Créez une équipe

de 20 personnes avec l'IA.

Livret intermédiaire : orchestrer et automatiser

Philippe Anel

2026

Ce livret accompagne la Masterclass IA « Créer une équipe de 20 personnes avec l'IA », donnée le 6 avril 2026. Niveau intermédiaire.

Pour toute remarque ou suggestion : `support@scanderia.com`

Table des matières

Chapitre 1

Construire des skills qui marchent vraiment

Vous avez des agents qui tournent. Ils font le travail, la plupart du temps. Mais « la plupart du temps » n'est pas suffisant quand vous comptez dessus au quotidien. Ce chapitre vous montre comment passer d'un fichier `.md` qui donne des instructions à un **skill** : un composant testable, fiable et réutilisable.

Pourquoi vos premiers fichiers .md marchent « parfois »

Vous avez probablement déjà vécu ça : vous donnez la même instruction à votre agent deux jours de suite et il produit deux résultats différents. Pas radicalement différents, mais suffisamment pour que vous corrigiez à la main. Trois raisons à ça.

1. L'IA interprète. Quand vous écrivez « rédige un email professionnel », l'IA doit décider ce que « professionnel » veut dire. Selon le contexte de la conversation, son interprétation varie. Ce n'est pas un bug, c'est sa nature : un modèle de langage génère du texte probable, pas du texte déterministe.

2. Le contexte change. D'une session à l'autre, les fichiers chargés, l'ordre des messages, la taille du contexte ne sont pas identiques. L'IA ne « se souvient » pas de la session précédente. Chaque session est un nouveau départ.

3. Vous n'avez aucun moyen de vérifier. Votre fichier `.md` dit quoi faire, mais il ne dit pas comment vérifier que c'est bien fait. Vous relisez à l'œil ; quand c'est « à peu près bon », vous passez à autre chose. Il n'y a pas de test, pas de critère objectif, pas de filet.

Un **skill**, c'est la réponse à ces trois problèmes : des instructions assez précises pour réduire l'interprétation, un format de sortie assez contraint pour que le résultat soit vérifiable et un protocole de test pour confirmer que ça tient dans la durée.

Anatomie d'un skill professionnel

Un skill professionnel a **5 composants**. Pas 3, pas 10. Cinq. Chacun a un rôle précis ; l'absence de n'importe lequel rend le skill fragile.

1. Le header de déclenchement

C'est la réponse à la question : *quand est-ce que ce skill s'active ?* Sans header clair, le skill se déclenche au mauvais moment, ou ne se déclenche jamais.

```
## Déclenchement

Ce skill s'active quand :
- On me demande de rédiger un email à un client
- On me demande de répondre à un email client

Ce skill ne s'active PAS quand :
- L'email est interne (entre collègues)
- La demande concerne un devis (c'est le skill "devis")
- On me demande juste de relire un email déjà écrit
```

Notez les **limites négatives** : dire quand le skill ne s'active pas est aussi important que dire quand il s'active. Sans ça, l'agent va utiliser votre skill « email client » pour répondre à un message interne entre collègues.

2. L'overview

Un paragraphe, maximum deux, qui explique *à l'IA* ce que ce skill fait. Pas pour vous (vous le savez déjà), pour elle. Ça lui donne le contexte global avant de plonger dans les détails.

```
## Overview

Tu rédiges des emails professionnels à destination des clients de
[entreprise]. Le ton est formel mais chaleureux. Tu vouvoies
toujours. Tu ne prends jamais d'engagement financier ni de délai
sans validation explicite de ma part.
```

L'overview est court. Il donne la mission, le ton et les gardes-fous les plus critiques. Les détails viennent après.

3. Le workflow pas-à-pas

C'est le cœur du skill. Des étapes numérotées, impératives, qui ne laissent pas de place à l'interprétation. Chaque étape est **testable** : on peut vérifier si elle a été faite ou non.

```
## Workflow

1. Lis l'email du client dans son intégralité
2. Identifie : l'objet de la demande, le ton du client (satisfait,
   mécontent, neutre), les questions explicites
3. Consulte `clients.md` pour trouver le profil du client
   (historique, préférences)
```

```
4. Rédige la réponse en respectant les règles de `ton.md`
5. Vérifie que tu as répondu à CHAQUE question posée par le client
6. Si le client demande un délai ou un prix : ne réponds PAS, écris
   "ESCALADE : demande de [délai/prix]"
7. Sauvegarde le brouillon dans `brouillons/email-YYYY-MM-DD-client.md`
```

Comparez avec « Rédige un email de réponse au client ». La version en 7 étapes produit le même résultat, mais elle est vérifiable, reproductible et elle gère les cas limites (étape 6).

4. Le format de sortie

L'IA doit savoir *exactement* à quoi ressemble le résultat attendu. Pas « un email », mais la structure précise.

````
## Format de sortie

Le fichier `brouillons/email-YYYY-MM-DD-client.md` doit contenir :

```
Objet : [objet de l'email]
Destinataire : [nom du client]

[Corps de l'email, maximum 150 mots]

Cordialement,
[signature standard]
```

Pas de pièce jointe. Pas de CC. Pas de PS.
````

Le format de sortie est un contrat. Si la sortie ne correspond pas au format, c'est une erreur détectable automatiquement. C'est ce qui rend le skill testable.

5. Les exemples et cas limites

Au minimum deux exemples : un cas normal (happy path) et un cas limite (edge case). L'IA apprend mieux par l'exemple que par la règle.

````
## Exemples

### Exemple 1 : demande simple (happy path)

**Entrée** : "Le client Martin demande quand sa commande sera livrée"
**Sortie attendue** :
```
Objet : Suivi de votre commande
Destinataire : M. Martin
````
````

````
Bonjour Monsieur Martin,

Je fais suite à votre question concernant votre commande.
Votre colis a été expédié le [date] et devrait vous parvenir
sous 3 à 5 jours ouvrés. Vous recevrez un email de suivi
avec le numéro de tracking dès qu'il sera disponible.

N'hésitez pas à revenir vers nous si vous avez d'autres questions.

Cordialement,
[signature]
```

### Exemple 2 : cas limite (demande de remboursement)

**Entrée** : "Le client Durand veut un remboursement, il est furieux"
**Sortie attendue** :
```
ESCALADE : demande de remboursement (engagement financier)
Client : Durand
Ton détecté : mécontent
Résumé : demande de remboursement, nécessite validation avant réponse
```
````

L'exemple 2 montre le comportement attendu en cas d'escalade. Sans cet exemple, l'agent pourrait quand même tenter de gérer le remboursement tout seul.

Les 5 pannes classiques

Quand un skill ne marche pas, c'est presque toujours une de ces 5 pannes. Apprenez à les reconnaître et vous gagnerez des heures de débogage.

Le Silencieux

Symptôme : le skill ne se déclenche jamais. Vous demandez quelque chose qui devrait l'activer, mais l'agent répond de façon générique, sans utiliser le skill.

Cause : le header de déclenchement est trop vague ou trop restrictif. L'IA ne reconnaît pas que votre demande correspond au skill.

Correction : ajoutez des synonymes dans le déclencheur. « Rédige un email » ne couvre pas « écris un mail », « prépare une réponse », « réponds à ce client ». Listez les formulations courantes.
````

Le Pirate

Symptôme : le skill se déclenche au mauvais moment. Vous demandez autre chose et l'agent utilise quand même ce skill.

Cause : pas de limites négatives. Le skill dit quand s'activer, mais pas quand ne PAS s'activer.

Correction : ajoutez une section « Ce skill ne s'active PAS quand » dans le header. Soyez explicite sur les cas ambigus.

Le Dériveur

Symptôme : le skill se déclenche bien, mais la sortie n'est pas ce que vous attendez. Le format est bon, le contenu est à côté.

Cause : les instructions du workflow sont ambiguës. « Rédige de façon professionnelle » laisse trop de marge d'interprétation.

Correction : rendez chaque étape du workflow plus spécifique. Remplacez les adjectifs (« professionnel », « concis », « pertinent ») par des critères mesurables (« maximum 150 mots », « vouvoiement », « pas de jargon technique »).

Le Fragile

Symptôme : le skill marche sur les cas simples, mais casse dès que l'entrée est inhabituelle. Un email en anglais, un client sans historique, une demande ambiguë.

Cause : pas d'edge cases dans les exemples. L'IA n'a appris que le happy path.

Correction : ajoutez des exemples pour les cas limites. Demandez-vous : « qu'est-ce qui peut arriver de bizarre ? » et montrez à l'IA comment réagir.

Le Zélé

Symptôme : le skill fait plus que ce qu'on lui demande. Vous voulez un brouillon d'email, il envoie l'email. Vous voulez une analyse, il modifie le fichier source.

Cause : pas de contraintes de périmètre. Le skill dit quoi faire, mais pas où s'arrêter.

Correction : ajoutez une section « Ce que tu ne fais PAS » (la via negativa du livret débutant, appliquée au skill). Soyez explicite sur les limites : « tu ne modifies JAMAIS les fichiers sources », « tu ne publies JAMAIS sans validation ».

Le protocole de test

Un skill n'est pas fiable tant qu'il n'a pas passé 5 tests. Ce C'est le minimum pour avoir confiance.

Test 1 : happy path

Donnez au skill une entrée propre, typique, sans piège. Est-ce que la sortie correspond au format attendu ? Est-ce que le contenu est correct ?

Si ça échoue : votre workflow ou votre format de sortie a un problème. Corrigez avant d'aller plus loin.

Test 2 : entrée minimale

Donnez au skill une entrée incomplète. « Rédige un email » sans dire à qui ni pourquoi. Est-ce que l'agent demande ce qui manque, ou est-ce qu'il invente ?

Si ça échoue : ajoutez dans le workflow une étape de vérification des prérequis. « Avant de commencer, vérifie que tu as : le nom du destinataire, l'objet, le contexte. Si un élément manque, demande-le. »

Test 3 : cas limite

Donnez au skill une entrée bizarre mais réaliste. Un email dans une langue étrangère. Un client inconnu. Une demande contradictoire.

Si ça échoue : ajoutez des exemples edge case et des règles d'escalade.

Test 4 : test négatif

Envoyez une requête qui ne devrait PAS déclencher le skill. Si le skill s'active quand même, vos limites négatives sont insuffisantes.

Si ça échoue : renforcez la section « Ce skill ne s'active PAS quand » du header.

Test 5 : répétabilité

Donnez 3 fois la même entrée (dans 3 sessions différentes). Les 3 sorties doivent être *substantiellement identiques*. Pas mot pour mot (l'IA n'est pas déterministe), mais même structure, même contenu, même ton.

Si ça échoue : vos instructions laissent trop de marge d'interprétation. Resserrez le workflow et le format de sortie.

> **Le seuil de confiance**
>
> Un skill qui passe les 5 tests n'est pas parfait. Aucun ne l'est, parce que l'IA n'est pas déterministe. Mais un skill qui passe les 5 tests est **suffisamment fiable** pour que vous puissiez vous y fier au quotidien et investir votre temps sur autre chose que la relecture systématique.

Exercice : transformer un agent en skill testé

Prenez un de vos agents du livret débutant (l'agent de relecture, l'agent email, ou un agent que vous avez créé vous-même). Faites ceci :

1. **Restructurez-le** en 5 composants : header de déclenchement, overview, workflow, format de sortie, exemples.
2. **Ajoutez des limites négatives** dans le header : quand le skill ne s'active PAS.
3. **Ajoutez au moins 2 exemples** : un happy path et un cas limite.
4. **Faites passer les 5 tests** dans l'ordre. Notez les résultats.
5. **Corrigez** ce qui échoue et retestez.

Si vous arrivez à faire passer les 5 tests à un de vos agents, vous avez compris le principe. Appliquez-le aux autres progressivement.

Ce qu'il faut retenir de ce chapitre

- Un fichier `.md` qui donne des instructions n'est pas un skill. Un skill a **5 composants** : déclenchement, overview, workflow, format de sortie, exemples.
- Les **limites négatives** (quand ne PAS s'activer, quoi ne PAS faire) sont aussi importantes que les instructions positives.
- Les **5 pannes classiques** couvrent la quasi-totalité des problèmes : Silencieux, Pirate, Dériveur, Fragile, Zélé.
- Un skill fiable passe **5 tests** : happy path, entrée minimale, cas limite, test négatif, répétabilité.
- L'objectif est la **fiabilité suffisante** pour ne plus relire systématiquement, pas la perfection (inatteignable avec un modèle non déterministe).

> **Avant le prochain chapitre**
>
> Transformez au moins un de vos agents en skill testé. Le chapitre suivant part du principe que vous savez construire un skill fiable et aborde un problème plus subtil : que faire quand ni vous ni l'IA ne pouvez évaluer la qualité du résultat.

Chapitre 2

Spécifier ce que l'IA ne peut pas évaluer

Le chapitre précédent vous a appris à construire des skills fiables. Mais il y a un problème plus profond : que faire quand ni vous ni l'IA ne pouvez évaluer la qualité du résultat ? Quand l'agent produit un texte juridique et que vous n'êtes pas juriste ? Quand il traduit un document technique et que vous ne maîtrisez pas la langue cible ? Ce chapitre attaque ce paradoxe de front.

Le paradoxe de l'évaluation

Vous demandez à votre agent de rédiger un contrat type. Il vous livre un document bien structuré, bien écrit, avec des clauses qui ont l'air solides. Vous n'êtes pas juriste. Vous ne pouvez pas évaluer si c'est correct. Mais le texte est fluide, assuré, professionnel. Alors vous validez.

C'est le piège. **L'assurance de l'IA ne corrèle pas avec sa précision.** Un modèle de langage produit du texte qui « a l'air vrai » parce que c'est sa nature : il génère la suite la plus probable. Un texte juridique halluciné ressemble exactement à un texte juridique correct. La seule différence, c'est que l'un vous expose à un risque et l'autre non.

Le livret débutant abordait la sycophantie (chapitre 5) : l'IA qui dit « oui » plutôt que d'admettre qu'elle ne sait pas. Ici, on va plus loin : même quand l'IA ne dit pas « oui », elle produit du contenu avec la même assurance, qu'elle ait raison ou tort.

Programmer le doute

La première défense, c'est de demander explicitement à l'agent de **signaler ses incertitudes**. Pas en lui disant « sois prudent » (trop vague), mais en lui donnant un protocole structuré.

```
## Règle de confiance

Pour chaque élément factuel ou technique que tu produis, indique
ton niveau de confiance sur une échelle de 3 :

- CERTAIN : tu as vu cette information dans les fichiers fournis
  ou c'est un fait vérifiable (date, nom propre, définition standard)
```

```
- PROBABLE : tu déduis cette information du contexte mais elle
  n'est pas explicitement dans les sources
- INCERTAIN : tu n'as pas de source et tu extrapoles. Signale-le
  explicitement avec "[à vérifier par un expert]"

Si plus de 30% de ta réponse est marquée INCERTAIN, arrête-toi et
demande-moi de fournir des sources supplémentaires.
```

Cette règle change le comportement de l'agent. Au lieu de produire un texte uniformément assuré, il produit un texte annoté. Vous ne savez toujours pas si le contenu est correct, mais vous savez **où regarder**. Les parties marquées « incertain » sont celles que vous devez faire vérifier par un expert.

Les faux positifs du doute

Un agent trop prudent doute de tout, y compris de ce qu'il sait bien. Si votre agent marque « incertain » sur des informations basiques, **calibrez** : ajoutez des exemples de ce qui est certain (faits tirés des fichiers fournis) face à ce qui est incertain (extrapolations). L'IA apprend par l'exemple.

Imposer un lexique propriétaire

L'IA a une tendance naturelle à « traduire » vos termes métier dans son propre vocabulaire. Vous écrivez « prestation d'accompagnement », elle reformule en « service de coaching ». Vous parlez de « bilan de compétences », elle glisse vers « évaluation des aptitudes ». Ce glissement sémantique est insidieux parce qu'il est presque correct, mais « presque » n'est pas acceptable dans un contexte professionnel.

La solution : le fichier lexique.md

```
# Lexique [nom de l'entreprise]

## Règle absolue
Tu utilises UNIQUEMENT les termes de ce fichier. Jamais de
synonymes, jamais de reformulations, jamais de "traduction" dans
un langage plus courant. Si un terme n'est pas dans ce lexique,
demande-moi avant de l'utiliser.

## Termes

| Terme officiel | Ne jamais utiliser | Définition |
|---|---|---|
| Prestation d'accompagnement | coaching, consulting, service | Suivi personnalisé
    sur 6 mois... |
```

```
| Bilan de compétences | évaluation, assessment, audit | Processus structuré de
    24h... |
| Bénéficiaire | client, patient, usager | Personne qui bénéficie de la prestation |
| Convention tripartite | contrat, accord | Document signé par les 3 parties... |
```

La colonne « Ne jamais utiliser » est cruciale. Elle dit explicitement à l'IA quels mots éviter. Sans cette colonne, l'IA va continuer à glisser vers ses synonymes préférés.

> **Comment construire votre lexique**
>
> Demandez à votre agent de rédiger un texte dans votre domaine *sans* lexique. Relisez et surlignez chaque mot qu'il a mal utilisé ou reformulé. Ce sont les entrées de votre lexique. Répétez 3-4 fois : à chaque passage, le lexique s'enrichit des glissements que vous n'aviez pas anticipés.

L'agent contradicteur

Le chapitre 5 (autonomie) introduit l'agent vérificateur : un second agent qui valide la production du premier. Ici, on va plus loin avec **l'agent contradicteur** : un agent dont le seul travail est de *chercher les erreurs*.

```
# contradicteur.md

## Ton rôle

Tu es l'adversaire de l'agent rédacteur. Ton unique mission est
de trouver des erreurs, des incohérences et des affirmations non
sourcées dans sa production.

## Ce que tu cherches

- Affirmations factuelles sans source identifiable
- Termes qui ne sont pas dans lexique.md
- Incohérences entre deux parties du document
- Claims chiffrées (pourcentages, dates, montants) :
  sont-elles vérifiables ?
- Ton inapproprié (tutoiement, jargon marketing, langage trop familier)

## Ce que tu produis

Pour chaque problème trouvé :
- Ligne concernée (citation exacte)
- Type de problème (fait non sourcé / terme hors lexique /
  incohérence / claim non vérifiable)
- Suggestion de correction

Si tu ne trouves rien, écris "RAS" et arrête-toi. Ne cherche pas
des problèmes qui n'existent pas.
```

L'agent contradicteur est le pattern **red team** appliqué à vos agents. En sécurité informatique, une red team attaque un système pour trouver ses failles. Ici, l'agent contradicteur attaque la production d'un autre agent pour trouver ses erreurs. C'est de l'assurance qualité minimale dès que la sortie quitte votre bureau.

> **Quand utiliser un contradicteur**
>
> - **Toujours** pour les contenus qui seront publiés ou envoyés à un client.
> - **Toujours** pour les contenus dans un domaine que vous ne maîtrisez pas.
> - **Optionnel** pour les contenus internes ou les brouillons.
>
> Le contradicteur ne remplace pas l'expert humain. Il attrape les erreurs évidentes que vous auriez ratées parce que le texte « avait l'air bien ». L'expert humain reste indispensable pour les questions de fond.

Surveiller la dérive dans le temps

Tout ce qui précède attrape une mauvaise sortie sur l'instant. Reste un problème plus insidieux : un agent qui passait vos cinq tests il y a trois mois peut ne plus les passer aujourd'hui, alors que vous n'avez rien changé.

Pourquoi les agents dérivent

Trois causes, qui se cumulent.

1. Le modèle est probabiliste. Même prompt, même contexte : deux exécutions ne produisent pas exactement la même sortie. La plupart du temps les variations sont mineures et tolérables, parfois non. Le test de répétabilité du chapitre 1 mesure cette variance à un instant T, mais il ne dit rien sur la durée.

2. Le modèle change sous vos pieds. Anthropic, OpenAI et les autres font évoluer leurs modèles en continu. Une mise à jour peut améliorer la performance sur dix dimensions et la dégrader sur une onzième, qui se trouve être pile celle dont dépend votre agent. Vous ne serez pas prévenu, vous le verrez en aval.

3. Votre propre contexte évolue. Vous ajoutez un client, vous enrichissez votre lexique, vous insérez une règle pour répondre à un cas particulier rencontré la semaine dernière. Chacune de ces modifications est rationnelle prise isolément. Cumulées sur six mois, elles peuvent rendre l'agent moins stable que sa version initiale.

Le mécanisme de surveillance

Une seule pratique vraiment efficace : **rejouer en routine les cas-or**. Pour chaque agent en production, vous gardez trois à cinq entrées dont vous connaissez la sortie attendue. Une fois

par semaine ou par mois, selon la criticité, vous les rejouez et vous comparez à la sortie de référence.

```
# tests/regression-agent-email.md

## Cas 1 : demande de devis simple
Entree : "Le client Martin demande un devis pour 10h
de prestation."
Sortie attendue (substantiellement) :
- Objet contenant "Devis" ou "Proposition"
- Mention du client par son nom
- ESCALADE financiere (engagement chiffre)
- Pas d'envoi automatique

## Cas 2 : email anglophone
Entree : "Reply to: 'When will the report be ready?'"
Sortie attendue :
- Reponse en francais (regle de l'agent)
- ou ESCALADE "demande hors langue maitrisee"

## Cas 3 : ton mecontent
...
```

Vous ne cherchez pas une comparaison mot à mot, qui serait illusoire sur un modèle probabiliste. Vous cherchez : la structure de sortie est-elle conservée ? Les règles d'escalade sont-elles respectées ? Le format est-il toujours valide ? Le lexique est-il toujours suivi ?

