

Rapport PolySnap

Tobias BONIFAY

Ayman BASSY

Igor MELNYK

Mathieu SCHALKWIJK

Analyse du besoin	1
Fonctionnalités	1
Caractéristiques requises	1
Définition du périmètre MVP	2
Diagramme d'architecture logicielle	3
Diagramme de déploiement cloud	5
Modèle de données	6
Coûts prévisionnels	7
Scalabilité	9
Bases de données	10
Stockage des messages	10
Stockage des médias	10
Supervision envisagée du système	11

Essayez l'application [en cliquant ici](#)

Analyse du besoin

Fonctionnalités

Messagerie instantanée

Les utilisateurs peuvent envoyer et recevoir des messages textuels, photos, vidéos et messages vocaux.

PolySmoke : Cette sous-fonctionnalité est cruciale pour la plateforme, permettant aux utilisateurs de partager des médias pour une durée limitée prédéfinie, offrant une sécurité accrue et renforçant la confidentialité des échanges.

Les utilisateurs peuvent échanger en privé ou en groupe. Dans les conversations de groupe, la disparition des messages est gérée pour chaque lecteur indépendamment des autres.

PolyStories

Permet aux utilisateurs de créer une séquence chronologique de médias, visible par leurs contacts pendant une durée de 24 heures.

PolyFindMe

Cette fonction offre aux utilisateurs la possibilité de partager leur position en temps réel avec d'autres utilisateurs au sein d'une conversation.

Caractéristiques requises

Performance et Scalabilité : La plateforme doit être capable de répondre à une charge importante, compte tenu de la nature sociale du produit et de la possible augmentation exponentielle des utilisateurs.

Pertinence des bases de données : Identifier les bases de données les plus adaptées pour gérer différents types de médias tout en garantissant rapidité, performance et sécurité.

Résilience : La solution doit être robuste, capable de gérer des pannes sans interrompre le service pour les utilisateurs.

Bonnes pratiques "12 facteurs" : Adopter une méthodologie moderne pour garantir une application cloud-native agile, évolutive et maintenable.

Prédictibilité des coûts : La structure de coûts doit être transparente, permettant une estimation claire et une allocation budgétaire appropriée.

Indépendance vis-à-vis du cloud provider : La solution doit être conçue de manière à ne pas être trop dépendante d'un fournisseur cloud spécifique et pouvoir ainsi migrer vers d'autres cloud providers en cas de problème.

Sécurité : Les données et interactions des utilisateurs doivent être sécurisées, compte tenu de la nature privée et éphémère de certains contenus partagés.

Définition du périmètre MVP

Au commencement du projet nous avons défini les fonctionnalités pertinentes de l'application.

