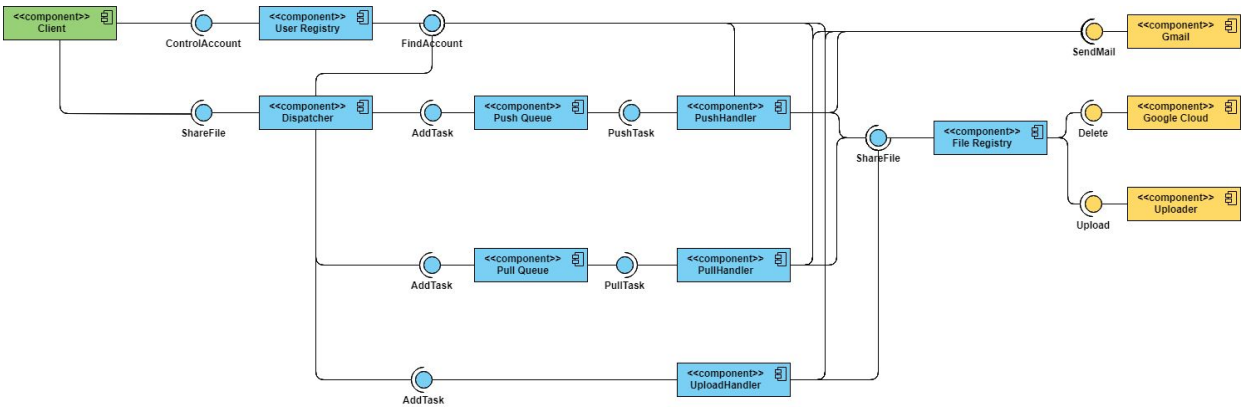
Rendu 1
Cloud computing

DALLA-NORA Enzo
DEGAND Sébastien
HUANG Shiyang
INVERNIZZI Tanguy

Architecture des composants

1) Diagramme des composants



2) Explication des composants

a) Le Dispatcher

Le Dispatcher est le point d'entrée de notre application pour les requêtes d'upload et download.
Il est en relation avec le composant User Registry afin de connaître le rang de l'utilisateur.
Le Dispatcher est également en relation avec la Push Queue afin de transmettre les requêtes de Download des Noobs et avec la Pull Queue afin de transmettre les requêtes de Download des autres utilisateurs.
En ce qui concerne les Upload, le Dispatcher est directement en relation avec le composant UploadHandler afin de transmettre les requêtes.

b) UserRegistry

Ce composant nous offre plusieurs fonctionnalités liées à la gestion des comptes. Il expose des services permettant à un client de modifier les utilisateurs. Il permet également de trouver diverses informations concernant les comptes clients, utilisées notamment dans la répartition des requêtes entre les différentes queues. Il sera également possible de savoir si un client est susceptible de faire une demande ou non.

c) PushHandler

Le PushHandler vient se placer en aval de la push queue des noobs. Ce composant est très similaire au PullHandler dans la façon de traiter les demandes, mais reste différent dans son obtention des demandes. En effet, les requêtes des noobs devant être traitées séquentiellement, il n'est pas utile de "choisir" la prochaine demande que l'on souhaite traiter. Il suffit seulement de vérifier que la limite de une demande par minute est respectée dans notre composant. C'est pour cela que nous avons décidé d'implémenter pour les noobs une push queue et que ce composant sera utilisée dans le traitement de leurs demandes.

d) PullHandler

Le PullHandler récupère les demandes des utilisateurs Casual et Leet contenues dans la Pull Queue selon les critères définis dans la spécification (limites d'opérations et de temps). Une fois que le PullHandler a choisi une requête, il demande à File Registry le lien de Download dans le cas d'un Download. Enfin il utilise l'API de mail afin d'envoyer un mail à l'utilisateur avec la réponse.

e) UploadHandler

Dans la continuité des deux derniers composants étudiés, l'UploadHandler se chargera de l'upload des fichiers, et ce pour n'importe quel utilisateur. Cependant, il ne sera lié à aucune queue puisque cette opération n'est pas compatible avec celles ci.

f) FileRegistry

Ce dernier composant nous offre la possibilité de gérer les fichiers que nous avons traité. Il s'occupe de l'upload et de retrouver les liens de download en fonction de la demande. Il sert également d'interface entre les services d'upload et de download de Google.

