

Cloud computing : étude & développement d'une application Serverless

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Jack BARKER

Conseiller au travail de Bachelor :

Rolf HAURI

Genève, 1^{er} juillet 2021

Haute École de Gestion de Genève (HEG-GE)

Filière Informatique de Gestion

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor HES

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : <https://www.orkund.com>.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 1^{er} juillet 2021

Jack Barker



Remerciements

La réalisation de ce travail de bachelor a été possible grâce à l'aide et au soutien de plusieurs personnes à qui je témoigne toute ma gratitude.

M. Rolf Hauri, maître d'enseignement HES, pour ses conseils judicieux et ses retours pertinents sur mon travail.

Krittiya Noipheng, pour son aide et ses conseils lors des révisions de ce mémoire.

Dejan Munjin, pour son assistance durant la phase initiale de recherche de sources sur le sujet du cloud computing.

Et le plus grand des mercis à ma mère pour son soutien constant.

Résumé

Ce mémoire a pour objectif d'étudier les concepts liés au cloud computing – un modèle de mise à disposition de ressources informatiques via le réseau par un fournisseur externe et selon un tarif lié à la l'utilisation – du point de vue d'un développeur.

Ce travail présente une définition du cloud computing et explore des concepts généraux tels que les types de services, les modes de déploiement, ainsi que les bénéfices et risques liés à ce modèle.

Le sujet des applications Serverless – un modèle d'application “sans serveur” faisant partie du paradigme du cloud computing – est abordé par la présentation d'un développement de prototype d'application web de messagerie instantanée.

Table des matières

Déclaration.....	i
Remerciements	ii
Résumé	iii
Table des matières.....	iv
Liste des tableaux	vii
Liste des figures.....	vii
1. Introduction.....	1
2. Serverless, l'informatique sans serveur.....	2
2.1 Les besoins côté serveur d'une application web	2
2.2 Avantages des applications sans serveur	3
3. Cloud computing : définition et concepts	3
3.1 Définition du cloud computing.....	4
3.2 Modèles de services	5
3.2.1 Software as a Service (SaaS)	6
3.2.2 Platform as a Service (PaaS)	6
3.2.3 Infrastructure as a Service (IaaS).....	6
3.3 Modes de déploiement.....	7
3.3.1 Cloud public	7
3.3.2 Cloud privé.....	7
3.3.3 Cloud hybride.....	7
3.4 Bénéfices et risques du cloud computing.....	8
3.4.1 Bénéfices	8
3.4.2 Réduction des coûts liés à l'infrastructure informatique	9
3.4.3 Flexibilité et scalabilité des ressources IT	9
3.4.4 Augmentation des capacité liées au cœur de métier	10
3.5 Risques.....	10
3.5.1 Risques liés à la sécurité.....	11

3.5.2	Réduction de la gouvernance opérationnelle.....	11
3.5.3	Portabilité limitée entre les fournisseurs de services cloud.....	11
4.	Les principaux fournisseurs de cloud computing public	12
4.1	Amazon Web Services	13
4.2	Microsoft Azure	14
4.3	Google Cloud Platform	15
5.	Développement d'un prototype Serverless avec AWS.....	15
5.1	Mise en place du projet.....	16
5.1.1	Accès à la console de gestion	16
5.1.2	Création d'un compte administrateur avec le service IAM	16
5.1.3	Le Serverless selon AWS.....	17
5.1.4	Architecture du prototype	18
5.2	Intégration continue du client	19
5.2.1	Gestion de versions	19
5.2.2	Compilation	19
5.2.3	Déploiement.....	20
5.3	Base de données.....	23
5.3.1	Services AWS de base de données	23
5.3.2	Création de tables DynamoDB	25
5.4	Tables DynamoDB du projet	27
5.5	Passerelles API	29
5.5.1	API Websocket	30
5.5.2	API HTTP	31
5.5.3	Déploiement de l'API et intégration à l'application cliente	32
5.5.4	Gestion des versions des API	34
5.6	Fonctions sans serveur	34
5.6.1	Fargate, alternative à Lambda.....	34
5.6.2	Création et configuration d'une fonction Lambda	35
5.6.3	Exemples de fonctions Lambda	37

5.6.4	Gestion des versions des fonctions Lambda	39
5.7	Surveillance des services.....	40
5.8	Authentification.....	42
5.8.1	User Pools	42
5.8.2	Enregistrement & connexion des utilisateurs à l'application cliente	43
5.8.3	Authentification des requêtes envoyées à API Gateway.....	44
5.9	Stockage de fichiers	46
5.10	Conclusion	49
Bibliographie		51
Annexe 1 : Code source de l'application		56
Annexe 2 : Captures d'écran de l'interface du prototype développé		57
Annexe 3 : Services AWS		59
Annexe 4 : Services Google Cloud Platform		60
Annexe 5 : Infrastructure mondiale AWS.....		61
Annexe 6 : Infrastructure mondiale Microsoft Azure		62
Annexe 7 : Infrastructure mondiale GCP		63

Liste des tableaux

Tableau 1 : Tables DynamoDB pour une application de messagerie.....	28
Tableau 2 : Routes de l'API WebSocket	30
Tableau 3 : Routes de l'API HTTP.....	31

Liste des figures

Figure 1 : IaaS, PaaS et SaaS	5
Figure 2 : Résultat du sondage mené par l'ENISA	8
Figure 3 : Le quadrant des fournisseurs cloud selon Gartner	12
Figure 4 : Création d'un utilisateur IAM.....	17
Figure 6 : Architecture du prototype	18
Figure 7 : Contenu du fichier buildspec.yml.....	20
Figure 8 : Vue d'ensemble du pipeline d'intégration continue	21
Figure 9 : Création d'un compartiment S3	22
Figure 10 : Détails du compartiment.....	22
Figure 11 : Configuration du compartiment pour l'hébergement d'un site	23
Figure 12 : Moteurs de base de données disponibles avec le service RDS.....	24
Figure 13 : Images disponibles lors de la création d'une instance EC2	25
Figure 14 : Exemple de création de table DynamoDB	26
Figure 15 : Options d'allocation de capacité d'écriture et de lecture	26
Figure 16 : Modèle de données.....	28
Figure 17 : Architecture API Gateway.....	29
Figure 18 : URL d'invocation et indicateurs d'activités de l'API déployée	33
Figure 19 : Code exécutant une requête vers l'URL d'invocation de l'API	33
Figure 20 : Options de création d'une fonction	35
Figure 21 : Configuration d'une fonction.....	36

Figure 22 : Déclencheur et destination de fonction.....	36
Figure 23 : Editeur de code en ligne.....	37
Figure 24 : Code source – création d’une conversation.....	38
Figure 25 : Utilisation du DocumentClient de l’API DynamoDB	39
Figure 26 : Gestion des versions de fonctions.....	40
Figure 27 : Rôle de création de logs.....	40
Figure 28 : Exemple de log d’erreur Lambda.....	41
Figure 29 : Exemple de log API Gateway	41
Figure 30 : User Pool, liste des utilisateurs.....	42
Figure 31 : Variables d’environnement Cognito	43
Figure 32 : Enregistrement d’un utilisateur depuis l’application Angular	44
Figure 33 : Création d’un mécanisme d’autorisation de type Cognito	45
Figure 34 : Récupération du jeton d’identité Cognito de l’utilisateur.....	45
Figure 35 : Envoi de l’en-tête d’authentification	46
Figure 36 : Accès publics au compartiment interdits par défaut.....	46
Figure 37 : Police d’autorisation accordée à l’utilisateur IAM	47
Figure 38 : Récupération d’une URL pré-signée.....	48

1. Introduction

Ce mémoire a pour objectif d'étudier le sujet du cloud computing du point de vue d'un développeur par la conception d'un prototype d'application Serverless, ou sans serveur.

La première partie de travail présente brièvement le concept des applications Serverless, un modèle d'application basé sur le cloud computing.

Puis ce travail présente une définition et des concepts généraux liés au cloud computing, tels que les modèles de services, les modes de déploiement, ainsi que les bénéfices et risques pour les entreprises.

Ensuite, les leaders du marché du cloud public – Amazon Web Services, Microsoft Azure et Google Cloud Platform – sont présentés.

La mise en œuvre d'un prototype d'application web de messagerie instantanée basée sur les principes Serverless est ensuite décrite. Cette application utilise les services du fournisseur de cloud computing Amazon Web Services.

Enfin, des conclusions sont présentées sur les sujets abordés : le cloud computing dans son ensemble et l'expérience de développement d'une application sans serveur.

2. Serverless, l'informatique sans serveur

Définissons d'abord le concept Serverless, ou l'informatique sans serveur. Selon le site de Red Hat, un des acteurs de l'industrie du cloud : « *Le serverless est un modèle de développement cloud-native¹ qui permet aux développeurs de créer et d'exécuter des applications sans avoir à gérer des serveurs.* » [1]

Le terme « sans serveur » ne veut pas dire qu'aucun serveur n'est nécessaire au fonctionnement de l'application, mais plutôt que la gestion de ce ou ces serveurs n'est pas prise en charge par le développeur mais par un fournisseur de services cloud.

L'informatique sans serveur est un concept étroitement lié au cloud computing, un modèle de fourniture de ressources informatiques décrit dans la prochaine partie de ce travail.

Le concept Serverless est aussi lié à celui de l'architecture client-serveur. Du point de vue d'un développeur, une application peut être considérée comme deux logiciels distincts :

- le logiciel client auquel l'utilisateur a accès (par exemple une application web) ;
- le logiciel serveur qui reçoit les requêtes du client, exécute une certaine logique et renvoie une réponse au client.

Dans le modèle Serverless, le développeur conçoit un logiciel client et les besoins côté serveur sont satisfaits par des services fournis par un fournisseur cloud.

2.1 Les besoins côté serveur d'une application web

Dans une architecture client-serveur, le serveur peut avoir une multitude de responsabilités. Prenons l'exemple du prototype développé dans le cadre de ce travail : une application de messagerie instantanée.

On peut estimer que les besoins d'une telle application pour la partie serveur sont les suivants :

- Communication : le serveur peut recevoir des requêtes du client et lui renvoyer une réponse ;

¹ Par cloud-native, on entend une application développée en vue d'être exécutée sur l'infrastructure d'un fournisseur cloud [2].

- Calcul : le serveur dispose d' une capacité de calcul pour traiter les requêtes reçues ;
- Authentification : le serveur peut identifier les utilisateurs et authentifier leurs requêtes afin de sécuriser l'application ;
- Stockage de données : le serveur peut stocker des données – par exemple, les messages et préférences des utilisateurs – dans une base de données afin de les transmettre au logiciel client lorsque nécessaire.

Il existe de nombreuses autres responsabilités qu'un serveur peut être appelé à assumer dans le contexte d'une application web, et je n'en ai cité ici que quelques-unes.

2.2 Avantages des applications sans serveur

Selon le site d'Azure, la plateforme cloud de Microsoft, les avantages du modèle Serverless sont les suivants [3] :

- Pas de gestion d'infrastructure : le modèle Serverless permettrait aux développeurs de se concentrer sur le développement de logiciel et non sur l'infrastructure ;
- Scalabilité dynamique : le fournisseur est responsable d'augmenter les ressources informatiques disponibles du côté serveur selon la charge d'utilisation afin de garantir une bonne performance ;
- Rapidité de développement d'application : puisque le fournisseur est en charge de l'infrastructure et propose des services cloud préexistants, ainsi le développeur pourrait plus rapidement créer et déployer une application ;
- Utilisation plus efficiente des ressources : l'adoption d'un modèle Serverless permettrait d'optimiser l'allocation des ressources et de ne payer que pour les ressources réellement utilisées.

3. Cloud computing : définition et concepts

Cette partie du mémoire a pour but de d'abord définir le terme cloud computing, puis d'explorer des concepts associés : les modèles de services et leurs modes de déploiement. Enfin, certains bénéfices et risques liés à l'adoption du cloud computing sont discutés.

3.1 Définition du cloud computing

Azure définit sur son site le cloud computing de la manière suivante :

« la fourniture de services informatiques (notamment des serveurs, du stockage, des bases de données, la gestion réseau, des logiciels, des outils d'analyse, l'intelligence artificielle) via Internet (le cloud) dans le but d'offrir une innovation plus rapide, des ressources flexibles et des économies d'échelle. » [4]

Il s'agit donc d'un modèle de fourniture de services informatiques permettant aux entreprises d'accéder à des ressources ou services en tout genre sans devoir posséder et maintenir soi-même des serveurs ou des centres de données. Le cloud computing n'est donc pas en soi une nouvelle technologie, mais plutôt une nouvelle manière d'organiser les systèmes informatiques.

Le concept de la consommation de ressources informatiques à la demande a été suggéré publiquement peut-être pour la première fois en 1961 par John McCarthy, chercheur en informatique, lors d'un discours à l'occasion du centenaire du Massachusetts Institute of Technology (MIT) :

« Le traitement informatique pourrait un jour être organisé tel un service d'utilité publique comme le système de téléphonie. Chaque abonné paie seulement pour la charge qu'il utilise, mais a accès à tous les langages de programmation caractéristiques des très gros systèmes. [...] Un tel service informatique pourrait devenir la base d'une nouvelle et importante industrie. » (traduction libre) [5]

Le terme cloud computing aurait été popularisé en 2006 lorsque Google a commencé à utiliser ce terme pour décrire un nouveau paradigme qui devait permettre aux utilisateurs d'accéder à leurs ressources informatiques par le web plutôt que par leur ordinateur personnel directement [6].

Selon la définition formulée par le National Institute for Standards and Technology en 2011, le cloud computing est défini selon cinq critères essentiels [7] :

- Self-service à la demande : Le consommateur du service peut lui-même provisionner unilatéralement les ressources informatiques dont il a besoin, sans avoir recours à une interaction humaine ;
- Accès par le réseau : Les ressources informatiques sont accessibles par le réseau informatique via des mécanismes standards et utilisables par des clients hétérogènes ;
- Mise en commun des ressources : Les ressources physiques ou virtuelles du fournisseur de service sont mises en commun afin de servir plusieurs clients

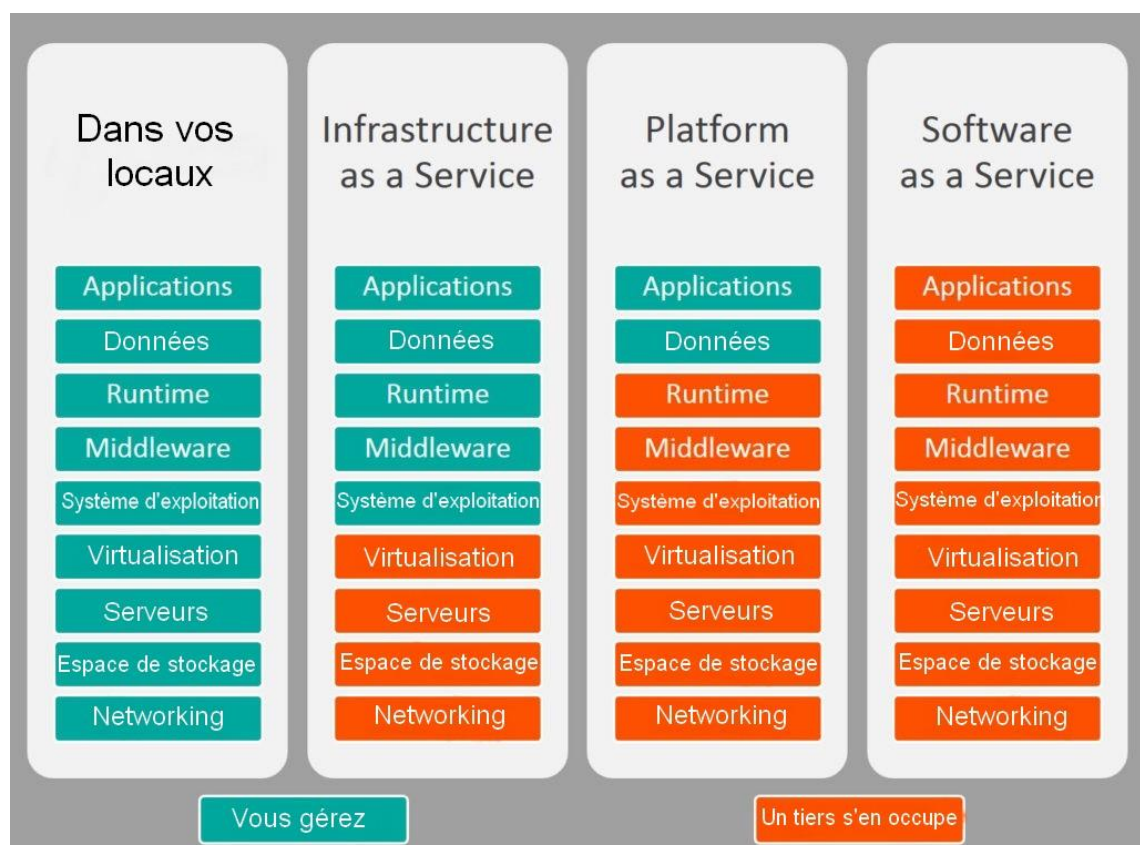
dans un modèle multi-tenant. Les ressources sont dynamiquement assignées et libérées selon la demande des clients ;

- Élasticité rapide : Les ressources utilisables par les clients peuvent être provisionnées et libérées manuellement ou automatiquement afin de monter en puissance pour répondre à la demande des clients ;
- Services mesurés : La consommation des ressources par un client est mesurée par une métrique selon le type de ressource, permettant au fournisseur de service d'optimiser son service et de facturer le client proportionnellement à l'utilisation des ressources.