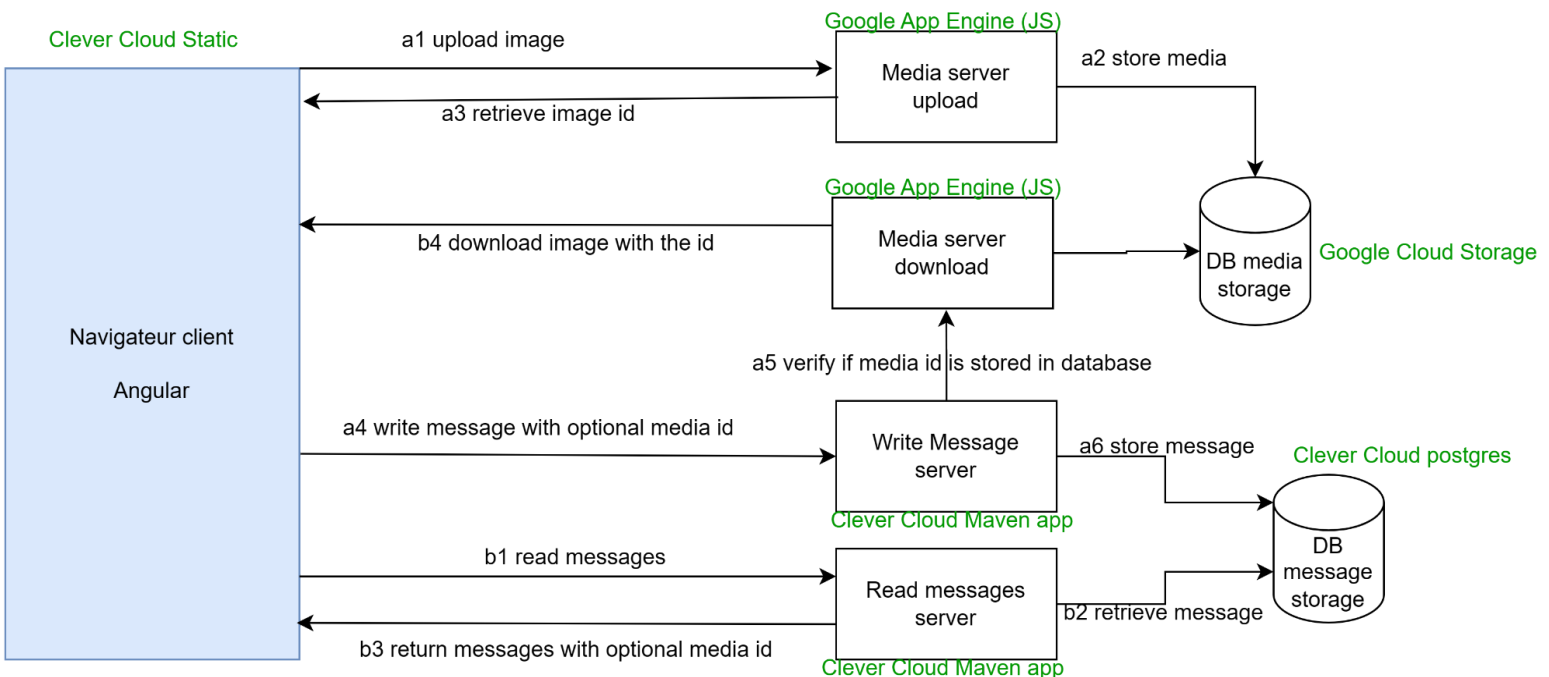
Fonctionnalités implémentées

Fonctionnalités écartées dès le début du projet

Fonctionnalités prévues mais pas implémentées par manque de temps

- Envoie de photos à un autre utilisateur
- Polystories (photos 24h)
- PolySmoke pour les messages dans une discussion privé
- Vidéos
- Garder les photos en bd un certain temps pour des raisons légale
- PolyFindMe pour envoyer la localisation à d'autres utilisateurs
- Groupes
- Messages vocaux

Diagramme d'architecture logicielle



L'architecture se compose des éléments suivants:

