UNIVERSITÉ DU QUÉBEC À MONTRÉAL

PROPOSITION D'UNE ARCHITECTURE LOGICIELLE ÉVOLUÉE D'UNE SOLUTION DE DNS AUTORITAIRE ET DYNAMIQUE

RAPPORT DE PROJET

PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE LOGICIEL

PAR
DOMINICK RIVARD

MARS 2017

Table des matières

Liste des figures

Liste des tableaux

Résumé

Nous aborderons dans ce projet de mémoire une solution architecturale cherchant à améliorer le délai de synchronisation des données du DNS autoritaire. Depuis au moins une trentaine d'années, les grands opérateurs d'accès Internet hébergent des solutions de DNS autoritaire et utilisent une architecture traditionnelle basée sur une hiérarchie maître/esclave. Sans cette infrastructure DNS, l'Internet d'aujourd'hui serait difficilement navigable. Il s'agit donc d'un des piliers de l'infrastructure de l'Internet et d'une application réseau extrêmement utile. L'architecture traditionnelle des implémentations DNS est relativement simple à mettre en œuvre, mais occasionne certains inconvénients lorsqu'il est question de mettre l'accent sur la performance de la réplication des données du serveur maître vers le ou les serveurs esclaves. Nous croyons que ceci est principalement dû au fait qu'il s'agit de logiciel conçu il y a une trentaine d'années et qu'elle n'exploite pas les dernières avancées en architecture en développement logicielle.

De plus, de nos jours il est souvent fréquent d'entendre les administrateurs de serveur DNS dire aux usagers d'attendre 24 à 48 heures, afin de voir les changements aux entrées DNS répliquées sur Internet, ce qui leurs donnent une assurance que dans ce délai les données seront mises à jour. Ceci est malheureusement lié à un enjeu de mise en cache, mais aussi de temps de réplication vers les serveurs esclaves et de la performance du mécanisme de synchronisation de données entre les serveurs.

L'objectif de ce projet est de proposer une solution de DNS autoritaire plus performante, qui permettrait de mettre à jour les entrées DNS en temps réel, sinon d'être au moins plus rapide que le 5 minutes réalisé traditionnellement sur les infrastructures existantes. D'ailleurs, nous voulions à la fois améliorer grâce à cette nouvelle architecture la performance, l'accessibilité et la facilité d'utilisation de ce type d'infrastructure.

À la suite de l'analyse des architectures existantes, nous en sommes venus à créer une nouvelle architecture, afin de rencontrer nos objectifs. Dans le document ci-présent, nous expliquerons comment cette architecture basée sur des blocs de construction peuvent interagir les uns avec les autres. Ensuite, nous expliquerons quels sont les mécanismes de mise à l'échelle pour créer et maintenir notre nouvelle infrastructure de DNS autoritaire. Finalement, nous expliquerons comment nous croyons être en mesure de réaliser nos objectifs à l'aide de l'architecture proposée.

Mots-clés

dns, dynamique, architecture logicielle, infrastructure, application réseautique

Introduction

De nos jours, il y a de plus en plus d'appareils électroniques connectés aux réseaux de télécommunications. Il faut donc assigner une adresse IP à tous ces équipements, qui sont connectés, afin que ces derniers puissent communiquer entre eux et échanger des données.

D'ailleurs, chacune des adresses données aux équipements peut se comparer facilement au système de numéros de téléphone ou encore aux adresses civiques données à chacune de nos maisons. Pour trouver une maison dans une municipalité, il faut donner les indications de l'identifiant telles que le code postal, la ville, la rue et l'adresse civique. Cela nous permettra d'identifier le lieu géographique de la maison et cette maison conservera son adresse civique pour toujours.

Il en est de même pour les numéros de téléphone, lorsque vous prenez un abonnement avec une compagnie téléphonique, celle-ci vous assignera un numéro selon votre région et ce numéro sera identifié par un indicatif régional [15] et un code de pays. Le numéro de téléphone vous appartiendra tant et aussi longtemps que vous serez abonné. Par contre, à l'inverse d'une maison, le numéro de téléphone peut être réutilisé par une autre personne à la suite de la résiliation ou la modification du contrat avec la compagnie téléphonique.

Sur le réseau Internet, l'adressage des appareils électroniques suit le même type de schéma que les numéros de téléphone et les adresses civiques, et ce à quelques différences près. Si les adresses ne sont pas statiques, elles peuvent être réutilisées à n'importe quel moment et réassignées à un objet connecté sur le réseau sans préavis. Il s'agit dans ce cas d'une adresse IP dite dynamique. Une adresse IP en version 4 est composée d'une série de chiffres, qui forme un numéro de 4 à 12 chiffre au total et les chiffres sont séparés par un point (ex. : 4.2.2.2).

Une adresse identifie exactement un équipement connecté sur le réseau Internet. Il y a bien sûr des équipements réseautiques, qui permettent le partage d'une adresse IP à plusieurs équipements, mais ce type de partage n'est absolument pas important dans le contexte de la recherche de ce projet informatique.

Le comportement dynamique décrit ci-haut est très intéressant, car il nécessite que nous trouvions un moyen de retrouver notre objet connecté sur le réseau, mais comment faire si ce dernier peut changer à tout moment d'adresse IP ?

À l'instar des adresses téléphoniques, qui utilisent un système de bottin, par exemple les pages jaunes, pour trouver le numéro de téléphone des individus ou compagnies, le réseau Internet s'est doté de son propre bottin. Il s'agit de la technologie de serveur de nom de domaine, communément appelé serveur DNS.

C'est donc dire qu'il y a sur le réseau Internet une infrastructure mise en place par des organismes publics [16] de serveurs de nom de domaines et ils sont accessibles publiquement. Ces derniers

peuvent être interrogés à tout moment par un serveur DNS récursif de nom de domaine. Nous expliquerons plus tard dans ce document la technologie des serveurs récursifs bien qu'elle ne soit pas nécessaire dans ce projet.

Nous pourrions donc penser que le problème fut réglé et qu'à l'aide de ce bottin, nous étions désormais en mesure de connaître l'adresse IP de nos équipements sur le réseau Internet. La réponse est que la problématique fut partiellement résolue, parce qu'à l'époque de son élaboration les réseaux et les appareils connectés au réseau n'étaient pas aussi nombreux qu'aujourd'hui. Or le serveur DNS a été conçu principalement pour les adresses IP, qui ne changent pas dans le temps et qui sont statiques. Notre problématique reste donc complète, car elle concerne les IP dynamiques, qui peuvent changer à tout instant.

Bien sûr, la norme publiée RFC2136 [1] souvent appelée DDNS est venue pallier le problème de manque de dynamisme dans la mise à jour du serveur DNS. La norme définit un mécanisme de mise à jour dynamique dans un délai de 300 secondes. Ainsi qu'un procédé par un système événementiel, qui permet d'avertir les serveurs lorsqu'une mise à jour est disponible et enclencher un échange entre ces derniers.

Il est possible d'implémenter une infrastructure de DDNS depuis bien des années déjà et c'est à partir de cette infrastructure que le projet informatique prendra ces racines. Il faut savoir que l'architecture traditionnelle d'une solution de DDNS est typiquement une architecture Maître-Esclave et que cette architecture ne convenait pas du tout à nos aspirations en matière de performance et de capacité. Dans ce projet, nous avons travaillé à créer une nouvelle forme d'architecture pour une solution de DNS dynamique.

L'architecture que nous proposons dans ce projet possède trois buts principaux, lesquels nous voulions atteindre à travers l'opération d'une solution de DNS dynamique. Ces objectifs sont les suivants :

- La mise à l'échelle horizontale de la solution.
- Les mises à jour d'un enregistrement en temps réel.
- Une architecture modulable et fiable.

Nous sommes donc partis de ces buts et nous avons tenté de créer une architecture de DNS autoritaire.

Dans ce document, nous décrirons le travail que nous avons fait pour atteindre les objectifs fixés plus haut et opérer dans un environnement de test notre solution de DNS dynamique.

L'origine du projet

Dans un contexte professionnel, nous avons dû surveiller un très grand volume d'équipements réseautiques connectés directement sur Internet. C'est de ce très grand nombre d'équipements, jumelé au besoin de surveillance du réseau et de détection des pannes de nos équipements, qu'est venu le besoin de penser à l'architecture d'un système de DDNS.

Le besoin d'origine était de trouver une solution à une problématique d'adressage IP [13] dynamique sur des équipements réseau, qui sont accessibles publiquement. Le problème était principalement relié au prix de l'obtention d'une adresse IP statique auprès des fournisseurs d'accès Internet pour chacun des équipements réseautiques, ainsi que le grand nombre de ces équipements. Les clients ou compagnies désiraient réduire les coûts et ne pas utiliser une adresse IP statique pour chacun des équipements constituant le réseau. Cela rendait notre travail de surveillance beaucoup plus difficile, car les équipements réseautiques se trouvaient tous derrière une adresse IP dynamique et pouvaient changer d'adresse IP à tout moment. Cela ajoutait donc à notre travail de surveillance du réseau une immense complexité de connexion et de recherche de nos appareils, qui étaient connectées sur le réseau Internet public à travers plusieurs fournisseurs Internet selon la région, où l'appareil était hébergé.

Nous devions donc être en mesure de détecter, et ce le plus rapidement possible le changement d'adresse IP des équipements réseautiques situés à travers le Canada et les États-Unis. Dès que l'équipement changeait d'adresse IP, il fallait être en mesure de détecter et signaler le changement à un système de surveillance des équipements tel que le logiciel OpenNMS ou Nagios. Si nous en étions incapables, il fallait alors contacter la personne responsable de l'équipement sur le lieu physique et faire le suivi avec elle. À savoir s'il s'agissait d'une panne d'équipement, un faux positif, une panne du réseau, un problème au niveau du fournisseur Internet ou autre.

Au moment, où le système de surveillance a été mis en place, il y avait un mécanisme de journalisation dans le portail d'accès aux réseaux sans-fil public. Ce mécanisme faisait en sorte que chaque fois qu'un client s'identifiait, l'adresse IP de l'équipement réseau était insérée en base de données avec le résultat de l'accès du client. Afin de ne pas créer de nouveau système et par manque de temps à ce moment, l'entreprise en question opta pour l'utilisation de ce mécanisme pour identifier les changements d'adresse IP des équipements réseautique.

Ce que ce mécanisme ne prenait pas en compte était l'achalandage des réseaux sans-fil. Bien que le réseau soit public, cela ne faisait pas en sorte qu'il y ait automatiquement un utilisateur connecté à tout moment. Par exemple, dans un lieu géographique éloigné un réseau pourrait tomber dans le scénario où il n'y aurait pas de nouveaux utilisateurs pendant plusieurs heures et changer d'adresse IP durant cette période de temps. Cela aurait eu la conséquence de nous faire rapporter une panne ou encore un faux positif sur notre système de surveillance, puisque nous n'étions pas au courant du changement d'adresse survenu sur l'équipement en question. Par faux positif, nous entendons la situation où l'appareil que nous sommes supposés surveiller

change d'adresse IP et que cette dernière est assignée à un autre appareil réseau, qui est un client du fournisseur d'accès Internet et non pas à notre appareil. Cela nous aurait causé le faux positif, en plus de brouiller nos rapports de pannes en y insérant un statut d'appareil fonctionnel, alors qu'en réalité notre appareil s'était fait assigner une autre adresse ou peut-être même qu'il était en panne.

Il n'y a malheureusement pas eu de suite sur ce système de surveillance par contre l'idée de vaincre cette dépendance à l'adressage d'IP statique, m'est toujours apparu comme une problématique à résoudre. Voulant pousser plus loin la réflexion envers cette problématique, il m'est venu l'idée d'utiliser une solution de DNS dynamique.

J'utilisais depuis plusieurs années ce type de solution, à travers un fournisseur de DNS grand public dans le but de me connecter à mon réseau personnel. Je pouvais ainsi accéder à mes fichiers personnels ou encore héberger des serveurs privés. Il ne m'était par contre pas venu à l'esprit de mettre en œuvre ce type de solution pour pallier l'adressage d'IP dynamique au moment de devoir surveiller les équipements réseautiques.

En résumé, la solution existait déjà et elle était disponible depuis plusieurs années. Il fallait par contre se l'approprier et la configurer, en plus de s'assurer qu'elle répondrait à notre besoin.

La solution : le DNS Dynamique

Le DNS dynamique est une solution tout à fait envisageable pour corriger la problématique précédemment énumérée. La solution est bien connue et elle est appuyée par des standards RFC. De plus, il y a plusieurs grands joueurs qui hébergent de telles solutions et qui sont disponibles au grand public, par exemple noip.com, changeip.com et dyn.com. Tous possèdent leurs avantages et leurs inconvénients. Par défaut, toutes les solutions gratuites ou payantes permettent une mise à jour dans un délai de 300 secondes, ce qui se trouve à être une valeur par défaut du standard de DNS dans le RFC2136 [1].

Il faut par contre noter que ce ne sont pas tous les logiciels de surveillance du réseau, qui permettent de résoudre les appareils réseautiques par leur nom DNS. Par exemple, comme nous utilisions OpenNMS [25], nous pouvions utiliser la fonctionnalité [26] d'importation de l'inventaire des équipements par le DNS, afin d'injecter le changement de nos adresses IP par le DNS.

Il aurait donc été possible de se servir d'une solution de DNS dynamique pour surveiller tous les équipements réseautiques décrits dans la section du besoin.

Les serveurs DNS

Dans cette section, nous ferons une rapide et brève description de l'état des logiciels libres des serveurs DNS. Nous expliquerons leurs forces et leurs faiblesses les plus connues.

Le plus connu des serveurs DNS est bien sur le logiciel Bind [9], dont la renommée n'est plus à faire. Dans le monde du logiciel libre, il existe aussi les logiciels PowerDNS [10] et NSD [11], qui sont eux aussi bien connus.

À titre informatif, la figure du tableau [5] suivant tiré du site Wikipédia affiche les fonctionnalités des serveurs DNS dont nous allons faire la comparaison.

Feature matrix [edit]

Server	Authoritative	Recursive	Recursion ACL	Slave mode	Caching	DNSSEC	TSIG	IPv6	Wildcard	Free Software	Interface	split horizon
BIND	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes (since 9.x)	Yes (since 4.x)	Yes	Web[Note 1], command line	Yes
PowerDNS	Yes	Yes	Yes	Yes[Note 2]	Yes	Yes (since 3.0) [Note 3]	Yes (since 3.0)	Yes[Note 2]	Yes	Yes	Web[Note 4], command line	Partial[Note 5]
djbdns	Yes	Yes	Yes	Yes[Note 6]	Yes	Partial[Note 7]	No	Partial via generic records. [10]	Partial[Note 8]	Yes	command line and web (VegaDNS & NicTool)[14]	Yes[Note 9]
dbndns	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Partial	Yes	command line and web	Yes
pdnsd	Partial	Yes	Partial	Partial d	Yes	No [15]	Partial	Yes	Yes	Yes	command line, pdnsd-ctl program	Partial
MaraDNS	Yes	Yes	Yes	Partial[Note 10]	Yes	No	No	Partial	Yes	Yes	command line	No
Posadis	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	command line, API	No
Unbound	Partial	Yes	Yes	N/A	Yes	Yes	No	Yes	N/A	Yes	command line, API	No
Dnsmasq	Partial[Note 11]	No	No	No	Yes	Yes (since 2.69) [Note 12]	No	Yes	Yes	Yes	command line	Partial[Note 13]
NSD	Yes	No	N/A	Yes	N/A	Yes	Yes	Yes	Yes	Yes	command line	No

Figure 1 : Tableau de comparaison des fonctionnalités des serveurs DNS [5].

Le serveur DNS : Bind

Bind est probablement le serveur de facto de l'industrie du DNS, par contre il possède plusieurs inconvénients à son opération. Premièrement, la configuration des zones du serveur DNS repose sur l'utilisation de fichiers statiques de configuration. Il y a, donc un fichier statique sur le système de fichier du serveur par nom de domaine. Imaginez un serveur, qui héberge plus de dix mille noms de domaine et cela voudrait dire qu'il y aurait sur le système de fichier dix mille fichiers de configuration.

Ces fichiers de configuration peuvent devenir rapidement difficiles à gérer. En plus, les opérateurs de serveurs DNS du SANOG [17] rapportent qu'à la suite d'une série de tests, qu'ils ont fait sur le serveur Bind, où ils ont testé le serveur contre une centaine de milliers de petites zones

(~ 113k), le serveur a pris plus d'une heure à démarrer à la suite de l'arrêt de ce dernier. Ils voulaient tester les cas de pannes ou de maintenance d'un serveur possédant plusieurs milliers de zones. Ce cas d'utilisation peut aisément survenir durant la mise en service d'un bureau [53] d'enregistrement de noms de domaine.

D'ailleurs, en plus de devoir héberger plusieurs fichiers statiques sur le système de fichier du serveur local, il est à noter qu'il peut être difficile à donner accès aux fichiers à des administrateurs DNS, qui ne sont pas nécessairement des administrateurs de serveurs.

En outre dans les tests du SANOG, le serveur Bind semblait rapporter des performances de trente mille requêtes par seconde, une fois démarré.

En résumé, le serveur DNS Bind est bien connu et permet de gérer le DNS de façon standard. Par contre, le cas d'utilisation préférable à son utilisation est celui, où il y a quelques noms de domaines avec plusieurs milliers d'enregistrements, car lorsque les enregistrements sont trop nombreux le serveur Bind peut devenir difficile, voire complexe à gérer.

Le serveur DNS : PowerDNS

Le logiciel de serveur DNS nommé PowerDNS [10] est très intéressant, car il permet grâce à un système de plug-in [22] de lire la configuration des noms de domaine depuis plusieurs types de formats. Il propose à la base la lecture de fichiers de configurations des noms de domaine depuis les types de fichiers: Bind, la base de données MySQL, Postgres, LDAP ou encore Oracle. Ceci est un énorme avantage lorsqu'il est nécessaire de transférer une zone d'un serveur Bind ou NSD vers un serveur PowerDNS. Il est à ce moment facile à démarrer l'administration du serveur en y gardant les fichiers au format Bind et en changeant dès que nous sommes plus familiers avec le logiciel la manière d'y conserver les zones.

Le serveur PowerDNS est donc légèrement plus flexible à la base et selon l'étude du SANOG [17], ce dernier propose d'aussi bonnes performances que le serveur Bind. Dans leur recherche, le SANOG en est arrivé à la conclusion, que PowerDNS s'adapte très bien au scénario dans lequel il y aurait plusieurs milliers de très petites zones, en utilisant un plug-in et une base de données SQL pour y conserver les informations des enregistrements de noms de domaine. À vrai dire, ils ont testé le serveur avec le même nombre de zones soit ~113 milles et le serveur a pu démarrer à servir les requêtes en seulement 6 minutes comparativement à Bind, qui en avait pris plus d'une heure.

Un autre aspect intéressant de PowerDNS est le schéma de la base de données de ce dernier. Dans l'éventualité d'utilisation d'un système de SGBD, afin de conserver les informations d'un nom de domaine, ce dernier est composé de deux tables, c'est-à-dire la table des domaines et la table des enregistrements. Ce schéma est donc très petit, en plus peu importe le nombre d'ajouts que vous voulez faire dans ces deux tables, le serveur PowerDNS continuera de fonctionner, tant qu'il y trouve ces deux tables et les colonnes dont il a besoin.

En plus, de cette flexibilité, la communauté derrière le logiciel PowerDNS porte une attention particulière à la sécurité informatique dans le logiciel. Par contre, plusieurs lui reprochent son manque de support des vues [24] DNS que Bind est en mesure d'accomplir. Le non-support des vues est dû principalement par le choix qu'ont fait les concepteurs du logiciel PowerDNS de ne pas les implémenter et par conséquent de forcer les administrateurs DNS d'héberger plusieurs serveurs, si le besoin des vues se fait ressentir.

