Università di Genova

Corso di laurea triennale in Informatica

Monitoraggio e recupero automatizzato di un layer d'integrazione in cloud

Candidato: Capiaghi Ludovico

Relatore: Ancona Davide

Anno Accademico: 2022/2023

1. Introduzione

1.1. Preambolo

Oggigiorno la migrazione in cloud è un processo che si afferma sempre in modo più evidente nel panorama aziendale, in quanto consente alle organizzazioni di spostare le proprie risorse informatiche, tra cui applicazioni, dati e infrastrutture, in un ambiente virtuale. Questo passaggio offre una serie di vantaggi significativi, tra cui la flessibilità operativa, l'accesso remoto ai dati, l'elasticità delle risorse e la riduzione dei costi. D'altro canto, questo processo è caratterizzato da numerose sfide tecnologiche che spaziano dall'organizzazione più ad alto livello delle risorse e degli applicativi fino alla ricerca di soluzioni implementative necessarie per integrare applicativi eterogenei.

Negli ultimi quattro mesi ho avuto la possibilità di svolgere il mio tirocinio curricolare presso l'azienda Axpo. Sono stato inserito nel team che si sta occupando del processo di migrazione in cloud. Il team si occupa dell'integrazione, dello sviluppo, del monitoraggio e del test degli applicativi migrati e in corso di migrazione.

1.2. Axpo

Axpo è una multinazionale svizzera che opera nel settore energetico. La sua attività principale riguarda la produzione, la commercializzazione e la distribuzione di energia elettrica, gas naturale e servizi energetici.

Nella sede di Genova, sono stato inserito in un ambiente lavorativo che media tra il grado di "grande azienda" e " piccola realtà", questo perché in Axpo è presente un'organizzazione tipica di una grossa realtà aziendale, con una struttura ramificata che occupa centinaia di persone in decine di team con compiti diversificati. Allo stesso tempo, però, si ha la possibilità di conoscere di persona la maggior parte dei colleghi nel proprio reparto e sentirsi parte di un gruppo che lavora all'unisono, pur lavorando in ambienti e su task in alcuni casi molto diversi. In particolare, nel mio team ho sempre trovato i miei colleghi disponibili ad aiutarmi ma allo stesso tempo sempre aperti ad ascoltare il mio parere.

Concludo questa breve parentesi sull'ambiente lavorativo, in cui ho avuto il piacere di svolgere la mia esperienza di tirocinio, poiché ritengo opportuno esplicitare che questo clima è stato fondamentale per raggiungere i risultati che discuterò in seguito.

1.3. Integrazione e Monitoraggio

Come accennato in precedenza, il team in cui sono stato inserito si occupa principalmente d'integrazione e monitoraggio.

Con integrazione si intende il voler garantire una connessione e uno scambio fluido di dati e informazioni tra le diverse componenti del sistema IT. Infatti, il focus principale della migrazione in cloud sono gli applicativi che costituiscono il layer d'integrazione. In particolare, la migrazione consiste nel passaggio da una soluzione on-premise ad una soluzione basata su tecnologia Mulesoft. Il monitoraggio consiste nel raccogliere dati e informazioni in tempo reale su risorse, applicazioni, reti e dispositivi utilizzati all'interno di un'infrastruttura tecnologica.

Tra le attività svolte dal team, il monitoraggio riveste un ruolo parallelo e complementare a quello dell'integrazione. L'attività d'integrazione, infatti, consiste nel progettare, produrre e testare applicativi che vengono migrati in cloud, mentre il monitoraggio consente di tracciare le esecuzioni ed è complementare alla fase di test. Allo stesso tempo, esso permette di individuare eventuali malfunzionamenti, errori che possano causare interruzioni del servizio o degradazioni delle prestazioni. Queste informazioni vengono utilizzate, in primo luogo, per intervenire rapidamente con il fine di ripristinare l'integrità del sistema e limitare gli impatti negativi sulle operazioni aziendali. Inoltre, esse possono essere utilizzate per ottimizzare l'utilizzo delle risorse, migliorare le prestazioni delle applicazioni e prendere decisioni informate per l'allocazione delle risorse.

1.4. Obiettivi del tirocinio

Il lavoro svolto durante la mia esperienza ad Axpo copre aspetti di entrambe le attività e, in particolare, si concentra sulle attività di *monitoraggio e recupero automatizzato.* Le attività di cui mi sono occupato sono:

- la realizzazione di dashboard di supporto all'attività di monitoraggio.
- lo sviluppo di un applicativo per l'automatizzazione di un'attività legata al monitoraggio di applicativi migrati in cloud

1.5. Premesse

Sono necessarie alcune premesse sulla terminologia pratica che verrà utilizzata durante la descrizione delle attività svolte:

- in Axpo, la gestione del ciclo di vita del software viene organizzata in diversi *ambienti di sviluppo*:
    - pre-produttivi `SIT`, `DEV`, `BUGFIX`, `UAT`: in cui vengono eseguite tutte le fasi di sviluppo e test:
    - produzione `PROD`: in cui le versioni ufficiali vengono rilasciate e sono quelle utilizzate dai clienti.
- con `stream` o `flusso` si intende un insieme di applicativi collegati tra di loro per svolgere un insieme di task tra di loro correlate. Per esempio, lo stream `ifsoutbound` si occupa di tutte le operazioni in uscita verso il servizio esterno `IFS`. Un applicativo in generale può implementare logiche di più stream.
- con `verticale funzionale`, si intende un insieme di flussi che hanno in comune la finalità delle operazioni. Per esempio, la verticale funzione `IFS` è composta dai flussi `ifsoutbound` e `ifsinbound`.

2. Tecnologie e strumenti utilizzati

2.1. ELK

ELK, ovvero lo stack ELK composto da Elasticsearch, Logstash e Kibana, è un insieme di strumenti open-source ampiamente utilizzati per la gestione dei dati di log e per il monitoraggio nel campo IT.

2.1.1. Lo stack ELK

Elasticsearch è il componente principale dello stack ELK ed è un potente motore di ricerca distribuito. È progettato per archiviare grandi quantità di dati strutturati o non strutturati, inclusi i dati di log, e offrire una rapida indicizzazione e ricerca full-text. Elasticsearch permette di eseguire ricerche complesse e di ottenere risultati in tempo reale, consentendo di individuare velocemente problemi, anomalie o tendenze nei log.

Logstash è una piattaforma di raccolta, trasformazione e invio di dati di log. Logstash consente di raccogliere i log da diverse fonti come file di log, eventi di rete, database e altri servizi, e di trasformarli e normalizzarli in un formato coerente. Inoltre, Logstash può arricchire i dati di log con metadati aggiuntivi e inviarli ad Elasticsearch per l'indicizzazione e la ricerca.

Kibana è un'interfaccia di visualizzazione dei dati che consente di esplorare e analizzare i dati di log archiviati in Elasticsearch. Kibana offre un'ampia gamma di strumenti di visualizzazione come grafici, tabelle, mappe e dashboard personalizzabili, che permettono di rappresentare i dati di log in modo chiaro e intuitivo. Risulta, quindi, utile utilizzare Kibana per monitorare le metriche chiave, identificare pattern, eseguire analisi approfondite e creare report dettagliati.

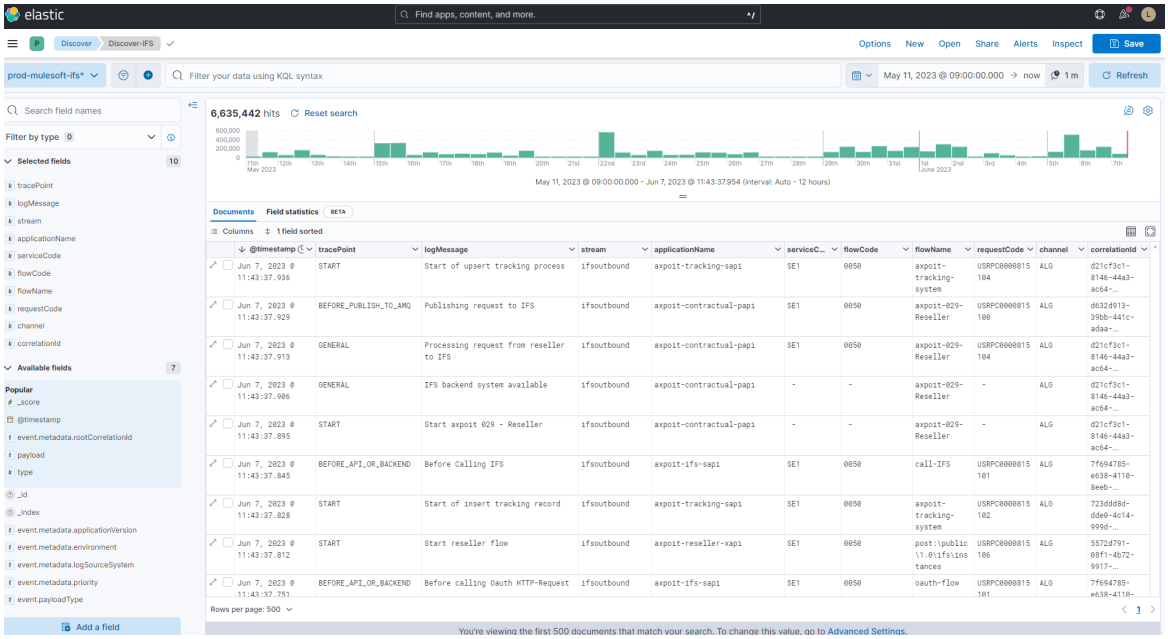
2.1.2. ELK in Axpo

In Axpo, lo stack ELK viene utilizzato attraverso la soluzione Elastic Cloud. Elastic Cloud è una soluzione basata sul cloud che offre l'infrastruttura necessaria per utilizzare lo stack ELK senza dover gestire l'installazione e la configurazione dei componenti su server locali. Invece di dover implementare e gestire autonomamente l'ambiente ELK su server dedicati, *Elastic Cloud* offre un servizio gestito che fornisce l'accesso a un'istanza preconfigurata e scalabile di Elasticsearch, Logstash e Kibana.

La componente Logstash, in Axpo, è configurata in modo da ricevere in input i log dai diversi sistemi e reindirizzare la loro "scrittura" su indici di Elasticsearch opportuni, identificati attraverso un pattern `ambiente-environment-stream`. Un esempio può essere `prod-mulesoft-ifsoutbound` , dove vengono reindirizzati i log degli applicativi Mulesoft in ambiente di produzione che costituiscono lo

stream `ifsoutbound`.

Gli indici di Elasticsearch sono l'elemento base su cui vengono "scritti" i log . A partire da questi si possono costruire strutture più ad alto livello, utilizzando Kibana. Tra queste le più importanti sono le *data view*, queste permettono di svolgere le ricerche e visualizzazioni su più indici. Le data view sono gli oggetti su cui si basa ***Discover***.



Discover, un esempio nell'immagine qui sopra riportata, offre un'interfaccia utente intuitiva che consente di eseguire ricerche complesse in modo rapido e intuitivo. Attraverso Discover si visualizzano le *data view*, ovvero, i dati contenuti negli indice ad esse associati. Si possono effettuare ricerche testuali e visualizzare i risultati in modo interattivo, applicando aggregazioni attraverso il linguaggio d'interrogazione KQL. L'interfaccia di Discover mostra una panoramica dei dati sotto forma di tabella, in cui ogni riga rappresenta un singolo documento. Si possono esplorare le colonne per ottenere informazioni dettagliate sui dati e filtrare i risultati in base ai loro criteri.

A partire dalle *data view* è possibile, attraverso gli strumenti ***Dashboard e Lens*** di Kibana, realizzare pannelli interattivi e personalizzati per visualizzare e analizzare i dati in modo chiaro e significativo. Tra le funzionalità più utili di Discover vi è sicuramente quella dei *filtri*. Attraverso i filtri è possibile memorizzare ricerche, queste possono essere interrogazioni KQL ma anche ricerche più complesse, per esempio, utilizzando espressioni regolari. Un filtro può essere applicato a Discover e questa operazione permette di mantenere la query del filtro attiva di default.

2.1.3. Evoluzione del monitoraggio

Il processo di migrazione è un processo dedicato a ristrutturare le precedenti soluzioni verso una direzione di maggiore scalabilità e più facile gestione. Sotto questo aspetto è, infatti, in corso anche una ristrutturazione delle modalità di gestione dei log dei sistemi migrati, con l'intento di centralizzarli sullo stack ELK e allo stesso tempo uniformarli. Questo poiché ELK rappresenta una risposta moderna a problemi di scalabilità e organizzazione di grandi quantità di dati, specialmente quando si ci occupa di log.

