MP-i

LIVRE BLANC TECHNIQUE

Choisir des outils open source pour son entreprise

Grille de décision en 12 questions

Grille de décision en 12 questions structurées en 4 dimensions pour adopter un outil open source sans risque — gouvernance, licence, communauté, TCO. Annexe Panorama des licences (OSI, Creative Commons, OSHWA, fausses licences).

Mp-i - Mattieu Pottier Indépendant - www.mp-i.pro

Consultant en transformation digitale — SIG, géomatique, Web, ERP Odoo

Version 1.0.0 — Mai 2026
Document distribué librement

SOMMAIRE

Pourquoi cette grille

Choisir un outil open source ne se résume pas à « *c'est libre, on prend* ». Une mauvaise sélection coûte plus cher qu'un outil propriétaire mal choisi — parce que les signaux de fragilité sont moins visibles. Cette grille trie un projet open source adoptable d'un projet à éviter, sans expertise technique avancée et en moins de deux heures par projet.

12
questions structurées en 4 dimensions

4
dimensions de décision — Besoin · Projet · Licence · Exécution

3
relicences majeures décortiquées — Elasticsearch, Terraform, Redis

2h
suffisent pour auditer un projet open source

◆ **À propos de ce document**

Ce livre blanc accompagne l'article #3 du blog MP-i — *Choisir des outils open source : grille de décision en 12 questions*. Il peut être lu seul. Distribution libre, pas de formulaire requis.

Préface

Deux postures se croisent en permanence sur le terrain. La première : « l'open source, c'est libre, c'est gratuit, on prend, on verra bien ». La seconde, symétrique : « l'open source, c'est risqué, c'est instable, on prend du propriétaire, au moins on sait qui appeler ». Les deux se trompent — et les deux coûtent cher.

Cet écart vient d'une absence : il n'existe pas de grille de lecture structurée et accessible aux décideurs non-techniques pour trancher entre un projet open source solide et un projet fragile. La connaissance existe — chez les CTO, dans les fondations, dans la presse spécialisée — mais elle reste illisible pour un dirigeant de PME ou un DSI qui n'a pas le temps de devenir expert en gouvernance logicielle.

Résultat : le choix se fait à l'intuition, ou pire, sur la base d'une réputation marketing.

Cette grille est issue de mes propres choix d'outils depuis 2020 et de mes missions auprès de clients PME et ETI.

Ce n'est pas une grille académique — c'est une grille terrain, construite par itérations à partir de ce qui a tenu et de ce qui s'est cassé. Elle a fait l'objet de douze à dix-huit mois de recul d'usage avant publication.

À qui s'adresse ce document : aux décideurs de PME qui doivent arbitrer une adoption open source, aux DSI qui cadrent leur stratégie de stack, aux éditeurs logiciels qui choisissent leurs briques techniques, aux intégrateurs qui sélectionnent un module avant de le déployer chez un client.

Ce n'est pas un manuel pour développeurs, ce n'est pas un traité juridique. C'est un outil de décision.

> ***Une mauvaise sélection open source coûte plus cher qu'un outil propriétaire mal choisi — parce que les signaux de fragilité sont moins visibles.***
>
> *— Mattieu Pottier*

Introduction

Le paysage open source en 2026 n'est pas uniforme. Entre PostgreSQL — trente ans d'existence, des milliers de contributeurs, gouvernance multi-vendor sans propriétaire unique — et un projet GitHub maintenu par trois personnes en soirée, il n'y a aucune comparaison possible, sauf qu'ils portent tous deux l'étiquette « open source ».
Cette confusion coûte cher aux entreprises qui choisissent sans grille de lecture.

Pourquoi cette grille est stratégique

Les écarts réels entre projets sont massifs. À une extrémité, des projets considérés comme infrastructure critique mondiale : Linux, PostgreSQL, QGIS, Odoo Community. Tous portent des fondations ou des gouvernances multi-vendor matures, des cycles de release prévisibles, des écosystèmes de prestataires denses.
À l'autre extrémité, des projets exposés à un risque de mainteneur unique non sponsorisé, comme l'a illustré en mars 2024 l'affaire xz utils — où un contributeur malveillant a réussi à introduire une porte dérobée dans un outil utilisé par la quasi-totalité des distributions Linux, en exploitant l'épuisement du mainteneur historique — backdoor heureusement détectée avant déploiement en production sur les versions stables. Les deux projets se présentent visuellement de la même façon sur GitHub.
Sans grille, rien ne les distingue.

À cette dispersion structurelle s'ajoute un phénomène plus récent : les **relicences sous pression commerciale**. Entre 2021 et 2024, trois projets majeurs ont basculé de licences open source standard vers des licences dites *source-available* — Elasticsearch en 2021, Terraform en 2023, Redis en 2024. À chaque fois, des entreprises qui s'étaient appuyées sur ces outils ont dû arbitrer en urgence : migrer vers un fork communautaire, payer une licence commerciale, ou changer de stack.
Aucune de ces décisions n'aurait été facile à anticiper sans poser, *avant* l'adoption, des questions sur la gouvernance et le modèle économique du mainteneur. Ces trois cas sont décortiqués dans l'annexe Panorama des licences du présent document.

Le marché lui-même évolue rapidement. Les fondations neutres (Linux Foundation, Apache, CNCF, Eclipse, Mozilla, OSGeo) gagnent en influence comme refuges anti-relicence — le fork Valkey, accepté par la Linux Foundation huit jours après la relicence de Redis, en est l'illustration la plus frappante.
Côté législation, le Cyber Resilience Act européen, entré en vigueur le 10 décembre 2024 et

pleinement applicable le 11 décembre 2027, introduit de nouvelles obligations pour les éditeurs de logiciels — y compris pour certains projets open source commercialisés. L'open source n'est plus un sujet de marge ; c'est un sujet de conformité et de stratégie d'achat.

Pourquoi douze questions, et pas quatre, et pas trente.
Quatre questions ne suffisent pas pour distinguer un projet solide d'un projet en apparence solide mais fragile sur la gouvernance, la licence ou la pérennité du mainteneur. Trente questions sortent du temps disponible pour un décideur — l'audit devient un projet en soi, jamais lancé en pratique.

Douze questions, organisées en quatre dimensions — trois pour le Besoin, quatre pour le Projet, deux pour la Licence, trois pour l'Exécution, couvrent l'essentiel et tiennent en deux heures de travail par projet. C'est la durée que peut consacrer un dirigeant ou un DSI pour valider une adoption structurante. Au-delà, on entre dans le détail technique qui n'est pas du ressort du décideur.

Le périmètre de ce document

Le présent livre blanc couvre les projets open source logiciels à usage entreprise — ERP, SIG, GED, CMS, bases de données, middleware, outils d'infrastructure. L'approche méthodologique est applicable à tout projet open source structurant, quel que soit le secteur d'activité.
Les exemples sont volontairement variés (PostgreSQL, Odoo, Redis, Nextcloud, Linux) pour illustrer les principes sans enfermer la grille dans une niche.

Le document ne couvre pas en analyse de fond les licences hardware (OSHWA) ni les licences de contenu (Creative Commons) — elles sont traitées en synthèse dans l'annexe Panorama des licences pour les décideurs concernés par l'IoT, l'électronique ou la documentation ouverte, mais ne font pas l'objet de fiches dédiées.

Sont également hors périmètre les outils open source à usage strictement développeur (compilateurs, IDE, gestionnaires de paquets) : la cible de la grille est le décideur qui adopte un outil métier ou structurant, pas l'équipe technique qui choisit son outillage interne.

Profils de lecture

TAB. 1 — COMMENT LIRE CE DOCUMENT SELON VOTRE PROFIL

PROFIL	PARTIES PRIORITAIRES	OBJECTIF
Décideur PME ou DSI	Introduction · Partie 2 · Guide décisionnel · Conclusion	Décider d'adopter ou non un projet open source en moins de deux heures
Éditeur logiciel B2B	Partie 2 · Partie 4 (Projet) · Partie 5 (Licence) · Annexe (Panorama)	Choisir des briques open source sûres pour sa stack produit
Intégrateur Odoo ou prestataire	Partie 4 · Partie 6 (Exécution) · Guide décisionnel	Valider un module open source avant intégration chez un client

✗ **Choisir le mauvais outil open source, c'est...**

Six à dix-huit mois de retard sur un projet, un coût de migration imprévu, et une perte de confiance interne durable sur le sujet open source. Les trois cas Elasticsearch / Terraform / Redis ont chacun généré des coûts de migration significatifs dans l'écosystème — la plupart évitables avec une grille de lecture appliquée *avant* l'adoption.

Partie 1 — D'où vient l'open source, comment il vit

Avant d'apprendre à choisir un projet open source, il vaut la peine de comprendre d'où il vient et comment il vit. Le mouvement n'est pas né dans les startups californiennes des années 2000, et il ne fonctionne pas comme un don philanthropique. Il a un demi-siècle de racines culturelles, quarante ans d'existence formelle, et un marché économique mature dont la valeur dépasse aujourd'hui celle de nombreuses industries propriétaires.

Cette partie pose le décor en quatre pages — la grille des douze questions qui suit en dépendra.

1.1 La double racine : hackers du MIT et contre-culture californienne (1960-1985)

Le point de départ est technique et universitaire. Dans les laboratoires américains des années 60 et 70 — MIT AI Lab, Berkeley, Stanford — le logiciel n'est pas un produit commercial. Les chercheurs s'échangent leur code librement, par courrier puis sur ARPANET, comme on s'échange des résultats scientifiques. La notion même de *propriété logicielle* est marginale : on partage ce qu'on écrit, on attend la même chose en retour.

Steven Levy formalisera cette éthique en 1984 dans *Hackers: Heroes of the Computer Revolution*, livre où le terme **hacker** désigne un programmeur ingénieux et curieux — pas un pirate informatique.

En parallèle, sur la côte ouest, Stewart Brand publie en 1968 le *Whole Earth Catalog* — bible de la contre-culture californienne qui prône les outils libres et partagés au service de l'émancipation individuelle. Brand est un personnage charnière : il porte simultanément l'éthique hippie du partage et la fascination pour la technologie. À la première **Hackers Conference**, organisée par lui-même à Marin County en novembre 1984, il prononce une phrase qui résume une tension toujours d'actualité.

> ***On the one hand information wants to be expensive, because it's so valuable. The right information in the right place just changes your life. On the other hand,***

> ***information wants to be free, because the cost of getting it out is getting lower and lower all the time. So you have these two fighting against each other.***
>
> *Stewart Brand, Hackers Conference, automne 1984 (transcrit dans Whole Earth Review, mai 1985)*

Cette double racine — **hacker MIT + hippie californien** — est ce que les chercheurs Richard Barbrook et Andy Cameron nomment en 1995 la *Californian Ideology* : la fusion de l'éthique libertaire du partage et de l'entrepreneuriat individuel. C'est l'ADN culturel de la Silicon Valley naissante, et c'est dans ce terreau que va germer le mouvement logiciel libre.

Le déclic vient d'un seul homme. Vers 1980, **Richard Stallman**, programmeur au MIT AI Lab, ne peut plus modifier le pilote d'une imprimante Xerox parce que le code source en a été rendu propriétaire. Le choc culturel est durable. Le **27 septembre 1983**, il poste sur ARPANET et Usenet l'annonce du **projet GNU** — un système d'exploitation compatible Unix, entièrement libre. En janvier 1984, il quitte le MIT pour s'y consacrer à plein temps.

À la Hackers Conference de novembre 1984 — la même où Brand prononce sa phrase devenue mythique —, Stallman fait sa **première déclaration publique connue** selon laquelle tout logiciel devrait être libre. En mars 1985, il publie le **Manifeste GNU** ; le 4 octobre 1985, il fonde la **Free Software Foundation (FSF)** ; en 1989, paraît la première version de la **GPL**, première licence formellement copyleft. La FSF inscrit la liberté logicielle comme **droit fondamental**, pas comme efficacité économique. Cette dimension militante restera la signature du mouvement Free Software.

1.2 Le schisme : Free Software vs Open Source (1991-1998)

En 1991, un étudiant finlandais de 21 ans, **Linus Torvalds**, publie sur le serveur FTP de l'université d'Helsinki un noyau Unix-like qu'il a écrit pour s'amuser. Le **17 septembre 1991**, c'est la version 0.01 de **Linux**. En 1992, Torvalds choisit de placer Linux sous **GPL v2** — décision pragmatique qui va, dans les années qui suivent, transformer un projet hobby en infrastructure mondiale.

Linux n'est pas un projet GNU, mais son association avec les outils GNU (compilateur, shell, utilitaires) crée la pile **GNU/Linux** qui équipera bientôt la majorité des serveurs web.

Le tournant idéologique se produit en 1997-1998. Le 27 mai 1997, **Eric S. Raymond** présente au Linux Kongress de Würzburg un essai intitulé *The Cathedral and the Bazaar* — texte qui décrit le développement collaboratif et distribué de Linux comme un modèle d'ingénierie **supérieur** à celui des grands éditeurs propriétaires. L'argument est délibérément pragmatique, économique, technique. Pas militant.

Le **29 janvier 1998**, Netscape annonce qu'il libère le code source de son navigateur Communicator — geste fondateur qui deviendra le projet Mozilla puis Firefox. Le 3 février 1998, à Palo Alto, lors d'une réunion stratégique organisée pour capitaliser sur ce mouvement, **Christine Peterson** (Foresight Institute) propose un terme nouveau : **« open source »**. Le choix est délibéré — il s'agit de gommer la dimension militante du *« free software »* pour faciliter l'adoption en entreprise, et de lever l'ambiguïté du mot anglais *free* (libre / gratuit). Fin février 1998, Eric Raymond et Bruce Perens fondent l'**Open Source Initiative (OSI)** ; Raymond en sera le premier président.

En octobre 1999, l'OSI publie sa première liste officielle de licences open source approuvées — la liste qui, aujourd'hui encore, fait référence pour distinguer un projet *vraiment* open source d'un projet qui en porterait seulement le nom.

Deux mouvements parents, deux philosophies, qui cohabitent toujours en 2026.

FREE SOFTWARE		OPEN SOURCE
Stallman · FSF · 1985		**Raymond & Perens · OSI · 1998**
✓ Vocabulaire militant : « libre », pas « gratuit »	VS	→ Vocabulaire pragmatique : « open », pas « free »
✓ Liberté comme droit fondamental, dimension éthique		→ Liberté comme efficacité économique et technique
✓ Copyleft fort par défaut (GPL, AGPL)		→ Toutes licences OSI (permissives incluses)
✓ Public cible : militants, hackers, communauté universitaire		→ Public cible : entreprises, décideurs, marché

Stallman n'a jamais accepté le vocabulaire *« open source »*, qu'il considère comme une trahison commerciale du *« free software »*. Bruce Perens lui-même quittera l'OSI dès février 1999 dans un texte intitulé *« It's Time to Talk About Free Software Again »*, défendant la

position FSF. Pour le décideur d'aujourd'hui, deux clés pratiques suffisent.

Premièrement, en 2026, le terme « *open source* » dans une fiche produit ou un README est presque toujours au sens **OSI 1998** — pragmatique, business-compatible. Deuxièmement, la liste OSI (dix critères dans l'Open Source Definition, plus de cent licences approuvées) reste la **référence canonique** pour trancher si un projet est *vraiment* open source ou s'il se contente d'en porter l'étiquette marketing. C'est la fondation de la Q8 de la grille.

1.3 Comment l'open source vit aujourd'hui : un marché à part entière

« *Si l'open source est gratuit, qui paye ?* » La question est légitime, et elle revient à chaque mission. La réponse est sans mystère : derrière le code libre, il y a un marché économique réel, mature, dont la valeur dépasse aujourd'hui celle de plusieurs industries propriétaires.

Comprendre ce marché évite deux pièges symétriques — croire que l'open source est insoutenable par nature, ou croire qu'il n'a besoin d'aucun modèle économique.

