



Presented By
Elies Jebri

Elies Jebri

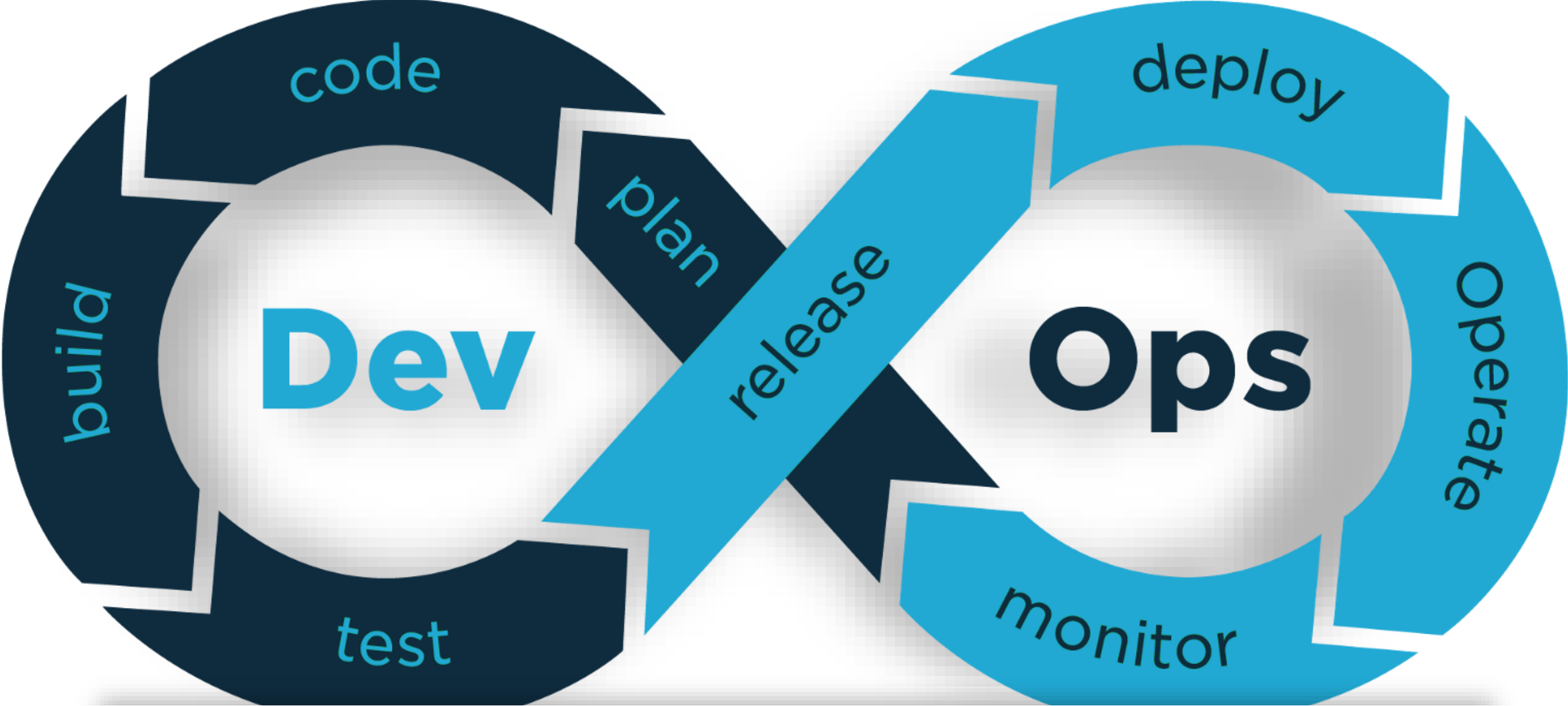
- **Titre:** Consultant expert en OSS et infrastructures IT.
- **Distinctions & expérience:**
 - Expert depuis 2000 auprès d'un grand nombre de compagnies locales et internationales.
- **Formation:**
 - Agrégé en informatique industrielle,
 - Certifié RHCE, LPIC-1,2,301/303/304, NCLA, MCITP, CCNA, CEH, TenStep, AWS Practitioner, AWS SysOps, AWS Architect, VMware Foundations, VMware Data Center Professional, VMware Certified Instructor, Huawei Cloud Services, LPI-DevOps, DevOps Generalist, Cisco ENCOR, DCCOR, AZ-900, VMCE
- **Contact:**
elies.jebri@gmail.com / ejebri@clevory.com



Introduction au DevOps

- Pourquoi parler de DevOps avant DevSecOps ?
- Exemple fictif : l'entreprise Delta
- Objectif : comprendre les rôles et problématiques Dev & Ops





QU'EST CE QUE LE DEVOPS

Mais d'abord

Bob le développeur

Missions principales :

- Développement de nouveaux produits
- Nouvelles fonctionnalités
- Correctifs de bugs
- Mises à jour de sécurité

Problèmes rencontrés :

- Longs délais avant mise en production
- Différences entre environnements (dev vs prod)
- Risques d'incidents/bugs

Alice l'opérationnelle

Missions principales :

- Administration des systèmes & réseaux
- Sécurité du SI
- Support utilisateurs
- Évolution de l'infrastructure

Difficultés :

- Outils obsolètes face à la croissance
- Charge de support accrue
- Manque de compréhension mutuelle avec les Devs
- Concept du Wall of frustration

