**Résumé du Chapitre 1**

**Approfondir la notion Client/Serveur**

**1. Principe Client/Serveur**

* **Définition** :
  + **Client** : Consommateur de services qui émet des requêtes (ex : navigateur web, client FTP).
  + **Serveur** : Fournisseur de services (matériel/logiciel) qui traite les requêtes et renvoie des réponses (ex : serveur web, SGBD).
  + **Middleware** : Logiciel médiateur qui gère la communication entre client et serveur (ex : conversion, sécurité, adressage).
* **Fonctionnement** :
  + Le client envoie une requête via un port spécifique (ex : port 80 pour HTTP).
  + Le serveur traite la requête et renvoie une réponse (codes HTTP : 200 = succès, 404 = erreur client, etc.).
  + Protocoles standardisés (RFC) garantissent l'interopérabilité.
* **Avantages/Inconvénients** :
  + ✅ Centralisation des ressources, sécurité, évolutivité.
  + ❌ Coût élevé, dépendance au réseau, maintenance complexe.

**2. Architectures 2-tiers, 3-tiers, et n-tiers**

* **Architecture 1-tiers** :
  + Toutes les couches (présentation, traitement, données) sur une même machine (ex : logiciel local avec base SQLite).
* **Architecture 2-tiers** :
  + **Client** : Interface utilisateur (IHM).
  + **Serveur** : Logique métier + base de données (ex : application avec MySQL distant).
  + **Limites** : Trafic réseau important, faible extensibilité.
* **Architecture 3-tiers** :
  + Séparation en 3 couches :
    1. **Client** (navigateur).
    2. **Serveur applicatif** (traitement métier, ex : Node.js).
    3. **Serveur de données** (SGBD, ex : PostgreSQL).
  + **Exemple** : Site e-commerce (Amazon).
* **Architecture n-tiers** :
  + Extension des 3-tiers avec des serveurs spécialisés (ex : microservices).
  + **Avantages** : Répartition de charge, maintenance simplifiée, réutilisation des services.
* **Comparatif** :

| **Critère** | **2-tiers** | **3/n-tiers** |
| --- | --- | --- |
| **Sécurité** | Faible | Élevée |
| **Performance** | Limitée | Optimisée |
| **Extensibilité** | Rigide | Flexible |

**3. Types de Serveurs Web**

* **Définition** :
  + Logiciel/hardware qui diffuse des contenus via HTTP/HTTPS (port 80/443).
  + **Exemples** : Apache, Nginx, IIS.
* **Serveur Statique vs Dynamique** :
  + **Statique** : Envoie des fichiers "tels quels" (HTML, CSS).
  + **Dynamique** : Traite les requêtes avant réponse (ex : PHP + MySQL).
* **Clients** : Navigateurs (Chrome, Firefox, etc.) qui envoient des requêtes (méthodes GET/POST).
* **Réponses du Serveur** :
  + Codes HTTP (200 OK, 404 Not Found, 500 Server Error).
  + Structure : En-tête (métadonnées) + corps (contenu HTML/JSON).

**Points Clés à Retenir**

1. **Client/Serveur** : Modèle basé sur requête/réponse avec des protocoles standardisés.
2. **Architectures** :
   * 2-tiers = simple mais limité ; 3/n-tiers = scalable et sécurisé.
3. **Serveurs Web** : Dynamiques pour les sites interactifs, statiques pour les contenus fixes.
4. **Codes HTTP** : Indiquent le statut des requêtes (succès, erreur, redirection).

**Exemples Concrets** :

* 2-tiers : Application de gestion avec client lourd + SGBD.
* 3-tiers : Site web avec backend PHP et base de données.
* Middleware : API REST pour connecter une app mobile à un serveur.