Le serveur DNS : NSD

Le serveur NSD peut facilement se résumer avec le même texte que le serveur Bind. Il utilise les fichiers de configurations statiques et il est préférable de l'utiliser lorsque le nombre de noms de domaine est petit, mais que ces derniers sont volumineux en quantité d'enregistrements. La différence majeure entre NSD [21] et Bind est que le serveur NSD compile à l'aide de l'outil « zonec » les fichiers de noms de domaine au format Bind dans un fichier de base de données par exemple « zones.db », afin de rendre le démarrage du serveur plus rapide. D'ailleurs, l'intérêt principal envers le serveur NSD est qu'il serait plus sécuritaire et plus performant que le serveur Bind, mais le projet ci-présent ne s'arrêtera pas à faire la comparaison de performance et de sécurité de ces logiciels.

Le serveur DNS récursif

Nous expliquerons brièvement la différence entre un serveur autoritaire et un serveur récursif, afin de nous assurer qu'il n'y a pas de malentendus sur le travail que chacun doit accomplir.

La différence entre un serveur récursif et un serveur autoritaire est simple. Le serveur autoritaire est propriétaire de l'information d'un nom de domaine, car c'est ce dernier, qui connaît la structure du domaine et qui en permettra la modification. Le rôle du serveur récursif est d'interroger les serveurs autoritaires, afin de trouver l'information contenue dans les zones DNS et de distribuer cette dernière au client s'informant de la zone. Les clients d'un serveur récursif sont généralement d'autres ordinateurs, ou serveurs, ou encore des appareils mobiles étant connectés à un réseau Internet cherchant à identifier le nom de l'enregistrement DNS.

La figure [7] suivante explique le travail que fait un serveur récursif pour déterminer l'information qui lui est demandée.

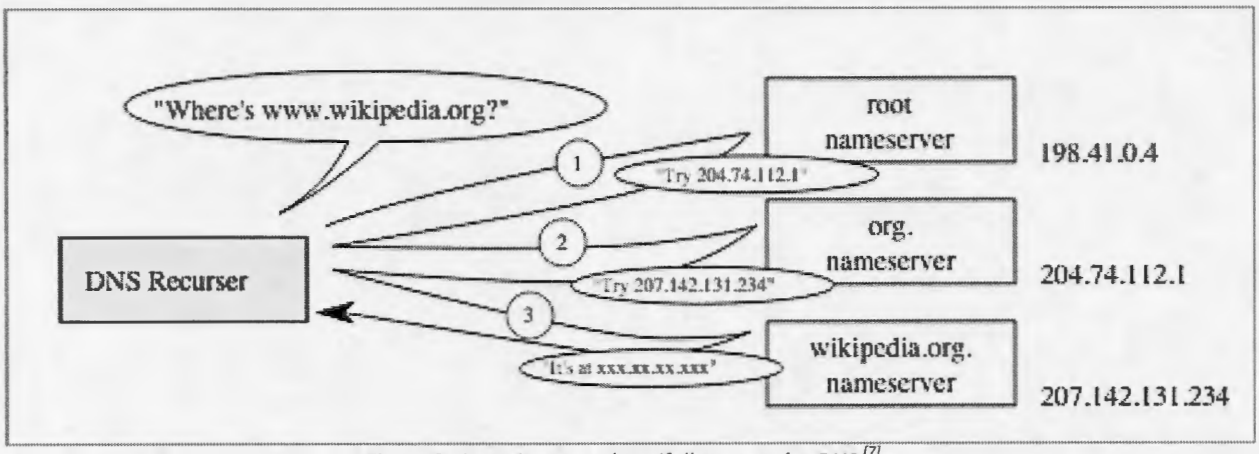


Figure 2 : Le traitement récursif d'une requête DNS [7].

En résumé, le rôle du serveur récursif est de rechercher l'information DNS demandée par un ordinateur et de retourner à ce dernier l'adresse derrière le nom d'enregistrement initialement recherché. Le serveur autoritaire a pour seul mandat de conserver et donner l'information sur les domaines qu'il héberge lorsqu'il est interrogé.

Dans ce travail, l'architecture proposée vise à proposer une amélioration de l'infrastructure des serveurs DNS autoritaires.

L'architecture conventionnelle d'une infrastructure DNS

Il est important dans ce travail de décrire l'architecture conventionnelle d'une infrastructure de serveurs DNS, car cette dernière décrit comment l'infrastructure est organisée et comment celle-ci peut en fait répondre à des problématiques de performance et de capacité. De plus, elle explique comment Bind ou un autre logiciel de serveur DNS suivant les normes RFC peut traduire l'insertion de changement dans la zone à l'aide du DNS dynamique.

D'ailleurs, ce travail propose une alternative à l'architecture conventionnelle de l'infrastructure du DNS, c'est donc dire que nous avons déployé ce type d'architecture en plus de l'opérer, afin de constater les problématiques, qu'elle occasionne. Il est donc nécessaire de comprendre cette architecture, afin de mieux évaluer les gains de l'architecture proposée.

L'architecture la plus répandue chez les opérateurs de serveurs DNS est l'architecture Maître-Esclave [2]. Elle fonctionne de sorte que l'administrateur du serveur DNS doit simplement mettre à jour le serveur maître et les changements à la zone seront répliqués aux esclaves. La mise à jour est effectuée à travers un échange de communication entre le maître et ses esclaves. Un protocole d'envoi de message appelé « DNS Notify » [6] et de transfert des informations de la zone appelée AXFR [28] sont, alors mis à exécution par le maître vers ses esclaves.

La figure [2] suivante démontre l'architecture conventionnelle et l'interaction entre le maître et ses esclaves.

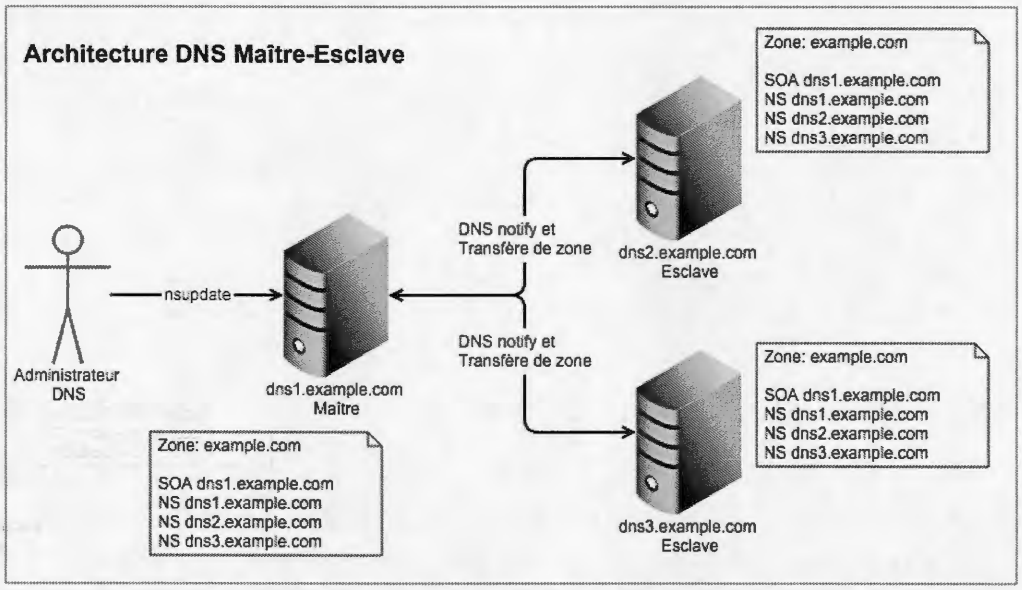


Figure 3 : Architecture Maître-Esclave la plus répandue.

Il existe plusieurs dérivés de cette architecture par exemple, il existe une version où l'on protège le serveur maître derrière le pare-feu de l'entreprise, afin de sécuriser ce dernier des attaques possibles, mais il reste que le serveur maître est une pièce critique de l'architecture Maître-Esclave et qu'il ne peut pas tomber en panne, qu'il soit protégé ou non.

À vrai dire, le maître est le point unique de défaillance de l'architecture conventionnelle, car sans lui le système ne peut pas être mis à jour. En fait, les esclaves seront en mesure de répondre aux requêtes DNS, qui leur seront acheminées, mais en servant l'information en date de la panne du maître. Cela explique pourquoi il existe des dérivés de cette architecture, puisqu'il faut à tout prix protéger le maître de toutes les possibilités de pannes auxquelles il fait face.

Commençons par détailler l'interaction entre le maître et ces esclaves. Le maître est celui sur lequel l'administrateur effectue les changements à la zone. Il peut y modifier les enregistrements et doit absolument incrémenter le numéro de série de la zone sinon les esclaves ne remarqueront pas la mise à jour de cette dernière. Lorsque la mise à jour de la zone est effectuée, l'administrateur redémarrera celle-ci et le serveur maître enverra une requête de « DNS Notify » aux esclaves. Lorsque ceux-ci recevront la requête, ils valideront avec le maître que c'est bien lui qui a envoyé cette mise à jour et demanderont au maître l'enregistrement SOA, qui détient le numéro de série de la zone. Si le numéro de série de la zone est bel et bien modifié, à ce moment précis l'esclave demandera un transfert de zone. À la suite de la réception du transfert de zone, l'esclave mettra à jour le fichier de configuration de la zone localement sur son système de fichier. Il pourra ensuite répondre aux requêtes DNS avec les nouvelles informations mises à jour.

Ce cas d'utilisation constitue le standard pour les zones DNS non dynamiques. Lorsqu'il y a une zone dynamique, le flux de communication n'est pratiquement pas modifié, à l'exception près

que le serveur maître utilise l'utilitaire « nsupdate », afin de mettre à jour l'enregistrement déjà présent dans la zone, pour ensuite alerter les esclaves du changement. L'utilitaire « nsupdate » a donc pour rôle la mise à jour des enregistrements, tels que « www.example.com » dans la zone « example.com ». Dans la prochaine section, nous aborderons les problématiques reliées à ces utilitaires, plus particulièrement à l'architecture Maître-Esclave décrite dans la section ci-présente.

Les problématiques de l'architecture conventionnelle

Premièrement, débutons avec le cas de panne le plus simple, c'est-à-dire la panne matérielle ou réseautique reliée au maître. Cette simple panne entraîne à elle seule, l'interruption totale des mises à jour du système de DNS. Tant et aussi longtemps que le bris d'opération ne sera pas réparé, il sera impossible pour quiconque d'ajouter, modifier ou supprimer un enregistrement dans la zone DNS. C'est donc dire que l'architecture Maître-Esclave possède un point de défaillance unique. Plusieurs méthodes ont été expérimentées pour régler cette situation comme l'opération d'une architecture avec plusieurs serveurs maîtres [29], que nous synchroniserons à l'aide d'outil tel que « rsync » ou encore « SCP ». La plus grosse problématique découlant de cette architecture survient lors de la mise à jour dynamique des enregistrements, qui est alors impossible parce que la synchronisation des fichiers peut entraîner la corruption des fichiers de configurations.

La deuxième problématique est reliée aux fichiers de configurations enregistrés sur le système de fichiers du serveur. Plus il y en a, plus il devient complexe de les gérer. D'ailleurs, à chacune des mises à jour, il faut forcer le serveur DNS à recharger les fichiers de configuration, ce qui peut prendre du temps et occasionner des délais dans la réponse de données fraîches. De plus, il faut s'assurer que le fichier est bien formaté et contient chacun des points là, où il le faut, sinon cela entraîne une erreur et la zone ou bien l'enregistrement ne seront plus desservis par le serveur DNS.

Troisièmement, si l'administrateur du DNS n'a pas correctement mis à jour le numéro de série, il peut arriver une désynchronisation DNS entre les esclaves et le maître, alors que le maître connaîtra la nouvelle valeur de l'enregistrement et les esclaves eux répondront par l'ancienne.

Quatrièmement, bien que la latence réseau soit aussi une problématique possible, car elle peut occasionner des délais dans la mise à jour des zones, il existe un problème, qui peut s'avérer extrêmement sournois à détecter. Il s'agit du problème de configuration des enregistrements de type « NS », qui contiennent l'information des adresses IP des serveurs maîtres et esclaves. Ces enregistrements sont utilisés lors de la validation entre les esclaves et le maître, afin de valider que le maître est l'initiateur de la mise à jour et non pas une fausse requête pouvant avoir été initiée par un attaquant extérieur pour causer un surplus de trafic sur les serveurs DNS.

Si nous résumons les problèmes de cette architecture, nous y retrouvons les problèmes suivants :

1. Complexité de gestion de la quantité des fichiers de zones.
2. Point unique de défaillance au niveau du maître.
3. Impossibilité d'utiliser le DNS dynamique avec plusieurs serveurs maîtres.
4. Possibilité de problème de synchronisation des zones.
5. Latence réseau occasionnant un délai dans le transfert de zone et sa mise à jour.
6. Mécanisme de validation des échanges pouvant être mal configuré et qui est inclus dans le fichier de zone.
7. Fichier de zone qui est extrêmement sensible et demande une minutie dans son édition.

En résumé, l'architecture conventionnelle Maître-Esclave est beaucoup mieux adaptée à l'opération d'infrastructure pour les zones statiques et très peu pour celles qui sont dynamiques. Elle contient plusieurs problèmes, que nous devons considérer avant de la mettre en opération, telle que l'accessibilité à l'API et aux serveurs DNS par cette dernière, ainsi que la protection en cas de panne du serveur maître, afin de conserver l'accessibilité aux modifications de la zone.

Le principe de base du DNS Dynamique (DDNS)

À la base de ce projet, il fallait mettre en œuvre une solution simple de DDNS et comprendre son mécanisme pour pallier la problématique des adresses IP dynamique. Il est par contre important de comprendre quel est le mécanisme de mise à jour d'une solution de DDNS et aussi comprendre comment cette dernière s'intègre avec le système de DNS hébergé à l'aide de l'architecture conventionnelle. Cette section décrira le mécanisme de base de mise à jour d'un enregistrement DNS dynamique en utilisant l'architecture conventionnelle.

Si nous considérons la forme la plus simple d'une solution de DNS dynamique, elle sera constituée d'un seul serveur DNS autoritaire. Si nous précisons un peu plus, il s'agit principalement d'installer plusieurs serveurs DNS de type autoritaire [7], tel que décrit dans la section ci-haut décrivant l'architecture conventionnelle d'une infrastructure DNS. Puis, il suffit d'utiliser le mécanisme de mise à jour décrit dans les RFC2136, afin de bénéficier de la mise à jour dynamique d'un enregistrement de nom de domaine. Ce mécanisme passe par l'utilisation de l'outil « nsupdate » sur le serveur DNS maître.

Nous utiliserons le domaine example.com [8] à titre d'exemple dans ce document.

Le flux de traitement de la mise à jour d'un enregistrement DNS dynamique est décrit dans la figure suivante, nous expliquerons cette dernière par la suite.

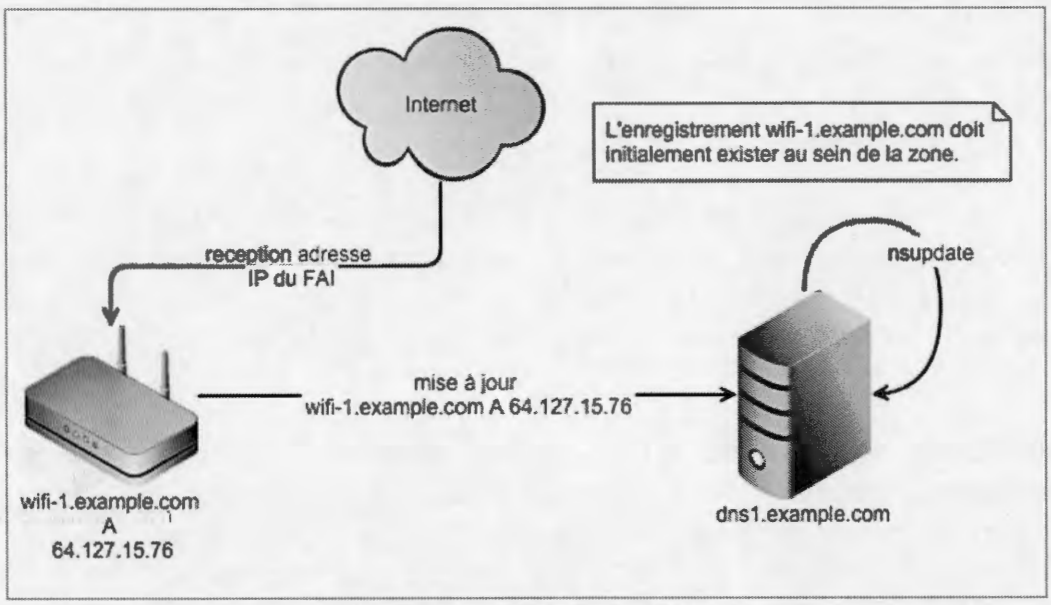


Figure 4 : Le flux de traitement DDNS.

La figure 4 présente l'exemple, où nous hébergeons le domaine *example.com* sur notre serveur DNS, en plus d'y avoir créé l'enregistrement *wifi-1.example.com*. Les étapes suivantes décrivent le processus de la figure 4 :

1. L'appareil sans-fil sera nommé *wifi-1.example.com.*
2. Le FAI assignera l'adresse IP dynamique suivante : 64.127.15.76.
3. L'appareil sans-fil enverra au serveur DNS une requête de mise à jour.
4. Le serveur DNS maître effectuera la mise à jour.

À la suite de ces étapes, l'enregistrement *wifi-1.example.com* de la zone DNS *example.com* est mis à jour. L'enregistrement DNS contiendra désormais l'adresse IP dynamique suivante: 64.127.15.76.

wifi-1.example.com A 64.127.15.76

D'ailleurs à titre informatif, l'enregistrement *wifi-1.example.com* possédera un TTL de 300 secondes. Le TTL représente le mécanisme de mise en cache par les serveurs DNS récursifs [7].

Lorsque le serveur maître reçoit la demande de changement, il utilisera l'outil « nsupdate » de la façon suivante [29] :

```
nsupdate -v -k /etc/named.d/admin-updater.key
> update delete wifi-1.example.com. A
> update add wifi-1.example.com. 300 A 64.127.15.76
> send
```

Figure 5 : Ligne de commande « nsupdate » pour mettre à jour un enregistrement DNS.

À la suite des lignes de commande exécutées dans la figure 5, l'enregistrement DNS *wifi-1.example.com* pointera désormais sur l'adresse IP 64.127.15.76 pour quiconque demandera l'adresse de cet enregistrement à partir d'un serveur de DNS récursif.

Dans la figure 5, nous donnons un exemple de mise à jour d'un enregistrement et il s'agit d'un cas simple, lequel pourrait être exécuté par l'administrateur sur le serveur maître. Par contre, nous nous devons d'automatiser cette tâche et de nous assurer que l'équipement réseautique est en mesure de mettre lui-même à jour son adresse IP, sans l'intervention d'un humain, auprès du serveur DNS maître.

Afin de réaliser cette automatisation, il y a plusieurs approches possibles, mais celle qui semble faire l'unanimité est celle de l'utilisation d'un API Web. Il est en réalité possible d'écrire un petit API servi par un serveur HTTP, qui lorsqu'il est contacté, lancerait l'exécution d'un script, qui exécuterait séquentiellement le code de l'outil « nsupdate » démontré dans la figure 5. Cette API devrait normalement être accessible depuis Internet, afin que tous les équipements réseautiques puissent le contacter à la demande et ce peu importe le fournisseur d'accès Internet connectant l'équipement réseautique.

La figure suivante démontre le flux de traitement du DDNS lorsqu'il est automatisé par un API HTTP. Nous expliquerons la figure 6 par la suite.

