Architecture DataEnginerFlow360 - Optimisée Sans Redondances

🎯 Stack Final Validé

COUCHE 1 : SOURCES DE DONNÉES

```
✅ MySQL       → Base transactionnelle (source)
✅ PostgreSQL  → Base analytique (source)
✅ MongoDB     → Base NoSQL (source)
✅ API/IoT     → Flux externes
✅ Fichiers    → CSV, JSON, Parquet
```

Justification : Diversité nécessaire pour simuler un environnement réel multi-sources.

COUCHE 2 : INGESTION

Streaming (temps réel)

```
✅ Apache Kafka       → Message broker (standard industrie)
✅ Kafka Connect      → Connecteurs sources (MySQL, Postgres, Mongo)
✅ Zookeeper          → Coordination Kafka
```

Batch (traitement par lots)

```
✅ Python Scripts     → Ingestion fichiers + API
  - pandas
  - Faker (données synthétiques)
  - requests
```

✅ **PAS DE REDONDANCE** : Kafka pour streaming, Python pour batch.

COUCHE 3 : ORCHESTRATION

```
✅ Apache Airflow     → Orchestration pipelines BATCH
  - DAGs (workflows)
  - Scheduling
  - Monitoring pipelines
```

Note : Kafka Streams fait son propre orchestration pour le temps réel.

✅ **PAS DE REDONDANCE** : Un seul orchestrateur batch.

COUCHE 4 : TRAITEMENT (PROCESSING)

Streaming

✅ Kafka Streams → Traitement temps réel (agrégations, filtres)

Batch - Big Data

✅ PySpark → Traitement gros volumes (>100 GB)
- Transformations complexes
- Machine Learning
- Agrégations massives

Batch - Data Warehouse

✅ dbt → Transformations SQL (analytics)
- Schéma étoile
- Data Marts
- Tests qualité données

✅ **PAS DE REDONDANCE :**

- Kafka Streams → temps réel
- PySpark → gros volumes Data Lake
- dbt → transformations SQL Data Warehouse

COUCHE 5 : STOCKAGE

Data Lake (données brutes)

✅ Hadoop HDFS → Stockage distribué fichiers
✅ Apache Hive → Requêtes SQL sur HDFS
✅ PySpark → Traitement sur HDFS

Data Warehouse (données structurées)

✅ PostgreSQL → Base relationnelle (schéma étoile)
✅ dbt → Transformations SQL

✅ **PAS DE REDONDANCE :**

- HDFS + Hive = ensemble cohérent (Hive utilise HDFS en backend)
- PostgreSQL = Data Warehouse séparé

❌ **RETIRÉ** : S3 (redondant avec HDFS dans contexte local/learning)

COUCHE 6 : MONITORING

Métriques système (temps réel)

```
✅ Prometheus      → Collecte métriques (CPU, RAM, latences)
✅ Grafana         → Dashboards métriques temps réel
```

Logs applicatifs

```
✅ Elasticsearch   → Indexation logs
✅ Logstash        → Parsing et enrichissement logs
✅ Kibana          → Visualisation logs et recherche
```

✅ **PAS DE REDONDANCE** :

- Prometheus/Grafana → MÉTRIQUES numériques
- ELK Stack → LOGS textuels

Pourquoi les deux ?

- Grafana : "Mon CPU est à 80%, ma latence Kafka = 50ms"
- Kibana : "Voici l'erreur exacte : ConnectionTimeout line 245"

COUCHE 7 : CI/CD & INFRASTRUCTURE

```
✅ GitHub Actions  → CI/CD (tests, build, deploy)
✅ Docker          → Containerisation
✅ Docker Compose  → Orchestration multi-conteneurs
✅ Vault           → Gestion secrets (passwords, API keys)
```

Optionnel selon déploiement :

```
⚠️ Terraform       → IaC (si déploiement cloud AWS/GCP)
⚠️ Kubernetes      → Si besoin auto-scaling production
```

✅ **PAS DE REDONDANCE** : Chaque outil a un rôle unique.

🗂️ Bases de Données - Clarification

Vous avez **5 PostgreSQL/MySQL/MongoDB** mais **PAS de redondance** :

Base	Rôle	Peut être factorisé ?
MySQL	Source de données à ingérer	❌ Non (source externe)
MongoDB	Source NoSQL à ingérer	❌ Non (source externe)
PostgreSQL-DWH	Data Warehouse (business data)	❌ Non (production data)
PostgreSQL-Airflow	Métadonnées Airflow	✅ Oui (si ressources limitées)
PostgreSQL-Hive	Métadonnées Hive	✅ Oui (si ressources limitées)

Optimisation possible (si RAM limitée) :

```sql
-- Fusionner en 1 seul PostgreSQL avec 3 databases :
PostgreSQL unique:
  ├─ database: datawarehouse     (DWH production)
  ├─ database: airflow_metadata  (Airflow)
  └─ database: hive_metastore    (Hive)
```

📋 RÉCAPITULATIF : Redondances ?

Composant	Redondant ?	Action
HDFS + Hive	✅ Non	Gardez (complémentaires)
HDFS + S3	❌ Oui	Retirez S3 (choisissez HDFS)
Kafka Streams + Airflow	✅ Non	Gardez (streaming vs batch)
PySpark + dbt	✅ Non	Gardez (Data Lake vs DWH)
Kibana + Grafana	✅ Non	Gardez (logs vs métriques)
ELK + Prometheus	✅ Non	Gardez (complémentaires)
3 PostgreSQL	✅ Non	Usages différents (fusionnable si besoin)

🎯 VERDICT FINAL

✅ Votre stack est bien pensé !

Seule vraie redondance à corriger :

- ❌ **Retirer S3** (si vous utilisez HDFS)
- ✅ **Garder tout le reste**

Optimisations optionnelles (si ressources limitées) :

1. Fusionner les 3 PostgreSQL en 1 instance (3 databases)
2. Choisir **soit** ELK Stack **soit** Prometheus/Grafana (mais mieux de garder les deux)
3. Retirer Terraform si pas de déploiement cloud

💡 Recommandation Pédagogique

Pour votre projet d'apprentissage, je recommande :

Version Minimale (démarrage) :

```
- Kafka (streaming)
- Airflow (orchestration)
- PySpark + HDFS + Hive (Data Lake)
- PostgreSQL + dbt (DWH)
- Prometheus + Grafana (monitoring)
- GitHub Actions + Docker
```

Version Complète (projet final) :

Ajoutez progressivement :

```
+ ELK Stack (logs)
+ Vault (sécurité)
+ Kafka Connect (facilite ingestion)
+ MongoDB/MySQL sources
```

📝 Commande de démarrage optimisée

Pour démarrer uniquement les services essentiels :

bash

```bash
# Version minimale
docker-compose up -d \
  namenode datanode hive-server \
  zookeeper kafka \
  postgres-dwh \
  airflow-webserver airflow-scheduler \
  spark-master spark-worker \
  prometheus grafana

# Ajoutez ensuite si besoin
docker-compose up -d elasticsearch kibana logstash vault
```

Besoin que je modifie le `docker-compose.yml` pour retirer S3 ou fusionner les PostgreSQL ?