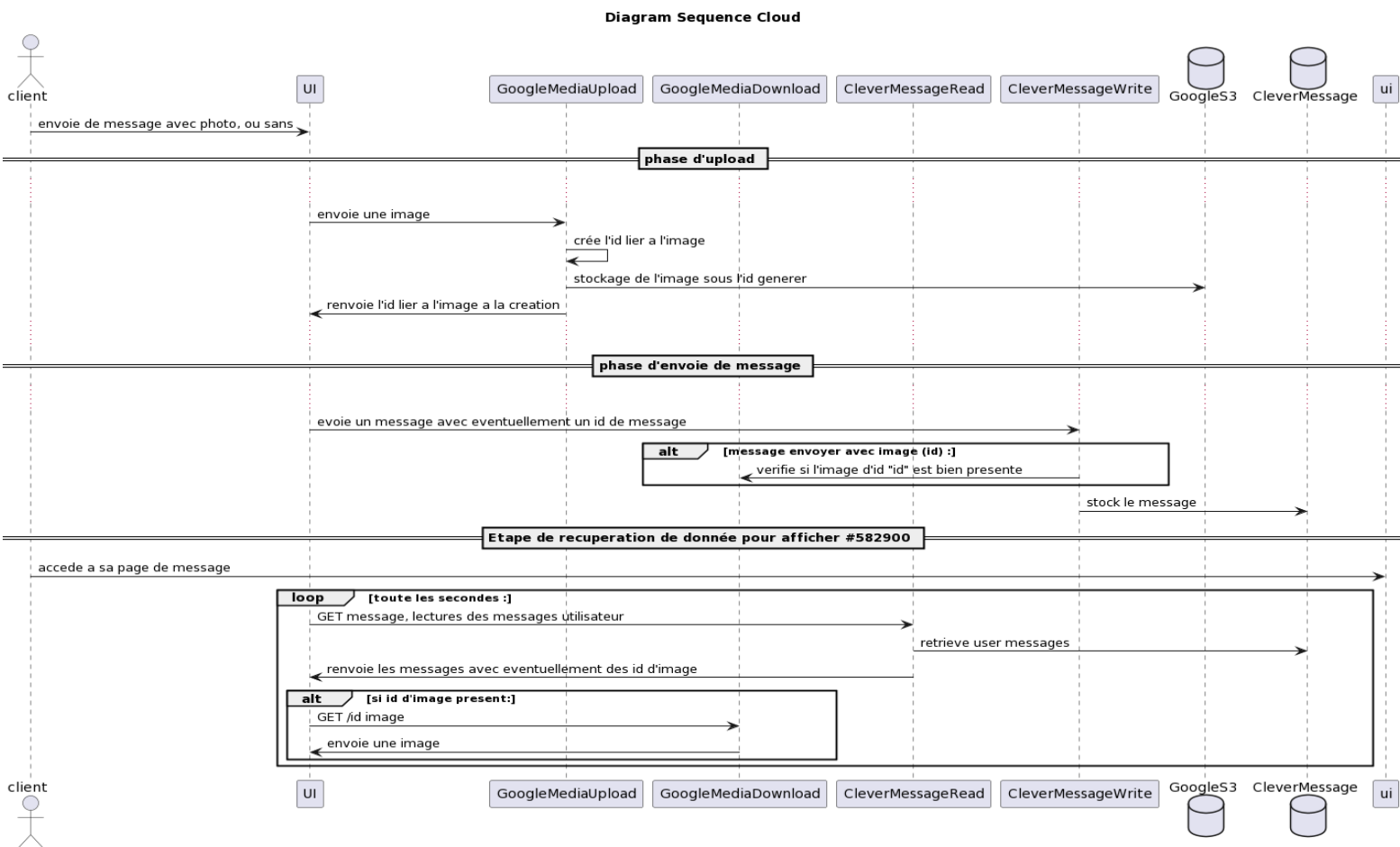
- Un frontend (Angular)
- Un serveur de lecture pour télécharger les médias (nodeJs)
- Un serveur pour envoyer les images du client à la base de donnée (nodeJs)
- Une base de donnée pour les médias (Google bucket)
- Un serveur de lecture des messages (java Spring)
- Un serveur d'écriture des messages (java Spring)
- Une base de donnée pour stocker les messages (PosgreSQL)

Sur le diagramme, le scénario "a" montre le déroulement de l'envoi d'un message avec image :

1. Le frontend envoie l'image au Media Upload Server
2. Le serveur génère un identifiant unique (UUID) et envoie l'image au bucket Google avec l'id généré
3. Le serveur renvoie au navigateur l'id du message
4. Le navigateur envoie le message contenant l'id de l'image au Write Message Server
5. Ce serveur demande au Media Server Download de vérifier si l'image correspondant à cet id unique est bien stocké en base de données
6. Il stocke ensuite le message contenant l'id

