SKILLS · CONNECTEURS · MCP · AGENTS · TOKENS

Le Guide Claude 2026

Exploiter toute la puissance de Claude — Fable 5, automatisations, agents, stacks par métier — et garder la main sur la facture quand tout passe au compteur.

Claude Fable 5 | Skills | Connecteurs | MCP | GitHub | Agents IA | Automatisations | Stacks par métier | Facturation à l'usage

Édité par Otium
ATELIER IA SUR-MESURE · OTIUM-AI.CO

ÉDITION JUIN 2026

AVANT-PROPOS

Pourquoi ce guide, pourquoi maintenant

> **MISE À JOUR — 15 JUIN 2026**
>
> **Fable 5 et Mythos 5 ont été suspendus le 12 juin 2026**, sur ordre de contrôle à l'export du gouvernement américain — partout, France comprise. Anthropic a coupé l'accès immédiatement. **Opus 4.8 et les modèles antérieurs restent pleinement disponibles.**
>
> Ce guide reste valable de bout en bout : tout ce qui suit (Skills, connecteurs, MCP, agents, automatisations, stacks, optimisation des coûts) fonctionne à l'identique sur Opus 4.8 et les autres modèles. Mieux : l'épisode confirme la leçon centrale du guide — ne jamais dépendre d'un seul modèle, et savoir router vers un repli (Opus 4.8, un modèle européen, un modèle ouvert hébergé en interne). Les passages ci-dessous qui décrivaient Fable 5 sont conservés à titre de contexte ; lisez-les en gardant cette suspension en tête.

Le 9 juin 2026, Anthropic mettait en ligne Claude Fable 5 — le modèle qu'il jugeait, deux mois plus tôt, trop puissant pour être commercialisé. Trois jours plus tard, l'État américain le suspendait. Entre les deux, une leçon qui survit à l'épisode : l'accès à un modèle de frontière n'est ni acquis ni stable, et la maîtrise de l'IA ne se joue plus sur le choix d'un modèle, mais sur la façon de l'équiper et de pouvoir en changer.

Car le fond ne bouge pas : les outils d'IA quittent le forfait prévisible pour le compteur qui tourne, et la dépendance à un fournisseur unique devient un risque opérationnel. Savoir *utiliser* Claude ne suffit plus — il faut savoir l'équiper, le connecter, l'automatiser, le faire au bon prix, et garder un repli.

C'est l'objet de ce guide : rassembler en un seul document le fonctionnement réel de l'écosystème Claude en juin 2026 — ses briques, ses outils, ses méthodes — pour construire des systèmes qui tiennent dans la durée, techniquement et budgétairement.

À qui s'adresse ce guide

À tout le monde, et c'est volontaire. Chaque chapitre se lit à deux niveaux : le texte principal est accessible à un dirigeant qui n'écrit pas une ligne de code ; les encadrés techniques et les extraits de commande s'adressent à ceux qui mettent les mains dans le moteur.

Comment il est construit

Partie 1 — Le moment Claude. Fable 5, Mythos, et les différents visages de Claude.

Partie 2 — Le socle. Skills, connecteurs, MCP, intégration GitHub.

Partie 3 — Construire. Automatisations, agents, écosystème d'outils, stacks par profil, méthode de déploiement.

Partie 4 — Maîtriser la facture. Optimisation des tokens, anticipation de la facturation à l'usage.

Sommaire

01

Le moment Claude

Un modèle jugé trop dangereux arrive sur le marché, une gamme entière se réorganise au-dessus de lui, et la manière même d'accéder à l'intelligence change de règles. Avant d'outiller quoi que ce soit, il faut comprendre le paysage.

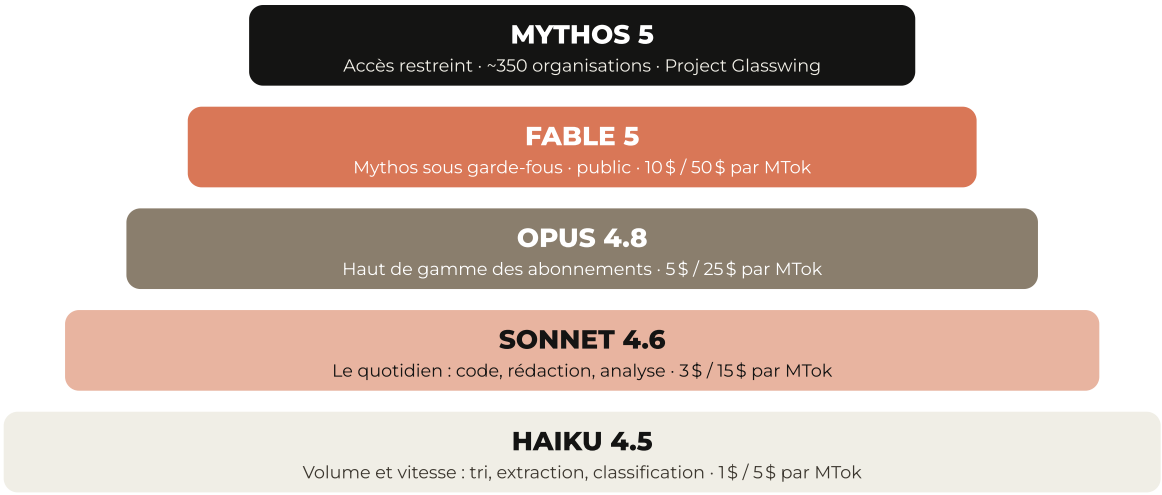
01

Claude en juin 2026 : ce qui vient de changer

Le 9 juin 2026, Anthropic annonçait deux modèles d'un coup : Claude Mythos 5, sa frontière interne réservée à quelques centaines d'organisations ; et Claude Fable 5, le même modèle sous garde-fous, ouvert à tous. Trois jours plus tard, le 12 juin, les deux étaient suspendus sur ordre de l'État américain (voir l'encart en ouverture). Cette section décrit cette gamme telle qu'elle a été présentée : elle reste la clé pour comprendre l'architecture — et ce à quoi vous reviendrez le jour où ces modèles seront rétablis. Au quotidien, aujourd'hui, votre modèle de tête est **Opus 4.8**, resté disponible.

Une gamme à cinq étages

Le catalogue Anthropic compte désormais cinq familles de modèles, du plus léger au plus puissant : **Haiku** (rapide et économique), **Sonnet** (le cheval de trait du quotidien), **Opus** (le haut de gamme historique, désormais en version 4.8), **Fable** (la classe Mythos sous garde-fous) et **Mythos** (la frontière intégrale, non commercialisée). Une classe entière vient donc de s'ajouter au-dessus de ce qui était, jusqu'en juin, le sommet de la gamme.



LA GAMME CLAUDE EN JUIN 2026 — CINQ FAMILLES, CINQ USAGES, CINQ PRIX

Fable 5 : Mythos sous garde-fous

La différence entre Fable 5 et Mythos 5 ne tient pas au modèle — c'est le même — mais à un dispositif de sécurité inédit. Chaque requête adressée à Fable 5 traverse trois classifieurs : cybersécurité offensive, biologie-chimie, et tentatives d'extraction des capacités du modèle. Si l'un d'eux se déclenche, la réponse n'est pas servie par Fable 5 mais par Opus 4.8 — et vous en êtes informé. Anthropic chiffre ce déclenchement à moins de 5 % des sessions : pour les 95 % restants, écrit l'entreprise, « la performance de Fable 5 est effectivement celle de Mythos 5 ».

Concrètement, pour vous : sur l'immense majorité des tâches professionnelles — rédaction, analyse, code, agents — vous travaillez avec la pleine capacité du modèle de frontière. Les premières semaines ont toutefois montré des faux positifs, notamment sur des tâches légitimes de cybersécurité défensive. Anthropic reconnaît des classifieurs « volontairement larges aujourd'hui », appelés à s'affiner.

> **CE QUE DIT LA FICHE TECHNIQUE**
>
> Fenêtre de contexte d'**un million de tokens** par défaut (de quoi charger plusieurs livres ou une base de code entière), **128 000 tokens de sortie**, raisonnement adaptatif toujours actif, et un identifiant API sans date — `claude-fable-5` — signe d'un modèle pensé pour durer. Anthropic le présente comme capable de travailler en autonomie « plus longtemps que n'importe quel modèle Claude précédent » — plusieurs jours d'affilée dans un harnais agentique.

Des records presque partout

Sur les benchmarks publics, le consensus s'est formé en quarante-huit heures : Fable 5 prend la tête de quasiment tous les classements, garde-fous actifs. Les chiffres ci-dessous proviennent de la system card officielle d'Anthropic — un document de 319 pages.

Benchmark	Fable 5	Opus 4.8	GPT-5.5	Gemini 3.1 Pro
SWE-bench Verified (code)	**95,0 %**	88,6 %	—	80,6 %
SWE-bench Pro	**80,0 %**	69,2 %	58,6 %	54,2 %
Terminal-Bench 2.1 (agents)	**84,3 %**	82,7 %	83,4 %	70,7 %
OSWorld-Verified (usage ordinateur)	**85,0 %**	83,4 %	78,7 %	76,2 %

Source : system card Claude Fable 5 & Mythos 5, 9 juin 2026. Moyenne sur 5 essais, garde-fous de production actifs.

Au-delà des benchmarks, deux démonstrations ont marqué le lancement. Ethan Mollick, professeur à Wharton, a documenté une session de **neuf heures et demie de travail autonome** sur une seule consigne, le modèle orchestrant lui-même des sous-agents pour collecter 2 200 vols et horaires de trains. Et Stripe, testeur précoce, rapporte une migration sur une base de code de 50 millions de lignes exécutée en une journée, là où une équipe entière aurait compté plus de deux mois.

> *L'an dernier, je parlais de travailler avec un sorcier ; je suis désormais plus proche d'un mécène. Je ne pilote plus : je commande.*
>
> ETHAN MOLLICK, PROFESSEUR À WHARTON, 9 JUIN 2026

Une séquence en trois actes — et sa morale

1. **9 juin — Lancement**

   Fable 5 arrive partout : claude.ai, Claude Code, API, AWS Bedrock, Vertex AI, Microsoft Foundry, inclus dans les abonnements. La version grand public d'un modèle jugé trop dangereux pour être vendu librement deux mois plus tôt.

2. **12 juin — Suspension**

   Sur ordre de contrôle à l'export du gouvernement américain (sécurité nationale), Anthropic coupe immédiatement Fable 5 et Mythos 5 — pour tous les ressortissants étrangers, donc pour la France. Trois jours d'existence commerciale.

3. **Après — Opus 4.8 prend le relais**

   Les autres modèles ne sont pas touchés. Opus 4.8 redevient le haut de gamme accessible. Toute organisation qui avait bâti sur Fable a dû router vers un repli du jour au lendemain — celles qui l'avaient prévu n'ont rien senti.

La morale tient en une phrase : **l'accès à un modèle de frontière n'est pas un acquis, c'est une autorisation révocable.** Le prix annoncé de Fable (10 $/50 $ le million de tokens, le double d'Opus 4.8) et son appétit en tokens en faisaient déjà un modèle à réserver aux tâches lourdes ; sa suspension ajoute une raison plus forte de ne jamais en dépendre seul.

CE QU'IL FAUT EN RETENIR

Que Fable soit en ligne ou suspendu, le bon réflexe est le même : ne jamais tout miser sur un modèle unique. Identifier les tâches qui justifient la pointe (travail long, migrations, analyses multi-documents), router tout le reste vers Opus 4.8, Sonnet ou Haiku, et garder un modèle de repli activable du jour au lendemain. Ce routage — et cette portabilité — sont le fil rouge de ce guide. La suspension du 12 juin n'invalide rien : elle démontre.

02

Les visages de Claude : application, terminal, bureau, API

« Claude » n'est pas un produit unique : c'est un même modèle accessible à travers cinq surfaces, chacune pensée pour un usage. Choisir la bonne surface pour le bon travail est la première optimisation — avant même de parler de tokens.

claude.ai — le poste de travail conversationnel

L'application web et mobile est la porte d'entrée. Au-delà du chat, trois fonctions en font un véritable outil professionnel. Les **Projets** regroupent conversations, fichiers et instructions permanentes autour d'un sujet : un projet « Appels d'offres » avec vos modèles de réponse et votre charte, un projet « Veille concurrentielle » avec vos grilles d'analyse. Les **Artefacts** génèrent des documents, des présentations ou de petites applications interactives directement dans la conversation. La **Recherche** approfondie compile des sources web avec citations.

C'est aussi dans claude.ai que vivent les connecteurs (chapitre 4) et les Skills (chapitre 3) : votre Gmail, votre Drive, votre Notion deviennent des sources que Claude consulte en séance.

Claude Code — l'agent dans le terminal

Claude Code est l'outil qui a changé la manière de développer. Ce n'est pas un assistant d'autocomplétion : c'est un agent qui lit votre base de code, planifie, écrit, exécute les tests, corrige, et recommence — dans votre terminal, votre éditeur (VS Code, JetBrains), sur le web ou dans l'application de bureau. Il s'appuie sur des fichiers d'instructions (`CLAUDE.md`) versionnés avec votre projet, des Skills spécialisées, des hooks d'automatisation et des sous-agents parallèles.

Point essentiel pour les non-développeurs : Claude Code n'est plus réservé au code. Trier des fichiers, produire des rapports, manipuler des données, orchestrer des outils — tout ce qui peut s'exprimer en fichiers et en commandes est à sa portée.

Claude Cowork — l'espace de travail agentique

Lancé en avant-première en janvier 2026 et disponible pour tous les abonnés payants depuis avril (macOS et Windows), Cowork transpose la logique agentique hors du terminal : vous confiez des dossiers entiers à un agent qui a accès à vos fichiers, enchaîne des tâches en plusieurs étapes, exécute des tâches planifiées récurrentes, et s'étend par plugins et connecteurs

MCP. C'est la surface la plus adaptée aux équipes métier — ce que Claude Code fait pour les développeurs, Cowork le fait pour les analystes, juristes, marketeurs — et Anthropic en décline des éditions métiers (juridique, petites entreprises, opérations marketing) depuis mai 2026.

Claude Design — les visuels sans designer

Lancé en avril 2026 par Anthropic Labs, Claude Design génère des prototypes, des maquettes, des présentations et des one-pagers à partir d'une description — puis les affine par retouches directes. Sa particularité, et ce qui en fait plus qu'un gadget : il sait **lire votre base de code et vos fichiers Figma pour en extraire votre système de design** (couleurs, typographies, composants) et l'appliquer à tout nouveau livrable. Pour un fondateur ou un chef de produit sans designer sous la main, c'est la différence entre une idée racontée et une idée montrée. Disponible en avant-première de recherche pour les abonnés Pro, Max, Team et Enterprise.

L'application de bureau et l'extension Chrome

L'application de bureau (macOS, Windows) ajoute ce que le navigateur ne permet pas : accès aux fichiers locaux, capture d'écran, serveurs MCP locaux, et le mode dictée. L'extension **Claude in Chrome**, elle, permet à Claude d'agir directement dans votre navigateur — remplir des formulaires, naviguer, extraire des données de pages ouvertes.

L'API et les plateformes cloud

Pour intégrer Claude dans vos propres produits et automatisations, l'API Anthropic est la voie directe — complétée par AWS Bedrock, Google Vertex AI et Microsoft Foundry si votre infrastructure vit déjà chez un hébergeur cloud. C'est par l'API que passent les systèmes que nous décrivons en partie 3 : agents sur mesure, automatisations n8n, traitements en volume.

Surface	Pour qui	Usage type
claude.ai	Tout le monde	Rédaction, analyse, recherche, projets d'équipe
Claude Code	Développeurs, profils techniques	Code, données, orchestration d'outils, agents
Cowork	Équipes métier	Dossiers entiers traités en autonomie
Claude Design	Produit, fondateurs, marketing	Prototypes, maquettes, présentations à votre charte
Desktop + Chrome	Utilisateurs avancés	Fichiers locaux, actions navigateur, MCP local
API / Bedrock / Vertex / Foundry	Équipes produit et IT	Intégrations, produits, traitements en volume

La mémoire : ce que Claude retient de vous

Question posée dans toutes les organisations : Claude se souvient-il d'une conversation à l'autre ? Oui — à quatre niveaux, qu'il faut connaître pour savoir où placer quelle information.

