APPRENDRE · THE STACK

# Architecture sans secrets : comment minimiser votre surface d'attaque

*La plupart des startups SaaS gèrent entre vingt et cinquante secrets en production : clés d'API, identifiants de base de données, jetons de déploiement, certificats. Pillar en gère environ six. Cet article explique l'architecture, les compromis et la méthode pour auditer puis réduire votre propre inventaire de secrets.*

---

10 min de lecture

Dernière mise à jour: 10 juin 2026

> *Chaque secret que vous détenez est à la fois une surface d'attaque et un fardeau opérationnel. Moins vous manipulez de secrets, plus votre exploitation est sûre et plus vos opérations sont simples ; Pillar conçoit donc pour le « zéro secret quand c'est possible ».*

## Thèse

- Pourquoi le nombre de secrets dans votre stack est un indicateur direct de risque opérationnel et de coût de maintenance
- Comment The Zero-Secret Stack réorganise les fonctions classiques (paiement, formulaires, déploiement, authentification) pour éliminer les clés côté serveur
- Quelles substitutions concrètes appliquer : Stripe Payment Links au lieu de Stripe API, Formspree au lieu de Mailgun, Cloudflare Access au lieu d'identifiants applicatifs
- Comment isoler les rares secrets restants dans un Cloudflare Worker, et avec quelle cadence les faire tourner
- Une checklist tactique pour auditer votre propre inventaire et le réduire de moitié en un trimestre

# 01 — Le modèle mental : The Zero-Secret Stack

## The Zero-Secret Stack

The Zero-Secret Stack est une discipline architecturale, pas un produit. Le principe est simple : pour chaque fonction de votre stack, demandez-vous s'il existe une variante « hosted » ou « edge » qui déplace le secret hors de votre périmètre. Cinq piliers structurent la démarche.

1

### Paiement délégué (Hosted Checkout)

Au lieu d'intégrer la Stripe API directement avec une clé secrète côté serveur, Pillar utilise les Stripe Payment Links. L'utilisateur est redirigé vers une page de checkout hébergée par Stripe ; aucune clé API ne réside dans le code de Pillar. Le seul secret du flux est le webhook secret de vérification, isolé dans un Cloudflare Worker dédié.

2

### Intake sans backend

Les formulaires de contact, candidatures et requêtes presse passent par Formspree. La distribution courriel, la validation anti-spam et le stockage sont gérés par le fournisseur. Pillar n'a donc ni clé Mailgun, ni clé SendGrid, ni serveur SMTP à entretenir.

3

### Déploiement local-first

Le déploiement Cloudflare Pages s'effectue via la CLI `wrangler` exécutée localement par l'opérateur authentifié. Aucun jeton de déploiement ne réside sur un serveur CI distant, donc aucun secret de pipeline à faire tourner ou à révoquer en urgence.

4

### Pas de base de données, pas de serveur

Le site est statique. Les contenus dynamiques sont calculés à la compilation, et les rares appels temps réel (webhook Stripe) tournent dans des Cloudflare Workers serverless. Sans base de données, il n'y a ni identifiants Postgres, ni chaîne de connexion Redis, ni clé d'API Supabase à protéger.

5

### Isolation et rotation des secrets résiduels

Les rares secrets inévitables (typiquement, le webhook secret de Stripe) vivent exclusivement dans les variables d'environnement du Cloudflare Worker concerné, jamais dans le repo. Rotation trimestrielle planifiée, audit annuel des accès, principe du moindre privilège appliqué sans exception.