La migrazione è, dunque, anche un processo di rinnovazione e miglioramento del monitoraggio in generale. Nel monitoraggio oggigiorno si è orientati verso soluzioni che puntino alla scalabilità orizzontale e che possano fornire prestazioni elevate per le ricerche e le aggregazione dei dati con lo scopo di raggiungere una sempre più dettagliata analisi dei sistemi in tempo reale.

Alla base di questo processo è di fondamentale importanza il processo di ristrutturazione dei log con lo scopo di renderli quanto più omogenei e completi possibile. Non è possibile, infatti, costruire analisi efficaci e avanzate sull'andamento dei sistemi se i log provenienti da essi non presentano una struttura organizzata e uniforme.

2.2. Mulesoft

Entrando nei dettagli del processo di migrazione in cloud, la tecnologia utilizzata per la migrazione del layer d'integrazione all'interno di Axpo è Mulesoft. MuleSoft è una piattaforma d'integrazione che consente di connettere sistemi, applicazioni e dati eterogenei utilizzando un'organizzazione centralizzata. La piattaforma utilizza un approccio basato su API e offre strumenti per semplificare l'integrazione, la gestione e il monitoraggio delle risorse aziendali.

La piattaforma supporta sia l'integrazione on-premise che nel cloud, offrendo la flessibilità necessaria per connettere applicazioni e servizi distribuiti su diverse piattaforme. MuleSoft fornisce un ambiente di sviluppo intuitivo e strumenti visivi per creare, testare e implementare le integrazioni. Inoltre, MuleSoft offre funzionalità di gestione delle API, consentendo alle organizzazioni di pubblicare, monitorare e gestire le proprie API in modo centralizzato.

2.2.1. Anypoint Platform

Nel corso dello sviluppo dell'applicativo Mulesoft, ma anche per comprendere le logiche degli applicativi su cui sviluppare le dashboard verranno, in particolare, utilizzati gli applicativi forniti dalle suite di Anypoint Platform.

Anypoint Platform è una piattaforma d'integrazione aziendale offerta da Mulesoft. Tra le componenti principali di Anypoint Platform, quelle che verranno utilizzate maggiormente sono:

- **Anypoint Studio**: è un ambiente di sviluppo basato su Eclipse che consente di creare rapidamente applicazioni d'integrazione, utilizzando un'interfaccia visuale intuitiva. Anypoint Studio supporta sia l'approccio di sviluppo a codice completo che il modello di sviluppo basato su componenti.
- **Anypoint Exchange**: è una libreria centralizzata che consente di accedere a connettori predefiniti, modelli d'integrazione, API, componenti riutilizzabili e altre risorse. Tramite questo strumento si possono condividere e scoprire asset, promuovendo il riutilizzo e l'efficienza.
- **Anypoint Design Center**: offre un ambiente visuale per la progettazione di API, flussi d'integrazione e logica di business. Attraverso esso si possono creare rapidamente API RESTful, definire i punti di connessione, definire le trasformazioni dei dati e orchestrare i flussi di lavoro d'integrazione.
- **Anypoint Runtime Manager**: è una piattaforma di gestione e monitoraggio delle applicazioni e che consente di distribuire, gestire e monitorare le applicazioni d'integrazione create con Anypoint Platform.

2.3. Postman

Postman è uno strumento di sviluppo API che consente di testare, documentare e collaborare sulle API. È ampiamente utilizzato per semplificare il processo di sviluppo e il test di applicazioni che utilizzano API. Consente di eseguire facilmente richieste alle API, verificare le risposte, automatizzare i test e collaborare con il team di sviluppo e di test.

Postman offrirà supporto per testare le configurazioni degli indici ELK durante il processo di remapping, eseguendo chiamate ad applicazioni in ambienti pre-produttivi e verificando la corretta "scrittura" dei log su ELK. Inoltre, verrà ampiamente utilizzato durante lo sviluppo dell'applicativo Mulesoft, per cui verrà sviluppata una suite di test per interfacciarsi con l' API creata.

2.4. Ciclo di vita del software

2.4.1. Azure DevOps

Azure DevOps è risultato sicuramente lo strumento centrale, se si considera lo sviluppo e la gestione del codice, con il ruolo di raccolta e organizzazione delle repository, della documentazione e della gestione delle task.

Tra i vantaggi DevOps, ho trovato fondamentale l'automazione. In particolare, grazie all'utilizzo di pipeline che combinano dati estratti automaticamente dal progetto a configurazioni pre-impostate è possibile automatizzare il deployment delle applicazioni in un ambiente Mulesoft.

2.4.2. Maven

Entrando ancora di più nei dettagli legati allo sviluppo, gli applicativi Mulesoft sono basati su Java. In particolare, i progetti delle applicazioni Mulesoft sono sviluppati come progetti Maven. Maven è uno strumento di gestione dei progetti software utilizzato principalmente per la compilazione, il packaging e la gestione delle dipendenze.

In Axpo, per le specifiche del `pom.xml`, file che definisce le configurazioni del progetto Maven, tutti i progetti ereditano le configurazioni di un `parent-pom.xml`. Questa configurazione permette in modo rapido di configurare il progetto con i componenti comuni usati dagli applicativi, per esempio il `logger`.

3. Dashboard ELK

Il primo progetto assegnatomi è stata la realizzazione di una serie di dashboard con lo scopo di riassumere in modo efficace lo stato di alcuni sistemi. In particolare, la task consisteva nel sviluppare una dashboard per l'insieme di flussi in entrata e in uscita da `IFS`.

`IFS` è un servizio terzo con cui gli applicativi di Axpo si interfacciano per le pratiche energetiche. Si tratta di una delle verticali funzionali più grandi tra quelle coinvolte nella migrazione. La migrazione della verticale sarebbe stata resa effettiva, ovvero, messa in produzione, nel corso del mese di Maggio. Le dashboard, quindi, sarebbero dovute essere pronte per il rilascio in produzione, in quel momento, infatti, si sarebbero rivelate utili per identificare più velocemente eventuali problematiche. Sul lungo periodo, invece, le dashboard avranno un ruolo di supporto all'attività di monitoraggio permettendo di acquisire una panoramica veloce della distribuzione degli errori nei sistemi monitorati.

Per sviluppare queste dashboard lo strumento centrale è stato ELK. Gli applicativi su cui ho lavorato hanno tutti in comune la caratteristica di utilizzare ELK per la gestione dei log. Non avendo competenze pregresse con lo strumento, mi sarei concentrato inizialmente su un flusso "meno complicato", `OTP`, in modo da familiarizzare con lo strumento.

Durante il processo di sviluppo dashboard ELK ho utilizzato assiduamente la *documentazione di Elasticsearch* [1] per quanto riguarda l'aspetto più tecnico dell'utilizzo dello stack ELK; nelle prossime sezioni seguiranno riferimenti più precisi ad essa.

3.1. Metodologia per lo sviluppo dashboard

Intendo ora presentare, sinteticamente, la metodologia di sviluppo dashboard utilizzata.

Il processo si articola nelle fasi di:

- analisi del dominio e comprensione delle necessità
- remapping indici Elasticsearch
- sviluppo dashboard

Il processo inizia con la fase di **analisi del dominio e comprensione delle necessità**. Solitamente il collega referente del flusso fornisce una spiegazione del flusso di carattere generale, sulle finalità e sull'architettura di esso. A seguito di questa prima introduzione, utilizzando lo strumento Discover di Kibana, segue una fase di analisi dei log dei vari applicativi del flusso per ritrovare e comprendere maggiori dettagli rispetto all'architettura e alle logiche interne di esso.

In questa fase si pone particolare attenzione sul seguire interamente delle serie di log correlati. Una sequenza di log correlati è caratterizzata, generalmente, dalla condivisione di un unico `correlationId`, ovvero, un identificativo alfa-numerico assegnato ad una richiesta quando viene inizializzata e poi propagato tra i vari applicativi e mantenuto costante fino alla terminazione di essa. Grazie a questa informazione è possibile ricostruire ed esplorare una richiesta dall'inizio alla fine attraverso le diverse applicazioni che compongono il flusso. Quest'analisi permette di identificare i log più significativi per una richiesta, ovvero, quelli che identificano il raggiungimento di un determinato stato del processo. Creando opportuni filtri è, dunque, possibile sviluppare una visualizzazione più chiara e minimale su Discover, su cui poi sviluppare le dashboard.

Inoltre, risulta determinante, a qualsiasi stadio del processo, il confronto con i colleghi direttamente interessati alla dashboard. Questo, poiché è fondamentale comprendere al meglio quali valori, intesi come numeriche o visualizzazioni, la dashboard dovrà riportare. A seguito della prima fase di analisi, si iniziano a delineare gli aspetti centrali della dashboard. Questo poiché entrano in campo quali errori vanno monitorati e di questi quali statistiche possono essere utili. Si inizia, inoltre, a comprendere quali visualizzazioni possono essere di supporto per riassumere lo stato generale del flusso.

Con un'idea più chiara del risultato finale desiderato si può iniziare la fase di **sviluppo dashboard**. Utilizzando gli strumenti *Dashboard e Lens* di Kibana, questa fase può essere svolta direttamente sui dati di produzione, dove è facile accorgersi se delle metriche o visualizzazioni pensate risultato essere significative.

Durante il mio primo sviluppo, però, mi sono imbattuto in problematiche rilevanti utilizzando lo strumento Lens. Esso permette di creare visualizzazioni più complesse in modo facile e intuitivo. Questo tipo di elementi può essere creato solo se gli indici di Elasticsearch, su cui si sta costruendo la visualizzazione, presentano determinate caratteristiche. Nel caso della situazione iniziale degli indici di Axpo, queste condizioni non risultavano soddisfatte per alcuni campi fondamentali per lo sviluppo delle visualizzazioni desiderate. Per questo, tra la fase iniziale di analisi e quella di sviluppo dashboard è sempre risultata necessaria una fase di **remapping indici Elasticsearch** in cui vengono effettuate modifiche sugli indici Elasticsearch per permettere lo sviluppo ottimale delle dashboard.

3.2. Remapping Indici Elasticsearch

Per comprendere meglio l'idea del remapping e della sua necessità è utile introdurre alcuni concetti di ELK più nel dettaglio.

Indici Elasticsearch

Gli indici in Elasticsearch sono strutture dati che organizzano i documenti in modo da consentire ricerche veloci e ad alto livello di performance. Un indice, in Elasticsearch, è simile a una tabella di database relazionale. Esso contiene una collezione di documenti JSON, dove ogni documento rappresenta un'entità o un record e ha una struttura flessibile. Gli indici Elasticsearch possono essere, quindi, visti come contenitori logici che raggruppano documenti simili.

Ogni campo in un documento Elasticsearch è indicizzato in modo da consentire ricerche efficienti. Gli indici utilizzano una struttura dati denominata ad *indice invertito* per creare un elenco di termini che puntano ai documenti corrispondenti. Ciò consente di eseguire ricerche full-text su testi, numeri e altri tipi di dati in modo molto rapido. Inoltre, gli indici Elasticsearch supportano anche *aggregazioni*, che consentono di calcolare statistiche e aggregazioni su grandi insiemi di dati.

Mapping

Il mapping di un indice Elasticsearch definisce la struttura e il tipo dei dati all'interno dei documenti JSON dell'indice, ovvero, specifica come i campi devono essere interpretati, indicizzati e trattati durante le operazioni di ricerca e aggregazione. In particolare, i campi possono essere aggregabili o non aggregabili. La differenza tra le due tipologie riguarda la possibilità di eseguire aggregazioni su quei campi. Alcuni campi, come i `text` o `date` sono di default non aggregabili, mentre altri come come gli `number` e le `keyword` sono di default aggregabili.

La possibilità di eseguire aggregazioni è condizione necessaria per lo sviluppo di visualizzazioni più complesse in Lens.

Inoltre, la distinzione tra campi aggregabili e non aggregabili è importante anche per le prestazioni e l'ottimizzazione delle query. I campi aggregabili richiedono una struttura di indicizzazione e memorizzazione dei dati specifica per supportare le aggregazioni, che può richiedere più spazio di archiviazione e risorse di calcolo. D'altra parte, i campi non aggregabili possono essere più efficienti per le ricerche se non è necessario eseguire aggregazioni su di essi.

Quando si crea un indice, Elasticsearch può generare automaticamente un mapping iniziale in base alla struttura dei primi documenti che vengono indicizzati, utilizzando, quindi, un *mapping dinamico*. Tuttavia, per avere un controllo più preciso sulla struttura dei dati e sulle opzioni di indicizzazione, è possibile definire un *mapping esplicito*.