La mémoire de claude.ai. Activable dans les paramètres, elle retient d'une conversation à l'autre votre contexte de travail : votre rôle, vos projets en cours, vos préférences de format. Vous pouvez la consulter, la corriger, la désactiver — et les conversations « incognito » n'y laissent aucune trace. Sur les plans Team et Enterprise, elle reste cloisonnée par espace de projet.

Les Projets. La mémoire organisée : instructions permanentes et documents de référence attachés à un sujet. C'est le bon endroit pour ce qui doit être vrai à chaque conversation d'un même dossier — le brief client, la charte, l'historique de décisions.

CLAUDE.md et les fichiers de mémoire. Côté Claude Code, la mémoire vit dans des fichiers versionnés avec le projet : conventions, commandes, interdits (chapitre 6). L'agent peut aussi y consigner lui-même ce qu'il apprend en cours de session — un correctif validé, une préférence exprimée — pour le retrouver à la suivante.

Les Skills. La mémoire des méthodes, réutilisable partout : c'est l'objet du chapitre 3.

Règle de placement : ce qui vous concerne *vous* va dans la mémoire de claude.ai, ce qui concerne *un dossier* va dans un Projet, ce qui concerne *un processus* va dans une Skill, ce qui concerne *un dépôt de code* va dans CLAUDE.md. Une information au bon endroit est une information que vous ne réexpliquerez plus jamais.

LE RÉFLEXE À PRENDRE

Avant chaque tâche récurrente, posez la question : quelle surface ? Une analyse ponctuelle vit très bien dans claude.ai. Une tâche hebdomadaire mérite un Projet avec instructions. Une tâche quotidienne sur des fichiers mérite Cowork ou Claude Code. Une tâche permanente en volume mérite l'API. Monter en puissance dans cet ordre évite de sur-ingénierer — et de sur-payer.

02

Le socle : connecter Claude à votre réalité

Un modèle brillant qui ne connaît ni vos méthodes, ni vos outils, ni vos données reste un généraliste. Quatre briques transforment Claude en collaborateur opérationnel : les Skills lui apprennent vos métiers, les connecteurs lui ouvrent vos applications, MCP standardise le tout, et GitHub structure le travail technique.

03

Les Skills : transformer Claude en spécialiste de vos métiers

Une Skill est un dossier contenant des instructions, des modèles et éventuellement des scripts, que Claude charge automatiquement quand la tâche s'y prête. C'est la réponse à la question que toute organisation se pose : comment faire en sorte que Claude travaille *comme nous*, à chaque fois, sans tout réexpliquer ?

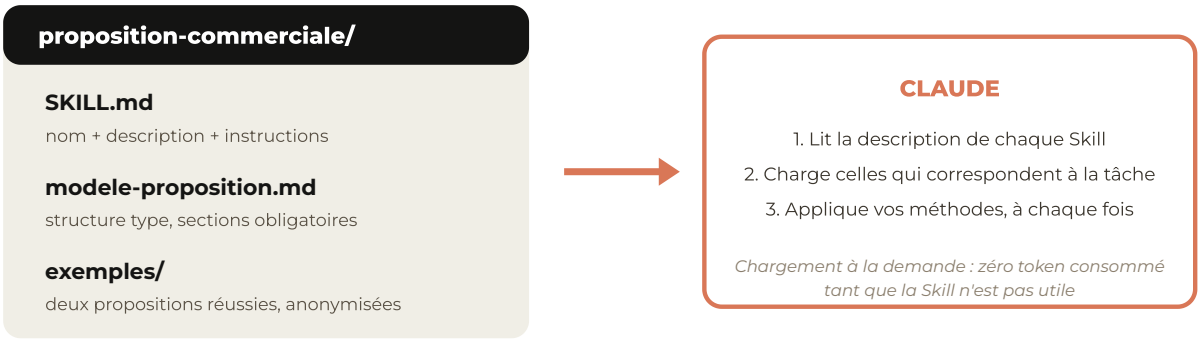
Le problème que les Skills résolvent

Sans Skills, chaque conversation repart de zéro. Vous réexpliquez votre charte éditoriale, votre format de compte rendu, vos règles de nommage, votre processus de validation. Vous copiez-collez les mêmes instructions, avec des variations, des oublis, des résultats inégaux selon les personnes. La connaissance opérationnelle reste dans les têtes — ou dans des documents que personne ne pense à joindre.

Une Skill capture cette connaissance une fois pour toutes, sous une forme que Claude consulte de lui-même. Vous ne dites plus « voici comment nous rédigeons une proposition commerciale » : Claude le sait, parce que la Skill « proposition-commerciale » existe, et il la mobilise dès que le sujet apparaît.

Anatomie d'une Skill

Techniquement, une Skill est un dossier avec un fichier `SKILL.md` à sa racine. Ce fichier commence par deux lignes essentielles — un nom et une description — suivies des instructions proprement dites. La description joue un rôle clé : c'est elle que Claude lit pour décider, en début de tâche, si la Skill est pertinente. Le dossier peut aussi contenir des fichiers de référence (modèles, exemples, grilles) et des scripts exécutables.



UNE SKILL = UN DOSSIER QUE CLAUDE DÉCOUVRE ET CHARGE SEUL, QUAND C'EST PERTINENT

```
---
name: proposition-commerciale
description: A utiliser pour rediger ou relire une proposition commerciale.
  Structure type, ton, regles de chiffrage et pieges a eviter.
---

# Proposition commerciale

## Structure obligatoire
1. Contexte du client (reformule, jamais copie-colle du brief)
2. Objectifs operationnels — trois maximum
3. Demarche proposee, par phases, avec livrables nommes
4. Calendrier realiste (prevoir les delais de validation client)
5. Conditions et perimetre — ce qui est inclus ET exclu

## Regles
- Vouvoiement. Aucun superlatif ("leader", "revolutionnaire").
- Chaque chiffre est source ou marque comme estimation.
- Le perimetre exclu est aussi important que le perimetre inclus.
```

Où vivent les Skills

Partout où Claude travaille. Dans **claude.ai**, elles s'activent depuis les paramètres de capacités, et Anthropic en fournit déjà pour les documents Office (Excel, PowerPoint, Word, PDF). Dans **Claude Code**, elles vivent dans le dossier `.claude/skills/` de votre projet — donc versionnées avec lui — ou au niveau de votre machine pour les Skills personnelles. Via l'**API**, elles équipent vos agents sur mesure. Une même Skill peut servir les trois surfaces.

Les Skills qui rapportent le plus, par fonction

Fonction	Skills à créer en premier
Direction	Format de note de décision, grille d'analyse d'opportunité, compte rendu de comité
Commercial	Proposition commerciale, relance par paliers, fiche de qualification prospect
Marketing	Charte éditoriale, formats par canal, brief créatif, check-list de publication
Finance / admin	Lecture de relevés, rapprochements, trame de reporting mensuel
Technique	Conventions de code, processus de revue, modèles d'architecture, procédures d'incident
RH	Fiches de poste, grilles d'entretien, parcours d'intégration

EXEMPLE TYPE · CABINET DE CONSEIL, 12 PERSONNES

Trois Skills, un mois pour changer les habitudes

Scénario représentatif d'un déploiement : un cabinet crée trois Skills — « proposition-commerciale », « compte-rendu-mission », « note-de-cadrage » — à partir de ses cinq meilleurs documents existants. Effort initial : une demi-journée par Skill, menée par la personne qui maîtrise le mieux chaque format.

Le gain ne vient pas seulement du temps de rédaction : il vient de l'homogénéité. Le consultant junior produit désormais un premier jet conforme aux standards de la maison, et la relecture senior se concentre sur le fond. La Skill devient aussi un outil de formation : elle documente noir sur blanc des règles qui n'existaient que dans la tête des associés.

La méthode pour créer une bonne Skill

1. **Partez de vos meilleurs livrables**

   Une Skill ne s'invente pas en salle de réunion. Prenez trois à cinq exemples réussis du document ou du processus visé, et faites-en extraire les règles implicites — par Claude lui-même, c'est l'exercice où il excelle.

2. **Soignez la description**

   C'est elle qui déclenche le chargement. Dites *quand* utiliser la Skill, pas seulement ce qu'elle contient : « À utiliser pour rédiger ou relire une proposition commerciale » fonctionne mieux qu'un titre vague.

3 **Écrivez les interdits**

Les meilleures Skills contiennent autant de « jamais » que de « toujours ». Les erreurs récurrentes de votre équipe — ou de l'IA — sont la matière première la plus précieuse.

4 **Testez sur des cas réels, puis itérez**

Confiez à Claude trois tâches réelles avec la Skill active. Chaque écart par rapport à vos attentes est une ligne à ajouter. Une Skill vivante s'enrichit à chaque usage ; une Skill figée meurt en trois mois.

> **LE PIÈGE CLASSIQUE**
>
> La Skill-encyclopédie. Une Skill de quarante pages que Claude charge intégralement à chaque usage coûte des tokens et dilue l'essentiel. Visez des instructions courtes et denses dans `SKILL.md`, et déportez le volumineux (modèles complets, longs exemples) dans des fichiers annexes que Claude ne lira que s'il en a besoin. C'est exactement le principe de la divulgation progressive — et c'est aussi une optimisation de coûts.

04

Les connecteurs : Gmail, Drive, Notion, Figma et les autres

Un connecteur branche Claude sur une application que vous utilisez déjà : il peut alors y lire — et parfois y agir — sans copier-coller. C'est la différence entre demander « résume ce que je te colle » et demander « retrouve dans mes mails et mon Drive tout ce qui concerne le dossier Martin, et fais-m'en la synthèse ».

Comment ça fonctionne

Les connecteurs s'activent dans les paramètres de claude.ai (ou de l'application de bureau), avec une authentification classique par OAuth : vous autorisez l'accès comme vous le feriez pour n'importe quelle application tierce, et vous pouvez le révoquer à tout moment. Sous le capot, la quasi-totalité des connecteurs reposent sur MCP, le protocole que nous détaillons au chapitre suivant — ce qui explique que le catalogue s'élargisse aussi vite : tout éditeur peut publier le sien.

Le catalogue qui compte, par usage

Usage	Connecteurs	Ce que Claude sait faire
Messagerie & agenda	Gmail, Google Calendar, Outlook	Rechercher des fils, préparer des brouillons, analyser un agenda, proposer des créneaux
Documents	Google Drive, SharePoint / OneDrive, Box	Retrouver, lire et croiser des documents ; alimenter une analyse multi-fichiers
Gestion de connaissances	Notion, Confluence	Interroger la base interne, créer et mettre à jour des pages
Gestion de projet	Linear, Jira, Asana, monday.com	Créer et qualifier des tickets, synthétiser l'état d'un sprint ou d'un portefeuille
Communication	Slack, Microsoft Teams	Retrouver des décisions dans les canaux, préparer des messages
Design & création	Figma, Canva	Lire des maquettes pour les implémenter, générer et décliner des visuels
Données & paiement	Stripe, HubSpot, Salesforce	Interroger ventes et pipeline, préparer des reportings
Développement	GitHub, Sentry, Vercel, Supabase	Lire issues et erreurs en production, relier le code aux incidents

QUELQUES CONNECTEURS ET SERVEURS MCP COURANTS

Catalogue indicatif au 12 juin 2026 — le répertoire s'enrichit chaque mois, et tout serveur MCP distant peut s'ajouter comme connecteur personnalisé.

EXEMPLE TYPE · DIRECTION COMMERCIALE

Le brief de rendez-vous qui se prépare seul

Avec Gmail, Calendar et le CRM connectés, la préparation d'un rendez-vous client change de nature. Une seule demande — « prépare mon rendez-vous de 14h » — déclenche : lecture de l'invitation, recherche des échanges récents avec ce contact, état du dossier dans le CRM, et synthèse en une page avec points ouverts et angles d'attaque. Quinze à vingt minutes de préparation ramenées à deux, et surtout : plus de rendez-vous abordé de mémoire.

La question de sécurité à se poser avant d'activer

Un connecteur donne à Claude les accès que *vous* avez. La bonne pratique en organisation tient en trois règles. **Un** : activez les connecteurs sur des comptes aux droits maîtrisés, pas sur un compte administrateur. **Deux** : préférez la lecture seule quand l'éditeur le propose, et réservez l'écriture aux flux où un humain valide (Claude prépare le brouillon, vous l'envoyez). **Trois** : sur les plans Team et Enterprise, passez par la gestion centralisée pour décider quels connecteurs sont disponibles, pour qui.

> **CONNECTEUR, SKILL OU MCP ?**
>
> Les trois se complètent et se confondent souvent. Repère simple : le **connecteur** donne l'accès (« Claude peut lire mon Drive »), la **Skill** donne la méthode (« voici comment nous rédigeons une synthèse »), et **MCP** est la prise standardisée qui rend le premier possible. Un système complet utilise les trois : l'accès, la méthode, la prise.

05

MCP : le protocole qui a unifié l'écosystème

Le Model Context Protocol est probablement la contribution d'Anthropic la plus importante après ses modèles. Publié en standard ouvert fin 2024, adopté depuis par l'ensemble de l'industrie — OpenAI et Google compris — il résout un problème simple : comment brancher n'importe quelle IA sur n'importe quel outil, sans développer une intégration spécifique à chaque paire.

L'idée en une image : le port USB-C de l'IA

Avant MCP, chaque éditeur d'IA développait ses propres intégrations, et chaque outil devait s'adapter à chaque IA. Dix modèles, cent outils : mille intégrations à construire et maintenir. MCP remplace cette multiplication par une norme unique : l'outil expose un **serveur MCP**, l'IA embarque un **client MCP**, et les deux se comprennent — quel que soit l'éditeur de chaque côté.

MCP : UNE NORME UNIQUE ENTRE LES IA ET LES MILLIERS D'OUTILS QUI S'Y BRANCHENT

Ce qu'un serveur MCP expose

Trois choses, dans le vocabulaire du protocole. Des **outils** : des actions que l'IA peut déclencher (« créer un ticket », « interroger la base », « envoyer un message »). Des **ressources** : des données que l'IA peut lire (un fichier, une table, un document). Des **prompts** : des gabarits de tâches prêts à l'emploi. Quand vous activez un connecteur dans claude.ai, vous branchez en

réalité un serveur MCP distant ; quand un développeur ajoute un serveur dans Claude Code, c'est le même mécanisme en local.

```
# Ajouter un serveur MCP a Claude Code — deux exemples

# Un serveur distant (heberge par l'editeur) :
claude mcp add --transport http linear https://mcp.linear.app/mcp

# Un serveur local (lance sur votre machine) :
claude mcp add filesystem -- npx -y @modelcontextprotocol/server-filesystem ~/Documents
```

Les serveurs incontournables en 2026

Serveur	Ce qu'il apporte
GitHub	Issues, pull requests, revues — le pont entre la conversation et le code
Supabase / Postgres	Interroger et administrer ses bases de données en langage naturel
Playwright / navigateur	Piloter un navigateur : tester un site, remplir des formulaires, extraire des données
Figma	Lire les maquettes avec leurs spécifications exactes pour les implémenter fidèlement
Notion / Confluence	La base de connaissances interne comme source interrogeable
Stripe	Paiements, abonnements, factures — interrogés et administrés en séance
Context7	Documentation à jour des bibliothèques de code — l'antidote aux API hallucinées
n8n / Zapier / Make	Déclencher et construire des automatisations multi-applications (chapitre 7)
Apify / Firecrawl	Extraction de données web à l'échelle : veille, prospection, études

Créer son propre serveur MCP

C'est là que MCP devient stratégique pour une organisation : vos outils internes — ERP métier, base clients maison, API propriétaire — peuvent exposer leur propre serveur MCP. Quelques dizaines de lignes avec les SDK officiels (Python ou TypeScript) suffisent pour un premier serveur : vous déclarez vos outils, leurs paramètres, et ce qu'ils renvoient. Dès lors, Claude — et n'importe quelle IA compatible — sait s'en servir.

```python
# Un serveur MCP minimal en Python (SDK officiel FastMCP)
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("inventaire")

@mcp.tool()
def stock_produit(reference: str) -> dict:
    """Renvoie le stock disponible pour une reference produit."""
    return interroger_erp(reference)   # votre logique interne

mcp.run()
```

EXEMPLE TYPE · PME INDUSTRIELLE

L'ERP qui répond enfin aux questions

Scénario : une PME expose trois fonctions de son ERP via un serveur MCP interne — stock par référence, encours client, délais fournisseurs. Coût de développement : quelques jours, car il ne s'agit que d'habiller des requêtes existantes. Résultat : les commerciaux interrogent l'ERP en français depuis claude.ai (« peut-on livrer 400 pièces de la référence X avant fin juillet ? »), sans formation à l'outil, sans solliciter le service ADV. Le serveur ne donne que la lecture : aucune écriture dans l'ERP n'est exposée.