Vers une définition du DevOps

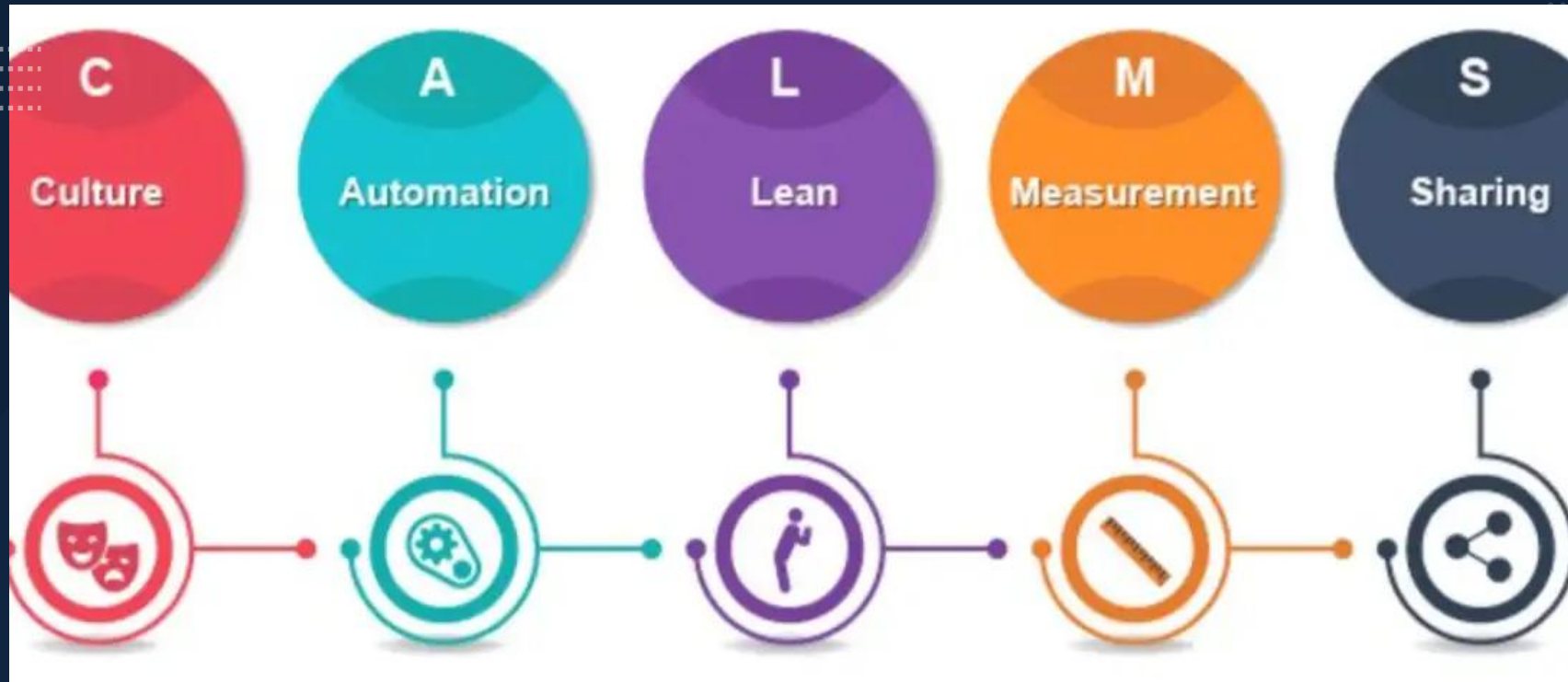
- Référence : The DevOps Handbook (Gene Kim, Jez Humble, etc.)
 - DevOps = état d'esprit + pratiques
 - Collaboration transversale Dev, Ops, Qualité, Sécurité
 - Objectif commun : succès global de l'organisation
 - Flux rapide + stabilité + sécurité

Exemples concrets

- Netflix :
- 220M abonnés, 150M heures/jour
- 4000 déploiements/jour (Spinnaker)
- Amazon (AWS) et Google :
- Résilience, disponibilité et efficacité accrues grâce au DevOps

Récapitulatif et transition DevSecOps

- Dev = développement (ingénieurs applicatifs, logiciels)
- Ops = opérations (déploiement, gestion systèmes/réseaux)
- Sec = sécurité (confidentialité, intégrité, disponibilité)
- DevOps = culture de collaboration + communication
- Automatisation des processus → livraison plus rapide et fiable
- 👉 Prochaine étape : intégrer la sécurité = DevSecOps

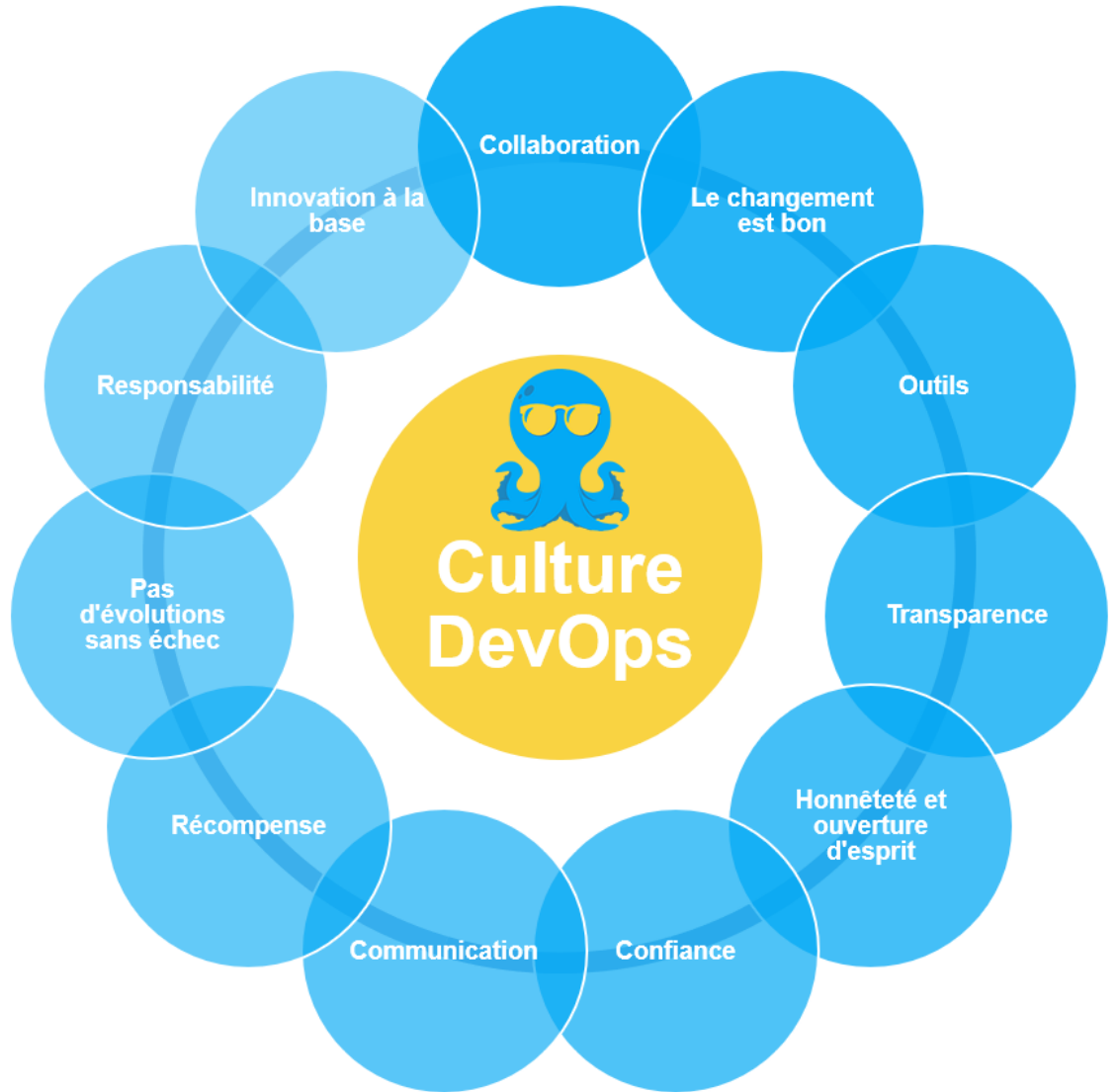


Le cadre CALMS

- DevOps n'a pas de référentiel officiel comme Agile/ITIL
- CALMS = cadre pour évaluer la maturité DevOps
- 5 piliers : Culture, Automatisation, Lean, Mesure, Sharing
- Permet d'accompagner le changement organisationnel

Pilier 1 : Culture

- Implique tous les acteurs (techniques, managers, sécurité)
- Culture de l'expérimentation (droit à l'erreur)
- Apprentissage par l'échec → maturité progressive
- Résilience opérationnelle : PCA et DR pour garantir la continuité



Pilier 2 : Automatisation



CI/CD → automatisation tests, build, déploiements



Infrastructure as Code (IaC) & Cloud



Kubernetes et orchestration des conteneurs



Réflexion clé : Qu'est-ce qui peut être automatisé ?