Alias

Come accennato precedentemente, gli applicativi di un flusso "scrivono" i propri log su un indice specifico definito come `ambiente-environment-flusso` dove, per esempio `prod-mulesoft-ifsoutbound` indica:

- `prod`: per l'ambiente di produzione
- `mulesoft`: per i flussi i cui applicativi girano su Mulesoft
- `ifsoutbound`: il flusso `ifsoutbound`

In realtà, la "scrittura" è un processo più complesso, costituito da diverse fasi:

- l'applicativo invia il log, utilizzando un logger che prepara il log secondo uno standard interno di Axpo, ad un server esterno Logstash
- Logstash analizza i log ricevuti, ricompone il pattern `ambiente-environment-flusso`, su cui effettivamente scrivere il log, esegue un'elaborazione opportuna del log e infine la scrittura di esso sull'indice Elasticsearch

A questo punto è necessario aggiungere un'importante precisazione. L'identificativo `ambiente-environment-flusso`, che fino ad ora è stato descritto come indice in realtà è gestito, nella configurazione di Axpo, come un alias. Un alias è un identificativo che astrae uno o più indici Elasticsearch. In particolare, solo uno di questi indici può essere di scrittura ed è quello su cui i log in arrivo vengono veramente salvati, mentre gli altri indici possono rimanere associati come indici di lettura.

L'utilizzo di alias è utile, per esempio, quando si vuole limitare la dimensione massima di un indice per ottimizzare le prestazioni e per aumentare la scalabilità della soluzione. Quando questo raggiunge la dimensione massima si può procedere con la creazione di un nuovo indice e associarlo come indice di scrittura per il relativo alias. L'indice pieno può essere mantenuto come indice in lettura per l'alias.

3.2.1. Necessità del remapping

3.2.1.1. Lo stack ELK in Axpo: dettagli

Lo stato iniziale dello stack ELK consisteva e, tutt'ora in parte, consiste in:

- un component template: `axpo-mapping-component` che definisce un mapping. Il mapping è composto da una parte di *mapping esplicito*, che va a mappare tutti i campi dei log dati dallo standard di log all'interno di Axpo, ed una parte di *mapping dinamico* per mappare i campi specifici di un determinato flusso o verticale funzionale. Questa soluzione essendo molto generale permette grande riusabilità ed era, infatti, alla base di tutti gli `index-template`.

- una lifecycle policy: `axpo-ilm-policy` che definisce una dimensione massima per un indice e la procedura per crearne uno nuovo quando questo fosse pieno. In particolare, si segue la convenzione che un indice sia definito come `ambiente-environment-flusso-n` dove `n` è un numero sequenziale, inizializzato a `000001`, che viene incrementato alla creazione di un nuovo indice legato all'alias `ambiente-environment-flusso`.
- gli index template: per ogni flusso e quindi per ogni alias è definito un *index template* `ambiente-environment-flusso-template` . Un index template specifica le impostazioni di un indice ed è utile poiché permette di velocizzarne la creazione. Questo poiché la primitiva di creazione di un indice, se chiamata senza specificare le configurazioni, cerca in automatico un *index template* il cui `target`, parte della configurazione del template e valutato utilizzando espressioni regolari, corrisponda al nome dell'indice.

  Gli index template vengono anche utilizzati dalla *lifecycle policy* per automatizzare la creazione dei nuovi indici.

  Tutti i template, `ambiente-environment-flusso-template`, erano configurati per avere come target `ambiente-environment-flusso` ed erano poi formati dalla combinazione del `axpo-mapping-component` e del `axpo-ilm-policy`. Di conseguenza, tutti gli indici creati fino a quel momento condividevano i mapping definiti in `axpo-mapping-component`.

3.2.1.2. Problema

Come accennato in precedenza, per lo sviluppo di dashboard con componenti avanzati risulta fondamentale avere a disposizione i campi opportuni come tipi aggregabili. La configurazione del `axpo-mapping-component`, essendo più orientata verso l'efficienza, non tiene conto di queste esigenza. Infatti, il mapping definito al suo interno mappa la maggior parte dei campi, tra cui molti di quelli che poi si sono rivelati utili per lo sviluppo dashboard, come tipi `text` e di conseguenza non aggregabili di default.

Da questo nasce la necessità di modificare il mapping degli indici interessati, in particolare, modificare il tipo dei campi utili per lo sviluppo delle dashboard in modo da renderli aggregabili. Inoltre, Elasticsearch non fornisce primitive per effettuare la modifica in-place del mapping di un indice.

3.2.1.3. Soluzione

Per risolvere questo problema l'idea è quella di eseguire una sequenza di operazioni, ovvero, primitive offerte dal API Elasticsearch per ottenere le modifiche desiderate. La procedura deve garantire che al termine del `remapping` su un indice `index` associato ad un alias `alias` siano vere le seguenti condizioni:

1. l'indice risultato `index_result` dovrà contenere tutti i log di `index` ma con il nuovo mapping
2. durante la procedura se vengono scritti dati su `alias` questi non devono essere persi e vanno anch'essi scritti su `new_index` con il nuovo mapping
3. al termine della procedura una ricerca qualsiasi su `alias` deve ritornare solo risultati con il nuovo mapping

3.2.1.4. Idea Implementazione

La soluzione si articola in due fasi:

- la creazione di un *component template*, `new_mapping_component`, per mappare i campi importanti per la realizzazione della dashboard come campi aggregabili. Generalmente, si tratta di convertire questi campi da un mapping iniziale di tipo `text` ad uno finale di tipo `keyword`.
- l'esecuzione di una procedura che permetta di effettuare la migrazione dall'indice attuale, rispettando le proprietà desiderate, in un nuovo indice che utilizza `new_mapping_component`.

3.2.1.5. Component Template

In primo luogo, partendo dal `axpo-mapping-component` bisogna creare un nuovo *component template* `new_mapping_componenent` . In generale, i campi definiti come `userDefined`, specifici del flusso, sono sempre risultati utili da mappare come campi aggregabili e quindi il mapping dinamico nel nuovo componente può essere modificato come segue.

```
[
  {
    "userdefined-string": {
      "path_match": "event.userDefined.*",
      "mapping": {
        "type": "keyword"
      },
      "match_mapping_type": "string"
    }
  }
]
```

Questa regola permette di mappare tutti i campi di tipo `string`, `"match_mapping_type": "string"`, di `userDefined`, ovvero, quelli che rispettano `"path_match": "event.userDefined.*"`, come `keyword`, `"mapping": { "type": "keyword"}`.

Kibana, poi, offre un interfaccia comoda per la modifica del mapping esplicito. Quindi, duplicando il template iniziale e modificando, attraverso Kibana, nel mapping esplicito i campi interessati da `text` a `keyword` e aggiungendo anche il nuovo mapping dinamico si ottiene il *component template* desiderato.

3.2.1.6. Procedura per il remapping

Per quanto riguarda la procedura di remapping, dato un component template `new_mapping_component`, l'alias `alias` e un indice ad esso associato, che rispetta lo standard ed è chiamato `alias-number`, l'idea è quella di eseguire in sequenza:

1. la creazione di un index template, `new_alias_template`, che combina il `new_mapping_component` alla `axpo_ilm_policy` e risulti essere il template con priorità maggiore per il `target`, l'espressione regolare, `alias*`. Infatti, Elasticsearch non permette la creazione di due template con stessa priorità per lo stesso `target`.
2. la creazione di un nuovo indice, `alias-number+1`, generato con le configurazioni di `new_alias_template`

3. la modifica delle configurazioni di `alias` per rendere `alias-number+1` l'indice di scrittura e contemporaneamente rimuovere `alias-number` dagli indici associati ad `alias` . Questo permettere di non perdere informazioni durante la migrazione dei dati poiché tutti i nuovi dati verranno scritti su `alias-number+1` (*proprietà 2*). Inoltre, le ricerche sull'alias restituiranno solo dati scritti su `alias-number+1` e quindi solo dati mappati con il nuovo mapping (*proprietà 3*).
4. la migrazione di tutti i dati salvati in `alias-number` su `alias-number+1`. Questo viene fatto utilizzando la funzionalità di reindex. Reindex permette di migrare i dati da un indice sorgente ad uno destinazione e durante questo passaggio i dati, di default, vengono rimappati secondo il mapping dell'indice di destinazione (*proprietà 1)*.

In seguito `alias-number` risulta essere inutilizzato e quindi può essere, se lo si ritiene necessario, eliminato. Nei prossimi capitoli vengono illustrate due possibili implementazioni della procedura di remapping descritta.

3.2.2. Remapping con Kibana Console

Kibana Console è un'interfaccia interattiva che fa parte della suite di strumenti di gestione dei dati di Elasticsearch. Consente di eseguire query e interagire direttamente con l' API di Elasticsearch. Attraverso la console si possono inviare richieste RESTful a Elasticsearch e ricevere risposte immediate, facilitando l'analisi e l'esplorazione dei dati.

Consideriamo di avere le variabili:

- `${alias}` : `alias` , per esempio, `prod-mulesoft-ifsoutbound`
- `${lifecycle_policy}` : la lifecycle policy, per Axpo sarà sempre `axpo_ilm_policy`
- `${new_mapping_component}` : `new_mapping_component`
- `${old_number}` : il numero dell'indice con il vecchio mapping, per esempio, `000001`
- `${new_number}` : il numero del nuovo indice, che avrà il nuovo mapping , per esempio, `000002`

```
#1: creo o sostituisco il template
PUT /_index_template/${alias}-template
{
  "index_patterns": [
    "${alias}.*"
  ],
  "template": {
    "settings": {
      "number_of_shards": 2,
      "index": {
        "lifecycle": {
          "name": "${lifecycle_policy}",
          "rollover_alias": "${alias}"
        }
      }
    }
  },
  "composed_of": [
    "${new_mapping_component}"
  ]
```

```
}

#2: creo nuovo indice
PUT /${alias}-${new_number}

#3: modifico gli alias per scrivere su un nuovo indice
POST /_aliases
{
  "actions": [
    {
      "add": {
        "index": "${alias}-${new_number}",
        "alias": "${alias}",
        "is_write_index": true
      }
    },
    {
      "remove": {
        "index": "${alias}-${old_number}",
        "alias": "${alias}"
      }
    }
  ]
}

#4: eseguo reindex asincrono -> ritorna task_id
POST _reindex?wait_for_completion=false
{
  "source": {
    "index": "${alias}-${old_number}"
  },
  "dest": {
    "index": "${alias}-${new_number}"
  }
}
```

Lo script rispecchia quasi totalmente la descrizione della procedura di remapping, infatti, i commenti nel codice risultano esplicativi per descrivere le funzionalità dei metodi utilizzati. Sono comunque necessarie alcune precisazioni:

- per la creazione del `index_template` attraverso `PUT /_index_template/${alias}-template`:
    - `index_patterns` specifica quello che, fino ad ora, è stato definito come il `target` del *index template*
    - `"template": {"settings": {}}` permette di specificare la *lifecycle policy* da utilizzare e l'alias a cui fare riferimento nel processo di creazione di nuovi indici. Inoltre, al suo interno `"number_of_shards": 2` specifica il numero di *shard* dell'indice. Ogni *shard* è un'unità di archiviazione che contiene una parte dei dati dell'indice completo. Questa configurazione quindi rappresenta il modo in cui i dati vengono distribuiti e suddivisi

per ottenere una migliore gestione delle prestazioni e dell'affidabilità. Ho mantenuto la configurazione a 2, come per gli `index_template` già presenti.

- la chiamata `POST /_aliases` esegue contemporaneamente una lista di operazioni, `actions` , tramite le quali si aggiungono e si rimuovono indici dall'alias. L'impostazione `"is_write_index": true` specifica quale indice deve essere utilizzato come indice di scrittura per l'alias.
- l'operazione di `reindex` va effettuata utilizzando il `wait_for_completion=false`, ovvero, rendendola non bloccante. Questo è necessario poiché Kibana Console definisce un timeout per ogni richiesta e questo potrebbe essere ecceduto in caso di operazioni `reindex` su indici di grosse dimensioni. In questo modo l'operazione viene eseguita in modo asincrono e le viene assegnato un `task_id`.

  Si può verificare lo stato dell'operazione, utilizzando la Task Management API, attraverso:

```
GET /_tasks/${task_id}
```

3.2.3. Remapping con python-Elasticsearch

La procedura implementata con Kibana Console richiede, per ogni indice che si vuole rimappare, un'inizializzazione delle variabili utilizzate. Per automatizzare questa procedura ho sviluppato una semplice soluzione python, utilizzando la libreria *Python Elasticsearch*[2] che permette di interfacciarsi con l' API Elasticsearch ad alto livello.

