

# HTTP[s]

## Hyper Text Transfert Protocol

Apache2 ~ nginx ~ caddyserver

Pierre Bettens

mars 2024  
v0.2 ~ March 22, 2024

A close-up, low-angle shot of a woman's face, looking upwards and to the left. She is wearing a traditional Native American headdress with a band of colorful beadwork and a large, flowing feathered plume. Her eyes are heavily lined with dark makeup, and her lips are painted a deep red. The background is a dark, textured blue with some light blue speckles, resembling a night sky or a deep sea. The overall mood is dramatic and ethereal.

# HTTP(s)

*Hyper Text Transfert Protocol*  
*Apache 2 ~nginx*

C'est **le** protocole du WEB<sup>1</sup>

- Sur TCP
- Port 80 (par défaut)
- 1990 v0.9, 1996 v1.0

RFC

<https://www.rfc-editor.org/rfc/rfc9110.html>

---

<sup>1</sup>Du **WEB** (*World Wide Web*), pas de l'**Internet**

# HTTP

```
$ nc -C localhost 80
GET / HTTP/1.1
host: localhost
```

## Réponse...

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2023 11:02:18 GMT
Server: Apache/2.4.54 (Debian)
Last-Modified: Mon, 20 Jan 2020 09:21:02 GMT
ETag: "44-59c8ecf63eele"
Accept-Ranges: bytes
Content-Length: 68
Content-Type: text/html

<h1>Think about</h1>
<p>There are no failures, only lessons.</p>
```

# HTTP

```
$ nc -C example.org 80
GET / HTTP/1.1
host: example.org
```

## Extrait...

```
HTTP/1.1 200 OK
Age: 541560
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Wed, 01 Mar 2023 11:33:27 GMT
(cut)

<!doctype html>
(cut)
<div>
  <h1>Example Domain</h1>
  <p>This domain is for use in illustrative examples in documents. You may use this
  domain in literature without prior coordination or asking for permission.</p>
  <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

curl permet aussi d'exécuter simplement une requête http.

```
$ curl -v example.org
```

Extrait...

```
* Trying 93.184.216 34:80...
* Connected to example.org (93.184.216.34) port 80 (#0)
> GET / HTTP/1.1
> Host: example.org
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Age: 601738
< Cache-Control: max-age=604800
< Content-Type: text/html; charset=UTF-8
< Date: Fri, 01 Mar 2024 07:35:17 GMT
< Etag: "3147526947+ident"
< Expires: Fri, 08 Mar 2024 07:35:17 GMT
[cut]
```

## Structure de la requête HTTP

```
GET / HTTP/1.1  
host: example.org
```

- GET, la méthode
- /, la page demandée
- HTTP/1.1, le protocole
- host: example.org, précise le nom du site (cfr. *vhost*)

À partir de la deuxième ligne, field-name: token

## Les méthodes:

Method Name	Description	Section
GET	Transfer a current representation of the target resource.	<a href="#">9.3.1</a>
HEAD	Same as GET, but do not transfer the response content.	<a href="#">9.3.2</a>
POST	Perform resource-specific processing on the request content.	<a href="#">9.3.3</a>
PUT	Replace all current representations of the target resource with the request content.	<a href="#">9.3.4</a>
DELETE	Remove all current representations of the target resource.	<a href="#">9.3.5</a>
CONNECT	Establish a tunnel to the server identified by the target resource.	<a href="#">9.3.6</a>
OPTIONS	Describe the communication options for the target resource.	<a href="#">9.3.7</a>
TRACE	Perform a message loop-back test along the path to the target resource.	<a href="#">9.3.8</a>

Table 4

Figure 1: Extrait de rfc-editor.org



Les **entêtes**.

L'entête host

```
host: example.org
```

- précise quel est le nom de l'hôte
- obligatoire
- permet de distinguer les ressources lorsque le serveur gère plusieurs hôtes virtuels (*vhosts*)

Il existe beaucoup d'autres entêtes de requête (*request context fields*), par exemple

- referer: <value>, URI de la ressource d'origine
- user-agent: <value>, identifie le client
- accept-charset: <value>, jeu de caractères accepté
- ...

