Thomas DIAS COSTA, Arthur HAZIZA et Marie-Alix JOHANNEL 3TD 09.02.2026

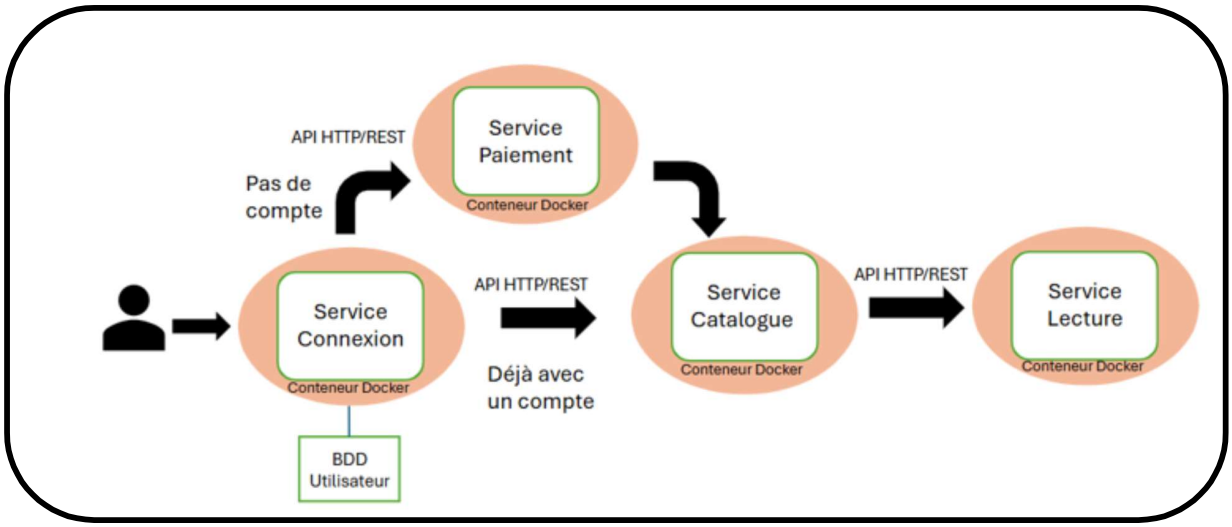
TP DevSecOps : Conception et implémentation d'un pipeline DevSecOps

Objectif : repenser et implémenter un pipeline de livraison selon une approche DevSecOps, en intégrant la sécurité de manière automatisée à chaque étape du cycle de vie logiciel.

Suite à un incident de sécurité, le pipeline existant ne permettra pas de détecter les vulnérabilités suffisamment tôt, de garantir un niveau de sécurité minimal avant le déploiement et d'avoir une visibilité claire sur l'état de sécurité du système.

I. Architecture applicative

Notre application est une plateforme de streaming type Netflix composée de microservices indépendants. Chaque service est conteneurisé avec Docker et les communication inter-services se font via des API HTTP/REST. Les déploiements sont fréquents grâce à un pipeline CI/CD.



A. Microservices et rôles

Service Connexion : point d'entrée principal de l'application qui permet à l'utilisateur de s'authentifier ou de créer un compte. Il gère la délivrance et la gestion des tokens d'accès. Il dépend d'une base de données utilisateur (comptes, hash des mots de passe, tokens).

Service Catalogue : gère le catalogue de contenus (séries/films).

Service Lecteur : gère l'affichage des informations liées au film sélectionné et sa lecture.

Service Paiement : gère les transactions liées aux abonnements et paiements ponctuels. Il communique avec un prestataire bancaire externe. Il dépend d'une base de données paiement et d'une API de paiement externe.

B. Communication REST entre services

Le service connexion est le point d'entrée de tous les parcours utilisateurs. Une fois authentifié, l'utilisateur peut accéder au service Catalogue pour consulter les contenus, au service lecteur pour lancer le visionnage et au service paiement pour souscrire à un abonnement ou faire un paiement ponctuel. Lors d'un nouvel abonnement ou paiement, le service paiement appelle un prestataire bancaire externe.

