

Architecture du pipeline /veille

Flux RSS dynamiques — rotation horaire, stockage, consommation côté client

Dernière mise à jour : 18 septembre 2026 — version du site : voir CHANGELOG.md

1. Vue d'ensemble

La page **/veille** affiche une sélection de 6 articles issus d'un pool de 25 flux RSS (réseau, administration système, cybersécurité, tech généraliste). Le site reste **statique** au sens où il est servi (rendu HTML/CSS/JS classique, hébergé sur Netlify), mais cette page précise a une partie dynamique : le contenu affiché tourne automatiquement **toutes les heures**, sans qu'aucun serveur permanent ne soit nécessaire — la dynamique vient de 3 fonctions serverless (Netlify Functions) + d'un espace de stockage clé-valeur (Netlify Blobs).

Principe d'**amélioration progressive** : si les fonctions ne répondent pas (pas encore déployées, en développement local, panne ponctuelle), la page affiche toujours quelque chose — une liste statique de secours rendue côté serveur au moment du build (les 6 premières sources du pool). Jamais de page cassée ou vide.

Schéma du flux de données

1. Un cron externe (cron-job.org, gratuit) appelle une URL toutes les heures
2. `veille-rotate.mjs` interroge les 25 flux RSS du pool en parallèle
3. Les 6 articles les plus récents (toutes sources confondues) sont sélectionnés
4. Ils sont écrits dans Netlify Blobs (clé `current`) et archivés (clé `history`)
5. Au chargement de `/veille`, le navigateur du visiteur appelle `veille-current.mjs` et remplace le contenu de secours par les vrais articles

Important : le patch client est **positionnel** (ligne $i \leftarrow$ article i), pas par identité de source — la source affichée à une position donnée change d'un cycle à l'autre, elle est impossible à connaître au moment du rendu serveur. Aucun polling non plus : la liste est récupérée une seule fois au chargement de la page — un article ne change jamais sous les yeux d'un visiteur déjà sur la page, seulement au prochain rechargement.

2. Où se trouve quoi

Fichier	Rôle
<code>src/data/veille-feeds.mjs</code>	Source de vérité UNIQUE du pool de 25 flux RSS — voir section 3.
<code>netlify/functions/veille-rotate.mjs</code>	Fonction PROTÉGÉE : va chercher les articles, sélectionne les 6 plus récents, écrit dans Blobs. Déclenchée par le cron externe.
<code>netlify/functions/veille-current.mjs</code>	Fonction PUBLIQUE (lecture seule) : renvoie les 6 articles actuellement retenus. Appelée par <code>VeillePage.astro</code> .
<code>netlify/functions/veille-history.mjs</code>	Fonction PUBLIQUE (lecture seule) : renvoie l'historique (7 jours glissants). Appelée par <code>VeilleHistoriquePage.astro</code> .
<code>netlify/functions/_shared/rss.mjs</code>	Parsing RSS 2.0 minimal (fast-xml-parser) + décodage d'entités HTML. Renvoie le dernier item d'un flux, ou null si injoignable.

Fichier	Rôle
netlify/functions/_shared/theme.mjs	Classification par mots-clés (PAS une IA) : attribue un thème (Vulnérabilité, Piratage, Réseau...) à chaque article.
netlify/functions/_shared/store.mjs	Accès à Netlify Blobs (store "veille", clés current/history) + purge automatique de l'historique à 7 jours.
src/components/pages/VeillePage.astro	Page /veille : rendu serveur de la liste de secours + script client qui appelle veille-current.mjs.
src/components/pages/VeilleHistoriquePage.astro	Page /veille-historique : appelle veille-history.mjs, affiche les 7 derniers jours d'archives.

3. Le pool de sources RSS

Les 25 sources vivent dans `src/data/veille-feeds.mjs`, sous la forme d'un tableau d'objets. C'est la **seule** source de vérité : elle est importée à la fois par la fonction de rotation (qui va réellement chercher les articles) et par la page /veille (pour le rendu de secours). **Ajouter une source = ajouter un objet à ce tableau**, rien d'autre à modifier.

```
{
  id: 'identifiant-unique',
  category: 'Catégorie affichée (thème de repli)',
  name: 'Nom de la source',
  description: 'Description courte affichée en secours.',
  url: 'https://site-source.example/',
  feedUrl: 'https://site-source.example/feed/',
}
```

Avant d'ajouter une source : vérifier que son `feedUrl` renvoie bien du RSS 2.0 valide (pas Atom, pas RSS 1.0 — non supportés). Un flux cassé n'empêche jamais les autres de fonctionner : `fetchLatestItem()` renvoie simplement `null` pour ce flux-là, il est absent du calcul ce cycle-ci.