3.2 Modèles de services

Les services de cloud computing sont généralement catégorisés selon les modèles suivants : Software as a Service, Platform as a Service et Infrastructure as a Service [8].

Figure 1 : IaaS, PaaS et SaaS



Source : [9]

On peut voir sur cette figure que ces modèles diffèrent selon la pile de technologie prise en charge par le fournisseur.

3.2.1 Software as a Service (SaaS)

Dans le modèle SaaS, le service offert par le fournisseur prend la forme d'un logiciel applicatif, prêt à l'emploi. Celui-ci est exécuté et hébergé sur l'infrastructure du fournisseur et rendu disponible au consommateur du service via le réseau par une interface web [8].

Toute la pile de technologie nécessaire au fonctionnement de l'application est gérée par le fournisseur cloud. Les services SaaS s'adressent normalement aux utilisateurs finaux des logiciels. Le fournisseur de service gère l'application dans son entièreté (développement, maintenance, administration, hébergement, etc.) [10].

On peut considérer Google Drive, la suite bureautique et de stockage de fichiers de Google, comme un exemple de service SaaS.

3.2.2 Platform as a Service (PaaS)

Le modèle PaaS regroupe les services fournissant aux clients du fournisseur cloud une plateforme technique, soit une pile de plusieurs technologies sur laquelle ils peuvent développer, déployer et administrer leurs propres applications spécifiques [8].

Les services PaaS sont particulièrement adaptés pour les entreprises qui souhaitent rapidement déployer des développements informatiques de manière autonome, sans devoir se soucier de la maintenance de la plateforme technique en elle-même.

Heroku, une plateforme permettant de déployer et exécuter des applications, est un exemple de service PaaS [11].

3.2.3 Infrastructure as a Service (IaaS)

Dans le modèle IaaS, ce sont les éléments fondamentaux des infrastructures informatiques qui sont fournies au client par le réseau : des machines (physiques ou virtuelles), des serveurs de base de données ou d'applications [8].

Autrement dit, le modèle IaaS propose des blocs élémentaires d'infrastructure informatique dans le but d'offrir une plus grande flexibilité aux organisations clientes. Celles-ci peuvent provisionner uniquement les éléments d'infrastructure dont elles ont besoin et les configurer librement.

Cette flexibilité a souvent pour contrepartie de nécessiter des compétences liées à l'administration de systèmes d'exploitation, de serveurs ou de systèmes de sécurité selon les éléments provisionnés par le client [10].

Linode, une plateforme permettant l'exécution de machines virtuelles, est un exemple de service PaaS [12].

3.3 Modes de déploiement

Les services de cloud computing peuvent aussi être catégorisés selon leur mode de déploiement : public, privé ou hybride.

3.3.1 Cloud public

Un cloud public est un service cloud qui utilise des ressources communes pour servir de nombreux clients en même temps tout en isolant les données de ceux-ci [13].

Dans un cloud public, les composants du service fourni sont exécutés sur l'infrastructure de fournisseur de service. Le client du service ne possède aucune machine ni aucune application liée au service qu'elle utilise.

Ce modèle permet au client de maximiser l'avantage lié au fait qu'il n'a pas besoin de posséder lui-même l'infrastructure informatique. Il permet aussi au fournisseur cloud de maximiser son efficacité en faisant levier de la mutualisation des ressources informatiques et des coûts associés.

3.3.2 Cloud privé

Contrairement à un cloud public, un cloud privé est basé sur une infrastructure dédiée à un seul client, qui peut être située derrière un pare-feu interne de l'entreprise [14].

Dans un cloud privé, les ressources informatiques ne sont pas mutualisées et ne servent qu'un seul client. Ce modèle ne présente pas les mêmes avantages que le modèle public, puisqu'il induit généralement un coût plus élevé qu'un cloud public complètement externalisé [14].

Néanmoins, il peut présenter des avantages aux organisations intéressées par l'utilisation de ressources dédiées et un plus grand contrôle sur la gouvernance de leurs ressources informatiques.

3.3.3 Cloud hybride

Le mode de déploiement hybride est une combinaison des deux modes précédents : public et privé.

Il permet de faire le lien entre une infrastructure publique et une infrastructure privée par une connexion sécurisée [14]. Les ressources informatiques ou applications basées sur le cloud public d'un fournisseur peuvent être connectées avec les systèmes internes de l'entreprise sur son site.

Un consommateur de services cloud peut, par exemple, décider de déployer une application utilisant des données sensibles sur un cloud privé basé sur une infrastructure localisée sur son propre site, mais externaliser le reste de ses ressources informatiques sur une infrastructure distante selon un mode de déploiement public.

Les entreprises clientes de services cloud peuvent alors opter pour un mode hybride lorsqu'elles veulent bénéficier de certains services cloud, tout en maintenant le contrôle total sur une partie de leur système informatique interne.

3.4 Bénéfices et risques du cloud computing

Le cloud computing, comme tout autre paradigme d'organisation des systèmes informatiques, apporte son lot de bénéfices et de risques.

3.4.1 Bénéfices

De nombreux bénéfices sont perçus comme étant associés à l'adoption du cloud computing par les entreprises. Un sondage a été réalisé en 2009 par l'Agence européenne chargée de la sécurité des réseaux et de l'information (ENISA) auprès de 70 PME dans différents pays européens afin d'analyser les raisons de l'engagement des entreprises dans le domaine du cloud computing.

Figure 2 : Résultat du sondage mené par l'ENISA

What are the reasons behind your possible engagement in the Cloud Computing area?		
Answer Options	Response Percent	Response Count
Remove economic/expertise barriers impeding to modernize business processes by the introduction of Information Technology	30,6%	22
Avoiding capital expenditure in hardware, software, IT support, Information Security by outsourcing infrastructure/platforms/services	68,1%	49
Flexibility and scalability of IT resources	63,9%	46
Increasing computing capacity and business performance	36,1%	26
Diversification of IT systems	11,1%	8
Local and global optimisation of IT infrastructure through automated management of virtual machines	25,0%	18
Business Continuity and Disaster recovery capabilities	52,8%	38
Assessing the feasibility and profitability of new services (i.e. by developing business cases into the Cloud)	29,2%	21
Adding redundancy to increase availability and resilience	27,8%	20
Controlling marginal profit and marginal costs	15,3%	11
Other (please specify)	13,9%	10
answered question		72
skipped question		2

Source : [15]

Comme on peut le voir dans la figure ci-dessus, les raisons principales qui motivent l'adoption du cloud computing par les PME sont :

- L'évitement des investissements en capital dans l'infrastructure informatique ;
- La flexibilité et la scalabilité des ressources informatiques ;
- Les capacités liées à la continuité des opérations et à la reprise après sinistre ;

3.4.2 Réduction des coûts liés à l'infrastructure informatique

Selon une analyse des avantages liés au cloud computing publiée par les chercheurs Chen, Chuang et Nakatani publiée en 2016, un des bénéfices principaux du cloud computing perçu par beaucoup d'entreprises est la possibilité de réduire les coûts liés à l'infrastructure des systèmes informatiques [16].

En effet, le cloud computing permet aux entreprises d'accéder à de puissantes infrastructures informatiques sans devoir faire d'investissements conséquents dans ces infrastructures, que ce soit en investissement initial de capital ou en coûts d'exploitation.

Au lieu d'investir dans des centres de données avec des machines physiques et des ressources humaines spécialisées – c'est-à-dire capables d'installer, de configurer, de maintenir et d'exploiter les ressources informatiques nécessaires pour répondre aux besoins informatiques de l'entreprise – celle-ci peut opter pour l'offre d'un fournisseur de cloud public.

L'avantage économique principal du cloud computing est donc la possibilité pour les entreprises de remplacer les dépenses de capital – nécessaires pour l'infrastructure informatique – par des coûts variables proportionnels à l'utilisation des ressources.

Il faut noter que la migration vers une stratégie cloud n'est pas une garantie de réduction des coûts du système informatique d'une entreprise. Une entreprise devrait chercher des avantages au-delà des aspects financiers dans leur stratégie cloud [16].

L'externalisation des ressources informatiques vers un fournisseur de cloud peut être vue comme une conversion des coûts d'investissements fixes en infrastructure, en un coût opérationnel variable.

3.4.3 Flexibilité et scalabilité des ressources IT

IBM, fournisseur de services cloud, a publié en 2016 un rapport sur les bénéfices du cloud computing, annonçant que ce modèle permet l'optimisation des systèmes

informatiques d'entreprise d'un point de vue coût-bénéfice, grâce à la scalabilité des architectures cloud :

« Cela est possible car le cloud computing permet une scalabilité massive pour satisfaire les périodes de demandes accrues tout en évitant des périodes de sous-utilisation de capacités informatiques. Avec un clic de la souris, des services peuvent être rapidement étendus ou réduits sans nécessiter de changement majeur sur le centre de données. » (traduction libre) [17]

Ce potentiel d'optimiser l'utilisation de ressources informatiques – c'est-à-dire soit de monter en puissance avec la demande envers le système informatique, soit de libérer les ressources pendant des périodes de sous-utilisation – est un des bénéfices clés de l'adoption d'une stratégie cloud.

3.4.4 Augmentation des capacité liées au cœur de métier

Un avantage potentiel du cloud computing selon une analyse de 47 publications académiques sur le cloud computing publiée par les chercheurs Müller, Holm et Sondergaard en 2015, est la liberté de se focaliser sur ses activités principales plutôt que sur la gestion de son infrastructure informatique [18].

En effet, la gestion de l'infrastructure étant prise en charge par le fournisseur cloud, l'entreprise cliente disposerait de plus de ressources pour investir dans ses activités principales.

3.5 Risques

De nombreux risques ont été associés avec l'adoption de la stratégie cloud par des chercheurs et professionnels. La compréhension, l'évaluation et la mitigation des risques sont essentielles lorsqu'il s'agit d'adopter une stratégie cloud [19].

Un article publié dans le journal de l'ISSA (Information Systems Security Association) en 2011 catégorise les risques du cloud computing dans plusieurs catégories [20] :

- Les risques liés aux données, tels que l'accès aux données chez le fournisseur cloud, la segmentation interne des données chez celui-ci, la propriété des données et le chiffrement ;
- Les risques liés à la disponibilité, tels que la dégradation des services ou les pannes ;
- Les risques liés au provisionnement des services, tels que les changements dans les services ou leur coût ;

- Les risques liés aux activités malveillantes: l'infrastructure d'un fournisseur cloud public doit être accessible publiquement et peut devenir la cible d'attaques.

3.5.1 Risques liés à la sécurité

La sécurité est souvent primordiale lors des évaluations de risques associés aux services cloud. Le centre de cybersécurité de l'Australie a publié en 2019 un document contenant des considérations sur la sécurité du cloud. Ce document mentionne notamment les risques liés à la continuité des opérations, l'accès non autorisé aux données par une tierce partie, un autre client du fournisseur cloud ou même un employé malveillant du fournisseur [21].

Le sondage de l'ENISA sur la perspective de 70 PME européennes montre également que la confidentialité des données d'entreprise, la disponibilité et l'intégrité des services font partie des craintes principales vis-à-vis d'une potentielle adoption de stratégie cloud [15].

3.5.2 Réduction de la gouvernance opérationnelle

Un risque souvent mentionné par rapport à l'adoption de services cloud est la perte potentielle de gouvernance opérationnelle sur les données. L'externalisation des ressources informatiques vers un fournisseur cloud réduit inévitablement le contrôle de l'entreprise sur le traitement et la sécurité des ses données.

Selon un rapport sur la sécurité du cloud computing publié par l'ENISA en 2012, ceci inclut aussi les risques liés à la conformité aux réglementations sur les données [22]. Une certification ou une exigence de conformité envers un client cloud peut dépendre de la capacité de son fournisseur à prouver sa propre conformité.

3.5.3 Portabilité limitée entre les fournisseurs de services cloud

Un risque qui doit également être pris en compte est la possibilité de l'enfermement propriétaire. L'enfermement propriétaire désigne une situation où un composant logiciel ou matériel fourni par un fournisseur n'est pas interopérable avec un composant équivalent proposé par un autre fournisseur [23].

En effet, les interfaces de programmation permettant d'utiliser les services cloud d'un fournisseur sont rarement compatibles avec celles d'un autre fournisseur. Cette portabilité limitée entre les infrastructures de fournisseurs de services amène un risque pour l'entreprise de devenir fortement dépendant d'un seul fournisseur une fois que le système informatique de l'entreprise dépend de celui du fournisseur.

4. Les principaux fournisseurs de cloud computing public

Gartner, société de conseil et de recherche, a publié en septembre 2020 une analyse du marché des plateformes de cloud computing.

D'après cette analyse, le marché du cloud computing est dominé par quatre fournisseurs : «*Alibaba Cloud en Chine et dans les pays asiatiques avoisinants, et AWS, Azure et GCP dans la plupart des autres régions du monde*» (traduction libre) [24].

Figure 3 : Le quadrant des fournisseurs cloud selon Gartner



Source : [24]

Le rapport de Gartner place Amazon Web Services (AWS), Microsoft Azure et Google Cloud Platform (GCP) dans la catégorie des leaders du marché [24]. Ce chapitre présente ces trois fournisseurs.

4.1 Amazon Web Services

Amazon Web Services (AWS) a été la première plateforme de cloud computing en 2006 avec l'annonce du lancement des services S3 et EC2.

«Amazon Elastic Compute Cloud (Amazon EC2) est un service Web qui fournit une capacité de calcul redimensionnable dans le cloud. Il est conçu pour faciliter l'informatique à l'échelle du Web pour les développeurs. Tout comme Amazon Simple Storage Service (Amazon S3) permet le stockage dans le cloud, Amazon EC2 permet le « calcul » dans le cloud. L'interface web simple d'Amazon EC2 vous permet d'obtenir et de configurer la capacité de calcul avec un minimum de friction.» (traduction libre) [25]

AWS propose aujourd'hui des centres de données 25 régions géographiques qui couvrent 80 zones de disponibilité [26] (cf. annexe 5). Un ou plusieurs centres de données sont compris dans chaque zone de disponibilité, et chaque zone d'une région est connectée aux autres par un réseau à fibre optique à haute bande passante, privé et chiffré [27].

Une région peut être choisie lors du provisionnement d'un service selon les besoins du client, ce qui représente un avantage intéressant lorsque celui-ci doit prendre en compte des problématiques juridiques liées aux régulations des données de certains pays.

Sur la base de cette infrastructure, AWS propose à ses clients un catalogue de plus de 200 services cloud [28].

Ci-suit une présentation des principaux services proposés par AWS :

- EC2 (Elastic Compute Cloud) : un service d'exécution de machines virtuelles permettant à chacun de louer des serveurs afin d'exécuter ses propres applications. AWS propose un large choix de systèmes d'exploitation basés sur des images (Amazon Machine Images) pour les machines virtuelles EC2 ;
- S3 (Simple Storage Service) : un service de stockage de données évolutif, sécurisé, hautement performant et disponible. Les données sont stockées sous la forme d'objets pouvant aller jusqu'à une taille de 5 téraoctets dans des espaces nommés "compartiments" ;
- Route 53 : un service de système de nom de domaines (DNS) dans le cloud AWS, permettant le routage des requêtes d'utilisateur vers des applications ou vers d'autres éléments de l'infrastructure ;
- DynamoDB : un service de base de données de type NoSQL basé sur la performance et la scalabilité ;

- RDS (Relational Database Service) : un service de base de données relationnelle disponible avec les moteurs de base de données PostgreSQL, MariaDB, Amazon Aurora, MySQL, Oracle et Microsoft SQL Server.

Ces services ne représentent qu'une petite partie du catalogue de services qui couvre un vaste éventail d'activités telles que l'intelligence artificielle, l'Internet des Objets, des services financiers ou même le développement de jeux vidéo (cf. annexe 3).

4.2 Microsoft Azure

Microsoft propose depuis 2010 des services de cloud computing via leur plateforme Azure [29]. La plateforme cloud de Microsoft propose des services de type SaaS, IaaS et PaaS compatibles avec beaucoup de technologies Microsoft préexistantes et oriente son offre vers les clients de type entreprise avec le mode de déploiement hybride.

