3PJT - CubbyHole

SUPINFO International University

INSTITUTE OF INFORMATION TECHNOLOGY

Projet de groupe
3PJT - CubbyHole

Rendu

2013 - 2014

Version 1.0
Dernière mise à jour : 16/06/2014
Utilisation : Correcteurs
Auteurs: Le groupe de projet

Sommaire

1 PRESENTATION DU GROUPE

1.1 MEMBRES DU GROUPE

Campus: **Lyon**
Class: **B.s.C**

ID Supinfo	Nom de famille	Prénom
161794	YILMAZ	Fatma
123750	CHEVALIER	Alexis
144074	HUYNH	Eddy
144897	EVIEUX	Jean-Baptiste
145731	CHOLIN	Théodore

2 REPARTITION DE L'EQUIPE

2.1 YILMAZ FATMA - RESPONSABLE INTEGRATIONS ET ETUDES MARKETING

Fatma a effectué plusieurs recherches et études en ce qui concerne la gestion des plans et des solutions de paiement, en effet, elle a implémenté le système de PayPal et de la gestion des plans pour permettre aux utilisateurs d'interagir avec CubbyHole et a contribué au développement du site internet. Les études concernant le marketing ont aussi été menées par Fatma afin d'analyser les coûts de la solution. Fatma a également réalisé plusieurs tests de qualité sur l'application.

2.2 CHEVALIER ALEXIS - CHEF DE PROJET - ARCHITECTE ET DEVELOPPEUR WEB

Alexis a été chargé de mettre en place le projet et de valider les choix techniques, il devait également suivre la progression afin de permettre une avancée linéaire et fluide du projet. Il a contribué à la réalisation de l'API, du système d'authentification, du client Web et du portail des développeurs. Enfin, Alexis a travaillé sur la conception de l'architecture logicielle de la solution CubbyHole.

2.3 HUYNH EDDY - ADMINISTRATEUR SYSTEME ET RESEAUX

Eddy a tout d'abord étudié les besoins de la solution CubbyHole pour pouvoir mettre en place une ébauche de l'architecture système. Après avoir validé la conformité de l'architecture avec le projet, il a mis en place une version de test de cette architecture afin de la soumettre a divers tests et de la rendre adaptable a toute évolution de la solution.

2.4 EVIEUX JEAN-BAPTISTE - LEAD DEVELOPPEUR LOGICIEL/WEB

Jean-Baptiste a principalement travaillé sur la liaison entre les serveurs et les logiciels utilisateurs comme l'application mobile. Il a donc participé au développement de l'API, du site internet et du système d'authentification pour satisfaire les besoins des logiciels utilisateurs. Il a ensuite travaillé sur ces logiciels afin de les rendre compatibles avec la solution.