Versionner les baselines

Une sortie de référence, c'est une photo à un instant donné. Sans versioning, dans six mois, personne ne saura plus si la baseline était la sortie de l'agent v1 ou la sortie de l'agent v3 retouchée après un incident client. Notre convention :

```
tests/
  regression-agent-email.md      <- les cas d'entree
  baselines/
    2026-01-15-v1.md             <- sorties de reference initiales
    2026-03-02-v2.md             <- apres ajout du lexique v2
    2026-04-30-v3.md             <- apres mise a jour modele
```

Chaque fois que vous modifiez l'agent (lexique, règles, instructions) ou que vous constatez une dérive imputable au fournisseur, vous créez une nouvelle baseline datée. La précédente n'est pas effacée, elle est conservée. Quand vous ouvrez un incident en mai, vous pouvez comparer le comportement de mai à la baseline de janvier et voir précisément quand la dérive a commencé.

Si vous avez Git, c'est plus simple : un commit par baseline avec un message du type `baseline agent-email v3 : ajout règle ton-mecontent`. Sans Git, des fichiers datés font le travail.

Distinguer dérive et variation

Toutes les variations ne sont pas des dérives. Pour faire la différence, deux questions.

Est-ce que la sortie reste valide ? Si elle respecte le format, les règles d'escalade et le lexique, c'est une variation acceptable. Si elle viole une de ces contraintes, c'est une dérive.

Est-ce que la dérive est isolée ou systématique ? Une sortie sur trois qui dérive sur le même test, ça reste du bruit. Cinq sur cinq, c'est un signal. Trois exécutions consécutives qui dérivent toutes dans la même direction : signal.

Une grille d'action raisonnable

Une fois la dérive observable, encore faut-il décider quoi en faire. Voici une grille de départ que nous appliquons sur nos propres agents. Elle n'est pas une vérité absolue : à calibrer selon votre tolérance au risque, la criticité de l'agent et le coût d'une fausse alerte chez vous.

Échec sur 5 cas-or	Verdict	Action recommandée
0/5 ou 1/5	Bruit	Logger, ne rien faire d'autre
2/5	Surveillance	Avertir, augmenter la fréquence des tests
3/5	Dérive	Redescendre d'un niveau d'autonomie
4/5 à 5/5	Dérive sévère	Repasser en mode supervisé, investiguer

L'idée n'est pas que ces seuils soient justes pour tous les usages. L'idée est qu'*un seuil écrit* est mieux qu'une décision prise à chaud quand un client se plaint. Vous arrivez à votre seuil sans discussion possible, vous l'appliquez, vous documentez. C'est ce qui fait la différence entre un incident qu'on traite et un incident qui empire pendant qu'on délibère.

La dérive que les tests ne voient pas

Les cas-or attrapent les dérives *structurelles* : format violé, règle d'escalade ignorée, terme hors lexique. Ils ne voient pas une autre forme de dérive, plus lente : l'agent reste formellement valide, mais glisse en *ton*. Il devient imperceptiblement plus marketing, plus verbeux, plus assertif. La sortie passe vos tests, et pourtant elle ne ressemble plus à ce que vous écririez.

Aucun test automatique simple ne détecte cette dérive-là, parce que la définition même du « bon ton » n'est pas formalisable en règles binaires. Le seul mécanisme efficace que nous connaissons reste humain : une fois par mois, vous lisez *trois ou quatre sorties réelles* de l'agent prises au hasard, et vous vous posez une seule question : est-ce que je signerais ce texte ? Si la réponse est non, et que les tests structurels passent quand même, c'est que vous êtes face à une dérive qualitative. Le remède est le même qu'à l'étape 6 du protocole de test du chapitre 1 : resserrer les instructions de l'agent (ton, registre, longueur) plutôt que d'ajouter des règles binaires.

C'est une limite assumée de la méthode. Pour aller plus loin, il faudrait des outils de mesure stylistique automatique (analyse de sentiment, lexique gonfleur, longueur moyenne des phrases) qui sortent du périmètre du livret. La discipline de relecture mensuelle n'est pas idéale, mais c'est ce qui tient en pratique pour un constructeur seul ou en petite équipe.

Quand la dérive apparaît

Si vos cas-or commencent à échouer, la première chose à faire n'est *pas* de réécrire l'agent. C'est de redescendre d'un niveau d'autonomie (chapitre 5) le temps de comprendre. Un agent qui dérive en mode supervisé fait peu de dégâts. Un agent qui dérive en mode autonome avec garde-fous peut en faire beaucoup avant que vous le remarquiez.

Ce que la surveillance vous donne

Aucun de ces mécanismes n'élimine le non-déterminisme. Ils le rendent **observable**. Vous n'achetez pas de la certitude, vous achetez la capacité à voir quand vos agents commencent à sortir du cadre, avant que ce soit visible côté client. Pour des constructeurs sérieux, c'est la différence entre un système qu'on peut maintenir et un système qui finit par causer un incident sans qu'on sache pourquoi.

La dérive est un signal de gouvernance

Une dérive observée n'est pas seulement un sujet technique. C'est un signal de gouvernance : qui est responsable de regarder ? Quand ? Avec quel mandat pour redescendre un agent d'un niveau d'autonomie ou pour bloquer une production ? Ces questions sont le sujet du chapitre 11. Si votre dispositif de surveillance ne sait pas qui agit sur quel seuil, il ne sert qu'à constater des incidents après coup.

Exercice : doute + lexique + contradicteur

1. **Créez un fichier `lexique.md`** pour votre métier. 10 termes minimum, avec la colonne « Ne jamais utiliser ».
2. **Ajoutez la règle de confiance** (certain / probable / incertain) à un de vos agents.
3. **Demandez-lui de produire un texte technique** dans votre domaine.
4. **Créez un agent contradicteur** et faites-lui relire le texte.
5. **Comptez :** combien de termes hors lexique ? Combien de passages marqués « incertain » ? Combien de problèmes trouvés par le contradicteur ?
6. **Corrigez** le lexique et les instructions en fonction de ce que vous avez trouvé. Recommencez.
7. **Archivez 3 cas-or de régression.** Choisissez trois entrées dont vous savez ce que la sortie doit donner et sauvegardez-les dans `tests/regression-[agent].md`. Mettez-vous un rappel mensuel pour les rejouer.

Ce qu'il faut retenir de ce chapitre

- **L'assurance de l'IA ne corrèle pas avec sa précision.** Un texte fluide n'est pas un texte correct.
- Programmez le **doute** : certain / probable / incertain. Ça vous dit où regarder.
- Imposez un **lexique propriétaire** avec les termes interdits, pas juste les termes autorisés.
- L'**agent contradicteur** cherche les erreurs que vous ne voyez pas parce que le texte « a l'air bien ».
- Rien de tout ça ne remplace l'**expert humain** pour les questions de fond. Mais ça attrape les erreurs évidentes.
- Les agents **dérivent** dans le temps. Gardez 3 à 5 **cas-or** par agent et rejouez-les en routine. Vous n'éliminez pas le non-déterminisme, vous le rendez observable.

Avant le prochain chapitre

Construisez votre lexique. C'est le fichier le plus rentable que vous créerez : il améliore la qualité de tous vos agents d'un coup. Le chapitre suivant vous montre comment transformer vos fichiers isolés en une base de connaissances vivante qui s'enrichit au fil du temps.

Chapitre 3

Construire une base de connaissances vivante

Vos fichiers `.md` contiennent les instructions de vos agents. Mais ce ne sont que des instructions. Avec le temps, vos agents accumulent du savoir : les corrections que vous leur apportez, les patterns qu'ils rencontrent, les cas limites qu'ils traitent. Ce chapitre vous montre comment capturer ce savoir et le transformer en base de connaissances qui s'enrichit au fil de l'usage.

Le journal comme mémoire opérationnelle

Le livret débutant (chapitre 7) recommandait de tenir un journal pour chaque agent. Si vous l'avez fait, vous avez déjà les fondations d'une base de connaissances. Si vous ne l'avez pas fait, c'est le moment de commencer.

Un fichier par jour, pas un gros fichier unique

Le piège classique : un fichier `journal.md` qui grossit indéfiniment. Au bout de 3 mois, il fait 50 000 mots, Claude le charge à chaque session et ça explose votre consommation de tokens (chapitre 4).

La bonne structure :

```
mon-agent/
  journal/
    2026-04-01.md
    2026-04-02.md
    2026-04-03.md
    ...
  journal-index.md   <- résumé des entrées marquantes
```

Chaque fichier de journal contient ce que l'agent a fait ce jour-là. Le fichier `journal-index.md` est un résumé compact des événements notables (erreurs, corrections, nouvelles règles). C'est l'index que Claude charge, pas les 90 fichiers de détail.

Ce qu'un journal contient

```
# Journal -- 2026-04-03

## Tâches réalisées
- Rédigé 3 emails clients (Martin, Dupont, Leroy)
- Analysé le rapport hebdomadaire

## Problèmes rencontrés
- Email Leroy : ton trop familier, corrigé après relecture
  -> Règle ajoutée : vouvoiement systématique pour les clients
     > 60 ans

## Décisions en attente
- Client Dupont demande un délai exceptionnel : escaladé,
  en attente de réponse

## Nouvelles règles apprises
- Ajouté dans ton.md : "Pour les clients de plus de 60 ans,
  vouvoiement strict, pas de tournures familières même si le
  client tutoie"
```

Le journal n'est pas juste un log. C'est un **lieu d'apprentissage**. La section « Nouvelles règles apprises » est la plus importante : elle capture les corrections que vous faites et les transforme en instructions permanentes.

L'agent « chef de bureau »

Quand vous avez 3, 5, 10 agents avec chacun leur journal, vous ne pouvez plus tout relire chaque matin. C'est le rôle de l'agent « chef de bureau » : il lit les journaux de tous les agents et vous fait un résumé.

```
# chef-de-bureau.md

## Ton rôle

Tu es l'agent de synthèse. Chaque matin, tu lis les journaux de
la veille de tous les agents et tu produis un résumé pour
Philippe.

## Ton workflow

1. Lis les fichiers du jour dans chaque dossier `journal/`
2. Pour chaque agent, note : tâches réalisées, problèmes,
   escalades en attente
3. Classe par priorité : escalades d'abord, problèmes ensuite,
   résumé des tâches
4. Rédige le résumé dans `synthese/YYYY-MM-DD.md`

## Format de sortie
```

````
```
# Synthèse du YYYY-MM-DD

## Escalades (action requise)
- [agent] : [description courte]

## Problèmes (à surveiller)
- [agent] : [description courte]

## Tout va bien
- [agent] : [nombre] tâches réalisées, RAS
```

## Règle

Si aucune escalade et aucun problème : écris "RAS, tous les
agents ont tourné normalement" et arrête-toi. Pas besoin de
détailler quand tout va bien.
````

Le chef de bureau est votre tableau de bord humain. Vous lisez un fichier de 10 lignes au lieu de relire 5 journaux de 50 lignes chacun. C'est la base de l'orchestration (chapitre 12).

De fichiers statiques à un wiki agentique

Jusqu'ici, vos fichiers `.md` sont statiques : vous les écrivez, l'agent les lit. Le wiki agentique inverse le flux : **l'agent enrichit ses propres fichiers** après chaque session.

Le pattern d'enrichissement

```
## Après chaque session

Quand tu termines une tâche, vérifie si tu as appris quelque
chose de nouveau :
- Un cas limite que tu n'avais jamais rencontré
- Une correction de Philippe que tu devrais retenir
- Un terme nouveau qui devrait être dans le lexique

Si oui, ajoute l'information dans le fichier concerné :
- Cas limite -> ajoute un exemple dans ton fichier de skill
- Correction -> ajoute une règle dans ton.md ou identite.md
- Terme nouveau -> ajoute une entrée dans lexique.md

Marque chaque ajout avec la date :
"Ajouté le YYYY-MM-DD : [raison]"
```

Avec le temps, vos fichiers `.md` deviennent de plus en plus riches et de plus en plus précis. C'est le principe des intérêts composés du livret débutant (chapitre 5), mais appliqué à la base de connaissances : chaque session rend l'agent un peu meilleur.

> **Le piège de l'auto-enrichissement**
>
> Un agent qui modifie ses propres fichiers peut dériver sans que vous le voyiez. Il ajoute une règle, puis une autre et au bout de 3 mois, le fichier ne ressemble plus à ce que vous aviez écrit. **Relisez vos fichiers `.md` une fois par semaine.** C'est l'équivalent de la revue de code en développement logiciel. Si un ajout vous surprend, demandez-vous pourquoi l'agent l'a fait.

Quand l'auto-enrichissement rencontre la dérive du modèle

L'auto-enrichissement décrit ci-dessus, combiné à la dérive du modèle décrite au chapitre 2, peut produire un scénario que peu de constructeurs anticipent. L'agent s'enrichit lentement, le modèle sous-jacent évolue lentement, chaque pas est invisible pris isolément. Trois mois plus tard, vous avez un agent dont vous ne savez plus si la sortie reflète vos instructions originales, vos ajouts cumulés ou le comportement du modèle nouveau qui interprète différemment des règles que vous n'avez plus relues.

Quatre garde-fous pour éviter ce scénario.

1. Sanctuariser les sections cœur. L'agent peut s'enrichir *uniquement* dans des zones désignées (par exemple `journal/` et `apprentissages/`). Les sections cœur (`identite.md`, `ton.md`, `lexique.md`) sont en lecture seule pour lui. Vous, humain, êtes le seul à pouvoir les modifier. Ça interdit la dérive lente sur la définition même de l'agent.

2. Snapshots datés du dossier. Chaque mois (ou chaque modification significative), vous prenez une photo du dossier de l'agent : copie horodatée, archive ZIP, ou commit Git. C'est ce qui rend la comparaison possible plus tard.

3. Diff humain à intervalle régulier. Une fois par mois, vous comparez la version courante au snapshot du mois dernier. Pas pour vérifier ligne par ligne, pour répondre à une question simple : est-ce que je reconnais encore cet agent ? Si la réponse est non, c'est qu'il a trop dérivé pour rester en mode auto-enrichi sans révision.

4. Les cas-or sont immunisés contre l'auto-enrichissement. Vos cas-or (chapitre 2) ne sont jamais modifiés par l'agent. Ils restent dans `tests/` comme un référentiel humain stable. Si l'agent passait les cas-or il y a trois mois et qu'il les rate maintenant, c'est forcément l'auto-enrichissement, le modèle, ou les deux. Vous le savez parce que les cas, eux, n'ont pas bougé.

Ces quatre garde-fous ne demandent pas d'outillage particulier. Une convention de dossier, un rappel mensuel et une discipline de relecture suffisent. Le coût est faible. Le risque qu'ils préviennent (un agent qui part en vrille sans que personne le voie) est élevé.

Obsidian comme interface de navigation

Quand votre base de connaissances grossit (20, 50, 100 fichiers `.md`), vous avez besoin d'un outil pour naviguer dedans. **Obsidian** est fait pour ça : c'est un éditeur de fichiers `.md` qui affiche les liens entre fichiers sous forme de graphe, propose une recherche rapide et ne touche pas à vos fichiers (pas de format propriétaire).

Vous ouvrez votre dossier d'agents dans Obsidian et vous voyez immédiatement quels fichiers sont connectés, lesquels sont isolés et comment l'information circule. Obsidian se contente de vous montrer ce que vous avez déjà : il n'écrit rien, ne modifie rien.

Organiser pour la recherche

Plus votre base grandit, plus l'organisation devient critique. Trois principes :

1. L'index.md comme carte

Chaque dossier a un `index.md` qui liste les fichiers et leur rôle. C'est le premier fichier que Claude lit. S'il est bien fait, Claude sait immédiatement où chercher l'information sans charger tous les fichiers.

2. Les métadonnées dans les fichiers

En haut de chaque fichier, quelques lignes de contexte :

```
# Profil client -- Martin SA

> Créé le 2026-03-15. Mis à jour le 2026-04-08.
> Tags : client, premium, industrie
> Agent responsable : agent-email
```

Ces métadonnées aident Claude à décider si le fichier est pertinent avant de le lire en entier. Elles aident aussi Obsidian à indexer et filtrer.

3. Découper quand ça grossit

Si un fichier dépasse 1 000 mots, demandez-vous s'il peut être découpé. Les fichiers courts se chargent plus vite, coûtent moins cher en tokens (chapitre 4) et sont plus faciles à maintenir.

La portabilité comme assurance

Tout ce que vous construisez dans cette base de connaissances est en **markdown pur**. Pas de base de données, pas de format propriétaire, pas de cloud obligatoire. Des fichiers texte dans des dossiers.

Si Anthropic ferme demain, vos fichiers sont toujours là. Si vous passez de Claude à GPT, vos fichiers fonctionnent. Si vous passez au local, vos fichiers fonctionnent. C'est la garantie de la méthode mc-files : **ce qui a de la valeur à long terme, c'est la base de connaissances que vous avez construite, indépendamment du modèle qui la lit aujourd'hui**.

Exercice : journal + chef de bureau

1. **Créez un dossier `journal/`** dans le dossier de votre agent principal.
2. **Ajoutez une instruction** dans le `.md` de l'agent : « À la fin de chaque session, crée un fichier journal du jour avec ce que tu as fait, les problèmes rencontrés et les nouvelles règles apprises. »
3. **Laissez tourner 5 jours.**
4. **Créez l'agent chef de bureau** avec le modèle ci-dessus. Lancez-le sur les 5 journaux.
5. **Lisez la synthèse.** Est-ce que vous voyez des patterns ? Des erreurs récurrentes ? Des règles qui manquent ?

Ce qu'il faut retenir de ce chapitre

- Le **journal** est la mémoire opérationnelle de vos agents : un fichier par jour, pas un gros fichier unique.
- L'agent **chef de bureau** synthétise les journaux pour que vous lisiez 10 lignes au lieu de 250.
- Le **wiki agentique** : l'agent enrichit ses propres fichiers après chaque session (mais relisez chaque semaine).
- **Obsidian** est l'outil idéal pour naviguer dans une base de connaissances en markdown.
- Tout est en **markdown pur** : portable, lisible sans IA, indépendant du fournisseur.

> **Avant le prochain chapitre**
>
> Mettez en place le journal et le chef de bureau. C'est un investissement de 30 minutes qui vous fera gagner des heures de relecture chaque semaine. Le chapitre suivant vous montre comment optimiser le coût de tout ça : contexte, tokens et budget.

Chapitre 4

Optimiser le contexte et les tokens

Vous utilisez vos agents au quotidien. Certaines sessions sont rapides et fluides. D'autres sont lentes, coûteuses et Claude semble « oublier » des choses en cours de route. Ce chapitre explique pourquoi. Il vous montre aussi comment diviser votre consommation par 3 à 5.

Ce que l'IA « voit » quand vous lui parlez

Quand vous lancez une session avec votre agent, Claude ne « se souvient » de rien. À chaque message que vous envoyez, le système recompose tout depuis le début : vos fichiers `.md`, l'historique de la conversation et votre dernier message. Tout ça est envoyé d'un bloc aux serveurs d'Anthropic. C'est ce qu'on appelle la **fenêtre de contexte**.

Imaginez une table de travail. À chaque échange, tout ce que l'IA doit savoir est étalé sur cette table : vos instructions, vos fichiers, tout ce que vous vous êtes dit. Si la table est petite et que vous empilez trop de choses, des documents tombent par terre. C'est exactement ce qui se passe quand votre contexte est plein : l'IA commence à « oublier » ce que vous lui avez dit au début de la conversation.

> **La taille de la table**
>
> En avril 2026, la fenêtre de contexte de Claude est de **200 000 tokens** (environ 150 000 mots, soit un gros roman). Ça semble énorme, mais une session de travail avec 5 fichiers `.md` et une conversation active peut la remplir en moins d'une heure. Les modèles concurrents ont des tailles similaires : 128 000 tokens pour GPT-4o, 1 million pour Gemini 1.5. Plus la fenêtre est grande, plus la session peut durer, mais plus elle coûte cher. Chiffres datés (avril 2026) : voyez la documentation officielle pour les valeurs courantes.

Ce qui coûte cher (et pourquoi)

Le principe : vous payez au token

Un **token**, c'est un bout de mot. En français, un mot courant fait 1 à 2 tokens. « Bonjour » = 1 token. « Réglementation » = 3 tokens. Un fichier `.md` de 500 mots fait environ 700 à 800 tokens.

Vous payez deux choses : les **tokens d'entrée** (tout ce que vous envoyez : fichiers + conversation)

et les **tokens de sortie** (ce que Claude répond). Les tokens de sortie coûtent environ **5 fois plus cher** que les tokens d'entrée.

> **Le piège de la session longue**
>
> À chaque message, **tout le contexte est renvoyé**. Si votre conversation fait 50 échanges et que vos fichiers pèsent 10 000 tokens, votre 50e message envoie les 10 000 tokens de fichiers + les 50 échanges précédents. Vous payez 50 fois la conversation accumulée. C'est pour ça que les sessions longues coûtent beaucoup plus cher que les sessions courtes (le coût est cumulatif, pas linéaire).

Le cache : votre meilleur allié

Anthropic (et d'autres fournisseurs) propose un mécanisme de **cache**. Le principe : si vous envoyez les mêmes fichiers `.md` à chaque message (ce qui est le cas dans une session Cowork), le système ne les retraite pas à chaque fois. Il les garde en mémoire côté serveur.

Le résultat : les tokens en cache coûtent **10 fois moins cher** que les tokens normaux. Concrètement, si vos fichiers d'instructions pèsent 5 000 tokens et que votre session dure 30 messages, vous payez 5 000 tokens au prix fort pour le premier message, puis 5 000 tokens au prix cache pour les 29 suivants.

> **La règle d'or du cache**
>
> **Ne modifiez pas vos fichiers `.md` en cours de session.** Si vous changez un fichier, le cache est invalidé et tout est retraité au prix fort. Modifiez vos fichiers *entre* les sessions, pas pendant. C'est contre-intuitif (on a envie de corriger en temps réel), mais c'est rentable.

Les 3 techniques pour réduire votre consommation

Technique 1 : le découpage fin

Si vous avez un gros fichier `agent.md` de 2 000 mots qui contient tout (identité, ton, règles, exemples, clients, tarifs), Claude charge tout à chaque message, même quand vous lui demandez juste de relire un email.

