Agent autonome borné

Agent qui tourne sans supervision en heures creuses pour auditer, documenter et corriger dans un périmètre strictement borné. Deux barrières de sécurité indépendantes, et une boucle de vérification par un juge qui contrôle factuellement ce que l'agent affirme avoir fait.

Fiche de méthode associée : `../../docs/16-agents-autonomes-garde-fous.md`

Architecture

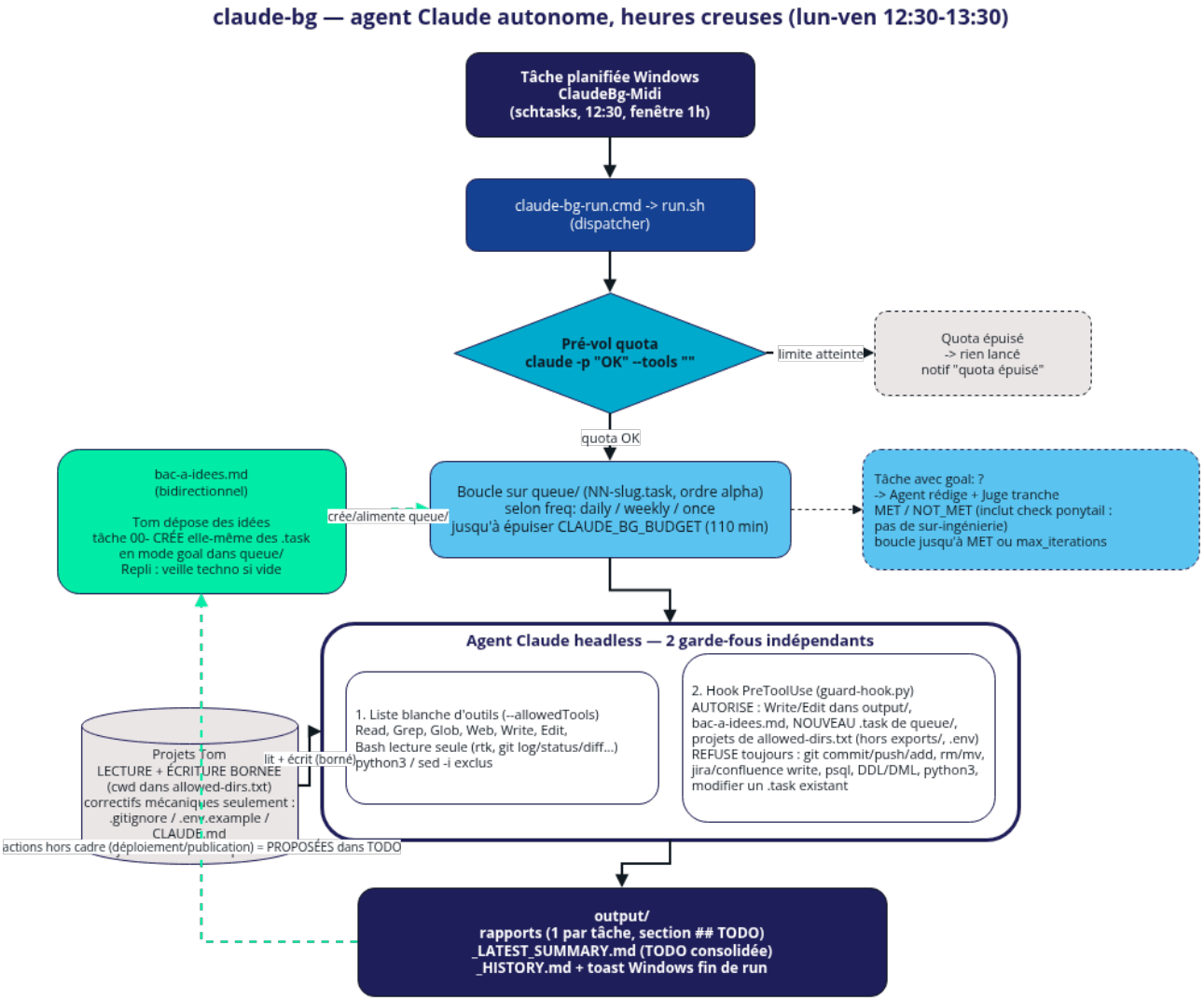


Figure 1: architecture

```
déclencheur planifié (fenêtre d'heures creuses)
      │
      ▼
lanceur ── vérification préalable de quota (appel minimal)
      │
      ▼
répartiteur (run.py) : boucle sur la file de tâches
      │
      ├─ verrou anti-double exécution, avec auto-guérison
      ├─ pour chaque tâche due :
      │     ├─ invocation de l'agent, répertoire de travail borné
      │     ├─ (mode objectif) boucle agent puis juge
      │     └─ déplacement du fichier de tâche selon le résultat
```

```
      │
      ▾
rapport agrégé ─── notification
      │
      └─ détection de plantage : si aucun rapport n'a été écrit, en écrire un
```

Ce qui vaut d'être regardé

1. Les deux barrières indépendantes

Barrière	Où	Nature
Liste blanche d'outils	invocation dans `run.py`	déclarative : simple, lisible, grossière
Crochet de contrôle	`guard-hook.py`	impérative : inspecte le contenu réel de chaque appel

Elles se recouvrent partiellement et couvrent chacune ce que l'autre laisse passer. La liste blanche autorise l'exécution de commandes shell (nécessaire pour lire des fichiers) ; le crochet refuse toute commande de mutation, de suppression, de publication ou d'accès réseau.