## 02 — Les données.

**~6**

Secrets totaux gérés en production par Pillar, tous environnements confondus

PILLAR ARCHITECTURE AUDIT 2026

**20–50**

Plage typique de secrets manipulés par une startup SaaS comparable

ESTIMATION SÉCURITÉ INDUSTRIE SAAS

**0**

Clés d'API Stripe présentes côté Pillar (paiement délégué aux Payment Links)

PILLAR ARCHITECTURE AUDIT 2026

**0**

Identifiants de base de données en production (pas de base de données)

PILLAR ARCHITECTURE AUDIT 2026

**1**

Cloudflare Worker isolé pour la vérification du webhook Stripe

PILLAR ARCHITECTURE AUDIT 2026

**0**

Clés SSH de production (pas de serveur long-running)

PILLAR ARCHITECTURE AUDIT 2026

## Pourquoi compter ses secrets est l'audit de sécurité le plus utile

La première question qu'un ingénieur sécurité devrait poser à toute équipe technique n'est pas « avez-vous un WAF ? » ni « quelle est votre politique MFA ? », mais simplement : « combien de secrets votre équipe détient-elle, et où sont-ils ? ». Le nombre est révélateur. Chaque clé d'API, chaque mot de passe de service, chaque jeton de déploiement, chaque certificat privé est un vecteur de compromission : un endroit où il peut fuiter (repo public, log, capture d'écran, employé sortant), un endroit où il doit être tourné, audité, restreint en portée.

Une startup SaaS classique cumule rapidement vingt à cinquante secrets : clé Stripe secrète, identifiants de base de données de production et de staging, clés Mailgun ou SendGrid, clés Twilio, clés Segment ou Mixpanel, clés DNS chez le registrar, jetons GitHub Actions, clés AWS IAM, certificats TLS, secrets OAuth. Chacun de ces secrets est inoffensif individuellement, mais la masse crée une dette opérationnelle latente : qui sait où chacun vit ? Qui sait quand il a été tourné pour la dernière fois ? Que se passe-t-il si l'ingénieur qui l'a configuré quitte l'entreprise ?

L'approche Pillar consiste à renverser la question. Au lieu de chercher à mieux gérer les cinquante secrets, on conçoit l'architecture pour n'en générer que six. Cela ne résout pas tous les problèmes de sécurité, mais cela réduit drastiquement la surface d'attaque et libère du temps opérationnel pour les enjeux qui restent.

## Substitutions concrètes : ce que vous gardez, ce que vous déléguez

Le paiement est le cas d'usage le plus instructif. L'intégration Stripe « classique » récupère une clé secrète côté serveur et instancie un client Stripe dans le backend. Cette clé a un pouvoir étendu (créer des charges, rembourser, lister des clients) et doit être protégée en conséquence. L'alternative est Stripe Payment Links : vous créez chaque lien depuis le Dashboard Stripe, vous intégrez le lien comme une simple URL HTML dans votre site, et Stripe héberge la page de checkout. Aucune clé API ne quitte le Dashboard. La seule pièce qui revient chez vous est l'événement webhook quand un paiement aboutit ; ce webhook est signé avec un secret partagé, isolé dans un Cloudflare Worker dédié.

Pour les formulaires, l'archétype classique est : une route backend reçoit le POST, valide les champs, stocke en base, envoie un mail via SendGrid ou Mailgun. Trois à quatre secrets potentiels en jeu (clé SMTP, clé d'API courriel, identifiants base de données). La substitution sans secret : Formspree (ou un équivalent comme Basin, Getform) reçoit le POST directement depuis le navigateur, applique sa propre couche anti-spam, et réexpédie par courriel. Le formulaire pointe vers un endpoint public ; il n'y a strictement aucune clé à protéger côté Pillar.

Pour l'authentification administrative, plutôt que de gérer un système d'identifiants applicatifs (et donc une base utilisateurs, un système de hash de mots de passe, des jetons de session, éventuellement un fournisseur OAuth tiers), Pillar s'appuie sur Cloudflare Access, qui délègue l'authentification au fournisseur SSO de l'organisation. La gestion des comptes vit au niveau Cloudflare ; le site lui-même ne stocke ni mot de passe, ni clé de session, ni jeton OAuth.