Les services Azure sont basés sur une infrastructure globale de plus de 200 centres de données dispersés dans le monde à travers de nombreuses régions et zones de disponibilité [30] (cf. annexe 6). Les centres de données sont reliés par un réseau Azure privé, ainsi le trafic IP ne passe jamais par le réseau public [30].

Parmi les services proposés par Azure, on peut compter :

- Virtual Machines : un service d'exécution de machines virtuelles comparable à Amazon EC2 permettant la provision d'instances de machines virtuelles Linux ou Windows Server ;
- Blob Storage : un service de stockage de données comparable à AWS S3 pour le stockage de tout type de données allant jusqu'à 200 téraoctets ;
- Cosmos DB : un service de base de données NoSQL comparable à Amazon DynamoDB ;
- Différents services de base de données relationnelle de type MySQL, MariaDB et PostgreSQL notamment ;
- Azure Active Directory : un service de gestion d'identité et d'accès pour les entreprises.

De nombreux autres services sont disponibles, par exemple des services de type sécurité, de bureaux virtuels, outils de développement logiciel ou même blockchain.

Tout comme AWS, Azure propose aux utilisateurs d'administrer leurs ressources cloud via une interface web ou un outil en ligne de commande (Azure CLI).

4.3 Google Cloud Platform

Google propose depuis 2008 propose la plateforme Google Cloud Platform (GCP) avec le lancement de son service Google App Engine, un service d'exécution d'application de type PaaS [31].

Depuis, son offre a évolué pour aujourd'hui proposer une large gamme de services aux entreprises. Les services proposés par GCP incluent notamment des solutions spécialisées liées à l'intelligence artificielle, la sécurité, le réseau, l'Internet des Objets et l'analyse de données (cf. annexe 4).

La plateforme GCP est basée sur une infrastructure mondiale comprenant des centres de données dans 25 régions et 76 zones de disponibilité accessibles depuis plus de 200 pays et territoires [32] (cf. annexe 7).

Voici quelques services disponibles sur la plateforme GCP :

- Compute Engine : un service d'exécution de machines virtuelles comparable à AWS EC2 et Azure Virtual Machines ;
- Cloud Storage : un service de stockage de données ;
- Google Kubernetes Engine : un service d'exécution de conteneurs d'applications ;
- Cloud CDN : un service CDN² permettant la diffusion optimisée de contenu sur le web.

Les interfaces permettant d'administrer les services GCP sont l'interface en ligne de commande gcloud et l'interface web Google Cloud Console.

5. Développement d'un prototype Serverless avec AWS

Le but de ce développement est d'approfondir ma compréhension des technologies cloud en développant un prototype d'application web Serverless tel que défini dans la première partie de ce mémoire.

En tant que développeur, le cloud computing me semble être un outil intéressant pour développer un prototype ou même une application complète en faisant levier de services

² Content Delivery Network, un réseau de diffusion de contenu [33].

préexistants et souvent communs à de nombreuses applications web, telles que des fonctionnalités de stockage ou d'authentification.

Cette section du mémoire présente la mise en œuvre d'un prototype d'application web de messagerie instantanée basé sur les services d'AWS.

Alors que les offres de Amazon, Microsoft et Google sont, à mon avis, toutes viables pour la réalisation de ce projet, j'ai choisi la plateforme AWS pour sa maturité et sa popularité, AWS étant la première plateforme arrivée sur le marché.

Le code source produit dans le cadre du développement de ce prototype est disponible publiquement (cf. annexe 1).

5.1 Mise en place du projet

5.1.1 Accès à la console de gestion

Comme cité précédemment, la console de gestion est l'interface web unifiée permettant entre autres de gérer l'utilisation des services, la tarification, le contrôle des utilisateurs, des permissions et des applications. C'est depuis cette interface web que l'on construit et configure des applications basées sur la plateforme AWS.

Afin d'avoir accès à la console de gestion et aux services de la plateforme, il faut d'abord créer un compte AWS. Pour la réalisation de ce prototype, un compte AWS gratuit a été suffisant. Ce compte est appelé l'utilisateur racine du compte, car il désigne l'identité unique de connexion à la plateforme qui a accès à tous les services et ressources de ce compte.

AWS propose également une interface en ligne de commande, AWS CLI, permettant de gérer les services depuis un terminal et d'automatiser des commandes de gestion de service.

5.1.2 Création d'un compte administrateur avec le service IAM

Après la création du compte racine, le premier service AWS important à comprendre est IAM (Identity and Access Management).

Ce service de gestion d'accès permet de créer des utilisateurs, des groupes d'utilisateurs et d'assigner des permissions d'accès aux utilisateurs ou même aux différents services et ressources disponibles sur la plateforme AWS [34].

Figure 4 : Création d'un utilisateur IAM

Définissez les informations utilisateur

Vous pouvez ajouter plusieurs utilisateurs en même temps avec les mêmes types et autorisations d'accès. [En savoir plus](#)

Nom d'utilisateur*

[+ Ajoutez un autre utilisateur](#)

Sélectionnez un type d'accès AWS

Sélectionnez comment ces utilisateurs accèderont à AWS. Les clés d'accès et les mots de passe générés automatiquement sont fournis à la dernière étape. [En savoir plus](#)

- Type d'accès* ☐ **Accès par programmation**
Active une **ID de clé d'accès** et **clé d'accès secrète** pour AWS API, CLI, SDK et d'autres outils de développement.
- ☐ **Accès à AWS Management Console**
Active un **mot de passe** qui permet aux utilisateurs de vous connecter à l'AWS Management Console.

Le compte racine cité plus haut n'est utilisé que pour créer ce premier utilisateur IAM qui sera l'administrateur de l'application développée.

Les permissions « `AWSCodeCommitFullAccess` » et « `AdministratorAccess` ». Ces permissions permettent respectivement d'avoir accès au service CodeCommit qui sera décrit plus bas ; ainsi qu'à tous les services et ressources AWS, mais pas le tableau de bord de facturation du compte racine.

5.1.3 Le Serverless selon AWS

Le prototype – une application web de messagerie instantanée – est développée selon une architecture Serverless telle que définie au début de ce mémoire. Dans le contexte d'AWS, le terme Serverless désigne un catalogue de services particulièrement adapté pour ce type d'application.

Ces services sont rendus accessibles aux développeurs par des interfaces de programmation (API) ou des kits de développement logiciel (SDK).

AWS structure ses services Serverless en trois catégories [35] :

- La couche de calcul ;
- La couche intégration d'applications ;
- La couche stockage de données.

La couche de calcul désigne les services permettant d'exécuter du code ou des containers sur l'infrastructure AWS, afin d'exécuter la logique métier de l'application côté serveur.

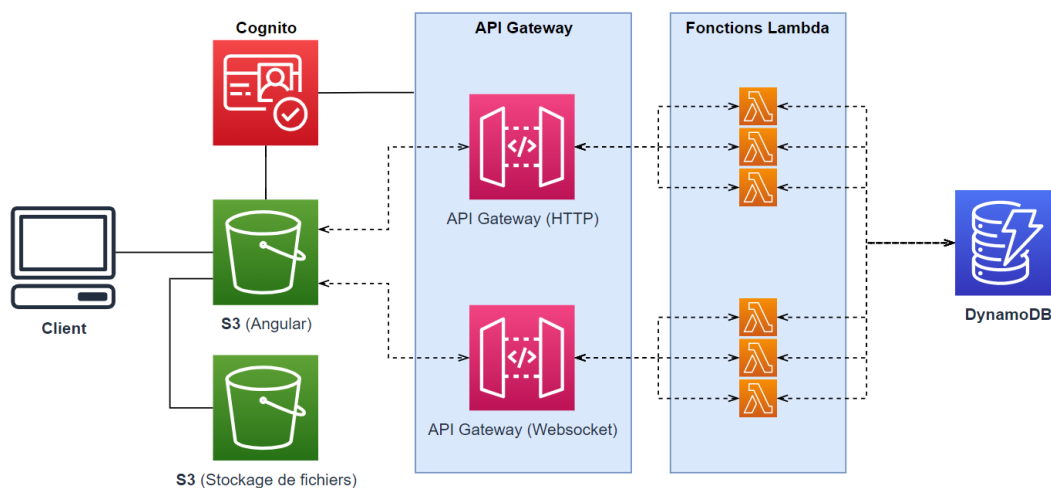
La couche intégration d'applications regroupe les services AWS liés à la communication entre les applications. Cette couche inclut le service de création d'API web API Gateway, mais aussi des services basés sur les événements ou l'envoi automatique de SMS et emails.

La couche stockage de données inclut des services tels que Amazon S3 et Amazon DynamoDB susmentionnées dans la liste des services proposés par AWS.

5.1.4 Architecture du prototype

Voici une figure représentant une vue d'ensemble de l'architecture du prototype avec les services AWS sélectionnés.

Figure 5 : Architecture du prototype



Comme on peut le voir, les services sélectionnés sont S3, Cognito, API Gateway, Lambda et DynamoDB. Les justifications des choix de chaque service pour ce prototype sont présentées dans les parties suivantes de la description de la mise en œuvre.

Le logiciel client – une application web Angular – est l'interface visible pour les utilisateurs. Cette interface web communique avec les services AWS sélectionnés afin d'implémenter les mécanismes nécessaires au bon fonctionnement de l'application.

La partie suivante explique comment l'application est intégrée et déployée de manière continue sur les serveurs d'AWS à l'aide des services CodeCommit, CodeBuild et CodePipeline.

5.2 Intégration continue du client

Cette section décrit la configuration de trois services AWS liés à l'intégration continue des logiciels : CodeCommit, CodeBuild et CodePipeline.

5.2.1 Gestion de versions

Le code source de l'application Angular est versionné dans un dépôt de code à l'aide du service CodeCommit, un AWS service de gestion de version de code source proposé par AWS basé sur le système Git [36]. Au fil du développement de l'application, le code Angular est enregistré dans ce dépôt.

Pour obtenir les permissions d'écriture, une clé SSH³ créée sur mon poste de travail est associée au compte IAM administrateur ayant la permission `AWSCodeCommitFullAccess`.

5.2.2 Compilation

A chaque changement du code sur la branche principale du dépôt, le code de l'application Angular doit être compilé afin de construire les fichiers finaux de l'application selon le fonctionnement normal de la commande « `ng build` ».

Pour cette étape, le service CodeBuild est activé. Ce service d'intégration continue permet d'exécuter automatiquement diverses commandes de compilation ou de préparation de code source nécessaire pour certains logiciels avant leur déploiement [37].

La configuration de cette phase de construction se fait par un fichier nommé `buildspec.yml` qui doit se situer à la racine du code source du projet.

³ Secure Shell (SSH), un protocole de communication sécurisé basé l'échange de clés de chiffrement en début de connexion [38].

Figure 6 : Contenu du fichier buildspec.yml

```
version: 0.1

phases:
  pre_build:
    commands:
      - npm i -g @angular/cli
      - npm install
  build:
    commands:
      - ng build --prod
artifacts:
  files:
    - "**/*"
  discard-paths: no
  base-directory: dist/serverless-app
```

On peut voir que ce fichier définit les commandes à exécuter lors de la construction de l'application Angular, notamment l'installation des dépendances du projet et la construction des fichiers finaux avec « ng build ».

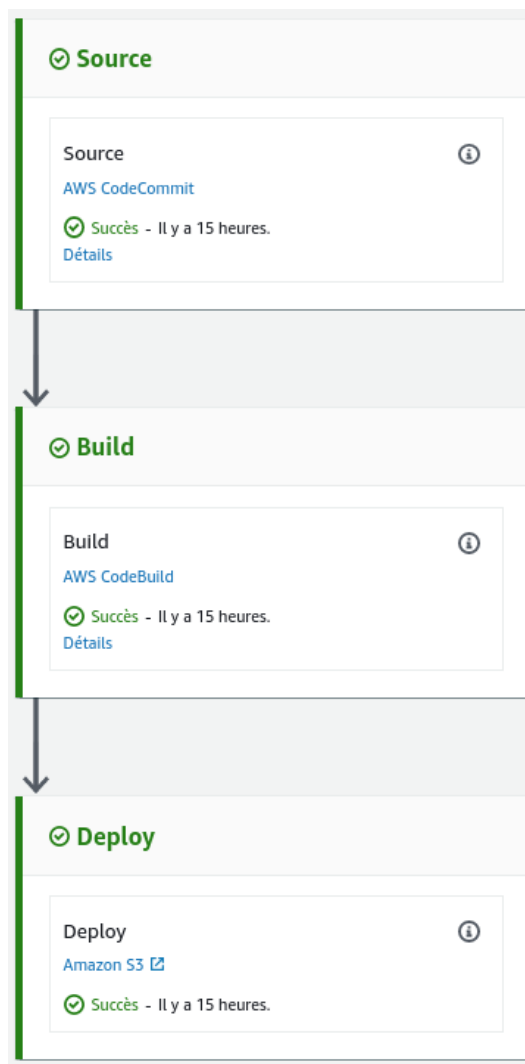
Cette phase de construction est activée automatiquement lors de chaque changement de code source détecté par le service CodeCommit.

5.2.3 Déploiement

La phase finale de cette procédure d'hébergement de l'application est la phase de déploiement. Le résultat de la compilation de code source doit maintenant être déployé sur un serveur AWS.

Pour cela, le service CodePipeline, un service permettant la configuration de pipelines d'intégration de code [39], est utilisé. Il permet de choisir une source de code, d'enchaîner des phases de construction sur le code source, et de choisir une destination de déploiement pour le résultat de cette construction.

Figure 7 : Vue d'ensemble du pipeline d'intégration continue



Pour ce projet, la source est le dépôt CodeCommit qui enregistre le code source Angular et la destination est un compartiment S3 créé spécialement pour héberger l'application Angular.

Un compartiment S3 est un espace de stockage ayant un nom unique et existant dans une région AWS [40]. Le compartiment créé dans cet exemple est nommé « compartiment-hebergement-s3 » et assigné à la région « eu-central-1 ».

Figure 8 : Création d'un compartiment S3

Configuration générale

Nom du compartiment

compartiment-hebergement-s3

Le nom du compartiment doit être unique et ne peut pas contenir d'espaces ou de majuscules. [Consulter les recommandations relatives à l'attribution de noms de compartiment](#)

Région AWS


UE (Francfort) eu-central-1

Copier les paramètres depuis un compartiment existant - *facultatif*
Seuls les paramètres de compartiment dans la configuration suivante sont copiés.

Sélectionner un compartiment

Une fois le compartiment créé, la console nous affiche les détails de celui-ci comme dans la figure ci-dessous.

Figure 9 : Détails du compartiment

Région AWS	Amazon Resource Name (ARN)	Date de création
UE (Francfort) eu-central-1	 arn:aws:s3::compartiment-hebergement-s3	30 Jun 2021 11:59:40 AM CEST

On remarque qu'un identifiant ARN (Amazon Resource Name) a été attribué au compartiment S3. Un ARN est un identifiant unique permettant d'identifier de manière non ambiguë une ressource (par exemple un utilisateur, une permission, ou une machine virtuelle) parmi toutes celles créées sur la plateforme AWS [41].

Par défaut, à un compartiment S3 est privé et nécessite des permissions spéciales afin d'accéder aux données sauvegardées. Cependant, AWS propose une option de configuration de compartiment S3 permettant l'accès public aux données par une adresse URL. Ce cas de figure fait de S3 une bonne option d'hébergement de sites webs statiques ou d'interfaces web tels que le client Angular de ce prototype.

Figure 10 : Configuration du compartiment pour l'hébergement d'un site

Hébergement de site Web statique

Utilisez ce compartiment pour héberger un site Web ou rediriger des demandes. [En savoir plus](#)

Hébergement de site Web statique

☐ Désactiver

☒ Activer

Type d'hébergement

☒ Héberger un site Web statique

Utilisez le point de terminaison du compartiment comme adresse Web. [En savoir plus](#)

☐ Rediriger des demandes pour un objet

Redirigez les demandes vers un autre compartiment ou domaine. [En savoir plus](#)

Une fois la configuration la pipeline terminée, l'application Angular est automatiquement construite et déployée vers un serveur accessible publiquement lors de chaque changement du code source.

5.3 Base de données

Pour la couche stockage de données du prototype, le service DynamoDB a été sélectionné car c'est un service de base de données NoSQL entièrement géré [42]. Cela veut dire que les aspects liés à la configuration des serveurs et du réseau sous-jacent au service sont entièrement pris en charge par la plateforme AWS.

La flexibilité du modèle NoSQL est un avantage à mon avis important lors du développement d'un premier prototype sur AWS. En tant que développeur, j'ai une préférence pour la liberté du modèle NoSQL, comparé aux contraintes d'un schéma relationnel.

5.3.1 Services AWS de base de données

De nombreux services de bases de données sont proposés par AWS. Les paragraphes suivants décrivent brièvement quelques alternatives à DynamoDB.

5.3.1.1 Relational Database Service (RDS)


RDS est un service de base de données relationnelle. Ce service permet de rapidement installer, gérer et mettre à l'échelle une instance de base de données selon le moteur de son choix [43].