... et des entêtes de réponse (*response context fields*)

- server: <value>, le type de serveur HTTP
- content-type: <value>, le type de document échangé
- ...

La **réponse** HTTP est constituée de *context fields* et commence par un **code de retour** (*status code*).

Extrait d'une réponse

```
HTTP/1.1 200 OK  
Date: Wed, 01 Mar 2023 11:02:18 GMT  
Server: Apache/2.4.54 (Debian)
```

## Statut 200

OK Standard response for succesful HTTP requests

Les *status code* sont catégorisés, le premier chiffre décrit le type de réponse; succès, erreur...

Il y a 5 catégories :

- 1xx (Informational): The request was received, continuing process
- 2xx (Successful): The request was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request
- 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- 5xx (Server Error): The server failed to fulfill an apparently valid request

Quelques codes de retour (*status code*) fréquents :

- 200 OK
- 301 *Move permanently*
- 307 *Temporary redirect*
- 403 *Forbidden*
- 404 *Not found*
- 500 *Internal server error*
- 502 *Bad gateway*

Know your HTTP [pdf]

HTTP fournit un mécanisme de **cache**; des messages de réponses HTTP peuvent être sauvegardés par le navigateur (*private*) ou un proxy (*public*) sous certaines conditions.

Cache-Control est un en-tête HTTP qui dicte le comportement de mise en cache du navigateur.

## Exemple

```
Cache-Control: public, max-age=604800
```

Quelques entêtes :

- `max-age=[secondes]` — indique la quantité de temps maximale où la représentation sera considérée fraîche
- `s-maxage=[secondes]` — similaire à `max-age`, sauf qu'elle ne s'applique qu'aux caches partagés (*proxy*)
- `public` — marque les réponses authentifiées comme cachables
- `no-cache` — force à chaque fois les caches à soumettre la requête au serveur original
- `no-store` — instruit les caches de ne pas garder de copie de la représentation dans toutes les conditions
- ...

Cfr.

- <https://fr.wikipedia.org/wiki/Cache-Control>
- <https://developer.mozilla.org/fr/docs/Web/HTTP/Caching>
- [Whats the best way to set the cache control header](#)

## Exercice

Une page html, un serveur... illustrer la gestion du cache  
*via* son navigateur.



Un cookie HTTP est une donnée de petite taille envoyée par le serveur au navigateur web de l'utilisatrice ou de l'utilisateur.

Le navigateur peut alors enregistrer le cookie et le renvoyer au serveur lors des requêtes ultérieures.

Les cookies ont trois usages principaux :

- La gestion de session
- La personnalisation
- Le pistage

Les cookies sont actuellement « remplacés » par *localStorage*, *sessionstorage* et *IndexedDB*.

Cfr. <https://developer.mozilla.org/fr/docs/Web/HTTP/Cookies>

## Envoi d'un formulaire

Prénom :

Mot de passe :

**\*Loopback: lo**

Fichier Editer Vue Aller Capture Analyser Statistiques Telephone Wireless Outils Aide

http

No.	Time	Source	Destination	Protocol	Length	Info
10	9.188879067	127.0.0.1	127.0.0.1	HTTP	698	POST /dev.null HTTP/1.1 (application/x-www-form-urlencoded)
12	9.190532556	127.0.0.1	127.0.0.1	HTTP	554	HTTP/1.1 404 Not Found (text/html)

> Frame 10: 698 bytes on wire (5584 bits), 698 bytes captured (5584 bits) on interface lo, id 0  
 > Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
 > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 > Transmission Control Protocol, Src Port: 42330, Dst Port: 80, Seq: 1, Ack: 1, Len: 632  
 > Hypertext Transfer Protocol