- **`remapping`**:

  Questa funzione è l'implementazione, in python, della procedura di remapping.

```
async def remapping(es: Elasticsearch, index_pattern: str, new_index: str,
old_index: str, new_mapping: str, index_settings: Mapping[str, Any] | None = None,
keep_old: bool = True, keep_template: bool = True):
    # verifico che indice (da remappare) esista
    if (not es.indices.exists(index=old_index)):
        raise RuntimeError("The old index doesn't exist")

    # creo index-template
    template_name = f'{index_pattern}-template'
    es.indices.put_index_template(name=template_name,
index_patterns=index_pattern+"*", composed_of=[new_mapping])

    # se il nuovo indice esiste già lo elimino
    if (es.indices.exists(index=new_index)):
        print(f'{new_index} already existed...removing it and recreating it
\t[{index_pattern}]')
        es.indices.delete(index=new_index)

    # credo il nuovo indice
    es.indices.create(index=new_index, settings=index_settings)
    print(f"Index Creation -> DONE \t[{index_pattern}]")

    # lista di azioni da eseguire su alias
    add_new_alias: Mapping[str, Any] = {
        "add": {
```

```
            "index": new_index,
            "alias": index_pattern,
            "is_write_index": True
        }
    }
    remove_old_alias: Mapping[str, Any] = {
        "remove": {
            "index": old_index,
            "alias": index_pattern
        }
    }
    alias_actions: List[Mapping[str, Any]] = [add_new_alias, remove_old_alias]

    # eseguo le azioni su alias
    es.indices.update_aliases(actions=alias_actions)
    print(f"Aliases -> DONE \t[{index_pattern}]")

    # effettuo il reindex
    reindex_source: Mapping[str, str] = {
        "index": old_index
    }
    reindex_dest: Mapping[str, str] = {
        "index": new_index
    }
    print("Reindex -> start")
    es.reindex(source=reindex_source, dest=reindex_dest)
    print(f"Reindex -> DONE \t[{index_pattern}]")

    # eliminazione indice e template se specificato
    if (not keep_old):
        es.indices.delete(index=old_index)
        print(f"Cleaning -> removed old index \t[{index_pattern}]")
    if (not keep_template):
        es.indices.delete_index_template(name=template_name)
        print(f"Cleaning -> removed template \t[{index_pattern}]")
```

- **axpo_elastic_utilities**:

  Un'insieme di funzionalità aggiuntive, specifiche per il caso del remapping in Axpo.

```
from typing import Callable
from elasticsearch import Elasticsearch

# definisce una "struct" utile per il remapping (alias, old_index, new_index)
class AxpoReindexStruct:
    def __init__(self, alias: str, old_index: str, new_index: str) -> None:
        self.alias: str = alias
        self.old_index: str = old_index
        self.new_index: str = new_index

def get_reindexStruct_from_alias(es: Elasticsearch, alias: str,
get_new_indexName_from_old: Callable[[str, str], str]) -> AxpoReindexStruct:
```

```
    if (not es.indices.exists_alias(name=alias)):
        raise RuntimeError("Alias " + alias + " do no exist")

    old_index: str = list(es.indices.get_alias(name=alias).keys())[0]
    new_index: str = get_new_indexName_from_old(old_index, alias)

    return AxpoReindexStruct(alias, old_index, new_index)

def axpo_increment_indexNumber(old_index: str, alias: str) -> str:
    actual_index_number: str = old_index.split(alias+'-')[1]
    next_index_number: str = str(int('9' + actual_index_number) + 1)[1:]
    return alias + "-" + next_index_number
```

In particolare,

- `get_reindexStruct_from_alias` ritorna, partendo da un `alias` e una funzione generica `get_new_indexName_from_old` , una `AxpoReindexStruct`, ovvero, una oggetto ausiliario contenente informazioni utili per il remapping.

  La funzione ricava il nome dell'indice associato all'alias, `old_index`, e chiama `get_new_indexName_from_old(old_index, alias)`. Questa funzione generica, `(str,str) -> str`, è pensata per `resistuire` il nome del nuovo indice `new_index` partendo dal nome dal nome dell'indice precedente e dall'alias.

- `axpo_increment_indexNumber` è una funzione, `(str,str) -> str`, che costituisce un'implementazione di `get_new_indexName_from_old` per generare il nome del nuovo indice secondo lo standard utilizzato in Axpo, `old_index = alias-number` e `alias-number -> alias-number+1`.

- **`Main : esempio`**

  Il seguente codice in python rappresenta un possibile esempio di remapping. Nel caso da me trattato, risulta particolarmente utile poter effettuare il remapping dell'indice partendo solo dall'alias, questo poiché le data view di Kibana sono legate a uno o più alias. In generale, si vorrà effettuare il remapping di tutti gli indici legati agli alias di una data view. Partendo da questi alias `ALIAS_LIST` e sfruttando le configurazioni impostate nelle variabili globali, la funzione asincrona `main()` permette di effettuare, in modo semplice, il "remapping della data view".

```
from typing import Any, Mapping, List
from elasticsearch import Elasticsearch
import asyncio
from concurrent.futures import ProcessPoolExecutor

from remapping import remapping
import axpo_elastic_utilities as utils

AXPO_CLOUD_ID = ""
AXPO_API_KEY = ""
client = Elasticsearch(cloud_id=AXPO_CLOUD_ID, api_key=AXPO_API_KEY)

AXPO_MAPPING_COMPONENT_MORE_KEYWORDS = "axpo-mapping-component-more-keywords"
AXPO_ILM_POLICY = "axpo_ilm_policy"

AXPO_INDEX_SETTINGS_TEMPLATE: Mapping[str, Any] = {
```

```
        "index": {
            "lifecycle": {
                "name": AXPO_ILM_POLICY,
                "rollover_alias": ""
            },
            "number_of_shards": "2"
        }
    }

ALIAS_LIST: List[str] = ["dev-mulesoft-creditcheckoutbound","dev-mulesoft-
creditcheckinbound", "dev-mulesoft-creditcheckresult"]

async def main():
    for alias in ALIAS_LIST:
        elem = utils.get_reindexStruct_from_alias(client, alias,
utils.axpo_increment_indexNumber)
        AXPO_INDEX_SETTINGS_TEMPLATE["index"]["lifecycle"]["rollover_alias"] =
elem.alias
        print(f"Retrieved index pattern : [{elem.alias}] -> calling Reindexing")
        await remapping(es=client,
index_pattern=elem.alias,new_index=elem.new_index, old_index=elem.old_index,
new_mapping=AXPO_MAPPING_COMPONENT_MORE_KEYWORDS,index_settings=AXPO_INDEX_SETTINGS_
TEMPLATE,keep_old=False)

if __name__ == "__main__":
    asyncio.run(main())
```

Conclusa la fase di remapping indici Elasticsearch si può procedere allo sviluppo della dashboard avendo a disposizione tutte le funzionalità offerte da *Lens*. Nei prossimi capitoli verranno riportati i risultati dello sviluppo dashboard per i flussi analizzati.

3.3. Dashboard OTP

I flussi `OTP` riguardano i due servizi, di richiesta e verifica, di un codice numerico valido per un solo tentativo, OneTimePassword. Questo codice è utilizzato come step di autenticazione a due fattori per alcune operazioni, tipicamente eseguite dall'area privata dei portali web e delle applicazioni.

Il servizio di richiesta riceve in input il numero di cellulare a cui inviare il codice e restituisce un `operationId` con cui eseguire successivamente la verifica, questo comportamento è definito dallo stream **`otpsendsms`**. Quando il portale web o l'applicazione ricevono dall'utente il codice generato, lo abbinano all'`operationId` per verificarlo con una chiamata sincrona al secondo servizio, definito dallo stream **`otpverify`**.

3.3.1. Analisi

Dopo aver compreso il comportamento del flusso ed aver analizzato i log in modo da identificare quali di questi selezionae, si passa ad identificare le caratteristiche specifiche del flusso.

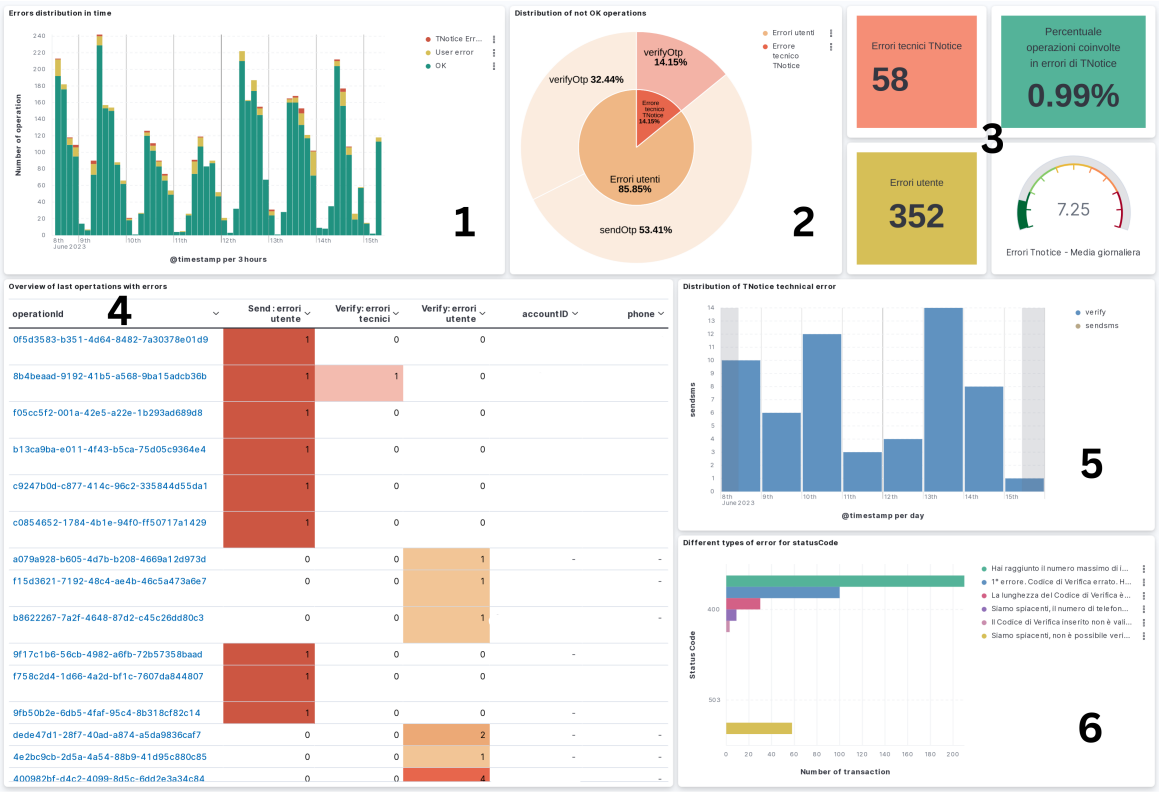
In particolare, per i flussi di `OTP` risulta importante differenziare tra gli **errori tecnici** e gli **errori utente**. Questo è facilitato dalla presenza di un campo specifico dei flussi, `statusCode`. Esso identifica lo stato dell'operazione. Partendo da questo è possibile differenziare gli errori per:

- **errori tecnici**: rappresentano errori dell'applicazione
    - `statusCode = 503`
- **errori utente**: errori causati da un comportamento errato dell'utente, risultano meno importanti da tracciare nelle analisi del monitoraggio
    - `statusCode = 400`

L'analisi iniziale aveva evidenziato, inoltre, casistiche di operazioni con `statusCode = 200` ma `debugMessage != 'OK'`, questo altro campo, anch'esso specifico del flusso, aggiunge informazione allo `statusCode` ed in particolare permette di differenziare i diversi tipi di errore. Queste casistiche, a seguito di una discussione con il collega referente dello sviluppo degli applicativi interessati, si è rivelata essere non prevista ed è quindi stata corretta. Infatti lo `statusCode = 200` deve rappresentare le operazioni senza alcun tipo di errori.

Segue ora una presentazione sintetica del risultato finale, ovvero, la dashboard. Per la descrizione di essa è necessario precisare che quando si utilizza il termine **operazioni** , si intende dire che è stato effettuato un raggruppamento per `operationId`. Come accennato in precedenza, l'`operationId` permette di identificare univocamente un'operazione tra tutte le operazioni dei flussi, poiché esso è mantenuto costante per la stessa un'operazione tra tutte le applicazioni di entrambi gli stream.

3.3.2. Dashboard



La dashboard OTP, qui sopra riportata, mostra di default le sue analisi sui dati degli ultimi 7 giorni. Inoltre, sono presenti degli strumenti "ricerca veloce" aggiuntivi per poter filtrare facilmente i dati analizzati dalla dashboard per alcuni campi, tra cui `operationId`, `phone` , `stream`.

