**Étude des outils d’analyse et de gestion de l’obsolescence au sein du parc applicatif**

**1. Analyse de l’obsolescence via scripting GitLab et API**

Cette approche consiste à **exploiter les API GitLab** à l’aide de scripts d’automatisation (en Shell, Python ou Java) afin de **collecter les métadonnées techniques** de l’ensemble des projets applicatifs hébergés dans la forge GitLab.  
L’objectif est de **recenser les frameworks, langages de programmation et leurs versions** utilisés dans le parc, puis de les **comparer à la politique de support interne** et à la **roadmap technologique de l’organisation**.

* **Périmètre** : concerne **l’ensemble du parc applicatif**, y compris les projets ne disposant pas de pipeline CI/CD standard.
* **Finalité** : fournir une **vision macro** de l’état d’obsolescence du parc à travers des indicateurs consolidés (versions de Java, Spring Boot, Angular, etc.).
* **Résultat attendu** : un **tableau de bord global d’obsolescence technologique**, permettant de prioriser les actions de modernisation en fonction du risque associé à chaque technologie.

*Exemple d’usage* : extraction automatique des fichiers pom.xml ou build.gradle, identification des versions de framework (Spring Boot, Hibernate, etc.) et des langages (Java, Kotlin, etc.), puis comparaison avec les versions supportées par l’organisation.

**2. Vérification de l’obsolescence dans le pipeline CI/CD**

Cette approche consiste à intégrer une **étape de contrôle ("step")** dans le pipeline CI/CD des applications construites via GitLab CI, Jenkins ou OpenShift Pipelines.  
Ce contrôle compare automatiquement les **versions de frameworks et langages** détectées dans le code source aux **versions approuvées par la DSI** selon la **roadmap technologique officielle**.

* **Périmètre** : concerne uniquement les **applications utilisant le pipeline standard de build** défini par l’organisation.
* **Fonctionnement** :
  + Exécution d’un script d’audit en amont du build.
  + Analyse des fichiers de configuration (pom.xml, build.gradle, package.json, etc.).
  + Vérification de conformité avec la matrice de support.
* **Action possible** : en cas de **non-conformité** (framework ou langage obsolète), un **blocage automatique du pipeline** peut être activé, obligeant la mise à jour avant déploiement.

*Avantage clé* : détection précoce des technologies obsolètes avant toute mise en production.

**3. Vérification de l’obsolescence des dépendances via JFrog Xray**

L’outil **JFrog Xray** permet d’analyser les dépendances logicielles embarquées dans chaque application afin de **détecter les composants obsolètes ou présentant un risque opérationnel** (failles de sécurité, versions non supportées, etc.).

Deux modes d’exploitation sont possibles :

1. **Mode global (audit du parc complet)** :
   * Génération de rapports d’analyse Xray consolidés sur l’ensemble des projets hébergés dans Artifactory.
   * Exportation des résultats sous forme de **rapports de risques opérationnels**.
   * Vision **micro** sur l’obsolescence au niveau des dépendances.
2. **Mode CI/CD intégré** :
   * Intégration d’un **scan Xray à la demande** dans le pipeline standard.
   * Blocage automatique du pipeline en cas de dépendance critique ou obsolète.

* **Périmètre** :
  + Peut concerner **l’ensemble du parc** (via rapports consolidés),
  + ou uniquement les **applications intégrées au pipeline standard** (via scan automatisé).
* **Finalité** : offrir une **vision fine de l’obsolescence des dépendances tierces**, complémentaire à la vision macro obtenue par le scripting GitLab.

**4. Synthèse comparative**

| **Approche** | **Périmètre** | **Niveau d’analyse** | **Outil principal** | **Mode d’intégration** | **Action possible** |
| --- | --- | --- | --- | --- | --- |
| Scripting GitLab via API | Tout le parc applicatif | Macro (langage, framework) | GitLab API | Script automatisé (cron job ou batch) | Reporting global |
| Vérification CI/CD | Applications avec pipeline standard | Macro (langage, framework) | GitLab CI / Jenkins | Étape “check obsolescence” dans le pipeline | Blocage possible |
| Scan Xray JFrog | Tout ou partie du parc | Micro (dépendances) | JFrog Xray / Artifactory | Rapport ou scan intégré CI | Blocage possible |

**5. Conclusion**

La **combinaison de ces trois approches** permet de couvrir **l’ensemble du spectre de l’obsolescence logicielle** :

* **Vision globale** via GitLab API pour identifier les technologies à risque.
* **Vision intégrée au cycle de développement** via le pipeline CI/CD.
* **Vision approfondie** via JFrog Xray pour les dépendances tierces.

Cette stratégie garantit une **surveillance continue** du parc applicatif et une **maîtrise proactive de l’obsolescence**, tout en facilitant la planification des **mises à jour techniques** conformément aux standards de l’organisation.