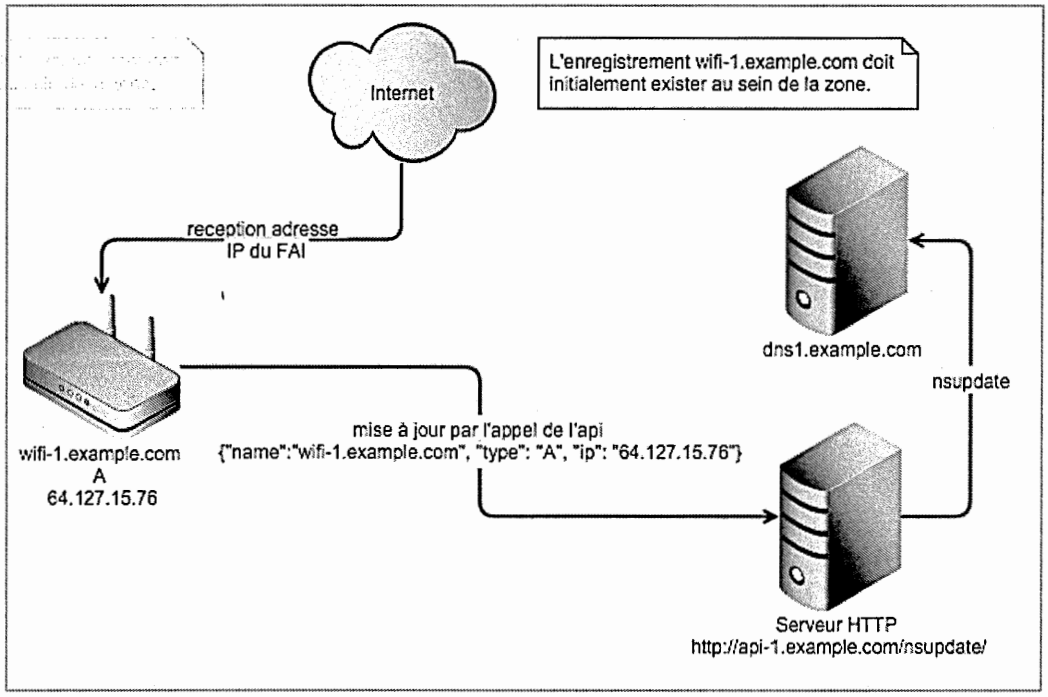


Figure 6 : Mise à jour DDNS par un API HTTP.

Nous pouvons remarquer que l'équipement réseautique devra en réalité faire un appel à l'API Web à l'aide d'un client HTTP exécuté depuis ce dernier et envoyer les informations de

modification de l'enregistrement DNS *wifi-1.example.com*. Lorsque l'API recevra l'information, elle pourra ensuite mettre à jour l'enregistrement à l'aide de l'exécution du script exécutant les commandes « nsupdate ». Il faut aussi penser que le serveur Web hébergeant l'API ne sera pas nécessairement sur le même serveur que le serveur DNS, car nous pourrions héberger l'API et le serveur DNS sur des serveurs différents. C'est pourquoi la figure 6 sépare les deux types de serveurs et cela ajoute à la complexité de l'infrastructure, car l'API doit être en mesure de contacter le serveur DNS maître de l'infrastructure, afin d'y exécuter le script de mise à jour des enregistrements DNS dynamiques.

La figure suivante démontre le cycle complet de la mise à jour d'un enregistrement DNS dynamique lorsque nous sommes en présence d'une infrastructure DNS réalisée à l'aide de l'architecture conventionnelle.

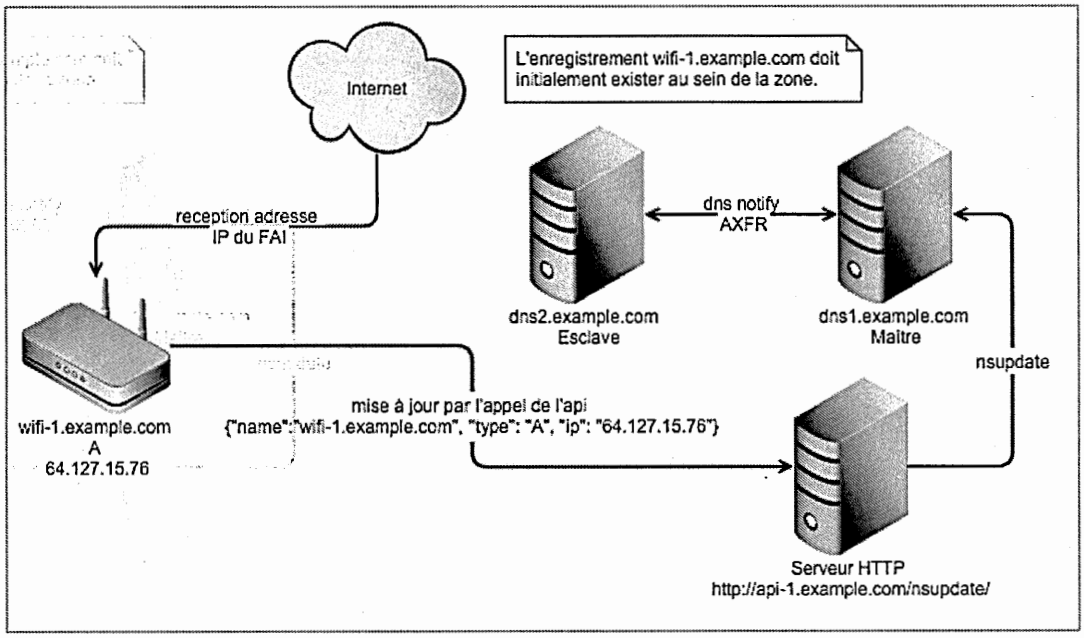


Figure 7 : Mise à jour DDNS sur une architecture DNS conventionnelle.

Nous pouvons remarquer dans la figure 7 que cette architecture comporte 3 cas possibles, où la latence du réseau pourrait ralentir la mise à jour de l'enregistrement DNS. Il est possible que le serveur maître et le serveur esclave ne soient pas nécessairement dans le même centre de données or, il pourrait arriver que la mise à jour entre les serveurs DNS soit lente ou encore qu'elle occasionne une incertitude dans la zone sur une courte période de temps, où les deux serveurs ne servent pas la même information due au temps de réception de la mise à jour. D'ailleurs, non seulement il y a possibilité d'un risque de délais de mise à jour entre le maître et l'esclave, mais il existe un risque de délais entre l'équipement réseautique et le serveur de l'API, puis un autre entre le serveur de l'API et le serveur DNS maître. Ces délais de mise à jour expliquent pourquoi le standard propose un délai de 300 secondes de rétention en cache des serveurs DNS récursif lors d'une mise à jour d'un enregistrement DNS dynamique. Dans les faits, cela permet de prévoir, que la mise à jour de l'adresse IP puisse prendre jusqu'à cinq minutes de réplication à travers l'infrastructure entière.

Dans cette section, nous avons expliqué le mécanisme de mise à jour d'un enregistrement DNS dynamique sur une infrastructure DNS conventionnelle, tout en expliquant les effets de cette architecture sur la mise à jour de l'enregistrement. Nous avons aussi introduit certains composants de l'architecture d'une solution de DNS dynamique telle que l'API Web, qui permet la réception et l'exécution automatisée des mises à jour des enregistrements.

La surveillance du réseau et le SLA

Nous discuterons brièvement de l'effet de la surveillance du réseau sur notre projet d'architecture DNS Dynamique, car celle-ci a eu un impact énorme sur l'architecture proposée. Débutons par la figure 8 qui démontre le schéma des flux de communication d'un système de surveillance du réseau dans un contexte d'adressage IP dynamique.

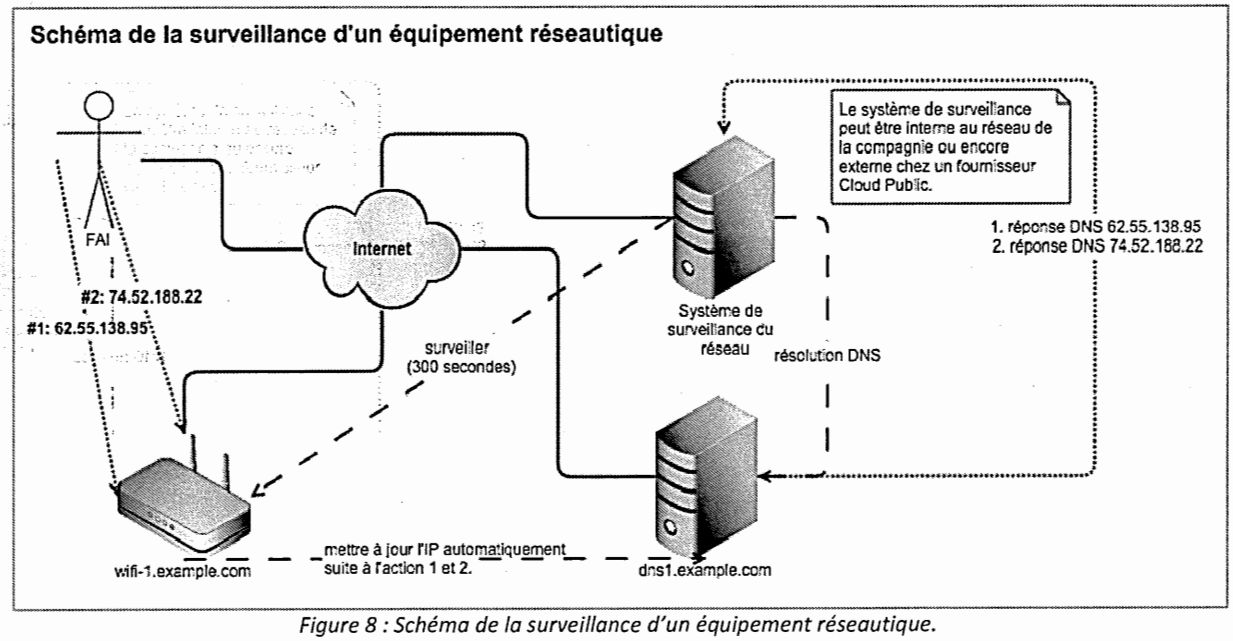


Figure 8 : Schéma de la surveillance d'un équipement réseautique.

Pourquoi cette dernière impacte-t-elle notre projet ?

Parce que le besoin derrière la détection rapide des changements d'adresse IP dynamique provient des systèmes de surveillance du réseau. Il faut se rappeler qu'à la base du projet le besoin d'affaires était de surveiller des équipements sur le réseau Internet, qui reçoivent de leur fournisseur d'accès Internet une adresse IP dynamique, comme démontré ci-haut dans la figure 8 à l'étape 1 et 2.

Les systèmes de surveillance du réseau font l'agrégat des statistiques des équipements surveillés toutes les trois cents secondes c'est-à-dire toutes les cinq minutes et ils collecteront dans un fichier « Round Robin Database » aussi nommé « rrd » la moyenne de ces statistiques. Nous

présentons dans la figure 9, un graphe « rrd » de la vitesse entrante et sortante des paquets UDP sur notre serveur DNS autoritaire *ns1.example.com* à titre d'exemple.

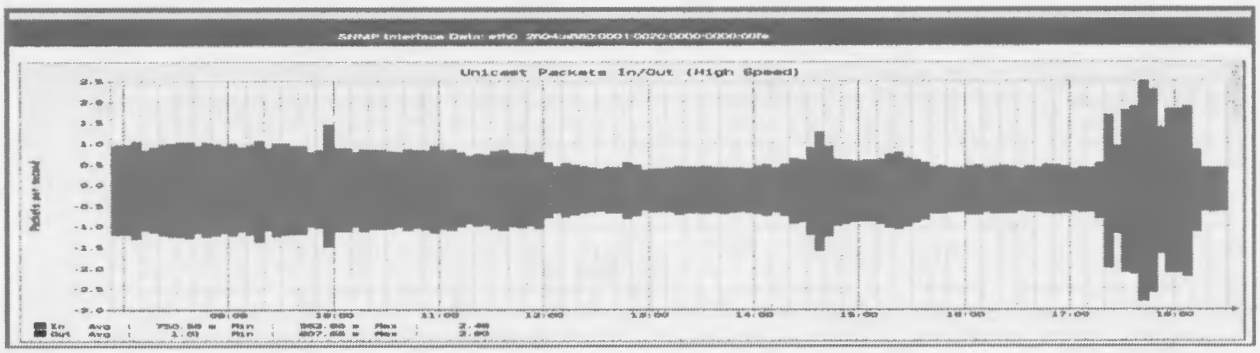


Figure 9 : Graphe RRD des paquets UDP sur le serveur DNS autoritaire.

Ainsi, afin que le système de surveillance puisse rapporter toutes les cinq minutes les statistiques d'opération du système distant, il doit être en mesure de contacter le serveur distant, sinon le graphe « rrd » comportera des données manquantes du même ordre que lorsqu'une panne survient. La figure 10 démontre ce type de données manquantes dans un fichier « rrd », dans notre exemple, c'est le processus SNMP, qui est tombé en panne pour une période de plus ou moins vingt minutes.

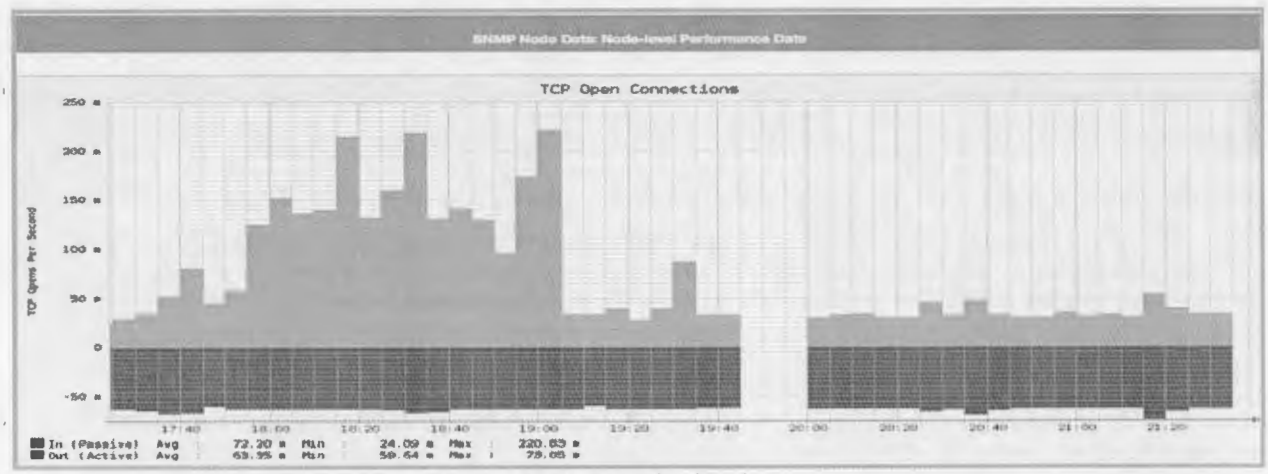


Figure 10 : Graphe RRD avec des données manquantes.

Il est facile de collecter les données telles que dans notre graphe de la figure 9, car l'IP de notre serveur DNS autoritaire ne changera pas puisqu'elle est statique. Par contre, lorsque vous devez surveiller des équipements dont l'adresse IP est dynamique, vous devez vous assurer, afin de ne pas créer de faux positif, que l'équipement est disponible à l'intérieur de la plage des trois cents secondes. De plus, dans un contexte de surveillance d'IP dynamique, si nous nous retrouvons à tout moment avec une mauvaise adresse IP, qui n'a pas encore reçu sa mise à jour, les fichiers « rrd » du système de surveillance n'auront plus de valeur. Les fichiers « rrd » seront troués de données manquantes telles que dans la figure 10. D'ailleurs, nous perdrions en plus la

fonctionnalité de statistiques depuis ces mêmes fichiers, alors que les systèmes de surveillance se servent de ces données pour créer des alertes basées sur des seuils ou encore par exemple lorsque nous voulons déclencher une alerte si l'unité centrale de traitement d'un serveur dépasse les 80% de charge pendant plus de dix minutes.

L'enjeu pour notre architecture de DNS dynamique était donc d'être en mesure de rapporter les changements d'adresse IP dynamique en dessous de la barre des trois secondes et tout cela dans le but de connaître l'adresse IP dynamique de l'équipement lorsque le système de surveillance du réseau démarre son travail d'agrégation des données depuis l'équipement distant. Il faut donc s'assurer que ce dernier puisse remarquer le changement d'adresse IP survenu sur l'équipement distant. Ainsi, si nous arrivons en dessous de la barre des trois secondes, il y a plus de chance que nous évitions un faux positif, tel que la fausse détection d'une panne d'équipement, qui en fait n'est pas en panne, mais dont l'adresse IP a simplement changé dans le temps, en plus d'éviter des données manquantes dans les fichiers « rrd » tels que vus dans la figure 10.

D'ailleurs, plus vite la détection surviendra, moins le système de surveillance fera de la revérification de l'état du système distant, car plusieurs systèmes de surveillance réseau afin de diminuer les faux positifs vérifient une deuxième fois après trente secondes l'état d'un équipement distant pour s'assurer qu'il s'agit bien d'une panne et qu'il rapporte adéquatement le statut de l'équipement distant.

En résumé, l'impact du système de surveillance du réseau est dû au fait que nous surveillions des équipements réseau dont l'adressage IP était dynamique et que nous devions trouver la nouvelle adresse IP avant même que le prochain horaire de vérification de l'équipement réseau soit exécutée par ce dernier.

Impact sur le DNS dynamique (DDNS)

Malgré l'enjeu de détection du changement d'adresse IP dynamique en dessous des trois cents secondes pour le système de surveillance, nous faisions face à un standard dans l'opération des serveurs DNS autoritaires de mise à jour des adresses IP pour les enregistrements dynamiques de trois cents secondes. La plupart des grands joueurs de l'industrie de l'hébergement DNS respectent le standard du TTL de trois cents secondes pour un enregistrement DNS dynamique. Ainsi, les serveurs DNS récursifs placeront en cache l'enregistrement dynamique pour une durée de cinq minutes.

Nous avons évoqué dans la section de l'architecture conventionnelle, que le délai entre la mise à jour et les serveurs esclaves pouvait prendre jusqu'à cinq minutes, afin de se synchroniser. Il y a aussi historiquement une autre raison pour ces trois cents secondes. Il s'agit de diminuer la pression de la charge sur les serveurs DNS autoritaires et de mettre en cache l'enregistrement dynamique le plus longtemps possible sur les serveurs DNS récursifs qui nécessitent peu de gestion, contrairement au serveur autoritaire.

D'ailleurs, cela signifie que les administrateurs de serveurs DNS ont accepté que les enregistrements dynamiques soient modifiés moins souvent. Quitte à perdre quelques secondes la connexion avec l'équipement distant, versus une augmentation de la charge de travail de l'infrastructure DNS autoritaire. Cela donnait l'impression de protéger l'infrastructure d'une charge trop grande si tous les enregistrements dynamiques devaient être mis à jour rapidement.

En résumé, le délai de trois cents secondes étant devenu un standard par les années et l'exploitation de l'infrastructure DNS par les grands fournisseurs de DNS. Nous voulions améliorer ce délai en le réduisant le plus possible, pour ainsi répondre à notre problématique de surveillance des équipements réseautiques initiale. Si nous arrivions à réduire le délai de la mise à jour d'un enregistrement DNS dynamique, nous arrivions à améliorer notre surveillance des équipements en nous assurant d'éviter les faux positifs ou la perte de données de surveillance.

L'architecture proposée

Dans ce chapitre, nous expliquerons l'architecture que nous proposons à la suite du projet de recherche, que nous avons entrepris. Nous voulions être en mesure d'héberger une infrastructure DNS, qui supporterait la mise à jour des enregistrements dynamiques en temps réel.

Dans la figure 11 à la page suivante, vous trouverez l'architecture simplifiée de la plateforme de DNS que nous proposons. C'est-à-dire que nous avons supprimé de cette dernière les systèmes, qui servent à la gestion de la plateforme et qui ne font pas nécessairement partie de la recherche et du développement de ce mémoire. Les éléments manquants sont les serveurs suivants :

1. Git : notre gestionnaire de code source.
2. Jenkins [50] : notre serveur d'intégration continue.
3. Uptime Robot [31] : le système de surveillance de l'infrastructure.
4. status.example.com : une page Web hébergée à l'externe de nos serveurs pour rapporter l'état de nos services.
5. SonarQube [30] : la plateforme de gestion de la qualité du code source.

