MANUEL D'HÉBERGEMENT WEB

CHAPITRE 1 — Comprendre l'hébergement web de A à Z ce qu'est un hébergement web, comment il fonctionne et comment il s'architecture. Il sert de base théorique solide pour toutes les manipulations futures.

1.1 Qu'est-ce qu'un hébergement web ?

Un site web ne vit pas dans l'ordinateur du développeur. Il doit être placé sur une machine :

- allumée 24h/24
- connectée en permanence à Internet
- capable de répondre à des requêtes entrantes
- capable d'interpréter du code (PHP, Node, Python…)
- capable de stocker des données durablement

Cette machine est appelée un **serveur web**.

Un hébergement web est donc :

> « Un environnement matériel + logiciel permettant de stocker, exécuter et diffuser un site ou une application web. »

1.2 Les composants fondamentaux d'un hébergement

1. Une machine (physique ou virtuelle)

Elle contient :

- CPU
- RAM
- stockage
- réseau

Elle peut être hébergée dans :

- un datacenter
- un cloud
- ton propre bureau
- un centre d'hébergement local

2. Un système d'exploitation serveur

En général :

- Ubuntu Server
- Debian
- CentOS / Rocky
- Windows Server

Pour ce cours : **Ubuntu Server 22.04 LTS**.

3. Un serveur HTTP

Le plus utilisé : **Apache HTTP Server**.

Exemple :

Tacitement, Apache écoute sur le port 80 (HTTP) et 443 (HTTPS).

Commande pour vérifier :

```
sudo systemctl status apache2
```

4. Un langage côté serveur

- PHP (le plus répandu)
- Node.js
- Python (Django, Flask)
- Ruby

Pour ce cours : **PHP 8.1**.

5. Une base de données

- MySQL / MariaDB
- PostgreSQL
- SQLite

Pour ce cours : **MySQL Server**.

1.3 Comment fonctionne réellement une requête web ?

Exemple : L'utilisateur tape : **https://monsite.fr/page**

Voici ce qui se passe :

1. Le navigateur contacte un serveur DNS → obtient l'adresse IP.
2. Le navigateur envoie une requête au serveur web (Apache).
3. Apache reçoit la requête, vérifie le VirtualHost.
4. Apache va soit : envoyer un fichier statique, exécuter un script via PHP, ou appeler un backend (ex : Node.js).
5. Si le code demande la base de données, Apache et PHP interrogent MySQL.
6. PHP génère une page HTML.
7. Apache renvoie cette page au navigateur.
8. Le navigateur l'affiche.

1.4 Les types d'hébergement

Mutualisé

Plusieurs sites dans la même machine. Ressources partagées.

Avantages : prix bas, simplicité.

Inconvénients : peu de contrôle, performances limitées.

Représentation :

```
[Serveur physique]
 ├── Site 1
 ├── Site 2
 └── Site 3
```

VPS (Virtual Private Server)

Le plus utilisé pour les projets pédagogiques et professionnels.

```
[Serveur physique]
 ├── VPS 1 (isolé)
 ├── VPS 2 (isolé)
 └── VPS 3 (isolé)
```

Avantages : autonomie, root, isolation.

Inconvénients : nécessite de savoir configurer.

Serveur dédié

Le serveur entier est pour toi.

> [Serveur physique] → votre machine

Avantage : puissance.

Inconvénient : coûte cher.

Cloud (scalable)

Infrastructure distribuée.

```
[Load Balancer]
 ├── Serveur Web A
 ├── Serveur Web B
 └── Serveur Web C

[Base de données répliquée]
```

Avantage : s'adapte à la charge.

Inconvénient : complexe à maîtriser.

1.5 Nom de domaine, DNS, IP & Serveur

A – IP publique du serveur

Pour afficher un site, il faut une adresse accessible :

- 127.0.0.1 → local
- 192.168.x.x → réseau interne
- x.x.x.x → IP publique (Internet)

Commande pour connaître l'IP :

```
hostname -I
```

B – Nom de domaine

Un domaine est un raccourci vers une IP.

Exemple DNS :