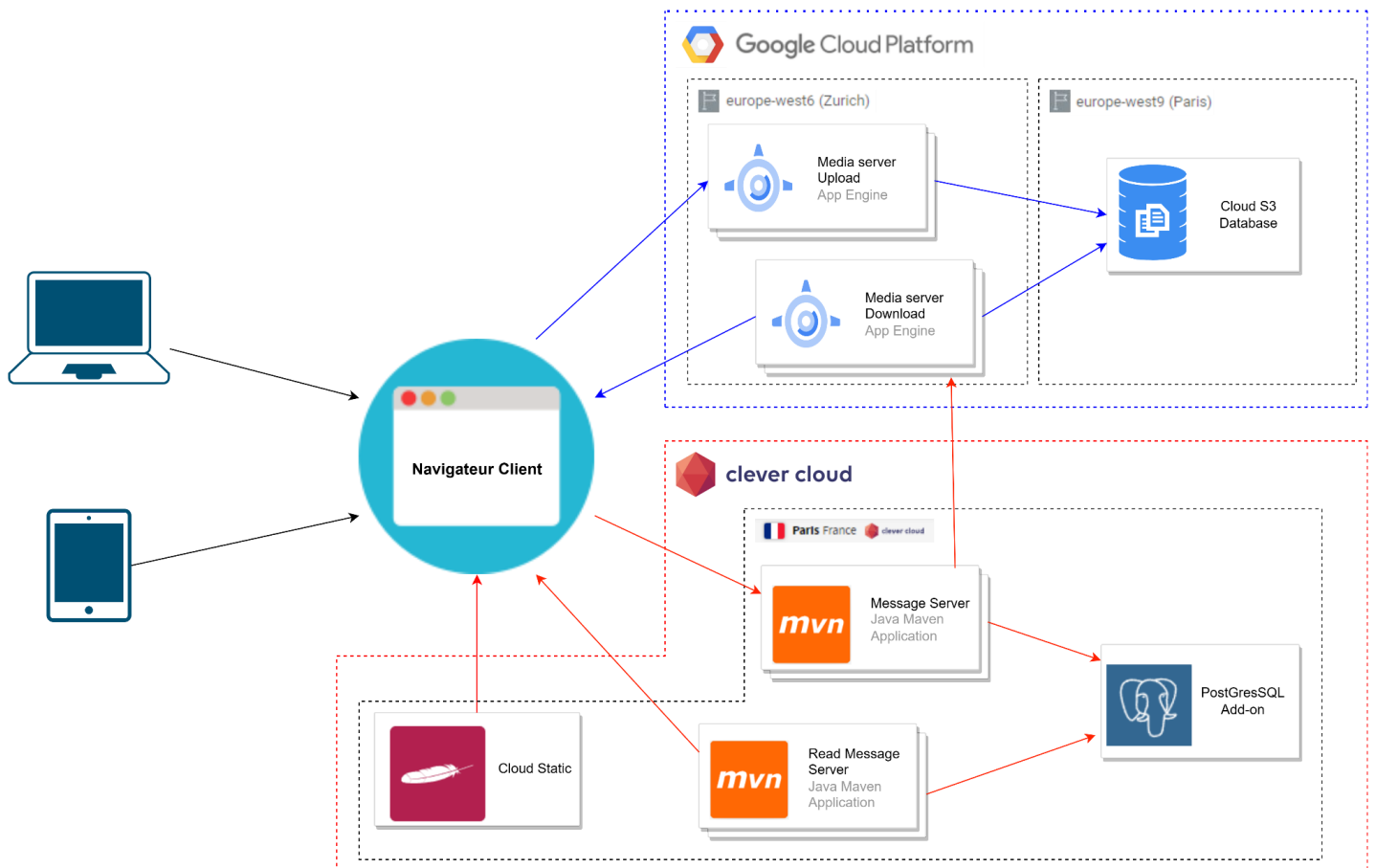
Quand le message ne contient pas d'image, le déroulement est le même mais il s'arrête à la réception du message par l'utilisateur.

Voici un diagramme de séquence pour visualiser le flux :



Potentiel amélioration : Créer un serveur intermédiaire qui fait l'upload de l'image et du message pour que le navigateur n'ai qu'une requête à faire. Le client peut actuellement quitter la page entre le moment où l'image est uploadée et le moment où l'on envoie le message, ce qui peut entraîner des images sans message associé. Cela permettrait aussi d'utiliser un seul serveur pour les requêtes côté frontend.

Diagramme de déploiement cloud



Notre application est hébergée par deux fournisseurs de cloud :

- Google Cloud
- Clever Cloud

L'une des principales motivations derrière notre choix d'héberger notre architecture sur Google Cloud et Clever Cloud est la volonté d'expérimenter et de comparer les services de deux fournisseurs différents. Cette diversification nous offre une perspective plus large et nous aide à éviter une dépendance unique. Le fait d'utiliser des technologies standards comme Maven ou Node nous permet aussi de pouvoir changer de "cloud provider" facilement car en effet, nous avons décidé de déployer sur des plateformes as a service (PAAS).

Une autre raison qui nous a motivés à utiliser plusieurs cloud provider était les coûts des bases de données, en effet, dans notre cas d'utilisation, une base de données PostgreSQL chez clever pouvait être utilisée en mode "nano", avec peu de ressources gratuitement, dans le cas de notre prototype au début, c'était parfait. Dans le cas des bases de données S3, celle de google était aussi la meilleure pour nous à prendre en compte de sa bonne documentation et simplicité d'utilisation. Facilement

utilisable aussi avec un CDN si l'on avait besoin de rendre l'upload plus rapide dans le globe.

