

# Choix des technologies

## Contraintes

Le projet nommé Cubbyhole est un service de partage de fichiers “dans le cloud” qui a pour ambition d’avoir un large public aux quatre coins du globe.

De ce fait, il faut que les données puissent se **répliquer dans des datacenters géographiquement éloignés**.

Nous voulions également absolument pouvoir **maitriser les coûts**, et assurer une **qualité de service** élevée. Dans ce contexte, il a été jugé important de pouvoir assurer une **scalabilité horizontale** des différents éléments de l’application.

Il faut donc notamment **pouvoir stocker un fichier plus gros que la taille du plus gros serveur de l’application**. Il nous tenait également à coeur de pouvoir proposer une technologie relativement **GreenIT**. Il faut cependant conserver une bonne **maintenabilité** de l’application pour faciliter les évolutions et la maintenance.

Il est également important de pouvoir **monitorer** les différents services afin de pouvoir intervenir au plus vite quand cela est nécessaire, mais de conserver un **fonctionnement dégradé en cas de défaillance.**

## Stockage de données

Les données stockées sur l’application ont différentes origines : Métadonnées (informations sur un fichier), données utilisateur (Email, mot de passe, etc.), et fichiers.

Ces données doivent être hautement accessibles, aisément scallables. et rapides.

Nous avons retenu l’utilisation de mongoDB comme moteur de stockage documentaire.

La limite principale de cette solution est la taille maximale d’un document (16Mo), qui est affranchie par la séparation logique des données. Permettant ainsi de stocker un fragment de fichier sur un serveur et un autre fragment sur un serveur différent.

La **scallabilité est assurée** par le système de sharding mongoDB. Pour une clef donnée, les serveurs mongoS détermineront sur quel shard la donnée est disponible.

Chaque datacenter aura **une copie des données** du datacenter “primaire” (concept de **Replica set**). Si le noeud “primaire” devient inaccessible dans un shard, un noeud secondaire sera élu “primaire”.

Les données le plus souvent accédées seront concervées dans le working set (partie de la RAM dédiée à mongoDB, où sont montés les documents) et les accès au disque ne seront pas faits à chaque accès. Le gain est énorme dans le cas d’un fichier public fréquemment téléchargé.

Le service gratuit de 10gen nommé MMS permet de **monitorer** facilement les i/o et la santé des machines et de synthétiser les données quand il y a un grand nombre de serveurs.

Enfin, ce système hautement disponible peut prendre en charge une quantité d’écriture plus grande que la capacité de backup, en garantissant l’intégrité des données à ce niveau, **le projet n’est pas contraint par un système de backups**.

Les données sont ainsi **répliquées dans différents datacenters**, et répliquées depuis le datacenter le plus proche (ou plus rapide).

## Solution logicielle

Afin de pouvoir load balancer les éléments facilement, il faut que les solutions logicielles servent uniquement de relais entre la base et le client (architecture III tiers). Il n’est donc pas nécessaire d’avoir de lourdes technologies telles que PHP. Le projet se servira de nodeJS. Le but étant de **réduire les coûts** liés aux connaissances (un seul langage de programmation à connaître pour le backend et le frontend) et à l’infrastructure, moins de serveurs sont nécessaires en nodejs qu’en php, il est à noter une bonne **rentabilité écologique** découlant de ce choix. De plus, il est possible de **monitorer** un serveur via une application nodeJS.

Les services API et web ont été séparés pour permettre une scallabilité plus fine, et améliorer cette rentabilité écologique.

La **QoS** ne sera que meilleure avec cette technologie récente et **plus performante** que ses principaux concurrents à l’heure actuelle.

Afin de bénéficier de la maintenance de la communauté et de leurs outils, il a été choisi un certain nombre de modules utilisés par le projet dont le **framework express**.

Un des points forts du choix de cette technologie est également le fait d’avoir une grande **synergie** entre MongoDB et NodeJS.

## Support logiciel

La solution logicielle est conçue pour être hébergée sur un serveur à noyau linux. Gagnant ainsi en rapidité et stabilité. Les solutions tel que pacemaker permettant de vérifier que le serveur est toujours en ligne.

Le load balancing peut être assuré par une machine hébergeant HAProxy ou par des **loadbalancers matériels** (LM-5400).

## Documents en relation

Merci de trouver joint les deux fichiers suivants représentants l’infrastructure par datacenter et pour une installation sur trois datacenters.

* Schema by datacenter.png
* Global pattern.png