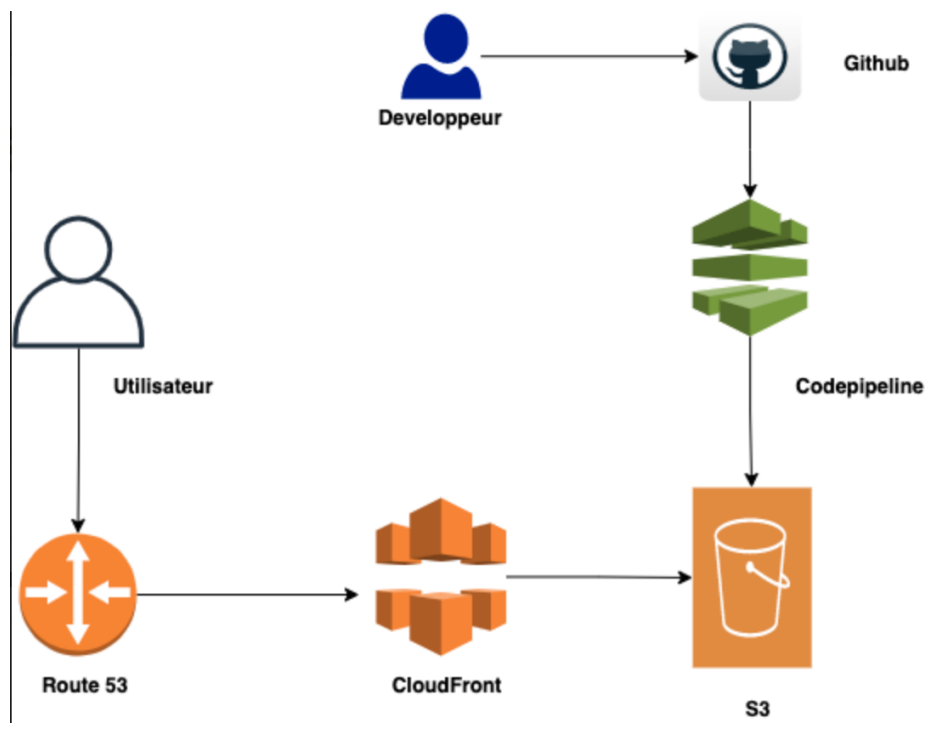


FIGURE 4.1 – Solution à base de S3

Solution2 : L'utilisation d'Amplify comme comme repos d'hébergement Cette solution consiste à utiliser le service d'hébergement qu'offre Amplify et son système de CI/CD en l'associant au système de versioning : Github. J'avais utilisé CloudFront pour la gestion du trafic et Route 54 pour le routage. NB : C'est la solution 2 qui avait été retenue. Celle-ci est simple et plus adaptée.

