Veille Technologique et Juridique : RUST

Rust est un langage de programmation moderne, créé par Mozilla Research et officiellement lancé en 2015. Son objectif est ambitieux : combiner la performance du C/C++ avec une sécurité mémoire exceptionnelle, le tout sans garbage collector.

C'est aujourd'hui l'un des langages les plus appréciés par les développeurs pour sa fiabilité, sa vitesse et sa philosophie de programmation.

Introduction : Pourquoi parler de Rust ?

Dans le cadre de ma deuxième année de BTS SIO (Option SLAM), Une veille technologique est attendu sur un sujet choisi par ma personne et en rapport avec mon option. Je vais donc présenter mon sujet de veille qui est le langage de développement "RUST"
Afin de pouvoir prendre en compte ses avantages et pourquoi ce langage serait à utiliser en priorité dans le futur.

Rust est un langage de développement apparu officiellement récemment en 2015 avec sa version 1.0
Il a été créé par Graydon Hoare (qui était chez Mozilla) en 2006.
Beaucoup le perçoivent comme étant un langage prometteur car il est similaire au C++ et ne possède pas de ''Garbage collector".
Il est également multi-plateforme puisque depuis 2022 est devenu compatible avec un noyau Linux

Sommaire

Cette veille fait le point sur :

- L’évolution du langage,
- Sa place dans l’écosystème technologique actuel,
- Les usages émergents,
- Les initiatives de la Rust Foundation,
- Les limites qui persistent,
- Juridique,
- Les perspectives qui se dessinent pour les prochaines années.

Qu’est ce que le garbage collector ?

La solution Garbage Collection (ou récupération de mémoire) est une fonctionnalité qui gère automatiquement l'allocation et la libération de mémoire

Rust garantit une gestion sûre de la mémoire, empêchant les bugs les plus courants et dangereux comme les débordements de tampons ou les dangling pointers

- L’évolution : De la possibilité, à l'usage

Rust gagne clairement du terrain dans les entreprises. Mais surtout, il n'est plus seulement populaire chez les développeurs : **il entre dans les plans stratégiques des DSI**.

Plusieurs facteurs expliquent cette montée en puissance :

1. La sécurité mémoire

C’est l’argument qui revient systématiquement.
Rust offre une garantie structurelle contre une large famille de vulnérabilités ainsi que des dépassements de buffer, use-after-free, double free (Une erreur de double libération se

produit si `free()` est appelée plusieurs fois avec la même adresse mémoire), sans recourir à un garbage collector.

Dans une période où les cyberattaques explosent et où la majorité des failles logicielles viennent de problèmes mémoire, cette promesse a de la valeur.

2. Des performances très proches du C/C++

Pour des applications système, du traitement de données en temps réel ou des services cloud à haute fréquence, Rust permet d'obtenir le meilleur des deux mondes :
performance + sécurité.

3. Une image de langage moderne et durable

Les organisations cherchent à attirer des développeurs. Rust est perçu comme un langage "de demain", ce qui aide le recrutement.

4. Un écosystème qui se stabilise

Il y a encore quelques années, l'écosystème Rust était jeune et parfois instable. Aujourd'hui, beaucoup de crates importantes sont maîtrisées, maintenues et suivies par des équipes dédiées.

En résumé : Rust commence à être intégré dans des stratégies long terme, et plus seulement dans des POC "pour voir".

- Les grandes évolutions du langage en 2025

Rust évolue, mais il évolue prudemment. Le langage évite de casser la compatibilité et avance par petits pas. Cette année, plusieurs évolutions marquantes se détachent.

Async, un chantier central

Le modèle asynchrone de Rust a beaucoup fait parler de lui : puissant mais complexe à appréhender. En 2025, plusieurs efforts sont en cours pour simplifier ce domaine :

- stabilisation progressive des closures async,
- travail sur les générateurs,
- amélioration de l'ergonomie autour de `Pin`,
- consolidation des outils de debug async.

Ces changements ne révolutionneront pas Rust d'un coup, mais ils réduisent la marche d'entrée.

Vers une spécification formelle

La communauté investit dans une **formalisation du langage**, notamment via le projet Ferrocene. (Evolution : Rust conforme aux exigences de l'industrie, notamment dans les domaines à haute fiabilité)
Ce n'est pas anecdotique : cela ouvre la porte à Rust dans les secteurs régulés (aéronautique, automobile, ferroviaire, médical).

C'est un signe clair de maturité : Rust veut être un langage certifiable.

Améliorations de la chaîne d'outils

Les gains récents sur le compilateur, le linker ou encore le caching permettent des temps de compilation plus raisonnables. Rien de miraculeux, mais une progression constante, indispensable pour une adoption large en entreprise.

- La sécurité : le véritable ADN de Rust

La sécurité est plus qu'un argument marketing pour Rust : c'est son positionnement stratégique.

1. Le renforcement de la supply chain Rust

La Rust Foundation a pris au sérieux la problématique des paquets compromis et de l'ingénierie sociale autour des mainteneurs.
On voit arriver :

- la signature des crates, : Un paquet se compose d'une ou plusieurs crates qui fournissent un ensemble de fonctionnalités
- un processus de publication sécurisé,
- des audits ciblés sur les crates critiques,
- une meilleure visibilité sur la chaîne de dépendances.