SÉCURITÉ : LA RÈGLE DES TROIS PÉRIMÈTRES

Un serveur MCP est une porte d'entrée vers vos données : traitez-le comme tel. Limitez le périmètre des outils exposés (n'exposez pas « exécuter du SQL » quand « lire le stock » suffit), limitez les droits du compte technique sous-jacent, et n'installez que des serveurs dont l'éditeur est identifié — un serveur MCP malveillant a, par construction, accès à ce que vous lui confiez.

GitHub et Claude Code : développer avec un agent

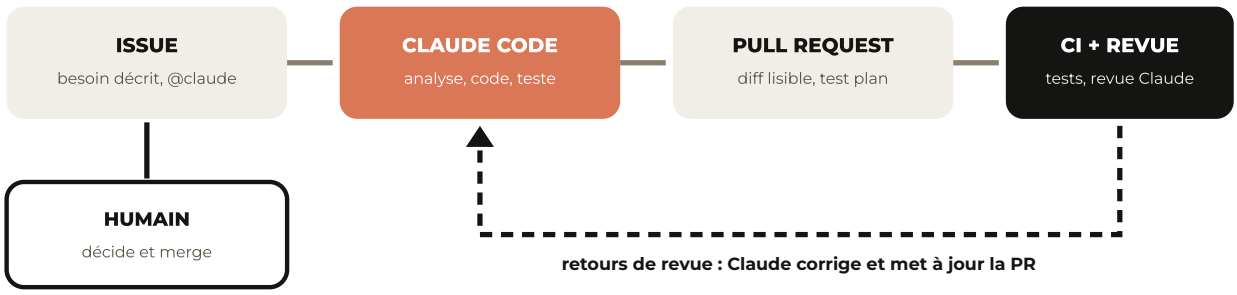
Pour les équipes techniques, l'intégration entre Claude et GitHub est devenue le cœur du réacteur : le code se discute, s'écrit, se relit et se corrige dans un aller-retour continu entre l'agent et le dépôt. Ce chapitre décrit le cycle complet, de l'issue au merge — y compris pour les lecteurs qui ne codent pas mais veulent comprendre ce que leurs équipes mettent en place.

Les trois niveaux d'intégration

Niveau 1 — Claude Code en local. Le développeur travaille dans son terminal ou son éditeur. Claude lit la base de code, propose des modifications, exécute les tests, prépare les commits et les pull requests via la ligne de commande GitHub (`gh`). Le fichier `CLAUDE.md` à la racine du dépôt — versionné comme le reste — fixe les conventions : style, architecture, commandes de build, interdits.

Niveau 2 — Claude dans GitHub. Avec l'application GitHub installée (commande `/install-github-app` depuis Claude Code), Claude devient un participant du dépôt : mentionnez `@claude` dans une issue ou une pull request, et il analyse, implémente ou corrige — directement depuis l'interface GitHub, sans terminal. Un product manager peut ainsi transformer une issue bien rédigée en proposition de code.

Niveau 3 — Claude dans la CI. Via GitHub Actions, Claude s'insère dans l'intégration continue : revue automatique de chaque pull request, détection de problèmes de sécurité, vérification de conformité aux conventions, tri automatique des issues entrantes.



LE CYCLE COMPLET : DE L'ISSUE AU MERGE, L'HUMAIN GARDE LA DÉCISION

Le fichier CLAUDE.md : la mémoire du projet

S'il ne fallait retenir qu'une pratique de ce chapitre, ce serait celle-ci. Le fichier `CLAUDE.md` à la racine du dépôt est lu automatiquement par Claude à chaque session. Les équipes qui en tirent le plus y consignent : les commandes du projet (build, test, lint), l'architecture en quelques lignes, les conventions de style, et surtout les interdits — « ne jamais modifier les migrations existantes », « jamais de secrets en clair ». Chaque erreur récurrente de l'agent corrigée dans CLAUDE.md est une erreur qui disparaît pour toute l'équipe.

```
# Extrait type d'un CLAUDE.md

## Commandes
- Build : npm run build      - Tests : npm test
- Lint avant tout commit : npm run lint

## Conventions
- TypeScript strict, pas de `any`
- Composants : un fichier par composant, props types en tete

## Interdits
- Ne jamais committer de fichier .env
- Ne jamais modifier les migrations deja appliquees
- Pas de nouvelle dependance sans validation humaine
```

La revue de code par Claude

La revue automatique est le niveau d'intégration au meilleur rapport effort/bénéfice. Une GitHub Action déclenche Claude sur chaque pull request : il relève les bugs probables, les failles de sécurité classiques (injections, secrets exposés, validations manquantes), et les écarts aux conventions du projet. Les équipes ne suppriment pas la revue humaine : elles la déchargent du mécanique pour la concentrer sur l'architecture et le métier.

EXEMPLE TYPE · ÉQUIPE PRODUIT, 6 DÉVELOPPEURS

Le cycle complet en pratique

Scénario d'équipe : les issues sont rédigées avec un gabarit précis (contexte, comportement attendu, critères d'acceptation). Les tâches standard — corrections de bugs cernés, petites fonctionnalités — sont confiées à `@claude` depuis l'issue. Chaque pull request, humaine ou agent, passe par la revue Claude en CI, puis par une revue humaine. Le fichier CLAUDE.md s'enrichit à chaque friction.

L'effet le plus durable n'est pas la vitesse : c'est que la qualité des spécifications devient le facteur limitant. Les équipes qui rédigent bien leurs issues tirent de l'agent un travail propre ; les autres découvrent que le problème n'était pas l'outil.

POUR LES LECTEURS NON TECHNIQUES

Ce que ce chapitre change pour vous : le coût marginal d'un développement standard baisse fortement, mais la valeur se déplace vers l'amont — la capacité à décrire précisément ce qu'on veut — et vers l'aval — la capacité à vérifier. Si vous pilotez une équipe ou un prestataire, exigez des spécifications écrites et des critères d'acceptation testables. C'est devenu le premier levier de productivité, avant le choix des outils.

03

Construire : automatisations, agents, systèmes

Les briques sont posées. Cette partie passe à l'assemblage : faire travailler Claude sans vous, lui donner des rôles spécialisés, choisir les bons outils dans un écosystème devenu foisonnant — et déployer le tout dans une organisation sans créer d'usine à gaz.

07

Les automatisations : hooks, agents planifiés, n8n

Tant que vous déclenchez chaque tâche à la main, Claude reste un outil. Il devient un système quand les tâches se déclenchent seules : à heure fixe, sur un événement, ou en réaction à sa propre activité. Trois mécanismes couvrent l'essentiel des besoins.

1. Les hooks : des règles qui s'exécutent toujours

Dans Claude Code, un hook est une commande qui s'exécute automatiquement à un moment précis du travail de l'agent : avant qu'il n'utilise un outil, après une modification de fichier, à la fin d'une session. La nuance est importante : une instruction dans CLAUDE.md est une consigne que le modèle *suit* ; un hook est une règle que le système *exécute*, à chaque fois, sans exception possible.

Usages types : reformater automatiquement chaque fichier modifié, lancer les tests après chaque série de changements, bloquer toute commande destructrice, notifier un canal d'équipe en fin de session. Règle de partage des rôles : ce qui relève du jugement va dans les instructions, ce qui doit être garanti va dans un hook.

2. Les agents planifiés : Claude à heure fixe

Deuxième mécanisme : déclencher Claude sur un calendrier. Claude Code s'exécute en mode non interactif (`claude -p "votre consigne"`), ce qui le rend programmable comme n'importe quelle tâche : un planificateur (cron, GitHub Actions programmées, ou les tâches planifiées intégrées à Claude Code) le lance chaque matin, chaque lundi, chaque fin de mois.

C'est le mécanisme des tâches récurrentes à faible décision : revue de presse interne à partir de sources définies, synthèse hebdomadaire des tickets clients, vérification de cohérence de données, rapport d'activité compilé depuis plusieurs sources. Le livrable arrive par mail, dans Slack ou dans un dossier — relu par un humain avant toute diffusion externe.

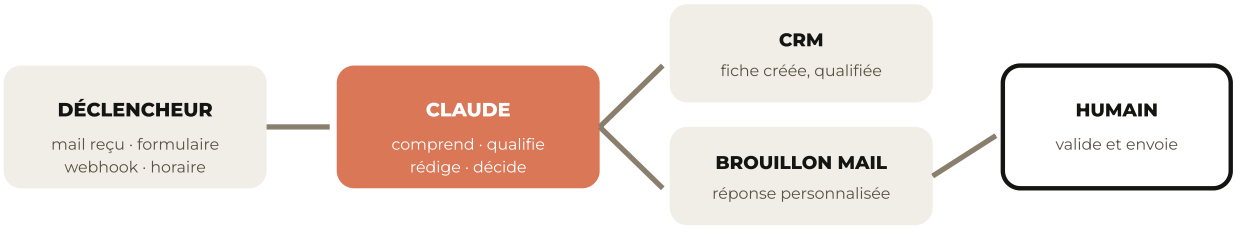
```
# Un agent planifie minimal : revue de veille chaque matin a 7h30
# (crontab + Claude Code en mode non interactif)

30 7 * * 1-5  claude -p "Lis les flux RSS listes dans veille/sources.txt. \
  Resume les 5 informations les plus utiles pour une PME du batiment, \
  avec liens. Ecris le resultat dans veille/brief-du-jour.md" \
  --allowedTools "Read,Write,WebFetch"
```

3. n8n, Zapier, Make : l'automatisation multi-applications

Troisième mécanisme, pour les flux qui traversent plusieurs applications : les plateformes d'automatisation. n8n (auto-hébergeable, le favori des équipes techniques européennes), Zapier et Make intègrent tous des nœuds Claude natifs : un événement déclencheur (mail reçu, formulaire soumis, ligne ajoutée, webhook), un passage par Claude (analyse, rédaction, extraction, décision), puis des actions dans vos outils.

La bonne répartition des rôles : la plateforme gère le *flux* (déclencheurs, branchements, reprises sur erreur), Claude gère l'*intelligence* (comprendre un texte libre, qualifier, rédiger). Résistez à la tentation de tout faire faire au modèle : un branchement conditionnel simple coûte zéro token dans n8n.



Exemple : qualification automatique des demandes entrantes — le flux dans n8n, l'intelligence dans Claude, la décision chez l'humain

L'ARCHITECTURE TYPE D'UNE AUTOMATISATION : FLUX, INTELLIGENCE, VALIDATION

EXEMPLE TYPE · AGENCE IMMOBILIÈRE

Les demandes entrantes triées avant le café

Scénario : chaque demande entrante (portails, site, mail) déclenche un flux n8n. Claude extrait les critères (budget, secteur, type de bien, horizon), qualifie l'urgence, crée la fiche dans le CRM et prépare un brouillon de réponse proposant deux créneaux de visite tirés de l'agenda. L'agent humain valide chaque envoi. Ordre de grandeur du coût modèle, aux tarifs officiels de juin 2026 : avec Sonnet 4.6, quelques centimes par demande traitée — à comparer aux dix minutes humaines que prenait chaque tri.

Quel mécanisme pour quel besoin ?

Besoin	Mécanisme	Exemple
Garantir une règle pendant que Claude travaille	Hook	Tests après chaque modification
Tâche récurrente à heure fixe	Agent planifié	Brief de veille quotidien
Réaction à un événement externe	n8n / Zapier / Make	Qualification d'une demande entrante
Flux complexe multi-étapes avec reprises	n8n + sous-flux Claude	Onboarding client de bout en bout

> **LA RÈGLE DES TROIS OCCURRENCES**
>
> N'automatisez pas une tâche que vous n'avez pas faite trois fois manuellement avec Claude. Les deux premières fois révèlent les cas particuliers, la troisième stabilise le prompt. Automatiser trop tôt fige les défauts — et un défaut exécuté deux cents fois par mois coûte plus cher que trois exécutions manuelles.

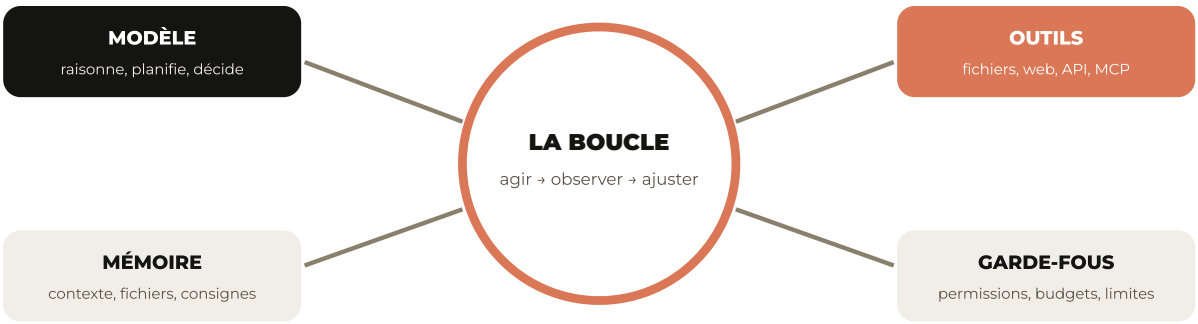
08

Créer ses propres agents IA, étape par étape

Un agent n'est pas un chatbot amélioré : c'est un système qui poursuit un objectif en enchainant des actions — lire, chercher, écrire, exécuter, vérifier — jusqu'au résultat. La bonne nouvelle de 2026 : en construire un ne demande plus une équipe de recherche. La mauvaise : un agent mal conçu échoue de manière plus coûteuse qu'un chatbot.

L'anatomie d'un agent

Tout agent, du plus simple au plus sophistiqué, combine quatre éléments. Un **modèle** qui raisonne et décide. Des **outils** qui lui donnent prise sur le monde — fichiers, navigateur, API, base de données, le plus souvent via MCP. Une **mémoire** : le contexte de la session, des fichiers de travail, et ce qu'il consigne pour durer au-delà d'une session. Et une **boucle** : il agit, observe le résultat, ajuste, recommence — jusqu'à l'objectif ou jusqu'à une limite que vous avez fixée.



MODÈLE + OUTILS + MÉMOIRE + GARDE-FOUS, AUTOUR D'UNE BOUCLE D'ACTION

Trois voies pour construire, selon votre profil

Voie 1 — Sans code : les sous-agents de Claude Code. Un fichier Markdown dans `.claude/agents/` suffit à définir un agent spécialisé : son rôle, ses instructions, les outils auxquels il a droit. Claude Code le délègue alors automatiquement quand la tâche correspond. C'est la voie royale pour démarrer : un « relecteur sécurité », un « rédacteur de tests », un « analyste de données » se créent en une heure.

Voie 2 — Avec une plateforme : n8n et assimilés. Les agents n8n combinent un modèle Claude, des outils (vos applications connectées) et une mémoire de conversation, dans une in-

terface visuelle. Adapté aux agents de flux : qualification, suivi, relance, support de premier niveau.

Voie 3 — En code : le Claude Agent SDK. Pour un agent produit — intégré à votre logiciel, exposé à vos clients — le SDK officiel (Python, TypeScript) fournit l'infrastructure complète de Claude Code en bibliothèque : la boucle d'action, les outils fichiers et exécution, MCP, les permissions, les sous-agents. Vous écrivez la spécialisation, pas la plomberie.

```
# Un sous-agent Claude Code : .claude/agents/relecteur-rgpd.md
---
name: relecteur-rgpd
description: Relit tout document ou formulaire collectant des donnees
  personnelles. A utiliser avant publication.
tools: Read, Grep
---
Tu es charge de la conformite RGPD. Pour chaque document :
1. Liste les donnees personnelles collectees et leur finalite declaree
2. Signale toute collecte sans finalite explicite
3. Verifie mentions d'information, duree de conservation, droits
4. Classe chaque point : conforme / a corriger / a faire valider
   par un juriste. Tu ne donnes jamais d'avis juridique definitif.
```

Les cinq règles qui séparent un agent fiable d'un jouet

1 — Un agent, une mission

« Assistant général de l'entreprise » est une impasse. « Qualifier les demandes entrantes selon notre grille » est un agent qui marche. La spécialisation est le premier facteur de fiabilité.

