Serveurs Web
Open-Source pour Linux

Guide Complet

Installation - Configuration - Fonctionnalites - Licences

Ce document presente 8 serveurs web disponibles sur Linux, des plus populaires (Apache, Nginx) aux plus specialises (Caddy, Hiawatha). Pour chaque serveur : description, installation, configuration, commandes d'administration, trucs & astuces, licence et liens de reference. Un lexique des serveurs web est inclus en fin de document.

Table des matieres

1. Apache HTTP Server - Le standard historique

SYNTHESE

Apache HTTP Server (httpd) est le serveur web le plus deploye historiquement, developpe depuis 1995 par l'Apache Software Foundation. Architecture modulaire avec plus de 500 modules disponibles. Il supporte HTTP/1.1, HTTP/2, HTTPS, les virtual hosts, le rewriting d'URL, l'authentification, les CGI/FastCGI, et s'integre avec PHP, Python, Perl, Ruby. Il propulse encore environ 23% des sites web mondiaux.

LICENCE :

Apache License 2.0 - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu/Mint
sudo apt update && sudo apt install apache2

# Fedora/RHEL/Rocky/Alma
sudo dnf install httpd

# Arch Linux
sudo pacman -S apache

# openSUSE
sudo zypper install apache2

# Alpine
sudo apk add apache2

# Demarrer et activer au boot
sudo systemctl start apache2      # Debian/Ubuntu
sudo systemctl enable apache2
sudo systemctl start httpd         # RHEL/Fedora
sudo systemctl enable httpd

# Verifier l'installation
curl -I http://localhost
```

CE QUE L'ON PEUT FAIRE

-> Hebergement web : servir des sites statiques (HTML/CSS/JS) et dynamiques (PHP, Python, Perl)
-> Virtual Hosts : heberger des dizaines de sites sur un seul serveur (par nom ou IP)
-> HTTPS / SSL/TLS : chiffrement via mod_ssl, integration Let's Encrypt avec certbot
-> Reverse proxy : rediriger les requetes vers des serveurs backend (mod_proxy)
-> Load balancing : repartir la charge entre plusieurs backends (mod_proxy_balancer)
-> URL rewriting : regles de reecriture puissantes avec mod_rewrite (.htaccess)
-> Authentification : Basic, Digest, LDAP, base de donnees (mod_auth_*)
-> Compression : gzip/deflate pour reduire la taille des reponses (mod_deflate)
-> Cache : mise en cache des reponses pour accelerer la livraison (mod_cache)
-> WebDAV : partage de fichiers via le protocole WebDAV (mod_dav)

CONFIGURATION & COMMANDES

```
# === FICHIERS DE CONFIGURATION (Debian/Ubuntu) ===
# /etc/apache2/apache2.conf        -> Config principale
# /etc/apache2/ports.conf           -> Ports d'ecoute
# /etc/apache2/sites-available/     -> Virtual hosts disponibles
# /etc/apache2/sites-enabled/       -> Virtual hosts actives (liens)
# /etc/apache2/mods-available/      -> Modules disponibles
# /etc/apache2/mods-enabled/        -> Modules actives

# === FICHIERS DE CONFIGURATION (RHEL/Fedora) ===
# /etc/httpd/conf/httpd.conf        -> Config principale
# /etc/httpd/conf.d/                -> Virtual hosts et modules

# === COMMANDES D'ADMINISTRATION ===
sudo systemctl start apache2          # Demarrer
sudo systemctl stop apache2           # Arreter
sudo systemctl restart apache2        # Redemarrer
sudo systemctl reload apache2         # Recharger la config (sans coupure)
sudo systemctl status apache2         # Verifier l'etat
sudo apache2ctl configtest            # Tester la configuration
sudo apache2ctl -M                    # Lister les modules charges
sudo apache2ctl -S                    # Lister les virtual hosts

# Activer/desactiver un site
sudo a2ensite monsite.conf
sudo a2dissite monsite.conf

# Activer/desactiver un module
sudo a2enmod rewrite ssl proxy headers deflate
sudo a2dismod autoindex

# === EXEMPLE DE VIRTUAL HOST ===
# /etc/apache2/sites-available/monsite.conf
# <VirtualHost *:80>
#     ServerName monsite.com
#     ServerAlias www.monsite.com
#     DocumentRoot /var/www/monsite
#     ErrorLog ${APACHE_LOG_DIR}/monsite-error.log
#     CustomLog ${APACHE_LOG_DIR}/monsite-access.log combined
# </VirtualHost>

# HTTPS avec Let's Encrypt
sudo apt install certbot python3-certbot-apache
sudo certbot --apache -d monsite.com

# Logs
sudo tail -f /var/log/apache2/access.log
sudo tail -f /var/log/apache2/error.log
```

TRUCS & ASTUCES

-> .htaccess : fichier de config par repertoire (URL rewrite, auth, redirections) - pratique mais plus lent
-> mod_rewrite : regles de reecriture puissantes. Activer avec a2enmod rewrite
-> MPM : choisir le bon modele (prefork pour mod_php, event pour performance haute)
-> ServerTokens Prod : cacher la version Apache dans les en-tetes (securite)
-> KeepAlive On : garder les connexions ouvertes pour accelerer le chargement des pages
-> mod_security : WAF (Web Application Firewall) pour proteger contre les attaques
-> mod_pagespeed : module Google pour optimiser automatiquement les pages
-> Monitoring : mod_status avec ExtendedStatus On pour surveiller les performances
-> Combiner avec Nginx : utiliser Nginx en frontal (reverse proxy) et Apache en backend
-> certbot --apache : renouvellement automatique du certificat SSL Let's Encrypt

LIENS DE REFERENCE

Site officiel : https://httpd.apache.org/
Documentation : https://httpd.apache.org/docs/current/
Modules : https://httpd.apache.org/docs/current/mod/
GitHub : https://github.com/apache/httpd
Wiki : https://cwiki.apache.org/confluence/display/httpd/
Let's Encrypt : https://letsencrypt.org/

2. Nginx - Haute performance & reverse proxy

SYNTHESE

Nginx (prononce 'Engine-X') est un serveur web haute performance cree par Igor Sysoev en 2004. Architecture asynchrone et evenementielle, il gere jusqu'a 500 000 connexions simultanees avec une faible empreinte memoire. C'est aussi un reverse proxy, load balancer, cache HTTP, proxy mail (IMAP/POP3) et API gateway. Avec ~34% de part de marche, c'est le serveur web le plus utilise.

LICENCE :

BSD 2-Clause License - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu/Mint
sudo apt update && sudo apt install nginx

# Fedora/RHEL/Rocky/Alma
sudo dnf install nginx

# Arch Linux
sudo pacman -S nginx

# openSUSE
sudo zypper install nginx

# Alpine
sudo apk add nginx

# Depot officiel Nginx (dernieres versions)
# Debian/Ubuntu
curl -fsSL https://nginx.org/keys/nginx_signing.key | \
    sudo gpg --dearmor -o /usr/share/keyrings/nginx.gpg
echo "deb [signed-by=/usr/share/keyrings/nginx.gpg] \
    http://nginx.org/packages/ubuntu $(lsb_release -cs) nginx" | \
    sudo tee /etc/apt/sources.list.d/nginx.list
sudo apt update && sudo apt install nginx

# Docker
docker run -d -p 80:80 --name nginx nginx:latest

# Demarrer et activer
sudo systemctl start nginx && sudo systemctl enable nginx
curl -I http://localhost
```