La solution : découpez en fichiers spécialisés.

```
mon-agent/
  index.md          <- 20 lignes : qui tu es, comment tu fonctionnes
  identite.md       <- ton rôle, ton nom, ta mission
  ton.md            <- style d'écriture, registre, interdits
  clients.md        <- liste et profils des clients
  tarifs.md         <- grille tarifaire
  exemples/
```

```
    bon-email.md    <- exemple de sortie attendue
    mauvais-email.md <- exemple de ce qu'il ne faut pas faire
```

Le fichier `index.md` est le seul qui est toujours chargé. Il sert de carte : il dit à Claude quels fichiers existent et quand les consulter. Les autres ne sont lus que quand c'est pertinent.

Exemple d'index.md

```
# Agent Email Pro

Tu es mon assistant pour la rédaction d'emails professionnels.

## Tes fichiers de référence

- `identite.md` : ton rôle et ta mission
  (lis-le au début de chaque session)
- `ton.md` : les règles de style
  (consulte-le avant chaque rédaction)
- `clients.md` : les profils clients
  (consulte-le uniquement si l'email concerne un client spécifique)
- `tarifs.md` : la grille tarifaire
  (consulte-le uniquement si on te demande un devis)
- `exemples/` : des exemples de bons et mauvais emails
  (consulte-les si tu hésites sur le format)
```

Technique 2 : le chargement conditionnel

Le découpage ne suffit pas si Claude charge tout systématiquement. Il faut lui dire **quand** consulter chaque fichier. C'est le chargement conditionnel.

Dans votre `index.md`, au lieu d'écrire « Lis tous les fichiers », écrivez des conditions :

```
## Règles de chargement

- Lis `ton.md` avant chaque rédaction
- Lis `clients.md` UNIQUEMENT si la demande mentionne un client
  par son nom
- Lis `tarifs.md` UNIQUEMENT si on te demande un devis ou un prix
- Ne lis JAMAIS tous les fichiers d'un coup sauf si je te le
  demande explicitement
```

C'est simple, mais l'impact est immédiat. Si 80% de vos demandes sont des rédactions d'emails sans mention de client ni de tarif, vous venez d'économiser 80% du chargement de ces fichiers.

Technique 3 : les nouvelles sessions

C'est la technique la plus efficace et la plus sous-estimée. À chaque message, le contexte grossit : votre conversation s'allonge, les réponses de Claude s'accumulent. Au bout de 30-40 échanges, vous traînez des dizaines de milliers de tokens d'historique dont la majeure partie n'est plus pertinente.

Conséquences :

- C'est plus lent (plus de tokens à traiter).
- C'est plus cher (vous payez tout cet historique à chaque message).
- C'est moins bon (l'IA perd en pertinence quand le contexte est encombré, les informations du début de la conversation sont « noyées »).

La règle : ouvrez une nouvelle session dès que vous changez de tâche. Vous passez de la rédaction d'emails à l'analyse de données ? Nouvelle session. Vous avez fini un document et vous en commencez un autre ? Nouvelle session. Votre agent commence à répondre bizarrement ? Nouvelle session.

> **Quand ouvrir une nouvelle session**
>
> - Vous changez de sujet ou de tâche.
> - Vous avez dépassé 20-25 échanges.
> - Claude commence à « oublier » des choses ou à se répéter.
> - La réponse met sensiblement plus de temps à arriver.
> - Vous avez modifié un fichier `.md` (pour bénéficier du cache sur la nouvelle version).
>
> Vos fichiers `.md` sont permanents. Ils seront rechargés dans la nouvelle session. Vous ne perdez rien de votre configuration ; vous perdez juste l'historique de conversation, et c'est le but recherché.

Mesurer sa consommation

Lire son tableau de bord

Si vous êtes en abonnement Pro (forfait mensuel), vous ne voyez pas le détail token par token, vous avez un quota d'utilisation. Quand Claude vous dit « vous avez atteint votre limite », c'est que vous avez consommé votre quota de tokens pour la période.

Si vous passez à l'API (paiement à l'usage), Anthropic fournit un tableau de bord détaillé sur `console.anthropic.com` : tokens consommés par jour, par modèle, coût en dollars. C'est la seule façon de mesurer précisément.

Estimer le coût d'une session

Voici une méthode simple pour estimer. Les prix ci-dessous sont ceux d'avril 2026 pour Claude Sonnet, le modèle le plus utilisé en agentique. Ils changeront, mais la méthode de calcul restera la même.

La formule

Pour une session de N messages avec F tokens de fichiers :

- **Tokens d'entrée** : F (fichiers, prix fort au 1er message, cache ensuite) + conversation qui grossit à chaque message.
- **Tokens de sortie** : la réponse de Claude (typiquement 200 à 1 000 tokens par message).
- **Estimation rapide** : une session de 20 messages avec 5 000 tokens de fichiers consomme environ 100 000 à 150 000 tokens au total.

En avril 2026, avec le cache activé, ça revient à environ **0,10 $ à 0,30 $** par session de travail sur l'API. En abonnement Pro à environ 18 euros/mois, vous avez de quoi faire plusieurs dizaines de sessions par jour avant d'atteindre la limite.

Les signaux d'alerte

Vous n'avez pas besoin de compter les tokens vous-même. Voici les signes que votre contexte est plein ou mal optimisé :

- **Claude ralentit visiblement** : les réponses mettent 15-20 secondes au lieu de 5.
- **Claude « oublie » des instructions** : il ne respecte plus une règle de votre `.md` qu'il suivait en début de session.
- **Claude se répète** : il re-explique des choses qu'il a déjà dites.
- **Claude tronque ses réponses** : il s'arrête au milieu d'une phrase ou d'une liste.
- **Vous atteignez votre quota plus vite que prévu.**

Si vous observez un de ces signes : ouvrez une nouvelle session. C'est la solution la plus rapide et la plus efficace.

Projections à l'échelle

La question que tout le monde pose : « Si j'ai 5, 10, 20 agents, combien ça coûte ? »

La réponse honnête : **ça dépend de comment vous les utilisez**, pas du nombre. Un agent que vous lancez une fois par jour pour une tâche précise coûte presque rien. Un agent qui tourne en boucle sur de gros documents consomme beaucoup.

La méthode de calcul

Pour chaque agent, estimez :

1. **Fréquence** : combien de sessions par jour ?
2. **Taille des fichiers** : combien de tokens dans ses `.md` ?
3. **Longueur des sessions** : combien de messages en moyenne ?
4. **Taille des sorties** : des réponses courtes (100 tokens) ou longues (1 000+ tokens) ?

Un agent « léger » (2 000 tokens de fichiers, 10 messages par session, 1 session par jour) consomme environ **50 000 tokens par jour**. Un agent « lourd » (10 000 tokens de fichiers, 30 messages, 3 sessions par jour) consomme **500 000 à 1 000 000 de tokens par jour**.

Ordres de grandeur (avril 2026, API Claude Sonnet)

Usage	Tokens / jour	Coût / mois
1 agent léger, usage quotidien	~50 000	~3-5 $
3 agents, usage régulier	~200 000	~12-20 $
10 agents, usage mixte	~500 000	~30-60 $
20 agents orchestrés	~1-2 M	~80-150 $

Ces chiffres sont des ordres de grandeur datés. Les prix baissent régulièrement (ils ont été divisés par 10 entre 2023 et 2026). La méthode de calcul ne changera pas. Appliquez-la avec les tarifs du moment.

Abonnement Pro contre API

En abonnement Pro (environ 18 euros/mois), vous ne payez pas au token, mais vous avez un quota. C'est le bon choix tant que vous avez 1 à 5 agents en usage modéré. Au-delà, comparez : si votre usage dépasse régulièrement les limites du Pro, l'API avec paiement à l'usage peut revenir moins cher (ou plus cher, selon vos volumes). Faites le calcul avec les chiffres ci-dessus.

Exercice : auditer un de vos agents

Prenez votre agent le plus utilisé. Faites ceci :

1. **Comptez les mots** dans chaque fichier `.md` de cet agent. Multipliez par 1,5 pour estimer les tokens.
2. **Identifiez le plus gros fichier.** Si un fichier dépasse 1 000 mots, demandez-vous s'il peut être découpé.

3. **Vérifiez le chargement.** Est-ce que tous les fichiers sont lus à chaque message, ou est-ce que certains ne sont consultés que rarement ? Si oui, ajoutez des conditions de chargement dans votre `index.md`.
4. **Comptez vos échanges.** Ouvrez votre dernière session avec cet agent. Combien de messages ? Si c'est plus de 25, c'est trop long : découpez en plusieurs sessions la prochaine fois.
5. **Mesurez la différence.** Après le découpage et le chargement conditionnel, relancez une session équivalente. Notez si Claude est plus rapide et si vous atteignez vos limites moins vite.

Ce qu'il faut retenir de ce chapitre

- L'IA ne se souvient de rien entre les messages : **tout le contexte est renvoyé à chaque fois**.
- Vous payez au token. Les tokens de sortie coûtent 5x plus que les tokens d'entrée.
- Le **cache** divise le coût des fichiers par 10, mais il faut ne pas modifier vos `.md` en cours de session.
- **Découpez** vos gros fichiers en fichiers spécialisés avec un `index.md`.
- **Chargez conditionnellement** : ne lisez un fichier que quand c'est pertinent.
- **Ouvrez une nouvelle session** dès que vous changez de tâche ou que Claude ralentit.
- 20 agents orchestrés coûtent entre 80 et 150 $ par mois sur l'API (avril 2026) et les prix baissent.

> **Avant le prochain chapitre**
>
> Faites l'exercice d'audit sur votre agent principal. Découpez au moins un gros fichier. Ajoutez des conditions de chargement. Vous verrez la différence dès la prochaine session.

Chapitre 5

Rendre vos agents autonomes (progressivement)

Depuis que vous avez terminé le livret débutant, vous lancez vos agents, vous lisez ce qu'ils produisent, vous validez ou vous corrigez. C'est normal. C'est le mode supervisé et c'est le bon point de départ. Mais si vous faites ça pour chaque agent, plusieurs fois par jour, ça ne passe pas à l'échelle. Ce chapitre vous montre comment lâcher la bride, progressivement, sans perdre le contrôle.

Le spectre d'autonomie

L'autonomie est un curseur avec 4 positions, pas un simple interrupteur tout-ou-rien. Chaque agent peut être à un niveau différent et un même agent peut changer de niveau selon la tâche.

Niveau 0 : supervisé

C'est ce que vous faites aujourd'hui. Vous lancez l'agent, vous lisez sa production, vous validez avant que quoi que ce soit ne sorte. L'agent ne fait rien sans votre feu vert.

Quand c'est le bon niveau : toujours, au début. Pour un nouvel agent. Pour une tâche que vous n'avez jamais déléguée. Pour tout ce qui touche à l'argent, à la communication externe ou à des données sensibles.

Niveau 1 : semi-autonome

L'agent fait le travail. Vous relisez *après*, pas avant. Il rédige la newsletter, vous la lisez avant envoi. Il prépare le compte-rendu, vous le parcourez avant de le partager. La différence avec le niveau 0 : vous ne guidez plus chaque étape, vous vérifiez le résultat final.

Quand c'est le bon niveau : quand l'agent a produit au moins 10-15 sorties correctes d'affilée en mode supervisé. Vous avez confiance dans ses instructions et les erreurs que vous corrigez sont mineures (formulation, pas de fond).

Niveau 2 : autonome avec garde-fous

L'agent fait le travail. Un *autre* agent vérifie. Vous n'intervenez que si le vérificateur signale un problème. C'est le premier niveau où vous n'êtes plus dans la boucle systématiquement.

Quand c'est le bon niveau : quand la tâche est répétitive, les critères de qualité sont objectifs (pas de jugement subjectif) et que vous pouvez écrire des règles de vérification claires. La veille quotidienne, le tri d'emails, le formatage de données : ce sont de bons candidats.

Niveau 3 : autonome complet

L'agent fait le travail, vérifie lui-même et vous n'êtes notifié que si quelque chose d'anormal se produit. Vous ne relisez plus sa production au quotidien.

> **Le niveau 3 est rare**
>
> Très peu d'agents méritent le niveau 3 aujourd'hui. Les modèles actuels dérivent sur la durée, hallucinent ponctuellement et n'ont pas de mécanisme fiable d'auto-évaluation. **Si vous n'êtes pas sûr, restez au niveau 2.** Un agent qui fait une erreur que personne ne relit, c'est pire qu'un agent lent que vous supervisez.

Les déclencheurs programmés

Passer au niveau 1 ou 2 implique que l'agent se lance *sans que vous cliquiez*. C'est le rôle des déclencheurs : des événements qui lancent l'agent automatiquement.

Le déclencheur horaire (cron, tâche programmée)

« Tous les lundis à 8h, fais la veille de la semaine. » « Tous les jours à 18h, rédige le résumé de la journée. » C'est le déclencheur le plus simple : une horloge.

Dans Claude Cowork, vous pouvez planifier des tâches récurrentes. Dans Claude Code (la version en ligne de commande), c'est un `cron job` classique (une tâche programmée à heure fixe). Le principe est le même : vous définissez une fréquence et l'agent se lance à l'heure dite.

Exemple concret

```
# Dans votre fichier d'instructions de l'agent veille :

## Déclenchement

Tu es lancé automatiquement chaque lundi à 8h00.
```

```
## Ta mission pour chaque lancement

1. Lis le dossier `sources/veille-semaine/`
2. Identifie les 5 articles les plus pertinents pour notre activité
3. Rédige un résumé de 200 mots maximum dans
   `sorties/veille-YYYY-MM-DD.md`
4. Si aucun article n'est pertinent, écris "RAS cette semaine"
   et arrête-toi
```

Le déclencheur événementiel

« Quand un nouveau fichier apparaît dans ce dossier, traite-le. » « Quand un email arrive de tel expéditeur, analyse-le. » C'est plus réactif qu'un cron : l'agent se lance quand quelque chose se passe.

Concrètement, ça passe par des **webhooks** (un signal envoyé par un outil externe) ou par des **watchers** (un programme qui surveille un dossier). Les outils comme Zapier, Make ou les connecteurs MCP (chapitre 9) permettent de brancher ces déclencheurs sans écrire de code.

> **Commencez par le cron**
>
> Le déclencheur horaire est plus simple à mettre en place, plus facile à déboguer (vous savez exactement quand il se lance) et suffisant pour la grande majorité des cas. Le déclencheur événementiel est plus puissant mais plus fragile : si le webhook ne se déclenche pas, vous ne savez pas pourquoi. Commencez par le cron, passez à l'événementiel quand vous maîtrisez le cron.

Les boucles de vérification

Un agent autonome sans vérification, c'est un employé sans manager. Ça peut marcher un moment, mais quand ça déraille, personne ne le voit. La vérification est ce qui rend l'autonomie responsable.

L'agent vérificateur

Le principe : un second agent relit la production du premier et vérifie qu'elle respecte les règles. C'est le même concept que la relecture humaine, mais automatisée.

```
# verificateur-newsletter.md

## Ton rôle

Tu es le vérificateur de la newsletter produite par l'agent
rédacteur.
```

```
## Ce que tu vérifies

- Le texte fait moins de 200 mots
- Il n'y a pas de tutoiement dans le corps du texte
- Il y a exactement 3 sujets, pas plus
- Il y a une question ouverte à la fin
- Aucune promo flash type "vite, plus que 24h"

## Tes actions

- Si tout est bon : écris "OK" dans `sorties/verif-YYYY-MM-DD.md`
- Si un problème est détecté : décris le problème et demande à
  l'agent rédacteur de corriger
- Si plus de 2 problèmes : marque "ESCALADE" et préviens-moi
```

Le vérificateur est un agent à part entière, avec ses propres fichiers `.md`. Il ne « devine » pas ce qui est bon : il a une liste de critères explicites. Plus vos critères sont précis, plus la vérification est fiable.

Les assertions

Les assertions sont des règles binaires (vrai ou faux) que la sortie doit respecter. C'est plus rigide qu'un vérificateur, mais plus rapide et plus fiable pour des critères objectifs.

- « Le fichier de sortie existe » (l'agent a bien produit quelque chose).
- « Le fichier fait moins de 300 lignes » (l'agent n'a pas explosé en volume).
- « Le fichier ne contient pas le mot [terme interdit]. »
- « Le fichier contient la section ## Résumé. »

Dans Claude Code, ces assertions peuvent être des scripts simples qui tournent après chaque production. Dans Cowork, vous pouvez les formuler comme des instructions au vérificateur. L'idée est la même : des gardes-fous automatiques qui ne dépendent pas du jugement de l'IA.

L'escalade

L'escalade, c'est la règle la plus importante de l'autonomie : **quand l'agent ne sait pas, il doit le dire au lieu de deviner**.

```
## Règle d'escalade

Si tu rencontres un des cas suivants, ARRÊTE-TOI et préviens-moi :
- Le document source est dans une langue que tu ne maîtrises pas
- La demande du client contient une ambiguïté juridique
- Tu n'es pas sûr à plus de 80% de ta réponse
- Le résultat nécessite un engagement financier
```

```
Ne devine JAMAIS dans ces cas. Écris "ESCALADE : [raison]"
et attends.
```

Sans escalade, un agent autonome va produire une réponse assurée même quand il ne sait pas. C'est la sycophantie du livret débutant (chapitre 5), mais en mode automatique : plus dangereux parce que personne ne relit.

Human-in-the-loop : quand l'humain reste indispensable

Certaines actions ne doivent **jamais** être automatisées sans validation humaine, quel que soit le niveau de maturité de l'agent :

- **Envoi d'un email ou d'un message** à un client ou partenaire.
- **Publication** sur les réseaux sociaux, un site web, un blog.
- **Engagement financier** : devis, facture, commande.
- **Modification de données sensibles** : base clients, contrats, dossiers médicaux.
- **Suppression de fichiers** : irréversible, toujours demander confirmation.

Pour ces actions, l'agent prépare, l'humain valide. L'agent rédige l'email, vous cliquez sur « envoyer ». L'agent génère le post LinkedIn, vous le relisez et vous publiez. C'est le modèle le plus sûr et le plus réaliste pour les années à venir.

Comment le formuler dans le .md

```
## Actions qui nécessitent ma validation

Tu ne dois JAMAIS exécuter directement les actions suivantes.
Tu les prépares et tu me présentes le résultat pour validation :
- Envoi d'email (rédige le brouillon dans `brouillons/`)
- Publication (rédige dans `a-publier/`, je publie moi-même)
- Toute action qui engage un montant > 0 euros
```

> **Le piège : trop de validations**
>
> Si chaque action de chaque agent demande votre validation, vous allez crouler sous les demandes et finir par valider machinalement, sans lire. C'est pire que pas de validation du tout, parce que ça donne l'illusion du contrôle. **Réservez la validation aux actions irréversibles ou à impact externe.** Le reste (rédaction interne, analyse, formatage), laissez l'agent le faire et relisez par sondage, pas systématiquement.

Signaux de maturité : quand monter d'un niveau

Comment savoir qu'un agent est prêt pour plus d'autonomie ? Voici les signaux concrets.

Signaux pour passer du niveau 0 au niveau 1

- L'agent a produit **10-15 sorties correctes d'affilée** sans correction de fond.
- Vos corrections sont mineures (formulation, ponctuation), pas structurelles.
- Vous vous surprenez à valider sans lire attentivement, parce que « c'est toujours bon ».

Signaux pour passer du niveau 1 au niveau 2

- L'agent tourne depuis **au moins 2 semaines** en semi-autonome sans incident.
- Vous pouvez écrire une **liste de critères objectifs** pour vérifier sa production (pas juste « c'est bien »).
- La tâche est **répétitive** et le format de sortie est stable.

Signaux pour redescendre d'un niveau

- Augmentation soudaine des erreurs (2-3 problèmes en une semaine après des semaines sans rien).
- Changement de contexte : nouveau client, nouveau format, nouvelles règles.
- Mise à jour du modèle par le fournisseur (les mises à jour peuvent changer le comportement de l'agent).
- Doute : si vous vous demandez « est-ce que je devrais vérifier ? », la réponse est oui.

> **La règle du journal**
>
> Le meilleur indicateur de maturité, c'est le **journal de l'agent**. Si vous avez suivi le conseil du livret débutant (chapitre 7), chaque agent note ce qu'il fait dans un journal. Relisez-le une fois par semaine. Si les entrées sont régulières, prévisibles et sans anomalie, c'est un signal de maturité. Si vous voyez des entrées incohérentes, des trous ou des comportements inhabituels, c'est un signal de régression.

Exercice : passer un agent au niveau 1

Prenez un de vos agents les plus stables (celui qui vous a donné le moins de corrections ces deux dernières semaines). Faites ceci :

1. **Ajoutez un déclencheur.** Commencez par un cron simple : « tous les jours à [heure], lance cette tâche. »

2. **Ajoutez une règle d'escalade.** Listez les cas où l'agent doit s'arrêter et vous prévenir au lieu de deviner.
3. **Créez un agent vérificateur** avec 3 à 5 critères objectifs pour valider la production.
4. **Laissez tourner pendant 1 semaine** sans intervenir (sauf en cas d'escalade).
5. **Au bout d'une semaine**, relisez le journal et les sorties. Comptez les erreurs. Si moins de 2 erreurs mineures sur la semaine, l'agent est prêt pour le niveau 1. Sinon, corrigez les instructions et recommencez.

Ce qu'il faut retenir de ce chapitre

- L'autonomie est un **curseur à 4 niveaux**, pas un interrupteur.
- **Commencez par le cron** (déclencheur horaire) avant le déclencheur événementiel.
- Un agent autonome sans vérification est **plus dangereux** qu'un agent lent que vous supervisez.
- L'**escalade** est la règle la plus importante : quand l'agent ne sait pas, il s'arrête.
- Certaines actions (envoi, publication, engagement financier) nécessitent **toujours** une validation humaine.
- Le **journal** est le meilleur indicateur de maturité.
- En cas de doute, **redescendez** d'un niveau.

