**Migration des scripts Control-M vers Zabbix**

**Objectif**

Rationaliser la surveillance des systèmes informatiques par la réécriture des scripts shell en une langue standardisée et la migration de leur exécution de Control-M vers Zabbix.

# Phase 0 : Choix de la langue de programmation

* Analyse des cas d’usage actuels : complexité des scripts, fréquence, intégration.
* Compétences des équipes internes et besoins de formation.
* Présentation des langages candidats (Python, Ruby, Shell).
* Évaluation des avantages/inconvénients.
* Choix collaboratif du langage cible.
* Validation par les responsables techniques et métiers.
* Communication officielle du choix à toutes les parties prenantes.

|  |  |  |
| --- | --- | --- |
| Langage | Avantages | Inconvénients |
| Shell | - Léger, natif Linux - Simple pour petites tâches - Pas de dépendances | - Peu structuré, difficile à maintenir - Limité pour tâches complexes/API - Moins sécurisé |
| Python | - Simple, lisible, large communauté - Bibliothèques riches pour l’automatisation - Compatible Zabbix (API, scripts) - Portabilité multi-plateforme | - Gestion des dépendances  - Formation nécessaire |
| Ruby | - Syntaxe élégante, objet - Bon support automatisation (ex: Puppet) - Bibliothèques réseaux/système | - Moins courant dans la supervision - Moins de ressources - Formation nécessaire |

**Phase 1 : Préparation et Analyse**

**1.1 Inventaire des scripts existants**

* Collecter **tous les scripts shell** actuellement utilisés dans Control-M.
* Identifier leur fréquence, dépendances, criticité, et propriétaires.
* Catégoriser par priorité métier (critique, important, etc. – P1-P4).

**1.2 Analyse technique**

* Évaluer qualité du code, gestion d’erreurs, sécurité.
* Identifier patterns communs et dépendances externes.

**1.3 Normes de codage Python**

* Élaboration d’un **framework de scripting Python** standard :
* Structure de script (main, fonctions, logging, gestion d’erreurs).
* Utilisation de **bibliothèques internes** (ex: subprocess, logging, os, argparse) et externes (si approuvé).
* Gestion des **fichiers de configuration** (YAML, JSON, etc.).
* Standards de **journalisation** compatibles avec Zabbix (format lisible, niveaux d’alerte).

# Phase 2 : Conception du Plan de Migration

### 2.1 Cartographie Control-M → Zabbix

* Identifier tous les **types de jobs** Control-M existants (batch, répétitifs, dépendants).
* Définir l'équivalence de chaque job dans Zabbix :
  + Scripts → **items personnalisés** ou **external scripts**.
  + Contrôles réguliers → **triggers programmés**.
  + Dépendances → **macros Zabbix** ou **conditions avancées**.
* Répertorier les **interactions spécifiques** : accès BDD, transferts, alertes.

### 2.2 Conception des Templates Zabbix

* Création de **templates types** pour les familles de jobs (ex: sauvegardes, transferts).
* Définition des **paramètres standards** (seuils, fréquence, timeout).

### 2.3 • Préparation de la documentation centrale du projet.

**Livrables**

* Inventaire complet et priorisé.
* Normes Python et framework.
* Scripts réécrits documentés.
* Templates Zabbix.
* Plans de tests et recettes.
* Guide d’exploitation post-migration.