Mise à jour d'un registre de noms de domaines en temps réel

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 janvier 2007. Dernière mise à jour le 17 janvier 2007

https://www.bortzmeyer.org/registre-temps-reel.html

Traditionnellement, les registres de noms de domaines rechargeaient leur zone DNS une ou deux fois par jour. Désormais, plusieurs registres proposent des mises à jour presque en temps réel.

On peut douter de l'intérêt d'un tel service : j'avoue ne pas comprendre pourquoi on ne peut pas attendre six ou douze heures pour avoir son nom de domaine mais certains semblent estimer qu'ils mourront dans d'atroces souffrances si l'Internet ne s'adapte pas immédiatement à leurs désirs. Alors, voyons les techniques possibles (je ne sais pas comment fait .org et je vais me concentrer sur ce qui est réalisable avec du logiciel libre).

Il y en a deux : mettre à jour le serveur DNS dynamiquement ou bien faire s'appuyer le serveur sur une base de données qui sera, elle, dynamique. La première méthode pose le problème de la synchronisation entre la « vraie » base de données et celle qui est dans la mémoire du serveur de noms. La deuxième méthode fait dépendre les serveurs de noms, systèmes très critiques, d'un composant extérieur pas toujours fiable. La documentation de BIND-DLZ <http://bind-dlz.sourceforge.net/>, très détaillée, contient plusieurs exemples de bonnes et de mauvaises pratiques à ce sujet. Par exemple, il ne sert à rien d'avoir plusieurs serveurs de noms s'ils interrogent le même SGBD, qui sera alors un SPOF.

La première possibilité est donc d'utiliser les mises à jour dynamiques du DNS, décrites dans le RFC 2136[1]. Nominet le fait, depuis février 2005 <http://lists.nominet.org.uk/pipermail/nom-announce/2005-February/000147.html> et a fait une bonne présentation au RIPE-NCC <http://www.ripe.net/ripe/meetings/ripe-52/presentations/ripe52-dns-dynamic-updates.pdf>. On notera que cette présentation détaille les difficultés rencontrées, notamment pour assurer la

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc2136.txt

synchronisation de la base avec ce que servent les serveurs de noms. Nominet a utilisé Net : :DNS <http://www.net-dns.org/> pour développer le logiciel client.

L'autre solution repose sur la base de données.

On peut utiliser le traditionnel logiciel BIND qui, à partir de la version 9.4, inclus officiellement **DLZ** ("*Dynamically Loadable Zones*" <http://bind-dlz.sourceforge.net/>), système qui permet à BIND de s'appuyer sur une base de données.

L'autre serveur de noms en logiciel libre qui peut utiliser une base de données est PowerDNS. PowerDNS permet de choisir un **dorsal** (la partie du serveur qui parle à la base) et, parmi les dorsaux existants, certains sont des interfaces à une base de données.

Comparons la configuration d'un tel registre en utilisant une base PostgreSQL (version 8.1.3) avec BIND-DLZ et PowerDNS. Dans les deux cas, je recommande, pour la configuration initiale, d'activer l'enregistrement systématique des requêtes par le SGBD. Avec PostgreSQL, cela se fait dans le `postgresql.conf` :

```
log_connections = on
log_disconnections = on
log_duration = on
log_statement = 'all'
log_hostname = on
```

Les requêtes SQL envoyées par le serveur de noms apparaitront alors dans le journal du SGBD. Une fois les tests effectués, il vaut mieux débrayer cet enregistrement, très coûteux en performances et en place disque.

BIND-DLZ peut utiliser le modèle de données qu'on veut, puisque l'administrateur système définit les requêtes SQL à émettre. Seul l'ordre des données retournées est important, la façon dont elles ont été obtenues ne dépend pas de DLZ. Ici, j'ai testé avec un modèle de données proche de celui décrit dans la documentation <http://bind-dlz.sourceforge.net/postgresql_driver.html>. Je crée ma base et la peuple de quelques enregistrements ainsi :

```
CREATE TABLE DNS_Records (
  zone TEXT NOT NULL,
  host TEXT NOT NULL, -- Should be named "domain", probably
  ttl INTEGER DEFAULT 60,
  type TEXT NOT NULL CONSTRAINT Supported_types CHECK (type IN ('SOA', 'NS', 'A', 'AAAA', 'MX')),
  mx_priority INTEGER,
  contact TEXT,
  serial INTEGER,
  refresh INTEGER,
  retry INTEGER,
  expire INTEGER,
  minimum INTEGER,
  data TEXT NOT NULL
);

INSERT INTO DNS_Records (zone, host, type, data) VALUES ('example', 'ns1.nic',
                                                         'A', '192.0.2.1');
INSERT INTO DNS_Records (zone, host, type, data) VALUES ('example', 'ns1.nic',
                                                         'AAAA', '2001:DB8::1035:1');
INSERT INTO DNS_Records (zone, host, type, contact, serial, refresh, retry, expire, minimum,
                                 data) VALUES
                  ('example', '@', 'SOA', 'me.nic', 2007011710, 300, 60, 86400,
                           10, 'ns1.nic');
INSERT INTO DNS_Records (zone, host, type, mx_priority, data)
               VALUES ('example', '@',
                    'MX', 10, 'mail.mypostfix.com');
```

Notez que certains champs (comme mx_priority ou expire) n'ont de sens que pour certains types d'enregistrement DNS.