> POST /dev.null HTTP/1.1\r\n
 Host: localhost\r\n
 User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/110.0\r\n
 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8\r\n
 Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3\r\n
 Accept-Encoding: gzip, deflate, br\r\n
 Content-Type: application/x-www-form-urlencoded\r\n
 Content-Length: 32\r\n
 Origin: http://localhost\r\n
 Connection: keep-alive\r\n
 Referer: http://localhost/form.html\r\n
 Upgrade-Insecure-Requests: 1\r\n
 Sec-Fetch-Dest: document\r\n
 Sec-Fetch-Mode: navigate\r\n
 Sec-Fetch-Site: same-origin\r\n
 Sec-Fetch-User: ?1\r\n
 \r\n
 [Full request URI: http://localhost/dev.null]  
 [HTTP request 1/1]  
 [Response in frame: 12]  
 File Data: 32 bytes

> HTML Form URL Encoded: application/x-www-form-urlencoded
 > Form item: "prenom" = "juste"
 > Form item: "password" = "mon secret"

```

02a0 3d 6a 75 73 74 65 26 70 61 73 73 77 6f 72 64 3d =juste&password=
02b0 6d 6f 6e 2b 73 65 63 72 65 74 mon+secret
  
```

Text item (text), 19 byte(s)

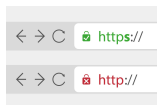
Paquets: 50 · Affichés: 2 (4.0%)

Profile: Default

A close-up, high-angle shot of a large, disorganized pile of various padlocks and keys. The padlocks are in many colors, including brass, silver, red, green, and blue. Some are combination locks, while others are traditional key locks. The keys are also visible, some attached to the padlocks and others lying loose. The background is a dense, chaotic mass of metal, suggesting a large quantity of these items. The lighting is warm, highlighting the metallic textures and colors.

[HTTP...S](https://)

## Qu'est-ce que HTTPS ?



HTTPS, c'est HTTP encapsulé dans TLS.

- Sur TCP encapsulé dans TLS
- Port 443 (par défaut)

La connection est sécurisée. Quelle **confiance** ?

- confiance dans le protocole (TLS)
- confiance dans le navigateur (Firefox, Chrome, Edge, Netscape...)
- confiance dans l'autorité de certification (CA)

## Fonctionnement

- ❶ Le client envoie une demande de connexion HTTPS au serveur.
  - Il présente les méthodes de chiffrement qu'il connaît.
- ❷ Le serveur répond en envoyant son **certificat** SSL/TLS au client après avoir choisi une des méthodes de chiffrement.
- ❸ Le client vérifie la validité du certificat SSL/TLS en vérifiant l'**autorité de certification** (CA) qui l'a émis.
- ❹ Si le certificat est valide, le client et le serveur établissent une connexion sécurisée.

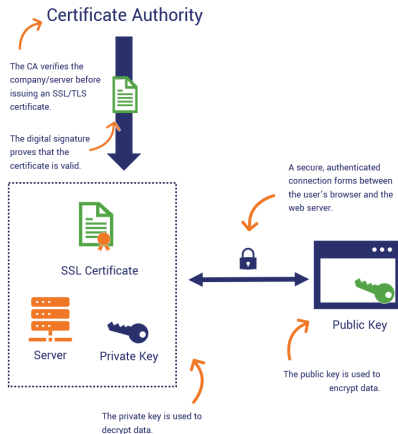
Navigateurs et validation de certificats

Qu'est-ce qu'une autorité de certification (CA) ?

## L'autorité de certification

### Les autorités de certification

- connues des navigateurs  
(*root CA*)
- certifie suivant
  - http
  - dns
  - mail
- il existe EV SSL  
(*extended validation*)



Source



```
$ openssl s_client -connect example.org:443
```

```
(cut)
```

```
GET / HTTP/1.1
```

```
Host: example.org
```

*Voir outssl*

# HTTPS

```
$ curl -v https://example.org
```

```
* Trying 93.184.216.34:443...
* Connected to example.org (93.184.216.34) port 443 (#0)
[cut]
* Server certificate:
* subject: [cut] CN=www.example.org
* start date: Jan 30 00:00:00 2024 GMT
* expire date: Mar 1 23:59:59 2025 GMT
* subjectAltName: host "example.org" matched cert's "example.org"
* issuer: C=US; O=DigiCert Inc; CN=DigiCert Global G2 TLS RSA SHA256 2020 CA1
```