Quelques ordres de grandeur

8,8 T$
Valeur côté demande de l'open source (Harvard Business School, Working Paper 24-038, janvier 2024)

34 Md$
Rachat de Red Hat par IBM (9 juillet 2019) — plus grosse acquisition open source de l'histoire

6,4 Md$
Rachat de HashiCorp par IBM (clôture 27 février 2025)

2,6x
Multiplicateur de la dépense logicielle mondiale annuelle si l'open source n'existait pas (étude Harvard, 2024)

L'étude Harvard menée par Hoffmann, Nagle et Zhou en janvier 2024 a calculé deux chiffres : un coût de remplacement *supply-side* (refaire le code de zéro une fois) à environ 4,15 milliards de dollars, et une valeur d'usage *demand-side* (ce qu'il faudrait à chaque entreprise pour reconstruire ce qu'elle utilise) à 8 800 milliards de dollars — soit environ

2,6 fois la dépense logicielle mondiale annuelle estimée à 3 400 milliards de dollars en 2020. L'open source n'est pas un sujet de marge ; c'est un pan structurel de l'économie logicielle.

Les six modèles de financement dominants

MODÈLE PÉRENNE

Fondations + sponsors entreprises

`Linux, Apache, Eclipse, Mozilla, CNCF`

Une fondation neutre héberge le projet. Les entreprises qui en dépendent (Google, Microsoft, AWS, Intel, IBM) la sponsorisent. Modèle dominant des projets d'infrastructure critique.

pérenne · infrastructure

MODÈLE COMMERCIAL

Open core

`GitLab, Mattermost, Nextcloud, Odoo`

Version communautaire gratuite + édition entreprise payante avec fonctionnalités avancées. Modèle dominant des éditeurs open source commerciaux contemporains.

rentable · single-vendor

MODÈLE RED HAT

Support et services

`Red Hat, EDB, Crunchy Data, Dalibo`

Code 100% libre. Le revenu vient du support, de la certification, de la formation, du conseil. Modèle qui a fait la fortune de Red Hat (34 Md$ chez IBM en 2019).

pérenne · services

MODÈLE À RISQUE

SaaS managé

`Elastic, MongoDB Atlas, Redis Cloud`

Le mainteneur héberge et facture le SaaS. Modèle rentable mais sous pression cloud (AWS, GCP, Azure peuvent proposer le même SaaS) — origine fréquente des relicences anti-cloud.

rentable · relicence-risquée

MODÈLE HYBRIDE

Dual-licensing

`Qt, MySQL pre-Oracle`

Licence libre + licence commerciale en parallèle. L'éditeur encaisse les licences commerciales, la communauté bénéficie de la libre. Pratique légitime, à comprendre avant d'adopter.

hybride · à-comprendre

MODÈLE FRAGILE

Donations et bénévolat

`nombreux projets de niche, xz utils`

Pas de modèle économique structuré. Le mainteneur travaille bénévolement, soutenu par donations occasionnelles. Modèle qui craque sous pression (xz utils, mars 2024).

fragile · à-éviter

Les acteurs économiques majeurs en 2026

La scène économique open source est en consolidation rapide. **IBM** est devenu le méga-acquéreur du secteur depuis 2019 : 34 milliards de dollars pour Red Hat le 9 juillet 2019, puis 6,4 milliards pour HashiCorp (annoncé en avril 2024, clôturé le 27 février 2025). La stratégie est explicite : construire un *« hybrid cloud open »* en s'appuyant sur les briques open source plutôt que sur des stacks propriétaires.

Microsoft a opéré une conversion stratégique après 2014, sous l'impulsion de Satya Nadella. En juin 2018, l'entreprise annonce le rachat de **GitHub** pour 7,5 milliards de dollars — opération clôturée en octobre 2018. Microsoft est aujourd'hui l'un des plus gros contributeurs au noyau Linux et l'éditeur de VSCode, le mainteneur de TypeScript et un sponsor majeur de fondations open source. L'éditeur historique du logiciel propriétaire est devenu un acteur structurant du libre.

AWS, Google Cloud et Alibaba financent largement les fondations neutres (CNCF, Linux Foundation, Apache) mais sont aussi à l'origine des tensions *« single-vendor vs cloud hyperscaler »* qui déclenchent les relicences récentes — Elastic en 2021, HashiCorp en 2023, Redis en 2024.

Les **fondations neutres** — Linux Foundation, Apache Software Foundation, CNCF (Cloud Native Computing Foundation), Eclipse, Mozilla, OSGeo — gagnent en influence depuis 2021 comme refuges contre les relicences single-vendor. Leur capacité à accueillir rapidement des forks (OpenSearch — fork 2021, transféré à la Linux Foundation en septembre 2024 ; OpenTofu en septembre 2023 ; Valkey en mars 2024) en a fait des acteurs systémiques de la résilience open source.

En France, l'écosystème compte des fournisseurs de support et d'intégration spécialisés (Smile, Dalibo, Camptocamp) et, côté souveraineté publique, la **DINUM SUITE** et **NUMSPOT** poussent l'adoption d'open source européen dans les administrations.

Frise historique 1968-2026

1968

Whole Earth Catalog (Stewart Brand)

Bible de la contre-culture californienne. Introduit l'idée d'outils libres et partagés au service de l'émancipation. Premier ADN culturel du mouvement open source.

1983

Annonce du projet GNU (Richard Stallman, 27 septembre)

Première initiative formelle pour créer un système d'exploitation entièrement libre. Réponse au verrouillage commercial d'Unix par AT&T.

1984

Hackers Conference + Hackers de Steven Levy (novembre)

Stewart Brand y prononce sa phrase mythique sur l'information qui veut être libre. Richard Stallman y fait sa première déclaration publique connue que tout logiciel devrait être libre. Steven Levy publie Hackers: Heroes of the Computer Revolution.

1985

Fondation de la Free Software Foundation (FSF, 4 octobre)

Stallman formalise le mouvement Free Software. Le Manifeste GNU est publié en mars. La GPL v1 paraîtra en 1989.

1991

Linus Torvalds publie Linux 0.01 (17 septembre)

Noyau libre publié comme projet hobby, placé sous GPL v2 dès 1992. Combiné aux outils GNU, il deviendra le socle de l'open source moderne.

1997

The Cathedral and the Bazaar (Eric Raymond, 27 mai)

Essai présenté au Linux Kongress de Würzburg. Décrit l'open source comme modèle d'ingénierie supérieur, pas comme cause militante. Pivot idéologique majeur.

1998

Netscape libère son code (29 janvier) → fondation de l'OSI (fin février)

Christine Peterson invente le terme "open source" le 3 février à Palo Alto. Eric Raymond et Bruce Perens fondent l'OSI fin février. Première liste officielle des licences publiée en octobre 1999.

2008

GitHub lancé par Tom Preston-Werner, Chris Wanstrath, PJ Hyett et Scott Chacon

La plateforme qui industrialise le développement open source distribué. Devient en quelques années l'épine dorsale du marché open source mondial.

2018

Microsoft rachète GitHub pour 7,5 Md$ (annonce juin, clôture octobre)

Tournant industriel. L'open source n'est plus l'ennemi des éditeurs propriétaires — il est leur stratégie. Microsoft devient l'un des premiers contributeurs au noyau Linux.

2019

IBM rachète Red Hat pour 34 Md$ (9 juillet)

Plus grosse acquisition open source de l'histoire. Démontre que le modèle « support pur, code 100% libre » est viable à grande échelle entreprise.

2021

Elasticsearch passe en SSPL/Elastic License v2 → fork OpenSearch

Première relicence single-vendor majeure d'un projet largement adopté. Inaugure la séquence Elastic → Terraform → Redis qui marquera la décennie.

2024

Crise mainteneur unique (xz utils, mars) + Redis relicencié (20 mars)

Backdoor CVE-2024-3094 sur xz utils — illustration de la fragilité du mainteneur unique non sponsorisé (cf. Q7), heureusement détectée avant déploiement en production sur les versions stables. Le 20 mars, Redis passe en SSPL/RSAL ; huit jours plus tard, la Linux Foundation accepte le fork Valkey. Le fork OpenTofu (sous MPL 2.0 d'origine de Terraform) est déjà actif depuis septembre 2023.

2024

Cyber Resilience Act adopté (10 décembre, entrée en vigueur)

Première régulation européenne horizontale sur la cybersécurité des produits avec éléments numériques. Reconnaît le statut spécifique des projets open source via le concept d'« open source steward ». Application pleine le 11 décembre 2027.

2025

IBM clôture le rachat de HashiCorp pour 6,4 Md$ (27 février)

Consolidation continue du marché open source commercial. Redis 8 sort en tri-licence SSPLv1 + RSALv2 + AGPLv3 le 1er mai 2025 — AGPLv3 ajouté comme option OSI-approved.

2026

Trois forks fondation actifs en production (OpenSearch, OpenTofu, Valkey)

Le modèle « fork sous fondation neutre » s'impose comme parade systémique aux relicences single-vendor. Le marché open source continue de mûrir, sur fond de Cyber Resilience Act et de souveraineté numérique européenne.

À retenir

L'open source n'est ni gratuit dans l'absolu, ni insoutenable. C'est un marché économique mature, structuré autour de six modèles de financement bien identifiés, et dont la valeur d'usage dépasse aujourd'hui plusieurs trillions de dollars. Le décideur PME doit comprendre **quel modèle** finance le projet qu'il adopte — c'est exactement ce que mesure la Q7 de la grille. La pérennité économique du mainteneur conditionne la pérennité technique de l'outil chez vous.

Partie 2 — Cadrage : les 4 dimensions de la grille

La grille organise l'évaluation d'un projet open source autour de **quatre dimensions** et **douze questions**. Cette partie pose la photo d'ensemble — chaque dimension, chaque question en aperçu, le verdict par question — avant que les parties suivantes ne traitent les douze fiches détaillées. Un lecteur pressé peut s'arrêter ici : il aura déjà une vision suffisante pour cadrer un audit interne.

2.1 Les quatre dimensions

L'évaluation d'un projet open source se fait en quatre temps successifs. Chaque dimension durcit ou allège les exigences des suivantes : un besoin court rend les exigences de gouvernance plus tolérables ; un besoin critique sur dix ans les durcit. L'ordre n'est pas négociable — qualifier le besoin **avant** de juger l'outil évite de tomber amoureux d'un projet inadapté.

Besoin

Qualifier le besoin avant de juger l'outil. Horizon temporel, criticité, existence d'un équivalent libre mature. Trois questions qui éliminent vite les fausses pistes.

Projet

Évaluer la solidité du projet open source visé. Gouvernance, activité du dépôt, diversité contributrice, modèle économique du mainteneur. Quatre questions — la dimension la plus chargée de la grille.

Licence

Distinguer le vrai open source du faux open source. Type de licence reconnue OSI, historique de relicence. Deux questions, mais l'une est éliminatoire — le mot « open » ne suffit pas.

Exécution

Vérifier que vous pouvez le mettre en œuvre. Capacité interne ou prestataire, support disponible, TCO sur cinq ans. Trois questions qui décident souvent du succès terrain.

2.2 Les douze questions en aperçu

"01"

Q1 — Quel est mon horizon temporel ?

"Combien de temps vais-je vivre avec cet outil ? Six mois, trois ans, dix ans ? L'horizon dicte la tolérance au risque sur toutes les autres questions."

"02"

Q2 — Cet outil est-il critique ou périphérique ?

"Au cœur du métier ou en périphérie ? Un outil critique exige une exigence de soutenabilité supérieure ; un outil périphérique tolère plus de fragilité."

"03"

Q3 — Existe-t-il un équivalent libre mature ?

"Y a-t-il un projet open source établi pour mon besoin, ou je m'aventure sur une niche ? Aller sur un projet de niche quand un équivalent mature existe demande de bonnes raisons."

"04"

Q4 — Quelle gouvernance ?

"Fondation neutre, mainteneur unique, éditeur commercial ? La forme juridique conditionne la pérennité — c'est le risque le plus sous-estimé par les décideurs."

"05"

Q5 — Le code bouge-t-il vraiment ?

"Activité du dépôt : commits récents, releases régulières, issues traitées. Quatre signaux visibles sur GitHub sans expertise technique."

"06"

Q6 — Quelle est la diversité contributrice ?

"Quel pourcentage des commits vient d'une seule entreprise ? La concentration est le risque numéro deux après la gouvernance."

"07"

Q7 — Comment le mainteneur gagne sa vie ?

"Donations, support pro, dual-licensing, open core ? Un projet sans modèle économique structuré est un projet à risque — la Partie 1 a posé les six modèles dominants."

"08"

Q8 — Quel type de licence ?

"Vraie licence reconnue OSI, source-available, propriétaire camouflé ? Le détail change tout, pas le mot « open »."

"09"

Q9 — Le projet a-t-il déjà changé de licence ?

"Combien de fois ? Sous quelle pression ? Un précédent de relicence vers du source-available est un signal fort, surtout combiné à un éditeur unique."

"10"

Q10 — Capacité interne ou prestataire ?

"Qui va le déployer et le maintenir ? Sans réponse claire, l'open source gratuit devient un open source abandonné chez vous."

"11"

Q11 — Quel support disponible ?

"Si ça casse à deux heures du matin, qui prend le téléphone ? Communauté, prestataire, mainteneur, SaaS managé — quatre niveaux possibles."

"12"

Q12 — Quel TCO sur cinq ans ?

"Coût total réel : licences évitées, déploiement, formation, maintenance, hébergement, sortie. Pas juste « c'est gratuit donc zéro euro »."

2.3 Verdicts par question et seuil de rejet

Chaque question reçoit un verdict qui décrit son poids dans la décision finale. Trois niveaux possibles.

STOP IMMÉDIAT

Éliminatoire

`5 questions sur 12`

Un seul échec sur une question éliminatoire suffit à écarter le projet. Ce sont les signaux qui n'ont pas de compensation possible — sans horizon défini, sans gouvernance saine, sans licence OSI, l'évaluation ne peut pas continuer.

rouge couperet

SURVEILLANCE

Sérieuse

`6 questions sur 12`

Un échec isolé ne suffit pas à écarter le projet, mais le cumul de plusieurs échecs sérieux devient rédhibitoire. Trois « non » sur ces six questions déclenchent un rejet par effet d'amplification.

orange cumul

À NOTER

Informative

`1 question sur 12`

Ne disqualifie pas mais oriente le choix de prestataire, le budget ou le calendrier. Donne une nuance utile à la décision sans peser sur le verdict.

vert contexte

Le tableau ci-dessous récapitule la photo de la grille — quelle dimension, quel verdict pour chaque question.

TAB. 2 — VERDICT PAR QUESTION — APERÇU

QUESTION	DIMENSION	VERDICT
Q1 — Horizon temporel	Besoin	[Éliminatoire]
Q2 — Criticité	Besoin	[Sérieuse]
Q3 — Équivalent mature	Besoin	[Éliminatoire*]
Q4 — Gouvernance	Projet	[Éliminatoire]
Q5 — Activité du dépôt	Projet	[Éliminatoire]
Q6 — Diversité contributrice	Projet	[Sérieuse]
Q7 — Modèle économique du mainteneur	Projet	[Sérieuse]
Q8 — Type de licence	Licence	[Éliminatoire]
Q9 — Historique de relicence	Licence	[Sérieuse]
Q10 — Capacité interne ou prestataire	Exécution	[Sérieuse]
Q11 — Support disponible	Exécution	[Informative]
Q12 — TCO sur 5 ans	Exécution	[Sérieuse]

** Q3 est éliminatoire avec exception : si **aucun** équivalent open source mature n'existe sur votre besoin, un projet de niche reste acceptable — à condition de **durcir** les exigences sur Q4 (gouvernance), Q6 (diversité contributrice) et Q10 (capacité de mise en œuvre).*

La répartition n'est pas le fruit du hasard. Les cinq questions éliminatoires couvrent les ruptures non négociables : sans horizon clair, la pondération du reste s'effondre ; sans équivalent mature identifié, on choisit à l'aveugle ; sans gouvernance saine, le projet peut basculer en source-available sous pression d'investisseurs ; sans activité de code, le projet est mort ; sans licence OSI, ce n'est pas du vrai open source.