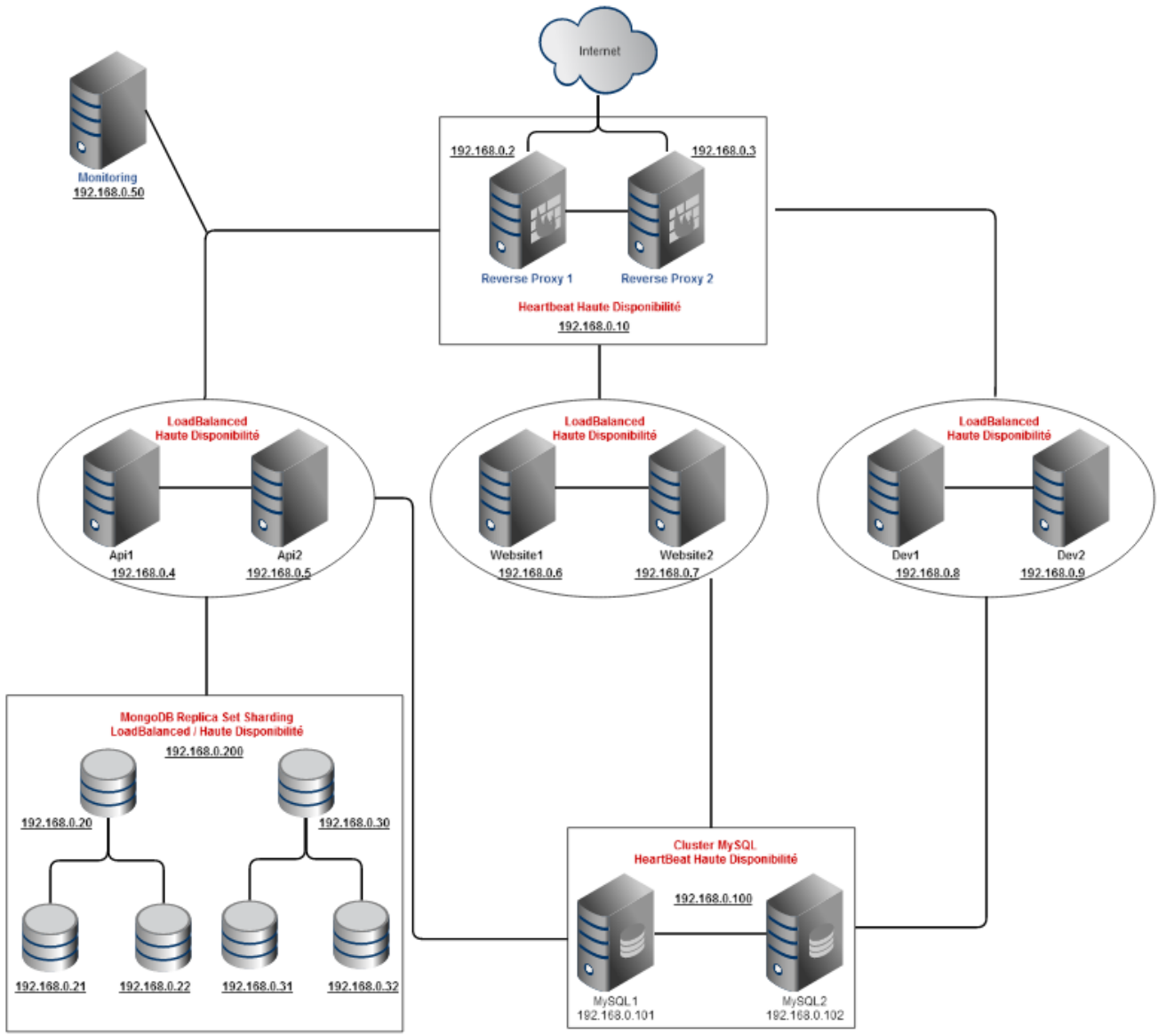
2.5 CHOLIN THEODORE - DEVELOPPEUR LOGICIEL/WEB

Théodore, en tant que développeur Mobile a réalisé une grande partie de l'application Android en analysant les besoins utilisateurs par rapport a une telle application afin de répondre au mieux a la demande. En collaborant avec Jean-Baptiste sur la liaison client-serveur, il a réalisé une application fonctionnelle et ergonomique pour l'utilisateur. Théodore a également contribué aux tests de l'application.

3 MANUEL DE LA SOLUTION

3.1 MANUEL ADMINISTRATEUR SYSTEME

Côté Architecture, nous avons adopté une architecture réseau selon notre schema ci-dessous à savoir que ce modèle est bien évidemment évolutif :



3.1.1 DMZ + Reverse Proxy

Pour protéger notre réseau, nous avons une DMZ sur la zone des Reverse Proxy. Ces Reverse Proxy sont en haute disponibilité grâce à HeartBeat. En résumé, le Reverse Proxy 1 va être le serveur primaire, et si jamais celui-ci tombe, c'est le Reverse Proxy 2 qui va reprendre le service. Nous utilisons HAproxy pour gérer nos Reverse Proxy.

3.1.2 Côté API

Nous avons choisi de mettre en place deux machines hébergeant chacune d'elle l'API. Pour gérer le Load Balancing de l'API, c'est en fait le Reverse Proxy (HAproxy) qui va gérer ceci. Le Load Balancing se fait en fonction de la charge du serveur de l'API. Le moins chargé recevra la prochaine requête. L'API est en haute disponibilité