La dashboard è così strutturata:

1. La distribuzione temporale, in numero di operazioni, degli errori: in rosso gli errori tecnici, in giallo gli errori utente e in verde le operazioni 'OK'.
2. La distribuzione delle operazioni per errori tecnici e utente e per stream. Da questa visualizzazione si evidenzia come gli errori tecnici provengano nella loro totalità dallo stream `otpverify`.
3. Il gruppo di visualizzazioni comprendere visualizzazioni alternative dei dati precedentemente descritti, ovvero, numero di errori tecnici e numero di errori utente. In aggiunta, sono presenti due metriche costruite a partire da esse, ovvero, la percentuale di errori tecnici sul totale delle operazioni e la media di errori tecnici per giorno.
4. Una tabella riassuntiva delle ultime operazioni con errori. In particolare, se letta da sinistra verso destra per colonna, permette di ricostruire lo stato dell'operazione, infatti, sono presenti il numero errori prima per `sendOtp` e poi per `verifyOtp`. Viene utilizzato il numero di errori poiché in caso di errori utente, questi possono essere anche più di uno per operazione, per esempio, se utente invia un codice sbagliato più volte.
5. La distribuzione temporale degli errori tecnici, classificati anche per stream. Questa visualizzazione risulta superflua poiché gli errori tecnici sono totalmente generati da un unico stream. Essa risultava utile durante l'analisi iniziale, per tracciare temporalmente gli errori tecnici presenti anche nello stream `sendopt`, non più presenti in seguito alla correzione effettuata.
6. Una distribuzione dei tipi di errore per `statusCode`, questa visualizzazione evidenzia come gli errori tecnici siano causati nella loro totalità da errori tecnici di TNotice, ovvero, il servizio esterno su cui si appoggiano le applicazioni del flusso `OTP`. Per questo nella dashboard è possibile trovare, talvolta, gli errori tecnici etichettati come *'errori di TNotice'*

Sviluppare questa prima dashboard mi ha permesso di familiarizzare con ELK. La fase di analisi e sviluppo dashboard è risultata veloce grazie alla presenza di log uniformi con campi specifici orientati a facilitare l'analisi durante il monitoraggio. Questa situazione non risulterà tale nel caso della verticale funzione `IFS`, caratterizzata da una maggiore complessità e eterogeneità tra gli applicativi che la compongono.

3.4. Dashboard IFS

La verticale funzionale `IFS` riguarda la richiesta di servizi e prestazioni tecniche sui punti di fornitura gas o energia verso i distributori finali, intermediati dal sistema IFS.

IFS agisce da hub, raccogliendo le richieste, flussi `0050`, attraverso l'unico web service esposto. Questo servizio risponde sincronicamente con l'ammissibilità formale, flusso `100HUB`. Solo in caso di richiesta formalmente corretta IFS la trasmette al distributore. Si attende quindi l'arrivo asincrono dell'effettiva presa in carico della richiesta, flusso `0100`, come primo esito. Se la presa in carico è positiva segue di norma l'esito della lavorazione, flusso `0150`. In base al tipo di servizio o prestazione possono esistere diversi flussi opzionali in aggiunta a questi.

Gli esiti emessi dal distributore sono sempre recapitati al richiedente, mentre il `100HUB` viene notificato solo in caso di non ammissibilità per consentire le necessarie azioni correttive. Le richieste di prestazioni Axpo hanno origine da diversi canali. Per ogni pratica IFS, infatti, viene mantenuto un parametro che identifica il canale, ovvero, l'applicativo da cui proviene la richiesta e a cui dovrà essere inoltrata la risposta. Internamente i canali sono, al momento, gli applicativi `SFDC` e `SAP`. Il servizio di

IFS è messo anche a disposizione dei reseller per alcune prestazioni, al momento uno solo, `ALG`. E' anche disponibile un servizio sincrono per la verifica puntuale dello stato di una singola pratica.

La verticale `IFS` è diviso in due stream **`ifsoutbound`** per le richieste verso IFS e **`ifsinbound`** per gli esisti e le notifiche in arrivo da IFS.

3.4.1. Analisi

L'analisi della verticale `IFS` è risultata in tutti i suoi aspetti più complicata. In primo luogo, a differenza dei flussi `OTP`, la verticale funzionale `IFS` deve gestire situazioni al suo interno molto articolate, specifiche e in molti casi con comportamenti eterogenei. I flussi sono caratterizzati da un numero elevato di prestazioni. Una prestazione è una tipologia di operazione da o verso IFS ed è identificata da una sigla alfanumerica. Ogni prestazione è , poi, caratterizzata da una sequenza logica di flussi, identificati da codici, `flowCode`.

Per esempio, un attivazione contrattuale è identificata da `VT1` e segue la sequenza logica di esecuzione:

- `0050` : flusso invio richiesta di attivazione contrattuale, `ifsoutbound`
- `0100HUB` : flusso di gestione risposta sincrona riguardo all'ammissibilità formale, `ifsoutbound`
- `0100` : flusso di gestione risposta asincrona dell'esito della presa incarico della pratica, `ifsinbound`
- `0150`: flusso di gestione risposta finale con l'esito di lavorazione, `ifsinbound`

Risulta, quindi, non pratico esplorare ogni possibile prestazione nei dettagli per analizzarne i casi limite ed esecuzioni tracciate più o meno correttamente dai log. L'idea è di sviluppare una vista il più generale possibile dello stato delle prestazioni della verticale, cercando allo stesso tempo di identificare gli errori principali. Oltre alla complessità del flusso, la possibilità di lavorare solo su dati di ambienti pre-produttivi, non essendo ancora `IFS` rilasciata in produzione, non facilita la validazione dell'efficacia delle soluzioni scelte, che sono spesso state modificate durante il processo di realizzazione.

Analizzando la gestione degli errori è necessario evidenziare che gli applicativi della verticale funzionale implementano la **`politica dei retries.`** Essendo gli applicativi di IFS fortemente basate sull'utilizzo di servizi esterni, alcuni errori potrebbero essere generati da disservizi momentanei di questi. Si utilizza quindi un flusso di risottomissione automatica della pratica in caso di errore. Questa risottomissione avviene per un numero massimo di 6 volte e in seguito al sesto fallimento viene salvata su database come pratica fallita. Di conseguenza, non tutte le pratiche che presentano errori sono pratiche fallite definitivamente.

3.4.2. Dashboard

Anche per la dashboard IFS, qui sopra riportata, le analisi sono impostate, di default, sugli ultimi 7 giorni e sono presenti strumenti di "ricerca veloce" aggiuntivi, per esempio, sul numero di pratica, `requestCode`.

La dashboard è così strutturata:

1. Una tabella riassuntiva dello stato delle pratiche. Leggendo questa visualizzazione da sinistra verso destra si può ricomporre lo stato di una pratica seguendo i `flowCode` in comune alla maggior parte delle pratiche. Questa visualizzazione, di carattere generale, risulta molto utile se combinata con la ricerca per numero di pratica.
2. Alcune metriche sul numero di pratiche gestite e fallite definitivamente, divise per stream: `ifsoutbound` per le richieste e `ifsinbound` per gli esiti e le notifiche. Le notifiche sono flussi in ricezione da IFS che rappresentano comunicazioni legate ad una pratica ma che vengono tracciate, in parte, con identificativi, ovvero `correlationId`, diversi.

Data la grande eterogeneità del sistema, molte casistiche, tra cui spiccano le notifiche, mancano di `requestCode` nei log, questo forza lo spostamento dell'analisi sui `correlationId`, operazioni, per non perdere queste casistiche nelle metriche.

A seguito del rilascio in produzione è risultato evidente come le numeriche delle pratiche in errore, specialmente notifiche, non rappresentassero "veri errori" ma semplicemente mancate gestioni di alcune casistiche, per esempio, la ricezione di un notifica da IFS non utile e che, quindi, fallisce sempre definitivamente anche se non risulta necessaria per il corretto sviluppo della pratica.

In seguito a queste osservazioni, un collega si è occupato dell'identificazione dei "falsi negativi" e della creazione di un filtro opportuno per tracciare solo i "veri errori". Questo filtro è integrato nella dashboard per mostrare le numeriche degli errori reali come errori `FINALI`. Risulta comunque utile avere una panoramica sui falsi positivi e quindi anch'essi sono stati mantenuti nelle visualizzazioni ed etichettati come errori `TOTALI`.

3. Alcune metriche sulle pratiche e operazioni fallite definitivamente e il rapporto tra pratiche con errori e pratiche fallite definitivamente.
4. Distribuzione temporale delle operazioni fallite definitivamente riportate in rosso e delle operazioni senza errori, in verde.
5. Una tabella riassuntiva sullo stato delle ultime operazioni fallite.
6. La distribuzione delle operazioni fallite definitivamente, `TOTALI`, per tipologia di pratica.
7. La distribuzione delle operazioni fallite definitivamente, `TOTALI`, per canale, e per stream.
8. La distribuzione delle richieste fallite definitivamente, `FINALI`, per tipologia di pratica.

4. Applicativo Mulesoft

Nella seconda parte del tirocinio mi sono occupato dello sviluppo di un applicativo Mulesoft. La task dell'applicativo è quella di automatizzare la sottomissione delle pratiche terminate in errore. Queste pratiche, come accennato in precedenza, in seguito ad un fallimento definitivo, vengono salvate su un database, nella tabella `MQ_RECOVERY`. Per il corretto funzionamento delle logiche d'integrazione tra gli applicativi, è necessario che queste pratiche riescano ad essere correttamente elaborate. Molti di questi fallimenti, però, non sono dovuti ad errori interni ma, spesso, si possono identificare pattern per automatizzare questo processo ad intervalli regolari. L'attività di recupero di queste pratiche è un compito manuale collegato all'attività di monitoraggio. Si vuole quindi sviluppare una soluzione per automatizzare una porzione dei recuperi e allo stesso tempo un API per facilitare il processo di recupero manuale delle pratiche.

4.1. Soluzione

Lo stream designato per l'applicativo di recupero automatizzato è **`recovery`**. La soluzione è orientata al supporto del team di tecnici che si occupa del monitoraggio ed è organizzata in :

- Un API per effettuare il recupero automatizzato di pratiche eseguendo una "query"
- Un batch, ovvero, un processo che, ad intervalli regolari, recupera automaticamente delle pratiche rispetto a delle logiche determinate a priori

Durante lo sviluppo, ho usufruito frequentemente della documentazione Mulesoft [3]. Nel corso delle presentazione delle soluzioni adottate, verranno riportati riferimenti più specifici ad essa.

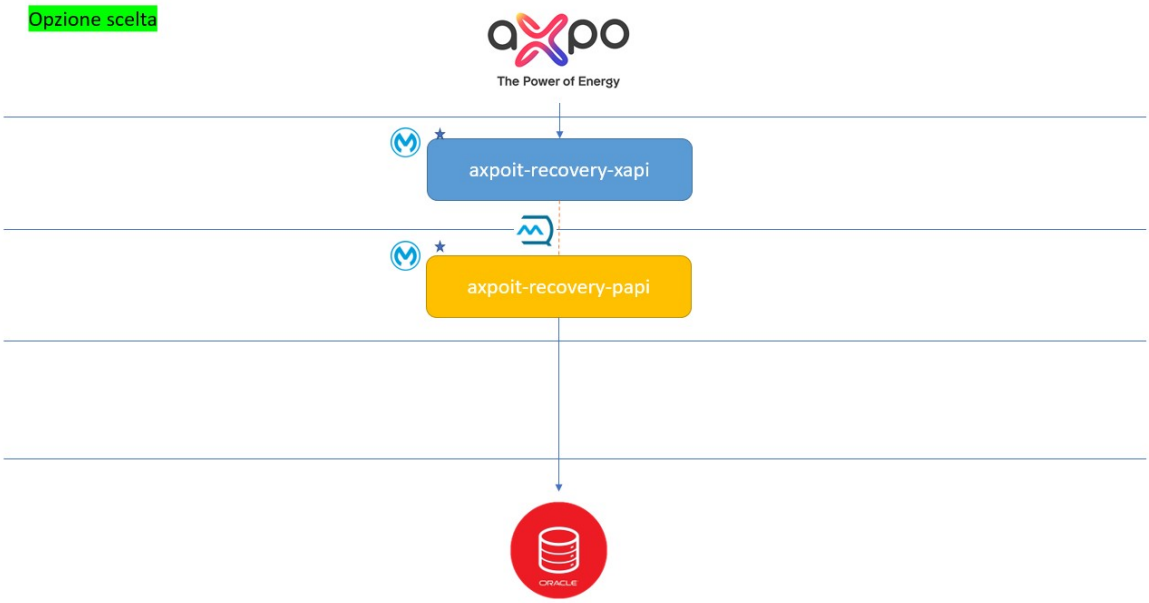
Prima di presentare l'architettura ad alto livello della soluzione, è necessario introdurre alcuni concetti.

- Gli applicativi MuleSoft implementano un *middleware d'integrazione* suddiviso in tre layer:
    - Il *layer di presentazione* è responsabile di fornire un'interfaccia standardizzata per consentire l'accesso e l'utilizzo delle funzionalità offerte dalle applicazioni e dai servizi esposti.
    - Il *layer di processo* è responsabile della definizione e dell'orchestrazione dei flussi. Qui, vengono gestiti i processi d'integrazione e viene garantita la sincronizzazione e la trasformazione dei dati tra diverse applicazioni e sistemi.
    - Il l*ayer di sistema*, il quale non sarà utilizzato nello sviluppo trattato, è il livello più basso del middleware d'integrazione. Esso si occupa della connessione e della comunicazione diretta con i diversi sistemi e applicazioni aziendali.

