

Onderhoudbare detectie- en vergelijkingsketen voor in-house software in EPC en WSIT/CMDB

Samevattingdocument

Quinten De Meyer

Student Bachelor in de Elektronica-ICT — Cloud & Cyber Security

Stagebedrijf: Pfizer Puurs — ICS Infrastructure

Stagecoach Thomas More: Cas Magnus

Stagementor Pfizer: Nikki Lissens

Inhoudsopgave

1. Context en Probleemstelling.....	3
2. De Gerealiseerde Oplossing: Architectuur en Workflow	3
<i>Fase 1: Bronbestandbeheer.....</i>	<i>3</i>
<i>Fase 2: Detectie en Dataverzameling</i>	<i>4</i>
<i>Fase 3: Vergelijking en CMDB-update</i>	<i>4</i>
<i>Overkoepelende Gebruikerslaag</i>	<i>4</i>
3. Inhoudelijke Keuzes, beperkingen en reflectie.....	5
4. Persoonlijke Reflectie en Conclusie	5
Bijlagen	7
Bijlage A — Begrippenlijst.....	7
Bijlage B — Diagram Figuur 1 <i>Architectuurdiagram van de EPC-CMDB Tool, opgedeeld in drie fasen: bronbestandbeheer, detectie en dataverzameling, en vergelijking en CMDB-update</i>	<i>9</i>

1. Context en Probleemstelling

Dit stageproject werd uitgevoerd bij Pfizer Puurs, een van de grootste productie- en verpakkingsfaciliteiten ter wereld. Binnen deze complexe IT-omgeving is een correcte en actuele software-inventarisatie een randvoorwaarde voor betrouwbaar infrastructuurbeheer, security-audits en licentiebeheer.

Het probleem situeerde zich bij de standaardinventarisatie via Endpoint Central (EPC). EPC leunt op Windows Installer-transacties (MSI) en herkent daardoor specifiek intern ontwikkelde of niet-standaardapplicaties (in-house software) vaak niet. Hierdoor ontstond een blinde vlek op de endpoints en groeiden er discrepanties tussen de werkelijke situatie en de administratieve registratie in de Configuration Management Database (CMDB) binnen de Workstation and Software Inventory Tool (WSIT, via Jira Assets). Er was nood aan een reproduceerbare, onderhoudbare methode om in-house software te detecteren en de CMDB automatisch bij te werken.

2. De Gerealiseerde Oplossing: Architectuur en Workflow

Om dit probleem op te lossen, werd de EPC-CMDB Tool ontwikkeld: een aaneengeschaalde, modulaire toolketen. De workflow is opgedeeld in drie logische fasen aangetoond in het diagram in bijlage B :

Fase 1: Bronbestandbeheer

De kern van de oplossing is een centraal YAML-bronbestand (Single Source of Truth) waarin alle softwarekenmerken (paden, executables, versies, services) zijn vastgelegd.

- **Software Source Maintainer:** Omdat YAML foutgevoelig is om handmatig te bewerken, is een Windows Presentation Foundation (WPF)-desktopapplicatie (C#) ontwikkeld waarmee beheerders de softwaredefinities via een grafische interface kunnen onderhouden.

- **Detection Profile Generator:** Een Python-script vertaalt de YAML-data automatisch naar gestructureerde, platte JSON-detectieprofielen die door de endpoints gelezen kunnen worden.

Fase 2: Detectie en Dataverzameling

- **Detection Engine:** Een geïsoleerd Python-script dat via Endpoint Central op duizenden endpoints wordt uitgevoerd. De engine leest de JSON-profielen in en zoekt deterministisch (read-only) naar de gedefinieerde software-indicatoren. Door optimalisaties duurt een scan slechts 4 tot 18 seconden per endpoint met minimale systeembelasting.
- **Result Merger / Normalizer:** Consolideert de duizenden individuele asynchrone endpoint-resultaten tot één genormaliseerd JSON-overzicht.
- **WSIT Assets Export Script:** Een PowerShell-script dat via REST API de actuele CMDB-registraties exporteert ter voorbereiding op de vergelijking.

Fase 3: Vergelijking en CMDB-update

- **Comparison Engine:** Vergelijkt de genormaliseerde endpoint-realiteit met de CMDB-export. De tool genereert een menselijk leesbaar HTML-discrepantierapport en een CSV-importbestand. Software buiten scope wordt hierbij veilig genegeerd om dataverlies in de CMDB te voorkomen.
- **WSIT API Import:** Een REST API-koppeling die de berekende doelstaat (inclusief *dry-run* simulatie) wegschrijft naar Jira Assets.

Overkoepelende Gebruikerslaag

Omdat bruikbaarheid en overdraagbaarheid topprioriteiten waren, is een centraal Launcher-menu (PowerShell) ontwikkeld. Vanuit deze interface kunnen beheerders de volledige workflow stap voor stap uitvoeren, output direct inzien en prerequisites (.NET, Python) automatisch laten installeren.