Pilier 3 : Lean (zéro déchet)

- Inspiré du Lean manufacturing
- Objectif : maximiser la valeur, réduire les gaspillages
- Limiter les goulots d'étranglement
- Releases petites et fréquentes = plus faciles à corriger

Pilier 4 : Mesure

- Mesurer pour prouver les gains → adoption facilitée
- TDD (Test Driven Development) : écrire les tests avant le code
- Règle clé : aucune modification sans mesure d'efficacité

Exemples de mesures clés

- Nombre/fréquence des déploiements
- Taux d'échec vs succès des déploiements
- MTTD (Mean Time to Discovery)
- MTTP (Mean Time to Patch)
- MTTR (Mean Time to Recovery)
- Lead Time
- Disponibilité des services, anomalies détectées
- Référence : DORA Metrics (Google)

Pilier 5 : Sharing (partage)

- Collaboration et communication au cœur du DevOps
- Partage entre Dev, Ops, Sec... et aussi avec les clients
- Transparence → renforce la confiance client
- Clés de succès : Focus équipe, lever les barrières, collaboration à distance

Transition vers DevSecOps

- DevSecOps = DevOps + sécurité intégrée dès le départ
- Security by Design et Shift Left
- Sécurité dans le workflow CI/CD (code, conteneurs, Cloud)
- Dans DevSecOps : la sécurité est l'affaire de tous
- DevSecOps = DevOps "bien fait" + la sécurité partout

LA TRANSFORMATION DES ÉQUIPES PAR LE DEVSECOPS

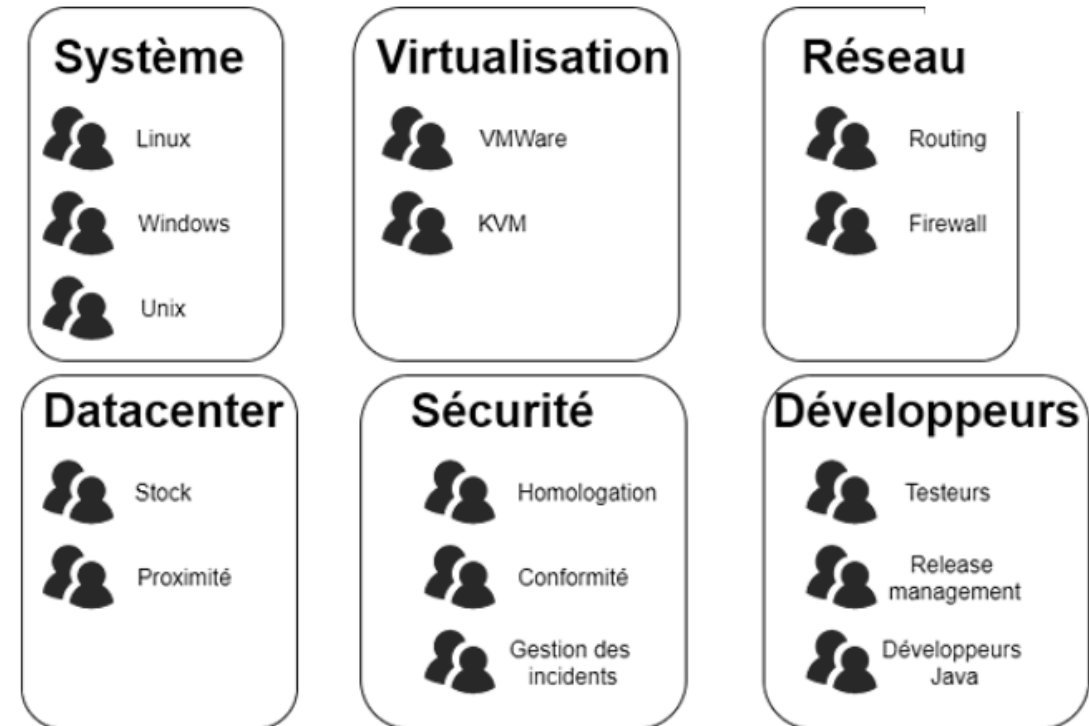


Le business d'aujourd'hui

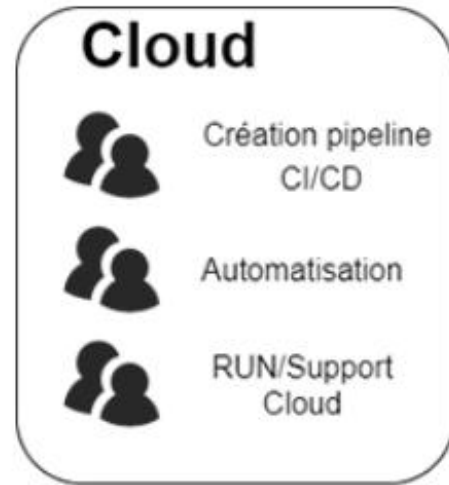
- L'IT = cœur du business des entreprises modernes
- Pression : livrer vite + qualité + sécurité
- Cyberattaques de plus en plus fréquentes
- DevSecOps = réponse à cette équation

L'ancien monde en silos

- Organisation en équipes séparées : Dev, Réseau, Linux, Virtualisation, Sécurité...
- Responsabilités floues = incidents difficiles à résoudre
- Exemple : bug → code ? réseau ? système ? droits ?