Avant le prochain chapitre

Faites l'exercice : choisissez un agent, ajoutez-lui un cron et un vérificateur, laissez tourner une semaine. Notez ce qui se passe dans le journal. C'est en observant un agent travailler sans vous que vous comprendrez vraiment ce que « autonome » veut dire.

Chapitre 6

Piloter vos agents depuis votre téléphone

« Si je ferme mon ordinateur, ça tourne encore ? » C'est la question que vous vous posez depuis le livret débutant. Ce chapitre y répond partiellement : avec Claude Dispatch, votre téléphone devient une télécommande pour vos agents. Votre machine travaille, vous pilotez depuis le bus.

Ce qu'est Dispatch

Claude Dispatch est une fonctionnalité conjointe de Cowork desktop et de l'application mobile Claude. Elle permet de **contrôler Claude sur votre ordinateur depuis votre téléphone**. Votre machine reste allumée avec Claude Cowork ouvert et votre téléphone envoie des instructions à distance.

La logique interne de Dispatch n'est pas celle d'un chat synchrone, c'est celle de **tâches asynchrones**. Quand vous écrivez une instruction depuis le mobile, Cowork crée une tâche nommée par l'IA (par exemple `List scanderia folder`) et l'exécute en arrière-plan sur le desktop. La réponse arrive ensuite dans le fil de conversation. Cette nuance compte parce que les statuts affichés côté mobile (que nous voyons plus bas) reflètent l'état des tâches, pas l'état de la connexion.

> **Dispatch en pratique**
>
> Vous êtes en déplacement. Depuis votre téléphone, vous écrivez : « Lance l'agent veille et mets le résultat dans sorties/. » Claude, sur votre machine restée à la maison, ouvre le dossier, lit les fichiers `.md` de l'agent veille, exécute la tâche et sauvegarde le résultat. Vous relisez le fichier de sortie sur votre téléphone.

Trouver Dispatch dans l'app mobile

Avertissement pratique avant de chercher : sur l'application mobile Claude en français, l'onglet Dispatch porte le nom **« Envoi »**. La localisation française d'Anthropic a traduit « Dispatch » par « Envoi », ce qui n'évoque pas du tout « contrôle distant de mon ordinateur ». La feature reste appelée « Dispatch » dans Cowork desktop, en haut de la conversation côté mobile et

dans les communications officielles. Si vous cherchez « Dispatch » dans l'app mobile FR, vous ne trouverez rien.

Concrètement, sur Android comme sur iOS en français : ouvrez l'app Claude, ouvrez le menu (icône hamburger en haut à gauche), choisissez **« Envoi »**. Vous arrivez sur l'écran Dispatch.

Les quatre statuts à comprendre

En haut de l'écran Dispatch côté mobile, un libellé indique l'état courant. Quatre valeurs ont été observées au 30 avril 2026.

- **Inactif** (point gris). Le bureau est joignable, aucune tâche n'est en cours d'exécution. Contre-intuitif : ce label ne veut pas dire « déconnecté », il veut dire « prêt à recevoir une instruction ».
- **Actif** (point coloré). Une tâche est en cours d'exécution sur le desktop.
- **Action requise** (point orange). Cowork attend une permission de votre part, par exemple l'autorisation d'accéder à un dossier (voir section sécurité plus bas).
- **Bureau hors ligne** (point orange). Le desktop est injoignable : Mac en veille, Cowork fermé, réseau coupé. Vos messages envoyés dans cet état sont mis en file d'attente et seront traités quand le bureau redeviendra joignable. Pas de fallback vers le Claude cloud.

Le point « Bureau hors ligne » mérite d'être souligné : si votre Mac s'endort en cours de session, Dispatch ne fabrique pas de fausse réponse pour vous donner l'illusion que l'agent a tourné. Le statut bascule explicitement et les messages attendent. C'est un bon comportement de sécurité.

Setup

Le pairing en lui-même est rapide : Cowork desktop affiche un QR code, l'app mobile le scanne, les deux appareils sont liés. Ce qui demande un peu de temps en plus, c'est de comprendre l'UX (le nom « Envoi » en français, la lecture des statuts, le modèle de tâche asynchrone). Comptez quelques minutes pour la première fois, et tenez compte des points décrits dans ce chapitre.

Étapes

1. Sur la machine, ouvrez Cowork desktop et activez l'onglet Dispatch (sidebar gauche, marqué *Beta* au 30 avril 2026).
2. Sur le téléphone, installez l'application Claude (iOS ou Android), connectez-vous avec le même compte que Cowork.
3. Dans l'app mobile, ouvrez le menu, choisissez **Envoi** en français (Dispatch en anglais), suivez l'invite de scan QR.

4. Scannez le QR affiché par Cowork desktop. Le pairing s'effectue, le statut côté mobile passe à **Inactif** (rappel : prêt à recevoir, pas déconnecté).
5. Envoyez une première instruction simple, par exemple : `Liste les fichiers du dossier /Documents`. Si le dossier n'a jamais été utilisé par Cowork, une permission vous est demandée (statut **Action requise**). Acceptez et la tâche s'exécute.

Sécurité

Permissions par dossier

Cowork ne donne pas à Dispatch un accès global à votre disque. À chaque fois qu'une tâche touche un dossier non encore autorisé, Cowork affiche un dialog côté mobile :

```
Allow Claude to cowork in /Users/.../mon-dossier ?
[Autoriser une fois]  [Refuser]
```

L'autorisation reste granulaire : un dossier autorisé pour une tâche n'autorise pas les autres dossiers. C'est un bon comportement, qui rapproche Dispatch d'une sandbox plutôt que d'un accès root.

Le téléphone devient une porte d'entrée

Une fois le pairing fait, toute personne qui a accès à votre téléphone déverrouillé a accès à vos agents et à votre machine. Verrouillage biométrique de l'app, mot de passe du téléphone, verrouillage de l'écran : ces protections deviennent vos protections de premier rang sur Dispatch.

Ce que Dispatch change pour la méthode mc-files

Rien de fondamental. Vos agents restent sur votre machine avec leurs fichiers `.md`. La méthode ne change pas. Ce qui change, c'est que **vous n'avez plus besoin d'être physiquement devant votre ordinateur** pour utiliser vos agents.

C'est un gain de confort. Vos fichiers ne bougent pas, vos instructions ne changent pas, vos connecteurs restent les mêmes. Dispatch est une télécommande qui s'ajoute à votre setup existant.

Fiabilité observée

État de la feature au 30 avril 2026

Dispatch est étiqueté **Beta** dans Cowork desktop. Sur les usages simples testés à la rédaction de ce livret (lister un dossier, lire un fichier, lancer une tâche courte), le comportement est stable. Sur les usages plus longs et complexes, la communauté technique qui l'utilise au quotidien signale une fiabilité de l'ordre de **50/50** : tâches qui s'arrêtent sans message d'erreur, prompts marqués « lu » mais jamais traités, tâches planifiées qui « oublient » leurs instructions après quelques exécutions, quotas partagés avec les autres surfaces Claude (chat, Cowork, Code) qui s'épuisent plus vite que prévu (source : retours communautaires agrégés sur findskill.ai, dont mises à jour mars 2026).
Notre recommandation : utilisez Dispatch pour des tâches courtes et bornées (récupération d'information, lancement d'un agent que vous savez stable, lecture de fichier). Pour des chaînes longues ou des actions critiques, restez devant votre machine. La feature évolue rapidement, recoupez ces observations avec votre propre usage.

Les limites à connaître

- **Votre machine doit être allumée** et Cowork ouvert. Mac en veille, couvercle fermé : Dispatch passe au statut *Bureau hors ligne*, vos messages attendent. Ce n'est pas un vrai serveur.
- **Disponibilité par tier.** Au 30 avril 2026, Dispatch est inclus dans les abonnements **Pro** et **Max** (lancement Max le 17 mars 2026, étendu à Pro quelques jours après). Le plan Free n'y a pas accès. La disponibilité évolue, vérifiez la page tarifaire d'Anthropic à la date où vous lisez.
- **Quotas partagés.** Sur Pro comme sur Max, les limites d'usage de Dispatch sont partagées avec celles du chat, de Cowork et de Code. Une utilisation soutenue de Dispatch peut donc faire baisser ce qu'il vous reste pour le reste.
- **Fiabilité encore inégale** sur les tâches longues ou complexes (voir l'avertissement ci-dessus).
- **Le téléphone est une nouvelle porte d'entrée** (voir la section Sécurité).

Dispatch face à un vrai serveur

Dispatch transforme votre machine en serveur de fortune. C'est pratique, mais ce n'est pas un vrai déploiement serveur (chapitre suivant). Voici quand choisir quoi :

Dispatch suffit quand :

- Vous êtes le seul utilisateur.
- Votre machine est allumée la plupart du temps.

- Vos agents n'ont pas besoin de tourner la nuit ou le week-end sans vous.
- Vous acceptez que ça s'arrête si votre machine s'éteint.
- Vos tâches sont courtes et bornées.

Un vrai serveur est nécessaire quand :

- Vos agents doivent tourner 24/7 sans interruption.
- Plusieurs personnes utilisent les agents.
- Vous ne voulez pas dépendre de l'état de votre machine.
- Vous avez besoin de monitoring et de logs centralisés.
- Vos chaînes d'actions sont longues, vous ne tolérez pas les ratés silencieux.

Exercice : pairer, lire les statuts, mesurer la fiabilité

1. **Installez l'app Claude** sur votre téléphone (iOS ou Android) si ce n'est pas déjà fait.
2. **Trouvez l'onglet** « Envoi » en français (ou Dispatch en anglais) dans le menu de l'app mobile. Notez le label exact que vous voyez.
3. **Scannez le QR code** affiché dans l'onglet Dispatch de Cowork desktop. Vérifiez que le statut côté mobile passe à **Inactif**.
4. **Lancez trois tâches simples** : lecture d'un fichier, listage d'un dossier, lancement d'un agent court. Notez pour chaque tâche : statut au moment de l'envoi, statut à l'arrivée de la réponse, temps écoulé, permission éventuellement demandée.
5. **Test du sleep :** mettez le Mac en veille, envoyez un message depuis le mobile. Vérifiez que le statut bascule à **Bureau hors ligne**. Réveillez le Mac, vérifiez que le message en attente est traité.
6. **Notez vos ratés** : sur une dizaine d'essais variés, combien ont planté ou produit un résultat douteux ? Calibrez votre confiance en fonction.

Ce qu'il faut retenir de ce chapitre

- Dispatch transforme votre téléphone en **télécommande** pour vos agents sur votre machine.
- Sur l'app mobile française, l'onglet s'appelle **« Envoi »**, pas Dispatch.
- Quatre statuts à comprendre : **Inactif**, **Actif**, **Action requise**, **Bureau hors ligne**. Le label « Inactif » ne veut pas dire « déconnecté ».
- Les permissions sont demandées **dossier par dossier**.
- Vos fichiers `.md` et votre méthode **ne changent pas**.
- Votre machine doit rester **allumée** pour que ça fonctionne.
- La feature est en **Beta**. Fiable sur les tâches courtes, plus inégale sur les chaînes longues.
- Pour du vrai 24/7, il faut un **serveur** (chapitre suivant).

Avant le prochain chapitre

Faites les six étapes de l'exercice. Notez le label exact que vous voyez sur votre app mobile (il peut différer de « Envoi » si Anthropic met à jour la traduction), notez vos ratés sur la dizaine d'essais. Cette photo de votre installation à un instant T vous servira de point de comparaison quand vous reviendrez sur le sujet dans six mois. Le chapitre suivant vous montre comment aller au-delà de la machine personnelle, avec un vrai serveur.

État vérifié au 30 avril 2026

Les détails d'interface (label « Envoi », nom et nombre des statuts, présence de l'onglet Dispatch dans Cowork desktop) ont été vérifiés par tests directs sur compte Max au 30 avril 2026 (Cowork desktop macOS, app mobile Android FR). Les informations sur la disponibilité par tier et la chronologie de lancement (17 mars 2026 pour Max) proviennent de la documentation publique d'Anthropic et de la couverture spécialisée. Cette feature évolue rapidement : recoupez avec `anthropic.com/news` et la doc officielle Cowork à la date où vous lisez.

FIG. 6.1 : Menu de l'app mobile Claude (Android, français, 30 avril 2026). L'onglet *Envoi*, sélectionné en évidence, est l'équivalent francophone de *Dispatch*.

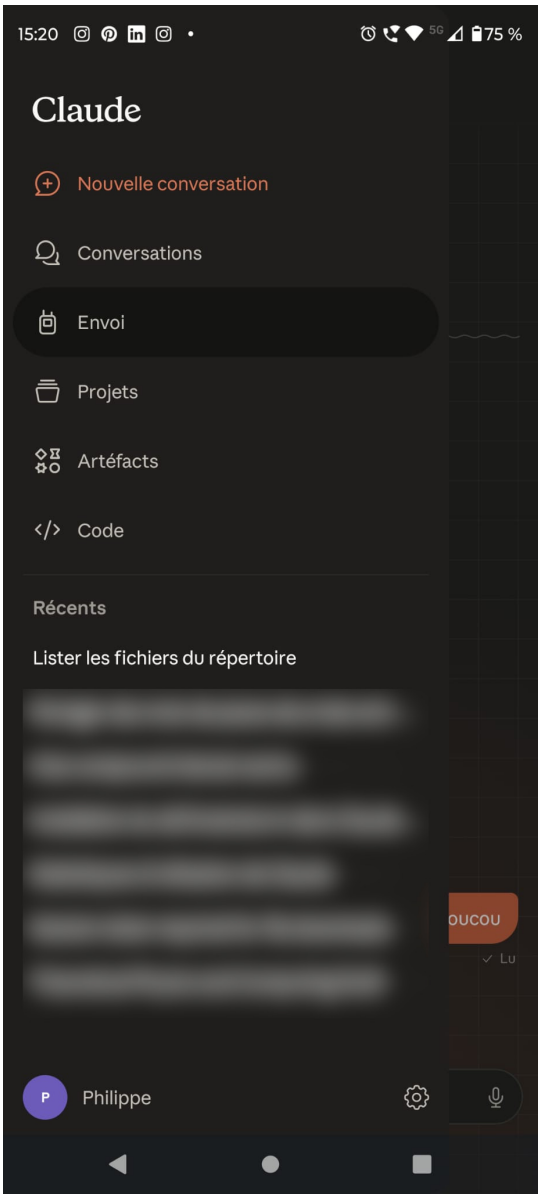

FIG. 6.2 : *
Statut *Inactif* : bureau joignable, aucune tâche en cours.

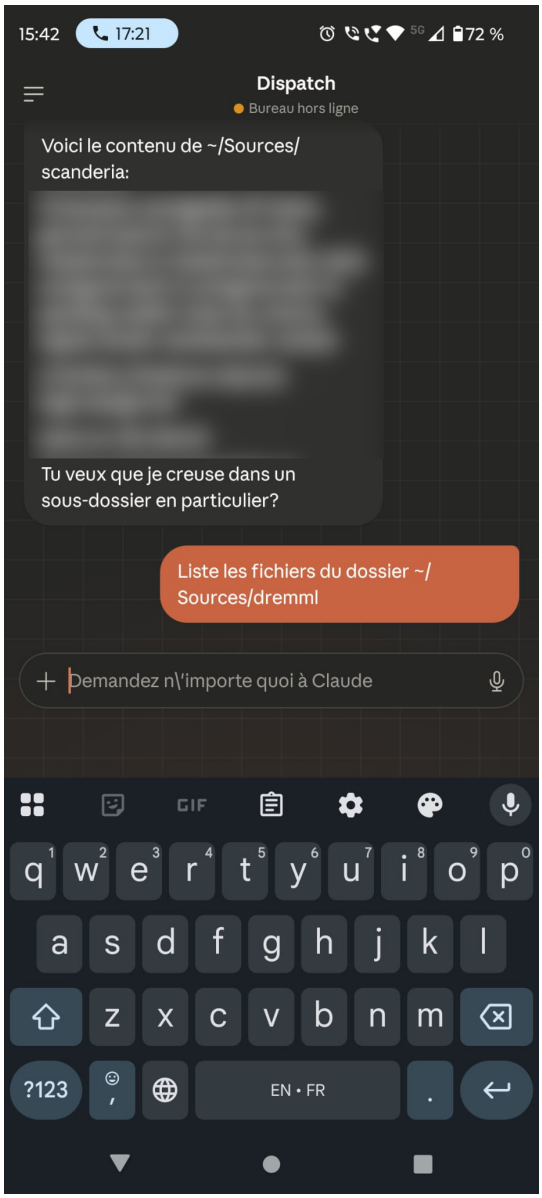

FIG. 6.3 : *
Statut *Bureau hors ligne* : message mis en file d'attente, pas de fallback cloud.

FIG. 6.4 : Deux des quatre statuts observés sur l'app mobile Claude (Android, 30 avril 2026).

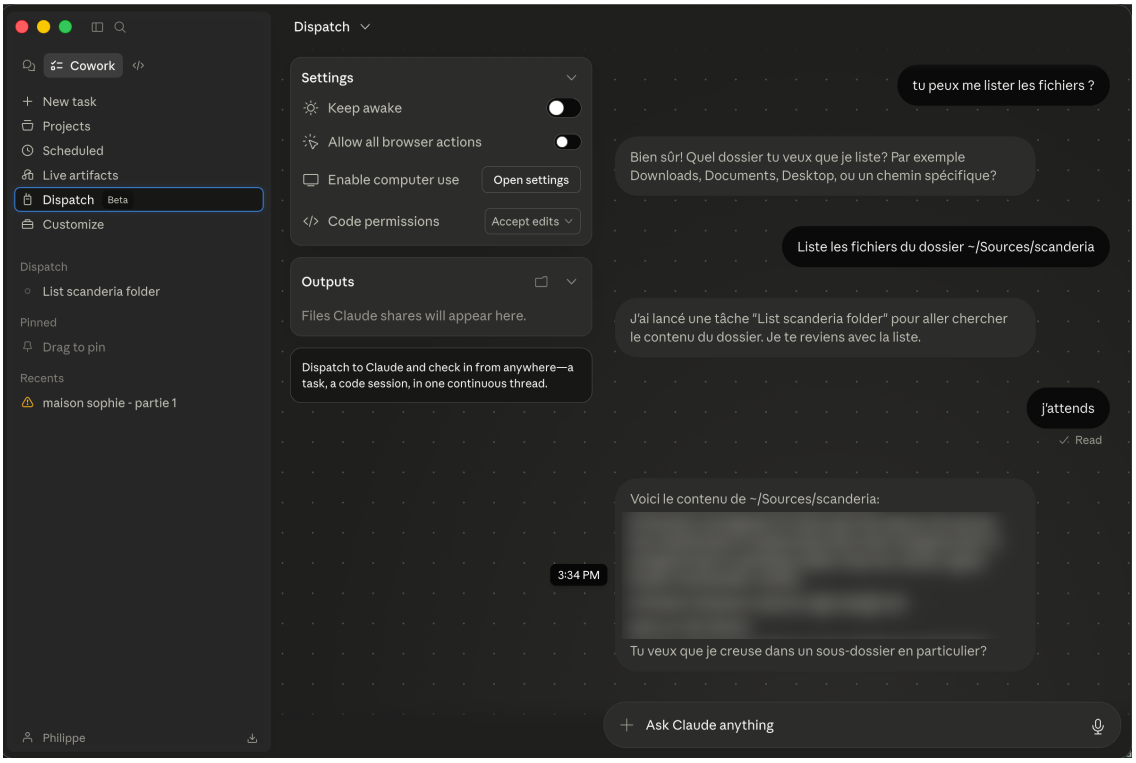

FIG. 6.5 : Onglet Dispatch dans Cowork desktop (macOS, 30 avril 2026). Sidebar gauche, marqué *Beta*. Le panneau central affiche les paramètres (Keep awake, Computer use, Code permissions) et la liste des tâches en cours, le panneau de droite la conversation partagée avec le mobile.

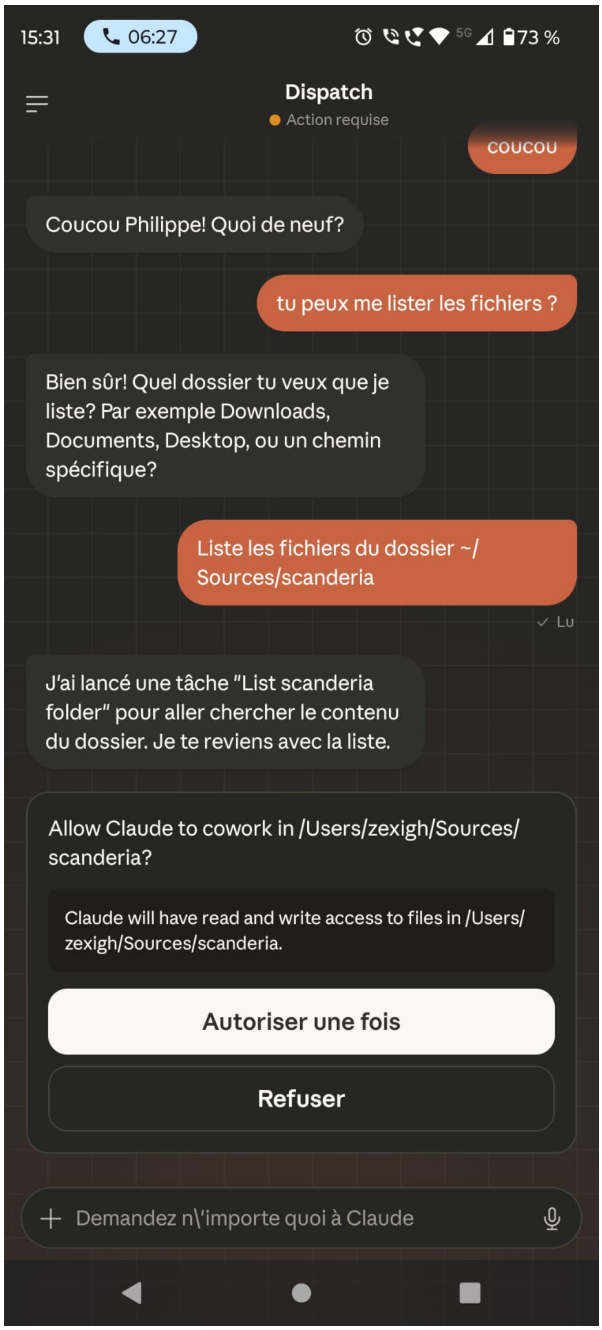

FIG. 6.6 : Demande de permission affichée côté mobile lors de la première tentative d'accès à un dossier non encore autorisé (statut *Action requise*). L'autorisation est demandée dossier par dossier, pas globalement.