  L'esistenza di questa suddivisione facilita l'integrazione delle applicazioni e dei sistemi esistenti all'interno di un'organizzazione, ottenendo, quindi, maggiore agilità, flessibilità e scalabilità nell'implementazione delle soluzioni d'integrazione.
- L'applicativo utilizzerà le code MQ. Le MQ, Message Queue, sono una tecnologia di messaggistica utilizzata per consentire la comunicazione asincrona tra diversi componenti di un sistema distribuito. Esse possono essere utilizzate come meccanismo d'integrazione per garantire la consegna affidabile dei messaggi tra i sistemi.

Le code MQ consentono di inviare e ricevere messaggi in modo asincrono, consentendo ai componenti del sistema di operare in modo indipendente e senza dipendere dalla disponibilità immediata degli altri componenti. Attraverso l'utilizzo delle code MQ, gli applicativi MuleSoft possono implementare il pattern di messaggistica di tipo *Publish-subscribe* [4].

Architettura ad alto livello



In seguito ad una fase di analisi e discussione iniziale, è stata delineata un'architettura ad alto livello, qui sopra riportata, per il flusso di recupero automatizzato. L'applicativo è così strutturato:

- **`axpoit-recovery-xapi`** : costituisce l'applicazione del layer di presentazione. Questa applicazione implementa un API, definita nella fase iniziale di Design. Essa si occupa di validare la correttezza della richiesta ricevuta, ed in seguito pubblica il messaggio, ovvero la richiesta, su una coda MQ dedicata, `recoveryQueue`.
- **`axpoit-recovery-papi`:** costituisce l'applicazione di processo. Questa implementa due funzionalità :

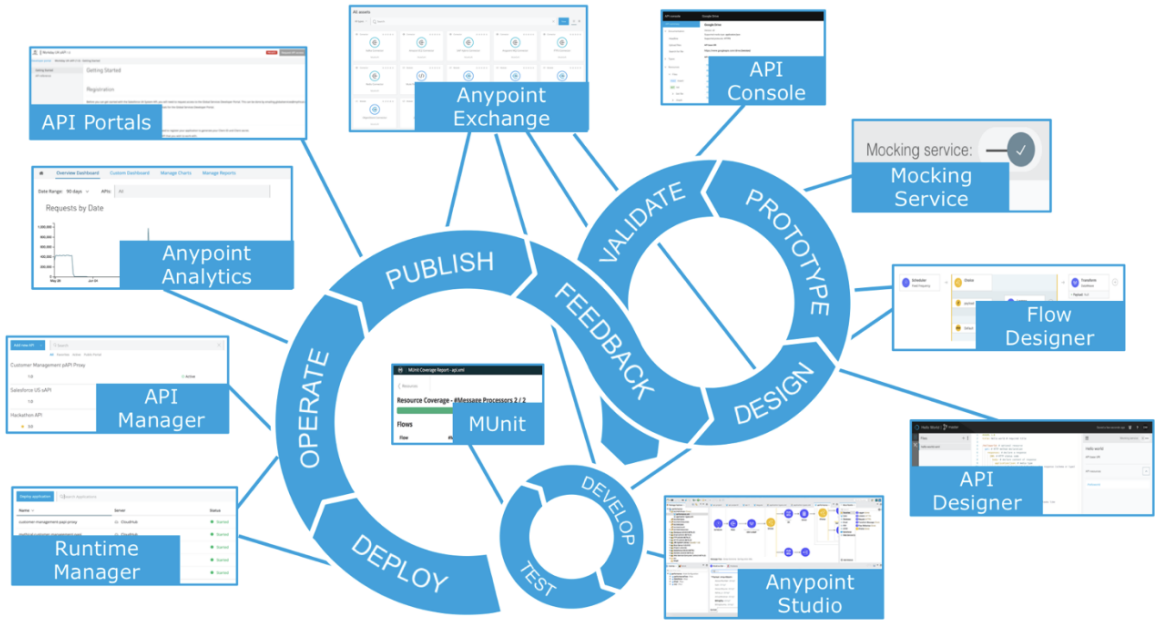
  1. esecuzione del recupero per le richieste pubblicate su `recoveryQueue`
  2. esecuzione di un batch per automatizzare il recupero periodico degli errori regolari e ripetitivi

  Entrambe queste due funzionalità condividono una logica comune basata sull'interrogazione di un *database Oracle* [5]. Essa esegue:

  - Estrazione dal database delle pratiche da recuperare. Questo avviene attraverso una query prefissata, nel caso del batch. Mentre, viene specificata, attraverso una opportuna interrogazione del API, nel caso del recupero dalla coda.
    - ogni record in `MQ_RECOVERY` corrisponde ad una pratica e contiene tutte le informazioni necessarie per il suo recupero
  - Recupero di ogni pratica estratta. Questo procedimento avviene tramite la ripubblicazione di essa sulla coda dell'applicativo opportuno.
  - Modifica sul database dello stato delle pratiche recuperate.

4.2. Sviluppo

Lo sviluppo che verrà trattato segue le linee guida di Mulesoft , riassunte nel seguente schema.



In particolare, le fasi del processo che verranno maggiormente approfondite sono:

- **Fase di design**: durante questa fase, si definisce l'architettura dell'integrazione. In particolare, l' API viene sviluppata nello strumento Design Center, utilizzando *RAML* [6], un linguaggio di modellazione delle API che offre una sintassi semplice per definire l' API in modo dettagliato. Una volta definita la specifica del API, essa viene pubblicata su Exchange, una piattaforma dove si può accedere a API, frammenti di codice, connettori e altri asset riutilizzabili. Attraverso Exchange è, inoltre, possibile eseguire una prima validazione del API attraverso l'utilizzo di un Mocking Service.
- **Fase di implementazione**: utilizzando Anypoint Studio, si implementano i flussi d'integrazione, creando i connettori per accedere ai sistemi esterni, scrivendo logica di business e configurando le trasformazioni dei dati. In particolare, Anypoint Studio offre la funzionalità di generazione automatica di un APIkit, partendo dalle specifiche di un API. Il componente APIkit utilizzerà le specifiche RAML per validare le richieste in ingresso e le risposte in uscita rispetto al contratto definito nel API.

  Lo sviluppo è seguito dall'attività di deployment. Questa è diversificata per ambiente di sviluppo. In Axpo, una pipeline permette di automatizzare il rilascio nell'ambiente pre-produttivo `SIT`. Una volta che l'applicativo risulta completo e correttamente funzionante, esso può essere promosso ad un ambiente pre-produttivo di livello superiore, per esempio `UAT`. Verrà eseguita un'ulteriore fase di test più accurata che cercherà di identificare problematiche negli applicativi, prima del rilascio in ambiente di produzione.

  - Per gestire le configurazioni dell'applicativo a seconda dell'ambiente di sviluppo si utilizzano le properties, ovvero, variabili di configurazione dell'applicativo. Le properties possono essere, per esempio, le impostazioni per la connessione ad un database, che sono, generalmente, diversificate per ambiente di sviluppo. Viene, dunque, creato un insieme di properties specifiche che riflettono la configurazione dell'applicativo per ambiente di sviluppo. Queste vengono caricate e utilizzate per configurare l'applicativo durante il deployment e possono essere modificate a runtime attraverso Runtime Manager. Mulesoft permette, inoltre, la configurazione di properties sicure, quali, per

esempio le credenziali. La configurazione e la gestione delle properties, non verrà approfondita maggiormente nel corso dello sviluppo. Risulta comunque necessario evidenziare la sua centrale importanza per il corretto funzionamento dell'applicazione.

- **Fase di test**: in contemporanea all'implementazione degli applicativi, è fondamentale testare la loro corretta funzionalità. In particolare, verrà utilizzato Postman come strumento per il test funzionale, partendo dall'interrogazione del API e analizzando la sequenza di log derivanti dall'esecuzione degli applicativi.

Il processo di sviluppo è, inoltre, caratterizzato da molteplici fasi intermedie necessarie per assemblare i risultati delle fasi principali e permettere il corretto funzionamento degli applicativi. Tra esse, le più importanti sono:

- La creazione e configurazione delle code MQ, specifica per ambiente di sviluppo.
- La configurazione del API. Una volta definita l' API e pubblicata su Exchange, l' API deve essere configurata e attivata per ambiente di sviluppo. Le configurazioni necessarie sono molteplici, tra le più importanti vi sono quelle di rete, impostazioni di routing, e di sicurezza, quali i metodi di autenticazione implementati.

  Una volta attivata, l' API è raggiungibile, esternamente e/o internamente, e può essere associata ad un applicativo del layer di presentazione. La configurazione avviene attraverso l'utilizzo di un `apiID` che viene utilizzato dal processo di Autodiscovery. In un ambiente di esecuzione Mulesoft, questo processo permette, a runtime, l'associazione tra un applicativo e una specifica API.

4.2.1. Design

La prima fase è quella di creazione delle specifica del API, utilizzando RAML. Si parte da un progetto template, che fornisce uno scheletro contenente librerie che gestiscono autenticazione ed errori comuni. In particolare, l'applicativo di recupero sarà reso disponibile solo internamente agli addetti all'attività di monitoraggio e recupero. Per questo, come autenticazione è stata mantenuta unicamente quella tramite JWT. Successivamente si passa alla configurazione del contenuto delle richieste e delle risposte. Le specifiche guidano la validazione delle richieste in entrata e inoltre forniscono documentazione specifica agli utilizzatori del API.

Dato che la specifica della risposta serve solo a scopo di documentazione, non verrà approfondita. Mentre si analizzerà la specifica per il `body` delle richieste che, più precisamente, sono di tipologia `HTTPS POST`.

4.2.1.1. Richiesta

La richiesta al API può avvenire in 2 modi principalmente:

- attraverso una `query`. Questa estrarrà i record che soddisfano i filtri inviati, che possono essere:
  - `correlationIds`: una lista di `correlationId`, ovvero, stringhe
  - `businessKeys`: una lista di `businessKey`, ovvero, stringhe
  - una range temporale, identificato da
    - `startDate`: la data d'inizio
    - `endDate`: la data di fine
- attraverso una lista di `IDs`, ovvero, una lista di identificatori univoci dei record in `MQ_RECOVERY`

Entrambe le due possibili richieste dispongono, inoltre, di un campo booleano `force`, impostato di default a `false`. Questo campo se valorizzato a `true` indica la volontà di ignorare il filtro di default applicato alla query di estrazione. Questo, attualmente, seleziona solo i record con `STATUS = FAILED`. In seguito al recupero, infatti, il campo `STATUS` verrà impostato a `RECOVERY_OK`, per tutti i record recuperati correttamente, altrimenti a `RECOVERY_KO`,per tutti i record il cui recuperò è fallito.

La definizione delle specifiche avviene utilizzando i RAML DataType. Quella che segue è una versione semplificata dei *DataType* prodotti, in cui si utilizzano nomi dei sottotipi invece che riferimenti a file o librerie. Nel capitolo sul test verranno riportano alcuni esempi, esplicativi, di richieste valide.

- Si parte dalla definizione dei tipi base:
    - `query`

```
#%RAML 1.0 DataType
type: object
additionalProperties: false
description: query to filter the elements to recover
minProperties: 1
properties:
  correlationIds:
      type: string[]
      uniqueItems: true
      minItems: 1
      required: false
      description: List of correlationIds to recover
      example: ["e1cf07e0-e433-11ed-abc6-02d2c7a76e46", "60de6e90-e475-11ed-
bddd-02b5b50e0cc2"]

  businessKeys:
      type: string[]
      uniqueItems: true
      minItems: 1
      required: false
      description: List of businessKeys to recover
      example: ["IfsInbound", "IfsOutbound"]

  startDate:
      type: datetime
      format: rfc3339
      required: false
      description: Recover starting from this date
      example: 2023-04-23T12:42:56.220Z

  endDate:
      type: datetime
      format: rfc3339
      required: false
      description: Recover starting from this date
      example: 2023-05-23T12:42:56.220Z
```

    - `ID`

```
#%RAML 1.0 DataType
type: integer
minimum: 1
format: int64
description: an ID of the table MQ_RECOVERY table
example: 42
```

- force_flag:

```
#%RAML 1.0 DataType
type: boolean
default: false
description: when set to True -> recover regardless of the state or error
example: true
```

- Costruendo, poi, i 2 tipi principali:
  - recover_query

```
#%RAML 1.0 DataType
type: object
additionalProperties: false
description: recovery with a query + force-flag
properties:
  query:
      type: query
      required: true
      description: query the elements to recover

  force:
    type: force_flag
    required: false
```

  - recover_IDs

```
%RAML 1.0 DataType
type: object
additionalProperties: false
description: recovery with IDs + force-flag
properties:
  IDs:
    type: id[]
    minItems: 1
    required: true
    description: list of the exact IDs to recover
    example: [33, 42]


  force:
    type: force_flag
    required: false
```

- Assemblando, infine, il body della richiesta utilizzando un RAML Union Type:

```
body:
    application/json:
        type: recover_query | recover_IDs
```

4.2.2. Test

I test sono implementati attraverso l'utilizzo di chiamate `HTTPS` al API, costruite ed eseguite utilizzando Postman. Si riportano unicamente i dettagli relativi al `body` delle richieste, con il solo scopo di chiarificare i dettagli delle richieste, precedentemente descritti nella fase di design del API.