monsite.fr → 51.178.22.94

1.6 Ports réseau essentiels

Port 22

SSH

Port 80

HTTP

Port 443

HTTPS

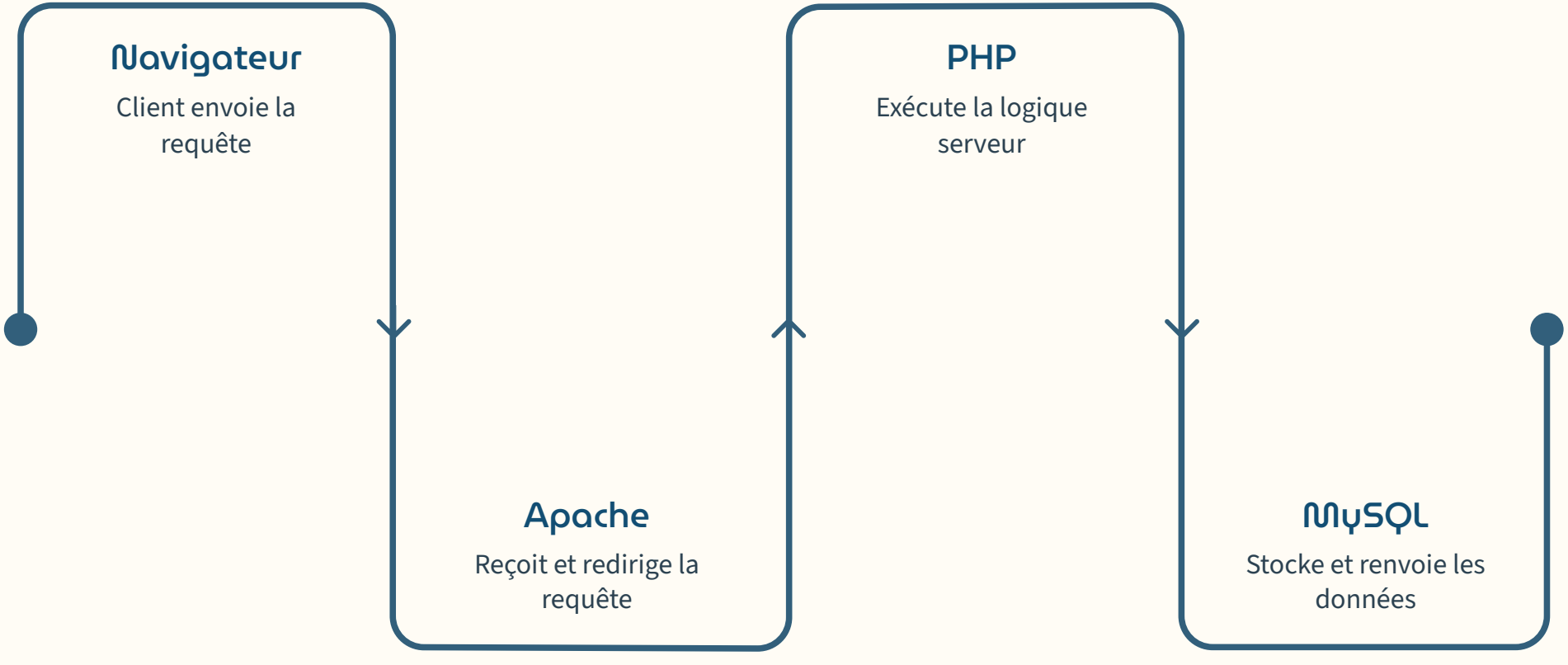
Commande pour vérifier l'écoute :

```
sudo ss -tulnp
```

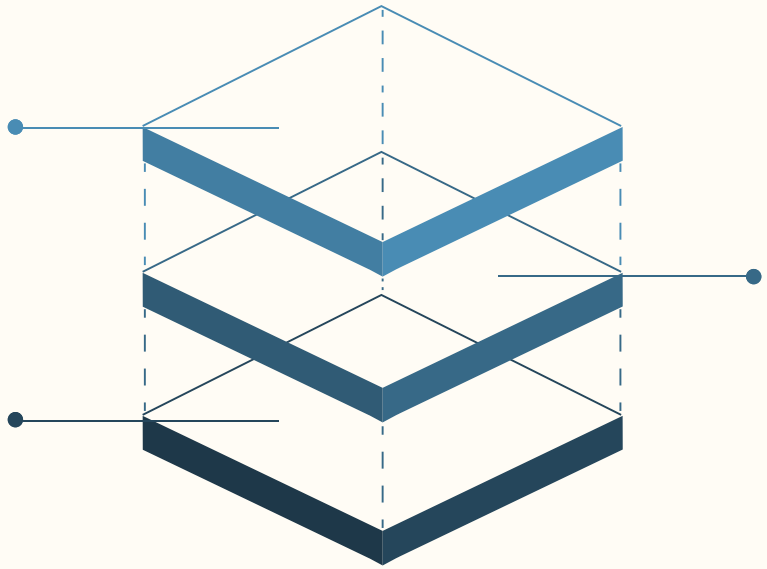
Vous devriez voir apache2 sur les ports 80 et 443.

1.7 Architecture minimale pour héberger un site

Voici un schéma textuel simple :



1.8 Architecture avancée (load balancing, failover)



Réplication MySQL

Base répliquée pour tolérance

Load Balancer

Distribue le trafic entrant

Serveurs Web

Deux nœuds en haute disponibilité

Principes clé :

- si un serveur tombe, l'autre prend le relais
- montée en charge automatique

1.9 Exercices théoriques

Ces exercices permettent de remplir une demi-journée :

1. Expliquer ce qu'est un hébergement avec vos mots.
2. Comparer VPS vs Mutualisé.
3. Décrire ce qui se passe lorsqu'on charge une page web.
4. Dire quel type d'hébergement convient pour : un blog personnel, une boutique e-commerce, une application d'école, un réseau social
5. Représenter une architecture haute disponibilité.
6. Expliquer pourquoi Apache n'est pas suffisant sans PHP.
7. Expliquer pourquoi un serveur doit écouter sur un port.