Divers paramètres liés à l'espace de stockage, la disponibilité (instance de secours dans une autre zone de disponibilité) et l'authentification peuvent être sélectionnés.


Figure 11 : Moteurs de base de données disponibles avec le service RDS


Options de moteur


Type de moteur [Infos](#)


☐ Amazon Aurora


☐ MySQL


☐ MariaDB


☒ PostgreSQL


☐ Oracle


☐ Microsoft SQL Server


Version

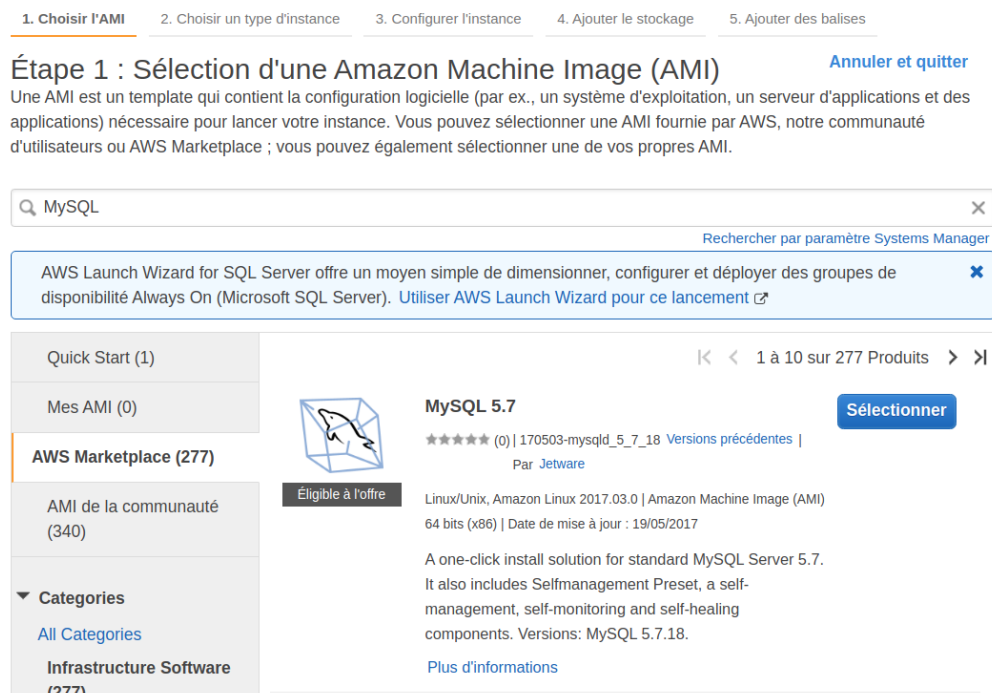
PostgreSQL 12.5-R1 ▼

5.3.1.2 Base de données EC2

Le système d'exécution de machines virtuelles EC2 mentionné plus haut dans ce mémoire peut également être utilisé pour lancer une machine virtuelle spécialement configurée pour accueillir un moteur de base de données.

Lors de la création d'une instance EC2, une image peut être sélectionnée parmi un vaste catalogue d'images correspondant à des pré-configurations de machine virtuelles [44]. Ainsi, une option pour le développement de ce prototype serait de déployer une instance EC2 basée sur une image de base de données.

Figure 12 : Images disponibles lors de la création d'une instance EC2



5.3.1.3 Elasticache

ElastiCache est un service permettant la création de magasins de données à haut débit compatibles avec Redis et Memcached, deux moteurs de bases de données en mémoire vive [45].

5.3.1.4 Neptune

Neptune permet la création de bases de données orientées graphes, un schéma de base de données permet des hautes performances lorsque les données stockées sont fortement basées sur les relations entre les points de données [46].

5.3.2 Création de tables DynamoDB

Pour ce projet, plusieurs tables ont été créées dans la base de données DynamoDB. Le processus de création de table permet de choisir un nom et une clé primaire.

La clé primaire est une valeur qui identifie de manière unique un élément dans la table. Celle-ci est composée d'une clé de partition et d'une clé de tri facultative [47].

La clé de partition est utilisée en interne par le service DynamoDB pour déterminer sur quel hôte physique la donnée est stockée. Ainsi, tous les éléments ayant la même clé de partition sont stockés ensembles, et triés selon la clé de tri facultative [47].

Figure 13 : Exemple de création de table DynamoDB

Créer une table DynamoDB
Didacticiel ?

DynamoDB est une base de données sans schéma qui requiert uniquement un nom de table et une clé primaire. La clé primaire de la table est composée d'un ou deux attributs qui identifient de façon unique les éléments, partitionnent les données et les trient au sein de chaque partition.

Nom de la table* MesObjets ⓘ

Clé primaire* Clé de partition

ObjetId ⓘ
Chaîne ▾
☒ Ajouter une clé de tri
ObjetTimestamp ⓘ
Chaîne ▾

La figure ci-dessus montre un exemple de création de table nommée « MesObjets » avec une clé primaire composée d'une clé de partition « ObjetId » et une clé de tri « ObjetTimestamp », correspondant à l'horodatage de l'enregistrement de l'objet.

Il est aussi possible de créer des index secondaires qui permettent d'exécuter des requêtes sur la table en utilisant des attributs qui ne font pas partie de la clé primaire, permettant une plus grande flexibilité des requêtes [47].

Figure 14 : Options d'allocation de capacité d'écriture et de lecture

Capacités allouées

	Unités de capacité de lecture	Unités de capacité d'écriture
Table	5	5

Coût estimé USD3,54 / mois ([Calculateur de capacité](#))

Auto Scaling

	Capacité de lecture	Capacité d'écriture
	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
		<input type="checkbox"/> Mêmes paramètres que pour la lecture
Utilisation de la cible	70 %	70 %
Capacité allouée minimum	5 unités	5 unités
Capacité allouée maximum	40000 unités	40000 unités
	<input checked="" type="checkbox"/> Appliquer les mêmes paramètres aux index secondaires globaux	<input checked="" type="checkbox"/> Appliquer les mêmes paramètres aux index secondaires globaux

La console propose ensuite de configurer les capacités de la table. Par défaut, l'option Auto Scaling est activée. Cette option permet d'augmenter ou de réduire

automatiquement les capacités d'écriture et de lecture de la table afin de pouvoir répondre aux variations du niveau d'utilisation.

Selon la documentation fournie par AWS, la capacité d'écriture et la capacité de lecture d'une table sont mesurées respectivement par une unité de capacité de lecture et une unité de capacité d'écriture [48] :

- Une unité de capacité de lecture correspond à une lecture à cohérence forte par seconde ou deux lectures à cohérence faible, jusqu'à une taille de 4 kilobits.
- Une unité de capacité d'écriture correspond à une écriture par seconde, jusqu'à une taille de 1 kilobits.

La cohérence forte ou faible est un concept lié aux systèmes informatiques distribués où les données sont stockées de manière redondante sur plusieurs machines [49] :

- Une cohérence forte garantit que la lecture d'une copie de données prendra en compte toute modification qui a précédé la lecture de cette donnée ;
- Une cohérence faible ne garantit pas que la donnée en lecture n'a pas été modifiée récemment sur une autre machine. Ainsi la donnée lue pourrait ne pas refléter tous les changements récents de manière immédiate.

Par prudence, les fonctionnalités d'Auto Scaling sont désactivées pour toutes les tables de ce projet afin d'éviter de surutiliser involontairement les capacités de lecture ou d'écriture.

Une telle surutilisation pourrait provoquer un dépassement des ressources allouées aux comptes AWS gratuits et produire une facture inattendue. Les tables sont donc paramétrées pour utiliser une seule capacité d'écriture et de lecture.

5.4 Tables DynamoDB du projet

Les tables DynamoDB créées pour ce projet ont pour but de permettre le stockage de données côté serveur pour les fonctionnalités suivantes :

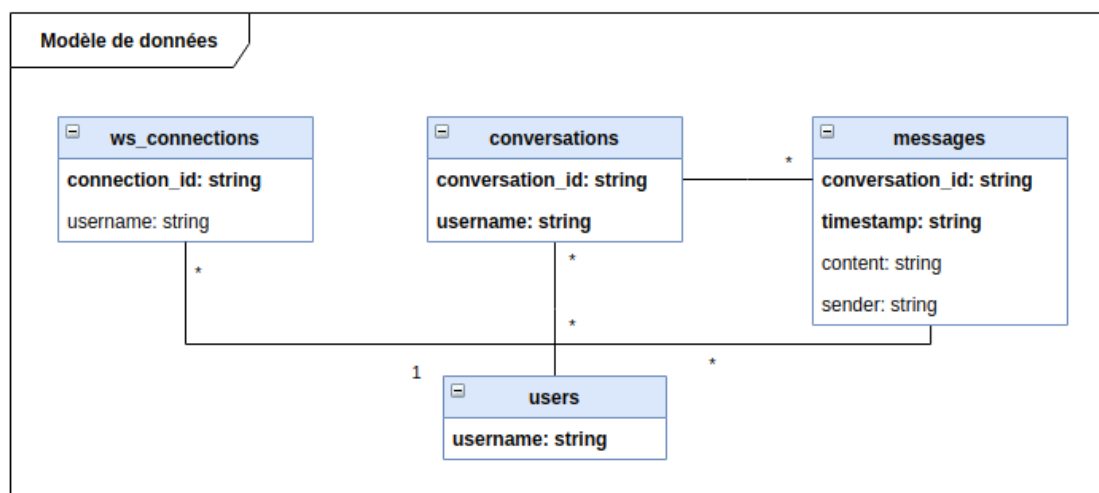
- L'enregistrement et la récupération des conversations entre les utilisateurs ;
- L'enregistrement et la récupération des messages dans ces conversations ;
- La mise à jour d'informations liées aux utilisateurs (par ex. : photo de profil et préférences) ;
- L'enregistrement des identifiants de connexions Websocket actives et leur mise à jour pour leur associer un nom d'utilisateur.

Tableau 1 : Tables DynamoDB pour une application de messagerie

Nom	Clé de partition	Clé de tri
conversations	conversation_id	username
messages	conversation_id	timestamp
users	username	
ws_connections	connection_id	

Toutes les clés de partition et de tri sont de type chaîne de caractère. Ceci est un choix arbitraire, car tous les identifiants autres que username auraient pu être des nombres.

Figure 15 : Modèle de données



Ci-dessus figure le modèle de données tel qu'il est implémenté par les tables DynamoDB décrites plus haut. On peut remarquer plusieurs choses :

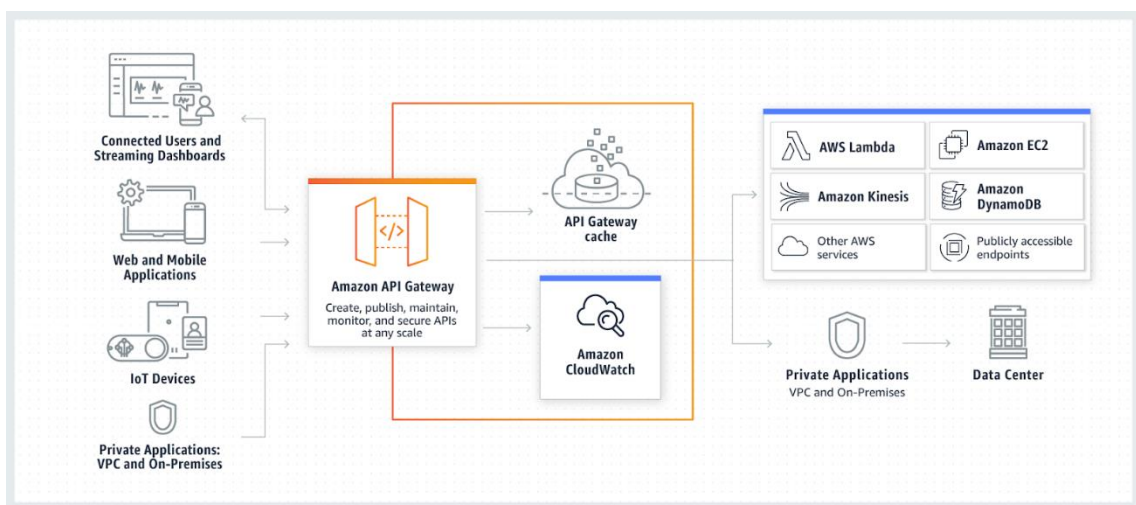
- Le stockage des conversations en tant qu'association entre un identifiant de conversation et un utilisateur fournit une flexibilité qui permettrait d'implémenter des fonctionnalités de conversations de groupe (plus de deux utilisateurs), même si le prototype ne permet que la conversation entre deux utilisateurs ;
- Les utilisateurs sont identifiés par le nom d'utilisateur Cognito. La politique d'identification définie avec le service d'authentification Cognito oblige le nom d'utilisateur à être unique et interdit son changement ;
- Les identifiants de connexions Websocket sont associés aux noms d'utilisateurs dans la table `ws_connections`.

5.5 Passerelles API

Cette partie concerne la création de la couche d'intégration d'application. Afin de permettre l'interaction entre le logiciel client et les autres couches applicatives – les couches de calcul et de stockage de données – l'application a besoin d'un point d'entrée vers lequel envoyer ses requêtes.

Le service API Gateway est utilisé pour permettre cette interaction entre le client et les services AWS. Le choix d'utiliser ce service est évident : c'est le service privilégié pour la création de passerelles API sur la plateforme AWS [50].

Figure 16 : Architecture API Gateway



Source : [50]

Comme la figure ci-dessus le montre, le rôle du service API Gateway est de recevoir les requêtes d'un logiciel client et de les router vers un autre service AWS, et de retourner la réponse du service au client une fois le traitement de la requête terminé.

Les passerelles API Gateway peuvent être créées selon diverses variantes [50] :

- API REST : la première variante proposée par AWS basée sur les principes REST ;
- API REST privée : une API REST déployable sur un réseau privé ;
- API Websocket : une API basée sur la technologie Websocket ;
- API HTTP : la dernière génération d'API basée sur le protocole HTTP mise à disposition par AWS.

Le prototype développé étant une application de messagerie instantanée, une API de type Websocket est créée afin de router les messages en temps réel.

Une deuxième API de type HTTP est créée afin de router toutes les autres requêtes de l'application, telles que la récupération des utilisateurs, de leurs conversations et leurs préférences. La variante HTTP est choisie pour ses promesses de performance et de réduction des coûts par rapport à l'alternative API REST.

5.5.1 API Websocket

Pour les API Websocket proposées par API Gateway, le routage des requêtes se fait par une clé de routage : un mot clé contenu dans le message Websocket reçu par API Gateway correspondant à une action [51]. Cette action détermine quelle route de l'API Websocket traite le message reçu.

Par défaut, trois routes sont nécessaires pour la configuration d'une API Websocket : \$connect, \$disconnect et \$default.

En plus de ces trois routes par défaut, deux routes additionnelles ont été configurées pour traiter les requêtes de l'application de messagerie. Le nom de ces routes personnalisées correspond à leur clé de routage.

Pour ce prototype, toutes les requêtes à l'API Websocket sont traitées par des fonctions sans serveur – des modules de code stockés et exécutés par les serveurs d'AWS à la demande – en utilisant le service AWS Lambda. Voici une description de la fonction de chaque route :

Tableau 2 : Routes de l'API Websocket

Route	Fonctionnalité	Fonction
\$connect	Enregistrer un identifiant de connexion unique à l'interface Websocket.	ServerlessOnConnect
\$disconnect	Supprimer l'identifiant de connexion unique à l'interface Websocket.	ServerlessOnDisconnect
\$default	Traiter les requêtes contenant une clé de routage qui ne correspond à aucune route, par exemple pour retourner un message d'erreur au client.	ServerlessOnDefault

sendMessage	Traiter les requêtes qui contiennent un message envoyé par un utilisateur à une autre.	ServerlessOnSendMessage
sendUsername	Traiter les requêtes qui envoient le nom de l'utilisateur immédiatement après sa connexion à l'API Websocket afin d'associer le nom d'utilisateur à la connexion Websocket.	ServerlessOnSendUsername

5.5.2 API HTTP

Pour toutes les autres requêtes du prototype qui ne sont pas liées aux connexions Websocket et nécessitant un traitement en temps réel, une API HTTP a été créée.

Les API HTTP, elles, ne sont pas basées sur des clés de routage mais sur des ressources et des méthodes HTTP [52]. Les routes de l'API sont définies par la combinaison de ces ressources et méthodes HTTP.