CE QUE L'ON PEUT FAIRE

-> Serveur web statique : performances exceptionnelles pour HTML/CSS/JS/images
-> Reverse proxy : router les requetes vers des backends (Node.js, Python, PHP-FPM, etc.)
-> Load balancer : distribuer le trafic (round-robin, least connections, IP hash)
-> Cache HTTP : mettre en cache les reponses des backends pour accelerer la livraison
-> HTTPS / SSL/TLS : terminaison TLS haute performance avec HTTP/2
-> API Gateway : routage, rate limiting et authentification pour les microservices
-> Proxy mail : proxy IMAP/POP3/SMTP avec authentification
-> Streaming : diffusion video HLS et RTMP
-> Compression : gzip et brotli pour reduire la bande passante
-> WebSocket : proxy transparent des connexions WebSocket

CONFIGURATION & COMMANDES

```
# === FICHIERS DE CONFIGURATION ===
# /etc/nginx/nginx.conf                -> Config principale
# /etc/nginx/sites-available/          -> Server blocks disponibles
# /etc/nginx/sites-enabled/            -> Server blocks actives (liens)
# /etc/nginx/conf.d/                   -> Config additionnelle

# === COMMANDES D'ADMINISTRATION ===
sudo systemctl start nginx              # Demarrer
sudo systemctl stop nginx               # Arreter
sudo systemctl reload nginx             # Recharger (zero downtime)
sudo systemctl restart nginx            # Redemarrer
nginx -t                                 # Tester la configuration
nginx -T                                 # Afficher la config complete
nginx -s reload                          # Signal de rechargement
nginx -V                                 # Version et modules compiles

# === EXEMPLE DE SERVER BLOCK (site statique) ===
# /etc/nginx/sites-available/monsite
# server {
#     listen 80;
#     server_name monsite.com www.monsite.com;
#     root /var/www/monsite;
#     index index.html;
#     location / {
#         try_files $uri $uri/ =404;
#     }
# }

# === REVERSE PROXY VERS NODE.JS ===
# server {
#     listen 80;
#     server_name api.monsite.com;
#     location / {
#         proxy_pass http://127.0.0.1:3000;
#         proxy_set_header Host $host;
#         proxy_set_header X-Real-IP $remote_addr;
#         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
#         proxy_set_header X-Forwarded-Proto $scheme;
#     }
# }

# === LOAD BALANCER ===
# upstream backend {
#     least_conn;
#     server 10.0.0.1:8080;
#     server 10.0.0.2:8080;
#     server 10.0.0.3:8080;
# }
# server {
#     listen 80;
#     location / { proxy_pass http://backend; }
# }

# Activer un site
sudo ln -s /etc/nginx/sites-available/monsite /etc/nginx/sites-enabled/
```

```
sudo nginx -t && sudo systemctl reload nginx

# HTTPS avec Let's Encrypt
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d monsite.com

# Logs
sudo tail -f /var/log/nginx/access.log
sudo tail -f /var/log/nginx/error.log
```

TRUCS & ASTUCES

-> worker_processes auto : utiliser tous les coeurs CPU automatiquement
-> worker_connections : augmenter a 4096 ou plus pour les sites a fort trafic
-> gzip on : activer la compression pour economiser la bande passante (sauf images)
-> Brotli : installer le module brotli pour une meilleure compression que gzip
-> Cache navigateur : ajouter expires 30d pour les fichiers statiques (images, CSS, JS)
-> Rate limiting : limit_req_zone pour proteger contre les attaques DDoS basiques
-> HTTP/2 : ajouter 'http2' dans la directive listen pour activer HTTP/2 (listen 443 ssl http2)
-> Logs JSON : configurer le format de log en JSON pour faciliter l'analyse (ELK, Grafana)
-> Stub status : activer stub_status pour le monitoring (Prometheus, Zabbix, etc.)
-> Nginx + PHP-FPM : la combinaison la plus performante pour servir du PHP

LIENS DE REFERENCE

Site officiel : https://nginx.org/
Documentation : https://nginx.org/en/docs/
Wiki : https://www.nginx.com/resources/wiki/
GitHub : https://github.com/nginx/nginx
Nginx Plus : https://www.nginx.com/products/nginx/
Directives : https://nginx.org/en/docs/dirindex.html
Let's Encrypt : https://letsencrypt.org/

3. Lighttpd - Ultra-leger (<1 Mo)

SYNTHESE

Lighttpd (prononce 'Lighty') est un serveur web open-source concu pour la vitesse et la legerete. Avec une empreinte inferieure a 1 Mo en memoire, il est ideal pour les systemes embarques, les Raspberry Pi et les serveurs a faibles ressources. Il supporte FastCGI, CGI, SCGI, SSL/TLS, les virtual hosts et la compression HTTP. Utilise par de grands sites comme Wikipedia (historiquement).

LICENCE :

BSD License - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu/Mint
sudo apt update && sudo apt install lighttpd

# Fedora/RHEL
sudo dnf install lighttpd

# Arch Linux
sudo pacman -S lighttpd

# openSUSE
sudo zypper install lighttpd

# Alpine (ideal pour conteneurs)
sudo apk add lighttpd

# Demarrer et activer
sudo systemctl start lighttpd && sudo systemctl enable lighttpd

# Verifier
curl -I http://localhost
```

CE QUE L'ON PEUT FAIRE

-> Serveur web ultra-leger : servir des sites statiques avec une empreinte memoire minimale
-> FastCGI/CGI : executer PHP, Python, Perl, Ruby via FastCGI pour le contenu dynamique
-> Virtual hosts : heberger plusieurs sites sur le meme serveur
-> HTTPS / SSL/TLS : chiffrement avec support des certificats Let's Encrypt
-> Compression HTTP : compression gzip pour reduire la bande passante (mod_compress)
-> URL rewriting : reecriture d'URL via mod_rewrite
-> Authentification : Basic et Digest via mod_auth
-> WebDAV : partage de fichiers via le protocole WebDAV
-> Systemes embarques : ideal pour Raspberry Pi, routeurs, NAS, IoT

CONFIGURATION & COMMANDES

```
# === FICHIERS DE CONFIGURATION ===
# /etc/lighttpd/lighttpd.conf         -> Config principale
# /etc/lighttpd/conf-available/       -> Modules disponibles
# /etc/lighttpd/conf-enabled/         -> Modules actives

# === COMMANDES ===
sudo systemctl start lighttpd
sudo systemctl reload lighttpd
sudo systemctl status lighttpd
lighttpd -t -f /etc/lighttpd/lighttpd.conf   # Tester la config
```