3.1.3 Côté Website

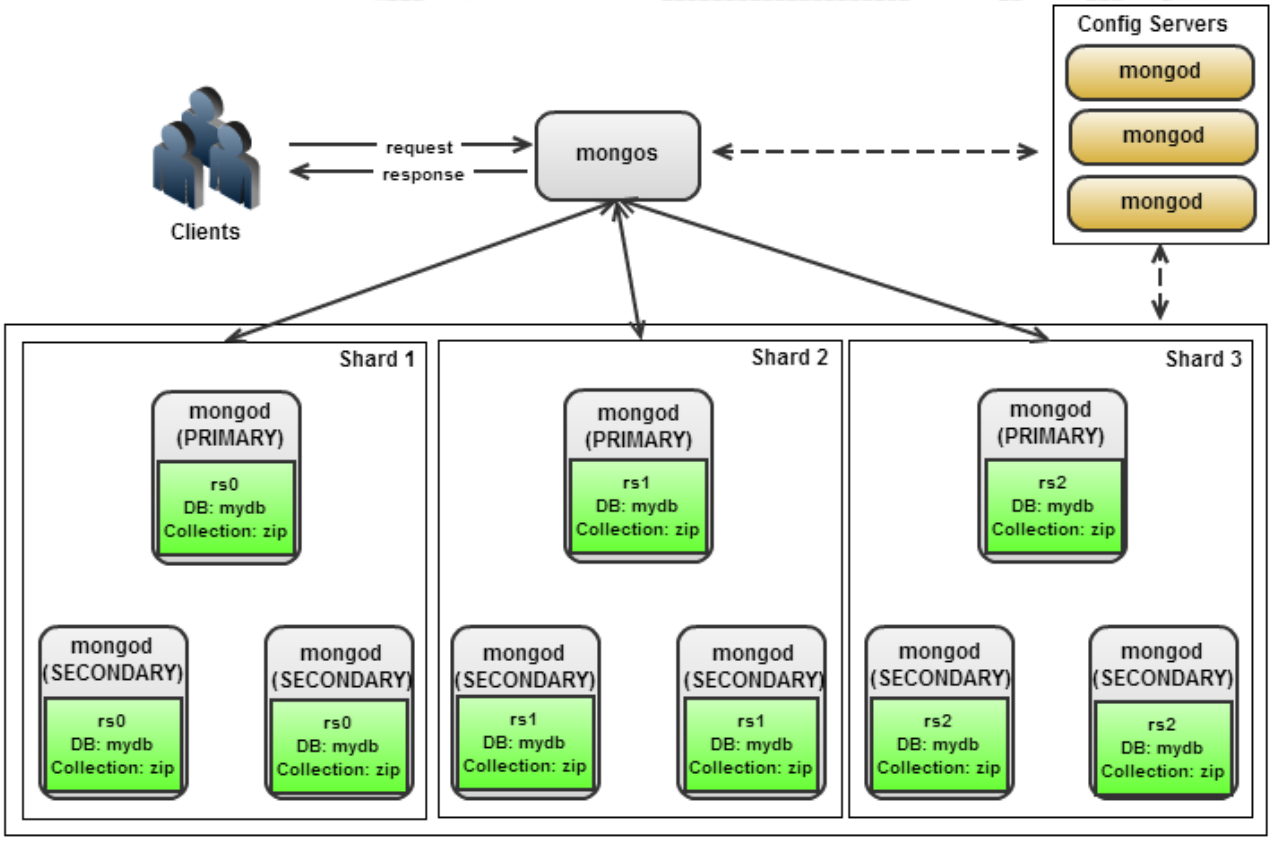
Même idée que pour l'API, nous avons décidé de déployer deux machines hébergeant chacune d'elle le Website. C'est également la même gestion pour le Load Balancing. C'est le Reverse Proxy qui va gérer cette fonction. Le moins chargé sera le serveur qui recevra la prochaine requête. Il y a de la haute disponibilité, car si l'un de deux serveurs tombe, l'autre peut fonctionner seul.

3.1.4 Côté Developers

Ici, nous rejoignons la même politique que l'API et le Website. Le Reverse Proxy va gérer tout ce qui est load balancing en fonction de la charge du server. De même, la haute disponibilité est gérée, puisque si l'un des deux serveurs tombe, le seul serveur pourra fonctionnement de manière autonome.

3.1.5 Storage

Nos fichiers sont stockés grâce à mongoDB. Pour répondre à la haute disponibilité et la réplication, le replicaSet était une solution. Cependant, il ne répondait pas au Load Balancing, qui est géré par le Sharding. Par conséquence, nous avons mis en place un replicaSet Sharding. Voici un schéma pour expliquer le concept :



En résumé, dans notre infrastructure, nous utilisons deux Replica Set , pour répliquer les données, et le sharding pour gérer le load balancing. Il faudra donc, 6 instances de mongoDB (donc 6 Machines) pour gérer notre stockage mongoDB.

3.1.6 Cluster MySQL

Nous avons décidé de gérer le MySQL dans un cluster pour pouvoir gérer la haute disponibilité ainsi que la replication. Dans notre cluster, nous avons un DRBD qui va s'occuper de de répliquer les données du premier serveur sur le second serveur. Le HeartBeat lui, va monter le second serveur MySQL si le premier tombe, avec les données répliquées.