La gestione delle configurazione aggiuntive, necessarie per effettuare le richieste in modo corretto, è affidata ad un Environment di Postman. Questo permette, in base all'ambiente di sviluppo impostato, di configurare dinamicamente l'indirizzo base della richiesta e facilita la richiesta e configurazione del token di autenticazione necessario per effettuarla.

Richiesta con lista di IDs: `recover_IDs`

```
{
  "IDs": [43, 22],
  "force": false
}
```

Richiesta con query : `recover_query`

- lista di correlationId:

```
{
  "query": {
    "correlationIds": ["ae7359b0-5959-11ed-ab3c-064ca0460556"]
  }
}
```

- lista di businessKeys e data d'inizio

```
{
  "query": {
    "businessKeys": ["Pricing"],
    "startDate" : "2023-06-08T08:12:29.868Z"
  }
}
```

4.2.3. Implementazione

Come accennato precedentemente, Anypoint Studio permette di sviluppare applicativi Mulesoft utilizzando la tecnica di sviluppo per componenti. Un'applicazione Mulesoft è composta da flows, ognuno dei quali è composto da una sequenza di componenti che si scambiano messaggi, utilizzando variabili e properties per effettuare la ricezione, l'elaborazione e l'invio di dati. Inoltre, l'applicativo farà uso di componenti per interfacciarsi con database, Database Connector e altri per effettuare i log, sviluppati internamente da Axpo.

4.2.3.1. DataWeave

Per trasformare i dati viene utilizzato il linguaggio di programmazione funzionale DataWeave [7]. DataWeave è un linguaggio che consente di trasformare e manipolare dati strutturati, per esempio `JSON`, in modo efficace durante il processo d'integrazione.

Nelle prossime sezioni verrà discussa l'implementazione degli applicativi ma non verrà analizzata nei dettagli le manipolazioni dei dati attraverso l'utilizzo di DataWeave. Segue, dunque, un esempio esplicativo di una possibile soluzione al problema di poter utilizzare i valori della query ricevuta, come richiesta attraverso l' API, combinandoli con altre variabili per generare la query SQL per l'estrazione dei record da recuperare.

Considerando di avere, inizializzate nello scope, le variabili `table_name`, `batch_size` e un `payload` contenente una richiesta, oggetto `JSON`, che rispetti il formato, precedentemente descritto, per una richiesta al API per i recuperi. La trasformazione è basata sulla concatenazione di stringhe, inizializzate attraverso funzioni opportune. L'esempio è riportato in blocchi separati per facilitare la descrizione di ognuno di essi e si articola in:

- Configurazioni DataWeave:

```
%dw 2.0
output text/plain
```

  Questa comando iniziale permette di specificare la versione e il formato di output. Per l'esempio trattato è utilizzato il formato `text/plain`, poiché esso è il formato utilizzato, per la query SQL da eseguire, dai componenti che si interfacciano con il database.

- Funzione ausiliaria per stabilire se la `query` è `force`:

```
fun get_force() : String =
    if(payload.force?)
        if(payload.force)
            'true'
        else
            'false'
    else 'false'
```

  L'operatore destro `?` ritorna `true` se la variabile è inizializzata, ovvero, se è diversa da `null`, `false` altrimenti.

- Variabili ausiliarie:

```
  //parte iniziale della query
  var query_start ="SELECT ID, CORRELATION_ID, BUSINESS_KEY, QUEUE_NAME,
USER_PROPERTIES, PAYLOAD FROM " ++  table_name ++ " WHERE "

  //filtro finale della query
  var filter_end = " AND (STATUS='FAILED' OR 'true' = '" ++ get_force() ++  "') AND
ROWNUM <= " ++ batch_size

  //stabilisce l'ordine delle variabili e quindi delle clausole nella query
  var list_key =
[payload.query.startDate,payload.query.endDate,payload.query.businessKeys]
```

L'operatore ++ permette la concatenazione di oggetti compatibili. In questo caso viene utilizzato per la concatenazione di stringhe.

- Funzioni di utilità per gestire la concatenazione:

```
//verifcica che esista l'elememento successivo nella query
fun hasNextKeyword(nexts : Array, lim_index : Number): Boolean=
   !isEmpty(nexts filter ((item, index) -> (item != null and index >= lim_index)))

//se esiste l'elemento successivo della query ritorna 'AND'
fun conditionalAnd(key_pos : Number): String=
    if(hasNextKeyword(list_key,key_pos))
        ' AND '
    else
        ''
//funzione ausiliaria per reduce
fun checkSeparator(accumulator : String) : String=
    if(accumulator == '')
        ''
    else
```

La funzione `hasNextKeyword` utilizza la funzione `filter`, predefinita in DataWeave. Questa prende in input un array e restituisce un nuovo `array` contenente solo gli elementi che soddisfano la condizione di filtro, in questo caso la presenza di elementi successivi nella query, a seconda della posizione interessata.

- Funzioni per la composizioni di clausole SQL di tipo IN:

```
fun inConversion(n : Array<String>) : String=
  n reduce ((item, accumulator = "") ->  accumulator ++ checkSeparator(accumulator)
 ++  "'" ++ item ++  "'")

fun inKeyword(keyword, name : String, key_pos : Number): String=
    if(keyword != null)
        name ++ ' IN (' ++ inConversion(keyword) ++ ')' ++ conditionalAnd(key_pos)
    else
        ''
```

La funzione `inConversion` utilizza la funzione `reduce`, predefinita in DataWeave. Essa viene utilizzata per ridurre un insieme di elementi a un singolo valore applicando una logica di aggregazione specificata.

- Funzioni specializzate per ogni clausola e composizione finale

```
fun correlationIds() =
    inKeyword(payload.query.correlationIds, 'CORRELATION_ID', 0)

fun startDate() =
  if(payload.query.startDate != null)
    'TIMESTAMP > TO_TIMESTAMP(\'' ++  payload.query.startDate ++ '\',\'YYYY-MM-
DD"T"HH24:MI:SS.FF"Z"\')' ++ conditionalAnd(1)
    else
        ''
```

```
fun endDate() =
  if(payload.query.endDate != null)
    'TIMESTAMP < TO_TIMESTAMP(\'' ++  payload.query.endDate ++ '\',\'YYYY-MM-
DD"T"HH24:MI:SS.FF"Z"\')' ++ conditionalAnd(2)
    else
        ''


fun businessKeys() =
    inKeyword(payload.query.businessKeys, 'BUSINESS_KEY', 3)
---
//composizione della query
query_start ++ correlationIds() ++ startDate() ++  endDate() ++  businessKeys() ++
filter_end
```

Non entrando in ulteriori dettagli sulla manipolazione dei dati, segue una descrizione sintetica dell'implementazione dei due applicativi che costituiscono il flusso di recupero.

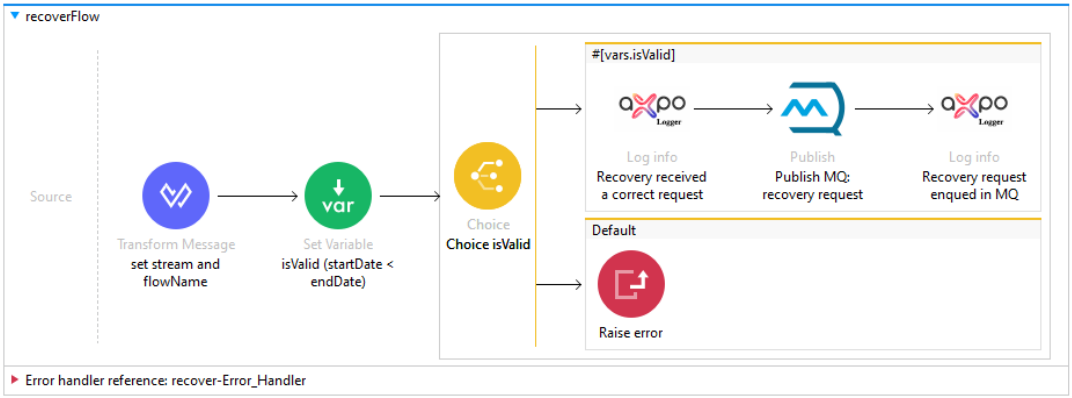
4.2.3.2. axpoit-recovery-xapi

Lo sviluppo del layer di presentazione, ovvero dell'applicativo `axpoit-recover-xapi`, risulta meno complesso poiché la sua finalità è quella di implementare una gestione delle richieste per l' API sviluppata e pubblicare quelle corrette sulla coda MQ, da cui poi leggerà l'applicativo del layer di processo attraverso il pattern *Publish-subscribe*.

In particolare, avendo descritto in RAML le specifiche del API, questa viene automaticamente implementata, utilizzando le funzionalità di Anypoint Studio, con l' APIkit. Risulta, poi, necessario implementare il vincolo `startDate < endDate` per le richieste di tipo `query`. Questo vincolo non può essere espresso in RAML e va quindi implementato nella logica dell'applicativo di presentazione. In caso di errore per questa validazione è opportuno ritornare un `BAD_REQUEST error`, implementato attraverso la gestione di un errore specifico.

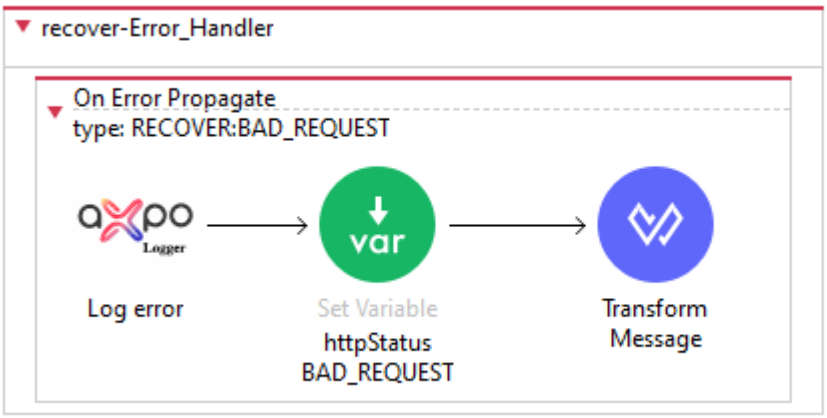
L'applicativo è, quindi, costituito da:

- il flusso principale, qui sotto riportato, che si occupa di:
    - inizializzare le variabili necessarie per i log, quali lo `stream` e il `flowName`. Il `flowName` permette di diversificare i recuperi per tipologia all'interno dello flusso `recovery`, classificandoli per recuperi via chiamata API o recuperi eseguiti dal batch ad intervalli regolari.
    - la verifica del vincolo `startDate < endDate` che determina:
        - in caso di correttezza, la pubblicazione della richiesta sulla coda MQ, preceduta e seguita da log per tracciare lo stato dell'esecuzione
        - altrimenti, il sollevamento di un errore, gestito dal `recover-Error-Handler`



Risulta utile evidenziare, come i componenti che si occupano del log, ovvero, quelli identificati dall'icona `AXPO_logger` , hanno anche una funzionalità implicita che è quella documentativa. Risulta, infatti, utile al fine di comprendere rapidamente la logica del flusso, utilizzare la descrizione dei componenti di log, poiché questi sono inseriti tra le varie fasi e quindi stati, del flusso.

- Il `recover-Error-Handler`, riportato nella seguente immagine, che gestisce l'errore sollevato dal flusso principale. Esso effettua un log dell'errore funzionale, imposta lo stato corretto per la risposta HTTP e prepara la risposta che verrà poi utilizzata dal APIkit per restituire una risposta al chiamante.