```
lighttpd -V                                     # Version et modules

# Activer un module
sudo lighttpd-enable-mod fastcgi ssl rewrite
sudo systemctl reload lighttpd

# === CONFIG MINIMALE ===
# server.document-root = "/var/www/html"
# server.port = 80
# server.username = "www-data"
# server.groupname = "www-data"
# mimetype.assign = ( ".html" => "text/html", ".css" => "text/css" )
# index-file.names = ( "index.html", "index.php" )

# === PHP VIA FASTCGI ===
# server.modules += ( "mod_fastcgi" )
# fastcgi.server = (
#     ".php" => (( "bin-path" => "/usr/bin/php-cgi",
#                  "socket"   => "/tmp/php-fastcgi.sock" ))
# )

# === VIRTUAL HOST ===
# $HTTP["host"] == "monsite.com" {
#     server.document-root = "/var/www/monsite"
# }

# HTTPS avec Let's Encrypt
sudo apt install certbot
sudo certbot certonly --webroot -w /var/www/html -d monsite.com

# Logs
sudo tail -f /var/log/lighttpd/access.log
sudo tail -f /var/log/lighttpd/error.log
```

TRUCS & ASTUCES

-> Ideal Raspberry Pi : lighttpd consomme 10x moins de RAM qu'Apache
-> mod_compress : activer pour economiser 60-80% de bande passante sur le texte
-> Event-driven : architecture asynchrone, pas de fork par connexion comme Apache prefork
-> Securite : server.tag = '' pour cacher la version du serveur dans les en-tetes
-> Alias : utiliser mod_alias pour mapper des URL vers des repertoires differents
-> Stats : activer mod_status pour un monitoring basique via URL
-> Combiner avec Nginx : utiliser lighttpd pour les sites simples, Nginx pour le reverse proxy

LIENS DE REFERENCE

Site officiel : https://www.lighttpd.net/
Documentation : https://redmine.lighttpd.net/projects/lighttpd/wiki
GitHub : https://github.com/lighttpd/lighttpd1.4
Modules : https://redmine.lighttpd.net/projects/lighttpd/wiki/Docs

4. Caddy - HTTPS automatique en Go

SYNTHESE

Caddy est un serveur web moderne ecrit en Go, dont la fonctionnalite phare est le HTTPS automatique : il obtient et renouvelle les certificats TLS (Let's Encrypt / ZeroSSL) sans aucune configuration. Son fichier de configuration (Caddyfile) est extremement simple et lisible. Il supporte le reverse proxy, le load balancing, HTTP/2, HTTP/3 (QUIC), les WebSockets, la compression et le file serving. Zero dependance externe, un seul binaire Go.

LICENCE :

Apache License 2.0 - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu/Mint (depot officiel)
sudo apt install -y debian-keyring debian-archive-keyring apt-transport-https
curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/gpg.key' | \
    sudo gpg --dearmor -o /usr/share/keyrings/caddy-stable.gpg
curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/debian.deb.txt' | \
    sudo tee /etc/apt/sources.list.d/caddy-stable.list
sudo apt update && sudo apt install caddy

# Fedora/RHEL
sudo dnf install caddy

# Arch Linux
sudo pacman -S caddy

# Docker
docker run -d -p 80:80 -p 443:443 \
    -v caddy_data:/data -v caddy_config:/config \
    caddy

# Binaire Go unique (toutes distros)
curl -OL https://github.com/caddyserver/caddy/releases/latest/download/\
caddy_*_linux_amd64.tar.gz
tar xzf caddy_*_linux_amd64.tar.gz
sudo mv caddy /usr/local/bin/
sudo chmod +x /usr/local/bin/caddy

# Demarrer
sudo systemctl start caddy && sudo systemctl enable caddy
```

CE QUE L'ON PEUT FAIRE

-> HTTPS automatique : obtient et renouvelle les certificats TLS sans aucune config
-> Reverse proxy : rediriger vers des backends avec une seule ligne de config
-> Load balancing : distribuer le trafic entre plusieurs backends
-> File server : servir des fichiers statiques avec directory listing
-> Compression : gzip et zstd automatiques
-> HTTP/3 (QUIC) : support natif du protocole HTTP/3 pour des performances optimales
-> WebSockets : proxy transparent des connexions WebSocket
-> API JSON : administrer Caddy dynamiquement via son API REST
-> On-demand TLS : generer des certificats a la volee pour les nouveaux domaines
-> Middleware : authentification, rate limiting, headers, redirections, templates

CONFIGURATION & COMMANDES

```
# === FICHIER DE CONFIGURATION ===
# /etc/caddy/Caddyfile               -> Config principale (Caddyfile)
# Ou API JSON sur localhost:2019

# === COMMANDES ===
sudo systemctl start caddy
sudo systemctl reload caddy             # Recharger la config
caddy validate --config /etc/caddy/Caddyfile  # Tester
caddy fmt --overwrite /etc/caddy/Caddyfile    # Formater
caddy version
caddy list-modules                      # Lister les modules

# === CADDYFILE : SITE STATIQUE AVEC HTTPS AUTO ===
# monsite.com {
#     root * /var/www/monsite
#     file_server
# }

# === CADDYFILE : REVERSE PROXY ===
# api.monsite.com {
#     reverse_proxy localhost:3000
# }

# === CADDYFILE : LOAD BALANCER ===
# monsite.com {
#     reverse_proxy 10.0.0.1:8080 10.0.0.2:8080 {
#         lb_policy least_conn
#     }
# }

# === CADDYFILE : PHP AVEC PHP-FPM ===
# monsite.com {
#     root * /var/www/monsite
#     php_fastcgi unix//run/php/php-fpm.sock
#     file_server
# }

# === CADDYFILE : AVEC BASIQUE AUTH ===
# monsite.com {
#     basicauth / {
#         admin $2a$14$hashed_password
#     }
#     file_server
# }
# Generer le hash : caddy hash-password

# === API REST ===
curl localhost:2019/config/ | jq .        # Config actuelle
curl -X POST localhost:2019/load \
    -H "Content-Type: text/caddyfile" \
    --data-binary @/etc/caddy/Caddyfile   # Recharger

# Logs
journalctl -u caddy --follow
```

TRUCS & ASTUCES

-> HTTPS par defaut : Caddy active HTTPS automatiquement pour tout domaine public valide
-> Caddyfile ultra-simple : 3 lignes suffisent pour un site complet avec HTTPS
-> caddy hash-password : generer un hash bcrypt pour l'authentification basique
-> caddy fmt : formater automatiquement le Caddyfile pour la lisibilite
-> On-demand TLS : ideal pour les SaaS multi-tenants avec des domaines personnalises
-> Placeholder : {host}, {path}, {remote_ip} utilisables dans les directives
-> Docker : Caddy est ideal pour le reverse proxy de conteneurs Docker
-> Wildcard TLS : supporter *.monsite.com avec un DNS challenge (Cloudflare, Route53)
-> Metriques : activer le module Prometheus pour le monitoring avec Grafana

LIENS DE REFERENCE

Site officiel :	https://caddyserver.com/
Documentation :	https://caddyserver.com/docs/
Caddyfile :	https://caddyserver.com/docs/caddyfile
GitHub :	https://github.com/caddyserver/caddy
Modules :	https://caddyserver.com/docs/modules/
Forum :	https://caddy.community/
API :	https://caddyserver.com/docs/api

5. OpenLiteSpeed - Performance & cache

SYNTHESE

OpenLiteSpeed est un serveur web open-source base sur LiteSpeed Enterprise. Architecture event-driven, il offre des performances exceptionnelles avec un cache integre intelligent (LSCache), un support natif PHP via LSAPI (plus rapide que PHP-FPM), et une interface d'administration web graphique. Compatible avec les regles .htaccess d'Apache. Ideal pour WordPress et les CMS PHP a fort trafic.