Nous avons opté pour une solution Platform-as-a-Service (PaaS) plutôt que Function-as-a-Service (FaaS) pour plusieurs raisons:

- Éviter le cold start: Les solutions FaaS peuvent souffrir de délais d'amorçage, particulièrement lors de la première invocation après une période d'inactivité.
- Moins de dépendance à un fournisseur spécifique: En utilisant PaaS, nous conservons une plus grande flexibilité dans notre choix de technologies, contrairement au FaaS qui pourrait nous lier davantage à des spécificités propres à Google.
- Développement: PaaS nous permet de rapidement coder et push de nouvelle fonctionnalité, des corrections ou des refactor, de les tester rapidement dans un contexte CI et de rapidement les déployer, nous permettant de tester dans un contexte d'environnement de test notre service avec les autres services.
- Temps : En effet, comme énoncer, les contraintes de temps nous pousse à devoir rapidement pouvoir déployer et tester notre service sans nous occuper de tout ce qui entoure la configuration du runtime. Et ainsi pouvoir se concentrer sur le modèle métier de notre application.

Notre approche basée sur des scripts et Terraform nous confère une résilience supplémentaire. Terraform nous permet d'automatiser nos déploiements et les configurations de notre infrastructure. La résilience de notre architecture se situe aussi dans la base de données PostgreSQL de Clever Cloud qui est automatiquement sauvegardée en backup.

Modèle de données

Les données transitant dans notre application sont constituées d'objets Message et ImageMessage.

Voici l'implémentation Angular

```
export interface Message {  
  sender: string,  
  receiver: string,  
  content: string,  
  durationInMinutes: number,  
  creationDate: Date | null,  
}
```

```
export interface ImageMessage extends Message {  
  imageStorageId: string
```

```
}
```

Cet héritage simplifie grandement l'implémentation du frontend et des serveurs de messages et il peut permettre de créer des messages audio ou vidéo sans avoir à refactorer toute la logique des serveurs.

Coûts prévisionnels

Lors du développement, nous avons choisi de limiter la taille des images, afin d'éviter l'augmentation des coûts de réseaux, étant donné l'impact de celui-ci sur le coût total de l'infrastructure. Nous sommes partis sur un poids maximal de 5 Mb qui est largement suffisant pour l'usage de photo éphémère.

Les polystories sont automatiquement supprimés au bout de 24h, par contre, les messages ne sont supprimés qu'après la lecture. Dans les estimations de coûts, il y a de nombreux facteurs... nous avons donc fait des hypothèses afin de pouvoir donner des chiffres approximatifs pour le coût de l'année 1.

Hypothèse 1: Nous prévoyons 1370 utilisateurs sur la première année. (1200 étudiants à polytech Sophia antipolis, avec une marge)

Hypothèse 2: Chaque utilisateur enverra en moyenne 2 images par jour (polystories + polysnaps, vu que tous les utilisateurs ne seront pas forcément actifs -> moyenne).



Hypothèse 3: On suppose que chaque polysnap sera (écrit... et lu) 1 fois. (Les stories seront un peu plus lues, mais certains messages ne seront peut-être jamais lus).

Pour le stockage des images sur le bucket de Google, nous avons un calculateur.

Stockage à Paris. Pour un stockage total de: $1370 \text{ utilisateurs} \times 0.005 \text{ Gb} \times 2 \text{ polysnap} = 13.7 \text{ Gb} / \text{jour}$. Pour l'année: $13.7 \times 365 = 5 \text{ To}$.

Vu que toutes les images ne font pas 5 Mb. La marge servira pour les stories qui sont de toutes manières supprimées au bout d'un temps < 24H.

Pour le réseau: 1 écriture + 1 lecture, en Europe (tant pis pour les polytechs hors Europe, ils sont plus que minoritaires et négligeables.) -> $5 \text{ To} \times 2 = 10 \text{ To}$.