Avec cette base, je peux configurer BIND. DLZ va remplacer certains jetons comme %zone% par leur valeur au moment de la requête (si je demande des informations sur `www.bortzmeyer.org`, il essaiera avec les zones `bortzmeyer.org` et `org`). Le `named.conf` contient :

```
dlz "PostgreSQL zone" {
   database "postgres 2
   {dbname=registry-bind}
   {SELECT zone FROM dns_records WHERE zone = '%zone%'}
   {SELECT ttl, type, mx_priority,
               case WHEN lower(type)='txt' then '\"' || data || '\"' else data end
         FROM dns_records WHERE zone = '%zone%' AND host = '%record%'
                              AND NOT (type = 'SOA' OR type = 'NS')}
   {SELECT ttl, type, mx_priority, data, contact, serial, refresh, retry, expire,
        minimum FROM dns_records WHERE zone = '%zone%' and (type = 'SOA' or type='NS')}
   {SELECT ttl, type, mx_priority, data, contact, serial, refresh, retry, expire,
        minimum FROM dns_records WHERE zone = '%zone%'}
   {}";
};
```

- La première requête SQL est utilisée lorsque le serveur de noms veut savoir s'il sert cette zone.
- La seconde renvoie les données, si on sert cette zone. Les données « d'autorité » comme le SOA sont explicitement exclues.
- La troisième requête renvoie les données d'autorité.
- La quatrième renvoie tout (je suppose qu'elle sert pour les transferts de zone).
- La cinquième sert à autoriser les transferts de zone et je ne l'ai pas définie ici.

Une fois BIND lancé avec cette configuration, tout marche :

```
% dig @localhost AAAA ns1.nic.example.
...
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
...
;; ANSWER SECTION:
ns1.nic.example.        60      IN      AAAA    2001:db8::1035:1

% dig @localhost AAAA www.foobar.example.
...
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 52332
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
...
;; AUTHORITY SECTION:
example.                10      IN      SOA     ns1.nic.example. me.nic.example. 2007011710 300 60 86400 10
```

Merveille de la mise à jour en temps réel, si je fais un INSERT SQL, je vois le résultat de suite :

```
% dig +short @localhost AAAA www.foobar.example.
%

registry-bind=> INSERT INTO DNS_Records (zone, host, type, data)
            VALUES ('example', 'www.foobar', 'AAAA',
                    '2001:DB8::3330:128');
INSERT 0 1

% dig +short @localhost AAAA www.foobar.example.
2001:db8::3330:128
```

DLZ ne semble pas avoir de cache : si j'active le journal de PostgreSQL (`log_statement = 'all'`), je vois une requête SQL à chaque requête DNS, ce qui peut être lent.

Contrairement à DLZ, PowerDNS impose un modèle de données. Nous créeons donc la base comme indiqué dans la documentation <http://doc.powerdns.com/generic-mypgsql-backends.html#AEN4585> :

```
CREATE TABLE domains (
 id              SERIAL PRIMARY KEY,
 name            VARCHAR(255) NOT NULL,
 master          VARCHAR(20) DEFAULT NULL,
 last_check      INT DEFAULT NULL,
 type            VARCHAR(6) NOT NULL,
 notified_serial INT DEFAULT NULL,
 account         VARCHAR(40) DEFAULT NULL
);

CREATE TABLE records (
        id              SERIAL PRIMARY KEY,
        domain_id       INT DEFAULT NULL,
        name            VARCHAR(255) DEFAULT NULL,
        type            VARCHAR(6) DEFAULT NULL,
        content         VARCHAR(255) DEFAULT NULL,
        ttl             INT DEFAULT NULL,
        prio            INT DEFAULT NULL,
        change_date     INT DEFAULT NULL,
        CONSTRAINT domain_exists
        FOREIGN KEY(domain_id) REFERENCES domains(id)
        ON DELETE CASCADE
);
```

puis nous la peuplons :

```
INSERT INTO domains (name, type) VALUES ('example', 'NATIVE');

INSERT INTO records (domain_id, name, content, type,ttl,prio)
          VALUES (1,'example','ns1.nic.example me.nic.example
            2007011701', 'SOA',86400,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
          VALUES (1,'ns1.nic.example','192.0.2.1','A',120,NULL);
INSERT INTO records (domain_id, name, content, type,ttl,prio)
          VALUES (1,'ns1.nic.example','2001:DB8::1035:1','AAAA',120,NULL);
```

Il reste à configurer PowerDNS pour utiliser cette base :

```
launch=gpgsql
gpgsql-user=registry
gpgsql-dbname=registry-pdns
```

Nous pouvons alors faire les mêmes tests qu'avec BIND-DLZ et obtenir, logiquement, les mêmes résultats.

Contrairement à DLZ, PowerDNS a un cache des requêtes. En activant le journal de PostgreSQL, on voit que la base de données n'est pas appelée systématiquement.

Les vrais essais comparatifs de performance <https://www.bortzmeyer.org/performances-serveur-dn
html> sont prévus mais pas encore réalisés.