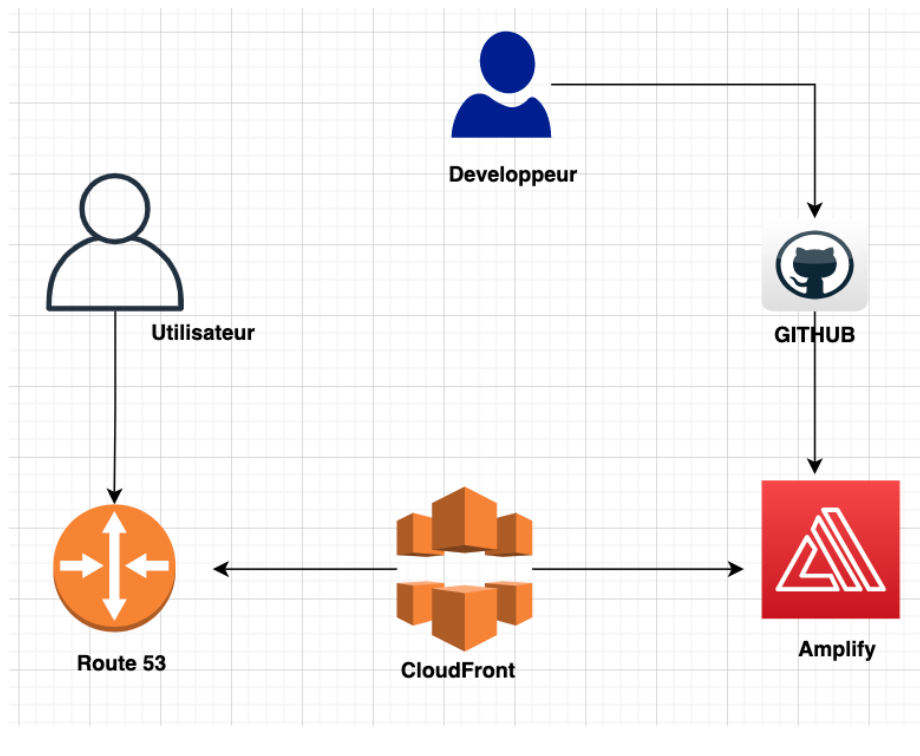


FIGURE 4.2 – Solution à base d'Amplify

4.2.2 Mission 2 :Le déploiement et l'intégration du service web (Reflex) dans le parcours de souscription

La souscription au produit d'assurance Assurly est subdivisée en trois parcours : P1, P2, et P3. Un utilisateur ne peut se retrouver que dans un seul parcours en fonction des données fournies. Si un utilisateur se retrouve dans le parcours P3 un questionnaire plus complexe est nécessaire, c'est à cet instant qu'intervient Reflex.

Solutions

Pipeline de déploiement

Quand nous recevons de notre partenaire des mises à jour des modules Reflex , nous les déployons(push) sur un repos Github. Notre repos Github est connecté à notre docker hub ; ce qui déclenche automatiquement la construction et le déploiement(push) de l'image de notre conteneur sur docker hub. Pour déployer Reflex sur notre EC2 : Il suffit soit de récupérer(pull) le contenu du repos Github puis construire l'image et déployer le conteneur ou de récupérer(pull) l'image de docker hub puis déployer le conteneur. La figure ci-dessous illustre le processus.