Les six questions sérieuses, elles, mesurent des fragilités qui ne tuent pas seules mais cumulent leur effet. La question informative — Q11 sur le support — nuance le choix de calendrier et de budget sans peser sur le go/no-go.

À retenir

La grille couvre quatre dimensions et douze questions, dont cinq éliminatoires, six sérieuses et une informative. Le seuil de rejet recommandé — détaillé dans le Guide décisionnel en fin de document — est simple : *un seul échec éliminatoire* ou *trois échecs sérieux cumulés*. C'est suffisant pour trancher en moins de deux heures sur la majorité des projets que rencontre une PME ou un éditeur logiciel.

Partie 3 — Dimension Besoin (Q1 à Q3)

Avant de juger un outil, qualifier le besoin. Cette dimension élimine vite : un horizon court ou un besoin périphérique relâche l'exigence de soutenabilité, alors qu'un besoin critique de longue durée durcit toutes les questions suivantes. Trois questions qui se traitent en moins d'une heure de réflexion interne, et qui conditionnent la suite de l'audit.

3.1 Q1 — Quel est mon horizon temporel ?

Sous-titre : *Combien de temps vais-je vivre avec cet outil ?*

`Éliminatoire` · `5 min d'analyse` · `Audit interne`

Pourquoi cette question

L'horizon dicte la tolérance au risque sur toutes les autres questions. À six mois, un projet open source modéré, voire fragile, peut suffire — le temps d'un POC, d'une migration intermédiaire, ou d'un usage saisonnier. À dix ans, il faut un projet qui survivra à dix ans, ce qui implique une fondation neutre, une communauté large, un modèle économique pérenne. Sans horizon défini, la pondération du reste de la grille devient arbitraire — c'est pour cela que cette question est éliminatoire.

La question piège la plus fréquente, en mission : confondre la **durée du contrat** avec la **durée de vie de la donnée**. Un ERP signé pour trois ans, c'est dix ans de données comptables, fiscales, RH derrière. Le contrat est court, la dépendance est longue. L'horizon réel est celui de la donnée, pas du contrat.

Comment y répondre

Identifier honnêtement l'usage cible. Trois cas typiques : remplacement court (POC, transition, douze à dix-huit mois), outil métier (trois à cinq ans), socle structurant (dix ans et plus). Le test : *« Si l'outil disparaît dans trois ans, est-ce que je peux vraiment changer, ou est-ce que je suis enchaîné par mes données et mes process ? »* Si la réponse est « enchaîné », l'horizon réel est long, même si le projet est présenté comme court.

TAB. 3 — SIGNAUX Q1 — HORIZON TEMPOREL

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Horizon défini explicitement avant de chercher l'outil	Choix de l'outil avant la qualification du besoin
Distinction usage / donnée — la donnée vit plus longtemps que l'outil	Confusion entre durée du contrat et durée de vie de la donnée
Sponsor interne identifié sur les 3 prochaines années	Personne en charge de l'outil à moyen terme
Plan de sortie pré-imaginé même si non activé	Aucune réflexion sur la sortie à l'entrée

Exemple appliqué

Une PME industrielle adopte un GMAO open source pour dix-huit mois, dans l'attente d'un ERP unifié qui couvrira aussi la maintenance. Horizon court, exigence modérée — un projet open source de niche, mainteneur unique, communauté restreinte, peut passer. La même PME adopte le même GMAO pour huit ans en cœur de production : exigence radicalement différente — fondation neutre ou multi-vendor, diversité contributrice, roadmap LTS, prestataire local. Le même outil ne supporte pas les deux horizons.

> ✗ **Verdict Q1 — Éliminatoire**
>
> Sans horizon clair, la grille ne peut pas pondérer les autres questions. Si vous ne savez pas combien de temps vous allez vivre avec cet outil, c'est trop tôt pour choisir. Reprendre la qualification du besoin avant tout audit technique.

3.2 Q2 — Cet outil est-il critique ou périphérique ?

Sous-titre : *Au cœur du métier, ou en périphérie ?*

`Sérieuse` · `15 min d'analyse` · `Audit interne`

Pourquoi cette question

La criticité durcit toutes les exigences suivantes. Un outil critique en panne arrête l'activité — facturation bloquée, livraisons stoppées, support client coupé. Un outil périphérique en panne ralentit, gêne, mais ne stoppe pas. La même fragilité de gouvernance ou de mainteneur n'a pas le même poids selon la criticité.

Cette question n'est pas éliminatoire seule — un outil périphérique mal classé reste tolérable, on perd seulement en confort. Mais combinée à un échec sur Q11 (support) ou Q10 (capacité interne), elle devient le critère qui fait basculer le verdict global.

Comment y répondre

Le test de criticité opérationnel se formule en une phrase : « *Si l'outil disparaît demain matin, combien de temps mon activité tient avant un impact métier visible ?* »

Plus de 24 heures → périphérique

Entre 4 et 24 heures → sérieux

Moins de 4 heures → critique

La criticité n'est pas une propriété intrinsèque de l'outil — c'est une propriété de son **usage** dans votre organisation. PostgreSQL en moteur de production : critique. PostgreSQL en environnement de test isolé : périphérique. Même outil, criticités opposées selon l'usage.

TAB. 4 — SIGNAUX Q2 — CRITICITÉ

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Criticité documentée dans le plan de continuité d'activité	Aucun plan de continuité pour l'outil
Plan de bascule pré-identifié en cas de défaillance	Aucune alternative envisagée, dépendance totale
Cohérence entre criticité et exigence de support choisi	Outil critique avec support communautaire seul
Test d'impact réalisé (chaos engineering, drill) au moins une fois	Hypothèse de disponibilité non vérifiée

Exemple appliqué

Une ETI de logistique utilise un outil open source de visualisation de logs en arrière-plan, consulté par l'équipe IT une fois par jour pour analyser les incidents passés. Périphérique — si l'outil tombe une semaine, personne ne s'en aperçoit côté métier. La même ETI utilise PostgreSQL comme moteur de sa base de données de tracking colis en temps réel : critique — toute coupure de plus de quinze minutes bloque la facturation et déclenche des appels clients. Les deux outils sont open source, mais l'exigence sur le reste de la grille n'a rien à voir.

> ▲ **Verdict Q2 — Sérieuse**
>
> Non éliminatoire en soi, mais durcit ou allège l'ensemble des questions suivantes. Un outil critique doit passer toutes les questions sérieuses au minimum — l'inverse n'est pas vrai pour un outil périphérique. Documenter la criticité avant de continuer l'audit.

3.3 Q3 — Existe-t-il un équivalent libre mature pour mon besoin ?

Sous-titre : *Sur quel terrain je m'aventure ?*

`Éliminatoire` · `30 min d'analyse` · `Communauté` + `Site projet`

Pourquoi cette question

Si un projet open source établi existe pour votre besoin — PostgreSQL pour une base de données relationnelle, QGIS pour de la cartographie desktop, Odoo Community pour un ERP PME, Nextcloud pour de la GED collaborative — s'aventurer sur un projet de niche est rarement justifié. Le risque sur les questions suivantes est démultiplié : un projet de niche est souvent porté par un mainteneur unique, dispose d'une communauté restreinte, propose peu de prestataires sur le marché, et offre rarement une trajectoire de sortie crédible.

Cette question est éliminatoire dans un sens précis : si un équivalent mature existe et que vous choisissez quand même un projet de niche, vous devez pouvoir justifier ce choix par au moins trois raisons techniques fortes — spécification métier introuvable ailleurs,

performance qui change l'économie du projet, contrainte légale ou sectorielle spécifique. Sans ces raisons, retenir l'équivalent mature est le bon choix. Inversement, si aucun équivalent mature n'existe sur votre besoin (cas réel pour certains métiers très spécifiques), le projet de niche est acceptable — à condition de **durcir** les exigences sur Q4 (gouvernance), Q6 (diversité contributrice) et Q10 (capacité de mise en œuvre).

Comment y répondre

Un tour rapide en trente minutes suffit pour cartographier l'existant :

- **OSI directory** sur `opensource.org` — liste officielle des projets reconnus
- **GitHub trending par tag métier** — pour identifier les projets actifs
- **Fondations sectorielles** — LF Energy (énergie), LF Decentralized Trust (blockchain), OSGeo (géomatique), CNCF (cloud native)
- **Presse spécialisée** — LWN, The Register, Phoronix pour les retours d'expérience
- **Annuaires d'intégrateurs français** — pour vérifier qu'il existe au moins quelques prestataires capables de déployer

La règle empirique : **si trois sources indépendantes** citent le même projet open source comme référence sur votre besoin, l'équivalent mature existe. Si vous n'en trouvez aucune, vous êtes sur un projet de niche — à assumer.

TAB. 5 — SIGNAUX Q3 — ÉQUIVALENT MATURE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Au moins un projet open source établi (10 ans et plus, fondation ou multi-vendor) couvre le besoin	Le projet visé est seul sur son créneau, sans concurrent open source
Plusieurs intégrateurs ou prestataires français le couvrent	Aucun prestataire local identifié, support assuré uniquement par le mainteneur
Compatibilité avec des standards ouverts (formats, API)	Aucun standard, format propriétaire, lock-in technique
Trois sources indépendantes citent le projet comme référence sectorielle	Aucune citation hors du site officiel du projet

Exemple appliqué

Besoin de GED entreprise pour une PME de cinquante personnes : Nextcloud (gouvernance fondation, multi-vendor), Alfresco Community, OpenKM — trois équivalents matures disponibles, tous installés depuis plus de dix ans, tous dotés d'un écosystème de prestataires. Choisir une GED open source de niche à la place demande une justification technique forte.

À l'inverse, besoin spécifique très métier — par exemple un logiciel open source de gestion d'arbres urbains avec relevés terrain et synchronisation hors ligne : pas d'équivalent mature en 2026. Un projet de niche est acceptable, à condition de durcir l'audit sur la gouvernance, la diversité contributrice et la capacité de mise en œuvre.

✗ **Verdict Q3 — Éliminatoire**

Si un équivalent open source mature existe sur votre besoin, choisir un projet de niche à la place doit être justifié par au moins trois raisons techniques fortes. Sinon, retenir l'équivalent mature. Si aucun équivalent mature n'existe, le projet de niche est acceptable — à condition de durcir Q4 (gouvernance), Q6 (diversité) et Q10 (capacité). Sans grille de comparaison, l'évaluation des questions suivantes est aveugle.

◆ **Récapitulatif Partie 3 — Besoin**

Trois questions, dont **deux éliminatoires** (Q1 horizon, Q3 équivalent mature) et **une sérieuse** (Q2 criticité). Cette dimension est la plus rapide à traiter — une heure suffit — et c'est aussi la plus souvent escamotée par les décideurs pressés. La pondération de toute la suite de la grille en dépend : sans horizon clair, sans criticité documentée, sans cartographie de l'existant, l'audit technique d'un projet open source perd son sens.

Partie 4 — Dimension Projet (Q4 à Q7)

Le besoin est qualifié. Vient la dimension la plus chargée de la grille : la solidité du projet open source lui-même. Quatre questions qui couvrent la gouvernance, l'activité du dépôt, la diversité contributrice et le modèle économique du mainteneur. C'est ici que la grille fait son travail le plus dur — distinguer un projet open source solide d'un projet en apparence solide mais fragile, sur des signaux **visibles sans expertise technique**.

4.1 Q4 — Quelle gouvernance ?

Sous-titre : *Fondation neutre, mainteneur unique, éditeur commercial ?*

`Éliminatoire` · `30 min d'analyse` · `Site projet`

Pourquoi cette question

La gouvernance est **le risque numéro un**. Plus que la qualité du code, plus que l'activité du dépôt, plus que la communauté. Un projet bien codé mais gouverné par un éditeur unique peut relicencier du jour au lendemain — c'est exactement ce qui est arrivé à Elasticsearch en 2021, Terraform en 2023, Redis en 2024. Un projet sous fondation neutre, lui, ne peut pas — la fondation est précisément là pour empêcher cette bascule.

Pour le décideur non-technique, c'est aussi le risque le plus sous-estimé. Le code source, l'activité GitHub, la qualité des releases — tout cela est visible et plutôt rassurant pour la plupart des projets actifs. La structure juridique qui possède le projet, elle, n'est jamais affichée en page d'accueil. Il faut aller la chercher. Et c'est pourtant elle qui décide de la pérennité de votre choix à cinq ou dix ans.

> ***La gouvernance d'un projet open source est le contrat de mariage que vous signez sans le lire — et qui décidera, en cas de divorce, si vous repartez avec vos données ou si vous payez pour les récupérer.***
>
> *— Mattieu Pottier*

Comment y répondre

Identifier la forme juridique qui possède et oriente le projet. Quatre formes typiques, par ordre décroissant de résilience :

Fondation neutre — Apache Software Foundation, Linux Foundation, Eclipse, Mozilla, OSGeo, CNCF. Le projet appartient à une entité à but non lucratif, gouvernée par un conseil d'administration multi-entreprises. Aucun acteur unique ne peut imposer une relicence.

Association indépendante — Document Foundation (LibreOffice), KDE e.V., PostgreSQL Global Development Group. Pas une fondation au sens strict, mais une gouvernance collégiale documentée et stable.

Entreprise unique avec gouvernance ouverte — Odoo SA, Nextcloud GmbH, GitLab Inc. L'éditeur commercial possède la marque et oriente la roadmap, mais le code communautaire est sous licence reconnue OSI, et un fork reste possible techniquement.

Entreprise unique avec gouvernance fermée — Elastic pre-2021, HashiCorp pre-2024, Redis Ltd. pre-2024. L'éditeur décide seul, les contributeurs externes signent un CLA qui cède leurs droits. C'est la configuration la plus risquée — et c'est exactement celle qui a produit les trois relicences majeures de la décennie.

Pour identifier la forme, trois sources suffisent : le fichier `GOVERNANCE.md` ou `CONTRIBUTING.md` à la racine du dépôt principal, la page « About » ou « Project Structure » du site officiel, et la composition du steering committee ou du conseil d'administration s'il est public.

TAB. 6 — SIGNAUX Q4 — GOUVERNANCE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Fondation neutre ou association indépendante multi-vendor	Mainteneur unique salarié d'une seule entreprise commerciale
Process de décision documenté et public (RFC, votes, mailing list)	Décisions opaques, pas de trace publique des arbitrages
Conseil ou steering committee composé de plusieurs entreprises	Tous les décideurs travaillent pour la même entreprise
Pas de CLA contributor, ou DCO seul (Linux Foundation)	CLA contributor qui cède des droits étendus à l'éditeur

Exemple appliqué

PostgreSQL est gouverné par la PostgreSQL Global Development Group, une association informelle multi-vendor : EDB, Crunchy Data, Microsoft, AWS, NTT et plusieurs autres contribuent au noyau. Aucune entreprise unique ne peut décider d'une relicence. Trente ans d'existence sans une seule controverse de gouvernance — gouvernance saine. À comparer à **Redis** en mars 2024 : Redis Ltd. possédait à la fois la marque, le dépôt principal, l'équipe de mainteneurs salariés, et le CLA contributor. L'entreprise a pu décider seule, en quelques semaines, de basculer en SSPL/RSAL — gouvernance single-vendor, risque matérialisé.

> ✗ **Verdict Q4 — Éliminatoire**
>
> Sans gouvernance neutre ou multi-vendor, le projet peut basculer en source-available sous pression d'investisseurs, d'IPO ou de rachat. C'est le risque le plus sous-estimé par les décideurs non-techniques, et c'est exactement celui qui s'est matérialisé pour Elastic, HashiCorp et Redis entre 2021 et 2024. Sur un outil critique de longue durée, exiger une fondation neutre.