2 — Des outils minimaux

Donnez à l'agent les outils de sa mission, rien de plus. Un agent de lecture n'a pas besoin du droit d'écrire ; un agent interne n'a pas besoin d'envoyer des mails. Chaque outil en plus est une surface d'erreur en plus.

3 — Des limites explicites

Budget de tokens par exécution, nombre maximal d'actions, périmètre de fichiers, liste d'actions interdites. Un agent sans limites finit toujours par les trouver tout seul — au pire moment.

4 — L'humain au bon endroit

Identifiez les actions irréversibles (envoi externe, paiement, suppression, publication) et placez-y une validation humaine. Tout le reste peut être autonome. C'est ce placement — pas le taux d'automatisation — qui fait un système sûr.

5 — Évaluez avant de déployer

Constituez vingt cas réels avec le résultat attendu, et mesurez. Un agent qui réussit 19 cas sur 20 se déploie avec une validation humaine sur le vingtième ; un agent à 12 sur 20 retourne en conception. Sans cette mesure, vous déployez une impression.

EXEMPLE TYPE · SUPPORT CLIENT E-COMMERCE

L'agent de premier niveau qui connaît ses limites

Scénario : un agent traite les demandes de support entrantes. Mission unique : répondre aux questions dont la réponse figure dans la base (livraison, retours, tailles), avec accès en lecture à la base de connaissances et au statut des commandes. Garde-fous : il ne promet jamais de remboursement, ne modifie jamais une commande, et transfère à un humain dès qu'un client exprime un mécontentement ou un cas hors périmètre. Résultat type de ce design : la majorité des demandes résolues en quelques minutes à toute heure, et — point décisif — des transferts humains *plus rapides* qu'avant, car déjà qualifiés.

Cinq agents clients, par secteur

Des exemples concrets, représentatifs de ce que nous déployons. Chacun a une mission unique, des outils minimaux et un point de validation humaine.

Secteur	Agent	Ce qu'il fait
Immobilier	Qualifieur de demandes	Lit chaque demande entrante, extrait budget/secteur/bien, crée la fiche CRM, propose deux créneaux de visite. Brouillon validé par l'agent humain.
E-commerce	Support de niveau 1	Répond aux questions livraison/retours/tailles depuis la base, suit les commandes, transfère tout mécontentement à un humain — qualifié.
Cabinet conseil	Rédacteur de propositions	Transforme un brief en proposition conforme à la charte (Skill dédiée), chiffrage et périmètre inclus/exclu. Relecture associé.
Industrie	Assistant ERP	Répond en langage naturel sur stock, encours et délais via un serveur MCP en lecture seule. Aucune écriture exposée.
Santé / libéral	Préparateur de dossiers	Compile et résume les pièces d'un dossier multi-documents, signale les manques, ne pose jamais de diagnostic. Validation praticien obligatoire.

Le fil commun : aucun de ces agents n'est « l'IA de l'entreprise ». Ce sont cinq spécialistes étroits, chacun mesuré sur ses vingt cas réels avant mise en service. C'est l'addition de petits agents fiables qui fait un système, pas un grand agent vague.

MODÈLE PAR RÔLE : LA RÈGLE D'OR DES AGENTS

Dans un système multi-agents, le routage des modèles fait la rentabilité : l'orchestrateur qui planifie mérite un modèle puissant (Opus 4.8, voire Fable 5 pour les missions longues), les exécutants spécialisés tournent très bien sur Sonnet 4.6, et les tâches de volume (tri, extraction, classification) sur Haiku 4.5 — à 2 % du coût de Fable. Le chapitre 12 chiffre précisément cet écart.

09

L'écosystème 2026 : les outils qui se connectent à Claude

Autour de Claude s'est constitué en dix-huit mois un écosystème complet : éditeurs de code, plateformes d'automatisation, outils de design, bases de données, navigateurs. Ce chapitre dresse la carte — pour que vous choisissiez en connaissance de cause, pas par effet de mode.

Les éditeurs et environnements de développement

Claude alimente aujourd'hui la plupart des outils de développement assisté. **Claude Code** reste la référence pour le travail agentique — terminal, extensions VS Code et JetBrains, application de bureau, version web. **Cursor**, l'éditeur IA le plus répandu, propose les modèles Claude en première intention — son PDG revendiquait dès le lancement la première place de Fable 5 sur son benchmark interne. **GitHub Copilot** intègre également les modèles Claude, ce qui les met à disposition de toute équipe déjà équipée. **Windsurf** et les environnements cloud (Replit, Devin) complètent le paysage.

Notre lecture : le choix de l'éditeur est secondaire ; le choix qui compte est celui du *mode de travail*. L'autocomplétion améliore un développeur ; l'agent (Claude Code, mode agent de Cursor) change l'organisation du travail. Les équipes les plus efficaces que nous observons utilisent les deux, mais structurent leurs projets pour l'agent : CLAUDE.md, Skills, tests solides.

L'automatisation et les agents

Outil	Positionnement	Pour qui
n8n	Automatisation visuelle, auto-hébergeable, nœuds agents natifs	Équipes techniques, exigence de souveraineté
Zapier	Le plus grand catalogue d'applications, MCP intégré	Équipes métier, mise en route rapide
Make	Flux visuels complexes, bon rapport prix/volume	Agences, PME multi-outils
Claude Agent SDK	L'infrastructure de Claude Code en bibliothèque	Agents produit, intégrations profondes
LangGraph / frameworks	Orchestration multi-agents fine, multi-fournisseurs	Équipes IA dédiées

LES PLATEFORMES QUI SE BRANCHENT À CLAUDE

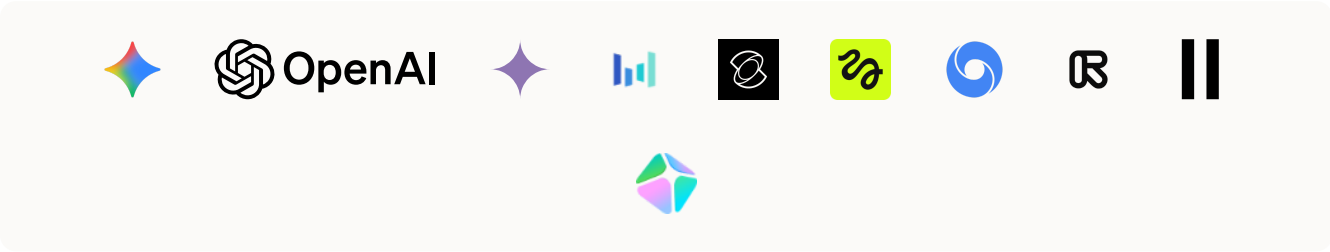
Données, design, navigateur : les serveurs MCP qui comptent

Le chapitre 5 a posé le principe ; voici la pratique, par fonction. **Données** : les serveurs Supabase et Postgres transforment vos bases en interlocuteurs — requêtes, migrations, diagnostics en langage naturel. **Design** : le serveur Figma donne à Claude les spécifications exactes des maquettes (espacements, couleurs, composants), ce qui a fait disparaître une bonne part des allers-retours design-développement ; Canva permet génération et déclinaison de visuels en volume ; et Claude Design (chapitre 2) ferme la boucle en produisant prototypes et présentations directement à votre charte. **Navigateur** : Playwright MCP et l'extension Claude in Chrome donnent à l'agent un navigateur complet — tests de bout en bout, extraction, opérations sur des sites sans API. **Documentation** : Context7 fournit la documentation à jour des bibliothèques, l'antidote le plus efficace au code obsolète.

La génération média : image, vidéo, voix

Claude ne génère pas d'image ni de vidéo : il *orchestre* les modèles qui le font. La plupart ne s'installent pas un par un — on branche un agrégateur. Le serveur MCP **fal.ai** donne accès à plus de mille modèles (image, vidéo, audio, 3D) en une commande :

Besoin	Modèles de référence	Usage
Image	Nano Banana Pro (Google), gpt-image-2 (OpenAI), Imagen 4, Seedream	Visuels de marque, illustrations d'articles, déclinaisons, retouche par prompt
Vidéo	Seedance, Kling, Google Veo 3, Higgsfield	Reels, démos produit, animations à partir d'un texte ou d'une image
Voix & audio	ElevenLabs (synthèse et clonage), CSM, ThinkSound	Voix off, doublage, narration de reels, podcasts



IMAGE, VIDÉO, VOIX — LES MODÈLES QUE CLAUDE ORCHESTRE

```
# Brancher fal.ai a Claude Code : 1000+ modeles (Seedance, Kling, Nano Banana...)
claude mcp add --transport http fal-ai https://mcp.fal.ai/mcp \
  --header "Authorization: Bearer VOTRE_CLE_FAL"
```

Une fois branché, Claude choisit et appelle le modèle adapté (Seedance ou Kling pour la vidéo, Nano Banana ou Imagen pour l'image) sans configuration supplémentaire. **Précision honnête :** tous les modèles n'ont pas de MCP dédié. Higgsfield, par exemple, s'utilise via fal.ai ou sa propre API ; ElevenLabs a son MCP officiel (`claude mcp add --transport http elevenlabs https://mcp.elevenlabs.io/mcp`) pour la voix. La règle : un agrégateur (fal) pour la majorité, un MCP dédié quand l'éditeur en publie un, l'API en dernier recours.

L'intérêt n'est pas le modèle isolé, c'est la **chaîne** : Claude écrit le script, ElevenLabs le dit, Seedance ou Kling l'illustre en mouvement, et un dernier passage assemble le tout. Un reel hebdomadaire qui mobilisait une demi-journée se produit en quelques minutes — le goût et la validation restant humains. C'est exactement ce type de chaîne que nous montons pour la production de contenu (voir les stacks marketing au chapitre 10).

Claude dans vos outils bureautiques

Le mouvement de 2026 le plus visible pour les équipes non techniques : Claude arrive *dans* les outils existants plutôt que l'inverse. Claude pour Excel travaille directement dans les classeurs ; les Skills Office génèrent des documents Word, PowerPoint et PDF conformes à vos gabarits ; l'extension Chrome ramène l'agent dans n'importe quelle application web. La conséquence

pratique : le déploiement de l'IA ne passe plus nécessairement par un changement d'outils — il passe par l'équipement des outils en place.

COMMENT CHOISIR SANS SE TROMPER

Trois questions suffisent à trier ce catalogue. **Où sont vos données ?** Les outils qui s'y connectent nativement gagnent. **Qui va opérer ?** Une équipe métier ira plus loin avec Zapier qu'avec un SDK ; l'inverse est vrai pour une équipe technique. **Le standard est-il ouvert ?** À fonctionnalités égales, préférez ce qui parle MCP : vous changerez d'outil sans tout reconstruire.

10

Les stacks qui fonctionnent, profil par profil

Une stack n'est pas une liste de courses : c'est un petit nombre d'outils qui se renforcent. Voici quatre configurations éprouvées, une par profil, avec le modèle adapté à chaque étage. Chacune démarre en quelques jours et grandit sans refonte.

Stack 1 — Dirigeant / indépendant : l'état-major personnel

Brique	Choix	Rôle
Surface	claude.ai (Pro ou Max) + application de bureau	Le poste de travail
Connecteurs	Gmail, Calendar, Drive	Mails, agenda et documents interrogeables
Projets	Un par dossier de fond	Contexte permanent, plus de répétition
Skills	Note de décision, compte rendu, relance	Vos formats, à chaque fois
Modèle	Opus 4.8 par défaut	Fable 5 réservé aux analyses lourdes

Premier résultat visible : la préparation des rendez-vous et la production des comptes rendus. Montez ensuite vers Cowork pour les dossiers épais (due diligence, réponse à appel d'offres).

Stack 2 — Marketing / contenu : la fabrique éditoriale

Brique	Choix	Rôle
Surface	claude.ai Team + Projets par campagne	Production et cohérence
Skills	Charte éditoriale, formats par canal, check-list publication	La voix de la marque, verrouillée
Connecteurs	Notion (calendrier éditorial), Canva, Drive	De l'idée au visuel sans rupture
Automatisation	n8n ou Zapier : veille quotidienne, recyclage des contenus	Le récurrent sans effort
Modèles	Sonnet 4.6 pour produire, Opus 4.8 pour les pièces maîtresses	Coût maîtrisé en volume

Le levier décisif de cette stack n'est pas la génération : c'est la Skill de charte éditoriale, qui fait la différence entre du contenu IA générique et du contenu qui vous ressemble.

Stack 3 — Équipe technique : l'usine logicielle

Brique	Choix	Rôle
Agent	Claude Code (terminal + éditeur)	Le cœur du travail
Dépôt	GitHub + app Claude + Actions (revue auto)	Le cycle issue → PR → merge du chapitre 6
MCP	Supabase/Postgres, Figma, Playwright, Sentry, Context7	Données, design, tests, incidents, docs
Conventions	CLAUDE.md + Skills + sous-agents (revue, tests, sécurité)	La qualité reproductible
Modèles	Sonnet 4.6 au quotidien, Opus 4.8 en architecture, Fable 5 sur les migrations lourdes	Routage par difficulté

Stack 4 — Opérations / support : le back-office augmenté

Brique	Choix	Rôle
Orchestration	n8n (auto-hébergé si données sensibles)	Flux, reprises, journalisation
Intelligence	API Claude — Haiku 4.5 pour trier, Sonnet 4.6 pour rédiger	Le bon modèle par étape
Accès données	Serveur MCP interne en lecture (ERP, CRM, base support)	Des réponses ancrées dans le réel
Validation	Brouillons et files d'attente humaines sur tout envoi externe	La sécurité du dispositif
Mesure	Jeu d'évaluation + suivi mensuel des coûts par flux	Le pilotage

ORDRE DE GRANDEUR · COÛT MODÈLE D'UN FLUX SUPPORT

Le calcul qu'il faut savoir refaire

Prenons 2 000 demandes par mois. Chaque demande : un tri par Haiku 4.5 (~1 500 tokens en entrée, 300 en sortie, soit environ 0,003 $), puis pour la moitié d'entre elles une réponse rédigée par Sonnet 4.6 avec contexte documentaire (~4 000 tokens en entrée, 500 en sortie, soit environ 0,02 $). Total modèle : autour de **26 $ par mois** aux tarifs officiels de juin 2026. Le même flux entièrement servi par Fable 5 coûterait plus de quinze fois plus — pour un tri de mails. C'est toute la thèse de ce guide : l'architecture fait le prix.

> **LE DÉNOMINATEUR COMMUN**
>
> Ces quatre stacks partagent trois invariants : des Skills qui capturent vos méthodes, MCP comme prise universelle vers vos données, et un routage de modèles explicite. Les outils autour peuvent changer — ces trois invariants survivront aux modes.

La grille d'application, secteur par secteur

Les profils ci-dessus disent *qui* utilise quoi. Voici *où* appliquer en priorité, secteur par secteur : la tâche à plus fort impact, la première brique à poser, le premier gain visible. Une grille de départ, à adapter à votre réalité.

Secteur	Tâche à fort impact	Première brique	Premier gain
Immobilier	Qualifier et répondre aux demandes entrantes	Agent qualifieur + CRM connecté	Zéro lead perdu, préparation de visite automatique
E-commerce / retail	Support de niveau 1, fiches produit	Agent support + base de connaissances MCP	Majorité des demandes résolues sous quelques minutes
Conseil / agence	Propositions, comptes rendus, cadrage	Skills de format (charte maison)	Premier jet conforme, relecture senior allégée
Industrie / PME prod	Interroger l'ERP, suivre stock et délais	Serveur MCP interne en lecture	Réponses ADV en langage naturel, sans ticket
Santé / libéral	Préparation et synthèse de dossiers	Agent de synthèse multi-documents	Dossiers prêts, manques signalés — validation praticien
BTP / artisanat	Devis, suivi de chantier, relances	Skills devis + relance par paliers	Devis sortis le jour même, relances tenues
Événementiel	Rétro-planning, briefs prestataires, suivi	Projets claude.ai + Skills briefs	Coordination multi-prestataires sans rien oublier
Restauration / hôtellerie	Avis en ligne, contenus, plannings	Skill réponse-avis + génération contenu	Réputation tenue, communication régulière
Juridique	Analyse documentaire, première lecture	Cowork (édition juridique) + Skills	Tri et résumé de volumes, l'avocat tranche
Comptable / finance	Lecture de relevés, reporting mensuel	Skills reporting + lecture pièces	Clôture commentée plus rapide, écarts signalés
RH / recrutement	Tri de candidatures, fiches de poste	Skill analyse-candidature	Pré-qualification factuelle, entretiens ciblés
Fashion / marque	Contenu, déclinaisons visuelles, lookbooks	Stack contenu + génération image (ch. 9)	Production visuelle en volume, charte tenue

Trois secteurs, en détail

ÉVÉNEMENTIEL

Le copilote de coordination

Un événement, c'est cent prestataires et mille détails. Un Projet claude.ai par événement réunit cahier des charges, planning et contacts ; des Skills produisent les briefs prestataires au format maison et les rétro-plannings. Connecté à l'agenda et aux mails, Claude prépare les points hebdomadaires, signale les jalons qui glissent et rédige les relances. Le gain n'est pas la créativité — elle reste humaine — c'est de ne *rien* laisser tomber entre deux réunions.