## Obtention du certificat

- payer une autorité de certification
  - créer une CSR (*certificate signing request*) contenant l'identité et une PK (*primary key*)

```
openssl req -nodes -newkey rsa:2048 -sha256  
-keyout myserver.key -out server.csr
```

- le CA signe avec sa clé privée
- utiliser Let's Encrypt
  - certbot
  - dehydrated

**Serveur web**

Mise à disposition de l'information sur le réseau

- Apache2, nginx...
- HTML (CSS/JS)
- PHP, Ruby on rails, ASP...

<https://news.netcraft.com/archives/category/web-server-survey/>

# Apache 2

## Installation

- Paquets à installer : `apache2`, `libapache2-mod-php5...`

## Configuration

- Fichiers
  - `apache2.conf`, `conf-[available|enabled]`
  - `sites-[available|enabled]`, gestion des hôtes virtuels (voir plus loin)
  - `mods-[available|enabled]`, gestion des modules
- Commandes `a2en[conf|site|mod]`, `apache2ctl`

Les hôtes virtuels permettent à un serveur web de servir plusieurs sites





# Apache2 - Virtual host



Un fichier de configuration par site (*vhost*). Extrait :

```
ServerName      example.org
DocumentRoot    /var/www/html/org.example
ServerAdmin     webmaster@example.org
```

Que fait `a2ensite example.org` ?

## Exercice

Installer un serveur web apache et deux *vhosts* : l'un répondant à `one.example.org` et l'autre `two.example.org`.

## MPM - Modules multi-processus

- Apache fonctionne avec un essaim de *processus*
- Un module MPM, gère le nombre de *processus* et de *threads* lancés par Apache2
- Il existe plusieurs modules MPM: *prefork*, *event*, *worker*...
  - *prefork* utilise plusieurs processus;
  - *worker* et *event* utilisent plusieurs processus... et plusieurs *threads* / processus
- Sur un système d'exploitation moderne, ce sera *event*

Apache MPM

MPM - Modules multi-processus

Exemple

```
<IfModule mpm_event_module>
    StartServers          2
    MinSpareThreads       25
    MaxSpareThreads       75
    ThreadLimit           64
    ThreadsPerChild       25
    MaxRequestWorkers     150
    MaxConnectionsPerChild 0
</IfModule>
```

## Gestion du contrôle d'accès (*access control list*)

- spécifie les autorisations (*allow*) et les interdictions (*deny*) d'accès à une arborescence d'un serveur
- peut se faire dans la configuration de *vhost* ou dans un fichier `.htaccess`

Un fichier `.htaccess` est un extrait de configuration du serveur se trouvant dans un répertoire servi par le le serveur web.

```
AccessFileName .htaccess
```

L 'authentification dans HTTP peut se faire grâce aux ACL de différentes manières

- BASIC (RFC 2617)
- DIGEST (RFC 2617)
- SSL/TLS

Une authentification *basic* à travers une connexion TLS est sûre. Une alternative est l'utilisation de certificat pour une authentification *via* ces certificats (voir exemples).

## Exemple

```
Order deny, allow  
Deny from all  
Allow from 127.0.0.1/8
```

## Exemple 2

(Fichier `.htaccess` à placer dans un répertoire)

```
AuthUserFile access/.htaccess_passwd
AuthGroupFile /dev/null
AuthName "Accès membres"
AuthType Basic
```

```
<FilesMatch verysecretfile.html>
<LIMIT GET POST>
require valid-user
</LIMIT>
</FilesMatch>
```

```
<FilesMatch .htaccess>
<LIMIT GET POST>
deny from all
</LIMIT>
</FilesMatch>
```