LICENCE :

GNU GPL v3 - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu
wget -O - https://rpms.litespeedtech.com/debian/enable_lst_debian_repo.sh | \
    sudo bash
sudo apt install openlitespeed

# RHEL/CentOS/Fedora
sudo rpm -Uvh http://rpms.litespeedtech.com/centos/litespeed-repo-1.3-1.el8.noarch.rpm
sudo dnf install openlitespeed

# Script d'installation universel
wget https://raw.githubusercontent.com/litespeedtech/ols1clk/master/ols1clk.sh
sudo bash ols1clk.sh

# Docker
docker run -d -p 80:80 -p 443:443 -p 7080:7080 \
    litespeedtech/openlitespeed

# Demarrer
sudo /usr/local/lsws/bin/lswsctrl start

# Acceder a la console WebAdmin
# https://localhost:7080
# Login par defaut : admin / mot_de_passe_a_changer
```

CE QUE L'ON PEUT FAIRE

-> Performance : architecture event-driven, 6x plus rapide qu'Apache selon les benchmarks
-> LSCache : cache integre au niveau du serveur (pas besoin de Varnish) - ideal pour WordPress
-> LSAPI : interface PHP native plus rapide que PHP-FPM (30-50% de gain)
-> Compatibilite .htaccess : lit les fichiers .htaccess d'Apache (mod_rewrite compatible)
-> Console WebAdmin : interface graphique web pour la configuration complete
-> HTTPS / HTTP/2 / HTTP/3 : support natif de QUIC/HTTP/3
-> Anti-DDoS : protection integree contre les attaques par debit
-> PHP multi-versions : executer plusieurs versions de PHP simultanement
-> WordPress optimise : plugin LSCache pour WordPress (acceleration 100x)

CONFIGURATION & COMMANDES

```
# === CHEMINS IMPORTANTS ===
# /usr/local/lsws/conf/httpd_config.conf  -> Config principale
# /usr/local/lsws/conf/vhosts/            -> Virtual hosts
# /usr/local/lsws/logs/                   -> Logs
# /usr/local/lsws/admin/                  -> Console WebAdmin

# === COMMANDES ===
sudo /usr/local/lsws/bin/lswsctrl start     # Demarrer
sudo /usr/local/lsws/bin/lswsctrl stop      # Arreter
sudo /usr/local/lsws/bin/lswsctrl restart   # Redemarrer
sudo /usr/local/lsws/bin/lswsctrl reload    # Recharger
sudo /usr/local/lsws/bin/lswsctrl status    # Etat

# Changer le mot de passe admin
sudo /usr/local/lsws/admin/misc/admpass.sh

# Console WebAdmin : https://localhost:7080

# Installer PHP via LSAPI
sudo apt install lsphp81 lsphp81-common lsphp81-mysql lsphp81-curl

# Logs
sudo tail -f /usr/local/lsws/logs/access.log
sudo tail -f /usr/local/lsws/logs/error.log
```

TRUCS & ASTUCES

-> LSCache + WordPress : installer le plugin 'LiteSpeed Cache' pour des performances extremes
-> Compatibilite Apache : migrer d'Apache a OpenLiteSpeed sans changer les .htaccess
-> Console WebAdmin : configurer les virtual hosts, SSL et PHP graphiquement
-> HTTP/3 : activer QUIC dans la console pour un chargement encore plus rapide
-> Multi-PHP : configurer differentes versions PHP par virtual host
-> Anti-DDoS : les limites de connexion et de bande passante sont configurables par IP
-> Benchmark : tester avec wrk ou ab pour comparer les performances vs Apache/Nginx
-> CyberPanel : installer CyberPanel pour un panneau de controle complet base sur OLS

LIENS DE REFERENCE

Site officiel : https://openlitespeed.org/
Documentation : https://openlitespeed.org/kb/
GitHub : https://github.com/litespeedtech/openlitespeed
LSCache : https://www.litespeedtech.com/products/cache-plugins
CyberPanel : https://cyberpanel.net/
Forum : https://forum.openlitespeed.org/

6. Hiawatha - Securite renforcee

SYNTHESE

Hiawatha est un serveur web leger ecrit en C, concu avec la securite comme priorite absolue. Il inclut des protections natives contre les attaques XSS, CSRF, SQL injection et path traversal. Monitoring integre, configuration simple, et empreinte minimale. Ideal pour les systemes embarques, les serveurs exposes et les environnements ou la securite est critique.

LICENCE :

GNU GPL v2 - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu
sudo apt update && sudo apt install hiawatha

# Fedora/RHEL (compilation ou COPR)
sudo dnf copr enable mstevens/hiawatha
sudo dnf install hiawatha

# Arch Linux (AUR)
yay -S hiawatha

# Compilation depuis les sources
wget https://www.hiawatha-webserver.org/files/hiawatha-latest.tar.gz
tar xzf hiawatha-latest.tar.gz && cd hiawatha-*
mkdir build && cd build
cmake .. && make && sudo make install

# Demarrer
sudo systemctl start hiawatha && sudo systemctl enable hiawatha
```

CE QUE L'ON PEUT FAIRE

-> Securite native : protection contre XSS, CSRF, SQL injection, path traversal
-> Serveur web : servir des fichiers statiques et dynamiques (CGI/FastCGI)
-> Virtual hosts : heberger plusieurs sites avec isolation de securite
-> HTTPS / SSL/TLS : chiffrement avec gestion fine des protocoles et ciphers
-> Monitoring integre : outil de surveillance des connexions actives et du trafic
-> Throttling : limitation de bande passante par connexion et par IP
-> URL toolkit : reecriture d'URL et redirections
-> Systemes embarques : empreinte minimale, ideal pour IoT et routeurs

CONFIGURATION & COMMANDES

```
# === FICHIERS DE CONFIGURATION ===
# /etc/hiawatha/hiawatha.conf         -> Config principale
# /etc/hiawatha/cgi-wrapper.conf      -> Config CGI
# /etc/hiawatha/mimetype.conf         -> Types MIME

# === COMMANDES ===
sudo systemctl start hiawatha
sudo systemctl reload hiawatha
sudo systemctl status hiawatha
hiawatha -k                              # Tester la config

# === CONFIG MINIMALE ===
# Binding {
```

```
#     Port = 80
# }
# Hostname = 127.0.0.1
# WebsiteRoot = /var/www/hiawatha
#
# VirtualHost {
#     Hostname = monsite.com
#     WebsiteRoot = /var/www/monsite
#     AccessLogfile = /var/log/hiawatha/monsite-access.log
# }

# === SECURITE ===
# PreventXSS = yes
# PreventCSRF = yes
# PreventSQLi = yes
# BanOnFlooding = 10/1:15         # Ban 15s si >10 req/s
# BanOnSQLi = 60                  # Ban 60s si SQL injection

# Logs
sudo tail -f /var/log/hiawatha/access.log
```