4.2 Q5 — Le code bouge-t-il vraiment ?

Sous-titre : *Comment lire un GitHub sans être développeur*

`Éliminatoire` · `15 min d'analyse` · `GitHub/GitLab`

Pourquoi cette question

Un projet *abandonware* est un projet mort. Le code peut être excellent à l'instant T — bien architecturé, bien documenté, fonctionnel — mais sans activité, il accumule la dette de sécurité (vulnérabilités non patchées), la dette de compatibilité (montées de version des dépendances non suivies) et la dette fonctionnelle (besoins métier qui évoluent sans réponse). Au bout de douze à vingt-quatre mois sans activité réelle, un projet open source devient un passif technique, pas un actif.

C'est éliminatoire pour une raison simple : il n'existe aucune façon de compenser l'absence d'activité. Un projet où personne ne travaille ne se rattrape pas par l'ajout d'un support payant ou d'un meilleur prestataire. Le projet doit vivre.

Comment y répondre

Quatre signaux suffisent, tous visibles sur GitHub en cinq minutes, sans lire une seule ligne de code.

"01" Onglet Code

- ✓ Date du dernier commit visible à côté de chaque fichier
- ✓ Repère : moins de 30 jours = sain, plus de 6 mois = alerte
- ✓ Aucun code à lire — juste la date

"02" Onglet Releases

- ✓ Date et fréquence des versions publiées
- ✓ Repère : 3 à 12 releases par an = sain
- ✓ Aucune release depuis 12 mois = alerte

"03" Onglet Issues

- ✓ Ratio open / closed et dates des dernières issues fermées
- ✓ Repère : issues récentes traitées en moins de 30 jours = sain
- ✓ Issues ouvertes depuis plus d'un an sans réponse = alerte

"04" Insights / Pulse

- ✓ Courbe d'activité sur les 12 derniers mois (Contributors)
- ✓ Résumé des 4 dernières semaines (Pulse)
- ✓ Repère : courbe régulière ou croissante = sain, effondrement = alerte

Quatre onglets, cinq minutes, une décision — sans lire une seule ligne de code.

TAB. 7 — SIGNAUX Q5 — ACTIVITÉ DU DÉPÔT

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Commits hebdomadaires ou plus fréquents sur les 12 derniers mois	Aucun commit depuis 6 mois
Releases régulières (3 à 12 par an, cycle prévisible)	Aucune release depuis 12 mois, ou releases erratiques
Issues récentes traitées en moins de 30 jours	Issues ouvertes depuis plus d'un an sans réponse
Pull requests fermées régulièrement (acceptées ou refusées)	PR ouvertes pendant des mois sans commentaire mainteneur

Exemple appliqué

Le noyau **Linux** affiche plusieurs milliers de commits par semaine et des releases régulières, avec un volume d'activité sur la *Linux Kernel Mailing List* (canal historique du noyau) caricaturalement élevé. Sain à un point caricatural.

À l'inverse, un projet GitHub de niche typique abandonné — deux contributeurs en tout, dernier commit il y a dix-huit mois, cinquante issues ouvertes sans réponse depuis plus d'un an, dernière release il y a vingt-deux mois : *abandonware* en cours. L'outil peut être techniquement correct à l'instant où vous l'évaluez, dans six mois il sera vulnérable et incompatible avec les versions courantes de ses dépendances.

> ✗ **Verdict Q5 — Éliminatoire**
>
> Pas d'activité, pas d'avenir. Un projet *abandonware* ne s'adopte pas, même s'il est techniquement excellent à l'instant T. Sur un outil critique de longue durée, exiger au minimum des commits réguliers sur 12 mois glissants et une cadence de releases prévisible.

4.3 Q6 — Quelle est la diversité contributrice ?

Sous-titre : *Quel pourcentage des commits vient d'une seule entreprise ?*

`Sérieuse` · `30 min d'analyse` · `GitHub/GitLab` + `Communauté`

Pourquoi cette question

La concentration contributrice est le risque numéro deux, juste après la gouvernance. Un projet multi-contributeurs en apparence mais où quatre-vingts pour cent des commits viennent d'une seule entreprise est en réalité un projet *single-vendor déguisé*. Si cette entreprise pivote, lève des fonds défavorablement, est rachetée ou ralentit son investissement, le projet meurt techniquement ou bascule juridiquement. C'est exactement ce qui s'est passé pour Redis : avant la relicence de mars 2024, Redis Ltd. représentait l'écrasante majorité des contributions au dépôt principal.

Cette question est sérieuse, pas éliminatoire — un single-vendor peut très bien vivre des années, voire des décennies. Mais **combinée à un échec sur Q4 (gouvernance single-vendor) ou Q9 (historique de relicence)**, elle bascule l'évaluation : trois signaux convergents sur la concentration suffisent à écarter le projet sur les usages critiques.

Comment y répondre

GitHub → Insights → Contributors donne la répartition brute. Reste à identifier l'employeur de chaque top contributeur — information souvent disponible sur leur profil GitHub, leur LinkedIn ou la signature de leurs commits. L'objectif : calculer le pourcentage des commits venant de l'entreprise dominante sur les douze derniers mois.

Quatre seuils indicatifs pour cadrer l'analyse :

✅ Moins de 30%

- ✓ Sain — diversité réelle
- ✓ Exemples : Linux, PostgreSQL, Kubernetes
- ✓ Risque single-vendor faible

⚠️ Entre 30 et 60%

- ✓ À surveiller — vigilance modérée
- ✓ Exemples : Odoo Community, Mattermost
- ✓ OK si gouvernance ouverte

⚠️ Entre 60 et 90%

- ✓ Single-vendor déguisé
- ✓ Exemples : HashiCorp pre-2024
- ✓ Risque combiné avec Q4 et Q9

❌ Plus de 90%

- ✓ Single-vendor pur
- ✓ Exemples : Redis pre-2024
- ✓ Rejet si gouvernance également fermée

TAB. 8 — SIGNAUX Q6 — DIVERSITÉ CONTRIBUTRICE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Moins de 30% des commits d'une seule entreprise	Plus de 60% des commits d'une seule entreprise
Diversité de pays, de fuseaux horaires, d'employeurs	Tous les contributeurs dans le même bureau, mêmes signatures
Présence de contributeurs individuels indépendants	Aucun contributeur hors employés salariés du mainteneur
Plusieurs entreprises sur le top 10 des contributeurs	Une seule entreprise sur le top 10

Exemple appliqué

Kubernetes est un cas pédagogique : Google a porté le projet à ses débuts en 2014 et concentrait à l'époque, avec Red Hat, plus de quatre-vingts pour cent des commits — quatre-vingt-trois pour cent précisément selon les chiffres CNCF au moment de la donation en 2016. L'entreprise a délibérément réduit sa part en transférant le projet à la CNCF, en finançant la formation de contributeurs externes, et en intégrant des employés d'autres entreprises au steering committee. Trois ans plus tard, le duo Google + Red Hat tombait à trente-cinq pour cent des contributions cumulées — chute qui n'a jamais été reprise depuis. En 2026, plus de huit mille entreprises contribuent à Kubernetes, avec un top 5 stable (Google, VMware, Red Hat, Microsoft, Kubermatic) où aucune ne domine. Concentration saine, transition exemplaire.

À comparer à **Redis** pré-2024, où Redis Ltd. représentait à elle seule l'écrasante majorité des commits, sans diversité réelle dans le top 10. Quand la pression cloud s'est durcie, l'entreprise a basculé en SSPL/RSAL sans contre-pouvoir interne au projet.

> ▲ **Verdict Q6 — Sérieuse**
>
> Pas d'effet couperet immédiat (un projet single-vendor peut très bien vivre), mais combinée à Q4 (gouvernance single-vendor) ou Q9 (historique de relicence), c'est un signal cumulatif fort. Sur un outil critique avec une concentration supérieure à 60%, exiger une compensation explicite sur la gouvernance ou la licence.

4.4 Q7 — Comment le mainteneur gagne sa vie ?

Sous-titre : *Le modèle économique qui finance la maintenance*

`Sérieuse` · `30 min d'analyse` · `Site projet` + `Bilan financier`

Pourquoi cette question

Un projet open source sans modèle économique identifié est un projet à risque structurel. Le mainteneur a un coût — temps de développement, infrastructure d'hébergement, réponse aux issues, montées de version de sécurité, gestion communautaire.

Sans revenus pour couvrir ce coût, deux scénarios se réalisent à terme : soit le mainteneur s'épuise et abandonne (cas xz utils en 2024), soit l'entreprise mainteneur bascule sous pression d'investisseurs vers du source-available (cas Elastic, HashiCorp, Redis).

> *La typologie des six modèles dominants est posée en Partie 1.3 ; cette fiche se concentre sur **comment identifier le modèle d'un projet précis** et **évaluer sa robustesse**.*

Comment y répondre

Pour identifier le modèle économique d'un projet précis, quatre sources se complètent :

- **Site officiel du projet** — page « Sponsor », « About », « Funding », « Enterprise ». Qui apparaît ? Quels logos, quels engagements ?
- **GitHub** — fichier `FUNDING.yml` à la racine, présence de GitHub Sponsors, lien OpenCollective. Indique qui finance directement le projet.

- **Bilan financier** — si le mainteneur est une entreprise cotée (Elastic, MongoDB, Confluent, GitLab), ses rapports annuels SEC sont publics et donnent la répartition de ses revenus.
- **Presse spécialisée** — LWN, The Register, TechCrunch couvrent les levées de fonds, les ralentissements, les plans sociaux et les changements stratégiques majeurs.

Une fois le modèle identifié, le situer dans la typologie de la Partie 1.3 — fondation + sponsors, open core, support et services, SaaS managé, dual-licensing, donations et bénévolat. Les trois premiers modèles sont pérennes en pratique. Le SaaS managé est rentable mais structurellement risqué (origine fréquente des relicences anti-cloud). Le dual-licensing est légitime mais à comprendre. Le bénévolat pur sur un outil critique est un risque systémique.

TAB. 9 — SIGNAUX Q7 — MODÈLE ÉCONOMIQUE DU MAINTENEUR

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Modèle économique identifiable et communiqué publiquement	Modèle flou, opaque, ou non documenté
Diversification des revenus (support + SaaS + édition + sponsoring)	Une seule source de revenu, un seul gros client, un seul produit
Bilans publics ou communications financières transparentes	Aucune visibilité sur la viabilité économique
Mainteneur stable depuis plus de 5 ans, croissance régulière	Mainteneur startup pré-Série B sur produit unique, croissance erratique

Exemple appliqué

PostgreSQL fonctionne sur le modèle *fondation diffuse + multi-sponsors entreprises* posé en Partie 1.3 : pas de mainteneur unique au sens entreprise, les contributeurs sont salariés d'EDB, Crunchy Data, Microsoft, AWS, NTT — chacune monétise du support, du managé ou du cloud autour de PostgreSQL. La diversification est structurelle : aucune de ces entreprises ne peut tuer le projet, et la concurrence entre elles maintient un investissement continu dans le code commun. Modèle pérenne.

À comparer à **xz utils** en 2024 : projet utilisé dans la quasi-totalité des distributions Linux, donc d'importance systémique, mais maintenu bénévolement par une seule personne (Lasse Collin), sans sponsoring d'entreprise, sans modèle économique. Le mainteneur

historique, épuisé, a fini par déléguer progressivement à un contributeur qui s'est révélé malveillant — c'est ainsi qu'a été introduite la porte dérobée CVE-2024-3094 découverte en mars 2024.

Le risque économique du *bénévolat pur* s'est transformé en risque de sécurité de portée mondiale. Aucun outil de cette importance ne devrait reposer sur un modèle aussi fragile.

▲ **Verdict Q7 — Sérieuse**

Combinée à Q4 (gouvernance) et Q6 (diversité), Q7 dessine le profil de risque économique du projet. Une réponse fragile à Q7 — bénévolat pur sans sponsor, ou SaaS managé en concurrence frontale avec les hyperscalers — sur un outil critique (Q2) doit être un signal d'alerte fort. À combiner avec Q9 (historique de relicence) pour anticiper le risque de bascule.

◆ **Récapitulatif Partie 4 — Projet**

Quatre questions, dont **deux éliminatoires** (Q4 gouvernance, Q5 activité) et **deux sérieuses** (Q6 diversité, Q7 modèle économique). C'est la dimension la plus dense de la grille, et celle qui distingue le mieux un projet open source solide d'un projet en apparence solide mais fragile. Le triplet **Q4 + Q6 + Q7** dessine le **profil structurel de risque** : single-vendor + concentration contributrice + modèle SaaS managé sous pression hyperscaler. Ce profil se combine avec Q9 (historique de relicence) pour former le **triangle prédictif Q4 + Q7 + Q9** détaillé en Partie 5 — c'est ce triangle qui a réellement précédé les bascules Elasticsearch 2021, Terraform 2023, Redis 2024. La grille existe précisément pour détecter ces configurations avant l'adoption.

Partie 5 — Dimension Licence (Q8 et Q9)

Deux questions courtes mais décisives. La licence n'est pas un détail juridique — c'est ce qui détermine si vous pouvez réellement utiliser, modifier, redistribuer et survivre à un changement de gouvernance du projet. Cette partie reste en mode décision rapide ; la cartographie complète des licences (vraies licences OSI, fausses licences source-available, Creative Commons, OSHWA, mécanismes CLA/DCO) est traitée dans l'**Annexe — Panorama des licences**, à laquelle ces deux questions renvoient.

5.1 Q8 — Quel type de licence ?

Sous-titre : *Vraie open source OSI, ou source-available déguisé en open source ?*

`Éliminatoire` · `5 min d'analyse` · `opensource.org` + `LICENSE`

Pourquoi cette question

Depuis 2021, le marché s'est rempli de **licences qui se présentent comme open source mais ne le sont pas**. SSPL (Server Side Public License) de MongoDB, BSL (Business Source License) de HashiCorp, Elastic License v2, Redis Source Available License — toutes interdisent au moins un usage commercial standard, généralement le SaaS managé concurrent. Aucune n'est reconnue par l'OSI. Aucune ne respecte les dix critères de l'*Open Source Definition*.

Mais leurs sites officiels et leurs pages marketing utilisent abondamment le vocabulaire « *open* », « *source-available* », « *open core* » — pour entretenir la confusion.

Pour le décideur, la conséquence est concrète. Sous licence source-available, vous **n'avez pas le droit** de faire tourner le logiciel en SaaS pour vos clients, de l'intégrer dans certains produits commerciaux, ou de le forker librement. Les obligations sont souvent rétroactives sur les versions futures, et certaines clauses sont juridiquement ambiguës — Elastic License v2 et SSPL ont fait l'objet d'analyses juridiques contradictoires entre 2021 et 2024.

Si vous adoptez un outil en pensant qu'il est open source alors qu'il est source-available, vous prenez un risque juridique structurel sur tout l'usage que vous en faites.

Pas de « presque OSI ». Une licence est sur la liste officielle ou elle ne l'est pas. Le test prend cinq minutes.

Comment y répondre

Trois étapes, dans cet ordre :

Si le mainteneur utilise plusieurs licences pour différentes parties du code (un cœur communautaire OSI et des modules entreprise sous licence commerciale), c'est du **dual-licensing** ou de l'**open core** — légitime, mais à comprendre composant par composant.

TAB. 10 — SIGNAUX Q8 — TYPE DE LICENCE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Licence reconnue par l'OSI (MIT, Apache, GPL, AGPL, MPL, BSD…)	Licence custom non listée par l'OSI
Fichier `LICENSE` clair, à la racine, sans ambiguïté	Licence floue, multiples versions, exceptions non documentées
Si dual-licensing : périmètre OSI vs commercial clairement délimité	Mélange OSI / non-OSI dans le même dépôt, sans documentation
Compatible avec votre usage (SaaS, redistribution, modification)	Clauses restrictives sur le SaaS, le commercial, ou le fork

Exemple appliqué

PostgreSQL est sous *PostgreSQL License* — licence permissive reconnue OSI, dérivée de BSD/MIT, sans aucune clause commerciale restrictive. Vous pouvez l'utiliser, le modifier, le redistribuer, l'intégrer dans un produit propriétaire, le faire tourner en SaaS pour vos clients. Aucune ambiguïté, aucun risque juridique sur l'usage.