Tableau 3 : Routes de l'API HTTP

Route	Fonction	Fonction
Méthode POST Ressource /conversations	ServerlessConversationsCreate	Créer une nouvelle conversation entre deux utilisateurs.
Méthode GET Ressource conversations/{id}	ServerlessConversationsGet	Récupérer les messages d'une conversation spécifiée par son identifiant unique.
Méthode GET Ressource /users	ServerlessUsersGet	Récupérer les utilisateurs enregistrés afin de permettre la recherche d'utilisateurs pour créer une nouvelle conversation.
Méthode	ServerlessUserConversationsGet	Récupérer les conversations d'un

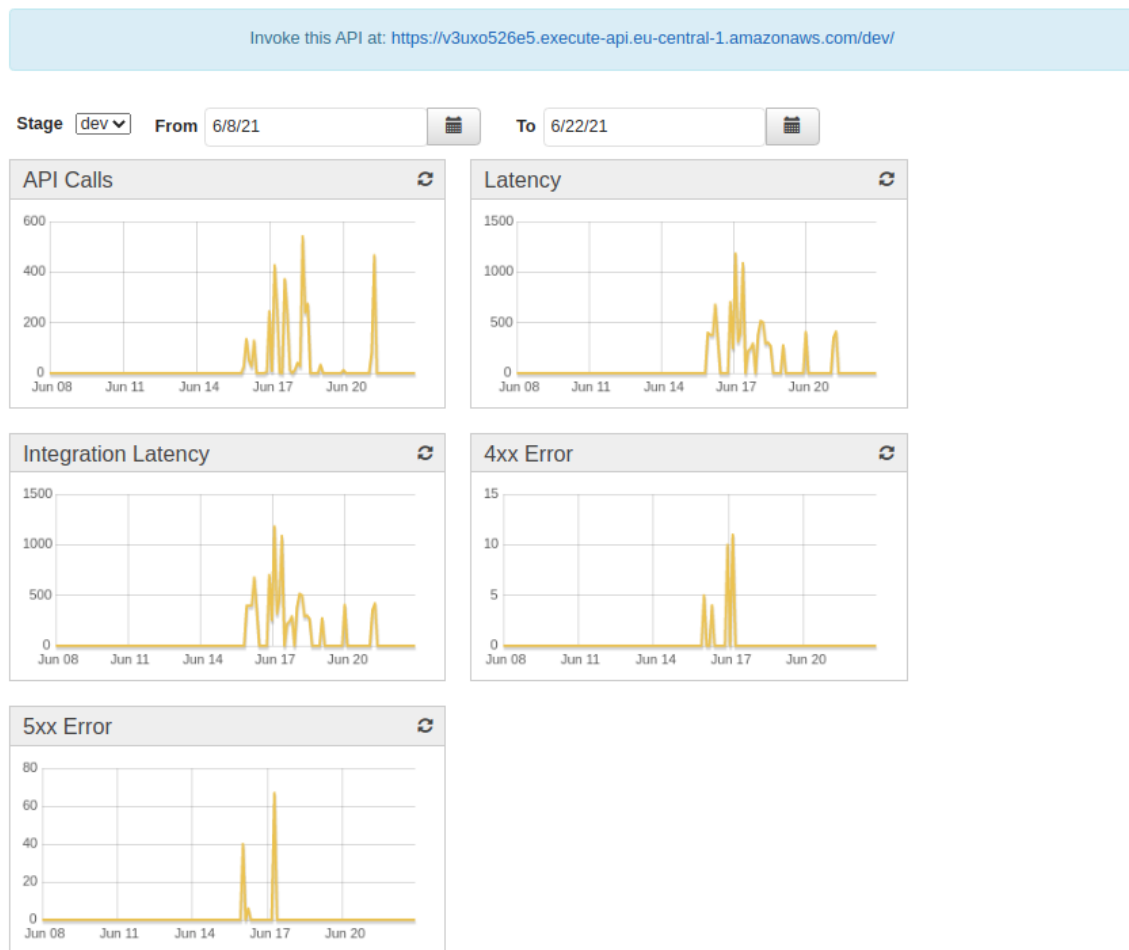
Méthode GET Ressource users/{username}/conversations		utilisateur identifié par son nom d'utilisateur.
Méthode GET Ressource users/{username}/preferences	ServerlessUserPreferencesGet	Récupérer les préférences d'un utilisateur identifié par son nom d'utilisateur.
Méthode PATCH Ressource users/{username}/preferences	ServerlessUserPreferencesUpdate	Mettre à jour les préférences d'un utilisateur identifié par son nom d'utilisateur.

5.5.3 Déploiement de l'API et intégration à l'application cliente

Une fois l'API configurée, il suffit de la déployer dans vers une étape. Une étape est une version de l'API qui correspond à une étape de développement, comme « développement » ou « production » . Le déploiement vers un étape fournit une URL d'invocation de l'API la rend publique et accessible par le logiciel client.

Les API configurés avec API Gateway peuvent être déployées vers ces différentes étapes afin de par exemple tester l'API dans un environnement de test ou de développement avant d'activer l'API pour un système en production.

Figure 17 : URL d'invocation et indicateurs d'activités de l'API déployée



Du côté de l'application cliente Angular, des classes de services sont codées pour interagir avec l'API HTTP au nom de l'application : UserService et ConversationService

Figure 18 : Code exécutant une requête vers l'URL d'invocation de l'API

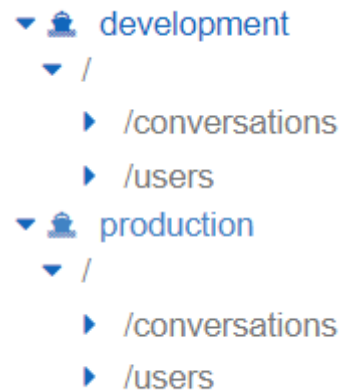
```
/**
 * URL d'invocation de la ressource "conversation"
 */
private conversationsUrl = 'https://v3uxo526e5.execute-api.eu-central-1.amazonaws.com/dev/conversations/'

/**
 * Créer une conversation entre deux utilisateurs
 * @param user1 - Nom d'utilisateur de l'utilisateur 1
 * @param user2 - Nom d'utilisateur de l'utilisateur 2
 */
createConversation(user1: string, user2: string): Observable<any> {
    return this.httpClient.post<any>(this.conversationsUrl, { user1, user2 })
}
```

La figure ci-dessus montre un extrait de code de la classe ConversationService permettant d'envoyer une requête de création de conversation à l'URL d'invocation de la ressource Conversation.

5.5.4 Gestion des versions des API

Le service API Gateway permet de créer plusieurs versions d'une API par la création d'étapes. La figure suivante montre deux étapes d'exemple : une étape de développement et une étape de production.



Cette fonctionnalité d'API Gateway permet de sauvegarder la configuration de l'API sous différentes versions et chacune de ces versions dispose d'URL d'invocation dédiée à cette version.

5.6 Fonctions sans serveur

Pour la couche de calcul de l'application de messagerie sans serveur, c'est le service Lambda qui est utilisé.

Ce service permet de télécharger des modules de code vers les serveurs d'AWS et de configurer leur déclenchement automatique depuis de nombreux services AWS [53]. Ces modules de code peuvent être écrits dans différents langages (dont Node.js, Python, Go et Java).

5.6.1 Fargate, alternative à Lambda

La catégorie de services Serverless de AWS propose un autre moteur de calcul sans serveur : AWS Fargate. Fargate est un service basé sur Elastic Container Service (ECS), un moteur d'exécution de conteneurs [54]. Ce service n'est donc pas basé sur des modules de code source mais sur des images de conteneurs à télécharger vers la plateforme AWS.

Lambda et Fargate sont deux options tout à fait viables pour la couche de ce projet de prototype. Les deux services permettent de se concentrer sur le développement d'applications sans devoir se soucier de provisionner, installer, configurer ou administrer

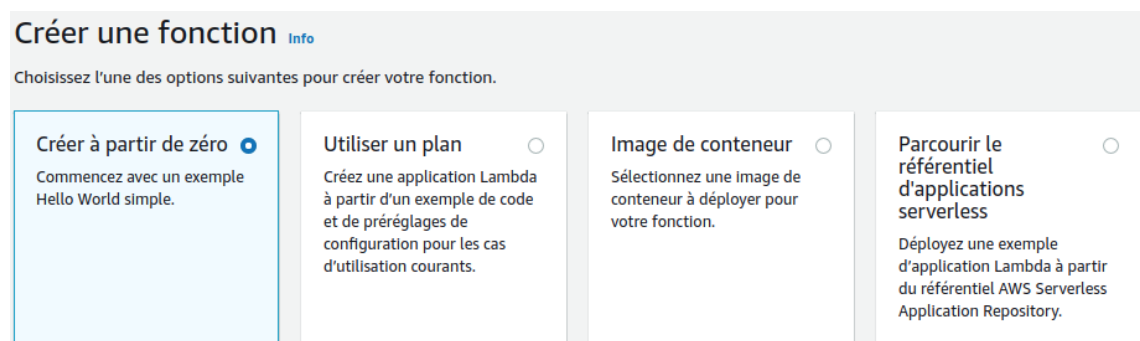
des serveurs. Derrière les coulisses, la plateforme AWS alloue automatiquement les ressources nécessaires pour exécuter le code téléchargé selon la demande.

Le service Lambda a été sélectionnée pour ce projet dû à la simplicité de configuration du service comparativement à Fargate, qui nécessite de containeriser les modules de code avant leur téléchargement et de configurer le moteur ECS responsable de l'exécution des conteneurs. De plus, le service Lambda permet également l'exécution de conteneurs de code, même si cette fonctionnalité ne sera pas utilisée dans ce projet.

5.6.2 Création et configuration d'une fonction Lambda

Ci-suit un exemple de création de fonction. La console propose de créer la fonction à partir de zéro, depuis un exemple de code, depuis un conteneur ou depuis un catalogue d'exemples d'application Serverless.

Figure 19 : Options de création d'une fonction



Pour ce projet, toutes les fonctions sont créées à partir de zéro. Il suffit ensuite de choisir un nom de fonction, une version d'un langage de programmation disponible (ici NodeJS 14), puis de configurer le rôle d'exécution de la fonction.

Le rôle d'exécution fait référence aux rôles IAM qui définissent quelles permissions sont données à la fonction. Ici, un rôle Lambda-DynamoDB a été créé afin de donner un accès complet aux tables DynamoDB à la fonction.

Figure 20 : Configuration d'une fonction

Informations de base

Nom de la fonction
Entrez un nom qui décrit l'objectif de votre fonction.

Utilisez uniquement des lettres, des chiffres, des traits d'union ou des traits de soulignement sans espace.

Exécution [Info](#)
Choisissez le langage à utiliser pour écrire votre fonction. Notez que l'éditeur de code de la console prend uniquement en charge Node.js, Python et Ruby.

Node.js 14.x

Autorisations [Info](#)
Par défaut, Lambda créera un rôle d'exécution avec les autorisations de charger des journaux dans Amazon CloudWatch Logs. Vous pouvez personnaliser ce rôle par défaut plus tard lors de l'ajout de déclencheurs.

▼ **Modifier le rôle d'exécution par défaut**

Rôle d'exécution
Choisissez un rôle qui définit les autorisations de votre fonction. Pour créer un rôle personnalisé, accédez à la [console IAM](#).

☐ Créer un nouveau rôle avec les autorisations Lambda de base

☒ Utiliser un rôle existant

☐ Créer un nouveau rôle à partir de la stratégie AWS templates

Rôle existant
Choisissez un rôle existant que vous avez créé et qui doit être utilisé avec la fonction Lambda. Le rôle doit disposer de l'autorisation de charger des journaux dans Amazon CloudWatch Logs.


Lambda-DynamoDB


[Affichez le Lambda-DynamoDB rôle](#) dans la console IAM.

Une fois la fonction créée, la console nous présente un résumé qui illustre le déclencheur et la destination.

Figure 21 : Déclencheur et destination de fonction

▼ **Présentation de la fonction** [Info](#)

 **MaFonctionLambda**


 Layers (0)

[+ Ajouter un déclencheur](#)

[+ Ajouter une destination](#)

Description
-

Dernière modification
Il y a 34 secondes

ARN de la fonction
 `arn:aws:lambda:eu-central-1:282165771638:function:MaFonctionLambda`

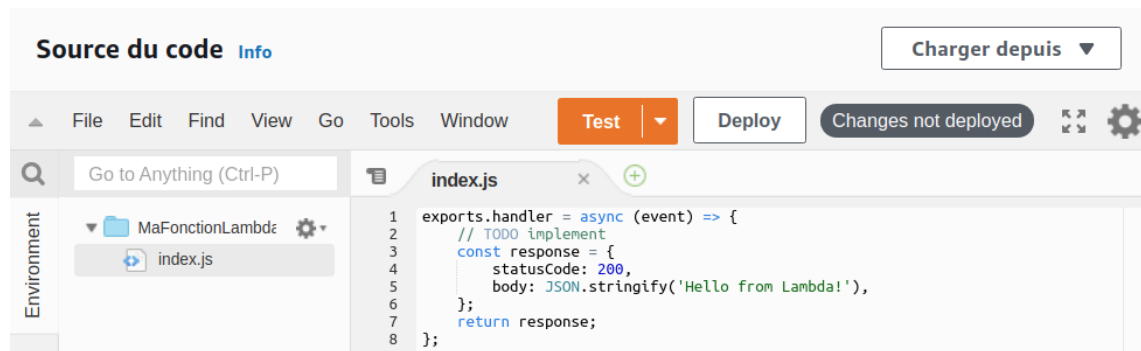
On peut voir sur cette figure qu'un déclencheur et une destination peuvent être assignées à la fonction. Pour les fonctions de ce projet, les fonctions Lambda ont toutes comme déclencheur le service API Gateway, car c'est celui-ci qui route les requêtes vers les fonctions créées.

Les fonctions Lambda de ce projet n'ont pas de destination, car Lambda est la destination finale des requêtes. Les fonctions traitent les requêtes et donnent une réponse au déclencheur (la passerelle API).

Dès la création d'une fonction, la console nous propose un éditeur en ligne de code source contenant un morceau de code d'exemple à modifier selon ses besoins. Le code

source peut être modifié depuis cet éditeur ou téléchargé depuis un poste de travail sous format d'archivage ZIP.

Figure 22 : Editeur de code en ligne



5.6.3 Exemples de fonctions Lambda

Les fonctions Lambda codées pour la réalisation de ce prototype sont disponibles publiquement sur le dépôt de code indiqué dans l'annexe 1. Ci-dessous deux exemples de fonctions du prototype sont présentés.

5.6.3.1 Fonction Lambda : ServerlessConversationsCreate

Ce premier exemple est la fonction responsable de créer une conversation entre deux utilisateurs lors d'une requête POST sur la ressource Conversation de l'API HTTP.

Figure 23 : Code source – création d'une conversation

```
const AWS = require('aws-sdk')
const dynamo = new AWS.DynamoDB()
const { uuid } = require('uuidv4')

exports.handler = async (event) => {
  const conversationId = uuid()
  const data = await dynamo.transactWriteItems({
    TransactItems: [{
      Put: {
        TableName: 'conversations',
        Item: {
          conversation_id: {
            S: conversationId
          },
          username: {
            S: event.user1
          }
        }
      }
    },
    {
      Put: {
        TableName: 'conversations',
        Item: {
          conversation_id: {
            S: conversationId
          },
          username: {
            S: event.user2
          }
        }
      }
    }
  ]
}).promise()
  return data
}
```

En observant le code on peut observer plusieurs choses :

- Deux opérations d'écritures sont exécutées en une transaction à l'aide de la fonction `transactWriteItems` et de l'API `DynamoDB` [55].
- Il est tout à fait possible d'importer une librairie Javascript autre qu'un SDK fourni par AWS (ici, la librairie `uuidv4` permettant de générer des identifiants uniques universels). Ceci est possible en téléchargeant la fonction vers le service Lambda

sous la forme d'un package NPM⁴ archivé au format ZIP et contenant les dépendances nécessaires.

5.6.3.2 Fonction Lambda : ServerlessOnConnect

Cet exemple de code vient de la fonction ServerlessOnConnect, appelée lors d'une nouvelle connexion à l'API Websocket depuis le client Angular.

Figure 24 : Utilisation du DocumentClient de l'API DynamoDB

```
const AWS = require('aws-sdk');
const DDB = new AWS.DynamoDB.DocumentClient();

exports.handler = async (event, context) => {
  const parameters = {
    TableName: 'ws_connections',
    Item: {
      connection_id: event.requestContext.connectionId,
      username: null
    }
  };

  try {
    await DDB.put(parameters).promise();
  } catch (err) {
    return { statusCode: 500, body: 'En error occured while connecting: ' + JSON.stringify(err) };
  }
  return { statusCode: 200, body: 'Successfully connected.' };
};
```

Comme on peut le voir, la fonction utilise le SDK de AWS afin d'instancier un objet DocumentClient, permettant d'exécuter des requêtes sur la table ws_connections. La classe DocumentClient est une classe qui simplifie la forme des requêtes DynamoDB [56].