CHAPITRE 2 — Installation et configuration d'un serveur Linux

2.1 Découvrir la ligne de commande Linux

Un serveur Ubuntu fonctionne sans interface graphique. Toutes les actions se font via la ligne de commande.

Commandes indispensables :

`ls`	lister les fichiers
`cd`	changer de dossier
`pwd`	afficher le dossier actuel
`mkdir`	créer un dossier
`rm -rf`	supprimer un dossier
`cp -R`	copier un dossier et son contenu
`nano`	éditer un fichier

Exemple d'usage :

```
cd /var/www
ls
```

2.2 Mise à jour du système

Avant d'installer quoi que ce soit, un serveur doit être mis à jour.

Commande :

```
sudo apt update
sudo apt upgrade -y
```

Explications à connaître :

- `apt update` met à jour la liste des paquets disponibles
- `apt upgrade` installe les mises à jour sur le système
- l'option `-y` évite les confirmations manuelles

Le serveur doit être redémarré uniquement si le système le demande.

Commande de redémarrage :

```
sudo reboot
```

2.3 Installation d'Apache

Apache est le serveur web requis

Installation :

```
sudo apt install apache2 -y
```

Vérification du service :

```
sudo systemctl status apache2
```

Résultat attendu : **active (running)**

Fichiers importants :

`/etc/apache2/apache2.conf`	configuration principale
`/etc/apache2/sites-available/`	VirtualHosts disponibles
`/etc/apache2/sites-enabled/`	VirtualHosts actifs
`/var/www/`	fichiers des sites

Commandes Apache utiles :

```
sudo systemctl reload apache2
sudo systemctl restart apache2
sudo apache2ctl configtest
```

2.4 Installation de PHP

Le programme exige de savoir :

- installer un interpréteur PHP
- installer les extensions essentielles
- vérifier la version

Commande :

```
sudo apt install php libapache2-mod-php php-mysql php-xml php-mbstring php-curl php-zip php-gd -y
```

Vérifier :

```
php -v
```

vous devez comprendre que :

- Apache appelle PHP pour exécuter du code dynamique
- PHP peut se connecter à une base de données
- les extensions permettent certaines fonctionnalités (ex. XML, image, MySQL)

2.5 Installation de MySQL Server

Exigence du référentiel : « Installer une base de données et la configurer pour une application web. »

Installation :

```
sudo apt install mysql-server -y
```

Sécurisation :

```
sudo mysql_secure_installation
```

Réponses à connaître :

- Oui → supprimer comptes anonymes
- Oui → désactiver root distant
- Oui → supprimer base de test

2.6 Création d'une base de données et d'un utilisateur

Le référentiel exige de créer une base et un utilisateur dédié, car c'est une bonne pratique de sécurité.

Entrer dans MySQL :

```
sudo mysql
```

Créer base et utilisateur :

```
CREATE DATABASE appweb DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'appuser'@'localhost' IDENTIFIED BY 'MotDePasseFort123!';
GRANT ALL PRIVILEGES ON appweb.* TO 'appuser'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

Notions obligatoires à comprendre :

- DATABASE = espace de stockage
- USER = compte dédié
- GRANT = autorisations
- ne jamais utiliser root pour une application web

2.7 Navigation dans /var/www/

Le référentiel ne demande pas de connaître d'autres dossiers, mais exige de maîtriser `/var/www`.

Commandes :

```
cd /var/www
ls -l
```

Permissions attendues : **www-data www-data**

Commande pour corriger :

```
sudo chown -R www-data:www-data /var/www/appweb
sudo chmod -R 755 /var/www/appweb
```

2.8 Ports et firewall (UFW)

Compétence demandée : "Savoir ouvrir et fermer les ports nécessaires au fonctionnement d'un serveur web."

Installation :

```
sudo apt install ufw -y
```

Autoriser SSH :

```
sudo ufw allow 'OpenSSH'
```

Autoriser Apache :

```
sudo ufw allow 'Apache'
```

Activer le firewall :

```
sudo ufw enable
```

Vérifier les règles :

```
sudo ufw status
```

2.9 Tester le serveur web

Commande pour récupérer l'IP :

```
hostname -I
```

Dans le navigateur :

http://ADRESSE_IP

La page : **Apache2 Ubuntu Default Page** est la validation du chapitre 2.

2.10 Erreurs fréquentes à connaître

Apache ne démarre pas

```
sudo systemctl status apache2
```

Port déjà utilisé

```
sudo ss -tulnp | grep 80
```

Page blanche

Problème PHP → logs :

```
sudo tail -f /var/log/apache2/error.log
```

Permission denied

Corriger :