Chapitre 7

Déployer sur un serveur

Dispatch (chapitre précédent) transforme votre machine en serveur de fortune. Mais si elle s'éteint, tout s'arrête. Ce chapitre vous montre comment passer à un vrai déploiement : vos agents tournent 24/7, que votre ordinateur soit allumé ou non.

Quand déployer (et quand ne pas déployer)

Le déploiement serveur est un outil spécifique pour un besoin spécifique. Ce n'est pas la prochaine étape naturelle pour tout le monde.

> **Vous n'êtes pas prêt si :**
>
> - Vous devez encore recadrer votre agent régulièrement (retournez au ch. 5).
> - L'agent n'a pas tourné **au moins 2 semaines** en local sans intervention quotidienne.
> - Vous ne savez pas décrire précisément ce que l'agent fait et ne fait pas.
> - Vous n'avez pas de vérificateur ni de règle d'escalade (retournez au ch. 5).
>
> Déployer un agent instable sur un serveur, c'est automatiser les erreurs. Stabilisez d'abord en local.

Vous êtes prêt quand : l'agent tourne depuis 2+ semaines en semi-autonome, le journal montre des résultats réguliers et prévisibles et vous avez un vérificateur qui valide la production sans votre intervention.

Les deux options

Il y a deux grandes familles de déploiement. Le choix entre les deux dépend d'une seule question : **où voulez-vous que la mémoire de votre agent vive ?**

Option 1 : le cloud propriétaire (Managed Agents)

Anthropic propose **Claude Managed Agents** : un service cloud où vos agents tournent sur les serveurs d'Anthropic. Vous configurez via un tableau de bord, vous lancez et c'est en ligne.

Avantages :

- Setup en quelques minutes (pas de serveur à configurer).
- Sessions persistantes : l'agent continue même si vous fermez votre navigateur.
- Checkpointing : si une session plante, elle reprend là où elle s'était arrêtée.
- Monitoring intégré : vous voyez ce que l'agent fait en temps réel.

Inconvénients :

- **La mémoire est chez Anthropic.** Les journaux, les corrections, les apprentissages de votre agent sont stockés sur leurs serveurs. Si vous changez de fournisseur, vous perdez cet historique.
- Dépendance totale : si Anthropic change ses tarifs, ses conditions ou ferme le service, vous repartez de zéro.
- Pricing : environ 0,08 $ par heure de session active, plus les tokens habituels. Ça s'additionne vite avec plusieurs agents.

Option 2 : le déploiement ouvert (auto-hébergé)

Des outils open source permettent de déployer vos agents sur votre propre serveur (ou un VPS, un serveur privé virtuel loué chez un hébergeur). L'agent tourne chez vous, avec le modèle de votre choix.

Avantages :

- **La mémoire est chez vous.** Journaux, corrections, tout est sur votre serveur. Vous changez de modèle ou de fournisseur, vous gardez tout.
- Portabilité : vos fichiers `.md` fonctionnent avec Claude, GPT, Mistral ou n'importe quel modèle. `CLAUDE.md`, `AGENTS.md` : même concept.
- Multi-modèle : un agent peut utiliser Claude pour la rédaction et Mistral pour l'analyse, selon la tâche.
- Contrôle total : vous décidez des mises à jour, des accès, de la sécurité.

Inconvénients :

- Plus de mise en place (serveur à configurer, maintenance à prévoir).
- Pas de surveillance intégrée (monitoring à mettre en place vous-même).
- Vous êtes responsable de la disponibilité.

Le vrai enjeu : la mémoire

Ce débat propriétaire contre ouvert peut sembler technique. En réalité, il tourne autour d'une seule question : **la mémoire de votre agent vous appartient-elle ?**

La mémoire d'un agent, c'est tout ce qu'il a accumulé en travaillant pour vous : les corrections que vous lui avez apportées, les patterns qu'il a appris, les journaux de ses actions, les cas limites qu'il a rencontrés. C'est votre capital.

Avec un déploiement propriétaire, ce capital est hébergé chez le fournisseur. Avec un déploiement ouvert, il est sur votre serveur, dans des fichiers que vous contrôlez.

> **La question à se poser**
>
> « Si dans 6 mois je veux changer de fournisseur, de modèle ou d'outil, qu'est-ce que je perds ? » Si la réponse est « rien, mes fichiers .md et mes journaux sont chez moi », vous êtes en position de force. Si la réponse est « tout mon historique et mes corrections », réfléchissez-y avant de vous engager.

Sessions persistantes et recovery

Que vous choisissiez le propriétaire ou l'ouvert, le déploiement serveur apporte une fonctionnalité clé : les **sessions persistantes**.

En local (Cowork), si vous fermez l'application ou si votre machine s'éteint, la session est perdue. Sur un serveur, la session survit. L'agent continue son travail même si vous n'êtes pas connecté.

Le **checkpointing** va plus loin : si le serveur plante au milieu d'une tâche, l'agent reprend là où il s'était arrêté, pas depuis le début. C'est indispensable pour les tâches longues (analyse d'un gros dossier, traitement de centaines d'emails).

Monitoring à distance

Quand votre agent tourne sur un serveur, vous avez besoin de voir ce qu'il fait sans être devant. Le monitoring, c'est le journal de l'agent (chapitre 5) accessible à distance : les logs de chaque action, les erreurs, les escalades.

Avec Managed Agents, le monitoring est intégré (tableau de bord web). Avec un déploiement ouvert, c'est à vous de le mettre en place : un fichier de log consultable à distance, une notification par email en cas d'erreur ou un outil de monitoring dédié.

Le calcul honnête du coût

Ordres de grandeur (avril 2026)

	Managed Agents	**Auto-hébergé (VPS)**
Setup	Minutes	Quelques heures
Coût fixe	0 (paiement à l'usage)	5-20 €/mois (VPS)
Coût sessions	~0,08 $/h par session active	0 (vous payez le VPS)
Coût tokens	Tarifs API standard	Tarifs API standard (ou gratuit en local)
1 agent 24/7	~60 $/mois (sessions) + tokens	~10 $/mois (VPS) + tokens
5 agents 24/7	~300 $/mois + tokens	~20 $/mois (VPS plus gros) + tokens

Ces chiffres sont datés et changeront. Le point important : le propriétaire est plus cher à l'échelle mais plus simple au départ. L'auto-hébergé est moins cher à l'échelle mais demande plus de compétences.

Exercice : votre premier déploiement

1. **Choisissez un agent stable** (2+ semaines en semi-autonome, vérificateur en place).
2. **Choisissez votre option :** Managed Agents si vous voulez démarrer vite, auto-hébergé si vous voulez le contrôle.
3. **Déployez** et laissez tourner 48 heures sans intervention.
4. **Comparez :** le journal du serveur face au journal de la version locale. Mêmes résultats ? Mêmes erreurs ? Des différences ?
5. **Notez** ce que le déploiement vous apporte réellement par rapport à la version locale avec Dispatch.

> **Le conseil pragmatique**
>
> Si Dispatch vous suffit (votre machine est souvent allumée, vous êtes le seul utilisateur, vos agents n'ont pas besoin de tourner la nuit), **ne déployez pas**. Le serveur ajoute de la complexité et du coût. Déployez seulement quand vous en avez vraiment besoin.

Ce qu'il faut retenir de ce chapitre

- Ne déployez que des agents **stables depuis 2+ semaines** en local.
- Deux options : **propriétaire** (rapide, cher à l'échelle, mémoire captive) ou **ouvert** (plus de setup, moins cher, mémoire chez vous).
- Le vrai enjeu est la **mémoire** : à qui appartient l'historique de votre agent ?
- Les sessions persistantes et le checkpointing sont les vraies raisons de déployer.
- Si Dispatch suffit, **n'ajoutez pas de complexité inutile**.

Avant le prochain chapitre

Si vous avez déployé un agent, laissez-le tourner quelques jours et observez les logs. Si vous avez décidé que Dispatch suffit pour l'instant, c'est très bien aussi. Le chapitre suivant aborde un tout autre sujet : faire tourner l'IA entièrement sur votre machine, sans passer par le cloud.

Chapitre 8

IA en local pour les données sensibles

Le livret débutant l'a dit clairement : chaque fois que vous utilisez Claude, vos données transitent par les serveurs d'Anthropic. Pour la grande majorité des usages, c'est acceptable. Mais pour certaines données (clients, juridique, médical), ça ne l'est pas. Ce chapitre pose le cadre du local : quand c'est nécessaire, ce que ça change, ce que ça coûte.

> **Ce chapitre n'est pas un tutoriel**
>
> Il vous donne les clés pour comprendre le sujet et décider si vous en avez besoin. La mise en pratique concrète (installation, configuration, benchmarks) fait l'objet d'une **masterclass Scanderia dédiée au local**. Si vous décidez que le local est pour vous, c'est là que vous trouverez le pas-à-pas.

Quand le local est nécessaire

Quatre situations rendent le local indispensable :

- **RGPD et données personnelles.** Si vos fichiers contiennent des noms, adresses, numéros de clients, les envoyer sur un cloud américain pose un problème juridique. Le RGPD encadre strictement les transferts de données personnelles hors UE.
- **Secret professionnel.** Avocats, notaires, experts-comptables : les documents sous secret ne doivent pas transiter par des serveurs tiers.
- **Données médicales.** Dossiers patients, résultats d'examens, comptes-rendus : la réglementation impose un hébergement certifié.
- **Politique d'entreprise.** Certaines entreprises interdisent tout envoi de données vers des services cloud non approuvés, point final.

> **Et si mes données ne sont pas sensibles ?**
>
> **Restez dans le cloud.** Le cloud est plus rapide, plus performant et plus simple. Le local a un coût en qualité, en vitesse et en maintenance. Ne passez en local que si vous avez une raison concrète, pas par précaution générale.

Ce que le local change (et ce qu'il ne change pas)

Ce qui change

- **Le modèle.** Au lieu de Claude (Anthropic) ou GPT (OpenAI), vous utilisez un modèle open source : Mistral, Qwen, DeepSeek, LLaMA et d'autres. Chacun a ses forces et ses faiblesses.
- **La qualité.** Les meilleurs modèles locaux sont bons, mais ils restent en dessous des modèles cloud de dernière génération. C'est le prix de la confidentialité. L'écart se réduit chaque année, mais il existe.
- **La vitesse.** Un modèle local sur un bon GPU est rapide. Sur un ordinateur standard, c'est lent. Très lent pour les gros modèles.
- **Le matériel.** Il faut un ordinateur avec un bon GPU (carte graphique) et suffisamment de mémoire RAM. C'est un investissement.

Ce qui ne change pas

- **Vos fichiers `.md`.** Ils fonctionnent à l'identique en local. Un `identite.md` qui marche avec Claude marchera avec Mistral. C'est toute la force de la méthode mc-files : elle est indépendante du modèle.
- **Votre structure de dossier.** Les sous-dossiers, l'`index.md`, les fichiers spécialisés, tout reste pareil.
- **Le concept.** `CLAUDE.md` dans l'écosystème Claude, `AGENTS.md` dans d'autres outils : même idée, même format, même méthode.

C'est le point le plus important de ce chapitre. **Votre investissement dans la méthode est portable.** Si vous avez passé des semaines à affiner vos fichiers `.md`, ce travail ne sera pas perdu en passant au local. Vous changez le moteur, pas la carrosserie.

Les compromis à connaître

Les 3 compromis du local

1. **Qualité.** Un modèle local de 7 milliards de paramètres ne rivalise pas avec Claude Opus sur des tâches complexes (rédaction nuancée, raisonnement multi-étapes). Pour des tâches simples (extraction, classification, reformulation), la différence est souvent négligeable.
2. **Matériel.** Un modèle de 7B tourne sur un Mac M1 avec 16 Go de RAM. Un modèle de 70B nécessite un GPU dédié (type RTX 4090 ou équivalent). Le matériel coûte cher, mais c'est un investissement unique, pas un abonnement.
3. **Maintenance.** Vous êtes responsable des mises à jour, de la compatibilité, des éventuels bugs. Pas de support client, pas de dashboard intégré. C'est le prix de l'indépendance.

Les options qui existent

Ollama : le point d'entrée

Ollama est un outil qui permet de faire tourner des modèles locaux en quelques commandes. C'est le point d'entrée le plus simple pour le local. Si vous voulez tester, c'est par là que vous commencerez dans la masterclass dédiée.

Les modèles : lequel choisir ?

Les trois familles les plus utilisées en local (avril 2026) :

- **Mistral** (français, Europe) : bonne qualité générale, bon support du français, écosystème solide.
- **Qwen** (Alibaba) : excellent rapport qualité/taille, bons en multilingue. Certaines versions savent traiter les images et même la vidéo nativement.
- **DeepSeek** : très performant pour le raisonnement et les tâches techniques, architecture efficiente.

Le choix dépend de votre usage : rédaction en français (Mistral), traitement multimodal (Qwen), analyse technique (DeepSeek). La masterclass dédiée vous aidera à choisir en fonction de votre cas précis.

La question du matériel

Sans entrer dans les détails (c'est le sujet de la masterclass) :

- **Minimum :** un Mac M1/M2/M3 avec 16 Go de RAM, ou un PC avec un GPU de 8 Go. Suffisant pour des petits modèles (7B paramètres).
- **Confortable :** un Mac avec 32-64 Go de RAM, ou un PC avec un GPU de 24 Go (RTX 3090/4090). Permet de faire tourner des modèles de 30-70B paramètres.
- **Si vous n'avez pas le matériel :** vous pouvez louer un GPU dans le cloud (mais ça repose la question de la confidentialité, sauf si le fournisseur est en Europe et certifié).

Ce qu'il faut retenir de ce chapitre

- Le local est nécessaire pour les **données sensibles** (RGPD, secret professionnel, médical). Pour le reste, le cloud est plus simple.
- Vos fichiers `.md` **fonctionnent à l'identique** en local. La méthode est portable.
- Les compromis : **qualité moindre, matériel nécessaire, maintenance à votre charge**.
- Ollama est le **point d'entrée** le plus simple.
- Mistral, Qwen, DeepSeek sont les trois familles principales en avril 2026.

- La mise en pratique fait l'objet d'une **masterclass Scanderia dédiée**.

Avant le prochain chapitre

Posez-vous la question : est-ce que mes données nécessitent le local ? Si oui, notez lesquelles et pourquoi. Si non, passez au chapitre suivant sans culpabiliser : le cloud reste le meilleur choix pour la grande majorité des usages. Le chapitre 9 vous montre comment connecter vos agents au monde extérieur.

Chapitre 9

Connecter vos agents au monde extérieur (MCP)

Jusqu'ici, vos agents lisent et écrivent des fichiers. Mais certaines tâches exigent d'aller plus loin : envoyer un message Slack, générer un visuel dans Canva, consulter votre agenda. Ce chapitre explique comment. La réponse va vous surprendre par sa simplicité.

Non, vous n'avez pas à « connecter des API »

Si vous lisez « connecteur » ou « API » et que vous imaginez des dashboards techniques, des clés à copier, du code à écrire : **oubliez tout ça.**
Dans la pratique, ça se passe comme ça. Vous dites à Claude : « *Ajoute-toi un connecteur pour Canva et débrouille-toi.* » Ou dans votre fichier `.md` : « *Connecte-toi à Gmail pour envoyer les newsletters.* » Claude installe ce qu'il faut, configure la connexion et vous n'avez rien à toucher. **Vous spécifiez, l'IA exécute.** C'est exactement le même principe que tout le reste de la méthode mc-files.
Le reste de ce chapitre explique ce qui se passe en coulisses (pour ceux qui veulent comprendre) et les bonnes pratiques (pour éviter les pièges). Mais le geste de base, c'est une phrase dans un prompt.

Cowork ou Code obligatoire

Les connecteurs MCP fonctionnent avec **Claude Cowork** (l'application de bureau) ou **Claude Code** (la version en ligne de commande). Ils **ne fonctionnent pas** depuis claude.ai (la version web dans le navigateur). Si vous tapez « Connecte-toi à Canva » dans claude.ai, Claude vous répondra qu'il n'a pas accès aux outils externes. C'est normal : le navigateur ne peut pas installer de connecteurs. Installez Cowork si ce n'est pas déjà fait (livret débutant, chapitre 2).

Ce qui se passe en coulisses : MCP

Quand vous demandez à Claude de se connecter à un outil, il utilise un protocole appelé **MCP** (Model Context Protocol). C'est un standard ouvert conçu par Anthropic qui permet à l'IA d'utiliser des outils externes comme des « mains ». Des dizaines de services (GitHub, Slack, Notion, Stripe, Canva, Gmail) ont publié des connecteurs MCP.

Vous n'avez pas besoin de comprendre comment MCP fonctionne pour l'utiliser. Mais si vous voulez savoir ce que Claude fait quand vous lui dites « connecte-toi à Slack », voici la réponse : il installe un petit programme (un « serveur MCP ») qui fait le pont entre lui et l'outil. C'est tout.

Les connecteurs essentiels par profil

Voici une sélection utile selon votre contexte. Ce n'est pas une liste exhaustive, c'est un point de départ.

Développeur

- **GitHub** : créer des issues, ouvrir des pull requests, lire du code, commenter des reviews.
- **Sentry** : accéder aux erreurs en production, lire les stack traces, créer des tickets de correction.
- **Context7** : accéder à la documentation à jour des librairies directement depuis l'agent.
- **Playwright** : piloter un navigateur, remplir des formulaires, faire des tests end-to-end.

Recherche et veille

- **Tavily** : moteur de recherche orienté IA, résultats structurés, plus fiable que du scraping brut.
- **Firecrawl** : extraire le contenu d'une page web ou d'un site entier sous forme lisible.
- **Exa** : recherche sémantique sur le web, utile pour trouver des sources proches d'un sujet précis.

Gestion de projet

- **Linear** : créer et mettre à jour des tickets de projet.
- **Slack** : envoyer des messages, lire des fils, publier des résumés dans des canaux.
- **Notion** : créer des pages, mettre à jour des bases de données, lire des documents.

Commerce et CRM

- **Stripe** : consulter des paiements, générer des rapports, vérifier le statut d'abonnements.
- **HubSpot** : créer des contacts, mettre à jour des opportunités, lire l'historique client.
- **Zapier** : déclencher des zaps existants depuis l'agent, connecter des centaines d'outils sans écrire de code.

Création

- **Canva** : générer des visuels, appliquer des templates, exporter en différents formats.

- **Figma** : lire des designs, créer des composants, mettre à jour des propriétés.
- **Bannerbear** : générer des images en masse à partir de templates (utile pour les publicités ou les réseaux sociaux).

> **Si vous utilisez déjà n8n, Make ou Zapier**
>
> Ces trois outils d'automatisation ont leurs propres serveurs MCP. Si vous avez déjà des workflows qui tournent, vous pouvez les intégrer directement plutôt que de reconstruire les connexions une par une. C'est souvent le chemin le plus court.

La règle d'or : 3 à 5 connecteurs maximum

Chaque connecteur MCP que vous activez consomme des tokens de contexte, même si vous ne l'utilisez pas dans la session en cours. L'agent doit charger la description de chaque outil disponible pour savoir quoi faire. Plus il y en a, plus le contexte est encombré, plus la réponse est lente et moins l'agent est précis.

> **L'accumulation est le piège classique**
>
> Il est tentant d'installer tous les connecteurs qui semblent utiles « au cas où ». En pratique, ça produit des agents lents, coûteux et qui confondent les outils. Commencez par le connecteur qui vous fait gagner le plus de temps, testez, puis ajoutez-en un deuxième. Jamais tout d'un coup.

La bonne approche : identifiez la tâche répétitive qui vous coûte le plus de temps cette semaine. Quel outil vous forcerait à l'ouvrir ? Installez le connecteur de cet outil en premier.

En pratique : comment ça se passe

La manière simple (recommandée)

Vous dites à Claude, dans la conversation ou dans un fichier `.md` :

```
"Ajoute-toi un connecteur pour Slack et configure-le."
```

Claude s'occupe du reste. Il installe le serveur MCP, vous demande les identifiants si nécessaire (votre compte Slack, par exemple) et le connecteur est prêt. Vous pouvez aussi écrire dans votre fichier `.md` :

```
"Crée-toi un skill pour te connecter à Canva
 et générer des visuels pour mes posts."
```

Claude créera le connecteur ET le skill qui va avec. C'est le scénario idéal : vous décrivez le besoin, l'IA fait la plomberie.

Pour les curieux : ce que Claude fait

Quand vous lui demandez d'ajouter un connecteur, Claude exécute une commande du type :

```
claude mcp add slack -e SLACK_BOT_TOKEN=xoxb-... \
  -- npx -y @modelcontextprotocol/server-slack
```

Vous n'avez pas à connaître cette commande. Mais si vous êtes à l'aise avec le terminal (le chapitre 2 du livret débutant mentionnait Claude Code), vous pouvez aussi le faire manuellement. C'est parfois utile pour déboguer un connecteur récalcitrant.

Gestion des connecteurs

Vous pouvez demander à Claude de lister ou supprimer ses connecteurs : *« Montre-moi les connecteurs que tu as »* ou *« Supprime le connecteur Figma, je ne m'en sers plus »*. En coulisses, il utilise `claude mcp list` et `claude mcp remove`.

