Entwicklung Python Script zum Prüfen und Korrigieren von RDF Files

Leistungsnachweis 2

von

Andrés Baumeler

Telefon: 076 443 04 71, E-Mail: andres@baumeler.dev

Alte Riedikerstrasse 5c, 8610 Uster

Betreuer

Ismail Prada

Universität Zürich, Philosophische Fakultät /
Zentralbibliothek Zürich

CAS Datenmanagement und Informationstechnologien

Inhaltsverzeichnis

Kapitel 1

Einleitung

Banken sind gesetzlich verpflichtet gewisse Dokumente aus ihren Geschäftstätigkeiten aufzubewahren. Um die aufzubewahrenden Dokumente zentral zu verwalten werden digitale Archivsysteme eingesetzt. Ein solches System ermöglicht es alle in einer Bank produzierten Dokumente im Überblick zu behalten und den Lebenszyklus der Dokumente zu verwalten. Das Archivsystem ist in der Lage Dokumente und Metadaten aus einer Vielzahl an unterschiedlichen Quellen entgegenzunehmen. Während die akzeptierten Formate zwar klar definiert sind, ergeben sich beim Betrieb eines solchen Archivsystems immer wieder Herausforderungen mit Dokumenten welche die Anforderungen an den Archivierungsprozess nicht erfüllen. In dieser Arbeit wurde ein Python Script entwickelt, welches den Betreiber einer Archivlösung dabei unterstützen kann solche Dokumente zu prüfen.

Konkret behandelt das in dieser Arbeit entwickelte Python Script Files welche Metadaten im Ressource Description Format (RDF) enthalten. Die Files sind in der eXtensible Markup Language (XML) strukturiert. In dieser Arbeit wird nur das Archivsystem Hyper Suite 5 (HS5) der Firma IMTF[1] betrachtet.

[1] https://imtf.com/

Kapitel 2

Hauptteil

2.1 Hintergrund

In einer Bank gibt es in der Regel eine Vielzahl an Systemen welche Aufbewahrungspflichtige Dokumente produzieren. Die Aufbewahrung der Dokumente wird aber zentral in einem Archivsystem durchgeführt. Dadurch ergibt sich der Bedarf für Schnittstellen von produzierenden Systemen zum Archivsystem. HS5 bietet dazu die Möglichkeit Dokumente über eine Filebasierte Schnittstelle als RDF und PDF Paar entgegenzunehmen. Das PDF enthält das eigentliche Dokument während das RDF File die Metadaten zum Dokument enthält. Der Import dieser File Paare erfolgt aus einem überwachten Ordner auf dem Filesystem des Archivservers. Sobald dort ein File Paar abgelegt wird, prüft das Archivsystem die Dokumente. Erfüllt das angelieferte File Paar die Anforderungen für die Archivierung wird das PDF archiviert. Bei der Archivierung werden die benötigten Metadaten aus dem RDF gelesen und in der Archiv Datenbank gespeichert. Das PDF wird auf eine Write Once Read Many (WORM) Storage geschrieben, um sicherzustellen, dass das Dokument selbst nicht mehr verändert werden kann. Das originale RDF File wird einige Tage nach erfolgreicher Archivierung des PDFs gelöscht.

Da die Dokumente auf eine WORM Storage Lösung geschrieben werden, ist ein Löschen bis zum Ablauf der Aufbewahrungsfrist nicht möglich. Aus diesem Grund müssen die gelieferten Metadaten nicht nur syntaktisch,

sondern auch inhaltlich korrekt sein. Das Archivsystem führt aus diesem Grund eine Datenbank mit Stammdaten der Bank. Darin enthalten sind etwa Kunden- und Kontonummern. Vor der Archivierung werden die im RDF angelieferten Dokumente gegen diese Datenbank geprüft. Wird beispielsweise eine Kontonummer in den Metadaten angegeben, welche dem Archiv nicht bekannt ist, wird das Dokument nicht archiviert. Für Dokumente, welche die Anforderung an die Archivierung nicht erfüllen, wird das RDF und PDF Paar zusammen mit einem kurzen Fehlerbeschrieb in ein Verzeichnis geschrieben. Dieses sogenannte failed Verzeichnis wird überwacht und die dort abgelegten Dokumente regelmässig geprüft.

2.2 Problemstellung

Die Prüfung von Dokumenten im failed Verzeichnis erfolgt von Hand und nimmt pro Dokument einiges an Zeit in Anspruch. Die Dokumente müssen auf der Commandline betrachtet werden da keine Benutzeroberfläche verfügbar ist. Danach müssen Werte in der Archivdatenbank geprüft werden. Dies kann entweder via Commndline oder via einem externen Tool gemacht werden.

Wenn ein Dokument im failed Verzeichnis liegt, kann es dafür mehrere Gründe geben. Die im täglichen Betrieb an häufigsten auftretenden Probleme sind die folgenden:

- Metadaten sind nicht vollständig (z.B. keine Kontonummer angegeben)
- Metadaten sind im falschen Format (z.B. Kontonummer hat zu wenig Stellen)
- Metadaten sind dem Archiv nicht bekannt (z.B. Kontonummer ist nicht in Archiv Datenbank)

Nicht vollständige Metadaten bzw. Metadaten im falschen Format können nicht automatisch geprüft werden und müssen weiterhin manuell abgeklärt werden. Grund hierfür ist, dass die Verantwortlichkeit über die Metadaten

beim anliefernden System liegt. Das Script kann hier aber unterstützen, indem File Paare mit unvollständigen oder falsch formatierten Metadaten in einen anderen Ordner verschoben werden.