## Le compromis assumé : ce que vous perdez en design zero-secret

Cette approche a des limites qu'il faut nommer. D'abord, vous cédez du contrôle sur l'expérience utilisateur : la page de checkout est celle de Stripe, pas la vôtre ; le rendu du courriel de notification est celui de Formspree. Si votre produit exige une expérience de paiement entièrement custom, ou un workflow de formulaire avec logique conditionnelle complexe, le compromis ne tient plus. Pour un site éditorial, une boutique d'infoproduits, un portfolio professionnel, l'expérience standard suffit largement ; pour un SaaS produit dont le checkout fait partie de la marque, la décision est plus nuancée.

Ensuite, vous dépendez davantage d'un nombre restreint de fournisseurs. Stripe pour le paiement, Cloudflare pour l'hébergement et l'authentification, Formspree pour les formulaires. Si l'un de ces fournisseurs tombe, votre service tombe avec lui. C'est un arbitrage classique entre risque de fournisseur (centralisé) et risque opérationnel (distribué sur de nombreux secrets que vous gérez vous-même). Pour la grande majorité des petites équipes, le risque opérationnel domine : une clé mal gérée ou un employé négligent cause plus d'incidents que les pannes de Stripe.

Enfin, certaines fonctionnalités sont simplement inaccessibles sans backend. La personnalisation profonde, les recommandations algorithmiques, les tableaux de bord client riches, la connexion temps réel à des services tiers : tout cela demande un serveur d'application. The Zero-Secret Stack ne prétend pas remplacer ces architectures ; il offre un point de départ radicalement simplifié pour les cas d'usage où le serveur n'est pas indispensable, et un modèle mental pour minimiser les secrets même lorsqu'un serveur reste nécessaire.

### Isoler les secrets résiduels : la méthode Cloudflare Worker

Même en poussant l'approche zero-secret au maximum, il restera presque toujours un ou deux secrets inévitables. Dans le cas Pillar, le webhook secret de Stripe est l'exemple type : il faut bien que quelqu'un vérifie la signature des notifications de paiement entrantes pour confirmer qu'elles viennent réellement de Stripe et pas d'un attaquant. La discipline consiste alors à donner à ce secret le périmètre le plus restreint possible.

Concrètement, Pillar isole ce secret dans un Cloudflare Worker dédié à cette seule fonction. Le Worker est déployé via `wrangler deploy` et le secret est injecté via `wrangler secret put STRIPE_WEBHOOK_SECRET` ; il ne quitte jamais l'infrastructure Cloudflare et n'apparaît dans aucun fichier versé au repo. Le Worker n'a aucune autre responsabilité que de vérifier la signature et d'enregistrer l'événement ; il n'a pas accès au reste du système. Si le secret fuit, l'attaquant ne peut que falsifier des webhooks Stripe sur ce point d'entrée : il ne peut pas pivoter vers une base de données, vers une clé de paiement, vers le DNS.

La rotation est planifiée trimestriellement : on régénère le webhook secret depuis le Dashboard Stripe, on met à jour la variable d'environnement du Worker, et on documente la date dans un journal d'opérations. Cette discipline ne supprime pas le risque, mais elle le borne et le rend audible.

## 03 — Regardez : une démonstration réelle

## 04 — Checklist tactique : auditer puis réduire votre inventaire de secrets

Cette séquence prend un trimestre pour une petite équipe et peut typiquement éliminer 60 à 80 % des secrets actifs sans perte fonctionnelle. Documentez chaque étape : la valeur d'un audit zero-secret tient autant à la trace écrite qu'au résultat technique.