TRUCS & ASTUCES

-> PreventXSS/CSRF/SQLi : activer ces directives pour bloquer automatiquement les attaques
-> BanOnFlooding : bannir les IPs qui font trop de requetes (anti-DDoS basique)
-> Monitoring : activer le module monitor pour une vue web des statistiques serveur
-> Leger : Hiawatha consomme encore moins de ressources que Lighttpd
-> Configuration lisible : la syntaxe est plus simple qu'Apache ou Nginx
-> URL toolkit : remplacer mod_rewrite par les URL toolkit rules natives de Hiawatha
-> TLS strict : configurer les ciphers et protocoles TLS pour un score A+ sur SSL Labs

LIENS DE REFERENCE

Site officiel : https://www.hiawatha-webserver.org/
Documentation : https://www.hiawatha-webserver.org/manpages
Telechargement : https://www.hiawatha-webserver.org/download
Changelog : https://www.hiawatha-webserver.org/changelog

7. Apache Tomcat - Serveur Java (Servlets/JSP)

SYNTHESE

Apache Tomcat est le serveur d'applications Java le plus deploye au monde. Il implemente les specifications Java Servlet, JavaServer Pages (JSP), Java Expression Language (EL) et WebSocket. Il permet de deployer des applications web Java (fichiers .war), de gerer des sessions, de pooler les connexions JDBC et de clusterer pour la haute disponibilite. Souvent utilise derriere Apache ou Nginx en reverse proxy.

LICENCE :

Apache License 2.0 - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu/Mint
sudo apt update && sudo apt install tomcat10

# Fedora/RHEL
sudo dnf install tomcat

# Arch Linux
sudo pacman -S tomcat10

# Installation manuelle (recommande pour le controle)
sudo apt install default-jdk     # Prerequis : Java JDK
TOMCAT_VER=10.1.20
wget https://dlcdn.apache.org/tomcat/tomcat-10/v$TOMCAT_VER/\
bin/apache-tomcat-$TOMCAT_VER.tar.gz
sudo tar xzf apache-tomcat-$TOMCAT_VER.tar.gz -C /opt/
sudo ln -s /opt/apache-tomcat-$TOMCAT_VER /opt/tomcat

# Docker
docker run -d -p 8080:8080 --name tomcat tomcat:latest

# Demarrer
sudo systemctl start tomcat10
# Ou manuellement :
/opt/tomcat/bin/startup.sh

# Verifier (port 8080 par defaut)
curl -I http://localhost:8080
```

CE QUE L'ON PEUT FAIRE

-> Applications Java : deployer des servlets, JSP, et applications Spring Boot (.war)
-> WebSocket : support natif de WebSocket pour les applications temps reel
-> Pool JDBC : gestion des pools de connexions base de donnees
-> Clustering : haute disponibilite avec replication de sessions entre instances
-> Manager App : deployer et gerer les applications via une interface web
-> Virtual Hosts : heberger plusieurs applications sur des domaines differents
-> SSL/TLS : chiffrement des connexions avec certificats Java Keystore ou PEM
-> Valves : filtres de requetes (logging, acces IP, single sign-on)
-> JNDI : acces aux ressources (DataSources, mail, etc.) via JNDI

CONFIGURATION & COMMANDES

```
# === CHEMINS IMPORTANTS ===
# $CATALINA_HOME/conf/server.xml     -> Config serveur
# $CATALINA_HOME/conf/web.xml        -> Config par defaut des apps
# $CATALINA_HOME/conf/context.xml    -> Contexte global
# $CATALINA_HOME/conf/tomcat-users.xml -> Utilisateurs/roles
# $CATALINA_HOME/webapps/            -> Deploiement des applications
# $CATALINA_HOME/logs/               -> Logs

# === COMMANDES ===
$CATALINA_HOME/bin/startup.sh          # Demarrer
$CATALINA_HOME/bin/shutdown.sh         # Arreter
$CATALINA_HOME/bin/catalina.sh run     # Lancer en foreground
$CATALINA_HOME/bin/version.sh          # Version

# === VARIABLES D'ENVIRONNEMENT ===
export CATALINA_HOME=/opt/tomcat
export JAVA_HOME=/usr/lib/jvm/default-java
export CATALINA_OPTS="-Xms512M -Xmx2048M -server"
export JAVA_OPTS="-Djava.awt.headless=true"

# Deployer une application
cp monapp.war $CATALINA_HOME/webapps/

# Manager App (activer dans tomcat-users.xml)
# <user username="admin" password="secret"
#       roles="manager-gui,admin-gui"/>

# Logs
tail -f $CATALINA_HOME/logs/catalina.out
```

TRUCS & ASTUCES

-> Derriere Nginx/Apache : toujours mettre Tomcat derriere un reverse proxy en production
-> Manager App : activer dans tomcat-users.xml pour deployer les .war via l'interface web
-> CATALINA_OPTS : ajuster la memoire JVM (-Xms512M -Xmx2G) selon les besoins
-> AJP : utiliser le protocole AJP (port 8009) pour la communication avec Apache mod_proxy_ajp
-> Hot deployment : deposer un .war dans webapps/ pour un deploiement automatique
-> Securiser : supprimer les applications exemples (docs, examples, manager) en production
-> Access log valve : configurer dans server.xml pour des logs d'acces detailles
-> JMX : activer JMX pour le monitoring avec VisualVM, JConsole ou Prometheus JMX Exporter

LIENS DE REFERENCE

Site officiel : https://tomcat.apache.org/
Documentation : https://tomcat.apache.org/tomcat-10.1-doc/
Wiki : https://cwiki.apache.org/confluence/display/TOMCAT/
GitHub : https://github.com/apache/tomcat
Telechargement : https://tomcat.apache.org/download-10.cgi
FAQ : https://cwiki.apache.org/confluence/display/TOMCAT/FAQ

8. Node.js - Runtime JavaScript serveur

SYNTHESE

Node.js est un environnement d'execution JavaScript cote serveur base sur le moteur V8 de Chrome. Son architecture evenementielle et non-bloquante le rend ideal pour les applications temps reel, les API REST, les microservices et les applications a forte concurrence. Le module http integre permet de creer un serveur web, mais il est generalement combine avec des frameworks (Express, Fastify, Koa, NestJS) et place derriere Nginx en reverse proxy.

LICENCE :

MIT License - Libre & Gratuit

INSTALLATION

```
# Debian/Ubuntu (depot NodeSource - recommande)
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install nodejs

