📚 Référence : Les grandes architectures logicielles

> **Document de référence** — À consulter quand vous rencontrez ces architectures en entreprise ou en stage.
> Ce document complète le CM1 qui se concentre sur les **principes fondamentaux**.

🗺️ Vue d'ensemble

Architecture	Idée principale	Quand l'utiliser
Monolithique	Tout en un bloc	Petits projets, MVP, prototypes
N-tiers / Couches	Séparation par responsabilités horizontales	Applications classiques, équipes moyennes
MVC / MVVM	Séparation UI / logique / données	Interfaces utilisateur (web, mobile)
SOA	Services métier mutualisés	Grands SI d'entreprise
Microservices	Petits services autonomes	Grandes équipes, forte scalabilité
Event-Driven	Réaction à des événements	Systèmes asynchrones,

1. Architecture monolithique

Principe

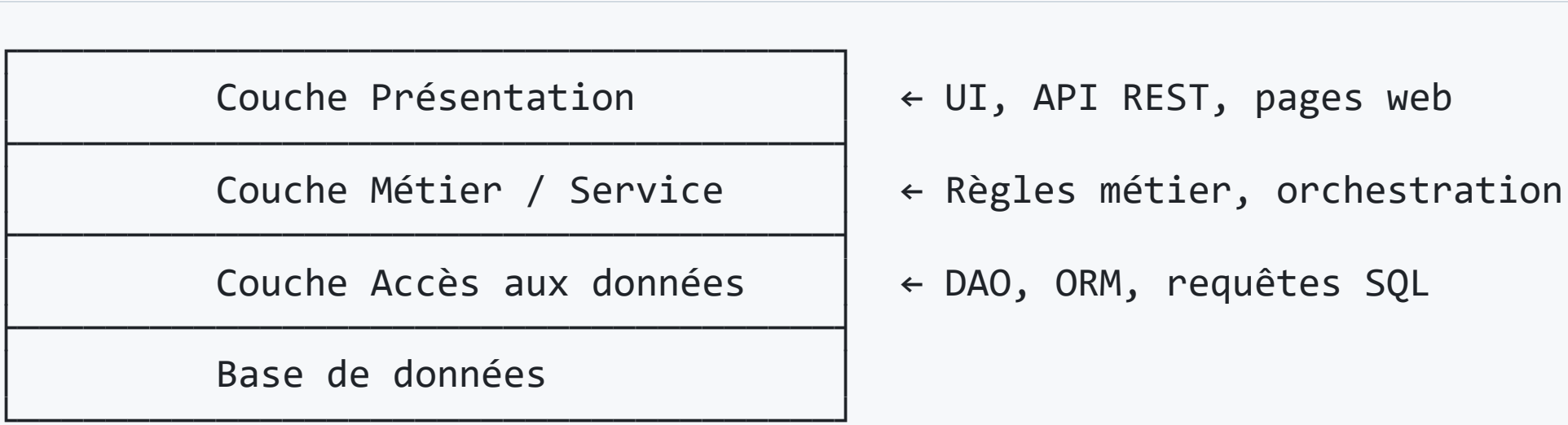
Tout le code de l'application est rassemblé dans **un seul artefact** déployable (un `.jar`, un `.war`, un binaire...).

Avantages

- ✅ **Simple à démarrer** : pas de complexité réseau, pas d'orchestration
- ✅ **Déploiement facile** : un seul artefact à livrer
- ✅ **Debugging simple** : tout est dans le même processus
- ✅ **Performance** : pas de latence réseau entre composants