```
sudo chown -R www-data:www-data /var/www/appweb
```

CHAPITRE 3 — VirtualHost, structure d'un site et configuration Apache

3.1 Pourquoi configurons-nous Apache ?

Apache est livré avec un site par défaut et une configuration générique.

Pour héberger une vraie application web, il faut :

1. Créer un dossier dédié dans `/var/www`
2. Créer un VirtualHost pour dire à Apache : quel domaine correspond à quel dossier, où se trouvent les logs, quelles règles appliquer
3. Activer le VirtualHost
4. Tester la configuration
5. Recharger Apache

C'est le cœur du métier d'hébergeur web.

3.2 Structure standard d'un site web sous Linux

Tous les sites se trouvent dans :

```
/var/www/
```

Exemple :

```
/var/www/appweb
```

Dans ce dossier, on place au minimum :

- un fichier `index.html` ou `index.php`
- les dossiers `css`, `js`, `images`, etc.

Commande pour créer un dossier :

```
sudo mkdir -p /var/www/appweb
```

Vérifier :

```
ls -l /var/www
```

3.3 Permissions indispensables

Apache utilise un utilisateur système nommé : **www-data**

Ce compte doit être propriétaire de tous les fichiers exécutés par le serveur.

Commandes obligatoires :

```
sudo chown -R www-data:www-data /var/www/appweb
sudo chmod -R 755 /var/www/appweb
```

Pourquoi ?

- www-data doit pouvoir lire les fichiers pour les envoyer au navigateur.
- Le mode 755 permet à Apache de lire les fichiers mais empêche les utilisateurs non autorisés de les modifier.

3.4 Créer une page de test

Pour valider l'installation de PHP ou d'Apache, on utilise souvent :

```
sudo nano /var/www/appweb/index.php
```

Contenu :

```
<?php
phpinfo();
```

Après enregistrement, Apache interprétera ce fichier et affichera les informations PHP.

3.5 Localisation des fichiers Apache

Chemin	Rôle
/etc/apache2/apache2.conf	configuration principale
/etc/apache2/ports.conf	ports écoutés
/etc/apache2/sites-available/	VirtualHosts disponibles
/etc/apache2/sites-enabled/	VirtualHosts activés
/var/log/apache2/	logs d'erreur et d'accès

3.6 Création d'un VirtualHost complet

Un VirtualHost dit à Apache :

- quel domaine doit pointer vers quel dossier
- où écrire les logs
- quelles permissions appliquer

Créer le fichier :

```
sudo nano /etc/apache2/sites-available/appweb.conf
```

Contenu :

```
<VirtualHost *:80>
    ServerName appweb.local
    DocumentRoot /var/www/appweb

    <Directory /var/www/appweb>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/appweb_error.log
    CustomLog ${APACHE_LOG_DIR}/appweb_access.log combined
</VirtualHost>
```

Points à retenir :

- ServerName = le domaine (même local)
- DocumentRoot = dossier du site
- <Directory> = règles d'accès
- ErrorLog = erreurs
- CustomLog = logs d'accès

3.7 Activer / désactiver un VirtualHost

Activer :

```
sudo a2ensite appweb.conf
```

Désactiver :

```
sudo a2dissite appweb.conf
```

Désactiver le site par défaut :

```
sudo a2dissite 000-default.conf
```

3.8 Vérifier la configuration Apache

Cette commande est obligatoire avant tout rechargement :

```
sudo apache2ctl configtest
```

Résultat attendu : **Syntax OK**

Si erreur :

- retourner dans le fichier de configuration
- vérifier les balises
- vérifier les chemins
- vérifier les permissions

3.9 Recharger Apache

Rechargement :

```
sudo systemctl reload apache2
```

Redémarrage complet (rare) :

```
sudo systemctl restart apache2
```

Le rechargement est utilisé 95% du temps.

3.10 Résolution des erreurs Apache

Les erreurs classiques :

Erreur : 403 Forbidden

Causes possibles :

- permissions incorrectes
- propriétaire incorrect (www-data)
- dossier non accessible

Solutions :

```
sudo chown -R www-data:www-data /var/www/appweb
sudo chmod -R 755 /var/www/appweb
```

Erreur : 404 Not Found

Causes possibles :

- mauvaise route
- mauvais dossier configuré dans DocumentRoot
- fichier manquant (index.php absent)

Erreur : 500 Internal Server Error

Causes possibles :

- erreur PHP
- erreur de .htaccess
- directive Apache non autorisée

Solution : lire le log :

```
sudo tail -f /var/log/apache2/error.log
```

3.11 Rediriger un domaine vers un autre

Même si ce n'est pas obligatoire, c'est utile.

Configuration simple :

```
Redirect / https://nouveausite.fr/
```

À placer dans un VirtualHost HTTP.

3.12 .htaccess et AllowOverride

Le fichier .htaccess permet d'adapter les règles Apache dans un dossier spécifique.

Mais il ne fonctionne que si :