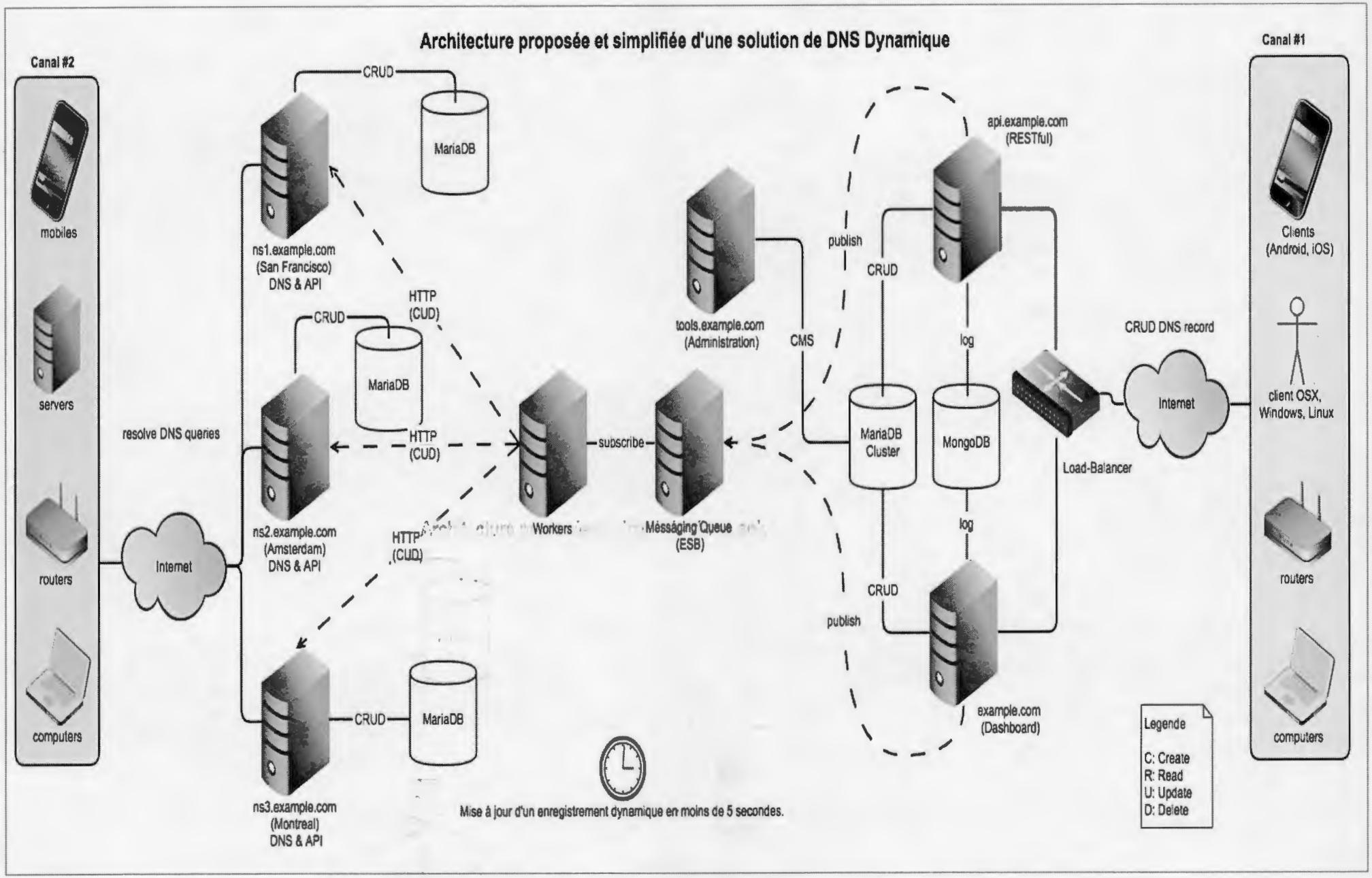
Ces éléments vous seront présentés dans la globalité de l'architecture à la figure numéro 23 à la fin de ce chapitre.

Nous énumérerons rapidement les blocs technologiques de l'architecture proposée dans la liste suivante et nous les expliquerons un à un tout au long du chapitre.

- MariaBD : le système de gestion de base de données.
- Les serveurs HTTP/Web : le serveur Nginx est utilisé pour chacun des serveurs Web.
- MongoDB : le serveur NoSQL de gestion de documents.
- PowerDNS : le serveur DNS utilisé avec le plug-in MariaDB/MySQL.
- RabbitMQ : un système de gestion de file d'attente de messages.

- Les abonnés « workers » : petit logiciel exécutant certaines tâches prédéfinies.
- Python et Django : le langage Python et le cadre d'application Django sont utilisés pour l'écriture des logiciels.

Les blocs de construction décrits ci-haut et l'architecture proposée sont présentés dans la figure 11 qui suit.


Architecture proposée et simplifiée d'une solution de DNS Dynamique
Canal #2
mobiles
servers
routers
computers
Canal #1
Clients
(Android, iOS)
client OSX,
Windows, Linux
routers
computers
CRUD
MariaDB
ns1.example.com
(San Francisco)
DNS & API
HTTP
(CUD)
ns2.example.com
(Amsterdam)
DNS & API
ns3.example.com
(Montreal)
DNS & API
resolve DNS queries
Internet
Workers
subscribe
Messaging Queue
(ESB)
tools.example.com
(Administration)
CMS
api.example.com
(RESTful)
publish
log
MariaDB
Cluster
MongoDB
Load-Balancer
CRUD DNS record
Internet
example.com
(Dashboard)
Mise à jour d'un enregistrement dynamique en moins de 5 secondes.
Legende
C: Create
R: Read
U: Update
D: Delete

Canaux de communications

Débutons par l'explication des canaux de communications sur l'architecture proposée. Le système possède deux canaux, qui sont représentés par les deux connexions au réseau Internet. Le premier canal de communication est celui par lequel une personne peut gérer les enregistrements DNS sur la plateforme. Ce canal est donc vital à la plateforme, car sans ce canal, nous ne pourrions pas créer, mettre à jour ou supprimer un enregistrement DNS. Il faut aussi comprendre que ce canal de communication permet deux types de communications, celle par le tableau de bord, auxquels un humain accédera et effectuera les modifications nécessaires à ces enregistrements DNS. Ainsi que le deuxième, qui consiste en l'accès à un API permettant aux appareils informatiques d'effectuer certaines opérations sur la plateforme.

L'API est généralement utilisé par nos clients logiciels, tel que notre client Android, un pare-feu à la maison ou à la ligne de commande. Ces clients permettent l'automatisation de la plateforme, en plus de permettre la mise à jour automatique des enregistrements dynamiques.

Le deuxième canal d'entrées est celui de la consommation des enregistrements DNS. Cette consommation prend toujours la forme d'une requête DNS émise depuis un serveur DNS récursif. Si nous voulions vérifier la disponibilité d'un équipement réseau tel que vu à la section « La surveillance réseau et le SLA », nous pourrions émettre un paquet ICMP, qui se traduirait par un « ping » sur l'adresse *wifi-1.example.com*. Ce qui est important à retenir de cet exemple, c'est que le logiciel « ping » exécute premièrement une demande de résolution DNS telle que vue à la figure 2, afin de trouver l'adresse IP. Puis lorsque celle-ci est complétée, il envoie le paquet à l'équipement désigné par le nom de domaine *wifi-1.example.com*. Il peut donc exister une multitude de scénarios par lesquels ce canal de communication sera utilisé, allant du simple usage d'un navigateur Internet à des applications réseau, jusqu'à l'hébergement d'une page Web. Même l'infonuagique peut utiliser ce type d'enregistrement, afin de nommer les machines virtuelles auxquelles est assignée une adresse IP dynamique.

Ces deux canaux de communications constituent les entrées publiques de la plateforme DNS. Nous devons rappeler qu'à la base une infrastructure de serveurs DNS constitue un système d'application réseau et qu'il est important que le réseau ne soit jamais engorgé par les paquets, qui y sont acheminés. La compréhension de ces canaux est importante, car c'est de là que le trafic réseau provient et c'est aussi cette partie de l'architecture, qui devra bénéficier des qualités logicielles suivantes : la fiabilité et la mise à l'échelle.

Dans un cas comme dans l'autre, nos canaux de communications constituent les blocs de constructions les plus sensibles et ceux qui risquent le plus de changer au fil du temps par l'augmentation des ressources physiques ou virtuelles, qui leur seront assignées. Afin de parler d'architecture neutre, il est donc important que ces blocs de constructions puissent être mis à l'échelle sans avoir à refaire notre architecture toutes les fois que nous subirons une augmentation du trafic réseau devant être traité. C'est pourquoi nous avons pris soin de bien penser à l'opération de nos canaux de communication, afin que nous puissions grandir l'infrastructure sans refaire l'architecture dès que le trafic ou le nombre d'utilisateurs augmente.

Pour nous aider à gérer l'augmentation du nombre d'utilisateurs et possiblement la mise à l'échelle de notre architecture, nous avons créé un système d'inventaire en base de données, afin de pouvoir modifier nos canaux de communication, sans avoir à changer leurs configurations. Ainsi, nous pourrons changer les serveurs pour en mettre de plus puissants ou simplement augmenter le nombre de ces derniers, tout simplement en ajoutant dans la base de données une configuration à cet effet. Jusqu'à maintenant, nous avons réussi ce pari, car nous avons mis en place de nouveaux systèmes en Europe et dans l'ouest des États-Unis sans avoir à modifier l'architecture.

Les blocs de constructions de l'architecture

PowerDNS

Nous avons choisi le logiciel de serveur de noms de domaine PowerDNS, parce qu'il nous permettait d'utiliser un système de gestion de base de données, afin de conserver les informations des fichiers de zones DNS. Cet avantage permettait de supprimer la complexité de la gestion des fichiers de zones décrite dans la section de l'architecture conventionnelle d'une infrastructure DNS. Nous rappellerons que chaque fichier de zones est un fichier local sur le système de fichiers du serveur hébergeant le logiciel de nom de domaine. Ceci pouvait éventuellement mener à la création de milliers de fichiers de configuration de zones sur un serveur. La fonctionnalité du logiciel PowerDNS permettant l'utilisation d'un SGBD, fais en sorte que le serveur DNS lit les fichiers de configuration des zones depuis deux tables de la base de données, soit la table des domaines « domains » et celle des enregistrements « records ». Cette simplification de l'hébergement des fichiers de configuration des zones était pour nous un élément essentiel à l'opération de l'infrastructure DNS.

D'ailleurs, il y a un avantage à pouvoir accéder aux fichiers de configuration des zones par un SGBD et c'est celui de pouvoir écrire du code applicatif permettant l'interaction avec la base de données.

De plus, le serveur PowerDNS selon l'étude réalisée par le SANOG [17] démontre qu'il est aussi performant que le serveur Bind, qui est traditionnellement utilisé. Donc ces qualités de performance jumelée à l'utilisation du SGBD en faisaient le choix idéal comme bloc de construction de notre architecture.

MariaDB

Le serveur de base de données possède deux rôles dans l'architecture. Son premier rôle, qui est le plus important, consiste à conserver les zones, la configuration, l'inventaire, les permissions, et les usagers de l'infrastructure DNS. Dans l'architecture, vous pourrez remarquer le premier serveur nommé « MariaDB Cluster » près du premier canal de communication. Cette grappe de serveurs permet d'héberger la base de données centrale, qui contient toutes les informations

vitales de la plateforme. Cette base de données est utile pour centraliser toutes les informations devant transiger sur l'infrastructure. Elle nous permet aussi de conserver une copie de toutes les zones pouvant être hébergées sur les serveurs DNS distants, ainsi en cas de panne ou d'ajout de serveurs DNS, nous pouvons nous fier aux données conservées dans cette base de données centrale pour mettre en œuvre rapidement un serveur ou tout simplement récupérer ce dernier. Elle agit donc à titre d'enregistrement de sauvegarde des données utilisées sur l'infrastructure DNS.

D'ailleurs, afin de sécuriser au maximum les données conservées dans cette grappe de serveur de base de données, nous avons utilisé la technologie nommée « Galera Cluster », qui a été implémentée dans le SGBD. Cette technologie nous permet d'héberger le SGBD selon une architecture Maître-Maître, qui nous permet de pallier toutes les pannes ou pertes de données éventuelles. Cela assure aussi la mise à l'échelle de notre système de SGBD. Si le nombre d'écriture ou de lecture venait à augmenter, dû à un plus grand volume de trafic réseau, il nous suffirait d'ajouter plus de serveurs physiques ou virtuels à notre grappe existante, afin d'accroître le nombre d'écritures et de lectures pouvant être supportées par cette dernière.

Le deuxième rôle de MariaDB est celui de conservation des fichiers de zones sur les serveurs DNS distants. Tel que vous pouvez le remarquer dans la figure 11, il existe une connexion à un serveur MariaDB sur chacun des serveurs DNS de l'architecture. Ce serveur de SGBD est jumelé au serveur PowerDNS, afin de conserver chacune des zones que ce dernier héberge. Nous conserverons les enregistrements dans les deux tables du schéma de la base de données. L'utilisation du SGBD pour les fichiers de configuration sera expliquée à la section sur les abonnés.

En résumé, le système de gestion des données MariaDB constitue un bloc de l'architecture important, car il contient toutes les informations pouvant être utilisées tant par l'infrastructure que par les usagers de la plateforme de DNS.

HTTP

Le rôle du protocole HTTP est essentiel dans notre architecture, il s'agit d'un protocole facile à mettre à l'échelle en plus d'être très facile d'utilisation. Le rôle principal de ce protocole dans notre architecture est un rôle de transmission d'informations. Il nous permet d'écrire des services de communication, qui peuvent être consommés par diverse partie de l'infrastructure tant localement qu'à distance.

De plus, si nous revenons à la figure 6 dans laquelle nous expliquons comment un API Web était utilisé pour mettre à jour les enregistrements DNS automatiquement et bien nous nous devons de conserver cette API dans notre architecture, afin de conserver cette automatisation des mises à jour des enregistrements DNS.
En utilisant un service Web, nous nous assurions que : ce bloc de construction serait flexible, que sa mise à l'échelle ne représenterait pas une tâche complexe et que nous serions capables de l'administrer facilement.

Architecture RESTful

L'architecture contient deux API. La première est disponible publiquement et la deuxième est réservée à l'usage interne de la plateforme. Ces dernières ont été écrites en utilisant le patron de conception logiciel RESTful [14]. Nous nous assurions ainsi d'encadrer l'écriture du code source de nos API dans un patron bien défini et connu, en plus de s'assurer une certaine facilité dans l'écriture de nos clients consommateurs de nos API.

La figure 12 suivante démontre le type d'URL que nous avons générée grâce à ce patron.

POST	http://remote:9119/api/domain/add/
DEL	http://remote:9119/api/domain/delete/
GET	http://remote:9119/api/domain/list/
POST	http://remote:9119/api/domain/update/
POST	http://remote:9119/api/record/add/
DEL	http://remote:9119/api/record/delete/
GET	http://remote:9119/api/record/list/
POST	http://remote:9119/api/record/update/

Figure 12 : URL de l'API REST à distance.

MongoDB

Nous avons fait l'utilisation du serveur d'hébergement de documents JSON MongoDB pour assurer la traçabilité de toutes les opérations pouvant être effectuées sur notre infrastructure DNS. Ce bloc de construction ne constitue pas un bloc essentiel à l'architecture, mais représente une qualité logicielle que nous voulions offrir à nos clients et celle-ci est la traçabilité de toutes les actions possibles sur l'infrastructure. Il est à noter que pour le moment nous traçons toutes les mises à jour des enregistrements DNS pouvant être faits depuis notre tableau de bord, ainsi que celles pouvant être réalisées depuis l'API ou les clients distants. Cela se traduit par la possibilité de consulter les fichiers de journaux depuis notre tableau de bord.

En plus, d'offrir à nos clients un moyen de tracer toutes les modifications aux zones pouvant être apportées dans le temps, nous pouvons aussi nous servir de ces fichiers de journaux, afin de comprendre et apprendre depuis notre infrastructure de DNS. Par exemple, il nous sera possible de savoir quels sont les FAI les plus utilisés sur notre infrastructure ; à quelle fréquence les adresses IP sont-elles mises à jour pour l'ensemble de leurs clients se servant de notre service et bien d'autres données pouvant être extraites des fichiers de journaux.

RabbitMQ

Nous avons utilisé le serveur de fils d'attente de messagerie RabbitMQ parce que ce dernier est extrêmement efficace et rapide. De plus, nous avions déjà de l'expérience professionnelle avec ce dernier. D'ailleurs, il permet d'emblée la mise à l'échelle grâce à sa fonctionnalité de haute

disponibilité, donc il consistait pour le moment, en un bloc de construction fiable, qui ne représentait aucun risque pour nous.

Une autre fonctionnalité intéressante de RabbitMQ est la fédération des files d'attente que nous allons éventuellement évaluer, si le besoin se présente, afin d'augmenter le nombre de messages pouvant être envoyés sur la file d'attente.

Abonnés (Workers)

Les abonnés sont de petits programmes provenant du patron de conception logiciel appelé « Publish and Subscribe » [19]. Ils sont la partie du code qui souscrit aux messages publiés dans la file d'attente. Ces petits programmes se sont avérés être des blocs de constructions extrêmement importants dans notre architecture, car il est possible de leur faire exécuter des tâches de façon asynchrone, en plus d'être très facile de les mettre à l'échelle.

Le premier rôle que ce bloc de construction remplit est celui de mettre à jour, à la suite de modifications effectuées à partir de notre API principale ou de notre tableau de bord, les serveurs DNS distants à l'aide du protocole HTTP et de la file d'attente de messages.

Les abonnées utilisent le protocole HTTP dans toutes leurs communications avec le reste de l'infrastructure. C'est justement pourquoi nous trouvions important de gérer les fichiers de zones à l'intérieur d'un SGBD. Les abonnés écoutent les messages qui sont publiés sur la file d'attente et s'ils arrivent à interpréter le message, ils pourront le consommer et exécuter la tâche, qui y est reliée. En utilisant le SGBD pour conserver nos fichiers de configuration, nous nous sommes aussi permis d'écrire une API, qui est hébergée sur le serveur DNS, où réside le SGBD. Cette API connectée directement sur le SGDB local nous permet de transmettre, par le protocole HTTP et les messages dans la file d'attente, les demandes de modifications sur l'infrastructure DNS.

Nous sommes en mesure d'effectuer plusieurs types de messages grâce à ce patron de « publier et souscrire » tel que :

1. Création, suppression et modification de domaines ou d'enregistrements.
2. Synchronisation des zones.
3. Synchronisation d'un enregistrement demandé par l'utilisateur final.
4. Approvisionnement d'un nouveau serveur.
5. Annulation des changements (gracieuseté de la traçabilité).

Il nous suffit de publier un message sur la file d'attente, puis d'attendre qu'un des abonnés capables d'interpréter le message accomplisse le travail demandé.

La figure 13 démontre le cas d'utilisation des abonnés dans le contexte de notre infrastructure DNS.