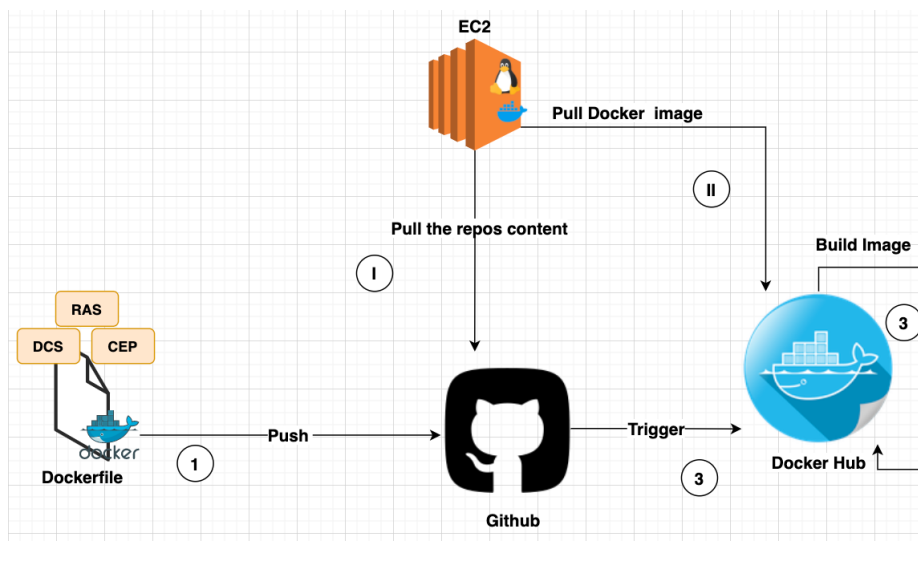


FIGURE 4.3 – Pipeline de déploiement de Reflex

```

1 FROM tomcat:latest
2 MAINTAINER Assurly dieumerci.kimpolo@assurly.com
3 RUN apt-get update
4 RUN apt-get -y upgrade
5 RUN apt-get -y install vim
6
7 RUN mkdir -p /usr/local/tomcat/reflex-config
8 RUN mkdir -p /usr/local/tomcat/logs
9 RUN mkdir -p /usr/local/tomcat/webapps/continue
10 RUN mkdir -p /heap.dump.dir
11
12
13 ENV REFLEX_CONFIG_ROOTDIR /usr/local/tomcat/reflex-config
14 ENV REFLEX_LOG_ROOTDIR /usr/local/tomcat/logs
15 ENV CATALINA_OPTS '-Xms512M -Xmx4096M -Dfile.encoding=UTF-8 -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/heap.dump.dir'
16
17 COPY ./Reflex/Assurly/webapps/ras-web.war /usr/local/tomcat/webapps/
18 COPY ./Reflex/Assurly/webapps/cep-core.war /usr/local/tomcat/webapps/
19 COPY ./Reflex/Assurly/webapps/dcs.war /usr/local/tomcat/webapps/
20 COPY ./cep-ui-custom-1606905004848 /usr/local/tomcat/webapps/continue
21 COPY ./Reflex/Assurly/reflex-config /usr/local/tomcat/reflex-config
22
23 VOLUME /usr/local/tomcat/webapps
24 VOLUME /usr/local/tomcat/webapps/continue
25 VOLUME /heap.dump.dir
26
27 EXPOSE 8080
  
```

FIGURE 4.4 – Dockerfile de construction de l'image Reflex

4.2.3 Mission 3 : Etude de la sécurité actuelle d'accès aux API

Les API sont devenues incontournables, il existe très peu ou presque pas de systèmes informatiques qui ne consomment et/ou ne fournissent des API. Dans la plus part des cas ou très souvent on est soit consommateur, soit fournisseur ou les deux.

Comme les applications classiques, les API n'échappent pas aux problèmes de sécurité. Elles font face aux mêmes défis que les applications ordinaires. Elles sont vulnérables à plusieurs types d'attaques comme : les attaques Dos et attaques par pollutions des paramètres http,...

La question de sécurité des API comme pour des applications ordinaires doit être prise en considération durant tout le cycle son développement : de sa conception à sa mise en production.

Dans le cadre de mon étude je m'étais focalisé sur les API de type Rest conçues et déployées dans le cloud d'Amazon.

Dans l'environnement AWS les Backend des API sont protégés par le service API Gateway, et une très grande partie des protocoles de sécurité y sont implémentés.

Rappel des principes de sécurité

Un système sécurisé doit garantir les points suivants :

- **L'intégrité** : garantir que les données sont bien celles que l'on croit être
- **La disponibilité** : maintenir le bon fonctionnement du système d'information
- **La confidentialité** : rendre l'information inintelligible à d'autres personnes que les seuls acteurs d'une transaction

La sécurité des API dans AWS

Pour sécuriser une API dans l'environnement AWS il faut prendre en compte deux aspects suivants :

1. Les contrôles d'accès
2. L'authentification et les autorisations

Il n'y a pas de choix à faire entre les contrôles d'accès et le système d'authentification & d'autorisation, ils sont complémentaires.

Les contrôles d'accès

Les mécanismes suivants peuvent être utilisés pour effectuer d'autres tâches liées au contrôle de l'accès aux API dans AWS.

- **Le partage des ressources cross-origin (CORS)** permet de contrôler la façon dont votre API REST répond aux requêtes de ressources inter-domaines.
- **Les certificats SSL côté client** peuvent être utilisés pour vérifier que les requêtes HTTP adressées à votre système backend proviennent d'API Gateway.
- **AWS WAF** peut être utilisé pour protéger votre API d'API Gateway contre les menaces web courantes telles que l'injection SQL et les attaques XSS (cross-site scripting).
- **API Key** peut être utilisé pour gérer les quotas des appels de l'API
- **API endpoints ressources policy** peut être utilisé pour refuser certains appels en fonction des adresses IP