```
AllowOverride All
```

est activé dans <Directory>.

Pourquoi ?

- cela autorise les règles de réécriture (rewrite)
- utile pour les frameworks PHP
- indispensable pour certains projets Git

CHAPITRE 4 — Sécurisation du serveur Linux et d'Apache

Un hébergement web n'est pas complet si le serveur n'est pas sécurisé. Les techniques enseignées sont celles utilisées dans 90 % des entreprises.

4.1 Sécuriser SSH

SSH est le principal accès administrateur au serveur. Protéger cet accès est la priorité numéro 1.

4.1.1 Générer une clé SSH

Sur le poste client :

```
ssh-keygen
```

Appuyer sur Entrée à toutes les questions pour accepter les valeurs par défaut.

Résultat attendu :

```
Your identification has been saved in /home/user/.ssh/id_rsa
Your public key has been saved in /home/user/.ssh/id_rsa.pub
```

4.1.2 Envoyer la clé SSH au serveur

```
ssh-copy-id utilisateur@ADRESSE_IP
```

Si un message "Are you sure you want to continue connecting?" s'affiche : taper yes.

Résultat attendu :

```
Number of key(s) added: 1
```

4.1.3 Tester la connexion

```
ssh utilisateur@ADRESSE_IP
```

Le serveur ne doit plus demander de mot de passe.

4.1.4 Désactiver l'authentification par mot de passe

Sur le serveur :

```
sudo nano /etc/ssh/sshd_config
```

Modifier les lignes suivantes :

```
PasswordAuthentication no
PermitRootLogin no
```

Sauvegarder, puis :

```
sudo systemctl reload ssh
```

À ce stade : seule la clé SSH permet l'accès (bonne pratique obligatoire en entreprise).

4.2 Firewall UFW : configuration complète

UFW (Uncomplicated Firewall) permet de contrôler quels ports sont ouverts.

4.2.1 Règles de base

Autoriser SSH :

```
sudo ufw allow 'OpenSSH'
```

Autoriser HTTP :

```
sudo ufw allow 80
```

Autoriser HTTPS :

```
sudo ufw allow 443
```

Activer UFW :

```
sudo ufw enable
```

Vérifier :

```
sudo ufw status
```

4.2.2 Politique de sécurité stricte (Hardening)

Par défaut, on bloque tout ce qui entre :

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
```

Cela signifie :

- aucune connexion entrante n'est acceptée sauf celles autorisées
- toutes les connexions sortantes sont autorisées

4.3 Installer HTTPS via Certbot

HTTPS est essentiel car il :

- chiffre les communications
- protège les données
- améliore le référencement
- est obligatoire pour certains services

4.3.1 Installer Certbot

```
sudo apt install certbot python3-certbot-apache -y
```

4.3.2 Générer un certificat SSL

```
sudo certbot --apache -d mondomaine.fr -d www.mondomaine.fr
```

Étapes :

- saisir un email
- accepter les conditions
- choisir la redirection HTTPS automatique

Résultat attendu :

```
Congratulations! Your certificate and chain have been saved...
```

4.3.3 Vérifier HTTPS dans le navigateur

Accéder à :

```
https://mondomaine.fr
```

Le navigateur doit afficher un cadenas.

4.4 Permissions et propriétaires

4.4.1 Lire les permissions

```
ls -l /var/www/appweb
```

Exemple de sortie :

```
-rw-r--r-- 1 www-data www-data 418 oct 12 10:23 index.php
```

Lecture :

- propriétaire = www-data
- groupe = www-data
- droits = rw- r-- r--
- fichier lisible mais non modifiable par d'autres utilisateurs

---

4.4.2 Corriger les permissions d'un site entier

```
sudo chown -R www-data:www-data /var/www/appweb
sudo chmod -R 755 /var/www/appweb
```

Ces deux commandes résolvent 80 % des erreurs d'accès.

4.5 Hardening Apache

L'objectif est d'éviter que le serveur divulgue des informations inutiles.

4.5.1 Désactiver ServerSignature et ServerTokens

```
sudo nano /etc/apache2/conf-available/security.conf
```

Changer :

```
ServerSignature Off
ServerTokens Prod
```

Enregistrer et recharger Apache :

```
sudo systemctl reload apache2
```

4.5.2 Vérifier les ports ouverts

```
sudo ss -tulnp | grep apache2
```

Le serveur ne doit écouter que sur :

- 80
- 443

4.5.3 Désactiver le listing des dossiers

Dans le VirtualHost :

```
Options -Indexes
```

Cela empêche les visiteurs de voir les fichiers du site.

CHAPITRE 5 — Logs, debugging et résolution d'erreurs serveur

Un administrateur web n'est pas évalué sur tout ce qu'il sait installer, mais surtout sur sa capacité à comprendre pourquoi un site ne fonctionne pas. Ce chapitre entraîne l'identifier, lire, analyser et résoudre une erreur sur un serveur Linux / Apache / PHP / MySQL.