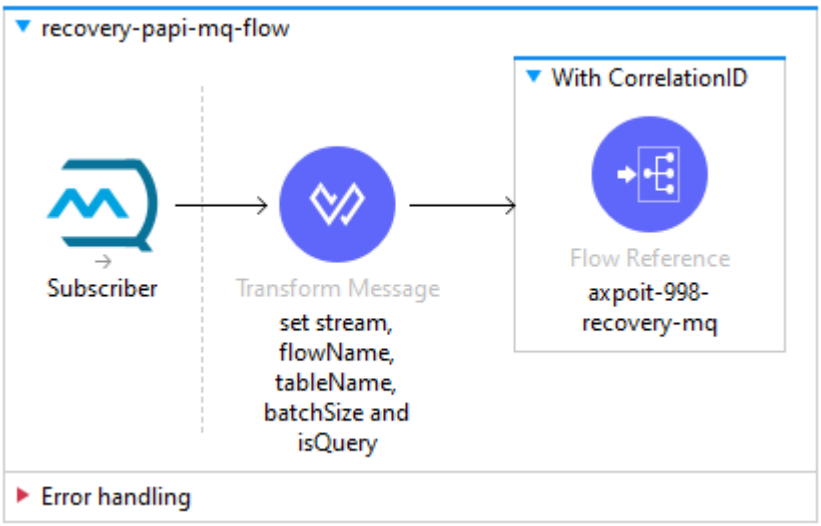
4.2.3.3. axpoit-recovery-papi

La logica dell'applicativo del layer di processo risulta più articolata ed è caratterizzata da:

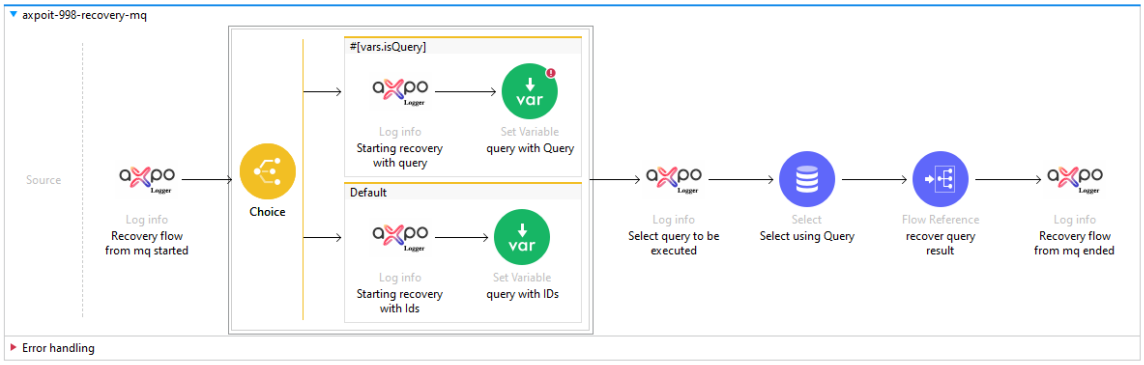
- Una logica per l'estrazione dei record da recuperare a partire da una richiesta sulla coda MQ.
- Un flusso per la configurazione e l'estrazione periodica di record da recuperare, ovvero, il batch.
- Un insieme di flussi comuni per la gestione del recupero e aggiornamento su database dei record estratti

Estrazione record da MQ

L'immagine qui sotto riportata descrive l'inizio della logica di gestione di una richiesta pubblicata su coda MQ. Attraverso la registrazione di un `susbscriber` alla coda MQ, si riceve un messaggio, ovvero, una richiesta. Si inizializzano, poi, le variabili necessarie per la manipolazione dei dati e per i log. Infine, viene chiamato il flusso `axpoit998-recovery-mq` a cui viene aggiunta l'informazione del `correlationId`, utile per tracciare, nei log, la sequenza di esecuzione per ogni richiesta

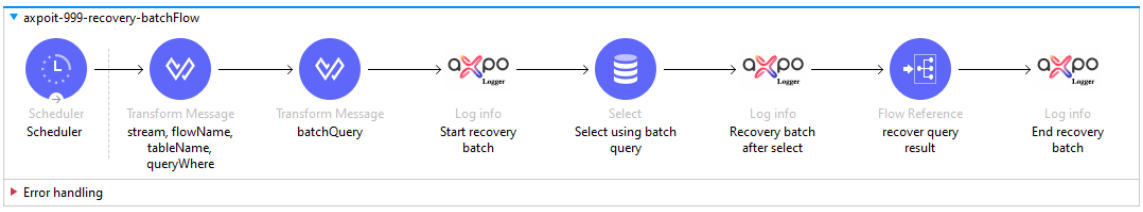


Il flusso `axpoit998-recovery-mq`, qui sotto riportato, gestisce la costruzione della query SQL per l'estrazione dei record da `MQ_RECOVERY` a seconda della tipologia di richiesta ricevuta, `query` o `lista di Ids`. La costruzione della query SQL avviene all'interno dei componenti identificati con `Transform Message`, attraverso l'utilizzo di logiche DataWeave come quella precedentemente descritta. La query SQL viene, poi, eseguita dal componente `Select`, che si interfaccia con il database Oracle. Infine, i risultati vengono passati al flusso comune di recupero e aggiornamento su database.



Estrazione record con Batch

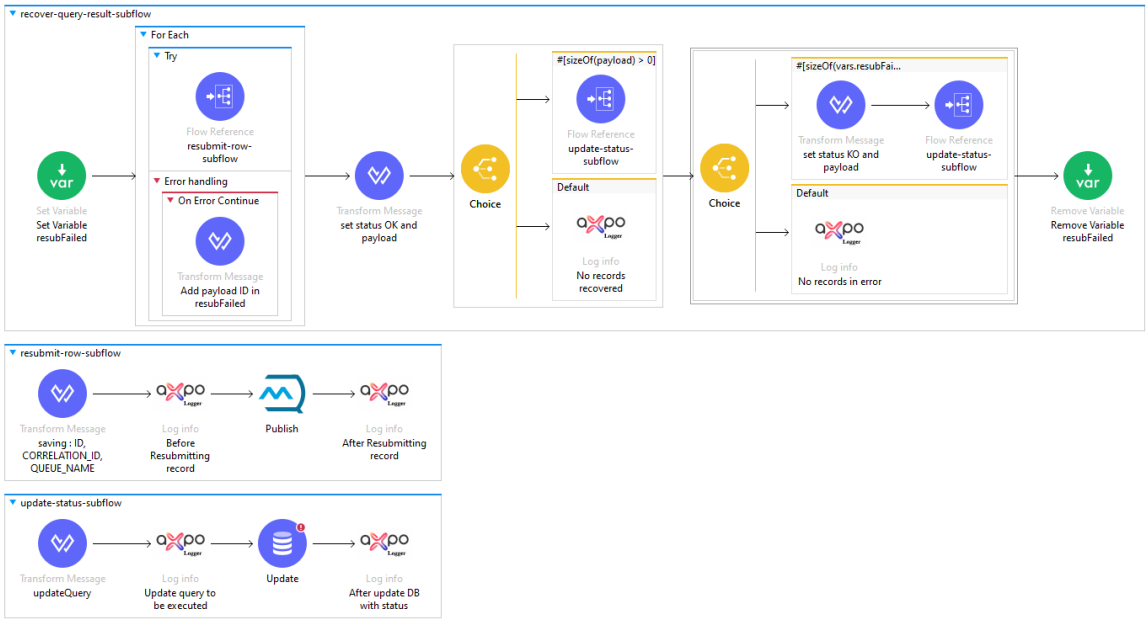
La logica di estrazione record del batch, qui sotto riportata, inizia con la definizione di un evento scheduler che, attraverso l'utilizzo di regole cron [7], è configurato, di default, per essere eseguito ogni 6 ore, `0 0 */6 ? * *`. Il flusso comincia con l'inizializzazione delle variabili ausiliarie, a cui segue la preparazione della query di estrazione da eseguire. Questa query, in particolare, la clausola SQL `WHERE` in essa, è configurabile attraverso una property, in modo da poter essere modificabile anche a runtime. In seguito, viene eseguita l'estrazione dei record attraverso il componente `Select` e i risultati di questa vengono passati al flusso comune di recupero e aggiornamento su database.



Flussi comuni di recupero e aggiornamento record

La logica comune di recupero e aggiornamento dei record estratti, qui sotto riportata, è articolata in tre flussi:

- `recovery-query-resul-subFlow` :
  - Il flusso inizia con l'iterazione, utilizzando il componente `For Each` , sull'iterabile di record estratti, ricevuto in input. Per ognuno di essi si chiama il flusso `resubmit-row-subFlow` che si occupa del recupero della pratica contenuta nel record. In caso di errori durante questo processo, i record per cui il recupero è fallito vengono inseriti nella lista `resubFailed`.
  - Successivamente si esegue l'aggiornamento dello stato delle pratiche su database. Per eseguire questa operazione si utilizza il flusso `update-status-subFlow`. Questo si occupa di valorizzare, nella tabella `MQ_RECOVERY`, il campo `RECOVERY_TIMESTAMP`, che indica il momento del recupero, e il campo `STATUS` che viene impostato a:
    - `RECOVERY_OK` per le pratiche correttamente recuperate, ovvero, la differenza tra l'iterabile di record iniziale e `resubFailed`
    - `RECOVERY_KO` per le pratiche il cui recupero è fallito.
- `resubmit-row-subFlow` : riceve in input un record estratto e utilizzando le informazioni della pratica, contenute in esso, pubblica nuovamente la pratica sulla coda del flusso opportuno.
- `update-status-subFlow`: riceve in input uno status e una lista di record con cui prepara la `updateQuery`, utilizzando logiche DataWeave, per poi eseguirla attraverso il componente `Update`.



5. Conclusioni

In conclusione, durante il mio percorso di tirocinio curriculare presso Axpo ho avuto la possibilità di utilizzare tecnologie moderne per affrontare le sfide dell'integrazione tra gli applicativi di un'organizzazione di scala medio grande, esplorando varie sfaccettature di esse.

Le dashboard sviluppate aiutano a comprendere velocemente lo stato dei sistemi analizzati e vengono utilizzate anche da membri di team differenti per trovare velocemente casistiche di errore, visualizzandole in modo efficace.

L'applicativo di recupero automatizzato è, attualmente, utilizzato solo attraverso l' API realizzata e fornisce una notevole velocizzazione per un processo che prima poteva essere effettuato unicamente in modo manuale. La versione automatizzata non è ancora attiva poiché è ancora in corso una fase di analisi che ha lo scopo di stabilire quali tipi di errori possano essere periodicamente recuperati in modo automatico, evitando di recuperare pratiche in errore , il cui errore si verificherebbe di nuovo, una volta recuperate.

5.1. Possibili sviluppi futuri

Per quanto riguarda le dashboard sviluppate, queste possono essere ampliate, aggiungendo, se ritenute utili, nuove metriche. Inoltre, per ogni dashboard risulta fondamentale eseguire un aggiornamento periodico dei filtri. Questo è utile poiché spesso analisi più dettagliate possono portare a classificare errori in un modo più chiaro o a cambiare classificazione di alcune casistiche analizzate nella dashboard. Inoltre, il processo di miglioramento dashboard dovrebbe essere accompagnato, per i flussi che ne necessitano, da un processo di miglioramento qualitativo dei log. Questo processo può essere condotto attuando diverse strategie, tra le quali:

- rendere i log di un flusso più uniformi
- aggiungere nei log maggiori informazioni specializzate per flusso
- aggiungere log per identificare i cambi di stato nel flusso, dove non presenti

Per all'applicativo di recupero automatizzato, come accennato in precedenza, deve essere ancora stabilita una query che permetta il recupero periodico automatizzato, evitando il recupero di pratiche i cui errori non sono dovuti a disservizi momentanei e quindi che terminerebbero di nuovo in errore. Inoltre, un possibile miglioramento potrebbe includere la realizzazione di una dashboard ELK per i recuperi. Questa fornirebbe una visualizzazione rapida dello stato delle pratiche recuperate, utilizzando dati provenienti da più flussi e quindi sfruttando la centralizzazione dei log di diversi applicativi sullo stack ELK.

6. Ringraziamenti

II risultati ottenuti in questi mesi di tirocinio non sarebbero stati tali se non fossi stato inserito in un clima lavorativo estremamente positivo. Pertanto, mi sento in dovere di ringraziare pubblicamente le persone che mi hanno aiutato in questi mesi.

In primo luogo voglio ringraziare Schenone Alessandro e Scaiano Rocco Alessandro per il costante supporto, il tempo dedicatomi e la libertà d'iniziativa che mi hanno lasciato in questi mesi in cui sono stato parte del loro team. Ringrazio il mio relatore, il professore Davide Ancona per la disponibilità fornitami. Ringrazio, poi, tutti i colleghi di Axpo con cui in questi mesi ho avuto la possibilità di condividere un ottimo ambiente lavorativo.

Il ringraziamento più importante va alla mia famiglia, in particolare, a mia madre e mio padre che mi hanno sostenuto, in questi mesi come sempre. Ringrazio mia sorella, le mie zie e mio zio per essere sempre al mio fianco e sopportarmi. Un ultimo ma non meno importante ringraziamento va a Sonia, una persona speciale, che trova sempre il suo modo di spronarmi e sostenermi.

7. Riferimenti

1. Documentazione Elasticsearch
2. Python Elasticsearch
3. Documentazione Mulesoft
4. Publish-subscribe pattern
5. Database Oracle
6. RAML
7. DataWeave
8. cron