## Exemple 3

```
SSLCACertificateFile "conf/ssl.crt/company-ca.crt"
```

```
<Directory "/usr/local/apache2/htdocs">
```

```
#  En dehors de subarea, seul l'accès depuis l'intranet est
```

```
#  autorisé
```

```
    Require             ip 192.168.1.0/24
```

```
</Directory>
```

```
<Directory "/usr/local/apache2/htdocs/subarea">
```

```
#  Dans subarea, tout accès depuis l'intranet est autorisé
```

```
#  mais depuis l'Internet, seul l'accès par HTTPS + chiffrement fort + Mot de passe
```

```
#  ou HTTPS + chiffrement fort + certificat client n'est autorisé.
```

```
#  Si HTTPS est utilisé, on s'assure que le niveau de chiffrement est fort.
```

```
#  Autorise en plus les certificats clients comme une alternative à
```

```
#  l'authentification basique.
```

```
SSLVerifyClient        optional
```

```
SSLVerifyDepth         1
```

```
SSLOptions              +FakeBasicAuth +StrictRequire
```

```
SSLRequire              %{SSL_CIPHER_USEKEYSIZE} >= 128
```

## Exemple 3 (suite)

```
#  ON oblige les clients venant d'Internet à utiliser HTTPS
```

```
RewriteEngine      on
RewriteCond        "%{REMOTE_ADDR}" "!^192\.168\.1\. [0-9]+$"
RewriteCond        "%{HTTPS}" "!=on"
RewriteRule        "." "-" [F]
```

```
#  On permet l'accès soit sur les critères réseaux, soit par authentication Basique
```

```
Satisfy            any
```

```
#  Contrôle d'accès réseau
```

```
Require            ip 192.168.1.0/24
```

```
#  Configuration de l'authentification HTTP Basique
```

```
AuthType           basic
AuthName           "Protected Intranet Area"
AuthBasicProvider   file
AuthUserFile        "conf/protected.passwd"
Require            valid-user
```

```
</Directory>
```

Source [https://httpd.apache.org/docs/2.4/ssl/ssl\\_howto.html](https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html)

Actuellement, l'authentification est plutôt **applicative**

- interface de *login* intégrée à l'application et pouvant être paramétrée
- authentification à travers TLS puis (bien) gérée par le *framework*
- peut-être déléguée (*SSO*)
- « facile » pour l'application de gérer ensuite la session
- ...

## Configurer Apache pour servir un site en HTTPS

- Se procurer les certificats
- Rendre le module `mod_ssl` disponible
- Configurer le *vhost* pour écouter sur le port 443.

## Extrait d'une configuration Apache

```
<IfModule mod_ssl.c>
  <VirtualHost _default_:443>
    ServerName example.org
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    SSLEngine on
    SSLCertificateFile /elsewhere/cert.pem
    SSLCertificateKeyFile /elsewhere/key.key
  </VirtualHost>
</IfModule>
```

nginx

## Installation

- Paquets à installer : `nginx`, `libnginx-mod-foo...`

## Configuration

- Fichiers
  - `nginx.conf`
  - `sites-[available|enabled]`, gestion des hôtes virtuels (*servers blocks*)
  - `modules-[available|enabled]`, gestion des modules

## freenginx

Le 14 fév. 2024, un *fork* est créé : *freenginx*.

Le but du projet est de conserver le développement de *nginx* libre.

Le premier *server block* est celui par défaut

```
http {  
    server {  
        root /data/www;  
    }  
}
```

- server détermine la configuration du *server block*
- root l'endroit où se trouvent les fichiers (/var/www est très bien aussi)



# Nginx - *Servers blocks*

```
http {  
    server {  
        location / {  
            root /data/www;  
        }  
        location /images {  
            root /data/images;  
        }  
    }  
}
```

- plusieurs *locations* peuvent être choisies pour un même *server block*

L'ajout d'une directive `server_name` permet de répondre pour un hôte virtuel

```
server {  
    listen      80;  
    server_name example.org;  
    root /data/www/org.example;  
    ; ...  
}
```

# Nginx - Servers blocks

```
server {  
    listen      80;  
    server_name _;  
    root /data/www/default;  
    ;...  
}  
  
server {  
    listen      80;  
    server_name example.org;  
    root /data/www/org.example;  
    ;...  
}
```

- `server_name _` est un *catch\_all*

Un simple *proxy* est un changement de *location*

```
server {  
    location / {  
        proxy_pass http://localhost:8080;  
    }  
}
```

## Exercice

Installer un serveur web nginx et deux *vhosts* : l'un répondant à `one.example.org` et l'autre `two.example.org`.