FASHION / MARQUE

La fabrique de contenu visuel

La stack marketing (chapitre 10) couplée à la génération d'image (chapitre 9) change l'échelle : une Skill de charte verrouille la voix, Nano Banana Pro et gpt-image-2 produisent et déclinent les visuels, une chaîne n8n recycle chaque pièce par canal. L'enjeu décisif reste la cohérence : c'est la Skill de charte, pas le modèle d'image, qui distingue une marque d'un flux de visuels génériques.

SANTÉ

L'assistant documentaire, jamais le diagnostic

En santé, la frontière est nette et se câble dans l'agent : Claude compile, résume, signale les pièces manquantes d'un dossier multi-documents — et ne pose *jamais* de diagnostic ni de recommandation clinique. Toute sortie passe par le praticien, qui engage sa responsabilité. Bien borné, l'agent fait gagner un temps considérable sur la préparation ; mal borné, il est un risque. C'est l'exemple type où le placement de l'humain (chapitre 11) prime sur le taux d'automatisation.

11

Déployer un système IA dans votre organisation : la méthode

La technologie de ce guide se met en place en quelques semaines. Ce qui échoue, dans les déploiements que nous observons, n'est presque jamais technique : c'est le choix des cas d'usage, l'adoption, et l'absence de mesure. D'où une méthode en cinq étapes, volontairement frugale.

1. **Cartographier le temps, pas les fantasmes**

   Une semaine d'observation : où partent les heures, équipe par équipe ? Cherchez les tâches récurrentes, textuelles, à règles explicables — c'est le gisement. Méfiez-vous des cas d'usage spectaculaires : le tri du courrier entrant rapporte souvent plus que le chatbot vitrine.

2. **Choisir deux cas, pas douze**

   Critères : fréquence élevée, coût d'erreur faible, résultat vérifiable, et un volontaire motivé pour le porter. Deux cas menés au bout valent mieux que douze pilotes qui s'enlisent — la dispersion est la première cause d'échec.

3. **Équiper avant d'automatiser**

   Phase manuelle d'abord : les utilisateurs travaillent avec Claude équipé (Skills, connecteurs, Projets) sur le cas choisi. C'est là que se révèlent les cas particuliers et que se construit la confiance. L'automatisation (chapitre 7) ne vient qu'après — règle des trois occurrences.

4. **Mesurer trois choses, et les publier**

   Temps gagné sur la tâche, qualité (taux de reprise humaine), coût (tokens par unité traitée). Publiés en interne, ces trois chiffres font plus pour l'adoption que n'importe quelle formation — et ils révèlent vite les flux à arrêter.

5. **Industrialiser ce qui a fait ses preuves**

   Le cas validé passe en automatisation, ses Skills deviennent un standard d'équipe, et vous attaquez le cas suivant avec ce qui est désormais une infrastructure : comptes, connecteurs, conventions, réflexes. Le deuxième déploiement coûte moitié moins que le premier.

Les trois questions qui fâchent — à traiter au début

Les données. Quelles données peuvent transiter par le modèle, lesquelles jamais ? Écrivez-le. Les offres Team et Enterprise n'entraînent pas les modèles sur vos données ; les accès par API se configurent finement ; et un serveur MCP interne en lecture seule vaut mieux qu'un export de base envoyé dans une conversation.

Les personnes. Dites clairement ce que l'IA change pour chaque poste — et ce qu'elle ne change pas. Les déploiements réussis nomment des référents par équipe, forment sur les cas réels de l'équipe (jamais sur des démos génériques), et valorisent les agents qui partagent leurs prompts et leurs Skills.

La responsabilité. Qui répond d'une sortie erronée ? Règle simple et robuste : toute production IA qui sort de l'organisation (mail client, devis, publication) a un valideur humain nommé. L'IA produit, quelqu'un signe. Cette règle unique désamorce 90 % des inquiétudes légitimes.

> **LE SIGNAL QUI NE TROMPE PAS**
>
> Un déploiement réussi se reconnaît à ceci : au bout de deux mois, ce sont les équipes qui demandent le cas d'usage suivant — pas la direction qui le pousse. Si ce signal n'apparaît pas, le problème est dans le choix des cas ou dans l'accompagnement, presque jamais dans l'outil.

04

Maîtriser la facture

Deux vérités se sont imposées en juin 2026 : l'IA de pointe rejoint l'électricité et le cloud — une ressource facturée à la consommation, donc à piloter — et son accès peut être coupé par une décision politique, comme l'a montré la suspension de Fable 5. Cette partie vous donne les instruments pour maîtriser les deux : le coût et la dépendance.

12

Optimiser ses tokens : la mécanique réelle

Tout le monde sait que l'IA se facture au token. Ce que peu de gens savent, c'est *où* partent réellement les tokens : dans la relecture d'historique, les définitions d'outils, le raisonnement interne — pas dans vos questions. Ce chapitre démonte la facture poste par poste, avec les mécanismes exacts de l'API.

L'anatomie d'une requête : ce que vous payez vraiment

Ouvrez `/context` dans Claude Code et regardez ce que coûte un simple « bonjour » : 20 à 30 000 tokens d'entrée avant même votre message. Le prompt système de l'outil, les définitions de chaque outil disponible (chaque serveur MCP connecté ajoute les siennes), votre CLAUDE.md, les Skills chargées, puis l'historique de la conversation — **relu intégralement à chaque tour**. C'est le point que tout le monde rate : dans une conversation de 40 tours, le 40° message paie la relecture des 39 précédents. Le coût d'une conversation croît avec le carré de sa longueur, pas linéairement.

Trois conséquences immédiates. Un : `/clear` entre deux sujets n'est pas de l'hygiène, c'est de l'argent. Deux : chaque serveur MCP connecté « au cas où » se paie à *chaque* requête, même jamais utilisé — ne gardez branché que l'utile. Trois : les sous-agents (chapitre 8) ne sont pas qu'une organisation du travail, c'est une isolation de contexte — l'exploration salissante part dans le contexte du sous-agent et n'encombre jamais plus la session principale.

Les tokens invisibles : le raisonnement se facture en sortie

Les modèles récents réfléchissent avant de répondre, et ces tokens de raisonnement sont **facturés comme des tokens de sortie** — au tarif fort. Sur Fable 5, le raisonnement adaptatif ne se désactive pas : c'est précisément pour ça que Simon Willison a mesuré 110 $ en une journée, et c'est l'angle mort des comparatifs de prix. Un modèle à 50 $/MTok en sortie qui raisonne longuement coûte bien plus que « deux fois Opus » sur les tâches où il s'obstine. Via l'API, les budgets de réflexion se contrôlent (paramètre de budget sur le thinking, `max_tokens` en plafond dur) — un agent en production sans plafond de sortie est une facture sans plafond.

Modèle	Entrée	Sortie	Cache (lecture)	Batch (entrée/sortie)
Fable 5	10 $	50 $	1 $	5 $ / 25 $
Opus 4.8	5 $	25 $	0,50 $	2,50 $ / 12,50 $
Sonnet 4.6	3 $	15 $	0,30 $	1,50 $ / 7,50 $
Haiku 4.5	1 $	5 $	0,10 $	0,50 $ / 2,50 $

$/MTok, tarifs officiels au 12 juin 2026. Cache : lecture à 10 % du tarif d'entrée, écriture à 125 %. Au-delà de 200k tokens de contexte, certains modèles appliquent un tarif majoré (Sonnet : 6 $/22,50 $) — vérifiez avant d'envoyer une base de code entière.

Le cache de prompt : les règles que personne ne lit

Le principe est connu — relire depuis le cache coûte 10 % du tarif. Les règles qui font qu'il *marche* le sont moins :

Le cache est un préfixe. Il couvre le début de la requête jusqu'à vos points de cache (4 maximum). Un seul caractère modifié en amont — un horodatage dans le prompt système, l'ordre des outils qui change — et tout le cache aval est invalidé, refacturé plein pot ×1,25. La règle de construction est mécanique : strictement stable en tête (instructions, Skills, références), variable en queue (la demande du jour). Jamais de date générée, jamais d'aléatoire dans la partie cachée.

Le cache expire en 5 minutes (TTL glissant, rafraîchi à chaque lecture — une option 1 heure existe, écriture plus chère). Conséquence contre-intuitive : un agent qui traite ses dossiers *en rafale* coûte beaucoup moins cher que le même agent déclenché toutes les 20 minutes — même volume, cache froid à chaque fois. Groupez les exécutions.

Il y a un minimum cachable (de l'ordre de 1 000 à 2 000 tokens selon le modèle) : inutile de poser un point de cache sur trois lignes. Et l'endpoint de **comptage de tokens** de l'API est gratuit — mesurez vos prompts avant de les optimiser, pas après.

Le contexte long : puissant, et piégé

La fenêtre d'un million de tokens de Fable 5 invite à tout charger. Trois raisons de résister. Le coût : un million de tokens d'entrée sur Fable, c'est 10 $ *par message* si le cache ne couvre pas. La qualité : un contexte gorgé de documents non pertinents dégrade la précision des réponses — le modèle retrouve moins bien l'aiguille quand on a versé trois bottes de foin. La bonne architecture : pour un corpus interrogé souvent, un serveur MCP qui va chercher le passage utile bat le corpus entier collé en contexte — à la fois moins cher et plus juste. Le

contexte long sert aux tâches qui exigent *vraiment* une vue d'ensemble : migration, audit transversal, synthèse multi-documents où tout compte.

Côté agents, l'API et Claude Code gèrent désormais le contexte automatiquement : compactage de l'historique en résumé, purge des vieux résultats d'outils. `/cost` et `/context` dans Claude Code, et le suivi par clé dans la console, vous donnent les chiffres réels — toute optimisation commence par dix minutes là-dedans, pas par une intuition.

La hiérarchie des leviers, dans l'ordre du rendement

1. **Router (facteur 3 à 50)**

   Le bon modèle par tâche — y compris à l'intérieur d'un même flux : tri Haiku, rédaction Sonnet, arbitrage Opus. Les sous-agents Claude Code acceptent un modèle par rôle (`model: haiku`) : la session principale reste sur le modèle fort, les tâches mécaniques descendent de gamme.

2. **Cacher (facteur 5 à 10 sur l'entrée)**

   Préfixe stable, points de cache bien posés, exécutions groupées dans la fenêtre de TTL.

3. **Batcher (÷2 sur tout)**

   Tout ce qui tolère 24 h passe en Batch — et le batch se cumule avec le cache et le routage (recette 5, chapitre 15 : 10 000 fiches pour ~4 $).

4. **Dégraisser le contexte (20 à 50 %)**

   `/clear` par sujet, serveurs MCP débranchés quand inutiles, Skills en divulgation progressive, documents joints à la demande et pas « au cas où ».

5. **Plafonner la sortie (l'assurance)**

   `max_tokens` et budgets de raisonnement sur tout flux automatisé. Ça n'économise rien en régime normal — ça empêche le jour anormal de coûter un mois de budget.

CAS CHIFFRÉ · LE MÊME FLUX, NAÏF PUIS OPTIMISÉ

Support client, 2 000 demandes par mois : de 400 $ à 26 $

Version naïve : tout sur Fable 5, base documentaire collée en contexte à chaque demande, temps réel, sortie non bornée — environ 400 $ par mois. Version optimisée : tri Haiku (levier 1), grille et documentation en préfixe caché (levier 2), enrichissements nocturnes en batch (levier 3), contexte réduit au passage pertinent via MCP (levier 4), 500 tokens de sortie max (levier 5) — environ 26 $, qualité mesurée égale sur le jeu d'évaluation. Aucun de ces cinq gestes ne prend plus d'une demi-journée.

ET DANS LES ABONNEMENTS ?

Les limites des forfaits (Pro, Max) se consomment selon la même mécanique : conversations-fleuves, MCP superflus et modèle surdimensionné épuisent vos quotas exactement comme ils gonfleraient une facture API. Mêmes réflexes, même rendement — la seule différence est que le coût se lit en heures d'attente du reset plutôt qu'en dollars.

13

Facturation à l'usage et dépendance : reprendre la main

Deux forces redessinent le coût et le risque de l'IA. La première : les modèles de pointe quittent le forfait prévisible pour la facturation à la consommation. La seconde, révélée brutalement par la suspension de Fable 5 le 12 juin : l'accès lui-même peut être coupé du jour au lendemain. Ce chapitre transforme les deux en plan d'action.

Le mouvement de fond : du forfait au compteur

Les abonnements Pro, Max, Team et Enterprise couvrent Opus 4.8, Sonnet et Haiku. Mais la trajectoire de l'industrie est claire : les modèles de frontière coûtent trop cher à servir pour tenir dans un forfait fixe ; Anthropic prévoyait d'ailleurs de faire sortir Fable 5 des abonnements vers une facturation en crédits. Que ce modèle précis soit aujourd'hui suspendu ne change rien au principe : dès que vous passez par l'API ou par un modèle hors forfait, le compteur tourne, et la discipline de coûts du chapitre 12 devient incontournable.

> **LE SIGNAL D'ALARME DE LA COMMUNAUTÉ**
>
> Simon Willison, testeur indépendant parmi les plus suivis : 110 $ de tokens en une seule journée d'usage intensif de Fable 5 via Claude Code, dont une session à 99 $. Un modèle de pointe qui raisonne longtemps écrit beaucoup de tokens, et les coûts se multiplient vite. La leçon vaut pour Opus 4.8 comme pour tout modèle à venir : sachez ce que chaque usage coûte avant qu'il ne devienne une habitude.

La seconde force : la dépendance devient un risque opérationnel

Le 12 juin, une décision prise à Washington a coupé Fable 5 et Mythos 5 partout dans le monde, France comprise, en quelques heures. Toute organisation qui avait bâti un produit, un agent ou un workflow critique sur ces modèles s'est retrouvée à l'arrêt. Ce n'est plus seulement une question de prix ou de performance : c'est une question de continuité. Trois réflexes deviennent stratégiques.

1. **Cartographier votre dépendance**
   Quelles fonctions de votre activité s'arrêtent si tel modèle disparaît demain ? La réponse distingue l'usage confortable de la dépendance vitale — et vous dit où mettre un repli en priorité.

2 **Prévoir un modèle de repli**

Une architecture qui sait router vers un autre modèle — Opus 4.8 resté disponible, un modèle européen comme Mistral, un modèle ouvert hébergé en interne — transforme une coupure en simple changement de configuration. C'est exactement le routage du chapitre 12, étendu à la disponibilité.

3 **Relire la souveraineté sans posture**

« IA souveraine » n'est plus un slogan : c'est la garantie qu'une décision étrangère ne coupe pas votre outil de production. Cela ne veut pas dire tout rapatrier — cela veut dire savoir, pour chaque fonction critique, où bascule le repli.

Chiffrez ensuite, par cas d'usage : fréquence mensuelle × consommation mesurée × tarif du modèle retenu, repli compris. Vous obtenez un budget réel et une règle écrite (quel modèle pour quoi, quel plafond, quel repli). Sans règle écrite, la décision se prend par défaut — au fil des factures, ou au pire des coupures.

Installer la discipline budgétaire durable

Des plafonds partout. La console Anthropic permet de fixer des limites de dépense par clé API et des alertes par seuil ; les organisations attribuent des budgets par équipe ou par application. Faites-le dès le premier euro : un plafond qui n'existe pas ne protège personne — les grandes entreprises technologiques elles-mêmes ont dû instaurer des plafonds de budget IA par développeur.

Un coût par unité de valeur. La bonne métrique n'est pas « notre facture IA » mais « le coût IA par demande traitée, par article produit, par dossier instruit ». C'est elle qui dit si un flux s'améliore ou dérive, et elle rend les arbitrages lisibles pour un comité de direction.