Ce sont des avancées importantes, car de nombreuses attaques récentes passent par des paquets compromis.

2. Les outils de vérification

Plusieurs travaux académiques et industriels émergent.

Le plus intéressant est **LiteRSan**, un analyseur mémoire spécialisé Rust qui limite le surcoût d'instrumentation.

Il s'appuie sur les propriétés structurelles de Rust (notamment les lifetimes) pour détecter des erreurs dans du code `unsafe`.

Ce type d'outil est essentiel pour faire entrer Rust dans des systèmes à forte exigence de fiabilité.

3. Rust et IA pour la sécurité

Des projets explorent comment les modèles de langage peuvent contribuer à la sécurité du code Rust.
L'un d'eux consiste à utiliser les "hallucinations" des modèles comme mécanisme de détection de vulnérabilités.
Ces travaux sont encore expérimentaux mais montrent une évolution intéressante des pratiques d'audit.

- Rust dans les usages industriels

Rust n'est plus réservé aux projets expérimentaux. On le voit dans :

L'embarqué et l'IoT

Grâce à sa légèreté, son absence de runtime et sa sécurité mémoire, Rust s'installe dans des microcontrôleurs, des firmwares, des systèmes IoT et des environnements à faibles ressources.

Les infrastructures cloud et edge

Les services cloud à haute performance ou à très faible latence s'appuient de plus en plus sur Rust.
Les opérateurs cloud ou CDN aiment Rust pour :

- sa robustesse,
- son coût opérationnel réduit,
- sa compatibilité avec WebAssembly.

Les navigateurs et moteurs de rendu

Le projet Servo (alimentation des versions de FIREFOX), bien que moins actif qu'à ses débuts, reste une référence technique.
Beaucoup de composants de sécurité du navigateur Firefox continuent également à migrer vers Rust.

Le développement desktop

Tauri (Tauri est un framework logiciel open source conçu pour créer des applications de bureau et mobiles multiplateformes sur Linux, macOS, Windows, Android et iOS à l'aide d'une interface Web.), un framework Rust + Web pour applications desktop légères, s'impose progressivement comme une alternative moins lourde à Electron.

- Rust et l'IA : un rapprochement progressif

Rust n'a pas vocation à devenir le langage principal du machine learning — la communauté scientifique est installée sur Python.
Mais Rust intervient de plus en plus **en production**, là où les modèles doivent être rapides, fiables et économes.

1. L'inférence haute performance

Des frameworks Rust-native comme Candle ou Burn prennent de l'ampleur.
Ils répondent à un besoin croissant : exécuter des modèles modernes sur des machines limitées (edge computing).

2. Extensions Python en Rust

Les équipes ML commencent à faire ce que font déjà de nombreux data-engineers :
prototyper en Python, puis remplacer les zones critiques par des modules Rust compilés.

Cela donne un mélange :
flexibilité du Python + performance et sécurité du Rust.

3. IA embarquée

Dans des contextes IoT, Rust est un candidat naturel pour exécuter de petits modèles d'inférence localement.

- Rust dans les entreprises : retours du terrain

La réalité opérationnelle est intéressante.

Les points plébiscités

- Les crashs de production deviennent rarissimes.
- Les performances sont très stables.
- Les revues de code Rust sont plus simples grâce au modèle d'emprunts et de propriété.
- Les équipes de sécurité apprécient le langage, qui réduit la surface d'attaque.

Les points qui freinent

- Le temps d'apprentissage reste élevé, surtout pour des équipes issues du Python ou du JavaScript.
- Le modèle async fait encore peur à beaucoup de développeurs.
- Les temps de compilation peuvent peser sur la productivité.
- Le recrutement d'ingénieurs Rust reste difficile.

Les types de projets Rust

Rust s'impose surtout dans :

- les systèmes embarqués,
- les moteurs d'exécution (runtimes, proxies…),
- les couches réseaux,
- les systèmes de fichiers et outils CLI,
- les plateformes de sécurité.

En revanche, Rust est encore très peu utilisé pour du front-end applicatif ou des solutions d'entreprise "classiques".

- Limites et défis de Rust

Rust est un langage solide, mais pas parfait. Plusieurs défis restent présents.

1. L'async, toujours un peu ardu

Même en 2025, on sent que Rust est puissant mais exigeant sur l'asynchrone.
Les mécanismes comme `Pin ( pinning / épingler )` restent difficiles à appréhender.
Les frameworks comme Tokio réduisent la complexité, mais l'écosystème manque encore de standardisation.

2. Le code unsafe (permet de spécifier les opérations potentiellement dangereuses, notamment celles qui peuvent endommager la mémoire, les threads, les types ou toute autre opération dangereuse)

Même si Rust force à le déclarer, l'unsafe reste un point sensible.
On voit parfois du code Rust qui finit par s'appuyer sur autant d'unsafe que du C++, ce qui annule une partie des bénéfices.

