Conteneurs Apache, .htaccess et équivalents Nginx

Oh, yes !

Le dixième docteur

On rappelle que les directives d'Apache, permettant de configurer ce serveur, peuvent être déclarées dans 4 contextes différents : de configuration globale, hôte virtuel, répertoire et .htaccess.

Un index des directives expliquant le détail de leur fonctionnement, ainsi qu'une liste des nombreux modules existants sont disponibles sur le site http://httpd.apache.org/docs.

Pour Nginx, la documentation se trouve sur http://nginx.org/en/docs/.

1 Les directives répertoire

1.1 Le conteneur `<Directory>`

Le conteneur `<Directory>` contient des directives qui s'appliquent à un répertoire ainsi qu'à ses sous-répertoires et aux fichiers situés dans ces sous-répertoires. Ce répertoire doit être désigné par son *chemin absolu*, éventuellement en utilisant le caractère « * », désignant une suite quelconque de caractères et d'autres caractères du *globbing*[1] : « ? » et les intervalles de caractères « [a-z] ».

Voici un exemple de définition :

```
<Directory /var/www/*/public_html>
    Options Indexes
</Directory>
```

Le chemin `/var/www/*/public_html` cible les dossiers nommés `public_html` appartenant aux répertoires présents dans le répertoire `/var/www/` : `/var/www/foo/public_html/`, mais pas `/var/www/bar/baz/public_html/`, par exemple.

Lorsque plusieurs définitions de conteneurs `<Directory>` peuvent s'appliquer à un répertoire donné, ces définitions sont évaluées de celle ayant la portée la plus large à celle ayant la portée la plus réduite, c'est à dire que la définition la plus spécifique surchargera la moins spécifique. Les différentes directives contenues dans ces conteneurs peuvent donc se cumuler, voire être surchargées au fil des conteneurs.

```
<Directory /var/www/html/>
    # Définition plus spécifique
</Directory>
<Directory /var/www/>
    # Définition large
</Directory>
```

Dans cet exemple, lors d'une requête qui ira chercher le fichier `/var/www/html/index.html`, bien que la définition plus large `/var/www/` soit en deuxième position, celle-ci sera évaluée en premier, puis ce sera au tour de `/var/www/html/`.

Si le caractère générique « * » n'est pas assez sophistiqué pour désigner en une seule expression une liste de répertoire, il est également possible d'utiliser des expressions régulières[2] pour décrire les noms de

1. https://en.wikipedia.org/wiki/Glob_(programming)
2. également appellées *expressions rationnelles*

répertoires concernés. La syntaxe de la première ligne devient alors : `<Directory ~ "regex">`. L'utilisation de cette syntaxe est possible, mais on lui préfère en pratique l'utilisation de la directive qui est conçue spécialement pour cet usage.

Équivalent dans Nginx

Pas de directive équivalente.

1.2 Le conteneur `<DirectoryMatch>`

Ce type de conteneur fonctionne exactement comme le conteneur `<Directory ~ "chemin">` et demande donc une expression régulière comme paramètre décrivant les répertoires concernés. Les principaux caractères permettant de décrire ces expressions régulières sont décrits ci-dessous :

- `*` : répète de 0 à n fois le motif précédent ce caractère ;
- `+` : répète de 1 à n fois le motif précédent ce caractère ;
- `?` : rend facultatif le motif précédent ce caractère (on parle aussi de répétition de 0 à 1 fois) ;
- `{n}` : répète exactement n fois le motif précédent ;
- `^` : désigne le début d'un chemin ;
- `$` : désigne la fin d'un chemin ;
- `[abc]` : exprime un choix à faire parmi les caractères `a`, `b` ou `c` ;
- `[a-z]` : exprime un choix à faire parmi les minuscules ;
- `[a-zA-Z0-9]` : exprime un choix à faire parmi les caractères alphanumériques ;
- `[^def]` : tous les caractères sauf les caractères `d`, `e` et `f` ;
- `.` : correspond à n'importe quel caractère ;
- `\.` : correspond au caractère « . ».

Ainsi les exemples suivants correspondent à :

- `"users"` : tous les chemins et fichiers qui contiennent la chaîne `users` (par ex : `/var/www/truc/users`, `/var/www/ihateusers`, `/var/www/usersarenotadmin`, `/home/user/truc`) ;
- `"^/users"` : tous les chemins commençant par `/users` ;
- `users[0-9]{4}` : tous les chemins contenant la chaîne `users` suivie de 4 chiffres.

