Concepts d'architecture

Architecture client-serveur

Une base de données fonctionne selon une **architecture client-serveur**.

Le SGBD est un service qui :

- **Écoute sur un port réseau**
- **Accepte les connexions** des clients
- **Authentifie** les utilisateurs
- **Exécute** les requêtes SQL
- **Retourne** les résultats au client

Chaque SGBD utilise un port par défaut :

- **MySQL / MariaDB** : 3306
- **PostgreSQL** : 5432

Les clients peuvent être :

- Des applications web (PHP, Node.js, Python, etc.)
- Des outils en ligne de commande (`mysql`, `psql`)
- Des interfaces graphiques (phpMyAdmin, pgAdmin, DBeaver, etc.)

Sécurité de base

Deux paramètres de sécurité sont importants :

- **Bind address** : L'adresse IP sur laquelle le serveur écoute
  - `127.0.0.1` : Écoute uniquement en local (connexions depuis la même machine)
  - `0.0.0.0` : Écoute sur toutes les interfaces (accessible depuis le réseau)
- **Firewall** : Règles de pare-feu limitant l'accès au port de la base de données

> Par défaut, il est recommandé de n'autoriser que les machines nécessaires à se connecter à la base de données. Ne jamais exposer directement une base de

> données sur Internet sans protection.

Cas particulier : SQLite

SQLite est un SGBD qui se distingue radicalement des bases de données traditionnelles comme MySQL ou PostgreSQL. Contrairement à ces dernières, SQLite **n'utilise pas d'architecture client-serveur**.

Une base de données embarquée

SQLite fonctionne comme une **bibliothèque intégrée** directement dans l'application :

- **Pas de serveur** : Aucun processus séparé à démarrer ou gérer
- **Pas de port réseau** : Aucune configuration réseau nécessaire
- **Pas d'authentification** : L'accès est contrôlé par les permissions du système de fichiers
- **Un seul fichier** : Toute la base de données est stockée dans un fichier unique (ex: `ma_base.db`)

Différences avec un SGBD traditionnel

Caractéristique	MySQL / PostgreSQL	SQLite
Architecture	Client-serveur	Embarquée (bibliothèque)
Processus serveur	Oui	Non
Port réseau	3306 / 5432	Aucun
Stockage	Répertoire dédié	Fichier unique
Accès concurrent	Multi-utilisateurs	Limité (verrouillage fichier)
Accès réseau	Oui (natif)	Non
Configuration	Complexe	Aucune

Cas d'usage

SQLite est idéal pour :

- **Applications mobiles** (Android, iOS)
- **Applications de bureau** (navigateurs web, lecteurs multimédia)
- **Prototypage** et développement local
- **Tests unitaires**

- **Petits sites web** à faible trafic

> SQLite est le moteur de base de données le plus déployé au monde. Il est intégré dans Android, iOS, tous les navigateurs web, et de nombreuses applications.

> SQLite n'est **pas adapté** aux applications web à fort trafic ou nécessitant des accès concurrents multiples. Dans ces cas, privilégiez MySQL, MariaDB ou PostgreSQL.

Déploiement avec Docker

Les SGBD peuvent être facilement déployés via Docker, ce qui offre plusieurs avantages :

- **Portabilité** : Même environnement en développement et production
- **Isolation** : Chaque base de données dans son propre conteneur
- **Simplicité** : Déploiement rapide sans installation complexe
- **Versioning** : Contrôle précis de la version du SGBD

Images Docker officielles

Les SGBD relationnels majeurs disposent d'images officielles sur Docker Hub :

- **MySQL** : hub.docker.com/_/mysql
- **MariaDB** : hub.docker.com/_/mariadb
- **PostgreSQL** : hub.docker.com/_/postgres

Exemple de lancement d'une base MySQL avec Docker :

```
docker run -d \
  --name ma-base-mysql \
  -e MYSQL_ROOT_PASSWORD=motdepasse \
  -e MYSQL_DATABASE=ma_base \
  -p 3306:3306 \
  mysql:latest
```

Persistance des données

Comme vu dans le cours Docker, les conteneurs sont volatiles. Pour persister les données de la base, il est **indispensable** d'utiliser des **volumes Docker**.

Exemple avec un volume :

```
docker run -d \
  --name ma-base-mysql \
  -e MYSQL_ROOT_PASSWORD=motdepasse \
  -v mysql-data:/var/lib/mysql \
  -p 3306:3306 \
  mysql:latest
```

> Le volume `mysql-data` stocke les fichiers de la base de données. Même si le conteneur est supprimé, les données persistent.

Le théorème de CAP

Le **théorème de CAP** est un concept fondamental en architecture distribuée. Il stipule qu'un système de base de données distribuée ne peut garantir simultanément que **deux des trois propriétés** suivantes :

- **C**onsistency (Cohérence) : Tous les nœuds voient les mêmes données au même moment
- **A**vailability (Disponibilité) : Le système répond toujours, même en cas de panne partielle
- **P**artition tolerance (Tolérance au partitionnement) : Le système continue de fonctionner même si des nœuds ne peuvent plus communiquer entre eux

En pratique :

- **Les SGBD relationnels** privilégient la **cohérence** (C) et la **disponibilité** (A), au détriment de la tolérance au partitionnement. Ils fonctionnent mieux dans un environnement centralisé.
- **Les bases NoSQL** privilégient souvent la **disponibilité** (A) et la **tolérance au partitionnement** (P), acceptant une cohérence éventuelle (eventual consistency).

Réplication : architecture Maître / Esclave

L'architecture **maître / esclave** (aussi appelée **primary / replica**) est une technique de réplication permettant d'améliorer les performances et la disponibilité.

Principe

Un **serveur maître** :

- Reçoit toutes les requêtes d'**écriture** (INSERT, UPDATE, DELETE)
- Peut également traiter les requêtes de **lecture** (SELECT)

Un ou plusieurs **serveurs esclaves** :

- Répliquent les données du maître en **lecture seule**
- Ne traitent que des requêtes de **lecture** (SELECT)
- Se synchronisent automatiquement avec le maître

Avantages

- **Performances** : Répartition de la charge de lecture sur plusieurs serveurs (load balancing)
- **Disponibilité** : En cas de panne du maître, un esclave est disponible
- **Sauvegardes** : Possibilité de faire des backups sur un esclave sans impacter le maître
- **Scalabilité en lecture** : Ajout de serveurs esclaves pour absorber plus de requêtes de lecture

Concepts avancés

Quelques concepts liés à la réplication :

- **Failover** : Basculement automatique vers un esclave en cas de panne du maître
- **Haute disponibilité** : Garantir un service continu même en cas de défaillance
- **Load balancing** : Répartition intelligente des requêtes de lecture entre les esclaves
- **Réplication asynchrone vs synchrone** : Délai de synchronisation entre maître et esclaves

---

Revision #5
Created 1 January 2026 10:35:56 by Thibaud FRICHET
Updated 15 January 2026 12:17:53 by Thibaud FRICHET