Une revue mensuelle de routage. Trente minutes par mois : les dix flux les plus consommateurs, et pour chacun la question du chapitre 12 — un modèle moins cher tiendrait-il la qualité ? Les modèles progressent vite : la tâche qui exigeait Opus il y a six mois tourne peut-être sur Sonnet aujourd'hui.

OUTIL · LE SIMULATEUR OTIUM

Chiffrez votre cas en deux minutes

Nous avons mis en ligne un simulateur de consommation : type de projet, volume d'usage, et coût mensuel estimé pour chaque modèle du marché — Opus 4.8, GPT-5.5, Gemini 3.1 Pro, DeepSeek, Mistral — aux tarifs officiels. De quoi arbitrer modèle et repli, chiffres en main : **otium-ai.co/outils/simulateur-tokens**

LA PERSPECTIVE À GARDER

Ne laissez pas la discipline budgétaire masquer l'essentiel : aux tarifs de juin 2026, une heure de travail qualifié coûte encore des dizaines de fois plus cher que l'exécution IA de la même tâche bien routée. Le risque pour une organisation n'est pas de dépenser 200 $ de tokens par mois — c'est de ne pas savoir lesquels rapportent. Mesurez, routez, et investissez le reste dans ce que ce guide a décrit : des Skills, des connexions, des systèmes.

05

La boîte à outils

Tout ce qui précède explique. Cette partie équipe : la configuration complète de Claude Code, cinq automatisations à copier telles quelles, une bibliothèque de Skills prêtes à l'emploi, et les erreurs qui coûtent réellement de l'argent. Chaque page contient quelque chose à copier-coller.

14

Claude Code de A à Z : la configuration qui change tout

Entre l'utilisateur qui tape des demandes dans Claude Code et celui qui en tire dix fois plus, la différence n'est pas le talent : c'est la configuration. Ce chapitre est le condensé de ce que les utilisateurs intensifs mettent en place — commandes, modes, permissions, hooks, sous-agents — avec les fichiers exacts.

Les commandes à connaître par cœur

Commande	Ce qu'elle fait	Quand
`/init`	Génère un CLAUDE.md en analysant votre projet	Première ouverture d'un dépôt
`/clear`	Vide le contexte, repart à neuf	Entre deux tâches sans rapport — réflexe anti-coût n°1
`/compact`	Résume l'historique pour libérer du contexte	Longue session qu'on veut poursuivre
`/model`	Change de modèle en cours de session	Basculer Sonnet ↔ Opus ↔ Fable selon la tâche
`/resume`	Reprend une session précédente, contexte inclus	Le lendemain, sur le même chantier
`/agents`	Crée et gère les sous-agents	Mise en place des rôles (chapitre 8)
`/mcp`	État et gestion des serveurs MCP connectés	Diagnostic quand un outil ne répond pas
`/hooks`	Configure les hooks interactivement	Automatiser les garanties (chapitre 7)
`/install-github-app`	Branche Claude sur vos dépôts GitHub	Une fois par organisation (chapitre 6)
`#` **en début de message**	Mémorise l'instruction dans CLAUDE.md	Dès que vous corrigez Claude deux fois sur la même chose

Les trois modes d'exécution — et quand les utiliser

Mode normal. Claude demande confirmation avant chaque action sensible (écriture de fichier, commande). C'est le mode de découverte ; il devient vite un goulot.

Mode plan (Maj+Tab). Claude lit, analyse et propose un plan, sans rien modifier. C'est le mode le plus sous-utilisé et le plus rentable : sur toute tâche non triviale, exiger un plan d'abord évite la moitié des allers-retours. Validez le plan, puis laissez l'exécution se dérouler.

Auto-accept. Claude enchaîne les actions sans confirmations. À réserver aux tâches bien bornées sur un dépôt propre — et c'est exactement pour le rendre sûr que les permissions ci-dessous existent.

Le fichier de permissions : autoriser finement

Le fichier `.claude/settings.json`, versionné avec le projet, définit ce que Claude peut faire sans demander — et ce qu'il ne pourra jamais faire. C'est lui qui rend l'auto-accept raisonnable :

```
{
  "permissions": {
    "allow": [
      "Bash(npm run build)", "Bash(npm test:*)",
      "Bash(git diff:*)", "Bash(git log:*)",
      "Read(./**)", "Edit(./src/**)"
    ],
    "deny": [
      "Bash(rm -rf:*)", "Bash(git push:*)",
      "Read(./.env)", "Read(./secrets/**)",
      "WebFetch"
    ]
  }
}
```

Lecture : Claude peut builder, tester, lire le dépôt et éditer `src/` sans demander ; il ne peut ni pousser sur le dépôt distant, ni lire les secrets, ni supprimer récursivement — même si on le lui demande. La liste `deny` prime toujours.

Des hooks réels, pas théoriques

Le chapitre 7 a posé le principe ; voici un bloc complet à adapter. Trois garanties : formatage automatique après chaque modification, blocage des commandes dangereuses, notification de fin de session.

```
{
  "hooks": {
    "PostToolUse": [{
      "matcher": "Edit|Write",
      "hooks": [{ "type": "command",
        "command": "npx prettier --write \"$CLAUDE_FILE_PATHS\"" }]
    }],
    "PreToolUse": [{
      "matcher": "Bash",
      "hooks": [{ "type": "command",
        "command": "./scripts/verifier-commande.sh" }]
    }],
    "Stop": [{
      "matcher": "",
      "hooks": [{ "type": "command",
        "command": "osascript -e 'display notification \"Session terminee\"'" }]
    }]
  }
}
```

Le script `verifier-commande.sh` reçoit la commande proposée en JSON sur l'entrée standard et peut la bloquer en renvoyant un code de sortie 2 — avec un message que Claude lira et comprendra. Une dizaine de lignes de shell suffisent pour interdire définitivement `DROP TABLE`, `rm -rf` ou tout accès à la production.

La hiérarchie CLAUDE.md : trois étages de mémoire

Claude Code lit et fusionne jusqu'à trois fichiers d'instructions, du plus général au plus spécifique : `~/.claude/CLAUDE.md` (vos préférences personnelles, valables partout — ton, langue, style de code), le `CLAUDE.md` à la racine du projet (conventions d'équipe, versionné), et des `CLAUDE.md` de sous-dossiers pour les zones particulières (le dossier `migrations/` qui ne se modifie jamais, le module légaté qu'on ne touche qu'avec des pincettes). Le plus spécifique l'emporte. Cette hiérarchie est exactement celle d'une équipe bien managée : des principes généraux, des règles locales.

Sous-agents : le fichier complet

Un sous-agent = un fichier Markdown dans `.claude/agents/`. L'exemple ci-dessous est un relecteur de pull requests volontairement strict — à adapter à vos conventions :

```
---
name: relecteur-pr
description: Relit les modifications de code avant commit ou PR.
  A utiliser apres toute serie de changements significatifs.
tools: Read, Grep, Glob, Bash(git diff:*)
model: sonnet
---
Tu es relecteur senior. Pour chaque revue :
1. Lis le diff complet (git diff) avant tout commentaire
2. Cherche specifiquement : secrets en clair, injections,
   erreurs non gerees, mutations d'etat partage
3. Classe chaque remarque : BLOQUANT / IMPORTANT / STYLE
4. Pas de remarque de style s'il y a un BLOQUANT non resolu
5. Termine par un verdict : PRET A MERGER ou CORRECTIONS REQUISES
```

Deux détails qui comptent : la ligne `tools` limite l'agent à la lecture (il ne peut pas « corriger en passant » — c'est voulu, un relecteur ne touche pas au code), et la ligne `model: sonnet` route ce travail récurrent vers un modèle trois fois moins cher que celui de la session principale.

Le mode non interactif : Claude comme commande Unix

La clé de toutes les automatisations du chapitre suivant : `claude -p` exécute une consigne et rend la main, comme n'importe quel programme. Combiné à `--output-format json`, il s'insère dans un pipeline :

```
# Resume du jour, en JSON exploitable par le script suivant
claude -p "Analyse les erreurs dans logs/app.log et resume les 3 plus frequentes" \
  --output-format json --max-turns 10 \
  --allowedTools "Read,Grep" > rapport.json
```

`--max-turns` borne le nombre d'actions (votre plafond de coût par exécution), `--allowedTools` borne les capacités. Ces deux drapeaux transforment un agent ouvert en brique d'automatisation prévisible.

> **L'ORDRE D'INSTALLATION QUI MARCHE**
>
> Jour 1 : `/init` puis enrichir CLAUDE.md à la main. Jour 2 : permissions (allow/deny) pour passer en auto-accept serein. Semaine 2 : premier hook (formatage), premier sous-agent (relecture). Semaine 3 : premier `claude -p` planifié. Chaque étape s'appuie sur la précédente — et à la fin, votre configuration vaut plus que n'importe quelle formation.

15

Des automatisations prêtes à copier

Des recettes complètes, du déclencheur au livrable, représentatives de ce que nous montons. Ce ne sont pas les seules : n8n à lui seul publie plus de 2 300 templates prêts à importer (voir les dépôts en fin de chapitre). Les coûts indiqués sont calculés aux tarifs API officiels du 12 juin 2026.

Recette 1 — La revue de presse qui vous attend au réveil

Besoin : une veille quotidienne sur vos sources, livrée en Markdown à 7h30. **Briques :** cron + Claude Code headless. **Mise en place :** 30 minutes.

```
# 1. Creer veille/sources.txt : une URL de flux RSS par ligne
# 2. crontab -e, puis :
30 7 * * 1-5  cd ~/veille && claude -p "Lis les flux listes dans \
  sources.txt (WebFetch). Selectionne les 5 infos les plus utiles \
  pour [VOTRE CONTEXTE]. Pour chacune : titre, 3 lignes, lien, \
  pourquoi ca nous concerne. Ecris dans brief-$(date +\%F).md" \
  --allowedTools "Read,Write,WebFetch" --max-turns 25
```

Coût : avec Sonnet 4.6, environ 0,15 $ par matin — 3 $ par mois. **Amélioration v2 :** ajouter en fin de consigne « compare avec les briefs des 5 derniers jours et signale ce qui est nouveau » — c'est là que l'agent dépasse l'agrégateur.

Recette 2 — La revue automatique de chaque pull request

Besoin : chaque PR reçoit une revue structurée en 2 minutes. **Briques :** GitHub Actions + Claude Code Action officielle. **Mise en place :** 15 minutes.

```yaml
# .github/workflows/revue-claude.yml
name: Revue Claude
on:
  pull_request:
    types: [opened, synchronize]
jobs:
  revue:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      pull-requests: write
    steps:
      - uses: actions/checkout@v4
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          prompt: |
            Relis cette PR. Cherche : bugs probables, failles
            (injection, secrets, validation manquante), ecarts aux
            conventions du CLAUDE.md. Classe : BLOQUANT / IMPORTANT /
            STYLE. Commente uniquement ce qui merite action.
```

Coût : 0,05 à 0,30 $ par PR selon la taille du diff, avec Sonnet. Une équipe qui fusionne 60 PR par mois paie moins de 15 $ — le prix d'une demi-heure de relecture humaine.

Recette 3 — Les demandes entrantes triées et pré-répondues

Besoin : chaque email entrant générique (contact@, info@) est qualifié, enregistré, et un brouillon de réponse attend dans la boîte. **Briques :** n8n. **Chaîne de nœuds :**

1. **Gmail Trigger**

   Sur nouveau message dans le label « Entrant ». Filtre : exclure les expéditeurs internes.

2. **Nœud Anthropic — extraction**

   Modèle Haiku 4.5, sortie JSON imposée : `{intention, urgence (1–3), entreprise, budget_mentionne, resume_2_lignes}`. Le point clé : exiger du JSON avec un schéma, jamais du texte libre entre deux nœuds.

3. **IF — aiguillage**

   Urgence 3 ou budget mentionné → notification Slack immédiate + ligne CRM. Sinon → flux standard.

4. **Nœud Anthropic — brouillon**

   Modèle Sonnet 4.6, avec votre Skill de réponse en contexte système (même texte que votre Skill claude.ai — une seule source de vérité).

5 **Gmail — créer un brouillon**

Jamais d'envoi direct : le brouillon attend votre validation. C'est la règle du valideur humain (chapitre 11), câblée dans le flux.

Coût : ~0,025 $ par demande traitée (tri Haiku + brouillon Sonnet). 500 demandes par mois : 12,50 $.

Recette 4 — Le rapport hebdomadaire qui se compile seul

Besoin : chaque vendredi 17h, une synthèse d'activité croise CRM, support et facturation. **Briques :** n8n planifié (ou cron) + serveurs MCP internes en lecture + Claude API. **Le cœur de la recette** est le prompt de synthèse — le voici, éprouvé :

```
Tu compiles le rapport hebdomadaire. Donnees jointes : export CRM
(JSON), tickets support (JSON), facturation (CSV).
Structure imposee :
1. CHIFFRE DE LA SEMAINE — le seul a retenir, avec contexte
2. PIPELINE — entrees, sorties, blocages > 14 jours
3. SUPPORT — volume, theme dominant, tickets > 48 h sans reponse
4. ALERTES — tout ecart > 20 % vs moyenne 4 semaines
Regles : aucun chiffre invente — si une donnee manque, ecris
"DONNEE MANQUANTE". Chaque alerte propose une action. 1 page max.
```

La ligne « DONNÉE MANQUANTE » n'est pas décorative : c'est elle qui rend le rapport digne de confiance. Un modèle à qui on interdit explicitement de combler les trous signale les trous — et un trou signalé vaut mieux qu'un chiffre plausible.

Recette 5 — L'enrichissement de base en nocturne, à moitié prix

Besoin : qualifier ou enrichir un stock de fiches (prospects, produits, contenus) sans urgence. **Brique :** l'API Batch — moitié prix, résultat sous 24 h. **Le code complet :**

```
import anthropic, json
client = anthropic.Anthropic()

fiches = json.load(open("prospects.json"))
batch = client.messages.batches.create(requests=[{
    "custom_id": f["id"],
    "params": {
        "model": "claude-haiku-4-5",
        "max_tokens": 300,
        "system": [{"type": "text",
            "text": GRILLE_DE_QUALIFICATION,        # texte stable...
            "cache_control": {"type": "ephemeral"}  # ...donc en cache
        }],
        "messages": [{"role": "user", "content": json.dumps(f)}],
    }} for f in fiches])
print(batch.id)  # recuperer les resultats sous 24 h
```

Le cumul des remises : Haiku (modèle à 1 $/MTok) × batch (−50 %) × cache sur la grille (−90 % sur le contexte répété) : qualifier **10 000 fiches** de 1 500 tokens revient à environ **4 $**. C'est le chiffre à retenir de ce chapitre : au bon étage de la gamme, le travail de volume ne coûte presque rien.

Recette 6 — La chaîne de production d'un reel, de bout en bout

Besoin : un format vidéo court récurrent (récap, témoignage, annonce) produit sans monteur. **Briques :** n8n + Claude + génération média (chapitre 9). **Chaîne de nœuds :**

1. **Déclencheur**
   Horaire (chaque vendredi) ou webhook depuis votre CMS quand un sujet est validé.

2. **Claude — script**
   Sonnet 4.6 avec votre Skill de charte : titre, 4 à 6 plans, texte de voix off, sortie JSON structurée (un objet par plan).

3. **ElevenLabs — voix**
   La voix off générée depuis le texte, voix de marque fixée.

4. **Image puis vidéo**
   Nano Banana Pro produit les visuels de chaque plan, Seedance ou Kling les anime à partir de ces images.

5. **Assemblage + validation**
   Montage automatique (sous-titres, logo, musique) déposé dans un dossier ; un humain valide avant publication.

Règle d'or : la sortie JSON structurée du nœud Claude (un objet par plan) est ce qui permet aux nœuds suivants de boucler proprement sur chaque scène. Texte libre = chaîne qui casse.

Recette 7 — Le vault de connaissances qui s'alimente seul

Besoin : capitaliser automatiquement décisions, comptes rendus et veille dans une base (Obsidian, Notion). **Briques :** n8n + serveur de fichiers MCP + Claude. **Chaîne :**

```
# Pseudo-flux n8n
Declencheur (fin de reunion / mail / note vocale)
  -> Claude (Skill compte-rendu-reunion) : decisions + actions, JSON
  -> Ecriture fichier MCP : vault/sessions/2026-06-12.md
  -> Mise a jour de l'index : append une ligne dans vault/log.md
  -> Notification : resume dans Slack
```

Le vault devient la mémoire vivante de l'équipe : chaque agent le lit avant d'agir (contexte frais via MCP) et l'enrichit après. C'est la différence entre une organisation qui réexplique tout à chaque fois et une organisation qui se souvient.