2. Architecture en couches (Layered / N-tiers)

Principe



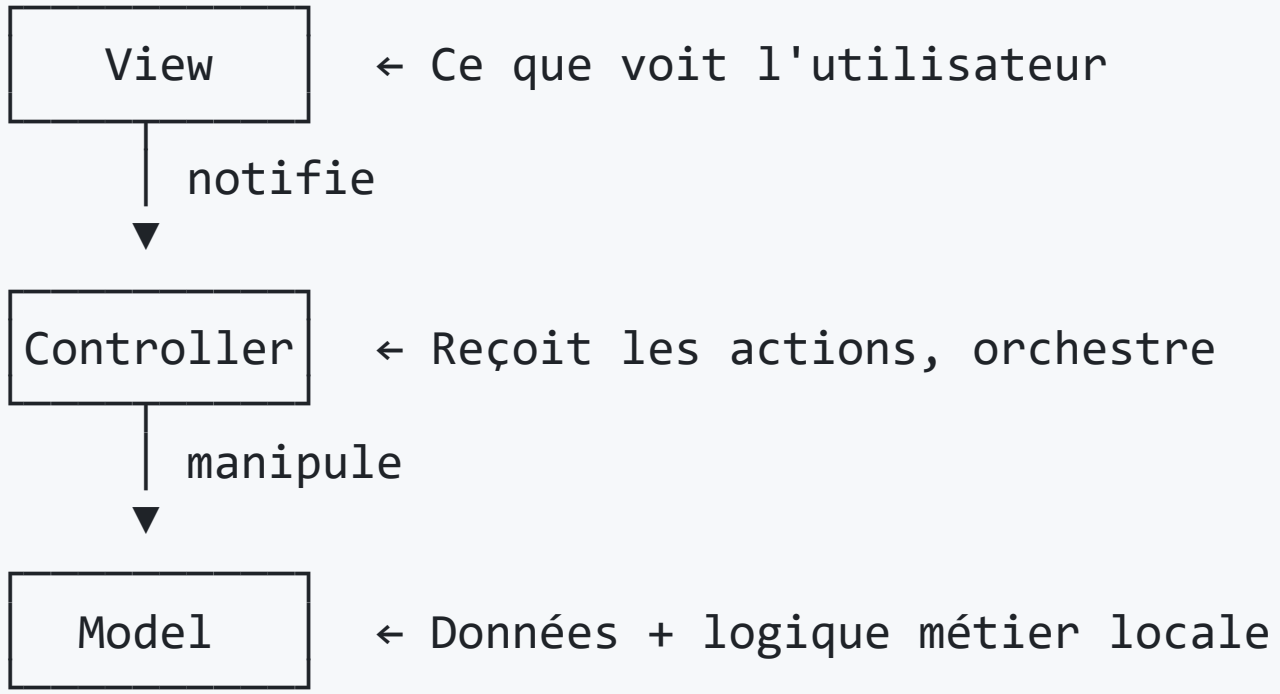
Chaque couche a une **responsabilité** et ne communique qu'avec les couches adjacentes.

Variantes

- **3-tiers** : Présentation → Métier → Données

3. MVC (Model – View – Controller) / MVVM

Principe MVC

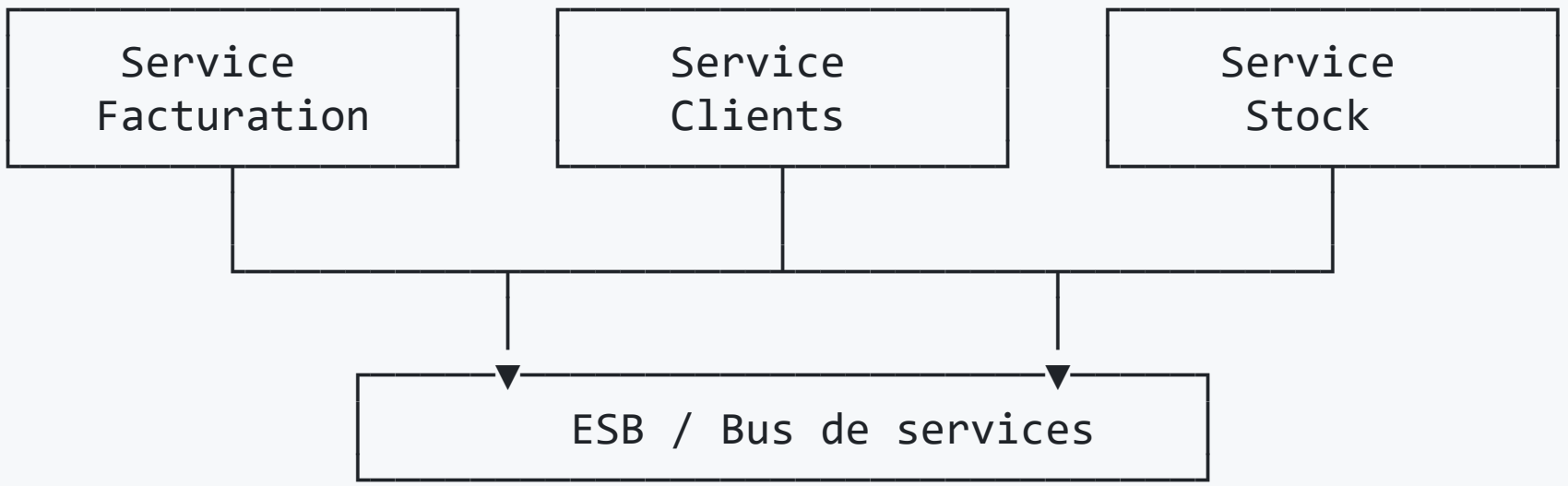


Variante MVVM (Model – View – ViewModel)

Populaire en développement mobile et front-end moderne :