5.6.4 Gestion des versions des fonctions Lambda

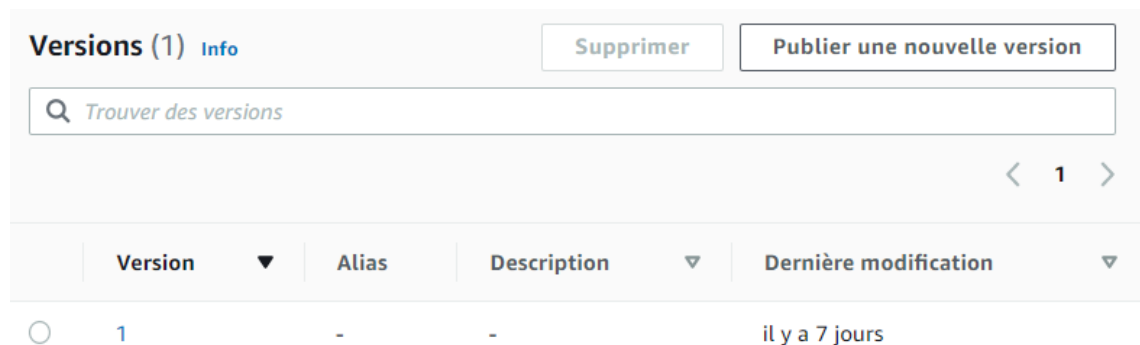
Le service Lambda offre une fonctionnalité de gestion des versions des fonctions.

Pour créer une nouvelle version d'une fonction, il suffit de choisir l'action « Publier une nouvelle version » dans la console de gestion des fonctions. Cette opération enregistre une image de la configuration actuelle de la fonction (y compris son code source) et incrémente la version de la fonction de 1.

⁴ NPM, Node Package Manager, gestionnaire de paquets de NodeJS. [57]

Ainsi, une historique des versions d'une fonction peut être enregistrée au fil des changements importants.

Figure 25 : Gestion des versions de fonctions



La figure ci-dessus montre les versions disponibles de la fonction « FonctionExemple ». Une seule version a été publiée, la version 1.

On remarque que cette fonctionnalité permet de gérer les versions de la couche de calcul de manière découplée à la gestion des versions de la couche API.

5.7 Surveillance des services

CloudWatch est un service permettant la surveillance du fonctionnement des services par la journalisation d'événements [58].

Afin d'activer la journalisation des requêtes sur les API HTTP et Websocket, un rôle a été créé avec les permissions d'écriture vers le service CloudWatch. Ce rôle a été manuellement assigné aux API dans la console de gestion.

Figure 26 : Rôle de création de logs

ARN de rôle	arn:aws:iam::282165771638:role/AmazonAPIGatewayPushToCloudWatchLogs
Description du rôle	Allows API Gateway to push logs to CloudWatch Logs. Modifier
ARN des profils d'instance	
Chemin	/
Heure de création	2021-04-24 14:26 UTC+0200
Dernière activité	2021-06-23 15:15 UTC+0200 (Aujourd'hui)
Durée maximale de la session	1 heure Modifier

La console permet de naviguer parmi les différents groupes de journaux – ou logs – correspondant aux différents ressources AWS émettant des évènements détectés par CloudWatch.

Figure 27 : Exemple de log d'erreur Lambda

Filtrer les événements

Clear

1m

30m

1h

12h

Custom

Horodatage

Message

Aucun ancien événement pour le moment [Réessayer](#)

2021-06-22T17:53:59.296+02:00

START RequestId: 01d51c35-24b0-4b49-b51a-37e5dec07b76 Version: \$LATEST

2021-06-22T17:54:00.582+02:00

2021-06-22T15:54:00.582Z 01d51c35-24b0-4b49-b51a-37e5dec07b76 ERROR Invoke Error {"errorType":

2021-06-22T15:54:00.582Z

01d51c35-24b0-4b49-b51a-37e5dec07b76 ERROR Invoke Error

```
{  "errorType": "ValidationException",  "errorMessage": "ExpressionAttributeValues contains invalid value: Supplied AttributeValue is empty, must contain exactly one of the supported datatypes for key :username",  "code": "ValidationException",  "message": "ExpressionAttributeValues contains invalid value: Supplied AttributeValue is empty, must contain exactly one of the supported datatypes for key :username",  "time": "2021-06-22T15:54:00.582Z",  "requestId": "V9BDJDA0PTE94UOMQRIPE450SBVV4KQNS05AEHVJF66Q9ASUAAJG",  "statusCode": 400,  "retryable": false,  "retryDelay": 33.56138302170029,  "stack": [    "ValidationException: ExpressionAttributeValues contains invalid value: Supplied AttributeValue is empty, must contain exactly one of the supported datatypes for key :username",    "    at Request.extractError (/var/runtime/node_modules/aws-sdk/lib/protocol/json.js:52:27)",    "    at Request.callListeners (/var/runtime/node_modules/aws-sdk/lib/sequential_executor.js:106:20)",    "    at Request.emit (/var/runtime/node_modules/aws-sdk/lib/sequential_executor.js:78:10)",    "    at Request.emit (/var/runtime/node_modules/aws-sdk/lib/request.js:688:14)",    "    at Request.transition (/var/runtime/node_modules/aws-sdk/lib/request.js:22:10)",    "    at AcceptorStateMachine.runTo (/var/runtime/node_modules/aws-sdk/lib/state_machine.js:14:12)",    "    at /var/runtime/node_modules/aws-sdk/lib/state_machine.js:26:10",    "    at Request.<anonymous> (/var/runtime/node_modules/aws-sdk/lib/request.js:38:9)",    "    at Request.<anonymous> (/var/runtime/node_modules/aws-sdk/lib/request.js:690:12)",    "    at Request.callListeners (/var/runtime/node_modules/aws-sdk/lib/sequential_executor.js:116:18)"  ]  }
```

Copier

L'exemple ci-dessus illustre une erreur survenue lors de l'exécution d'une fonction Lambda.

Figure 28 : Exemple de log API Gateway

Clear

1m

30m

1h

12h

Custom

▶

Horodatage

Message

Aucun ancien événement pour le moment [Réessayer](#)

▶

2021-06-30T14:25:42.388+02:00

(BvRfBHLiFiAF18Q=) Extended Request Id: BvRfBHLiFiAF18Q=

▼

2021-06-30T14:25:42.391+02:00

(BvRfBHLiFiAF18Q=) WebSocket API [fr058vhcw5] received message from client [BvRE3dy0liACG-g=]

(BvRfBHLiFiAF18Q=) WebSocket API [fr058vhcw5] received message from client [BvRE3dy0liACG-g=]. Message: [

{

"requestContext": {

"routeKey": "sendMessage",

"messageId": "BvRfBeZJliACG-g=",

"eventType": "MESSAGE",

"extendedRequestId": "BvRfBHLiFiAF18Q=",

"requestTime": "30/Jun/2021:12:25:42 +0000",

"messageDirection": "IN",

"stage": "dev",

"connectedAt": 1625055774925,

"requestTimeEpoch": 1625055942388,

"identity": {

"userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36",

"sourceIp": "213.55.244.50"

},

"requestId": "BvRfBHLiFiAF18Q=",

"domainName": "fr058vhcw5.execute-api.eu-central-1.amazonaws.com",

"connectionId": "BvRE3dy0liACG-g=",

"apiId": "fr058vhcw5"

},

"body": "{\n\"action\": \"sendMessage\", \"content\": \"Lorem ipsum dolor sit amet\", \"sender\": \"jack\", \"conversation_id\": \"7fa8aeab-b275-4a84-9639-0511ba6e3a8d\"}",

"isBase64Encoded": false

}

].

Copier

La figure ci-dessus montre un exemple de log d'une requête Websocket reçue par l'API. On remarque l'identifiant de connexion Websocket dans le contexte de la requête ainsi que le corps de la requête.

5.8 Authentification

Les fonctionnalités liées à l'enregistrement et la connexion des utilisateurs, ainsi que l'authentification de leurs requêtes vers la passerelle API sont implémentées à l'aide du service Amazon Cognito. Ce service a été sélectionné car il est le service privilégié pour gérer des groupes d'utilisateurs d'application [59].

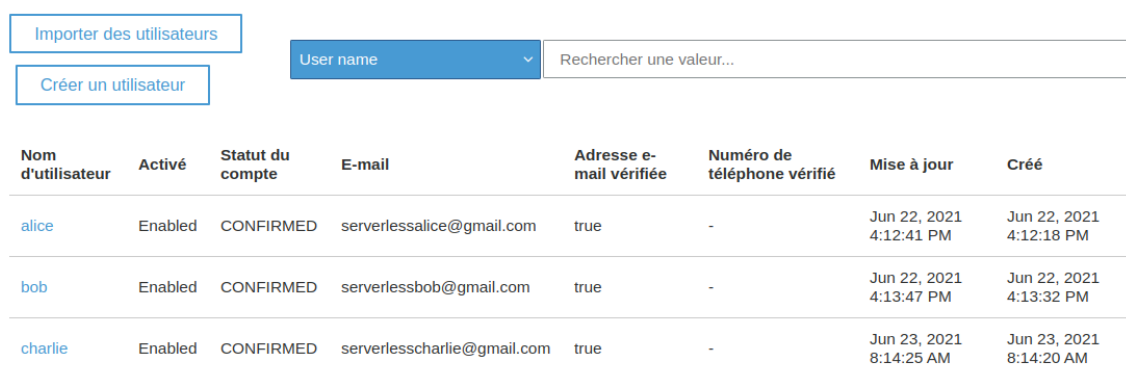
Ce service permet de créer des identités pour les utilisateurs d'une application créée sur AWS. La création de ces identités passe par la création d'une ressource nommée User Pool.

5.8.1 User Pools

Un User Pool est un registre d'utilisateurs stocké sur l'infrastructure AWS [60]. De nombreux paramètres d'un User Pool peuvent être configurés selon les besoins en identification et en authentification de l'application, tels que les champs obligatoires à stocker lors de la création d'un utilisateur et la police de création ou de réinitialisation de mot de passe.

Ces registres d'utilisateurs offrent des fonctionnalités de sécurité disponibles par défaut telles que l'authentification multifacteur (MFA) et peuvent également être fédérés avec des fournisseurs d'identité externes, comme Amazon, Facebook, Google et Apple [60].

Figure 29 : User Pool, liste des utilisateurs



Nom d'utilisateur	Activé	Statut du compte	E-mail	Adresse e-mail vérifiée	Numéro de téléphone vérifié	Mise à jour	Créé
alice	Enabled	CONFIRMED	serverlessalice@gmail.com	true	-	Jun 22, 2021 4:12:41 PM	Jun 22, 2021 4:12:18 PM
bob	Enabled	CONFIRMED	serverlessbob@gmail.com	true	-	Jun 22, 2021 4:13:47 PM	Jun 22, 2021 4:13:32 PM
charlie	Enabled	CONFIRMED	serverlesscharlie@gmail.com	true	-	Jun 23, 2021 8:14:25 AM	Jun 23, 2021 8:14:20 AM

Pour ce projet, un User Pool nommé serverless-demo-pool est créé afin d'enregistrer les utilisateurs de l'application et activer les fonctionnalités d'inscription, de connexion et de réinitialisation de mot de passe.

5.8.2 Enregistrement & connexion des utilisateurs à l'application cliente

Les fonctionnalités suivantes ont été implémentées à l'aide du service Cognito :

- Enregistrement d'un utilisateur avec une adresse email, un nom d'utilisateur et un mot de passe ;
- Confirmation de l'enregistrement avec un code de vérification reçu par email ;
- Connexion à l'application avec un nom d'utilisateur et son mot de passe ;
- Déconnexion de l'application.

Ces fonctionnalités d'authentification sont implémentées directement dans le code source de l'application Angular à l'aide d'un SDK fourni par AWS permettant d'utiliser une API d'authentification sur le User Pool de l'application.

L'application Angular est mise en relation avec le User Pool par la création d'un client d'application dans la console Cognito. Ce mécanisme génère un identifiant unique pour l'application cliente, qu'elle doit utiliser en combinaison avec l'identifiant du User Pool pour faire des requêtes vers le service Cognito en utilisant le SDK.

Du côté de l'application Angular, cet identifiant d'application et de User Pool sont mémorisés dans les variables d'environnement de l'application. Elles sont ensuite utilisées par la classe Angular CognitoService, qui a pour responsabilité d'interagir avec le service Cognito d'AWS au nom de l'application.

Figure 30 : Variables d'environnement Cognito

```
export const environment = {  
  (...)  
  awsUserPoolId: 'eu-central-1_38CR8ssVT',  
  awsUserPoolClientId: '5jifajl0npgedvb04plh28i3gu',  
};
```

Ce service Angular dépend de la librairie NPM amazon-cognito-identity-js, le SDK publié par AWS permettant aux applications Javascript de se connecter à Cognito [61].

Figure 31 : Enregistrement d'un utilisateur depuis l'application Angular

```
import {environment} from "../../environments/environment";
import {CognitoUserAttribute, CognitoUserPool} from "amazon-cognito-identity-js";

const poolData = {
  UserPoolId: environment.awsUserPoolId,
  ClientId: environment.awsUserPoolClientId
}

const userPool = new CognitoUserPool(this.poolData)

signup = (username: string, email: string, password: string, callback) => {
  const cognitoUserEmail = new CognitoUserAttribute({
    Name: 'email',
    Value: email
  })
  this.userPool.signup(username, password, [cognitoUserEmail], null, callback)
}
```

Les vues d'inscription, d'enregistrement, et de confirmation d'enregistrement (cf. annexe 2) ont été développées dans l'application et font toutes appel au service Angular CognitoService de la même manière que cet exemple.

5.8.3 Authentification des requêtes envoyées à API Gateway

Le SDK permet d'implémenter l'enregistrement et la connexion des utilisateurs à une session, mais il reste à sécuriser les requêtes envoyées par le client au service API Gateway.

Pour ce faire, un mécanisme d'autorisation est créé et activé sur l'API en question. Un mécanisme d'autorisation peut être basé sur une fonction Lambda responsable d'autoriser les requêtes ou sur le service Cognito [62].

Figure 32 : Création d'un mécanisme d'autorisation de type Cognito

Créer un mécanisme d'autorisation

Nom *

Cognito

Type * ⓘ

☐ Lambda ☒ Cognito

Groupe d'utilisateurs Cognito * ⓘ

eu-central-1 serverless-demo-pool

Source de jeton * ⓘ **Validation du jeton ⓘ**

Authorization

Ce mécanisme d'autorisation active une vérification de la valeur associée à l'en-tête HTTP Authorization des requêtes reçues par la passerelle API.

Si cette valeur est validée comme étant un jeton d'identité d'utilisateur Cognito, l'API transmet normalement la requête à la fonction chargée de la traiter. Sinon, la passerelle API retourne une réponse HTTP de status 401 (accès non autorisé).

L'application Angular obtient les jetons d'identité d'utilisateur par l'API du SDK lors de la connexion de l'utilisateur.

Figure 33 : Récupération du jeton d'identité Cognito de l'utilisateur

```
this.cognitoService
  .login(username, password, {
    onSuccess: result => {
      const token = result.getIdToken().getJwtToken()
      this.localStorageService.setItem('token', token)
    },
    onFailure: error => {
      this.error = error
    }
  })
```

Ce jeton d'identité est ensuite envoyé avec chaque requête vers l'API dans les en-têtes HTTP.

Figure 34 : Envoi de l'en-tête d'authentification

```
const token = this.localStorageService.getItem('token')
const options = { headers: {'Authorization': token } }

return
  this.httpClient
    .get<Conversation[]>(this.usersUrl + username + '/conversations', options)
```

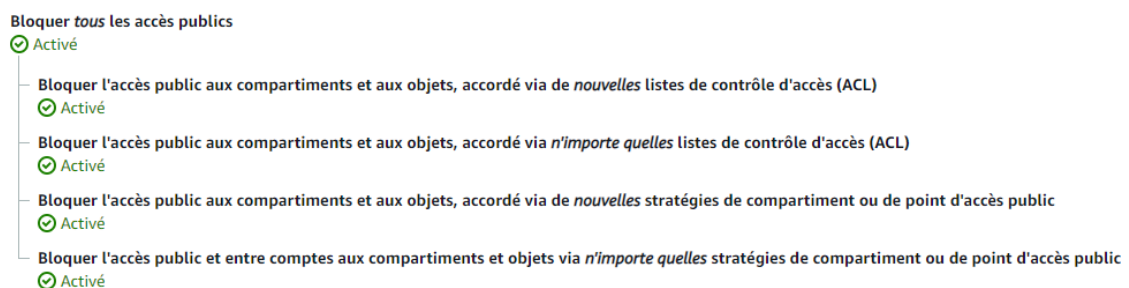
5.9 Stockage de fichiers

Le service S3 est à nouveau utilisé, cette fois pour y stocker les images de profil des utilisateurs.

Un nouveau compartiment S3 est créé, ainsi qu'un utilisateur IAM ayant l'autorisation « S3GetAndPutObjet » permettant de lire et écrire des données stockées dans l'espace de stockage.

Contrairement au compartiment utilisé pour héberger l'application Angular, l'accès public aux données de ce nouveau compartiment est interdit.