3. L'écosystème encore jeune

Même si les crates principales sont solides, certains domaines sont encore trop jeunes pour un usage à grande échelle.

4. La migration depuis C++

La cohabitation avec C++ progresse, mais reste complexe dans les grands systèmes hérités.

5. Les générations de code par IA

Les outils IA produisent parfois du code Rust qui ne compile pas ou brise le modèle d'emprunts.
Les benchmarks récents montrent que le langage reste difficile pour les LLMs.

Rust et les obligations du RGPD

1. Sécurité des données et conformité RGPD

Le RGPD impose aux organisations de garantir la protection des données personnelles en réduisant au maximum les risques techniques. Cela inclut la prévention des failles de sécurité, la maîtrise des accès et l'application du principe de *privacy by design* (sécurité intégrée dès la conception). Ces obligations concernent aussi les technologies utilisées pour développer les applications, car un langage mal sécurisé peut entraîner des vulnérabilités exploitables.

Rust apporte une réponse technique à ces enjeux grâce à son modèle de gestion mémoire basé sur :

- **Ownership** (propriété d'une donnée : un seul élément du programme contrôle une donnée à la fois)
- **Borrowing** (emprunt contrôlé : un élément peut utiliser une donnée sans en devenir propriétaire)
- **Lifetimes** (durée de vie : le compilateur vérifie que les données ne sont pas utilisées après leur suppression)

Ces mécanismes empêchent des failles critiques comme les **buffer overflows** (dépassement de mémoire), les **use-after-free** (utilisation d'une donnée déjà supprimée) ou les **dangling pointers** (pointeurs vers une zone mémoire invalide). Ces failles sont souvent responsables de fuites de données, ce qui constitue une violation du RGPD.

En réduisant fortement ces risques, Rust contribue à renforcer la sécurité des systèmes manipulant des données personnelles, ce qui facilite la conformité aux exigences du RGPD.

2. Rust comme outil facilitateur du “Privacy by Design”

Le RGPD impose que la sécurité soit intégrée dès la conception des systèmes (*privacy by design*). Rust s’inscrit naturellement dans cette logique grâce à sa philosophie orientée sécurité. Son fonctionnement strict empêche les comportements indéfinis (actions imprévisibles du programme), améliore la fiabilité du code et réduit les surfaces d’attaque (zones où un pirate peut exploiter une faille).

Rust permet donc de développer des applications plus robustes, cohérentes avec les principes du RGPD :

- **Sécurité par défaut** (*privacy by default*) : les erreurs dangereuses sont bloquées avant l’exécution
- **Réduction des risques techniques** : moins de failles mémoire exploitables
- **Stabilité accrue** : moins de comportements imprévus pouvant exposer des données

Rust ne rend pas une application automatiquement conforme au RGPD : l’organisation doit toujours gérer les droits d’accès, documenter les traitements et assurer la transparence. Cependant, en limitant les failles critiques, Rust devient un **outil facilitateur** pour atteindre les objectifs du RGPD et réduire les risques juridiques liés à la protection des données.

- Rust dans 5 à 10 ans : tendances lourdes

Plusieurs mouvements structurants se dessinent.

1. Rust en systèmes critiques

Avec la formalisation du langage, Rust sera probablement utilisé dans :

- l’aéronautique,
- le ferroviaire,
- les systèmes médicaux,
- l’automobile autonome.

2. Rust dans la cybersécurité

Les équipes de sécurité adoptent Rust pour développer :

- scanners réseau,
- outils de forensic, (investigation numérique)
- agents sur endpoints, (Un endpoint est tout appareil que les employés utilisent pour se connecter aux réseaux d'entreprise représente un risque potentiel que les cybercriminels peuvent exploiter pour voler des données d'entreprise)
- solutions zero-trust. (Les produits dits Zero Trust sont vus comme des solutions permettant de pallier certaines limitations des mesures traditionnelles telles que la protection des flux par VPN ou le filtrage réseau par des pares-feux périmétriques)

3. Rust + WebAssembly

Le binôme Rust/WASM va probablement devenir une alternative sérieuse aux runtimes lourds et aux conteneurs pour certaines charges légères.

4. Rust sur mobile

Tauri pourrait devenir une solution multiplateforme légère pour le desktop ET le mobile, ce qui changerait le paysage.

5. Rust en edge computing

Avec la réduction du coût du compute, l'exécution de modèles ML ou de services à la périphérie du réseau se développe.
Rust a un rôle naturel dans cet environnement.

- Rust : un langage qui a trouvé sa place

Rust n’est pas un simple langage “à la mode”.
Il répond à un problème réel : faire du logiciel performant *et* robuste dans un monde où les attaques augmentent et où les systèmes deviennent de plus en plus complexes.

En 2025, Rust se place comme **une alternative crédible au C et au C++**, sans les remplacer totalement, mais en offrant une voie plus sûre et plus moderne pour de nouveaux projets ou des plans entiers d'infrastructures.

Les défis persistent, mais la trajectoire est claire :
Rust va continuer de gagner du terrain, surtout là où la fiabilité, la sécurité et les performances sont critiques.