5.1 Pourquoi les logs sont indispensables ?

Quand un site web plante, trois éléments peuvent être en cause :

1. Apache (le serveur web)
2. PHP (le code qui s'exécute)
3. MySQL (la base de données)

Les logs permettent de comprendre exactement ce qui se passe.

Sans logs, on devine. Avec les logs, on diagnostique.

5.2 Les logs Apache

Chemin principal des logs :

```
/var/log/apache2/error.log
```

Commande pour lire les 20 dernières lignes :

```
sudo tail -n 20 /var/log/apache2/error.log
```

Commande pour suivre les logs en temps réel :

```
sudo tail -f /var/log/apache2/error.log
```

Action typique : ouvrir le fichier, rafraîchir le site → voir instantanément l'erreur.

5.2.1 Exemples d'erreurs Apache

Erreur : Permission denied

```
AH00035: access to /index.php denied (filesystem path '/var/www/appweb/index.php')
```

Cause :

- permissions incorrectes
- propriétaire incorrect

Solution :

```
sudo chown -R www-data:www-data /var/www/appweb
sudo chmod -R 755 /var/www/appweb
```

Erreur : File not found

```
AH00128: File does not exist: /var/www/appweb/favicon.ico
```

Cause : un fichier demandé n'existe pas.

Impact : souvent mineur.

Erreur : RewriteEngine not allowed

```
AH00526: Invalid command 'RewriteEngine', perhaps mis-spelled or defined by a module not included in the server configuration
```

Cause : module rewrite pas activé.

Solution :

```
sudo a2enmod rewrite
sudo systemctl reload apache2
```

5.3 Les logs PHP

PHP écrit ses erreurs dans Apache, mais peut aussi avoir un log dédié :

```
/var/log/php8.1-fpm.log
```

Exemple d'erreur PHP classique :

```
PHP Fatal error: Uncaught Error: Call to undefined function mysqli_connect()
```

Cause : extension manquante.

Solution :

```
sudo apt install php-mysql -y
sudo systemctl reload apache2
```

5.4 Les logs MySQL

Chemin :

```
/var/log/mysql/error.log
```

Lecture :

```
sudo tail -n 20 /var/log/mysql/error.log
```

Erreurs courantes :

1. Accès refusé

```
Access denied for user 'appuser'@'localhost'
```

Cause :

- mauvais mot de passe
- utilisateur non créé

Solution :

```
sudo mysql
ALTER USER 'appuser'@'localhost' IDENTIFIED BY 'NouveauMDP!';
FLUSH PRIVILEGES;
EXIT;
```

2. Base inexistante

```
Unknown database 'appweb'
```

Cause : base non créée.

Solution :

```
CREATE DATABASE appweb;
```

5.5 Erreurs HTTP : comment les analyser

Les étudiants doivent absolument savoir reconnaître les grandes familles d'erreurs.

5.5.1 Erreur 400 — Mauvaise requête

Cause : URL mal formée.

5.5.2 Erreur 403 — Forbidden

Cause :

- permissions
- propriétaire des dossiers incorrect
- règles Apache restrictives

Solution :

```
sudo chown -R www-data:www-data /var/www/appweb
sudo chmod -R 755 /var/www/appweb
```

5.5.3 Erreur 404 — Page non trouvée

Cause :

- fichier manquant
- mauvaise route
- DocumentRoot incorrect

À vérifier :

```
ls /var/www/appweb
```

5.5.4 Erreur 500 — Internal Server Error

Cause possible :

- erreur PHP
- erreur .htaccess
- module manquant

La solution = lire les logs :

```
sudo tail -f /var/log/apache2/error.log
```

5.5.5 Erreur 503 — Service indisponible

Cause : Apache crashé.

Solution :

```
sudo systemctl restart apache2
```

5.6 Debugging des VirtualHost

Commande :

```
sudo apache2ctl configtest
```

Résultat attendu : **Syntax OK**

Si erreur :

Exemple :

```
AH00526: Syntax error on line 10 of /etc/apache2/sites-enabled/appweb.conf:
Invalid command 'Documentroot'
```

Cause : faute de frappe → "Documentroot" au lieu de "DocumentRoot".

5.7 Debugging de PHP avec un fichier test

Créer un fichier :

```
sudo nano /var/www/appweb/test.php
```

Contenu :

```
<?php
echo "Test PHP OK";
```

Accéder dans le navigateur :

http://appweb.local/test.php

Si la page ne s'affiche pas → problème PHP.

5.8 Debugging MySQL avec un script PHP simple

Créer :

```
sudo nano /var/www/appweb/mysqltest.php
```

Contenu :

```
<?php
$conn = mysqli_connect("localhost", "appuser", "MotDePasseFort123!", "appweb");
if (!$conn) {
    die("Erreur MySQL : " . mysqli_connect_error());
}
echo "Connexion MySQL OK";
```

Si erreur → le message sera affiché.