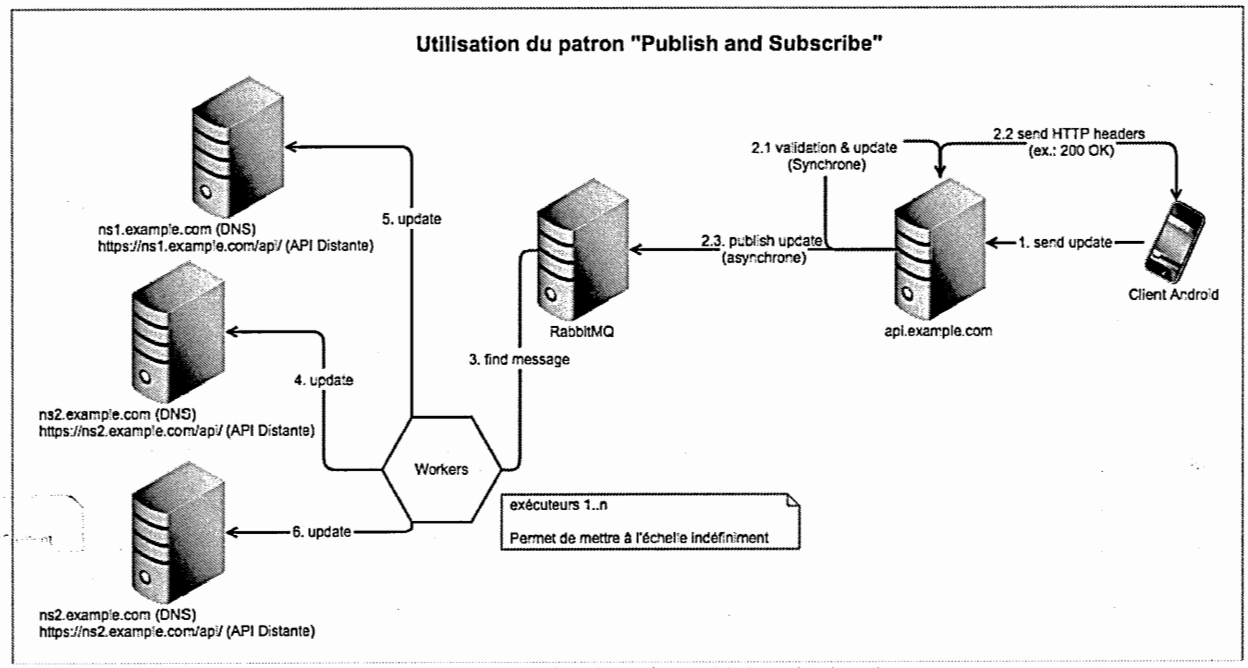


Figure 13 : Utilisation des abonnés « Publish and Subscribe ».

Être asynchrone pour le message 2.3 de la figure 13 permet de répondre très rapidement à nos clients ou notre tableau de bord le résultat de l'opération, qu'il ait été un succès ou un échec. En plus, de permettre le retour rapide sur l'opération et une meilleure qualité de l'interface d'utilisation. Cela nous permet donc de continuer le travail de mise à jour sur les serveurs DNS, car l'exécution peut être plus longue ou bloquante lors de l'exécution sur les serveurs distants. Il faut se rappeler que les serveurs DNS peuvent être situés à travers différents centres de données, voir même différents pays. Cela empêche aussi que notre interface utilisatrice soit extrêmement lente et désagréable d'utilisation, dans l'éventualité où nous attendrions que toutes les opérations de la mise à jour soient terminées pour en servir le statut à l'usager.

Chaque abonné consomme un message à la fois, ainsi en ayant un nombre égal à trois ou plus grand que le nombre de serveurs DNS, dans notre cas nous en avons quatre pour le moment, nous nous assurons une certaine rapidité d'exécution des mises à jour sur les serveurs DNS. Cela permet de traiter les mises à jour DNS sur les serveurs distants de façon parallèle au lieu de les traiter séquentiellement. Nous nous assurons ainsi de réduire le délai de synchronisation de tous nos serveurs à la suite d'une mise à jour.

D'ailleurs, si nous voulons augmenter la quantité de mise à jour pouvant être supportée sur notre infrastructure DNS, il suffit d'ajouter des abonnés, qui sont de très petits programmes pouvant s'exécuter partout, tant sur une machine virtuelle que sur des serveurs physiques. Ainsi nous nous assurons de ne jamais avoir de temps d'attente de traitement de nos messages de mise à jour. De plus, dans l'éventualité d'un succès à l'internationale, nous pourrions aussi permettre la géolocalisation des messages dans la file d'attente des abonnés consommant les messages. Par exemple, si notre plateforme est très utilisée en Europe nous pourrions décider d'ajouter des abonnés sur des serveurs en Europe afin que la latence réseau entre les abonnés et les serveurs

DNS soit réduite à un minimum de délai. En ce moment, tous nos abonnés sont situés à Montréal ou aux États-Unis et la latence avec notre serveur à Amsterdam est d'environ quatre-vingt-dix millisecondes, ce qui ne représente pas encore une problématique pour justifier l'hébergement d'abonnés en Europe.

En résumé, les abonnés sont un bloc de construction, qui s'est avéré plus qu'efficace et extrêmement précieux dans l'accomplissement de tâches en différé sur notre infrastructure DNS.

Les composants applicatifs Web

Le tableau de bord

Le tableau de bord est l'outil de gestion de l'infrastructure DNS dédié aux humains.

Cette application Web est écrite à l'aide du langage de programmation Python et du cadre d'applications Django. Ce tableau de bord nous permet en outre de gérer les permissions sur les noms de domaines et les enregistrements, ainsi que de gérer les utilisateurs du système. Elle nous permet aussi de gérer les quotas d'utilisation.

Il s'agit donc d'une application Web traditionnelle dans laquelle nous avons inclus le contexte et le domaine d'affaires de l'opération de serveurs DNS.

Notre tableau de bord est bien sûr connecté sur notre base de données centrale, en plus de maintenir une connexion sur notre file d'attente de messages, afin d'y publier les messages à la suite des modifications effectuées depuis ce dernier.

La figure 14 à la page suivante donne une vue d'ensemble du tableau de bord. Lors de la conception du tableau de bord, nous avions un but très important et c'était celui de rendre le concept de création de zone ou d'enregistrements DNS le plus simple qu'il soit possible de faire.

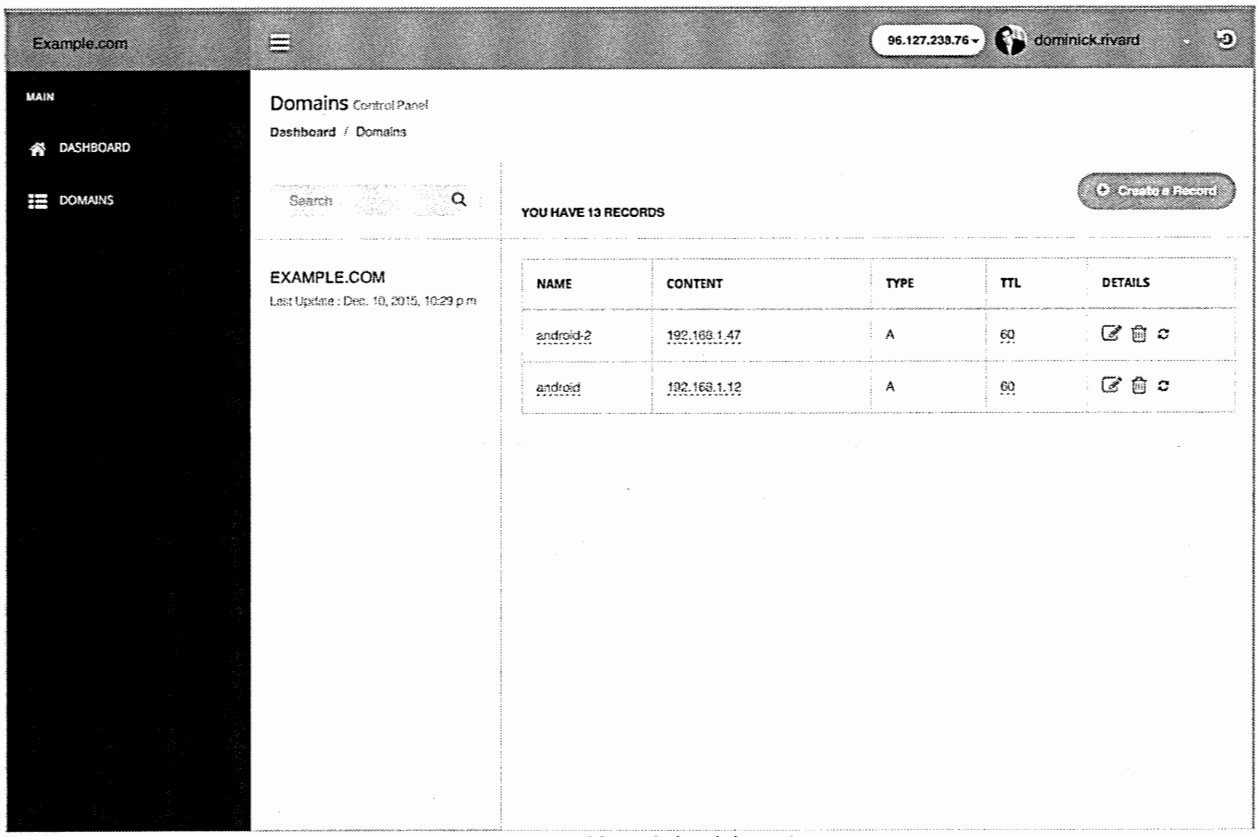


Figure 14 : Le tableau de bord du projet.

Le tableau de bord est donc notre premier contact avec nos usagers et la façon la plus simple à créer un premier enregistrement DNS, qui sera répliqué en quelques secondes sur notre infrastructure DNS.

L'API principale

L'API consiste à aider les applications tierces à pouvoir utiliser notre infrastructure DNS sans nécessité d'action humaine. L'API permet les actions de bases présentes depuis le tableau de bord, tel que l'authentification de l'usager et la mise à jour de l'adresse IP. La figure 15 suivante présente notre client Android, lequel utilise notre API pour effectuer les mises à jour des enregistrements DNS.

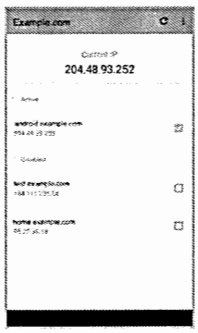


Figure 15 : Le client Android de la plateforme.

Le projet de l'API est lui aussi écrit à l'aide du langage de programmation Python, du cadre d'applications Django et du plug-in Django-Rest-Framework. Nous avons choisi d'écrire tous nos projets d'applications Web en limitant le nombre de langages de programmation utilisés.

Le CMS

Le CMS permet de modifier les données sur l'infrastructure, telle que l'inventaire des serveurs, ainsi que tous les modèles de données que nous avons écrites dans notre application Web. À vrai dire, le CMS est la partie que nous n'avons pas écrite, mais dont nous avons réutilisé la fonctionnalité du cadre d'applications Django, qui propose une solution appelée Django-Admin, qui lorsqu'elle est activée et configurée, nous donne accès à une interface CRUD [36] de nos modèles de données conservées dans le SGBD.

La figure 16 présente la partie d'édition de nos données et cet outil s'est avéré très utile lorsque nous devons modifier les données dans la base de données sans passer par le langage SQL.

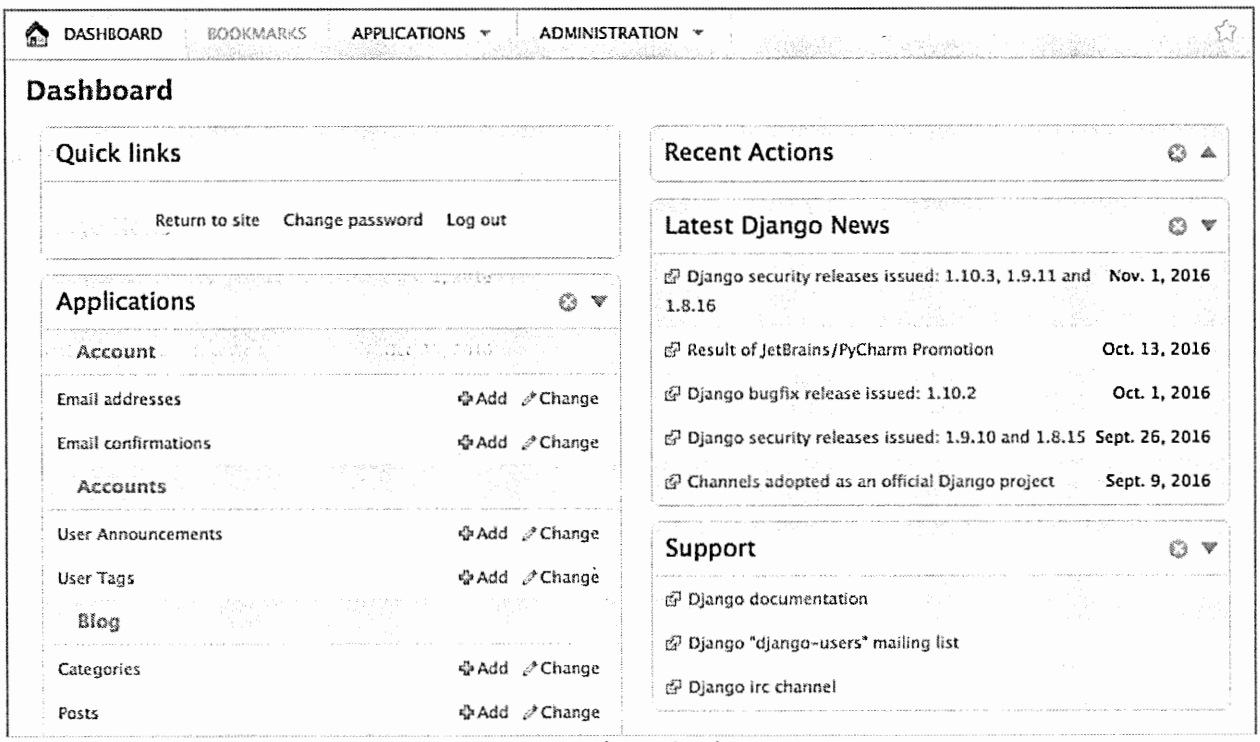


Figure 16 : Présentation du CMS.

L'API distant

L'API distant est une copie simplifiée de notre API publique. Elle permet à nos abonnés d'effectuer les demandes de modifications sur nos serveurs DNS en utilisant le protocole HTTP. Lorsque l'API reçoit la requête HTTP, par la suite le code source Python exécute la mise à jour dans le SGBD local du serveur DNS ayant été contacté par l'abonné. Cette API est sécurisée par

un mécanisme d'authentification HTTP et par l'utilisation du chiffrement SSL. Seuls les abonnés connaissent la clé pour contacter ces API.

Afin de faire un léger retour sur nos blocs de construction, nous pouvons énumérer ceux-ci comme étant une suite de logiciel et de serveurs devant être hébergés, afin de créer le résultat final, qu'est l'architecture d'une solution complète de DNS dynamique. Cette liste de blocs comprend : le tableau de bord, le SGBD, l'API, l'API distant, les serveurs DNS, les abonnés, un serveur de file d'attente de messages, un outil de CMS et une base de données de documents JSON.

Les qualités logicielles

Description de la fiabilité de l'architecture

Le choix de nos blocs de construction pour notre architecture de DNS dynamique a été fait en suivant un critère de robustesse. Il fallait que nous puissions trouver facilement une manière de rendre nos blocs robustes aux pannes logicielles tout autant que matérielles. C'est donc dire que pour chacun des blocs de constructions nous avons travaillé à trouver un procédé ou une fonctionnalité nous permettant de décréter qu'un bloc était robuste. Il est à noter que le schéma de l'architecture ne présente pas le nombre réel de serveurs constituant l'infrastructure, que nous hébergeons aujourd'hui. Par contre, nous pouvons à la suite de l'expérience que nous avons acquise depuis l'opération du projet, dire que chacun des blocs de construction de notre architecture est robuste aux pannes.

Le tableau suivant énumère le nombre de serveurs par bloc de construction que nous devons héberger afin que l'architecture soit minimale et robuste.

Bloc	Nombre de serveurs
MariaDB (SGBD)	2 serveurs, dont un exécute le processus « Galera Arbitrator », ce qui nous permet de ne pas héberger le troisième serveur requis à la grappe et éviter certains frais.
RabbitMQ	2 serveurs, l'un maître et l'autre esclave
PowerDNS	3 serveurs
Serveurs Web	2 serveurs, ils hébergent les projets suivants : CMS, Tableau de bord et l'API
MongoDB	2 serveurs, l'un maître et l'autre esclave, en plus d'un processus d'arbitrage entre les deux
Abonnés	2 serveurs minimum, mais pour la rapidité d'exécution nous en avons 4

Table 1 : Les blocs de construction de l'architecture.

Nous énumérerons dans le tableau suivant pour chacun des blocs comment celui-ci est robuste aux pannes.

Bloc	Robustesse
MariaDB (SGBD)	Nous utilisons la technologie Galera de grappe permettant la création d'une architecture de base de données Maître-Maître. Pour réaliser cette robustesse, nous devons héberger au minimum trois serveurs du SGBD et nous pouvons perdre deux serveurs avant d'être en panne complète. Il est possible de tomber en instance de « split brains », par contre il est possible de les éviter en surveillant les serveurs de près, afin de partir la restauration des serveurs dans l'ordre des pannes.
RabbitMQ (file d'attente)	Ce logiciel nous permet une utilisation de la synchronisation des files d'attente sur un ou plusieurs serveurs esclaves. Lorsque le serveur maître tombe en panne, le serveur esclave prend la relève.
PowerDNS	Comme les serveurs DNS sont toujours aux nombres minimums de 2 et que nous en hébergeons trois, nous pouvons perdre l'usage de deux serveurs avant de tomber en panne complète. Nous pouvons d'ailleurs en ajouter plus si nous le désirons, afin d'être plus robustes encore. Par exemple, Google met quatre adresses IP disponibles, alors qu'Apple met sept adresses disponibles.
Tableau de bord, CMS et API	Ces applications sont des applications Web traditionnelles, il suffit de les héberger sur plusieurs serveurs Web et de mettre un équilibreur de charge en avant de ces derniers, afin de balancer la charge sur chacun d'eux. Nous avons utilisé un répartiteur de charge DNS, utilisant un algorithme de « Round-Robin » sur deux serveurs Web. Ce qui nous a permis de réduire les coûts du répartiteur de charge.
MongoDB	Ce serveur de gestion de document NoSQL nous permet une architecture Maître-Esclave, afin d'être robuste aux pannes, il suffit d'héberger plusieurs serveurs et de les interconnecter ensemble. Lorsque l'interconnexion prendra effet, l'un d'eux se nommera Maître et les autres deviendront esclaves. Lorsque le maître tombe en panne, un des esclaves prend la relève. Le pilote de connexion Python de MongoDB inclut une fonctionnalité de répartition de charge, dès la connexion à ce dernier. Il répartit les opérations d'écriture sur le maître et les opérations de lecture sur les esclaves. En cas de panne, ce dernier se connectera automatiquement au nouveau maître pour les opérations d'écriture et de lecture.
Abonnés	Les abonnés sont probablement le bloc le plus facile à rendre robuste aux pannes quelconques. Ce petit programme est sans état « stateless », car il ne conserve aucune donnée. Il suffit d'en héberger autant que possible et de détecter à l'aide d'un système de surveillance lorsqu'un de ceux-ci tombe en panne. Plus, nous en hébergeons à travers plusieurs centres de données, plus nous réduirons les chances que ce bloc puisse tomber en panne.

Table 2 : Description des blocs de construction de l'architecture.

Le tableau à la page précédente résume bien comment chacun des blocs de construction de l'architecture a été choisi et comment ces derniers possédaient tous une caractéristique de robustesse et de fiabilité. Sans cette caractéristique, nous n'aurions pas pu dire que notre architecture était complètement robuste aux pannes et entièrement fiable pour l'opération en production d'une telle solution de DNS dynamique.

Ces choix ont nécessité plusieurs jours, voire plusieurs semaines de travail : à tester les blocs de construction pour les maîtriser, à effectuer les tests de robustesse et à valider les résultats. Par la suite, lorsque chacun de ces blocs fut accepté et validé, nous avons intégré ces derniers à la solution architecturale.

Description de la mise à l'échelle de l'architecture « scalability »

Lors du travail d'architecture de notre solution de DNS dynamique, nous avons pris le temps de faire plusieurs essais et erreurs dans le choix de nos blocs de constructions de notre architecture. Chacun des blocs sélectionnés a été choisi en prévision d'une augmentation drastique du trafic réseau sur cette dernière et de son support d'une fonctionnalité de mise à l'échelle simple et peu coûteuse.

