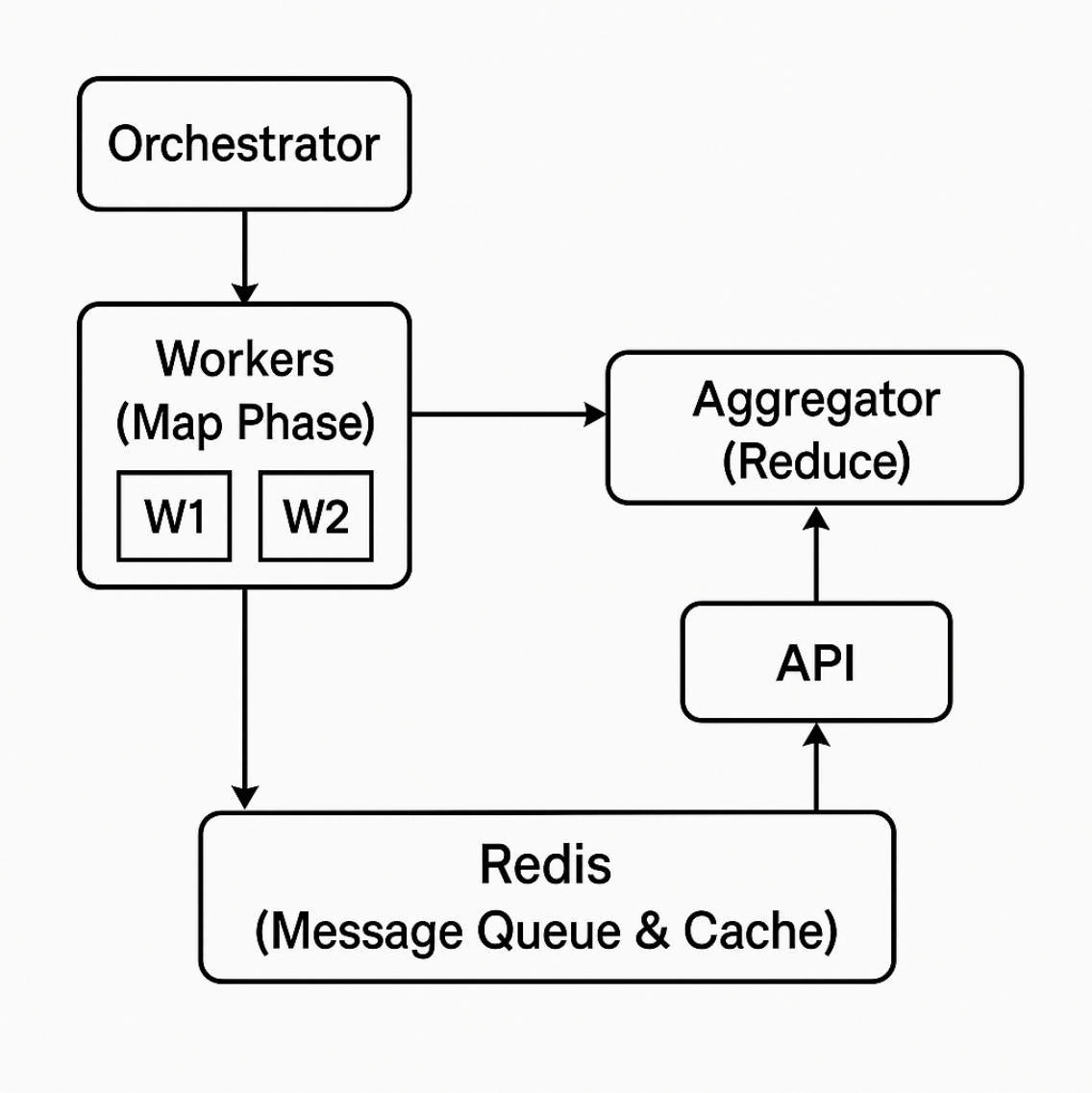
CHINE Dmitri - GALLIC Axel M2 AL 23 juin 2025

Schéma d'architeture



1. Justification du modèle Map-Reduce

Nous avons choisi le modèle Map-Reduce car:

- Il permet une parallélisation efficace des traitements de données
- Il s'adapte naturellement au traitement par ville (chaque ville peut être traitée séparément)
- Il facilite l'ajout de nouvelles villes sans modification du code
- Il est hautement scalable (ajout simple de workers)
- Il sépare clairement les responsabilités entre traitement parallèle et agrégation

2. Découpage des Conteneurs

1. Orchestrator

Charge les données depuis le fichier CSV

Distribue les tâches aux workers

Gère le cycle de vie du traitement

2. Workers (3+ instances)

Exécutent les calculs en parallèle sur un sous-ensemble de données

Traitent chaque lot de transactions indépendamment

Stockent les résultats intermédiaires dans Redis

3. Aggregator

Collecte et combine tous les résultats intermédiaires

Calcule les métriques finales

Stocke les résultats dans Redis pour accès par l'API

4. Documentation de l'API

L'API expose plusieurs endpoints pour accéder aux données analysées :

Endpoints disponibles

GET `/api/villes`

Récupère la liste des villes présentes dans les données.

Réponse :

```
{
  "villes": ["Paris", "Lyon", "Marseille", "..."]
}
```

GET `/api/ca-mensuel`

Récupère le chiffre d'affaires mensuel par ville.

Paramètres optionnels :

- `ville` : Filtre les résultats pour une ville spécifique
- `mois` : Filtre par mois au format YYYY-MM

Exemple : `/api/ca-mensuel?ville=Paris&mois=2023-01`

GET `/api/repartition`

Récupère la répartition vente/location par ville.

Paramètres optionnels :

- `ville` : Filtre les résultats pour une ville spécifique
- `format` : Si égal à "percentage", renvoie les pourcentages plutôt que les comptages bruts

Exemple : `/api/repartition?ville=Lyon&format=percentage`

GET `/api/top-modeles`

Récupère les modèles de véhicules les plus populaires par ville.

Paramètres optionnels :

- `ville` : Filtre les résultats pour une ville spécifique

Exemple : `/api/top-modeles?ville=Marseille`

POST `/api/process`

Déclenche un nouveau traitement des données.

Réponse :

```
{
  "status": "processing_started",
  "job_id": "uuid-généré"
}
```

Exemples d'utilisation

```
# Obtenir la liste des villes
curl http://localhost:5000/api/villes

# Obtenir le CA mensuel pour Paris en janvier 2023
curl http://localhost:5000/api/ca-mensuel?ville=Paris&mois=2023-01

# Lancer un nouveau traitement des données
curl -X POST http://localhost:5000/api/process
```

5. Redis

Sert de système de messagerie entre les composants

Stocke temporairement les données intermédiaires

Stocke les résultats finaux pour un accès rapide

3. Déployer l'architecture

```
autoconnect/
├── docker-compose.yml
```

```
├── data/
│   └── transactions_autoconnect.csv
├── orchestrator/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── main.py
├── worker/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── main.py
├── aggregator/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── main.py
└── api/
    ├── Dockerfile
    ├── requirements.txt
    └── main.py
```

Lancement

Pour lancer l'architecture, placez-vous dans le répertoire de projet et exécutez la commande suivante :

```
docker-compose build
docker-compose up -d
```

Vérification

Pour vérifier que tout fonctionne correctement, accédez à l'API via l'URL suivante :

```
# Vérifier que tous les conteneurs sont en cours d'exécution
docker-compose ps

# Consulter les log`
docker-compose logs -f
```