3.1.7 Le Monitoring

Pour monitorer nos serveurs et services, nous avons alloué une machine sur laquelle est installé Shinken. Via le WebClient, nous allons pouvoir observer le fonctionnement de nos serveurs, ainsi qu'être alerté si un service tombe par mail.

3.2 MANUEL ADMINISTRATEUR LOGICIEL

En tant qu'administrateur de la solution logicielle, vous pourrez modifier les plans proposé par la solution CubbyHole.

Pour ce faire, il vous faut disposer d'un compte qualifié du statut d'administrateur pour avoir accès au lien vers le panel d'administration que vous trouverez en haut du site internet.

Une fois arrivé sur cette interface, vous aurez la possibilité de changer les valeurs du plan, qu'il s'agisse du nom, de la description, ou des quotas (Notez que les quotas doivent être saisis en Octets), ou même la disponibilité du plan, vous pouvez tout personnaliser. Cliquez sur 'Mettre a jour' pour modifier le plan en question.

3.3 Manuel Utilisateur Final

L'utilisateur final a accès a plusieurs applications pour interagir avec la solution CubbyHole, nous allons les étudier un peu plus en détail.

3.3.1 Portail d'authentification

Le portail d'authentification est l'élément commun a toutes les applications CubbyHole. Peu importe l'application avec laquelle vous utilisez la solution, vous devrez passer par ce portail pour vous authentifier, une fois authentifiés, vous serez libre d'utiliser l'application avec votre compte, et donc vos fichiers.

Ce portail vous permet également de vous inscrire, pour cela vous avez plusieurs choix :

- Compte 'classique' avec Nom prénom, email et mot de passe
- Compte Facebook (Récupération de votre email et de votre nom)
- Compte Google+ (Récupération de votre email et de votre nom)

Peu importe le système utilisé, vous aurez accès aux même fonctionnalités, sachez que si vous utilisez un compte social, nous ne publierons **jamais** sur votre compte.

Enfin, vous pouvez également demander une restauration de mot de passe au cas ou vous perdez ce dernier, en demandant ceci, nous vous fournirons un nouveau mot de passe par mail.

3.3.2 Client Web

Le client web est la principale interface car il est totalement intégré au site internet, cette application permet de gérer son CubbyHole via un navigateur web de manière rapide et sécurisée.

Le site est disponible en Français, Anglais, Turque et Allemand, vous n'aurez donc aucun problème pour naviguer !

Le site internet est simple, il permet d'effectuer les actions suivantes :

- Modification du profil (Email, Nom, Mot de passe)
- Suppression du profil
- Modification du plan
- Révocation des autorisations pour les applications tierces
- Paiement d'un plan via PayPal
- Téléchargement des applications
- Exploration d'un dossier partagé publiquement
- Téléchargement de fichiers partagés publiquement

Toutes ces fonctionnalités sont facilement utilisables depuis le site. La fonctionnalité la plus

intéressante étant l'explorateur de fichier. En effet, nous proposons a l'utilisateur de gérer son CubbyHole entièrement depuis une interface web.

L'interface permet toutes les opérations classiques d'un système de fichier via les boutons situés à la droite des fichiers/dossiers :

- Modification du nom d'un fichier/dossier
- Ajout de dossiers
- Transfert de fichiers (Téléchargement ou Envoi)
- Copie de fichiers/dossiers
- Déplacement de fichiers/dossiers

Vous disposez d'un gestionnaire de transferts pour suivre la progression de vos envois vers votre CubbyHole, sont également affichés vos quotas disponibles (Espace disque et bande passante).

Enfin, vous pouvez également partager vos fichiers et dossiers, un menu spécial vous est dédié pour ces actions, il vous est possible de partager entre utilisateurs en définissant des droits d'écritures pour chaque utilisateur. Ou de générer un lien de partage public qui permettra à n'importe quel utilisateur d'accéder a vos éléments stockés dans la limite de vos quotas.

3.3.3 Client Android

L'application Android vient compléter le couteau suisse qu'est votre CubbyHole. Vous pouvez gérer vos fichiers quand vous voulez, ou vous voulez.

Respectant notre interface habituelle, vous retrouverez vos marques rapidement et pourrez alors faire ce que vous désirez.