Les directives des conteneurs `<DirectoryMatch>` et `<Directory ~>` sont appliquées après celles des conteneurs `<Directory>` et celles des fichiers `.htaccess`.

NB : il n'est pas possible d'utiliser les conteneurs `<Directory>` et `<DirectoryMatch>` dans un `.htaccess`.

Équivalent dans Nginx

Pas de directive équivalente.

1.3 Les conteneurs `<Files>` et `<FilesMatch>`

Ce type de conteneur fonctionne comme les conteneurs `<Directory>`, mais s'applique cette fois uniquement à des fichiers. Les noms des répertoires contenant ces fichiers ne sont pas considérés dans le paramètre fourni au conteneur `<Files>`.

Un exemple s'appliquant à tous les fichiers XML :

```
<Files *.xml>
    ...
</Files>
```

Tout comme pour `<Directory>`, on peut utiliser une expression régulière en ajoutant le caractère `~` : `<Files ~ "\.(gif|jpe?g|png)$">`.

La variante `<FilesMatch>` fonctionne exactement comme le conteneur `<Files ~>`, à l'instar de `<DirectoryMatch>` pour `<Directory>`.

Les fichiers correspondant à ces conteneurs sont les fichiers qui seraient normalement retournés par le serveur. Ainsi, `<Files index.html>` s'appliquera, que l'on demande `/` ou `/index.html` (pour peu que la directive `DirectoryIndex` soit positionnée pour servir `index.html` lorsqu'on donne l'adresse correspondant à un répertoire).

`<Files>` et `<FilesMatch>` peuvent être utilisés dans un `.htaccess`.

Les directives des conteneurs `<Files>` et `<FilesMatch>` seront évaluées en même temps mais ne se surchargeront pas comme `<Directory>` mais dans l'ordre dans lequel les conteneurs sont placés dans les fichiers de configuration (évaluation des directives du 1er conteneur qui correspond, puis surcharge éventuelles avec celles du 2ème conteneur correspondant).

Notez que les sections `<Files>` peuvent être imbriquées dans les sections `<Directory>` afin de restreindre la portion du système de fichiers à laquelle ces dernières vont s'appliquer.

Équivalent dans Nginx

Pas de directive équivalente.

1.4 Les conteneurs `<Location>` et `<LocationMatch>`

Ces conteneurs fonctionnent selon les mêmes principes que les autres conteneurs, mais s'appliquent uniquement sur les URL demandées au serveur, *indépendamment* de la localisation physique de ces ressources (ces directives *ne considèrent pas* l'arborescence de fichiers).

Un exemple typique d'utilisation :

```
<Location /status>
    SetHandler server-status
    Require ip 127.0.0.1
</Location>
```

Les URL définies en paramètres correspondent aux caractères situés après la base de l'URL (en considérant que la base est constituée du protocole et du nom de domaine, comme `http://www.example.org` par exemple).

On peut utiliser le *globbing* comme pour les conteneurs vus plus haut.
On peut utiliser une expression régulière en ajoutant le caractère `~` : `<Location ~ "/(extra|special)/data">`.

La variante `<LocationMatch>` fonctionne exactement comme le conteneur `<Location ~>`.

Les directives des conteneurs `<Location>` et `<LocationMatch>` seront évaluées en même temps mais ne se surchargeront pas comme `<Directory>` mais dans l'ordre dans lequel les conteneurs sont placés dans les fichiers de configuration (évaluation des directives du 1er conteneur qui correspond, puis surcharge éventuelles avec celles du 2ème conteneur correspondant), comme pour les conteneurs `<Files>` et `<FilesMatch>`.

Par exemple, une requête pour `/foo/bar` correspondra à `<Location "/foo/bar">` et `<Location "/foo">` : les deux sections seront évaluées mais selon l'ordre dans lequel elles apparaissent dans le fichier de configuration.

NB : il n'est pas possible d'utiliser ces conteneurs dans un `.htaccess`.

Équivalent dans Nginx

L'équivalent Nginx est sans surprise le contexte `location`.

```
location /foo {
    ...
}
```

Il fonctionne cependant différemment.

En effet, `location` peut être utilisée avec un *modifier*, placé juste après la directive (et donc avant le chemin de l'URL, appelé ici préfixe), qui modifie son comportement :

- `~` va utiliser le chemin comme une expression régulière, en prenant en compte la casse ;
- `~*` fera de même, mais sans tenir compte de la casse ;
- `=` ne prendra en compte que le chemin exact (`location = /foo/` ne fonctionnera que pour `/foo/`, pas pour `/foo/index.html`) ;
- `^~` n'utilise pas d'expression régulière, nous verrons son utilité après.