CHAPITRE 6 — Architecture haute disponibilité & scalabilité

Durée estimée : 1h30 à 2h

6.1 Pourquoi la haute disponibilité ?

Un site web en production doit rester en ligne en permanence.

Un serveur unique peut tomber en panne à cause de :

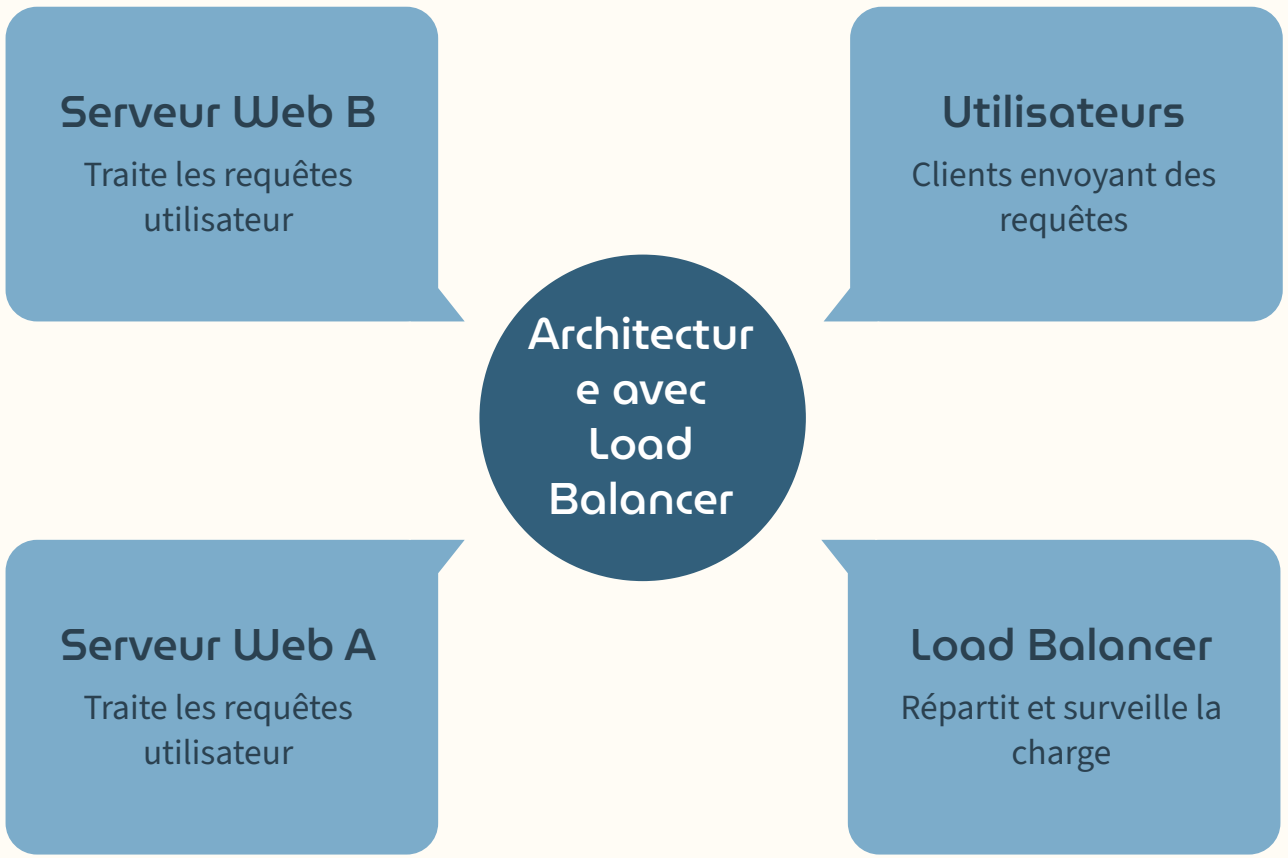
- surcharge
- coupure réseau
- mise à jour ratée
- crash matériel
- bug logiciel

La haute disponibilité (HA) consiste à mettre en place une architecture capable de continuer à fonctionner même si un composant tombe.

6.2 Le rôle du load balancer

Le load balancer est un serveur spécialisé chargé de répartir les requêtes entre plusieurs serveurs web.

Schéma simple :



Ses fonctions :

- distribuer la charge
- vérifier si un serveur tombe
- rediriger automatiquement le trafic

Il ne stocke aucun fichier : il ne fait que router.

6.3 Réplication de base de données

Une application web dynamique utilise une base MySQL. Si la base tombe, même si les serveurs web fonctionnent, le site tombe.

On crée donc une réplication :

```
[ Master DB ]
     |
[ Replica DB ]
```

Principes :

- Le "master" reçoit les écritures.
- Le "replica" reçoit une copie en temps réel.
- Si le master tombe, on bascule sur le replica.

6.4 Synchronisation des fichiers

Les fichiers d'un site ne peuvent pas être différents sur chaque serveur web.