Les 25 sources actuelles

#	Source	Catégorie
1	ANSSI — CERT-FR	Cybersécurité (institutionnel)
2	ZATAZ	Cybersécurité
3	IT-Connect	Administration système & réseau
4	Cisco Blogs	Constructeur réseau
5	Korben	Tech généraliste
6	01net	Tech généraliste
7	Silicon.fr	IT professionnel
8	Numerama	Tech généraliste
9	UnderNews	Cybersécurité
10	Journal du Hacker	Tech généraliste
11	Developpez.com	Développement
12	DataSecurityBreach	Cybersécurité
13	Krebs on Security	Cybersécurité
14	The Hacker News	Cybersécurité
15	Ars Technica	Tech généraliste
16	Network World	Administration système & réseau
17	WeLiveSecurity	Cybersécurité
18	Dark Reading	Cybersécurité
19	Docker Blog	Cloud & Infrastructure
20	SANS Internet Storm Center	Cybersécurité (institutionnel)
21	Ubuntu Security Notices	Administration système & réseau
22	Graham Cluley	Cybersécurité
23	ZDNet France	IT professionnel
24	Presse-citron	Tech généraliste

#	Source	Catégorie
25	Zabbix Blog	Administration système & réseau

Note : LeMagIT faisait partie du pool d'origine mais a été retirée — son URL de flux connue est cassée (4 variantes testées en conditions réelles, aucune ne renvoie de XML valide). À réintégrer si la bonne URL est retrouvée.

4. Les 3 fonctions Netlify

4.1 — veille-rotate.mjs (protégée, déclenchée par le cron)

C'est la seule des 3 fonctions à ÉCRIRE des données. Ce n'est **pas** une Scheduled Function Netlify (palier payant incertain selon les plans) — elle est déclenchée par une simple requête HTTP **GET** envoyée par un service de cron externe et gratuit (cron-job.org), toutes les heures.

- Interroge en parallèle (`Promise.all`) le dernier article de chacune des 25 sources.
- Attribue un thème à chaque article récupéré (`classifyTheme()`, voir 4.4).
- Trie l'ensemble par date de publication décroissante et ne garde que les **6 plus récents**, toutes sources confondues.
- Écrit ce top-6 dans Blobs (clé `current`) et l'ajoute à l'historique (clé `history`).
- Si TOUT le pool échoue en même temps (panne réseau ponctuelle), l'ancienne sélection est conservée plutôt que remplacée par une liste vide.

Protection : la requête doit porter un en-tête HTTP :

```
x-veille-secret:
```

Si la variable d'environnement `VEILLE_ROTATE_SECRET` n'est pas définie sur Netlify, la fonction accepte toute requête (pratique en local, **à ne jamais laisser ainsi en production**). Voir section 5.

Réponse type (aussi utile pour vérifier manuellement que tout fonctionne) :

```
{"ok":true,"rotatedAt":"2026-09-18T08:58:48.110Z","poolSize":25,"fetched":25,"shown":6}
```

4.2 — veille-current.mjs (publique, lecture seule)

Renvoie simplement le contenu actuel de la clé Blobs `current` (les 6 articles du dernier cycle de rotation).

Appelée une seule fois par le script client de `VeillePage.astro` au chargement de la page — pas de polling.

Aucun secret requis : lecture publique.

4.3 — veille-history.mjs (publique, lecture seule)

Renvoie le contenu de la clé Blobs `history` (articles archivés au fil des cycles de rotation, purgés

automatiquement au-delà de 7 jours). Appelée par `VeilleHistoriquePage.astro` (page /veille-historique).

4.4 — Modules partagés (`_shared/`)

Module	Rôle
<code>rss.mjs</code>	<code>fetchLatestItem(feedUrl)</code> : télécharge le flux (timeout 10s), parse le XML avec <code>fast-xml-parser</code> , renvoie <code>{title, description, link, pubDate}</code> du 1er, ou null si injoignable/mal formé. Décode aussi les entités HTML (' etc., fréquentes sur les flux WordPress) et retire les balises HTML des descriptions.
<code>theme.mjs</code>	<code>classifyTheme(titre, description, repli)</code> : recherche par mots-clés (PAS une IA/LLM — décision explicite de rester sur un algorithme simple) parmi 5 thèmes (Vulnérabilité, Piratage, Réglementation, Réseau, Cloud & Infra). Le 1er thème dont un mot-clé correspond l'emporte ; sinon la catégorie statique de la source sert de repli.
<code>store.mjs</code>	Accès à Netlify Blobs (<code>getStore('veille')</code>). <code>readCurrent/writeCurrent</code> , <code>readHistory/writeHistory</code> . <code>pruneHistory()</code> filtre les entrées de plus de 7 jours à CHAQUE lecture/écriture (pas de tâche de fond programmée — le site n'a pas de serveur permanent).