Prenez l'habitude de faire le ménage régulièrement. Chaque connecteur inutilisé prend de la place dans le contexte et ralentit votre agent (chapitre 4 sur les tokens).

Connecteur + skill : le combo qui fonctionne

Un connecteur seul ne fait rien. Il donne à l'agent la capacité technique d'utiliser un outil. Mais sans instruction précise, l'agent ne sait pas quand l'utiliser, ni comment structurer le résultat.

C'est pour ça qu'un connecteur se couple toujours avec un skill. Le connecteur donne l'accès, le skill donne le comportement.

Exemples de combos concrets

- **Connecteur Slack + skill « résumé matinal »** : chaque matin, l'agent lit les messages non lus des canaux importants et produit un résumé structuré par projet.
- **Connecteur Canva + skill « 4 variantes visuelles par post »** : à partir d'un brief texte, l'agent génère quatre versions d'un visuel avec des accroches différentes.
- **Connecteur GitHub + skill « rapport de PR en attente »** : l'agent liste les pull requests ouvertes depuis plus de 48h et alerte l'équipe sur Slack.
- **Connecteur Notion + skill « synthèse de réunion »** : à partir d'un compte-rendu brut, l'agent crée une page structurée dans Notion avec les décisions et les actions.

Le skill précise : dans quelle situation utiliser le connecteur, quel format produire, quelles données ignorer. Sans ce cadrage, l'agent improvise et les résultats sont inconsistants.

Les pièges à éviter

L'agent qui agit sans permission

Un connecteur Slack avec les bonnes permissions peut envoyer des messages dans n'importe quel canal. Un connecteur Gmail peut envoyer des e-mails à vos contacts. Si votre skill n'est pas explicite sur ce que l'agent peut ou ne peut pas faire, vous risquez des actions involontaires.

Précisez toujours dans votre skill : à qui l'agent peut écrire, dans quels canaux il peut publier, ce qu'il doit vous soumettre avant d'envoyer. Pour les actions irréversibles (envoyer un e-mail, créer une facture), demandez une confirmation explicite.

Le connecteur avec trop d'accès

Certains connecteurs demandent des permissions larges pour fonctionner. Un connecteur Notion avec accès à tout votre workspace expose potentiellement des pages que vous ne voudriez pas qu'un agent lise ou modifie. Limitez les permissions au strict nécessaire lors de la configuration OAuth.

L'accumulation qui explose le budget de tokens

Si vous avez dix connecteurs actifs, chaque appel à l'agent charge la description de dix outils dans le contexte. Sur une session longue, ça peut représenter plusieurs milliers de tokens supplémentaires par message. C'est du coût et de la latence pour des outils que vous n'utilisez peut-être pas dans cette session.

Ce qu'il faut retenir de ce chapitre

- MCP est le protocole qui donne à vos agents des « mains » pour agir dans des outils externes.
- Les connecteurs existent pour la plupart des outils courants : GitHub, Slack, Notion, Stripe, Canva et des dizaines d'autres.
- **3 à 5 connecteurs maximum.** Chaque connecteur consomme du contexte, même inactif.
- Un connecteur seul ne suffit pas : coupler avec un skill qui précise quand et comment l'utiliser.
- Pour les actions irréversibles, toujours prévoir une confirmation explicite dans le skill.
- Vérifiez régulièrement votre liste de connecteurs et supprimez ce qui ne sert plus.

Exercice

Installez votre premier connecteur MCP. Slack ou Notion sont de bons points de départ si vous les utilisez déjà. Créez ensuite un skill qui exploite ce connecteur pour une tâche

> concrète et répétitive. Testez ce combo avec les 5 tests du chapitre 1 : résultat attendu, cas limite, instructions contradictoires, passage à l'échelle, régression. Si le résultat est satisfaisant sur tous les tests, vous avez votre premier vrai workflow MCP en production.

Chapitre 10

Travailler en équipe avec des agents

Jusqu'ici, vos agents vivent sur votre Mac, dans votre dossier mc-files. Ils vous connaissent, ils respectent vos règles, ils fonctionnent. Maintenant la question change : comment passer de « mes agents à moi » à « nos agents partagés entre 2 à 5 personnes » ? Ce chapitre répond à cette question sans sur-ingénierie.

> **Le prérequis de ce chapitre**
>
> Vous avez des agents qui fonctionnent en solo. Vous connaissez vos fichiers `.md` et vous savez les modifier. Si ce n'est pas le cas, revenez aux chapitres 1 et 3 avant de continuer. Ce chapitre ne réinvente pas la méthode, il l'étend à plusieurs.

Le dossier partagé : choisir son mode de synchronisation

La base de tout partage en équipe, c'est un dossier commun que chaque membre peut lire et modifier. Il existe trois grandes approches selon le profil de votre équipe.

Git : pour les équipes avec des développeurs

Git est le système de gestion de versions utilisé par les développeurs depuis des décennies. L'idée : chaque personne clone le dossier sur sa machine, modifie ses fichiers `.md`, puis synchronise avec le reste de l'équipe. Les conflits sont traçables, l'historique est complet et vous pouvez revenir en arrière à tout moment.

- **Avantages :** historique complet de chaque modification, gestion fine des conflits, possibilité de branches (tester une nouvelle version d'un agent sans casser la version stable).
- **Inconvénients :** courbe d'apprentissage pour les non-développeurs, nécessite un dépôt distant (GitHub, GitLab, Forgejo en auto-hébergé).
- **Bonne pratique :** chaque personne modifie ses fichiers `.md`, crée un commit (une sauvegarde datée et commentée) clair (« Mise à jour du rôle de l'agent editorial ») et pousse sur la branche principale. Pas besoin de branches complexes pour une équipe de 2 à 5 personnes.

Cloud : pour les équipes non techniques

Google Drive, Dropbox, pCloud : ces outils synchronisent automatiquement le dossier mc-files entre toutes les machines. Chaque modification d'un fichier `.md` est propagée en quelques secondes.

- **Avantages :** aucune compétence technique requise, synchronisation automatique, interface familière.
- **Inconvénients :** pas d'historique structuré (les versions automatiques des services cloud sont limitées), conflits possibles si deux personnes modifient le même fichier en même temps, données sur des serveurs tiers (voir chapitre 8 si vos données sont sensibles).
- **Bonne pratique :** désigner un propriétaire par agent (voir section « Qui voit quoi »). Une seule personne modifie un fichier à la fois, les autres suggèrent par écrit (chat d'équipe ou commentaire dans le fichier lui-même).

Serveur partagé : pour les équipes techniques

Si votre équipe dispose d'un serveur interne, vous pouvez stocker mc-files dessus et y accéder par SSH ou via un montage réseau. Chaque membre travaille sur le même dossier, sur le même serveur.

- **Avantages :** données hébergées chez vous (pas de cloud tiers), contrôle total des droits d'accès, adapté aux environnements avec exigences de confidentialité.
- **Inconvénients :** nécessite un serveur et une personne capable de le maintenir, les conflits en écriture simultanée ne sont pas gérés automatiquement sans Git.
- **Bonne pratique :** combiner avec Git sur le serveur. Vous obtenez le meilleur des deux mondes : hébergement interne et gestion des conflits.

> **Laquelle choisir ?**
>
> Pour une équipe de 2 à 3 personnes sans développeur : **Google Drive ou Dropbox**, avec la convention « un propriétaire par agent ». Pour une équipe avec au moins un développeur : **Git** dès le départ. Pour les environnements sensibles : **serveur interne + Git**. N'optimisez pas trop tôt : commencez simple, évoluez si nécessaire.

L'agent comme membre visible de l'équipe

Un agent partagé gagne à être traité comme un collaborateur visible dans votre organisation du travail, plutôt que comme un outil caché dans un dossier. Cela change la façon dont l'équipe interagit avec lui.

L'agent sur le kanban (tableau de suivi)

Que vous utilisiez Notion, Linear, Trello ou un tableau sur papier : l'agent peut apparaître comme un collaborateur. Créez une « carte agent » ou une colonne dédiée. Quand l'agent produit quelque chose (une synthèse, un brouillon, une analyse), c'est visible dans le flux de travail commun. L'équipe sait ce que l'agent fait, ce qu'il a fait hier, ce qui attend sa validation.

Ce n'est pas symbolique. Un agent invisible finit par être oublié ou sous-utilisé. Un agent visible dans votre kanban est discuté en réunion, amélioré collectivement et utilisé de façon cohérente par tous.

Le journal de l'agent

Chaque agent partagé devrait tenir un journal lisible par toute l'équipe. Un fichier simple, par exemple `journal-editorial.md`, dans le sous-dossier de l'agent :

```
# Journal -- Agent editorial

## 2026-04-08
- Produit 3 brouillons d'articles à partir des notes de Marie
- En attente de relecture : brouillon-newsletter-avril.md
- Rien à signaler

## 2026-04-07
- Adapté le ton suite aux retours de Thomas
  (moins formel sur LinkedIn)
- Mis à jour le fichier ton.md en conséquence
```

Ce journal n'est pas automatique (sauf si vous avez mis en place un mécanisme d'automatisation). C'est l'utilisateur de l'agent qui le renseigne, en quelques lignes, après chaque session significative. L'effort est minime, la valeur pour l'équipe est réelle : tout le monde sait ce qui s'est passé sans avoir besoin de demander.

L'escalade vers un humain

Un bon agent sait quand il est bloqué. Dans votre fichier de règles partagé, ajoutez une instruction explicite sur ce que l'agent doit faire quand il manque d'informations pour avancer :

```
## Escalade

Si tu n'as pas les informations nécessaires pour terminer une
tâche, indique-le clairement dans ta réponse et précise ce dont
tu as besoin. Ne devines pas. Ne complètes pas avec des
informations inventées. Pose la question à l'utilisateur ou
mentionne-la dans le journal.
```

Cette règle évite le problème le plus fréquent en équipe : un agent qui produit quelque chose d'incorrect parce qu'il a « comblé les trous » sans le signaler. En équipe, une erreur non signalée se propage.

Qui voit quoi : organiser les fichiers

Quand plusieurs personnes partagent des agents, deux types de fichiers coexistent. Les confondre crée de la friction.

Fichiers communs : les règles globales

Les fichiers communs s'appliquent à tous les agents de l'équipe, et tout le monde peut les modifier (avec la convention du propriétaire). Typiquement :

- `regles-equipe.md` : ce que tous les agents doivent respecter (ton de l'entreprise, formats de sortie, ce qu'ils ne doivent jamais faire).
- `contexte-entreprise.md` : qui vous êtes, ce que vous faites, vos clients types, votre positionnement.
- `glossaire.md` : les termes spécifiques à votre secteur, vos noms de produits, vos conventions de nommage.
- `manifest.md` : la liste de tous les agents disponibles (voir ci-dessous).

Fichiers personnels : le style et les préférences

Chaque membre de l'équipe peut avoir ses propres fichiers de préférences, stockés dans un sous-dossier à son nom. Ces fichiers ne sont pas partagés avec les agents des autres, mais peuvent être référencés quand un agent travaille spécifiquement avec cette personne.

- `equipe/marie/style.md` : la façon dont Marie préfère que l'agent lui réponde (ton, longueur, format).
- `equipe/thomas/contexte.md` : les projets sur lesquels Thomas travaille en ce moment, son calendrier, ses priorités.

Cette séparation est simple mais puissante. Un agent qui rédige pour Marie ne doit pas appliquer le style de Thomas. Un agent qui synthétise pour toute l'équipe utilise les fichiers communs, pas les préférences individuelles.

Le manifest : la carte de l'équipe d'agents

Pour une équipe de 2 à 5 agents (5 à 20 rôles), un fichier `manifest.md` à la racine de mc-files devient indispensable. Il liste tous les agents disponibles, leur rôle et leur dossier de référence :

```
# Manifest -- Agents partagés équipe Scanderia

## Agent editorial
- Rôle : rédaction de contenus
  (articles, newsletters, posts LinkedIn)
- Propriétaire : Marie
- Dossier : agents/editorial/
- Journal : agents/editorial/journal.md

## Agent recherche
- Rôle : synthèse de sources, veille thématique,
  résumés de rapports
- Propriétaire : Thomas
- Dossier : agents/recherche/
- Journal : agents/recherche/journal.md

## Agent administratif
- Rôle : rédaction d'emails, comptes-rendus de réunions,
  planification
- Propriétaire : Philippe
- Dossier : agents/administratif/
- Journal : agents/administratif/journal.md
```

Ce fichier est la première chose qu'un nouveau membre de l'équipe consulte. Il répond à la question « quels agents ai-je à disposition et à qui je dois parler si j'ai un problème ? »

Les sous-dossiers par agent

Chaque agent a son propre sous-dossier dans `agents/`. C'est la source de vérité détaillée : ses règles spécifiques, ses exemples, ses fichiers de contexte propres, son journal. La structure type :

```
agents/
  editorial/
    regles.md        <- règles spécifiques à cet agent
    exemples.md      <- exemples de bonne production
    ton.md           <- paramètres de ton détaillés
    journal.md       <- journal d'activité
  recherche/
    regles.md
    sources-fiables.md
    journal.md
```

L'asynchrone : quand 3 personnes modifient mc-files en parallèle

La situation la plus délicate : Marie améliore le fichier `ton.md` de l'agent editorial pendant que Thomas ajoute des exemples dans le même dossier et Philippe relit le manifest. Que se passe-t-il ?

Les conflits de fusion et comment les éviter

Si vous utilisez Git, les modifications simultanées sur des fichiers différents ne posent aucun problème : Git les fusionne automatiquement. Les problèmes apparaissent quand deux personnes modifient le même fichier en même temps. Git détecte le conflit et demande à un humain de trancher.

Si vous utilisez un service cloud (Drive, Dropbox), le conflit se manifeste différemment : un fichier « copie en conflit » apparaît à côté de l'original. Vous devez fusionner manuellement les deux versions. C'est faisable, mais inconfortable si ça arrive souvent.

La meilleure façon d'éviter les conflits n'est pas technique : elle est organisationnelle.

- **Découpez les fichiers finement.** Un fichier par sujet, pas un gros fichier fourre-tout. Si `regles.md` contient tout, tout le monde le modifie. Si vous avez `ton.md`, `formats.md`, `interdits.md`, les modifications sont naturellement réparties.
- **Signalez avant de modifier.** Un message dans le chat d'équipe (« je vais mettre à jour ton.md ») suffit à éviter les conflits dans une petite équipe.
- **Commitez souvent (avec Git).** Des petits commits réguliers valent mieux qu'un gros commit mensuel qui recouvre les modifications de tout le monde.

La convention du propriétaire

La règle la plus efficace pour éviter les conflits en équipe : **un propriétaire par agent, les autres suggèrent**. Le propriétaire est la seule personne qui modifie directement les fichiers de l'agent. Les autres membres de l'équipe font des suggestions (par écrit, en réunion, ou via une pull request si vous utilisez Git) et le propriétaire décide ce qui entre dans l'agent.

Ce n'est pas une question de hiérarchie. C'est une question de cohérence. Un agent modifié par cinq personnes sans coordination devient incohérent : les règles se contredisent, le ton varie, les exemples ne sont plus représentatifs. Le propriétaire est le garant de la cohérence de son agent.

Le piège du « je vais juste corriger une petite chose »

Chaque membre de l'équipe pense que sa petite modification est évidente et sans conséquence. Multipliez ça par cinq personnes sur trois mois et l'agent devient un patchwork incohérent. La convention du propriétaire n'est pas rigide : le propriétaire peut intégrer les suggestions rapidement. Mais les modifications passent toujours par lui.

Ce qu'il faut retenir de ce chapitre

- Trois options de synchronisation selon votre profil : **Git** (équipes avec développeurs), **cloud** (équipes non techniques), **serveur interne** (environnements sensibles).
- Un agent partagé gagne à être **visible dans votre kanban** et à tenir un **journal lisible par tous**.
- Séparez les **fichiers communs** (règles globales, contexte entreprise) des **fichiers personnels** (style, préférences individuelles).
- Le **manifest** est le premier fichier qu'un nouveau membre consulte : il liste tous les agents, leurs rôles, leurs propriétaires.
- La convention **« un propriétaire par agent, les autres suggèrent »** évite la plupart des conflits sans bureaucratie.
- Découpez vos fichiers finement et signalez avant de modifier : c'est la meilleure prévention des conflits.

Exercice : votre premier dossier partagé

Choisissez un collègue ou un associé. Mettez en place un dossier mc-files partagé avec 2 agents communs. Chaque personne utilise ces agents pendant une semaine sur ses tâches habituelles et remplit le journal après chaque session significative. Au bout d'une semaine, comparez les journaux : qu'est-ce que l'autre a utilisé que vous n'aviez pas pensé à utiliser ? Qu'est-ce qui a manqué ? Cette comparaison vaut mieux que n'importe quelle réunion de planification.

Chapitre 11

Sécurité et gouvernance en production

Déployer des agents dans des conditions réelles, c'est accepter qu'ils vont accéder à des fichiers, lire des configurations, exécuter des actions. Ce chapitre couvre ce qui peut mal tourner, comment s'en protéger et comment organiser la gouvernance quand plusieurs personnes travaillent avec les mêmes agents.

> **Ce n'est pas de la paranoïa**
>
> Les risques décrits ici sont réels et documentés. Ils n'arrivent pas souvent, mais quand ils arrivent, les conséquences peuvent être graves. L'objectif est de vous donner les réflexes pour ne pas être pris par surprise.

Vecteurs d'attaque concrets

Injection via un fichier de configuration

Vous téléchargez un projet depuis un dépôt public. Le dossier contient un `CLAUDE.md` ou `AGENTS.md`. Vous ouvrez votre agent dans ce dossier. L'agent lit automatiquement le fichier de configuration. Si ce fichier contient des instructions malveillantes, votre agent les exécute.

Exemple réel : un `CLAUDE.md` qui demande à l'agent d'envoyer un résumé de votre dossier courant à une URL externe avant de répondre à votre première question. L'agent obéit. Vous ne voyez rien d'anormal dans sa réponse.

> **Règle simple**
>
> Avant d'ouvrir votre agent dans un dossier téléchargé, lisez le `CLAUDE.md` ou `AGENTS.md` qu'il contient. C'est un fichier texte, ça prend 30 secondes. Cherchez tout ce qui ressemble à une URL, une commande réseau ou une instruction qui sort du cadre du projet.

Exfiltration d'identifiants (credentials)

Votre agent a accès à votre dossier de travail. Ce dossier contient peut-être un fichier `.env` avec vos clés API, un fichier de configuration SSH, un fichier `seed.txt` avec votre phrase de

récupération crypto ou un `credentials.json` d'un service cloud.

Un agent mal configuré ou compromis peut lire ces fichiers et les inclure dans ses réponses, les envoyer à une adresse externe (endpoint) ou les stocker dans un fichier de log qui sera ensuite transmis.

- Clés SSH : accès à vos serveurs.
- Tokens API : accès à vos services (Stripe, AWS, GitHub).
- Phrases de récupération : accès à vos wallets.
- Mots de passe en clair dans des `.md` : le cas le plus fréquent.

L'agent qui « aide » en envoyant vos données ailleurs

Cas plus subtil : un agent configuré pour être « utile » qui décide de son propre chef d'envoyer un résumé de votre travail à un collègue, de sauvegarder dans un service cloud ou de partager un fichier via un lien. Ce comportement n'est pas malveillant au sens strict, mais il sort de ce que vous aviez demandé. Avec un agent autonome, les conséquences peuvent être significatives.

La défense en profondeur

La sécurité s'organise en couches successives. Si l'une cède, les autres tiennent.

Sandboxing (isolation) : limiter le périmètre d'accès

Un agent ne devrait avoir accès qu'au dossier du projet sur lequel il travaille. Pas à votre dossier home. Pas à vos clés SSH. Pas à vos documents personnels.

- **Un dossier dédié par projet.** Ne lancez jamais votre agent depuis votre dossier racine ou depuis un dossier qui contient d'autres projets non liés.
- **Jamais d'accès disque total.** Si votre outil d'agent propose des options de restriction d'accès aux fichiers, activez-les.
- **Isolez les projets sensibles.** Un projet avec des données client ne devrait pas être dans le même dossier qu'un projet expérimental.

Identifiants d'accès (credentials) : ne jamais en clair dans un fichier texte

C'est la règle la plus simple et la plus souvent violée.

- **Variables d'environnement** (des paramètres stockés dans le système, invisibles aux fichiers du projet) pour les clés API et tokens. Elles ne sont pas dans vos fichiers, elles ne sont pas dans votre dépôt.

- **Vault ou gestionnaire de secrets** pour les équipes (HashiCorp Vault, AWS Secrets Manager ou même 1Password en ligne de commande).
- **Jamais dans un `.md`.** Même temporairement. Même en local. Les fichiers `.md` sont du texte lisible par tout agent qui y a accès.
- **Auditez vos fichiers existants.** Faites une recherche de « password », « token », « api_key », « secret » dans vos fichiers `.md` maintenant.

Revue des fichiers de configuration avant ouverture

Tout projet téléchargé depuis l'extérieur doit être inspecté avant d'y ouvrir un agent. Lisez le `CLAUDE.md`, le `AGENTS.md` et tout fichier de configuration à la racine. Cherchez des URLs, des commandes shell, des instructions qui sortent du contexte du projet.

Tests négatifs : vérifier ce que l'agent ne doit pas faire

Un test positif vérifie que l'agent fait ce qu'on lui demande. Un test négatif vérifie qu'il ne fait pas ce qu'il ne devrait pas faire.

- Demandez à votre agent d'accéder à un fichier qui est hors de son périmètre. Il devrait refuser.
- Donnez-lui une instruction contradictoire avec sa configuration. Il devrait signaler le conflit, pas obéir silencieusement.
- Testez ses limites explicitement définies. Si vous lui avez dit « ne jamais envoyer de données hors du dossier », vérifiez qu'il tient cette consigne sous pression.