2. Le crochet de sécurité (`guard-hook.py`)

Fermeture par défaut : entrée non analysable, fichier de périmètre absent, exception interne, cas non prévu, tout est refusé. Testé explicitement avec des entrées malformées.

Deux subtilités qui comptent :

- la modification d'un fichier de tâche existant est interdite (seule la création est permise), ce qui empêche l'agent d'écraser une tâche déjà traitée ou de réécrire ses propres consignes ;
- l'exécution d'un script identifié du périmètre est autorisée, mais pas l'interpréteur en mode expression libre, ce qui permet de vérifier un correctif sans ouvrir une porte d'exécution arbitraire.

`allowed-dirs.txt` porte le périmètre d'écriture, hors répertoires de données et de secrets.

3. La boucle agent puis juge (`run.py`, `parse_verdict.py`)

Le mécanisme qui rend le résultat fiable. Si une tâche déclare un critère d'acceptation, le répartiteur alterne deux rôles au lieu d'un simple passage.

L'instruction la plus importante du juge : vérifier factuellement avant tout verdict positif. Un rapport peut affirmer un changement de code sans que ce changement existe. Le juge doit rechercher la chaîne annoncée, lire le fichier cité, consulter l'état réel du dépôt.

Cas réel : à la première itération, l'agent affirmait avoir implémenté trois fonctionnalités et annonçait un diff de plusieurs dizaines de lignes. Le juge a cherché les chaînes annoncées : rien. L'état du dépôt : propre. Verdict négatif, avec citation de la recherche qui contredit le rapport. C'est le garde-fou anti-affabulation central du système.

Trois raffinements :

- un modèle différent pour le juge, pour éviter un biais partagé ;
- la sur-ingénierie comme motif de rejet, au même titre que le travail incomplet ;
- un mécanisme de contestation à partir de la deuxième itération, qui protège contre deux dérives symétriques : l'agent qui capitule systématiquement, et celui qui conteste par réflexe.

Détail du parsing du verdict dans `parse_verdict.py` : la recherche est ancrée en début de ligne, car une réponse qui traite d'abord une contestation peut contenir le mot ailleurs.

4. La file de tâches sur système de fichiers

Pas de gestionnaire de messages, pas de base : un répertoire de fichiers texte et des déplacements entre dossiers d'état. Format documenté dans `docs/technique-format-task.md`, exemples dans `tasks-seed/`.

Ce que ce choix apporte : chaque tâche est un fichier lisible et modifiable à la main, l'état est visible d'un coup d'œil, aucune infrastructure à maintenir.

Deux détails qui encodent de la logique sans mécanisme supplémentaire : la numérotation des fichiers (un préfixe réservé à l'orchestration qui tourne en premier, une plage pour les tâches auto-générées), et la boucle qui repioche la file à chaque tour plutôt que de prendre un instantané, ce qui permet à une tâche d'en créer d'autres consommées dans la même session.

5. Le verrou avec auto-guérison (`run.py`, `acquire_lock`)

La création d'un répertoire est atomique sur un système de fichiers : c'est un verrou plus fiable qu'un fichier de numéro de processus.

Détection d'un verrou périmé en combinant deux critères : le processus propriétaire est-il vivant, et le verrou est-il plus vieux que le budget maximal. Sans cette auto-guérison, un arrêt brutal bloque toutes les exécutions suivantes indéfiniment, et personne ne s'en aperçoit puisque l'agent tourne sans supervision.

6. La détection de plantage

Un bloc de finalisation vérifie qu'un rapport a bien été écrit. Sinon il en écrit un explicitement marqué comme plantage, avec le code de sortie et le chemin du journal.

Un processus automatisé sans supervision ne doit jamais échouer en silence. L'absence de résultat est en soi un résultat à publier.

7. Les tests (`tests/`)

103 tests, bibliothèque standard uniquement, exécutés en intégration continue (`.gitlab-ci.yml`).

Le test du crochet crée un environnement temporaire complet (répertoire de base, fichier de périmètre, tâche existante) et vérifie le code de sortie pour chaque type d'appel simulé. Reproductible en local comme en CI, sans dépendre de l'environnement réel.

Vérification après anonymisation

Contrôle	Résultat
Compilation des 4 fichiers Python	ok
Tests du crochet de sécurité	50 passent
Tests du parsing de verdict	7 passent
Tests du répartiteur	46 passent
Validité du XML du planificateur (UTF-16)	ok
Motifs sensibles résiduels	aucun

Ce qui a été retiré

Retiré	Raison
`logs/` (130 fichiers)	transcriptions complètes d'exécutions : chemins, noms, contenus de travail
`output/`	rapports de production sur des projets internes
`done/`, `archive-taches/`	tâches réelles référençant des projets internes
`graphify-out/`	sortie générée, régénérable
historique de contrôle de version	contient ce qui a été supprimé

`tasks-seed/` est conservé : ce sont les gabarits de tâches, qui documentent le format sans référencer de travail réel.

Dans le XML du planificateur (encodage UTF-16), les marqueurs d'anonymisation sont écrits sans chevrons pour rester un XML valide.