Cloud Storage	
1x Standard Storage  	
Location: Paris	
Total Amount of Storage: 5,000 GiB	USD 115.00
Multi-region Egress - Europe: 10,000 GiB	USD 200.00
Always Free usage included: No	
USD 315.00	

La base de données CleverCloud sur Postgre est de 256 Mb. Les messages ne dépassant que très rarement quelques kilo-octets. On peut supposer que 256Mb sera suffisant pour notre base d'utilisateurs.

Type	Shared
Max connection limit	5
Migration Tool	Yes
Max DB size	256 MB
vCPUS	Shared
Memory	Shared

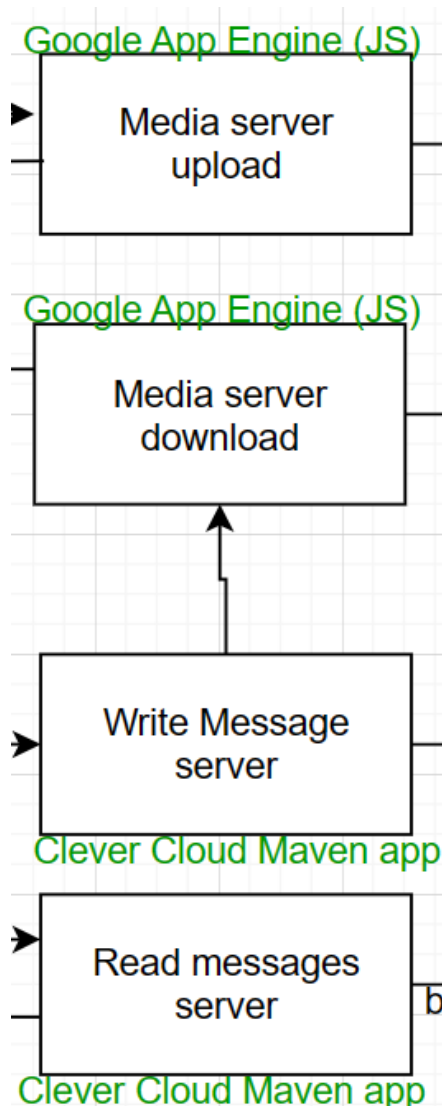
Nous hébergeons aussi, un frontend, un readmessage et un writemessage.

Le frontend est actuellement à 6€ pour de l'Angular et ne coûtera probablement jamais autant que le stockage. (Cela dépend du pic de charge avec le "flexible scaling").

Read et write ont été séparés pour profiter du scaling sur un type spécifique d'échange. Ses applications étant en Java - Spring, ils ont besoin d'un minimum de ram pour fonctionner.. mais scale très bien. Actuellement ils sont en flexible scaling à partir de 1 Gb de ram. Soit 1 x 16€ + 1 x 16€.

Your application will consume between **16.00€** and **2560.00€** for 30 days.

Scalabilité



Les 4 serveurs déployés sont stateless, ce qui permet d'augmenter le nombre d'instances lorsque de nombreuses requêtes arrivent simultanément sans avoir à se soucier de l'état de session de l'utilisateur. L'augmentation du nombre d'instances est effectuée par le cloud provider (Google ou Clever Cloud) selon nos configurations.

En séparant clairement les commandes (écritures) des requêtes (lectures), comme avec `ReadMessageServer` et `WriteMessageServer`, nous améliorons la flexibilité et la scalabilité. Cette séparation nous permet d'optimiser, de scaler, et de maintenir chaque aspect de manière indépendante, maximisant ainsi la performance et la réactivité.

L'intégration d'un potentiel load balancer nous permettrait de distribuer efficacement les requêtes entrantes entre les différentes instances de serveurs. Cela assure non seulement une meilleure utilisation des ressources, mais aussi une réponse rapide, même en cas de forte demande.

Un "Content Devliery Network" (CDN) pourrait améliorer la performance en mettant en cache les fichiers statiques du frontend à proximité des utilisateurs.

La mise en cache des médias des conversations privées ne paraît pas très intéressante car ces médias sont en général téléchargés une seule fois par les destinataires. Cependant une mise en cache pourrait s'avérer utile pour les stories des gros comptes par exemple (si les comptes avaient des abonnés).

Bases de données

Stockage des messages

Le stockage des messages se fait grâce à une base de données PostgreSQL hébergée par Clever Cloud et à laquelle les serveurs de messages se connectent. L'une des raisons de notre choix de PostgreSQL réside dans les compétences existantes de notre équipe. De plus, PostgreSQL s'intègre harmonieusement avec Spring et permet de gagner du temps dans le développement.

