Architecture TinyPoly - Premier rendu

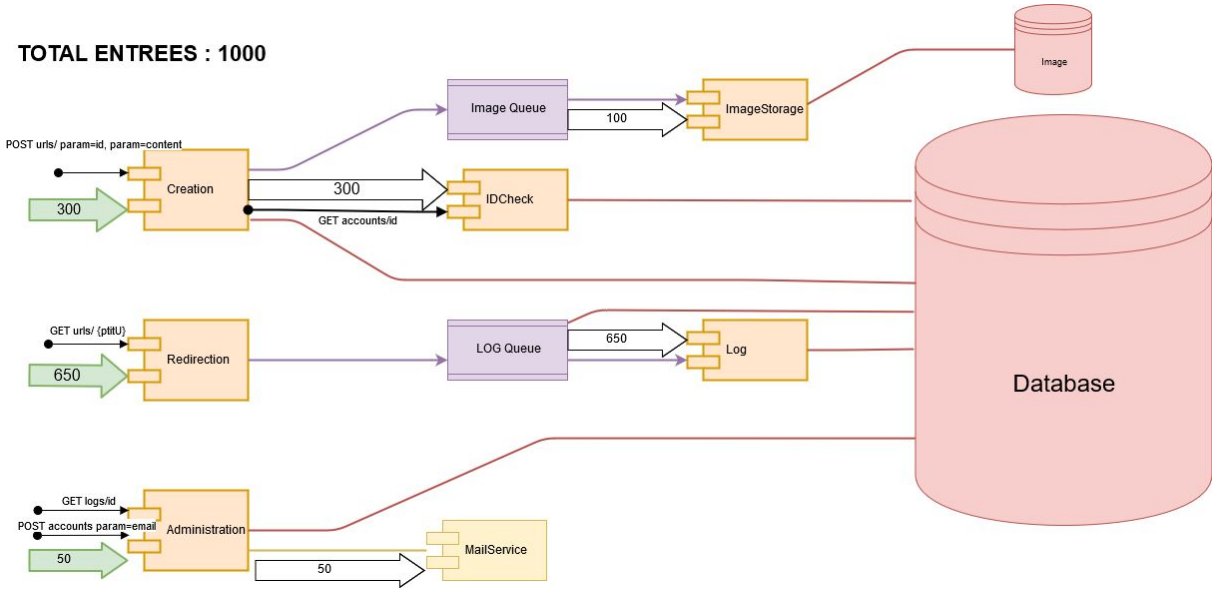
Alexandre Bolot - Hugo Croenne - Hugo François - Prune Pillone

[1. Architecture globale et description précise de l'utilisation des queues](#)
- [1.1 Détail des estimations de charge](#)
- [1.2 Utilisation des queues](#)

[2. Utilisation d'élasticité](#)

[3. Estimation du coût de la plateforme](#)

1. Architecture globale et description précise de l'utilisation des queues



[Lien du diagramme](#)

Les composants oranges (et le mailService) peuvent chacun être déployés sur un serveur séparé afin de permettre à certaines parties de l'application de fonctionner indépendamment, même en cas de défaillance d'une autre.

Nous avons fait le choix d'avoir 2 bases de données :

- Une base de données pour les **images**, utilisant Google Cloud FileStore. Cela correspond à notre besoin, car intégré dans le GoogleCloud Service, et les débits d'écriture/lecture maximaux sont de 100Mo/s, alors que nos images font moins de 4Mo.

- Une base de données pour le stockage des **ptitU**, des **URLS longues**, des **comptes** des **logs** associés, en utilisant Google Cloud Firestore.

1.1 Détail des estimations de charge

Sur **1000** requêtes directes :

- 650 concernent la redirection
- 300 concernent le raccourcissement d'URL
- 50 concernent la création de compte et la demande de log

Les requêtes indirectes sont :

- 300 communiquant sur le service IDCheck provenant de Creation
- 100 passant par la imageQueue pour communiquer avec le ImageStorageService
- 650 passant par la logQueue pour écrire les logs depuis la création et la redirectoin des ptitU

1.2 Utilisation des queues

Nous avons déterminés l'usage de 2 queues :

- ImageQueue
- LogQueue

ImageQueue permet à ImageStorage de traiter les images à enregistrer dans la base de données une à une, et ainsi de diminuer la charge du traitement assez lourd d'enregistrer les images. De plus, on peut se permettre une latence sur le temps de traitement, car il ne s'agira pas de la fonctionnalité la plus utilisée.
La logQueue permet elle à LogService d'absorber la quantité de données à traiter, puisque les logs doivent être très fréquemment mis à jour (à chaque appel de redirection). On peut également se permettre une petite latence ici car les logs ne doivent pas être à jour forcément à chaque instant.

2. Utilisation d'élasticité

En ce qui concerne l'élasticité, nous avons déterminé plusieurs endroits où elle pourrait être appliquée.
Tout d'abord, au niveau de la création et de la redirection des URLs, car ces 2 services sont des points centraux et critiques de notre projet, il est donc nécessaire qu'ils supportent une montée en charge.
Afin que ces fonctionnalités restent disponibles aux utilisateurs. De plus, on peut se permettre une légère latence en limitant la scalabilité sur l'AdministrationService, car la création de compte et la récupération des logs est moins importante en termes de demande que la redirection des URLs.
L'utilisation des queues permettra aussi une scalabilité sur les logs et le traitement des images.

3. Estimation du coût de la plateforme

Globalement, nous allons utiliser un environnement flexible afin de supporter l'élasticité. Les tarifs ci dessous sont calculés à partir des données présentes ici.
Le prix horaire actuel des instances est de 0.05$, Notre choix étant de ne déployer qu'une seule instance pour le développement nous nous paierons uniquement 0.05$ par heure.
Si on suppose 1000 redirections d'URL, cela implique :

- 1000 appels au service redirection soit 0.04$
- Gestion de la queue en comptant 500MB on reste dans ce qui est "offert" par Google. (on considère que cette opération reste rapide et la queue devrait se libérer rapidement).
- 1000 requêtes de réseau sortant de quelques MB, en considérant que les réponses pèsent au maximum 4MB, donc un total de 4GB, cela revient à 0.36$. (1 GB étant offert par mois)

Le reste des requêtes sont des communications internes, donc gratuites.
On arrive donc à 0.40$/1000 redirections, le stockage étant alors pas inclus dans notre calcul.