4. SOA (Service-Oriented Architecture)

Principe

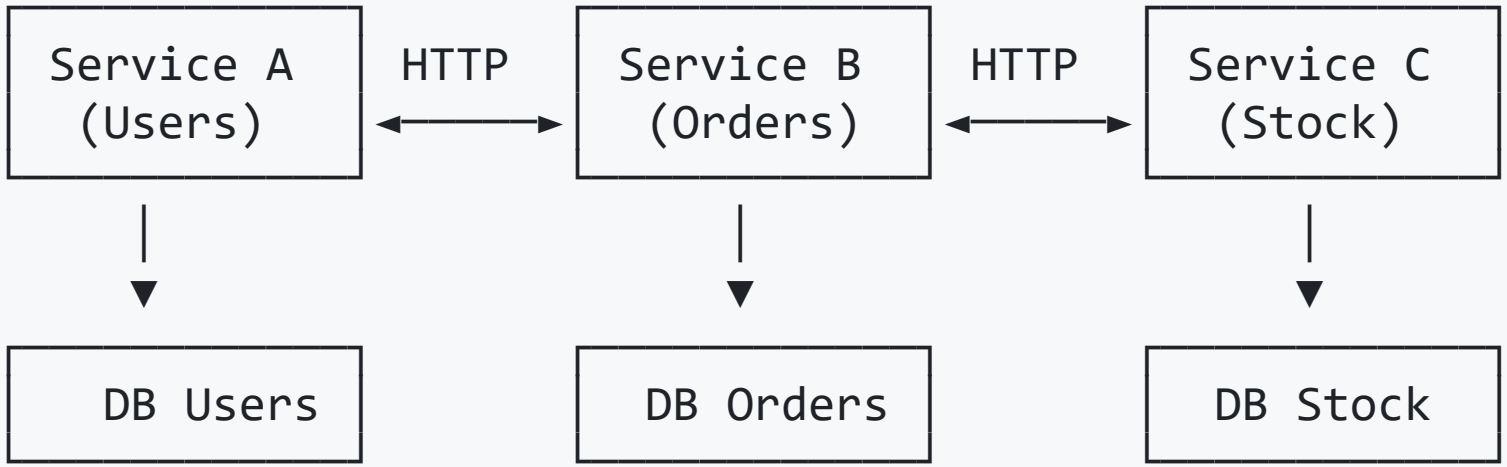


Chaque **service** correspond à un grand domaine métier. Les services communiquent via un **bus** (ESB = Enterprise Service Bus).