L'application vous propose les fonctionnalités suivantes :

- Création de dossiers
- Modification du nom d'un fichier/dossier
- Copie de fichiers/dossiers
- Déplacement de fichiers/dossiers
- Téléchargement de fichiers

Grâce a l'utilisation des contrôles natifs du système, vous pourrez rapidement accéder a ces options, ainsi qu'a la fonctionnalité de partage, que vous retrouvez sur cette application. Il est en effet possible de partager entre utilisateurs des dossiers et fichiers directement depuis l'application Android, cependant vous ne pouvez pas encore administrer complètement ces partages ou gérer vos partages publics.

L'application n'est pas encore complète et proposera d'autres fonctionnalités très utiles comme l'ajout de fichiers et des options de partage plus abouties sur ses prochaines versions.

3.3.4 Client bureau

Le client de bureau vous permettra de synchroniser vos fichiers, cependant il n'est pas terminé a l'heure du rendu.

4 DOCUMENTATION TECHNIQUE

4.1 ARCHITECTURE LOGICIELLE

4.1.1 Choix techniques

Voici les divers choix que nous avons effectués :

- Application Web : Node.JS et Angular.JS
- Serveur oAuth2 : Node.JS avec la librairie oauth2orize
- API : Node.JS avec Express.JS
- Centre des développeurs : Node.JS
- Nettoyeur : Node.JS
- Système de fichier Virtuel : MongoDB GridFS
- Application bureau : Java SE
- Application Android : Java Android
- Librairie java CubbyHole

Pour l'application web, l'oAuth2, l'API et le centre des développeurs, nous avons décidé de partir sur Node.JS pour plusieurs points :

- La performance
- La capacité a subir de nombreuses requêtes en parallèle
- Les librairies disponibles dans la communauté
- Les compétences de nos développeurs
- L'API de flux
- Et autres choix mineurs...

Pour le nettoyeur (Plus de détails plus tard), nous avons choisis Node.JS car nous avons développé des librairies de gestion de données pour les applications précédentes et nous voulions nous en resservir.

Pour ce qui est de l'application de bureau, Java SE a été largement choisi pour deux points :

- La compatibilité inter-systèmes
- Les compétences de nos développeurs

Pour l'application mobile, nous avons été obligés de choisir Java Android car l'application devait être native.

4.1.2 Cœur de la solution

Le cœur de la solution se compose de l'API et du serveur oAuth2.

Tout d'abord, il faut comprendre que nous avons choisi de confier la sécurité de nos utilisateurs et de nos applications au protocole oAuth2 pour favoriser l'extensibilité de CubbyHole. En effet, grâce au portail développeurs, chaque personne souhaitant étendre les fonctionnalités de CubbyHole peut créer une application et proposer à des utilisateurs existants de se connecter via notre système.

Une fois connectés, l'API propose 27 fonctionnalités différentes, on citera notamment les fonctionnalités de partage et de transfert de fichiers.

Le transfert des fichiers est optimisé afin de permettre un tunnel direct des donnés depuis le client vers le serveur. En effet, il s'agit d'un flux direct, les données sont stockées en temps réel et la limitation de vitesse est appliquée en direct.

4.1.3 Application Web

L'application web est un navigateur de fichiers réalisé avec Angular.JS, le framework est rapide et propose beaucoup de fonctionnalités pratiques.

Cette application communique avec l'API via un Backend Web qui transfert les requêtes en ajoutant le code d'authentification pour que l'API puisse vérifier l'utilisateur.

4.1.4 Centre des développeurs

Le point fort de notre solution est la mise en place d'un réel écosystème grâce a oAuth2, pour permettre l'extension de cette écosystème il fallait mettre en place un portail proposant une documentation et un formulaire pour créer des applications aux développeurs.

4.1.5 Nettoyeur

Cette application est toute petite, mais elle est cruciale au bon fonctionnement de la solution !

Le nettoyeur est chargé de supprimer les éléments inutiles du système, et notamment les fichiers physiques. En effet, nous avons choisis d'éviter a l'API le lourd travail de suppression des fichier. Cette application est lancée régulièrement par une tache Cron.

L'application aurait du fournir un système de réparation automatisée que nous n'avons pas eu le temps de mettre en place, pour faire simple, en cas de crash d'une des applications, il y a des chances pour que le Système de fichiers virtuel soit corrompu. Si c'est le cas, cette application se chargerait de détecter les erreurs en analysant les parents et les enfants des éléments concernés pour reconstruire l'architecture correcte, ou au moins éliminer les éléments corrompus.

4.1.6 Système de fichiers virtuel

La logique de stockage de notre application repose entièrement sur des documents mongoDB représentant des fichiers et des dossiers.