- **VPC endpoints policy** peut être utilisé pour gérer les endpoints accessibles seulement pour les adresse IP privées

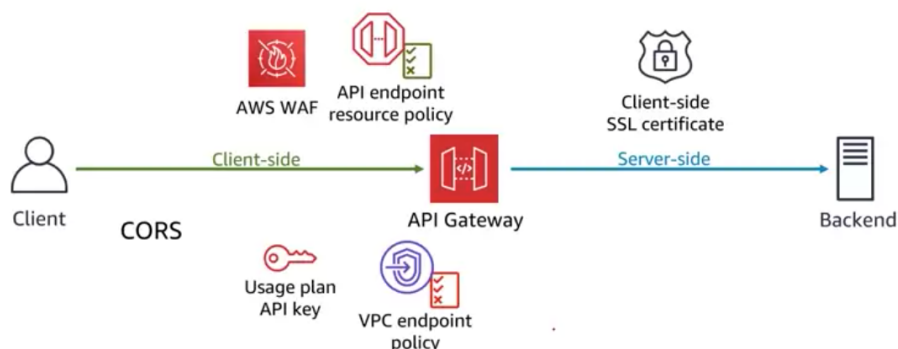


FIGURE 4.7 – Les contrôles d'accès

Authentification et autorisations

Le protocole de base de gestion des authentifications et des autorisations au niveau des API Rest est le protocole OAuth2, décrit dans la section état de l'art. Aws utilise essentiellement le service Cognito pour gérer les questions d'autorisation et d'authentification sur des API. Cognito dans son fonctionnement suit les principes de base du protocole OAuth2.

L'existant

Points positifs

L'Api est accessible en https, les CORS et les contrôles d'accès par API Key sont implémentés sur l'API Gateway.

Points négatifs

Le service Cognito est implémenté mais consommé via de fonctions Lambda personnalisées, le service Cognito implémenté et consommé via de fonctions Lambda personnalisées, les données des utilisateurs filtrées avec des paramètres autres le token d'accès valide aussi un chiffrement est implémenté sur les applications frontend en plus du ssl.

Observations

1. Cognito à la base ne peut supporter qu'un seul user pool (support de stockage des utilisateurs) sur une API
2. L'API est type Rest et déployé dans le cloud d'Amazon
3. Les utilisateurs stockés dans plusieurs users pool
4. L'API est consommée par des applications web et mobiles

Pour supporter plusieurs user pools sur nos API, nous avons développé un système d'autorisation personnalisé à base d'une Lambda. Dans notre implémentation nous avons exploré deux flows : l'implicit grant(en utilisant la web UI de Cognito) et password grant.