1. Établir l'inventaire complet : listez chaque clé d'API, mot de passe, jeton, certificat utilisé en production et en préproduction. Précisez pour chacun : où il vit, qui y a accès, date de dernière rotation, portée des privilèges.
2. Catégoriser chaque secret : est-il réellement nécessaire au fonctionnement actuel, ou est-il l'héritage d'une fonctionnalité abandonnée ? Éliminez immédiatement tout secret sans usage actif.
3. Remplacer la clé Stripe côté serveur par Stripe Payment Links pour les cas d'usage de checkout standard ; conserver l'API uniquement si une expérience custom est strictement requise.
4. Remplacer les clés SMTP, Mailgun ou SendGrid par un service de formulaires hébergé (Formspree, Basin) pour les flux de contact et de candidature.
5. Remplacer la base de données par du JSON statique compilé au build et des Cloudflare Workers serverless pour les rares accès temps réel ; cela élimine d'un coup tous les identifiants Postgres, MySQL ou Mongo.
6. Remplacer l'authentification applicative par Cloudflare Access ou un équivalent SSO-géré ; vous déléguez la gestion des comptes à un produit fait pour ça.
7. Pour les secrets restants, inévitables, les isoler dans un Cloudflare Worker dédié via `wrangler secret put`, planifier une rotation trimestrielle, et documenter la date de chaque rotation dans un journal d'opérations consultable.

# Questions fréquentes.

### Combien de secrets devrais-je viser pour mon propre site ou produit ?

Il n'y a pas de cible universelle, mais une heuristique utile : pour un site éditorial ou un portfolio professionnel, moins de dix secrets en production est réaliste ; pour un SaaS léger, moins de vingt est ambitieux mais atteignable ; au-delà de cinquante, vous avez probablement une dette opérationnelle qui mérite un audit. Le bon indicateur n'est pas le nombre absolu, mais votre capacité à répondre à trois questions pour chaque secret : où vit-il, quand a-t-il été tourné la dernière fois, que se passe-t-il s'il fuit ?

### Stripe Payment Links suffit-il pour un vrai business ?

Pour la majorité des usages (vente de produits numériques, abonnements simples, sessions de coaching, services professionnels), oui. Vous perdez le contrôle total sur l'apparence du checkout et certains workflows avancés (tarification dynamique calculée à la volée, par exemple) demandent l'API. Mais pour un site éditorial, une boutique d'infoproduits ou un portfolio d'autorité comme Pillar, Payment Links couvre l'essentiel sans introduire de clé secrète côté serveur.

---

### Et si Cloudflare ou Stripe tombe ? Je n'ai plus rien.

C'est un risque réel, à arbitrer en connaissance de cause. The Zero-Secret Stack échange un risque opérationnel distribué (cinquante secrets à gérer) contre un risque de fournisseur concentré (trois ou quatre acteurs). Empiriquement, pour une petite équipe, les incidents causés par une mauvaise gestion de secret sont bien plus fréquents que les pannes globales de Stripe ou Cloudflare. Pour une organisation régulée ou critique, l'analyse de risque doit être formelle et incluse dans la documentation de continuité.

---

### Comment isoler concrètement un secret dans un Cloudflare Worker ?

Créez un Worker dédié à la fonction qui consomme le secret (par exemple, la vérification du webhook Stripe). Injecté-le via `wrangler secret put NOM_DU_SECRET` au lieu de le versér dans un fichier de configuration. Limitez le code du Worker à la stricte fonction nécessaire : pas d'accès à d'autres services, pas de log du secret en clair, pas de chemin de fallback qui le révélerait. Planifiez la rotation trimestrielle et tenez un journal des dates.

---

### Comment passer de cinquante secrets à dix en pratique ?

Trimestre par trimestre, fonction par fonction. Commencez par la fonction la plus simple à déléguer (typiquement les formulaires : une migration vers Formspree prend une après-midi). Passez ensuite au paiement (Stripe Payment Links), puis à l'authentification (Cloudflare Access ou équivalent SSO), puis à la base de données si votre cas d'usage le permet. Pour chaque migration, documentez le secret éliminé et vérifiez qu'il n'a pas survécu dans un script oublié, une variable d'environnement de staging ou un ancien runbook.

---