Caractéristiques

5. Microservices

Principe

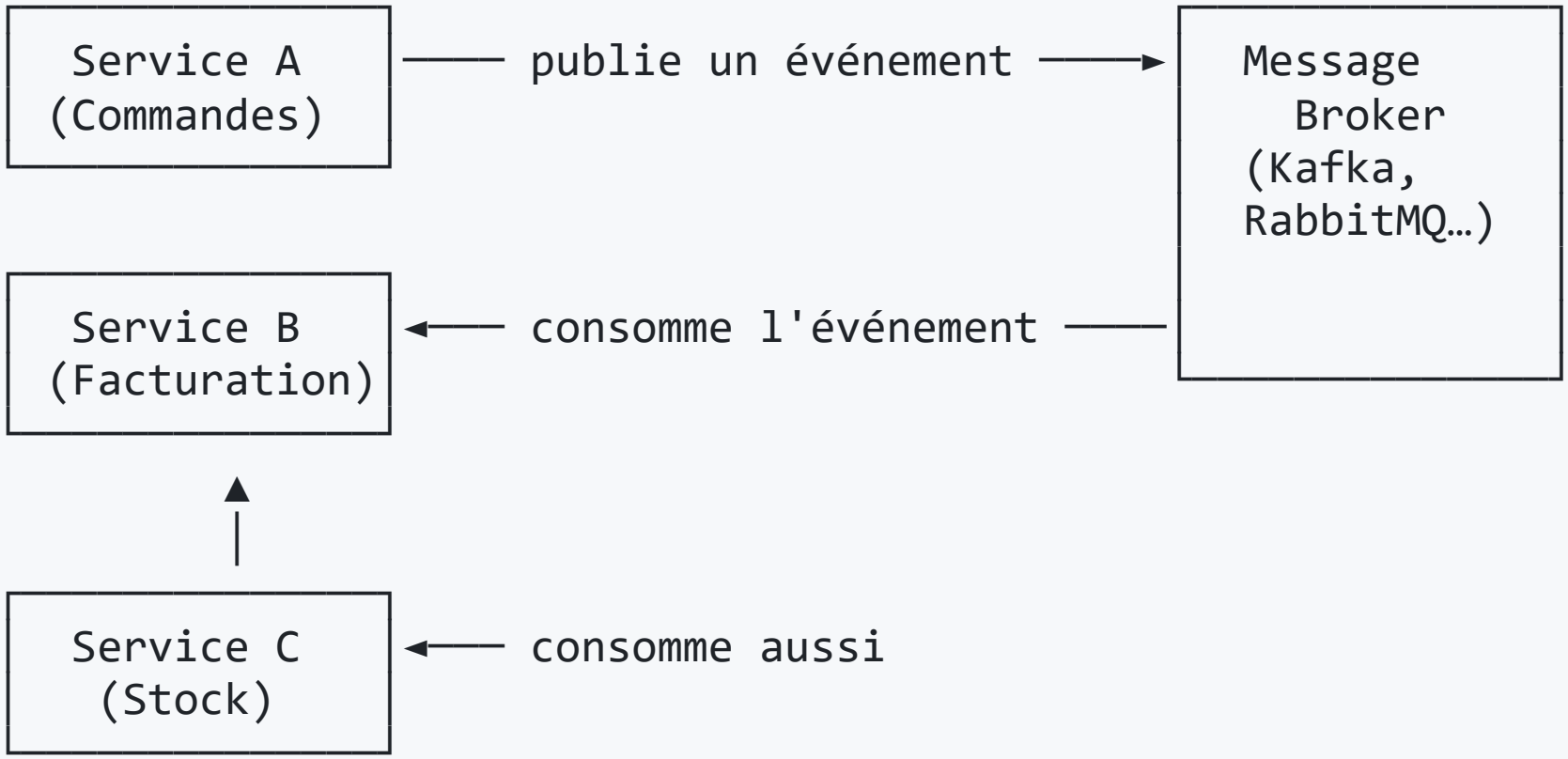


Chaque **microservice** est une petite application autonome :

- Son propre code
- Sa propre base de données
- Son propre déploiement

6. Event-Driven Architecture (EDA)

Principe



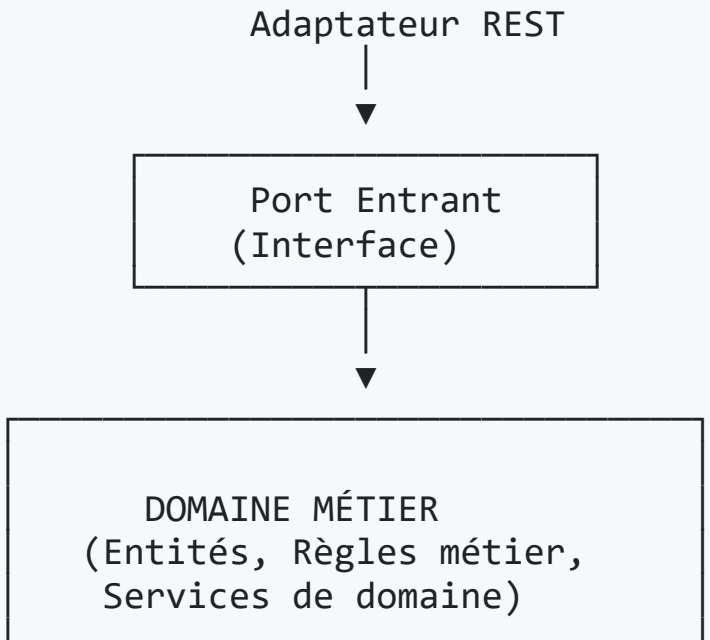
Les services communiquent via des **événements** plutôt que des appels directs.

7. Architectures centrées domaine (Hexagonale / Clean / Onion)

Principe commun

> **Le code métier (domaine) est au centre et ne dépend de rien d'externe.**
> C'est la technique (framework, base de données, UI) qui dépend du domaine.

Architecture Hexagonale (Ports & Adapters)



🎯 Tableau récapitulatif : comment choisir ?

Critère	Monolithe	N-tiers	Microservices	Hexagonale
Taille d'équipe	Petite	Moyenne	Grande	Toute
Complexité métier	Faible	Moyenne	Variable	Forte
Besoin de tests	Basique	Moyen	Fort	Très fort
Scalabilité	Limitée	Moyenne	Forte	Dépend infra
Courbe apprentissage	Faible	Faible	Forte	Moyenne

📖 Pour aller plus loin

- **Clean Architecture** — Robert C. Martin (Uncle Bob)
- **Domain-Driven Design** — Eric Evans
- **Building Microservices** — Sam Newman
- **Patterns of Enterprise Application Architecture** — Martin Fowler

> 💡 **Rappel** : Ces architectures ne sont pas mutuellement exclusives.
> On peut avoir un **monolithe** structuré en **hexagonal**, ou des **microservices** communiquant via **events**.
> L'important est de comprendre les **principes** pour faire des choix éclairés.