Le changement DevOps

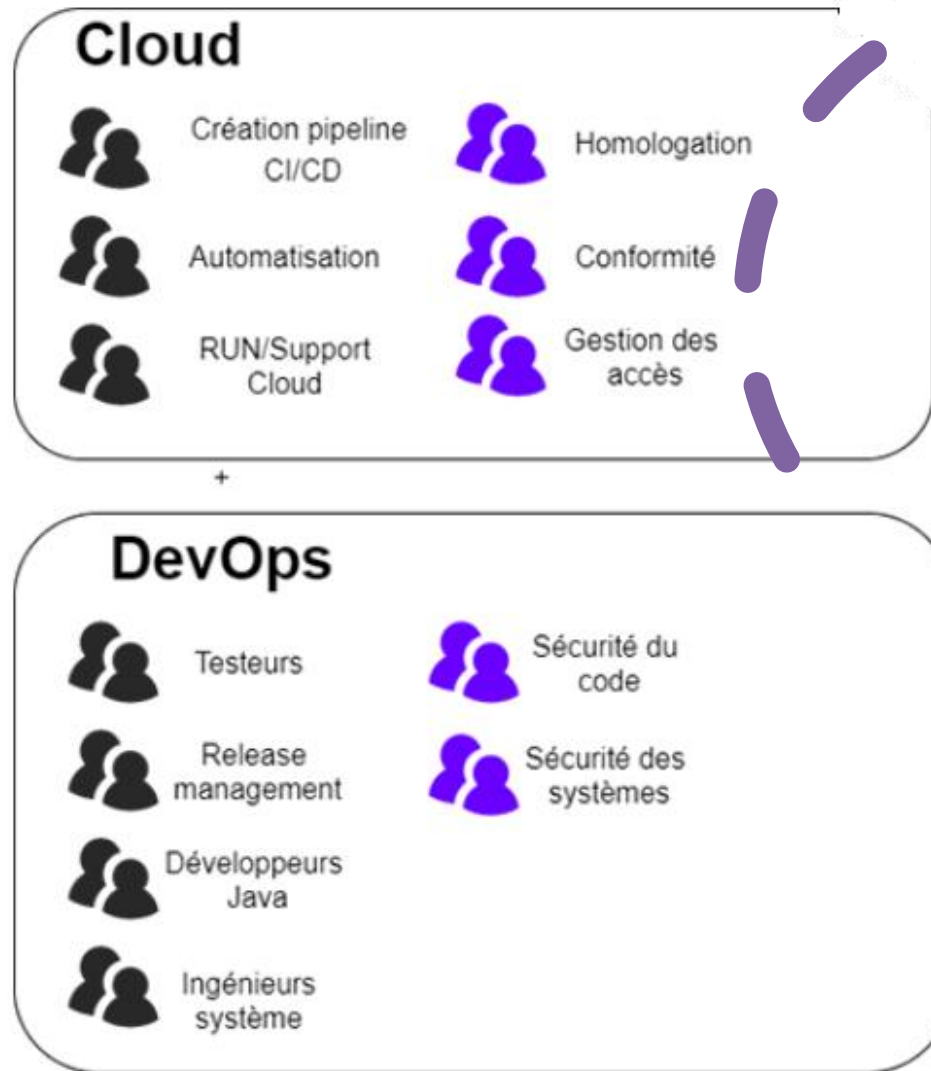


+



- Cloud + DevOps = équipes de delivery intégrées
- Devs + Ops + Cloud → une seule équipe
- Pipelines CI/CD pour autonomie et rapidité
- Mais la sécurité reste à l'écart...

L'optimisation DevSecOps



- Sécurité intégrée dans les équipes Dev et Cloud
- Méthodes Cloud réutilisées par tous (IaC, CI/CD sécurisés)
- Responsabilités claires en matière de sécurité



Clés d'adoption du DevSecOps

- Intégrer des Security Champions dans chaque équipe
- Automatiser au maximum (tests, scans, conformité)
- Utiliser les guides CIS Benchmarks
- Référentiels Cloud (ex : AWS IAM Best Practices)

Extrait CIS Benchmarks

- Benchmarks par technologie (Linux, Windows, DB, Kubernetes, Cloud...)
- Recommandations concrètes : durcissement OS, IAM, logs, chiffrement...
- Adoption = gain de maturité sécurité rapide

Vers les T-Shaped Peoples

- DevSecOps = transformation des équipes ET des individus
- T-Shaped = compétences larges (transverses) + expertise forte
- Tout le monde doit maîtriser les bases sécurité et gestion des risques

Conclusion de la transformation

- DevSecOps = évolution logique du DevOps
- Organisation + pratiques + culture sécurité
- Bénéfices : rapidité, qualité, sécurité → ensemble
- Transformation humaine aussi importante que technique

TRANSFORMATION DU BUSINESS PAR LE DEVSECOPS

Une nouvelle façon de travailler



DevSecOps = transformation technique et business

Coopération Dev, Ops, Sec + Business

Automatisation et “Shift Left”

Mesure de l'efficacité → adoption facilitée

La règle du IN (Input, Involve, Invest)

Impliquer les
stakeholders dès le
début

Contributions
précoces = éviter
de partir dans la
mauvaise direction

Anticiper audits et
conformité en
amont

Investir temps et
ressources
(formation,
architecture)

Le “Shift Left”

Tests de sécurité
le plus en amont
possible

S’applique au
code, à
l’architecture, aux
images Docker...

Exemple : images
durcies avant
d’être mises à
dispo des Devs

Moins de pertes
de temps, plus de
sécurité dès le
début

Quelques statistiques clés

93 % des entreprises →
stratégie digitale
adoptée ou en cours

Objectifs : productivité
(51 %), sécurité des
données (40 %),
meilleure visibilité

Axes : IA, ML, Cloud-
Native, cybersécurité,
AR/VR

DevSecOps, business et sécurité

Sécurité intégrée
au quotidien =
“Secure by Design”

Outils : SIEM
(Splunk, ELK,
Sentinel,
GuardDuty)

Mise en
conformité (ISO,
PCI-DSS, sectoriels)

Mutation business
profonde, pas juste
du code

IA GÉNÉRATIVE & DEVSECOPS

Introduction à l'IA générative

IA générative = création
de contenus (texte,
code, images,
musique...)

Basée sur GANs et LLMs
(GPT-4o, Gemini, etc.)

Applications IT : analyse
de code,
documentation,
détection de
vulnérabilités

Impact sur les piliers CALMS

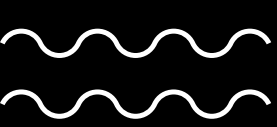
Culture :
collaboration &
innovation
(Copilot, ChatGPT)

Automatisation :
scripts IaC/CI-CD
générés

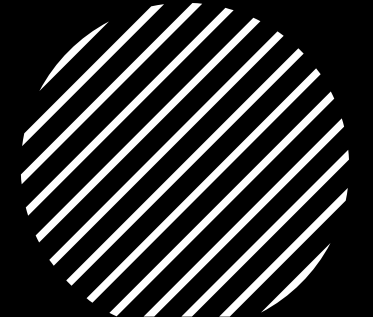
Lean : élimination
du waste via
automatisation

Mesure : analyse
big data & rapports
temps réel

Sharing :
documentation et
handbooks
automatisés



Exemples d'utilisation



Testing
amélioré par
IA

Analyse de
code
renforcée

Correction &
patch
automatiques

Défis et considérations éthiques

Biais des modèles → décisions injustes



Coût élevé des requêtes LLMs

Confidentialité des données sensibles

Confiance, transparence, responsabilité

Conclusion et vision

IA générative = accélérateur puissant du DevSecOps



Automatisation + détection + documentation

Attention aux biais, éthique et gouvernance

Vision : IA copilote (Vibecoding), humain décideur

CULTURE ET CONNAISSANCES EN CYBERSÉCURITÉ

Meilleures pratiques pour DevSecOps

Vocabulaire DevSecOps - Introduction

Sécurité intégrée dès le
début du cycle : shift-
left

Formation des équipes :
attaques, outils, bonnes
pratiques

Culture : communication, personnes, processus & technologie

Un leadership fort impulse une culture favorable au changement.

En DevSecOps, clarifier qui est responsable de la sécurité (processus & produit).

Les devs/ingénieurs deviennent propriétaires du processus et responsables de leur travail.

Les équipes conçoivent leur propre système de travail (technos, protocoles, workflow) adapté au projet.

En donnant de l'autonomie, l'équipe devient partie prenante et engagée sur les résultats.

Traçabilité, auditabilité & visibilité

Traçabilité : relier
exigences → code
→ configurations
sur tout le cycle.

- Favorise conformité, réduit les bugs, renforce la sécurité & la maintenabilité.

Auditabilité :
contrôles
techniques,
procéduraux et
administratifs

- Documentés, vérifiables, appliqués par tous pour répondre aux exigences de conformité.

Visibilité :
supervision
continue pour
mesurer le “pouls”
des opérations

- Alertes en temps réel, détection des changements & cyberattaques, redevabilité sur tout le cycle de vie.