À comparer à **MongoDB** depuis octobre 2018 : passé sous **SSPL** (Server Side Public License) que l'OSI a explicitement rejetée en mars 2019 comme non conforme à l'*Open Source Definition*. SSPL contient une clause qui oblige tout fournisseur de SaaS managé MongoDB à publier en open source l'intégralité de son infrastructure de SaaS — clause volontairement dissuasive contre AWS et les autres hyperscalers.

Si vous adoptez MongoDB en pensant que c'est de l'open source, vous pourriez vous retrouver hors-clauses sur certains usages commerciaux, sans le savoir. C'est exactement le piège que cette question éliminatoire est conçue pour détecter.

✗ **Verdict Q8 — Éliminatoire**

Pas de « presque open source ». Une licence est OSI ou elle ne l'est pas. Si la licence du projet n'est pas dans la liste officielle de `opensource.org/licenses`, vous adoptez une licence source-available, et il faut traiter le projet comme un produit propriétaire — analyse juridique au cas par cas avant adoption. Sur un outil critique de longue durée, exiger une licence OSI sans exception. Détail complet des familles de licences en annexe.

5.2 Q9 — Le projet a-t-il déjà changé de licence ?

Sous-titre : *Un précédent de relicence est un signal d'alerte fort*

`Sérieuse` · `15 min d'analyse` · `Historique git` + `Presse spécialisée`

Pourquoi cette question

Trois projets majeurs ont basculé d'une vraie licence open source vers du source-available entre 2021 et 2024 : **Elasticsearch** en janvier 2021 (passage de Apache 2.0 vers SSPL/Elastic License v2), **Terraform** en août 2023 (passage de MPL 2.0 vers BSL), **Redis** en mars 2024 (passage de BSD 3-Clause vers SSPL/RSAL). À chaque fois, des entreprises qui s'étaient appuyées sur ces outils ont dû arbitrer en urgence — migrer vers un fork communautaire (OpenSearch, OpenTofu, Valkey), payer une licence commerciale, ou changer de stack. Aucune de ces décisions n'a été facile.

Le signal que Q9 capte : **si un projet a déjà relicencié une fois, il peut le faire à nouveau**. La direction qui a pris cette décision, ses investisseurs, sa pression concurrentielle face aux hyperscalers — rien de tout cela ne change automatiquement. Inversement, un projet qui n'a jamais relicencié et qui est sous gouvernance neutre (Q4) est structurellement protégé contre ce risque.

Q9 n'est pas éliminatoire — un projet peut très bien continuer à fonctionner après une relicence, et certains adoptants y restent fidèles. Mais combinée à Q4 (gouvernance single-vendor) et Q7 (modèle SaaS managé sous pression hyperscaler), elle complète le

triangle de risque posé en Partie 4 : ces trois signaux convergents prédisaient les relicences avant qu'elles ne surviennent. Et ils prédiront les prochaines.

Comment y répondre

Trois sources se complètent en quinze minutes :

- **Historique git** — `git log --follow LICENSE` sur le dépôt principal montre tous les changements de licence et leurs dates.
- **Page Wikipédia ou page « History » du projet** — toute relicence majeure y est documentée avec contexte et date.
- **Presse spécialisée** — LWN, The Register, TechCrunch, Hacker News. Chercher le nom du projet + « *relicensing* » ou « *license change* ».

Quatre cas typiques :

Aucune relicence sur toute l'histoire du projet — signal positif, structurel

Une relicence en gain de liberté (passage d'une licence permissive vers GPL, ou inverse, sans restriction commerciale ajoutée) — neutre, parfois positif

Une relicence vers source-available (Elastic, HashiCorp, Redis) — signal d'alerte fort, surtout si la gouvernance reste single-vendor

Plusieurs relicences en peu de temps sous pression d'investisseurs ou d'IPO — signal très négatif, indique une direction qui cherche un modèle économique au détriment de la communauté

TAB. 11 — SIGNAUX Q9 — HISTORIQUE DE RELICENCE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Aucune relicence sur 10 à 30 ans d'existence	Relicence récente (moins de 5 ans) vers du source-available
Si relicence : passage entre deux licences OSI documenté et justifié	Relicence opaque, communication marketing évasive
Engagement public du mainteneur à rester sous licence OSI	Aucune garantie publique, CLA contributor qui autorise la relicence
Sous gouvernance fondation depuis le départ (impossible de relicencier)	Sous éditeur unique avec marque et CLA — relicence techniquement possible à tout moment

Exemple appliqué

PostgreSQL est sous *PostgreSQL License* depuis 1996 — trente ans sans une seule relicence. Combiné à sa gouvernance multi-vendor (Q4) et à son modèle économique diffus (Q7), c'est un profil de risque relicence quasi nul. Aucun signal négatif sur Q9.

À comparer à **Redis** : sous BSD 3-Clause depuis ses débuts en 2009, l'entreprise Redis Ltd. a basculé vers le couple SSPLv1 + RSALv2 en mars 2024 sous pression de la concurrence cloud (AWS notamment proposait Redis managé sans contribuer en retour). Un an plus tard, le 1er mai 2025, Redis 8 est sorti en **tri-licence** ajoutant AGPLv3 (OSI-approved) aux deux licences source-available — manœuvre destinée à regagner les distributions Linux et la confiance de l'écosystème après le succès du fork Valkey sous Linux Foundation. Deux mouvements de licence en quatorze mois sur un projet structurant.

Combiné à une gouvernance single-vendor (Q4 fermée) et à un modèle SaaS managé (Q7 sous pression hyperscaler), Redis cochait avant 2024 toutes les cases du triangle de risque. Q9 transforme ce constat rétrospectif en outil prédictif : un autre projet qui présente la même combinaison aujourd'hui — single-vendor + SaaS managé + précédent de relicence — sera très probablement le prochain à basculer.

▲ **Verdict Q9 — Sérieuse**

Pas d'effet couperet seul, mais signal d'alerte fort combiné à Q4 (gouvernance) et Q7 (modèle économique). Le **triangle prédictif de relicence** — Q4 single-vendor + Q7 SaaS managé sous pression cloud + Q9 précédent de relicence — prédit les bascules vers source-available. Ce triangle est distinct du **profil structurel** Q4 + Q6 + Q7 évoqué en Partie 4, qui mesure la fragilité contributrice ; Q9 ajoute la dimension historique qui en fait un outil prédictif. Sur un outil critique, deux signaux sur trois suffisent à recommander un report ou une migration vers un fork sous fondation neutre. La cartographie complète des relicences récentes et des familles de licences est dans l'annexe Panorama des licences.

◆ **Récapitulatif Partie 5 — Licence**

Deux questions seulement, mais l'une est **éliminatoire** (Q8 type de licence) et l'autre constitue le sommet du triangle de risque relicence (Q9). C'est la dimension la plus rapide à traiter — vingt minutes au total — et c'est aussi celle qui produit le plus de surprises en mission. Beaucoup de décideurs PME pensent que tout ce qui est sur GitHub est open source ; une part significative des projets dits « *open* » sur les pages marketing 2026 sont en réalité sous licence source-available — selon l'expérience terrain, la proportion est loin d'être marginale. L'annexe Panorama des licences traite le sujet en détail.

Partie 6 — Dimension Exécution (Q10 à Q12)

Le besoin est qualifié, le projet est jugé sain, la licence est OSI. Reste la dernière dimension — la plus terrain : pouvez-vous réellement déployer et faire vivre cet outil ? Trois questions qui décident souvent du succès ou de l'échec de l'adoption, et qui se traitent une fois que les neuf premières ont passé le filtre.

6.1 Q10 — Capacité interne ou prestataire ?

Sous-titre : *Qui va le déployer et le maintenir ?*

`Sérieuse` · `30 min d'analyse` · `Audit interne` + `Annuaire prestataires`

Pourquoi cette question

Le code open source est gratuit ; sa mise en œuvre ne l'est pas. Sans réponse claire à « *qui va déployer, configurer, monter en version, dépanner cet outil* », l'adoption échoue dans une part significative des cas observés en mission.

L'outil tourne pendant six à douze mois grâce à la personne qui l'a installé, puis cette personne change de poste ou quitte l'entreprise, et plus personne ne sait comment ça marche. La dette opérationnelle s'accumule. Au bout de deux ans, l'outil est officiellement encore là, mais en réalité abandonné — figé sur une version obsolète, jamais patché, jamais mis à jour.

Q10 force la décision **avant** l'adoption : qui prend la responsabilité opérationnelle, sur quel horizon, avec quel budget. Trois configurations sont viables, une seule ne l'est pas.

Comment y répondre

Quatre options à considérer, par ordre décroissant de robustesse :

- **Compétence interne dédiée** — un ou plusieurs salariés maîtrisent l'outil avec du temps alloué. Idéal pour les ETI avec une équipe IT structurée. Coûteux mais résilient.
- **Prestataire local identifié** — intégrateur français ou européen avec contrat de support et maintenance. Modèle dominant pour Odoo, PostgreSQL, Nextcloud.

- **Mainteneur du projet en direct** — pour les éditeurs commerciaux (GitLab, Mattermost, Nextcloud GmbH). Coût plus élevé, accès direct aux développeurs.
- **Aucune capacité identifiée** — adoption « *on verra bien* », savoir-faire d'une personne unique en interne. Configuration qui produit les abandons silencieux à dix-huit mois.

Le test : « *Si la personne qui a installé l'outil quitte l'entreprise demain, qui prend le relais, dans quel délai, à quel coût ?* » Sans réponse précise, vous êtes en configuration 4 — à corriger avant l'adoption.

TAB. 12 — SIGNAUX Q10 — CAPACITÉ DE MISE EN ŒUVRE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Compétence interne dédiée OU prestataire identifié avant adoption	Adoption « on verra bien » sans capacité identifiée
Contrat de support signé ou en cours de négociation	Aucun contrat formel, dépendance à une personne unique
Plan de transmission documenté (procédures, accès, mots de passe)	Connaissance dans la tête d'une seule personne, non écrite
Au moins 2 prestataires alternatifs disponibles sur le marché	Un seul prestataire possible, lock-in fournisseur déguisé

Exemple appliqué

Une PME de cinquante personnes adopte **Odoo Community** pour son ERP. Trois options se présentent : embaucher un développeur Odoo en interne (coût annuel chargé de l'ordre de soixante mille euros, viable seulement sur un horizon de cinq ans et plus), signer avec un intégrateur français Odoo (coût de mise en œuvre de quinze à trente mille euros, puis maintenance annuelle de cinq à dix mille euros, modèle dominant), ou se reposer sur les compétences en interne de son responsable IT qui « connaît Python ».

La troisième option est tentante côté budget mais c'est celle qui produit les abandons : le responsable IT n'a pas le temps réel d'apprendre Odoo en profondeur, et au premier bug bloquant, personne ne sait quoi faire. Les deux premières sont structurellement viables.

> ▲ **Verdict Q10 — Sérieuse**
>
> Non éliminatoire en soi, mais combinée à Q2 (criticité) ou Q11 (support), une absence de capacité identifiée devient rédhibitoire. Sur un outil critique, exiger soit une compétence interne dédiée avec budget alloué, soit un prestataire identifié et un contrat signé **avant** l'adoption. Sur un outil périphérique, la communauté peut suffire — à condition d'assumer la dette opérationnelle qui va avec.

6.2 Q11 — Quel support est disponible ?

Sous-titre : *Si ça casse à deux heures du matin, qui prend le téléphone ?*

`Informative` · `15 min d'analyse` · `Site projet` + `Annuaire prestataires`

Pourquoi cette question

Le support n'est pas la même chose que la capacité de mise en œuvre. Q10 répond à *« qui déploie et maintient au quotidien »* ; Q11 répond à *« qui intervient quand ça casse en production, hors heures ouvrées, sur un problème critique »*. Pour un outil périphérique, la réponse peut être *« on attend lundi matin »* — pas de problème. Pour un outil critique, la réponse doit être *« quelqu'un répond en moins de quatre heures, vingt-quatre heures sur vingt-quatre »* — ce qui change radicalement le budget et le profil de prestataire.

Cette question est **informative au niveau grille** parce qu'elle ne disqualifie jamais un projet seule : on peut toujours acheter du support si on en a les moyens. Mais elle est **de facto sérieuse pour une PME sans équipe IT 24/7**, parce que c'est la question qui détermine si l'outil peut tenir sur un usage critique sans risque opérationnel inacceptable.

Comment y répondre

Quatre niveaux de support possibles, à connaître avant de choisir :

> **Communauté seule** — forums, Discord, GitHub Issues, mailing list. Gratuit, mais aucune SLA. Réponse en quelques heures à quelques semaines selon la popularité du projet et la difficulté du problème. Acceptable pour un outil périphérique ou pour une équipe IT compétente qui sait diagnostiquer seule.

Prestataire avec contrat de support — intégrateur ou société de service avec engagement contractuel (heures ouvrées, ou 24/7 selon la formule). SLA mesurables — temps de réponse, temps de résolution. C'est le standard PME pour un outil critique. Coût annuel typique : cinq à vingt mille euros selon la formule.

Mainteneur du projet — pour les éditeurs open source commerciaux, contrat de support direct avec l'entreprise mainteneur. Accès aux développeurs eux-mêmes, parfois patches custom. Coût plus élevé : dix à cinquante mille euros annuels selon l'outil et le périmètre.

SaaS managé chez le mainteneur ou un tiers — la question du support disparaît : le fournisseur SaaS prend la responsabilité opérationnelle. Coût mensuel selon usage, mais simplification radicale du modèle. Modèle pertinent pour les PME sans équipe IT.

TAB. 13 — SIGNAUX Q11 — SUPPORT DISPONIBLE

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
Communauté active avec réponses sous 24h sur les questions courantes	Forums déserts, dernières réponses datant de plusieurs mois
Au moins 2 prestataires français proposent du support sur l'outil	Aucun prestataire français identifié, support assuré uniquement en anglais
SLA contractualisables (temps de réponse, temps de résolution)	Aucune formalisation des engagements de service
Documentation officielle complète, à jour, en plusieurs langues	Documentation rare, fragmentaire, datée

Exemple appliqué

PostgreSQL offre les quatre niveaux : communauté massive et réactive (Stack Overflow, mailing lists), écosystème de prestataires français dense (Dalibo, EDB France, freelances spécialisés), support direct chez les éditeurs commerciaux (EDB, Crunchy Data), et SaaS managé multi-fournisseurs (AWS RDS, Google Cloud SQL, Scaleway, OVH). Quelle que soit la criticité et le budget, il existe une réponse adaptée — c'est l'un des atouts structurels de l'outil.

À comparer à un projet open source de niche typique : la seule réponse possible est « *GitHub Issues* », sans SLA, sans alternative locale, sans formule managée. Acceptable pour un outil périphérique, intenable pour un outil critique.

◆ **Verdict Q11 — Informative (mais de facto sérieuse pour PME sans IT)**

Au niveau de la grille, Q11 ne disqualifie pas — elle oriente le budget et le choix de prestataire. Mais sur un usage critique dans une PME sans équipe IT 24/7, elle bascule de fait en sérieuse : sans réponse claire à *« qui répond à deux heures du matin »*, l'outil ne peut pas être déployé en production critique sans risque inacceptable. À combiner avec Q2 (criticité) et Q10 (capacité) pour décider si le profil de support disponible correspond au profil d'usage.