# Fedora/RHEL
curl -fsSL https://rpm.nodesource.com/setup_20.x | sudo bash -
sudo dnf install nodejs

# Arch Linux
sudo pacman -S nodejs npm

# NVM (Node Version Manager - recommande pour le dev)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.bashrc
nvm install 20          # Installer Node.js 20 LTS
nvm use 20              # Utiliser cette version
nvm ls                  # Lister les versions installees

# Alpine
sudo apk add nodejs npm

# Docker
docker run -d -p 3000:3000 node:20-alpine

# Verification
node --version
npm --version
```

CE QUE L'ON PEUT FAIRE

-> Serveur HTTP : creer un serveur web avec le module http natif ou Express/Fastify
-> API REST : construire des API RESTful performantes avec Express, Fastify, NestJS
-> Temps reel : applications WebSocket avec Socket.io, ws ou uWebSockets
-> Microservices : architecture microservices avec communication inter-services
-> Server-Side Rendering : rendu cote serveur avec Next.js (React) ou Nuxt.js (Vue)
-> Streaming : traitement de flux de donnees (fichiers, video, uploads) sans surcharge memoire
-> CLI tools : creer des outils en ligne de commande (npm, yarn, webpack, etc.)
-> Serverless : fonctions cloud (AWS Lambda, Google Cloud Functions, Vercel)
-> Full-stack : JavaScript du frontend au backend avec le meme langage

CONFIGURATION & COMMANDES

```
# === COMMANDES ===
node app.js                          # Lancer une application
node -e "console.log('hello')"       # Executer du code inline
node --inspect app.js                # Lancer avec debugger
node --watch app.js                  # Redemarrer sur modification (18+)

# === NPM ===
npm init -y                          # Initialiser un projet
npm install express                  # Installer un package
npm install -D nodemon               # Installer en dev dependency
npm start                            # Lancer (script start de package.json)
npm run dev                          # Script personnalise
npm ls                               # Lister les packages
npm audit                            # Verifier les vulnerabilites
npm update                           # Mettre a jour

# === SERVEUR HTTP MINIMAL ===
# const http = require('http');
# const server = http.createServer((req, res) => {
#     res.writeHead(200, {'Content-Type': 'text/html'});
#     res.end('<h1>Hello World</h1>');
# });
# server.listen(3000, () => console.log('Port 3000'));

# === EXPRESS MINIMAL ===
# const express = require('express');
# const app = express();
# app.get('/', (req, res) => res.send('Hello'));
# app.listen(3000);

# === VARIABLES D'ENVIRONNEMENT ===
export NODE_ENV=production           # Mode production
export PORT=3000                     # Port d'ecoute
export NODE_OPTIONS="--max-old-space-size=4096"  # Memoire max

# Process manager (production)
npm install -g pm2
pm2 start app.js --name monapp -i max  # Cluster mode
pm2 status                           # Etat des processus
pm2 logs                             # Voir les logs
pm2 reload monapp                    # Zero-downtime reload
pm2 startup                          # Demarrage au boot
```

TRUCS & ASTUCES

-> JAMAIS en frontal en production : toujours placer Node.js derriere Nginx ou Caddy
-> PM2 : utiliser PM2 en production pour le clustering, le monitoring et le restart automatique
-> NODE_ENV=production : activer le mode production pour desactiver le debug et optimiser
-> NVM : utiliser NVM pour gerer plusieurs versions de Node.js sur la meme machine
-> npm audit : verifier regulierement les vulnerabilites des dependances
-> Cluster mode : pm2 start app.js -i max pour utiliser tous les coeurs CPU
-> .env : utiliser le package dotenv pour charger les variables d'environnement
-> Graceful shutdown : gerer les signaux SIGTERM/SIGINT pour fermer proprement les connexions
-> Health check : exposer un endpoint /health pour le monitoring (Prometheus, Uptime Kuma)
-> Helmet : npm install helmet pour securiser les en-tetes HTTP avec Express

LIENS DE REFERENCE

Site officiel :	https://nodejs.org/
Documentation :	https://nodejs.org/docs/latest/api/
npm Registry :	https://www.npmjs.com/
GitHub :	https://github.com/nodejs/node
Express.js :	https://expressjs.com/
Fastify :	https://fastify.dev/
NestJS :	https://nestjs.com/
PM2 :	https://pm2.keymetrics.io/
NVM :	https://github.com/nvm-sh/nvm

9. Python http.server - Serveur web en une commande

SYNTHESE

Python integre un serveur HTTP dans sa bibliotheque standard (module http.server). En une seule commande, il est possible de servir les fichiers du repertoire courant sur le reseau local. Ideal pour le developpement, les tests rapides, le partage de fichiers en LAN, les demos et le prototypage. Pas concu pour la production (pas de HTTPS, pas de concurrence, pas de securite), mais extremement pratique au quotidien. Python est pre-installe sur la quasi-totalite des distributions Linux.

LICENCE :

PSF License (Python Software Foundati

INSTALLATION

```
# Python est pre-installe sur la plupart des distributions Linux
python3 --version

# Si absent, installer :
# Debian/Ubuntu/Mint
sudo apt install python3

# Fedora/RHEL
sudo dnf install python3

# Arch Linux
sudo pacman -S python

# Alpine
sudo apk add python3

# === LANCER UN SERVEUR WEB EN UNE COMMANDE ===
# Servir le repertoire courant sur le port 8000
python3 -m http.server

# Specifier un port
python3 -m http.server 9090

# Specifier une adresse d'ecoute
python3 -m http.server 8080 --bind 0.0.0.0

# Servir un repertoire specifique
python3 -m http.server --directory /chemin/vers/dossier

# Avec Python 2 (ancien, deprecie)
python -m SimpleHTTPServer 8000
```

CE QUE L'ON PEUT FAIRE

-> Serveur de fichiers instantane : partager des fichiers via le navigateur en une commande
-> Developpement web : tester un site statique (HTML/CSS/JS) localement sans installer Nginx/Apache
-> Partage LAN : partager un dossier avec d'autres machines sur le reseau local
-> Demo rapide : presenter un projet web sans deploiement
-> Documentation : servir une documentation generee (Sphinx, MkDocs, Javadoc) pour la relecture
-> Telechargement : permettre le telechargement de fichiers depuis une machine distante (SSH + tunnel)
-> Tests API : combiner avec des scripts CGI basiques pour tester des endpoints
-> Listing de repertoire : navigation dans l'arborescence des fichiers via le navigateur
-> Prototypage : tester des appels fetch() JavaScript sans erreurs CORS de fichier local

CONFIGURATION & COMMANDES

```
# === COMMANDES ESSENTIELLES ===

# Servir le repertoire courant (port 8000)
python3 -m http.server
# -> Accessible sur http://localhost:8000

# Port personnalise
python3 -m http.server 3000

# Ecouter sur toutes les interfaces (LAN)
python3 -m http.server 8080 --bind 0.0.0.0
# -> Accessible depuis d'autres machines : http://<votre-ip>:8080

# Servir un dossier specifique
python3 -m http.server --directory ./build
python3 -m http.server --directory /var/www/html 8080