Par exemple, un dossier va posséder plusieurs paramètres comme ses parents, ses enfants directs, son nom etc.. alors qu'un fichier aura des informations sur son fichier comme la taille ou le mime/type.

Les fichiers physiques sont liés uniquement par référence, ce qui permet de diminuer l'espace disque utilisé par ces fichiers dans certains cas (notamment la copie).

4.1.7 Application Bureau, Android et librairie Java CubbyHole

Ces trois points sont réunis car ils sont liés, en effet nous avons développé une librairie java pour faciliter l'utilisation de l'API de CubbyHole. Cette librairie fournit des méthodes simples et adaptées aux plateformes Java SE et Android pour permettre aux développeurs de ne pas dupliquer du code.

Les applications ont été développées de manière trés propre grâce a cette librairie nous pouvons voir une séparation propre et claire des différents éléments de l'application.

Malheureusement nous n'avons pas pu terminer a 100% ces applications.

4.2 ARCHITECTURE SYSTEME

4.2.1 Ensemble de l'architecture

Notre architecture entière tourne sous l'OS Debian 7. Nous avons choisi cet OS, car il est efficace et consomme peu en ressource. Tous nos serveurs sont sans environnement de bureau pour optimiser les performances. Il est clair qu'il est plus difficile à maintenir un service en ligne de commande, cependant nous avons voulu privilégier la performance au-delà de la difficulté de la mise en place de l'architecture.

4.2.2 Reverse Proxy avec HAproxy

Nous avons choisi de mettre en place notre Reverse Proxy avec HAproxy qui est une solution simple et efficace pour répondre à nos deux besoins, qui sont la haute disponibilité et le Load Balancing. Ce service a été installé sur nos machines, en mettant à jour le fichier sources.list et en récupérant le paquet grâce à apt-get install. Une fois le paquet installé, il suffit de configure son fichier de config : /etc/haproxy.cfg . C'est dans ce fichier de config que nous allons préciser nos serveurs API / Website / Developers pour qu'ils soient en haute disponibilité, et en load balancing selon la charge.

4.2.3 Haute Disponibilité avec HeartBeat

Nos serveurs MySQLs et Reverse Proxy tournent avec un HeartBeat qui nous permet d'avoir de la haute disponibilité entre nos services. Sa fonction est simple, dès qu'un des serveurs primaires soit MySQL, soit Reverse Proxy tombe, le HeartBeat va remonter le service manquant avec le deuxième serveur. Trois fichiers sont importants pour gérer le HeartBeat. Ces fichiers sont les suivants :

Ha.cf qui permet de gérer la configuration générale de HeartBeat

Haresources qui permet de configurer les ressources à mettre en haute disponibilité

Authkeys qui permet d'avoir une clef partagé entre les serveurs du cluster

Nous avons donc essentiellement configuré ces fichiers pour le bon fonctionnement du HeartBeat.

4.2.4 Haute Disponibilité sur nodeJS avec Forever

Notre application tourne sous nodeJS. Nous avons pensé à la mettre également en haute disponibilité en cas de perte de service sur le serveur même. Grâce à Forever, l'application se relancera automatiquement si celle-ci crash. En plus de cela, si le serveur venait un jour à redémarrer, nous avons mis en place un init script avec le System-V, permettant de relancer forever à chaque redémarrage.

4.2.5 Réplication sur MySQL avec DRBD

Notre base de données MySQL gère la réplication de données en cas de perte de données du premier serveur. Pour répondre à cette solution de réplication, nous avons utilisé DRBD. Sur les deux machines, nous avons deux disques durs ajoutés pour qu'ils soient répliqués entre eux. Ainsi, nos données sont sur plusieurs disques, et il y aura donc un moyen de Backup ces données.

4.2.6 MongoDB Réplication / Haute Dispo (Replica Set Sharding)

Notre cœur d'application se gère au niveau du stockage. Sachant que c'est un point sensible et important de notre infrastructure, nous avons voulu mettre en place une base de stockage puissante, redondante, et haute disponibilité. La mise en place est assez complexe car il faut :

Mettre en place un premier Replica Set comprenant donc 3 instances de mongoDB sur les 3 différents serveurs.

Créer un second Replica Set dans 3 autres instances de mongoDB sur 3 autres différents serveurs.

Mettre en place le sharding entre les deux replica set afin que le load balancing puisse s'effectué.

Pour cela, bien créer des fichiers conf pour chaque instance et les lancer en fond de tâche avec la commande mongod –f /cheminducfichier .