6.3 Q12 — Quel TCO sur cinq ans ?

Sous-titre : *Coût total réel, pas « c'est gratuit donc zéro euro »*

`Sérieuse` · `60 min d'analyse` · `Audit interne` + `Devis prestataires`

Pourquoi cette question

Le mythe le plus tenace de l'open source : *« c'est gratuit »*. Le code l'est, certes. Le déploiement, la formation, la maintenance, l'hébergement, le support, la sortie — non. Le TCO (Total Cost of Ownership, coût total de possession) d'un outil open source sur cinq ans est rarement nul, et souvent du même ordre de grandeur qu'un outil propriétaire équivalent — parfois inférieur, parfois supérieur, jamais nul.

Calculer le TCO avant l'adoption, c'est ce qui distingue une décision rationnelle d'une décision émotionnelle. Et c'est aussi ce qui permet de comparer honnêtement à une alternative propriétaire — sans tomber dans le piège *« open source = gratuit donc forcément mieux »*.

Comment y répondre

Six lignes de coût à estimer sur cinq ans, dans cet ordre :

"01" Licence évitée

- ✓ Coût annuel des licences propriétaires alternatives (SAP, Oracle, Microsoft, etc.)
- ✓ C'est ce que l'open source vous fait économiser
- ✓ Souvent la ligne la plus visible — et la plus surestimée

"02" Déploiement initial

- ✓ Installation, configuration, intégration aux systèmes existants
- ✓ Inclut le temps de cadrage, les tests, la mise en production
- ✓ De quelques milliers à plusieurs dizaines de milliers d'euros

"03" Formation

- ✓ Formation initiale des utilisateurs et de l'équipe IT
- ✓ Formation continue sur les montées de version majeures
- ✓ Souvent oubliée dans les premières estimations

"04" Maintenance et support

- ✓ Contrat de support (cf. Q11) — annuel récurrent
- ✓ Temps interne dédié aux montées de version, patches, monitoring
- ✓ Ligne récurrente la plus lourde sur 5 ans

"05" Hébergement

- ✓ Serveurs cloud ou on-premise, sauvegardes, sécurité
- ✓ Croît avec le volume de données et d'utilisateurs
- ✓ Souvent sous-estimée pour les outils gourmands

"06" Sortie ou migration

- ✓ Coût d'export des données, de migration vers un autre outil en fin de vie
- ✓ Inclut la formation aux nouveaux process, la double-saisie pendant transition
- ✓ À budgéter même si on espère ne jamais l'engager

L'open source économise systématiquement la ligne 1 et souvent une partie de la ligne 4 (pas de support obligatoire). Il ne change rien aux lignes 2, 3, 5. Il peut renchérir la ligne 6 si la communauté de prestataires est étroite. Le solde sur cinq ans est généralement

favorable à l'open source pour les outils structurants — mais l'écart est rarement aussi spectaculaire que les discours marketing le suggèrent, et il peut être négatif sur des outils de niche mal supportés.

TAB. 14 — SIGNAUX Q12 — TCO SUR 5 ANS

✅ BONS SIGNAUX	❌ MAUVAIS SIGNAUX
TCO calculé sur 5 ans, toutes lignes documentées	TCO réduit à « c'est gratuit » sans calcul réel
Comparaison honnête avec l'alternative propriétaire	Biais open source : seules les lignes favorables comptées
Sortie budgétée même si peu probable	Coût de sortie ignoré ou marginalisé
Marge de sécurité de 20 à 30% sur l'estimation initiale	Estimation au plus juste, sans aléa budgétaire

Exemple appliqué

Une PME de cent personnes compare **Odoo Community** (open source, intégrateur français, hébergement Scaleway) à **SAP Business One** (propriétaire, partenaire SAP local, hébergement SAP HANA Cloud) sur cinq ans, en cœur ERP. Les ordres de grandeur typiques observés en mission : pour Odoo, économie de licence d'environ vingt à trente mille euros par an sur cinq ans (cent à cent cinquante mille euros au total), à mettre en balance avec un déploiement initial entre trente et soixante mille euros, une maintenance annuelle de cinq à quinze mille euros, et un coût de sortie potentiellement plus élevé si l'on doit migrer vers un autre ERP.

Le solde reste favorable à Odoo dans la majorité des configurations, mais l'écart réel est de l'ordre de trente à cinquante pour cent, pas de quatre-vingt-dix pour cent comme un discours superficiel pourrait le laisser croire. Et cet écart ne se matérialise que si Q4 à Q11 ont été correctement validées en amont — sinon, les surcoûts de défaillance opérationnelle annulent vite l'économie de licence.

▲ **Verdict Q12 — Sérieuse**

Le TCO n'est pas nul. C'est généralement favorable à l'open source sur les outils structurants, mais l'écart est souvent moindre que ne le suggèrent les discours marketing, et il dépend entièrement de la qualité des réponses aux questions précédentes. Sur un outil critique, exiger un calcul TCO complet sur 5 ans avant adoption, avec marge de sécurité de 20 à 30%. C'est ce qui permet une comparaison honnête avec l'alternative propriétaire et évite les abandons à dix-huit mois pour cause de coûts cachés sous-estimés.

◆ **Récapitulatif Partie 6 — Exécution**

Trois questions, dont **deux sérieuses** (Q10 capacité, Q12 TCO) et **une informative au niveau grille mais de facto sérieuse pour une PME sans IT** (Q11 support). C'est la dimension la plus opérationnelle de la grille, et celle qui décide souvent du succès terrain de l'adoption. Sans capacité de mise en œuvre identifiée, sans support proportionné à la criticité, sans TCO calculé honnêtement, un projet open source techniquement excellent peut quand même échouer chez vous — pas à cause du code, mais à cause de l'écosystème d'exécution. La grille est complète : place au guide décisionnel.

Annexe — Panorama des licences open source

Cette annexe traite en profondeur ce que les Q8 et Q9 abordent en mode décision rapide. Elle répond à la promesse faite dans l'article #1 du blog MP-i (panorama des licences open source). Destinée aux décideurs qui veulent comprendre le paysage avant de choisir, aux CTO qui cadrent leur politique de stack, aux intégrateurs qui analysent les licences de leurs clients. Elle peut être lue de façon autonome.

7.1 Les vraies licences OSI — cartographie des familles

L'Open Source Initiative maintient une liste officielle de plus de cent licences approuvées. Elles se répartissent en trois familles selon leur degré de copyleft — c'est-à-dire l'obligation de redistribuer les modifications sous la même licence.

Licences permissives — aucune obligation de copyleft. Vous pouvez utiliser, modifier, redistribuer le code dans un produit propriétaire, en SaaS, dans une application commerciale, sans publier vos modifications.

Les licences MIT, Apache 2.0 et BSD (2-Clause et 3-Clause) sont les représentants canoniques. MIT est la plus populaire sur GitHub — c'est la licence de Node.js, Ruby on Rails, React, jQuery. Apache 2.0 ajoute une clause de brevet explicite : les contributeurs accordent un droit de brevet sur leurs contributions. BSD-3-Clause interdit l'utilisation du nom du projet pour promouvoir des dérivés sans autorisation. Pour le décideur, toutes les licences permissives sont équivalentes dans leurs effets pratiques : liberté totale d'usage commercial.

Licences copyleft faibles — obligation de copyleft limitée aux fichiers ou composants que vous modifiez. Mozilla Public License 2.0 (MPL-2.0) et GNU Lesser GPL (LGPL-2.1 et 3.0) sont les deux exemples principaux. MPL-2.0 est une licence fichier-par-fichier : les fichiers MPL doivent rester MPL si vous les redistribuez, mais votre code propriétaire peut les appeler sans être affecté.

LGPL est une licence bibliothèque : vous pouvez utiliser une bibliothèque LGPL dans un produit propriétaire si vous la liez dynamiquement et fournissez les modifications. Terraform était sous MPL 2.0 jusqu'en août 2023 — ce qui permettait à toutes les

entreprises de l'intégrer dans leurs produits commerciaux sans obligation de publier leur code.

Licences copyleft fort — obligation de copyleft sur tout le programme. Si vous redistribuez un logiciel GPL, votre code qui s'intègre avec lui doit être publié sous GPL.

GNU GPL v2 (Linux), GPL v3 (FSF et nombreux projets), GNU Affero GPL v3 (AGPL-3.0) en sont les représentants. AGPL-3.0 est le cas le plus restrictif : l'obligation de publier le code source s'applique même si le logiciel est utilisé *via le réseau* — c'est-à-dire en SaaS. Redis 8 a adopté AGPLv3 en mai 2025 pour cette raison précise : AGPL oblige les SaaS managés à publier leurs modifications, ce que le simple GPL ne couvre pas. Pour le décideur, AGPL est le signal qu'il faut consulter un juriste avant d'intégrer le logiciel dans un produit SaaS.

TAB. 15 — TROIS FAMILLES DE LICENCES OSI — REPÈRES PRATIQUES

FAMILLE	EXEMPLES	LIBERTÉ COMMERCIALE	OBLIGATION COPYLEFT	CAS D'USAGE PME TYPIQUE
Permissive	MIT, Apache 2.0, BSD, PostgreSQL Licence	Totale — usage, redistribution, intégration propriétaire	Aucune	Tous usages : intégration produit, SaaS, redistribution
Copyleft faible	MPL-2.0, LGPL-2.1, LGPL-3.0	Large — usage SaaS et commercial OK	Sur les fichiers modifiés uniquement	Bibliothèques intégrées dans un produit mixte
Copyleft fort	GPL-2.0, GPL-3.0, AGPL-3.0	Restreinte pour SaaS (AGPL)	Sur tout le programme dérivé	Usage interne, ou produit entièrement libre

Ressource de référence : `opensource.org/licenses` — liste complète, classée par catégorie et par critères de l'OSD (Open Source Definition). C'est la seule source canonique.

7.2 Les fausses licences : source-available déguisé en open source

Depuis 2018, une nouvelle catégorie de licences est apparue sur le marché — dite *source-available*. Le code est visible, parfois modifiable, mais des restrictions commerciales empêchent certains usages courants.

Ces licences **ne sont pas reconnues par l'OSI** et ne satisfont pas aux dix critères de l'*Open Source Definition*. Leurs éditeurs les présentent pourtant avec le vocabulaire *« open »*, *« community edition »*, *« free tier »* — d'où la confusion.

SSPL (Server Side Public License) — créée par MongoDB en octobre 2018. Clause centrale : si vous exploitez un service SaaS utilisant le code SSPL, vous devez publier en open source *l'intégralité de l'infrastructure de ce service* — pas seulement vos modifications, mais tous les composants qui font tourner le service (monitoring, load balancing, déploiement, etc.). Clause intentionnellement dissuasive contre les hyperscalers. MongoDB a demandé l'approbation OSI en 2018, puis a retiré sa demande en 2019 quand il est devenu clair qu'elle serait refusée.

L'OSI a qualifié SSPL de licence *« fauxpen »* — contraction des mots *« faux open source »*. Elastic a adopté SSPL comme option en janvier 2021, puis ajouté AGPLv3 (vraie OSI) en septembre 2024.

BSL (Business Source License v1.1) — créée initialement par MariaDB. HashiCorp a basculé Terraform sous BSL le 10 août 2023. Principe : le code est utilisable librement jusqu'à un certain seuil commercial (défini dans le paramètre *« Additional Use Grant »*). Au-delà du seuil, l'usage commercial est restreint — notamment l'offre de Terraform comme service managé concurrent de HashiCorp. Après une période de conversion (typiquement trois à quatre ans), le code bascule automatiquement vers une vraie licence open source (Apache 2.0 pour HashiCorp).

Le BSL n'est pas approuvé par l'OSI. L'ambiguïté de la clause *« competitive »* est sa principale critique — personne ne sait exactement où commence la compétition.

Elastic License v2 (ELv2) — licence propriétaire simplifiée de Elastic, introduite en même temps que le passage SSPL en janvier 2021. Trois interdictions : ne pas offrir le produit comme service managé ; ne pas contourner les features payantes ; ne pas supprimer les marques Elastic. Pas OSI. C'est sous ELv2 que tournent les distributions binaires d'Elasticsearch et Kibana depuis 2021.

RSAL (Redis Source Available License v2) — licence source-available de Redis Ltd., introduite en mars 2024 conjointement avec SSPL. Interdit les services cloud managés concurrents. Pas OSI. Redis 8 a ajouté AGPLv3 comme troisième option en mai 2025 — permettant un usage véritablement open source, mais sous AGPL (copyleft fort).

Le point commun de ces licences : elles permettent toutes de *voir* le code, parfois de le modifier pour usage interne, mais elles bloquent l'usage en SaaS concurrent ou l'intégration dans certains produits commerciaux. Pour un décideur, la règle est simple : si la licence n'est pas sur `opensource.org/licenses`, elle est source-available — traiter comme propriétaire.

7.3 Frise chronologique des relicences majeures (2018-2026)

OCTOBRE 2018

MongoDB passe sous SSPL v1

Premier éditeur open source majeur à adopter une licence source-available anti-cloud. MongoDB retire sa demande d'approbation OSI en 2019. L'OSI qualifie SSPL de licence « fauxpen ».

14 JANVIER 2021

Elasticsearch et Kibana passent sous SSPL + Elastic License v2

Elastic annonce le passage d'Apache 2.0 vers la double licence SSPL / ELv2, applicable depuis la version 7.11. AWS annonce le fork OpenSearch dès le 21 janvier 2021. OpenSearch transféré à la Linux Foundation en septembre 2024.

10 AOÛT 2023

Terraform passe sous BSL 1.1

HashiCorp change de licence pour Terraform (et Vault, Nomad, etc.) — de MPL 2.0 vers BSL 1.1. Le 15 août, le manifeste OpenTF est publié. Le 25 août, le fork est annoncé. Le 20 septembre 2023, le projet rejoint la Linux Foundation sous le nom OpenTofu.

SEPTEMBRE 2023

Fork OpenTofu accepté par la Linux Foundation (20 septembre)

41 jours après l'annonce de HashiCorp, le fork OpenTofu — sous MPL 2.0 d'origine de Terraform — rejoint la Linux Foundation. Première version stable en janvier 2024. Accepté par la CNCF en avril 2025.

20 MARS 2024

Redis passe sous SSPL v1 + RSALv2

Redis Ltd. abandonne BSD 3-Clause pour une double licence source-available. Le 28 mars 2024, la Linux Foundation accepte le fork Valkey huit jours plus tard — réponse communautaire la plus rapide enregistrée. IBM rachète HashiCorp (annoncé avril 2024, clôturé février 2025).

SEPTEMBRE 2024

Elastic réintroduit AGPLv3 (OSI) pour Elasticsearch et Kibana

Elastic ajoute AGPLv3 comme troisième option de licence aux côtés de SSPL et ELv2, applicable à partir de la version 8.16. Première vraie licence OSI pour Elasticsearch depuis janvier 2021. OpenSearch est transféré à la Linux Foundation le même mois.

1ER MAI 2025

Redis 8 sort en tri-licence : SSPL + RSAL + AGPLv3

Redis Ltd. ajoute AGPLv3 comme troisième option, sous l'impulsion de Salvatore Sanfilippo (créateur de Redis, revenu chez Redis Ltd. comme développeur en novembre 2024). Valkey continue de progresser — deux mouvements de licence en 14 mois n'ont pas rétabli la confiance perdue en mars 2024.

2026

Trois forks fondation actifs en production

OpenSearch (Linux Foundation, sept. 2024), OpenTofu (Linux Foundation, sept. 2023), Valkey (Linux Foundation, mars 2024) sont tous les trois stables et en production active. Le modèle « fork sous fondation neutre » s'impose comme la réponse systémique aux relicences single-vendor.

7.4 Les trois cas décortiqués

Elasticsearch (2021) : le pionnier malheureux

Elastic a annoncé le 14 janvier 2021, en publiant un billet intitulé *« Doubling down on open »* — titre ironiquement opposé au sens de la décision — que Elasticsearch et Kibana passeraient d'Apache 2.0 vers une double licence SSPL/ELv2, applicable depuis la version 7.11.