Implicit grant

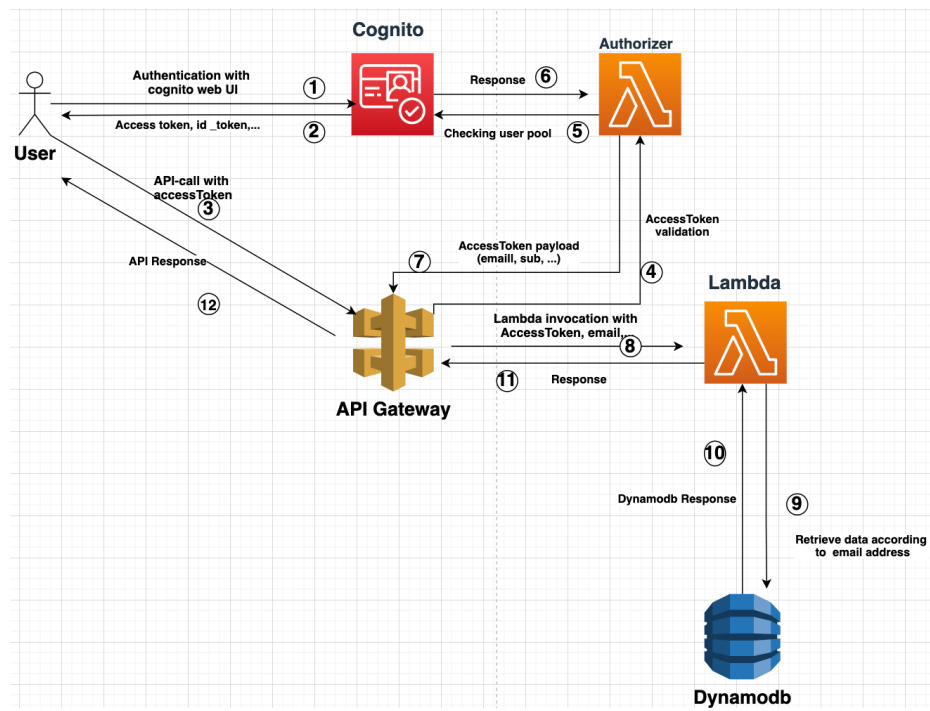


FIGURE 4.8 – L'architecture du système de sécurité, Implicit grant

4.2.4 Mission 4(Mise en place d'un ETL d'intégration de données)

Gigamesh disposait une quantité importantes des données stockées dans des fichiers txt; des données nécessaires pour effectuer des campagnes de marketing ciblées. Ma mission consistait à intégrer ces données dans une base de données sql pour pouvoir y effectuer facilement des requêtes.

Observations

1. Tous les fichiers avaient la même structure
2. Le caractère | était utilisé comme séparateur des colonnes
3. Les libellés des colonnes étaient de fois des phrases
4. Les données n'étaient pas typées

Solution

Pour résoudre le problème j'avais mise en place un **ETL**, qui récupère les données brutes(fichiers txt) de leur support de stockage, les transforme avec un **script Python** et les stocke dans base de données Mysql dans l'environnement AWS via le service Aurora.

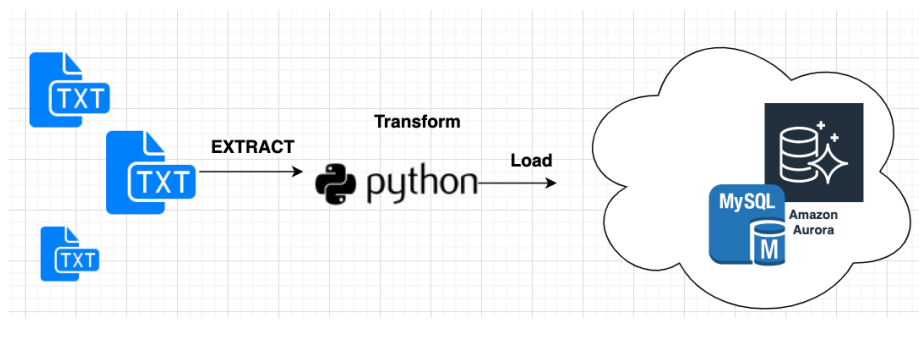


FIGURE 4.9 – Etl d'intégration de données

La partie de transformation de mon ETL m'avait servi à la création de la table où sont stockées les données et à leur typage.

```

req_nombre =pd.read_sql("select count(id) from xxxx", cnx)
req_nombre

```

count(id)	
0	17340033

FIGURE 4.10 – La taille de la table où sont stockées nos données

Conclusion

Le stage que j'ai passé chez GigaMesh était très bénéfique pour moi. C'était non seulement un moment d'apprentissage mais aussi d'application. Grâce à ce stage j'ai pu découvrir le domaine de l'assurance emprunteur, de travailler avec la méthodologie agile et scrum, de renforcer et d'acquérir des nouvelles compétences dans le domaine de Cloud.

Ce stage était aussi un moment de confronter mes compétences acquises à l'école à la réalité des entreprise.

J'ai également par ce stage eu la chance de déployer et d'intégrer le service web Reflex, d'implémenter les protocoles de sécurité sur l'API Gateway, de déployer des applications web dans le cloud d'Amazon et d'intégrer des données sur Mysql dans le service Aurora d'Amazon.

Bibliographie

1. Rapport de stage de GUYOT Antoine
2. La web doc d'AWS