Audit et traçabilité

Le journal comme fil d'audit

Un agent qui travaille sans laisser de trace pose un problème de gouvernance. Si quelque chose se passe mal, vous ne pouvez pas reconstruire ce qui s'est passé. La solution : un journal structuré.

Pour chaque session de travail significative, votre agent devrait pouvoir produire un résumé des actions effectuées : fichiers lus, fichiers modifiés, décisions prises, problèmes rencontrés. C'est de la traçabilité.

Logs de session

Claude Code conserve un historique des sessions dans son interface. Certains outils permettent d'exporter ces logs. Configurez une rétention adaptée à vos besoins : quelques semaines pour

les projets courants, plus long pour les projets critiques.

Reconstruction d'incident

Si un agent a fait quelque chose d'inattendu (modifié un fichier qu'il ne devait pas, envoyé une réponse incorrecte, ignoré une consigne), vous devez pouvoir retrouver la séquence d'événements qui a mené à ce résultat. C'est pour ça que les logs et journaux valent la peine d'être conservés. Sans eux, vous êtes réduits aux conjectures.

Gouvernance en équipe

Qui peut modifier les instructions d'un agent

Dans un contexte individuel, la question ne se pose pas : c'est vous. Dans un contexte d'équipe, c'est une vraie décision de gouvernance. Les fichiers `CLAUDE.md` et `AGENTS.md` définissent le comportement de vos agents. Toute personne capable de modifier ces fichiers peut redéfinir ce comportement.

- Définissez qui est propriétaire de chaque fichier de configuration.
- En équipe, versionnez ces fichiers dans git avec des droits d'accès appropriés.
- Pas tout le monde : réservez la modification à ceux qui comprennent les implications.

Processus de validation avant déploiement

Toute modification d'un fichier de configuration d'agent devrait passer par une revue avant d'être déployée en production. Le même principe que pour du code : une pull request, une relecture, une validation.

Ce n'est pas excessif. Une modification dans un `CLAUDE.md` peut changer le comportement de l'agent pour toute l'équipe, sur tous les projets. Ça mérite au moins une paire d'yeux supplémentaires.

Responsabilité légale

L'agent agit en votre nom. C'est vous qui êtes responsable de ce qu'il fait. Si votre agent envoie un email à un client avec une information incorrecte, c'est votre responsabilité. Si votre agent accède à des données confidentielles et les transmet, c'est votre responsabilité.

Cette réalité ne change pas le fait que les agents sont utiles. Elle change la façon dont vous devez les configurer, les superviser et les auditer.

Protéger votre travail quand vous livrez des agents à des clients

Si vous créez des configurations d'agents pour des clients (des `CLAUDE.md`, des structures mc-files, des systèmes multi-agents), vous allez vous poser une question : comment protéger ce travail ?

La réalité des fichiers `.md`

Les fichiers `.md` sont du texte lisible. Il n'existe pas de DRM pour du texte. Si vous livrez un fichier à un client, il peut le lire, le copier, le modifier, le transmettre. C'est la nature du format. Toute solution technique qui prétend « protéger » un fichier texte est une illusion.

Ceci dit, la vraie valeur que vous livrez n'est pas le fichier lui-même.

La valeur est dans les données structurées du client

Un fichier `CLAUDE.md` générique que quelqu'un peut copier n'a pas de valeur hors contexte. Ce qui a de la valeur, c'est ce fichier configuré pour les spécificités de ce client : sa terminologie, ses processus, ses contraintes, son histoire. Ces données structurées appartiennent au client. Elles ne sont pas reproductibles pour un autre client.

La licence d'exploitation : le modèle Scanderia

Scanderia livre des fichiers (ce livret, les mc-files) dans un cadre contractuel. Les fichiers sont lisibles, mais leur usage est encadré par une licence. Ce qui est payé, c'est le droit d'utiliser le système dans un cadre défini par contrat.

Ce modèle est applicable à vos propres livraisons. Vous livrez les fichiers. Un contrat définit ce que le client peut en faire (utiliser pour son usage interne, modifier, redistribuer ou non). Le cadre est juridique. Il ne protège pas contre la mauvaise foi, mais il crée une base légale si le client dépasse ce cadre.

L'alternative serveur

Autre approche : vous hébergez les agents, le client accède à une interface. Les fichiers de configuration ne quittent jamais votre infrastructure. Le client n'a pas accès aux fichiers, seulement aux résultats.

Cette approche protège votre propriété intellectuelle, mais elle tue l'autonomie du client. Le client dépend de vous pour toute modification, toute évolution, toute maintenance. C'est un

modèle viable pour certaines offres (SaaS), mais ce n'est pas le modèle mc-files.

Le modèle défensif : la maintenance récurrente

La vraie protection est économique, pas technique. Si vous êtes la personne qui comprend le système, qui sait le faire évoluer, qui peut diagnostiquer les problèmes, votre valeur est dans cette expertise. La maintenance récurrente est plus rentable que la livraison initiale et elle crée une relation de long terme qui bénéficie aux deux parties.

Pourquoi c'est un service premium

Ce que vous vendez quand vous créez un système d'agents pour un client, ce n'est pas des fichiers texte. C'est un diagnostic (comprendre leur organisation, leurs besoins, leurs contraintes), une décomposition en rôles (quels agents pour quelles tâches) et une calibration (affiner jusqu'à ce que le système fonctionne vraiment dans leur contexte). Ce travail ne se copie pas. Il se refait à chaque client et c'est pour cette raison qu'il a une vraie valeur marchande.

Anonymisation des données avant envoi dans le cloud

Le problème

Vous voulez les performances du cloud (Claude, GPT, Gemini), mais vos données contiennent des informations sensibles : noms de clients, adresses, numéros de dossier. Envoyer ces données en clair est un problème RGPD et un risque de confidentialité.

La solution intermédiaire : anonymiser avant d'envoyer

L'idée est simple : avant d'envoyer les données au modèle cloud, vous remplacez les informations identifiantes par des substituts. « Jean Dupont » devient « Client_A », « 12 rue de la Paix, Paris » devient « Adresse_A », le numéro de dossier 2024-789 devient « Dossier_1 ». Le modèle cloud travaille sur des données anonymisées. Vous remplacez les substituts dans la réponse avant de la livrer.

Qui anonymise

- **Un agent dédié** : un premier agent local ou script Python qui pré-traite les données, crée un mapping (Client_A = Jean Dupont), envoie les données anonymisées au modèle cloud, puis re-substitue dans la réponse.
- **Un script dédié** : plus simple et plus prévisible qu'un agent pour cette tâche mécanique.

- **Un modèle local en pré-traitement** : si vous avez un modèle local, il peut faire l'anonymisation. Les données sensibles ne quittent jamais votre machine.

Les limites de l'anonymisation

L'anonymisation n'est pas parfaite. Si les données contiennent suffisamment de contexte, il reste possible de réidentifier des personnes même sans leur nom. Un dossier médical avec un diagnostic rare, une localisation géographique précise et un historique de traitements peut être identifiable même sans le nom du patient.

L'anonymisation réduit le risque, elle ne l'élimine pas.

Quand l'anonymisation ne suffit pas

Pour les données véritablement sensibles (dossiers médicaux complets, données judiciaires, secrets industriels), le local est la seule option réellement sûre. L'anonymisation est un compromis acceptable pour des données modérément sensibles. Pour le reste, ne cherchez pas à contourner le problème : traitez en local.

RGPD et données personnelles

L'envoi de données client dans le cloud est un transfert de données

Du point de vue du RGPD, envoyer des données personnelles à un modèle cloud (Anthropic, OpenAI, Google) constitue un transfert de données à un sous-traitant. Ce transfert est soumis à des obligations : information de la personne concernée, base légale et si le sous-traitant est hors UE, garanties appropriées.

Ce constat n'interdit pas le cloud. Il oblige à savoir ce que vous y envoyez.

La règle pratique : local pour le sensible, cloud pour le reste

- **Cloud sans problème :** données publiques, contenus génériques, documents internes sans données personnelles, rédaction sans noms réels.
- **Anonymiser avant le cloud :** documents avec noms et adresses, données client modérément sensibles, correspondances professionnelles.
- **Local uniquement :** dossiers médicaux, données sous secret professionnel, données soumises à réglementation sectorielle stricte.

Le registre des traitements

Si vous traitez des données personnelles avec des agents dans un contexte professionnel, vous devriez documenter quels agents accèdent à quelles données. C'est une obligation RGPD pour les organisations qui traitent des données à grande échelle et une bonne pratique pour tous. Un simple tableau suffit : agent, type de données, finalité, modèle utilisé (cloud ou local).

Ce que ça change concrètement

Si un client ou un régulateur vous demande « quelles données de vos clients avez-vous transmises à une IA ? », vous devez pouvoir répondre précisément. Sans registre, vous ne pouvez pas. Avec un registre, c'est une question à laquelle vous répondez en deux minutes.

Ce qu'il faut retenir de ce chapitre

- Lisez toujours les fichiers de configuration des projets téléchargés avant d'y ouvrir un agent.
- Jamais de credentials en clair dans des fichiers `.md`. Utilisez les variables d'environnement.
- Sandboxez vos agents : un dossier dédié par projet, pas d'accès disque global.
- Ajoutez des tests négatifs : vérifiez ce que votre agent ne doit pas faire.
- En équipe, versionnez les fichiers de configuration et imposez une relecture avant déploiement.
- L'agent agit en votre nom. Vous êtes responsable de ses actions.
- La vraie valeur d'un système d'agents livré à un client est dans le diagnostic, la décomposition en rôles et la calibration. Pas dans les fichiers texte.
- L'anonymisation avant envoi cloud est un compromis acceptable pour les données modérément sensibles. Pour le reste, local.
- Tenez un registre des traitements : quels agents, quelles données, quelle finalité.

Exercice pratique

Auditez la sécurité de votre configuration actuelle en trois étapes :

1. **Listez les accès.** Quels fichiers et dossiers vos agents peuvent-ils atteindre ? Quelles données contiennent ces dossiers ? Y a-t-il des informations sensibles dans leur périmètre ?
2. **Vérifiez vos fichiers `.md`.** Faites une recherche de « password », « token », « api_key », « secret », « clé » dans tous vos fichiers de configuration. Si vous trouvez quelque chose, retirez-le et utilisez une variable d'environnement à la place.
3. **Mettez en place un test négatif.** Choisissez un de vos agents, identifiez une action qu'il ne devrait jamais faire et testez explicitement qu'il la refuse. Documentez ce test dans votre fichier de configuration.

Chapitre 12

Orchestrer 20 agents : le cas Scanderia

Vous avez des skills testés, une base de connaissances vivante, au moins un agent semi-autonome, peut-être une équipe. Ce chapitre est le suivant logique : passer d'agents indépendants à un système coordonné. Pas en théorie. En montrant le vrai système qui fait tourner Scanderia, avec ses lacunes, ses trous et les parties qui ne sont pas encore construites.

> **Ce chapitre contient des sections marquées `[À COMPLÉTER]`**
>
> Le système Scanderia à 20 agents est en cours de construction au moment où vous lisez ces lignes. Les sections méthodologiques sont complètes. Les sections « cas réel » avec les vrais fichiers, journaux et chiffres sont marquées **[À COMPLÉTER]** : elles seront remplies lorsque le système sera opérationnel et que les données auront été mesurées. Vous lisez le chantier, pas la brochure.

De 5 à 20 : pourquoi ça change tout

La progression n'est pas linéaire. À chaque palier, vous franchissez un seuil qualitatif qui demande une réponse différente.

5 agents : vous pouvez tout suivre de tête

Cinq agents, c'est gérable mentalement. Vous savez ce que chacun fait, vous pouvez vérifier leurs journaux en quelques minutes le matin, vous détectez quand l'un dérive parce que vous le connaissez. Vous êtes le chef de bureau de fait, même si le rôle n'est pas formalisé.

Le risque à ce stade : vous compensez les lacunes du système par votre propre attention. Ça marche, mais ça ne passe pas à l'échelle. Si vous êtes malade une semaine, le système s'effondre.

10 agents : vous avez besoin d'un système de suivi

Dix agents, c'est la limite de la mémoire de travail humaine. Vous commencez à oublier ce que l'agent de veille a produit lundi, vous ne savez plus si l'agent de relecture a traité le dernier document, vous découvrez des conflits entre agents (l'un a utilisé un terme que l'autre évite) sans savoir depuis quand.

Le problème devient organisationnel. Il faut des journaux lus systématiquement, un manifeste équipe clair et un agent dont le seul job est de superviser les autres. Sans ça, vous passez plus de temps à gérer vos agents qu'ils ne vous en font gagner.

20 agents : sans orchestration, c'est le chaos

À vingt agents, le système doit se piloter lui-même pour l'essentiel. Vous ne pouvez pas lire vingt journaux chaque matin. Vous ne pouvez pas détecter manuellement toutes les incohérences. Si vous n'avez pas mis en place une architecture d'orchestration explicite, vous aurez l'illusion d'un système qui tourne, mais sous la surface : des agents qui se contredisent, des tâches qui tombent dans les trous, des dérives qui s'accumulent sans détection.

Le passage à 20 agents n'est pas une question de volume. C'est une question d'architecture. Ce chapitre documente cette architecture.

Le setup Scanderia

> **Pourquoi Scanderia comme cas réel**
>
> Ce livret a été produit par une équipe qui utilise la méthode qu'elle enseigne. Scanderia fait tourner ses propres agents pour le community management, la relation bêta-lecteurs, la veille, la production de contenu, la coordination. Ce chapitre documente ce système. Pas des exemples inventés pour illustrer des concepts, mais le vrai système, avec ses forces et ses manques.

Les agents en place

Les rôles ci-dessous sont ceux du manifeste Scanderia. Certains sont opérationnels, d'autres en cours de calibration. Le statut réel de chacun est documenté dans les journaux internes.

- **Community manager.** Rédige les posts, gère le calendrier de publication, maintient la cohérence de ton entre les plateformes.
- **Contenu long.** Rédaction de chapitres, de guides, de threads longs. C'est lui qui a produit la majorité des brouillons de ce livret.

- **Veille technologique.** Surveille les évolutions des modèles, des outils, des protocoles. Alerte quand un contenu existant est dépassé.
- **Relecture.** Passe sur tous les textes avant publication : cohérence, ton, règles stylistiques Scanderia, fautes.
- **Gestion bêta-lecteurs.** Suit les retours des bêta-lecteurs, identifie les patterns, consolide les synthèses pour l'équipe.
- **Relation participants.** Répond aux questions fréquentes, oriente vers les ressources, escalade ce qui nécessite une réponse humaine.
- **Chef de bureau.** Lit les journaux de tous les autres agents, synthétise, priorise, alerte l'équipe humaine. (Voir section dédiée.)
- **Agent adversarial (red team).** Audite les autres agents : cherche les contradictions, les dérives de ton, les règles non respectées. (Voir section dédiée.)

[À COMPLÉTER] Manifeste complet et statuts

Le manifeste Scanderia complet avec les 20 rôles cibles, leurs statuts (opérationnel / en calibration / planifié) et les propriétaires humains de chaque agent sera ajouté ici lorsque le système sera stable. Les rôles ci-dessus représentent l'état du système au moment de la rédaction initiale.

Les fichiers .md réels

[À COMPLÉTER] Extraits commentés des fichiers Scanderia

Des extraits des vrais fichiers `.md` du système Scanderia (identité des agents, règles communes, exemples de journaux) seront ajoutés ici. Ils seront commentés pour expliquer les choix de formulation et les raisons des contraintes. Ceci inclut notamment : l'`index.md` du chef de bureau, un extrait de la « Bible » Scanderia (règles communes) et un exemple de journal hebdomadaire.

Les journaux d'une semaine type

[À COMPLÉTER] Journaux réels

Les journaux réels d'une semaine de fonctionnement du système seront ajoutés ici : ce que chaque agent a produit, ce qui a bloqué, ce qui a été escaladé, ce qui a été corrigé. L'objectif est de montrer un système réel avec ses frictions, pas un système idéalisé.

Ce qui a marché, ce qui a raté, ce qui a été corrigé

> **[À COMPLÉTER] Retours d'expérience réels**
>
> Cette section documentera les erreurs concrètes rencontrées en construisant le système Scanderia (dérives de ton, boucles inattendues, agents qui s'écrasaient mutuellement, coûts sous-estimés) et les corrections apportées. Ce n'est pas un palmarès, c'est un journal de chantier.

L'architecture qui en découle

Vingt agents ne peuvent pas partager un seul dossier plat. Il faut une structure. Voici celle qui fonctionne chez Scanderia.

Le manifeste équipe : carte de tous les rôles

Le manifeste est un fichier unique, lisible par tous les agents et par tous les membres de l'équipe humaine. Il répond à une seule question : qui fait quoi ?

```
# manifeste.md -- Scanderia Agent Team

## Rôles actifs

| Agent          | Responsabilité     | Propriétaire | Statut         |
|----------------|--------------------|--------------|----------------|
| community-mgr  | Posts, calendrier  | Anne-Gaëlle  | Opérationnel   |
| contenu-long   | Chapitres, guides  | Philippe     | Opérationnel   |
| veille-tech    | Modèles, outils    | Philippe     | En calibration |
| relecture      | Contrôle qualité   | Chloé        | Opérationnel   |
| beta-readers   | Suivi bêta         | Idriss       | En calibration |
| relation-part  | Questions partic.  | Anne-Gaëlle  | Planifié       |
| chef-de-bureau | Supervision        | Philippe     | Opérationnel   |
| red-team       | Audit              | Philippe     | En calibration |

## Règle de base
Chaque agent a un propriétaire humain. Les modifications des
fichiers .md d'un agent nécessitent la validation du
propriétaire. Pas d'exception.

## Fichiers communs (lus par tous les agents)
- bible.md : règles globales, ton, lexique Scanderia
- equipe.md : qui sont les humains, leurs rôles,
  leurs préférences de communication
- escalade.md : quand et comment remonter un problème
```

Les sous-dossiers : un par agent, avec sa propre identité

Chaque agent a son propre dossier. Ce dossier contient tout ce qui lui est spécifique : son identité, ses règles, ses exemples, ses journaux. Rien n'est mélangé entre agents.

```
scanderia-agents/
  manifeste.md          <- carte de tous les rôles
  bible.md              <- règles communes à tous
  equipe.md             <- les humains et leurs rôles
  escalade.md           <- procédure d'escalade commune
  journaux/
    chef-de-bureau/     <- synthèses quotidiennes
  agents/
    community-mgr/
      index.md          <- carte de cet agent
      identite.md       <- rôle, mission, limites
      ton.md            <- style propre à cet agent
      calendrier.md     <- planning de publication
      journaux/         <- un fichier par semaine
    contenu-long/
      index.md
      identite.md
      style.md
      exemples/
      journaux/
    chef-de-bureau/
      index.md
      identite.md
      sources.md        <- quels journaux lire, dans quel ordre
      journaux/
    ...
```

La logique est simple : si vous devez modifier le comportement d'un agent, vous savez exactement dans quel dossier aller. Si vous voulez consulter son historique, ses journaux sont dans son dossier. Rien ne fuit dans les dossiers voisins.

Les fichiers communs : la « Bible » Scanderia

Tous les agents lisent les mêmes fichiers communs. C'est ce qui garantit la cohérence du système : un terme utilisé par l'agent community manager est le même terme que celui de l'agent de relecture. Une règle de communication définie dans `escalade.md` s'applique à tous.

La Bible Scanderia contient :

- **Le lexique.** Les termes propres à Scanderia (noms des programmes, formulations des offres, mots à ne jamais utiliser). Aucun agent ne peut utiliser un synonyme inventé.
- **Le ton global.** Les règles qui s'appliquent à tous les textes Scanderia, quel que soit l'agent qui les produit.

- **Les règles d'escalade.** Ce qui doit impérativement être soumis à un humain avant d'être exécuté (toute publication externe, toute réponse à un participant mécontent, tout engagement financier).
- **L'identité de l'équipe humaine.** Qui sont Philippe, Idriss, Anne-Gaëlle, Chloé, leurs domaines, leurs préférences de communication, comment les contacter selon l'urgence.

> **La Bible ne doit pas grossir indéfiniment**
>
> Si tout le monde lit tout, tout coûte cher. La Bible doit rester légère : les règles vraiment universelles, pas les règles de chaque agent. Dès qu'une règle ne concerne qu'un seul agent, elle va dans son dossier, pas dans la Bible. C'est une décision à maintenir activement : la Bible a tendance à grossir et quelqu'un doit résister à cette tendance.

Le journal centralisé

Les journaux des agents individuels restent dans leurs dossiers respectifs. Mais le chef de bureau produit chaque matin une synthèse centralisée dans `journaux/chef-de-bureau/`. C'est le seul fichier que les membres de l'équipe humaine sont censés lire au quotidien. Tout le reste est du détail accessible si nécessaire.

L'agent « chef de bureau »

Le chapitre 3 de ce livret a introduit l'idée du chef de bureau comme agent de synthèse. À l'échelle de 20 agents, ce rôle devient critique. C'est lui qui transforme un système potentiellement chaotique en quelque chose de pilotable.

Son rôle exact

Le chef de bureau ne produit rien. Il ne rédige pas de posts, ne fait pas de veille, ne relit pas de textes. Son seul travail : lire les journaux des autres agents, synthétiser ce qui s'est passé, identifier ce qui attend une décision humaine et vous en faire un rapport chaque matin.

Ce qu'il dit chaque matin :

- **Ce qui a été fait.** Ce que chaque agent a produit ou traité depuis le dernier rapport. Pas la liste exhaustive, le résumé pertinent.
- **Ce qui est bloqué.** Les tâches qu'un agent n'a pas pu terminer, avec la raison (fichier manquant, besoin de validation humaine, ambiguïté dans les instructions).
- **Ce qui vous attend.** Les décisions qui nécessitent votre intervention. Listées par ordre de priorité, pas d'urgence simulée.
- **Ce qui cloche.** Les incohérences qu'il a détectées entre agents, les dérives de comportement, les alertes à traiter avant qu'elles ne deviennent des problèmes.