Ordre d'interprétation des contextes `location` :

- `=` tout d'abord, les correspondances exactes. Si une d'entre elles est trouvée, Nginx arrête l'interprétation et utilise ce bloc ;
- les autres `location` à l'exception de ceux utilisant les expressions régulières (donc sans *modifier* ou avec le *modifier* `^~`). Si le bloc correspondant le mieux (eg. le plus précis) possède le *modifier* `^~`, Nginx arrête l'interprétation et utilise ce bloc ;
- `~` et `~*` sont interprétés. Le premier (dans l'ordre dans lequel ils sont écrits dans le fichier de configuration) qui correspond est utilisé et Nginx arrête l'interprétation ;
- enfin, si aucun des `location` avec expression régulière ne convient, c'est celui avec le préfixe correspondant le mieux qui s'appliquera.

On remarquera que si les contextes `location` ne se surchargent pas, au contraire des `<Location>` d'Apache, ils peuvent être imbriqués :

```
location /a {
    # On peut mettre ici des directives qui
    # seront héritées par les autres locations
    location /a {
        ...
    }
    location /a/b {
        ...
    }
}
```

À noter : si le `location` de plus haut niveau utilise des regex, les `location` imbriqués devront eux aussi utiliser des regex.

1.5 Le conteneur `<If>`

Ce conteneur contient des directives appliquées si une condition est satisfaite au cours du traitement d'une requête.

On pourra par exemple évaluer des en-têtes de la requête, vérifier que la requête est sécurisée (`HTTPS`)...

La page https://httpd.apache.org/docs/2.4/fr/expr.html contient la documentation des conditions qu'il est possible de tester.

Par exemple, pour activer une directive si la requête est sécurisée :

```
<If %{HTTPS} == 'on'>
    ...
</If>
```

On peut placer `<If>` dans tous les contextes (configuration globale, serveur virtuel, répertoire, .htaccess).

Pour faire des « *else* » et des « *else if* », on utilise les conteneurs `<Else>` et `<ElsIf>` :

```
<If "-R '10.1.0.0/16'">
  #...
</If>
<ElseIf "-R '10.0.0.0/8'">
  #...
</ElseIf>
<Else>
  #...
</Else>
```

Équivalent dans Nginx

L'équivalent Nginx est le contexte `if`.

```
if ($https = 'on') {
    ...
}
```

On peut utiliser `if` dans les contextes `server` et `location` mais il n'existe pas de système de « *else* ».

Attention ! L'utilisation de `if` dans un contexte `location` peut avoir des effets de bords insoupçonnés et devrait être évité ! La page https://www.nginx.com/resources/wiki/start/topics/depth/ifisevil/ donne des exemples de comportements inattendus.

2 Le fichier `.htaccess`

Les directives décrites dans le fichier `apache2.conf`, ainsi que les fichiers qui y sont intégrés (via des `Include`) stipulent des règles permettant une configuration totale d'Apache. Cependant, il est également possible de configurer localement un répertoire géré par Apache au moyen d'un fichier, appelé `.htaccess`, situé dans ce répertoire. Ce mécanisme présente quelques avantages :

- Les directives associées à un répertoire (c-à-d ce qu'on peut mettre dans un conteneur `<Directory>`) peuvent être totalement décrites dans le fichier `.htaccess` associé, ce qui permet alors d'en avoir une vision globale. Si seules quelques directives y sont décrites, les autres directives décrites dans le fichier `apache2.conf` sont alors considérées.
- Un administrateur Apache peut donner un droit de modification à un fichier `.htaccess` à un utilisateur non administrateur. Celui-ci a alors la possibilité de modifier les directives qui le concerne, sans avoir à modifier le fichier général de configuration `apache2.conf`.
- Les directives contenues dans un fichier `.htaccess` sont automatiquement prises en compte, pour toute requête concernant le répertoire correspondant, sans avoir à relancer Apache.

Par défaut, Apache cherche systématiquement le fichier `.htaccess` dans chaque répertoire concerné par une requête. Si ce fichier est trouvé, les directives qu'il contient sont fusionnées avec les directives globales pour déterminer comment répondre à la requête reçue.

Un administrateur peut cependant restreindre les possibilités de configuration disponibles dans un fichier `.htaccess` au moyen des directive `AllowOverride` [3] et `AllowOverrideList` [4] (**NB** : celles-ci ne peuvent être utilisée que dans les sections `<Directory>` définies sans expressions rationnelles).

