

Rapport — Agent planificateur ETL / Data Pipeline

Devoir 3 — Architecture de l’agent et évaluation promptfoo

1. Narration fonctionnelle de l’agent

1.1 Objectif

L’agent **planificateur** transforme une **intention en langage naturel** en livrables techniques : DAG Airflow, code ETL, tests automatisés, documentation. L’utilisateur décrit son objectif (ex. « Je veux ingérer une API, nettoyer les données, les stocker dans BigQuery et générer un dashboard ») ; l’agent produit des artefacts cohérents entre eux et reproductibles, sans inventer hors périmètre.

1.2 Entrées

Entrée principale : une chaîne en langage naturel (user_goal). Exemples : « Ingérer une API, nettoyer les données, les stocker dans BigQuery et générer un dashboard » ; « Ingérer les données d’une API REST et les écrire en CSV » ; « Connecter une base, faire un nettoyage et publier les résultats ».

1.3 Sorties

Réponse structurée en cinq sections obligatoires :

Section	Contenu
1. DAG Airflow	Code ou description du DAG (tâches, ordre, schedule).
2. Code ETL	Extraits ou logique des scripts (extraction, transformation, chargement).
3. Tests	Cas de test et commandes d’exécution.
4. Documentation	Usage, configuration, schémas.
5. Points à configurer / incertitudes	Placeholders [À PRÉCISER], [CONNECTION_*].

1.4 Règles de priorisation

Ordre de génération : DAG → ETL → Tests → Documentation. Cohérence inter-artefacts : mêmes noms de tâches et de fichiers entre DAG, ETL, tests et doc. En cas d’ambiguïté : placeholders explicites et section « À configurer » plutôt qu’invention.

2. Itérations promptfoo (modifications, contraintes, exemples)

Configuration : un seul prompt contenant rôle système, instructions, outils, protocoles et gestion des erreurs ; variable `{{user_goal}}` pour les tests.

Contraintes : pas de librairies inventées (usage de noms courants : apache-airflow, pandas, google-cloud-bigquery, pytest) ; format de sortie strict (sections `## 1. ... ## 5.`) ; marquage [À PRÉCISER] en cas d'ambiguïté.

Exemples de cas : (1) Cas nominal — API → nettoyage → BigQuery → dashboard ; (2) Cas minimal — API REST → CSV ; (3) Cas ambigu — « Connecter une base, nettoyage, publier » pour vérifier la gestion des imprécisions.

Évolutions possibles : assertions contains sur les titres de sections ; LLM-as-a-judge pour cohérence ; nouveaux cas limites.

3. Mécanismes de validation

Jeux d'essai : `defaultTest` + trois scénarios dans `tests` (nominal, minimal, ambigu), reproductibles avec `promptfoo eval`.

Métriques : présence des sections (assertions), cohérence des noms (manuelle ou script), qualité sémantique (extension possible avec modèle juge).

Limites : pas d'exécution réelle des outils ; dépendance au modèle ; placeholders à traiter en aval ; prompt en français.

4. Dépendances aux outils et précautions

Outils simulés : générateurs de DAG Airflow, code ETL, tests, documentation — en `promptfoo` le LLM produit directement le texte.

Techniques : OpenRouter (GPT-3.5-turbo), clé API dans `.env`.

Précautions : éviter hallucinations (consignes + placeholders) ; cohérence (protocole et rappel dans le prompt) ; format de sortie strict ; pas d'exécution de code dans l'éval ; en production, sandbox et validation des artefacts avant déploiement.

5. Synthèse

L'architecture de l'agent planificateur est décrite dans **`d3/promptfooconfig.yaml`** (rôle, instructions, outils, protocoles, gestion des erreurs). Le rapport couvre la narration fonctionnelle, les itérations `promptfoo`, les mécanismes de validation et les dépendances/précautions pour des évaluations reproductibles et une évolution contrôlée.

PDF : Ctrl+P → Enregistrer au format PDF.