Le but de notre architecture est de conserver ses qualités de rapidité et d'efficacité à mettre à jour les enregistrements DNS, qu'elle reçoive un petit ou un très grand volume d'utilisateurs. Le plus grand compétiteur dans le monde du DNS dynamique est probablement NoIP.com. Ce fournisseur clame que son service est utilisé par plus de vingt millions d'utilisateurs [38] à travers le monde. L'un des buts de notre architecture était d'être en mesure de supporter quelques utilisateurs à plusieurs millions d'utilisateurs sans avoir à modifier l'architecture de la solution. C'est-à-dire de ne pas avoir à modifier le flux de communication de cette dernière. Bien entendu, nous devrons ajouter plus de serveurs à notre infrastructure, mais nous ne voulions pas avoir à réécrire de code source ou changer des composants logiciels, afin de pouvoir supporter une plus grande charge sur notre système.

Cela explique pourquoi chacun de nos blocs de construction offre un mécanisme de mise à l'échelle. À l'instar de la qualité de fiabilité, nous avons choisi chacun de nos blocs de construction en suivant un deuxième critère de sélection et il s'agissait de celui de la possibilité de mettre à l'échelle le bloc de construction de façon simple et rapide.

Avant de commencer l'explication de la mise à l'échelle de notre architecture, nous devons décrire deux concepts de mise à l'échelle. Il existe deux manières de mettre à l'échelle les serveurs et applications informatiques. La première est une mise à l'échelle verticale et la deuxième est une mise à l'échelle horizontale [40].

La mise à l'échelle verticale

La mise à l'échelle verticale consiste à grossir les ressources matérielles d'un serveur, c'est-à-dire qu'une application s'exécutant sur un serveur donné, si le serveur en question ne possède plus assez de mémoire vive, il suffira d'ajouter dans ce serveur plus de mémoire vive. Cela consiste donc à bonifier l'environnement actuel de l'application sans nécessairement passer par un processus de déménagement de serveur. Il peut par contre arriver que le matériel physique ne soit plus adéquat et que nous changions complètement le matériel pour un plus gros serveur, cela est plus fréquent quand il s'agit de remplacer l'unité centrale d'un serveur par exemple. Il est donc à noter que ce n'est pas l'environnement applicatif qui est modifié, mais bien l'environnement physique, où est hébergée l'application.

En résumé, la mise à l'échelle verticale consiste à augmenter les ressources physiques permettant d'exécuter l'application.

La mise à l'échelle horizontale

La mise à l'échelle horizontale consiste à ajouter plus de matériel physique à la solution en plus de nécessiter la configuration de l'application, afin d'accepter les nouvelles ressources dans son environnement d'exécution. Lorsque nous sommes dans un contexte de mise à l'échelle horizontale, il est souvent question de créer une grappe de serveurs exécutant l'application. Il y aura ensuite un protocole réseau permettant d'équilibrer de quelconque façon qu'elle qu'elles soient les requêtes vers les serveurs de la grappe. L'exemple le plus courant est la mise à l'échelle horizontale d'un site Web. Lorsque le serveur du site Web reçoit trop de requêtes par rapport à sa capacité de les traiter, nous ajouterons un deuxième serveur Web pour aider le premier serveur et nous placerons un équilibreur de charge ayant le rôle de distribuer également les requêtes entre les deux serveurs. Si jamais nos deux serveurs venaient, qu'à ne plus suffire à la demande, il suffirait d'ajouter un troisième serveur derrière l'équilibreur de charge et nous augmenterions le nombre de requêtes pouvant être traité par les serveurs et cette opération serait répétée jusqu'à ce que le nombre de serveurs suffise à la demande.

En résumé, la mise à l'échelle horizontale consiste à modifier l'environnement d'exécution de l'application, afin d'augmenter le nombre de ressources physiques pouvant satisfaire les requêtes envoyées à celle-ci.

La figure 17 suivante démontre bien la différence entre les deux types de mise à l'échelle.

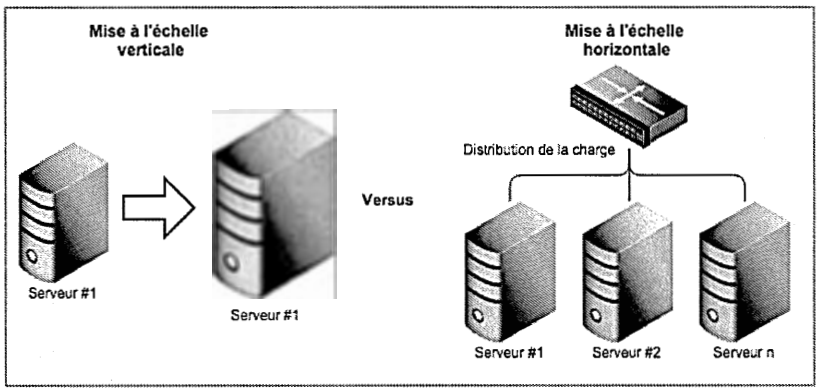


Figure 17 : Les types de mise à l'échelle possible.

D'ailleurs, il est préférable d'avoir une solution avec laquelle nous pouvons faire de la mise à l'échelle horizontale, parce que la formation de la grappe est beaucoup plus flexible dans le temps que l'augmentation des ressources physique. Il est plus difficile d'augmenter seulement le matériel physique, parce qu'à un certain moment, il n'existera pas nécessairement un ordinateur assez gros pour supporter à lui seul toute la charge d'une application. Alors, que lorsque nous ajoutons des serveurs et ceux-ci peuvent être de différentes tailles, il devient plus facile d'augmenter la capacité de l'application. De plus, avec la mise à l'échelle horizontale, il est possible de distribuer géographiquement les requêtes entre plusieurs centres de données. Ce type de mise à l'échelle est donc beaucoup plus flexible que la mise à l'échelle verticale.

Dans les sections suivantes, nous énumérerons de quelle façon nos blocs de construction peuvent être mis à l'échelle. Nous avons d'ailleurs toujours préféré un bloc de construction nous permettant une mise à l'échelle horizontale, c'est donc dire que chacun des blocs peut être constitué en grappe de serveurs, qui nous permettra d'augmenter la capacité de notre infrastructure DNS.

Mise à l'échelle de nos blocs de construction

MariaDB

La mise à l'échelle d'un SGBD se réalise en augmentant le nombre d'écriture et de lecture depuis les données que ce dernier conserve. Or, si nous voulons mettre à l'échelle le SGBD, nous devons pouvoir augmenter le nombre de serveurs physiques constituant la grappe de serveurs actuelle. C'est pourquoi nous avons choisi le SGBD MariaBD, car il nous permet d'ajouter la technologie de grappe nommée Galera. Cette technologie permet au serveur MariaBD de se connecter entre eux et à l'aide d'un algorithme de quorum [41] les serveurs sont en mesure de décider, qui autorisent l'écriture sur de multiples serveurs maîtres. L'algorithme de quorum effectue les validations de transaction au niveau de la grappe de serveurs. Lorsque la grappe valide que la transaction est un succès, les données sont insérées et conservées. Cette technologie nous permet donc de faire une mise à l'échelle dite horizontale, parce que nous pourrons ajouter

plusieurs serveurs dans la grappe, afin d'en augmenter le nombre de lectures des données ou encore le nombre d'écritures.

D'ailleurs, ce qui nous a menés à ce choix de bloc de construction est une expérience professionnelle ayant été un très grand succès, à la suite de la construction d'une grappe de serveur avec l'usage de Galera dans un centre de données montréalais. En plus, d'avoir la possibilité de créer une grappe de serveurs SGBD, les créateurs ont démontré dans un article publié sur leur site Internet [39] que leur solution de grappe résistait à la réplication des données par le réseau Internet étendu. Ils ont fait l'expérience de créer une grappe de serveurs sur plusieurs centres de données d'Amazon. Ils ont créé une grappe de 4 serveurs, le premier serveur était situé à Sao Paulo au Brésil, le deuxième à Sydney en Australie, le troisième en Virginie aux États-Unis et le dernier serveur était situé une autre fois à Sao Paulo. Puis, ils ont démontré qu'ils étaient en mesure d'opérer leur grappe de serveurs de base de données avec de très bonnes performances, ce même lorsque les serveurs sont très éloignés les uns des autres.

Le serveur MariaDB jumelé à la technologie de grappe Galera nous promet une facilité de mise à l'échelle horizontale extrêmement intéressante pour un système de SGBD, et ce à peu de frais.

RabbitMQ

Le serveur de file d'attente RabbitMQ offre plusieurs [42] types de mise à l'échelle. Nous utilisons pour le moment la mise à l'échelle verticale, c'est-à-dire que nous avons un serveur maître et un serveur esclave pour nous assurer de la haute disponibilité du service, mais cette configuration ne nous permettra pas de mettre à l'échelle le nombre de messages en attente ou encore la rapidité de livraison des messages. Par contre, cette configuration convient actuellement très bien au niveau d'utilisation de l'infrastructure.

Lorsque nous aurons le besoin d'augmenter la rapidité de livraison des messages ou encore le nombre de messages pouvant être traités, RabbitMQ offre un mécanisme de mise en grappe et de routage. La mise en grappe consiste à construire plusieurs serveurs RabbitMQ. Puis d'installer un équilibreur de charge pour y envoyer les messages. Ensuite, nous pourrons utiliser le routage, afin de définir la destination des messages vers une grappe désignée. Cela permettra de distribuer le volume des messages sur plusieurs grappes, si le volume devient très grand.

Le bloc de construction RabbitMQ a un potentiel de flexibilité que nous n'avons pas encore entièrement exploré, mais promet d'être assez flexible, afin d'être capable de supporter une augmentation du volume de messages traités sans grande complexité.

PowerDNS

Les serveurs DNS sont généralement hébergés au nombre de deux serveurs, afin d'assurer une certaine disponibilité, parce qu'il s'agit d'une pièce maîtresse des réseaux informatiques. C'est pourquoi notre architecture ne fait pas exception et nous avons pour le moment trois serveurs hébergés dans différents centres de données géographiquement très éloignés l'un de l'autre.

Il est possible avec nos serveurs DNS de placer un serveur dans plusieurs pays, puis de se servir de mécanisme de routage pour diriger le trafic provenant d'un fournisseur Internet vers le serveur le plus proche de l'utilisateur. La plupart des grands opérateurs de DNS utilisent les réseaux Anycast [43] et éventuellement notre infrastructure pourrait faire usage de ce type de réseau, mais cela n'a aucun impact sur notre architecture DNS. Il y a encore moins d'impact sur le temps de réplication de nos données. La différence se trouve dans la configuration de notre inventaire de serveurs DNS, laquelle possédera une entrée en base de données par serveur. L'impact se fera ressentir au niveau de nos abonnés lorsqu'ils contacteront les serveurs DNS à mettre à jour, car il y en aura un plus grand nombre qu'auparavant et nous devrons probablement augmenter la quantité des abonnés. Éventuellement au lieu de mettre à jour trois serveurs DNS, peut-être mettront-ils à jour une dizaine tous situés dans divers pays ou localement. La distribution du trafic réseau depuis un réseau Anycast ou un autre procédé n'affecte donc pas du tout notre mécanisme de mise à jour de l'infrastructure DNS, ce qu'elle affecte est plutôt la distribution du volume de paquet DNS sur l'entièreté de l'infrastructure DNS.

La mise à l'échelle de nos serveurs DNS est donc une mise à l'échelle horizontale, car il suffit d'ajouter un plus grand nombre de serveurs, afin de traiter un plus grand nombre de requêtes. Puis, lorsque le nombre de serveurs est augmenté, il suffit de configurer le réseau informatique de sorte que les requêtes soient acheminées de façon aléatoire à l'ensemble des serveurs par le réseau Anycast.

Tableau de bord, CMS et API

Étant donné, qu'il s'agit d'application Web traditionnelle, ces dernières sont très simples à mettre à l'échelle, de plus il s'agira dans ce cas précis de mise à l'échelle horizontale. Il suffira de configurer un plus grand nombre de serveurs Web, afin d'accepter plus de requêtes HTTP et de configurer tous ces serveurs derrière un équilibreur de charge. Ce type de mise à l'échelle est de nos jours très commun et ne représente aucunement un défi.

D'ailleurs, nous pourrons dans ce cas-ci utiliser une technologie de distribution de charge permettant d'envoyer la requête HTTP vers le serveur le plus proche de l'utilisateur.

Il s'agit donc du bloc de construction le plus simple à mettre à l'échelle.

MongoDB

Le serveur de documents MongoDB nous offre plusieurs façons de mettre à l'échelle une grappe de serveurs. La première méthode consiste à créer une architecture Maître-Esclave et la deuxième inclut la présence de partition de données. Pour le moment, nous utilisons la première méthode, car celle-ci nous convient amplement, de plus elle nous permet de mettre à l'échelle horizontalement la lecture des fichiers journaliers de l'application. La mise à l'échelle horizontale de l'écriture des fichiers journaliers devra quant à elle passer par la mise en place de partition de données.

Nous rappellerons que pour le moment MongoDB représente un bloc de construction qui est optionnel à notre architecture, mais qui est plutôt pratique pour les fins de journalisation des événements sur la plateforme.

D'ailleurs, nous l'avons choisi pour sa facilité de mise en service et parce que ce dernier nous permet de mettre à l'échelle l'écriture et la lecture des documents.

Abonnés

Comme décrit dans le tableau #2, les abonnés sont de petits programmes sans états et ils peuvent être exécutés n'importe où depuis n'importe quel centre de données. Ils n'ont pas besoin d'être hébergés dans un centre de données nous appartenant, ni même d'être connectés à notre réseau local. Tout ce dont ils ont besoin c'est de se connecter à notre serveur de messagerie RabbitMQ et d'être capable d'appeler l'API REST, qui est hébergé sur nos serveurs DNS. Donc, dès qu'ils ont accès à une connexion Internet nos abonnés sont fonctionnels.

Si nous désirons mettre à l'échelle ce bloc de construction, ce que nous faisons déjà, il suffit d'exécuter plusieurs abonnés sur différents serveurs dans différents centres de données. Par exemple, nous en avons hébergé trois derrière nos connexions Internet personnelles et un dernier sur le serveur de base de données qui lui est hébergé aux États-Unis chez un fournisseur d'hébergement cloud à Austin au Texas.

Ce bloc de construction, nous permet de créer un logiciel extrêmement performant, en plus de nous fournir la capacité de le mettre à l'échelle facilement.

En résumé, dans la section présente nous avons décrit chacune des méthodes de mise à l'échelle de nos blocs de construction. Tous les blocs peuvent être mis à l'échelle et s'il advenait que nous augmentions drastiquement le trafic sur notre infrastructure DNS, nous serions en mesure de réagir à l'impact de ce nouveau trafic rapidement sur chacun de nos blocs de construction. Il nous apparaît évident que nous serons en mesure de mettre à l'échelle notre infrastructure sans en changer l'architecture de notre solution DNS.

Neutralité de l'architecture

Comme décrit, à la section de la mise à l'échelle de notre solution, nous ne voulions pas avoir à retravailler l'architecture de notre solution, si nous avions à traiter un très grand nombre de requêtes ou encore un très grand volume de mises à jour des enregistrements DNS. Notre but était, donc de rendre notre architecture la plus neutre possible.

Nous traiterons les deux cas possibles, où il y aurait une possibilité de rapidement devoir mettre à l'échelle notre solution DNS. Les deux cas d'utilisation sont les suivants :

1. Très grand nombre de mises à jour des entrées DNS depuis l'API ou le tableau de bord.
2. Très grand volume de requêtes DNS sur les serveurs PowerDNS.

Dans le premier cas d'utilisation, nous devrons mettre à jour minimalement trois des blocs de construction de notre architecture, soit l'API, le tableau de bord et les abonnés. Il est aussi possible que nous ayons à mettre à jour la base de données, mais cela n'est pas certain, car il s'agit d'une grappe qui supporte l'écriture et la lecture depuis tous les membres de la grappe. Il existe aussi la possibilité que nous ayons à modifier un cinquième bloc, c'est-à-dire la configuration de la solution RabbitMQ pour la transformer en grappe de serveurs. Comme vu à la section précédente, nous sommes en mesure de mettre à l'échelle les trois composants minimalement requis, sans grande problématique. Afin d'augmenter la capacité de réception de requêtes HTTP depuis l'API ou le tableau de bord, il suffit d'ajouter des serveurs Web à notre grappe existante. Puis d'ajouter les serveurs à la configuration de notre distributeur de charge. En ce qui concerne les abonnés, nous devrons en augmenter le nombre, car l'augmentation de mise à jour des entrées DNS implique que les abonnés devront exécuter plus de mise à jour depuis les messages reçus sur RabbitMQ vers les serveurs PowerDNS.

Ainsi dans la première situation de mise à l'échelle, il suffit d'ajouter des serveurs, qu'ils soient physiques ou virtuels. Lorsque ceux-ci sont ajoutés, nous augmentons automatiquement le nombre de requêtes pouvant être traitées par notre solution architecturale.

Dans le cas d'utilisation numéro deux, où le nombre de requêtes DNS traitées doit être augmenté, nous aurons à modifier principalement le bloc de construction PowerDNS et sa configuration réseautique. Il s'agit encore une fois d'augmenter le nombre de serveurs physiques ou virtuels. Ensuite, nous devrons configurer le protocole réseau BGP, afin de créer notre réseau Anycast [43] et distribuer les requêtes DNS sur tous les serveurs DNS.

Pourquoi utiliser un réseau Anycast ?

Parce que ce type de réseau nous permet d'envoyer au serveur le plus proche la requête du client et ce type de réseau possède qu'une seule adresse d'accès. Ainsi le client voit sa requête traitée et retournée très rapidement. Il s'agit ici de minimiser le délai dans le transport des paquets réseau du client à notre serveur, puis de notre serveur vers le client. De plus, un réseau Anycast possède l'avantage d'annoncer une seule adresse à contacter et c'est le protocole, qui par la suite

distribue les requêtes DNS vers le serveur le plus proche du client. La figure [43] suivante explique le principe d'un réseau Anycast.

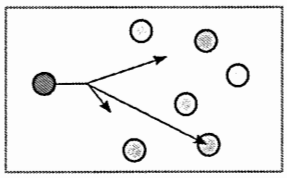


Figure 18 : Un réseau Anycast.

D'ailleurs, les deux cas d'utilisation nécessitent une mise à l'échelle. En plus, les deux cas risquent fortement de survenir dans le futur, mais nous ne craignons pas de devoir modifier le code ou l'architecture de notre solution, afin de répondre à l'augmentation de la charge des requêtes sur notre solution DNS, car tous les blocs de construction peuvent être mis à l'échelle sans nécessiter de modifications à l'architecture de la solution DNS.

Les avantages découverts à la suite de la mise en place de l'architecture

La facilité de gestion des noms de domaine

En plus de posséder plusieurs qualités logicielles, l'architecture proposée nous permet d'ajouter de la facilité à la gestion des noms de domaine hébergés sur nos serveurs. En se servant de nos abonnés, il devient extrêmement facile d'approvisionner ou de supprimer de nouvelles zones ou encore un nouveau serveur DNS, afin de desservir une plus grande capacité de requêtes DNS.

Jan-Piet Mens un expert dans le monde du DNS ayant publié le livre « Alternative DNS Server » [20] explique dans un article nommé « Automatic Provisioning of slave DNS server » [18] sur son blogue la difficulté, qu'il a pu rencontrer lorsqu'il est nécessaire d'ajouter ou supprimer des zones sur des serveurs DNS tels que Bind ou NSD. La problématique survient lorsque le serveur maître ajoute une zone. Le serveur esclave doit connaître le fichier de configuration de la zone or, il n'est pas toujours évident de transférer le fichier de configuration de la zone avant que le serveur maître ait lui de son côté déjà envoyé le message de notification de changement de la zone.

Notre architecture nous permet d'éviter ce type de situation, où la mise à jour des zones est réglementée par le mécanisme du RFC2136. Dans notre cas, un approvisionnement automatique d'un serveur esclave passerait simplement par deux petites étapes réalisées dans notre tableau de bord, la première étant l'ajout dans l'inventaire du nouveau serveur DNS depuis le CMS, puis l'envoi d'un message dans la file d'attente du serveur RabbitMQ demandant d'approvisionner le nouveau serveur. Ensuite, un abonné prendrait le relais en mettant à jour la ou les zones indiquées dans le message.

