CGI

> Brilliant. Take me to your leader.
> I've always wanted to say that !
>
> Le dixième docteur

Les serveurs Web ne sont pas seulement des serveurs capables de fournir des pages statiques, ils sont également à l'interface des utilisateurs et des langages de script et de programmation qui permettent, eux, de fournir des pages dynamiques.

1 CGI

NB : Nginx ne supporte pas le CGI.

CGI (*Common Gateway Interface*) est une interface permettant à un serveur Web d'appeler un programme extérieur. L'objectif, outre l'exécution du programme en question, est alors de récupérer la sortie de ce programme et de la renvoyer comme réponse à une requête. Tout programme pouvant être exécuté en ligne de commande peut être utilisé avec CGI. Ce mécanisme offre quelques avantages : CGI est indépendant de l'architecture du serveur et des langages de programmation, ce qui en fait une solution *flexible*, et chaque programme CGI s'exécute dans un processus indépendant.

Le principe de fonctionnement est le suivant : CGI crée un processus permettant d'exécuter le programme considéré, tout en laissant connectée la sortie standard de ce processus au serveur Web.

C'est un mécanisme coûteux en ressources (le programme est lancé à chaque requête) et ancien. Il possède toutefois l'avantage de ne consommer aucune ressource lorsqu'aucune ressource n'est effectuée. Il n'est plus guère utilisé de nos jours, mais on le rencontre encore parfois.

Le programme correspondant peut être écrit en n'importe quel langage, compilé ou interprété : il suffit qu'il soit exécutable. Voici un exemple d'un tel programme écrit en shell :

```
#!/bin/sh
echo Content-type: text/plain
echo
env
```

Il existe ensuite plusieurs manières de configurer Apache pour qu'il puisse exécuter ce script.

1.1 Les directives `ScriptAlias` et `ScriptAliasMatch`

Vous aurez besoin du module `cgi` (ou du module `cgid`, à préférer quand on utilise un module MPM multi-threadé) : `a2enmod cgi && systemctl restart apache2`.

Les directives `ScriptAlias` [1] et `ScriptAliasMatch` permettent de désigner les *répertoires* pouvant contenir des exécutables. L'extrait suivant est un exemple de configuration de CGI avec la directive `ScriptAlias`. La directive `ScriptAliasMatch` fonctionne selon le même principe que la directive `ScriptAlias` mais avec des expressions régulières (voir cours précédents).

```
ScriptAlias /cgi-bin/ /var/www/cgi-bin/
```

1. https://httpd.apache.org/docs/2.4/fr/mod/mod_alias.html#scriptalias

1.2 La définition de fichiers au moyen de `AddHandler`

Pour définir les *extensions* des fichiers exécutables/interprétables, plutôt que les répertoires où se trouvent les programmes exécutables, il est possible d'utiliser la directive `AddHandler`[2]. Cette directive s'utilise de la manière suivante :

```
AddHandler *gestionnaire* *extension* [*autre extension*]
```

Le *gestionnaire* est la composante d'Apache qui va gérer les fichiers dont la liste d'*extensions* est fournie à la suite. Pour les scripts CGI, ce gestionnaire est nommé `cgi-script` (fourni par le module `cgi` ou `cgid`). Voilà ainsi un exemple de définition au moyen de cette directive (déclarant ainsi tous les fichiers à extension `.cgi` ou `.pl` comme étant des scripts exécutables) :

```
AddHandler cgi-script .cgi .pl
```

Notez que le dossier ou l'URL contenant les scripts doit disposer de l'option `ExecCGI` :

```
<Directory /var/www/html/>
    Options +ExecCGI
</Directory>
# Ou
<Location />
    Options +ExecCGI
</Location>
```

1.3 L'association de scripts à des types MIME

Enfin, il est également possible d'associer des scripts à des types MIME grâce à la directive `Action`[3]. Voici un exemple d'utilisation :

```
Action text/html /cgi-bin/index.sh
```

Toute requête demandant un résultat de type MIME `text/html` sera traitée par `/cgi-bin/test.cgi`.

NB : `/cgi-bin/test.cgi` est un chemin d'une URL, pas un chemin du système de fichiers, et cette URL doit correspondre à une ressource déclarée comme script CGI avec `ScriptAlias` ou `AddHandler` :

```
Action text/html /cgi-bin/index.sh
ScriptAlias /cgi-bin/ /var/www/cgi-bin/
```

1.4 Combinaison d'`AddHandler` et `Action`

On pourra combiner `AddHandler` et `Action` ainsi :

```
AddHandler foo-bar-baz .foo
Action foo-bar-baz "/cgi-bin/foo.cgi"
```

Ainsi, une requête pour un fichier dont l'extension est `.foo` sera traitée par `/cgi-bin/foo.cgi`.

Au lieu d'un *type MIME* en premier argument de la directive `Action`, on a utilisé un *gestionnaire*, gestionnaire défini avec `AddHandler`. Notez que cela ne dispense pas de déclarer `foo.cgi` comme script CGI avec `ScriptAlias` ou `AddHandler`.

2. https://httpd.apache.org/docs/2.4/fr/mod/mod_mime.html#addhandler
3. https://httpd.apache.org/docs/2.4/fr/mod/mod_actions.html#action

1.5 Exemples

Formulaire

La récupération des paramètres des formulaires est prévue par l'interface CGI. Dans le cas de passage de paramètres via un HTTP GET la norme définit les caractères `?` et `&` comme séparateurs respectivement du nom du script avec les paramètres et entre les paramètres.

