RFC 8040 : RESTCONF Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 février 2017

Date de publication du RFC : Janvier 2017

https://www.bortzmeyer.org/8040.html

Pour configurer à distance un équipement réseau (par exemple un routeur ou bien un commutateur), il existait déjà le protocole NETCONF (RFC 6241 [1]). Fondé sur un échange de données en XML, il n'a pas convaincu tout le monde, qui aurait préféré un protocole REST. C'est désormais fait avec RESTCONF, décrit dans ce RFC.

RESTCONF est donc bâti sur HTTP (RFC 7230) et les opérations CRUD. Comme NETCONF, RESTCONF utilise des modèles de données décrits en YANG (RFC 7950). Il devrait donc permettre de configurer plus simplement les équipements réseau. (Les historiens noteront que l'ancêtre SNMP avait été prévu, non seulement pour **lire** des informations, mais également pour **écrire**, par exemple pour modifier la configuration d'un routeur. Cette possibilité n'a eu aucun succès. Cet échec est une des raisons pour lesquelles NETCONF a été développé.)

Les opérations CRUD sont donc faites avec les méthodes classiques de HTTP (RFC 7231). Lire l'état de l'engin se fait évidemment avec la méthode `GET`. Le modifier se fait avec une des méthodes parmi `PUT`, `DELETE`, `POST` ou `PATCH` (cf. section 4 du RFC). Les données envoyées, ou bien lues, sont encodées, au choix, en XML ou en JSON. À noter que RESTCONF, conçu pour les cas relativement simples, est plus limité que NETCONF (il n'a pas de fonction de verrou, par exemple). Il ne peut donc pas complètement remplacer NETCONF (ainsi, si un client NETCONF a mis un verrou sur une ressource, le client RESTCONF ne peut pas y accéder, voir la section 1.4 du RFC). Mais le RFC fait quand même plus de 130 pages, avec plein d'options.

La syntaxe des URL utilisés comme paramètres de ces méthodes est spécifiée dans ce RFC en utilisant le langage de gabarit du RFC 6570. Ainsi, dans ce langage, les URL de Wikipédia sont décrits par

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6241.txt`

http://fr.wikipedia.org/wiki/{topic} ce qui signifie « le préfixe http://fr.wikipedia.org/wiki/ suivi d'une variable (qui est le sujet de la page) ».

En connaissant le modèle de données YANG du serveur, le client peut ainsi générer les requêtes REST nécessaires.

Les exemples du RFC utilisent presque tous la configuration d'un... juke-box (le module YANG est décrit dans l'annexe A). Cet engin a une fonction « jouer un morceau » et voici un exemple de requête et réponse REST encodée en JSON où le client demande au serveur ce qu'il sait faire :

```
GET /top/restconf/operations HTTP/1.1
Host: example.com
Accept: application/yang-data+json

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Sun, 22 Apr 2016 01:00:14 GMT
Content-Type: application/yang-data+json

{ "operations" : { "example-jukebox:play" : [null] } }
```

operations est défini (section 3.3.2) comme {+restconf}/operations dans le langage des gabarits du RFC 6570. Cela veut dire (section 3.2.3 du RFC 6570) « Remplacez la variable restconf par sa valeur, puis ajoutez /operations. La variable restconf est la racine du serveur, ici /top/restconf. (On peut la découvrir avec un fichier XRD en /.well-known/host-meta - RFC 6415.)

L'opération play, elle, est décrite dans l'annexe A, le modèle de données du juke-box :

```
rpc play {
        description "Control function for the jukebox player";
        input {
          leaf playlist {
            type string;
            mandatory true;
            description "playlist name";
            }
  leaf song-number {
            type uint32;
            mandatory true;
  ...
```

Autre exemple, où on essaie de redémarrer le juke-box (et, cette fois, on encode en XML pour montrer la différence avec JSON - notez les différents types MIME, comme application/yang-data+xml ou application/yang-data+json, et le fait que la racine est /restconf et plus /top/restconf) :

```
POST /restconf/operations/example-ops:reboot HTTP/1.1
Host: example.com
Content-Type: application/yang-data+xml

<input xmlns="https://example.com/ns/example-ops">
```

```
    <delay>600</delay>
    <message>Going down for system maintenance</message>
    <language>en-US</language>
</input>

HTTP/1.1 204 No Content
Date: Mon, 25 Apr 2016 11:01:00 GMT
Server: example-server
```

(Le serveur a accepté - le code de retour commence par 2 - mais n'a rien à dire, d'où le corps de la réponse vide, et le code 204 au lieu de 200.)

Et voici un exemple de récupération d'informations avec `GET` :

```
GET /restconf/data/example-jukebox:jukebox/library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
Accept: application/yang-data+xml

HTTP/1.1 200 OK
Date: Mon, 23 Apr 2016 17:02:40 GMT
Server: example-server
Content-Type: application/yang-data+xml
Cache-Control: no-cache
ETag: "a74eefc993a2b"
Last-Modified: Mon, 23 Apr 2016 11:02:14 GMT

<album xmlns="http://example.com/ns/example-jukebox"
             xmlns:jbox="http://example.com/ns/example-jukebox">
        <name>Wasting Light</name>
        <genre>jbox:alternative</genre>
        <year>2011</year>
</album>
```

Pour éviter les collisions d'édition (Alice lit une variable, tente de l'incrémenter, Bob tente de la multiplier par deux, qui va gagner ?), RESTCONF utilise les mécanismes classiques de HTTP, `If-Unmodified-Since:`, `If-Match:`, etc.

Allez, encore un exemple, je trouve qu'on comprend mieux avec des exemples, celui-ci est pour modifier une variable :

```
PUT /restconf/data/example-jukebox:jukebox/library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json

{
        "example-jukebox:album" : [
          {
            "name" : "Wasting Light",
            "genre" : "example-jukebox:alternative",
            "year" : 2011
          }
        ]
}
```

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:04:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:04:00 GMT
ETag: "b27480aeda4c"
```

Et voici un exemple utilisant la méthode `PATCH`, qui avait été introduite dans le RFC 5789, pour changer l'année de l'album :

```
PATCH /restconf/data/example-jukebox:jukebox/library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com
If-Match: "b8389233a4c"
Content-Type: application/yang-data+xml

<album xmlns="http://example.com/ns/example-jukebox">
       <year>2011</year>
</album>

HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2016 17:49:30 GMT
ETag: "b2788923da4c"
```

Et on peut naturellement détruire une ressource :

```
DELETE /restconf/data/example-jukebox:jukebox/library/artist=Foo%20Fighters/album=Wasting%20Light HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2016 17:49:40 GMT
Server: example-server
```

L'URL utilisé dans la requête comprend les différentes parties habituelles d'un URL (RFC 3986) : le chemin, la requête, l'identificateur, etc. (Dans les exemples précédents, je n'ai pas mis de requête ou d'identificateur.)

Notre RFC impose HTTPS, puisque modifier la configuration d'un équipement réseau est évidemment une opération sensible. Pour la même raison, le serveur RESTCONF doit évidemment authentifier le client. La méthode recommandée est le certificat client (section 7.4.6 du RFC 5246) mais on peut aussi utiliser une autre méthode d'authentification HTTP <https://www.iana.org/assignments/http-authschemes/http-authschemes.xml>.