Il faut donc une synchronisation :

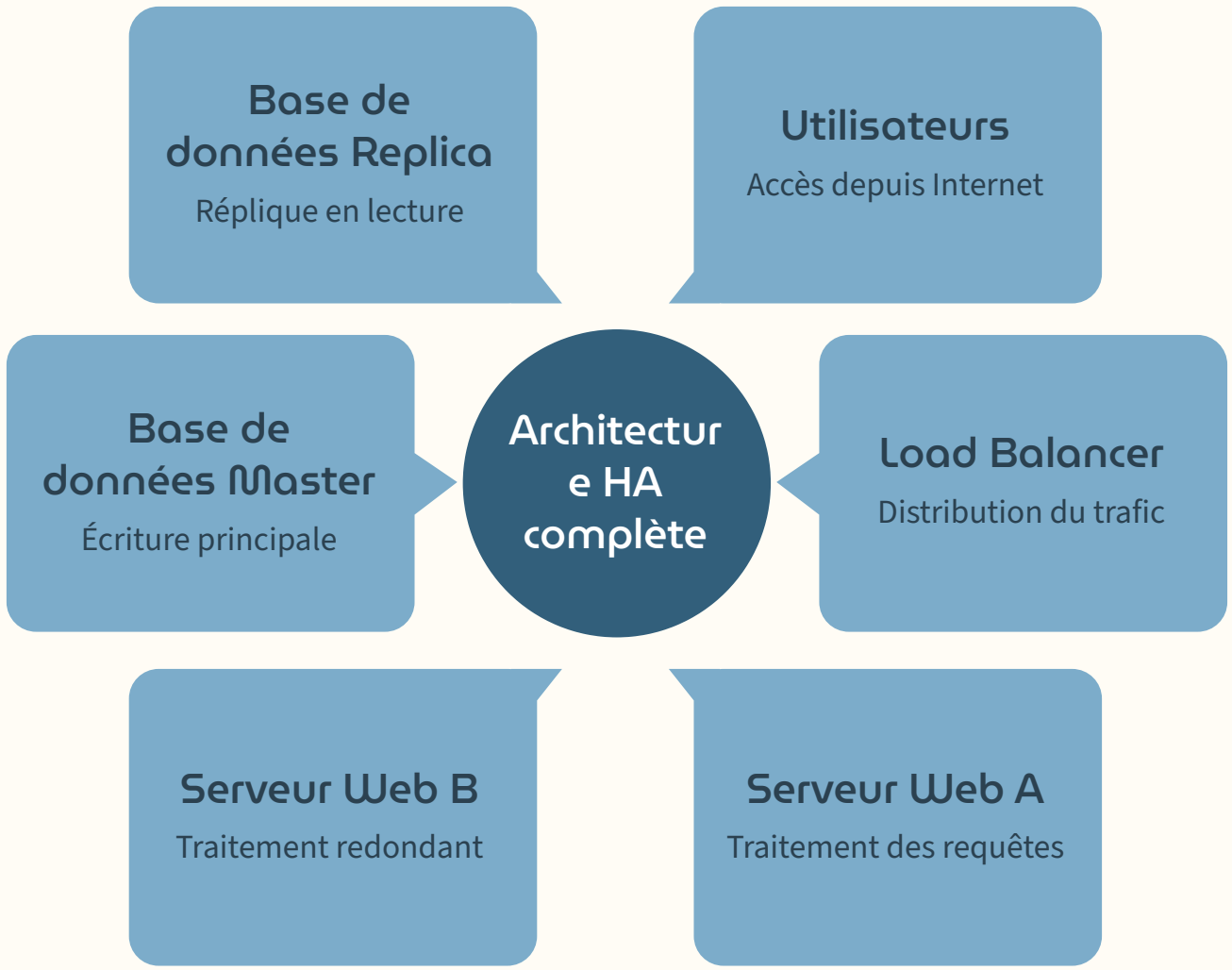
```
[ Serveur Web 1 ] ⟷ [ Serveur Web 2 ]
```

Solutions possibles (à connaître sans les configurer) :

- rsync
- stockage partagé
- NFS

6.5 Représentation complète d'une architecture HA simple

Schéma textuel complet :



Compétences à acquérir :

- reconnaître le rôle de chaque élément
- expliquer pourquoi cette architecture est plus robuste
- comprendre qu'un serveur peut tomber sans couper tout le service

6.6 Scalabilité horizontale (obligatoire – niveau compréhension)

Scalabilité horizontale = ajouter plus de serveurs web lorsque la charge augmente.

Exemple :

```
Load Balancer
 ├── Web 1
 ├── Web 2
 └── Web 3
```

Si les utilisateurs augmentent :

```
Load Balancer
 ├── Web 1
 ├── Web 2
 ├── Web 3
 └── Web 4 (ajout dynamique)
```

6.7 Scalabilité verticale (moins importante)

Scalabilité verticale = augmenter la puissance d'un seul serveur :

- plus de CPU
- plus de RAM
- plus de stockage

C'est plus simple, mais moins flexible.