Ne partez pas de zéro : les bibliothèques de workflows

Ces recettes sont des points de départ — il en existe des milliers, déjà construits. Avant d'écrire un flux, cherchez-le :

> **OÙ TROUVER DES WORKFLOWS PRÊTS**
>
> **n8n.io/workflows** — plus de 2 300 templates officiels, par catégorie (ventes, support, contenu, data), importables en un clic
>
> **github.com/czlonkowski/n8n-mcp** — le MCP qui laisse Claude construire, tester et corriger vos workflows n8n en langage naturel (1 851 nœuds, 2 352 templates indexés)
>
> **github.com/anthropics/claude-cookbook** — recettes d'agents et de chaînes côté API, prêtes à exécuter

```
# Laisser Claude Code piloter n8n directement
claude mcp add n8n -- npx -y n8n-mcp
```

Branché ainsi, Claude lit votre instance n8n, propose un workflow à partir d'un template, le crée et le teste — vous décrivez le besoin, il assemble les nœuds. La bibliothèque fait le gros ; vous gardez l'adaptation et la validation.

LE POINT COMMUN DE CES RECETTES

Aucune ne demande plus d'une journée de mise en place, aucune n'envoie quoi que ce soit à l'extérieur sans validation humaine, et toutes bornent le modèle (outils limités, tours limités, schéma de sortie imposé). C'est la signature d'une automatisation qui vivra plus de trois semaines.

16

La bibliothèque de Skills prêtes à l'emploi

Avant d'écrire les vôtres, sachez qu'il existe déjà des centaines de Skills prêtes à installer — open source ou intégrées. En voici un panorama réel, regroupé par usage, pour mesurer l'étendue de ce qui se branche en une commande.

Domaine	Skills disponibles (exemples réels)
Développement	`python-patterns`, `golang-patterns`, `rust-patterns`, `springboot-patterns`, `laravel-patterns`, `django-patterns`, `api-design`, `tdd-workflow`, `code review`, `security-review`
IA & agents	`mcp-builder`, `mcp-server-patterns`, `claude-api`, `agent-harness-construction`, `eval-harness`, `prompt-optimizer`, `cost-aware-llm-pipeline`, `token-budget-advisor`
Automatisation	`n8n-workflow-patterns`, `n8n-node-configuration`, `n8n-code-javascript`, `n8n-mcp-tools-expert`
Contenu & marketing	`article-writing`, `content-engine`, `crosspost`, `humanizer`, `market-research`, `deep-research`, `brand-guidelines`
Design	`frontend-design`, `design-consultation`, `design-review`, `canvas-design`, `theme-factory`
Documents	`pdf`, `docx`, `pptx`, `xlsx`, `document-release`
Métiers & opérations	`production-scheduling`, `inventory-demand-planning`, `logistics-exception-management`, `quality-nonconformance`, `customs-trade-compliance`
Direction & méthode	`blueprint`, `plan-ceo-review`, `plan-eng-review`, `retro`, `skill-creator`, `verification-loop`

Échantillon d'un écosystème de plus de 250 Skills publiques au 12 juin 2026. `skill-creator` sert à générer les vôtres ; `configure-ecc` à les installer.

Où trouver ces Skills : les dépôts GitHub

La plupart de ces Skills et méthodes sont open source et s'installent depuis GitHub, via le gestionnaire de plugins de Claude Code (`/plugin marketplace add`) ou en clonant le dépôt dans `.claude/skills/`. Les sources de référence :

```
# Installer un catalogue complet (250+ skills) :
/plugin marketplace add affaan-m/everything-claude-code
/plugin install ecc@ecc

# Ou une marketplace ciblee (methodes) :
/plugin marketplace add obra/superpowers
/plugin install superpowers@superpowers-dev
```

> **DÉPÔTS DE RÉFÉRENCE (JUIN 2026)**
>
> **github.com/affaan-m/everything-claude-code** — ECC : plus de 250 Skills, agents, règles et hooks (82k ★). `/plugin install ecc@ecc`
>
> **github.com/obra/superpowers** — méthodes : brainstorming, TDD, débogage, plans, revue de code
>
> **github.com/anthropics/skills** — Skills officielles Anthropic (Office, documents, agents)
>
> **github.com/anthropics/claude-cookbook** — recettes de code prêtes à l'emploi
>
> **github.com/thedotmack/claude-mem** — mémoire persistante pour Claude Code

Côté outillage : `skill-creator` génère le squelette d'une nouvelle Skill à partir d'une description, et `configure-ecc` installe et organise un lot de Skills au niveau projet ou machine. Vous n'êtes jamais obligé de partir de la page blanche.

Les frameworks et méthodes qui s'installent comme des Skills

Au-delà des Skills unitaires, des **méthodes complètes** se distribuent de la même manière. La famille *superpowers* (brainstorming structuré, TDD, débogage systématique, écriture de plans, revue de code) impose une discipline de travail à l'agent. Des frameworks de planification produit type **BMAD** (PRD, architecture, user stories, revues éditoriales) transforment Claude en chef de projet. Les revues croisées (`plan-ceo-review`, `plan-eng-review`, `plan-design-review`) font passer un plan par trois regards avant exécution. Installer une méthode, c'est importer une façon de travailler éprouvée — pas seulement un outil.

La base de connaissances comme mémoire d'équipe

Un cas qui revient dans presque tous nos déploiements : brancher Claude sur une base de connaissances (**Obsidian**, Notion, Confluence) via un serveur de fichiers MCP ou un connecteur. Le vault devient à la fois la source que Claude consulte et l'endroit où il consigne — comptes rendus, décisions, sessions de travail datées. La mémoire de l'organisation cesse de vivre dans les têtes : elle devient un dossier interrogeable, versionné, que chaque agent lit avant d'agir et enrichit après.

Voici des Skills complètes, à copier dans claude.ai (paramètres → capacités) ou dans `.claude/skills/`. Ce sont des modèles — pas un catalogue fermé : observez *comment* elles sont écrites, puis branchez les centaines déjà disponibles via les dépôts plus bas. La qualité d'une Skill tient à sa précision et à ses interdits, jamais à leur nombre.

1. compte-rendu-reunion

```
---
name: compte-rendu-reunion
description: A utiliser pour transformer des notes ou une transcription
  de reunion en compte rendu actionnable.
---
A partir des notes fournies, produis :
1. DECISIONS — uniquement ce qui a ete tranche. Une ligne chacune.
2. ACTIONS — tableau : action / responsable / echeance.
   Si un responsable ou une echeance manque : "[A ATTRIBUER]" —
   ne devine jamais.
3. POINTS OUVERTS — ce qui a ete discute sans etre tranche.
4. HORS SUJET UTILE — idees evoquees a ne pas perdre.
Regles : aucune phrase de remplissage ("la reunion s'est bien
passee"). Pas de resume chronologique. Si les notes sont trop
maigres pour distinguer decision et discussion, dis-le.
```

2. veille-concurrentielle

```
---
name: veille-concurrentielle
description: A utiliser pour analyser un concurrent, son site, ses
  annonces ou ses changements de prix.
---
Pour chaque element analyse :
1. FAIT — ce qui est observable, avec la source exacte
2. LECTURE — ce que ca suggere de leur strategie (marque "hypothese")
3. IMPACT POUR NOUS — concret : offre, prix, discours, rien d'autre
4. ACTION — une seule, ou "AUCUNE ACTION REQUISE" si c'est du bruit
Regles : distinguer fait/hypothese systematiquement. Pas de
"il faut surveiller" — soit une action, soit rien. Comparer avec
la veille precedente si fournie : signaler le NOUVEAU uniquement.
```

3. relance-par-paliers

```
---
name: relance-par-paliers
description: A utiliser pour rediger une relance client ou prospect
  (devis sans reponse, facture, dossier en attente).
---
Demande d'abord : quel palier ? (1 = rappel leger, 2 = relance
ferme, 3 = derniere relance avant cloture)
Palier 1 : pretexte utile (information nouvelle, precision) —
  jamais "je me permets de revenir vers vous".
Palier 2 : factuel — rappel de l'engagement, date, question fermee
  appelant un oui/non.
Palier 3 : cloture annoncee sans menace — "sans retour au [date],
  je classe le dossier" + porte ouverte.
Regles : 6 phrases maximum. Une seule question par message.
Jamais de culpabilisation ("malgre mes nombreuses relances").
```

4. analyse-candidature

```
---
name: analyse-candidature
description: A utiliser pour analyser un CV ou une candidature par
  rapport a une fiche de poste.
---
Entrees requises : le CV + la fiche de poste. Sans fiche de poste,
refuse : "l'analyse sans referentiel produit des impressions".
Sortie :
1. CORRESPONDANCES — exigence par exigence, preuve a l'appui
   (experience citee, pas supposee)
2. ECARTS — ce que le CV ne demontre pas (different de "ne sait pas")
3. QUESTIONS D'ENTRETIEN — 5, ciblees sur les ecarts et les
   transitions de carriere inexpliquees
4. CE QUE CE CV NE DIT PAS — rappel : motivation, savoir-etre et
   contexte de depart ne se lisent pas dans un CV.
Interdit : score global, recommandation embaucher/rejeter.
L'outil eclaire, l'humain decide.
```

5. reponse-avis-clients

```
---
name: reponse-avis-clients
description: A utiliser pour repondre aux avis en ligne (Google,
  Trustpilot, stores).
---
Avis positif : remercier en reprenant UN detail specifique de
l'avis (preuve de lecture), pas de formule generique, pas de
relance commerciale.
Avis negatif :
1. Reconnaitre le probleme precis — jamais "nous sommes desoles
   que vous ayez ressenti cela" (deni deguise)
2. Si erreur averee : le dire. Si contexte manquant : le demander
   en prive, sans contester en public
3. Une action concrete, un canal direct
Regles : 4 phrases max, signer d'un prenom, aucune reponse
copiee-collee entre deux avis. Ne jamais contester les faits en
public, meme si le client a tort.
```

6. brief-createur

```
---
name: brief-createur
description: A utiliser pour transformer une demande floue en brief
  exploitable (design, video, redaction, dev).
---
Transforme la demande en brief structure :
1. OBJECTIF — une phrase, mesurable si possible
2. AUDIENCE — qui voit/utilise le livrable, dans quel contexte
3. CONTRAINTES — format, dimensions, charte, deadline, budget
4. REFERENCES — ce qui plait et POURQUOI (le pourquoi est la
   partie utile)
5. ANTI-REFERENCES — ce qu'on ne veut surtout pas
6. CRITERES D'ACCEPTATION — comment on saura que c'est reussi
Pour chaque rubrique sans information : pose UNE question fermee.
Maximum 4 questions par brief — au-dela, c'est un atelier qu'il
faut, pas un brief.
```

7. reporting-mensuel

```
---
name: reporting-mensuel
description: A utiliser pour analyser des chiffres mensuels (ventes,
  tresorerie, marketing) et en tirer un commentaire de gestion.
---
Entrees : les chiffres du mois + ceux des 6 mois precedents minimum.
Avec moins de 3 mois d'historique, refuse l'analyse de tendance.
Sortie :
1. EN UNE PHRASE — le mois en une phrase factuelle
2. ECARTS — tout ce qui sort de +/- 15 % vs moyenne 6 mois,
   avec le chiffre exact et l'ecart en %
3. EXPLICATIONS CANDIDATES — pour chaque ecart : les causes
   possibles A VERIFIER (marquees comme hypotheses)
4. DECISION SUGGEREE — une par ecart majeur, ou "SURVEILLER"
Interdits : lisser les mauvaises nouvelles, moyenner ce qui
masque un probleme, conclure sur une cause sans la marquer
"hypothese". Les chiffres absents sont signales, jamais estimes.
```

8. brief-seo

```
---
name: brief-seo
description: A utiliser pour preparer le brief d'un article ou d'une
  page visant une requete de recherche.
---
Entree : la requete visee + l'audience.
Sortie :
1. INTENTION — ce que cherche vraiment la personne qui tape ca
   (info, comparaison, achat, action) et ou elle en est
2. ANGLE — ce que les contenus existants ne traitent pas ou mal
3. STRUCTURE — H2/H3 proposes, avec la question que chaque
   section doit fermer
4. PREUVES REQUISES — chiffres, sources, exemples a reunir AVANT
   d'ecrire. Un brief sans preuves a reunir = un futur article creux
5. MAILLAGE — pages internes a lier, dans les deux sens
Interdits : promettre une position, bourrer de mots-cles,
proposer un titre putaclic que le contenu ne tient pas.
```

9. onboarding-client

```
---
name: onboarding-client
description: A utiliser au demarrage d'une mission ou d'un nouveau
  client pour structurer la collecte d'informations.
---
A partir du contexte fourni (proposition signee, echanges) :
1. CE QUE NOUS SAVONS — synthese des infos deja disponibles,
   avec leur source
2. CE QUI MANQUE — classe en : bloquant pour demarrer /
   necessaire sous 15 jours / utile plus tard
3. QUESTIONNAIRE — questions fermees pour les bloquants
   uniquement, groupees par interlocuteur probable
4. ACCES A DEMANDER — outils, documents, contacts, avec le
   niveau de droit minimal necessaire
5. PREMIERE ECHEANCE — le livrable qui cree la confiance en
   moins de 15 jours
Regle : ne jamais redemander une information presente dans les
documents fournis — c'est le signal qui ruine un demarrage.
```

> **CE QUE CES NEUF SKILLS ONT EN COMMUN**
>
> Relisez-les : chacune contient des **interdits explicites** (la moitié de la valeur), des **sorties structurées** en sections nommées, un **mécanisme anti-invention** (« [À ATTRIBUER] », « DONNÉE MANQUANTE », refus si entrée insuffisante) et une **limite de format**. Quatre ingrédients, toujours les mêmes. Une Skill sans interdits est une liste de vœux.

17

Les erreurs qui coûtent cher, et la FAQ

Tout ce qui précède dit quoi faire. Cette dernière partie dit quoi arrêter de faire — douze erreurs observées de manière récurrente, puis les questions qui reviennent dans chaque déploiement.

Les douze erreurs récurrentes

L'erreur	Le correctif
La conversation-fleuve de 200 messages qui mélange six sujets	`/clear` entre les sujets. Le contexte accumulé coûte à chaque tour et dégrade les réponses.
Tout faire tourner sur le modèle le plus puissant « pour être tranquille »	Routage (chapitre 12) : tester d'abord le modèle le moins cher. Écart réel : facteur 10 à 50.
Réexpliquer ses formats à chaque conversation	Une Skill par format (chapitre 16). Si vous l'avez expliqué deux fois, c'est une Skill.
Corriger la même erreur de Claude Code semaine après semaine	`#` en début de message : la correction entre dans CLAUDE.md pour toujours.
Automatiser une tâche jamais testée manuellement	Règle des trois occurrences (chapitre 7). Un défaut automatisé s'exécute 200 fois par mois.
Agent « à tout faire » avec accès à tout	Une mission, des outils minimaux, des limites chiffrées (chapitre 8).
Texte libre entre deux nœuds d'automatisation	JSON avec schéma imposé, partout. Le texte libre casse silencieusement.
Envoi externe sans validation humaine « parce que ça marchait en test »	Brouillon + valideur nommé sur tout ce qui sort (chapitre 11). Sans exception.
Coller un export de base dans la conversation	Serveur MCP en lecture seule (chapitre 5) : accès frais, périmètre contrôlé, zéro copie qui traîne.
Documents entiers joints « au cas où »	Joindre ce que la question exige. Le reste coûte et dilue.
Pas de plafond de dépense « on verra la facture »	Limites par clé API + alertes dès le premier euro (chapitre 13).
Juger l'IA sur une impression (« ça a l'air bien »)	Vingt cas réels avec résultat attendu, mesurés avant et après chaque changement (chapitre 8).

La FAQ des déploiements

Mes données servent-elles à entraîner les modèles ?

Sur les offres Team, Enterprise et l'API : non, par défaut contractuel. Sur les offres grand public, un réglage explicite contrôle l'usage des conversations pour l'amélioration des modèles — vérifiez-le dans les paramètres de confidentialité. Cas particulier de juin 2026 : les modèles classe

Mythos (dont Fable 5) impliquent une rétention de sécurité de 30 jours, avec engagement écrit d'Anthropic que ces données ne servent pas à l'entraînement.

Abonnement ou API : comment choisir ?