Le motif déclaré : empêcher AWS d'offrir Elasticsearch comme service managé sans contribuer au projet ni payer Elastic. AWS proposait en effet un service Elasticsearch sur AWS sans reverse engineering ni contribution en retour — ce que la licence Apache 2.0 autorisait légalement.

La communauté réagit immédiatement. Le 21 janvier 2021, AWS annonce le fork OpenSearch, maintenu sous Apache 2.0. En septembre 2024, OpenSearch est transféré à la Linux Foundation.

Le résultat pour Elastic : si l'objectif était de ralentir AWS, il a produit l'exact opposé — AWS dispose désormais d'un fork propre, communautaire, sous fondation neutre, que toutes les entreprises qui refusaient SSPL ont migré. En septembre 2024, Elastic admet implicitement l'échec de la stratégie en ajoutant AGPLv3 comme troisième option aux côtés de SSPL et ELv2 — la même version qui voit OpenSearch passer sous Linux Foundation.

Terraform (2023) : la relicence la plus choquante

Le 10 août 2023, HashiCorp annonce — *« with little or no advance notice »*, selon les propres termes du manifeste OpenTofu — que Terraform, après neuf ans sous MPL 2.0, bascule sous BSL 1.1. Avec Terraform, HashiCorp touche un outil d'infrastructure critique pour des dizaines de milliers d'entreprises, intégré dans des pipelines CI/CD, des modules tiers, des outils commerciaux. La restriction BSL cible les offres de Terraform-as-a-Service — Spacelift, env0, Scalr et d'autres plateformes commerciales qui utilisaient Terraform dans leur offre managée.

La réponse communautaire est la plus organisée observée à ce jour : dans les dix jours suivant l'annonce, le manifeste OpenTF réunit plus de 32 000 étoiles GitHub et 140 entreprises signataires. Le fork OpenTofu est public le 5 septembre, accepté par la Linux Foundation le 20 septembre 2023 — 41 jours après l'annonce de HashiCorp.

Le BSL crée une ambiguïté juridique persistante sur ce que signifie *« concurrence »* avec les offres HashiCorp, ce qui affecte aussi les intégrateurs et les outils adjacents.

Redis (2024) : la relicence la plus rapide et la plus symbolique

Le 20 mars 2024, Redis Ltd. annonce le passage de BSD 3-Clause vers SSPL v1 + RSAL v2. Redis n'est pas seulement populaire — c'est l'un des outils les plus déployés dans les architectures web mondiales, utilisé comme cache, broker de messages, session store. BSD 3-Clause est la licence la plus permissive qui soit : aucune restriction, pas même de copyleft.

Passer directement de BSD à SSPL en une seule décision, sans transition, est perçu comme une rupture de confiance particulièrement violente.

La réponse record : le 28 mars 2024 — *huit jours plus tard* — la Linux Foundation accepte le projet Valkey, fork Redis basé sur la dernière version BSD 3-Clause (7.2.4). Le fork est porté par AWS, Google Cloud, Oracle, Ericsson, Snap. Valkey 7.2.5 sort en mai 2024. De nombreuses distributions Linux (Fedora, Debian) migrent vers Valkey dans les mois qui suivent.

IBM achète HashiCorp (dont Terraform) en avril 2024. La tentative de retour partiel de Redis avec l'ajout d'AGPLv3 en mai 2025 n'a pas rétabli la confiance : Valkey continue de progresser.

7.5 Licences hors logiciel : Creative Commons et OSHWA

Pour les décideurs concernés par des projets open source non-logiciels, deux familles de licences complètent le panorama.

Creative Commons (CC) — licences pour la documentation, les contenus, les designs. Elles ne s'appliquent pas au code logiciel (la CC elle-même le déconseille explicitement). Six variantes principales, combinées de quatre modules : BY (attribution obligatoire), SA (partage à l'identique — copyleft), NC (pas d'usage commercial), ND (pas de dérivés). CC0 est la dédicace au domaine public — aucune obligation. CC BY-SA est l'équivalent copyleft pour les contenus.

Pour une PME, l'usage courant est CC BY ou CC BY-SA pour la documentation interne partagée. Pour les données ouvertes, deux écosystèmes distincts coexistent : les licences CC (textes, médias, statistiques publiques) et les licences spécifiques aux bases de données (ODbL — Open Database License, portée par l'Open Knowledge Foundation, utilisée notamment par OpenStreetMap). Les licences CC sont soutenues par la Creative Commons Foundation et largement utilisées dans l'édition, la cartographie et les commons numériques.

OSHWA (Open Source Hardware Association) — certification et bonnes pratiques pour le hardware open source (schémas électroniques, fichiers CAO, PCB). La certification OSHWA s'applique aux conceptions matérielles. Pas d'équivalent unique à l'OSI pour le hardware — les projets utilisent souvent une combinaison de licences logicielles pour le firmware, de CERN Open Hardware Licence pour les schémas, et de CC pour la documentation.

Pour les décideurs confrontés à l'IoT, à l'électronique industrielle ou à la fabrication, vérifier la certification OSHWA est l'équivalent de vérifier l'approbation OSI pour le logiciel.

7.6 CLA, DCO et marques déposées — pourquoi ça change tout

Deux mécanismes juridiques souvent ignorés par les décideurs ont une importance stratégique directe sur le risque de relicence.

CLA (Contributor License Agreement) — accord que chaque contributeur externe signe avant que sa contribution soit acceptée. Il existe deux types radicalement différents. Le premier est un *copyright assignment* : le contributeur cède l'intégralité de ses droits à l'entreprise mainteneur. C'est le mécanisme qu'utilisaient Elastic, HashiCorp et Redis Ltd. avant leurs relicences — c'est précisément ce qui a *permis* ces relicences. L'entreprise possédait l'intégralité du copyright sur le code communautaire, et pouvait donc le relicencier sans demander l'accord des contributeurs.

Le second type de CLA ne transfère que des droits de licence, sans céder le copyright — moins risqué pour les contributeurs, mais sans signification sur le risque de relicence pour les utilisateurs.

DCO (Developer Certificate of Origin) — approche alternative utilisée par la Linux Foundation (Linux, OpenTofu, Valkey). Le contributeur certifie simplement qu'il a le droit de contribuer et que sa contribution peut être intégrée sous la licence actuelle du projet. Il ne cède aucun copyright. Résultat : personne ne peut imposer une relicence sans l'accord de chaque contributeur — ce qui rend la relicence unilatérale structurellement impossible pour un projet multi-contributeurs sous DCO.

Marques déposées — un point souvent négligé. L'entreprise mainteneur peut posséder la marque déposée du projet (« *Elasticsearch* », « *Redis* », « *Terraform* ») indépendamment du code. Quand le fork Valkey est créé, la marque « *Redis* » reste la propriété de Redis Ltd. — le fork doit adopter un nouveau nom. Même chose pour OpenSearch (pas « *Elasticsearch* ») et OpenTofu (pas « *Terraform* »). Pour un décideur qui évalue un projet single-vendor, demander « *qui possède la marque déposée ?* » est aussi important que « *qui possède le code ?* ».

7.7 Mini-checklist licence pour le décideur

Six questions à poser sur n'importe quel projet open source avant adoption, dans l'ordre :

La licence est-elle dans la liste OSI ? → `opensource.org/licenses`. Si non : source-available, traiter comme propriétaire.

Quelle famille de copyleft ? → Permissive (MIT/Apache/BSD) = liberté totale. Copyleft faible (MPL/LGPL) = OK avec précautions. Copyleft fort (GPL/AGPL) = consultation juridique avant SaaS.

Le projet utilise-t-il un CLA avec cession de copyright ? → Si oui : relicence unilatérale possible par l'éditeur. Signal de risque combiné à Q4 single-vendor.

Ou un DCO seul (Linux Foundation) ? → Relicence unilatérale structurellement impossible.

Qui possède la marque déposée ? → Si l'éditeur unique possède la marque, le fork devra changer de nom.

Y a-t-il un précédent de relicence vers source-available ? → Signal fort (Q9). Un deuxième mouvement est toujours possible.

◆ **À retenir — Annexe**

La frontière entre open source et source-available n'est pas un détail juridique — c'est une ligne stratégique. Une licence est OSI ou elle ne l'est pas. Le CLA avec cession de copyright est le mécanisme technique qui permet les relicences unilatérales. Le DCO est le mécanisme qui les empêche. Comprendre ces trois points suffit à éviter quatre-vingt pour cent des pièges relevés dans les cas Elastic, HashiCorp et Redis.

Partie 8 — Internaliser ou externaliser la mise en œuvre

La grille a répondu à « *quel outil choisir* ». Cette partie répond à « *qui s'en occupe* » — question distincte, souvent tranchée trop vite, qui conditionne le succès opérationnel autant que le choix technique. Elle s'adresse aux décideurs qui ont passé les douze questions et veulent cadrer leur modèle de mise en œuvre avant de signer quoi que ce soit.

8.1 Internaliser — le modèle le plus résilient

Internaliser signifie que votre entreprise porte la responsabilité technique complète : déploiement initial, configuration, montées de version, patches de sécurité, dépannage, formation des nouveaux arrivants. C'est le modèle le plus résilient sur le long terme — et le plus coûteux à mettre en place.

> ▲ **Trois conditions cumulatives pour internaliser**
>
> L'internalisation n'est viable que si les **trois** conditions ci-dessous sont réunies. Si l'une manque, basculer sur prestataire ou SaaS.

- ✓ **Compétence interne réelle et documentée** — pas « quelqu'un qui a lu la doc », mais quelqu'un qui a déployé l'outil en production au moins une fois, connaît ses points de fragilité, et peut diagnostiquer les incidents courants sans aide extérieure.
- ✓ **Temps budgété explicitement** — la maintenance open source n'est pas gratuite en temps. Une montée de version majeure d'un ERP représente plusieurs jours de travail ; un patch de sécurité urgent peut interrompre n'importe quelle journée. Sans budget-temps dédié, la compétence interne se dilue dans la charge opérationnelle quotidienne.
- ✓ **Plan de succession documenté** — que se passe-t-il si la personne qui porte la compétence quitte l'entreprise ? La réponse « on cherchera quelqu'un » n'est pas un plan. Documentation des procédures, partage des accès, double compétence minimale.

- ✓ **ETI et grandes entreprises** avec une équipe IT structurée

- ✓ **Projets à personnalisation profonde** — modules métier spécifiques, intégrations complexes
- ✓ **Outils en cœur de production** sur un horizon de dix ans ou plus

8.2 Le modèle prestataire — le standard PME

Confier la mise en œuvre à un prestataire externe — intégrateur, SSII, indépendant spécialisé — est le modèle dominant pour les PME. Il transfère la responsabilité opérationnelle à une entité dont c'est le métier, sans exiger de compétence interne profonde.

AVANTAGES

Trois avantages structurels

Expérience cumulée — le prestataire a déjà vu les problèmes que vous n'avez pas encore rencontrés sur des projets similaires. **Écosystème de montée en compétence** — formations, certifications, participation aux communautés, mutualisé sur plusieurs clients. **Capacité d'absorption des pics** — montée de version ou incident critique sans perturber la production interne.

RISQUES

Trois risques à gérer

Lock-in prestataire — vérifier qu'au moins deux autres prestataires du marché pourraient reprendre le projet. **Perte de connaissance interne** — un minimum de documentation et de formation interne reste nécessaire. **Coût de sortie** — prévoir un transfert de connaissance explicite dans les contrats.

8.3 SaaS managé — simplification radicale, dépendance nouvelle

Pour les outils qui le proposent (Nextcloud, Odoo, GitLab, Mattermost…), le SaaS managé chez le mainteneur ou un hébergeur tiers est une troisième voie.

La question des montées de version, des patches de sécurité et de la disponibilité est traitée par le fournisseur. Le décideur ne gère plus que l'usage, pas l'infrastructure.

- › **Quand le SaaS managé est pertinent** — PME sans équipe IT dédiée, outils périphériques où la charge de maintenance interne n'est pas justifiée, phases de démarrage où l'urgence est de lancer, pas d'optimiser l'infrastructure.

- **Le coût qui rattrape** — mensuel récurrent, croissant avec l'usage, dépasse souvent sur cinq ans le coût d'un déploiement auto-hébergé bien maintenu. À réintégrer dans le TCO de Q12.
- **La dépendance qui se déplace** — dépendre d'un fournisseur SaaS crée une nouvelle forme de lock-in. Sur un outil critique, vérifier Q4 (gouvernance) et Q9 (historique de relicence) du fournisseur SaaS lui-même, pas seulement du projet open source sous-jacent.

8.4 Matrice de décision rapide

TAB. 16 — QUEL MODÈLE POUR QUEL PROFIL

PROFIL D'ENTREPRISE	OUTIL CRITIQUE	OUTIL PÉRIPHÉRIQUE
ETI avec équipe IT structurée	Internalisation ou prestataire long terme	Prestataire ou SaaS
PME 20-100 personnes, IT généraliste	Prestataire spécialisé avec SLA	SaaS managé
TPE ou startup sans IT dédié	SaaS managé uniquement	SaaS managé
Éditeur logiciel intégrant l'outil	Internalisation obligatoire	Prestataire ponctuel

◆ **À retenir — Partie 8**

La question « *qui s'en occupe* » est aussi importante que « *quel outil choisir* ». Sur un outil critique, un prestataire identifié et un contrat signé **avant** la mise en production ne sont pas optionnels. Sur un outil périphérique, le SaaS managé simplifie radicalement l'équation opérationnelle — au prix d'un coût récurrent à calculer dans le TCO (Q12).

Guide décisionnel

Douze questions posées, quatre dimensions évaluées. Cette section traduit les réponses en verdict final — comment agréger les signaux, quel seuil de rejet appliquer, et comment ça se passe concrètement sur trois projets réels ou représentatifs.

Seuils de rejet

Le verdict de la grille repose sur deux règles simples, applicables sans calcul complexe.

Règle 1 — Éliminatoire : un seul échec sur une question éliminatoire (Q1, Q3, Q4, Q5, Q8) suffit à écarter le projet. Ces questions mesurent des ruptures non négociables — sans horizon défini, sans gouvernance saine, sans licence OSI, sans activité de code, l'évaluation ne peut pas continuer. Il n'y a pas de compensation possible.

Règle 2 — Cumul sérieux : trois échecs ou plus sur les questions sérieuses (Q2, Q6, Q7, Q9, Q10, Q12) déclenchent un rejet par effet d'amplification. Un seul signal sérieux est un signal de vigilance. Deux signaux sérieux convergents sur la même dimension (par exemple Q4 single-vendor + Q6 concentration + Q7 SaaS managé) sont un signal d'alerte fort. Trois signaux sérieux, quelle que soit leur dimension, produisent un verdict *Déconseillé* — l'adoption est possible mais exige des compensations explicites pour chaque signal (plan de sortie documenté, prestataire alternatif identifié, horizon réduit).

TAB. 17 — GRILLE DE VERDICT GLOBAL

SITUATION	VERDICT	ACTION RECOMMANDÉE
0 éliminatoire + 0 à 2 sérieuses	[Recommandé]	Procéder à l'adoption selon le modèle défini en Partie 8
0 éliminatoire + 3 à 4 sérieuses	[Déconseillé]	Adoption possible avec compensations explicites documentées
0 éliminatoire + 5 à 6 sérieuses	[À écarter]	Trop de fragilités cumulées — chercher une alternative
1 éliminatoire ou plus	[Stop]	Écarter immédiatement, quelle que soit la note sur les sérieuses

Cas appliqué 1 — Odoo Community pour un ERP PME

Contexte : PME de soixante personnes, secteur BTP, cherche un ERP pour remplacer un outil vieillissant. Horizon dix ans, outil critique (facturation, projets, achats).