PostgreSQL est réputé pour sa forte cohérence des données, une caractéristique essentielle pour nous assurer que les messages de nos utilisateurs sont conservés de manière fiable et sans perte d'intégrité.

Concernant le côté financier, Clever Cloud propose un coût avantageux pour l'hébergement d'une base de données PostgreSQL.

Enfin, les types de données stockés (voir la section [Modèle de données](#)) sont adaptés à une structure relationnelle, par exemple pour faire des opérations sur les utilisateurs où les médias.

Optimisations futures pour la scalabilité : Dans le cadre d'une augmentation significative du nombre d'utilisateurs, l'adoption d'une stratégie de réplication PostgreSQL est pertinente. Un système de réplication maître-esclave (master-slave) permet de dédier le maître (master) aux opérations d'écriture, tandis que plusieurs esclaves (slaves) peuvent gérer les lectures. Cette approche distribue la charge, optimisant la performance et la disponibilité. De plus, en cas de défaillance du maître, l'un des esclaves peut être promu pour assurer la continuité des opérations, renforçant ainsi la résilience de notre infrastructure.

Stockage des médias

Pour stocker les médias (seulement image pour l'instant), nous avons décidé d'utiliser Google Cloud Storage. Les buckets de Google sont conçus spécialement pour le stockage de fichiers volumineux et non structurés, ce qui est un avantage conséquent par rapport à du stockage relationnel sql.

Une caractéristique particulièrement utile de GCS est la possibilité de définir des règles de durée de vie pour les objets. Cela signifie que nous pouvons configurer des fichiers pour qu'ils soient automatiquement supprimés après une période définie, ce qui est pratique pour PolySmoke et pour garantir que les données obsolètes ne s'accumulent pas.

Pour aller plus loin, nous pourrions lier ce stockage au CDN de Google pour distribuer les médias de manière stratégique aux utilisateurs.

Supervision envisagée du système

La supervision et la gestion de l'application s'effectuent sur les consoles de Google Cloud et Clever Cloud où de manière générale on peut avoir les informations sur le statut de tous les services et la charge que ceux-ci reçoivent. En cas de besoin on peut être notifié pour intervenir manuellement s'il le faut ou tout simplement gérer les déploiements.

Risques potentiels

Potentiellement chaque brique de l'application peut tomber à tout moment pour des raisons variées. Clever Cloud affiche son taux de disponibilité à 99.99% par an ce qui se traduit par environ **8,76 heures** de temps d'arrêt non planifié par an. Cloud Storage lui est conçu pour offrir une durabilité annuelle de 99,999999999 % (onze fois le chiffre 9). L'environnement de développement AppEngine de Google affiche aussi un taux élevé de disponibilité à 99,9988%. Cela donne finalement une application hautement disponible et on peut considérer donc que ça reste tout à fait acceptable. Dans notre cas précis, le risque est aussi que le compte Google associé à l'hébergement épuise ses ressources financières et donc il faut vite pouvoir déployer l'ensemble de l'application sur un autre compte.

Solutions

L'architecture elle-même est déjà une solution préparée au fait qu'une des briques peut casser ce qui empêchera d'avoir toutes les fonctionnalités certes mais l'application sera toujours en état de marche. Par exemple, si le serveur d'upload des médias est down on pourra toujours envoyer et recevoir les messages juste on ne pourra pas attacher de photos temporairement en envoyant son message. Le fait d'être chez deux cloud providers diminue les risques car il est très peu probable que les deux tombent en même temps.

Pour ce qui concerne le redéploiement sur un autre compte Google Cloud ou même juste tout redéployer, il suffit de lancer les scripts terraform en donnant les bonnes variables et l'ensemble de l'architecture Google Cloud sera mis en place.

La base de données du côté Clever Cloud fait également des backups réguliers pour éviter qu'on perd toutes les données en cas de problème.

Pour aller plus loin on aimerait pouvoir en cas de besoin déployer l'intégralité de l'architecture qui se trouve du côté Clever sur Google et inversement ce qui permettra de très vite s'adapter au fait qu'un des provider soit en état d'arrêt de manière prolongée.