**caddy server**

Atelier de recherche

# **caddy web server**

- Dæmon caddy géré *comme d'habitude*
- Fichier de configuration JSON ou CaddyFile (/etc/caddy/CaddyFile)
- La configuration est visible *via* `curl localhost:2019/config/`
  - *Tips* `|jq` or `|python3 json.tools`
- *Multiple sites* définis dans *CaddyFile* (cfr. infra)

Caddy doc, getting started

Caddy doc, API tutorial



# Caddy server

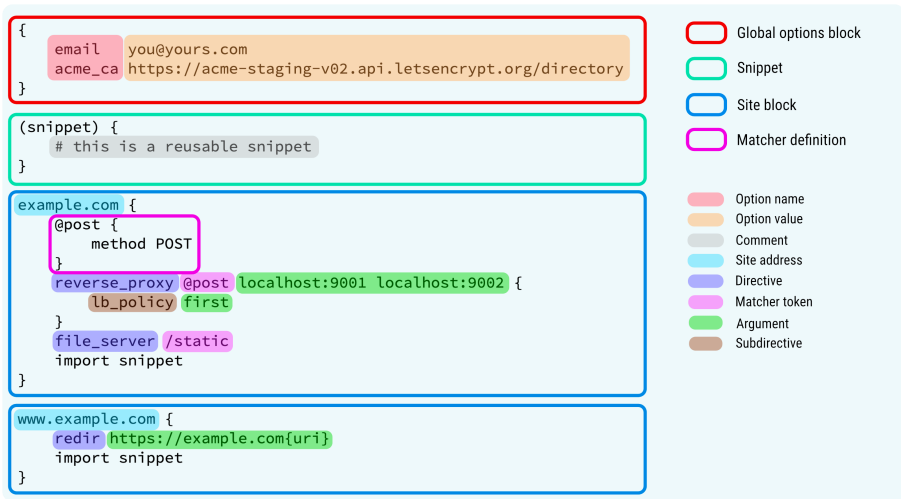


Figure 2: Source : Site web Caddy

## CaddyFile exemple

```
example1.com {  
    root * /www/example.com  
    file_server  
}  
  
example2.com {  
    reverse_proxy localhost:9000  
}
```

Vocabulaire : block, directives, *snippets*, adresses, *matchers*, *placeholders*

# Caddy server

Address	Effect
example.com	HTTPS with managed <a href="#">publicly-trusted certificate</a>
*.example.com	HTTPS with managed <a href="#">wildcard publicly-trusted certificate</a>
localhost	HTTPS with managed <a href="#">locally-trusted certificate</a>
http://	HTTP catch-all, affected by <a href="#">http_port</a>
https://	HTTPS catch-all, affected by <a href="#">https_port</a>
http://example.com	HTTP explicitly, with a <a href="#">Host</a> matcher
example.com:443	HTTPS due to matching the <a href="#">https_port</a> default
:443	HTTPS catch-all due to matching the <a href="#">https_port</a> default
:8080	HTTP on non-standard port, no <a href="#">Host</a> matcher
localhost:8080	HTTPS on non-standard port, due to having a valid domain
https://example.com:443	HTTPS, but both <a href="#">https://</a> and <a href="#">:443</a> are redundant
127.0.0.1	HTTPS, with a locally-trusted IP certificate

## Défi 1

Écrire dans le langage de son choix, un *client http* minimaliste.

Un programme qui affiche la page html demandée : `yaur1  
example.org/index.html`

## Défi 2

Écrire dans le langage de son choix, un *serveur http* (très) minimaliste.

Un programme qui écoute sur le port 80 et envoie la page html demandée.

Slides pour mes cours.



Pierre Bettens

bettensp@helha.be ~ <http://blog.namok.be>

## **Crédits**

Linux, pandoc, beamer,  $\text{\LaTeX}$