Ce mécanisme, nous l'utilisons aussi afin d'assurer la synchronisation des serveurs à la suite d'une panne quelconque étant survenue lors de l'opération de notre solution de DNS dynamique. Qu'il s'agisse d'une panne logicielle, matériel ou réseau, nous pouvons faire en sorte de resynchroniser les zones en envoyant un message depuis notre tableau de bord logiciel.

Ceci s'est avéré être un avantage que nous avons découvert à la suite de la mise en production de notre architecture, car la gestion est de ce fait très flexible et nous permet d'écrire des outils d'automatisation de nos zones DNS.

Autocorrection et resynchronisation des zones

Il est possible de corriger une entrée DNS grâce aux abonnés. C'est-à-dire que nous avons écrit un programme, qui est exécuté par les abonnés et qui peut être déclenché à la demande depuis notre tableau de bord par un administrateur ou un client. Lorsqu'une opération de mise à jour d'une entrée DNS échoue et que pour une quelconque raison n'atteignait pas le serveur DNS distant, il est alors possible de déclencher l'action de forcer la mise à jour une nouvelle fois. Cette exécution est idempotente, car elle rejoue l'état tel qu'il est connu par notre tableau de bord à sa dernière mise à jour.

D'ailleurs, il est possible de demander à la volée une synchronisation d'une zone entière. Il faut se remémorer le fait que toutes les zones sont conservées sur notre SGBD centrale. Cette action est par contre réservée aux administrateurs de la plateforme et peut être déclenchée lorsqu'un bris physique survient sur un serveur. À ce moment, l'administrateur peut décider qu'il doit remettre à jour une zone et il pourra le faire depuis le tableau de bord, ce qui enverrait aux abonnés un message de synchronisation de la zone.

Ces avantages proviennent principalement de l'utilisation des abonnés, ce qui nous permet d'écrire des applications d'automatisation de la plateforme, puis de pouvoir exécuter ces applications depuis notre tableau de bord.

Désavantage de notre solution architecturale

Le désavantage de notre solution architecturale est le nombre de blocs de construction. Nous sommes conscients que le nombre de blocs augmente la complexité d'opération de notre solution DNS. L'infrastructure devant être mise en place afin de supporter notre architecture est volumineuse et nécessite l'apprentissage de plusieurs technologies. De plus, le nombre de blocs augmente aussi le risque de pannes sur notre infrastructure et c'est pourquoi nous avons cherché à réduire ce risque en utilisant des blocs possédant la qualité d'être fiable et facilement mis à l'échelle.

Le temps réel

Pourquoi discuter de temps réel ?

Toutes les entreprises de nos jours se servent du DNS. Le DNS consiste en une pièce maîtresse du réseau Internet que nous connaissons aujourd'hui. Au cours de ce projet, nous avons parlé d'un standard commun à l'industrie du DNS, qui est celui du TTL ou plus simplement du temps de mise en cache par les serveurs DNS récursifs d'un enregistrement DNS qu'ils auront dû résoudre. Ce standard est souvent associé à une période de temps de 300 secondes, soit 5 minutes. De plus en plus de fournisseurs de DNS permettent désormais de diminuer ce délai à 60 secondes, soit une minute. Ceci est généralement vrai pour les entrées DNS statiques, qui ne changent pas dans le temps ou pratiquement pas. Le délai court permet à certaines entreprises de créer des mécanismes de reprise de service en cas de panne. Ainsi, si leur service principal tombe en panne, ils pourront modifier l'entrée DNS et la faire pointer sur un centre de données ou de relève différent, tout en sachant qu'au-delà des 60 secondes de délais tous leurs clients seront désormais redirigés sur le service qui est quant à lui fonctionnel. Par contre, cela a un impact extrêmement négatif sur une infrastructure DNS et c'est celle de la surcharge de trafic.

Autrefois, nous attribuions un TTL plus élevé et les serveurs récursifs gardaient en cache l'adresse DNS durant 1 heure voire jusqu'à 24 heures. Ceci faisait en sorte que les opérateurs de DNS autoritaires n'avaient pas à héberger des centaines de serveurs pour répondre à la charge du trafic DNS, car c'étaient les serveurs récursifs, qui recevait cette charge. D'ailleurs, héberger un serveur DNS récursif est beaucoup plus simple, car une fois qu'il est configuré convenablement, l'administrateur risque de ne plus jamais s'en occuper, à moins qu'une panne survienne.

Nous voulions donc être en mesure de diminuer le temps de mise en cache des entrées DNS dynamique, car celle-ci est par défaut de 300 secondes et très peu de fournisseurs proposent de la diminuer. Nous voulions appliquer la même qualité de temps de réponse sur une entrée DNS dynamique que celle qui est statique. C'est à dire 60 secondes ou moins si possible. Par contre, cela a pour conséquence d'ajouter une charge de trafic sur notre infrastructure DNS et aussi sur notre API, qui reçoit les demandes de changements des entrées DNS.

De plus, si nous revenons à notre besoin de départ, qui était de surveiller des équipements réseau dans un délai de 300 secondes, si notre entrée DNS dynamique se fait modifier entre deux périodes de vérification du service, alors nous sommes certains que l'adresse IP sera redemandée entre chaque vérification. Cela aura pour conséquence d'assurer une détection très rapide des pannes et des reprises de services.

À la suite du travail d'architecture que nous avons proposée pour une infrastructure de DNS, nous avons fait les expérimentations nécessaires, afin de déterminer s'il était réaliste de dire que nous étions en mesure de supporter les mises à jour des entrées dynamiques de façon à ce qu'elles soient en temps réels.

Expérimentations

À la suite des expérimentations sur l'architecture de développement, nous avons réussi notre pari de réaliser la mise à jour soit instantané lors de la première requête et au maximum dans un délai de 5 secondes, à la deuxième requête. Nous avons été capable de soutenir une mise à jour en deçà de 2 secondes pendant toute la période de test sur l'environnement de laboratoire. Nous nous sommes servis de notre client Android ainsi que de la fonction de changer l'adresse IP entre l'IP local (LAN) et public (WAN). Ceci nous permettait de faire des changements d'adresse IP instantanés. Nous nous sommes aussi servis de l'outil à la ligne de commande « dig », qui permet d'interroger les serveurs DNS. Dans notre expérimentation, nous avons interrogé les serveurs DNS autoritaires directement sans passer par un serveur DNS récursif.

Nous décrirons brièvement le procédé derrière l'expérimentation. Vous aurez aussi besoin d'un script Bash, qui contient le code suivant :

```
#!/bin/bash
for server in ns1 ns2 ns3
do
        dig @$server.example.com +short $1 $2
done
```

Figure 19 : Script Bash permettant d'interroger nos serveurs DNS.

Ce script peut être appelé de la manière suivante :

```
~:digit android.example.com
192.168.1.12
192.168.1.12
192.168.1.12
```

Figure 20 : L'exécution du script Bash et son résultat.

Le client Android pour sa part doit être changé à la demande par un humain sur l'option nommée « Use LAN IP for updates ». Tel que démontré dans les figures suivantes :

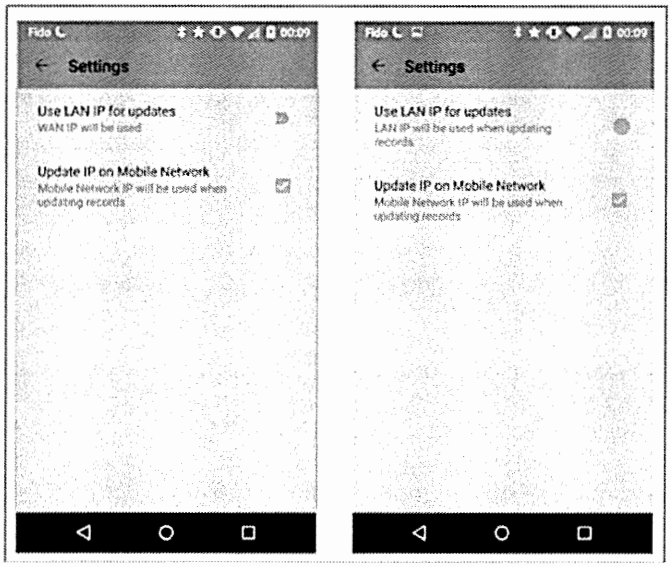


Figure 21 : Option d'utiliser l'adresse LAN.

L'utilisation de cette option permet d'émettre rapidement une requête HTTP à l'API, afin de signaler un changement d'adresse IP pour l'entrée DNS dynamique « android.example.com ».

Nous aurions aussi pu utiliser de manière programmée l'appel à l'API permettant de mettre à jour une entrée DNS dynamique à l'aide de l'outil à la ligne de commande « cURL ». Ceci aurait donné le résultat suivant :

```
curl 'https://api.example.com/nic/update/?username=drivard@example.com&password=example&ip=8.8.8.8&hostname=android.example.com'
good 8.8.8.8
```

Figure 22 : Mise à jour de l'adresse IP par l'API et l'utilitaire cURL.

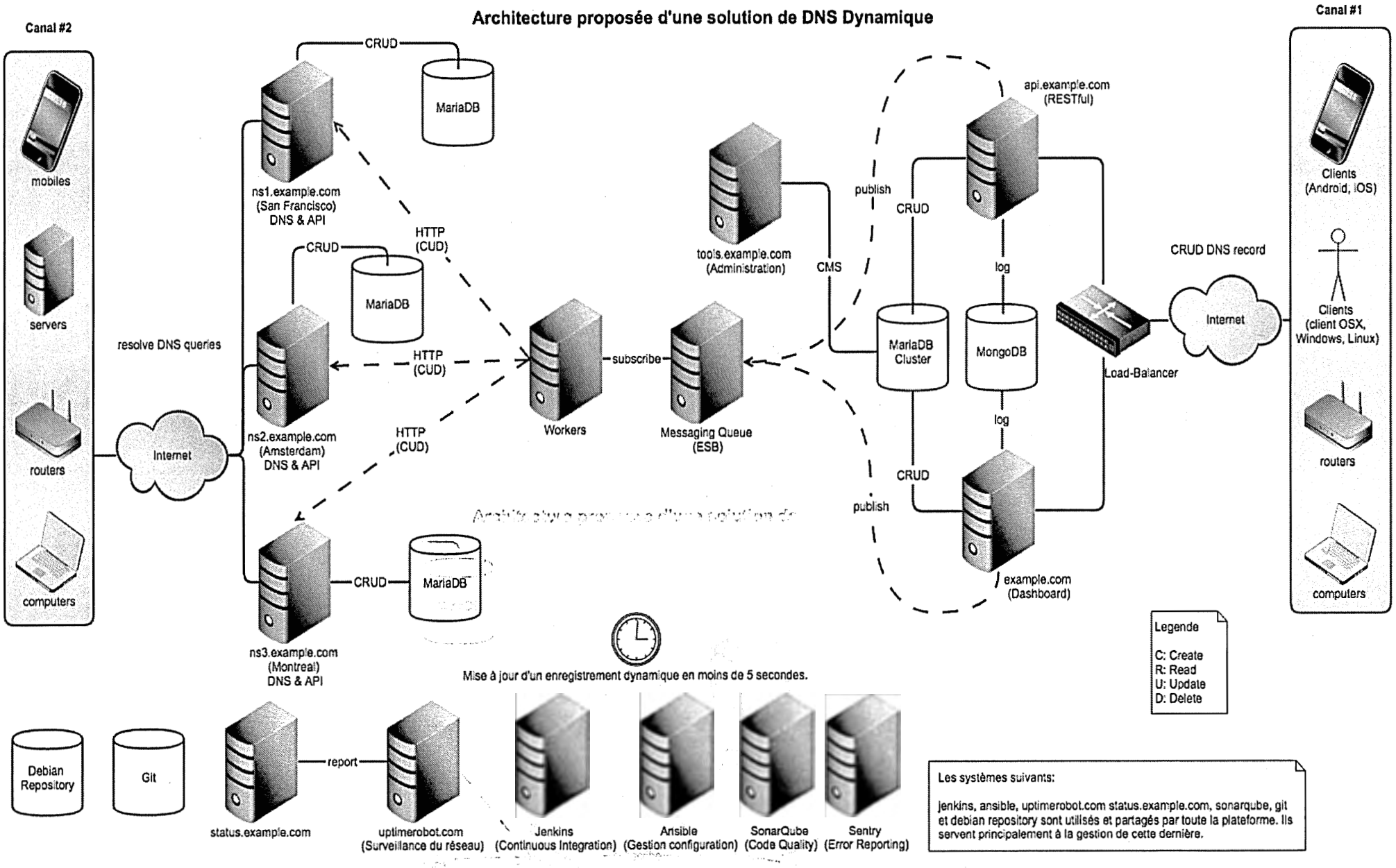
À la suite de la mise à jour de l'adresse IP à l'aide du client mobile, nous avons pu remarquer que la requête DNS auprès du serveur prenait jusqu'à 20 secondes avant de retourner la nouvelle adresse, à la suite de la lecture des configurations du serveur PowerDNS, nous avons été en mesure de modifier cette dernière, de façon à ce que nous puissions invalider la requête au bout de 5 secondes au niveau de la cache du serveur PowerDNS. Par contre, nous ne voulions pas non plus mettre à risque l'opération de la plateforme, c'est pourquoi pour le moment nous trouvions acceptable de réduire le risque de surcharge en laissant les requêtes en cache un maximum de 5 secondes. Cela réduit la charge de travail des serveurs DNS légèrement et permet d'être pratiquement en temps réel lors de nos mises à jour des adresses IP dynamiques.

En résumé, nous sommes grâce à l'architecture proposée en mesure de mettre à jour de manière programmée les enregistrements DNS dynamiques en temps réel, et ce sur tous les serveurs DNS en même temps. Tout cela en supprimant le temps d'attente de la réplication de nos zones, tel que le serveur Bind le ferait. Il s'agit dans ce cas-ci du bénéfice de la parallélisation de notre procédé versus la méthode plutôt séquentielle qu'utilise le serveur Bind.

Architecture finale

L'architecture finale vous sera présentée dans la figure 23 à la page suivante. Elle inclut tous les éléments, qui constituent l'univers technologique du projet. Vous y verrez en outre l'inclusion du système de révision du code source, l'intégration continue, la surveillance de la plateforme, la gestion de la configuration, ainsi que la gestion des billets et la gestion de la qualité du code source par SonarQube [47].

D'ailleurs, dans l'annexe de ce document vous pourrez y trouver une section dédiée aux résultats de SonarQube de l'analyse sur l'état du code source. L'analyse survient, à la suite des « commit » sur le gestionnaire de code source « git ». Cela nous permet de voir et comprendre la qualité de notre code, sa complexité et les améliorations à lui apporter.


Architecture proposée d'une solution de DNS Dynamique
Canal #2
mobiles
servers
routers
computers
resolve DNS queries
Internet
CRUD
MariaDB
ns1.example.com
(San Francisco)
DNS & API
HTTP
(CUD)
CRUD
MariaDB
ns2.example.com
(Amsterdam)
DNS & API
HTTP
(CUD)
HTTP
(CUD)
CRUD
MariaDB
ns3.example.com
(Montreal)
DNS & API
Workers
subscribe
Messaging Queue
(ESB)
tools.example.com
(Administration)
CMS
publish
CRUD
api.example.com
(RESTful)
log
MariaDB
Cluster
MongoDB
log
CRUD
publish
example.com
(Dashboard)
Load-Balancer
CRUD DNS record
Internet
Canal #1
Clients
(Android, iOS)
Clients
(client OSX,
Windows, Linux)
routers
computers
Mise à jour d'un enregistrement dynamique en moins de 5 secondes.
Legende
C: Create
R: Read
U: Update
D: Delete
Debian
Repository
Git
report
status.example.com
uptimerobot.com
(Surveillance du réseau)
Jenkins
(Continuous Integration)
Ansible
(Gestion configuration)
SonarQube
(Code Quality)
Sentry
(Error Reporting)
Les systèmes suivants:
jenkins, ansible, uptimerobot.com status.example.com, sonarqube, git
et debian repository sont utilisés et partagés par toute la plateforme. Ils
servent principalement à la gestion de cette dernière.

Conclusion

En résumé, nous voulions proposer une architecture ayant pour but l'amélioration d'une infrastructure de serveurs DNS autoritaire, qui serait en mesure de répondre aux besoins suivants :

- Les mises à jour en temps réel des entrées DNS,
- La mise à l'échelle horizontale de la solution,
- Une architecture modulaire suivant les besoins requis.

Au cours de ce présent projet, nous avons tenté de répondre à ces objectifs. Les sujets traités dans ce projet sont principalement l'architecture d'une plateforme d'une solution de DNS autoritaire et la problématique du délai de mise à jour dans le standard des enregistrements de DNS dynamiques. Notre intention est de trouver une solution au délai de 300 secondes imposé par le RFC2136 et respecté à quelques exceptions près à travers l'ensemble de l'industrie du DNS. D'ailleurs, nous avions à répondre à un besoin, qui exigeait de mettre à jour les enregistrements dynamiques plus rapidement que les 300 secondes du standard. Nous avions en fait une exigence d'être pratiquement en temps réel. La recherche et l'expérimentation exigées par ce projet nous ont permis d'entrevoir une solution architecturale qui pourrait possiblement atteindre cet objectif.

La problématique à résoudre était complexe puisqu'il faut arriver à diminuer considérablement le délai de mise à jour d'un enregistrement DNS dynamique. Nous ne voulions pas non plus réécrire un logiciel de serveur DNS autoritaire, puisqu'il existe de très bon logiciel libre, qui répondent déjà à ce besoin et qui respectent tous les standards déjà établis. Notre intention est de partir de logiciels libres existants et de construire autour de ceux-ci une solution à notre problématique.

Pour la construction de cette architecture nous nous sommes inspirés de la méthode TOGAF [48] et de la notion de blocs de constructions. L'architecture proposée est un amalgame de logiciels libres ou ayant été écrit pour répondre à notre besoin. La méthode TOGAF préconise d'utiliser les blocs de constructions technologiques, afin de créer la solution technologique recherchée. Dans l'architecture proposée, nous nous sommes inspirés sur le concept des blocs de constructions technologiques comme étant le point central de nos choix de solutions. Chaque bloc fut analysé et testé, puis classé selon les exigences préétablies. Ensuite, nous avons pu assembler la solution architecturale en sachant que nous respections les exigences de robustesse, de mise à l'échelle horizontale et de fiabilité recherchée pour notre solution.

Nous avons opté pour une approche de prototype itératif [49], afin de construire l'architecture selon les exigences des objectifs visés. De plus, ce procédé nous a permis de facilement revenir en arrière et changer un bloc de construction en cours de route. Nous avons utilisé le prototype itératif puisque nous voulions être en mesure de voir nos progrès rapidement et ainsi ajuster l'architecture dès que nous détections que nos objectifs seraient mis à risque. La gestion de notre

environnement de test nous a aussi aidés à tester le prototype. Chaque construction du code devait obligatoirement être déployée sur l'environnement de développement.

D'ailleurs, ce projet nous a permis de dégager plusieurs résultats intéressants quant à l'hébergement d'une solution de DNS autoritaire. Premièrement, nous pensons être en mesure de décrire l'implémentation d'une architecture logicielle nous permettant d'optimiser la mise à jour des enregistrements DNS dynamiques. En outre, tel que vu à la section de l'expérimentation, le plus important de ces résultats est probablement la mise à jour en deçà de 2 secondes d'un enregistrement DNS dynamique et cela dans un environnement de laboratoire contrôlé. Bien que nous ayons été capable de diminuer le délai à 2 secondes lors de nos tests programmés, nous avons par la suite augmenter ce dernier à 5 secondes puisque le serveur DNS utilisé lors des tests demandait l'utilisation d'un mécanisme de cache, afin de livrer de meilleures performances. C'est suite à la lecture et à la compréhension du fonctionnement interne du logiciel de serveur DNS autoritaire utilisé dans nos tests, que nous avons augmenté le délai de cache puisqu'il s'avérait plus efficace et performant de laisser à 5 secondes le délai de mise en cache. Nous avons découvert ce comportement puisque le mécanisme de cache nous empêchait de reproduire la mise à jour en deçà de 2 secondes dès le deuxième essai de test. Nos premières expérimentations consistaient en un premier délai très court suivit d'un délai plus long de quelques secondes. Nous avons donc décidé de normaliser le délai à 5 secondes, ainsi nous nous assurons de toujours avoir un délai précis et un comportement reproductible. De plus, augmenter à 5 secondes le délai de mise en cache a pour effet de diminuer la charge sur le serveur de DNS autoritaire.