Voici un exemple de script shell affichant la chaîne des différents arguments passés dans l'URL (de la forme `/cgi-bin/args.sh?argument1&argument2`).

```
#!/bin/sh
echo Content-type: text/html
echo
cat <<EOF
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html><head><title>View args CGI shell script</title></head>
<body><h1>HTTP GET argument string ?</h1><p>
EOF
[ ! -z "$QUERY_STRING" ] && echo "Got '$QUERY_STRING'" || echo "None :("
echo "</p></body></html>"
```

Scripts en Perl et en Ruby

Voici maintenant un exemple plus complet en `Perl`[4] (nécessite l'installation de `libcgi-pm-perl`).

```
#!/usr/bin/env perl
use warnings;
use strict;
use 5.10.0;

# load standard CGI routines
use CGI qw/:standard/;
# create the HTTP header
print header(-expires => '+3d'),
    # start the HTML
    start_html(
        -title => 'Sample echo env. Perl CGI',
        -dtd => 'html4'
    ),
    # level 1 header
    h1('Sample echo env. Perl CGI'),
    pre(
        join ("\n", map { "$_ --> ".escapeHTML($ENV{$_}) } sort keys %ENV)
    ),
    end_html;
```

Voici quand même le script en `Ruby`[5] (nécessite l'installation de `ruby`) :

```
#!/usr/bin/env ruby
require 'cgi'
```

4. cela devait initialement être en Ruby, mais faut pas déconner quand même !
5. bon sang, ce que c'est laid le Ruby tout de même

```
# Create an instance of CGI, with HTML 4 output
cgi = CGI.new("html4")
cgi.header('expires' => Time.now + (3 * 24 * 60 * 60))
# Send the following to the CGI object's output
cgi.out do
  cgi.html do
    cgi.head { cgi.title { "Sample echo env. Ruby CGI" } } + cgi.body do
      cgi.h1 { "Sample echo env. Ruby CGI" } + cgi.pre do
        ENV.collect do |key, value|
          key + " --> " + CGI.escapeHTML(value.chomp) + "\n"
end; end; end; end; end
```

2 FastCGI

L'exécution d'un programme CGI nécessite la création d'un nouveau processus à chaque requête, ce qui est souvent pénalisant au niveau du temps d'exécution. C'est pour répondre à ce problème que `FastCGI` a été développé. Il permet de conserver en mémoire un programme après une requête et ainsi d'améliorer les performances relatives aux autres requêtes destinées à ce programme.

Les modifications des programmes destinés à être utilisés par `FastCGI` sont généralement limitées : une simple ligne de code à ajouter dans la plupart des cas, dépendant du langage en question.

2.1 Apache

Pour Apache, le module idoine est `mod_fcgid`[6] (`libapache2-mod-fcgid` dans Debian) et pour Nginx, il s'agit de `ngx_http_fastcgi_module`[7] (fourni de base dans les paquets Debian).

Pour Apache, le fonctionnement est très proche de ce qu'on a vu pour le CGI. Par exemple :

```
AddHandler fcgid-script .fcgi
```

Attention : le dossier ou l'URL contenant les scripts doit, tout comme pour le CGI, disposer de l'option `ExecCGI` :

2.2 Nginx

Nginx a cette particularité de ne pas être capable de lancer lui-même un script fastcgi. Il faut donc le lancer pour lui avec, par exemple, le programme `spawn-fcgi` et le faire écouter sur un port ou un socket accessible à Nginx.

```
location ~ /munin/ {
    include fastcgi_params;
    fastcgi_split_path_info ^(/munin)(.*);
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_pass unix:/var/run/munin/fastcgi-html.sock;
}
```

Il existe de nombreuses directives pour l'utilisation du fastcgi dans Nginx. Elles sont généralement regroupées dans un fichier à part afin de pouvoir les réutiliser rapidement grâce à la directive `include`.

6. https://httpd.apache.org/mod_fcgid/mod/mod_fcgid.html
7. https://nginx.org/en/docs/http/ngx_http_fastcgi_module.html

Le fichier `fastcgi_params` inclus ci-dessus est généralement fourni par le paquet de `nginx` de votre distribution.

La lecture de l'article https://www.nginx.com/resources/wiki/start/topics/examples/fastcgiexample/ ainsi que de la documentation du module `ngx_http_fastcgi_module` vous aidera à utiliser fastcgi avec Nginx.

2.3 Exemple

Et pour finir un exemple de script FastCGI [8] :

En Perl :

```
#!/usr/bin/env perl
use warnings;
use strict;

use CGI::Fast qw(:standard);

sub print_env() {
    my ($title, $e) = @_;
    $title.'<br/><pre>'.join ("\n", map { "$_=$e->{$_}"} sort keys %{$e})."\n";
}

my %init_env = %ENV;
my $counter  = 0;

while (my $q = new CGI::Fast) {
    $counter++;
    print $q->header(),
          '<title>FastCGI echo</title><h1>FastCGI echo</h1>',
          sprintf("Request number %d, Process ID: %d<p>\n", $counter, $$),
          &print_env('Request environment', \%ENV),
          &print_env('Initial environment', \%init_env);
}
```

La version Ruby ne fonctionne plus, je vous l'épargne.

8. pensez à bien installer le module Apache kivabien, ainsi que les bibliothèques fastcgi des langages si vous voulez les tester