Figure 35 : Accès publics au compartiment interdits par défaut



L'autorisation explicite de l'accès au compartiment pour l'utilisateur créé est définie dans la stratégie d'autorisation au format JSON.

Figure 36 : Police d'autorisation accordée à l'utilisateur IAM

```
{
  "Version": "2012-10-17",
  "Id": "Policy1624023824390",
  "Statement": [
    {
      "Sid": "Stmt1624023814128",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::282165771638:user/s3-user"
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
      ],
      "Resource": "arn:aws:s3:::serverless-chat-profile-images/*"
    }
  ]
}
```

Cette police d'autorisation accorde les droits de lecture et d'écriture d'objet à l'utilisateur identifié par son ARN.

Les identifiants de l'utilisateur IAM sont ensuite utilisés dans l'application Angular pour autoriser les écritures et lectures de données. Le client Angular utilise des URL pré signées permettant un accès autorisé explicitement à un compartiment protégé [63].

Figure 37 : Récupération d'une URL pré-signée

```
const credentials = {
  accessKeyId: '*****',
  secretAccessKey: '*****'
};
AWS.config.update({credentials: credentials, region: 'eu-central-1'})

putObject(key: string, file: any) {
  const S3 = new AWS.S3()
  const presignedPostUrl = S3.getSignedUrl('putObject', {
    Bucket: 'serverless-chat-profile-images',
    Key: key
  })

  this.http.put<any>(presignedPostUrl, file)
    .subscribe({
      next: data => {
        console.log(data)
      },
      error: error => {
        console.error(error)
      }
    })
}
```

Comme on peut le voir, les URL non signées sont obtenues par l'utilisation du SDK. Cette méthode d'autorisation garantit que l'accès au compartiment est interdit par défaut, mais qu'un accès spécifique et limité pour le logiciel client est autorisé.

5.10 Conclusion

Ce travail a d'abord présenté une étude des concepts généraux du cloud computing. Cette étude m'a aidé à mieux comprendre un sujet que je connaissais peu, sous un angle académique.

A mon sens, le cloud computing n'est pas une solution miracle pour tous les cas d'usages et toutes les entreprises, mais simplement une évolution naturelle dans la manière d'organiser les systèmes informatiques qui offre de nouvelles possibilités.

Toute adoption de technologie cloud par une entreprise ou un développeur devrait, à mon sens, être accompagnée de toute la due diligence nécessaire pour mitiger les risques associés à ce modèle.

Ce mémoire a ensuite présenté la mise en œuvre d'une application sans serveur construite entièrement sur la plateforme AWS.

Pour le développeur, le cloud computing propose des abstractions de différentes parties de l'infrastructure informatique sous forme de services afin de faciliter le développement d'applications.

Cette expérience pratique m'a permis de découvrir concrètement le cloud computing en tant que développeur avec plusieurs années d'expérience dans le développement d'applications web plus traditionnelles.

Construire une application entièrement à l'aide de services cloud m'a demandé de revoir mes habitudes en tant que développeur. La notion de la tarification des services selon l'utilisation m'a notamment rendu particulièrement conscient du besoin d'optimiser les interactions entre le client et le serveur.

Le débogage de l'application lors des itérations de développement demande aussi des changements d'habitudes mais les outils à disposition, dont CloudWatch, sont suffisamment utiles une fois le flux de travail adapté.

Je pense aussi qu'il est difficile de réellement évaluer certains avantages du cloud computing – comme la scalabilité des ressources informatiques ou la tarification selon l'utilisation – dans le périmètre d'un projet d'expérimentation académique comme celui présenté dans ce mémoire.

Certains aspects du développement d'applications web mériteraient d'avantage d'expérimentation, comme la gestion de plusieurs versions d'application (par exemple, Versions de développement ou production).

Cependant, la mise en œuvre de ce projet sans serveur m'a fait découvrir une méthode de développement, qui, je pense, est très intéressante pour sa flexibilité et sa rapidité.

En conclusion, j'ai beaucoup tiré de ce travail, tant au niveau de mes connaissances théoriques qu'au niveau de mes compétences pratiques de développeur, et j'espère pouvoir continuer à développer ces compétences dans les années à venir.

Bibliographie

- [1] RED HAT. *Le serverless ou informatique sans serveur, qu'est-ce que c'est ?* [en ligne]. [Consulté le 2 juin 2021]. Disponible à l'adresse : <https://www.redhat.com/fr/topics/cloud-native-apps/what-is-serverless>
- [2] RED HAT. *Comprendre ce qu'est une application cloud-native* [en ligne]. [Consulté le 5 mai 2021]. Disponible à l'adresse : <https://www.redhat.com/fr/topics/cloud-native-apps>
- [3] MICROSOFT AZURE. *Serverless computing* [en ligne]. [Consulté le 2 juin 2021]. Disponible à l'adresse : <https://azure.microsoft.com/en-us/overview/serverless-computing/>
- [4] MICROSOFT AZURE. *Qu'est-ce que le cloud computing ?* [en ligne]. [Consulté le 4 juin 2021]. Disponible à l'adresse : <https://azure.microsoft.com/fr-fr/overview/what-is-cloud-computing/>
- [5] GARFINKEL, Simson, 2011. *The Cloud Imperative*. In : MIT Technology Review [en ligne]. 3 octobre 2011. [Consulté le 7 juin 2021]. Disponible à l'adresse : <https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative>
- [6] REGALADO, Antonio, 2011. *Who Coined 'Cloud Computing' ?* In : MIT Technology Review [en ligne]. 31 octobre 2011. [Consulté le 7 juin 2021]. Disponible à l'adresse : <https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/>
- [7] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2011. *The NIST Definition of Cloud Computing* [en ligne]. Septembre 2011. [Consulté le 7 mai 2021]. Disponible à l'adresse : <https://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>
- [8] RED HAT. *IaaS, PaaS, SaaS : quelle est la différence ?* [en ligne]. [Consulté le 30 juin 2021]. Disponible à l'adresse : <https://www.redhat.com/fr/topics/cloud-computing/iaas-vs-paas-vs-saas>
- [9] DELPHISOFT. *Solutions Cloud | Delphisoft France* [en ligne]. [Consulté le 30 juin 2021]. Disponible à l'adresse : <https://www.delphisoft.fr/solutions/solutions-cloud>
- [10] PLOUIN, Guillaume, 2016. *Cloud Computing : Sécurité, gouvernance du SI hybride et panorama du marché*. 4^e éd. Paris : Dunod. ISBN 9782100745128
- [11] HEROKU. *The Heroku Platform* [en ligne]. [Consulté le 10 juin 2021]. Disponible à l'adresse : <https://www.heroku.com/platform>
- [12] LINODE. *What is Cloud Computing?* [en ligne]. [Consulté le 12 juin 2021]. Disponible à l'adresse : <https://www.linode.com/docs/guides/what-is-cloud-computing/>
- [13] CLOUDFARE. *What is a public cloud? | Public vs. private cloud* [en ligne]. [Consulté le 30 juin 2021]. Disponible à l'adresse : <https://www.cloudflare.com/learning/cloud/what-is-a-public-cloud/>
- [14] ALIBABA CLOUD. *Différents Types de Cloud: Cloud Public, Cloud Privé et Cloud Hybride - Alibaba Cloud* [en ligne]. [Consulté le 30 juin 2021]. Disponible à l'adresse : <https://www.alibabacloud.com/fr/knowledge/public-cloud-private-cloud-hybrid-cloud/>

- [15] EUROPEAN NETWORK AND INFORMATION SECURITY AGENCY, 2009. *An SME perspective on Cloud Computing* [en ligne]. Novembre 2009. [Consulté le 17 juin 2021]. Disponible à l'adresse : https://www.enisa.europa.eu/publications/cloud-computing-sme-survey/at_download/fullReport
- [16] CHEN, Thomas; CHUANG, Ta-Tao; NAKATANI, Kazuo 2016. *The Perceived Business Benefit of Cloud Computing: An Exploratory Study*. [en ligne]. [Consulté le 11 juin 2021]. Disponible à l'adresse : <https://scholarworks.lib.csusb.edu/cgi/viewcontent.cgi?article=1297&context=jitim>
- [17] IBM, 2016. *The Benefits of Cloud* [en ligne]. [Consulté le 12 juin 2021]. Juillet 2016. Disponible à l'adresse : https://www.ibm.com/ibm/files/H300444G23392G14/13Benefits_of_Cloud_Computing_634KB.pdf
- [18] MÜLLER, Sune Dueholm; HOLM, Stefan Rubæk; Søndergaard, Jens 2015. *Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective* [Consulté le 11 juin 2021]. Disponible à l'adresse : <https://core.ac.uk/reader/301377170>
- [19] BRENDER, Nathalie et MARKOV, Iliya, 2013. Risk perception and risk management in cloud computing: Results from a case study of Swiss companies. In : *International Journal of Information Management*. octobre 2013. Vol. 33, n° 5, pp. 726-733.
- [20] MOSHER, Richard, 2011. Cloud Computing Risks. ISSA Journal [en ligne]. [Consulté le 11 juin 2021]. Disponible à l'adresse : https://www.experis.us/Website-File-Pile/Articles/Experis/FIN_Cloud-Computing-Risks_071111.pdf
- [21] AUSTRALIAN CYBER SECURITY CENTRE, 2019. Cloud Computing Security Considerations [en ligne]. Avril 2019. [Consulté le 13 juin 2021] 2016. Disponible à l'adresse : https://www.cyber.gov.au/sites/default/files/2019-05/PROTECT%20-%20Cloud%20Computing%20Security%20Considerations%20%28April%202019%29_0.pdf
- [22] EUROPEAN NETWORK AND INFORMATION SECURITY AGENCY, 2009. Cloud Computing: *Benefits, risks and recommendations for information security* [en ligne] December 2012. [Consulté le 17 juin 2021]. Disponible à l'adresse : <https://resilience.enisa.europa.eu/cloud-security-and-resilience/publications/cloud-computing-benefits-risks-and-recommendations-for-information-security>
- [23] Enfermement propriétaire. *Wikipedia, l'encyclopédie libre* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Enfermement_propri%C3%A9taire&oldid=178622355
- [24] GARTNER. *Magic Quadrant for Cloud Infrastructure and Platform Services* [en ligne]. 1 septembre 2020. Disponible à l'adresse : <https://www.gartner.com/doc/reprints?id=1-1ZDZDMTF&ct=200703&st=sb>
- [25] AMAZON WEB SERVICES. *Announcing Amazon Elastic Compute Cloud (Amazon EC2) – beta* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/>

- [26] AMAZON WEB SERVICES. *Infrastructure mondiale* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/about-aws/global-infrastructure/>
- [27] AMAZON WEB SERVICES. *Régions et zones de disponibilité de l'infrastructure mondiale* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : https://aws.amazon.com/fr/about-aws/global-infrastructure/regions_az/
- [28] AMAZON WEB SERVICES. *What is AWS* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/what-is-aws/>
- [29] MICROSOFT AZURE. *Windows Azure General Availability* [en ligne]. 1 février 2010. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://blogs.microsoft.com/blog/2010/02/01/windows-azure-general-availability/>
- [30] MICROSOFT AZURE. *Global Infrastructure* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://azure.microsoft.com/en-us/global-infrastructure/>
- [31] GOOGLE APP ENGINE BLOG. *Introducing Google App Engine + our new blog* [en ligne]. 7 avril 2008. [Consulté le 29 juin 2021]. Disponible à l'adresse : <http://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html>
- [32] GOOGLE CLOUD. *Cloud locations* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://cloud.google.com/about/locations>
- [33] Réseau de diffusion de contenu. *Wikipedia, l'encyclopédie libre* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=R%C3%A9seau_de_diffusion_de_contenu
- [34] AMAZON WEB SERVICES. *Gestion d'identité et d'accès | AWS Identity and Access Management (IAM)* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/iam/>
- [35] AMAZON WEB SERVICES. *Sans serveur sur AWS* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/serverless/>
- [36] AMAZON WEB SERVICES. *AWS CodeCommit | Service de contrôle de source géré* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/codecommit/>
- [37] AMAZON WEB SERVICES. *AWS CodeBuild – Service de génération entièrement géré* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/codebuild/>
- [38] Secure Shell. *Wikipedia, l'encyclopédie libre* [en ligne]. [Consulté le 1 juillet 2021]. Disponible à l'adresse : https://fr.wikipedia.org/w/index.php?title=Secure_Shell
- [39] AMAZON WEB SERVICES. *AWS CodePipeline | Diffusion et intégration continues* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/codepipeline/>

- [40] AMAZON WEB SERVICES. *AWS | Amazon S3 – Stockage de données en ligne dans le cloud* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/s3/>
- [41] AMAZON WEB SERVICES. *AWS | Amazon Resource Names (ARNs) - AWS General Reference* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html>
- [42] AMAZON WEB SERVICES. *AWS | Amazon DynamoDB – Service de base de données NoSQL dans le cloud* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/dynamodb/>
- [43] AMAZON WEB SERVICES. *AWS | Amazon RDS - Gestionnaire de bases de données relationnelles* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/rds/>
- [44] AMAZON WEB SERVICES. *AWS | Fonctionnalités d'Amazon EC2 - Amazon Web Services* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/ec2/features/>
- [45] AMAZON WEB SERVICES. *Amazon ElastiCache – stockage et cache de données en mémoire* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/elasticache/>
- [46] AMAZON WEB SERVICES. *AWS | Amazon Neptune : base de données orientée graphe fiable et rapide, conçue pour le cloud* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/neptune/>
- [47] AMAZON WEB SERVICES. *Core Components of Amazon DynamoDB - Amazon DynamoDB* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html>
- [48] AMAZON WEB SERVICES. *Read/Write Capacity Mode - Amazon DynamoDB* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html>
- [49] Cohérence (données). *Wikipedia, l'encyclopédie libre* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : [https://fr.wikipedia.org/wiki/Coh%C3%A9rence_\(donn%C3%A9es\)](https://fr.wikipedia.org/wiki/Coh%C3%A9rence_(donn%C3%A9es))
- [50] AMAZON WEB SERVICES. *Conception et gestion des API | Amazon API Gateway* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/api-gateway/>
- [51] AMAZON WEB SERVICES. *Working with routes for WebSocket APIs - Amazon API Gateway* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/apigateway/latest/developerguide/websocket-api-develop-routes.html>
- [52] AMAZON WEB SERVICES. *Working with routes for HTTP APIs - Amazon API Gateway* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-develop-routes.html>


- [53] AMAZON WEB SERVICES. *AWS | Lambda - Service PaaS de calculs distribués* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/lambda/>
- [54] AMAZON WEB SERVICES. *AWS Fargate | Moteur de calcul sans serveur | Amazon Web Services* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/fargate/>
- [55] AMAZON WEB SERVICES. *DynamoDB API - Amazon DynamoDB* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.API.html>
- [56] AMAZON WEB SERVICES. *Utilisation du client de document DynamoDB - AWSKit SDK pour JavaScript* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : https://docs.aws.amazon.com/fr_fr/sdk-for-javascript/v2/developer-guide/dynamodb-example-document-client.html
- [57] NPM. *About npm* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://www.npmjs.com/>
- [58] AMAZON WEB SERVICES. *Amazon CloudWatch – Surveillance des applications et des infrastructures* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/cloudwatch/>
- [59] AMAZON WEB SERVICES. *AWS | Amazon Cognito - gestion des accès et synchronisation des données* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://aws.amazon.com/fr/cognito/>
- [60] AMAZON WEB SERVICES. *Amazon Cognito user pools - Amazon Cognito* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html>
- [61] GITHUB. *aws-amplify/amplify-js* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://github.com/aws-amplify/amplify-js/tree/main/packages/amazon-cognito-identity-js>
- [62] AMAZON WEB SERVICES. *Control access to a REST API using Amazon Cognito user pools as authorizer - Amazon API Gateway* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-integrate-with-cognito.html>
- [63] AMAZON WEB SERVICES. *Sharing an object with a presigned URL - Amazon Simple Storage Service* [en ligne]. [Consulté le 29 juin 2021]. Disponible à l'adresse : <https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html>

Annexe 1 : Code source de l'application

Le code source du prototype développé est disponible à l'adresse suivante :

<https://github.com/jckbrkr/serverless-webchat>


Annexe 2 : Captures d'écran de l'interface du prototype développé



Serverless Web Chat

Log in


[Sign up](#)



Serverless Web Chat

Submit

[Log in](#)




Serverless Web Chat


Confirm

[Sign in](#)

[Sign up](#)