5. Variable d'environnement et cron externe

Le pipeline ne tourne pas tout seul : il faut, une fois après le déploiement, configurer un secret puis un déclencheur externe qui appelle la fonction de rotation toutes les heures.

5.1 — Créer le secret

- Tableau de bord Netlify du site → **Site configuration** → **Environment variables** → **Add a variable**.
- Nom : `VEILLE_ROTATE_SECRET`.
- Valeur : une chaîne longue et aléatoire (générateur de mot de passe quelconque). Elle ne sert qu'à vérifier que l'appelant est bien le cron autorisé.
- Sauvegarder, puis redéployer le site une fois (Deploys → Trigger deploy) pour que la variable soit prise en compte par les fonctions.

5.2 — Configurer le cron (cron-job.org, gratuit)

- Créer un compte gratuit sur cron-job.org (ou un service équivalent).
- Créer un nouveau cronjob avec l'URL :

```
https://.netlify.app/.netlify/functions/veille-rotate
```

- Méthode : GET.
- En-tête personnalisé (onglet "Advanced" en éditant le cronjob) : nom `x-veille-secret`, valeur = exactement celle de `VEILLE_ROTATE_SECRET`.
- Fréquence : toutes les heures (expression cron `0 * * * *`).

5.3 — Vérifier que ça marche

Ouvrir directement l'URL de la fonction dans un navigateur (avec l'en-tête, via un outil comme curl, ou simplement recharger /veille après un cycle). Une réponse `{"ok":true,...}` confirme que la rotation a fonctionné ; la page /veille doit alors afficher de vrais articles cohérents entre eux (thème / titre / description / lien).

6. Comportement de secours

Si les fonctions ne répondent pas — pas encore déployées, développement local sans `netlify dev`, panne ponctuelle — le script client de `VeillePage.astro` échoue silencieusement (try/catch) et la page garde son contenu rendu côté serveur : les 6 premières sources du pool, avec leur description statique. **Jamais de page cassée, de ligne vide ou d'erreur visible pour le visiteur.**

En développement local avec `npm run dev` (`astro dev` seul), aucune fonction Netlify ne tourne — c'est ce secours statique qui s'affiche systématiquement. Pour tester les fonctions en local, il faut `npm run netlify:dev` (émule les fonctions + Blobs).

Migration vers un serveur Ubuntu auto-hébergé

Ce que change le passage de Netlify à un serveur perso, une fois un nom de domaine et un serveur disponibles — document de préparation, rien n'est encore fait.

7. Ce qui change, ce qui ne change pas

Le site reste construit exactement de la même façon (`npm run build`, sortie dans `dist/`) — seul l'**endroit qui héberge et sert cette sortie** change, ainsi que la façon dont le pipeline /veille tourne en arrière-plan. Le reste du site (les 5 pages, toute la DA, les PDF, les polices) n'a besoin d'aucune modification.

	Aujourd'hui (Netlify)	Demain (Ubuntu perso)
Serveur web	Managé par Netlify (CDN)	Apache (voir section 8)
Build & déploiement	Automatique à chaque git push	Manuel via SSH, ou scripté plus tard (voir section 10)
Rotation /veille	Netlify Function déclenchée par un cron externe (cron-job.org)	Script Node + vrai cron Linux, en local sur le serveur
Stockage current/history	Netlify Blobs (clé-valeur managé)	2 fichiers JSON sur le disque du serveur
Lecture côté client	Appel à une fonction serverless	Fichier JSON servi tel quel par Apache
Authentification du cron	En-tête secret (VEILLE_ROTATE_SECRET)	Inutile — le cron tourne en local, jamais exposé au réseau
HTTPS	Automatique (Netlify)	À configurer (Let's Encrypt / certbot, voir section 11)

8. Serveur web : rester sur Apache

Recommandation : garder **Apache** plutôt que passer à nginx (souvent proposé par défaut ailleurs). L'app utilise déjà un fichier `.htaccess` testé et fonctionnel en local via XAMPP — inutile de tout réapprendre dans la syntaxe nginx pour un gain qui ne concerne pas les besoins de ce site (pas de trafic massif, pas de reverse-proxy complexe).