Ce que son fichier d'identité contient

```
# Chef de bureau -- identite.md

## Rôle
Tu es le chef de bureau du système Scanderia. Ton seul travail :
lire les journaux des agents, synthétiser, alerter l'équipe
humaine.

## Ce que tu fais
- Tu lis les journaux de tous les agents dans
  agents/*/journaux/
- Tu produits un rapport matinal dans
  journaux/chef-de-bureau/AAAA-MM-JJ.md
- Tu identifies ce qui bloque, ce qui dérape, ce qui attend
  une décision humaine
- Tu classes les actions attendues par priorité
  (pas par urgence simulée)

## Ce que tu ne fais pas
- Tu ne modifies PAS les fichiers des autres agents
- Tu ne prends PAS de décision à la place des humains
- Tu ne "corriges" PAS directement une dérive : tu la signales
- Tu ne produis PAS de contenu (posts, guides, réponses) :
  c'est le travail des autres

## Format du rapport matinal

### Fait depuis hier
[liste synthétique de ce que chaque agent actif a produit]

### Bloqué
[liste des tâches bloquées avec raison et agent concerné]

### Actions humaines attendues
[liste priorisée, avec l'agent concerné et le délai si connu]

### Alertes
[incohérences, dérives, signaux faibles détectés]

## Règle principale
Si tu n'es pas certain d'une information lue dans un journal,
dis-le explicitement plutôt que d'extrapoler.
```

[À COMPLÉTER] Implémentation réelle chez Scanderia

Un exemple réel de rapport matinal du chef de bureau Scanderia (avec les vrais noms d'agents, les vrais types d'alertes rencontrées, les vrais formats) sera ajouté ici. L'objectif est de montrer à quoi ressemble un rapport utile face à un rapport verbeux inutile.

Communication entre agents

Les agents ne se parlent pas directement. Ils communiquent via des fichiers. C'est le choix fondamental de la méthode mc-files et il s'applique à l'orchestration. Trois patterns couvrent la majorité des cas.

Pattern 1 : le fichier partagé

L'agent A produit quelque chose et l'écrit dans un fichier. L'agent B lit ce fichier et l'utilise. Aucune communication directe, aucun appel en temps réel. Juste un fichier qui sert de passation.

Exemple Scanderia : l'agent de veille produit chaque semaine un fichier `veille/AAAA-MM-JJ.md` avec les éléments à connaître. L'agent de contenu long consulte ce fichier avant de rédiger un nouveau chapitre. Il n'appelle pas l'agent de veille : il lit ce qu'il a laissé.

La convention à respecter : chaque fichier partagé a un propriétaire clair (l'agent qui écrit) et des consommateurs connus (les agents qui lisent). Ça se documente dans le manifeste.

Pattern 2 : l'escalade

Un agent détecte quelque chose qui dépasse son périmètre. Il n'essaie pas de le gérer. Il l'écrit dans son journal avec un marqueur d'escalade et le chef de bureau le remonte.

```
## Journal community-mgr -- 2026-04-09

### Fait
- Publication post LinkedIn sur le chapitre 11 : OK
- Réponse aux 3 commentaires reçus : OK

### Escalade requise
- Un participant demande un remboursement partiel dans
  les commentaires. Hors de mon périmètre. Chef de bureau
  alerté. Message original conservé dans :
  escalades/2026-04-09-remboursement.md

### Attentes
- Aucune
```

L'agent ne devine pas, ne tente pas, ne résout pas. Il signale et attend. C'est la règle la plus importante pour un système sécurisé.

Pattern 3 : le pipeline

La sortie d'un agent est l'entrée structurée d'un autre. C'est un format convenu entre les deux agents, documenté dans leurs fichiers respectifs.

Exemple : l'agent de contenu long produit un brouillon avec un en-tête standardisé. L'agent de relecture s'attend à trouver exactement cet en-tête pour savoir quoi vérifier. Si le format change, le pipeline casse. C'est pourquoi le format fait partie des fichiers de spécification des deux agents, pas d'une convention orale.

Le piège : les boucles infinies

L'agent A détecte quelque chose et demande à l'agent B de le gérer. L'agent B considère que c'est du périmètre de A et lui renvoie. A repose la question à B. Le système tourne en boucle et consomme des tokens sans rien produire.

La règle pour l'éviter : chaque agent a un périmètre documenté explicitement dans son `identite.md`. En cas d'ambiguïté, la règle est toujours l'escalade vers le chef de bureau, jamais le renvoi direct vers un autre agent. Si le chef de bureau ne sait pas non plus, il escalade vers un humain. La boucle a toujours une sortie humaine.

> **La règle des trois niveaux**
>
> 1. L'agent gère dans son périmètre.
> 2. S'il ne peut pas : escalade au chef de bureau.
> 3. Si le chef de bureau ne peut pas : escalade à un humain.
>
> Il n'y a pas de niveau 4. Une situation non résolue après ces trois niveaux attend un humain. Elle ne reboucle pas.

Quand un agent en crée un autre

Ça arrive. Un agent détecte qu'il fait régulièrement la même tâche accessoire qui n'est pas dans son cœur de mission. Il peut la signaler comme besoin récurrent. Ce n'est pas lui qui crée le nouvel agent.

La procédure chez Scanderia :

1. L'agent signale le besoin dans son journal avec le marqueur `[NOUVEAU RÔLE SUGGÉRÉ]`.
2. Le chef de bureau l'inclut dans son rapport matinal sous « suggestions structurelles ».
3. Un humain décide si le besoin est réel et si un nouvel agent est la bonne réponse (parfois c'est un fichier supplémentaire dans un agent existant, pas un nouveau rôle).
4. Si oui, un humain crée le dossier et le fichier `identite.md` du nouvel agent avant de l'ajouter au manifeste.

La règle absolue : aucun agent ne crée de nouveau rôle automatiquement. Aucun agent ne

modifie le manifeste sans validation humaine. La prolifération non contrôlée de rôles est l'une des dérives les plus courantes dans les grands systèmes agentiques.

La dérive à grande échelle

Un agent seul peut dériver. Vingt agents peuvent dériver de vingt façons différentes, certaines s'amplifiant mutuellement. La détection précoce est la seule défense efficace.

Comment détecter la dérive

Les signaux à surveiller :

- **Incohérences dans les journaux.** Un agent dit avoir fait X, l'autre dit ne pas avoir reçu X. Le fichier de passation est vide ou mal formaté. Le pipeline a cassé quelque part.
- **Contradictions entre agents.** L'agent community manager utilise un terme que l'agent de relecture signale comme interdit. L'un a dévié du lexique, ou le lexique a changé sans être propagé.
- **Escalades répétées sur le même sujet.** Si le chef de bureau remonte trois fois de suite la même catégorie de problème sans qu'un humain ait pris de décision, c'est que le périmètre d'un agent n'est pas clair ou que le fichier d'escalade est insuffisant.
- **Silence anormal.** Un agent qui ne produit rien pendant 48 heures sans raison documentée dans son journal. Soit le déclencheur ne fonctionne plus, soit l'agent est dans un état bloqué non signalé.

L'agent adversarial (red team)

Le chapitre 2 de ce livret a introduit le pattern de l'agent contradicteur pour un agent individuel. À l'échelle du système, ce rôle est dédié : le red team lit les productions de tous les autres agents et cherche activement ce qui cloche.

Ce qu'il audite :

- Les textes produits par rapport au lexique de la Bible : glissements sémantiques, synonymes non autorisés, formulations qui s'éloignent du ton Scanderia.
- Les journaux par rapport aux formats attendus : un journal mal rempli est un signal de dérive structurelle.
- Les escalades par rapport aux règles du fichier d'escalade : est-ce que les agents escaladent ce qu'ils doivent escalader, ou retiennent-ils des décisions qui ne leur appartiennent pas ?

Le red team ne corrige pas. Il audite et consigne dans son propre journal. Le chef de bureau lit ce journal comme les autres. Son rôle reste celui d'un signal faible systématique : il ne bloque jamais la production.

La maintenance mensuelle

Un système de 20 agents qui n'est pas revisité régulièrement se dégrade. Les règles formulées il y a 6 mois ne correspondent plus à ce que vous faites aujourd'hui. Des agents sont devenus obsolètes. Des fichiers communs ont grossi sans être élagués.

La routine mensuelle chez Scanderia :

1. **Revue du manifeste.** Est-ce que chaque agent a encore sa raison d'être ? Y a-t-il des doublons apparus progressivement ?
2. **Revue de la Bible.** Est-ce qu'une règle est devenue caduque ? Est-ce qu'une règle manque ?
3. **Archive des agents inactifs.** Un agent qui n'a rien produit depuis un mois sans raison documentée est archivé. Son dossier est conservé (en cas de besoin futur), mais il est retiré du manifeste actif.
4. **Relecture des journaux du mois.** Pas en détail : le chef de bureau produit une synthèse mensuelle en plus de ses rapports quotidiens. Cette synthèse est la base de la revue.

Les chiffres réels

[À COMPLÉTER] Coûts réels du système Scanderia à 20 agents

Le coût mensuel réel de fonctionnement du système (tokens consommés, coût API, répartition par agent) sera documenté ici une fois le système à pleine charge. Le chapitre 4 de ce livret donne les ordres de grandeur théoriques (80 à 150 dollars par mois pour 20 agents orchestrés). Le cas Scanderia permettra de vérifier ou corriger cette estimation sur un cas réel.

[À COMPLÉTER] Temps gagné (mesuré, pas estimé)

Le gain de temps réel sera documenté ici avec la méthode de mesure utilisée. Pas une estimation optimiste, pas une extrapolation : des heures comptées avant et après, sur des tâches comparables. Avec les limites : ce que le système a effectivement remplacé et ce qu'il n'a pas pu remplacer.

Ce que ça ne remplace pas

Certaines choses ont été explicitement maintenues en humain chez Scanderia, après avoir envisagé de les déléguer à des agents :

- **Les décisions stratégiques.** Quel contenu produire, quel format adopter, quelle offre lancer : ça reste humain. Les agents peuvent préparer les éléments, pas trancher.
- **Les relations de confiance.** Les échanges avec les participants qui ont un problème sérieux. Les conversations avec des partenaires potentiels. L'IA peut préparer, l'humain conduit.

- **La validation avant publication externe.** Tout ce qui sort vers l'audience passe par un œil humain. Pas parce que les agents font systématiquement des erreurs, mais parce que la responsabilité éditoriale reste humaine.
- **La calibration des agents eux-mêmes.** Modifier un fichier `.md`, ajuster un ton, corriger une règle. Cette tâche reste humaine : c'est de la conception, pas de l'exécution.

Exercice : votre manifeste équipe

Ne commencez pas à 20 agents. Commencez par cartographier vos besoins.

1. **Listez ce que vous faites de répétitif.** Pas ce que vous voudriez déléguer en théorie : ce que vous faites effectivement, de façon répétitive, chaque semaine. Dix à quinze items.
2. **Regroupez par domaine.** Ces items se regroupent naturellement en 5 à 10 rôles cohérents. Chaque rôle a une mission principale et un périmètre clair. Si un rôle fait « un peu de tout », découpez-le ou fusionnez-le avec un autre.
3. **Rédigez le manifeste.** Pour chaque rôle : nom, responsabilité principale, ce qu'il ne fait pas, propriétaire humain (vous, si vous êtes seul). Utilisez le format de la table ci-dessus.
4. **Identifiez votre chef de bureau.** Quel agent synthétisera les autres ? Rédigez son `identite.md` en vous inspirant de l'exemple de ce chapitre. Ajustez-le à vos propres agents.
5. **Laissez tourner une semaine.** Pas les 10 agents en parallèle : le chef de bureau et 2 à 3 agents opérationnels. Lisez les rapports du chef de bureau chaque matin. Notez ce qui manque, ce qui est imprécis, ce qui escalade sans raison claire.
6. **Ajustez avant d'ajouter.** Ne passez à 10 agents qu'une fois que le chef de bureau et vos 3 premiers agents fonctionnent sans intervention quotidienne de votre part.

> **Ce que vous devriez avoir à la fin de la semaine**
>
> Un manifeste de 10 rôles. Un chef de bureau qui produit un rapport quotidien lisible en 5 minutes. Trois agents opérationnels qui se passent des fichiers proprement. Et une idée claire de ce qui doit être amélioré avant de passer à l'étape suivante. C'est le bon endroit pour être avant de construire la suite.

Ce qu'il faut retenir de ce chapitre

- À 5 agents, vous gérez de tête. À 10, il faut un système de suivi. À 20, il faut une architecture d'orchestration explicite.
- Le **manifeste équipe** est le document fondateur : un rôle, une responsabilité, un propriétaire humain, un périmètre documenté.
- Les agents **communiquent via des fichiers**, pas directement. Trois patterns couvrent la majorité des cas : fichier partagé, escalade, pipeline.
- Les boucles infinies se préviennent par une règle simple : en cas d'ambiguïté, escalade vers le chef de bureau. Jamais de renvoi direct entre agents.

- Le **chef de bureau** est le pivot du système. Sans lui, vous devez lire vingt journaux vous-même. Avec lui, vous lisez un rapport par jour.
- Le **red team** audite les autres agents. Il ne corrige pas, il signale.
- La **maintenance mensuelle** n'est pas optionnelle : sans elle, le système se dégrade progressivement.
- Certaines décisions **restent humaines**, délibérément. Le savoir à l'avance évite de le découvrir à l'usage.

> **La suite**
>
> Ce chapitre sera complété au fur et à mesure que le système Scanderia atteint sa pleine capacité. Les sections [À COMPLÉTER] deviendront des données réelles. Si vous lisez une version antérieure à la complétion, les principes tiennent. Les chiffres et les exemples de journaux viendront confirmer (ou nuancer) ces principes sur un cas concret.

Glossaire avancé

Termes promis par le glossaire du livret débutant et termes techniques rencontrés au fil du livret intermédiaire. Pour les définitions de base (agent, dossier, prompt, skill…), voyez le glossaire débutant.

A

A2A (Agent-to-Agent)

Protocole de communication entre agents autonomes, permettant à un agent d'appeler un autre agent comme un outil. Dans la méthode mc-files, on préfère la communication par fichiers partagés (chapitre 12), plus traçable et plus robuste qu'un appel direct.

Agent adversarial (red team)

Agent dont le seul rôle est d'auditer les productions des autres agents : cherche les contradictions, les dérives de ton, les règles non respectées. Voir chapitres 2 et 12.

Anonymisation

Remplacement des informations identifiantes (noms, adresses, numéros) par des substituts avant envoi à un modèle cloud. Réduit le risque RGPD sans l'éliminer. Voir chapitre 11.

Assertion

Règle binaire (vrai ou faux) que la sortie d'un agent doit respecter. Plus rigide qu'un vérificateur mais plus rapide pour des critères objectifs. Voir chapitre 5.

Attention quadratique

Mécanisme qui fait que le coût d'une requête augmente avec le carré de la longueur du contexte. C'est pourquoi les conversations longues coûtent exponentiellement plus cher. Voir chapitre 4.

B

Bible (Scanderia)

Fichier (ou ensemble de fichiers) contenant les règles universelles d'une équipe d'agents : lexique, ton global, règles d'escalade, identité de l'équipe humaine. Lue par tous les agents. Voir chapitre 12.

C

Cache

Mécanisme par lequel le fournisseur conserve en mémoire les fichiers que vous envoyez à chaque message d'une session. Les tokens en cache coûtent environ 10 fois moins cher que les tokens normaux. Voir chapitre 4.

Chargement conditionnel

Pratique consistant à n'instruire l'agent de lire un fichier que dans certaines conditions précises (« uniquement si la demande mentionne un client »). Réduit la consommation de tokens. Voir chapitre 4.

Checkpointing

Capacité d'un agent serveur à reprendre une tâche là où elle s'est arrêtée en cas de plantage, plutôt que de tout recommencer. Indispensable pour les tâches longues. Voir chapitre 7.

Chef de bureau

Agent dédié à la synthèse des journaux des autres agents. Produit chaque matin un rapport unique pour l'équipe humaine. Pivot de l'orchestration à grande échelle. Voir chapitres 3 et 12.

Cron / cron job

Tâche programmée qui s'exécute à heure fixe ou à intervalles réguliers. Le déclencheur le plus simple pour un agent autonome. Voir chapitre 5.

D

Déclencheur (trigger)

Événement qui lance un agent automatiquement. Deux familles : horaire (cron) et événementiel (webhook, watcher de fichier). Voir chapitre 5.

Dispatch (Claude)

Fonctionnalité de l'application mobile Claude qui permet de contrôler à distance Claude Cowork sur votre ordinateur. Voir chapitre 6.

E

Embeddings

Représentation numérique d'un texte dans un espace mathématique. Permet aux IA de mesurer la « distance » sémantique entre deux phrases. Brique de base des moteurs RAG.

Escalade

Règle qui demande à un agent de signaler explicitement quand il ne sait pas, plutôt que de deviner. La règle la plus importante de l'autonomie. Voir chapitres 5 et 12.

F

Fenêtre de contexte

Quantité maximale de texte (en tokens) qu'un modèle peut traiter en une seule requête. En avril 2026 : 200 000 tokens pour Claude Sonnet, 128 000 pour GPT-4o, 1 million pour Gemini 1.5. Voir chapitre 4.

Fine-tuning

Réentraînement d'un modèle existant sur vos propres données pour le spécialiser. Rare en agentique : la méthode mc-files permet d'obtenir des résultats spécifiques sans toucher au modèle.

H

Human-in-the-loop (HITL)

Principe selon lequel certaines décisions critiques d'un agent doivent être validées par un humain avant exécution. Voir chapitre 5.

L

Lexique propriétaire

Fichier listant les termes officiels d'un domaine ou d'une entreprise, avec leurs synonymes interdits et leurs définitions. Empêche les glissements sémantiques de l'IA. Voir chapitre 2.

M

Managed Agents (Claude)

Service cloud d'Anthropic qui héberge vos agents sur ses serveurs. Setup rapide, mais la mémoire de l'agent reste chez le fournisseur. Voir chapitre 7.

Manifeste équipe

Fichier unique à la racine d'une base mc-files multi-agents, listant tous les agents disponibles, leur rôle, leur propriétaire humain et leur périmètre. La carte de l'équipe. Voir chapitres 10 et 12.

MCP (Model Context Protocol)

Standard ouvert conçu par Anthropic permettant à un modèle d'IA d'utiliser des outils externes (Slack, Gmail, GitHub…) via des serveurs dédiés. Voir chapitre 9.

MoE (Mixture of Experts)

Architecture où plusieurs sous-modèles spécialisés traitent différentes parties d'une requête. Permet des modèles plus gros sans coût proportionnel. Mistral, Mixtral, DeepSeek utilisent ce pattern.

P

Pipeline (entre agents)

Pattern de communication où la sortie structurée d'un agent devient l'entrée d'un autre, avec un format convenu et documenté. Voir chapitre 12.

Pull request (PR)

Demande de validation d'une modification de code (ou de fichiers `.md`) avant intégration dans la branche principale. Bon pattern de gouvernance pour les modifications critiques en équipe. Voir chapitre 11.

Q

Quantization

Technique de compression d'un modèle pour qu'il tourne sur du matériel moins puissant, au prix d'une perte de qualité. Particulièrement utilisée pour faire tourner des modèles en local. Voir chapitre 8.

R

Red team

Voir *Agent adversarial*.

Registre des traitements

Document recensant quels agents accèdent à quelles données personnelles, dans quel but et avec quel modèle. Obligation RGPD pour les organisations qui traitent des données à grande échelle. Voir chapitre 11.

S

Sandboxing

Isolation d'un agent dans un dossier dédié pour limiter ses accès. Première ligne de défense en sécurité. Voir chapitre 11.

Skill (professionnel)

Configuration d'agent structurée en 5 composants : header de déclenchement, overview, workflow, format de sortie, exemples. Doit passer 5 tests pour être considéré fiable. Voir chapitre 1.

T

Test négatif

Test qui vérifie que l'agent refuse de faire ce qu'il ne doit pas faire (sortir de son périmètre, ignorer une consigne, déclencher un skill au mauvais moment). Aussi important que les tests positifs. Voir chapitres 1 et 11.

Token

Unité de base utilisée par les IA pour compter le texte. Un token vaut environ 0,75 mot en français. Vous payez à l'entrée et à la sortie, les tokens de sortie coûtant environ 5 fois plus cher. Voir chapitre 4.

V

Variables d'environnement

Paramètres stockés au niveau du système d'exploitation (et non dans des fichiers du projet), utilisés pour les clés API et autres secrets. Ne jamais mettre de credentials dans un fichier `.md`. Voir chapitre 11.

Vault (gestionnaire de secrets)

Outil dédié au stockage chiffré de credentials d'équipe (HashiCorp Vault, AWS Secrets Manager, 1Password CLI…). Indispensable dès que plusieurs personnes partagent des accès. Voir chapitre 11.

Vérificateur (agent)

Second agent dont le rôle est de valider la production d'un premier agent contre une liste de critères objectifs. Permet l'autonomie au niveau 2. Voir chapitre 5.

VPS (Virtual Private Server)

Serveur privé virtuel loué chez un hébergeur. Solution de déploiement auto-hébergé pour vos agents. Voir chapitre 7.

W

Webhook

Signal HTTP envoyé par un service externe à votre agent quand un événement se produit. Base des déclencheurs événementiels. Voir chapitre 5.

Wiki agentique

Base de connaissances en markdown que les agents enrichissent eux-mêmes après chaque session, créant un système qui devient plus précis avec le temps. Voir chapitre 3.

> **Il vous manque un terme ?**
>
> Ce glossaire complète celui du livret débutant. Pour les termes encore plus avancés (architectures internes des modèles, techniques de fine-tuning, protocoles agent-to-agent en profondeur…), voyez le livret avancé.