Find users

**bob**

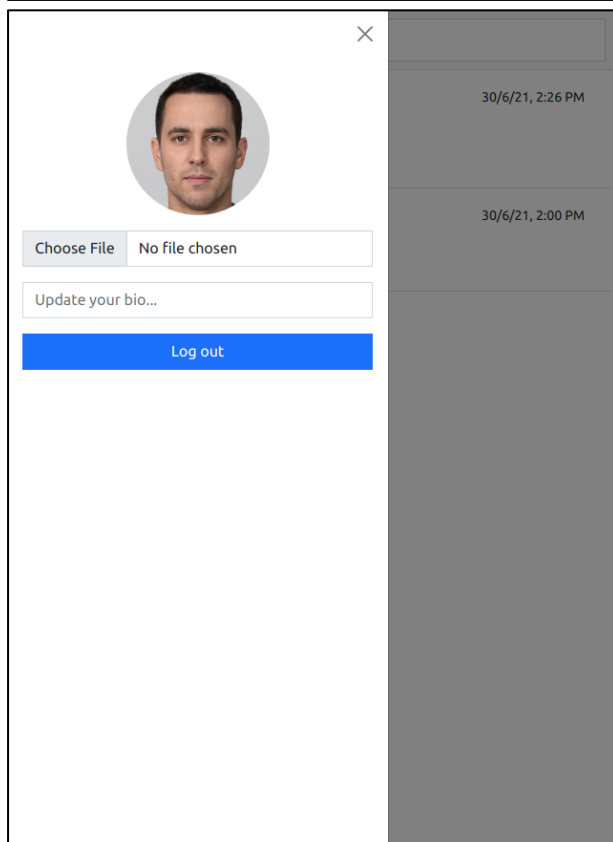
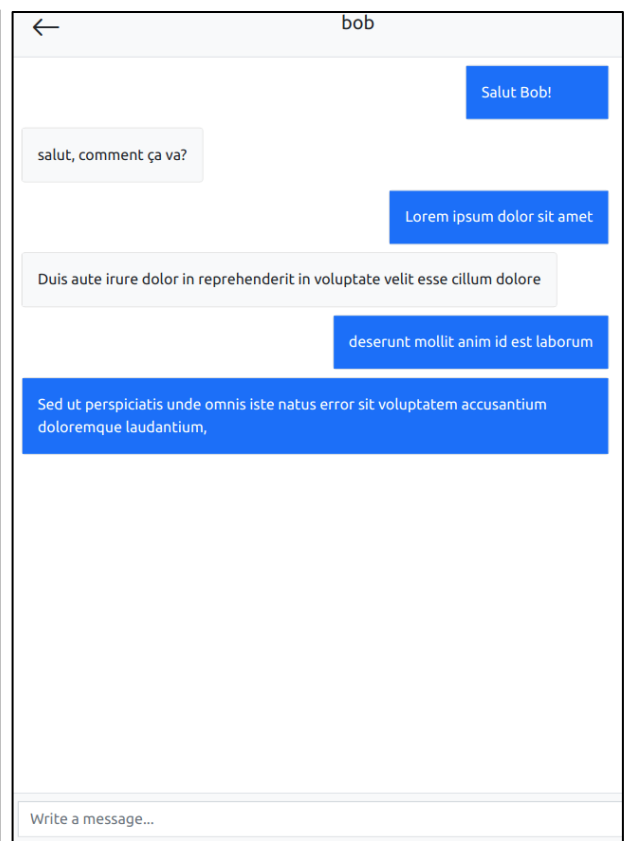
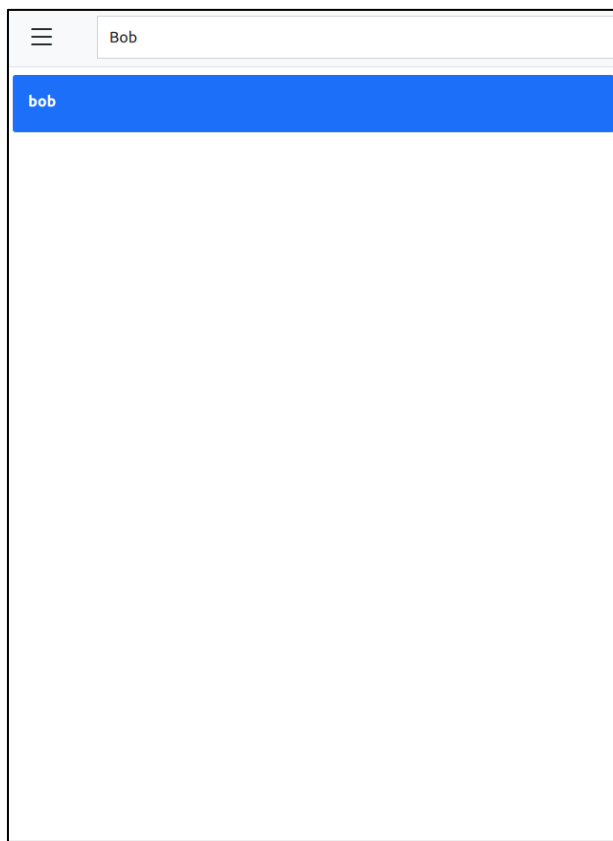
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium,

30/6/21, 2:26 PM

**alice**

Coucou

30/6/21, 2:00 PM



Annexe 3 : Services AWS

Capture d'écran des services AWS proposés depuis la console de gestion disponible à <https://aws.amazon.com/fr/console/>.

▼ All services			
Compute	Customer Enablement	Machine Learning	AWS Cost Management
EC2	AWS IQ	Amazon SageMaker	AWS Cost Explorer
Lightsail	Support	Amazon Augmented AI	AWS Budgets
Lambda	Managed Services	Amazon CodeGuru	AWS Marketplace Subscriptions
Batch	Activate for Startups	Amazon DevOps Guru	AWS Application Cost Profiler
Elastic Beanstalk		Amazon Comprehend	
Serverless Application Repository	Robotics	Amazon Forecast	Front-end Web & Mobile
AWS Outposts	AWS RoboMaker	Amazon Fraud Detector	AWS Amplify
EC2 Image Builder	Blockchain	Amazon Kendra	Mobile Hub
AWS App Runner	Amazon Managed Blockchain	Amazon Lex	AWS AppSync
		Amazon Personalize	Device Farm
Containers	Satellite	Amazon Polly	Amazon Location Service
Elastic Container Registry	Ground Station	Amazon Rekognition	
Elastic Container Service		Amazon Textract	AR & VR
Elastic Kubernetes Service	Quantum Technologies	Amazon Transcribe	Amazon Sumerian
Red Hat OpenShift Service on AWS	Amazon Braket	Amazon Translate	
		AWS DeepComposer	Application Integration
Storage	Management & Governance	AWS DeepLens	Step Functions
S3	AWS Organizations	AWS DeepRacer	Amazon AppFlow
EFS	CloudWatch	AWS Panorama	Amazon EventBridge
FSx	AWS Auto Scaling	Amazon Monitron	Amazon MQ
S3 Glacier	CloudFormation	Amazon HealthLake	Simple Notification Service
Storage Gateway	CloudTrail	Amazon Lookout for Vision	Simple Queue Service
AWS Backup	Config	Amazon Lookout for Equipment	SWF
	OpsWorks	Amazon Lookout for Metrics	Managed Apache Airflow
Database	Service Catalog		
RDS	Systems Manager	Analytics	Business Applications
DynamoDB	AWS AppConfig	Athena	Amazon Connect
ElastiCache	Trusted Advisor	Amazon Redshift	Amazon Pinpoint
Neptune	Control Tower	EMR	Amazon Honeycode
Amazon QLDB	AWS License Manager	CloudSearch	Amazon Chime
Amazon DocumentDB	AWS Well-Architected Tool	Elasticsearch Service	Amazon Simple Email Service
Amazon Keyspaces	Personal Health Dashboard	Kinesis	Amazon WorkDocs
Amazon Timestream	AWS Chatbot	QuickSight	Amazon WorkMail
	Launch Wizard	Data Pipeline	Alexa for Business
Migration & Transfer	AWS Compute Optimizer	AWS Data Exchange	
AWS Migration Hub	Resource Groups & Tag Editor	AWS Glue	End User Computing
AWS Application Migration Service	Amazon Grafana	AWS Lake Formation	WorkSpaces
Application Discovery Service	Amazon Prometheus	MSK	AppStream 2.0
Database Migration Service	AWS Proton	AWS Glue DataBrew	WorkLink
Server Migration Service	Incident Manager	Amazon FinSpace	
AWS Transfer Family			Internet of Things
AWS Snow Family	Media Services	Security, Identity, & Compliance	IoT Core
DataSync	Kinesis Video Streams	IAM	FreeRTOS
	MediaConnect	Resource Access Manager	IoT 1-Click
Networking & Content Delivery	MediaConvert	Cognito	IoT Analytics
VPC	MediaLive	Secrets Manager	IoT Device Defender
CloudFront	MediaPackage	GuardDuty	IoT Device Management
Route 53	MediaStore	Inspector	IoT Events
API Gateway	MediaTailor	Amazon Macie	IoT Greengrass
Direct Connect	Elemental Appliances & Software	AWS Single Sign-On	IoT SiteWise
AWS App Mesh	Amazon Interactive Video Service	Certificate Manager	IoT Things Graph
AWS Cloud Map	Elastic Transcoder	Key Management Service	
Global Accelerator	Nimble Studio	CloudHSM	Game Development
		Directory Service	Amazon GameLift
Developer Tools		WAF & Shield	
CodeStar		AWS Firewall Manager	
CodeCommit		Artifact	
CodeArtifact		Security Hub	
CodeBuild		Detective	
CodeDeploy		AWS Audit Manager	
CodePipeline		AWS Signer	
Cloud9		AWS Network Firewall	
CloudShell			
X-Ray			
AWS FIS			

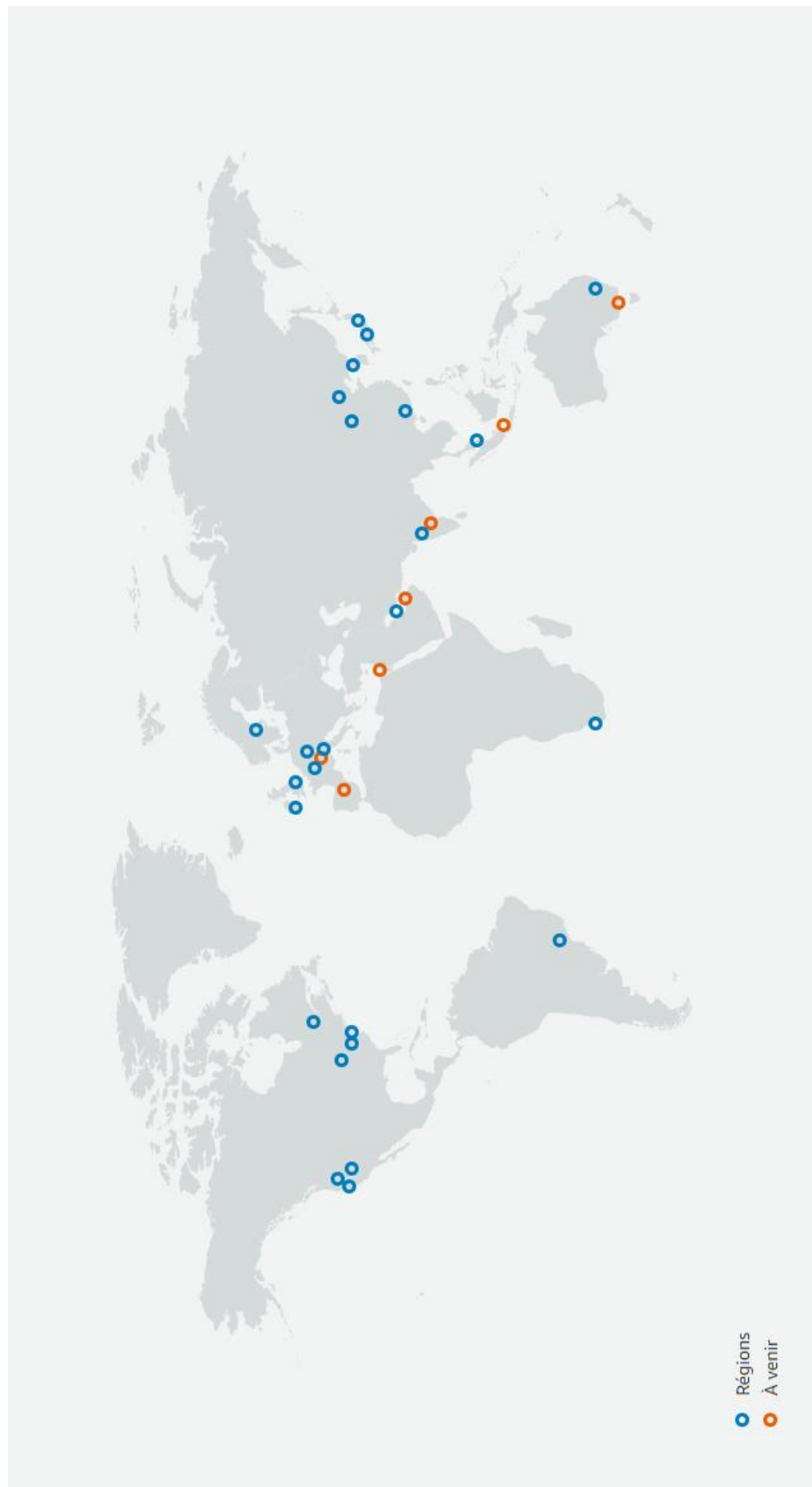
Annexe 4 : Services Google Cloud Platform

Image produite par l'équipe des relations avec les développeurs de Google décrivant tous les services offerts par GCP en 4 mots ou moins. Disponible à l'adresse : <https://github.com/gregsramblings/google-cloud-4-words>.

[illegible]

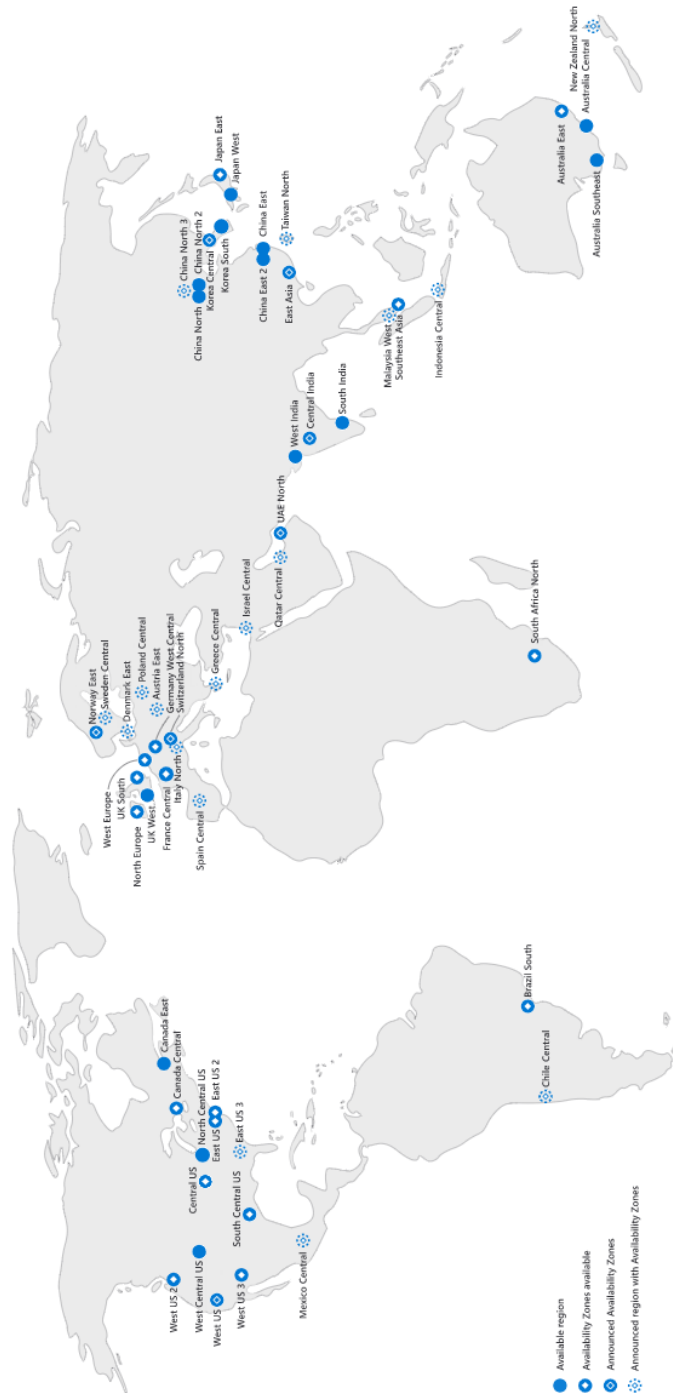
Annexe 5 : Infrastructure mondiale AWS

Carte de l'infrastructure cloud globale de AWS [26].



Annexe 6 : Infrastructure mondiale Microsoft Azure

Carte de l'infrastructure mondiale de Microsoft Azure [30].



Annexe 7 : Infrastructure mondiale GCP

Carte de l'infrastructure mondiale de Google Cloud Platform [32].