Der dritte Punkt kann automatisiert werden. Aufgrund der asynchronen Verarbeitung der Dokumente, kommt es immer wieder vor, dass Dokumente geliefert werden, bevor die Stammdaten dafür im Archiv sind. Wurde verifiziert, dass die Stammdaten im Archiv sind, kann der Archivierungsprozess für das Dokument nochmals gestartet werden. Die Prüfung ob Metadaten mittlerweile in der Archivdatenbank bekannt sind, war der Hauptpunkt, welcher durch das Script gelöst werden sollte.

2.3 Lösungsansatz

Als Lösung für das beschriebene Problem sollte ein Script entwickelt werden, welches die im RDF File gelieferten Metadaten gegen die Archivdatenbank prüft. Sind alle Daten dem Archiv bekannt, soll das Script die File Paare auch gleich in den korrekten Input Ordner verschieben können. Als Unterstützung sollte das Script auch anzeigen, wenn die Metadatenfiles im falschen Format sind (kein gültiges XML) oder die Metadaten im falschen Format sind. Diese Files sollen automatisch in einen anderen Ordner verschoben werden. Zur Umsetzung wurde Python gewählt. Grund für die Wahl von Python ist, dass eine entsprechende Laufzeitumgebung auf dem Zielsystem bereits vorhanden ist und die Sprache für den Umgang mit Dokumenten im XML Format sowie bei der Darstellung gegenüber Alternativen wie Shell Script einige Annehmlichkeiten bietet. Ziel war es ein Script zu entwickeln welches die folgenden Anforderungen erfüllt:

1. Automatisches Prüfen von Metadaten gegen Archiv Datenbank
2. Easy of Use: Als terminal UI Applikation oder via Commandline
3. Erweiterbarkeit und Wartbarkeit

2.4 Ergebnis

Das Ergebnis besteht aus einem Python Script welches über ein Terminal User Interface (TUI) verfügt. Dadurch wird das Script um einiges intuitiver bedienbar. Zudem können die fehlerhaften Metadatenfiles gleich neben den möglichen Aktionen angezeigt werden. Ein Arbeiten mit mehreren SSH Sessions entfällt damit. Durch das TUI können für die Bedienung notwendige Informationen in einem ansprechbaren Format angezeigt werden, was die Benutzung für unerfahrene Benutzer vereinfacht. Gleichzeitig kann das Script auch als normales Shell Script verwendet werden. Dies ermöglicht es das Script in andere Script oder Automationslösungen einzubauen.

2.5 Dokumentation

Die aktuellste technische Dokumentation sowie eine Anleitung zum Aufsetzen einer Referenzumgebung ist in den Files *start.md* sowie *README.md* enthalten.

Der Source Code für das Script ist auf GitHub verfügbar. Das Script benötigt mindestens Python 3.11 sowie die im requirements.txt festgehaltenen Pakete.

Kapitel 3

Schluss

3.1 Offene Punkte & Ausblick

Es gibt diverse Punkte, welche noch verbessert werden könnten oder Features welche noch nützlich sein könnten. Ein Verbesserungspunkt wäre das Log in ein File zu schreiben, damit auch bei einer automatisierten Verarbeitung im Nachhinein festgestellt werden kann was genau gemacht wurde. Weiter wäre es wohl Sinnvoll automatisierte Unit-Tests einzubauen, um die Weiterentwicklung zu vereinfachen. Sollte die Codebasis noch weiter wachsen muss gegebenenfalls auch darüber nachgedacht werden den Code auf mehrere Files zu verteilen, um die Lesbarkeit zu erhöhen.

3.2 Fazit und Reflexion

Meine wichtigsten Anforderungen an das Script konnte ich realisieren. Ich habe aber noch viele Ideen für weitere Features welche noch eingebaut werden könnten.

Das Projekt hat mir die Gelegenheit gegeben meine Python Kenntnisse zu vertiefen sowie etwas Erfahrungen mit dem Bereitstellen von Entwicklungsinfrastruktur mittels Docker und Docker-Compose zu sammeln. Im Bereich Python konnte ich gerade in den Bereichen Filehandling, asynchrone Verarbeitung und TUI viel dazulernen. Aus dem Grund, dass ich das Projekt

losgelöst vom täglichen Betrieb umsetzen konnte, wahr ich in der Lage meine Qualitätsansprüche zu erhöhen und Features einzubauen, welche ich nicht eingebaut hätte, wenn ich das Script auf Arbeitszeit entwickelt hätte. Ein TUI bringt zwar viele Vorteile und Annehmlichkeiten aber doch auch einiges an Komplexität in das Script. Dazu hätte mir im Betrieb die Zeit gefehlt und ich hätte das Script wohl als reines Shell Script umgesetzt.

Abkürzungen

HS5	Hyper Suite 5
RDF	Ressource Description Format
WORM	Write Once Read Many
XML	eXtensible Markup Language
TUI	Terminal User Interface

Literaturverzeichnis