La diminution du délai à 5 secondes dans un environnement de tests contrôlés, contrairement aux 300 secondes du standard, représente une diminution de 98% du délai de mise à jour d'un enregistrement dynamique. Même si finalement nous avons accepté de plafonner le délai à 5 secondes, afin d'éviter la surcharge de l'infrastructure de notre laboratoire, nous croyons qu'il serait possible d'éliminer ou baisser d'avantage ce plafond. Cependant, nous devrions investiguer deux types d'améliorations pour diminuer le délai de 5 secondes et il s'agirait d'analyser la possibilité d'ajouter plus de serveur à une grappe de serveurs DNS autoritaire pour distribuer la charge ou encore augmenter la capacité physique du serveur informatique. Puisque nous avons préféré la mise à l'échelle horizontale dans ce document, il serait préférable selon nous de valider l'hypothèse de la formation de grappe de serveurs pour distribuer la charge. Nous croyons que l'architecture proposée serait donc en mesure de supporter la mise à jour en temps réel des enregistrements sans grande amélioration autre que la capacité physique et la quantité des serveurs hébergeant la solution. Par contre, il faudrait à la base avant d'implémenter ce type d'architecture faire le travail de recherche sur la volumétrie devant être supporté par le ou les serveurs de DNS autoritaires.

Le deuxième résultat tout aussi intéressant est la possibilité d'utiliser l'architecture proposée afin de mettre à l'échelle l'infrastructure de DNS autoritaire. Puisque nous avons utilisé des blocs de construction afin de bâtir l'architecture, nous avons été en mesure de choisir des blocs permettant la mise en haute disponibilité des services requis par cette dernière. Il est important de comprendre que la mise à l'échelle d'une solution informatique est souvent complexe, surtout dans le cas où la plateforme est utilisée dans un environnement de production. C'est la raison

pour laquelle dans ce projet nous avons considéré l'aspect de la mise à l'échelle horizontale dès le début du projet. D'ailleurs, pendant la rédaction de ce travail, il est survenu une attaque [55] par déni de services distribués sur une infrastructure publique de DNS, qui est hébergée par l'opérateur Dyn.com et nous croyons que cela renforce l'idée qu'en proposant une architecture de DNS autoritaire, il fallait être en mesure de prendre davantage de charge par une mise à l'échelle horizontale. D'ailleurs, nous pensons avoir démontré qu'il est possible de mettre à l'échelle horizontale chacun des blocs technologiques constituant l'architecture proposée dans la section de la description de la mise à l'échelle de notre architecture.

Le dernier but de modularité selon les besoins, nous croyons l'avoir démontré par la description de l'architecture et de chacun des blocs technologiques utilisés dans la création de cette dernière. Nous croyons que notre architecture est flexible et malléable selon les besoins qui seront définis avant son utilisation. Par exemple, il est possible pour quiconque qui ne maitrise pas le SGBD MariaDB de choisir une alternative à ce dernier. La modification du bloc technologique du SGBD est possible et ne requiert aucune modification à l'architecture en elle-même. Elle nécessiterait une modification aux codes sources mais sans plus. Bien évidemment, nous suggérons de toujours considérer les alternatives permettant la mise à l'échelle horizontale.

En ce qui a attrait aux limites ou désavantages d'une architecture logicielle, tel que nous la proposons, il est évident que l'infrastructure derrière l'opération de cette solution peut être complexe ou le devenir. L'architecture proposée comporte plusieurs technologies et un grand nombre de serveurs à héberger. Par contre, il est à noter que c'est cet amalgame de technologies, qui nous procure la flexibilité dont nous avons discutée tout au long du document ci-présent. Afin de pallier cette complexité, il suffit de se pourvoir d'un excellent système de surveillance et de s'assurer que la plateforme est entièrement surveillée. Lorsque l'opérateur de DNS possédera le bon niveau de surveillance et le bon niveau d'alertes, la complexité de l'opération de la plateforme devrait être grandement diminuée. De plus, l'audit des événements se produisant sur la plateforme aidera par la suite l'opérateur à comprendre comment les utilisateurs ont utilisé cette dernière. Cela permettra un débogage rapide et efficace par ce dernier.

Puisque nous ayons hébergé notre architecture dans un environnement de laboratoire, nous nous sommes questionné à savoir comment nous hébergerions cette dernière dans un contexte réel d'utilisation. Nous avons alors pensé à l'opérationnalisation du réseau permettant d'accéder à la plateforme de DNS autoritaire. Afin d'ajouter à la mise à l'échelle horizontale de l'architecture, nous recommanderions d'utiliser un réseau dont la topologie serait de type Anycast. La mise en place d'un réseau Anycast pour tout hébergement public des serveurs DNS permettrait en cas de panne logicielle ou matérielle une reprise du service sans impact pour les utilisateurs finaux. De plus, le réseau Anycast permettrait de distribuer les paquets réseautiques vers le serveur le plus proche de l'utilisateur. Ceci améliorerait les performances du service et en cas de panne, l'utilisateur serait alors desservi par un serveur plus loin en matière de temps de réponse, par contre il ne ressentirait pas l'impact de la panne sur le service, qui lui est offert.

Une deuxième recommandation serait d'utiliser une grappe de serveurs par centre de données, nous avons hébergé que le nombre minimum nécessaire dans l'environnement de laboratoire,

mais il serait par exemple possible de créer une grappe de serveur par région géographique, ainsi nous pourrions augmenter le nombre de requêtes DNS desservies par une région. Cela serait facilement possible de créer une grappe de serveurs par région en utilisant une technologie tel que le serveur Nginx [52], qui permet le balancement de la charge des applications UDP tel que le DNS. Puisque le serveur Nginx est déjà utilisé pour l'hébergement HTTP de l'API, il ne s'agirait pas dans ce cas-ci d'un ajout d'une technologie, mais bien de réutilisation d'un bloc de construction faisant déjà partie de l'architecture de la plateforme. Du moins, il s'agit du procédé que nous recommanderions, mais il existe d'autres pistes de solutions et beaucoup d'autres technologies de balanceur de charge, qui pourrait être utilisé.

À la suite de ces améliorations, nous proposerions une piste de recherche, que nous apparaît comme étant une suite logique à l'architecture proposée. Il s'agirait de chercher à améliorer le partitionnement des noms de domaines sur divers serveurs. Nous pouvons facilement imaginer qu'un hébergeur grand public de service DNS tel que noip.com ou dyn.com puisse vouloir créer diverses grappes de serveurs DNS permettant de dédier une grappe à un client distinct. Dans ce contexte, il serait intéressant d'approfondir la stratégie de répartition des noms de domaines sur une grappe dédiée tout en utilisant l'API centrale à notre architecture. Le but consisterait à trouver une solution de partage des ressources tant physiques qu'informationnel qui est contenu dans la solution de DNS autoritaire. L'hypothèse derrière cette recherche est que nous arriverions à mettre à l'échelle tant nos ressources que possiblement celle des clients d'un opérateur en allant vers un hébergement hybride entre les centres de données de l'opérateur et ceux de son client. Cela permettrait de se protéger dans des cas d'attaques [55] tel que vu récemment chez l'opérateur Dyn.com.

En conclusion, nous pensons avoir proposé une architecture qui à première vue semble répondre aux objectifs préalablement définis. Nous croyons que l'architecture proposée rendrait possible l'opération d'un hébergeur de DNS autoritaire capable de mettre à jour des entrées DNS dynamiques pratiquement en temps réelle et que l'opération serait assez flexible et modulaire afin qu'elle puisse prendre facilement un grand volume de requête sans nécessité de modifications.

Annexes

Table de références

1. Internet Engineering Task Force (IETF) P. Vixie ISC. 2016. « Dynamic Updates in the Domain Name System (DNS UPDATE)». En ligne. <https://www.ietf.org/rfc/rfc2136.txt>. Consulté le 28 décembre 2015.
2. EfficientIP SAS, "DNS architectures" efficient iP the global IPAM company, 2012. En ligne. <http://www.callevanetworks.com/downloads/efficientip/eipwebinar-dnsarchitectures-022013.pdf>. Consulté le 16 décembre 2015.
3. ZyTrax, Inc., 2016. « Chapter 2 DNS Concepts ». En ligne. <http://www.zytrax.com/books/dns/ch2/index.html#structure>. Consulté le 16 décembre 2015.
4. OlinData, 2016. "DNS: A Journey Within (Part II) | OlinData". En ligne. <http://www.olindata.com/blog/2014/02/dns-journey-within-part-ii>. Consulté le 14 janvier 2016.
5. Wikimedia Foundation, Inc. 2016. "Comparison of DNS server software—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Comparison_of_DNS_server_software>. Consulté le 14 janvier 2016.
6. Internet Engineering Task Force (IETF), P. Vixie ISC. 2016. "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)". En ligne. <https://www.ietf.org/rfc/rfc1996.txt>. Consulté le 15 janvier 2016.
7. Wikimedia Foundation, Inc. 2016. "Domain Name System—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Domain_Name_System>. Consulté le 15 janvier 2016.
8. The Internet Assigned Numbers Authority (IANA). 2016. "IANA-managed Reserved Domains". En ligne. <http://www.iana.org/domains/reserved>. Consulté le 17 janvier 2016.
9. Internet Systems Consortium. 2016. « BIND | Internet Systems Consortium". En ligne. <https://www.isc.org/downloads/bind/>. Consulté le 18 janvier 2016.
10. PowerDNS.COM BV. 2016. "Welcome to PowerDNS". En ligne. <https://www.powerdns.com>. Consulté le 18 janvier 2016.
11. Stichting NLnet Labs. 2016. "nlnetlabs.nl:: Name Server Daemon (NSD)::". En ligne. <http://www.nlnetlabs.nl/projects/nsd/>. Consulté le 20 janvier 2016.
12. Geoffrey G. Filippi (2008). A High-Availability Architecture for the Dynamic Domain Name System. Virginia Polytechnic Institute and State University. Thesis. En ligne. <http://scholar.lib.vt.edu/theses/available/etd-05162008-160840/unrestricted/Filippi.Thesis.pdf>. Consulté le 16 décembre 2015.
13. Internet Engineering Task Force (IETF), Information Sciences Institute University of Southern California. 2016. "INTERNET PROTOCOL". En ligne. <https://tools.ietf.org/html/rfc791>. Consulté le 21 janvier 2016.
14. Wikimedia Foundation, Inc. 2016. "Representational State Transfer—Wikipedia, the free encyclopedia". En ligne.

<https://en.wikipedia.org/wiki/Representational_state_transfer>. Consulté le 21 janvier 2016.

15. Wikimedia Foundation, Inc. 2016. "Plan de numérotation nord-américain—Wikipedia, the free encyclopedia". En ligne. <https://fr.wikipedia.org/wiki/Plan_de_num%C3%A9rotation_nord-am%C3%A9ricain>. Consulté le 21 janvier 2016.
16. The Internet Assigned Numbers Authority (IANA). 2016. "Root Server Technical Operations Assn". En ligne. <http://www.root-servers.org/>. Consulté le 21 janvier 2016.
17. South Asian Network Operators Group (SANOG). 2016. « Scaling DNS ». En ligne. <http://www.sanog.org/resources/sanog14/sanog14-devdas-dns-scalability.pdf>. Consulté le 25 janvier 2016.
18. Jan-Piet Mens. 2013. [Blogue]. "Jan-Piet Mens:: Automatic provisioning of slave DNS servers". En ligne. <http://jpmens.net/2013/02/13/automatic-provisioning-of-slave-dns-servers/>. Consulté le 25 janvier 2016.
19. Wikimedia Foundation, Inc. 2016. "Publish-subscribe—Wikipedia, the free encyclopedia". En ligne. <https://fr.wikipedia.org/wiki/PubSub>. Consulté le 27 janvier 2016.
20. Jan-Piet Mens (2009). [Blogue]. Choice and deployment, and optional SQL/LDAP back-ends. UIT CAMBRIDGE LTD. CAMBRIDGE, ENGLAND. En ligne. <http://jpmens.net/2010/10/29/alternative-dns-servers-the-book-as-pdf/>. Consulté le 27 janvier 2016.
21. Wikimedia Foundation, Inc. 2016. "NSD—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/NSD>. Consulté le 27 janvier 2016.
22. PowerDNS.COM BV. 2016. « Introduction ». En ligne. <https://doc.powerdns.com/md/authoritative/>. Consulté le 27 janvier 2016.
23. Wikimedia Foundation, Inc. 2016. "Comparison of DNS server software—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Comparison_of_DNS_server_software>. Consulté le 14 janvier 2016.
24. ZyTrax, Inc., 2016. "Chapter 7 DNS BIND view Clause". En ligne. <http://www.zytrax.com/books/dns/ch7/view.html>. Consulté le 15 février 2016.
25. The OpenNMS Group, Inc., 2015. "The OpenNMS Project". En ligne. <http://opennms.org/>. Consulté le 26 février 2016.
26. The OpenNMS Group, Inc., 2015. « DNS Importing - OpenNMS ». En ligne. <https://www.opennms.org/wiki/DNS_Importing>. Consulté le 26 février 2016.
27. Internet Engineering Task Force (IETF), E. Lewis, NeuStar, Inc. 2016. "DNS Zone Transfer Protocol (AXFR)". En ligne. <https://tools.ietf.org/html/rfc5936>. Consulté le 26 février 2016.
28. Cricket Liu. Safari Books Online. 2016. "Running Multiple Primary Master Name Servers for the Same Zone—DNS & Bind Cookbook [Book]". En ligne. <https://www.safaribooksonline.com/library/view/dns-bind/0596004109/ch05s26.html>. Consulté le 26 février 2016.

29. Erik Rossen. 2008. "Using the dynamic DNS editor, nsupdate". En ligne. <http://www.rtfm-sarl.ch/articles/using-nsupdate.txt>. Consulté le 26 février 2016.
30. SonarSource S. A, Switzerland. 2008-2015. « SonarQube™ ». En ligne. <http://www.sonarqube.org/>. Consulté le 1er mars 2016.
31. Buzpark Bilisim Tarim Urunleri Sanayi Tic. Ltd. Sti. 2016. "Uptime Robot". En ligne. <https://uptimerobot.com/>. Consulté le 1er mars 2016.
32. Software in the Public Interest, Inc. 1997–2015. « Actualités Debian ». En ligne. <https://www.debian.org/>. Consulté le 1er mars 2016.
33. MariaDB Corporation Ab. 2016. "What is MariaDB Galera Cluster? —MariaDB Knowledge Base". En ligne. <https://mariadb.com/kb/en/mariadb/what-is-mariadb-galera-cluster/>. Consulté le 1er mars 2016.
34. Pivotal Software, Inc. 2007-2016. "RabbitMQ—Highly Available Queues". En ligne. <https://www.rabbitmq.com/ha.html>. Consulté le 1er mars 2016.
35. MongoDB, Inc. 2016. "MongoDB for GIANT Ideas | MongoDB". En ligne. <https://www.mongodb.com/>. Consulté le 1er mars 2016.
36. Wikimedia Foundation, Inc. 2016. "Create, read, update and delete—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Create, _read, _update_and_delete>. Consulté le 1er mars 2016.
37. Codership Ltd. 2014. « Galera Arbitrator - Galera Cluster Documentation ». En ligne. <http://galeracluster.com/documentation-webpages/arbitrator.html>. Consulté le 3 mars 2016.
38. No-IP.com. 2016. « 20 Million Thanks ! | No-IP Blog — Managed DNS Services ». En ligne. <http://www.noip.com/blog/2014/12/11/20-million-thanks/>. Consulté le 3 mars 2016.
39. Codership Oy. 2014. "Can't Be Any Faster Than That: A Real-Life Experiment with Latency in a Geo-distributed Environment | Galera Cluster for MySQL". En ligne. <http://galeracluster.com/2015/07/cant-be-any-faster-than-that-a-real-life-experiment-with-latency-in-a-geo-distributed-environment/>. Consulté le 3 mars 2016.
40. Wikimedia Foundation, Inc. 2016. "Scalability—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Scalability #Horizontal_and_vertical_scaling>. Consulté le 4 mars 2016.
41. Wikimedia Foundation, Inc. 2016. "Quorum_(distributed_computing)—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Quorum_ (distributed_computing)>. Consulté le 5 mars 2016.
42. Pivotal Software, Inc. 2016. "RabbitMQ Hits One Million Messages Per Second on Google Compute Engine | Pivotal P.O.V. ». En ligne. <https://blog.pivotal.io/pivotal/products/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine>. Consulté le 8 mars 2016.
43. Wikimedia Foundation, Inc. 2016. "Anycast—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Anycast>. Consulté le 8 mars 2016.
44. Wikimedia Foundation, Inc. 2016. "MongoDB—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/MongoDB>. Consulté le 23 mars 2016.

45. Tom Christie. 2011-2016. "Django REST framework". En ligne. <http://www.django-rest-framework.org/>. Consulté le 21 avril 2016.
46. Python Software Foundation. 1990–2015. "PyPI—the Python Package Index: Python Package Index". En ligne. <https://pypi.python.org/>. Consulté le 21 avril 2016.
47. SonarSource S. A, Switzerland. 2008-2015. « SonarQube™ ». En ligne. <http://www.sonarqube.org/>. Consulté le 5 juillet 2016.
48. The Open Group. 1995–2016. "TOGAF®, an Open Group standard | The Open Group". En ligne. <http://www.opengroup.org/subjectareas/enterprise/togaf>. Consulté le 5 juillet 2016.
49. Wikimedia Foundation, Inc. 2016. "Iterative design—Wikipedia, the free encyclopedia". En ligne. <https://en.wikipedia.org/wiki/Iterative_design/>. Consulté le 6 juillet 2016.
50. Kohsuke Kawaguchi, Sun Microsystems, Inc., and a number of other of contributors. 2004–2016. "Jenkins". En ligne. <https://jenkins.io/>. Consulté le 6 juillet 2016.
51. Red Hat, Inc. 2016. "Ansible is Simple IT Automation". En ligne. <https://www.ansible.com/>. Consulté le 6 juillet 2016.
52. NGINX Inc. 2016. "Announcing UDP Load Balancing in NGINX and NGINX Plus". En ligne. <https://www.nginx.com/blog/announcing-udp-load-balancing/>. Consulté le 7 juillet 2016.
53. Wikimedia Foundation, Inc. 2016. «Bureau d'enregistrement - Wikipedia, the free encyclopedia». En ligne. <https://fr.wikipedia.org/wiki/Bureau_d%27enregistrement>. Consulté le 11 juillet 2016.
54. Python Software Foundation. 2001–2016. "PEP 8 -- Style Guide for Python Code". En ligne. <https://www.python.org/dev/peps/pep-0008/>. Consulté le 21 juillet 2016.
55. Dynamic Network Services, Inc. 1998–2016. "Dyn Statement on 10/21/2016 DDoS Attack | Dyn Blog". En ligne. <http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/>. Consulté le 5 novembre 2016.

Glossaire

Nom	Définition
CRUD	De l'anglicisme "Create, Read, Update and Delete".
DNS	Serveur de nom de domaine.
DDNS	Serveur de nom de domaine dynamique.
FAI	Fournisseur d'accès Internet.
IP	Adresse sur le réseau Internet ou local.
SGBD	Système de gestion de base de données.
Zone	Une zone représente un nom de domaine, par exemple le nom de domaine example.com est une zone. Le sous-domaine www.example.com est un sous-domaine du nom de www de la zone example.com, qui est à ce moment un enregistrement de cette zone.