# Combinaison complete
python3 -m http.server 9090 --bind 0.0.0.0 --directory ./dist

# En arriere-plan
python3 -m http.server 8000 &
# Pour arreter : kill %1 ou kill $(lsof -ti:8000)

# Via SSH tunnel (partager un fichier depuis un serveur distant)
# Sur le serveur : python3 -m http.server 8000 --directory /data
# Sur le client :  ssh -L 8000:localhost:8000 user@serveur
# -> http://localhost:8000 sur le client

# === SERVEUR PERSONNALISE (script Python) ===
# Fichier : server.py
# ---
# from http.server import HTTPServer, SimpleHTTPRequestHandler
# import ssl
#
# # Serveur HTTPS basique (dev uniquement)
# server = HTTPServer(('0.0.0.0', 4443), SimpleHTTPRequestHandler)
# ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
# ctx.load_cert_chain('cert.pem', 'key.pem')
# server.socket = ctx.wrap_socket(server.socket, server_side=True)
# print("HTTPS sur https://localhost:4443")
# server.serve_forever()
# ---
# Generer un certificat auto-signe :
# openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem \
#     -days 365 -nodes -subj '/CN=localhost'
# python3 server.py

# === SERVEUR CGI (contenu dynamique basique) ===
python3 -m http.server --cgi 8000
# Les scripts dans ./cgi-bin/ sont executables

# === ALTERNATIVES PYTHON PLUS AVANCEES ===
# Flask (micro-framework)
pip install flask
```

```
# python3 -c "from flask import Flask; app=Flask(__name__); \
#   app.route('/')(lambda:'Hello'); app.run(host='0.0.0.0')"