TAB. 18 — RÉSULTATS Q1-Q12 — ODOO COMMUNITY ERP PME

QUESTION	RÉSULTAT	VERDICT
Q1 — Horizon temporel	Horizon défini — dix ans	✅
Q2 — Criticité	Criticité élevée — facturation, projets, achats	⚠️ sérieuse
Q3 — Équivalent mature	Odoo est l'équivalent mature pour ce besoin	✅
Q4 — Gouvernance	Odoo SA gouvernance ouverte, code LGPL communautaire	✅
Q5 — Activité du dépôt	Releases trimestrielles, milliers de commits	✅
Q6 — Diversité contributrice	Odoo SA concentre les commits, mais gouvernance ouverte	⚠️ sérieuse
Q7 — Modèle économique	Open core documenté, Odoo SA profitable	✅
Q8 — Type de licence	LGPL-3.0 OSI ✅	✅
Q9 — Historique relicence	Aucune relicence vers source-available	✅
Q10 — Capacité mise en œuvre	Intégrateurs Odoo disponibles en France	✅
Q11 — Support	Prestataire + communauté large (informatif)	✅ informatif
Q12 — TCO	Déploiement 40 k€, maintenance 8 k€/an, économie ~25 k€/an vs propriétaire	✅

Bilan : 0 éliminatoire, 2 sérieuses (Q2 criticité + Q6 diversité). Verdict : **Recommandé** — avec prestataire spécialisé identifié avant mise en production (Q2 criticité élevée) et suivi des signaux de gouvernance Odoo SA à chaque version majeure (Q6).

Cas appliqué 2 — Outil de niche SIG terrain pour une collectivité

Contexte : collectivité de taille moyenne cherche un outil open source pour la gestion de ses réseaux d'eau (inspection, incidents, cartographie terrain). Horizon cinq ans, usage sérieux mais pas critique au sens « *arrêt d'activité immédiat* ».

TAB. 19 — RÉSULTATS Q1-Q4 — AUDIT ARRÊTÉ À Q4 (ÉLIMINATOIRE)

QUESTION	RÉSULTAT	VERDICT
Q1 — Horizon temporel	Horizon cinq ans défini	✅
Q2 — Criticité	Criticité modérée — perte de visibilité réseaux, 5h d'impact	⚠️ sérieuse
Q3 — Équivalent mature	Pas d'équivalent généraliste — projet de niche justifié par trois raisons métier	✅
Q4 — Gouvernance	Mainteneur unique, PME française, pas de fondation, CLA contributor	❌ éliminatoire — audit arrêté

Bilan : 1 éliminatoire (Q4). Verdict **Stop** — la gouvernance single-vendor avec CLA sur un outil critique de cinq ans est inacceptable. Options recommandées : chercher un fork ou un équivalent sous fondation (OSGeo propose des projets géomatiques), ou définir un horizon court (18 mois) et accepter le risque en connaissance de cause avec plan de sortie documenté.

Cas appliqué 3 — Outil source-available détecté en évaluation

Contexte : éditeur logiciel B2B envisage d'intégrer un outil de recherche full-text dans son produit SaaS. L'outil est présenté comme « *open source* » dans les benchmarks. Horizon : cinq ans, critique (cœur du produit).

TAB. 20 — RÉSULTATS Q1-Q8 — AUDIT ARRÊTÉ À Q8 (ÉLIMINATOIRE)

QUESTION	RÉSULTAT	VERDICT
Q1 — Horizon temporel	Horizon cinq ans défini	✅
Q2 — Criticité	Critique — cœur du produit SaaS	⚠️ sérieuse
Q3 — Équivalent mature	Elasticsearch (Apache 2.0 jusqu'en 2021) existait comme équivalent mature	✅
Q4 — Gouvernance	Single-vendor, CLA contributor	⚠️ sérieuse
Q5 — Activité du dépôt	Activité correcte	✅
Q6 — Diversité contributrice	Concentration > 80% commits	⚠️ sérieuse
Q7 — Modèle économique	SaaS managé sous pression hyperscaler	⚠️ sérieuse
Q8 — Type de licence	Licence non-OSI — ELv2 + SSPL	❌ éliminatoire — audit arrêté

Bilan : 1 éliminatoire (Q8). Verdict **Stop** — sous licence ELv2, un éditeur SaaS ne peut pas intégrer cet outil dans son produit sans risque juridique. Options : OpenSearch (Apache 2.0, Linux Foundation) ou Typesense (GPL-3.0) selon les contraintes techniques.

◆ **Le guide en trois principes**

Un éliminatoire, c'est terminé. Pas de pondération, pas de compensation, pas de « *mais le reste est bon* ». **Trois sérieuses, c'est une alerte.** Deux sérieuses dans la même dimension (Q4 + Q6 + Q7) sont plus préoccupantes que trois dans des dimensions distinctes. **Le contexte module, il ne supprime pas.** Un horizon court allège les exigences — il ne fait pas disparaître les questions éliminatoires.

Tendances et horizon 2027

Cinq signaux à surveiller pour les décideurs qui adoptent des outils open source sur un horizon de trois à cinq ans.

Cyber Resilience Act

Application pleine le 11 décembre 2027. Obligations de sécurité, documentation, reporting et SBOM pour les éditeurs commercialisant un logiciel — y compris certains projets open source. Le statut d'*open source steward* protège les contributions non-commerciales, mais pas les éditeurs qui monétisent. Favorise les projets sous fondation neutre.

Fondations comme arbitres

Linux Foundation, Apache, CNCF, OSGeo voient leurs demandes d'hébergement augmenter depuis 2021. Le modèle *fork sous fondation neutre* (OpenSearch, OpenTofu, Valkey) s'est imposé comme la réponse crédible aux relicences single-vendor. Davantage de transferts attendus d'ici 2027.

IA générative et open source

Tension juridique en cours — peut-on entraîner un modèle sur du code GPL et redistribuer les poids ? L'OSI a publié en 2024 une première définition de l'*Open Source AI*, controversée, en cours de révision. La grille des douze questions s'applique aussi aux composants IA intégrés.

SBOM — chaîne d'approvisionnement

L'affaire xz utils (mars 2024) a accéléré l'adoption des *Software Bill of Materials*. Le CRA les rend de facto obligatoires. Pour les décideurs, le signal est simple : préférer les projets qui publient leur SBOM et documentent leurs dépendances.

EU

Souveraineté numérique européenne

DINUM SUITE et NUMSPOT accélèrent l'adoption d'open source européen dans les administrations. D'ici 2027, les appels d'offres publics inclueront de plus en plus des critères de gouvernance et de localisation. Opportunité pour les projets européens bien gouvernés (QGIS, LibreOffice, Nextcloud, Odoo).

Conclusion

Cinq idées à retenir de ce livre blanc.

"01" L'open source n'est pas gratuit, il est libre

- ✓ La confusion entre les deux a produit des milliers d'abandons silencieux
- ✓ Libre = vous avez des droits (utiliser, modifier, redistribuer, forker, changer de prestataire)
- ✓ Gratuit = vous ne payez pas de licence. Le TCO, lui, n'est jamais nul

"02" La gouvernance prédit tout le reste

- ✓ Un projet sous fondation neutre avec DCO n'a pas relicencié en quarante ans
- ✓ Un projet single-vendor avec CLA a relicencié trois fois en deux ans
- ✓ C'est de la structure, pas du hasard. Devrait être la première question posée

"03" Une licence est OSI ou ne l'est pas

- ✓ Le vocabulaire *open*, *community*, *source-available* est du marketing
- ✓ La liste officielle de `opensource.org/licenses` est le seul juge
- ✓ Trente secondes de vérification évitent un risque juridique structurel

"04" Le triangle Q4 + Q7 + Q9 prédit les relicences

- ✓ Single-vendor + SaaS managé sous pression hyperscaler + précédent de relicence
- ✓ Profil exact d'Elastic 2021, HashiCorp 2023, Redis 2024 — modèle prédictif validé
- ✓ Un projet qui présente ces trois signaux est un candidat probable à la prochaine relicence

"05"

L'exécution décide du succès, pas le code

- ✓ Le meilleur outil open source du monde échoue sans capacité de mise en œuvre identifiée
- ✓ Sans support proportionné à la criticité, sans TCO calculé honnêtement
- ✓ La grille structure l'information — la décision finale reste la vôtre

Citation signée

> *Choisir un outil open source, c'est signer un contrat avec une communauté — pas avec un éditeur. La gouvernance, c'est la clause de résiliation. La licence, c'est le périmètre des droits. Le TCO, c'est la réalité économique derrière la gratuité apparente. Posez ces trois questions avant toute autre, et vous aurez déjà évité l'essentiel des mauvaises surprises.*
>
> — Mattieu Pottier · mai 2026

Références et ressources

Open Source Initiative

→ Open Source Definition : opensource.org ↗

→ Liste officielle des licences OSI : opensource.org ↗

→ Déclaration OSI sur les licences « fauxpen » : opensource.org ↗

Licences — textes officiels

→ SSPL v1 (MongoDB, 2018) : www.mongodb.com ↗

→ BSL v1.1 (HashiCorp, 2023) : www.hashicorp.com ↗

→ Elastic License v2 (2021) : www.elastic.co ↗

Études économiques

→ Hoffmann, Nagle, Zhou — *The Value of Open Source Software* (Harvard Business School, Working Paper 24-038, janvier 2024) : www.hbs.edu ↗

→ Linux Foundation — *State of Global Open Source 2025* (novembre 2025) : www.linuxfoundation.org ↗

Lectures complémentaires

→ Eric S. Raymond — *The Cathedral and the Bazaar* (1999) : catb.org ↗

→ Steven Levy — *Hackers: Heroes of the Computer Revolution* (1984, Anchor Press/Doubleday)

Cyber Resilience Act

→ Texte officiel UE : eur-lex.europa.eu ↗

→ Synthèse ANSSI : www.cyber.gouv.fr ↗

Événements clés — sources primaires

→ Elastic — annonce licence (14 jan. 2021) : www.elastic.co ↗

→ HashiCorp — annonce BSL (10 août 2023) : www.hashicorp.com ↗

→ Redis — annonce relicence (20 mars 2024) : redis.io ↗

→ Valkey — acceptation Linux Foundation (28 mars 2024) : www.linuxfoundation.org ↗

→ OpenTofu — manifeste : opentofu.org ↗

Fondations de référence

→ Linux Foundation : linuxfoundation.org ↗

→ Apache Software Foundation : apache.org ↗

→ CNCF : cncf.io ↗

→ OSGeo : osgeo.org ↗

→ Open Source Hardware Association : oshwa.org ↗

→ Creative Commons : creativecommons.org ↗

Glossaire

Abandonware

Logiciel dont le développement actif a cessé. Toujours fonctionnel à l'instant T, mais sans patches de sécurité, sans compatibilité maintenue, sans évolution fonctionnelle.

AGPL (Affero General Public License)

Licence copyleft fort (OSI). L'obligation de publier le code source s'applique même pour un usage en SaaS (via le réseau).

Apache 2.0

Licence permissive (OSI). Libre usage commercial, redistribution, modification. Inclut une clause de brevet explicite.

BSL (Business Source License)

Licence source-available. Libre usage interne, mais restreint l'usage commercial concurrent. Non approuvée OSI. Utilisée par HashiCorp pour Terraform (2023).

CLA (Contributor License Agreement)

Accord signé par les contributeurs. Selon le type, peut impliquer une cession de copyright à l'éditeur — mécanisme qui rend les relicences unilatérales possibles.

Copyleft

Principe juridique selon lequel les modifications d'un logiciel libre doivent être redistribuées sous la même licence libre. Déclinaisons : fort (GPL, AGPL), faible (MPL, LGPL), réseau (AGPL).

DCO (Developer Certificate of Origin)

Mécanisme de contribution utilisé par la Linux Foundation. Le contributeur certifie son droit de contribuer sans céder son copyright. Rend les relicences unilatérales structurellement impossibles.

Eclipse Public License (EPL)

Licence copyleft faible (OSI) portée par la Eclipse Foundation. Copyleft au niveau module. Utilisée par l'IDE Eclipse, Jakarta EE et de nombreux projets Java d'entreprise.

ELv2 (Elastic License v2)

Licence source-available d'Elastic (2021). Trois interdictions : SaaS concurrent, contournement des features payantes, suppression des marques. Non OSI.

Fondation neutre

Entité à but non lucratif hébergeant un projet open source avec gouvernance multi-entreprises. Exemples : Linux Foundation, Apache, CNCF, Eclipse, OSGeo.

Faux open source (fauxpen)

Terme de l'OSI pour les licences source-available présentées comme open source mais ne respectant pas l'Open Source Definition.

Fork

Copie d'un projet open source développée indépendamment. Légal tant que le code original est sous licence permissive ou copyleft. Exemples : OpenSearch (fork Elasticsearch), Valkey (fork Redis), OpenTofu (fork Terraform).

GPL (General Public License)

Famille de licences copyleft fort (OSI). GPL v2 : Linux. GPL v3 : nombreux projets FSF. Obligation de publier le code dérivé sous GPL.

Licence OSI

Licence approuvée par l'Open Source Initiative selon les dix critères de l'Open Source Definition. La seule définition canonique de ce qu'est une vraie licence open source.

LGPL (Lesser GPL)

Famille de licences copyleft faible (OSI). Permet l'utilisation d'une bibliothèque LGPL dans un produit propriétaire si elle est liée dynamiquement. Utilisée par Qt, glibc, Odoo Community.

LTS (Long Term Support)

Version d'un logiciel garantie supportée (patches de sécurité, corrections de bugs) sur une durée étendue, typiquement 3 à 10 ans. Critère clé pour les outils critiques de longue durée (cf. Q1, Q11).

MIT

Licence permissive (OSI), la plus populaire sur GitHub. Aucune restriction d'usage. Obligation minimale d'attribution.

MPL-2.0 (Mozilla Public License 2.0)

Licence copyleft faible (OSI). Copyleft au niveau fichier. Les fichiers MPL modifiés doivent rester MPL ; le reste du code peut être propriétaire.

Open Source Definition (OSD)

Dix critères définis par l'OSI qui doivent être respectés par toute licence pour être approuvée open source. Disponible sur opensource.org/definition.

RSAL (Redis Source Available License)

Licence source-available de Redis Ltd. (2024). Interdit les services cloud managés concurrents. Non OSI.

SBOM (Software Bill of Materials)

Inventaire structuré des composants logiciels d'un projet. Rendu de facto obligatoire par le Cyber Resilience Act pour les produits dans son champ.

Source-available

Catégorie de licences où le code source est visible mais où des restrictions commerciales empêchent certains usages. Ni propriétaire pur, ni open source. Exemples : SSPL, BSL, ELv2, RSAL.

SSPL (Server Side Public License)

Licence source-available créée par MongoDB (2018). Clause centrale : obligation de publier l'intégralité de l'infrastructure SaaS si le logiciel est utilisé dans un service managé. Non OSI (demande d'approbation retirée par MongoDB en 2019).

TCO (Total Cost of Ownership)

Coût total de possession sur une période donnée. Inclut les licences évitées, le déploiement, la formation, la maintenance, l'hébergement et le coût de sortie.

Vendor lock-in

Dépendance d'une entreprise envers un fournisseur, rendant le changement coûteux voire impraticable. Concerne autant les outils propriétaires (formats, contrats) que les projets open source single-vendor (Q4, Q6) ou les prestataires uniques sur une stack OSS (Q10).

Historique des révisions

VERSION	DATE	AUTEUR	MODIFICATIONS
v1.0.0	Mai 2026	Mattieu Pottier	Publication initiale

À propos de l'auteur

Mattieu Pottier

Consultant indépendant — intégration SIG × ERP, webmapping, stack open source

Il accompagne les PME, ETI et éditeurs logiciels dans leurs choix de stack open source — de l'évaluation technique au déploiement en production. Ce livre blanc s'appuie sur ses choix d'outils depuis 2020 et sur ses missions d'audit et d'intégration menées auprès de clients dans les secteurs BTP, SIG, infrastructure et logistique.