La directive `AllowOverride` décrit quelles sont les types de directives autorisées à être surchargées dans un fichier `.htaccess`. Les valeurs suivantes peuvent être associées à cette directive :

- `All` : autorise toutes les surcharges dans un fichier `.htaccess` ;
- `AuthConfig` : autorise les directives d'autorisation ;
- `None` : interdit toutes les surcharges dans un fichier `.htaccess` ;

3. https://httpd.apache.org/docs/2.4/mod/core.html#allowoverride
4. https://httpd.apache.org/docs/2.4/mod/core.html#allowoverridelist

— D'autres types de directives sont disponibles dans la documentation.

La directive `AllowOverrideList` permet de décrire précisément les directives autorisées en prenant comme arguments leurs noms (et non des types de directives, qui en regroupent plusieurs). Cela permet d'être plus précis que `AllowOverride`.

Équivalent dans Nginx

Pas de `.htaccess` dans Nginx.

3 Ordre d'évaluation des différentes directives

3.1 Apache

Lorsque plusieurs conteneurs correspondent à une seule requête, des règles de priorité sont appliquées dans l'ordre suivant :

1. Les directives du contexte de configuration globale sont tout d'abord considérées.
2. Les conteneurs `<Directory>` sont ensuite examinés (sauf ceux utilisant des expressions régulières) dans l'ordre des portées décroissantes (du plus général au plus spécifique). Lorsque des fichiers `.htaccess` existent dans ces répertoires, les directives qu'ils contiennent, si elles peuvent être prises en compte (voir la directive `AllowOverride` plus haut), surchargent celles définies dans les conteneurs `<Directory>`.
3. Les conteneurs `<DirectoryMatch>` et `<Directory ~>`, contenant des expressions régulières sont ensuite examinées. Les directives qu'ils contiennent surchargent celles définies à l'étape 1 de cette liste.
4. Les conteneurs `<Files>` et `<FilesMatch>` sont ensuite examinés selon l'ordre d'apparition dans la configuration. Les directives qu'ils contiennent surchargent celles définies aux étapes précédentes de cette liste.
5. Les conteneurs `<Location>` et `<LocationMatch>` sont ensuite examinés selon l'ordre d'apparition dans la configuration. Les directives qu'ils contiennent surchargent celles définies aux étapes précédentes de cette liste.
6. Enfin, les conteneurs `<If>` sont examinés. Les directives qu'ils contiennent, si les conditions des `<If>` sont remplies, surchargent celles définies aux étapes précédentes de cette liste.

Quand la requête est servie par le module `mod_proxy` [5], le conteneur `<Proxy>` [6] prend la place du conteneur `<Directory>` dans l'ordre de traitement.

La documentation de tout ceci est sur https://httpd.apache.org/docs/2.4/fr/sections.html#merging.

3.2 Nginx

Voir la partie sur le contexte `location`.

4 Les conteneurs `<Limit>` et `<LimitExcept>`

Le conteneur `<Limit>` regroupe les directives qui ne s'appliquent qu'aux méthodes passées en paramètres parmi les méthodes suivantes : `GET`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `PATCH`, `PROPFIND`, `PROPPATCH`, `MKCOL`, `COPY`, `MOVE`, `LOCK`, et `UNLOCK` (Attention, le nom des méthodes est sensible à la casse). Le conteneur `<LimitExcept>` regroupe les directives qui s'appliquent à toutes les méthodes HTTP, sauf celles passées en paramètres.

5. https://httpd.apache.org/docs/2.4/fr/mod/mod_proxy.html
6. https://httpd.apache.org/docs/2.4/fr/mod/mod_proxy.html#proxy

Ces conteneurs sont à placer dans les conteneurs `<Directory>` et `<DirectoryMatch>`.

Voici un squelette d'exemple :

```
<Limit GET POST DELETE>
    ...
</Limit>
```

NB : lorsqu'on autorise la méthode `GET`, la méthode `HEAD` est automatiquement autorisée aussi.

Les conteneurs `<Limit>` et `<LimitExcept>` peuvent être placés dans les contextes répertoire et .htaccess.

Équivalent dans Nginx

Il n'y a pas de `<Limit>` dans Nginx, juste le contexte `limit_except`. Son fonctionnement est similaire à celui de `<LimitExcept>`. Ce contexte est à placer dans un contexte `location`.

```
limit_except GET POST {
    ...
}
```