CIA -
Confidentialité,
Intégrité,
Disponibilité

Confidentiality : accès limité
aux autorisés

Integrity : données non
altérées sans autorisation

Availability : données
accessibles quand nécessaire

Le SIEM - Rôle et objectifs

Agrégation des
données

Corrélation des
événements

Collecte temps réel

SIEM -
Corrélation et
archivage

Profil type de
l'infrastructure

Détection anomalies en
temps réel

Archivage long terme
pour conformité

Solutions SIEM

Splunk (Commercial, puissant mais cher)

ELK (Open Source, modulaire, plus complexe)

IBM QRadar (Commercial, automatisation avancée)

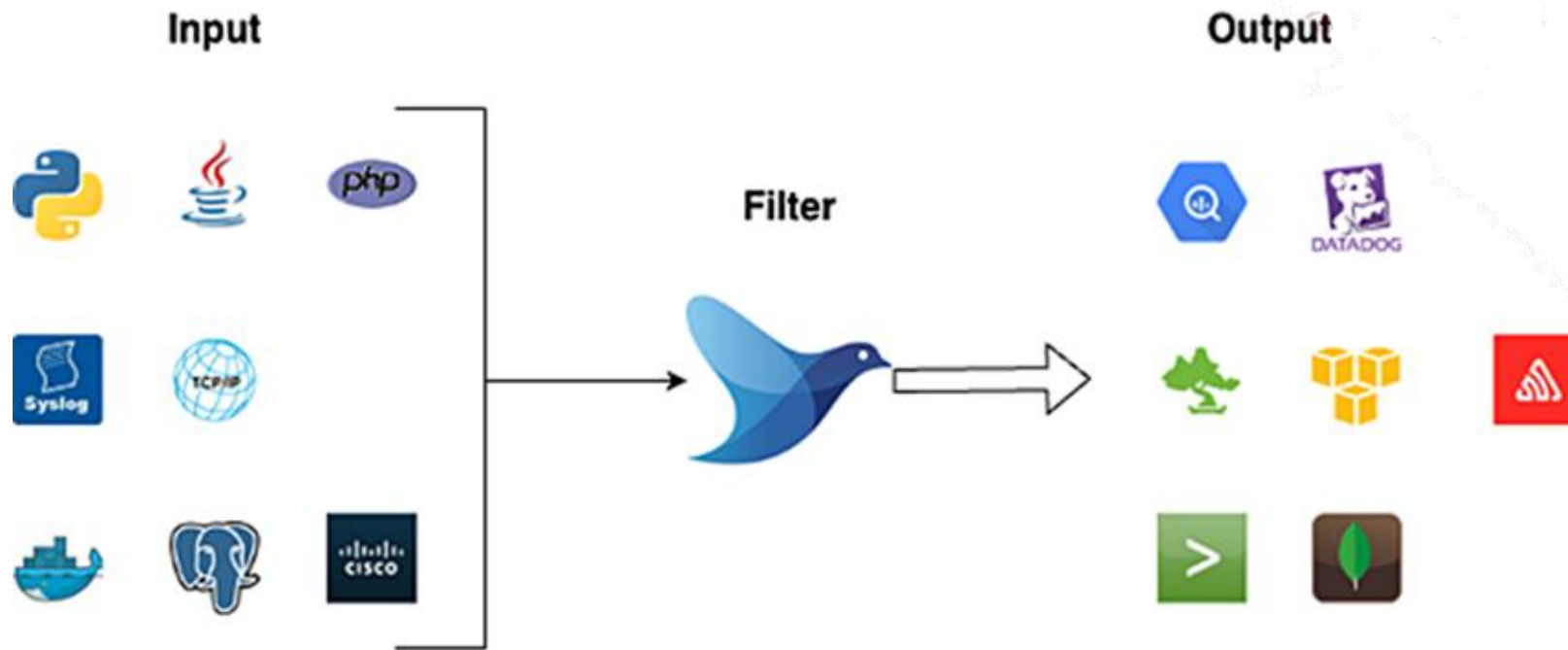
Wazuh (Open Source, léger pour PME)

Comparatif des solutions SIEM

Solution	Type	Cas d'usage	Avantages	Inconvénients
Splunk	Commercial	Grandes entreprises Conformité réglementaire	Interface intuitive Capacité à scale	Coût très élevé pour des volumes de données importants
ELK	Open source	Environnements agiles Startups	Modulaire Gratuit	Complexité de mise en œuvre
IBM QRadar	Commercial	Grandes entreprises avec de fortes exigences réglementaires	Automatisation plus élaborée	Courbe d'apprentissage élevée
Wazuh	Open source	PME	Léger	Fonctionnalités limitées

Intégration SIEM & DevSecOps avec agents de collecte Fluentd/Filebeat

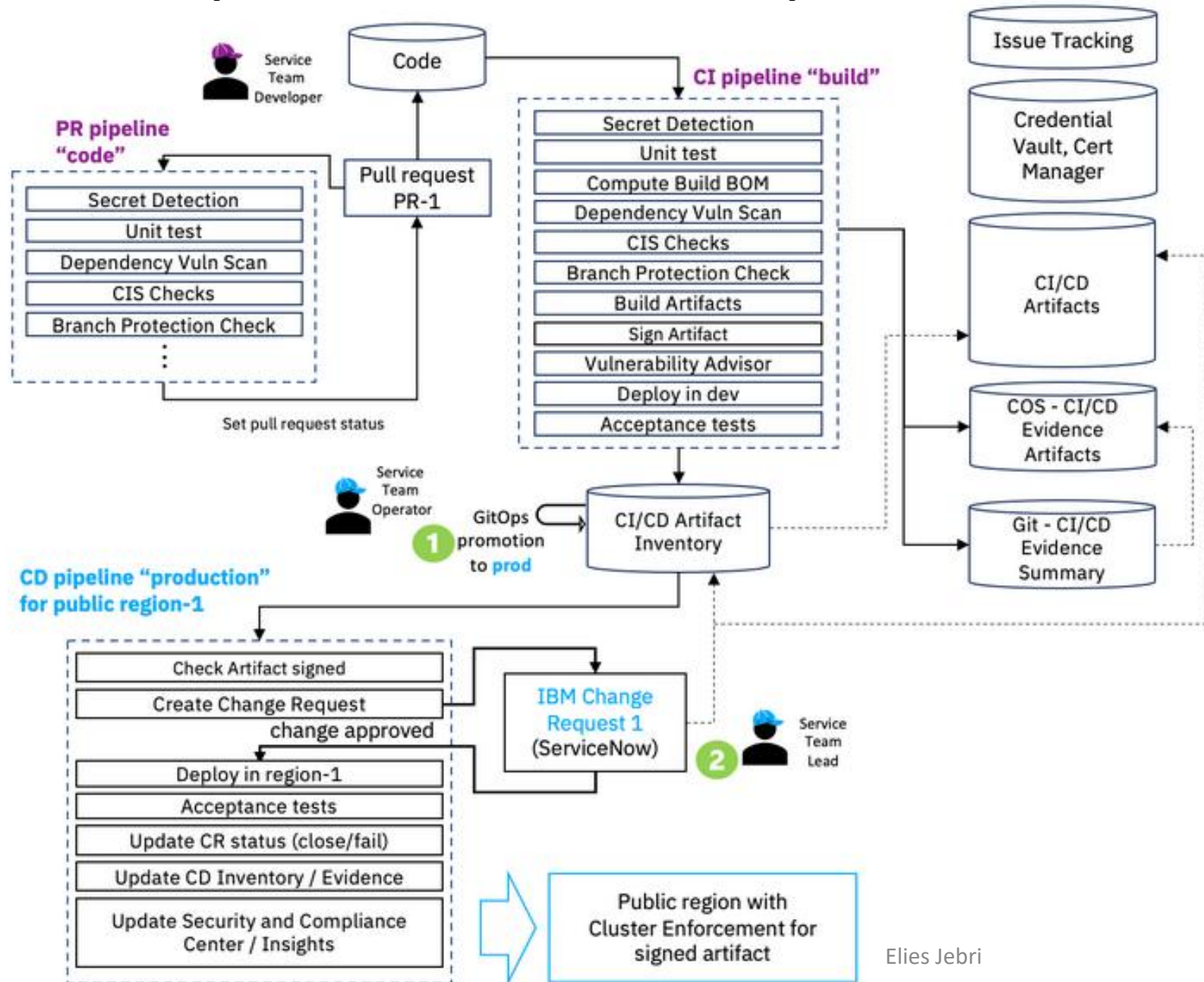
- Collecte des logs des pipelines CI/CD
- Stratégique pour :
 - détecter les anomalies en temps réel, comme les activités suspectes ou les modifications non autorisées à l'intérieur des pipelines CI/CD ;
 - automatiser les réponses en arrêtant les processus malveillants et en bloquant les IP de manière automatique ;
 - maintenir la conformité de l'infrastructure pour respecter les exigences et réglementations comme le RGPD ou le PCI-DSS.
- Fluentd : input, filter, output
- 500+ plugins, intégration SIEM & Cloud