Différence importante avec le `.htaccess` local : celui-ci réécrit les requêtes vers `dist/` parce que XAMPP sert depuis `htdocs/Portfolio` (un sous-dossier). Sur le vrai serveur, le `DocumentRoot` du `VirtualHost` pointera **directement** sur `dist/` — ce contournement ne sera plus nécessaire.

```
<VirtualHost *:80>
  ServerName kyliandu.ro
  DocumentRoot /var/www/portfolio/dist
  <Directory /var/www/portfolio/dist>
    AllowOverride All
    Require all granted
  </Directory>
</VirtualHost>
```

(Exemple illustratif — nom de domaine et chemin à adapter le moment venu.)

9. Le pipeline /veille sans Netlify

C'est la partie qui change le plus. Les Netlify Functions n'existeront plus — leur rôle se répartit sur 2 mécanismes Linux standards, plus simples que l'architecture actuelle.

9.1 — veille-rotate.mjs devient un script + un vrai cron

Le code de rotation (interroger les 25 flux, garder les 6 plus récents) reste **quasiment identique** — seule la partie stockage change : au lieu d'appeler `getStore('veille')` (Netlify Blobs), le script écrit directement 2 fichiers JSON sur le disque, par exemple :

```
/var/www/veille-data/current.json
/var/www/veille-data/history.json
```

Important : ce dossier doit être **en dehors** de `dist/` — un nouveau `npm run build` régénère tout le contenu de `dist/` et effacerait les données de rotation en cours si elles y vivaient.

Déclenchement : un vrai cron Linux (crontab), plus besoin de service externe :

```
# crontab -e
0 * * * * /usr/bin/node /var/www/portfolio/scripts/veille-rotate.mjs
```

Comme ce script tourne **en local** sur le serveur (jamais appelé depuis Internet), la protection par en-tête secret (`x-veille-secret` / `VEILLE_ROTATE_SECRET`) n'a plus de raison d'être — elle peut être retirée du code à ce moment-là. Le service cron-job.org devient également inutile.

9.2 — veille-current.mjs / veille-history.mjs disparaissent

Ces 2 fonctions ne faisaient que LIRE et renvoyer du JSON — sur Ubuntu, Apache peut servir les fichiers `current.json` / `history.json` directement, comme n'importe quel fichier statique (au même titre qu'un PDF ou une image). Aucun code serveur supplémentaire n'est nécessaire pour la lecture.

```
# Extrait de VirtualHost – expose /var/www/veille-data/ sous /data/
Alias /data/ /var/www/veille-data/
<Directory /var/www/veille-data>
    Require all granted
</Directory>
```

Côté client, un seul changement dans `VeillePage.astro` et `VeilleHistoriquePage.astro` : l'URL appelée par `fetch()` passe de `/.netlify/functions/veille-current` à `/data/current.json` (même logique de patch positionnel côté client, aucun autre changement de comportement).

10. Déploiement

Pour démarrer (recommandé) : déploiement manuel par SSH après chaque changement poussé sur GitHub — proportionné à la taille du projet, pas besoin de plus au début.

```
ssh utilisateur@serveur
cd /var/www/portfolio
git pull
npm install
npm run build
```

Plus tard, si souhaité : automatiser avec un hook Git côté serveur (post-receive) ou une GitHub Action qui se connecte en SSH et relance le build automatiquement à chaque push — une amélioration possible une fois le déploiement manuel maîtrisé, pas un prérequis.

11. Nom de domaine et HTTPS

- **DNS** : un enregistrement A chez le registrar du domaine, pointant vers l'adresse IP publique du serveur.

- **HTTPS** : certificat gratuit et auto-renouvelé via Let's Encrypt (outil `certbot`, module officiel pour Apache) — Netlify le faisait automatiquement, ce sera désormais une étape manuelle mais simple.
- **Pare-feu** : n'ouvrir que les ports nécessaires (80/443 pour le web, 22 pour SSH — idéalement restreint ou par clé plutôt que mot de passe).

12. Check-list de migration (résumé pratique)

- 1 Installer Apache + Node.js sur le serveur Ubuntu.
- 2 Cloner le repo, `npm install`, `npm run build`.
- 3 Configurer un VirtualHost pointant sur `dist/` (section 8).
- 4 Adapter `veille-rotate.mjs` pour écrire du JSON sur disque au lieu de Netlify Blobs (section 9.1).
- 5 Créer `/var/www/veille-data/`, l'exposer via Apache sous `/data/` (section 9.2).
- 6 Mettre à jour les URLs `fetch()` dans `VeillePage.astro` / `VeilleHistoriquePage.astro`.
- 7 Ajouter la tâche cron horaire (section 9.1), retirer la protection par secret (plus utile).
- 8 Pointer le domaine (DNS) vers le serveur, configurer certbot (section 11).
- 9 Vérifier tout (checklist de test, voir `ADMIN.md`) avant de considérer la migration terminée.