C. Dépendances critiques

Dépendances constituant des points de fragilité majeurs :
- Système d'exploitation (OS)
- Bases de données (utilisateur, paiement)
- Gestionnaire de tokens
- Prestataire de paiement externe
- Images Docker de base

D. Flux de données sensibles

Flux justifiant de l'intégration de contrôles de sécurité automatisé dans la pipeline :
- Identifiants utilisateurs : entre client et service connexion
- Tokens d'authentification : permet l'accès aux autres microservices
- Données de paiement : service paiement et API bancaire
- Données de navigation / visionnage : service lecteur

II. Description détaillée du pipeline CI/CD (DevSecOps)

Le pipeline CI/CD est conçu pour détecter les failles le plus tôt possible et empêcher le déploiement d'une version non sécurisée.

Phase de build : automatisée
- Entrées : code source, Dockerfile, base de données
- Contrôles : scan de secrets et images Docker, analyse statique du code (SAST)
- Sortie : image Docker, rapports de sécurité, arrêt pipeline si vulnérabilité critique

Phase de test : automatisée
- Entrées : image Docker de chaque microservice et BDD de test
- Contrôles : tests unitaires et vérification du bon fonctionnement des endpoints

- Sortie : rapport de tests et blocage du pipeline si tests échouent

Phase de déploiement : automatisée
- Entrées : image Docker validée par tests, configuration validée
- Contrôles : déploiement en environnement de staging et scan des API exposées
- Sortie : environnement staging accessible et rapports post-déploiement

Phase de supervision :
- Éléments supervisés : logs applicatifs, nombre connexions, erreurs authentification, anomalies d'accès, performances des microservices
- Permet une surveillance continue du système

III. Tableau d'analyse des risques et mapping des contrôles

Risque identifié	Origine technique	Impact	Contrôle DevSevOps
Fuite de secrets	Secrets (clé API, tokens, mdp) stockés en dur dans le code, les fichiers de configuration ou les Dockerfiles	Compromission des services, accès non autorisé, fuite de données sensibles	Secret scanning
Vulnérabilités	Code non sécurisé (injections SQL, injections de commandes, absence de validation des entrées)	Accès non sécurisé, exécution de code malveillant, fuite ou altération de données	SAST
Vulnérabilités dépendances et config Docker	Configuration Docker non sécurisé (image obsolète, paquets non MAJ, dépendances vulnérables)	Élévation de privilèges, compromission du conteneur, exécution de code malveillant	SCA / dépendances
Images conteneur vulnérables	Images Docker de base non mises à jour ou mal configurées	Exploitation de failles système, élévation de privilèges, compromission du conteneur	Scan image conteneur

Pour l'ensemble des risques identifiés, la probabilité est estimée à moyenne.

IV. Configuration des gates de sécurité

Des gates de sécurité sont mises en place tout au long du pipeline CI/CD afin de bloquer automatiquement le déploiement d'une version non conforme aux exigences de sécurité.

Les microservices sont exposés sur les ports suivants :
- Service Connexion : 5000
- Service Catalogue : 5001

- Service Lecteur : 5002
- Service Paiement : 5003

Gate en phase de build :

- Blocage du pipeline si un secret (clé API, mdp, token) est détecté dans le code ou les fichiers de configuration
- Blocage si des vulnérabilités critiques sont détectées dans les dépendances ou les images Docker.

Gate en phase de test :

- Blocage du pipeline si les tests unitaires ou fonctionnels échouent
- Garantit que le microservice fonctionne correctement avant déploiement

Gate en phase de déploiement :

- Blocage si le scan dynamique des API détecte une vulnérabilité de niveau élevé
- Vérification de la bonne exposition des ports et de la présence de mécanismes d'authentification

Ces gates garantissent que seules des versions fonctionnelles et sécurisées de l'application peuvent être déployées.

V. Guide de déploiement et supervision

Le déploiement de l'application en environnement de staging est entièrement reproductible grâce à Docker compose.

Etape à suivre :

- Se placer dans le dossier du projet contenants les microservices
- Supprimer les anciens conteneurs si nécessaire : « docker compose down -v »
- Construire les images Docker : « docker compose build »
- Déployer l'application : « docker compose up »
- Cliquer sur le lien localhost port 5000

La supervision permet de surveiller le bon fonctionnement et la sécurité de l'application après son déploiement.

Éléments supervisés :

- Logs de chaque microservice
- Nombre de connexions utilisateurs
- Erreurs d'authentification
- Erreurs lors des paiements
- Temps de réponse des services

Cette supervision permet de détecter rapidement des comportements anormaux, des tentatives d'attaque et des problèmes de performance ou de disponibilité.