- Fluentd repose sur trois concepts :
 - **Input** : collections des logs provenant de diverses sources (Python, HTTP, Docker, Cisco, etc.) ;
 - **Filter** : transformation des logs et ajouts d'informations ;
 - **Output** : transmission des logs formatés et enrichis à des systèmes cibles (Elasticsearch, Kibana, Datadog, etc.).

Fluentd

Principales caractéristiques de l'architecture DevSecOps



Git Issues

Change Request

- Cumulative list of changes
- Traceability: Issue/Pull Request
- Bill of material
- Status on each control:
 - ✓ Secret detection
 - ✓ Unit test passed
 - ✓ Dependency code vulnerability scan (CVE)
 - ✓ Signed artifact
 - ✓ Binary vulnerability scan – VA (CVE)
 - ✓ Acceptance tests in PRE-PROD
 - ...
- Links to Evidence (COS)



Processus de mise en œuvre de l'architecture DevSecOps

- La validation de demande d'extraction ou de fusion utilise un pipeline de demande d'extraction spécifique qui renvoie le statut à la demande d'extraction. cette fonction permet aux développeurs d'identifier les problèmes au début du cycle de développement.
- L'intégration est réalisée à l'aide d'un pipeline d'intégration continue qui implémente de nombreux contrôles prêts à l'emploi et collecte des preuves normalisées pour ces intégrations dans un référentiel de preuves. Ce pipeline écrit également des métadonnées sur les artefacts à déployer dans un référentiel d'inventaire d'artefacts, activant ainsi les pratiques GitOps.
- La livraison est effectuée en utilisant un pipeline de déploiement continu pour traiter le contenu de l'inventaire des artefacts et collecter des preuves pour générer une demande de changement pour chaque déploiement. Ce pipeline peut également enregistrer des faits de conformité dans Security and Compliance Center, lesquels peuvent être évalués par rapport à un profil de conformité.



ÉLÉMENTS DE BASE DE LA CHAÎNE D'OUTILS DEVSECOPS

Pipeline de demande d'extraction (Pull Request Pipeline)

- Processus automatisé qui exécute des vérifications de conformité et de qualité de code sur chaque demande d'extraction avant la fusion dans la branche principale (master ou main).

Principales Étapes

- Ouverture ou mise à jour d'une Pull Request
 - Déclenche automatiquement le pipeline.
- Exécution des vérifications prédéfinies
 - Tests unitaires et d'intégration
 - Analyse de sécurité et de vulnérabilités
 - Contrôle des normes de codage et de style
- Résultats de conformité
 - ✓ Tous les tests passent → Fusion autorisée
 - ✗ Échec d'un test → Blocage automatique de la fusion

- Automatise la construction, la vérification et la signature des artefacts déployables issus des référentiels d'applications.
- Analyse & Tests du Code
 - Vérification automatique de la qualité du code
 - Exécution des tests unitaires et d'intégration
 - Analyse de sécurité (SAST, DAST, dépendances)
- Génération des Artefacts
 - Construction des packages, images Docker ou binaires
 - Analyse de vulnérabilités post-build
 - Signature cryptographique des artefacts
- Validation & Traçabilité
 - Marquage "Prêt pour déploiement"
 - Collecte de preuves à chaque étape (logs, rapports, scans)
 - Corrélation entre les artefacts, les résultats de tests et la gestion des changements

Pipeline d'Intégration Continue (CI Pipeline)

Bonnes pratiques dans la chaîne d'outils d'intégration continue

La mise en œuvre de la chaîne d'outils d'intégration continue DevSecOps suit ces pratiques.

- Exécute un scanner de code statique sur les dépôts d'application pour détecter les secrets dans le code source de l'application et les paquets vulnérables qui sont utilisés comme dépendances de l'application.
- Construit une image de conteneur sur chaque Git commit, en définissant une balise basée sur le numéro de build, l'horodatage et l'ID de commit pour la traçabilité.
- Teste le Dockerfile avant la création de l'image.
- Enregistre l'image construite dans un registre d'images privé.
- Configure automatiquement les autorisations d'accès pour le déploiement du cluster cible en utilisant des jetons API à révoquer.
- Recherche les failles de sécurité dans l'image du conteneur.
- Ajoute une signature Docker en cas de réussite.
- Insère automatiquement la balise built-image dans le manifeste de déploiement.
- Utilise un espace de noms explicite, dans le cluster, pour isoler chaque déploiement (commande `kubectl delete namespace`).
- Construit, valide et déploie automatiquement tout code fusionné dans la branche cible du dépôt Git.

- Automatise le déploiement des artefacts validés vers des environnements cibles (préproduction, production...) tout en assurant la traçabilité complète des changements.
- Préparation du Déploiement
 - Récupération des artefacts signés depuis le pipeline CI
 - Vérification des versions et des dépendances
 - Génération du résumé de la demande de changement (Change Request Summary)
- Déploiement Contrôlé
 - Déploiement automatique vers les environnements cibles
 - Intégration avec les outils d'orchestration (Ansible, ArgoCD, Jenkins, etc.)
 - Possibilité de déploiement progressif / canary / blue-green
- Collecte des Preuves et Journaux
 - Capture de tous les logs, rapports, artefacts et preuves de conformité
 - Téléversement automatique dans le casier de preuves (Evidence Locker)
 - Suivi des changements pour audit et conformité

Pipeline de Déploiement Continu (CD Pipeline)

Bonnes pratiques pour déployer une application sécurisée

Pour déployer une application sécurisée, la chaîne d'outils de déploiement continu DevSecOps ne contient qu'un seul pipeline avec les pratiques suivantes.