3) Workflow

Noob:
Demande d'upload:

- Lorsqu'on reçoit une demande d'upload
- on transmet au UploadHandler
    - On regarde en DB la date de la dernière requête de l'utilisateur concerné
        - Si la date est < 1 min:
            - on rejette la requête et on envoie un mail
        - Si la date est > 1 min:
            - on transmet l'upload au service d'upload
            - on met à jour la date de la dernière requête de l'utilisateur concerné

Demande de download:

- Lorsqu'on reçoit une demande de download
- On l'envoie à la Push Queue. Elle sera traitée par PushHandler
    - Si la date est < 1 min:
        - on rejette la requête et on envoie un mail
    - Si la date est > 1 min:
        - on place la demande dans une push queue réservée pour les noobs. Elle sera traitée séquentiellement avec toutes les autres demandes.
        - on met à jour la date de la dernière requête de l'utilisateur concerné

Casual et Leet:
Demande d'upload:

- Lorsqu'on reçoit une demande d'upload
- on transmet la requête au UploadHandler

a) Si le nombre de demandes est trop important durant la dernière minute:
    i) on rejette la requête et on envoie un mail
b) Si la limite de demande n'est pas dépassée:
    i) on transmet l'upload au service d'upload
    ii) on met à jour les conditions de demandes de l'utilisateur

Demande de download:

- Lorsqu'on reçoit une demande de download
- on place la demande dans une pull queue. Elle sera traitée par un PullHandler
    a) Si le nombre de demandes est trop important durant la dernière minute:
        i) la requête est laissée dans la queue.
    b) Si la limite de demande n'est pas dépassée:
        i) on traite la requête
        ii) on met à jour les conditions de demandes de l'utilisateur

4) Choix des queues

Nous avons choisi d'utiliser 2 queues pour ce projet. Dans un premier temps, nous allons mettre en place une push queue dans le cas des demandes de noobs. En effet, le comportement attendu face aux noobs est un traitement séquentielle par une seule entité. Le parallélisme n'étant pas attendu, une push queue reliée à une seule machine pour traiter ces requêtes répond à tous nos critères.

Cependant pour le cas des utilisateurs de rang Casual et Leet, nous allons mettre en place une Pull Queue. Le comportement attendu est un traitement en parallèle, ce qui signifie qu'il y aura potentiellement plusieurs machine en sortie de celle ci. L'idée derrière cette Pull Queue est que chaque machine (PullHandler) choisie la requête la plus à même à être traitée.

5) Elasticité

Bien que tous nos composants sont pensés pour passer à l'échelle, nous limiterons le nombre d'instance pour certains.

D'une part, nous voulons que les demandes des Noobs seront traitées séquentiellement entre tous le rang. Il faudra alors supprimer l'élasticité et limiter à une seule instance la partie traitement des Noobs. Ainsi, le composant PushHandler ne sera pas passé à l'échelle.

Nous avons choisi cependant d'ouvrir l'élasticité aux autres utilisateurs. Ainsi, si les instances sont dépassées par le travail à fournir, l'application pourra accéder à des machines supplémentaires pour répondre à la demande. Il aura donc la possibilité d'avoir plusieurs instance du composant PullHandler. UploadHandler bénéficiera également de l'élasticité horizontale et devra être partagé par tous les utilisateurs. La queue n'étant pas disponible pour cette fonctionnalité, nous avons choisi de débrider les noobs quant au traitement séquentielle des uploads et de leur offrir un traitement en parallèle si plusieurs instances sont disponible afin de ne pas limiter les autres utilisateurs.

Nous nous sommes également posé la question pour les autres composants de l'application. Le FileRegistry pourra passer à l'échelle si besoin. Cependant, ce composant

n’effectuant pas d’action gourmande en calcul, il pourra supporter beaucoup de requête avant de démarrer une autre instance. Il en est de même pour le composant UserRegistry.

6) Coût de la plateforme

Estimer le coût de la plateforme est complexe et dépend de nombreuses variables. En estimant plusieurs de ces variables, comme le nombre d’utilisateurs ou la charge qui sera portée par les utilisateurs sur ce système, nous avons été capable d’estimer approximativement le coût de la plateforme par utilisateur.

Tout d’abord, nous avons estimé le nombre d’utilisateurs :

	Number of users	
Noob	100	
Casual	5000	
Leet	2000	
	Total users	7100

Pour simplifier les calculs, nous avons ignoré les quotas gratuits que Google offre sur ses services.

	Prix par service par mois	Détails
Backend Noob	36	Pour les noobs, nous avons estimé qu’une simple instance B1, tournant constamment (30*24 heures par mois), au prix de 0.05$ par heure serait suffisante. En effet, celui ci héberge principalement le PushHandler, qui ne sera pas mis à l’échelle.
Backend Casual/Leet	72	Le besoin en ressources de calcul pour les utilisateurs casual/leet est variable, car les composants PullHandler et UploadHandler risque d'être répliqués. Nous avons ainsi doublé le prix du backend des Noob, estimant qu’il faudra parfois seulement utiliser une instance, mais d’autre fois plusieurs en concurrence.
Mail API Service	0,639	Ceci est le calcul du prix des emails : Outbound Data Price in Mb * Average weight in Mb of an email * Number of users * 15 emails per month (estimé)

Au total, en additionnant nos valeurs et en les divisant par nos nombres d’utilisateurs, nous obtenons les coûts suivants :

Total cost of the service per month	Total cost of the service per hour	Total cost of the service per month per user
108,639	0,1508875	0,01530126761