# Uvicorn + FastAPI (async, haute performance)
pip install uvicorn fastapi
# uvicorn main:app --host 0.0.0.0 --port 8000

# Gunicorn (serveur WSGI production)
pip install gunicorn
# gunicorn -w 4 -b 0.0.0.0:8000 app:app

# Twisted (serveur asynchrone)
pip install twisted
# twistd web --path . --port 8080
```

TRUCS & ASTUCES

-> Usage #1 : python3 -m http.server dans un dossier pour le partager instantanement
-> Bind 0.0.0.0 : par defaut, ecoute sur localhost uniquement. Ajouter --bind 0.0.0.0 pour le LAN
-> Trouver son IP LAN : hostname -I ou ip addr show pour partager l'URL avec les collegues
-> Pas pour la production : aucune securite, pas de HTTPS, pas de concurrence, pas de cache
-> HTTPS dev : utiliser le script personnalise avec ssl.SSLContext pour tester en HTTPS local
-> Port 80 : sudo python3 -m http.server 80 pour servir sur le port standard (necessite root)
-> One-liner Flask : flask est souvent meilleur pour du prototypage avec du contenu dynamique
-> Combinaison SSH : ideal pour recuperer des fichiers depuis un serveur distant via tunnel
-> Alias pratique : ajouter dans .bashrc : alias serve='python3 -m http.server'
-> Ctrl+C : arreter le serveur proprement avec Ctrl+C dans le terminal
-> Log : chaque requete est loguee dans le terminal (IP, methode, chemin, code HTTP)
-> CGI : activer --cgi pour executer des scripts dans le dossier cgi-bin/ (Python, Bash, Perl)

LIENS DE REFERENCE

Documentation : https://docs.python.org/3/library/http.server.html
Python.org : https://www.python.org/
Flask : https://flask.palletsprojects.com/
FastAPI : https://fastapi.tiangolo.com/
Gunicorn : https://gunicorn.org/
Uvicorn : https://www.uvicorn.org/
Twisted : https://twisted.org/

Annexe A : Lexique des Serveurs Web

HTTP (HyperText Transfer Protocol)

Protocole de communication entre un navigateur (client) et un serveur web. HTTP/1.1 (1997), HTTP/2 (2015, multiplexage), HTTP/3 (2022, base sur QUIC/UDP).

HTTPS

HTTP securise par chiffrement TLS/SSL. Le navigateur et le serveur echangent un certificat pour chiffrer toutes les communications. Port standard : 443.

TLS / SSL

TLS (Transport Layer Security) est le successeur de SSL. Protocole de chiffrement des communications. TLS 1.2 et 1.3 sont les versions actuelles. SSL est obsolete.

Certificat SSL/TLS

Fichier numerique authentifiant l'identite d'un serveur et permettant le chiffrement. Delivre par une Autorite de Certification (CA). Let's Encrypt offre des certificats gratuits.

Let's Encrypt

Autorite de certification gratuite et automatisee. Fournit des certificats TLS via le protocole ACME. Certbot est le client le plus utilise.

Virtual Host / Server Block

Configuration permettant d'heberger plusieurs sites web sur un seul serveur. Apache utilise le terme 'Virtual Host', Nginx utilise 'Server Block'.

Reverse Proxy

Serveur intermediaire qui recoit les requetes des clients et les transmet a un ou plusieurs serveurs backend. Avantages : securite, load balancing, cache, SSL termination.

Forward Proxy

Serveur intermediaire entre les clients et Internet. Les clients se connectent au proxy qui fait les requetes a leur place. Utilise pour le filtrage ou l'anonymat.

Load Balancer

Repartiteur de charge distribuant le trafic entre plusieurs serveurs. Algorithmes : round-robin, least connections, IP hash, weighted.

CGI (Common Gateway Interface)

Protocole standard permettant au serveur web d'executer des programmes externes (scripts) pour generer du contenu dynamique. Ancien et lent (un processus par requete).

FastCGI

Evolution de CGI gardant les processus actifs en permanence au lieu de les recreer a chaque requete. Beaucoup plus performant. Utilise par PHP-FPM, Python, Ruby.

PHP-FPM (FastCGI Process Manager)

Gestionnaire de processus FastCGI pour PHP. Pool de processus PHP prets a repondre. La methode standard pour executer PHP avec Nginx, Caddy et OpenLiteSpeed.

LSAPI (LiteSpeed Server API)

Interface proprietaire de LiteSpeed pour communiquer avec PHP. Plus rapide que PHP-FPM (30-50% de gain). Disponible avec OpenLiteSpeed et LiteSpeed Enterprise.

Servlet

Composant Java cote serveur traitant les requetes HTTP. Execute dans un conteneur de servlets (Tomcat, Jetty). Equivalent Java de PHP/Python cote serveur.

JSP (JavaServer Pages)

Technologie Java permettant de creer des pages HTML dynamiques avec du code Java integre. Compile en servlets par le conteneur (Tomcat).

WAR (Web Application Archive)

Format d'archive Java contenant une application web complete (classes, JSP, HTML, config). Deploye dans un conteneur comme Tomcat par simple copie.

API Gateway

Point d'entree unique pour les microservices. Gere le routage, l'authentification, le rate limiting et la transformation des requetes. Nginx et Caddy peuvent servir d'API Gateway.

WebSocket

Protocole de communication bidirectionnelle full-duplex sur une connexion TCP. Permet les echanges temps reel entre client et serveur (chat, jeux, notifications).

HTTP/2

Version 2 du protocole HTTP (2015). Multiplexage (plusieurs requetes sur une connexion), server push, compression des en-tetes, priorisation des flux. Necessite TLS.

HTTP/3 (QUIC)

Version 3 du protocole HTTP (2022). Base sur QUIC (UDP) au lieu de TCP. Connexion plus rapide, meilleure gestion de la perte de paquets, migration de connexion.

CDN (Content Delivery Network)

Reseau de serveurs distribues geographiquement pour livrer le contenu au plus pres des utilisateurs. Exemples : Cloudflare, Fastly, AWS CloudFront, Akamai.

Cache HTTP

Stockage temporaire des reponses HTTP pour eviter de regenerer le contenu a chaque requete. Cache navigateur (Cache-Control), cache proxy (Varnish, Nginx), cache applicatif.

En-tetes HTTP (Headers)

Metadonnees associees aux requetes et reponses HTTP. Exemples : Content-Type, Cache-Control, Authorization, X-Forwarded-For, Strict-Transport-Security.

301 / 302 Redirect

301 = redirection permanente (le navigateur memorise). 302 = redirection temporaire. Utilise pour changer d'URL, forcer HTTPS ou rediriger vers www.

Codes HTTP

200 (OK), 301 (Moved), 302 (Found), 304 (Not Modified), 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), 500 (Internal Error), 502 (Bad Gateway), 503 (Unavailable).

MPM (Multi-Processing Module)

Module Apache definissant le modele de concurrence. prefork : un processus par requete. worker : threads dans des processus. event : asynchrone (recommande).

.htaccess

Fichier de configuration Apache par repertoire. Permet le rewriting d'URL, l'authentification et les redirections sans modifier la config principale. Desavantage : surcharge de performance car lu a chaque requete.

CORS (Cross-Origin Resource Sharing)

Mecanisme de securite des navigateurs controlant les requetes entre domaines differents. Configure via les en-tetes Access-Control-Allow-Origin sur le serveur.

WAF (Web Application Firewall)

Pare-feu applicatif filtrant les requetes HTTP pour proteger contre les attaques (SQL injection, XSS, CSRF). Exemples : ModSecurity (Apache/Nginx), Hiawatha natif.

Rate Limiting

Limitation du nombre de requetes par IP/utilisateur par periode. Protege contre les attaques DDoS et l'abus d'API. Configure dans Nginx (limit_req), Caddy, etc.

Keep-Alive

Mecanisme HTTP permettant de reutiliser une connexion TCP pour plusieurs requetes HTTP. Reduit la latence en evitant la negociation TCP repetee.

Gzip / Brotli

Algorithmes de compression des reponses HTTP. Gzip : standard et universel. Brotli (Google) : 15-25% plus efficace que Gzip. Active dans le serveur web.

Worker / Event-driven

Worker : modele ou des threads traitent les requetes en parallele. Event-driven : modele asynchrone ou une boucle d'evenements gere des milliers de connexions (Nginx, Node.js).

Document Root

Repertoire racine contenant les fichiers du site web. Par defaut : /var/www/html (Apache/Nginx). Les fichiers dans ce repertoire sont accessibles via le navigateur.

Access Log / Error Log

Fichiers de journalisation du serveur. Access log : chaque requete recue. Error log : erreurs du serveur et des applications. Essentiels pour le debugging et le monitoring.

Annexe B : Tableau Comparatif

Serveur	Licence	Langage	Usage principal	HTTP/2	HTTP/3	Rev.Proxy	Empreinte
Apache	Apache 2.0	C	Hebergement generaliste	Oui	Mod	Oui	Moyenne
Nginx	BSD	C	Haute perf., reverse proxy	Oui	Exp.	Oui	Faible
Lighttpd	BSD	C	Serveur ultra-leger	Oui	Non	Non	Tres faible
Caddy	Apache 2.0	Go	HTTPS auto, moderne	Oui	Oui	Oui	Faible
OpenLiteSpeed	GPL v3	C++	Performance, cache PHP	Oui	Oui	Oui	Faible
Hiawatha	GPL v2	C	Securite renforcee	Non	Non	Non	Tres faible
Tomcat	Apache 2.0	Java	Applications Java	Oui	Non	Non	Moyenne
Node.js	MIT	C++/JS	API, temps reel, JS	Via proxy	Via proxy	Non	Faible
Python http	PSF	Python	Dev, tests, partage LAN	Non	Non	Non	Minimale

Parts de marche (estimations 2025)

Nginx ~34% *Leader mondial, en forte croissance*
Apache ~23% *Historique, en lent declin*
Cloudflare ~22% *CDN/proxy, croissance rapide*
OpenLiteSpeed/LS ~2% *Niche WordPress haute perf.*
Caddy ~0.5% *Croissance rapide, projets modernes*
Lighttpd <0.5% *Niche embarque/IoT*
Hiawatha <0.1% *Niche securite*

Annexe C : Liens de Reference Globaux

Serveurs Web

Apache HTTP Server	https://httpd.apache.org/
Nginx	https://nginx.org/
Lighttpd	https://www.lighttpd.net/
Caddy	https://caddyserver.com/
OpenLiteSpeed	https://openlitespeed.org/
Hiawatha	https://www.hiawatha-webserver.org/
Apache Tomcat	https://tomcat.apache.org/
Node.js	https://nodejs.org/
Python http.server	https://docs.python.org/3/library/http.server.html

Outils & Frameworks Associes

Let's Encrypt	https://letsencrypt.org/
Certbot	https://certbot.eff.org/
PHP-FPM	https://www.php.net/manual/en/install.fpm.php
Express.js	https://expressjs.com/
PM2	https://pm2.keymetrics.io/
ModSecurity (WAF)	https://modsecurity.org/
Varnish (cache HTTP)	https://varnish-cache.org/
Flask (Python)	https://flask.palletsprojects.com/
FastAPI (Python)	https://fastapi.tiangolo.com/
Gunicorn (Python WSGI)	https://gunicorn.org/
Uvicorn (Python ASGI)	https://www.uvicorn.org/

Monitoring & Performance

Prometheus	https://prometheus.io/
Grafana	https://grafana.com/
Uptime Kuma	https://github.com/louislam/uptime-kuma
GoAccess (logs)	https://goaccess.io/
ab (Apache Bench)	https://httpd.apache.org/docs/current/programs/ab.html
wrk (benchmark HTTP)	https://github.com/wg/wrk
k6 (load testing)	https://k6.io/

Securite

SSL Labs (test TLS)	https://www.ssllabs.com/ssltest/
Security Headers	https://securityheaders.com/
OWASP	https://owasp.org/
Mozilla SSL Config	https://ssl-config.mozilla.org/
CIS Benchmarks	https://www.cisecurity.org/

Communautes & Apprentissage

DigitalOcean Tutorials	https://www.digitalocean.com/community/tutorials
Nginx Blog	https://www.nginx.com/blog/
Caddy Community	https://caddy.community/
Stack Overflow	https://stackoverflow.com/questions/tagged/nginx
Reddit r/selfhosted	https://www.reddit.com/r/selfhosted/

Reddit r/webdev	https://www.reddit.com/r/webdev/