- Automatise la gestion des changements pour aider les développeurs, les approbateurs et les auditeurs à suivre les déploiements du point de vue de la conformité.
- Crée un résumé des preuves à partir des preuves collectées au cours de l'exécution du pipeline d'intégration continue associé pour un ensemble donné de changements.
- Crée une demande de changement dans le référentiel de gestion des changements basé sur Git Repos and Issue Tracking et ajoute des preuves de déploiement.
- Utilise le référentiel d'inventaire pour promouvoir les artefacts construits dans les environnements de déploiement tels que la mise en scène et la production.
- Valide la demande de modification et l'approuve automatiquement si tous les critères de déploiement sont remplis.
- Si une demande de modification est approuvée ou marquée comme urgente, le pipeline déploie l'image de l'inventaire vers la production.

Bonnes pratiques pour la chaîne d'outils CI IaC

La mise en œuvre de la chaîne d'outils d'intégration continue (CI) pour Infrastructure as Code (IaC) DevSecOps suit ces pratiques.

- Exécute un scanner de code statique sur les dépôts de code de l'infrastructure et effectue des contrôles de linting Terraform sur le code de l'infrastructure.
- Exécute des contrôles de conformité sur le code de l'infrastructure afin de détecter les secrets et les failles de sécurité.
- Construit des artefacts sur chaque Git commit.
- Stocke les métadonnées des artefacts construits dans le référentiel d'inventaire.
- Construit et valide automatiquement tout code qui est fusionné dans la branche cible du dépôt Git.

Pipeline de Conformité Continue

- Assure une surveillance post-déploiement des applications et artefacts en production, afin de maintenir leur conformité et leur sécurité dans le temps.
- Analyse Périodique des Artefacts Déployés
 - Scan régulier des artefacts en production
 - Détection de nouvelles vulnérabilités apparues depuis le dernier déploiement
 - Corrélation avec les référentiels source pour évaluer l'impact
- Suivi des Écarts et Conformité Réglementaire
 - Contrôle des écarts de conformité par rapport aux politiques internes ou réglementaires
 - Suivi automatique des dates d'échéance de correction
 - Génération d'alertes ou de tickets correctifs

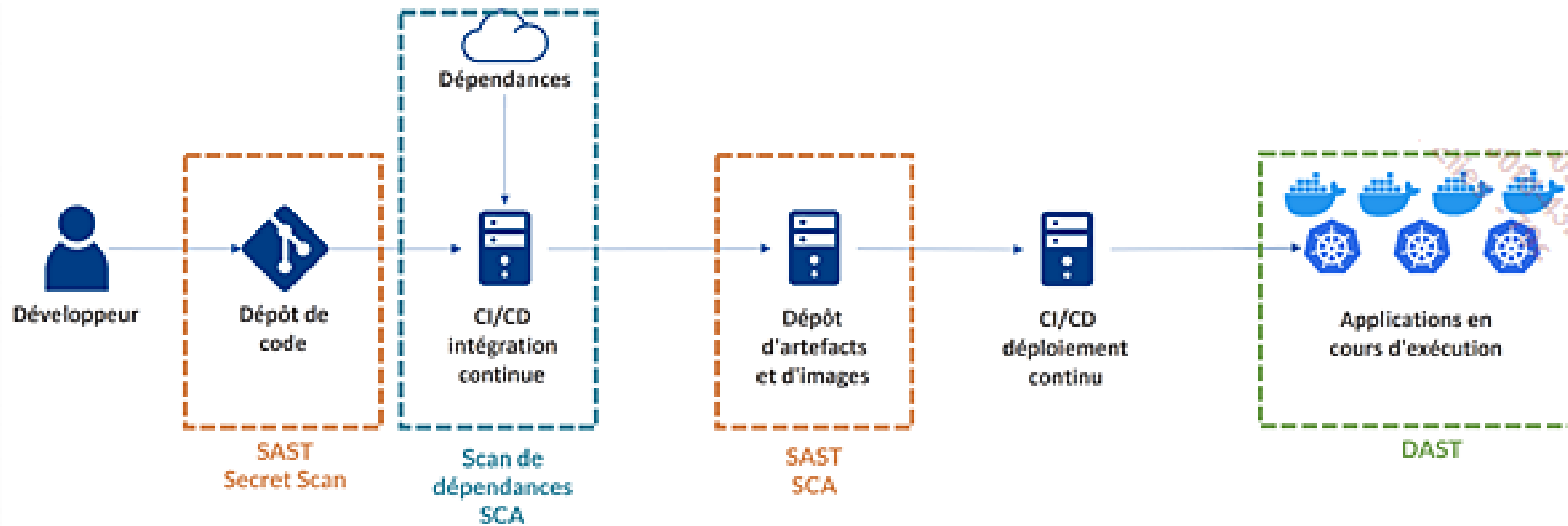
Bonnes pratiques dans la chaîne d'outils de conformité continue

Le pipeline CC est utile pour une analyse continue des artefacts déployés existants et de leurs référentiels sources, indépendamment de votre calendrier de déploiement.

- La mise en œuvre de la chaîne d'outils de conformité continue DevSecOps suit ces pratiques.
 - Exécute un scanner de code statique à intervalles prédéfinis sur les dépôts d'applications.
 - Détecte les secrets pour empêcher les fuites dans le code source de l'application.
 - Analyser l'image du conteneur pour détecter d'éventuelles failles de sécurité.
 - Ouvre des questions d'incident pour suivre les problèmes trouvés au cours de l'exécution du pipeline et est marqué d'une date d'échéance.
 - Génère un fichier summary.json et le stocke pour résumer les résultats de l'analyse.



DEVSECOPS TOOLS



Threat Modeling

- Le Threat Modeling correspond à la première véritable étape à effectuer.
- Viennent ensuite les vérifications liées aux tests de « precommit » et de vérification des secrets au sein des dépôts de code. Nous retrouvons le SCA (Software Composition Analysis) permettant de référencer et de vérifier l'utilisation de dépendances qui pourraient être vulnérables, puis le SAST et le DAST.

SCA (Software Composition Analysis)

Sécuriser les dépendances open-source (90%) au cœur du DevSecOps

- Le SCA est historiquement centré sur l'open source
- Mais le SCA peut aussi analyser des composants propriétaires

Objectif :

- Comprendre comment identifier, analyser et maîtriser les risques liés aux composants tiers utilisés dans les applications modernes.
- Gérer les milliers de dépendances open source utilisées dans les projets modernes ;
- Détecter les vulnérabilités CVE et les problèmes de licences ;
- Produire des SBOM (Software Bill of Materials) listant tout ce qui compose un logiciel.

Contexte

- 90% du code d'une application moderne est open-source ou tierce.
- Ces composants (bibliothèques, frameworks, conteneurs...) peuvent contenir :
 - des vulnérabilités connues (CVE),
 - des licences non conformes,
 - des versions obsolètes ou compromises.
- Les attaquants exploitent ces failles dans la supply chain logicielle.