Règle simple : les humains sur abonnement, les machines sur API. Un abonnement Pro ou Max couvre largement un usage individuel intensif à prix fixe. Dès qu'un traitement tourne sans humain (automatisation, agent, volume), l'API s'impose — c'est elle qui offre plafonds, suivi par clé et choix fin du modèle. La plupart des organisations finissent avec les deux, et c'est normal.

Que mettre en place en premier avec un budget de zéro euro ?

Trois Skills sur vos formats les plus fréquents, et la mémoire activée. Coût : une demi-journée. C'est l'étape au meilleur rendement de tout ce guide — avant les connecteurs, avant MCP, avant les agents.

Faut-il attendre que les modèles se stabilisent ?

L'argument revient à chaque génération depuis trois ans, et chaque génération lui donne tort. Ce qui se périme, ce sont les modèles ; ce que vous construisez — Skills, CLAUDE.md, serveurs MCP, jeux d'évaluation, réflexes d'équipe — survit aux changements de modèle et prend de la valeur à chacun. Attendre, c'est différer l'actif, pas le risque.

Comment convaincre une équipe réticente ?

Pas par la démonstration — par le problème. Prenez la tâche que l'équipe déteste le plus (le compte rendu, le tri, la relance), équipez-la, mesurez, et laissez le résultat circuler. Les déploiements qui commencent par « l'IA va transformer notre métier » braquent ; ceux qui commencent par « plus personne ne rédige les comptes rendus à la main » font des envieux.

Et si Anthropic change ses prix ou ses produits ?

C'est une certitude, pas un risque — la suspension de Fable 5 le 12 juin le prouve. La protection n'est pas de deviner les décisions futures : c'est l'architecture de ce guide. Un système routé (chapitre 12), mesuré (coût par unité de valeur), et construit sur des standards ouverts (MCP) se rebranche ailleurs en jours, pas en mois. La dépendance se gère par la portabilité, pas par l'abstinence.

LE MOT DE LA FIN

Si vous ne retenez que trois gestes de ce guide : écrivez vos trois premières Skills cette semaine, testez Fable 5 sur un cas lourd avant le 22 juin en notant la consommation, et fixez vos plafonds de dépense avant le 23. Le reste — MCP, agents, automatisations — viendra dans l'ordre où vos besoins l'exigeront. L'IA récompense ceux qui l'équipent.

ANNEXES

A

Glossaire : parler couramment l'écosystème Claude

Agent	Système qui poursuit un objectif en enchaînant des actions (lire, chercher, exécuter, vérifier) en boucle, jusqu'au résultat ou jusqu'à une limite fixée.
Agent SDK	Bibliothèque officielle d'Anthropic (Python, TypeScript) qui fournit l'infrastructure de Claude Code pour construire ses propres agents.
Artefact	Document, visuel ou petite application générés par Claude dans un panneau dédié de claude.ai, modifiables par itérations.
Batch (API)	Mode de traitement différé (sous 24 h) facturé moitié prix. Pour tout ce qui peut attendre.
Cache de prompt	Mécanisme qui évite de payer plusieurs fois la lecture d'un même contexte : la relecture coûte dix fois moins cher.
Claude Code	L'agent d'Anthropic dans le terminal, l'éditeur, le bureau et le web. Lit, écrit, exécute et vérifie — pour le code et au-delà.
CLAUDE.md	Fichier d'instructions à la racine d'un projet, lu automatiquement par Claude Code : conventions, commandes, interdits.
Connecteur	Branchement d'une application (Gmail, Drive, Notion…) dans claude.ai, fondé sur MCP, avec autorisation révocable.
Contexte (fenêtre de)	Tout ce que le modèle « voit » à un instant donné : instructions, documents, historique. Fable 5 : un million de tokens.
Cowork	L'espace de travail agentique de Claude pour les équipes métier : des dossiers entiers traités en autonomie supervisée.
Fable 5	Le modèle classe Mythos rendu public le 9 juin 2026, équipé de garde-fous avec bascule vers Opus 4.8 sur les sujets sensibles.
Hook	Commande exécutée automatiquement à un moment précis du travail de Claude Code. Une garantie, pas une consigne.
MCP	Model Context Protocol : le standard ouvert qui connecte les IA aux outils et données. La prise universelle de l'écosystème.
Mythos 5	La frontière interne d'Anthropic, non commercialisée, réservée aux partenaires vérifiés du programme Glasswing.
Projet (claude.ai)	Espace qui regroupe conversations, fichiers et instructions permanentes autour d'un sujet.
Routage de modèles	Affecter chaque tâche au modèle le moins cher qui la réussit. Le premier levier d'optimisation.

Serveur MCP	Programme qui expose des outils, ressources et gabarits à une IA via le protocole MCP. Distant (hébergé) ou local.
Skill	Dossier d'instructions (fichier SKILL.md, références, scripts) que Claude charge automatiquement quand la tâche s'y prête.
Sous-agent	Agent spécialisé défini dans Claude Code (fichier Markdown), auquel l'agent principal délègue les tâches correspondantes.
System card	Document technique publié avec un modèle : capacités, limites, évaluations de sécurité. Celle de Fable 5 fait 319 pages.
Token	Unité de découpe du texte (≈ trois quarts d'un mot). L'unité de compte de toute facturation IA.

B

La checklist de mise en route

Tout ce guide en une page. Cochez dans l'ordre — chaque bloc s'appuie sur le précédent.

Semaine 1 — Le socle

- [ ] Choisir la surface adaptée à chaque usage récurrent (chapitre 2)
- [ ] Activer les connecteurs essentiels : messagerie, agenda, documents (chapitre 4)
- [ ] Créer un Projet par dossier de fond, avec instructions permanentes
- [ ] Tester Fable 5 sur vos cas lourds pendant la fenêtre incluse — mesurer la consommation (chapitre 13)

Semaines 2-3 — Les méthodes

- [ ] Écrire vos trois premières Skills à partir de vos meilleurs livrables (chapitre 3)
- [ ] Pour les équipes techniques : CLAUDE.md, app GitHub, revue automatique en CI (chapitre 6)
- [ ] Brancher les serveurs MCP de vos outils principaux — en lecture d'abord (chapitre 5)

Semaines 4-6 — Les systèmes

- [ ] Automatiser la première tâche validée trois fois manuellement (chapitre 7)
- [ ] Créer votre premier agent spécialisé, avec mission unique et garde-fous (chapitre 8)
- [ ] Monter la stack de votre profil (chapitre 10) — sans ajouter d'outil non justifié

En continu — Le pilotage

- [ ] Router chaque flux vers le modèle le moins cher qui tient la qualité (chapitre 12)
- [ ] Activer cache de prompt et traitement par lots sur les flux en volume
- [ ] Fixer plafonds, alertes de dépense et modèle de repli (chapitre 13)
- [ ] Revue mensuelle : coût par unité de valeur, routage, flux à arrêter
- [ ] Valideur humain nommé sur toute production qui sort de l'organisation (chapitre 11)

C

Le plan des 30 premiers jours

La checklist de l'annexe B dit quoi faire ; ce plan dit quand, pour une organisation qui part de zéro ou presque. Une demi-journée de travail effectif par semaine suffit.

Jours	Actions	Livrable vérifiable
1-2	Choisir le périmètre pilote (une équipe, deux cas d'usage — chapitre 11). Activer la mémoire et les conversations d'équipe. Écrire la règle des données : ce qui peut transiter, ce qui ne peut pas.	Une page : périmètre, deux cas, règle des données, valideur nommé
3-5	Première Skill sur le format le plus fréquent du pilote, construite à partir des cinq meilleurs exemples existants (chapitre 3). Test sur trois cas réels, deux itérations.	Skill v1 qui produit un premier jet jugé « utilisable » par son futur utilisateur
6-10	Connecteurs messagerie + agenda + documents pour le pilote (chapitre 4), droits minimaux. Deuxième et troisième Skills. Pour les profils techniques : `/init` sur le dépôt principal, premier CLAUDE.md enrichi.	La préparation de rendez-vous fonctionne de bout en bout sur un cas réel
11-15	Constituer le jeu d'évaluation : vingt cas réels avec résultat attendu (chapitre 8). Mesurer les Skills dessus. Première revue de consommation : qui consomme quoi, sur quel modèle.	Tableau : 20 cas, taux de réussite, modèle utilisé, coût unitaire
16-20	Première automatisation — celle validée trois fois manuellement (recettes du chapitre 15). Brouillons uniquement, jamais d'envoi direct. Plafonds et alertes de dépense sur chaque clé API.	L'automatisation tourne depuis 3 jours sans intervention corrective
21-25	Serveur MCP en lecture sur la source de données la plus demandée (chapitre 5) — ou connecteur équivalent. Former deux relais internes sur les cas réels de leur équipe, pas sur des démos.	Une question métier obtient sa réponse depuis la vraie base, sans export
26-30	Bilan chiffré : temps gagné, taux de reprise humaine, coût par unité traitée (chapitre 11). Décision : industrialiser, corriger, ou arrêter. Choisir les deux cas suivants — demandés par les équipes, pas imposés.	Bilan d'une page diffusé en interne, avec les trois chiffres

LE PIEGE DU JOUR 31

Le risque n'est pas l'échec du pilote — c'est sa dilution : trois mois plus tard, dix outils testés, rien d'industrialisé. Le bilan du jour 30 doit se conclure par une décision écrite, même si c'est « on arrête ». Un pilote sans verdict est un coût sans leçon.

D

Modèle de politique IA interne

La plupart des organisations n'ont besoin ni d'une charte de quarante pages ni d'un comité — mais d'une page claire que tout le monde a lue. Voici un modèle à adapter, volontairement court. Supprimez ce qui ne vous concerne pas ; n'ajoutez que ce que vous ferez vraiment respecter.

```
POLITIQUE D'USAGE DE L'IA — [ORGANISATION] — v1, [DATE]

1. OUTILS AUTORISES
   Claude (offre [Team/Enterprise]) pour tout usage professionnel.
   Tout autre outil IA : accord prealable de [ROLE].
   Les comptes personnels gratuits sont interdits pour les
   donnees de l'organisation.

2. DONNEES
   Peuvent transiter par l'IA : documents internes non sensibles,
   contenus publics, donnees clients NECESSAIRES a la tache.
   Ne transitent jamais : donnees de sante, paie, mots de passe
   et cles, donnees sous NDA explicite, [VOS AJOUTS].
   En cas de doute : demander a [ROLE] avant, pas apres.

3. VALIDATION
   Toute production IA qui SORT de l'organisation (mail client,
   devis, contrat, publication, code en production) a un valideur
   humain nomme. L'IA produit, quelqu'un signe.
   Le valideur engage sa responsabilite comme s'il etait l'auteur.

4. TRANSPARENCE
   En interne : pas d'obligation de mentionner l'usage de l'IA.
   En externe : ne jamais nier l'usage de l'IA si la question
   est posee. [Si secteur reglemente : obligations specifiques.]

5. COUTS
   Budgets et plafonds par equipe definis par [ROLE].
   Tout besoin depassant le plafond : demande motivee, pas de
   contournement par compte personnel.

6. ERREURS
   Une erreur de l'IA validee par un humain est une erreur
   humaine : on la corrige, on enrichit la Skill ou la consigne,
   on ne cherche pas de coupable. Signaler vaut toujours mieux
   que masquer.

Cette politique est revue tous les 6 mois. Contact : [ROLE].
```

Deux choix délibérés dans ce modèle : la règle 3 (le valideur engage sa responsabilité) est ce qui empêche la validation-tampon où l'on clique « ok » sans lire ; la règle 6 est ce qui fait re-

monter les problèmes au lieu de les enterrer. Une politique punitive produit des usages cachés — c'est le pire des deux mondes : les risques sans la visibilité.

E

L'annuaire de l'écosystème

Les références du guide, regroupées. État au 12 juin 2026 — l'écosystème bouge vite, les principes de choix du chapitre 9 restent.

Officiel Anthropic

claude.ai	L'application — web, iOS, Android, macOS, Windows
platform.claude.com	Console API : clés, plafonds, suivi de consommation, Workbench
docs (platform.claude.com/docs)	Documentation API, modèles, prompt caching, Batch, Agent SDK
modelcontextprotocol.io	Le standard MCP : spécification, SDK Python/TypeScript, serveurs de référence
anthropic.com/news	Annonces officielles — la source à vérifier avant de relayer un chiffre
status.claude.com	État des services — à consulter avant de déboguer votre propre code

Serveurs MCP cités dans ce guide

Serveur	Type	Usage principal
GitHub	Distant	Issues, PR, revues (chapitre 6)
Linear / Jira / Asana	Distant	Gestion de projet en langage naturel
Notion / Confluence	Distant	Base de connaissances interrogeable
Supabase / Postgres	Distant ou local	Bases de données (lecture d'abord)
Figma / Canva	Distant	Design vers code, génération de visuels
Stripe	Distant	Paiements, abonnements, factures
Slack / Teams	Distant	Recherche et préparation de messages
Playwright	Local	Pilotage de navigateur, tests, extraction
Context7	Distant	Documentation de bibliothèques à jour
Apify / Firecrawl	Distant	Extraction web à l'échelle
Zapier / n8n / Make	Distant	Déclencher des automatisations multi-applications
Filesystem (référence)	Local	Accès contrôlé à un dossier de la machine

Apprendre à la source : ressources et vidéos officielles

Anthropic publie une quantité de matériel pédagogique gratuit. Les points d'entrée qui valent le détour :

Chaîne YouTube Anthropic	youtube.com/@anthropic-ai — démos de lancement, explications de fonctionnalités, sessions techniques
Code with Claude	claude.com/code-with-claude — la conférence développeurs, sessions enregistrées (Claude Code, Agent SDK, MCP, déploiements partenaires)
Anthropic Academy / Learn	anthropic.com/learn — parcours guidés sur le prompting, les agents, l'API, l'usage en entreprise
Documentation officielle	platform.claude.com/docs — quickstarts, prompt caching, tool use, Batch, Agent SDK, avec exemples
Cookbook Anthropic	github.com/anthropics/anthropic-cookbook — recettes de code prêtes à exécuter (RAG, agents, vision, outils)
Prompt Library	docs.claude.com/prompt-library — prompts de référence par cas d'usage

Liens vérifiés au 12 juin 2026. Les sessions Code with Claude (San Francisco, Londres, Tokyo) sont diffusées sur la chaîne YouTube.

Les voix à suivre, citées dans ce guide

Simon Willison (simonwillison.net) — le testeur indépendant de référence, méthodique et chiffré. **Ethan Mollick** (oneusefulthing.org) — l'usage professionnel des modèles, expériences documentées. **Andrej Karpathy** — la lecture technique de fond. Trois sources qui testent avant d'écrire — le critère qui manque à l'essentiel du commentaire IA.

> **MÉTHODE DE VÉRIFICATION EXPRESS**
>
> Avant de relayer ou de décider sur la foi d'une annonce IA : (1) la source primaire existe-t-elle — billet officiel, system card, dépôt public ? (2) le chiffre a-t-il une méthodologie — benchmark nommé, conditions, date ? (3) un testeur indépendant l'a-t-il reproduit ? Deux non sur trois : c'est du marketing, pas une information.

F

À propos d'Otium

Otium est un atelier IA : nous concevons et mettons en place des systèmes d'intelligence artificielle sur mesure — audits, automatisations, agents, formation des équipes.

Pas de slides théoriques : des systèmes qui tournent, construits avec vos outils et vos données, transmis à vos équipes. Nous accompagnons des organisations de tous secteurs — services, industrie, santé, événementiel — du premier audit au système en production.

Ce guide prolonge notre média, où nous décryptons chaque semaine l'actualité de l'IA pour les décideurs : **otium-ai.co/media**

Pour échanger sur vos projets : **otium-ai.co/contact**

> **UNE RÈGLE D'HONNÊTETÉ**
>
> Tous les chiffres de ce guide — tarifs, benchmarks, dates — proviennent de sources officielles vérifiées au 12 juin 2026. Les scénarios chiffrés sont des exemples types construits à partir des tarifs officiels : ils illustrent une méthode de calcul, pas des promesses de résultats.

Le Guide Claude 2026 — édition de juin 2026. Rédigé et vérifié par l'équipe Otium. Les tarifs, dates et chiffres cités proviennent de sources officielles au 12 juin 2026 et peuvent évoluer.

L'IA ne remplace pas votre métier.
Elle récompense ceux qui l'équipent.

Édité par Otium
ATELIER IA SUR-MESURE · OTIUM-AI.CO