3. Inhoudelijke Keuzes, beperkingen en reflectie

De oplossing levert directe meerwaarde: er is nu een betrouwbaar, reproduceerbaar systeem om de blinde vlek rond in-house software weg te werken. Tijdens de realisatie zijn bewuste architecturale en procedurele keuzes gemaakt:

- **Scope-afbakening (Gedecommissioneerde software):** Er is expliciet gekozen om enkel *actieve, vooraf gemodelleerde* software te detecteren. Het in de gaten houden van historische, gedecommissioneerde software zou het bronbestand exponentieel vergroten en de onderhoudslast disproportioneel maken. Binnen Pfizer is het formele ticket- en deïnstallatieproces de primaire controle voor uitfasering, niet deze detectietool.
- **Scheiding Ontwikkeling en Productie:** Python-tools worden via een buildscript gecompileerd naar uitvoerbare bestanden (.exe) voor de release-map. Dit is een pragmatische *release management* keuze: zo zien toekomstige beheerders duidelijk het onderscheid tussen broncode (_dev) en de operationele tool, wat menselijke fouten voorkomt.
- **Infrastructuurlimitaties:** 32-bit endpoints en specifieke Windows 7 Enterprise-systemen vallen momenteel buiten scope door uitvoeringsrestricties vanuit Endpoint Central. Gezien de succesvolle scan op duizenden moderne Windows 10-machines is de impact hiervan op de globale bruikbaarheid echter minimaal.

4. Persoonlijke Reflectie en Conclusie

Dit project was een intensieve en uiterst leerrijke ervaring. Op technisch vlak heb ik mijn kennis verbreed met technologieën zoals Python, PowerShell 7, C#/WPF, JSON, YAML en de Jira REST API. Vooral het programmatisch aansturen van enterprise-applicaties via API's en het werken met dynamische groepen in Endpoint Central waren nieuwe en waardevolle competenties.

De belangrijkste leercurve bevond zich echter op het vlak van software architectuur en pragmatisme. Waar schoolprojecten vaak focussen op pure functionaliteit, lag de uitdaging hier bij het bouwen van een *onderhoudbare* en *overdraagbare* tool. Problemen waren vaak niet puur technisch, maar vereisten compromissen met bestaande enterprise-beperkingen. Door het bouwen van de WPF-maintainer en de PowerShell launcher leerde ik dat het succes van een interne tool grotendeels afhangt van het gebruiksgemak. Code die technisch werkt maar lastig te bedienen is, wordt in de praktijk niet gebruikt.

De EPC-CMDB Tool beantwoordt met succes de initiële onderzoeksvraag. Het bewijst dat het mogelijk is om binnen een sterk gereguleerde enterprise-omgeving een stabiele, reproduceerbare workflow te implementeren die de kloof tussen endpoint-realiteit en CMDB-administratie effectief dicht.

Bijlagen

Bijlage A — Begrippenlijst

AQL: *Assets Query Language*; querytaal binnen Jira Assets waarmee objecten gefilterd en opgezocht kunnen worden.

CD-shares: gedeelde netwerklocaties waarop installatiebestanden van software beschikbaar worden gesteld, doorgaans aangeduid met een cd-nummer in het formaat CDxxxx.

CMDB: *Configuration Management Database*; de asset-database die binnen WSIT wordt beheerd via Jira Assets. Hierin worden software en endpoints als objecten beheerd in een objectschema.

Discrepantie: verschil tussen de software-endpointrelaties in de CMDB en de feitelijke softwaretoestand op een endpoint.

Dynamic Custom Group: endpointgroep in Endpoint Central waarvan het lidmaatschap automatisch wordt bepaald op basis van filtercriteria zoals besturingssysteem of service pack.

Endpointrealiteit: de feitelijke toestand van software-installaties op een endpoint, vastgesteld door de detectie-engine op basis van JSON-detectieprofielen. Dit begrip wordt gebruikt als tegenhanger van de administratieve registratie in de CMDB.

EPC: Endpoint Central; platform voor endpointbeheer, scriptuitvoering op endpoints en aanvullende beheerfunctionaliteiten.

IID: *Installation Instruction Document*; documentatie waarin de installatie-informatie en -procedure van in-house software worden beschreven, inclusief een deïnstallatieprocedure.

In-house software: intern ontwikkelde en niet-standaardsoftware die niet betrouwbaar door de standaardinventarisatie van EPC wordt gedetecteerd.

JSON-detectieprofielen: gegenereerde artefacten op basis van het YAML-bronbestand, die door de detectie-engine worden gebruikt voor endpointcontrole.

Object key: unieke identificatiesleutel van een object in Jira Assets, gebruikt om endpoints en software eenduidig te koppelen bij import en export.

PowerShell SecretStore: module voor PowerShell waarmee gevoelige informatie, zoals authenticatietokens, lokaal en versleuteld kan worden opgeslagen.

PyInstaller: open-source hulpmiddel waarmee Python-scripts worden gecompileerd naar zelfstandige uitvoerbare bestanden die zonder Python-installatie kunnen worden uitgevoerd.

REST API: *Representational State Transfer Application Programming Interface*; interface waarmee registraties uit Jira Assets programmatisch kunnen worden opgehaald.

WSIT: *Workstation and Software Inventory Tool*; de overkoepelende Jira Service Management-omgeving van Pfizer, waarbinnen onder meer de CMDB via Jira Assets wordt beheerd.

YAML-bronbestand: centraal bronbestand waarin in-house software gestructureerd wordt beschreven, inclusief installatiepaden, bronmedia, versies en detectiekenmerken.

Bijlage B — Diagram

Figuur 1 Architectuurdiagram van de EPC-CMDB Tool, opgedeeld in drie fasen: bronbestandbeheer, detectie en dataverzameling, en vergelijking en CMDB-update