Ce que le SCA couvre réellement dans un projet hybride

Type de dépendance	Exemples	Analyse SCA possible ?	Base CVE publique ?
Open Source (OSS)	requests, log4j, lodash, flask	✅ oui	✅ oui
Commerciale / propriétaire	Oracle JDBC driver, SAP SDK, DLL interne	⚙️ partiel (inventaire seulement)	❌ non
Framework maison	core-lib interne, helper API	⚙️ oui (inventaire)	❌ non
Container image base	python:3.10-slim, ubuntu:22.04	✅ oui	✅ oui





Fonctionnement du SCA

- **Inventaire (SBOM)** : génération du Software Bill of Materials
 - liste complète des bibliothèques, versions, fournisseurs, licences
- **Scan** : comparaison avec les bases de vulnérabilités (NVD, GitHub Advisory, OSS Index...)
- **Analyse** : évaluation de la gravité (CVSS), impact et correctifs disponibles
- **Reporting & remediation** : priorisation et mise à jour des dépendances vulnérables
- **Monitoring continu** : détection des nouvelles vulnérabilités post-déploiement
- Sortie typique :
 - SBOM.json
 - Rapport SCA HTML / JSON (avec CVE, CVSS, licences)

CVE Common Vulnerabilities and Exposures

-
- Base de données publique mondiale qui recense toutes les failles de sécurité connues dans les logiciels, bibliothèques, systèmes d'exploitation, paquets, etc.
 - Chaque faille documentée reçoit un identifiant unique CVE, du type :
CVE-2023-12345

Criticité (CVSS Score)

Niveau	Score	Couleur	Signification
LOW	0.1 – 3.9		Faible impact
MEDIUM	4.0 – 6.9		Impact modéré
HIGH	7.0 – 8.9		Impact fort, exploitable
CRITICAL	9.0 – 10		Impact majeur, prise de contrôle possible

Outils SCA populaires

Outil	Mainteneur	Type	Intégration CI/CD
Trivy	Aqua Security	Open Source	Docker, GitLab, GitHub
OWASP Dependency-Check	OWASP	Open Source	Java, Maven, Gradle
Syft + Gype	Anchore	Open Source	Containers, Images
Snyk	Snyk Ltd	SaaS / Cloud	GitHub, GitLab, Jenkins
Whitesource / Mend	Mend.io	Commercial	Multi-langages
GitLab SCA	GitLab	Intégré	Auto dans Security Dashboard

Avantages du SCA

Bénéfices pour l'entreprise

- Réduction du risque lié à la supply chain logicielle
- Conformité réglementaire (ISO 27001, NIST, DORA, etc.)
- Déploiement plus rapide grâce à l'automatisation du patching
- Visibilité sur les composants utilisés (SBOM)
- Amélioration de la confiance client / partenaire

Limites & Bonnes pratiques

Limites :

- Détection incomplète si dépendances non déclarées
- Peu ou pas d'analyse du code propriétaire
- Fausse sécurité si le SBOM n'est pas mis à jour

Bonnes pratiques :

- Générer un **SBOM** à chaque build
- Scanner **toutes les dépendances (directes & transitives)**
- Activer le **monitoring post-déploiement**
- Automatiser la **remédiation** via CI/CD

SAST (Static Application Security Testing)

- Analyse du **code source** avant compilation
- Objectif : **détecter les vulnérabilités** dans le code dès les premières phases du cycle de développement
- Le SAST analyse :
 - les **mauvaises utilisations** de fonctions ou bibliothèques,
 - les **pratiques de développement non sécurisées**.
- Avantage : corriger **avant compilation** → gain de temps & réduction du coût de correction.

SAST Principe de fonctionnement

- Le SAST **n'exécute pas** le code.
Il **inspecte le code source statiquement** pour trouver les failles.
- Domaines analysés :
 - Fichiers de configuration
 - Syntaxe et logique applicative
 - Validation des entrées/sorties
 - Structures de données
 - Matériel cryptographique et classes

Débat : automatisation vs formation

Deux approches :

- **Tout automatiser** avec des outils SAST puissants
- **Former les développeurs** aux pratiques sécurisées
- La réalité efficace : **un équilibre entre les deux**
 - Former les équipes + utiliser SAST pour renforcer la détection.
- Attention aux **faux positifs** :
 - Alertes signalées par le SAST mais non pertinentes selon le contexte.

Types de vulnérabilités détectées

- Les outils SAST permettent de repérer :
- **Secrets hardcodés** (mots de passe, clés API)
- **Mauvaises configurations** (headers manquants, erreurs exposées)
- **Injections SQL / XSS**
- **Variables non initialisées, boucles infinies**
- **Usage de cryptographie faible**
- SAST = cartographie des **risques structurels internes** à l'application

Intégration dans le pipeline



Pour maximiser l'efficacité :

- Intégrer le SAST **le plus tôt possible**
- Position idéale : **après lint, avant build**
- Peut être exécuté :
 - dans l'**IDE** du développeur
 - dans les **pipelines GitLab CI / Jenkins / GitHub Actions**

Avantage :

Détection précoce = correction immédiate = sécurité “by design”

Avantages & limites

 Avantages	 Limites
Détection précoce des failles	Faux positifs fréquents
Automatisation facile	Peu de contexte d'exécution
Réduction des coûts de correction	Peut nécessiter ajustement manuel
Compatible CI/CD et IDE	Nécessite formation minimale devs

DAST (DYNAMIC APPLICATION SECURITY TESTING)

DAST — Dynamic Application Security Testing

Scans 'black-box'
d'applications en cours
d'exécution

Complémentaire au
SAST et SCA ; se place
en fin de pipeline

Fonctionnement d'un scan DAST — Phases

Reconnaissance / crawling de
l'application

Identification des vulnérabilités
(matching vs DB)

Tentatives d'exploitation /
validation

Génération de rapport (sévérité,
exploitability)

Placement dans CI/CD

S'exécute après le déploiement
sur staging/preview

Ne pas scanner la prod sans
règles strictes

Utiliser environnements isolés
et données mock

DAST — Avantages & limites

Avantages : faibles faux positifs, langage-agnostic, simule attaques réelles

Limites : lent, nécessite app déployée, coverage dépend du crawler

Outils DAST — Open source & commerciaux

Open-source : OWASP
ZAP, W3af, Nikto

Commerciaux : Burp Suite,
Acunetix, Netsparker

Intégration : GitLab DAST
template, ZAP Docker in CI

Scans authentifiés & apps stateful

Gérer sessions & login
(headless browser)

Utiliser comptes test
éphémères et données seed

Limiter l'impact (rate limits,
sandbox)

Triage &
réduction des
faux positifs

Automatiser triage
initial (confidence)

Prioriser findings
exploitables

Enrichir avec
request/response & PoC

Bonnes pratiques pour DAST en DevSecOps

Scanner sur staging isolé; fail pipeline on critical findings

Combiner SAST/SCA/DAST + SBOM + image signing

Documenter runbooks & intégrer résultats aux dashboards

Examples – automation & snippets

```
OWASP ZAP (Docker): zap-  
baseline.py -t  
http://staging.example -r  
report.html
```

Enable GitLab DAST template
or run ZAP image in job